Politechnika Wrocławska
Wydział Informatyki i Telekomunikacji

Szeregowanie zadań wieloprocesorowych w warunkach niepewności

mgr inż. Dariusz Dorota

Rozprawa doktorska
napisana pod kierunkiem naukowym
prof. dr hab. inż. Czesława Smutnickiego

Wrocław, 2023

Składam serdeczne podziękowania:
Panu Profesorowi dr hab. inż. Czesławowi Smutnickiemu
za okazaną życzliwość, poświęcony czas
oraz wskazówki udzielone mi podczas pisania pracy.
Mojej żonie
za cierpliwość i wsparcie podczas pisania pracy.

Spis treści

Streszczenie

Rozprawa dotyczy problemów optymalizacji realizacji zadań wieloprocesorowych w warunkach niepewności danych. Motywacją rozprawy są zastosowania rozpatrywanych problemów w zarządzaniu systemami wbudowanymi. Rozważane są zadania obliczeniowe wymagające równoczesnego, synchronicznego dostępu do wielu niezależnych procesorów w celu realizacji systemu wbudowanego o podwyższonej niezawodności poprzez redundancję sprzętową i programową. Systemy tego typu mają zastosowanie, między innymi, w lotnictwie, pojazdach kosmicznych, samochodach, urządzeniach wojskowych, środkach transportu materiałów niebezpiecznych, instalacjach jądrowych, instalacjach chemicznych, górnictwie, dronach. Niepewność dotyczy parametrów takich jak np. czasy wykonywania zadań, terminy zgłoszenia zadań, liczba zadań. Przedstawiono przegląd podejść do modelowania i rozwiązywania problemów szeregowania wieloprocesorowego, ze szczególnym uwzględnieniem problemów z danymi niepewnymi. Zdefiniowano zakres i sposoby reprezentacji niepewności danych. Przedyskutowano alternatywne sposoby modelowania niepewności: rozmyte, stochastyczne, on-line, i inne. Wychodząc od deterministycznych algorytmów off-line dla zadań jednoprocesorowych, sformułowano odpowiednie warianty algorytmów dla zadań wieloprocesorowych z danymi niepewnymi w wersjach off-line i on-line. Zaproponowano, a także udowodniono twierdzenia dotyczące pewnych własności algorytmów w wersji off-line jak i on-line. Przeprowadzono badania testowe wszystkich algorytmów, zaś rezultaty przedstawiono z wykorzystaniem właściwych narzędzi, m.in. diagram Gantta, wykresy wyników, tabele. Dostarczono aktualne kompendium wiedzy w zakresie architektur systemów wbudowanych.

Słowa kluczowe: szeregowanie, algorytm, systemy wbudowane, niepewność

Abstract

The following thesis concerns the problems of optimization of multiprocessor tasks in conditions of data uncertainty. The dissertation is motivated by the application of the discussed problems in the management of embedded systems. Computational tasks, that require simultaneous access to many independent processors in order to realize an embedded system with increased reliability through redundancy, are considered. Applications relate to, among others, aviation and space systems, cars, military devices, means of transport of hazardous materials, nuclear installation, chemical installations, mining, drones. It also has a relation to humanoid robot movement, ribbon cutting. Uncertainty concerns parameters such as task execution times, task submission deadlines, number of tasks. An overview of approaches to modeling and scheduling problem-solving is provided, with particular emphasis on problems with uncertain data. The scope and ways of representing data uncertainty were defined. Alternative methods of uncertainty modeling were discussed: fuzzy, stochastic, on-line, and others. Starting from the deterministic offline algorithms for single-processor tasks, there have been formulated appropriate algorithms for multiprocessor task with uncertain data in off-line and on-line cases. Theorems concerning certain properties of off-line and online algorithms have been proposed and proved. All algorithms were tested and results were presented with the use of appropriate tools, including Gantt chart diagram, other charts and tables. An up-to-date compendium of knowledge on embedded system architectures has been provided.

Keywords: scheduling, algorithm, embedded systems, uncertainty

Rozdział 1

Wstęp

Zagadnienia szeregowania występują w rzeczywistych systemach dedykowanych dla różnych zastosowań. W systemach wbudowanych, IoT (ang. *Internet of Things*), IoV (ang. *Internet of Vehicles*) o zwiększonym poziomie bezpieczeństwa występuje szczególna klasa zagadnień optymalizacji szeregowania zadań wieloprocesorowych, wymagających do realizacji równoczesnego, synchronicznego dostępu do kilku pracujących równolegle procesorów. Biorąc pod uwagę tendencje w zakresie uniformizacji i modularyzacji sprzętu, zwiększoną niezawodność osiąga się poprzez wprowadzenie redundancji sprzętowej, pozwalającej także na redundancję programową. Redundancja ma na celu automatyczne podejmowanie właściwych decyzji drogą "głosowania" powielonych urządzeń wykonujących te same funkcje lub programy dla identycznych danych. Systemy takiego typu są już powszechnie stosowane między innymi w lotnictwie, pojazdach kosmicznych, samochodach, urządzeniach wojskowych, środkach transportu materiałów niebezpiecznych, instalacjach jądrowych, instalacjach chemicznych, górnictwie, dronach. Łatwość realizacji powoduje rozszerzanie takich implementacji na kolejne obszary działalności człowieka.

1.1. Ogólny zarys problemu

Metody i algorytmy efektywnego harmongramowania działań rozwijane od lat 50-tych XX wieku osiągnęły znaczący poziom rozwoju po wprowadzeniu kom-

puterów do modelowania, analizy i ich rozwiązywania. W rezultacie wieloletniej ewolucji dziedziny sformułowano oraz rozwinięto znacząco teorię i praktykę szeregowania zadań, obejmując swoim zakresem różnorodne zagadnienia o praktycznym znaczeniu. Obecnie, rozpatrywane są m.in. problemy deterministyczne, rozmyte, stochastyczne, jedno- oraz wielo-kryterialne, z dodatkowymi zasobami czy też ograniczeniami, z różną strukturą przepływu zadań, z różnymi modelami czas/koszt, a także algorytmy ich rozwiązywania o różnych cechach numerycznych. Wyniki znajdują zastosowanie w wielu dziedzinach, począwszy od transportu, budownictwa, poprzez przemysłowe procesy wytwórcze, a skończywszy na systemach operacyjnych komputerów.

Identyfikacja każdego rozpatrywanego problemu w kontekście taksonomii problemów szeregowania oraz analiza jego cech szczególnych pozwala na właściwe dobranie istniejącego, lub opracowanie nowego algorytmu rozwiązania. Zatem nadrzędnym celem dziedziny szeregowania zadań jest, poprzez klasyfikację problemów oraz sformułowanie ich szczególnych własności, zaprojektowanie odpowiednich algorytmów optymalizacji użytecznych w praktyce. Należy przy tym uwzględniać złożoność obliczeniową, dokładność aproksymacji, możliwość równoległości obliczeń, przydatność praktyczną algorytmu. NP-trudność większości praktycznych problemów czyni nietrywialnym zagadnienie projektowania i analizy odpowiednich algorytmów optymalizacyjnych.

Niepewność w problemach szeregowania może dotyczyć różnych danych i założeń problemu, [13]. We wszystkich przypadkach z niepełną informacją (B) – (I) wymienionych poniżej stosowane są oceny jakości odnoszące się do rozwiązań optymalnych możliwych potencjalnie do uzyskania przy założeniu pełnej wiedzy a posteriori o problemie, patrz przypadek (A), których dostarcza zwykle:

(A) deterministyczna teoria szeregowania, traktowana jako baza dla dalszych oszacowań.

Zauważmy, że niepewność może być statyczna lub dynamiczna i może być różnie postrzegana, przykładowo:

(B) zbiór zadań jest znany a priori, wszystkie zadania z tego zbioru muszą być wykonane, lecz niektóre parametry zadań są niepewne, np. czasy wykonywania zadań, żądane terminy zakończenia, etc.,

(C) zbiór zadań jest znany a prori, czasy wykonywania zadań są znane, jednak nie wszystkie zadania będą realizowane, zaś terminy napływu zadań oraz ich repertuar w zadanym odcinku czasu są niepewne,
(D) relacja poprzedzania zadań jest niepewna,
(E) zbiór zadań jest nieznany a priori, zadania napływają strumieniem w sposób nieskończony, czasy przetwarzania zadań są nieznane a priori, znane bądź niepewne w chwili zgłoszenia się zadania,
(F) żądania zasobowe zadań są niepewne,
(G) dostępność zasobów jest niepewna,
(H) rezultat poprawności wykonania zdania jest niepewny (kontrola jakości),
(I) nieznana jest liczba i zbiór zadań do wykonywania.

Dodatkowo, przypadki (A) – (I) mogą uwzględniać różne kryteria optymalizacji. Odpowiednio do (A) – (I) stosowane są odmienne techniki modelowania, analizy i interpretacji wyników. W punkcie (A) będziemy odnosić się znanych deterministycznych problemów szeregowania zadań, biorąc pod uwagę ich taksonomię, aktualne tendencje w zakresie modelowania i metod rozwiązywania. W tym też zakresie zaproponujemy w rozprawie rozwinięcie algorytmu Muntza-Coffmana [120] na przypadek zadań wieloprocesorowych, patrz rozdz. 3.3 implementowaną dalej dla danych niepewnych. Nieokreślone parametry liczbowe (B)...(I) można modelować i rozwiązywać w systemach rozmytych lub za pomocą zmiennych losowych. Choć podejścia takie występują generalnie w szeregowaniu, to jednak dotychczas nie formułowano ich i nie rozważano ich w kontekście zadań wieloprocesorowych. Tematyka tych badań pozostaje wciąż otwarta. Warianty (G) i (H) nie występowały dotychczas w literaturze. Z kolei wariant (I) wymaga sprecyzowania założeń dotyczących napływających zadań i ich obsługi. Nieskończony napływ zgłoszeń jest założeniem trudnym do przyjęcia w zamkniętych systemach IoT, IoV, choć ma sensowny charakter. Możemy bowiem założyć, że powtarzalne zadania obliczeniowe są unikalne ze względu na dane, napływają losowo w strumieniu. Alternatywnym założeniem jest realizacja skończonego, lecz nieznanego z góry podzbioru zadań, wybranego ze znanego większego zbioru zadań. Założenie pierwsze pozwala nam modelować i rozwiązywać problem z pomocą kolejek otwartych lub zamkniętych. Założenie drugie – za pomocą obsługi online. W ni-

niejszej pracy, podejście z pracy własnej [51] do szeregowania zadań wieloprocesorowych, zostało zmodyfikowane oraz dostosowane do wieloprocesorowego szeregowania online.

1.2. Tezy pracy

Obserwowane w ostatnich latach tendencje w zakresie projektowania i realizacji systemów sterowania obiektami i/lub procesami są dość wyraźne. Szybkie zmiany technologiczne wypierają sprzętowe rozwiązania systemów sterowania, zastępując je specjalizowanym oprogramowaniem sterowników urządzeń. Niskie koszty oraz masowość wytwarzania pozwala na zwiększenie niezawodności systemów poprzez redundancję sprzętową i programową. Rośnie lawinowo potencjalna liczba obszarów zastosowań, cywilnych, militarnych, kosmicznych, itp. Wspólnym elementem dla wymienionych zastosowań są identyczne zadania obliczeniowe realizowane równocześnie i synchronicznie na kilku procesorach. Zgodnie z tematem rozprawy, formułuję następujące tezy:

- zadania wieloprocesorowe modelują procesy obliczeniowe wymagające zwiększonej niezawodności programów i urządzeń, zwłaszcza w systemach wbudowanych, poprzez redundancję sprzętową i programową,
- problemy szeregowania zadań wieloprocesorowych z niepewnymi danymi lepiej modelują rzeczywiste systemy sterowania niż problemy deterministyczne,
- możliwe jest skonstruowanie algorytmów szeregowania zadań wieloprocesorowych z danymi niepewnymi poprzez rozszerzenie dotychczasowych wyników znanych dla problemów deterministycznych,
- różne podejścia do modelowania dostarczają różne algorytmy, różne wyniki teoretyczne i symulacyjne.

Prawdziwość postawionych cech zostanie potwierdzone poprzez:

- zaproponowanie modelu matematycznego algorytmu realizującego uszeregowanie zadań wieloprocesorowych, zarówno dla logiki rozmytej jak i problemów deterministycznych oraz stochastycznych,
- udowodnienie poprawności zaproponowanych algorytmów,

- ocena złożoności obliczeniowej algorytmów,
- zaimplementowanie zaproponowanych algorytmów,
- przeprowadzenie eksperymentów dla rozważanych wariantów rozwiązania,
- zestawienie i porównanie eksperymentów w celu oceny podejść.

1.3. Zawartość pracy

Rozprawa składa się 6 rozdziałów. W rozdziale pierwszym sformułowane zostały tezy, cel i zakres pracy. Rozdział drugi dostarcza wprowadzenia w tematykę szeregowania zadań, w tym podstawowych pojęć i definicji, opisu problemów od deterministycznych, poprzez rozmyte oraz stochastyczne, adaptacyjne szeregowanie online, systemy masowej obsługi, czy też systemy odporne, a na dynamicznej zmianie zbioru zadań skończywszy. Szczególną uwagę zwrócono na różne technologie modelowania niepewności oraz najnowsze wyniki literaturowe.

Rozdziały 3-5 tworzą zasadniczą część pracy i zawierają wyniki własne Autora, zarówno teoretyczne, jak i procedury oraz algorytmy, a także wyniki badań eksperymentalnych dla weryfikacji koncepcji. I tak rozdział 3 przedstawia motywację dla zastosowania zadań wieloprocesorowych oraz aktualny stan wiedzy w tym zakresie. W tym rozdziale opisano także pewien bazowy algorytm, który został następnie zaadaptowany i odpowiednio modyfikowany dla zastosowań omawianych w rozprawie. Dalej, rozdział 4 zawiera wyniki teoretyczne oraz algorytmy szeregowania zadań wieloprocesorowych z danymi niepewnymi, zrealizowane w różnych podejściach. Punktem wyjścia jest model deterministyczny. Po nim rozważono model rozmyty, stochastyczny, on-line, system kolejkowy oraz dynamiczny zbiór zadań. Dla zaproponowanych algorytmów i podejść przeprowadzone zostały eksperymenty obliczeniowe i badania numeryczne, których wyniki opisano szczegółowo w rozdziale 5. Otrzymane rezultaty pozwoliły na sformułowanie wniosków zawartych w rozdziale 6.

Integralną część pracy stanowi dodatek A przedstawiający klasyfikację i przegląd architektur systemów wbudowanych. Zaprezentowano w nim skrótowo rozwój struktur oraz koncepcji systemów wbudowanych ukazując tym samym zalety, zastosowanej w niniejszej rozprawie, architektury i modelu systemu szeregowania.

Rozdział 2

Problemy szeregowania

Dynamicznie rozwijająca się dziedzina systemów wbudowanych stymuluje powstawanie nowych technologicznych rozwiązań sprzętowych i programowych w celu stworzenia maksymalnie niezawodnych, oraz wydajnych systemów tego typu. Standaryzacja rozwiązań sprzętowych plus rozwój techniki mikroprocesorowej zwiększyły znaczenie rozwiązań programowych i algorytmów sterowania. Stąd jednym ze znaczących składników systemów wbudowanych są efektywne metody zarządzania zadaniami obliczeniowymi, przy dodatkowych ograniczeniach, sprowadzane zwykle do bardzo specyficznych zagadnień harmonogramowania, nazywanych czasami problemami szeregowania. Generalnie, harmonogramowanie może być postrzegane jako pewien proces decyzyjny, polegający na alokowaniu zasobów do realizacji zadań. Występuje on w wielu gałęziach przemysłu i usług. Każde z zadań charakteryzowane może być poprzez cechy specyficzne i ograniczenia dla konkretnej aplikacji, przykładowo: dedykowane dla systemów operacyjnych komputerów, odmienne w procesach produkcyjnych przedsiębiorstwa wytwarzającego samochody, całkowicie inne dla systemów wbudowanych Internetu Rzeczy (ang. *Internet of Things, IoT*), oraz Internetu Pojazdów (ang. *Internet of Vehicles, IoV*). Dodatkowo niezależnie od branży, w metodach harmonogramowania przyjmuje się określone cele optymalizacji, którymi mogą być, na przykład, minimalizacja czasu realizacji zadań w systemie lub minimalizacja liczby zadań zakończonych po upływie dopuszczalnego czasu ich realizacji. Teoria szeregowania jest zorientowana na problemy, pochodzące z wybranych gałęzi

przemysłu i gospodarki. Dobrze zbadane problemy deterministyczne (z pełną informacją a priori) stanowią solidną podstawę do analizy i porównań dla innych podejść. W tej rozprawie, wychodząc od problemów deterministycznych, skupimy się na wieloprocesorowych problemach niedeterministycznych [19, 132], głównie rozmytych i stochastycznych, w których wykonywanie zadań obliczeniowych wymaga równoczesnego użycia wielu elementów przetwarzających (tzw. redundancja sprzętowa i programowa). Biorąc pod uwagę, że znaczna liczba, nawet najprostszych, deterministycznych problemów szeregowania należy do klasy problemów silnie NP-trudnych, otrzymanie optymalnego rozwiązania rozważanych problemów jest kosztowne obliczeniowo [76, 154]. Ponieważ rozwój systemów wbudowanych dokonuje się zarówno w zakresie oprogramowania jak i sprzętu, implementowane są głównie algorytmy pozwalające na wykonanie zadania obliczeniowego w praktycznie akceptowalnym czasie. Liczne aplikacje IoT oraz IoV, oraz rozwój technologii wymagają zwiększenia wydajności systemów wbudowanych, mierzonych szybkością działania oraz poziomem niezawodności [76].

2.1. Wprowadzenie

Rozpoczniemy od formalnego zdefiniowania pojęć, oznaczeń i problemów występujących w rozprawie. *Zadanie* polega na wykonaniu określonego zestawu (zwykle ciągu) czynności, złożonego z *operacji*. Dalej przyjmujemy, że elementarną czynność stanowi bądź operacja, bądź zadanie (w przypadku, gdy zadania są jedno-operacyjne). Zarówno zadania jak i ewentualne operacje wymagają użycia do ich realizacji *zasobów*. Zasobami mogą być np. personel, kapitał, surowce, energia, urządzenia. Zadania i operacje mogą być charakteryzowane przez parametry takie jak [1]:

- *czas wykonania zadania* - liczba jednostek czasu wymaganych do zrealizowania zadania,
- *czas wykonania operacji* - liczba jednostek czasu wymaganych do zrealizowania operacji,
- *najwcześniejszy możliwy termin rozpoczęcia zadania* - najmniejsza możliwa

[1] każda z wymienionych wielkości może być deterministyczna, rozmyta, lub zmienną losową

liczba jednostek czasu, po której można rozpocząć wykonywanie zadania,

- *najwcześniejszy możliwy termin rozpoczęcia operacji* - najmniejsza możliwa liczba jednostek czasu, po której można rozpocząć wykonywanie operacji,
- *najpóźniejszy możliwy termin zakończenia zadania* - największa możliwa liczba jednostek czasu, po której należy zakończyć wykonywanie zadania,
- *najpóźniejszy możliwy termin zakończenia operacji* - największa możliwa liczba jednostek czasu, po której należy zakończyć wykonywanie operacji,
- *podzielność operacji* - parametr określający czy istnieje możliwość podziału operacji na mniejsze części,
- *czas ograniczenia* - zależność czasowa pomiędzy terminami rozpoczęcia i zakończenia dla ścieżki zawierającej niektóre operacje w grafie poprzedzania .

Oznaczmy przez $T = \{T_1, T_2, T_3, ..., T_n\}$ zbiór zadań do wykonania na maszynach (procesorach) zdefiniowanych jako $M = \{M_1, M_2, M_3, ..., M_m\}$ w przypadku deterministycznym. Analogiczne oznaczenia będą stosowane przeze mnie w przypadku występowania niepewności w procesie szeregowania, przy czym rozróżnienie polega na użyciu specyficznego indeksu górnego F lub S. Przykładowo, dla opisania zadań z użyciem logiki rozmytej mamy T^F, natomiast w przypadku stochastycznym stosuje się symbol T^S, patrz tabela 2.1. Podstawowo przyjmuje się, że zadanie T_i zawiera jedną niepodzielną operację realizowaną równocześnie i synchronicznie na predefiniowanej liczbie procesorów $a_i \geq 1$. i wymaga czasu p_i. W przypadku dopuszczenia przerywania zadań zakłada się, że zadanie $T_i \in T$ składa się z sekwencji pewnej liczby x_i operacji $O_i = (o_{i,1}, o_{i,2}, o_{i,3}, ..., o_{i,x_i})$, wykonywanych w tej kolejności, z których każda wymaga synchronicznego użycia a_i procesorów. Oczywiście, musi zachodzić $\sum_{k=1}^{x_i} |o_{i,k}| = p_i$, gdzie $|o_{i,k}|$ oznacza czas trwania operacji $o_{i,k}$. W tym przypadku podział zdania na operacje nie jest ustalony i podlega wyborowi. Zadania/operacje wieloprocesorowe są czasami w literaturze charakteryzowane łącznie przez tzw. współczynnik wieloprocesorowości MPR (ang. *Multi-Processor Ratio coefficient*). Przyjmuje on wartości: (1) $MPR = 1$, co oznacza, że wszystkie zadania są jednoprocesorowe, tzn. $a_i = 1$, $T_i \in T$, (2) $MPR = k$, co oznacza, że wszystkie zadania wymagają takiej samej liczby procesorów k do realizacji, tzn. $a_i = k \leq m$, $T_i \in T$, (3)

$MPR \leq k$, co oznacza, że różne zadania wymagają różnej liczby procesorów, zatem $a_i \leq k \leq m$, $T_i \in T$. W praktyce systemów wbudowanych spotykamy najczęściej $a_i = 2..3$ – dla zadań wieloprocesorowych w zależności od wymaganego poziomu niezawodności lub prawdziwego zrównoleglenia (ang. *True Parallelism*).

2.2. Modelowanie problemów szeregowania

W drugiej połowie XX wieku zaobserwowano dynamiczny rozwój teorii szeregowania zadań. Był on stymulowany głównie konkurencją ekonomiczną na rynku wytwarzania i usług. Badania obejmowały szeroko pojętą optymalizację systemów produkcyjnych oraz komputerowych, znajdując liczne zastosowania praktyczne w dziedzinach takich jak na przykład zarządzanie zasobami ludzkimi, produkcja, transport, budownictwo, systemy operacyjne komputerów. Aktualnie, problemy szeregowania są analizowane w ścisłym związku z teorią złożoności obliczeniowej oraz oceną dokładności wyznaczania rozwiązania. Podział uwzględniający pierwsze kryterium klasyfikacyjne wyróżnia [19, 58, 60, 92, 132, 133, 154]:

- problemy wielomianowe, gdzie czas wyznaczenia rozwiązania optymalnego jest wielomianem zależnym od rozmiaru problemu [2].
- problemy pseudowielomianowe, w których czas wyznaczenia rozwiązania optymalnego jest wielomianem zależnym od rozmiaru problemu oraz pewnych danych liczbowych problemu.
- problemy silnie NP-trudne, dla których wyznaczenie rozwiązania optymalnego wymaga czasu wykładniczo zależnego od rozmiaru problemu [3].
- problemy otwarte (nierozstrzygnięte).

Teoria złożoności obliczeniowej ma istotny wpływ na wybór algorytmu lub algorytmów rozwiązujących poszczególne problemy, lub klasy problemów. Repertuar podejść jest dość różnorodny: od programowania matematycznego, poprzez algorytmy heurystyczne, meta-heurystyczne, aż do algorytmów losowych i zrandomizowanych. Klasa problemów NP-trudnych była i jest przedmiotem intensywnych badań, bowiem zawiera zdecydowanie więcej przypadków niż klasa proble-

[2] Nazywane są także problemami “łatwymi”

[3] Nazywane są także problemami “trudnymi”

Podejście: D-deterministyczne, F-rozmyte, S-stochastyczne, O-online, M - systemy masowej obsługi, Z - dynamiczna zmiana zbioru zadań						
D	F	S	O	M	Z	Znaczenie
T	T^F	T^S	T^O	T^M	T^Z	zbiór zadań $T = \{T_1, \dots T_n\}$
T_i	T_i^F	T_i^S	T_i^O	T_i^M	T_i^Z	zadanie i-te ze zbioru zadań T
$t_{i,CT}$	$t_{i,CT}^F$	$t_{i,CT}^S$	$t_{i,CT}^O$	$t_{i,CT}^M$	$t_{i,CT}^Z$	czas zadania krytycznego $T_{i,CT}$
a_i	a_i^F	a_i^S	a_i^O	a_i^M	a_i^Z	liczba procesorów żądanych przez T_i
TG	TG^F	TG^S	TG^O	TG^M	TG^Z	graf zadań, $TG = (T, E)$, $E \subset T \times T$
H_i	H_i^F	H_i^S	H_i^O	H_i^M	H_i^Z	poziom zadania T_i w grafie, $H_i = \Sigma_{k=1}^{i} E_k$, dla $k \in A_i$
A_i	A_i^F	A_i^S	A_i^O	A_i^M	A_i^Z	ścieżka zadań w grafie TG kończąca się w T_i
$C_{\max}$	$C_{\max}^F$	$C_{\max}^S$	$C_{\max}^O$	$C_{\max}^M$	$C_{\max}^Z$	długość uszeregowania
p_i	p_i^F	p_i^S	p_i^O	p_i^M	p_i^Z	czas wykonania zadania T_i; $p_i = \sum_{j=1}^{x_i} \lvert o_{ij}\rvert$; $x_i = \lvert O_{ij}\rvert$,
o_{ij}	o_{ij}^F	o_{ij}^S	o_{ij}^O	o_{ij}^M	o_{ij}^Z	operacja j zadania T_i
$\lvert o_{ij}\rvert$	$\lvert o_{ij}^F\rvert$	$\lvert o_{ij}^S\rvert$	$\lvert o_{ij}^O\rvert$	$\lvert o_{ij}^M\rvert$	$\lvert o_{ij}^Z\rvert$	czas wykonania operacji j zadania T_i,
S_i	S_i^F	S_i^S	S_i^O	S_i^M	S_i^Z	najwcześniejszy możliwy termin rozpoczęcia zadania T_i
d_i	d_i^F	d_i^S	d_i^O	d_i^M	d_i^Z	najpóźniejszy możliwy termin rozpoczęcia zadania T_i
D_i	D_i^F	D_i^S	D_i^O	D_i^M	D_i^Z	najpóźniejszy możliwy termin zakończenia T_i; $D_i = d_i + p_i$
w_i	w_i^F	w_i^S	w_i^O	w_i^M	w_i^Z	priorytet zadania T_i, $w_i = a_i \cdot p_i$
$w_{i,CT}$	$w_{i,CT}^F$	$w_{i,CT}^S$	$w_{i,CT}^O$	$w_{i,CT}^M$	$w_{i,CT}^Z$	najwyższy priorytet zadania T_i w jednostce czasu

Tabela 2.1: Zestawienie oznaczeń parametrów szeregowania zadań dla przypadku deterministycznego, rozmytego, stochastycznego, on-line, systemów masowej obsługi, dynamicznego zbioru zadań

mów wielomianowych. Biorąc pod uwagę charakter rzeczywistych danych instancji, można posłużyć się także inną klasyfikacją, rozróżniającą problemy [154]: (A) deterministyczne, (B) stochastyczne, (C) rozmyte. Dalsze kryteria podziału uwzględniają strukturę zadań i systemu wytwórczego. Wyróżniamy tu, między innymi, problemy: jedno lub wielomaszynowe, otwarte, przepływowe, gniazdowe, hybrydowe. Rozpatrywanie charakteru danych, klas problemu i jego złożoności umożliwia podjęcie właściwej decyzji odnośnie użycia narzędzi do modelowania i analizy problemu, odpowiednio [154]: (A) deterministycznej teorii szeregowania, (B) procesów stochastycznych, teorii kolejek, (C) teorii zbiorów rozmytych.

Przegląd literatury, patrz np. [13, 132] wskazuje, że istnieje liczna grupa problemów charakteryzująca się danymi niepewnymi i nieprecyzyjnymi. Problemy niedeterministyczne często odwołują się referencyjnie do wyników analogicznych problemów deterministycznych. Podejście takie pozwala zastosować deterministyczną teorię szeregowania do zaprezentowanych wcześniej problemów, skorelowanych ze sposobami reprezentacji danych, niezależnie od klasy tych problemów [132, 154]. Ponieważ problemy NP-trudne nie pozwalają na wyznaczenie optimum w rozsądnym czasie, dość często stosuje się *relaksację* (redukcję) do problemu posiadającego zwykle mniejszą złożoność obliczeniową. Powielając to postępowanie, można skonstruować pewien łańcuch kolejnych redukcji. Pośrednią konsekwencją redukcji jest pewien hierarchiczny system klasyfikacyjny. Taksonomia z punktu widzenia złożoności obliczeniowej problemów umożliwia ocenę jak zmiana pojedynczego parametru w klasyfikacji, wpływa na złożoność procesu harmonogramowania, co ostatecznie pozwala (lub nie) na wprowadzanie pewnych uogólnień problemu. Każdy z modeli uszeregowania określa takie elementy jak: liczbę zadań w systemie (skończoną lub nieskończoną), zależności pomiędzy zadaniami, czas wykonania zadań, czasy ograniczeń oraz inne dodatkowe parametry i ograniczenia, [19, 132]. Mnogość rozpatrywanych elementów służących do przedstawienia problemu szeregowania wymusiła opracowanie jednolitej i uniwersalnej klasyfikacji, pozwalającej na opis na tyle szczegółowy, aby możliwe było efektywne dobranie narzędzi służących wyznaczaniu optimów.

Formułując problem szeregowania zadań należy wziąć pod uwagę klasyczną teorię szeregowania, stosując się do już wcześniej zaproponowanej nomenklatury.

Niezależnie od tego jakiej dziedziny dotyczyć będzie proces szeregowania, zazwyczaj stosuje się ujednolicone oznaczenia. W procesie harmonogramowania określa się pewien zbiór elementarnych pojęć jak zadania czy zasoby. Oczywiście definicja zarówno zadań jak i zasobów jest zależna od gałęzi przemysłu w obszarze której rozpatrywane jest szeregowanie [19, 154]. Przykładem może być proces konstrukcji elementu na linii produkcyjnej, realizacja inwestycji w budownictwie, opracowanie nowego sposobu realizacji układów ASIC, gdzie każda ze wspomnianych aktywności należy do innej gałęzi przemysłu i także określenie zasobów jest tutaj zależne od obszaru zastosowania [81, 154]. Podobnie jak w przypadku zadań, zasoby także mogą być określane, a co za tym idzie klasyfikowane w obrębie pewnych cech. Można tutaj wymienić trzy podstawowe kategorie [81, 122, 154]:

- odnawialne, z ograniczonym strumienieniem zasobów w każdej chwili,
- nieodnawialne, z ograniczoną globalną ilością konsumowanych zasobów,
- podwójnie ograniczone, posiadających obie powyższe cechy.

Zasoby charakteryzowane mogą być również przez inne cechy takie jak: ilość, koszt, podzielność, dostępność czy też wywłaszczalność. Pewna swoboda w interpretacji i postrzeganiu może zasobów może wpływać na sformułowanie algorytmu rozwiązywania, a w konsekwencji na określenie złożoności obliczeniowej [154]. Zauważmy, że problem postawiony w rozprawie może być także postrzegany jako szeregowanie zadań przy ograniczeniach zasobowych, w którym zadanie T_i wymaga a_i jednostek zasobu odnawialnego ($a_i \leq m$ procesorów) podzielnego z dokładnością co jednostkę, zatem jest to specyficzny problem klasy RCPS (ang. *Resource Constrained Project Scheduling*).

2.3. Deterministyczne problemy szeregowania

W celu klasyfikacji problemów rozważanych w rozprawie powołamy się na notację trójpolową Grahama i in. [75] używaną w szeregowaniu deterministycznym. Jej zaletą jest to, że nie wszystkie pojęcia teorii są wymagane do jej przedstawienia; wystarczą tylko te niezbędne do opisu rozpatrywanego przypadku. Problemy są charakteryzowane symbolem $\alpha|\beta|\gamma$, gdzie α oznacza typ problemu (dokładniej marszruty zadań w systemie), β specyfikuje dodatkowe ograniczenia, γ

określa postać funkcji celu, [19, 58, 132, 154]. Niestety, przyjęta taksonomia pozwala łatwo określić problemy proste, jednak jest kłopotliwa dla problemów wymagających obsługi równoległej zadań, w tym w szczególności obsługi równoległej synchronicznej. W książce [132] zaproponowano pewne rozszerzenie symbolu α dla potrzeb przetwarzania równoległego zadań. Przyjmuje się tam, że $\alpha = \alpha_1\alpha_2\alpha_3$. Wielkość α_1 może być liczbą (wówczas definiuje skończoną liczbę maszyn w systemie) lub symbolem pustym $\circ$, co sygnalizuje, że liczba maszyn jest dowolna m. Symbol α_2 określa typ zagadnienia $\alpha_2 \in \{\circ, O, F, F^*, J\}$. Odpowiada to kolejno oznaczeniom: $\circ$ - jedno stanowisko obsługi z maszynami równoległymi wynikającymi z α_3; O - system otwarty, w którym marszruty zadań podlegają wyborowi; F - system przepływowy (o strukturze szeregowej), w którym marszruty wszystkich zadań są ustalone i identyczne; F^* - uproszczony system przepływowy o zredukowanej liczbie rozwiązań; J - system z dowolnymi różnymi, ale ustalonymi marszrutami zadań. Parametr $\alpha_3 \in \{\circ, P, Q, R\}$ specyfikuje tryb wykorzystania równoległego przetwarzania w stanowiskach. Symbol $\circ$ określa brak możliwości przetwarzania równoległego; kolejne symbole specyfikują stanowiska z przetwarzaniem równoległym, dopuszczając odpowiednio procesory/maszyny: P - identyczne, R - jednorodne, Q - niejednorodne. Współczynnik β pozwala na wprowadzenie dodatkowych ograniczeń w problemie szeregowania. Typowo wymienia się pewne oznaczenia umowne, np. $\beta \subseteq \{r_i, q_i, prec, tree, w_i = 1, w_i = p_*, d_i \leq C_i, no-wait, no-store\}$. Parametr γ specyfikuje kryterium optymalności. Jako przykładowe wymienić można: długość uszeregowania, maksymalne opóźnienie, średnie opóźnienie, średnie ważone opóźnienie, liczba opóźnionych zadań, ważona liczba opóźnionych zadań, suma czasów ukończenia, itd., lub ich kombinacje, patrz [58, 60]. Czytelnika odsyłamy do prac zawierających bardziej szczegółowe omówienie zasad klasyfikacji [19, 58, 132, 154]. Dla przykładu: $Q||C_{\max}$ to jedno stanowisko zawierające równolegle pracujące maszyny jednorodne z minimalizacją długości uszeregowania. Oznaczenie $FP||\overline{C}$ określa problem hybrydowy, szeregowo-równoległy z kryterium średniego czasu przepływu.

Niestety, wymieniona wcześniej modyfikacja α nie pozwala na opisanie problemu rozpatrywanego w tej rozprawie. Dlatego też zamiast α dokonano rozszerzenia zakresu znaczeń symbolu β, a mianowicie: *fuzzy (p)*, *random(p)*, *online-list*,

oraz zaproponowano dodatkowy parametr *mpr* dla β. Ten ostatni jest współczynnikiem wieloprocesorowości (tożsamy z a_i), który specyfikuje liczbę procesorów konieczną do wykonania pojedynczego zadania i może przyjmować wartości z zakresu $1 \ldots k$. Będziemy używali go w zapisie $mpr = k$ (wszystkie zadania są k-procesorowe, lub $mpr \leq k$ co oznacza, że zadania są "wymieszane" $1, 2, \ldots, k$-procesorowe. Propozycja ta jest zmianą w stosunku do koncepcji przedstawionej w pracach [14, 59, 174], bowiem wprowadza model wieloprocesorowy poprzez ograniczenia problemu, a nie poprzez strukturę systemu przetwarzania. Krzystając z podanej klasyfikacji, problemy rozpatrywane w rozprawie są klasy $P|\beta|C_{\max}$, gdzie $\beta \in \{\circ, mpr = k, mpr \leq k, pmtn, prec, fuzzy(p), random(p)\}$. Brak występowania parametru mpr jest tożsamy z $mpr = 1$.

Metody rozwiązywania problemów deterministycznych są problemowo-zorientowane, odmienne dla problemów wieloekstremalnych, jedno- lub wielokryterialnych, NP-trudnych. Często dla tego samego problemu NP-trudnego istnieje kilka- lub kilkanaście algorytmów rozwiązywania różniących się istotnie cechami numerycznymi, takimi jak dokładność wyznaczania optimum, czas obliczeń, szybkość zbieżności do optimum. Zwykle cechy te mają przeciwstawny charakter, w tym sensie, że wyznaczenie rozwiązania lepszego w sensie wartości funkcji celu wymaga dłuższego czasu działania algorytmu. Opisana poprzednio klasyfikacja pozwala na bardziej precyzyjne określenie metod(y) rozwiązywania, choć nie precyzuje żadnego konkretnego algorytmu. Bardziej szczegółowe omówienie metod rozwiązywania przedstawiono w rozdziale 2.11. Unikając świadomie pełnego przeglądu tego obszernego tematu, odsyłamy czytelnika do książki [154].

2.4. Rozmyte problemy szeregowania

Niepewność w zakresie sformułowania problemu można uwzględnić w tak zwanym modelu rozmytym. Zakładając, że dane (parametry) dla problemu są wielkościami rozmytymi, to harmonogram oraz wskaźnik jego oceny są także wielkościami rozmytymi. W przeglądowej pracy [13] został przedstawiony bilans problemów i metod szeregowania uwzględniający niektóre rozmyte parametry. Wymieniono tam m.in.: (A) żądany termin zakończenia (ang. *due dates*); (B) czas prze-

twarzania (ang. *processing time*); (C) relacja poprzedzania (ang. *precedence relation*); (D) opóźnienie realizacji poprzedzania (ang. *precedence delay*); (E) zakłócenia (ang. *disruption*); (F) awarie maszyn (ang. *machine breakdown*); (G) ograniczenia zasobów (ang. *resource constraint*). W przeglądzie [13] stwierdzono, że najczęściej rozpatrywanymi funkcjami celu w modelu rozmytym są: (a) długość uszeregowania (ang. *makespan*); (b) suma czasów wykonania wszystkich zadań (ang. *total completion time*); (c) czas przepływu (ang. *flow time*).

Szczegóły podejścia rozpoczniemy od podstaw logiki wielowartościowej (rozmytej), stosowanej głównie do problemów sterowania, patrz przykładowo autonomiczna nawigacja robota [109]. Sięgając do genezy koncepcji, elementem kluczowym było sformalizowanie pojęcia zbioru rozmytego oraz podstawowych operacji na elementach tego zbioru. Prekursorem w tej dziedzinie był Zadeh, patrz np. [175]. Warto tu także wspomnieć o zastosowaniu języka naturalnego do opisu zjawisk zachodzących w obrębie logiki rozmytej, [90, 108, 131, 141].

Zbiory rozmyte stosowane mogą być również do *jakościowej* zamiast *ilościowej* charakterystyki danych, co pozwala na inne (lingwistyczne) postrzeganie modelu matematycznego [108, 131, 175]. Możliwe jest zastosowanie następujących szczegółowych podejść wykorzystujących logikę wielowartościową [90, 135, 173, 175]: (A) teoria zbiorów przybliżonych, (B) teoria rozmytych liczb skierowanych, (C) teoria liczb rozmytych. W niniejszej rozprawie wykorzystano głównie podejście (C). Przyjmuje się, że każda dana (sygnał wejściowy) przyjmuje nieokreśloną wartość należącą do pewnego zbioru (rozmytego), w stopniu określonym funkcją przynależności. Działania na zbiorach i danych rozmytych wymagają zastosowania dedykowanej, niestandardowej arytmetyki. Koncepcja ta wraz z różnymi funkcjami przynależności oraz różnymi operacjami arytmetycznymi została szeroko omówiona w literaturze, między innymi w: [90, 105, 108, 131, 139, 141, 162, 175]. Zauważmy, że istnieją dwie rozłączne ścieżki wprowadzenia modelu rozmytego. Pierwsza polega na rozmytych wartościach ilościowych wejścia plus konwencjonalna arytmetyka liczb rozmytych. Druga polega na lingwistycznych wartościach wejścia oraz regułach wnioskowania. Dla potrzeb optymalizacji uszeregowania odwołujemy się do typowego systemu rozmytego działającego według następującego schematu. Dane wejściowe są dostarczane bądź w formie rozmytej,

bądź celowo rozmywane (fuzyfikacja). Dalej następuje proces podejmowania decyzji (na danych rozmytych) z użyciem zdefiniowanych wcześniej reguł wnioskowania lub arytmetyki rozmytej. W końcowym etapie realizowany jest proces wyostrzania (defuzyfikacji), aby otrzymać wynik ilościowy [139, 143]. Zatem system posiada strukturę sekwencyjną złożoną z następujących bloków: (A) blok fuzyfikacji; (B) blok wnioskowania; (C) blok defuzyfikacji.

Blok fuzyfikacji przekształca dane wejściowe z postaci liczbowej do jakościowej lub wielowartościowej oraz określa funkcje przynależności, zwykle formułowaną w postaci parametrycznej. Postać i parametry funkcji przynależności mają znaczący wpływ na dokładność i jakość rozpatrywanego modelu [131, 141, 168].

Blok wnioskowania, zwany również inferencyjnym, na podstawie funkcji przynależności danych oblicza wynikową funkcję przynależności wyniku, poprzez zastosowanie reguł wnioskowania [131, 168] lub rozmytych operacji arytmetycznych. Przeprowadzenie inferencji musi zostać poprzedzone wyborem następujących elementów: (A) baza reguł, sformułowana z zasad logicznych określających zależności przyczynowo-skutkowe istniejące w systemie pomiędzy rozmytymi wejściami i wyjściami, określona poprzez zbiór instrukcji warunkowych w postaci: (a) IF ... THEN, dla reguł prostych, (b) IF ... AND ... OR ... THEN, dla reguł złożonych; (B) mechanizm inferencyjny, realizujący obliczanie wynikowej funkcji przynależności, zawierający: (B-1) obliczania stopnia spełnienia przesłanek poszczególnych reguł, (B-2) obliczanie stopnia aktywizacji konkluzji poszczególnych reguł, [4]. (B-3) określenie wynikowej funkcji przynależności na podstawie stopni aktywizacji reguł; (C) obliczanie funkcji przynależności wyjścia.

Defuzyfikacja jest procesem konwersji wynikowej (wyjściowej) funkcji przynależności na wartość liczbową rzeczywistą. Do jednych z najpopularniejszych należą metody [131, 168]: (A) środka maksimum, która uwzględnia wpływ tylko najbardziej zaktywizowanego zbioru, przy czym stopień aktywizacji tylko w nieznacznym stopniu wpływa na metodę; należy uwzględnić czułość defuzyfikacji, a w rezultacie czułość całego modelu rozmytego na zmiany stopnia aktywizacji;

[4] Obliczenie stopnia aktywizacji konkluzji poszczególnych reguł (R_k) jest równoznaczne z określeniem stopnia przynależności μ relacji będących wynikiem operacji złożenia zbioru rozmytego (wyniku przesłanki) i relacji rozmytej (reguły) [70]

(B) pierwszego maksimum, odznacza się identycznym stopniem skomplikowania jak metoda poprzednia, wykazując jednocześnie większą wrażliwość na stopień aktywizacji; wadą jest uwzględnienie tylko jednego najbardziej zaktywizowanego zbioru; (C) ostatniego maksimum, o cechach podobnych jak w przypadku B; (D) środka ciężkości, która bierze pod uwagę wszystkie zaktywizowane funkcje przynależności rozpatrywane w procesie defuzyfikacji; wymagany jest w tym przypadku duży nakład obliczeniowy związany z całkowaniem powierzchni o nieregularnym kształcie, zależnym od użytych funkcji przynależności; (E) środka sum, charakteryzująca się mniejszą złożonością obliczeniową niż D oraz udziałem wszystkich reguł w procesie wnioskowania; (F) wysokości, która stanowi uproszczoną wersję E; ma mniejszy nakład obliczeniowy oraz charakteryzuje się ciągłością i wrażliwością na stopień aktywizacji funkcji.

Klasyfikacja szeregowania wykorzystującego logikę rozmytą obejmuje, podobnie jak w przypadku deterministycznym, problemy jedno- oraz wielomaszynowe, a także problem przepływowy, zadaniowy oraz otwarty [13]. Z analizy przeprowadzonej w pracy [73] wynika, że w modelu najczęściej uwzględniany tylko jeden rodzaj parametru jako niepewny. Spośród ogółu rozpatrywanych prac dotyczących szeregowania z uwzględnieniem niepewności tylko 21% związanych jest z logiką rozmytą, natomiast 57% stanowią systemy, w których uwzględnia się prawdopodobieństwo. Z analizy zawartej w pracy [23] wynika, że część badaczy łączy specyfikacje parametrów rozmytych z wyznaczaniem optimum za pomocą algorytmów ewolucyjnych czy też genetycznych. W rozważanych przepływowych problemach szeregowania stosowane są podejścia oparte na schemacie podziału i ograniczeń, czy też dedykowane algorytmy pochodzące od Johnsona, McMahona oraz Lee (patrz przegląd np. [73]). W znacznej większości podejść rozpatrywana jest minimalizacja rozmytego terminu zakończenia zadań.

W pracy [1] zaprezentowano wielokryterialną optymalizację z użyciem logiki rozmytej wykorzystując do wyznaczenia harmonogramu symulowane wyżarzanie oraz metodę tabu. W założeniach tej pracy lokalne sąsiedztwo jest określane poprzez zmienne rozmyte. Stosując w tym przypadku lingwistyczne zmienne do opisu lokalnego sąsiedztwa autorzy osiągnęli rezultaty lepsze niż w przypadku tradycyjnych metaheurystyk bez zastosowania ich rozszerzenia. Kolejny ciekawy

przypadek obejmuje zastosowanie rozmytego procesu decyzyjnego poprzez wprowadzenie czasów ograniczeń jak i priorytetów z wykorzystaniem zmiennych lingwistycznych. Wykorzystując 11 reguł opartych o wnioskowania bazujące na logice rozmytej, uzyskano w efekcie system, który efektywniej szereguje zadania o najwyższych priorytetach w systemach czasu rzeczywistego, a także efektywniej wykorzystuje sprzętowe zasoby systemowe [153] w porównaniu do istniejących systemów. W pracy [78] zaproponowano szeregowanie zadań na maszynach równoległych, specyfikując czas wykonania zadań jako zmienne rozmyte opisane trójkątną funkcją przynależności. Celem tego problemu optymalizacyjnego jest minimalizacja czasu opóźnienia zadań oraz uszeregowania zadań opóźnionych dzięki zastosowaniu algorytmu EDD (ang. *Earliest Due Date*).

W pracy [161] rozważany jest problem elastycznego planowania tras pojazdów, co wpływa na wydajność transportu, a także zmniejszenie kosztów logistycznych. Do optymalizacji harmonogramu zaproponowano trójkątną funkcję przynależności opisującą koszt transportu. Ponieważ w przypadku transportu nie jest możliwe oszacowanie dokładnego kosztu przed jego zakończeniem, tworzone są zatem różne scenariusze przetwarzania. Następnie w wyniku działania algorytmu genetycznego oraz algorytmu optymalizacji rojem wykonywana jest minimalizacja maksymalnych wartości czasowych dla transportu. Problem gniazdowy w zakresie logiki rozmytej jest wykorzystywany w pracy [64], w dwuetapowym algorytmie szeregującym zadania realizowane w obrębie elastycznego systemu produkcyjnego. Niepewność jest w tym przypadku wykorzystywana do wyznaczania priorytetów, uwzględniając złożenie trzech trójkątnych funkcji przynależności. Dla typów części w pierwszym etapie oraz docelowo w drugim etapie przedstawiony został algorytm alokacji oraz szeregowania realizowanego w procesie produkcyjnym. Często logika rozmyta wykorzystywana była w połączeniu z algorytmami ewolucyjnymi, co zostało zaprezentowane w zestawieniu [73]. Również w pracy [103] został użyty algorytm genetyczny. Sterowany algorytm genetyczny użyty w tej pracy wykorzystuje złożone funkcje przynależności do wyboru operatorów mutacji lub krzyżowania w proponowanym algorytmie. Kolejna praca [121] zawiera zestawienie oraz analizę systemów opartych o logikę rozmytą. Analizie poddane są w tym przypadku systemy sterowania, które odwołują się do schematu

systemu rozmytego przedstawionego w pracy [142]. Z przedstawionego zestawienia wynika, że niezależnie od gałęzi przemysłu wykorzystywane są różne modele logiki rozmytej ze szczególnym uwzględnieniem modeli dynamicznych i tak zwanych modeli T-S (ang. *Takagi–Sugeno fuzzy models*). Podobnie jak w przypadku omawianych już prac, także w tym przypadku logika rozmyta była rozpatrywana w połączeniu z wybranymi algorytmami. Omówione zostało tutaj także zagadnienie stabilności sterownika w oparciu o funkcję Lapunowa. W pracy [78] rozpatrywany jest przegląd oraz wyzwania dla systemów szeregowania z zagadnieniem niepewności, gdzie rozpatruje się ją w trzech postaciach: jako logika rozmyta, opis probabilistyczny oraz ograniczone formy. W zestawieniu tym metody szeregowania podzielono na dwie grupy, w zależności od sposobu postrzegania niepewności w systemie i tym zakresie wyróżniono: harmonogramowanie reaktywne i harmonogramowanie zapobiegawcze. Grupa pierwsza obejmuje szeregowanie dynamiczne, zmianę harmonogramu oraz szeregowanie online, zapewniając przy tym optymalne szeregowanie w przypadku występowania niepewności w systemie. Planowanie zapobiegawcze obejmuje zaś: (A) planowanie stochastyczne; (B) systemy odporne (ang. *robust systems*); (C) metodę programowania rozmytego; (D) metodę programowania parametrycznego.

Algorytm genetyczny w połączeniu z metodą tabu zaproponowano w pracy [65] podejmującej wielokryterialne podejście do szeregowania zadań uwzględniające niepewne parametry. Użyta w tym przypadku logika rozmyta służy do specyfikacji określonych czasów ograniczeń, które są modelowane w oparciu o czasy wykonania zadań. Jest to kolejne podejście do szeregowania z wykorzystaniem niepewności stosujące algorytm hybrydowy. W pracy [65] zauważono, że obecność rozmytych danych powoduje również rozmytość celów szeregowania. Zaproponowany w tym podejściu hybrydowy algorytm, w celu wyszukania docelowego uszeregowania maksymalizuje zagregowaną ocenę stopnia satysfakcji wyznaczaną przy rozpatrywaniu średniej arytmetycznej wszystkich stopni satysfakcji.

Praca [13] obejmuje kompletną analizę szeregowania zadań w związku z artykułem [23]. Praca ta skupia się na analizie wykorzystania logiki rozmytej także w problemach jednomaszynowych, równoległych, otwartych, zadaniowych oraz przepływowych. W przypadku problemów jednozadaniowych procesowi fuzyfika-

cji poddawane są najczęściej czas ograniczenia oraz czas przetwarzania zadania, często łączone z algorytmami heurystycznymi oraz metaheurystycznymi. Mimo że formalnie rozpatrywanych jest wiele celów optymalizacyjnych, to zdecydowana większość prac dotyczy długości uszeregowania (ang. *makespan*). W obrębie problemu zadaniowego niemal we wszystkich pracach rozmywany jest czas przetwarzania, zatem kryterium optymalności jest (rozmyta) długość uszeregowania. Podobne podejście do rozwiązywania problemów harmonogramowania jest zastosowane dla problemów otwartych, gdzie rozpatrywane jest wykorzystanie heurystyk oraz metaheurystyk do systemów z wyspecyfikowanymi rozmytymi czasami przetwarzania zadań. Z omawianego przeglądu w [13] wynika, że w okresie (1993 - 2014) model rozmyte były stosowane do: (A) Problem gniazdowy (ang. *job shop*) (33.3%), (B) Problem przepływowy (ang. *flow shop*) (28.1%), (C) Problem jednomaszynowy (ang. *single machine*) (25.1%), (D) Problem maszyn równoległych (ang. *parallel machine*) (10.4%), (E) Problem otwarty (ang. *open shop*) (3.0%).

Najczęściej wykorzystywanymi w tym celu są algorytmy genetyczne, a także schemat podziału i ograniczeń (ang. *branch and bound, B&B*). Dalej, używane są metaheurystyki różnego rodzaju, przykładowo algorytmy stadne, mrówkowe, itp. W wielu przypadkach wykorzystywany jest również algorytm hybrydowy w różnych konfiguracjach. W pracy tej pokazany został znaczny wzrost zainteresowania tematyką szeregowania zadań z niepewnymi parametrami w ostatnich latach. Logika rozmyta znajduje również zastosowanie w systemach opartych o architekturę chmury, czy też połączeniem wielu systemów zorganizowanych w obrębie takiego rozwiązania. Podejście takie wykorzystywane jest do zapewnienia oraz polepszenia jakości usług systemu, w odniesieniu do preferencji klientów dotyczących QoS (ang. *Quality of Service)* [62]. Logika rozmyta znajduje również zastosowanie w analizach rozpatrujących połączenie z serwomechanizmami na przykładzie pracy [138] gdzie opisano ramię robota identyfikujące wyciek gazu. W omawianym przypadku stosowane jest ramię robota o trzech punktach swobody wykorzystującego cztery sensory. Zebrane dane wejściowe z czujników są następnie poddane fuzyfikacji, zgodnie z ogólnie przyjętą koncepcją budowy systemów rozmytych, a następnie przy wykorzystaniu złożenia trzech funkcji przynależności podejmowana jest decyzja dotycząca reakcji ramienia robota na źródło zidentyfi-

kowane przez sensory.

W przepływowych problemach szeregowania zadań wykorzystujących logikę rozmytą do opisania niepewności najczęściej do rozwiązania problemu stosowane są algorytmy heurystyczne lub metaheurystyczne [13, 23]. Podobne rozwiązania, choć nie w tak dużym stopniu intensywności są wykorzystywane w systemach opartych o rozwiązania chmurowe czy też w zakresie robotyki. Jako interesujące kierunki rozwoju wskazać można wyznaczenie optymalnego uszeregowania dla systemów, w których specyfikacje obejmują więcej niż jeden rodzaj parametru jako niepewny, przykładowo: czas przepływu, całkowity czas wykonania, opóźnienie, a także podejścia badające wielu niepewnych parametrów jednocześnie. Rozpatrując jakiekolwiek parametry niepewne, czy to związane z logiką rozmytą, czy też probabilistyką, należy wziąć pod uwagę stabilność systemu [23, 73]. Ten istotny parametr systemu pozwala na określenie przydatności proponowanego systemu szeregowania [23]. Szeregowanie z uwzględnieniem niepewności pozwala na opracowanie algorytmów, które w lepszym stopniu odzwierciedlają rzeczywiste potrzeby w różnych gałęziach przemysłu [13, 23].

2.5. Stochastyczne problemy szeregowania

Tym mianem określa się problemy posiadające ustalony zbiór zadań o danych parametrach będących realizacjami zmiennych losowych o znanych lub estymowanych rozkładach prawdopodobieństwa. Chociaż w problemach szeregowania dążymy do unikania nadmiernego skomplikowania modelu, problemy stochastyczne prowadzą do dość złożonych modeli i algorytmów rozwiązywania. Wymieniane są co najmniej trzy argumenty dla przyjęcia takiego poglądu [61]: (A) kombinatoryczny wzrost liczby zmiennych decyzyjnych; (B) trudności w ocenie funkcji celu spowodowane zastosowaniem zmiennych losowych; (C) warunki, które zmieniają się znacząco w zależności od pozycji, przy czym ten ostatni wymaga całej serii rozwiązań w czasie, zamiast pojedynczego rozwiązania w danym momencie. Do opisu czasu trwania zadania, stosuje się rozkład geometryczny funkcji prawdopodobieństwa (czas dyskretny), lub rozkład wykładniczy (czas ciągły), [112]. Jednym z częściej stosowanych podejść do modelowania i rozwiązywania jest na-

turalna konwersja modelu deterministycznego w model stochastyczny, np. poprzez analizę wartości średnich odpowiednich rozkładów, metoda MVA (ang. *Mean Value Analysis*) lub poprzez analizę scenariuszy. Stochastyczne algorytmy szeregowania można podzielić na, (patrz [122]): (A) dwuetapowe; (B) wieloetapowe; (C) podejście oparte na programowaniu z ograniczeniami.

Statystyka sięgając swoimi korzeniami do historii [127], gdzie opisywała podstawowe potrzeby państw czy zagadnienia społeczne, rozwinęła się na przestrzeni lat aż do czasów współczesnych, proponując rozkłady prawdopodobieństwa opisujące w dokładniejszy sposób wybrane zjawiska czy też właściwości [12, 110, 155]. Zgodnie ze obecnym stanem wiedzy z wykorzystaniem podejścia stochastycznego można specyfikować czasy wykonania zadań [89, 163], najgorszego czasu wykonania zadania [114, 147], znajdując zastosowania w algorytmach gwarantujących poprawne uszeregowania zadań okresowych [43]. Przedstawione metody są również wykorzystywane w celu szeregowania w systemach o podwyższonej niezawodności [128] oraz w tak zwanych systemach odpornych (ang. *robust*) [146].

Praca [119] porównuje szeregowanie uwzględniające logikę rozmytą oraz probabilistykę. Z zestawień wynika, że zastosowanie logiki rozmytej do metod opartych o analizę PERT (ang. *Program Evaluation and Review Technique*) wykazuje wadę związaną z wyznaczaniem maksymalnej wartości rozmytej. Analiza pozwala na określenie, że podejście z logiką rozmytą nie wymaga, jak w przypadku metod stochastycznych, zastosowania identycznego rozkładu dla wszystkich rozpatrywanych czasów zadań. Dodatkowo ustalone zostało, że operacje arytmetyczne w przypadku rozkładów prawdopodobieństwa są bardziej skomplikowane niż przy zastosowaniu logiki rozmytej. Porównanie przeprowadzone w połowie lat dziewięćdziesiątych XX-wieku do dziś przedstawia aktualne tendencje w zakresie wymienionych podejść do szeregowania zadań.

W pracy [150] założono, że probabilistyka jest wykorzystywana do określenia prawdopodobieństwa zakończenia zadania, a co za tym idzie wpływu tego zdarzenia na kolejno realizowane zadania. Rozpatrywany jest w tym przypadku SJSS (ang. *Stochastic Job Shop Scheduling Problem*), w którym inaczej niż zazwyczaj, niepewny nie jest jeden z parametrów w procesie szeregowania, ale niepewność obejmuje powodzenie (lub jego brak) wykonania zadania produkcyjnego. Zapro-

ponowany tutaj model systemu optymalizuje koszt poprzez zaproponowaną heurystykę. Przeanalizowany w tym przypadku rzeczywisty proces frezowania pokazuje wpływ rezultatu operacji na zakończenie zadania przed terminem oraz ewentualny wpływ na konieczność powtórzenie operacji poprzedzających rozpatrywane zadania. Użyte w tym celu podejście stochastyczne pozwala na dokładniejsze odzwierciedlenie rzeczywistego systemu, zakłada opisanie prawdopodobieństwa zakończenia zadania z wykorzystaniem normalnego rozkładu prawdopodobieństwa. W przypadku tym stosowana jest zasada 3σ ściśle związana z rozkładem normalnym. W pracy [172] jest rozpatrywane wyszukiwanie błędów współbieżności w losowo generowanym harmonogramie. Testowanie odbywa się poprzez wielokrotne uruchomienie programu z danymi, gdzie zakłada się, że parametr głębokości błędu jest wybieranym losowo rozkładem prawdopodobieństwa. Analiza probabilistyczna dla ustalonych priorytetów szeregowania z wywłaszczaniem dla mieszanych systemów krytycznych tworzonych w schematach AMC (ang. *Adaptive Mixed Criticality*) oraz SMC (ang. *Static Mixed Criticality*). W zaprezentowanej analizie za pomocą mieszanych rozkładów prawdopodobieństwa jest opisywany najgorszy czas wykonania, najgorszy czas odpowiedzi oraz najgorsze możliwe przekroczenie czasu ograniczeń. Zastosowanie probabilistyki jako elementu niepewności zostało przedstawione w pracy [146] podejście do szeregowania w systemach odpornych (ang. *robust*). Zaproponowany w pracy algorytm oparty na ograniczeniach oraz wykorzystujący probabilistyczny formatymlizm wnioskowania czasowego zwany PSTP (ang. *Probabilistic Simple Temporal Problem*). Niepewnymi parametrami specyfikowanymi jako rozkłady prawdopodobieństwa są czasy trwania zadań. Zaproponowana metoda szeregowania maksymalizuje prawdopodobieństwo otrzymania optymalnego i działającego poprawnie systemu. Autorzy pracy [128] zakładają utworzenie systemu szeregowania zadań okresowych w systemach złożonych z identycznych procesorów. W tym przypadku zaproponowano duplikację zadań, biorąc pod uwagę maksymalne tolerowane prawdopodobieństwo awarii, minimalizując rozmiar całej proponowanej platformy, tak aby przynajmniej jedno z replikowanych zadań zostało uszeregowane, spełniając założone ograniczenia. Nie zakłada się w tym przypadku zdefiniowanej a priori liczby awarii, jest to określane za pomocą prawdopodobieństwa. Praca [147] poddaje ba-

daniu systemy czasu rzeczywistego. Zaproponowany w tym przypadku model szeregowania rozpatruje zarówno systemy z miękkimi jak i twardymi ograniczeniami czasowymi. W tym przypadku zarówno dostarczane zasoby jak i zapotrzebowanie na zasoby wykonujące zadania jest modelowane w kategoriach granic probabilistycznych. W tym modelu założono występowanie probabilistycznych okresowych zadań powiązanych z ograniczeniami czasowymi. W pracy tej każde zadanie jest opisane za pomocą trzech parametrów, z których każdy jest opisany rozkładem probabilistycznym, są to odpowiednio: (A) pWCET, probabilistyczny najgorszy czas wykonania (B) pMIT, probabilistyczny minimalny czas przybycia (C) D_i, czas ograniczenia wykonania zadania

W pracy [163] jest rozpatrywany problem zadaniowy z niezależnie generowanymi probabilistycznymi czasami trwania zadań. W podejściu tym użyto algorytmu Monte Carlo oraz B&B do wyznaczania górnego ograniczenia długości uszeregowania, oraz częściowego rozwiązania, następnie definiowany jest problem zadaniowy z dolnym ograniczeniem tego kryterium. Następnie dwie wyznaczone części są wykorzystywane do wyznaczenia uszeregowania z wykorzystaniem programowania z ograniczeniami lub metody tabu search. Zazwyczaj zakłada się wykorzystanie koncepcji stochastycznej do specyfikacji czasów trwania zadań, jak zostało to wykonane w pracach [22, 35, 89]. Autorzy założyli w tym przypadku specyfikację czasu, reprezentowanych w postaci grafu, zadań z wykorzystaniem rozkładu prawdopodobieństwa Erlanga. W szeregowaniu zadań, w tym przypadku badania wykorzystują z powodzeniem algorytm, bliski optymalnemu, HLF (ang. *Highest Levels First*). W pracy [126] zaproponowano wykorzystanie czterech metod redundancji w problemie szeregowania zadań. Wykorzystywane podejście dla oszacowania czasu trwania dla każdej metody jest nieco inne, ale wyróżnia się wspólne elementy takie jak: (A) Podstawowy czas trwania aktywności, parametr opisywany z wykorzystaniem rozkładu prawdopodobieństwa w idealnych warunkach działania, (B) Identyfikacja ryzyka, w przypadku tego projektu zostało losowo wygenerowane, a następnie przypisane do działań w ramach projektu, (C) Prawdopodobieństwo wystąpienia ryzyka. Dla każdego czynnika ryzyka jest określone prawdopodobieństwo jego wystąpienia, (D) Dodatkowe zadanie, wykonywane w przypadku wystąpienia ryzyka.

Zaprezentowane w tym przypadku metody pozwalają na tworzenie stabilnych systemów z wykorzystaniem prawdopodobieństwa. W pracy [130] prawdopodobieństwo nie jest wprost wykorzystywane w szeregowaniu. W przypadku tym metody losowe służą do generacji oraz wyznaczenia optymalnego ruchu ramienia robota, biorąc pod uwagę rzeczywistą trajektorię ruchów człowieka na stanowisku montażowym, a także zajętość operatora. Przedstawiony sposób generacji jest interesujący ze względu na praktyczną możliwość szeregowania bazującą wprost na koncepcji stochastycznej. Kolejnym wyzwaniem dla zastosowania niepewności jest szeregowanie w środowiskach opartych o rozwiązania chmurowe, co przedstawione zostało w pracy [46]. Zaproponowano w tym przypadku probabilistyczne podejście do udostępniania zasobów oraz planowania zadań, co wpływa na oszacowania, które zasoby mogą być udostępniane, zmniejszając w ten sposób ryzyko i wpływ przeszacowania lub niedopasowania zasobów.

2.6. Adaptacyjne szeregowanie online

W harmonogramowaniu *offline* uszeregowanie jest wyznaczane a priori dla wszystkich zadań przy pełnej informacji o nich. Często używa się tym celu bądź priorytetów statycznych (sekwencja zadań jest otrzymywana zgodnie z uporządkowanymi wartościami priorytetów), bądź też priorytetów dynamicznych (rozszerzając krokowo sekwencję częściową zadań). Złożoność obliczeniowa czasowa algorytmów tego typu nie jest znacząca. Zaletę tego podejścia jest także możliwość względnie prostej implementacji dodatkowych ograniczeń, takich jak czasy oczekiwań czy preferencje zadań, [19]. W szeregowaniu *online* proces przypisywania priorytetów odbywa się na bieżąco, w trakcie funkcjonowania systemu, przy nieznanym z góry zbiorze zadań napływających, realizując ustaloną politykę szeregowania. Podejście to jest bardziej elastyczne niż offline. W literaturze wymieniana jest także trzecia grupa metod szeregowania stanowiąca połączenie cech dwóch wymienionych klas, nazywana szeregowaniem semionline [11, 34].

Zasadniczo, wymieniane są najczęściej dwa podstawowe modele szeregowania online [26, 42, 58, 132], chociaż w pracy [4] pojawia się również inny (trzeci) model. Pomijając różnice, wszystkie modele można opisać w następujący zuni-

fikowany sposób: (A) lista online (ang. *online list)*; (B) czas online (ang. *online time*); (C) aktywność maszyn (ang. *machine activity*). Pojęcie, *lista online* określa system, gdzie parametry zadania (czas wykonywania, pilność, etc.) stają się znane dopiero w chwili pojawieniu się zadania w systemie. Na podstawie tych parametrów dokonuje się wpisania zadania na listę zadań oczekujących, z której pobierane są następnie zadania do wykonania. Kolejność wpisania na listę jest zgodna z kolejnością napływu zgłoszeń. W *online time* nieznane są ani momenty pojawienia się zadania w systemie, ani czas trwania zadania aż do chwili zakończenia tzw. czasu krytycznego dla obsługi zadań. Dodatkowo w tym modelu istnieje możliwość pojawienia się kilku zadań w tej samej chwili czasowej. Trzeci model skupia się na szeregowaniu z uwzględnieniem nieoczekiwanych przestojów maszyn spowodowanych przezbrojeniami, losowymi awariami czy też nieplanowaną konserwacją. Ocenę jakości algorytmów onlineowych przeprowadza się poprzez tzw. *analizę konkurencyjności*. Polega ona na porównaniu a posteriori wartości funkcji celu rozwiązania online z odpowiednim rozwiązaniem offline. Wymienia się także inne metody oceny stanowiące alternatywę do analizy konkurencyjności (faktycznie analizy pesymistycznej) dla algorytmów online. Za pracą [11], indywidualne podejścia oceny algorytmów mogą uwzględniać: (A) algorytmy zrandomizowane (ang. *randomized algorithms*); (B) techniki rozszerzenia zasobów (ang. emph resource augmentation); (C) algorytmy częściowo online (ang. *semionline algorithms*); (D) analiza przypadków średnich (ang. *average-case*).

Zasadnicza różnica pomiędzy szeregowaniem online i offline to dostępność informacji o danych: (a) offline – przed rozpoczęciem procesu szeregowania; (b) online – udostępniane stopniowo w trakcie realizacji procesu szeregowania. Dodatkowymi argumentami do wyboru podejścia (b) mogą być niekompletność informacji o danych, niepewność lub indeterminizm zdarzeń czasowych [34, 93]. Wymienia się następującą klasyfikację algorytmów szeregowania online [11, 84]: (A) przewidujące (ang. *clairvoyant*), w których czas obsługi zadania jest znany w chwili zgłoszenia, (B) nie-przewidujące (ang. *non-clairvoyant*), w których czas obsługi zadań pozostaje nieznany do zakończenia obsługi. Do grupy pierwszej, zalicza się następujące algorytmy: (A) Najkrótszy pozostały czas wykonania SRPT (ang. *Shortest Remaining Processing Time*), (B) Najkrótsze zadanie SFJ (ang,

Shortest Job First), (C) Najwyższy priorytet/przeznaczenie pierwsze HDF (ang. *Highest Destinity First*), Do standardowych algorytmów nie-przewidujących zaliczyć można [11]: (A) Algorytm Karuzelowy RR (ang. *Round Robin*); (B) Ważony Algorytm Karuzelowy WRR (ang. *Weighted Round Robin*); (C) Pierwszy Zgłoszony Pierwszy Obsłużony FIFO (ang. *First In First Out*); (D) Najkrótszy Pozostały Czas Wykonania SETF (ang. *Shortest Elapsed Time First*); (E) Ważony Najkrótszy Pozostały Czas Wykonania WSETF (ang. *Weighted Shortest Elapsed Time First*); (F) Przydział Najnowszego Procesora LAPS (ang. *Latest Arrival Processor Sharing*); (G) Ważony Przydział Najnowszego Procesora WLAPS (ang. *Weighted Latest Arrival Processor Sharing*); (H) Wielopoziomowe Sprzężenie Zwrotne MLF (ang. *Multi-Level Feedback*). W praktyce najczęściej stosowane są: (A) Algorytm listowy, [56]; (B) SRPT, [132]; (C) FIFO; (D) LAPS, [77]; (E) SJF, (F) HDF, [77]; (G) RR, [178]; (H) SETF; (I) MLF, [8, 170].

Nieznany a priori napływ zadań do wykonywania generuje nietrywialne zagadnienie oceny jakości zastosowanych algorytmów szeregowania on-line. Na rysunku 2.1 przedstawiona została typowa krzywa średniej wydajności (ang. *performance*) systemów klient-serwer w funkcji obciążenia (ang. *load*). Podczas konstruowania algorytmów online zazwyczaj dąży się do uzyskania lub zbliżenia się do rozwiązania optymalnego wyznaczanego algorytmem offline. Do oceny algorytmów wprowadzono też kilka innych miar, patrz m.in. [42]: (A) Granice wykorzystania (ang. *Utilization Bounds*), (B) Współczynnik przybliżenia (ang. *Approximation Ratio*), (C) Zwiększenie zasobów (ang. *Resource Augmentation*), (D) Miary empiryczne (ang. *Empirical Measures*).

Rysunek 2.1: Krzywa wydajności algorytmu. Źródło : [11]

Granice wykorzystania (dolna, górna) pozwalają na oszacowanie zakresu wy-

dajności algorytmu A dla szeregowania zadań z nieokreślonym a priori czasem wykonywania. Pesymistyczna ocena wydajności algorytmu A (ocena najgorszego przypadku) jest formułowana tak, by była słuszna dla dowolnego zbioru zadań z nieokreślonymi czasami wykonywania.

Współczynnik przybliżenia ocenia wydajność rozwiązań generowanych rozpatrywanym algorytmem A do odpowiednich rozwiązań optymalnych. Zakładając problem określenia minimalnej liczby procesorów wymaganych do zaplanowania danego zestawu zadań τ. Dla liczby procesorów wymaganych według algorytmu optymalnego to MO(τ), oraz liczba wymaganej według algorytmu A to MA(τ) dokonano stosowanych kalkulacji; Obliczenie wartości współczynnika można znaleźć w pracy [42]. Przyjmuje on wartości ≥ 1. Wartość 1 jest osiągana jeśli A jest algorytmem optymalnym. Algorytm A nazywamy przbliżonym jeśli współczynnik ma wartość skończoną.

Współczynnik zwiększenia zasobów stanowi alternatywną do poprzedniej metodę porównywania. Prezentowany parametr, zamiast brać pod uwagę zwiększoną liczbę procesorów, które byłyby wymagane do uzyskania harmonogramu w ramach algorytmu A, współczynnik zwiększania zasobów uwzględnia wzrost szybkości przetwarzania, który byłby wymagany, przy założeniu liniowego skrócenia czasu wykonywania zadania wraz z szybkością przetwarzania. Zakłada się liniowy spadek czasu wykonania zadań wraz z szybkością przetwarzania.

Empiryczna miara wydajności algorytmu szeregowania może być wyznaczona poprzez ocenę a posteriori pewnej próbki losowo generowanych instancji. Zazwyczaj stosowane jest do porównania względnej wydajności dwóch lub większej liczby algorytmów przybliżonych. Eksperyment jest planowany czynnikowo pozwalając na badanie wpływu poszczególnych pojedynczych parametrów problemu na wartość miary.

Jak zostało przedstawione na rysunku 2.6 oraz szczegółowo opisane m.in. w pracy [26], algorytmy online są oceniane w ramach tzw. analizy konkurencyjności. Porównuje ona rozwiązania otrzymane algorytmem online do odpowiednich rozwiązań optymalnych otrzymanych a posteriori algorytmem offline. W przypadku niemożności wyznaczenia rozwiązania optymalnego offline oceniana jest zbieżność harmonogramu online do najlepszego rozwiązania offline osiąganego

w podejściu statycznym [19, 58].

Definicja współczynnika konkurencyjności rozróżnia rodzaj algorytmu, sposób napływu i wykorzystania danych. Współczynnik dla algorytmów zrandomizowanych definiowany jest zawsze w odniesieniu do pewnego *adwersarza* (przeciwnika). Zadaniem przeciwnika jest dobór takich danych wejściowych, aby działanie badanego algorytmu A zwracało możliwie najgorsze rozwiązanie. Adwersarz przygotowuje instancję danych wejściowych dla algorytmu A, a także sam wyznacza rozwiązanie, które będzie służyło jako rozwiązanie referencyjne. Adwersarz zawsze zna postać algorytmu A. Jednakże, jeśli algorytm A wyznacza rozwiązanie w sposób losowy, a zatem jest algorytmem zrandomizowanym, adwersarz nie zawsze potrafi wskazać, jakie rozwiązanie zostało wyznaczone przez algorytm. Wyróżniamy następujące rodzaje adwersarzy: adwersarz nieświadomy (ang. *Oblivious Adversary*), adwersarz adaptujący się (ang. *Adaptative Online Adversary*), silny adwersarz adaptujący się (ang. *Adaptative Offline Adversary*).

Definicja współczynnika r-konkurencyjności algorytmu A zależy od wyboru adwersarza. I tak w pracy [26] definiuje się następujące współczynniki:

- dla algorytmów deterministycznych:

$$f(A, I) \leq r \cdot f(OPT, I) + \alpha, \tag{2.1}$$

- dla algorytmów zrandomizowanych:

$$\mathbb{E}|f(A, I)| \leq r \cdot f(OPT, I) + \alpha, \tag{2.2}$$

- przeciwko adwersarzowi adaptującemu się:

$$\mathbb{E}|f(A, I)| \leq r \cdot \mathbb{E}|f(ADV, I)| + \alpha, \tag{2.3}$$

- przeciwko silnemu adwersarzowi adaptującemu się:

$$\mathbb{E}|f(A, I)| \leq r \cdot \mathbb{E}[f(OPT, I)] + \alpha, \tag{2.4}$$

Nierówności muszą być spełnione dla każdej instancji I. OPT oznacza algorytm optymalny. $\mathbb{E}(*, I)$ jest wartością średnią wyborów dokonanych odpowiednio przez A, OPT, ADV.

2.7. Szeregowanie odporne

Rzeczywiste problemy szeregowania muszą rozpatrywać wiele trudnych do oszacowania ryzyk. Sytuacje nieprzewidziane mogą wynikać z niepełnych lub niedokładnych danych, przypadkowych przerw w trakcie realizacji zadań, awarie maszyn czy też anulowanie niektórych zadań. Przedstawione zdarzenia wpływają na duży stopień niepewności w szeregowaniu, zwłaszcza w obszarze systemów produkcyjnych [107]. W związku z istnieniem nieprzewidywalnych sytuacji, czy też poszczególnych elementów systemu opracowano podejścia gwarantujące realizację harmonogramu z najlepszymi z możliwych wynikami w najgorszym przypadku [31]. Zakładając, że mianem problemu odpornego (ang. *robust*) określa się uszeregowanie niewrażliwe na nieprzewidziane zakłócenia.

Zaproponowano wiele podejść do rozwiązywania tego problemu, w literaturze wymienia się, co najmniej kilka sposobów klasyfikacji. Jeden z podziałów wymienia systemy: (A) reaktywne; (B) proaktywne; (C) hybrydowe. W proaktywnych systemach szeregowania z uwzględnieniem niepewności wymienia się:

- miary odporności (ang. *robustness measures*),
- techniki oparte na redundacji (ang. *redundancy based techniques*),
- metody probabilistyczne (ang. *probabilistic methods*),
- szeregowanie warunkowe (ang. *contingent scheduling*),
- techniki oparte na optymalizacji (ang. *techniques based on optimization*)

Z kolei wśród reaktywnych systemów szeregowania, wśród których wymienia się: (A) metody rozproszone (ang. *distributed approaches*); (B) metody scentralizowane (ang. *centralized approaches*); (C) zasady priorytetowe (ang. *priority rules*); (D) dynamiczne ustalanie priorytetów (ang. *dynamic choice of priority rules*).

Można przyjąć, że S oznacza uszeregowanie, które określa kolejność wykonywania poszczególnych operacji na zaproponowanych maszynach oraz $M_0(S)$ oznacza długość uszeregowania (ang. *makespan*) dla S bez żadnych zakłóceń, natomiast $M(S)$ oznacza długość uszeregowania dla S z wprowadzoną losową wartością zakłócenia [27]. Odporność uszeregowania jest określana jako

$$\delta = M(S) - M_0(S). \tag{2.5}$$

Do konstruowania odpornych harmonogramów używane są najczęściej tech-

niki oparte na redundancji czasowej i/ lub powielaniu zasobów. Redundancja czasowa ma na celu maskowanie awarii poprzez wprowadzenie zwiększonych okresów bezczynności pomiędzy zadaniami lub też sztucznego wydłużenia czasu trwania operacji. Powielenie zasobów daje możliwość wprowadzenie elastyczności do realizacji całego zbioru zadań [31].

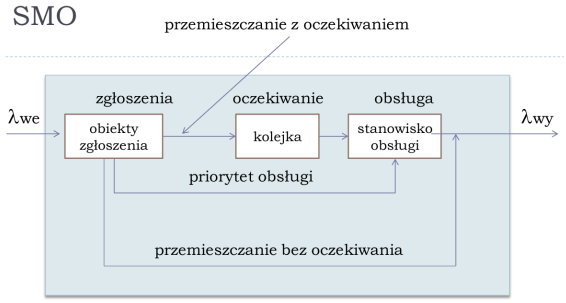
Projektowanie odpornych systemów szeregowania jest możliwe także przy wykorzystaniu metod probabilistycznych. Przykładowo w [31] wprowadzono pojęcie β-odporność (ang. β-*robustness*) dla problemu jedno-maszynowego z czynnikiem niepewności. Poszukiwany jest harmonogram, który maksymalizuje prawdopodobieństwo osiągnięcia określonej wydajności poprzez rozważanie całkowitego czasu przebywania zadania na maszynie. Z kolei koncepcja zwana *szeregowaniem warunkowym* zakłada przygotowanie wielu harmonogramów dla wielu możliwych wyników. Jedynym z bardziej znanych jej zastosowaniem jest szeregowanie JIC (ang. *Just-In-Case*) zaproponowane do harmonogramowania obserwacji przez teleskop z niepewnym czasem jej trwania [31].

Reaktywne systemy odporne skupiają się głównie na szeregowaniu realizowanym w czasie rzeczywistym, ponieważ realizacja harmonogramów offline jest w dłuższej perspektywie trudna do zrealizowania. Tego rodzaju odporne harmonogramowanie zazwyczaj jest stosowane w systemach z dużymi zakłóceniami, w których elementy niepewne występują często [31].

2.8. Systemy kolejkowe

Teoria kolejek jest dziedziną wiedzy wykorzystującą statystykę matematyczną i rachunek prawdopodobieństwa do analizy zachowania się systemów obsługi napływających losowo zgłoszeń w sposób nieskończony. Ma związek z badaniami operacyjnymi i matematyką stosowaną. Zasadniczym celem systemu obsługi jest minimalizacja czasu oczekiwania zgłoszenia na obsługę i/lub kosztów z tym związanych, poprzez dobór algorytmu obsługi. Systemy kolejkowe odwołują się do następujących elementarnych pojęć: (A) zgłoszenie – obiekt lub klient oczekujący na obsługę; zgłoszenia napływają ze źródła, które może być ograniczone lub nieograniczone. (B) kanały obsługi – jedno lub więcej „urządzeń” obsługi cha-

rakteryzujących się różną wydajnością. (C) kolejka – zbiór zgłoszeń oczekujących na obsługę, (D) obsługa – czynność pozwalająca na zaspokojenie określonych potrzeb, (E) regulamin – zbiór reguł dotyczących kolejki lub obsługi zgodnie z którymi zgłoszenia tworzą kolejkę i następnie są obsługiwane. Na rysunku 2.2 zo-



Rysunek 2.2: Schemat systemu masowej obsługi, źródło [67]

stał przywołany typowy schemat systemu obsługi, przy założeniu nieskończonego strumienia wejściowego jednorodnych zgłoszeń (tzw. system otwarty), gdzie λ_{we} i λ_{wy} oznaczają intensywności odpowiednich strumieni. Dla opisu systemów kolejkowych używana jest powszechnie notacja Kendall'a w postaci $X/Y/m$ gdzie: X – rozkład odstępów czasowych pomiędzy zgłoszeniami, Y – rozkład czasów obsługi, m – liczba kanałów. Symbole X oraz Y mogą przyjmować następujące wartości określające rozkłady: D – deterministyczny, M – wykładniczy, E_k – Erlanga, k-tego rzędu, H_r – hiperwykładniczy rzędu r, C_k – Cox'a rzędu k, GI – dowolny i niezależny, G – dowolny. W celu eliminacji pewnych niedogodności klasyfikacji Kendall'a zaproponowano jej dalsze rozszerzenia. I tak, notacja Lee używa zapisu $X/Y/m/d/l$, gdzie: d – kod polityki obsługi kolejki, zaś l – wymiarowość systemu. Oprócz przedstawionej klasyfikacji pojawia się rozróżnienie przez sposób tworzenia kolejki: (A) zabroniona, (B) dozwolona, (C) ze stratami, (D) bez strat. Według modelu Erlanga zakłada się, że wszystkie zgłoszenia zostają obsłużone w kolejności ich napływu. Praktyczne zastosowania często powołują się na typowe priorytetowanie, np.: FIFO (ang. *First in First Out*), SIRO (ang. *Selection in Random Order*), LIFO (ang. *Last in First Out*). Niekiedy wprowadza się priorytet bezwzględny, oznaczający konieczność wywłaszczenia obsługiwanych zgłoszeń. Bardziej dokładne omówienie systemów kolejkowych można znaleźć

w książce [67].

W systemach kolejkowych dane zadania są zmiennymi losowymi, zatem analizę prowadzi się dla miar probabilistycznych, przykładowo wartości średnie wielkości poniżej, patrz szersze omówienie w pracach [66–68, 124]:

- intensywności napływu zgłoszeń (λ) oraz obsługi zgłoszeń (μ), gdzie T_λ - średni czas pomiędzy zgłoszeniami, T_μ - średni czas obsługi zgłoszenia,

$$\lambda = \frac{1}{T_\lambda}, \quad \mu = \frac{1}{T_\mu}, \tag{2.6}$$

- p - prawdopodobieństwo zajętości stanowiska,

$$p = \sum_{i=1}^{n} \frac{\lambda_i}{\mu_i} \tag{2.7}$$

- w_i - średni czas oczekiwania na wykonanie zadania o priorytecie i dla zadań z wywłaszczaniem, gdzie m_{2j} (patrz [68]),

$$\omega_i = \frac{\sum_{j=1}^{K} \lambda_j m_{2j}}{2(1 - \sum_{j=1}^{i-1} p_j)(1 - \sum_{j=1}^{i} p_j)} \tag{2.8}$$

- S_i - czas potrzebny do obsłużenia zadań o wyższych priorytetach niż zadania realizowanego

$$S_i = \frac{\sum_{j=1}^{i-1} p_j}{1 - \sum_{j=1}^{i-1} p_j} \tag{2.9}$$

- q_i - średni czas przebywania zadania i w systemie

$$q_i = \omega_i + S_i + \frac{1}{\mu_i} \tag{2.10}$$

W kontekście problemu rozważanego w rozprawie należałoby rozpatrywać system kolejkowy zamknięty, z różnymi strumieniami zgłoszeń, z obsługą wielokanałową synchroniczną, lub nietypową polityką obsługi w oparciu o zasoby. Taki model nie jest znany w literaturze przedmiotu.

2.9. Stabilność algorytmów

Zakładając niepewność danych wprowadza się parametr liczbowy w celu scharakteryzowania stabilności algorytmu. Stabilność można rozumieć jako parametr

pozwalający na ocenę wpływu niepewności (zaburzenia) danych na zmianę wartości funkcji celu [24, 25]. W pracach tych zaproponowano podejście do stabilności uwzględniające zakłócenia zarówno dla deterministycznej wersji algorytmu, jak i algorytmu wykorzystującego logikę rozmytą czy też probabilistykę. Omawiane w tych przypadkach zagadnienie stabilności bierze pod uwagę następujące parametry: (A) współczynnik stabilności algorytmu A na zbiorze ω, (B) $D(\delta)$, zbiór danych zaburzonych, (C) δ, zbiór danych.

Na podstawie obserwacji i badań w pracach, o których mowa powyżej zaproponowano wzór (2.11) pozwalający określić stabilność algorytmu:

$$S(A,\Omega) = \frac{1}{|\Omega|}\sum_{\delta\in\Omega}(A,\delta,D(\delta)) \tag{2.11}$$

Zgodnie ze stwierdzeniami z prac [24] i [25] wielkość skali współczynnika (2.11) jest skorelowana ze stabilnością i jak zostało zauważone, niewielkie zmiany tej wartości wpływają w nieznacznym stopniu na zmiany funkcji celu.

2.10. Metody reprezentacji problemu

Metody reprezentacji problemu warunkują wybór narzędzi do prowadzenia analizy, algorytmu rozwiązywania lub obszaru poszukiwania rozwiązania. Jako podstawowe metody opisu można wymienić [158]: (A) diagram przepływu danych; (B) synchroniczny diagram przepływu danych; (C) graf zadań; (D) sieć Petri; (E) SystemC. Najczęściej używaną metodą specyfikacji (również w tej pracy) są metody grafowe. Graf definiujemy jako parę $G = (V, E)$ (czasami oznaczane TG), gdzie V jest zbiorem węzłów, zaś $E \subseteq 2^V$ jest zbiorem nieskierowanych krawędzi. Krawędź jest reprezentowana zbiorem dwuelementowym $\{i, j\}$, $i, j \in V$. Graf skierowany jest definiowany analogicznie, tzn. $G = (V, A)$, gdzie $A \subseteq V \times V$ jest zbiorem (skierowanych) łuków, [58]. W grafie skierowanym źródłem jest węzeł, który nie posiada poprzedników. Węzeł niezawierający następników jest nazywany ujściem. Stopień węzła i, oznaczamy jako $deg(i)$ i jest on liczbą krawędzi będących w incydencji z danym węzłem, [58]. Stopień grafu definiujemy jako $deg(G) = \max_{i\in V} deg(i)$. Liczba krawędzi wejściowych oraz wyjściowych do węzła wraz z określeniem ich stopni to odpowiednio: $indeg(i)$ oraz

$outdeg(i)$. Ścieżkę (drogę) definiujemy jako sekwencję węzłów $(i_1, i_2, ..., i_n)$, gdzie $\{i_k, i_{k+1}\} \in E$ (odpowiednio $(i_k, i_{k+1}) \in A$). Cyklem jest ścieżka (droga) taka, że $i_1 = i_n$. Graf może mieć przypisane wagi do wierzchołków i krawędzi o znaczeniu zależnym od modelowanego problemu. Dla modelowania sieci operacyjnych będziemy używać reprezentacji grafowej AoN, w której czynność jest reprezentowana węzłem, z obciążeniem równym czasowi trwania tej czynności.

W literaturze do specyfikacji poprzedzania zadań najczęściej stosowane są acykliczne grafy skierowane DAG (*Directed Acyclic Graph*) [19], znane także pod nazwą TGFF oraz STG, [48, 118]. W pracy [48] zaprezentowany został algorytm do generowania grafów specyfikacji systemów z ograniczeniami stanowiący podstawę dla konstrukcji optymalnych algorytmów szeregowania zadań. W rozprawie do generacji grafów specyfikacji systemu używany będzie zmodyfikowany algorytm TGFF. Modyfikacja polega na uwzględnieniu zadań wieloprocesorowych. Zostało to omówione dalej w rozprawie oraz w publikacjach własnych [51, 52].

2.11. Algorytmy rozwiązywania

Analiza i projektowanie algorytmów rozwiązywania są zwykle prowadzone łącznie w celu uzyskania kompromisu pomiędzy jakością dostarczanego rozwiązania a czasem działania algorytmu. Uwzględnienie złożoności problemów i algorytmów rozróżnia metody klasyfikowane tradycyjnie jako: (A) dedykowane wielomianowe dla problemów P-klasy, (B) dokładne, wykładnicze, dla problemów silnie NP-trudnych, znajdują zastosowanie przy niewielkich rozmiarach rozpatrywanego problemu, (C) przybliżone, dla problemów silnie NP-trudnych, wyznaczają rozwiązanie “bliskie” rozwiązaniu optymalnemu.

Algorytmy dokładne przeszukują pośrednio lub bezpośrednio całą przestrzeń rozwiązań w celu wyznaczenia rozwiązania optymalnego. Odbywa się to poprzez generowanie rozwiązań częściowych, z których następnie jest konstruowane rozwiązanie końcowe. Użycie algorytmów dokładnych gwarantuje wyznaczenie optimum w dość długim, ale skończonym czasie. Złożoność obliczeniowa pozwala na dokonanie podziału na następujące algorytmy: (A) dedykowane; (B) wykorzystujące schemat B&B; (C) wykorzystujące schemat PD; (D) wykorzystujące metody

MILP; (E) wykorzystujące metody subgradientowe.

Dla metod dokładnych problemów NP-trudnych zarówno czas jak i ilość obliczeń znacząco wzrasta wraz ze wzrostem rozmiaru instancji. Alternatywą są algorytmy przybliżone, wśród których można wymienić [92, 154, 160]: (A) heurystyczne; (B) metaheurystyczne, (C) schematy aproksymacyjne. Algorytmy przybliżone są zwykle problemowo-zorientowane. Błąd przybliżenia względem optimum można ocenić a posteriori na drodze analitycznej lub eksperymentalnej. Definicja błędu może być dowolna, ale sensowna: sensowność definicji, adekwatność oceny własności, możliwość zachowania się błędu dla różnych przykładów. W tym celu błąd musi być poddany analizie, która może być rozpatrywany w dwóch wariantach: analitycznie lub eksperymentalnie [154].

Dwa pierwsze rodzaje algorytmów przybliżonych rekomendowane są gdy nie są znane wystarczająco szybkie algorytmy dokładne. W klasie rozwiązań metaheurystycznych są przeszukiwane wybrane punkty wraz z ich otoczeniem. W każdym kroku w tego typu algorytmach dopuszcza się rozwiązanie zarówno lepsze jak i gorsze od rozwiązania aktualnego, co pozwala na wyznaczenie globalnych optimów [160]. W przypadku heurystycznych algorytmów przeszukiwana jest zawężona przestrzeń projektowa. Docelowe rozwiązania są wyznaczane poprzez konstruowanie go z rozwiązań optymalnych wyznaczanych przy wykorzystaniu wybranych metryk [160]. Wybór metody rozwiązywania w klasie algorytmów przybliżonych zależy od wielu czynników: (A) rodzaju obszaru decyzyjnego (operacyjny, taktyczny, strategiczny); (B) częstotliwość podejmowania decyzji; (C) osiągalności czasu wykonania; (D) analityczne kwalifikacje decydenta; (E) rozmiar problemu szeregowania; (F) obecność elementów stochastycznych.

Problemy szeregowania związane są z klasą kombinatorycznych problemów optymalizacyjnych. Rozwiązania takich problemów zależą silnie od złożoności obliczeniowej danego zagadnienia. Warunkiem koniecznym do zastosowania tej klasy algorytmów w praktyce jest określenie górnego ograniczenia wielomianem niskiego stopnia. Jako heurystyczne można określić każde podejście niezapewniające formalnej gwarancji wykonania. Wśród heurystycznych algorytmów można wymienić priorytetowe, w których wszystkie schematy bazujące na częściowym uszeregowaniu podzielić można na reguły. Wspomniane reguły decydują na pod-

stawie określonych mechanizmów o priorytetach kolejnych operacji w zbiorze poddanym procesowi harmonogramowania [19, 133, 169]. Do najistotniejszych reguł zaliczają się [154]: (A) Najkrótszy czas wykonywania SPT (ang. *Shortest Processing Time*), (B) Najdłuższy czas wykonywania LPT (ang. *Longest Processing Time*), (C) Największa wielkość pozostałej pracy MWR (ang. *Most Work Remaining*), (D) Najmniejsza wielkość pozostałej pracy LWR (ang. *Least Work Remaining*), (E) Pozostała większość operacji MOR (ang. *Most Operation Remaining*), (F) Najmniejsze pozostałe operacje LOR (ang. *Least Operations Remaining*), (G) Najwcześniejszy czas wykonania ERT (ang. *Earliest Ready Time*), (H) Najwcześniejszy termin EDD (ang. *Earliest Due Date*), (I) Pierwszy wchodzący/wychodzący FIFO (ang. *First Input First Output*), (J) Losowy RANDOM (ang. *Random*).

Algorytmy heurystyczne, należą do klasy aproksymacyjnej pozwalającej na znajdowanie przybliżonych rozwiązań problemów optymalizacyjnych. Zazwyczaj ten rodzaj algorytmów stosowany jest w przypadku braku rozwiązań dokładnych dla rozpatrywanego problemu. Jako przykład tej klasy algorytmów wymienić można także algorytm zachłanny, który wybierając kolejne rozwiązanie, bierze pod uwagę najkorzystniejsze rozwiązanie z aktualnie rozpatrywanego punktu. Efektem takich decyzji najczęściej jest wyznaczanie minimów lokalnych. Pomimo problemów z wyznaczaniem optimum globalnego, ta klasa algorytmów charakteryzuje się dużo większą szybkością w porównaniu do rozwiązań dokładnych [19, 133, 160, 169]. Algorytmy heurystyczne, zwane także konstruktywnymi różnią się od siebie: metrykami, ilością jednocześnie podejmowanych zmian lub sposobem wyjścia poza rozwiązania lokalnie optymalne. Do najpopularniejszych algorytmów zalicza się [160]: (A) Algorytm zachłanny; (B) Logika rozmyta; (C) Prawdopodobieństwo hipotez; (D) Klastrowanie; (E) Heurystyka GCLP; (F) sieci BBN.

Dziedzina metaheurystycznych algorytmów wyszukiwania charakteryzuje się długą historią adaptowania zainspirowanych naturą systemów. W klasyfikacji tej wymienia się algorytmy genetyczne, mrówkowe i podobne. Szczególnie w dwóch ostatnich dekadach nastąpił rozkwit tego typu rozwiązań, [29]. Praca [29] oraz [28] kataloguje znane w literaturze algorytmy metaheurystyczne uwzględniając zwierzęta, rośliny, drobnoustroje, zjawiska naturalne czy też nadprzyrodzone. Różnice

między poszczególnymi rozwiązaniami polegać mogą na, [160]: (A) metodzie generacji rozwiązań: deterministyczne, losowe; (B) sposobie przeszukiwania przestrzeni rozwiązań: zbiorowym lub pojedynczym; (C) określeniu liczby rozwiązań: populacja lub jednostkowe; (D) wyznaczeniu zakresu przeszukiwań w rozwiązaniach sąsiednich: bliskie lub dalekie.

2.12. Inne modele niepewności

Niepewność w systemie wbudowanym może dotyczyć parametrów innych niż czas wykonania zadania, choć w literaturze brak jest danych na ten temat. Omówimy to w odniesieniu do przypadku niepewnych danych o żądaniach zasobowych zadania[5]. Rozpatrzmy bardziej szczegółowo przypadek wykonania pewnego zadania synchronicznie na dwóch identycznych procesorach tak, że uzyskane wyniki końcowe obliczeń są sprzeczne. W celu rozstrzygnięcia wątpliwości poprawności obliczeń można (patrz ilustracja w rys. 4.1): (a) powtórzyć zadanie jako trzyprocesorowe (głosuje większość); (b) wykonać identyczne "dodatkowe" zadanie jednoprocesorowe, rozstrzygające wątpliwości. Można też przyjąć inną interpretację. Zadanie ma przypisaną pewną miarę niepewności (np. prawdopodobieństwo) różną dla różnych żądań zasobowych. Przykładowo, zadanie ma "tryb" jednoprocesorowy, dwu-procesorowy, trzy-procesorowy, z różnym prawdopodobieństwem i różnym czasem wykonywania. Realizowany jest jeden z alternatywnych trybów. Miara niepewności ma wpływ na relację poprzedzania, czas wykonywania zadania, funkcję kryterialną.

[5] Żądania te są rozumiane jako liczba procesorów angażowanych do wykonania zadania.

Rozdział 3

Zastosowanie zadań wieloprocesorowych

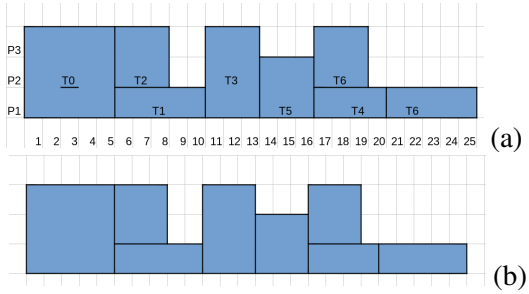
W tym rozdziale przedstawiona została motywacja oraz koncepcja wykorzystania zadań wieloprocesorowych. Pojawiają się one w kontekście niezawodności systemów wbudowanych (redundancja zadań i zasobów), jednoczesnego synchronicznego wykonywania równoległych operacji na wielu procesorach w określonej jednostce czasu, w optymalizacji procesu cięcia wstęgi.

3.1. Motywacja

Istnieje kilka powodów rozważania modeli wieloprocesorowych, niepewnych. W klasycznych deterministycznych problemach szeregowania typowo przyjmuje się, że zadanie jest realizowane przez pojedynczy procesor, nawet w przypadku gdy dysponujemy systemem wieloprocesorowym. W celu zwiększenia niezawodności systemu wprowadzana jest redundancja sprzętowa i programowa poprzez synchroniczne wykonywanie identycznych zadań na kilku identycznych procesorach pracujących równolegle, a następnie "głosowanie" w celu ustalenia decyzji. Typowym rozwiązaniem w tym zakresie są systemy zawierające 2 lub 3 procesory, dążąc do pewnej równowagi pomiędzy kosztem i poziomem niezawodności. Zauważmy, że zastosowanie 2 procesorów w przypadku sprzecznych wyników obliczeniowych nie pozwala na podjęcie jednoznacznej decyzji, pociągając za sobą

ponowne, lecz nieprzewidywalne wykonanie tego samego zadania na trzech procesorach (wówczas głosuje większość). Idea szeregowania zadań w architekturach wieloprocesorowych została bardziej szczegółowo dyskutowana w rozdziale 2. W dodatku A zamieszczono przegląd rzeczywistych złożonych architektur systemów wbudowanych, wykorzystujących zadania wieloprocesorowe. Podobne zastosowania to jednoczesny dostęp do pamięci współdzielonej, testowanie układów VLSI [15–17, 20, 30, 57, 59, 117, 171, 177] lub realizacja rzeczywistej równoległości [83, 136, 176]. Faktycznie, dla zwiększenia niezawodności, redundancji podlegają w pierwszej kolejności zadania krytyczne (znajdujących się na ścieżce krytycznej w grafie poprzedzań), których nieterminowe lub niepoprawne wykonanie może spowodować przekroczenie narzuconych ograniczeń czasowych.

Architektury systemów wbudowanych odnoszą się do niewielkiej liczby procesorów, co może wydawać się pewnym ograniczeniem problemu, np. $mpr \leq 3$. Ciekawszą interpretacją problemu $P|mpr|C_{\max}$ z $mpr \gg 3$ jest problem pakowania lub cięcia wstęgi (ang. *strip packing/cutting problem, SPP*). Rozpatrzmy dane do problemu $P|mpr|C_{\max}$ w formie: n, m, a_i, $i = 1, \ldots, n$, p_i, $i = 1, \ldots, n$. W problemie SPP, rozważmy wstęgę szerokości m i nieograniczonej długości. Na wstędze należy rozmieścić prostokątne elementy (bez możliwości ich rotacji) każdy o długości p_i i szerokości a_i, tak by długość zajętej taśmy była minimalna. Zauważmy, że problem SPP nie wymaga całkowitoliczbowych a_i, choć równoważność między SPP i $P|mpr|C_{\max}$ zachodzi tylko dla całkowitych a_i oraz m. Inaczej, problem $P|mpr|C_{\max}$ jest szczególnym przypadkiem SPP. Jest oczywiste, że długość harmonogramu tak skonstruowanego problemu jest równa długości zajętej wstęgi. Po raz pierwszy problem SPP został przedstawiony w pracy [85], gdzie także wykazano jego NP-trudność (faktycznie, NP-trudność wynika natychmiast z NP-trudności problemu szczególnego $P|mpr = 1|C_{\max}$). Do rozwiązywania problemu SPP stosuje się klasyczne algorytmy aproksymacyjne, metody heurystyczne czy też meta-heurystyczne, metody dokładne, a także inne algorytmy przybliżone [37, 106]. W klasie algorytmów przybliżonych dla problemu $P|mpr|C_{\max}$ obecnie najlepszy rezultat $(5/3 + \epsilon)$ osiągnięto w pracy [80]. Wg [113] można SPP transformować do Problemu Pakowania Pojemników (ang. *Bin Packing Problem, BPP*), gdzie problem pakowania otrzymujemy dla takiej instan-



Rysunek 3.1: Związek SPP z szeregowaniem zadań. (a) Uszeregowanie zadań 1..3-procesorowych, (b) Rozwiązanie SPP dla wstęgi szerokości 3.

cji SPP w której $h_i = 1$. Powołując się dalej na [113], jeżeli wyznaczona zostaje wartość optymalna dla 2SP (dwu wymiarowy SPP) to również otrzymamy wartość optymalną dla 1BP (jedno wymiarowy BPP). SPP jest ściśle związany z harmonogramowaniem, trasowaniem, czy też alokacją zasobów, jednocześnie mając powiązania z teorią rozbieżności, metodami iteracyjnymi, czy też zaokrągleniem entropii [37]. Najogólniejszą postacią jest dwuwymiarowy problem pakowania, gdzie dany jest zbiór prostokątnych elementów do zapakowania w minimalną liczbę kwadratowych pojemników 2D. Innym wariatem jest pakowanie wektorów o wymiarach d, gdzie każdy z wektorów musi zostać zapakowany do odpowiedniego pojemnika. Drugi wariant ma znaczenie w szeregowaniu z ograniczonymi zasobami. Obie wersje problemów rozważane mogą być jako offline-owe oraz online-owe [37]. Zauważmy, że wersja druga odpowiada rozpatrywanemu w pracy problemowi szeregowania zadań wieloprocesorowych. Związek pomiędzy SPP oraz rozpatrywanym w pracy szeregowaniem zadań wieloprocesorowych został zilustrowany na rysunku 3.1.

3.2. Obecny stan wiedzy

Jedną z pierwszych prac dotyczących użycia synchronicznego co najmniej dwóch identycznych procesorów do wykonania zadań w przypadku deterministycznym jest [15]. Zaprezentowano tam algorytmy o złożoności wielomianowej jednakże dla bardzo szczególnego przypadku jednostkowych czasów wykonania

zadań oraz liczby żądanych procesorów $1...a_i \leq k$, gdzie $k \leq m$ jest pewną ustaloną liczbą. Ponadto wykazano, że problem jest NP-trudny nawet dla wspomnianych jednostkowych czasów wykonywania i dowolnych a_i. Podano także algorytm wielomianowy dla przypadku z dowolnymi czasami wykonywania oraz żądaniami liczby procesorów 1 lub k. W szczególnym przypadku dokonano transformacji problemu optymalizacji do zadania programowania liniowego.

Kolejną pracą rozpatrującą problem szeregowania zadań wieloprocesorowych jest [57], gdzie analizowany był przypadek systemu cztero-procesorowego. Wykazano, że problem szeregowania bez wywłaszczeń jest NP-zupełny już dla $m = 2$. Następnie autorzy rozważają szeregowanie bez wywłaszczania określonego zadania na dowolnym procesorze ze zbioru elementów wykonawczych dla architektury złożonej z czterech procesorów. Zaproponowano algorytm, który rozwiązuje problem w czasie pseudowielomianowym.

Autorzy pracy [96] starając się uogólnić problem rozpatrywany w [15] z wykorzystaniem sieci neuronowych jedno oraz wielowarstwowych wyznaczają zbiór N niepodzielnych, niezależnych, wielowariantowych i wieloprocesorowych zadań na wielu identycznych i równoległych procesorach tak, aby zostały wykonane wszystkie zadania bez naruszenia ograniczeń czasowych, spełniając nałożone kryterium. Zakładają przy tym występowanie wielu wariantów jednego zadania, które wymagać może dowolnej liczby procesorów do wykonania z zakresu $1..M$.

Inną pracą rozpatrującą zadania wieloprocesorowe jest [16]. Opisano w niej pewne podejście odnosząc się do przykładu szeregowania zadań dwu-procesorowych w architekturze trzy-procesorowej. Zaproponowano podejście heurystyczne do szeregowania zadań bez wywłaszczania w ogólnym przypadku, w kategoriach analizy najgorszego przypadku, dla dedykowanych procesorów. Celem optymalizacji była minimalizacja długości uszeregowania.

Rozszerzenie przedstawionego powyżej podejścia, podkreślające wkład autorów w dziedzinie szeregowania zadań, zostały zaprezentowane w pracy [20], gdzie badane było szeregowanie zadań z dodatkowymi ograniczeniami. W pracy przyjęto występowanie skończonej liczby m procesorów oraz skończonej liczby n zadań. Założone zostało jak w powyższych przypadkach, że wybrane zadania mogą wymagać do realizacji więcej niż jednego procesora w danej jednostce czasowej.

Dodatkowo przyjęto, że każde zadanie może być podzielone na niezależne operacje $o_1, ..., o_{x_i}$, na które nałożone mogą być specyficzne ograniczenia. W pracy konstruowane jest optymalne uszeregowanie minimalizujące maksymalny termin zakończenia wszystkich zadań. Rozważono przykład realizacji na docelowej architekturze złożonej z trzech procesorów. W pracy tej wykazano również, że jeżeli wymagania zadań w łańcuchu były zgodnie lub odwrotnie monotoniczne, a liczba dostępnych procesorów była mniejsza niż dwukrotność maksymalnej ilości żądanych procesorów przez zadania, to przypadki mogą być rozwiązane w czasie wielomianowym $O(n \lg n)$.

Praca [117] rozważa problem redundancji zadań w kontekście wydajności i niezawodności. Zwiększenie poziomu niezawodności osiąga się poprzez replikację zadań, co ma wpływ na wydajność systemu. Wprowadzono matematyczną definicję współczynnika replikacji zadań. Zaproponowano podejście, którego celem jest maksymalizacja wskaźnika wydajności.

Z kolei w przeglądowej pracy [59] autor systematyzuje znane w literaturze modele i podejścia do szeregowania zadań wieloprocesorowych. Omawiane są przypadki użycia procesorów równoległych i procesorów dedykowanych. Przedstawione jest zestawienie wyników dla różnych funkcji celu i różnych struktur systemów procesorowych.

Autorzy pracy [171] rozważają m maszyn (M), gdzie na każdej z maszyn jest dostępna pewna liczba procesorów funkcjonujących równolegle. Na każdej maszynie M można realizować n zadań ($1..n$). Rozważane zostały jednostkowe czasy przetwarzania przy założeniu, że pojedyncze zadanie wymaga użycia od 1 lub k procesorów. W tak zdefiniowanym środowisku zaproponowano liniowy algorytm optymalny czasowo dla specjalnego przypadku rozpatrywanego problemu.

W pracy [111] rozpatruje się szeregowanie zadań wieloprocesorowych z funkcją celu minimalizacji średniego czasu przepływu zadań. Faktycznie minimalizowany jest sumaryczny czas zakończenia przetwarzania zadań, począwszy od terminów ich udostępnienia. Wykazano, że problem jest NP-trudny w przypadku ogólnym, natomiast przy założeniu niepodzielności zadań jest wielomianowy. Przedstawiono podejścia deterministyczne i online-list.

Problem obsługi dynamicznej redundancji na żądanie w obszarze krytycznych

systemów wbudowanych był rozpatrywany w pracy [30]. Projektowanie systemów tego typu jest ukierunkowane na zachowanie odpowiedniego poziomu niezawodności zwłaszcza w przypadku występowania przejściowych uszkodzeń. W pracy wygenerowano pewien zestaw zadań dla badania różnych strategii realizacji redundancji na żądanie. Przeprowadzone testy pokazały, że w rozpatrywanym przypadku jakość usług QoS (Quality of Service) została poprawiona średnio o 29% w stosunku do podejścia stosującego redundancję statyczną.

Kolejna praca [177] przedstawia zagadnienie niezawodności w przepływowym problemie szeregowania przy zmniejszonej ilości zasobów. Analizowana jest dolna granica poziomu niezawodności uszeregowania przy zarządzaniu zachłannym algorytmem minimalizacji redundancji. W pracy zaproponowano trzy algorytmy w celu spełnienia zadanych ograniczeń:

- algorytm RR w celu spełnienia wymogu niezawodności przy minimalnym zużyciu zasobów,
- algorytm DRR zapewniający spełnienie wymagań niezawodności i ograniczeń czasowych przy minimalnym zużyciu zasobów,
- algorytm dynamiczny dla długookresowego przewidywania awarii, zapewniający spełnienie ograniczeń niezawodnościowych oraz czasowych.

Przeprowadzone badania wykazują, że zaproponowane podejście znacznie zmniejsza poziom wykorzystania zasobów obliczeniowych oraz komunikacyjnych.

Innym argumentem uzasadniającym użycie zadań wieloprocesorowych jest tak zwana prawdziwa równoległość (ang. true parallelism), określająca konstrukcję, w której jednostki funkcjonalne są duplikowane i więcej niż jedna instrukcja programowa jest realizowana w każdej chwili czasowej [176]. W pracy wymienia się następujące rodzaje obliczeń synchronicznych równoległych: (A) podstawowy; (B) superskalarny; (C) wektor równoległy; (D) multiprocessor.

Prawdziwą równoległością zajmowano się także w pracy [83], w obszarze reprogramowalnej platformy sprzętowej. Zrównoleglenie łączone było z funkcjami FPGA, co poprawia zaprojektowany system w wielu aspektach, takich jak zwiększenie przepustowości systemu, minimalizacja kosztów, czy też zużycia energii przez system oraz złożoności systemu. Na podstawie przeprowadzonych badań stwierdzono, że zastosowanie koncepcji równoległości (true parallelism) pozwo-

liło na przetwarzanie modułom systemu wielu danych w danej jednostce czasowej w celu wykonania procesów obejmujących wiele wejść oraz wyjść. Wprowadzenie takiego rodzaju przetwarzania jednoczesnego pozwala na minimalizację złożoności docelowo tworzonego systemu. Dodatkowo zaproponowana w tym przypadku koncepcja pozwala na zintensyfikowanie przepustowości przy jednoczesnej minimalizacji zapotrzebowania na energię. Dzięki zastosowaniu prawdziwego zrównoleglenia w obrębie układów programowalnych istnieje możliwość lepszego wykorzystania modułów tworzonych w VHDL oraz układów logicznych. W pracy [136] rozpatrywane jest wspomniane powyżej tak zwane prawdziwe zrównoleglenie w pozycjonowaniu ramienia robota o pięciu stopniach swobody. W tym przypadku zaproponowano szybki sposób rozpoznawania gestów w systemie czasu rzeczywistego z wykorzystaniem strumienia wideo. Zaproponowano rozpoznawanie gestów dłoni w ograniczonym z góry ustalonym przez algorytm obszarze. W pracy tej zrównoleglenie polega na jednoczesnym sterowaniu siłownikami oraz rozpoznawaniu gestów dzięki zastosowaniu układów FPGA. Tak zaproponowane komplementarne podejście w efekcie pozwala na przyśpieszenie precyzyjnego pozycjonowania ramienia robota.

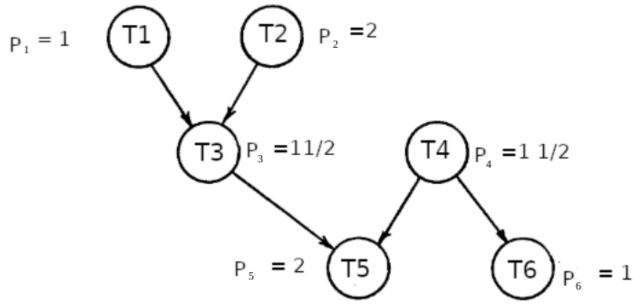
Kolejną pracą skupioną na procesie sterowania ramieniem robota jest [49], w której zastosowanie techniki rozwiązywania redundancji priorytetów, pozwala na eliminację osobliwości algorytmicznych. Zaproponowaną metodę stosuje się do manipulatora o siedmiu stopniach swobody, pozwalając na minimalizację osobliwości kinematycznych. Praca [7] dotycząca tematu prawdziwego zrównoleglenia w obszarze systemów wbudowanych wykorzystuje jednocześnie do działania zarówno układ FPGA oraz GPP (ang. *General Purpose Processor*).

3.3. Algorytm bazowy

We rozpoczniemy od opisu wielomianowego algorytmu Muntza-Coffmana, [120], który wyznacza rozwiązanie optymalne dla uszeregowania zadań w problemie $P2|mpr = 1, pmtn, prec|C_{max}$, oraz $P|mpr = 1, pmtn, in-tree|C_{\max}$. Przypadek out-tree jest symetryczny i może być rozwiązany poprzez szeregowanie zadań od końca (backward scheduling). Ten znany algorytm będzie stanowił

podstawę do stworzenia kilku kolejnych, nowych algorytmów szeregowania zadań wieloprocesorowych dla dowolnej liczby procesorów, tj. $P|mpr, prec, pmtn|C_{\max}$, w przypadku deterministycznym i wszystkich przypadkach niepewnych , patrz np. [51]. Wszystkie te nowe algorytmy będziemy nazywać w dalszej części rozprawy MC z odpowiednim górnym indeksem. Należy nadmienić, że klasa algorytmów MC nie wyczerpuje listy algorytmów proponowanych i analizowanych w tej rozprawie.

Przykładowy graf z danymi pokazano na rysunku 3.2. Działanie metody polega



Rysunek 3.2: Przykładowy graf zadań (na podstawie pracy [120]).

na wyznaczeniu priorytetów zadań h_j (nazywanych w dalszym ciągu *poziomem* lub *wysokością* zadania T_j). Wielkość h_j jest równa długości najdłuższej drogi w acyklicznym grafie poprzedzań $G = (T, E)$, $E \subset T \times T$, wychodzącej od zadania T_j (włączając jego czas wykonywania) i biegnącej do wszystkich następników tego zadania w G. Graf G może być reprezentowany alternatywnie poprzez zbiory poprzedników $B_j = \{T_i \in T : \ (T_i, T_j) \in E\}$, $j = 1, 2, \ldots, n$. Długość tak określonej drogi jest wyznaczana w G, w którym obciążenia węzłów T_i są równe p_i. Odpowiedni algorytm ma złożoność obliczeniową $O(n + |E|)$. Wielkości h_j są dynamiczne i będą podlegać zmianie w trakcie krokowo realizowanego procesu szeregowania. Podobnie, zbiór zadań T (dokładniej jego kopia) będzie modyfikowany poprzez usuwanie zadań już uszeregowanych (zakończonych). Proces szeregowania będzie kontrolowany poprzez redukcję (kopii) czasów wykonywania zadań p_j. Zadania, dla których $h_j = 0$ (także $p_j = 0$), są traktowane jako zakończone. Bez straty ogólności rozważań, możemy przyjąć indeksację zadań taką, że

$h_1 \geq \ldots \geq h_n$. Kolejnym parametrem związanym z procesem szeregowania jest *pojemność* β_j, która jest frakcją (częścią) całkowitego czasu obsługi zadania T_j zrealizowanego do chwili obecnej t. Początkowo $\beta_j = 0$, $T_j \in T$. Dla zwięzłości zapisu będziemy oznaczać $\beta = (\beta_1, \beta_2, \ldots \beta_n)$ oraz $h = (h_1, h_2, \ldots h_n)$.

Algorytm działa iteracyjnie, realizując kolejno następujące kroki, patrz także dla szczegółów [51]: (1) wyznacz poziomy zadań gotowych do wykonywania $Q \subset T$ (tzn. takich, które posiadają już wykonane poprzedniki), (2) przydziel zadania o najwyższym poziomie dowolnej maszynie; jeżeli liczba zadań jest większa niż liczba dostępnych procesorów, to każdemu zadaniu przypisz odpowiednio obliczoną jednostkę czasową procesora; jeżeli wszystkie zadania o najwyższym poziomie zostaną przypisane, wróć do kroku pierwszego, dopóki nie zostaną przeanalizowane wszystkie zadania, (3) zastosuj algorytmu McNaughton'a do uzyskania optymalnego uszeregowania.

Zgodnie z wyznaczonymi poziomami, zadania zostają przypisane w sposób przerywalny do jednego z dwóch procesorów, przy przyjętej jednostce upływu czasu. Algorytm wykonuje przedstawione powyżej operacje, dopóki nie zostaną rozpatrzone wszystkie zadania. Wymaga się aby czasy wykonania zadań stanowiły wielokrotność pewnej przyjętej elementarnej jednostki czasowej [120]. Opis oryginalnego algorytmu 3.3.1 przedstawimy w formie pseudokodu zamieszczonego poniżej, odnosząc się jednocześnie do tego algorytmu jako bazowego.

Algorytm 3.3.1. *Algorytm Muntza-Coffmana*

1 *t=0; for* $T_j \in T$ *oblicz* h_j*;*

2 *while* $T \neq \emptyset$ *do*

3 $Q = \{T_i \in T : \ (p_j = 0, \ j \in B_i) \wedge (p_i > 0)\}$ */* zbiór zadań gotowych do wykonywania */*

4 $\beta = calculate\_capability(Q)$; */* określ* β *dla zadań z* Q **/*

5 *oblicz czasy*

- $\tau' = \min(\infty, \min_{T_j, T_{j+1} \in Q}\{(\frac{h_j - h_{j+1}}{\beta_j - \beta_{j+1}}) : \ \beta_j \neq \beta_{j+1}, \ h_j > h_{j+1}\}$
- $\tau'' = \min_{T_j \in Q}\{\frac{p_j}{\beta_j} : \ \beta_j > 0\}$,

6 $\tau = \min(\tau', \tau'')$

7 *uszereguj kawałek* $\tau\beta_j$ *każdego zdania* $T_j \in Q$ *w przedziale* $[t, t + \tau]$ *zgodnie z algorytmem McNaughton'a; /*uszereguj zadania* $T_j \in Q$ *w okre-*

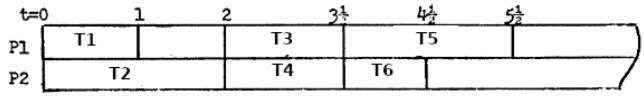
ślonym interwale czasowym $(t, t+\tau)$**/*
for $T_j \in Q$ *do*
$p_j = p_j - \tau \cdot \beta_j$;
$h_j = h_j - \tau \cdot \beta_j$;
$if(p_j = 0)$ *then*
$Q = Q - T_j$; $T = T - T_j$; */* usuń zadania zakończone*/*
endif;
end for;
$t = t + \tau$;
end while

Proces szeregowania zadań jest wspierany przez poniższą funkcję obliczania β, która stanowi integralną część algorytmu:

Algorytm 3.3.2. $function\ calculate\_capability(X)$

$\beta = 0;\ \ av = m;$ */* liczba dostępnych procesorów od chwili t */;*
while $av > 0$ *and* $X \neq \emptyset$ *do*
$M = \max_{T_j \in X} h_j$; */*wyznacz maksymalny poziom*/*
$H = \{T_j \in X : h_j = M\}$; */*zadania z poziomem M*/*
if $|H| \leq av$ *then*
for $T_j \in H$ *do* $\beta_j = 1$;
$av = av - |H|$;
end if
else
for $T_j \in H$ *do* $\beta_j = \frac{av}{|H|}$;
$av = 0$;
end if-else;
$X = X - H$;
end while
return β
end function

Na rysunku 3.3 zostało przedstawione przykładowe uszeregowanie otrzymane opisanym algorytmem dla instancji z rysunku 3.2. Opisany algorytm bazowy stał



Rysunek 3.3: Przykładowe uszeregowanie zbioru zadań wyspecyfikowanych w rysunku 3.2 (na podstawie pracy [120]).

się punktem wyjścia do konstrukcji kolejnych zmodyfikowanych wariantów (oznaczanych dalej jako MC, i chociaż jego nazwa jest taka sama jak w pracy [56] to algorytm ten opiera się na zupełnie innej koncepcji), patrz prace własne autora rozprawy [50–52, 54], oraz [56] dotyczące szeregowania zadań wieloprocesorowych z uwzględnieniem, oraz bez uwzględnienia atrybutu podzielności zadań, a także z uwzględnieniem czynnika niepewności. I tak przykładowo, w pracy [56] zaproponowano algorytm szeregowania zadań niezależnych, z wywłaszczaniem, dla nieznanej a priori liczby procesorów wymaganych do realizacji pojedynczego zadania, w tym także o żądaniach zmiennych w czasie. Tym samym autor wskazuje szerokie możliwości adaptacyjne podejścia MC. Badania te są rozszerzeniem koncepcji harmonogramowania zadań jednoprocesorowych na równoległych procesorach przy ograniczonej i zmiennej w czasie dostępności procesorów, patrz praca [18]. Zgodnie z obecnym stanem wiedzy, oprócz prac [15, 47, 56] algorytmy inne niż wymienione modyfikacje oryginalnego algorytmu Muntza-Coffmana nie były rozpatrywane.

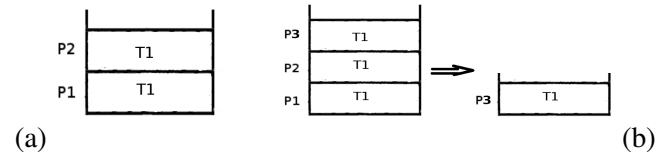
Rozdział 4

Niepewne szeregowanie zadań wieloprocesorowych

W tym rozdziale rozwiniemy koncepcję przedstawioną wcześniej w rozdziale 3. Rozpoczniemy od algorytmu, zwanego dalej MC, dla przypadku deterministycznego, wzorowanego na algorytmie bazowym opisanym w rozdziale 2. Kolejno przedstawimy różne modele z niepewnością pochodzące z publikacji, między innymi [51, 52] oraz innych własnych wymienionych w rozdziale 3. Przedstawimy również wyniki rozważań teoretycznych oraz eksperymentów komputerowych. Dodatkowo niepewność rozpatrzeć można, stosując opis innych parametrów z wykorzystaniem logiki wielowartościowej. Można do nich zaliczyć: (A) niepewny czas zakończenia; (B) niepewny czas przepływu; (C) niepewność w zakresie czasów relacji dla zadań zależnych; (D) niepewność w zakresie wykorzystywanych zasobów.

4.1. Użycie modelu deterministycznego

Rozpatrzmy bardziej szczegółowo realizację zależnych zadań wieloprocesorowych w kontekście niezawodności w oparciu o klasyczny model deterministyczny. Zadanie dwu-procesorowe można traktować jako wzajemną weryfikację wyniku synchronicznych obliczeń, tak jak to pokazane zostało na rysunku 4.1. Jeśli odpowiedzi są identyczne, to zadanie uważa się za poprawnie przetworzone, zaś

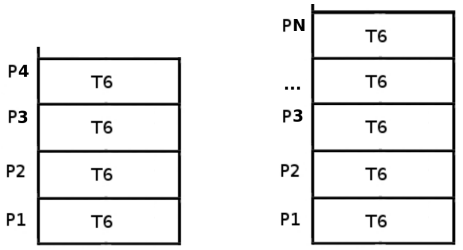


Rysunek 4.1: Propozycje zastosowania zadań wieloprocesorowych: a) dwu-procesorowych, b) trzy-procesorowych (możliwość wykonania jednego zadania na procesorze innym niż dotychczasowe).

wynik nie podlega wątpliwości. W przeciwnym przypadku, na skutek rozbieżnych wyników, potrzebny jest niezależny arbiter rozstrzygający o poprawności wyniku. Można to zrealizować trojako: (1) realizując a priori proces obliczeniowy jako zadanie trzy-procesorowe (głosuje większość), (2) realizując dodatkowe zadanie trzy-procesorowe uruchomione w trybie on-line na żądanie po negatywnym wyniku zadania dwu-procesorowego, (3) realizując w trybie on-line "dodatkowe" arbitrażowe zadanie jedno-procesorowe, uruchamiane na żądanie po negatywnym wyniku zadania dwu-procesorowego. Oczywiście warianty (2) i (3) są warunkowe jedynie w przypadku, gdy zadanie dwu-procesorowe nie dało jednoznacznego rozstrzygnięcia. Zatem najbardziej ogólna koncepcja zakłada równoczesne występowanie zadań jedno-, dwu- i trzy-procesorowych, przy czym zadania jedno-procesorowe mogą występować zawsze (nie jest wymagana redundancja obliczeń) lub na żądanie. Aby nie zwiększać niepotrzebnie rozmiarów całego systemu, redundancja jest realizowana tylko dla zadań krytycznych (znajdujących się na ścieżce krytycznej). Działania te skutkują ostatecznie zwiększeniem stopnia niezawodności systemu, co zostało wykazane m.in. pracy w [52]. Naturalnym rozwinięciem koncepcji wieloprocesorowości na zadania wymagające $a_i > 3$ procesorów są zaawansowane systemy robotyczne, równoległe kanały komunikacyjne, oraz tzw. prawdziwa równoległość, patrz np. rysunek 4.2.

Specyfikacji problemu zwykle towarzyszy graf poprzedzania zadań, patrz dla przykładu rysunek 4.3. Zastosowano tam konwencję AoN opisu zadań poprzez węzły grafu[1]. Dla obliczenia poziomu niezawodności zaproponowano wzór (4.1).

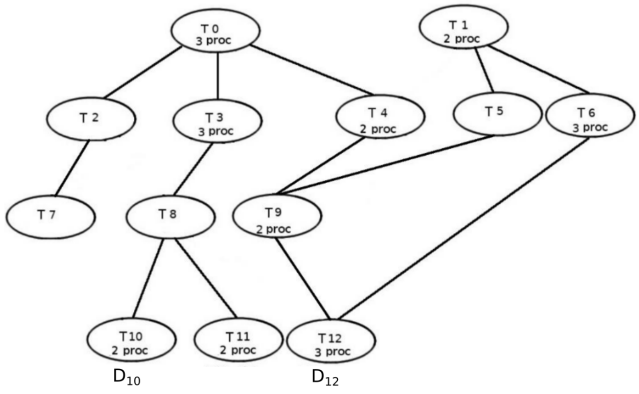
[1]Brak wpisu a_i oznacza $a_i = 1$



Rysunek 4.2: Przykład zadań cztero- oraz N-procesorowych

$$LoD = \frac{\sum_{i=1}^{n} w_i}{mC_{\max}}. \tag{4.1}$$

Wzór ten określa relację pomiędzy nakładem pracy na wykonanie wszystkich zadań (mianownik), a faktycznym czasem pracy systemu $mC_{\max}$. Zauważmy,żr jeslis zadania są niezleżne to warość współczynnika LoD zależy tylko od $C_{\max}$ Wartości współczynnika LoD oraz $C_{\max}$ dla przypadku wymieszanych zadań z $a_i \leq 3$, wykonywanych w środowisku $m = 3$ oraz $m = 4$ procesorów, podano w tabeli 4.1. Zaobserwować można wzrost współczynnika LoD skorelowanego z liczbą zadań wymagających więcej niż jednego procesora do ich wykonania, zwłaszcza z ilością zadań trzy-procesorowych.



Rysunek 4.3: Graf zadań, zawierający zadania 1-, 2- oraz 3-procesorowe.

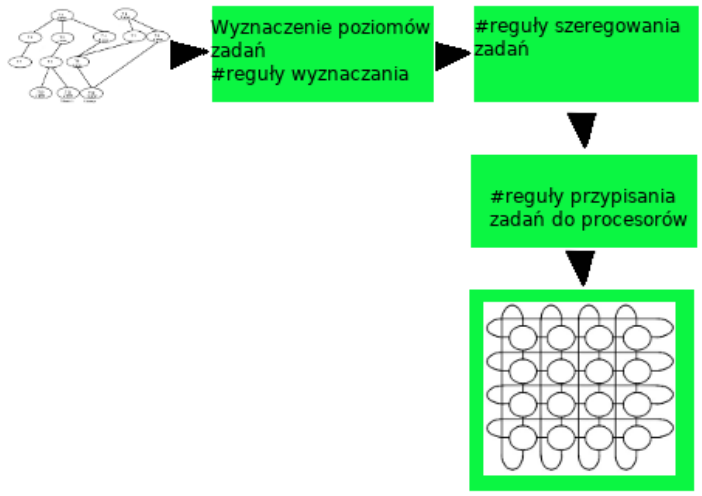
Instancja		a_i			$C_{\max}$		
nr	n	1	2	3	m=3	m=4	LoD
1	9	4	2	3	85	70	0,76
2	13	5	4	4	170	152	0,77
3	15	5	5	5	185	168	0,79
4	18	5	7	7	245	224	0,86
5	21	6	6	10	305	270	0,89
6	25	8	7	6	335	290	0,87
7	31	11	8	13	440	355	0,89
8	35	10	8	18	640	589	0,92
9	42	11	10	22	780	696	0,88
10	45	12	12	24	825	750	0,86

Tabela 4.1: Uszeregowanie deterministyczne, $C_{\max}$ oraz LoD dla $m = 3$ i $m = 4$

4.2. Algorytm MC

W pracy własnej [19] zaproponowany został algorytm MC nawiązujący w swojej idei do algorytmu Muntza-Coffmana omówionego w rozdziale 2. Istnieje kilka cech różniących istotnie MC od wspomnianego powyżej algorytmu. I tak nakład pracy dla realizacji zadania T_i jest charakteryzowane przez dwie wielkości p_i i a_i zamiast jak dotychczas jednej p_i (mającej poprzednio kluczowe znaczenie dla określenia poziomu zadania). Stąd odmiennie definiowany tutaj priorytet zadania określony dalej jako $w_i = p_i a_i$. Może on jednak osiągać jednakową wartość dla zadań o istotnie różnych a_i i p_i. Dla zachowania intuicyjnej zgodności nazw, w_i będzie nazywany dalej alternatywnie jako "priorytet" lub "poziom" zadania. Mamy świadomość, że potrzebne są dodatkowe wskaźniki preferencji przy wyborze kolejnego zadania do szeregowania. Zakładając, że graf poprzedzań może być dowolny (a nie tylko drzewem/lasem) konstrukcja priorytetu staje się bardziej skomplikowana i wymaga operowaniu na zbiorze zadań gotowych do wykonywania (ang. *schedulable*). Dalej, do symbolu h_i przypiszemy wielkość równą maksymalnej liczbie krawędzi w G prowadzących do T_i. Ostatecznie, Algorytm MC jest metodą przybliżoną dla ogólnego problemu szeregowania zadań wieloprocesorowych. Jako kryterium przyjęto minimalizację długości uszeregowania $C_{\max}$, przy jednoczesnym spełnieniu dodatkowych ograniczeń, takich jak np.: (a) D_i, czas ograniczenia, jeżeli występuje, to wszystkie ograniczenia zarówno dla ścieżki, jak i pojedynczego zadania zostaną zachowane przy wyznaczaniu docelowego uszeregowania, (b) NOC – w pełni zostaną wykorzystane zasoby elementów wykonawczych zorganizowane w architekturze Network On Chip.

Koncepcja algorytmu przedstawiona na rysunku 4.4 została dalej zapisana w formie pseudokodu, patrz algorytm 4.2.1, pozwalając na opis, analizę i implementację niezależną od języka programowania. Danymi do algorytmu są: T, n, m, $((p_i,\, a_i,\, D_i),\, T_i \in T)$ oraz acykliczny skierowany graf poprzedzania zadań $G = (T, E)$, $E \subset T \times T$. Dla potrzeb wyznaczania priorytetów będziemy posługiwać się trzema wariantami grafu G: (1) graf $G^{(w)}$, w którym obciążenie węzła T_i jest równe $a_i \cdot p_i$; (2) graf $G^{(p)}$, w którym obciążenie węzła T_i jest równe p_i; (3) graf $G^{(1)}$, w którym obciążenie węzła T_i jest równe 1. Oznaczmy odpowied-



Rysunek 4.4: Koncepcja algorytmu MC.

nio długość najdłuższej drogi do węzła T_i (wraz z obciążeniem tego węzła) przez w_i w grafie $G^{(w)}$, przez t_i w grafie $G^{(p)}$, oraz przez h_i w grafie $G^{(1)}$. W każdym z przypadków algorytm wyznaczania długości odpowiednich najdłuższych dróg ma złożoność obliczeniową $O(n+|E|)$. Algorytm wykorzystuje kopie niektórych wielkości z oryginalnego zestawu danych, mianowicie P (kopia T) oraz q_i (kopia p_i) dla śledzenia zmian.

Algorytm 4.2.1. *Algorytm MC:*

1. $t = 0$;
2. *for* $T_i \in T$ *do oblicz* w_i*;*
3. *for* $T_i \in T$ *do oblicz* t_i*;*
4. *for* $T_i \in T$ *do oblicz* h_i*;*
5. *if* $E \neq \emptyset$ *then*
6. *for* $T_i \in T$ *do* $w_i = w_i + t_i$*; /*korekta dla zadań zależnych*/*
7. *end if;*
8. *for* $T_i \in T$ *do if* $t_i > D_i$ *then stop: uszeregowanie nie istnieje;*
9. *P=T; /* kopia* T **/*
10. *for* $T_i \in T$ *do* $q_i = p_i$*; /* kopia* p_i **/*
11. *while* $P \neq \emptyset$ *do*

12. *av=m;*
13. $Q = \{T_i \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; */*zbiór zadań gotowych*/*
14. *if* $Q = \emptyset$ *then goto end-while;*
15. $M = \max_{T_j \in Q} w_j$*; /*maksymalny priorytet w* Q**/*
16. $W = \{T_j \in Q : \ w_j = M\}$; */*znajdź zadania z priorytetem* M**/*
17. $A = \max_{T_j \in W} a_j$*; /*maksymalne* a_j *w* W**/*
18. $F = \{T_j \in W; \ a_j = A\}$; */*zadania z* W *z żądaniem* A**/*
19. $C = \max_{T_j \in F} h_j$*; /*maksymalne* h_j *w* F**/*
20. $J = \{T_j \in F; \ h_j = C\}$; */*zadania z* F *o* $h_j = C$**/*
21. *wybierz dowolne* $T_i \in J$*;*
22. $x = 1$*;*
23. *if* $A \leq av$ *then*
24. *uszereguj* T_i *dla jednej jednostki czasowej* $[t, t+1]$*;*
25. $av = av - a_i$; $q_i = q_i - x$; $Q = Q - \{T_i\}$*;*
26. *end-if*
27. *else*
28. $Q = Q - \{T_i\}$*;*
29. *end if-else;*
30. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili* t**/*
31. *else*
32. $t = t + x$*;*
33. *end if-else;*
34. *if* $q_i = 0$ *then* $P = P - T_i$*;*
35. *end while;*

Dla opisanego algorytmu oraz dla grafu G z rysunku 4.3, w tabeli 4.2 zamieszczono kolejne kroki obliczeniowe. W celu skrócenia zapisu tabeli podano kroki w pewnej zagregowanej formie. Dla rozwiązania wyznaczano LoD zgodnie z (4.1). Zaproponowano także zmodyfikowaną definicję LoD, patrz (4.2), gdzie licznik jest sumą dla zadań o najwyższych priorytetach. Wyliczone współczynniki przedstawiono w tabeli 4.1, a także w tabelach 5.1, 5.2 (patrz rozdz. 5).

$$LoD = \frac{\sum_{i=1}^{n} w_{i,CT}}{mC_{\max}} \tag{4.2}$$

Nr. Zadania	Czas Zadania	Priorytet Zadania	Poziom zadania / czas do zakończenia zadania									
T0	30	195	195/20	195/5	0	0	0	0	0	0	0	0
T1	10	155	0	0	0	0	0	0	0	0	0	0
T2	5	30	30	30	30	30	0	0	0	0	0	0
T3	10	80	80	80	80	80	80	80	0	0	0	0
T4	15	105	0	0	0	0	0	0	0	0	0	0
T5	20	110	110	110/5	0	0	0	0	0	0	0	0
T6	25	135	135	135	135/10	135/5	0	0	0	0	0	0
T7	25	25	25	25	25	25	25	25	25	25/10	0	0
T8	10	50	50	50	50/5	0	0	0	0	0	0	0
T9	15	90	90	90	90	0	0	0	0	0	0	0
T10	20	40	40	40	40	40	40	40/15	40/5	0	0	0
T11	10	20	20	20	20	20	20	20	20	20	20	0
T12	20	60	60	60	60	60	60	60	60/15	0	0	0

Tabela 4.2: Przykładowa priorytetyzacja dla specyfikacji systemu w postaci grafu zadań 4.3

Analiza algorytmu 4.2.1 może być prowadzona w kilku kontekstach: (a) ocena złożoności obliczeniowej, (b) poprawność semantyczna. Na podstawie pseudokodu sformułowano Obserwacje 4.2.1, a następnie zaproponowano pewien formalizm matematyczny (4.3) opisujący algorytm.

Obserwacja 4.2.1. *Zaobserwujmy, że:*

1. *Kroki wstępne 1-8 algorytm wykonuje jednokrotnie, w czasie $O(|T| + |E|)$. Krok 8 jest wstępnym testem na istnienie rozwiązania.*
2. *Kroki 9-35 analizują poszczególne zadania $T_i \in P$ w kolejnych jednostkach czasowych $t = 0, 1, 2, \ldots, H$, zgodnie z ich dostępnością Q, gdzie $H = O(\sum_{i=1}^{n} p_i)$ jest oszacowaniem horyzontu czasowego harmonogramu.*
3. *Decyzja o alokacji zadań jest podejmowana dla każdej chwili t oraz zadań $T_i \in Q$, na podstawie priorytetu (w_i, t_i, h_i) oraz liczby dostępnych procesorów i obowiązuje przez jedną jednostkę czasową.*
4. *W każdej chwili czasowej t istnieje co najmniej jedno zadanie $T_i \in Q$ takie, że jedna jego jednostka zostanie przypisana do procesorów.*
5. *kolejne jednostki tego samego zadania T_i będą alokowane w kolejnych chwilach czasowych aż do uzyskania wartości $q_i = 0$.*

Model matematyczny algorytmu zapiszemy w następującej postaci (4.3), gdzie J, Q, H są wielkościami zdefiniowanymi dla algorytmu:

$$\begin{aligned} \forall\, T_i \in P\ \exists\ (w_i, t_i, h_i) \text{ oraz} \\ \forall\, T_i \in P\ (t_i \leq D_i) \text{ oraz} \\ \forall\, t \leq H\ J \subseteq Q \subseteq P \text{ oraz} \\ \forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } q_i = q_i - 1, \forall T_i \in I \text{ oraz} \\ \forall T_i \in \{T_j \in T : q_j = 0\}\ T_i \notin P \text{ oraz} \\ \forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } \sum_{T_i \in I} a_i \leq m \end{aligned} \tag{4.3}$$

Definicja 4.2.1. [39] *Algorytm A jest semantycznie poprawny względem warunków początkowego α i końcowego β, jeżeli dla każdych danych wejściowych spełniających warunek α obliczenie algorytmu A dochodzi do punktu końcowego oraz końcowe wartościowanie zmiennych spełnia warunek β.*

Twierdzenie 4.2.1. *Dla danych specyfikowanych przez α: acykliczny $G = (V, E)$, (a_i, p_i), $T_i \in T$, algorytm MC w skończonej liczbie kroków dochodzi do wyniku końcowego β: $P = \emptyset$, $(q_i = 0,\ T_i \in T)$.*

Dowód. Odpowiednio do definicji 4.2.1, specyfikujemy α jako: $G = (T, E)$ acykliczny, $T \neq \emptyset$, oraz (a_i, p_i), $T_i \in T$, $n = |T|$. Z kolei β specyfikujemy jako $P = \emptyset$, $(q_i = 0, T_i \in T)$. Faktycznie β oznacza, że zadania z T zostały poprawnie uszeregowane w całości. Dowód przeprowadzimy przez indukcję względem n.

"Krok podstawowy." Dla $n = 1$ mamy $T = \{T_1\}$ oraz musi być $E = \emptyset$. Wówczas mamy $P = Q = \{T_1\}$ oraz pomijając wielkości M, W, A, F, C jako oczywiste ze względu na $T = \{T_1\}$, otrzymujemy ostatecznie $J = \{T_1\}$. Ponieważ $a_1 \leq m$, zatem w chwili czasowej $t = 0$ uszeregujemy jedną jednostkę zadania T_1 po czym zbiór Q stanie się pusty. Zgodnie z algorytmem dalsze części zdania T_1 uszeregujemy w kolejnych chwilach czasowych począwszy od chwili czasu zero aż do osiągnięcia warunku $q_i = 0$ oraz $Q = \emptyset$. Zatem końcowe wartościowanie zmiennych wyjściowych osiągnie w tym przypadku β.

"Krok indukcyjny". Załóżmy, że algorytm MC osiąga β dla $G = (T, E)$ zawierającego $|T| = n$ zadań. Wykażemy, że MC jest w stanie osiągnąć β dla α określonego przez $G' = (T', E')$, zawierającego $|T'| = n + 1$ zadań, tj. $T' = T \cup \{T_{n+1}\}$, $E \subseteq E'$. Zadanie T_{n+1} zostanie wpisane do zbioru $P' = P \cup T_{n+1}$ oraz przypisana zostanie mu wartość $q_{n+1} = p_{n+1}$. Ponieważ G' jest acykliczny, zatem niech Q^* będzie pierwszym zbiorem zadań gotowych do wykonywania (mających wszystkie poprzedniki już wykonane) zawierającym zadanie T_{n+1}, oraz niech t' będzie pierwszą chwilą czasową, w której wyznaczono Q^*. Zatem począwszy od chwili t' zadanie T_{n+1} będzie występowało w zbiorze P' oraz w zbiorze zadań gotowych, aż do osiągnięcia $q_{n+1} = 0$, co spowoduje jego usunięcie z P'. Oznacza to, że osiągnięto β.

"Wnioskowanie indukcyjne". Wnioskowanie jest standardowe. □

Twierdzenie 4.2.2. *Algorytm 4.2.1 jest poprawny semantycznie.*

Mimo iż algorytm MC jest formalnie semantycznie poprawny, jego koncepcja jest istotnie odmienna od algorytmu Muntza-Coffmana. W MC priorytety są statyczne, wielowymiarowe, szeregowanie jest realizowane w pojedynczych jednost-

kach czasowych. Pociąga to za sobą pseudowielomianową złożoność obliczeniową MC równą $O(nH)$. Wprawdzie można zaproponować inną koncepcję algorytmu, bardziej bliską do w/w algorytmu, jednak w dalszym ciągu pozostanie on algorytmem przybliżonym. Alternatywne do MC algorytmy nie były dalej przedmiotem analizy.

W dalszym ciągu zajmiemy się analizą algorytmów dla klasy problemów $P|pmtn, mpr|C_{\max}$ dla zadań niezależnych tzn. $prec = \emptyset$. Wykorzystamy w tym celu znane rezultaty literaturowe.

Twierdzenie 4.2.3. *Istnieje rozwiązanie optymalne problemu* $P|pmtn, mpr = 1|C_{\max}$ *takie, że*

$$C^*_{\max} = \max\{\frac{1}{m}\sum_{j=1}^{n} p_j, \max_{j=1,..,n} p_j\} \tag{4.4}$$

Twierdzenie 4.2.3 jest powszechnie znanym rezultatem w teorii szeregowania. Algorytm wyznaczania harmonogramu pochodzi oryginalnie od McNaughtona [116], lecz w tej rozprawie proponuję inną własną oryginalną, bardziej wygodną wersję związaną z pewnym wariantem problemu pakowania pojemników (ang. *bin-packing problem*). Istotna różnica w porównaniu do klasycznego literaturowego problemu pakowania polega na użyciu skończonej liczby m pojemników o jednakowej wielkości $C^*_{\max}$. Proponowany algorytm składa się z następujących kroków: (1) wyznacz $C^*_{\max}$, (2) zapakuj m kontenerów, każdy o pojemności $C^*_{\max}$, w kolejności $1, 2, \ldots, m$, przedmiotami o wielkościach p_j dzieląc przedmioty przy pakowaniu pomiędzy pojemniki gdy załadowanie przekracza pojemność kontenera, (3) zbuduj harmonogram szczegółowy poprzez zaplanowanie wykonywania zadań na procesorze i zgodnie z kolejnością i wielkością elementów zapakowanych do pojemnika i. Zauważmy, że harmonogram ten jest realizowalny, bowiem w konsekwencji wzoru (4.4) każdy z przedmiotów spełnia warunek $p_j \leq C^*_{\max}$ oraz zostanie zapakowany całkowicie lub co najwyżej dwóch częściach niekolizyjnych wzajemnie w sensie terminów realizacji do dwóch kolejnych pojemników. Złożoność obliczeniowa tego algorytmu to $O(n)$. Istnieje także alternatywna wersja algorytmu, także o złożoności $O(n)$: (a) uszereguj zadania na jednym procesorze w dowolnej kolejności, (b) "odcinaj" kolejno fragmenty o długości $C^*_{\max}$ i przypisuj je do realizacji na kolejnych procesorach. Konsekwencją twierdzenia

4.2.3 jest następujący rezultat.

Twierdzenie 4.2.4. *Istnieje rozwiązanie optymalne problemu* $P|pmtn, mpr = 2|C_{\max}$ *takie, że*

$$C^*_{\max} = \begin{cases} \max\{\frac{2}{m}\sum_{j=1}^{n} p_j, \max_{j=1,..,n} p_j\} & m \text{ parzyste} \\ \max\{\frac{2}{m-1}\sum_{j=1}^{n} p_j, \max_{j=1,..,n} p_j\} & m \text{ nieparzyste} \end{cases} \tag{4.5}$$

Dowód. Ponieważ zadanie wykorzystuje procesory w sposób synchroniczny istnieje co najwyżej $[m/2]$ par procesorów równocześnie dostępnych do realizacji zadania. Bez straty ogólności rozważań załóżmy, że sa to pary $(1,2)$, $(2,3)$, $\ldots$,$(m-1,m)$ jeśli m jest parzyste oraz $(1,2)$, $(2,3)$, $\ldots$,$(m-2,m-1)$ jeśli m jest nieparzyste. Zauważmy, że w przypadku nieparzystego m nie ma możliwości użycia procesora m, ponieważ nie ma on pary. Dalej transformujemy problem $P|pmtn, mpr = 2|C_{\max}$ z danymi n, m, $(p_j,\ j = 1,2,\ldots n)$ do pomocniczego problemu $P|pmtn, mpr = 1|C_{\max}$ z danymi n', m', $(p'_j,\ j = 1,2,\ldots,n)$ w następujący sposób: $n' = n$, $m' = [m/2]$, $(p'_j = p_j,\ j = 1,2,\ldots,n)$. Zgodnie z twierdzeniem 4.2.3 zachodzi

$$C^*_{\max} = \max\{\frac{1}{m'}\sum_{j=1}^{n} p'_j, \max_{j=1,..,n} p'_j\}. \tag{4.6}$$

Ponieważ $m' = [m/2] = m/2$ dla m parzystego oraz $m' = [m/2] = (m-1)/2$ dla m nieparzystego, podstawiając m' do (4.6) otrzymujemy (4.5). □

Z powyższego dowodu wynika natychmiast koncepcja nowego algorytmu szeregowania zadań wieloprocesorowych, przerywalnych, niezależnych, o ustalonym $a_i = k,\ T_i \in T$. Danymi do algorytmu są T, a_i, p_i m oraz k.

Algorytm 4.2.2.

1. *skonstruuj problem pomocniczy z danymi* $p'_i = p_i$, $a'_i = 1$, $m' = [\frac{m}{k}]$,
2. *znajdź rozwiązanie dla problemu pomocniczego* $P|pmtn, mpr = 1|C_{\max}$,
3. *dokonaj transformacji odwrotnej.*

Korzystając z idei powyższego dowodu, możemy dalej wykazać przez analogię następujące twierdzenie. Niech $[x]$ oznacza część całkowitą liczby x.

Twierdzenie 4.2.5. *Istnieje rozwiązanie optymalne dla problemu* $P|pmtn, mpr = k|C_{\max}$ *takie, że:*

$$C^*_{\max} = \max\{\frac{1}{[m/k]}\sum_{j=1}^{n} p_j, \max_{j=1,..,n} p_j\}. \tag{4.7}$$

Zajmijmy się teraz klasą problemów szeregowania zadań niepodzielnych wieloprocesorowych. Najprostsza wersja tego problemu $P|mpr = 1|C_{\max}$ jest NP-trudna, [132]. W literaturze do wyznaczenia rozwiązania polecane są głównie algorytmy przybliżone, takie jak np. szeregowanie listowe według reguły LPT (ang. *Longest Processing Time)*.

Twierdzenie 4.2.6. *Współczynnik najgorszego przypadku przybliżonego algorytmu LPT dla problemu* $P|mpr = 1|C_{\max}$ *jest równy*

$$r = \frac{4}{3} - \frac{1}{3m} \tag{4.8}$$

Twierdzenie 4.2.6 jest powszechnie znanym wynikiem literaturowym, [74]. Korzystając z niego oraz z agregacji procesorów używanej w dowodzie twierdzenia 4.2.4 możemy wykazać następujące fakty.

Twierdzenie 4.2.7. *Współczynnik najgorszego przypadku przybliżonego algorytmu LPT dla problemu* $P|mpr = k|C_{\max}$ *jest równy*

$$r = \frac{4}{3} - \frac{k}{3m} \tag{4.9}$$

Problemy $P|mpr \leq k|C_{\max}$ oraz $P|mpr|C_{\max}$ wymagają prócz rozstrzygnięcia złożoności obliczeniowej także sformułowania pewnego algorytmu rozwiązywania. Oczywiście, oba wymienione problemy są NP-trudne, bowiem już $P|mpr = 1|C_{\max}$ jest NP-trudny. Niestety, próba rozszerzenia LPT na przypadek "mieszanych" zadań wieloprocesorowych jest kłopotliwa.

Podane twierdzenia zweryfikowano na przykładach poniżej.

Przykład 1

W przykładzie zakłada się wykorzystanie architektury *5-procesorowej* ($m = 5$), występowanie zadań tylko *3-procesorowych* ($k = 3$), oraz liczba zadań w grafie stanowiącym specyfikację wynosi 51. Wyznaczone na podstawie algorytmu $C_{\max} = 1504$. Wyliczone na podstawie wzoru ograniczenie wynosi:

$$r = \frac{4}{3} - \frac{3}{5 * 3} = \frac{20 - 3}{15} = \frac{17}{15} \approx 1,13 \tag{4.10}$$

Zatem zastosowanie ograniczenia górnego do zaproponowanego współczynnika daje rezultat:

$$r = 1,14 * 1504 \approx 1705 \tag{4.11}$$

Przykład 2

W przykładzie zakłada się wykorzystanie architektury *5-procesorowej* ($m = 3$), występowanie zadań tylko *3-procesorowych* ($k = 3$), oraz liczba zadań w grafie stanowiącym specyfikację wynosi 51. Wyznaczone na podstawie algorytmu $C_{\max} = 1504$. Wyliczone na podstawie wzoru ograniczenie wynosi:

$$r = \frac{4}{3} - \frac{3}{3 * 3} = 1 \tag{4.12}$$

Zatem zastosowanie ograniczenia górnego do zaproponowanego współczynnika daje rezultat:

$$r = 1 * 1504 = 1504 \tag{4.13}$$

Przykład 3

W przykładzie zakłada się wykorzystanie architektury *4-procesorowej* ($m = 4$), występowanie zadań tylko *3-procesorowych* ($k = 3$), oraz liczba zadań w grafie stanowiącym specyfikację wynosi 51. Wyznaczone na podstawie algorytmu $C_{\max} = 1504$. Wyliczone na podstawie wzoru ograniczenie wynosi:

$$r = \frac{4}{3} - \frac{3}{4 * 3} = \frac{16 - 1}{12} = \frac{15}{12} = 1,25 \tag{4.14}$$

Zatem zastosowanie ograniczenia górnego do zaproponowanego współczynnika daje rezultat:

$$r = 1,25 * 1504 \approx 1880 \tag{4.15}$$

Dla zaproponowanego algorytmu zostały przeprowadzone badania dla szeregowania zadań w architekturze 3-, 4- oraz 5-procesorowej. Czasy uszeregowań

przedstawione zostały w tabeli 5.1 w kolumnie o nazwie ToE. Z przedstawionego w tabeli 5.1 zestawienia wynika, że czas realizacji zadań jest zależny od liczby procesorów w architekturze NoC, zwiększając co prawda koszt całego systemu, ale istotnie wpływając na redukcję czasu. Dodatkowo zauważa się zależność wprost proporcjonalna w stosunku do liczby zadań wieloprocesorowych. Przykładowe uszeregowanie dla wszystkich rozpatrywanych architektur zostało zaprezentowane na rysunku 5.1. Dla rozpatrywanych problemów zaproponowano obliczanie niezawodności w dwóch wariantach. Pierwsza wersja opisana wzorem (4.1). Poziom niezawodności nie zależy od liczby elementów wykonawczych, ale skorelowany jest z liczbą zadań, biorąc pod uwagę ich priorytety. Oraz w drugiej wersji rozpatrywane są zadania określane jako krytyczne w stosunku do ich priorytetów. W niniejszym podejściu zaproponowano losowe wyznaczanie zadań w specyfikacji systemu, które rozumiane mogą być jako krytyczne, co pozwala na uniknięcie tendencyjności obliczeń. Zestawienie poziomów niezawodności zadań zarówno dla pierwszej jak i drugiej wersji zostało przedstawione w tabelach, odpowiednio dla wariantu pierwszego w tabeli 5.1 oraz wariantu drugiego w tabeli 5.2. Oczywiście analogicznie do logiki rozmytej również w przypadku podejść z zastosowaniem niepewności jako czasów realizacji zadań co zostało zobrazowane w odpowiednich tabelach zaprezentowanych w rozdziale 5. Wersja pierwsza poziomu niezawodności jest opisana wzorem (4.1), dla wersji drugiej zaproponowano następujący wzór (4.2). Pierwsza metoda rozpatruje ogół wszystkich zadań oraz ich priorytetów, natomiast w drugim przypadku brane są pod uwagę tylko zadania krytyczne oraz ich priorytety. Z definicji zadania takie powinny zostać alokowane w pierwszej kolejności, a zatem zaproponowano, aby rozpatrywane były zadania oraz priorytety alokowane w procesorze o minimalnym indeksie. Podobną koncepcję niezawodności rozważać można również w przypadku zadań n-procesorowych.

4.3. Podejście z logiką rozmytą

Przedstawione w podrozdziale wcześniejszym podejście zostało rozszerzone w kierunku modelu uwzględniającego niepewność poprzez “rozmytość danych i zmiennych decyzyjnych”. Biorąc pod uwagę przegląd stanu wiedzy oraz kon-

cepcji omówionych w rozdziale 2, zaproponowano tutaj specyfikację czasów wykonywania zadań jako parametru niepewnego dla dwóch odmiennych podejść, odpowiedni dla zmiennych lingwistycznych (A) oraz dla zmiennych ciągłych (B). Dla celów rozpatrywania logiki rozmytej wprowadzone zostały nowe oznaczenia dedykowane dla wartości rozmytych, mianowicie; $A_i^F = \mu(T_i^F)$ - rezultat funkcji przynależności dla i-tego zadania, gdzie $A_i^F = (i, \mu_A(i); i \in X$, oraz $\mu_A : X \to [0, 1]$.

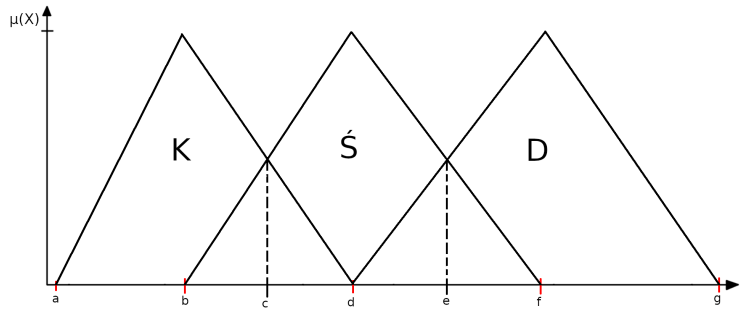
Podejście A

Podejście to zakłada rozmyte czasy wykonywania (trwania) zadań. Do opisu wielkości rozmytych zastosowano zmienne lingwistyczne z trójkątną funkcją przynależności o wzorze (4.17). Zakładamy klasyfikację w odniesieniu do czasu wykonywania (trwania) zadania, odpowiednio: A) krótkie, B) średnie, C) długie. Do opisania zadań wykorzystywana jest również funkcja przedstawiona wzorem (4.17). Na podstawie tak wyspecyfikowanych danych obliczane są priorytety zadań. Oczywiście, priorytety są także liczbami rozmytymi. Do wyznaczania priorytetów wykorzystywany jest prezentowany w pracy zmodyfikowany algorytm MC. Wyniki (długość harmonogramu) na podstawie takiego podejścia został zaprezentowany w tabeli 5.7. Przeprowadzono eksperyment porównawczy, podejścia A z zaproponowanym dalej (opisanym jako podejście B). Okazuje się, że co do rezultatu (długości uszeregowania oraz sekwencji zadań na zadanej architekturze) w obydwu przypadkach otrzymywane są identyczne rezultaty. Zatem zakłada się, że rozwiązania te są takie same, stąd opisane poniżej zastosowanie algorytmu 4.3.2.

Dla takiego podejścia przeprowadzono eksperymenty, z których wynika, że sekwencja zadań w harmonogramie nie ulega zmianie, niezależnie od wartości funkcji przynależności. Eksperymenty zostały zrealizowane dla trójkątnej funkcji przynależności. Otrzymane wyniki ukazują, że można zastosować inne podejście, określone mianem podejście B (opisane w rozdziale poniżej).

Poniżej dostarczono pseudokod algorytmu 4.3.1 wariant A prezentujący jego sposób implementacji oraz działania.

Algorytm 4.3.1. *Algorytm MC- Fuzzy (A):*



Rysunek 4.5: Zmienne lingwistyczne opisujące czasy trwania zadań

1. $t = 0$;
2. *LV =* $linguistic\_vars(T)$*; T = LV; /* zapisanie zdekodowanych zmiennych lingwistycznych w zbiorze LV (ang. Linguistic Variables)*/*
3. *for* $T_i \in T$ *do oblicz* w_i*;*
4. *for* $T_i \in T$ *do oblicz* t_i*;*
5. *for* $T_i \in T$ *do oblicz* h_i*;*
6. *if* $E \neq \emptyset$ *then*
7. *for* $T_i \in T$ *do* $w_i = w_i + t_i$*; /*korekta dla zadań zależnych*/*
8. *end if;*
9. *for* $T_i \in T$ *do if* $t_i > D_i$ *then stop: uszeregowanie nie istnieje;*
10. *P=T; /* kopia* T **/*
11. *for* $T_i \in T$ *do* $q_i = p_i$*; /* kopia* p_i **/*
12. *while* $P \neq \emptyset$ *do*
13. *av=m;*
14. $Q = \{T_i \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; */*zbiór zadań gotowych*/*
15. *if* $Q = \emptyset$ *then goto end-while;*
16. $M = \max_{T_j \in Q} w_j$*; /*maksymalny priorytet w Q*/*
17. $W = \{T_j \in Q : \ w_j = M\}$; */*znajdź zadania z priorytetem* M **/*
18. $A = \max_{T_j \in W} a_j$*; /*maksymalne* a_j *w* W **/*
19. $F = \{T_j \in W; \ a_j = A\}$; */*zadania z* W *z żądaniem* A **/*
20. $C = \max_{T_j \in F} h_j$*; /*maksymalne* h_j *w* F **/*

21. $J = \{T_j \in F;\ h_j = C\}$; /*zadania z F o $h_j = C$*/
22. $T^F = fusification(T)$ /*funkcja wykonująca fuzyfikację zgodnie z opisanymi regułami trójkątnej funkcji przynależności*/
23. *wybierz dowolne* $T_i^F \in J$;
24. $x = 1$;
25. *if* $A \leq av$ *then*
26. *uszereguj* T_i^F *dla jednej jednostki czasowej* $[t, t+1]$;
27. $av = av - a_i$; $q_i = q_i - x$; $Q = Q - \{T_i^F\}$;
28. *end-if*
29. *else*
30. $Q = Q - \{T_i^F\}$;
31. *end if-else;*
32. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili t*/*
33. *else*
34. $t = t + x$;
35. *end if-else;*
36. *if* $q_i = 0$ *then* $P = P - T_i^F$;
37. *end while;*

Syntetyczny opis algorytmu jest następujący:

- Po wczytaniu danych (rozmytych) są wyznaczane zakresy zmiennych lingwistycznych określających czasy trwania zadań. Wyróżnia się następujące trzy rodzaje zmiennych lingwistycznych określających długość czasu trwania zadania (co zostało pokazane na rysunku 4.5): krótkie (K), średnie (Ś), długie (D). Konsekwencją zastosowania rozmytych czasów są również rozmyte priorytety zadań, które również określa się w 3 grupach jako: (A) mały, (B) średnie, (C) duży.
- Zmienne lingwistyczne są określane poprzez wzięcie pod uwagę zakres czasów zadań, pomiędzy najkrótszym i najdłuższym czasem trwania zadania. Następnie, dzieląc ten zakres na 4 równe części, brane są pod uwagę najkrótsze i najdłuższe czasy zadań w każdej z grup. Na tej podstawie wyznaczane są zakresy poszczególnych zmiennych. Zatem poprzez podział taki można określić 5 wartości, (odnoszących się procentowo do czasów trwania za-

dań) wśród wszystkich zadań są to: $a - 0\%$, $b - 25\%$, $c - 50\%$, $d - 75\%$, $e - 100\%$. Pierwszy zakres, zadania krótkie (opisane funkcją trójkątną przynależności) mieści się w przedziale od $[a \dots c)$, przy wartości $b\mu(X) = 1$. Zakres drugi określający zadania średnie mieści się w przedziale od $[b \dots d)$, przy wartości $c\mu(X) = 1$. Zakres trzeci określający zadania długie mieści się w przedziale $[c \dots e]$, przy wartości $d\mu(X) = 1$. Stosując tę regułę, określa się przynależność zadań do poszczególnej grupy. Analogiczne podejście ma zastosowanie przy wyznaczaniu priorytetów. Zostało to zobrazowane na rysunku 4.5.

- Na podstawie wyznaczonych priorytetów następuje szeregowanie zadań według zasady przyjętej w algorytmie MC, gdzie jako pierwsze są wybierane zadania o priorytetach z grupy "wysokie", sposród których wybiera się wartości maksymalne.
- Jeżeli istnieją/istnieje procesor/procesory niezajęte w danej jednostce czasu to zostają wybrane możliwe do wykonania zadania z określonej grupy.

Obserwacja 4.3.1. *Korzystajac z przyjętych oznaczeń, zaproponowano następujący model matematyczny algorytmu 4.3.1:*

$$\begin{aligned}
\forall\, T_i^F \in P\ \exists\ (w_i, t_i, h_i) \text{ oraz} \\
\forall\, T_i^F \in P\ (t_i \leq D_i) \text{ oraz} \\
\forall\, t \leq H\ J \subseteq Q \subseteq P \text{ oraz} \\
\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } q_i = q_i - 1, \forall T_i^F \in I \text{ oraz} \\
\forall T_i^F \in \{T_j^F \in T^F : q_j = 0\}\ T_i^F \notin P \text{ oraz} \\
\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } \sum_{T_i^F \in I} a_i \leq m
\end{aligned} \tag{4.16}$$

Na podstawie obserwacji 4.3.1 oraz modelu matematycznego (4.16) można sformułować twierdzenie 4.3.1.

Twierdzenie 4.3.1. *Dla danych specyfikowanych przez α: acykliczny $G = (V, E)$, (a_i, p_i), $T_i^F \in T^F$, algorytm MC - fuzzy (podejście A) w skończonej liczbie kroków dochodzi do wyniku końcowego β: $P = \emptyset$, $(q_i = 0,\ T_i^F \in T^F)$.*

Dowód tego twierdzenia jest analogiczny do dowodu twierdzenia 4.2.1, zatem zostanie pominięty.

Podejście B

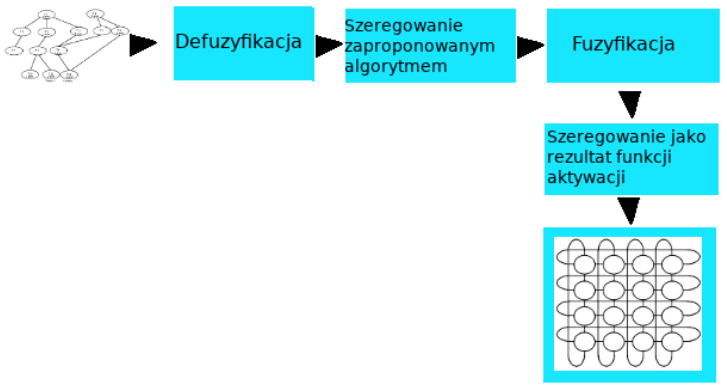
W rozpatrywanym podejściu zakłada się specyfikację zadań z wykorzystaniem trójkątnej funkcji przynależności, przy założeniu, że wartość minimalna tej funkcji jest o 20% mniejsza niż wartość średnia zaś maksimum jest o 20% większa od wartości średniej. Opisano to wzorem (4.17), strona lewa. To podejście określono dalej nazwą Model A (w podejściu B)

$$T_i^F = \begin{cases} a = b - 20\% \\ b = \mu(T_i^F) = 1 \\ c = b + 20\% \end{cases} \qquad T_i^{F*} = \begin{cases} a = b - 10\% \\ b = \mu(T_i^F) = 1 \\ c = b + 30\% \end{cases} \tag{4.17}$$

Rozważono również zastosowanie trójkątnej funkcji przynależności w innym wariancie, patrz strona prawa wzoru (4.17). Założono tam, że minimalna wartość funkcji przynależności jest o 10% mniejsza niż wartość średnia dla $\mu = 1$ oraz maksymalna wartość jest o 30% większa od wartości średniej. Ten przypadek będzie występował dalej pod nazwą Model B (w podejściu B). Ta modyfikacja została wprowadzona dla oceny jak zmiana rozmytości wpływa na docelową długość uszeregowania, patrz T_{ALL}^F. Zostało to przedstawione w tabelach zawierających zestawienie wyników uszeregowania w rozdziale 5.

Działanie algorytmu MC-fuzzy (B) opiera się na zasadach podobnych, jak w przypadku algorytmu MC. Wczytywana są dane (p_i, a_i, $T_i \in T$), $n = |T|$, m, oraz graf zadań $G = (T, E)$, $E \subset T \times T$. Odmienność tego podejścia od podejścia polega na sposobie reprezentacji czasów w grafie zadań z wykorzystaniem metod logiki rozmytej. Po wczytaniu ścieżki w grafie zadań wyznaczane są poziomy (priorytety) dla każdego z zadań. Dla wyznaczenia priorytetu rozmyte dane zostają poddane defuzyfikacji, wykorzystując metodę środka maksimum. Mając docelowo wartości z zakresu podejścia deterministycznego, możliwe jest zaadaptowanie priorytetyzacji z tego przypadku. Po określeniu poziomów dla

poszczególnych zadań algorytm ponownie wykonuje fuzyfikację. Kolejnym krokiem jest wykonanie alokacji zadań o najwyższych priorytetach zgodnie z przedstawioną powyżej procedurą szeregowania dla logiki rozmytej. Docelowo otrzymane uszeregowanie jest przedstawiane z użyciem wykresu Gantta. Ze względu na logikę rozmytą i postać funkcji przynależności zachodzi konieczność przedstawienia uszeregowania na trzech wykresach Gantta: odpowiednio dla wartości skrajnych oraz wartości średniej długości uszeregowania.



Rysunek 4.6: Reprezentacja graficzna zaproponowanego podejścia z wykorzystaniem logiki rozmytej

Na rysunku 4.6 przedstawiono koncepcję algorytmu szeregowania z wykorzystaniem logiki rozmytej. Modyfikacja w tym przypadku, w stosunku to tradycyjnego algorytmu, polega na zastosowaniu bloków defuzyfikacji oraz fuzyfikacji. Jako dane wejściowe przyjmowany jest graf zadań zawierający specyfikację czasów zadań w formie wybranej funkcji przynależności. Następnie określone dane zostają przekazane jako wejście do algorytmu szeregowania zadań. Przeprowadzone operacje szeregowania zostają poddane następne procesowi fuzyfikacji, aby możliwe było określenie ekstremów funkcji przynależności. Zgodnie z zaproponowaną zasadą proponowany jest harmonogram zadań dla różnych wartości, skorelowanych z wartościami funkcji przynależności. Dla zaproponowanego algorytmu przeprowadzono badania w takim samym zakresie, jak w przypadku algorytmu ze

specyfikacją z zakresu podejścia deterministycznego.

Poniżej zaprezentowano pseudokod algorytmu 4.3.2 przedstawionego na rysunku 4.6 prezentujący sposób implementacji oraz działania.

Algorytm 4.3.2. *Algorytm MC - fuzzy (Podejście B):*

1. $t = 0$;
2. $T = defuzz(T^F)$; /* użycie metody środka maksimum/
3. *for* $T_i \in T$ *do oblicz* w_i;
4. *for* $T_i \in T$ *do oblicz* t_i;
5. *for* $T_i \in T$ *do oblicz* h_i;
6. *if* $E \neq \emptyset$ *then*
7. *for* $T_i \in T$ *do* $w_i = w_i + t_i$; /*korekta dla zadań zależnych*/
8. *end if;*
9. *for* $T_i \in T$ *do if* $t_i > D_i$ *then stop: uszeregowanie nie istnieje;*
10. P=T; /* kopia T */
11. *for* $T_i \in T$ *do* $q_i = p_i$; /* kopia p_i */
12. *while* $P \neq \emptyset$ *do*
13. av=m;
14. $Q = \{T_i \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; /*zbiór zadań gotowych*/
15. *if* $Q = \emptyset$ *then goto end-while;*
16. $M = \max_{T_j \in Q} w_j$; /*maksymalny priorytet w Q*/
17. $W = \{T_j \in Q : \ w_j = M\}$; /*znajdź zadania z priorytetem M*/
18. $A = \max_{T_j \in W} a_j$; /*maksymalne a_j w W*/
19. $F = \{T_j \in W; \ a_j = A\}$; /*zadania z W z żądaniem A*/
20. $C = \max_{T_j \in F} h_j$; /*maksymalne h_j w F*/
21. $J = \{T_j \in F; \ h_j = C\}$; /*zadania z F o $h_j = C$*/
22. *wybierz dowolne* $T_i \in J$;
23. $T^F = fusification(T)$;
24. $x = 1$;
25. *if* $A \leq av$ *then*
26. *uszereguj* T_i *dla jednej jednostki czasowej* $[t, t+1]$;
27. $av = av - a_i$; $q_i = q_i - x$; $Q = Q - \{T_i^F\}$;
28. *end-if*

29. *else*
30. $Q = Q - \{T_i^F\}$;
31. *end if-else;*
32. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili t*/*
33. *else*
34. $t = t + x$;
35. *end if-else;*
36. *if* $q_i = 0$ *then* $P = P - T_i^F$;
37. *end while;*

Komentarz do pseudokodu jest następujący:

- Jako specyfikacja systemu przyjmowany jest graf, z których zadaniami czasy opisane są z wykorzystaniem logiki rozmytej, następnie poddawane są one procesowi defuzyfikacji. W celu defuzyfikacji wykorzystana została metoda środka maksimum. Zatem z zaproponwanych rozmytych czasów trwania zadań jest pobierana wartość maksymalna funkcji przynależności.
- Z wyznaczonych priorytetów wybierane są zadania o najwyższych wartościach, uwzględniając położenie każdego z zadań na ścieżce/ścieżkach.
- Wybierane są zadania z określonymi powyżej priorytetami i następnie zadania te poddawane są fuzyfikacji. Po procesie rozmywania zadania zostają alokowane w wybranych procesorach dla wybranych wartości funkcji rozmytej (MIN, MED, MAX), tak aby (o ile to możliwe) nie występował/ nie występowały niewykorzystywane (będące bezczynnymi) procesor/procesory w danej jednostce czasu.
- Jeżeli w jednostce czasu nie występują bezczynne procesory, lub nie ma możliwości przypisania żadnego zadania do nieużywanych procesorów/procesora to czas zostanie przesunięty o kolejną jednostkę, i algorytm wykonuje się ponownie, dopóki nie zostaną uszeregowane wszystkie zadania.

Dzięki zaproponowaniu podejścia do szeregowania zadań w takiej formie istnieje możliwość opisania złożoności algorytmu. Do złożoności algorytmu oczywiście brany jest pod uwagę przebieg programu. Natomiast przyjmując oznaczenia dla przebiegu ogólnego z oznaczeniem α zakłada się, że:

- α_0 - wczytanie specyfikacji w postaci grafu zadań,

- α_1 - wyznaczenie ścieżek w grafie, o ile istnieją,
- α_2 - Szeregowanie zadań w docelowej architekturze NoC.

Zakładając oznaczenie β dla algorytmu rozpatrującego zastosowanie logiki rozmytej wymienić można:

- β_0 - wczytanie specyfikacji w postaci grafu zadań,
- β_1 - blok defuzyfikacji,
- β_2 - Proces wyznaczania priorytetów według przebiegu ogólnego algorymu,
- β_3 - proces fuzyfikacji,
- β_4 - alokacja zadań w docelowej architekturze sieci NoC.

Dodatkowo opisany został algorytm w sposób szczegółowy z wykorzystaniem symbolu λ, gdzie przedstawić można:

- λ_1 - przypisanie priorytetów według założonych reguł,
- λ_2 - przypisanie zadań do wybranego procesora,
- λ_3 - zasymulowanie realizacji zadań w procesorach.

Obserwacja 4.3.2. *Złożoność obliczeniowa algorytmu 4.3.2 jest określona wzorem (4.18).*

$$Z = \beta_0 + \beta_1 + \lambda_1 + \sum_{i=1}^{N}(\lambda_2 + \lambda_3) + \beta_3 + \beta_4 \tag{4.18}$$

Na podstawie powyższej obserwacji 4.3.2, biorąc pod uwagę różnice pomiędzy algorytmem deterministycznym oraz dla logiki wielowartościowej można zaproponować model matematyczny adaptujący logikę rozmytą w następującej postaci:

$$\begin{aligned}
\forall\, T_i^F \in P\ \exists\ (w_i, t_i, h_i) \text{ oraz} \\
\forall\, T_i^F \in P\ (t_i \leq D_i) \text{ oraz} \\
\forall\, t \leq H\ J \subseteq Q \subseteq P \text{ oraz} \\
\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } q_i = q_i - 1, \forall T_i^F \in I \text{ oraz} \\
\forall T_i^F \in \{T_j^F \in T^F : q_j = 0\}\ T_i^F \notin P \text{ oraz} \\
\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } \sum_{T_i^F \in I} a_i \leq m
\end{aligned} \tag{4.19}$$

Na podstawie obserwacji 4.3.2 oraz modelu matematycznego (4.19) można sformułować twierdzenie 4.3.2, analogiczne do twierdzenia 4.2.1. Formalna różnica pomiędzy tymi twierdzeniami polega na zmianie sposobu reprezentacji danych wejściowych, zatem kolejny dowód można przeprowadzić "przez analogię".

Twierdzenie 4.3.2. *Dla danych specyfikowanych przez α: acykliczny $G = (V, E)$, (a_i, p_i), $T_i^{F'} \in T^F$, algorytm MC - fuzzy (podejście B) w skończonej liczbie kroków dochodzi do wyniku końcowego β: $P = \emptyset$, $(q_i = 0,\ T_i^{F'} \in T^F)$.*

4.4. Podejście stochastyczne

Kolejnym aspektem niepewności jest zastosowanie koncepcji stochastycznej. W wyniku przeglądu obecnego stanu wiedzy (Rozdział 2), zaproponowano model opisujący czasy zadań z wykorzystanie podejścia stochastycznego. Wykorzystany w tym przypadku opis, stosując zasadę σ pozwala na wyznaczenie uszeregowania docelowego na założonym poziomie prawdopodobieństwa.

Stosując się podczas szeregowania zadań do reguły σ istnieje możliwość realizacji zadań na określonym poziomie prawdopodobieństwa według zasady: (A) $\overline{x}$ - dla 40%, (B) 2σ - dla uszeregowania na poziomie $68,3\%$, (C) 4σ - dla uszeregowania na poziomie $95,5\%$, (D) 6σ - dla uszeregowania na poziomie $99,7\%$.

W stosunku do wcześniejszych podejść do szeregowania zaproponowano opracowanie harmonogramu z pewnym założonym prawdopodobieństwem, zarówno pod względem czasu realizacji systemu, jak i poziomu niezawodności. Przyjęto wektor $V = (v1, v2, v3, v4)$ oznaczający zakres czasu uszeregowania, zgodnie z zaprezentowaną powyżej regułą trzech σ. Zaproponowany w tym przypadku algorytm został przedstawiony poniżej w postaci pseudokodu 4.4.1:

Algorytm 4.4.1. *Algorytm MC - stochastic:*

1. $t = 0$;
2. $V = decode\_probability(T)$ */* Oznaczenie rozkładu normalnego prawdopodobieństwa dla specyfikacji systemu, gdzie $V(0) = 40\%$ */*
3. *for $V_i \in V$ do oblicz T;*
4. *for $V_i \in V$ do*

5. *for* $T_i \in T$ *do oblicz* w_i*;*
6. *for* $T_i \in T$ *do oblicz* t_i*;*
7. *for* $T_i \in T$ *do oblicz* h_i*;*
8. *if* $E \neq \emptyset$ *then*
9. *for* $T_i \in T$ *do* $w_i = w_i + t_i$*; /*korekta dla zadań zależnych*/*
10. *end if;*
11. *for* $T_i \in T$ *do if* $t_i > D_i$ *then stop: uszeregowanie nie istnieje;*
12. *P=T; /* kopia* T **/*
13. *for* $T_i \in T$ *do* $q_i = p_i$*; /* kopia* p_i **/*
14. *while* $P \neq \emptyset$ *do*
15. *av=m;*
16. $Q = \{T_i \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; */*zbiór zadań gotowych*/*
17. *if* $Q = \emptyset$ *then goto end-while;*
18. $M = \max_{T_j \in Q} w_j$*; /*maksymalny priorytet w* Q**/*
19. $W = \{T_j \in Q : \ w_j = M\}$; */*znajdź zadania z priorytetem* M**/*
20. $A = \max_{T_j \in W} a_j$*; /*maksymalne* a_j *w* W**/*
21. $F = \{T_j \in W; \ a_j = A\}$; */*zadania z* W *z żądaniem* A**/*
22. $C = \max_{T_j \in F} h_j$*; /*maksymalne* h_j *w* F**/*
23. $J = \{T_j \in F; \ h_j = C\}$; */*zadania z* F *o* $h_j = C$**/*
24. *wybierz dowolne* $T_i \in J$*;*
25. $x = 1$*;*
26. *if* $A \leq av$ *then*
27. *uszereguj* T_i *dla jednej jednostki czasowej* $[t, t+1]$*;*
28. $av = av - a_i$*;* $q_i = q_i - x$; $Q = Q - \{T_i\}$*;*
29. *end-if*
30. *else*
31. $Q = Q - \{T_i\}$*;*
32. *end if-else;*
33. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili* t**/*
34. *else*
35. $t = t + x$*;*

36. end if-else;
37. if $q_i = 0$ *then* $P = P - T_i$*;*
38. end while;
39. end for;

Króki opis pseudokodu jest następujący:

- Jako specyfikacja systemu przyjmowany jest graf zadań, gdzie czasy trwania zadań zostały opisane z wykorzystaniem rozkładu prawdopodobieństwa.
- Następuje określenie wartości zmiennych dla zastosowanego rozkładu prawdopodobieństwa, zgodnie z jego właściwościami. Dla rozkładu normalnego zastosowano opis z wykorzystaniem reguły trzech σ. Rozpoczynając od wartości średniej, dla której prawdopodbieństwo wynosi 40%. Następnie w kolejnych krokach są realizowane obliczenia dla pozostałych wartości określonych przez wektor V.
- Z wyznaczonych priorytetów wybierane są zadania o najwyższych wartościach, uwzględniając położenie każdego z zadań na ścieżce/ścieżkach.
- Wybierane są zadania z określonymi powyżej priorytetami i następnie zadania te zostają alokowane w wybranych procesorach, tak aby (o ile to możliwe) nie występowały niewykorzystywane (będące bezczynnymi) procesor/procesory w danej jednostce czasu.
- Jeżeli w jednostce czasu nie występują bezczynne procesory, lub nie ma możliwości przypisania żadnego zadania do nieużywanych procesorów/procesora to czas zostanie przesunięty o kolejną jednostkę, i algorytm wykonuje się ponownie, dopóki nie zostaną uszeregowane wszystkie zadania.
- Jeżeli uszeregowano wszystkie zadania w założonym poziomie prawdopodobieństwa, następuje wybranie kolejne wartości. Po tym algorytm rozpoczyna priorytetyzację ponownie inicjalizując swoje działanie.

Z powyższego opisu algorytmu wynika także różnica, pomiędzy tradycyjnym podejściem a wykorzystaniem niepewności w szeregowaniu. Zarówno w przypadku zastosowania logiki rozmytej, jak i probabilistyki modyfikacja polega na dodaniu elementów rozpatrujących niepewność. W przypadku probabilistyki jest to albo wariancja dla uszeregowania na poziomie 40%, albo odpowiedni zakres uszeregowania w zależności od wartości odchylenia standardowego.

Działanie programu opiera się na zasadach podobnych jak w przypadku logiki rozmytej. Zatem również wczytywana jest specyfikacja, obejmując opis grafu zadań oparty metodę *TGFF* ze specyfikacją zadań wieloprocesorowych. Różnica w tym przypadku polega na sposobie reprezentacji czasów w grafie zadań z wykorzystaniem metod prawdopodobieństwa, w tym konkretnym przypadku rozkładu normalnego. Po wczytaniu specyfikacji systemu zostają wyznaczane poziomy dla każdego z zadań. Poziomy zadań zostają wyznaczone na podstawie wartości średniej stanowiącej prawdopodobieństwa na poziomie 40%. Mając docelowo wartości deterministyczne, możliwe jest zaadaptowanie priorytetyzacji tego podejścia, analogicznie jak w podejściu z wykorzystaniem logiki rozmytej. Kolejnym krokiem jest wykonanie alokacji zadań o najwyższych priorytetach zgodnie z przedstawioną powyżej procedurą szeregowania dla każdego z czterech poziomów zakresu prawdopodobieństwa. Zaprezentowanie wyników na wykresie Gantta jest w tym przypadku bardzo utrudnione, zatem zaproponowano pewne wykresy obrazujące rezultaty szeregowania w rozdziale 5. Reprezentacja szeregowania obejmuje wartość średnia oraz wartości skrajne w postaci poszczególnych odchyleń standardowych. Konieczne jest przedstawienie rezultatu na czterech poziomach, stanowiących reprezentację uszeregowania dla problemu rozpatrującego liczby specyfikowane z wykorzystaniem rozkładu normalnego prawdopodobieństwa.

Biorąc pod uwagę opis algorytmu w niniejszym podrozdziale 4.4 oraz obserwacje 4.2.1, oraz 4.3.2 a także modele dla podejścia deterministycznego, oraz logiki rozmytej można zaproponować następujący opis algorytmu dla podejść wykorzystujących koncepcję stochastyczną:

$$\begin{aligned}
\forall\, T_i^S \in P\ \exists\, (w_i, t_i, h_i) \text{ oraz} \\
\forall\, T_i^S \in P\ (t_i \leq D_i) \text{ oraz} \\
\forall\, t \leq H\ J \subseteq Q \subseteq P \text{ oraz} \\
\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } q_i = q_i - 1, \forall T_i^S \in I \text{ oraz} \\
\forall T_i^S \in \{T_j^S \in T^S : q_j = 0\}\ T_i^S \notin P \text{ oraz} \\
\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } \sum_{T_i^S \in I} a_i \leq m
\end{aligned} \tag{4.20}$$

Z podejścia takiego wynika, że dla uszeregowania o podwyższonej wartości prawdopodobieństwa realizacji danego zadania, a co za tym idzie także całego systemu, wykonywane jest uszeregowanie o danym zakresie. Zgodnie z regułą σ dla wyższej niż 40% wartości prawdopodobieństwa, należy wziąć pod uwagę odpowiedni zakres czasu trwania zadania, co przekłada się także na długość uszeregowania w określonym zakresie prawdopodobieństwa. Poniżej przedstawiony został przykład dla czasu uszeregowania zadań w systemie z określonym czasem całkowitym $T^S_{ALL} = 230$: (A) $v1$ - 230; (B) $v2$ - $224 - 236$; (C) $v3$ - $218 - 242$; (D) $v4$ - $212 - 248$.

Podobnie jak w omówionych wcześniej przypadkach szeregowania, również tutaj wyznaczany jest współczynnik niezawodności. W tym przypadku jednak wzór na wyznaczanie ulega zmianie poprzez wprowadzenie współczynnika skorelowanego z prawdopodobieństwem. Zgodnie z zaproponowanym wzorem poziomy niezawodności również zostały wyznaczone w zależności od rozpatrywanego elementu zbioru V. Niezawodność jest w przypadku tego typu uszeregowania dana wzorem:

$$P_p(D_x, \rho), \tag{4.21}$$

gdzie

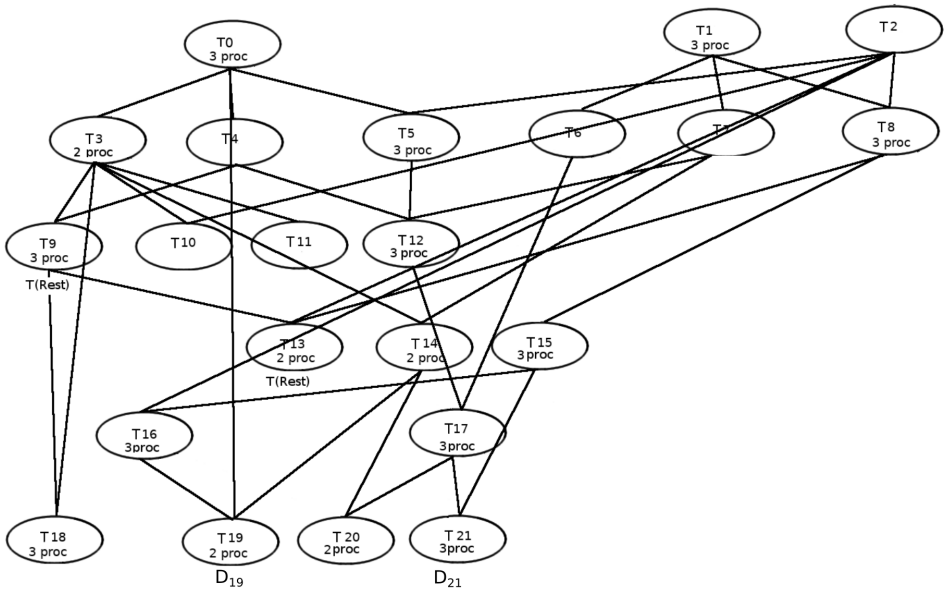
$\rho -$ odchylenie standardowe

$$P_p(\frac{\sum_{i=x}^{n}(w_i)}{\sum_{i=x}^{n}(p_i)}, \rho) \tag{4.22}$$

W omawianym podejściu zastosowano oznaczenia w postaci indeksu górnego S do wszystkich symboli związanych z szeregowaniem. Dodatkowo w każdym z rozpatrywanych przypadków zadania realizowane są na wybranym poziomie prawdopodobieństwa w zależności od wybranego elementu zbioru V.

Twierdzenie 4.4.1. *Dla danych specyfikowanych przez α: acykliczny $G = (V, E)$, (a_i, p_i), $T^S_i \in T^S$, algorytm MC - stochastic w skończonej liczbie kroków dochodzi do wyniku końcowego β: $P = \emptyset$, $(q_i = 0,\ T^S_i \in T^S)$.*

Jak można zauważyć twierdzenie 4.4.1 jest niemal identyczne z twierdzeniem 4.2.1. Różnica polega w tym przypadku na uwzględnieniu zmiennych niepewnych, gdzie uwzględniana jest logika rozmyta. Zatem formalna różnica polega



Rysunek 4.7: Graf zadań

tylko i wyłącznie na zmianie sposobu reprezentacji danych wejściowych, poprzez analogię można zastosować dowód dla podejścia deterministycznego.

Dla ilustracji podejścia, w tabeli 4.3 zaprezentowano sposób obliczania priorytetów oraz konstrukcję uszeregowania dla zadań opisanych grafem z rys. 4.7. Graf zawiera 22 zadania, gdzie czasy wykonywania zadań są zmiennymi losowymi. Rozkład długości uszeregowania (w wykresach użyto określenia “czas uszeregowania”) dla prawdopodobieństwa na poziomie 40% przedstawiono w rysunkach 5.13, 5.14 oraz 5.15 (patrz rozdział 5.3). Długość uszeregowania dla prawdopodobieństwa v1 (40%) wynosi 205 i dla wartości pozostałych: (A) $v2$ - $195 - 215$; (B) $v3$ - $185 - 225$; (C) $v4$ - $175 - 235$.

Nr.	Poziom zadania / czas do zakończenia																								
T0	10	210	210	210	210	210	210	210	210	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T1	30	415	415 (20)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T2	10	190	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T3	10	95	95	95	95	95	95	95	95	95	95	95	95	95	95	0	0	0	0	0	0	0	0	0	0
T4	15	150	150	150	150	150	150	150	150	150	150	150	150 (10)	150 (10)	0	0	0	0	0	0	0	0	0	0	0
T5	15	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180	180 (10)	0 0	0	0	0	0	0
T6	5	110	110	110	110	110	110	110	110	110	110	110	0	0	0	0	0	0	0	0	0	0	0	0	0
T7	20	155	155	155	155	155 (15)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T8	25	325	325	325	325	325 (20)	325 (5)	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T9	10	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70 (5)	0	0	0
T10	25	25	25	25	25	25	25	25	25	25	25	25	25	25	25 (15)	25 (15)	25 (15)	25 (15)	25 (15)	25 (5)	25 (5)	25 (5)	0	0	0
T11	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	0	0	0	0	0	0
T12	15	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	95	0	0	0	0	0	0	0	0
T13	20	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	0
T14	15	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70	70 (5)	0	0	0	0	0	0	0
T15	20	250	250	250	250	250	250	250	250	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
T16	50	190	190	190	190	190	190	190	190	190	190	190	190 (45)	190 (45)	190 (35)	190 (25)	190 (10)	0	0	0	0	0	0	0	0
T17	15	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	105	0	0	0	0	0
T18	5	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15	15
T19	20	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40	40 (5)	0	0	0	0
T20	10	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20
T21	20	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	60	0	0

Tabela 4.3: Przykładowa priorytetyzacja dla specyfikacji systemu w postaci grafu zadań 4.7

Podobnie jak w innych przypadkach tutaj również do reprezentacji użyty został diagram Gantta, który w sposób przystępny i zrozumiały obrazuje proces uszeregowania. Oczywiście dla logiki wartościowej jest generowany ostatecznie tylko jeden harmonogram, natomiast dla metod używających jako specyfikacji parametrów nieprecyzyjnych (logika rozmyta, prawdopodobieństwo) wyznaczana jest większa liczba uszeregowań w zależności od stosowanej metody opisu zadań. W przypadkach takich diagram Gantta okazać może się niewystarczającym narzędziem dla porównania ostatecznych całościowych wyników, trzeba raczej zastosować porównanie kilku zaprezentowanych diagramów, co lepiej uwidacznia się poprzez zestawienie czasu całkowitego uszeregowania. Wszystkie przeprowadzone badania zostały wykonane na podstawie zaprezentowanych w niniejszej pracy metodyk oraz wykorzystując zaproponowane algorytmy. Proces przeprowadzania badań oraz ich rezultaty, jak i zestawienie wyników przedstawiono w rozdziale 5.

4.5. Szeregowanie online

Rozpatrywane w tym przypadku koncepcja niepewności jest szersza niż w innych przypadkach, zależy to oczywiście od klasy rozpatrywanych problemów zaprezentowanych w podrozdziale 2.6 oraz uwzględniając wyniki pracy [13], mogą to być odpowiednio: (A) nieokreślone są czasy pojawienia się zadania; (B) nieokreślone są czasy zakończenia/czasy trwania zadania; (C) nieznane są czasy (długości) transmisji pomiędzy zadaniami; (D) nieznane są czasy przepływu; (E) nieznane są czasy pojawienia się awarii maszyn; (F) nieznane są czasy przestoju maszyn. Zaprezentowane podejście do szeregowania zadań wieloprocesorowych zostało zmodyfikowane oraz dostosowane do koncepcji wieloprocesorowego szeregowania online. W nawiązaniu do obecnego stanu wiedzy oraz klasyfikacji tego rodzaju podejścia wprowadza się dodatkowe oznaczenia. Stosując rozszerzenie notacji Grahama, rozpatrywany problem szeregowania wieloprocesorowych zadań zapisać można jako: $P|prec, online - list, mpr|C_{\max}$. Specyficzne sygnatury dla problemu szeregowania online:

- r_i - czas udostępnienia zadania, jeżeli został zdefiniowany, to realizacja zadania nie może zostać rozpoczęta wcześniej niż zdefiniowana wartość,

- c_i^O - czas zakończenia zadania i,
- $f_i^O = c_i^O - s_i^O$, czas przepływu, gdzie s_i^O jest czasem wykonania zadania,

Przypadek szeregowania zadań online-owych rozpatrywać może zarówno zadania zależne jak i niezależne. Realizując harmonogramowanie dla zadań zależnych, należy uwzględniać nie tylko określone relacje, ale przede wszystkim czas pojawienia się zadania w systemie. Zaprezentowane w powyższych rozdziałach podejścia obejmujące logikę dwu- jaki i wielowartościową zostały zaadaptowane do rozwiązań online-owych.

Zaproponowany algorytm można opisać za pomocą ogólnego wzoru algorytmu dla podejść:

$$\begin{aligned} &\forall\, T_i^O \in P\ \exists\ (w_i, t_i, h_i) \text{ oraz} \\ &\forall\, T_i^O \in P\ (t_i \leq D_i) \text{ oraz} \\ &\forall\, t \leq H\ J \subseteq Q \subseteq P \text{ oraz} \\ \forall\, t \leq H\ \exists I \subseteq J &\text{ takie, że } q_i = q_i - 1, \forall T_i^O \in I \text{ oraz} \\ &\forall T_i^O \in \{T_j^O \in T^O : q_j = 0\}\ T_i^O \notin P \text{ oraz} \\ &\forall\, t \leq H\ \exists I \subseteq J \text{ takie, że } \sum_{T_i^O \in I} a_i \leq m \end{aligned} \tag{4.23}$$

Zakłada się, że zadania są reprezentowane w jednolitej formie TG, gdzie znaczenie wtóre mają zależności pomiędzy nimi. Rozpatrywać można dwa warianty, w których: (A) występują zależności, (B) nie występują zależności. Założenie powyższe wpływa oczywiście na wyznaczanie priorytetów, gdzie oprócz tak zwanego współczynnika wieloprocesorowości (MPR) mogą być wzięte pod uwagę zależności. Poniżej zaprezentowany został pseudokod algorytmu 4.5.1 w wersji online-owej prezentujący sposób implementacji oraz działania niezależny od języka programowania. Koncepcja szeregowania wieloprocesorowych zadań online, podzielnych jak i niepodzielnych została zaprezentowana w pracy [55].

Algorytm 4.5.1. *Algorytm MC - online:*

1. $t = 0$;
2. *for* $T_i^O \in T^O$ *do oblicz* w_i*;*
3. *for* $T_i^O \in T^O$ *do oblicz* t_i*;*

4. *for* $T_i^O \in T^O$ *do oblicz* h_i*;*
5. *if* $E \neq \emptyset$ *then*
6. *for* $T_i^O \in T^O$ *do* $w_i = w_i + t_i$*; /*korekta dla zadań zależnych*/*
7. *end if;*
8. *for* $T_i^O \in T^O$ *do if* $t_i > D_i$ *then stop: uszeregowanie nie istnieje;*
9. *P=*T^O*; /* kopia* T^O **/*
10. *for* $T_i^O \in T^O$ *do* $q_i = p_i$*; /* kopia* p_i **/*
11. *while* $P \neq \emptyset$ *do*
12. *av=m;*
13. $Q = \{T_i^O \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; */*zbiór zadań gotowych*/*
14. *if* $Q = \emptyset$ *then goto end-while;*
15. $M = \max_{T_j^O \in Q} w_j$*; /*maksymalny priorytet w* Q**/*
16. $W = \{T_j^O \in Q : \ w_j = M\}$; */*znajdź zadania z priorytetem* M**/*
17. $A = \max_{T_j^O \in W} a_j$*; /*maksymalne* a_j *w* W**/*
18. $F = \{T_j^O \in W; \ a_j = A\}$; */*zadania z* W *z żądaniem* A**/*
19. $C = \max_{T_j^O \in F} h_j$*; /*maksymalne* h_j *w* F**/*
20. $J = \{T_j^O \in F; \ h_j = C\}$; */*zadania z* F *o* $h_j = C$**/*
21. *wybierz dowolne* $T_i^O \in J$*;*
22. $x = 1$*;*
23. *if* $A \leq av$ *then*
24. *uszereguj* T_i^O *dla jednej jednostki czasowej* $[t, t+1]$*;*
25. $av = av - a_i$*;* $q_i = q_i - x$*;* $Q = Q - \{T_i^O\}$*;*
26. *end-if*
27. *else*
28. $Q = Q - \{T_i^O\}$*;*
29. *end if-else;*
30. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili* t**/*
31. *else*
32. $t = t + x$*;*
33. *end if-else;*
34. *if* $q_i = 0$ *then* $P = P - T_i^O$*;*

35. end while;

Powyżej przedstawiona została koncepcja szeregowania online zadań niezależnych. Istotne jest także rozważenie wieloprocesorowych zadań niezależnych, gdzie zdefiniować można również czas ich pojawienia się. Model taki uprościć można do zadań niepodzielnych, rozpatrując w pierwszej kolejności tylko zadania wieloprocesorowe.

Opis zastosownanego podejścia:

- Jako specyfikacja systemu przyjmowany jest graf z zadaniami, gdzie wprowadzono oznaczenia dla zadań wieloprocesorowych (wybierane w sposób losowy, co do liczby żądanych zasobów oraz numeru zadania).
- Ze zbioru priorytetów wybierane są zadania o najwyższych wartościach, uwzględniając położenie każdego z zadań na ścieżce (o ile takie występuje).
- Wybierane są zadania z określonymi powyżej priorytetami (które są gotowe do wykonania w rozpatrywanej chwili czasowej, uwzględniając parametr $ready \quad time$) i następnie zadania te zostają alokowane w wybranych procesorach, tak aby (o ile to możliwe) nie występował/ nie występowały niewykorzystywane (będące bezczynnymi) procesor/procesory w danej jednostce czasu.
- Jeżeli w jednostce czasu nie występują bezczynne procesory, lub nie ma możliwości przypisania żadnego zadania do nieużywanych procesorów/procesora to czas zostanie przesunięty o kolejną jednostkę, i algorytm wykonuje się ponownie, dopóki nie zostaną uszeregowane wszystkie zadania.

Twierdzenie 4.5.1. *Dla danych specyfikowanych przez α: acykliczny $G = (V, E)$, (a_i, p_i), $T_i^O \in T^O$, algorytm MC - online w skończonej liczbie kroków dochodzi do wyniku końcowego β: $P = \emptyset$, $(q_i = 0,\ T_i^O \in T^O)$.*

Jak można zauważyć twierdzenie 4.5.1 jest niemal identyczne z twierdzeniem 4.2.1. Różnica polega w tym przypadku na uwzględnieniu zmiennych niepewnych, gdzie uwzględniana jest logika rozmyta. Zatem formalna różnica polega tylko i wyłącznie na zmianie sposobu reprezentacji danych wejściowych, poprzez analogię można zastosować dowód dla podejścia deterministycznego.

Analiza online, zadania niepodzielne

W pracy [74] przedstawione zostały zagadnienia szeregowania zadań na maszynach równoległych. Zdefiniowany zbiór zadań T wraz z czasami ich wykonania t. Przyjęto założenia, że zadania są udostępniane pojedynczo oraz w chwili pojawienia się są przypisywane do wybranego procesora. Dla tak postawionego problemu założono kryterium $C_{\max}$. Dla określonego w powyższy sposób problemu, w pracy [74] określono algorytm.

Algorytm 4.5.2. *Algorytm m-LIST*

Niech C_k, $k \in M$ będzie terminem, począwszy, od którego procesor k jest stale wolny (ang. idle). Rozpocznij od $C_k = 0$, $k \in M$. Z listy zadań oczekujących pobierz kolejne zadanie i nazwij je T_i. Wyznacz najwcześniejszy moment czasowy S_i taki, że a_i procesorów jest w stanie idle począwszy od S_i. Zaplanuj i zaktualizuj odpowiednie C_k. Powtórz proces dla kolejnych zadań z listy.

Stosując zaproponowane oznaczenia notacji Grahama, rozważany problem opisać można jako:

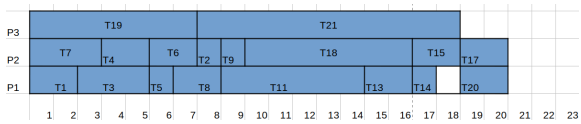
$$P|online - list, mpr|C_{\max} \tag{4.24}$$

Odwołując się do przeglądu optymalnych algorytmów online-owych w pracy [33, 74], wymienić można następujące podejścia do problemu online-list:

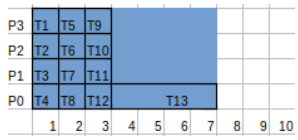
- algorytm listowy,
$$r = 2 - \frac{1}{m} \tag{4.25}$$
- algorytm Imbal
$$r = 1 + \sqrt{\frac{1 + ln2}{2}} \tag{4.26}$$

Przy założeniu, że rozpatrywane będą jedynie zadania wieloprocesorowe, dla których $MPR = const$ oraz jest zdefiniowane dla wszystkich zadań apriori przyjmuje się, że można pogrupować maszyny dla rozpatrywanych zadań w sposób następujący:

- dla zadań 2-procesorowych, grupowanie po 2 maszyny,
- dla zadań 3-procesorowych, grupowanie po 3 maszyny,
- dla zadań k-procesorowych, grupowanie po k maszyn.



Rysunek 4.8: Przykładowe uszeregowanie algorytmem m-LIST, mpr=1, k=3.

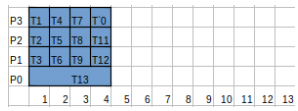


Rysunek 4.9: Uszeregowanie algorytmem on-line m-LIST, mpr=1, k=4.

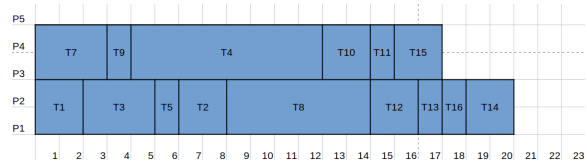
Biorąc pod uwagę założenie w niniejszym podejściu, a także specyficzne ograniczenia dla algorytmu Imbal [21], można stwierdzić, że są one nieprzydatne w wyznaczeniu współczynnika konkurencyjności. Autorzy podejścia założyli działanie algorytmu dla liczby maszyn wynoszącej co najmniej 64 oraz dążącej do nieskończoności, w takim przypadku osiągnięty został współczynnik konkurencyjności wynoszący jak we wzorze (4.26).

Twierdzenie 4.5.2. *Algorytm m-LIST dla problemu* $P|online - list, mpr = 1|C_{\max}$ *ma współczynnik konkurencyjności* $r = 2 - \frac{1}{m}$.

Jest to wynik znany z literatury, patrz np. [74]. Przykładowa instancja wraz uszeregowaniem przez $m - LIST$ została pokazana na rysunku 4.8. W rysunkach 4.9 oraz 4.10 podano ideę rozbieżności pomiędzy polityką on-line $m - LIST$ a rozwiązaniem OPT otrzymanym off-line. Rozszerzenie tej instancji na przypadek dowolnego m uzasadnia oszacowanie $r = 2 - \frac{1}{m}$ dla przypadku podanego twierdzenia. Dla dalszych twierdzeń, instancje mogą być kreowane przez analogię (przez agregację), zatem będą pomijane.



Rysunek 4.10: Uszeregowanie optymalne off-line, mpr=1, k=4.



Rysunek 4.11: Uszeregowanie zadań, gdzie mpr=2, k=5

Twierdzenie 4.5.3. *Algorytm m-LIST dla problemu* $P|online - list, mpr = 2|C_{\max}$ *ma współczynnik konkurencyjności*

$$r = \begin{cases} 2 - \frac{2}{m} & m \text{ parzyste} \\ 2 - \frac{2}{m-1} & m \text{ nieparzyste} \end{cases} \tag{4.27}$$

gdzie $[x]$ *oznacza część całkowitą liczby* x*.*

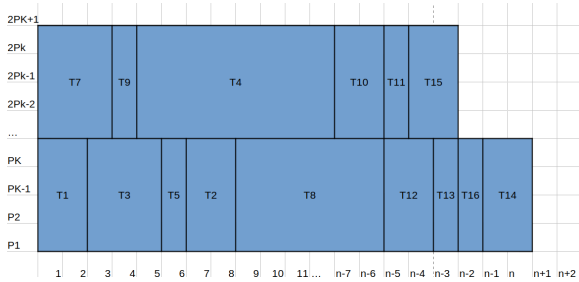
Dowód. Ponieważ zadanie wykorzystuje procesory w sposób synchroniczny, istnieje co najwyżej $[m/2]$ par procesorów równocześnie dostępnych do realizacji zadania. Bez straty ogólności rozważań załóżmy, że są to pary $(1, 2)$, $(2, 3)$, $\ldots$,$(m-1, m)$ jeśli m jest parzyste oraz $(1, 2)$, $(2, 3)$, $\ldots$,$(m-2, m-1)$ jeśli m jest nieparzyste. Zauważmy, że w przypadku nieparzystego m nie ma możliwości użycia procesora m, ponieważ nie ma on pary. Dalej transformujemy problem $P|online - list, mpr = 2|C_{\max}$ z danymi n, m, $(p_j,\ j = 1, 2, \ldots n)$ do pomocniczego problemu $P|online - list, mpr = 1|C_{\max}$ z danymi n', m', $(p'_j,\ j = 1, 2, \ldots, n)$ w następujący sposób: $n' = n$, $m' = [m/2]$, $(p'_j = p_j$, $j = 1, 2, \ldots, n)$. Zgodnie z twierdzeniem 4.5.2 zachodzi

$$r = 2 - \frac{1}{m'}. \tag{4.28}$$

Ponieważ $m' = [m/2] = m/2$ dla m parzystego oraz $m' = [m/2] = (m - 1)/2$ dla m nieparzystego, podstawiając m' do (4.28) otrzymujemy (4.27). □

Przykładowa instancja wraz uszeregowaniem przez m-LIST została pokazana na rysunku 4.11.

Korzystając z idei powyższego dowodu, możemy udowodnić przez analogię następujące twierdzenie. Niech $[x]$ oznacza część całkowitą liczby x.



Rysunek 4.12: Uszeregowanie zadań, gdzie mpr=k

Twierdzenie 4.5.4. *Algorytm m-LIST dla problemu* $P|online-list, mpr = k|C_{\max}$ *ma współczynnik konkurencyjności* $r = 2 - \frac{1}{[\frac{m}{k}]}$*, gdzie* $[x]$ *oznacza część całkowitą liczby* x*.*

Przykładowa instancja wraz uszeregowaniem przez m-LIST została pokazana na rysunku 4.12.

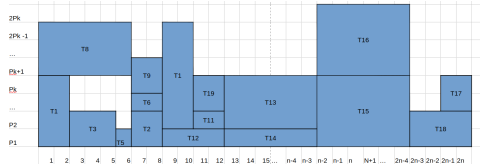
Kolejny wariant, który należy rozpatrzeć, uwzględnia wykorzystanie do szeregowania wszystkich rodzajów zadań (1-, 2-, 3- a także k-procesorowych). W przypadku tym zakładamy również że $k \leq m$.

Wprowadzone zostają następujące oznaczenia:

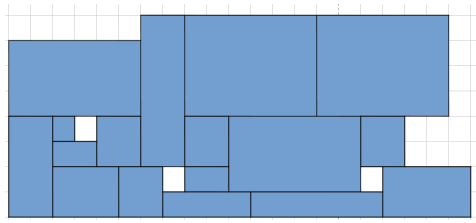
- $S(I,t)$ - oznacza zbiór zadań o liczebności $s \leq n$, takie, że wszystkie zadania s są *t-procesorowe*, oraz $t \leq m$,
- $C_{\max}(s,t)$ - oznacza wartość kryterium dla instancji określonej zbiorem $I(s,t)$,
- $LB(s,t)$ - oszacowanie od dołu instancji określonej zbiorem $I(s,t)$,

Biorąc pod uwagę powyższe założenia konstruujemy instancję złożoną zawierającą zbiór zadań $J(n,v)$, gdzie $n = (s_1 + s_2 + ... + s_u)$, $v = (t_1, t_2, ..., t_u)$ dla pewnego zbioru u. Zbiór $J(n,v)$ stanowi mnogościową sumę zbiorów $I(s_i, p_i)$. Współczynnik konkurencyjności powinien obowiązywać dla wszystkich instancji $J(n,v)$, zatem także w przypadku gdy w wektorach składowych występuje tylko jedna wartość 1 a pozostałe wynoszą 0.

Twierdzenie 4.5.5. *Algorytm m-LIST dla problemu* $P|online-list, mpr \leq k|C_{\max}$*, gdzie* $k \leq m$*, ma współczynnik konkurencyjności* $r = 2 - \frac{1}{[\frac{m}{k}]}$*, gdzie* $[x]$ *oznacza część całkowitą liczby* x*.*



Rysunek 4.13: Uszeregowanie zadań, gdzie mpr$\leq k$



Rysunek 4.14: Niegilotynowy sposób rozwiązania SPP

Funkcja $r = 2 - \frac{1}{\lceil \frac{m}{k} \rceil}$ jest malejąca względem k, zaś wartość maksymalną osiąga dla $k = 1$. Wynika stąd, że współczynnik konkurencyjności dla tej wersji algorytmu nie może być mniejszy niż dla $k = 1$.

Dla zadań dwu-procesorowych przeprowadzić można podobny ciąg następstw jak w przypadku jednoprocesorowych, rezultatem czego jest określenie minimalnej wartości współczynnika konkurencyjności na poziomie $k = 2$.

Uogólniając stosowane podejście w ten sam sposób wprowadzić można zadania $t - procesorowe$, gdzie współczynnik konkurencyjności jest określony na poziomie $k = t$. Zatem można przedstawić ogólny wzór w postaci:

$$r = 2 - \frac{1}{\lceil \frac{m}{k} \rceil}, \tag{4.29}$$

gdzie $[x]$, jest częścią całkowitą liczby x.

Dla rozważań powyższych aktualne są zaproponowane w podrozdziale 4.2 obostrzenia, gdzie $t \leq m$. Postępowanie takie doprowadza nas do wniosków z twierdzenia 4.5.4. Instancję uszeregowania dla przedstawionego problemu pokazano na rysunku 4.13. Zaś 4.14 przedstawia instancję dla problemu SPP, tym samym pokazując analogię pomiędzy uszeregowaniem zaproponowanym algorytmem a SPP.

Analiza online, zadania podzielne niezależne

Zaprezentowany model można także opisać z wykorzystaniem zadań podzielnych, rozpatrując w pierwszej kolejności tylko zadania wieloprocesorowe i przyjmując tok postępowania, jak w przypadku zadań niepodzielnych. Stosując zaproponowane oznaczenia notacji Grahama, rozważany problem opisać można jako:

$$P|pmnt, online - list, rand, mpr|C_{\max} \tag{4.30}$$

Autorzy algorytmu 4.5.3 [33] zakładają, że zadania są udostępniane w chwili gdy $t \geq 0$, zatem bezpośrednio po uszeregowaniu zadania t. Obciążenie maszyny jest oznaczone jako $\{L_i^t\}$, gdzie $L_1^t \leq L_2^t ... L_m^t$, oraz poszczególne maszyny opisane są jako M_m^t. Aktualny optymalny makespan (w kroku t) jest oznaczony jako OPT_t, natomiast suma wszystkich czasów zadań jest oznaczona jako S^t.

Podstawową cechą algorytmu jest utrzymywanie maszyn (najmniej obciążonych) na niskim poziomie obciążenia, w oczekiwaniu na dłuższe zadania. Zasada powyższa dotyczy każdego kroku algorytmu ($t \geq 0$),

1. $L_m^t \leq tOPT^t$.
2. Dla każdego $1 \leq k \leq m$,

$$\Sigma_{i=1}^k L_i^t \leq \frac{\alpha^k - 1}{\alpha^m - 1} S^t. \tag{4.31}$$

Algorytm 4.5.3. *Algorytm w kroku J_{t+1}, najpierw oblicza optymalny makespan OPT^{t+1} zgodnie z algorytmem opisanym w pracy [179]. Jeżeli $L_m^t + p_{t+1} \leq rOPT^{t+1}$, to uszereguj zadanie J_{t+1} na maszynie M_m^t. W przeciwnym wypadku $\ell = min\{1 \leq i \leq m : L_i^t + p_{t+1} \geq rOPT^{t+1}\}$. Uszeregowanie $rOPT^{t+1} - L_\ell^t$ części zadania J_{t+1} na maszynie M_ℓ^t, oraz pozostała część zadania J_{t+1}, jeżeli istnieje, na maszynie $M_{\ell-1}^t$.*

Odwołując się przeglądu optymalnych algorytmów online w pracy [26], wymienić można następujące podejście do problemu online-list:

- algorytm 4.5.3, gdzie współczynnik konkurencyjności określa się jako:

$$r = \frac{1}{1 - (1 - \frac{1}{m})^m} \tag{4.32}$$

Twierdzenie 4.5.6. *Algorytm 4.5.3 dla problemu szeregowania* $P|pmnt, online-list, rand, mpr = 1|C_{\max}$ *ma współczynnik konkurencyjności* $\frac{1}{1-(1-\frac{1}{m})^m}$.

Dowód. Odwołać się można do znanego rezultatu literaturowego, omówionego między innymi w pracy [33], ponieważ został przedstawiony jako rozwiązanie, które przy liczbie maszyn dążącej do nieskończoności dąży do $\frac{e}{e-1} \approx 1,58$, zatem przy przedstawionej specyfikacji również wykazuje taką cechę. □

Twierdzenie 4.5.7. *Algorytm 4.5.3 dla problemu* $P|online-list, mpr = 2|C_{\max}$ *ma współczynnik konkurencyjności*

$$r = \begin{cases} \frac{1}{1-\sqrt{(1-\frac{2}{m})^m}} & m \text{ parzyste} \\ \frac{1}{1-\sqrt{(1-\frac{2}{m-1})^{m-1}}} & m \text{ nieparzyste} \end{cases} \tag{4.33}$$

Dowód. Ponieważ zadanie wykorzystuje procesory w sposób synchroniczny, istnieje co najwyżej $[m/2]$ par procesorów równocześnie dostępnych do realizacji zadania. Bez straty ogólności rozważań załóżmy, że są to pary $(1,2)$, $(2,3)$, ...,$(m-1,m)$ jeśli m jest parzyste oraz $(1,2)$, $(2,3)$, ...,$(m-2,m-1)$ jeśli m jest nieparzyste. Zauważmy, że w przypadku nieparzystego m nie ma możliwości użycia procesora m, ponieważ nie ma on pary. Dalej transformujemy problem $P|pmnt, online-list, rand, mpr = 2|C_{\max}$ z danymi n, m, $(p_j, j = 1,2,\ldots n)$ do pomocniczego problemu $P|pmnt, prec, online-list, r_i, mpr = 1|C_{\max}$ z danymi n', m', $(p'_j,\ j = 1,2,\ldots,n)$ w następujący sposób: $n' = n$, $m' = [m/2]$, $(p'_j = p_j, j = 1,2,\ldots,n)$. Zgodnie z twierdzeniem 4.5.6 zachodzi

$$r = \frac{1}{1-(1-\frac{1}{m'})^{m'}}. \tag{4.34}$$

Ponieważ $m' = [m/2] = m/2$ dla m parzystego oraz $m' = [m/2] = (m-1)/2$ dla m nieparzystego, podstawiając m' do (4.34) otrzymujemy (4.33).

□

Twierdzenie 4.5.8. *Algorytm 4.5.3 dla problemu szeregowania* $P|pmnt, online-list, rand, mpr = k|C_{\max}$, *gdzie* $k \leq m$, *z kryterium* $C_{\max}$ *ma współczynnik konkurencyjności*

$$r = \frac{1}{1 - (1 - \frac{1}{[\frac{m}{k}]})[\frac{m}{k}]}, \tag{4.35}$$

gdzie $[x]$ oznacza część całkowitą liczby x.

Twierdzenie 4.5.9. *Problem $P|pmnt, online - list, rand, mpr \leq k|C_{\max}$, sprowadza się do problemu zadań jednorodnych, gdzie $mpr = const$, zatem współczynnik konkurencyjności opisuje się wzorem (4.35).*

Dowód. Funkcja opisana wzorem 4.35 jest malejąca względem k, zaś wartość maksymalną osiąga dla $k = 1$. Wynika stąd, że współczynnik konkurencyjności dla tej wersji algorytmu nie może być mniejszy niż dla $k = 1$. □

4.6. Systemy kolejkowe

Niepewność w znanych systemach kolejkowych została rozszerzona o zaproponowaną koncepcję zadań wieloprocesorowych. Istotne w systemach masowej obsługi są czasy gotowości (ang. *ready time*), czy też rozkład strumienia wejściowego. Zaproponowane podejście do SMO (Systemów Masowej Obsługi) rozstrzyga obsługę zadań wykorzystujące modyfikację algorytmu MC opisanego jako algorytm 4.6.1.

Algorytm 4.6.1. *Algorytm MC - SMO:*

1. $t = 0$;
2. *for $T_i \in T$ do oblicz w_i;*
3. *for $T_i \in T$ do oblicz t_i;*
4. *for $T_i \in T$ do oblicz h_i;*
5. *if $E \neq \emptyset$ then*
6. *for $T_i \in T$ do $w_i = w_i + t_i$; /*korekta dla zadań zależnych*/*
7. *end if;*
8. *for $T_i \in T$ do if $t_i > D_i$ then stop: uszeregowanie nie istnieje;*
9. *P=T; /* kopia T */*
10. *for $T_i \in T$ do $q_i = p_i$; /* kopia p_i */*
11. *while $P \neq \emptyset$ do*

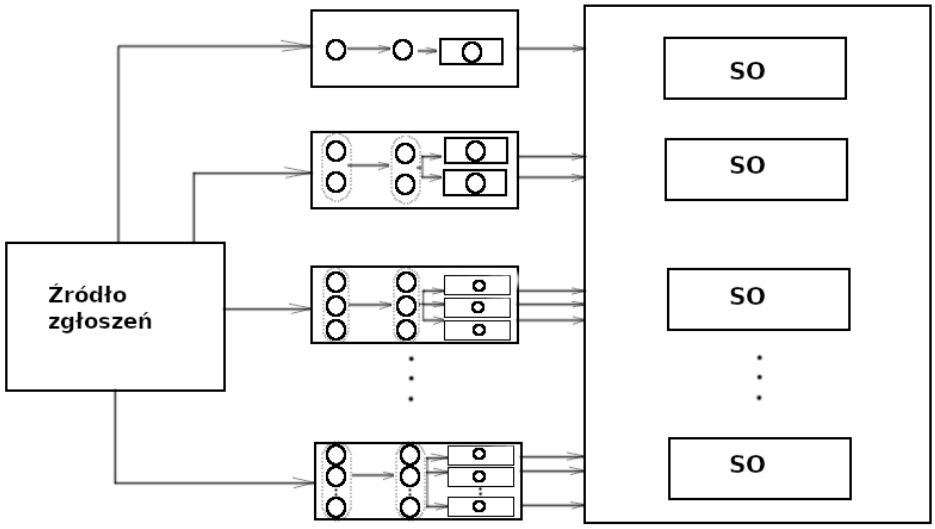
12. *av=m;*
13. $Q = \{T_i \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; */*zbiór zadań gotowych*/*
14. *if* $Q = \emptyset$ *then goto end-while;*
15. $M = \max_{T_j \in Q} w_j$*; /*maksymalny priorytet w* Q**/*
16. $W = \{T_j \in Q : \ w_j = M\}$; */*znajdź zadania z priorytetem* M**/*
17. $A = \max_{T_j \in W} a_j$*; /*maksymalne* a_j *w* W**/*
18. $F = \{T_j \in W; \ a_j = A\}$; */*zadania z* W *z żądaniem* A**/*
19. $C = \max_{T_j \in F} h_j$*; /*maksymalne* h_j *w* F**/*
20. $J = \{T_j \in F; \ h_j = C\}$; */*zadania z* F *o* $h_j = C$**/*
21. *wybierz dowolne* $T_i \in J$*;*
22. $K_j = K_j + T_i$ */* Do odpowiedniej kolejki (obsługującej zadania, gdzie* $a_k = j$ *dodaj zadanie* T_i*)*/*
23. *for* $K_j \in K$*,*
24. *for* $T_j \in K_j$
25. $x = 1$*;*
26. *if* $A \leq av$ *then*
27. *uszereguj* T_i *dla jednej jednostki czasowej* $[t, t+1]$*;*
28. $av = av - a_i$*;* $q_i = q_i - x$; $Q = Q - \{T_i\}$*;*
29. *end-if*
30. *else*
31. $Q = Q - \{T_i\}$*;*
32. *end if-else;*
33. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili* t**/*
34. *else*
35. $t = t + x$*;*
36. *end if-else;*
37. *if* $q_i = 0$ *then* $P = P - T_i$*;*
38. *end for;*
39. *end for;*
40. *end while;*

Zaproponowany model systemu mieszanego rozszerzony o obsługę zadań wieloprocesorowych został pokazany na rysunku 4.15. Można zaobserwować, że każdy

kanał jest odpowiedzialny za obsługę zadań o określonej liczbie k, gdzie zadania są dodawane do kolejek w odpowiednich kanałach. Następnie w zależności od wyznaczonego priorytetu oraz parametru $ready\ \ time$ są szeregowane na docelowej architekturze (odpowiednich SO, Stanowiskach Obsługi).

Opis zastosowanego podejścia:

- Jako specyfikacja systemu przyjmowany jest graf z zadaniami, gdzie wprowadzono oznaczenia dla zadań wieloprocesorowych (wybierane w sposób losowy, co do liczby żądanych zasobów oraz numeru zadania). Zadania mogą mieć wzajemne zależności.
- Zadania są przekazywane do odpowiedniego kanału obsługi w zależności od współczynnika mpr. Dla każdego typu zadania (uzależnionego od mpr) istnieje osobny kanał obsługi, co zostało przedstawione na rysunku 2.2.
- Ze zbioru priorytetów wybierane są zadania o najwyższych wartościach, uwzględniając położenie każdego zadania na ścieżce (o ile taka występuje).
- Stosując się do opisanych powyżej zasad, wybierane są zadania z określonymi powyżej priorytetami (które są gotowe do wykonania w rozpatrywanej chwili czasowej $ready\ \ time$) i następnie zadania te zostają alokowane w wybranych procesorach, tak aby (o ile to możliwe) nie występował/ nie występowały niewykorzystywane (będące bezczynnymi) procesor/procesory w danej jednostce czasu.
- Jeżeli w jednostce czasu nie występują bezczynne procesory, lub nie ma możliwości przypisania żadnego zadania do nieużywanych procesorów/procesora to czas zostanie przesunięty o kolejną jednostkę, i algorytm wykonuje się ponownie, dopóki nie zostaną uszeregowane wszystkie zadania.



Rysunek 4.15: Model Systemu Masowej Obsługi - z zastosowaniem zadań wielo-procesorowych

4.7. Dynamiczna zmiana zbioru zadań

Zaproponowany model systemu realizuje niepewność w zakresie żądań zasobowych, co opisano w podrozdziale 2.12. Uwzględniając kolejny parametr niepewności w systemie, należy mieć na uwadze, że koncepcja ta polega na wykonaniu zadania, że każde zadanie może wymagać do działania określonej liczby procesorów zorganizowanych w architekturze sieci NoC. Jednocześnie realizowany jest w sposób synchroniczny na określonej liczbie procesorów. Zgodnie ze specyfikacją systemu, wybrane zadania (ustalane losowo) wymagają 2 lub 3 elementów wykonawczych, pozostałe zadania realizowane są z wykorzystaniem 1 procesora. Zakładając wykorzystanie maksymalnie 3 procesorów przez jedno zadanie przyjmuje się następujące założenia:

(a) Wszystkie zadania wymagające więcej niż jednego procesora są zadaniami dwuprocesorowymi,
(b) Wszystkie zadania wymagające więcej niż jednego procesora to zadania z trzema procesorami,
(c) Zadania, które wymagają więcej niż jednego procesora, niektóre są zadaniami dwuprocesorowymi, a inne zadaniami trzyprocesorowymi,
(d) W systemie istnieją tylko zadania dwuprocesorowe,
(e) W systemie istnieją tylko zadania wymagające do realizacji trzech procesorów.

Dla przypadków a) i d) przyjmuje się również, że zadania trzyprocesorowe są wykonywane, gdy synchroniczne wykonywanie zadań dwuprocesorowych skutkuje różnymi wynikami realizacji zadania na wykorzystywanych procesorach. Tak więc, podczas gdy zadania dwuprocesorowe zwiększają niezawodność systemu, rozsądne staje się również użycie zadań trzyprocesorowych w przypadku błędnych odpowiedzi. Można rozważyć dwa skrajne przypadki a), d) i b), e) oraz szereg rozwiązań pośrednich c).

Szeregowanie z wykorzystaniem dynamicznej zmiany zbioru zadań polega na warunkowym wykonywaniu zadań (w przypadku, gdy zadanie wymaga więcej niż jednego procesora), zgodnie z algorytmem 4.7.1. Jeżeli odpowiedzi na procesorach są różne, po wykonaniu zadania dwuprocesorowego wykonywane jest zada-

nie trzyprocesorowe. Skutkiem tego jest oczywiście zwiększenie $C_{\max}$ i w niektórych przypadkach wymaga to realokacji zadań w docelowej architekturze.

Zatem zakładając istnienie prawdopodobieństwa skorelowanego z zadaniem opisującym odpowiednio: (A) prawdopodobieństwo wykonania zadania, (B) prawdopodobieństwo nieprawidłowego wykonania zadania. Można przyjąć, że podpunkt (B) implikuje podpunkt (A). Rozpoczynając od pierwotnie zdefiniowanego zbioru T, możemy zapisać zbiór zadań wymagających do realizacji zadań dwuprocesorowych:

$$Q = \{T_i \in T : \ \ a_i = 2\} \tag{4.36}$$

Bez utraty ogólności możemy założyć, że zadania T_i są indeksowane w taki sposób, że $Q = \{T_1, T_2, \ldots, T_q\}$, $q = |Q|$, $q \leq n$. Zdefiniujmy wektor zmiennych losowych $X = (X_1, X_2, \ldots, X_q)$ gdzie zmienna losowa X_i oznacza końcowy wynik przebiegu T_i, $X_i = 0$ jeśli oba procesory podały ten sam wynik i $X_i = 1$, jeśli oba procesory dostarczyły różne wyniki. W pierwszym przypadku zadanie T_i nie jest powtarzane, w drugim przypadku musimy rozszerzyć zestaw zadań T o nowe zadanie trzyprocesorowe T_i' o czasie przetwarzania równym p_i. Jest oczywiste, że X przyjmuje różne wartości 2^q. Tak więc opisuje to zbiór zdarzeń dla X. Ponieważ X_i niezależnie przyjmuje wartość 0 lub 1, to prawdopodobieństwo, że X przyjmie określoną wartość $x = (x_1, \ldots x_q)$ wynosi:

$$P(X = x) = \prod_{T_i \in Q; x_i = 0} p \prod_{T_i \in Q; x_i = 1} (1 - p) = p^r (1 - p)^{q-r}, \tag{4.37}$$

W rzeczywistości nie znamy z góry aktualnej realizacji x zmiennej losowej X, ma ona charakter dynamiczny. Wartości x zostaną ujawnione podczas procesu planowania. Dlatego został zaproponowany algorytm, który ma wbudowany mechanizm oceniania i reagowania na bieżące x.

Algorytm 4.7.1. *Algorytm MC - DZZZ:*

1. $t = 0$;
2. *for* $T_i \in T$ *do oblicz* w_i*;*
3. *for* $T_i \in T$ *do oblicz* t_i*;*

4. *for* $T_i \in T$ *do oblicz* h_i*;*
5. *if* $E \neq \emptyset$ *then*
6. *for* $T_i \in T$ *do* $w_i = w_i + t_i$*; /*korekta dla zadań zależnych*/*
7. *end if;*
8. *for* $T_i \in T$ *do if* $t_i > D_i$ *then stop: uszeregowanie nie istnieje;*
9. *P=T; /* kopia* T **/*
10. *for* $T_i \in T$ *do* $q_i = p_i$*; /* kopia* p_i **/*
11. *while* $P \neq \emptyset$ *do*
12. *av=m;*
13. $Q = \{T_i \in P : \ (q_j = 0, \ j \in B_i) \wedge (q_i > 0)\}$; */*zbiór zadań gotowych*/*
14. *if* $Q = \emptyset$ *then goto end-while;*
15. $M = \max_{T_j \in Q} w_j$*; /*maksymalny priorytet w* Q**/*
16. $W = \{T_j \in Q : \ w_j = M\}$; */*znajdź zadania z priorytetem* M**/*
17. $A = \max_{T_j \in W} a_j$*; /*maksymalne* a_j *w* W**/*
18. $F = \{T_j \in W; \ a_j = A\}$; */*zadania z* W *z żądaniem* A**/*
19. $C = \max_{T_j \in F} h_j$*; /*maksymalne* h_j *w* F**/*
20. $J = \{T_j \in F; \ h_j = C\}$; */*zadania z* F *o* $h_j = C$**/*
21. *wybierz dowolne* $T_i \in J$*;*
22. $x = 1$*;*
23. *if* $A \leq av$ *then*
24. *uszereguj* T_i *dla jednej jednostki czasowej* $[t, t+1]$*;*
25. $av = av - a_i$*;* $q_i = q_i - x$; $Q = Q - \{T_i\}$*;*
26. *if (*$a_i == 2$*)* $find(x_j)$
27. *if (*$a_i == 2$*) and (*$x_j == 1$*)*
28. *dodaj do zbioru T dodatkowe zadanie (*$T\prime_j$ *gdzie* $a_j = 3$*)*
29. *end-if*
30. *else*
31. $Q = Q - \{T_i\}$*;*
32. *end if-else;*
33. *if* $av > 0$ *go to step 14; /* gdy są jeszcze wolne procesory w chwili* t**/*
34. *else*
35. $t = t + x$*;*

36. end if-else;

37. if $q_i = 0$ *then* $P = P - T_i$*;*

38. end while;

Opis zastosowanego podejścia:

- Specyfikację stanowi graf zadań, gdzie wprowadzono oznaczenia dla zadań wieloprocesorowych (wybierane w sposób losowy, co do liczby żądanych zasobów oraz numeru zadania). Zadania mogą mieć wzajemne zależności. Dla każdego zadania dwu-procesorowego istnieje jego odpowiednik 3-procesorowy. Każde z zadań na przypisane odpowiednią zmienną opisaną rozkładem prawdopodobieństwa, która może wpłynąć na jego realizację.
- Każde zadanie może mieć zdefiniowany czas ograniczenia D_i. Jak również czas gotowości do wykonania.
- Ze zbioru priorytetów wybierane są zadania o najwyższych wartościach, uwzględniając położenie każdego zadania na ścieżce (o ile taka występuje).
- Stosując się do opisanych powyżej zasad, wybierane są zadania z określonymi powyżej priorytetami (które są gotowe do wykonania w rozpatrywanej chwili czasowej $ready \quad time$) i następnie zadania te zostają alokowane w wybranych procesorach, tak aby (o ile to możliwe) nie występował/ nie występowały niewykorzystywane (będące bezczynnymi) procesor/procesory w danej jednostce czasu.
- Do każdego zadania dwuprocesorowego jest przypisane pewne prawdopodobieństwo i na jego podstawie jest podejmowana decyzja o ewentualnym wykonaniu dodatkowego zadania 3-procesorowego.
- Jeżeli w jednostce czasu nie występują bezczynne procesory, lub nie ma możliwości przypisania żadnego zadania do nieużywanych procesorów/procesora to czas zostanie przesunięty o kolejną jednostkę, i algorytm wykonuje się ponownie, dopóki nie zostaną uszeregowane wszystkie zadania.

Dla podejścia omawianego w podrozdziale 2.12 i opisanym powyżej algorytmie 4.7.1 wykonano eksperymenty symulacyjne ukazujące słuszność koncepcji. Przeprowadzone eksperymenty zostały omówione i zobrazowane w rozdziale 5.

Rozdział 5

Przeprowadzone badania

Rozdział niniejszy przedstawia wyniki przeprowadzonych eksperymentów obliczeniowych dla weryfikacji i oceny algorytmów zaproponowanych w rozprawie. Zestawienie rezultatów pozwoli na porównanie przydatności różnych podejść. Rozważane są przykłady testowe charakterystyczne dla rzeczywistych systemów wbudowanych. Przeprowadzone badania są kontynuacją i rozwinięciem prac własnych [50–54] dotyczących szeregowania zadań wieloprocesorowych.

Badaniem objęto następujące podejścia: (A) deterministyczne; (B) wykorzystujące logikę rozmytą (wersja klasyczna oraz zmodyfikowana); (C) stochastyczne; (D) online-owe; (E) kolejkowe (masowej obsługi); (F) z dynamiczną zmianą zbioru zadań. Wychodząc początkowo od przypadku deterministycznego szeregowania wieloprocesorowego, badaniem objęto różne czynniki niepewności, w tym m.in. niepewny czas trwania zadania, skończony, lecz nieznany zbiór zadań, nieskończony losowy zbiór zadań, wariantowy zbiór zadań. Oceniano także wpływ liczby zadań, żądań zasobowych oraz grafu poprzedzań na poziom niezawodności. Badano także wpływ konfiguracji poszczególnych algorytmów na otrzymywane wyniki. Rozważono przykłady zawierające zadania niezależne oraz zależne dla wybranych grafów zależności zadań. Badania dostarczają praktycznej oceny efektywności i niezawodności, jako uzupełniającej do ocen teoretycznych. przy wykorzystaniu uniwersalnej koncepcji, przy czym dla każdego z nich używany jest odmienny sposób formułowania danych. Nie wpływa to w znaczący sposób na złożoność obliczeniową kolejnych algorytmów. Szczegółowe metody obliczenia

priorytetów zadań opisano w rozdziałach 3 oraz 4. Porównując strategie wyznaczania priorytetów zadań, można stwierdzić, że w większości przypadków nie ma znaczących różnic pomiędzy nimi, pomijając fakt różnych postaci niepewności czasów zadań. W pierwszej kolejności zaprezentowane zostały tabele zawierające zestawienie uszeregowań dla podejścia deterministycznego, a następnie dla pozostałych podejść. Przedstawione zostały nie tylko oceny rozwiązania, w sensie $C_{\max}$, ale również wyliczenia niezawodności systemu w różnych konfiguracjach. Rezultaty eksperymentów zostały przedstawione z wykorzystaniem tabel oraz wykresów Gantta dla podejść rozpatrujących podejście deterministyczne, oraz niepewność w postaci metod logiki rozmytej. Ponieważ zaprezentowanie klasycznego harmonogramu w postaci wykresu Gantta okazało się trudne koncepcyjnie w przypadku zastosowanego podejścia stochastycznego, zaproponowano wykresy obrazujące zbiorcze uszeregowanie zadań w proponowanym systemie na różnych poziomach prawdopodobieństwa. W rozdziale tym zaprezentowano również wykorzystane w celu łatwiejszej analizy uzyskanych wyników zestawienie zmian procentowych zarówno czasów uszeregowania, jak i poziomów niezawodności. Przeprowadzone badania zostały zebrane i zaprezentowane w formach graficznych ułatwiających interpretację rezultatów, odpowiednio:

- Wykres Gantta, dla podejścia deterministycznego,
- Wykres Gantta, dla podejścia z logiką rozmytą,
- Wykres zbiorczy (linia uszeregowania), dla różnych poziomów prawdopodobieństwa.
- Wykres obrazujący średni czas przepływu w SMO,
- Wykresy Gantta prezentujące długości uszeregowania dla poszczególnych wartości zmian żądań zasobowych, oraz wykresy obrazujące zmiany długości uszeregowania w wybranych przypadkach.

Ponieważ przy zastosowaniu metod prawdopodobieństwa (z uwzględnionym w tym przypadku rozkładem normalnym) nie ma możliwości w inny przystępny sposób zaprezentować na diagramie Gantta. W omawianym przypadku wymagałoby to użycia znacznej ilości diagramów, z tego powodu skonstruowany został jeden diagram dla wielu poziomów prawdopodobieństwa, co zostało dokładnie opisane w rozdziale 3.

5.1. Szeregowanie deterministyczne

Przeprowadzone zostały eksperymenty obliczeniowe dla deterministycznych różnych wybranych zbiorów zadań i grafów poprzedzania zadań. Celem badań była ocena jakości algorytmów w zależności od parametrów problemu i konfiguracji algorytmów.

5.1.1. Przykłady testowe

Wszystkie używane acykliczne skierowane grafy zadań zostały przygotowane przy wykorzystaniu podejścia z pracy [48]. Filozofia z [48] została zmodyfikowana poprzez wprowadzenie dla zadań możliwości ich wykonywania jako wieloprocesorowe. Bardziej szczegółowo, modyfikacja polegała na wprowadzeniu w każdej instancji testowej mieszanki zadań 1-, 2-, oraz 3- procesorowych, według kolekcji zapisanej w tabelach 5.1 oraz 5.2. Całkowita liczba zadań w 10 instancjach wahała się od 9 do 51. Dla każdej ustalonej liczby zadań został wybrany losowy, acykliczny graf poprzedzania zadań. Tym samym otrzymaliliśmy pewną reprezentatywną próbkę 10 różnych instancji.

5.1.2. Wyniki

Przykłady testowe rozwiązywano algorytmem 4.2.1 przedstawionym w rozdziale 4.2, dla przypadków 3-, 4- jak i 5-procesorów. Wyniki zestawiono w tabelach 5.1, 5.2. Tabela 5.1 zawiera dla każdej instancji długość uszeregowania w zależności od m (patrz sekcja ToE) oraz współczynnik niezawodności LoD obliczony według wzoru (4.1), nazywany dalej jako “Przypadek 1”. Dla większej przejrzystości wyników tabela 5.2 zawiera LoD obliczane według alternatywnego wzoru (4.2) i jest nazywane dalej jako “Przypadek 2”. Sekcje “instancje” oraz ToE w obu tabelach są takie same.

Kolejno badano wpływ zmiany liczby m na długość uszeregowania ToE (patrz tabela 5.3) oraz na poziom niezawodności LoD w dwóch przypadkach, patrz tabele 5.4, 5.5. Przykładowo, zapis $3 \rightarrow 4$ oznacza, że rozpatrywane są dwa warianty wykonywania zadań: w środowisku o $m = 3$ oraz $m = 4$. Dla każdej instancji wyliczano procentową relatywną zmianę odpowiedniego współczynnika przy zmia-

nie odpowiednio z $m = 3$ do $m = 4$. Dodatnia wartość współczynnika oznacza poprawę, to jest zmniejszenie ToE lub zwiększenie LoD.

5.1.3. Wnioski i uwagi

Z tabeli 5.1 wynika oczywisty wniosek, że długość uszeregowania ToE (tzn. $C_{\max}$) maleje wraz ze zwiększeniem liczby procesorów m. Własność ta powoduje jednak zwiększenie kosztu tworzonego systemu, a także powierzchni docelowej architektury sieci NoC. Dlatego też istotne jest uwzględnienie koniecznych i wymaganych ograniczeń dla tworzonego systemu już we wstępnej jego analizie, co zostało omówione w rozdziałach 2, 4 oraz A.

Przykładowe uszeregowania dla ustalonej instancji zawierającej taki sam zbiór zadań i graf poprzedzań, lecz różną liczbę procesorów przedstawiono na rysunku 5.1. Redukcja długości uszeregowania zależy nie tylko od liczby procesorów, ale także od liczby zadań, zarówno 2- jak 3-procesorowych, a także od postaci relacji pomiędzy wyspecyfikowanymi zadaniami.

Z tabeli 5.3 wynika, że przejście od architektury 3-procesorowej do 4-procesorowej redukuje długość uszeregowania ToE w granicach 6-40%. Redukcja ta jest wyraźnie większa (do 77%) przy przejściu z 3 procesorów na 5. Ten sam zabieg powoduje podniesienie poziomu niezawodności LoD, patrz tabela.

instancja		liczba zadań			ToE			LoD		
		procesory			procesory			procesory		
nr	n	1	2	3	3	4	5	3	4	5
1	9	4	2	3	86	80	55	0,76	0,61	0,53
2	13	5	4	4	170	130	95	0,77	0,76	0,59
3	15	5	5	5	190	125	105	0,79	0,63	0,46
4	19	5	7	7	240	185	130	0,86	0,65	0,45
5	22	6	6	10	305	255	205	0,89	0,73	0,60
6	26	8	7	11	305	285	240	0,87	0,71	0,57
7	32	11	8	13	440	365	100	0,89	0,70	0,52
8	41	13	10	18	670	545	400	0,92	0,70	0,57
9	46	12	10	24	954	820	780	0,88	0,72	0,61
10	51	16	11	24	718	764	630	0,86	0,67	0,64

Tabela 5.1: Szeregowanie deterministyczne. Poziom niezawodności. Przypadek 1. $m = 3, 4, 5$

instancja		liczba zadań			ToE			LoD		
		procesory			procesory			procesory		
nr	n	1	2	3	3	4	5	3	4	5
1	9	4	2	3	86	80	55	0,85	0,69	0,80
2	13	5	4	4	170	130	95	0,86	0,85	0,93
3	15	5	5	5	190	125	105	0,87	0,97	0,92
4	19	5	7	7	240	185	130	0,87	0,84	0,95
5	22	6	6	10	305	255	205	0,95	0,85	0,84
6	26	8	7	11	305	285	240	0,94	0,82	0,78
7	32	11	8	13	440	365	100	0,99	0,89	0,86
8	41	13	10	18	670	545	400	0,95	0,88	0,69
9	46	12	10	24	954	820	780	0,96	0,84	0,67
10	51	16	11	24	718	764	630	1	0,88	0,85

Tabela 5.2: Szeregowanie deterministyczne. Poziom niezawodności, przypadek 2, $m = 3, 4, 5$

instancja		liczba zadań			Różnica Procentowa		
		procesory			procesory		
nr	n	1	2	3	$3 \Rightarrow 4$	$4 \Rightarrow 5$	$3 \Rightarrow 5$
1	9	4	2	3	6,98	31,25	36,05
2	13	5	4	4	23,53	26,92	44,12
3	15	5	5	5	32,43	16,00	43,12
4	19	5	7	7	22,92	29,73	45,83
5	22	6	6	10	16,39	19,61	32,79
6	26	8	7	11	14,93	15,79	28,36
7	32	11	8	13	39,77	16,44	30,68
8	41	13	10	18	18,66	-0,92	17,91
9	46	12	10	24	14,05	0,00	14,05
10	51	16	11	24	16,82	17,54	29,84

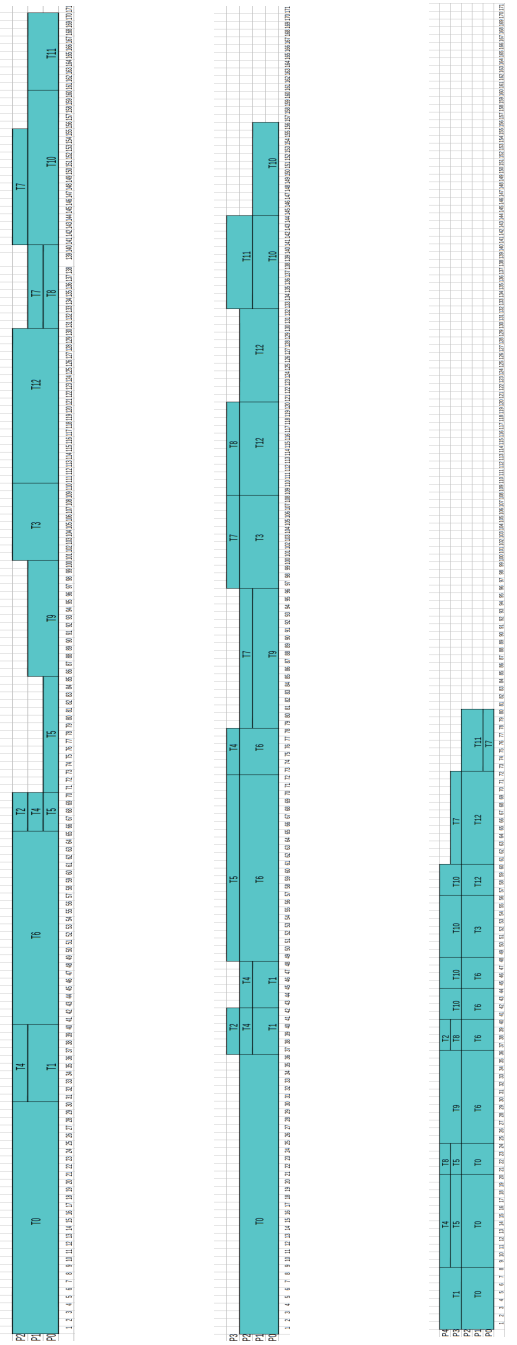
Tabela 5.3: Uszeregowanie deterministyczne, różnica czasów, $m = 3, 4, 5$

instancja		liczba zadań			Różnica procentowa		
		procesory			procesory		
nr	n	1	2	3	$3 \Rightarrow 4$	$4 \Rightarrow 5$	$3 \Rightarrow 5$
1	9	4	2	3	19,38	13,47	30,24
2	13	5	4	4	1,92	22,40	23,89
3	15	5	5	5	20,53	27,44	42,34
4	19	5	7	7	24,76	30,03	47,30
5	22	6	6	10	18,15	16,62	31,76
6	26	8	7	11	18,89	19,16	34,44
7	32	11	8	13	21,52	24,91	41,07
8	41	13	10	18	24,15	18,96	38,53
9	46	12	10	24	17,71	15,61	30,56
10	51	16	11	24	22,46	3,93	25,51

Tabela 5.4: Różnica poziomów niezawodności. Przypadek 1. $m = 3, 4, 5$

instancja		liczba zadań			Różnica Procentowa		
		procesory			procesory		
nr	n	1	2	3	$3 \Rightarrow 4$	$4 \Rightarrow 5$	$3 \Rightarrow 5$
1	9	4	2	3	19,38	-16,68	-5,85
2	13	5	4	4	1,92	-9,47	-19,60
3	15	5	5	5	-11	4,76	-16,53
4	19	5	7	7	3,48	-13,85	-10,77
5	22	6	6	10	10,29	0,49	4,79
6	26	8	7	11	11,84	5,00	10,03
7	32	11	8	13	9,59	4,46	3,87
8	41	13	10	18	7,80	20,73	24,74
9	46	12	10	24	12,74	20,00	23,89
10	51	16	11	24	11,85	2,89	1,23

Tabela 5.5: Różnica poziomów niezawodności. Przypadek 2. $m = 3, 4, 5$



Rysunek 5.1: Uszeregowania deterministyczne: $m = 3$ (górny), $m = 4$ (środkowy), $m = 5$ (dolny)

5.2. Szeregowanie rozmyte

Podrozdział niniejszy przedstawia spostrzeżenia z eksperymentów przeprowadzonych dla szeregowania zadań wieloprocesorowych wykorzystujących jako specyfikację metody logiki rozmytej. Pierwszym podejściem było klasyczne znane z literatury zastosowanie zmiennych lingwistycznych do opisu danych wejściowych (czasów trwania zadań). Badania przeprowadzone z wykorzystaniem takiego podejścia (Opisane w podrozdziale 4.3 jako podejście A) doprowadziły do wniosku, że możliwe jest zastosowanie innego modelu opisanego w niniejszym podrozdziale. Uogólniając, można stwierdzić, że wykorzystana została koncepcja już wcześniej zaproponowanego algorytmu dla podejścia deterministycznego, zmieniając podejście do specyfikacji systemu. Oczywiście konsekwencją takiego podejścia było zastosowanie odpowiednich bloków: fuzyfikacji, wyostrzania oraz defuzyfikacji, co zostało zaprezentowane i omówione w rozdziale 2. Również w tym przypadku zaproponowane zostało uszeregowanie na zestawie grafów, analogicznie jak w przypadku deterministycznym bazując na podobnych założeniach. Jak zostało zaprezentowane w rozdziale 3 wykorzystano w tym podejściu koncepcję zadań wieloprocesorowych, w której dodatkowo założono niepewność czasów wykonania zadań, co omówiono w rozdziale 2. Analogicznie jak w przypadku z podejściem deterministycznym także w tym przypadku, wśród rozpatrywanych grafów liczba zadań waha się w zakresie od 9 do 51, stanowiąc różne połączenia. Przejrzystość eksperymentów zapewniona została dzięki uogólnieniu przeprowadzonych eksperymentów do próbki 10 reprezentacyjnych grafów zadań. Przeprowadzone eksperymenty z wykorzystaniem algorytmu omówione w podrozdziale 4.3 na przykładzie schematu 4.6. Eksperymenty szeregujące zadania zrealizowano również dla docelowej architektury 3-, 4- jak i 5-procesorowych zorganizowanych z wykorzystaniem idei Network On Chip. W tabelach 5.6, 5.8, 5.9, 5.10, 5.11 oraz 5.12 zostały zaprezentowane:

- Liczba Zadań, uwzględniając ilość zadań 1-, 2- oraz 3- procesorowych,
- Czas zadań, dla funkcji przynależności 1,
- Czas uszeregowania zadań, dla funkcji przynależności 2,

Eksperymenty w zakresie logiki rozmytej przeprowadzono dla dwóch rodzajów

funkcji przynależności. Pierwsza omówiona została w podrozdziale 4.3, drugą zaprezentowano poniżej. Oba z wyspecyfikowanych w niniejszej pracy podejść wykorzystują koncepcję trójkątnej funkcji przynależności. Przy czym w przypadku drugim można opisać podejście z wykorzystaniem wzoru (4.17), gdzie a, b oraz c jest wyspecyfikowane jak we wzorze (4.17) (wzór po prawej stronie).

Każde z zaprezentowanych uszeregowań jest zrealizowane dla określonego czasu w zależności od rezultatu funkcji przynależności. Istnieje zatem możliwość wyznaczenia uszeregowania: (A) minimum; (B) medium; (C) maximum.

Gdzie poszczególne wartości stanowią rezultat trójkątnej funkcji przynależności μ_A. Stosowane figury prezentujące uszeregowania dla określonej wartości zostały przedstawione poniżej, jako rysunek 5.2, 5.2 oraz 5.3, realizujące harmonogram dla docelowego systemu o architekturze czerto-procesorowej. Dodatkowo, aby analiza wyników okazała się czytelniejsza stosowne zestawienia czasów uszeregowań zostały zaprezentowane w tabeli 5.6, 5.8, 5.9, 5.10, 5.11 oraz 5.12 odpowiednio dla architektur 3-, 4- oraz 5-procesorowej. Dla każdej z architektur eksperymenty przeprowadzono na takim samym zestawie grafów zadań, rezultaty natomiast przedstawione zostały w odpowiednich tabelach. Dla każdej architektury wyniki reprezentowane są w postaci diagramu Gantta, przedstawiając uszeregowania dla trzech rodzajów wartości omówionych powyżej. Analogicznie dla drugiej funkcji przynależności zaproponowano identyczne podejście. Docelowo wyniki zestawiono w trzech tabelach, gdzie każda z nich zawiera zestawienie uszeregowania dla właściwej liczby procesorów.

Wyniki

instancja		liczba zadań			czasy uszeregowania		
		procesory					
nr	n	1	2	3	MIN	MED	MAX
1	9	4	2	3	68	86	102
2	13	5	4	4	136	170	204
3	15	5	5	5	148	185	222
4	19	5	7	7	192	240	288
5	22	6	6	10	244	305	366
6	26	8	7	11	268	335	402
7	32	11	8	13	352	440	528
8	41	13	10	18	534	667	800
9	46	12	10	24	763	954	1145
10	51	16	11	24	718	898	1078

Tabela 5.6: Czasy uszeregowań. Model A. $m = 3$

instancja		liczba zadań			czasy uszeregowania		
		Zmienne lingwistyczne					
nr	n	Krótkie	Średnie	Długie	MIN	MED	MAX
1	9	4	2	3	44	55	66
2	13	5	4	4	76	95	114
3	15	5	5	5	84	105	126
4	19	5	7	7	104	130	156
5	22	6	6	10	164	205	246
6	26	8	7	11	192	240	288
7	32	11	8	13	248	305	397
8	41	13	10	18	320	400	480
9	46	12	10	24	624	780	936
10	51	16	11	24	504	630	756

Tabela 5.7: Czasy uszeregowań. Zastosowanie zmiennych lingwistycznych, podejście A. $m = 5$

instancja		liczba zadań			czasy uszeregowania		
		procesory					
nr	n	1	2	3	MIN	MED	MAX
1	9	4	2	3	77	86	112
2	13	5	4	4	153	170	221
3	15	5	5	5	167	185	241
4	19	5	7	7	216	240	312
5	22	6	6	10	274	305	397
6	26	8	7	11	302	335	436
7	32	11	8	13	396	440	572
8	41	13	10	18	670	667	871
9	46	12	10	24	859	954	1240
10	51	16	11	24	808	898	1167

Tabela 5.8: Czasy uszeregowań. Model B. $m = 3$ ModelB

instancja		liczba zadań			czasy uszeregowania		
		procesory					
nr	n	1	2	3	MIN	MED	MAX
1	9	4	2	3	64	80	97
2	13	5	4	4	104	130	156
3	15	5	5	5	100	125	150
4	19	5	7	7	148	185	222
5	22	6	6	10	204	255	306
6	26	8	7	11	228	285	342
7	32	11	8	13	292	365	438
8	41	13	10	18	436	545	654
9	46	12	10	24	656	820	984
10	51	16	11	24	611	764	917

Tabela 5.9: Czasy uszeregowań. Model A. $m = 4$

instancja		liczba zadań			czasy uszeregowania		
		procesory					
nr	n	1	2	3	MIN	MED	MAX
1	9	4	2	3	72	80	104
2	13	5	4	4	117	130	169
3	15	5	5	5	113	125	163
4	19	5	7	7	167	185	241
5	22	6	6	10	230	255	332
6	26	8	7	11	257	285	371
7	32	11	8	13	329	365	474
8	41	13	10	18	491	545	709
9	46	12	10	24	738	820	1066
10	51	16	11	24	688	764	933

Tabela 5.10: Czasy uszeregowań. Model B. $m = 4$

instancja		liczba zadań			czasy uszeregowania		
		procesory					
nr	n	1	2	3	MIN	MED	MAX
1	9	4	2	3	44	55	66
2	13	5	4	4	76	95	114
3	15	5	5	5	84	105	126
4	19	5	7	7	104	130	156
5	22	6	6	10	164	205	246
6	26	8	7	11	192	240	288
7	32	11	8	13	248	305	397
8	41	13	10	18	320	400	480
9	46	12	10	24	624	780	936
10	51	16	11	24	504	630	756

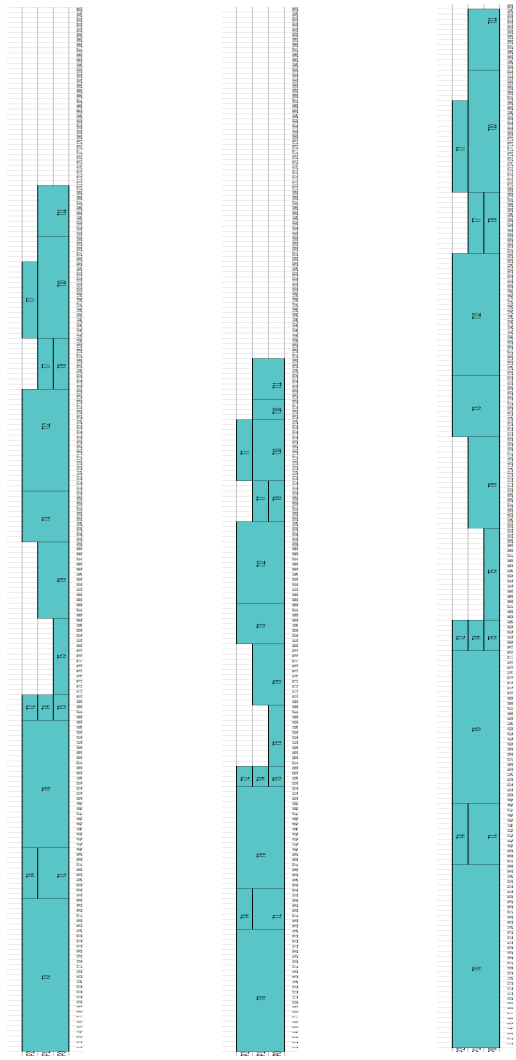
Tabela 5.11: Czasy uszeregowań. Model A. $m = 5$

instancja		liczba zadań			czasy uszeregowania		
		procesory					
nr	n	1	2	3	MIN	MED	MAX
1	9	4	2	3	50	55	72
2	13	5	4	4	86	95	124
3	15	5	5	5	95	105	137
4	19	5	7	7	117	130	169
5	22	6	6	10	185	205	267
6	26	8	7	11	216	240	312
7	32	11	8	13	244	305	366
8	41	13	10	18	360	400	520
9	46	12	10	24	702	780	1014
10	51	16	11	24	567	630	819

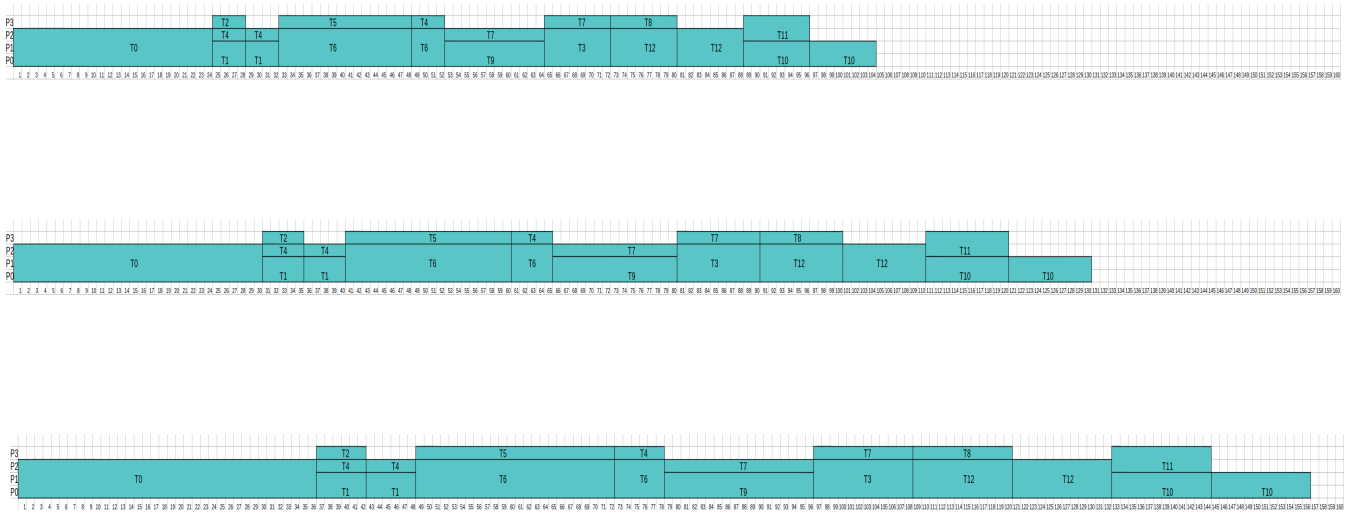
Tabela 5.12: Czasy uszeregowań. Model B. $m = 5$

instancja		liczba zadań			poziom niezawodności — procesory					
		procesory			3		4		5	
nr	n	1	2	3	D_{x1}	D_{x2}	D_{x1}	D_{x2}	D_{x1}	D_{x2}
1	9	4	2	3	2,29	2,59	2,44	2,75	2,64	267
2	13	5	4	4	2,32	2,59	3,04	3,38	2,95	4,63
3	15	5	5	5	2,38	2,62	2,52	3,88	2,29	4,62
4	19	5	7	7	2,33	2,58	2,59	3,35	2,27	4,77
5	22	6	6	10	2,62	2,84	2,90	3,39	3,02	4,22
6	26	8	7	11	2,61	2,81	2,82	3,30	2,85	3,92
7	32	11	8	13	2,65	2,97	2,79	3,58	2,62	4,28
8	41	13	10	18	2,59	2,85	2,80	3,50	2,84	3,47
9	46	12	10	24	2,61	2,87	2,89	3,34	3,05	3,34
10	51	16	11	24	2,59	2,99	2,68	3,51	3,21	4,26

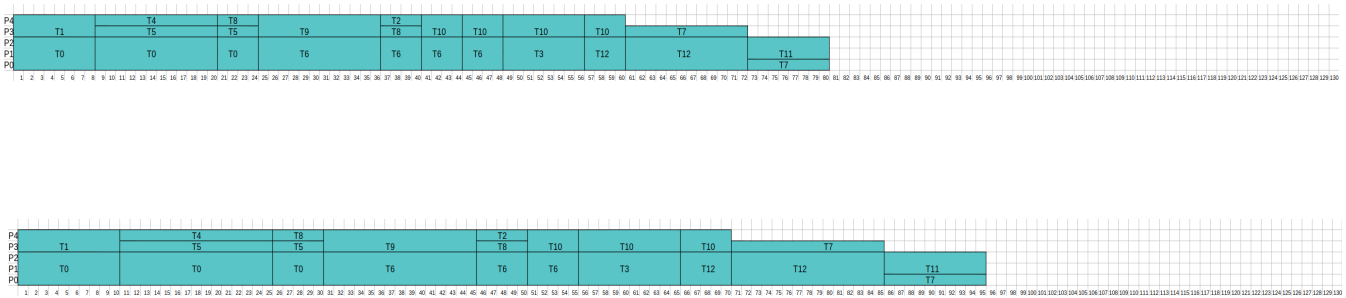
Tabela 5.13: Poziomy niezawodności. Podejście rozmyte. $m = 3, 4, 5$



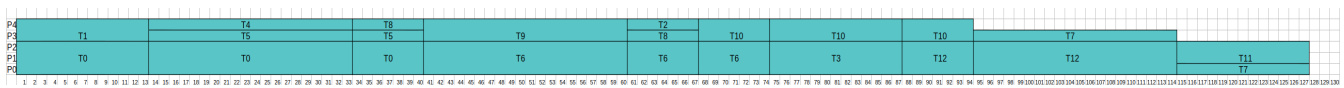
Rysunek 5.2: Uszeregowania dla $m = 3$, model A: $a = b - 20\%$ (górny), $b = 1$ (środkowy), $c = b + 20\%$ (dolny).



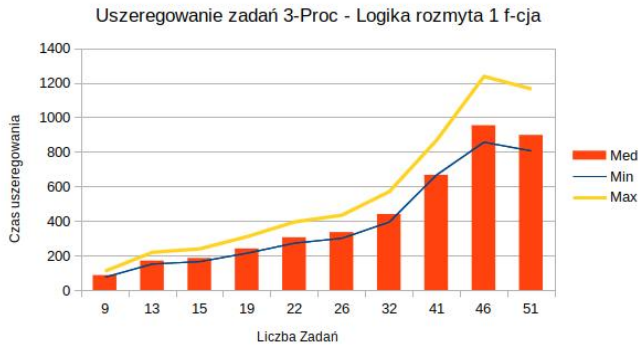
Rysunek 5.3: Uszeregowania dla $m = 4$, model A: $a = b - 20\%$ (górny), $b = 1$ (środkowy), $c = b + 20\%$ (dolny).



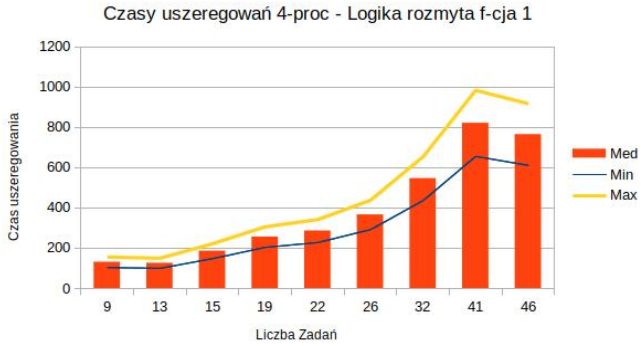
Rysunek 5.4: Uszeregowania dla $m = 5$: $a = b - 20\%$ (górny), $b = 1$ (dolny). Model A



Rysunek 5.5: Uszeregowanie dla $m = 5$: $c = b + 30\%$. Model B



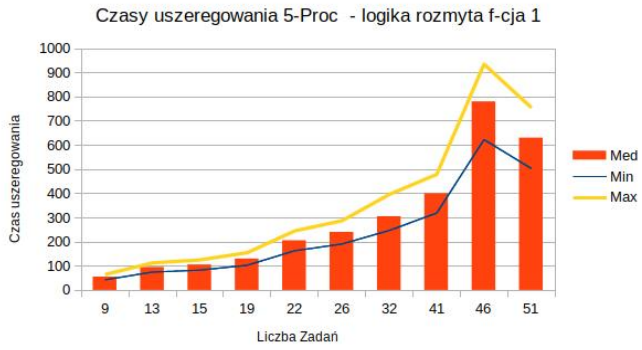
Rysunek 5.6: Różnica czasów uszeregowania, m=3, Model A



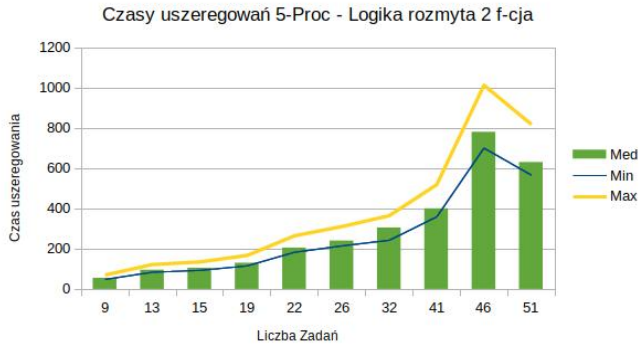
Rysunek 5.7: Różnica czasów uszeregowania, m=4, Model A

5.3. Szeregowanie stochastyczne

W podrozdziale niniejszym przedstawiono rezultaty eksperymentów dla zaproponowanych w rozdziałach powyższych metod szeregowania uwzględniających niepewność probabilistyczną. Zastosowane tutaj podejście również sprowadza się również w pierwszej wersji algorytmu ze specyfikacją zadań wykorzystującą określony rozkład prawdopodobieństwa.

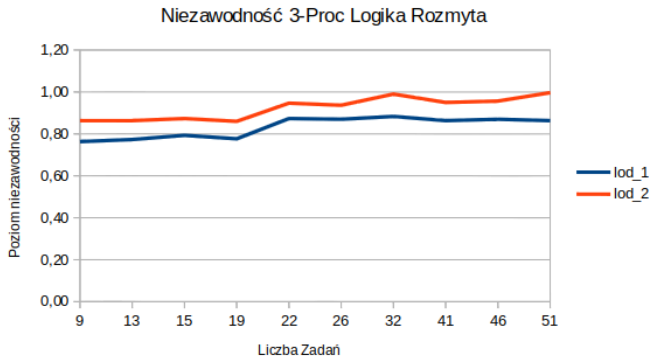


Rysunek 5.8: Różnica czasów uszeregowania, m=5, Model A

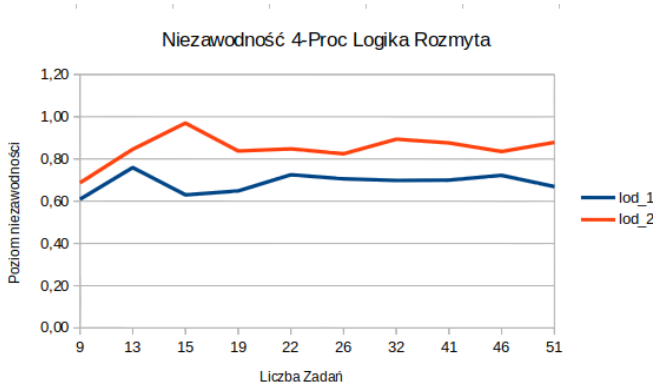


Rysunek 5.9: Różnica czasów uszeregowania, m=5, Model A

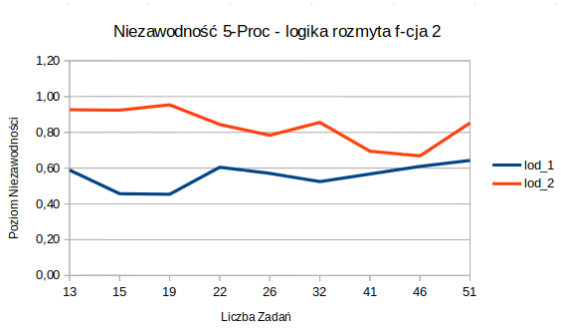
Przyjęcie w tym przypadku rozkładu normalnego jako specyfikacji czasów zadań pociąga za sobą pewne konsekwencje z tym związane. Wykorzystanie reguły σ pozwala na przeprowadzenie uszeregowania o określonej wartości prawdopodobieństwa. Zgodnie z zasadami przedstawionymi w podrozdziale 4.4 zaproponowano tutaj uszeregowanie na czterech poziomach prawdopodobieństwa w zakresie od 40% do 99.7%. Koncepcja zastosowania omówiona w rozdziale 4.4 opiera się na wykorzystaniu wektora V przedstawiającego poszczególne zakresy zgodnie z zasadą σ. Wykonane eksperymenty zrealizowane zostały analogiczne do tych



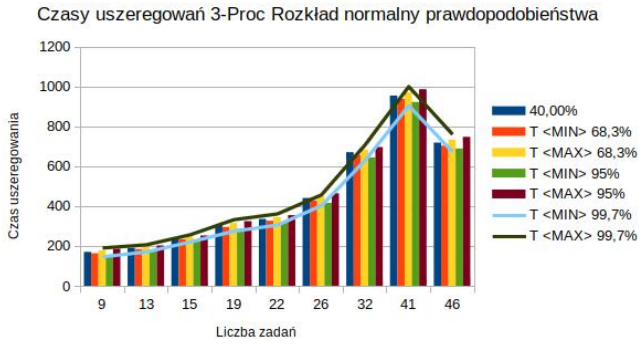
Rysunek 5.10: Różnica poziomu niezawodności: LoD - wersja 1 i 2, Model A, m=3



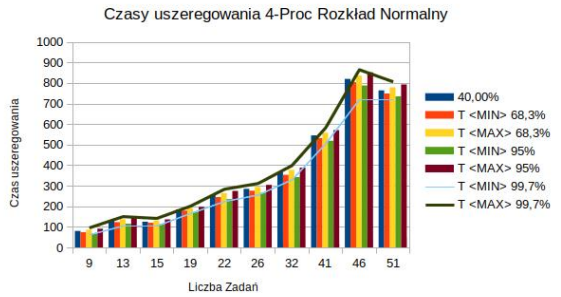
Rysunek 5.11: Różnica współczynnika niezawodności: LoD - wersja 1 i 2, Model A, m=4



Rysunek 5.12: Różnica współczynnika niezawodności: LoD - wersja 1 i 2, Model A, m=5

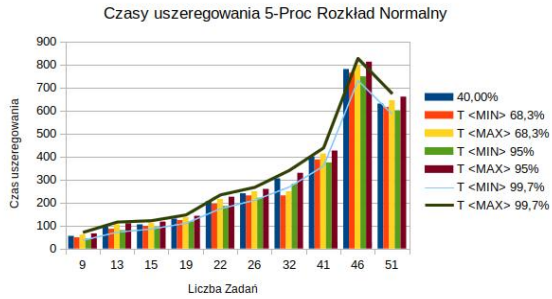


Rysunek 5.13: Uszeregowanie stochastyczne, m=3



Rysunek 5.14: Uszeregowanie stochastyczne, m=4

obejmujących podejście deterministyczne oraz logikę rozmytą (dla określonego zestawu grafów). Przestrzeń badawcza obejmuje identyczne acykliczne grafy za-



Rysunek 5.15: Uszeregowanie stochastyczne, m=5

dań jak w zrealizowanych dwóch poprzednich podejściach. Wykresy Gantta również w tym przypadku posłużyły do reprezentacji wyników uszeregowania na zadanym poziomie prawdopodobieństwa.

Na wymienionych poniżej rysunkach zaprezentowany został harmonogram na różnych poziomach prawdopodobieństwa dla omawianych w pracy architektur zorganizowanych zgodnie z koncepcją Network On Chip:

- na rysunku 5.13 dla architektury 3-procesorowej,
- na rysunku 5.14 dla architektury 4-procesorowej,
- na rysunku 5.15 dla architektury 5-procesorowej.

Ponieważ w przypadku metod prawdopodobieństwa, w rozpatrywanym w niniejszej pracy normalnym rozkładzie prawdopodobieństwa, poza poziomem 40% jest tworzony harmonogram w pewnym zakresie. Nie ma zatem możliwości zaprezentowania w łatwy sposób danego uszeregowania na diagramie Gantta jako rozwiązanie zaproponowano zestawienie uszeregowania na przywołanych powyżej rysunkach. Rozwiązanie takie nie pozwala na zaprezentowanie czasów realizacji poszczególnych zadań, a tylko rezultat docelowej długości harmonogramu. Aby przedstawić uszeregowanie szczegółowej realizacji systemu dla każdego poziomu prawdopodobieństwa (poza 40%) należy zaproponować dwa diagramy dla wartości minimum oraz maksimum, przy czym wszystkie kombinacje harmonogramu znajdujące się pomiędzy wyznaczonymi elementami są dopuszczalne.

Następnie uszeregowania dla poszczególnych architektur zostały zaprezentowane w tabelach:

- dla architektury 3-procesorowej, w tabeli 5.14,
- dla architektury 4-procesorowej, w tabeli 5.15,
- dla architektury 5-procesorowej, w tabeli 5.16,

W przypadku deterministycznym stosunkowo łatwe staje się zobrazowanie wyników uszeregowania zadań w systemie z wykorzystaniem wykresu Gantta. Dla każdej z rozpatrywanych architektur w omawianym podejściu jako rezultat otrzymywany jest jeden diagram. Komplikacja takiego sposobu reprezentacji występuje przy zastosowaniu niepewności jako specyfikacji czasu. Przy zastosowaniu do opisu logiki rozmytej przy odwzorowaniu na wybraną architekturę sporządzane są trzy wykresy dla zastosowanej funkcji przynależności. Dla podejścia stochastycznego, zwłaszcza zaproponowanego w niniejszej pracy rozkładu normalnego, niemożliwe okazuje się zaprezentowanie w prosty sposób uszeregowania na poszczególnych poziomach prawdopodobieństwa. W celu łatwiejszego zobrazowania zależności pomiędzy czasami w harmonogramie czy też poziomami niezawodności zaproponowano przedstawienie zbiorcze w postaci wykresów umieszczonych w niniejszym rozdziale.

Wyniki

instancja		liczba zadań			Rozkład normalny			
		procesory						
nr	n	1	2	3	V1	V2	V3	V4
1	9	4	2	3	86	80 - 90	75 - 97	69 - 103
2	13	5	4	4	170	163 - 178	155 - 185	148 - 192
3	15	5	5	5	190	184 - 196	178 - 202	172 - 208
4	19	5	7	7	240	234 - 264	223 - 252	222 -258
5	22	6	6	10	305	295 - 315	286 - 324	276 - 334
6	26	8	7	11	335	326 - 344	316 - 354	307 - 363
7	32	11	8	13	440	428 - 452	416 - 464	404 - 458
8	41	13	10	18	670	657 - 683	644 - 696	631 - 709
9	46	12	10	24	954	938 - 970	922 - 986	906 - 1002
10	51	16	11	24	718	703 - 733	689 - 747	674 - 762

Tabela 5.14: Czasy uszeregowań. Podejście stochastyczne. $m = 3$

instancja		liczba zadań			Rozkład normalny			
		procesory						
nr	n	1	2	3	V1	V2	V3	V4
1	9	4	2	3	80	74 - 86	69 - 91	63 - 97
2	13	5	4	4	130	123 - 137	115 - 145	107 - 152
3	15	5	5	5	125	120 - 131	113 - 136	107 - 143
4	19	5	7	7	185	179 - 191	172 - 197	167 - 203
5	22	6	6	10	255	245 - 164	235 - 275	225 - 285
6	26	8	7	11	285	276 - 294	266 - 304	257 - 313
7	32	11	8	13	365	353 - 377	342 - 388	330 - 400
8	41	13	10	18	545	532 - 558	519 - 571	506 - 584
9	46	12	10	24	820	804 - 836	788 - 852	722 - 867
10	51	16	11	24	764	749 - 779	735 - 793	720 - 808

Tabela 5.15: Czasy uszeregowań. Podejście stochastyczne. $m = 4$

instancja		liczba zadań			Rozkład normalny			
		procesory						
nr	n	1	2	3	V1	V2	V3	V4
1	9	4	2	3	55	49 - 61	44 - 66	38 - 72
2	13	5	4	4	95	86 - 105	80 - 110	73 -117
3	15	5	5	5	105	99 - 111	93 - 117	87 -123
4	19	5	7	7	130	124 - 136	118 - 142	112 -148
5	22	6	6	10	205	195 - 215	185 - 225	175 -235
6	26	8	7	11	240	231 - 249	221 - 124	212 - 268
7	32	11	8	13	305	293 - 317	281 - 329	269 - 341
8	41	13	10	18	400	387 - 413	374 - 426	360 - 439
9	46	12	10	24	780	764 - 769	748 - 812	732 - 828
10	51	16	11	24	630	615 - 645	601 - 659	585 - 674

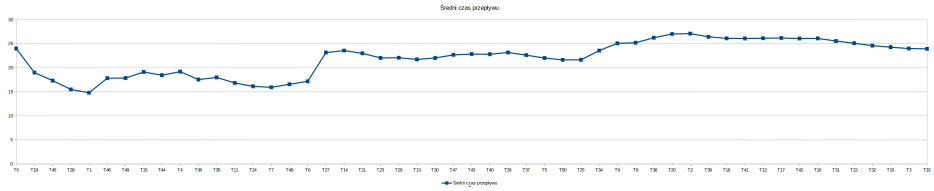
Tabela 5.16: Czasy uszeregowań. Podejście stochastyczne. $m = 5$

instancja		Czasy uszeregowania w architekturze			Procentowa zmiana Czasy uszeregowania Proc [%]		
nr	n	3	4	5	$3 \to 4$	$4 \to 5$	$3 \to 5$
1	9	86	80	55	31,35	6,98	36,05
2	13	170	130	95	26,92	25,53	44,12
3	15	190	125	105	16,00	32,43	43,24
4	19	240	185	130	29,73	22,92	45,83
5	22	305	255	205	19,61	16,39	32,79
6	26	335	285	240	15,75	14,93	28,36
7	32	440	365	305	62,26	39,77	77,27
8	41	670	545	400	26,61	18,66	40,30
9	46	954	820	780	4,88	14.05	18,24
10	51	718	764	630	15,66	16,82	29,84

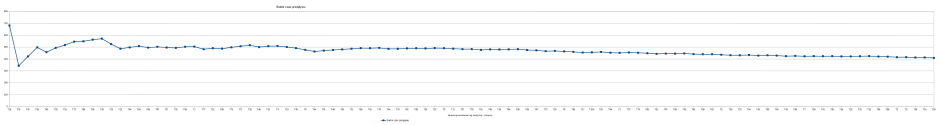
Tabela 5.17: Różnica czasów uszeregowania deterministycznego. $m = 3, 4, 5$

5.4. Szeregowanie w systemach masowej obsługi

W podrozdziale przedstawione zostały rezultaty uzyskane w wyniku symulacji przeprowadzonych dla zaproponowanego systemu kolejkowego. W tym mieszanym systemie masowej obsługi niepewność jest specyfikowana poprzez parametr $ready \quad time$ pozwalający także na określenie częstotliwości zgłoszeń kolejnych zadań. Parametr ten wpływając bezpośrednio na rozpoczęcie wykonania zadań, razem z algorytmem MC pozwala określić stabilność tego typu systemów, co zostało zobrazowane na wykresach dla 50 oraz 101 zadań, rysunki odpowiednio 5.16 oraz 5.17. Eksperymenty preprowadzono dla architektury NoC złożonej z 5-ciu procesorów, przy założeniu współczynnika wieloprocesorowości $k \leq 4$. Sporządzone wykresy ukazują dążenie systemu do ustabilizowania. Ze względów objętości została zaprezentowana tylko część rzeczywistej specyfikacji systemu w tabeli 5.18 zawierającej podstawowe parametry zadań w systemie kolejkowym (sporządzona dla systemu zawierającego 101 zadań), na podstawie tej specyfikacji został wygenerowany wykres 5.17.



Rysunek 5.16: System kolejkowy, średni czas przepływu, m=5, $k \leq 4$, $|T| = 50$



Rysunek 5.17: System kolejkowy, średni czas przepływu, m=5, $k \leq 4$, $|T| = 101$

Nr.	p_i	a_i	w_i	Ready Time	Finish Time	Avg. Time	AVG
T0	23	4	92	0	24	24	24,00
T19	14	2	28	52	66	14	19,00
T45	14	1	14	65	79	14	17,33
T28	5	2	10	71	81	10	15,50
T1	13	3	39	120	132	12	14,80
T46	6	4	24	124	157	33	17,83
T49	18	4	72	138	156	18	17,86
T15	28	1	28	147	175	28	19,13
T44	6	2	12	150	163	13	18,44
T4	26	4	104	175	201	26	19,20
T0	23	4	92	0	24	24	24,00
T19	14	2	28	52	66	14	19,00
T45	14	1	14	65	79	14	17,33
T28	5	2	10	71	81	10	15,50
T1	13	3	39	120	132	12	14,80
T46	6	4	24	124	157	33	17,83
T49	18	4	72	138	156	18	17,86
T15	28	1	28	147	175	28	19,13
T44	6	2	12	150	163	13	18,44
T4	26	4	104	175	201	26	19,20
T0	23	4	92	0	24	24	24,00
T19	14	2	28	52	66	14	19,00
T45	14	1	14	65	79	14	17,33
T28	5	2	10	71	81	10	15,50
T1	13	3	39	120	132	12	14,80

Tabela 5.18: System Masowej Obsługi, $m = 5$, $n = 50$

5.5. Szeregowanie z dynamicznym zbiorem zadań

Dla zaproponowanego podejścia w celu sprawdzenia jakości pomysłu. przeprowadzone zostały eksperymenty dla specyfikacji systemu zawierającej 9,13 oraz 15 zadań. Zarówno dla zadań zależnych jak i niezależnych. W celu specyfikacji zadań zostały wykorzystane grafy zadań, dla których specyfikację przedstawiono w tabeli 4.1. W celu zrealizowania eksperymentów do każdego z zadań wyspecyfikowanych w poszczególnych grafach zostało dodane prawdopodobieństwo ich wykonania zgodnie z regułami przedstawionymi w podrozdziale 2.12.

Rozważamy kilka przypadków o różnych właściwościach. Instancja główna ma n = 13 i występuje w dwóch wersjach, z zadaniami niezależnymi i zależnymi. Dane dla tej instancji podane są w tabeli 5.17 i dane te są wykorzystywane w przypadku zadań niezależnych, dlatego priorytety w tabeli zostały obliczone dla wersji niezależnej. Dla wersji zależnej otrzymujemy dane z Tabeli 5.17 oraz grafu zadań podanego na rysunku 4.3. Instancja ma cztery zadania z a_i = 2, a mianowicie T2, T10, T11, T12. Mamy więc Q = T2, T10, T11, T12, r = 4 i |X| = 2^4 = 16 nazywane dalej jako przypadki. Dla każdej realizacji x zmiennej losowej X obliczamy prawdopodobieństwo wystąpienia zdarzenia oraz makespan. W tym eksperymencie zostało przyjęte prawdopodobieństwo poprawnej odpowiedzi dla zadań dwuprocesorowych na 1 - p = 0,9, więc $p = 0,1$. Dla przedstawionych specyfikacji zadań przeprowadzono eksperymenty zarówno dla zadań zależnych jak i niezależnych. Dla lepszego zobrazowania ewentualnej korelacji w obu przypadkach, w eksperymentach przy zadaniach niezależnych usunięto relacje pomiędzy zadaniami. Zatem czasy oraz współczynniki mpr pozostały takie same. Badania przeprowadzono w sposób taki, aby jako pierwszą wersję, dla której wykonano szeregowanie była ta zawierająca wszystkie zadanie dwu-procesorowe (bez zmiany żądań zasobowych) z wczytanej specyfikacji w postaci T. Następnie dla określonej liczby zadań dwu-procesorowych ze zbioru T wybierane jest odpowiednio: jedno, dwa, trzy, itd. zadania (z najwyższym prawdopodobieństwem dla odpowiadającego mu zadania 3-procesorowego). Dla każdego z wariantu został przeprowadzony proces priorytetyzacji oraz szeregowania zgodnie z algorytmem 4.7.1. Zatem dla 5 zadań 2-procesorowych w grafie 15 zadań wykonano odpowiednio 32 uszeregowania,

dla 4 zadań 2-proceosrowych - 16 uszeregowań. Przedstawienie powyżej opisanej sytuacji na wykresach Gantta dla większej liczby zadań ($|T^2| \leq 4$) okazałoby się nieczytelne, przedstawiono zatem wykres zadań dla $|T^2| = 3$. Pozostałe rezultaty zobrazowano w tabelach 5.19, 5.20. Wyniki dla zadań niezależnych przedstawiono w Tabeli 5.20, a także na Rysunku 5.25 - dla zadań niezależnych i zadań zależnych na Rysunku 5.24. Dla pozostałych eksperymentów zobrazowano wyniki na Rysunkach 5.23 oraz 5.26. Można zaobserwować malejące prawdopodobieństwo wraz ze wzrostem liczby usterek. Rzeczywiście zero błędów pojawia się z prawdopodobieństwem 0,65, natomiast cztery błędy z prawdopodobieństwem 0,0001. Z tabeli 5.17 można obliczyć średnią wartość rozpiętości makespan E[Cmax(x)] - 94,1. Bardziej szczegółowa analiza ujawnia, że przypadek podstawowy (zero wad) ma udział w średniej na poziomie - 63%, a te z co najwyżej dwoma wadami po - 1%. Wpływ pozostałych przypadków (więcej niż dwa błędy) na E(.) jest nieznaczny. Rzeczywiście, obliczając średnią wartość dla przypadków z co najwyżej dwoma błędami, otrzymujemy 93,6, co jest dość dobrym przybliżeniem.

Oznacza to, że możemy w tym przypadku zmniejszyć liczbę różnych przypadków do kilku konfiguracji pierwotnych (przy założeniu niewielkiej liczby błędów).

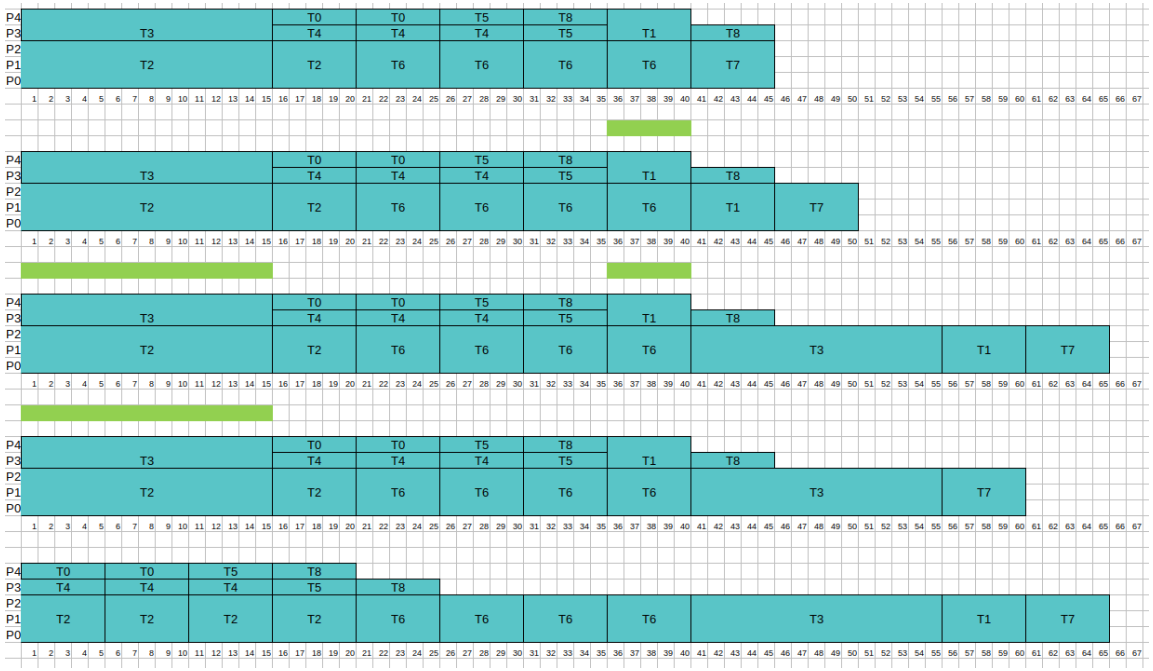
Dla każdej specyfikacji zadań dodano dwa przypadki skrajne: A) oryginalna specyfikacja systemu bez zmiany żądań zasobowych; B) zastąpienie wszystkich zadań 2-procesorowych zadaniami 3-procesorowymi. Te referencyjne przypadki pozwalają na przyjęcie punktu odniesienia w docelowym uszeregowaniu. Eksperymenty przeprowadzono na architekturze złożonej z 5 procesorów zorganizowanych w sieci NOC. Na wykresach 5.20, 5.19, 5.22 oraz 5.21 zobrazowano zmiany długości uszeregowania (oś y) w zależności od liczby zmiany żądań zasobowych (oś x). Na osi x wartości rozpoczynają się od 0, co oznacza uszeregowanie oryginalnie wczytanej specyfikacji (bez zmian żądań zasobowych), następne liczby oznaczają odpowiednio 1, 2, 3, itd... żądań zmian zasobów. Na każdym z wykresów niezależnie od specyfikacji ostatnią wartością jest $3 - Prc$ oznaczająca zastąpienie wszystkich zadań dwu-procesorowych zgłaszających żądanie zmiany zasobowej, zadaniami 3-procesorowymi. Zmiany długości uszeregowania w stosunku do liczby zmiany żądań zasobowych zostały także zobrazowane na rysunkach 5.24, 5.25, 5.23 oraz 5.26.

Nr.	p_i	a_i	w_i
T0	30	3	90
T1	10	2	20
T2	5	1	5
T3	10	3	30
T4	15	1	15
T5	20	1	20
T6	25	3	75
T7	25	1	25
T8	10	1	10
T9	20	2	40
T10	20	3	40
T11	10	2	20
T12	20	3	60

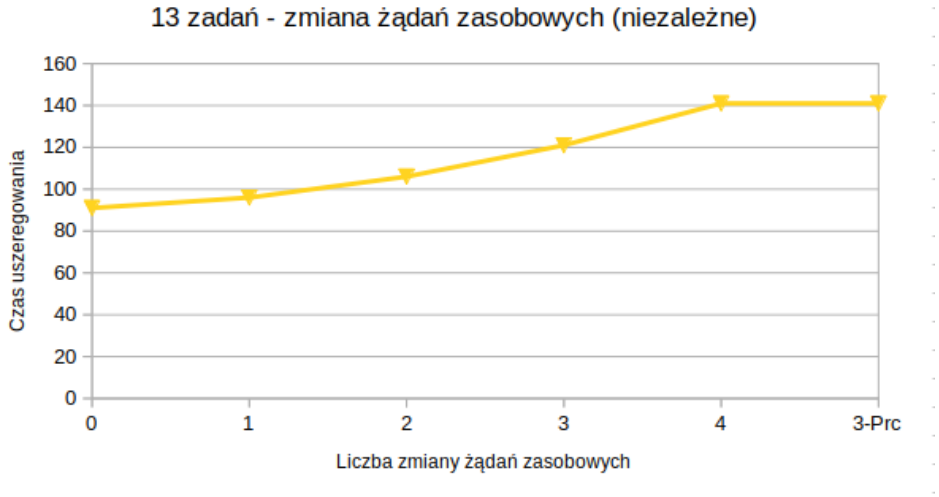
Tabela 5.19: Specyfikacja zadań, $n = 13$, zadania niezależne

Nr.	Prawdo-podobień-stwo	Liczba zmian zaso-bów	Taski z żądaniem zmiany zasobów	$C_{\max}$ $2->1$ proc.	$C_{\max}$ $2->3$ proc.
1	0.6561	0/4		95	95
2	0,0729	1/4	T1	95	95
3	0,0729	1/4	T9	105	110
4	0,0729	1/4	T10	105	105
5	0,0729	1/4	T11	105	105
6	0,0081	2/4	T1, T9	105	110
7	0,0081	2/4	T1, T10	105	115
8	0,0081	2/4	T1, T11	95	105
9	0,0081	2/4	T9, T10	95	120
10	0,0081	2/4	T9, T11	105	105
11	0,0081	2/4	T10, T11	105	115
12	0,0009	3/4	T1, T9, T10	105	130
13	0,0009	3/4	T1, T9, T11	105	120
14	0,0009	3/4	T1, T10, T11	105	125
15	0,0009	3/4	T9, T10, T11	100	130
16	0,0001	4/4	T1, T9, T10, T11	105	140

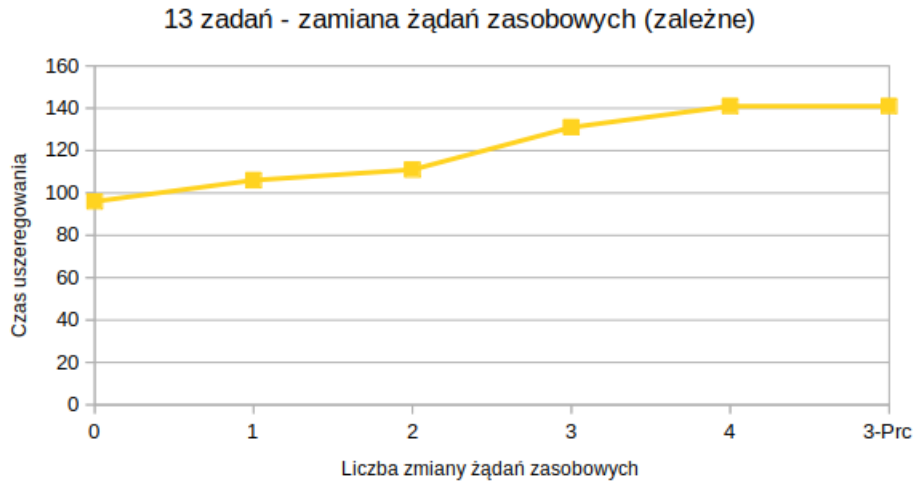
Tabela 5.20: Zmiana zasobów, $n = 13$, zadania niezależne



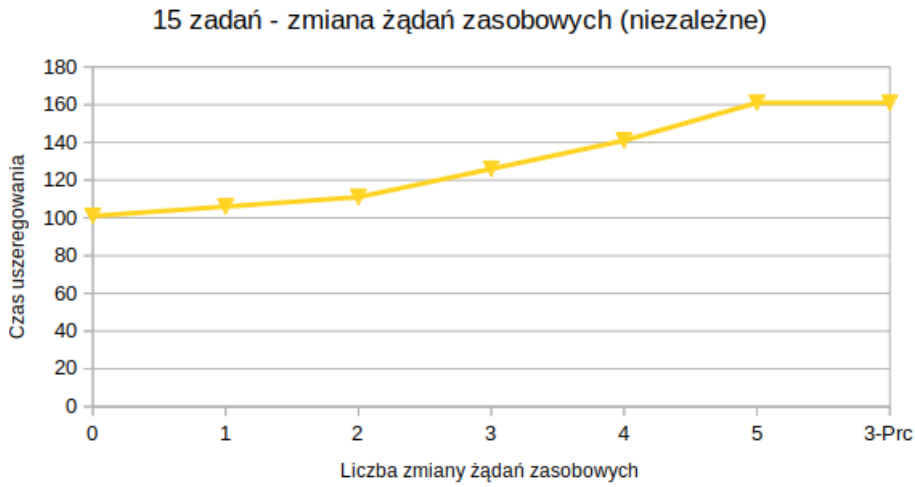
Rysunek 5.18: Dynamiczna Zmiana Zbioru Zadań,$m = 5$, $k \leq 3$, $|T| = 9$, zadania niezależne



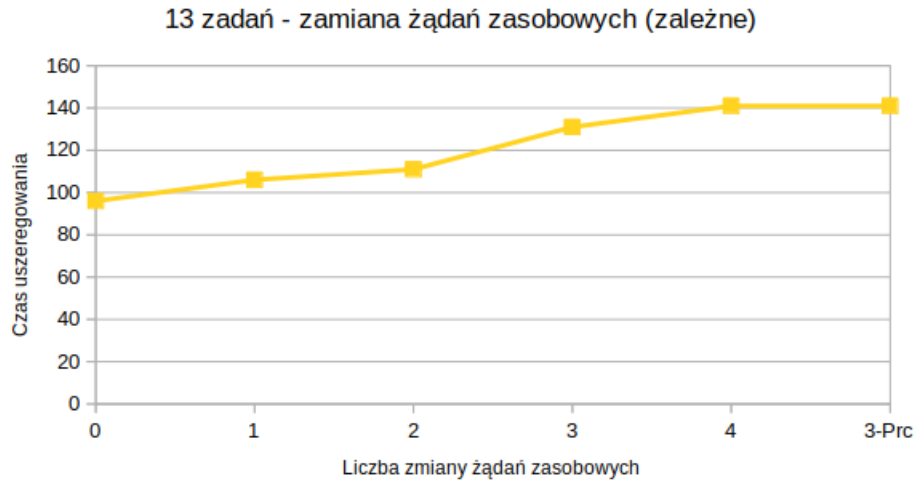
Rysunek 5.19: dZZZ, Długość uszeregowania - liczba dynamicznej zmiany zbioru zadań, m=5, $k \leq 3$, $|T| = 13$, zadania niezależne



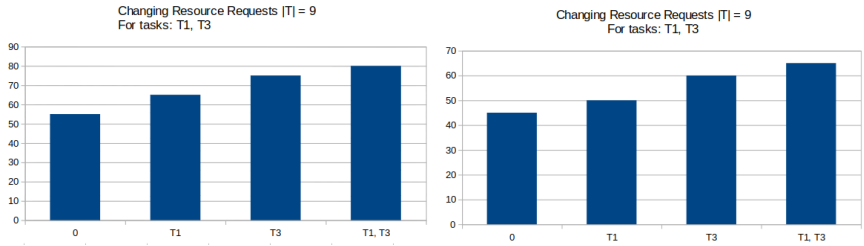
Rysunek 5.20: dZZZ, Długość uszeregowania - liczba dynamicznej zmiany zbioru zadań, m=5, $k \leq 3$, $|T| = 13$, zadania zależne



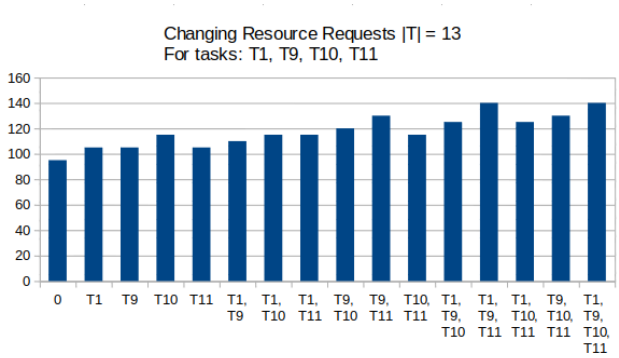
Rysunek 5.21: dZZZ, Długość uszeregowania - liczba dynamicznej zmiany zbioru zadań, m=5, $k \leq 3$, $|T| = 15$, zadania niezależne



Rysunek 5.22: dZZZ, Długość uszeregowania - liczba dynamicznej zmiany zbioru zadań, m=5, $k \leq 3$, $|T| = 15$, zadania zależne



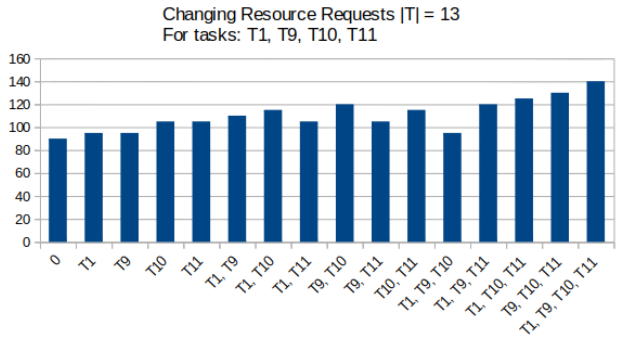
Rysunek 5.23: Histogramy dla uszeregowania zadań ze zmianą żądań zasobowych, |T| = 9, Zadania niezależne (rysunek po stronie prawej), zadania zależne (rysunek po stronie prawej)



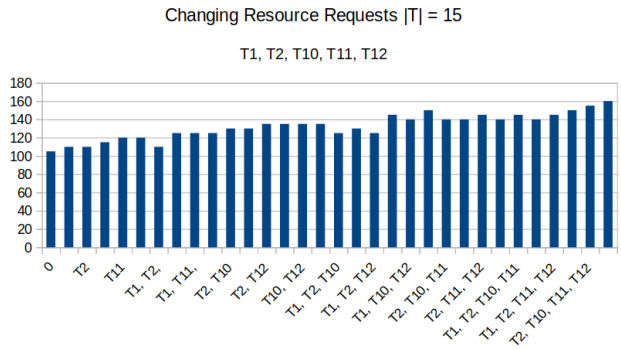
Rysunek 5.24: Histogram dla uszeregowania zadań ze zmianą żądań zasobowych, |T| = 13, zadania zależne

Zaobserwować można wprost proporcjonalny wzrost długości uszeregowania do zwiększającej się liczby zmiany żądań zasobowych. Parametr ten wzrasta zarówno w przypadku zadań zależnych, jak i niezależnych oraz jest w większości przypadków (maksymalnej możliwej liczby zgłoszeń żądań zmiany procesorowej) taki sam jak dla zastąpienia wszystkich zadań 2-procesorowych zadaniami 3-procesorowymi. W kilku przypadkach długość uszeregowania jest mniejsza niż przy zmianie zadań na 3-procesorowe, dotyczy to jednak tylko przypadków z zadaniami zależnymi. Długość uszeregowania ma mniejsze tendencje wzrostowe przy wystąpieniu dodatkowego zadania jednoprocesorowego (zamiast zadania trzy-procesorowego) w wyniku różnych odpowiedzi na zadaniach dwu-procesorowych.

Można zatem stwierdzić, że w systemach obarczonych dużym prawdopodobieństwem nieprawidłowych wyników przy realizacji zadań na dwóch procesorach dla wszystkich zadań, korzystniejsza jest ich zmiana na zadania 3-procesorowe. W pozostałych przypadkach (innych niż maksymalna możliwa liczba zgłoszeń żądań zmian zasobowych) korzystne jest wykorzystanie zmiany żądań zasobowych z zadań dwu- na trzy- a tym bardziej jedno-procesorowe. Biorąc pod uwagę wcześniejsze rezultaty $93,6\%$ dla co najwyżej 2 błędów to korzystne jest zastosowanie zmiany żądań zasobowych zwłaszcza stosując zadania jedno-procesorowe.



Rysunek 5.25: Histogram dla uszeregowania zadań ze zmianą żądań zasobowych, |T| = 13, zadania niezależne



Rysunek 5.26: Histogram dla uszeregowania zadań ze zmianą żądań zasobowych, |T| = 15, zadania zależne

Rozdział 6

Wnioski i uwagi

Wyniki badań z rozdziału 5, potwierdzają zasadność zaproponowanych metod. Zaprezentowano nowe podejścia do uszeregowania zadań w warunkach niepewności, rozszerzając podejście zastosowane do oryginalnego algorytmu własnego MC. Przedstawiona metoda bazuje na wcześniejszych pracach autora, wprowadzając nie tylko uwzględnianie zadań wymagających do realizacji wielu procesorów w jednej chwili czasowej, ale także rozpatrując istotny czynnik niepewności.

Przywołamy tutaj postawione tezy rozprawy w celu odniesienia sie do nich:

- zadania wieloprocesorowe modelują procesy obliczeniowe wymagające zwiększonej niezawodności programów i urządzeń, zwłaszcza w systemach wbudowanych, poprzez redundancję sprzętową i programową,
- problemy szeregowania zadań wieloprocesorowych z niepewnymi danymi lepiej modelują rzeczywiste systemy sterowania niż problemy deterministyczne,
- możliwe jest skonstruowanie algorytmów szeregowania zadań wieloprocesorowych z danymi niepewnymi poprzez rozszerzenie dotychczasowych wyników znanych dla problemów deterministycznych,
- różne podejścia do modelowania dostarczają różne algorytmy, różne wyniki teoretyczne i symulacyjne.

Tezy zostały potwierdzone poprzez skonstruowanie algorytmów uwzględniających niepewne parametry zamodelowane poprzez wybrane koncepcje omówione w treści pracy oraz przeprowadzenie eksperymentów dla zaproponowanych algoryt-

mów, czy też zaproponowanie i przeprowadzenie dowodów twierdzeń.

W podrozdziale 4.1 zaprezentowano oraz omówiono koncepcję redundancji zadań i wzajemnego ich testowanie, która wpływa na podniesienie niezawodności całego systemu. Oczywiste jest, że zastosowanie modeli rzeczywistych systemów musi uwzględniać pewne parametry nacechowane niepewnością, jak np. niewłaściwa realizacja zadań, nieoczekiwane wystąpienie awarii, itp.

W pracy niniejszej dla algorytmu zaproponowanego w rozdziale 4 przeprowadzono badania w zakresie: (A) podejścia deterministycznego, (B) logiki rozmytej, (C) stochastyki, (D) podejścia online'owego, (E) systemów kolejkowych, (F) dynamicznego zbioru zadań. Na podstawie analizy podejścia omówionego w rozdziale 4 zaproponowany został model matematyczny. Przedstawiono koncepcję, rozpatrując wszystkie elementy modelu jako operacje na zbiorach zawierających podstawowe elementy specyfikujące system we wszystkich aspektach. Została określona teoretyczna złożoność obliczeniowa algorytmu dla podejścia deterministycznego. Ponieważ wszystkie zaproponowane podejścia opierają się na algorytmie zastosowanym dla podejścia deterministycznego, również i złożoność obliczeniowa może zostać oszacowana w ten sposób, dla pozostałych koncepcji. Podejście poprzez analogię skupia się na tym, że w pozostałych zaproponowanych przypadkach wykonywane są identyczne kroki (jak w przypadku deterministycznym), przy założeniu, że są one rozszerzone o odpowiednie elementy rozpatrujące niepewność. Te dodatkowe czynniki nie wpływają na złożoność obliczeniową. Można zatem uznać, że wyznaczona złożoność jest identyczna dla wszystkich podejść. Wywnioskować można, że bloki fuzyfikacji jak i defuzyfikacji oraz rozpatrywanie różnych wartości prawdopodobieństwa nie są czynnikiem ograniczającym czas uzyskiwania rezultatu. W systemach ze zamianą żądań zasobowych zaobserwować można wzrost długości uszeregowania wprost proporcjonalny do rosnącej liczby zmiany żądań zasobowych. Można zatem stwierdzić, że w systemach obarczonych dużym prawdopodobieństwem nieprawidłowych wyników przy realizacji zadań na 2-procesorach dla wszystkich zadań, korzystniejsza jest ich zmiana na zadania 3-procesorowe. W pozostałych przypadkach (innych niż maksymalna możliwa liczba zgłoszeń żądań zmian zasobowych) korzystne jest wykorzystanie zmiany żądań zasobowych. Wykorzystanie niepewności pozwoliło

na zamodelowanie systemów bardziej odzwierciedlających rzeczywiste systemy oraz czynniki w nich występujące, które ukierunkowane są na podniesienie niezawodności.

Zgodnie z koncepcją przedstawioną w podrozdziale 4.2, algorytm wyznacza priorytety dla zadań, faworyzując przy tym wieloprocesorowość. Poziom zadania określany także priorytetem jest wprost proporcjonalny do MPR. Zaproponowany algorytm stanowi rozwiązanie optymalne dla rozpatrywanego problemu. Zgodnie z zaproponowanym modelem przeprowadzonych zostało szereg eksperymentów, których rezultaty zobrazowano w rozdziale 5.

Eksperymenty zostały wykonane z wykorzystaniem zestawu grafów TG zawierających od 9 do 51 zadań, zakładając, że każdy graf zadań zawiera określoną liczbę zadań. Każdy z wygenerowanych grafów zrealizowano z różną konfiguracją połączeń pomiędzy zadaniami. Począwszy od grafów zawierających 9 zadań oraz zawierające nieskomplikowane połączenia pomiędzy zadaniami aż po te zawierające maksymalną przewidzianą liczbę zadań. Można określić, że dla każdej liczby zadań przeprowadzono testy, począwszy od prostych a, skończywszy na skomplikowanych przykładach. W rozdziale 5 przedstawione zostały w poszczególnych tabelach rezultaty przeprowadzonych eksperymentów. Wszystkie uszeregowania wykazują, że zaproponowane podejście pozwala na realizację efektywnego harmonogramu dla każdej ze specyfikacji. Docelowe uszeregowanie systemu zostało, w każdym przypadku, zrealizowane na architekturze opartej na koncepcji NoC.

Dla każdej specyfikacji systemu zaproponowanej przez graf zadań wyznaczony został optymalny harmonogram. W zaproponowanym podejściu jako pierwsza realizowana jest priorytetyzacja zadań zgodnie z przyjętymi zasadami. Kolejnym krokiem jest alokacja zadań w poszczególnych procesorach w zależności od priorytetów oraz symulacja ich wykonania. Eksperymenty potwierdzają wysnute wnioski co do efektywności oraz optymalności algorytmu w rozpatrywanym zakresie. Wykonano szeregowanie zadań 1-, 2- oraz 3-procesorowych dla architektur zorganizowanych jako 3-, 4- oraz 5-procesorowe sieci NoC. Rezultatem oczywistym jest wykonanie zadań w systemie, w krótszym czasie przy zastosowaniu większej liczby procesorów. Dodatkowym aspektem jest rozpatrywana w przypadku każdego harmonogramu niezawodność, która jest wprost proporcjonalna do

liczby zadań wieloprocesorowych. Zarówno zadania dwu- jak i trzy-procesorowe wpływają na zwiększenie docelowej niezawodności systemu. Na poziom niezawodności nie wpływa architektura, a w tym konkretnym przypadku liczba użytych procesorów w NoC. Przeprowadzone badania dowodzą, że zaproponowany model wieloprocesorowości istotnie wpływa na wzrost niezawodności docelowego systemu, przy zaproponowaniu najlepszego z możliwych harmonogramu.

Dzięki rozszerzeniu notacji Grahama o parametry opisane w rozdziale 2.3 oraz 4.5 można precyzyjnie określić i usystematyzować ten aspekt harmonogramowania. W kontekście zaprezentowanej tezy rozpatrywany jest aspekt związany z nieokreślonym czasem udostępnienia zadań, przy założeniu niewystępowania zależności pomiędzy zadaniami. Został uwzględnieniony również parametr podzielności zadań. Kolejnym parametrem niepewności jest zamiana żądań zasobowych dla rozważanych w pracy specyfikacji systemu. Biorąc pod uwagę przedstawioną skrótowo koncepcję, zaproponowano poparte dowodami twierdzenia dotyczące zarówno identycznych zadań o stałym współczynniku wieloprocesorowości, jak i tak zwanych zadań wymieszanych. Na podstawie teoretycznie rozpatrzonych aspektów szeregowania można zaproponować modele eksperymentalne. Określenie klasy problemów dla proponowanych podejść pozwala na dobranie adekwatnych do złożoności algorytmów, co w przyszłości usprawni praktyczne zweryfikowanie postawionych założeń.

Jednym z możliwych kierunków rozszerzenia może być rozpatrzenie niepewności czasów zadań, badając inne funkcje przynależności w obszarze klasycznej logiki rozmytej. Innym podejściem może być uwzględnienie rozkładu χ^2 lub też Erlanga, dokładniej opisujące tego typu zdarzenia. Ciekawym aspektem okazać się może dokładniejszy opis niepewności danych z wykorzystaniem skierowanych liczb rozmytych oraz teorii zbiorów przybliżonych. Natomiast w obszarze algorytmów online-owych z całą pewnością interesującym obszarem będzie uwzględnienie parametru określającego zależności pomiędzy zadaniami. Oczywistym kierunkiem rozwoju jest także uwzględnienie niepewności w szerszym aspekcie, zarówno w klasie algorytmów deterministycznych jak i online-owych, rozpatrując więcej niż jeden nieprecyzyjny parametr.

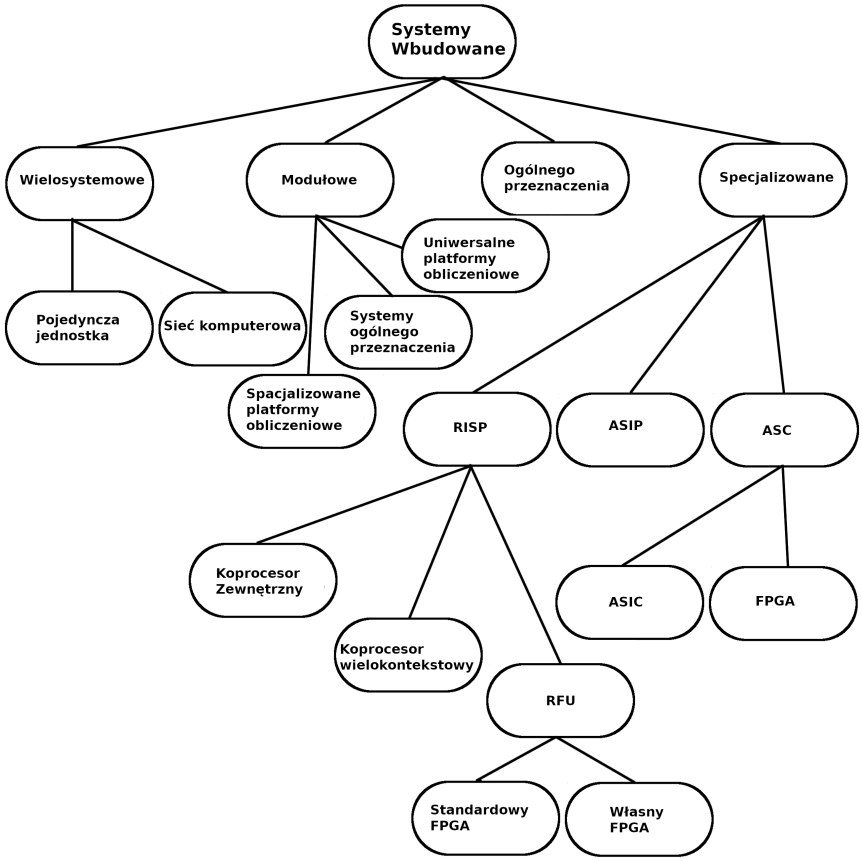
Dodatek A

Architektury systemów wbudowanych

W rozdziale poniższym przedstawiono i omówiono podstawowe architektury systemów wbudowanych. Scharakteryzowane struktury stosowane w celu utworzenia układu osadzonego zostały ułożone hierarchicznie, zgodnie z następującym w czasie ich rozwojem oraz ewolucją. Szczególna uwaga poświęcona została architekturom NOC (ang. *Network On Chip*). Ze względu na niejednoznaczne tłumaczenia w literaturze w rozdziale tym używane będą następujące określenia, które traktowane są jako tożsame: (A) systemy wbudowane i systemy osadzone, (B) magistrala danych i szyna danych, (C) architektura komputerów i organizacja architektury komputerów.

A.1. Klasyfikacja systemów wbudowanych

W obecnych czasach z systemami wbudowanymi można spotkać się niemal w każdej dziedzinie nie tylko przemysłu, ale i życia codziennego. Systemy osadzone mogą zatem znaleźć zastosowanie w telefonach komórkowych, aparatach fotograficznych, pralkach automatycznych, a także w wielu innych urządzeniach z zakresu elektroniki użytkowej. Swoje zastosowanie mają także w specyficznych branżach jak np. zastosowanie sterowników modułu ABS w przemyśle motoryzacyjnym, a także wielu innych. Każde ze wspomnianych urządzeń, które powstały,



Rysunek A.1: Klasyfikacja systemów wbudowanych

czy też były rozwijane w trakcie ostatnich dziesięcioleci wymuszało zastosowanie coraz to nowych technologii wytwarzania tego rodzaju systemów. Nieustanny oraz w ostatnich czasach bardzo szybki rozwój technologii półprzewodnikowych pozwala na wytwarzanie coraz mniejszych, a jednocześnie odznaczających się większą wydajnością systemów wbudowanych.

W pracy [95] została zaproponowana klasyfikacja, w myśl której można podzielić systemy wbudowane między innymi ze względu na przeznaczenie. Wyróżnia się systemy wbudowane ogólnego przeznaczenia oraz specjalizowane. Stała architektura koprocesorów spełnia istotną rolę w obszarze systemów osadzonych. Dzięki zastosowaniu wielkoseryjnej produkcji w tym rodzaju układów można zminimalizować koszty wytwarzania. Jednakże uniwersalność zastosowania tych sys-

temów powoduje nadmierne użycie układów scalonych w koprocesorze, co przekłada się na wykorzystanie tych układów tylko na średnim poziomie [159]. Jako poprawę efektywności dla systemów wbudowanych ogólnego przeznaczenia zaproponowano wykonanie specjalizowanego koprocesora realizującego określone funkcje lub zastosowanie rekonfigurowalnego koprocesora. Rozwiązanie to definiuje specjalizowane systemy osadzone, które można z kolei podzielić na ASC (ang. *Application Specific Coprocessor*), ASIP (ang. *Application Specific Instruction-Set Procesor*) oraz RISP (ang. *Reconfigurable Instruction Set Processor*). Spośród ASC można wyodrębnić ASIC (ang. *Application Specific Integrated Circuit*) oraz FPGA (ang. *Field Programmable Gate Array*). Procesory programowalne podzielić można na te zawierające: (A) Koprocesor zewnętrzny, (B) Koprocesor wielokontekstowy, (C) RFU (ang. *Reconfigurable Functional Unit*) [95, 104, 159]. Specjalizowane systemy wbudowane złożone są ze ściśle ze sobą współpracujących komponentów programowych oraz sprzętowych. Utworzenie docelowego systemu wymaga, umiejętnego zróżnicowania funkcjonalności pomiędzy sprzętem a oprogramowaniem przy użyciu specjalistycznych metod uwzględniając niezbędne ograniczenia. Spośród wymienionych tutaj rodzajów systemów wbudowanych szczególna uwaga poświęcona została sieciom NOC, które są wykorzystywane do utworzenia docelowego systemu. W rozdziale niniejszym zostały zaprezentowane i omówione najistotniejsze kwestie wybranych rodzajów systemów wbudowanych: ASIC, ASIP, RISP, SOC, MPSOC, FPGA, SOC-FPGA.

ASIC

W połowie lat osiemdziesiątych dwudziestego wieku zaczęła rozwijać się technologia VLSI (ang. *Very Large Scale of Integration*), przekraczając granicę integracji układów ULSI (ang. *Ultra Large Scale Integration*). Pomimo swoich ograniczeń co do powierzchni projektowanych układów ULSI miały wpływ na projektowanie wczesnych systemów mikroprocesorowych, [101]. Wspomniane rodzaje układów są naturalnymi następstwami procesów tworzenia układów, aby docelowo umożliwić powstanie technologii ASIC (ang. *Application Specific Integration Circuit*) [36], [100], [6]. Ponieważ ASIC wykorzystuje tylko dobrze sprawdzoną technologię, staje się tym samym podejściem o co najmniej jedną generację

technologiczną opóźnione w stosunku do ULSI. Mimo to ASIC stało się najbardziej popularną filozofią wytwarzania układów. Ich powstanie zrewolucjonizowało technologię projektowania oraz produkcję w branżach związanych z elektroniką. Przed zaproponowaniem omawianego podejścia, na projektowanie szeroko rozumianych układów elektronicznych składało się dobór oraz integracja mikroprocesorów, oraz standardowych układów zintegrowanych. Natomiast podejście do tworzenia układów ASIC można traktować jako integrację systemów w układach scalonych. Taka filozofia tworzenia układów pozwala na lepsze zrozumienie wymagań aplikacji, charakterystyki wykorzystywanych algorytmów, głównych założeń architektonicznych tworzonego systemu, a także wszelkich interakcji pomiędzy elementami składowymi układu [101]. Przed projektantami systemów wbudowanych stawiane są wymagania optymalizacyjne takie jak: (A) zmniejszenie powierzchni układu; (B) zwiększenie wydajności aplikacji; (C) zmniejszenie zużycia energii; (D) krótki czas wprowadzenia na rynek. Podejście tradycyjne do projektowania, którym jest zastosowanie procesorów ogólnego przeznaczenia lub układów ASIC, niekoniecznie spełniają zaprezentowane ograniczenia. Jednak dzięki możliwości tworzenia dedykowanych układów z wykorzystaniem tego podejścia, zarówno powierzchnię układu jak i jego wydajność można zoptymalizować [87].

Układy ASIC są to układy VLSI, które mogą zostać utworzone z wykorzystanie układów cyfrowych lub mieszanych. Na podstawie tej technologii projektowej dla ściśle określonych wymagań konstruowane są docelowo systemy wbudowane charakteryzujące się niskim zapotrzebowaniem na energię, a także niewielkimi rozmiarami. Dzięki możliwości konstruowania od podstaw na potrzeby bardziej złożonego systemu elektronicznego mogą zastąpić wiele układów ogólnego przeznaczenia, [144]. Do podstawowych zalet ASIC należą niewątpliwie niezawodność, kompaktowość oraz ogólna trwałość. Jako wadę można wymienić koszty projektu oraz wdrożenia do produkcji, a także brak uniwersalności zastosowania.

ASIP

Ciekawą alternatywą stanowią układy ASIP (ang. *Application Specific Instruction-set Processors*). Układy ASIP są zaprojektowane specjalnie dla określonej aplikacji lub zestawu aplikacji. Dodatkowo pozwalają podłączać prefabry-

kowane koprocesory oraz wstępnie zaprojektowane jednostki funkcjonalne. ASIP określa się jako konfigurowalny akcelerator sprzętowy, który może zostać użyty jako DSP (ang. *Digital Signal Processor*) lub mikroprocesor. Układ ten stanowiący rozszerzenie ASIC może zawierać jeden lub wiele elementów wykonawczych [157]. W układach ASIP można wdrożyć specjalnie zaprojektowane instrukcje, aby poprawić wydajność aplikacji. Jakkolwiek nie tak wydajne jak ASIC to układy te stanowią bardziej elastyczne rozwiązanie dla powiązanych systemów lub ich kolejnych generacji, minimalizując przy tym ryzyko w części programowej [94]. Tworzenie dedykowanego układu odbywa się jednak kosztem jego uniwersalności i niemożliwości zastosowania sprzętu do utworzenia docelowego systemu wymagającego innego zestawu instrukcji. Coraz większa złożoność szczegółowych instrukcji, wykorzystywanych jednostek funkcjonalnych powoduje większe skomplikowanie oraz wydłużenie procesu projektowania.

RISP

Rozwiązaniem przedstawionych powyżej problemów może być zastosowanie układu rekonfigurowalnego w tworzonym systemie. Koncepcja taka zakłada wykonywanie powtarzalnych i czasochłonnych części aplikacji na dynamicznej i adaptacyjnej jednostce funkcyjnej zwanej jednostką rekonfigurowalną. Zastosowanie takiego podejścia pozwala na rozszerzenie architektury zestawu instrukcji podczas działania systemu lub po jego utworzeniu [40]. Reconfigurable Functional Unit stanowią odpowiedni kompromis pomiędzy wydajnością a elastycznością, stwarzając przy tym możliwości dostosowania projektu do zmian rynkowych. Nowym zastosowaniem układów RISP było zaproponowanie rozszerzenie zintegrowanych układów MPSOC (ang. *Multiprocessor System On Chip*) o układy konfigurowalne, docelowo tworząc układy nazywane MRPSOC (ang. *Multi Reconfigurable instruction set Processor System-on-Chip*) gdzie każdy ze zintegrowanych systemów zawiera element lub elementy rekonfigurowalne [156]. Procesor rekonfigurowalny można podzielić na dwa główne podejścia. Pierwsze obejmuje zdefiniowanie interfejsu pomiędzy mikroprocesorem a logiką rekonfigurowalną. Stosując taką koncepcję określić należy wszystkie problemy związane z dwukierunkowym transferem danych pomiędzy logiką rekonfigurowalną a pozostałymi ele-

mentami systemu. Drugie podejście obejmuje samo zaprojektowanie logiki, którą można rekonfigurować [69] [41]. Według pracy [10] RFU może zostać umieszczone w trzech głównych miejscach względem procesora, są to: (A) Dołączony procesor, gdzie logika rekonfigurowalna jest umieszczona na magistrali wejścia/ wyjścia. (B) Koprocesor, logika reprogramowalna jest umieszczana obok procesora, natomiast komunikacja odbywa się przy użyciu protokołu analogicznego jak w przypadku jednostek zmiennoprzecinkowych. (C) Jednostka funkcjonalna, dekoder instrukcji w procesorze ma możliwość korzystania z niej jak w przypadku standardowych jednostek funkcjonalnych.

SOC

Pomimo tego, że rozwój układów scalonych rozpoczął się powoli, to na przestrzeni lat kolejne technologie przyczyniały się do skrócenia czasu utworzenia systemów wbudowanych, zmniejszenia zapotrzebowania na energię oraz zminimalizowania powierzchni docelowego układu [129]. Z czasem rozwiązaniem stało się zastosowanie technologii SOC (ang. *System On Chip*). Projektanci, stosując to podejście, zapewniają zintegrowane rozwiązania trudnych problemów projektowych w sieciach telekomunikacyjnych, multimedialnych i elektroniki użytkowej. Faza projektowania skupia się na integracji komponentów niezbędnych do utworzenia docelowego układu, przy zastosowaniu odpowiednich technologii projektowych i procesowych, a także możliwości wzajemnego łączenia istniejących komponentów. Tworzenie złożonych układów SOC wymaga podejścia modułowego zarówno na etapie projektowania sprzętu, jak i oprogramowania [45]. Układ SOC jest rozpatrywany jako mikroukład elementów, gdzie połączenia sieciowe stanowią abstrakcyjną warstwę komunikacji między komponentami. Docelowy układ musi także spełniać ograniczenia takie jak niezawodność, wydajność i ograniczenia energetyczne. W układach SOC architekturę sieci połączeń można dostosować do klasy aplikacji, która sama w sobie jest celem projektowym [45].

SOC można przedstawić jako układy wysokiej złożoności. Pomimo tego, że jako System On Chip można określić wiele układów to, istnieje kilka elementów, które mogą stać się wyróżnikiem tak zwanego standardowego SOC. Wśród

nich jest na pewno podsystem CPU, mogący zawierać jeden lub wiele procesorów zawierających lokalną pamięć podręczną, które połączone są z wykorzystaniem standardowych magistral lub dedykowanych bloków [3]. Te półprzewodnikowe wysokiej wartości produkty mogą zawierać bloki konstrukcyjne z wielu źródeł, takie jak procesory programowalne ogólnego przeznaczenia, koprocesory, procesory DSP, dedykowane przyśpieszacze sprzętowe, bloki pamięci czy też urządzenia wejścia — wyjścia, układy cyfrowe, analogowe, cyfrowo-analogowe, w tym także układy bezprzewodowe [167]. W systemach SOC spotyka się wysoki stopień integracji układów scalonych w obrębie jednego systemu, który stosowany jest w celu obniżenia kosztów produkcji oraz zmniejszanie wymaganego pola powierzchni dla System On Chip. Konieczność ciągłej minimalizacji stała się powszechna w obrębie systemów wbudowanych. Skalowanie napotkało jednak problemy, co doprowadziło do sytuacji, w której zwiększenie częstotliwości powyżej punktu optymalnego dla projektu ma znaczący i wykładniczy wpływ na zapotrzebowanie na moc, wzrost złożoności mikroarchitektury nie pozwala uzyskać większej ilości ILP (ang. *Integer Linear Programming*) [3]. Projektowanie wysokowydajnych układów SOC wymusza traktowanie mocy układu jako kluczowego czynnika optymalizacyjnego, co jest brane pod uwagę zarówno w systemach zasilanych bateryjnie jak i stosujących tradycyjne połączenia przewodowe. W drugim przypadku zbyt duże zużycie energii wytwarza nadmierną ilość ciepła, co wymusza stosowanie wyszukanych rozwiązań odprowadzających ciepło [167]. Podejście takie generuje nadmierne koszty, zwłaszcza w produkcji wielkoseryjnej. Minimalizacja mocy opóźniała inne rodzaje skalowania w kilku generacjach układów, zatem pojawiający się przy tym problem ciemnego krzemu (ang. *dark silicon problem*) nie jest całkiem nowy. W przeszłości powszechnie stosowano trzy podejścia do wykorzystania dodatkowego obszaru krzemowego w wysokowydajnych układach SOC o niskiej mocy [3]: (A) Dodanie większej ilości pamięci; (B) Minimalizacja wielkości układu; (C) Zmiana równania mocy.

MPSOC

Z czasem potrzeba większej mocy obliczeniowej oraz graficznej wymusiła w SOC zintegrowanie wielu rdzeni obliczeniowych, a także GPU (ang. Graphics Processing Unit). Systemy takie określa się jako MPSOC (ang. *Multi-Processor System-On-Chip*). Można stwierdzić, że tego rodzaju systemy składają się z kilku elementów przetwarzających (ang. *Processing Element, PE*) połączonych ze sobą strukturą połączeń. W układach MPSOC elementy PE są ściśle powiązane z wymaganiami tworzonego systemu [3]. Rozróżnia się dwie architektury tych wieloprocesorowych elementów: heterogeniczne oraz homogeniczne. Do systemów pierwszego rodzaju zaliczyć można systemy składające się z różnego rodzaju PE takich jak procesory, pamięci czy też urządzenia peryferyjne. Poza architekturą sprzętową przyjmuje się, że w tego rodzaju systemach uruchamiane są aplikacje podzielone na zadania, które za pośrednictwem systemu operacyjnego przypisywane są (z wykorzystaniem zestawu sterowników) do realizacji przez sprzęt lub oprogramowanie. Dzięki heterogenicznej platformie sprzętowej można dostosować system do określonych rozwiązań, jest to wykonalne poprzez odpowiednie wykorzystanie infrastruktury procesor-pamięć-szyna oraz opcjonalnej bibliotece akceleratorów i urządzeń peryferyjnych. Jakkolwiek podejście takie pozwala skrócić czas od zaprojektowania do wdrożenia, to jest podejściem mało elastycznym, gdyż w utworzonym systemie nie ma możliwości przekonfigurowania funkcjonalności dedykowanych akceleratorów. Kolejnym rodzajem są systemy o architekturze homogenicznej składające się, z co najmniej kilku instancji takiego samego elementu przetwarzającego. Stworzona tak architektura wykazująca cechy przetwarzania równoległego polega na zwiększeniu liczby zasobów w celu podzielenia czasu wykonania każdego zasobu. Teoretycznie architektura składająca się z N-elementów wykonawczych może zapewnić przyspieszenie co najwyżej N, co w praktyce jest niezwykle trudne lub wręcz niemożliwe do uzyskania. Systemy tego typu będące alternatywą dla architektur heterogenicznych charakteryzują się bardziej elastyczną i skalowalną strukturą w porównaniu z systemami złożonymi z elementów o niejednorodnej strukturze [164]. I chociaż systemy heterogeniczne zapewniają lepsze kompromisy w zakresie wydajności obliczeniowej, czy też ener-

getycznej to mają ograniczoną skalowalność [164]. W literaturze model taki nosi nazwę architektury równoległej. Dla systemów wbudowanych, zwłaszcza wieloprocesorowych istotnym zagadnieniem jest skalowalność. Jest to parametr, który należy uwzględnić już na wczesnych etapach projektowania systemu podczas definiowania ogólnych celów MPSOC. Przyjmuje się, że jako skalowalny można określić system, którego wydajność poprawia się po dodaniu sprzętu, proporcjonalnie do dodanej pojemności [3]. Zgodnie z podejściem homogenicznym w celu przydzielenia wszystkich zasobów w jednostce czasu należy zwiększyć liczbę fizycznych elementów przetwarzających w architekturze. Jako kolejną zaletę takiego podejścia wymienia się możliwość zmniejszenia częstotliwości taktowania, a co za tym idzie także napięcia (co skutkuje minimalizację zapotrzebowania na moc) to z kolei przekłada się na zmniejszenie dynamicznego zapotrzebowania na energię całego układu [3].

Nieustannie zwiększająca się popularność wieloprocesorowych systemów mikroukładowych wymusiła stosowanie bardziej złożonych architektur, co staje się wyzwaniem projektowym pod względem optymalizacyjnym. Sprzeczne wymagania takie jak wydajność aplikacji, zapotrzebowanie na energię, ilość wydzielanego ciepła, powierzchnia systemu, jego waga, czy też zrównoważenie obciążenie muszą być optymalizowane zarówno w fazie projektowania jak i działania systemu. W literaturze można znaleźć wiele propozycji przedstawiających różnorakie podejścia do efektywnego projektowania systemu. Rozważać tutaj można zarówno statyczną jak i dynamiczną optymalizację [164]. W kontekście MPSOC podejście do optymalizacji statycznej określa się jako poprawę parametrów systemu w czasie jego projektowania. W literaturze można spotkać się z różnymi propozycjami rozwiązania tego problemu, począwszy od technik optymalizacji wybranych parametrów (np. wydajności energetycznej) a skończywszy na podejściu kompleksowym w procesie kosyntezy sprzętowo-programowej. Niezależnie od podejścia można skupić się na badaniu różnych metryk w tym także ruchu komunikacyjnego, zajętości pamięci czy aspektach przepustowości stosując wybrane podejścia algorytmiczne w celu skonstruowania docelowego systemu. Chociaż statyczna optymalizacja okazuje się konieczna i niezbędna, to ciągły rozwój systemów wbudowanych i konieczność dostosowania ich do nieustannie rosnących wymagań wy-

musza stosowanie optymalizacji już utworzonego systemu. Biorąc pod uwagę rosnącą niepewność technologii wdrożeniowych, a także scenariuszy aplikacyjnych tego rodzaju systemów; w celu zapewnienia elastycznego podejścia oraz tworzenia niezawodnych systemów, optymalizacja dynamiczna staje się niezbędna [164]. Kolejnym rozpatrywanym aspektem optymalizacji jest podejście scentralizowane, które w zasadzie oferując możliwość dostosowania, stosują scentralizowany podsystem optymalizacji odpowiadający za zarządzanie całym systemem. W podejściu takim przedstawiane są informacje globalne dotyczące systemu, a następnie wykonywana jest optymalizacja każdego elementu wykonawczego [164].

W podrozdziale powyższym ukazany został przegląd najistotniejszych cech systemów wieloprocesorowych. Charakterystyka ta wskazuje, że nie można jasno określić systemu, który odznaczałby się równie dobrą elastycznością, skalowalnością, a także wydajnością obliczeniową czy też energetyczną. Jako rozwiązanie tego problemu zaproponowano, przedstawione w następnym podrozdziale, systemy programowalne [63].

FPGA

Programowalne macierze bramkowe można określić mianem prefabrykowanych urządzeń o strukturze krzemowej, które dając możliwość zaprogramowania mogą przyjąć postać dowolnego układu cyfrowego. W przypadku produkcji mało oraz średnioseryjnej, układy FPGA stają się tańszym rozwiązaniem od układów ASIC, które wymagają dużo większych zasobów czasowych, a także finansowych, aby uzyskać pierwsze urządzenie [63], [6]. W zależności od zmieniających się wymagań do docelowego systemu ten rodzaj układów może być w pełni lub częściowo rekonfigurowalny. W zależności od potrzeb układy te mogą działać niezmiennie w pewnej części, podczas gdy pozostała część układu może zostać zmieniona. Niewątpliwie do największych zalet FPGA zalicza się elastyczność tego rodzaju układów, co stanowi jednocześnie główną przyczynę wad.

Elastyczny charakter układów FPGA powoduje, że zajmują one więcej powierzchni zużywając przy tym więcej energii i są wolniejsze niż ich odpowiedniki ASIC. Wady te wynikają głównie z programowalnego połączenia routingu układów FPGA, które stanowi prawie 90% całkowitej powierzchni układów FPGA.

Pomimo tych wad, układy FPGA stanowią istotną alternatywę dla wdrożenia systemu cyfrowego ze względu na ich krótszy czas wprowadzania na rynek oraz niskie koszty. W celu utworzenia architektur rekonfigurowalnych zaproponowanych zostało wiele technologii. Każda z wykorzystywanych podejść ma różne cechy i znaczący wpływ na programowalną architekturę. Spośród znanych technologii można wymienić: (A) pamięć statyczna; (B) pamięć flash; (C) zabezpieczenie przed przepaleniem.

Ten rodzaj układów składa się z programowalnych bloków logicznych CLB (ang. *Configurable Logic Block*). Każdy blok logiczny jest połączony z pozostałymi z wykorzystaniem przełączników oraz bloków połączeń. Segmentowane połączenia pomiędzy poszczególnymi blokami mogą charakteryzować się różnymi długościami łączy. W układach takich również przełączniki są programowalne. Jednym z istotniejszych elementów tychże bloków są programowalne komórki LUT (ang. *Look-up-Table*). W ciągu ostatnich dziesięcioleci złożoność, a także szybkość układów PLD (ang. *Programmable Logic Devices*) a w szczególności programowalnych macierzy (FPGA) uległa zwielokrotnieniu. Po zadebiutowaniu układów tego typu w połowie lat 80-tych dwudziestego wieku pierwsze układy Xlink posiadały 64 programowalne komórki LUT, podczas gdy współczesne układy tej firmy liczą ponad 663 tysiące komórek w programowalnej macierzy [82, 97, 99, 165]. Pierwsze układy stanowiły stosunkowo proste konstrukcje o nieskomplikowanej strukturze, w trakcie rozwoju nauk technicznych zwłaszcza elektrotechniki i informatyki stawały się coraz bardziej wyszukane oraz o architekturze wykorzystującej pojawiające się nowości techniczne. Oprócz zwiększenia liczby komórek programowalnych w macierzy w nowoczesnych układach programowalnych występuje duża liczba bloków makr, takich jak wbudowane pamięci, bloki DSP, wbudowane procesory, moduły IP, szybkie układy wejścia/wyjścia i synchronizacja zegara. Tego typu układu FPGA są używane w procesie wytwarzania złożonych projektów SOC. Stosując takie podejście istnieje możliwość modelowania także bardzo złożonych projektów z zakresu MPSOC zawierający nawet dziesiątki rdzeni procesorów [71, 97]. Termin „programowalny / rekonfigurowalny" w układach FPGA wskazuje na ich zdolność na wdrożenia nowej funkcji układu po ukończeniu procesu tworzenia. Ta konfigurowalność układu FPGA opiera się na

podstawowej technologii programowania, która może spowodować zmianę w zachowaniu prefabrykowanego układu po jego wytworzeniu [63].

Pierwszą spośród technologii programistycznych wykorzystywanych w układach programowalnych są statyczne komórki pamięci będące podstawowymi komórkami stosowanymi w układach FPGA opartych na SRAM. Większość dostawców komercyjnych wykorzystuje taką technologię programowania. Urządzenia te wykorzystują statyczne komórki pamięci, zapewniając w ten sposób konfigurowalność. Układy realizowane w oparciu o pamięć statyczną najczęściej wykorzystywane są, aby zaprogramować połączenia routingu oraz w celu realizacji konfiguracji bloków CLB [97]. Alternatywą dla tego rodzaju technologii jest zastosowanie układów opartych o flash lub EEPROM. Jedną z głównych zalet zastosowania tej technologii jest jej nieulotność. Zastosowanie pamięci trwałej jest również korzystniejsze pod względem pola powierzchni tworzonego układu. Do wad zaliczyć można na pewno ograniczoną możliwość rekonfiguracji układu [97]. Kolejnym możliwym rozwiązaniem jest zastosowanie tak zwanej technologii antifuse. W porównaniu do przedstawionych wcześniej technik programistycznych rozwiązanie to ma niższą odporność oraz pojemność pasożytniczą. Podejścia takie również ma charakter nieulotny, jednak jako znaczącą wadę można wymienić niemożliwość reprogramowania urządzeń utworzonych w tej technologii [97].

Inną możliwością programowania układów jest wykorzystanie narzędzi CAD służących do graficznego zaprojektowania i zaprogramowania tychże układów. W porównaniu z przedstawionymi w powyższych podrozdziałach układami systemy FPGA mają wiele zalet. Wśród nich wymienić można [149]:

- długą czasową dostępność, która ukazuje się w niezależności klienta od producentów wykorzystywanych komponentów,
- możliwość re-programowania przez klienta,
- relatywnie krótki czas wprowadzenia na rynek objawiający się w możliwości szybkiego opracowania prototypu sprzętu
- wzrost wydajności aplikacji,
- tworzenie szybkich i wydajnych systemów, także dedykowanych, a co za tym idzie efektywnych systemów
- masowo równoległe przetwarzanie,

- zapewnienie obsługi aplikacji czasu rzeczywistego

W przeciwieństwie do układów ASIC zużycie energii w FPGA jest większe, dodatkowo w tego rodzaju układach nie ma możliwości kontroli nad optymalizacją mocy. Ten rodzaj układów jakkolwiek okazuje się efektywny przy projektowaniu, to jednak dotyczy to tylko ich produkcji w małych ilościach. Przy znacznym wzroście wytwarzanych układów występuje także wzrost kosztów ich produkcji, odwrotnie niż w przypadku ASIC [99].

Przeprojektowanie układów ASIC wymaga poprawy niewłaściwych komponentów sprzętowych, co wiąże się niejednokrotnie z wysokimi kosztami utworzenie nowej maski produkcji tego rodzaju układów. Ryzyko związane z koniecznością przeprojektowania w takich przypadkach jest zbyt duże i niejednokrotnie nie do przejęcia, stąd propozycja wykorzystanie układów programowalnych. Rosnąca złożoność układów elektrycznych oraz coraz większe wymagania co do elastyczności, zachowując jednocześnie wysoki poziom niezawodności, wymusza używanie platform konfigurowalnych istniejących, a także projektowanych układów [71]. Od czasu pojawienia się układów FPGA w latach 80-tych dwudziestego wieku do czasów współczesnych układy Xlink oraz IntelFPGA (dawniej Altera) znacznie zyskały na popularności [99, 149].

SOC-FPGA

Technologia SOC polega na połączeniu na jednym układzie wszystkich niezbędnych obwodów elektronicznych w obrębie jednego IC (ang. *Integrated Circuit*). Dzięki elastyczności System On Chip objawiającej się w możliwości integracji różnego typu elektronicznych układów. Technologia ta jest coraz powszechniej stosowana w małych i odznaczających się coraz większą złożonością urządzeniach elektronicznych. Programowalna macierz bramek również jest przykładem zintegrowanego układu scalonego, który ma możliwość fizycznej zmiany już istniejącego systemu. FPGA oprócz możliwości wykorzystania przy projektowaniu wyspecjalizowanych układów znajduje zastosowanie także w możliwości dostosowania mikroprocesorów do własnych indywidualnych potrzeb [115] jednak o ograniczeniach wspomnianych w podrozdziale poprzednim. Rozwiązaniem tego problemu ograniczonej wydajności, a także zbyt dużego zapotrzebowania na ener-

gię jest zastosowanie połączenia układów SOC wraz z FPGA na jednej platformie. W podejściu takim FPGA oraz SOC są łączone z wykorzystaniem urządzenia HPS (ang. *Hard Processor System*). Podejście takie umożliwia zmniejszenie niezbędnej wielkości docelowego układu, a co za tym idzie redukcję niezbędnej mocy oraz kosztów układu. Dodatkowo elastyczność układu objawia poprzez możliwość dowolnego zróżnicowania docelowego układu pomiędzy sprzętem i oprogramowaniem [9]. Każdy z elementów tego rodzaju układów może być zasilany jednocześnie lub niezależnie od drugiego. Dodatkowo istnieje możliwość zmniejszenia częstotliwości lub nawet całkowitego wyłączenia zegarów, aby możliwe było zmniejszenie mocy dynamicznej. SOC-FPGA pozwalają również na niezależne uruchamianie HPS oraz układu programowalnego [9]. Tego typu systemy dzięki integracji dwóch opisanych powyżej rozwiązań stanowią atrakcyjną alternatywę dla systemów wymagających wysokiej wydajności obliczeniowej podzielnej pomiędzy komponenty sprzętowe oraz programowe. Jednym z kluczowych wyzwań w tego typu urządzeniach staje się opracowanie narzędzi zapewniających jednolity interfejs pomiędzy dwoma domenami, aby docelowy system był jak najefektywniej zrealizowany. Efektywnym rozwiązaniem tego problemu może być synteza wysokiego poziomu HLS (ang. *High-Level Synthesis*) [9, 152].

A.2. Organizacja systemów wieloprocesorowych

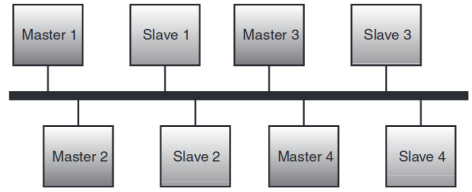
W przedstawionych w powyższych podrozdziałach opisach uwidacznia się trend do wykorzystania systemów zawierających wiele elementów przetwarzających zintegrowanych w jednym systemie. Wieloprocesorowe systemy osadzone mogą zostać zorganizowane w oparciu o różne architektury. Podrozdział niniejszy prezentuje najpopularniejsze spośród nich wraz z przedstawieniem ich najistotniejszych cech. Wśród architektur wieloprocesorowych systemów można wymienić: (A) Architektura magistrali; (B) Przełącznica krzyżowa; (C) Przełącznica krzyżowa z buforami; (D) Sieć omega; (E) Sieć o wybranej strukturze; (F) Sieć NOC.

Architektura magistrali

Stanowi najbardziej podstawową architekturę połączenia elementów systemu wbudowanego zorganizowanych w architekturę SOC. W tym obszarze jest jednym z najpopularniejszych podejść do łączenia nie tylko wykonawczych, ale i wejściowo/wyjściowych elementów układu. Architektura magistrali służy do łączenia niezbędnych składowych systemu wykorzystujących pojedynczy współdzielony kanał komunikacyjny. Konstrukcja ta może zostać zorganizowana jako architektura szeregowa (wykorzystująca pojedyncze połączenie) lub równoległa (korzystająca z wielu równoległych połączeń). Głównym założeniem w procesie komunikacji z wykorzystaniem tej architektury jest koncepcja przyjęcia urządzenia nadrzędnego (master) i podrzędnego (slave). Urządzenie nadrzędne inicjuje proces komunikacji z dowolnym urządzeniem lub urządzeniami podrzędnymi, [32]. Magistralę komunikacyjną można określić jako zespół lini oraz układów przełączających, służących do przesyłania sygnałów między połączonymi urządzeniami w systemach mikroprocesorowych, złożony z trzech współdziałających linii komunikacyjnych, a mianowicie: (A) adresowej; (B) danych; (C) sterującej. Magistralę charakteryzują następujące parametry: szerokość, szybkość, metoda arbitrażu, koordynacja czasowa. W przypadku gdy do magistrali przyłączonych jest wiele układów (co najmniej dwa), musi istnieć sterownik zwany arbitrem. Arbiter służy do przydzielenia czasu magistrali poszczególnym układom zgłaszającym zapotrzebowanie na przesyłanie danych. Zastosowanie arbitrażu jest koniecznością w sytuacji, w której więcej niż jeden moduł w danym momencie wymaga połączenia z magistralą. Metody stosowane do arbitrażu są dzielone na scentralizowane oraz rozproszone. W pierwszym przypadku jest używany sterownik magistrali. W drugim przypadku – poszczególne moduły wyposażone są we współpracujące ze sobą układy logiczne. Obie metody mają na celu wyznaczenie urządzenia nadrzędnego odpowiedzialnego za inicjowania transferu danych z urządzeniem podrzędnym, [32]. Architektura magistrali może zostać zaimplementowana na kilka sposobów, [32]: (A) pojedyncza (ang. *Single Bus*); (B) hierarchiczna (ang. *Hierarchical Bus*); (C) podzielona (ang. *Split Bus*); (D) w pełni połączona (ang. *Full Bus Crossbar*); (E) połączona częściowo (ang. *Partial Bus Crossbar*); (F) pierścieniowa (ang. *Ring Bus*).

W pracach [98, 102] zaprezentowano także zwielokrotnienie liczby współpracujących szyn, aby zwiększyć wydajność w ten sposób tworzonych systemów oraz zapewnić wyższy poziom ich niezawodności. Kolejnym istotnym zagadnieniem jest koordynacja czasowa, która odnosi się do sposobu zharmonizowania zdarzeń na magistrali. Wyróżnić można synchroniczną oraz asynchroniczną. Pierwszy rodzaj koordynacji polega na tym, że zdarzenia na magistrali wyznaczane są przez zegar. Pojedyncza transmisja nazywana jest cyklem zegarowym lub cyklem magistrali. Wszystkie zdarzenia rozpoczynają się równocześnie z cyklem zegarowym. Natomiast koordynacja asynchroniczna polega na tym, że zdarzenia na magistrali są zależne od zdarzenia poprzedniego. Zapewnia to większą elastyczność, ale jest trudniejsze do wdrożenia.

Najistotniejszą wadą magistrali jest zwiększenie opóźnień propagacji wprost proporcjonalne do liczby urządzeń dołączanych do magistrali. W przypadku gdy transferowane szyną dane są bliskie pojemności magistrali, staje się ona wąskim gardłem, wprowadzając znaczne opóźnienia, [32].

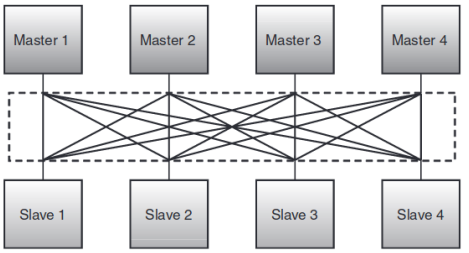


Rysunek A.2: Przykładowa architektura magistrali. Źródło : [32]

Przełącznica krzyżowa

Kolejne podejście w projektowaniu architektur MPSOC, mające na celu poradzenie sobie z ograniczeniami wydajności magistral, to przełącznica krzyżowa zwana również macierzą przełączającą (ang. *crossbar switch, crossbar matrix*). Przeprowadzone badania wykazały znaczący wzrost przepustowości danych w porównaniu z architekturą magistrali, czy też magistrali hierarchicznej [32]. Przedstawiona tutaj architektura jest przykładem rozwiązania problemu wąskiego gardła przy dostępie do wspólnej pamięci. Crossbar Switch jest przykładem sieci w pełni połączonych (Fully Connected Networks) stanowiąc przy tym sieć nieblokującą.

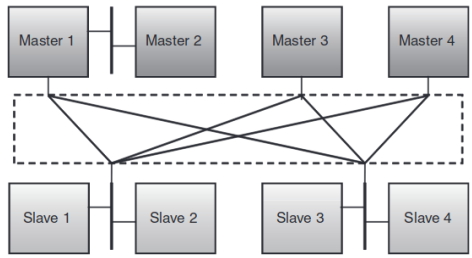
Jej zaletą jest jednoczesny dostęp wielu procesorów do wielu modułów pamięci. Dostęp do modułu jest możliwy w przypadku, gdy nie jest on zajęty przez inny procesor. Pomimo tego, że architektura ta zapewnia zrównoleglenie w wysokim stopniu, to występuje jednak nadmiarowość stosowanych połączeń. Element ten stanowiący równocześnie zaletę jak i wadę tego rodzaju architektury jest przyczyną użycia dodatkowego obszaru zwiększającego zarazem powierzchnię systemu, co wpływa na zwiększone zapotrzebowanie energetyczne. Rozwiązaniem tego problemu jest stosowanie przełącznicy krzyżowej nie w pełni połączonej, [2, 32]. Oprócz wykorzystania kanałów fizycznych istnieje możliwość zastosowania kanałów wirtualnych. Podobnie jak w przypadku kanałów komunikacyjnych fizycznych tak też w przypadku kanałów wirutualnych, dla każdego połączenia, stosuje się metody arbitrażu odbywające się na zasadach podobnych jak w przypadku magistral. W celu rozwiązania problemów komunikacyjnych stosuje się podejście z buforami. Modyfikacja taka polega na dodaniu buforów dla każdego modułu pamięci. Można w takim przypadku wyróżnić dwie strategie pracy: przypisanie natychmiastowe lub oczekiwanie aż moduł będzie wolny, [88]. W przypadku Crossbar Switch bez buforów istnieje konieczność pracy synchronicznej. Jeżeli istnieje konieczność obsługi pakietów o zmiennej długości, muszą one być segmentowane. Przy zastosowaniu modyfikacji polegającej na dodaniu buforów cały układ może pracować jako asynchroniczny, [134].



Rysunek A.3: Przełącznica krzyżowa w pełni połączona. Źródło: [32]

Sieci wielostopniowe

Przykładem rozwiązania problemów związanych z zastosowaniem technologii Crossbar Switch są sieci wielostopniowe (ang. *Multistage Networks*). Stanowią



Rysunek A.4: Przedstawienie przełącznicy krzyżowej częściowo połączonej. Źródło: [32]

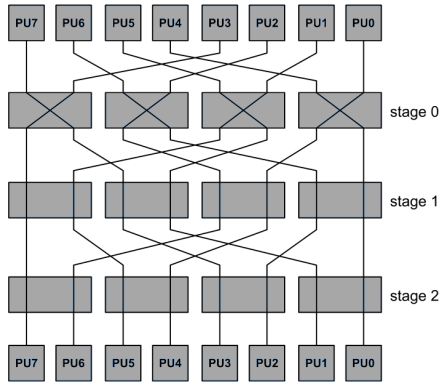
one kompromis pomiędzy kosztami a wydajnością. Wykazują większą wydajność niż konstrukcje ze wspólną magistralą i mniejszy koszt niż rozwiązania z pełną przełącznicą krzyżową. Sieci wielostopniowe można scharakteryzować z wykorzystaniem trzech parametrów, [148]: (A) topologia; (B) rodzaj komunikacji; (C) struktura sterowania. Układy zorganizowane w architekturze sieci wielostopniowych sklasyfikować można jako jedną z taksonomi, [140]:

- Pojedynczy strumień instrukcji (ang. *Single instruction stream, SI*),
- Strumień wielu instrukcji (ang. *Multi instruction stream, MI*),
- Pojedynczy strumień danych (ang. *Single data stream, SD*),
- Wiele strumieni danych (ang. *Multi data stream, MD*).

Jednym z najlepszych przykładów tego typu rozwiązania jest sieć omega. Jest ona powszechnie stosowana w równoległych systemach komputerowych należących do kategorii sieci blokujących, [148]. Elementy przełączające w sieci sterują ruchem sieciowym w zależności od żądania emisji poszczególnych elementów. Każdy element przetwarzający może uzyskać dostęp do własnej pamięci wewnętrznej lub za pomocą sieci [148]. W sieci omega można obsługiwać wiele połączeń między wejściami i wyjściami. Permutacja nazywa się dopuszczalną, jeżeli można ustanowić ścieżki wolne od konfliktów, po jednej dla każdej pary (wejścia / wyjścia) [79]. Na rysunku A.5 przedstawiona została przykładowa architektura dla wieloprocesorowej sieci omega.

Interesującym zagadnieniem realizacji sieci wielostopniowych jest ich topologia. Z wykorzystaniem tego typu konstrukcji systemy mogą być zorganizowane jako sieć typu Flip, sieć pośrednią zorganizowaną w architekturze kostki, a także

inne pochodne architektury, [137].



Rysunek A.5: Sieć omega zawierająca 8 procesorów. Źródło : [148]

Sieć NOC

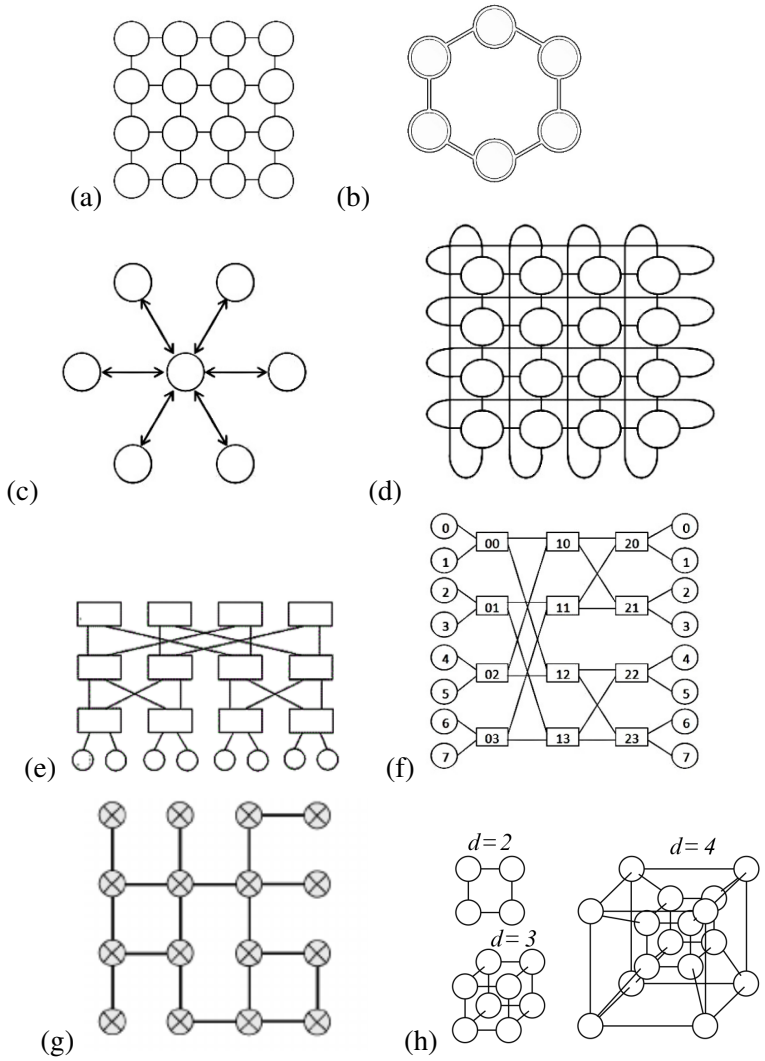
Komunikacja pomiędzy dwoma elementami wykonawczymi oraz elementem wykonaczym i pamięcią może zostać zrealizowana sprzętowo na kilka sposobów: A) poprzez wieloportową (ang. *multiported*) lub multipleksowaną pamięć dzieloną, B) przełącznicę krzyżową (ang. *crossbar switch*), współdzieloną magistralę (ang. *shared bus*), C) sieć o jednej z wielu topologii (gwiazdy, pierścienia, drzewa, hiperkostki i innych). Jednym z istotniejszych dla rozwoju branży systemów wbudowanych rozwiązań było zaproponowanie jednoukładowej sieci NOC, opartej na modelu sieci komputerowych OSI [38, 151].

Według analizy przeprowadzonej przez autorów pracy [38] nie ma możliwości jednoznacznego określenia początków sieci Noc. Wskazuje się wiele opracowań proponujących zastosowanie nowego paradygmatu tworzenia architektur systemów organizacji systemów osadzonych. Komunikacja w systemach SOC odbywa się z wykorzystaniem fizycznych łączy na układzie. W stosunku do opóźnień na bramkach logicznych łącza komunikacyjne generują znaczne opóźnienia. Z powyższego faktu wynika, że przy złożonych SOC wpływać to może na nierównomierne obciążenie całego systemu, a co za tym idzie nieefektywne wykorzystanie układu. Przedstawione czynniki bezpośrednio wpłynęły na konieczność opracowania nowej architektury pozbawionej wyżej wymienionych niedogodno-

ści [38]. Głównym celem projektowania zorientowanego na komunikację układu jest zwiększenie efektywności i wydajności układu poprzez obsługę wzrastającego czynnika zrównoleglania układu. Podejście takie jest możliwe dzięki wspieraniu integracji w układach SOC poszczególnych elementów systemu z wykorzystaniem architektury mikrosieciowych połączeń [5, 38]. Wśród podstawowych komponentów sieci NOC można wymienić takie jak [151]: (A) Adaptery sieciowe, służące jako interfejs, za pomocą którego łączone są bloki IP (B) Węzły routingu, służące do obsługi ruchu sieciowego zgodnie z przyjętymi protokołami (C) Łącza, służące do łączenia węzłów, składać mogą się z jednego lub wielu kanałów logicznych, lub fizycznych W układach SOC wzorce komunikacji mogą być statycznie analizowane do konkretnego rozwiązania, natomiast w przypadku architektur sieci NOC są one dostosowane do konkretnego zachowania aplikacji. Oprócz wymagań projektowych każdy z układów może mieć różne ograniczenia projektowe. Kluczowym zagadnieniem w obszarze systemów wbudowanych jest zużycie energii, dodatkowym, lecz niezwykle istotnym jest fakt, że większość tego typu układów stanowią systemy czasu rzeczywistego oraz zorientowane na jakość usług. Wszystkie z przedstawionych kwestii powinny rozpatrywać jak najefektywniejsze sposoby łączenia komponentów docelowego systemu [86]. Do najważniejszych etapów projektowania sieci NOC należą projektowanie topologii lub struktury sieci oraz ustalenie pozostałych parametrów projektowych jak częstotliwość działania, lub szerokość łącza. Układy dla prezentowanej architektury mogą mieć charakter hetero lub homogeniczny, przy czym większość tego typu systemów jest niejednorodna [86, 145]. Układy zorganizowane w architekturze tego rodzaju sieci jednoukładowych mogą być zrealizowane jako dwu- lub trzy-wymiarowe [44, 91, 145]. Do najpopularniejszych technologii dla sieci jednoukładowych można zaliczyć [38, 91, 145]: (A) Topologia pierścienia, stanowi jedną z najpopularniejszych topologii. W topologii tego typu każdy węzeł połączony jest z dwoma sąsiednimi. Stopień każdego węzła wynosi dwa, natomiast jako wadę można przedstawić rosnącą średnicę wraz z dodawaniem kolejnych węzłów. Dodatkowo awaria jednego z węzłów powoduje unieruchomienie całej sieci; (B) Topologia gwiazdy, polega na połączeniu wszystkich węzłów do węzła centralnego. W przypadku takim wadą może się okazać awaria węzła centralnego. Dodatkowo węzeł centralny wraz ze

wzrostem złożoności sieci staje się wąskim gardłem, natomiast jako zaletę można wymienić prostotę jej konstrukcji; (C) Topologia mesh, stanowi połączenie ze sobą wszystkich sąsiadujących węzłów. Niewątpliwą zaletą jest dobry współczynnik skalowalności oraz różnorodności ścieżek. Wadą również jak w przypadku pierścienia okazać może się średnica rosnąca wraz ze złożonością sieci; (D) Topologia torus, jest modyfikacją topologii mesh poprzez dodanie połączeń do węzłów krańcowych i w ten sposób zniwelowany został problem rosnącej średnicy docelowego systemu. Jako wadę można wymienić wzrost opóźnienia, który spowodowany jest dodatkowym dodaniem połączeń; (E) Topologia grubego drzewa wykorzystuje routery pośrednie (pełniące rolę routerów przekierowujących) oraz routery opuszczające, za pomocą których połączone są elementy końcowe sieci. Zaletą jest różnorodność ścieżek, jako wadę wymienić można konieczność używania kilku rodzajów routerów dla podłączenia elementów docelowych co wpływa na maksymalizację wykorzystania sprzętu. (F) Topologia motyla, główną jej wadą stanowi brak różnorodności ścieżek, kolejna wada objawia się w długiej średnicy sieci, co wpływa na zwiększone zużycie energii; (G) Topologia mieszana występują jako modyfikacje dwóch lub większej liczby topologii docelowej, aby wykorzystując wady oraz zalety topologii podstawowych zmaksymalizować efektywność docelowego systemu; (H) Topologie wielowymiarowe, hipersześciany, stanowią siatkę toroidalną na zasadzie topologii torusa (połączonych wzajemnie), gdzie wielowymiarowość można rozumieć poprzez zastosowanie jako węzła całej struktury sieci; (I) Topologie nieregularne, rozumieć można jako sieć dedykowaną tworzoną poprzez usunięcie jednego lub kilku węzłów (z topologii regularnej), tak aby docelowy system był jak najbardziej efektywny dla rozpatrywanego rozwiązania. Wyżej wymienione topologie omawianej sieci jednoukładowej zostały zaprezentowane na rysunkach A.6 oraz A.7. Jak wcześniej zostało zaznaczone, Noc oparte jest na modelu OSI i wyróżnić można tutaj: (A) warstwę fizyczną w połączeniu z warstwą sieciową/łącza danych określaną jako flit; (B) warstwa sesji/transportu, w której przysyłane są pakiety; (C) warstwa aplikacji/prezentacji, w której operacje wykonywane są na wiadomościach.

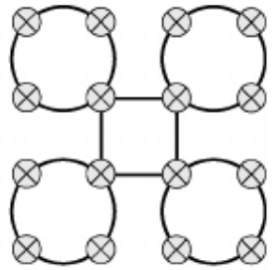
Istotnym zagadnieniem w sieciach tego typu jest routing. Od czasów zaproponowania koncepcji tego rodzaju sieci jednoukładowych do obecnych trwają prace



Rysunek A.6: Topologie sieci NOC cz. 1:) Mesh, b) Ring, c) Star, d) Torus, e) Fat Tree, f) Butterfly, g) Irregular, h) Toroidal. Źródło: [91, 145]

nad efektywnymi metodami komunikacji oraz konstrukcjami routerów proponującymi lepsze wykorzystanie elementów przetwarzających.

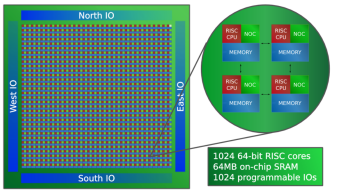
Zgodnie z architekturą sieci NOC utworzonych zostało wiele fizycznych układów [123, 125, 166] wytwarzanych jako MPSOC czy też FPGA. Zorganizowane w ten sposób układy wykorzystywane są zazwyczaj w obszarze głębokiego uczenia, autonomicznych pojazdów jak i autonomicznych obiektów latających oraz



(i)

Rysunek A.7: Topologie sieci NOC cz. 2, i) Mixed. Źródło: [91, 145]

wszelkiego rodzaju systemach wymagających zwiększenia wydajności przetwarzania w celu uzyskania pełnego potencjału sprzętu. Przykładem takiego układu może być 1024 rdzeniowy projekt procesora składający się z rdzeni RISC zrealizowanych jako system MIMD. Układ ten, przedstawiony na rysunku 4.7, charakteryzuje się dużą efektywnością przetwarzania sięgającą 75 GFLOPS/Wat, a także redukcją kosztów i znacznymi oszczędnościami energii [123].



Rysunek A.8: Układ Ephipany V. Źródło: [123]

Wśród zalet NOC wymienić można to, że dołączenie jednostek ma charakter lokalny, nie pogarszając parametrów elektronicznych całego układu. Ścieżki w sieci NOC są krótkie i łatwo nimi sterować za pomocą sygnałów zegarowych. Routing w sieci może być zdecentralizowany, można wykorzystywać wirtualne kanały komunikacyjne w zależności od architektury docelowego systemu. Ten sam typ routera może być stosowany w sieciach dowolnych rozmiarów i dla dowolnych aplikacji, nowością w stosunku do układów o innych architekturach jest fakt, że przepustowość rośnie wraz z rozbudową systemu. Jakkolwiek zalety i wady zależą od wybranej topologii, to transmisje mogą powodować kolizje w sieci NOC, tworząc zatory spowalniające komunikację [72].

Bibliografia

1. Abd Elraheem, A. K., Shikhin, V. A. & Kouzalis, A. *Optimization problem statement for power generation management and control in multi-agent Microid* w *2019 III International Conference on Control in Technical Systems (CTS)* (2019), 176–179.
2. Adeli, H. & Vishnubhotla, P. R. w *Parallel processing in computational mechanics* 1–20 (CRC Press, 2020).
3. Aitken, R., Flautner, K. & Goodacre, J. w *Multiprocessor System-on-Chip* 223–239 (Springer, 2011).
4. Albers, S. & Hellwig, M. Online makespan minimization with parallel schedules. *Algorithmica* **78,** 492–520 (2017).
5. Alimi, I. A. *i in.* Network-on-chip topologies: Potentials, technical challenges, recent advances and research direction. *Network-on-Chip-Architecture, Optimization, and Design Explorations* (2021).
6. Alkhafaji, F. S., Hasan, W. Z., Isa, M. & Sulaiman, N. Robotic controller: ASIC versus FPGA—A review. *J. Comput. Theor. Nanosci* **15,** 1–25 (2018).
7. Anderson, A. R., Morris, G. R. & Abed, K. H. *Achieving true parallelism on a high performance heterogeneous computer via a threaded programming model* (IEEE, 2011).
8. Arpaci-Dusseau, R. H. & Arpaci-Dusseau, A. C. Scheduling: The multi-level feedback queue. *Operating Systems: Three Easy Pieces* (2014).

9. *Arria V Device Handbook: Volume 1: Device Interfaces and Integration.* spraw. tech. https://www.mouser.com/pdfDocs/av_5v2.pdf (Altera, 2020). (2022).

10. Babu, P. & Parthasarathy, E. Reconfigurable FPGA architectures: A survey and applications. *Journal of The Institution of Engineers (India): Series B* **102,** 143–156 (2021).

11. Bansal, N. & Pruhs, K. The geometry of scheduling. *SIAM Journal on Computing* **43,** 1684–1698 (2014).

12. Bąk, I., Markowicz, I., Mojsiewicz, M. & Wawrzyniak, K. *Wzory i tablice: metody statystyczne i ekonometryczne* (CeDeWu, 2019).

13. Behnamian, J. Survey on fuzzy shop scheduling. *Fuzzy Optimization and Decision Making* **15,** 331–366 (2016).

14. Behnamian, J. & Fatemi Ghomi, S. A survey of multi-factory scheduling. *Journal of Intelligent Manufacturing* **27,** 231–249 (2016).

15. Blazewicz, J., Drabowski, M. & Weglarz, J. Scheduling multiprocessor tasks to minimize schedule length. *IEEE Transactions on Computers,* 389–393 (1986).

16. Blazewicz, J. *i in.* Communication Delays and Multiprocessor Tasks. *Handbook on Scheduling: From Theory to Practice,* 199–241 (2019).

17. Błażewicz, J., Drozdowski, M. & Ecker, K. w *Handbook on Parallel and Distributed Processing* 263–341 (Springer, 2000).

18. Błażewicz, J., Drozdowski, M., Formanowicz, P., Kubiak, W. & Schmidt, G. Scheduling preemptable tasks on parallel processors with limited availability. *Parallel Computing* **26,** 1195–1211 (2000).

19. Błażewicz, J., Ecker, K. H., Pesch, E., Schmidt, G. & Weglarz, J. *Handbook on Scheduling: From Theory to Applications* (Springer Science & Business Media, 2007).

20. Błażewicz, J. & Liu, Z. Scheduling multiprocessor tasks with chain constraints. *European Journal of Operational Research* **94,** 231–241 (1996).

21. Boyar, J., Favrholdt, L. M., Kudahl, C., Larsen, K. S. & Mikkelsen, J. W. Online algorithms with advice: A survey. *ACM Computing Surveys (CSUR)* **50,** 1–34 (2017).

22. Bozejko, W., Hejducki, Z. & Wodecki, M. Flowshop scheduling of construction processes with uncertain parameters. *Archives of Civil and Mechanical Engineering* **19,** 194–204 (2019).

23. Bożejko, W., Rajba, P. & Wodecki, M. Stable scheduling of single machine with probabilistic parameters. *Bulletin of the Polish Academy of Sciences. Technical Sciences* **65,** 219–231 (2017).

24. Bożejko, W., Hejducki, Z., Rajba, P. & Wodecki, M. Algorytmy Mmemetyczne dla pewnego problemu potokowego w budownictwie. *Oficyna Wydawnicza Polskiego Towarzystwa Zarządzania Produkcją,* 251–262 (2012).

25. Bożejko, W., Rajba, P. & Wodecki, M. *Scheduling problem with uncertain parameters in Just in Time system* w *International Conference on Artificial Intelligence and Soft Computing* (2014), 456–467.

26. Bożejko, W. & Pempera, J. *Optymalizacja dyskretna w informatyce, automatyce i robotyce /* (Oficyna Wydaw. Politech. Wrocławskiej, Wrocław : 2012).

27. Burggräf, P., Wagner, J., Koke, B. & Steinberg, F. Approaches for the prediction of lead times in an engineer to order environment—A systematic review. *IEEE Access* **8,** 142434–142445 (2020).

28. Campelo, F. *EC-Bestiary. Evolutionary Computation Bestiary* spraw. tech. 26 (Aston University, list. 2019). `http://fcampelo.github.io/EC-Bestiary/`.

29. Campelo, F. & Aranha, C. *EC Bestiary: A bestiary of evolutionary, swarm and other metaphor-based algorithms* wer. v2.0.1. Czer. 2018. `https://doi.org/10.5281/zenodo.1293352` (2022).

30. Caplan, J., Al-Bayati, Z., Zeng, H. & Meyer, B. H. Mapping and scheduling mixed-criticality systems with on-demand redundancy. *IEEE Transactions on Computers* **67,** 582–588 (2017).

31. Chaari, T., Chaabane, S., Aissani, N. & Trentesaux, D. *Scheduling under uncertainty: Survey and research directions* w *2014 International Conference on Advanced Logistics and Transport (ICALT)* (2014), 229–234.

32. Charles, S. & Mishra, P. A survey of network-on-chip security attacks and countermeasures. *ACM Computing Surveys (CSUR)* **54,** 1–36 (2021).

33. Chen, B., Van Vliet, A. & Woeginger, G. J. *An optimal algorithm for preemptive on-line scheduling* w *European Symposium on Algorithms* (1994), 300–306.

34. Chen, X. *Selected problems of online scheduling on parallel machines* prac. dokt. (Rozprawa Doktorska, Politechnika Poznańska, Poznań, 2014).

35. Chin, M. K., Kek, S. L., Sim, S. Y. & Seow, T. W. Probabilistic Completion Time in Project Scheduling. *International Journal of Engineering Research & Science* **3,** 44–48 (2017).

36. Chinnery, D. & Keutzer, K. *Closing the gap between ASIC & custom: tools and techniques for high-performance ASIC design* (Springer Science & Business Media, 2002).

37. Christensen, H. I., Khan, A., Pokutta, S. & Tetali, P. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review* **24,** 63–79 (2017).

38. Cilardo, A. & Fusella, E. Design automation for application-specific on-chip interconnects: A survey. *Integration* **52,** 102–121 (2016).

39. Cormen, T. H. [a. *Wprowadzenie do algorytmów* pol. online. podręczniki akademickie.

40. D'Arnese, E., Conficconi, D., Santambrogio, M. D. & Sciuto, D. w *Emerging Computing: From Devices to Systems: Looking Beyond Moore and Von Neumann* 435–456 (Springer, 2022).

41. Dao, N., Attwood, A., Healy, B. & Koch, D. *Flexbex: A risc-v with a reconfigurable instruction extension* w *2020 International Conference on Field-Programmable Technology (ICFPT)* (2020), 190–195.

42. Davis, R. I. & Burns, A. A survey of hard real-time scheduling for multiprocessor systems. *ACM computing surveys (CSUR)* **43,** 1–44 (2011).

43. Davis, R. I. & Cucu-Grosjean, L. A survey of probabilistic timing analysis techniques for real-time systems. *LITES: Leibniz Transactions on Embedded Systems,* 1–60 (2019).

44. Day, K. & Al-Towaiq, M. H. *A Parallel Gauss-Seidel Algorithm on a 3D Torus Network-on-Chip Architecture* w *2015 Ninth International Workshop on Interconnection Network Architectures: On-Chip, Multi-Chip* (2015), 13–16.

45. De Micheli, G. & Benini, L. Networks on chips: 15 years later. *Computer* **50,** 10–11 (2017).

46. Della Vedova, M. L., Tessera, D. & Calzarossa, M. C. *Probabilistic provisioning and scheduling in uncertain Cloud environments* w *2016 IEEE Symposium on Computers and Communication (ISCC)* (2016), 797–803.

47. Delorme, M., Iori, M. & Martello, S. Bin packing and cutting stock problems: Mathematical models and exact algorithms. *European Journal of Operational Research* **255,** 1–20 (2016).

48. Dick, R. P., Rhodes, D. L. & Wolf, W. *TGFF: task graphs for free* w *Proceedings of the Sixth International Workshop on Hardware/Software Codesign.(CODES/CASHE'98)* (1998), 97–101.

49. Dietrich, A., Ott, C. & Albu-Schäffer, A. An overview of null space projections for redundant, torque-controlled robots. *The International Journal of Robotics Research* **34,** 1385–1400 (2015).

50. Dorota, D. Divisibility attribute tasks in the process of scheduling. *Elektronika: konstrukcje, technologie, zastosowania* **59** (2018).

51. Dorota, D. *Multiprocessor Tasks Scheduling. Fuzzy Logic Approach* w *New Advances in Dependability of Networks and Systems* (red. Zamojski, W., Mazurkiewicz, J., Sugier, J., Walkowiak, T. & Kacprzyk, J.) (Springer International Publishing, Cham, 2022), 50–62.

52. Dorota, D. *Scheduling Tasks in a System with a Higher Level of Dependability* w *International Conference on Dependability and Complex Systems* (2019), 143–153.

53. Dorota, D. w *Advances in Dependability Engineering of Complex Systems* 131–140 (Springer, 2017).

54. Dorota, D. *Scheduling Tasks with Uncertain Times of Duration* w *International Conference on Dependability and Complex Systems* (2020), 197–209.

55. Dorota, D. Szeregowanie online zadań wieloprocesorowych. *XXII KRAJOWA KONFERENCJA AUTOMATYZACJI PROCESÓW DYSKRETNYCH*.

56. Drozdowski, M. New applications of the Muntz and Coffman algorithm. *Journal of Scheduling* **4,** 209–223 (2001).

57. Drozdowski, M. On the complexity of multiprocessor task scheduling. *Bulletin of the Polish Academy of Sciences Technical Sciences* **43** (1995).

58. Drozdowski, M. *Scheduling for parallel processing* (Springer, London, 2009).

59. Drozdowski, M. Scheduling multiprocessor tasks—an overview. *European Journal of Operational Research* **94,** 215–230 (1996).

60. Drozdowski, M. *Selected problems of scheduling tasks in multiprocessor computer systems* (Politechnika Poznańska, Poznań, 1997).

61. El Tanab, M. & Hamouda, W. Resource allocation for underlay cognitive radio networks: A survey. *IEEE Communications Surveys & Tutorials* **19,** 1249–1276 (2016).

62. Esposito, C., Ficco, M., Palmieri, F. & Castiglione, A. Smart cloud storage service selection based on fuzzy logic, theory of evidence and game theory. *IEEE Transactions on computers* **65,** 2348–2362 (2015).

63. Farooq, U., Marrakchi, Z. & Mehrez, H. w *Tree-based heterogeneous FPGA architectures* 7–48 (Springer, 2012).

64. Fazel Zarandi, M. H., Sadat Asl, A. A., Sotudian, S. & Castillo, O. A state of the art review of intelligent scheduling. *Artificial Intelligence Review* **53,** 501–593 (2020).

65. Fazel Zarandi, M. H., Sadat Asl, A. A., Sotudian, S. & Castillo, O. A state of the art review of intelligent scheduling. *Artificial Intelligence Review* **53,** 501–593 (2020).

66. Ficoń, K. Wykorzystanie teorii kolejek w procesie decyzyjnym przedsiębiorstwa transportowego. *Systemy Logistyczne Wojsk,* 65–78 (2019).

67. Filipowicz, B. *Modele stochastyczne w badaniach operacyjnych: analiza i synteza systemów obsługi i sieci kolejkowych* (Wydawnictwa Naukowo-Techniczne, 1996).

68. Gaj, P. Wybrane zagadnienia projektowania systemów informatyki przemysłowej. *Studia Informatica* **37,** 1–400 (2016).

69. Galuzzi, C. & Bertels, K. The instruction-set extension problem: A survey. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* **4,** 1–28 (2011).

70. Godyń, I. Analiza i prognoza wodochłonności sektorów gospodarki z zastosowaniem wnioskowania rozmytego. *Czasopismo Techniczne. Środowisko* **104,** 77–98 (2007).

71. Göhringer, D., Hübner, M. & Becker, J. w *Multiprocessor System-on-Chip* 127–151 (Springer, 2011).

72. Gomez-Rodriguez, J. R. *i in.* A survey of software-defined networks-on-chip: Motivations, challenges and opportunities. *micromachines* **12,** 183 (2021).

73. González-Neira, E., Montoya-Torres, J. & Barrera, D. Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations* **8,** 399–426 (2017).

74. Graham, R. L. Bounds for certain multiprocessing anomalies. *Bell system technical journal* **45,** 1563–1581 (1966).

75. Graham, R. L., Lawler, E. L., Lenstra, J. K. & Kan, A. R. w *Annals of discrete mathematics* 287–326 (Elsevier, 1979).

76. Gubbi, J., Buyya, R., Marusic, S. & Palaniswami, M. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future generation computer systems* **29,** 1645–1660 (2013).

77. Gupta, A., Im, S., Krishnaswamy, R., Moseley, B. & Pruhs, K. *Scheduling heterogeneous processors isn't as easy as you think* w *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete algorithms* (2012), 1242–1253.

78. Gupta, D., Sharma, S. & Aggarwal, S. Bi-objective scheduling on parallel machines with uncertain processing time. *Advances in Applied Science Research* **3,** 1020–1026 (2012).

79. Gupta, S. & Pahuja, G. L. SEGIN-Minus: A new approach to design reliable and fault-tolerant MIN. *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)* **13,** 370–380 (2020).

80. Harren, R., Jansen, K., Prädel, L. & Van Stee, R. A (5/3+ ε)-approximation for strip packing. *Computational Geometry* **47,** 248–267 (2014).

81. Hartmann, S. & Briskorn, D. An updated survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of operational research* **297,** 1–14 (2022).

82. Hsiung, P.-A., Santambrogio, M. D. & Huang, C.-H. *Reconfigurable system design and verification* (CRC Press, 2018).

83. Ibrahim, H. & Salih, M. H. Design and Implementation of Embedded True Parallelism Jammer System using FPGA-SoC for Low Design Complexity. *ARPN Journal of Engineering and Appllied Sciences* **13,** 9410–9420 (2018).

84. Im, S. *Online scheduling algorithms for average flow time and its variants* prac. dokt. (University of Illinois at Urbana-Champaign, 2012).

85. Iori, M., De Lima, V. L., Martello, S., Miyazawa, F. K. & Monaci, M. Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research* **289,** 399–415 (2021).

86. Jang, H., Han, K., Lee, S., Lee, J.-J. & Lee, W. MMNoC: Embedding memory management units into network-on-chip for lightweight embedded systems. *IEEE Access* **7,** 80011–80019 (2019).

87. Jerraya, A. & Bacivarov, I. w *EDA for IC System Design, Verification, and Testing* 6–1 (CRC Press, 2018).

88. Jiang, N. *i in. A detailed and flexible cycle-accurate network-on-chip simulator* w *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)* (2013), 86–96.

89. Jiang, X., Lee, K. & Pinedo, M. L. Ideal schedules in parallel machine settings. *European Journal of Operational Research* **290,** 422–434 (2021).

90. Kacprzyk, J. *Zbiory rozmyte w analizie systemowej* (Państ. Wydaw. Naukowe, 1986).

91. Kalita, A., Ray, K., Biswas, A. & Hussain, M. A. *A topology for network-on-chip* w *2016 International Conference on Information Communication and Embedded Systems (ICICES)* (2016), 1–7.

92. Karkula, M. Selected aspects of simulation modelling of internal transport processes performed at logistics facilities. *Archives of Transport* **30** (2014).

93. Karp, R. M. *On-Line Algorithms Versus Off-Line Algorithms: How Much* w *Algorithms, Software, Architecture: Information Processing 92: Proceedings of the IFIP 12th World Computer Congress, Madrid, Spain, 7-11 September 1992* **1** (1992), 416.

94. Keutzer, K., Malik, S. & Newton, A. R. *From ASIC to ASIP: The next design discontinuity* w *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors* (2002), 84–90.

95. Klemiato, M., Augustyn, J., Duda, J. T. & Sterna, K. INSTEPRO: Zintegrowany System Sterowania Produkcją (2). *Pomiary Automatyka Robotyka* **15,** 53–57 (2011).

96. Kłopotek, M., Michalewicz, M., Wierzchoń, S. T., Czarnowski, I. & Jędrzejowicz, P. *Artificial Neural Network for Multiprocessor Tasks Scheduling* w *Intelligent Information Systems: Proceedings of the IIS'2000 Symposium, Bystra, Poland, June 12–16, 2000* (2000), 207–216.

97. Koch, D., Hannig, F. & Ziener, D. *FPGAs for software programmers* (Springer, 2016).

98. Krichen, F., Hamid, B., Zalila, B. & Jmaiel, M. *Design-time verification of reconfigurable real-time embedded systems* w *2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems* (2012), 1487–1494.

99. Krishna, G. & Roy, S. w *Advanced Engineering* 12–30 (Technical i Scientific Publisher, 2017).

100. Lee, J., Seo, H., Seok, H. & Kim, Y. A novel approximate adder design using error reduced carry prediction and constant truncation. *IEEE Access* **9,** 119939–119953 (2021).

101. Leung, S. S. & Shanblatt, M. A. *ASIC system design with VHDL: a paradigm* (Springer Science & Business Media, 2012).

102. Li, F. & Liu, W. An Efficient Algorithm for Reliability Evaluation of the Bus Network. *IEEE Access* **10,** 121772–121783 (2022).

103. Liaqait, R. A., Hamid, S., Warsi, S. S. & Khalid, A. A critical analysis of job shop scheduling in context of industry 4.0. *Sustainability* **13,** 7684 (2021).

104. Liu, L. *i in.* A survey of coarse-grained reconfigurable architecture and design: Taxonomy, challenges, and applications. *ACM Computing Surveys (CSUR)* **52,** 1–39 (2019).

105. Liu, Y., Eckert, C. M. & Earl, C. A review of fuzzy AHP methods for decision-making with subjective judgements. *Expert Systems with Applications* **161,** 113738 (2020).

106. Lodi, A., Martello, S. & Monaci, M. Two-dimensional packing problems: A survey. *European journal of operational research* **141,** 241–252 (2002).

107. Lu, C.-C., Lin, S.-W. & Ying, K.-C. Robust scheduling on a single machine to minimize total flow time. *Computers & Operations Research* **39,** 1682–1691 (2012).

108. Łachwa, A. w *Informatyka. (red.) Pękala, M., Chmielowski, W.* 35–49 (Oficyna Wydawnicza AFM, Kraków, 2006).

109. Mac, T. T., Copot, C., Tran, D. T. & De Keyser, R. Heuristic approaches in robot path planning: A survey. *Robotics and Autonomous Systems* **86,** 13–28 (2016).

110. Makać, W. & Urbanek-Krzysztofiak, D. *Metody opisu statystycznego* (Wydawnictwo Uniwersytetu Gdańskiego, 2006).

111. Małafiejski, M. *Uszeregowania zadań wieloprocesorowych minimalizujace średni czas przepły-wu.* (Politechnika Gdańska, 2002).

112. Mao, H. *i in.* Learning deterministic probabilistic automata from a model checking perspective. *Machine Learning* **105,** 255–299 (2016).

113. Martello, S., Monaci, M. & Vigo, D. An exact approach to the strip-packing problem. *INFORMS Journal on Computing* **15,** 310–319 (2003).

114. Maxim, D., Davis, R. I., Cucu-Grosjean, L. & Easwaran, A. *Probabilistic analysis for mixed criticality systems using fixed priority preemptive scheduling* w *Proceedings of the 25th International Conference on Real-Time Networks and Systems* (2017), 237–246.

115. Maxim, V. & Zidek, K. Design of high performance multimedia control system for UAV/UGV based on SoC/FPGA Core. *Procedia Engineering* **48,** 402–408 (2012).

116. McNaughton, R. Scheduling with deadlines and loss functions. *Management science* **6,** 1–12 (1959).

117. Miedema, L., Rouxel, B. & Grelck, C. *Task-level Redundancy vs Instruction-level Redundancy against Single Event Upsets in Real-time DAG scheduling* w *2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)* (2021), 373–380.

118. Moreno Vega, A. Synthesis of variability-tolerant circuits with adaptive clocking (2019).

119. Moselhi, O. & Lorterapong, P. *Fuzzy vs probabilistic scheduling* w *Proc. of the 12th Conference "Automation and Robotics in Construction"(ISARC)* (1995), 441–448.

120. Muntz, R. R. & Coffman, E. Optimal preemptive scheduling on two–processor systems. *IEEE Transactions on Computers* **100,** 1014–1020 (1969).

121. Nguyen, A.-T. *i in.* Fuzzy control systems: Past, present and future. *IEEE Computational Intelligence Magazine* **14,** 56–68 (2019).

122. Ning, C. & You, F. Optimization under uncertainty in the era of big data and deep learning: When machine learning meets mathematical programming. *Computers & Chemical Engineering* **125,** 434–448 (2019).

123. Olofsson, A. Epiphany-v: A 1024 processor 64-bit risc system-on-chip. *arXiv preprint arXiv:1610.01832* (2016).

124. Oniszczuk, W. *Metody modelowania* (Wydaw. Politech. Białostockiej, 1995).

125. Orthner, K. *White Paper. Applying the Benefits of Network on a Chip Architecture to FPGA System Design* spraw. tech. (Intel® Corporation, California, US, 2010).

126. Ortiz-Pimiento, N. R. & Diaz-Serna, F. J. A comparison of different redundancy based methods to solve the project scheduling problem with probabilistic activivies duration. *Management and Production Engineering Review* **10,** 73–80 (2019).

127. Ostasiewicz, K. Uwagi o prehistorii statystyki. *Studia Ekonomiczne,* 95–105 (2013).

128. Pathan, R. M. Real-time scheduling algorithm for safety-critical systems on faulty multicore environments. *Real-Time Systems* **53,** 45–81 (2017).

129. Pavlidis, V. F., Savidis, I. & Friedman, E. G. *Three-dimensional integrated circuit design* (Newnes, 2017).

130. Pellegrinelli, S., Moro, F. L., Pedrocchi, N., Tosatti, L. M. & Tolio, T. A probabilistic approach to workspace sharing for human–robot cooperation in assembly tasks. *CIRP Annals* **65,** 57–60 (2016).

131. Piegat, A. *Modelowanie i sterowanie rozmyte* (Akademicka Oficyna Wydawnicza "Exit", 1999).

132. Pinedo, M. *Scheduling* (Springer, New York, NY, 2015).

133. Popieralski, W. *Algorytmy stadne w optymalizacji problem przepływowego szeregowania zadań* prac. dokt. (Ph. D. thesis, 2013).

134. Prasanth, N. N., Balasubramanian, K. & Devi, R. C. Prioritized queue with round robin scheduler for buffered crossbar switches. *ICTACT Journal on Communication Technology* **5,** 890–893 (2014).

135. Prokopowicz, P., Czerniak, J., Mikołajewski, D., Apiecionek, Ł. & Ślezak, D. *Theory and Applications of Ordered Fuzzy Numbers: A Tribute to Professor Witold Kosiński* (Springer Nature, 2017).

136. Raj, M. D., Gogul, I., Thangaraja, M. & Kumar, V. S. *Static gesture recognition based precise positioning of 5-DOF robotic arm using FPGA* w *2017 Trends in Industrial Measurement and Automation (TIMA)* (2017), 1–6.

137. Renzini, F., Cuppini, M., Mucci, C., Franchi Scarselli, E. & Canegallo, R. Quantitative analysis of multistage switching networks for embedded programmable devices. *Electronics* **8,** 272 (2019).

138. Rivai, M. & Purwanto, D. *Implementation of fuzzy logic control in robot arm for searching location of gas leak* w *2015 International Seminar on Intelligent Technology and Its Applications (ISITIA)* (2015), 69–74.

139. Rogowska, D. Zastosowanie logiki rozmytej w zarządzaniu zapasami. *Logistyka,* 1240–1247 (2011).

140. Roosta, S. H. *Parallel processing and parallel algorithms: theory and computation* (Springer Science & Business Media, 2012).

141. Roszkowska, E. *i in.* O możliwościach wykorzystania skierowanych liczb rozmytych do podejmowania decyzji wielokryterialnych. *Przegląd Statystyczny* **64,** 373–398 (2017).

142. Rutkowski, L. *Metody i techniki sztucznej inteligencji: inteligencja obliczeniowa* ISBN: 9788301157319 (Wydawnictwo Naukowe PWN, 2019).

143. Rykaczewski, K. Systemy rozmyte i ich zastosowania. *Wstęp do metod sztucznej inteligencji,* 1–17 (2006).

144. Saha, P., Banerjee, A., Bhattacharyya, P. & Dandapat, A. *High speed ASIC design of complex multiplier using vedic mathematics* w *IEEE Technology Students' Symposium* (2011), 237–241.

145. Sahu, P. K. & Chattopadhyay, S. A survey on application mapping strategies for network-on-chip design. *Journal of systems architecture* **59,** 60–76 (2013).

146. Saint-Guillain, M., Vaquero, T., Chien, S., Agrawal, J. & Abrahams, J. Probabilistic temporal networks with ordinary distributions: Theory, robustness and expected utility. *Journal of Artificial Intelligence Research* **71,** 1091–1136 (2021).

147. Santinelli, L. & Cucu-Grosjean, L. A probabilistic calculus for probabilistic real-time systems. *ACM Transactions on Embedded Computing Systems (TECS)* **14,** 1–30 (2015).

148. Schäck, C., Heenes, W. & Hoffmann, R. *A multiprocessor architecture with an omega network for the massively parallel model gca* w *International Workshop on Embedded Computer Systems* (2009), 98–107.

149. Shashidhara, S. Field Programmable Gate Arrays And Theri Applications. *International Journal of Electrical and Electronic Engineering & Telecommunications,* 19–25 (2014).

150. Shoval, S. & Efatmaneshnik, M. A probabilistic approach to the stochastic job-shop scheduling problem. *Procedia Manufacturing* **21,** 533–540 (2018).

151. Singh, A. K., Dziurzanski, P., Mendis, H. R. & Indrusiak, L. S. A survey and comparative study of hard and soft real-time dynamic resource allocation strategies for multi-/many-core systems. *ACM Computing Surveys (CSUR)* **50,** 1–40 (2017).

152. Sjövall, P., Virtanen, J., Vanne, J. & Hämäläinen, T. D. *High-level synthesis design flow for HEVC intra encoder on SoC-FPGA* w *2015 Euromicro Conference on Digital System Design* (2015), 49–56.

153. Slimani, K., Hadaoui, R. & Lalam, M. Hardware Fuzzy Scheduler for Real-Time Independent Tasks. *Journal of Circuits, Systems and Computers* **31,** 2250155 (2022).

154. Smutnicki, C. *Algorytmy szeregowania zadań* (Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 2012).

155. Sobczyk, M. *Statystyka matematyczna* (Wydawnictwo CH Beck, 2010).

156. Soleymanpour, R. & Mohammadi, S. *A platform for multi reconfigurable instruction set processor system on chip (MRPSoC)* w *The 17th CSI International Symposium on Computer Architecture & Digital Systems (CADS 2013)* (2013), 99–104.

157. Srikanteswara, S., Palat, R. C., Reed, J. H. & Athanas, P. An overview of configurable computing machines for software radio handsets. *IEEE communications magazine* **41,** 134–141 (2003).

158. Sriram, S. & Bhattacharyya, S. S. *Embedded multiprocessors: Scheduling and synchronization* (CRC press, 2018).

159. Stasiak, A. *Automatyczna dekompozycja specyfikacji behawioralnej sprzętowo–programowego mikrosystemu cyfrowego* prac. dokt. (Rozprawa Doktorska, Uniwersytet Zielonogórski, Zielona Góra, 2007).

160. Strachacki, M. *Projektowanie i optymalizacja sprzętowo-programowych wbudowanych systemów przetwarzania danych* prac. dokt. (Ph. D. thesis, 2012).

161. Sun, L., Lin, L., Li, H. & Gen, M. Flexible Vehicle Scheduling Optimization with Uncertainty in Intelligent Logistic Systems. *Sensors and Materials* **31,** 2131–2142 (2019).

162. Szeptuch, A. Zastosowanie logiki rozmytej w badaniach gotowości operacyjnej przedsiębiorstw w procesach zarządzania strategicznego. *Zarządzanie Przedsiębiorstwem* **16,** 35–44 (2013).

163. Terekhov, D., Down, D. G. & Beck, J. C. Queueing-theoretic approaches for dynamic scheduling: a survey. *Surveys in Operations Research and Management Science* **19,** 105–129 (2014).

164. Torres, L. *i in.* w *Multiprocessor System-on-Chip* 1–21 (Springer, 2011).

165. Ultra, X. V. UltraScale Architecture and Product Data Sheet: Overview. *Accessed: Dec* **28,** 2018 (2018).

166. Uma, R., Sarojadevi, H. & Sanju, V. *Network-on-chip (NoC)-routing techniques: A study and analysis* w *2019 Global Conference for Advancement in Technology (GCAT)* (2019), 1–6.

167. Venkataraman, N. & Kumar, R. Design and analysis of application specific network on chip for reliable custom topology. *Computer Networks* **158,** 69–76 (2019).

168. Wang, L.-X. *Fuzzy systems are universal approximators* w *[1992 Proceedings] IEEE International Conference on Fuzzy Systems* (1992), 1163–1170.

169. Wenzelburger, P. & Allgöwer, F. Model predictive control for flexible job shop scheduling in industry 4.0. *Applied Sciences* **11,** 8145 (2021).

170. Wilmshurst, T. w *Designing Embedded Systems with PIC® Microcontrollers - Principles and Applications (2nd Edition)* (Elsevier, 2011). ISBN: 978-0-08-096184-2.

171. Xin, X., Mou, M. & Mu, G. *A Polynomially Solvable Case of Scheduling Multiprocessor Tasks in a Multi-Machine Environment* w *2017 2nd International Conference on Materials Science, Machinery and Energy Engineering (MSMEE 2017)* (2017), 1746–1749.

172. Xu, M., Kashyap, S., Zhao, H. & Kim, T. *Krace: Data race fuzzing for kernel file systems* w *2020 IEEE Symposium on Security and Privacy (SP)* (2020), 1643–1660.

173. Yao, Y. A comparative study of fuzzy sets and rough sets. *Information sciences* **109,** 227–242 (1998).

174. Ye, D., Chen, D. Z. & Zhang, G. Online scheduling of moldable parallel tasks. *Journal of Scheduling* **21,** 647–654 (2018).

175. Zadeh, L. A. w *Fuzzy Sets, Fuzzy Logic, and Fuzzy Systems: Selected Papers by Lotfi a Zadeh* 796–804 (World Scientific, 1996).

176. Zahid, Y., Khurshid, H. & Memon, Z. A. On Improving Efficiency and Utilization of Last Level Cache in Multicore Systems. *Information Technology and Control* **47,** 588–607 (2018).

177. Zhao, L., Ren, Y. & Sakurai, K. Reliable workflow scheduling with less resource redundancy. *Parallel Computing* **39,** 567–585 (2013).

178. Zhu, X., Wen, S., Camtepe, S. & Xiang, Y. Fuzzing: a survey for roadmap. *ACM Computing Surveys (CSUR)* **54,** 1–36 (2022).

179. Żurowski, M. *i in. Podzielne szeregowanie zadań z pozycyjno-zależnymi czasami wykonywania na dwóch równoległych identycznych maszynach* prac. dokt. (Adam Mickiewicz University, Poznań, 2019).

Spis rysunków

Spis tabel