POLITECHNIKA KOSZALIŃSKA

WYDZIAŁ ELEKTRONIKI I INFORMATYKI

ROZPRAWA DOKTORSKA

Równoległa i potokowa realizacja wybranych algorytmów algebry liniowej w układach reprogramowalnych

mgr inż. Piotr Ratuszniak

Promotor: prof. dr hab. inż. Michał Białko
Członek rzeczywisty PAN

Koszalin 2011

Rozprawę dedykuję mojej najbliższej
Rodzinie, Nauczycielom oraz pamięci
dr hab. inż. Olega Maslennikowa

Podziękowania

Chciałbym serdecznie podziękować wszystkim za okazaną pomoc i wsparcie podczas powstawania tej pracy. Szczególne wyrazy wdzięczności chciałbym złożyć swojej najbliższej rodzinie: żonie Lidii, synowi Wiktorowi oraz rodzicom za cierpliwość, zrozumienie i okazane bezcenne wsparcie.

Równie gorąco chciałbym podziękować moim wszystkim Nauczycielom i Przewodnikom Naukowym, którzy w znacznym stopniu przyczynili się do powstania tej pracy. Rozprawę zadedykowałem pamięci dr hab. inż. Olega Maslennikowa, gdyż pod jego opieką powstała większa część opisanych badań. Dziękuje całej jego rodzinie za to, że ich Mąż i Ojciec poświecił mi tak wiele swojego cennego czasu. Ponadto dr inż. Natalii Maslennikow dziękuję za pomoc w redagowaniu pracy oraz bardzo cenne wskazówki merytoryczne. Chciałbym również bardzo serdecznie podziękować swojemu obecnemu promotorowi prof. dr hab. inż. Michałowi Białko za cierpliwość, poświęcony czas i okazaną pomoc w pisaniu rozprawy. Pragnę jeszcze wyrazić wdzięczność mojej nauczycielce Pani mgr Teresie Mysik za wzbudzenie ciekawości, poświęcony dodatkowy czas oraz pokazanie jaką satysfakcję może dawać rozwiązywanie różnego rodzaju zadań i problemów.

Na koniec chciałbym jeszcze podziękować moim współpracownikom za okazaną pomoc i owocne dyskusje naukowe, które również miały wpływ na powstanie tej rozprawy, m.in.: dr inż. Adamowi Słowikowi, dr inż. Grzegorzowi Bocewiczowi oraz pozostałym koleżankom i kolegom.

Piotr Ratuszniak

Wykaz skrótów:

ALU – ang. *Arithmetic Logic Unit* - jednostka arytmetyczno logiczna

ASIC – ang. *Application Specific Integrated Circuit* - typ elektronicznych układów scalonych zaprojektowanych do realizacji z góry ściśle określonego zadania

BPP – ang. B*in Packing Problem* – kombinatoryczny problem pakowania(plecakowy) klasy NP trudnej

BRAM – ang. *Block RAM* - bloki pamięci dostępnej w strukturze układów FPGA

CAD – ang. C*omputer-Aided Design* – projektowanie wspomagane komputerowo

CLB – ang. *Configurable Logic Block* – konfigurowalny blok logiczny – jednostka pojemności układu FPGA

CP – ang. C*onstraints Programming*- programowanie z ograniczeniami

DSP – ang. *Digital Signal Processing*- cyfrowe przetwarzanie sygnałów

FPGA – ang. *Field Programmable Gate Array* – rodzaj programowlanego układu logicznego

GPP – ang. *General Purpose Processor* – procesor ogólnego przeznaczenia

GPU – ang. *Graphic Processor* – procesor graficzny

GZI – graf zależności informacyjnych

HDL – ang. *Hardware Description Language*- język opisu sprzętu

HLL – ang. *High Level Programming Language* – język programowania wysokiego poziomu

HPC – ang. *High Performance Computing* - obliczenia wysokiej wydajności

IPCore – ang. *Intellectual Property Core* – zaprojektowany blok logiczny do wielokrotnego wykorzystania na platformie FPGA

LRGS – lokalnie równoległa - globalnie szeregowa koncepcja realizacji obliczeń

LSGR – lokalnie szeregowa - globalnie równoległa koncepcja realizacji obliczeń

NOP – ang. *No Operation* – instrukcja pusta

PK – pamięć konfiguracyjna

RFA – ang. *Rational Fraction Arithmetic* - arytmetyka ułamkowa

SoC – ang. *System on Chip* – systemy jednoukładowe

VHDL – ang. *Very High Speed Integrated Circuits Hardware Description Language*- język opisu sprzętu wysokiego poziomu

VLSI – ang. *Very-Large-Scale Integration* – układy elektroniczne wysokiej skali integracji

U2 – kod uzupełniania do dwóch, system reprezentacji liczb całkowitych w sys. dwójkowym

1. Wstęp

1.1. Uzasadnienie aktualności oraz określenie głównych kierunków badań autora rozprawy.

Obecnie w wielu obszarach nauki i przemysłu wymagane jest stosowanie wydajnych platform obliczeniowych, np. w kryptografii, do symulacji złożonych modeli, w badaniach sejsmicznych, w algorytmach przetwarzania obrazów, przeszukiwaniu i sortowaniu danych, w symulacjach finansowych i badaniach genetycznych. Często do rozwiązania tych problemów niezbędna jest realizacja algorytmów algebry liniowej na dużych macierzach lub wektorach. Nowoczesnym systemom komputerowym stawiane są więc coraz większe wymagania, między innymi w zakresie zwiększania wydajności, co spowodowało dynamiczny rozwój nowej dziedziny nauki w postaci wysokowydajnych obliczeń komputerowych HPC (ang. *High-Performance Computing*) Radykalnym sposobem zwiększenia wydajności systemów komputerowych jest wykorzystywanie architektur równoległych [1,2,3,4]. Jeszcze kilkanaście lat temu wysokowydajne obliczenia wykonywano przede wszystkim w wyspecjalizowanych centrach komputerowych z wykorzystaniem np. superkomputerów zawierających nawet setki procesorów GPP (ang. *General Purpose Processor*. W ostatnich latach można zaobserwować zdecydowany wzrost zainteresowania wykorzystywaniem nowych równoległych platform sprzętowych do akceleracji obliczeń komputerowych. Obecnie powszechnie wykorzystywanymi platformami sprzętowymi do akceleracji obliczeń i przetwarzania danych są: wielordzeniowe procesory CPU[5,6] procesory graficzne GPU(ang. *Graphic Processor Unit*) [7,8] oraz układy FPGA (ang. *Field Programmable Gate Array*)[9,10,11,12]. Powstały również wyspecjalizowane platformy sprzętowe do akceleracji obliczeń, czy wydajnego przetwarzania danych, np. zaprojektowana przez firmy Sony Corporation Sony Computer Entertainment, IBM oraz Toshiba architektura „Cell"[1] czy architektura Intel „Larrabee"[2] zawierająca w sobie najlepsze cechy architektury CPU i GPU. Ze względu na kolejne bariery technologiczne „komputery nie stają się coraz szybsze, ale coraz szersze"[9].

Przykładem prac badawczych nad wykorzystaniem różnego rodzaju architektur równoległych do realizacji algorytmów algebry liniowej, w tym procesorów wielordzeniowych i procesorów GPU, są prace prof. J.Dongarry i jego zespołu[13,14,15,16], zaimplementowane m.in. w bibliotekach LAPACK, MAGMA i PLASMA[3]. Prace te skupiają się jednak na tworzeniu nowych bibliotek programowych realizowanych przy wykorzystaniu istniejących, gotowych do wykorzystania sprzętowych platform równoległych.

Alternatywą dla wymienionych platform sprzętowych pozostają wyspecjalizowane architektury równoległe implementowane w układach FPGA[9,10], w tym również

---

[1] http://www.research.ibm.com/cell/home.html
[2] http://software.intel.com/en-us/articles/larrabee/
[3] http://icl.cs.utk.edu/iclprojects/

akceleratory dla algorytmów algebry liniowej[17,18,19]. Tylko w przypadku tej platformy sprzętowej dostosowuje się architekturę systemu do realizowanego algorytmu, co pozwala na bardziej efektywną organizacje obliczeń komputerowych. Pomimo niższej częstotliwości pracy układów FPGA w stosunku do częstotliwości pracy procesorów CPU, w wielu zastosowaniach akceleratory budowane na tej platformie sprzętowej mogą być wydajniejsze dzięki zastosowaniu przetwarzania równoległego i potokowego[1,9,17].

W ostatnich latach pojawiły się również komercyjne akceleratory obliczeniowe oparte na układach FPGA, np. akceleratory firmy DRC[4] lub Mitronics[5]. Wadą niektórych z tych rozwiązań jest wolna komunikacja z pamięcią komputera, np. przez magistralę PCI. Są to również rozwiązania znacznie droższe w stosunku do standardowych, powszechnie dostępnych układów FPGA. Również w przypadku wykorzystywania wymienionych gotowych rozwiązań wymagana jest znajomość języków opisu sprzętu HDL (ang. *Hardware Description language*), np. VHDL czy Verilog, lub wykorzystywanie jednego z narzędzi do opisywania projektu na poziomie HLL (ang. *High Level Language*), np. w języku C i automatycznego generowania struktury opisanej w jednym z języków HDL, np. „Impulse C"[6] lub „Mitrion C"[7]. Stosowanie wymienionych narzędzi projektowych pozwala skrócić czas projektowania, jednak często architektury generowane automatycznie charakteryzują się gorszą wydajnością[9].

Przedmiotem badań opisanych w niniejszej rozprawie jest projektowanie i optymalizacja akceleratorów obliczeniowych wybranych algorytmów algebry liniowej, wykorzystujących przetwarzanie równoległe i potokowe, przeznaczonych do implementacji na platformie FPGA.

1.2. Analiza budowy wewnętrznej nowoczesnych układów FPGA pod kątem możliwości efektywnej realizacji podstawowych operacji arytmetycznych

Pierwsze układy FPGA zaprojektowane w 1985 roku zawierały jedynie 64 programowalne komórki. W kolejnych latach obserwowano dynamiczny rozwój projektowanych układów. Obecnie dominującymi na rynku producentami układów FPGA są firmy Xilinx i Altera. Nowoczesne reprogramowalne układy cyfrowe FPGA, np. rodziny Xilinx Virtex4,5,6[8], czy Altera Stratix IV i StratixV[9] zawierają setki tysięcy rekonfigurowalnych komórek logicznych, wbudowane bloki pamięci, setki a nawet tysiące wbudowanych bloków mnożących lub bloków DSP(ang. *Digital Signal Processing*), jak również wbudowane rdzenie

---

[4] http://www.drccomputer.com

[5] http://www.mitrionics.com/?page=hybrid-computing-solutions

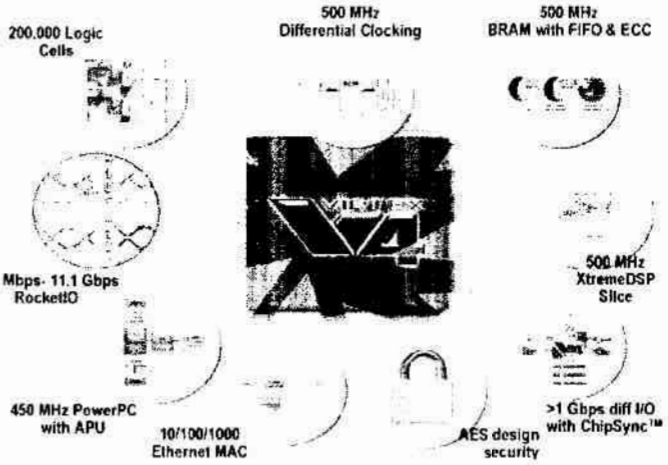
[6] http://www.impulseaccelerated.com/products_universal.htm

[7] http://www.mitrionics.com/?page=FPGAs_in_HPC_5

[8] http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

[9] http://www.altera.com/products/devices/stratix-fpagas/stratix-v/overview/stxv-overview.html

mikroprocesorów, czy bloki transmisji danych. W związku z tak rozwiniętą architekturą współczesnych układów FPGA w pojedynczym układzie możliwa jest implementacja dziesiątek a nawet setek bloków operacyjnych dla podstawowych operacji algorytmów algebry liniowej, które mogą być realizowane w sposób równoległy.



Rysunek 1. Rozbudowana architektura współczesnego układu FPGA (źródło[10])

Tak rozwinięta technologia VLSI(ang. *Very-large-scale Integration*) pozwala na umieszczenie w pojedynczym układzie reprogramowalnym całego systemu jednoukładowego SoC (*ang.* System-on-Chip)[1]. Umieszczenie całego systemu w pojedynczym układzie pozwala, np. w telefonii komórkowej, na zmniejszenie rozmiarów całego systemu, zmniejszenie kosztów oraz zmniejszenie poboru mocy. Jednym z nowoczesnych trendów w projektowaniu systemów jednoukładowych jest wykorzystywanie reprogramowalnej platformy sprzętowej[20].Podstawową zaletą platformy FPGA, jest możliwość rekonfiguracji, co umożliwia w systemach jednoukładowych wykorzystywać tą samą platformę sprzętową do realizacji wielu zadań, np. w przypadku akceleratorów obliczeniowych ten sam akcelerator może przyspieszać różne metody obliczeniowe w zależności od bieżącej konfiguracji. Platforma FPGA jest również atrakcyjna pod względem ceny w stosunku do innych sprzętowych platform, zwłaszcza w przypadku opracowywania modeli prototypowych lub niewielkiej liczby egzemplarzy dla wyspecjalizowanych systemów komputerowych. W stosunku do innych platform sprzętowych układy FPGA wypadają korzystnie również w zakresie poboru mocy, co ma zasadnicze znaczenie np. dla systemów z ograniczonym zasilaniem. Dzięki korzystnemu współczynnikowi GFlops/Watt, nawet 3 krotnie większemu w stosunku do innych współczesnych równoległych platform sprzętowych[21,17], akceleratory FPGA są wykorzystywane w nowej dziedzinie „Green Computing".

[10] http://www.midasireland.com/xilinx.htm

Obecnie trudnym zadaniem staje się jednak efektywne zagospodarowanie tak dużych zasobów sprzętowych [22], tj. zaprojektowanie systemu SoC bezbłędnie i w rozsądnym czasie, przy równoczesnym zachowaniu pożądanej wydajności i funkcjonalności układu oraz minimalnym poborze mocy. W związku z tym, nowoczesne tendencje w projektowaniu systemów jednoukładowych są ukierunkowane m. in. na wykorzystanie gotowych projektów (IP Core-ang. *Intelectual Property Core*) dla poszczególnych bloków systemu [20, 23,24] oraz na automatyzację procesu projektowania i weryfikacji projektu na wszystkich poziomach [25]. Nowoczesne narzędzia programowe do syntezy pozwalają projektantowi zdecydować o wykorzystaniu wbudowanych bloków mnożących, pamięci BRAM nawet bez konieczności znajomości ich budowy i zasad działania. Jednak w przypadku optymalizacji projektowanych jednostek przetwarzających np. pod względem wydajności, która w dużym stopniu zależy od maksymalnej możliwej częstotliwości działania systemu, należy brać pod uwagę budowę i zasady działania wbudowanych bloków lub wykorzystywać gotowe opisy jednostek przetwarzających i bloków operacyjnych, wybierając te opisy z istniejących bibliotek lub z odpowiednich generatorów IP Core [1].

Przedstawione w rozprawie metody projektowania architektur równoległych do akceleracji obliczeń wybranych algorytmów algebry liniowej zostały zaimplementowane w autorskich generatorach IP Core, opisanych w dodatkach 1 i 2 oraz w publikacjach[26,27,28].

Wiodące obecnie na rynku oprogramowanie do syntezy i implementacji układów cyfrowych na platformie FPGA, np. Xilinx ISE[11] lub Altera Quartus[12], zawiera w sobie generatory zoptymalizowanych wybranych bloków funkcjonalnych. Wymienione generatory pozwalają na generowanie bloków funkcjonalnych, zarówno dla reprezentacji danych w postaci stało i zmiennoprzecinkowej. Warto zaznaczyć, że do efektywnej realizacji algorytmów algebry liniowej, operujących zazwyczaj na danych w postaci dużych macierzy lub wektorów, pod względem dokładności przeprowadzanych obliczeń wymagana jest przynajmniej reprezentacja zmiennoprzecinkowa pojedynczej precyzji. Dla arytmetycznych operacji sumowania, mnożenia i mnożenia z akumulacją wyniku dostępne są specyficzne generatory, pozwalające na wykorzystywanie wbudowanych w FPGA bloków DSP. Wykorzystanie tych bloków, jak twierdzą producenci, pozwala zwiększyć szybkość pracy projektowanych systemów, zmniejszyć zużycie dostępnej uniwersalnej przestrzeni programowalnej w postaci bloków CLB(ang. *Configurable Logic Block*) oraz zmniejszyć ich pobór mocy[29,30].

Dla niektórych operacji arytmetycznych występujących w algorytmach algebry liniowej, dostępne oprogramowanie nie zawiera generatorów wykorzystujących wbudowane bloki DSP. Problem stanowi efektywna implementacja operacji dzielenia, podczas której w/w bloki nie mogą być wykorzystane. Mianowicie, realizacja szybkich bloków dzielenia, jak również

[11] http://www.xilinx.com/tools/designtools.htm
[12] http://www.altera.com/products/software/sfw-index.jsp

bloków operacyjnych działających na liczbach zmiennoprzecinkowych, wymaga ogromnych zasobów układu reprogramowalnego [19,31,32].

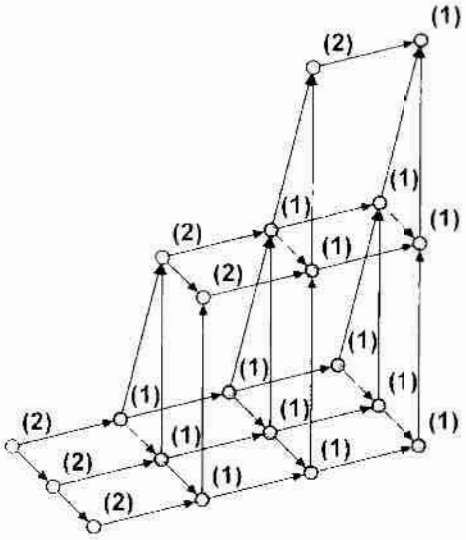
Wraz z rozwojem układów FPGA pojawiły się również układy wielokontekstowe (ang. *multicontext FPGA devices*)[33,34,35]. Wielokontekstowy układ FPGA posiada pewną liczbę jednakowych bloków pamięci konfiguracyjnych. W danym momencie czasu aktywnym może być tylko jeden blok takiej pamięci, jednak taka organizacja pozwala na szybką zmianę konfiguracji układu(w ideale – w ciągu jednego taktu zegarowego), nawet w trakcie funkcjonowania systemu. Wiele zespołów badawczych, w tym również polskie zajmowały się tematyką wykorzystania tych układów[36,37,82].

Jednym z zagadnień omawianych w rozprawie jest opracowanie algorytmu organizacji równoległych obliczeń dla algorytmów algebry liniowej przeznaczonych do implementacji w wielokontekstowych układach FPGA[38,39,40].

Zapewne akceleratory oparte na pojedynczych układach FPGA nie mogą konkurować pod względem wydajności obliczeniowej dla algorytmów algebry liniowej z wyspecjalizowanymi klastrami obliczeniowymi[6], jednak jest to platforma sprzętowa zdecydowanie bardziej dostępna i z powodu wcześniej opisanych zalet może być z powodzeniem stosowana w wyspecjalizowanych systemach jednoukładowych.

1.3. Analiza zależności informacyjnych i zbiorów operacji podstawowych algorytmów algebry liniowej pod kątem ich równoległej i potokowej realizacji w układach reprogramowalnych FPGA.

Algorytmy algebry liniowej mają szerokie zastosowanie w wielu dziedzinach nauki i przemysłu. Do powszechnie wykorzystywanych algorytmów można zaliczyć, np. mnożenie macierzy, algorytm rozkładu macierzy LU metodą Gaussa, rozkład LL^T metodą Cholesky'ego, rozkład QR metodą Givensa, rozwiązywanie układu równań metodą podstawienia, redukcji wstecznej, Jordana-Gaussa itp. Zależności informacyjne w algorytmach przedstawiane są za pomocą grafów zależności informacyjnej GZI[41,42,43], który są ich geometrycznymi reprezentacjami. Przykładowy graf zależności informacyjnej dla algorytmu rozkładu macierzy LU metodą Gaussa wygenerowany w opracowanym przez autora rozprawy generatorze (opis w podrozdz. 4.2) przedstawiony jest na rys.2. W generatorze tym zaimplementowano metodę konstruowania grafów opisaną w pracy[1], polegającą na symulacyjnym wykonaniu analizowanego programu uzupełnionego o niewielką liczbę instrukcji przypisania w celu zebrania informacji o węzłach i łukach grafu.



Rysunek 2. Graf zależności informacyjnych dla algorytmu rozkładu macierzy LU metodą Gaussa dla rozmiaru macierzy N=4 uzyskany w programie JGEN (podrozdz. 4.2).

Wierzchołki grafu odpowiadają wykonywanym w algorytmie operacjom, natomiast łuki – przekazywanym danym[33,34,35,36]. W przedstawionym na rys. 2 grafie węzły oznaczone symbolem (2) odpowiadają operacji dzielenia, natomiast węzły oznaczone symbolem (1) odpowiadają operacji mnożenia z odejmowaniem. Większość algorytmów algebry liniowej cechuje się regularnymi grafami zależności informacyjnej, gdyż wielokrotnie w trakcie realizacji algorytmu wykonywane są rekursywnie te same operacje oraz cechują się zazwyczaj krótkimi łukami, łączącymi sąsiadujące węzły. Dlatego równoległa realizacja tych algorytmów, np. w postaci wyspecjalizowanych macierzy procesorowych[44-57], lub ich uproszczonej wersji - procesorach systolicznych[58-71], jest najbardziej efektywna w stosunku do innych równoległych systemów, a układy FPGA są efektywną platformą sprzętową do ich realizacji, zarówno pod względem kryterium *wydajność/cena, wydajność/złożoność sprzętowa* oraz *wydajność/zużycie energii*)[1]. Macierze procesorowe można zobrazować jako jedno lub dwuwymiarowe kraty, w węzłach których są umieszczone elementy przetwarzające. Elementy te powiązane są w macierzy pomiędzy sobą siecią lokalnych połączeń, które służą do wymiany przetwarzanych danych. Przetwarzanie to jest zorganizowane w sposób systoliczny[45,58,61], co oznacza, że dane wprowadzane i wyprowadzane są tylko z granicznych procesorów macierzy w trakcie wykonywania obliczeń. Zazwyczaj zaprojektowana macierz procesorowa przeznaczona jest do realizacji wybranego algorytmu, z określonym rozmiarem danych wejściowych. Tak wąska specjalizacja była początkowo powodem zahamowania rozwoju macierzy procesorowych implementowanych w postaci układów ASIC ze względu na wysoką cenę. Wadę tę jednak pozwala wyeliminować zastosowanie platformy reprogramowalnej w postaci układów FPGA. Układy te cechują się możliwością ponownej rekonfiguracji, co umożliwia implementację kolejnej wyspecjalizowanej architektury macierzy procesorowej.

Do najczęściej spotykanych, w algorytmach algebry liniowej, rodzajów operacji arytmetycznych można zaliczyć: mnożenie, mnożenie z akumulacją wyniku, sumowanie, dzielenie i porównanie. Jak wspomniano w poprzednim podrozdziale dla większości tych operacji, zwłaszcza mnożenia i sumowania, dostępne są generatory IPCore, które pozwalają generować efektywne struktury na platformie FPGA dla reprezentacji stało i zmiennoprzecinkowej. Najmniej efektywną strukturę uzyskuje się dla operacji dzielenia[19,31,32], zarówno pod względem wykorzystywanych bloków CLB układu FPGA, maksymalnej częstotliwości pracy oraz liczby stopni w potoku. Powoduje to zasadnicze obniżenie efektywności implementacji na platformie FPGA algorytmów, nie tylko algebry liniowej, ale takich, które posiadają operację dzielenia w ścieżce krytycznej grafu GZI, od której zależy minimalny czas realizacji algorytmu. Przykładowo Xilinx „Divider Generator 2.0" umożliwia generowanie bloków dzielenia dla maksymalnie 32 bitowej stałoprzecinkowej reprezentacji danych, co jest ze względu na dokładność przeprowadzanych obliczeń w algorytmach algebry liniowej wartością niewystarczającą. Natomiast parametry bloków dzielenia wygenerowanych z wykorzystaniem generatora Xilinx „Floating-point 4.0" charakteryzują się zazwyczaj dużą liczbą wykorzystywanych bloków CLB oraz dużą liczbą stopni w potoku, co często utrudnia organizacją obliczeń w macierzach procesorowych oraz powoduje spadek wydajności całej macierzy w przypadku występowania tych operacji pojedynczo.

W rozprawie przedstawiono wykorzystanie arytmetyki ułamkowej RFA (ang. *Rational Fractions Arithmetic)*[72,73,74] do sprzętowej realizacji algorytmów algebry liniowej, co umożliwia bardziej efektywną realizację na platformie FPGA podstawowych operacji arytmetycznych, przy zachowaniu pożądanej dokładności obliczeń. Prace badawcze nad wykorzystaniem arytmetyki ułamkowej na platformie FPGA były prowadzone, m.in. przez autora w latach 2006-2008 w ramach grantu badawczego Ministerstwa Nauki i Szkolnictwa Wyższego N51500232/0176 pod tytułem „Zastosowanie arytmetyki ułamkowej w reprogramowalnych jednostkach przetwarzających systemów jednoukładowych". W rozdziale 3 przedstawiono szczegółowe porównanie parametrów bloków operacyjnych podstawowych operacji arytmetycznych zrealizowanych przy wykorzystaniu arytmetyki ułamkowej na platformie FPGA z analogicznymi blokami wykorzystującymi reprezentacje stało i zmiennoprzecinkową.

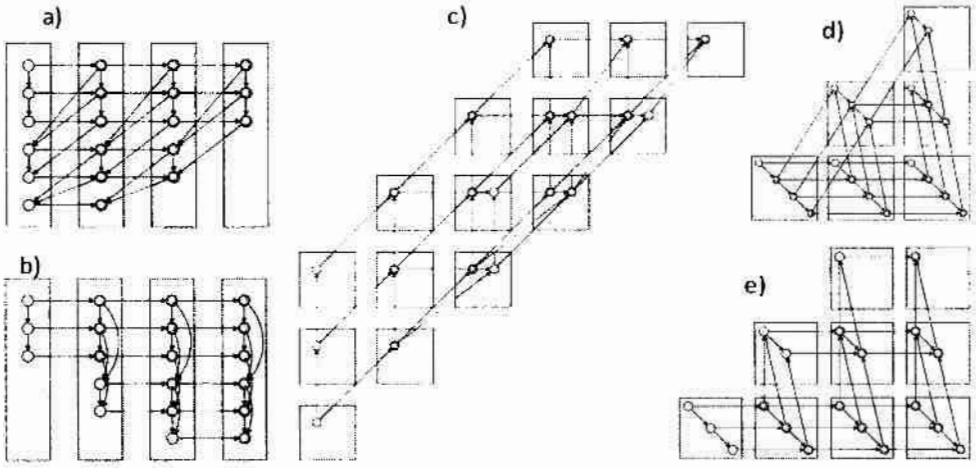
1.4. Przegląd znanych architektur równoległych jednostek przetwarzających przeznaczonych do realizacji algorytmów algebry liniowej oraz metod i narzędzi programowych wspomagających ich projektowanie

Jak wspomniano w poprzednich podrozdziałach macierze procesorowe są najbardziej efektywną architekturą równoległą przeznaczoną do realizacji algorytmów algebry liniowej, ze względu na dopasowanie swojej struktury do realizowanego algorytmu. Natomiast najbardziej efektywną platformą sprzętową do ich implementacji jest platforma FPGA, ze

względu na kryteria *wydajność/złożoność sprzętowa*, *wydajność/cena* oraz *wydajność/zużycie energii*. Miedzy innymi dynamiczny rozwój platformy FPGA spowodował ponowne zainteresowanie projektowaniem macierzy procesorowych oraz przetwarzaniem systolicznym. Prace na temat koncepcji przetwarzania systolicznego oraz projektowania macierzy procesorowych pojawiły się już po koniec lat 80-tych[44,47,58]. Proces projektowania macierzy procesorowych składa się z dwóch podstawowych etapów: utworzenia grafu zależności informacyjnej algorytmu oraz jego odwzorowania w architekturę macierzy procesorowej. Metody odwzorowania zwane są również odwzorowaniem czasowo – przestrzennym. Przy odwzorowaniu przestrzennym (ang. *allocation mapping*) każdemu wierzchołkowi grafu algorytmu (tj. każdej operacji w algorytmie) przypisywany jest numer procesora w którym zostanie on wykonany, natomiast przy odwzorowaniu czasowym(szeregującym) (ang. *time mapping* lub *schedule mapping*) przypisywany jest numer taktu, w którym ten wierzchołek (operacja) ma być zrealizowany. Opracowano już kilkanaście metod odwzorowania algorytmów o regularnych grafach zależności informacyjnych w wyspecjalizowane architektury równoległe[44-71]. Metody te opracowywano w różnych krajach, przez różne zespoły badawcze. Za jedną z najbardziej znanych, często przytaczanych jest metoda opracowana przez amerykański zespół prof. S.Y.Kunga[44]. Do najważniejszych wad tej metody można zaliczyć: brak metody konstruowania grafów zależności informacyjnych, intuicyjna i złożona metoda określania zbioru architektur macierzy procesorowych, brak systematycznej metody odwzorowania przestrzennego oraz heurystyczny sposób odwzorowania czasowego. Również w krajach byłego Związku Radzieckiego prowadzono intensywne badania nad odwzorowaniem algorytmów regularnych w macierze procesorowe. Na uwagę zasługują metody opracowane w zespołach prof. S.Seduchina[75] i prof. J.Kaniewskiego[49-51,63,76]. Do najistotniejszych wad metody opracowanej w zespole prof. S.Seduchina można zaliczyć brak formalnej metody konstruowania grafu zależności informacyjnej oraz możliwość odwzorowania *n*-wymiarowego grafu w *n-1*-wymiarowe architektury macierzy procesorowej. W następnych latach eliminowano kolejne niedociągnięcia. Niektóre metody stały się na tyle sformalizowane, że umożliwiły ich implementację w postaci narzędzi programowych CAD(ang. *Computer-Aided Design*) służących do projektowania macierzy procesorowych, takich jak: MMAplha[77], S^4[75], oraz AKOSS[78,79]. Najbardziej rozpowszechnionym i złożonym spośród wymienionych jest opracowywane w francuskim zespole prof. P.Quintona – środowisko MMAplha(ostatnia wersja V2-1-0, 13.06.2010). Do wad tego środowiska można zaliczyć konieczność opisania zadanego algorytmu w specjalnie opracowanym języku Aplha[13], konieczność znajomości metody odwzorowania prof. P.Quintona[45,47], brak możliwości konstruowania grafów zależności informacyjnych do ich wizualizacji, konieczność określenia odwzorowania przestrzennego oraz wprowadzenie

[13] http://www.irisa.fr/cosi/ALPHA/alpha_english.html

dodatkowych zmiennych do programu wejściowego. Jednak ciągle największą wadą tego środowiska, podobnie jak środowiska S^4, jest możliwość odwzorowania *n*-wymiarowego grafu w *n-1*-wymiarowe architektury macierzy procesorowych. Bardzo dobrze sformalizowaną, nie posiadającą tej wady metodą, jest metoda opracowana w zespole prof. J.Kaniewskiego. Pozwala ona na odwzorowanie *n*-wymiarowego grafu w *m*-wymiarowe architektury, dla wartości $n\text{-}m \geq 1$. Metoda ta oparta jest o rachunek macierzowy. Węzły(operacje) grafu zależności informacyjnej rozpięte są na *n*-wymiarowej kracie całkowitoliczbowej. Odwzorowanie przestrzenne w tej metodzie polega na dobraniu odpowiednich wartości elementów macierzy odwzorowania przestrzennego, a następnie na wymnożeniu współrzędnych wszystkich wierzchołków przez tę macierz, w efekcie czego otrzymuje się współrzędne elementu przetwarzającego(procesora) w macierzy procesorowej, w którym dany węzeł(operacja) będzie wykonana. Jednak w wyniku takiej operacji otrzymujemy wynikową strukturę macierzy procesorowej, zależnej od macierzy odwzorowania przestrzennego oraz często od rozmiaru macierzy wejściowej algorytmu. Możliwe jest dokonanie odwzorowania przestrzennego automatycznie, z wykorzystaniem podstawowych projekcji liniowych oraz wybór najlepszej architektury za względu na przyjęte kryterium z zazwyczaj kilkunastu różnych rozwiązań. Podobnie odwzorowanie czasowe polega na określeniu wartości wektora projekcji, następnie poprzez wymnożenie współrzędnych kolejnych wierzchołków grafu przez ten wektor otrzymujemy numer taktu, w którym dany wierzchołek będzie wykonany. Liniowe odwzorowanie czasowe często jednak powoduje powstanie dużej liczby taktów „pustych" – NOP (ang. *No Operations*), co powoduje obniżenie wydajności całej macierzy procesorowej. Możliwe jest odnalezienie bardziej efektywnej nieliniowej funkcji odwzorowania czasowego, jednak znajdowana jest ona w sposób heurystyczny i wymaga doświadczenia projektanta. Ze względu na duży stopień formalizacji metoda konstruowania grafów zależności informacyjnej, przedstawiona w pracy[1] oraz opisana metoda odwzorowania grafów w macierze procesorowe została zaimplementowana w postaci programowej, w rozwijanym przez autora środowisku JGEN(podrozdz. 4.2)). Przykładowe architektury macierzy procesorowych, uzyskane z wykorzystaniem metody opracowanej w zespole prof. J.Kaniewskiego, dla podstawowych wariantów liniowej projekcji przestrzennej, wygenerowane przy pomocy środowiska JGEN przedstawia rys.3



Rysunek 3 Przykładowe struktury macierzy procesorowych, uzyskanych w wyniku liniowego odwzorowania przestrzennego grafu zależności informacyjnej z rys.2.

W zespole prof. J.Kaniewskiego, w latach 1990-1999, powstało wiele wyspecjalizowanych architektur macierzy procesorowych do realizacji wybranych algorytmów, w tym algorytmów algebry liniowej[49-51,63,76]. W celu otrzymywania bardziej efektywnych architektur macierzy procesorowych możliwe jest dokonywanie transformacji grafów zależności informacyjnych przed operacją jego odwzorowania. Przykłady takich transformacji opisano w pracy[1], jednak są to transformacje przystosowane do jednego rodzaju grafu zależności informacyjnej, wobec czego nie da się ich zastosować dla większej liczby algorytmów algebry liniowej. Również w ostatnich latach przedstawiano kolejne prace na temat projektowania architektur macierzy procesorowych lub przetwarzania systolicznego, w tym również dedykowane do implementacji na platformie FPGA, jednak wykorzystują one zazwyczaj wcześniej opracowane metody odwzorowania[64] lub nadal wykorzystują liniowe projekcje przestrzenne i czasowe[80]. Niektóre z nich zostały zaimplementowane w kolejne środowiska wspomagające projektowanie jak PARO[54] i PICO-NPA[81].

1.5. Cel i teza pracy

Celem pracy jest opracowanie metod projektowych pozwalających na skrócenie czasu realizacji wybranych algorytmów algebry liniowej, w nowoczesnych układach reprogramowalnych FPGA, poprzez ich dostosowanie do osobliwości architektury tych układów oraz organizację obliczeń w sposób równoległy i potokowy.

Obecnie powszechnie dostępne są nowoczesne układy FPGA zawierające w swojej architekturze dużą liczbę komórek programowalnych CLB, wbudowanych bloków DSP, bloków pamięci operacyjnej BRAM oraz inne wyspecjalizowane bloki funkcjonalne. Opracowano również wielokontekstowe układy reprogramowalne, zawierające większą liczbę pamięci konfiguracyjnej, co pozwala na zmianę konfiguracji w trakcie działania układu. W pracy założono, że zmiana kontekstu, oznaczająca zmianę architektury implementowanego w FPGA systemu może być wykonana w bardzo krótkim czasie (np. w ciągu kilku taktów zegara). Znane są również metody konstruowania grafów algorytmów

regularnych i ich odwzorowania w architektury akceleratorów równoległych (np. macierzy procesorowych). W znanych metodach odwzorowania funkcja odwzorowania szeregującego jest zazwyczaj funkcją liniową. To powoduje (np. dla większości algorytmów algebry liniowej) pojawienie się, podczas działania systemu, dużej liczby tzw. „taktów pustych", w których jeden lub kilka procesorów nie wykonuje operacji algorytmu. Efektem końcowym jest wydłużenie czasu obliczeń. Możliwym rozwiązaniem problemu redukcji liczby pustych taktów jest stosowanie nieliniowych funkcji szeregujących, lecz ich odnalezienie odbywa się w sposób heurystyczny i wymaga doświadczenia projektanta. Często w znanych metodach również odwzorowanie przestrzenne jest funkcją liniową, co narzuca wynikający z takiego odwzorowania kształt architektury równoległej, np. macierzy procesorowej (kształt macierzy oraz liczba elementów przetwarzających). W celu polepszenia właściwości projektowanych macierzy procesorowych lub modyfikacji kształtu projektowanej macierzy procesorowej stosuje się przekształcenia grafów zależności informacyjnych przed procesem odwzorowania[1], jednak są to metody wyspecjalizowane dla konkretnego realizowanego algorytmu obliczeniowego. Nie da się wiec na ich podstawie opracować uniwersalnej metody np. dla wielu algorytmów algebry liniowej.

Propozycją autora jest stosowanie algorytmu ewolucyjnego do odwzorowania przestrzennego grafów zależności informacyjnych w architektury równoległe w postaci macierzy procesorowych lub w strukturę kontekstów dla wielokontekstowego układu FPGA. Zastosowanie algorytmu ewolucyjnego pozwala na dokonanie odwzorowania grafu algorytmu w architekturę macierzy procesorowej lub wektora kontekstów o praktycznie dowolnym kształcie, co jest istotną zaletą, zwłaszcza przy ograniczonych zasobach sprzętowych w postaci układu reprogramowalnego. Proponowaną metodę można stosować do projektowania architektur równoległych dla rożnych algorytmów, w tym algorytmów algebry liniowej. W przeciwieństwie do wcześniej opracowanych metod, proponowany algorytm nadaje się do pełnej automatyzacji i implementacji w postaci IPCore dla szerokiej gamy algorytmów i może być wykorzystywany przez wielu użytkowników, w tym użytkowników nie będących specjalistami w projektowaniu architektur równoległych, gdyż nie wymaga określenia funkcji odwzorowania przestrzennego i czasowego. Po dokonaniu odwzorowania przestrzennego, w sposób iteracyjny określane są dla każdej operacji(węzła grafu) minimalne możliwe takty, w których dane operacje będą wykonane. Zastosowanie algorytmu ewolucyjnego do odwzorowania przestrzennego, w przeciwieństwie do funkcji liniowych, powoduje również możliwość lepszej projekcji czasowej, tzn. przypisania mniejszych możliwych taktów, chociażby ze względu na możliwość bardziej równomiernego przypisania(rozłożenia) operacji do elementów przetwarzających macierzy czy kontekstu układu wielokontekstowego. Algorytm ewolucyjny dokonuje odwzorowania przestrzennego, następnie realizowane jest odwzorowanie czasowe oraz minimalizowany jest czas realizacji całego algorytmu w zadanej architekturze. W pracy przedstawiono projektowanie uwzględniające przede wszystkim minimalizację czasu realizacji, jednak istnieje możliwość łatwej modyfikacji algorytmu ewolucyjnego do projektowania architektur równoległych

z wykorzystaniem wielu kryteriów optymalizacji, jak: średnie obciążenie procesorów, liczba portów wejściowych i wyjściowych macierzy itp.

W celu efektywnej pracy równoległych architektur na platformie FPGA, należy również opracować wydajne architektury jednostek przetwarzających. W przypadku implementacji jednostek arytmetyczno-logicznych ALU (ang. *Arithmetic Logic Unit*) w istniejących układach FPGA, wysoką częstotliwość działania tych jednostek (porównywalną do maksymalnej częstotliwości działania układu FPGA), co oznacza ich wysoką wydajność, można osiągnąć tylko w przypadku organizacji w jednostce ALU potokowego trybu obliczeń. Algorytmy algebry liniowej (np. eliminacja Gaussa, redukcja wsteczna, rozkład LLT Choleskiego) poza najczęściej spotykanymi operacjami mnożenia i sumowania, zawierają również operację dzielenia i często znajduje się ona dodatkowo w ścieżce krytycznej grafu algorytmu, a zatem ich realizacja ma decydujący wpływ na czas realizacji całego algorytmu. Potokowy tryb obliczeń jest efektywny wyłącznie w przypadku wykonania tej samej operacji na wektorze danych wejściowych (nie na pojedynczej parze danych), niestety operacje dzielenia są w większości algorytmów algebry liniowej właśnie pojedyncze. Ponadto wbudowane bloki DSP, których w jednym układzie FPGA może być nawet kilkaset, są dostosowane do efektywnego wykonania operacji mnożenia i mnożenia z dodawaniem liczb stałoprzecinkowych, natomiast do realizacji operacji dzielenia nie można ich zastosować bezpośrednio. Pewną alternatywą dla tradycyjnego sposobu realizacji sprzętowej operacji x/y jest obliczenie wartości $1/y$, a następnie wymnożenie wyniku tej operacji przez wartość x. Jednak metoda obliczenia wartości $1/y$ jest metodą iteracyjną (wymagająca wykonania dwóch operacji mnożenia i jednego dodawania w każdej iteracji), w której liczba iteracji zależy od wymaganej dokładności wyniku[1]. Podsumowując, realizacja zmiennoprzecinkowych bloków operacyjnych wymaga wykorzystania dużej liczby komórek CLB układu FPGA, przy czym ta liczba drastycznie wzrasta(nawet 10-krotnie) w przypadku realizacji bloków dzielenia lub bloków ALU z możliwością wykonania w/w operacji, ponieważ w tym przypadku nie da się bezpośrednio wykorzystać wbudowanych bloków mnożących wchodzących w skład bloków DSP. Pomimo możliwości wykorzystania wbudowanych bloków mnożących, czas trwania operacji dzielenia pary liczb zmiennoprzecinkowych przedstawionych w standardowych formatach IEEE 754 *single* (32-bitowych), a tym bardziej *double* (64-bitowych) jest kilkadziesiąt razy dłuższy od czasu wykonania mnożenia tych liczb.

Propozycją autora jest stosowanie bloków operacyjnych i jednostek ALU działających w arytmetyce ułamkowej RFA (Rational Fraction Arithmetic). Kilka takich bloków i jednostek autor opracował i przetestował w latach 2007-2008, będąc członkiem zespołu prowadzącego (w ramach grantu MNiSW) badania nad zaletami stosowania arytmetyki ułamkowej w jednostkach ALU, realizujących algorytmy algebry liniowej, przeznaczonych do implementacji w układach FPGA. W ramach pracy w projekcie wykazano, że zastosowanie arytmetyki ułamkowej w w/w jednostkach ALU pozwala zmniejszyć ich złożoność sprzętową w porównaniu do odpowiednich jednostek przetwarzających działających w arytmetyce stałoprzecinkowej, przy zachowaniu zadanej dokładności obliczeń, dzięki dwukrotnemu

zmniejszeniu złożoności sprzętowej kombinacyjnych bloków mnożenia i zamiany bloków dzielenia na bloki mnożenia. Ostatecznie w niniejszej pracy założono, że wymagania dotyczące dokładności przeprowadzanych obliczeń nie są zbyt wysokie, czyli nie jest wymagane stosowanie formatu zmiennoprzecinkowego 64-bitowego Double IEEE 754. Za wystarczający uznano np. format 32-bitowy. Zastosowanie arytmetyki ułamkowej pozwala zwiększyć wydajność jednostek przetwarzających, dzięki zmniejszeniu czasu opóźnienia bloków mnożenia i szczególnie dzielenia, możliwości wykorzystania wbudowanych bloków DSP oraz możliwości realizacji, w tym samym obszarze reprogramowalnym SoC, większej liczby bloków operacyjnych. Z drugiej strony arytmetyka ułamkowa pozwala zmniejszyć moc pobieraną przez jednostkę przetwarzającą, szczególnie przez jednostkę równoległą, dzięki zmniejszeniu jej złożoności sprzętowej lub zmniejszeniu częstotliwości zegara systemowego do wartości zapewniającej jednostce pożądaną wydajność.

Główne tezy rozprawy:

1. Stosowanie opracowanych przez autora algorytmów ewolucyjnych wspomagających zrównoleglenie algorytmów algebry liniowej pozwala doprowadzić czas ich realizacji, w układach reprogramowalnych, do wartości zbliżonej do ścieżki krytycznej grafu algorytmu lub do wartości mniejszych w porównaniu do wartości uzyskiwanych z wykorzystaniem innych znanych metod.
2. Stosowanie arytmetyki ułamkowej w jednostkach arytmetyczno-logicznych akceleratorów przeznaczonych do realizacji algorytmów algebry liniowej w nowoczesnych układach FPGA pozwala zwiększyć ich wydajność lub zmniejszyć ich złożoność sprzętową w porównaniu do odpowiednich jednostek przetwarzających działających w arytmetyce stałoprzecinkowej, przy zachowaniu porównywalnej dokładności obliczeń.

1.6. Układ pracy

W rozdziale pierwszym przedtawiono zakres tematyczny rozprawy oraz uzasadniono aktualność realizowanych badań. Scharakteryzowano możliwości i najważniejsze cechy nowoczesnych układów FPGA. Przedstawiono operacje arytmetyczne najczęściej występujące w algorytmach algebry liniowej oraz scharakteryzowano zależności informacyjne w tych algorytmach. Opisano współczesne metody implementacji podstawowych operacji arytmetycznych występujących w algorytmach algebry liniowej oraz przedstawiono problemy w ich realizacji z wykorzystaniem arytmetyki stało i zmiennoprzecinkowej. Uzasadniono zastosowanie proponowanej arytmetyki ułamkowej na platformie FPGA do realizacji operacji arytmetycznych. Przedstawiono również krótką charakterystykę istniejących metod projektowania architektur równoległych w postaci macierzy procesorowych. Zdefiniowano cel pracy oraz przedstawiono postawione tezy.

W rozdziale drugim przedstawiono nową koncepcję równoległej implementacji algorytmów algebry liniowej z wykorzystaniem wielokontekstowych układów reprogramowalnych oraz scharakteryzowano metodę ich projektowania. Opisano implementację programową tej

metody oraz przedstawiono wyniki odwzorowania wybranych algorytmów algebry liniowej dla macierzy pasmowych oraz porównano je z wartościami krytycznymi, charakteryzującymi rozwiązanie optymalne. Przedstawiono również metodę projektowania klasycznych macierzy procesorowych, scharakteryzowano jej implementację programową, zestawiono otrzymane wyniki projektowania oraz porównano je z wynikami uzyskanymi z wykorzystaniem innej znanej metody. Scharakteryzowano również najważniejsze wady i zalety proponowanych metod projektowania architektur równoległych.

W rozdziale trzecim przedstawiono zaproponowaną arytmetykę ułamkową, przeznaczoną do implementacji operacji arytmetycznych na platformie FPGA. Przedstawiono wyniki implementacji podstawowych operacji algorytmów algebry liniowej w arytmetyce ułamkowej, otrzymane wyniki porównano z parametrami analogicznych bloków zaimplementowanych z wykorzystaniem arytmetyki stało i zmiennoprzecinkowej. Przedstawiono wybrane autorskie projekty akceleratorów przeznaczonych do realizacji algorytmu redukcji wstecznej oraz eliminacji Gaussa.

W podsumowaniu krótko scharakteryzowano wady i zalety proponowanych rozwiązań. Wskazano najważniejsze obszary badań, o które, zdaniem autora, warto byłoby uzupełnić przedstawioną rozprawę oraz wskazano dalsze możliwe perspektywy badań. Przedstawiono konkluzję o zrealizowaniu celu pracy.

2. Projektowanie i optymalizacja równoległych architektur akceleratorów z wykorzystaniem algorytmów ewolucyjnych i programowania z ograniczeniami

2.1. Nowa koncepcja architektury macierzy procesorowej dostosowana do realizacji w wielokontekstowych układach FPGA (osobliwości organizacji obliczeń, zalety i wady nowej architektury)

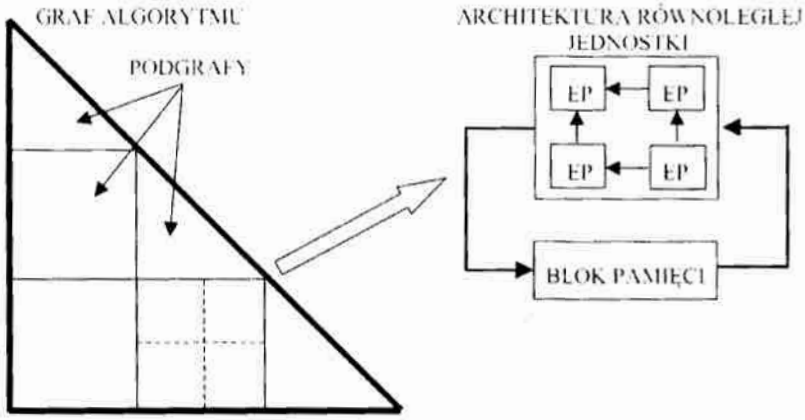
W wyniku bardzo dynamicznego rozwoju układów reprogramowalnych obecnie istnieje możliwość implementacji w pojedynczym układzie dziesiątek, a nawet setek wybranych operacji arytmetycznych. Pojawiły się także koncepcje i architektury wielokontekstowych układów reprogramowalnych. Dzięki takim architekturom istnieje możliwość przeprogramowania układu w trakcie jego działania. Powyższe możliwości nowoczesnych układów reprogramowalnych pozwalają na realizację nowej koncepcji obliczeń równoległych.

Przyspieszenie działania zadanego algorytmu algebry liniowej można uzyskać, m. in. poprzez przetwarzanie równoległe. W przypadku algorytmów algebry liniowej, przetwarzanie równoległe oznacza, że w tym samym momencie czasu, wykonywana jest więcej niż jedna operacja arytmetyczna. Przyspieszenie realizacji zależne jest od tego ile operacji może być wykonane jednocześnie, a z kolei możliwa liczba jednocześnie wykonywanych operacji zależy od dostępnych zasobów oraz od zależności informacyjnych w algorytmie. Zależności informacyjne przedstawiane są za pomocą grafów zależności informacyjnych i obrazują kolejność wykonywania operacji(rys. 2).

Najszybsza równoległa realizacja algorytmu możliwa jest np. w strukturze odpowiadającej dokładnie budowie grafu zależności informacyjnych, tzn. w takiej architekturze w której danemu węzłowi grafu(operacji) przypisany jest jeden element przetwarzający, który realizuje tylko ten węzeł(operację). Taka struktura jest odzwierciedleniem grafu zależności informacyjnych i przetwarza dane w sposób systoliczny, przy maksymalnym zrównolegleniu obliczeń. Problemem jest tu implementacja wszystkich operacji algorytmu w pojedynczym układzie reprogramowalnym, np. dla dużych rozmiarów macierzy danych wejściowych algorytmów algebry liniowej, przy których wykonywane są tysiące operacji arytmetycznych.

W metodach odwzorowania grafów zależności informacyjnych w wyspecjalizowane architektury równoległe wykorzystywane są dwie zasadnicze metody realizacji operacji[1,60] – lokalnie sekwencyjna globalnie równoległa (LSGR) lub lokalnie równoległa globalnie sekwencyjna (LRGS). W metodzie LRGS każdy kolejny podgraf odwzorowywany jest w macierz procesorową w której wierzchołki bieżącego podgrafu realizowane są współbieżnie, natomiast podgrafy realizowane są sekwencyjnie. Wszystkie niezbędne wyniki pośrednie, odpowiadające zależnościom informacyjnym pomiędzy podgrafami, przechowywane są w zewnętrznych blokach pamięci. Na rys. 4 przedstawiono przykład

dekompozycji grafu zależności informacyjnej oraz jego odwzorowania zgodnie z metodą LRGS w architekturę równoległą zawierającą wiele elementów przetwarzających EP.



Rysunek 4. Przykład stosowania metody LRGS (źródło [1] str. 61)

Proponowana w rozprawie koncepcja obliczeń wykorzystuje metodę LRGS, w której operacje kolejnych podgrafów realizowane są w kolejnych kontekstach układu reprogramowalnego.

Realizacja równoległych jednostek przetwarzających w wielokontekstowych układach FPGA może być bardziej efektywna w porównaniu do ich realizacji w zwykłych układach FPGA pod względem wymaganej objętości układu FPGA oraz wydajności systemu [82]. Jednak efektywność ta zależy od sposobu podziału struktury całego systemu na pewną liczbę podstruktur, lub sposobu projektowania tych podstruktur, który ma zapewnić ich porównywalną złożoność sprzętową przy zachowaniu wymaganej wydajności jednostki. Istniejące metody podziału struktury systemu na *s* podstruktur, np. [1, 83], działają na poziomie opisu VHDL systemu i nie są przeznaczone do podziału architektur jednostek równoległych. Natomiast w pracy [1] zaproponowano, zamiast próby podziału gotowego projektu równoległej jednostki na s podukładów (podstruktur), opracowywanie projektów jej *s* podstruktur w oparciu o graf zależności informacyjnych, reprezentującego algorytm który jednostka ma realizować. Ta idea została zrealizowana w pracy[82] w postaci dwuetapowej metody otrzymania podstruktur równoległej jednostki z architekturą macierzy procesorowej. Niestety, wadą opisanej metody jest to, że jej pierwszy etap realizowany jest w sposób heurystyczny, nie pozwalający na zautomatyzowanie. Poza tym, otrzymanie architektur macierzy procesorowych odbywa się w oparciu o liniowe i nieliniowe funkcje odwzorowania przestrzennego i szeregującego, które nie zawsze gwarantują uzyskanie pożądanej wydajności jednostki. W metodach uzyskiwania architektur równoległych opisanych w pracach [64, 82] projektant definiuje funkcję projekcji przestrzennej i czasowej, najczęściej w postaci wektora, co uniemożliwia całkowitą automatyzację tego procesu. Proponowana metoda polega na automatycznym generowaniu projekcji przestrzennej oraz automatycznym przypisaniu poszczególnym operacjom dla tej projekcji możliwie najmniejszego numeru taktu w którym dana operacja ma zostać wykonana. Głównym

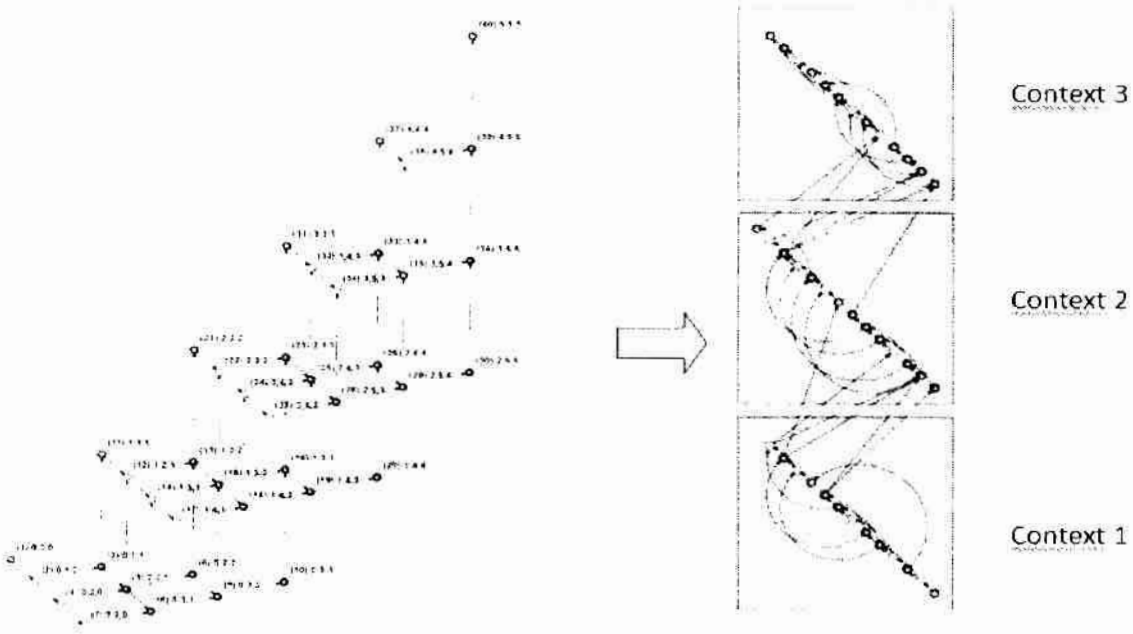
kryterium optymalizacji jest minimalna liczba taktów niezbędna do realizacji całego algorytmu algebry liniowej we wszystkich kontekstach układu reprogramowalnego dla różnych projekcji przestrzennych przy założeniu, że wszystkie operacje(węzły) realizowane są w jednym takcie lub makrotakcie tj. kroku.

W proponowanej w rozprawie koncepcji obliczeń strukturę całego złożonego systemu dzieli się na p podstruktur o porównywalnej złożoności sprzętowej w ten sposób, aby wyniki obliczeń produkowane przez i-tą podstrukturę były wykorzystane jako dane wejściowe w $(i+1)$-ej podstrukturze. Podział struktury całego systemu na podstruktury sprowadza się do podziału całego grafu zależności informacyjnych na kolejne podgrafy, przy zachowaniu zależności informacyjnych oraz pewnych ograniczeń. W takim przypadku system może być zrealizowany w znacznie mniejszym (do p razy) p-kontekstowym układzie FPGA. W poszczególnych blokach pamięci konfiguracyjnej PK takiego układu zapisane są konfiguracje wszystkich podstruktur systemu. W trakcie obliczeń wywołanie właściwych podstruktur odbywa się poprzez uaktywnienie odpowiednich bloków PK, natomiast wyniki operacji zrealizowanych w poprzedniej podstrukturze, niezbędnie do realizacji operacji w kolejnej mogą być przechowywane w pamięci zbudowanej na wewnętrznych blokach BRAM układu reprogramowalnego[39]. Czas realizacji zadanego algorytmu algebry liniowej w takim systemie zależy jednak od podziału struktury systemu na podstruktury, co odpowiada podziałowi grafu tego algorytmu na podgrafy.

Dekompozycja grafu zależności informacyjnej dla wielkontekstowych układów reprogramowalnych (rys 5) posiada kilka ograniczeń dla projekcji przestrzennej. Pierwszym ograniczeniem jest warunek przyczynowości i lokalności, tzn. wszystkie operacje niezbędne do wykonania bieżącej operacji muszą być wykonane bezpośrednio w poprzednim lub bieżącym kontekście układu reprogramowalnego. W praktyce oznacza to uzyskanie argumentów dla bieżących operacji w tym samym lub poprzednim kontekście (przyczynowość) oraz aby dane zapisane w pamięci przechowywane na czas zmiany kontekstu układu musiały być pamiętane tylko dla następnej konfiguracji (lokalność). Ograniczenia te zapewniają poprawną kolejność wykonywanych operacji oraz zmniejszenie wykorzystanej pamięci, niezbędnej do przechowywania danych przekazywanych z bieżącego kontekstu układu do kolejnego. Kolejnym ograniczeniem projekcji przestrzennej jest ograniczenie w postaci maksymalnej wielkości pojedynczego kontekstu (analogia do BPP – ang. *bin packing problem*). Układy reprogramowalne zawierają ograniczone zasoby w postaci komórek programowalnych CLB, wbudowanych bloków DSP oraz wbudowanych bloków pamięci RAM. Dlatego przed dokonaniem dekompozycji grafu na podgrafy, określane są niezbędne zasoby sprzętowe(komórki CLB i bloki DSP) do realizacji każdego rodzaju operacji arytmetycznej w algorytmie. Na tej podstawie szacowany jest rozmiar podstruktury realizującej wszystkie operacje odpowiedniego podgrafu przy zachowaniu wybranej reprezentacji i dokładności obliczeń. Wielkość podstruktury nie może przekraczać zasobów dostępnych dla przyjętego do implementacji modelu układu reprogramowalnego, z pewnym przyjętym marginesem, gdyż trudno jest, z wykorzystaniem istniejących narzędzi do syntezy,

wykorzystać wszystkie dostępne zasoby układu reprogramowalnego. Parametrami proponowanej metody będą więc: maksymalny rozmiar kontekstu, określany na podstawie wyboru modelu docelowego układu reprogramowalnego oraz rozmiary bloków realizujących wszystkie rodzaje operacji arytmetycznych występujących w danym algorytmie (mierzone w CLB) w wybranej arytmetyce przy zachowaniu pożądanej dokładności obliczeń. Zamiast określenia maksymalnego rozmiaru kontekstu istnieje również możliwość zdefiniowania liczby kontekstów układu reprogramowalnego.

Algorytmy mogą być realizowane w pojedynczym wielokontekstowym układzie reprogramowalnym, lub w strukturze składającej się z wielu połączonych takich układów. Do realizacji algorytmu w pojedynczym układzie wielokontekstowym dokonywana jest dekompozycja gafu zależności informacyjnej w wektor kolejnych kontekstów układu (rys 5).



Rysunek 5. Graf zależności informacyjne dla algorytmu rozkładu macierzy pasmowej LLT metodą Cholesky'ego (rozmiar macierzy N=6, szerokość pasma L=4) i jego odwzorowanie w wektor kontekstów dla układu reprogramowalnego.

Dla kilku połączonych szeregowo układów wielokontekstowych dekompozycji grafu należy dokonać np. do macierzy kontekstów , której jeden z rozmiarów odpowiada liczbie układów. Oczywiście dekompozycji należy dokonać przy zachowaniu podobnych ograniczeń, przy nieco zmienionym ograniczeniu przyczynowości. Możliwa jest również realizacja algorytmu w macierzy wielokontekstowych układów, wówczas dekompozycji grafu należałoby dokonać do np. trójwymiarowej struktury kontekstów układu.

Obliczenia wybranego algorytmu algebry liniowej w tak zaprojektowanej architekturze(rys.5) realizowane są w kilku etapach. Na pierwszym etapie konfigurowany jest układ zgodnie z zależnościami informacyjnymi w pierwszym podgrafie, wczytywane są dane wejściowe, wykonywane są operacje w zaprogramowanej strukturze i zapamiętywane są dane

wyjściowe w blokach pamięci wbudowanej. W kolejnym etapie układ przeprogramowany jest w celu realizacji operacji kolejnego podgrafu, następuje odczytanie danych z pamięci oraz realizacja bieżących operacji. W końcu dane otrzymane podczas realizacji ostatniego podgrafu wyprowadzane są na zewnątrz układu lub do innej część systemu przetwarzania danych.

Nowa koncepcja równoległej realizacji obliczeń zaproponowana przez autora rozprawy posiada kilka istotnych zalet, ale również i pewnych wad. Do zalet można zaliczyć możliwość realizacji algorytmów w czasie(mierzonym w taktach lub makrotaktach) zbliżonym do wartości minimalnej, określonej ścieżką krytyczną grafu zależności informacyjnej, gdyż algorytm realizowany jest w prawie maksymalnie możliwy równoległy sposób, natomiast czas przeprogramowania układu wynosi zaledwie kilka taktów zegarowych. Kolejną zaletą takiej architektury jest praktycznie brak bloków sterowania i linii do przekazania sygnałów sterujących dla elementów przetwarzających architektury równoległej, gdyż każdy element wykonuje tylko jedna operację. Proponowana koncepcja posiada również wadę. Dokonywana jest synteza architektury odpowiadającej budowie grafu zależności informacyjnej do układu reprogramowalnego . W celu zachowania wysokiej wydajności struktury implementowanej na platformie FPGA niezbędne jest zachowanie lokalności połączeń, co zapewnia wysoką maksymalną częstotliwość pracy układu. Problematyczne może być zachowanie lokalności połączeń w przypadku implementacji np. trójwymiarowych grafów zależności informacyjnych w dwuwymiarową strukturę układu FPGA. Z tego powodu proponowaną koncepcję organizacji obliczeń zaleca się do realizacji algorytmów o dwuwymiarowych grafach zależności informacyjnych, jak np. algorytmy rozwiązywania układów równań liniowych metodą podstawienia, redukcji wstecznej, Gaussa-Seidela, mnożenia macierzy czy splotu dwóch funkcji. Dla algorytmów o trójwymiarowych grafach zależności informacyjnych nową koncepcję proponuje się jedynie dla macierzy pasmowych dla niewielkich szerokościach pasma, które są często wykorzystywane w różnego rodzaju obliczeniach numerycznych.

Podsumowując propozycją autora jest wykorzystanie algorytmu ewolucyjnego i programowania z ograniczeniami do podziału grafów zależności informacyjnych na podgrafy, które realizowane są w kolejnych kontekstach układów reprogramowalnych, przy zachowaniu opisanych ograniczeń lokalności, przyczynowości oraz maksymalnego rozmiaru kontekstu, natomiast głównym kryterium optymalizacji jest czas realizacji wejściowego algorytmu algebry liniowej.

W kolejnych podrozdziałach przedstawiono otrzymany czasy(mierzone w taktach) realizacji wybranych algorytmów dla przeprowadzonych dekompozycji grafów zależności informacyjnych oraz porównano go z czasami realizacji otrzymanymi dla innych równoległych architektur, a także opisano algorytm wykorzystywany do przeprowadzania dekompozycji.

2.2. Projektowanie i analiza architektur nowego typu z wykorzystaniem algorytmów genetycznych i programowania z ograniczeniami

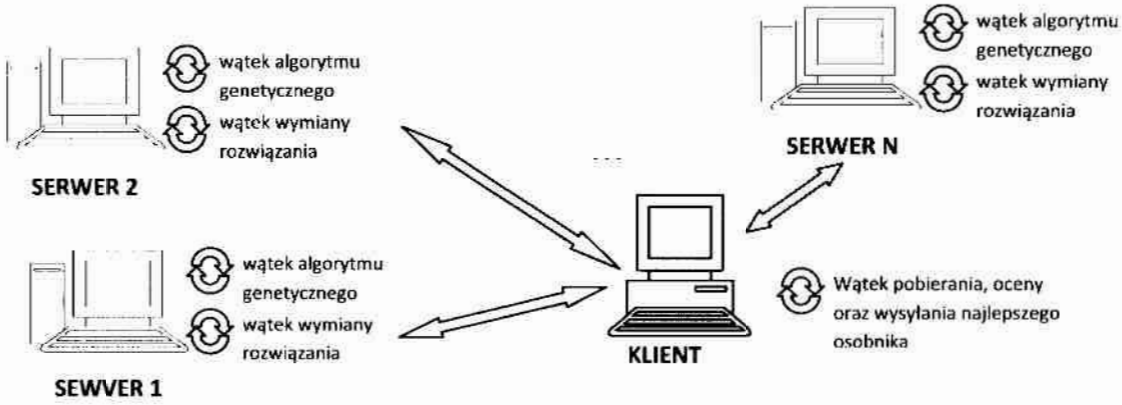
Obecnie istnieje wiele algorytmów służących do rożnego rodzaju podziałów i kolorowania grafów[84], jak: kolorowanie węzłów, kolorowanie łuków, kolorowanie z wykorzystaniem *k* kolorów. Pierwsze algorytmy kolorowania grafów opracowano już pod koniec lat 60-tych[85,86] , w następnych latach pojawiło się wiele kolejnych metod, w tym również metody wykorzystujące algorytmy genetyczne[87,88,89,90]. Większość dostępnych metod opracowywano dla dobrze znanego, standardowego algorytmu kolorowania węzłów grafów, w którym należy pokolorować wszystkie węzły grafu z wykorzystaniem minimalnej liczby kolorów (liczba chromatyczna), tak aby sąsiadujące węzły zawsze były pokolorowane różnymi kolorami. Trudno jednak znaleźć metodę dla grafów ukierunkowanych z wagami dla węzłów, będącą połączeniem algorytmu pakowania(ang. *bin packing problem*) z algorytmem kolejkowania (ang. *scheduling algorithm*), przy zachowaniu wcześniej opisanych ograniczeń przyczynowości, lokalności i pojemności. Z kolei metoda podziału grafu zależności informacyjnych opisana w pracy [82] nie pozwala na jej automatyzację, a dodatkowo w przypadku zastosowania liniowych lub nieliniowych funkcji projekcji przestrzennych trudne jest uzyskanie dowolnej struktury architektury równoległej. Niektóre metody wykorzystują specyficzne dla danego algorytmu przekształcenia grafów[1], które powodują większy wpływ na kształt równoległej architektury i na parametry elementów przetwarzających, jednak nie da się tych przekształceń uogólnić na różne algorytmy.

W dalszej części pracy opisano opracowany algorytm genetyczny służący do odwzorowania grafów zależności informacyjnych algorytmów algebry liniowej w równoległe architektury przeznaczone do implementacji w wielokontekstowych układach reprogramowalnych. Odwzorowanie to sprowadza się do przypisania(pokolorowania) każdego węzła grafu do wybranego kontekstu układu reprogramowalnego, przy zachowaniu wcześniej opisanych ograniczeń.

Jako parametr proponowanej metody podziału grafu możemy podać liczbę kontekstów lub określić maksymalny rozmiar kontekstu zależny od konkretnego modelu układu. Po określeniu maksymalnego rozmiaru kontekstu oraz określeniu szacowanego rozmiaru struktury niezbędnej do realizacji każdego rodzaju operacji w algorytmie niezbędna liczba kontekstów może być wyznaczona automatycznie. W praktyce minimalną liczbę kontekstów wyznacza się z pewnym założonym marginesem gdyż dla wybranego układu FPGA trudno jest wykorzystać całkowitą dostępną przestrzeń programowalną.

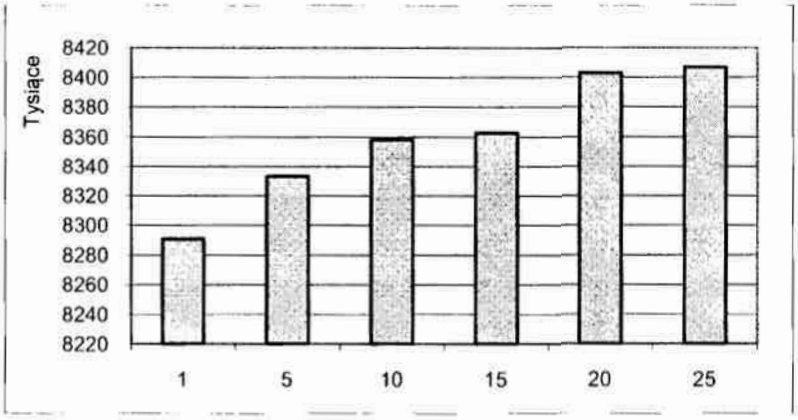
Pierwsze próby wykorzystania algorytmów ewolucyjnych do podziału grafów zależności informacyjnych dla proponowanej metody realizacji algorytmów algebry liniowej przedstawiono w pracy [39], jednak po dalszych badaniach problematyczna okazała się dekompozycja grafów zależności informacyjnych dla większych rozmiarów macierzy wybranych algorytmów algebry liniowej(dla grafów o liczbie węzłów>500) przy zastosowaniu standardowych operatorów rekombinacji (mutacji i krzyżowania). Niektóre algorytmy

genetyczne wykorzystywane do różnych zadań podziału i innych operacji na grafach, w tym do podziału z ograniczeniami z różnymi modyfikacjami operatorów genetycznych są przedstawione w pozycji[91]. Zaznaczono jednak , że algorytmy te były stosowane dla grafów o niezbyt dużej liczbie wezłów rzędu 100 lub 900. W utworzonym przez autora rozprawy algorytmie również zaproponowano pewne modyfikacje operatorów genetycznych oraz wykorzystanie programowania z ograniczeniami (ang. *constraint programming*) co pozwoliło uzyskać wyniki podziału grafów zawierających nawet do 2500 węzłów w założonym czasie obliczeń do 15 min. W celu przyspieszenia obliczeń algorytmu genetycznego zdecydowano się na jego zrównoleglenie co pozwoliło na uzyskanie większej liczby pokoleń(generacji) tego algorytmu w zadanym okresie czasu. Rozważano różne metody zrównoleglenia obliczeń genetycznych, jednak ostatecznie utworzono własną aplikacje rozproszoną typu jeden klient – wiele serwerów obliczeniowych zrealizowanych na popularnych komputerach klasy PC[28](rys.6). W aplikacji tej zastosowano model algorytmu genetycznego podobny do wyspowego(ang. *Island model*)[92,93]. Wykorzystano podobną zasadę niezależnych populacji, jednak nieco inny jest sposób wymiany najlepszego rozwiązania, która zachodzi pomiędzy klientem a wieloma serwerami. Wymiana ta dokonywana jest w stałych odstępach czasu definiowanych przez użytkownika programu. Aplikacja klienta w zadanych odstępach czasu pobiera najlepsze rozwiązania ze wszystkich populacji na wszystkich serwerach, dokonuje ich oceny oraz następnie po selekcji najlepszego osobnika wysyła go do wszystkich populacji.



Rysunek 6. Równoległa implementacja opracowanego algorytmu genetycznego.

Równoległa implementacja algorytmu genetycznego pozwala zwiększyć liczbę obliczanych generacji oraz umożliwia zastosowanie różnych modyfikacji algorytmu na różnych serwerach obliczeniowych jednocześnie. Większa liczba wykorzystywanych serwerów obliczeniowych pozwala szybciej uzyskać lepsze rozwiązania, zwłaszcza dla krótkiego przyjętego czasu działania algorytmu (5, 10 lub 15min)(rys.7).



Rysunek 7. Wartości maksymalizowanej funkcji celu uzyskiwane w ciągu 5 min obliczeń dla różnej wykorzystywanej liczbie serwerów obliczeniowych (źrodło[28]).

2.2.1. Charakterystyka opracowanego algorytmu genetycznego.

Reprezentacja danych.

W opisywanym algorytmie zastosowano kodowanie grup podziałów za pomocą liczb naturalnych. Podziały są reprezentowane jako łańcuchy całkowitoliczbowe o wymiarze *n*, gdzie *n* odpowiada liczbie węzłów grafu zależności informacyjnej, natomiast przedział wartości liczb jest ograniczony od 0 do *m*-1, gdzie *m* jest liczbą kontekstów układu reprogramowalnego. Podobny sposób kodowania podziałów grafów przedstawiono w pozycji [91]. Poniżej przedstawiono przykładowy podział i jego reprezentacje dla grafu i kontekstów z rys 8.

[0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 1 1 1 1 1 1 1 1 1 2 1 2 2 2 1 2 2 2 2 2 2 2 2]

Rysunek 8. Chromosom reprezentujący odwzorowanie grafu w wektor kontekstów przedstawiony na rys5.

W przedstawionej reprezentacji danych każdemu węzłowi grafu odpowiada osobny gen tj. pozycja w chromosomie. Wartość liczby naturalnej reprezentującej poszczególny gen określa natomiast numer kontekstu w którym dany węzeł grafu(operacja algorytmu) będzie realizowany. Przykładowo pierwszej pozycji w chromosomie o wartości „0" odpowiada przypisanie pierwszego węzła grafu do pierwszego kontekstu, których numery indeksowane są od wartości „0". Ostatnia pozycja w chromosomie o wartości „2" odpowiada przypisaniu ostatniego węzła grafu do trzeciego kontekstu układu.

Generowanie populacji początkowej.

W algorytmie zaimplementowano dwie metody generowania populacji początkowej. W pierwszej metodzie wartości były generowane w sposób losowy(losowe przypisanie węzłów grafu do kontekstów), natomiast w drugiej generowano startową populację z wykorzystaniem programowania z ograniczeniami. Wykorzystanie programowania z ograniczeniami miało na celu wygenerowanie w jak najkrótszym czasie rozwiązań dopuszczalnych, które następnie mogły być dalej optymalizowane poprzez algorytm genetyczny. Pozwoliło to na skrócenie czasu w którym program znajdował dopuszczalne

rozwiązania. Dokładne wyniki porównujące wykorzystanie obu metod przedstawiono w dalszej części rozprawy. Cały program realizujący algorytm genetyczny utworzono w technologii .NET Remoting (C#) na platformie Microsoft .NET dlatego w module do programowania z ograniczeniami zdecydowano się na wykorzystanie gotowej biblioteki działającej na tej platformie programowej opracowanej w zespole prof. Andy Chun'a o nazwie NSolver[94]. Moduł generujący początkową populację z wykorzystaniem programowania z ograniczeniami kończył pracę po eksperymentalnie dobranym okresie maksymalnie 2 min lub po wygenerowaniu całej populacji startowej (100 osobników). Zbadano wykorzystanie 2 heurystyk wykorzystywanych do przeszukiwania przestrzeni rozwiązań: heurystykę losową oraz heurystykę *„MinSizeMinValue"*, która polegała na przeszukiwaniu przestrzeni rozwiązań od najmniejszych wartości dla najmniejszych indeksów w chromosomie. Na podstawie eksperymentów stwierdzono, ze heurystyka losowa daje lepsze rezultaty przy ograniczonej liczbie rozwiązań dla mniejszych grafów (o liczbie węzłów <200) natomiast heurystyka *„MinSizeMinValue"* pozwala na szybsze uzyskanie rozwiązań dla większych grafów. Problemem było znalezienie rozwiązania dla liczby kontekstów k>5, co mogło wynikać z binarnej reprezentacji wszystkich węzłów grafu we wszystkich kontekstach, która została wykorzystana ze względu na ograniczoną liczbę operacji na zmiennych tablicowych.

Operatory rekombinacji.

W utworzonym algorytmie genetycznym wykorzystano standardowy algorytm krzyżowania jednopunktowego ze stałym prawdopodobieństwem, którego wartość została eksperymentalnie dobrana na poziomie 0.2. Większy wpływ na zbieżność algorytmu miały modyfikacje wprowadzone w operatorze mutacji. W procesie mutacji zmiana wartości poszczególnego genu powodowała przypisanie operacji algorytmu, odpowiadającej temu genowi, do innego kontekstu układu reprogramowalnego. Początkowo stosowano mutację o stałym prawdopodobieństwie, co powodowało mutowanie różnej liczby pozycji w chromosomie dla różnych rozmiarów grafów. Przy opisanych wcześniej ograniczeniach powodowało to długi okres dochodzenia do dopuszczalnych rozwiązań dla większych grafów (z liczbą węzłów > 500). Pierwszą wprowadzoną modyfikacją było określenie prawdopodobieństwa mutacji pojedynczej pozycji w chromosomie w zależności od rozmiaru grafu, tak aby mutacji podlegała niewielka liczba genów (zmiana przypisania kontekstu dla niewielkiej liczby węzłów). Kolejną zmianą było wprowadzenie zmiennej wartości prawdopodobieństwa dla mutacji w zależności od czasu działania algorytmu[91]. Program był uruchamiany z takim prawdopodobieństwem mutacji aby zmieniała się wartość tylko dla jednej pozycji w chromosomie. Następnie z kolejnymi przedziałami czasu dla których nie zachodziła poprawa najlepszego rozwiązania prawdopodobieństwo mutacji zwiększano w taki sposób, aby zmianie podlegał jeden gen więcej. Następną modyfikacją wprowadzoną do operatora mutacji było ograniczenie przedziału wartości jakie mógł przyjmować gen w chromosomie. Zakres możliwych wartości dla danego genu w chromosomie wyznaczono na podstawie liniowej projekcji z marginesem (-1,+1), tzw. „oknem". Szerokość „okna" mogła

wzrastać z czasem działania algorytmu bez poprawy najlepszego rozwiązania, podobnie jak prawdopodobieństwo mutacji pojedynczego genu w chromosomie. Opisane modyfikacje pozwoliły na zdecydowanie szybsze uzyskiwanie rozwiązań dopuszczalnych (podziałów grafów spełniających opisane wcześniej ograniczenia).

Funkcja oceny.

W prezentowanej metodzie podziału grafu zastosowano dwuetapowy algorytmu genetyczny: etap przed znalezieniem dopuszczalnego rozwiązania oraz etap po znalezieniu takiego rozwiązania. W każdym etapie pracy algorytmu dla każdego osobnika wartość funkcji oceny była maksymalizowana. Przed znalezieniem dopuszczalnego rozwiązania wyznaczana była dla wszystkich osobników liczba niespełnionych warunków lokalności i przyczynowości , czyli wyznaczana jest projekcja przestrzenna. Funkcja oceny F w tym etapie została zdefiniowana eksperymentalnie i została uzależniona od błędów lokalności i przyczynowości oraz błędów przepełnienia. Na tym etapie wartość funkcji oceny była obliczana zgodnie ze wzorem (1):

$$F1 = 1 + EN * (3 * EN - SE) + EN * \frac{CN-OE}{CN} + EN * \frac{GS-CL}{GS} \qquad (1)$$

, gdzie:

EN – liczba łuków grafu,
SE – liczba łuków niespełniających ograniczenia,
CN – liczba kontekstów układu reprogramowalnego,
OE – liczba kontekstów przepełnionych,
GS – rozmiar struktury dla wszystkich operacji grafu (CLB),
CL – różnica wielkości kontekstów(CLB).

Na podstawie eksperymentów funkcję oceny na tym etapie uzależniono również od równomiernego rozłożenia elementów przetwarzających we wszystkich kontekstach (w CLB), co spowodowało szybszą minimalizację błędów przepełnienia.

W drugim etapie pracy algorytmu, po znalezieniu dopuszczalnego rozwiązania wartość funkcji przystosowania była obliczana według jednego z dwóch wzorów w zależności od tego czy osobnik reprezentuje dopuszczalne rozwiązanie czy nie. Dla osobników nie spełniających ograniczeń (niedopuszczalnych) zastosowano funkcję przyjmującą zdecydowanie mniejsze wartości (funkcja kary[91]), natomiast dla osobników spełniających ograniczenia funkcja celu dodatkowo uzależniona była od liczby taktów niezbędnych do realizacji wszystkich kontekstów (całego grafu). W drugim etapie pracy algorytmu genetycznego wartość funkcji przystosowania(oceny) dla osobników spełniających ograniczenia była wyznaczana zgodnie ze wzorem (2):

$$F2 = 3 * EN^2 + CN * EN + EN + EN - T \qquad (2)$$

,gdzie

T – liczba taktów niezbędna do realizacji całego algorytmu algebry liniowej.

Dla osobników nie spełniających ograniczeń (rozwiązania niedopuszczalne) wartość funkcji oceny była wyznaczana na podstawie wzoru (3):

$$F3 = 3 - \frac{SE}{3*EN} - \frac{OE}{CN} \qquad (3)$$

Wartości funkcji F3 są zawsze mniejsze od wartości funkcji F2, co jest odpowiednikiem funkcji kary dla osobników, które nie spełniają ograniczeń, gdyż wartości wszystkich funkcji oceny były maksymilizowane.

Selekcja

W prezentowanym algorytmie wykorzystano selekcję elitarystyczną w celu przekazania najlepszego rozwiązania do kolejnej populacji. W opracowanym modelu równoległym algorytmu genetycznego najlepsze rozwiązania są przekazywane do wszystkich populacji – wysp[92,93]. Model selekcji elitarystycznej zapewnia „utrzymanie" znalezionego najlepszego rozwiązania spełniającego wszystkie ograniczenia (przyczynowości, lokalności oraz wielkości kontekstu), które w procesie mutacji może być łatwo zmienione w rozwiązanie nie spełniające wszystkich ograniczeń (rozwiązanie niedopuszczalne). Selekcja elitarystyczna pozwala na utrzymanie najlepszych uzyskanych rozwiązań co wpływa na zbieżność algorytmu i ma duże znaczenie w przypadku silnych ograniczeń czasowych dla znalezienia dopuszczalnego rozwiązania.

Warunek zakończenia algorytmu genetycznego.

Ze względu na docelowe wykorzystanie prezentowanej metody w generatorze IPCore, służącym do projektowania architektur równoległych dla algorytmów algebry linowej (projekt JGEN – pozdrozdz. 4.2), przyjęto znaczne ograniczenia czasowe dla działania algorytmu. Czas obliczeń algorytmu genetycznego był ograniczony do maksymalnie 15 min. Koniec obliczeń następował również przy braku poprawy najlepszego rozwiązania w czasie 5 min.

2.2.2. Rezultaty dekompozycji grafów zależności informacyjnych uzyskane z wykorzystaniem opracowanego algorytmu genetycznego.

W trakcie realizacji badań przeprowadzono podział grafu zależności informacyjnych dla algorytmu rozkładu macierzy pasmowych metodą Cholesky'ego o szerokościach pasma 3, 5 i 7, które są często spotykanymi wartościami w praktycznych obliczeniach numerycznych. Wstępnego oszacowania rozmiaru implementowanych operacji arytmetycznych algorytmu dokonano z wykorzystaniem generatora Xilinx IPCore Floating Point Generator v.3.0. Po procesie syntezy dla rodziny układów Xilinx Virtex5 uzyskano następujące parametry: dla operacji pierwiastkowania 220 bloków SLICE dla reprezentacji danych zmiennoprzecinkowej pojedynczej precyzji, dla operacji dzielenia 192 i dla operacji mnożenia z odejmowaniem 350 bloków. Na podstawie szacowanych rozmiarów operacji arytmetycznych oraz wielkości docelowego modelu układu reprogramowalnego oszacowano niezbędną liczbę kontekstów. Liczba kontekstów może być wyznaczana automatycznie lub definiowana przez użytkownika programu. W zaprezentowanych wynikach dekompozycji

wykorzystywano zarówno jedną jak i drugą metodę określania liczby kontekstów. Podstawowym parametrem optymalizacji była liczba taktów niezbędna do realizacji operacji we wszystkich kontekstach układu FPGA. W tabelach 1, 2 i 3 zestawiono niezbędną liczbę taktów uzyskaną przy projekcji liniowej i nieliniowej dla różnych rozmiarów macierzy pasmowych otrzymanych zgodnie z metodą [1] z wynikami uzyskanymi przy pomocy proponowanej metody wykorzystującej algorytm genetyczny z losową populacją startową oraz populacją startową uzyskaną za pomocą programowania z ograniczeniami. Liczbę taktów dla projekcji liniowej wyznaczono na podstawie wzoru (4), natomiast dla projekcji nieliniowej na podstawie wzoru (5):

$$T1 = (N-1)(L+1) - 1 \qquad (4)$$

$$T2 = N(N-1)/2 + 2N + 1 \qquad (5)$$

gdzie:

N – rozmiar macierzy, L – szerokość pasma.

Dla każdego zestawu parametrów wejściowych uruchomiono program 10-krotnie, natomiast w zestawieniu umieszczono zarówno wyniki średnie jak i najlepsze.

Tabela 1. Czasy realizacji algorytmu Cholesky'ego dla macierzy pasmowej (szerokość pasma =3) uzyskane w wyniku dekompozycji grafu zależności informacyjnych z wykorzystaniem proponowanej metody oraz projekcji liniowej i nieliniowej.

Szerokość pasma macierzy		3	3	3	3	3	3	3	3
Rozmiar macierzy (NxN)		30	40	50	60	70	80	90	100
Liczba węzłów grafu		172	232	292	352	412	472	532	592
Liczba łuków grafu		20885	27935	34985	42035	49085	56135	63185	70235
Liczba kontekstów		2	3	3	4	5	5	6	7
Max. Rozmiar kontekstu (CLB) XC4VLX100= 12288 CLB		12000	12000	12000	12000	12000	12000	12000	12000
Ścieżka krytyczna grafu		88	118	148	178	208	238	268	298
Liczba taktów dla projekcji liniowej(4)		117	157	197	237	277	317	357	397
Liczba taktów dla projekcji nieliniowej (5)		494	859	1324	1889	2554	3319	4184	5149
Liczba taktów dla proponowanej metody (alg. gen.)	średnia	88	118	148	178	208	238	268	298
	najlepsza	88	118	148	178	208	238	268	298
Liczba taktów dla proponowanej metody (alg. gen. i prog. z ogr.)	średnia	88	118	148	178	208	238	268	298
	najlepsza	88	118	148	178	208	238	268	298

Na podstawie wyników zestawionych w tabeli 1 można zauważyć, że wyniki uzyskane za pomocą proponowanego algorytmu są lepsze od wyników otrzymanych z wykorzystaniem liniowego i nieliniowego odwzorowania przestrzennego, opisanego w metodzie [1]. Można również stwierdzić ze są to wyniki optymalne, gdyż pokrywają sie z wartościami dla ścieżki krytycznej grafu. Dla tej przestrzeni możliwych kombinacji (maksymalnie około 7^{600} – wariacja z powtórzeniami), algorytm zawsze kończył działanie przed upływem 15 min, po okresie 5 min bez poprawy najlepszego rozwiązania.

Wyniki podziału grafu dla macierzy o szerokości pasma równej 5 przedstawiono w tabeli 2. Również w tym przypadku przy wykorzystaniu proponowanej metody uzyskano zdecydowanie lepsze wyniki od wyników uzyskanych zgodnie z metodą [1] i zbliżone do wartości optymalnych, określonych ścieżką krytyczną grafu. W niektórych przypadkach wygenerowanie startowej populacji z wykorzystaniem programowania z ograniczeniami powodowało uzyskanie lepszych rezultatów (np. dla rozmiaru macierzy 40 i 50) w porównaniu z losową populacją startową. Były też przypadki uzyskania gorszego rezultatu (dla rozmiaru 30 i 60), gdyż prawdopodobnie uzyskane rozwiązanie początkowe przesunęło obszar przeszukiwań w minimum lokalne, co spowodowało zbieżność algorytmu wokół rozwiązania suboptymalnego w krótkim, zadanym przedziale czasowym działania algorytmu. Należy jednak podkreślić iż programowanie z ograniczeniami pozwalało na uzyskanie dopuszczalnych rozwiązań w krótszym czasie.

Tabela 2. Czasy realizacji algorytmu Cholesky'ego dla macierzy pasmowej (szerokość pasma =5) uzyskane w wyniku dekompozycji grafu zależności informacyjnych z wykorzystaniem proponowanej metody oraz projekcji liniowej i nieliniowej.

Szerokość pasma macierzy		5	5	5	5	5	5	5	5
Rozmiar macierzy (NxN)		30	40	50	60	70	80	90	100
Liczba węzłów grafu		410	560	710	860	1010	1160	1310	1460
Liczba łuków grafu		28950	39050	49150	59250	69350	79450	89550	99650
Liczba kontekstów		2	3	4	4	5	6	6	7
Max. Rozmiar kontekstu (CLB) XC4VLX100= 12288 CLB		16000	16000	16000	16000	16000	16000	16000	16000
Ścieżka krytyczna grafu		88	118	148	178	208	238	268	298
Liczba taktów dla projekcji liniowej(4)		175	235	295	355	415	475	535	595
Liczba taktów dla projekcji nieliniowej (5)		494	859	1324	1889	2554	3319	4184	5149
Liczba taktów dla proponowanej metody (alg. gen.)	średnia	88,5	122,5	153,3	183,3	214,8	246	285	317,8
	najlepsza	88	120	152	182	214	245	285	316
Liczba taktów dla proponowanej metody (alg. gen. I prog. z ogr.)	średnia	89,0	120,8	152,3	187	214,8	246	285	317,8
	najlepsza	89	120	151	187	214	245	285	316

Kolejnymi badanymi grafami były grafy algorytmu Cholesky'ego dla macierzy pasmowej o szerokości pasma = 7. Wyniki podziału tych grafów przedstawiono w tabeli 3.

Tabela 3. Czasy realizacji algorytmu Cholesky'ego dla macierzy pasmowej (szerokość pasma =7) uzyskane w wyniku dekompozycji grafu zależności informacyjnych z wykorzystaniem proponowanej metody oraz projekcji liniowej i nieliniowej.

Szerokość pasma macierzy		7	7	7	7	7	7	7	7
Rozmiar macierzy (NxN)		30	40	50	60	70	80	90	100
Liczba węzłów grafu		728	1008	1288	1568	1848	2128	2408	2688
Liczba łuków grafu		39835	54385	68935	83485	98035	112585	127135	141685
Liczba kontekstów		3	3	4	5	6	7	7	8
Max. Rozmiar kontekstu (CLB) XC4VLX100= 12288 CLB		20000	20000	20000	20000	20000	20000	20000	20000
Ścieżka krytyczna grafu		88	118	148	178	208	238	268	298
Liczba taktów dla projekcji liniowej(4)		233	313	393	473	553	633	713	793
Liczba taktów dla projekcji nieliniowej (5)		494	859	1324	1889	2554	3319	4184	5149
Liczba taktów (alg. gen.)	średnia	95,8	131,5	171,5	206,0	258,0	280,8	310,5	367,8
	najlepsza	95	130	171	205	258	280	309	367
Liczba taktów (alg. gen. I prog. z ogr.)	średnia	97,0	129,0	167,0	206,0	258,0	280,8	310,5	367,8
	najlepsza	97	128	167	205	258	280	309	367

Również w tym przypadku widoczna jest zdecydowana przewaga proponowanego algorytmu nad metodą liniową i nieliniową projekcji przestrzennej opisanej w pozycji[1]. Warto zauważyć ze dokonywano podziału nawet dla grafów o liczbie węzłów ok. 2700, przy zachowaniu opisywanych ograniczeń czasowych (15 min.). Na podstawie wyników przedstawionych w tabelach 1, 2 i 3 można zauważyć, że wraz ze wzrostem szerokości pasma macierzy, a co za tym idzie liczby węzłów w grafie, uzyskiwane projekcje wymagały większej liczby taktów do realizacji całego algorytmu Cholesky'ego. Uzyskiwano jednak ciągle wyniki zbliżone do wartości optymalnych (ścieżki krytycznej grafu).

Kolejnym przedmiotem badań w ramach rozprawy doktorskiej był wpływ generowania startowej populacji z wykorzystaniem programowania z ograniczeniami na szybkość uzyskiwania dopuszczalnego rozwiązania przez algorytm genetyczny dla opisywanego problemu. Podczas badań zbierano dane o najlepszych rozwiązaniach po upływie 1, 2 i 3 minut czasu działania algorytmu. Wyniki porównania pracy algorytmu z losową i wygenerowaną, przy użyciu programowania z ograniczeniami, populacją startową przedstawiono w tabeli 4.

Tabela 4. Rezultaty podziału grafu zależności informacyjnych algorytmu Cholesky'ego dla macierzy pasmowych uzyskane po 1, 2 I 3 minutach działania algorytmu genetycznego z wygenerowana z wykorzystaniem programowania z ograniczeniami oraz losową populacja początkową.

Szerokość pasma		5	7	7	7	5	7	7	7	5	7	7	7
Rozmiar macierzy (NxN)		60	30	40	50	60	30	40	50	60	30	40	50
Liczba węzłów grafu		860	728	1008	1288	860	728	1008	1288	860	728	1008	1288
Liczba łuków grafu		59250	39835	54385	68935	59250	39835	54385	68935	59250	39835	54385	68935
Liczba kontekstów		4	3	3	4	4	3	3	4	4	3	3	4
Max. rozmiar kontekstu XC4VLX100= 12288 CLB		16000	20000	20000	20000	16000	20000	20000	20000	16000	20000	20000	20000
Ścieżka krytyczna grafu		178	88	118	148	178	88	118	148	178	88	118	148
Czas dekompozycji		1 min				2 min				3 min			
Liczba taktów dla proponowanej met. (alg. gen.)	średnia	189	99	-	-	188	98	136	172	186	97	134	171
	najlepsza	190,4	100,4	-	-	188,4	99,4	137,0	172,0	187,1	98,1	135,4	171,1
	rozw. (%)	90	70	0	0	90	70	80	40	90	80	50	80
Liczba taktów dla pr. met. (alg. gen. I prog. z ogr.)	średnia	187	99	132	173	187	98	131	172	187	98	131	172
	najlepsza	187,2	100,4	132,4	173	187,2	99,2	132	172,8	187,2	98,6	131,6	172,4
	rozw.(%)	100	100	100	100	100	100	100	100	100	100	100	100

Na podstawie wyników zestawionych w tabeli 4 można zauważyć, że przy bardzo ostrych ograniczeniach czasowych działania algorytmu genetycznego wyniki uzyskiwane wykorzystaniem programowania z ograniczeniami do generowania początkowej populacji algorytmu genetycznego są zazwyczaj lepsze, jednak przewaga ta maleje wraz ze wzrostem czasu obliczeń.

Nowoczesne układy FPGA, np. z rodziny Xilinx Virtex6 zawierają ogromną przestrzeń programowalną, wbudowane bloki pamięci RAM oraz nawet do 2 tysięcy wbudowanych

bloków DSP, zawierających dwa bloki mnożące (25 x 18 bitów) z akumulatorem[14]. Wykorzystywanie wbudowanych bloków pozwala zmniejszyć wykorzystywaną uniwersalną przestrzeń programowalną układu, zmniejszyć pobór mocy oraz często pozwala również zwiększyć wydajność projektowanego systemu[29,30]. W związku z tym przeprowadzono badania mające na celu określenie dokładnych parametrów implementacji operacji arytmetycznych przykładowych algorytmów, również z wykorzystaniem wbudowanych bloków DSP, ponieważ od tych parametrów zależna jest niezbędna liczba kontekstów przy realizacji systemu w wielokontekstowym układzie FPGA. Ze względu na dużą liczbę operacji arytmetycznych oraz niezbędną dokładność prowadzonych obliczeń dla algorytmów algebry liniowej wykorzystano zmiennoprzecinkową reprezentację liczbową pojedynczej precyzji. Testowe bloki operacyjne zostały wygenerowane z wykorzystaniem generatora „Xilinx Floating-point v.3.0 IPCore" ze środowiska Xilinx ISE 11.5 dla układów z nowoczesnej rodziny Virtex6. W tabelach 5 i 6 zaprezentowano wyniki implementacji operacji arytmetycznych algorytmów rozkładu Cholesky'ego LL^T oraz Gaussa LU.

Tabela 5. Parametry implementacji operacji odejmowania i mnożenia dla układów Xilinx Virtex6 z wykorzystaniem zmiennoprzecinkowej, pojedyńczej pracyzji reprezentacji danych.

Odejmowanie			Mnożenie		
bloki Slice	**DSP48**	**opóźnienie**	**bloki Slice**	**DSP48**	**opóźnienie**
166	0	0	81	1	0
169	0	2	81	1	2
154	0	4	84	1	4
201	0	8	116	1	8

Tabela 6. Parametry implementacji operacji dzielenia i pierwiastkowania dla układów Xilinx Virtex6 z wykorzystaniem zmiennoprzecinkowej, pojedyńczej pracyzji reprezentacji danych.

Dzielenie			Pierwiastkowanie		
bloki Slice	**DSP48**	**opóźnienie**	**bloki Slice**	**DSP48**	**opóźnienie**
227	0	0	143	0	0
213	0	2	218	0	2
145	0	4	122	0	4
102	0	8	77	0	8

Ze względu na ograniczoną liczbę wbudowanych bloków DSP zdecydowano się na wykorzystanie tych bloków do realizacji operacji mnożenia, gdyż właśnie dla tej operacji były największe oszczędności wykorzystywanych bloków programowalnych układu – *Slices*. Ostatecznie zdecydowano o wykorzystaniu jednego wbudowanego bloku DSP dla każdej operacji mnożenia. Dla wielu różnych algorytmów algebry liniowej więcej operacji arytmetycznych będzie mogło być implementowane z wykorzystaniem bloków DSP, co umożliwi dalsze zredukowanie niezbędnej przestrzeni programowalnej, co w konsekwencji doprowadzi również do zredukowania liczby kontekstów niezbędnej do realizacji

[14] http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

zadanego algorytmu. Przy mniejszej liczbie kontekstów rozwiązanie problemu podziału grafu jest łatwiejsze do zrealizowania przez obie proponowane metody, co pozwala na projektowanie akceleratorów dla większych rozmiarów macierzy wejściowych. Również opóźnienie („latency" - mierzone w taktach) projektowanych bloków funkcjonalnych ma zasadniczy wpływ na parametry implementacji. Ostatecznie zdecydowano o przyjęciu wartości opóźnienia równej 4 takty, gdyż dla tej wartości wszystkie testowane bloki cechowały się względnie wysokim stosunkiem wykorzystywanej powierzchni do liczby taktów opóźnienia. Parametry implementowanych operacji przedstawiono w tabeli 7 dla algorytmu rozkładu LL^T Cholesky'ego oraz w tabeli 8 dla algorytmu LU Gaussa. Jednakowa wartość opóźnienia niezbędna jest do synchronizacji obliczeń w całej równoległej architekturze.

Tabela 7. Parametry implementacji operacji arytmetycznych algorytmu Cholesky'ego LL^T dla reprezentacji zmiennoprzecinkowej pojedyńczej precyzji uzyskane z wykorzystaniem generatora IPCore „Xilinx Floating-point v.3.0"

Operacje algorytmu Cholesky'ego LLT	DSP48	Bloki Slice	opóźnienie
mnożenie z odejmowaniem	1	250	4
dzielenie	0	145	4
pierwiastkowanie	0	122	4

Tabela 8. Parametry implementacji operacji arytmetycznych algorytmu Gaussa LU dla reprezentacji zmiennoprzecinkowej pojedyńczej precyzji uzyskane z wykorzystaniem generatora IPCore „Xilinx Floating-point v.3.0"

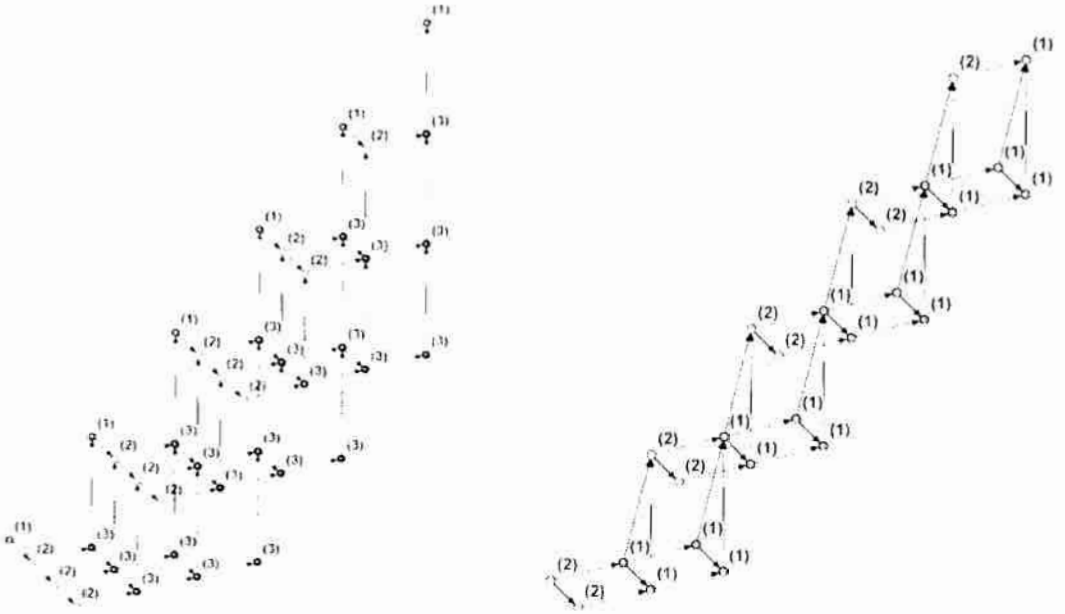
Operacje algorytmu Gaussa LU	DSP48	Bloki Slice	opóźnienie
dzielenie	1	145	4
mnożenie z odejmowaniem	0	250	4

W procesie podziału grafu suma wykorzystywanych bloków *Slice* programowalnych w każdym podgrafie nie może przekroczyć przyjętej maksymalnej wartości zależnej od wielkości docelowego układu reprogramowalnego. Przedstawione w tabelach parametry implementacji operacji arytmetycznych zostały wykorzystane w dalszych badaniach jako parametry algorytmu podziału grafu zależności informacyjnej.

Rezultaty uzyskiwane z wykorzystaniem programowania z ograniczeniami okazały się na tyle interesujące, że w kolejnym etapie badań zdecydowano się na niezależne porównanie metody podziału grafu z wykorzystaniem algorytmu genetycznego z losową populacją początkową oraz metody wykorzystującej programowanie z ograniczeniami.

Przedmiotem badań było przeprowadzenie podziału grafów zależności informacyjnych dla algorytmów rozkładu macierzy pasmowych LL^T Cholesky'ego oraz LU Gaussa za pomocą obu metod z nowymi parametrami implementacji operacji arytmetycznych określonymi dla układów z rodziny Virtex6. Na rysunku 9a przedstawiono przykładowy graf dla algorytmu rozkładu LL^T Cholesky'ego dla macierzy pasmowej (szerokość pasma 4). Graf

ten zawiera 3 rodzaje węzłów odpowiadające poszczególnym operacjom: mnożenia z odejmowaniem (3), dzielenia(2) oraz pierwiastkowania(1). Analogiczny graf dla algorytmu rozkładu Gaussa LU przedstawiono na rysunku 9b. Graf ten zawiera 2 rodzaje węzłów dla operacji dzielenia(2) oraz odejmowania z mnożeniem(1). Parametry implementacji wszystkich rodzajów operacji arytmetycznych obu algorytmów zostały przedstawione w tabelach 7 i 8.



a) b)

Rysunek 9. Przykładowe grafy zależności informacyjnych dla macierzy pasmowych dla algorytmu: a) Cholesky'ego LL^T; b) Gaussa LU

W badaniach porównano dwie metody podziału grafu zależności informacyjnej. Pierwsza metoda wykorzystuje wcześniej opisany algorytm ewolucyjny, natomiast w drugiej metodzie generowane były rozwiązania z wykorzystaniem samego programowania z ograniczeniami z wyborem najlepszego rozwiązania. Minimalny czas realizacji zadanego algorytmu algebry liniowej, obliczano po podziale grafu zależności informacyjnej na podgrafy odpowiadające poszczególnym kontekstom układu. W badaniach założono, że każda operacja arytmetyczna może być obliczana w jednym kroku, a jeden krok wykonywany w 4 taktach. Podobnie jak algorytm genetyczny, druga metoda również posiada dwa etapy pracy. W pierwszym etapie program generuje rozwiązania dopuszczalne bez ich oceny. W kolejnym wyznaczane są czasy realizacji zadanego algorytmu dla każdego rozwiązania i dokonywana jest selekcja najlepszego rozwiązania. Opisywana metoda wykorzystująca programowanie z ograniczeniami wchodzi w skład tego samego projektowanego programu JGEN(podrozdz. 4.2) co opisywany program ewolucyjny, dlatego moduł ten również został zaprogramowany w języku C# z wykorzystaniem biblioteki NSolver[94]. W ramach wykorzystania programowania z ograniczeniami badano wykorzystanie dwóch heurystyk przeszukiwania przestrzeni rozwiązań: losową oraz heurystykę o nazwie „MinSizeMinValue", która polegała na przeszukiwaniu od najmniejszych wartości dla najmniejszych indeksów w chromosomie.

Po eksperymentach stwierdzono, że wykorzystanie losowej heurystyki pozwala na otrzymywanie lepszych rozwiązań dla małych grafów (poniżej 200 węzłów), natomiast druga pozwalała uzyskać szybciej rozwiązania dla większych grafów. Wszystkie przedstawione rezultaty otrzymane zostały przy wykorzystaniu heurystyki „MinSizeMinValue". W trakcie realizacji eksperymentów pojawił sie problem z wykorzystaniem biblioteki NSolver dla liczby kontekstów większej niż 4. Zdaniem autora rozprawy problem ten spowodowany był binarną reprezentacją wszystkich węzłów grafu we wszystkich kontekstach. Taka reprezentacja została wybrana ze względu na ograniczoną liczbę zaimplementowanych operacji na zmiennych tablicowych w wykorzystywanej bibliotece, co utrudniało konstrukcje wszystkich ograniczeń podziału grafu. W efekcie takiej reprezentacji liczba konstruowanych ograniczeń była proporcjonalna do liczby węzłów w grafie.

Poniżej przedstawiono rezultaty dekompozycji grafów dla dwóch popularnych algorytmów algebry liniowej. W tabeli 9 przedstawiono czasy realizacji algorytmu Cholesky'ego dla macierzy pasmowych (szerokość pasma 5) dla układu XC6VLX550T(85920 Slice, 864 DSP) i przyjęto maksymalny rozmiar kontekstu 82K bloków Slice. Analogiczne czasy realizacji przedstawiono w tabeli 10 dla szerokości pasma 7 w układzie XC6VLX760 (118560 Slice, 864 DSP) z przyjętym rozmiarem kontekstu 115K bloków Slice. Rezultaty otrzymane dla dekompozycji grafu algorytmu Gaussa dla macierzy pasmowych o szerokościach pasma 7 i 9 przedstawiono w tabelach 11 i 12. Dla tego algorytmu wykorzystano parametry układów XC6VSX475T(74400 Slice, 2016 DSP) oraz XC6VLX550T.

Tabela 9. Czasy realizacji algorytmu LL^T Cholesky'ego dla macierzy pasmowej o szrerokości pasma 5 przy maksymalnym rozmiarze kontekstu 82k bloków Slice

Rozmiar macierzy	30	40	50	60	70	80	90	100
Liczba węzłów grafu	410	560	710	860	1010	1160	1310	1460
Liczba kontekstów układu	2	2	2	3	3	4	4	4
Rozmiar grafu(Slices)	87110	119130	151150	183170	215190	247210	279230	311250
Minimalny czas realizacji (kroki)	88	118	148	178	208	238	268	298
Algorytm ewolucyjny (kroki)	88	120	149	181	211	245	277	309
Programowanie z ograniczeniami (kroki)	88	118	148	180	210	243	273	305

Tabela 10. Czasy realizacji algorytmu LL^T Cholesky'ego dla macierzy pasmowej o szrerokości pasma 7 przy maksymalnym rozmiarze kontekstu 115k bloków Slices

Rozmiar macierzy	30	40	50	60	70	80	90	100
Liczba węzłów grafu	728	1008	1288	1568	1848	2128	2408	2688
Liczba kontekstów układu	2	2	3	4	4	5	5	6
Rozmiar grafu(Slices)	161465	223885	286305	348725	411145	473565	535985	598405
Minimalny czas realizacji (kroki)	88	118	148	178	208	238	268	298
Algorytm ewolucyjny (kroki)	94	126	161	206	231	265	296	348
Programowanie z ograniczeniami (kroki)	88	120	157	195	-	-	-	-

Tabela 11. Czasy realizacji algorytmu LU Gaussa dla macierzy pasmowej o szrerokości pasma 7 przy maksymalnym rozmiarze kontekstu 70k bloków Slice

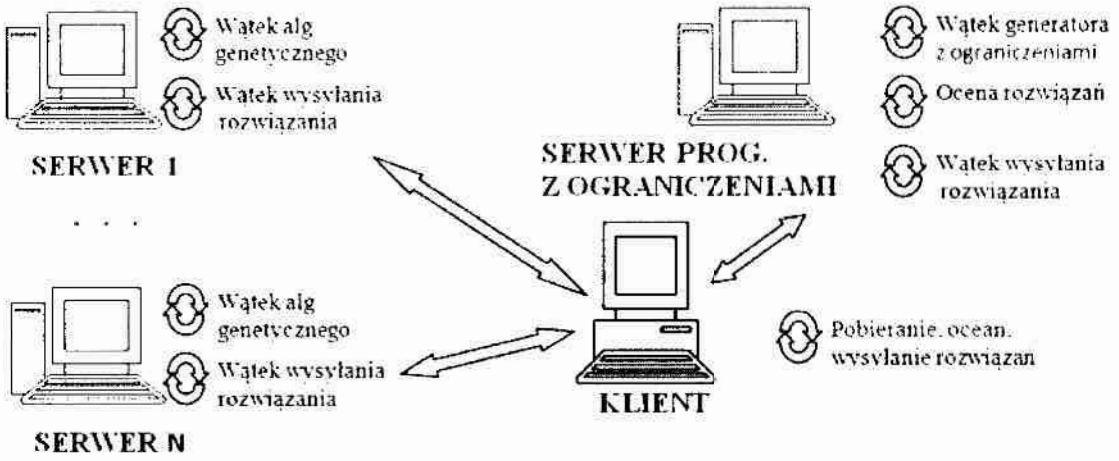
Rozmiar macierzy	30	40	50	60	70	80	90	100
Liczba węzłów grafu	320	440	560	680	800	920	1040	1160
Liczba kontekstów układu	2	2	2	3	3	4	4	4
Rozmiar grafu(Slices)	71495	98345	125195	152045	178895	205745	232595	259445
Minimalny czas realizacji (kroki)	83	113	143	173	203	233	263	293
Algorytm ewolucyjny (kroki)	83	113	144	174	204	235	266	297
Programowanie z ograniczeniami (kroki)	83	113	143	174	204	234	267	297

Tabela 12. Czasy realizacji algorytmu LU Gaussa dla macierzy pasmowej o szrerokości pasma 9 przy maksymalnym rozmiarze kontekstu 82k Slice

Rozmiar macierzy	30	40	50	60	70	80	90	100
Liczba węzłów grafu	520	720	920	1120	1320	1520	1720	1920
Liczba kontekstów układu	2	3	3	4	4	5	5	6
Rozmiar grafu(Slices)	118870	164670	210470	256270	302070	347870	393670	439470
Minimalny czas realizacji (kroki)	83	113	143	173	203	233	263	293
Algorytm ewolucyjny (kroki)	85	120	147	186	211	246	276	318
Programowanie z ograniczeniami (kroki)	83	117	147	182	-	-	-	-

Wszystkie rezultaty zostały otrzymane w czasie nie przekraczającym 10 min na komputerze Dell OPTIPLEX 755 z procesorem Intel Core2 Quad 2,2 GHz. Wyniki przedstawione w tabelach 9–12 wskazują niewielką przewagę metody wykorzystującej programowanie z ograniczeniami dla grafów z liczbą węzłów poniżej 1500. Na podstawie wyników przedstawionych w tabelach 9 i 11 można zauważyć, że dla większych rozmiarów macierzy wejściowych moduł NSolver nie mógł znaleźć rozwiązania w zadanym okresie czasu, natomiast algorytm ewolucyjny ciągle pozwalał uzyskać poprawną, spełniającą wszystkie ograniczenia dekompozycje grafów.

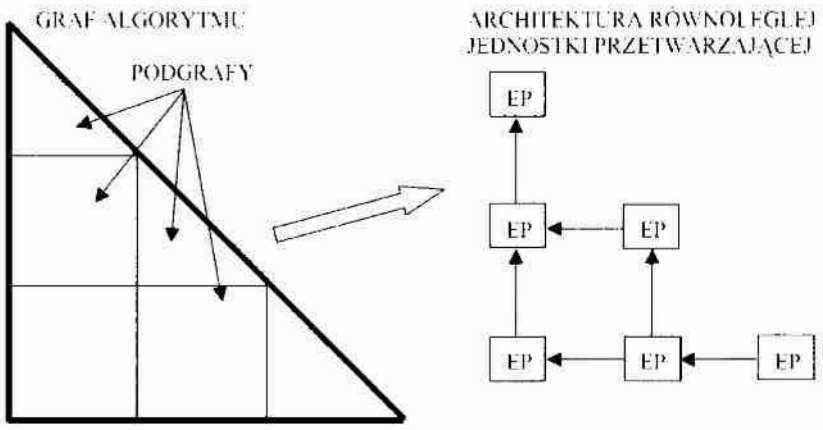
Podsumowując, obie proponowane metody dają lepsze rezultaty w porównaniu z innymi znanymi liniowymi i nieliniowymi metodami odwzorowania grafów w macierze procesorowe przedstawionymi w pracach[1,76]. Proponowane metody zostały wykorzystane w tym samym generatorze architektur równoległych(podrozdz. 4.2) dla algorytmów algebry liniowej, jednak dotychczas moduł programowania z ograniczeniami był jedynie wykorzystywany do generowania populacji startowej algorytmu ewolucyjnego. Przedstawione w tabelach 9, 10, 11 i 12 rezultaty wskazują na możliwość poprawy otrzymywanych rezultatów w równoległej wersji algorytmu ewolucyjnego, w przypadku zastąpienia jednej populacji(wyspy) generatorem osobników wykorzystujących tylko moduł programowania z ograniczeniami, gdyż dla niektórych rozmiarów macierzy wejściowych uzyskano w tym przypadku lepsze rozwiązania(mniejszą liczbę kroków). Ponadto moduł programowania z ograniczeniami znajdował zdecydowanie więcej różnych dopuszczalnych rozwiązań, niż liczba osobników w populacji algorytmu genetycznego, dlatego warto zastosować ten moduł do znajdowania rozwiązań nie tylko w celu wygenerowania populacji startowej. Przykładowy model wyspowy algorytmu genetycznego z wykorzystaniem modułu generowania rozwiązań metodą programowania z ograniczeniami przedstawiono na rys. 10.



Rysunek 10. Rozproszona aplikacja do obliczeń z wykorzystaniem modelu wyspowego algorytmu genetycznego z dodatkowym serwerem generującym osobniki z wykorzystaniem programowania z ograniczeniami.

2.3. Projektowanie macierzy procesorowych przeznaczonych do implementacji w klasycznych układach FPGA z wykorzystaniem algorytmu ewolucyjnego

W podrozdziale 2.2 przedstawiono projektowanie architektur równoległych, przetwarzających dane w sposób systoliczny i przeznaczonych do implementacji w wielokontekstowych układach FPGA. Podczas projektowania tego rodzaju architektur dokonywano odwzorowania przestrzennego grafu zależności informacyjnych zgodnie z zasadą LRGS – lokalnie równolegle i globalnie szeregowo. W niniejszym podrozdziale przedstawia się nową metodę odwzorowania przestrzennego w macierzach procesorowych zgodnie z drugim podejściem lokalnie szeregowym globalnie równoległym – LSGR[1,60]. Zgodnie z metodą LSGR każdemu podgrafowi odpowiada jeden element przetwarzający EP, w którym wszystkie wierzchołki odpowiedniego podgrafu realizowane są w sposób sekwencyjny. Przykładowa dekompozycja grafu zależności informacyjnej i uzyskana zgodnie z metodą LSGR architektura macierzy procesorowej przedstawiona została na rys.11.



Rysunek 11. Przykład stosowania metody LSGR. (źródło [1] str. 60)

W podrozdziale 1.2 przedstawiono przegląd znanych metod projektowania macierzy procesorowych z ich krótką charakterystyką. Przedmiotem badań przedstawionych w bieżącym podrozdziale jest zastosowanie algorytmu genetycznego do odwzorowania przestrzennego w procesie projektowania macierzy procesorowych. Po tym odwzorowaniu w sposób iteracyjny dokonywane było odwzorowanie czasowe, polegające na przypisaniu minimalnych możliwych taktów, w których wszystkie operacje mogą być realizowane. Zastosowanie algorytmu genetycznego pozwala na dokonywanie odwzorowania przestrzennego w macierz procesorową o zadanym kształcie, w przeciwieństwie do innych metod, w których zazwyczaj wybieramy macierz procesorową z kilku podstawowych możliwych kombinacji rozwiązań. Ponadto proponowane w tym podrozdziale odwzorowanie powoduje krótszą realizację zadanych algorytmów algebry liniowej w stosunku do innych znanych metod wykorzystujących liniowe i nieliniowe metody odwzorowania przestrzennego i czasowego, przedstawione zazwyczaj w postaci wektorów. Zastosowanie algorytmu genetycznego do projekcji przestrzennej pozwala na pełną automatyzację procesu projektowania, nawet bez specjalistycznej wiedzy osoby projektującej, gdyż nie wymaga określenia projekcji przestrzennej i również projekcji czasowej. Odwzorowanie przestrzenne dla architektur realizowanych zgodnie z metodą LSGR posiada nieco inne ograniczenia w stosunku do metody LRGS, przedstawionej w podrozdziale 2.2, przeznaczonej do implementacji w wielokontekstowych układach FPGA. Mianowicie podstawowym ograniczeniem przy realizacji architektur macierzy procesorowych zgodnie z podejściem LSGR jest ograniczenie lokalności połączeń pomiędzy elementami przetwarzającymi macierzy procesorowej(procesorami). Lokalność połączeń niezbędna jest do zachowania wysokiej częstotliwości pracy docelowego układu reprogramowalnego, zawierającego zaimplementowaną macierz procesorową, w celu zapewnienia wysokiej wydajności całego akceleratora. Ponadto dla większych macierzy procesorowych, zawierających nie tylko brzegowe elementy przetwarzające, należy uwzględnić ograniczenie w postaci umieszczenia wszystkich operacji wymagających danych „zewnętrznych" oraz operacji, w wyniku których otrzymywane są rezultaty końcowe w brzegowych elementach przetwarzających macierzy. Problematyczne może być wykorzystanie algorytmów genetycznych do odwzorowania grafów zależności informacyjnych o dużych rozmiarach. Z tego względu zaproponowano dekompozycję grafów zależności informacyjnej przed dokonaniem odwzorowania przestrzennego. W dalszej części podrozdziału zaprezentowano rezultaty wykorzystania algorytmu genetycznego do odwzorowania przestrzennego dla przykładowych algorytmów algebry liniowej. Przedstawiono szacunkowe czasy realizacji tych algorytmów mierzone w taktach lub krokach, uzyskanych z wykorzystaniem proponowanej metody odwzorowania dla przykładowej macierzy procesorowej o zadanym kształcie.

2.3.1. Charakterystyka opracowanego algorytmu genetycznego.

Podobnie jak w algorytmie zaprezentowanym w podrozdziale 2.2.1. również w tym przypadku wykorzystano kodowanie grup podziałów za pomocą liczb (rys. 8). Elementom w projektowanej macierzy procesorowej nadano kolejne numery. Pozycja

w chromosomie odpowiada numerowi węzła grafu, natomiast wartość określa numer elementu przetwarzającego w macierzy procesorowej. Rozmiar i kształt projektowanej macierzy procesorowej był określany przed uruchomieniem algorytmu genetycznego, przez użytkownika programu. Populacja prezentowanego algorytmu genetycznego zawierała 100 osobników, a populacja startowa generowana była w sposób losowy. W podrozdziale 2.2.1 zaprezentowano wykorzystanie programowania z ograniczeniami do generowania populacji startowej, co umożliwiało znalezienie dopuszczalnego rozwiązania w krótkim czasie, nawet dla grafów zależności informacyjnej o dużych rozmiarach. W bieżącym algorytmie dla większych grafów zastosowano ich dekompozycję – graf zależności informacyjnej był automatycznie dzielony na zadaną liczbę pogdrafów. Natomiast operatory rekombinacji operowały jednie na pozycjach w chromosomie odpowiadającym węzłom bieżącego podgrafu. Liczba podgrafów definiowana była przez użytkownika programu. Przy zastosowaniu wstępnej dekompozycji grafu losowe generowanie populacji startowej okazało się zazwyczaj sposobem wystarczającym na uzyskanie dopuszczalnego rozwiązania w stosunkowo krótkim czasie działania algorytmu genetycznego(poniżej 20 min). W pierwszym etapie pracy programu realizującego algorytm genetyczny wyznaczano zakres węzłów grafu na którym działały operatory rekombinacji oraz czas pracy programu dla każdego z podgrafów. Graf, na podstawie którego wyznaczane były wartości funkcji przystosowania, zwiększał się sukcesywnie w trakcie pracy programu, poprzez dodawanie kolejnych podgrafów, natomiast operatory rekombinacji działały na ostatnich pozycjach w chromosomie, które odpowiadały węzłom ostatnio dołączonego podgrafu. W algorytmie wykorzystano standardowy operator krzyżowania ze stałym prawdowo-dobieństwem, którego wartość została ustalona eksperymentalnie na poziomie 0.2. Operator mutacji natomiast działał ze zmiennym prawdopodobieństwem, którego wartość była wyznaczana w taki sposób, że w początkowej fazie pracy algorytmu zazwyczaj mutacji podlegała dokładnie jedna pozycja w chromosomie, niezależnie od liczby węzłów w bieżącym podgrafie. Wartość prawdopodobieństwa dla operatora mutacji wzrastała wraz z czasem działania algorytmu dla kolejnych okresów czasu bez poprawy najlepszego rozwiązania. Zazwyczaj dla 20 min działania programu mutacji podlegało nie więcej jak 10 pozycji w chromosomie. Prezentowany algorytm genetyczny pracował w dwóch etapach dla każdego z podgrafów. W pierwszym etapie wartość funkcji oceny zależała od liczby niespełnionych warunków projekcji przestrzennej, takich jak niespełnione warunki lokalności połączeń w projektowanej macierzy procesorowej, dla których łuk(krawędź) grafu łączył węzły przypisane do niesąsiadujących elementów przetwarzających macierzy. Funkcja oceny wykorzystana w pierwszym etapie pracy algorytmu dla każdego z podgrafów została opisana wzorem (6):

$$F4 = 1 + EN * (EN - SE) \qquad (6)$$

, gdzie:

EN – liczba łuków w bieżącym podgrafie,
SE – liczba łuków niespełniających ograniczeń.

Po znalezieniu dopuszczalnego rozwiązania dla projekcji przestrzennej wartość funkcji oceny była wyznaczana z wykorzystaniem dwóch metod. Dla osobników spełniających ograniczenia projekcji przestrzennej dokonywano odwzorowania czasowego, natomiast wartość funkcji oceny F5 uzależniona była od liczby taktów niezbędnych do realizacji wszystkich operacji dla bieżącego podgrafu zgodnie ze wzorem (7):

$$F5 = 1 + EN^2 + NN - T \qquad (7)$$

, gdzie:

EN – liczba łuków bieżącego podgrafu,
NN – liczba wezłów bieżącego pografu ,
T – liczba taktów niezbędna do realizacji wszystkich operacji podgrafu w macierzy procesorowej.

Dla osobników niespełniających ograniczenia lokalności połączeń wartość funkcji oceny F6 przyjmowała zdecydowanie mniejsze wartości, analogicznie do funkcji kary[91] i była wyznaczana na podstawie zależności określonej wzorem (8):

$$F6 = 1 + EN - SE \qquad (8)$$

, gdzie:

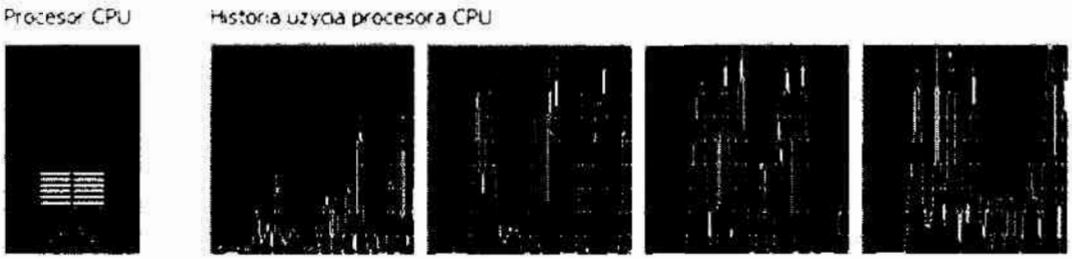
EN – liczba łuków w bieżącym podgrafie,
SE – liczba łuków niespełniających ograniczeń.

W prezentowanym algorytmie genetycznym również wykorzystano standardowy model selekcji elitarystycznej, ponieważ w tym modelu najlepsze rozwiązanie zawsze jest przekazywane do kolejnej populacji, co miało znaczenie dla uzyskiwania dopuszczalnych rozwiązań w stosunkowo krótkim czasie działania algorytmu, gdyż operator mutacji łatwo mógł zmieniać rozwiązania dopuszczalne w niedopuszczalne. Przyjęto stosunkowo krótki, jak na algorytm ewolucyjny do rozwiązywania dużych problemów, czas działania algorytmu ze względu na jego wykorzystanie w opracowywanym generatorze IPCore – JGEN(podrozdz. 4.2). Maksymalny czas działania algorytmu ograniczony był do 20 lub 30min, dla wszystkich testowanych algorytmów algebry liniowej.

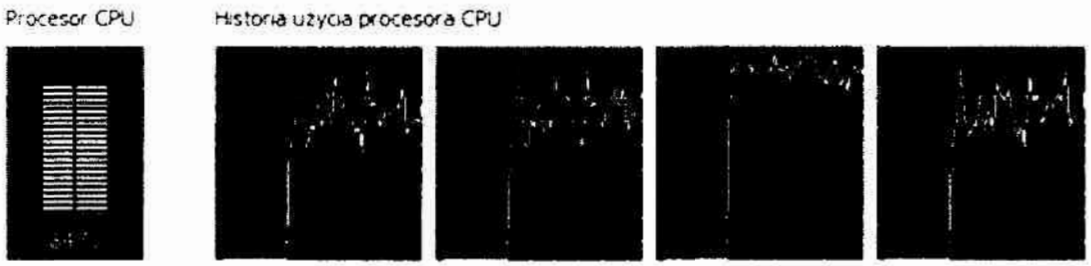
Również w przypadku algorytmu genetycznego, opisywanego w bieżącym podrozdziale, podjęto próbę jego równoległej realizacji. W algorytmie opisanym w podrozdziale 2.2.1 wykorzystano rozproszoną aplikacje jeden klient – wiele serwerów obliczeniowych oraz model wyspowy algorytmu genetycznego, natomiast w bieżącym algorytmie postanowiono zrównoleglić obliczenia wartości funkcji przystosowania z wykorzystaniem powszechnie dostępnych procesorów wielordzeniowych z wykorzystaniem popularnych komputerów klasy PC. Program realizujący opisywany algorytm genetyczny, podobnie jak większość metod środowiska JGEN(podrozdz. 4.2), zastał utworzony z wykorzystaniem platformy

Microsoft .NET[15], natomiast do jego równoległej realizacji wykorzystano rozszerzenie Microsoft ParallelFX[16]. Równoległa realizacja funkcji oceny, dzięki wykorzystaniu wszystkich dostępnych rdzeni w procesorze, pozwoliła na uzyskiwanie większych liczb generacji algorytmu genetycznego. Powodowało to uzyskanie lepszych rozwiązań lub zwiększenie procentowej wartość określającej liczbę uruchomień programu dla których znaleziono dopuszczalne rozwiązania. Program, w którym uzyskano wszystkie zaprezentowane w dalszej części podrozdziału rezultaty, wykonywano z wykorzystaniem komputera Dell Optiplex 755 z procesorem Intel Core 2 Quad 2,2 GHz.

Na rys. 12 przedstawiono przykładowe obciążenie rdzeni procesora dla uruchomionego programu z szeregową implementacja funkcji oceny, natomiast na rysunku 13 przedstawiono analogiczne obciążenie dla programu z równoległą implementacją funkcji oceny.



Rysunek 12. Wykres przedstawiający średnie procentowe wykorzystanie procesora Intel Core 2 Quad 2,2 GHz oraz historię wykorzystania poszczególnych rdzeni podczas algorytmu genetycznego z szeregową realizacją obliczeń funkcji celu.



Rysunek 13. Wykres przedstawiający średnie procentowe wykorzystanie procesora Intel Core 2 Quad 2,2 GHz oraz historię wykorzystania poszczególnych rdzeni podczas algorytmu genetycznego z równoległą realizacją obliczeń funkcji celu.

Średnie obciążenie procesora dla szeregowej implementacji funkcji oceny oscylowało wokół wartości 25%, natomiast przy jej równoległej implementacji przekraczało wartość 80%. W celu zobrazowania przyspieszenia programu realizującego algorytm genetyczny podjęto próby określenia liczy jego generacji(populacji). Opisywany algorytm genetyczny pracuje jednak dwuetapowo, więc inna funkcja oceny(wzór 6) o innej złożoności obliczeniowej jest

[15] http://www.microsoft.com/net/

[16] http://msdn.microsoft.com/en-us/concurrency/default.aspx

wykorzystywana przed znalezieniem dopuszczalnego rozwiązania odwzorowania przestrzennego oraz po znalezieniu takiego rozwiązania(wzór 7 lub 8). Z tego powodu liczba generacji była również mocno uzależniona od czasu, w którym algorytm znalazł dopuszczalne rozwiązanie. W kolejnym podrozdziale zaprezentowano procentowe wartości określające liczbę uruchomień programów(dla krótkiego czasu działania) pozwalających otrzymać dopuszczalne rozwiązania, co pozwoliło wykazać korzyści z równoległej implementacji funkcji oceny.

2.3.2. Rezultaty dekompozycji grafów zależności informacyjnych uzyskane z wykorzystaniem proponowanego algorytmu genetycznego.

Podobnie jak w odwzorowaniu przestrzennym dla architektur przeznaczonych do implementacji w wielokontekstowych układach reprogramowalnych, również w przypadku projektowania macierzy procesorowych zgodnie z metodą LSGR decydującym kryterium optymalizacji był czas realizacji algorytmu algebry liniowej, mierzony w taktach lub krokach, co jest również powiązane z kryterium średniego obciążenia procesorów w macierzy. Zaprojektowany algorytm można jednak łatwo zmodyfikować dodając nowe kryteria optymalizacji projektowanej architektury, jak liczba portów wejściowych i wyjściowych. Pierwsze próby odwzorowania macierzy grafów zależności informacyjnych w macierze procesorowe z wykorzystaniem algorytmów genetycznych również okazały się problematyczne dla grafów zawierających ponad 100 węzłów. Z tego względu postanowiono dokonać dekompozycji grafu na podgrafy i dokonywać odwzorowania dla kolejnych podgrafów algorytmu. Otrzymane w wyniku eksperymentów minimalne liczby taktów zależały więc zarówno od rozmiarów macierzy wejściowych oraz od przyjętej liczby podgrafów. W tabeli 13 zaprezentowano parametry przykładowej macierzy procesorowej, zawierającej 4 elementy przetwarzające(macierz o rozmiarach 2x2), zaprojektowanej przy wykorzystaniu opisanego algorytmy genetycznego. Program uruchamiano 5-krotnie przy zachowaniu maksymalnego czasu działania do 20 min. W tabeli zestawiono otrzymane najkrótsze oraz średnie czasy realizacji algorytmu rozkładu Gaussa LU dla macierzy pasmowej o szerokości pasma 9. Dodatkowo umieszczono średnie obciążenia procesorów w zaprojektowanych macierzach procesorowych. Eksperymenty przeprowadzono dla różnych rozmiarów macierzy wejściowych zarówno bez podziału oraz z podziałem grafów zależności informacyjnych na 50, 100 oraz 150 podgrafów.

Tabela 13. Parametry zaprojektowanych macierzy procesorowych(2x2) przeznaczonych do realizacji algorytmu rozkładu macierzy metodą Gaussa LU dla macierzy pasmowej o szerokości pasma 9 otrzymane po 20 min działania programu

Rozmiar macierzy danych wejściowych (NxN)		30	40	50	60	70
Liczba węzłów grafu		520	720	920	1120	1320
Liczba łuków grafu		1213	1683	2153	2623	3093
bez dekompozycji	rozwiązania dopuszczalne	0%	0%	0%	0%	0%
50 podgrafów	najkrótszy czas realizacji(takty)	200	275	355	-	-
	średni czas realizacji (takty)	207	282	355	-	-
	rozwiązania dopuszczalne	80%	80%	20%	0%	0%
	średnie obciążenie procesorów	63%	64%	65%	-	-
100 podgrafów	najkrótszy czas realizacji(takty)	209	307	368	-	-
	średni czas realizacji (takty)	223	320	385	-	-
	rozwiązania dopuszczalne	100%	100%	100%	0%	0%
	średnie obciążenie procesorów	58%	56%	60%	-	-
150 podgrafów	najkrótszy czas realizacji(takty)	245	294	375	478	576
	średni czas realizacji (takty)	258	305	384	498	576
	rozwiązania dopuszczalne	100%	100%	60%	80%	20%
	średnie obciążenie procesorów	50%	59%	60%	56%	57%

W tabeli 14 zestawiono analogiczne parametry zaprojektowanych macierzy procesorowych, również dla 5 krotnego uruchomienia programu, lecz przy maksymalnym czasie działania programu do 30min.

Tabela 14. Parametry zaprojektowanych macierzy procesorowych (2x2) przeznaczonych do realizacji algorytmu rozkładu macierzy metodą Gaussa LU dla macierzy pasmowej o szerokości pasma 9 otrzymane po 30 min działania programu

Rozmiar macierzy danych wejściowych (NxN)		30	40	50	60	70
Liczba węzłów grafu		520	720	920	1120	1320
Liczba łuków grafu		1213	1683	2153	2623	3093
bez dekompozycji	rozwiązania dopuszczalne	0%	0%	0%	0%	0%
50 podgrafów	najkrótszy czas realizacji(takty)	200	264	-	-	-
	średni czas realizacji (takty)	210	274	-	-	-
	rozwiązania dopuszczalne	100%	100%	0%	0%	0%
	średnie obciążenie procesorów	62%	66%	-	-	-
100 podgrafów	najkrótszy czas realizacji(takty)	215	297	370	483	-
	średni czas realizacji (takty)	229	313	378	483	-
	rozwiązania dopuszczalne	100%	80%	100%	20%	0%
	średnie obciążenie procesorów	57%	57%	61%	58%	-
150 podgrafów	najkrótszy czas realizacji(takty)	256	269	380	492	550
	średni czas realizacji (takty)	264	297	393	494	557
	rozwiązania dopuszczalne	100%	100%	100%	40%	60%
	średnie obciążenie procesorów	49%	61%	59%	57%	59%

W tabelach 13 i 14 przedstawiono czasy realizacji przykładowych algorytmów rozkładu Gaussa LU zawierających ponad 1000 operacji w zaprojektowanej przykładowej macierzy procesorowej z wykorzystaniem algorytmu genetycznego w czasie do 30 min. Zastosowanie algorytmu genetycznego do odwzorowania przestrzennego pozwala praktycznie dowolnie określić kształt projektowanej macierzy procesorowej, choć naturalnie złożoność problemu

projektowania rośnie wraz ze wzrostem liczby elementów przetwarzających w macierzy. Trudno porównać uzyskane czasy realizacji przykładowego algorytmu w stosunku do innych rozwiązań, gdyż w znanych metodach odwzorowania nie da się zaprojektować macierzy procesorowej o dowolnie wybranym kształcie – w tym przypadku macierz 2 x 2 zawierającą 4 elementy przetwarzające. Warto podkreślić, iż w zaprojektowanych macierzach procesorowych uzyskano wysokie średnie obciążenie procesorów, które zależy od czasu realizacji oraz liczby procesorów w macierzy. W zaprojektowanych macierzach uzyskano średnie obciążenie w granicach od około 50 do 60%. Dla porównania z wykorzystaniem, stosowanych w innych metodach, liniowego odwzorowania przestrzennego i czasowego uzyskuje się często zdecydowanie niższe wartości obciążenia. Przykładowo dla algorytmów redukcji wstecznej, czy iteracji prostej stosowanych do rozwiązywania układu równań dla macierzy pasmowych dla różnych kombinacji liniowych funkcji odwzorowania, można uzyskać średnie wartości obciążenia procesorów w macierzy w granicach od 20 do 50%[17].

W celu wykazania korzyści z równoległego obliczania funkcji celu algorytmu genetycznego w tabeli 15 zestawiono najkrótsze czasy realizacji algorytmu Gaussa LU w zaprojektowanej macierzy, otrzymane przy maksymalnym czasie działania programu 30 min, zarówno dla szeregowej i równoległej implementacji programowej funkcji celu, z podziałem grafu na 50 podgrafów. Celowo nie umieszczono w tabeli liczby generacji algorytmu genetycznego, gdyż wartość ta zależała nie tylko do rodzaju implementacji funkcji celu ale również od tego jak szybko program znalazł dopuszczalne rozwiązanie. Po znalezieniu dopuszczalnego rozwiązania liczba obliczanych generacji zdecydowanie spadała, gdyż do obliczenia wartości funkcji celu, poza dokonaniem odwzorowania przestrzennego grafu, niezbędne było dokonanie również odwzorowania czasowego, realizowanego metodą iteracyjną.

Tabela 15. Parametry macierzy procesorowej(2x2) zaprojektowanej do realizacji algorytmu Gaussa LU dla macierzy pasmowej o szerokości pasma 9 otrzymane z wykorzystaniem podziału grafu na 50 podgrafów oraz szeregowej i równoległej implementacji funkcji celu przy maksymalnym czasie działania programu do 30 min

Rozmiar macierzy (NxN)		30	40	50	60	70
Realizacja szeregowa	najkrótszy czas realizacji(takty)	200	254	-	-	-
	rozwiązania dopuszczalne	100%	100%	0%	0%	0%
Realizacja równoległa	najkrótszy czas realizacji(takty)	207	259	335	404	-
	rozwiązania dopuszczalne	100%	100%	80%	20%	0%

Zaprezentowane w tabeli 15 rezultaty potwierdzają zalety równoległej implementacji programowej funkcji celu, przede wszystkim w postaci większej liczby znalezionych dopuszczalnych rozwiązań przy zachowaniu tego samego czasu działania algorytmu ewolucyjnego. Dzięki zastosowaniu dekompozycji grafu zależności informacyjnej na podgrafy oraz dzięki równoległej realizacji obliczeń funkcji oceny algorytmu genetycznego można projektować macierze procesorowe dla większych rozmiarów macierzy danych wejściowych

[17] http://kik.weii.tu.koszalin.pl/psi/Projekty.zip

lub dla tych samych rozmiarów w krótszym czasie działania programu. Podczas przeprowadzonych eksperymentów stwierdzono, że najlepiej dobierać taką liczbę podgrafów i czas działania programu, aby operatory rekombinacji pracowały na zakresie do kiludziesięciu węzłów w czasie kilkudziesięciu sekund. Ze względu na implementację programu w docelowym IPCore dla algorytmów algebry liniowej oraz ze względu na liczbę przeprowadzanych eksperymentów przyjęto stosunkowo krótkie czasy działania programu, jak na czas działania algorytmu genetycznego dla tak złożonych problemów. Rozmiary macierzy wejściowych dla których projektowano macierze procesorowe mogą ulec zwiększeniu poprzez dobór odpowiedniego czasu działania programu oraz odpowiednią liczbę podgrafów. Możliwe jest również wprowadzanie drobnych modyfikacji programu tak, aby przy każdym uruchomieniu otrzymywać rozwiązania dopuszczalne. Wówczas operatory rekombinacji pracowałyby na bieżącym podgrafie, przynajmniej do momentu uzyskania dopuszczalnego rozwiązania, a nie jak dotychczas w stałym, wyznaczonym dla każdego podgrafu, okresie czasu.

2.4. Podsumowanie proponowanych metod projektowych.

W podrozdziale 2.2 przedstawiono nową koncepcję realizacji równoległych obliczeń algorytmów algebry liniowej z wykorzystaniem wielokontekstowych układów FPGA. Nowa koncepcja pozwala na realizację wybranych algorytmów algebry liniowej w czasie (mierzonym w taktach lub krokach) zbliżonym do wartości optymalnej, określonej czasem realizacji ścieżki krytycznej grafów tych algorytmów. Ponadto proponowana w rozprawie koncepcja architektury równoległej pozwala wyeliminować bloki sterowania elementami przetwarzającymi równoległej architektury co pozwala zwiększyć jej wydajność lub zmniejszyć złożoność sprzętową. Organizacja sterowania rodzajem wykonywanych operacji w elementach przetwarzających macierzy procesorowych stanowi osobny, rozległy obszar badań naukowych[1], natomiast w przypadku proponowanej organizacji obliczeń problem ten praktycznie nie istnieje, ponieważ każdy element przetwarzający realizuje tylko jeden rodzaj operacji. Opisano również opracowane metody dekompozycji grafów zależności informacyjnych, mające na celu odwzorowanie przestrzenne tych grafów, służące do projektowania równoległych architektur wykorzystujących proponowaną koncepcję obliczeń. Metody dekompozycji zostały zaimplementowane programowo, natomiast w niniejszej pracy przedstawiono rezultaty dekompozycji grafów zależności informacyjnych dla przykładowych algorytmów algebry liniowej. Problematyczne okazało się wykorzystanie proponowanych metod dla większych rozmiarów macierzy wejściowych, dlatego zaproponowano ich równoległą implementację w postaci aplikacji rozproszonej typu wiele serwerów obliczeniowych – jeden klient, z wykorzystaniem powszechnie dostępnych, standardowych komputerów klasy PC. Otrzymane rezultaty odwzorowania grafów zależności informacyjnych pozwalają wskazać przewagę proponowanych metod w stosunku do znanych metod odwzorowania liniowego i nieliniowego pod względem czasu realizacji wybranych algorytmów algebry liniowej.

W podrozdziale 2.3 zaprezentowano wykorzystanie algorytmu genetycznego do projektowania klasycznych macierzy procesorowych[1, 44, 50]. Wykorzystanie algorytmu genetycznego, w przeciwieństwie do innych znanych metod, pozwala na zdefiniowanie przez użytkownika kształtu projektowanej macierzy procesorowej. Metoda została zaimplementowana w postaci środowiska programowego, a przedstawione rezultaty pozwalają stwierdzić, że zaprojektowane macierze procesorowe charakteryzują się większym średnim obciążeniem elementów przetwarzających macierzy w stosunku do macierzy zaprojektowanych z wykorzystaniem znanych metod projekcji liniowych.

Ponadto w obu przypadkach projektowania architektur równoległych, zgodnie z podejściem LGSR oraz LSGR, wykorzystujących algorytm genetyczny, opisane metody projektowe nadają się do całkowitej automatyzacji procesu projektowania i mogą być wykorzystywane przez użytkowników nie związanych z dziedziną projektowania, gdyż nie wymagają od użytkownika definiowania funkcji odwzorowania przestrzennego i czasowego. Zastosowanie algorytmu genetycznego posiada również pewne wady. W przeciwieństwie do liniowych i nieliniowych funkcji odwzorowania proponowane metody charakteryzują się znacznie dłuższym czasem projektowania macierzy procesorowej oraz problematyczne jest odwzorowanie grafów zależności informacyjnych dla dużych rozmiarów macierzy danych wejściowych algorytmów algebry liniowej. Mniejszy nacisk w proponowanych metodach położono na czas projektowania architektury równoległej, natomiast głównie skupiono się na parametrach projektowanych architektur, takich jak: czas realizacji algorytmu w taktach, czy średnie obciążenia procesorów. W celu zwiększania rozmiarów macierzy wejściowych zaproponowano równoległą realizację obliczeń algorytmu genetycznego z wykorzystaniem powszechnie dostępnych komputerów klasy PC. Ponadto w metodzie opisanej w podrozdziale 2.2 zaproponowano również generowanie rozwiązań odwzorowania przestrzennego z wykorzystaniem programowania z ograniczeniami. W metodzie opisanej w podrozdziale 2.3 wykorzystano natomiast wstępną dekompozycję grafów zależności informacyjnych przed dokonaniem odwzorowania przestrzennego. Dodatkowo można zwiększyć zakres rozmiarów macierzy danych wejściowych poprzez modyfikację algorytmu genetycznego, nie ograniczając jego czasu działania dla poszczególnych podgrafów, wydłużając jednak w ten sposób czas projektowania. Obiecujące, w dalszej perspektywie czasu, wydaje się wykorzystanie algorytmu genetycznego do odwzorowania przestrzennego dla reprezentatywnego podgrafu zależności informacyjnej, następnie powielenie tego odwzorowania na pozostałą część grafu. W ten sposób można połączyć zalety wykorzystania algorytmu genetycznego oraz liniowych funkcji odwzorowania przestrzennego. Takie rozwiązanie pozwoli zachować dowolny kształt projektowanej macierzy, uzyskiwać krótsze czasy realizacji zadanych algorytmów lub większe wartości średniego obciążenia procesorów w stosunku do liniowych metod odwzorowania oraz spowoduje skrócenie czasu projektowania dla wielkich rozmiarów macierzy danych wejściowych.

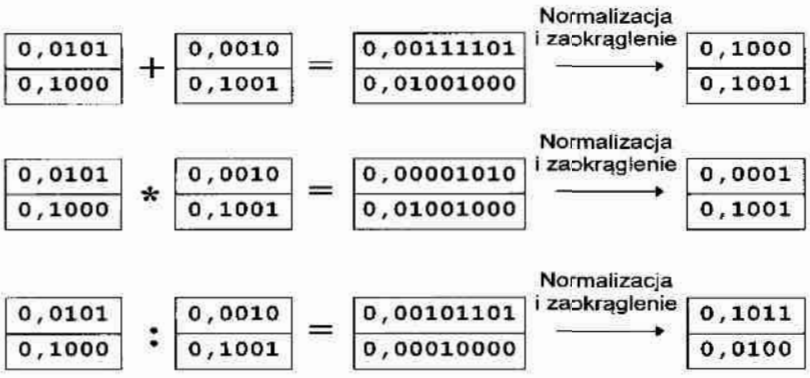
3. Projektowanie i optymalizacja potokowych architektur akceleratorów algorytmów algebry liniowej z uwzględnieniem budowy wewnętrznej nowoczesnych układów FPGA

Zgodnie z analizą przedstawioną w podrozdziale 1.3 do najczęściej spotykanych operacji arytmetycznych w algorytmach algebry liniowej można zaliczyć: mnożenie, mnożenie z akumulacją wyniku, sumowanie, dzielenie oraz operację porównania. W podrozdziale 1.2 przedstawiono z kolei krótką charakterystykę budowy i możliwości nowoczesnych układów FPGA, zawierających ogromną przestrzeń programowalną oraz wbudowane bloki funkcjonalne, jak bloki DSP lub bloki pamięci BRAM. Trudnym zadaniem staje się efektywne zagospodarowanie tak dużych zasobów, w rozsądnym czasie projektowym. Ponadto współczesne architektury układów FPGA są przystosowane przede wszystkim do efektywnej implementacji operacji mnożenia i sumowania danych przedstawionych w formatach stało i zmiennoprzecinkowych. Współczesne narzędzia do syntezy układów cyfrowych posiadają generatory IPCore podstawowych operacji arytmetycznych, wykorzystujących zarówno reprezentacje stało jak i zmiennoprzecinkową. Problematyczna jest efektywna implementacja operacji dzielenia, podczas której nie można wykorzystać wbudowanych bloków DSP. Istniejące generatory IPCore nie umożliwiają generowania bloków funkcjonalnych operacji dzielenia pracujących z pożądaną dokładnością obliczeń oraz często wygenerowane bloki charakteryzują się dużą złożonością sprzętową, dużą liczbą stopni w potoku lub niską maksymalną częstotliwością pracy[19,31,32]. Z tego powodu w niniejszej pracy proponuje się wykorzystanie arytmetyki ułamkowej do efektywnej implementacji algorytmów algebry liniowej. Algorytmy te często zawierają operację dzielenia umieszczoną w ścieżce krytycznej grafu zależności informacyjnej, od której to ścieżki zależy czas realizacji całego algorytmu. Ponadto w większości algorytmów algebry liniowej operacje dzielenia są często wykonywane raz w każdym kroku algorytmu, np. raz dla całego wiersza macierzy wejściowej i bez wyniku tej operacji nie można często realizować kolejnych obliczeń. Z jednej strony oznacza to, że operacja dzielenia powinna być realizowana jak najszybciej, a z drugiej – wprowadzenie potokowego trybu obliczeń nie może jej przyspieszyć, ponieważ operacja jest wykonywana zazwyczaj na pojedynczej parze danych.

3.1. Charakterystyka arytmetyki ułamkowej.

Pierwsze koncepcje wykorzystania arytmetyki ułamkowej RFA w systemach komputerowych pojawiły się już na przełomie lat 70-tych i 80-tych[72,73]. Jednym z przedmiotów badań przedstawionych w niniejszej rozprawie było zastosowanie arytmetyki ułamkowej do realizacji jednostek przetwarzających algorytmy algebry liniowej na platformie FPGA. Osobliwości wykorzystania arytmetyki ułamkowej na tej platformie zostały między innymi opisane w pracach[74,95,96]. W blokach operacyjnych działających z wykorzystaniem arytmetyki ułamkowej każda liczba x powinna być przedstawiona w postaci ułamka wymiernego a/b, gdzie a i b są to n-bitowe liczby całkowite. Dokładność takiej $2n$-bitowej liczby x jest zazwyczaj większa od dokładności zapisu tejże liczby w $2n$-bitowym formacie stałoprzecinkowym. Na przykład, liczba π z dokładnością siedmiu cyfr dziesiętnych może być

przedstawiona ułamkiem $x = 355/113$, zawierającym tylko 6 cyfr dziesiętnych. Więcej analogicznych przykładów przedstawiających różne stałe matematyczne przedstawiono w pracy[72]. Format ułamkowy pozwala na dokładniejsze przedstawienie liczb, w porównaniu do formatu stało-przecinkowego, również dlatego, że liczby rzeczywiste zwykle przedstawione są w formacie stałoprzecinkowym z pewnymi błędami zaokrągleń, których wielkość zależy od liczby bitów w reprezentacji liczby. Dzięki temu, np. (1/9)*9 zawsze będzie równe 1 w arytmetyce ułamkowej, ale nie zawsze będzie równe 1 w arytmetyce stałoprzecinkowej lub nawet zmiennoprzecinkowej. Ponadto, wykonanie działań arytmetycznych na niedokładnie przedstawionych argumentach zwykle powoduje powstanie jeszcze bardziej niedokładnego wyniku. Przykładowo, po wykonaniu operacji mnożenia pary liczb *n*-bitowych, dokładny 2*n*-bitowy wynik mnożenia zostaje (w formacie stałoprzecinkowym) zaokrąglony do *n*-bitowego. W arytmetyce ułamkowej również po wykonaniu operacji mnożenia dwóch ułamków *n*-bitowych *a/b* i *c/d* powstaje ułamek 2*n*-bitowy $a \cdot c / (b \cdot d)$, tj. liczba bitów w otrzymywanych wynikach ciągle wzrasta (ta sama sytuacja powstaje przy wykonaniu operacji dzielenia, a nawet dodawania, która wymaga wykonania 3 operacji mnożenia(wzory 9 i 12)). Badania wykazały jednak, że w arytmetyce ułamkowej błędy zaokrągleń można dość skutecznie zredukować poprzez wykonanie operacji normalizacji wyniku mnożenia. Normalizacja oznacza przesunięcie w lewo licznika i mianownika o jednakową liczbę bitów tak, aby co najmniej jeden z nich miał najbardziej znaczący bit tuż za bitem znaku.) po operacji normalizacji dokonywane jest zaokrąglenie do ułamka *n*-bitowego (rys.14).



Rysunek 14. Operacja normalizacji i zaokrąglenia w arytmetyce ułamkowej(źródło[95]).

Jedną z największych zalet stosowania arytmetyku ułamkowej do realizacji bloków operacyjnych na platformie FPGA, jest fakt, że operacja dzielenia sprowadza się do operacji wymnożenia pierwszej liczby ułamkowej przez odwrotność drugiej. Poniżej przedstawione zależności (9), (10), (11) i (12) przedstawiają realizację podstawowych operacji arytmetycznych w arytmetyce ułamkowej:

- dodawanie (odejm.): $a/b \pm c/d = (a \cdot d \pm b \cdot c)/(b \cdot d)$ (9)

  wymaga 3 operacji mnożenia i 1 dodawania (odejmowania),

- porównanie: a/b i c/d: $a \cdot d - b \cdot c$ (10)

  wymaga realizacji 2 operacji mnożenia i odejmowania ,

- mnożenie: $(a/b) * (c/d) = a \cdot c / (b \cdot d)$ (11)

  wymaga realizacji 2 operacji mnożenia ,

- dzielenie: $(a/b) / (c/d) = a \cdot d / (b \cdot c)$ (12)

  wymagające 2 operacji mnożenia .

Operacja mnożenia dwóch ułamków a/b i c/d w RFA sprowadza się do wykonania dwóch operacji mnożenia liczb n - bitowych, zamiast jednego mnożenia dwóch liczb $2n$-bitowych w formacie stałoprzecinkowym. Dlatego złożoność sprzętowa kombinacyjnego bloku mnożącego RFA i jego opóźnienie są około 2 razy mniejsze w porównaniu do kombinacyjnego bloku mnożącego dwie $2n$ - bitowe liczby w formacie stałoprzecinkowym[95]. Porównanie złożoności sprzętowej bloku dzielenia RFA z blokiem dzielenia dwóch $2n$ - bitowych liczb w formacie stałoprzecinkowym wypada jeszcze lepiej, ponieważ dzielenie w RFA sprowadza się do wykonania dwóch operacji mnożenia liczb n - bitowych.

W celu zbadania dokładności obliczeń realizowanych w arytmetyce ułamkowej przeprowadzono szereg badań opisanych między innymi w pracach[74, 96]. Badano, m.in. dokładność obliczeń dla algorytmu rozkładu macierzy LL^T metodą Choleskiego, algorytmu redukcji wstecznej, generatorów funkcji sinus i cosinus dla różnych reprezentacji danych. Porównanie dokładności obliczeń przeprowadzonych na liczbach stałoprzecinkowych i ułamkach wykonane zostały na przykładzie wielokrotnego obliczenia wyrażenia matematycznego (13)[74]. Wyrażenie to jest często wykorzystywane w zadaniach cyfrowego przetwarzania sygnałów, np. w generatorach funkcji sin$x$ oraz algorytmach algebry liniowej, np. w metodzie QR obliczenia wartości własnych macierzy:

$$x_{i+1} = x_i \cdot \cos\beta + y_i \cdot \sin\beta \quad \text{oraz} \quad y_{i+1} = y_i \cdot \cos\beta - x_i \cdot \sin\beta , \quad \text{gdzie } i=1,2,...,N. \qquad (13)$$

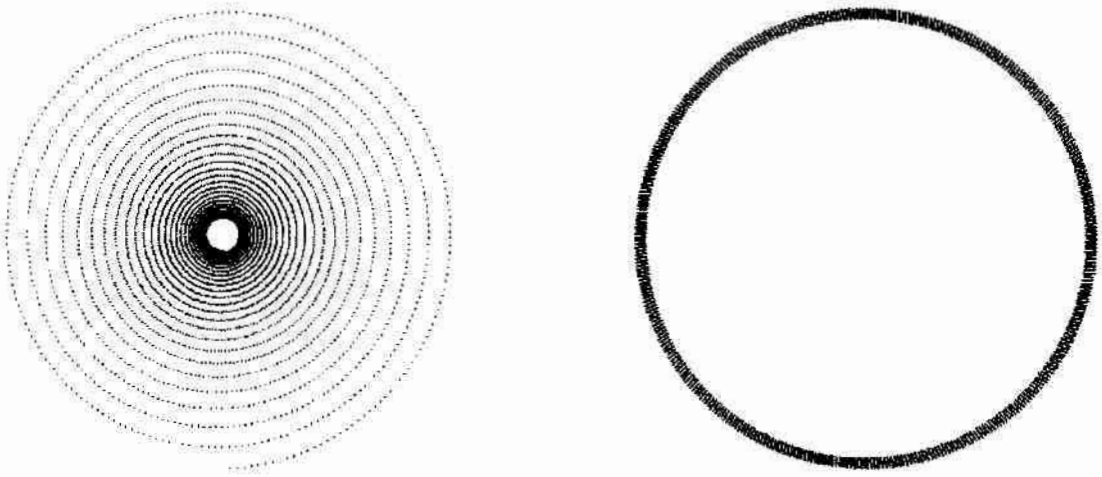
Głównymi przyczynami powstania błędów przy obliczeniu wyrażenia (13) są: niedokładne przedstawienie argumentów $\sin\beta$ i $\cos\beta$ (powodujące, że $(\sin\beta)^2+(\cos\beta)^2 \neq 1$) oraz zaokrąglenie wyników pośrednich. Stwierdzono, że błąd obliczeń rośnie wraz ze wzrostem liczby iteracji N. W celu oszacowania błędów obliczeń opracowano specjalny program komputerowy, który oblicza współrzędne $N = 200$ punktów (x_i, y_i) według wzoru (13). Obliczenia były przeprowadzone na 32-bitowych liczbach stałoprzecinkowych, a współczynniki $\sin\beta$ i $\cos\beta$ były reprezentowane jako 32-bitowe liczby stałoprzecinkowe równe odpowiednio $\sin(2\pi/N)$ oraz $\cos(2\pi/N)$. Dla obliczeń bezbłędnych punkty te powinny

formować koło o zadanym promieniu R. Dla wykonanych $K = 5000$ iteracji wzoru (13) powinno było powstać 5000/200=25 jednakowych, nakładających się na siebie kół o promieniu R. Wyniki obliczeń przeprowadzonych na liczbach stałoprzecinkowych reprezentuje rysunek 15a. Na podstawie rysunku można zauważyć, że wartość promienia R z każdą iteracją ulegał zmniejszeniu i dążył do zera. Dla drugiego przypadku obliczenia były prowadzone na modelu VHDL jednostki przetwarzającej RFA, w której wartości $\sin\beta$ i $\cos\beta$ były formowane z wykorzystaniem trójek Pitagorejskich - liczb całkowitych spełniających równanie pitagorasa $a^2+b^2=c^2$, np. liczby {3, 4, 5}, {12, 5, 13}, itd. Wartości liczb $\{a, b, c\}$ były określane za pomocą specjalnego programu komputerowego na podstawie powszechnie znanych wzorów (14):

$$a = 2mn,\ b = m^2-n^2,\ c = m^2+n^2,\ m > n, \tag{14}$$

gdzie m i n – liczby naturalne, względnie pierwsze, jedna z nich jest parzysta, druga – nieparzysta oraz $m > n$. Danymi wejściowymi dla programu były: wartość kąta β, dopuszczalna wartość błędu δ_{max} oraz maksymalna wartość zmiennej m (m_{max}). Program obliczał wszystkie możliwe trójki $\{a, b, c\}$ z zadanego przedziału ($m = 2,..., m_{max}$, $n = 1,..., m-1$) oraz odpowiednie wartości błędów $\delta < \delta_{max}$. Jeśli dla zadanego zbioru danych wejściowych nie udało się znaleźć rozwiązań, należało zwiększyć wartość m_{max} i uruchomić program ponownie. Na przykład, dla zadanych wartości β=10°, $m_{max} = 1000$, $\delta_{max} = 1{,}5 \cdot 10^{-3}$ trójka {92,525,533} określa kąt β z błędem 0,1%, natomiast trójka {1120, 6351, 6449} – ten sam kąt z błędem 0,002%. Wyniki działania opracowanego modelu VHDL po wykonaniu $K = 5000$ iteracji wzoru (13) dla obliczeń przeprowadzonych na 16-bitowych ułamkach pokazano na rysunku 15b. Na podstawie rysunków 15a i 15b można zauważyć, że dokładność obliczeń realizowanych z wykorzystaniem arytmetyki ułamkowej znacznie wzrosła.

a) b)

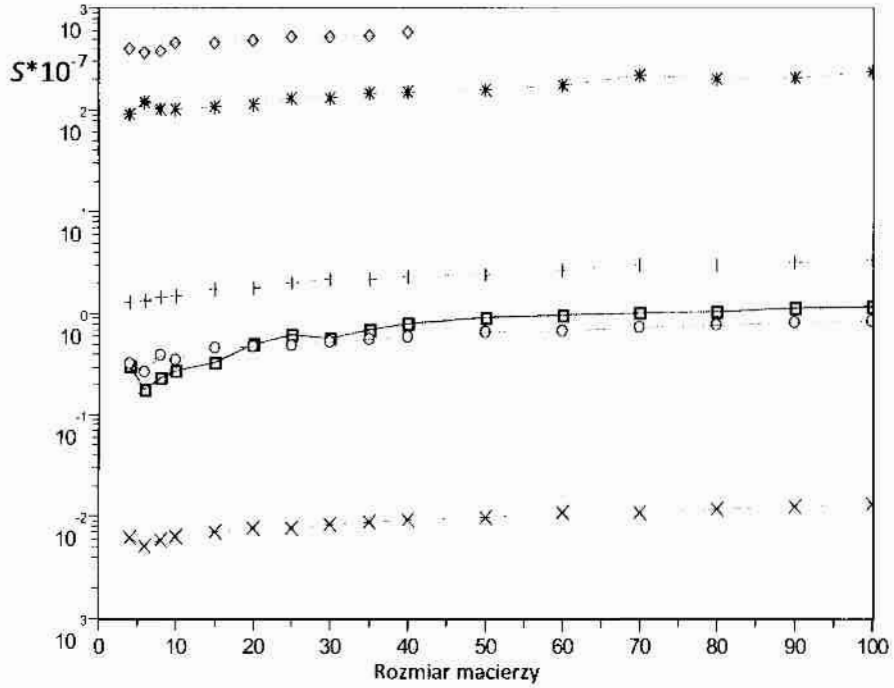


Rysunek 15. Graficzna reprezentacja wyników wielokrotnego obliczenia wyrażenia (14) w arytmetyce stałoprzecinkowej (a) i ułamkowej (b)(źródło[74] str.28)

Większą dokładność przeprowadzanych obliczeń z wykorzystaniem arytmetyki ułamkowej można zauważyć na podstawie samego porównania działania opracowanych modeli VHDL

generatorów funkcji $\sin\beta$ i $\cos\beta$. Pierwszy generator działał na 32-bitowych danych stałoprzecinkowych i generował na wyjściu sinusoidę, której amplituda ciągle się zmniejszała z powodu błędów zaokrągleń, i po 47 tys. iteracji okazała się równa zero (zamiast 1). Drugi generator działał na 20-bitowych ułamkach w arytmetyce RFA i generował sinusoidę, której amplituda po 256 tys. iteracjach wyniosła 1,029 [95].

Przeprowadzono również badania dokładności obliczeń w dla algorytmu Cholesky'ego LL^T [74, 97] na podstawie opracowanych modeli VHDL oraz narzędzi ActiveHDL[18] i Scilab[19]. Błąd obliczeń *S* był wyznaczany jako odchylenie standardowe znormalizowane w stosunku do wyników wzorcowych otrzymanych z użyciem modeli VHDL, w których wszystkie wykorzystywane zmienne miały typ *Real*[96], a więc były reprezentowane z możliwą maksymalną dokładnością jaką mógł zapewnić sprzęt komputerowy. Wyniki badania dokładności realizowanych obliczeń na dobrze uwarunkowanych macierzach o różnych rozmiarach $N<100$, których elementy były przedstawione jako ułamki 16-bitowe (◊), 18-bitowe (*), 24-bitowe (+), 26-bitowe (o) i 32-bitowe (×) oraz jako liczby 32-bitowe zmiennoprzecinkowe (□) przedstawiono na rysunku 16.



Rysunek 16. Porównanie błędów obliczeń dla ułamków 16, 18,24, 26 i 32-bitowych oraz na liczbach 32-bitowych zmiennoprzecinkowych dla algorytmu Cholesky'ego; *S*-błąd obliczeń (źródło [74] str.50).

Na podstawie zaprezentowanego wykresu z rysunku 16 można zauważyć że dokładność obliczeń przeprowadzonych na 32 bitowych ułamkach jest nie mniejsza niż przy wykorzystaniu 32-bitowego formatu zmiennoprzecinkowego. W przeciwieństwie od formatu

[18] http://www.aldec.com/activehdl/
[19] http://www.scilab.org/

zmiennoprzecinkowego, błędy obliczeń w RFA w dość dużym stopniu zależą od wielkości danych wejściowych. Jeśli skala wielkości danych wejściowych jest taka, że średnia ich wartość wynosi 1, to obliczenia w RFA mają największą dokładność. Obliczenia z wykorzystaniem 26-bitowych ułamków dają zbliżoną dokładność do obliczeń realizowanych z wykorzystaniem 32-bitowych danych zmiennoprzecinkowych. Przy przetwarzaniu danych, których średnia wartość jest większa od jedności w *M* razy, dokładność w arytmetyce RFA maleje również *M* razy, natomiast w arytmetyce zmienno-przecinkowej maleje tylko około $\sqrt[4]{M}$ razy[74, 97].

Autorskie porównanie dokładności obliczeń prowadzonych z wykorzystaniem 32-bitowych ułamków oraz 32-bitowych danych zmiennoprzecinkowych zrealizowano dla przypadku algorytmu redukcji wstecznej. Dokładność otrzymywanych wyników obliczeń była weryfikowana z wykorzystaniem modeli funkcjonalnych VHDL jednostek wykorzystujących reprezentacje: 32-bitową ułamkową, 32-bitową zmiennoprzecinkową oraz z wykorzystaniem danych typu *Real*. Zgodnie ze standardem VHDL dane typu *Real* reprezentowane są w komputerze z maksymalną możliwą dokładnością wykorzystywanego sprzętu komputerowego. Na wszystkich komputerach, na których realizowane były obliczenia typ *Real* był modelowany z wykorzystaniem reprezentacji zmiennoprzecinkowej zgodnie ze standardem IEEE 754 o precyzji *Double Extended* (80-bitów). Z tego powodu wyniki otrzymywane z wykorzystaniem danych typu *Real* uznane zostały za wzorcowe. Badania dokładności obliczeń dla algorytmu redukcji wstecznej były realizowane zgodnie ze wzorem (15), określającym znormalizowane odchylenie standardowe

$$S = \sqrt{\frac{\sum_{i=1}^{N} (x_i^* - x_i)^2}{N \cdot x_N}}, \qquad (15)$$

gdzie: *S*-błąd obliczeń, *N*- rozmiar zadanej macierzy wejściowej, x_i^*- elementy otrzymanego wektora z wykorzystaniem danych typu *Real*, x_i- elementy otrzymanego wektora z wykorzystaniem 32-bitowej reprezentacji ułamkowej lub 32-bitowej reprezentacji zmiennoprzecinkowej, x_N- ostatni element otrzymanego wektora. Badania były realizowane dla dobrze uwarunkowanych macierzy wejściowych, w których elementy leżące na jej głównej przekątnej były generowane losowo z przedziału od 0,75 do 1,25 , natomiast wartości elementów powyżej przekątnej były brane z w/w przedziału, którego granice były stopniowo pomniejszone o odległość elementów od przekątnej macierzy. Elementy wektorów wyrazów wolnych były generowane losowo również z przedziału od 0,75 do 1,25. Wyniki obliczeń wyrażenia (15) dla różnych rozmiarów macierzy wejściowych przy reprezentacji 32-bitowej ułamkowej oraz 32-bitowej zmiennoprzecinkowej przedstawiono w tabeli 16. Analiza danych z tabeli 16 świadczy o tym, że wraz ze wzrostem rozmiaru macierzy błędy arytmetyki ułamkowej zbliżają się do błędów arytmetyki zmiennoprzecinkowej (dla danych 32-bitowych typu Single).

Tabela 16. Porównanie błędów obliczeń przeprowadzonych na 32-bitowych ułamkach oraz 32-bitowych danych zmiennoprzecinkowych

Rozmiar macierzy	10	20	30	50	80	150	200	250
Błąd dla RFA ($*10^{-7}$)	6,71	16,63	13,29	23,28	19,60	33,42	41,01	60,12
Błąd dla SINGLE ($*10^{-7}$)	1,66	4,33	4,51	11,59	14,55	29,64	38,27	54,09
Stosunek błędów RFA/SINGLE	4,04	3,84	2,95	2,01	1,35	1,13	1,07	1,11

Na podstawie zaprezentowanych wyników badania dokładności obliczeń można stwierdzić, że potokowe i równoległe jednostki przetwarzające działające w arytmetyce ułamkowej mogą być efektywnie stosowane do realizacji w czasie rzeczywistym algorytmów algebry liniowej na platformie FPGA, w których wymagania dotyczące dokładności przeprowadzanych obliczeń uniemożliwiają stosowanie arytmetyki stałoprzecinkowej, jednakże zaleca się stosowanie arytmetyki zmiennoprzecinkowej o pojedynczej precyzji (32-bitowych). Ponadto jednostki pracujące w arytmetyce ułamkowej charakteryzują się mniejszą złożonością sprzętową i wyższą wydajnością w porównaniu do podobnych jednostek stało i zmiennoprzecinkowych, co zostanie zaprezentowane w kolejnych podrozdziałach.

W trakcie badań nad arytmetyką ułamkową RFA, zostało ustalone m.in., że

- jeśli liczba $x = a/b$ jest ułamkiem, np. $x = -1/9$, to w RFA liczba ta jest przedstawiona - następująco: $a = -1$ i $b = 9$; jeśli x jest liczbą całkowitą, np. 7, to $a = 7$ i $b = 1$; jeśli $x = 0$, to $a = 0$ i $b = 11.....1$; jeśli liczba x jest liczbą rzeczywistą, np. 8.512, to $a = 8512$ i $b = 1000$; przy tym w celu zwiększenia dokładności obliczeń warto wykonywać normalizację zarówno danych wejściowych, jak i wyników pośrednich (poprzez przesunięcie w lewo licznika i mianownika o jednakową liczbę bitów tak, aby co najmniej jedna z nich miała najbardziej znaczący bit tuż za bitem znaku);
- w dowolnej liczbie ułamkowej a/b tylko licznik może być przedstawiony jako liczba ze znakiem (najlepiej w kodzie uzupełnieniowym do dwóch UD, ponieważ wbudowane bloki mnożące działają na liczbach w kodzie UD, lub na liczbach bez znaku); mianownik może być przechowywany jako liczba całkowita bez znaku;
- do przechowania n-bitowej liczby zespolonej $x = a/b$ potrzebne są dwa n-bitowe liczniki i jeden n-bitowy mianownik, tj. 1,5 razy mniej bitów, niż dla przechowania tej samej liczby w formacie stałoprzecinkowym (przy zachowaniu tej samej precyzji);
- operacja porównania dwóch liczb a/b oraz c/d sprowadza się do obliczenia znaku wyrażenia $(ad - cb)$ i następnie obliczenia sumy *modulo* 2 trzech zmiennych jednobitowych, reprezentujących znak tego wyrażenia, znak b i znak d; stąd wynika, że operacja ta może być realizowana w dowolnej jednostce ALU RFA (zawierającej blok mnożenia i sumator) prawie bez dodatkowych nakładów sprzętowych;

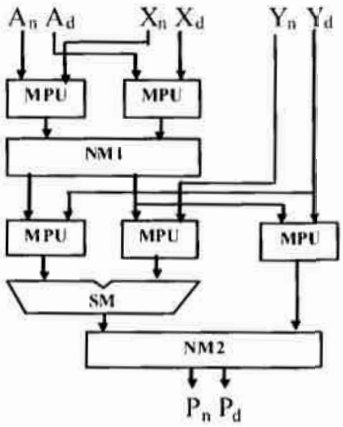
- mnożenie dwóch ułamków *a/b* i *c/d* w RFA sprowadza się do wykonania dwóch operacji mnożenia liczb *n*-bitowych, zamiast jednego mnożenia dwóch liczb *2n*-bitowych w formacie stałoprzecinkowym. Dlatego złożoność sprzętowa kombinacyjnego bloku mnożącego RFA i jego opóźnienie są około 2 razy mniejsze w porównaniu do kombinacyjnego bloku mnożącego dwie *2n*-bitowe liczby w formacie stałoprzecinkowym. Porównanie złożoności sprzętowej bloku dzielenia RFA z blokiem dzielenia dwóch *2n*-bitowych liczb w formacie stałoprzecinkowym wypada jeszcze lepiej, ponieważ dzielenie w RFA sprowadza się do wykonania dwóch operacji mnożenia liczb *n*-bitowych.

3.2. Projektowanie podstawowych bloków operacyjnych działających w arytmetyce ułamkowej.

Autor rozprawy jest współautorem opracowania jednostki arytmetyczno-logiczną ALU RFA realizującej podstawowy zbiór operacji arytmetycznych. Opracowana jednostka arytmetyczno-logiczna, jej wykorzystanie oraz porównanie z analogicznymi rozwiązaniami zaprezentowano w pracach[74,98,99,100]. Jednostka ta została zaprojektowana do realizacji następujących operacji arytmetycznych realizowanych z wykorzystaniem *n*-bitowej reprezentacji ułamkowej:

- dodawania (odejmowania) P = X+Y,
- mnożenia P = A·X,
- dzielenia P = A/X,
- mnożenia z dodawaniem (odejmowaniem) P = A·X+Y,
- dzielenia z dodawaniem (odejmowaniem) P = A/X+Y.

Struktura zaprojektowanej jednostki ALU RFA jest przedstawiona na rys. 17, gdzie A_n,A_d, X_n, X_d, Y_n, Y_d oznaczają odpowiednie liczniki i mianowniki argumentów, natomiast P_n oraz P_d licznik i mianownik wyniku realizowanej operacji.



Rysunek 17. Struktura zaprojektowanej jednostki ALU pracującej z wykorzystaniem arytmetyki ułamkowej (źrodło:[74]).

Jednostka zawiera pięć n-bitowych bloków mnożących (MPU), jeden n-bitowy sumator (SM) i dwa bloki normalizacji wyniku (NM1 i NM2). W przypadku wykonania operacji dzielenia lub dzielenia z dodawaniem wartości X_n i X_d podawane na wejścia ALU należy zamienić miejscami, a w przypadku operacji mnożenia lub dzielenia przyjąć Y = 0, tj. $Y_n = 0$ i $Y_d = 1$. W trakcie badań jednostki ALU RFA stwierdzono, że przy implementacji w układach FPGA Vitex4, bloki MPU i SM są realizowane w oparciu o wbudowane bloki DSP (DSP48Slice), natomiast bloki NM1 i NM2 z wykorzystaniem bloków CLB (Slice) układu FPGA. Stwierdzono również, że wymagana dokładność obliczeń osiąga się poprzez wybór odpowiedniej liczby bitów n w reprezentacji ułamków i konstruowanie odpowiednich bloków mnożących. W układach reprogramowalnych rodzin Virtex IIPro i Virtex 4, n –bitowe bloki mnożące RFA (w przypadku $n > 18$) realizowane są z wykorzystaniem kilku wbudowanych 18-bitowych bloków mnożących. Przykładowo do realizacji 35-bitowego bloku mnożenia MPU wykorzystane są 4 wbudowane w bloki DSP 18-bitowe układy mnożące. Szczegółowa struktura takiego 35-bitowego bloku mnożącego została przedstawiona w pracy [74]. W celu zwiększenia maksymalnej częstotliwości działania opracowana jednostka ALU RFA pracuje w 9-stopniowym trybie potokowym. Przedstawioną jednostkę wykorzystano w jednostce przetwarzającej przeznaczonej do rozwiązywania układów równań liniowych w oparciu o metodę gradientów sprzężonych [15]. Porównanie nakładów sprzętowych opracowanej 35-bitowej jednostki RFA ze znanymi ALU zmiennoprzecinkowymi [31,102], wykonującymi mniejszy zbiór operacji, chociaż z większą dokładnością przedstawiono w tabeli 17.

Tabela 17. Wyniki implementacji proponowanego ALU ze znanymi w układzie FPGA Virtex IIPro(źródło[98])

Parametry ALU	ALU RFA	ALU [31]	ALU [102]*
Liczba wykorzystanych bloków CLB (slice), Liczba wykorzystanych bloków mnożących	1005 20	4625 9	2825 9
Liczba stopni w potoku	9	34	13
Maksymalna częstotliwość działania, MHz	138	120	140

* brak operacji dzielenia

Analiza danych z tab. 17 świadczy o ponad trzykrotnie mniejszej złożoności sprzętowej jednostki RFA biorąc pod uwagę liczbę wykorzystanych bloków CLB przy porównywalnej maksymalnej częstotliwości zegara systemowego. Ponadto, opracowana jednostka ma tylko 9 stopni w potoku, tj. może efektywniej działać na krótkich tablicach danych wejściowych. Autorskie badania skupiały się głownie w obrębie opracowania różnych wariantów bloków operacyjnych realizujących podstawowe operacje arytmetyczne oraz określeniu ich parametrów po implementacji w wybranych rodzinach układów FPGA. Listę zaprojektowanych jednostek przetwarzających pracujących w arytmetyce ułamkowej realizujących wybrane operacje przedstawiono w tab. 18.

Tabela 18. Lista opracowanych projektów podstawowych bloków operacyjnych RFA

Nazwa bloku	Realizowana operacja
ADD_FR	dodawanie-odejmowanie
CMP_FR	porównanie
MUL_FR	mnożenie
MUL_ADD_FR	mnożenie z akumulacją wyniku
DIV_FR	dzielenie
SORT_FR	pierwiastkowanie (metoda iteracyjna)
INT2FR	przekształcenie liczby całkowitej w ułamek
FR2INT	przekształcenie ułamka w liczbę całkowitą

Wstępny etap projektowania w języku VHDL oraz weryfikację poprawności funkcjonowania zaprojektowanych bloków operacyjnych zrealizowano z wykorzystaniem symulatora ActiveHDL firmy Aldec, natomiast proces syntezy oraz określenie parametrów bloków operacyjnych RFA został przeprowadzony dla układów FPGA Xilinx4 xc4vSX35-12 w środowisku Xilinx ISE 9.2.04i oraz FPGA Altera StratixII Eps215F672C5 w środowisku Quartus II 7.2 SP2 Web Edition. Proces syntezy ukierunkowano na optymalizację szybkości działania projektowanych jednostek z automatycznym wykorzystywaniem wbudowanych bloków DSP. Wyniki implementacji podstawowych bloków operacyjnych arytmetyki ułamkowej przedstawiono w pracach [26,74]. W tabelach 19 oraz 20 przedstawiono najważniejsze parametry zaprojektowanych bloków operacyjnych, mianowicie liczbę wykorzystanych komórek CLB (dla układu firmy Xilinx) lub ALUT(dla układu Altera), liczbę wykorzystanych bloków DSP oraz maksymalną częstotliwość działania dla różnych dokładności reprezentacji(szerokości bitowych) danych wejściowych.

Tabela 19. Parametry wygenerowanych modeli bloków operacyjnych dla układu Xilinx xc4vSX35-12

Nazwa modułu	Liczba CLB slices + 18-bitowe bloki DSP dla szerokości danych (bit):				Max częst. pracy (MHz) przy szer. danych:			
	18	24	32	35	18	24	32	35
ADD_FR	200 +3	311 +12	372 +12	528 +12	241	170	166	149
CMP_FR	7 +2	70 +8	80 +8	84 +8	484	181	181	181
MUL_FR	188 +2	195 +8	273 +8	402 +8	220	181	181	176
DIV_FR	188+ 2	195 +8	273 +8	402 +8	220	181	181	176
SQRT_FR	544 +5	724 +20	959 +20	1262 +20	270	150	147	146
INT2FR	96	122	167	197	384	327	311	326
FR2INT	555	921	1579	1827	191	173	153	148

Tabela 20. Parametry wygenerowanych modeli bloków operacyjnych dla układu Eps215F672C5

Nazwa modułu	Liczba bloków ALUT oraz 9-bitowych bloków DSP				Max częst., MHz przy szer. danych			
	18	**24**	**32**	**35**	**18**	**24**	**32**	**35**
ADD_FR	226 +6	303 +24	396 +24	500 +24	172	146	137	125
CMP_FR	45 +4	62 +16	82 +16	90 +16	270	227	183	169
MUL_FR	132 +4	175 +16	224 +16	352 +16	225	220	196	174
DIV_FR	132 +4	175 +16	224 +16	352 +16	225	220	196	174
SQRT_FR	414 +10	532 +40	1228 +40	1680 +40	162	124	124	121
INT2FR	83	114	190	212	289	268	215	208
FR2INT	629	1026	1773	2042	138	117	102	90

W celu porównania wyników warto podkreślić, że pojedynczy blok DSP dla wymienionego układu Xilinx Virtex4 zawiera dwa 18-bitowe bloki mnożące[20], natomiast bloki DSP w układach Altera Stratix II zawierają cztery takie bloki mnożące[21]. Na podstawie powyższych wyników można wywnioskować, że w obu układach FPGA wykorzystanych zostaje identyczna liczba wbudowanych 18-bitowych bloków mnożących zawartych w blokach DSP (blok DSP w rodzinie Virtex4 zawiera 2 18-bitowe bloki mnożenia, w rodzinie StratixII - 4 takie bloki), natomiast dla układu Virtex4 uzyskano wyższe częstotliwości pracy. Warto zauważyć, że blok dzielenia zajmuje identyczne zasoby co blok operacji mnożenia, gdyż w arytmetyce ułamkowej operacja dzielenia sprowadza się do operacji mnożenia (przez odwrotność drugiego operandu operacji). Ponadto zbadano, jaki wpływ na liczbę wykorzystanych bloków CLB i częstotliwość pracy jednostek arytmetycznych RFA ma wykorzystanie wbudowanych w układ Xilinx xc4vSX35-12 bloków DSP. Parametry zaprojektowanych jednostek z wykorzystaniem oraz z pominięciem wbudowanych bloków DSP przedstawiono w tabeli 21. Zgodnie z przewidywaniami wykorzystanie wbudowanych bloków DSP powoduje wykorzystanie znacznie mniejszej liczby bloków CLB oraz zwiększenie maksymalnej częstotliwości pracy bloku operacyjnego RFA.

[20] http://www.xilinx.com/support/documentation/data_sheets/ds112.pdf

[21] http://www.altera.com/products/devices/stratix-fpgas/stratix-ii/stratix-ii/overview/st2-overview.html

Tabela 21. Parametry 18-bitowych bloków operacyjnych FRA implementowanych w układzie xc4vSX35-12 z wykorzystaniem oraz bez wykorzystania wbudowanych bloków DSP

Nazwa modułu	Stopnie potoku	CLB slices + bloków DSP	Max częst. (MHz)
ADD_FR	5	200+3	241
		866+0	140
CMP_FR	4	7+2	484
		479+0	140
MUL_FR	4	188+2	220
		592+0	140
DIV_FR	4	188+2	220
		592+0	140
SQRT_FR	48	544+5	270
		1171+0	142

Podczas optymalizacji bloku pierwiastkowania stwierdzono, że pierwiastkowanie w arytmetyce ułamkowej najlepiej wykonać w sposób iteracyjny z wykorzystaniem metody Newtona. W każdej iteracji wykonywana jest jedna operacja dzielenia, dodawania i przesunięcia o 1 bit w prawo. Dodanie operacji pierwiastkowania do listy operacji wykonywanych przez jednostkę ALU RFA, która zawiera blok mnożenia i sumowania, prawie nie zwiększa jej złożoności sprzętowej, ponieważ dzielenie w RFA sprowadza się do wykonania dwóch operacji mnożenia. Ponadto, blok pierwiastkowania powinien zawierać układ pamięci stałej ROM o niewielkiej pojemności, w którym będą przechowywane początkowe (przybliżone) wartości wyniku. Wydłuża się jednak czas obliczeń, ponieważ w celu uzyskania wysokiej dokładności wyniku trzeba wykonać kilka iteracji (zwykle od 3 do 5). Ostatecznie udało się zmniejszyć pojemność ROM w bloku pierwiastkowania do 16 komórek, podając na wejścia adresowe ROM 4 starsze bity z wyjścia układu normalizacji wyniku[96].

Wszystkie zaprojektowane bloki operacyjne zostały przedstawione w postaci syntezowalnych modeli VHDL i są sparametryzowane. Głównym parametrem każdego modelu jest szerokość bitowa danych wejściowych i wyjściowych. Możliwe jest również definiowanie liczby stopni w potokowym trybie pracy, co ma zasadniczy wpływ na opóźnienie oraz maksymalną częstotliwość pracy zaprojektowanej jednostki. Większość opracowanych bloków ma opóźnienie potoku równe 4 taktom zegarowym (bloki dodawania oraz mnożenia z akumulacją wyniku posiada opóźnienie 5 taktów), natomiast kolejne wyniki pojawiają się na wyjściach z każdym taktem zegarowym. Wyjątkiem jest blok pierwiastkowania, w którym wykonywano od 4 (ułamki 35-bitowe) do 6 iteracji algorytmu Newtona. Złożoność sprzętowa tego bloku została znacznie zwiększona w stosunku do złożoności sprzętowej pozostałych bloków w celu zachowania podobnej częstotliwości pracy. Pomimo opóźnienia potoku wynoszącego 48 taktów zegarowych w tym bloku, dzięki

równoległemu przetwarzaniu 12 danych wejściowych, wyniki pojawiają się na wyjściu bloku co 4 takty. Porównując wyniki implementacji zaprojektowanych bloków operacyjnych dla obu rodzin układów: Virtex4 firmy Xilinx oraz StratixII firmy Altera warto zauważyć, że dla układu Virtex4 uzyskano wyższe częstotliwości pracy o około 20-25%.

W oparciu o projekty bloków operacyjnych FRA został opracowany autorski program-generator IP-core generujący na wyjściu implementowalne modele VHDL wybranych przez użytkownika operacji arytmetycznych realizowanych w arytmetyce ułamkowej[26]. Szczegółowy opis zaprojektowanego generatora wraz z przykładowymi wygenerowanymi modelami VHDL przedstawiono w załączniku 1.

W kolejnym etapie badań porównano złożoność sprzętową oraz maksymalną częstotliwość pracy zaprojektowanych bloków operacyjnych pracujących w arytmetyce ułamkowej RFA z analogicznymi blokami wykorzystującymi reprezentację liczbową stałoprzecinkową oraz zmiennoprzecinkową. Modele analogicznych bloków zostały wygenerowane z wykorzystaniem generatorów Coregen, wchodzących w skład popularnego pakietu oprogramowania Xilinx Foundation ISE v.9.2. Do wygenerowania bloków funkcjonalnych dla podstawowych operacji arytmetycznych wykorzystano generatory: „Multiplier 10.0", „Divider 1.0", „Adder/Subtractor 7.0". Synteza logiczna wszystkich bloków była przeprowadzona w środowisku Foundation ISE v.9.2 dla układu Xilinx FPGA Virtex4 (xc4vSX35-12).

Wymienione generatory Xilinx IPCore, często nie pozwalały na określanie w szerokim zakresie parametrów wejściowych projektowanych modułów funkcjonalnych. Dla przykładu generator Divider 1.0 nie umożliwiał wygenerowania stałoprzecinkowego bloku operacji dzielania dla danych wejściowych o szerokości większej niż 32 bity, natomiast generator Multiply Accumulator 4.0 nie pozwalał na ustawienie szerokości danych wejściowych powyżej 18 bitów. Pozostały parametry generowanych bloków starano się tak definiować aby funkcjonalnie odpowiadały zaprojektowanym blokom operacyjnym wykorzystującym arytmetyką ułamkową, np. uzyskiwanie wyniku operacji z każdym taktem zegarowym lub możliwie zbliżoną wartość liczby stopni w potoku. Z powodu opisanych ograniczeń w tabeli 22 przedstawiono parametry tylko tych bloków wykorzystujących reprezentację stało i zmiennoprzecinkową, których modele udało się uzyskać z wykorzystaniem opisanych generatorów.

Tabela 22. Podstawowe parametry wybranych bloków operacyjnych RFA, stało- i zmiennoprzecinkowych po ich implementacji w układzie Xilinx Virtex 4 (xc4vSX35-12)

Rodzaj bloku	RFA									Xilinx CoreGen (Multiplier 10.0, Divider 1.0, Adder/Subtractor 7.0)					
	ułamki 16-bitowe			ułamki 18-bitowe			ułamki 35-bitowe			liczby 32-bitowe stałoprzecinkowe			(liczby 32-bitowe zmiennoprzecinkowe)* (liczby 64-bitowe stałoprzecinkowe)**		
	Slice +DSP	MHz	Stopni potoku	Slice +DSP	MHz	Stopni potoku	Slice +DSP	MHz	Stopni potoku	Slice +DSP	MHz	Stopni potoku	Slice +DSP	MHz	Stopni potoku
X*Y	122+2	404	4	188+2	220	4	402+8	176	4	17+4	264	4	(51+10)**	88**	4**
X/Y	122+2	404	4	188+2	220	4	402+8	176	4	1229+0	254	36	(251+10)*	117*	10*
X+Y	134+3	247	5	200+3	241	5	372+12	166	5	96+0	416	5	(176+0)**	383**	5**

Wyniki przedstawione w tabeli 22 dowodzą tezy o mniejszej złożoności sprzętowej lub większej wydajności zaprojektowanych bloków funkcjonalnych RFA w stosunku do odpowiedników stało i zmiennoprzecinkowych przy zachowaniu podobnej dokładności obliczeń. Dla przykładu 16-bitowe bloki mnożenia RFA są realizowane w oparciu o dwa bloki DSP (DSP48Slice) zamiast 4 takich bloków w 32-bitowym bloku stałoprzecinkowym. Większa liczba wykorzystanych bloków Slice w blokach RFA (122 zamiast 17) prawie nie wpływa na porównanie złożoności sprzętowej rozpatrywanych bloków z tego powodu, że złożoność sprzętowa 122 takich bloków jest ok. 4 razy mniejsza od złożoności sprzętowej jednego bloku DSP. Ponadto, te 122 bloki Slice są wykorzystane do realizacji bloku normalizacji wyniku co ma korzystny wpływ na dokładność obliczeń, natomiast stałoprzecinkowy blok mnożący takiego bloku nie posiada. Należy podkreślić, że 16-bitowe bloki mnożące RFA działają z maksymalną częstotliwością 404 MHz w przeciwieństwie do 264 MHz w przypadku 32-bitowych bloków stałoprzecinkowych dla jednakowej liczby stopni w potoku. Oznacza to, że bloki RFA mają o ok. 55% większą wydajność. W przypadku 35-bitowych ułamków i 64-bitowych liczb stałoprzecinkowych przewagą w złożoności sprzętowej bloku mnożącego RFA maleje do 25%, ale z kolei przewaga w wydajności bloku RFA okazuje się ponad 2-krotna; mianowicie, maksymalna częstotliwość pracy wynosi 176 MHz w stosunku 88 MHz. Dodatkowo 35-bitowy blok mnożący RFA cechuje się większą dokładnością realizowanych obliczeń, w stosunku do jego 64-bitowego odpowiednika stałoprzecinkowego. W przypadku, gdy użytkownik modułu Multiplier 10.0 wybierze opcję „pełny wynik"(128-bitowy), wówczas liczba wykorzystanych bloków DSP wzrośnie z 10 do 16, co oznacza prawie dwukrotną przewagę w złożoności sprzętowej bloku mnożącego RFA. Najkorzystniej wypada porównanie 16-bitowych bloków dzielenia RFA z 32-bitowymi blokami stałoprzecinkowymi, zarówno pod względem wykorzystanych bloków CLB (Slice), jak i wydajności. Należy zaznaczyć, że do implementacji bloku stałoprzecinkowego nie zostały wykorzystane

wbudowane bloki DSP. Z tego powodu blok ten cechuje dziesięciokrotnie większa liczba wykorzystanych bloków Slice oraz 36-stopniowym potokiem. Prawdopodobnie z tego powodu specjaliści firmy Xilinx w projekcie 32-bitowego bloku dzielenia zmiennoprzecinkowego zmienili sposób wykonania tej operacji na iteracyjny, co spowodowało możliwość wykorzystania bloków DSP oraz skrócenie liczby stopni w potoku do 10. Pomimo tego, 35-bitowy blok dzielenia RFA zapewniający podobną dokładność obliczeń co 32-bitowy blok dzielenia zmiennoprzecinkowego cechuje się jednak mniejszą złożonością sprzętową (około 20%), w przybliżeniu 50% większą wydajnością oraz 2,5 razy mniejszą liczbą stopni w potoku. Z porównywanych operacji arytmetycznych operacja dodawania (odejmowania) w arytmetyce ułamkowej jest najbardziej skomplikowana. Realizacja tej operacji w arytmetyce ułamkowej wymaga wykonania 3 operacji mnożenia i jednego dodawania(odejmowania). Z tego powodu złożoność sprzętowa 16-bitowego sumatora RFA jest znacznie większą od złożoności sprzętowej 32-bitowego sumatora stałoprzecinkowego (m.in. dlatego, że wykorzystuję 3 bloki DSP), a jego wydajność jest ok. 2 razy mniejsza. Trudno sobie jednak wyobrazić jednostkę przetwarzającą do realizacji algorytmów algebry liniowej, która powinna realizować wyłącznie operację dodawania. W przypadku gdy do zbioru operacji wykonywanych przez jednostkę RFA należałoby dołączyć jeszcze przynajmniej jedną operację arytmetyczną, choćby tylko mnożenie, wówczas znajdujące się w jednostce wbudowane bloki mnożące lub DSP można wykorzystać w obu operacjach: dodawania i mnożenia. Więc sumaryczna liczba wykorzystanych bloków prawie się nie zwiększy.

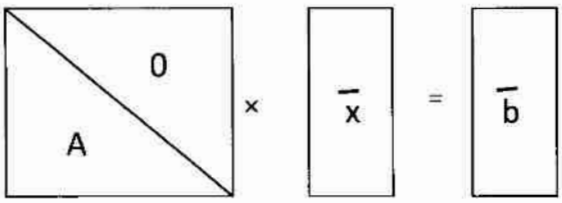
3.3. Projekty akceleratorów wybranych algorytmów algebry liniowej pracujące w arytmetyce ułamkowej

W trakcie realizacji badań nad wykorzystaniem arytmetyki ułamkowej na platformie FPGA opracowano modele bloków operacyjnych realizujących podstawowe operacje arytmetyczne oraz jednostkę arytmetyczno logiczną ALU realizującą zbiór podstawowych operacji opisane w pracach[26,74,96,97]. Zaprojektowaną jednostkę ALU wykorzystano do budowy wyspecjalizowanych akceleratorów realizujących wybrane algorytmy algebry liniowej. Zaprojektowano m.in. akceleratory przeznaczone do realizacji algorytmu rozkładu Cholesky'ego LL^T[74,98,99] oraz algorytmu rozwiązywania układu równań Gaussa-Jordana[100]. Zaprojektowany został również wyspecjalizowany procesor do sprzętowej realizacji analizy autoregresyjnej sygnałów[106]. Udział autora rozprawy w badaniach polegał na projektowaniu, optymalizacji i badaniu parametrów podstawowych bloków operacyjnych wykorzystujących arytmetykę ułamkową oraz na opracowaniu generatora modeli VHDL tych bloków. Zaprojektowane akceleratory cechują się mniejszą złożonością sprzętową w porównaniu do innych analogicznych rozwiązań, co zostało wykazane na podstawie porównania parametrów wykorzystywanego ALU pracującego z wykorzystaniem arytmetyki ułamkowej RFA. Ponadto zaprojektowane ALU RFA cechuje się mniejszą liczbą stopni w potoku, przy zachowaniu podobnej maksymalnej częstotliwości pracy. Jedynie

operacja pierwiastkowania w zaprojektowanym ALU RFA charakteryzuje się dużą liczbą stopni w potoku (45 stopni[99]), jednak w wyniku nowej organizacji obliczeń algorytmu dekompozycji Cholesky'ego LL^T, wszystkie operacje pierwiastkowania wykonywane są w trybie potokowym na ostatnim etapie obliczeń algorytmu. W związku z tym duża liczba stopni w potoku dla operacji pierwiastkowania nie wprowadziła znaczącego opóźnienia realizacji całego algorytmu. Porównanie parametrów wykorzystywanego ALU RFA z odpowiednikami wykorzystującymi reprezentację zmiennoprzecinkową przedstawiono w poprzednim podrozdziale w tabeli 17. W sumie zaprojektowano kilka akceleratorów wybranych algorytmów algebry liniowej, jednak największy wkład autorski został włożony w projekty układów realizujących algorytm redukcji wstecznej[101] oraz algorytm rozkładu LU dla macierzy pasmowej Hessenberga[74].

3.3.1. Sprzętowy akcelerator algorytmu redukcji wstecznej zaimplementowany na platformie FPGA.

Metoda redukcji wstecznej wraz z metodą podstawienia (*ang. back and forward substitution*) są podstawowymi metodami rozwiązywania układów równań liniowych $Ax=b$, w których macierz współczynników układu $A(N,N)$ jest macierzą trójkątną górną lub dolną (rys.18).

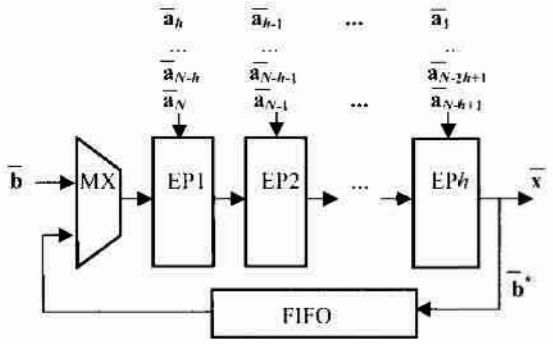


Rysunek 18. Reprezentacja danych w algorytmie redukcji wstecznej rozwiązywania układu równań liniowych.

Obie metody cechują się złożonością obliczeniową – $O(N^2/2)$ dla operacji mnożenia z odejmowaniem oraz $O(N)$ dla operacji dzielenia oraz jednakowym sposobem przetwarzania elementów macierzy A. Z tego powodu jednostka przetwarzająca przeznaczona do realizacji metody redukcji wstecznej może być również wykorzystywana do realizacji metody podstawienia. Metoda redukcji wstecznej może być realizowana zgodnie z poniższym fragmentem programu (1), gdzie przez $a(i, j)$ oznaczono odpowiednie elementy macierzy (trójkątnej dolnej) współczynników zadanego układu równań, $b(i)$ reprezentują elementy wektora wyrazów wolnych, natomiast $x(i)$ – poszukiwane elementy wektora pierwiastków:

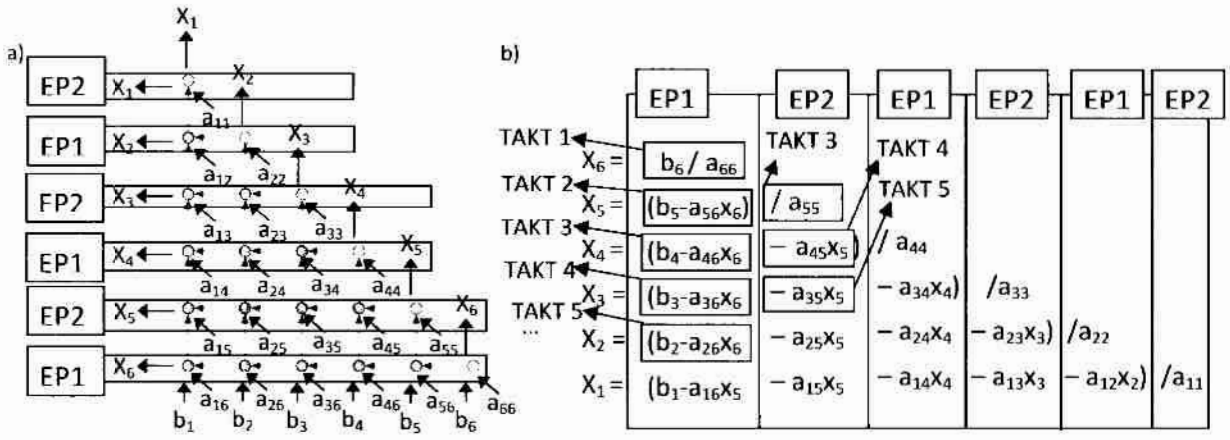
```
for i = N downto 1 do
begin
for j = N downto i+1 do
     b(i) = b(i) - a(i, j)*x( j);          (1)
x(i) = b(i)/a(i, i);
end;
```

Rysunek 19 przedstawia strukturę S_1 równoległej jednostki przetwarzającej realizującej algorytm redukcji wstecznej(1). Struktura ta została otrzymana w wyniku odwzorowania algorytmu w architekturę macierzy procesorowych zgodnie z metodą P. Quintona przedstawioną w pracy[45]. Zaprojektowana jednostka zawiera *h* jednakowych elementów przetwarzających (EP), gdzie $h < N$, na których wejścia podawane są szeregowo elementy odpowiednich kolumn macierzy współczynników A – od ostatniej kolumny $\bar{a}_N$ do pierwszej $\bar{a}_1$ przy założeniu, że *N/h* jest liczbą całkowitą. Na wejście pierwszego elementu przetwarzającego EP1, poprzez multiplekser MX, dodatkowo podawane są w sposób szeregowy najpierw elementy wektora wyrazów wolnych $\bar{b}$, a następnie elementy przekształconego wektora $\bar{b}^*$ uzyskane w trakcie wykonywania obliczeń z wyjścia bloku kolejki FIFO. Elementy wektora pierwiastków $\bar{x}$ otrzymywane są z wyjścia ostatniego EP.



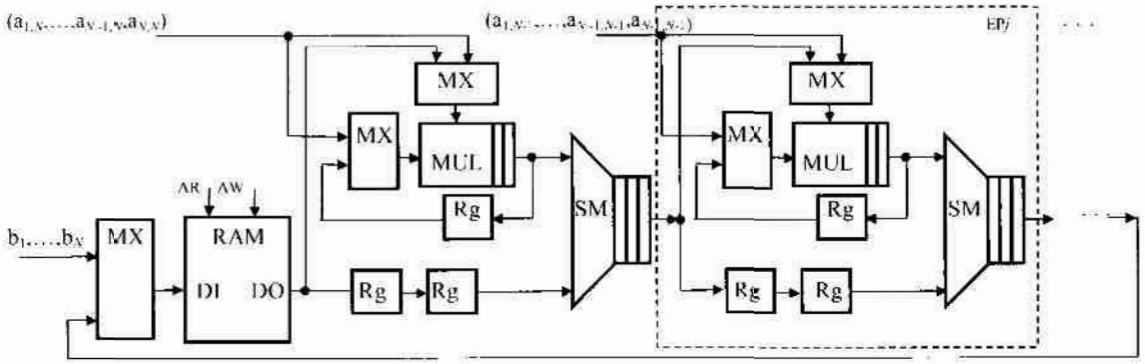
Rysunek 19. Struktura S_1 równoległej jednostki przetwarzającej realizującej metodę redukcji wstecznej (źródło[101]).

Na rysunku 20 przedstawiono przykładowe odwzorowanie przestrzenne i czasowe dla algorytmu redukcji wstecznej dla struktury zobrazowanej na rys.20, dla przypadku dwóch elementów przetwarzających EP1 oraz EP2. Rysunek 20a przedstawia odwzorowanie przestrzenne przedstawione na grafie zależności informacyjnych, polegające na przypisaniu operacji algorytmu(węzłów grafu) do jednego z elementów przetwarzających EP. Pierwszy element przetwarzający EP1 realizuje operacje wykorzystujące elementy z 6,4 i 2 kolumny macierzy A, natomiast drugi EP2 realizuje operacje z pozostałych kolumn. Elementy wektora pierwiastków x_6,x_4 i x_2 przekazywane są z pierwszego elementu przetwarzającego do drugiego – ostatniego. Na rysunku 20b przedstawiono przykładowe możliwe odwzorowanie czasowe operacji algorytmu dla pierwszych 5 taktów (makrotaktów – kroków algorytmu), mające na celu zobrazowanie kolejność wykonywanych operacji. W pierwszym kroku algorytmu realizowane są operacje wykorzystujące elementy macierzy A z ostatnich dwóch kolumn: w EP1 z kolumny 6 oraz w EP2 z kolumny 5. W kolejnym kroku realizowane są operacje wykorzystujące dwie kolejne kolumny macierzy A.



Rysunek 20. Przykład odwzorowania algorytmu redukcji wstecznej dla architektury przedstawionej na rys.19 zawierającej dwa elementy przetwarzające (dla N=6 i h=2): odwzorowanie przestrzenne dla grafu zależności informacyjnej(a), odwzorowanie przestrzenne i czasowe zobrazowane na operacjach arytmetycznych algorytmu(b).

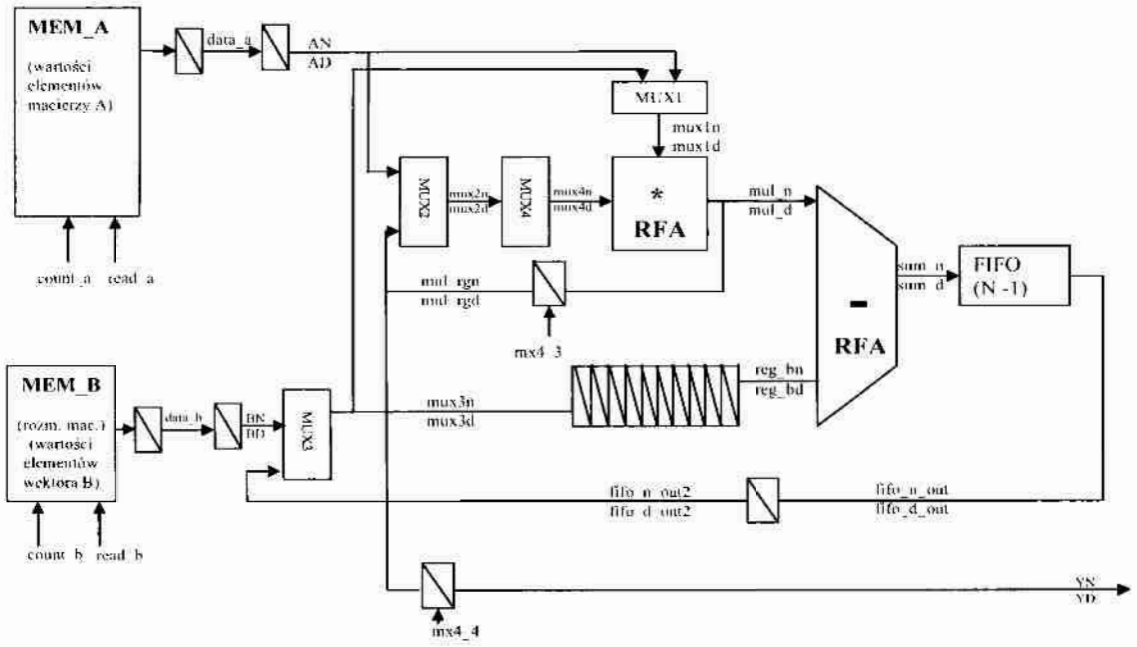
Podczas projektowania struktury przedstawionej na rys.19 założono, że czas realizacji wszystkich rodzajów operacji: mnożenia z odejmowaniem oraz dzielenia jest jednakowy i wynosi 1 takt zegarowy lub 1 takt wirtualny (makrotakt), równy *p* taktom zegarowym. Warto zaznaczyć że każdy element przetwarzający prezentowanej struktury wykonuje oba rodzaje operacji arytmetycznych: mnożenie z odejmowaniem oraz dzielenie. Z tych powodów implementacja takiej struktury zawierającej nawet niewielką liczbę elementów przetwarzających EP (dla małych wartości *h*) wymaga dużych nakładów sprzętowych, a maksymalna częstotliwość działania jednostki będzie dość niska w stosunku do maksymalnej częstotliwości działania układu FPGA [1,102,103]. Efektywniejsze wykorzystanie zasobów układu FPGA oraz wyższa maksymalna częstotliwość pracy projektowanego akceleratora została uzyskana dzięki zamianie arytmetyki stało lub zmiennoprzecinkowej na arytmetykę ułamkową oraz dzięki wprowadzeniu potokowego trybu obliczeń. Wykorzystanie arytmetyki ułamkowej RFA oraz potokowego trybu obliczeń spowodowało wprowadzenie istotnych zmian w strukturze zarówno macierzy procesorowej, jak i jej poszczególnych EP. Powiązane jest to z tym, że w blokach operacyjnych działających w arytmetyce ułamkowej każda liczba x jest przedstawiona w postaci ułamka wymiernego *a/b*, gdzie *a* i *b* są *n*-bitowymi liczbami całkowitymi. Dzięki zastosowaniu arytmetyki ułamkowej operacje dzielenia i mnożenia można realizować znacznie szybciej w blokach operacyjnych pracujących z większą częstotliwością maksymalną, przy wykorzystaniu mniejszych zasobów sprzętowych, w porównaniu do analogicznych bloków stałoprzecinkowych, co zostało wykazane w poprzednim podrozdziale. Uproszczona budowa nowej równoległej jednostki przetwarzającej przedstawiona jest na rys. 21, gdzie RAM oznacza blok pamięci dwuportowej z wejściami adresowymi zapisu AW i odczytu AR danych.



Rysunek 21. Struktura równoległej jednostki przetwarzającej realizującej metodę podstawienia w arytmetyce RFA z uwzględnieniem 5-stopniowego potokowego trybu obliczeń(źródło[101]).

Uproszczenie schematu jednostki polega na tym, że magistrale danych nie są podzielone na magistrale liczników i mianowników, zgodnie z reprezentacją ułamkową RFA oraz że pominięte są bloki sterowania. Dodatkowo przyjęto, że blok mnożenia RFA ma 2 stopnie w potoku, a blok dodawania RFA – 3 stopnie. Powoduje to, że czas wykonania każdej operacji mnożenia z odejmowaniem lub dzielenia wynosi 5 taktów zegarowych, ale równocześnie w każdym EP jednostki przetwarzane jest 5 elementów macierzy współczynników A. Struktura taka cechuje się wysokim, bliskim 100% stopniem obciążenia poszczególnych elementów przetwarzających EP. Ze względu na obiecujące oszacowane parametry nowej architektury jednostki przetwarzającej kolejnym etapem badań autorskich była próba implementacji w układzie FPGA Virtex4 kilku jej wariantów.

Na rys. 22 przedstawiono strukturę opracowanego akceleratora do realizacji algorytmu redukcji wstecznej zawierającego jeden potokowy element przetwarzający. Ze względu na taką samą budowę elementów przetwarzających w architekturze równoległej przedstawionej na rys. 21 opracowana jednostka może być bezpośrednio wykorzystana do budowy jednostki równoległej. Przedstawiona na rys. 22 struktura również została uproszczona, gdyż każda magistrala danych została zaimplementowana w postaci dwóch magistrali: 35-bitowej magistrali licznika oraz 35-bitowej magistrali mianownika. Ponadto na schemacie pominięto blok sterowania wraz z liniami rozprowadzającymi sygnały sterujące.



Rysunek 22. Projekt układu potokowego realizującego algorytm redukcji wstecznej w arytmetyce ułamkowej wraz z nazwami sygnałów wykorzystanymi w procesie implementacji(źródło[101]).

Do budowy jednostki wykorzystano dwa bloki pamięci RAM: MEM_A oraz MEM_B, posiadające wspólny układ sterujący. Pierwszy blok MEM_A przechowuje wartości elementów macierzy *A*, natomiast drugi blok MEM_B - rozmiar macierzy *A* oraz wartości elementów wektora $\overline{b}$ (rys.21). Rozmiar macierzy wejściowej ma oczywisty wpływ na rozmiar wykorzystywanej pamięci, natomiast nie wpływa bezpośrednio na liczbę wykorzystywanych komórek CLB układu FPGA, gdyż bloki RAM zostały opisane zgodnie z zaleceniami producenta układu FPGA [104] i do jej implementacji wykorzystywane są wbudowane bloki pamięci BRAM. Rozmiar zadanej macierzy wyjściowej był zczytywany z pierwszej komórki pamięci MEM_B w celu odpowiedniego ustawienia bloku sterowania. Do budowy akceleratora wykorzystano również dwa bloki operacyjne pracujące w arytmetyce ułamkowej RFA: blok mnożenia oraz blok odejmowania. Bloki te realizowane są w oparciu o wbudowane bloki DSP48. Bloki operacyjne RFA działają wyłącznie w trybie potokowym, dlatego maksymalna częstotliwość ich działania w dużym stopniu zależy od liczby stopni w potoku. Szczegółowe badanie parametrów bloków operacyjnych zostały opisane w poprzednim podrozdziale. Warto przypomnieć, że dzięki zastosowaniu arytmetyki RFA blok dzielenia jest zrealizowany w oparciu o 2 bloki mnożenia. W projekcie jednostki wykorzystano również kolejkę FIFO o zmiennej długości, ponieważ liczba niezbędnych do zapamiętania wyników pośrednich zmniejsza się o jeden po przetworzeniu każdej kolejnej kolumny macierzy *A*. Przeprowadzono autorskie badania dwóch sposobów implementacji kolejki FIFO w układach FPGA firmy Xilinx: z wykorzystaniem bloków CLB (Slice) jako tzw. *„shift register"* z wykorzystaniem przerzutników D typu *„flip-flop"* oraz z wykorzystaniem

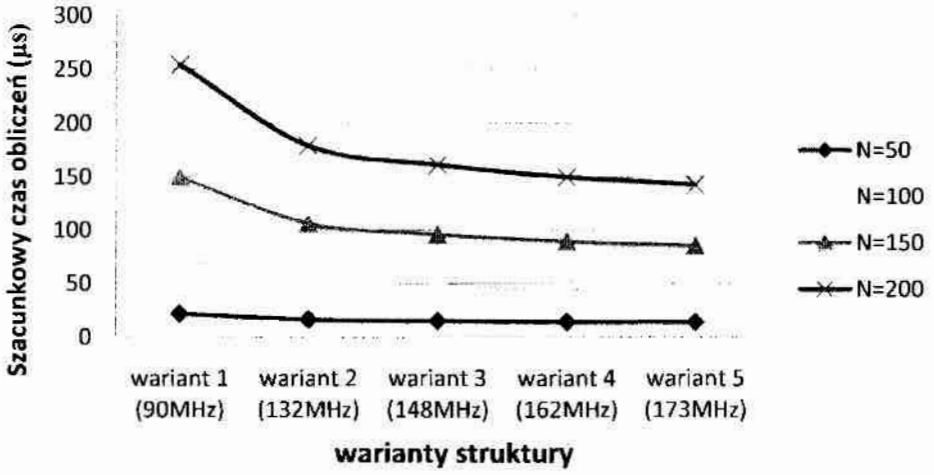
odpowiednich gotowych bloków - prymitywów, w tym przypadku bloków SRL16 – implementowanych jako *„dynamic shift register"*. Wybór sposobu implementacji zależy m.in. od dokładnego opisu wyżej wymienionych bloków w języku opisu sprzętu, np. VHDL [102,104]. Sposób implementacji kolejki FIFO okazał się na tyle istotny dla parametrów całego układu, iż postanowiono dokładniej zbadać jej parametry przy zastosowaniu obu rodzajów implementacji. Stwierdzono, że kolejka FIFO zrealizowana jako *„shift register"* może pracować z częstotliwością zbliżoną do maksymalnej częstotliwości układu, tj. 550MHz dla układu Xilinx xc4vSX35-12, natomiast maksymalna częstotliwość pracy kolejki FIFO zrealizowanej w oparciu o bloki SRL16 *„dynamic shift register"* wynosiła około 350MHz. W związku z tym, że pierwszy sposób realizacji kolejki FIFO wymaga wykorzystania większej liczby bloków *Slice*, w przypadku wzrostu jej długości obserwowano zmniejszenie maksymalnej częstotliwości pracy całej jednostki przetwarzającej. Dla przykładu w jednym z wariantów układu z kolejką FIFO zrealizowaną jako *„shift register"* o długości mniejszej od 100 komórek układ mógł pracować z częstotliwością około 147MHz. Dla kolejki o długości 110 komórek maksymalna częstotliwość pracy malała do wartości 143MHz, natomiast dla długości kolejki równej 128 komórek częstotliwość ta wynosiła już 133MHz.W przypadku implementacji kolejki FIFO jako *„dynamic shift register"* zarówno dla długości kolejki 128, jak i 256 komórek nie obserwowano jeszcze zmniejszenia częstotliwości pracy całego układu. Na podstawie opisanych badań zdecydowano się na implementację kolejki FIFO z wykorzystaniem prymitywów SRL16, tj. jako *„dynamic shift register"*. W celu zapewnienia odpowiedniego przepływu danych wykorzystano również bloki multiplekserów sterowane ze wspólnego bloku sterującego. Blok ten generował sygnału sterujące dla wszystkich multiplekserów w jednostce oraz dla wykorzystywanych bloków pamięci. Dla zapewnienia odpowiedniej synchronizacji danych zastosowano blok rejestrów opóźniających za multiplekserem MUX3, którego długość jest zależna od liczby stopni w potoku bloku mnożenia/dzielenia. W celu uzyskania większej maksymalnej częstotliwości pracy dodano dodatkowe stopnie rejestrów pomiędzy blokami pamięci oraz multiplekserami MUX1, MUX2 i MUX3. Wybór miejsca wstawiania dodatkowych rejestrów dokonywany był na podstawie komunikatów narzędzia do syntezy, określających połączenia, które powodowały największe opóźnienia sygnałów. Dalsze dodawanie dodatkowych stopni rejestrów w celu zwiększenia maksymalnej częstotliwości pracy okazało się bezcelowe, gdyż źródłem największych opóźnień w projektowanym układzie były połączenia pomiędzy poszczególnymi wbudowanymi blokami DSP. Projekt jednostki z rysunku 18 został opracowany w języku VHDL. Weryfikację projektu przeprowadzono z wykorzystaniem symulatora Active-HDL 7.2 firmy Aldec, natomiast syntezę i implementację zrealizowano z wykorzystaniem środowiska Xilinx ISE Foundation 9.2. Zmieniając liczbę stopni potoku w blokach mnożenia i odejmowania oraz wprowadzając dodatkowe rejestry na wyjściu bloków pamięci opracowano 5 wariantów struktury układu, które różniły się liczbą taktów zegarowych niezbędnych do realizacji całego algorytmu oraz maksymalną częstotliwością pracy. Wzory określające liczbę taktów pracy w zależności od rozmiaru wejściowej macierzy współczynników wraz z maksymalną częstotliwością pracy dla każdego z pięciu wariantów

układu przedstawiono w tabeli 23. W zestawieniu parametrów zaprojektowanych wariantów jednostki przetwarzającej pominięto złożoność sprzętową (określaną w komórkach CLB), ze względu na fakt, że wartości te były bardzo zbliżone.

Tab. 23. Podstawowe parametry różnych wariantów zaprojektowanej jednostki RFA w układzie Xilinx xc4vSX35-12 (35-bitowe liczniki i mianowniki)

Wariant układu	Charakterystyka struktury			Parametry pracy	
	Blok mnożenia (stopnie potoku)	Blok odejmowania (stopnie potoku)	Dodatkowe rejestry bl. pamięci	Maksymalna częstotliwość pracy (MHz)	Liczba taktów potrzebnych do realizacji obliczeń (N-rozmiar macierzy A)
Wariant 1	3	4	NIE	90	$T = \sum_{i=0}^{N-2} (N - i + 14) + 7$
Wariant 2	5	6	NIE	132	$T = \sum_{i=0}^{N-2} (N - i + 18) + 9$
Wariant 3	5	6	TAK	148	$T = \sum_{i=0}^{N-2} (N - i + 19) + 10$
Wariant 4	7	6	TAK	162	$T = \sum_{i=0}^{N-2} (N - i + 21) + 12$
Wariant 5	7	8	TAK	173	$T = \sum_{i=0}^{N-2} (N - i + 23) + 12$

Dzięki wprowadzeniu kolejnych stopni potoku wyeliminowano największe opóźnienia w strukturze jednostki, co zaowocowało zwiększeniem jej maksymalnej częstotliwości pracy. Jednak wraz ze wzrostem stopni w potoku, któremu odpowiadał wzrost liczby rejestrów, wydłużał się czas wykonania algorytmu. Liczba taktów, niezbędna do realizacji algorytmu, najbardziej zależała od liczby stopni potoku w bloku mnożenia/dzielenia, a następnie od liczby stopni potoku w bloku odejmowania. Najmniejszy wpływ miało zaś wprowadzenie dodatkowych rejestrów na wyjściu bloków pamięci. Jako przykład na rys. 23 przedstawiono wykresy określające szacunkowy czas realizacji obliczeń całego algorytmu redukcji wstecznej dla wszystkich opracowanych pięciu wariantów jednostki RFA przy maksymalnej częstotliwości pracy, dla rozmiarów macierzy współczynników A odpowiednio: 10, 100, 150 i 200.



Rysunek 23. Porównanie czasu realizacji algorytmu redukcji wstecznej w różnych wariantach jednostek RFA(źródło[101]).

Na podstawie wykresu przedstawionego na rysunku 19 można wywnioskować, że kolejne modyfikacje wprowadzane w kolejnych wariantach struktury zwiększały ogólną wydajność zaprojektowanego układu. Wprowadzanie dodatkowych stopni potoku oraz dodatkowych stopni rejestrów za blokami pamięci w większym stopniu zwiększało maksymalną częstotliwość pracy, niż liczbę taktów niezbędną do obliczenia całego algorytmu redukcji wstecznej, w związku z czym każdy kolejny wariant okazywał się wydajniejszy. Jak wspomniano wcześniej, dalsze wstawianie dodatkowych rejestrów było bezcelowe, gdyż największe opóźnienia powodowały połączenie wykorzystujące wbudowane bloki DSP o stałej architekturze. Przeprowadzono również badania mające na celu określenie parametrów struktury akceleratora z wykorzystaniem większej liczby elementów przetwarzających (h>1)(rys.19). Badania te wykazały liniowy wzrost złożoności sprzętowej wraz ze wzrostem liczby h elementów EP. Zależność ta okazała się liniowa ze względu na jednakową budowę wszystkich elementów przetwarzających. Zaobserwowano również praktycznie brak zależności maksymalnej częstotliwości działania jednostki od liczby elementów przetwarzających h. Ewentualny wpływ na częstotliwość pracy jednostki wieloprocesorowej może mieć organizacja przechowywania macierzy współczynników w blokach pamięci BRAM.

Z powodu braku dostępnych kompletnych projektów jednostek przetwarzających realizujących algorytm redukcji wstecznej z wykorzystaniem innej reprezentacji danych, wykonane zostało porównanie parametrów opracowanej jednostki RFA z parametrami znanych bloków operacyjnych realizujących cząstkowe operacje arytmetyczne wchodzące w skład tego algorytmu. W porównywanych blokach operacyjnych wykorzystano reprezentację zmiennoprzecinkową pojedynczej precyzji (*float point 32*) oraz 64-bitową stałoprzecinkową (*fixed point 64*), gdyż te formaty liczb zapewniają dokładność obliczeń zbliżoną do dokładności 35-bitowych liczb ułamkowych [9, 11]. Zestawienie parametrów

zaprojektowanego układu w wariancie 3 z długością kolejki FIFO równą 128 komórek z parametrami wymienionych bloków operacyjnych przedstawiono w tabeli 24.

Tabela 24. Parametry implementacji bloków operacyjnych stałoprzecinkowych, zmiennoprzecinkowych oraz całej jednostki RFA (zgodnie z wariantem 3 tab.23) zapewniających podobną dokładność obliczeń (implementacja w układzie FPGA Xilinx xc4vSX35-12)

Blok operacyjny	**Format danych**	**Liczba Slices + DSP**	**Częstotliwość maks. MHz**	**Liczba stopni potoku**
Blok dzielenia Xilinx Divider 1.0	Fixed point 32 **(wart. maksym.)**	1229 + 0	254	36 (szerokość danych +4)
Blok odejmowania Adder Subtracter 7.0	Fixed point 64	215 + 0	393	6
Blok dzielenia Xilinx Divider 1.0	Float point 32	203+ 11	110	5
Blok odejmowania	Float point 32	195+4	209	6
Jednoprocesorowa jednostka przetwarzająca RFA	**RFA 35**	**1887+20**	**147**	**5 – mnoż/dziel. 6 – odejmowania**

Analiza danych przedstawionych w tabeli 24 świadczy o tym, że wielkość bloku dzielenia w arytmetyce stałoprzecinkowej przy 32 bitowej reprezentacji liczb mierzona w liczbie komórek CLB jest niewiele mniejsza od struktury całego zaprojektowanego układu do realizacji algorytmu redukcji wstecznej zrealizowanego w arytmetyce ułamkowej RFA. Należy podkreślić dużą liczbę stopni w potoku dla bloku dzielenia w arytmetyce stałoprzecinkowej co z pewnością miałoby znaczący wpływ na liczbę taktów niezbędną do realizacji całego algorytmu redukcji wstecznej. Warto podkreślić również, że reprezentacja 32-bitowa dla arytmetyki stałoprzecinkowej była wartością maksymalną dla generatora IP-core Xilinx Divider 1.0, a podobną dokładność obliczeń do 35-bitowej arytmetyki ułamkowej zapewnia dopiero 64-bitowa arytmetyka stałoprzecinkowa. Jedynie liczba wykorzystywanych bloków DSP jest większa, jednak w nowoczesnych układach FPGA wraz z pojawieniem się kolejnych nowszych rodzin liczba tych bloków ciągle wzrasta. Przykładowo w rodzinie SX Xilinx Virtex 4 mamy do 512 wbudowanych bloków DSP, natomiast w rodzinie SX Xilinx Virtex 6 wartość ta wzrasta do 2016[22]. Natomiast w przypadku reprezentacji 32-bitowej zmiennoprzecinkowej można zauważyć, że blok dzielenia posiada mniejszą częstotliwość pracy od całego układu RFA realizującego algorytm redukcji wstecznej zawierającego również blok dzielenia o identycznej liczbie stopni w potoku. Otrzymane wyniki badań pozwalają stwierdzić, że zaprojektowana jednostka RFA realizująca algorytm redukcji wstecznej charakteryzuje się zdecydowanie mniejszą złożonością sprzętową w porównaniu do analogicznej jednostki działającej w arytmetykach stało lub zmiennoprzecinkowej, zapewniającej zbliżoną dokładność obliczeń. Ponadto zaprojektowana

[22] http://www.xilinx.com/support/documentation/data_sheets/ds150.pdf

jednostka RFA cechuje się większą maksymalną częstotliwością pracy przy zbliżonej liczbie stopni w potoku, co oznacza jej większą wydajność.

3.3.2. Potokowy i równoległy akcelerator realizujący algorytm rozkładu macierzy pasmowej LU metodą eliminacji Gaussa

Kolejnym projektem z dużym wkładem autorskim jest projekt równoległej i potokowej jednostki akceleratora wykorzystującego arytmetykę ułamkowa, przeznaczonego do realizacji algorytmu rozkładu LU macierzy trójdiagonalnej Jacobi'ego A(*N,N*)[74]. Algorytm ten realizowany był w oparciu o wzory (15), które reprezentują najprostszą wersję algorytmu eliminacji Gaussa.

$$u_1 = b_1$$

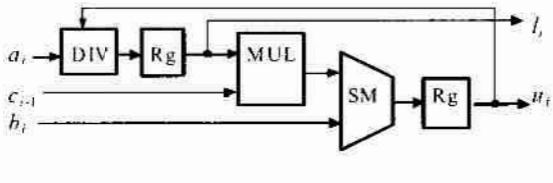
$$l_i = a_i / u_{i-1} \qquad i=2,3,...,N \qquad (16)$$

$$u_i = b_i - l_i * c_{i-1} \qquad i=2,3,...,N$$

Na rysunku 24 przedstawiono rozkład LU macierzy Jacobi'ego za pomocą równości w postaci macierzowej, natomiast wstępny projekt architektury jednostki przetwarzającej wraz z ilustracją jego działania przedstawiono na rys. 25, na którym MUL, DIV, SM i Rg oznaczają odpowiednio bloki: mnożenia, dzielenia, sumatora i rejestru.

$$\begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & & \\ & a_3 & b_3 & c_3 & & & \\ & & a_4 & b_4 & c_4 & & \\ & & & \cdot & \cdot & \cdot & \\ & & & & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & & a_N & b_N \end{bmatrix} = \begin{bmatrix} 1 & & & & & & \\ l_2 & 1 & & & & & \\ & l_3 & 1 & & & & \\ & & l_4 & 1 & & & \\ & & & \cdots & \cdots & & \\ & & & & l_N & 1 \end{bmatrix} * \begin{bmatrix} u_1 & c_1 & & & & & \\ & u_2 & c_2 & & & & \\ & & u_3 & c_3 & & & \\ & & & \cdots & \cdots & & \\ & & & & u_{N-1} & c_{N-1} \\ & & & & & u_N \end{bmatrix}$$

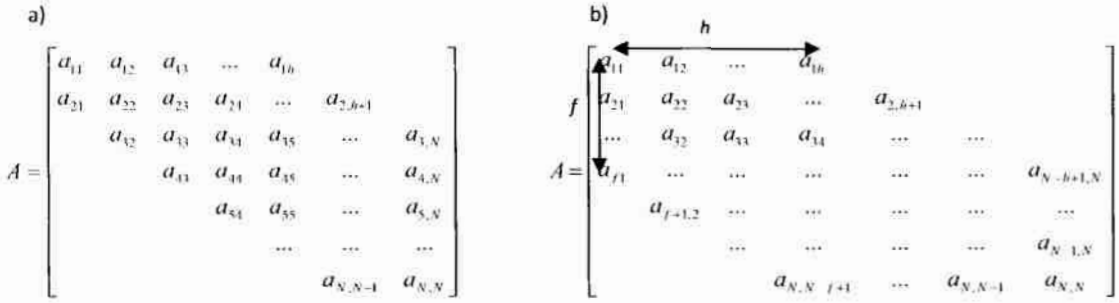
Rysunek 24. Rozkład Lu macierzy Jacobi'ego przedstawiony za pomocą równania w postaci macierzowej



TAKTY											
...	11	10	9	8	7	6	5	4	3	2	1
...		a_6		a_5		a_4		a_3		a_2	
...	c_5		c_4		c_3		c_2		c_1		$c_0=0$
...	b_6		b_5		b_4		b_3		b_2		b_1
WYNIKI											
...	u_6	l_6	u_5	l_5	u_4	l_4	u_3	l_3	u_2	l_2	u_1

Rysunek 25. Architektura jednostki przeznaczonej do realizacji eliminacji Gaussa na macierzy Jacobi'ego wraz z ilustracją kolejności przeprowadzenia obliczeń(źródło[74]).

W kolejnym etapie projektowania do algorytmu eliminacji Gaussa została wprowadzona strategia lokalnego wyboru elementu wiodącego, aby jednostka przetwarzająca mogła wykonywać rozkład LU macierzy pasmowej Hessenberga lub nawet dowolnej macierzy pasmowej. Na rysunku 26a przedstawiono graficzną ogólne postać macierzy Hessenberga oraz na rys. 26b przypadek bardziej ogólny – macierz pasmową.



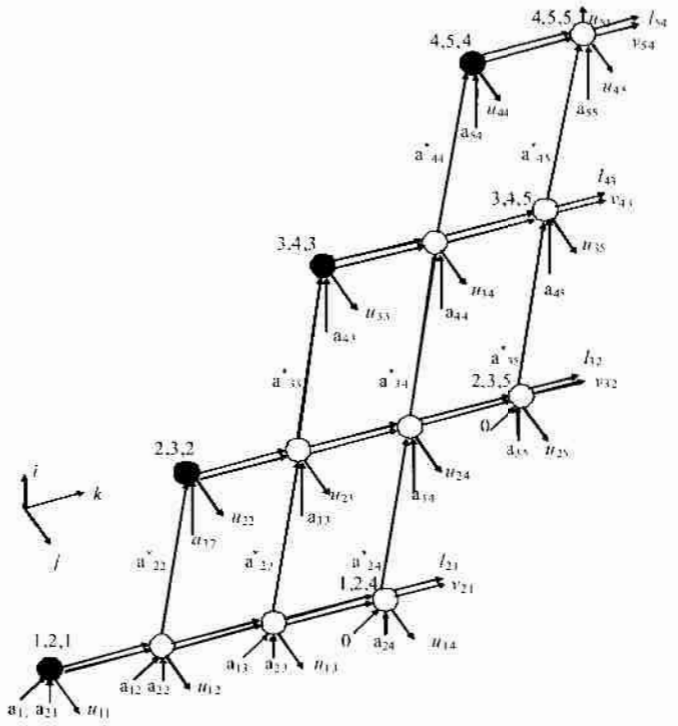
Rysunek 26. Postacie ogólne macierzy pasmowej Hessenberga (a) oraz macierzy pasmowej (b)

Dla macierzy pasmowej Hessenberga (f=2), której szerokość pasma wynosi (h+1), metoda eliminacji Gaussa z lokalnym wyborem elementu wiodącego może być przedstawiona za pomocą poniższego fragmentu programu (2).

```
for i = 1 to N-1 do
  begin
    if abs(a(i, i)) < abs(a(i+1,i))
       then /*zamiana wierszy*/
          for k = i to h+1 do
            begin
              u(i, k)=a(i+1, k); a(i+1,k)=a(i, k);
            end
       else /*brak zamiany wierszy*/
          for k = i to h+1 do u(i, k)=a(i, k);
    /*eliminacja*/
    l(i+1, i)= a(i+1,i)/u(i, i);
    for k = i+1 to h+1 do a(j, k) = a(j, k) - l(j, i)*u(i, k);
  end;
```

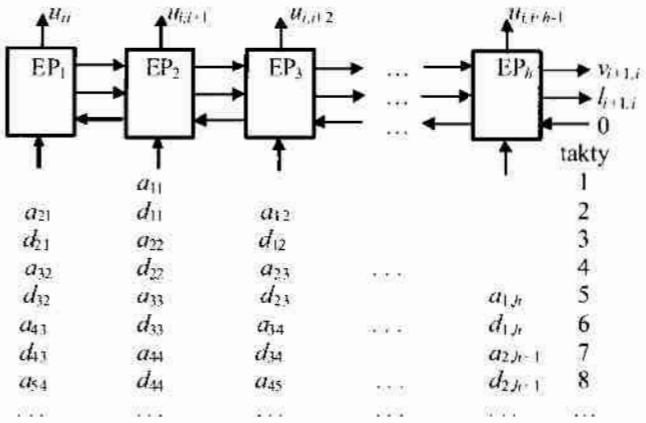
(2)

Na rys. 27 przedstawiono graf zależności informacyjnej algorytmu opisanego fragmentem programu (2), na którym czarne wierzchołki reprezentują operacje sprawdzenia warunku $|a(i,i)|<|a(i+1,i)|$, zamianę elementów $a(i, i)$ z $a(i+1,i)$ jeśli warunek jest spełniony oraz operację dzielenia. Pozostałe wierzchołki grafu reprezentują operacje zamiany elementów $a(i, k)$ z $a(i+1,k)$ jeśli warunek jest spełniony oraz operację mnożenia z dodawaniem. Zaznaczona na grafie zmienna binarna v_{ji} jest znacznikiem warunku $|a(i,i)|<|a(i+1,i)|$, jej wartość jest równa 1, gdy warunek ten jest spełniony.



Rysunek 27. Graf algorytmu eliminacji Gaussa z lokalnym wyborem elementu wiodącego dla macierzy pasmowej Hessenberga A(5,5), h=4

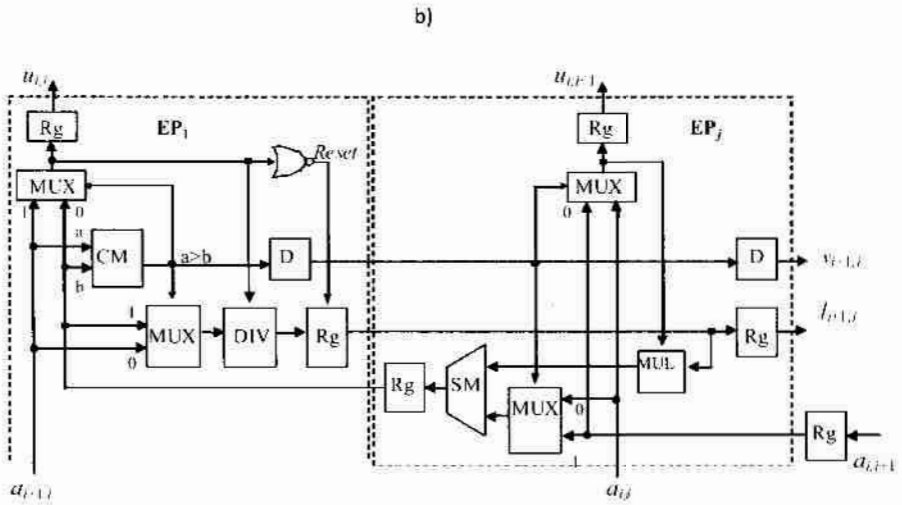
Zgodnie z metodą odwzorowania przedstawioną w pracy [1], na podstawie grafu przedstawionego na rys.27 została otrzymana struktura równoległej jednostki przetwarzającej S_2, przedstawiona na rys.28.



Rysunek 28. Struktura jednostki S_2 przeznaczonej do realizacji eliminacji Gaussa na macierzy pasmowej Hessenberga (źródło[74]).

Zaprojektowana jednostka zawiera h elementów przetwarzających EP o dwóch różnych typach. Element pierwszego typu EP_1 realizuje operacje odpowiadające czarnym wierzchołkom grafu, natomiast elementy drugiego typu EP_j (j=2,...,h) realizują operacje odpowiadające pozostałym wierzchołkom grafu algorytmu. Na rys.29a przedstawiono

uogólnioną strukturę wewnętrzną elementów przetwarzających typu EP_1, a na rys.29b elementów przetwarzających EP_j, gdzie symbol CM oznacza komparator, D – przerzutnik typu D, a MUX – multiplekser dwuwejściowy.



Rysunek 29. Architektura wewnętrzna EP_1 (a) oraz EP_j (b) równoległej jednostki przetwarzającej S_2

Czas realizacji metody eliminacji Gaussa w strukturze S_2 równy jest w przybliżeniu $2N$ taktom, co odpowiada około 50% stopniowi obciążenia poszczególnych elementów przetwarzających równoległej jednostki. Jednak analiza architektury wykazała, że jednostka ta może równocześnie przetwarzać dwie macierze danych wejściowych, np. macierzy A i D o tych samych rozmiarach N i h, przy zachowaniu tego samego czasu wykonania rozkładu LU. W tym przypadku stopień obciążenia elementów przetwarzających jednostki S_2 bliski jest wartości 100%. Sposób podawania elementów macierzy A i D na wejścia jednostki, w przypadku ich równoległego przetwarzania, przedstawiono w dolnej części rys. 28.

W celu realizacji architektury S_2 w arytmetyce ułamkowej opracowano nowy blok operacyjny wykonujący porównanie wartości bezwzględnych dwóch ułamków $|x|>|y|$. W algorytmie rozkładu LU macierzy pasmowej operacja ta jest niezbędna do wyboru elementu wiodącego i polega ona na porównaniu wartości bezwzględnych dwóch sąsiednich elementów tej samej kolumny macierzy $|a_{ii}|$ i $|a_{i+1,i}|$. Następnie realizowane jest dzielenie mniejszego elementu przez większy. W zależności od znaków liczników x_n, y_n i mianowników x_d, y_d , sprawdzenie warunku określonego wzorem (17)

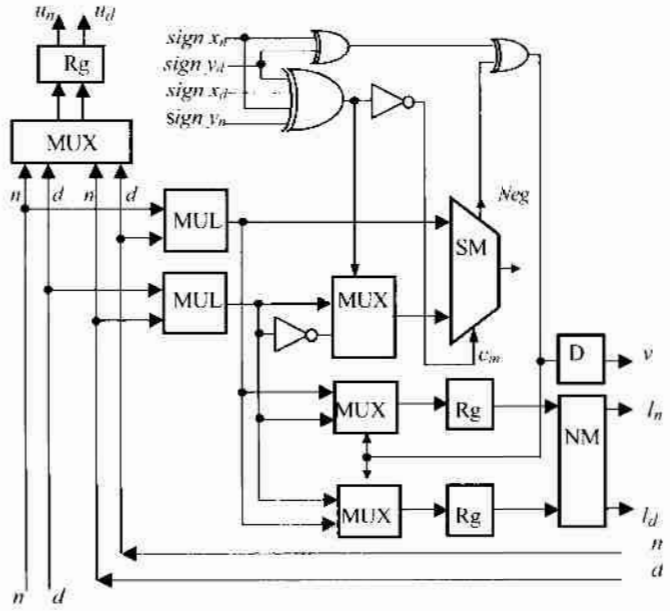
$$\left|\frac{x_n}{x_d}\right| > \left|\frac{y_n}{y_d}\right| \qquad (17)$$

sprowadza się do obliczenia znaku (funkcja *sign* lub jej negacji) wyrażenia przedstawionego w tab. 25 w kolumnie „Operacja", tj. sprawdzenia znaku sumy lub różnicy iloczynów x_n*y_d oraz x_d*y_n . Więc podstawową operacją w procedurze porównania ułamków jest mnożenie z dodawaniem lub odejmowaniem.

Tabela 25. Wzory pozwalające wykonać porównanie wartości bezwzględnych dwóch ułamków

$sign(x_n)$	$sign(x_d)$	$sign(y_n)$	$sign(y_d)$	Operacja
0	0	0	0	$sign(x_n \cdot y_d - y_n \cdot x_d)$
0	0	0	1	$\overline{sign(x_n \cdot y_d + y_n \cdot x_d)}$
0	0	1	0	$sign(x_n \cdot y_d + y_n \cdot x_d)$
0	0	1	1	$\overline{sign(x_n \cdot y_d - y_n \cdot x_d)}$
0	1	0	0	$sign(x_n \cdot y_d + y_n \cdot x_d)$
0	1	0	1	$\overline{sign(x_n \cdot y_d - y_n \cdot x_d)}$
0	1	1	0	$sign(x_n \cdot y_d - y_n \cdot x_d)$
0	1	1	1	$\overline{sign(x_n \cdot y_d + y_n \cdot x_d)}$
1	0	0	0	$\overline{sign(x_n \cdot y_d + y_n \cdot x_d)}$
1	0	0	1	$sign(x_n \cdot y_d - y_n \cdot x_d)$
1	0	1	0	$\overline{sign(x_n \cdot y_d - y_n \cdot x_d)}$
1	0	1	1	$sign(x_n \cdot y_d + y_n \cdot x_d)$
1	1	0	0	$\overline{sign(x_n \cdot y_d - y_n \cdot x_d)}$
1	1	0	1	$sign(x_n \cdot y_d + y_n \cdot x_d)$
1	1	1	0	$\overline{sign(x_n \cdot y_d + y_n \cdot x_d)}$
1	1	1	1	$sign(x_n \cdot y_d - y_n \cdot x_d)$

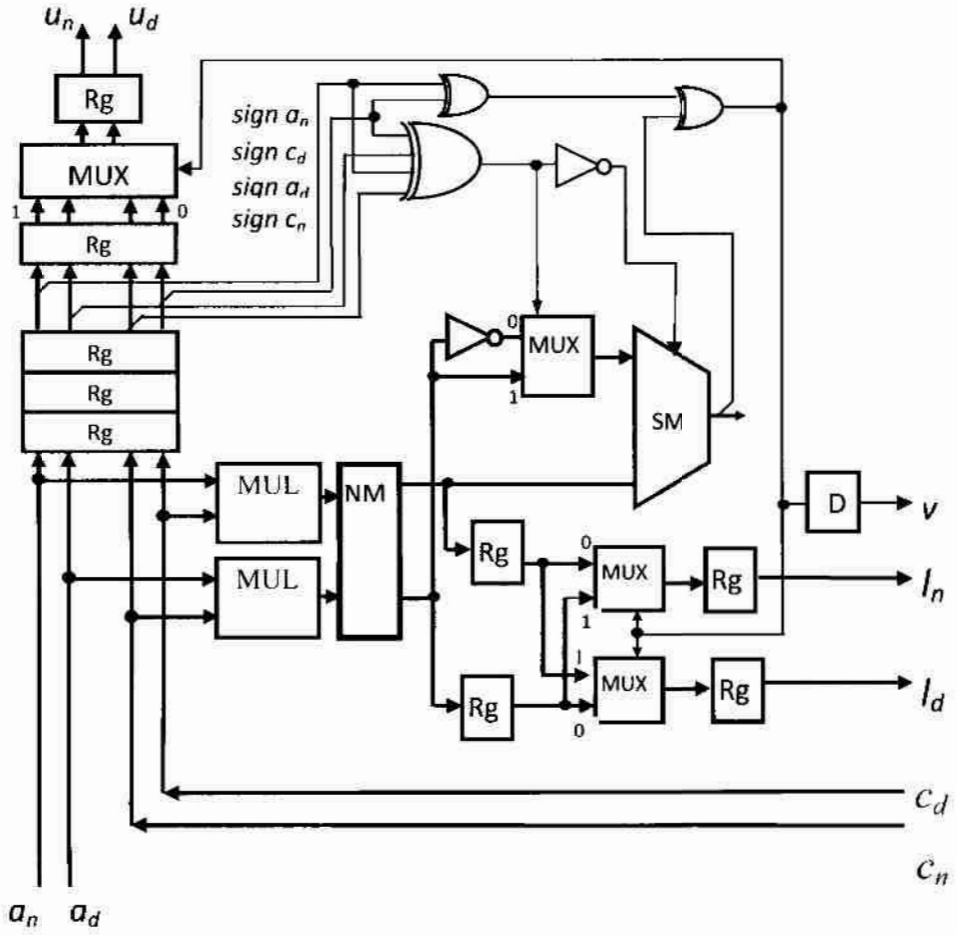
Akcelerator, podobnie jak wszystkie zaprojektowane bloki operacyjne RFA, wykorzystuje reprezentację danych w kodzie z uzupełnieniem do dwóch U2, zatem operacje odejmowania jest realizowana na sumatorze z uprzednim zanegowaniem bitów drugiego składnika oraz podaniem wartości „1" na wejście C_{IN} sumatora. Znak obliczonej sumy lub różnicy określany jest zgodnie z reprezentacją kodu U2 na podstawie wartości najstarszego bitu wyniku. Opracowano oryginalną strukturę bloku operacyjnego realizującego operację porównania wartości bezwzględnej liczb ułamkowych wykorzystując obecny w EP_1 blok dzielenia. Zgodnie ze wzorem (12) przedstawionym w podrozdziale 3.1 operacja dzielenia *x/y* w arytmetyce ułamkowej sprowadza się do obliczenia wartości licznika x_n*y_d i mianownika x_d*y_n wyniku. Blok operacji porównania wartości bezwzględnych został zatem zrealizowany przez wprowadzenie tylko jednego dodatkowego sumatora, dwóch dwu-wejściowych multiplekserów i bloku normalizacji NM. Struktura wewnętrzna EP_1 działającego w arytmetyce ułamkowej RFA przedstawiona została na rys.30 , gdzie *n* i *d* oznaczają odpowiednio licznik i mianownik odpowiedniej zmiennej, *Neg* oznacza znacznik ujemnego wyniku, a c_{in} – przeniesienie wejściowe sumatora SM, natomiast pozostałe bloki funkcjonalne zostały opisane w następujący sposób: MUL – blok mnożący, MUX – multiplekser, Rg – rejestr, D- przerzutnik wyzwalany narastającym zboczem.



Rysunek 30. Struktura wewnętrzna EP_1 jednostki przetwarzającej S_2 działającej w arytmetyce ułamkowej

W przedstawionej na rys.30 jednostce każdy blok mnożenia MUL może mieć 2 do 4 stopni w potoku, natomiast blok normalizacji wyniku posiada 2 stopnie w potoku. Dla bloków mnożenia z 2 stopniami w potoku czas realizacji każdego wierzchołka grafu algorytmu wynosi 5 taktów zegarowych, ale równocześnie w każdym EP jednostki mogą być przetwarzane elementy pięciu różnych macierzy wejściowych A. W takim przypadku czas wykonania metody eliminacji Gaussa z lokalnym wyborem elementu wiodącego mógłby wynosić około $2N$ taktów, co również odpowiada wysokiemu około 67% stopniowi obciążenia poszczególnych EP równoległej jednostki.

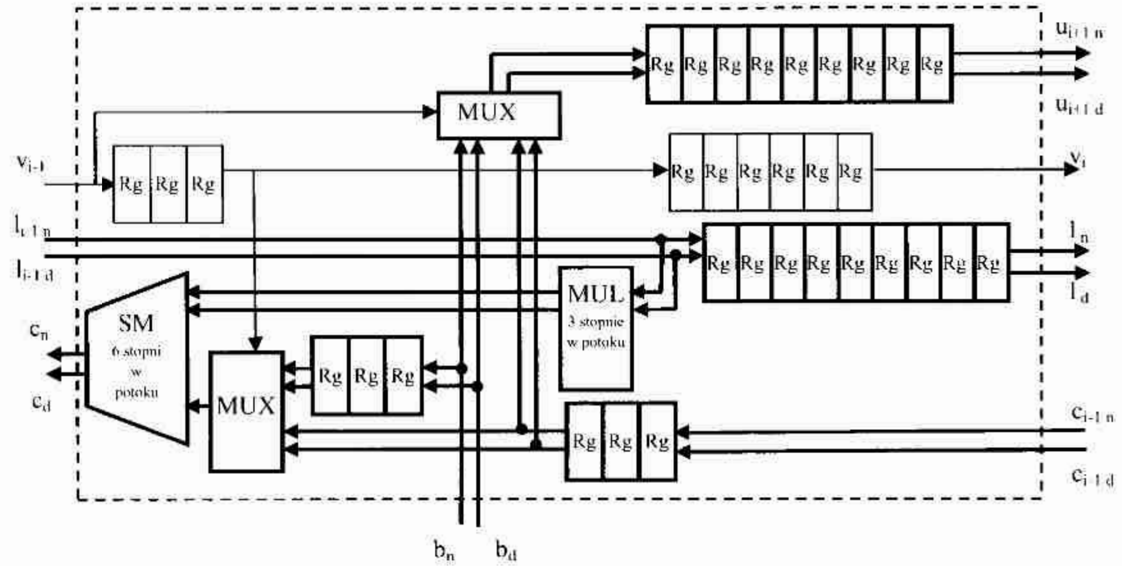
Wkład autorski przy projektowaniu akceleratora polegał na wspólnym opracowaniu ogólnej architektury elementów przetwarzających oraz na samodzielnym zamodelowaniu, weryfikacji oraz określeniu parametrów zaprojektowanych jednostek przetwarzających oraz całego akceleratora. Schemat autorskiej zamodelowanej i zaimplementowanej jednostki przedstawiono na rys.31. Na rysunku tym zamieszczono dodatkowe rejestry opóźniające oraz pogrubionymi liniami zaznaczono 35-bitowe magistrale danych.



Rysunek 31. Szczegółowa struktura wewnętrzna EP_1 jednostki przetwarzającej S_2, zaimplementowanej w układzie FPGA.

Projekt opracowanej potokowej jednostki przetwarzającej również został zaimplementowany w układzie FPGA Virtex 4 (4vlx15sf363-12) w środowisku Xilinx ISE 9.2i, natomiast weryfikacja poprawności działania została przeprowadzona w symulatorze ActiveHDL 7.2 firmy Aldec. Jednostka EP_1 została zaimplementowana dla 35-bitowych danych wejściowych. Blok normalizatora NM został przesunięty na wyjście bloku dzielenia RFA, wykorzystującego 2 bloki mnożące MUL, natomiast liczba stopni w potoku bloku dzielenia razem z blokiem normalizacji została ustalona na wartość 3. W związku z tym cała jednostka EP_1 pracowała w 5-stopniowym trybie potokowym. Po przeprowadzeniu procesu syntezy i implementacji maksymalna częstotliwość działania jednostki przetwarzającej wyniosła około 149,5MHz. Próbowano zwiększyć maksymalną częstotliwość pracy przez zwiększenie liczby stopni w potoku bloku dzielenia(mnożenia) co miało również wpływ na liczbę stopni w potoku całej jednostki. Przy zwiększeniu stopni w potoku o wartość 2 zaobserwowano wzrost maksymalnej częstotliwości tylko do wartości 151,7 MHz. Dalsze zwiększenie liczby stopni potoku nie wpływało na wzrost tej wartości. Ponadto wyniki implementacji jednostki świadczą o tym, że dla 35-bitowych danych wejściowych EP_1 wykorzystuje 8 bloków DSP oraz 433 bloki *Slice* układu FPGA, które stanowią około 7% liczby tych bloków, dla zastosowanego stosunkowo małego układu Virtex4 VLX15.

Podobne autorskie badania przeprowadzono dla elementu przetwarzającego drugiego typu EP_j, przedstawionego w postaci ogólnej na rys.29. Na rys.32 przedstawiono szczegółową strukturę wewnętrzną zaprojektowanego elementu przetwarzającego.



Rysunek 32. Zaimplementowana struktura wewnętrzna elementu przetwarzającego EP_j jednostki przetwarzającej S_2(rys.29)

Element przetwarzający EP_j zbudowany jest z jednego bloku mnożącego oraz jednego bloku sumowania(odejmowania) RFA. Oba bloki arytmetyczne zawierają w swojej strukturze bloki normalizacji wyników. Ponadto wykorzystano 2 multipleksery oraz dodatkowe stopnie rejestrów wykorzystywane do zsynchronizowania pracy całego elementu przetwarzającego. W celu uzyskania zadowalającej maksymalnej częstotliwości pracy całego elementu przetwarzającego ostatecznie stopnie w potoku dla bloku mnożenia i odejmowania ustalono na wartościach odpowiednio 3 i 6. Podobnie jak we wcześniejszych projektach weryfikację funkcjonalną przeprowadzono z wykorzystaniem symulatora ActiveHDL 7.2., natomiast proces syntezy i implementacji z wykorzystaniem środowiska ISE 9.2. Maksymalna częstotliwość pracy zaprojektowanego EP_j, została określona jako 140,h MHz; do jego implementacji wykorzystano 1089 bloków *Slice* i 20 bloków DSP układu reprogramowalnego.

Przeprowadzono również autorskie badania syntezy i implementacji całej równoległej i potokowej architektury S_2 (rys. 29) realizującej metodę eliminacja Gaussa dla macierzy pasmowej Hessenberga (rys. 26a), zawierającej elementy przetwarzające zarówno pierwszego i drugiego typu. W celu wyrównania liczby taktów niezbędnych do realizacji operacji w elemencie przetwarzającym pierwszego typu EP_1 oraz elementach typu drugiego EP_j, pomiędzy pierwszym i drugim elementem przetwarzającym wprowadzono dodatkowe stopnie rejestru. W tabeli 26 przedstawiono wyniki uzyskane po procesie syntezy

i implementacji określające wielkość struktury i maksymalną częstotliwość pracy dla architektury S_2 zawierającej w sumie odpowiednio 4, 6 i 8 elementów przetwarzających.

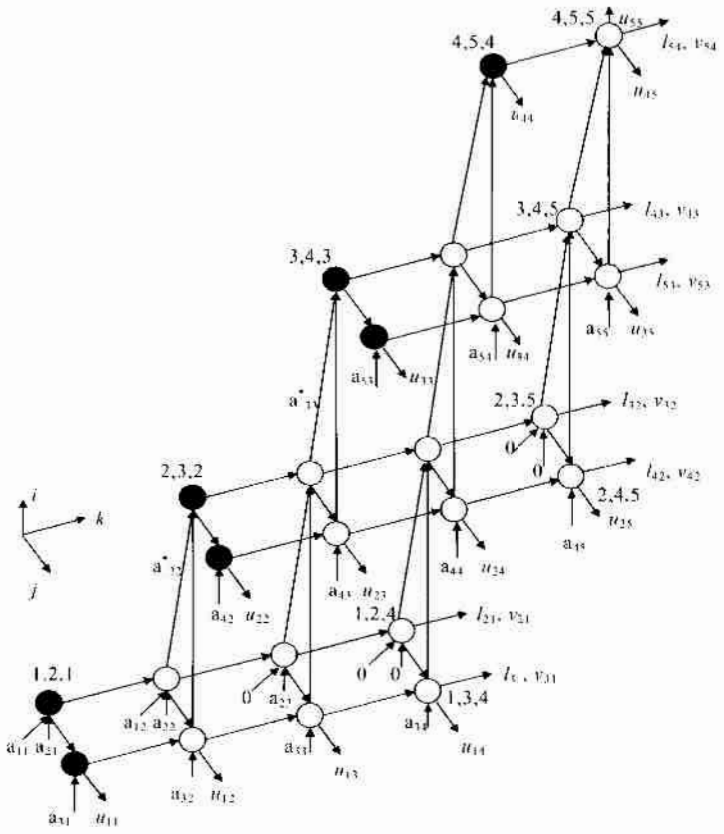
Tabela 26. Parametry implementacji struktury S_2 zawierającej 4, 6 i 8 elementów przetwarzających EP.

	4 EP (1xEP1, 3xEP2)	6 EP (1xEP1, 5xEP2)	8 EP (1xEP1, 7xEP2)
Maksymalna częstotliwość pracy	101,4 MHz	101,4 MHz	101,4 MHz
Liczba wykorzystanych bloków Slice	3630	5652	7813
Liczba wykorzystanych bloków DSP	68	108	148

Proces implementacji przeprowadzono dla stosunkowo niewielkiego układu FPGA, mianowicie Xilinx Virtex4SX35, zawierającego 192 wbudowane bloki DSP oraz 15360 bloków Slice. Do implementacji całej struktury S_2 w tym układzie wykorzystano odpowiednio 23, 36 i 50% dostępnych bloków Slice. Warto podkreślić, że największe układy rodziny Virtex6 zawierają nawet do 118tys bloków *Slice* i do ponad 2tys. bloków DSP, natomiast najnowsze rodziny Virtex7[23] odpowiedni ponad 300tys bloków *Slice* i około 4 tys. bloków DSP. W takich układach możliwa zatem jest implementacja struktury S_2 realizującej metodę eliminacji Gaussa dla macierzy Hessenberga o szerokości pasma macierzy ponad 100 elementów.

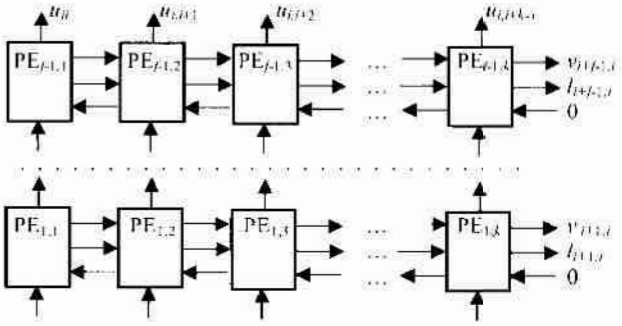
W trakcie opracowania projektu architektury S_2 stwierdzono, że może być ona wykorzystana do budowy akceleratora przeznaczonego do realizacji eliminacja Gaussa w przypadku macierzy pasmowych postaci ogólnej (rys. 26b), z liczbą niezerowych elementów w pierwszej kolumnie *f*, a pierwszym wierszu *h*. Na rysunku 33 przedstawiono przykładowy graf zależności informacyjnych algorytmu eliminacji Gaussa dla macierzy pasmowej.

[23] http://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf



Rysunek 33. Przykładowy graf zależności informacyjnej algorytmu rozkładu LU macierzy pasmowej A(*N,N*) z lokalnym wyborem elementu wiodącego dla przypadku *N*=5, *f*=3, *h*=2, *k*=*f*+*h*-1=4.

W celu realizacji algorytmu dla wyżej opisanej macierzy zaprojektowana struktura S_2 przekształcona zostaje w strukturę dwuwymiarową S_3 poprzez „kaskadowe" połączenie (*f*-1) struktur S_2. Ogólną strukturę jednostki S_3 przedstawiono na rys.34.



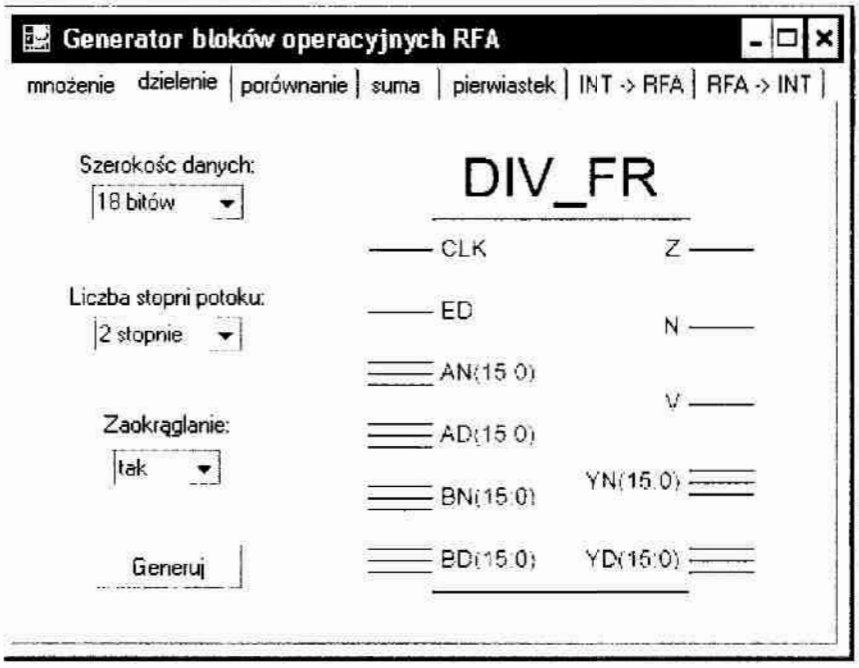
Rysunek 34. Struktura jednostki S_3 przeznaczonej do realizacji eliminacji Gaussa na macierzy pasmowej

W architekturze S_3 zarówno struktury wewnętrzne EP_1 i EP_j, jak i współczynnik wykorzystania EP, pozostają bez zmian. Wszystkie EP znajdujące się pierwszej kolumnie architektury mają strukturę jednakową z EP_1, a wszystkie pozostałe EP maja strukturę EP_j. Zarówno maksymalna częstotliwość działania architektury S_3, jak i zasoby układu FPGA wykorzystywane przez poszczególne EP pozostają bez zmian.

4. Opracowanie podstawowych modułów środowiska programistycznego wspomagającego projektowanie akceleratorów do równoległej realizacji algorytmów algebry liniowej.

4.1. Generator bloków operacyjnych pracujących w arytmetyce ułamkowej.

W ramach prac badawczych nad zastosowaniem arytmetyki ułamkowej w reprogramowalnych jednostkach przetwarzających systemów jednoukładowych[97] opracowano generator IP-core bloków realizujących podstawowe operacje arytmetyczne z wykorzystanie arytmetyki ułamkowej. Szczegółowy opis zaprojektowanego generatora przedstawiono w pracy[26]. Komponenty uzyskane z wykorzystaniem generatora IP-core mogą być przedstawione w różnych formatach[1]: Hard-core – poziom topografii układu, Firm-core – lista połączeń standardowych komórek układu reprogramowalnego(ang. net list level) lub Soft-core – w postaci opisu w języku HDL. Format Soft-core jest formatem najbardziej funkcjonalnym, gdyż umożliwia ingerencję strukturę generowanego bloku oraz daje możliwość implementacji w różnych rodzinach układów reprogramowalnych. Z wymienionych powodów oraz ze względu na możliwość odtworzenia metod realizacji określonych funkcji bloku, często chronionych prawami autorskimi, format ten nie jest wykorzystywany przez firmy komercyjne(Xilinx, Altera). Z uwagi na duża uniwersalność zaprojektowany generator umożliwia tworzenie bloków funkcjonalnych na poziomie Soft-core. Generator został napisany w języku C# z wykorzystaniem środowiska Visual Studio 2008 oraz platformy Microsoft . NET Framework 3.5, która jest niezbędna do jego uruchomienia. Interfejs graficzny zaprojektowanego generatora bloków operacyjnych RFA przedstawiono na rys. 1.



Rysunek 31. Interfejs graficzny zaprojektowanego generatora bloków operacyjnych.

Generator tworzy opis w języku VHDL wybranego przez użytkownika bloku operacyjnego realizującego jedną z operacji: mnożenia, dzielenia, dodawania, porównania lub pierwiastkowania. Ponadto funkcjonalność generatora poszerzono o możliwość generowania opisów VHDL bloków konwersji liczb z formatu stałoprzecinkowego do formatu ułamkowego i odwrotnie, gdyż bloki te mogą okazać się niezbędne w przypadku syntezy układu zawierającego bloki funkcjonalne wykorzystujące inną reprezentację danych. Bloki konwersji mogą być również wykorzystywane do wprowadzania i wyprowadzania danych z układu RFA w postaci stałoprzecinkowej.

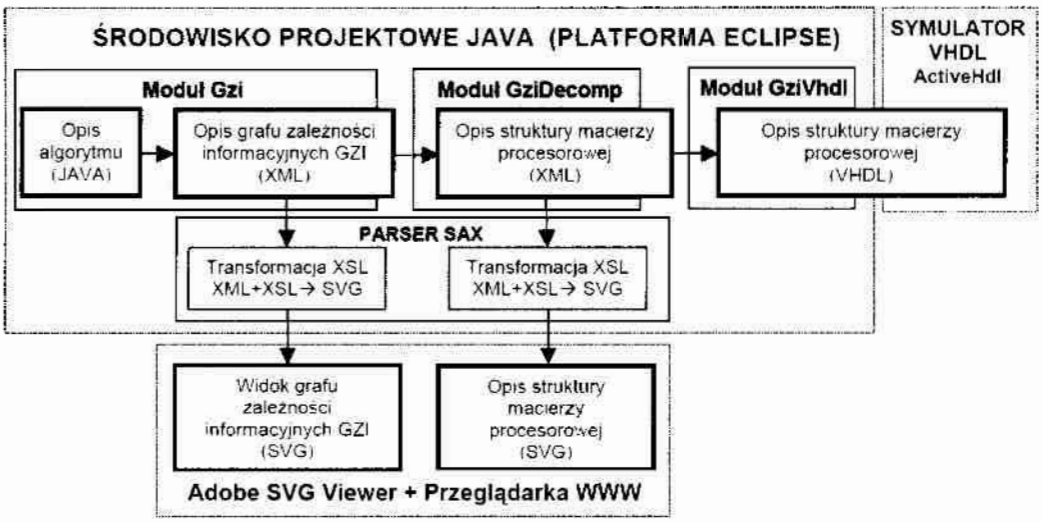
Wszystkie modele VHDL bloków operacyjnych mogą być wygenerowane dla standardowych wartości szerokości danych wejściowych i wyjściowych: 24 i 32 bitów. Dodatkowo, po uwzględnieniu możliwości wbudowanych bloków mnożących i bloków DSP w układach FPGA firmy Xilinx, wprowadzono szerokości danych 18 i 35 bitów. Umożliwiono również wybór liczby stopni potoku dla projektowanego bloku operacyjnego, co ma duży wpływ na maksymalną częstotliwość pracy oraz wydajność bloku. Liczba stopni w potoku projektowanych bloków została sparametryzowana, gdyż często ma ona zasadnicze znaczenie w przypadku synchronizacji obliczeń lub w przetwarzaniu danych w projektowanych architekturach układów, zwłaszcza równoległych. Generator umożliwia również włączenie operacji zaokrąglania wyników cząstkowych po wykonaniu operacji normalizacji danych, która jest wykonywana praktycznie po każdej podstawowej operacji arytmetycznej wykorzystującej arytmetykę ułamkową. Badania generowanych bloków operacyjnych nie wykazały jednak istotnego wpływu opcji zaokrąglania na wyniki realizowanych operacji arytmetycznych. Parametry wygenerowanych bloków operacyjnych określone w procesie syntezy i implementacji w środowiskach Xilinx ISE 9.2 oraz Altera Quartus II 7.2 przedstawiono w podrozdziale 3.2 niniejszej rozprawy.

Generowane bloki funkcjonalne posiadają wejście zegarowe oraz wejście ED (ang. enabled) określające włączenie bloku. Poza magistralami danych wejściowych i wyjściowych wygenerowane bloki posiadają flagi określające wynik realizowanej operacji: Z – wynik zerowy, N – wynik ujemny oraz V – oznaczające przepełnienie wyniku. Generowane bloki nie poosiadają wejścia zerującego (ang. reset), gdyż uniemożliwiało to wykorzystanie wbudowanych bloków DSP48 do implementacji operacji mnożenia oraz SRL16 do implementacji kolejek FIFO. Wykorzystanie tych bloków ma zasadniczy wpływ na wydajność projektowanych bloków oraz liczbę wykorzystywanych komórek CLB do ich implementacji.

Zaprojektowany generator IP-core może być wykorzystany przy projektowaniu jednostek przetwarzających RFA przeznaczonych do implementacji w układach FPGA. Wykorzystanie generatora pozwala skrócić czas projektowania jednostek operacyjnych i zwiększyć jego jakość(bezbłędność). Generator pozwala również zwiększyć jakość projektowanych systemów poprzez zmniejszenie złożoności sprzętowej lub zwiększenie maksymalnej częstotliwości pracy jednostek przetwarzających.

4.2. Środowisko JGEN wspomagające projektowanie architektur równoległych akceleratorów przeznaczonych do realizacji wybranych algorytmów algebry liniowej.

Wraz z rozwojem nowych metod odwzorowania algorytmów w architektury macierzy procesorowych opracowywano nowe środowiska CAD wspomagające ich projektowanie [54,78,79,81]. Rozwijane w ciągu ostatnich kilku lat, autorskie metody projektowe również zostały zaimplementowane w analogiczne środowisko o nazwie JGEN[24]. Jako jeden z pierwszych modułów tego środowiska powstał moduł do generowanie grafów zależności informacyjnych(moduł Gzi z rys.32) dla opisywanych w języku JAVA programów regularnych, zawierających zagnieżdżone pętle programowe(ang. *nested loops*), w tym wybranych algorytmów algebry liniowej. Wygenerowany graf zależności informacyjnych, zawierający dane o wszystkich węzłach i łukach grafu zapisywany jest do pliku w formacie XML. Tak wygenerowany graf jest wykorzystywany jako dane wejściowe dla kolejnych modułów, np. dla modułu liniowego odwzorowania przestrzennego w architektury macierzy procesorowych z wykorzystaniem rachunku macierzowego zgodnie z metodą[1,49,50,76] (moduł GziDecomp z rys.32). Moduł ten generuje kilka lub kilkanaście architektur macierzy procesorowych, uzyskanych dla podstawowych projekcji grafu zależności informacyjnych na wybrane płaszczyzny lub osie przestrzeni całkowitoliczbowej[1] oraz umożliwia również ich wizualizację. Pierwsze moduły projektu JGEN przedstawione na rys.32 zostały dokładniej opisane w pracy[107].



Rysunek 32. Pierwsze moduły środowiska JGEN.

Pierwszym modułem ze znacznym wkładem autorskim jest moduł generowania modeli VHDL zaprojektowanych macierzy w celu ich weryfikacji[27]. Generowane modele VHDL nie są modelami, które można implementować w układy FPGA, gdyż wykorzystują dane typu *Real*,

[24] http://kik.weii.tu.koszalin.pl/mvl/

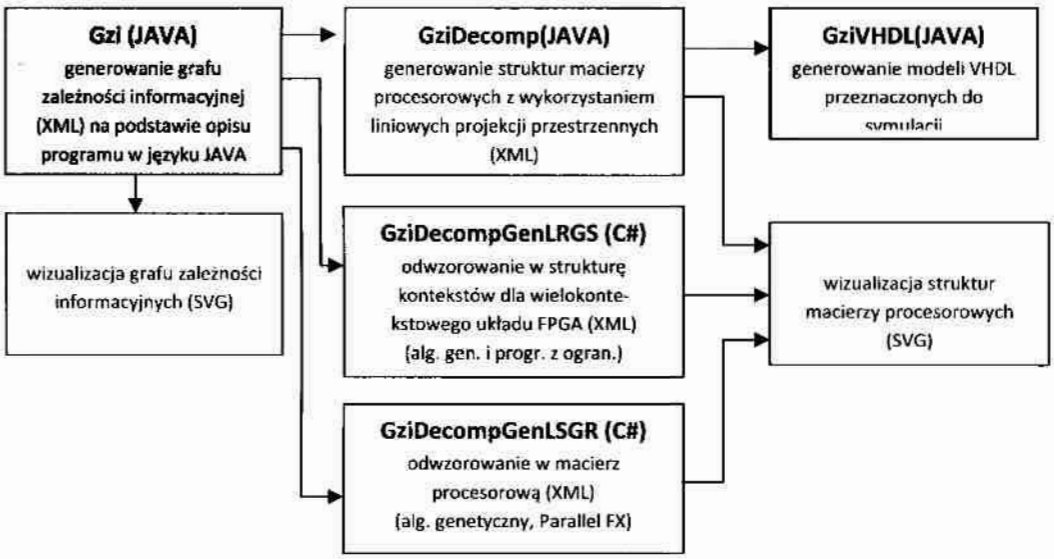
służą więc one jedynie do przeprowadzenia procesu symulacji. W opisywanym środowisku JGEN przetestowano generowanie grafów zależności informacyjnych oraz liniowe odwzorowanie w architektury macierzy procesorowych dla takich algorytmów algebry liniowej jak: dekompozycja macierzy LL^T Cholesky'ego, rozkład LU macierzy metodą Gaussa, rozwiązywania układów równań liniowych metodami redukcji wstecznej, podstawienia i Gaussa-Seidela, mnożenie macierzy. Również moduł wizualizacji macierzy procesorowych został zrealizowany ze znaczącym wkładem autorskim.

Pierwszym autorskim modułem środowiska JGEN jest moduł do odwzorowania grafów zależności informacyjnych zgodnie z metodą LRGS[1] w architektury równoległe, przeznaczone do implementacji w wielokontekstowych układach FPGA opsany w pracy [39] (moduł GziDecompGenLRGS z rys.33). Moduł ten wykorzystuje algorytm genetyczny do odwzorowania przestrzennego grafu i w przeciwieństwie do innych metod odwzorowania, nie wymaga określenia funkcji projekcji przestrzennej. Na tym etapie rozwoju środowiska pojawiły się pierwsze problemy z odwzorowaniem dużych grafów zależności informacyjnych (zawierających setki węzłów) w architektury macierzy procesorowych. Ze względu na potrzebę zwiększenia liczby generacji algorytmu genetycznego, zdecydowano się na zmianę platformy programowej z języka JAVA na platformę Microsoft .NET(język C#), gdyż na tej platformie testowe procedury algorytmu genetycznego były wykonywane w prawie 2-krotnie krótszym czasie. Kolejne prace nad modułem były poświęcone optymalizacji samego algorytmu genetycznego poprzez dostrajanie operatorów rekombinacji. Na kolejnym etapie rozwoju środowiska, w celu akceleracji algorytmu, zaimplementowano model wyspowy[92,93] w aplikacji rozproszonej zawierającej wiele serwerów obliczeniowych, utworzonej z wykorzystaniem technologii .NET Remoting(rys. 6)[28]. W opisywanym module w celu szybszego uzyskiwania dopuszczalnych rozwiązań dekompozycji zaimplementowano funkcję generowania startowej populacji z wykorzystaniem metod programowania z ograniczeniami[38], zaimplementowanych w bibliotece NSolver[25]. Kolejne badania zaprojektowanego modułu pokazały zalety zastosowania programowania z ograniczeniami nawet jako osobny serwer obliczeniowy, we wspomnianym wyspowym modelu algorytmu genetycznego(rys. 10)[40].

Kolejnym autorskim modułem środowiska JGEN jest moduł służący do odwzorowania grafów zależności informacyjnych w architektury macierzy procesorowych zgodnie z metodą LSGR[1], przeznaczonych do implementacji w standardowych układach FPGA. Również w tym module wykorzystano algorytm genetyczny do odwzorowania przestrzennego grafów w architektury macierzy procesorowych. Spowodowało to możliwość uzyskania zadanego kształtu projektowanej macierzy procesorowej oraz zwiększenie średniego obciążenia elementów przetwarzających macierzy w stosunku do znanych metod liniowego odwzorowania przestrzennego[50,76]. Ponadto dzięki zastosowaniu algorytmu

[25] http://www.cs.cityu.edu.hk/~hwchun/nsolver/

genetycznego w procesie odwzorowania nie ma konieczności definiowania przez użytkownika funkcji określającej projekcję przestrzenną. Trwają ciągłe prace nad akceleracją algorytmu genetycznego w celu zwiększenia liczby generacji dla przyjętego czasu działania programu. Do tej pory udało się zaimplementować rozszerzenie Microsoft Parallel FX[26], dzięki czemu można zwiększyć liczbę generacji algorytmu lub uzyskać większą liczbą dopuszczalnych rozwiązań odwzorowania, w krótkim przyjętym czasie działania algorytmu(do 20 min). W najbliższym czasie planowana jest równoległą realizacja utworzonego algorytmu genetycznego z wykorzystaniem procesorów graficznych GPU. Ponadto, w omawianym module, planowana jest dalsza modyfikacja operatora mutacji algorytmu, polegająca na ograniczeniu wartości do których może mutować gen, powodując w ten sposób zwiększenie prawdopodobieństwa uzyskania dopuszczalnego rozwiązania projekcji przestrzennej. Również połączenie odwzorowania przestrzennego z wykorzystaniem algorytmu genetycznego oraz liniowych funkcji projekcji wydaję się znacznie zwiększyć rozmiary grafów zależności informacyjnych, dla których będzie dokonywane odwzorowanie. Aktualne moduły projektu JGEN zostały zaprezentowane na rys. 33.



Rysunek 33. Aktualne moduły środowiska JGEN.

W przyszłości na ostatnim etapie pracy nad środowiskiem JGEN planowane jest utworzenie modułu generującego opisy VHDL dla zaprojektowanych architektur równoległych w modułach: GziDecomp, GziDecompGenLRGS oraz GziDecompGenLSGR przeznaczonych do implementacji w układach FPGA. Dzięki temu modułowi środowisko JGEN stanie się kompletnym generatorem IP-core dla algorytmów zawierających pętle zagnieżdżone, w tym algorytmów algebry liniowej. Generowane modele VHDL projektowanych architektur będą mogły wykorzystywać zarówno reprezentację stałoprzecinkową, jak i arytmetykę ułamkową.

[26] http://msdn.microsoft.com/en-us/magazine/cc163340.aspx

5. Podsumowanie

Podsumowując zrealizowane badania, nowa koncepcja implementacji macierzy procesorowych w wielokontekstowych układach reprogramowalnych, przedstawiona w podrozdziale 2.1., pozwala na realizację wybranych algorytmów algebry liniowej w czasie bliskim do czasu określonego ścieżką krytyczną grafu zależności informacyjnych. Ponadto w podrozdziale 2.2 przedstawiono metodę odwzorowania tych grafów dla nowej koncepcji obliczeń, która została zaimplementowana programowo z wykorzystaniem algorytmu genetycznego i programowania z ograniczeniami. Przedstawiono uzyskane rezultaty oraz porównano je z rezultatami uzyskanymi z wykorzystaniem innych znanych metod. Wyniki porównania świadczą o możliwości uzyskiwania krótszego czasu realizacji wybranych algorytmów algebry liniowej. Ponadto w przedstawionej koncepcji równoległej realizacji nie ma potrzeby projektowania i implementacji bloków generujących sygnały sterujące dla elementów przetwarzających równoległej architektury. Proponowana metoda odwzorowania grafów zależności informacyjnych w architektury równoległe wymaga jeszcze pewnych ulepszeń. Pomimo zastosowanych technik zrównoleglających obliczenia algorytmu genetycznego problematyczne jest uzyskanie dekompozycji dla grafów zależności informacyjnych, zawierających kilkadziesiąt tysięcy węzłów, w krótkim rozsądnym czasie działania programu, np. poniżej 1 godziny. Trudności może również sprawić wykorzystanie samych wielokontekstowych układów FPGA. Obecnie istnieje wiele koncepcji budowy takich układów, jednak są one trudnodostępne na rynku. Istnieje, oczywiście, możliwość samodzielnej budowy takiej platformy sprzętowej wykorzystującej popularne, dostępne układy FPGA, jednak budowa takiej platformy jest dość trudna. W podrozdziale 2.3 przedstawiono wykorzystanie algorytmu genetycznego do projektowania macierzy procesorowych. Do najważniejszych zalet tej metody można zaliczyć możliwość projektowania macierzy procesorowych o zadanym przez użytkownika kształcie oraz pełna automatyzacja procesu projektowania. Ponadto istnieje możliwość projektowania macierzy procesorowych bez znajomości tematyki z nimi powiązanych, gdyż proces projektowania nie wymaga od użytkownika określenia projekcji przestrzennej i czasowej grafu zależności informacyjnych. Podobnie jak dla wyżej opisywanej metody problematyczne staje się odwzorowanie grafów zależności informacyjnych z liczbą wierzchołków większą od 1000 w krótkim czasie projektowania. W celu pokonania opisywanych trudności autor rozprawy proponuje stosowanie metody zrównoleglenia obliczeń algorytmu genetycznego przedstawioną w podrozdziale 2.2. Dodatkowo w podrozdziale 2.3 przedstawiono drugą metodę zrównoleglenia algorytmu genetycznego z wykorzystaniem procesorów wielordzeniowych. Ponadto zastosowano dekompozycję grafów zależności informacyjnych, którą zaimplementowano w opracowanym algorytmie ewolucyjnym. Dzięki temu udało się dokonać odwzorowania dla grafów z liczbą węzłów większą od tysiąca w czasie kilkunastu minut. Obiecujące w dalszej perspektywie czasu wydaje się połączenie proponowanych metod odwzorowania przestrzennego wykorzystujących algorytm genetyczny ze znaną metodą wykorzystującą liniowe funkcje odwzorowania. Przy niewielkim nakładzie pracy możliwa jest modyfikacja proponowanego algorytmu genetycznego, tak aby dokonywał on

odwzorowania przestrzennego dla pewnego reprezentatywnego podgrafu zależności informacyjnej, a następnie dzięki wykorzystaniu liniowej projekcji dokonał tego odwzorowania dla pozostałej części grafu. Podejście takie umożliwi zachowanie opisanych korzyści wynikających z zastosowania proponowanej metody oraz zniweluje trudności w dokonywaniu odwzorowania dla większych grafów zależności informacyjnych. Wszystkie opisane w rozdziale 2 autorskie metody projektowe zostały zaimplementowane i przetestowane w środowisku projektowym JGEN(podrozdz. 4.2). Na podstawie rezultatów badań przedstawionych w rozdziale 2 oraz na podstawie porównania ich z rezultatami uzyskanymi z wykorzystaniem innych znanych metod, autor uważa że punkt 1 postawionej tezy został udowodniony.

Zastosowanie arytmetyki ułamkowej RFA do sprzętowej realizacji operacji arytmetycznych algorytmów algebry liniowej pozwoliło na implementację jednostek operacyjnych realizujących operacje arytmetyczne, przy zachowaniu zbliżonej dokładności obliczeń jak w przypadku liczb zmiennoprzecinkowych pojedynczej precyzji, z wykorzystaniem mniejszych zasobów sprzętowych układów FPGA lub działających z większą maksymalną częstotliwością, zwłaszcza przy podobnej liczbie stopni w potoku. Przewaga układów RFA jest jeszcze bardziej widoczna w stosunku do układów wykorzystujących reprezentację stałoprzecinkową, przy zachowaniu niemniejszej precyzji obliczeń. Arytmetyka ułamkowa posiada jednak niedogodność w postaci nieco gorszej implementacji operacji dodawania. Z tego względu jest ona najbardziej efektywna w zastosowaniach dla algorytmów, które w ścieżce krytycznej grafu zawierają operację dzielenia. Takimi przykładowymi algorytmami są niektóre algorytmy algebry liniowej oraz algorytm analizy autoregresyjnej, dla której również zaprojektowano akcelerator sprzętowy z wykorzystaniem arytmetyki RFA. W efekcie badań nad zastosowaniem arytmetyki ułamkowej RFA do realizacji wybranych algorytmów algebry liniowej zaprojektowano przykładowe architektury akceleratorów dla algorytmów redukcji wstecznej oraz rozkładu LU macierzy pasmowych metodą eliminacji Gaussa. Ponadto został opracowany autorski generator IPCore dla podstawowych operacji arytmetycznych (podrozdz. 4.1). Na podstawie wyników implementacji struktur przeznaczonych do realizacji operacji arytmetycznych, ich porównanie z innymi współczesnymi rozwiązaniami oraz na podstawie opisów implementacji projektów przykładowych akceleratorów wybranych algorytmów algebry liniowej również drugi punkt postawionej tezy autor uważa za udowodniony.

Podsumowując, na podstawie zaprezentowanych metod projektowych oraz rezultatów badań parametrów zaprojektowanych architektur równoległych i potokowych przeznaczonych do realizacji algorytmów algebry liniowej na platformie FPGA, autor pracy uważa postawiony cel za osiągnięty.

Do swoich osiągnięć naukowych autor zalicza:

- opracowanie metody (wykorzystującej algorytm genetyczny i programowanie z ograniczeniami) do projektowania architektur równoległych przeznaczonych do implementacji w wielokontekstowych układach FPGA, pozwalającej na skrócenie czasu realizacji zadanych algorytmów algebry liniowej w czasie zbliżonym do wartości krytycznej ścieżki algorytmu
- opracowanie metody projektowania macierzy procesorowych o zadanym kształcie (wykorzystującej algorytm ewolucyjny) pozwalającej na skrócenie czasu realizacji zadanego algorytmu algebry liniowej lub na zwiększenie średniego obciążenia elementów przetwarzających macierzy procesorowej w porównaniu do innych znanych metod
- implementacja w/w metod w postaci programowej (projekt JGEN – podrozdz. 4.2)
- zaprojektowanie struktur realizujących podstawowe operacje matematyczne na platformie FPGA z wykorzystaniem arytmetyki ułamkowej oraz zbadanie ich parametrów
- opracowanie generatora IPCore(soft-core – podrozdz. 4.1) podstawowych bloków operacyjnych, wykorzystujących arytmetykę ułamkową, przeznaczonych do implementacji w układach FPGA
- opracowanie, zbadanie i porównanie parametrów wyspecjalizowanych równoległych i potokowych akceleratorów przeznaczonych do realizacji wybranych algorytmów algebry liniowej pracujących w arytmetyce ułamkowej

Przedstawione zrealizowane badania mogłyby być uzupełnione o dodatkowe obszary. Przykładem badań które na pewno warto by jeszcze zrealizować byłoby rzeczywiste zaprogramowanie układów FPGA jedną z zaprojektowanych struktur oraz pomierzenie czasu realizacji algorytmu dla różnych rozmiarów macierzy danych wejściowych. Otrzymane w ten sposób wyniki warto by porównać np. z czasem realizacji tego samego algorytmu na komputerze klasy PC w celu wykazania ilościowej różnicy. Zrealizowanie takich badań wiązałoby się jednak z niemałym nakładem dodatkowej pracy raczej o charakterze czysto inżynierskim. Ciekawym również byłoby porównanie parametrów zaprojektowanych akceleratorów z parametrami przykładowej struktury uzyskanej z wykorzystaniem istniejących narzędzi HLL jak ImpulseC czy MitrionC. Jednak nie da się w sposób łatwy i szybki wygenerować takich struktur dla bardziej złożonych algorytmów. Istniejące przykłady obrazują np. proste algorytmy sortowania, natomiast w celu zrealizowania analogicznych algorytmów algebry liniowej niezbędne byłoby głębsze poznanie wymienionych narzędzi i zasad projektowania z ich wykorzystaniem. Tematy akceleracji obliczeń są zagadnieniami bardzo dynamicznie rozwijanymi i interesujące byłoby porównanie czasów realizacji wybranych algorytmów z różnymi innymi współczesnymi metodami.

Podstawową zaletą zastosowania platformy FPGA pozostaje możliwość implementacji całego akceleratora w postaci systemu jednoukładowego SoC, którego złożoność sprzętowa i pobór mocy będą znacznie mniejsze w porównaniu do takich platform jak procesory wielordzeniowe, wieloprocesorowe klastry obliczeniowe czy akceleratory oparte o procesory GPU. Ponadto w związku z ciągłym dynamicznym rozwojem tych układów pojawiają się coraz większe możliwości w zakresie liczby implementowanych elementów przetwarzających wyspecjalizowanych architektur równoległych.

W dalszej perspektywie czasu autor uważa za obiecujące połączenie zalet wykorzystania algorytmu genetycznego i liniowej funkcji odwzorowania przestrzennego w procesie projektowania macierzy procesorowych. Ponadto obiecujące wydaje się być wykorzystywanie arytmetyki ułamkowej do realizacji różnego rodzaju algorytmów wymagających akceleracji sprzętowej, np. przetwarzanie obrazów w czasie rzeczywistym. Ciekawym również wydaje się być zastosowania platformy FPGA do realizacji samych algorytmów genetycznych, które można w efektywny sposób zrównoleglić. Pojawiają się co prawda opracowania wyspecjalizowanych algorytmów, jednak trudno znaleźć uniwersalne narzędzia do implementacji różnego rodzaju algorytmów, np. w postaci generatora IPCore.

Literatura:

1. Maslennikow O., *"Podstawy teorii zautomatyzowanego projektowania reprogramowalnych równoległych jednostek przetwarzających dla jednoukładowych systemów czasu rzeczywistego."* , Monografia habilitacyjna. Wyd. Uczelniane Politechniki Koszalińskiej, 2004.
2. Kung H.T.: *Notes on VLSI Computations In Parallel Processing Systems*, D.J. Evans, ed. New York: Cambrige University Press, 1983, s.339-356
3. *Parallel and Distributed Computing Handbook*, Albert Y. Zomaya editor, McGraw-Hill, 1996
4. Culler D.E., Singh J.P, Gupta A.: *Parallel Computer Architectures*, Morgan Kaufmann Publishers, Inc.,USA,1999
5. Akhter S., Roberts J.: *Multi-Core Programming*, Intel Press, 2006
6. Wyrzykowski R.: *Klastry komputerów PC i architektury wielordzeniowe. Budowa i wykorzystanie.* Wydawnictwo EXIT, Warszawa, 2009
7. Kirk. D.B, Hwu W.W.: *Programming Massively Parallel processors*, Morgan Kaufmann Publishers, USA, 2010
8. Kruger J., Westerman r.: *Linear algebra operators for GPU implementation of numerical algorithms*, ACM Transaction on Graphics, Vol.22, No.3, str 908-916, ACM Press, 2003
9. Peterson M.: *FPGA Acceleration for outstanding performance.* Challenges and Opportunities, Parallel Processing and Applied Mathematics, Wrocław, Poland 2009
10. Dąbrowska A., Jamro E., Janiszewski M., Machaczek K., Russek P., Wiatr K., Wielgosz M.: *Akceleracja obliczeń HPC z zastosowaniem architektur FPGA,* Konferencja I3:Internet – infrastruktury – innowacje, Poznań, 2009
11. Hauck S. ed., Dehon A. ed.: *Reconfigurable computing. The theory and practice of FPGA-based computing.* Morgan Kaufman Publishers, Elsvier, 2008
12. Gokhale M.B, Graham p.S.: *Reconfigurable Computing. Accelerating Computation with Field-Programable Gate Array.*, Springer, 2005
13. Tomov S., Dongarra J., Baboulin M.: *Towards dense linear algebra for hybrid GPU accelerated manycore systems.* Parallel Computing. Volume 36, 2010
14. Tomov S. Nath R.,Ltaief H. Dongarra J.: *Dense linear Algebra Solvers for Multicore with GPU Accelerators*, International Parallel and Distributed Processing Symposium, Atlanta 2010
15. Kurzak J, Ltaief H., Dongarra J. Badia R.M: *Scheduling Linear Algebra Operations on Multicore Processors*, LAPACK Working Note 213, UT-CS-09-636, 2009
16. Kurzak J., Buttari A., Dongarra J.: *Solving system of Linear Equations on the CELL Processor Using Cholesky Factorization.*, IEEE Transactions on Parallel and Distributed Systems, Vol.19, Number 9, 2008
17. Kestur S.,Davis j. Williams O.: *BLAS Comparision on FPGA,CPU and GPU,* IEEE Computer Society Symposium on VLSI, 2010(porownanie, niski pobor mocy)

18. Zhuo L. Prasanna V.K.: *High preformance linear algebra on a reconfigurable supercomputer*, In. Supercomputing, 2005
19. Zhuo L. Prasanna V.K.: *High preformance designs for linear algebra operations on reconfigurable hardware*, IEEE transactions on Computers, Vol.57, No.8, 2008
20. Chen Y.K., Kung S.Y.: *Trend and Challenge on System-on-a-Chip Designs*, Journal of Signal Processing Systems 53, str.217-229, Springer, 2008
21. Williams J., George A.D., Richardson J., Gosrani K., Suresh S.,*"Computational Density of Fixed and Reconfigurable Multi-Core Devices for Application Acceleration"*, Proc. 4th Reconfigurable Systems Inst., Nat'l Center for Supercomputing Applications, Urbana, Illinois, 2008
22. C. Berthet. Going Mobile: The *Next Horizon for Multi-million Gate Designs in the Semi-Conductor Industry*. Proc. IEEE Conf. DAC'2002, pp. 375-378.
23. C. Fields. *Design reuse strategy for FPGAs*. Xcell jornal, Xilinx, 2000, pp. 40-42.
24. M. Keating, P. Bricaud. *Reuse Methodology Manual For System-on-a-Chip Design*. Kluwer Academic Publishers, 1999
25. L. Rizzatti. *How to achieve design productivity increases using architectural synthesis*. EDA Vision Magazine - January 2002.
26. Maslennikow O., **Ratuszniak P.**, Sergiyenko A.: *Generator opisów VHDL bloków operacyjnych działających w arytmetyce ułamkowej.* Pomiary, Automatyka, Kontrola, nr 8, 2008 r., s. 514-516.
27. **Ratuszniak P.**, Maslennikow O., Sołtan P.: *Generacja modeli VHDL równoległych jednostek przetwarzających.* Prace XII Konferencji Krajowej Komputerowe wspomaganie badań naukowych, Wrocław – Polanica Zdrój, 2005, pp. 127-132.
28. **Ratuszniak P.**, Bernatowicz D.: *Równoległa programowa realizacja algorytmów ewolucyjnych z wykorzystaniem technologii .NET Remoting*, Prace XIV Konferencji Krajowej Komputerowe wspomaganie badań naukowych, Szklarska Poręba, 2007
29. *XtremeDSP for Virtex4 FPGAs User Guide*. Xilinx, 31.10.2007
30. *DSP blocks in Stratix II and Stratix II GX Devices*, Altera, styczeń 2008
31. K.D. Underwood, K. S. Hemmert. *Closing the Gap: CPU and FPGA Trends in sustained Floating Point BLAS Performance.* Proc. IEEE Symp. Field Programmable Custom Computing Machines, FCCM 2004.
32. Underwood K.: *FPGAs vs. CPUs: trends in peak floating-point performance*, ACM/SIGDA 12th International Symposium on Field Programmable Gate Arrays, New York, USA, 2004
33. D. Kawakami, Y. Shibata, H. Amano: *„A prototype chip of multicontext with DRAM for virtual hardware"*, Design Automation Conference, 2001 r., Asia and South Pacific, Yokohama, Japan
34. Kazuteru Namba, Hideo Ito: *Proposal of Testable Multi-Context FPGA Architecture* - IEICE Transactions on Information and Systems, 2006

35. Weisheng Chong, Sho Ogata, Masanori Hariyama, Michitaka Kameyama: *Architecture of a Multi-Context FPGA Using Reconfigurable Context Memory*, IPDPS, vol. 4, pp.144a, 19th IEEE International Parallel and Distributed Processing Symposium, Denver, Colorado, USA, 2005
36. Kiełbik R., Moreno J.M., Napieralski A., Szymański T.: *High-Level partitioning for dynamically reconfigurable logic*, Mixed Design of Integrated Circuits Systems, Gdynia, Poland 2000
37. Moreno J.M, Napieralski A., Kiełbik R., Lacadena I., Insenser J.M.: *FIPSOC – New concept of programmable devices*, DDECS, Smolenice, Slovakia, 2000
38. **Ratuszniak P.**, Maslennikow O.: *New Conception and Algorithm of Allocation Mapping for Processor Arrays Implemented into Multi-context FPGA Devices*, International Multiconference on Computer Science and Information Technology, Mathematica Balkanica, Vol. 23, 2009
39. **Ratuszniak P.**, Maslennikow O., Sołtan P., Słowik A.: *Zastosowanie algorytmów ewolucyjnych w projektowaniu równoległych jednostek przetwarzających przeznaczonych do realizacji w wielokontekstowych układach FPGA*, Prace V Krajowej Konferencji Elektroniki, Darłówko Wschodnie, 2006
40. **Ratuszniak P.:** *Dekompozycja grafów zależności informacyjnych z wykorzystaniem algorytmu ewolucyjnego i programowania z ograniczeniami.* Metody wytwarzania i zastosowania systemów czasu rzeczywistego, Praca zbiorowa pod red. L Trybusa i L. Samoleja, Rozdział 31,Wydawnictwo Komunikacji i Łączności, 2010
41. Banerjee U.: *An introduction to a formal theory of dependiencies analysis*, J. Supercomput., 1988
42. Wyrzykowski R. Kanevski J., Maslennikow O., Maslennikow N., Lacinski L.: *Analitycal Method of Deriving Dependence Graph of Recursive Algorithms*. Proc. Int. Workshop Parallel Numeric'95, Italy,1995
43. Kaniewski J., Maslennikow O., Maslennikowa N., Lacinski L.: *Analitycal Method of Deriving Dependence Graph of Recursive Algorithms*. Proc. Int. Workshop Parallel Numeric'99, Austria, 1999
44. Kung S.Y.: *VLSI array processors*. Englewood Clifs, N.J., Prentice-Hall inc., 1988
45. Quinton P., Robert Y.: *Systolic algorithms and architectures*, Prentice Hall, Englewood Cliffs, 1991
46. Le Verge H., Mauras C., Quinton P.: *The ALPHA language and its use for the design arrays*, Journal of VLSI Signal Processing, 1991
47. Quinton P., Van Dongen V.: *The Mapping of Linear Recurence Equation on Regular Arrays*, Journal of VLSI Signal Processing, 1989
48. Wilde D.K., Sie O.: *Regular Array Synthesis using ALPHA*, Int. Conference on Application Specific Array Processors, 1994

49. Sergyienko A., Kaniewski Ju., Maslennikow O., Wyrzykowski R.: *A metod for mapping DSP algorithm into application specific processor*. Proc. 24-th Euromicro Conference on Parallel and Distributed Processing, Vasteras, Sweden, IEEE Comp. Soc. Press, Vol.1,1998
50. Wyrzykowski R., Kanevsky Ju., Maslennikov O.: *Mapping recursive algorithms into processor arrays*, Proc. Int . Workshop "Parallel numerics'94", Smolenice, Slovakia, 1994, str. 169-191
51. Kanevski J. Maslennikow O., Wyrzykowski R.: *VLSI implementation of Linear Algebraic Operations based on the orthogonal Faddeev Algorithm*, Parallel Computing:State-of-the-Art and Perspectives. Elsvier Science, 1996
52. Fortes J.A.B., Wah B.W., Shang W., Ganapathy K.: *Algorithm-Specific Parallel Processing with Linear Processor Arrays*, Advanced in Computers, Vol.38, ed. Yovitz M., vol. 33,197-245, Academic Press, 1994
53. Fimmel D., Merker R.: *Design of processor Arrays for Reconfigurable Architectures*, The Journal of Supercomputing, 19, str.41-56, Kluwer Academic Publishers, 2001
54. Bednara M., Teich J.: *Automatic Synthesis of FPGA Processor Arrays from Loop Algorithms*, The Journal of Supercomputing, Kluwer Academic Publishers, 2003
55. Fernando J., Jean J.N.: *Processor Array Design with FPGA Area Constraint*, IEEE Trans. On Computer Aided Design 18, str. 253-264, 1999.
56. Ganapathy k., Wah. B.W., Chien-Wei Li: *Designing a scalable processor array for recurrent computation*, IEEE Transaction on Parallel and Distributed Systems, str.840-856, 1997
57. Ganapathy k., Wah. B.W.: *Optimal Synthesis of Algorithm-Specific Lower-Dimensional Processor Arrays*, IEEE trans. Parallel and Distributed Systems, 1993
58. Fortes J.A.B., Fu K,S, Wah B.W.: *Systematic Design Approches for Algorithmically Specified Systolic Arrays*, Computer Architecture: From Concepts to Systems, ed. Milutinovic V.M., str. 454-494, Elsvier 1988.
59. P-Z. Lee,Z.M. Kedem.: *Mapping Nested Loop Algorithms into Multidimensional Systolic Array*, IEEE Trans. On Parallel and Distributed Systems, Vol. 5, str. 64-76, 1991
60. Moreno j.H. Lang T.: *Matrix computations on systolic-type arrays*, Kluwer Academic Publishers, 1992
61. Petkow N.: *Systolic Parallel Processing*, Elsvier Science Publishers b.V., 1993
62. Wyrzykowski R.: *Synteza architektur systolicznych na przykładzie systemu realizujqcego rozkład trójkqtny macierzy symetrycznych*, Zeszyty Naukowe Politechniki Częstochowskiej nr. 141. Seria Elektrotechnika(14), Częstochowa, 1990
63. Wyrzykowski R, Kanevski Ju., Maslennikow O.: *Systolic-Type Implementation of Matrix Computation Based on the Feddeev Algorithm*, IEEE Int. Conf. on Massively Parallel Computing Systems'94, Los Alamitos, CA, 1994
64. Lipowska-Nadolska E., Kwapisz M., Lichy K.: *Systoliczne przetwarzanie sygnałów cyfrowych*, Akademicka Oficyna Wydawnicza EXIT, Warszawa, 2007

65. Lipowska-Nadolska E., Kwapisz M.: *Systolic Realization of LU Decomposition*, Proc. XXVII International Conference IC SPETO, Ustroń, 2005
66. Lipowska-Nadolska E., Kwapisz M.: *Systolic Array Design with Dependence Graph*, Materiały XXVI international Conference SPETO, str. 443-446, Niedzica, 2003
67. Lenders P., Rajopadhye S.: *Multirate VLSI arrays and their synthesis*, IEEE Transactions on Computers, 1997
68. Sakdana G., Arias-Estrada M.: *Compact FPGA-based systolic array architecture suitable for vision systems.*, International Journal of High Performance Systems Architectures, vol. 1, no. 2, 2007
69. Azadfar M. M.: *Imlementation of Optimized Systolic Array Architecture for FSBMA using FPGA for Real-time Applications*, International Journal of Computer Science and Network Security, Vol. 8, No.3, 2008
70. Pramod Kumar Meher: *Design of Full-Pipelined Systolic Array for flexible Transportation-Free VLSI of 2-D DFT*, IEEE transactions on Circuit and Systems, Vol.52, No.2,2005
71. Hasan I., Khawaja Y., Bais A.: *A systolic array architecture for the Smith-Waterman algorithm with High Performance Cell Design*, Proc. of IADIS European Conference on Data Mining, 2008
72. Horn B. K. P.: *Rational Arithmetic for Minicomputers*. Software – Practice and Experience, Vol. 8, 1978, pp. 171-176.
73. P. Kornerup, D. W. Matula: *Finite-precision rational arithmetic: an arithmetic unit*. IEEE Transactions on Computers, C-32, 1983, pp. 378-388.
74. Maslennikow O., **Ratuszniak P.**, Sergyienko A., Pawłowski P.: *Zastosowanie arytmetyki ułamkowej w reprogramowalnych jednostkach przetwarzających systemów jednoukładowych.* Zeszyty naukowe Wydziału Elektroniki i Informatyki politechniki Koszalińskiej. Koszalin, 2009
75. Seduchin S.G.: *Projektirowanie i analiz sistoliczeskich algoritmow i struktur. Programirowanie.* N 2, str. 20-40, 1991
76. Kanevski j, Maslennikow O., Maslennikowa N.: *Design of FPGA-based Processor Array Architecture for Linear Algebra Algorithms Implementation*, Proc. of 3-th int. Conf. parallel Processing and Applied mathematics, Kazimierz Dolny, Poland, 1999.
77. De Dinechin D.F., Quinton P., Rajopadhye S., Risset T.: *First step in ALPHA*. Rapport de Recherche Irisa, no 1244, IRISA, 1999
78. Maslennikow O., Wąsik A., kaniewski J., Maslennikowa N.: *Program environment for designing of application specific FPGA-based parallel architectures*. proc. of the 7-th Int. Conf. on Mixed Design, MIXDES 2000, Gdynia, Poland, 2000, str. 605-608
79. Maslennikow O.: *CAD-Environment for deriving Dependence graphs of regular Algorithms and their mapping onto FPGA-Based ASIC Architectures*, proc. of the 4-th Int. Conf. CAD DD'2001, Mińsk, Białoruś, 2001

80. Bondhugula U., Ramanujam J.: *Automatic mapping of nested loops to FPGAs*, Proc. of the 12-th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, San Jose, California, USA, 2007
81. Schreiber R., Aditya S., Mahlke S., Kathail F., Rau B.R., Cronquist D., Sivaraman M.: *PICO-NPA: High-Level synthesis of non-programmable hardware accelerators.* J. VLSI Signal Process. Syst., 31(2), str. 127-142, 2002
82. O. Maslennikow: *Realizacja architektur macierzy procesorowych w dynamicznie reprogramowalnych układach FPGA.* Prace VII Konferencji Krajowej Reprogramowalne układy cyfrowe, RUC'2004, Szczecin, 2004, pp. 225-232.
83. Canto E., Moreno J.M., Cabestany J., Lacadena I., Inserser J.M. *A Method for Improving the Functional Density on Dynamically Reconfigurable Logic by Temporal Bipartitioning.* Proc. 7-th Int.Conf. Mixed design of integrated circuits systems, MIXDES'2000, Gdynia, Poland, pp. 155-160.
84. red. M. Kubale: *Optymalizacja dyskretna. Modele i metody kolorowania grafów*, WNT, Warszawa, 2002
85. D.J.A. Welsh., M.B. Powell: *An upper bound for the chromatic number of a graph and its application to timetabling problems*. The Computer Journal, 10(1), 1967
86. N. Christofides: *An algorithm for the chromatic number of a graph.* The Computer Journal, 14(1), 1971
87. Porumbel D.C, Hao J., Kuntz P.: *An Evolutionary Approach with Diversity Guarantee and Well-Informed Grouping Recombination for Graph Coloring*, Computers & Operations Research, Volume 37, Elsvier, 2010
88. Fleurent Ch.,Ferland J.A.: *Genetic and hybrid algorithms for graph coloring*, Annals of Operations Research 63, 1996,s.437-461
89. Galinier P., Hao J.: *Hybrid evolutionary algorithms for graph coloring*, Journal of Combinatorial Optimization, 1999
90. E. Malaguti, M. Monaci, P. Toth: *A metaheuristics approach for the vertex coloring problem*. INFORMS Journal on Computing, 20(2), 2008
91. Michalewicz Z.: *Genetic Algorithms + Data Structures = Evolutionary programs*, Springer-Verlag Berlin Heidelber, 1996
92. Gordon V. S., Whitley D.: *Serial and parallel genetic algorithms as function optimizers*, Proceedings of the fifth international Conference on genetic Algorithms, Illinois, 1993
93. Cantau-Paz E.: *Designing efficient and accurate parallel genetic algorithms.* PhD thesis, University of Illinois at Urbana-Champaign, 1999
94. H.W. Chun, "*NSolver*",.NET constraint-programming software library, http://www.cs.cityu.edu.hk/~hwchun/ [Accesed Apr. 1, 2009]
95. Maslennikow O., Maslennikowa N., Pawłowski P., Khadzhynov W., Sergyenko A.: *Realizacja w FPGA jednostek operacyjnych działających w arytmetyce ułamkowej*, Elektronika, nr 11, 2007, s. 34-36

96. Maslennikow O., **Ratuszniak P.**, Khadzhynov W., Pawłowski P., Berezowski R., Sergiyenko A.: *Osobliwości stosowania arytmetyki ułamkowej w nowoczesnych układach FPGA*, Elektronika, nr 11, 2008
97. Maslennikow O.: Sprawozdanie merytoryczne z realizacji projektu badawczego N515 002 32/0176 *„Zastosowanie arytmetyki ułamkowej w reprogramowalnych jednostkach przetwarzających systemów jednoukładowych"*, 2009
98. Maslennikow O., Sergyienko A., **Ratuszniak P.** *Implementation of Cholesky LL^T-Decomposition Algorithm in FPGA-Based Rational Fraction Parallel Processor. Proc. 14-th Int. Conf. on Mixed Design of Integrated Circuits and Systems, MIXDES'2007,* Poland, Ciechocinek, 2007, pp. 287-292
99. Maslennikow O., Lepekha V., Sergiyenko A., Wyrzykowski R.: *Cholesky LL^T – Algorithm implementation in FPGA-based processor,* 7th Int. Conf. on Parallel processing and Applied Mathematics, Gdańsk, Poland, 2007
100. Maslennikow O., **Ratuszniak P.**, Sergiyenko A.: *Implementation of Linear Algebra Algorithms in FPGA-based Rational Fraction Arithmetic Units. Proc. 9-th Int. Conf. Experience of Designing and Application of CAD Systems in Microelectronics, CADSM'2007,* Lwów-Polyana, 2007, pp. 228-234, IEEE Catalog Number 07EX1594
101. Maslennikow O., **Ratuszniak P.**, Maslennikowa N.: *Sprzętowa realizacja algorytmu redukcji wstecznej w układach FPGA Xilinx Virtex4, Elektronika,* nr 10, 2009 r., str. 102-105
102. Dou, Y., Vassiliadis, S., Kuzmanov, G.K, Gaydadjiev, G.N.: *64-bit Floating point FPGA Matrix Multiplication.* ACM/SIGDA 13-th Int. Symp. on Field Programmable Gate Arrays, Feb., 2005, FPGA-2005, (2005), 86-95
103. Scrofano R., Zhuo L., Parsana V. *„Area-Efficient Aritmetic Expression Evaluation Rusing Deeply Pineplined Floating-Point Cores", IEEE Trans. on VLSI Systems, Vol.16, No2,* 2008
104. *XST User Guide 10.01,* © 2002–2008 Xilinx
105. Maslennikow O., Sergiyenko A., Lesyk T. *Mapping DSP Algorithms into FPGA. Proc. 6-th IEEE East-West Design & Test Symposium,* EWDTS 2008, Lviv, Ukraine
106. Maslennikow O., Sergiyenko A., Maslennikowa N., **Ratuszniak P.**, Tomas A.: *Application Specific Processors for the Autoregressive Signal Analysis.* Parallel Processing and Applied Mathematics ,PPAM'2009, Lecture Notes in Computer Science 6067, Part I, pp.80-86,Springer-Verlag Berlin Heidelberg 2010
107. Sołtan P., Maslennikow O., **Ratuszniak P.**: *„Wizualizacja struktur macierzy procesorowych w standardzie SVG"*. Prace XI Konferencji Krajowej „Komputerowe wspomaganie badań naukowych", KOWBAN'2004, Wrocław – Polanica Zdrój, 2004, pp. 325-330.