UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI

Școala Doctorală de Electronică, Telecomunicații și Tehnologia Informației

Decizie nr. 965 **din** 16-11-2022

REZUMAT TEZĂ DE DOCTORAT

Ing. George-Vlăduț POPESCU

ARHITECTURI ȘI STRUCTURI PENTRU CALCUL ETEROGEN - ÎMBUNĂTĂȚIRI ALE TRANSFERULUI DE DATE PENTRU UN SISTEM ETEROGEN DE CALCUL

ARCHITECTURES AND STRUCTURES FOR HETEROGENEOUS COMPUTING - IMPROVEMENTS IN DATA TRANSFER FOR A HETEROGENEOUS COMPUTING SYSTEM

COMISIA DE DOCTORAT

Prof. Dr. Ing. Gheorghe BREZEANU Universitatea Politehnica din București	Președinte
Prof. Dr. Ing. Gheorghe ȘTEFAN Universitatea Politehnica din București	Conducător de doctorat
Prof. Dr. Ing. Corneliu BURILEANU Universitatea Politehnica din București	Referent
Prof. Dr. Ing. Aurel-Ștefan GONTEAN Universitatea Politehnica Timișoara	Referent
Prof. Dr. Ing. Dan NICULA Universitatea Transilvania din Brașov	Referent

BUCUREȘTI 2023

Cuprins

Capitolul 1

Introducere

1.1 Arhitecturi pentru Calcul Paralel

Nevoia tot mai mare de putere de calcul impusă de aplicațiile care necesită prelucrarea unor volume mari de date, precum cele care prelucrează sunetul sau imaginile sau cele utilizate în inteligența artificială, a condus la dezvoltarea arhitecturilor pentru calcul paralel, care, pe lângă problema reducerii timpului de execuție, încearcă să asigure și un consum redus de energie.

Fiecare aplicație care necesită calcul paralel este caracterizată de un model de calcul strâns legat de mecanismele matematice implicate. În [1] și [2] au fost identificate principalele modele de calcul, precum și prezența lor în mai multe aplicații generale. Din perspectiva hardware, fiecare model de calcul este definit de un comportament similar în ceea ce privește calculul și transferul de date.

De-a lungul timpului au fost propuse mai multe arhitecturi ce oferă suport hardware pentru diverse aplicații care necesită calcul paralel, fiecare dintre ele încercând să îmbunătățească raportul *număr de operații/putere consumată*. Exemple de astfel de arhitecturi sunt: Arhitectura Many Integrated Core (MIC), o soluție propusă de Intel [3–6], Graphics Processing Units (GPU) [7–10], procesoare cu un număr mare de nuclee concepute inițial pentru a accelera sarcinile legate de grafică, sau Tensor Processing Unit (TPU) de la Google, un circuit ASIC, conceput pentru aplicații de învățare automată [11, 12].

Deși arhitecturile prezentate anterior au performanțe teoretice ridicate, performanța lor reală atunci când sunt utilizate pentru diferite aplicații poate fi mult mai scăzută. După cum se demonstrează în [6, 13–15] și este sintetizat în [16], în funcție de aplicație, performanța reală poate scădea sub 50% din cea teoretică.

Această scădere poate fi explicată, printre altele, prin gradul scăzut de optimizare software și modul neadecvat în care este accesată memoria. Problema nepreluării datelor într-un mod optimizat, împreună cu discrepanța dintre creșterea performanței procesoarelor și cea a memoriilor [12], pot provoca întârzieri importante. Mai mult, este

dificil ca o arhitectură să fie optimizată din punct de vedere al timpului de execuție și al consumului de energie pentru o gamă largă de aplicații.

Necesitatea unei soluții care să ofere atât flexibilitate, cât și performanțe bune a dus la apariția sistemelor eterogene. Aceste sisteme integrează CPU-uri și alte elemente de procesare specializate pentru anumite sarcini. Pentru a sprijini dezvoltarea acestor sisteme, producătorii de FPGA au creat sisteme care combină un CPU și un FPGA pe același cip, permițând utilizatorului să-și descrie propriul nucleu de procesare, care poate fi configurat și adaptat aplicației țintă.

1.2 Motivația și Obiectivele Tezei

Cererea tot mai mare de putere de calcul la cele mai mici costuri posibile, privind atât fabricarea cipurilor, cât și energia necesară exploatării acestora, precum și diferența importantă dintre performanța teoretică maximă și performanța reală atunci când sunt utilizate în anumite aplicații, motivează cercetarea în domeniul arhitecturilor pentru calcul paralel.

Adăugarea cât mai multor nuclee de procesare nu este întotdeauna o soluție, deoarece se poate ajunge într-o situație în care, din cauza mai multor factori, o mare parte a puterii de calcul nu este activată în cazul unui număr important de aplicații. În plus, concentrarea asupra optimizării pentru o anumită aplicație va face ca arhitectura să fie lipsită de flexibilitate și ineficientă pentru alte aplicații.

O soluție ar putea fi sistemele de calcul eterogene, care pot oferi eficiența necesară pentru un număr mai mare de aplicații. Mai mult, un sistem care este și reconfigurabil, precum cele implementate folosind FPGA-uri, oferă flexibilitate, putându-se adapta la cerințele anumitor aplicații [17, 18].

Obiectivul general al acestei cercetări este de a propune un sistem de calcul eterogen complet funcțional, bazat pe o arhitectură îmbunătățită a unui Accelerator MapReduce, care oferă o soluție alternativă eficientă și flexibilă pentru aplicațiile care necesită procesare paralelă a datelor.

Pentru a realiza acest lucru, primul obiectiv major este ca, pornind de la o arhitectură MapReduce deja existentă, să fie propusă o nouă arhitectură cu transfer I/O de date îmbunătățit. Transferul de date a fost ales ca țintă a eforturilor de îmbunătățire deoarece, așa cum a fost menționat în secțiunea anterioară, neoptimizarea acestora reprezintă un factor important ce influențează negativ performanța unui sistem.

Pentru a fi utilizat, Acceleratorul trebuie să fie integrat într-un sistem care să îi asigure funcționarea prin transmiterea de instrucțiuni și date și prin citirea rezultatelor și transferarea lor în memoria principală. Prin urmare, al doilea obiectiv major este de a propune o arhitectură de sistem de calcul eterogen care integrează Acceleratorul MapReduce și de a implementa acest sistem pe o placă PYNQ-Z2 care conține SoC Zynq-7020.

Pentru ca sistemul să fie complet funcțional, utilizatorul trebuie să aibă acces la resursele sale cu ajutorul unui mediu software. Astfel, al treilea obiectiv major al cercetării este dezvoltarea unui mediu de programare care să permită scrierea programelor și executarea acestora pe sistemul de calcul eterogen.

Ultimul obiectiv major al tezei este dezvoltarea unor medii de testare, care vor fi folosite pentru a testa corectitudinea functionării sistemului și pentru a-l caracteriza din punct de vedere al timpului de executie și al energiei consumate.

1.3 Conținutul Tezei

Teza este structurată în șase capitole și descrie succesiv etapele de dezvoltare ale sistemului eterogen, de la înțelegerea problemei care motivează cercetarea curentă, până la mediul de evaluare, rezultatele obținute în urma acesteia, concluziile generale și propunerile de dezvoltare și îmbunătățire ulterioară a sistemului.

Capitolul 1 prezintă o imagine de ansamblu asupra principalelor arhitecturi pentru calcul paralel, dar și câteva detalii despre limitările acestora, evidențiate în literatura de specialitate în urma evaluării performanțelor atunci când sunt utilizate în diverse aplicații. Pornind de la acestea, se conturează motivația și obiectivele cercetării.

Capitolul 2 descrie principalele componente ale Acceleratorului MapReduce, care reprezintă punctul de plecare al cercetării, dar și setul de instrucțiuni specific acestuia. Apoi, sunt prezentate arhitectura propusă a sistemului eterogen și modul în care aceasta este implementată pe Zynq-7020. Pe lângă imaginea de ansamblu, sunt prezentate motivele alegerilor făcute în timpul dezvoltării sistemului, precum și detaliile de implementare și funcționare.

Capitolul 3 este dedicat descrierii detaliate a principiilor și modificărilor aduse de noul Accelerator MapReduce și modului în care aceste modificări duc la un transfer îmbunătățit de date între memoria principală și Accelerator.

Capitolul 4 se concentrează pe descrierea ultimului element al sistemului eterogen propus, un mediu de programare bazat pe limbajul Python, ce permite utilizatorului să acceseze resursele sistemului, să scrie biblioteci de funcții și programe folosind limbajul de asamblare specific Accelerator și să le ruleze. În plus, acest mediu poate fi folosit pentru a evalua corectitudinea funcționării sistemului.

În Capitolul 5 sunt prezentate cele două componente ale evaluării sistemului: evaluarea timpului de execuție folosind un mediu de testare în simulare și evaluarea corectitudinii funcționării sistemului folosind un mediu de testare hardware. În plus, a fost obținut un estimat al energiei consumate. Acest capitol conține, de asemenea, rezultatele testelor și concluziile care decurg din acestea.

În final, în ultimul capitol, Capitolul 6, sunt sintetizate principalele concluzii și rezultate, contribuțiile originale ale cercetării curente și perspectivele de îmbunătățire și dezvoltare ulterioară a sistemului de calcul eterogen propus.

Capitolul 2

Sistemul Eterogen de Calcul

În acest capitol este prezentat un sistem eterogen, pseudo-reconfigurabil, capabil să execute eficient operații de algebră liniară. Acesta integrează un procesor (CPU) cu unul sau mai multe nuclee, numit procesor Host, și un Accelerator, specializat în calcul paralel. În acest sistem, partea complexă a programului este executată de Host, în timp ce secvențele intensive de calcul sunt executate de către Accelerator.

O structură simplificată a sistemului de calcul eterogen este prezentată în Figura 2.1.

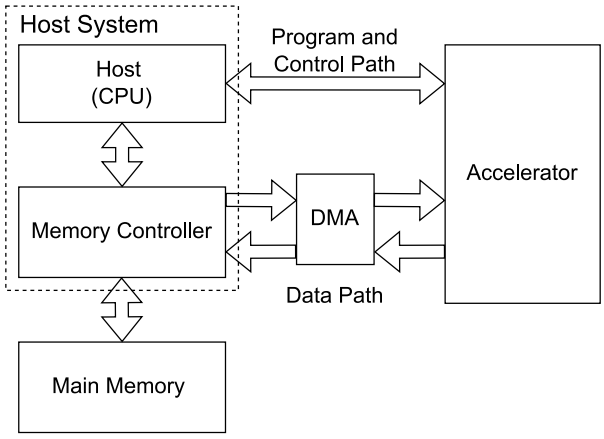

Figura 2.1 Arhitectura sistemului eterogen de calcul

Acceleratorul este un Accelerator MapReduce care comunică cu sistemul Host prin Calea de Program și Control (Program and Control Path) și prin Calea de Date (Data Path). Calea de Program și Control este folosită de Host pentru a accesa registrele de stare și control ale Acceleratorului și pentru a transmite comenzi. Calea de Date este folosită de Accelerator pentru a comunica cu memoria principală a sistemului. Pentru a transfera datele în mod eficient între memoria principală și Accelerator, Calea de Date este controlată de un DMA.

Mai multe arhitecturi ale acestui tip de Accelerator au fost propuse anterior, unele dintre ele fiind implementate pe siliciu [19, 20].

Arhitectura actuală se bazează pe Zynq-7020 de la Xilinx [21]. Pe această platformă, sistemul Host este reprezentat de sistemul de procesare (PS), construit în jurul unui

procesor dual-core ARM Cortex-A9, în timp ce Acceleratorul, modulul DMA și celelalte module necesare funcționării corecte sunt implementate pe FPGA-ul Artix-7 (PL).

2.1 Acceleratorul MapReduce

Acceleratorul MapReduce este un nucleu de calcul paralel care poate procesa cantități mari de date, fiind potrivit pentru partea de calcul intensivă a unui program.

Principalele componente ale Acceleratorului sunt Controlerul (Controller) și Unitatea de Procesare Paralelă (Parallel Processing Unit). Controlerul este folosit în principal pentru coordonarea activității Acceleratorului și pentru efectuarea de operații cu date scalare [22]. Unitatea de Procesare Paralelă este responsabilă pentru procesarea datelor organizate ca vectori, iar componentele sale principale sunt modulele Array, o matrice de celule de procesare, Distribution Network (Rețeaua de Distribuție) și Reduction Network (Rețeaua Scan-Reduce). O imagine de ansamblu a structurii Acceleratorului este prezentată în Figura 2.2.

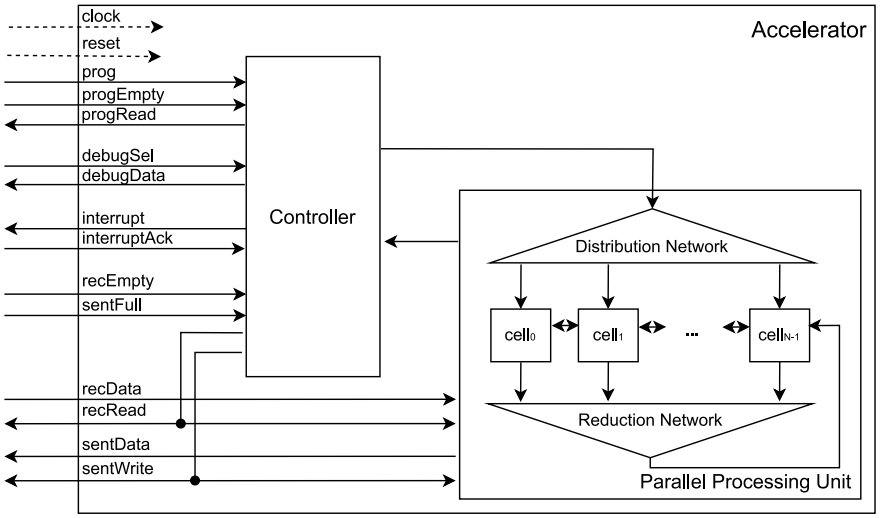

Figura 2.2 Structura Acceleratorului [22]

2.1.1 Controlerul

Rolul principal al Controlerului este de a coordona procesul de execuție în Accelerator. Structura sa este împărțită în trei secțiuni principale: o secțiune de program, responsabilă pentru controlul execuției, o secțiune de date, utilizată pentru a efectua operații cu scalari, având propria sa unitate de decodare, memorie de date scalară, acumulator și nucleu de procesare și o secțiune conector, folosit pentru pregătirea și trimiterea comenzilor către Unitatea de Procesare Paralelă.

Memoria de program stochează în fiecare locație de memorie două instrucțiuni, una pentru Controler și una pentru Unitatea de Procesare Paralelă. Interacțiunea cu

Acceleratorul presupune că acesta aplică funcții deja stocate în memoria de program asupra datelor primite pe Calea de Date.

Controlerul oferă, de asemenea, acces la resursele utilizate pentru depanare și măsurători de performanță, cum ar fi valoarea curentă a *PC* sau valoarea unui contor de ciclii de ceas (cycle counter).

2.1.2 Unitatea de Procesare Paralelă

Unitatea de Procesare Paralelă este responsabilă de prelucrarea datelor organizate ca vectori sau matrice. Componentele sale principale sunt Rețeaua de Distribuție (Distribution Network), Matricea de celule de procesare (Array) și Rețeaua Scan-Reduce (Scan-Reduce Network). O imagine detaliată a structurii Unității de Procesare Paralelă este prezentată în Figura 2.5.

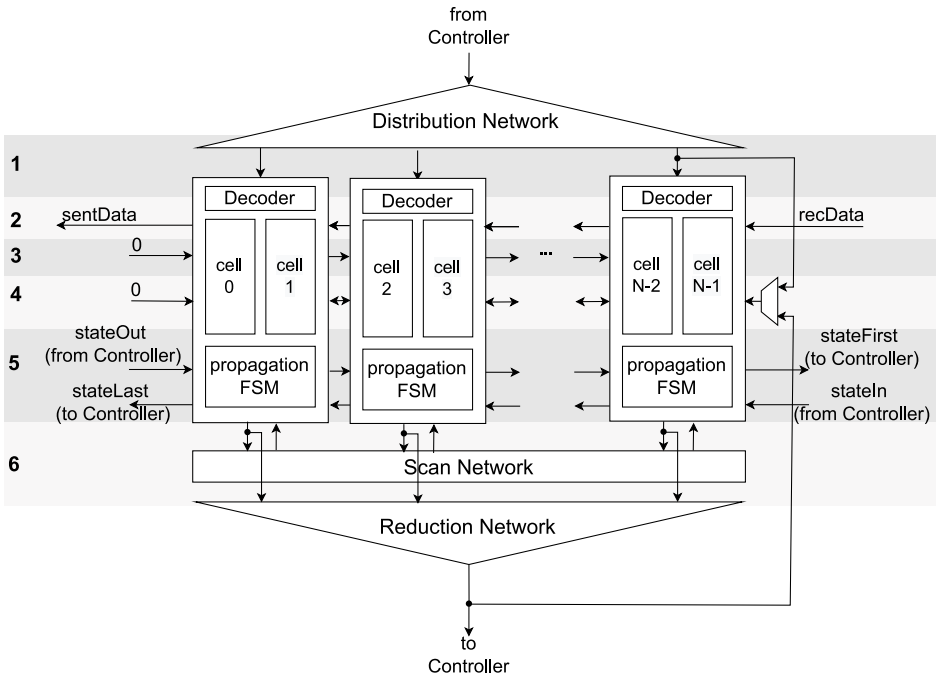

Figura 2.5 Structura Unității de Procesare Paralelă

Rețeaua de Distribuție este o structură de tip pipe-line folosită pentru a trimite instrucțiuni, adrese, date, sau comenzi pentru memorie de la Controler către Array.

Elementul central al Unității de Procesare Paralelă este modulul Array. Cu ajutorul acestuia, o anumită instrucțiune poate fi executată asupra unor date diferite în același timp, fiecare celulă având propriul controler, unitate de execuție și memorie de date [22].

Având în vedere prezența unei memorii de date locale în fiecare celulă și structura Array-ului, resursele interne de stocare a datelor pot fi văzute ca o matrice $2^V \times N$, unde V este numărul de biți ai adresei și N este numărul de celule.

În Figura 2.5, se poate observa că fiecare celulă din Array comunică cu celelalte celule și elemente din Unitatea de Procesare Paralelă prin mai multe canale de comandă, date și stare. Canalul de distribuție (1) asigură distribuirea comenzilor de la Controler

către fiecare celulă din Array, canalele de transfer I/O de date (2), stare de activare (3), transfer de date (4) și canalul de stare de propagare (5) asigură comunicarea datelor și stărilor între celule, iar canalul Scan-Reduce (6) asigură comunicarea între fiecare celulă și Rețeaua Scan-Reduce.

În funcție de modul în care este organizat Array-ul, o celulă poate împărți cu alte celule o unitate de decodare a instrucțiunilor și un automat de propagare. În implementarea actuală, celulele din Array sunt organizate în grupuri de două (double-cells), ceea ce înseamnă că fiecare unitate de decodare și automat de propagare deservește două celule.

Canalul de transfer I/O de date este implementat ca un lanț de registre de deplasare, distribuite în fiecare grup de celule. Propagarea datelor este controlată local de un automat, care asociază o stare de propagare fiecărei celule duble: *full*, dacă celula nu poate accepta date, sau *empty*, în caz contrar.

Un canal bidirecțional conectează celulele, oferind posibilitatea de a muta date în interiorul Array-ului. Acesta este implementat ca un registru serial-paralel numit registru global de deplasare, distribuit în fiecare celulă.

Rețeaua Scan-Reduce are două componente: Rețeaua Reduce, care efectuează operații pe vectori, cum ar fi adunarea, găsirea valorii maxime sau găsirea valorii minime, oferind ca rezultat un scalar, și Rețeaua Scan, care efectuează operații pe vectori. oferind ca rezultat un vector. Rezultatele vectoriale sunt trimise înapoi în Array, iar cele scalare sunt trimise înapoi la Array sau la Controler.

2.1.3 Setul de Instrucțiuni

Setul de instrucțiuni al Acceleratorului include instrucțiuni care vizează resursele din Controler sau pe cele din Unitatea de Procesare Paralelă. Mai mult, există instrucțiuni care asigură transferul de date și comenzi între cele două componente principale ale Acceleratorului, sau între Accelerator și interfața acestuia.

Fiecare locație de memorie pe 32 de biți din memoria de program a Acceleratorului stochează două instrucțiuni: una pentru Controler și una pentru Unitatea de Procesare Paralelă. Aproape toate instrucțiunile au un format de 16 biți. Singurele excepții sunt instrucțiunile de salt și apel, care vizează Controlerul, dar folosesc pentru adresă toți biții disponibili pe Calea de Program și Control, permițând accesul la o arie mai mare de memorie.

2.2 Arhitectura Sistemului

În această secțiune va fi prezentată arhitectura sistemului de calcul eterogen care integrează Acceleratorul MapReduce. Modul în care este potrivit pentru unele dintre aplicațiile ce necesită calcul paralel a fost analizat în [23] și [24].

2.2.1 Prezentare Generală a Platformei de Implementare

Având în vedere arhitectura sistemului de calcul eterogen, în care sunt necesare atât un procesor Host, cât și un Accelerator personalizat, a fost ales ca platformă de implementare sistemul Zynq-7020 de la Xilinx. Acesta integrează un Sistem de Procesare (PS), construit în jurul unui procesor dual-core ARM Cortex-A9 și a unui FPGA Artix-7 (PL), care oferă utilizatorului flexibilitatea implementării unui circuit logic personalizat. Arhitectura familiei Zynq-7000 este prezentată în Figura 2.8.

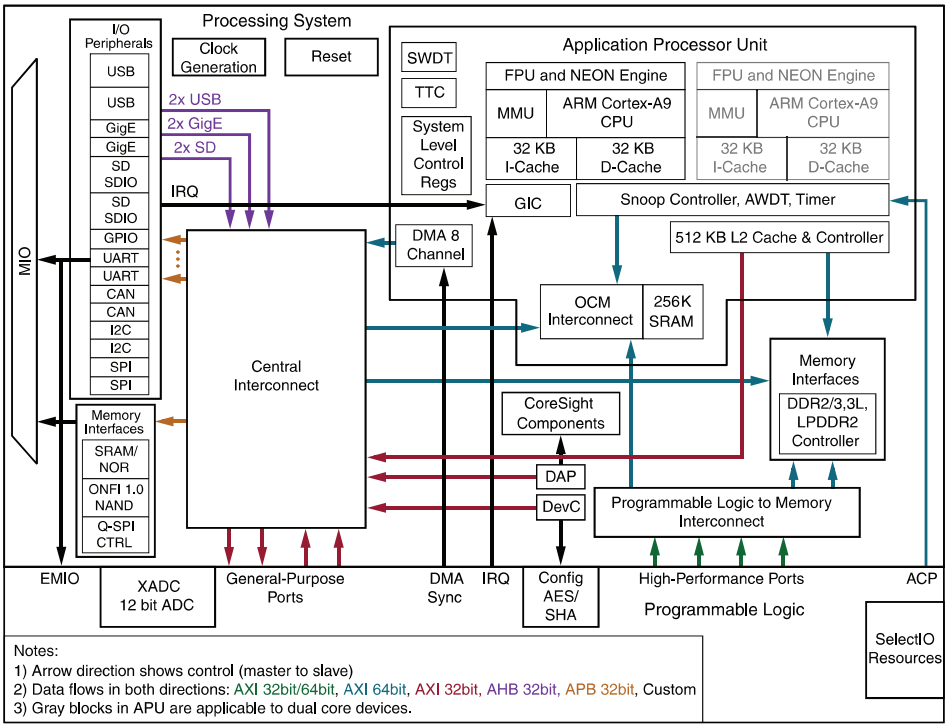

Figura 2.8 Arhitectura Zynq-7000 [21]

Pe lângă modulul Application Processing Unit (APU) ce reprezintă elementul central al Sistemului de Procesare, alte componente importante pentru sistemul eterogen sunt Generic Interrupt Controller (GIC) care gestionează întreruperile ce vin de la periferice și de la FPGA, DDR Memory Controller, care gestionează interacțiunile cu memoria DDR și interfețele PS-PL bazate pe AXI3. Deși modulele PS sunt compatibile cu AXI3, cele din PL folosesc de obicei AXI4.

2.2.2 Arhitectura de Nivel Superior a Sistemului

Fiind implementat pe Zynq-7020, procesorul Host al sistemului eterogen este reprezentat de Application Processing Unit din PS, în timp ce Acceleratorul este implementat în secțiunea PL.

Arhitectura sistemului, prezentată în Figura 2.9, ia în considerare modul în care Acceleratorul trebuie să interacționeze cu procesorul Host și cu memoria principală pentru a funcționa eficient.

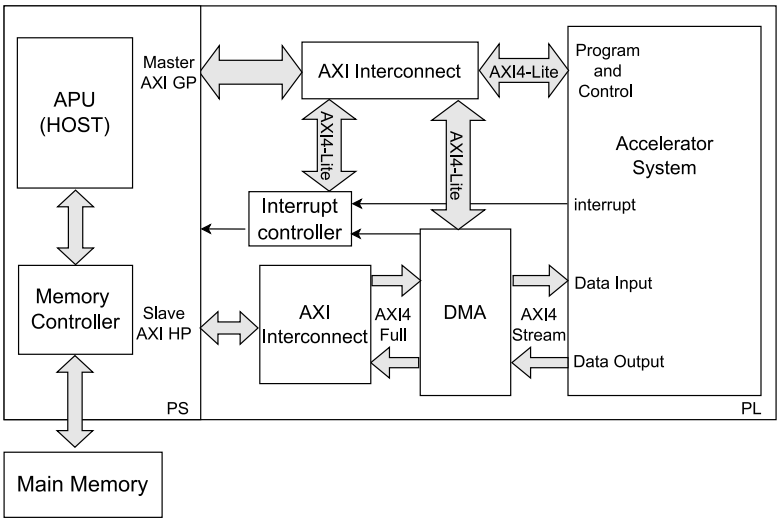

Figura 2.9 Arhitectura de nivel superior a sistemului eterogen de calcul

AXI4-Lite este o interfață de tip memory-mapped utilizată în arhitectura curentă pentru a conecta Calea de Program și Control la Host și pentru a configura alte componente din PL, cum ar fi modulul DMA și Controlerul de Întreruperi (Interrupt Controller). În PS, această magistrală este conectată la una dintre interfețele de uz general, care este conectată mai departe la modulul Central Interconnect.

Modulul DMA este implementat folosind IP-ul Xilinx LogiCORE AXI DMA. Canalele de date ale DMA sunt conectate la Accelerator folosind două magistrale AXI4-Stream și la PS folosind unul dintre cele patru porturi High-Performance, care implementează protocolul AXI4-Full. Magistrala High-Performance este conectată în continuare în interiorul PS la modulul DDR Memory Controller, care asigură interacțiunea cu memoria.

Controlerul de Intreruperi este responsabil pentru gestionarea întreruperilor de la DMA și Accelerator și este implementat folosind IP-ul Xilinx LogiCORE AXI Interrupt Controlller. Semnalul său de ieșire este conectat la modulul Generic Interrupt Controller (GIC).

După cum a fost menționat anterior, modulele din PS sunt compatibile cu AXI3. Pentru a asigura compatibilitatea între PS și modulele din PL, se adaugă module suplimentare de interconectare (AXI Interconnect).

2.2.3 Calea de Program și Control

Calea de Program și Control este folosită în principal pentru trimiterea instrucțiunilor de la Host către Accelerator și pentru accesarea registrelor de control și stare. În plus, este utilizată pentru a configura și controla modulul DMA, responsabil pentru transferurile de date, și Controlerul de Întreruperi (Interrupt Controller), responsabil pentru gestionarea întreruperilor de la DMA și Accelerator.

Calea de Program și Control se bazează pe magistrala AXI4-Lite [25]. Sistemul de Procesare controlează această magistrală printr-unul dintre porturile sale AXI de uz general (AXI GP). Compatibilitatea dintre portul AXI3 PS și magistrala AXI4-Lite este asigurată de un modul de interconectare (Xilinx LogiCORE AXI IP). Arhitectura Căii de Program și Control este prezentată în Figura 2.10.

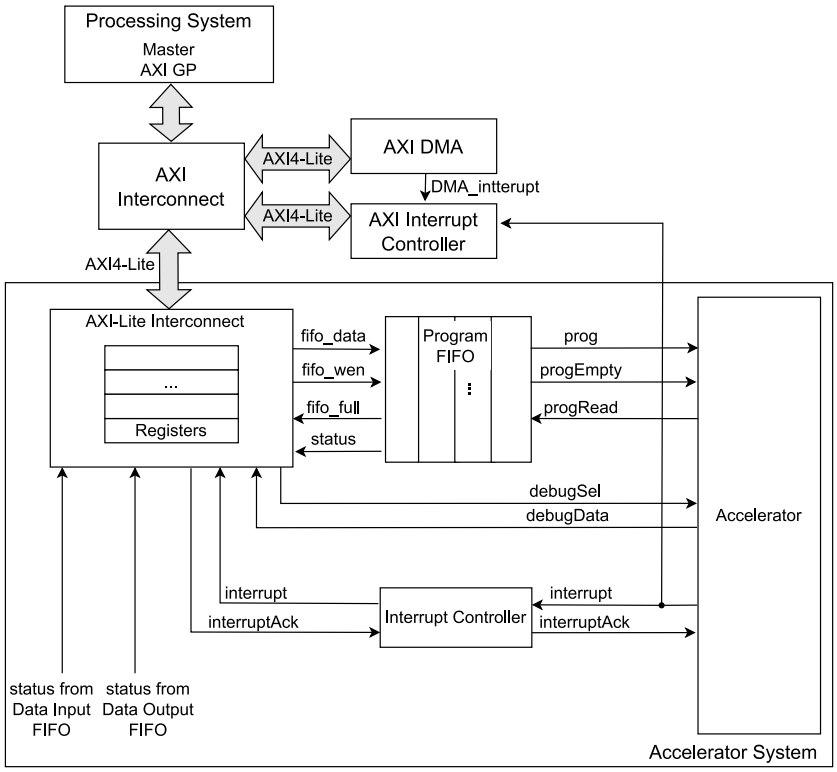

Figura 2.10 Arhitectura Căii de Program și Control

Acceleratorul interacționează cu Calea de Program și Control folosind 3 grupuri de semnale: grupul de program, grupul de întreruperi și grupul de depanare. Grupul de program este folosit pentru a transfera instrucțiunile de la Host la Accelerator, grupul de întreruperi este folosit de Accelerator pentru a emite o întrerupere și a primi o confirmare de la Host, iar grupul de depanare este folosit pentru a transfera informațiile de depanare de la Accelerator la registrul corespunzător din modulul AXI-Lite Interconnect.

AXI-Lite Interconnect este modulul care interfațează Acceleratorul cu magistrala AXI4-Lite. Principalele sale componente sunt registrele de stare și control și logica care asigură scrierea și citirea corectă a acestora.

Modulul Program FIFO este folosit ca buffer pentru instrucțiunile trimise de procesorul Host. Datele de intrare pentru acest FIFO vin de la modulul AXI-Lite Interconnect. De fiecare dată când Host scrie o instrucțiune la adresa 0x00 pe interfața AXI4-Lite, aceasta va fi redirecționată către FIFO.

Controlerul de Întrerupere din interiorul Acceleratorului este o modalitate alternativă de a gestiona întreruperile generate de Accelerator.

2.2.4 Calea de Date

Calea de Date este folosită pentru a transfera cantități mari de date între memoria principală și Accelerator. Are o componentă Data Input, prin care Acceleratorul primește de la memoria principală datele ce trebuie prelucrate și o componentă Data Output, prin care rezultatele sunt citite și stocate în memoria principală.

Pentru a putea transfera cantități mari de date în cel mai eficient mod, Calea de Date este controlată de un modul DMA. Arhitectura Căii de Date este prezentată în Figura 2.13.

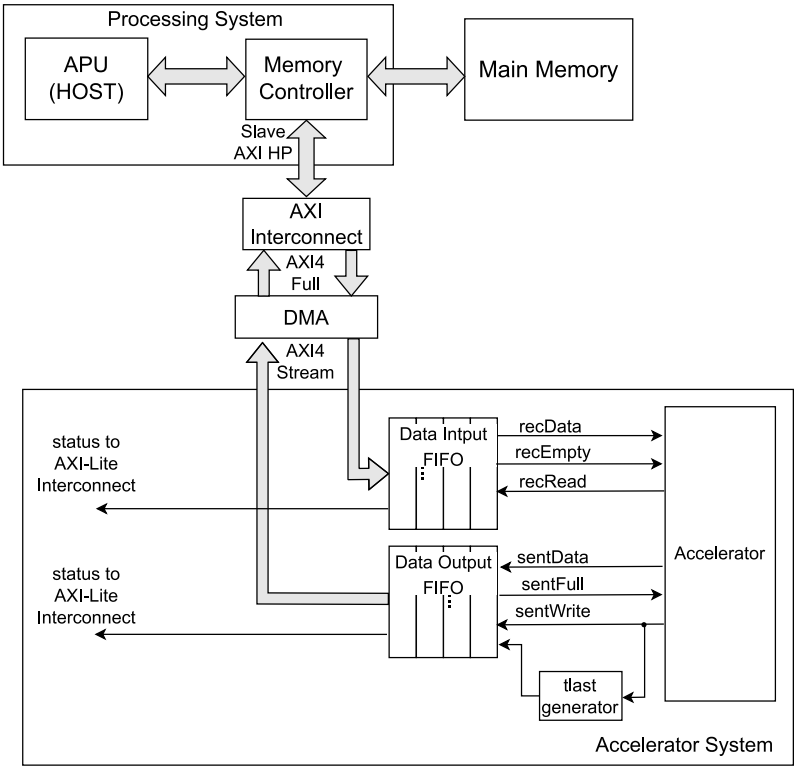

Figura 2.13 Arhitectura Căii de Date

Memoria principală este accesată atât de Host, cât și de DMA folosind modulul DDR Memory Controller, integrat în secțiunea PS a Zynq-7020. Calea de Date din PL este conectată la DDR Memory Controller prin unul dintre cele patru porturi de tip AXI High-Performance ale PS.

Funcționarea modulului AXI DMA se bazează pe două canale care operează independent: canalul Memory-Map to Stream (MM2S), utilizat pentru a transfera date din memoria principală la Accelerator și canalul Stream to Memory-Mapped (S2MM), folosit pentru a transfera date de la Accelerator în memoria principală. Cele două canale independente ale DMA sunt responsabile pentru conversia protocolului de la memory-mapped, care asigură conexiunea cu DDR Memory Controller, la AXI4-Stream [26], care asigură conexiunea cu Acceleratorul.

Interfațarea Acceleratorului cu cele două canale AXI4-Stream ale DMA se realizează prin două structuri FIFO configurabile.

Modulul Data Input FIFO este utilizat pentru a stoca temporar datele transferate din memoria principală. Datele vor rămâne aici până când Acceleratorul le va citi. Acesta este un FIFO sincron cu două protocoale diferite pentru scriere și citire. Capătul de intrare al FIFO este o interfață simplificată AXI4-Stream, iar capătul de ieșire este o interfață FIFO de bază, prin care Acceleratorul extrage date și le mută în memoria sa internă.

Modulul Data Output FIFO este utilizat pentru a stoca rezultatele de la Accelerator până când un transfer de citire este inițiat de către DMA. Capătul de intrare al FIFO este o interfață FIFO de bază, iar capătul de ieșire este o interfață simplificată AXI4-Stream. Pe lângă semnalele de bază ale interfeței FIFO, intrarea Data Output FIFO necesită un semnal *tlast*, care va fi folosit de DMA pentru a indica limita unui pachet citit de date.

Informații precum starea *full* și *empty* sau numărul de locații ocupate din fiecare FIFO sunt disponibile procesorului Host prin intermediul registrelor de control și stare.

Capitolul 3

Îmbunătățiri ale Transferului de Date pentru Acceleratorul MapReduce

3.1 Punctele Slabe ale Transferului de Date

În procesul de îmbunătățire a transferului de date, arhitectura actuală a Acceleratorului trebuie analizată din două perspective: identificarea acelor elemente care introduc întârzieri pe Calea de Date și analizarea posibilității de separare a fluxului de transfer I/O de date de cel de procesare. Dacă această separare este posibilă, datele pot fi aduse în memoria Acceleratorului la momentul potrivit, astfel încât timpul în care o secvență de procesare își așteaptă datele de intrare să fie cât mai scurt posibil.

În arhitectura inițială, transferurile de date între modulul Array și FIFO-urile de date sunt coordonate de Controler. Automatele de propagare din fiecare celulă dublă utilizate în procesul de deplasare a datelor introduc o întârziere de un ciclu de ceas la fiecare pas de deplasare, având în vedere că datele nu pot merge mai departe până când următoarea celulă este în starea *empty*. De exemplu, deplasarea datelor prin lanțul de registre I/O pentru un Array cu 8 celule este prezentată în Figura 3.1.

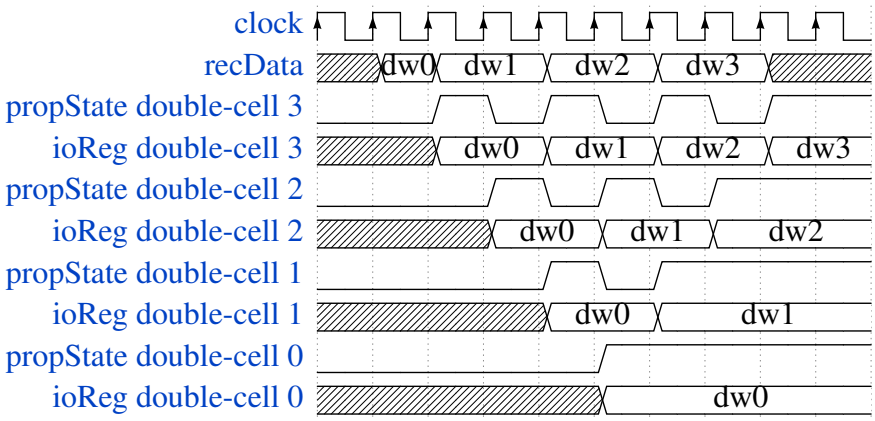

Figura 3.1 Deplasarea datelor prin lanțul de registre I/O pentru un Array cu 8 celule de procesare

Pentru a realiza separarea fluxului de transfer I/O de date de cel de procesare, cele două trebuie să utilizeze resurse diferite cu căi de control separate. Deoarece operația de transfer de date utilizează lanțul de registre I/O, care este o resursă de stocare separată de cele utilizate pentru prelucrarea datelor, independența acesteia depinde doar de implementarea unui modul care preia controlul de deplasare de la Controler.

În plus, în arhitectura curentă, transferurile de date între registrele I/O și memoriile interne ale Array-ului trebuie să treacă prin Acumulator. În contextul separării fluxurilor de date și de procesare, este necesar ca transferurile de date între registrele I/O și memoriile interne să fie efectuate independent de Acumulator. În plus, deoarece pot exista solicitări simultane de acces la memoria internă de la cele două fluxuri, este necesar să se implementeze o logică suplimentară de arbitraj [22].

Având în vedere observațiile anterioare, fluxurile de execuție din arhitectura curentă a Acceleratorului MapReduce și din cea îmbunătățită sunt prezentate în Figura 3.3. P1, P2 și P3 sunt secvențe de procesare, iar D1, D2 și D3 sunt secvențele de transfer ale datelor de intrare corespunzătoare. După cum se poate observa, în arhitectura inițială, noile date pot fi transferate numai după terminarea secvenței curente de procesare. Dacă fluxurile de transfer I/O de date și de procesare devin independente, transferul de date noi poate începe imediat după ce transferul anterior s-a încheiat [22].

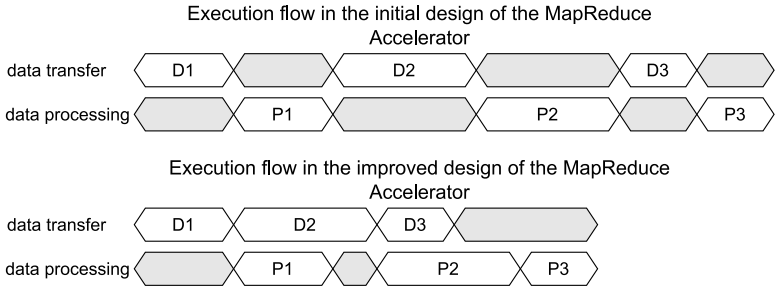

Figura 3.3 Fluxurile de execuție pentru arhitecturile inițială și îmbunătățită ale Acceleratorului MapReduce [22]

3.2 Arhitectura Îmbunătățită a Modulului Array

Unul dintre elementele care încetinește propagarea datelor prin lanțul de registre I/O este întârzierea de un ciclu de ceas introdusă la fiecare pas de deplasare de către automatul de propagare. Această întârziere este cauzată de faptul că fiecare celulă dublă așteaptă ca următoarea să devină disponibilă pentru a putea transfera mai departe datele.

O modalitate de a elimina această întârzierea nedorită este împărțirea celulelor duble în două grupuri cu căi diferite de propagare a datelor, ce funcționează alternativ. În acest fel, vor putea fi acceptate de către Array date noi la fiecare ciclu de ceas.

O observație importantă este că noul mod de organizare a celulelor afectează doar structura canalului de transfer I/O de date (2) și a celui de stare de propagare (5),

prezentate în Figura 2.5. Din perspectiva funcționalității, comportamentul va rămâne același, această modificare fiind transparentă pentru utilizator.

Noul mod de organizare a celulelor în Array este prezentat în Figura 3.4. Sunt reprezentate doar noile conexiuni ale celor două canale implicate în transferul I/O de date, deoarece modul de conectare a celorlalte rămâne neschimbat.

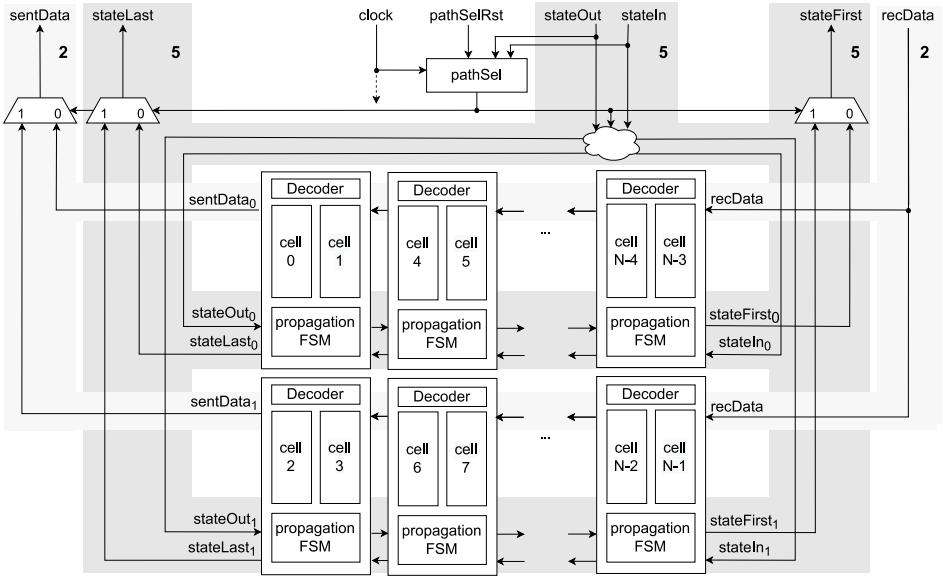

Figura 3.4 Arhitectura îmbunătățită a modulului Array

Folosind această nouă arhitectură, este posibil transferul de date către și de la Array la fiecare ciclu de ceas, așa cum este prezentat în Figura 3.5 pentru un Array cu 8 celule.

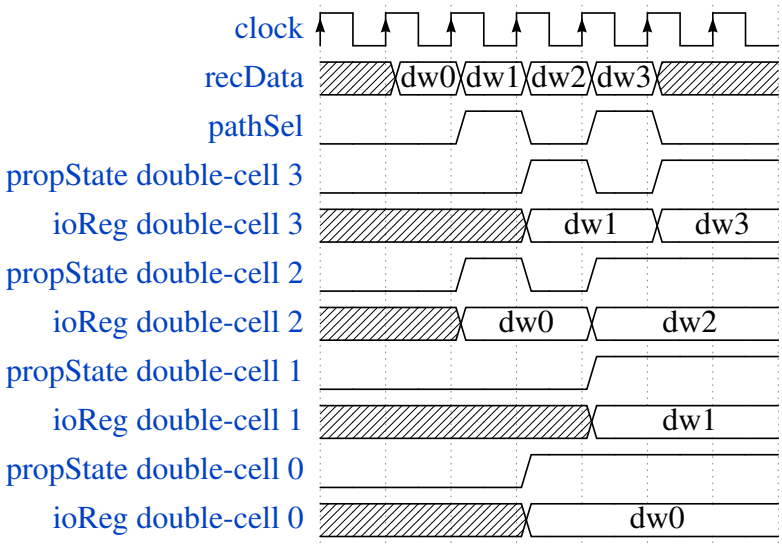

Figura 3.5 Deplasarea datelor prin lanțul de registre I/O pentru un Array cu 8 celule de procesare, implementând arhitectura îmbunătățită

3.3 Modulul Data Transfer Engine

Pentru a îndeplini cerințele prezentate în secțiunile anterioare, a fost implementat un nou modul numit Data Transfer Engine [22]. Acesta are rolul de a coordona transferul de date între cele două structuri FIFO ale Căii de Date și Accelerator, având un grad ridicat de independență față de stările Controlerului și Unității de Procesare Paralelă. În acest fel, cele două module pot continua prelucrarea datelor deja prezente în memoria internă de date, în timp ce datele care vor fi utilizate în procesările ulterioare sunt transferate. Data Transfer Engine poate accesa memoria internă pentru transferuri de citire și scriere dacă instrucțiunile curente ce rulează în Array nu inițiază solicitări similare. În caz contrar, va aștepta ca accesul cu prioritate mai mare să se termine.

Pentru ca Data Transfer Engine să poată efectua corect transferurile I/O de date și să le sincronizeze cu alte module atunci când este necesar, sunt introduse următoarele comenzi și instrucțiuni noi:

- TINRUN: Transfer Input Run: Comanda pentru Data Transfer Engine care efectuează transferul unei matrice de date în memoria de date din Array. Această comandă trebuie să fie urmată de trei parametri: adresa memoriei interne de date unde va începe stocarea și numărul de linii și de coloane ale matricei. Dacă numărul de coloane este mai mic decât numărul de celule din Array, fiecare linie de date va fi completată cu zerouri.

- TOUTRUN: Transfer Output Run: Comanda pentru Data Transfer Engine care efectuează transferul unei matrice de date din memoria de date a modulului Array către Data Output FIFO. Această comandă trebuie să fie urmată de trei parametri: adresa memoriei interne de date de unde va începe citirea, numărul de linii și numărul de coloane. Dacă numărul de coloane este mai mic decât numărul de celule din Array, transferul unei linii de date va fi considerat finalizat după deplasarea datelor necesare.

- WAITRESREADY: Wait for Result to be Ready: Comanda pentru Data Transfer Engine care îi spune modulului să aștepte până când Controlerul marchează un rezultat ca disponibil înainte de a începe transferul acestuia către ieșire.

- cWAITMATW(scalar): Wait for Matrices to be Written: Instrucțiune pentru Controler care determină blocarea operațiilor până când Data Transfer Engine confirmă că un număr de matrice egal cu *scalar*, necesare pentru secvența de procesare, au fost transferate în memoria internă a modulului Array.

- cRESREADY: Result Ready: Instrucțiune pentru Controler prin care acesta confirmă că o secvență de procesare este terminată și Data Transfer Engine poate citi rezultatul.

3.3.1 Arhitectura Modulului Data Transfer Engine

Modulul Data Transfer Engine va prelua responsabilitatea transferurilor I/O de date către și dinspre memoria internă a modului Array și, de asemenea, se va asigura că transferurile sunt sincronizate cu secvențele de procesare pentru o funcționare corectă a Acceleratorului.

Interfața modulului este compusă din semnale prin care acesta stabilește conexiuni cu celelalte componente ale Acceleratorului și cu restul sistemului de calcul eterogen. Structura Acceleratorului care integrează modulul Data Transfer Engine este prezentată în Figura 3.6.

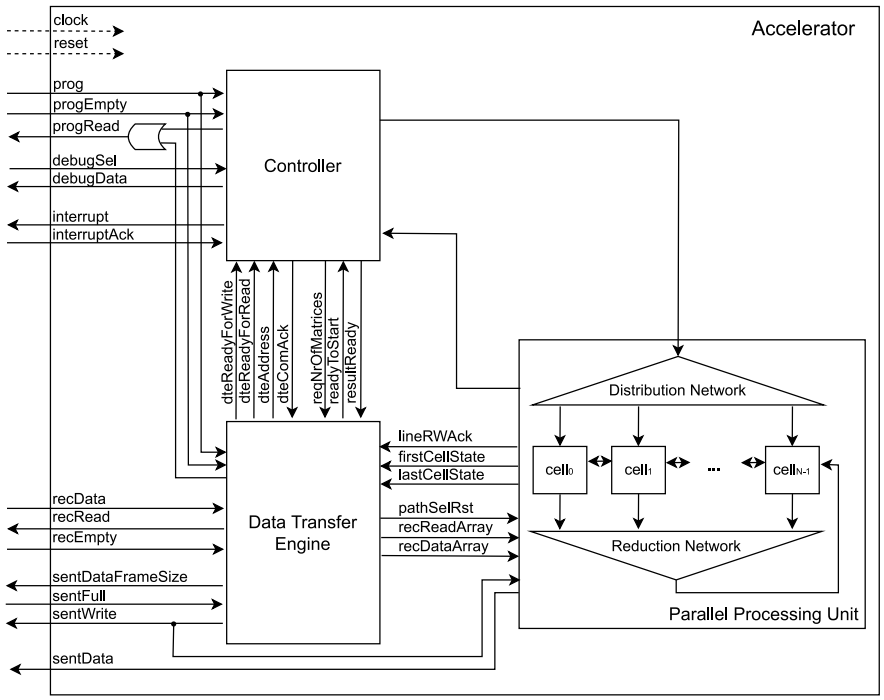

Figura 3.6 Structura Acceleratorului integrând modulul Data Transfer Engine [22]

Structura internă a Data Transfer Engine este prezentată în Figura 3.7. Nucleul acestuia este un automat care coordonează și sincronizează funcționarea tuturor celorlalte componente.

Modulul Command FIFO este o structură de memorie First-In, First-Out utilizată pentru a stoca comenzile și parametrii acestora. Scrierea și citirea comenzilor și parametrilor sunt controlate de modulele Parameter Write Counter și Parameter Read Counter.

Modulele Command Register, Address Register, Line Counter, Column Counter și Padding Counter sunt folosite pentru a păstra informațiile despre transfer până la sfârșitul acestuia. Modulul de tip contor Ready to Start Counter este folosit pentru a sincroniza începutul unei secvențe de procesare în Unitatea de Procesare Paralelă cu finalizarea transferului matricelor necesare în memoria internă, iar Result Ready Counter este folosit pentru a sincroniza sfârșitul unei secvențe de procesare cu citirea matricelor rezultate.

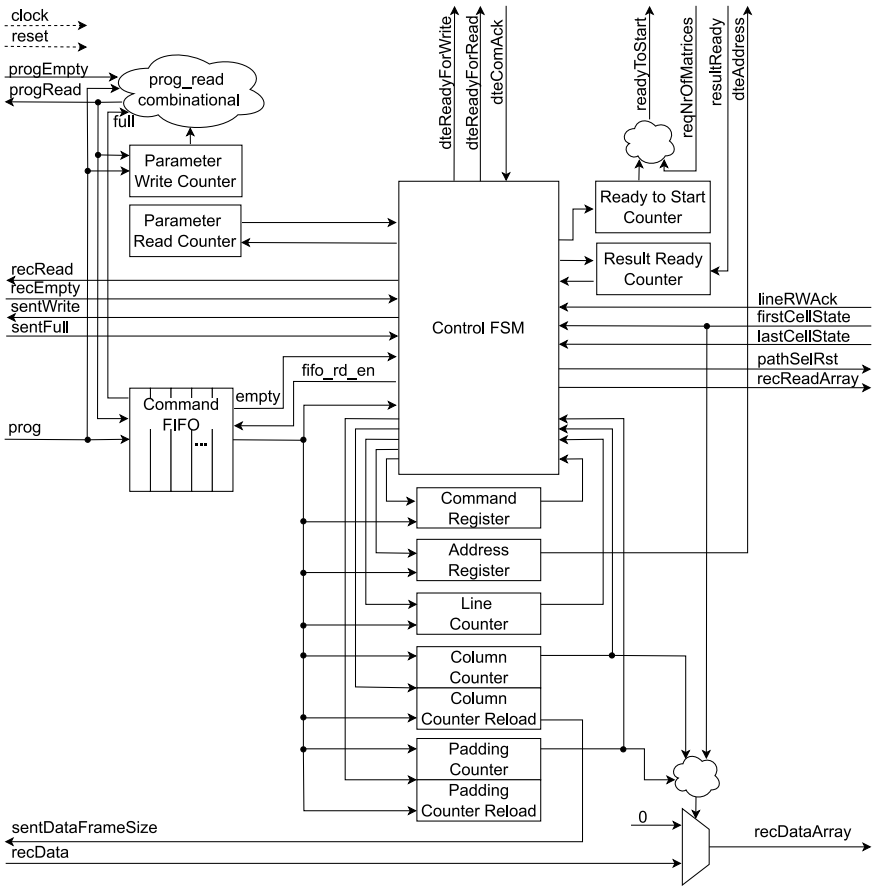

Figura 3.7 Arhitectura modulului Data Transfer Engine

3.3.2 Transferul de Date Folosind Data Transfer Engine

Pentru ca modulul Data Transfer Engine să funcționeze corect, au fost făcute câteva modificări suplimentare în Controler și în Unitatea de Procesare Paralelă.

Pentru a putea gestiona cererile de acces la memoria internă a fiecărei celule, provenite din cele două surse (instrucțiunile trimise de Controler pentru a fi executate de către Array și solicitările de la Data Transfer Engine), a fost adăugată logica de arbitraj în secțiunea conector a Controlerului. Detaliile de arhitectură sunt prezentate în Figura 3.9.

Deoarece pachetele de date și comenzi trimise de Controler către toate celulele din Array prin Rețeaua de Distribuție au câmpuri separate pentru informațiile implicate în lucrul cu memoria internă, decizia de aprobare a solicitărilor de la Data Transfer Engine se ia prin testarea acestor câmpuri. Dacă pachetul curent nu conține accesări ale memoriei, dar există solicitări de scriere sau citire de la Data Transfer Engine, se ia decizia de aprobare a acestora și se trimite o confirmare.

Logica care se ocupă de intrarea și ieșirea datelor din memoria internă a fost modificată pentru a permite o conexiune directă între registrul I/O și memorie. În acest fel, datele nu mai trebuie să treacă prin Acumulator, permițând ca acesta să fie utilizat pentru alte instrucțiuni în timp ce o linie de date este scrisă sau citită din memoria internă.

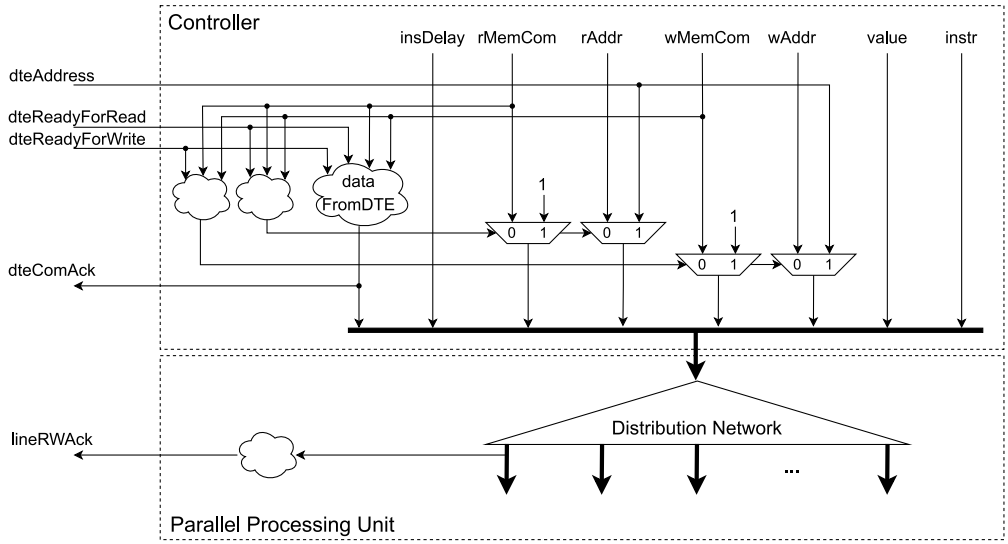

Figura 3.9 Arhitectura logicii de arbitrare a accesului la memoria internă de date

Pentru a începe un transfer de date între FIFO-urile de date și memoriile de date din Array, Host trebuie să transmită pe Calea de Program și Control o comandă TINRUN sau TOUTRUN, urmată de parametrii acesteia. Atât comenzile, cât și parametrii lor ocupă toți cei 32 de biți disponibili pe Calea de Program și Control.

Data Transfer Engine monitorizează ieșirea modulului Program FIFO și, când este detectată o comandă specifică, o salvează în Command FIFO, dacă acesta nu este *full*. Odată ajunsă la ieșirea Command FIFO, dacă Data Transfer Engine este în starea IDLE (nu este în curs niciun transfer), comanda și parametrii acesteia sunt citiți, actualizându-se modulele Command Register, Address Register, Line Counter și grupul Column Counter și Padding Counter, împreună cu registrele lor de reload

În cazul unui transfer de scriere, dacă datele sunt disponibile în Data Input FIFO, Data Transfer Engine începe să genereze comenzi de citire pentru FIFO și comenzi de deplasare pentru lanțul de registre I/O din Array. Comenzile de deplasare vor fi, de asemenea, generate pentru zerourile suplimentare atunci când numărul de coloane este mai mic decât numărul de celule din Array. Când o linie este complet deplasată prin lanțul de registre I/O și se află în poziția sa finală, modulul Data Transfer Engine trimite o cerere de scriere în memorie către Controler. Dacă cererea Data Transfer Engine este aprobată, acesta așteaptă ca scrierea să fie efectuată și apoi începe să transfere următoarea linie a matricei. Procesul se repetă până când toate liniile sunt transferate în memoria internă de date din Array.

În cazul unui transfer de citire, Data Transfer Engine trimite o cerere către Controler. Dacă cererea este aprobată, așteaptă ca citirea să fie efectuată și apoi transferă datele în Data Output FIFO. Când linia este complet deplasată, este trimisă o nouă solicitare de citire, iar procesul se repetă până când toate elementele matricei sunt citite din memoria internă. O comandă suplimentară, WAITRESREADY, menține Data Transfer Engine în starea IDLE până când Controlerul marchează un rezultat ca fiind disponibil.

Capitolul 4

Mediul de Programare bazat pe limbajul Python

Sistemul de calcul eterogen prezentat anterior a fost implementat pe Zynq-7020, integrat pe o placă PYNQ-Z2. Pentru a putea interacționa cu acesta, a fost proiectat și implementat un mediu de programare bazat pe Python [27], folosind pachetul software PYNQ [28], preinstalat pe Zynq.

Mai multe medii de programare similare pentru alte arhitecturi MapReduce au fost dezvoltate în trecut [29–31].

Mediul de programare rulează pe Host și permite accesul la resursele sistemului, gestionând programarea și transferurile I/O de date între sistemul Host și Acceleratorul MapReduce.

Acesta se bazează pe mai multe clase Python, organizate astfel încât să permită o adaptare rapidă atunci când se modifică arhitectura țintă: Clasa Instruction, utilizată pentru a genera codul binar al instrucțiunilor de nivel scăzut care vizează Acceleratorul, clasa Kernel, utilizată pentru a genera codul binar al kernel-urilor (pentru mediul curent, un kernel este definit ca un grup de instrucțiuni de nivel scăzut care îndeplinesc o anumită sarcină), clasa Library, utilizată pentru a genera codul binar al unei biblioteci, care este o colecție de kernel-uri și Clasa Machine, utilizată pentru a configura hardware-ul și a interacționa cu resursele Acceleratorului. Structura mediului de programare cu clasele sale principale este prezentată în Figura 4.2.

Utilizarea mediului de programare are două faze care corespund modului în care funcționează Acceleratorul MapReduce:

- Faza de pregătire: În această fază, FPGA este configurat cu sistemului eterogen implementat. Mai mult, alte elemente care permit accesul la resursele Acceleratorului sunt configurate acum: adresa de bază a Căii de Program și adresele tuturor registrele de control și stare ale Acceleratorului, numele registrelor, dimensiunile datelor și modulul DMA împreună cu canalele sale de date.

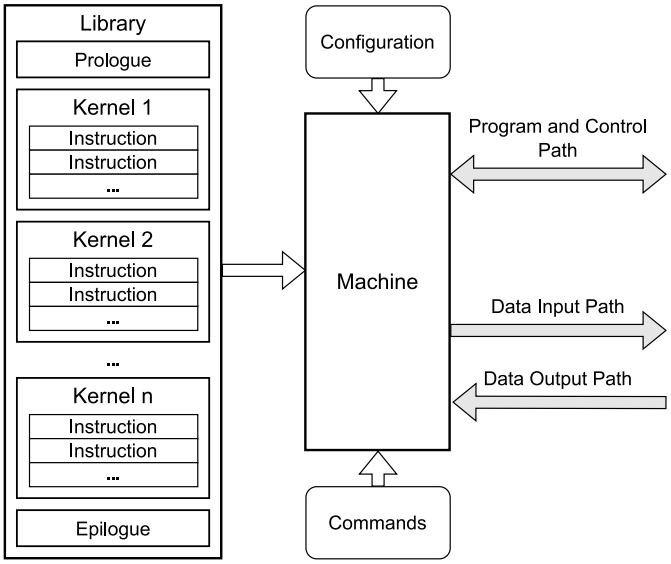

Figura 4.2 Structura mediului de programare bazat pe limbajul Python

- Faza de execuție: Înainte de a începe transmiterea de comenzi către Accelerator, o bibliotecă de kernel-uri trebuie să fie încărcată în memoria de program. După aceasta, procesorul Host va interacționa cu Acceleratorul prin trimiterea de comenzi de apelare a kernel-urilor sau de transfer I/O de date.

4.1 Elementul Instruction

Instrucțiunile sunt elementele de bază ale mediului de programare. O instrucțiune poate viza fie Controlerul, fie Unitatea de Procesare Paralelă. Pentru că la fiecare ciclu de ceas Acceleratorul are nevoie de două instrucțiuni, câte una pentru fiecare dintre cele două module, acestea trebuie asamblate în perechi de câte două. Excepție de la această regulă sunt instrucțiunile de salt și apel de program, care folosesc pentru adresă toți biții disponibili pe Calea de Program și Control.

4.2 Elementul Kernel

Pentru Acceleratorul MapReduce, un kernel reprezintă o secvență de instrucțiuni de nivel scăzut, utilizate pentru a realiza o anumită sarcină. Cu alte cuvinte, un kernel este o funcție scrisă într-un limbaj personalizat de nivel scăzut.

Deoarece Acceleratorul are nevoie de două instrucțiuni la fiecare pas de execuție, una pentru Controler și una pentru Unitatea de Procesare Paralelă, kernel-ul le grupează în perechi de două, formând o instrucțiune de Accelerator pe 32 de biți. Excepția este reprezentată de instrucțiunile pe 32 de biți, care sunt transmise singure. Pentru aceste instrucțiuni, Controlerul transmite intern instrucțiuni NOP către Unitatea de Procesare Paralelă.

Structura tipică a unui kernel este prezentată în Figura 4.4.

```
        cINSTR   aINSTR
        cINSTR   aINSTR
LB(n)   cINSTR   aINSTR
        cINSTR   aINSTR
        ...
        cINSTR   aINSTR
        JMP(n)
        HALT
```

Figura 4.4 Structura tipică a unui kernel

4.3 Elementul Library

O bibliotecă (library) este o colecție de kernel-uri încărcate împreună în memoria de program a Acceleratorului. La construirea bibliotecii, kernel-urile sunt extrase dintr-un fișier tip bibliotecă scris anterior și împachetate împreună.

Obiectul clasei Library este construit pe baza numelui fișierului bibliotecă, a adresei de pornire și a listei de kernel-uri care urmează să fie incluse în codul binar rezultat. Pe lângă kernel-urile definite de utilizator, fiecare bibliotecă are două kernel-uri predefinite numite Prolog și Epilog, care vor fi plasate la începutul, respectiv la sfârșitul bibliotecii.

4.4 Elementul Machine

Clasa Machine este elementul mediului de programare care asigură interacțiunea cu sistemul de calcul eterogen. Implementarea sa depinde de arhitectura sistemului și de resursele oferite de PYNQ [28].

Clasa Machine folosește un dicționar Python de configurare pentru a extrage informațiile necesare și a configura mediul. Resursele clasei Machine care mapează resursele sistemului de calcul eterogen sunt prezentate în Figura 4.6.

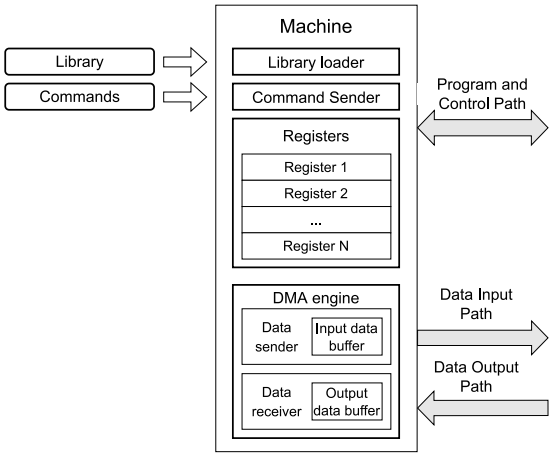

Figura 4.6 Resursele clasei Machine

Capitolul 5

Evaluarea Performanțelor Sistemului

5.1 Bibliotecă de Funcții Elementare de Algebră Liniară

Pentru a testa atât funcționarea corectă a Acceleratorului, cât și performanța acestuia, a fost implementată o bibliotecă de funcții elementare de algebră liniară. Biblioteca conține funcții matematice pentru operații cu matrice ce au un număr de coloane mai mic sau egal cu numărul de celule din Array, dar și alte funcții utilizate pentru a controla resursele Acceleratorului MapReduce [22]:

- Start cycle counter
  START_CC(): pornește contorul cycle counter din Controler.
- Stop cycle counter
  STOP_CC(): oprește contorul cycle counter din Controler.
- Trimite întrerupere
  SEND_INT(): setează fanionul de întrerupere al Acceleratorului. Acesta poate fi utilizat pentru a semnaliza sfârșitul unei secvențe de instrucțiuni.
- Operație Matrice-Matrice element cu element
  MM_EWO(*destination, source1, source2, linesNr, operation, waitMatricesNr*): efectuează o operație element cu element (ADD, SUB, MULT, AND, OR și XOR).
- Multiplicare Scalar-Matrice
  SM_MULT(*destination, scalar, source, linesNr, waitMatricesNr*): multiplică *scalar* cu fiecare element al matricei *source*.
- Multiplicare Matrice-Matrice
  MM_MULT(*destination, source1, source2, linesNr, waitMatricesNr*): multiplică matricea *source1* cu *source2*, considerând pe cea de-a doua deja transpusă.
- Multiplicare și acumulare Matrice-Matrice
  MM_MAC(*destination, source1, source2, linesNr, waitMatricesNr*): multiplică

matricele *source1* și *source2*, considerând pe cea de-a doua deja transpusă, iar apoi adună rezultatul cu *destination*.

Pe lângă biblioteca de kernel-uri descrisă mai sus, au fost implementate trei comenzi utilizate în transferul de date între FIFO-urile de date și Unitatea de Procesare Paralelă: WRITE_MATRIX, READ_MATRIX și WAIT_RES_READY.

5.2 Algoritmi de Evaluare

Pentru a evalua performanța Acceleratorului și îmbunătățirile aduse de Data Transfer Engine, au fost propuși patru algoritmi care extind pentru matrice mari operațiilor de algebră liniară descrise mai sus. O matrice mare este considerată a fi o matrice ale cărei dimensiuni sunt mai mari decât numărul de celule din Array (N).

Algoritmii se bazează pe faptul că o matrice mare poate fi împărțită în matrice mai mici ce sunt procesate, obținându-se astfel părți din matricea mare rezultat, care sunt apoi utilizate pentru a compune rezultatul final. Pentru simplitate, algoritmii consideră că toate matricele mari (A, B și R) sunt pătratice, cu dimensiuni egale cu $2^x \cdot N$, unde N este numărul de celule și x este un număr întreg. În acest caz particular, o matrice mare poate fi împărțită în n^2 matrice $N \times N$, unde $n = 2^x$.

- **Operații Matrice-Matrice element cu element pentru matrice mari**

Operațiile element cu element pentru matrice mari sunt definite de (5.5) și (5.6):

$$R_{ij} = A_{ij} o B_{ij}, \qquad i = 1...n,\ j = 1...n \tag{5.5}$$

$$\begin{bmatrix} R_{11} & R_{12} & ... & R_{1n} \\ R_{21} & R_{22} & ... & R_{2n} \\ ... & ... & ... & ... \\ R_{n1} & R_{n2} & ... & R_{nn} \end{bmatrix} = \begin{bmatrix} A_{11} o B_{11} & A_{12} o B_{12} & ... & A_{1n} o B_{1n} \\ A_{21} o B_{21} & A_{22} o B_{22} & ... & A_{2n} o B_{2n} \\ ... & ... & ... & ... \\ A_{n1} o B_{n1} & A_{n2} o B_{n2} & ... & A_{nn} o B_{nn} \end{bmatrix} \tag{5.6}$$

, unde R_{ij}, A_{ij} și B_{ij} sunt matrice $N \times N$, componente ale matricelor mari R, A și B.

- **Multiplicare Scalar-Matrice pentru matrice mari**

Multiplicarea Scalar-Matrice pentru matrice mari este definită de (5.7) and (5.8):

$$R_{ij} = s \cdot A_{ij}, \qquad i = 1...n,\ j = 1...n \tag{5.7}$$

$$\begin{bmatrix} R_{11} & R_{12} & ... & R_{1n} \\ R_{21} & R_{22} & ... & R_{2n} \\ ... & ... & ... & ... \\ R_{n1} & R_{n2} & ... & R_{nn} \end{bmatrix} = \begin{bmatrix} s \cdot A_{11} & s \cdot A_{12} & ... & s \cdot A_{1n} \\ s \cdot A_{21} & s \cdot A_{22} & ... & s \cdot A_{2n} \\ ... & ... & ... & ... \\ s \cdot A_{n1} & s \cdot A_{n2} & ... & s \cdot A_{nn} \end{bmatrix} \tag{5.8}$$

, unde R_{ij} și A_{ij} sunt matrice $N \times N$, componente ale matricelor mari R și A.

- **Multiplicare Matrice-Matrice pentru matrice mari**

Multiplicarea Matrice-Matrice pentru matrice mari este definită de (5.9) și (5.10). Pentru algoritmul curent, se consideră că transpusa matricei B (B^t) a fost deja calculată.

$$R_{ik} = \sum_{j=1}^{n} A_{ij} \times B^t_{kj}, \qquad i = 1...n, k = 1...n \tag{5.9}$$

$$\begin{bmatrix} R_{11} & ... \\ R_{21} & ... \\ ... & ... \\ R_{n1} & ... \end{bmatrix} = \begin{bmatrix} A_{11} \times B^t_{11} + A_{12} \times B^t_{12} + ... + A_{1n} \times B^t_{1n} & ... \\ A_{21} \times B^t_{11} + A_{22} \times B^t_{12} + ... + A_{2n} \times B^t_{1n} & ... \\ ... & ... \\ A_{n1} \times B^t_{11} + A_{n2} \times B^t_{12} + ... + A_{nn} \times B^t_{1n} & ... \end{bmatrix} \tag{5.10}$$

, unde R_{ij}, A_{ij} și B^t_{ij} sunt matrice $N \times N$, componente ale matricelor mari R, A și B^t. B^t este transpusa matricei B.

- **Multiplicare și acumulare Matrice-Matrice pentru matrice mari**

Multiplicarea și acumularea Matrice-Matrice pentru matrice mari este definită de (5.11) și (5.12). Pentru algoritmul curent, se consideră că transpusa matricei B (B^t) a fost deja calculată.

$$R_{ik} = R_{ik} + \sum_{j=1}^{n} A_{ij} \times B^t_{kj}, \qquad i = 1...n, k = 1...n \tag{5.11}$$

$$\begin{bmatrix} R_{11} & ... \\ R_{21} & ... \\ ... & ... \\ R_{n1} & ... \end{bmatrix} = \begin{bmatrix} R_{11} + A_{11} \times B^t_{11} + A_{12} \times B^t_{12} + ... + A_{1n} \times B^t_{1n} & ... \\ R_{21} + A_{21} \times B^t_{11} + A_{22} \times B^t_{12} + ... + A_{2n} \times B^t_{1n} & ... \\ ... & ... \\ R_{n1} + A_{n1} \times B^t_{11} + A_{n2} \times B^t_{12} + ... + A_{nn} \times B^t_{1n} & ... \end{bmatrix} \tag{5.12}$$

, unde R_{ij}, A_{ij} și B^t_{ij} sunt matrice $N \times N$, componente ale matricelor mari R, A și B^t. B^t este transpusa matricei B.

5.3 Rezultate ale Evaluării Performanței

Performanța Acceleratorului a fost analizată atât din perspectiva timpului de execuție, cât și a puterii consumate. Pentru aceasta, au fost create medii de testare pentru simulare și măsurători hardware. Mediul de simulare a fost folosit pentru a evalua timpul de execuție exprimat ca număr de cicluri de ceas. Mediul hardware a fost folosit pentru a testa corectitudinea funcționării sistemului și, de asemenea, pentru a determina consumul de energie.

Pentru a evidenția impactul asupra timpului de execuție și consumului de energie al îmbunătățirilor aduse transferului I/O de date, au fost analizate trei versiuni ale Acceleratorului:

- MRA V0: Versiunea inițială a Acceleratorului MapReduce, fără nicio îmbunătățire (așa cum este descrisă în Capitolul 2). În această versiune, Controlerul este responsabil pentru transferurile I/O de date, iar celulele sunt organizate pe o singură cale de propagare.

- MRA V1: O versiune îmbunătățită a Acceleratorului MapReduce având celulele reorganizate pentru a elimina întârzierea introdusă de propagarea în doi pași. În această versiune, Controlerul este încă responsabil pentru transferurile I/O de date.

- MRA V2: O versiune îmbunătățită a Acceleratorului MapReduce, în care au fost implementate ambele îmbunătățiri descrise în Capitolul 3: celulele sunt reorganizate ca în MRA V1, iar Data Transfer Engine este responsabil pentru transferurile I/O de date.

5.3.1 Analiza Timpului de Execuție

Pentru a analiza variația timpului de execuție în funcție de resursele de calcul ale Acceleratorului pentru algoritmii propuși în secțiunea anterioară, dar și pentru a testa corectitudinea funcționării acestuia, a fost creat un mediu de simulare capabil să execute mai multe scenarii de test.

Au fost utilizate două scenarii de test pentru a evalua variația timpului de execuție în funcție de resursele disponibile și volumul de calcul necesar, care depinde de dimensiunea matricelor. În primul scenariu, o operație asupra unor matrice 16×16, 32×32, 64×64 și 128×128 a fost executată pe un Acelerator MapReduce cu 16 celule de procesare. Acest scenariu de test va arăta cum evoluează timpul de execuție pe măsură ce volumul de calcul crește prin creșterea dimensiunii matricelor, dacă se folosesc aceleași resurse de calcul. În al doilea scenariu de test, o operație asupra unor matrice 128×128 este executată pe Acceleratoare MapReduce cu 16, 32, 64 și 128 de celule de procesare. Acest scenariu de test va arăta cum evoluează timpul de execuție pe măsură ce resursele de calcul cresc, în timp ce volumul necesar de calcul este constant.

Cele două scenarii de test au fost aplicate pentru operațiile de Adunare Matrice-Matrice (Matrix-Matrix ADD), Multiplicare Scalar-Matrice (Scalar-Matrix MULT), Multiplicare Matrice-Matrice (Matrix-Matrix MULT) și Multiplicare și Acumulare Matrice-Matrice (Matrix-Matrix MAC), executate pe toate cele trei versiuni ale Acceleratorului MapReduce prezentate anterior.

Pentru a avea o imagine clară asupra performanței Acceleratorului, punctul de plecare pentru măsurarea numărului de cicluri de ceas a fost considerat a fi momentul în care biblioteca este deja încărcată în memoria de program și primele date ajung la intrarea Acceleratorului.

Rezultatele obţinute pentru operaţia Adunare Matrice-Matrice pentru cele două scenarii de test sunt prezentate în Figura 5.7 şi Figura 5.8.

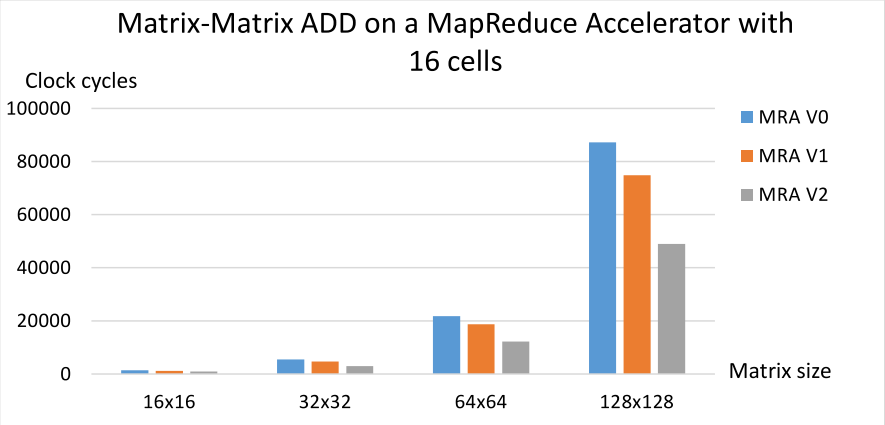

Figura 5.7 Adunare Matrice-Matrice pe un Accelerator cu 16 celule de procesare

În Figura 5.7, se poate observa că timpul de execuție este proporțional cu numărul de elemente din matrice pentru toate cele trei versiuni ale Acceleratorului: crește de 4× când numărul elementelor crește de 4×. Mai mult, se poate observa că cea mai bună îmbunătățire este adusă de versiunea care conține Data Transfer Engine (MRA V2), pentru care timpul de execuție scade cu aproximativ 44% pentru toate operațiile cu matrice mari incluse în scenariul de test.

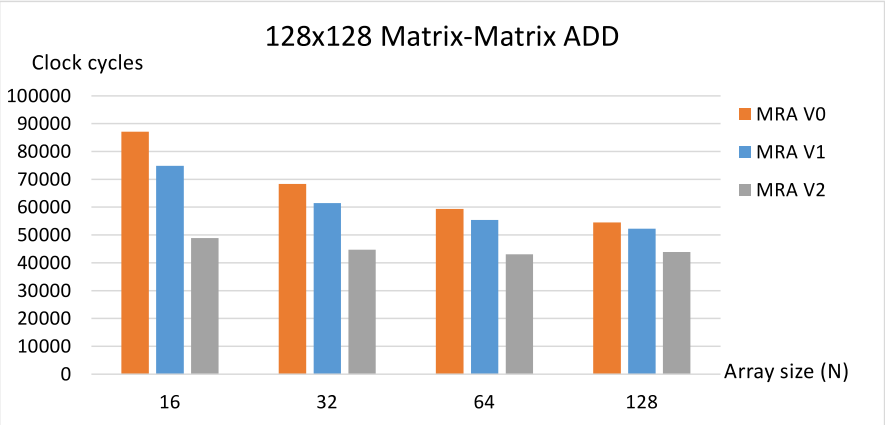

Figura 5.8 Adunare Matrice-Matrice 128×128 pe Acceleratoare cu diferite dimensiuni

În Figura 5.8 se poate observa influența resurselor de calcul asupra operației de adunare a două matrice 128×128. Acest scenariu de test relevă, de asemenea, îmbunătățirea adusă de prezența Data Transfer Engine asupra timpului total de execuție, acesta fiind mai mare pe măsură ce resursele de calcul scad: o îmbunătățire de aproximativ 43% la adunarea a două matrice 128×128 pe un Accelerator cu 16 celule de procesare. În plus, se poate observa că prezența Data Transfer Engine reduce considerabil diferența dintre timpii de execuție ai aceleiași operații pe Acceleratoare cu număr diferit de celule de procesare.

Rezultatele obținute pentru operația Multiplicare Scalar-Matrice pentru cele două scenarii de test sunt prezentate în Figura 5.9 și Figura 5.10. Pentru această operație, trebuie transferată o singură matrice din memoria principală în Accelerator.

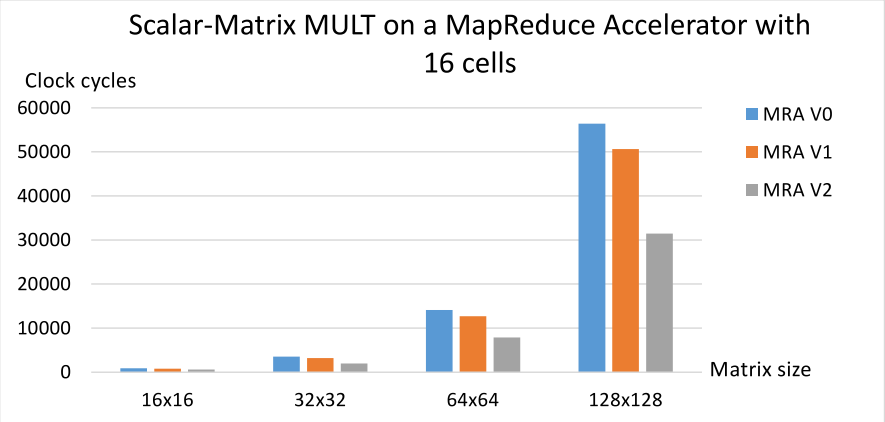

Figura 5.9 Multiplicare Scalar-Matrice pe un Accelerator cu 16 celule de procesare

În Figura 5.9, se poate observa că, la fel ca în cazul operației anterioare, cea mai importantă îmbunătățire este adusă de versiunea care conține Data Transfer Engine (MRA V2), pentru care timpul de execuție scade cu aproximativ 44% pentru toate operațiunile cu matrice mari incluse în scenariul de test. Totuși, deoarece operațiunea necesită o singură matrice ca operand, timpul total de execuție este mai mic decât cel obținut pentru operația Adunare Matrice-Matrice (de exemplu, timpul de execuție este mai mic cu 35% când se utilizează MRA V2, comparativ cu la operația Adunare Matrice-Matrice pe aceeași arhitectură).

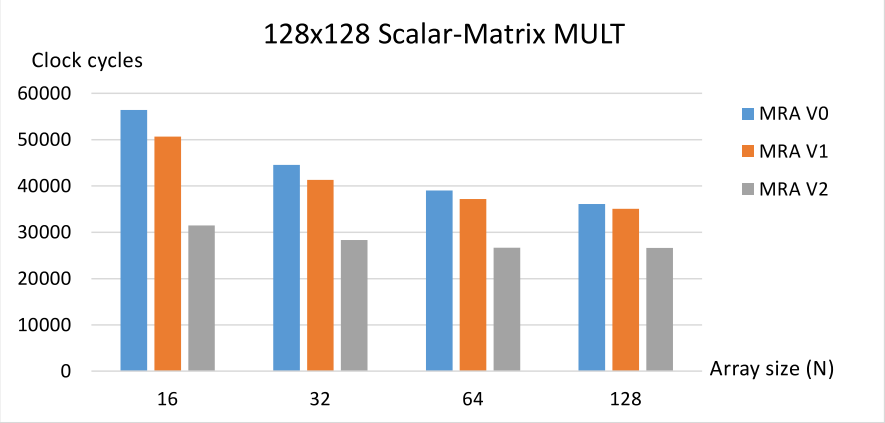

Figura 5.10 Multiplicare Scalar-Matrice 128×128 pe Acceleratoare cu diferite dimensiuni

În Figura 5.10 se poate observa influența resurselor de calcul asupra operației de multiplicare a fiecărui element dintr-o matrice 128×128 cu un scalar. Ca și în cazul operației anterioare, îmbunătățirea adusă de prezența Data Transfer Engine combinată cu reorganizarea celulelor asupra timpului total de execuție este mai mare pe măsură ce resursele de calcul scad: o îmbunătățire de aproximativ 44% la efectuarea operaţiei pe un Accelerator cu 16 celule de procesare. Mai mult, se mai poate observa că prezența Data Transfer Engine reduce considerabil diferența dintre timpii de execuție ai aceleiași operații pe Acceleratoare cu număr diferit de celule de procesare, atenuând impactul asupra timpului de execuție atunci când resursele de calcul sunt reduse.

Rezultatele obținute pentru operația Multiplicare Matrice-Matrice pentru cele două scenarii de test sunt prezentate în Figura 5.11 și Figura 5.12. Față de operațiile analizate anterior se observă o creștere importantă a timpului de execuție. Acest lucru se explică prin creșterea semnificativă a numărului de transferuri I/O de date, determinată de faptul că o matrice $N \times N$ trebuie transferată de mai multe ori din memoria principală în memoria internă a modulului Array în timpul execuției algoritmului.

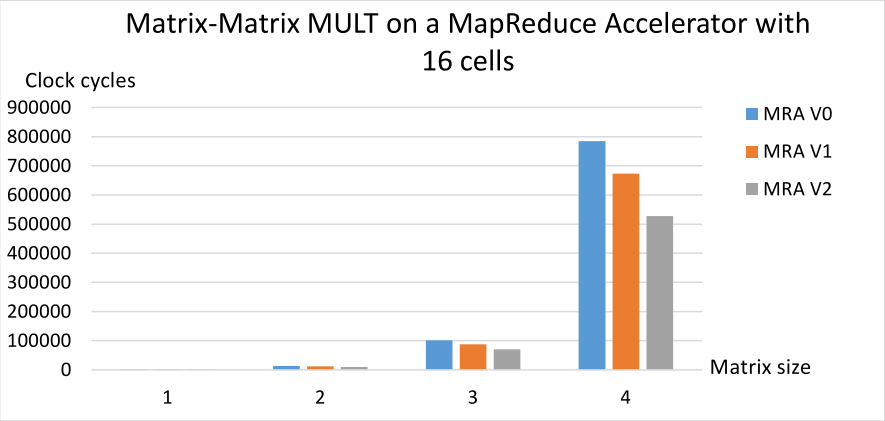

Figura 5.11 Multiplicare Matrice-Matrice pe un Accelerator cu 16 celule de procesare

În Figura 5.11, se poate observa că îmbunătățirea timpului de execuție crește odată cu creșterea numărului de elemente din matrice. Îmbunătățirea mai mare (de aproximativ 32%) se obține atunci când se înmulțesc două matrice 128×128, prin urmare, atunci când numărul de transferuri de date necesare este cel mai mare. Valoarea mai mică a îmbunătățirii timpului de execuție față de cele obținute pentru algoritmii anteriori poate fi explicată prin faptul că, în timpul înmulțirii a două matrice, memoria internă este accesată mai des și solicitările cu prioritate mai mică de la Date Transfer Engine sunt acceptate cu întârziere.

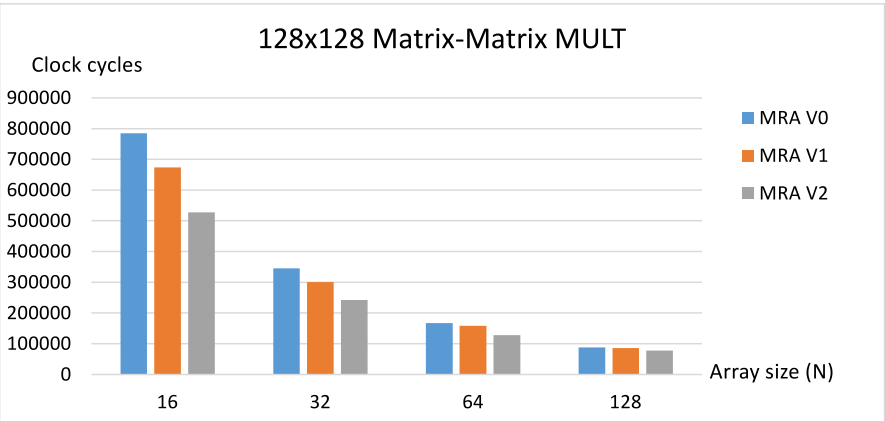

Figura 5.12 Multiplicare Matrice-Matrice 128x128 pe Acceleratoare cu diferite dimensiuni

Influența resurselor de calcul asupra operației de multiplicare a două matrice 128×128 este prezentată în Figura 5.12. Versiunea MRA V2 prezintă îmbunătățiri ale timpilor de

execuție pentru toate variațiile scenariului de test, cea mai mare obținându-se în cazul înmulțirii matricelor pe un Accelerator cu 16 celule de procesare: aproximativ 32%. Cu toate acestea, rezultatele sugerează că rata de creștere a îmbunătățirii tinde să scadă pe măsură ce resursele de calcul scad.

Rezultatele obținute pentru operația Multiplicare și Acumulare Matrice-Matrice pentru cele două scenarii de test sunt prezentate în Figura 5.13 și Figura 5.14. Față de operatia de înmulțire se poate observa o ușoara creștere a timpului de execuție.

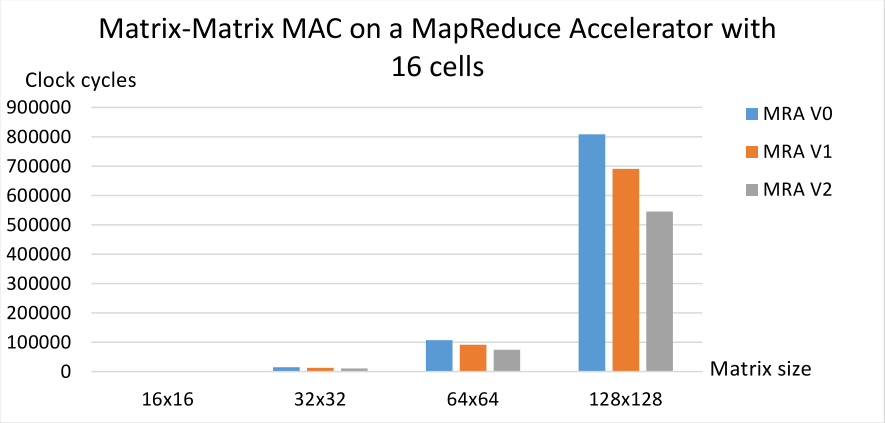

Figura 5.13 Multiplicare și Acumulare Matrice-Matrice pe un Accelerator cu 16 celule de procesare

În Figura 5.13 se poate observa că evoluţia timpului de execuţie în funcţie de mărimea matricelor este similară cu cea obţinută în cazul multiplicării. Ca și în cazul precedent, cea mai mare îmbunătățire (de aproximativ 32%) se obține la multiplicarea și acumularea a două matrice de 128×128. În concluzie, îmbunătățirea crește pe măsură ce crește numărul de transferuri de date între memoria principală și Accelerator.

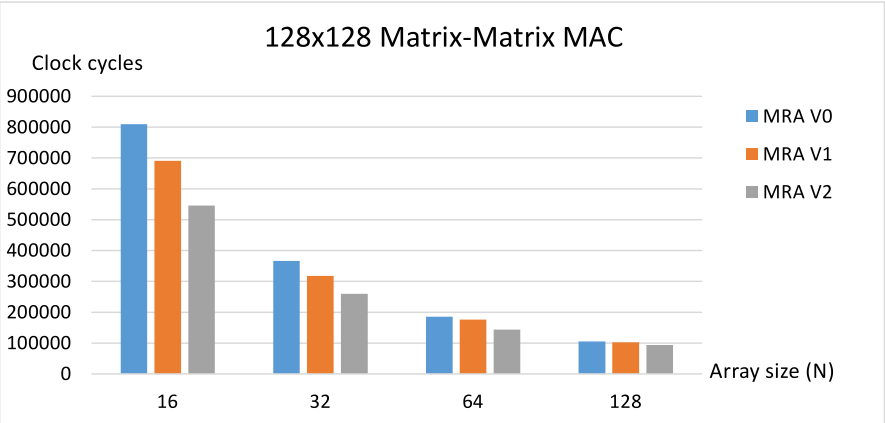

Figura 5.14 Multiplicare și Acumulare Matrice-Matrice 128x128 pe Acceleratoare cu diferite dimensiuni

Influența resurselor de calcul asupra operației de înmulțire și acumulare a două matrice 128×128 este prezentată în Figura 5.14. Rezultatele confirmă influența pozitivă a Data Transfer Engine asupra timpului de execuție dar și scăderea ratei de îmbunătățire odată cu creșterea volumului de date transferate între memoria principală și Accelerator.

5.3.2 Analiza Implementării Hardware

Pe lângă mediul de simulare, a fost realizat și un mediu de test hardware pentru a testa corectitudinea funcționării sistemului și, de asemenea, pentru a evidenția impactul platformei de implementare (adică, a plăcii PYNQ-Z2 și a pachetului software PYNQ) asupra timpului de execuție. Datorită limitărilor date de resursele FPGA-ului integrat în ZYNQ-7020, au putut fi sintetizate Acceleratoare cu 16, 32 și 64 de celule de procesare și au fost alese pentru testare matrice 64×64.

Corectitudinea funcționării a fost testată prin compararea rezultatelor returnate de Accelerator cu cele calculate de către Host folosind biblioteca Python *numpy*.

Pentru a evidenția influența sistemului, au fost măsurați pentru fiecare dintre cele trei versiuni ale sistemului (MRA V0, MRA V1 și MRA V2) timpii de execuție a doi algoritmi de procesare de matrice mari. Rezultatele obținute au fost similare pentru aceeași combinație de dimensiuni ale matricei și număr de celule pentru toate cele trei versiuni ale Acceleratorului MapReduce. Acest lucru poate fi explicat prin întârzierile mari introduse de interpretarea codului scris în Python și de accesarea resurselor din clasele mediului de programare.

Intensitatea curentului și tensiunea au fost măsurate în timpul execuției celor doi algoritmi pentru toate cele trei versiuni ale Acceleratorului, dar nu au fost observate diferențe notabile. Acest lucru poate fi explicat prin ponderea semnificativă pe care o are consumul de putere statică al plăcii PYNQ-Z2 în consumul total de energie, mascând astfel diferențele date de variația circuitului implementat pe FPGA. Folosind timpul de execuție obținut în simulare și valorile intensității curentului și ale tensiunii măsurate, a fost calculată o estimare a consumului de energie pentru MRA_V2 cu diferite dimensiuni (Tabel 5.10).

Tabel 5.10 Estimarea consumului de energie pentru adunarea și multiplicarea unor matrice 64×64

Accelerator	**Matrix-Matrix ADD**				**Matrix-Matrix MULT**			
	U [V]	**I [A]**	Δt_{sim} **[μs]**	**E [μJ]**	**U [V]**	**I [A]**	Δt_{sim} **[μs]**	**E [μJ]**
MRA V2 16	4.96	0.35	122	212	4.96	0.36	702	1253
MRA V2 32	4.96	0.37	112	206	4.96	0.39	344	665
MRA V2 64	4.96	0.37	116	213	4.96	0.39	200	387

După cum se observă în tabelul anterior, cea mai mare influență asupra energiei consumate este timpul de execuție, ceea ce înseamnă că, cu cât dimensiunea Acceleratorului este mai mică, operația cu matrice mari durează mai mult, iar consumul este mai mare. Cu toate acestea, folosind configurația actuală, cu multe alte elemente care consumă energie, nu poate fi trasă o concluzie definitivă cu privire la influența dimensiunii Acceleratorului asupra energiei totale consumate. Cu toate acestea, este de așteptat ca implementarea sistemului de calcul eterogen pe siliciu folosind un nod tehnologic avansat să aibă ca rezultat un consum de energie mult mai mic.

Capitolul 6

Concluzii

6.1 Obiective și rezultate

Cercetarea a fost dedicată dezvoltării unui sistem de calcul eterogen capabil să proceseze eficient date în paralel. Pentru dezvoltarea unui sistem complet funcțional au fost luate în considerare trei elemente principale: proiectarea și implementarea unei noi arhitecturi pentru un Accelerator MapReduce având un transfer I/O de date eficient, proiectarea arhitecturii unui sistem de calcul eterogen care integrează Acceleratorul și implementarea acestuia pe platforma suport, precum și dezvoltarea unor medii de testare prin simulare și prin interacțiunea cu implementarea hardware.

Dezvoltarea unei noi arhitecturi MapReduce a fost realizată pe baza unui Accelerator deja existent, care a fost analizat pentru a evidenția punctele sale slabe ce au un impact negativ asupra timpului total de execuție al unei secvențe de program. În urma acestei analize s-a constatat că transferul I/O de date reprezintă o parte importantă din timpul total de execuție, devenind astfel ținta eforturilor de îmbunătățire. Principiul care stă la baza noii arhitecturi este separarea fluxurilor de transfer I/O de date și de procesare, astfel încât transferurile de date necesare prelucrărilor viitoare să poată fi efectuate în prealabil, în paralel cu secvența actuală de prelucrare a datelor. Mai mult, celulele de procesare din unitatea responsabilă cu prelucrarea paralelă a datelor au fost reorganizate astfel încât Acceleratorul să poată accepta date de intrare noi la fiecare ciclu de ceas.

Astfel, au fost implementate două noi arhitecturi, fiecare integrând progresiv măsurile de îmbunătățire propuse.

Pentru evaluarea performanței, a fost dezvoltat un mediu de simulare, care să permită utilizatorului să scrie programe în limbajul de asamblare specific Acceleratorului, să interacționeze cu acesta și să evalueze rezultatele obținute.

Pentru a evidenția îmbunătățirea performanței datorită transferului I/O de date îmbunătățit, au fost propuși patru algoritmi de evaluare pentru matrice mari: Matrix-Matrix ADD, Scalar-Matrix MULT, Matrix-Matrix MULT și Matrix-Matrix MAC. Atunci când se lucrează cu matrice mari, numărul de coloane este mai mare decât

numărul de celule de procesare, depășind astfel capacitățile Acceleratorului de stocare și prelucrare a datelor. Acest lucru forțează Acceleratorul să efectueze transferuri I/O de date succesive pentru a aduce părți ale matricele operanzi și pentru a trimite înapoi părți ale matricei rezultat.

Testele au fost folosite pentru a caracteriza dependența timpului de execuție de resursele Acceleratorului și complexitatea algoritmului. Rezultatele au arătat o îmbunătățire a timpului de execuție pentru ambele arhitecturi noi față de cea inițială. Îmbunătățirea mai mare a fost obținută pentru arhitectura care implementează atât reorganizarea celulelor de procesare, cât și separarea fluxului de transfer I/O de date de cel de procesare: o îmbunătățire de până la 44% pentru Matrix-Matrix ADD și Scalar-Matrix MULT și până la 32% pentru Matrix-Matrix MULT și Matrix-Matrix MAC. Valoarea mai mică a îmbunătățirii timpului de execuție obținută pentru ultimele două operații față de primele două poate fi explicată prin faptul că, în timpul înmulțirii matricelor, memoria internă a Acceleratorului este accesată mai des de către fluxul de procesare, iar cererile din fluxul de transfer I/O de date sunt acceptate cu întârziere, reducând astfel capacitatea de a transfera date noi.

Mai mult, rezultatele testelor au arătat că, pentru noile arhitecturi, îmbunătățirea timpului de execuție față de cea inițială este mai mare pe măsură ce dimensiunile matricelor ce trebuie prelucrate cresc, în comparație cu numărul de celule de procesare din Accelerator. Cu alte cuvinte, noua arhitectură își arată utilitatea cu cât volumul de date ce urmează a fi procesate de către Accelerator este mai mare.

Pentru a putea folosi resursele Acceleratorului, acesta a fost integrat într-un sistem de calcul eterogen. Într-un astfel de sistem, programul principal este executat pe un procesor de uz general numit Host, în timp ce Acceleratorul este responsabil pentru execuția secvențelor de calcul intensiv. Arhitectura propusă a sistemului ia în considerare modul în care Acceleratorul trebuie să interacționeze cu procesorul Host și memoria principală pentru a funcționa cât mai eficient posibil. Mai mult, arhitectura sistemului a fost adaptată la resursele oferite de platforma pe care a fost implementată (Zynq-7020 Soc).

De asemenea, a fost dezvoltat și un mediu de programare bazat pe limbajul Python pentru a permite utilizatorului să interacționeze cu sistemul de calcul eterogen implementat pe Zynq-7020. Acest mediu, ca și cel dezvoltat pentru simulare, permite descrierea unei biblioteci de funcții în limbajul de asamblare specific, dezvoltarea programelor și executarea acestora pe Accelerator. Cu toate acestea, deși este ușor de utilizat și permite dezvoltarea și testarea rapidă a programelor, a fost observată o întârziere importantă introdusă de nivelurile software ale mediului, făcându-l neadecvat pentru utilizare intensivă.

În plus, a fost dezvoltat și un mediu de măsurare a consumului de energie, realizându-se o evaluare preliminară a energiei necesare sistemului pentru a efectua diferite operații cu matrice. Deși influența numărului de celule de procesare din Accelerator asupra

cantității totale de energie consumată în timpul unei operații cu matrice nu a putut fi evidențiată din cauza influenței majore a altor componente ale platformei pe care a fost implementat sistemul de calcul eterogen, s-a putut realiza totuși o evaluare preliminară a acesteia.

Ca o concluzie generală, a fost dezvoltat și îmbunătățit un sistem de calcul eterogen complet funcțional, care, supus îmbunătățirii ulterioare și implementat pe o platformă adecvată, poate oferi performanțe bune, devenind competitiv cu soluțiile existente deja pe piață.

6.2 Contribuții originale

Principalele contribuții originale ale acestei cercetări pot fi clasificate după cum urmează, pe baza celor trei componente principale ale sistemului dezvoltat:

Arhitectura Acceleratorului MapReduce

- Analiza arhitecturii unui Accelerator MapReduce existent și propunerea unei noi arhitecturi pentru îmbunătățirea timpului total de execuție. Deciziile privind elementele care trebuie îmbunătățite au fost luate pe baza studiului literaturii și a mediului de testare, prin care au fost analizate execuțiile mai multor algoritmi pentru a evidenția punctele slabe ale Acceleratorului.

- Proiectarea și implementarea unui nou mod de organizare a celulelor de procesare în Accelerator, astfel încât propagarea datelor I/O să fie mai eficientă, fără a modifica comportamentul funcțional (această modificare este complet transparentă pentru utilizator).

- Proiectarea și implementarea unui modul de control numit Data Transfer Engine, care este responsabil pentru coordonarea transferurilor I/O de date. Acest lucru, împreună cu separarea resurselor utilizate în transferurile I/O de date, a condus la separarea fluxurilor de transfer de date și procesare.

- Proiectarea și implementarea unei proceduri de sincronizare între fluxul de transfer I/O de date și cel de procesare.

- Proiectarea și implementarea structurilor responsabile cu arbitrarea cererilor de accesare a memoriei interne din Array, provenite de la cele două fluxuri.

Arhitectura sistemului

- Proiectarea arhitecturii unui sistem de calcul eterogen care integrează Acceleratorul MapReduce și implementarea acestuia având în vedere resursele disponibile pe Zynq-7020, astfel încât interacțiunea dintre componentele sale să fie cât mai eficientă.
- Proiectarea și implementarea unei structuri care să interfațeze Acceleratorul MapReduce cu restul sistemului, și prin care acesta să poată fi configurat și utilizat.

Componenta software

- Implementarea unei biblioteci elementare de algebră liniară folosind limbajul de asamblare specific Acceleratorului, pentru realizarea de operații cu matrice.
- Propunerea unei forme de segmentare pentru memoria internă a Array-ului astfel încât capabilitățile de transfer de date ale Data Transfer Engine să fie utilizate cât mai eficient posibil și adaptarea la resursele specifice Acceleratorului a algoritmilor pentru operații cu matrice mari.
- Implementarea unui mediu de programare bazat pe limbajul Python, care susține dezvoltarea și execuția de programe software specifice sistemului de calcul eterogen.

6.3 Lista lucrărilor originale

1. **George-Vlăduț Popescu**, Improvements in Data Transfer for a MapReduce Accelerator, *Romanian Journal of Information Science and Technology*, 25(3-4), 2022 [22].
2. **George-Vlăduț Popescu**, Călin Bîră, Python-Based Programming Framework for a Heterogeneous MapReduce Architecture, *2022 14th International Conference on Communications (COMM)*, București, Iunie 2022, pp. 1-6, DOI: 10.1109/COMM54429.2022.9817183 [27].
3. Mihaela Maliţa, **George-Vlăduţ Popescu**, Gheorghe M. Ştefan, Pseudo Reconfigurable Heterogeneous Solution for Accelerating Spectral Clustering, *2020 IEEE International Conference on Big Data (Big Data)*, Decembrie 2020, pp. 5138-5145, DOI: 10.1109/BigData50022.2020.9378150, WOS:000662554705026 [24].
4. Mihaela Maliţa, **George-Vlăduţ Popescu**, Gheorghe M. Ştefan, Heterogeneous Computing System for Deep Learning. In *Deep Learning: Concepts and Architectures*, Pedrycz, W., Chen, S.M., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 287–319, ISBN: 978-3-030-31756-0, DOI: 10.1007/978-3-030-31756-0_10 [16].

5. Mihaela Maliţa, **George-Vlăduţ Popescu**, Gheorghe M. Ştefan, Heterogenous Computing for Markov Models in Big Data, *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA, Decembrie 2019, pp. 1500-1505, DOI: 10.1109/CSCI49370.2019.00279, WOS: 000569996300272 [23].

6. Alexandru Gheolbănoiu, **George-Vlăduț Popescu**, Radu Hobincu, Lucian Petrică, A Software-Defined FPGA Vector Processor with Application-Aware Reconfiguration, *University Politehnica of Bucharest Scientific Bulletin, Series C: Electrical Engineering and Computer Science*, 78(4), pp.43-56, 2016, ISSN 2286-3540, WOS:000393328400004 [18].

7. Mihaela Malița, **George Vlăduț-Popescu**, Gheorghe M. Ștefan, Hybrid System for Deep Learning, *ICON4N 2018: 1st International Conference on Neuroscience, Neuroinformatics, Neurotechnology and Neuro-Psycho-Pharmacology*, București, România, Noiembrie 15-18, 2018, [nepublicat].

8. **George Vlăduț-Popescu**, State of the Art in sound source recognition using neural networks, Raport Științific Nr. 1, Universitatea Politehnica din București, 2018.

9. **George Vlăduț-Popescu**, Computational components of Deep Neural Networks used in sound space investigation, Raport Științific Nr. 2, Universitatea Politehnica din București, 2018.

10. **George Vlăduț-Popescu**, Computer architectures for Deep Neural Networks, Raport Științific Nr. 3, Universitatea Politehnica din București, 2019.

6.4 Perspective de Dezvoltare Ulterioară

Pentru a dezvolta capabilitățile sistemului de calcul eterogen propus și pentru a îmbunătăți performanța acestuia astfel încât să devină competitiv cu celelalte sisteme aflate în prezent pe piață, trebuie făcute investigații și îmbunătățiri ulterioare ale tuturor componentele sale principale: arhitectura Acceleratorului, arhitectura sistemului și mediul software utilizat pentru a accesa resursele acestuia.

Arhitectura Acceleratorului MapReduce poate fi dezvoltată în continuare pentru a îmbunătăți atât capabilitățile de calcul, cât și transferul I/O de date.

Pentru a îmbunătăți capabilitățile de calcul, pot fi adăugate noi module care permit operații noi, extinzând astfel domeniul aplicațiilor țintă. De exemplu, poate fi implementat calculul în virgulă mobilă, fie prin module specializate în fiecare celulă de procesare, fie prin celule de procesare complet separate.

În plus, transferul I/O de date poate fi îmbunătățit în continuare prin implementarea de structuri care permit atât transferurile de citire, cât și de scriere în același timp. O

soluție pentru a realiza acest lucru este separarea fluxurilor de date de intrare și de ieșire în cadrul matricei de celule de procesare prin implementarea a două lanțuri de registre I/O cu secțiuni de control separate, câte unul pentru fiecare tip de transfer de date. O altă soluție ar putea fi implementarea transferurilor de citire-scriere, care încarcă mai întâi datele citite în registrele I/O și apoi le transferă la ieșire în timp ce noi date sunt transferate la intrarea lanțului de registre I/O.

La nivel de sistem, pot fi adaugate mai multe Acceleratoare și grupate apoi în clustere, fie identice, fie cu capabilități de calcul diferite, dar folosind aceleași căi de date și de program. În acest fel, capacitatea sistemului de a procesa date este imbunătățîtă considerabil. Pentru un astfel de sistem, este necesar să se investigheze numărul optim de celule de procesare dintr-un Accelerator și câte Acceleratoare ar trebui grupate împreună pentru a obține cel mai bun raport *număr de operații / putere consumată*.

La nivel software, deși mediul de programare propus este ușor de utilizat și oferă un ciclu scurt de dezvoltare pentru algoritmii care folosesc date organizate ca vectori și matrice, acesta introduce întârzieri mari, reducând performanța sistemului. Astfel, o îmbunătățire a interacțiunii cu sistemul ar putea fi dezvoltarea unui mediu care utilizează drivere scrise in limbaj de nivel scăzut pentru a controla resursele sistemului de calcul eterogen. Mai mult, biblioteca actuală de funcții poate fi îmbunătățită prin includerea de funcții suplimentare, permițând ca Acceleratorul să fie folosit și pentru alte aplicații.

Bibliografie

[1] Krste Asanovic et al., *The Landscape of Parallel Computing Research: A View from Berkeley*, rap. teh. UCB/EECS-2006-183, Electrical Engineering și Computer Sciences, University of California at Berkeley, Dec. 2006.

[2] Krste Asanovic et al., "A View of the Parallel Computing Landscape", în *Commun. ACM* 52.10 (Oct. 2009), pp. 56–67, ISSN: 0001-0782, DOI: 10.1145/1562764.1562783.

[3] Alejandro Duran și Michael Klemm, "The Intel® Many Integrated Core Architecture", în *2012 International Conference on High Performance Computing & Simulation (HPCS)*, 2012, pp. 365–366, DOI: 10.1109/HPCSim.2012.6266938.

[4] George Chrysos, "Intel® Xeon Phi coprocessor (codename Knights Corner)", în *2012 IEEE Hot Chips 24 Symposium (HCS)*, 2012, pp. 1–31, DOI: 10.1109/HOTCHIPS.2012.7476487.

[5] *Intel® Xeon Phi™ Coprocessor System Software Developers Guide*, https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/xeon-phi-coprocessor-system-software-developers-guide.pdf, California, USA: Intel Corporation, 2014.

[6] Evangelos Georganas et al., "Anatomy of High-Performance Deep Learning Convolutions on SIMD Architectures", în *SC18: International Conference for High Performance Computing, Networking, Storage and Analysis*, 2018, pp. 830–841, DOI: 10.1109/SC.2018.00069.

[7] *NVIDIA TESLA V100 GPU Architecture*, https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf, California, USA: NVIDIA Corporation, 2017.

[8] *NVIDIA TURING GPU Architecture*, https://images.nvidia.com/aem-dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf, California, USA: NVIDIA Corporation, 2018.

[9] *NVIDIA A100 Tensor Core GPU Architecture*, https://images.nvidia.com/aem-dam/en-zz/Solutions/data-center/nvidia-ampere-architecture-whitepaper.pdf, California, USA: NVIDIA Corporation, 2020.

[10] *NVIDIA H100 Tensor Core GPU Architecture*, https://resources.nvidia.com/en-us-tensor-core, California, USA: NVIDIA Corporation, 2022.

[11] Norman P. Jouppi et al., "A Domain-Specific Supercomputer for Training Deep Neural Networks", în *Commun. ACM* 63.7 (Iun. 2020), pp. 67–78, ISSN: 0001-0782, DOI: 10.1145/3360307.

[12] John L. Hennessy și David A. Patterson, *Computer Architecture, Sixth Edition: A Quantitative Approach*, 6th, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017, ISBN: 978-0-12-811905-1.

[13] Norman P. Jouppi et al., "In-Datacenter Performance Analysis of a Tensor Processing Unit", în *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, Toronto, ON, Canada: Association for Computing Machinery, 2017, pp. 1–12, ISBN: 9781450348928, DOI: 10.1145/3079856.3080246.

[14] Sharan Chetlur et al., "cuDNN: Efficient Primitives for Deep Learning", în *CoRR* abs/1410.0759 (2014), arXiv: 1410.0759, URL: http://arxiv.org/abs/1410.0759.

[15] Chao Li et al., "Optimizing Memory Efficiency for Deep Convolutional Neural Networks on GPUs", în *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2016, pp. 633–644, DOI: 10.1109/SC.2016.53.

[16] Mihaela Malița, George Vlăduț Popescu și Gheorghe M. Ștefan, "Heterogeneous Computing System for Deep Learning", în *Deep Learning: Concepts and Architectures*, ed. de Witold Pedrycz și Shyi-Ming Chen, Cham: Springer International Publishing, 2020, pp. 287–319, ISBN: 978-3-030-31756-0, DOI: 10.1007/978-3-030-31756-0_10.

[17] Gheorghe M. Ștefan, "Pseudo-Reconfigurable Computing", în *Romanian Journal of Information Science and Technology (ROMJIST)* 24.4 (2021), pp. 366–383, ISSN: 1453-8245.

[18] Alexandru Gheolbănoiu et al., "A Software-Defined FPGA Vector Processor with Application-Aware Reconfiguration", în *University Politehnica of Bucharest Scientific Bulletin Seris C* 78.4 (2016), pp. 43–56, ISSN: 2286-3540.

[19] Lazar Bivolarski et al., "The CA1024: A fully programmable system-on-chip for costeffective HDTV media processing", în *2006 IEEE Hot Chips 18 Symposium (HCS)*, 2006, pp. 1–26, DOI: 10.1109/HOTCHIPS.2006.7477854.

[20] Mihaela Malița, Gheorghe Ștefan și Dominique Thiébaut, "Not Multi-, but Many-Core: Designing Integral Parallel Architectures for Embedded Computation", în *SIGARCH Comput. Archit. News* 35.5 (Dec. 2007), pp. 32–38, ISSN: 0163-5964, DOI: 10.1145/1360464.1360474.

[21] *Zynq-7000 SoC Technical Reference Manual*, https://docs.xilinx.com/v/u/en-US/ug585-Zynq-7000-TRM, California, USA: XILINX Corporation, 2021.

[22] George-Vlăduț Popescu, "Improvements in Data Transfer for a MapReduce Accelerator", în *Romanian Journal of Information Science and Technology (ROMJIST)* 25.3-4 (2022), pp. 368–380, ISSN: 1453-8245.

[23] Mihaela Malița, George-Vlăduț Popescu și Gheorghe M. Ștefan, "Heterogenous Computing for Markov Models in Big Data", în *2019 6TH INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE AND COMPUTATIONAL INTELLIGENCE (CSCI 2019)*, 2019, pp. 1500–1505, ISBN: 978-1-7281-5584-5, DOI: 10.1109/CSCI49370.2019.00279.

[24] Mihaela Malița, George-Vlăduț Popescu și Gheorghe M. Ștefan, "Pseudo- Reconfigurable Heterogeneous Solution for Accelerating Spectral Clustering", în *2020 IEEE INTERNATIONAL CONFERENCE ON BIG DATA (BIG DATA)*, IEEE International Conference on Big Data, IEEE; IEEE Comp Soc; IBM; Ankura, 2020, pp. 5138–5145, ISBN: 978-1-7281-6251-5, DOI: 10.1109/BigData50022.2020.9378150.

[25] *AMBA AXI and ACE Protocol Specification*, https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI3-and-AXI4-Protocol-Specification, Cambridge, UK: ARM Company, 2013.

[26] *AMBA AXI and ACE Protocol Specification*, https://developer.arm.com/documentation/ihi0051/a/Introduction/About-the-AXI4-Stream-protocol, Cambridge, UK: ARM Company, 2010.

[27] George-Vlăduţ Popescu și Călin Bîră, "Python-Based Programming Framework for a Heterogeneous MapReduce Architecture", în *2022 14th International Conference on Communications (COMM)*, 2022, pp. 1–6, DOI: 10.1109/COMM54429.2022.9817183.

[28] Xilinx, *PYNQ: Python productivity for Xilinx platforms*, https://pynq.readthedocs.io/en/v2.7.0/index.html, Version 2.7.0.

[29] Călin Bîră, Lucian Petrică și Radu Hobincu, "OPINCAA: A Light-Weight and Flexible Programming Environment For Parallel SIMD Accelerators", în *Romanian Journal of Information Science and Technology (ROMJIST)* 16.4 (2013), pp. 336–350, ISSN: 1453-8245.

[30] Alexandru E. Şuşu, "A Vector-Length Agnostic Compiler for the Connex-S Accelerator with Scratchpad Memory", în *ACM Trans. Embed. Comput. Syst.* 19.6 (Oct. 2020), ISSN: 1539-9087, DOI: 10.1145/3406536.

[31] Alexandru E. Șușu, "A C Compiler for the Wide, Low-Power CONNEX-S Vector Accelerator", în *University Politehnica of Bucharest Scientific Bulletin Seris C* 82.2 (2020), pp. 143–156, ISSN: 2286-3540.