ELEKTROTEHNIŠKI VESTNIK 87(5): 235–242, 2020
IZVIRNI ZNANSTVENI ČLANEK

Pospeševanje množenja matrik z vezjem FPGA in razvojnim okoljem Vitis

Miloš Ljubotina[1], **Andrej Žemva**[2], **Anton Biasizzo**[1]

[1]*Institut "Jožef Stefan", Jamova c. 39, 1000 Ljubljana, Slovenija*
[2]*Univerza v Ljubljani, Fakulteta za elektrotehniko, Tržaška c. 25, 1000 Ljubljana, Slovenija*
E-pošta: milos.ljubotina@gmail.com

Povzetek. Namen opravljenega dela je ovrednotiti programsko okolje Vitis Unified Software Platform za pospeševanje opravil, kot je množenje matrik. V vezju FPGA na razširitveni kartici PCI-e v gostiteljskem računalniku je z omenjenim orodjem implementiran sistoličen sistem za množenje matrik. Njegova zmogljivost je primerjana z zmogljivostjo obstoječe rešitve Intel MKL. Primerjava temelji na izmerjenih hitrostih izvajanja operacij in izračunanem številu opravljenih aritmetičnih operacij na enoto časa ter izkoriščenosti sredstev integriranih vezij. Rezultati kažejo v korist implementiranih sistemov, a je treba upoštevati, da se uporabljen podatkovni tip med uporabljenimi rešitvami razlikuje. Bistvo je, da je zmogljivost primerljiva in programsko okolje Vitis zmore proizvesti uporabne rezultate.

Ključne besede: programirljivo polje logičnih vrat, množenje matrik, Vitis Unified Software Platform, sistolično polje, pospeševanje algoritmov, načrtovanje digitalnih vezij

Accelerating matrix multiplication with an FPGA device and the Vitis development environment

The intent of this work is to evaluate the Vitis Unified Software Platform development environment for acceleration of tasks like matrix multiplication. For that, a system for multiplying matrices is implemented in an FPGA device on a PCI-e extension card in a host computer with the aforementioned development tools. The system performance is compared with the performance of an existing Intel MKL solution. The basis for the comparison are the measured execution speed, and the calculated number of performed arithmeric operations per unit time and integrated circuit resource utilisation efficiency. The results favor the implemented systems, however, the data type used in each solution is different. The key takeaway is that the performance is comparable and the Vitis development platform is able to provide useful results.

1 UVOD

Jedro tako rekoč vsakega računalnika je centralna procesna enota (angl. central processing unit oziroma CPU) ali procesor. To je integrirano vezje, ki med delovanjem računalnika vrši ukaze programov in obdeluje podatke. V kontekstu računske znanosti je procesor orodje, s katerim izvajamo algoritme oziroma rešujemo računske probleme. Kljub veliki zmogljivosti modernih procesorjev uporabljamo metode za pospeševanje izvajanja algoritmov, kajti to omogoča reševanje težjih in bolj kompleksnih problemov. K temu lahko pristopimo z uporabo različne dodatne strojne opreme. Na področju visokozmogljivega računalništva (angl. high performance computing) se v ta namen uporabljajo grafični procesorji, centralne procesne enote, programirljiva polja logičnih vrat (angl. field-programmable gate array oziroma FPGA) in posebej načrtovana integrirana vezja.

Prejet 10. avgust, 2020
Odobren 3. september, 2020

Lastnosti naštetih rešitev se razlikujejo [1]. Moderne centralne procesne enote so zelo zmogljive, a so namenjene širokemu spektru aplikacij. Imajo zelo kompleksno arhitekturo, ki jo je za posamezno aplikacijo težko maksimalno izkoristiti. Vsebujejo namreč elemente, ki niso potrebni pri vseh aplikacijah. Grafični procesorji so podobni centralnim procesnim enotam, a so namenjeni ožjemu naboru aplikacij. Posledično so v okviru tega nabora bolj zmogljivi. Arhitektura FPGA-jev je spremenljiva; delujejo pri nižji delovni frekvenci, a jih je mogoče prilagajati aplikacijam. To omogoča boljši izkoristek sredstev integriranega vezja in manjšo porabo električne energije kot pri splošnonamenskih procesorjih. Za aplikacije, ki so dobro podprte tudi v splošnonamenskih procesorjih, pa niso nujno hitrejši.

Pogosto omenjena slaba stran FPGA-jev je njihova kompleksnost za uporabo. Načrtovanje in implementacija logičnega vezja v FPGA lahko trajata veliko več časa kot razvoj programske opreme. Poleg tega zahtevata tudi globlje znanje o strojni platformi in razvojnih orodjih. Pred razvojem visokonivojskih orodij je bilo za realizacijo sistema v FPGA potrebnih več korakov: načrtovanje logičnega vezja, opis vezja v jeziku za opis digitalnih vezij, kot je VHDL (angl. very high speed integrated circuit hardware description language), sistemska integracija, vodenje postopka implementacije in morebiten razvoj gonilnikov za pospeševalnik [1].

Novejša orodja, kot je Vivado HLS (angl. high le-

vel synthesis), omogočajo visokonivojsko sintezo opisa vezja iz izvorne kode v prilagojeni obliki programskih jezikov C in C++ [2]. Pri pospeševanju algoritmov se glavni program običajno izvaja v centralni procesni enoti, računsko zahtevni del pa v pospeševalniku. Zato implementacija algoritma v pospeševalniku sama po sebi ne zadostuje. Realizirati je treba celovit sistem, ki omogoča predvidljivo delovanje pospeševalnika in komunikacijo s procesorjem. To pomeni, da so še vedno potrebni sistemska integracija, implementacija in razvoj gonilnikov. Moderna orodja, kot je Vitis Unified Software Platform ali Vitis, poleg visokonivojske sinteze ponujajo tudi avtomatiziran postopek sistemske integracije in implementacije ter gonilnike za pospeševalnik [3]. Uporabnik orodju poda le opis algoritma v jeziku za opis digitalnih vezij ali izvorno kodo za visokonivojsko sintezo. Orodje realizira jedra (angl. kernels), kot pravimo implementacijam algoritmov v pospeševalniku, v vezju FPGA in sistem, ki uporabniku omogoča dostop do jeder preko standardnega aplikacijskega programskega vmesnika OpenCL [1]. To omogoča hitrejši razvoj aplikacij in zmanjša obseg potrebnega znanja za pospeševanje algoritmov z vezji FPGA.

Povod za opravljeno raziskovalno delo je bila želja po pospeševanju učenja nevronskih mrež z vezjem FPGA. V prvem koraku smo se odločili za implementacijo množenja matrik, saj je to poglavitni del računske zahtevnosti učenja nevronskih mrež in se uporablja tudi na drugih področjih znanosti. Za realizacijo sistema smo se odločili uporabiti visokonivojsko razvojno okolje Vitis. Ker je okolje novo, nas je zanimal potek dela z orodjem in kakovost rezultatov. Cilj dela je zato ovrednotiti razvojno okolje Vitis za pospeševanje opravil, kot je množenje matrik. Za ta namen smo v vezju FPGA na razširitveni kartici z vodilom PCI-e (angl. peripheral component interconnect express) realizirali dva sistema za množenje matrik in njune zmogljivosti primerjali z zmogljivostjo obstoječe rešitve, Intel MKL. Načrtovali smo tudi primerjavo z rešitvijo Vitis BLAS, a so bili rezultati operacij s to knjižnico nepravilni, zato smo realizirana sistema primerjali le s knjižnico Intel MKL.

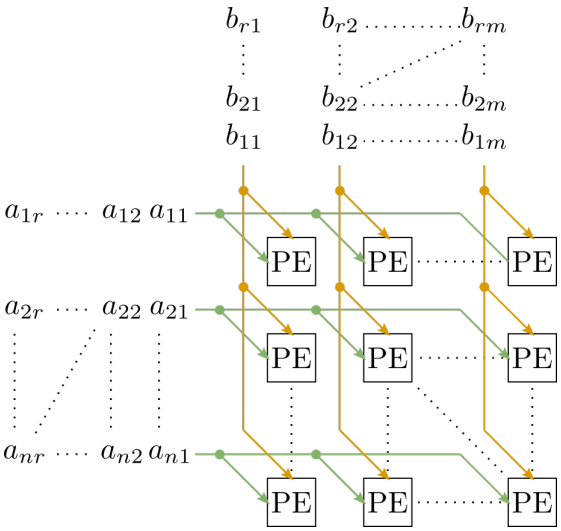
2 MNOŽENJE MATRIK

Matrike so pravokotna polja elementov, ki so razporejeni v vrstice in stolpce. Množenje matrik lahko opišemo z enačbama 1:

$$\begin{aligned} \mathbf{C} &= \mathbf{AB}, \\ c_{i,j} &= \sum_{k=1}^{r} a_{i,k} b_{k,j}, \end{aligned} \tag{1}$$

kjer je r število stolpcev matrike $\mathbf{A}$ oziroma vrstic matrike $\mathbf{B}$. Element v i-ti vrstici in j-tem stolpcu matrike $\mathbf{C}$ je enak vsoti zmnožkov soležnih elementov i-te vrstice leve matrike in j-tega stolpca desne matrike. Od tu sledi naiven algoritem za množenje matrik z računsko zahtevnostjo $\mathcal{O}(nmr)$, pri čemer so matrike $\mathbf{A}$, $\mathbf{B}$ in $\mathbf{C}$ velikosti $n \times r$, $r \times m$ in $n \times m$. Za vsak element matrike $\mathbf{C}$ je treba izvesti $\mathcal{O}(r)$ seštevanj in množenj, elementov pa je nm. Če predpostavimo, da so matrike kvadratne, dobimo kubično zahtevnost, $\mathcal{O}(n^3)$. Obstajajo tudi algoritmi, kot sta Strassenov in Coppersmith-Winogradov algoritem, ki po asimptotični notaciji zahtevajo manjše število operacij [4], [5]. Kljub temu se v praksi večinoma uporabljajo algoritmi z računsko zahtevnostjo $\mathcal{O}(n^3)$, Coppersmith-Winogradov algoritem namreč postane hitrejši šele pri velikosti matrik, ki s trenutno tehnologijo ni dosegljiva, Strassenov algoritem pa se uporablja samo pri zelo velikih matrikah, ker je za manjše matrike težaven za implementacijo v moderni strojni opremi [5], [6].

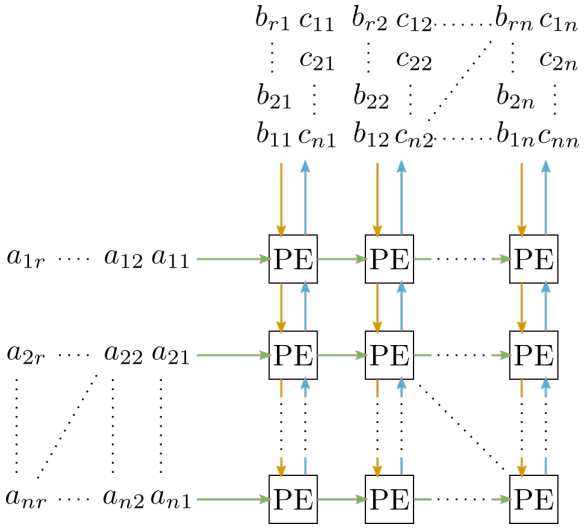
Za implementacijo množenja matrik v vezju FPGA smo uporabili enostavno sistolično polje. To je planarna mreža med seboj povezanih procesnih elementov (PE), ki sinhrono prenašajo in obdelujejo podatke [7]. Pri načrtovanju smo se zgledovali po obstoječih raziskavah o implementiranju množenja matrik s sistoličnim poljem [8], [9]. Na sliki 1 je prikazan diagram uporabljenega polja, s katerim je mogoče sočasno računati nm elementov produkta dveh matrik.



Slika 1: Diagram enostavnega sistoličnega polja za množenje matrik.

Naj bo matrika $\mathbf{C}$ enaka zmnožku matrik $\mathbf{A}$ in $\mathbf{B}$, razsežnosti $n \times r$ ter $r \times m$. V eni periodi ure vsak PE zmnoži podatka na svojih podatkovnih vhodih in rezultat prišteje vrednosti v notranjem registru oziroma akumulatorju. Na podatkovne vhode polja sta vsak cikel ure pripeljana stolpec matrike $\mathbf{A}$ in vrstica matrike $\mathbf{B}$. Po r urinih ciklih akumulatorji procesnih elementov vsebujejo elemente matrike $\mathbf{C}$. Taka arhitektura polja za kvadratne matrike zahteva n^2 procesnih elementov in n urinih ciklov, v katerih izvede n^3 operacij, a je lahko težavna za implementacijo v vezju FPGA. Vhodni podatki morajo priti do vsakega PE, kar pri večjih poljih povzroča večjo obremenitev izhodov (angl. fan-

out) ter zahteva večji delež in gostoto infrastrukture za povezovanje blokov vezja FPGA. Podobno velja za branje rezultatov iz akumulatorjev procesnih elementov. To lahko omeji maksimalno delovno frekvenco implementacije, ali, če infrastruktura ne premore zadostne gostote povezav, to celo prepreči. Bolj primerne so arhitekture, pri katerih povezave tečejo le med sosednjimi elementi polja. Tako potrebna gostota povezav in fan-out podatkovnih linij nista odvisna od velikosti polja. Zgled tega prikazuje slika 2.



Slika 2: Primer sistoličnega polja za množenje matrik, ki je primerno za vezja FPGA. Prikazani so le vrstni redi prenosa podatkov za posamezne podatkovne poti, ne pa njihova medsebojna odvisnost.

Računa se produkt, $\mathbf{C}$, matrik $\mathbf{A}$ in $\mathbf{B}$, razsežnosti $n \times r$ ter $r \times n$. V prvem urinem ciklu sta v polje vstavljeni le vrednosti a_{11} in b_{11}, katerih produkt je izračunan v prvem procesnem elementu. V naslednjem ciklu je vrednost a_{11} posredovana naslednjemu procesnemu elementu v prvi vrstici, vrednost b_{11} pa naslednjemu elementu v prvem stolpcu. Vstavljene so vrednosti a_{12}, a_{21}, b_{21} in b_{12} ter izračunani produkti $a_{12}b_{21}$, $a_{21}b_{11}$ in $a_{11}b_{12}$. Po r korakih je v akumulatorju prvega PE vrednost c_{11}, v naslednjem sta v sosednjih PE na voljo vrednosti c_{12} in c_{21}, po nadaljnjih $2n-3$ ciklov pa tudi c_{nn} v zadnjem procesnem elementu. Po tem, ko se izračun posameznega elementa matrike $\mathbf{C}$ zaključi, je ta postopoma prenesen do podatkovnega izhoda polja.

Za izračun in ekstrakcijo zmnožka kvadratnih matrik s takim sistoličnim poljem je potrebnih n^2 procesnih elementov in $4n - 3$ urinih ciklov, a je z ustrezno nadzorno logiko mogoče ustvariti cevovod, ki omogoča enak pretok podatkov kot v primeru na sliki 1.

Ker je število procesnih elementov, ki jih premorejo vezja FPGA, omejeno, je na tak način mogoče množiti le majhne matrike. Za večje matrike se lahko uporabi pristop deljenja na podmatrike, kar se uporablja tudi pri centralnih procesnih enotah, ker lahko tako bolje izkoristimo arhitekturo modernih procesorjev [10]. Enostaven način deljenja je ponazorjen z enačbo 2. Podmatrike so velikosti $n_{sys} \times r$ in $r \times n_{sys}$, pri čemer n_{sys} označuje velikost kvadratnega polja procesnih elementov. Če število vrstic leve matrike ali število stolpcev desne ni deljivo z n_{sys}, matriko dopolnimo z ničlami. Posamezne produkte podmatrik je nato treba sestaviti v ustrezno celoto.

$$\begin{aligned} \mathbf{AB} &= \begin{bmatrix} \mathbf{A_1} \\ \mathbf{A_2} \\ \vdots \\ \mathbf{A_N} \end{bmatrix} \begin{bmatrix} \mathbf{B_1} & \mathbf{B_2} & \dots & \mathbf{B_M} \end{bmatrix} = \\ &= \begin{bmatrix} \mathbf{A_1B_1} & \mathbf{A_1B_2} & \dots & \mathbf{A_1B_M} \\ \mathbf{A_2B_1} & \mathbf{A_2B_2} & \dots & \mathbf{A_2B_M} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{A_NB_1} & \mathbf{A_NB_2} & \dots & \mathbf{A_NB_M} \end{bmatrix} \end{aligned} \tag{2}$$

Dobra lastnost sistoličnih polj je učinkovita raba podatkov. Med neprekinjenim delovanjem opisanih polj je vsak cikel vstavljenih $2n_{sys}$ elementov, opravi pa se n_{sys}^2 operacij. Vsak element, ki pride na vhod polja, je uporabljen n_{sys}-krat. Večkrat je podatek uporabljen znotraj polja, manjkrat mora biti pripeljan do polja. Posledično je podatkovna pot zunaj polja manj obremenjena oziroma je lahko bolj enostavna. To ne le olajša delo načrtovalca, ampak tudi omogoča bolj kakovosten rezultat implementacije vezja v FPGA.

Možnih je še mnogo drugih arhitektur sistoličnih polj za množenje matrik, ki se razlikujejo po številu procesnih elementov, zakasnitvi, pretoku podatkov, kompleksnosti za implementacijo in ustreznosti za realizacijo v vezju FPGA.

3 Strojna platforma

Za realizacijo sistemov smo uporabili računalnik z razširitveno kartico Alveo U250, ki je namenjena pospeševanju računsko zahtevnih aplikacij [11]. Kartica vključuje vezje FPGA Virtex UltraScale+ XCU250-2LFIGD2104E, štiri med seboj neodvisne module DIMM (angl. dual in-line memory module) pomnilnika DDR4-2400 ECC (angl. error-correcting code) s skupno kapaciteto 64 GiB in druge elemente [12]. V tabeli 1 so našteta pomembnejša sredstva, ki jih ponujata kartica in vezje FPGA. Poleg komunikacijskih vodil in pomnilnika so v tabeli tudi vnosi za vpogledne tabele (angl. look-up table oziroma LUT), bistabilne multivibratorje (FF), digitalne signalne procesorje (DSP), bloke statičnega bralno-pisalnega pomnilnika (angl. block random-access memory oziroma BRAM) in bloke pomnilnika UltraRAM (URAM) [1].

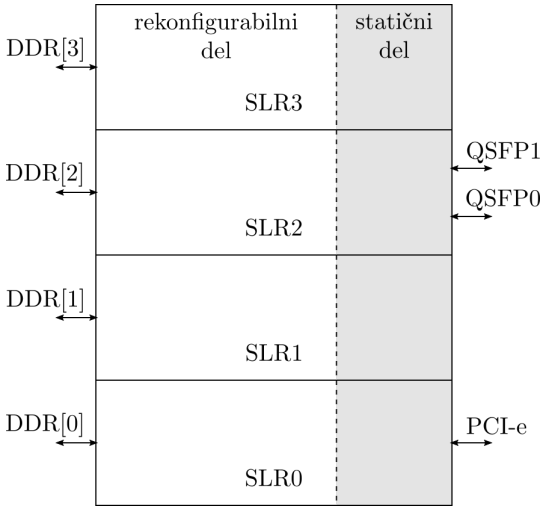
Celotno vezje FPGA je izdelano s tehnologijo SSI (angl. stacked silicon interconnect), kar pomeni, da je sestavljeno iz več integriranih vezij, ki so med seboj povezana preko skupne rezine silicija [13]. Uporabljeno vezje FPGA vključuje štiri integrirana vezja oziroma regije SLR (angl. super logic region). Ker so povezave

Tabela 1: Razpoložljiva sredstva razširitvene kartice Alveo U250 in vsebovanega vezja FPGA.

Sredstvo	Količina
vmesnik PCI-e	Gen3 x16
omrežni vmesnik	2 x QSFP28
zunanji pomnilnik	4 x 16 GiB DDR4-2400
LUT	1 727 040
FF	3 454 080
DSP	12 288
BRAM	2688
URAM	1280

med regijami daljše in počasnejše kot povezave znotraj regij, je dobro, da se posamezno jedro ne razpenja preko več regij. To ima velik vpliv pri načrtovanju arhitekture sistema za tako vezje FPGA.

Pri uporabi razširitvene kartice z visokonivojskim programskim okoljem Vitis del sredstev vezja FPGA zavzame statičen del načrta, ki ga lahko obravnavamo kot del strojne platforme. Na sliki 3 je prikazana topologija regij in pomembnejših zunanjih povezav. V vsaki regiji je na voljo približno četrtina sredstev rekonfigurabilnega dela vezja.



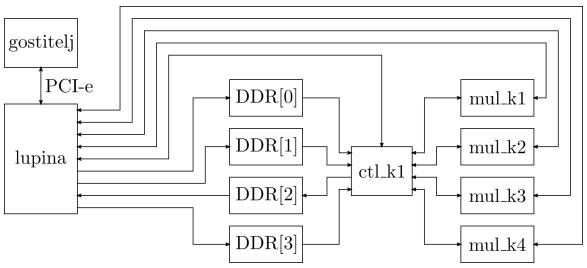
Slika 3: Topologija regij in pomembnejših zunanjih povezav za uporabljeno strojno platformo.

4 ARHITEKTURA

Realiziran sistem podpira množenje matrik velikosti $n \times r$ in $r \times m$, pri čemer so parametri n, r in m večkratniki števila 64 ter niso večji od 1024. Matrike, ki ne zadoščajo tem pogojem, je mogoče ustrezno dopolniti z ničlami oziroma razdeliti na podmatrike. Uporabljena je 16-bitna predstavitev podatkov s fiksno vejico. Pozicija vejice je podana pred implementacijo sistema, zato ne more biti spremenjena med obratovanjem pospeševalnika. Elementi matrik se med gostiteljem in razširitveno kartico prenašajo v vrstnem redu vrstic in stolpcev, torej ne v transponirani obliki. Z vsakim ukazom je mogoče pospeševalniku podati več parov matrik. To v situacijah, v katerih je treba izvesti več neodvisnih operacij, skrajša potreben režijski čas (angl. overhead) in omogoča večji pretok podatkov.

Arhitektura realiziranega sistema temelji na enostavnem sistoličnem polju, ki je bilo predstavljeno v poglavju 2, velikosti 64×64 [1]. Implementirano je bilo tudi drugo predstavljeno polje, a je rezultat slabši. Vzrok za to je verjetno neusklajenost interpretacije opisa vezja med načrtovalcem in prevajalnikom. Za procesne elemente so uporabljeni digitalni signalni procesorji, ki izvajajo operacijo MAC (angl. multiply-accumulate). Ker je teh znotraj ene regije premalo za celotno polje, je to razdeljeno na štiri manjša polja velikosti 32×32. Tako je v posamezni regiji mogoče implementirati dve polji, širina podatkovnih priključkov polj pa je enaka širini vodil pomnilniških vmesnikov, ki je 512 bitov. S tem so lahko širine vseh podatkovnih poti enake, kar poenostavi nadzorno logiko, dovoljuje bolj varčno uporabo sredstev in lahko pomeni višjo delovno frekvenco implementacije.

Na sliki 4 je prikazana topologija realiziranega sistema. Bloki ctl_k1, mul_k1, mul_k2, mul_k3 in mul_k4 so računske enote oziroma posamezne realizacije jeder, lupina pa je ime za statični del vezja in nadzoruje delovanje računskih enot. Blok ctl_k1 skrbi za pretok podatkov med zunanjimi vodili in preostalimi računskimi enotami, ki vsebujejo sistolična polja. Prva dva modula zunanjega pomnilnika sta uporabljena za podajanje operandov, tretji za rezultate in zadnji za prenos informacij o velikosti matrik. Računska enota ctl_k1 je postavljena v regijo SLR1, mul_k1 in mul_k2 v regijo SLR0 ter mul_k3 in mul_k4 v regijo SLR2.



Slika 4: Topologija realiziranega sistema.

5 IMPLEMENTACIJA

Za prevajanje izvorne kode je bil poleg obveznih parametrov prevajalniku v++ podan še opis topologije sistema in sledeče nastavitve za optimizacijo:

```
–O3
––vivado.prop:run.impl_1.STEPS.
    PHYS_OPT_DESIGN.IS_ENABLED=true
```

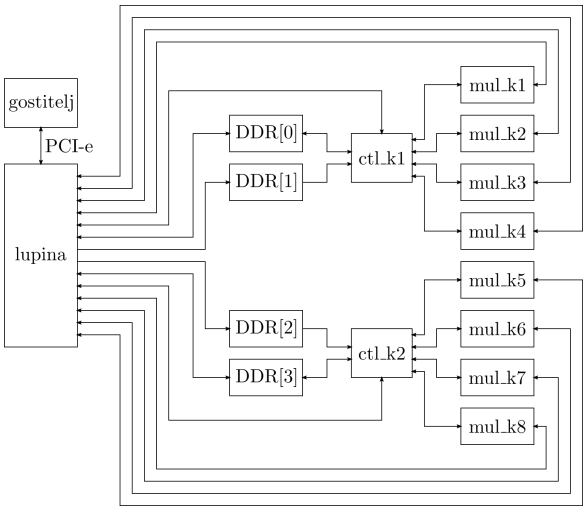
```
--vivado.prop:run.impl_1.STEPS.
    PHYS_OPT_DESIGN.ARGS.DIRECTIVE=
    Explore
```

Privzeta delovna frekvenca jeder na izbrani platformi je 300 MHz, vendar se samodejno zmanjša, če prevajalniku ne uspe zadostiti časovnim zahtevam. Realiziran sistem deluje pri frekvenci 231 MHz in zaseda sredstva vezja FPGA, kot je zapisano v tabeli 2. Del vpoglednih tabel je uporabljen kot porazdeljeni pomnilnik (DRAM) ali pomikalni register (SRL), kar je prikazano ločeno.

Tabela 2: Zasedenost sredstev za implementiran sistem.

ime sredstva	razpoložljiva količina	uporabljena količina
LUT	1 499 566	128 028 (8,5 %)
DRAM ali SRL	765 160	3081 (0,4 %)
FF	3 112 686	311 539 (10,0 %)
BRAM	2281	437 (19,2 %)
URAM	1280	128 (10,0 %)
DSP	12 272	4098 (33,4 %)

Ker je bila implementacija uspešna in je na voljo dovolj sredstev, smo realizirali še alternativno različico sistema z dvema neodvisnima podsistemoma. Ta sta enaka že predstavljenemu sistemu, le topologija celote je drugačna, kot je vidno na sliki 5. Modula pomnilnika DDR[0] in DDR[3] sta uporabljena za posredovanje elementov levih matrik in informacij o velikosti matrik, DDR[1] in DDR[2] pa za elemente desnih matrik in rezultate. V regiji SLR0 sta računski enoti mul_k1 in mul_k2, v regiji SLR1 so ctl_k1, mul_k7 in mul_k8, v regiji SLR2 so ctl_k2, mul_k3 in mul_k4, v regiji SLR3 pa mul_k5 in mul_k6. Dosežena je bila delovna frekvenca 187 MHz in uporaba sredstev, kot je prikazana v tabeli 3.



Slika 5: Topologija alternativne različice sistema.

Tabela 3: Zasedenost sredstev za alternativno različico implementiranega sistema.

ime sredstva	razpoložljiva količina	uporabljena količina
LUT	1 499 566	255 465 (17,2 %)
DRAM ali SRL	765 160	6162 (0,8 %)
FF	3 112 686	622 238 (20,2 %)
BRAM	2281	874 (39,5 %)
URAM	1280	256 (20,0 %)
DSP	12 272	8196 (66,8 %)

6 MERITVE

Primerjava zmogljivosti obstoječih rešitev in razvitih sistemov temelji na hitrosti obdelave podatkov posamezne rešitve. Hitrosti so bile določene empirično, pri čemer sta bila upoštevana dva različna primera uporabe množenja matrik. Pri prvem je bil merjen čas izolirane operacije oziroma njena sekvenčna zakasnitev, pri drugem pa največja prepustnost (angl. throughput) sistema. Za slednjega velja predpostavka, da je treba izvršiti več neodvisnih operacij hkrati, torej da so vsi vhodni podatki na voljo pred začetkom izračunov in je mogoče izkoristiti cevovodne in paralelne značilnosti strojne opreme. V obeh primerih je bilo na podlagi meritev in znanega števila operacij, potrebnih za izračun rezultatov, določeno število opravljenih aritmetičnih operacij na enoto časa. Poleg tega je bila določena tudi izkoriščenost sredstev integriranih vezij kot razmerje med izmerjenim številom aritmetičnih operacij na enoto časa in teoretičnim maksimumom. Spremenljivke t_{sek}, N_{sek} in γ_{sek} predstavljajo čas izvedbe množenja dveh matrik, število operacij na enoto časa in izkoriščenost sredstev za prvi tip meritve, t_{pre}, N_{pre} in γ_{pre} pa za drugega. Vse meritve so bile ponovljene 100-krat in rezultati povprečeni.

Pri meritvah sekvenčne zakasnitve je bil merjen čas ene operacije množenja kvadratnih matrik velikosti $n \times n$, pri meritvah prepustnosti pa čas množenja M parov matrik velikosti $n \times n$. Število M je bilo določeno tako, da je bilo trajanje vsake meritve razreda 100 ms. Količine t_{sek}, t_{pre}, N_{sek}, N_{pre}, γ_{sek} in γ_{pre} so definirane z relacijami 3:

$$\begin{aligned} t_{sek} = \text{izmerjen čas},\ t_{pre} = \frac{\text{izmerjen čas}}{M}, \\ N_{sek} = \frac{P}{t_{sek}},\ N_{pre} = \frac{P}{M t_{pre}}, \\ \gamma_{sek} = \frac{N_{sek}}{N_{max}},\ \gamma_{pre} = \frac{N_{pre}}{N_{max}}, \end{aligned} \tag{3}$$

kjer je P število izvršenih aritmetičnih operacij, N_{max} pa teoretično največje možno število izvršenih aritmetičnih operacij na enoto časa.

Za meritve časa je bila uporabljena centralna procesna enota posamezne platforme. S sistemskim klicem clock_getres je bilo ugotovljeno, da sistemski klic

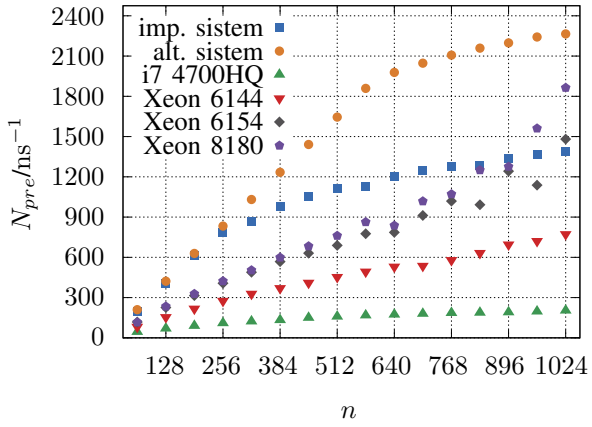
clock_gettime omogoča merjenje časa z ločljivostjo 1 ns. Uporabljen je bil tip ure CLOCK_MONOTONIC_RAW, ki omogoča dostop do strojne ure brez sinhronizacijskih popravkov in vpliva sistema NTP (angl. network time protocol).

Meritve realiziranih sistemov so bile opravljene na platformi z razširitveno kartico Alveo U250, meritve s knjižnico Intel MKL pa na sistemih z Intelovimi procesorji: i7 4700HQ, Xeon Gold 6144, Xeon Gold 6154 in Xeon Platinum 8180 [1]. Delovna frekvenca prvega je bila nastavljena na njegovo nominalno frekvenco, 2,4 GHz, frekvence preostalih pa na njihove nominalne frekvence za nabor ukazov AVX-512 (angl. advanced vector extensions 512). Ta je za procesor Intel Xeon Gold 6144 enaka 2,2 GHz, za Intel Xeon Gold 6154 2,1 GHz in za Intel Xeon Platinum 8180 1,7 GHz. Ker knjižnica Intel MKL ne podpira 16-bitnega podatkovnega tipa s fiksno vejico, je bil uporabljen 32-bitni tip s plavajočo vejico.

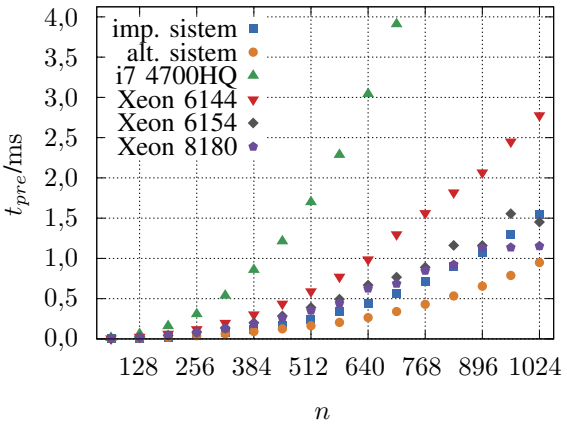
Pri uporabi knjižnice Vitis BLAS je prišlo do težav, zaradi katerih je izvzeta iz primerjave. Pri množenju več parov matrik, četudi ne takoj enega za drugim, je zakasnitev množenja z vsakim parom naraščala in rezultati so bili nepravilni.

7 Analiza rezultatov

Na slikah 6 in 7 so prikazane karakteristike prepustnosti sistemov. Največjo prepustnost izkazuje alternativna različica realiziranega sistema, za to pa prvotna oblika implementiranega sistema in knjižnica Intel MKL na procesorju Intel Xeon Platinum 8180. Graf N_{pre} slednje platforme proti koncu narašča veliko hitreje kot grafa implementiranih sistemov, kar nakazuje možnost, da je ta pri množenju večjih matrik hitrejša. Za zastavljen cilj to ni ključnega pomena, a se pojavi vprašanje o izkoriščenosti sredstev strojne opreme.



Slika 6: Število opravljenih aritmetičnih operacij na enoto časa v odvisnosti od velikosti matrik pri meritvah prepustnosti.



Slika 7: Zakasnitev operacije v odvisnosti od velikosti matrik pri meritvah prepustnosti.

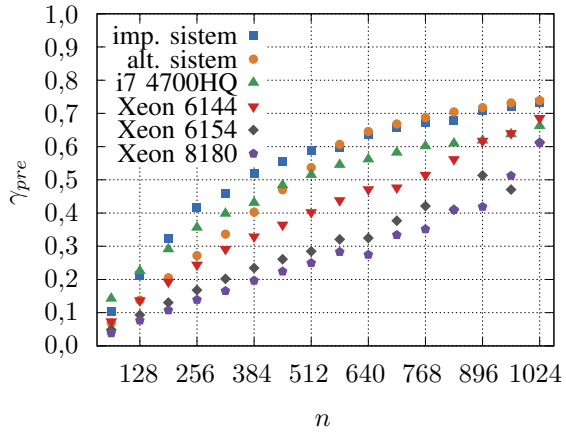
V tabeli 4 so naštete frekvence, f, največje število aritmetičnih operacij na periodo ure, OPC, in N_{max} za vse sisteme. OPC implementiranih sistemov je določen na podlagi uporabljenih, ne pa vseh razpoložljivih sredstev. Namen je namreč prikazati učinkovitost platforme in prevajalnika, ne načrtovalca.

Tabela 4: Delovne frekvence, število operacij na periodo ure in število operacij na enoto časa za uporabljene strojne platforme.

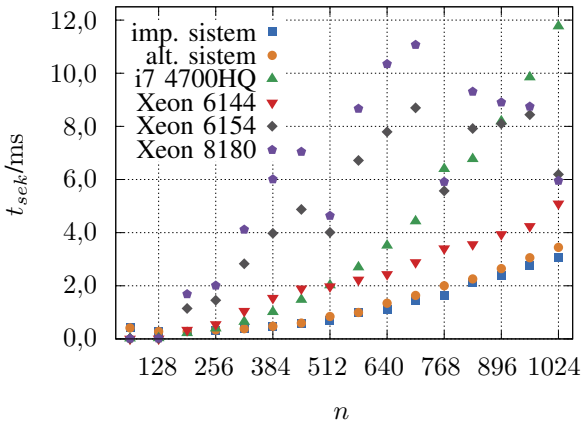
platforma	f/GHz	OPC	N_{max}/ns^{-1}
impl. sistem.	0,231	8192	1892,352
alt. impl. sistem.	0,187	16 384	3063,808
Intel i7 4700HQ	2,400	128	307,200
Intel Xeon 6144	2,200	512	1126,400
Intel Xeon 6154	2,100	1152	2419,200
Intel Xeon 8180	1,700	1792	3046,400

Slika 8 prikazuje grafe izkoriščenosti sredstev strojne opreme v odvisnosti od velikosti matrik za meritve prepustnosti. Ker je pri majhnih matrikah vpliv prenosa podatkov in režijskega časa večji, imajo za manjše n platforme z manj sredstvi boljši izkoristek. Pri največjih uporabljenih matrikah imata najboljši izkoristek implementirana sistema, a razlike med vsemi platformami tu niso velike. Tako kot na grafih N_{pre} je vidno, da proti koncu še vedno naraščajo. To pomeni, da bi pri večjih matrikah vse platforme verjetno dosegle še boljši izkoristek. Vzrok za to je v naraščanju razmerja med številom aritmetičnih operacij, potrebnih za množenje matrik, in količine prenesenih podatkov. Večje so matrike, manjši delež zakasnitve operacije predstavljata prenos in režijski čas ter večji delež delovanje procesnih elementov.

Pri meritvah sekvenčne zakasnitve, katerih rezultati so na slikah 9 in 10, so okoliščine precej drugačne. V tem primeru ima medpomnilnik centralnih procesnih enot lahko velik vpliv na hitrost izvršitve operacije, množenje

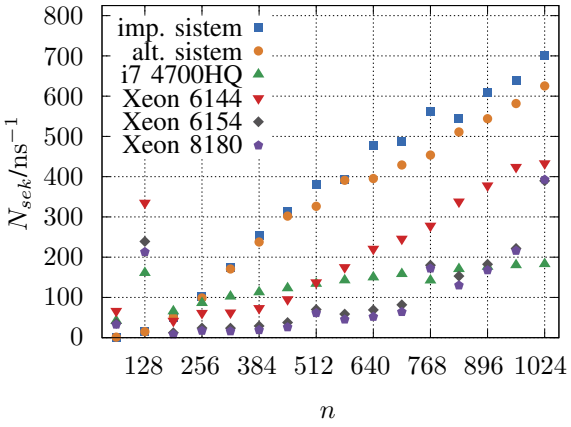


Slika 8: Izkoriščenost sredstev strojne opreme v odvisnosti od velikosti matrik pri meritvah prepustnosti.



Slika 10: Zakasnitev operacije v odvisnosti od velikosti matrik pri meritvah sekvenčne zakasnitve.

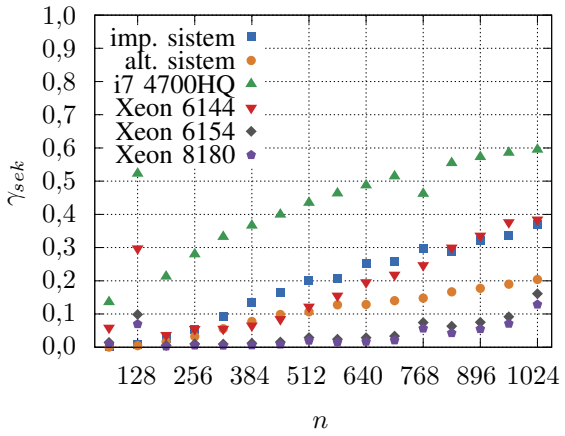
se namreč izvede takoj za inicializacijo podatkov [1]. To pomeni, da so lahko za majhne n ob začetku operacije vsi dotični podatki že v medpomnilniku, kar zelo zmanjša vpliv zakasnitve zaradi dostopa do delovnega spomina. Vpliv medpomnilnika je mogoč vzrok za velik N_{sek} centralnih procesnih enot pri $n = 64$ in $n = 128$.



Slika 9: Število opravljenih aritmetičnih operacij na enoto časa v odvisnosti od velikosti matrik pri meritvah sekvenčne zakasnitve.

Zanimivo je, da sta pri večjih matrikah, kljub dodatni zakasnitvi prenosa podatkov preko vodila PCI-e, implementirana sistema občutno hitrejša od preostalih platform. Morda so matrike premajhne za učinkovito porazdelitev dela na več procesorskih jeder.

Na sliki 11 je prikazana izkoriščenost sredstev strojne opreme za meritve sekvenčnih zakasnitev. Za vse platforme, z izjemo MKL i7 4700HQ, je ta občutno manjša kot pri meritvah prepustnosti. Pri procesorjih Xeon verjetno zaradi omejene velikosti matrik, pri implementiranih sistemih pa zaradi neizkoriščenosti cevovodne arhitekture sistema. Procesor Intel i7 4700HQ ima manj jeder in procesorske moči kot preostali, zato je s tem lažje doseči večjo izkoriščenost sredstev.



Slika 11: Izkoriščenost sredstev strojne opreme v odvisnosti od velikosti matrik pri meritvah sekvenčne zakasnitve.

8 Zaključek

Pokazalo se je, da je zmogljivost implementiranih sistemov primerljiva oziroma celo boljša kot zmogljivost knjižnice Intel MKL na uporabljenih strojnih platformah. Pri tem je treba upoštevati, da Intel MKL za množenje matrik ne podpira 16-bitnega podatkovnega tipa s fiksno vejico in je bil uporabljen 32-bitni tip s plavajočo vejico. Tudi če bi bil 16-bitni tip s fiksno vejico podprt, hitrost obdelave podatkov ne bi mogla biti veliko večja, saj bi frekvenca ostala nespremenjena, OPC pa bi se zaradi polovične velikosti podatkovnega tipa kvečjemu podvojil. Vendar tudi to ni nujno. Nabora ukazov FMA (angl. fused multiply-add) in AVX-512 namreč nimata neposredne podpore za tak podatkovni tip. Bistvo je, da so zmogljivosti rešitev primerljive in se je pokazala uporabnost programskega okolja Vitis. To

podpirajo tudi izračuni izkoriščenosti uporabljenih sredstev strojnih platform, kajti pri implementiranih sistemih dosežejo tudi več kot 70 %, kar bi lahko z nadaljnjo optimizacijo še izboljšali.

Kljub uporabnosti razvojnega okolja to ne nadomešča nizkonivojskih pristopov k načrtovanju sistemov za vezja FPGA. Vsak način dela ima svoje prednosti. Pri Vitisu je verjetno najpomembnejša prednost abstrakcija sistemske integracije, implementacije, gonilnika in opisnega jezika, kar precej zmanjša obseg potrebnega znanja o strojni opremi. Glede na osebne izkušnje pa je vseeno treba nameniti kar nekaj časa spoznavanju orodja.

V nadaljnjem delu bi lahko implementirana sistema primerjali še z ekvivalentno implementacijo z nizkonivojskim pristopom. Pri tem bi bilo smiselno dati poudarek izkoriščenosti sredstev in podobnim parametrom, ne pa nujno hitrosti obdelave podatkov.

Ker imajo grafični procesorji zelo dobro podporo za množenje matrik, je z vezji FPGA težko preseči zmogljivost grafičnih procesorjev za ta namen. Prednost FPGA-jev je v možnosti prilagajanja vezja aplikaciji. Zato bi bila bolj zanimiva implementacija celotnega postopka učenja nevronskih mrež, ne samo množenja matrik. Na področju nevronskih mrež se uporabljajo razne metode in podatkovni tipi, ki niso vedno dobro podprti v obstoječi strojni opremi. To bi lahko bila zelo zanimiva usmeritev za nadaljnje raziskovalno delo z razvojnim okoljem Vitis.

9 Zahvala

Raziskavo je omogočila Javna agencija za raziskovalno dejavnost Republike Slovenije v okviru programa P2-0098 - Računalniške strukture in sistemi.

Literatura

[1] M. Ljubotina, "Pospeševanje množenja matrik z vezjem FPGA in razvojnim okoljem Vitis," magistrsko delo, Fakulteta za elektrotehniko, Univerza v Ljubljani, Ljubljana, 2020.

[2] *Vivado Design Suite User Guide: High-level Synthesis,* v2019.2, Xilinx, Inc., 13. jan. 2020. Dostopano: 7. sep. 2020. Dosegljivo: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug902-vivado-high-level-synthesis.pdf

[3] *Vitis Unified Software Platform Documentation: Application Acceleration Development,* v2019.2, Xilinx, Inc., 28. feb. 2020. Dostopano: 7. sep. 2020. Dosegljivo: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2019_2/ug1393-vitis-application-acceleration.pdf

[4] V. Strassen, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, str. 354–356, 1969.

[5] D. Coppersmith in S. Winograd, "Matrix multiplication via arithmetic progressions," v *Proceesings of the nineteenth annual ACM symposium on the Theory of computing*, str. 1–6, 1987.

[6] M. Thottethodi, S. Chatterjee in A. R. Lebeck, "Tuning strassen's matrix multiplication for memory efficiency," v *SC '98: Proceesings of the 1998 ACM/IEEE Conference on Supercomputing*, str. 36–36, 1998.

[7] I. S. Duff in G. W. Stewart *et al.*, *Sparse matrix proceedings, 1978,* vol. 8. Siam, 1979.

[8] I. Milentijević, I. Z. Milovanović, E. Milovanović in M. Stojcev, "The design of optimal planar systolic arrays for matrix multiplication," *Computers & Mathematics with Applications*, vol. 33, no. 6, str. 17–35, 1997.

[9] C. Wan in D. J. Evans, "Nineteen ways of systolic matrix multiplication," *International journal of computer mathematics*, vol. 68, no. 1-2, str. 39–69, 1998.

[10] M. D. Lam, E. E. Rothberg in M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *ACM SIGOPS Operating Systems Review*, vol. 25, no. Special Issue, str. 63–74, 1991.

[11] *Alveo Data Center Accelerator Card Platforms User Guide,* v1.2, Xilinx, Inc., 26. jun. 2020. Dostopano: 7. sep. 2020. Dosegljivo: https://www.xilinx.com/support/documentation/boards_and_kits/accelerator-cards/ug1120-alveo-platforms.pdf

[12] *Ultrascale Architecture and Product Data Sheet: Overview,* v3.13, Xilinx, Inc., 21. jul. 2020. Dostopano: 7. sep. 2020. Dosegljivo: https://www.xilinx.com/support/documentation/data_sheets/ds890-ultrascale-overview.pdf

[13] K. Saban, "Xilinx Stacked Silicon Interconnect Technology Delivers Breakthrough FPGA Capacity, Bandwisth, and Power Efficiency," Xilinx, Inc., 11. dec. 2012. Dostopano: 7. sep. 2020. Dosegljivo: https://www.xilinx.com/support/documentation/white_papers/wp380_Stacked_Silicon_Interconnect_Technology.pdf

Miloš Ljubotina je v letih 2017 in 2020 diplomiral in magistriral na Fakulteti za elektrotehniko Univerze v Ljubljani. Med študijem je delal na Institutu "Jožef Stefan", kjer se je ukvarjal z načrtovanjem in implementacijo digitalnih vezij v programirljivih poljih logičnih vrat.

Andrej Žemva je v letih 1989, 1993 in 1996 diplomiral, magistriral in doktoriral na Fakulteti za elektrotehniko Univerze v Ljubljani. Njegova raziskovalna in razvojna dejavnost obsega načrtovanje in testiranje digitalnih elektronskih vezij in sistemov, načrtovanje in izdelavo vgrajenih sistemov ter sočasno načrtovanje strojne in programske opreme.

Anton Biasizzo je v letih 1991, 1995 in 1998 diplomiral, magistriral in doktoriral na Fakulteti za elektrotehniko Univerze v Ljubljani. Od leta 1991 je raziskovalec na Institutu "Jožef Stefan" v Ljubljani. Od leta 2008 je docent na Mednarodni podiplomski šoli Jožefa Stefana v Ljubljani. Njegovi raziskovalni interesi se usmerjajo na področje rekonfigurabilnih sistemov ter načrtovanja in testiranja digitalnih sistemov.