Hardware reconfigurabil: o nouă paradigmă arhitecturală

Mihai Budiu — mihaib+@cs.cmu.edu
http://www.cs.cmu.edu/~mihaib/

7 iunie 1998

Subiect: hardware reconfigurabil prin software

Cunoştinţe necesare: cunoştinţe elementare despre arhitectura calculatoarelor

Cuvinte cheie: poartă, memorie, procesor, FPGA, ASIC

Cuprins

Despre hardware reconfigurabil am auzit cu puţina vreme în urmă; atracţia a fost însă imediată. Acest articol va încerca să vă transmită o parte din motivele pentru care domeniul mi se pare demn de atenţie. Unul dintre ele este faptul că relativ puţină cercetare sistematică s-a făcut în domeniul circuitelor reconfigurabile. Noţiunea în sine este destul de veche: primul prototip a fost introdus de firma Xilinx în 1986, deci cu 12 ani în urmă. O serie de conjuncturi au ţinut însă zona în umbră faţă de atenţia marilor companii şi a cercetătorilor. Brusc însă, în urmă cu cîţiva ani, o creştere a activităţii orientate spre acest subiect a putut fi observată. Principalul motiv este tehnologia.

1 Tehnologia

În anii '60 Gordon Moore a observat o interesantă lege de evoluţie a densităţii şi performanţei circuitelor integrate (cele două sunt strîns legate: cu cît un circuit este mai mic cu atît poate fi făcut să funcţioneze mai repede, pentru că semnalele electrice trebuie să circule pe distanţe mai scurte). Moore a pronosticat că densitatea circuitelor se dublează la fiecare 18 luni. Evident, această evoluţie nu poate ţine la infinit, pentru că după o vreme dimensiunile componentelor ar deveni sub-atomice. Dar cît de mult se poate merge cu această rată de îmbunătăţire a tehnologiei? Ei bine, experţii s-au înşelat în numeroase rînduri prezicînd încetinirea evoluţiei: şi la ora actuală legea este valabilă. Se estimează că în următorii cinci–zece ani rata de evoluţie va putea fi menţinută.

Evoluţia tehnologiilor semiconductoarelor însă dă naştere la o creştere în salturi a celorlalte tehnologii. De exemplu, pentru că cipurile de memorii RAM sunt pătrate[1], capacitatea lor în general creşte în multipli de 4 (se dublează lăţimea şi lungimea). Deci în loc să se dubleze la fiecare 18 luni, RAM-urile se cvadruplează în capacitate la fiecare 3 ani.

Un salt mai spectaculos este evident în următorul exemplu: la începutul anilor '70 densitatea a devenit suficient de mare pentru a permite plasarea tuturor componentelor unei unităţi centrale pe un singur integrat. Aşa a apărut microprocesorul (primul microprocesor, 4004, a fost creat de firma Intel). La începutul anilor '80 aceeaşi creştere a permis înghesuirea pe o singură pilulă de siliciu a tuturor tranzistorilor (între 25.000 şi 50.000) pentru un microprocesor complex pe 32 de biţi. Atunci cînd poţi pune toate părţile la un loc de fapt saltul calitativ este mult mai mare, pentru că legăturile dintre circuite se reduc substanţial, şi performanţa creşte brusc.

La ora actuală cele mai complexe microprocesoare (Pentium II de pildă) au peste 15,000,000 de tranzistori! Cu aceeaşi rată de evoluţie, în acelaşi spaţiu în 2005 vor încăpea peste 1 miliard de tranzistori.

Tot evoluţia tehnologică a scos şi circuitele reconfigurabile, despre care vorbeam mai sus, din anonimat. Subit ele au devenit suficient de mari şi de complicate pentru a putea implementa în mod eficient aplicaţii interesante.

Două motive fac deci interesant studiul circuitelor reconfigurabile:

- Circuitele hardware reconfigurabile sunt acum suficient de mari pentru a fi folosite în aplicaţii utile;

[1]Asta are de-a face cu modul în care DRAM-ul funcţionează: jumătate din biţii de adrese selectează o *linie*, iar cealaltă jumătate din biţii de adrese selectează un cuvînt de memorie din linie.

- Proiectanţii de microprocesoare ai viitorului vor avea la dispoziţie o cantitate uriaşă de tranzistoare pe care vor trebui să le folosească pentru a creşte eficienţa. O parte dintre aceste tranzistoare vor putea fi utilizate pentru a implementa circuite reconfigurabile.

Ce este deci hardware-ul reconfigurabil? Înainte de a răspunde la această întrebare voi face o scurtă şi incompletă trecere în revistă a celorlalte paradigme hardware folosite. Prin contrast vom înţelege mai bine care sunt calităţile şi lipsurile noii generaţii de circuite.

2 Hardware-ul tradiţional

Putem distinge două feluri de circuite: care pot fi programate şi care nu. Din prima categorie fac parte microprocesoarele, din a doua circuitele integrate specifice unei aplicaţii (ASIC: Application Specific Integrated Circuits). Vom vedea şi nişte categorii intermediare.

2.1 Microprocesoare

Un microprocesor este practic o colecţie de unităţi funcţionale plasate pe ceea ce se numeşte "data-path" (cărare a datelor?). Simplificată la maximum, funcţionarea unui procesor poate fi descrisă în următorii termeni:

1. Procesorul citeşte din memorie următoarea instrucţiune de executat;
2. Procesorul decodifică instrucţiunea: sunt trimise semnale de comandă spre feluritele unităţi funcţionale. Datele de prelucrat sunt extrase de unde se află (probabil din regiştrii) şi trimise spre una dintre unităţile funcţionale (de exemplu, dacă trebuie executată o instrucţiune de adunare, datele sunt trimise la unitatea care face adunări);
3. Unitatea funcţională activă prelucrează datele şi generează rezultatul;
4. Rezultatul este trimis la destinaţia sa (posibil un registru);
5. Ciclul se reia de la pasul 1.

Să observăm următoarele caracteristici ale unui microprocesor, pe care le vom contrasta apoi cu cele ale hardware-ului reconfigurabil:

- Procesorul are în interior un număr relativ limitat de unităţi funcţionale. Orice procesare de date trebuie exprimată în termenii operaţiilor pe care aceste unităţi funcţionale le pot executa.
- Toate operaţiile se execută pe date de aproximativ aceeaşi mărime, numită "cuvînt" (unele procesoare pot efectua prelucrări pe mai multe dimensiuni, cum ar fi 8, 16 şi 32 de biţi, dar oricum este vorba de un set foarte limitat de dimensiuni).
- Teoretic, pentru fiecare program am putea construi un procesor special, care execută numai acest program: dacă programul nostru are două adunări independente am putea construi un procesor cu două unităţi aritmetice. Microprocesoarele nu sunt astfel construite: ele *refolosesc* aceeaşi unitate; o *multiplexează* în timp. Toate adunările sunt făcute una după alta, cu aceeaşi unitate aritmetică.

- Un procesor nu poate executa foarte multe instrucţiuni simultan. Procesoarele moderne beneficiază de două tehnici care sporesc viteza de execuţie:
  - *Pipelining:* un procesor poate avea simultan mai multe instrucţiuni succesive în curs de execuţie, în faze diferite. Să zicem că am acasă o maşină de spălat şi una de uscat. Atunci pot usca o încărcătură de rufe în timp ce spăl alta. Practic pot face de două ori mai multe rufe în acelaşi timp decît în cazul în care am o singură maşină care face amîndouă operaţiile. Asta se cheamă "pipelining"; toate procesoarele moderne folosesc această tehnică: în timp ce o instrucţiune este decodificată, alta este executată şi alta este citită, etc.
  - *Multiple issue:* un procesor are de obicei mai multe unităţi funcţionale; de pildă înmulţitorul este diferit de sumator. Atunci procesorul poate lansa în execuţie mai multe instrucţiuni care folosesc unităţi diferite simultan. Astfel de procesoare se numesc "superscalare".

  Cu toate acestea, cantitatea de paralelism disponibilă unui procesor este relativ limitată. Un procesor Pentium II poate teoretic lansa în execuţie 4 instrucţiuni simultan, dar în medie, măsurînd performanţa pe programe mari, reuşeşte să execute doar în jur de 2 pe fiecare ciclu de ceas (din felurite motive, cum ar fi instrucţiuni succesive care vor aceeaşi unitate funcţională).

Un microprocesor trebuie să fie "bun la toate" şi nu este excelent pentru nimic. Anumite tipuri de calcule nu pot fi făcute cu procesoare din această cauză; este nevoie de ASIC-uri.

2.2 ASIC-uri

Atunci cînd ştii exact ce ai de calculat şi timpul disponibil este foarte puţin, nu ai de ales altceva decît să proiectezi un circuit care face numai acel lucru. Un astfel de integrat, construit să rezolve o singură problemă, se numeşte ASIC.

Un exemplu tipic de ASIC este cipul DES care criptează cu algoritmul de criptografie cu cheie privată Data Encryption Standard. Într-un PC se găsesc o grămadă de ASIC-uri, de la controlerele de magistrală la controlerele perifericelor. (Într-un PC se pot găsi şi mai multe procesoare; unele periferice sunt controlate de procesoare al căror program este fixat odată pentru totdeauna.)

Un ASIC este mult mai simplu, mai ieftin şi mai rapid decît un microprocesor. Dar nu este deloc flexibil: nu poate rezolva decît o singură problemă.

2.3 Microcod; circuite configurabile

Atît microprocesoarele[2] cît şi ASIC-urile sunt construite din fabrică şi aşa rămîn pînă la moarte. Există însă nişte clase de circuite care pot fi modificate de către utilizator, la o scară redusă.

Procesoarele cu set de instrucţiuni complex (CISC: Complex Instruction Set Computer) sunt adesea implementate folosind în interior *microcod*. Practic acţiunile indicate mai sus: decodificarea instrucţiunii, execuţia, scrierea rezultatului, etc., sunt mult mai complexe pentru un CISC, şi sunt fiecare descompuse la rîndul lor în instrucţiuni mai simple. Pentru fiecare pas se execută deci un progrămel, numit *microprogram*, care indică fiecărei unităţi funcţionale ce să facă. Microprogramul

[2] Această aserţiune este valabilă pentru procesoarele RISC; procesoarele CISC sunt de obicei microprogramate.

este de obicei înscris într-o memorie ROM din interiorul procesorului. 'Procesoarele CISC mai vechi uneori permiteau utilizatorilor să schimbe microcodul în timpul funcţionării, permiţînd sinteza de noi instrucţiuni, care nu existau la fabricarea procesorului. Libertatea era însă relativ limitată. (Microprocesorul Pentium Pro permite “microcode patching”: peticirea microcodului. Aceasta este o tehnologie introdusă de Intel după apariţia celebrului bug din Pentium, care efectua greşit împărţirea. Algoritmul de împărţire era scris în microcod şi avea un bug. Acum Intel poate corecta astfel de erori chiar după ce procesorul a fost livrat utilizatorilor, prin software. Acest subiect însă depăşeşte cadrul acestui articol.)

O altă varietate de circuite configurabile conţine, printre altele, clasa numită PLA: Programmable Logic Array. Acest tip de circuite poate fi modificat “off-line” (adică nu în timp ce funcţionează). Un PLA tipic conţine un rînd de porţi SAU şi un rînd de porţi ŞI. Fiecare ieşire de la o poartă SAU este conectată la fiecare intrare la o poartă ŞI, avînd astfel o matrice de conexiuni. Cel care foloseşte circuitul poate suda sau desface unele dintre conexiuni, obţinînd felurite funcţionalităţi. Pentru că orice funcţie logică se poate exprima folosind o conjuncţie de disjuncţii, astfel se poate (teoretic) sintetiza orice circuit. Desigur că metoda nu este practică pentru circuite complicate, dar este folosită cîteodată pentru a sintetiza “glue logic”: circuite care fac legătura între alte circuite.

Motivul pentru care PLA-urile sunt importante este unul *economic*: desigur că am putea fabrica un ASIC pentru fiecare funcţionalitate a unui PLA, dar atunci ar costa mult mai mult. Toate PLA-urile sunt de fapt la fel, deci sunt fabricate de o singură linie tehnologică, pe cînd fiecare ASIC care calculează o altă funcţie este diferit, deci are nevoie de propria lui linie de fabricaţie.

3 Hardware reconfigurabil

Am văzut deci că hardware-ul tradiţional are în general o funcţionalitate fixată de la bun început de către fabricant; această funcţionalitate poate fi controlată şi multiplexată în timp, ca la microprocesoare, pentru a sintetiza funcţii noi, sau poate fi fixată odată pentru totdeauna, ca la ASIC-uri. Anumite genuri de circuite pot fi făcute să facă lucruri pentru care nu au fost proiectate (ca procesoarele cu microcod), sau pot fi schimbate cu un efort considerabil (ca PLA-urile).

Ce-ar fi însă dacă am avea un circuit a cărui funcţionalitate să poată fi schimbată la fel de repede pe cît funcţionează (spre deosebire de PLA, la care treaba asta durează minute sau ore), şi care să permită o flexibilitate totală (spre deosebire de microcod, care poate doar controla unităţile funcţionale existente)? Ce-ar fi dacă circuitele astea s-ar putea configura prin aceleaşi mecanisme folosite pentru a le folosi (spre deosebire de anumite memorii EEPROM care sunt şterse prin expunere la ultraviolete)? Ce-ar fi dacă am avea un circuit care ar putea să-şi schimbe funcţionalitatea în timp ce lucrează?

Există aşa ceva? Da; se numeşte hardware reconfigurabil.

3.1 Caracteristici fizice

Un circuit hardware reconfigurabil constă din două entităţi distincte: porţi logice şi sîrme care leagă porţile. Spre deosebire de circuitele VLSI obişnuite, funcţionalitatea unei porţi nu este fixată, ci poate fi schimbată dinamic. De asemenea, sîrmele se întind de-a lungul circuitului, iar la fiecare intersecţie se află un mic comutator. Fiecare comutator poate fi închis sau deschis prin software. Un circuit reconfigurabil generic arată cam ca în figura 1.

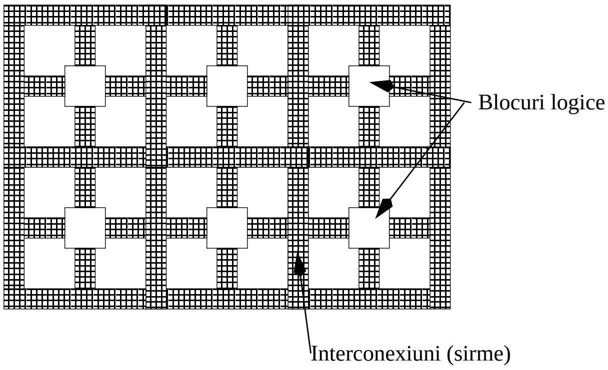

Figura 1: Circuit reconfigurabil

Tehnologic problema este foarte simplă; ambele dispozitive se pot face reconfigurabile folosind mici celule de memorie.

3.1.1 Poarta universală

O poartă logică "universală" cu două intrări de 1 bit şi o ieşire de 1 bit se poate sintetiza folosind o memorie cu capacitatea de 4 cuvinte de 1 bit. Cele două intrări sunt adresa cu care se indexează în memorie, iar conţinutul memoriei este ieşirea. Iată de exemplu care cum se pot sintetiza felurite porţi în acest fel:

	Conţinutul memoriei		
adresa	$A \vee B$	$A \wedge B$	$\neg A$
0	0	0	1
1	1	0	0
2	1	0	1
3	1	1	0

E clar cum lucrează? De pildă, dacă vrem să folosim memoria ca o poartă SAU ($\vee$) o să încărcăm în cei 4 biţi valorile 0, 1, 1, respectiv 1. Atunci cînd vrem să calculăm "$1 \vee 0$", primul bit la intrare va fi 1, iar al doilea 0. Asta formează la un loc adresa 10 (2 în baza 2); cuvîntul de memorie de la adresa 2 este 1, deci $1 \vee 0 = 1$. Corect.

În terminologia hardware-ului reconfigurabil, o astfel de memorie folosită pentru a calcula valoarea unei funcţii booleene se numeşte, din motive evidente, "tabelă de căutare", *Look-Up Table*, LUT. Blocurile logice folosite în circuitele comerciale conţin mai multe astfel de "porţi".

3.1.2 Switch-ul

Cu ajutorul LUT-urilor putem schimba funcţionalitatea unei porţi logice. Cu ajutorul comutatoarelor (switch) putem schimba interconexiunile. Imaginaţi-va o grilă formată din "sîrme", iar la fiecare intersecţie de sîrme un tranzistor, cu emitorul pe o sîrmă şi cu colectorul pe cealaltă. Baza tranzistorului este cuplată la o memorie care conţine un singur bit. Dacă în memorie punem un "1", atunci tranzistorul va conduce electric între emitor şi colector, realizînd o conexiune între cele două sîrme. Dacă baza este la “0”, atunci sîrmele sunt izolate electric una de cealaltă. Figura 2 ilustrează cum se poate conecta ieşirea unei porţi la intrarea alteia.

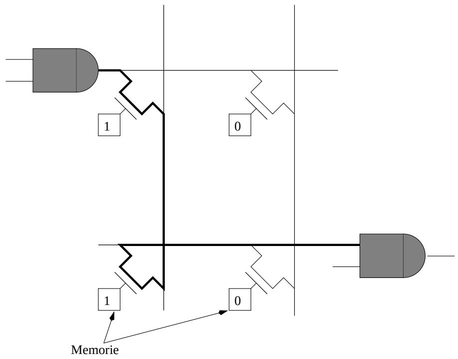

Figura 2: Conexiunile între sîrme pot fi făcute şi desfăcute cu ajutorul unui tranzistor controlat de o memorie de un bit.

Putem deci configura foarte simplu legăturile dintre elementele de procesare, folosind un bit pentru fiecare intersecţie.

Aceste două elemente reconfigurabile, “porţile” şi comutatoarele, fac întregul circuit extrem de flexibil. Prin simpla operaţiune de scriere într-o memorie putem schimba dinamic funcţionalitatea circuitului! Din această cauză voi numi astfel de circuite *re*configurabile, prin contrast cu circuitele configurabile: putem oricînd să le schimbăm cu uşurinţă configuraţia, într-un timp relativ scurt.

La ora actuală există mai multe firme care produc circuite integrate reconfigurabile comerciale. Cele mai importante sunt, în ordinea volumului de circuite de acest gen: Xilinx, Altera, AT&T, Actel, Quicklogic. Fiecare din aceste firme produce mai multe tipuri de circuite, folosind tehnologii diferite, cu diferite complexităţi ale tabelelor, cu mici bucăţele de hardware specializat vîrîte printre porţile generice, etc. Ideea de bază însă este cea descrisă mai sus.

Denumirea acestui gen de circuite nu este încă bine împămîntenită; termenul care capătă însă cea mai mare acceptare este cel de “matrici de porţi logice programabile cu ajutorul cîmpului [electromagnetic]”: Field Programmable Gate Arrays, FPGA.

Înainte de a vedea care sunt deficienţele circuitelor reconfigurabile şi problemele care sunt de rezolvat de către cercetare în ceea ce le priveşte, să aruncăm o privire la domeniile în care aceste circuite sunt foarte necesare.

3.2 Sisteme cu hardware reconfigurabil

Trebuie de la bun început înţeles că hardware-ul reconfigurabil nu îşi propune să înlocuiască celelalte paradigme de calcul, ci să le *suplimenteze*. Locul unităţilor reconfigurabile este alături de cele clasice (procesoare, ASIC-uri), pentru a executa acea parte din treabă care li se potriveşte cel mai bine. Există mai multe propuneri de utilizare a acestui tip de hardware, care diferă prin gradul de integrare cu sistemul. Iată unele dintre posibilităţi enumerate în ordine descrescătoare a integrării:

Unitate funcţională: fiecare microprocesor ar putea avea, pe lîngă unitatea aritmetică şi logică, o unitate a cărei funcţiune este dinamic configurabilă. Instrucţiuni speciale ar configura unitatea şi alte instrucţiuni ar folosi-o.

Coprocesor: aşa cum înainte de 80486 Intel avea un procesor principal şi un coprocesor, putem să ne imaginăm o unitate reconfigurabilă plasată pe aceeaşi magistrală şi comandată de procesorul principal. Modul de folosire este asemănător cu cel precedent, dar timpii de comunicare între module sunt mai mari.

Procesor independent: putem să ne imaginăm unitatea reconfigurabilă ca fiind un master de magistrală autonom pe lîngă microprocesor, capabil să acceseze memoria, să preia date şi să le prelucreze. Microprocesorul central ar iniţia doar acţiunile sale, indicînd de exemplu unde în memorie se află configuraţia. Unitatea reconfigurabilă ar citi configuraţia, s-ar configura singură şi apoi ar începe să execute aplicînd operaţii asupra unor date a căror adresă a fost indicată tot de procesorul central.

Sistem complet de calcul autonom: pentru unele aplicaţii (destul de puţine) se pot construi sisteme complete de calcul bazate pe circuite reconfigurabile; unele exemple apar mai jos.

3.3 Aplicaţii

Putem distinge două clase de aplicaţii: unele care în mod tradiţional sunt rezolvate folosind ASIC-uri, şi altele care necesită procesoare. Iată ce poate "fura" de la fiecare din ele noua tehnologie:

3.3.1 FPGA versus ASIC

Trebuie dinainte spus că dacă putem implementa aceeaşi soluţie pe un ASIC şi pe un circuit reconfigurabil, soluţia cu ASIC-ul va fi mai rapidă în execuţie. Motivele sunt multiple, şi le vom explora în secţiunea consacrată deficienţelor. Cu toate acestea circuitele reconfigurabile nu sunt deloc inutile! Marile lor avantaje faţă de ASIC-uri sunt preţul scăzut şi reconfigurabilitatea.

Iată deci genurile de aplicaţii la care se pretează excelent hardware-ul reconfigurabil:

Produse de serie mică. Costul construirii unei linii de fabricaţie pentru ASIC-uri este foarte mare, pentru că fiecare ASIC cere o serie de dispozitive unice. În schimb toate FPGA-urile sunt la fel; funcţionalitatea uneia este programată ulterior, prin software.

Dezvoltare de ASIC-uri. Proiectarea şi depanarea unui ASIC este o treabă deosebit de complicată[3]. Testarea mai ales este dificil de făcut. Desigur, nimeni nu-şi permite să fabrice un ASIC numai ca să găsească un bug şi să schimbe apoi întreaga linie de fabricaţie. Metoda uzual folosită este simularea în software, dar aceasta este extrem de lentă (de mii de ori mai lentă decît viteza de funcţionare a circuitului real). O soluţie foarte eficientă este de a implementa schema ASIC-ului pe un FPGA şi de a-l rula astfel. Raportul de viteză este cam de genul 1/2. În plus, orice bug se poate foarte uşor corecta schimbînd configuraţia. Există o firmă, Quickturn, care cu asta se ocupă: îţi vinde plăci mari cu sute de FPGA-uri şi software care ia descrierea unui circuit într-un limbaj de descriere hardware (HDL) şi construieşte automat o versiune a circuitului folosind FPGA-uri. Rezultatul apoi poate fi "înfipt" în locul unde ar trebui să fie circuitul real şi poate fi folosită pentru simulare.

[3]Proiectarea, descrierea şi simularea de circuite hardware se face folosind o clasă specială de limbaje, numite HDL: Hardware Description Languages. Cele mai răspîndite limbaje de acest gen sunt VHDL (VLSI HDL) şi Verilog.

“Rapid prototyping”. Construirea unei linii de fabricaţie de ASIC-uri durează cîteva săptămîni în cel mai bun caz; un FPGA poate fi reconfigurat în milisecunde. Atunci cînd nu-ţi poţi permite să aştepţi, hardware-ul reconfigurabil poate fi singura soluţie fezabilă.

Toleranţă la defecte. Dacă o parte din circuitul reconfigurabil se defectează, putem reconfigura o altă parte pentru a-i prelua funcţionalitatea: orice LUT poate calcula orice funcţie. Există clase de circuite reconfigurabile care se auto-verifică şi se auto-reconfigurează în cazul unei malfuncţii, ocolind partea cu defecte, pe care pur şi simplu nu o folosesc! Acest gen de aplicaţii abia emerge, dar este foarte promiţător. NASA se interesează cu precădere de astfel de aplicaţii; este destul de greu să faci service pe un satelit...

Aplicaţii care se schimbă. Cum spuneam, un ASIC rezolvă întotdeauna o singură problemă. Dacă anticipăm că problema se va schimba în timp (gîndiţi-vă de pildă la “upgrades” pentru modemuri, telefoane celulare, etc.) trebuie să folosim o soluţie flexibilă. Dacă problema nu este prea complicată, atunci este sinucidere financiară să folosim un microprocesor pentru a o rezolva; un circuit reconfigurabil va fi aproape întotdeauna mai ieftin. Şi vom vedea mai jos că un circuit reconfigurabil are potenţialul de a avea o performanţă net superioară microprocesorului, pentru că microprocesorul este extrem de general, însă hardware-ul poate fi special reconfigurat pentru problema pe care vrem să o rezolvăm.

Algoritmi sistolici. Hardware-ul reconfigurabil pare a fi ţinta ideală pentru a implementa anumite clase de algoritmi masivi paraleli. De exemplu, algoritmii sistolici (în care o mulţime de elemente de procesare funcţionează sincron şi pasează date pentru prelucrat de la unul la altul) sau automatele celulare (în care o mulţime de procesoare foarte simple operează independent şi comunică cu vecinii în mod asincron) par a fi făcute pentru acest mediu. Simetria circuitelor reconfigurabile ne permite să împărţim circuitul în bucăţele care implementează astfel de automate simple.

3.3.2 FPGA versus procesoare

Hardware-ul reconfigurabil are şansa să bată microprocesoarele în performanţă pentru anumite genuri de aplicaţii. Care sunt acestea? Pentru a răspunde trebuie să înţelegem de fapt unde-şi petrece timpul un program.

Un procesor care rulează la 300Mhz şi lansează în medie 2 instrucţiuni pe ciclu (tipic pentru un Pentium II modern) va executa *600 de milioane* de instrucţiuni pe secundă. Mai mult decît numărul total de instrucţiuni din cel mai mare program scris vreodată. Este clar, timpul este petrecut re-executînd o serie de instrucţiuni de multe ori. De fapt orice program are undeva o buclă în care-şi petrece majoritatea timpului. O regulă empirică spune că 90% din timp se execută 10% din cod.

Asta înseamnă că dacă reuşim să optimizăm acele 2% din program pentru a mearge de 2 ori mai rapid, întreaga durată a programului se reduce aproape la jumătate! Orice cîştig de performanţă în “cea mai interioară buclă” (innermost loop) este foarte important.

Dacă vă uitaţi la caracteristicile procesoarelor enumerate mai sus, vedeţi care sunt exact caracteristicile care pot fi exploatate de un circuit reconfigurabil:

Dimensiuni neobişnuite ale datelor. Un procesor RISC o să facă adunări pe 32 de biţi chiar dacă trebuie să adune numere pe 3 biţi. Folosind hardware reconfigurabil putem însă genera sumatoare de orice dimensiune, adaptate pentru problema pe care vrem să o rezolvăm. Un sumator foarte mic este mult mai rapid; de aici creştere de viteză şi de eficienţă în ocuparea spaţiului.

Instrucţiuni greu sintetizabile de către un microprocesor. Consideraţi o subrutină care trebuie să inverseze biţii dintr-un număr de 32 de biţi: primul să devină ultimul, ş.a.m.d. Cu un procesor obişnuit aceasta se face folosind o buclă şi o serie complicată de instrucţiuni care extrage biţi şi îi "vîră" în cealaltă parte; rezultatul este un program care durează sute de cicli de ceas. În hardware reconfigurabil un astfel de program nu are nevoie de nici un fel de calcule: pur şi simplu tragi nişte sîrme care se încrucişează: prima merge la ultima poziţie, etc. Timp de execuţie: aproape 0!

De fiecare dată cînd un program are nevoie de astfel de operaţii care se sintetizează greu din instrucţiuni primitive ale unui procesor este un candidat bun la implementarea pe un circuit reconfigurabil.

Programe care beneficiază de paralelism masiv. Am văzut mai sus că paralelismul oferit de un procesor este relativ limitat. Uneori însă avem de-a face cu aplicaţii care pot prelucra cantităţi extrem de mari de date într-un mod uniform. Prelucrarea digitală de semnal (sunet, imagine, etc.) este un exemplu. Un FPGA poate cîştiga creînd atîtea unităţi funcţionale cîte încap în hardware, poate zeci!

Programe care pot fi executate în regim "pipeline". Anumite genuri de programe (aplicaţiile vectoriale) aplică o serie lungă de operaţii fiecărei date dintr-un set mare. Putem sintetiza toate operaţiile în hardware şi putem aplica simultan operaţii diferite unor date diferite.

Sinteza unor circuite specifice unei instanţe (generare dinamică de cod). Această metodă de creştere a eficienţei este disponibilă parţial şi procesoarelor obişnuite. Să ne gîndim la un program de criptare. Criptarea se face în funcţie de o cheie, pe care utilizatorul o introduce. După ce utilizatorul introduce cheia, aceasta va rămîne constantă pentru toată durata criptării. Algoritmul poate beneficia de forma cheii pentru a deveni mai eficient. De exemplu dacă ultima cifră din cheie este 0, toate înmulţirile cu acea valoare pot fi sărite.

Această tehnologie este un subiect foarte fierbinte în teoria compilatoarelor, şi se numeşte *dynamic code generation*: generezi cod abia atunci cînd ştii maximum de informaţie. Compilatoarele de tip JIT (Just In Time), sau Hot-Spot de la Sun de pildă, folosesc chiar această tehnică.

Metoda este cu atît mai eficientă în cazul circuitelor reconfigurabile, pentru că libertatea este mult mai mare decît în cazul unui procesor în ceea ce priveşte "cărămizile" de bază.

La ora actuală cele mai rapide cipuri de criptare sunt construite folosind hardware reconfigurabil, tocmai pentru că folosesc toate tehnicile de mai sus: după ce cheia este cunoscută se generează un circuit special care criptează cu o singură cheie, şi care este extrem de eficient.

Alt exemplu interesant: să considerăm setul de instrucţiuni MMX introdus de Intel pentru procesoarele sale. MMX un set de instrucţiuni foarte specializat, destinat procesării de semnale pentru aplicaţii multimedia. Problema principală a lui Intel este de compatibilitate: lumea se fereşte să scrie programe folosind instrucţiuni MMX pentru că acestea nu se pot executa pe procesoare care nu au această extensie (sau se pot executa numai foarte lent, folosind emulare software).

MMX sunt exact instrucţiuni care se pot implementa foarte simplu şi eficient în hardware reconfigurabil: se aplică la date de dimensiuni bizare (de obicei pe cîte 8 biţi), sunt destinate unor procesări vectoriale, care poate fi de obicei eficient tradusă în regim de pipe-line. Dacă Intel ar fi avut nişte hardware reconfigurabil în Pentium, nu ar mai fi avut grija compatibilităţii şi eficienţei instrucţiunilor MMX: oricine le putea sintetiza automat la momentul execuţiei, şi eventual, dacă se dovedea că unele dintre instrucţiuni nu erau prea folosite, puteau fi scoase din configuraţie, fără a schimba nimic în linia de fabricaţie a chipului! (Motorola a introdus de curînd o serie de procesoare care conţine astfel de hardware.)

3.4 Ineficienţe; subiecte de cercetare

Să nu credeţi că FPGA-urile sunt un panaceu; există o grămadă de probleme caracteristice numai lor; unele pot fi rezolvate, altele sunt probabil intrinseci:

Viteza de propagare a semnalului este cea mai neplăcută problemă. O sîrmă într-un ASIC este un obiect aproape demn de ignorat; timpul de propagare printr-un switch de la o încrucişare într-un FPGA este însă semnificativ. Propagarea dă măsura tactului de ceas care poate fi folosit. Dacă întîmplător un semnal trebuie să traverseze mai multe încrucişări, am zbîrcit-o. Viteza unui program este deci dependentă de plasamentul fizic, care este greu de controlat de către programator.

Sîrmele: din cauză că nu se ştie dinainte ce cu ce este legat, un FPGA trebuie să conţină o mare cantitate de sîrme, pentru a permite tot felul de combinaţii posibile. Peste 90% din suprafaţa unui FPGA este ocupată de sîrme, dintre care multe rămîn nefolosite. Asta înseamnă eficienţă scăzută şi densitate efectivă mică de porţi. Cel mai mare FPGA are de ordinul a sute de mii de porţi, la aceeaşi suprafaţă cu un procesor. De aici o serie întreagă de constrîngeri pentru proiectanţii de aplicaţii.

Compilatoarele ineficiente sunt probabil cel mai dificil obstacol de trecut pentru acceptarea pe scară largă a fenomenului. La ora actuală lumea proiectează programe pentru circuite reconfigurabile aşa cum proiectează hardware: folosind limbaje HDL şi scule CAD (Computer Aided Design) pentru aranjare pe circuit. Acest procedeu este mult mai dificil decît proiectarea în limbaje de nivel înalt, şi în general compilatoarele sunt mult mai lente (pentru a sintetiza programele de mii de linii durata poate fi de ordinul orelor!).

O altă dificultate majoră este proiectarea de programe care folosesc simultan microprocesorul şi o unitate reconfigurabilă auxiliară. Cu tehnologia curentă, programatorul trebuie atunci nu numai să scrie programele pentru ambele procesoare, ci şi să aibă grijă de comunicaţia dintre programul de pe procesor şi cel de pe circuitul auxiliar. Programatorii nu sunt obişnuiţi să gîndească în astfel de termeni.

În al treilea rînd avem o problemă de inadecvare a limbajelor. Un limbaj de nivel înalt, ca C-ul, exprimă uneori concepte care nu sunt uşor sau eficient de modelat pe un circuit reconfigurabil. Luînd exemplul de mai sus, cu inversarea biţilor, un programator nici nu poate scrie aşa ceva în C decît folosind o buclă. Cum ar putea atunci un compilator pentru FPGA să observe că utilizatorul vrea de fapt doar să mişte nişte sîrme? Nu e un lucru evident nici pentru un om. Încă nu este clar ce fel de limbaje ar trebui folosite în domeniu.

Mărime limitată: pentru calculatoarele obişnuite, mecanisme ca memoria virtuală şi RAM mare au făcut o non-problemă mărimea unui program. Resursele unui circuit reconfigurabil sunt însă încă foarte limitate şi foarte vizibile pentru utilizator. Compilarea unui program pentru un FPGA poate eşua de pildă dacă nu sunt destule sîrme într-o anumită zonă pentru a conecta două porţi, chiar dacă circuitul sintetizat este relativ mic faţă de cantitatea totală de resurse disponibile.

3.5 Prototipul CMU

Voi încheia acest text cu descrierea sumară a cercetării care se desfăşoară la Universitatea Carnegie Mellon. Proiectul are o pagină de web, dar deocamdată acp;p se află destul de puţină informaţie pentru că încă nu avem rezultate substanţiale[4]. Puteţi ajunge acolo folosind o legătură din pagina mea personală de web.

Carnegie Mellon are două grupuri care lucrează la un prototip special de circuit reconfigurabil numit *PipeRench*. Asta vine de la “pipe”: ţeavă (subliniind natura “pipelined” a circuitului) şi “wrench”: cheie franceză. Circuitul acesta se vrea o unealtă generală pentru a scrie aplicaţii “pipelined”. Unul dintre grupuri lucrează la proiectarea, depanarea şi construirea unui circuit prototip şi a interfeţei sale cu un sistem de calcul (bazat pe o magistrală PCI), iar celălalt la un compilator care să genereze cod pentru PipeRench. Între cele două grupuri există o oarecare independenţă, în sensul că, deşi pentru moment compilatorul dezvoltat are ca singură ţintă prototipul PipeRench, în mod ideal ar trebui să fie uşor portat şi pentru alte feluri de circuite reconfigurabile, cum ar fi cele comerciale.

De la început au fost făcute o serie de decizii în spaţiul posibilelor design-uri, care limitează aplicabilitatea, dar fac problema tractabilă. Ca oricînd în cercetarea ştiinţifică, nu este clar de la început că aceste decizii au fost cele mai bune; numai viitorul va arăta dacă această cărare este profitabilă, sau dacă rezultatele vor fi foarte specifice.

Voi discuta pe scurt unele dintre deciziile mai interesante, pentru a ilustra natura unora dintre problemele întîlnite şi soluţiile oferite.

3.5.1 Aplicaţii vectoriale

Principalele aplicaţii care sunt ţinta implementării pe PipeRench sunt cele care aplică o aceeaşi procesare relativ complicată asupra unui set mare de valori (unui vector). Exemple ideale: compresia, criptarea, codificarea, filtrarea de semnal, analiza statistică, etc. PipeRench va implementa atunci funcţiunea care trebuie aplicată repetat şi datele vor fi trimise una cîte una spre el; la ieşire se vor culege rezultatele.

3.5.2 Un coprocesor master de magistrală

PipeRench are un controler special care este capabil să genereze adrese pentru magistrală pentru a citi şi scrie singur din memorie. Controlerul are mai multe scopuri:

[4] Multe alte universităţi au proiecte de acest gen; legături spre unele dintre proiectele lor se pot găsi tot în această pagină de web.

- Controlerul citeşte chiar informaţia de configurare pentru PipeRench din memorie. Cînd procesorul principal vrea ca PipeRench să facă ceva anume, îi trimite doar un semnal şi adresa în memorie unde se află configuraţia de încărcat. PipeRench apoi se încarcă singur.

- Controlerul lui PipeRench este capabil să citească de unul singur vectori din memorie şi să-i proceseze. Procesorul central îi spune lui PipeRench unde sunt datele, cîte şi cum sunt aranjate, iar PipeRench aplică întregul program fiecăreia dintre date, în regim de pipeline.

- Controlerul manipulează de asemenea un cache intern circuitului, în care configuraţiile mai des folosite sunt ţinute, pentru a evita accesul frecvent la magistrala calculatorului şi la memoria lentă.

3.5.3 Hardware virtualizat

Probabil cea mai interesantă decizie este aceea de a "virtualiza" hardware-ul. Aceasta simplifică enorm scrierea de programe, pentru că utilizatorul nu mai trebuie să se preocupe de resursele disponibile. Ideea este asemănătoare cu cea de la memoria virtuală:

- Utilizatorul scrie un program pentru o maşină cu mărime nelimitată;

- Programul este împărţit în felii "virtuale"; să zicem că sunt m astfel de felii;

- Hardware-ul disponibil poate cuprinde numai cîteva din aceste felii, să zicem $n < m$ (cazul $n \geq m$ este un caz particular);

- Primele n felii din cele m sunt încărcate şi se execută. Cînd vine rîndul execuţiei feliei virtuale $n + 1$, din hardware este scoasă felia 1 şi înlocuită cu $n + 1$.;

- La fiecare ciclu de ceas ulterior o nouă felie virtuală este scoasă şi înlocuită cu alta.

În felul acesta circuitul hardware "alunecă" (ca în figura 3) peste program. Figura ilustrează primii 6 paşi din procesarea unui vector `v[]` de valori. Fiecare valoare a vectorului trebuie procesată de un program în 4 paşi. Fiecare "felie" virtuală este un pas din program. Datele intră în felia 0, trec apoi prin 1, 2 şi ies din 3, după care sunt scrise în memorie. Aceeaşi operaţie este aplicată tuturor elementelor din vector. Pentru figură deci $m = 4, n = 2$.

Metoda are o sumedenie de avantaje. De exemplu, atunci cînd apare un circuit mai mare pe piaţă, programele se vor executa mai repede, pentru că vor avea mai mult hardware la dispoziţie. Nu trebuie deloc recompilate. În plus, putem executa azi (mai lent) programe pentru care nu vom avea destul hardware decît mîine.

3.5.4 O arhitectură "pipe-line"

După cum vedeţi fiecare felie lucrează la un moment asupra altor date de intrare; la fiecare m/n paşi se produce un rezultat.

Din cauză că felia 1 este scoasă cînd apare $n+1$ e clar că datele pot curge numai într-o direcţie în acest circuit. Acest model este asemănător cu al maşinilor "data-flow" (subiectul depăşeşte cadrul acestui articol). În principiu orice aplicaţie poate fi adusă în această formă, dar nu orice aplicaţie este eficient prelucrată astfel.

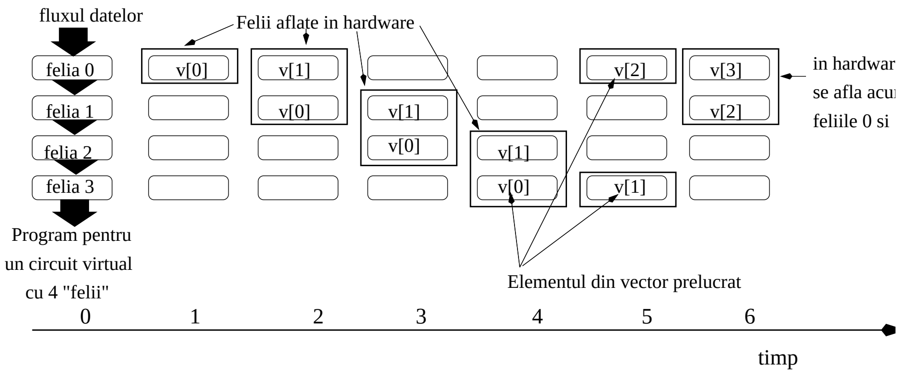

Figura 3: Funcţionarea unui program pentru un circuit cu 4 "felii" virtuale folosind hardware real cu 2 "felii".

3.5.5 Un limbaj intermediar independent de arhitectură

Pentru a uşura sarcina utilizatorilor dezvoltăm un compilator a cărui ţintă este PipeRench. Există foarte puţină experienţă în a scrie compilatoare pentru limbaje imperative gen C pentru astfel de maşini:

- Un compilator clasic de C are la bază cărămizi relativ complicate, din care trebuie să sintetizeze programul (şi anume instrucţiunile procesorului). Un compilator pentru hardware reconfigurabil are nişte elemente mult mai simple (porţi şi sîrme), pe care trebuie să le manipuleze într-un fel eficace.

- Compilatorul pentru PipeRench are sarcini care nu au echivalent în compilatoarele clasice, cum ar fi plasamentul porţilor şi alegerea legăturilor între porţi dintre sîrmele disponibile (Place and Route). În sculele CAD acesta este un proces extrem de complicat care durează foarte mult; ţinta compilatorului nostru este însă să fie comparabil cu alte compilatoare de C.

- Dacă vrea să fie competitiv, compilatorul pentru PipeRench trebuie să folosească calităţile pe care acest gen de circuite le are şi care lipsesc unui procesor. Astfel, trebuie să lucreze cu mărimi de date variabile, să genereze programe "pipelined" şi să fie capabil să folosească capacităţile suplimentare ale hardware-ului (ca în exemplu de mai sus cu inversatul biţilor).

Pentru a fi relativ independent de limbajul sursă (poate va fi Java, poate C, poate ML?) şi de circuitul destinaţie, compilatorul trebuie să folosească nişte abstracţii pentru hardware şi limbajul surşa. Aceste abstracţii trebuie să nu fie atît de generale încît să ascundă complet hardware-ul, pentru că atunci nu se poate obţine eficienţă.

Deocamdată compilatorul nostru ia ca sursă un limbaj intermediar special funcţional (numit DIL: Data Intermediate Language) care descrie eficace calcule pe şiruri de biţi de lungimi arbitrare. Compilatorul traduce programe Dil în asamblare pentru PipeRench. "Gaura" dintre C şi Dil este (sperăm) ceva mai uşor de umplut decît direct dintre C şi asamblare; cum se face asta este subiectul cercetării viitoare.

3.6 Sculele

Iată pe scurt în figura 4 care sunt sculele pe care proiectul le-a dezvoltat şi cu care experimentează tot felul de parametri arhitecturali şi tehnici de compilare.

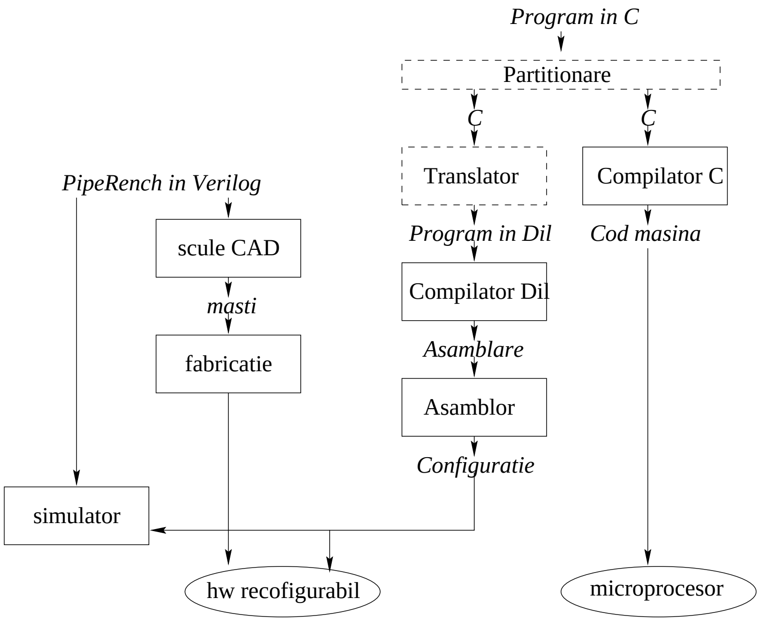

Figura 4: Sculele folosite în proiectul PipeRench

Sculele hardware sunt în partea stîngă a desenului; din programe scrise în Verilog (un HDL) se poate sintetiza fie direct circuitul configurabil, fie se poate simula funcţionarea circuitului.

Sculele desenate cu linie punctată încă nu există. În principiu compilarea va trece prin următorii paşi:

1. Programul sursă este partiţionat în două, separînd instructîunile care vor fi efectuate folosind hardware reconfigurabil;

2. Partea rămasă este compilată cu un compilator ordinar de C;

3. Partea destinată circuitului reconfigurabil este tradusă în limbajul Dil;

4. Compilatorul de Dil compilează programul, optimizează şi plasează operatorii pe circuit; rezultatul este un program într-un limbaj de asamblare special pentru PipeRench;

5. Un asamblor ia programul şi generează configuraţia corespunzătoare.

Configuraţia este apoi oferită circuitului reconfigurabil (sau simulatorului), care se execută în tandem cu procesorul.

La ora aceasta o versiune preliminară a tuturor programelor desenate cu linie continuă există şi este în curs de evaluare. Deşi partea din stînga a figurii pare mult mai mică, este extrem de complicată, pentru că hardware-ul reconfigurabil conţine circuitul propriu-zis, controlerul pentru memorie şi configurare, şi o interfaţă pentru o magistrală PCI care este în proiectare.

4 Concluzii

Hardware-ul reconfigurabil promite un viitor frumos, datorită capacităţii sale de a-şi schimba funcţionalitatea extrem de rapid, poate chiar în timpul funcţionării. Abilitatea de a folosi operaţiuni de bază extrem de simple îl face un candidat pentru a implementa procesări care în mod convenţional nu se pot exprima cu uşurinţă. Capacitatea sa de reconfigurare îi poate da şi alte atribute dezirabile, cum ar fi rezilienţa la erori şi capacitatea de virtualizare a resurselor.

Nu mai este o dilemă dacă circuitele reconfigurabile vor intra sau nu în galeria uneltelor ordinare ale arhitectilor calculatoarelor şi utilizatorilor lor. Singura întrebare este "cînd" se va petrece acest lucru. Răspunsul depinde numai de cei care cercetează în acest domeniu: cît de repede vor reuşi să găsească optica cea mai potrivită şi uneltele cele mai eficace pentru a exploata puterea expresivă a acestor circuite.