

DIRECȚII DE CERCETARE ÎN DOMENIUL SISTEMELOR MULTICORE

Lucian N. Vințan^{1,2}

¹ Universitatea „Lucian Blaga”, Sibiu

lucian.vintan@ulbsibiu.ro, <http://webspaces.ulbsibiu.ro/lucian.vintan>

² Academia de Științe Tehnice din România

www.astr.ro

Rezumat: S-a realizat un studiu asupra impactului arhitecturilor multicore și manycore în cadrul ingineriei calculatoarelor. S-a pornit de la geneza acestor sisteme, determinată de limitele paradigmei monoprosesor. S-au identificat și s-au analizat în mod sistematic, pe baza unei literaturi de specialitate recente, următoarele provocări importante pentru cercetarea și dezvoltarea acestor sisteme: arhitecturi multicore omogene vs. eterogene, exploatarea sinergică a tipurilor de paralelism, ierarhia de memorii cache, coerența și consistența variabilelor partajate, rețele de interconectare, modele de programare paralelă, paralelizarea aplicațiilor, simularea ca instrument de cercetare, benchmarking, explorarea automată a spațiului parametrilor, arhitecturi multicore cu procesări anticipative, putere consumată, disipație termică. Concluzia de bază este că sunt necesare progrese importante în toate aceste domenii, pentru ca proiectarea și utilizarea sistemelor multicore și manycore să fie adecvate. Cu sau fără voia noastră, aceste sisteme vor constitui dispozitivele de calcul universale. Abordarea oricăreia dintre aceste provocări trebuie să fie una de tip holistic. S-a arătat cum vor schimba aceste noi arhitecturi, paradigma științei ingineriei calculatoarelor. În particular, programarea acestor sisteme constituie o provocare majoră, pentru care nu suntem încă pregătiți suficient.

Cuvinte cheie: arhitectura calculatoarelor, sisteme multicore și manycore, sisteme paralele.

Abstract: It is presented the impact brought by the new multicore and manycore systems in the computer engineering field. There were explained the limits of the actual mono-processor paradigm that involved the multicore shift. A state of the art in multicore architecture and compiler research was presented. We identified and analysed some important research challenges in multicores' research and development, like the followings: homogenous vs. heterogeneous multicores, synergistic exploitation of parallelism grains, cache hierarchy, cache coherence and consistency protocols, interconnection networks, parallel programming models, applications' parallelisation methods, simulation, benchmarking, automatic design space exploration, speculative multicore architectures, power consumption and thermal effects. There are necessary important progresses related to all these scientific challenges, in order to adequately develop and program multicore and manycore systems. The research approach must be a holistic one for each of the above topics. It is shown how these systems would change the computer science and computer engineering paradigm.

Keywords: advanced computer architecture, parallel computing, multicore and manycore.

1. Geneza arhitecturilor tip multicore

Arhitecturile de procesoare de tip single-core, care să exploateze în mod agresiv paralelismul la nivel de instrucțiuni prin execuții speculative de tip out of order, trebuie cercetate și dezvoltate în continuare, deși frecvența tactului nu mai poate crește prea mult, din cauza consumului de putere dinamică ($P_d = kCV^2f$) și a disipației termice. La actualele frecvențe de tact de câțiva GHz, densitățile de putere din chip-uri sunt enorme, de câteva sute de W/cm^2 . Supercalculatoarele actuale, cu mii de procesoare, consumă puteri de ordinul MWatt. Totodată, la o frecvență de tact de 14 GHz (70 ps), estimabilă în anul 2014, întârzierea semnalului pe 5 mm de conductor devine enormă (390 ps). Evident că, în aceste condiții, creșterea frecvenței de tact practic nu mai este posibilă, performanța putând crește doar prin inovații arhitecturale. Provocarea esențială aici constă în determinarea compromisului optimal între performanța procesării (Instructions Per Cycle – IPC) și complexitatea arhitecturală (puterea consumată, disipația termică, aria de integrare și bugetul de tranzistori). Astfel, metrici de evaluare de tip MIPS (Million Instructions per Second) per watt, MIPS per area of silicon etc. sunt tot mai frecvent utilizate. Aceste microarhitecturi monoprosesor vor exploata paralelismul la nivel fin din cadrul aplicațiilor cu secvențialitate intrinsecă (scrise în limbaje secvențiale). În acest sens, pot fi avute în vedere inclusiv metode și tehnici de compilare adaptivă a codului obiect, bazate pe informații de tip profilings (tipul algoritmilor, gradul de utilizare al resurselor hardware, nivelul și granularitatea paralelismelor la nivelul thread-urilor etc.). Aceste metode pot determina adaptarea sau reconfigurarea microarhitecturii la cerințele aplicației, în vederea maximizării IPC –ului și minimizării puterii consumate. Și totuși, metodele de exploatare a ILP-ului (Instruction Level Parallelism) au ajuns la o oarecare saturație – ILP Wall (superpipeline cu frecvențe mari de tact, procesarea out of order, branch prediction, trace-cache, procesări speculative, metode de scheduling static, metode de eliminare a memory-wall etc.)

2. Sisteme multicore și manycore

La ora actuală, soluția cea mai frecventă pentru creșterea performanței și evitarea limitărilor anterior schițate, constă însă în dezvoltarea de arhitecturi multicore și manycore. Deși vor face mai dificilă activitatea calculatoriștilor, acestea oferă o rată de performanță/Watt mai bună decât sistemele monoprosesor, la o performanță similară. În plus, aceste sisteme care exploatează paralelismul thread-urilor, au șanse mai mari decât sistemele monoprosesor ca să mai micșoreze din prăpastia de comunicare între microprocesor și memorie (DRAM). Cercetarea în acest domeniu este extrem de necesară având în vedere faptul că în anul 2015 se așteaptă microprocesoare comerciale de uz general de 256 de nuclee. Se consideră că sistemele multicore (eterogene) vor deveni dispozitivul universal de calcul. Se speră că aceste sisteme vor putea corela eficiența procesării cu creșterea densității de integrare, care evoluează conform legii lui Moore. Compania Intel a fabricat deja un chip cu 80 de core-uri integrate, v. <http://techfreep.com/intel-80-cores-by-2011.htm>, pe care îl va lansa în producția de masă în curând. Procesoarele grafice Nvidia Tesla C1060, cu 240 de nuclee integrate în chip, oferă performanțe de până la un Teraflop/s. Procesorul multicore Sony/Toshiba/IBM Cell, cu 9 nuclee neomogene integrate per chip, atinge rate de procesare de 200 Gflop/s. Programarea și utilizarea eficientă, în asemenea cazuri, nu vor fi posibile până când nu vor avea loc schimbări radicale în modelele de programare și în instrumentele software disponibile.

Metricile succesului în cadrul dezvoltării sistemelor multicore se referă, în principal, la productivitatea programării și la performanța aplicației. Performanța trebuie să aibă în vedere minimizarea acceselor la datele aflate în afara memoriilor locale (prin managementul „localității” datelor), optimizarea balansării încărcării procesoarelor și, respectiv, optimizarea comunicațiilor și sincronizărilor. În acest scop, se impune o nouă proiectare a algoritmilor, în vederea mapării lor optimale pe sistemele de tip multicore.

3. Arhitecturi multicore omogene vs. eterogene

Nu putem fi de acord în totalitate cu opinia specialiștilor de la Berkeley care afirmă că nucleele viitoarelor multiprocesoare vor consta în procesoare simple [1]. În acest caz, paralelismul la nivel de instrucțiuni, (singurul) exploatabil la nivelul programelor secvențiale, ar fi mult diminuat. Acest dezavantaj este inacceptabil având în vedere că peste 99% din programele scrise până acum au fost scrise în limbaje secvențiale. Din acest motiv, credem că viitorul, în calculul de uz general, este cel al unor sisteme multicore eterogene (câteva nuclee superscalare out of order cu execuții speculative + multe nuclee superscalare mai simple, de tip in order, cu structuri pipeline scurte, de 5-9 stagii). Eterogenitatea permite adaptarea dinamică la diferitele caracteristici ale programelor rulate. Acest fapt este agreat de multe cercetări recente și este justificabil în baza unui exemplu simplu preluat din [9]. Se presupune, spre exemplu, că $f=10\%$ din timp, un program nu poate fi accelerat prin paralelizare pe un sistem cu $N=100$ de procesoare. Se consideră că, pentru a rula această parte secvențială de două ori mai rapid decât pe un procesor simplu, este nevoie de un procesor complex, care are de 10 ori mai multe resurse (complexitate) decât procesorul simplu. Aplicând legea lui Amdahl, accelerarea obținută pe un sistem omogen cu 100 de nuclee simple este:

$$S = \frac{1}{f + \frac{(1-f)}{N}} = 1 / (0.1 + 0.9/100) = 9.2 \quad (1)$$

Accelerarea obținută pe un sistem neomogen de complexitate echivalentă, având 90 de nuclee simple și un nucleu complex, este superioară:

$$S' = 1 / (0.1/2 + 0.9/90) = 16.7 \quad (2)$$

Totuși, superioritatea sistemelor neomogene față de sistemele omogene, va trebui dovedită mult mai convingător prin simulări complexe atât pe benchmark-uri secvențiale, cât și paralelizate (IPC, energie consumată, buget tranzistori, arie integrare etc.). De altfel, chiar și actualele multiprocesoare comerciale sunt neomogene, având un nucleu superscalar out of order foarte puternic (IBM Cell BE, Intel IXP – procesoare de rețea etc.) și totuși, la nivelul unor manycore – uri eterogene, modelele de programare ar putea deveni foarte complicate. Virtualizarea va juca un rol important atât în vederea portabilității aplicațiilor, cât și în vederea evidențierii concurențelor, mai facilă la nivelul mașinii virtuale, care poate beneficia de metainformații derivate din codul HLL (High Level Language). În schimb, în multe aplicații din calculul dedicat (aplicații grafice, aplicații numerice etc.) este foarte probabil ca sistemele multicore omogene să aibă succes.

4. Exploatarea sinergică a tipurilor de paralelism

Cercetările viitoare trebuie să vizeze și exploatarea sinergică a diverselor tipuri de paralelism (de tip pipeline prin suprapunerea execuției fazelor instrucțiunilor, ILP, TLP – Thread Level Parallelism, ori chiar la nivelul unor task-uri independente) Extensiile de paralelizare ale limbajelor de programare (ex. OpenCL, Grandcentral) vor ajuta aplicațiile să beneficieze de toate resursele disponibile (CPU, multi-cores, GPU).

Nici paralelismul la nivelul datelor (bit, cuvânt – arhitecturi vectoriale) nu trebuie neglijat. Spre exemplu, modelul OpenMP permite exploatarea paralelismelor la nivel de task-uri, bucle și date, deopotrivă. Aplicațiile de tip web, mobile (cloud computing) sau bazele de date sunt caracterizate de paralelism la nivel de fire sau la nivelul acceselor la memorie și mai puțin la nivel ILP. În asemenea cazuri, fiecare client poate fi servit de către un nucleu separat. Aceste aplicații necesită multe procesoare simple, cu memorii performante de mare capacitate.

5. Ierarhia de memorie

Se arată în literatură că ierarhia de cache-uri este departe de a fi optimal proiectată. Circa 50% din blocurile din cache sunt „moarte” (analog, cca. 50 % din datele dintr-o pagină sunt nefolosite). Sistemul de cache-uri nu mai trebuie văzut ca un modul fix, pe care soft-ul trebuie să se plieze. Acest sistem trebuie să devină mai maleabil, adaptat la cerințele rulării dinamice a aplicațiilor. Sunt necesare cercetări în vederea optimizării ierarhiei de memorii cache în sistemele multiprocesor, prin exploatarea agresivă a „localităților” (vecinătăților) spațiale și temporale. Astfel, spre exemplu, în [18] autorii se focalizează pe mecanisme hardware, care adaptează dimensiunea logică a blocului din L2 cache la caracteristicile aplicației. Accesarea L2 cache se face în paralel cu accesarea unor așa numite cache-uri de monitorizare (Observation Cache - OC), fiecare având lungimi diferite ale blocului. Cache-ul OC care contorizează pe un anumit interval de timp un număr minim de miss-uri, determină blocul optimal din punct de vedere al dimensiunii, necesar a fi adoptat în L2 cache. Un bloc de o asemenea dimensiune exploatează cel mai bine vecinătatea spațială a datelor și a instrucțiunilor. Un controller special de întreruperi monitorizează periodic starea cache-urilor OC și, în consecință, modifică în mod corespunzător lungimea blocului logic din L2 cache. Mecanismul este unul simplu, eficient și robust, care nu implică intervenția compilatorului ori a aplicației software. Sunt necesare cercetări care să dezvolte noi scheme de protocoale de coerență a cache-urilor în sistemele MIMD, scalabile, flexibile, reconfigurabile chiar, care să suporte sute și chiar mii de nuclee de procesare integrate pe un singur chip de uz general. Există studii empirice care arată că, la o tehnologie de integrare de 30 nm, se pot integra 1000-1500 de nuclee simple pe o singură pastilă de siliciu (actualmente Intel utilizează tehnologii la 45 nm). Tehnicile de coerență actuale de tip snooping (MSI, MESI, MOESI etc.), precum și cele de tip directory-based sunt depășite, fiind nescalabile la nivelul sutelor și miilor de nuclee integrabile pe un chip. Totuși, cercetări incrementale utile, bazate în principal pe predicție și speculație, ar putea perfecționa chiar și aceste protocoale. Tehnicile de coerență vor trebui să aibă în vedere noile caracteristici specifice sistemelor multicore (latențele miss-urilor cache to cache, lărgimea de bandă a bus-urilor de interconectare etc., care sunt mult diferite decât cele aferente sistemelor multiprocesor clasice.) În consecință, viitorul aparține sistemelor multicore cu memorie partajată distribuită (Distributed Shared Memory sau Non Uniform Memory Architectures) cu protocoale de coerență hardware-software sau de tip Message Passing (NoC – Networks on a Chip) și în tehnologii de integrare de tip 3 D [14]. În optimizarea mecanismelor de coerență trebuie ținut cont în mod deosebit de minimizarea puterii consumate.

6. Coerența și consistența

Un program paralel trebuie să specifice, printre altele, ordinea operațiilor de scriere-citire. Cel mai simplu, ar fi să se păstreze ordinea impusă de dependențele de date din program. Când ordinea implicită a acestor operații nu este determinată în mod unic, sunt necesare operații explicite de sincronizare între fire (creare/unificare fire, excludere mutuală, alocare memorie partajată, bariere etc.). Coerența cache-urilor asigură o viziune consistentă a memoriei pentru diversele procesoare din sistem. Nu se răspunde însă la întrebarea „cât de consistentă?”, adică în ce moment trebuie să vadă un procesor că o anumită variabilă a fost modificată de un altul? Cea mai simplă soluție constă în forțarea fiecărui procesor care scrie o variabilă partajată, de a-și întârzia această scriere până în momentul în care toate invalidările cauzate de către procesul de scriere, se vor fi terminat. Această strategie simplă se numește consistență secvențială. Ea impune ca

orice simulare a procesării unui anumit program, prin respectarea secvențialității interne din cadrul unui fir (data-flow order) și, respectiv, prin ordinea aleatoare de execuție a firelor între ele, atunci când nu este specificată una anumită, să conducă la aceleași rezultate finale. Altfel spus, un program scris corect va duce la aceleași rezultate, indiferent de întreținerea firelor. Deși consistența secvențială prezintă o paradigmă simplă, totuși ea reduce performanța, în special pentru sistemele cu un număr mare de procesoare sau cu rețele de interconectare de latențe ridicate. Așadar, este necesară dezvoltarea unor noi modele, mai relaxate decât cel al consistenței secvențiale (release consistency) de asigurare a consistenței memoriei partajate. Unele cercetări alternative se focalizează pe anumite mecanisme de prefetch a datelor, implementate în cadrul mecanismului simplu al consistenței secvențiale, încercând astfel să reducă degradarea de performanță pe care acest mecanism o implică prin secvențialitatea acceselor la memoria de date.

7. Rețele de interconectare

Sunt necesare rețele de interconectare de lărgime de bandă ridicată, în cadrul unor sisteme cu memorie partajată on-chip, scalabilă. Actualmente, rețelele de interconectare (on chip) au lărgimi de bandă relativ modeste, fiind limitate de capacitățile parazite, datorate modulelor interconectate, întârzierilor de arbitraj etc. În plus, ele sunt nescalabile la nivelul sutelor ori miilor de nuclee integrate. O excepție pozitivă o constituie HyperTransport, o rețea de interconectare de tip point to point (pachete), bidirecțională, serial/paralelă, scalabilă, de latență mică și de lărgime de bandă ridicată, v. <http://www.hypertransport.org>. Ea permite interconectarea procesoarelor, a acestora cu interfețele de I/O și cu diferitele acceleratoare. AMD utilizează această rețea flexibilă în multe dintre multicore-urile sale (Athlon 64, Athlon 64 X2, Athlon 64 FX, Opteron, Sempron și Phenom). HyperTransport oferă legături distincte între procesor – memorie și respectiv între procesor - sub-sistemul de I/O. Mai mult, ea oferă legături fizice distincte pentru citiri și scrieri din/în spațiul de I/O, oferind, deci, paralelizări agresive ale acestora. La ora actuală, rețeaua oferă rate de transfer de până la 10400 MB/s (versiunea a 3-a).

În domeniul rețelelor de interconectare implementate on-chip, cercetările vor avea în vedere, în special, arhitecturile switch-urilor, topologiile și algoritmi de rutare. Tehnologia VLSI utilizată influențează în mod direct soluțiile arhitecturale. Probabil că cercetările se vor inspira din rețelele de interconectare ale supercomputerelor actuale (spre exemplu, supersistemul IBM BlueGene care conține 65536 de noduri, clusterizate în cate 80 CPU 2-SMT PIM - Processing in Memory, conectate Crossbar/cluster. Aceste clustere sunt interconectate prin intermediul unei rețele tip 3 D Torus Network care face ca fiecare nod să poată comunica direct cu alți 6 vecini situați pe axele ortogonale XYZ. Aici și subsistemul ierarhic de cache-uri va juca un rol extrem de important, fiind necesară îmbunătățirea sa prin noi idei. De remarcat că eficientizarea acestui subsistem va conduce la scăderea presiunii asupra rețelelor de interconectare și a memoriei principale partajate. Detecția și abortarea execuției instrucțiunilor Store, care doresc să scrie o valoare deja existentă în memoria partajată, numite Silent Stores, ajută de asemenea în mod semnificativ la reducerea presiunii asupra rețelei de interconectare [16]. Cercetări novatoare în arhitectura memoriilor DRAM sunt foarte necesare, având în vedere că un chip DRAM de 512 Mbit, spre exemplu, conține sute de blocuri fizice independente, oferind, deci, un potențial uriaș în creșterea lărgimii de bandă, prin accesări întreținute. Integrarea on-chip a memoriilor DRAM nu mai presupune multiplexarea adreselor (necesară în memoriile off-chip datorită costurilor mari ale terminalelor). Acest fapt conduce la scăderea semnificativă a timpului de acces. De altfel, sistemele masiv paralele se bazează pe conceptul de PIM. În acest caz, memoria DRAM este integrată în cadrul procesorului, cu mari beneficii asupra latenței și lărgimii de bandă procesor-memorie. O rețea de procesoare PIM se numește arhitectură celulară. În arhitecturile celulare, se pot conecta milioane de procesoare, fiecare procesor fiind conectat doar la câțiva vecini din cadrul topologiei de interconectare. Sistemele multicore au accentuat și mai mult gap-ul între CPU și sistemul secundar de memorie (discuri), conducând la scăderea lărgimii de bandă per core. Este deci imperios necesar un sistem de I/O mult mai rapid, bazat pe abilitatea de a mapa în mod eficient un număr mare de operații concurente de I/O, pe sutele de unități de stocare. În acest scop, trebuie aduse îmbunătățiri aplicațiilor, mașini virtuale, sistemului de operare și controlerului dedicat al unităților de disc.

8. Modele de programare paralelă

Conform [10], în următoarea decadă, în cadrul aplicațiilor, vor fi deosebit de pregnante următoarele tendințe: accesul ubicuu, servicii personalizate și delocalizate, sisteme masive de procesare a datelor, realitate virtuală de înaltă calitate, senzori inteligenți. Aceste tendințe se vor manifesta în aplicații concrete, precum cele legate de roboți domestici, vehicule auto-pilotate, teleprezență, jocuri, implanturi și extensii ale corpului uman (human++) etc. Pentru programarea acestor aplicații pe sistemele multicore și

manycore, sunt necesare modele eficiente, performante, dar și simple (!) de programare paralelă. Cercetările în acest domeniu trebuie să investigheze metodele de evidențiere a concurențelor la nivelul limbajelor de programare. În general, este de dorit ca aceste modele să fie independente de numărul de procesoare din sistem. Paralelismul reprezintă rezultatul exploatarei concurenței pe o platformă paralelă. Rolul modelului de programare este acela de a exprima concurența, într-un mod independent de platformă. Este sarcina compilatorului și a mașinii hardware să decidă cum să exploateze concurența, prin procesări paralele. Gradul de abstractizare al modelului de programare trebuie ales prin prisma compromisului optimal între productivitatea și eficiența programării paralele. Aceste modele trebuie să pună în evidență paralelismele inter-thread-uri, alocările de memorie, accesul la zonele de date partajate și modurile de sincronizare. Ingineria programării se focalizează în continuare pe corectitudinea și pe reutilizarea codului, pe productivitatea dezvoltării, dar nu și pe exploatarea paralelismelor. Se estimează că, în viitorul apropiat, circa 10% dintre programatori vor dezvolta programe paralele în mod explicit. Actualmente, aceste modele de programare paralelă sunt nesatisfăcătoare, conducând la o programare, testare și depanare extrem de dificile. Spre exemplu, standardul Posix este considerat ca fiind puțin flexibil și de nivel jos. Standardul OpenMP nu exploatează „localitatea” datelor. De asemenea, modelele de programare actuale nu iau în considerare eterogenitatea arhitecturii. În plus, scalabilitatea acestor modele este una scăzută. Productivitatea actualelor paradigme de programare paralelă (shared memory - memorie partajată respectiv message passing - memorie distribuită logic) este una scăzută. Astfel, de exemplu, metodele actuale de sincronizare, bazate pe secțiuni critice atomice (excluziune mutuală prin lock/unlock), nu mai sunt fezabile, fiind nevoie de metode noi, mai productive.

În acest sens, conceptul de memorie tranzacțională (Transactional Memory - TM) pare a fi unul promițător, deși cercetările sunt încă într-un stadiu incipient. Tranzacția constituie o secvență de cod care se execută atomic, în mod speculativ, prin mai multe citiri și/sau scrieri la nivelul unei memorii partajate. Rularea programului nu ține cont de secțiunile critice. Dacă apar conflicte la nivelul variabilelor partajate accesate de fire multiple, rezidente pe diferite procesoare, aceste conflicte se vor detecta și firul violat își va relua execuția tranzacției în mod corespunzător (roll-backs). Așadar, gestiunea coerenței nu se mai face la nivelul fiecărei scrieri aferente unei variabile partajate, ci la nivelul unor pachete atomice, fiecare pachet conținând mai multe astfel de scrieri. Tranzacția este atomică (se execută dpdv logic în totalitate sau deloc), consistentă (dpdv al variabilelor partajate inter-tranzacții) și durabilă (odată începută, nu mai poate fi abortată). TM simplifică tehnicile de excluziune mutuală din programarea paralelă. Avantajul principal al conceptului de TM nu o constituie atât performanța rulării, cât corectitudinea acesteia, chiar și în condițiile în care programatorul (compilatorul) efectuează în mod eronat paralelizarea aplicației. Productivitatea și facilitarea programării constituie alte obiective importante, asociate acestui concept. Probabil că cercetările în domeniul TM trebuie să abordeze scheme hibride, de tip hardware - software. Aceste cercetări trebuie dezvoltate în paralel cu extensia și optimizarea setului de instrucțiuni mașină (ISA - Instruction Set Architecture) și a interfeței hardware-software, în vederea facilitării programării paralele.

În lucrarea [8], cercetători de la Stanford propun un nou model de memorie partajată, numit Transactional memory Coherence and Consistency (TCC), practic o memorie tranzacțională implementată în hardware. Aici, tranzacțiile atomice sunt întotdeauna unitățile de bază ale procesării paralele. TCC trebuie să grupeze, în hardware, toate scrierile dintr-o tranzacție, într-un singur pachet. Acest pachet se trimite în mod atomic la memoria partajată, iar scrierile variabilelor partajate se efectuează la finele execuției tranzacției (commit). Se controlează prin hardware roll-back-urile tranzacțiilor procesate în mod speculativ. Aceste tranzacții speculative necesită roll-back atunci când mai multe procesoare încearcă să citească și să scrie, în mod simultan, aceeași dată. Protocoalele de coerență de tip snoopy se implementează la nivelul acestor tranzacții atomice, și nu la nivelul scrierilor individuale. Ele permit detecția faptului că tranzacția curentă a utilizat date care au fost deja modificate de o altă tranzacție (dependence violation) și deci, este necesar roll-back-ul tranzacției în curs. În consecință, consistența secvențială se implementează la nivelul tranzacțiilor, care se vor termina în ordinea secvențială a programului original, și nu la nivelul scrierilor individuale. Întreșterea între scrierile diferitelor procese este permisă numai la nivelul tranzacțiilor. Acest model TCC impune la nivelul programatorului să insereze explicit tranzacțiile în codul sursă, ca pe niște regiuni paralelizabile. Aceste tranzacții pot fi rafinate iterativ și adaptiv, în urma diferitelor rulări ale programului. Evident că o anumită tranzacție nu poate separa un Load de un Store succesiv, care accesează aceeași variabilă partajată ca și Load-ul.

O altă provocare, extrem de importantă, o constituie dezvoltarea unui model de programare care să permită utilizarea transparentă și simultană atât a modelului cu memorie partajată, cât și a celui cu memorie distribuită. Ceva încercări în acest sens se regăsesc în limbajele Co-Array Fortran, UPC, X10, Fortress, Chapel etc. [10]. De asemenea, sunt necesare mecanisme novatoare care să permită compilatorului și sistemului run-time să optimizeze structurile de date partajate, adaptându-le la condițiile execuției. Ideea de esență este că o structură de date partajată să fie distribuită automat în sistem.

Cercetările în domeniul arhitecturilor reconfigurabile, în special în lumea embedded, sunt și ele extrem de necesare întrucât aceste arhitecturi se pot adapta în mod static sau/și dinamic mai bine la cerințele aplicației specifice. Avantajul limbajului Java în arhitecturile multiprocesor constă în faptul că implementează în mod nativ conceptul de fir de execuție. Din acest motiv, cercetările în domeniul sistemelor multicore, care utilizează procesoare capabile să execute direct în hardware bytecode-urile Java, sunt deosebit de utile, în special în lumea sistemelor dedicate.

Este necesar un suport hardware pentru programarea paralelă, inclusiv pentru depanarea programelor paralele prin monitorizarea execuțiilor. Depanarea unor sisteme multi-core cu sute de thread-uri procesate în limbaje native diferite, de către procesoare neomogene, este o problemă deschisă de mare actualitate și interes. O singură sesiune a depanării trebuie să vizualizeze toate instrucțiunile mașină, informații legate de variabile și funcții, punerea în evidență a erorilor de comunicare între module, dar și a erorilor locale etc. Evitarea violărilor de timing în cazul sistemelor în timp real având constrângeri tari (Worst Case Execution Time) și a erorilor de execuție având cauze incerte (Heisenbugs), constituie provocări majore. Fără ajutorul hard-ului, care să ofere o viziune globală a stării mașinii multicore, asemenea cerințe par imposibil de îndeplinit. Deci, proiectarea hardware trebuie să aibă în vedere și observabilitatea rulărilor multiple, fără a genera însă prea mari cantități de date nerelevante.

9. Paralelizarea aplicațiilor

Paralelizarea automată constituie un obiectiv maximal, vizat de vreo 40 de ani de cercetări asidue. În acest sens, s-au obținut realizări notabile, îndeosebi în paralelizarea automată a programelor științifice, scrise în limbaje orientate pe vectori (Fortran, Matlab etc.) și pentru arhitecturi omogene cu memorie partajată, în special din lumea embedded. Aceste realizări trebuie extinse la tipuri cât mai diverse de aplicații, la limbaje bazate pe pointeri și la sisteme multiprocesor eterogene, cu diferite modele de memorie. În acest scop, sunt necesare, în special, tehnici statice noi de analiză a programelor scrise în limbaje bazate pe pointeri. O altă direcție insistă pe limbajele orientate pe domenii, unde punerea în evidență a concurențelor este mai facilă decât în cele de uz general. Paralelizările speculative, incluzând aici tehnici de multithreading speculativ, vor juca un rol tot mai important. Planificarea dinamică a thread-urilor în vederea optimizării comunicațiilor și sincronizărilor este deosebit de importantă. Astfel, de exemplu, în [7] autorii dezvoltă o așa numită Decoupled Threaded Architecture (DTA) în vederea exploatarei paralelismelor fine și medii la nivel TLP în cadrul unor sisteme many-core eterogene. Comunicațiile și sincronizările între fire sunt interesante, făcându-se pe un model de tip flux de date (data-flow) și prin mecanisme de sincronizare de tip non-blocking. La crearea unui fir, acestuia i se asignează un numărator de sincronizare (SC), reprezentând numărul datelor de intrare pe care firul trebuie să-l primească de la alte fire. Acest numărator este decrementat de fiecare dată când firul primește o astfel de dată în memoria sa locală, destinată comunicațiilor interfir. Când SC-ul firului a ajuns la zero, se va starta execuția firului respectiv prin încărcarea datelor din memoria locală în registre. Din păcate, metodologia de simulare din această lucrare este una superficială. De asemenea, transmiterea de către compilator a unor informații de semantică a aplicației către sistemul multicore pe care aceasta va rula, ar putea conduce la optimizări semnificative în procesarea aplicației.

10. Simularea ca instrument de cercetare

Sunt necesare metode de simulare adecvate, stăpânirea complexității cercetării-dezvoltării sistemelor multicore, inclusiv prin simulare tranzacțională, Transaction-Level Modeling - TLM, precum permite mediul de dezvoltare pentru sisteme multicore UniSim, v. - www.unisim.org. Astfel, metoda TLM lucrează la un nivel de abstractizare superior celui utilizat la nivelul simulării ciclu cu ciclu. Aici, simulările se focalizează preponderent pe comunicațiile între modulele componente. Este importantă simularea întregului sistem de calcul, inclusiv a sistemului de operare, cu toate nivelurile ierarhice funcționale pe care acesta le deține (full-system simulation – FSS, exemple: M5, Simics, GEMS etc.). Simulatoarele monolitice, gen SimpleScalar, M-Sim, etc., vor fi înlocuite cu simulatoare modulare, care să exploateze actualele tehnici ale ingineriei programării obiectuale, în vederea facilitării scrierii programelor și a reutilizării codului (exemple SystemC, Liberty, MicroLib, GEMS, SimFlex, M5, UniSim sau ASIM). O idee importantă în acest sens constă în maparea facilă, intuitivă, a blocurilor hardware pe modulele (funcțiile, clasele) software ale simulatorului. Prin instanțierea acestor blocuri și specificând conexiunile lor, se poate realiza o dezvoltare ierarhizată a sistemului multicore. Astfel, dezvoltarea proiectului va fi una mult mai facilă decât într-un mediu monolitic de simulare. Calitatea unui simulator de sisteme multicore este dată de caracteristici precum: posibilități de dezvoltare (modularitate), benchmark-uri & compilatoare, posibilități full-system

simulation, tipuri de procesoare oferite, caracteristici multiprocesor oferite (UMA/NUMA, protocoale de coerență, rețele interconectare, modele de programare pe care le pune la dispoziție, modele de consistență etc.), facilitate de calcul putere/temperatură, viteză de simulare, acuratețe de simulare, gradul de parametrizare (flexibilitatea arhitecturală). Un simulator pentru sistemele multicore, care simulează întregul sistem de calcul este Simics (Virtutech - v. <http://www.virtutech.com>). Acesta oferă un set larg de procesoare (Alpha, ARM, MIPS, PowerPC, SPARC, x86-64) și interfețe. Simics poate încărca și rula sisteme de operare precum Linux, Solaris, Windows XP. Poate virtualiza mașina țintă în sisteme multiprocesor, clustere de procesoare sau rețele. Un simulator pentru sistemele multicore, dezvoltat în mediul academic american, este RSim - v. <http://rsim.cs.uiuc.edu>. Implementează o memorie partajată distribuită fizic (fiecare procesor având o memorie locală) și protocoale de coerență de tip directory (MSI, MESI). Oferă procesoare superscalare puternice cu execuții out-of-order și arhitecturi complexe de memorie (CC-NUMA), inclusiv cu posibilități de adresare întretesută. Comunicarea între procesoare se face prin intermediul unei rețele bidimensionale, de tip plasă. De asemenea, oferă suport pentru implementarea consistenței secvențiale sau chiar a unor modele mai relaxate. Un alt simulator interesant, la nivel de ciclu mașină (emulator de arhitectură MIPS), destinat sistemelor multicore cu un număr configurabil de nuclee, este SESC, <http://sesc.sourceforge.net>. Acesta deține un modul de calcul al timing-ului de mare acuratețe. M5 este un FSS-simulator dezvoltat în tehnologia programării pe obiecte (C++), permițând deci instanțierea facilă a modulelor sistemului multiprocesor. Conține modele pentru procesoarele Alpha, MIPS, ARM și SPARC. Implementează ierarhii complexe de cache cu protocoale de coerență de tip snoopy. Încarcă sistemele de operare Linux și Unix (Solaris). Permite 3 moduri de lucru în vederea compromisului între viteza de simulare și acuratețea simulării. În domeniul sistemelor multicore dedicate aplicațiilor multimedia, simulatorul Sesame, dezvoltat la Universitatea din Amsterdam, este unul cunoscut și apreciat.

O altă problemă majoră constă în viteza simulărilor. La ora actuală, simularea cu acuratețe totală a unui sistem monoprocessor real, la nivel RTL – Register Transfer Logic, necesită în jur de o zi de simulare, pe sisteme performante. Pentru sisteme multicore, acest timp de simulare va crește cel puțin într-o manieră liniară. Având în vedere complexitatea enormă a simulărilor de tip cycle by cycle, care le face nefezabile pentru optimizarea unor sisteme cu sute de nuclee integrate, există un mare interes inclusiv pentru dezvoltarea unor metode de simulare la nivel de tranzacții sau chiar a unor metode analitice de optimizare. Se au în vedere inclusiv metode de simulare statistică, bazate pe eșantioane reprezentative ale procesării (sampling simulation). Aici, ideea de esență constă în selectarea unor subseturi ale benchmark-urilor având un comportament suficient de similar cu cel al întregului set de benchmark-uri. O bună sinteză a cercetărilor din acest domeniu se găsește în [19]. Se pune problema extragerii caracteristicilor benchmark-urilor în urma rulării și construcția unor benchmark-uri sintetice având aceleași caracteristici, reprezentative și mai scurte (statistical simulation). O altă soluție în vederea reducerii timpului de simulare constă în paralelizarea simulatorului și rularea lui pe sisteme multicore performante precum IBM Cell BE, de exemplu. O altă soluție în vederea reducerii timpului de simulare o constituie clasificarea / clusterarea prin metode statistice a benchmark-urilor. În fine, o altă provocare importantă o constituie îmbunătățirea acurateței simulărilor prin metodologii specifice. Acuratețea relativă a simulărilor poate fi mai importantă decât acuratețea absolută a acestora, în fazele incipiente ale proiectului.

11. Benchmarking

Ca metodologie de cercetare – dezvoltare, se impune tot mai mult proiectarea hardware-software integrată (hardware-software co-design). Tot aici, se pune problema dezvoltării unor metode de benchmarking, adecvate evaluării sistemelor de tip multicore. Problema este una extrem de delicată, întrucât companiile își protejează atent aplicațiile comerciale. Astfel, în locul uzitatelor benchmark-uri (SPEC – pentru procesoare de uz general, EEMBC și MiBench – pentru sisteme dedicate, Mediabench, ALPBench – pentru sisteme multimedia, TPC – pentru baze de date, Livermoore, Parsec, SPLASH 2 – pentru sisteme cu paralelism masiv etc.) s-au propus așa numitele dwarfs, constând în metode algoritmice, tipice calculului paralel, care să conțină atât pattern-uri computaționale, cât și, mai ales, pattern-uri de comunicație între nucleele componente [1]. Și totuși, problema rămâne una deschisă, cu multe semne de întrebare (spre exemplu, benchmarking-ul trebuie să țină cont și de tendința accentuată a virtualizării, ca și metodă esențială în asigurarea compatibilității și portabilității).

12. Automatic Design Space Exploration

Se pune problema dezvoltării unor metode de căutare euristică optimizată în spațiul enorm al parametrilor aplicațiilor și arhitecturii cercetate - Automatic Design Space Exploration (ADSE). Spre

exemplu, proiectarea unui 4-core în care fiecare nucleu poate fi ales dintr-o bibliotecă conținând 480 de modele, impune evaluarea a peste 2.2 miliarde de posibilități! Dacă evaluarea fiecărei posibilități ar necesita o zi, evaluarea tuturor posibilităților ar necesita aproape 1 milion de ani. Relativ la aplicații, se au în vedere transformările algoritmice și de limbaj, ce pot crește gradul de paralelism TLP. Scopul este evident, anume determinarea acelor parametri care conduc la optimizarea raportului performanță/cost (IPC, energie, arie de integrare etc.) Optimizările au, deci, obiective multiple. Evident că nu se pune problema căutării complete în spațiul enorm al tuturor parametrilor arhitecturii sau/și aplicațiilor investigate. Tehnici din domeniile învățării automate (machine learning), a cercetărilor operaționale ori data mining ar putea fi utile, în vederea reducerii spațiului de căutare și deci, a reducerii numărului de simulări. Aceste metode de optimizare automată a parametrilor arhitecturii se vor integra sub forma unor API-uri în cadrul simulatoarelor dezvoltate. Tot aici, sunt necesare tehnici euristice, inspirate din domeniul machine learning, în vederea optimizării incrementale a compilării. Ideea este ca programul să se adapteze la schimbările din hardware (spre exemplu, creșterea numărului de procesoare), dar și la evenimentele dinamice, care apar pe parcursul procesării (spre exemplu miss-uri în cache). La ora actuală, nu există un instrument ADSE matur, care să fie larg folosit în optimizarea sistemelor de calcul. Obiectivele constau în creșterea performanței, dar și în reducerea consumului de putere (atât dinamică, dar și statică). Evident că, în acest sens, sunt necesare informații de feedback, captate în urma rulărilor aplicației. Este necesară, deci, o abordare mai strânsă între cercetările în domeniul microarhitecturilor și cel al compilatoarelor (compiler-architecture co-design). Compilatorul trebuie să poată manipula infrastructura microarhitecturii, în timp ce aceasta trebuie să beneficieze de informațiile transmise ei prin intermediul ISA, de către compilator. Din păcate, este foarte dificilă trecerea de la actualele compilatoare, orientate pe exploatarea LLP, la compilatoare noi, orientate pe exploatarea TLP și a paralelismelor la nivelul task-urilor. Totodată, această abordare holistică este una mare consumatoare de timp. În scopul reducerii timpului de proiectare integrată a dualității arhitectură-compilator, se dezvoltă modele bazate pe învățare automată, care pot predicționa performanța unui compilator optimizat pentru o anumită arhitectură, fără ca să îl construiască efectiv. Modelele folosesc ca intrare o parte infimă din spațiul parametrilor compilator-arhitectură.

Așadar, avem nevoie de tehnici inteligente și eficiente de căutare în spațiul enorm de parametri ai sistemului. Adaptarea tehnicilor de căutare - optimizare din domeniul învățării automate a fost investigată de către mulți autori. În [15], spre exemplu, autorii dezvoltă un instrument complex de accelerare a DSE pentru arhitecturi multicore, numit Magellan. Acesta determină parametrii cvasioptimali, cei care maximizează rata de procesare pentru un buget prestabilit al ariei de integrare și puterii disipate. Magellan folosește algoritmi euristici de căutare de tip hill climbing, genetici, stigmergici etc. Clasificarea soluțiilor și a benchmark-urilor funcție de caracteristicile acestora, poate accelera căutarea.

Algoritmul Steepest Ascent Hill Climbing (SAHC) implică, în acest caz, căutarea în vecinătatea celui mai bun k-procesor curent. Un procesor vecin este un procesor care diferă de cel curent, prin valoarea unui singur parametru. Următorul procesor optimal este ales ca fiind cel mai bun vecin, dacă este superior procesorului curent. Algoritmul se oprește în proximal punct de extrem. Avantajul principal al acestui algoritm simplu constă în rapiditatea convergenței. Dezavantajele constau în complexitatea exponențială cu numărul de core-uri (k) și în posibilitatea eșuării în extreme locale. Evident, rafinamente ale acestui algoritm au fost propuse și investigate (spre exemplu, algoritmi tip annealing search → SAHC + alegere random).

Algoritmul genetic (AG) utilizează, printre alți operatori, și operatorul de reproducere, care transferă în populația următoare vecinul cel mai bun. Se asigură astfel că algoritmul este cel puțin la fel de bun ca SAHC. Operatorul crossover generează noi procesoare, prin combinarea a două procesoare din populație. Se implementează și mutația, care modifică aleator numărul de nuclee. În implementarea citată, se păstrează în populația următoare doar cele mai bune 4 soluții. Astfel, algoritmul scalează foarte bine cu creșterea numărului de nuclee (k), evaluând doar 4 k-procesoare într-o iterație. Nu există la acest moment o soluție optimă de oprire a algoritmului.

Algoritmii stigmergici se bazează pe comportamentul furnicilor din lumea reală (ant colony optimizations - ACO). În căutarea hranei, furnicile marchează drumurile cu feromoni, astfel încât traseele să poată fi urmate și de alte furnici. Feromonii se evaporă în timp, eficientizând astfel căutarea (spre exemplu, se evită explorarea unor căi devenite, între timp, fără de succes.) Autorii forțază căutarea pe câte o cale diferită în fiecare iterație, prin pornirea căutării de la o soluție situată în vecinătatea celei precedente. Astfel, se evită eșuarea în extreme locale, explorându-se însă mai mult spațiul soluțiilor.

Concluzia acestor cercetări a arătat că aceste tehnici euristice de explorare sunt cel puțin de 3800 de ori mai rapide decât căutarea exhaustivă, generând soluții cu maximum 1% mai puțin performante decât aceasta, fapte remarcabile. Scalabilitatea este de asemenea asigurată.

Dacă în științele tari, mature din punct de vedere teoretic, precum fizica sau biologia, posibilitatea reproducerii rezultatelor experimentale constituie o condiție sine qua non, în arhitectura calculatoarelor acest lucru este de multe ori imposibil, datorită faptului că nu există o metodologie standardizată a cercetării. În cadrul cercetărilor dintr-o companie, simularea modulară ar putea constitui o soluție, dacă toți cercetătorii ar folosi același simulator. Din păcate, la nivelul cercetărilor academice ori la nivel global intercompanii, această abordare este, practic, imposibilă. În [4], se prezintă o metodologie automată de optimizare euristică a parametrilor unei microarhitecturi (ADSE), implementată sub forma unui site web numit ArchExplorer.org. Aceasta este oarecum independentă de simulatorul care implementează blocul hardware ce se dorește a fi optimizat, chiar dacă, evident, îl utilizează pentru căutarea în spațiul enorm al parametrilor aferenți arhitecturii, dar și compilatorului. Desigur că integrarea simulatorului corespunzător unui anumit bloc hardware (de tip cache, TLB, predictoare de branch-uri, module DRAM, unități funcționale, rețele de interconectare etc.) în ArchExplorer.org este mai facilă dacă acesta are deja o construcție modulară, cu interfețe bine definite între module (analoage API-urilor), compatibile cu cele definite în ArchExplorer.org. Un astfel de simulator este UniSim. Metodologia de explorare a spațiului parametrilor arhitecturali utilizată în [4] se bazează pe algoritmi genetici. Fiecare modul are asociată o genă, iar aceasta la rândul ei, deține mai multe subgene, codificând parametrii modulului. Din păcate, modelarea operatorilor genetici utilizați nu este clar explicată în lucrare. În cadrul ADSE, relația arhitectura-compilator, deși subtilă, este una puternică și adesea subestimată de către cercetători. Modificările arhitecturale pot produce modificări importante în strategiile de optimizare din compilator, dar și reciproc. Dacă pentru un compilator neoptimizat performanțele procesorului sunt mai bune pentru setul de parametri P1 decât pentru P2, atunci este posibil ca, folosind un compilator optimizat pentru P1 respectiv P2, performanțele în punctul P2 să fie mai bune decât în P1. Interesant este faptul că mediul ArchExplorer.org oferă o explorare continuă a spațiului stărilor pe un web-server special dedicat.

O altă platformă web DSE este implementată prin proiectul colaborativ intitulat cTuning.org - http://ctuning.org/wiki/index.php/Main_Page, care oferă în mod gratuit utilizatorilor o tehnologie inteligentă în vederea optimizării ansamblului arhitectură – compilator - aplicație, pe baza unor metode din teoria învățării statistice și automate. Se poate accesa o bază de date (Collective Optimization Database) care conține detalii asupra optimizărilor aplicate unor sisteme de calcul complexe. Accesarea acestei baze de date oferă utilizatorilor și posibilitatea schimbului reciproc de experiențe, vizând optimizări interesante, pentru diferite aplicații și, respectiv, platforme utilizate pentru rularea acestora. În principiu, oricine poate să-și optimizeze în mod automat, cu instrumentele puse la dispoziție, o anumită aplicație sau benchmark. Trebuie specificate arhitectura hardware, sistemul de operare și compilatorul utilizate, urmând ca apoi serviciul web cTuning.org să încerce optimizarea parametrilor acestora, în vederea obținerii unui timp minimal de execuție, dar și a unei lungimi cât mai mici a codului sursă. În final, se generează automat parametrii optimali aferenți procesorului, compilatorului (parametrii de optimizare a codului sursă), sistemului de operare și aplicației. De asemenea, se determină și câștigurile (performanță, lungime de cod) obținute față de un compilator GCC standard. Proiectul FP7 intitulat MultiCube se ocupă și el de asemenea cercetări.

13. Arhitecturi multicore cu procesări anticipative

Puține sunt cercetările care analizează tehnicile Value Prediction (VP) și Dynamic Instruction Reuse (DIR) [16], în cadrul procesării firelor concurente pe procesoare SMT (Simultaneous Multithreading) sau multicore (personal nu cunosc nici o cercetare pe problematica grefării DIR în multicores). De aceea, grupul nostru de cercetare dorește să activeze în aceasta nișă, unde avem deja realizări apreciate (a se vedea <http://acaps.ulbsibiu.ro/research.php>). În asemenea arhitecturi, a prezice valoarea unei instrucțiuni și, apoi, a verifica predicția, după ce valoarea a fost produsă, nu este întotdeauna suficient. Mai mult, acest proces poate chiar implica erori de consistență a variabilelor predicționate. Astfel, spre exemplu, în unele cazuri, predicția poate să fie corectă, dar execuția incorectă datorită violării consistenței unor variabile. În [13], se exemplifică în mod convingător asemenea anomalii pe baza unui exemplu ce descrie un fir care inserează un element (nod) în capul (head) unei liste simplu înlănțuite. Problema este că, înainte de a insera noul nod în capul listei, firul îi modifică valoarea. Un alt fir, citește primul element al listei. Această citire bazată pe o VP corectă poate fi însă eronată. Între predicția valorilor și problema consistenței cache-urilor, în sistemele multiprocesor există legături subtile, neexplorate încă în mod aprofundat.

Instrucțiunile cu latență ridicată reprezintă o sursă de limitare a paralelismului la nivelul instrucțiunilor. În [6], am prezentat dezvoltarea un mecanism de anticipare selectivă a valorilor instrucțiunilor cu latență ridicată de execuție, care include o schemă de reutilizare pentru instrucțiunile Mul și Div, respectiv un predictor de valori pentru instrucțiunile Load critice, adică cele cu miss în

ierarhia de cache-uri. Rezultatele simulărilor efectuate, au arătat creșteri de performanță (IPC) de 3,5% pe benchmark-urile SPEC 2000 întregi, respectiv 23,6%, pe cele flotante și o scădere importantă a consumului relativ de energie (a EDP-ului), de 6,2% pe întregi respectiv 34,5% pe flotante. După ce am arătat utilitatea anticipării selective a instrucțiunilor cu latență ridicată într-o arhitectură superscalară, am analizat eficiența acestor metode și într-o arhitectură SMT, focalizându-ne pe aceleași instrucțiuni: Mul și Div, respectiv Load-uri critice. În acest context, firele de execuție conținând Load-uri critice ori alte instrucțiuni de latență ridicată pot bloca resursele partajate ale procesorului și, în consecință, pot bloca celelalte fire și deci, pot reduce performanța globală. Rezultatele au arătat îmbunătățiri ale IPC pe toate configurațiile SMT evaluate. Cu cât numărul de fire este mai mare, cu atât creșterea de performanță devine însă tot mai puțin semnificativă, datorită exploatarea tot mai eficiente a unităților de execuție partajate de către procesorul SMT. Plastic spus, cu motorul SMT mergând în plin, sporul de performanță aferent tehnicilor anticipative implementate adițional devine mai mic. Cele mai bune performanțe medii, de 2,29 IPC pe benchmark-urile de numere întregi, respectiv de 2,88 IPC pe cele flotante, s-au obținut cu șase fire de execuție.

În [2], autorii se ocupă tot de problema instrucțiunilor Load critice în arhitecturile SMT, însă abordarea lor este una diferită de a noastră. Ideea de bază este ca la detecția unui Load critic, în anumite condiții, să se declanșeze execuția speculativă a instrucțiunilor următoare în scopul exploatarea paralelismelor la nivelul memoriei prin procese de pre-fetch (MLP – Memory Level Parallelism). Gradul de MLP este dat de numărul de accese independente la memorie. Evident că un astfel de Load de mare latență va genera automat un checkpoint, salvând, deci, starea curentă a procesorului (registrii logici, istoria branch-urilor etc.). Execuția anticipată a instrucțiunilor care urmează unui Load critic, în cadrul unui anumit fir, se face numai dacă gradul de MLP estimat este unul suficient de ridicat. În plus, ele nu vor afecta starea arhitecturală (logică) a procesorului. Există implementat câte un predictor dinamic al gradului de MLP per thread. Acesta estimează gradul de MLP, după detectarea run-time a fiecărei instrucțiuni Load critice. În caz contrar (grad MLP redus), firul respectiv este rejețat din procesor, evitând astfel blocarea resurselor partajate ale acestuia. În cadrul procesării speculative, unele instrucțiuni independente de Load-ul critic pot cauza miss-uri în ierarhia de cache-uri. Latența acestora se suprapune peste cea a Load-ului critic aflat în curs de execuție. Astfel, se exploatează gradul de MLP existent în programe. Când un Load critic se încheie, procesorul iese din modul speculativ de execuție, golește pipe-urile, restaurează checkpoint-ul și reia procesarea normală, cu instrucțiunea următoare Load-ului. Această execuție normală se va face mai rapid având în vedere aducerile anticipate în cache-uri, efectuate pe parcursul execuțiilor speculative. Așadar, performanța per thread crește. Foarte important, această execuție speculativă a instrucțiunilor evită blocajul procesării în momentul în care un Load critic a atins vârful Reorder Buffer-ului (reducând deci Memory Wall-ul). Se arată că această arhitectură duce la beneficii considerabile, în special în cazul unor programe care lucrează intensiv cu memoria.

Asemenea idei merită aprofundate, îmbunătățite și investigate în continuare. Un studiu comparativ între aceste două abordări, care urmăresc reducerea influenței instrucțiunilor Load critice într-o arhitectură SMT, ar fi unul interesant și util. Desigur, comparația trebuie efectuată multicriterial (IPC, energie consumată, disipație termică, arie de integrare etc.) Mediul de simulare utilizat ar putea fi M-Sim întrucât permite și calculul puterii (prin simulatorul Watch pe care îl conține) și, totodată, l-am folosit deja în dezvoltarea arhitecturii SMT îmbunătățite prin anticiparea valorilor instrucțiunilor critice. Astfel, sperăm să obținem rezultate interesante într-un timp rezonabil.

Grefarea unor asemenea tehnici anticipative precum cele de reutilizare dinamică a instrucțiunilor sau de predicție dinamică a instrucțiunilor în cadrul arhitecturilor multicore și manycore reprezintă o problemă de mare interes în opinia noastră. Implementarea unor scheme de VP și DIR, care să nu violeze consistența variabilelor partajate în sisteme SMT și multicore este o problemă de mare interes. După cum se arată în literatura de specialitate, chiar benchmark-urile paralele (SPLASH-2, PARSEC etc.) nu conțin suficient paralelism pentru sistemele multiprocesor moderne [12]. În aceste condiții, creșterea performanței porțiunilor secvențiale de cod peste bariera impusă de celebra lege a lui Gene Amdahl (practic, bariera generată de dependențele de date între instrucțiuni), se poate realiza prin anticiparea rezultatelor instrucțiunilor (înainte ca acestea să fie produse, încă din faza de decodificare). Aceste tehnici anticipative vor reduce semnificativ accesul la rețeaua de interconectare în vederea comunicării prin variabile partajate.

La ora actuală, influența predicției dinamice a valorilor și, cu atât mai puțin, a reutilizării dinamice a instrucțiunilor, sunt departe de a fi înțelese în cadrul sistemelor SMT sau multiprocesor. Acest fapt este valabil chiar și la nivel teoretic-principial, după cum vom arăta în continuare printr-un exemplu simplu. Este binecunoscuta legea lui Amdahl's pentru un sistem MIMD cu N procesoare:

$$S(N) = \frac{1}{f + \frac{(1-f)}{N}} \quad (3)$$

În [12], autorii încearcă să generalizeze această formulă, la un sistem paralel cu predicția valorilor instrucțiunilor. Notând cu p acuratețea medie a predicției valorilor instrucțiunilor, autorii obțin relația de mai jos:

$$Svp(N) = \frac{1}{f(1-p) + \frac{fp + (1-f)}{N}} \quad (4)$$

Această relație, în opinia noastră, nu este corectă. Pentru $N=1$ (monoprocesor) $\rightarrow Svp(1)=1$, ceea ce ar însemna că tehnica de predicție a valorilor în sistemele monoprocesor nu ar aduce nici un avantaj. În mod evident, această afirmație este falsă. Așa cum s-a arătat în [5], [16], considerând un program pur secvențial ($f=1$), câștigul de performanță este în acest caz $\frac{1}{1-p}$.

O formulă corectă pentru $Svp(N)$ trebuie să țină cont că:

- $f(1-p)$ [%] din program este procesat în același timp în care ar face o mașină secvențială (SISD) echivalentă;
- fp [%] din program s-ar procesa teoretic instantaneu din cauza predicției perfecte a valorilor instrucțiunilor;
- $(1-f)$ [%] din program s-ar procesa teoretic de N ori mai rapid decât pe mașina secvențială.

În aceste condiții, formula corectă este:

$$Svp(N) = \frac{1}{f(1-p) + \frac{(1-f)}{N}} \quad (5)$$

Pentru $N=1$ rezultă:

$$Svp(1) = \frac{1}{1-fp} \geq 1. \quad (6)$$

Particularizând $f=1$ în formula precedentă, în perfect acord cu [5], rezultă:

$$S = \frac{1}{1-p} \quad (7)$$

Implementarea unor mecanisme DIR în arhitecturile multicore impune ca invalidările din cadrul buffer-ului de reutilizare (Reuse Buffer – RB) aferent procesorului considerat, să fie efectuate în mod global, deci inclusiv de către instrucțiunile Store executate de către un alt procesor care scrie la o adresă existentă în RB-ul procesorului considerat. Mecanismele globale de asigurare a coerenței cache-urilor, bazate pe invalidări la scrieri, ar putea ajuta la menținerea consistenței datelor din buffer-ul de reordonare. Invalidarea unei date din cache-ul de date ar trebui să determine automat invalidarea datei din RB, dacă aceasta este stocată și acolo.

Paradigma de procesare pe baza fluxurilor de date (Data-Flow, DF) găsește adepți și în domeniul multiprocesoarelor. Modelul DF, dezbătut puternic în anii '80, reprezintă un model formal distribuit, inerent paralel, funcțional și asincron. Execuția într-un nod startează de îndată ce toate datele de intrare sunt disponibile. Programul se reprezintă sub forma unui graf în care nodurile reprezintă operații, iar arcele reprezintă sursele (căile) de date. Modelul Data Driven Multithreading (DDM), relativ nou, exploatează modelul DF la nivelul firelor de execuție, în cadrul unor arhitecturi multicore. Modelul DDM se bazează pe conceptul paralelizării automate la nivelul hardware – software, incluzând tehnici de reprezentare poliedrală a programelor. Actualmente se cercetează ansambluri multicore tip DDM – compilator atât la nivel de virtualizare, cât și la nivelul unor implementări FPGA.

14. Putere consumată, disipație termică

Un obiectiv important constă și în grefarea unor module care să calculeze automat puterea disipată și disipația termică pe procesorul nostru SMT, îmbunătățit cu predicția valorilor instrucțiunilor Load critice

(cu miss în L2 cache) respectiv cu reutilizarea instrucțiunilor de latență ridicată (Mul/Div etc.). Astfel, pentru calculul puterii statice și dinamice, am putea utiliza în cadrul simulatorului SMT îmbunătățit de noi, simulatorul Wattch (<http://www.eecs.harvard.edu/~dbrooks/wattch-form.html>) care este deja integrat în mediul M-Sim utilizat. Acesta se bazează pe simulatorul SimpleScalar sim-outorder, ver. 3.0 și calculează puterea disipată pentru structuri de tip vector, structuri asociative de tip Reorder Buffer, Load Store Queue etc., logica combinațională (pentru partea de comenzi ale procesorului și unități funcționale) etc. Astfel, vom extinde calculul puterii disipate la întregul procesor (până în acest moment, am calculat puterea doar la nivelul unor structuri regulate, de tip cache). De asemenea, ne va interesa și calculul puterii statice, în special în cadrul unor memorii de capacități mari precum L2 caches. În particular, unii autori au integrat simulatorul Simics cu Wattch-ul și au implementat un simulator performant numit SimWattch (simulează inclusiv procesarea sistemului de operare + putere disipată).

Un modul pentru investigarea efectelor de disipație termică, aferente acestor tehnici, implementat ca un API al simulatorului M-Sim, ar fi de mare utilitate. Se știe că hotspot-urile conduc la erori de procesare, iar temperatura prea mare micșorează durata de viață a circuitului. Este interesant de evaluat efectul acesteia în contextul noilor structuri anticipative pe care noi le-am introdus în arhitectura SMT M-Sim. În acest sens, deja am început un studiu, în cooperare cu Politehnica din Milano, asupra problemelor, metodologiilor și instrumentelor utilizate în cercetarea efectelor disipației termice și de energie în cadrul microprocesoarelor. Cele două procese nu sunt corelate. Apoi, va trebui să interfațăm pe simulatoarele deja dezvoltate de noi, un software specializat în analiza disipației termice (de exemplu, simulatorul HotSpot - v. <http://lava.cs.virginia.edu/hotspot>, larg utilizat, și care nu necesită costuri de achiziție). În continuare, vom analiza aceste efecte în cadrul arhitecturilor deja dezvoltate de noi, pe diverse benchmark-uri (în special, în cadrul arhitecturii SMT îmbunătățite cu DIR și VP selective). Analiza comparativă a hărților termale, cu și fără îmbunătățirile arhitecturale propuse de noi, va fi deosebit de utilă. Pe baza puterilor dinamice, calculate pentru fiecare modul într-un interval (ex. 10 keicli), a frecvenței de tact, a tehnologiei de integrare și a geometriei de plasare a modulelor pe pastila de siliciu (floorplan, matrice de adiacență), HotSpot-ul va calcula densitatea de temperatură pentru fiecare modul. Unul dintre obiectivele acestei cercetări constă în proiectarea unui run-time thermal manager adaptiv, care să valideze - invalideze modulele de DIR și VP în funcție de comportamentul termic și energetic al modulelor microprocesorului. În acest scop, există deja elaborate mai multe strategii: oprirea tactului, adaptarea tensiunii de alimentare sau a frecvenței tactului funcție de temperatură, mutarea calculului în module de rezervă etc. Ideea de bază este aceea de a reduce densitatea de putere în punctele fierbinți. Aceste cercetări vor fi apoi extinse pe noile simulatoare multicore pe care le vom dezvolta.

15. Concluzii

Am realizat un studiu asupra impactului arhitecturilor multicore și manycore în cadrul ingineriei calculatoarelor. S-a pornit de la geneza acestor sisteme, determinată de limitele paradigmei monoprosesor. S-au identificat și s-au analizat în mod sistematic, pe baza unei literaturi de specialitate recente, următoarele provocări importante pentru cercetarea și dezvoltarea acestor sisteme:

- arhitecturi multicore omogene vs. eterogene;
- exploatarea sinergică a tipurilor de paralelism;
- ierarhia de memorii cache;
- coerența și consistența variabilelor partajate;
- rețele de interconectare;
- modele de programare paralelă;
- paralelizarea aplicațiilor;
- simularea ca instrument de cercetare;
- benchmarking;
- explorarea automată a spațiului parametrilor;
- arhitecturi multicore cu procesări anticipative – o idee originală;
- putere consumată, disipație termică.

Concluzia de bază este că sunt necesare progrese importante în toate aceste domenii, pentru ca proiectarea și utilizarea sistemelor multicore și manycore să fie adecvate. Cu sau fără voia noastră, aceste sisteme vor constitui dispozitivele de calcul universale. Abordarea oricăreia dintre aceste provocări trebuie să fie una de tip holistic. S-a arătat cum vor schimba aceste noi arhitecturi, paradigma științei ingineriei calculatoarelor. În particular, programarea acestor sisteme constituie o provocare majoră, pentru care nu suntem încă pregătiți suficient.

Bibliografie

1. **ASANOVIC, K.** et al.: The Landscape of Parallel Computing Research: A View from Berkeley, Technical Report No. UCB/EECS-2006-183, December 2006, p. 22.
2. **CRAEYNES, K. VAN, S. EYERMAN, L. EECKHOUT**: MLP-Aware Runahead Threads în a Simultaneous Multithreading Processor, Proc. of The 4th HiPEAC Int. Conf., Paphos, Cyprus, January 2009, pp. 110-124.
3. **CULLER, D., J. SINGH, A. GUPTA**: Parallel Computer Architecture: A Hardware/Software Approach, Morgan Kaufmann Publishers, 1999.
4. **DESMET, V., S. GIRBAL, O. TEMAM**: ArchExplorer.org: Joint Compiler/Hardware Exploration for Fair Comparison of Architectures, The 6-th HiPEAC Industrial Workshop, Thales, Paris, Nov. 26th, 2008.
5. **GABBAY, F., A. MENDELSON**: Using Value Prediction to Increase the Power of Speculative Execution Hardware, ACM Transactions on Computer Systems, Vol. 16, No. 3, 1998.
6. **GELLERT, A., A. FLOREA, L. VINTAN**: Exploiting Selective Instruction Reuse and Value Prediction în a Superscalar Architecture, Journal of Systems Architecture, Vol. 55, Issue 3, Elsevier, 2009.
7. **GIORGI, R., Z. POPOVIC, N. PUZOVIC**: Implementing fine/medium Grained TLP Support în a Many-core Architecture, International Symposium on Systems, Architectures, Modeling and Simulation, Samos, Greece, July 20-23, 2009.
8. **HAMMOND, L.** et al., Transactional Memory Coherence and Consistency, The 31st Annual International Symposium on Computer Architecture (ISCA), Munich, 2004.
9. **HENNESSY, J., D. PATTERSON**: Computer Architecture. A Quantitative Approach, 4th Edition, Morgan Kaufmann Publishers, 2007.
10. * * *: HiPEAC - High-Performance Embedded Architecture and Compilation (HiPEAC) FP6/FP7 Network of Excellence, Research Challenges în High-Performance Embedded Architecture and Compilation, <http://www.HiPEAC.net>, 2008.
11. **JORDAN, H., G. ALAGHBAND**: Fundamentals of Parallel Processing, Pearson Education, Inc., Prentice Hall, 2003.
12. **LIU, S., J. L. GAUDIOT**: The Potential of Fine-Grained Value Prediction în Enhancing the Performance of Modern Parallel Machines, The 13th IEEE Asia-Pacific Computer Systems Architecture Conference (ACSAC 2008), Taiwan, August 2008.
13. **MARTIN, M.**, et al.: Correctly Implementing Value Prediction în Microprocessors that Support Multithreading or Multiprocessing. Proc. of the 34-th Annual ACM/IEEE International Symposium on Microarchitecture, Austin, Texas, December 3-5, 2001.
14. **MOGA, A., M. DUBOIS**: A comparative evaluation of hybrid distributed shared-memory systems, Journal of Systems Architecture, Vol. 55, Issue 1, Elsevier, 2009, pp. 43-52.
15. **SUKHUN KANG, RAKESH KUMAR, MAGELLAN**: A Framework for Fast Multi-core Design Space Exploration and Optimization Using Search and Machine Learning, Illinois at Urbana-Champaign, CRHC Technical Report CRHC-07-05, October 2007.

16. **VINȚAN, L.:** Prediction Techniques în Advanced Computing Architectures (in limba engleza), Matrix Rom Publishing House, Bucharest, 2007; <http://www.matrixrom.ro/romanian/editura/domenii/informatica.php?id=867#867>.
17. **VINȚAN, L., A. FLOREA, A. GELLERT:** Random Degrees of Unbiased Branches, Proceedings of The Romanian Academy, Series A: Mathematics, Physics, Technical Sciences, Information Science, Vol. 9, No. 3, Bucharest, 2008 - <http://www.academiaromana.ro/sectii2002/proceedings/doc2008-3/13-Vintan.pdf>.
18. **WATKINS, M., S. MCKEE, L. SCHAELOCKE:** Revisiting Cache Block Superloading, Proc. of the 4th HiPEAC Int. Conf., Paphos, Cyprus, January 2009.
19. **YI, J., D. LILJA:** Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations, IEEE Transactions on Computers, Vol. 55, No. 3, 2006.