Principii de interconectare a unui cluster hibrid într-un sistem grid

Ovidiu GHERMAN, Ioan UNGUREAN, Ştefan Gheorghe PENTIUC

***Abstract*—The computing systems built with general purpose processors are superseded today by systems built around specialized cores - a type of CPU that is designed to operate with massive amount of arithmetic operations, typically found in graphical accelerators or in systems that manage graphical environments. A similar step was made by IBM when they proposed that a unit dedicated to arithmetical operations to be used as an accelerator node for a cluster controlled by nodes with general purpose architecture. A new approach has been made towards high performance computing, in the form of hybrid architectures. However, using machines with different architecture as a single system can pose problems in application deployment.**

***Index Terms*—Cell BE, cluster interconnect, hybrid cluster, MPI, RISC architecture**

I. INTRODUCERE

ÎN acest articol se prezintă câteva dintre caracteristicile unui cluster hibrid ce foloseşte procesoare dedicate cu arhitectura RISC (model Cell BE) pentru execuţia calculelor aritmetice, procesoare controlate de noduri de management construite în jurul unor unităţi CPU de uz general. Apoi sunt puse în discuţie câteva din problemele pe care le generează conectarea acestui tip de sistem cu alte clustere cu arhitectură clasică într-o reţea distribuită de tip grid, probleme ce pot apare în cazul dezvoltării de aplicaţii care să folosească corespunzător resursele eterogene (dedicate sau generale) într-un mod cât mai performant.

Interconectarea sistemelor de calcul paralel în platforme de tip grid (sisteme distribuite) a pus permanent probleme referitoare la infrastructura hardware pe care rulează aplicaţia de calcul distribuit. Numărul mare de arhitecturi posibile, de combinaţii de sisteme de calcul sau de tipuri de procesoare a dus la dezvoltarea unor aplicaţii de management capabile să ruleze pe multiple arhitecturi, astfel încât să elibereze utilizatorul de necesitatea de a lua în calcul arhitectura şi tipul fiecărui nod pe care îşi va rula programul. Deşi natura hardware a nodurilor poate influenţa capacitatea de interconectare a resurselor, folosirea unui asemenea middleware - similar unui ”ciment” între aplicaţiile client şi cele de management dezvoltate de administratori – poate ajuta la uniformizarea accesului la aceste resurse. Unul din cele mai folosite astfel de medii este Globus Toolkit (ajuns acum la versiunea 4.2.1), un produs al Globus Alliance (o comunitate ce face cercetări asupra tehnologiilor, standardelor şi a sistemelor de tip grid, în scopul creării unei structuri care să permită colaborarea distribuită între diverşi utilizatori - în scop ştiinţific, ingineresc, economic, etc).

Acest middleware implementează câteva standarde bine definite în lumea tehnologiei grid (OGSA, OGSI sau GSI fiind numai câteva). Utilitarele sale permit planificarea sarcinilor care vor rula pe noduri, asigurarea distribuţiei acestora, asigurarea securităţii aplicaţiilor şi a datelor furnizate de acestea, managementul resurselor, transferul datelor între noduri şi altele.

II. ARHITECTURI HIBRIDE INTERCONECTATE

O problemă mai complicată apare atunci când este vorba de o arhitectură hibridă. Un exemplu de astfel de sistem îl constituie platforma IBM Roadrunner, ce are la bază atât procesoare x86 (arhitectură CISC) cât şi procesoare Cell BE bazate pe o arhitectură similară procesoarelor PowerPC (de tip RISC). Structura unui asemenea sistem permite un model de programare specială, în care aplicaţia scrisă pentru sistem trebuie să ţină cont de caracteristicile ambelor tipuri de procesoare. Compilarea aplicaţiei este de asemenea supusă acestor restricţii. Utilizatorul trebuie să manipuleze programul său astfel încât acesta să se folosească de capacităţile deosebite de calcul ale procesoarelor de accelerare (PowerXCell 8i).

Procesorul dedicat este alcătuit de fapt din două tipuri de nuclee interne [1]. Acestea – în număr de nouă - împreună cu magistrala de interconectare şi de controlerul DMA formează o unitate de calcul puternică, controlată de unul din nucleele interne. Acest nucleu PPE (PowerPC Processing Element) are la bază o arhitectură PowerPC pe 64 de biţi dar cu toate acestea deţine – pe lângă setul de instrucţiuni original, compatibil cu procesoarele de serie cu arhitectură similară – şi un Vector / SIMD Multimedia Extension, un set de instrucţiuni creat special pentru procesoarele de calcul vectorial. Acest nucleu reprezintă principalul element de procesare, asigurând interfaţa dintre 8 celelalte nuclee subordonate şi nodul de control. Astfel, el se ocupă mai ales de managementul proceselor precum şi la împărţirea problemei în sub-probleme uşor de manipulat la nivelul unităţilor de calcul aritmetic. Necesită un compilator special (pentru aplicaţiile în C/C++ se poate folosi de obicei *ppu-g++*, instalat în mod standard pe aceasta platformă).

Elementul sinergistic de procesare (SPE) este format din opt unităţi, procesoare SIMD optimizate pentru operaţiuni intensive cu date (alocate de către PPE – unitatea SPE nu poate comunica direct cu exteriorul). Fiecare dintre aceste elemente identice conţine un nucleu RISC cu 256 KB de memorie formând un sistem de stocare local pentru instrucţiuni şi date, controlat prin software. Mai există de

asemenea şi un fişier registru unificat pe 128 de biţi având 128 de intrări, pentru operaţii în virgulă mobilă sau de tip integer. SPE suportă un set special de instrucţiuni SIMD (creat special pentru accelerarea calculelor aritmetice în virgulă mobilă) şi se bazează pe transferuri DMA asincrone pentru mutarea datelor şi instrucţiunilor între sistemul principal de stocare (spaţiul de adresare efectiv care include memoria principală) şi sistemul local de stocare. Unităţile SPE nu sunt făcute pentru a rula un sistem de operare, astfel că depind de unitatea PPE pentru a primi sarcini de lucru [2].

Avantajul acestor unităţi este performanţa lor în execuţia calculelor vectoriale (din acest motiv mai sunt numite şi acceleratoare). Ca urmare, aplicaţia utilizator trebuie să fie optimizată pentru ca aceste unităţi să fie folosite la întreg potenţialul.

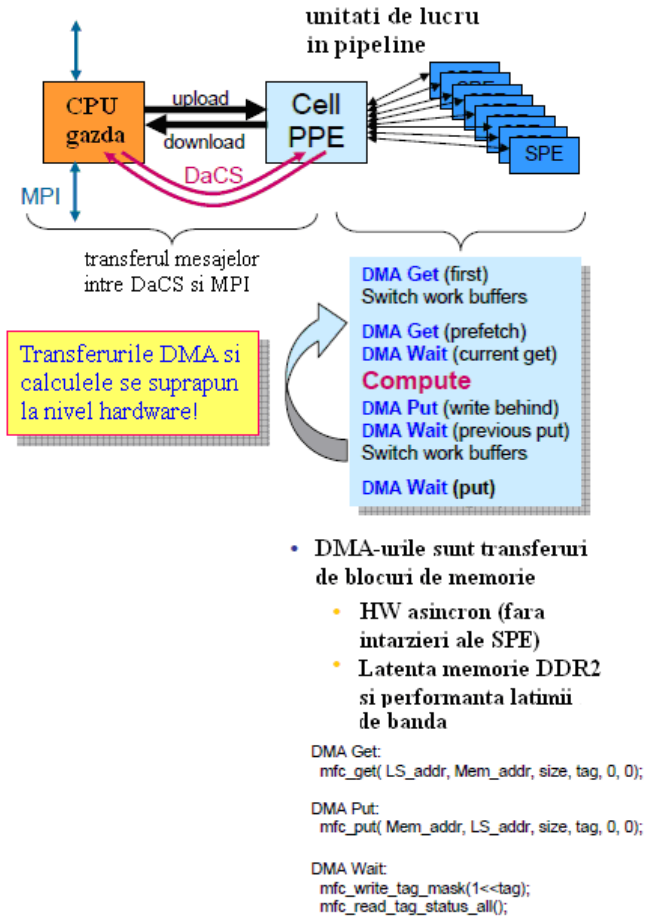

Fig. 1. Modul de lucru intern al unui procesor Cell [3].

Fiecare unitate SPE are la rândul ei un design modular, fiind formată dintr-o unitate SPU (Synergistic Processing Unit) şi o unitate MFC (Memory Flow Controller) independentă – un controller DMA ce foloseşte un modul de management al memoriei pentru a realiza operaţii sincronizate cu alte unităţi SPU de pe cip sau cu unitatea PPU. SPU foloseşte instrucţiunile şi datele preluate din sistemul de stocare local [4] şi comunică printr-un canal dedicat – integrat în MFC - cu memoria principală sau cu alte sisteme locale (aflate pe acelaşi cip fizic) – Figura 1.

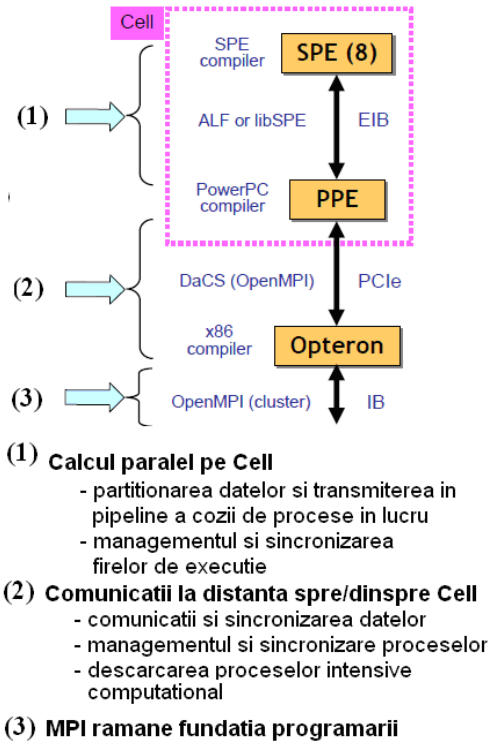

Fig. 2. Programarea în cazul arhitecturii hibride[3].

O dată problema adusă la mărimi mici, acestea sunt trimise către celelalte 8 nuclee de calcul sinergistic (SPE). Aceste procesoare vectoriale nu beneficiază de multă memorie locală (cei 256KB de memorie disponibili trebuie să conţină atât programul ce rulează pe cele 8 nuclee de calcul ale procesorului Cell) dar lucrează la frecvenţă ridicată (3.2 GHz) şi beneficiază de transferuri DMA cu memoria. Datorită acestui mod de lucru, procesorul Cell BE are performanţe deosebite când prelucrează blocuri de date de mici dimensiuni. Din această cauză este folosit cel mai eficient atunci când o aplicaţie primară (rulând pe nucleul x86 al procesorului AMD Opteron care controlează unitatea Cell BE) doreşte să facă rapid o serie de calcule de granularitate mică (partea intensivă computaţional). Mai mult, se recomandă ca aceste calcule să implice structuri de date care pot fi prelucrate vectorial (de exemplu vectorii bidimensionali trebuie prelucraţi linie cu linie). Această tehnică de ”*offloading*” necesită folosirea unei biblioteci de accelerare proprie procesorului Cell BE (DaCS şi ALF) la nivelul nucleelor SPE şi PPE, în timp ce la nivelul unităţilor de control (Opteron) se va folosi protocolul MPI pentru transmiterea datelor programelor (Figura 2) – practic vom avea un nivel dublu de paralelizare (de granularitate diferită). Scopul aplicaţiei la acest nivel poate fi fie computaţional (deşi probabil nu va atinge performanţele unui procesor dedicat de accelerare) fie de management al problemei (de exemplu formatarea datelor ce se vor prelucra sau împărţirea problemei în subprobleme alocate diverselor procesoare).

Acest fapt complică scrierea unui program care să ruleze pe o astfel de arhitectură. Pe lângă faptul că este necesară utilizarea împărţirea codului între cele două tipuri de

procesoare (în cazul aplicațiilor deja existente este nevoie de rescrierea cel puțin parțială a codului sursă), trebuie să se țină cont şi de particularitățile procesorului Cell BE dacă se doreşte menținerea performanțelor (accesul la memorie, modul de lucru cu structurile de date, modul de transfer al datelor, etc.). Dar atunci când se sistemul hibrid este folosit în conjuncție cu alte platforme este necesară abordarea unei noi tehnici de realizare a programelor [5].

III. INTERCONECTAREA SISTEMELOR HIBRIDE ŞI CLASICE

Adăugarea de noi resurse într-un sistem de tip grid poate implica noi unități de calcul cu arhitectură clasică (x86 – de obicei AMD sau Intel) dar din moment ce un procesor Cell BE nu poate comunica decât cu un procesor master (x86) de la care primeşte sarcinile de lucru, utilizatorul va trebui să îşi structureze aplicația astfel încât procesele instanțiate pe aceste noi noduri să primească sarcinile care nu necesită calcule vectoriale intensive sau sarcini ce presupun manipularea structurilor de date (înaintarea sarcinilor către procesoarele de accelerare, manipularea datelor de intrare şi de ieşire, verificări CRC, primirea de date de intrare, asigurarea descompunerii problemelor în grupuri suficient de mici şi compatibile, care pot lucra pe nucleele SPE individuale), rezervând partea computațională procesoarelor de accelerare. Modelele clasice de structurare a programelor paralele (master-slave sau cooperativ) trebuie abandonate sau regândite pentru a funcționa corespunzător şi a oferi performanțele maxime din partea procesoarelor de accelerare.

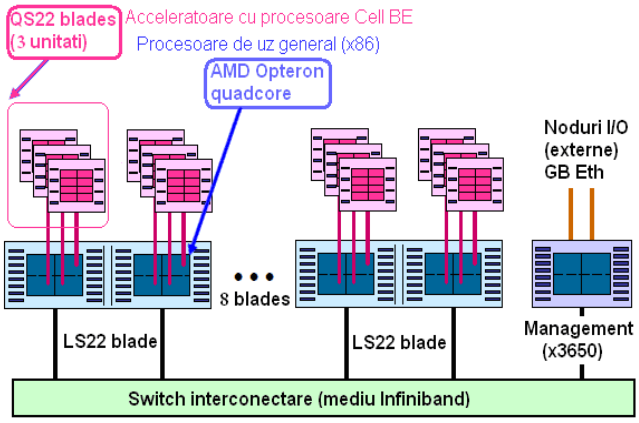

Fig. 3. Structura hibridă a clusterului ”miniRoadrunner”.

Din punct de vedere funcțional, structura clusterului este una hibridă, cu noduri de management optimizate pentru managementul proceselor şi al aplicațiilor distribuite. Aceste noduri (LS22 dual-CPU) nu sunt folosite de obicei la executarea calculelor, ci la distribuția sarcinilor şi a job-urilor. Pentru aplicațiile distribuite acestea folosesc mediul MPI (pe cluster de test fiind configurat OpenMPI 1.3.3, instalat pe sistemul de operare Red Hat Linux Enterprise Edition 5). Aceste noduri sunt în comunicație cu nodurile subordonate de tip QS22 (noduri de accelerare), fiecare procesor Opteron quadcore gestionând câte trei procesoare PowerXCell 8i – Figura 3. Acceleratoarele integrate în aceeaşi lamă nu pot comunica direct cu celelalte acceleratoare (de pe lamele vecine), ci doar cu procesorul master (prin intermediul unei magistrale PCI Express), iar sistemul de operare folosit este fie RHEL 5 (dacă se doreşte utilizarea gamei variate de unelte software de programare a aplicațiilor hibride – compilatoare, debugger-e, tracer-e, medii de dezvoltare) sau un mediu Fedora Core 9 puternic optimizat atunci când se doreşte performanță (uneltele de dezvoltare sunt mai sărace) – cele două sisteme de operare fiind interschimbabile. Aplicațiile pre-existente nu pot rula ca atare pe această platformă, ci trebuie parțial rescrise şi recompilate corespunzător.

Din moment ce procesoarele de accelerare nu pot comunica direct cu celelalte procesoare din structura de calcul ci doar prin intermediul procesorului-gazdă, vor depinde de acestea pentru a primi sarcini de calcul. Această tehnică de programare presupune practic paralelizare în paralelizare, şi poate fi privită fie din punctul de vedere al unității Cell (”*Cell-centric*”) fie din cel al procesorului-gazdă (”*host-centric*”) [3][6]. La nivelul clusterului putem folosi oricare din cele două metode pentru a scrie aplicația, mai ales că nodurile de management de tip LS22 nu contribuie de obicei la realizarea calculului efectiv, ci la managementul respectivelor unități, astfel încât nu vom mai avea trei variante de program pentru compilare, ci numai două.

În cadrul unei aplicații puternic distribuită pe grid ultima variantă este cea de dorit deoarece presupune mai multe avantaje decât alternativa. În acest caz, utilizatorul care trimite o sarcină spre prelucrare, daca doreşte să utilizeze atât resursele accelerate cât şi resurse generale (adică alte clustere sau unități de calcul conectate în grid), va trebui să îşi adapteze programul pentru compilare/rulare pe toate platformele disponibile. Bineînțeles că se poate considera dificilă programarea pe un asemenea sistem distribuit, dar adevărata problemă constă de fapt în uneltele de dezvoltare pentru arhitectura PowerPC. Testarea unui nod pentru a putea detecta arhitectura de la baza sa este relativ simplă comparativ cu problemele puse de efortul de a scrie codul sursă la un nivel dublu de paralelizare. Cu toate acestea, performanțele unui asemenea sistem (raportate de exemplu la consumul electric) sunt foarte bune (în cazul sistemului testat, acesta a fost validat ca şi performanță de un test Linpack). De aceea o soluție care rezolvă elegant această aparentă metodă ar fi descompunerea problemei în elemente caracteristice, în funcție de complexitatea reclamată de calcule.

De exemplu, o problemă care implică seturi mari de date care necesită formatare şi prelucrarea datelor se poate face în două faze distincte, în care sistemele de uz general pot să realizeze formatarea primară a fişierelor sau a datelor, urmând ca procesoarele acceleratoare să primească joburi dimensionate potrivit (trebuie să se țină cont de tehnica unităților de stocare locală de mici dimensiuni implementate la nivelul SPE) pentru a putea aplica funcțiile aritmetice necesare. Acest lucru poate fi un avantaj dacă se foloseşte de exemplu un sistem de fişiere cu acces paralele concurent (cum este GPFS, instalat pe clusterul analizat). În acest caz, mai

multe procesoare pot avea acces la diverse simultan şi concurent la diverse părți din fişier (drept absolut, de acces şi modificare), părți pe care le prelucrează şi apoi semnalizează unităților de accelerare că pot interveni, pentru a termina procesarea. De aceea este important a se folosi structuri de date care se pretează la prelucrarea în structura pipeline a unității Cell (de exemplu date care se prelucrează liniar, care nu au nevoie de valori anterioare,etc.), pentru a păstra viteza de procesare ridicată. Practic în orice problemă intervine şi momente de management,, de organizare, de asigurare a unor servicii, pentru care unitățile cu arhitecturi comune (de exemplu CISC x86) sunt mai potrivite (şi chiar mai rapide) decât acceleratoarele.

Deşi acest stil de programare este la început, tendința de a folosi astfel de unități (de procesare vectorială multicore) creşte exponențial. Datorită acestui lucru, IBM încearcă îşi extindă bibliotecile hibride şi către sisteme non-hibride (mai ales ALF şi DaCS), tocmai pentru a evita probleme de genul acesta ce pot apărea (al compatibilității). Astfel vor face probabil concurență directă mediilor de programare distribuită cum este MPI[3].

Datorită arhitecturii hibride, programarea aplicației distribuite pe nodurile eterogene trebuie să fie făcută în funcție de natura platformei pe care va rula aplicația. Dacă pentru arhitecturile x86 (cele mai des întâlnite) nu se pun probleme deosebire deoarece compilatoarele şi librăriile folosite sunt cele standard venite cu pachetul MPI la instalare, în cazul procesoarelor CELL trebuie să se folosească două compilatoare, pentru a se putea exploata codul atât pentru unitate PPU a unității Cell, cât şi al unităților SPU. Din acest motiv, este nevoie de a se folosi un wrapper mai complex pentru executarea operațiilor de copiere şi compilare în grid, dar în acelaşi timp este necesar a se scrie cod optimizat pentru unitățile de accelerare, astfel încât acestea să fie exploatate la maxim, ceea ce ar duce la o creştere de performanță mult mai mare decât în cazul folosirii acestora pentru bucăți de cod uzual, care să folosească instrucțiuni generice de lucru, şi nu calcule aritmetice.

Un exemplu de sistem distribuit interconectat care formează o astfel de arhitectură hibridă constă dintr-un cluster de tip IBM ”Roadrunner” (bazat pe 48 noduri de calcul cu procesoare Cell BE eDP, aşa cum a fost amintit mai sus) şi un cluster construit în jurul unor noduri (28) dotate cu procesoare de uz general Intel Xeon. Ambele sisteme beneficiază de conexiune externă (Ethernet Gigabit) şi de server de management pentru a se permite conectarea nodurilor de calcul la rețele externe [7]. Nodurile de calcul sunt interconectate şi comunica – la nivelul aplicației – prin intermediul mesajelor MPI (Figura 4).

Pentru ca aplicația să poată rula corespunzător, trebuie să se țină cont de platforma de lucru a fiecărui nod. Acest lucru poate deveni problematic în cazul unor sisteme necunoscute sau diversificate (resurse eterogene). De exemplu, în gridul considerat în exemplu nostru vom avea nu două ci trei tipuri de resurse, aşa cum se vede în Figura 5.

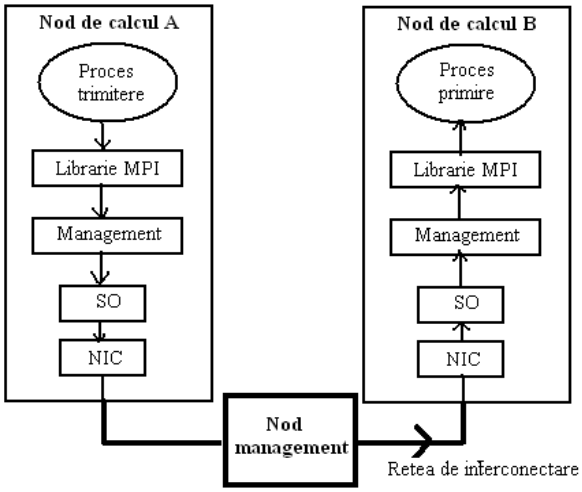

Fig. 4. Interconectarea nodurilor de calcul (nivel MPI).

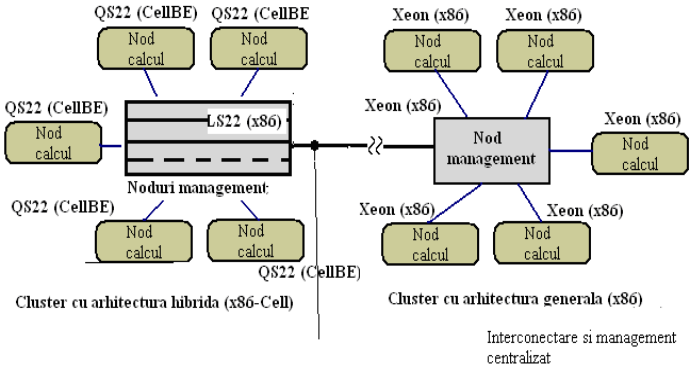

Fig.5. Sistemul distribuit (cazul exemplificat).

În acest sistem, rețeaua de comunicații este şi ea o resursă eterogenă: în timp ce clusterul bazat pe procesoare Xeon este conectat intern şi extern printr-o rețea Ethernet Gigabit, sistemul ”miniRoadrunner” are o conexiune externe Ethernet Gigabit, dar intern foloseşte protocolul Infiniband, ceea ce aduce un spor de performanță în ceea ce priveşte transferul de date precum şi o scalabilitate mai bună în momentul folosirii unui număr mai mare de noduri de calcul.

Un algoritm care să permită folosirea ambelor tipuri de sisteme în mod unitar (realizându-se astfel un sistem de grid) poate uşura munca programatorului ajutându-l să editeze, compileze şi eventual sa ruleze aplicația mult mai rapid şi dintr-o singura sursă. Un astfel de script ar trebui să poată face următoarele:

- *se editează fişierul sursă (codul programului);*
- *se transferă sursa către unitatea de calcul la distantă;*
- *se compilează sursa atât pe resursa la distanță, cât şi pe cea locala (folosind compilatoare specifice);*
- *se lansează în execuție aplicația pe ambele resurse, cu specificarea nodurilor pe care vor rula instanțele acesteia;*
- *se centralizează rezultatele.*

Dacă codul sursă dezvoltat foloseşte doar instrucţiuni din librăria MPI (generice am spune spune), aplicaţia rezultată va putea rula atât pe nodurile de uz general cât şi pe cele de accelerare, fără a fi nevoie de a scrie cod special pentru ambele tipuri de unităţi. Dar în acest caz nu se vor folosi unităţile sinergistice de accelerare, ceea ce ar duce la o exploatare incompletă a capacităţilor pentru care a fost de altfel proiectat. Oricum, procesoarele Cell sunt suficient de rapide încât să facă faţă unităţilor de uz general, chiar şi în mod ”brut”. Oricum, puterea de calcul disponibilă poate fi exploatată pentru rezolvarea diverselor probleme.

O implementare de test a unui asemenea *shell script* ce ajută la interconectarea celor două clustere descrise mai sus (in care sistemul de tip ”Roadrunner”, mai performant, a folosit doar *patru* procesoare comparativ cu sistemul bazat pe procesoare Xeon) ne arată că interconectarea sistemelor la nivelul aplicaţiei este posibilă, iar primul sistem (care, deşi beneficiază de conectică Infiniband, conţine mai multe noduri care prelungesc timpul în care o problemă de mici dimensiuni este distribuită) este eficient o dată cu creşterea cantităţii de date furnizate. Frecvenţa ridicată de funcţionare, accesele DMA la memorie şi structura pipeline cresc performanţa sistemului începând cu un anumit punct, punct de la care overhead-ul datelor este minimizat de volumul acestora. Un grafic care indică creşterea timpului de execuţie în funcţie de mărimea problemei (între componentele de arhitecturi diferite – în acest caz programul de test a fost o aplicaţie ce obţine produsul a două matrici, operaţie ce se pretează unităţii Cell BE, care se observă că este avantajat o dată cu creşterea volumului de date prelucrat) şi este redat în Figura 6. Timpii mai mici pentru număr de elemente mai mare sunt mai buni.

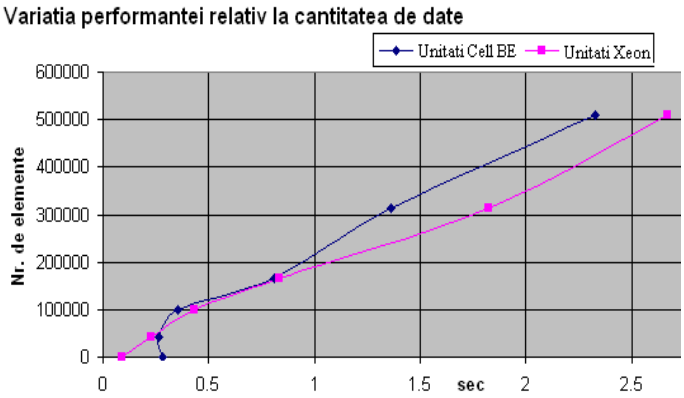

Fig. 6. Variaţia performanţei relativ la creşterea volumului de date prelucrate (comparativ arhitecturi).

O abordare mai calculată ar permite descompunerea problemei în *tipuri* de sarcini, care apoi vor fi emise numai nucleelor de calcul potrivite pentru sarcina respectiva. Această metodă de programare solicită în plus crearea a trei tipuri de programe: pentru unităţile SPU şi PPU precum si pentru cele pe platformă x86. În plus, trebuie folosit un script pentru a compila aplicaţia în mod simultan pe toate maşinile, şi apoi, bineînţeles, rularea pe nodurile specificate (nodurile de management al lamelor de accelerare nu vor rula de obicei aplicaţii distribuite (la nivel de MPI) pentru a nu fi congestionate cu sarcini suplimentare).

IV. CONCLUZII

Sistemul de tip cluster hibrid este un sistem puternic pentru îndeplinirea sarcinilor HPC, care beneficiază de o tehnologie de ultimă oră. Scalabilitatea deosebită a acestui tip de sistem (aşa cum a fost dovedit pe sistemul mult mai mare aflat în posesia laboratoarelor de la Los Alamos, această scalare este aproape liniară chiar şi pe un sistem cu peste 10.000 de procesoare) duce la posibilitatea de extindere a numărului de unităţi de calcul într-un mod simplu şi fără modificări de infrastructură (prin adăugarea de lame LS22 şi QS22 în proporţia 1:6 în cazul sistemului testat).

De asemenea, acest tip de sistem este foarte eficient din punct de vedere al consumului energetic, depăşind în această privinţă toate platformele similare ca performanţă (formate – în marea majoritate – din procesoare de uz general). Astfel, eficienţa energetică a acestui sistem hibrid este de 437 MFlops/watt, comparativ cu sisteme echivalente, ale căror eficienţă nu depăşeşte de obicei 378 MFlops/watt (BlueGene).

Totuşi, probleme ce pot apare la integrarea acestui cluster într-o reţea grid mai mare pot fi atenuate folosind o metodă de programare care să împartă problema de rezolvat în funcţie de natura ei, şi apoi de trimis fiecare parte procesorului specializat în rezolvarea tipului respectiv de problemă. Se pot delega către arhitecturile clasice toate problemele care implică lucru cu fişiere, formatări, interogări, cu alte cuvinte tot ceea ce nu ţine de calcul brut. Calculul vectorial este de preferat, sau cel matricial. De exemplu chiar în acest caz, procesoarele generice pot descompune matricea în vectori care să fie uşor de folosit de unităţile acceleratoare, apoi să le asambleze la final maximizându-se astfel câştigul de pe urma rulării pe o astfel de arhitectură hibridă. Natura specializată a procesoarelor Cell BE face ca acest tip de conectare să prezinte avantaje semnificative în cadrul anumitor probleme matematice (în care nodurile construite pe baza arhitecturilor de uz general nu dezvoltă aceeaşi performanţă), în timp ce pentru problemele ce nu folosesc intensiv astfel de rutine matematice, nodurile acceleratoare pot fi chiar limitate din anumite puncte de vedere (lipsa unei memorii locale mari, de ex.). Folosirea unei aplicaţii ce utilizează în mod general procesorul sinergistic (în mod ”brut”, adică fără a apela la bibliotecile speciale de accelerare) poate permite rularea unei aplicaţii la nivel distribuit, pe un sistem grid, dar nu în cele mai bune condiţii de performanţă. Dacă se doreşte o accelerare semnificativă a calculelor, este nevoie dezvoltarea unei aplicaţii speciale, care să furnizeze cod atât pentru arhitectura de uz general, cât şi pentru cea hibridă. Dezavantajul acestui lucru constă în faptul că este necesară practic scrierea a două aplicaţii (ceea ce duce de altfel la un timp de dezvoltare crescut) şi compilarea lor separată pe fiecare arhitectură.

MULȚUMIRI

Clusterul hibrid analizat a fost achiziționat din proiectul grid intitulat ”Grid pentru dezvoltarea aplicațiilor de recunoaştere a formelor şi aplicații distribuite de inteligență artificială - GRIDNORD”, 80/13.09.2007, PN II, Programul Capacități.

REFERINȚE

[1] Thomas Chen, Ram Raghavan, Jason Dale, Eiji Iwata, *Cell Broadband Engine Architecture*, IBM Journal of Research and Development, vol. 51, issue 5, 2007

[2] ****, *Cell Broadband Engine – Programming Handbook v1.1*, IBM Systems and Technology Group, https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/7A77CCDF14FE70D5852575CA0074E8ED

[3] Ken Koch, *Roadrunner Platform Overview*, Roadrunner Technical Seminar Series, Los Alamos National Laboratory, martie 2008

[4] ****, *SPU Application Binary Interface Specification v1.9*, CBEA JSRE Series, Cell Broadband Engine Architecture, Join Software Reference Environment Series, July 2008

[5] Christoph Kessler, *Programming Techniques for the CELL Processor*, Multicore Day 2009, Kista, Suedia

[6] John A. Tuner, *The Los Alamos Roadrunner Petascale Hybrid Supercomputer – Overview of Applications, Results and Programming*, Roadrunner Technical Seminar Series, Los Alamos National Laboratory, martie 2008

[7] Brett M. Bode, Jason J. Hill, Troy R. Benjegerdes, *Cluster Interconnect Overview*, Proceedings of the annual conference on USENIX Annual Technical Conference, Boston, 2004