Budowa i użytkowanie klastrów w opaciu o układy Cell BE oraz GPU

Daniel Kubiak

Wydział Inżynierii Mechanicznej i Informatyki
Kierunek informatyka, Rok V
sor@czlug.icis.pcz.pl

Streszczenie

Celem pracy jest przedstawienie wyników badań nad efektywnością oraz opłacalnością budowy klastrów HPC przy użyciu wielordzeniowych układów Cell B.E. oraz GPU. Badania zostały przeprowadzone przy użyciu implementacji algorytmów mnożenia macierzy A * B = C. Zostały również przedstawione problemy wynikające ze specyficznego podejścia w programowaniu układów wielordzeniowych wymuszonego przed producentów poszczególnych SDK.

1 Wstęp

W obecnych czasach w branży IT obowiązuje moda na tzw. Green IT, czyli budowanie energooszczędnych systemów komputerowych, a problem ten przede wszystkim dotyczy tematyki HPC (High Performance Computing). Ograniczenia wynikające z taktowania układów scalonych bazujących na krzemie przyczyniły się do rozwoju procesorów wielordzeniowych, co spowodowało prześciganie się producentów w ilości rdzeni obliczeniowych w jednym układzie. Wielordzeniowe układy CPU zużywają jednak dość dużo energii a przy tym wydzielają sporo ciepła zwiększając nakłady na utrzymanie infrastruktury IT. Wielkim krokiem w dziedzinie obliczeń wysokiej wydajności było stworzenie przez konsorcjum STI układu Cell/B.E., który umożliwił budowanie klastrów obliczeniowych o dużej wydajności przypadającej na Wat pobieranej energii. W ostatniej liście Green500 (lista najbardziej energooszczędnych superkomputerów opracowywanej na podstawie listy Top 500) aż 6 pierwszych miejsc to klastry oparte na w/w układach. Największy wyścig wydajności układów można jednak zaobserwować w innej branży, co obserwujemy od kilku ostatnich lat. Dwaj czołowi producenci układów dla kart graficznych prześcigają się w wydajności, natomiast architektura układów GPU kart graficznych pozwala na użycie ich do akceleracji obliczeń z bardzo dobrym skutkiem. Wszystko to sprawia, że stosunek mocy obliczeniowej przypadającej na Wat zużywanej energii stale rośnie i to w dość szybkim tempie.

2 Środowisko testowe

Do testów zostały użyte maszyny obliczeniowe zainstalowane w Instytucie Informatyki Teoretycznej i Stosowanej:

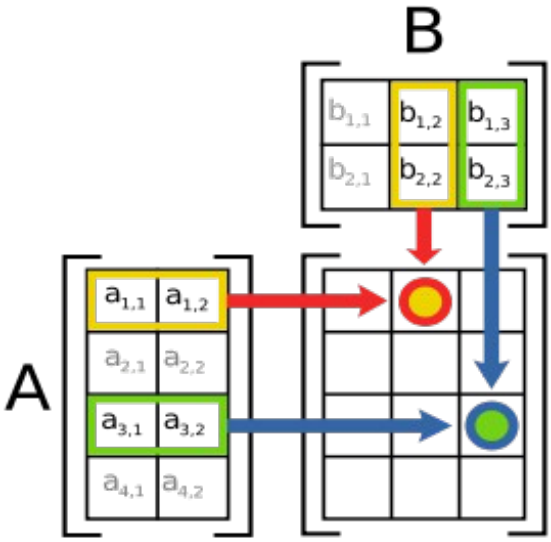
- clusterix architektura Itanium2 (4 węzły 4x1.4GHz)
- blade architektura Cell B.E. IBM QS21 (2x3.2GHz)
- tesla architektura nVidia CUDA (nVidia Tesla C1060)

W poniższej tabeli zestawiono wybrane procesory w celu porównania wydajności, ilości rdzeni oraz zużycia energii

Procesor	Itanium 2	Cell/B.E	Tesla C1060	PowerXCell8i	Tesla C2070
TDP	104 W	80 W	188 W	92 W	247 W
Wydajność SP	~4 Gflops	204,8 Gflops	933 Gflops	204,8 Gflops	1,03 Tflops
Wydajność DP	~4 Gflops	14,6 Gflops	78 Gflops	102,4 Gflops	515 Gflops
Ilość rdzeni	2	8+1	240	8+1	448

1 Podstawowy algorytm mnożenia macierzy

Podstawowy algorytm mnożenia macierzy $A \cdot B = C$ polega na odpowiednim sumowaniu składowych iloczynów poszczególnych części wektorów kolumn i wierszy. Można to zobrazować w następujący sposób:



Implementacja w/w algorytmu w języku C++ jest stosunkowo prosta:

```
for(_uint i=0; i<rsize; ++i)
     for(_uint j=0; j<csize; ++j)
       for(_uint k=0; k<csize2; ++k)
            C[i][j]+=A[i][k]*B[k][j];
```

:

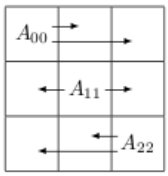
2 Algorytm Foxa

W programowaniu równoległym i rozproszonym stosuje się różnorakie algorytmy mnożenia macierzy, a jednym z nich jest algorytm Foxa.

Algorytm Foxa składa się z tylu kroków (K), na ile wierszy podmacierzy zostały podzielone macierze A i B. Każdy krok składa się z:

- rozesłania wzdłóż wierszy podmacierzy A_{ij} z k-tej przekątnej, przy czym k jest numerem kroku, a k-tą przekątną tworzą podmacierze $A_{i,(i+k) \bmod K}$
- mnożenia otrzymanych podmacierzy A_{ij} z podmacierzami B_{ij} na poszczególnych procesorach i dodawanie do sumy częściowej
- rolowanie podmacierzy B, wzdłuż kolumn w kierunku mniejszych indeksów

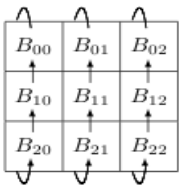
Operacje w poszczególnych krokach możemy przedstawić graficznie:
dla k=0:
Rozesłanie przekątnej macierzy A:



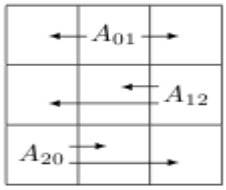
Mnożenie podmacierzy:

$$\begin{pmatrix} C'_{00} & C'_{01} & C'_{02} \\ C'_{10} & C'_{11} & C'_{12} \\ C'_{20} & C'_{21} & C'_{22} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{00} & A_{00} \\ A_{11} & A_{11} & A_{11} \\ A_{22} & A_{22} & A_{22} \end{pmatrix} \cdot \begin{pmatrix} B_{00} & B_{01} & B_{02} \\ B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \end{pmatrix}$$

Rolowanie macierzy B:



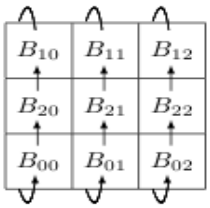
Dla k=1:
Rozesłanie przekątnej macierzy A:



Mnożenie podmacierzy:

$$\begin{bmatrix} C''_{00} & C''_{01} & C''_{02} \\ C''_{10} & C''_{11} & C''_{12} \\ C''_{20} & C''_{21} & C''_{22} \end{bmatrix} = \begin{bmatrix} C'_{00} & C'_{01} & C'_{02} \\ C'_{10} & C'_{11} & C'_{12} \\ C'_{20} & C'_{21} & C'_{22} \end{bmatrix} + \begin{bmatrix} A_{01} & A_{01} & A_{01} \\ A_{12} & A_{12} & A_{12} \\ A_{20} & A_{20} & A_{20} \end{bmatrix} \cdot \begin{bmatrix} B_{10} & B_{11} & B_{12} \\ B_{20} & B_{21} & B_{22} \\ B_{00} & B_{01} & B_{02} \end{bmatrix}$$

Rolowanie macierzy B:



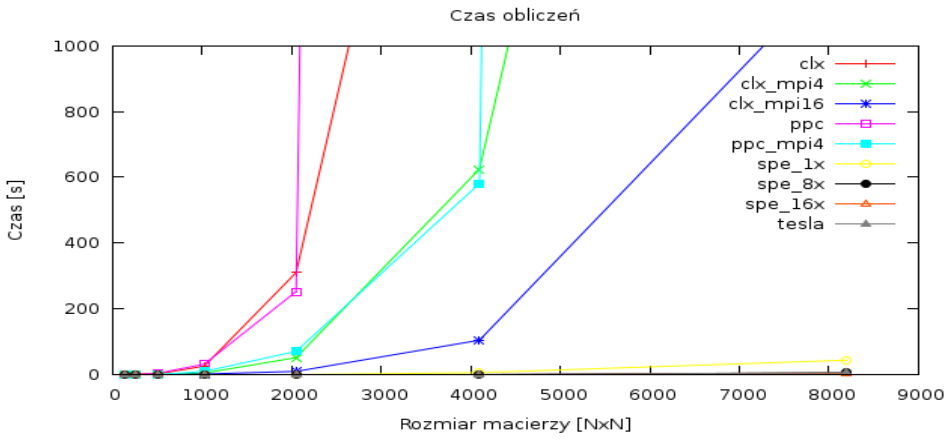
1 Badanie wydajności

W wersji MPI zaimplementowano algorytm Foxa. W wersji dla pojedynczych rdzeni zaimplementowano algorytm sekwencyjny. W wersjach korzystających z rdzeni SPE wykorzystano implementację Daniela Hackenberga (biblioteka zwiększająca wydajność obliczeń). Dla GPU wykorzystano przykładowe kody dostarczane z SDK CUDA. Wszystkie obliczenia zostały wykonane dla liczb pojedynczej precyzji (float).

Czas obliczeń:

clx	clx_mpi_4	clx_mpi_16	ppc	ppc_mpi_4	spe_1x	spe_8x	spe_16x	GPU
0	0,01	0,09	0	0,02	-	-	-	0
1	0,07	0,13	0	0,13	-	-	-	0
2	0,41	0,34	4	0,78	0,02	0,02	0,02	0
25	4,36	1,06	31	9,12	0,09	0,02	0,02	0,02
310	51,66	9,45	252	70,31	0,68	0,09	0,06	0,1
2707	625,69	104,02	35547	581,08	5,41	0,69	0,04	0,74
22452	5420,63	1261,02		71730,5	43,24	5,42	2,73	5,75

Na poniższym wykresie przedstawiono czasy obliczeń dla wszystkich testów:

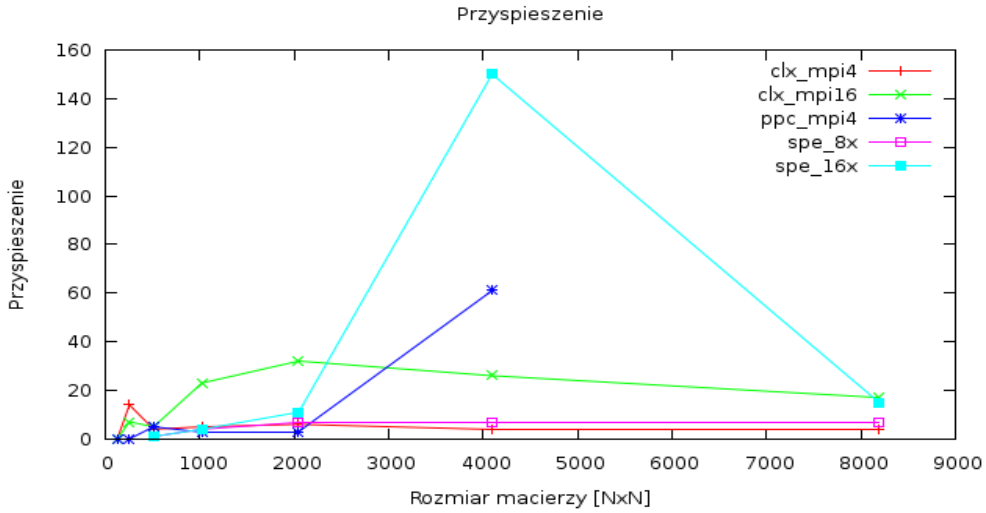


Przyspieszenie Sp jest to stosunek czasu wykonywania wersji sekwencyjnej Ts do czasu wykonywania wersji równoległej Tr:

$$Sp = \frac{Ts}{Tr}$$

	clx_mpi_4	clx_mpi_16	ppc_mpi_4	spe_8x	spe_16x
128	0	0	0	-	-
256	14,49	7,52	0	-	-
512	4,94	5,87	5,12	1	1
1024	5,73	23,61	3,4	4,5	4,5
2048	6	32,81	3,58	7,56	11,33
4096	4,33	26,02	61,17	7,84	150,28
8192	4,14	17,8	-	7,98	15,84

Na poniższym wykresie przedstawiono przyspieszenie dla poszczególnym obliczeń:

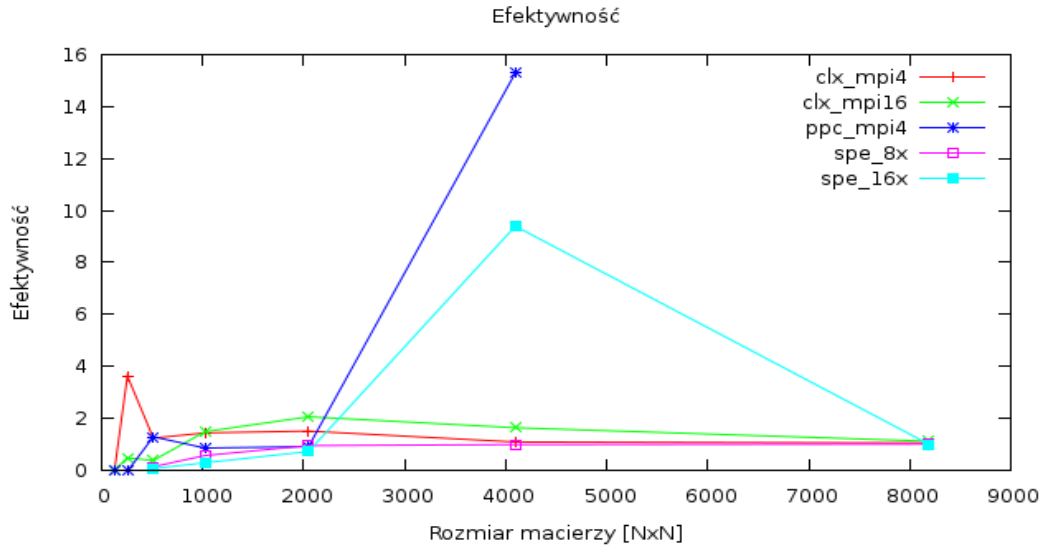


Efektywność Ep to stosunek przyspieszenia Sp do ilości rdzeni obliczeniowych lub wątków P.

$$Ep = \frac{Sp}{P}$$

	clx_mpi_4	clx_mpi_16	ppc_mpi_4	spe_8x	spe_16x
128	0	0	0	-	-
256	3,62	0,47	0	-	-
512	1,23	0,37	1,28	0,13	0,06
1024	1,43	1,48	0,85	0,56	0,28
2048	1,5	2,05	0,9	0,94	0,71
4096	1,08	1,63	15,29	0,98	9,39
8192	1,04	1,11	-	1	0,99

Na poniższym wykresie przedstawiono efektywność Ep dla poszczególnych obliczeń:



2 Podsumowanie

Powyższe wyniki jednoznacznie pokazują profity ze stosowania akceleratorów do obliczeń równoległych. Przewagą wielordzeniowych procesorów ogólnego przeznaczenia (CPU) zdecydowanie jest łatwość pisania aplikacji na tą platformę. W przypadku układów Cell/B.E. oraz kart graficznych (GPU) programowanie nie jest sprawą łatwą, gdyż wymagają od nas stosowania zupełnie innego podejścia do programowania. Biorąc pod uwagę oszczędności z czasu obliczeń oraz zużytej energii elektrycznej warto poświęcić więcej czasu na dostosowanie obecnych aplikacji równoległych do korzystania z akceleratorów. Teoretyczna wydajność procesorów GPU oraz szybkość rozwijania środowisk programowania (CUDA, OpenCL) sprawia, że stają się one przyszłością rynku komputerów wysokiej wydajności, co możemy zauważyć przy okazji chińskiego superkomputera Nebulae, którego teoretyczna wydajność wynosi prawie 3 Pflopsy.

Bibliografia

[1] Roman Wyrzykowski, *Klastry komputerów PC i architektury wielordzeniowe: budowa i wykorzystanie*, Warszawa 2009
[2] Łukasz Łaciński, *Praktyczne algorytmy mnożenia macierzy*, 2002
[4] nVidia, nVidia CUDA Programming Guide Verssion 3.0, 20/2/2010
[5] Top 500 List *http://www.top500.org*
[6] Green 500 List *http://www.green500.org*
[7] Lam-MPI Documenation, *http://www.lam-mpi.org/using/docs/*
[8] Daniel Hackenberg - *Fast Matrix Multiplication on Cell (SMP) Systems* – http://www.tu-dresden.de/zih/cell/matmul