Dynamic, unbalanced distribution of tasks on a PS3 cluster system for double precision calculation

Cristian Andy Tănase · Vasile Gheorghită Găitan

Published online: 14 September 2012

© The Author(s) 2012. This article is published with open access at Springerlink.com

Abstract The article aims to describe an algorithm for dynamic distribution of tasks in a parallel computing system consisting of a network of 9 PlayStation3 (PS3) stations. Due to the fact that the SPU cores do not have units of double precision calculation, these calculations require a longer time. Therefore, an unbalanced distribution of computing tasks on a heterogeneous cluster system as $9 \times \text{PS3}$ is necessary. The algorithm produces an estimation of the value of π by calculating the integral $\pi = \int_0^1 \frac{4}{1+x^2}$ and makes PPU-SPU empirical distribution of tasks on a smaller number of intervals. After determining the optimum loading between cores $18 \times \text{PPU}$ (PowerPC Unit) and $54 \times \text{SPU}$ (Synergetic Processor Unit), the integral calculation algorithm goes to a larger number of intervals.

Keywords Cell B.E. · Cluster PS3 · π calculation · Unbalanced distribution · Double precision

1 Introduction

The authors want to build a heterogeneous multicore-manycore computing system and implement a dynamic distribution algorithm in double precision calculations. Thus, an HPC system that consists of 9 PS3 systems (one of them is the master node) connected through a Gbyte router was assembled and set up. After installing the operating system on each node, an OpenMPI library and Cell SDK 3.0 development tools is installed. The algorithm is developed to calculate the approximate value of π , using

C.A. Tănase (✉) · V.G. Găitan
University “Stefan cel Mare” of Suceava, 13 Universitatii, 720229 Suceava, Romania
e-mail: andy@eed.usv.ro

V.G. Găitan
e-mail: gaitan@eed.usv.ro

$\pi = \int_0^1 \frac{4}{1+x^2}$ formula. MPI library was initially used to distribute the calculation area on the 18 threads with the following command: `mpirun -np 18 pi value`, where *value* is the computing resolution and was chosen with the following values: 10^5 , 10^6 , 10^7 , 10^8 , 10^9 , 10^{10} , 10^{11} , 10^{12} . For each resolution, we obtained different values of error and computing time. After these tests, the intent is to introduce in the calculation the SPU cores on each node (9×6 SPU). Thus, it makes a distribution area integral calculation for the 72 processes available for $9 \times$ PS3 cluster. Because the SPU cores do not have a double precision unit, these calculations are performed with great delay. So, we have two possibilities to solve this dilemma: The first one is to return to the calculations distribution only on the PPU 18 processes and the second one is to introduce also the SPU systems into account, but giving them a smaller amount of calculation than to the PPU units. An adaptive algorithm that distributes the total amount of computation between PPU and SPU units is performed. Distribution report is given by a k coefficient that is empirically determined from a mathematical formula for a small number of intervals to calculate the integral. After determining the k factor, it applies automatically to all values in the range: 10^5 – 10^{12} . The coefficient k is determined for the minimum time, when the PPU and SPU units complete their calculations in an equal time (with a margin of 5 %), from the tests results that the k coefficient has different values depending on the number of mathematical calculations performed. The calculation of the coefficient k is presented for different methods of calculating the integral: the trapeze method, the rectangle method, and the experimental results obtained in different distribution models: MPI (only PPU threads) and MPI-SDK (the PPU and SPU threads).

2 Experimental resources

2.1 PS3 cluster architecture and setting

We chose to use the PS3 system in the cluster construction, due to their outstanding characteristics that make it suitable for scientific calculation. Some of these features would be:

- PS3 is open-platform—meaning, it can run different operating systems (e.g., Fedora Core 8 for PPC).
- The PS3 System contains Cell B.E. processor slightly different from the original (and $6 \times$ SPU $1 \times$ PPU).
- Very low price approximately \$300 makes it very attractive as a computing node in a cluster system.

The architecture of the communication system is the tree type shown in Fig. 1.

The following steps are presented in cluster configuration:

- Formatting the 9 PS3 nodes.
- Installing the operating system Fedora Core 8 for PPC 64.
- Installing the SSH service on each station and NFS, used to MPI communication and file sharing.
- NFS setting up master node station and the other eight client stations.

Fig. 1 PS3 cluster architecture

- Installed and configures OpenMPI library on all stations.
- Installing Cell SDK 3.0 on each station.

The PS3 nodes are connected to 1000BASE-T Gigabit Ethernet switch. For this algorithm, the Gbyte switch (24 ports) is not a bottleneck because there is not a high data traffic through it. The total time to set up the threads on every node and calculation is greater than communication time. The following equation is total time of program execution $\text{Total Time} = t_{\text{setup}}(n) + \max(t_{\text{com}}(i)) + t_{\text{SPU}}(6) + \max(t_{\text{calculation}}(1/n)) + t_{\text{PPU}\pi} + \max(t_{\text{com}R}(i)) + t_{\text{master}_n\text{ode}\pi}$;

- $t_{\text{setup}}(n)$ —time to set up the nodes; initiate PPU and SPU threads.
- $\max(t_{\text{com}}(i))$ —max of times to transmit initial parameters to PPU threads (MPI-library).
- $t_{\text{SPU}}(6)$ —max of times to transmit initial parameters to SPU threads (CellSDK-library).
- $\max(t_{\text{calculation}}(1/n))$ —max time of π fractions.
- $t_{\text{PPU}\pi}$ —time to add π fractions from SPU to PPU.
- $\max(t_{\text{com}R}(i))$ —max time to send the partial result from nodes to master node using MPI library.
- $t_{\text{master}_n\text{ode}\pi}$ —time to add π fractions from nodes to master node.

$\max(t_{\text{com}}(i)) + t_{\text{SPU}}(6) + t_{\text{PPU}\pi} + \max(t_{\text{com}R}(i)) + t_{\text{master}_n\text{ode}\pi}$ is negligible in relation to $\max(t_{\text{calculation}}(1/n))$; $\max(t_{\text{calculation}}(1/n)) \gg t_{\text{setup}}(n)$.

Even if it increases the number of nodes (e.g., 23×23 nodes) in a two-level tree architecture, the total time of calculation decreases with $t_{\text{calculation}}(1/n)$ and increases with time to set up the 1058 threads on 529 nodes.

2.2 Cell architecture

CELL processor is a heterogeneous multicore architecture, being developed by a IBM-Sony-Toshiba consortium. It is built around a PowerPC processor, 64 bit (PPE), eight-core computing SIMD (SPEn), a memory controller, and an I/O controller interface. The communication between computing elements PPU-SPUs is done through a high speed bus—Element Interconnect Bus (EIB). At a clock frequency of 3.2 GHz, the maximum theoretical performance for SPE (Single Precision SP) is 25.6 GFlops, resulting in a performance of 204.8 GFlops overall for 8 SPUs. For (DP-Double Precision) theoretical maximum performance for a single SPE is 12.08 GFlops and 102.4 GFlops for 8 SPUs [8].

EIB provides a maximum bandwidth of 204.8 GB/s data transfer on-chip between the PPU, SPU, memory interface, and I/O controller. Memory controller interface provides a bandwidth of 25.2 GB/s with main memory. PPU unit runs the operating system and drives the SPUs. PPU memory hierarchy is similar to the units of conventional processors with 32 KB L1 cache and 512 KB L2 cache.

SPEs are designed for high performance processing of massive data and intensive calculation. The SPSs memory hierarchy consists of a set of 128×128 -bit SIMD registers, 256 KB of local memory (LS-Local Store) and off-chip main memory shared through PPU unit. SIMD operations can run with four different granularity: 16 of 8-bit integer, 8 of 16-bit integer, 4 of 32-bit integer, or single precision floating point numbers and 2 of 64-bit double precision floating point numbers. Local memory (LS) and an SPS can hold code and data and can only access code and data of his memory.

Data transfers between LS memories and the main memory, as well as, the transfers between LS memories are achieved by DMA. DMA transfers are asynchronous and allow SPEs to overlap computation times with transfer times. LS memory management is done exclusively by software. Mailboxes mechanisms for communication between the PPE and SPEs are provided. Each mailbox is able to retain up to four 32-bit data elements at any time [4].

Block diagram of the Cell Broadband Engine processor is shown in Fig. 2.

3 Experimental work

The algorithm which will be described can be ported to another AMP architecture (asymmetric multicore platforms) which includes two levels of performance processors: Fast Cores and Slow Cores—dual ISA heterogeneous multicore architecture [2]. The k coefficient is used to unbalanced distribution of tasks on PPUs and SPUs so that its time work and energy consumption is balanced. This applies only to systems with performance asymmetry (SPU-Slow Core and PPU-Fast Core) [7]. In fact, SPU's is much faster, but they are made for image processing. In mathematical calculation is slower because they don't have a double precision unit (only in PS3 systems).

Fig. 2 Block diagram of the Cell/B.E.

In [6], the authors present an implementation of core features of MPI for Cell. This implementation views each SPU as a node for an MPI process. In our implementation, we use not only SPU's but also PPU units. On every PPU unit is started two Linux threads. The application is started on the master node with *mpirun -np 18 pi value* command. The *-np 18* argument means that the master node will initiate 2 processes on every node using MPI library. In the same time, the master node send to the other nodes the k coefficient and a part of equally distributed from total amount quantity of calculation using MPI library. The even thread on every PPU unit (every node) starts one thread on every SPU processor and sends to SPU threads the quantity of calculation, using SDK-mailbox facility. The quantity of calculation for SPU threads is determined using k coefficient. After that, PPU even thread makes own calculation and waits for receive the fraction of π calculation from every SPU thread and add them (using the same SDK-mailbox facility). The odd PPU thread makes only own π fraction calculation. Finally, the master node receives the all fractions of π value from every nodes (18 threads) and adds them (using MPI library).

The k coefficient depends on the ratio of power computing between Fast Cores (PPU) and Slow Cores (SPU) and number of them. Even if we increase the number of nodes the optimal k is the same, because the nodes have the same hardware architecture (k is linear with scalability of the system). In [9], the authors are using a variety of algorithms for a few communications operations. In our situation, we do not need to use these types of communications because we are using only mailbox communication to send to each SPU only start, stop, and resolution values for every interval of integral assigned to SPUs.

3.1 Trapezoid area calculation method using the MPI distribution

Initially, we started to implement a calculating algorithm which had the approximate value of π by calculating integral area with trapezoid method (Fig. 3) [5]. Using the MPI library, two threads (PPU is dual-threading) are distributed to each node the

Fig. 3 Formula for π calculation using trapezoid method

command `mpirun-np 18 pi value` starts 18 threads on the 9 stations. Each of these threads calculates an equally distributed part of integral, defined in C code [3]:

```
h = 1. / (double) n;
for (i = me + 1; i <= n; i += nprocs) {
x = (i - 1) * h;
piece=piece + (4 / (1 + (x) * (x)) +
4 / (1 + (x + h) * (x + h))) / 2 * h;
}
```

me variable is the rank of the process (between 1 and 18). *n* variable defines the computing precision and receives the value of *value* parameter. *h* variable is the step through which the *x* variable is incremented in integral area calculation of $0 \rightarrow 1$.

After running the test program, the following results are obtained (trapezoid method), with *n* between 10^5 and 10^{12} (Fig. 4—marked with circle). The Fig. 4a graph is showing that the increase of the resolution determines the increase of computing time of calculation. The error (Fig. 4b) decreases to the value of 10^{11} and after that is growing because of using double precision variables. We used the notation $\text{SQRT}(\text{Time})$ to get a legible notation. The same for the error, we used $(20 + \text{LOG}(\text{Error}))$ to obtain a graph over the vertical axis.

3.2 Trapezoid area calculation method using MPI-SDK distribution

We want to use SPU cores in calculating the approximate value of π . Thus, the even process on each PPU, starts the six SPU processes. There is a total of $6 \times 9 = 54$ processes. After initialization, the mailbox is used to send the required values for calculation of a certain segment of the area integral. With a total of $18 \text{ PPU} + 54 \text{ SPU} = 72$ processing units, it is hoped to yield approximately 400 %. We designed an algorithm for computing balanced distribution of fractions for each processing unit. In fractions, distribution was used MPI library for sending the computing values on every node and CellSDK 3.0 libraries for factions distribution on SPU's within each node. The final result was three times weaker. The reason is that SPU units do not have a double precision computing unit. These calculations are performed using several times the unit simple precision. Thus, we try to design an architecture for dynamic unbalanced

(a)

(b)

Fig. 4 Comparison between values of time (a) and error (b) obtained by two methods: classical MPI 18 PPU threads and optimal unbalanced method MPI-SDK 18 PPU threads and 54 SPU threads (trapezoid method)

distribution of computing fractions between PPU and SPU's sites. Below is shown the algorithm that divides $0 \rightarrow 1$ interval between PPU and SPU units depending on the coefficient k . The value of n' is recalculated to a value as close to the received argument (between 10^5 and 10^{12}) but equally distributed between all the PPUs and

Fig. 5 Distribution of segments dedicated to each node

SPUs. Mathematical formulas for calculating the number of intervals for each process are listed below:

$$nppu = \left\lceil \frac{n - \frac{n}{k}}{nprocs} \right\rceil \cdot nprocs \quad (1)$$

$$n' = \left(\left\lceil \frac{nppu}{nprocs} \right\rceil + \left\lceil \frac{n - nppu}{nprocs} \right\rceil \right) \cdot nprocs \quad (2)$$

$$nsps = n' - nppu \quad (3)$$

A graphical representation of distributions is given in Fig. 5.

The coefficient k is determined by the algorithm by successive testing small values of n and determined by the optimal time achieved in a series of count steps (Fig. 6). k calculation has two steps: first— k determination and second π calculation using k coefficient for optimal distribution. First step start with $n = 10^3$. Then determine the mean of times calculation for PPU threads and SPU threads. If this two mean of times is dramatically different, then increase or decrease k coefficient until this means is approximatively equal. If k coefficient is determined for $n = 10^3$ and count = 0, then make $n = 10^4$, count = 20 and restart a finest calculation of k .

After determine the k coefficient, the value of n gradually increases from 10^5 to 10^{12} to obtain a better approach of π . Optimal value for k was determined to be approximately: $6 < k < 10$. For values of k greater than 10, decreases the effectiveness of using SPU units. The chart to determine the coefficient k is shown in Fig. 6. $nppu$ variable represents the number of points calculated for a PPU unit. $nsps$ variable represents the number of points calculated for the SPU unit. Variable $nprocs$ is the number of processes launched on the cluster system (between 1 and 18).

Figure 4 (marked with plus) shows the results from running the test programs `ppu.c` and `spu.c` on 18 threads ($9 \times 2PPU$) and 54 threads ($9 \times 6SPU$) with an optimal ratio unbalance, determined on small values of n , $k = 7.3$ and n' between 10^5 and 10^{12} . It can be seen that the way to increase computing time meets the same form as shape marked with circle. It can be concluded that the coefficient k , determined for the optimal time for small values of n , is optimal for large values of it that is linear with increasing resolution.

Figure 4 shows the results which are obtained from comparison of the two methods for calculating the π -trapezoid method (classical MPI 18 PPU threads and optimized MPI-SDK unbalanced PPU 18 threads and 54 threads SPU). Are presented more values:

- $\pi \rightarrow 20 + \text{LOG}(\text{Error})$ obtained for the classical version of MPI.
- $\text{Err } 7.3 \rightarrow 20 + \text{LOG}(\text{Error})$ obtained for the optimized version of MPI-SDK unbalanced with a coefficient $k = 7.3$.

Fig. 6 The chart to determine the coefficient k and calculating the π

Fig. 7 Formula for π calculation using rectangle method

You can see an improvement of the calculation accuracy within the optimized MPI-SDK version unbalanced with a coefficient $k = 7.3$.

- $\pi \rightarrow \text{SQRT}(\text{Time(s)})$ obtained for the classical version of MPI.
- $\text{Time } 7.3 \rightarrow \text{SQRT}(\text{Time(s)})$ obtained for the optimized version of MPI-SDK unbalanced with a coefficient $k = 7.3$.

Comparing the two sets of values, we can observe an improvement of the calculation time (approximately 20 %) of the optimized version of MPI-SDK unbalanced with a coefficient $k = 7.3$, approximately 20 %. We can get the following conclusions:

- Using SPUs in the double precision calculation, greatly delayed his performance.
- Making a PPU-SPU unbalanced distribution brings an improvement in computing time, in comparison with the use of PPU units only.
- Distribution coefficient must be determined for a small number of calculation points (a few tenths of a second) and then used to calculate a high resolution (10^{12} —several hours).
- Distribution coefficient is linear with resolution increasing.

3.3 Calculation of the rectangle area method using MPI distribution

To check if the value determined for k is dependant on computing power in double precision units PPU-SPU and on the number of double precision operations performed on these units, we continue with the implementation of an algorithm for calculating the integral area by rectangles method (Fig. 7).

Distribute the two processes on each node (PPU). Each of these processes calculates an equally distributed part of the integral defined by block:

```
h = 1. / (double) n;
w = h / 2;
for (i = me + 1; i <= n; i += nprocs) {
x = (i-1) * h;
piece = piece +
4 / (1 + (x + w) * (x + w)) * h
}
```

Fig. 8 A comparison between values of time (a) and error (b) obtained by two methods: classical MPI 18 PPU threads and optimal unbalanced method MPI-SDK 18 PPU threads and 54 SPU threads (rectangle method)

The variable w is the half step increments. The following results are obtained after running the test program (method rectangles), with n between 10^5 and 10^{12} (Fig. 8—marked with circle): We notice an improvement in execution time and error calculation to MPI trapezoid method, due to fewer calculations. Like in trapezoid method, we want to use SPU cores in the π approximate calculation. The aim is to

Fig. 9 Comparison between values of time (a) and error (b) obtained by the two methods MPI: trapezoid method and the rectangle method

determine the distribution coefficient k to check if it depends on the complexity of mathematical calculation.

3.4 Rectangle area calculation method using MPI-SDK distribution

After running the program, we determine the optimal ratio $k = 9.3$ and n between 10^5 and 10^{12} (Fig. 8—marked with minus). Figure 8 show the results that are obtained by comparing the two methods used in calculating the π value—rectangle method (classical MPI 18 PPU threads and optimized MPI-SDK unbalanced PPU 18 threads and 54 threads SPU).

As the trapezoid method, we found an improvement in time calculation between MPI distribution only on PPUs and MPI-SDK unbalanced distribution between PPU and SPSs, about 10 %.

A comparison between the two calculation variants MPI trapezoid method and MPI rectangles method is shown in Fig. 9.

As we can be seen from Fig. 9, there is a significant improvement in computing time for the same distribution (MPI) between the two methods (trapezoidal-rectangle) because fewer math computations run in the second method.

Fig. 10 Comparison between values of time (a) and error (b) obtained by MPI-SDK unbalanced trapezoid method $k = 7.3$ and rectangle method $k = 9.3$

A comparison between MPI-SDK unbalanced trapezoid method $k = 7.3$ and MPI-SDK unbalanced rectangle method $k = 9.3$ is shown in Fig. 10.

As above, in Fig. 10 reveals a 25 % improvement in computing time for the same distribution (MPI-SDK-unbalanced) between the two methods (trapezoidal–rectangle). This is because fewer mathematical calculations run in the second method on the one hand and on the other hand, the modification factor k makes PPU units to take more calculations than the second method, minimizing the time calculation. To prove this, Fig. 11 shows a graph of time to calculate the rectangle method in MPI-SDK unbalanced distribution for $k = 7.3$ and $k = 9.3$.

4 Conclusions

By introducing the SPU units in calculating the value of π and using an unbalanced loading distribution between PPU-SPU units, we obtain the following values of cal-

Fig. 11 Comparison of computing time for the rectangle method in MPI-SDK unbalanced distribution for $k = 7.3$ and $k = 9.3$

ulation time and errors (Fig. 12). The follows conclusions are derived from analysis of the two graphs:

- The introduction of the SPU units into the calculation of (DP-Double Precision) improves calculation time by about 50 % but is well below of SPU units potential if they had a unit DP.
- Also, we get a lower error in calculating of the π value.
- To obtain maximum efficiency of computing time, you must use an unbalanced distribution formula calculations between SPU and PPU units.
- Distribution coefficient k takes different values depending on the number and complexity of the calculations.
- Preferably use SPU units in double precision calculations, but reserve their use to a lower level.

This algorithm can be used to increase the speed of double precision calculation in scientific computing.

5 Future works

In [1], the authors present a comparison of two phylogenetic trees inference codes, RAxML and PBPI. They present an experimental data by a comparison between PS3 cluster performance and QS20 dual-Cell/BE blades with the same total number of SPEs. They observed that the performance of the PS3 cluster is reasonably close to the performance of the QS20 cluster. We make the same comparison with our

Fig. 12 The gain in time (a) and error (b) obtained by MPI-SDK unbalanced $k = 9.3$ versus the classical MPI

algorithm and observed that k was very dependent by hardware architecture. For example, if $k = 1.1$, Time PS3 > Time QS22, for $k = 7.3$, Time PS3 < Time QS22. We want to use SIMD-ization and unrolling characteristics in the calculation made by SPU units. It is expected to increase about 4 times the speed of calculation. Amending the balance PPU-SPU upload units in that SPU units will process a greater number of computing intervals (Fig. 13).

As we can observe, the k coefficient was determined as 1.1 for the SIMD version. That means that PPU and SPU get approximatively the same loading.

Acknowledgements This paper was supported by the project “Progress and development through post-doctoral research and innovation in engineering and applied sciences—PriDE—Contract no. POS-DRU/89/1.5/S/57083”, project cofunded from European Social Fund through Sectoral Operational Program for Human Resources 2007–2013.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

(a)

(b)

Fig. 13 Comparison between values of time (a) and error (b) obtained by MPI-SDK unbalanced rectangle method $k = 9.3$ and SIMD rectangle method $k = 1.1$

References

1. Blagojevic F, Curtis-Maury M, Yeom J-S, Schneider S, Nikolopoulos DS (2008) Scheduling asymmetric parallelism on a PlayStation3 cluster. In: CCGRID '08 proceedings of the 2008 eighth IEEE international symposium on cluster computing and the grid, pp 146–153
2. Calandrino JM, Baumberger D, Li T, Hahn S, Anderson JH (2007) Soft real-time scheduling on performance asymmetric multicore platforms. In: Real time and embedded technology and applications symposium, pp 101–112
3. <http://www.ps3cluster.umassd.edu/>
4. IBM-Sony, Toshiba Corp. (2006–2008) Cell broadband engine programming handbook
5. Koranne S (2009) Practical computing on cell broadband engine. Springer Science, Dordrecht
6. Krishna M, Kumar A, Jayam N, Senthilkumar G, Baruah PK, Sharma R, Kapoor S, Srinivasan A (2007) A synchronous mode MPI implementation on the cell BE architecture. In: Parallel and distributed processing and applications, vol 4742. Springer, Berlin, pp 982–991
7. Li T, Brett P, Hohlt B, Knauerhase R, McElderry SD, Hahn S (2010) Operating system support for shared-ISA asymmetric multi-core architecture. In: Proceedings of the 16th IEEE international symposium on high-performance computer architecture, January 2010, pp 19–26
8. Scarpino M (2008) Programming the cell processor: for games, graphics, and computation, vol 744. Springer Science, Upper Saddle River
9. Velamati MK, Kumar A, Jayam N, Senthilkumar G, Baruah PK, Sharma R, Kapoor S, Srinivasan A (2007) Optimization of collective communication in Intra-Cell MPI. In: High performance computing—HiPC 2007, vol 4873. Springer, Berlin, pp 488–499