Feasibility Study of MPI Implementation on the Heterogeneous Multi-Core Cell BE™ Architecture

Arun Kumar
Sri Sathya Sai University
Prasanthi Nilayam, India
t.arunkumar@yahoo.com

Ganapathy Senthilkumar
Sri Sathya Sai University
Prasanthi Nilayam, India
rgskumar1983@gmail.com

Murali Krishna
Sri Sathya Sai University
Prasanthi Nilayam, India
kris.velamati@gmail.com

Naresh Jayam
Sri Sathya Sai University
Prasanthi Nilayam, India
nareshjs@gmail.com

Pallav K. Baruah
Sri Sathya Sai University
Prasanthi Nilayam, India
baruahpk@gmail.com

Raghunath Sarma
Sri Sathya Sai University
Prasanthi Nilayam, India
raghunath.sarma@gmail.com

Ashok Srinivasan
Florida State University
Tallahassee, Florida, USA
asriniva@cs.fsu.edu

Shakti Kapoor
IBM, Austin
Texas, USA
skapoor@us.ibm.com

ABSTRACT

The Cell Broadband Engine™ is a new heterogeneous multi-core processor from IBM, Sony, and Toshiba. It contains eight co-processors, called Synergistic Processing Elements (SPEs), which operate directly on distinct 256 KB local stores, and also have access to a shared 512 MB to 2 GB main memory. The combined peak speed of the SPEs is 204.8 Gflop/s in single precision and 14.64 Gflop/s in double precision. There is, therefore, much interest in using the Cell BE™ for high performance computing applications. However, the unconventional architecture of the SPEs, in particular their local stores, creates some programming challenges. We describe our implementation of certain core features of MPI, such as blocking point-to-point calls and collective communication calls, which can help meet these challenges, by enabling a large class of MPI applications to be ported to the Cell BE™ processor. This implementation views each SPE as a node for an MPI process. We store the application data in main memory in order to avoid being limited by the local store size. The local store is abstracted in the library and thus hidden from the application with respect to MPI calls. We have achieved bandwidth up to 6.01 GB/s and latency as low as 0.41 μs on the ping-pong test. The contribution of this work lies in (i) demonstrating that the Cell BE™ has good potential for running intra-Cell BE™ MPI applications, (ii) enabling such applications to be ported to the Cell BE™ with minimal effort, and (iii) evaluating the performance impact of different design choices.

Categories and Subject Descriptors

D.1.3 [**Programming Techniques**:] Concurrent Programming – Parallel programming.

General Terms

Performance, Design, Experimentation.

SPAA'07, June 9–11, 2007, San Diego, California, USA.
ACM 978-1-59593-667-7/07/0006.

Keywords

Cell BE™ processor, heterogeneous multi-core processors, MPI.

1. INTRODUCTION

Even though the Cell BE™ was aimed at the Sony PLAYSTATION®3, there is much interest in using it for High Performance Computing, due to the high flop rates it provides. Preliminary studies have demonstrated its effectiveness for important computational kernels [1]. However, a major drawback of the Cell BE™ is its unconventional programming model; applications do need significant changes to exploit the novel architecture. Since there exists a large code base of MPI applications, and much programming expertise in it in the high performance computing community, our solution to the programming problem is to provide an MPI 1.1 implementation that uses each SPE as if it were a node for an MPI process.

Existing MPI implementations for the shared memory architectures cannot be directly ported to the Cell BE™, because the SPEs have somewhat limited functionality. For example, they cannot directly operate on the main memory – they need to explicitly DMA the required data to local store and then use it. In fact, they cannot even dynamically allocate space in main memory.

In order for an MPI application to be ported to the Cell BE™ processor, we need to deal with the small local store sizes on the SPEs. If the application data is large, then the local store needs to be used as a software-controlled cache and data-on-demand, with the actual data in main memory. If the application code size is large, then a code overlaying technique can be used to bring in the required functions to the local store as needed, keeping the rest in the main memory. These features are available in the recent release of the Cell BE™ SDK 2.0. These will allow MPI applications to be ported to the Cell BE™ with minimal effort, in combination with our MPI implementation. We ported some application kernels before SDK 2.0 was released, and hand-coded the transformations for keeping their data in main memory. We also have a version of our implementation (which we do not

discuss further here) that works without the above transformations for small memory applications.

Empirical results show that our implementation achieves good performance, with throughput as high as *6.01* GBytes/s and latency as low as *0.41* µs on the pingpong test. We expect the impact of this work to be broader than just for the Cell BE™ processor due to the promise of heterogeneous multi-core processors in the future, which will likely consist of large numbers of simple cores as on the Cell BE™.

2. BLOCKING POINT-TO-POINT COMMUNICATION

We have implemented blocking point to point communication calls using four different designs, and have implemented collective communication calls on top of these calls. In the first design, the PPE is involved in buffer management and synchronization. The PPE spawns one thread to serve each SPE, and the SPEs communicate with those threads to handle the required functionalities. In the remaining designs, the PPE is involved only in initialization and termination – all buffer management and synchronization are performed by the SPEs. In the second design, we implemented the calls in synchronous mode, avoiding the buffer management issue. In the third design, we implemented buffered mode communication. In the fourth design, we used the buffered mode for small messages and the synchronous mode for large messages.

In the first design, the involvement of the PPE turns out to be a bottleneck, and performance is worse for all message sizes, compared with subsequent designs. However, since much of functionality is borne by the PPE, the footprint of the library in the SPE is small compared with that of the other designs. The second design has better large-message latency than the third one, but a slightly higher (0.65 µs versus 0.41 µs) 0-byte latency than the third one. The fourth design inherits the good small-message latency of the third design, and the good large-message performance of the second design. The performance numbers quoted for our MPI implementation are those obtained with this fourth design. Further details are given in [2].

3. PERFORMANCE ANALYSIS

We performed our experiments on a 3.2 GHz Rev 2 Cell BE™ blade with 1 GB main memory running Linux 2.6.16 at IBM, Rochester. We had dedicated access to the machine while running our tests. Table 1 compares the latencies and bandwidths of our results with those of some good intra-node implementations on other hardware. We can see that MPI on the Cell BE™ has performance comparable to those on processors with full-fledged cores.

In table 1, *Cell BE™* refers to our implementation on the *Cell BE™* processor. *Nemesis* refers to MPICH using the Nemesis communication subsystem. *Shm* refers to MPICH using the shared-memory (shm) channel. *Xeon* refer to timings on a dual-SMP 2 GHz Xeon, reported in [3]. *Opteron* refers to timings on a 2 GHz dual-core Opteron, reported in [4].

In our preliminary work, we have demonstrated the use of our MPI implementation on large memory applications by transforming two applications, double precision matrix multiplication and matrix-vector multiplication, to store data in main memory. These applications were not optimized for best performance. Space in main memory is allocated by the PPE during initialization, in a fairly automatic manner. But the application code had to be changed while accessing the data, to perform DMA reads and writes.

We achieved a double precision flop rate of 5.66 Gflop/s in matrix multiplication, and 7.8 Gflop/s in matrix-vector multiplication, for matrices of dimension 1024. In contrast, a 2 GHz single-core Opteron processor yielded 40 Mflop/s and 292 Mflop/s respectively, with the same (inefficient) algorithms.

Table 1. Performance Comparisons

MPI/Platform	Latency (0 Byte)	Maximum throughput
Cell BE™	0.41 µs	6.01 GB/s
Cell BE™ Congested	NA	4.48 GB/s
Nemesis/Xeon	≈ 1.0 µs	≈ 0.65 GB/s
Shm/Xeon	≈ 1.3 µs	≈ 0.5 GB/s
Nemesis/Opteron	≈ 0.34 µs	≈ 1.5 GB/s
OpenMPI/Opteron	≈ 0.6 µs	≈ 1.0 GB/s

4. FUTURE WORK

We are in the process of implementing non-blocking point to point communication calls. We are also implementing optimized collective communication calls. We also intend studying the latencies introduced due to the use of overlays and software managed cache, and plan to evaluate larger applications and optimized kernels.

5. REFERENCES

[1] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick, The *Potential of the Cell BE Processor for Scientific Computing*, Proceedings of the ACM International Conference on Computing Frontiers, 2006.

[2] M. Krishna, A. Kumar, N. Jayam, G. Senthilkumar, P.K. Baruah, R. Sharma, A. Srinivasan, and S. Kapoor,. *A Buffered Mode MPI Implementation for the Cell BE™ Processor*, to appear in Proceedings of the International Conference on Computational Science (ICCS), Lecture Notes in Computer Science, (2007). See also: http://www.cs.fsu.edu/research/reports/TR-061215.pdf, for details on the synchronous mode.

[3] D. Buntinas, G. Mercier and W. Gropp, *Design and Evaluation of Nemesis, a Scalable, Low-Latency, Message-Passing Communication Subsystem*, Proceedings of the International Symposium on Cluster Computing and the Grid, 2006.

[4] D. Buntinas, G. Mercier and W. Gropp, *Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem*, Proceedings of the Euro PVM/MPI Conference, 2006.