Exploration of the Potential of Cell Architecture For MPI Applications

Arun Kumar[1], Naresh Jayam[1], Ganapathy Senthilkumar[1], Murali Krishna[1], Pallav K Baruah[1], Ashok Srinivasan[3], Shakti Kapoor[2], Raghunath Sharma[1]

[1]Dept. of Mathematics and Computer Science, Sri Sathya Sai Institute of Higher Learning, Prashanthi Nilayam, India.

[2]IBM, Austin.

[3]Dept. of Computer Science, Florida State University.

Abstract

The Cell Broadband Engine Architecture (CBEA) designed by Sony, Toshiba and IBM is a heterogeneous multi-core processor. Though aimed at the gaming industry the Cell processor has a good potential for scientific computation. Our work is aimed at analyzing the potential of the Cell processor for running MPI (Message Passing Interface) applications. Our work involves the implementation of the basic calls of the Message Passing Interface, study of the bandwidth and latency of the library. Also we have studied different models for the implementation of MPI on the Cell processor. Our initial experiments have shown encouraging results.

1. Introduction

The introduction of multi-core processors and simultaneous multi-threaded (SMT) processors has opened up the possibility of parallel computation on a single chip. The Cell Broadband Engine Architecture developed by Sony, Toshiba and IBM is a move towards providing supercomputing power on a single chip through a heterogeneous multi-core processor design. This new design with eight computation-intensive cores, each having its own private high speed memory encourages the idea of highly parallel applications.

We explore different approaches for implementing a Message Passing Interface library that would enable the large number of MPI applications that are found in the scientific community to run on the Cell processor. The Cell processor has one 64-bit SMT PowerPC core, the PPE (PowerPC Processor Element). It acts as the controller for the eight computationally intensive SIMD cores called as the Synergistic Processor Elements (SPEs). Each of the SPEs has a private memory of size 256KB called the Local Store (LS). The load and store instructions of the SPE act on the data from its own local store. Each SPE has an associated DMA engine that is used for moving data between the main memory and the local store, allowing for overlapping of computation and communication. The SPEs can communicate among themselves and also with the PPE through mailboxes and signal notification registers. Our work comprises of implementing a minimal MPI library that would enable MPI applications to run on the Cell processor with each SPE being considered as one MPI node.

2. Related work

The coming of a heterogeneous multi-core processor like the Cell processor which has enormous computational power has caught the attention of the scientific community. Work has been done to port a number of computational kernels like DGEMM, SGEMM, 1D FFT and 2D FFT to the Cell processor [1]. Results of the work have shown the potential of the Cell processor for scientific computation [1]. Some amount of work has been done in the direction of developing frameworks that would enable the programming of Cell at an abstract level [2, 3, and 4]. Comprehensive work has been done in the past on the implementation of MPI library on shared memory machines [5]. Since the local store of each of the Synergistic Processor Elements is too small for most of the applications, research on developing compiler and runtime support for running programs with data and text sizes larger than the size of the local store is being carried out [6 , 7]. However, to date, an attempt to study the potential of Cell for supporting the large number of existing MPI applications through the implementation of an MPI library for Cell processor does not appear in the literature.

2. Design and Architecture of MPI on Cell

We have tried three different approaches for implementing the MPI library on the Cell processor. In the first model each MPI node has two components: one SPE thread that runs on the SPE core and a pthread which runs on the PPE and serves the SPE thread with some of the functionality. In the second model the PPE is involved only during the initialization and finalization phase of the MPI application. All the computation and communication functionalities run on the SPE. The third model is an extension to the second model to support applications with large data sets and is implemented in the 'synchronous' mode. Section 2.1 describes the basic communication architecture. Sections 2.2, 2.3 and 2.4 describe the first, second and third models respectively.

2.1 Communication Architecture

Let N be the number of MPI nodes. Our runtime environment allocates N*(N-1) buffers, since each node can send messages to N-1 nodes (a process is not allowed to send messages to itself in standard mode). Each buffer is designated to a sender-receiver pair P_i and P_j. The buffer corresponding to sender as P_i and receiver as P_j is different from the buffer corresponding to P_j as sender and P_i as receiver. All the buffers corresponding to a sender are managed by that sender. Figure 1 shows the communication architecture in detail. The organization of the message buffers and the structure of the meta-data entry are shown.

The sending and receiving processes communicate information about the location, tag, and size of the messages through meta-data stored in buffers corresponding to each pair. There are N*(N – 1) buffers used for storing the meta-data corresponding to each sender- receiver pair. The buffers containing the meta-data are organized as an array of entries. The meta-data buffer is managed in a manner similar to message queues in the sense that a new meta-data entry cannot overtake a previous meta-data entry and is also searched in a linear fashion to ensure message ordering. An entry is made by a sender after copying the message into the message buffer. The entry contains the information about the location of the message within the buffer, the tag associated with it, the data type and size of the message and flag bits.

The location of the buffers and the functionalities of the PPE and SPEs vary between the different models. In the first model the management of the buffer memory is done by the PPE at the request of the SPE, whereas in the second and third models, it is done by the SPEs. Also, the number of buffers is N*N, in the third model.

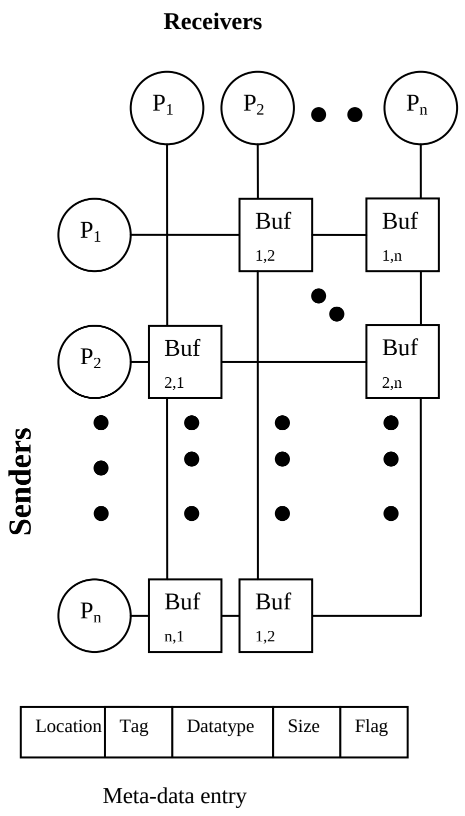

Figure 1: Communication Architecture

2.2 SPE-Centric Single Program Multiple Data model (Design 1)

Each MPI node is composed of two components, one running on the SPE and a pthread running on the PPE. The point to point communication calls are implemented in buffered mode. The application code runs on the SPE and the pthread corresponding to each SPE acts as a server performing certain functionalities related to management of the message buffers and synchronization between two SPEs. The SPEs perform all of the computation and make requests to the pthread through mailbox messages.

In this model buffers used by the point to point communication operations are located in the SPE local store, since the SPE to SPE DMA transfers are faster than SPE to main memory DMA transfers. The send and receive operations are as described in the section 2.1. This model has the advantage of high speed message transfers since on-chip DMAs are faster. But the mailbox communication between the SPE and PPE is slow and the functionalities provided by the PPE are also slow, bringing down the overall performance.

2.3 A Fully SPE-Centric Model with Application Data in Local Store (Design 2)

In this model the whole of MPI library runs on the SPE. All the functionalities of buffer management and synchronization provided by the pthreads in the model described in section 2.2 are moved in to the SPE. Each MPI node is assigned to one SPE. The send and receive operations are as described in the section 2.1.In this model the whole of application data and code is in the SPE local store. The message buffers are moved to the main memory to give way for the application data.

This model performed better than the model described in section 2.2. But since the code and data must fit in the SPE local store which is only 256 KB we are restricted to a small number of applications.

2.4 A Fully SPE-Centric Model with Application Data in Main Memory (Design 3)

This model is an extension to the model described in the section 2.3. In order to overcome the restriction on the size of the application data owing to the small size of the SPE local store we moved the application data to the main memory. In this model a PPE process is spawned which then creates the SPE threads. All the application code runs in the SPE, but the application data is moved to the main memory area of the PPE program. The data is moved between the main memory and the local store as and when needed for processing.

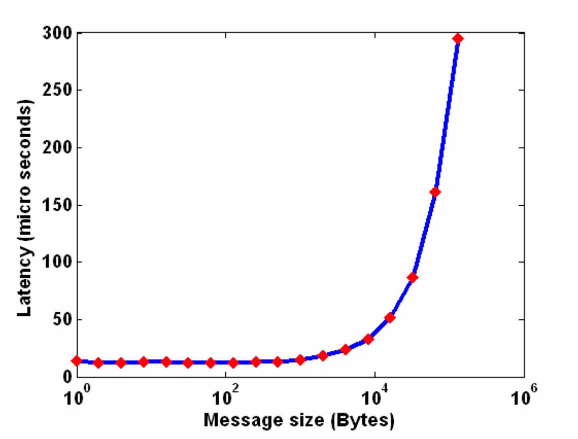

Figure 2 Latency results for point-to-point communication for Design 2 (data in local store).

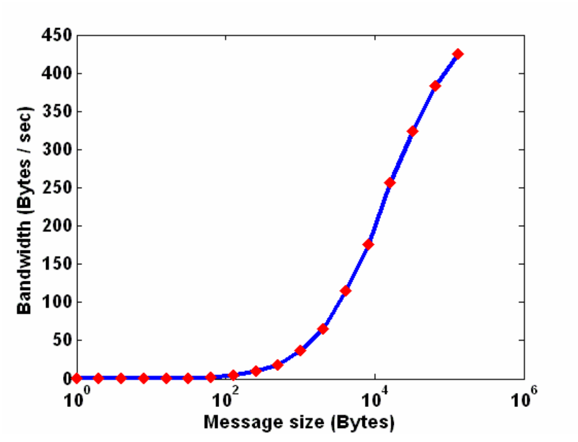

Figure 3 Bandwidth results for point-to-point communication for Design 2 (data in local store).

Send and receive operations copy messages from main memory to main memory. They are implemented in a synchronous manner to eliminate the necessity of buffering for large messages. This approach has removed the memory restrictions on the size of the application data. The best results were obtained using this model.

3. Experimental Results

The main purpose of our experiments was to study the feasibility of MPI on Cell. We evaluated the bandwidth and latency results for the point to point calls. All our results were obtained by running on a Cell blade running at 3.2 GHz.

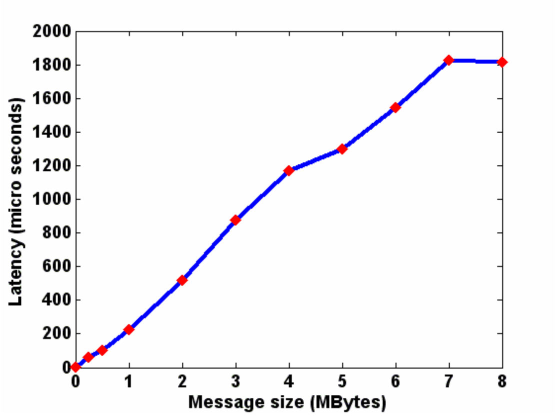

Figure 4 Latency results for point-to-point communication for Design 3 (data in main memory).

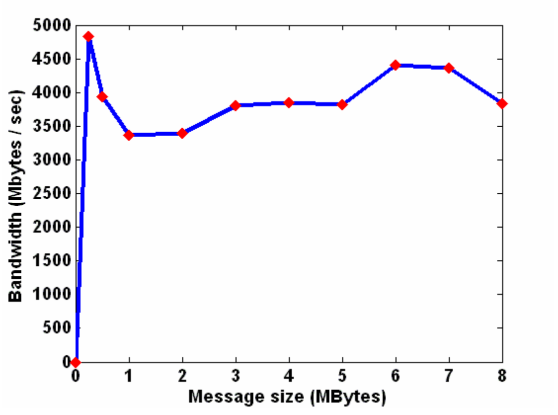

Figure 5 Bandwidth results for point-to-point communication Design 3 (data in main memory).

Figure 2 shows the latency results and Figure 3 shows the bandwidth results for point-to-point communication primitives for the implementation where the data and code fit into the local store. Figure 4 shows the latency results and Figure 5 shows the bandwidth results for point-to-point communication primitives for the implementation where data is in the main memory. The latency is substantially lower than for the case where data is in local store with the best value being 1.6 microseconds.

The bandwidth results are also better for this version on the hardware with a best value of 4.8 GB/s.

4. Conclusions and Future Work

We have presented different approaches to the design and implementation of core features of MPI on the Cell Architecture. We have studied the bandwidth and latency results for the point to point communication calls. Our results are very encouraging and support the idea of running MPI applications on the Cell processor.

In future we intend to work on overcoming the limitations on the application code size using code overlaying. We also intend to implement other features of MPI on the Cell Architecture.

Acknowledgment

We express our gratitude to Sri Sathya Sai Baba for bringing us all together to perform this work, and for inspiring and helping us toward our goal. Our sincere thanks to IBM Corporation, for giving us the opportunity to learn Cell Architecture and Programming through a summer project.

References

[1] Samuel Williams, John Shalf, Leonid Oliker Shoaib Kamil, Parry Husbands, Katherine Yelick, The Potential of the Cell Processor for Scientific Computing *CF'06*, May 3–5, 2006, Ischia, Italy.

[2] M. Ohara, H. Inoue, Y. Sohda, H.Komatsu, T. Nakatani MPI microtask for programming the Cell Broadband Engine^{TM} processor IBM SYSTEMS JOURNAL, VOL 45, NO 1, 2006.

[3] Kayvon Fatahalian, Timothy J. Knight, Mike Houston, Mattan Erez, Sequoia: Programming the Memory Hierarchy SC2006 November 2006, Tampa, Florida, USA 2006 IEEE

[4] MultiCore Framework, Harnessing the Performance of the Cell BE™ Processor 2006 Mercury Computer Systems, Inc.

[5] Hong Tang, Kai Shen, Tao Yang., Program Transformation and Runtime Support for Threaded MPI Execution on Shared-Memory Machines, ACM Transactions on Programming Languages and Systems, Vol. 22, No. 4, July 2000, Pages 673-700.

[6] An introduction to compiling for the Cell Broadband Engine architecture, Part 4: partitioning large tasks Feb 2006 http://www-128.ibm.com/developerworks/edu/pa-dw-pa-cbecompile4-i.html

[7] An introduction to compiling for the Cell Broadband Engine architecture, Part 5: Managing memory. Feb 2006 http://www-128.ibm.com/developerworks/edu/pa-dw-pa-cbecompile5-i.html

[8] T. Chen, R. Raghavan, J. Dale, E. Iwata. *Cell Broadband Engine Architecture and its first implementation.* November 2005.