

DDM-Cell: Data-Driven Multithreading on the Cell Processor

Samer Arandi, Kyriakos Stavrou,
Pedro Trancoso, Paraskevas Evripidou

Department of Computer Science, University of Cyprus,
75 Kallipoleos Ave., P.O.Box 20537 - 1678 Nicosia, Cyprus
E-mail: {samer,tsik,pedro,skevos}@cs.ucy.ac.cy

ABSTRACT

Cell Broadband Engine is a new heterogeneous multi-core chip processor designed to provide a high computational power on a single chip. The advent of multi-core chips posed a challenge for a suitable programming and execution model that exploits this new technology. This challenge is greater with Cell, because of the different properties it has compared to other multi-core systems.

In this paper we provide an overview of a work-in-progress in which we argue that Data-Driven Multithreading (DDM), a non-blocking multithreading model that decouples computation and synchronization and allows them to execute asynchronously, is a suitable model to be used with Cell. We justify this by illustrating that both DDM and Cell are compatible on many aspects, as our preliminary analysis has shown, especially in the approaches they both utilize to achieve better performance. Finally, we provide an overview of our preliminary implementation of the DDM model on the Cell processor, which we call DDM-Cell.

KEYWORDS: Data-Driven Multithreading; Chip Multiprocessors; Cell Processor; Multi-core

1 Introduction

1.1 The Cell Processor

Cell Broadband Engine processor (Cell[3]) is a heterogeneous multi-core chip jointly developed by IBM, Sony and Toshiba. It is composed of one general-purpose RISC processor called the Power Processor Element (PPE), and 8 fully-functional independent SIMD co-processors called the Synergistic Processor Elements (SPE), high-speed memory and I/O interface controllers, all communicating through a high-speed ring bus called the Element Interconnect Bus (EIB) (see Figure 1).

To overcome the effects of the power and memory walls, Cell adopts a relatively high frequency and power efficient design, in addition to a number of design decisions regarding

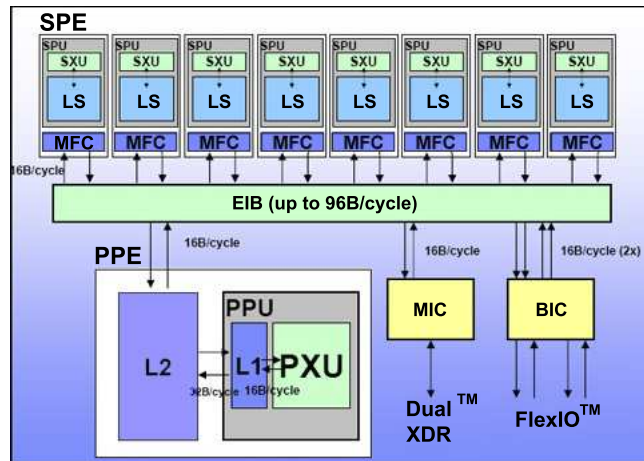


Figure 1: Cell Processor Architecture (Image from IBM).

the PPE and SPE processors, most notably the specialization between control and execution, the asynchronous coherent DMA engines, inside the Memory Flow Controller (MFC) units, that allow the SPE to interleave memory transfer and computation, as well as the SPE's large and wide register file and a 256 KByte software controlled local store memory (LS).

1.2 The DDM Model

Among the models suggested for supporting execution on multi-core architectures is the Data-Driven Multithreading (DDM)[1]. This model depends on the Data-Flow model, where an instruction is scheduled for execution when all the data it needs are available. The DDM model applies the same principle but on a larger granularity. Instead of scheduling a single instruction, it schedules one thread (a group of instructions) when all its input data have been produced and placed in the processor's local memory. Thus, no synchronization nor communication latencies are experienced once the execution of the thread starts. Furthermore, this model decouples the synchronization and execution parts of a program, which allows it to tolerate synchronization and communication delays by allowing computation to produce useful work while a long latency event is in progress. All synchronization tasks are handled by a modular hardware unit called the Thread Scheduling Unit (TSU)[1].

2 DDM-Cell

Cell, like all multi-processor systems, requires employing a suitable concurrent programming and execution model to facilitate an efficient usage of its resources. This model must also accommodate the unique properties of the Cell processor, like the heterogeneity and the relatively high number of the cores, the software controlled local memory, the DMA units, and the nature of the communication mechanisms. We believe that DDM is such a model.

2.1 DDM and Cell Mapping

A careful examination of both the DDM model and the Cell processor reveals an interesting match on many aspects between the two. First, the specialization between execution

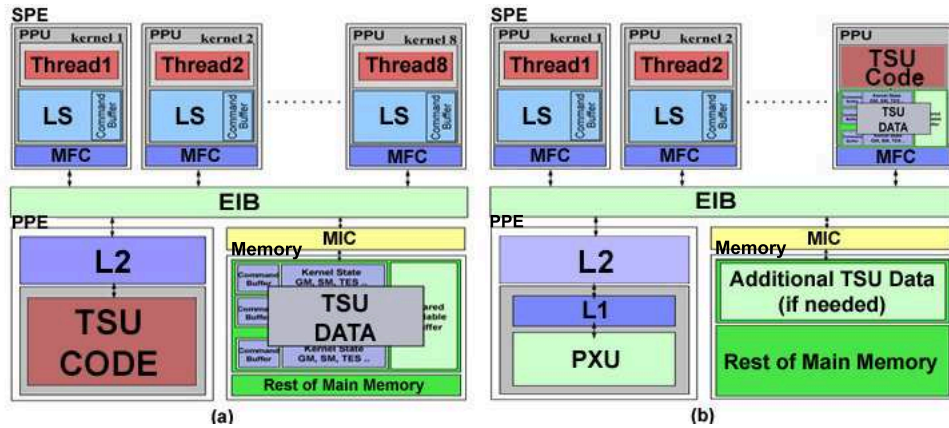


Figure 2: (a)PPE-TSU configuration (b) SPE-TSU configuration.

and control tasks on Cell, where the PPE handles the former and the SPE handles the latter. This resembles the decoupling between the execution and synchronization the DDM model adopts. Furthermore, in DDM the scheduling of threads is done internally without the intervention of the O.S. (except on startup) incurring a significantly smaller context switching overhead, compared to the overhead of saving the state of each SPE (more than 258KByte) if scheduling of multi-tasking threads is done at O.S. level in Cell.

Another factor is that both DDM and Cell can achieve deterministic execution once the data a thread requires, resides in the local store memory of the SPE where the thread is executing. A final factor is that the software controlled local store memory allows DDM the flexibility it needs to apply its CacheFlow policy[2], taking advantage of data prefetching to the maximum potential, compared to the limited control the hardware cache allows.

2.2 DDM-Cell Implementation Overview

To implement the DDM model on Cell (or any other processor), we need to consider carefully how we are going to implement each of the fundamental aspects of DDM, namely, thread scheduling, execution and communication mechanism used for synchronization and data transfer.

2.2.1 Thread Scheduling and Execution

Basically, the DDM threads will be executing on the SPE cores, while the TSU unit will be emulated using software running on one of the cores. The core assigned to execute the TSU software can be the PPE (PPE-TSU configuration) or any of the SPEs (SPE-TSU configuration), see Figure 2. Both configurations are possible, each with different properties. In our current implementation we have adopted the PPE-TSU configuration. This configuration is a better utilization of resources since the PPE is already used for controlling execution in Cell, which leaves the rest of the cores free for executing the computational loads. However, in this configuration, the TSU code will be sharing execution and memory resources with other non-DDM programs on the system which might reduce the responsiveness of the TSU. At a later stage the SPE-TSU configuration will also be implemented to evaluate it and compare it to the PPE-TSU configuration.

2.2.2 Communication Mechanisms

During the execution of a DDM application two types of communications are needed: synchronization and data-transfer. Cell provides a number of communication mechanisms, between the cores, the cores and memory or I/O. Such mechanisms include mailboxes and notification signals for small granularity communication (1 to 4 32-bit per message) and DMA transfers for large granularity communication (128 byte to 32MByte per message).

2.2.3 Synchronization Communication:

This involves issuing commands from the DDM thread to the TSU relating synchronization information (e.g. when a thread finishes execution) or reading synchronization information from the TSU (e.g. ID of a thread that will execute next). A simple command exchange system is implemented using DMA transfers of command buffers between the SPE and the TSU core. When the granularity of communication is small enough, mailbox mechanism is used for maximum efficiency.

2.2.4 Data-transfer Communication

In the DDM model, data communication is needed in two cases, first for handling shared variables, and second and most important for handling values that a DDM thread consumes or produces. Before a thread starts, all the data values it needs must be present in the local store memory where it executes. This requires that these values are fetched from the local stores of the producer threads or from the main memory if they were initialized there (as in the case of the shared variables). In our implementation this is done using the DMA transfer mechanism that Cell provides.

3 Conclusion and Future Work

In this report we have presented an overview of the implementation of our work-in-progress DDM-Cell model, explaining the rationale of our work and describing the preliminary structure of our model. At present we have a working prototype of our DDM-Cell model that we are evaluating. On a later stage, we will investigate other configurations and possible optimizations.

References

- [1] Kyriacou, C., Evripidou, P., Trancoso, P.: Data-Driven Multithreading Using Conventional Microprocessors. *IEEE Transactions on Parallel and Distributed Systems* (2005).
- [2] Kyriacou, C., Evripidou, P., Trancoso, P.: CacheFlow: A Short-Term Optimal Cache Management Policy for Data Driven Multithreading. (In: *Proceedings of the 2004 EuroPar*).
- [3] J. A. Kahle, et al. Introduction to the Cell Multiprocessor. *IBM Journal of Research and Development*, issue 49-4/5, 2005.