Implementing DTA support in CellSim

Roberto Giorgi[1,3], Nikola Puzovic[2,3], Zdravko Popovic[2,3]

Dept of Information Engineering, University of Siena, Via Roma 56, 53100 Siena, Italy

ABSTRACT

Decoupled Threaded Architecture (DTA) is designed to exploit Thread Level Parallelism (TLP) by using many simple cores grouped into a cluster for providing a scalable solution that copes with wire delay. Cell Broadband Engine (CBE) is a multiprocessor on a chip developed by Sony, Toshiba and IBM that contains one general purpose core and eight coprocessor elements that accelerate the multimedia and vector processing.
Here we illustrate the work that has been performed in implementing DTA support in the Cell processor using CellSim and we present the initial results that we have obtained.

KEYWORDS: multiprocessor, thread level parallelism, multithreading

1 Introduction

The Decoupled Threaded Architecture (DTA) [1] is based on SDF execution paradigm [2, 3]. DTA addresses scalability by a hierarchical structure of the nodes and a distributed scheduler. The architecture is divided into clusters, such that each element of the cluster can be reached within one cycle. Each thread that runs on DTA has a portion of local memory (called frame) associated to it, where data that are needed for the execution are kept. Only when all data that are needed for execution have arrived to the frame, thread will execute. Since frames are located near to the processor, accesses to frame memory should have very low latency, and will not cause any misses. Hence, the pipeline will not stall because of frame memory accesses. Each cluster in DTA contains a Distributed Scheduling Element (DSE) that is responsible for allocating tasks for processors inside the cluster and for maintaining balanced workload on each of them. Local Scheduling Element (LSE) is located inside each processor, and it is responsible for managing frames and execution of threads in the processor.
Cell Broadband Engine Architecture (CBEA) [4] combines one Power Architecture core with multiple SIMD processors called Synergetic Processing Elements (SPE). The current implementation has eight SPEs, which are interconnected by circular ring with four channels that is called Element Interconnect Bus (EIB). Each SPE in the architecture has a local storage that does not participate in cache coherency, and PPE has L1 and L2 caches. A main memory can be attached trough the Memory Interface Controller (MIC).

---

[1] HiPEAC Member.
[2] HiPEAC PhD Student.
[3] http://www.dii.unisi.it/~{giorgi,popovic,puzovic}

So far, the simulator that we used in order to test the DTA was using DTA-specific instruction set, and because of limited availability of benchmarks we have decided to implement a DTA support in a simulator for one of the existing and widely used architectures. Doing this would also allow us to focus only on DTA-related research because we would be able to utilize components that exist in other simulators (such as a sophisticated memory system, interconnection, etc). Since CBE has gained a lot of attention recently, and it has shown excellent performance, we have decided to use Cell processor as the starting point for implementing the DTA support. The DTA threads would execute in this case on SPEs, and PPE would be responsible for managing workload and for dispatching threads.

2 Mapping DTA to Cell

In order to execute the DTA programs, Cell SPE pipelines are used together with additional logic for managing thread creation and distribution. A tool has been developed for translating existing DTA benchmarks for the SPE architecture. In order for the benchmarks to run, few new instructions have been added that will support the creation and management of the DTA threads, and that will be used to manage loads and stores to the frame memory.

The system is developed starting from CellSim [5], which is a modular simulator developed in the UNISIM environment that is intended to simulate the Cell processor. Figure 1 shows the mapping of DTA architecture to Cell.

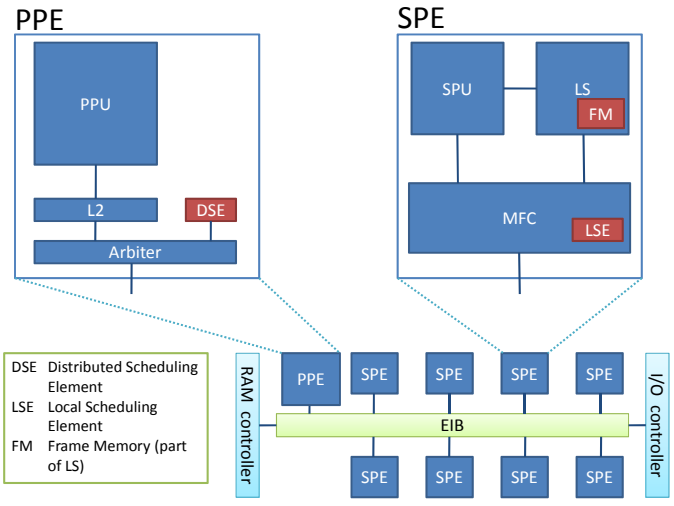

Figure 1. Mapping DTA to Cell

When performing this mapping, we have considered that one PPE and eight SPEs that are attached to it are making one cluster. Distributed Scheduling Element (DSE) is added alongside each PPU, and it is in charge of distributing workload among different SPEs. Inside each SPE, we have added a Local Scheduling Element (LSE) that is in charge of threads that are running on the SPE and a certain amount of LS is dedicated to hold frames for DTA threads. Figure 2 on the left presents a detailed organization of the SPE.

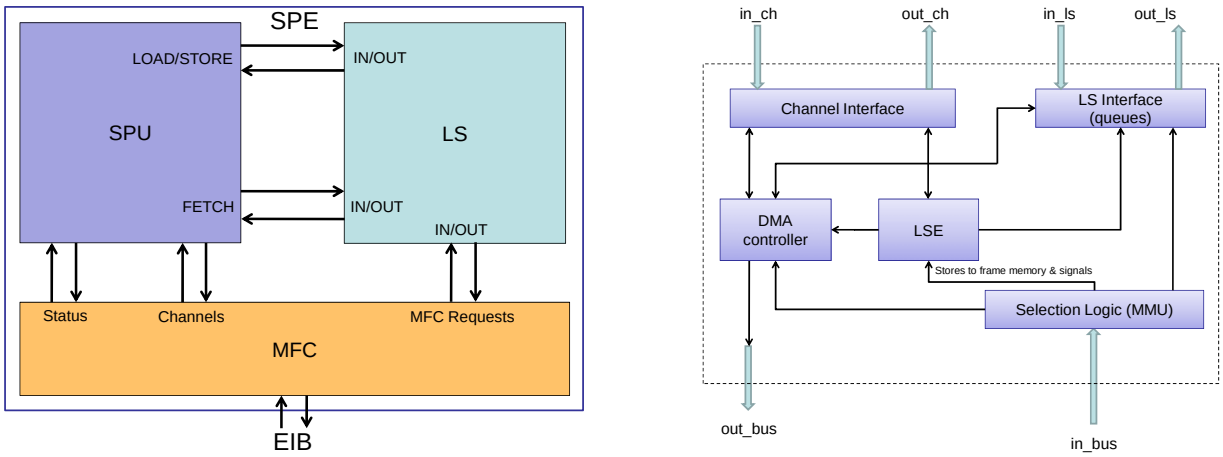

Figure 2 Detailed SPE (on the left) and MFC (on the right) organization

SPE is implemented using three modules: SPU that is the actual processor, LS that represents a Local Store and the MFC (Memory Flow Controller) module that is responsible for running DMA operations, and allows the PPE to control the SPE. Figure 2 on the right presents the organization of the MFC with the modifications needed to run DTA threads. Local storage has remained unchanged, and one part of it has been reserved in order to hold the frames. LSE is a part of MFC, and it uses the capabilities of MFC for communication with the rest of the system. Through the channel interface, SPE can access the LSE, and LSE can send data and commands to SPE. By using the LS interface (queues), LSE can write to frame memory.

CellSim provides with the mapping of the memory in which all local stores, main memory and control registers for PPEs and SPEs are mapped inside a single address space. In this mapping, the local store of each SPU has a predefined range of addresses and we are using part of this range for accessing frames. We have used free locations in the mapping of control registers in order to map control registers for local and distributed schedulers.

3 Initial Results

In order to perform the initial tests we have use the simplest available benchmark that calculates Fibonacci numbers. Figure 3 shows the results that were obtained. We can see that Cell with DTA support (DTA on Cell) scales well, even better than basic DTA, but execution time for basic DTA decreases not as fast as for Cell with DTA support when the number of SPE increases.

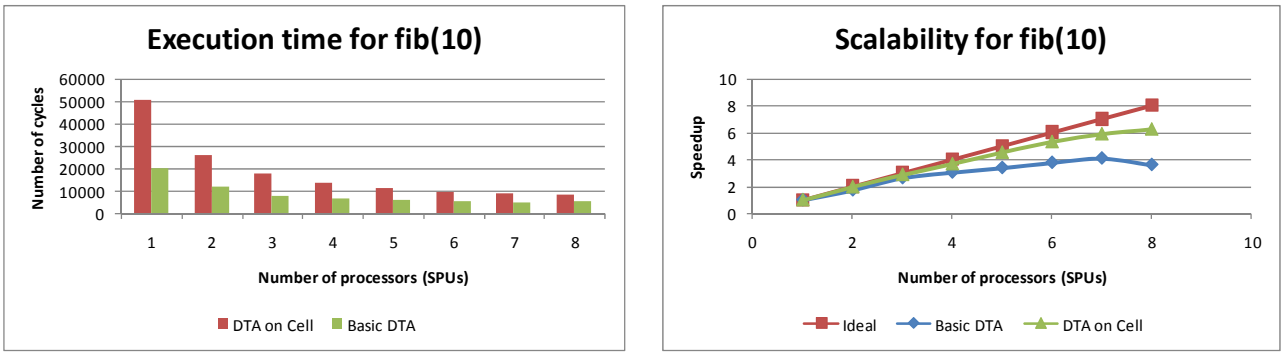

Figure 3. Comparison of "DTA on Cell" execution and basic DTA execution

The basic DTA model that was used for obtaining results uses perfect memory (no latencies) and a bus as interconnection among processors. This explains why the execution time with eight processors is worse than the execution time with seven processors. Since the network gets congested when number of processors is increased from seven to eight, the average latency of the messages that are exchanged among processors is higher, and total execution time gets longer. When executing DTA on Cell, the memory model is realistic and the SPEs are modelled with more detail then the pipelines of DTA (such as pipeline hazards). This yields the higher execution time compared to basic DTA. This also leads to the fact that messages are sent with bigger distance in time between them (with respect to basic DTA version), and message latency does not saturate the scalability when executing DTA programs on Cell.

4 Conclusions and future work

Here we have presented the work performed so far in implementing the DTA support in the Cell processor. The initial results that we have obtained show that we are on the right track for achieving our goal, which is implementing a scalable and efficient version of the DTA architecture. Future work will focus on further development of the simulator which will eventually lead to fully non-blocking execution of threads, and to provide mechanisms for decoupling memory accesses from execution.

5 Acknowledgments

This work was supported by the European Commission in the context of the SARC integrated project #27648 (FP6).

6 References

[1] R. Giorgi, Z. Popovic, and N. Puzovic, "DTA-C : A Decoupled multi-Threaded Architecture for CMP Systems," in *19th International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2007* Gramado, Brasil, 2007, pp. 263-270.

[2] K. M. Kavi, R. Giorgi, and J. Arul, "Scheduled Dataflow: Execution Paradigm, Architecture, and Performance Evaluation," *IEEE Transaction on Computers,* vol. 50, pp. 834-846, August 2001.

[3] K. Kavi, J. Arul, and R. Giorgi, "Execution and Cache Performance of the Scheduled Dataflow Architecture," *SPRINGER Journal of Universal Computer Science,* vol. 6, pp. 948-967, October 2000.

[4] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," *IBM J. Res. Dev,* vol. 49, pp. 589-604, 2005.

[5] F. Cabarcas, A. Rico, D. Rodenas, X. Martorell, A. Ramirez, and E. Ayguade, "CellSim: A Cell Processor Simulation Infrastructure," in *HiPEAC ACACES-2007*, L'Aquila, Italy, 2007, pp. 279-282.