Benchmarking the Memory Interface Controller bus of the Cell processor

Nathalie Casati
nathalie.casati@epfl.ch

November 20, 2007

Abstract

In this project, the goal is to find out how many SPEs can be used at the same time to access the main memory using their DMA Unit without causing a too large loss of performance. To get this limit number, different benchmarks will be performed from the PPE. This process will reveal the overhead generated by the well-known bottleneck of the Cell Processor, namely the bus to the Memory Interface Controller.

1 Introduction to the Cell architecture

The Cell Broadband Engine (CBE) is a microprocessor designed in collaboration by IBM, Sony and Toshiba since 2000. It is intended to be faster than other architectures by removing the slowest parts of the design.
This document will now focus on the particular case of the PlayStation 3, which includes a CBE running at 3.2GHz.
The main elements of the processor are :

- The Power Processing Element (PPE)
- The 6 (usable) Synergistic Processing Elements (SPEs)
- The Element Interconnect Bus (EIB)
- The I/O Controller (XIO)
- The Memory Interface Controller (MIC)

They are represented in figure 1.

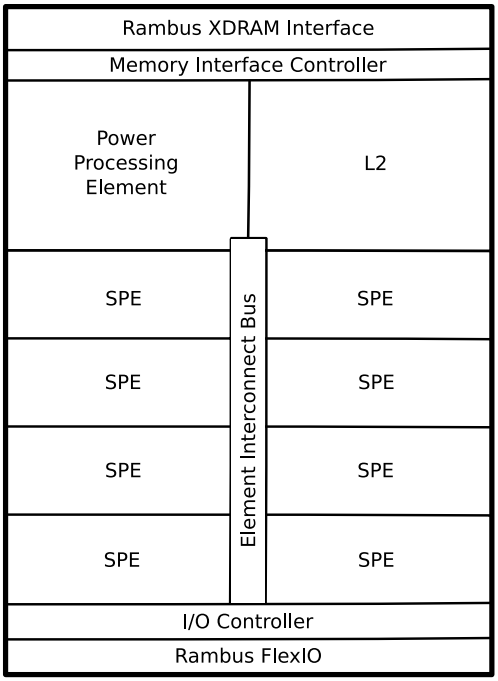

Figure 1: Cell Broadband Engine Architecture

The PPE consists of a 64-bit in-order PowerPC processor connected to a 512K L2 cache. It is dual-threaded and is able to run multiple operating systems including Linux. It acts as a coordinator for the SPEs.
The latter are independent vector processors. Each of them has a 128x128 bit register file and 256KB of private memory instead of a cache. It is called the Local Store (LS). Like the PPE, they are

dual-issue in-order processors and have no branch predition logic. As shown in figure 2, an SPE has two pipelines and is able to issue one SIMD computation operation and one memory access operation per cycle. The Memory Flow Controller (MFC) connects the SPU to the Element Interconnect Bus (EIB) and manages the DMA operations between the main storage and the LS. These operations must be performed explicitly by the programmer as well as the parallelization of the code.

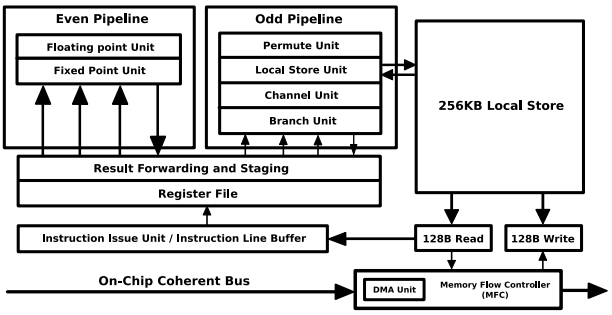

Figure 2: SPE Architecture

Each element is connected to the EIB, which is a circular bus running at half clock speed (1.6GHz) with four 128 bit wide rings. Two of them go clockwise and the others counterclockwise. It means that the data can flow in only one direction on a given ring. Every element can push 16B/cycle onto the bus, which means a bandwidth of 25.6GB/s. The bus can handle 204.8GB/s. Multiple transactions can be performed at the same time (max 3 per ring) if they don't overlap. It is also possible that they can block each other if the chosen elements use an entire ring (depending on which of them are involved). Moreover, the further an element is from another, the more expensive a transaction will be. However, a transaction cannot go more than half way around the EIB in a given direction. For this reason, the PPE is adjacent to the MIC.
The Memory Interface Controller connects an external two channel Rambus XDR memory to the EIB. It contains four 16 bit wide DRAM chips that form a 256MB memory. It has a bandwidth of 25.6GB/s (4 * 16bits * 3.2Gbps / 8) just like the other elements. The memory is not cached, the PPE only has its own cache hierarchy. A Resource Allocation Manager is attached to the memory and I/O system of the Cell processor. The latter moderates the usage of resources by giving tokens to prevent individual elements from becoming overwhelmed. Figure 3 below shows a representation of the XDR memory system (the unidirectional address and command bus is not shown).

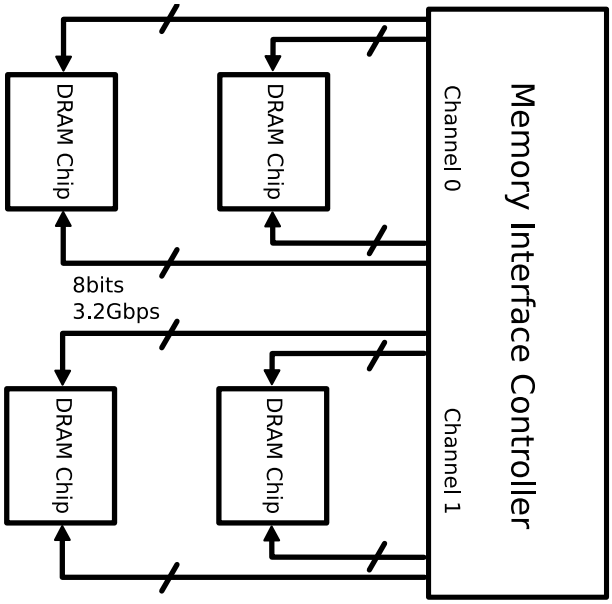

Figure 3: The two channel XDR Memory System

The main parts involved in this project are the SPE DMA Unit and the Memory Interface Controller. Their behavior will be explained in a more detailed way later.

2 Project outline

As mentioned above, the aim of this work is to investigate the bottleneck located at the entry of the MIC. Indeed, every SPE has an ingoing/outgoing bandwidth of 25.6GB/s. This implies that, when

using them all at the same moment, we are limited to what the MIC can handle. The Cell memory subsystem has 16 banks, each a cacheline wide (128 bytes). Addresses 2KB apart access the same bank (address 0 is in bank 0, adress 128 in bank 1 and so on). An individual bank can only run at 3.2GB/s, thus only 8 banks are required (4 on each channel) to get the full bandwidth. System memory throughput is maximized if sequential accesses that read or write equal amounts of data to all memory banks are made using 128 bytes of data naturally aligned in a 128-byte block.
Thus, when writing a program for the CBEA, the programmer must ensure that he uses the resources in an optimal way to get the maximal bandwidth.

It would be interesting to vary different parameters like the number of SPEs, the number of requests, the directions (r/w/rw) for a sequential access and see which actual bandwidth we get.
With one SPE, we should be able to get the 25.6GB/s when reading or writing. However, in practice, it depends on the way the data is distributed in memory and on the number of requests sent. In effect, the DMA transfers could be made more efficient by using multi-buffering to avoid wasting time during the transmission of the buffer on the bus. To achieve this, we need to send at least 2 requests. Moreover, we have to take into account that the Resource Allocation Manager gives tokens every now and then to the SPEs. It means that a single SPE could be waiting to access the memory while nothing else is actually happening on the EIB. In fact each ring is able to issue a new operation every three cycles.
While considering these properties, it seems difficult to give predictions about the real bandwidth we can get. Thus, writing a benchmark program could help understanding the situation.

References

[1] David T. Wang, *ISSCC 2005: The CELL Microprocessor*, updated on 2/10/2005, `http://www.realworldtech.com/page.cfm?ArticleID=RWT021005084318`

[2] David T. Wang, *CELL Microprocessor Revisited*, updated on 2/28/2005, `http://www.realworldtech.com/page.cfm?ArticleID=RWT022805234129`

[3] *Cell Broadband Engine Architecture and its first implementation*, updated on 11/29/2005, `http://www.ibm.com/developerworks/power/library/pa-cellperf/`

[4] *Maximizing the power of the Cell Broadband Engine processor: 25 tips to optimal application performance*, updated on 6/27/2006, `http://www.ibm.com/developerworks/power/library/pa-celltips1/?S_TACT=105AGX16&S_CMP=LP`

[5] *Just like being there: Papers from the Fall Processor Forum 2005: Unleashing the Cell Broadband Engine Processor*, updated on 11/29/2005, `http://www.ibm.com/developerworks/power/library/pa-fpfeib/`

[6] *Cell Broadband Engine Architecture and its first implementation*, updated on 11/29/2005, `http://www.ibm.com/developerworks/power/library/pa-cellperf/`

[7] *Meet the experts: David Krolak on the Cell Broadband Engine EIB bus*, updated on 12/6/2005, `http://www.ibm.com/developerworks/power/library/pa-expert9/`

[8] *Cell Broadband Engine Architecture*, *Cell Broadband Engine*

Programming Tutorial, *Cell Broadband Engine Programmer's Guide*, *Cell Broadband Engine Programming Handbook* available at `http://www-128.ibm.com/developerworks/power/cell/documents.html?S_TACT=105AGX16&S_CMP=LP`