IBM CELL PROCESSOR – *GAME PROCESSOR*!!

Compiled by Bharadwaj Veeravalli

It is indeed a collective effort of ***IBM, Toshiba, and Sony*** several years ago in creating Cell - an amazing new ***multiprocessor microprocessor chip*** that is now in the market!

Where can I find? Sony's PlayStation 3 video game console, Toshiba's high-end televisions, and IBM's blade servers, all have this wonderful powerful machine!

Cell is the beginning of a new family tree for all three companies, and it promises to branch into consumer, computer, and embedded systems for many years to come. It is truly an impressive achievement in computer architecture and semiconductor manufacturing.

INSIDE the CELL – *Embarrassingly Complex, but a Cool machine*!

Cell is a complex chip with about **250 million transistors** (compared with 125 million for Pentium 4) that runs at ***more than 4GHz***. With just the right conditions Cell can handle easily upto 256 billion floating-point operations every second!!

What's to be noted is that Cell was not developed for scientific applications, military computers, or code breaking!!!! Instead, Cell is primarily intended for ***entertainment***. But this does not mean Cell is restricted to just toys! It

can be also be used for more "serious" applications. Cell is a parallel processor, which can be useful for scientific simulations and medical imaging.

CELL COMPONENETS

Take a look at Figure 1 - a top-level block diagram of the first processor in the Cell family, more formally known as the ***Cell Broadband Engine*** (CBE)

Cell is fundamentally based around a single 64-bit PowerPC processor surrounded by eight(8) identical coprocessors. The central PowerPC processor can execute two instructions at a time – dual-issue processor.

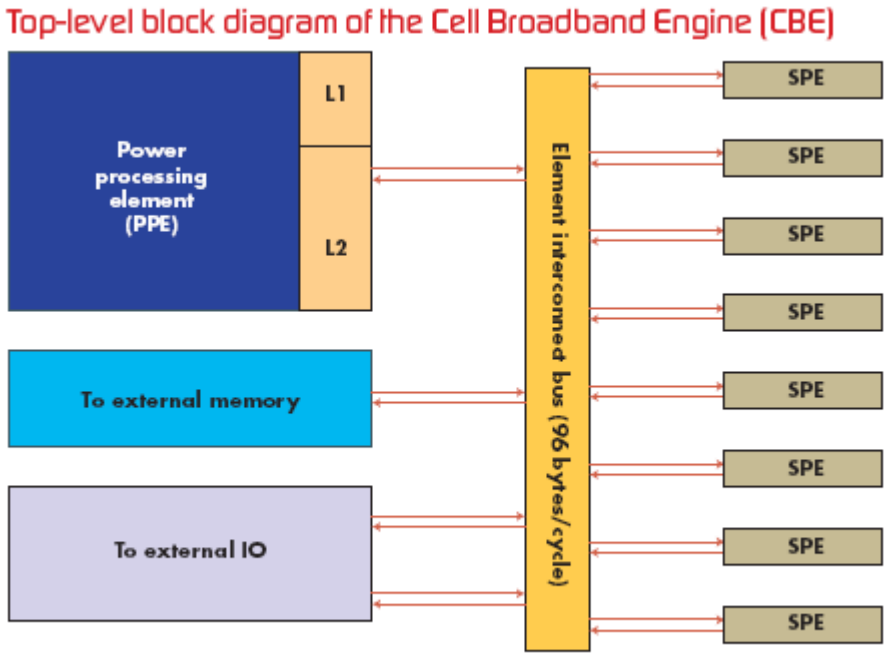

Figure 1

Although the central processor is based on IBM's PowerPC architecture (~~PowerPC 970), it has a new design and does not have an existing PowerPC core. The central processor includes the VMX (visual media

extensions, similar to *AltiVec*) instruction-set extensions to the base PowerPC instruction set. The central processor has a pair of 32K first-level (L1) caches and a unified 512K second-level (L2) cache. This should keep the processor humming along at 4GHz.

SPEED-UP by MAGIC !!

The real magic in achieving a *super-duper* gain in speed-up by cell lies with its eight "Synergistic Processor Elements," (SPEs) - See Figure 2. These are very specially designed processors built from scratch by the IBM/Sony/Toshiba team just for Cell. They are not compatible with Power or PowerPC code in any way; they have their own distinct instruction set and internal architecture. *For most code, and particularly for parallel vector operations, the SPEs do the heavy lifting*. Each SPE is identical to its neighbors, and all share the same common bus with the central Power Processing Element (PPE).

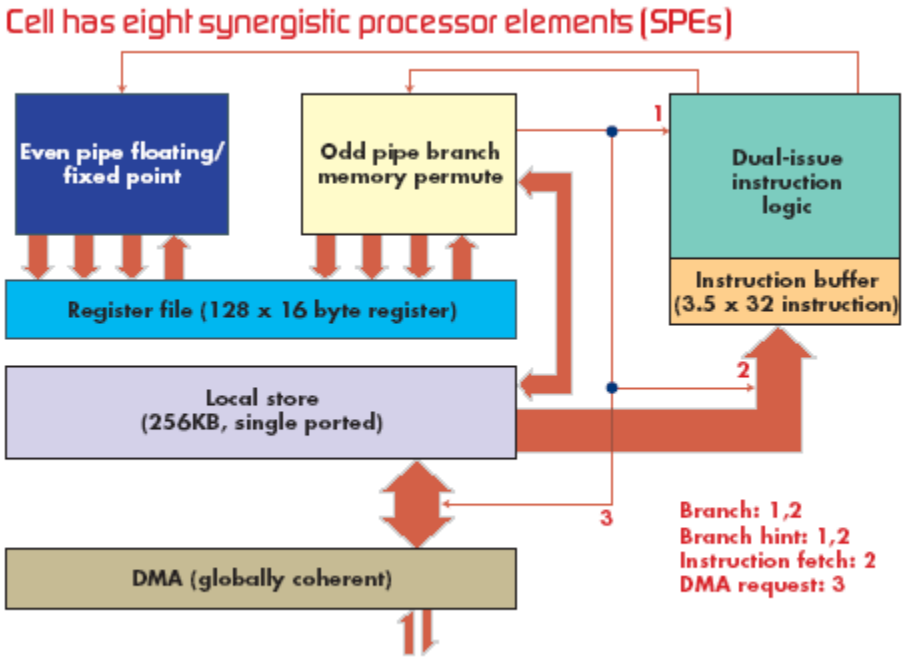

Figure 2

Like the central cell core processor, each SPE is a dual-issue machine but unlike the PPE the *two execution pipelines are not symmetrical*. In other words, each SPE can execute two instructions simultaneously but not two of the same instruction. The SPE's pipeline is "unbalanced," in that it can execute only arithmetic operations on one side (either fixed- or floating-point) and only logic, memory, or flow-control operations on the other side. Most modern x86 chips, for example, have internal execution units dedicated to math, logic, or flow-control instructions and the hardware (or the compiler) determines how many of those can actually be used each cycle. It is the combination of these elements that determines the processor's ultimate performance and suitability to a task.

Each SPE is a 128-bit machine, with 128 registers that are each 128 bits wide. Its internal execution units are also 128 bits wide, which allows each SPE to handle either very large numbers or several small numbers at once. For example, each SPE can process two double-precision floats, four single-precision floats or long integers, eight 16-bit short integers, or 16 chars or other byte-sized quantities, all in a single cycle.

Although it stretches the definition considerably, each SPE has a RISC-like instruction set. They can load and store only quad-word (128-bit) quantities and all transactions must be on aligned addresses. If you want to load or store a byte or char, you have to transfer the whole 16-byte quantity first and then mask off, merge, or extract the bits you want!!

Each SPE actually has seven different execution units, although only two can be used at a time, as mentioned previously. Because one of the two execution pipelines is dedicated to arithmetic operations, an SPE can process

fixed- or floating-point numbers nonstop while the other execution unit(s) in the other pipeline handles program flow. This reduces pipeline "bubbles" that get in the way of streaming data at top speed without interruption. Some DSP processors have similar internal architectures that separate program flow from data manipulation, and it works quite well most of the time. If the code tries to execute two arithmetic operations at once, the chip simply runs them in sequence instead of side-by-side.

Internal Data Flow mechanism

Unlike the PPE, the SPEs do not have caches. Instead, they each have a 256K "local store" which is what they can see. All code and data for the SPE must be stored within this 256K local area. In fact, the SPEs cannot "see" the rest of the chip's address space at all. They cannot access others' local stores nor can they access the PPE's caches or other on-chip or off-chip resources. In effect, each SPE is blind and limited to just its own little domain!

Why the crippled address map? Each SPE is limited to just a single memory bank with deterministic access characteristics in order to guarantee its performance. Off-chip (or even on-chip) memory accesses take time -- sometimes an unpredictable amount of time, and that defeats the purpose of SPE's! They are, after all, designed to be ultra-fast and ultra-reliable units for processing streaming media, often in real-time situations where the data cannot be retransmitted. By limiting their options and purpose, Cell's designers gave the SPEs deterministic performance.

This is where the DMA controllers come in. Each SPE has its own 128-bit wide DMA controller (64 bits in, 64 bits out) between it and Cell's local bus.

The PPE and all eight SPEs share the same bus, called the Element Interconnect Bus (EIB). Through this bus each DMA controller fetches the instructions and data that its attached SPE will need. The DMA controller also pushes results out onto the shared bus, where it can be exported off-chip, sent to on-chip peripherals, or cached by the PPE.

The central processor's L1 and L2 caches snoop the EIB, so the caches are always fully coherent. The SPEs do not snoop the bus; in fact, they do not monitor bus traffic at all. That means that the central PowerPC processor is aware of what data the SPEs may transfer but the SPEs are totally unaware of any traffic amongst their neighbors. Again, this keeps the SPEs relatively simple and limits interruptions or unwanted effects on their behavior. If the SPEs need to be made aware of external data changes, their respective DMA controllers will have to fetch it. In this case, this would be under the control of the central PPE.

Super Cell

Embedded systems developers already have experience programming multiprocessor systems; some have even coded multi-core processors. But Cell promises to up the game. Each of the chip's nine individual processor elements is itself a dual-issue machine with complex pipeline interlocks, cache-coherence issues, and synchronization problems. Keeping all eight SPEs fed at once promises to be a real chore. Yet the results are bound to be spectacular.

If your application can benefit from sustained high-speed floating-point operations and can be parallelized across two or more SPEs you should be in for a real treat, once you get the code running.

Some remarks:

IBM is working on an "Octopiler" that compiles C code and balances it across Cell's eight SPEs. Tools like that are absolutely necessary if Cell is to be a success. To take another example from the video game industry, Sega's Saturn console was not all that successful largely because its four-processor architecture (three SuperH chips and a 68000) was too difficult to program. Developers working under tight deadlines simply ignored much of the system's power because they could not harness it effectively. Cell brings that problem in spades. It is truly an impressive achievement in computer architecture and semiconductor manufacturing. Products based on Cell promise to be equally impressive. But bringing Cell to life will require real software alchemy!!

NOTE:

IBM Full System Simulator for the Cell Broadband Engine [*IBM. Cell Broadband Engine Programming Tutorial v1.1. IBM developerWorks., 2006*] is a generalized simulator that can be configured to simulate a broad range of full-system configurations. The simulator supports full functional simulation and is able to simulate and capture many levels of operational details on instruction execution, cache and memory subsystems, communications, and other important system functions.

Furthermore, it supports cycle-accurate simulation, which not only models functional accuracy but also timing. It considers internal execution and timing policies as well as the mechanisms of system components, such as arbiters, queues, and pipelines.