

TOTALVIEW TECHNOLOGIES WHITE PAPER

**Overcoming the Challenges
of Developing Programs for
the Cell Processor**

Chris Gottbrath,
TotalView Technologies





W H I T E P A P E R

Abstract

The cell processor provides a tremendous amount of computational power to a wide range of users of platforms from entertainment to high-end scientific and technical simulations and computations. Applications in all of these domains need to be carefully written for or ported to the Cell processor to be able to obtain the kind of performance that the cell is uniquely capable of delivering. When an application is ported to the Cell there are a number of unique architectural features that need to be taken into account. This paper briefly reviews these architectural features and the challenges that they pose to application developers. This porting process generally involves the introduction of a significant amount of additional code and the refactoring of some existing code within applications – both of which can easily introduce new bugs. TotalView, an advanced source code debugger, is ideally suited to troubleshooting applications during development on multicore systems. It was recently ported and is available to simplify development for anyone writing from scratch, porting, or maintaining applications on the cell. Particular attention will be paid to the changes that needed to be made to the TotalView debugger in order to extend its capabilities so that developers have the kind of visibility and control they need in order to understand what is happening in a program running on the Cell.

Introduction

Different varieties of the Cell processor are available for use in a wide variety of contexts. It is used in PlayStation3 gaming consoles, IBM QS20, QS21 and QS22 blades, and Sony BCU-100 systems. It is also being used in a number of high profile High Performance Computing clusters, including the LANL RoadRunner which tops the November 2008 list of the world's fastest supercomputers (as recorded on top500.org).

Like the more familiar multi-core processors from Intel and others, the Cell incorporates multiple cores so that multiple streams of computation can proceed simultaneously within the processor. But unlike typical multi-core CPUs, which typically include a set of four, eight or more cores that behave essentially like previous generation processors in the same family, the Cell has two different kinds of processing cores. One is a general-purpose processor, referred to as the Power Processing Element (PPE), and the other eight are highly optimized for performing intensive single-precision and double-precision floating point calculations. These eight cores, called Synergistic Processing Elements (SPEs), are capable of performing about 100 billion double-precision floating point operations per second (100 Gflops).

Recognizing the significant interest in the Cell and concerns about porting and programmability, TotalView Technologies has introduced a version of the TotalView debugger specifically adapted for debugging on the Cell. Doing so required significant changes to the way that the debugger models what takes place within threads and processes in order to provide computational scientists and developers with a clear representation of what is happening in a Cell program.

The Challenges of the Cell Architecture

The architecture of the Cell presents two general challenges that software developers must solve when writing new software or porting existing software to the Cell. The first challenge is breaking the key components of a program into small chunks that can execute on the eight SPEs. In numerical programs, this often means dividing the data into small independent units. In other cases, individual tasks or pipeline stages may be delegated to specific SPEs. This issue of problem decomposition is similar to that of adapting an application to a distributed memory cluster environment, although the granularity is different due to the limited memory space directly available to each SPE.

Each of the eight SPE cores that is part of the Cell processor has its own independent registers and a small amount of local memory (256KB) used for storing instructions and data. This memory acts similarly to a cache in a general-purpose processor, in that it has a limited size and can be read and written very quickly. Unlike a cache, however, its contents are directly managed by the application. The code units running on the SPE elements are allowed to initiate direct memory access

W H I T E P A P E R

operations that can copy chunks of data from the main memory into the SPE local store or back to main memory from the local store. The same memory is used for the machine instructions and global memory, heap memory and stack. The heap and the stack change size over time and the programmer must take care to avoid collisions between these different memory ranges since there is no memory protection. This explicit management of memory is the second major challenge in designing Cell implementations. Because each processor has a completely separate local store, the structure of a program for the Cell is very different than the structure of a traditional multi-threaded application, where all the threads share the same memory.

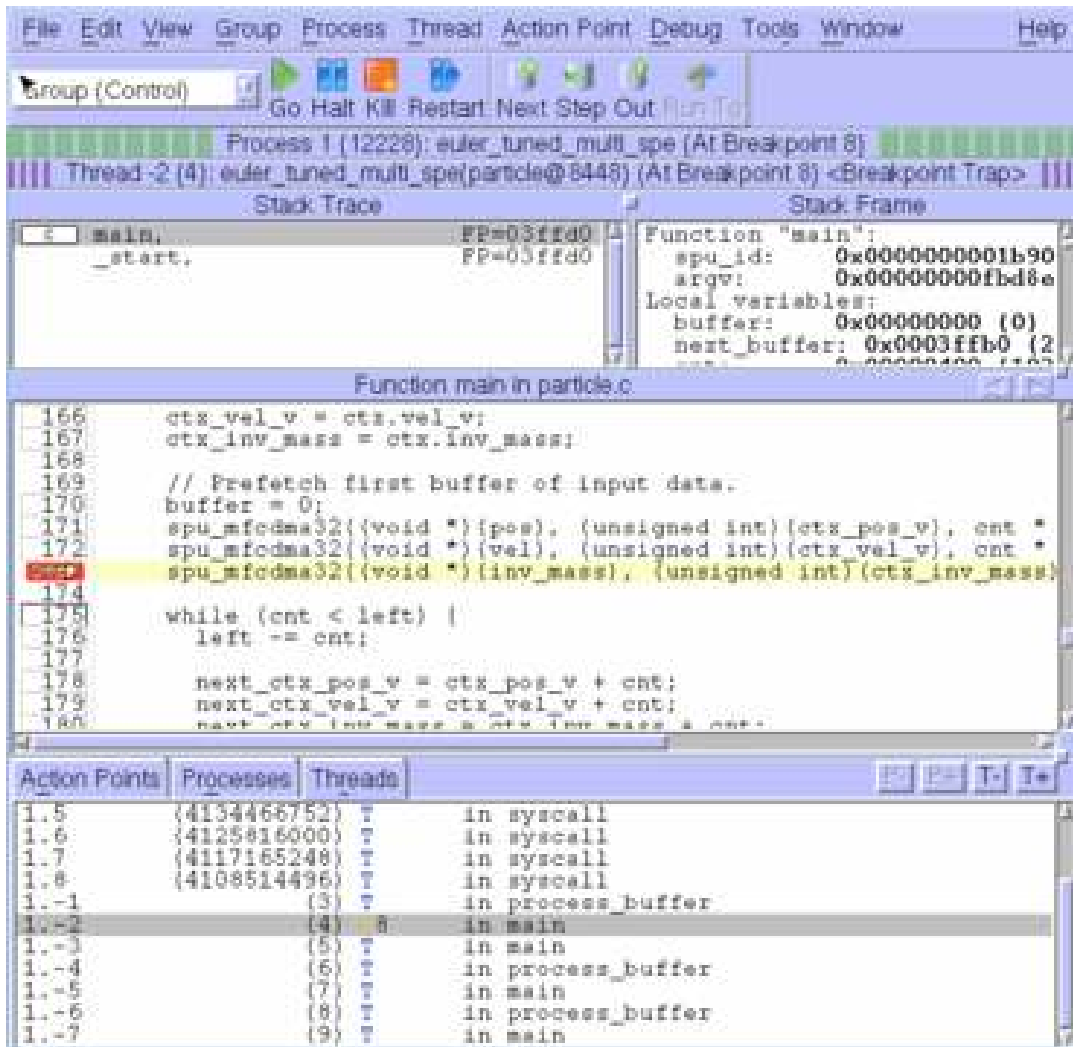


Figure 1 A simple display of code running on a Cell processor. In this example, one of the SPE threads is requesting data from main memory in preparation for a computation. Other SPE and PPE threads are stopped in a variety of states.

Developers who want to write code for the Cell processor will need to find ways to address these two issues in their programs. Architecture abstraction layers, like the one provided by RapidMind, may allow developers to write code that will run on a range of processors, including the Cell. For developers to obtain the best possible performance, however, they need to make use of the control obtained by programming for the SPE themselves, at the expense of longer and more complex code.



W H I T E P A P E R

The Cell provides exceptional performance by making it possible to achieve and sustain a high level of concurrent execution. Each SPE element can execute simultaneously and asynchronously, and has 128-bit wide registers and support for vectorized instructions that can apply a mathematical operation synchronously to more than one number at a time. Performance gains do come with tradeoffs, however, including increased complexity and unpredictability. When working with a serial application in which all computations happen in a single sequence, the system tends to be highly predictable. Given a set of input conditions, a serial program tends to behave the same way every time, even if it is malfunctioning.

Concurrent applications, especially malfunctioning concurrent applications, may behave in different ways if the sequence of operations in the various threads differs from run to run. These differences lead to elusive race conditions when only some sequences provide the correct behavior. In order to eliminate race conditions, developers can use synchronization constructs such as barriers to constrain the set of execution sequences whenever two or more threads need to read or write the same data. Synchronization needs to be applied very carefully, as its misuse will result in reduced performance or deadlocks.

Adapting TotalView to the Cell

TotalView is a highly interactive graphical source code debugger that provides developers working in C, C++ or FORTRAN with a way to explore and control their programs. Originally designed to debug one of the first distributed memory architectures, the BBN Butterfly, TotalView has since been continually enhanced with a focus on multi-process and multi-threaded applications. Today, TotalView is used to develop, troubleshoot and maintain applications in a wide range of situations.

For example, one group uses TotalView to develop Linux-based commercial embedded computing applications consisting of a single process with a hundred or more separate threads that simultaneously interact with a graphical user interface (GUI), sensors, online databases and network services. Other users troubleshoot sophisticated mathematical models of complex physical systems in astrophysics, geophysics, climate modeling and other areas. These models typically consist of thousands of separate processes that run on large-scale high performance computing (HPC) clusters on the top 500 list. In both cases, TotalView provides users with a debugging session in which they can examine in detail any one of the many threads or processes that make up the application, control each thread or process singly or as a part of various predefined and user-defined groups, and display multiple types of data and state information from across many processes and threads.

The version of TotalView for the Cell applies these same user-interface techniques and capabilities to the multiple threads of execution that are running on the PPE and SPEs within each Cell processor. It also extends to Cell applications that run as distributed memory parallel applications on clusters of Cell BE blades. To accomplish this, it was necessary to significantly extend the definitions and models of processes and threads that the debugger uses internally to represent what is happening.

TotalView's traditional model for programs has three tiers: *threads*, *processes*, and *groups of processes*. A *thread* is a single execution context, with its own stack (and therefore its own copy of any automatic or local variables) and its own state (either running or stopped). A thread may also have additional characteristics like a handle to "thread private" memory, an identity related to a specific OpenMP construct in another thread that is part of the application, or an operating system-assigned thread ID. A thread shares a single virtual memory address space, including both text (instructions) and data (global variables, heap memory, and stack memory) sections, with the other threads that may make up a process.

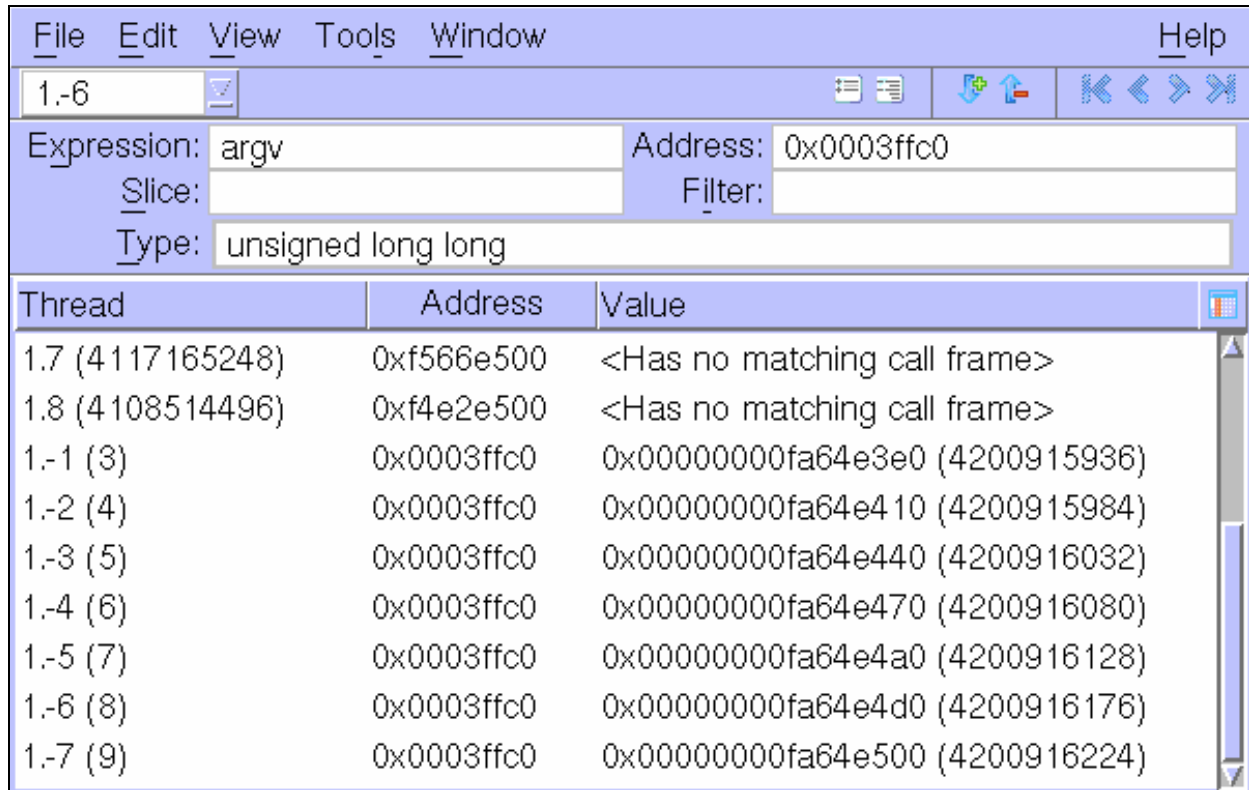
A *process* is made up of one or more threads that are all executing a single executable "image" that represents a program. The image involves the base program and a list of library images that are loaded either at launch or during run time. A process also has a number of resources attached to it that are managed by the operating system, including a virtual memory address space, file handles, parent and/or child relationship with other processes, and an operating system-assigned process ID.

Groups are made up of one or many processes running on one or more machines. These processes can be all executing copies of the same program image or, more rarely, even a different image. TotalView notes which processes in a group share the same image and shares and/or reuses information wherever possible. When the user sets a breakpoint on a line of code

W H I T E P A P E R

within a group, TotalView ensures that the breakpoint appears at the right place in each process that shares the same image, even if those images have different offsets (which can happen due to randomized library load addresses across the many nodes of a cluster).

In order to support the Cell processor correctly, several changes to this model were necessary. Most significantly, the SPE threads that make up a Cell process have a more distinct identity than threads created on a traditional processor. The SPE core uses a different instruction and register set than the PPE core, so the Cell version of TotalView can handle both, transparently switching between them depending on the kind of thread under operation. The SPE thread actually executes in a separate local memory space, so all the characteristics of memory space traditionally associated with processes are now associated in TotalView with threads on the Cell. As a result, looking up a variable or function in any specific SPE thread or in the PPE thread will give very different results, depending on the variables and functions that exist within the memory local to that thread.



Thread	Address	Value
1.7 (4117165248)	0xf566e500	<Has no matching call frame>
1.8 (4108514496)	0xf4e2e500	<Has no matching call frame>
1.-1 (3)	0x0003ffc0	0x00000000fa64e3e0 (4200915936)
1.-2 (4)	0x0003ffc0	0x00000000fa64e410 (4200915984)
1.-3 (5)	0x0003ffc0	0x00000000fa64e440 (4200916032)
1.-4 (6)	0x0003ffc0	0x00000000fa64e470 (4200916080)
1.-5 (7)	0x0003ffc0	0x00000000fa64e4a0 (4200916128)
1.-6 (8)	0x0003ffc0	0x00000000fa64e4d0 (4200916176)
1.-7 (9)	0x0003ffc0	0x00000000fa64e500 (4200916224)

Figure 2 A simple display of variable data as seen by the SPE threads of a Cell program. In this example, there are eight SPE threads executing the same bit of code. All eight have a variable named args at 0x3ffc0, but because each SPE has its own local store, the value of the variable can be different for each thread.

This is important to clearly represent what is actually happening in the program. Depending on the Cell executable's construction, the SPE threads may look like separate programs (which happen to do all of their IO via DMA calls) with their own main() routine. Much of the key information about the code running in the SPE threads is different from the PPE process that contains it. User-defined data types, functions, and global variables that accidentally or intentionally have the same name in both the SPE and PPE probably refer to distinct things, and the code may be different between two different SPE threads that are part of the same process.



W H I T E P A P E R

A single Cell program may load many distinct images at the same time to run on different SPEs, and a breakpoint set on a specific line of a specific SPE thread may or may not need to be duplicated across other threads (the user can choose). It certainly shouldn't be set at that same address in threads that don't execute that line of code there. Since the threads are executing and issuing DMA memory read and write requests and synchronization operations around those requests, there is a significant opportunity for race conditions and deadlocks. TotalView provides a set of thread control commands that allow developers to approach these problems systematically by exploring specific sequences of execution among the various SPE and PPE threads.

The intensive explicit memory management required for each unit of data moved in and out of the SPE local memory means that developers are concerned with low-level issues of data representation, layout and alignment. Since they are moving units of memory from the PPE to the SPE and back again, it is quite common to want to examine the data on both sides of the transfer. TotalView provides the ability to explore the data in any thread in exactly the same terms as it is being used within that thread. This is a significant benefit. One computational scientist noted that before TotalView was available on the Cell, he had to perform offset calculations with a calculator every time he wanted to look at a variable on the SPE.

To understand the data that has been transferred to or from the SPE threads, scientists and programmers can navigate data structures and large arrays, following pointers as if they were HTML links in a browser, and sort, filter and graph the data they find.

The TotalView process group model applies to the version on the Cell with very little modification. Users can work freely with large groups of processes running on one or many blades. The only change is a more nuanced model that allows breakpoints to be shared between threads that are all executing the same image, rather than between processes executing the same image.

The changes made to TotalView in order to provide for clear troubleshooting on the Cell processor highlight the architectural distinctions that are uniquely challenging for scientists and programmers writing software for the Cell. The capabilities that were developed in response to these distinctions make the process of adapting software to the Cell a bit less daunting for those scientists and programmers. Now, TotalView provides a ready and efficient tool for doing exactly that.

References

- Cell Architecture, <http://www.ibm.com/developerworks/power/cell/documents.html#6>
- IBM QS22, <http://www-03.ibm.com/systems/bladecenter/hardware/servers/qs22/index.html>
- Rapid Mind, <http://www.rapidmind.net/>
- RoadRunner, <http://www.lanl.gov/roadrunner/>
- Sony BCU-100, <http://pro.sony.com/bbsccms/ext/ZEGO/ZEGO.shtml>
- Sony PS3, <http://www.us.playstation.com/PS3>
- Top500 list, <http://www.top500.org/>
- TotalView, <http://www.totalviewtech.com/products/totalview.html>

To find out more

Contact [TotalView Technologies](#).