

TOTALVIEW TECHNOLOGIES WHITE PAPER

A COMPREHENSIVE DEBUGGING FRAMEWORK FOR THE MULTI-CORE AGE

Dick Andersen
Vice President of Marketing
TotalView Technologies
dick.andersen@totalviewtech.com
www.totalviewtech.com



A COMPREHENSIVE DEBUGGING FRAMEWORK FOR THE MULTI-CORE AGE

Dick Andersen — *Vice President of Marketing*
TotalView Technologies
dick.andersen@totalviewtech.com
www.totalviewtech.com

Over the last several years, the computing industry has experienced a sea change in how processing technology delivers performance gains. Increasing the clock speed of the single core processor has given way to parallel, or multi-core, processing technology.

With multiple processing cores on the same chip, computers can provide fully parallel execution of multiple software threads.[i] Multi-core processing technology provides performance and productivity enhancements that are far beyond the capabilities of single core processors, resulting in a broad range of new computing opportunities and possibilities.

This is an exciting time for software developers. Before the advent of multi-core, parallel processing was strictly the domain of high-performance (or high productivity) computing (HPC). But as multi-core brings parallel processing to the desktop, application development is evolving to keep pace. Software applications developed on multi-core processing technologies are moving beyond the realm of super-computing and into the mainstream programming world, including development environments in special effects and animation, oil and gas, finance, internet application development, CAD/CAM, automotive, aerospace, and telecommunications.

But writing applications for multi-core and multi-processor machines means more complex code and its inherent frustration, delays, and headaches. Programmers must be able to easily manage the complexities of a new style of computing and programming. The need for additional productivity-enhancing tools has never been greater. For programmers developing applications powered by multi-core and multi-processor technology, software debugging and performance analysis tools play a critical role in enhancing development productivity and application performance.

This white paper discusses the emergence of multi-core and multi-processing and their impact on application development. We discuss how software debugging technology is evolving to meet the requirements of multi-core and multi-processing. Finally, we suggest a framework-based approach to debugging and describe the technologies that would comprise such a framework.

Emergence of Multi-Core Processing and Impact on Application Development

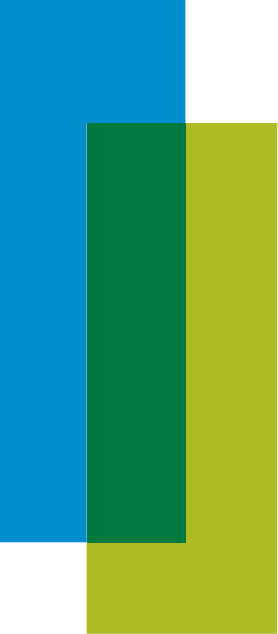
HPC application developers have made innovations based on the dependable theory of Moore's Law, stating that the number and speed of transistors on a processor would double every two years. Moore's Law has held firm since 1965, but a new HPC world has emerged — one that doesn't rely on increasing the speed of single core processors. And it compels application developers to revisit Moore's Law and to rethink the future of software innovation.

Simply put, the single core processor has hit a wall. To increase processing speed and fit all those transistors on a single chip, the components have become smaller and smaller and require much more power. That makes them harder to cool. What good is a 10 GHz chip that can't manage its heat and power requirements?

Enter multi-core processors. Semiconductor manufacturers have abandoned the decades-old approach of increasing clock speed and in its place have adopted the strategy of increasing the number of execution cores in a single processor. Sharing a single cache and bus interface, multiple "brains" on a single processor operate independently, maximizing processing speed and power efficiency. Multi-core, says analyst Nathan Brookwood of Insight64, is the new megahertz.[ii]

Dual and quad core processors are only the beginning. Experts predict six, eight, sixteen, and even more cores on a chip in the market very soon. Moore's Law isn't dead. But instead of predicting the number and speed of transistors on a chip, the principle will be used to forecast the number of cores in a processor.[iii]

What does this mean for application developers? Historically, new software releases add more features, and with each upgrade, an application carries out more (and more sophisticated) tasks. Programmers determine how to meet the increasing demands on the processor without slowing the application's speed and throughput. Developing applications for the single core processor was easy: programmers confidently added more features knowing that future processors with faster clock rates would keep pace with performance requirements.



However, as the use of single core processors is waning due to production issues, application developers need a new programming model — one that is adapted to multi-core processors. Multi-core processing provides an unprecedented opportunity to improve the performance, user interface, and responsiveness of parallel and multi-threaded applications and to develop applications more efficiently.

Extremely high-performance applications previously commanded the processing power of a supercomputer because of space, speed, and power requirements. Now, multi-core processing technology makes complex parallel processing techniques — formerly used almost exclusively in the HPC industry — available to mainstream software developers working in any industry, on any system.

Not Just a Debugger... an Integrated Debugging Framework

The growth of high-performance computing in various industry sectors is exciting news for application developers. But multi-core processors have added another layer of complexity to technical servers. Running multiple processes and threads in parallel poses tremendous development challenges, such as multiple processor support, shared memory debugging, register and instruction level debugging across multiple processors, parallel process communication and thread control. The potential to widen the gap between performance and productivity is high.

The proliferation of multi-core, multi-processor technology and its complex, multi-threaded and parallel code increases the need for robust tools that will help developers as they write or adapt software to the parallel architecture. For example, advanced debugging tools are a critical component of software development in the multi-core age.

But to create multi-core, multi-processor applications, programmers need more than just a debugger. They require a well-defined, logical, and coherent support structure for organizing and managing the complex debugging process. Today's application developers need a complete framework of interlinking technologies that will enable them to develop the next generation of multi-threaded, multi-process programs. The five key technologies that comprise a comprehensive approach to multi-core debugging are:

- **Source Code Debugging.** For applications that rely on concurrency, source code debugging techniques enable developers to find source code errors, reliably reproduce race condition and find deadlocks and other related problems.

- **Memory Debugging.** Because memory bugs can create erroneous behavior that varies from program run to program run, or only become apparent long after the error occurs, they are especially difficult to find.
- **Performance Analysis.** Developers must be able to measure, understand, and improve the performance and efficiency of both serial and parallel applications across a wide range of platforms.
- **Data-Centric Debugging.** In a multi-core environment, data transformation occurs throughout the system. Programmers must be able to easily find data-related problem areas and analyze data from multiple threads as the program executes.
- **Active Web Debugging.** The use of active web technologies such as Java, AJAX and other Web 2.0 continues to grow. Developers need fully integrated capabilities for debugging applications containing Java code interfaced with native C/C++ code through JNI.

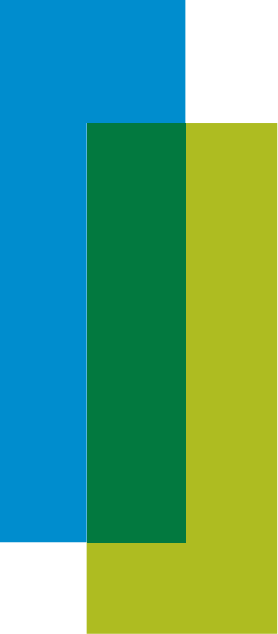
An integrated debugging framework that includes these five technologies provides developers with the architecture, methodology, and tools to address the most common multi-core application problems and simplify the addition of newly developed debugging technologies. A comprehensive architecture such as this reduces the complexities of the multi-core age and improves developer productivity and quality.

A debugging strategy that addresses each of these areas is needed to meet the challenges of complex multi-core, multi-process application development. Debugging solutions that do not include these five core technologies won't stand up to the next generation of software programming.

Source Code Debugging

Source code debugging is the major component of the software development cycle. It starts with the first prototype and continues whenever the code is changed, the application is moved to a new platform (hardware or software), or problems identified in testing are being resolved.

Debugging parallel programs barely resembles serial debugging. In concurrent applications, race conditions, deadlocks, and related problems are common, and can be difficult to understand and resolve. Focusing on an individual process or a thread when a program has thousands or tens of thousands of each is impractical and can be daunting for even the most seasoned programmer.



While a developer may ultimately want to look at the behavior of one process or thread, he or she must have the ability to organize processes and threads into groups sharing similar characteristics, and debug the program by group. The developer must be able to asynchronously control a thread, a process, a group of threads, or a group of processes and run these groups on one or more systems and cores.[iv]

Source code debugging tools should be flexible, providing the programmer with multiple ways to control program execution. They must allow automatic grouping, which creates predefined groups, or manual grouping, which enables the developer to make ad hoc groupings based on his or her specific needs.

Multi-process and multi-threaded programs must be controlled in ways that minimally disrupt normal run-time patterns. When debugging serial programs, it is possible to stop the entire program. But when debugging more sophisticated parallel programs, some processes and threads should be allowed to run while others should not.

The ideal source code debugging solution provides precise control of individual threads or processes of concurrent applications, drastically improving developer productivity and reducing the cost of developing multi-core and parallel applications.

Memory Debugging

Memory problems, such as memory leaks and buffer overflows, are among the most frequent and challenging problems faced by developers. Errors caused by memory bugs often vary between program runs, and many times, the error is not apparent until long after it occurs.

Memory bugs can be caused by the faulty allocation and deallocation of dynamic memory. When a program fails to deallocate memory that is no longer needed, it reduces the amount of available memory. A build-up of memory consumption leads to a memory leak, which diminishes the application's performance and can eventually cause the application to fail. Memory leaks are most common in C/C++, languages that do not provide automatic memory management.

Buffer overflows occur when an application attempts to allocate data beyond the boundaries of a temporary buffer. The additional data corrupts data values in adjacent memory locations, and may cause the application to crash or produce incorrect results.

Effective memory debugging requires insight into the memory heap, so that developers can track and interpret allocated, deallocated, and leaked memory blocks. Programmers can take advantage of a debugger's ability to track memory events, stop program execution, and display event information. With memory usage reports, developers can see a program's overall usage of memory. For example, the amount of memory used by text and data areas can be displayed, as well as heap, stack, and virtual memory sizes. Tracking this information over time allows programmers to determine when memory usage becomes unacceptable.

Other valuable tools provided by memory debuggers include reports on leak detection and corrupted memory. Leak detection reports simplify the process of finding leaks by providing detailed information about the leaked memory block, such as the program line number and how many bytes were leaked. Corrupted memory detection shows when heap memory bounds have been exceeded.

Performance Analysis

Good performance and efficient resource use is critical to the success of many applications. Addressing application performance during the development cycle — instead of waiting until a customer complains — makes good business sense.

Developers must be able to measure, understand, and improve the performance and efficiency of serial and parallel applications across a wide range of platforms. Performance analysis allows programmers to visualize how their programs will perform in a multi-core environment.

Unfortunately, improving performance and efficiency is tedious and challenging, often consuming large amounts of valuable development time. Gathering accurate data about the performance and efficiency of programs with multiple processes and threads is difficult. Understanding how a program's efficiency and speed can be improved requires making sense of the data gathered — and that's even harder.

The ideal performance analysis solution allows developers to identify critical sections, potential bottlenecks, and potential problem areas in their code; make quantitative measurements of behavior and improvements; and track improvement throughout the process. It includes a variety of tools, such as those that identify race conditions, visualize performance bottlenecks, improve computer utilization, and trace arbitrary events.

Data-Centric Debugging

Data-related programming problems occur when a program executes but gives an incorrect answer. Developers face the following challenges in debugging these types of data errors in a multi-core environment:

- Data transformation is complicated because it does not occur in a unique location. It occurs throughout the system.
- Data-related errors can be confusing because the program may run to completion, yet still give an incorrect answer.
- Programmers must analyze data from multiple threads as the program executes.
- Information about data-related errors is often presented in a confusing format.

In addition, developers have historically relied on manual solutions to address data errors. Putting a print statement in the application to try to understand the operations and sections of code that were executed before the problem occurred is common, for example. Or, to see how selected variables change over the sequence of program execution, a developer might temporarily add code to output the desired variables or even develop an elaborate framework to control which internal variables and information are exposed. These types of solutions are typically time-consuming and inefficient, because the program may need to be recompiled and relinked and the information generated is incomplete.

A data-centric debugging solution allows the recombination of data as the program executes, enabling programmers to more easily identify data transformation problems. This debugging approach does not incur the overhead of managing changes, building a framework, recompiling and relinking, or creating a useful output format.

Active Web Debugging

Multi-core applications are increasingly dynamic and interactive. Software engineers face the challenge of developing applications using active web technologies such as Java, AJAX and other Web 2.0 technologies. For example, many applications contain Java code front-ends interfaced with native C/C++ code back-ends, which traditionally can only be debugged by manually switching between two debuggers. As the use of active web technologies and languages expands, developers need a simple way to integrate the debugging of back-end and front-end code.

The best active web debugging solutions provide developers with a simple interface that allows users to work with a complex array of interrelated processes running on many machines that operate together, creating a seamless debugging environment for mixed code applications. A JNI bridge allows developers to easily move between Java and C/C++ code, without switching debugging tools.

Selecting an Integrated Multi-Core Debugging Framework

Organizations seeking to optimize software development capabilities should choose their debugging solution carefully. Besides having the five integrated technologies described above, a truly effective debugging framework will provide users with:

- **Intuitive user interface and easy-to-learn features** that enable developers of all levels to analyze and debug complex multi-process and multi-threaded applications.
- **Multi-platform compatibility**, including Linux, UNIX, and Mac OS X. This means that developers do not need to learn a different tool for each project.
- **Support for multiple programming languages**, including C/C++, FORTRAN, and MPI/Open MP applications.
- **Scalability**. A robust solution ensures consistently accurate and reliable results when debugging challenging applications that require massive amounts of data, millions of lines of code, and extensive parallelism. In addition, the ideal debugging solution should scale from a single workstation running a small number of processes to supercomputers running thousands of processes and everything in between.
- **Minimum impact on application performance**. A non-intrusive solution has the least impact on applications being debugged. Where possible, the program should run at nearly full speed when being debugged, without cumbersome binary or source code instrumentation process programs.
- **Multiple licensing options** provide programmers with affordability and flexibility and meet an organization's evolving needs. This ensures that the technology is available to software teams of all sizes.



Conclusion

Multi-core processors have added another layer of complexity to software development. Formerly the domain of supercomputing, multi-core technology brings parallel programming into the mainstream programming world.

But as the performance enhancements of parallel programming become available to developers of all capabilities and programming organizations of any size, software developers need comprehensive productivity improvement tools such as software debuggers. Technology is evolving to meet the complex debugging requirements of multi-core processing.

A comprehensive support structure — an integrated multi-core debugging framework — for organizing and managing the debugging process is a critical component of successful software development in the multi-core age. A complete debugging architecture includes five key debugging technologies:

- Source Code Debugging
- Memory Debugging
- Performance Analysis
- Data-Centric Debugging
- Active Web Debugging

To meet the challenges of complex multi-core application development, programming organizations require a debugging strategy that addresses each of these areas. Debugging solutions that don't include these five core technologies won't stand up to the challenges of multi-core processing.

[i] Intel® Multi-Core: An Overview, <http://www.intel.com/multi-core/overview.htm>

[ii] The Role of Intelligent Design in the Evolution of Multi-Core Processors, <http://www.insight64.com/downloads/IntelligentDesign.pdf>, Nathan Brookwood, Insight64, February 28, 2006.

[iii] Technology Assessment: What will be the Next Phase Change in HPC, and Will It Require New Ways of Looking at HPC Systems? Joseph, Snell, et. al. IDC #205025: January 2007.

[iv] Organizing Processes and Threads for Debugging, Barry Kingsbury. TotalView Technologies: 2007.