Deploying HPC Tools into Embedded Software Development to Save Time and Money

Read About

Amdahl's law

HPC applications for HPEC

Debuggers

Profilers

Mapping

Introduction

Estimating and managing a software development effort is a combination of art and science. The cost and schedule estimates of the program are based on the predicted lines of code, and the number of lines of code that can be designed, coded and debugged in a mythical man-month. The lines of code per man-month can vary greatly depending on factors of code reuse, complexity, level of documentation, and even the skill set of the developers. For a project to be successful, it must be on time, within budget, and deliver the features and functions as promised. According to the Standish Group's CHAOS report, of 50,000 software projects, only 19% of the projects were successful. Meanwhile, 52% of the studied projects were complete but came in over cost, over time, and/or missing some of the specified features and functions. Finally, a whopping 29% were cancelled. [1] Given these statistics, the choice of tools for software development becomes even more important in today's multi-processor environment. For years in our embedded defense community, developers have used VxWorks® and its associated tools, and vendors of embedded defense hardware had also started making their own tools. These tools had varying levels of success with the single core processors, and of course, the traditional "printf" was and continues to be a debugging favorite of most developers.

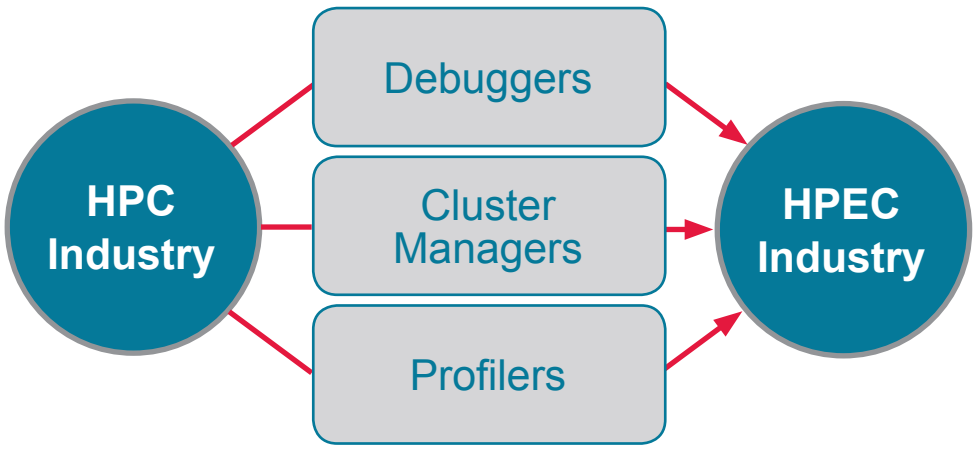


Figure 1: HPC tools migrate to embedded market

Info

curtisswrightds.com

Email

ds@curtisswright.com

With the new generation of multi-core Central Processing Units (CPUs), Graphics Processing Units (GPUs), and field-programmable gate arrays (FPGAs) as building blocks, embedded defense systems have become even more complicated to develop and debug. To take advantage of these embedded "supercomputers", the code must be executed efficiently in parallel across many nodes. If serial coding techniques were used, it would be comparable to buying a Ferrari and driving it around in first gear. Though the data interface may be high speed serial, once captured, it is more efficient to program multiple independent and concurrent activities to process the data. One could ask: how much improvement can be achieved if the price is paid to develop parallel code? The answer is Amdahl's law, which predicts the theoretical maximum speed-up for a program using multiple processors. Amdahl's law states that in parallelization, if P is the proportion of a system or program that can be made parallel, and 1-P is the proportion that remains serial, then the maximum speedup that can be achieved using N number of processors is 1 / ((1 - P) + (P / N). As N approaches infinity then the maximum speedup approaches 1 / (1 - P). This means concurrency needs to be identified in the algorithm and plans made to exploit it by breaking up the computation into tasks that can be divided among the processors. Given this transition in our market space toward parallel programming, how can we more intelligently manage, develop and debug the increasingly more complex code? Traditional tools such as traces analysis, serial debuggers, and even our dear friend the "printf" fall short of this increased challenge.

Some vendors and customers in the embedded defense space have tried to develop parallel tools, which they have found is not a trivial exercise. Most of these efforts are being supported by small teams of less than ten people, and when adopted, are only being used by a handful. Therefore, the products are not as full-featured or reliable as tools with a much wider user base, and using these immature tools, could negatively impact your program's cost and schedule. With this paper, we explore the potential role of High Performance Computing (HPC) tools in the embedded market, speeding delivery time while decreasing costs. We will look at the power of combining HPC tools such as cluster managers, debugger and profilers, and mapping solutions.

HPC Tools to the Rescue

Over the past decade, HPC – High Performance Embedded Computing (HPEC) without the 'Embedded' – has evolved into a mature and feature rich set of software development tools including math libraries, communications APIs, testing tools and cluster managers. The same hardware building blocks (processors, GPUs, and fabrics) from the HPC environment are now used in the HPEC world, but on a smaller scale with lower power consumption. For example, the University of Texas's super computer, Stampede, uses Intel® dual Xeons, NVIDIA® Tesla GPUs, and Mellanox® FDR InfiniBand®; the same hardware as in our world. With over 6,400 processors with 522,080 cores, Stampede will consume a maximum of 4.5 megawatts of power. Consider the tools needed to manage a system of that size, and then imagine debugging applications spread across that number of processors. Floating point performance, throughput, latency and standard software APIs are only a few examples of concerns that must be addressed in both the HPC and HPEC industries. Think that supercomputers only perform complex simulations and no real-time applications or control-loops? One need only look at the HPCs used for High Frequency (stock) Trading (HFT). High-Frequency traders move in and out of short-term positions at very high volumes and speeds, aiming to capture sometimes a fraction of a cent in profit on every trade. Typically, these systems can currently handle huge amounts of volume while boasting round-trip order execution speeds (from hitting "transmit order" to receiving an acknowledgment) of 10 milliseconds or less. Given these similarities between HPC and HPEC, the HPEC industry can turn to well-developed and trusted HPC development tools to apply the same high standards to the embedded market.

Simplifying the process with cluster managers

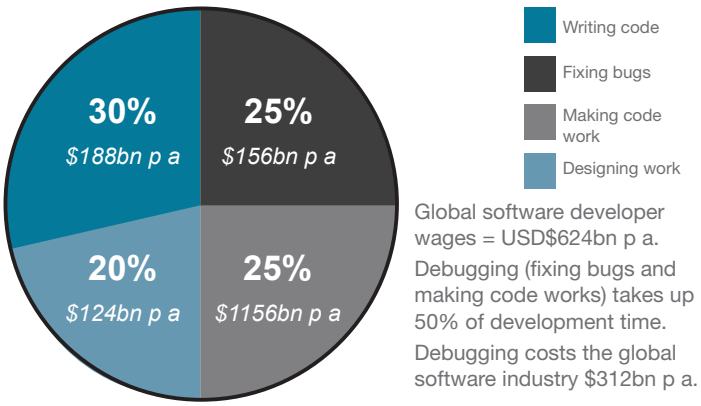
One of the first challenges with a new embedded system is the setup and configuration of the software and hardware interfaces. The system may first be setup and configured to the corporate network, and then moved to a classified network, and then deployed into the field. A board could go bad and need to be replaced or you may want to add new boards to a system in unused slots. Several years into the program, you may want to do a tech-refresh and

update all the boards. Have you planned for a Linux® system administrator in your schedule and budgeted to handle all these eventual scenarios? Turning to a cluster management system provides you with all the tools you will need to build, manage, operate and maintain a cluster in your HPEC System. In this context, a cluster is considered the collection of nodes in the system. These cluster management installation wizards guide you through the process of installing your cluster from bare boards to a full development system in a matter of minutes. By answering a few questions about the system being built, it configures all of the resources such as custom kernels, disks, and networks. Using the image-based provisioning, kernel images can be maintained for different board types in the system, including GPUs. Adding, deleting or moving a board to another slot becomes as simple as a click. Cluster managers also support developers loading their own kernel images to a single board or a combination of boards. This gives the advantage of enabling multiple developers to work on separate groups of processors on the system – a limited resource – simultaneously. The separate kernel images allow the developers to always start their work session at a known point, eliminating any doubt regarding the state the system was left in by a previous user. This image-based provisioning architecture can also guarantee the same version(s) of software is loaded on all the boards, of all processing types, for full system testing and delivery, eliminating the headache of replacing boards. The revision control also empowers you to track changes to the software images using standardized methods, and, effortlessly roll nodes back to a previous revision if needed.

The health and monitoring features included in the cluster managers provide a visual status of the entire system. Temperature, CPU loading, and disk space are just a few of the parameters monitored. It also logs and displays the boot up messages for all of the boards and any errors and warnings from the system logs. This removes the need to connect a terminal to serial ports to debug and configure the compute nodes, saving you time spent searching for the right serial cable, a serial-to-USB adapter, or the driver for the adapter (especially if you are on a network that can access the outside world), and installing terminal software on your computer. This total management of the system supports the complete lifecycle, enabling a seamless transition from lab development, to flight readiness, through manufacturing and delivery.

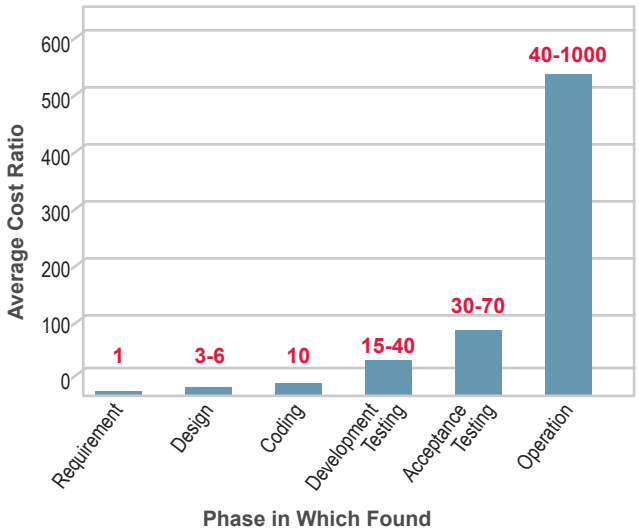
Building an ecosystem to minimize time spent debugging

When the system is up and you are ready to start developing code, every manager knows controlling software costs and schedules is like herding cats. From numerous company surveys and research studies, it has been determined that 50% of a software effort is consumed by fixing bugs and making the code work correctly. While only 20% is designing the code itself, and 30% is spent writing code. [2] Once a system is fielded, the cost of fixing an error can be 100 times as high as it would have been during the development stage. [3] Situations like having to fix a bug on a flight line at China Lake emphasize the need for the careful selection of an integrated development, debugger, and profiler ecosystem.



Source: Evans Data Corporation (2012), Payscale (2012), RTI (2002), CVP Surveys (2012)

Figure 2: Impact of debugging on time spent developing code and its cost wages per annum



Source: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036670.pdf

Figure 3: Relative cost to fix software errors per life cycle phase

By incorporating a high-end debugger and profiler into your system, bugs can be addressed in a much more timely and efficient manner. One example used by more than 70% of the world's supercomputers and taught in universities worldwide is ARM® Forge™. It provides debugging, profiling, editing, and building - with version control integration - within a single easy user interface. The shallow learning curve ensures developers achieve the maximum value from their time and the tools.

The advanced capabilities of debuggers will help quickly discover the root causes of software defects. The sooner an error is detected and corrected, the better. It's like knitting a sweater. If you spot a missed stitch right after you make it, you can simply unravel a bit of yarn and move on. The longer the process continues, the more threads will need to be unraveled. For example, a static analysis tool will easily discover an uninitialized variable, but if the tool is not used to find it, weeks or months of effort could be spent trying to solve the problem in a deployed system.

A popular example of a debugger is ARM DDT. Unlike other debuggers, the DDT debugger visualizes data across multiple processes allowing developers to quickly spot unexpected data points and generate statistical summaries of data structures. Does the input data contain Nans or does it contains Nans after the processing? Another valuable, time-saving feature of debuggers like DDT is the automatic recording of debugging sessions with comments into online logbooks. Being able to repeat the same test-setup again and again and to have the testing correctly logged is simply invaluable. This eliminates any transcription errors or omissions when writing into a log book, or more commonly, forgetting to write it in the log at all. Teams are no longer burdened with hearing, "I don't remember what test parameter I used" or "I have seen that problem before, but I don't remember how I solved it."

Debuggers can log variables and events in the background without affecting system timing enabling the system to collect data overnight or however long it takes for the problem to occur. This is invaluable to catch seemingly random non-repeatable bugs. Errors tend to breed more errors, causing a schedule nightmare and forcing your projects to take much longer than anticipated. The more errors you "fix", the more errors seem to arise. By integrating with common version control tools to highlight recently changed code, these debuggers help to pinpoint the new errors.

As all developers have become aware, it is much more difficult to solve issues that span multiple processors than serially executed code. The debugger specifically must have the ability to debug and control threads and processes, both individually and collectively. With these tools or with this type of debugger you can create groups of processes, based on variable or expression values, current code location, or process state. Then you can set breakpoints, step, play the individual or your pre-defined groups of threads. In addition, you can also display parallel stacks; a scalable view of all the stacks of every process and thread. This narrows down problems for any level of concurrency, from a handful of processes and threads to your entire system. With these tools, the finding and fixing of deadlocks, live locks, race conditions, message synchronization and "I don't know what is wrong" becomes less daunting and schedule bloating. Despite this breath of capabilities, if the debugger carries too much overhead, it effects the timing of the system and becomes worthless. Therefore, the debugger must be able to work interactive with many processes within certain time constraints.

Map it!

Even though the code may be up and running, it may not be meeting timeline or it may require some fine tuning for optimal performance. The conventional approach is to insert timers into the code, run the code, analyze the results, and change the code; repeat, change, repeat. Even inserting a timer and then removing it can introduce errors into the code. Using a profiler for C, C++ code provides in-depth analysis and bottleneck pinpointing to the source line. Profilers allow you to compile your code without instrumenting it and remembering arcane compilation settings. Adaptive sampling rates combined with on-cluster merge technology ensures exactly the right amount of data is recorded - whether you run a few processes for several minutes or thousands of processes for a week.

Most profilers starts by identifying the procedures and source lines that are the time hogs. Using CPU instruction analysis, it shows where vectorization could be employed. A rough rule of thumb is that an application would spend 60-70% time performing computations and 10-20% for communications. These tools can point to computation and communication imbalances, including those performance issues causes by MPI or pthread synchronization issues. Profilers perform equally well in finding and solving memory and I/O bottlenecks, saving your developers from spending frustrating late nights in the lab.

Authors

Tammy Carter, M. S.
Sr Product Manager
Curtiss-Wright Defense Solutions

The Solution

Cluster managers, debuggers, and profilers are the software equivalent of the combination of an oscilloscope, spectrum analyzer, and logic/network analyzer. You wouldn't try to develop hardware without these tools, so why would you possibly consider developing software without them? Because of the similarity of the hardware, the maturity of the tools, and the larger installed user database, importing tools from the HPC community into the embedded market provides you with a solution that can reduce both cost and time to deployment. The cluster manager saves time and money by easing the set-up and maintenance of the system configuration. It also eases the strain on developer resources and scheduling by providing a simple method for sharing the lab development system. In production, it helps ensure quality and customer satisfaction by exact duplication of the software images. Research consistently shows 50% of programming time is spent debugging. Using a comprehensive system debugger can slash this time, reducing schedule creep and cost over-runs. Profilers help you optimize and benchmark your code, and can perform regression testing as well. Combining these tools creates a fully integrated HPEC development environment that is fully tested, validated, and benchmarked.

Learn More

White Paper: Understanding HPEC Computing: The Ten Axioms

Technology: High Performance Embedded Computing (HPEC)

Products

- OpenHPEC Accelerator Suite
- OpenHPEC Commercial Development System (CDS)
- OpenHPEC Quick Start Quick (QSK)
- CHAMP-XD1
- CHAMP-XD2

References

[1] Standish Group 2015 Chaos Report (Oct 2015). Retrieved March 22, 2016 from http://www.infoq.com/articles/standish-chaos-2015

[2] http://embedded-computing.com/articles/speeding-software-debugging/#

[3] http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036670.pdf