The ECN Roundtable - Multicore Adoption

This month's question, **"What is the biggest challenge facing the adoption of multicore processors by the engineering community for more applications?"** was answered by people from Texas Instruments, Freescale, The Mathworks, and National Instruments.

From Casey Weltzin, LabVIEW and Multicore Product Manager, National Instruments

I am convinced that software development tools have a lot to do with the fact that more engineers are not incorporating multicore processors into their designs. While today's multicore CPUs promise higher computational throughput than previously possible, the problem is that achieving this throughput requires programming and debugging multithreaded applications - something that can require significant investment when working with traditional sequential tools and developers that are not familiar with parallel programming.

To face this challenge and lower the investment required to program multicore processors, the engineering community has started to shift towards using higher-level parallel programming APIs and other task-based abstractions. In short,

languages have began to evolve to automatically handle low-level threads so that programmers can write applications while focusing on the parallel tasks they are trying to accomplish. Furthermore, we are seeing a resurgence of dataflow as a model that builds on this task-based programming method. Since dataflow applications inherently contain information about data dependencies, intelligent compilers can automatically map them onto multicore processors and handle thread synchronization transparently. It needs to be stressed that this is not merely theory; dataflow languages such as LabVIEW are able to automatically take advantage of multicore CPUs today.

We must continue striving to lower the effort required to program multicore processors by improving our software tools. At the end of the day, it will only be once we enable programmers to concentrate on the tasks they are trying to accomplish, rather than the low-level implementation details, that we will see mass-adoption of multicore processors in the engineering world.

 *From Stephen Turnbull, Portfolio Manager, High Performance Embedded Processors, Freescale Semiconductor*

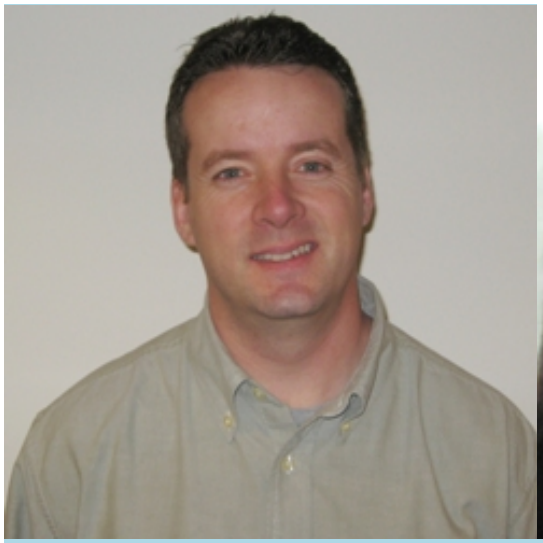
As multicore processors grow in popularity the issues facing new users and new applications are crystallizing. By far the most common challenge is how to truly harness the increased processing power of this new breed of processors. While the hardware has evolved, board design, thermal and power solutions are readily keeping pace. However, there are still software related issues to be resolved.

For example, how does one partition an application over multiple cores in an application? One option is to run SMP and have the OS distribute the work evenly, but perhaps that is done inefficiently. Also, you can run AMP and hope that your system partitioning is optimal, that you have not left some cores overloaded and others idle. Ultimately this is very application specific with no single right answer. In fact the answer may be some combination with yet other tasks offloaded to on-chip hardware accelerators.

Running control, data & services on one processor, likely with more than one OS,

brings new challenges for the software engineer. One approach to simplify the software challenge is to provide some level of virtualization. There are many 3rd party hypervisors available which provide a high level of virtualization and simplify the programming challenge, but this comes with a cost in terms of associated overhead and performance loss. Freescale's response to this is a hypervisor mode in the core itself that supports efficient partition management. Closely coupled to the chip architecture, this allows programmers to simply create secure partitions with dedicated memory and peripherals for a single thread running on a single OS without the overhead of full virtualization.

As the momentum behind multicore processors continues to grow and the architectures advance, we will all continue to look to the software community to help us maximize the benefits of these leading edge architectures.




From Silvina Grad-Freilich, manager of parallel computing and application deployment marketing and Loren Dean, senior engineering manager in MATLAB development

We see the lack of tools and languages that allow users to easily leverage multicore environments as the biggest challenge facing the adoption of multicore processors. Because these tools are still in their infancy, engineers must learn the skills traditionally associated with the HPC community. This brings significant challenges to the engineering community in terms of resources, skill sets and time that cannot be easily overcome.

Language and tool providers need to develop parallel interfaces that let users have a range of options available to them to leverage concurrent environments. These options should vary to meet user preference; in one option, for example, users choose to make no code changes and just implicitly leverage multicore systems.

Another option would allow users to work at a very low level, dealing with data and algorithms for concurrent environments. Ideally, these interfaces should also let users operate not only on a single machine but across distributed environments. MathWorks, as both a language and tool provider, is an example of one company that has been working hard to provide engineers these needed capabilities across its product line and the flexibility to choose the option that best suits their needs.

From Brian Glinsman, General Manager, Texas Instruments' Communications Infrastructure business

Until recently, one of the biggest challenges to multicore programming was the overall SoC design and ensuring that multicore cores had non-blocking access to all resources. The goal was not to have to wait for access to memory or peripherals; in other words, no unnecessary delays.

Since many vendors have designed architectures to address this challenge, the next big hurdle to overcome is being able to effectively use a multicore design in a real-time and processing efficient system. The extra cores in a multicore design boost performance and accelerate the resolution of high computational problems (and low computational problems aren't ones designers particularly concern themselves with). In a PC environment, for example, with 90 percent of what you do, you don't really have to wait for anything. In rare cases where heavy computation is required, the extra cores kick in and result in better performance. Still though, at the end of the day, a quad core chip design doesn't really give you 4x the performance of a single core chip.

As you move into real-time systems, if you have eight cores running at 1GHz,

ultimately, the designer wants to view it as a single core running at 8GHz. This is extremely difficult to achieve today. The only time we have come close to it is when you have similar tasks that only use a fraction of a core and you want to run hundreds of them simultaneously. In that instance, you can utilize each core optimally. In the case when tasks are larger and not uniform, it becomes a load management problem. If you look at a load management gant chart for scheduling critical paths, if a task is completed in core 1 and core 2 is busy, you wait. If core 2 finishes its tasks but can’t start something else before another processor is done with its task, you stall. The real challenge is partitioning tasks across all cores so that they’re utilized to their maximum capacity.

I’ve outlined several of the challenges that the multicore community is facing, but what are the solutions? Ultimately, this problem needs to be solved both in hardware and software: smarter hardware enabling the dynamic partitioning of tasks, and a much more advanced software development environment that can simulate measure and reallocate tasks based on actual usage statistics. While some of these tools do exist, today it really comes down to human beings effectively partitioning tasks. Ultimately, this will transition to symmetric multiprocessing utilizing cache and memory coherency, including efficient share memory architecture.