GETTING STARTED WITH SEQUENCECEL
Optimizing Parallel Processing Performance and Coding Efficiency with AMD APUs and Texas Multicore Technologies’ SequenceL Auto-parallelizing Programming Solution

Heterogeneous system architectures, exemplified by AMD's Embedded R-Series and G-Series accelerated processing units (APUs), deliver new levels of parallel processing performance with minimal power consumption. Meanwhile, multicore programming solutions optimized for parallel processing, exemplified by Texas Multicore Technologies' SequenceL programming solution, equip software application developers to achieve new levels of system performance and development agility. The combination of these advanced technologies ultimately yields ultra-responsive, parallel processing-driven embedded systems and liberates developers from conventional sequential coding processes.

Ongoing innovation in the multicore processor domain has had a transformative impact on a wide range of applications, unlocking incremental performance gains with each successive transition from one to two/four/eight core processors and beyond. But traditional CPU architectures optimized for scalar data structures and serial algorithms are underequipped for the vector processing requirements associated with more compute-intensive applications.

With the advent of AMD’s Embedded R-Series and G-Series APUs, the silicon-level integration of general-purpose, programmable scalar and vector processor cores for high-speed parallel processing establishes a new level of processing performance for embedded systems, at previously unobtainable performance-per-watt ratios. This merging of low-power x86 CPU cores with the parallel processing performance of a discrete-level general-purpose graphics processing unit (GPGPU) in a single device is enabling the high-speed processing required to handle intensive numerical computations.

APU-CALIBER COMPUTATIONAL PERFORMANCE

AMD Embedded R-Series and G-Series APUs outperform the computational capabilities of traditional multicore CPU-GPU hybrid compute models. Consider AMD’s Embedded R-Series APUs – with between 128 and 384 compute units delivering a calculated 172 to 563 SP GFLOPs\(^1\) of performance and Thermal Design Power (TDP) ranging from 17 to 35 Watts (average power below 13 Watts\(^2\)), the performance-per-watt benefits yielded via these APUs surpasses traditional CPU-GPU configurations. And with the AMD APU architecture, direct access to the unified memory shared between the CPU and GPU enables a streamlined data transfer path and the lowest possible latency, yielding a fully optimized data pipeline.
MAKING THE MOST OF PARALLEL PROCESSING

While many systems today run on multicore CPUs, many software applications have yet to take full advantage of the parallel processing potential of these systems. Indeed, parallel programming limitations remain a lingering but wholly surmountable barrier to exploiting the full performance benefits of multicore architectures, including massively multicore AMD APUs.

The ultimate goal of any parallel processing approach is to achieve 100% parallelization, thereby unleashing the greatest possible processing performance for a multicore-driven application. Anything less than 100% parallelization, in accordance with Amdahl’s law, constrains the upper limits of achievable performance acceleration regardless of the number of processor cores added to the system. Sequential applications tuned for 90% parallelization, for example, do not achieve speedup greater than 10X no matter how many compute cores are present. While a 10X acceleration gain is certainly a significant improvement, it’s well under the theoretical performance gains that can be achieved via AMD APU architectures as we approach 98% or 99% parallelization.
But getting to 90% parallelization is extraordinarily difficult – some would say impossible – to achieve for large-scale applications via manual sequential coding processes, even with a large number of programmers on the job. Auto-parallelizing software compilation technologies like Texas Multicore Technologies’ advanced SequenceL solution can unlock the potential to achieve high levels of parallelization for AMD APU-based systems, while introducing an abstraction layer above the sequential code generation layer that liberates application developers, engineers, and scientists from concentrating their efforts on the obscure intricacies of sequential program implementation. By enabling these innovators to develop and refine their applications with whiteboard-like ease, auto-parallelizing software compilers equip them to achieve new levels of creativity, productivity, and application code transportability.

**SEQUENTIAL PROGRAMMING CHALLENGES**

The fundamental challenge with parallel programming is that the sheer complexity of the code and data structures obscures opportunities to employ parallel processing. Using traditional software development methodologies originally developed for sequential programming, there are three well-defined manual steps required for parallel programming.

1. **Identify parallelisms**: Analyze the problem to identify tasks that can execute in parallel.
2. **Expose parallelisms**: Restructure a problem so tasks can be effectively exploited. This often requires finding the dependencies between tasks and organizing the source code to manage them effectively.
3. **Express parallelisms**: Express the parallel algorithm in source code using a parallel programming notation.

The expertise, time and effort required to manually develop parallel programs using these traditional techniques can be significant. It isn’t enough to simply understand how an application can be split into parallel threads. The second barrier to parallel programming is the need for the application developer to understand how the multicore system will process the multiple threads. Among the details of the computing system that the application developer may need to understand to optimize the application are such things as load balancing, memory access and processor communications.

- **Load balancing**: Though the target system might have multiple cores, some of those cores will be engaged by the operating system or by other critical applications. The developer must understand how to distribute the application across available resources, and how to keep all the processors fed with work at the right times.
- **Memory access**: As processes are distributed, the data accesses may also be distributed across processors. The user must take into account the available memory and how the processors communicate with memory to avoid processors sitting idle waiting for data, creating memory conflicts, or worse, corrupting data.
- **Processor communications**: Application developers must become experts in multicore bus protocols to ensure the inter-processor communications don’t create bottlenecks that slow the operation of the application, making the parallel code slower than the sequential implementation.
And so it is necessary, in order to optimize an application for parallel processing, that the application developer understand the target multicore system in great detail, manipulating processes that are normally taken for granted in sequential programming. It is for this reason that software applications developed for specific multicore architectures typically need to be rewritten to accommodate new and different multicore architectures—and this, of course, requires significant investment in labor and support resources. As a result, software applications that previously gained performance with every new processor release have become static, impaired by the ability of developers to add new features and functionality and/or pivot to a new processing architecture.

**TAKING THE BURDEN OFF OF PARALLELIZING CODE**

Texas Multicore Technologies’ SequenceL multicore programming technology neutralizes the aforementioned challenges and helps application developers take advantage of new generation AMD APU architectures. This approach ultimately allows developers to specify application functionality without the need to manually identify parallel structures in the code.

[Note that while this approach will allow developers to let compilers manage application segments they have already parallelized manually, they will still need to work through the three defined steps and will need at least some understanding of the target multicore system.]

Texas Multicore Technologies’ auto-parallelizing programming approach introduces a simple declarative language for expressing algorithm intent—without the need to specify parallelisms or implementation structure—and employs compiler mechanisms that automatically identify, expose and express the parallelisms, enabling application developers to quickly generate optimized threaded application code from simple algorithmic descriptions of the code functionality. This approach avoids the need for developers to become experts in heterogeneous processor design, since the technology distributes the application across the available processors and optimizes memory access and communications for parallel implementation. Parallelizing programming technology actually becomes the abstract layer that allows the developer to focus on what the function is without concern about how it will be computed—leading to dramatic improvements in innovation and productivity.

Texas Multicore Technologies’ SequenceL software compiler automatically extracts parallelism from the application code, enabling code compilation to any parallel architecture from common source code. As a result, the auto-parallelized code remains independent of computing system architecture, eliminating the need to re-write application code for new multicore processor releases. Applications can be ported to new multicore processors, new machine configurations and new operating systems without incurring the costs, time and effort associated with traditional programming methodologies.
GETTING STARTED WITH SEQUENCEL

SequenceL provides developers with a declarative, functional language that self-parallelizes, which means that if you express the algorithm in SequenceL, the SequenceL compiler will generate massively parallelized C++ code, which is race- and deadlock-free. It will also generate GPU code to support parallel processing on an AMD APU.

The best way to view SequenceL programming is to think in terms of operators that guide the interaction of data structures. Stripped away is the need to break data structures apart or reassemble them (which is required of other languages). There are four ways to approach a problem in SequenceL:

1. If there is a formal, precise definition of a problem, transcribe the solution in SequenceL.
2. If the sequences involved do not require knowledge of subscripts, simply define the “base case” of the solution.
3. If indices of Sequences need to be referenced, provide an indexed function.
4. When all else fails, provide a recursive definition to the problem solution, which may require the sequences to be broken apart and/or reassembled.

The way that SequenceL is converted into an executable program is as follows. The compilation process consists of two passes. First, a compiler converts SequenceL into parallel C++ source code (it’s a hybrid compilation/interpretation process). It carries out ‘large grain parallelization’ producing parallel C++ code. The second pass is through a C++ compiler that has a special set of parallel libraries. You can think of that as a ‘fine grain parallelization,’ augmenting the parallelization of the first pass.
CONCLUSION

By utilizing AMD Embedded R-Series and/or G-Series APUs in concert with Texas Multicore Technologies’ SequenceL auto-parallelizing programming technology, developers can achieve significant gains in performance, energy efficiency, and developer productivity. The SequenceL programming technology automatically identifies parallelisms in application code and generates highly parallel, race-free code optimized to the AMD APU. This approach affords application developers of all levels of expertise the ability to quickly and easily create high quality, parallelized code for any AMD APU-based embedded system, thus reducing time to market and unlocking the full potential of APU-caliber vector processing for today’s most compute-intensive applications.

About Texas Multicore Technologies

Texas Multicore Technologies’ mission is to enable multicore acceleration for software developers by delivering transparent programming solutions that make applications transportable across parallel processing architectures. For more information visit www.texasmulticoretechnologies.com.

About AMD

AMD is a semiconductor design innovator leading the next era of vivid digital experiences with its groundbreaking AMD Accelerated Processing Units (APUs) that power a wide range of computing devices. AMD Embedded Solutions give designers ample flexibility to design scalable, x86-based, low-cost and feature-rich products, and drive energy conservation into their systems without compromising application performance or compatibility, graphics performance or features. For more information, visit www.amd.com/embedded.

©2012 AMD, the AMD Arrow logo and combinations thereof, are trademarks of Advanced Micro Devices. Windows is a registered trademark of Microsoft Corporation in the U.S. and/or other jurisdictions. 3DMark is a registered trademark of Futuremark Corporation. Other names are for informational purposes only and may be trademarks of their respective owners. PID 52860A

1 Calculated SP GFLOPs = (# of x86 cores x (128 bit (FPU) / 32-bit (SP Operation)) * CPU Base Frequency) + (# of shader units * (64 bit (shader) / 32-bit (SP Operation)) * GPU Max Frequency)

2 The average power for the 35W TDP AMD R-464L APU when system is running one iteration of 3DMark™ 06 default run was 12.861 Watts. R-464L testing was performed on an equivalent A10 Series APU. System configuration: AMD A10 2.3GHz 4/1/D, “Pumori” development platform, 4 GB RAM, Windows® 7 Ultimate.