

Preparing for the Revolution  
Maximizing Dual Core Technology

January 2006

Prepared By:  
Douglas Eadline, Ph.D.  
Basement Supercomputing

## Introduction

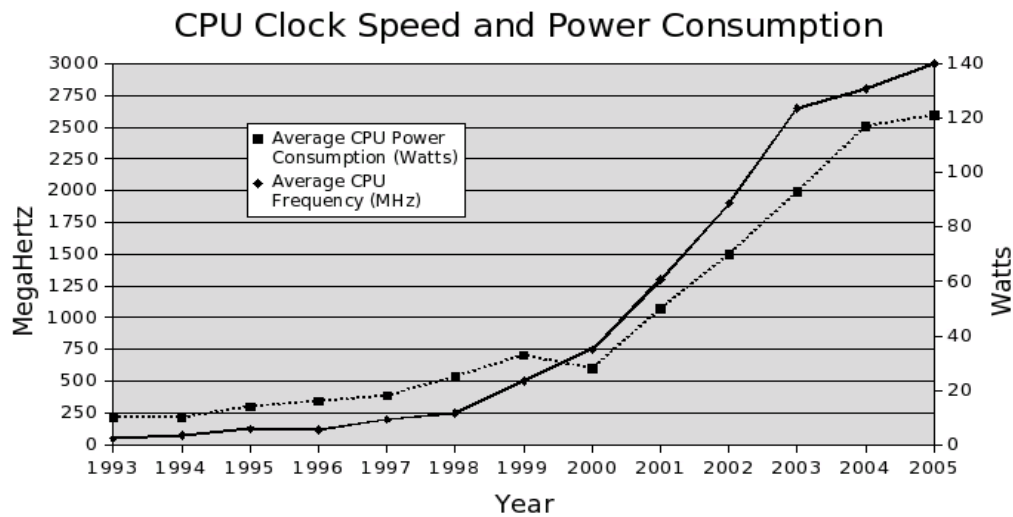
There is revolution (or evolution) occurring in the computer industry. Recently, both AMD and Intel have introduced new chips that utilize multiple processing units in a single package. Instead of having one central processor, or brain, computers will now have multiple brains with which to run programs. While this technique is not necessarily new, it is the first time these types of architectures have been mass produced and sold to the commodity PC and server markets.

This revolution will effect everyone who uses a computer. From laptops, to game consoles, to large servers, the age of the *multi-core* has begun. From an end-users perspective, this change will remain hidden, however the expectation for continued price to performance gains similar to those experienced over the past twenty years will remain.

Programmers will find providing additional price to performance for multi-core designs a challenging task. There is no silver bullet or automated technology that can adapt current software to operate on multi-core systems. This white paper will address these challenges and provide programmers and managers with a basic understanding of the issues and the solutions that will be required to leverage the new multi-core revolution.

## The Road to Multi-core

The computer market has long enjoyed the steady growth of processor speeds. A processor's speed is largely determined by how fast a clock tells the processor to perform instructions. The faster the clock, the more instruction's that can be performed in a given time frame. The physics of semiconductors have placed some constraints on the rate at which processor clock speeds can be increased. This trend is shown quite clearly in *Figure One* where the average clock speed and heat dissipation for Intel and AMD processors plotted over time.



From a power consumption perspective, it is clear that something had to be done. The continued climb in power consumption (and thus heat generation) would require additional cooling and electrical service to keep the processor operating. The solution was to scale out processor cores instead of scaling up the clock rate. The drop-off in clock speed on the graph indicates the delivery of the first dual-core processors from AMD and Intel. These processors are designed to run at a slower clock rate than single core designs due to heat issues. These dual-core chips can, in theory, deliver twice the performance of a single-core chip and thus help continue the processor performance march.

## Multi-core Road Maps

Both Intel and AMD are selling multi-core processors today. From publicly available documents, the companies expect to release quad-cores in the 2007 time frame and speculation is that eight-way cores will be introduced in the 2009-2010 time frame.

2005	Dual Cores
2007	Four Cores
2009+	Eight Cores

For servers and workstations which traditionally have two processor sockets available, this means the total number of cores per motherboard can easily reach sixteen by the end of the decade. In addition, AMD hypertransport (Direct Connect) technology currently allows eight-way motherboard designs (two 4 processor motherboards). Extrapolating this to eight way cores means that sixty four core servers are not an unreasonable expectation.

## The Challenges

The challenge facing the industry is how to use the sudden doubling of processor power. Fortunately modern operating systems are equipped to take advantage of multiple processors and may extend some immediate benefits to the end users in the near term. Using dual-core processors to their fullest potential on a per application basis is harder (requires re-programming) and is considered a longer term benefit. An analogy will help explain the situation.

### *The multi-processor store*

We have all waited in line at the food store. The speed at which we get our order "checked out" (processed) is related to the number of cash registers the store uses.

A store with one cash register is like a modern day single processor computer. Each customer has a cart full of items (program) that is to be tabulated (computed) by a cash register (processor). Modern operating systems use a trick, called time sharing (or multitasking), to make it look like there are multiple programs running at the same time. For instance, in the store analogy, if an extremely efficient cashier with a smart cash register processes some of your order, then processes some of the next customer, you both would appear to be moving through the line at the same time. Using this method,

customers get the illusion that they are moving through the line, but in reality, they will always go faster if they are the only customer.

The obvious solution, to anyone waiting in line, is to use more than one cash register. Indeed, this is often what large stores do to improve the flow of customers through the checkout line. The same effect will happen when dual-core processors become mainstream over next several years. More customers (programs) can be serviced (run) at the same time, but you will never get through the line any faster than you would if there was only your order and one cash register. In computer terminology, this is referred to as Symmetric Multi-Processing or SMP.

The market has grown accustomed to faster and faster "cashiers" over the last twenty years. Thus orders that once took minutes to tabulate, now take seconds and customers (programs) move faster than before. As mentioned above, processor technology is having trouble making the processors (cashiers) faster. So instead, they have introduced more cash registers.

In the near term, more processors (cash registers) means more of the users programs can run at the same time without impacting each others performance. Using modern SMP enabled operating systems, this benefit will be immediate and transparent to all users. The longer term challenge facing software developers is how to make individual programs go faster using more than one processor.

### *The Long Term Performance Challenge*

Going back to our store analogy, it is obvious that breaking your order into smaller orders and distributing them over two or more cash registers allows you to get processed faster. The same applies to computer programs. If the program is amenable to distribution, it can use multiple processors and execute faster. Commonly referred to as *parallel computing*, this method will be responsible for almost all performance gains in the immediate future. Parallel computing almost always requires re-programming existing *sequential* applications to execute in parallel. The amount of reprogramming can be trivial or monumental depending on the application. The choice of tools and techniques for this task will be critical for success in the future. Fortunately, there are existing software methods and tools for exploiting parallelism in applications. Many of these techniques are currently used successfully in the High Performance Computing (HPC) market.

## Programming Methods

Dealing with multiple CPUs is not a new idea. It has been around for many years and studied quite extensively. There is no general consensus, however, on how to program multiple processors. There are two general methods that the programmer may use. The first is *threaded programming* and the second is *message passing*. Both have their advantages and disadvantages. The correct choice largely depends on the application and target hardware.

### *Threads*

The thread model is a way for a program to split itself into two or more concurrent tasks.

These tasks can be run on a single processor in a time shared mode, or on a separate processors (e.g. the two cores on a dual-core processor can each run threads). The term thread comes from "thread of execution" and is similar to how a fabric (computer program) can be pulled apart into threads (concurrent parts). In the cash register analogy, it would be similar to breaking your order up into components and using separate cash registers. Threads are different from individual processes (or independent programs) because they inherit much of the state information and memory from the parent process.

On Linux and Unix systems, threads are often implemented using a POSIX Thread Library (pthreads). There are several other thread models (Windows threads) with which the programmer can choose, however, using a standards based implementation, like POSIX, is highly recommended. As a low level library, pthreads can be easily included in almost all programming applications.

Threads provide the ability to share memory and offer very fine grained synchronization with other sibling threads. These low level features can provide very fast and flexible approaches to parallel execution. Software coding at the thread level is not without its challenges. Threaded applications require attention to detail and considerable amounts of extra code to be added to the application. Finally, threaded applications are ideal for multi-core designs because the processors share local memory.

### *OpenMP*

Because native thread programming can be cumbersome, a higher level abstraction has been developed called OpenMP. As with all higher level approaches, there is the sacrifice of flexibility for the ease of coding. At its core, OpenMP uses threads, but the details are hidden from the programmer. OpenMP is most often implemented as compiler directives in program comments. Typically, computationally heavy loops are augmented with OpenMP directives that the compiler uses to automatically "thread the loop". This type of approach has the distinct advantage that it may be possible to leave the original program "untouched" (except for directives) and provide simple recompilation for a sequential (non-threaded) version where the OpenMP directives are ignored.

There are several commercial and open source (C/C++, Fortran) OpenMP compilers available. Like pthreads, OpenMP is ideal for multi-core designs.

### *MPI (Message Passing Interface)*

In the High Performance Computing (HPC) market, parallelism is often expressed using the MPI programming interface. In contrast to threaded approaches, MPI uses messages to copy memory from one process space (program) to another. This approach is very effective when the processors do not share local memory (i.e. they are located on another motherboard). It can be used, however, for multi-core programming as well. In particular, there are many programs that have already been ported to MPI that can take advantage of multiple-cores without any re-programming.

In our cash register analogy, an "MPI cashier" would call other stores on the phone and tell them the items in the shopping cart that they would have to tabulate. The advantage of this is that the size (scale) of the order can get very large and exceed the capacity of

the cash registers of any one store (computer). MPI is available as a library for most languages (C/C++, Fortran) and is available in both commercial and open source packages.

## Debugging Methods

There is an old joke that says, "every program works just fine, getting it to work the way you want it to is the trick." Parallel programs, like their sequential program cousins are no exception. Indeed, parallel programs represent a much harder proposition because unlike serial programs there is the notion of synchronization and data sharing. These properties can make it difficult to fully understand program behavior in a real world environment where the program execution may not be easily replicated.

The choice of debugger can be critical to the success of any multi-core programming project. Without the ability to see what the program is doing in real time, multi-core applications can be impossible to develop in any meaningful time frame.

## Recommendations

The multi-core revolution requires software developers to evaluate their codes in terms of multi-core performance now and as the number of cores increases in the future. The following recommendations are designed to help aid the transition to multi-core architectures.

**Assess the level of concurrency in your application:** Concurrency is not present (or necessary) in all applications. Some applications, by virtue of their algorithms, can only be executed in a serial fashion. Examining the algorithm is the only effective way to assess the presence of concurrency in your application. Careful attention to memory access patterns by various parts of the program is important as dependencies will inhibit the ability to operate concurrently.

**Assess, if possible, whether concurrency will improve performance:** If you can identify concurrent parts of your application, look carefully to see if executing them in parallel will result in application speed-up (reduced execution time). There are often parts of your program that could be executed concurrently, but will have no effect on the execution time. There is no point in spending time on these sections of code. Typically, you should focus your attention on computationally heavy parts of your application.

**Assess the scalability of your applications:** Another important question to ask is how scalable is my application. As more processors are added, parallel execution will always hit a point of diminishing returns. This means that creating more threads will not improve performance (it may actually hurt performance). If you find that your application can be scaled to a large number of processors, then using MPI may be a good choice. If however, you do not envision using more than four processors, then pthreads or OpenMP are probably your best choices.

**Make sure there is an adequate tool chain for your application:** This last recommendation is critical to the success of your application. There are many methods to use multiple processors, but not all of them have tool chains that provide professional

support and capabilities needed to produce software in a reasonable time and cost. Research projects should be avoided and use of mainstream compilers, debugger, and profiles should be a high priority.

## Conclusion

Taking advantage of multi-core technologies is the next step in software development. Once a plan and infrastructure are in place, the benefits of dual core processors can be realized by customers and users alike. A commitment now to parallel computing will provide solid and sustained performance growth into the future.

## References

The following sources can be consulted for general information about multi-core hardware and software. The world wide web is a good source of additional information as well.

### *Hardware*

AMD multi-core White Paper:

[http://multicore.amd.com/WhitePapers/Multi-Core\\_Processors\\_WhitePaper.pdf](http://multicore.amd.com/WhitePapers/Multi-Core_Processors_WhitePaper.pdf)

Intel Multi-core White Paper

<ftp://download.intel.com/technology/computing/multi-core/multi-core-revolution.pdf>

### *Programming Issues*

*The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software*,  
By Herb Sutter, <http://www.gotw.ca/publications/concurrency-ddj.htm>

### *pthread*

*Programming with POSIX Threads*, by David R. Butenhof, ISBN 0-201-63392-2

Pthread Tutorial: <http://www.llnl.gov/computing/tutorials/pthreads/>

### *OpenMP*

*Parallel Programming in OpenMP*, by [Rohit Chandra](#), [Ramesh Menon](#), [Leo Dagum](#), [David Kohr](#), [Dror Maydan](#), [Jeff McDonald](#), ISBN: 1558606718

*Nuts and bolts of Multithreaded Programming* by Tim Mattson  
<http://www.intel.com/cd/ids/developer/asmo-na/eng/219575.htm>

OpenMP Website: <http://www.openmp.org/drupal/>

### *MPI*

*Using MPI - 2nd Edition: Portable Parallel Programming with the Message Passing Interface* by William Gropp, Ewing Lusk, Anthony Skjellum, ISBN: 0262571323

MPI Website: <http://www-unix.mcs.anl.gov/mpi/>

## Author Information

Douglas Eadline, Ph.D., has over twenty five years experience in high performance computing. You may contact him via email [deadline@Basement-Supercomputing.com](mailto:deadline@Basement-Supercomputing.com). Further information about multi-core and high performance computing can be found at <http://basement-supercomputing.com>.

This article was sponsored by Concurrent, <http://www.ccur.com>, a global leader in providing digital on-demand systems to the broadband industry and real-time/HPC computer systems for industry and government.

©Copyright 2006, Douglas J. Eadline, All rights Reserved