

# Effective Use of Multi-Core Commodity Systems in HPC

Kent Milfeld, Kazushige Goto, Avi Purkayastha, Chona Guiang and Karl Schulz

Texas Advanced Computing Center  
The University of Texas at Austin  
Austin, TX 78758

**Abstract.** To decrease the power consumption in processors and yet increase performance, processor architects have recently moved away from increasing the clock speeds and now focus on including multiple cores on a single chip. This approach keeps power and thermal demands in check, while at the same time increases compute throughput on a single chip to satiate consumer demand and continue along the path of Moore's Law. The shift to core level parallelism (CLP) introduces new complexities into the optimization of application execution, both from a programming paradigm perspective and a system level with the need to manage effectively shared resources such as the caches, buses, and main memory. In this work we look at these and other system issues both from the perspective of micro benchmarks and application benchmarks. These tests are conducted on Intel dual-core Woodcrest nodes and AMD dual-core Opteron nodes

## 1. Introduction

Node configurations in large HPC systems have grown from single-processor to multi-processor systems as large as 32 CPUs. In the commodity space, dual-processor systems have maintained a sweet spot in price performance, as processor clock speeds have accelerated and the instruction pipeline lengthened over the last six years. Lately, the advances in clock speed have become limited by power consumption, but the growth in density remains exponential; and it is now through the replication of processing units to form multiple cores on a single chip (socket) that the peak performance of commodity processing continues to track the curve of Moore's Law. The change in architectural focus from increasing clock speed to using multiple cores introduces a new core level parallelism (CLP) in the processing layer. More importantly, as the number of cores increases per chip, data locality, shared cache and bus contention, and memory bandwidth limitations become more important to control, as resource sharing increases. Hence, it is reasonable to expect a paradigm shift in programming to synchronize processing and memory sharing through threading methodologies found in programming paradigms such as OpenMP.

In this work we discuss and evaluate architectural and software elements that will be important in optimizing programs to run on multi-core systems. In the first section, **Multi-Core Elements**, the basic architectural components of the multi-core configurations of two leading commodity processors are reviewed. In the next section, **Threads**, the costs of creating Pthreads and OpenMP thread teams on tuples of cores are measured. Also, performance considerations of thread/process scheduling and affinity in multi-cache systems are evaluated. In the **Synchronization** section, thread synchronization methods and costs are evaluated. The **L2 Cache Characteristics** section investigates the latency and throughput in multi-core caches, and the benefits and detriments of a shared cache. Also, in this section the advantages of using large-page memory support are assessed. Finally, in the **Application Benchmarks** section some core scaling information is measured for several benchmark kernels and a few applications. **Conclusions** follow at the end.

## 2. Multi-Core Elements

The change to multi-core architectures in the Intel and AMD commodity microprocessors is a major departure from the SSE and simultaneous threading extensions (Hyper Threading, HT) lately introduced into the PC world. It is a natural advancement, though, for optimizing instruction execution and conserving

power. To the architects and the chip design teams it is another concurrent component, following the parallelisms of SSE and HT. To the HPC community it is another form of parallelism, Core Level Parallelism (CLP), and must be accommodated by the software stacks and libraries to achieve optimal performance. Actually, IBM dual-core Power4 systems [1] were the first dual-core platforms in the server industry in 2001. Also, the gaming industry has already adapted multiple SIMD cores in the Cell Broadband Engine Architecture [2], which is radically different in its memory architecture and instruction set.

A multi-core system consists of replicated instruction pipelines on a single chip (for instruction fetching and decoding, arithmetic operations, memory load and store, cache lookup, etc.). Each core executes its own process or thread instructions independently. The cores can share a single L2 cache (as in the Intel dual-core Woodcrest), or connect to independent L2 caches (AMD dual-core Opteron). In the Intel quad-core system just released, two complete dual-core systems are replicated; hence, one pair of cores connects to a shared cache, as does the other pair, and the two caches are independent. This Intel quad-core (Clovertown) is a hybrid of shared and independent L2 caching. Figure 1 illustrates the shared and independent L2 configurations for the two dual-core systems used in this work.

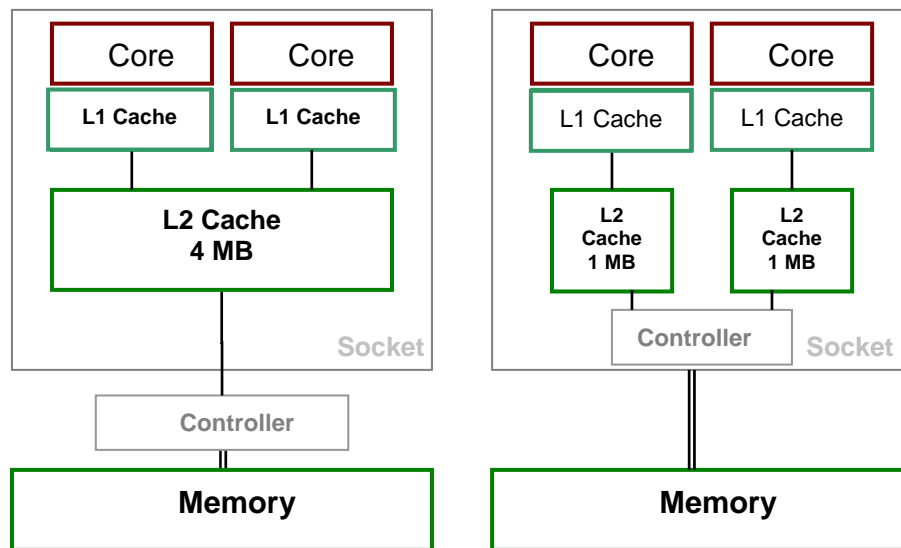


Figure 1: L2 cache configuration for Woodcrest (left) and Opteron (right).

The important characteristics of these systems for this study are listed in Table 1. Certain characteristics are not listed here, but supplied as measured values in Section 5. Some of the noteworthy differences are: the Intel and AMD L2 caches are shared and independent, respectively; the Intel/AMD L2 total cache-size ratio is 2/1; the L1-L2 occupation policy for Woodcrest is inclusive and the Opteron is exclusive; the L1 cache associativities are 8-way for Intel and only 2-way for the AMD processors. While the effects of these hardware configurations are recognized on certain genre of algorithms, there are some other less known hardware and instruction set features that can influence application performance, such as SSE2 read “transfer rates” and packed move operations. The evaluation systems in this study were configured with 2 Intel 5150 sockets (4 cores) and 4 AMD 880 sockets (8 cores).

Table 1: Summary of Opteron (AMD 880) and Woodcrest (Intel 5150) test processor configurations.

Cache Configurations						
System	Sockets	Cores per Socket	L2 Caches per Socket	L2 Cache per Socket	L2 Cache Assoc./Occupation	L1 Cache Size/Assoc./Type
AMD Opteron 880	2	2	2	1MB x 2= 2MB	16-way/exclusive	64KB/2-way/Write Back
Intel Woodcrest 5150	2	2	1	4MB x 1= 4MB	16-way/inclusive	32KB/8-way/Write Back

Processor Speed, TLB Table, and Memory Information				
System	Speed (GHz)	TLB Size	FSB (MHz)	Memory Controller
AMD Opteron	2.4	512	1000 (HT)	Integrated
Intel Woodcrest	2.66	256	1333	North Bridge

An accurate timer is an important tool for evaluating well defined benchmark kernels, memory operations, etc. In the 32- and 64-bit x86 architectures the RDTSC assembly instruction can be used to read the local core clock period counter (and store it), within a single clock period. However, when issued back-to-back there is a throughput (or latency) that represents a minimum amount of work (CPs) that must be covered. Also, when timing large loops one should remember that there may be an addition cost for reclaiming the timing instructions from the L2 cache. RDTSC counter accuracy, throughput, and total cost with instruction miss overhead, in clock periods (CPs), are listed in Table 2.

**Table 2: RDTSC Accuracy and Function Wrapper Overheads (CP = clock period).**

Platform	Accuracy	Throughput	Instruction Miss Overhead
Woodcrest	1 CP	64 CP	104 CP
Opteron	1 CP	6 CP	23 CP

### 3. Threads

Now that the total number of cores in a commodity cluster node is on the rise, and may reach 16 or 32 within the next years, even more developers will be experimenting with threading methods such as OpenMP to increase application performance. While there wasn't too much concern for process scheduling on 2-way processor nodes with MPI codes, in multicore-multiprocess systems there is a greater concern to minimize process and thread "hopping" between cores.

CPU affinity [3] is the propensity for a process or thread (process/thread) to resist migration from one CPU to another CPU. In multi-core systems, a core is simply another (logical) CPU. When threads or processes are created in a multi-core and/or multi-processor system, the Linux 2.6 kernel uses *soft affinity* to migrate threads and processes to underutilized CPUs, but with a minimal frequency to avoid accumulating a high overhead. A *cpus\_allowed* bitmask for each process/thread contains a zero or one in bit locations corresponding to logical CPUs. A bitmask of all ones, the default, means that the process/thread can run on any of the CPUs. The bitmap can be modified within a running application (by each thread/process) or changed by an external command through API calls using the process id to identify the process/thread. Process/thread migration does occur in HPC executions during I/O, when tasks become unbalanced, and in situations where CPUs are idled (as in applications that must use all memory and less than all of the CPUs).

When the user sets the mask bits, the scheduling is called *hard affinity*, and a single set bit in the mask means that the process/thread is to be bound to a single CPU. It is fairly simple to assign N processes/threads to N CPUs on a node; but it is a bit more complicated to determine which logical CPUs are assigned to cores on a single processor. The core-to-socket assignments were determined from similar absolute RDTSC values, and is not discussed here. Measurements in the sections 4 and 5 use *hard affinity*, in the Pthread kernels and OpenMP codes.

### 4. Synchronization

At the OS level, thread management for the kernel hasn't changed in complexity with the introduction of multi-core systems because a core to the OS appears and behaves as just another logical CPU. However, the new structure of the memory system has consequences on the cost of thread operations (team formation, locks, etc.); importantly, it can have an impact on temporal characteristics when multiple cores reuse the

same data. Two important concerns are the ability (and capability) of cores to share data from the same cache, and to avoid false sharing whenever cache lines cannot be shared from the same L2 cache. The latter occurs in independent (non-shared) caches, but can often be controlled by the data structure in programs (preventing threads from sharing the same cache line). The former seems to be a natural benefit that will manifest itself as an additional performance gain, automatically. However, even the processes or threads of a highly balanced (data parallel) algorithm can easily become out of step with each other, and inhibit what might seem to be an assured benefit.

In large, shared-memory systems OpenMP is often used as the parallel paradigm for scientific applications. Of course, MPI applications are also used because they can run on both distributed and shared memory systems. Because of its “finer” granularity of parallelism, one’s first choice for optimizing Core Level Parallelism in HPC codes is with OpenMP threads. Utilizing Pthreads is also a viable, albeit much harder, approach to manage threads in an HPC parallel application. Most of the measurements in this work were performed with Pthreads to provide better control at a lower level of synchronization. OpenMP programming has higher overheads, must form teams of threads for well defined parallel regions with startup costs over 200,000 CPs, but provides a compiler directive-based interface for scheduling threads, distributing work and scoping variables. For the latter reason, however, OpenMP remains the preferred paradigm for parallel HPC applications.

The cost of team synchronization scales with the team size. The scaling is also affected by the number of cores occupied per processor. Figure 2 shows the costs of synchronizing through a barrier, and the scaling for different core occupations of the Opteron (PathScale compiler, OpenMP, SuSE OS) and Woodcrest (Intel compiler, OpenMP, RedHat OS). For the AMD 4-socket evaluation system, the costs for single-core/socket versus dual-core/socket barriers were measured up to 8 cores; only 4 cores were available in the Intel 2-socket evaluation system. For the Opteron system, scaling depends only on the number of threads, and not the core occupation per processor. With only a single data point and some caution, we conclude from Figure 2(b) that the Woodcrest system synchronization cost is dependent upon the core occupation—on-socket synchronization is faster.

Note, that the cost for a barrier between two threads in Figure 2 is about 2000 CPs, without startup overhead. With Pthreads, the cost to create the first thread is ~250,000/175,000 CPs (Woodcrest/Opteron) with only a ~35,000/35,000 CP cost for each subsequent thread. The cost for synchronizing two threads using a volatile C variable as a semaphore and two spin-wait loops is ~1000/1000 CPs.

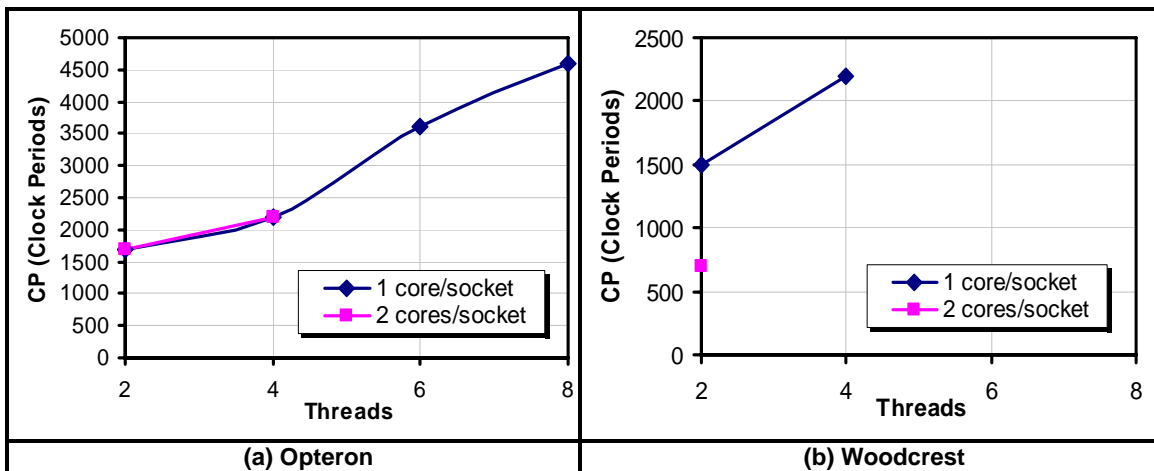


Figure 2: OMP Barrier performance comparisons for 1 and 2 cores/socket: (a) Opteron measurements using the PathScale compiler, (b) Woodcrest measurements using the Intel compiler.

## 5. L2 Cache Characteristics

The Intel Woodcrest uses a shared cache (cf. Figure 1). In the context of cache architecture design, a shared L2 cache means that multiple cores are directly connected to it (accessible by load and store operations). (For both systems the L2 caches are unified, in that both data and instructions share the L2 cache.) Independent caches, as employed in the AMD Opteron multi-core systems, are described by AMD as an “Individual L2 cache per core” [4]. Any core that needs data in another core’s L2 cache must access it through external memory in these systems. The cache configurations for the Opteron and Intel Systems used here are listed in Table 1.

While size is important for capacity, more dynamic features such as throughput, sharing, associativity and TLB coverage have a significant impact on performance. In the following subsections we will investigate these features.

### Latencies

Figures 3 and 4 show the latencies for the Intel Woodcrest and AMD Opteron systems, using small (default, 4KB) and large (HugeTLB, 2MB) pages for the allocated arrays. In these measurements, the number of cycles (clock periods) is recorded for reading a number of elements from an array using a vector stride of 64, 128, 256, 512, 1024, 2048, and 4096 bytes. Because the strides are longer than a single cache line (64B), the times constitute the latency, as opposed to throughput when sequential elements (within a cache line) are accessed. Larger strides use a larger number of TLB entries.

In Figure 3 the three regions of the small-page latencies represent the L1 cache, L2 cache, and memory accesses for the Woodcrest system. The latencies are 3, 14-16 and 313-320 CPs respectively, and memory latencies are within 10% of their asymptotic value (not shown). The lowest memory curve uses a stride of 64B (a single cache line) and hence solicits the hardware prefetch mechanism. There is an apparent increase in the L2 latency beyond the 1MB L2 cache range, up to the 4MB L2 cache size. This is due to a 256 entry limit of the Translation Lookaside Buffer (TLB). More specifically, the coverage of the L2 area is only 256 entries x 4096B (page size) = 1MB. Hence, any type of access (including indirect and random) that spans more than 256 pages will see a slight degradation within this cache system. The large-page curves are similar to the small-page curves, but the L2 coverage is complete and no degradation occurs for the range of strides. (However, there are only 8 and 32 entries in the table for large pages for the Opteron and Woodcrest, respectively; and accesses using more than the allotted pages will exhibit irregular degradation in latency.) Another feature of the large pages is a sharper transition from the L2 to the memory latency region, with an extension of the effective L2 cache range closer to the real cache size of 4MB.

The latency curves for the Opteron in Figure 4 are similar to the Woodcrest, but have more structure. In general the latencies are 3, 12-14 and 143-150 CPs for the L1 cache, L2 cache, and memory. The small-page and large-page L2 regions have no coverage degradation, because there are 512 entries in the TLB table (coverage = 512 entries x 4096B = 2MB). Also, the transition between L2 and memory is sharper and extends to higher L2 cache values for large pages, just as for the Woodcrest system. Because of the large amount of structure in the figures it makes sense to first look at the simpler large-page curves. First, most of the strided access curves in the L2 region are superimposed on the lowest level (15CPs), and have an asymptote of 145 CPs. The sequential access of lines (stride 64) requires 20 CPs in the L2 region, but only 57 CPs in the memory region. The prefetching hardware hides the latency in the memory section; the cause for the other higher L2 cache accesses is unknown, but could be due to engagement of hardware prefetching. The other anomalous feature is a higher latency for strides of 4096 throughout the L2 cache and memory regions. The cause of the oscillations is unknown at this time.

With an understanding of the large-page results, the differences between the curves in the small- and large-page allocated memories are now a bit more discernable. The sequential accesses are essentially the same, except for the expected effective range extension and sharper cutoff. Handling the small pages generally incurs a 5CP overhead penalty in the L2 region (for strides between 128B and less than a page,

4096B). In the memory region there is a significant and varied increase in latency for using a larger number of pages beyond the 2MB coverage.

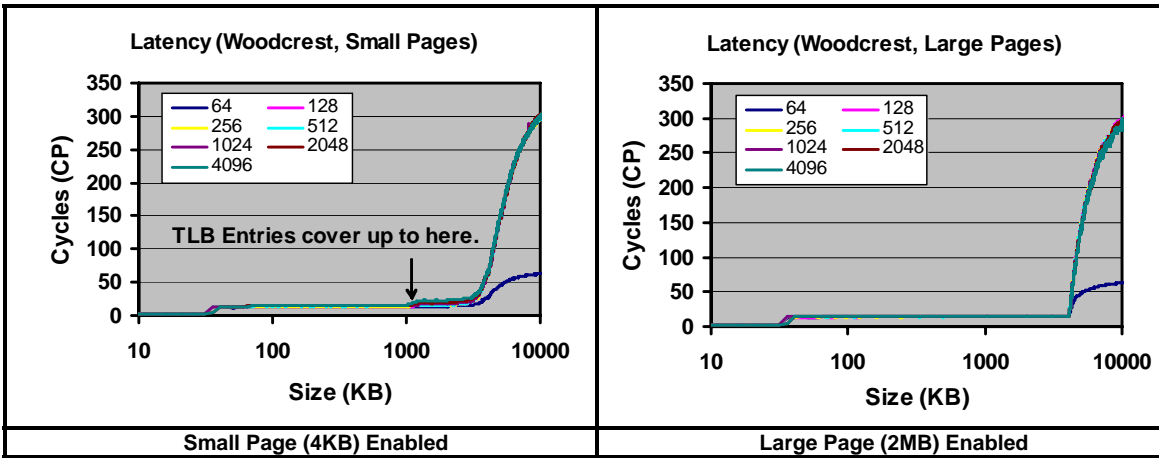


Figure 3: L1 cache, L2 cache, and memory latencies for Woodcrest system, using strided accesses between 64 B (one cache line) and 4096 B. Small-page and large-page (HugeTLB) allocations are shown in left and right plots, respectively.

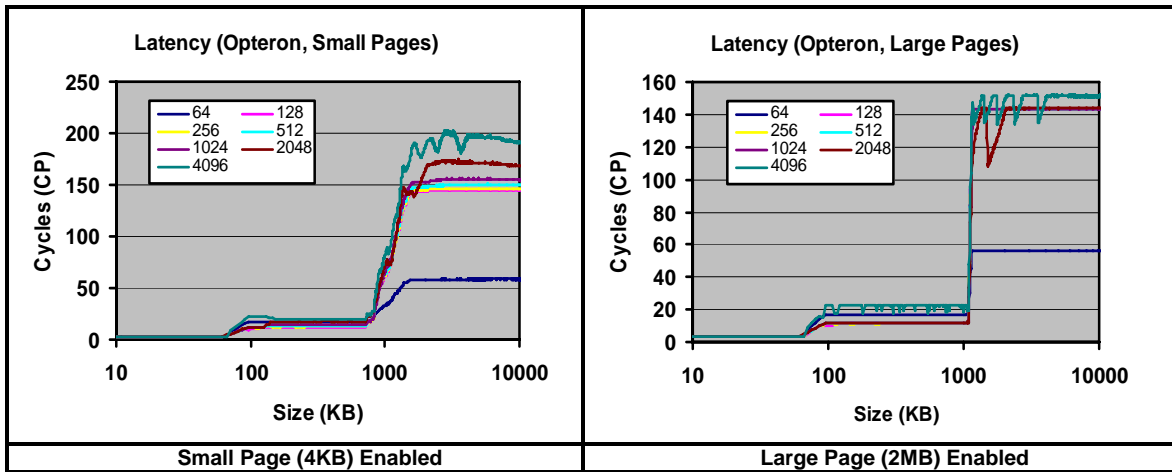


Figure 4: L1 cache, L2 cache, and memory latencies for Opteron system, using strided accesses between 64 B (one cache line) and 4096 B. Small-page and large-page (HugeTLB) allocations are shown in top and bottom plots, respectively.

### Simultaneous Core Operations

Several measurements were performed to assess simultaneous reads and writes in the L2 Cache(s). The metric used in these measurements is efficiency, which is defined here to be “words (read or written)/CP relative to the peak capacity of 1 word/CP”. Efficiency is defined relative to 1 word/CP to make it unitless.

Table 3A shows the efficiency for reading 500KB of data out of the L2 cache using two methods: **readdata**, an assembler routine (compiled with gcc); and **sum**, a C code compiled with hardware and -O3 optimization (the PathScale compiler was used for the Opteron, and the Intel compiler for the Woodcrest). The assembler code should show an upper bound for reading efficiency.

On the Woodcrest a single-core **readdata** reads from the L2 cache with an efficiency of 0.96; whereas when both cores read from the shared cache the efficiency of each core is only 0.85. For **sum**, the corresponding values are 0.83 and 0.84. Indeed, as expected, the assembler read algorithm appears to be at the upper bound of reading efficiency—a value that libraries may obtain; and for the C compiled code, and possibly Fortran compiled code, the read efficiencies are the same, and poorer than when contention is observed for the assembler code. Since the Opteron cores have independent caches for each core, there should be no difference when a single core or two cores of a socket are reading. Indeed this is the case for the Opteron; but the **readdata** and **sum** efficiencies are significantly less than those found in the Woodcrest.

Table 3B shows the efficiency for first writing to the L2 cache, followed by reading the same data. For the Woodcrest a write, followed by a read, delivers efficiency values of 0.82 and 0.96, respectively, for a single core. When two cores on a single socket write and then read from the shared cache, the write efficiency varies between 0.50 and 0.65; and the read efficiency is 0.83, as expected (see Table 3A.). For the Opteron, the single and dual occupation writes and reads produce identical efficiencies of 0.49 and 0.54 for the write and read operations, respectively.

Table 3C shows a measurement of the cost for sharing data between the independent L2 caches of Opteron cores. In this “cross read” experiment core1 and core2 each write to their L2 cache, then core1 reads data from core2’s L2 cache, and vice versa. (The write and read are each synchronized.). The write efficiency is 0.49 for each core, and 0.095 for the read operations. The latter shows that L2 data is shared through memory. The same experiment was performed on a single core in two different sockets. The write and read efficiencies are 0.49 and 0.104; hence L2 cache data sharing across sockets is only slightly higher than on socket for the Opteron.

**Table 3: L2 Cache Read/Write Efficiencies: (A) Synchronized, one- and two-core reads from L2 using assembler loads (readdata) and C-code (with sum operation); (B) Similar to Test A, but synchronized read is preceded by synchronized write; (C) Synchronized write followed by “cross” read, core reads data written by other core.**

<b>Test A: Synchronized Reads</b>			
<b>System</b>	<b>Cores Per Socket</b>	<b>Read:Assembler</b>	<b>Read:C code</b>
Woodcrest	1	0.96	0.83
	2	0.85	0.84
Opteron	1	0.54	0.44
	2	0.54	0.44
<b>Test B: Synchronized Read Preceded by Synchronized Write</b>			
<b>System</b>	<b>Cores Per Socket</b>	<b>Write: Assembler</b>	<b>Read: Assembler</b>
Woodcrest	1	0.82	0.96
	2	0.50-0.65	0.83
Opteron	1	0.49	0.54
	2	0.49	0.54
<b>Test C: Synchronized Write Followed by Cross Read</b>			
<b>System</b>	<b>Cores Per Socket</b>	<b>Write: Assembler</b>	<b>Cross Read: Assembler</b>
Opteron	1	0.49	0.104
	2	0.49	0.095

## 6. Application Benchmarks

### Introduction

Within the next several years both library and application developers will re-optimize programs and algorithms to accommodate new L2 cache architectures for multiple cores. Advanced (dynamic) scheduling techniques [5] by compiler and library developers [6] will refine thread (re)assignments; and new temporal and locality paradigm patterns of data movement will be devised. However, many

applications will still use standard parallel paradigms and algorithms for solving problems. In this section we evaluate the performance and scaling for the NPB benchmarks. While the multi-core aspects of the scaling are very limited (scaling from 1 to 2 cores per socket), they do give an indication of what is to come with more cores per socket. Multi-Core Linpack HPCL and SPEC® CPU2000 benchmark measurements on the Opteron have been reported elsewhere<sup>7</sup>.

### NPB MPI Benchmarks

Four representative NPB benchmarks, EP, IS, LU and CG [8] were run as parallel MPI executions on the Woodcrest and Opteron systems. It is to be noted that all of the experiments for application performance for MPI measurements were carried out on a single node; that is, no external interconnect was used.

For the Woodcrest system a 2-task execution with a task on a single core in each socket was run first. Next the same benchmark was run with 4 tasks, one in each core.

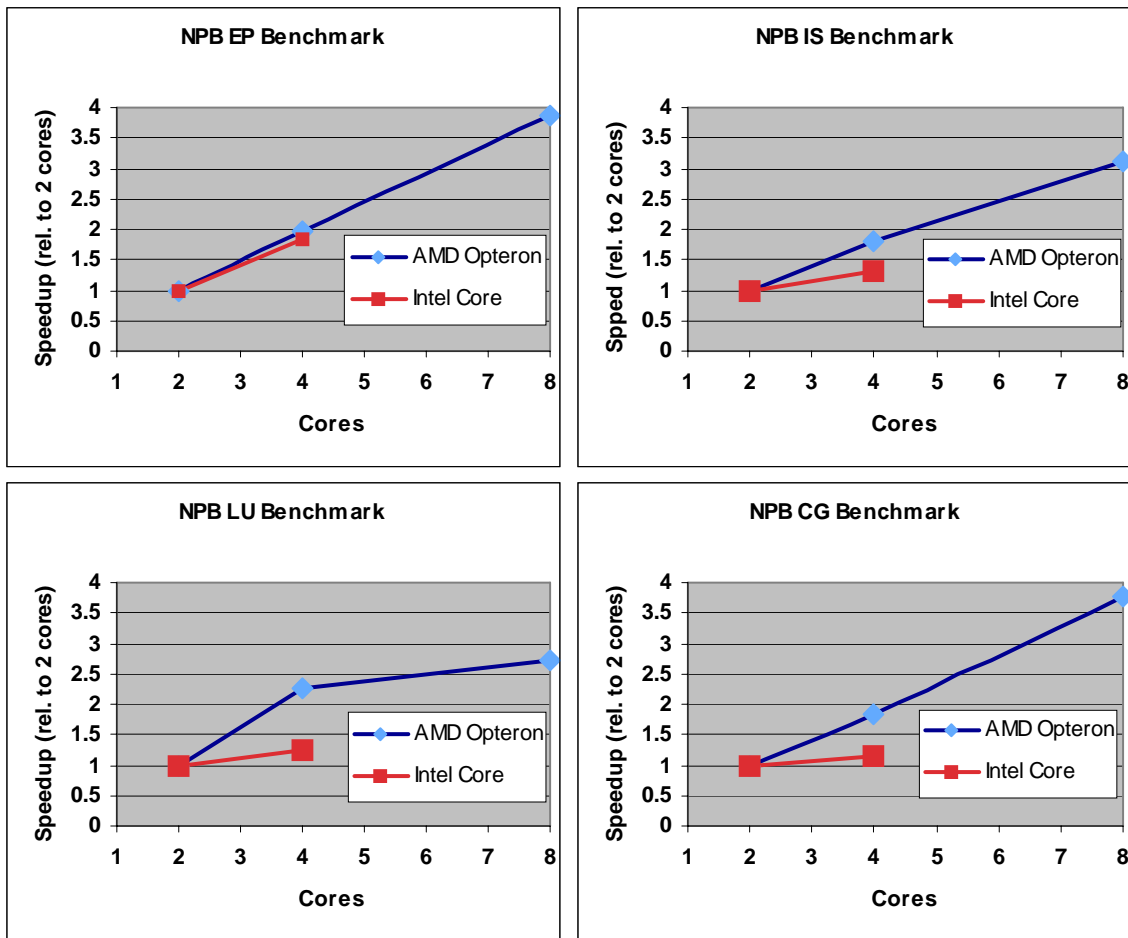


Figure 5: NPB EP, IS, LU and CG benchmark scaling. Woodcrest 2- and 4-task executions use 1 and 2 cores per socket, respectively. Opteron 2-, 4- and 8-task executions use 1, 1, and 2 cores per socket, respective. Scaling is relative to 2-task times.

For the Opteron system a 2-task execution with a task on a single core in two sockets was run first, in order to compare timings of the same measurement on the Woodcrest system. Next the benchmarks were run with 4-tasks on 4-socket. In the final measurement, 8 tasks were employed with each core on a socket occupied by a task.

Figure 5 shows the relative scaling. The 2- to 4-core line for the Woodcrest represents scaling from 2 to 4 cores with a concurrent change from single- to double-core occupation on a socket. Likewise the 4- and 8-core line for the Opteron represents a core count and occupation doubling. The “core scaling” (in changing from 1- to 2-core occupation on a socket) for the EP and LU benchmarks are similar for the Opteron and Woodcrest. The EP benchmark scales perfectly, as expected; while the LU factorization scale factor is only 1.3. For the Integer Sort (IS) and Conjugate Gradient (CG) benchmarks the Opteron scaling is significantly better, 1.8 compared to 1.3 for the Intel IS, and 2.0 compared to 1.2 for the Intel CG. Apart from the scaling issues, the IS benchmark is not floating-point intensive and consequently do not benefit from the advanced FPU units on the Woodcrest. This is reflected in the absolute performance (results not shown) for all cases where the Opteron system “bests” the Woodcrest systems for this case. The LU benchmark is the most CPU and memory intensive application. Consequently, the advanced FPU unit on the Woodcrest system plays a positive role for all cases. The performance difference is more significant in the two-task case than the four-task case, where the better memory utilization for the Opteron system narrows the performance gap. The CG benchmark relies more heavily on the unstructured matrix-vector multiplication and thus the Woodcrest performs best when there is least stress on the memory subsystem for the two-task case and as this stress increases in the four-task case, the Opteron system performs best.

### **NPB OpenMP Benchmarks**

Four class B NPB benchmarks, CG, EP, IS and LU were run as parallel OpenMP jobs on the Woodcrest and Opteron systems. For the Woodcrest system, three different topologies using 2 and 4 threads were run for each NPB kernel. The topologies 11, 20, and 22 refer to the thread occupation on the 2 sockets. Next, the same benchmarks were run with 4 and 8 threads on the Opteron system. The topologies for the 4 sockets were 1111 and 2200 for the 4-thread runs and 2222 for the 8-thread run.

The benchmark results in Table 4 and

Table 5 give an indication of OpenMP performance on a fully-loaded socket (2 cores occupied per socket) versus a half-populated socket workload (1 core occupied per socket) in the outer columns. Performance differences between a fully-loaded system and a half-populated system with 2-core occupation are given in the two right-hand columns. With the exception of IS, the numbers reported are MFLOPS/s per thread. This metric was adopted for a fair comparison across different thread counts. For the IS benchmark, the numbers correspond to millions of integers ranked per second per thread.

The LU benchmark solves a seven-block-diagonal system using the SSOR method, which performs an LU decomposition of the matrix. The OpenMP implementation of LU uses the pipelining algorithm for performing the computation in parallel. A shared data array (of dimension equal to the thread count) is used for synchronization among threads, and is used to indicate availability of matrix elements from neighboring threads for computation by the current thread. On both the Intel and AMD systems, there is decreased LU performance at full versus half-socket occupation when the system is half-loaded. The matrix size corresponding to the class B benchmark is 102x102x102, which occupies slightly above 8MB of memory. This data array does not entirely fit into the L2 cache on one socket but can be accommodated when spread out across two sockets. On the Woodcrest system, the nearly 2x performance boost in going from full- to half-socket occupation in the dual-threaded execution of LU could be attributed to the effective doubling of the cache size. The corresponding performance increase on the Opteron is less dramatic because the cache size remains the same, although the doubling of the available bandwidth from memory certainly helps.

**Table 4: Average MFLOPS per thread for single- and double-occupation of cores on AMD Opteron. See text for topology notation.**

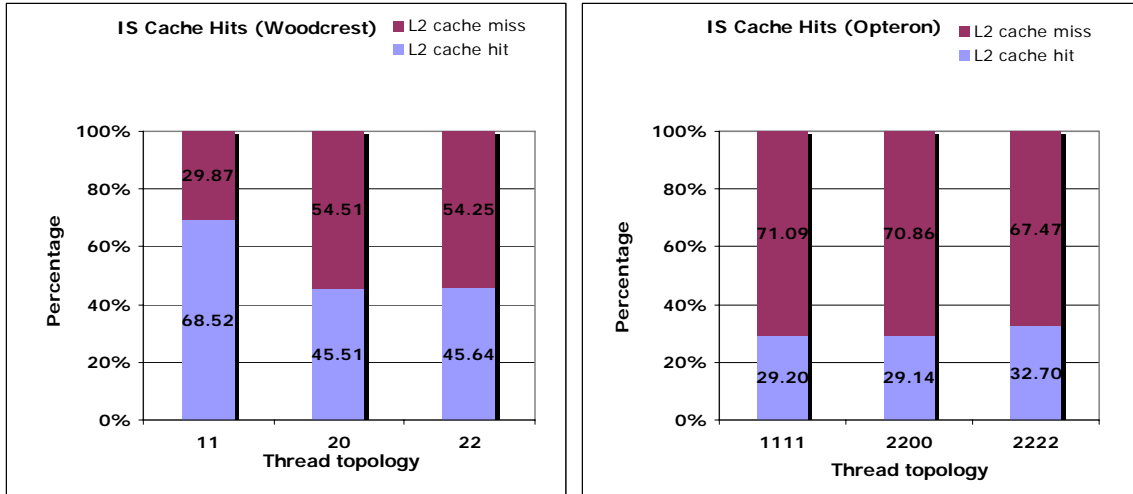
NPB Benchmark	1111-topology	2200-topology	2222-topology
CG	118.695	106.02	86.41
EP	25.3	25.995	25.885
IS	27.95	13.70	12.25
LU	772	581.33	498.24

**Table 5: Average MFLOPS per thread for single- and double-occupation of cores on Intel Woodcrest. See text for topology notation.**

NPB Benchmark	11-topology	20-topology	22-topology
CG	380.28	277.975	239.44
EP	33.98	33.98	33.78
IS	59.19	23.10	18.44
LU	1059.07	630.145	837.53

The CG benchmark performs parallel sparse matrix-vector multiplication and separate reduction operations for each conjugate gradient iteration. Except for the reduction operations, each CG iteration involves irregular memory access. When the system is half-loaded, the CG results corresponding to fully-populated sockets on the Woodcrest and Opteron exhibit worse performance than the half-populated cases. Since there is little data sharing among threads and most of the data access is indirect, application performance is enhanced in the half-populated socket runs since each thread has exclusive access to the memory controller.

The IS benchmark ranks  $2^{25}$  integers, which occupy 128MB of memory not counting the work arrays. Sorting involves irregular access of large data structures, although unit stride access to data is also performed. Lower memory latency, higher bandwidth and larger cache sizes will enhance IS performance because of the indirect addressing and (to a lesser degree) its ability to benefit from data locality. The speedup observed in going from the (20) to (11) thread topology on the Woodcrest may be attributed to a doubling of the memory bandwidth and effective cache size per thread. PAPI measurements of the number of L2 cache hits, misses and total access provide evidence for the latter possibility (cf. left bar chart in Figure 6). The L2 cache hit rate at half socket occupation is 69% versus 45% when the socket is fully populated. From (20) to (22), there is a slight decrease in performance caused by additional thread overhead, load imbalance and synchronization. A similar performance scaling is observed when comparing (1111) versus (2200) Mops/s/thread numbers on the AMD Opteron. An increase in cache efficiency, however, is not observed from the PAPI L2 cache measurements on the AMD Opteron shown in Figure 6. This invariance is not unexpected as the cache size per thread remains the same independent of thread scheduling on the sockets. Since the cache efficiency remains fairly constant regardless of socket occupation and the latency of accessing data from memory is independent of thread scheduling on the sockets as well, the performance advantage of the (1111) versus (2200) thread topology may be attributed to the increased memory bandwidth when only a single thread is assigned per socket. Both the (2222) and (2200) configurations have half the available per-thread memory bandwidth of (1111), which coincides closely with the observed 2x difference between the half versus full socket population Mop/s numbers. (2222) performance is slightly less than that observed for (2200) because of the higher synchronization and thread assembly cost with increasing thread count.



**Figure 6: PAPI L2 cache access measurement results for the class B OpenMP implementation of the NPB Integer Sort (IS) benchmark. Shown here are the percentage cache hit and miss rates for the Intel Woodcrest (left) and AMD Opteron system (right) for different thread counts and thread-to-socket mappings.**

Generally speaking, given a fixed thread count (two on the Woodcrest and four on the AMD), the Mops/s performance is best when the sockets are half populated (single-core occupation), except for the EP benchmark. On the AMD the performance degradation is less. Hence the occupation scaling is better with the independent caches for CG and LU; however, the Woodcrest absolute performance is better overall. (This may change significantly in the next Opteron quad-core system when the floating-point operations per clock period are doubled and L2 transfer performance is increased.) The embarrassingly-parallel EP results are constant, as expected.

## 7. Conclusions

The previous sections presented an overview of multi-core architectural implementations and implications for HPC programming. Measurements of basic thread operations were performed to help evaluate efficient Core Level Parallelism (CLP) design. In this work, two multi-core memory system designs were evaluated, one with an L2 cache shared among multiple cores, the other with independent caches for each core.

Application and library developers will be optimizing for CLP, but it is still uncertain which cache system, independent or shared, will be best. Shared systems will promote L2 cache data reuse among threads, but suffer from contention (even with just two threads accessing a single cache). Also, low level synchronization methods for multiple cores, which may use as much as 5,000 clock periods, will need to be implemented in thread control at the user-program level to assure locality of the data. Independent cache systems are contention free at the L2 cache level, and are probably more ideal for MPI codes, which employ non-shared paradigms. Of course, both systems are hindered by a single channel to the memory controller that must be shared with all cores on a socket. L2 cache latencies, and even memory latencies for the Woodcrest, are affected by TLB operations. By minimizing TLB lookups with large-page allocated memory, the effective cache size increases, and L2 latencies are decreased (in areas where the coverage may have been incomplete). While this paper reveals important aspects at the core level, there are still other issues to be investigated at the memory level which will also significantly impact applications as the number of cores per socket increases.

## References

---

- [1] [http://www-03.ibm.com/servers/eserver/pseries/hardware/whitepapers/p690\\_hpc.pdf](http://www-03.ibm.com/servers/eserver/pseries/hardware/whitepapers/p690_hpc.pdf)
- [2] <http://www.research.ibm.com/journal/rd/494/kahle.html>
- [3] <http://www-128.ibm.com/developerworks/linux/library/l-affinity.html?ca=dgr-lnxw09Affinity>
- [4] <http://developer.amd.com/articlex.jsp?id=28>
- [5] CASC: A Cache-Aware Scheduling Algorithm For Multithreaded Chip Multiprocessors, Alexandra Fedorova, Margo Seltzer, Michael D. Smith and Christopher Small, <http://research.sun.com/scalable/pubs/CASC.pdf>
- [6] SuperMatrix Out-of-Order Scheduling of Matrix Operations for SMP and Multi-Core Architectures, Ernie Chan, Enrique S. Quintana-Orti, Gregorio Quintana-Orti, Robert van de Geijn, (submitted to SPAA; The University of Texas at Austin, Department of Computer Sciences. Technical Report TR-06-67: <http://www.cs.utexas.edu/users/echan/supermatrix.pdf>).
- [7] A Comparison of Single-Core and Dual-Core Opteron Processor Performance for H, Douglas M. Pase and Matthew A. Eckl, IBM Publications, [ftp://ftp.software.ibm.com/eserver/benchmarks/wp\\_Dual\\_Core\\_072505.pdf](ftp://ftp.software.ibm.com/eserver/benchmarks/wp_Dual_Core_072505.pdf)
- [8] NAS parallel benchmarks. <http://www.nas.nasa.gov/Software/NPB/>