

# Using Large Page and Processor Binding to Optimize the Performance of OpenMP Scientific Applications on an IBM POWER5+ System

Xingfu Wu and Valerie Taylor

*Department of Computer Science & Engineering  
Texas A&M University, College Station, TX 77843*

*Email: {wuxf, taylor}@cse.tamu.edu*

## Abstract

*Multicores are widely used for high performance computing and are being configured in a hierarchical manner to compose a multicore system. While this presents significant new opportunities, such as high inter-core bandwidth and low inter-core latency, it also presents new challenges in the form of inter-core resource conflict and contention. A challenge to be addressed is how well current shared-memory parallel programming paradigms, such as OpenMP, exploit the potential offered by such a multicore system for scientific applications. In this paper, we analyze the performance of OpenMP scientific applications such as NAS parallel benchmarks, an OpenMP Matrix multiplication, and a large-scale scientific application: a 3D particle-in-cell application Gyrokinetic Toroidal Code (GTC) in magnetic fusion on an IBM POWER5+ system, and use large page and processor binding to significantly optimize the OpenMP performance for no requirement of any program modifications. Our experimental results show that using the large page of 64KB on the multicore system results in up to 33.92% performance improvement for OpenMP NAS parallel benchmarks, and the OpenMP benchmarks benefited more from the large page than their MPI counterparts; using the large page of 64KB and processor binding results in up to 67.18% performance improvement for matrix multiplications, and up to 10.13% performance improvement for the GTC. Our results also indicate that the OpenMP performance can be improved by using conventional loop optimization techniques such as blocking and unrolling inside the OpenMP parallel regions.*

## 1. Introduction

Today, multicores are widely used for high performance computing and are being configured hierarchically to form a multicore system. For example,

a multicore system - IBM POWER5+ system Hydra at Texas A&M University Supercomputer Facility [TSCF] consists of nodes that have 8 DCMs (Dual-Chip Modules) with a dual-core POWER5+ processor per DCM. While multicore presents significant new opportunities such as on-chip high inter-core bandwidth and low latency, it also presents new challenges in the form of inter-core resource conflict and contention. In [OH07, LL07], it is argued that the full benefit of these architectures will not be harnessed until the software industry and community fully embrace parallel programming.

OpenMP [OMP, PR08] is the most popular shared memory parallel programming paradigm. Can OpenMP parallel programming paradigm efficiently exploit the potential offered by such multicore system for scientific applications? In this paper, we systematically analyze the performance of OpenMP scientific applications such as NAS parallel benchmarks, an OpenMP Matrix multiplication, and a large-scale scientific application: a 3D particle-in-cell application Gyrokinetic Toroidal Code (GTC) in magnetic fusion [ES05] on the multicore system Hydra which has 16 cores per node, and use large page and processor binding to optimize their performance. The standard OpenMP programming paradigms of NAS Parallel Benchmarks (NPB) Version 3.2.1 [NPB3] make it possible for us to use these benchmarks as a basis for a systematically comparative performance analysis of OpenMP programs on the multicore system. In addition to using these benchmarks, we also use a large-scale scientific application GTC, and develop an OpenMP matrix multiplication to analyze and optimize the performance of the OpenMP programs.

Our experimental results indicate that, using the large page of 64KB on Hydra results in up to 33.92% performance improvement for OpenMP NPB benchmarks, and the OpenMP benchmarks benefited more from the large page than their MPI counterparts; using the large page of 64KB and processor binding results in up to 67.18% performance improvement for

matrix multiplications, and up to 10.13% performance improvement for the GTC. Our results also indicate that the OpenMP performance can be improved by using conventional loop optimization techniques such as blocking and unrolling inside the OpenMP parallel regions. However, whether the OpenMP parallel programming is the most suitable for multicore systems or not depends on the nature of a scientific application, available parallel programming software, and compiler support on these multicore systems.

Hepkin [HE06] gave an overview of the IBM support for two new virtual memory page size – 64KB and 16GB for the POWER5+ processor and AIX 5L version 5.3, and how a user can use these new page sizes for an application’s memory to potentially improve performance. But only 64KB pages are general-purpose for users. Winwood, Shuf and Franke [WS02] discussed the ongoing modifications to the Linux kernel to allow applications to vary the size of pages used to map their address spaces and to reap the performance benefits associated with the use of large pages, and used sequential SPEC benchmarks to demonstrate the performance improvements.

Kleikamp and Pulavarty [KP06] proposed a change to the Linux kernel to allow file data to be more efficiently stored in memory when the size of the file or the data at the end of a file is significantly smaller than the page of 64KB. In our previous work [WT09], we found that processor binding resulted in up to 7.16% performance improvements for MPI scientific applications. In contrast to the approaches, we focus on exploring how OpenMP scientific applications really benefit from using large pages and processor binding on the multicore system.

The remainder of this paper is organized as follows. Section 2 discusses the architecture and memory hierarchy of the multicore system Hydra used in our experiments. Section 3 analyzes and optimizes the performance of OpenMP programs using NPB 3.2.1, and presents how a large page impacts the performance of MPI and OpenMP programs. Section 4 develops an OpenMP matrix multiplication, investigates its performance and scalability on the multicore system, and discusses how a large page, processor binding and loop blocking impact its performance. Section 5 investigates performance and scalability of the OpenMP GTC on the multicore system, and discusses how a large page, processor binding and loop blocking and unrolling impact its performance. Section 6 concludes this paper.

In the remainder of this paper, we assume that the job scheduler for the multicore system always dispatches one thread to one core. All experiments were executed multiple times to insure consistency of the performance data. Prophesy system [TW03] is used to collect application performance data.

## 2. Execution Platform

Details about the multicore system Hydra used for our experiments are given in Table 1. Hydra at Texas A&M University Supercomputer Facility [TSCF] is an IBM POWER5+ cluster with 40 p5-575 nodes, and each node has 32 GB of memory and 8 DCMs (Dual-Chip Modules) with a dual-core POWER5+ processor and one L3 chip per DCM shown in Figure 1, which is the high level structure of an IBM POWER5+ chip (light green portion). Two processor cores share L2 cache. Hydra has the default page of 4KB and IBM AIX 5.3, and it supports user-level large page of 64KB using the *ldedit* or *ld* commands [HE06]. The large page uses hardware prefetch mechanisms to eliminate costly TLB misses at the expense of an increase in process start-up time. The SMT (Simultaneous Multi-Threading) mode is not enabled for regular use in the system.

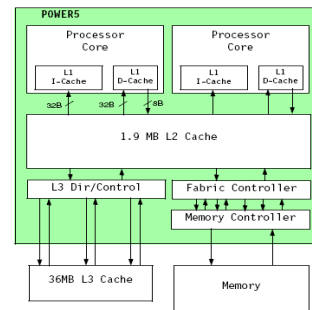


Figure 1. High-level POWER5+ [GA05]

Table 1. Specification of the multicore system Hydra

Total Nodes	40
Cores/chip	2
Cores / Node	16
CPU type	1.9GHz IBM POWER5+
Memory/Node	32GB
L1 Cache/CPU	64/32 KB
L2 Cache/chip	1.92MB
L3 Cache/chip	36MB

The processor affinity policy for POWER5 is discussed in [GA05]. When a virtual processor is dispatched, it is first dispatched onto the same physical processor that it last ran on. Otherwise, it will be dispatched onto the first available processor in the following order: on the same chip, then to another chip on the same multi-chip module (MCM), then to a chip on the same node. The IBM AIX operating system provides the command *bindprocessor* to bind a process to a physical processor, and provides the environment variable *XLSMPOPTS* to bind a thread to a physical processor. The goal for using processor binding is to reduce on-chip inter-core resource conflict and contention.

### 3. Performance Analysis and Optimization of NPB Benchmarks

In this section, we use the standard NPB benchmarks (MPI and OpenMP) (NPB version 3.2.1) with problem size of Class B to analyze and optimize the performance of OpenMP NPB using large page on the multicore system Hydra. NPB has 8 benchmarks (5 kernel benchmarks: CG, EP, FT, IS, and MG; 3 application benchmarks: BT, LU, and SP). All OpenMP benchmarks are executed on a power-of-two number of cores.

Figure 2 shows the scalability of OpenMP NPB on Hydra except the benchmark IS because the OpenMP benchmark IS did not work on Hydra. This indicates that the OpenMP implementations of NPB with Class B all scale well on Hydra.

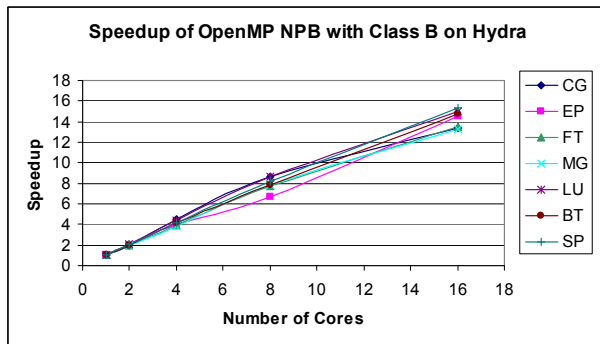


Figure 2. Scalability of OpenMP NPB with Class B on Hydra

As we know, using large virtual memory pages for a scientific application's memory can significantly improve the application's performance and throughput due to hardware efficiencies associated with large pages [HE06]. In this section, we discuss how large pages impact the performance of OpenMP programs from NPB benchmarks.

4KB is the default page for the multicore system Hydra. Hydra supports a user-level large page with the size of 64KB, which means no system-level configuration changes are necessary to enable Hydra to use 64KB pages. We use the *ldeit* command to set the page size to 64KB in all existing executables of NPB OpenMP benchmarks, then re-run these executables on Hydra. Figure 3 presents the performance improvement percentages for OpenMP NPB benchmarks. The performance improvement percentage is up to 33.92% for OpenMP NPB benchmarks. Figure 4 presents the performance improvement percentages for MPI NPB benchmarks. The performance improvement percentage is up to 31.17% for the MPI NPB benchmarks. Comparing Figure 3 with Figure 4, we find that OpenMP benchmarks benefit more from the large page

of 64KB than their MPI counterparts because large pages of 64KB result in reducing the overhead of translating a program page address to a memory page address (reducing TLB miss rate and L1 cache miss).

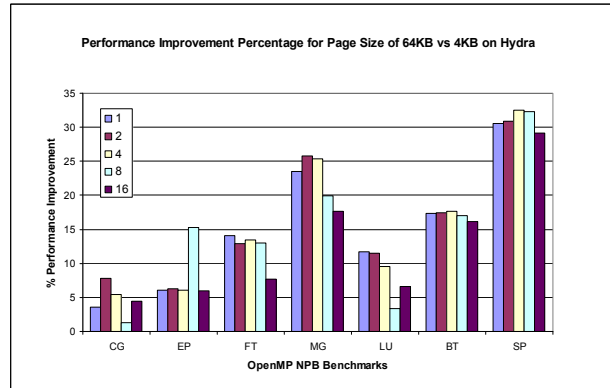


Figure 3. OpenMP performance improvement percentage on Hydra using large page of 64KB vs default page of 4KB (for Class B)

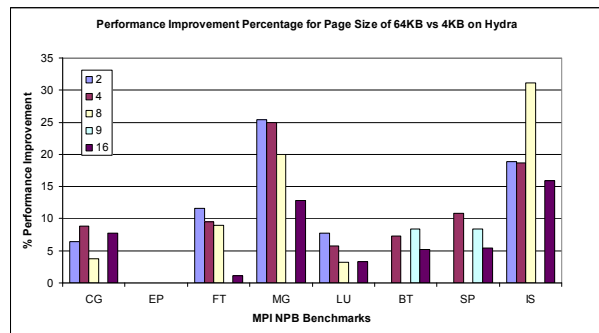


Figure 4. MPI performance improvement percentage on Hydra using large page of 64KB vs default page of 4KB (for Class B)

Further, as described in [BB94] for MPI implementations, FT is a 3D partial differential equation solution using FFTs, and it is a rigorous test of long-distance communication performance; MG is a simplified multigrid kernel, and requires highly structured long distance communication and tests both short and long distance data communication; LU does not perform a LU factorization but instead employs a symmetric successive over-relaxation (SSOR) numerical scheme to solve a regular sparse, block (5x5) lower and upper triangular system, and it is very sensitive to the small-message (40 bytes) communication performance and sends large number of very small (40 bytes) messages. So the three benchmarks focus on testing MPI communication performance. OpenMP implementations of FT, MG and LU try to reduce the communication overhead of MPI at the expense of introducing OpenMP overhead due to thread creation and increased memory bandwidth

contention to take advantage of on-chip inter-core high bandwidth and low latency provided by multicore. And as described in [JF99], the OpenMP implementation of FT also eliminates a 3D data array which was needed in the MPI version; the OpenMP implementation of BT includes touching the data pages by inserting initialization loops in the beginning of the program in favor of the x and y solver, and several end-of-loop synchronizations have been removed by the use of “!\$omp end do nowait”; the OpenMP implementation of SP is the similar to that of OpenMP BT. These changes have improved the memory utilization such that OpenMP NPB benchmarks benefit more from the large page of 64KB than their MPI counterparts.

#### 4. Performance Analysis and Optimization of an OpenMP Matrix Multiplication

In this section, we use OpenMP to parallelize the commonly used matrix multiplication  $C[L, N]=A[L, M] \times B[M, N]$ , and use large page and processor binding to optimize its performance. We also find that in the OpenMP parallel region, using the traditional loop optimization to optimize the matrix multiplication triple-nested loop can achieve the better performance by taking advantage of memory hierarchy on the multicore system.

##### 4.1 Using Loop Blocking to optimize the double-nested loop inside OpenMP parallel region

As you know, a simple OpenMP matrix multiplication triple-nested loop looks like:

```
!$omp parallel default(shared) private(i,j,k)
.....
!$omp do
do i=1,L
do j=1,N
do k=1,M
c(i,j) = c(i,j) + a(i,k) * b(k,j)
enddo
enddo
enddo
!$omp end do
!$omp end parallel
```

Where we set  $L=M=N=2000$  so that any-level cache of the multicore system cannot hold two  $2000 \times 2000$  matrixes with real8 elements. In this simple example, the outer-most loop ( $do i=1,L$ ) is parallelized with the “!\$omp do” directive. As you know, conventional loop optimization techniques can improve the performance. So we use loop blocking to optimize the inner double-nested loop inside the OpenMP parallel region as follows:

```
!$omp parallel default(shared) private(i,j,k,jj,kk)
.....
!$omp do
do i=1,L
do jj=1,N, NB
do kk=1,M, NB
do j=jj, MIN(N,jj+NB-1)
do k=kk, MIN(M,kk+NB-1)
c(i,j) = c(i,j) + a(i,k) * b(k,j)
enddo
enddo
enddo
enddo
!$omp end do
!$omp end parallel
```

Where we find the optimal  $NB=4$  is for the POWER5+ [GA05]. We use the simple example to demonstrate how much improvement we can obtain for different optimization techniques.

##### 4.2 Performance Analysis and Optimization

In this section, we use the two OpenMP matrix multiplications described in Section 4.1 as strong scaling examples to analyze and optimize their performance. Figure 5 shows scalability of the OpenMP matrix multiplications on Hydra, where “Original” stands for the performance for the original OpenMP matrix multiplication, “Blocking” stands for the optimized matrix multiplication using loop blocking, and “64KB” stands for using the large page of 64KB to optimize the performance of the original matrix multiplication. This figure indicates that our OpenMP implementations of matrix multiplication scale almost linearly, especially for using loop blocking because this technique transforms the memory domain of an application into smaller chunks, such that computations are executed on the chunks that easily fit into cache to maximize data reuse.

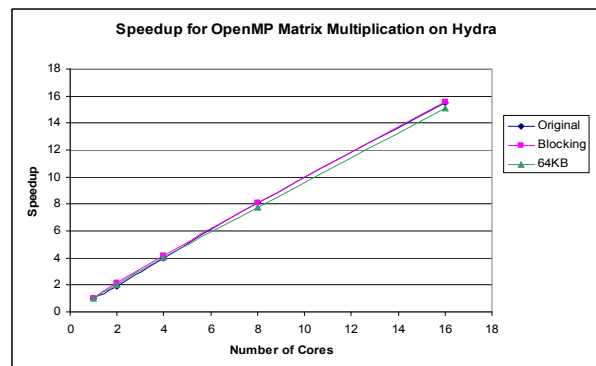


Figure 5. Scalability of OpenMP matrix multiplications on Hydra

Figure 6 presents the performance comparisons for OpenMP matrix multiplications on Hydra. Compared to

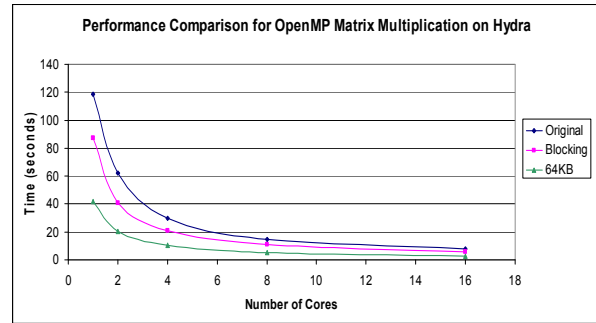
the performance for the original matrix multiplication, using the large page of 64KB achieves much better performance than using loop blocking because the large page uses hardware prefetch mechanisms to eliminate costly TLB misses shown in Table 2. Table 3 indicates the performance improvement percentages using different optimization techniques. Using loop blocking results in between 26.58% and 34.84% performance improvement. Using the large page of 64KB results in between 63.19% and 67.05% performance improvement. Using the large page of 64KB and processor binding can also achieve between 63.60% and 67.18% performance improvement. These are very big performance improvement.

**Table 2. L1 cache miss rate and TLB miss rate for matrix multiplications using 2 OpenMP threads**

Methods	L1 miss (%)	TLB miss (%)
Original	3.68	0.962
Blocking	2.59	0.329
64KB	1.06	3.80E-05

The goal for using processor binding is to reduce contention of chip resources. For the IBM POWER5+ system, each POWER5+ chip has two cores which shared L2 cache shown in Figure 1. For the default processor affinity policy, two threads are dispatched to

the same chip with one thread per core. Therefore, there is cache contention occurred in shared L2 cache. Setting the environment variable  $XLSMPOPTS=startproc=0:stride=2$  to bind each thread to a different chip can avoid the L2 cache conflict between two threads. This is useful when running 2, 4 or 8 threads on the 16-cores system. Our experimental results indicate that using one core per chip results in a decrease in execution time versus using two cores per chip because of reducing shared L2 cache contentions. While the difference is small, it is the case that processor binding can aid thread schedulers in maximizing the application performance in dedicated usage of the multicore node.



**Figure 6. Performance comparisons for OpenMP matrix multiplications on Hydra**

**Table 3. Performance improvement percentages of OpenMP matrix multiplications on Hydra**

#Threads	1	2	4	8	16
Blocking	26.70%	34.84%	29.99%	26.58%	27.02%
64KB	64.75%	67.05%	65.58%	63.19%	63.84%
64KB+Binding	--	67.18%	65.88%	63.60%	--

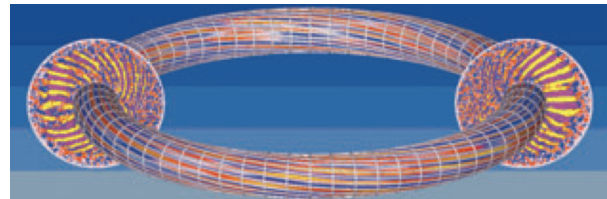
## 5. Performance Analysis and Optimization of a Scientific Application GTC

In this section, we use a hybrid MPI/OpenMP Gyrokinetic Toroidal Code (GTC) [ES05] to analyze and optimize the performance of the OpenMP GTC using large page and processor binding, and compare its performance with its MPI and hybrid counterparts. This is a weak scaling example.

### 5.1 Descriptions of Gyrokinetic Toroidal Code (GTC)

The Gyrokinetic Toroidal code (GTC) [ES05] is a 3D particle-in-cell application developed at the Princeton Plasma Physics Laboratory to study turbulent transport in magnetic fusion. GTC is currently the flagship SciDAC fusion microturbulence code written in Fortran90, MPI and OpenMP. Figure 6 shows a

visualization of potential contours of microturbulence for a magnetically confined plasma using GTC. The finger-like perturbations (streamers) stretch along the weak field side of the poloidal plane as they follow the magnetic field lines around the torus [SD06]. Figure 7 presents the basic steps in the GTC code.



**Figure 6. Potential contours of microturbulence for a magnetically confined plasma [SD06].**

The test case for GTC studied in this paper is 100 particles per cell and 100 time steps. The problem sizes for the GTC code are listed in Table 4, where micell is

the number of ions per grid cell, *mecell* is the number of electrons per grid cell, *mzetamax* is the total number of toroidal grid points, and *npartdom* is the number of particle domain partitions per toroidal domain.

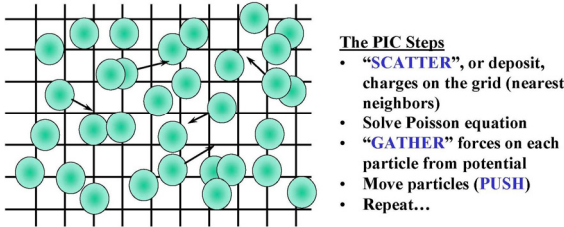


Figure 7. Particles in cell (PIC) steps [ES05]

Table 4. Datasets with scaling the number of processors

#Procs	1	2	4	8	16
<i>micell</i>	100	100	100	100	100
<i>mecell</i>	100	100	100	100	100
<i>mzetamax</i>	1	2	4	8	16
<i>npartdom</i>	1	1	1	1	1

## 5.2 Performance Analysis and Optimization

In this section, we use OpenMP GTC to analyze and optimize its performance using large page, loop optimization techniques and processor binding, and analyze how these techniques impact the performance of its MPI and hybrid counterparts.

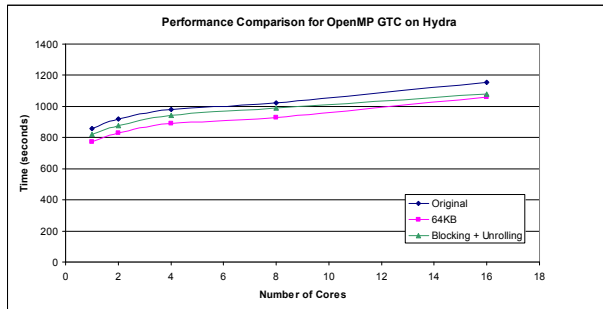


Figure 8. OpenMP performance comparisons for OpenMP GTC on Hydra

In the previous sections, using large page of 64KB results in that the performance improvement percentage is up to 33.92% for OpenMP NPB benchmarks, and up to 31.17% for MPI counterparts, and most than 60% for the OpenMP matrix multiplications on Hydra. So we apply the same techniques to the OpenMP GTC. Figure 8 presents the OpenMP performance comparisons for the OpenMP GTC on Hydra. “Original” stands for the performance for the original OpenMP GTC code. “64KB” stands for the performance of the original OpenMP GTC code using the large page of 64KB. “Blocking+Unrolling” stands for the performance for using loop blocking and unrolling inside OpenMP parallel regions to optimize the original GTC code. We find that the results for the GTC shown in Figure 8 are similar to that for the matrix multiplications in Figure 6. Using large page results in much better performance than using loop optimization techniques such as loop blocking and unrolling.

Table 5 shows the performance improvement percentages of the OpenMP GTC using different techniques. Using loop optimization techniques such as blocking and unrolling results in between 3.31% and 6.33% performance improvement, using the large page of 64KB results in between 7.90% and 9.67% performance improvement, and using the large pages of 64KB and processor binding results in between 8.97% and 10.13% performance improvement. These are big performance improvements.

Table 6 indicates that using large pages of 64KB also results in performance improvements for MPI and hybrid GTC on Hydra. Our experimental results also show that the OpenMP GTC benefits more from the large page of 64KB than its MPI counterparts. Recall that when the executable is executed, the text, data and bss sections are loaded into separate areas of virtual memory. Each virtual memory address is mapped by an operating system to a physical memory address in a page table. Using a large page of 64KB is to use the large page for each of the three regions (Data (data, bss), text and Stack) of a process’s address space. This results in reducing the overhead of translating a program page address to a physical memory page address (reducing the TLB miss and L1 cache miss). Therefore, the large page is very useful for improving the performance of scientific applications.

Table 5. Performance improvement percentages of OpenMP GTC on Hydra

#Threads	1	2	4	8	16
<b>64KB</b>	9.67%	9.50%	9.02%	9.04%	7.90%
<b>Blocking + Unrolling</b>	4.06%	4.45%	3.84%	3.31%	6.33%
<b>64KB+Binding</b>	--	9.60%	10.13%	8.97%	--

**Table 6. MPI and Hybrid performance improvement percentages for GTC on Hydra using large page of 64KB vs default page of 4KB**

#Processors	32	64	128	256	512
MPI	8.19%	7.58%	7.70%	8.52%	5.84%
Hybrid MPI/OpenMP	7.39%	8.34%	7.95%	7.54%	8.6%

## 6. Conclusions

In this paper, we analyzed the performance of OpenMP scientific applications such as NAS parallel benchmarks, an OpenMP Matrix multiplication, and a large-scale scientific application GTC on the multicore system Hydra, and used large page and processor binding to optimize the OpenMP performance for no requirement of any program modifications. Our experimental results showed that using the large page of 64KB on Hydra resulted in up to 33.92% performance improvement for OpenMP NPB benchmarks, and the OpenMP benchmarks benefited more from the large page than their MPI counterparts; using the large page of 64KB and processor binding resulted in up to 67.18% performance improvement for matrix multiplications, and up to 10.13% performance improvement for the GTC. These methods are also effective on optimizing the performance of MPI and hybrid MPI/OpenMP scientific applications. Our results also indicated that the OpenMP performance can be improved by using conventional loop optimization techniques such as blocking and unrolling inside the OpenMP parallel regions to take advantage of memory hierarchy on the multicore system.

## Acknowledgements

The authors would like to acknowledge TAMU Supercomputing Facilities for the use of the Hydra. We would also like to thank Stephane Ethier from Princeton Plasma Physics Laboratory and Shirley Moore from University of Tennessee for providing the GTC code and datasets.

## References

[BB94] D. Bailey, E. Barszcz, et al., *The NAS Parallel Benchmarks*, Tech. Report RNR-94-007, 1994.  
 [ES05] S. Ethier, First Experience on BlueGene/L, *BlueGene Applications Workshop*, ANL, April 27-28, 2005. [http://www.bgl.mcs.anl.gov/Papers/GTC\\_BGL\\_20050520.pdf](http://www.bgl.mcs.anl.gov/Papers/GTC_BGL_20050520.pdf).  
 [GA05] B. Gibbs, B. Atyam, et al., *Advanced POWER Virtualization on IBM@server p5 Servers:*

*Architecture and Performance Considerations*, IBM Redbooks, Nov. 2005.  
 [HE06] D. Hepkin, Guide to Multiple Page Size Support on AIX 5L Version 5.3, IBM, 2006.  
 [JF99] H. Jin, M. Frumkin and J. Yan, *The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance*, NAS Technical Report NAS-99-011, October 1999.  
 [JJ03] G. Jost, H. Jin, D. Mey, and F. Hatay, Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster, the *Fifth European Workshop on OpenMP (EWOMP03)*, September 2003.  
 [LL07] J. Levesque, J. Larkin, et al., *Understanding and Mitigating Multicore Performance Issues on the AMD Opteron Architecture*, LBNL-62500, March 7, 2007.  
 [KP06] D. Kleikamp and B. Pulavarty, Efficient Use of the Page Cache with 64KB Pages, *Ottawa Linux Symposium 2006*.  
 [NPB3] NAS Parallel Benchmarks 3.2.1, <http://www.nas.nasa.gov/Resources/Software/npb.html>.  
 [OH07] K. Olukotun, L. Hammond, and J. Laudon, *Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency*, Morgan & Claypool Publishers, 2007.  
 [OMP] OpenMP, <http://openmp.org/wp/resources/>  
 [PA08] Ruud Van der Pas, OpenMP Tutorial and Getting OpenMP Up To Speed, *the 4th International Workshop on OpenMP*, May 2008.  
 [SD06] Scientific Discovery, A progress report on the US DOE SciDAC program, 2006.  
 [TSCF] Texas A&M University Supercomputer Facility Hydra, <http://sc.tamu.edu/systems/hydra>.  
 [TW03] Valerie Taylor, Xingfu Wu, and Rick Stevens, *Prophesy: An Infrastructure for Performance Analysis and Modeling System of Parallel and Grid Applications*, *ACM SIGMETRICS Performance Evaluation Review*, Vol. 30, Issue 4, March 2003.  
 [WS02] S. Winwood, Y. Shuf, and H. Franke, Multiple Page Size Support in the Linux Kernel, *Ottawa Linux Symposium 2002*.  
 [WT09] Xingfu Wu, Valerie Taylor, Charles Lively and Sameh Sharkawi, Performance Analysis and Optimization of Parallel Scientific Applications on CMP Clusters, *Scalable Computing: Practice and Experience*, Vol. 10, No. 1, 2009.