Journal of Computer Science 6 (4): 406-412, 2010
ISSN 1549-3636

Analyzing Performance and Power of Multicore Architecture Using Multithreaded Iterative Solver

Ingyu Lee
Information System, Sorrell College of Business, Troy University, Troy, AL, USA 36081

Abstract: **Problem statement:** Scientific modeling and simulations have been popularly used with experiments and theoretical analysis in science and engineering communities. **Approach:** Consequently, computational demands are growing exponentially to afford large scale modeling and simulations. **Results:** As a result, multicore computing architectures had been proposed and several products are already available. However, we do not have a proper study on the performance, power and thermal issues of real science and engineering problems because software, which takes advantage of multicore architecture, is not available. **Conclusion/Recommendations:** In this study, we explored the performance and power characteristics of scientific algorithms on multicore architectures using a multithreaded version of sparse iterative linear solver, named mtCG, with real scientific application problems.

Key words: Multicore architecture, performance, multithread, iterative solver

INTRODUCTION

Computational modeling and simulations have been popularly used in science and engineering community to describe and understand complex phenomena instead of expensive or dangerous experiments such as drug design, global climate simulation, radiation simulation, crash testing aerodynamics and combustion (Heath, 2002). These modeling and simulations are usually represented as Partial Differential Equations (PDEs) which require meshes and sparse matrices. In these applications, we could not achieve the peak performance since those mesh and sparse matrix algorithms lack data reuse and locality.

At the same time, high performance computing community increases the number of transistors in a given area to improve performance. The latter meets physical limitation and generates new problems such as power consumption and thermal issues. To overcome these problems, multicourse architecture has been proposed and several products are already available in the market. However, we do not have a proper study on performance, power and thermal issues on multicore processors since the lack of scientific applications which benefits from multicore architectures. Several researches to characterize the performance of multicore architecture have been done with multiprogramming or loop level parallel benchmark programs (Jaleel *et al*., 2006; Li *et al*., 2005; Manjikian, 2001).

In this study, we profile the performance of scientific applications using a cycle accurate simulator to further understand the characteristics of multithreaded program on multicore architecture. We also explore the scalability, power and thermal issues on multicore architectures with real scientific application codes. Finally, we provide one variant of scientific application benefits multicore architectures. The latter could be used as a benchmark program in computing architecture community.

This study consists of the followings. We introduce some background information about benchmark programs and related researches. Then, we describe our simulation environments and multithreaded iterative solver. Experimental results of multithreaded iterative solver on multicore architectures are following. Finally some concluding remarks and future plans are described.

Background: SPEC Corporation (2000) has been used widely in computer architecture community to measure the performance of newly developed computing architectures. Several researches have been done with SPEC to measure the performance of multicore architecture (Li *et al*., 2005; Manjikian, 2001). However, these benchmark programs only support a single thread. Several single thread benchmark programs are used together in experiments. Consequently, the results are very close to multiprogramming characteristics rather than multithreaded program. Even with OpenMP version of SPEC supports loop level parallelism which is different with general multithreaded programs which have task level parallelism.

At the same time, NAS (Bailey *et al*., 1992) has been used to represent the workloads of scientific and engineering problems. NAS supports various versions of benchmark programs such as serial, Message Passing Interface (MPI), OpenMP and High Performance Fortran (HPF). However, it does not support multithread version of benchmark programs. The reason based on the fact that NAS has been used heavily to measure the performance of clustered systems rather than a single processor machine.

Splash-2 (Woo *et al*., 1995) has been used to represent multithreaded workloads for Shared Memory Processor (SMP) computers. In addition, it has real scientific kernels, cholesky, fft and lu and real scientific applications such as barnes-hut, fmm and water. However, this benchmark programs use synthetic data rather than real scientific application data. The behavior and memory access patterns of the benchmark programs with synthetic data are different with that of the real scientific applications.

There are several other benchmarks or variants of the traditional benchmark such as MinneSPEC (Klein Osowski and Lilja, 2002) which are developed to reduce simulation time and BioBench (Albayraktaroglu *et al*., 2005) which represents bioinformatics workloads and its parallel version using OpenMP (Jaleel *et al*., 2006) and MineBench (Narayanan *et al*., 2006) which represents data mining workloads on single and parallel machines.

On the other hand, many research on characterizing the performance, power and thermal of multicore architectures have been done. Jaleel *et al*. (2006) characterized the last level cache performance on Chip Multi Processor (CMP) using OpenMP version of Biobench. Li *et al*. (2005) characterized performance, energy and thermal of Simultaneous Multi Thread (SMT) and CMP with replicating single threaded applications. Monchiero *et al*. (2006) explores the design space for multicore architecture in performance, power and thermal view using Splash-2 (Woo *et al*., 1995) benchmark programs. However, present study is the first contribution to characterize the multicore architecture with real multithreaded scientific applications in author's awareness.

Multithreaded iterative solver: mtCG: The most common algorithm in scientific modeling and simulation is a sparse iterative solve such as Conjugate Gradient (CG). Fig. 1 contains an outline of a generic CG algorithm used in many applications. This CG scheme uses standard data structures for storing the sparse matrix A and vectors p,q,r. Only the nonzero of sparse matrix A and its corresponding indices are explicitly stored using a standard sparse format. The vectors p, q and r are stored as one-dimensional arrays in contiguous locations in memory. A single iteration of CG requires one matrix-vector multiplication, two vector inner products, three vector additions and two floating point divisions. Among these operations, the matrix-vector multiplication dominates the computational cost accounting for more than 90% of the overall execution time. Due to the sparse nature of the matrix A, the number of floating point operations per access to the main memory is relatively low during matrix vector multiplication. Additionally, the access pattern of the elements in the vector p depends on the sparse structure of A.

To provide a multithreaded version of CG algorithm, we divided the matrix by row-wise as shown in Fig. 2. Each thread multiplies row block of matrix with the specific source vector members and stores at the destination vector members. Since we do not share the destination vector, this module has a perfect parallelism. However, sparse matrix algorithm could not benefit from cache as does in dense matrix since it lacks data reuse and locality.

The total operation time required for sparse matrix multiplication is represented as:

$$T_{smv} = (m*T_{mul} + (\tilde{m}1)*T_{add})*N/N_{thread} + T_{mem} \quad (1)$$

N = Size of the matrix
N_{thread} = Number of threads
m = Average nonzero values in a row
T_{mem} = Memory accessing time
T_{mul} and T_{add} = Floating point operations

CG(A,b)
%% A - input matrix, b - right hand side
x_0 is initial guess;
$r = b - Ax_0$; $\rho = r^T r$; $p = r$
for i=1,2,... do
  $q = Ap$
  $\alpha = \rho/(p^T q)$
  $x_i = x_{i-1} + \alpha p$
  $\tilde{r} = r - \alpha q$
  if $||\tilde{r}||/||b||$ is small enough then stop
  $\tilde{\rho} = \tilde{r}^T \tilde{r}$
  $\beta = \tilde{\rho}/\rho$
  $p = r + \beta p$
  $r = \tilde{r}$; $\rho = \tilde{\rho}$
end for

Fig. 1: The Conjugate Gradient (CG) scheme

Memory access time, T_{mem} is determined by the cache architecture inside a system. Assume we have multicore architecture which has two levels of cache, L1 and L2. L1 cache is dedicated cache for each core and L2 cache is shared by all cores. Then, using formula from (Jaleel *et al*., 2006, Hennessy and Patterson, 2003), the memory access time is represented as:

$$T_{mem} = hitrate_{L1} * T_{L1} + missrate_{L1} * penalty_{L1} \quad (2)$$

$$penalty_{L1} = hitrate_{L2} * T_{L2} + missrate_{L2} * penalty_{L2} \quad (3)$$

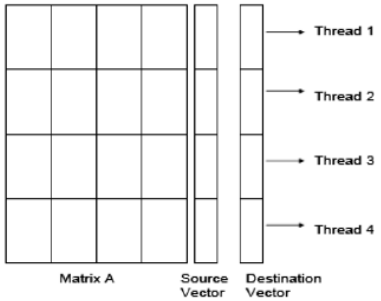
$hitrate_{L1}$ and $hitrate_{L2}$ = Cache hit ratio
$missrate_{L1}$ and $missrate_{L2}$ = Cache miss ratio
T_{L1} and T_{L2} = Cache access time
$penalty_{L1}$ and $penalty_{L2}$ = Cache miss penalty

Algorithm Fig. 2 shows that we need two vectors which have length N and two other vectors which have length nonzero. Then, without considering memory prefetcher or cache replacing scheme, average $missrate_{L1}$ and $missrate_{L2}$ is defined as:

$$missrate_{L1} = 2*(N + nonzero) / CacheLine_{L1} \quad (4)$$

$$missrate_{L2} = 2*(N+nonzero)/(CacheLine_{L2}/N_{thread}) \quad (5)$$

$CacheLine_{L1}$ and $CacheLine_{L2}$ = Entries in cache
nonzero = Nonzero elements in the matrix




```
mtSMV(myid,startx,endx)
% myid = my thread id;
% startx = starting row index for my thread;
% endx = ending row index for my thread;
% row_index = row index vector
% col_index = column index vector
% nonz_val = nonzero values vector
for i=startx,endx do
  for j=row_index[i],row_index[i+1] do
    destination[i]=nonz_val[j]*source[col_index[j]];
  end for
end for
```

Fig. 2: Multithread matrix vector multiplication

Finally, $hitrate_{L1}$ and $hitrate_{L2}$ can be computed from $missrate_{L1}$ and $missrate_{L2}$. Computing the exact value of $penalty_{L2}$ is difficult in real computing environment. However, even with a simple memory prefetcher, the value is negligibly small in our algorithm since it accesses memory in sequential direction (Malkowski *et al*., 2005a; 2005b).

Since, sparse matrix multiplication is one of those embarrassingly parallel algorithm, we can define the speed up as:

$$SpeedUp = T_s / T_p \quad (6)$$

T_s = Single processor execution time
T_p = Execution time with p processors

Considering, T_p for sparse matrix vector multiplication is T_s/N_{thread}, we can achieve N_{thread} times speed up in theory (Grama *et al*., 2002).

MATERIALS AND METHODS

We used SESC (Renau *et al*., 2005), a cycle accurate architecture simulator, which supports Chip Multi Processor (CMP) and Simultaneous Multi Threading (SMT) architecture. Each core is an out-of-order superscalar processor with private L1 caches (separated instruction and data cache) and a shared L2 cache (hybrid instruction and data cache). The details of the parameters we used for SESC simulator are described in Table 1. We used Wattch (Brooks *et al*., 2000) to measure power usage on processor core and Orion (Wang *et al*., 2002) to measure shared bus power usage. Then, we applied Hotspot (Skadron *et al*., 2003) to get thermal characteristics based on the results of power consumption trace of SESC simulator. Since the memory access pattern of artificially generated data set is different with that of a real application, we used bcsstk16 from MatrixMarket (Boisvert, 1997). The latter is a sparse matrix generated from a real structure analysis application and popularly used in a scientific computation community.

Table 1: Simulation parameters

Parameters	Description
Frequency	2 GHz
Machine	32 bits, 4/4/6 fetch/issues/retire ROB (126), intRegs (90), fpRegs (68)
L1 Icache	16*1024, 64, 2-way associative, hit latency (2)
L1 dcache	16*1024, 64, 4-way associative, hit latency (2)
L2 cache	1024*1024, 64, 8-way associative, hit latency (10) 10 cycle hit latency
BTB	2048 entry, LRU policy
TLB	512 entry (D), 256 entry (I), LRU policy
Functional units	3 integer and 3 FP units
Cores	4 (0-3)
Shared bus	width (64)
Memory BW	6G bytes/sec
Memory latency	LRU, 64, 490 cycles

RESULTS

Figure 3 shows our experimental results of mtCG benchmark program using bcsstk16 as an input matrix. Since the input matrix has regular nonzero pattern, mtCG has good balance between cores by assigning the same junk of rows to each thread. As a result, Instruction Per Cycle (IPC) numbers are similar even with different number of cores. In addition, we can achieve linearly increasing speed up with an increasing number of cores. Especially, L2 cache miss rates are decreasing by adding more cores up to four cores. We conjecture that the number of L2 accesses dramatically decreases with eight cores since the data per core is small enough to fit in L1 cache. Consequently, each core uses a similar amount of power as shown in Fig. 3.

In addition to the performance and power, the temperature becomes an important factor in advanced computing architectures. To better understand the thermal characteristics, we traced the changes of temperature during benchmark program executions using Hotspot 3.0 (Skadron *et al*., 2003) based on the floor plan as shown in Fig. 4a-c. The detail experimental parameters related to thermal are shown in (Renau *et al*., 2005). We investigated three multicore floorplans. The first layout is spreading hot areas around the corner and keeping L2 cache at the center. The second layout is lining up cores to arrange functional units at the center. The last layout is clustering functional units at the center to improve the performance by having functional units nearby each others. Based on the floor plan in (Renau *et al*., 2005), we scale down four cores into a single processor. As a result, every units are 1/4 scale of the floor plan. In addition, we locate shared bus at the center to keep cache consistency using MESI protocol.

Figure 5 shows the temperature difference between different floor plans of four cores architectures: Spread (Fig. 5a), Lineup (Fig. 5b) and Centered (Fig. 5c). All floor plans have hotspots on issue related units and floating point units. The hottest unit is load/store queue with this benchmark program. The multithreaded version of algorithm mtCG has several synchronization stops between threads during executions. The latter raises the temperature of load/storequeues. In addition, store queue is also suffering from corner effects; the inside chip has plenty area to spread energy but corner area cannot dissipate energy as does in center area.

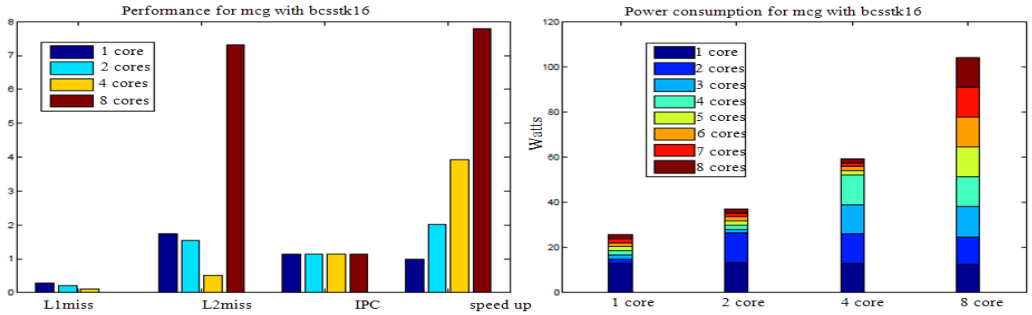
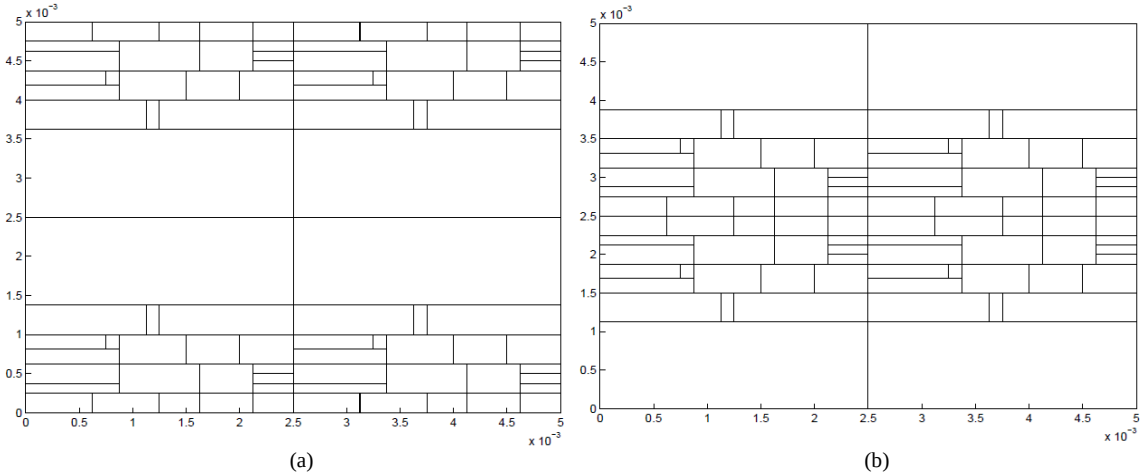
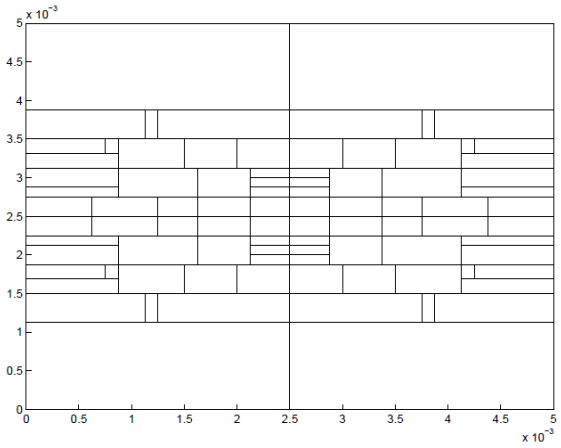


Fig. 3: Performance and power usage of mtCG benchmark program

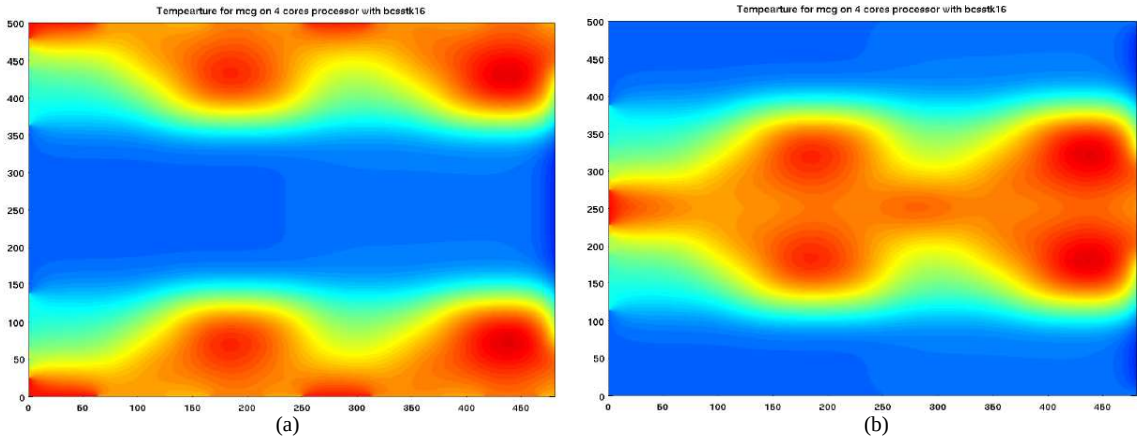


(a) (b)

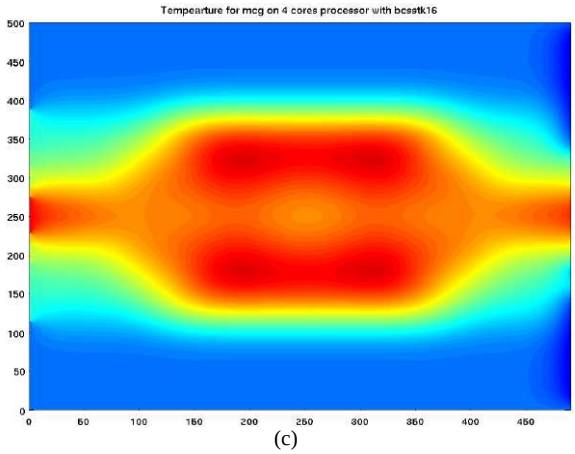


(c)

Fig. 4: Multicore floor plans: (a) spread, (b) lineup and (c) centered



(a) (b)



(c)

Fig. 5: Temperature for mtCG benchmark program on different floor plans: (a) spread, (b) lineup and (c) centered

The latter causes the temperature of right side of core has higher temperature than that of left side of core in our lineup displacement. The centered displacement has the coupling thermal effect which two hotspots held up

high temperatures and affects each other. The temperature difference between spread, lineup and centered layout is not noticeable in our experiment. The centered layout has slightly higher temperature in overall chip area.

DISCUSSION

In our experiments, sparse iterative solver shows linearly increasing performance with an increasing number of cores. Considering the sparse matrix vector multiplication, which is scalable, dominates the cost of sparse iterative solver, the observation is not surprising. Accordingly, each core uses similar amount of power during the computation since each core executes approximately similar number of floating point operations in our scheme. Consequently, theoretical N_{thread} speed up with using N thread as in our analysis is possible with a multicore computing architecture if the algorithm is designed to take benefits from multicore architecture.

Multithreaded sparse iterative solver raises temperature on issue related units and floating point units since the algorithm requires synchronization between threads and also executes huge number of floating point multiplications and additions. The latter raise the temperature of the related units including load/store queue and floating point units. To relieve the temperature in the related units, we could arrange the related units to locate far apart or use L2 cache as a coolant to surround the hot units.

CONCLUSION

In this study, we explore performance, power and thermal issues on modern computing architectures with using real scientific applications. We investigate multicore architectures with multithreaded benchmark programs mtCG with real scientific application data. Multicore architectures could provide an incredible speed up with the given power and thermal constraints as long as the algorithms are scalable. Finally, we provide a multithreaded version of CG benchmark program, named mtCG, which can be used to measure the performance of multicore architectures.

We are planning to develop more multithreaded benchmark programs based on real scientific and engineering applications. In the near future, we could provide multithreaded NAS benchmark programs to evaluate multicore architectures. In addition, we are studying on additional computing architecture issues such as cache sharing policy and thread scheduling to lower temperature.

REFERENCES

Albayraktaroglu, K., A. Jaleel, X. Wu, M. Franklin and B. Jacob *et al*., 2005. Biobench: A benchmark suite of bioinformatics applications. Proceeding of the ISPASS 2005, Mar. 20-22, IEEE Computing Society Press, Austin, TX, USA., pp: 2-9.

Bailey, D.H., L. Dagum, E. Barszcz and H.D. Simon, 1992. NAS parallel benchmark results. Proceedings of the 1992 ACM/IEEE Conference on Supercomputing, Nov. 13-18, IEEE Computer Society Press, Los Alamitos, CA, USA., pp: 386-393.

Brooks, D., V. Tiwari and M. Martonosi, 2000. Wattch: A framework for architectural-level power analysis and optimizations. Proceedings of the 27th Annual International Symposium on Computer Architecture, June 10-14, ACM Press, Vancouver, BC, Canada, pp: 83-94.

SPEC Corporation, 2000. CFP2000 (Floating Point Component of SPEC CPU 2000). http://www.spec.org/cpu2000/CFP2000

Grama, A., G. Karypis, V. Kumar and A. Gupta, 2002. Introduction to Parallel Computing. 2nd Edn., Prentice Hall, USA., ISBN: 10: 0201648652.

Heath, M., 2002. Scientific Computing: An Introductory Survey. 2nd Edn., Prentice Hall, USA., ISBN 10: 0072399104.

Hennessy, J. and D. Patterson, 2003. Computer Architecture: A Quantitative Approach. 3rd Edn., Morgan Kaufmann Publishers, USA., ISBN: 10: 1558605967.

Jaleel, A., M. Mattina and B. Jacob, 2006. Last-Level Cache (LLC) performance of data-mining workloads on a CMP-a case study of parallel bioinformatics workloads. Proceeding of the 12th HPCA, Feb. 11-15, IEEE Computer Society Press, Austin, TX., pp: 1-1. http://www.lib.umd.edu/drum/handle/1903/7453

Klein Osowski, A. and D.J. Lilja, 2002. Minnespec: A new spec benchmark workload for simulation-based computer architecture research. IEEE Comput. Architect. Lett., 1: 1-7. http://portal.acm.org/citation.cfm?id=1115804

Li, Y., Brooks, D., Z. Hu and K. Skadron, 2005. Performance, energy and thermal considerations for SMT and CMP architectures. Proceeding of the 11th HPCA, Feb. 12-16, IEEE Computer Society Press, San Francisco, CA., pp: 71-82. http://portal.acm.org/citation.cfm?id=1043406

Malkowski, K., I. Lee, P. Raghavan and M. Irwin, 2005a. Conjugate gradient sparse iterative solvers: Performance-power characteristics. Proceeding of the International Parallel and Distributed Processing Symposium (IPDPS), Apr. 25-29, IEEE Computer Society Press, Rhodes Island, Greece, pp: 1-8. http://fortknox.csc.ncsu.edu/proj/hppac/papers/son.pdf

Malkowski, K., I. Lee, P. Raghavan and M. Irwin, 2005b. On improving performance and energy profiles of sparse scientific applications. Proceeding of the International Parallel and Distributed Processing Symposium (IPDPS), Apr. 25-29, IEEE Computer Society Press, Rhodes Island, Greece, pp: 1-1. http://d.wanfangdata.com.cn/NSTLHY_NSTL_HY13711487.aspx

Manjikian, N., 2001. multiprocessor enhancements of the simpleScalar toolset. ACM SIGARCH Computer Architect., 29: 8-15. http://portal.acm.org/citation.cfm?id=373578

Boisvert, R., 1997. Matrix market: A web resource for test matrix collection. Town meeting on Online Delivery of NIST Reference Data, NIST, Gaithersburg, MD. http://math.nist.gov/MatrixMarket

Monchiero, M., R. Canal and A. Gonzalez, 2006. Power/Performance/Thermal design-space exploration for multicore architectures. IEEE Trans. Paral. Distribut. Syst., 19: 666-681.

Narayanan, R., B. Ozisikyilmaz, H. Zambreno, J.P.G. Memik and A. Choudhary, 2006. Minebench: A benchmark suite for data mining workloads. Proceeding of the International Symposium on Workload Characterization, Oct. 25-27, IEEE Computer Society Press, San Jose, CA., pp: 1-7. http://www.bioperf.org/NOZ06.pdf

Renau, J., B. Fraguela, J. Tuck, W. Liu and M. Prvulovic *et al*., 2005. SESC simulator, http://sesc.sourceforge.net.

Skadron, K., M.R. Stan, W. Huang, S. Velusamy and K. Sankaranarayana *et al*., 2003. Temperature-aware microarchitecture ACM SIGARCH Computer Architect., 31: 2-13. http://portal.acm.org/citation.cfm?id=871656.859620

Wang, H.S., X. Zhu, L.S. Peh and S. Malik, 2002. Orion: A power-performance simulator for interconnection network. Proceeding of the MICRO 35, Nov. 18-22, IEEE Computer Society, Istanbul, Turkey, pp: 294-305.

Woo, S.C., M. Ohara, E. Torrie, J.P. Singh and A. Gupta, 1995. The splash-2 programs: Characterization and methodological considerations. Proceedings of the 22nd Annual International Symposium on Computer Architecture. June 22-24, ACM, Ligure, Italy, pp: 1-13. http://www.csd.uoc.gr/~hy527/papers/splash2_isca.pdf