Conjugate Gradient Sparse Solvers: Performance-Power Characteristics *

Korad Malkowski, Ingyu Lee, Padma Raghavan, Mary Jane Irwin

The Pennsylvania State University
Department of Computer Science and Engineering
University Park, PA. 16802 USA
{malkowski, inlee, raghavan, mji}@cse.psu.edu

Abstract

We characterize the performance and power attributes of the conjugate gradient (CG) sparse solver which is widely used in scientific applications. We use cycle-accurate simulations with SimpleScalar and Wattch, on a processor and memory architecture similar to the configuration of a node of the BlueGene/L. We first demonstrate that substantial power savings can be obtained without performance degradation if low power modes of caches can be utilized. We next show that if Dynamic Voltage Scaling (DVS) can be used, power and energy savings are possible, but these are realized only at the expense of performance penalties. We then consider two simple memory subsystem optimizations, namely memory and level-2 cache prefetching. We demonstrate that when DVS and low power modes of caches are used with these optimizations, performance can be improved significantly with reductions in power and energy. For example, execution time is reduced by 23%, power by 55% and energy by 65% in the final configuration at 500MHz relative to the original at 1GHz. We also use our codes and the CG NAS benchmark code to demonstrate that performance and power profiles can vary significantly depending on matrix properties and the level of code tuning. These results indicate that architectural evaluations can benefit if traditional benchmarks are augmented with codes more representative of tuned scientific applications.

1 Introduction

Computational modeling and simulation is typically used to understand complex systems or physical phenomena in many scientific and engineering disciplines such as aerodynamics [14] and fluid flows [7]. The models in many of these applications are described by partial differential equations (PDEs). The computational solution of such nonlinear PDE based models require the use of high performance computing architectures. The simulation software is typically designed to exploit architectural features of such machines. In such tuned simulation codes, advanced sparse matrix algorithms and their implementations play a critical role; sparse solution time typically dominates total application time, which can be easily demonstrated.

In this paper, we consider the performance, power and energy characteristics of a widely used sparse solver in scientific applications, namely a conjugate gradient (CG) sparse solver. Our goal is to increase the power and energy efficiency of the CPU and the memory subsystem for such a CG based sparse solver, without degrading performance and potentially even improving it. Toward this goal, we begin by specifying a processor and memory system similar to that in the BlueGene/L, the first supercomputer developed for power-aware scientific computing [24]. We use cycle-accurate simulations with SimpleScalar [25] and Wattch [4] to study performance and power characteristics when low power modes of caches are used with Dynamic Voltage Scaling (DVS) [6]. Towards improving performance, we explore the impact of memory subsystem optimizations in the form of level 2 cache prefetching (LP), and memory prefetching (MP). Finally, towards the improved evaluations of architectural features of future systems, we consider how interactions between the level of code tuning, matrix properties and memory subsystem optimizations affect performance and power profiles.

Our overall results are indeed promising. They indicate that tuned CG codes on matrices representative of the majority of PDE-based scientific simulations can benefit from memory subsystem optimizations to execute faster at substantially lower levels of system power and energy. Our results, at the level of a specific solver and specific architectural features, further the insights gained at a higher system level through earlier studies on the impact of DVS on scien-

*The work was supported in part by the National Science Foundation through grants NSF ACI-0102537 and NSF CCF-0444345, and by the Director, Office of Science, Division of Mathematical, Information, and Computational Sciences of the U.S. Department of Energy under contract number DE-AC03-76SF00098.

1-4244-0054-6/06/$20.00

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

Figure 1. The conjugate gradient scheme.

tific computing workloads on clusters [10, 19].

The remaining part of this paper is organized as follows. In Section 2, we describe the CG based sparse solvers in terms of their underlying operations. In Section 3, we describe our base RISC PowerPC architecture, our methodology for evaluating performance, power and energy attributes through cycle-accurate simulation, and our memory subsystem optimizations. Section 4 contains our main contributions characterizing improvements in performance and energy, and differences in improvements resulting from the interplay between code and architectural features. Section 5 contains some concluding remarks and plans for future related research.

2 Sparse Linear System Solution Using Conjugate Gradients

In this section, we describe the CG algorithm with an emphasis on how its main computations access the memory subsystem. Figure 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 nonzeroes of the sparse matrix, A, and its corresponding indices are explicitly stored using a standard sparse format. The vectors p, q, r are stored as one-dimensional arrays in contiguous locations in memory.

A single iteration of the 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 sparsity structure of A. Sparse matrices from typical scientific applications can be ordered to a *band* form where the nonzeroes are clustered around the diagonal using a scheme such as RCM [12]. This improves the locality of access in p. Such re-orderings can be used with other techniques like register-blocking and loop-unrolling to further improve the performance [15, 26]; some of these techniques may actually increase number of floating-point operations.

SPARSITY [15] is a highly optimized matrix-vector multiplication routine. It utilizes register and cache blocking to improve performance typically after an RCM ordering is applied to the matrix. Figure **??** shows how SPARSITY reduces the row and column index overhead while enabling data local accesses on a sample sparse matrix by using a "2x1" register blocking with loop unrolling. Although there is a slight increase in floating point operations and storage requirements due to the addition of explicit zeroes into the matrix structure, the scheme considerably outperforms the unoptimized kernel [15].

To demonstrate the interplay between code optimizations, matrix properties, and architectural optimizations, we examine four forms of the CG algorithm. The first corresponds to a natural un-optimized implementation (CG-U) of a plain conjugate gradient algorithm. The second corresponds to its optimized version (CG-O) using SPARSITY [15]. For each, we consider a sparse matrix typical of scientific applications, namely `bcsstk31` from a well known test collection [13]. The matrix is considered with two different orderings (i) using RCM, with corresponding CG forms labeled CG-O RCM, CG-U RCM, and (ii) using a random ordering (CG-O Random, CG-U Random). The latter are used primarily to discuss how our CG codes relate to a well known benchmark code, the CG code from the NAS benchmark (CG-NAS) [2] and to thus characterize how features of CG-NAS impact power and performance evaluations. In all cases, we evaluate the performance and power characteristics for a set of 25 complete CG iterations; i.e. 25 iterations of the for loop in Figure 1. This is consistent with the number of CG iterations used in CG-NAS. This is sufficient to capture the behavior of typical applications which require several hundred iterations for convergence to a solution of desired accuracy [3].

3 Modeling Performance and Power Characteristics

We now describe our methodology for emulating architectural features to evaluate their performance and energy characteristics. The power and performance of sparse kernels are emulated by SimpleScalar3.0 [5] and Wattch1.02d [4] with extensions to model memory subsystem enhancements. We use Wattch [4] to calculate the power consumption of the processor components, includ-

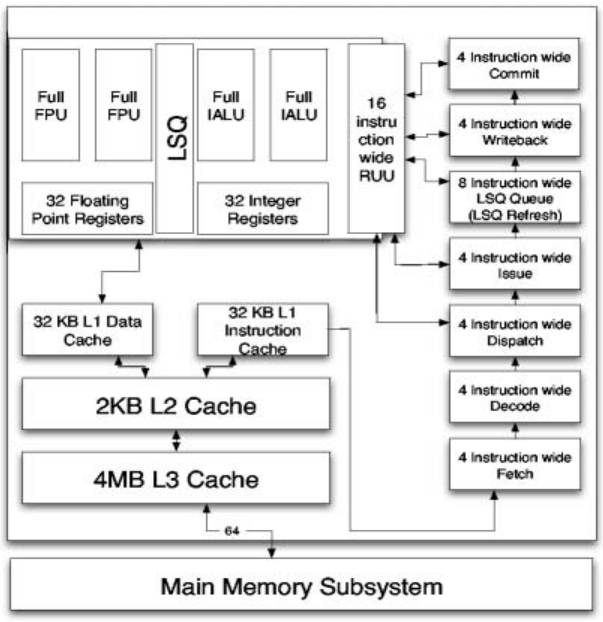

Figure 2. The base architecture.

ing the pipeline, registers, branch prediction logic. Wattch does not model the power consumed by the off-chip memory subsystem. We therefore developed a DDR2 type memory performance and power simulator for our use. More details of our power modeling through Wattch can be found in [22].

3.1 Base Architecture

Our base architecture has two floating point units and two integer ALUs. Each FPU has a multiplication/division module and modules for other arithmetic and logic; thus, the system is capable of issuing 4 floating point instructions each cycle. The data path width of the CPU is 4 instructions and data paths between memory and L3 cache are 128 bit wide with cache lines of 128 bytes, i.e., 16 double precision operands or 32 integer operands.

We model a cache hierarchy with three levels on chip, including a 32KB data/32KB instruction level 1 cache (L1), a 2KB level 2 cache (L2), and a 4MB unified level 3 cache (L3). Wattch is configured to model only two levels of cache, but we added new functions to model our hierarchy. We used CACTI [4] modes for a 4MB SRAM L3 cache. We operate the SRAM L3 at system frequency and voltage levels when we consider different frequency-voltage pairs to simulate the effects of utilizing DVS.

The specifications of the off-chip memory are particularly important for characterizing sparse matrix computations. We assume a DDR2 type memory power model with our base architecture to start with a lower-power configuration than what is possible with standard DDR type memory. We model nine 256MBit x 8 chips to provide 256MB of memory with specifications obtained from Micron Technology Inc. data sheets [16] with memory operating at 266MHz. We verified our approach by observing that the timing delays for our system with the CPU clock frequency at 700MHz is similar to those reported in the literature [1, 24].

Our base architecture is similar to the configuration in the BlueGene/L[24] based on the PowerPC440 embedded core. The main difference is that we only model a single-core because Wattch currently does not allow for the simulation of more than one thread of execution. This is still realistic, because the BlueGene/L processor is not designed to work in the SMP mode; Level 1 caches are not consistent and the recommended usage for the second core is as a communication co-processor. Furthermore, if both cores are fully utilized for computation, the demands on the memory subsystem will only increase, potentially magnifying the interactions between code tuning and memory subsystem optimizations. More details of our system and its relation to the processor in the BlueGene/L can be found in [22].

3.2 Memory Subsystem Optimizations

Starting with the base architecture, henceforth denoted by 'B,' we consider memory subsystem optimizations that can potentially benefit sparse applications. These optimizations focus on prefetching to mask the latencies of memory access; such prefetchers have been considered in other contexts [8, 18, 21, 23]. Details include:

- Memory prefetching (on or off), stride-1, at the memory controller, labeled 'MP'; this feature reduces the effective latency of memory access depending on data access patterns. It is modeled by adding a prefetch buffer to the memory controller in the form of a 16 element table where each element holds a cache line of 128 bytes. On a load, if data is not available in the prefetch buffer, a new entry is allocated in the table using a full LRU replacement policy, and the memory controller does a prefetch for the next cache line upon completion of the original load; see Figure 3. We model the power consumed by our prefetch buffer as the cost of operating a small 16 entry, direct mapped cache with a 128 byte cache line due to problem with CACTI.

- Level 2 cache prefetching (on or off), stride-1, denoted by 'LP'; this feature can benefit codes with locality of data access but poor data re-use in caches. When a read at address a completes, a read is scheduled for address a + L2 cache-line size. The extra energy consumption is modeled as second cache access.

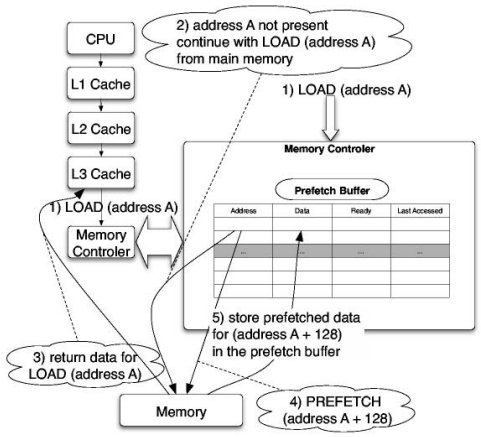

Figure 3. A memory prefetcher implementation.

3.3 Emulating Low-Power Modes of Caches and Processors

Future architectures are expected to exploit power-saving modes of caches and processors. These include sleepy or low-power modes of caches [17] where significant fractions of the cache can be put into a low power mode to reduce power. Additionally, power can be decreased by Dynamic Voltage Scaling (DVS) [6] where the frequency and the voltage are scaled down. We now model how these features can be utilized by sparse scientific applications if they are exposed in the Instruction Set Architecture (ISA).

The 4MB SRAM L3 cache in our base architecture, typical of many high-end processors, accounts for a large fraction of the on-chip power. To simulate the potential impact of cache low power modes of future architectures on CG codes, we perform evaluations by considering the following L3 cache sizes: 256Kb, 512KB, 1MB, 2MB and 4MB.

To model DVS [6], we consider eight CPU frequencies with corresponding nominal V_{dd} voltages. The frequency-V_{dd} pairs we used are: 300MHz-0.46V; 400MHz-0.66V; 500MHz-0.77V; 600MHz-0.84V; 700MHz-0.88V; 800MHz-0.93V; 900MHz-1.03V; and 1000MHz-1.20V.

4 Empirical Results

In this section, we evaluate the impact of using low power modes of caches, and low power modes of the processor using DVS, along with memory subsystem optimizations for sparse CG solvers with different levels of tuning typical of scientific applications. We are interested in both the effect of memory subsystem optimizations and software optimizations. To evaluate performance, we use execution time (in seconds) as our metric. Other metrics include the average system power consumed the processor and memory (in Watts) and energy (in Joules), computed as the product of the system power and execution time.

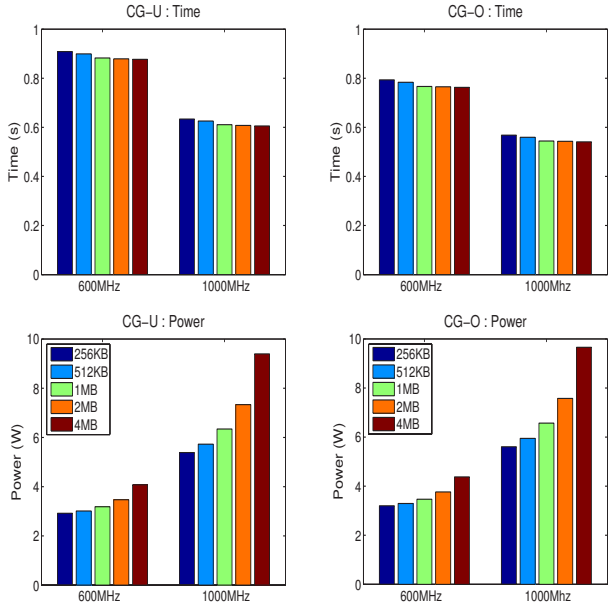

Figure 4. Execution time (top) and average system power (bottom) for CG-U (left) and CG-O (right) for bcsstk31 with RCM. Values are shown at two frequencies, 600MHz and 1000MHz with 5 cache sizes: 256KB, 512KB, 1MB, 2MB and 4MB.

In figures in this section, plots for the base architecture are labeled 'B', plots for base architecture with either a memory or an L2 prefetcher are labeled 'MP' and 'LP' respectively, and plots for the base architecture enhanced by both features are labeled 'LP+MP'. The x-axis shows some or all frequencies in the range 300MHz to 1000 MHz (1GHz) while the y-axis is used to show execution time, power and energy values.

4.1 Power and Performance Tradeoffs Using DVS and Low Power Modes of Caches

We begin by evaluating the power and performance tradeoffs for CG when DVS and the low power modes of caches can be utilized. Figure 4 shows execution time and average power consumption of a CG based solver for the base architecture at the original frequency of 1 GHz and when the frequency is scaled down to 600 MHz for a series of cache sizes 256KB, 512KB, 1MB, 2MB and 4MB. The plots on the left are for the unoptimized version of the CG (CG-U) algorithm. The plots on the right show results for an optimized implementation of the CG solver (CG-O), utilizing register blocking and loop unrolling. In both cases, we consider the `bcsstk31` matrix with an RCM ordering typical of scientific applications.

inherently have cache reuse. The RCM ordering improves the locality of access in the source vector and the

SMV-O blocking allows some reuse but these levels of reuse are sufficiently small and unaffected by larger cache sizes.

Observe that the cache size has very little impact on the performance of both CG-U and CG-O while significantly reducing the power. This is primarily because SMV-U, SMV-O and CG inherently have low cache reuse. Consequently, utilizing low power modes of caches can lead to significant decreases power consumption without adverse impacts on performance. This effect is more pronounced at higher frequencies. Observe also that as DVS is utilized and both frequency and voltage are scaled down, the execution time increases and the power decreases for both CG-U and CG-O. Observe also that power reductions are significantly more than the increase in execution time when both are measured relative to their values at 1000MHz. This is as expected; a linear scaling down of the frequency and voltage results in cubic improvements in power, while the CPU cycle time (and hence the execution time) is in inverse linear proportion to the frequency.

Henceforth, we will only report results for system configurations with the smallest cache size of 256KB. This will represent the impact of using low power modes of caches as we continue with for CG-U (left) and CG-O (right) run on RCM ordered bcsstk31. Results relative to CG-U run on bcsstk31 ordered with RCM at 1000MHz, 4MB cache.

4.2 Impact of L2-Cache and Memory Prefetchers

We now consider in detail the impact on performance and power when memory and L2 prefetchers are used with DVS and sleepy modes of the level 3 cache.

Figure 5 shows relative values of the execution time, power, and energy for unoptimized CG-U (top) and optimized CG-O (bottom). The values are relative to those observed for CG-U for the base architecture, 'B' at 1000MHz with a 4MB level 3 cache, set at 1. Thus, presented values capture the impacts on power, performance and energy from the combined effects of hardware and software optimizations. Observe that values smaller than 1 indicate improvements in each metric while values greater than 1 indicate degradations. We provide relative values for frequencies of 400, 500, 600 and 700 MHz for the smallest level 3 cache size of 256KB and values at the reference point, i.e., 'B' at 1000 MHz and 4MB L3 cache. At each frequency, we provide grouped bars representing the addition of either memory prefetching (MP), or L2 prefetching (LP), or their combination (LP+MP). In all cases, as stated earlier in Section 2, we consider 25 CG iterations for the `bcsstk31` matrix with an RCM ordering, typical of scientific applications as discussed earlier in Section 2.

Observe that for all frequencies reported in this figure, the optimized version CG-O has significantly better performance than CG-U. We will report results for the optimized version first, and then return to the unoptimized version.

Consider the performance and power metrics for CG-O along the bottom half of Figure 5. At 400MHz with either LP or both LP and MP active, we achieve performance better than that of CG-U at the reference point of 1000 MHz with over 60% savings in power. At 500MHz, we improve the performance by 23% for LP + MP with approximately 55% saving in power and and over 62% savings in energy. More importantly, at all frequencies with both LP and MP, executions times for CG-O are *better or equal to values observed not only with respect CG-U at the reference point but also with respect to CG-O at the reference configuration*.

Similar improvements are also observed for the unoptimized CG (CG-U) shown at the top half of Figure 5. However, because register blocking and loop unrolling are not present in CG-U, performance improvements from the use of LP and MP are smaller, and the frequencies at which performance is better than or equal to that at the reference point are higher. For example, at 600MHz, with LP + MP, there is a 10% reduction in execution time, with approximately 51% reduction in power and over 55% for energy savings.

These results show that significant power and energy savings are possible along with performance improvements when relatively simple memory subsystem optimizations are used with DVS and power saving modes of the L3 cache.

4.3 Impact of Sparse Matrix Properties

Our results so far indicate that the level of tuning of the CG codes affects the degree to which the same set of architectural features, including low power modes of caches, DVS, and memory subsystem optimizations impact the performance and power metrics.

In all our experiments thus far, CG-U and CG-O were executed on the same matrix which had been reordered using RCM to increase locality of access in the source vector during matrix vector multiplication. It is a property of the sparse matrix which allows such reorderings to be effective in clustering nonzeroes about the diagonal and it arises from the fact that PDE-based simulations based on finite-difference or finite-elements matrices give rise to matrices with localized connectivity [11]. Sparse matrices from other applications can have random connectivity. More importantly, even sparse matrices from PDE-based applications may occur with a random ordering although they can be reordered to improve locality.

The commonly used CG code from the popular NAS benchmark [2] used in earlier performance and power evaluations, considers CG on a highly unstructured, near random sparse matrix. We now consider this code with three forms of our CG-U and CG-O codes to show how architectural evaluations are particularly sensitive to the the matrix properties and level of tuning in the code.

Figure 6 shows relative execution time and power for

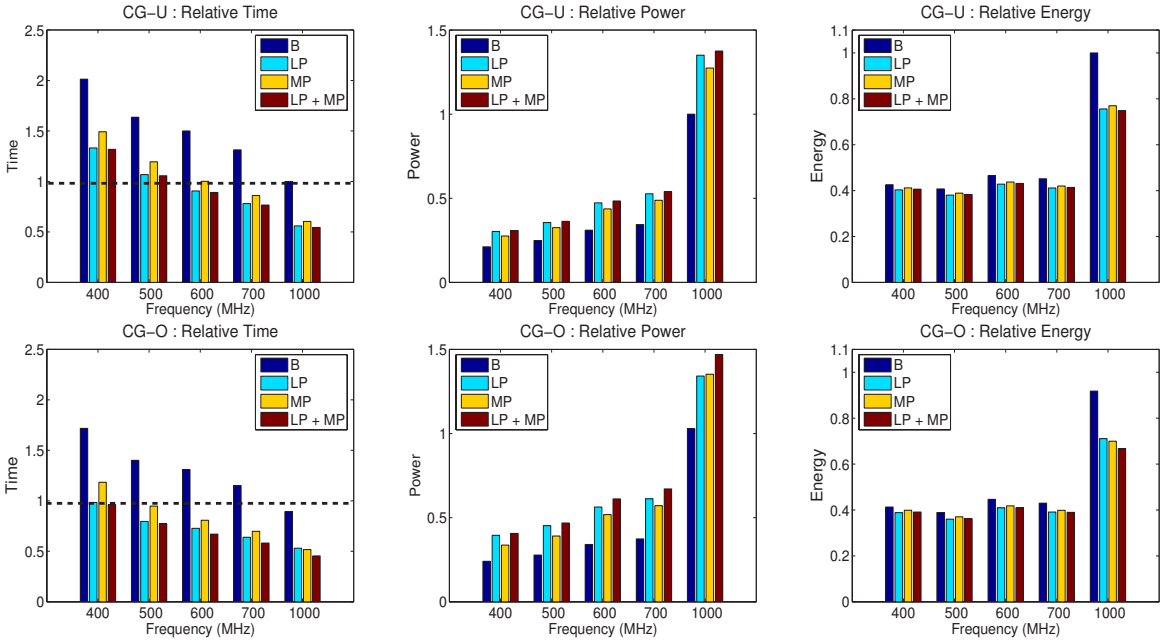

Figure 5. Relative Time, Power and Energy of unoptimized CG (CG-U top) and optimized CG (CG-O bottom) algorithms solving the bcsstk31 matrix. Results shown for base, LP, MP and LP + MP configurations run at frequencies of interest with 256KB L3. Results are relative to a fixed reference point set at 1000MHz, 4MB L3 running CG-U algorithm on RCM ordered bcsstk31 matrix.

CG-O on `bcsstk31` and an RCM ordering, CG-O on `bcsstk31` with a random ordering, CG-U on `bcsstk31` with a random ordering and the CG from NAS on a matrix with dimensions close to `bccsstk31`. Once again, the values are relative to CG-U on `bcsstk31` with an RCM ordering on the base architecture at 1000MHz and a 4MB level 3 cache. Values at lower frequencies than 1000MHz are shown for B, LP, MP, and LP+MP configurations with a 256KB level 3 cache.

Even from a cursory look at Figure 6, it is apparent that the sparse matrix structure is critical. When it is near random, there are significant performance degradations despite memory subsystem optimizations at all frequencies. Observe that performance and power profiles of the NAS CG is very similar to that of CG-U on `bcsstk31` with a random ordering and in dramatic contrast to that for CG-O on `bcsstk31` with RCM. The best performing configuration for CG-O RCM, namely (LP + MP), significantly degrades performance for all instances operating on random matrices. For CG-O Random, CG-U Random and CG-NAS, the (LP + MP) configuration performs approximately 20% worse than the base configuration operating at the same frequency. Not surprisingly, CG-O performs worse than CG-U with a random ordering for `bcsstk31` because explicit zeroes added for 2x1 blocking simply increase the number of arithmetic operations without benefiting from the locality of access in the source vector. Although, power savings are possible from DVS and smaller level 3 cache sizes, these come at significant performance degradations for randomly ordered matrices. Consequently, in evaluating the impact of architectural features on power and performance, it is important to use codes representative of the application space of high-performance scientific computing.

4.4 Improving Performance and Power

In the scientific computing community, the primary focus traditionally has been on performance to enable scaling to larger simulations. However, trends in the microprocessor design community indicate that scaling to future architectures will necessarily involve an emphasis on power-aware optimizations. Consequently, it is important to consider sparse computations which occur in a large fraction of scientific computing codes. Such sparse computations are significantly different from dense benchmarks [9, 20] which can effectively utilize deep cache hierarchies and high CPU frequencies. We use CG as an example of such sparse computations to demonstrate that it can benefit from low power modes of the processors and caches and simple memory subsystem optimizations to reduce power and improve performance.

Figure 7 shows relative values of execution time, power and energy for CG-U and CG-O on `bcsstk31` with RCM. The reference point corresponds to the highest frequency 1000MHz for the base architecture with a 4MB level 3 cache, with CG-U; This is the rightmost point in the plots to the left in Figure 7. The plots for CG-U (to the left)

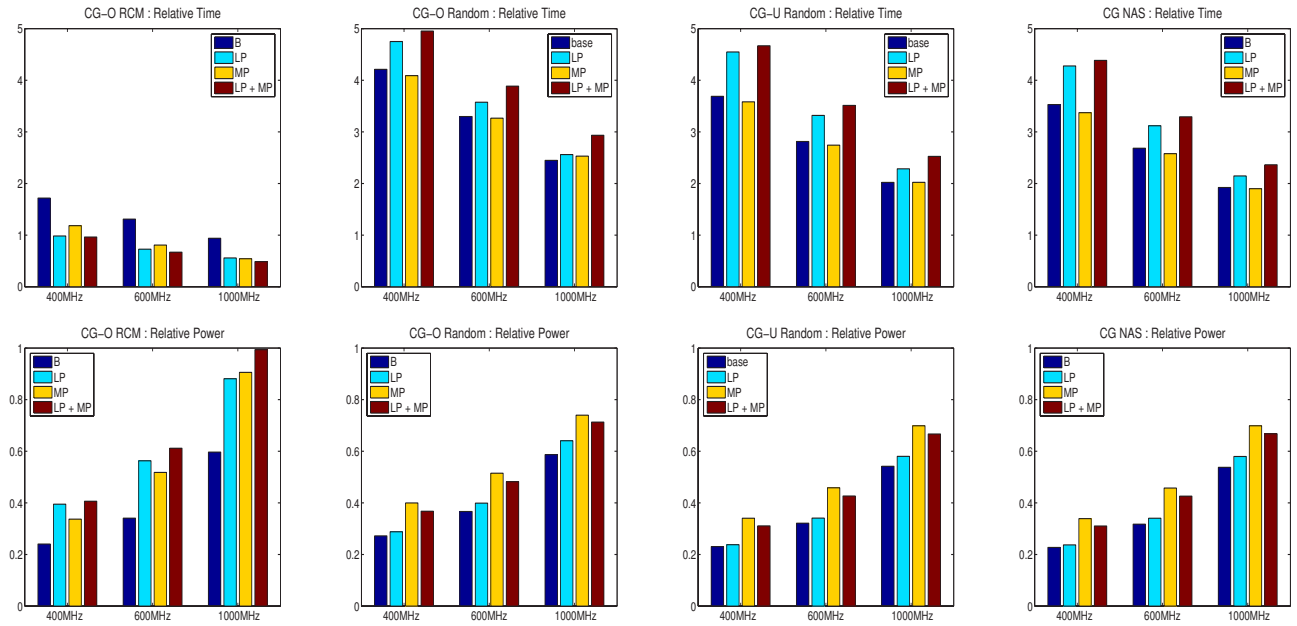

Figure 6. Relative execution time (top row) and relative power (bottom row) for (i) CG-O on bcsstk31 with RCM, (ii) CG-O bcsstk31 with a random ordering, (iii) CG-U on bcsstk31 with a random ordering and (iv) the NAS benchmark CG (CG NAS) on a matrix with dimensions similar to bcsstk31. Values are relative to those of CG-U on bcsstk31 with RCM for the base architecture at 1000MHz and a 4MB L3 cache. At frequencies lower than 1000MHz, values are shown for the base and LP, MP, and LP+MP configurations with a 256KB L3 cache.

correspond to the base architecture without any memory subsystem optimizations, while the plots for CG-O (to the right) correspond to the LP+MP configuration, at different frequencies with a 256KB level 3 cache.

The plots in Figure 7 show that power savings from DVS and low power modes of caches can be realized only at the expense of performance degradations for CG-U on the base architecture. However, the plots for CG-O clearly indicate that a tuned code with memory optimizations can utilize low power modes of caches and DVS for significant improvemtns in both execution time and power. Starting at 400MHz CG-O with LP + MP shows performance improvemnts while reducing power by approximately 60%. Maximum energy savings of 64% are observed at 500MHz with 22% improvements in execution time and 55% improvements in power. Energy savings close to this value can also be observed at 700MHz, with greater improvements (62%) in execution time.

5 Conclusions

In this paper, we have shown how low power modes of processors and caches can be used with simple memory subsystem enhancements to improve the performance and energy characteristics of a sparse CG solver. Furthermore, we show that a tuned code can benefit to a greater degree than an untuned code from the same set of architectural enhancements. For example, the best improvements in execution time observed with LP + MP for CG-U is 45% at 1000MHz while CG-O shows a 55% improvement. Likewise with respect to power, CG-U shows reductions of 52% at 600MHz with a 256KB level 3 cache, with LP + MP; corresponding values for CG-O are 60%.

Our results indicate that power-aware scientific computing with sparse matrix can be achieved without performance degradation. However, interactions between the matrix properties, level of tuning in the code and architectural optimizations impact performance and power profiles. Consequently, we conjecture that using representative tuned codes in addition to more traditional benchmarks will enable a more comprehensive assessment of architectural optimizations for future high end systems.

References

[1] G. Almási, R. Bellofatto, J. Brunheroto, C. Caşcaval, and et al.,. An Overview of the Blue Gene/L System Software Organizaation. In *Lecture Notes in Computer Science, Euro-Par 2003 Parallel Processing: 9th International Euro-Par Conference*, pages 543–555, January 2003.

[2] D. H. Bailey, L. Dagum, E. Barszcz, and H. D. Simon. NAS parallel benchmark results. In *Supercomputing '92: Proceedings of the 1992 ACM/IEEE conference on Super-*

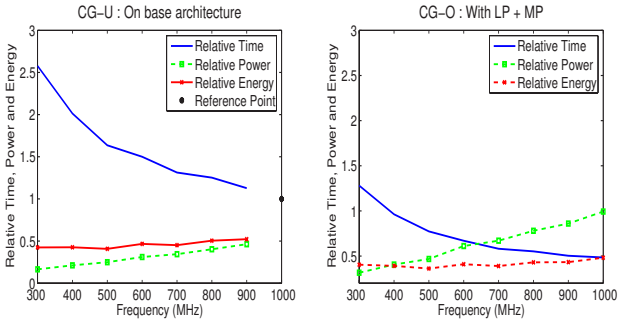

Figure 7. Relative values of Time, Power and Energy are shown with respect to CG-U for bcsstk31 with RCM on the base architecture at 1000MHz with 4MB level 3 cache. Values for CG-U are shown to the left for all frequencies with 256KB level 3 cache on the base architecture, except for the reference point at 1000MHz, shown with 4MB level 3 cache. Values for CG-O are shown to the right for the optimized system with LP + MP at all frequencies with a 256KB level 3 cache. Values smaller than 1 indicate improvements.

computing, pages 386–393, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[3] R. Barrett, M. Berry, T. F. Chan, and et al.,. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, 1994.

[4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *ISCA '00: Proceedings of the 27th annual international symposium on Computer architecture*, pages 83–94. ACM Press, 2000.

[5] D. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *SIGARCH Comput. Archit. News*, 25(3):13–25, 1997.

[6] K. Choi, R. Soma, and M. Pedram. Fine-grained dynamic voltage and frequency scaling for precise energy and performance trade-off based on the ratio of off-chip access to on-chip computation times. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 10004. IEEE Computer Society, 2004.

[7] T. Chung. *Computational fluid dynamics*. Cambridge University Press, 2002.

[8] F. Dahlgren and P. Stenstrm. Evaluation of stride and sequential hardware-based prefetching in shared-memory multiprocessors. *IEEE Trans. on Parallel and Distributed Systems*, 7(4):385–398, April 1996.

[9] J. Dongarra. Top500 list and the LINPACK benchmark. `http://www.top500.org/lists/linpack.php`.

[10] X. Feng, R. Ge, and K. W. Cameron. Power and energy profiling of scientific applications on distributed systems. In *IPDPS'05*, 2005.

[11] J. George. Nested dissection of a regular finite element element mesh. *SIAM J. Numerical Analysis*, 10:345–363, 1973.

[12] J. A. George and J. W.-H. Liu. *Computer Solution of Large Sparse Positive Definite Systems*. Prentice-Hall Inc., Englewood Cliffs, NJ, 1981.

[13] I. D. R. Grimes and J. Lewis. User's guide for the harwell-boeing sparse matrix collection (release i). Technical Report TR/PA/92/86, CERFACS, Toulouse Cedex, France, Oct. 1992.

[14] W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Performance modeling and tuning of an unstructured mesh CFD application. In *Proceedings of SC2000*. IEEE Computer Society, 2000.

[15] E.-J. Im and K. A. Yelick. SPARSITY. `http://www.cs.berkely/yelick/sparsity`.

[16] M. Inc. Micron Technical Note TN-47-04 Calculating Memory System Power for DDR2, 2004. `http://download.micron.com/pdf/technotes/ddr2/TN4704.pdf`.

[17] T. Ishihara and F. Fallah. A non-uniform cache architecture for low power system design. In *ISLPED '05: Proceedings of the 2005 international symposium on Low power electronics and design*, pages 363–368, New York, NY, USA, 2005. ACM Press.

[18] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *ISCA '90: Proceedings of the 17th annual international symposium on Computer Architecture*, pages 364–373, New York, NY, USA, 1990. ACM Press.

[19] N. Kappiah, V. Freeh, D. Lowenthal, and F. Pan. Exploiting slack time in power-aware, high-performance programs. *IEEE/ACM Suptercomputing 2005*, 2005.

[20] C. Lazou. LINPACK results refuel IBM/INTEL chip debate. *HPCWire*, 12(29), July 2003.

[21] W.-F. Lin, S. K. Reinhardt, and D. Burger. Designing a modern memory hierarchy with hardware prefetching. *IEEE Trans. Comput.*, 50(11):1202–1218, 2001.

[22] K. Malkowski, G. Link, P. Raghavan, and M. J. Irwin. Energy-aware memory optimizations for fast sparse scientific computations. Submitted to IEEE Trans. Computer.

[23] S. A. McKee, R. H. Klenke, K. L. Wright, W. A. Wulf, M. H. Salinas, J. H. Aylor, and A. P. Batson. Smarter memory: Improving bandwith for streamed references. *IEEE Computer*, pages 54–63, February 1998.

[24] T. B. Team. An overview of the BlueGene/l Supercomputer. In *SC '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–22. IEEE Computer Society Press, 2002.

[25] E. L. D. E. Todd Austin. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, pages 59–67, February 2002.

[26] S. Toledo. Improving the memory-system performance of sparse-matrix vector multiplication. *IMB Journal of Research and Development*, 41(6):711–72, 1997.