# On Improving Performance and Energy Profiles of Sparse Scientific Applications

Konrad Malkowski, Ingyu Lee, Padma Raghavan and Mary Jane Irwin
Department of Computer Science and Engineering
The Pennsylvania State University
University Park, PA, 16802. TEL. 814-865-9505
E-mail:{malkowsk, inlee, raghavan, mji}@cse.psu.edu

***Abstract*— In many scientific applications, the majority of the execution time is spent within a few basic *sparse* kernels such as sparse matrix vector multiplication (SMV). Such sparse kernels can utilize only a fraction of the available processing speed because of their relatively large number of data accesses per floating point operation, and limited data locality and data re-use. Algorithmic changes and tuning of codes through blocking and loop unrolling schemes can improve performance but such tuned versions are typically not available in benchmark suites such as the SPEC CFP 2000. In this paper, we consider sparse SMV kernels with different levels of tuning that are representative of this application space. We emulate certain memory subsystem optimizations using SimpleScalar and Wattch to evaluate improvements in performance and energy metrics. We also characterize how such an evaluation can be affected by the interplay between code tuning and memory subsystem optimizations. Our results indicate that the optimizations reduce execution time by over 40%, and the energy by over 85%, when used with power control modes of CPUs and caches. Furthermore, the relative impact of the same set of memory subsystem optimizations can vary significantly depending on the level of code tuning. Consequently, it may be appropriate to augment traditional benchmarks by tuned kernels typical of high performance sparse scientific codes to enable comprehensive evaluations of future systems.**

KEYWORDS

benchmarks, memory optimizations, voltage scaling, performance evaluation, application code tuning, power optimizations, sparse matrix kernels

## I. INTRODUCTION

Research in scientific computing algorithms and software is closely aligned with developments in the area of high-performance computing architecture. This alignment is primarily from the necessity of utilizing such architectures effectively to enable knowledge discovery and design through computational modeling and simulation. The latter typically require large, refined models for capturing multiscale, multiphysics phenomena. The limiting factor is often the hardware required to solve the underlying computations with even larger matrices and meshes. Many of the computational models from diverse fields, representing complex multiscale phenomenon are in the form of partial-differential equations(PDEs). The computational simulation of such models has lead to a broad array of new applications involving *sparse matrices and meshes* [21].

In broad terms, architectural optimizations and performance tuning schemes for the more traditional *dense matrix* computations have co-evolved in the last decade, leading to near-peak execution rates for such kernels [1], [12]. Sparse matrix computations differ intrinsically from their dense matrix counterparts in their utilization of architectural features, as discussed later in Section II. As a consequence, they utilize only a fraction of the computing power of modern microprocessors despite sophisticated attempts at performance tuning [28], [30]. This presents a unique opportunity for architectural optimizations as power-aware microprocessor, memory and network design are becoming essential for scaling to future systems [13].

We conjecture that significant advances in high-performance architectures and scientific computing will be possible by considering the co-evolution of architectural optimizations and their interaction with tuned sparse application features. For example, new architectural optimizations can be developed to enable more efficient sparse kernels that better utilize the architecture and thus complete faster. Additionally, utilizing low power modes that are present in many processors, memory (DRAMs), and interconnects can potentially lead to reduced power without significant performance degradation. Taken together, they can enable faster and more efficient solution of larger models while scaling to future power-aware high-performance systems.

In this paper, we evaluate energy-aware architectural optimizations through simulations with SimpleScalar [27] and Wattch [3]. Our goals are to enable more efficient use of

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

the CPU and memory subsystem by sparse matrix kernels. Our results indicate that when these optimizations are used in conjunction with power control modes such dynamic voltage scaling (DVS), we can reduce time by over 60%, and the energy by over 85%. Additionally, we characterize variations in the relative impact of system optimizations on performance and energy metrics from interactions with the level of tuning of the sparse code. We demonstrate that observed relative improvements can vary by over 40% when the same combination of system optimizations is evaluated using different levels of tuning for the sparse kernel.

In Section II we describe the role of sparse matrix kernels in large-scale PDE-based applications and we introduce the codes we will use in our experiments. In Section III we discuss our methodology for evaluating performance and energy through simulation, specific memory subsystem optimizations, and our base RISC PowerPC architecture which is similar to the processor in BlueGene/L [10], the top ranked supercomputer. Section IV contains our main contributions characterizing improvements in performance and energy and differences in relative improvements from the interplay between code and architectural features.

## II. Sparse Matrix Computations In Modeling and Simulation

In recent years, the LINPACK benchmark [12], [22] of *dense* numeric kernels have been accepted as the standard for measuring the efficiency of high-performance architectures for scientific computing. The significance of LINPACK for architecture evaluations lies in the fact that the codes are tuned to include techniques for data-reuse and data-locality [1], [11]. Consequently, when architectural changes aimed at improved memory bandwidth are evaluated, it is important to use LINPACK because it more accurately represents the impact on actual high-performance dense scientific applications.

More recently, there has been a significant growth in computational modeling and simulation applications in which the underlying computations are typically *sparse* [6], [15], [23] and hence can allow scaling to larger and more defined models. However, sparse solution schemes differ from dense kernels in how they utilize architectural features. For example, sparse kernels can utilize only a fraction of the available processing speed because they have a large number of data accesses per floating point operation, and limited data locality and data reuse despite algorithmic changes and considerable tuning of codes through blocking and loop unrolling schemes.

A large fraction of recent research in scientific computing concerns enabling sparse applications through the development of scalable algorithms with tuned implementations in toolkits and libraries [2], [16], [17]. Many of these tuned implementations, rely on a tuned form for *sparse matrix vector multiplication*. Incidentally, this kernel also occurs in several codes in the SPEC CPU 2000 suite [7] such as `mgrid`, `swim`, and `equake`. However, it is not explicitly identified and the implementation may use application specific data structures that are likely not optimized for performance. As architectural changes are optimized for performance and energy, it is especially important to use tuned implementations of such a sparse kernel to accurately represent the space of high performance scientific computing applications. In this paper, we demonstrate the interplay between code optimizations and architectural optimizations by using four forms of the sparse matrix vector (SMV) multiplication kernel.

The general purpose library function forms SMV use standard data structures for storing the sparse matrix $A$, the source vector $x$ and the destination vector $y$. The latter two are stored as simple arrays in contiguous locations in memory. Only the nonzeroes in the matrix and its corresponding indices are explicitly stored using a standard sparse format with a list of subscripts and nonzeroes and a list to index into these two lists for each row. Sparse matrix vector multiplication requires one floating-point multiplication and addition per nonzero element in $A$. Note that in addition to the nonzero element, its indices in the matrix also have to be loaded, thus increasing the number of data accesses per floating point operation. There is potential for re-use with elements of the source vector $x$, but the access pattern on $x$ depends on the sparsity structure of $A$ which can be re-ordered to a 'band form' using, for example, a Reverse Cuthill McKee (RCM) scheme [8] to improve locality of access in $x$. Such re-orderings can be used with other techniques like register-blocking and loop-unrolling to further improve the performance [28], [30]; some of these techniques may actually increase floating-point operations while decreasing loads from memory.

We use SMV-U, a natural implementation of sparse matrix vector multiplication or, equivalently, an untuned version of the code in Sparsity [18]. We use SMV-O from Sparsity [18] with an appropriate level of loop unrolling and register blocking for the best performance on our base architecture, described in Section III. We use the following four sparse

matrices. The name, dimension ($10^3$), number of nonzeroes ($10^6$), and the percentage of nonzeroes relative to a dense matrix of the same dimension are: `bcsstk31`, 35.6, 1.2, .09%; `fdm21`, 32.1, .16, .01%; `qa8fm`, 66.1, 1.6, .03%; and `msc23052`, 23.0, 1.1, .21%. These matrices are first reordered using RCM before applying the two versions of the kernel.

We next consider an application-specific sparse matrix vector multiplication kernel, namely the one in the `equake` code from SPEC CFP 2000 [7]. This application simulates the propagation of elastic waves in large, highly heterogeneous valleys and more than 90% of execution time is spent in its SMV function. The data structure for the matrix is application specific and it reflects the relationship of the matrix to the mesh. The sparse matrix nonzero elements in a row of the matrix typically do not occur contiguously in memory. Furthermore corresponding portions of the source vector may also not be contiguous in memory. We use Equake-A to denote the SMV kernel for this application specific format in `equake`. We made a minor change in this kernel to obtain the tuned version Equake-AT while still using the application specific data structure. The tuning reflects a change in how memory is allocated in the code to ensure a greater degree of contiguous allocations to improve the locality of data accesses. We used Equake-AT in `quake` and we verified that the simulation results were correct and unchanged after the replacement of the original kernel with its modified version. The original data structure and its mapping to memory in Equake-A and the modified mapping to memory in Equake-AT are shown in Figure 1. We would like to observe that we have implemented only a very small modification and we conjecture that the kernel could benefit from the application of further optimizations for increasing instruction level parallelism (ILP) and improving data-reuse.

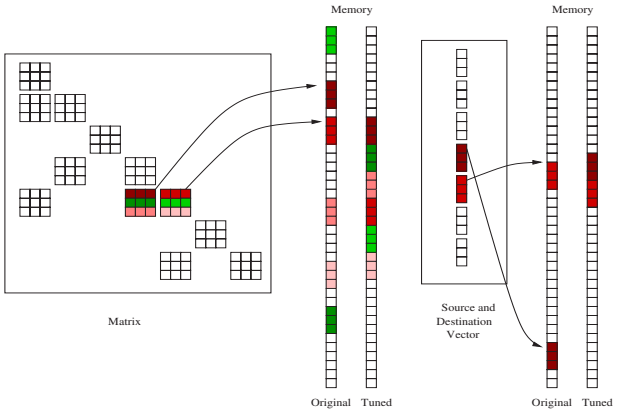


Fig. 1. Mapping of sparse matrix, and source and destination vector elements to memory in Equake-A and an illustration of tuning to improve contiguous allocation of data elements to increase locality of access in Equake-AT.

## III. Modeling Power and Performance Characteristics

We use cycle-accurate emulations of the sparse kernels using SimpleScalar3.0 [4] and Wattch1.02d [3] with extensions to model memory subsystem enhancements [24]. We model a single-core processor with some of the features of the Blue-Gene [26] starting from a PowerPC440 embedded core and including memory subsystem optimizations for prefetching as described in our earlier paper [24].

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. 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). Starting with the base architecture, henceforth denoted by 'B,' we consider first the effects of doubling the width of the data paths, indicated using the label 'W'.

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 [5]. We consider eight CPU frequencies from 300MHz to 1000 GHz with with corresponding nominal $V_{dd}$ voltages in the 0.46V to 1.2V. The SRAM L3 cache may not benefit sparse kernels and and energy efficient alternative could include utilizing power control modes of caches; we simulate this by considering five L3 cache sizes of 256K, 512K, 1MB, 2MB and 4MB. Additionally, we consider the impacts of the memory subsystem optimizations including memory page policy and prefetching at the memory controller and L2 cache.

Memory page policy: open or closed labeled 'MO' or 'MC'. This feature can impact performance depending on the data access pattern and its interaction with data layouts in memory (temporal and spatial locality). The closed page policy is more suitable for random memory accesses, when each access is preceded by an 'activate' operation and followed by a 'precharge' operation [9], [20]. On the other hand, with temporal and spatial locality of data accesses, an open

page policy could reduce latencies, at the expense of greater complexity of the controller. An activated row stays active until a read/write operation to another row in the same bank. The latencies can be reduced to 8 cycles from 16 cycles for successive reads without bank conflicts [19].

Memory prefetching (stride-1) at the memory controller, labeled 'MP'. MP can reduce the effective latency of memory access and it is emulated by adding a prefetch buffer to the memory controller. This buffer is a 16 element table, with each element holding a cache line of 64 bytes or 128 bytes for 'W' and it uses a a full LRU replacement policy. We model the power consumed by our prefetch buffer as the cost of operating a small 16 entry, direct mapped cache with a 64 or 128 byte cache line [24].

Level 2 cache prefetching (stride-1) labeled 'LP'. Once again, this feature can reduce the latency of data access. The extra energy consumption is modeled as second cache access.

## IV. Empirical Results

In this section, we evaluate the impact of memory subsystem optimizations for sparse matrix vector multiplication kernels representing different levels of tuning.

We consider metrics such as execution time and energy, where energy is computed as the system power $\times$ time. Our contributions include the following.

- Characterizing *improvements in time and energy* when memory subsystem optimizations are used in conjunction with DVS and low power modes of caches.
- Characterizing *relative improvements* (RI) starting from the base system for fixed feature sets relative to a fixed base line.
- Modeling *relative incremental improvements* (RII) from adding a feature to the system after a sequence of earlier optimizations. For example, evaluating the incremental impact of adding memory prefetching (MP) for the base system with wider data paths (W) and an open page policy (MO).

As described earlier in Section II, we consider two variants of a general purpose SMV, labeled SMV-U and SMV-O for a total of four matrices with an RCM ordering. We also use the `equake` code from SPEC CFP 2000 with its application specific sparse matrix vector kernel (Equake-A) and with a slightly tuned version of the kernel (Equake-AT). We start with in-depth analysis for SMV-U and SMV-O and conclude with an overview of results for `equake`.

We use several plots in this section with the following general format.

- The X-axis indicates 40 configurations corresponding to distinct frequency and L3 cache size pairs. The X-axis value 1 represents a CPU at 300 MHz with 256 KB L3, the value 2 represents a 300 MHz CPU with a 512KB L3, and so on with 40 representing the 1GHz, 4MB L3 configuration.
- The Y-axis shows either absolute or relative values of metrics such as time and energy and other derived metrics to capture relative improvements.
- Relative values show scaling with respect to a certain fixed point for the *same* kernel. Metrics for a kernel are *not* shown relative to values for a different kernel.
- Plots for base architecture are labeled 'B' and the features include wider data paths (W), open page memory policy (MO), a memory prefetcher (MP), and an L2-prefetcher (LP). When these features are added incrementally starting from the base 'B', the order is shown using labels of the form 'B+W+MO+MP' for base with wider data paths followed by adding an open page memory policy and a memory prefetcher.

We also use stacked and grouped bars to summarize results for a fixed L3 size across frequencies.

### A. Performance and Energy Metrics: Profiles and Summary

Figures 2 and 3 show the execution times and energy for SMV-U (left) and SMV-O (right) when the features are added incrementally in the order 'B+W+MO+MP+LP'. Both sets of plots show reductions in execution time from the optimizations and it is easy to see that both codes could benefit from significant energy savings by using DVS, at improved execution times. Furthermore, at a given frequency for a specific memory subsystem optimization, the L3 cache size has negligible impact on execution time for both codes, thus allowing further energy savings if power saving modes of caches can be utilized. The plots indicate that at wider data paths (W) and the memory open page policy (MO) are particularly useful in reducing both time and energy. It is also interesting to note that SMV-O on the system with all optimizations is faster at even the lowest frequency (300MHz) than for the base configuration at 1GHz.

These plots in Figures 2 and 3 are useful for identifying general trends but they do not give insights into the relative effectiveness of different optimizations. We therefore use the

data presented in these plots to define and compute additional metrics. Consider a specific code such as SMV-U. Let $T_{f,c,q}$ denote the observed execution time, at frequency $f$, cache size $c$ and feature set $q$. We define relative improvement (RI) with respect to $T_{1G,4M,B}$ as:

$$\text{RI}(T)_{f,c,q} = \frac{T_{1G,4M,B} - T_{f,c,q}}{T_{1G,4M,B}}.$$

Using the conventional definition of speedup as $\text{S}(T)_{f,c,q} = \frac{T_{1G,4M,B}}{T_{f,c,q}}$, the metric $\text{RI}(T)_{f,c,q} = 1 - \frac{1}{\text{S}(T)_{f,c,q}}$. Some observations regarding this metric include:

- Values greater than 0 indicate improvements (for example, decreases in time) while negative values indicate degradation (for example, increases in time).
- This metric can also be used to study improvements in energy; we indicate it as $\text{RI}(E)_{f,c,q}$.
- RI values are defined with respect to a specific kernel and hence its base performance or energy values.

Figure 4 shows the RI values for time, and energy for SMV-U at all frequencies for the smallest L3 cache size of 256KB. Observe that in all cases, the improvements for SMV-O are higher than for SMV-U for same configuration. On average with all optimizations, execution time for SMV-O is improved by 48% at average system energy improvements of 85%. Corresponding average improvements for SMV-U are 35% in time and 83% in energy.

### *B. Relative Incremental Improvements (RII): Measuring Incremental Impact per Feature Addition*

The RI values represent speedups for specific configurations relative to a fixed base (at 1GHz, 4MB L3). Thus, they model improvements from the specific set of optimizations as well as from other factors including frequency related scaling and the effect of cache sizes. It would be appropriate to devise a metric that removes the effects of frequency and cache sizes for modeling incremental improvements when the existing configuration is augmented by one more optimization. Otherwise, effects of frequencies and caches may dominate over improvements strictly from the new feature. To model the speedup for adding one optimization $r$ to an existing configuration $q$ (at a specific frequency and cache size), we define the relative incremental improvement (RII) as $\text{RII}(T)_{q+r} = \frac{T_q - T_{q+r}}{T_q}$. If we use the conventional definition of speedup $S_{q+r} = \frac{T_q}{T_{q+r}}$, then it can be seen that $S_{q+r} = \frac{1}{1-\text{RII}(T)_{q+r}}$. For the sequence of optimizations given by $B + W + MO + MP + LP$, it

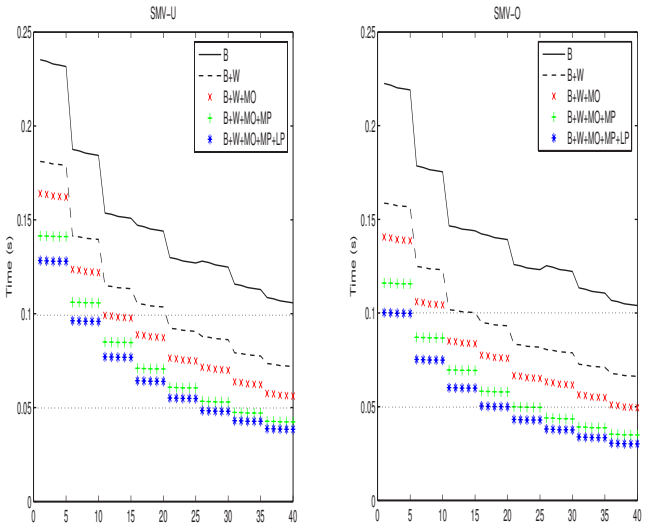


Fig. 2. Execution time in seconds for SMV-U and SMV-O when features are added in the order 'B+W+MO+MP+LP'.

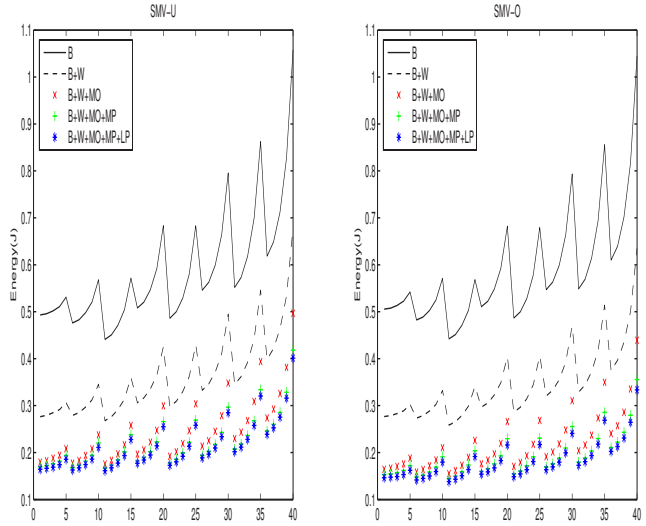


Fig. 3. Energy (J) for SMV-U and SMV-O when features are added in the order 'B+W+MO+MP+LP'.

can be easily seen that the time with all features, namely $T_{B+W+MO+MP+LP}$ is given by subtracting from $T_B$, the time at the base configuration, values of $\text{RII}(T)_{B+W} T_B$, $\text{RII}(T)_{B+W+MO} T_{B+W}$, $\text{RII}(T)_{B+W+MO+MP} T_{B+W+MO}$, and $\text{RII}(T)_{B+W+MO+MPLP} T_{B+W+MO+MP}$.

In the RII definition above, incremental optimizations to the system in a given order are seen as providing incremental speedups; RII values for energy can be defined similarly. Note that these RII values are sensitive to the ordering and they are shown in Figure 5 for SMV-U and SMV-O. These RII values

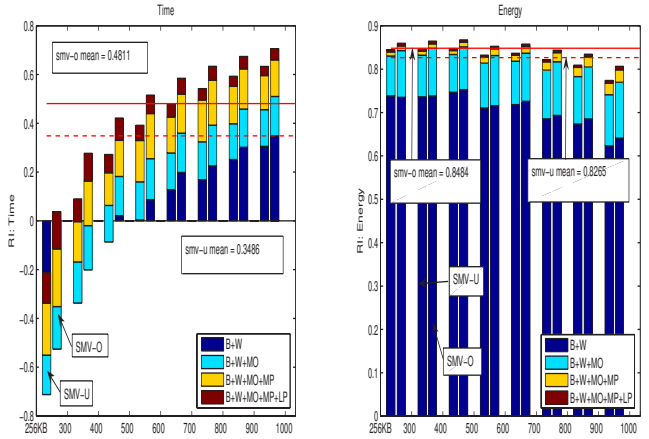


Fig. 4. Relative improvements (RI) in execution time and energy for SMV-U (left bar in pair) and SMV-O (right bar in pair). Values are relative to time and energy values for the base configuration at 1GHz with a 4MB L3 cache. The lines indicate average improvements for the final configuration 'B+W+MO+MP+LP' across frequencies (solid SMV-O, dotted SMV-U).

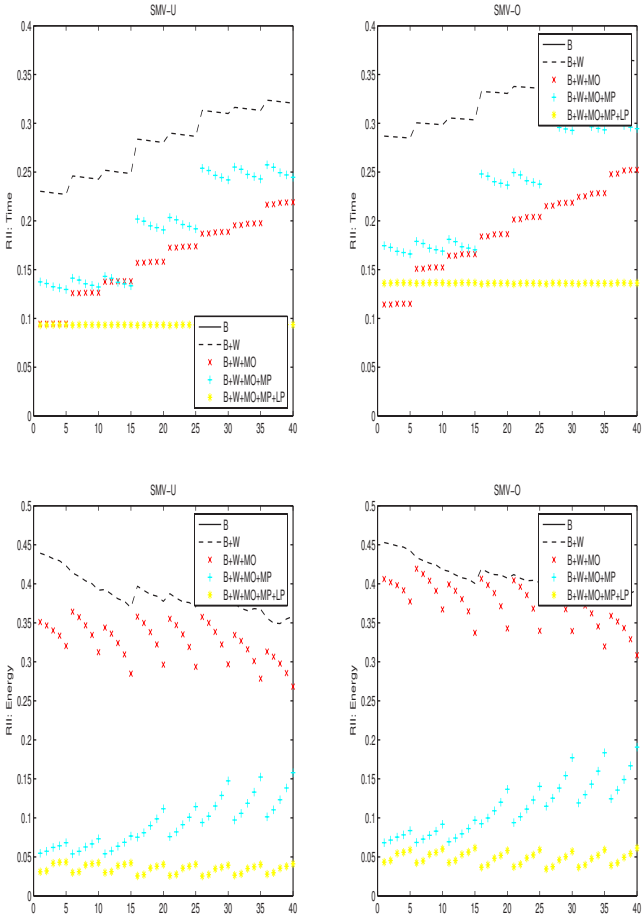


Fig. 5. Relative incremental improvements (RII) in execution time (top) and energy (bottom) for SMV-U (left) and SMV-O (right). Plots correspond to the incremental addition of memory subsystem optimizations in the order: the base configuration (B), adding wider data paths (B+W), an open page policy (B+W+MO), a memory prefetcher (B+W+MO+MP), and an L2-prefetcher (B+W+MO+MP+LP).

indicate the wider data paths have greater impact for SMV-O than for SMV-U and they are the most energy efficient. The open page policy (MO) improves energy efficiency at all frequencies while improving time significantly at higher frequencies. Additionally, the relative impact of MP is more marked in certain frequency ranges than in others. More significantly, all optimizations benefit SMV-O more than SMV-U.

Observe that the plots in Figure 5 indicate significant differences in RII values between SMV-U and SMV-O when the same feature is added to same configuration. These differences are clearly from the difference in the tuning levels of the two codes. Consequently, if RII values are used to select features, then the outcome can be impacted by the choice of kernel. For example, if only features yielding execution time RII values greater than .1 are to selected, then values in Figure 5, indicate that 'LP' will not be selected if SMV-U is used. Likewise, the RII values indicate greater improvements in time and energy with MO for SMV-O. Consequently, design choices could be different depending on the choice of the kernel used in the evaluation.

### C. Results for Equake-A and Equake-AT

We now summarize performance and energy results when evaluations are performed using Equake-A and Equake-AT. Figure 6 shows the plots for execution time and energy for on the base configuration and when all optimizations are applied, i.e., for the configuration $B+W+MO+MP+LP$. Observe that the optimizations result in very small improvements in the execution time of Equake-A while Equake-AT benefits to a larger degree. As mentioned earlier in Section II, the SMV kernel could potentially be tuned further to include features for increasing data-reuse and locality of access. Nonetheless, the slight tuning does enable the code to utilize the memory subsystem optimizations in larger measure than `Equake-A`.

Next, we compute RII values for execution time using Equake-A and Equake-AT; these are shown in Figure IV-C. There is significant divergence in the RII values for the two codes; for example, LP has a negative impact for Equake-A while Equake-AT benefits to a small degree. Once again, these differences arise from the differences in the level of tuning of the SMV kernel representing the dominant computation in `equake`.

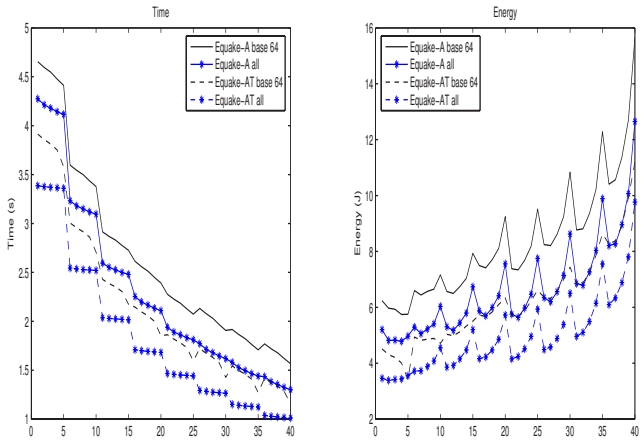


Fig. 6. Execution time (left) and energy (right) for `equake` on the base configuration 'B' and with all optimizations 'all'. Equake-A indicates `equake` with its original SMV and Equake-AT is `equake` with a slightly tuned SMV.

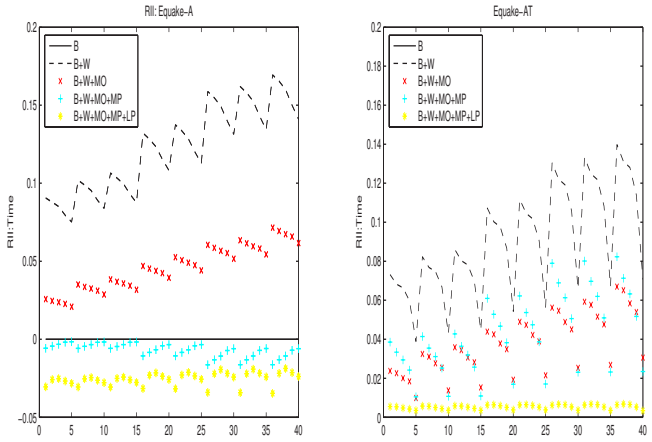


Fig. 7. Relative incremental improvements (RII) in execution time for Equake-A with original SMV (left) Equake-AT with a slightly tuned SMV (right). Different plots correspond to the addition of memory subsystem optimizations, starting with the base configuration (`B`), adding wider data paths (`B+W`), an open page policy (`B+W+MO`), a memory prefetcher (`B+W+MO+MP`), and an L2-prefetcher (`B+W+M0+MP+LP`).

## V. Conclusions

In this paper, we have considered several memory subsystem enhancements for energy-aware high performance sparse computations. These optimizations benefit both the tuned and natural forms of sparse matrix vector multiplication, a function common to many codes in an emerging class of scientific applications. For the untuned kernel, SMV-U, optimizations improve time relative to the base configuration at 1GHz by over 30% starting at frequencies as low as 500MHz with energy reductions of over by over 80%. Corresponding figures for the tuned form, SMV-O, are in excess of 40% for time and 85% for energy. Similarly, the tuned form of `equake` also benefits more from the optimizations. Thus considerable savings in energy are possible with improvements in execution time if memory subsystem optimizations are used in conjunction with DVS and low-power modes of caches. Not surprisingly, tuned kernels realize greater benefits from the memory subsystem optimizations. However, the differences in relative incremental improvements from the same set of optimizations, independent of frequency or cache size effects, are considerable depending on the level of code tuning. These differences indicate a distinct interplay between code and system optimizations.

There is increasing interest in such sparse applications because they allow scaling to solve larger and more refined models. However, tuned implementations representing the types of optimized codes found in high-performance scientific software [14], [21], [25], [29] are typically not available in current benchmark suites. We conjecture that performance analysis with such tuned codes in addition to more traditional benchmarks, will enable a more comprehensive assessment of architectural optimizations for future high end systems.

Our contributions are primarily empirical and our simulation codes and sparse kernels will be available upon request. We also plan to develop and make available to the architecture community, a benchmark suite of sparse kernels that better represent features of scientific applications and are suitable for studying performance and power trade-offs through architectural emulation.

## References

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Publications, 2nd edition, 1995.

[2] S. Balay, K. Buschelman, W. Gropp, and et al.,. The PETSc users manual. Technical report, Argonne National Laboratory, 2004. See `http://www.mcs.anl.gov/petsc`.

[3] David Brooks, Vivek Tiwari, and Margaret 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.

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

[5] Kihwan Choi, Ramakrishna Soma, and Massoud 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.

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

[7] Standard Performance Evaluation Corporation. CFP2000 (Floating Point Component of SPEC CPU2000). http://www.spec.org/cpu2000/CFP2000, 2000.

[8] E. Cuthill. Several strategies for reducing bandwidth of matrices. In D. J. Rose and R. A. Willoughby, editors, *Sparse Matrices and their Applications*, New York, 1972. Plenum Press.

[9] Brian Thomas Davis. *Modern Dram Architectures*. PhD thesis, The University of Michigan, 2001.

[10] Kei Davis, Adolfy Hoisie, Greg Johnson, and et al.,. A performance and scalability analysis of the BlueGene/L architecture. In *SC '04: Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, page 41. IEEE Computer Society, 2004.

[11] J. J. Dongarra, J.D. Croz, S. Hammarling, and I. S. Duff. An extended set of basic linear algebra subprograms. *ACM Trans. Math. Software*, 14:1–17, 1988.

[12] Jack Dongarra. Top500 list and the LINPACK benchmark. http://www.top500.org/lists/linpack.php.

[13] Patrick Gelsinger. GigaScale integration for teraops performance – challenges, opportunities, and new frontiers, 2004. DAC 2004 Keynote. Talk and slides at http://www.dac.com.

[14] J. Glimm, D. Brown, and L. Freitag Diachin. Terascale Simulation Tools and Technologies (TSTT) Center. http://www.tstt-scidac.org, 2004.

[15] 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.

[16] Michael A. Heroux and James M. Willenbring. Trilinos Users Guide. Technical Report SAND2003-2952, Sandia National Laboratories, 2003.

[17] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro. Aztec user's guide: Version 1.1. Technical Report SAND95-1559, Sandia National Laboratories, 1995. See http://www.cs.sandia.gov/CRF/aztec1.html.

[18] Eun-Jin Im and Katherine A. Yelick. SPARSITY. http://www.cs.berkely/yelick/sparsity.

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

[20] Nuwan Jayasena and William J. Dally. Streams and Vectors: A Memory System Perspective. In *MSP-6: 6th WorkShop on Media and Streaming Processors*, 2004.

[21] D. Keyes. Terascale Optimal PDE Simulations (TOPS) Center. http://tops-scidac.org/, 2004.

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

[23] Zi-Kui Liu, Long-Qing Chen, Qiang Du, Stephen Langer, Padma Raghavan, Jorge Sofo, and Christopher Wolverton. Computational tools for multicomponent materials design, 2002–2007. NSF ITR project; see http://matcase.psu.edu/index.html.

[24] K. Malowski, I. Lee, P. Raghavan, and M.J. Irwin. Conjugate gradient sparse solvers: Performance-power characteristics. *High-Performance, Power-Aware Computing Workshop in conjunction with IDPSD2006*, 2006.

[25] NERSC. Performance Evaluation Research (PERC) Center. http://perc.nersc.gov/main.htm, 2004.

[26] The Bluegene 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.

[27] Eric Larson Dan Ernst Todd Austin. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, pages 59–67, February 2002.

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

[29] U. S. Dept. of Energy. Scientific Discovery through Advanced Computing (SciDAC) Initiative. http://www.osti.gov/scidac/, 2003.

[30] Richard Vuduc, James W. Demmel, Katherine A. Yelick, Shoaib Kamil, Rajesh Nishtala, and Benjamin Lee. Performance optimizations and bounds for sparse matrix-vector multiply. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–35. IEEE Computer Society Press, 2002.