2013 IEEE 27th International Symposium on Parallel & Distributed Processing Workshops and PhD Forum

Energy-efficient Sparse Matrix Autotuning with CSX - A Trade-off Study

Jan Christian Meyer
HPC Section, IT Dept.
NTNU, Norway
Jan.Christian.Meyer@ntnu.no

Juan Manuel Cebrian
Lasse Natvig
Dept. of Computer and Info. Science (IDI)
NTNU, Norway
{juanmc, lasse}@idi.ntnu.no

Vasileios Karakasis
Dimitris Siakavaras
Konstantinos Nikas
School of ECE
NTUA, Greece
{bkk, jimsiak, knikas}@cslab.ece.ntua.gr

***Abstract*—In this paper, we apply a method for extracting a running power estimate of applications from hardware performance counters, producing power/time curves which can be integrated over particular intervals to estimate the energy consumption of individual application stages. We use this method to instrument executions of a conjugate gradient solver, to examine the energy and performance impacts of applying the Compressed Sparse eXtended (CSX) and classic Compressed Sparse Row (CSR) matrix compression methods to sparse linear systems from different application areas. The CSX format requires a preprocessing stage which identifies and exploits a range of matrix substructures, incurring a one-time cost which can facilitate more effective sparse matrix-vector multiplication (SpMV). As this numerical kernel is the primary performance bottleneck of conjugate gradient solvers, we take the approach of isolating the energy cost of preprocessing from a short sample of application iterations, obtaining measurements which enlighten the choice of which compression scheme is more appropriate to the input data. We examine the impact variable degrees of parallelism, processor clock frequency, and Hyperthreading have on this trade-off. Our results include comparisons of empirically obtained results from all combinations of up to 8 threads on 4 hyperthreaded cores, 3 clock frequencies, and 5 sample application matrices. We assess program-hardware interactions with views to structural properties of the data and hardware architectural features, and evaluate the approach with respect to integrating the energy instrumentation with present automatic performance tuning. Results show that our method is sufficiently precise to identify non-trivial tradeoffs in the parameter space, and may become suitable for a run-time automatic tuning scheme by applying a faster preprocessing mode of CSX.**

I. Introduction

Dynamic power has been the dominant source of power dissipation in CMOS devices for several years. Its quadratic relation to the supply voltage has led to considerable savings in power consumption as new technology allowed the safe reduction of the former. The clock frequency has a double effect on the dynamic power dissipation: apart from its direct relation to the total dissipated power, sustaining a higher frequency often requires a proportionally higher supply voltage, leading to a cubic relation between frequency and dynamic power [1], [2]. This significant impact of frequency on the total dissipated power has led to the wider adoption of a dynamic management of voltage and frequency, in order to reduce the overall power dissipation without compromising performance.

DVFS techniques allow the selective scaling of both voltage and frequency in less CPU-intensive phases of an application, *e.g.* memory-bound or latency-tolerant parts, in order to achieve high performance at a lower power budget [3], [4]. More advanced techniques separate the processor in multiple independent domains of voltage control, allowing the selective scaling of voltage in different processor components depending on the needs of the running application [5], [6], [7]. To tackle the leakage power problem efficiently, modern microarchitectures incorporate advanced power management with the ability for example to dynamically 'shut down' idle cores.

At the same time, the integration of multiple fast cores into the same processor socket has placed a significant burden to the memory subsystem. Both in terms of latency and bandwidth, the main memory cannot sustain the very high rates at which a modern processor can consume data, and this difference tends to grow exponentially with every new processor generation [8]. Therefore, large caches come to fill in and hide this performance gap, and it is not extraordinary to encounter today caches as large as 24 MB per socket. However, such large caches take up a significant portion of the total die area and, as a result, are chief contributors to the processor's total leakage power[1] [2]. Several techniques have been proposed for reducing the leakage power loss from caches, ranging from specific memory cell technologies [9] to techniques for shutting down whole unused portions of the cache [10] or specific unused cache lines [11], [12]. Such techniques are soon becoming mainstream and allow energy-efficient designs for caches even exceeding 20 MB.

Similarly, software applications and techniques can play a significant role in reducing the overall power dissipation of the system. In this work we focus on data compression schemes that are used to reduce the cache accesses, and therefore the power footprint of the caches. More specifically, we evaluate the energy efficiency of CSX [13], a storage format for sparse matrices that aims to minimize the memory footprint of the matrix and as a result to achieve better cache utilization.

[1]The dynamic power dissipation of very large caches is not so significant, since they usually reside in a different voltage and frequency domain than the rest of the processor, running at much lower frequencies.

978-0-7695-4979-8/13 $26.00 © 2013 IEEE
DOI 10.1109/IPDPSW.2013.219

IEEE computer society

A. Related Work

The dynamic power of a processor is directly affected by the utilization of its units during the different phases of an application. In power terms, the utilization of a hardware unit can be viewed as an indication of its activity factor. Based on this assumption, Isci and Martonosi [14] build a quite accurate model for predicting the power dissipation of processors based on the Intel Netburst microarchitecture [15]. The authors infer the activity factor of 22 hardware components (caches, TLBs, execution units etc.), by calculating their utilization using hardware performance counters, and use a weighted sum of the per-component power predictions, in order to calculate the overall dynamic power dissipation of the processor. A more detailed and generic approach in predicting the power dissipation of a modern processor is the McPAT power model framework [16]. McPAT also predicts power dissipation through the use of performance monitoring events, but conversely to the model in [14], it is not bound to a specific architecture technology. Instead, it uses low-level power models for modeling the behavior of fundamental components of a chip multiprocessor, allowing the exploration of design tradeoffs for new architectures.

Collecting a multitude of performance monitoring events in order to obtain an accurate power estimate might require multiple runs of the profiled application, since the number of performance monitoring registers is rather restricted in current microarchitectures. As a result, such methods are not suitable for online power estimation of a running application. Curtis-Maury et al. [17] take a hybrid offline/online approach for predicting the execution configuration (thread count and core frequency) that provides an optimal balance between performance and energy consumption. The proposed runtime system collects performance monitoring information from a running application and feeds a regression model trained offline, which predicts the optimal configuration for the next phase of the application. In effect, this technique intends to allocate the minimum of resources (frequency, thread count) to an application, so that it proceeds at a low power budget without compromising performance. A similar hybrid approach is adopted also by Singh et al. [18] in a power-aware thread scheduling scheme that tries to keep the overall system power dissipation within a user-specified envelope.

While software control over dynamic power consumption is feasible, thanks to smart use of frequency scaling and thread placement, software control of leakage power is not yet widely adopted. In order to achieve this, the software must be able to 'switch off' specific functional units of the processor (ALU, FPU etc.) or deactivate unused parts of the cache, depending on its own needs. Assuming a hardware support for such fine-grained leakage power control, Zhang et al. [19] and You et al. [20] explore compiler techniques for identifying basic blocks where a specific functional unit is unused and instrument the resulting code with explicit activation/deactivation instructions for controlling the state of a functional unit. Similarly, Zhang et al. [21] assume a fine-grained control of the instruction cache lines and propose compiler optimization techniques for selectively putting cache lines into a low leakage mode.

CPU model	Intel(R) Core(TM) i7-2600
Model #	42
Stepping	7
Manufacturing process	32nm
Clock frequency (min-max)	1.60GHz - 3.40GHz
Number of physical cores	4
Number of logical cores	8
Main memory	16GB

TABLE I
TEST SYSTEM SPECIFICATIONS

B. Background

CSX is a compact storage format for sparse matrices that is able to detect and encode in a single representation a diverse set of matrix substructures. A *substructure* is any regular one- or two-dimensional sequence of non-zero elements inside the sparse matrix. The number of all the possible substructures detected from CSX is indefinitely large and a priori unknown for a specific sparse matrix. For this reason, CSX employs runtime code generation in order to provide high-performance SpMV implementations adapted to the specificities of every matrix. Compared to other storage formats that exploit a single substructure type, e.g., the BCSR format that exploits only dense block substructures [22], CSX is able to achieve consistent high performance by successfully adapting to a great variety of sparse matrices, ranging from regular ones to those with a rather irregular structure.

Supported substructures are horizontal, vertical, diagonal, anti-diagonal, and two-dimensional (row- or column-aligned).

CSX/CG execution proceeds in 5 phases:

1) Matrix loading (either from disk or from CSR)
2) Detection of substructures
3) Selection and encoding of substructures
4) Code generation
5) Iteration, bounded by SpMV performance

All steps except iteration are performed once. Iteration to solution for a system requires a number of repetitions which is proportional to the rank of the system, potentially amortizing the cost of one-time preprocessing steps by a reduction in the repeated cost of iterations. The substructure detection, encoding and iteration phases are parallelized using POSIX threads. Affinity of threads to cores is controlled using the *numactl* library interface.

II. Experimental Methodology

A. Test System

All tests are performed on a quad-core machine featuring a 4-core processor with the Sandy Bridge architecture, admitting explicit control of clock frequency, and energy profiling using RAPL MSR features, as described below. The specifications of the test system are tabulated in Tab. I.

System	Rank	Type
offshore	259.789	Irregular
parabolic_fem	525.825	Irregular
thermal2	1.228.045	Irregular
af_5_k101	503.625	Regular
boneS10	914.898	Regular

TABLE II
TEST MATRICES

B. Test Problems

Five test matrices from the University of Florida Sparse Matrix Collection [23] are chosen in order to display the effects of differing sizes and regularity. All are symmetric and positive-definite, to capture the real application behavior of a conjugate gradient solver, although our tests will not require the systems to be iterated to convergence. The matrices are broadly classified as regular or irregular, based on the extent to which their nonzero components are regularly arranged in patterns which make them suitable for compression. Table II summarizes the matrices, and their key characteristics.

C. CPU Power Instrumentation

Intel®microarchitectures starting with Sandy Bridge expose Model Specific Registers (MSRs) which provide a running estimate of the energy consumption of the processor. We use a function library developed at NTNU [24] in order to read these for continuous intervals of approximately 0.1 seconds.

Interval timing is performed using the Linux kernel real-time clock through the *setitimer* system call, which produces a periodic call to a registered signal handler. The *cg* solver of CSX was augmented with a signal handler function to trap this event, and print the running energy status to log files, marking time using the standard *gettimeofday* system call.

Use of the latter clock is necessary because it displays a slight drift from the kernel timers, but is used by the energy counter library in order to convert MSR-internal register values to milliJoules. The magnitude of this drift was determined to remain below $0.01s$; as this accounts for only 10% of the energy timer's interval, our further discussion is restricted to the time series recorded using *gettimeofday*.

Our periodic recording of energy use over intervals provides a discretized, running estimate of CPU power, which we approximate with a simple linear model, dividing the energy per interval by the recorded time step. This permits us to extract power/time graphs throughout entire CSX runs, which can be related to its stages of execution. Energy consumption is then estimated by numerical integration of this series, using the trapezoid method, in accordance with our linear approximation of energy variations.

In order to relate the power consumption to the stages of execution that cause it, both the *cg* program and the SpMV-related internal routine which reads and compresses matrices were extended to log the beginning and end of these phases, by causing the running program to signal itself in a similar manner to the interval timer. Power graphs can thus be partitioned into distinct energy totals for the phases of

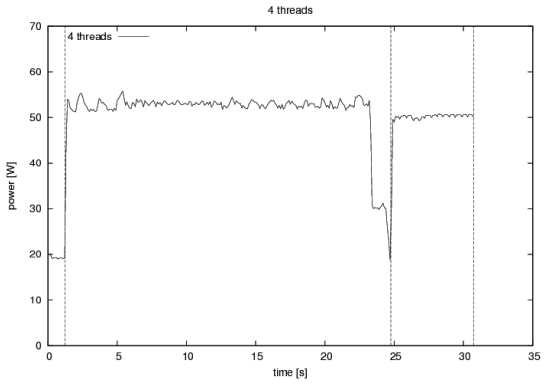

Fig. 1. CSX format with full compression, at 3.4 GHz using 4 threads

1) an initial I/O stage caused by reading the matrix file,
2) preprocessing and matrix compression, and
3) a fixed number of iterations

with the preprocessing stage encompassing phases 2, 3, and 4 of CSX execution, as described in Section I-B.

The number of iterations is fixed at 1024. This is far from sufficient to bring either of our test systems to convergence, but it suffices to reveal a steady-state power consumption in the iteration phase, which admits estimation of costs per iteration, and hence, the trade-off between preprocessing and computation cost.

Space does not permit a complete exposition of our captured execution traces, but as an example, Fig. 1 displays the power/time curve of a *parabolic_fem* run with 4 threads at 3.4GHz. The vertical lines mark the boundaries between initialization, preprocessing, and iteration phases captured in the event log. Using these to indicate integration intervals enables us to compute energy costs of $17.659J$ for the initialization stage, $1115.753J$ for preprocessing, and a $257.366J$ iteration cost which suggests a per-iteration energy of $0.251\frac{J}{iter}$ for this run.

Because the power figures are derived from energy counters which do not discriminate the running program, our method relies on exclusive access to the test system, as any context switches produce altered regions or spikes in power consumption, in addition to their well-known impact on overall run time. For the same reason, it also includes the power consumption of the operating system. Experiences from preliminary testing suggest that the running power estimate could form an effective estimator of detecting context switches and unmaskable interrupts, but exploring that direction is beyond the scope of this paper. For our purposes, we will compute energy estimates based on runs showing the characteristics of Fig. 1 without distinguishing O/S jitter, on the argument that such a level of noise is natural to expect also for actual application execution, and that they remain representative insofar as they are not markedly dominated by interference.

The extraction of energy counter values from the CPU implies that the energy figures we report are only for part of the total system power budget. Although arguably the most signif-

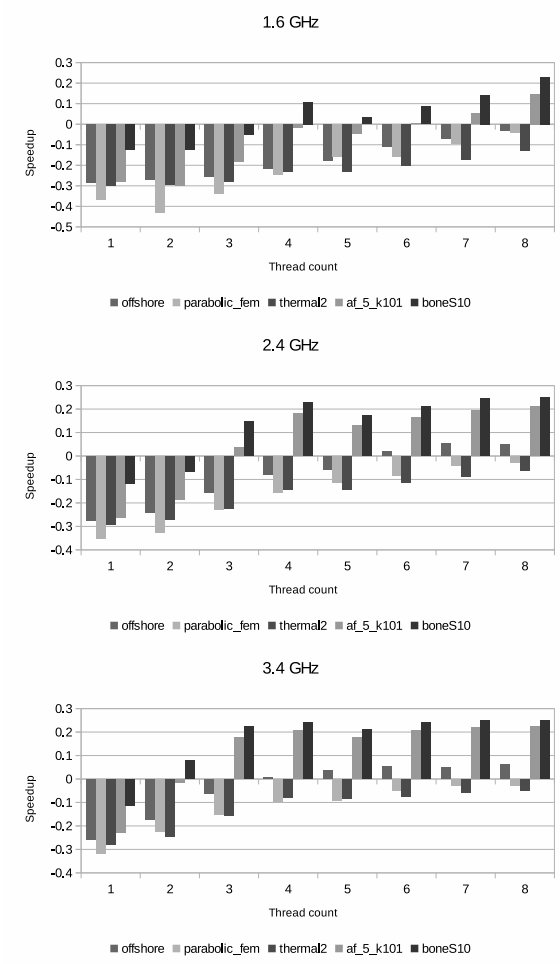

Fig. 2. CSR/CSX iteration phase speedup, offset by -1

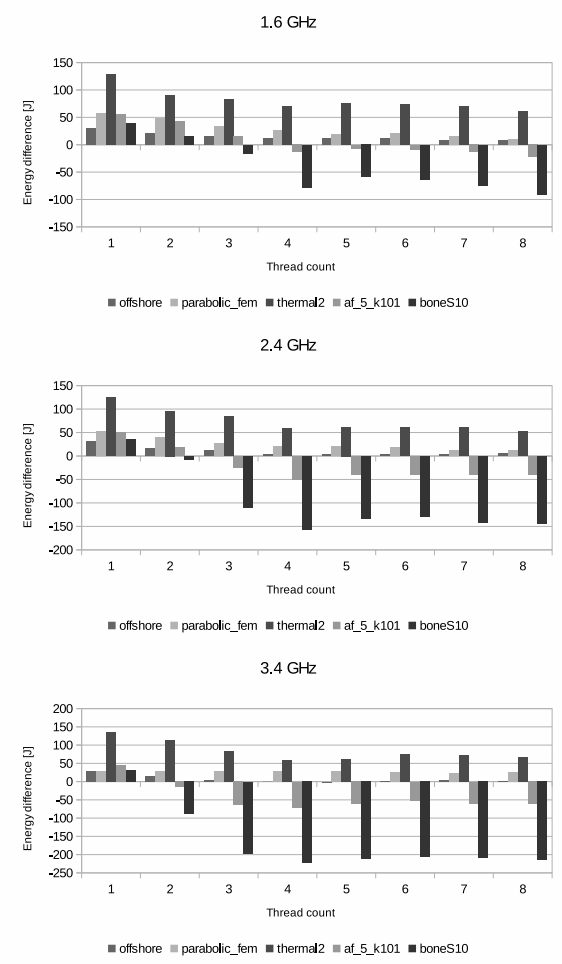

Fig. 3. CSX-CSR iteration phase energy differences

icant single component to consider, it should be noted that the considerations we provide pertain to CPU energy only, which overlooks any impact of the memory subsystem. Because the performance improvement of compression is due to conserving memory bandwidth by fitting greater parts of compressed matrices on chip, we expect that including memory traffic would further amplify the difference between the cases where compression is more and less effective, and taking CPU energy consumption as a representative component. We note that none of the test matrices fit in the last level cache memory of the test system, and that the computation is memory-bound. A sustained, constant memory traffic would account for a constant power consumption, which has appeared to be present in preliminary experiments instrumenting full system power. This creates the expectation that including memory traffic costs would produce similar results to within a constant factor, but validating this assumption is beyond the scope of the present study.

III. Results and Discussion

Figures 2 and 3 show timing and energy results captured from test runs of all matrices, for 1 through 8 threads, pinning clock frequency at 1.6GHz, 2.4GHz and 3.4GHz. Figure 2 represents the speedup of CSX relative to CSR for the fixed iteration count, with otherwise identical thread counts and frequencies. The ratio is displayed with an offset of -1, to center the relationship between the two compression schemes around the horizontal axis.

First and foremost, the results provide clear evidence of our expectation that the speedup attainable by compressing matrices with regular structure also translates into reduced energy consumption. The three irregular matrices *offshore*, *parabolic_fem* and *thermal2* attain neither speedup nor energy gains from increased parallelism, suggesting that within our present parameter space, CSR compression is favorable for these. For the regular matrices *af_5_k101* and *boneS10*, speedup gains are attained up to 4 threads, corresponding to fully populating the physical cores of the test system. We note that the relative benefit of CSX not only provides faster

execution with increasing clock frequencies, but also a higher benefit relative to CSR. The most pronounced effect of this is visible in the fully populated 4-thread case for *boneS10*; the attainable speedup over CSR saturates at 2.4GHz, but further increasing the clock frequency still translates to further reductions in CPU energy consumption. It is interesting to note that for each of the tested clock frequencies, there are points where a reduction in energy requirements is attained before speedup is achieved, as witnessed for *boneS10* with 3 threads at 1.6GHz, 2 threads at 2.4GHz, and for *af_5_k101* with 2 threads at 3.4GHz.

The 5-8 thread results should be viewed with the awareness that they are obtained using hyperthreaded execution, which affects their characteristics. With multiple threads sharing physical resources, dividing the workload into even logical partitions causes an imbalance in the utilization of the physical cores, as some cores are subjected to any contention issues related to handling two affine tasks, while the others are not. This results in a transition area where additional parallelism is detrimental to overall efficiency. The effect is most pronounced for *boneS10* in the 1.6GHz results, where it can be seen in contrast to the *af_5_k101* results showing improvements in speedup and reductions in energy throughout. Approaching full utilization of physical resources this effect diminishes, ending with the best speed and energy at full utilization. The corresponding 2.4GHz results show the 4-thread case producing slightly greater energy savings with similar speed improvement, suggesting that hyperthreading provides no improvement at this frequency.

Empirically observing that these variations are strongly dependent on the interplay between the architectural features of the test system and the structural properties of input data, suggests that there is a relationship between the optimal energy savings and program configuration which is non-trivial to derive from a priori application knowledge. This presents the trade-off of how extensive the investment required to determine favorable parameters can be before breaking even with the attainable benefit.

Fig. 4 shows our experimentally observed preprocessing cost figures in terms of how many CSR iterations they equate to with respect to energy. The irregular matrices all show a fairly low count, but the significance of testing this in an automatic tuning scenario would be to conclude that the resulting iteration phase is the less desirable option. The regular matrices for which the preprocessing is effective show relatively high counts. In practice, this cost is quite important, as it would form part of the profitability analysis of an automatic tuning approach. As the cost of sampling to find ideal parameters must not exceed the gain in a run-time scenario, our present approach of testing all combinations of parameters would apply to *e.g.* generate an offline database with characteristics of known matrices.

To further illustrate this trade-off under the assumption that a suitable subset of parameter combinations could be identified from architecture-dependent tendencies such as our hyperthreading observation, Fig. 5 plots the preprocessing

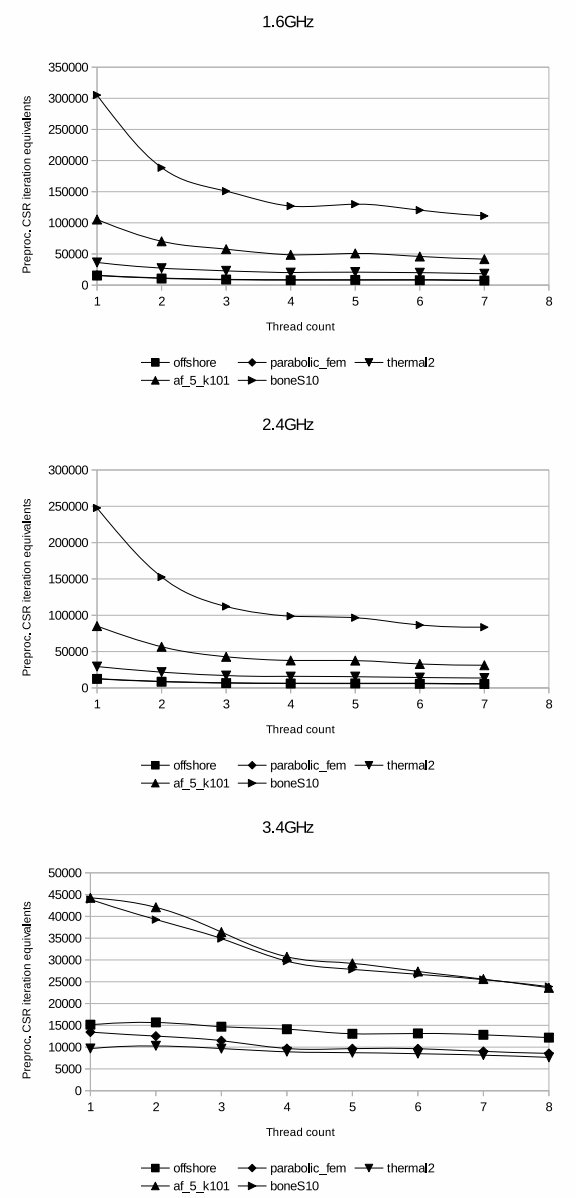

Fig. 4. Preprocessing energy as iteration equivalents

cost for the regular matrix cases with speedup, in terms of the energy gained per iteration. These high numbers of iterations required to amortize the preprocessing cost suggest that restricting the parameter space may not suffice to make the analysis of multiple combinations feasible for an online solution without matching reductions in the preprocessing cost.

We expect that applying the statistical preprocessing features of CSX can provide such reductions, but evaluating this is beyond the scope of investigating energy and performance trade-offs, leaving it as an interesting direction for extension of this work.

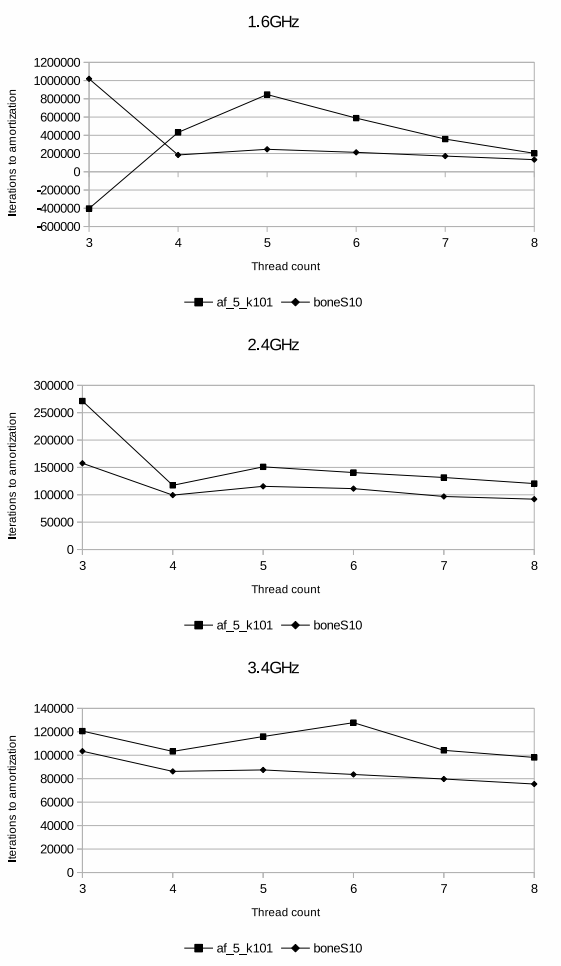

Fig. 5. Iteration count to amortization of preprocessing cost

IV. Conclusions and Future Work

In this paper, we have used energy counter features of the Intel Sandy Bridge architecture to investigate performance and energy impacts of the CSX auto-tuning matrix compression for sparse matrix vector multiplication, contrasting it to the conventional CSR method using a conjugate gradient solver for benchmarking purposes. Our results showed that the structural properties of input matrices determine which method is favorable for speed and energy gains. Energy consumption proved predominantly tied to execution speed, but some transition cases also showed reductions in energy consumption using the slower compression method. Variations in the level of threading and processor clock frequency showed that increasing the level of parallelism provides improvements in both speed and energy up to the number of physical cores on chip. The optimal decision on whether or not to apply Hyperthreading differed with variations in clock frequency.

Evaluating the applied method with respect to its suitability for incorporating energy metrics in the automatic tuning of CSX, we found that our developed method is presently suitable for offline analysis only. An immediate and interesting direction for extending this work would be to assess the effectiveness of applying statistical preprocessing features of CSX, as this is expected to lower the cost of preprocessing. It would also be most interesting to instrument full system power, to relate the significance of the memory system and other components to CPU-based energy measurements we have presented. Finally, as CSX is not restricted to conjugate gradients, but generally addresses sparse matrix-vector products, researching its impact on further applications is relevant for future work.

Acknowledgment

The authors gratefully acknowledge the support of the PRACE 2IP project.

References

[1] D. M. Brooks, P. Bose, S. E. Schuster, H. Jacobson, P. N. Kudva, A. Buyuktosunoglu, J.-D. Wellman, V. Zyuban, M. Gupta, and P. W. Cook, "Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors," *IEEE Micro*, vol. 20, no. 6, pp. 26–44, 2000.

[2] S. Kaxiras and M. Martonosi, *Computer Architecture Techniques for Power-Efficiency*, ser. Synthesis Lectures on Computer Architecture. Morgan & Claypool, 2008, vol. 3.

[3] F. Xie, M. Martonosi, and S. Malik, "Compile-time dynamic voltage scaling settings: Opportunities and limits," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming Language Design and Implementation*. San Diego, CA, USA: ACM, 2003, pp. 49–62.

[4] K. Choi, R. Soma, and M. Pedram, "Dynamic voltage and frequency scaling based on workload decomposition," in *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*. Newport, CA, USA: ACM, 2004, pp. 174–179.

[5] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*. Cambridge, MA, USA: IEEE, 2002, pp. 29–40.

[6] S. Herbert and D. Marculescu, "Analysis of dynamic voltage/frequency scaling in chip-multiprocessors," in *ACM/IEEE International Symposium on Low Power Electronics and Design*. Portland, OR, USA: IEEE, 2007, pp. 38–43.

[7] T. G. Mattson, M. Riepen, T. Lehnig, P. Brett, W. Hass, P. Kennedy, J. Howard, S. Vangal, N. Borkar, G. Ruhl, and S. Dighe, "The 48-core SCC processsor: the programmer's view," in *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC'10)*. New Orleans, LA, USA: ACM, 2010, pp. 1–11.

[8] A. W. Wulf and S. A. McKee, "Hitting the memory wall: implications of the obvious," *ACM SIGARCH Computer Architectures News*, vol. 23, no. 1, 1995.

[9] M. Powell, S.-H. Yang, B. Falsafi, K. Roy, and T. N. Vijaykumar, "Gated-Vdd: a circuit technique to reduce leakage in deep-submicron cache memories," in *Proceedings of the International Symposium on Low Power Electronics and Design*. Rapallo, Italy: ACM, 2000, pp. 90–95.

[10] S. Yang, M. D. Powell, B. Falsafi, K. Roy, and T. N. Vijaykumar, "An integrated circuit/architecture approach to reducing leakage in deep-submicron high-performance I-caches," in *Seventh International Symposium on High-Performance Computer Architecture*. Monterey, Mexico: IEEE, 2001, pp. 147–157.

[11] S. Kaxiras, H. Zhigang, and M. Martonosi, "Cache decay: Exploiting generational behavior to reduce cache leakage power," in *28th Annual International Symposium on Computer Architecture*. Göteborg, Sweden: IEEE, 2001, pp. 240–251.

[12] K. Flautner, K. Nam Sung, S. Martin, D. Blaauw, and T. Mudge, "Drowsy caches: Simple techniques for reducing leakage power," in *29th Annual International Symposium on Computer Architecture*. Anchorage, AL, USA: IEEE, 2001, pp. 148–157.

[13] V. Karakasis, T. Gkountouvas, K. Kourtis, G. Goumas, and N. Koziris, "An extended compression format for the optimization of sparse matrix-vector multiplication," *IEEE Transactions on Parallel and Distributed Systems*, (to appear).

[14] C. Isci and M. Martonosi, "Runtime power monitoring in high-end processors: Methodology and empirical data," in *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture*. San Diego, CA, USA: IEEE Computer Society, 2003.

[15] D. Koufaty and D. T. Marr, "Hyperthreading technology in the Netburst microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 56–65, 2003.

[16] L. Sheng, H. A. Jung, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *42nd Annual IEEE/ACM International Symposium on Microarchitecture*. New York, NY, USA: IEEE, 2009, pp. 469–480.

[17] M. Curtis-Maury, A. Shah, F. Blagojevic, D. S. Nikolopoulos, B. R. de Supinski, and M. Schulz, "Prediction models for multi-dimensional power-performance optimization on many cores," in *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques*. Toronto, Ontario, Canada: ACM, 2008, pp. 250–259.

[18] K. Singh, M. Bhadauria, and S. A. McKee, "Real time power estimation and thread scheduling via performance counters," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 2, pp. 46–55, 2009.

[19] W. Zhang, M. Kandemir, N. Vijaykrishnan, M. J. Irwin, and V. De, "Compiler support for reducing leakage energy consumption," in *Conference and Exhibition of Design, Automation and Test in Europe*. Munich, Germany: IEEE, 2003, pp. 1146–1147.

[20] Y.-P. You, C. Lee, and J.-K. Lee, "Compilers for leakage power reduction," *ACM Transactions on Design Automation of Electronic Systems*, vol. 11, no. 1, pp. 147–164, 2006.

[21] W. Zhang, J. S. Hu, V. Degalahal, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, "Compiler-directed instruction cache leakage optimization," in *35th Annual IEEE/ACM International Symposium on Microarchitecture*. Istanbul, Turkey: IEEE, 2002, pp. 208–218.

[22] E.-J. Im and K. A. Yelick, "Optimizing sparse matrix computations for register reuse in SPARSITY," in *Proceedings of the International Conference on Computational Sciences – Part I*. Springer-Verlag, 2001, pp. 127–136.

[23] "The University of Florida sparse matrix collection," 2013, [Online; http://www.cise.ufl.edu/research/sparse/matrices/, accessed 21-January-2013].

[24] H. Lien, L. Natvig, A. A. Hasib, and J. C. Meyer, "Case studies of multi-core energy efficiency in task based programs," in *Proceedings of ICT as Key Technology against Global Warming (ICT-GLOW)*, ser. Lecture Notes in Computer Science, vol. 7453. Springer-Verlag, 2012, pp. 44–54.