Exploring the Performance-Energy Tradeoffs in Sparse Matrix-Vector Multiplication

Vasileios Karakasis   Georgios Goumas   Nectarios Koziris
National Technical University of Athens
{bkk,goumas,nkoziris}@cslab.ece.ntua.gr

Abstract

The Sparse Matrix-Vector Multiplication (SpMV) kernel scales poorly on modern multicore architectures due to its high demands on memory bandwidth. Adding more threads and keeping the processor frequency high might not attain the expected performance benefits, therefore leading to a waste of energy. In this paper, we explore the performance-energy landscape of the SpMV kernel by examining the impact of frequency scaling and thread placement on the performance and the energy requirements of the SpMV kernel. We use a simple power metric specifically targeted to the streaming nature of SpMV to rank the different execution configurations in terms of power dissipation, and use the notion of Pareto optimality from the theory of multi-objective optimization to analyze and characterize the tradeoffs. Finally, we propose a prediction model based on machine learning, which is capable of accurately predicting the execution configurations leading to the best performance-energy tradeoffs for SpMV execution.

1. Introduction

Sparse Matrix-Vector Multiplication (SpMV) is one of the most significant and widely used scientific computational kernels arising in a variety of application domains. Its optimization has drawn the interest of the research community over the years and a variety of optimization techniques have been proposed [8, 9, 15]. The key performance characteristic of SpMV on modern multicore machines is its high demands on memory bandwidth [4, 16], which prevent it from scaling to multiple cores, especially when the size of the sparse matrix exceeds the aggregate cache size of the underlying architecture. Considerable work has been done toward the direction of minimizing the memory footprint of SpMV [8, 9, 15]. Kourtis et al. [9] proposed recently the CSX format that applies aggressive compression of the column indexing structure of the standard CSR format. CSX searches for dense substructures inside the sparse matrix and instead of storing a single index for every nonzero element of the sparse matrix, it stores a short description for each substructure found. CSX can reduce considerably the memory footprint of the SpMV kernel and achieve significant performance improvements over other standard sparse matrix storage formats.

Minimizing the storage requirements of the column indexing structure of CSR, however, will soon have diminishing returns, since it accounts for only $1/3$ of the total CSR memory footprint. Drawing motivation from this fact, we try to investigate the performance-energy tradeoffs of SpMV. The idea is simple: since SpMV is memory-bound by nature, trying to get a single bit of performance will be increasingly difficult and energy consuming. In this paper, we try to find the *execution configuration* (core frequency and thread placement) that will yield the best performance-energy tradeoff. Voltage and frequency scaling is a key parameter for reducing power consumption in modern commodity processors [2, 7], while thread placement can significantly affect the performance of memory-bound applications on modern multicore architectures with multiple-level hierarchies of shared caches. Since modern processors' power dissipation scales with the third power of the frequency [1, 7], we expect that throttling the cores and using a careful placement of the threads will yield comparable performance at a lower energy budget. Additionally, blindly filling up all the cores of a shared memory machine to solve a memory-bound application, such as SpMV, will not necessarily yield the best performance and will most probably be a waste of energy.

Performance and energy consumption are usually conflicting requirements, especially for SpMV. In this paper, we use the notion of Pareto optimality [12] to analyze the tradeoffs of the different execution configurations. We also show that configurations minimizing energy-delay products are indeed Pareto optimal solutions, i.e., best tradeoffs. Finally, using the substructure metadata provided by CSX and some characteristics of the underlying microarchitecture, we propose a simple machine learning approach based on clustering to predict the execution configurations that are close to the best performance-energy tradeoffs for solving large SpMV problems. Machine learning and statistical modeling have been gaining a growing interest in the high performance computing community recently, since they are able to mine important relations in complex optimization spaces. They have already been successfully used in autotuning high-performance scientific codes [11, 14], in compiler optimizations [10, 13], and in predicting power dissipation [2, 5] of modern microarchitectures.

The rest of the paper is organized as follows: Section 2 provides some basic background information about the power dissipation of modern processors and about the CSX sparse matrix storage format. Section 3 presents our simple power metric used to qualify the different execution configurations, describes the notion of Pareto optimality and discusses the performance-energy tradeoffs of SpMV. Section 4 explains our approach for predicting the best execution configurations and Section 5 presents an experimental evaluation of this approach. Finally, Section 6 concludes the paper and proposes future work.

Workshop on Emerging Supercomputing Technologies (ICS'11) May 31, Tucson, Arizona, USA
Copyright © 2011 ACM . . . $10.00

2. Background

2.1 Basics of processor power consumption

The power dissipation of a CMOS device can be summarized by the following equation [1, 7]:

$$P = \frac{1}{2}CV^2\alpha f + P_{\text{leakage}} + P_{\text{short-circuit}}. \qquad (1)$$

In this equation, C is the circuit capacitance, V is the supply voltage, α ($0 \leq \alpha \leq 1$) is the activity factor of the circuit, i.e., how often the device switches, and f is the operating frequency of the circuit. The last two terms correspond to power dissipation due to leakage and short-circuit currents and can be ignored at the operating voltage range of modern commodity processors. In this voltage range, the operating frequency is proportional to the supply voltage of the circuit, which reduces equation (1) down to

$$P = K\alpha f^3, \qquad (2)$$

where K is a design-specific constant. Summing up this product for every unit in the processor chip yields the total power dissipation of the processor. The important aspect of this equation is that the operating frequency of the processor greatly affects its power dissipation, while the activation of the different processor components (cores, caches, branch prediction units, etc.) play also a significant role.

2.2 A quick overview of CSX

The Compressed Sparse eXtended (CSX) [9] sparse matrix storage format is a recently proposed storage format focusing on the optimization of the SpMV kernel. The main goal of the CSX format is to minimize the space requirements of the column indexing structure of the traditional Compressed Sparse Row (CSR) format. The CSR format utilizes three arrays to store the elements of the original matrix: (a) the values array, which stores the nonzero elements of the matrix, (b) the col_ind array, which stores the column index of each nonzero element, and (c) the row_ptr array, which stores the indices of the values array indicating the start of each row in the original matrix. Since the number of nonzero elements in a sparse matrix greatly exceeds the number of its rows, the col_ind array takes up a substantial part of the total size of the sparse matrix in memory. CSX mines the matrix for dense substructures and instead of storing a 'full' 4-byte integer for each nonzero element, it just uses a 2-byte descriptor for each substructure found. The first byte stores the size (in number of nonzeros) of the encoded substructure, while the second stores its type and some encoding-specific metadata. This technique significantly reduces the storage requirements of the original col_ind array, thus alleviating the pressure to the memory subsystem. The substructures detected by CSX fall into the following categories: (a) *horizontal*, contiguous elements in the horizontal direction, (b) *vertical*, contiguous elements in the vertical direction, (c) *diagonal* and *anti-diagonal*, contiguous elements in the diagonal or anti-diagonal directions, and (d) *two-dimensional blocks*. The remainining elements, i.e., elements not grouped in any substructure, are encoded as *delta units*, which is essentially a delta-encoding scheme for the rest of the col_ind array. Finally, CSX generates at runtime optimized SpMV code for each of the encoded substructure types.

The preprocessing done during CSX construction reveals valuable information about the sparse matrix structure, since different substructures have different computational characteristics that could affect performance [6]. In this paper, we use this information to make an educated guess on the execution configurations that lead to the best energy-performance tradeoffs for SpMV using CSX.

3. Performance-Energy tradeoffs

Modern shared memory architectures have multiple levels of caches that are shared among different cores on the same die. For example, the six-core Intel Dunnington architecture (Figure 1) has three levels of caches: (a) private L1 instruction/data caches per core, (b) an L2 cache shared by every two cores, and (c) a huge 16 MB L3 cache shared by all the six cores. The presence of so large caches

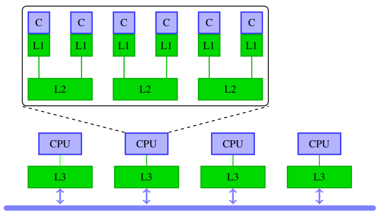


Figure 1. A typical high-end shared memory architecture: a four-way Intel Dunnington platform.

in modern shared memory architectures is essential in order to cope with the limited bandwidth of the common bus and keep threads away from accessing the main memory very often. The sharing of caches among processing cores can also affect significantly the performance of a memory bandwidth-bound application, such as SpMV, since the amount of cache that each thread 'sees' depends on the way the threads are placed on cores. For example, suppose a 64 MB sparse matrix is split equally among 4 threads; if we place the threads on the same package, all threads will experience significant L3 capacity misses, since they contend for cache space. On the other hand, if we place one thread per package, the L3 cache will suffice. Unfortunately, this is not at free: large caches in modern architectures take up almost 50% of the total die area dissipating a significant amount of power [7], and streaming applications, such as SpMV, make heavy use of them. Therefore, distributing threads sparsely among the different physical packages will lead to increased power dissipation. The increased power dissipation, however, can be compensated with a much lower execution time, leading eventually to lower energy consumption.

3.1 A simple power metric for SpMV

In order to qualify the different execution configurations in terms of power dissipation, we use a simple power metric based on equation (2) and similar in principle to the one presented in [5]. Our purpose is to rank the different execution configurations in terms of power dissipation rather than predict the exact power dissipation, so our metric is higher level and less detailed. We view the processor as comprising of three major components: (a) the cores, (b) the caches (L2 and L3), and (c) the bus interface. These components take up significant area on the chip die. For example, the six cores of the Dunnington chip take up approximately 28% of the processor chip, the L2 caches 17%, the L3 cache 31%, and the system bus interface about 24%[1]. Our power metric sums up the activation of each component multiplied by the third power of the processor frequency for each physical package that a thread placement uses. Specifically, our power metric is

$$P = \sum_{\text{pkg}} \sum_{i \in \text{comp.}} \alpha_i \cdot area_i \cdot f^3. \qquad (3)$$

In this equation, α_i is the activity factor of the i-th component in the package, $area_i$ is the percentage of the total area of the chip occupied by the i-th component, and f is the operating frequency of the processor. Since, we are concerned about a single application, namely SpMV, with specific computational characteristics that do not differ a lot from matrix to matrix, we use a more qualitative approach for computing activity factors: for each core we set

[1] The percentages are computed from an annotated die photo of the Dunnington chip published by Intel.

$\alpha_{\text{core}} = 1$, for each cache we set $\alpha_{\text{cache}} = 1$ if the sparse matrix does not fit in this cache, otherwise we set it equal to the percentage of it occupied by the sparse matrix. For the bus interface, we set $\alpha_{\text{bus}} = 1$, if the sparse matrix size is larger than 20% of the aggregate cache size of the packages used, otherwise we set it to zero.

Figure 2 shows the normalized performance versus our normalized energy metric (power metric by execution time) for 112 different execution configurations (core frequency and thread placement) on a four-way six-core Intel Dunnington architecture (Figure 1). The behavior between matrices that fit in the aggregate cache and matrices that do not is quite different. For example, the most energy-efficient configuration for matrix FEM_3D_thermal2 (approximately 30 MB stored with the CSX format) is when using 4 threads, one per physical package, at 2.14 GHz, while the least efficient is when using 6 threads on the same package at 2.67 GHz. Although the former thread placement is more power-hungry, since it turns on more caches, it eventually pays off as it leads to a much lower execution time, because in this configuration the matrix fits in the aggregate cache of the system. On the other hand, 'stuffing' threads in the same package and keeping the frequency high yields not only lower performance (lower than a lot of 2.14 GHz configurations), but also high energy consumption. In this case, the matrix does not fit in the cache and the execution time rises significantly.

For matrices that do not fit in the aggregate cache, as is matrix kkt_power (approximately 140 MB stored with the CSX format), the situation is inversed. The most energy-efficient configuration is 2 threads at 2.14 GHz sharing the L3 cache (but not sharing the L2 cache). Contrary to the smaller matrix, spreading the threads across the packages in this case does not give significant performance benefits, since the problem still does not fit in the aggregate cache and the penalty paid in power dissipation cannot be payed off in reduced execution time. Another interesting observation is that the best performance for this matrix is achieved using only 12 threads spread across the 4 packages, assigning one thread per L2 cache; using more threads will be a waste of energy.

3.2 Characterizing the tradeoffs

Looking at the performance-energy tradeoffs depicted in Figure 2, it is apparent that points in the upper left corner are good tradeoffs, since they maximize performance, while keeping energy requirements low. However, the question that might arise is how we could compare tradeoffs against each other and if there exist a single best tradeoff. Suppose two points, let (e, p) and (e', p'), in the energy-performance landscape of Figure 2 that correspond to execution configurations E and E'. If $e < e'$ and $p \geq p'$ or $e \leq e'$ and $p > p'$, then E is definitely a better tradeoff than E', since with lower energy requirements, it achieves at least the same performance or with at most the same energy requirements achieves strictly greater performance. In this case, we say that configuration E *strictly dominates* configuration E'. However, if $e \leq e'$ and $p \leq p'$, we cannot assess if E is a better tradeoff than E'. The set of points (and the corresponding execution configurations) that are not dominated by any other point in the energy-performance landscape form the *non-dominated Pareto front* [12]. The configurations on the Pareto front form the set of optimal tradeoffs, since they are strictly better than any other configuration in at least one objective. An important property of the Pareto front is that when moving from one point to another, we experience the least possible loss in energy consumption and the greatest gain in performance and vice versa. In that sense, all points of the Pareto front are formally equivalent tradeoffs. This means that there is not a single best tradeoff of performance and energy (unless the Pareto front collapses to a single point), but rather a set of best tradeoffs.

Energy-delay products A common way for finding a good performance-energy tradeoff is trying to minimize the product of energy by execution time, or *energy-delay* product (ED-product). In fact, you can try to minimize ED^n for $n \geq 1$, to bias toward higher performance. The question is whether the configuration minimizing an ED^n product is Pareto optimal, i.e., one of the best tradeoffs, or not. The answer is positive, and it can be easily proved using the notion of Pareto optimality descibed previously.

Assume that the execution configuration M minimizes the ED^n product for some $n \geq 0$, i.e., $e_M d_M^n < e_{M'} d_{M'}^n$ for each $M' \neq M$. If we suppose that (e_M, d_M) does not lie on the Pareto front, then there must be a configuration C, such that either $e_C < e_M$ and $d_C \leq d_M$ or $e_C \leq e_M$ and $d_C < d_M$. Therefore, the energy-delay product of this configuration will be $e_C d_C^n \leq e_M d_M^n$ for $n = 0$ or $e_C d_C^n < e_M d_M^n$ for $n > 0$, which contradicts with our initial assumption that the configuration M minimizes the energy-delay product. Therefore, M is a non-dominated solution, i.e., a point on the Pareto front. As a corollary, every point of the Pareto front is a minimal energy-delay product for some $n \geq 0$.

Figure 3 shows the Pareto front of matrix kkt_power along with the points corresponding to ED and ED^2 products. Each Pareto point is annotated with the corresponding execution configuration.

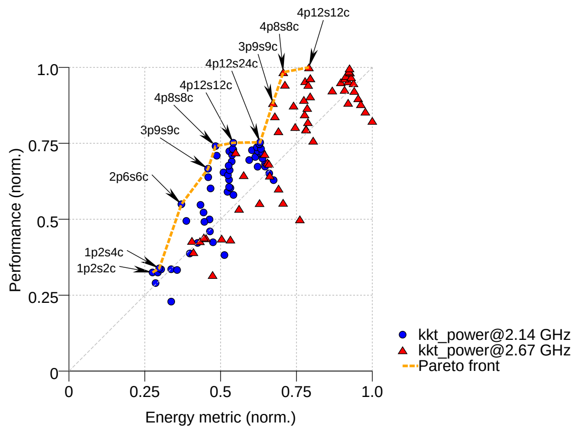
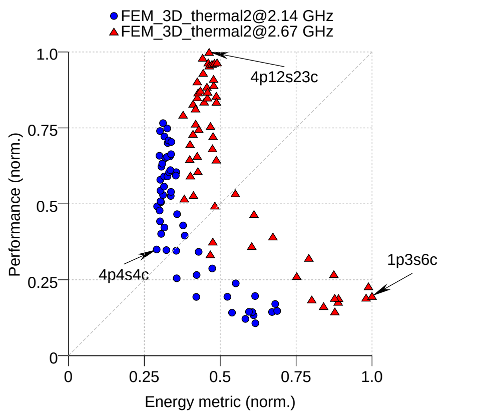


Figure 3. Best performance-energy tradeoffs and the corresponding execution configurations for matrix kkt_power. See also Figure 2 for a description of the configuration annotations.

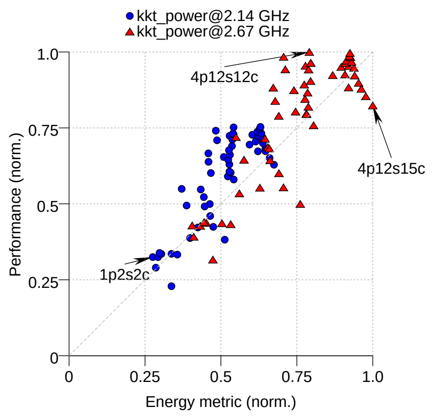
4. Predicting the best configurations

The definitive parameter for SpMV performance is the matrix size. If the matrix is small enough to fit in the aggregate cache of the underlying architecture, SpMV will experience a significant performance improvement, otherwise performance will be severely hindered by contention in the memory subsystem. As discussed in Section 3, even the best energy-performance tradeoffs differ significantly between these two large matrix categories. However, the matrix size is not the only parameter to characterize SpMV performance; previous work [6] has showed that the computational characteristics of the individual substructures inside the sparse matrix play also a role in the final SpMV performance.

The idea behind learning the execution configurations that lead to the best performance-energy tradeoffs is that we expect matrices with similar structural characteristics, e.g., large block-dominated matrices, to have similar performance-energy tradeoffs. Our learning mechanism is based on an initial clustering of matrices based on their size and the substructures encoded by CSX. For each cluster we construct a representative Pareto front from the execution configurations that lie on the Pareto fronts of the matrices in the cluster. Each cluster is then represented by its geometric center (in the space of the matrix attributes) and its Pareto front.



(a) Small matrix.



(b) Large matrix.

Figure 2. The energy-performance tradeoffs of a small (fitting in the aggregate cache) and a large matrix. The thread placements of the minimum/maximum energy consumption and maximum performance are shown; 'p' stands for package, 's' for sub-package (shared L2) and 'c' for core.

4.1 Clustering the matrices

The first step toward grouping together similar matrices based on their execution characteristics is to separate our matrix suite into two major categories: (a) matrices that significantly exceed (more than 20%) the aggregate cache size of the underlying architecture, and (b) matrices that do not. We focus on the first category, since the difference in energy consumption between the most energy-efficient and the most performant configuration is much larger than in the case of a small matrix. For a small matrix, power dissipation is not of such a concern, since adding more threads translates almost directly to a decrease in execution time, therefore keeping energy requirements almost the same (see Figure 2). To further separate our matrices into categories based on the encoded substructures, we assign each matrix a vector of attributes corresponding to the nonzero elements encoded by each major substructure type of CSX and use a hierarchical clustering technique to actually perform the clustering. Specifically, each matrix is assigned the following four attributes:

(1) Percentage of nonzeros covered by horizontal and vertical substructures,
(2) percentage of nonzeros covered by diagonal and anti-diagonal substructures,
(3) percentage of nonzeros covered by block substructures, and
(4) percentage of nonzeros covered by delta units.

As a distance metric between the matrices we use the euclidean distance. Hierarchical clustering starts by considering each observation to be in its own cluster and proceeds by merging together individual clusters until a certain distance threshold is reached. Figure 4 shows the clusters formed from our matrix suite with the distance threshold set to 0.5.

4.2 Constructing the cluster Pareto front

Assuming a good clustering of the matrices, the cluster Pareto front would consist of the most common execution configurations present in the Pareto fronts of every matrix in the cluster. To construct the cluster Pareto front, we start iterating the Pareto fronts of the matrices in the cluster from the most performant configuration to the most energy-efficient, i.e., from the last Pareto point down to the first. At each iteration, we record the execution configurations that correspond to the current point for every matrix in the cluster and select the most frequent. If two or more configurations have the same frequency, we select all of them if their frequency is more than 25% of the cluster size, otherwise we select the one with the

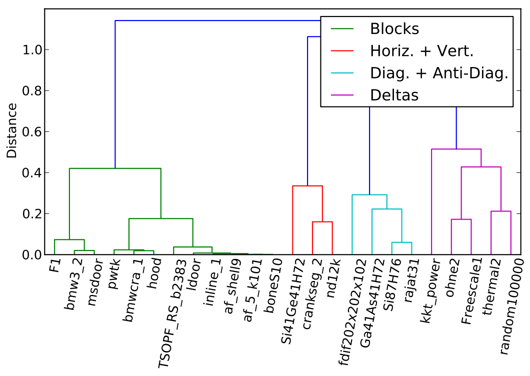


Figure 4. Clustering of the largest matrices of our matrix suite based on the substructures encoded by CSX. The different clusters are characterized by their dominating substructure.

least number of threads, in order to cover better the low-energy configurations. We also use this technique to provide predictions about the configurations minimizing the ED and ED^2 products.

Avoiding overfitting The Pareto fronts of the different matrices in a cluster do not necessarily have the same amount of points, so with our method for constructing the cluster Pareto, there is a risk of creating a Pareto front with so many points that will not be representative of the cluster, since it will cover every 'corner case' in the matrix cluster. To avoid this situation of overfitting, we stop the construction of the Pareto front as soon as we 'run out' of points for more than half of the matrices in the cluster. Figure 5 shows an example of the construction of the cluster Pareto front.

4.3 Classification and testing

After having trained our model, we store the cluster centers and the associated cluster Pareto fronts for future use. When a new matrix appears, we run CSX to retrieve the metadata about the encoded substructures and classify it to the closest cluster based on its euclidean distance from each of the cluster centers. The best performance-energy configurations for this matrix are then assumed to be the configurations of the Pareto front of the selected cluster.

In order to assess the quality of our prediction, we need a metric to characterize the predicted execution configurations as

Matrix 0	*Matrix 1*	*Matrix 2*	*Cluster Pareto*
4p12s24c@2.67	**4p12s24c@2.67**	**4p12s24c@2.67**	*4p12s24c@2.67*
4p12s12c@2.67	**4p12s12c@2.67**	**4p12s12c@2.67**	*4p12s12c@2.67*
4p8s8c@2.67	**4p8s8c@2.67**	**4p8s8c@2.67**	*4p8s8c@2.67*
4p12s24c@2.14	**4p12s24c@2.14**	**4p12s24c@2.14**	*4p12s24c@2.14*
4p12s12c@2.14	**4p12s12c@2.14**	**4p12s12c@2.14**	*4p12s12c@2.14*
4p8s8c@2.14	**4p8s8c@2.14**	**4p8s8c@2.14**	*4p8s8c@2.14*
2p6s6c@2.14	1p2s2c@2.14	**2p6s6c@2.14**	*2p6s6c@2.14*
1p3s3c@2.14	n/a	1p3s6c@2.14	*1p3s3c@2.14*
1p2s2c@2.14	n/a	1p2s4c@2.14	*1p2s2c@2.14*
n/a	n/a	1p2s2c@2.14	n/a

Figure 5. Construction of the cluster Pareto front assuming a cluster of three matrices. Bold typeface shows the configurations that are selected to be part of the constructed Pareto front.

Matrix	*Size*	*Problem*	*Matrix*	*Size*	*Problem*
thermal2	86	thermal	TSOPF_RS_b2383	123	power
bmwcra_1	83	structural	af_5_k101	135	structural
Si87H76	85	chemistry	af_shell9	135	structural
hood	84	structural	Ga41As41H72	151	chemistry
ohne2	100	semicond.	Freescale1	201	circuit sim.
bmw3_2	89	structural	msdoor	163	structural
pwtk	91	structural	rajat31	173	circuit sim.
crankseg_2	114	structural	F1	214	structural
nd12k	115	2d/3d	fdif202x202x102	213	other
random100000	143	n/a	inline_1	291	structural
Si41Ge41H72	121	chemistry	ldoor	358	structural
kkt_power	140	optim.	boneS10	431	model red.

Table 1. Matrix suite. The 'Size' column corresponds to the size of the matrix in MB when encoded with the CSX format.

performance-energy tradeoffs. According to our discussion in Section 3, the set of configurations lying on the Pareto front are all equivalent optimal tradeoffs. If we take out these configurations and recompute the Pareto front with the remaining configurations, then we have the set of the second best configurations, if we also omit these and recompute, we have the third best and so forth. We can therefore assign each predicted configuration a distance value $d \geq 0$ denoting how far from the optimal set it lies. For example, $d = 0$ means that our prediction is optimal, while $d = 2$ means that our prediction is one of the third best configurations. If we take an average of the distance of every configuration on the predicted Pareto front from the optimal one, we have a metric of how far from the optimal the predicted Pareto front is.

5. Experimental evaluation

For the evaluation of our prediction method, we used a four-way six-core Intel Xeon X7460 (Dunnington) as a target architecture (see also Figure 1). Each processor runs at 2.67 GHz, has private 32 KB instruction and data caches, a 3 MB L2 data cache shared by every two cores, and a 16 MB L3 cache shared by all cores in the package. Each core supports two different frequency steps: one at 2.14 GHz and one at the full speed of 2.67 GHz. For scaling the core frequency we use the cpufrequtils userspace tools for Linux. All experiments were run on a GNU/Linux 2.6.30.5 system and for the matrix encoding we used the publicly available CSX sparse matrix storage format compiled with GCC 4.3.2 and LLVM 2.7.

The matrix suite used for our experiments is the same as the one used in [8] and consists of matrices from the University of Florida Sparse Matrix Collection [3]. We will list here only the 24 matrices that do not fit in the 64 MB aggregate cache of our target platform (see Table 1) and are of our main concern in this paper according to the discussion in Section 4.1.

Execution configurations The thread placements that we consider fall into three categories: (a) *full sharing*, in which threads are placed as close as possible, sharing all the levels of the cache hierarchy, (b) *semi sharing*, in which threads are placed so that they share the L3 cache, but not the L2, and (c) *no sharing*, in which threads are placed as sparsely as possible, so as to minimize the sharing of the cache hierarchy. In the following, to describe the different thread placements we will use the notation 'PpSsCc', denoting that we use C cores, S sub-packages (i.e., shared L2 cache) and P physical packages. For example, the possible thread placements for 4 threads would be 1p2s4c, 2p4s4c, and 4p4s4c. We consider 56 different thread placements ranging from 1 to 24 threads for both frequency steps, giving us a total of 112 execution configurations. For each configuration we run 128 consecutive SpMV operations with the CSX format for every matrix in our suite and store the results to train and test our model.

Fold No.	*Avg. Dist.*	*95% C.I.*	*Dist. ≤ 2*	*Dist. ≤ 3*
Fold 1	0.822	0.783	99.8%	100.0%
Fold 2	1.410	1.235	82.5%	99.4%
Fold 3	0.225	0.168	100.0%	100.0%
Fold 4	1.253	1.412	85.0%	99.2%
Fold 5	0.294	0.350	100.0%	100.0%

Table 2. Cross-validation results. The average Pareto distance and a 95% confidence interval are depicted for each fold. The probability for the average Pareto distance to be lower than a threshold is also depicted.

Training and testing The training of the model is performed offline, since we must perform a large number of SpMV operations to record the energy-performance tradeoffs of the matrices in the training set, a procedure which can last several hours. In order to build our model, we used SciPy (Scientific Tools for Python) 0.7.0, which is a collection of Python modules for mathematics, science, and engineering. The method we use to test our model is a 5-cross-validation (5-CV) technique, which consists of the following steps: (a) shuffle uniformly the initial matrix suite and split it into five equal parts (a.k.a. folds), (b) keep one fold for testing and use the rest for training, and (c) repeat until all folds are tested. Cross-validation is a common statistics technique for assessing the prediction accuracy of a predictive model.

Table 2 summarizes the results of the 5-CV test for our model. For each fold, it depicts the average distance (as described in Section 4.3) of the predicted Pareto front from the original one along with a 95% confidence interval. It also depicts the probability that the average distance of the predicted Pareto is below a certain threshold. Specifically, the distance of the predicted Pareto will be less than or equal to 2 for more than 80% of the time, while it is almost certain that its distance will be less than 3. In practice, this means that the predicted execution configurations are very likely to lie in the first three sets of optimal configurations. Figure 6 shows graphically the predicted execution configurations for matrix Freescale1 along with the ED and ED^2 predictions. Table 3 shows the execution configurations for minimizing the ED and ED^2 products for 10 representative matrices. In most cases we either predict correctly or our prediction lies on the real Pareto front, which means that it still is an optimal tradeoff. Specifically, for more than 80% of the matrices in our suite, our model predicted either the correct configuration or a configuration lying on the Pareto front for energy (83%), ED (88%), and ED^2 (79%) products.

Limitations Although our method can predict fairly accurately the execution configurations leading to the best energy-performance tradeoffs, it cannot currently predict how far these configurations are in the energy-performance landscape. For example, although the configurations 4p8s8c, 4p12s12c, and 4p12s24c in Figure 3 are equivalent in terms of Pareto optimality, the last two are not so interesting, since they lead to a 20% increase in energy consumption without any significant performance benefit. This could be improved by a more careful analysis of the points on the Pareto

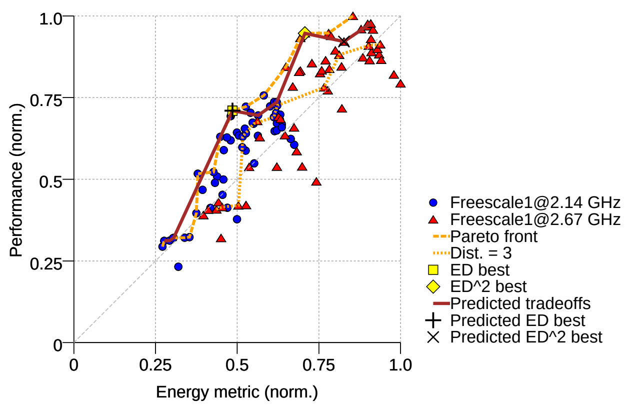


Figure 6. The predicted Pareto front for matrix Freescale1. The predicted best ED and ED^2 configurations are also shown.

	Energy		ED		ED^2	
Matrix	*Real*	*Pred.*	*Real*	*Pred.*	*Real*	*Pred.*
thermal2	1p2s3c@2.14	No(1)	1p2s3c@2.14	No(3)	4p12s23c@2.67	No(4)
bmwcra_1	1p2s2c@2.14	**Yes**	4p12s12c@2.14	No(0)	4p12s12c@2.67	**Yes**
Si87H76	1p2s2c@2.14	No(0)	4p12s12c@2.67	No(4)	4p12s12c@2.67	No(1)
pwtk	1p2s2c@2.14	**Yes**	4p8s8c@2.14	**Yes**	4p12s12c@2.67	**Yes**
nd12k	1p2s2c@2.14	No(2)	4p8s8c@2.14	No(0)	4p12s12c@2.67	**Yes**
af_shell9	1p2s2c@2.14	**Yes**	4p8s8c@2.14	**Yes**	4p8s8c@2.67	No(0)
rajat31	1p2s2c@2.14	**Yes**	4p8s8c@2.14	**Yes**	4p12s12c@2.67	No(0)
Freescale1	1p1s2c@2.14	No(0)	4p8s8c@2.14	**Yes**	4p8s8c@2.67	No(2)
F1	1p2s2c@2.14	**Yes**	4p8s8c@2.14	**Yes**	4p12s12c@2.67	**Yes**
ldoor	1p2s2c@2.14	**Yes**	4p8s8c@2.14	**Yes**	4p12s12c@2.67	**Yes**

Table 3. Configurations for minimizing the ED products for 10 selected matrices. It is also noted whether they were predicted by our model or not. In case of incorrect predictions, the distance of the predicted configuration from the real Pareto front is showed inside parenthesis.

front, possibly ignoring such configurations. Additionaly, the prediction accuracy for two CV folds (see Table 2) is not so high; this is mainly due to the fact that our initial dataset is small enough to leave some clusters with very few observations, thus deteriorating the quality of training.

6. Conclusions and Future Work

In this paper, we took a first step toward exploring the performance-energy tradeoffs of Sparse Matrix-Vector Multiplication showing that the matrix and the underlying platform play a significant role in determining the best tradeoffs. For the purpose of our analysis, we used a simple power metric based on the activation of the different physical packages of the underlying platform. We tried to characterize the performance-energy tradeoffs using the concept of Pareto optimality from the multi-objective optimization theory and showed that energy-delay products are indeed best tradeoffs in the Pareto sense. Finally, we proposed a machine learning-based technique for creating a model for predicting the optimal performance-energy tradeoffs depending on the input matrix and the underlying architecture.

In the future, we plan to ameliorate our prediction model, since it still has some limitations. We currently work on identifying the key performance parameters of CSX, in order to obtain a better clustering of the matrices, more biased on their performance characteristics. Finally, we plan to test our prediction methodology on a variety of multiprocessor architectures.

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*, 20(6):26–44, 2000.

[2] 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 (PACT'08)*, pages 250–259, Toronto, Ontario, Canada, 2008. ACM.

[3] T. Davis. The University of Florida sparse matrix collection. NA Digest, vol. 97, no. 23, June 1997. `http://www.cise.ufl.edu/research/sparse/matrices`.

[4] G. Goumas, K. Kourtis, N. Anastopoulos, V. Karakasis, and N. Koziris. Understanding the performance of sparse matrix-vector multiplication. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, Toulouse, France, 2008. IEEE Computer Society.

[5] 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, California, USA, 2003. IEEE Computer Society.

[6] V. Karakasis, G. Goumas, and N. Koziris. Exploring the effect of block shapes on the performance of sparse kernels. In *2009 IEEE International Symposium on Parallel & Distributed Processing*, pages 1–8, Rome, Italy, 2009. IEEE Computer Society.

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

[8] K. Kourtis, G. Goumas, and N. Koziris. Exploiting compression opportunities to improve SpMxV performance on shared memory systems. *ACM Transactions on Architecture and Code Optimization (TACO)*, 7(3), 2010.

[9] K. Kourtis, V. Karakasis, G. Goumas, and N. Koziris. CSX: An Extended Compression Format for SpMV on Shared Memory Systems. In *PPoPP'11*, pages 247–256, San Antonio, Texas, USA, 2011. ACM.

[10] H. Leather, E. Bonilla, and M. O'Boyle. Automatic Feature Generation for Machine Learning Based Optimizing Compilation. In *Proceedings of the 7th annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO'09)*, pages 81–91, Seattle, WA, USA, 2009. IEEE Computer Society.

[11] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh, and S. A. McKee. Methods of inference and learning for performance modeling of parallel applications. In *PPoPP'07*, pages 249–258, San Jose, CA, USA, 2007. ACM.

[12] S. Luke. *Essentials of Metaheuristics*. Lulu, first edition, 2011.

[13] G. Tournavitis, Z. Wang, B. Franke, and M. O'Boyle. Towards a holistic approach to auto-parallelization: integrating profile-driven parallelism detection and machine-learning based mapping. In *PLDI'09*, pages 177–187, Dublin, Ireland, 2009. ACM.

[14] R. Vuduc, J. W. Demmel, and J. Bilmes. Statistical Models for Automatic Performance Tuning. In *International Conference on Computational Science*, pages 117–126, San Fransisco, CA, USA, 2001. Springer.

[15] J. Willcock and A. Lumsdaine. Accelerating sparse matrix computations via data compression. In *Proceedings of the 20th annual International conference on Supercomputing*, pages 307–316, Cairns, Queensland, Australia, 2006. ACM.

[16] S. Williams, L. Oilker, R. Vuduc, J. Shalf, K. Yelick, and J. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, Reno, NV, USA, 2007. ACM.