EPEM: A General and Validated Energy Complexity Model for Multithreaded Algorithms (IFI-UiT Technical Report 2016-77)

Vi Ngoc-Nha Tran and Phuong Hoai Ha

The Arctic Green Computing Group
Department of Computer Science
UiT The Arctic University of Norway
Tromso, Norway
{vi.tran, phuong.hoai.ha}@uit.no

May 26, 2016

Abstract

Like time complexity models that have significantly contributed to the analysis and development of fast algorithms, energy complexity models for parallel algorithms are desired as crucial means to develop energy efficient algorithms for ubiquitous multicore platforms. Ideal energy complexity models should be validated on real multicore platforms and applicable to a wide range of parallel algorithms. However, existing energy complexity models for parallel algorithms are either theoretical without model validation or algorithm-specific without ability to analyze energy complexity for a wide range of parallel algorithms.

This paper presents a new general validated energy complexity model for parallel (multithreaded) algorithms. The new model abstracts away possible multicore platforms by their static and dynamic energy of computational operations and data access, and derives the energy complexity of a given algorithm from its work, span and I/O complexity. The new energy complexity model is validated by different sparse matrix vector multiplication (SpMV) algorithms and dense matrix multiplication (matmul) algorithms running on high performance computing (HPC) platforms (e.g., Intel Xeon and Xeon Phi). The new model is able to characterize and compare the energy consumption of SpMV and matmul kernels according to three aspects: different algorithms, different input matrix types and different platforms. The prediction of the new model regarding which algorithm consumes more energy with different inputs on different platforms, is confirmed by the experimental results. In order to improve the usability and accuracy of the new model for a wide range of platforms,
the platform parameters of EPEM model are provided for eleven platforms including HPC, accelerator and embedded platforms.

1 Introduction

Understanding the energy complexity of algorithms is crucially important to improve the energy efficiency of algorithms and reduce the energy consumption of computing systems. One of the main approaches to understanding the energy complexity of algorithms is to devise energy models.

Significant efforts have been devoted to developing power and energy models in literature [2, 13, 12, 19, 20, 18, 24, 25]. However, there are no analytic models for multithreaded algorithms that are both applicable to a wide range of algorithms and comprehensively validated yet (cf. Table 1). The existing parallel energy models are either theoretical studies without validation or only applicable for specific algorithms. Modeling energy consumption of parallel algorithms is difficult since the energy models must take into account the complexity of both parallel algorithms and parallel platforms. The algorithm complexity results from parallel computation, concurrent memory accesses and inter-process communication. The platform complexity results from multicore architectures with deep memory hierarchy.

The existing models and their classification are summarized in Table 1. The previous studies have not covered all considered aspects: ability to analyze the energy complexity of parallel algorithms (i.e. Energy complexity analysis for parallel algorithms), whether the model is applicable to a wide range of algorithms (i.e., Algorithm generality), whether the model is validated (i.e., Validation). Table 1 also shows how our new EPEM (Energy-aware Parallel External Memory) model is different from the other studies. The energy complexity model EPEM proposed in this study is for general multithreaded algorithms and validated on three aspects: different algorithms for a given problem, different input types and different platforms. The proposed model is an analytic model which characterizes both algorithms (e.g., representing algorithm by their work, span and I/O complexity) and platform (e.g., representing platforms by their static and dynamic energy of memory accesses and computational operations). By considering work and span complexity, the new EPEM model is applicable to any multithreaded algorithms.

Since the new EPEM energy model focuses on analyzing the energy complexity of algorithms, the model does not give the estimation of absolute energy consumption. The new model, instead, provides the algorithm designers with the understanding of how an algorithm consumes energy and give the insight into how to choose one algorithm over the others for different input types and platforms. The EPEM model has been validated for two SpMV algorithms and two matmul algorithms running on two high performance platforms (Intel Xeon and Xeon Phi). The validation results confirm the practicability and applicability of the EPEM energy complexity model. In this work, the following contributions have been made.
Table 1: Energy Model Summary

| Study                  | Energy complexity analysis for parallel algorithms | Algorithm generality       | Validation |
|------------------------|---------------------------------------------------|-----------------------------|------------|
| LEO [24]               | No                                                | General                     | Yes        |
| POET [18]              | No                                                | General                     | Yes        |
| Koala [20]             | No                                                | General                     | Yes        |
| Roofline [10, 9]       | No                                                | General                     | Yes        |
| Energy scalability [10, 20] | Yes                              | General                     | No         |
| Sequential energy complexity [25] | No                             | General                     | Yes        |
| Alonso et al. [2]      | Yes                                               | Algorithm-specific          | Yes        |
| Malossi et al. [23]    | Yes                                               | Algorithm-specific          | Yes        |
| EPEM model (this study)| Yes                                               | General                     | Yes        |

- Devising a new general energy model EPEM for analyzing the energy complexity of a wide range of multithreaded algorithms based on their work, span and I/O complexity (cf. Section 3). The new EPEM model abstracts away possible multicore platforms by their static and dynamic energy of computational operations and memory access. The new EPEM model complements previous energy models such as energy roofline models [10, 9] that abstract away possible algorithms to analyze the energy consumption of different multicore platforms.

- Conducting two case studies (i.e., SpMV and matmul) to demonstrate the methodology of how to apply the EPEM model to find energy complexity of parallel algorithms. The selected parallel algorithms for SpMV are three algorithms: Compressed Sparse Column(CSC), Compressed Sparse Block(CSB) and Compressed Sparse Row(CSR)(cf. Section 4). The selected parallel algorithms for matmul are two algorithms: a simple matmul algorithm and a cache-oblivious algorithm (cf. Section 6).

- Validating the EPEM energy complexity model with both data-intensive (i.e., SpMV) and computation-intensive (i.e., matmul) algorithms according to three aspects: different algorithms, different input types and different platforms. The results show the precise prediction on which validated SpMV algorithm (i.e., CSB or CSC) consumes more energy when computing different matrix input types from Florida matrix collection [12] (cf. Section 5). The results also show the precise prediction on which validated matmul algorithm (i.e., simple or cache-oblivious) consumes more energy (cf. Section 6). The model platform-related parameters for 11 platforms, including x86, ARM and GPU, are provided to facilitate the deployment of the EPEM model.
2 EPEM Shared Memory Machine Model

Generally speaking, the energy consumption of a parallel algorithm is the sum of i) static energy (or leakage) \( E_{\text{static}} \), ii) dynamic energy of computation \( E_{\text{comp}} \) and iii) dynamic energy of memory accesses \( E_{\text{mem}} \). The static energy \( E_{\text{static}} \) is proportional to the execution time of the algorithm while the dynamic energy of computation and the dynamic energy of memory accesses are proportional to the number of computational operations and the number of memory accesses of the algorithm, respectively [20]. As a result, in the new EPEM energy complexity model the energy complexity of a multithreaded algorithm is analyzed based on its span complexity [11] (for the static energy), work complexity [11] (for the dynamic energy of computation) and I/O complexity (for the dynamic energy of memory accesses) (cf. Section 3).

This section describes shared-memory machine models supporting I/O complexity analysis for parallel algorithms. We first describe the parallel external memory (PEM) model [3] used for analyzing the energy scalability of parallel algorithms on shared memory multicore platforms [19] and explain why the PEM model is not appropriate for analyzing the energy complexity of multithreaded algorithms. We then describe the ideal distributed cache (IDC) model [15] that is used in the EPEM energy complexity model.

2.1 The PEM Model

The PEM model [3] is an extension of the Parallel Random Access Machine (PRAM) model that includes a two-level memory hierarchy. In the PEM model, there are \( n \) cores (or processors) each of which has its own private cache of size \( Z \) (in bytes) and shares the main memory with the other cores (cf. Figure 1). Unlike other I/O models for multicore platforms [6, 7], the PEM model enables analyzing the I/O complexity of parallel algorithms without additional assumption on how the cores are connected nor how the algorithm tasks are scheduled. In the PEM model, data is transferred between the shared memory and the cache in the form of blocks of size \( B \) (i.e., cache lines). The number of parallel block transfers between the shared memory and the caches is defined as I/O complexity. Namely, when \( n \) cores access \( n \) distinct blocks from the shared memory simultaneously, the I/O complexity in the PEM model is \( O(1) \) instead of \( O(n) \). Like the PRAM shared-memory parallel model, the PEM model has three variations according to how multiple cores access the same block of shared memory, namely: Concurrent Read, Concurrent Writes (CRCW); Concurrent Read, Exclusive Write (CREW) and Exclusive Read, Exclusive Write (EREW). In the cases of exclusive write (i.e., CREW and EREW), there are write conflicts between \( n \) conflicting writes to the same block in the main memory. A solution to the conflict is to serialize the \( n \) writes, resulting in \( n \) I/Os. The I/O complexity of \( n \) conflicting writes can be improved to \( O(\log n) \) by using extra memory to combine the writes in a binary tree fashion [3].
2.2 The IDC Model

Although the PEM model is appropriate for analyzing the I/O complexity of parallel algorithms in terms of time performance [4], we have found that the PEM model is not appropriate for analyzing parallel algorithms in terms of the dynamic energy of memory accesses. In fact, even when the $n$ cores can access data from the main memory simultaneously, the dynamic energy consumption of the access is proportional to the number $n$ of accessing cores (because of the load-store unit activated within each accessing core and the energy compositionality of parallel computations [17, 22]), rather than a constant as implied by the PEM model.

As a result, we choose the ideal distributed cache (IDC) model [15] to analyze I/O complexity of multithreaded algorithms in terms of dynamic energy consumption. Like the PEM model, the IDC model has $n$ cores and a two-level memory hierarchy as shown in Figure 1. Each core has its own private cache of size $Z$, which cannot be accessed by the other cores, and shares the main memory with the other cores. All the inter-core communication is conducted through writing to and reading from the main memory. The core must have data in its cache in order to operate on the data and the data is transferred between the main memory and its cache in blocks of size $B$ (i.e., cache line size).

Unlike the PEM model, the IDC model defines I/O complexity (or cache complexity) of a computation as the number of cache misses caused by the computation on an ideal cache starting and ending with an empty cache. An ideal cache is a fully associative cache that uses optimal offline cache replacement policy. If a core does not have the data word it wants to access in its private cache, it incurs a cache miss to bring the data from the main memory to its private cache. The private caches are non-interfering, namely the number of cache misses incurred by a core can be analyzed independently of the other cores’ actions. Since the cache complexity of $m$ misses is $O(m)$ regardless of whether or not the cache misses are incurred simultaneously by the cores, the IDC model reflects the aforementioned dynamic energy consumption of memory access.
accesses by the cores.

However, the IDC model is mainly designed for analyzing the cache complexity of divide-and-conquer algorithms, making it difficult to apply to general multi-threaded algorithms targeted by our new EPEM energy model. Constraining the new EPEM energy model to the IDC model would limit the applicability of the EPEM model to a wide range of multithreaded algorithms.

In order to make our new EPEM energy model applicable to a wide range of multithreaded algorithms, we show that the cache complexity analysis using the traditional (sequential) ideal cache (IC) model [14] can be used to find an upper bound on the cache complexity of the same algorithm using the IDC model (cf. Lemma 2.1). As the sequential execution of multithreaded algorithms is a valid execution regardless of whether they are divide-or-conquer algorithms, the ability to analyze the cache complexity of multithreaded algorithms via their sequential execution in the EPEM energy model improves the usability of the EPEM model.

Let $Q_{1}(Alg, B, Z)$ and $Q_{P}(Alg, B, Z)$ be the cache complexity of a parallel algorithm $Alg$ analyzed in the (uniprocessor) ideal cache (IC) model [14] with block size $B$ and cache size $Z$ (i.e., running $Alg$ with a single core) and the cache complexity analyzed in the (multicore) IDC model with $P$ cores each of which has a private cache of size $Z$ and block size $B$, respectively. We have the following lemma:

**Lemma 2.1.** The cache complexity $Q_{P}(Alg, B, Z)$ of a parallel algorithm $Alg$ analyzed in the ideal distributed cache (IDC) model with $P$ cores is bounded from above by the product of $P$ and the cache complexity $Q_{1}(Alg, B, Z)$ of the same algorithm analyzed in the ideal cache (IC) model. Namely,

$$Q_{P}(Alg, B, Z) \leq P \times Q_{1}(Alg, B, Z)$$  \hspace{1cm} (1)

**Proof.** (Sketch) Let $Q_{iP}^{'}(Alg, B, Z)$ be the number of cache misses incurred by core $i$ during the parallel execution of algorithm $Alg$ in the IDC model. Because caches do not interfere with each other in the IDC model, the number of cache misses incurred by core $i$ when executing algorithm $Alg$ in parallel by $P$ cores is not greater than the number of cache misses incurred by core $i$ when executing the whole algorithm $Alg$ only by core $i$. That is,

$$Q_{iP}^{'}(Alg, B, Z) \leq Q_{i}(Alg, B, Z)$$  \hspace{1cm} (2)

or

$$\sum_{i=1}^{P} Q_{iP}^{'}(Alg, B, Z) \leq P \times Q_{1}(Alg, B, Z)$$  \hspace{1cm} (3)

On the other hand, since the number of cache misses incurred by algorithm $Alg$ when it is executed by $P$ cores in the IDC model is the sum of the numbers of cache misses incurred by each core during the $Alg$ execution, we have

$$Q_{P}(Alg, B, Z) = \sum_{i=1}^{P} Q_{iP}^{'}(Alg, B, Z)$$  \hspace{1cm} (4)
From Equations 3 and 4 we have

\[ Q_P(Alg, B, Z) \leq P \star Q_1(Alg, B, Z) \]  

(5)

We also make the following assumptions regarding platforms.

- Algorithms are executed with the best configuration (e.g., maximum number of cores, maximum frequency) following the race-to-halt strategy.
- The I/O parallelism is bounded from above by the computation parallelism. Namely, each core can issue a memory request only if its previous memory requests have been served. Therefore, the work and span (i.e., critical path) of an algorithm represent the parallelism for both I/O and computation.

3 Energy Complexity in EPEM model

This section describes two energy complexity models, a platform-supporting energy complexity model considering both platform and algorithm characteristics and platform-independent energy complexity model considering only algorithm characteristics. The platform-supporting model is used when platform parameters in the model are available while platform-independent model analyses energy complexity of algorithms without considering platform characteristics.

3.1 Platform-supporting Energy Complexity Model

This section describes a methodology to find energy complexity of algorithms. The energy complexity model considers three groups of parameters: machine-dependent, algorithm-dependent and input-dependent parameters. The reason to consider all three parameter-categories is that only operational intensity [27] is insufficient to capture the characteristics of algorithms. Two algorithms with the same values of operational intensity might consume different levels of energy. The reasons are their differences in data accessing patterns leading to performance scalability gap among them. For example, although the sequential version and parallel version of an algorithm may have the same operational intensity, they may have different energy consumption since the parallel version would have less static energy consumption because of shorter execution time.

The energy consumption of a parallel algorithm is the sum of i) static energy (or leakage) \( E_{\text{static}} \), ii) dynamic energy of computation \( E_{\text{comp}} \) and iii) dynamic energy of memory accesses \( E_{\text{mem}} \): \[ E = E_{\text{static}} + E_{\text{comp}} + E_{\text{mem}} \]  

[10][19][20]. The static energy \( E_{\text{static}} \) is the product of the execution time of the algorithm and the static power of the whole platform. The dynamic energy of computation and the dynamic energy of memory accesses are proportional to the number of computational operations \( \text{Work} \) and the number of memory accesses \( \text{I/O} \), respectively. Since computation time and memory-access time can be overlapped,
Table 2: EPEM Model Parameter Description

| Machine | Description                                      |
|---------|--------------------------------------------------|
| $\epsilon_{op}$ | dynamic energy of one operation (average)         |
| $\epsilon_{I/O}$ | dynamic energy of a random memory access (1 core) |
| $\pi_{op}$      | static energy when performing one operation     |
| $\pi_{I/O}$     | static energy of a random memory access          |

| Algorithm | Description                                      |
|-----------|--------------------------------------------------|
| Work      | Number of work in flops of the algorithm [11]    |
| Span      | The critical path of the algorithm [11]          |
| I/O       | Number of cache line transfer of the algorithm [11] |

the execution time of the algorithm is assumed to be the maximum of computation time and memory-access time [10]. Therefore, the energy consumption of algorithms is computed by Equation 6, where the values of EPEM parameters, including $\epsilon_{op}$, $\epsilon_{I/O}$, $\pi_{op}$, and $\pi_{I/O}$ are described in Table 2 and computed by the Equation 7, 8, 9, and 10, respectively.

$$E = \epsilon_{op} \times \text{Work} + \epsilon_{I/O} \times I/O + P_{sta} \times \max(T_{comp}, T_{mem})$$  (6)

$$\epsilon_{op} = P_{op} \times \frac{F}{Freq}$$  (7)

$$\epsilon_{I/O} = P_{I/O} \times \frac{M}{Freq}$$  (8)

$$\pi_{op} = P_{sta} \times \frac{F}{Freq}$$  (9)

$$\pi_{I/O} = P_{sta} \times \frac{M}{Freq}$$  (10)

The dynamic energy of one operation by one core $\epsilon_{op}$ is the product of the consumed power of one operation by one active core $P_{op}$ and the time to perform one operation. Equation 7 shows how $\epsilon_{op}$ relates to frequency $Freq$ and the number of cycles per operation $F$. Similarly, the dynamic energy of a random access by one core $\epsilon_{I/O}$ is the product of the consumed power by one active core performing one I/O (i.e., cache-line transfer) $P_{I/O}$ and the time to perform one cache line transfer computed as $M/Freq$, where $M$ is the number of cycles per cache line transfer (cf. Equation 8). The static energy of operations $\pi_{op}$ is the product of the whole platform static power $P_{sta}$ and time per operation. The static energy of one I/O $\pi_{I/O}$ is the product of the whole platform static power and time per I/O, shown by Equation 9 and 10.
Table 3: Platform parameter summary. The parameters of the first nine platforms are derived from [9] and the parameters of the two new platforms are found in this study.

| Platform          | Processor          | $\epsilon_{op}$ (nJ) | $\pi_{op}$ (nJ) | $\epsilon_{I/O}$ (nJ) | $\pi_{I/O}$ (nJ) |
|-------------------|--------------------|----------------------|-----------------|----------------------|-----------------|
| Nehalem i7-950    | Intel i7-950       | 0.670                | 2.455           | 50.88                | 408.80          |
| Ivy Bridge i3-3217U | Intel i3-3217U     | 0.024                | 0.591           | 26.75                | 58.99           |
| Bobcat CPU        | AMD E2-1800        | 0.199                | 3.980           | 27.84                | 387.47          |
| Fermi GTX 580     | NVIDIA GF100       | 0.213                | 0.622           | 32.83                | 45.66           |
| Kepler GTX 680    | NVIDIA GK104       | 0.263                | 0.452           | 27.97                | 26.90           |
| Kepler GTX Titan  | NVIDIA GK110       | 0.094                | 0.077           | 17.09                | 32.94           |
| XeonPhi KNC       | Intel 5110P        | 0.012                | 0.178           | 8.70                 | 63.65           |
| Cortex-A9         | TI OMAP 4460       | 0.302                | 1.152           | 51.84                | 174.00          |
| Arndale Cortex-A15| Samsung Exynos 5   | 0.275                | 1.385           | 24.70                | 89.34           |

In order to compute work, span and I/O complexity of the algorithms, the input parameters also need to be considered. For example, SpMV algorithms consider input parameters listed in Table 4.

The details of how to obtain the EPEM parameters of recent platforms are discussed in Section 5.3. The actual values of EPEM platform parameters for 11 recent platforms are presented in Table 3.

The computation time of parallel algorithms is proportional to the span complexity of the algorithm, which is $T_{comp} = \frac{\text{Span} \times F}{\text{Freq}}$ where $\text{Freq}$ is the processor frequency, and $F$ is the number of cycles per operation. The memory-access time of parallel algorithms in the EPEM model is proportional to the I/O complexity of the algorithm divided by its I/O parallelism, which is $T_{mem} = \frac{\text{I/O}}{\text{I/O-parallelism}} \times \frac{M}{\text{Freq}}$. As I/O parallelism, which is the average number of I/O ports that the algorithm can utilize per step along the span, is bounded by the computation parallelism $\frac{\text{Work}}{\text{Span}}$, namely the average number of cores that the algorithm can utilize per step along the span (cf. Section 2), the memory-access time $T_{mem}$ becomes: $T_{mem} = \frac{\text{I/O} \times \text{Span} \times M}{\text{Work} \times \text{Freq}}$ where $M$ is the number of cycles per cache line transfer. If an algorithm has $T_{comp}$ greater than $T_{mem}$, the algorithm is a CPU-bound algorithm. Otherwise, it is a memory-bound algorithm.

3.1.1 CPU-bound Algorithms

If an algorithm has computation time $T_{comp}$ longer than data-accessing time $T_{mem}$ (i.e., CPU-bound algorithms), the EPEM energy complexity model be-
comes Equation 11 which is simplified as Equation 12
\[ E = \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O + P_{sta} \times \frac{Span \times F}{Freq} \] (11)
or
\[ E = \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O + \pi_{op} \times Span \] (12)

3.1.2 Memory-bound Algorithms
If an algorithm has data-accessing time longer than computation time (i.e., memory-bound algorithms): \( T_{mem} \geq T_{comp} \), energy complexity becomes Equation 13 which is simplified as Equation 14.
\[ E = \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O + P_{sta} \times I/O \times \frac{Span \times M}{Work \times Freq} \] (13)
or
\[ E = \epsilon_{op} \times Work + \epsilon_{I/O} \times I/O + \pi_{I/O} \times \frac{I/O \times Span}{Work} \] (14)

3.2 Platform-independent Energy Complexity Model
This section describes the energy complexity model that is platform-independent and considers only algorithm characteristics. When the platform parameters (i.e., \( \epsilon_{op}, \epsilon_{I/O}, \pi_{op}, \) and \( \pi_{I/O} \)) are unavailable, the energy complexity model is derived from Equation 6, where the platform parameters are constants and can be removed. Assuming \( \pi_{max} = \max(\pi_{op}, \pi_{I/O}) \), after removing platform parameters, the platform-independent energy complexity model are shown in Equation 15.
\[ E = O(Work + I/O + \max(Span, \frac{I/O \times Span}{Work})) \] (15)

4 A Case Study of Sparse Matrix Multiplication
SpMV is one of the most common application kernels in Berkeley dwarf list. It computes a vector result \( y \) by multiplying a sparse matrix \( A \) with a dense vector \( x \): \( y = Ax \). SpMV is a data-intensive kernel and has irregular memory-access patterns. The data access patterns for SpMV is defined by its sparse matrix format and matrix input types. There are several sparse matrix formats and SpMV algorithms in literature. To name a few, they are Coordinate Format (COO), Compressed Sparse Column (CSC), Compressed Sparse Row (CSR), Compressed Sparse Block (CSB), Recursive Sparse Block (RSB), Block Compressed Sparse Row (BCSR) and so on. Three popular SpMV algorithms, namely CSC, CSB and CSR are chosen to validate the proposed energy complexity model. They have different data-accessing patterns leading to different values of I/O, work and span complexity. Since SpMV is a memory-bound application kernel, Equation 14 is applied. The input matrices of SpMV have different parameters listed in Table 4.
Table 4: SpMV Input Parameter Description

| SpMV Input | Description                                      |
|------------|--------------------------------------------------|
| n          | Number of rows                                  |
| nz         | Number of nonzero elements                      |
| nr         | Maximum number of nonzero in a row              |
| nc         | Maximum number of nonzero in a column           |
| β          | Size of a block                                 |

4.1 Compressed Sparse Row

CSR is a standard storage format for sparse matrices which reduces the storage of matrix compared to the tuple representation \[21\]. This format enables row-wise compression of \(A\) with size \(n \times n\) (or \(n \times m\)) to store only the non-zero \(nz\) elements. Let \(nz\) be the number of non-zero elements in matrix \(A\). The work complexity of CSR SpMV is \(\Theta(nz)\) where \(nz \geq n\) and span complexity is \(O(nr + \log n)\) \[8\], where \(nr\) is the maximum number of non-zero elements in a row. The I/O complexity of CSR in the sequential I/O model of row-major layout is \(O(nz)\) \[5\] namely, scanning all non-zero elements of matrix \(A\) costs \(O(\frac{nz}{B})\) I/Os with \(B\) is the cache block size. However, randomly accessing vector \(x\) causes the total of \(O(nz)\) I/Os. Applying the proposed model on CSR SpMV, their total energy complexity are computed as Equation 16.

\[
E_{CSR} = O(\epsilon_{op} \times nz + \epsilon_{I/O} \times nz + \pi_{I/O} \times (nr + \log n)) 
\]  
(16)

4.2 Compressed Sparse Column

CSC is the similar storage format for sparse matrices as CSR. However, it compresses the sparse matrix in column-wise manner to store the non-zero elements. The work complexity of CSC SpMV is \(\Theta(nz)\) where \(nz \geq n\) and span complexity is \(O(nc + \log n)\), where \(nc\) is the maximum number of non-zero elements in a column. The I/O complexity of CSC in the sequential I/O model of column-major layout is \(O(nz)\) \[5\]. Similar to CSR, scanning all non-zero elements of matrix \(A\) in CSC format costs \(O(\frac{nz}{B})\) I/Os. However, randomly updating vector \(y\) causing the bottle neck with total of \(O(nz)\) I/Os. Applying the proposed model on CSC SpMV, their total energy complexity are computed as Equation 17.

\[
E_{CSC} = O(\epsilon_{op} \times nz + \epsilon_{I/O} \times nz + \pi_{I/O} \times (nc + \log n)) 
\]  
(17)

4.3 Compressed Sparse Block

Given a sparse matrix \(A\), while CSR has good performance on SpMV \(y = Ax\), CSC has good performance on transpose sparse matrix vector multiplication \(y =\)
\[ E_{CSB} = O(\epsilon_{op} \times \left( \frac{n^2}{\beta^2} + nz \right) + \epsilon_{I/O} \times \left( \frac{n^2}{\beta^2} + \frac{nz}{B} \right) + \pi_{I/O} \times \left( \frac{n^2}{\beta^2} + \frac{nz}{B} \right) \times \left( \beta \times \log \frac{n}{\beta} + \frac{n}{\beta} \right) \times \left( \frac{n^2}{\beta^2} + nz \right) \times \left( \beta \times \log \frac{n}{\beta} + \frac{n}{\beta} \right) \) \]

(18)

Table 5: SpMV Complexity Analysis

| Complexity | CSC | CSB | CSR |
|------------|-----|-----|-----|
| Work       | \(\Theta(nz)\) [8] | \(\Theta\left(\frac{n^2}{\beta^2} + nz\right)\) [8] | \(\Theta(nz)\) [8] |
| I/O        | \(O(nz)\) [5] | \(O\left(\frac{n^2}{\beta^2} + \frac{nz}{B}\right)\) [this study] | \(O(nz)\) [5] |
| Span       | \(O(nc + \log n)\) [8] | \(O(\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})\) [8] | \(O(nr + \log n)\) [8] |

AT \times x, Compressed sparse blocks (CSB) format is efficient for computing either \(Ax\) or \(A^T x\). CSB is another storage format for representing sparse matrices by dividing the matrix \(A\) and vector \(x, y\) to blocks. A block-row contains multiple chunks, each chunk contains consecutive blocks and non-zero elements of each block are stored in Z-Morton-ordered [8]. From Beluc et al. [8], CSB SpMV computing a matrix with \(nz\) non-zero elements, size \(n \times n\) and divided by block size \(\beta \times \beta\) has span complexity \(O(\beta \times \log \frac{n}{\beta} + \frac{n}{\beta})\) and work complexity as \(\Theta\left(\frac{n^2}{\beta^2} + nz\right)\).

I/O complexity for CSB SpMV is not available in the literature. We do the analysis of CSB manually by following the master method [11]. The I/O complexity is analyzed for the algorithm CSB\_SpMV(A,x,y) from Beluc et al. [8]. The I/O complexity of CSB is similar to work complexity of CSB \(O\left(\frac{n^2}{\beta^2} + nz\right)\), only that non-zero accesses in a block is divided by \(B\): \(O\left(\frac{n^2}{\beta^2} + \frac{nz}{B}\right)\), where \(B\) is cache block size. The reason is that non-zero elements in a block are stored in Z-Morton order which only requires \(\frac{nz}{B}\) I/Os. The energy complexity of CSB SpMV is shown in Equation 18.

From the complexity analysis of SpMV algorithms using different layouts, the complexity of CSR, CSC and CSB are summarized in Table 5.

5 Validation of EPEM Model with SpMV

This section describes the experimental study to validate the EPEM model, including: describing SpMV implementation and sparse matrix types used in this validation (cf. Section 5.1), introducing the two experimental platforms and how to obtain their parameters for the EPEM model (cf. Section 5.3) and discussing the validation results.
5.1 SpMV Implementation

We want to conduct complexity analysis and experimental study with two SpMV algorithms, namely CSB and CSC. Parallel CSB and sequential CSC implementations are available thanks to the study from Buluç et al. [8]. Since the optimization steps of available parallel SpMV kernels (e.g., pOSKI [1], LAMA [13]) might affect the work complexity of the algorithms, we decided to implement a simple parallel CSC using Cilk and pthread. To validate the correctness of our parallel CSC implementation, we compare the vector result \( y \) from \( y = A \ast x \) of CSC and CSB implementation. The comparison shows the equality of the two vector results \( y \). Moreover, we compare the performance of the our parallel CSC code with Matlab parallel CSC-SpMV kernel. Matlab also uses CSC layout as the format for their sparse matrix [16]. Our CSC implementation has out-performed Matlab parallel CSC kernel when computing the same targeted input matrices. Figure 2 shows the performance comparison of our CSC SpMV implementation and Matlab CSC SpMV kernel. The experimental study of SpMV energy consumption is then conducted with CSB SpMV implementation from Buluç et al. [8] and our CSC SpMV parallel implementation.

5.2 SpMV Matrix Input Types

We conducted the experiments with nine different matrix-input types from Florida sparse matrix collection [12]. Each matrix input has different properties listed in Table 4 including size of the matrix \( n \times m \), the maximum number of non-zero of the sparse matrix \( nz \), the maximum number of non-zero elements in one column \( nc \). Table 6 lists the matrix types used in this experimental validation with their properties.
Table 6: Sparse matrix input types. The maximum number of non-zero elements in a column $nc$ is derived from $S$.

| Matrix type     | n      | m      | nz        | nc       |
|-----------------|--------|--------|-----------|----------|
| bone010         | 986703 | 986703 | 47851783  | 63       |
| kkt_power       | 2063494| 2063494| 12771361  | 90       |
| ldoor           | 952203 | 952203 | 42493817  | 77       |
| parabolic_fem   | 525825 | 525825 | 3674625   | 7        |
| pds-100         | 156243 | 517577 | 1096002   | 7        |
| rajat31         | 4690002| 4690002| 20316253  | 1200     |
| Rucci1          | 1977885| 109900 | 7791168   | 108      |
| sme3Dc          | 42930  | 42930  | 3148656   | 405      |
| torso1          | 116158 | 116158 | 8516500   | 1200     |

5.3 Experiment Set-up

For the validation of the EPEM model, we conduct the experiments on two HPC platforms: one platform with two Intel Xeon E5-2650l v3 processors and one platform with Xeon Phi 31S1P processor. The Intel Xeon platform has two processors Xeon E5-2650l v3 with $2 \times 12$ cores, each processor has the frequency 1.8 GHz. The Intel Xeon Phi platform has one processor Xeon Phi 31S1P with 57 cores and its frequency is 1.1 GHz. To measure energy consumption of the platforms, we read the PCM MSR counters for Intel Xeon and MIC power reader for Xeon Phi.

5.4 Identifying Platform Parameters

We apply the energy roofline approach [10, 9] to find the platform parameters for the two new experimental platforms, namely Intel Xeon E5-2650l v3 and Xeon Phi 31S1P. Moreover, the energy roofline study [9] has also provided a list of other platforms including CPU, GPU, embedded platforms with their parameters considered in the Roofline model. Thanks to authors Choi et al. [9], we extract the required values of EPEM parameters for nine platforms presented in their study.

The EPEM parameter values of the two new HPC platforms (i.e., Xeon and Xeon-Phi 31S1P) used to validate the EPEM model are obtained by using the same approach as energy roofline study [10]. We create micro-benchmarks for the two platforms and measure their energy consumption and performance. The EPEM parameter values of each platform are obtained from energy and performance data by regression techniques. Along with the two HPC platforms used in this validation, we provide parameters required in the EPEM model for a total of 11 platforms. Their platform parameters are listed in Table 5 for further uses.
Figure 3: Energy consumption comparison between CSC-SpMV and CSB-SpMV on the Intel Xeon platform, computed by $E_{CSC} / E_{CSB}$. Both the EPEM model estimation and experimental measurement on Intel Xeon platform show the consistent results that $E_{CSC} / E_{CSB}$ is greater than 1, meaning CSC SpMV algorithm consumes more energy than the CSB SpMV algorithm on different input matrices.

5.5 Validating EPEM Using Different SpMV Algorithms

From the model-estimated data, CSB SpMV consumes less energy than CSC SpMV on both platforms. Even though CSB has higher work complexity than CSC, CSB SpMV has less I/O complexity than CSC SpMV. Firstly, the dynamic energy cost of one I/O is much greater than the energy cost of one operation (i.e., $\epsilon_{I/O} >> \epsilon_{op}$) on both platforms. Secondly, CSB has better parallelism than CSC, computed by $\frac{Work}{Span}$, which results in shorter execution time. Both reasons contribute to the less energy consumption of CSB SpMV. The measurement data confirms that CSB SpMV algorithm consumes less energy than CSC SpMV algorithm, shown by the energy consumption ratio between CSC-SpMV and CSB-SpMV greater than 1 in the Figure 3 and 4. For all input matrices, the EPEM model has confirmed that CSB SpMV consumes less energy than CSC SpMV algorithm.
Table 7: Comparison of Energy Consumption of Different Matrix Input Types.

| Platform   | Xeon | Xeon | Xeon | Xeon | Xeon-Phi | Xeon-Phi | Xeon-Phi |
|------------|------|------|------|------|----------|----------|----------|
| Model/Exprmt | model | exprmt | model | exprmt | model | exprmt | model | exprmt |
| Increasing | sme3Dc | pds-100 | pds-100 | pds-100 | pds-100 | pds-100 | parabolic | parabolic |
| Energy     | torsol | parabolic | sme3Dc | parabolic | torsol | parabolic | sme3Dc | pds-100 |
| Consumption | pds-100 | parabolic | sme3Dc | parabolic | pds-100 | Rucci1 | parabolic | Rucci1 |
| Order      | parabolic | torso1 | sme3Dc | parabolic | torso1 | rajat31 | parabolic | rajat31 |
| Rucci1     | kkt | torso1 | kkt | ldoor | kkt | kkt | kkt |
| torso1     | rajat31 | rajat31 | rajat31 | Rucci1 | rajat31 | rajat31 | ldoor |
| ldoor      | rajat31 | rajat31 | rajat31 | rajat31 | rajat31 | rajat31 | ldoor |
| bone010    | rajat31 | bone010 | bone010 | rajat31 | bone010 | bone010 | bone010 |
| rajat31    | bone010 | bone010 | bone010 | bone010 | bone010 | bone010 | bone010 |

Table 8: CSC Energy Comparison of Different Input Matrix Types on Xeon

| Correctness | pds-100 | parabolic | sme3Dc | Rucci1 | kkt | torso1 | rajat31 | ldoor | bone010 |
|-------------|--------|-----------|-------|--------|-----|--------|---------|-------|---------|
| pds-100     | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| parabolic   | x      | 0         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| sme3Dc      | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| Rucci1      | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| kkt         | x      | 0         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| torso1      | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| rajat31     | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| ldoor       | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |
| bone010     | x      | 1         | 1     | 1      | 1   | 1      | 1       | 1     | 1       |

Table 9: Comparison accuracy of SpMV energy consumption computing different input matrix types

| Algorithm | CSB | CSC |
|-----------|-----|-----|
| Xeon      | 75% | 94% |
| Xeon Phi  | 63.8% | 80.5% |
Figure 4: Energy consumption comparison between CSC-SpMV and CSB-SpMV on the Intel Xeon Phi platform, computed by $\frac{E_{\text{CSC}}}{E_{\text{CSB}}}$. Both the EPEM model estimation and experimental measurement on Intel Xeon Phi platform show the consistent results that $\frac{E_{\text{CSC}}}{E_{\text{CSB}}}$ is greater than 1, meaning CSC SpMV algorithm consumes more energy than the CSB SpMV algorithm on different input matrices.

5.6 Validating EPEM Using Different Input Types

To validate the EPEM model regarding input types, the experiments have been conducted with nine matrix types listed in Table 6. The model can capture the energy-consumption relation among different inputs. The increasing order of energy consumption of different matrix-input types are shown in Table 7, from both model estimation and experimental study.

For instance, in order to validate the comparison of energy consumption for different input types, a validated table as Table 8 is created for CSC SpMV on Xeon to compare model prediction and experimental measurement. For nine input types, there are $\frac{9 \times 9}{2} - 9 = 36$ input relations. If the relation is correct, meaning both experimental data and model data are the same, the relation value in the table of two inputs is 1. Otherwise, the relation value is 0. From Table 8, there are 34 out of 36 relations are the same for both model and experiment, which gives 94% accuracy on the relation of the energy consumption of different inputs. Similarly, the input validation for CSC and CSB on both Xeon and Xeon Phi platforms is provided in Table 9.
5.7 Validating The Applicability of EPEM on Different Platforms

The energy comparison of CSB and CSC SpMV is concluded for eleven platforms listed in Table 3. Like two Xeon and Xeon Phi 31S1P platforms used in experiments, Figure 5 shows the prediction that CSB SpMV consumes less energy than CSC SpMV, on all platforms listed in Table 3. This confirms the applicability of EPEM model to compare the energy consumption of algorithms for different input types on different platforms.

6 A Case Study of Dense Matrix Multiplication

Besides SpMV, we also apply the EPEM model to dense matrix multiplication (matmul). Unlike SpMV, a data-intensive kernel, matmul is a computation-intensive kernel used in high performance computing. It computes output matrix C (size $n \times p$) by multiplying two dense matrices A (size $n \times m$) and B (size $m \times p$): $C = A \times B$. In this work, we implemented two matmul algorithms (i.e., a simple algorithm and a cache-oblivious algorithm [14]) and apply the EPEM analysis to find their energy complexity. Both algorithms partition matrix A and C to equally to N sub-matrices (e.g., $A_i$ with $i=(1,2,...,N)$), where N is the number of cores in the platform. The partition approach is shown in Figure 6. Each core computes a sub-matrix $C_i$: $C_i = A_i \times B$. Since matmul is a computation-bound application kernel, Equation 12 is applied.

6.1 Simple Matmul Algorithm

The simple matmul algorithm is described in Figure 7. Its work complexity is $\Theta(2nmp)$ [28] and span complexity is $\Theta(\frac{2nmp}{N})$ because the computational work is divided equally to N cores due to matrix partition approach. When matrix
Figure 6: Partition approach for parallel matmul algorithms. Each sub-matrix \( A_i \) has size \( \frac{n}{N} \times m \) and each sub-matrix \( C_i \) has size \( \frac{n}{N} \times p \).

size of \( B \) is bigger than platform cache size matrix, the simple algorithm loads matrix \( B \) \( n \) times (i.e., once for computing each row of \( C \)), results in \( \frac{nmp}{B} \) cache block transfer, where \( B \) is cache block size. In total, I/O complexity of the simple matmul algorithm is \( \Theta\left(\frac{nm+nmp+np}{B}\right) \). Applying the EPEM model on this algorithm, the total energy complexity are computed as Equation \( 19 \).

\[
E_{\text{simple}} = O(\epsilon_{op} \times 2nmp + \epsilon_{I/O} \times \frac{nm + nmp + np}{B} + \pi_{op} \times \frac{2nmp}{N}) \tag{19}
\]

6.2 Cache-oblivious Matmul Algorithm

The cache-oblivious matmul (CO-matmul) algorithm \[14\] is a divide-and-conquer algorithm. It has work complexity the same as the simple matmul algorithm \( \Theta(2nmp) \). Its span complexity is also \( \Theta\left(\frac{2nmp}{N}\right) \) because of the used matrix partition approach shown in Figure 6. The I/O complexity of CO-matmul, however, is different from the simple algorithm: \( \Theta(n + m + p + \frac{nm+nmp+np}{B} + \frac{nmp}{B\sqrt{Z}}) \) \[14\].
Applying the EPEM model to CO-matmul, the total energy complexity are computed as Equation 20.

6.3 Validating EPEM With Matmul Algorithms

From the model-estimated data, Simple-Matmul consumes more energy than CO-Matmul on both platforms. Even though both algorithms have the same work and span complexity, Simple-Matmul has more I/O complexity than CO-Matmul, which results in greater energy consumption of Simple-Matmul compared to CO-Matmul algorithm. The measurement data confirms that Simple-Matmul algorithm consumes more energy than CO-Matmul algorithm, shown by the energy consumption ratio between Simple-Matmul and CO-Matmul greater than 1 in the Figure 8 and 9. For all input matrices, the EPEM model has confirmed that Simple-Matmul consumes more energy than CO-Matmul algorithm.

7 Related Work - Overview of energy models

Energy models for finding energy-optimized system configurations for a given application have been recently reported [12, 16, 19]. Imes et al. [18] used controller theory and linear programming to find energy-optimized configurations for an application with soft real-time constraints at runtime. Mishra et al. [24] used hierarchical Bayesian model in machine learning to find energy-optimized configurations. They used offline learning to train the Bayesian model with a training set of applications with different patterns, and used online learning to quickly estimate the optimal configuration for a given application. Snowdon et al. [26] developed a power management framework called Koala which models the energy consumption of the platform and monitors an application’s energy behavior. By matching an application’s behavior with the system policy, an
energy consumption comparison between Simple-Matmul and CO-Matmul on the Intel Xeon platform, computed by $\frac{E_{\text{Simple}}}{E_{\text{CO}}}$. Both the EPEM model estimation and experimental measurement on Intel Xeon platform show that $\frac{E_{\text{Simple}}}{E_{\text{CO}}}$ is greater than 1, meaning Simple-Matmul algorithm consumes more energy than the CO-Matmul algorithm.

energy-optimized configuration is determined at runtime. Although the energy models for finding energy-optimized system configurations have resulted in energy saving in practice, they focus on characterizing system platforms rather than applications and therefore are not appropriate for analyzing the energy complexity of application algorithms.

Another direction of energy modeling study is to predict the energy consumption of applications by analyzing applications without actual execution on real platforms which we classify as analytic models.

Energy roofline models [10, 9] are some of the comprehensive energy models that abstract away possible algorithms in order to analyze and characterize different multicore platforms in terms of energy consumption. The models abstract possible algorithms by their operational intensity, the ratio of computation to communication (i.e., flop/byte), and characterize a platform’s properties by running a set of micro-benchmarks on the platform. Our new energy model, which abstracts away possible multicore platform and characterize the energy complexity of algorithms based on their work, span and I/O complexity, complements the energy roofline models.

Validated energy models for specific algorithms have been reported recently [2, 23]. Alonso et al. [2] provided an accurate energy model for three key dense matrix factorizations. Malossi et al. [23] focused on basic linear-algebra kernels and characterized the kernels by the number of arithmetic operations, memory
Figure 9: Energy consumption comparison between Simple-Matmul and CO-Matmul on the Intel Xeon Phi platform, computed by $\frac{E_{\text{Simple}}}{E_{\text{CO}}}$. Both the EPEM model estimation and experimental measurement on Intel Xeon Phi platform show that $\frac{E_{\text{Simple}}}{E_{\text{CO}}}$ is greater than 1, meaning Simple-Matmul algorithm consumes more energy than the CO-Matmul algorithm.

accesses, reduction and barrier steps. Although the energy models for specific algorithms are accurate for the target algorithms, they are not applicable for other algorithms and therefore cannot be used as general energy complexity models for parallel algorithms.

The energy scalability of a parallel algorithm has been investigated by Korthikanti et al. [19, 20]. The energy scalability studies are to find the optimal number of cores for a given algorithm with a real-time constraint which minimizes energy consumption. Our studies complement the energy scalability studies by addressing the following energy complexity question: *Given two parallel algorithms A and B for a given problem, which algorithm consumes less energy analytically?* Unlike the energy scalability studies that have not been validated on real platforms, our new energy complexity model is validated on HPC and accelerator platforms, confirming its usability and accuracy.

The energy complexity of sequential algorithms on a uniprocessor machine with several memory banks has been studied by Roy et al. [25]. Our energy complexity studies complement Roy et al.’s studies by investigating the energy complexity of parallel algorithms on a multiprocessor machine with a shared memory bank and private caches, a machine model that has been widely adopted to study parallel algorithms [15, 3, 20].
8 Conclusion

In this study, we have devised a new general model for analyzing the energy complexity of multithreaded algorithms. The energy complexity of an algorithm is derived from its work, span, and I/O complexity. Moreover, two case studies are conducted to demonstrate how to use the model to analyze the energy complexity of SpMV algorithms and matmul algorithms. The energy complexity analyses are validated for two SpMV algorithms and two matmul algorithm on two HPC platforms with different input matrices. The experimental results confirm the theoretical analysis with respect to which algorithm consumes more energy. The EPEM energy complexity model gives the algorithm designers the insight into which design of algorithm should be used to minimize energy consumption.

References

[1] “Poski: Parallel optimized sparse kernel interface,” http://bebop.cs.berkeley.edu/poski, accessed: 2015-11-17.

[2] P. Alonso, M. F. Dolz, R. Mayo, and E. S. Quintana-Orti, “Modeling power and energy consumption of dense matrix factorizations on multicore processors,” Concurrency Computat., 2014.

[3] L. Arge, M. T. Goodrich, M. Nelson, and N. Sitchinava, “Fundamental parallel algorithms for private-cache chip multiprocessors,” in Proceedings of the Twentieth Annual Symposium on Parallelism in Algorithms and Architectures, ser. SPAA '08. New York, NY, USA: ACM, 2008, pp. 197–206. [Online]. Available: http://doi.acm.org/10.1145/1378533.1378573

[4] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plisker, J. Shalf, S. W. Williams, and K. A. Yelick, “The landscape of parallel computing research: A view from berkeley,” Technical Report No. UCB/EECS-2006-183, University of California, Berkeley, 2006.

[5] M. Bender, G. Brodal, R. Fagerberg, R. Jacob, and E. Vicari, “Optimal sparse matrix dense vector multiplication in the i/o model,” Theory of Computing Systems, vol. 47, no. 4, pp. 934–962, 2010. [Online]. Available: http://dx.doi.org/10.1007/s00224-010-9285-4

[6] G. E. Blelloch, J. T. Fineman, P. B. Gibbons, and H. V. Simhadri, “Scheduling irregular parallel computations on hierarchical caches,” in Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures, ser. SPAA '11, 2011, pp. 355–366.

[7] G. E. Blelloch, P. B. Gibbons, and Y. Matias, “Provably efficient scheduling for languages with fine-grained parallelism,” J. ACM, vol. 46, no. 2, pp. 281–321, 1999.
[8] A. Buluç, J. T. Fineman, M. Frigo, J. R. Gilbert, and C. E. Leiserson, “Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks,” in Proceedings of the Twenty-first Annual Symposium on Parallelism in Algorithms and Architectures, ser. SPAA '09. ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1583991.1584053

[9] J. Choi, M. Dukhan, X. Liu, and R. Vuduc, “Algorithmic time, energy, and power on candidate hpc compute building blocks,” in Proceedings of the 2014 IEEE 28th International Parallel and Distributed Processing Symposium, ser. IPDPS '14, Washington, DC, USA, 2014, pp. 447–457.

[10] J. W. Choi, D. Bedard, R. Fowler, and R. Vuduc, “A roofline model of energy,” in Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing, ser. IPDPS '13, Washington, DC, USA, 2013, pp. 661–672.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, Third Edition, 3rd ed. The MIT Press, 2009.

[12] T. A. Davis and Y. Hu, “The university of florida sparse matrix collection,” ACM Trans. Math. Softw., vol. 38, no. 1, pp. 1:1–1:25, Dec. 2011. [Online]. Available: http://doi.acm.org/10.1145/2049662.2049663

[13] M. Forster and J. Kraus, “Scalable parallel amg on ccnuma machines with openmp,” Computer Science - Research and Development, vol. 26, no. 3-4, pp. 221–228, 2011. [Online]. Available: http://dx.doi.org/10.1007/s00450-011-0159-z

[14] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, “Cache-oblivious algorithms,” in Proceedings of the 40th Annual Symposium on Foundations of Computer Science, ser. FOCS, 1999.

[15] M. Frigo and V. Strumpen, “The cache complexity of multithreaded cache oblivious algorithms,” in Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, ser. SPAA '06. New York, NY, USA: ACM, 2006, pp. 271–280. [Online]. Available: http://doi.acm.org/10.1145/1148109.1148157

[16] J. R. Gilbert, C. Moler, and R. Schreiber, “Sparse matrices in matlab: Design and implementation,” SIAM J. Matrix Anal. Appl., vol. 13, no. 1, pp. 333–356, Jan. 1992. [Online]. Available: http://dx.doi.org/10.1137/0613024

[17] P. Ha, V. Tran, I. Umar, P. Tsigas, A. Gidenstam, P. Renaud-Goud, I. Walulya, and A. Atalar, “Models for energy consumption of data structures and algorithms,” EU FP7 project EXCESS deliverable D2.1 (http://www.excess-project.eu), Tech. Rep., 2014.
[18] C. Imes, D. Kim, M. Maggio, and H. Hoffmann, “Poet: a portable approach to minimizing energy under soft real-time constraints,” in Real-Time and Embedded Technology and Applications Symposium (RTAS), 2015 IEEE, April 2015, pp. 75–86.

[19] V. Korthikanti and G. Agha, “Analysis of parallel algorithms for energy conservation in scalable multicore architectures,” in International Conference on Parallel Processing, 2009. ICPP '09., Sept 2009, pp. 212–219.

[20] V. A. Korthikanti and G. Agha, “Towards optimizing energy costs of algorithms for shared memory architectures,” in Proceedings of the Twenty-second Annual ACM Symposium on Parallelism in Algorithms and Architectures, ser. SPAA ’10. New York, NY, USA: ACM, 2010, pp. 157–165.

[21] V. Kotlyar, K. Pingali, and P. Stodghill, “A relational approach to the compilation of sparse matrix programs,” Ithaca, NY, USA, Tech. Rep., 1997.

[22] L. Li and C. Kessler, “Validating energy compositionality of GPU computations,” in HIPEAC Workshop on Energy Efficiency with Heterogeneous Computing (EEHCO), Jan. 2015.

[23] A. C. I. Malossi, Y. Ineichen, C. Bekas, A. Curioni, and E. S. Quintana-Orti, “Systematic derivation of time and power models for linear algebra kernels on multicore architectures,” Sustainable Computing: Informatics and Systems, vol. 7, pp. 24 – 40, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210537915000037

[24] N. Mishra, H. Zhang, J. D. Lafferty, and H. Hoffmann, “A probabilistic graphical model-based approach for minimizing energy under performance constraints,” in Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems, ser. ASPLOS ’15. New York, NY, USA: ACM, 2015, pp. 267–281. [Online]. Available: http://doi.acm.org/10.1145/2694344.2694373

[25] S. Roy, A. Rudra, and A. Verma, “An energy complexity model for algorithms,” in Proceedings of the 4th Conference on Innovations in Theoretical Computer Science, ser. ITCS ‘13. New York, NY, USA: ACM, 2013, pp. 283–304. [Online]. Available: http://doi.acm.org/10.1145/2422436.2422470

[26] D. C. Snowdon, E. Le Sueur, S. M. Petters, and G. Heiser, “Koala: A platform for os-level power management,” in Proceedings of the 4th ACM European Conference on Computer Systems, ser. EuroSys ’09. ACM, 2009. [Online]. Available: http://doi.acm.org/10.1145/1519065.1519097

[27] S. Williams, A. Waterman, and D. Patterson, “roofline: An insightful visual performance model for multicore architectures,” Commun. ACM, vol. 52, no. 4, pp. 65–76, 2009.
[28] K. Yelick. (2004, Sept) Cs 267 parallel matrix multiplication. [Online].
Available: http://www.cs.berkeley.edu/~yelick/cs267