Fill Estimation for Blocked Sparse Matrices and Tensors

by

Helen Jiang Xu

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science in in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

Author . . . .
Department of Electrical Engineering and Computer Science
May 23, 2018

Certified by . . . .
Charles E. Leiserson
Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Fill Estimation for

Blocked Sparse Matrices and Tensors

by

Helen Jiang Xu

Submitted to the Department of Electrical Engineering and Computer Science
on May 23, 2018, in partial fulfillment of the
requirements for the degree of
Master of Science in in Electrical Engineering and Computer Science

Abstract

Many sparse matrices and tensors from a variety of applications, such as finite element methods and computational chemistry, have a natural aligned rectangular nonzero block structure. Researchers have designed high-performance blocked sparse operations which can take advantage of this sparse structure to reduce the complexity of storing the locations of nonzeros. The performance of a blocked sparse operation depends on how well a particular blocking scheme, or tiling of the sparse matrix into blocks, reflects the structure of nonzeros in the tensor. Since sparse tensor structure is generally unknown until runtime, blocking-scheme selection must be efficient. The ***fill*** is a quantity which, for some blocking scheme, relates the number of nonzero blocks to the number of nonzeros. Many performance models use the fill to help choose a blocking scheme. The fill is expensive to compute exactly, however.

This thesis presents a sampling-based algorithm called PHIL that efficiently estimates the fill of sparse matrices and tensors in any format. Much of the thesis will appear in a paper coauthored with Peter Ahrens and Nicholas Schiefer. We provide theoretical guarantees for sparse matrices and tensors, and experimental results for matrices. The existing state-of-the-art fill-estimation algorithm, which we will call OSKI, runs in time linear in the number of elements in the tensor. In contrast, the number of samples PHIL needs to compute a fill estimate is unrelated to the number of nonzeros in the tensor.

We compared PHIL and OSKI on a suite of hundreds of sparse matrices and found that on most inputs, PHIL estimates the fill at least 2 times faster and often more than 20 times faster than OSKI. PHIL consistently produced accurate estimates and was faster and/or more accurate than OSKI on all cases. Finally, we found that PHIL and OSKI produced comparable speedups in parallel blocked sparse matrix-vector multiplication.

Thesis Supervisor: Charles E. Leiserson
Title: Professor of Computer Science and Engineering

Acknowledgments

I am grateful to my advisor, Charles Leiserson, for his guidance throughout the course of this thesis. Despite the challenges along the road to publication of this work, he has been nothing but supportive. Specifically, my writing and technical presentations would be much less intelligible without his advice.

My coauthors Peter Ahrens and Nicholas Schiefer have been invaluable to this thesis not only as technical collaborators but also as good friends. Our IPDPS paper [1] will contain much of the content of this thesis.

Also, I would like to thank the Supertech research group for listening to my presentations and providing feedback, answering any and all questions I have, and being a great group of researchers to learn from.

Finally, I would like to thank my family and friends without whom this thesis (and everything else) would not have been possible.

This work was supported in part by NSF Grants 1314547 and 1533644 as well as a National Physical Sciences Consortium Fellowship.

Contents

Chapter 1

Introduction

In the spring of 2017, Peter Ahrens came to me and Nicholas Schiefer with the "fill-estimation problem" and an idea for a randomized sampling-based algorithm (which we later named PHIL) for approximating a property of blocked sparse matrices called the "fill". Practitioners developed blocked sparse storage formats to exploit the natural blocked structure of some sparse matrices for performance optimizations. Im *et al.* [14] introduced a quantity called the ***fill***, or the ratio of introduced zeros to the original number of nonzeros, to determine an optimal blocking for a given sparse matrix. The fill measures how well each blocking captures the natural blocked structure of a given sparse matrix. Vuduc *et al.* [28] then showed that choosing the correct matrix blocking can speed up sparse matrix-vector multiplication, a common numerical kernel, by more than a factor of 2 on matrices with blocked structure.

Since computing the fill exactly may take hundreds of times the cost of one sparse matrix-vector multiplication, researchers developed heuristics for estimating the quantity with reasonable accuracy. Vuduc *et al.* [26] proposed a randomized algorithm for estimating the fill of a sparse matrix. We call this fill-estimation algorithm OSKI since Vuduc *et al.* implemented the algorithm in the Optimized Sparse Kernel Interface (OSKI) [27]. OSKI approximates the fill much more quickly than exact algorithms and demonstrates the potential for randomized algorithms in computing the fill. Vuduc *et al.* [26] showed that OSKI empirically approximates the fill with reasonable error but lacks theoretical guarantees about either its accuracy or runtime.

Peter, Nicholas, and I decided to work on the "fill-estimation problem" and explore the potential for a fill-estimation algorithm with provable guarantees about its accuracy and runtime. We devised PHIL, a sampling-based fill-estimation algorithm that requires a number of samples independent of the input size and has both accuracy and runtime guarantees. We then showed empirically that PHIL estimates the fill faster than OSKI and generated pathological inputs for OSKI where it does not provide any useful estimate of the fill.

This thesis contains my joint work with Peter Ahrens and Nicholas Schiefer on PHIL, as well as additional experimental results that I did myself. Our joint work will appear in [1], In this thesis, I review prior work on unblocked and blocked sparse storage formats, the role of the fill in performance modeling of blocked sparse kernels, and OSKI. Finally, I conclude with PHIL's theoretical guarantees and an empirical evaluation of PHIL and OSKI.

Sparse Matrices

Sparse matrices allow performance engineers to write fast algorithms and efficient data structures with complexity proportional to the number of nonzero entries. But sparse matrices introduce substantial storage and computational overhead per element. In contrast, dense formats have almost no computational overhead but may require much more space in total than sparse formats because they must store zeros. That is, ***the number*** $\boldsymbol{k(\mathcal{A})}$ ***of nonzero entries*** in an $m \times n$ sparse matrix $\mathcal{A}$ may be much smaller than $m \times n$. For example, Figure 1-1 compares the memory footprint of a matrix stored in a common sparse matrix format (Compressed Sparse Rows) and a matrix stored in a dense format, as a function of matrix density. Although sparse storage formats require extra space, they still may have an advantage over dense representations if the matrix has enough sparsity. Since sparse matrices have far more zeros than nonzeros, algorithms for sparse matrices may admit substantial performance improvements in performance over algorithms for dense matrices.

For example, sparse matrix-vector multiplication (SpMV) is one of the most heavily used numerical kernels in scientific computing because of its performance compared to

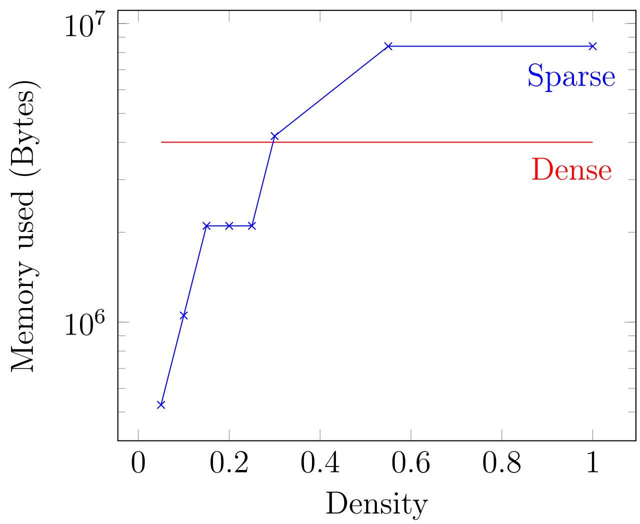


Figure 1-1: Size of a random sparse matrix $\mathcal{A}$ with $n = 1000$ and varying sparsity. For comparison, the size of a dense representation is shown as well. We used a full n^2 matrix as the dense representation and Compressed Sparse Rows as the sparse matrix representation. The x-axis represents the matrix density (i.e., $k(\mathcal{A})\ /\ n^2$), while the y-axis represents the size of the matrix representation.

dense implementations. Unfortunately, parallel implementations of SpMV are usually limited by memory bandwidth [6, 29]. Sparse matrix-vector multiplication on purely sparse matrix formats that store nonzeros individually usually results in irregular memory traffic due to the locations of the nonzeros.

Blocked Formats

Blocked matrices and tensors (multidimensional generalizations of matrices) often appear in scientific computing. Specifically, sparse matrices from finite element methods [26] and sparse tensors from quantum chemistry [8] both exhibit regular block structure.

Since blocked structure varies across different sparse tensors, storage formats that take advantage of natural blocked structure must choose "blocking schemes" according to the structure of a tensor to avoid unnecessary overhead.

Definition 1.1 (Blocking Scheme) *Suppose that $\mathcal{A}$ is a tensor of with R dimensions, or an* ***R-tensor****. A* ***blocking scheme*** *for $\mathcal{A}$ is a vector $\mathbf{b}$ of R block sizes $(b_1, b_2, \ldots, b_R)$ such that for all $i = 1, 2, \ldots R$, $i \in \mathbb{N}$. A blocking scheme $\mathbf{b} = (b_1, b_2, \ldots, b_R)$ applied to a tensor $\mathcal{A}$ tiles $\mathcal{A}$ into blocks of size $b_1 \times b_2 \times \ldots \times b_R$.*

For convenience, blocking schemes are sometimes called blockings.

Figure 1-2 shows an example of a blocking scheme $\mathbf{b} = (2, 3)$ on a sparse matrix. If any entry b_i does not divide the corresponding tensor dimension evenly, one can pad the tensor to the nearest next multiple of b_i.

Researchers have developed ***blocked formats*** which store dense blocks of nonzeros instead of storing the nonzeros individually to take advantage of the natural blocked structure of some blocked sparse matrices and tensors. Blocked formats may also represent some zeros explicitly if they appear in nonempty blocks as shown in Figure 1-2. Several storage formats and tensors reduce the complexity of storing individual entries by taking advantage of structural patterns in the locations of nonzeros [2, 6, 16, 22, 30]. The exact representation of a tensor in a blocked format depends on the selected blocking scheme.

Blocked storage formats are hybrid storage formats between fully sparse and dense storage formats and therefore take advantage of both sparsity and dense subarrays while reducing overhead. They simplify memory traffic and admit performance optimizations such as vectorization [16].

Whether a blocking scheme captures the structure of a sparse tensor determines the performance of a blocked sparse operation. Since zeros in the dense blocks must be stored explicitly, an ideal blocking scheme would perform well on a given architecture while minimizing the "filling in," or explicit representation, of zeros. The quality of a given blocking scheme depends on how well it captures the structure of the sparse tensor. A blocking scheme that fails to capture the structural patterns of a sparse matrix may introduce storage overhead because of introduced zeros without yielding any performance benefits. Vuduc *et al.* [28] shows that choosing the correct blocking can speed up sparse matrix-vector multiplication by more than a factor of 2 on matrices with blocked structure.

The Fill in Performance Modeling

The benefits of blocked sparse formats raise a natural question: how do we choose an optimal blocking scheme for a sparse matrix or tensor?

To measure how well a blocking scheme captures the structure of a sparse tensor,

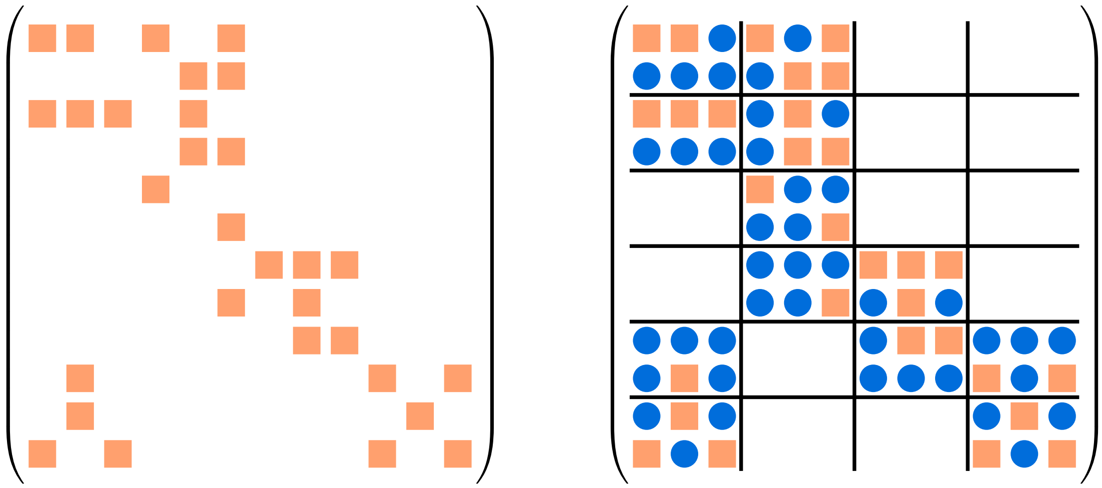


Figure 1-2: On the left, a sparse matrix before blocking. On the right, the same sparse matrix after blocking. The squares denote nonzero elements and circles are explicit zeros that are introduced due to the storage format. In this example, the blocking scheme $\mathbf{b} = (2, 3)$ and $k_{\mathbf{b}}(\mathcal{A}) = 12$. The number of nonzero elements $k(\mathcal{A}) = 30$, so the *fill* $f_{\mathbf{b}}(\mathcal{A}) = (2 \times 3 \times 12)/30 = 2.4$.

Im *et al.* [14] introduced a quantity called the fill. Given a sparse tensor $\mathcal{A}$ and a blocking $\mathbf{b}$, the ***fill*** $f_{\mathbf{b}}(\mathcal{A})$ is the ratio of introduced zeros to the original number $k(\mathcal{A})$ of nonzeros. Intuitively, a blocking scheme captures the structure of a sparse tensor well when it introduces relatively few explicit zeros. Since the fill is directly proportional to the number of filled-in zeros, it measures how well a blocking matches the blocked structure of a sparse matrix. Figure 1-2 shows the fill of a sparse matrix under blocking scheme $\mathbf{b} = (2, 3)$.

Researchers have developed "performance models" to determine an the performance of blocked sparse operations based on the structure of a sparse matrix $\mathcal{A}$ and a blocking scheme $\mathbf{b}$. A ***performance model*** of a tensor $\mathcal{A}$ under blocking scheme $\mathbf{b}$ on a machine M is a function $P : \mathbb{R} \to \mathbb{R}$ that maps the fill $f_{\mathbf{b}}(\mathcal{A})$ to the expected performance in in FLOP/s of a blocked sparse operation on $\mathcal{A}$ under $\mathbf{b}$.

The fill appears in performance models for a wide variety of blocked sparse kernels. Notably, it appears in several BCSR matrix-vector multiply performance prediction models [7, 13–15, 26–28] and performance models for for sparse triangular solve and sparse $\mathcal{A}^T\mathcal{A}\mathbf{x}$ [26]. The number of nonzero blocks (proportional to the fill) has been used in performance models for general blocked format sparse matrix-vector multiply [9, 17, 29]. Finally, an estimate of the fill can easily be added as an additional feature in feature-based machine learning approaches to sparse kernel performance modeling [20].

Example: SPARSITY *Performance Model for Blocked SpMV*

As an example, let us examine the SPARSITY performance model for blocked sparse matrix-vector multiply due to Vuduc *et al.* [28]. We call the model SPARSITY because it appears in the SPARSITY library. There are more accurate performance models which still depend on the fill, but we shall focus on computing the fill and not performance modeling. It was later shown that, when the fill is known exactly, performance of the resulting blocking scheme was optimal or within 5% of optimal [26].

The SPARSITY performance model P_{SPARSITY} is an empirical model that is computed once per machine type and then used many times for different tensors and blocking schemes. It takes as input a profile of how a given machine M performs on dense blocks over all blockings, as well as an estimate of the fill $f_{\mathbf{b}}(\mathcal{A})$ of a matrix $\mathcal{A}$ under blocking scheme $\mathbf{b}$. Once per machine, we compute a profile of how the machine performs for each blocking scheme. Let $\text{PERF}(\mathbf{b})$ be the performance of the machine (in FLOP/s) on a dense matrix stored with blocking scheme $\mathbf{b}$. The measure $\text{PERF}(\mathbf{b})$ indicates how efficiently we can process nonzeros when nonzeros are stored under $\mathbf{b}$. The SPARSITY model estimates the expected performance of a blocked SpMV (in FLOP/s) of $\mathcal{A}$ under $\mathbf{b}$, as $\text{PERF}(\mathbf{b})/f_{\mathbf{b}}(\mathcal{A})$, then chooses a blocking scheme that maximizes the estimated performance.

Computing the Fill in Practice

Computing the fill exactly over all blocking schemes often takes hundreds of times as long as a single sparse matrix-vector multiplication. Since the structure of the sparse tensor is generally not known before runtime, blocking scheme selection must occur at runtime and must therefore be efficient. Thus, our problem is to quickly compute an estimate of the fill over all blocking schemes with reasonable accuracy. Recently, Langr, Šimeček, and Dytrych [19] attempted to parallelize exact computation of the fill for matrices. They were only able to provide competitive results, however, by computing a much smaller number of quantities. Since blocking scheme selection remains a difficult problem for tensors as it is costly to compute the fill exactly, developers have adopted empirical search techniques [25].

Although we limit the limited number of blockings in the case of sparse-matrix vector multiplication, computing the fill exactly over all possible blockings is still too costly. For dense blocks in matrices, let us focus on blocking schemes $\mathbf{b} = (b_1, b_2)$ that are small enough to fit b_1 elements of the input vector, b_2 elements of the output vector, and at least one input matrix element in registers. In practice [26], this requirement usually limits our attention to $b_1, b_2 \leq 12$.

OSKI*: a Fill-estimation Algorithm*

Vuduc *et al.* [13, 26] introduced the OSKI algorithm, which is the first and (to our knowledge) only existing algorithm that estimates the fill instead of computing it exactly. OSKI is the first known algorithm to produce an empirically accurate approximation of the fill over all blocking schemes in reasonable time.

Given a maximum block size B, OSKI uses randomization to compute the fill over a subset of a sparse matrix. For each block row size $b_1 = 1, 2, \ldots, B$, OSKI samples a fraction of block rows. For each sampled block row, OSKI computes the fill exactly for all block column sizes $b_2 = 1, 2, \ldots, B$ simultaneously. OSKI does this by iterating through coordinates (i, j) of nonzeros in the block row and using a perfect hash table for each block column size to record the number of unique block column coordinates $(\lceil j/b_2 \rceil)$ seen. The fraction of block rows evaluated is specified by a parameter σ which is usually set to 0.02.

Although OSKI can estimate the fill of most matrices, it does not give predictable results. Notably, OSKI randomly samples block rows but may fail on matrices where the nonzeros are concentrated in a few rows because it may not evaluate those rows. In our work, we show that it is vulnerable to special cases. To our knowledge, there are no theoretical guarantees on the accuracy of OSKI, and no existing algorithm which estimates the fill of arbitrary tensors beyond matrices.

Moreover, OSKI lacks runtime guarantees. It samples random block rows and computes the fill based on all the nonzeros in those block rows. If OSKI samples

Property	OSKI	PHIL
Described for	Sparse matrices	Arbitrary sparse tensors
Implemented for	Sparse matrices	Sparse matrices
What it samples	Block rows	Nonzeros
Estimates fill over	All blockings	All blockings
Number of samples	$\sigma(m/B)$	$B^{2R}\ln(2B^R/\delta)/(2\epsilon^2)$
Operations to process a sample	$O(\sigma \cdot k(\mathcal{A}))$ (on average)	$(R+1)(2B)^R + B^R$
Error guarantee	None	Within a factor of ϵ

Figure 1-3: A comparison of OSKI and PHIL. OSKI requires the probability of sampling a block row σ and a sparse $m \times n$ matrix. PHIL computes an (ϵ, δ)- approximation of the fill of an R-tensor over all blockings with maximum block dimension B.

block rows with probability σ, it evaluates $\sigma \times k(\mathcal{A})$ nonzeros on average, where $k(\mathcal{A})$ is the number of nonzeros in the matrix $\mathcal{A}$. If most of the nonzeros were concentrated in the selected block rows, however, OSKI's runtime would be linear in the number of nonzeros.

Approximation Algorithms

PHIL does not guarantee to find the exact solution to the fill-estimation problem. It achieves theoretical guarantees on its accuracy based on the parameters ϵ and δ where ϵ is a multiplicative error bound and δ is a failure probability. We call such an algorithm an (ϵ, δ)-approximation algorithm.

An (ϵ, δ)-approximation algorithm guarantees concentration of an estimator around the actual quantity x we are trying to estimate.

Definition 1.2 *Let $\epsilon > 0, 1 > \delta > 0$. An (ϵ, δ)-approximation algorithm produces an approximation x^* to a quantity x such that*

$$(1-\epsilon)x \leq x^* \leq (1+\epsilon)x$$

with probability $1-\delta$.

Contributions

Our main contribution is PHIL, the first fill-estimation algorithm with provable guarantees for sparse matrices and tensors. PHIL is a sampling-based, (ϵ, δ)-approximation algorithm that randomly chooses a subset of the nonzeros in a tensor. PHIL uses prefix sums [4] to efficiently compute an estimate of the fill for all blocking schemes around each chosen nonzero.

PHIL takes as input the following parameters:

- a sparse R-tensor $\mathcal{A}$,
- the error bound ϵ,
- the failure probability δ,
- and the maximum block size B.

For an R-tensor (a tensor with R dimensions), the maximum block volume is therefore B^R.

Figure 1-3 summarizes the differences between PHIL and OSKI. We provide an exact bound on the number of samples that PHIL requires that *does not depend* on the number of nonzeros in the tensor. In contrast, OSKI runs in time linear in the number of nonzeros and is described only for matrices in one sparse format (CSR). As long as the tensor storage format allows fast (sublinear in the size of the input) access to elements of the tensor, PHIL runs in time sublinear in the number of nonzeros. Moreover, PHIL does not require a specific tensor storage format.

PHIL requires a number of samples and a total runtime independent of the size of the input tensor. Given an R-tensor and a maximum block size B, PHIL only needs $B^{2R} \ln(2B^R/\delta)/(2\epsilon^2)$ samples to compute an (ϵ, δ)-approximation. In addition to the time taken to find the neighboring nonzeros, each sample (for all B^R blocking schemes) can be processed with $(R+1)(2B)^R$ integer additions and B^R floating point divisions and additions.

We experimentally evaluated the runtime, accuracy, and resulting SpMV times of PHIL and OSKI on a large suite of sparse matrices. We demonstrated experimentally that PHIL provides more accurate estimates than OSKI, while requiring only half the time, and often outperforming OSKI by more than a factor of 20. PHIL consistently provided accurate results even when OSKI produced results with a complete loss of accuracy. In all cases we tested, PHIL was faster and/or more accurate than OSKI. PHIL and OSKI produced fill estimates that resulted in almost identical sparse matrix-vector multiplication times when we used the SPARSITY performance model to select a blocking scheme.

Our contributions are as follows:

- PHIL, the first probably accurate fill-estimation algorithm for arbitrary sparse tensors.
- A theorem proving that PHIL requires exactly $B^{2R} \ln(2B^R/\delta)/(2\epsilon^2)$ samples to compute an (ϵ, δ)-approximation of the true fill of an R-tensor over all block sizes given a maximum block dimension B.
- A scheme involving prefix sums that requires at most $(R+1)(2B)^R$ integer additions to process each sample.
- An implementation of PHIL in `C`.
- An empirical evaluation of PHIL and OSKI on a large suite of sparse matrices that shows PHIL estimated the fill over ten times faster than OSKI and yielded almost identical SpMV speedups.
- The construction, theoretical analysis, and empirical evaluation of pathological inputs for PHIL and OSKI.
- A parallel implementation of PHIL in `Cilk` [5], which demonstrates that PHIL can be efficiently parallelized.

Outline

The remainder of this thesis is organized as follows. Chapter 2 formalizes the mathematical preliminaries used in PHIL. Chapter 3 describes how PHIL samples nonzeros to estimate the fill. Chapter 4 proves worst-case error bounds on the fill estimate. Chapter 5 shows empirically that PHIL performs much better than its worst-case error bound. We conclude with open problems and extensions of PHIL in Chapter 6.

Chapter 2

Background

This chapter formalizes mathematical preliminaries required to understand PHIL. Since PHIL operates on sparse tensors, we review tensor notation. PHIL randomly samples nonzeros, and we use tensor notation to represent the location of samples. Next, we review various sparse tensor storage formats. Although PHIL does not require a specific storage format, we choose to explain PHIL in terms of the common Blocked Compressed Sparse Rows (BCSR). Finally, we formally define the ***fill-estimation problem*** as the problem of computing an (ϵ, δ)-approximation of the fill.

Tensor Notation

Tensors are multidimensional arrays over some field. Specifically, an R-tensor (tensor of order or rank R) is an array with R dimensions with elements from some field $\mathbb{F}$ (usually the real or complex numbers). We denote tensors by capital script letters $\mathcal{A}$ and vectors by lowercase boldface letters $\mathbf{a}$.

We now define how to index coordinates and ranges of coordinates in tensors. Let I_r be the size of the rth dimension of an R-tensor $\mathcal{A} \in \mathbb{F}^{I_1 \times I_2 \times \cdots \times I_R}$. A ***coordinate*** $\mathbf{i}$ is a list of R indices $(i_1, i_2, \ldots, i_R)$ where $1 \leq i_r \leq I_r$. We denote the element of $\mathcal{A}$ addressed by coordinate $\mathbf{i}$ as $\mathcal{A}[i_1, i_2, \ldots, i_R]$. For compactness of notation, we sometimes specify a coordinate as an R-component vector $\mathbf{i} = (i_1, i_2, \ldots, i_R)$. We represent the range of indices $i, i+1, \ldots, i'$ with the syntax $i : i'$. We represent a range of coordinates as $\mathbf{i} : \mathbf{i}'$, meaning $(i_1 : i'_1) \times \cdots \times (i_R : i'_R)$. Subtensors are formed when we fix a subset of coordinates. We also use ":" without bounds to indicate all elements along a particular dimension.

For convenience, we occasionally redefine the starting coordinate of a tensor. For example, the middle $n/2$ columns of a matrix $\mathcal{A} \in \mathbb{F}^{n\times n}$ are written $\mathcal{A}[:, n/4 : 3n/4]$. Thus, $\mathcal{A} \in \mathbb{F}^{\mathbf{I}:\mathbf{I}'}$ is an $(I_1' - I_1 + 1) \times \cdots \times (I_R' - I_R + 1)$ tensor whose smallest coordinate is $\mathbf{I}$ and largest coordinate is $\mathbf{I}'$.

We denote the number of nonzero entries in a tensor $\mathcal{A}$ as $k(\mathcal{A})$.

When we compare a vector to a scalar, our comparison is true if and only if the comparison is true for each entry of the vector pointwise. For example, a blocking scheme $\mathbf{b} \leq B$ if and only if for all $i = 1, 2, \ldots, R, b_i \leq B$.

Sparse Tensor Representations

Although we mention a few specific sparse formats, PHIL applies to any sparse tensor format which admits iteration over nonzero coordinates. Since most sparse formats store only the coordinates which correspond to nonzeros and the nonzero values themselves, PHIL applies to many different sparse storage formats.

The simplest sparse matrix and tensor format is ***Coordinate (COO)*** [2]. In this format, all coordinates which correspond to nonzeros are stored in an unordered list. Entries are stored in sorted order of their coordinates. Figure 2-1 shows an example of a matrix and its COO representation.

Perhaps the most popular sparse matrix format is ***Compressed Sparse Rows (CSR)*** [22]. In CSR format, the indices of nonzeros in each row are stored in sorted order. Each row has an associated list of coordinates of nonzeros. The nonzeros are stored in a single array with the same ordering as their coordinates. Figure 2-2 shows the same matrix from Figure 2-1 in CSR format.

CSR extends to tensor formats in many ways [2], such as ***Compressed Sparse Fibers (CSF)*** [18, 24]. In CSF format, each coordinate $\mathbf{i}$ is stored in a tree structure where a node in level r represents an index i_r that corresponds to a set of nonzeros. CSR is the matrix case of CSF.

Performance engineers use ***blocked storage formats*** to store blocks of nearby

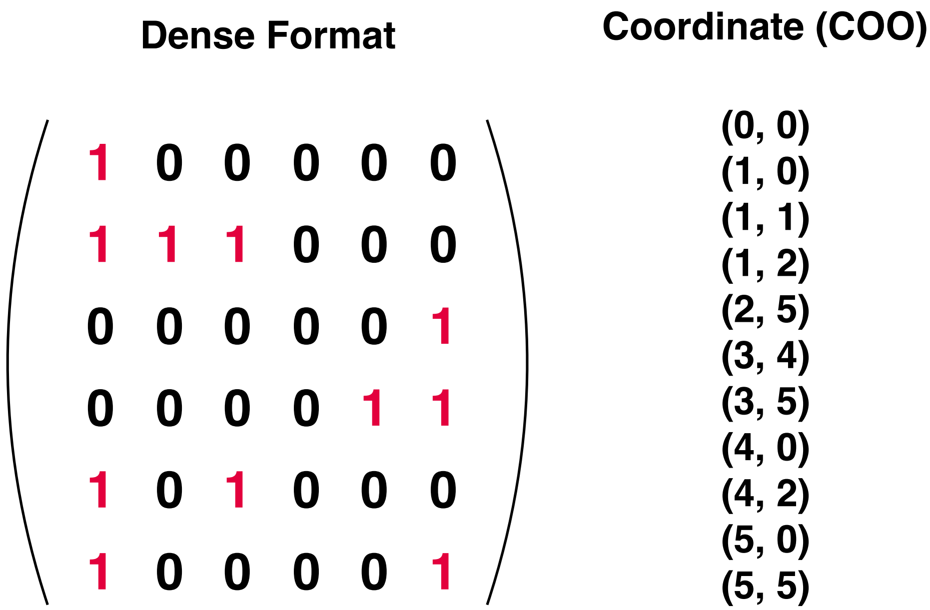


Figure 2-1: An example of a matrix (left) stored in coordinate (COO) format. COO stores the nonzeros in sorted order of their coordinates.

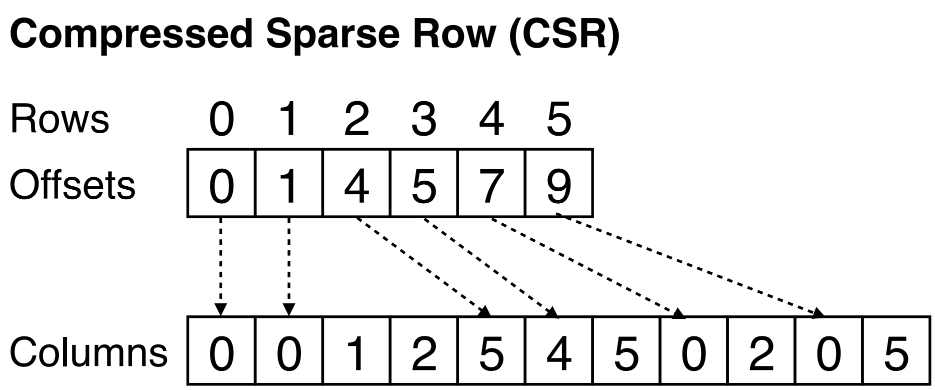


Figure 2-2: The same matrix from Figure 2-1 in CSR format. CSR stores a row array of offsets and a separate list of column indices.

nonzeros together and therefore decrease the complexity of storing the coordinates of individual nonzeros. Blocked storage formats can reduce the memory usage of sparse operations by reducing the complexity of locating nonzeros. Programmers and compilers can optimize linear algebra on small dense blocks using standard techniques such as loop unrolling, register and cache blocking, and instruction-level parallelism. The effectiveness of these optimizations depends heavily on the structure of the tensor and the blocked storage format [16, 21].

Proposed blocked storage formats are diverse, altering parameters such as the size and alignment of blocks, or the storage format for locations of blocks and nonzeros within blocks [16]. Some formats [22, 30] involve reordering to improve the block structure of the tensor (in this case, blocks may not represent contiguous entries in the original tensor).

Regular Blocking

In this thesis, we focus on "regular blocking" for simplicity. In ***regular blocking***, all nonzero blocks are aligned rectangular blocks of equal size. Each block represents contiguous entries in the original tensor. We formally define regular blocking in Definition 2.1.

We used a blocked extension of CSR called ***Blocked Compressed Sparse Rows (BCSR)*** [22] in our experiments. The locations of the nonzero blocks in BCSR are recorded using CSR format. Figure 2-3 shows an example of the same matrix from Figure 2-1 in BCSR format under different blocking schemes. The BCSR format generalizes naturally to ***Blocked Compressed Sparse Fiber (BCSF)*** format [18,25] for arbitrary tensors. In BCSR and BCSF, each block is stored in a dense format, with zeros represented explicitly, and only blocks which contain nonzeros are stored.

Definition 2.1 (Regular Blocking Scheme) *Let $\mathcal{A} \in \mathbb{F}^{I_1 \times I_2 \times \cdots \times I_R}$ be an R-tensor. A (regular) blocking scheme $\mathbf{b}$ of $\mathcal{A}$ is a vector $\mathbf{b} = (b_1, b_2, \ldots, b_R)$ that partitions $\mathcal{A}$ into R-dimensional aligned subtensors of equal size with b_r entries along the r^{th} dimension. Each component of $\mathbf{b}$ is a block size.*

Each coordinate of $\mathcal{A}$ has a corresponding ***block coordinate*** *under blocking scheme $\mathbf{b}$. Specifically, a nonzero at coordinate $\mathbf{i}$ has block coordinate*

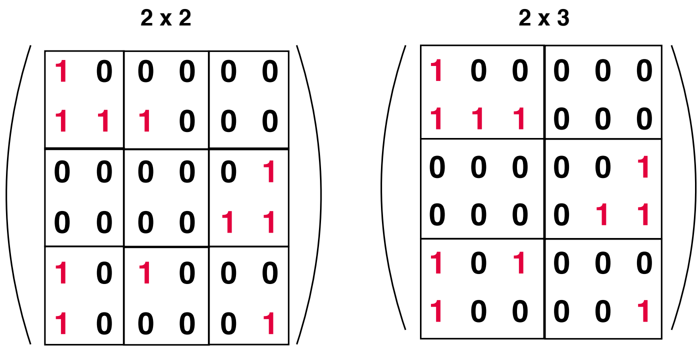
$$\left(\left\lceil \frac{i_1}{b_1} \right\rceil, \left\lceil \frac{i_2}{b_2} \right\rceil, \ldots, \left\lceil \frac{i_R}{b_R} \right\rceil \right).$$

Fill-estimation Problem

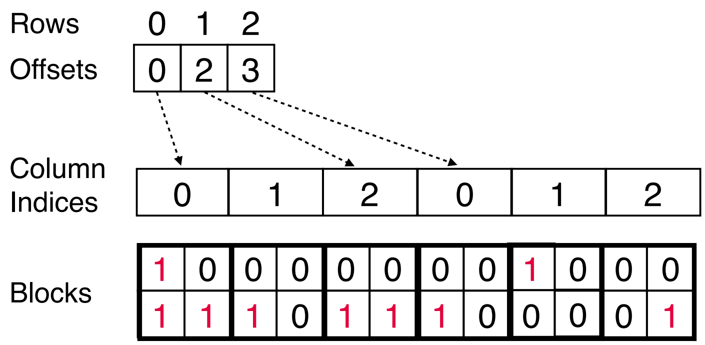
Since the performance of blocked sparse tensor operations depends on the blocking scheme and the structure of the tensor, our goal is to choose the blocking scheme that achieves the best performance for our given tensor. Larger blocks generally admit more opportunities for performance optimizations in blocked sparse formats with dense

Figure 2-3: Examples of different blockings on the same matrix from Figure 2-1 and their representation in blocked compressed sparse row (BCSR).

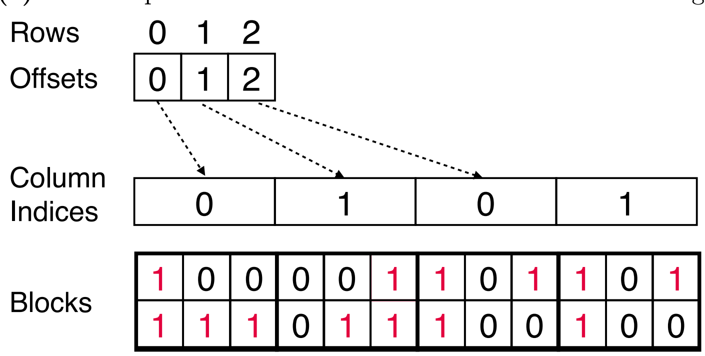
(a) Different blockings of the same matrix.



(b) BCSR representation of the matrix under a 2×2 blocking.



(c) BCSR representation of the matrix under a 2×3 blocking.



blocks. If the blocks do not capture the structure of the tensor, however, larger blocks hurt performance because they require computing over more explicitly represented (filled-in) zeros.

At a high level, a "good" blocking scheme includes all of the nonzero entries of a tensor in as few blocks as possible while minimizing the number of explicitly represented zeros.

Definition 2.2 *Supposed we have an R-tensor $\mathcal{A}$ and a regular blocking scheme* $\mathbf{b}$*.*

We define the number $k_{\mathbf{b}}((A))$ of blocks containing a nonzero under $\mathbf{b}$.

Notice that $k_{\mathbf{1}}(\mathcal{A}) = k(\mathcal{A})$, since tiling $\mathcal{A}$ into unit-size blocks will have exactly one non-empty block for every nonzero.

Specifically, a "good" blocking scheme $\mathbf{b}$ for a tensor $\mathcal{A}$ minimizes the number $k_{\mathbf{b}}(\mathcal{A})$ of nonempty blocks while also minimizing the number of introduced zeros.

We now formally define the ***fill*** as a metric which uses the number of nonzero blocks to formally express this notion of blocking scheme quality:

Definition 2.3 (Fill [14]) *The* fill *of an R-tensor $\mathcal{A}$ with respect to a particular blocking scheme $\mathbf{b}$ is the ratio*

$$f_{\mathbf{b}}(\mathcal{A}) = \frac{b_1 \times b_2 \times \cdots \times b_R \times k_{\mathbf{b}}(\mathcal{A})}{k(\mathcal{A})}\,.$$

That is, the fill is the ratio of the number of entries in nonempty blocks of $\mathcal{A}$ under $\mathbf{b}$ to the number $k(\mathcal{A})$ of nonzeros in $\mathcal{A}$. Where it is clear which tensor we refer to, we often write the fill as $f_{\mathbf{b}}$.

The fill $f_{\mathbf{b}}(\mathcal{A})$ is directly proportional to the number of nonzero blocks $k_{\mathbf{b}}(\mathcal{A})$.

Exact computation of the fill for many blocking schemes is costly in comparison to the cost of a sparse matrix-vector multiplication. Instead of exactly computing the fill, our problem is to compute an estimate of the fill.

Problem 2.4 (Fill Estimation) *Given an R-tensor $\mathcal{A}$ and a maximum block size B, the **fill-estimation problem** is the problem of computing an (ϵ, δ)-approximation $F_{\mathbf{b}}(\mathcal{A})$ to the true fill $f_{\mathbf{b}}(\mathcal{A})$ for all (square or rectangular) regular blocking schemes $\mathbf{b} \leq B$.*

Equivalently, we want to compute a random variable $F_{\mathbf{b}}(\mathcal{A})$ such that

$$\Pr\left[\max_{\mathbf{b} \leq B} \frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}} > \epsilon\right] \leq \delta\,.$$

Since $f_{\mathbf{b}}(\mathcal{A})$ differs from $k_{\mathbf{b}}(\mathcal{A})$ by a multiplicative factor of $b_1 b_2 \cdots b_R / k(\mathcal{A})$ (which can easily be computed in constant time), estimating the fill with respect to a blocking scheme is equivalent to estimating the number of nonzero blocks under that blocking scheme.

We will use these formal definitions of tensor notation and regular blocking to exactly define our PHIL algorithm in Chapter 3. Moreover, we show that PHIL solves the fill-estimation problem in Chapter 4.

Chapter 3

PHIL

In this chapter we describe the PHIL algorithm for fill estimation and detail its important subroutines. At a high level, PHIL randomly samples nonzeros. We first show that this random sampling results in an accurate estimate of the fill. Next, we explain how to efficiently estimate the fill over all block schemes for each sampled nonzero in a function called COMPUTE$\mathcal{X}$. evaluating the entire neighborhood of a sample We conclude by explaining a key step in processing each sample: finding all the nonzeros around a sample in time sublinear in the input size.

PHIL solves the fill-estimation problem by randomly sampling nonzero entries and counting the number of nonzero entries around each sampled nonzero. Suppose we want to estimate the fill of a sparse tensor $\mathcal{A}$ given a maximum block size B. PHIL repeatedly samples a coordinate $\mathbf{i}$ of a nonzero with replacement from $\mathcal{A}$. For each blocking scheme $\mathbf{b} \leq B$, it computes the number $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ of nonzero entries in the block that $\mathbf{i}$ appears in under the blocking scheme $\mathbf{b}$. Next, we show how PHIL uses $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ to estimate the fill.

Unbiased Estimation of the Fill

PHIL computes an accurate estimate of the fill by counting the number of nonzeros in each block for each sample. Let $\mathcal{A}$ be a tensor and $\mathbf{i}$ be a randomly chosen nonzero from $\mathcal{A}$. We define $F_{\mathbf{b}}$, a quantity proportional to the average of the reciprocals $1/z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$, and show that $F_{\mathbf{b}}$ is an ***unbiased estimator*** for the fill $f_{\mathbf{b}}$ (a random variable with expectation equal to the fill). We give a concentration bound for $F_{\mathbf{b}}$ in Theorem 3.1 and formally prove it in Theorem 4.2.

Theorem 3.1 (Maximum Number of Samples) *Suppose we want to estimate the fill* $f_{\mathbf{b}}$ *for all blocking schemes* $\mathbf{b} \leq B$ *where* B *is the maximum block size. If* PHIL *samples at least*

$$S \geq S_0 = \frac{B^{2R}}{2\epsilon^2} \ln\left(\frac{2B^R}{\delta}\right)$$

samples with replacement, then it produces a fill estimate $F_{\mathbf{b}}$ *over all blockings such that*

$$\Pr\left[\max_{\mathbf{b} \leq B} \frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}} \leq \epsilon\right] \geq 1 - \delta\,.$$

Notably, the number of samples PHIL requires to compute an (ϵ, δ)-approximation to the fill over all blocking schemes depends only on the maximum block size, desired accuracy, and failure probability. The required number of samples S_0 is independent of the input size, which is a clear advantage on large tensors where performance matters the most.

We describe how PHIL computes an unbiased estimator for the fill. First, we introduce the concept of the ***head*** and ***tail*** of a block because we will use it in later definitions.

Definition 3.2 (Head and Tail of Blocks) *The* ***head*** *of a block is the unique coordinate in the block with the lowest index along all dimensions. Let* $\mathbf{b}$ *be a regular blocking scheme and* $\mathbf{i}$ *be the coordinate in a tensor* $\mathcal{A}$*. We use* $h_{\mathbf{b}}(\mathbf{i})$ *to denote the head of* $\mathbf{i}$*'s block under the blocking scheme* $\mathbf{b}$*. Similarly, the* ***tail*** $t_{\mathbf{b}}(\mathbf{i})$ *of a block is the unique coordinate in the block containing* $\mathbf{i}$ *under* $\mathbf{b}$ *with the highest index along all dimensions.*

Next, we formally define the "fill component" of a nonempty block under some blocking. The ***fill component*** of a block is directly proportional to the number of nonzeros in that block. It is the reciprocal of the number of nonzeros in the block containing

Definition 3.3 *Suppose we want to estimate the fill of a tensor $\mathcal{A}$ under a blocking scheme $\mathbf{b}$. Let $\mathbf{i}$ be the coordinate of a nonzero of $\mathcal{A}$. The* ***fill component*** *is the reciprocal of the number of nonzeros in the block of $\mathcal{A}$ containing $\mathbf{i}$ under $\mathbf{b}$.*

Formally, the fill component $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ with respect to a nonzero $\mathbf{i}$ of $\mathcal{A}$ under a blocking $\mathbf{b}$ as

$$x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}) = \frac{1}{z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})} = \frac{1}{k(\mathcal{A}[h_{\mathbf{b}}(\mathbf{i}) : t_{\mathbf{b}}(\mathbf{i})])},$$

where $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ the number of nonzeros in the block of $\mathbf{i}$ under blocking scheme $\mathbf{b}$.

The number of nonzeros in a block is not directly proportional to the fill. The average of the fill component over all nonzeros, however, is exactly the number of nonempty blocks, which is proportional to the fill. PHIL therefore estimates the fill by averaging $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ over S coordinates $\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_S$ sampled with replacement from the set of coordinates of nonzeros in $\mathcal{A}$.

We show in Definition 3.4 that the fill estimate $F_{\mathbf{b}}$ is closely related to the average of $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ over all coordinates $\mathbf{i}$. We explain in Theorem 3.5 how the fill estimate $F_{\mathbf{b}}$ is an unbiased estimator of the fill.

Definition 3.4 (Fill Estimate) *For all $\mathbf{b} \leq B$:*

$$F_{\mathbf{b}} := \frac{b_1 b_2 \cdots b_R}{S} \sum_{j=1}^{S} x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_j)$$

Theorem 3.5 (Unbiased Estimator of the Fill) *For any blocking scheme $\mathbf{b}$, the random variable $F_{\mathbf{b}}$ is an unbiased estimator for the fill: that is, $\mathbb{E}[F_{\mathbf{b}}] = f_{\mathbf{b}}(\mathcal{A})$.*

Proof. By definition, the sum over all nonzeros $\mathbf{i}$ within a particular block of fill components $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ is 1 if the block is not empty. Thus, the sum of $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ over all nonzeros $\mathbf{i}$ in $\mathcal{A}$ is equal to $k_{\mathbf{b}}(\mathcal{A})$, the number of blocks that contain nonzeros. Thus, we may multiply the average of $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ over $\mathbf{i}$ by $b_1 b_2 \cdots b_R$ to obtain an estimator of $f_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$, by Definition 2.3. $\square$

EstimateFill

The remainder of this chapter provides details about how PHIL computes a fill estimate. Algorithm 3.6 shows the highest level of PHIL and abstracts away how to process samples into a subroutine called Compute$\mathcal{X}$. Algorithm 3.7 shows how to efficiently process each sample to compute the fill over all blocking schemes. Since Compute$\mathcal{X}$ requires finding all nonzeros in a range, we conclude by explaining how to quickly find nonzeros in a range.

Algorithm 3.6 *Given a sparse tensor* $\mathcal{A} \in \mathbb{F}^{I_1 \times I_2 \times \cdots \times I_R}$*,* $\mathbf{i}$*, and* B*, compute an approximation to* $f_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ *for all blocking schemes* $\mathbf{b} \leq B$*.*

Require:

$0 \leq \delta \leq 1\,, \quad \epsilon > 0\,, \quad B \geq 1$

function EstimateFill*($\mathcal{A}$, B, ϵ, δ)*

  $\mathcal{Y} \in \mathbb{R}^{B \times \cdots \times B}$

  $\mathcal{F} \in \mathbb{R}^{B \times \cdots \times B}$

  $S \leftarrow \left\lceil \frac{B^{2R}}{2\epsilon^2} \ln\left(\frac{2B^R}{\delta}\right) \right\rceil.$

  $\mathcal{Y} \leftarrow 0$

  for $\mathbf{i} \in$ *sample of size* S *with replacement from the nonzero coordinates of* $\mathcal{A}$ ***do***

    $\mathcal{Y} \leftarrow \mathcal{Y} + \text{Compute}\mathcal{X}(\mathcal{A}, B, \mathbf{i})$

  for $\mathbf{b} \in B \times \cdots \times B$ ***do***

    $\mathcal{F}[\mathbf{b}] \leftarrow \frac{b_1 b_2 \cdots b_R \mathcal{Y}[\mathbf{b}]}{s}$

  return $\mathcal{F}$

Ensure:

$(1 - \epsilon) f_{\mathbf{b}}(\mathcal{A}) \leq \mathcal{F}[\mathbf{b}] \leq (1 + \epsilon) f_{\mathbf{b}}(\mathcal{A})$ *with probability at least* $(1 - \delta)$*.*

Compute$\mathcal{X}$

PHIL estimates the fill efficiently over all blocking schemes using prefix sums in a routine called Compute$\mathcal{X}$. Let $\mathbf{i}$ be a nonzero that PHIL randomly sampled from an R-tensor $\mathcal{A}$. PHIL computes the number $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ of nonzeros in each block that $\mathbf{i}$ appears in for each blocking scheme $\mathbf{b} \leq B$. The first step of Compute$\mathcal{X}$ is to find the coordinates of all nonzeros near $\mathbf{i}$ in a routine called NONZEROSINRANGE. Once we find the coordinates of all nonzeros near $\mathbf{i}$, we use multidimensional prefix sums (cumulative sums) to compute $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ for all blocking schemes $\mathbf{b} \leq B$ in less than $(R+1)(2B)^R$ integer additions. Note that we expect both B and R to be small, and that we are compute B^R separate quantities simultaneously with this scheme.

We now describe how PHIL efficiently computes the number of nonzeros in all possible blockings around a sample $\mathbf{i}$ using prefix sums. A naive implementation of computing $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ for a sample coordinate $\mathbf{i}$ by might take time B^R in an R-tensor by looking up all the nonzeros in a block corresponding to $\mathbf{i}$. many nonzeros are in the block corresponding to $\mathbf{i}$ and In contrast, PHIL reuses the computations of $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ for the same $\mathbf{i}$ over different blocking schemes $\mathbf{b}$. Suppose PHIL samples a nonzero at coordinate $\mathbf{i}$. After finding the locations of all the nonzeros within a $2B$ radius of $\mathbf{i}$, PHIL computes $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ for all $\mathbf{b} \leq B$ at the same time.

We describe the details of this routine in Algorithm 3.7 and provide an example in Figure 3-1. We abstract the process of finding the nonzeros in a range of a tensor into a subroutine NONZEROSINRANGE and discuss potential efficient implementations after Algorithm 3.7.

The main idea behind COMPUTE$\mathcal{X}$ is to count the number of nonzeros in blocks containing a sampled nonzero over all blocking schemes. Specifically, COMPUTE$\mathcal{X}$ outputs a tensor $\mathcal{Z}_0$ corresponding to the number of nonzeros of an R-tensor $\mathcal{A}$ in subtensors surrounding a sampled nonzero $\mathbf{i} = (i_1, i_2, \ldots, i_R)$. Each entry of the tensor $\mathcal{Z}_0$ has the number of nonzeros in a corresponding blocking. We take the differences between relevant entries to find the number of nonzeros in all blockings around a sample $\mathbf{i}$. More formally, we construct an R-tensor $\mathcal{Z}_0 \in \mathbb{N}^{\mathbf{i}-B:\mathbf{i}+B-1}$ such that for all coordinates $\mathbf{j} = (j_1, j_1, \ldots, j_R)$ within a $2B$ radius of $\mathbf{i}$, $\mathcal{Z}_0[\mathbf{j}]$ is equal to the number of nonzeros in the subtensor $\mathcal{A}[\mathbf{i} - B : \mathbf{j}]$. In one dimension, we can compute $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ as $\mathcal{Z}_0[t_{\mathbf{b}}(\mathbf{i})] - \mathcal{Z}_0[h_{\mathbf{b}}(\mathbf{i}) - 1]$. In two dimensions, we can compute $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ as $\mathcal{Z}_0[t_{\mathbf{b}}(\mathbf{i})] - \mathcal{Z}_0[t_{b_1}(i_1), h_{b_2}(i_2) - 1] - \mathcal{Z}_0[h_{b_1}(i_1) - 1, t_{b_2}(i_2)] + \mathcal{Z}_0[h_{\mathbf{b}}(\mathbf{i}) - 1]$.

We briefly describe how to use prefix sums to efficiently construct $\mathcal{Z}_0$ over all blocking schemes. We initialize $\mathcal{Z}_0[\mathbf{j}]$ to 1 if $\mathcal{A}[\mathbf{j}] \neq 0$ and 0 otherwise. Next, we take a prefix sum along each dimension in turn. After the first prefix sum, $\mathcal{Z}_0[\mathbf{j}]$ is the number of nonzeros in $\mathcal{A}[i_1 - B : j_1, j_2, \ldots, j_R]$. After the r^{th} prefix sum, $\mathcal{Z}_0[\mathbf{j}]$ is the number of nonzeros in $\mathcal{A}[i_1 - B : j_1, \ldots, i_r - B : j_r, j_{r+1}, \ldots, j_R]$. After the R^{th} prefix sum (one along each dimension), we have computed $\mathcal{Z}_0$.

We find the number $z_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ of nonzeros in each block using differences between elements of $\mathcal{Z}_0$. Let $\mathbf{b} = (b_1, b_2, \ldots, b_R) \leq B$ be a blocking scheme. For each value of b_1, we set $\mathcal{Z}_1[j_2, \ldots, j_R]$ to the number of nonzeros in the subtensor $\mathcal{A}[h_{b_1}(i_1) : t_{b_1}(i_1), i_2 - B : j_2, \ldots, i_R - B : j_R]$ as $\mathcal{Z}_0[t_{b_1}(i_1), j_2, \ldots, j_R] - \mathcal{Z}_0[h_{b_1}(i_1) - 1, j_2, \ldots, j_R]$.

We now show how to generalize $\textsc{Compute}\mathcal{X}$ to arbitrary dimensions. After computing $\mathcal{Z}_1$ for a particular value of b_1, we take the difference between elements of $\mathcal{Z}_1$ for each value of b_2 to compute $\mathcal{Z}_2$, where $\mathcal{Z}_2[j_3, \ldots, j_R]$ is the number of nonzeros in the subtensor $\mathcal{A}[h_{b_1}(i_1) : t_{b_1}(i_1), h_{b_2}(i_2) : t_{b_2}(i_2), i_3 - B : j_3, \ldots, i_R - B : j_R]$. We do a similar computation for all R dimensions of the tensor until $\mathcal{Z}_R$ is just the scalar $z_{\mathbf{b}}(\mathcal{A}, \mathbf{j})$.

We conclude by analyzing how many operations we need to process each sample. PHIL takes prefix sums in each of the R dimensions where each prefix sum takes at most $(2B)^R$ additions to compute, and we compute R prefix sums. In the final loop, $\mathcal{Z}_r$ is of size $(2B)^{R-r}$. We must compute $\mathcal{Z}_r$ exactly B^r times. Therefore, the block difference computation incurs $\sum_{r=1}^{R} 2^{-r}(2B)^R$ subtractions. Thus, $\textsc{Compute}\mathcal{X}$ uses at most $(R+1)(2B)^R$ integer additions to compute $\mathcal{Z}$.

Algorithm 3.7 *Given a sparse tensor* $\mathcal{A} \in \mathbb{F}^{I_1 \times I_2 \times \cdots \times I_R}$, $\mathbf{i}$, *and* B, *compute* $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ *for all blocking schemes* $\mathbf{b} \leq B$. *Note that* $\mathcal{A}$ *may be stored in a sparse format, whereas all other tensors are stored in a dense format.*

Require:

$\mathcal{A}[\mathbf{i}] \neq 0\,, \quad B \geq 1$

function COMPUTE$\mathcal{X}$($\mathcal{A}$, $\mathbf{i}$, B)

  $\mathcal{Z}_0 \in \mathbb{N}^{\mathbf{i}-B:\mathbf{i}+B-1}$

  $\mathcal{Z}_0 \leftarrow 0$

  for $\mathbf{j} \in \text{NONZEROSINRANGE}(\mathcal{A}, \mathbf{i} - B, \mathbf{i} + B - 1)$ ***do***

    $\mathcal{Z}_0[\mathbf{j}] \leftarrow 1$

  for $r \in 1 : R$ ***do***

    for $j \in i_r - B + 1 : i_r + B - 1$ ***do***

      $\mathcal{Z}_0[\underbrace{:,\ldots,:,j}_{r},:,\ldots,:] \leftarrow \mathcal{Z}_0[\underbrace{:,\ldots,:,j}_{r},:,\ldots,:] + \mathcal{Z}_0[\underbrace{:,\ldots,:j-1}_{r},:,\ldots,:]$

  for $b_1 \in 1 : B$ ***do***

    $\mathcal{Z}_1 \leftarrow \mathcal{Z}_0[t_{b_1}(i_1),\underbrace{:,\ldots,:}_{r-1}] - \mathcal{Z}_0[h_{b_1}(i_1)-1,\underbrace{:,\ldots,:}_{r-1}]$

    for $b_2 \in 1 : B$ ***do***

      $\mathcal{Z}_2 \leftarrow \mathcal{Z}_1[t_{b_2}(i_2),\underbrace{:,\ldots,:}_{r-2}] - \mathcal{Z}_1[h_{b_2}(i_2)-1,\underbrace{:,\ldots,:}_{r-2}]$

      for $b_R \in 1 : B$ ***do***

        $\mathcal{Z}_R \leftarrow \mathcal{Z}_{R-1}[t_{b_R}(i_R)] - \mathcal{Z}_{R-1}[h_{b_R}(i_R)-1]$

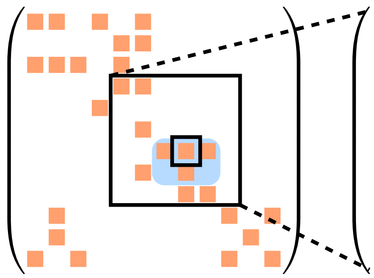
        $\mathcal{X}[\mathbf{b}] \leftarrow \frac{1}{\mathcal{Z}_R}$

Ensure:

$\mathcal{X}[\mathbf{b}] \leftarrow x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$

Figure 3-1: Here we visualize the execution of COMPUTE$\mathcal{X}$ as it computes one element of its output X. Specifically, we show how it computes $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}) = \mathcal{X}[\mathbf{b}]$. In this example, our maximum block size is $B = 3$ and our nonzero of interest is $\mathbf{i} = (7, 8)$. Continuing our example in Figure 1-2, we will show computation of $\mathcal{X}$ only for the blocking scheme $\mathbf{b} = (2, 3)$. Our goal is to compute the reciprocal of the number of nonzero elements in $\mathbf{i}$'s block (depicted by the shaded region).

(a) First, COMPUTE$\mathcal{X}$ uses NONZEROSINRANGE to find the nonzeros within a box of size $2B$ around $\mathbf{i}$. Then, it creates a matrix of the same size as the box and fills it with 0 where there are zeros in the original matrix and 1 where there are nonzeros.

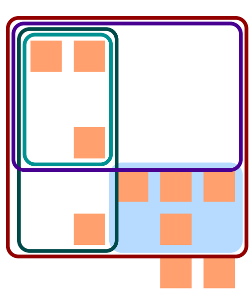


$$\begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

(b) Next, COMPUTE$\mathcal{X}$ performs a prefix sum on the rows and then columns of the matrix. Notice that element $\mathbf{j}$ of the matrix is now equal to the number of nonzero elements in the box extending from the upper left of the matrix to element $\mathbf{j}$.

$$\begin{pmatrix} 1 & 2 & 2 & 2 & 2 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 2 & 3 & 3 \\ 0 & 1 & 1 & 2 & 2 & 2 \\ 0 & 0 & 0 & 1 & 2 & 2 \end{pmatrix} \quad \begin{pmatrix} 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 & 3 & 3 \\ 1 & 3 & 4 & 5 & 6 & 6 \\ 1 & 4 & 5 & 7 & 8 & 8 \\ 1 & 4 & 5 & 8 & 10 & 10 \end{pmatrix}$$

(c) Finally, COMPUTE$\mathcal{X}$ computes the number of elements in the desired block by subtracting the number of nonzeros in each medium sized box from the large box, and adding back in the small box to avoid double-counting. Since all of these boxes begin in the upper left corner of our matrix, the number of nonzeros in these boxes are given by the prefix sum results in their lower right corners. The difference operation tells us that the shaded region contains $8 - 4 - 3 + 3 = 4$ nonzeros. Thus, $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}) = 1/4$. At this point, it is easy to compute $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ for different $\mathbf{b}$ by repeating the difference operation with different blocks.



$$\begin{matrix} 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 3 & 3 & 3 & 3 & 3 \\ 1 & 3 & 4 & 5 & 6 & 6 \\ 1 & 4 & 5 & 7 & 8 & 8 \\ 1 & 4 & 5 & 8 & 10 & 10 \end{matrix}$$

NonzerosInRange

Since $\mathcal{A}$ may be stored in an arbitrary sparse format, we abstract the process of finding the coordinates of nonzeros within a certain range into an algorithm called NonzerosInRange. NonzerosInRange($\mathcal{A}, \mathbf{j}, \mathbf{j}'$) returns a list of all $\mathbf{i} \in \mathbf{j} : \mathbf{j}'$ such that $\mathcal{A}[\mathbf{i}] \neq 0$.

The implementation of NonzerosInRange depends on the initial format of the sparse matrix $\mathcal{A}$. We discuss two implementations to show why this routine should not be costly in theory or practice.

If $\mathcal{A}$ is a matrix in CSR format (where coordinates of nonzeros in each row are stored in sorted order of their column index), we do not need any preprocessing to quickly query nonzeros. Specifically, using a binary search within each row yields an $O(B \log_2(I_2) + B^2)$ time implementation, where the B^2 term is the maximum number of coordinates that may need to be returned. This search technique generalizes to arbitrary tensors in CSF format, yielding an $O\left(\sum_{r=2}^{R} B^{r-1} \log_2(I_r) + B^R\right)$ time implementation.

If $\mathcal{A}$ is stored in any other format (e.g. COO), we can preprocess the tensor such that we can query for nonzeros in a range in time independent of the input size. Before we run EstimateFill, we block the entire R-tensor $\mathcal{A}$ into blocks of size B^R (i.e. with blocking $\mathbf{b} = (B, B, \ldots, B)$). and store the blocks in a sparse format (without explicit zeros). We store each block that contains at least one nonzero in a hash table. Since PHIL only calls NonzerosInRange with ranges of size $2B \times \cdots \times 2B$, there are at most 3^R blocks which might contain zeros in the target range. To find all nonzeros in a range, we scan through these blocks to find nonzeros which are actually in the target range, and return the relevant nonzeros. This implementation of NonzerosInRange has a setup time of $O(k(\mathcal{A}))$ and an individual query time of $O(3^R B^R)$. After preprocessing, the time to complete query of NonzerosInRange is independent of the size of the input.

Chapter 4

Theoretical Analysis

This chapter proves that PHIL produces an accurate estimate of the fill with a number of samples independent of the input size. We now show concentration bounds on the accuracy of PHIL's estimate using Hoeffding's inequality [12]. The number S of samples required for an accurate estimate only depends on the desired accuracy and probability of that accuracy. Notably, S is constant with respect to the input size, which is especially advantageous when $S \ll k(\mathcal{A})$. Finally, we propose solutions in case the number of required samples exceeds the number of nonzeros in a tensor, which may occur if the tensor or matrix is small.

Concentration Bounds on PHIL*'s Error*

Theorem 4.1 (Hoeffding's inequality) *Let $X_1, X_2, \ldots, X_M$ be M independent random variables bounded such that $0 \leq X_j \leq 1$. Let $\overline{X} = \frac{1}{M}\sum_{j=1}^{M} X_j$ be their mean. Then for any $t \geq 0$,*

$$\Pr\left[\left|\overline{X} - \mathbb{E}[X]\right| \geq t\right] \leq 2\exp(-2Mt^2)\,.$$

We can directly apply Hoeffding's inequality to PHIL's estimate to bound the error given the number of samples. Given a sparse tensor $\mathcal{A}$, a blocking scheme $\mathbf{b}$, and a tensor element $\mathbf{i}$, the fill component $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ is a random variable bounded between 0 and 1. Furthermore, since the samples $\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_S$ are chosen independently from among the nonzeros, the random variables $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_1), x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_2), \ldots, x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_S)$ are independent. Therefore, we obtain our concentration bound from Theorem 4.1.

Theorem 4.2 (Restatement of Theorem 3.1) *Suppose we want to estimate the fill $f_{\mathbf{b}}$ for all blocking schemes $\mathbf{b} \leq B$ where B is the maximum block size. If* PHIL *samples at least*

$$S \geq S_0 = \frac{B^{2R}}{2\epsilon^2} \ln\left(\frac{2B^R}{\delta}\right)$$

samples with replacement, then it produces a fill estimate $F_{\mathbf{b}}$ over all blockings such that

$$\Pr\left[\max_{\mathbf{b} \leq B} \frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}} \leq \epsilon\right] \geq 1 - \delta\,.$$

PROOF. By Definition 3.4, $F_{\mathbf{b}} = b_1 b_2 \cdots b_R (1/S) \sum_{j=1}^{S} x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_j)$ by definition. By Theorem 3.5, $\mathbb{E}[F_{\mathbf{b}}] = f_{\mathbf{b}}$. Since each examined block contains at least 1 and at most B^R nonzeros, $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_1), x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_2), \ldots, x_{\mathbf{b}}(\mathcal{A}, \mathbf{i}_S)$ are independent and bounded between $1/B^R$ and 1. Similarly, $k_b(\mathcal{A})/k(\mathcal{A})$ in Definition 2.3 is bounded to the same range. By Theorem 4.1,

$$\Pr\left[\frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}} \geq \epsilon\right] = \Pr\left[\left|\frac{F_{\mathbf{b}} - \mathbb{E}[F_{\mathbf{b}}]}{b_1 b_2 \cdots b_R}\right| \geq \epsilon \frac{f_{\mathbf{b}}}{b_1 b_2 \cdots b_R}\right]$$
$$\leq 2\exp\left(-2S\left(\frac{\epsilon k_b(\mathcal{A})}{k(\mathcal{A})}\right)^2\right) \leq 2\exp\left(\frac{-2S\epsilon^2}{B^{2R}}\right),$$

since $F_{\mathbf{b}}$ is $b_1 b_2 \cdots b_R$ times an average of S values, each of which is at least $1/B^R$. By the union bound over the B^R possible blocking schemes $\mathbf{b}$,

$$\Pr\left[\max_{\mathbf{b} \leq B} \frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}} \geq \epsilon\right] \leq 2B^R \exp\left(\frac{-2S\epsilon^2}{B^{2R}}\right).$$

Therefore, if $S \geq S_0 = \frac{B^{2R}}{2\epsilon^2} \ln\left(\frac{2B^R}{\delta}\right)$,

$$\Pr\left[\max_{\mathbf{b} \leq \mathbf{B}} \frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}} \geq \epsilon\right] \leq \delta\,.$$

□

The bound S on the number of samples PHIL needs to compute an (ϵ, δ)-

approximation to the true fill is dependent only on the maximum block size, the order of the input tensor, and the desired approximation accuracy. Let $\mathcal{A}$ be an R-tensor. PHIL requires a number of samples that is only only dependent on B, R, ϵ, and δ. If ϵ and δ are independent of the number $k(\mathcal{A})$ of nonzeros, the bound S on the number of samples is also constant with respect to $k(\mathcal{A})$. Sampling is therefore especially advantageous when $S \ll k(\mathcal{A})$.

Obtaining a high probability bound with $\delta \leq 1/k(\mathcal{A})^w$ for some w would indeed require dependence on $k(\mathcal{A})$, albeit only logarithmically. In practice, however, a small constant δ such as 0.01 suffices.

Sampling for High Accuracy or Small Tensors

PHIL may require more samples than the number of nonzeros in a small or very sparse tensor if one requests strong guarantees on its fill estimate. For example, a run of PHIL on a matrix ($R = 2$) may set the parameters $B = 12$, $\epsilon = 0.1$ and $\delta = 0.01$. The number of required samples (10,645,998) may exceed the number of nonzeros in smaller matrices.

We can avoid this issue by sampling without replacement. If we sample without replacement, we can apply a variant of the Hoeffding-Serfling inequality [3] to obtain a bound which scales with the number of nonzeros. This bound is more complicated to describe, and requires the implementation to generate samples without replacement. Furthermore, this bound would still require sampling a significant fraction of the nonzeros.

Instead, we suggest that practitioners who need strong guarantees on small problems use an efficient exact algorithm or lower the maximum block size B. In our example, $B = 4$ needs only 103,308 samples. We show in Chapter 5 that PHIL empirically provides far more accurate estimates than the worst-case guaranteed theoretical bound. In practice, for $B = 12$, running PHIL with $\epsilon = 3$ and $\delta = 0.01$ (11,829 samples) results in a mean maximum relative error of at most 0.05 for all cases we tested.

Chapter 5

Experimental Results

We tested PHIL and OSKI on a large suite of sparse matrices and found that PHIL estimates the fill more accurately in much less OSKI for many of the matrices in our test suite. There were no cases in PHIL was both less accurate and slower than OSKI.

Since OSKI lacks theoretical guarantees on its accuracy, we generated a pathological input matrix where OSKI produces useless fill estimates whereas PHIL produces accurate estimates. PHIL computes a provably accurate estimate of the fill for all inputs (as shown in Chapter 5). We also generate a worst-case input for PHIL and show in Figure 5-1 that PHIL still produces a more accurate estimate than OSKI on this input.

We also found that when using optimized BCSR matrix-vector multiplication routines generated by the Tensor Algebra Compiler (TACO) [18] and the SPARSITY performance model (described in Chapter 1), the estimates produced by PHIL yield BCSR matrix-vector multiply performance comparable to the performance obtained using estimates from OSKI.

We also chose a few matrices and ran PHIL and OSKI with multiple parameter settings on those matrices. Different parameter settings correspond to different runtimes. For example, the runtime of PHIL increases as ϵ and δ decrease. Figure 5-1 shows that the return on (time) investment for PHIL is better than OSKI on four matrices, including on synthetic matrices designed to bring out the worst in our PHIL algorithm.

Pathological Inputs for PHIL *and* OSKI

We describe two pathological cases we invented to induce worst-case behavior in PHIL and OSKI, respectively. We generated these pathological matrices and call them `pathological_PHIL` and `pathological_OSKI`, respectively. We will show that `pathological_PHIL` is indeed a worst-case input for PHIL.

Definition 5.1 (Pathological PHIL Matrices) *Pathological* PHIL *matrices are worst-case inputs for* PHIL. *These matrices have an equal number of completely full blocks and blocks with only one nonzero.*

We first try to provide some intuition about why pathological PHIL matrices are the worst-case inputs for PHIL. At a high level, pathological PHIL matrices maximize the variance of the PHIL estimator $F_{\mathbf{b}}(\mathcal{A})$. Let $\mathcal{A}$ be a worst-case tensor for a blocking scheme $\mathbf{b}$. Assume for contradiction that there are nonzero blocks which are not completely full and contain more than one nonzero. We can add nonzeros to more than half full blocks and remove nonzeros from more than half empty blocks to increase the *variance* of each of each fill component $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$. This reassignment increases the variance of the PHIL estimator $F_{\mathbf{b}}(\mathcal{A})$, which increases the probability that it will deviate farther from its mean. Thus, our worst case matrix has only completely full blocks and blocks with only one nonzero.

We formalize this intuition that the variance of the fill estimate $F_{\mathbf{b}}$ is maximized if full blocks and blocks with only one nonzero occur in equal number by showing that such matrices are maximally likely to cause a deviation between the true fill $f_{\mathbf{B}}$ and the PHIL estimator $F_{\mathbf{b}}$.

Theorem 5.2 *Consider a matrix $\mathcal{M}$ with an even number T of nonzero blocks under a particular blocking scheme $\mathbf{b}$, such that precisely $T/2$ of the nonzero blocks are completed filled with nonzeros and $T/2$ of the nonzero blocks contain only one nonzero.*

Then for any $\epsilon > 0$ and matrix $\mathcal{M}'$ with T nonzero blocks under blocking scheme $\mathbf{b}$,

$$\begin{aligned}&\Pr\left[|f_{\mathbf{b}}(\mathcal{M}') - F_{\mathbf{b}}(\mathcal{M}')|/f_{\mathbf{b}}(\mathcal{M}') > \epsilon\right] \\ &\leq \Pr\left[|f_{\mathbf{b}}(\mathcal{M}) - F_{\mathbf{b}}(\mathcal{M})|/f_{\mathbf{b}}(\mathcal{M}) > \epsilon\right]\end{aligned}$$

PROOF. Given a matrix $\mathcal{M}'$ with T nonzero blocks, exactly one of the following statements must hold:

1. Every block in $\mathcal{M}'$ is either completely filled with nonzeros, or contains a single nonzero.

2. There are some blocks S that are not completely filled but contain more than one nonzero.

For any matrix for which (2) holds, we may pick a block in S and add a nonzero to it (if it more than half full) or remove a nonzero from it (if it is more than half empty). This increases the *variance* of each of each value $x_{\mathbf{b}}(\mathcal{M}', \mathbf{i})$, and therefore also increases the variance of the PHIL estimator $F_{\mathbf{b}}(\mathcal{M}')$. Increasing the variance increases the probability $\Pr\left[|f_{\mathbf{b}}(\mathcal{M}') - F_{\mathbf{b}}(\mathcal{M}')|/f_{\mathbf{b}}(\mathcal{M}') > \epsilon\right]$. By induction on the number of applications of this procedure, there exists a matrix $\mathcal{A}$ where every block is either completely filled or contains a single nonzero such that $\mathcal{A}$ has a higher failure probability (i.e. is "more pathological") than $\mathcal{M}'$.

Suppose that $\mathcal{A}$ has pT blocks filled completely with ℓ nonzeros and $(1-p)T$ blocks containing a single nonzero, for some $0 \leq p \leq 1$. Therefore, every $x_{\mathbf{b}}(\mathcal{A}, \mathbf{i})$ is either $1/\ell$ or 1, in the case where $\mathbf{i}$ is in a completely filled block or a nearly-empty block, respectively. The variance of the PHIL estimator $F_{\mathbf{b}}(\mathcal{A})$ is given by $p(1-p)/\ell$, which is maximized when $p = 1/2$. Thus, $\Pr\left[|f_{\mathbf{b}}(\mathcal{A}) - F_{\mathbf{b}}(\mathcal{A})|/f_{\mathbf{b}}(\mathcal{A}) > \epsilon\right]$ is maximized when $\mathcal{A}$ is $\mathcal{M}$. □

For our concrete test case, we create a $10,000 \times 10,000$ matrix called `pathological_PHIL` with $10,000$ full 12×12 blocks and $10,000$ sparse 12×12 blocks. PHIL should perform poorly on this matrix.

We also devised an empirically pathological matrix called `pathological_OSKI` to bring out the worst in the OSKI algorithm. Since OSKI samples rows with equal probability, hiding many blocks which look different from the rest of the matrix in a single row should cause OSKI to perform poorly. We tested PHIL and OSKI on a `pathological_OSKI` matrix of size $100,000 \times 100,000$ where the first 6 rows are dense, while all other rows have only a single nonzero in the first column.

Evaluation Metrics

Since program autotuning algorithms typically run at runtime before execution of the tuned operation, the speedups gained by autotuning must be weighed against the execution time of the algorithm. Because we tested an example of autotuning blocked SpMV, we normalize the time OSKI and PHIL take to estimate the fill by the duration of an unblocked parallel CSR SpMV.

We use the SPARSITY performance model to select a blocking scheme. Since the estimated performance is proportional to the fill, we judge the quality of a fill estimate using the maximum relative error.

Definition 5.3 *The maximum relative error of a fill estimate f over all blockings $\mathbf{b} \leq B$ is*

$$\max_{\mathbf{b} \leq B} \frac{|f_{\mathbf{b}} - F_{\mathbf{b}}|}{f_{\mathbf{b}}}.$$

Note that a maximum relative error is greater than 1 represents a complete loss of accuracy, as a bogus algorithm that returns 0 for the estimated fill of all blocking schemes would achieve a better maximum relative error.

Empirical Study with Fixed Parameters

We tested PHIL and OSKI on almost all of the matrices with more than one million nonzeros from the sparse matrix collection using the default recommended settings of both algorithms. All but two are from the University of Florida Sparse Matrix Collection (Suitesparse) [10]. These matrices were chosen to represent a variety of application domains and block structures.

Appendix A contains all of the results from our comparison of PHIL and OSKI with fixed parameters. The default parameters to PHIL are $\epsilon = 3$ and $\delta = 0.01$ when

$B = 12$, and they are $\epsilon = 0.25$ and $\delta = 0.01$ when $B = 4$. The parameters to OSKI are $\sigma = 0.02$ (the recommended setting) for all cases.

These extensive experiments show that for a fixed setting of parameters, the runtime and relative error of our fill estimation algorithms varies substantially from matrix to matrix (although the relative error of PHIL is consistently small).

We compare PHIL and OSKI with fixed settings in terms of runtime, mean maximum relative error, and the resulting BCSR SpMV time. Figure 5-2 shows an example of our with study with fixed parameters on our two synthetic matrices. Our results show that that in most cases, PHIL was more accurate and much faster than OSKI. PHIL always produced results with a mean maximum relative error less than .05, while in a few cases OSKI produced results with a mean maximum relative error which was worse or much worse than 1. Figure A-1 provides a list of tables of results for matrices from the Sparse Matrix Collection. Finally, we test PHIL and OSKI on the synthetic pathological matrices and report our findings in Figure 5-2.

Since PHIL uses a fixed number of samples, PHIL's normalized runtime appears higher for small matrices because PHIL takes longer relative to the parallel CSR matrix-vector multiplication time on smaller matrices. On larger matrices (when autotuning is most important), however, PHIL usually takes at most 10 matrix-vector multiplies, outperforming OSKI by factors of 10 to 40.

Both the PHIL and OSKI estimates led to remarkably similar BCSR matrix-vector multiplication times. It may be possible to improve the chosen blocking schemes with a more complex performance model [7], but our focus is on estimating the fill and not on modeling the performance of sparse kernels.

Accuracy Return on Time Investment

Since running both algorithms under fixed settings is only one way to execute PHIL and OSKI, we compared the algorithms using a range of parameters on a selection of matrices in Figure 5-1. Figure 5-1 shows the mean maximum relative error as a function of the runtime of the estimation algorithm on four different matrices.

We chose four matrices as a representative sample of inputs. We compared PHIL and OSKI on the matrices `ct20stif` and `gupta1` from Suitesparse because Vuduc *et al.* [26] used them to measure OSKI. We also tested PHIL and OSKI on our pathological inputs.

We found that PHIL provides better estimates of the fill than OSKI for any amount of time invested. On these four matrices, PHIL is both more efficient and more accurate than OSKI. On `pathological_PHIL`, PHIL performs better than OSKI, but the performance difference is smaller than the difference between PHIL and OSKI on `ct20stif` and `gupta1`. On `pathological_OSKI`, OSKI fails to estimate the fill in any reasonable time.

Experimental Setup

We now explain how we generated our empirical results. We implemented[1] both PHIL and OSKI for sparse matrices in CSR format in `C`, which can efficiently execute the dense integer and floating point operations in $\text{COMPUTE}\mathcal{X}$ (Algorithm 3.7). Finally, both implementations run serially and use the `mt19937` random number generator from the `C++` Standard Library.

We also parallelized[2] PHIL using Cilk [5] and compiled our code with Tapir [23].

We chose blocking schemes to maximize estimated performance of blocked SpMV according to the SPARSITY performance model. To create the performance matrix PERF for the SPARSITY performance model, we timed BCSR matrix-vector multiplication performance for 100 trials on a 1000×1000 dense matrix. We chose We used TACO to generate parallel BCSR kernels for each blocking scheme, which we ran on one socket with 12 threads.

We ran all of our experiments on a node with two sockets, each with a 12-core Intel® Xeon™ Processor E5-2695 v3 "Ivy Bridge" at 2.4 GHz. Each core has 32 KB of L1 cache and 256 KB of L2 cache. Each socket has 30 MB of shared L3 cache.

[1]Our serial code is available under the BSD 3-clause license at `https://github.com/peterahrens/FillEstimation/releases/tag/IPDPS2018`.

[2]Our parallel code will be available in the full version.

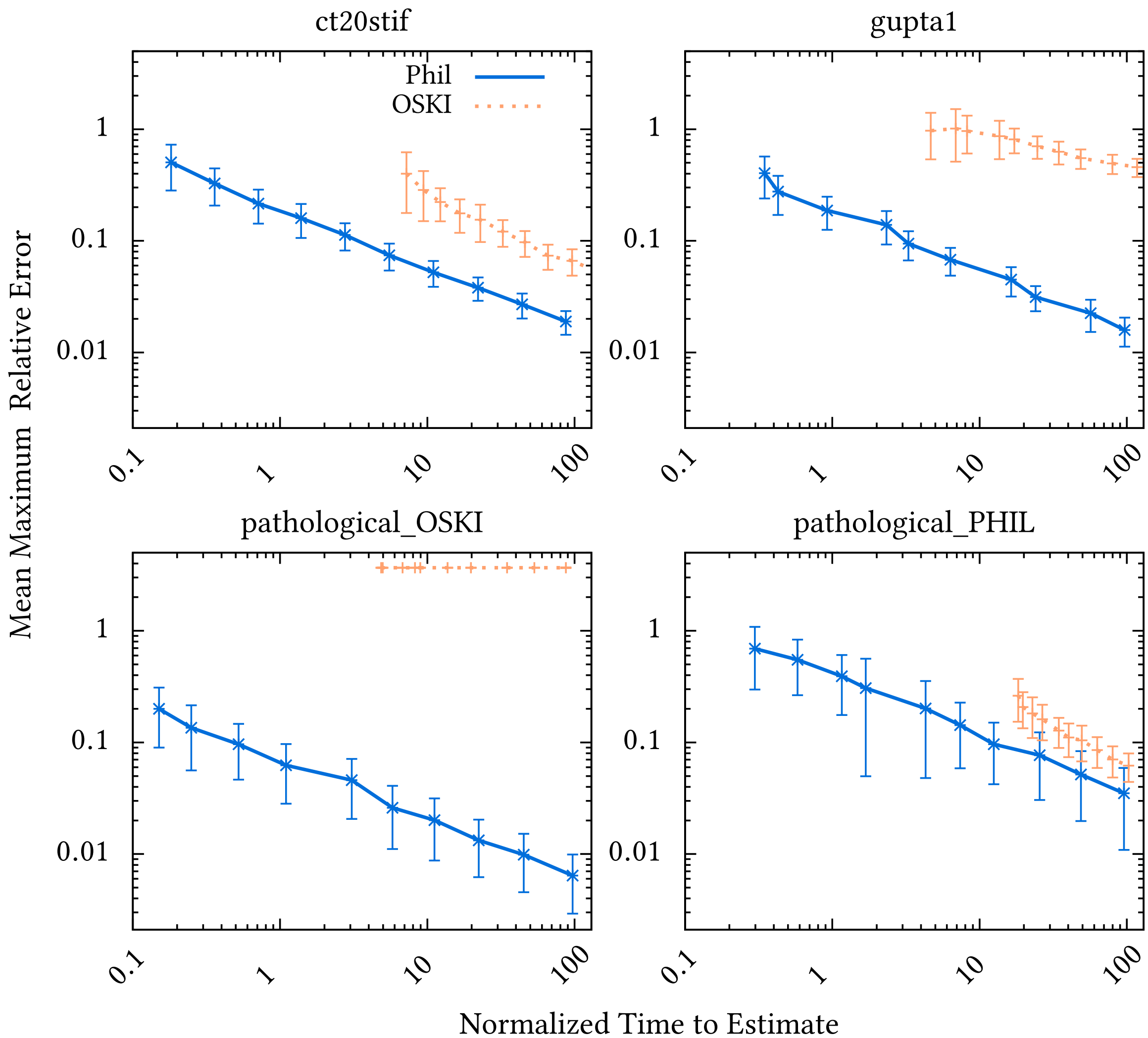


Figure 5-1: Mean maximum relative error (Definition 5.3) as a function of mean estimation time (normalized to the mean time it takes to perform a parallel sparse matrix-vector multiplication in CSR format using TACO [18]) for four matrices. Both axes use logarithmic scale. All means are the average of 100 trials. The error bars reflect one standard deviation above and below the mean. The blue solid line represents PHIL and the orange dotted line represents OSKI. Each point is a separate setting for the parameters. `ct20stif` is the stiffness matrix arising from the application of finite element methods to a structural problem with some block structure. `gupta1` is the matrix representation of a linear programming problem, and has no obvious block structure. The pathological matrices are described in more detail in Chapter 5. Note that errors above 1 represent a complete loss of accuracy.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: Synthetic														
pathological_PHIL	72,356	23,989	695.7	177.4	0.046	0.383	1.0*	1.0*	2.769	90.79	0.092	0.037	1.0*	1.0*
pathological_OSKI	69,994	20,000	164.0	33.30	0.012	3.666	0.635	0.635	0.793	17.05	0.060	1.800	0.713	0.809

Figure 5-2: On our synthetic matrices, we show the mean estimation time, mean maximum relative error (Definition 5.3), and the resulting mean parallel sparse matrix-vector multiply (SpMV) time in BCSR format with the optimal blocking scheme according to the SPARSITY performance model. Times are normalized to the mean time taken to perform one parallel sparse matrix-vector multiply (SpMV) on the unblocked CSR matrix. All means are the average of 100 trials. All blocked and non-blocked matrix-vector multiplies are performed using TACO. Highlighted cells show the better result between PHIL and OSKI. The left group of columns corresponds to a maximum block size $B = 12$. The right group of columns corresponds to a maximum block size of $B = 4$. * Results with an asterisk are cases where a slowdown was observed when the performance model was used with the given estimates. Since most autotuners will try both an unblocked CSR format and the predicted best blocking scheme with BCSR format, they may choose to use CSR if no speedup is observed and so these results are listed as 1.0.

Chapter 6

Conclusion

We presented PHIL, the first fill-estimation algorithm with provable guarantees. PHIL computes an (ϵ, δ)-approximation to the fill and requires a number of samples independent of the input size.

We also showed empirically that PHIL estimates the fill of a sparse matrix at least 2 times faster than OSKI on most of our real-world inputs and provides useful estimates of the fill even in pathological test cases. PHIL and OSKI produced comparable speedups in blocked sparse matrix-vector multiply in most cases using their recommended parameters. PHIL produced far more accurate estimates of the fill than its worst-case accuracy guarantee.

Sampling techniques are useful in program autotuning since we can often sacrifice some accuracy in the heuristics for a faster autotuner. As libraries for numerical computation evolve and autotuning moves from compile-time to run-time implementations, developers will need efficient heuristics [11]. PHIL's empirical success suggests broader potential for sampling techniques in the design of autotuned numerical software. Faster sampling algorithms with provable guarantees will allow library developers to write software that can more accurately specialize to user data and provide the best possible performance for their application and hardware.

Future Work

Future work includes an optimized, vectorized implementation of PHIL and an extension to handle sparse tensors in multiple storage formats. COMPUTE$\mathcal{X}$ should benefit from instruction-level parallelism. One of our goals in the design of PHIL was to express the fill-estimation problem as a dense set of operations that can be computed efficiently.

We found that the blocked SpMV times due to blocking schemes chosen according to the SPARSITY performance model were similar for both PHIL and OSKI. Perhaps a more complex performance model [7] would lead to different choices of blocking schemes and therefore different blocked SpMV performance.

Coarse Fill Estimation

Some blocked formats [6, 30] store their blocks in a sparse format. These blocks are usually much larger than the blocks we considered in this thesis, but we can extend any algorithm (e.g. PHIL) for Problem 2.4 to estimate the fill of larger blocks by limiting our attention to multiples of some base block size.

Problem 6.1 (Coarse Fill Estimation) *Given a tensor $\mathcal{A} \in \mathbb{F}^{I_1 \times I_2 \times \cdots \times I_R}$, a base block size $\mathbf{q}$, and a maximum multiplier B, compute an approximation $F_{\mathbf{b}}(\mathcal{A})$ accurate to within a factor of ϵ for all $\mathbf{b}$ where $b_r = b'_r q_r$ and $1 \leq \mathbf{b}' \leq B$ with probability $1 - \delta$.*

Let $\mathcal{A}' \in \mathbb{F}^{I'_1 \times I'_2 \times \cdots \times I'_R}$ be a tensor. We first set $\mathcal{A}'[\mathbf{j}]$ to the number of nonzeros in block $\mathbf{j}$ of $\mathcal{A}$ under the blocking scheme $\mathbf{q}$. Notice that $f_{\mathbf{b}'}(\mathcal{A}') = f_{\mathbf{b}}(\mathcal{A})$, so a solution to Problem 2.4 on $\mathcal{A}'$ is a solution to Problem 6.1 on $\mathcal{A}$. Since $k(\mathcal{A}') \leq k(\mathcal{A})$, $\mathbf{I}' \leq \mathbf{I}$, and we can construct $\mathcal{A}'$ in $O(k(\mathcal{A}))$ time, most algorithms (including PHIL) that solve Problem 2.4 can solve Problem 6.1 with an addition of $O(k(\mathcal{A}))$ to their asymptotic running time.

Appendix A

Empirical Study

We tested PHIL and OSKI on almost all of the matrices with more than one million nonzeros from the sparse matrix collection using the default recommended settings. We report the normalized mean fill estimation time, mean maximum relative error, and resulting mean parallel sparse matrix-vector multiply (SpMV) time. We provide further details about the experimental setup in Figure A-2. Our results are organized as follows:

Figures	Number of nonzeros in matrices (in millions)
Figures A-2 and A-3	[1, 1.5)
Figures A-4 and A-5	[1.5, 2)
Figure A-6	[2, 2.5)
Figure A-7	[2.5, 3)
Figure A-8	[3, 4)
Figure A-9	[4, 5)
Figure A-10	[5, 7)
Figure A-11	[7, 10)
Figure A-12	[10, 17)
Figure A-13	[17, 35)
Figure A-14	[35-100)
Figures A-15 and A-16	[1, 1.5) (Serial vs. Parallel PHIL)

Figure A-1: Guide to figures for experiments on the Suitesparse matrix collection. Each figure shows results for matrices with number of nonzeros in the given range. All results are for serial implementations of PHIL and OSKI unless specified otherwise.

Matrix Information			B = 12						B = 4					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
heart1	1,387,773	7,114	86.16	82.61	0.020	0.252	0.794	0.816	85.46	85.25	0.020	0.253	0.852	0.873
torso2	1,033,473	231,934	79.64	182.4	0.033	0.040	1.0*	1.0*	79.23	181.9	0.031	0.039	1.0*	1.0*
Dubcova2	1,030,225	130,050	80.57	142.7	0.020	0.074	1.000	1.000	80.30	142.9	0.019	0.064	1.0*	1.0*
Domain: Chemical Process Simulation														
lhr71	1,528,092	140,608	76.66	161.7	0.028	0.085	1.0*	1.0*	77.61	162.1	0.030	0.090	1.0*	1.0*
std1_Jac3	1,455,848	43,964	61.52	70.33	0.030	0.411	1.0*	0.954	61.38	71.29	0.028	0.404	0.985	0.972
std1_Jac2	1,248,731	43,964	60.48	63.82	0.028	0.335	0.833	0.810	60.72	64.00	0.029	0.347	0.761	0.773
Domain: Circuit Simulation														
ASIC_320ks	1,827,807	643,342	30.95	165.7	0.020	0.090	1.000	1.0*	30.43	175.9	0.018	0.088	1.0*	1.0*
Raj1	1,302,464	527,486	55.88	260.0	0.019	0.192	1.0*	1.0*	56.44	262.7	0.018	0.199	1.0*	1.0*
Domain: Combinatorial Problem														
n4c6-b10	1,456,422	318,960	56.64	188.5	0.018	0.015	1.000	1.000	56.28	189.1	0.018	0.015	0.945	0.945
relat8	1,334,038	358,035	61.50	333.9	0.010	0.020	1.000	1.000	61.38	331.0	0.009	0.019	1.0*	1.0*
n4c6-b7	1,305,720	267,330	57.21	200.8	0.017	0.013	1.000	1.000	57.97	201.0	0.019	0.013	1.0*	1.0*
IG5-17	1,035,008	58,106	98.17	121.1	0.012	0.071	1.0*	1.0*	98.74	120.4	0.012	0.073	0.987	0.987
Domain: Computational Fluid Dynamics Problem														
raefsky3	1,488,768	42,400	89.98	119.4	0.024	0.031	0.598	0.598	89.91	119.6	0.023	0.033	0.625	0.625
ex11	1,096,948	33,228	106.9	107.7	0.031	0.062	1.0*	1.0*	107.2	108.5	0.032	0.063	1.0*	1.0*
rim	1,014,951	45,120	120.8	124.4	0.022	0.072	1.0*	1.0*	120.7	125.4	0.021	0.073	0.891	0.893
Domain: Counter Example Problem														
denormal	1,156,224	178,800	100.9	214.6	0.027	0.018	1.0*	1.0*	99.92	215.4	0.028	0.018	1.0*	1.0*
Domain: Economic Problem														
mac_econ_fwd500	1,273,389	413,000	50.49	189.5	0.014	0.027	1.000	1.000	50.86	188.3	0.015	0.027	0.645	0.645
Domain: Electromagnetics Problem														
vfem	1,434,636	186,952	51.30	113.4	0.021	0.023	1.000	1.000	51.16	113.7	0.022	0.023	0.817	0.817
pli	1,350,309	45,390	96.50	121.1	0.029	0.074	1.0*	1.0*	95.42	119.6	0.029	0.075	1.0*	1.0*
Domain: Frequency Domain Circuit Simulation														
twotone	1,224,224	241,500	87.85	229.3	0.016	0.059	1.000	1.000	87.79	232.7	0.016	0.058	1.0*	1.0*
Domain: Graph														
web-NotreDame	1,497,134	651,458	32.19	154.7	0.021	0.187	1.0*	1.0*	32.09	154.2	0.023	0.186	1.0*	1.0*
598a	1,483,868	221,942	33.53	90.34	0.005	0.026	1.000	1.000	33.70	90.66	0.004	0.025	1.0*	1.0*
NotreDame_actors	1,470,404	520,223	15.11	90.60	0.007	0.025	1.000	1.000	15.11	92.16	0.007	0.023	0.975	0.975
rgg_n_2_17_s0	1,457,506	262,144	39.38	113.4	0.010	0.011	1.0*	1.0*	39.88	113.7	0.010	0.011	0.702	0.702
ga2010	1,418,056	582,172	29.56	145.1	0.007	0.013	1.000	1.000	29.56	145.1	0.007	0.013	1.0*	1.0*
nc2010	1,416,620	577,974	34.63	168.1	0.007	0.014	1.000	1.000	36.89	175.1	0.007	0.013	1.0*	1.0*
va2010	1,402,128	571,524	27.16	131.3	0.006	0.012	1.0*	1.0*	27.55	133.1	0.007	0.012	1.0*	1.0*
fe_rotor	1,324,862	199,234	56.18	134.0	0.014	0.055	1.0*	1.0*	56.50	142.1	0.013	0.055	1.0*	1.0*
in2010	1,281,716	534,142	37.64	168.9	0.008	0.015	1.0*	1.0*	37.47	170.7	0.008	0.015	1.0*	1.0*
ok2010	1,274,148	538,236	37.79	168.0	0.006	0.011	1.0*	1.0*	37.41	167.9	0.006	0.012	1.0*	1.0*
amazon0302	1,234,877	524,222	28.71	127.0	0.009	0.017	1.000	1.000	29.02	127.9	0.008	0.017	0.817	0.817
al2010	1,230,482	504,532	31.06	130.3	0.006	0.013	1.000	1.000	31.75	130.8	0.006	0.012	1.0*	1.0*
mn2010	1,227,102	519,554	39.36	169.5	0.008	0.016	1.000	1.000	39.50	171.8	0.008	0.015	0.990	0.990
caidaRouterLevel	1,218,132	384,488	20.94	69.64	0.005	0.016	1.000	1.000	20.96	69.50	0.005	0.016	1.0*	1.0*
language	1,216,334	798,260	26.04	165.3	0.014	0.163	1.000	1.000	26.04	164.9	0.016	0.189	0.961	0.961
wi2010	1,209,404	506,192	39.45	165.4	0.008	0.016	1.0*	1.0*	39.12	165.5	0.008	0.015	1.0*	1.0*
Linux_call_graph	1,208,908	648,170	31.99	156.2	0.010	0.020	1.000	1.000	31.56	156.4	0.010	0.021	0.984	0.984
az2010	1,196,094	483,332	30.77	130.4	0.006	0.013	1.0*	1.0*	31.15	124.9	0.006	0.013	1.0*	1.0*
tn2010	1,193,966	480,232	31.69	126.5	0.007	0.015	1.0*	1.0*	31.27	128.7	0.007	0.015	0.777	0.777
connectus	1,127,525	395,304	40.31	35.64	0.019	1.356	1.0*	1.0*	39.41	32.80	0.018	1.426	0.790	1.0*
ks2010	1,121,798	477,200	33.32	132.0	0.008	0.016	1.0*	1.0*	34.01	131.8	0.008	0.016	0.943	0.943
vsp_finan512_scagr7-2c_rlfddd	1,104,040	279,504	20.91	54.71	0.012	0.095	1.0*	1.0*	20.86	54.70	0.012	0.094	0.818	0.818
ia2010	1,021,170	432,014	42.98	152.8	0.008	0.017	1.000	1.000	42.76	160.1	0.009	0.017	0.937	0.937
G_n_pin_pout	1,002,396	200,000	43.53	90.18	0.006	0.008	1.000	1.000	43.06	90.48	0.006	0.008	0.720	0.720

Figure A-2: On a subset of the matrices from Suitesparse [10] between 1 and 1.5 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			B = 12						B = 4					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: Least Squares														
Maragal_8	1,308,415	108,289	19.72	30.02	0.016	0.398	1.000	1.0*	19.85	30.78	0.015	0.385	0.874	0.917
Maragal_7	1,200,537	73,409	17.63	25.87	0.020	0.802	0.876	0.959	17.43	25.91	0.020	0.763	0.892	0.952
landmark	1,151,232	74,656	78.80	144.9	0.027	0.043	0.816	0.818	77.40	145.8	0.027	0.044	0.832	0.832
Domain: Linear Programming														
lp_osa_60	1,408,073	253,526	17.89	20.89	0.017	1.339	1.000	1.0*	17.80	22.59	0.018	1.357	1.0*	1.0*
dbir2	1,158,159	64,783	36.15	39.03	0.024	0.405	1.0*	1.0*	35.85	39.48	0.022	0.429	1.0*	1.0*
pds-100	1,096,002	670,820	36.81	113.0	0.004	0.027	1.000	1.000	36.96	112.4	0.004	0.028	0.975	0.975
dbic1	1,081,843	269,517	36.82	61.39	0.014	0.207	1.0*	1.0*	36.04	61.17	0.015	0.199	0.716	0.716
dbir1	1,077,025	64,579	42.62	43.87	0.022	0.418	1.0*	1.0*	42.40	43.12	0.022	0.431	1.0*	1.0*
ts-palko	1,076,903	69,237	74.82	83.39	0.014	0.144	1.000	1.0*	74.58	84.19	0.013	0.163	0.841	0.852
watson_1	1,055,093	588,147	53.56	208.4	0.018	0.060	1.000	1.000	54.43	208.2	0.018	0.059	1.0*	1.0*
nemsemm1	1,053,986	79,297	122.9	87.94	0.027	0.964	0.737	0.778	123.1	90.37	0.025	1.050	1.0*	1.0*
pds-90	1,014,136	618,271	37.27	104.0	0.004	0.030	1.0*	1.0*	37.26	109.8	0.003	0.028	0.882	0.882
Domain: Materials Problem														
xenon1	1,181,120	97,200	106.2	157.6	0.017	0.046	0.815	0.815	106.4	158.7	0.017	0.049	0.863	0.863
viscorocks	1,162,244	75,524	106.1	151.7	0.027	0.031	0.865	0.865	104.5	150.7	0.026	0.032	0.874	0.874
Domain: Model Reduction Problem														
windscreen	1,482,390	45,384	66.74	93.84	0.031	0.027	0.808	0.808	66.62	93.98	0.030	0.025	0.535	0.535
gyro	1,021,159	34,722	126.4	113.7	0.020	0.097	0.607	0.607	126.6	113.9	0.020	0.110	0.701	0.701
Domain: Optimization														
net75	1,489,200	46,240	45.35	71.02	0.021	0.143	0.966	0.966	45.02	71.24	0.021	0.140	0.855	0.855
c-73	1,279,274	338,844	22.30	75.33	0.019	0.334	1.000	1.0*	22.61	73.35	0.020	0.313	1.0*	1.0*
boyd1	1,211,231	186,558	26.46	50.71	0.028	0.616	0.957	0.940	26.80	47.26	0.028	0.622	0.870	0.908
struct3	1,173,694	107,140	90.07	138.9	0.027	0.031	1.0*	1.0*	89.98	139.6	0.027	0.032	1.0*	1.0*
EternityII_Etilde	1,170,516	214,358	35.38	48.65	0.015	0.370	1.0*	1.0*	35.10	51.15	0.015	0.367	1.0*	1.0*
Domain: Power Network Problem														
TSOPF_RS_b300_c1	1,474,325	29,076	48.99	57.12	0.043	0.198	0.576	0.614	49.49	57.13	0.039	0.201	0.561	0.559
hvdc2	1,347,273	379,720	55.68	194.0	0.018	0.037	1.0*	1.0*	55.81	194.1	0.018	0.036	1.0*	1.0*
TSOPF_RS_b39_c30	1,079,986	120,196	58.85	92.56	0.030	0.105	0.762	0.762	59.26	91.90	0.030	0.098	0.943	0.943
case39	1,042,160	80,432	38.62	48.27	0.031	0.606	0.698	0.727	38.29	48.23	0.029	0.614	0.771	0.779
Domain: Semiconductor Device Problem														
matrix_9	2,121,550	206,860	53.68	159.6	0.024	0.034	0.723	0.723	53.83	160.1	0.025	0.040	0.795	0.795
Domain: Structural														
bcsstk35	1,450,163	60,474	93.48	125.6	0.023	0.078	0.826	0.836	96.39	126.7	0.022	0.070	0.983	0.977
raefsky4	1,328,611	39,558	90.37	109.5	0.027	0.062	0.980	0.980	90.52	109.2	0.027	0.065	0.718	0.720
msc10848	1,229,778	21,696	92.13	88.08	0.021	0.131	0.593	0.593	91.42	87.44	0.022	0.134	0.804	0.804
bcsstk31	1,181,416	71,176	100.7	120.2	0.025	0.087	1.0*	1.0*	94.79	120.5	0.026	0.087	0.864	0.864
msc23052	1,154,814	46,104	108.5	120.9	0.024	0.080	1.0*	1.0*	103.8	121.3	0.024	0.079	1.0*	1.0*
bcsstk36	1,143,140	46,104	91.35	98.43	0.028	0.075	0.849	0.850	91.32	99.30	0.027	0.076	0.833	0.834
bcsstk37	1,140,977	51,006	98.32	107.2	0.030	0.085	0.927	0.929	98.18	107.4	0.029	0.080	0.942	0.945
dawson5	1,010,777	103,074	94.37	134.8	0.026	0.075	0.981	0.981	93.92	134.3	0.024	0.080	1.0*	1.0*
Domain: Subsequent Theoretical/Quantum Chemistry Problem														
nemeth21	1,173,746	19,012	137.6	107.8	0.025	0.020	0.952	0.952	136.6	107.2	0.025	0.021	0.915	0.915
Domain: Theoretical/Quantum Chemistry														
nemeth22	1,358,832	19,012	123.5	108.5	0.021	0.019	0.922	0.922	121.1	109.3	0.022	0.019	0.914	0.914
SiO	1,317,655	66,802	74.55	117.2	0.022	0.152	1.0*	1.0*	74.50	116.2	0.023	0.146	1.0*	1.0*
Domain: Thermal Problem														
thermomech_dM	1,423,116	408,632	27.75	114.6	0.008	0.009	1.0*	1.0*	27.67	114.7	0.008	0.009	0.793	0.793

Figure A-3: Over the remaining matrices from Suitesparse [10] with between 1 and 1.5 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

			$B = 12$						$B = 4$					
Matrix Information			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
turon_m	1,690,876	379,848	55.65	223.0	0.021	0.021	1.000	1.000	0.178	94.13	0.090	0.006	1.0*	1.0*
av41092	1,683,902	82,184	34.35	67.15	0.017	0.194	1.000	0.724	0.146	16.83	0.081	0.084	0.612	0.612
d_pretok	1,641,672	365,460	58.51	229.8	0.022	0.022	1.0*	1.0*	0.182	96.70	0.094	0.006	1.0*	1.0*
Domain: Acoustics Problem														
qa8fm	1,660,579	132,254	78.67	175.5	0.028	0.025	1.0*	1.0*	0.220	52.41	0.134	0.008	1.0*	1.0*
qa8fk	1,660,579	132,254	76.77	172.7	0.029	0.024	1.0*	1.0*	0.212	53.71	0.141	0.008	1.0*	1.0*
Domain: Chemical Process Simulation														
Zd_Jac3	1,916,152	45,670	59.34	86.82	0.029	0.329	1.000	0.841	0.189	16.48	0.115	0.088	1.0*	1.0*
Zd_Jac6	1,711,983	45,670	55.98	76.30	0.030	0.335	0.835	0.829	0.173	14.78	0.117	0.087	0.793	0.809
Zd_Jac2	1,642,833	45,670	63.98	84.10	0.028	0.307	1.0*	1.0*	0.199	16.75	0.122	0.078	0.847	0.850
lhr71c	1,528,092	140,608	65.37	137.9	0.029	0.086	1.0*	1.0*	0.192	44.06	0.092	0.022	1.0*	1.0*
Domain: Circuit Simulation Problem														
ASIC_320k	2,635,364	643,642	21.91	139.0	0.017	0.302	1.000	1.0*	0.072	61.01	0.086	0.152	1.0*	1.0*
ASIC_680ks	2,329,176	1,365,424	24.89	278.6	0.018	0.050	1.000	1.000	0.080	144.7	0.081	0.012	1.0*	1.0*
rajat24	1,948,235	716,344	29.05	186.9	0.018	0.215	1.0*	1.0*	0.116	91.31	0.084	0.076	1.0*	1.0*
rajat21	1,893,370	823,352	39.23	278.4	0.018	0.247	1.000	1.0*	0.128	139.5	0.083	0.072	1.0*	1.0*
Domain: Combinatorial Problem														
ch8-8-b4	1,881,600	493,920	44.90	320.1	0.017	0.011	1.0*	1.0*	0.146	157.0	0.072	0.004	1.0*	1.0*
n4c6-b9	1,865,580	385,453	56.36	252.2	0.018	0.012	1.0*	1.0*	0.188	102.1	0.081	0.005	1.0*	1.0*
GL7d14	1,831,183	218,646	23.58	86.92	0.002	0.004	1.000	1.000	0.083	37.07	0.005	0.000	1.0*	1.0*
IG5-18	1,790,490	89,444	58.63	121.6	0.012	0.051	1.000	1.000	0.233	29.78	0.053	0.011	0.979	0.979
n4c6-b8	1,790,055	362,110	59.55	272.4	0.019	0.012	1.000	1.000	0.186	114.1	0.076	0.005	1.0*	1.0*
bibd_18_9	1,750,320	48,773	73.94	71.41	0.017	0.700	1.0*	1.0*	0.311	7.881	0.093	0.502	0.875	1.0*
TF18	1,597,545	219,235	59.66	155.0	0.011	0.041	1.0*	1.0*	0.239	53.32	0.051	0.008	0.952	0.952
ch7-9-b4	1,587,600	423,360	35.52	209.9	0.017	0.013	1.0*	1.0*	0.131	103.5	0.075	0.005	0.915	0.915
Domain: Computational Fluid Dynamics														
mixtank_new	1,995,041	59,914	46.80	89.76	0.024	0.068	1.0*	1.0*	0.136	18.24	0.108	0.028	0.968	0.968
cfd1	1,828,364	141,312	59.03	144.3	0.027	0.043	1.0*	1.0*	0.194	41.69	0.118	0.015	1.0*	1.0*
invextr1_new	1,793,881	60,824	50.30	90.01	0.026	0.098	1.0*	0.910	0.154	19.42	0.105	0.033	1.0*	1.0*
bbmat	1,771,722	77,488	57.66	97.19	0.031	0.067	0.902	0.910	0.169	22.53	0.088	0.022	0.772	0.796
ns3Da	1,679,599	40,828	57.68	92.78	0.009	0.055	1.0*	1.0*	0.171	16.89	0.044	0.017	0.994	0.994
Domain: Electromagnetics Problem														
fem_filter	1,731,206	148,124	48.23	111.3	0.020	0.151	1.0*	1.0*	0.150	33.34	0.083	0.047	1.0*	1.0*
2cubes_sphere	1,647,264	202,984	69.22	180.8	0.013	0.033	1.000	1.000	0.213	61.55	0.054	0.008	1.0*	1.0*
Domain: Graph														
coAuthorsDBLP	1,955,352	598,134	17.93	89.25	0.011	0.049	1.0*	1.0*	0.060	42.16	0.071	0.019	1.0*	1.0*
appu	1,853,104	28,000	45.01	72.75	0.008	0.014	1.0*	1.0*	0.133	10.95	0.022	0.002	0.886	0.886
oh2010	1,768,240	730,688	27.60	171.9	0.008	0.012	1.000	1.000	0.118	86.99	0.038	0.004	0.990	0.990
ny2010	1,709,544	700,338	22.54	136.1	0.008	0.012	1.000	1.000	0.076	68.62	0.039	0.004	0.867	0.867
mo2010	1,656,568	687,130	29.66	170.7	0.007	0.012	1.000	1.000	0.117	86.83	0.036	0.003	1.0*	1.0*
coAuthorsCiteseer	1,628,268	454,640	27.06	105.3	0.015	0.073	1.0*	1.0*	0.109	49.08	0.082	0.027	1.0*	1.0*
dblp-2010	1,615,400	652,372	29.43	148.0	0.017	0.068	1.0*	1.0*	0.119	74.96	0.094	0.022	1.0*	1.0*
mi2010	1,578,090	659,770	31.11	172.8	0.008	0.014	1.000	1.000	0.127	87.69	0.035	0.003	1.0*	1.0*
delaunay_n18	1,572,792	524,288	38.65	172.9	0.018	0.028	1.0*	1.0*	0.109	83.78	0.080	0.008	0.985	0.960
Domain: Linear Programming														
watson_2	1,846,391	1,029,237	32.91	224.5	0.015	0.058	1.000	1.000	0.108	105.0	0.067	0.023	0.797	0.797
karted	1,770,349	179,617	35.58	66.83	0.013	0.222	1.000	1.0*	0.103	17.19	0.055	0.147	1.0*	1.0*
lp_nug30	1,567,800	431,610	39.67	93.00	0.009	0.099	1.000	1.000	0.151	26.19	0.058	0.028	1.0*	1.0*
neos	1,526,794	995,024	40.44	330.4	0.014	0.020	1.0*	1.0*	0.146	171.9	0.069	0.006	1.0*	0.938

Figure A-4: Over a subset of matrices from Suitesparse [10] with between 1.5 and 2 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			B = 12						B = 4					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: Materials Problem														
crystk03	1,751,178	49,392	79.95	125.5	0.028	0.048	0.618	0.618	0.221	25.15	0.086	0.010	0.635	0.635
Domain: Model Reduction Problem														
gas_sensor	1,703,365	133,834	77.09	177.3	0.025	0.045	1.0*	1.0*	0.240	53.43	0.127	0.017	1.0*	1.0*
Domain: Optimization Problem														
crashbasis	1,750,416	320,000	46.51	166.4	0.029	0.018	0.959	0.959	0.132	67.91	0.128	0.004	1.0*	1.0*
majorbasis	1,750,416	320,000	65.10	235.8	0.029	0.018	1.0*	1.0*	0.188	94.69	0.127	0.004	1.0*	1.0*
lp1	1,643,420	1,068,776	20.89	181.5	0.020	0.442	1.0*	1.0*	0.079	99.80	0.090	0.195	1.0*	1.0*
EternityII_E	1,503,732	273,221	16.91	30.41	0.015	0.408	1.0*	1.0*	0.068	4.595	0.063	0.185	0.889	0.889
boyd2	1,500,397	932,632	31.93	235.9	0.017	0.289	1.000	1.0*	0.112	129.0	0.079	0.161	0.925	1.0*
Domain: Power Network Problem														
TSOPF_FS_b39_c19	1,977,600	152,432	20.79	50.40	0.031	0.636	0.593	0.636	0.071	15.01	0.124	0.188	0.693	0.703
TSOPF_FS_b162_c3	1,801,300	61,596	26.31	40.42	0.043	0.485	0.650	0.628	0.080	8.946	0.109	0.134	0.693	0.711
Domain: Structural														
trdheim	1,935,324	44,196	64.07	99.25	0.019	0.054	0.582	0.582	0.232	19.13	0.032	0.012	0.776	0.785
opt1	1,930,655	30,898	73.66	107.8	0.021	0.084	1.0*	0.998	0.271	19.04	0.088	0.032	0.797	0.769
Lin	1,766,400	512,000	40.30	203.5	0.024	0.018	1.0*	1.0*	0.115	93.28	0.100	0.005	1.0*	1.0*
pkustk09	1,583,640	67,920	67.37	106.6	0.018	0.052	0.591	0.591	0.244	26.67	0.048	0.020	0.605	0.605
sparsine	1,548,988	100,000	37.70	72.97	0.007	0.009	1.000	1.000	0.114	19.21	0.023	0.001	1.0*	1.0*
Domain: Subsequent Computational Fluid Dynamics														
venkat25	1,717,792	124,848	51.33	117.1	0.017	0.031	0.580	0.580	0.168	35.41	0.062	0.012	0.790	0.790
venkat50	1,717,792	124,848	57.55	135.1	0.017	0.029	0.638	0.638	0.346	39.53	0.063	0.013	0.575	0.575
venkat01	1,717,792	124,848	69.89	157.9	0.017	0.031	0.819	0.819	0.212	47.44	0.063	0.012	0.779	0.779
Domain: Subsequent Theoretical/Quantum Chemistry Problem														
nemeth26	1,511,760	19,012	107.4	105.2	0.024	0.019	0.763	0.766	0.274	16.57	0.088	0.010	0.638	0.637
nemeth25	1,511,758	19,012	96.52	93.17	0.024	0.020	0.802	0.802	0.244	14.71	0.088	0.010	0.843	0.868
nemeth23	1,506,810	19,012	94.45	92.29	0.021	0.018	0.798	0.798	0.233	14.57	0.085	0.010	0.849	0.872
nemeth24	1,506,550	19,012	107.8	104.9	0.024	0.021	0.925	0.923	0.266	16.55	0.086	0.010	0.969	0.993
Domain: Theoretical/Quantum Chemistry Problem														
conf5_4-8x8-10	1,916,928	98,304	72.22	147.9	0.018	0.046	0.768	0.768	0.207	37.89	0.073	0.016	0.895	0.895

Figure A-5: Over the remaining matrices from Suitesparse [10] with between 1 and 1.5 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
wave	2,118,662	312,634	39.98	157.0	0.022	0.032	1.0*	1.0*	0.142	58.03	0.096	0.009	1.0*	1.0*
mario002	2,101,242	779,748	22.39	156.9	0.014	0.016	1.0*	1.0*	0.070	77.14	0.077	0.006	1.0*	1.0*
darcy003	2,101,242	779,748	19.33	140.4	0.015	0.016	1.000	1.000	0.066	66.06	0.071	0.006	1.0*	1.0*
mc2depi	2,100,225	1,051,650	26.00	233.2	0.019	0.010	1.0*	1.0*	0.081	122.9	0.097	0.003	1.0*	1.0*
Domain: Combinatorial Problem														
c8_mat11	2,462,970	10,323	51.33	82.71	0.024	0.259	1.0*	1.0*	0.221	10.57	0.111	0.096	1.0*	1.0*
wheel_601	2,170,814	1,625,708	10.68	158.5	0.007	0.046	1.0*	1.0*	0.033	87.88	0.050	0.023	1.0*	1.0*
Domain: Computational Fluid Dynamics														
poisson3Db	2,374,949	171,246	17.06	52.28	0.006	0.023	1.000	1.000	0.057	15.06	0.034	0.006	0.789	0.789
rma10	2,374,001	93,670	41.57	87.80	0.023	0.054	0.745	0.736	0.127	20.41	0.098	0.018	0.797	0.803
water_tank	2,035,281	121,480	66.45	153.1	0.027	0.083	1.0*	1.0*	0.279	54.40	0.110	0.036	1.0*	1.0*
Domain: Graph														
vsp_msc10848_300sep_100in_1Kout	2,442,056	43,992	25.33	53.96	0.006	0.012	1.0*	1.0*	0.080	8.791	0.017	0.003	0.623	0.623
fl2010	2,346,294	968,962	16.00	132.9	0.007	0.009	1.000	1.000	0.057	67.34	0.033	0.003	1.0*	1.0*
citationCiteseer	2,313,294	536,990	15.85	85.89	0.000	0.001	1.000	1.000	0.061	37.33	0.002	0.000	1.0*	1.0*
Stanford	2,312,497	563,806	17.29	93.91	0.002	0.104	1.0*	1.0*	0.044	36.13	0.011	0.022	1.0*	1.0*
web-Stanford	2,312,497	563,806	14.61	78.14	0.002	0.005	1.000	1.000	0.051	36.15	0.011	0.001	0.870	0.870
il2010	2,164,464	903,108	18.52	142.8	0.007	0.012	1.0*	1.0*	0.061	54.65	0.033	0.003	0.651	0.651
144	2,148,786	289,298	22.30	79.90	0.007	0.044	1.000	1.000	0.076	29.54	0.052	0.013	1.0*	1.0*
pa2010	2,058,462	843,090	22.53	162.7	0.008	0.011	1.000	1.000	0.078	81.99	0.037	0.003	1.0*	1.0*
cage12	2,032,536	260,456	39.26	146.8	0.018	0.037	1.000	1.000	0.138	50.87	0.081	0.010	1.0*	1.0*
Domain: Least Squares Problem														
Delor295K	2,401,323	2,119,662	22.43	160.1	0.015	0.024	1.0*	1.0*	0.075	63.39	0.068	0.007	1.0*	1.0*
Domain: Linear Programming														
neos3	2,055,024	1,031,041	25.09	226.5	0.018	0.030	1.0*	1.0*	0.084	118.7	0.080	0.008	1.0*	1.0*
Domain: Model Reduction Problem														
CurlCurl_1	2,472,071	452,902	33.49	177.2	0.020	0.011	1.0*	1.0*	0.113	69.89	0.088	0.007	1.0*	1.0*
Domain: Optimization Problem														
net4-1	2,441,727	176,686	30.73	104.9	0.019	0.136	1.000	1.000	0.102	31.85	0.088	0.062	0.982	1.0*
c-big	2,341,011	690,482	15.63	102.0	0.016	0.072	1.0*	1.0*	0.071	62.65	0.092	0.038	1.0*	1.0*
exdata_1	2,269,501	12,002	18.21	27.95	0.033	3.759	0.455	0.443	0.059	3.557	0.053	0.024	0.451	0.459
gupta1	2,164,210	63,604	29.87	54.54	0.022	0.533	0.976	0.995	0.099	11.18	0.105	0.228	0.997	1.0*
net100	2,033,200	59,840	25.93	54.25	0.021	0.142	1.0*	1.0*	0.082	10.85	0.090	0.051	1.0*	1.0*
Domain: Power Network Problem														
TSOPF_FS_b162_c4	2,398,220	81,596	19.78	40.94	0.041	0.532	0.598	0.679	0.064	8.725	0.113	0.132	0.714	0.717
TSC_OPF_1047	2,016,902	16,280	32.70	47.06	0.051	1.068	0.496	0.492	0.096	6.640	0.105	0.062	0.511	0.513
Domain: Semiconductor Device Problem Sequence														
barrier2-9	3,897,557	231,250	17.72	82.00	0.018	0.063	1.0*	1.0*	0.064	21.62	0.092	0.014	1.0*	1.0*
barrier2-1	3,805,068	226,152	18.70	84.45	0.019	0.076	1.0*	1.0*	0.063	22.20	0.091	0.015	1.0*	1.0*
Domain: Structural														
oilpan	3,597,188	147,504	33.72	115.3	0.026	0.034	0.590	0.590	0.110	27.36	0.082	0.012	0.824	0.829
tsyl201	2,454,957	41,370	58.62	111.8	0.021	0.064	0.565	0.565	0.180	19.10	0.086	0.019	0.835	0.835
pkustk07	2,418,804	33,720	51.46	91.90	0.019	0.125	0.560	0.560	0.153	15.03	0.082	0.038	0.629	0.629
vanbody	2,336,898	94,144	50.45	109.5	0.025	0.066	0.753	0.796	0.154	25.81	0.099	0.024	0.956	0.955
pkustk05	2,205,144	74,328	62.53	128.6	0.018	0.051	0.613	0.613	0.234	28.05	0.049	0.019	0.940	0.940
bcsstk39	2,089,294	93,544	64.01	138.6	0.023	0.030	0.875	0.875	0.194	33.26	0.087	0.023	1.0*	1.0*
sme3Db	2,081,063	58,134	42.30	87.89	0.009	0.055	1.000	1.000	0.131	16.83	0.046	0.014	0.947	0.947
bcsstk30	2,043,492	57,848	66.39	114.2	0.022	0.072	0.736	0.748	0.181	23.47	0.095	0.028	0.746	0.753
bcsstk32	2,014,701	89,218	69.00	132.3	0.028	0.074	0.942	0.932	0.196	33.05	0.104	0.025	0.854	0.853
Domain: Subsequent Semiconductor Device Problem														
para-10	5,416,358	311,848	16.00	102.9	0.019	0.053	1.000	1.000	0.056	26.19	0.096	0.012	1.0*	1.0*
Domain: Theoretical/Quantum Chemistry Problem														
H2O	2,216,736	134,048	43.31	123.7	0.021	0.013	1.0*	1.0*	0.130	32.27	0.106	0.005	1.0*	1.0*

Figure A-6: Over the matrices from Suitesparse [10] with between 2 and 2.5 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
helm2d03	2,741,935	784,514	26.63	209.8	0.016	0.027	1.000	1.000	0.120	124.0	0.073	0.006	1.0*	1.0*
cop20k_A	2,624,331	242,384	14.96	53.66	0.016	0.053	0.793	0.793	0.050	17.27	0.094	0.018	0.892	0.892
Domain: Circuit Simulation Problem														
ASIC_680k	3,871,773	1,365,724	9.764	122.8	0.016	0.335	1.0*	1.0*	0.034	58.43	0.072	0.198	1.0*	1.0*
Domain: Combinatorial Problem														
Trec14	2,872,265	19,064	48.62	87.41	0.025	0.148	1.0*	1.0*	0.218	15.21	0.094	0.042	1.0*	1.0*
GL7d23	2,695,430	454,497	10.66	38.33	0.001	0.005	1.0*	1.0*	0.037	11.64	0.004	0.000	0.910	0.910
Domain: Computational Fluid Dynamics														
ramage02	2,866,352	33,660	47.91	104.0	0.021	0.073	0.926	0.943	0.147	15.88	0.087	0.024	1.0*	1.0*
Domain: Linear Programming														
stat96v2	2,852,184	986,521	43.47	107.9	0.018	0.028	0.698	0.698	0.104	17.91	0.071	0.017	0.708	0.708
Domain: Model Reduction Problem														
filter3D	2,707,179	212,874	31.95	110.3	0.014	0.037	1.0*	1.0*	0.106	32.93	0.082	0.014	0.975	0.901
ch7-9-b5	2,540,160	740,880	24.76	213.1	0.016	0.010	1.0*	1.0*	0.082	99.48	0.071	0.004	1.0*	1.0*
Domain: Optimization Problem														
ins2	2,751,484	618,824	8.533	53.72	0.017	0.321	1.0*	1.0*	0.029	23.09	0.080	0.110	1.0*	1.0*
net125	2,577,200	73,440	17.51	45.80	0.021	0.132	0.863	0.863	0.053	8.873	0.113	0.044	0.956	0.956
Domain: Power Network Problem														
TSOPF_RS_b300_c2	2,943,887	56,676	24.69	57.23	0.040	0.111	0.506	0.516	0.070	10.22	0.096	0.016	0.624	0.639
Domain: Semiconductor Device Problem														
para-4	5,326,228	306,452	13.64	85.27	0.019	0.056	1.0*	1.0*	0.050	22.36	0.090	0.012	1.0*	1.0*
Domain: Structural														
srb1	2,962,152	109,848	36.75	102.0	0.021	0.042	0.467	0.467	0.102	23.46	0.039	0.009	0.519	0.519
pct20stif	2,698,463	104,658	45.45	111.7	0.025	0.068	0.789	0.789	0.130	25.73	0.098	0.022	0.801	0.800
ct20stif	2,698,463	104,658	46.00	113.1	0.026	0.066	1.0*	1.0*	0.135	25.99	0.101	0.022	0.767	0.765
nasasrb	2,677,324	109,740	42.68	105.5	0.020	0.045	0.541	0.541	0.125	24.93	0.062	0.020	0.558	0.558
pkustk06	2,571,768	86,328	51.65	130.2	0.018	0.047	0.614	0.614	0.169	26.86	0.043	0.019	0.626	0.626
Domain: Thermal Problem														
thermomech_dK	2,846,228	408,632	14.38	76.07	0.010	0.009	0.542	0.542	0.057	29.43	0.052	0.004	0.532	0.532

Figure A-7: Over the matrices from Suitesparse [10] with between 2.5 and 3 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			B = 12						B = 4					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
Dubcova3	3,636,649	293,378	17.04	83.35	0.022	0.065	1.0*	1.0*	0.058	25.22	0.106	0.016	1.0*	1.0*
Chevron3	3,413,113	762,762	25.79	197.4	0.031	0.009	1.0*	1.0*	0.081	85.70	0.147	0.004	1.0*	1.0*
nd3k	3,279,690	18,000	42.79	81.42	0.029	0.037	0.568	0.568	0.131	11.26	0.078	0.006	0.649	0.696
stomach	3,021,648	426,720	35.10	190.2	0.023	0.022	1.0*	1.0*	0.126	69.29	0.112	0.010	0.866	0.866
Domain: Circuit Simulation														
rajat29	4,866,270	1,287,988	12.20	149.8	0.017	0.387	1.0*	1.0*	0.031	51.34	0.084	0.176	0.892	0.928
Domain: Combinatorial Problem														
ch8-8-b5	3,386,880	940,800	18.40	211.7	0.017	0.009	1.0*	1.0*	0.062	98.28	0.076	0.003	0.922	0.922
bibd_19_9	3,325,608	92,549	35.40	67.78	0.019	0.726	1.0*	1.0*	0.140	6.940	0.089	0.492	1.0*	1.0*
Domain: Computational Fluid Dynamics														
laminar_duct3D	3,833,077	134,346	21.82	83.22	0.028	0.051	0.684	0.684	0.075	17.87	0.107	0.012	0.673	0.673
parabolic_fem	3,674,625	1,051,650	19.67	214.0	0.017	0.020	1.0*	1.0*	0.068	96.08	0.087	0.006	1.0*	1.0*
3dtube	3,213,618	90,660	39.90	111.5	0.024	0.071	0.595	0.595	0.154	29.77	0.113	0.014	0.579	0.596
cfd2	3,087,898	246,880	38.54	157.2	0.026	0.039	1.0*	1.0*	0.130	47.59	0.122	0.010	1.0*	1.0*
Domain: Graph														
roadNet-TX	3,843,320	2,786,766	9.455	198.9	0.013	0.012	1.000	1.000	0.031	109.0	0.056	0.003	1.0*	1.0*
IMDB	3,782,463	1,324,748	6.621	60.23	0.001	0.004	1.000	1.000	0.023	25.11	0.011	0.001	0.945	0.945
ca2010	3,489,366	1,420,290	12.65	155.7	0.006	0.007	1.0*	1.0*	0.046	77.26	0.031	0.002	1.0*	1.0*
amazon0601	3,387,388	806,788	9.463	74.53	0.010	0.020	1.0*	1.0*	0.034	32.89	0.061	0.008	1.0*	1.0*
m14b	3,358,036	429,530	10.42	57.64	0.009	0.045	1.0*	1.0*	0.038	20.94	0.061	0.012	0.768	0.768
amazon0505	3,356,824	820,472	9.920	79.55	0.010	0.021	1.0*	1.0*	0.036	35.25	0.061	0.008	0.971	0.971
cnr-2000	3,216,152	651,114	20.58	123.5	0.027	0.094	1.0*	1.0*	0.067	53.19	0.109	0.033	1.0*	1.0*
amazon0312	3,200,440	801,454	9.925	77.74	0.009	0.020	1.0*	1.0*	0.036	34.77	0.063	0.007	0.999	0.999
delaunay_n19	3,145,646	1,048,576	17.18	155.8	0.019	0.020	1.0*	1.0*	0.060	73.25	0.078	0.006	1.0*	1.0*
webbase-1M	3,105,536	2,000,010	8.058	122.4	0.017	0.130	1.0*	1.0*	0.028	66.03	0.078	0.053	0.982	0.964
belgium_osm	3,099,940	2,882,590	7.595	170.2	0.019	0.015	1.000	1.000	0.025	96.33	0.080	0.004	0.958	0.958
rgg_n_2_18_s0	3,094,566	524,288	20.97	124.6	0.009	0.007	1.0*	1.0*	0.074	48.00	0.025	0.002	0.991	0.991
roadNet-PA	3,083,796	2,181,840	13.11	218.7	0.012	0.014	1.000	1.000	0.042	122.1	0.061	0.004	1.0*	1.0*
Domain: Linear Programming														
stormG2_1000	3,459,881	1,905,491	17.76	180.4	0.018	0.030	1.0*	1.0*	0.064	80.16	0.085	0.010	0.992	0.992
stat96v3	3,317,736	1,147,621	29.59	103.8	0.018	0.025	0.716	0.716	0.088	17.18	0.075	0.016	0.767	0.763
Domain: Materials														
xenon2	3,866,688	314,928	31.09	147.4	0.017	0.025	0.709	0.709	0.113	45.17	0.085	0.009	0.879	0.881
Domain: Optimization Problem														
net150	3,121,200	87,040	14.14	44.54	0.020	0.131	1.0*	1.0*	0.045	8.673	0.087	0.041	1.0*	1.0*
Domain: Power Network Problem														
TSOPF_FS_b39_c30	3,121,160	240,432	12.48	47.26	0.030	0.588	0.736	0.785	0.043	14.45	0.120	0.188	0.699	0.708
Domain: Structural														
ship_003	8,086,034	243,456	19.41	133.0	0.024	0.031	0.765	0.765	0.061	27.46	0.092	0.014	0.812	0.854
shipsec1	7,813,404	281,748	14.51	106.5	0.018	0.026	0.738	0.738	0.049	23.31	0.047	0.010	0.724	0.724
shipsec8	6,653,399	229,838	15.81	96.26	0.022	0.038	0.931	0.931	0.051	21.17	0.095	0.018	0.927	0.927
ship_001	4,644,230	69,840	25.02	88.91	0.028	0.060	0.958	0.960	0.081	14.65	0.100	0.023	0.895	0.893
s3dkt3m2	3,753,461	180,898	30.35	120.7	0.034	0.022	0.873	0.873	0.092	29.74	0.088	0.009	0.884	0.880
s4dkt3m2	3,753,461	180,898	30.97	124.0	0.034	0.021	0.883	0.883	0.096	30.39	0.089	0.009	0.903	0.887
smt	3,753,184	51,420	29.98	83.87	0.023	0.065	0.912	0.912	0.091	13.56	0.105	0.023	0.909	0.894
pkustk08	3,226,671	44,418	40.16	96.35	0.019	0.107	0.493	0.493	0.121	15.91	0.089	0.030	0.563	0.563
sme3Dc	3,148,656	85,860	15.11	46.85	0.008	0.045	1.000	1.000	0.050	9.190	0.042	0.011	0.989	0.989
pkustk03	3,130,416	126,672	37.87	116.6	0.019	0.039	0.570	0.570	0.127	27.51	0.046	0.017	0.816	0.816
Domain: Theoretical/Quantum Chemistry Problem														
GaAsH6	3,381,809	122,698	25.10	86.24	0.025	0.218	1.0*	1.0*	0.086	18.61	0.117	0.091	1.0*	1.0*
Domain: Thermal Problem														
FEM_3D_thermal2	3,489,300	295,800	27.16	128.9	0.029	0.024	1.0*	1.0*	0.105	39.24	0.122	0.008	1.0*	1.0*

Figure A-8: Over the matrices from Suitesparse [10] with between 3 and 4 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
ecology1	4,996,000	2,000,000	13.79	242.3	0.028	0.008	1.0*	1.0*	0.044	119.2	0.122	0.002	1.0*	1.0*
torso3	4,429,042	518,312	19.18	142.3	0.025	0.020	1.0*	1.0*	0.072	47.50	0.119	0.007	1.0*	1.0*
cant	4,007,383	124,902	27.48	100.0	0.027	0.032	0.605	0.605	0.096	20.82	0.087	0.008	0.742	0.742
Domain: Circuit Simulation														
LargeRegFile	4,944,201	2,912,528	10.92	351.5	0.016	0.009	1.0*	1.0*	0.050	264.3	0.080	0.003	1.0*	1.0*
Domain: Combinatorial														
TF19	4,370,721	558,984	10.44	72.10	0.011	0.025	1.0*	1.0*	0.037	23.44	0.047	0.004	0.958	0.958
Domain: Computational Chemistry														
iChem_Jacobian	4,137,369	548,174	18.93	145.8	0.024	0.017	1.000	1.000	0.070	50.46	0.099	0.005	1.0*	0.993
Domain: Electromagnetics														
t2em	4,590,832	1,843,264	13.39	223.0	0.026	0.007	1.0*	1.0*	0.042	108.6	0.113	0.002	1.0*	0.963
tmt_unsym	4,584,801	1,835,650	13.76	226.6	0.026	0.006	1.0*	1.0*	0.043	110.6	0.099	0.002	1.0*	1.0*
offshore	4,242,673	519,578	14.13	96.27	0.010	0.018	1.000	1.000	0.048	32.82	0.041	0.004	0.692	0.692
Domain: Graph														
kron_g500-logn16	4,912,469	131,072	8.407	41.88	0.005	0.068	1.000	1.000	0.030	7.987	0.021	0.017	0.914	0.914
netherlands_osm	4,882,476	4,433,376	6.123	213.7	0.014	0.010	1.000	1.000	0.022	120.7	0.068	0.003	0.938	0.938
tx2010	4,456,272	1,828,462	8.324	132.9	0.007	0.008	1.0*	1.0*	0.029	65.06	0.033	0.002	0.978	0.978
pdb1HYS	4,344,765	72,834	28.72	85.63	0.024	0.040	0.506	0.506	0.087	15.21	0.077	0.010	0.549	0.549
debr	4,194,298	2,097,152	12.12	230.8	0.015	0.007	1.0*	1.0*	0.041	118.1	0.059	0.003	1.0*	1.0*
vsp_bcsstk30_500sep_10in_1Kout	4,033,156	116,696	8.460	31.96	0.003	0.003	1.0*	1.0*	0.028	6.538	0.008	0.001	0.870	0.870
Domain: Least Squares														
Delor338K	4,211,599	1,230,294	15.18	124.5	0.021	0.030	1.0*	1.0*	0.050	47.54	0.104	0.009	1.0*	1.0*
Domain: Model Reduction Problem														
t3dh_e	4,352,105	158,342	25.29	107.8	0.021	0.036	1.0*	1.0*	0.078	26.30	0.098	0.016	1.0*	1.0*
Domain: Optimization														
gupta2	4,248,286	124,128	23.92	83.85	0.024	0.425	1.0*	1.0*	0.086	19.46	0.098	0.177	1.0*	1.0*
Domain: Power Network														
TSOPF_FS_b300	4,400,122	58,428	12.26	34.98	0.040	0.290	0.568	0.611	0.039	5.513	0.101	0.055	0.574	0.572
Domain: Structural														
shipsec5	10,113,096	359,720	11.11	103.7	0.026	0.026	0.966	0.966	0.036	22.46	0.100	0.012	0.988	0.985
s3dkq4m2	4,820,891	180,898	21.71	100.2	0.033	0.020	0.832	0.832	0.064	22.64	0.089	0.007	0.799	0.785
apache2	4,817,870	1,430,352	20.11	255.3	0.023	0.009	1.0*	1.0*	0.048	87.31	0.103	0.002	1.0*	1.0*
engine	4,706,073	287,142	8.920	44.52	0.018	0.036	0.515	0.515	0.034	12.43	0.080	0.011	0.464	0.464
thread	4,470,048	59,472	29.28	95.39	0.020	0.051	0.598	0.598	0.091	14.90	0.084	0.018	0.578	0.578
pkustk10	4,308,984	161,352	25.37	106.2	0.019	0.036	0.602	0.602	0.078	24.21	0.041	0.011	0.632	0.632
pkustk04	4,218,660	111,180	25.22	90.47	0.021	0.111	0.606	0.606	0.081	18.11	0.082	0.033	0.529	0.529

Figure A-9: Over the matrices from Suitesparse [10] with between 4 and 5 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
nd6k	6,897,316	36,000	20.97	86.27	0.031	0.025	0.736	0.736	0.070	11.04	0.086	0.004	0.736	0.729
Chevron4	6,376,412	1,422,900	12.30	178.7	0.033	0.009	1.0*	1.0*	0.040	76.46	0.137	0.003	1.0*	1.0*
consph	6,010,480	166,668	18.38	99.85	0.028	0.036	0.780	0.780	0.061	19.85	0.093	0.011	0.773	0.773
Domain: Circuit Simulation														
rajat30	6,175,377	1,287,988	6.927	96.05	0.018	0.385	1.0*	1.0*	0.024	40.63	0.091	0.140	1.0*	1.0*
Hamrle3	5,514,242	2,894,720	8.272	204.7	0.019	0.031	1.0*	1.0*	0.036	142.0	0.091	0.009	1.0*	1.0*
Domain: Combinatorial														
GL7d15	6,080,381	631,636	4.209	46.06	0.001	0.001	1.0*	1.0*	0.016	18.01	0.003	0.000	0.937	0.937
Domain: Frequency Domain Circuit Simulation														
pre2	5,959,282	1,318,066	12.35	169.4	0.017	0.032	1.000	1.000	0.045	71.05	0.076	0.011	1.0*	1.0*
Domain: Graph														
auto	6,629,222	897,390	4.744	53.47	0.006	0.024	1.0*	1.0*	0.018	19.78	0.042	0.007	0.871	0.871
rgg_n_2_19_s0	6,539,532	1,048,576	11.13	135.5	0.008	0.004	1.000	1.000	0.040	53.68	0.022	0.001	0.862	0.862
delaunay_n20	6,291,372	2,097,152	8.466	155.6	0.017	0.014	1.000	1.000	0.028	71.40	0.079	0.004	1.0*	1.0*
NACA0015	6,229,636	2,078,366	4.688	93.02	0.009	0.007	1.0*	1.0*	0.016	42.91	0.054	0.003	0.621	0.621
roadNet-CA	5,533,214	3,942,562	6.240	193.9	0.013	0.009	1.000	1.000	0.028	103.3	0.060	0.003	0.874	0.874
Domain: Least Squares														
sls	6,804,304	1,810,851	5.070	137.6	0.011	0.002	1.0*	1.0*	0.018	75.93	0.064	0.001	1.0*	1.0*
ESOC	6,019,939	364,892	13.03	128.5	0.013	0.008	0.854	0.854	0.059	57.45	0.075	0.003	0.865	0.865
Domain: Model Reduction														
boneS01	6,715,152	254,448	16.75	100.7	0.026	0.026	0.689	0.689	0.056	25.75	0.084	0.007	0.686	0.686
Domain: Power Network														
TSOPF_RS_b2052_c1	6,761,100	51,252	11.01	51.41	0.038	0.089	0.672	0.673	0.041	7.244	0.065	0.011	0.615	0.615
Domain: Semiconductor Device Problem														
ohne2	11,063,545	362,686	12.49	126.9	0.024	0.035	1.0*	1.0*	0.052	35.49	0.107	0.009	1.0*	1.0*
Domain: Structural														
pkustk13	6,616,827	189,786	17.74	96.44	0.019	0.050	0.867	0.867	0.056	19.71	0.098	0.020	0.833	0.812
Domain: Theoretical/Quantum Chemistry														
Ga10As10H30	6,115,633	226,162	14.62	93.90	0.024	0.085	1.0*	1.0*	0.052	20.51	0.119	0.037	1.0*	1.0*
Ga3As3H12	5,970,947	122,698	15.02	78.86	0.029	0.196	1.0*	1.0*	0.059	14.21	0.135	0.074	1.0*	1.0*

Figure A-10: Over the matrices from Suitesparse [10] with between 5 and 7 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
torso1	8,516,500	232,316	13.90	85.04	0.048	0.505	0.966	0.948	0.058	16.96	0.120	0.068	0.929	0.957
Domain: Circuit Simulation														
G3_circuit	7,660,826	3,170,956	6.910	174.7	0.023	0.020	1.0*	1.0*	0.026	113.2	0.113	0.002	1.0*	1.0*
Domain: Combinatorial														
bibd_22_8	8,953,560	320,001	15.73	72.07	0.020	0.802	1.0*	1.0*	0.064	7.706	0.092	0.470	1.0*	1.0*
bibd_20_10	8,314,020	184,946	20.81	82.91	0.019	0.690	1.0*	1.0*	0.083	9.349	0.091	0.462	1.0*	1.0*
GL7d22	8,251,000	1,172,365	3.273	38.52	0.001	0.001	1.000	1.000	0.012	11.68	0.002	0.000	1.0*	1.0*
Domain: Computational Fluid Dynamics														
atmosmodj	8,814,880	2,540,864	7.652	195.1	0.021	0.005	1.0*	1.0*	0.026	86.52	0.100	0.002	1.0*	1.0*
atmosmodd	8,814,880	2,540,864	8.291	212.8	0.021	0.005	1.0*	1.0*	0.029	96.60	0.101	0.002	1.0*	1.0*
PR02R	8,185,136	322,140	12.80	101.3	0.030	0.013	1.0*	1.0*	0.043	22.66	0.086	0.007	1.0*	1.0*
Domain: Computer Vision														
specular	7,647,616	479,576	12.66	146.0	0.019	0.023	0.990	0.989	0.041	53.82	0.078	0.006	0.949	0.937
Domain: Graph														
flickr	9,837,214	1,641,756	1.314	22.01	0.007	0.040	1.000	1.000	0.005	8.569	0.035	0.013	1.0*	1.0*
web-BerkStan	7,600,595	1,370,460	11.08	144.3	0.021	0.052	1.0*	1.0*	0.037	58.08	0.094	0.015	1.0*	1.0*
Stanford_Berkeley	7,583,376	1,366,892	9.226	133.8	0.021	0.280	1.0*	1.0*	0.037	52.47	0.095	0.151	1.0*	1.0*
cage13	7,479,343	890,630	10.59	138.6	0.017	0.020	1.0*	1.0*	0.039	46.42	0.078	0.005	0.918	0.918
Domain: Least Squares														
Rucci1	7,791,168	2,087,785	8.923	283.6	0.010	0.006	1.0*	1.0*	0.031	154.7	0.065	0.002	1.0*	1.0*
Domain: Linear Programming														
degme	8,127,528	844,916	12.99	101.0	0.016	0.076	1.0*	1.0*	0.039	22.68	0.069	0.060	1.0*	1.0*
rail2586	8,011,362	925,855	10.27	60.75	0.018	0.568	1.0*	1.0*	0.045	6.562	0.083	0.164	1.0*	1.0*
cont1_l	7,031,999	3,839,995	7.464	238.0	0.020	0.007	1.0*	1.0*	0.026	125.6	0.091	0.002	1.0*	1.0*
Domain: Model Reduction Problem														
CurlCurl_2	8,921,789	1,613,058	8.858	170.1	0.021	0.006	1.0*	1.0*	0.030	64.34	0.092	0.003	1.0*	1.0*
Domain: Optimization														
pattern1	9,323,432	38,484	11.65	77.32	0.016	0.129	1.0*	1.0*	0.055	12.40	0.065	0.021	0.906	0.906
gupta3	9,323,427	33,566	4.729	22.47	0.031	0.220	0.676	0.685	0.016	2.790	0.088	0.054	0.640	0.613
Domain: Power Network														
TSOPF_RS_b678_c2	8,781,949	71,392	10.95	59.75	0.034	0.059	0.693	0.684	0.038	8.595	0.060	0.008	0.683	0.673
TSOPF_FS_b300_c2	8,767,466	113,628	7.832	41.06	0.039	0.260	0.715	0.801	0.027	6.441	0.093	0.055	0.806	0.801
Domain: Structural														
hood	10,768,436	441,084	10.09	101.3	0.024	0.031	1.0*	1.0*	0.035	24.47	0.101	0.010	0.995	1.0*
x104	10,167,624	216,768	12.69	97.01	0.018	0.034	0.739	0.739	0.041	17.88	0.040	0.011	0.707	0.707
m_t1	9,753,570	195,156	12.86	93.38	0.020	0.038	0.683	0.683	0.041	16.85	0.073	0.014	0.683	0.683
gearbox	9,080,404	307,492	12.96	99.68	0.022	0.035	0.662	0.662	0.042	21.43	0.083	0.010	0.737	0.737
pkustk12	7,512,317	189,306	13.19	82.46	0.022	0.103	0.860	0.860	0.048	16.25	0.092	0.042	0.853	0.850
bmw7st_1	7,339,667	282,694	14.87	101.5	0.026	0.038	0.891	0.891	0.048	25.09	0.092	0.014	0.904	0.899
Domain: Theoretical/Quantum Chemistry														
Ga19As19H42	8,884,839	266,246	10.43	86.54	0.025	0.080	1.0*	1.0*	0.038	17.74	0.123	0.033	0.924	0.924
Ge99H100	8,451,395	225,970	13.39	99.38	0.024	0.062	1.0*	1.0*	0.050	19.78	0.112	0.027	1.0*	1.0*
Ge87H76	7,892,195	225,970	13.67	101.4	0.024	0.061	1.0*	1.0*	0.050	20.07	0.120	0.028	1.0*	1.0*
CO	7,666,057	442,238	12.73	123.1	0.022	0.009	1.0*	1.0*	0.046	32.31	0.101	0.004	1.0*	1.0*
Domain: Thermal Problem														
thermal2	8,580,313	2,456,090	5.731	129.6	0.015	0.019	1.0*	1.0*	0.019	57.66	0.080	0.004	1.0*	1.0*

Figure A-11: Over the matrices from Suitesparse [10] with between 7 and 10 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			B = 12						B = 4					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
nd12k	14,220,946	72,000	13.14	90.35	0.030	0.020	0.787	0.787	0.045	12.15	0.080	0.003	0.793	0.775
BenElechi1	13,150,496	491,748	9.369	107.0	0.023	0.010	0.741	0.741	0.030	24.32	0.039	0.003	0.747	0.744
kim2	11,330,020	913,952	9.809	141.1	0.034	0.006	1.0*	1.0*	0.035	41.80	0.137	0.002	1.0*	1.0*
Domain: Circuit Simulation														
Freescale2	23,042,677	5,998,698	3.401	190.4	0.013	0.031	1.0*	1.0*	0.013	82.10	0.071	0.012	1.0*	1.0*
circuit5M_dc	19,194,193	7,046,634	2.992	186.1	0.023	0.012	1.0*	1.0*	0.011	88.93	0.096	0.002	1.0*	1.0*
memchip	14,810,202	5,415,048	4.143	199.0	0.022	0.012	1.0*	1.0*	0.015	95.64	0.110	0.003	1.0*	1.0*
Domain: Combinatorial														
GL7d16	14,488,881	1,415,389	2.137	49.11	0.000	0.000	1.000	1.000	0.007	18.81	0.001	0.000	0.964	0.964
Domain: Computational Fluid Dynamics														
atmosmodl	10,319,760	2,979,504	6.372	188.3	0.023	0.007	1.0*	1.0*	0.022	86.08	0.094	0.001	1.0*	1.0*
atmosmodm	10,319,760	2,979,504	6.366	188.1	0.023	0.007	1.0*	1.0*	0.022	84.88	0.098	0.001	1.0*	1.0*
Domain: Graph														
in-2004	16,917,053	2,765,816	3.916	96.23	0.033	0.077	0.973	0.973	0.014	37.99	0.133	0.022	1.0*	1.0*
great-britain_osm	16,313,034	15,467,644	1.553	175.0	0.019	0.006	1.000	1.000	0.006	97.10	0.085	0.001	1.0*	1.0*
venturiLevel3	16,108,474	8,053,638	3.547	214.1	0.019	0.004	1.0*	1.0*	0.012	107.5	0.073	0.001	0.886	0.886
patents	14,970,767	7,549,536	0.981	62.42	0.001	0.001	1.000	1.000	0.003	30.69	0.009	0.000	0.907	0.907
italy_osm	14,027,956	13,372,986	2.038	200.7	0.023	0.008	1.000	1.000	0.008	112.6	0.089	0.002	1.0*	1.0*
rgg_n_2_20_s0	13,783,240	2,097,152	5.877	140.5	0.007	0.002	1.0*	1.0*	0.021	52.47	0.018	0.000	0.939	0.939
hugetrace-00000	13,758,266	9,176,968	1.803	137.5	0.012	0.006	1.000	1.000	0.006	72.48	0.067	0.001	0.990	0.990
delaunay_n21	12,582,816	4,194,304	3.927	142.4	0.017	0.009	1.0*	1.0*	0.014	66.51	0.082	0.002	1.0*	1.0*
kron_g500-logn17	10,228,360	262,144	6.093	59.58	0.004	0.045	1.0*	1.0*	0.023	11.44	0.017	0.012	1.0*	1.0*
Domain: Linear Programming														
tp-6	11,537,419	1,157,053	7.655	92.30	0.016	0.268	1.000	1.0*	0.025	17.53	0.071	0.171	1.0*	1.0*
rail4284	11,284,032	1,101,178	5.169	37.03	0.018	0.375	0.712	0.712	0.021	3.970	0.087	0.132	0.835	0.830
Domain: Materials Problem														
3Dspectralwave2	14,322,744	584,016	8.463	122.4	0.023	0.009	1.000	1.000	0.030	25.68	0.079	0.004	1.0*	1.0*
Domain: Model Reduction														
CurlCurl_3	13,544,618	2,439,148	5.948	165.8	0.021	0.005	1.0*	1.0*	0.020	62.85	0.090	0.003	1.0*	1.0*
Domain: Optimization														
kkt_power	14,612,663	4,126,988	2.771	106.5	0.008	0.014	1.000	1.000	0.010	47.92	0.051	0.003	0.959	0.959
mip1	10,352,819	132,926	9.055	60.79	0.030	0.388	0.755	0.788	0.032	9.327	0.087	0.067	0.780	0.787
Domain: Power Network Problem														
TSOPF_RS_b2383	16,171,169	76,240	6.753	56.26	0.035	0.070	0.690	0.683	0.025	7.140	0.064	0.008	0.681	0.683
TSOPF_FS_b300_c3	13,135,930	168,828	5.640	40.06	0.038	0.272	0.672	0.742	0.019	6.259	0.088	0.054	0.716	0.741
Domain: Structural														
pkustk14	14,836,504	303,852	8.110	86.00	0.023	0.038	0.914	0.914	0.026	15.81	0.103	0.015	0.979	0.979
crankseg_2	14,148,858	127,676	8.572	71.38	0.025	0.047	0.816	0.816	0.029	10.56	0.102	0.017	0.843	0.834
halfb	12,387,821	449,234	9.968	105.8	0.027	0.027	0.863	0.871	0.031	23.82	0.091	0.009	0.872	0.888
troll	11,985,111	426,906	10.49	105.1	0.023	0.030	0.733	0.733	0.034	23.42	0.082	0.009	0.796	0.796
fullb	11,708,077	398,374	9.953	98.09	0.027	0.027	0.861	0.861	0.032	21.40	0.098	0.010	0.874	0.867
pwtk	11,634,424	435,836	13.95	148.9	0.034	0.019	1.0*	1.0*	0.034	24.51	0.090	0.006	0.954	0.968
fcondp2	11,294,316	403,644	10.20	100.3	0.023	0.029	0.702	0.702	0.032	22.04	0.069	0.007	0.732	0.732
bmw3_2	11,288,630	454,724	10.13	100.3	0.025	0.028	0.893	0.897	0.033	23.58	0.102	0.011	0.914	0.904
bmwcra_1	10,644,002	297,540	13.49	111.7	0.024	0.030	0.771	0.771	0.043	22.65	0.096	0.010	0.799	0.799
crankseg_1	10,614,210	105,608	11.44	77.50	0.024	0.050	0.933	0.933	0.039	11.61	0.103	0.019	0.898	0.898
Domain: Theoretical/Quantum Chemistry														
Si41Ge41H72	15,011,265	371,278	8.467	100.4	0.025	0.064	1.0*	1.0*	0.031	19.17	0.113	0.027	1.0*	1.0*
SiO2	11,283,503	310,662	6.888	66.77	0.028	0.263	0.982	0.982	0.036	17.22	0.123	0.095	0.920	0.920
Si87H76	10,661,631	480,738	9.844	112.3	0.022	0.035	1.0*	1.0*	0.035	26.35	0.108	0.015	1.0*	1.0*
Domain: Tomography Problem														
JP	13,734,559	154,936	9.114	91.63	0.015	0.022	1.0*	1.0*	0.031	14.62	0.084	0.009	1.0*	1.0*

Figure A-12: Over the matrices from Suitesparse [10] with between 10 and 17 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			B = 12						B = 4					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
nd24k	28,715,634	144,000	7.769	96.18	0.031	0.016	0.820	0.824	0.026	12.78	0.078	0.002	0.792	0.790
Domain: Circuit Simulation														
FullChip	26,621,990	5,974,024	1.345	76.90	0.019	0.280	1.0*	1.0*	0.005	32.63	0.093	0.141	1.0*	1.0*
rajat31	20,316,253	9,380,004	3.260	258.4	0.014	0.003	1.0*	1.0*	0.013	127.6	0.087	0.001	0.991	0.991
Freescale1	18,920,347	6,857,510	2.376	146.5	0.019	0.011	1.0*	1.0*	0.008	69.28	0.085	0.003	1.0*	0.867
Domain: Combinatorial														
rel9	23,667,183	10,162,717	1.006	127.2	0.008	0.003	1.000	1.000	0.004	74.52	0.046	0.001	1.0*	0.977
Domain: Computational Fluid Dynamics														
StocF-1465	21,005,389	2,930,274	4.700	163.2	0.022	0.009	1.0*	1.0*	0.016	57.71	0.094	0.003	1.0*	1.0*
Domain: Computer Vision														
bundle_adj	20,208,051	1,026,702	2.195	32.37	0.025	0.095	0.777	0.777	0.008	8.444	0.074	0.023	0.688	0.688
Domain: Electromagnetics														
dielFilterV3clx	32,886,208	840,816	3.676	93.07	0.022	0.024	1.0*	1.0*	0.012	18.17	0.100	0.007	1.0*	1.0*
dielFilterV2clx	25,309,272	1,214,464	4.390	108.0	0.021	0.015	1.0*	1.0*	0.015	26.07	0.103	0.005	1.0*	1.0*
gsm_106857	21,758,924	1,178,892	1.865	45.51	0.013	0.016	1.000	1.000	0.006	11.36	0.072	0.004	1.0*	1.0*
fem_hifreq_circuit	20,239,237	982,200	4.675	94.36	0.016	0.013	0.870	0.870	0.017	23.54	0.070	0.004	0.892	0.892
Domain: Graph														
packing-500x100x100-b050	34,976,486	4,291,704	3.004	168.3	0.024	0.005	1.0*	1.0*	0.011	56.91	0.118	0.002	1.0*	1.0*
coPapersCiteseer	32,073,440	868,204	3.572	72.49	0.028	0.056	1.0*	1.0*	0.011	14.35	0.105	0.019	1.0*	1.0*
coPapersDBLP	30,491,458	1,080,972	2.587	56.79	0.025	0.036	0.907	0.907	0.008	12.40	0.102	0.013	1.0*	1.0*
mouse_gene	28,967,291	90,202	4.719	81.16	0.013	0.066	1.000	1.000	0.017	9.417	0.050	0.019	0.842	0.842
adaptive	27,248,640	13,631,488	1.548	173.2	0.018	0.003	1.000	1.000	0.006	88.24	0.079	0.000	1.0*	1.0*
cage14	27,130,349	3,011,570	3.506	149.1	0.018	0.012	1.000	1.000	0.013	48.73	0.078	0.003	1.0*	1.0*
asia_osm	25,423,206	23,901,514	1.415	231.0	0.022	0.006	1.0*	1.0*	0.006	129.2	0.088	0.001	0.870	0.870
delaunay_n22	25,165,738	8,388,608	2.076	149.0	0.018	0.007	1.0*	1.0*	0.007	68.04	0.087	0.001	1.0*	1.0*
NLR	24,975,952	8,327,526	1.242	99.59	0.008	0.004	1.000	1.000	0.004	43.69	0.059	0.001	0.986	0.986
germany_osm	24,738,362	23,097,690	1.067	172.6	0.019	0.005	1.000	1.000	0.004	95.58	0.075	0.001	1.0*	1.0*
human_gene1	24,669,643	44,566	6.897	91.53	0.019	0.150	1.000	1.000	0.026	11.08	0.076	0.045	1.0*	1.0*
AS365	22,736,152	7,598,550	1.377	99.83	0.008	0.004	1.000	1.000	0.005	44.12	0.055	0.001	0.999	0.999
12month1	22,624,727	885,093	4.268	58.62	0.013	0.237	1.000	1.000	0.016	6.608	0.059	0.077	1.0*	1.0*
333SP	22,217,266	7,425,630	1.529	101.7	0.013	0.011	1.000	1.000	0.005	46.92	0.077	0.003	0.998	0.998
as-Skitter	22,190,596	3,392,830	1.330	44.02	0.012	0.142	1.0*	1.0*	0.005	16.74	0.071	0.057	0.690	0.680
hugetric-00020	21,361,554	14,245,584	1.031	121.0	0.010	0.005	1.0*	1.0*	0.004	61.94	0.055	0.001	0.957	0.957
M6	21,003,872	7,003,552	1.505	100.7	0.009	0.004	1.000	1.000	0.005	44.72	0.057	0.001	0.962	0.962
hugetric-00010	19,771,708	13,185,530	1.125	122.9	0.010	0.005	1.0*	1.0*	0.004	63.05	0.056	0.001	0.758	0.758
eu-2005	19,235,140	1,725,328	5.396	117.9	0.027	0.047	1.0*	1.0*	0.019	37.40	0.114	0.015	1.0*	1.0*
human_gene2	18,068,388	28,680	9.284	91.50	0.019	0.163	1.0*	1.0*	0.035	11.32	0.087	0.060	1.0*	1.0*
hugetric-00000	17,467,046	11,649,108	1.996	193.6	0.014	0.005	1.000	1.000	0.008	100.6	0.065	0.001	1.0*	1.0*
Domain: Materials Problem														
3Dspectralwave	33,650,589	1,361,886	4.150	134.4	0.021	0.004	1.0*	1.0*	0.015	28.32	0.080	0.002	1.0*	1.0*
Domain: Model Reduction														
CurlCurl_4	26,515,867	4,761,030	3.243	176.9	0.021	0.003	1.0*	1.0*	0.012	67.39	0.096	0.002	1.0*	1.0*
Domain: Optimization														
nlpkkt80	28,704,672	2,124,800	3.811	132.5	0.026	0.007	1.0*	1.0*	0.013	37.65	0.117	0.002	1.0*	1.0*
Domain: Structural														
Fault_639	28,614,564	1,277,604	4.559	113.4	0.021	0.012	0.786	0.786	0.016	28.15	0.088	0.004	0.774	0.774
ML_Laplace	27,689,972	754,004	5.259	106.2	0.029	0.013	0.804	0.804	0.016	21.34	0.086	0.002	0.792	0.792
F1	26,837,113	687,582	2.921	54.36	0.018	0.018	0.698	0.698	0.009	10.97	0.085	0.006	0.659	0.659
Transport	23,500,731	3,204,222	4.006	160.2	0.026	0.005	1.0*	1.0*	0.014	56.02	0.119	0.001	1.0*	1.0*
CoupCons3D	22,322,336	833,600	5.297	101.4	0.023	0.014	0.728	0.728	0.018	23.24	0.057	0.003	0.726	0.726
msdoor	20,240,935	831,726	5.889	106.5	0.024	0.031	1.0*	1.0*	0.020	25.17	0.099	0.008	1.0*	1.0*
af_shell1	17,588,875	1,009,710	6.674	118.7	0.025	0.007	0.784	0.784	0.023	31.85	0.091	0.004	0.992	0.987
af_0_k101	17,550,675	1,007,250	6.662	116.7	0.026	0.007	0.797	0.797	0.023	31.95	0.088	0.004	0.947	0.941
Domain: Theoretical/Quantum Chemistry														
Ga41As41H72	18,488,476	536,192	6.781	103.3	0.025	0.052	1.0*	1.0*	0.025	21.07	0.123	0.022	1.0*	1.0*

Figure A-13: Over the matrices from Suitesparse [10] with between 17 and 35 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			$B = 12$						$B = 4$					
			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)		Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI	PHIL	OSKI
Domain: 2D/3D Problem														
PFlow_742	37,138,461	1,485,586	3.561	116.4	0.027	0.008	1.0*	1.0*	0.012	26.12	0.100	0.003	1.0*	1.0*
Domain: Circuit Simulation														
circuit5M	59,524,291	11,116,652	0.582	55.80	0.020	0.345	1.0*	1.0*	0.002	22.68	0.102	0.178	1.0*	1.0*
Domain: Computational Fluid Dynamics														
RM07R	37,464,962	763,378	4.020	105.2	0.022	0.018	1.0*	1.0*	0.012	18.69	0.095	0.012	1.0*	1.0*
Domain: Electromagnetics														
dielFilterV3real	89,306,020	2,205,648	1.411	92.23	0.022	0.013	1.0*	1.0*	0.004	17.74	0.093	0.004	1.0*	1.0*
dielFilterV2real	48,538,952	2,314,912	2.298	105.5	0.021	0.011	1.0*	1.0*	0.008	25.55	0.093	0.004	0.917	0.911
Domain: Graph														
channel-500x100x100-b050	85,362,744	9,604,000	1.375	175.6	0.025	0.004	1.0*	1.0*	0.005	56.68	0.109	0.000	1.0*	1.0*
wb-edu	57,156,537	19,691,450	1.604	218.0	0.022	0.026	1.0*	1.0*	0.006	103.6	0.087	0.010	1.0*	1.0*
delaunay_n23	50,331,568	16,777,216	1.134	154.2	0.018	0.005	1.0*	1.0*	0.004	69.74	0.081	0.001	1.0*	1.0*
Domain: Linear Programming														
spal_004	46,168,124	331,899	3.238	60.52	0.015	0.026	0.967	0.967	0.012	6.616	0.062	0.008	0.957	0.943
Domain: Model Reduction														
bone010	71,666,325	1,973,406	2.205	112.6	0.028	0.006	0.783	0.783	0.006	22.48	0.094	0.001	0.779	0.779
boneS10	55,468,422	1,829,796	2.668	112.4	0.027	0.009	0.809	0.809	0.009	24.07	0.084	0.002	0.782	0.782
Domain: Optimization														
nlpkkt120	96,845,792	7,084,800	1.271	146.6	0.027	0.004	1.0*	1.0*	0.005	45.36	0.126	0.001	1.0*	1.0*
Domain: Structural														
Long_Coup_dt0	87,088,992	2,940,304	1.672	117.4	0.020	0.007	0.765	0.765	0.006	25.37	0.065	0.002	0.802	0.802
audikw_1	77,651,847	1,887,390	1.527	76.98	0.019	0.009	0.819	0.819	0.005	14.91	0.083	0.003	0.822	0.822
Serena	64,531,701	2,782,698	2.307	120.5	0.020	0.007	0.899	0.899	0.008	29.19	0.082	0.002	0.840	0.840
Geo_1438	63,156,690	2,875,920	2.206	120.8	0.020	0.007	0.864	0.864	0.008	32.34	0.086	0.002	0.828	0.828
Hook_1498	60,917,445	2,996,046	2.377	122.3	0.019	0.007	0.904	0.904	0.008	31.23	0.090	0.002	0.805	0.805
af_shell10	52,672,325	3,016,130	2.477	127.7	0.024	0.004	0.852	0.852	0.009	34.77	0.082	0.002	1.0*	1.0*
ldoor	46,522,475	1,904,406	2.755	109.2	0.023	0.011	0.761	0.761	0.010	25.82	0.098	0.005	0.995	0.993
Emilia_923	41,005,206	1,846,272	3.405	120.4	0.020	0.010	0.815	0.815	0.012	29.51	0.085	0.003	0.821	0.821
inline_1	36,816,342	1,007,424	3.039	78.05	0.020	0.013	0.748	0.748	0.010	15.95	0.084	0.005	0.703	0.703

Figure A-14: Over the matrices from Suitesparse [10] with between 35 and 100 million nonzeros, we report the results of the fixed-parameter study. Chapter 5 provides details about the experimental setup and measurements.

Matrix Information			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	Ser	Par	Ser	Par	Ser	Par
Domain: 2D/3D Problem								
heart1	1,387,773	7,114	86.16	26.04	0.020	0.062	0.794	0.576
torso2	1,033,473	231,934	79.64	28.26	0.033	0.109	1.0*	1.0*
Dubcova2	1,030,225	130,050	80.57	28.95	0.020	0.061	1.000	1.0*
Domain: Chemical Process Simulation								
lhr71	1,528,092	140,608	76.66	26.09	0.028	0.101	1.0*	1.0*
std1_Jac3	1,455,848	43,964	61.52	17.49	0.030	0.096	1.0*	0.872
std1_Jac2	1,248,731	43,964	60.48	15.85	0.028	0.090	0.833	0.833
Domain: Circuit Simulation								
ASIC_320ks	1,827,807	643,342	30.95	9.749	0.020	0.069	1.000	1.0*
Raj1	1,302,464	527,486	55.88	19.24	0.019	0.061	1.0*	1.0*
Domain: Combinatorial Problem								
n4c6-b10	1,456,422	318,960	56.64	19.93	0.018	0.056	1.000	1.0*
relat8	1,334,038	358,035	61.50	22.39	0.010	0.029	1.000	1.0*
n4c6-b7	1,305,720	267,330	57.21	20.45	0.017	0.061	1.000	0.850
IG5-17	1,035,008	58,106	98.17	30.44	0.012	0.041	1.0*	0.959
Domain: Computational Fluid Dynamics Problem								
raefsky3	1,488,768	42,400	89.98	37.27	0.024	0.052	0.598	0.664
ex11	1,096,948	33,228	106.9	32.00	0.031	0.105	1.0*	1.0*
rim	1,014,951	45,120	120.8	36.75	0.022	0.068	1.0*	1.0*
Domain: Counter Example Problem								
denormal	1,156,224	178,800	100.9	33.95	0.027	0.088	1.0*	1.0*
Domain: Economic Problem								
mac_econ_fwd500	1,273,389	413,000	50.49	18.56	0.014	0.045	1.000	0.998
Domain: Electromagnetics Problem								
vfem	1,434,636	186,952	51.30	14.13	0.021	0.072	1.000	0.676
pli	1,350,309	45,390	96.50	35.11	0.029	0.062	1.0*	1.0*
Domain: Frequency Domain Circuit Simulation								
twotone	1,224,224	241,500	87.85	28.42	0.016	0.051	1.000	0.958
Domain: Graph								
web-NotreDame	1,497,134	651,458	32.19	10.12	0.021	0.074	1.0*	1.0*
598a	1,483,868	221,942	33.53	11.32	0.005	0.016	1.000	1.0*
NotreDame_actors	1,470,404	520,223	15.11	6.311	0.007	0.015	1.000	0.933
rgg_n_2_17_s0	1,457,506	262,144	39.38	12.44	0.010	0.036	1.0*	0.699
ga2010	1,418,056	582,172	29.56	9.758	0.007	0.023	1.000	1.0*
nc2010	1,416,620	577,974	34.63	11.38	0.007	0.025	1.000	1.0*
va2010	1,402,128	571,524	27.16	9.227	0.006	0.024	1.0*	0.920
fe_rotor	1,324,862	199,234	56.18	22.64	0.014	0.030	1.0*	0.998
in2010	1,281,716	534,142	37.64	13.43	0.008	0.024	1.0*	1.0*
ok2010	1,274,148	538,236	37.79	12.40	0.006	0.021	1.0*	1.0*
amazon0302	1,234,877	524,222	28.71	12.34	0.009	0.017	1.000	0.918
al2010	1,230,482	504,532	31.06	10.44	0.006	0.021	1.000	0.909
mn2010	1,227,102	519,554	39.36	13.10	0.008	0.027	1.000	1.0*
caidaRouterLevel	1,218,132	384,488	20.94	7.695	0.005	0.015	1.000	0.987
language	1,216,334	798,260	26.04	10.57	0.014	0.039	1.000	0.879
wi2010	1,209,404	506,192	39.45	13.22	0.008	0.030	1.0*	0.993
Linux_call_graph	1,208,908	648,170	31.99	12.92	0.010	0.020	1.000	1.0*
az2010	1,196,094	483,332	30.77	10.49	0.006	0.020	1.0*	0.916
tn2010	1,193,966	480,232	31.69	10.43	0.007	0.025	1.0*	0.782
connectus	1,127,525	395,304	40.31	10.06	0.019	0.054	1.0*	1.0*
ks2010	1,121,798	477,200	33.32	11.24	0.008	0.028	1.0*	0.791
vsp_finan512_scagr7-2c_rlfddd	1,104,040	279,504	20.91	6.809	0.012	0.045	1.0*	0.580
ia2010	1,021,170	432,014	42.98	14.50	0.008	0.030	1.000	1.0*
G_n_pin_pout	1,002,396	200,000	43.53	13.44	0.006	0.021	1.000	1.0*

Figure A-15: We compared the serial and parallel implementation of PHIL on a subset of the matrices between 1 and 1.5 million nonzeros. Both were run with the same default parameters of $B = 12, \epsilon = 3, \delta = 0.01$.

Matrix Information			Normalized Time to Estimate Fill		Mean Maximum Relative Error		Normalized TACO SpMV Time (Vuduc et al. Model)	
Name	NNZ (k)	Size (m + n)	Ser	Par	Ser	Par	Ser	Par
Domain: Least Squares								
Maragal_8	1,308,415	108,289	19.72	6.122	0.016	0.048	1.000	0.950
Maragal_7	1,200,537	73,409	17.63	5.311	0.020	0.070	0.876	0.946
landmark	1,151,232	74,656	78.80	28.21	0.027	0.086	0.816	1.0*
Domain: Linear Programming								
lp_osa_60	1,408,073	253,526	17.89	6.664	0.017	0.037	1.000	1.0*
dbir2	1,158,159	64,783	36.15	10.14	0.024	0.069	1.0*	0.637
pds-100	1,096,002	670,820	36.81	12.94	0.004	0.014	1.000	0.689
dbic1	1,081,843	269,517	36.82	13.14	0.014	0.047	1.0*	0.813
dbir1	1,077,025	64,579	42.62	11.83	0.022	0.076	1.0*	1.0*
ts-palko	1,076,903	69,237	74.82	21.47	0.014	0.047	1.000	1.0*
watson_1	1,055,093	588,147	53.56	20.90	0.018	0.054	1.000	1.0*
nemsemm1	1,053,986	79,297	122.9	33.47	0.027	0.085	0.737	0.652
pds-90	1,014,136	618,271	37.27	12.93	0.004	0.012	1.0*	0.973
Domain: Materials Problem								
xenon1	1,181,120	97,200	106.2	33.79	0.017	0.053	0.815	1.0*
viscorocks	1,162,244	75,524	106.1	35.96	0.027	0.083	0.865	1.0*
Domain: Model Reduction Problem								
windscreen	1,482,390	45,384	66.74	21.47	0.031	0.102	0.808	0.770
gyro	1,021,159	34,722	126.4	45.83	0.020	0.043	0.607	1.0*
Domain: Optimization								
net75	1,489,200	46,240	45.35	15.19	0.021	0.072	0.966	1.0*
c-73	1,279,274	338,844	22.30	7.458	0.019	0.067	1.000	1.0*
boyd1	1,211,231	186,558	26.46	7.715	0.028	0.088	0.957	0.899
Domain: Power Network Problem								
TSOPF_RS_b300_c1	1,474,325	29,076	48.99	15.33	0.043	0.153	0.576	0.534
hvdc2	1,347,273	379,720	55.68	18.54	0.018	0.063	1.0*	1.0*
TSOPF_RS_b39_c30	1,079,986	120,196	58.85	20.98	0.030	0.099	0.762	0.744
case39	1,042,160	80,432	38.62	12.60	0.031	0.101	0.698	0.727
Domain: Semiconductor Device Problem								
matrix_9	2,121,550	206,860	53.68	17.64	0.024	0.084	0.723	0.775
Domain: Structural								
bcsstk35	1,450,163	60,474	93.48	28.79	0.023	0.077	0.826	0.722
raefsky4	1,328,611	39,558	90.37	26.47	0.027	0.083	0.980	0.673
msc10848	1,229,778	21,696	92.13	26.37	0.021	0.067	0.593	0.854
bcsstk31	1,181,416	71,176	100.7	34.92	0.025	0.053	1.0*	1.0*
msc23052	1,154,814	46,104	108.5	34.46	0.024	0.073	1.0*	0.945
bcsstk36	1,143,140	46,104	91.35	27.63	0.028	0.090	0.849	0.914
bcsstk37	1,140,977	51,006	98.32	29.61	0.030	0.092	0.927	0.730
dawson5	1,010,777	103,074	94.37	28.15	0.026	0.078	0.981	0.876
Domain: Subsequent Theoretical/Quantum Chemistry Problem								
nemeth21	1,173,746	19,012	137.6	46.46	0.025	0.054	0.952	0.942
Domain: Theoretical/Quantum Chemistry								
nemeth22	1,358,832	19,012	123.5	34.72	0.021	0.072	0.922	0.904
SiO	1,317,655	66,802	74.55	23.28	0.022	0.075	1.0*	1.0*
Domain: Thermal Problem								
thermomech_dM	1,423,116	408,632	27.75	9.780	0.008	0.025	1.0*	0.867

Figure A-16: We compared the serial and parallel implementation of PHIL on the remaining matrices between 1 and 1.5 million nonzeros. Both were run with the same default parameters of $B = 12, \epsilon = 3, \delta = 0.01$.

Bibliography

[1] Peter Ahrens, Helen Xu, and Nicholas Schiefer. A fill estimation algorithm for sparse matrices and tensors in blocked formats. In *International Parallel and Distributed Processing Symposium*. IEEE, 2018. To appear.

[2] Brett W. Bader and Tamara G. Kolda. Efficient MATLAB computations with sparse and factored tensors. *SIAM Journal on Scientific Computing*, 30(1):205–231, January 2008.

[3] Rémi Bardenet and Odalric-Ambrym Maillard. Concentration inequalities for sampling without replacement. *Bernoulli*, 21(3):1361–1385, August 2015.

[4] Guy E Blelloch. Prefix sums and their applications. 1990.

[5] Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: An efficient multithreaded runtime system. *Journal of parallel and distributed computing*, 37(1):55–69, 1996.

[6] Aydin Buluç, Jeremy T. Fineman, Matteo Frigo, John R. Gilbert, and Charles E. Leiserson. Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. page 233. ACM Press, 2009.

[7] Alfredo Buttari, Victor Eijkhout, Julien Langou, and Salvatore Filippone. Performance optimization and modeling of blocked sparse kernels. *The International Journal of High Performance Computing Applications*, 21(4):467–484, November 2007.

[8] Justus A. Calvin, Cannada A. Lewis, and Edward F. Valeev. Scalable task-based algorithm for multiplication of block-rank-sparse matrices. pages 1–8. ACM Press, 2015.

[9] Jee W. Choi, Amik Singh, and Richard W. Vuduc. Model-driven autotuning of sparse matrix-vector multiply on GPUs. *ACM SIGPLAN Notices*, 45(5):115, May 2010.

[10] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1–25, November 2011.

[11] Jack Dongarra and Victor Eijkhout. Self-Adapting Numerical Software for Next Generation Applications. *The International Journal of High Performance Computing Applications*, 17(2):125–131, May 2003.

[12] Wassily Hoeffding. Probability Inequalities for Sums of Bounded Random Variables. *Journal of the American Statistical Association*, 58(301):13, March 1963.

[13] Eun-Jin Im. *Optimizing the Performance of Sparse Matrix-Vector Multiplication*. PhD thesis, EECS Department, University of California, Berkeley, June 2000.

[14] Eun-Jin Im and Katherine A. Yelick. Optimizing sparse matrix computations for register reuse in SPARSITY. In G. Goos, J. Hartmanis, J. van Leeuwen, Vassil N. Alexandrov, Jack J. Dongarra, Benjoe A. Juliano, RenÃľ S. Renner, and C. J. Kenneth Tan, editors, *Computational Science âĂŤ ICCS 2001*, volume 2073, pages 127–136. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[15] Eun-Jin Im, Katherine A. Yelick, and Richard W. Vuduc. Sparsity: Optimization framework for sparse matrix kernels. *International Journal of High Performance Computing Applications*, 18(1):135–158, February 2004.

[16] Vasileios Karakasis, Georgios Goumas, and Nectarios Koziris. A comparative study of blocking storage methods for sparse matrices on multicore architectures. pages 247–256. IEEE, 2009.

[17] Vasileios Karakasis, Georgios Goumas, and Nectarios Koziris. Performance models for blocked sparse matrix-vector multiplication kernels. In *2009 International Conference on Parallel Processing*, pages 356–364, September 2009.

[18] Fredrik Kjolstad, Shoaib Kamil, Stephen Chou, David Lugato, and Saman Amarasinghe. The tensor algebra compiler. *Proc. ACM Program. Lang.*, 1(OOPSLA):77:1–77:29, October 2017.

[19] Daniel Langr, Ivan Šimeček, and Tomáš Dytrych. Block Iterators for Sparse Matrices. pages 695–704, October 2016.

[20] Jiajia Li, Guangming Tan, Mingyu Chen, and Ninghui Sun. SMAT: an input adaptive auto-tuner for sparse matrix-vector multiplication. *ACM SIGPLAN Notices*, 48(6):117, June 2013.

[21] Rajesh Nishtala, Richard W. Vuduc, James W. Demmel, and Katherine A. Yelick. When cache blocking of sparse matrix vector multiply works and why. *Applicable Algebra in Engineering, Communication and Computing*, 18(3):297–311, May 2007.

[22] A. Pinar and M. T. Heath. Improving performance of sparse matrix-vector multiplication. In *Supercomputing, ACM/IEEE 1999 Conference*, page 30, November 1999.

[23] Tao B. Schardl, William S. Moses, and Charles E. Leiserson. Tapir: Embedding fork-join parallelism into llvm's intermediate representation. *SIGPLAN Not.*, 52(8):249–265, January 2017.

[24] Shaden Smith and George Karypis. Tensor-matrix products with a compressed sparse tensor. pages 1–7. ACM Press, 2015.

[25] Shaden Smith, Niranjay Ravindran, Nicholas D. Sidiropoulos, and George Karypis. SPLATT: Efficient and parallel sparse tensor-matrix multiplication. pages 61–70. IEEE, May 2015.

[26] Richard W. Vuduc. *Automatic performance tuning of sparse matrix kernels*. PhD thesis, University of California, Berkeley, CA, USA, January 2004.

[27] Richard W. Vuduc, James W. Demmel, and Katherine A. Yelick. OSKI: A library of automatically tuned sparse matrix kernels. *Journal of Physics: Conference Series*, 16:521–530, January 2005.

[28] Richard W. Vuduc, James W. Demmel, Katherine A. Yelick, Shoaib Kamil, Rajesh Nishtala, and Benjamin C. Lee. Performance optimizations and bounds for sparse matrix-vector multiply. page 26. IEEE, 2002.

[29] Samuel Williams, Leonid Oliker, Richard W. Vuduc, John Shalf, Katherine A. Yelick, and James W. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Parallel Computing*, 35(3):178–194, March 2009.

[30] Albert-Jan N. Yzelman. Generalised vectorisation for sparse matrix-vector multiplication. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, IA^3 '15, pages 6:1–6:8, New York, NY, USA, 2015. ACM.