The Pennsylvania State University

The Graduate School

Department of Computer Science and Engineering

ANALYZING THE BENEFITS OF SCRATCHPAD MEMORIES FOR SCIENTIFIC MATRIX COMPUTATIONS

A Thesis in

Computer Science and Engineering

by

Bryan Alan Cover

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

May 2008

The thesis of Bryan Alan Cover was reviewed and approved* by the following:

Mary Jane Irwin
Evan Pugh Professor of Computer Science and Engineering
Thesis Co-Advisor

Padma Raghavan
Professor of Computer Science and Engineering
Thesis Co-Advisor

Raj Acharya
Professor of Computer Science and Engineering
Head of the Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

Scratchpad memories (SPMs) have been shown to be more energy efficient, have faster access times, and take up less area than traditional hardware-managed caches. This, coupled with the predictability of data presence and reduced thermal properties, makes SPMs an attractive alternative to cache for many scientific applications. In this work, SPM based systems are considered for a variety of different functions. The first study performed is to analyze SPMs for their thermal and area properties on a conventional RISC processor. Six performance optimized variants of architecture are explored that evaluate the impact of having an SPM in the on-chip memory hierarchy. Increasing the performance and energy efficiency of both dense and sparse matrix-vector multiplication on a chip multi-processor are also looked at. The efficient utilization of the SPM is ensured by profiling the application for the data structures which do not perform well in traditional cache. The impact of using an SPM at all levels of the on-chip memory hierarchy is evaluated through three SPM based architectures.

When looking at the chip layout, the results show an average decrease in the average component temperature of a chip by as much as 1.6%. The total area of a chip can be reduced by nearly 30%. The dense matrix-vector multiplication kernel showed as much as a 60% increase in performance and a decrease in the energy-delay product by as much as 84%. For the sparse matrix-vector multiplication kernel, the experimental results show an increase in performance by as much as 41% and a decrease in the om-chip

energy-delay product by as much as 67%. This depends on which level of the hierarchy the SPM is utilized and the specific sparse matrix being multiplied.

Table of Contents

List of Tables

List of Figures

Acknowledgments

I would like to thank my family and friends for their support through all of my life, especially during my education here at Penn State University.

I would also like to thank Dr. Irwin, Dr. Raghavan, and the other professors and colleagues who helped me during my research by providing much needed advice. My opportunities to take part in their research groups were very rewarding.

Chapter 1

Introduction

The high-performance community is migrating towards using commodity hardware to take advantage of the ever increasing performance of those systems. Some recent studies have proposed additional changes to existing commodity systems to increase the performance of scientific computation kernels [36]. One of the main concerns in using commodity systems is the sometimes poor performance of conventional hardware-managed caches in such systems. For example, the erratic access patterns of many sparse scientific applications, coupled with phenomena such as cache pollution, can lead to poor performance from the memory hierarchy.

Caches are temporary on-chip storage structures that allow a low latency access to frequently utilized data. What gets put into and taken out of the cache is determined by hardware on the processor. The hardware will usually populate the cache with data that has been recently used or with nearby data to try and predict usage. The majority of today's modern processors have a multi-level on-chip cache hierarchy to provide the fastest access to data with the minimum number of misses. The cache in a system is typically made using Static Random Access Memory (SRAM). SRAM allows the data to be stored without having to continually refresh the data's value like some other memory technologies require. A one bit memory cell will typically be stored using four CMOS transistors and having another two for reading and writing to the memory cell. Further,

cache memories can have different levels of data associativity, which can affect their effectiveness. Memory can be considered fully associative in which data can be placed anywhere in the cache, direct mapped in which data can only go to one particular spot in the cache, or it may be some level in between these two. Generally, the greater the associativity of a memory, the slower it will perform since hardware must check more locations in memory.

Scratchpad memories (SPMs) are an alternate way to ensure a low latency access to data. SPMs are fast, on-chip memories that are managed by software instead of by hardware [2; 3; 26]. If the SPM is used properly, software can be tuned to ensure that a search for data in memory always results in a hit. This can be done by having predictable data being placed in the SPM before it is used. Data movement can be controlled by three main ways. The first is to have the assembly programmer directly control the SPM using the processor's Instruction Set Architecture. The second way is similar in that a programmer using a higher level language would specify what should be put in it. The compiler will then properly interpret the code using libraries. The last way is to have the compiler analyze the program and add SPM instructions where ever it wants to control accessed data. The basic structure of SPMs remains the same to that of traditional, fully associative cache. Data can be placed anywhere in the cache, but the software can be made aware of where the data has been placed so a search of the entire memory is not necessary. To simplify the software required and the applications run on them, SPMs have been studied extensively in the context of embedded systems and real time applications. Each memory cell in an SPM will use the same 6 transistor layout. The main difference is that the rest of the scratchpad will be much simpler since the

memory is software controlled. SPMs have no comparator, signal hit/miss acknowledging circuitry, tag array, hardware to maintain coherency, or hardware to predict what data to load and keep in memory [2; 3; 6; 7].

Both caches and SPMs have advantages and disadvantages. SPMs require additional programmer knowledge or compiler support to function properly. In addition, caches are the only viable choice when the working set size varies or is unpredictable, where the working set is the data that is being used by a particular process. However, caches suffer from pollution and low data predictability [13]. Cache pollution results from unnecessary data being loaded into the cache, causing possibly needed data to be evicted to lower levels of the memory hierarchy. When data can't be predicted correctly, many unnecessary cache misses and accesses to main memory may result. Because caches must employ extra hardware to ensure data integrity and prediction, they are not the ideal choice for energy efficient on-chip memory [10; 29; 32; 33].

The total energy required by a processor is made up of the total leakage energy and dynamic energy. The leakage energy consumed by every component on a processor's die is dependent on the number of transistors the hardware uses. The proportion of leakage energy required is increasing as transistors are made using smaller process technologies which reduce the amount of material used to insulate the charge at a transistor. The leakage current is always present in the operation of a chip as long as the transistors are not switched off. This means that one way to reduce the energy required for an application due to leakage energy is to reduce its run time. Other ways would include cutting the transistor count and turning off portions of the chip. Unlike leakage energy, dynamic energy is only consumed when a transistor switches. This can occur whenever a

memory location is read or written, for instance. By having less unnecessary or wasteful reads or writes to memory, an application would have lower energy requirements due to dynamic energy considerations.

Traditionally, scratchpads have been proposed as an application specific memory where the working set can be profiled statically and an ideal size for the SPM estimated based on the same. While most general purpose processors have only cache for their on-chip storage, new SPM based architectures are being considered. For example, the IBM Cell Processor [17] uses scratchpads for its local storage. Thus, it is necessary to evaluate the impact of having an SPM as a component of the on-chip memory hierarchy.

Limitations in performance and power consumption of single core architectures have driven general purpose processors to contain multiple cores on a single die. The power consumption of a chip multi-processor is still a concern and many techniques, both architectural and at the software level, have been proposed to reduce the power consumption of a multi-core chip [21]. This work evaluates an SPM being present in the multi-core domain.

Along with an increase in the number of cores, the total on-chip memory capacity is increasing. Thus, reducing the power and energy consumption of the memory sub-system is an important task that will have profound effects for the entire chip. The relatively low power consumption of an SPM when compared to a traditional cache makes it an important alternative for on-chip memory. Likewise, an SPM will generate much less heat than a traditional cache since it will consume less power. When a processor runs at a cooler temperature, it will have much lower cooling costs, greater reliability, higher performance, lower leakage power, and is more environmentally conscious [27].

In addition, scratchpads allow for the shrinking of processor sizes. This can have many benefits including the possibility of adding more memory to the processor. Also, a smaller processor will mean that it is less prone to manufacturing failure including timing problems and increasing yield overall. In addition, a smaller processor, paired with a smaller area of memory, can reduce the probability of experiencing a soft error from a particle strike [24; 27; 34].

With these trends occurring in the microprocessor industry, the use of scratchpads on a RISC processor based system is also evaluated to achieve a processor that is both cooler and smaller than one using traditional cache. The use of scratchpads on a multi-core system is also looked at to achieve better performance and better energy consumption. In this work, the performance and energy benefits of scratchpad memories will be evaluated in a multi-core system for Dense Matrix-Vector (DMV) and Sparse Matrix-Vector (SPMV) multiplication. Both keeping a fixed problem size as the number of cores grow larger and increasing the problem size to keep the isoefficiency relatively equal are looked at.

Matrix-vector multiplication is an important routine in solving systems of linear equations using iterative methods [11]. Also, standard discretizations of partial differential equations typically lead to large sparse matrices. Thus, improving the performance of the DMV and SPMV multiplication routines would also improve several scientific computation kernels. Sparse matrices are matrices that have zeros for the majority of their elements, while dense matrices have values for most of their elements. One way to take advantage of sparsity in a matrix is to have data structures which minimize the storage of the matrix. Several such data structures have been proposed in literature [12; 28].

However, many of these data structures lead to poor performance on a cache based system, as they usually have a level of indirection. In this work, the Compressed Sparse Row (CSR) format is used to store the sparse matrices [8; 11]. Sparse matrices can be categorized into either unstructured or structured matrices [8; 11; 28]. An unstructured matrix would consist of non-zero elements not being confined to any part of the matrix and not forming any readily noticeable pattern. In structured matrices, the non-zero pattern can be exploited to speed up computations or reduce memory requirements. An example of a structured matrix is one where all the non-zero elements are confined within a band around the principal diagonal. For this study, reordered structured matrices are used for the fixed problem size SPMV multiplication. The other experiments utilized matrices that had a natural structure. For the reordering, the Reverse Cuthill McKee (RCM) reordering of a sparse matrix was taken and stored using CSR format. An example of such a reordering is shown in Figures 1.1 and 1.2. Figure 1.1 shows the original ordering of the msc23052 matrix. Figure 1.2 shows the RCM reordering that was used for this study.

To quantify the benefits of SPMs over conventional caches, SPMs are evaluated in all levels of the on-chip memory hierarchy. Six performance optimized variants of processor architecture are explored for temperature and area layout analysis. To perform the DMV and SPMV multiplication analysis, only four architectures were explored due to the memory requirements of the algorithms.

The experiments indicate that when the SPM is part of the Level 1 hierarchy, the total area of the chip will decrease by 1.07% and the average temperature can decrease by as much as 1.59%. Further, when an SPM is used in the Level 2 hierarchy, the area

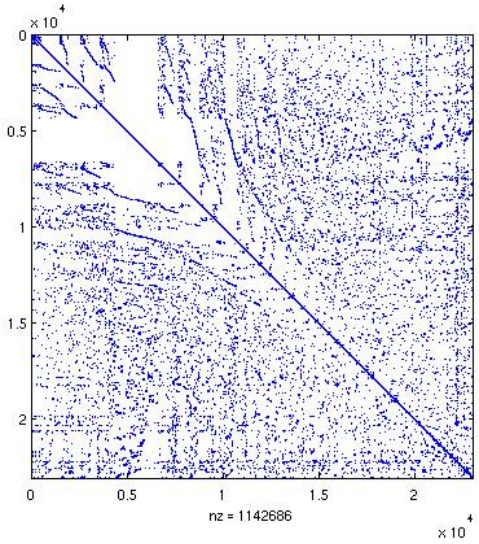


Figure 1.1: Original msc23052 matrix ordering.

will decrease by 28.61% and the average temperature drops by 0.9%. Lastly, if SPMs are present at all levels of the on-chip memory hierarchy, the area will decrease by 29.84% and the average temperature drops by 1.22%. The dense matrix-vector multiplication kernel showed as much as a 60% increase in performance and a decrease in the energy-delay product by as much as 84%. For the sparse matrix-vector multiplication kernel, the experimental results show an increase in performance by as much as 41% and a decrease in the on-chip energy-delay product by as much as 67%. This depends on which level of the hierarchy the SPM is utilized and the specific sparse matrix being multiplied. This is the first study of the SPM performance of matrix-vector multiplication in the context of Chip Multi-Processors (CMPs).

The rest of this paper is organized as follows. Chapter 2 discusses related work to this study. Chapter 3 describes the preliminaries, the architectures explored, and the experimental setup of this work. Also discussed are topics such as application profiling

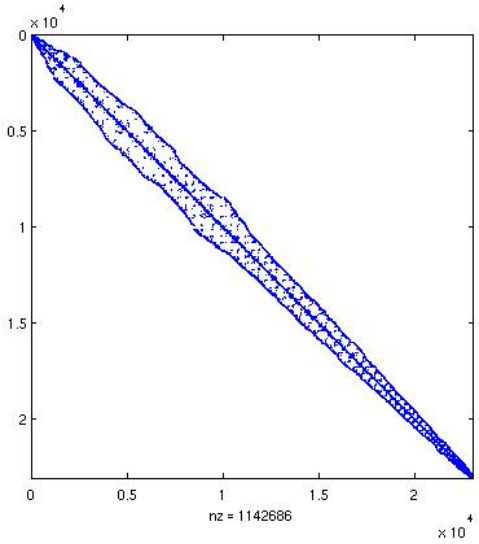


Figure 1.2: RCM reordered msc23052 matrix.

and the data staging algorithm. Chapter 4 presents and discusses the results. I end the study with concluding remarks and a discussion of possible future work in Chapter 5.

Chapter 2

Background of Previous Research

The basic definition of a scratchpad memory is that it is a software-controlled cache. By using specific load and store commands, a programmer or compiler can explicitly state what gets put into the SPM and when it gets removed. By doing this, they can ideally achieve a perfect hit rate in the on-chip memory. In an actual SPM system, using the scratchpad requires some support from the running software. One way to use the SPM is directly through assembly programming. Similarly, a compiler could use specific libraries to translate the specific assembly instructions into instructions in a higher level language. Lastly, the compiler could be made aware of the presence of a scratchpad and act accordingly by scheduling loads and stores for the programmer on predictable data to make sure that the SPM is utilized correctly [2].

Earlier studies in the embedded systems domain indicate that the SPM is faster and more energy efficient than a conventional cache [2; 3; 18; 29]. This is because scratchpad memories typically do not contain tag memory, tag comparators, data prediction hardware, and hardware required to maintain coherency. Not having this extra hardware enables SPMs to consume less energy, have a reduced area, run cooler, and have faster access times than caches. Scratchpad memories also provide better application performance than traditional caches. This is because data in an SPM is guaranteed to hit unlike in a cache which is managed by hardware. Several prior efforts have shown

that SPMs improve upon the area delay product of applications when compared to traditional caches [2; 3; 26]. In the experiments, the SPM is used to stage large data arrays. Read-only data is brought into the SPM through DMA transfers by the programmer. At some point, the compiler may be able to help support this by detecting applicable data and bringing it into the SPM. Many other previous works enable this [1; 15; 19; 20; 29].

In literature, scratchpads have been used for storing instructions [9; 16; 33; 23], data [6; 7], or both [29] for improving the performance and power on an embedded system. Since the operation of an SPM is controlled in software, its effectiveness can be improved by both application analysis and compilation techniques [4; 31]. Alternately, placing compressed data in the SPM has been shown to be effective in improving system performance [25]. SPMs have also been proposed to assist the operation of thread creation and management in chip multi-processors, thus increasing the efficiency of parallel programs in that system [5]. Leverich evaluates the streaming programming model using SPMs [22]. Their work does not look at SPMs in the Level 2 of the on-chip memory hierarchy. Also, they do not evaluate DMV and SPMV multiplication for their architecture, which are considered in this work.

Dense and sparse matrix-vector multiplications are important kernels to many scientific codes. Several studies have been made to improve the performance of these important calculations [8; 11; 35]. In this study, the performance of the basic kernel is improved through architectural changes.

Multi-core chips have been studied extensively in many previous works. The IBM Cell processor, which uses an SPM for its local storage, has been profiled for scientific computations [36]. Previous work with the Cell processor however, does not compare

SPM and cache based systems nor does it do a thermal analysis. The use of SPMs in the memory hierarchy of a multi-core system has been shown in other studies [15]. However, the work performed does not consider a traditional cache to be a part of the on-chip memory. Further, the study does not characterize the power, performance, area, and temperature gains in those systems for any particular application. This work focuses on showing the benefits of using scratchpads on a similar system for an important scientific computation and improved area and temperature.

In this work the use of SPMs is evaluated on both a single-core processor and a chip multi-processor at all levels of the on-chip memory hierarchy. Also considered are hybrid scenarios in which both the cache and the SPM are present at the same level in the memory hierarchy. While some of the configurations do not exist in commodity hardware (an SPM as the only Level 2 memory for instance), it is the goal to show the impact of using an SPM for all possible scenarios.

From the previous work completed, it can be seen that a study on SPMs can be highly rewarding. Scratchpad memories can increase the performance of a processor by having faster access times. They can also shrink the die size of a chip from their reduced complexity leading to higher manufacturing yield or more on-chip memory or features. Due to an SPM having lower energy requirements, more benefits can be seen such as lower cooling costs, greater reliability, even higher performance, lower leakage power, and the design of the chip being environmentally conscious.

Chapter 3

Experimental Design

This section discusses the preliminaries, the setup of the designs, and the necessary steps that must be taken to perform the experimentation.

3.1 Preliminaries

To store all of the sparse matrices used in the SPMV multiplication experiments, the CSR format was chosen [8; 11]. This format stores the matrices in a very efficient manner. An example of a sparse matrix in the CSR format is shown in Figure 3.1. In this format, three arrays are required to store the sparse matrix, namely the row array ($Rows$), column array ($Cols$), and value array ($Vals$). If N_R represents the number of rows (and columns) and N_Z represents the number of non-zeros in the matrix, $Vals$ is an [N_Z x 1] array containing the values in the matrix stored in the order of their rows. $Cols$ is also an [N_Z x 1] array. It contains the column numbers for each of the non-zero elements in the order which it is stored in the $Vals$ array. $Rows$ is an [(N_R + 1) x 1] array where the i^{th} element points to the first entry of the i^{th} row in $Vals$ and $Cols$.

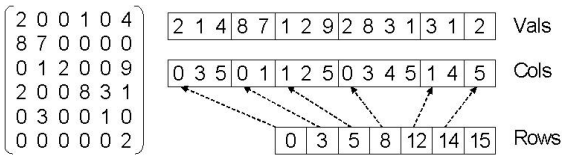


Figure 3.1: Compressed Sparse Row format.

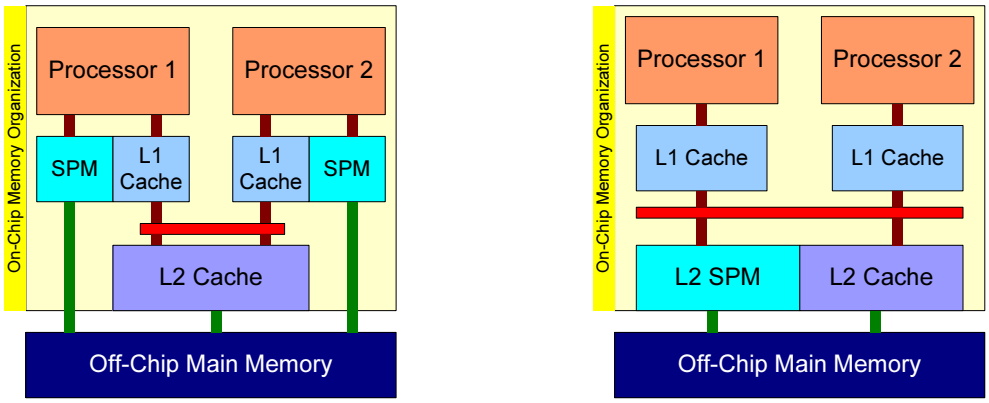
Algorithm 1 *Basic Sparse Matrix-Vector Multiplication.*

```
1: for I : all elements in the row array do
2:    for J := Rows(I) to Rows(I+1)-1 do
3:       result_I +:= Vals_J * X_{Cols_J}
4:    end for
5: end for
```

The basic sparse matrix-vector multiplication on a single core using CSR formatted matrices is shown in Algorithm 1. In Line 3 of the algorithm, the i^{th} element of the *result* vector is being computed. X is the input vector with which the matrix is being multiplied. The multi-core version of this code entails creating multiple threads, one for each core, and distributing equal number of rows, in a non SPM based system, to each of the threads. In an SPM based system, the work is distributed based on what can fit in the SPM.

Figure 3.2 shows some of the basic memory hierarchies that are explored. As can be seen, some of the architectures have two memory components at the same level of the memory hierarchy (an SPM and a conventional cache) for a processor. At these levels, coherency must be maintained. This can be done in hardware by having a protocol which maintains coherency. Alternately, the compiler can be assigned to make sure no coherency violations occur between the memory components. This overhead is avoided in this work by only using the SPM for data that is read and not written to. All this information is readily known in the application, and thus, it is proposed to modify the application to be aware of the presence of an SPM.

One last preliminary concern is the running of legacy applications that have not been optimized for use with the SPM. These can be run on some of the architectures that will be covered shortly including variations that have a split memory hierarchy with



SPM in Level 1 of memory (a)

SPM in Level 2 of memory (b)

Figure 3.2: SPM in on-chip memory hierarchy of a chip-multi processor.

both an SPM and a traditional cache at the same level. If a legacy application is to be run on an SPM only architecture, it will have to be recompiled using a compiler that is aware of the SPM and will profile the application. Recompilation could also be done for applications running on the split memory hierarchy. This would produce a program that is more highly optimized for use on a system with a SPM.

3.2 Architectures Explored

Figure 3.2 shows two of the six different architectures explored in this work. Figure 3.2(a) shows the general case when the SPM is present in the first level of the on-chip memory hierarchy. The SPM is filled in directly by the off chip memory through DMA transfers scheduled in the software. Figure 3.2(b) shows the general case when the SPM is present in the second level of the cache hierarchy. Here again, the SPM is controlled by the software and it services the requests of the Level 1 cache.

In addition to the two architectures shown, the exploration includes cache only architectures at both levels of the on-chip memory hierarchy. This configuration serves

as the *Baseline*. Also explored are architectures that only have an SPM for its level of memory and architecture with only SPM for all levels of memory. Figure 3.2 is considered to be correct when the sizes of the memory elements are non zero.

The on-chip memory architectures where a scratchpad is present in both the levels of the memory hierarchy and where the entire Level 1 is an SPM are only explored from a layout view. In matrix-vector multiplication, when an SPM is present in the Level 1 stage, the program is able to perform the multiplication by staging most of the data in the SPM, and thus, the utilization of Level 2 is very low. As a result, the pressure on a Level 2 cache is not high enough to justify using an SPM in Level 2 when there is a Level 1 SPM. However, the all SPM architecture hybrid might be more interesting to study for other applications where the utilization of the Level 2 cache is still high even in the presence of a Level 1 SPM. When the entire Level 1 is an SPM, the architectural software required becomes increasingly difficult due to the size of the SPM. In addition, it becomes difficult to scale the problem of matrix-vector multiplication while maintaining proper use of the Level 2 memory system. As a result, these two architectures were not evaluated for matrix-vector multiplication.

In total, six memory architectures were analyzed for layout and temperature. Each of these were variants of a DEC Alpha EV6 processor and were performance optimized. The basic layout of such a processor is shown in Figure 3.3. The six architectures are detailed in Table 3.1 and can be found visually in Appendix A. These include a "Baseline" example of 32KB L1 data cache and 2MB L2 unified cache. These numbers closely parallel what would have been in an actual EV6 processor. The next architecture is a "Split L1D" design. This takes the regular L1 data cache and splits it in half so that

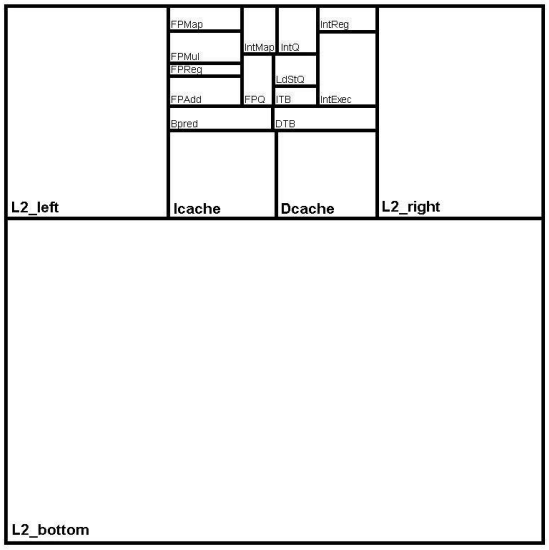


Figure 3.3: DEC Alpha EV6 Baseline Architecture.

16KB are traditional cache and 16KB are SPM. Only the data cacheis focused on since instructions perform well in traditional cache and would require extra analysis at run-time to determine what to load. The next architecture is "L1D SPM". This architecture replaces the entire L1 data cache with an equally sized SPM. The next architecture is a "Split L2". This architecture is similar to the previous split model in that the original cache is split in half to form two equal sized caches and SPMs. Each will be 1MB in size. An entire "L2 SPM" is looked at in which the unified L2 cache will now be one 2MB SPM. The last architecture explored is one that is "ALL SPM". This replaces the L1 data cache and the L2 cache entirely with two SPMs.

This many test cases are included as different applications and different circumstances would require different properties of the memory system. For instance, if an application is very memory bound, it may be beneficial for the entire memory system to be SPM to increase the performance substantially. Also, split models of the cache are included to support mixed use systems. On these systems, there may be legacy code that

Table 3.1: System memory configurations.

Config	L1D Cache	L1 SPM	L2 Cache	L2 SPM
Baseline	32KB	0KB	2MB	0MB
Split L1D	16KB	16KB	2MB	0MB
L1D SPM	0KB	32KB	2MB	0MB
Split L2	32KB	0KB	1MB	1MB
L2 SPM	32KB	0KB	0MB	2MB
ALL SPM	0KB	32KB	0MB	2MB

has not been optimized by a supported compiler, in which those programs would have to be limited to the traditional cache. One last assumption that is made is to keep the sizes of the memory system the same throughout the analysis. It was assumed that the chip's die could only contain the maximum memory sizes specified in the base configuration. To make room for the SPM on the chip, some of the cache has to be sacrificed. This is to observe the size differences between the different architectures and to keep the memory elements equal to exact powers of two.

Four of the memory configurations from Table 3.1 were tested using dense and sparse matrix-vector multiplication. Specifically, the "Baseline", "Split L1D", "Split L2", and "L2 SPM" memory system setups were tested. As before, the *Baseline* configuration is defined to have no scratchpads, 32KB Level 1 data (L1D) cache per core, 32KB Level 1 instruction cache and 2MB of unified Level 2 cache. The remaining three system setups were compared to this configuration for analysis and remained the same as before. For matrix multiplication, an SPM could simply be added to an already existing chip design making it larger, but this method would not offer a fair comparison. Adding the SPM would increase the total memory space and would not allow comparisons of performance or energy consumption.

Table 3.2: 0.13 micron energy details of on-chip memory components.

Component	Energy Required (mW)
32KB L1	168.2
16KB L1	91.4
32KB SPM	66.3
16KB SPM	35.8
2MB L2	4573.7
1MB L2	2491.0
2MB SPM	2000.6
1MB SPM	1071.2

3.3 Simulation Environment

All of the thermal simulations were performed using the University of Virginia's HotSpot application [14]. Using a block level analysis, it is possible to break the processor into each of its components and analyze them separately. Due to limited availability of data on the processor being used, the simulations had to be done assuming 0.13 micron process technology. Although this technology is not the most recent, it is still able to point out the benefits of scratchpads. All of the energy numbers for the memory systems were obtained from HP Labs' CACTI 4.2 [30]. These were combined with typical use approximations to get the basic energy numbers that are available in Table 3.2. As can be seen, the scratchpad requires far less energy than traditional cache. The energy numbers for the rest of the processor came with HotSpot's example simulation.

Simics [15], a cycle-accurate, full system simulator, was used to simulate and test the CMP matrix-vector multiplication architectures. The system simulation had Solaris 9 running as the operating system on the architecture. The processors are modeled to be in-order execution cores running the SPARC ISA. Multi-core systems that have up to 16 processors are analyzed. Each of the cores operates at 1GHz. The access time for

Table 3.3: Details of on-chip memory components.

Size	No. of Lines	Line Size	Assoc.	Penalty
32KB L1	512	64	2	1
16KB L1	256	64	2	1
16KB SPM	256	64	N/A	1
2MB L2	8192	256	8	10
1MB L2	4096	256	8	10
2MB SPM	8192	256	N/A	6
1MB SPM	4096	256	N/A	6

all of the memory components were obtained from HP Labs' CACTI 4.2 at 70nm. It is assumed that an access to main memory would result in a stall of 200 cycles.

The characteristics of the memory configurations tested are outlined in Table 3.3. As the SPM is used to store data, the instruction cache configuration remains the same for all the experiments. A scratchpad placed in the Level 1 memory hierarchy is set to have a penalty of one cycle. This is equal to that of an identically sized conventional cache. Access to the second level of the memory hierarchy is assumed to have a default penalty of 10 cycles. Access to an SPM at this level will have a penalty of 6 cycles. Previous work shows that the access time to a scratchpad will be less than that of a similarly sized cache [2; 3; 26]. This explains why the SPM will have a smaller penalty than a cache when both of them are the same size. Although smaller memory structures and non-contiguous address spaces will provide smaller access times, it is assumed the penalty to be the same for all sizes of Level 2 cache to isolate the effects of using a scratchpad. The specific power and energy numbers used for the matrix-vector multiplication systems are given in Table 3.4. All of these numbers were obtained or approximated from CACTI [30]

Table 3.4: 70nm power and energy details of on-chip memory components.

Size	Dynamic Read Energy (nJ)	Dynamic Write Energy (nJ)	Leakage (mW)
32KB L1	0.0839	0.0236	277
16KB L1	0.0654	0.0213	146
16KB SPM	0.0151	N/A	57
2MB L2	2.3624	0.3314	7838
1MB L2	1.5172	0.1932	4145
2MB SPM	0.6496	N/A	3428
1MB SPM	0.3958	N/A	1782

using 70nm process technology. It was assumed in obtaining these numbers that the same optimizations used for caches as they increase in capacity could also be used for SPMs.

For the dense matrix-vector multiplication work, five different sizes of matrices were used to perform two different studies. The dimensions of the matrices used were 256x256, 512x512, 1024x1024, 2048x2048, and 4096x4096. Each of these matrices was populated with double precision values based on their location in the matrix. The matrices themselves were kept in memory as a single array of values. The first study performed with these matrices was a fixed problem size experiment. Each of the four memory configurations were simulated on chips that have 1, 2, 4, 8, and 16 cores. Each of these setups performed DMV multiplication with the 1024x1024 element matrix. The second study performed with the densely populated matrices was a scaled problem size experiment. All of the same memory and core configurations were used, but a system with 1 core used the 256x256 matrix, 2 cores used the 512x512 matrix, 4 cores used the 1024x1024 matrix, 8 cores used the 2048x2048 matrix, and 16 cores used the 4096x4096 matrix.

Table 3.5: Sparse Matrices used in the experimental evaluation.

Name	No. of elements (in millions)	No. of non-zeros (in thousands)	Density
fdm2	1,025	159	0.02%
bcsstk31	1,267	1,181	0.09%
msc23052	531	1,143	0.22%
qa8fm	4,373	1,661	0.04%
cfd1	4,992	1,826	0.04%
c-71	5,873	860	0.01%
filter3D	11,329	2,707	0.02%
Ga19As19H42	17,722	8,885	0.05%
bmw7st_1	19,979	7,318	0.04%
Si41Ge41H72	34,462	15,011	0.04%

To fully analyze sparse matrix-vector multiplication, two studies were performed that are very similar to the ones performed for DMV multiplication. The first was a fixed problem size experiment. Each of the four memory configurations were simulated on chips that have 1, 2, 4, and 8 cores. Each of these setups then performed SPMV multiplication with 10 different sparse matrices. These matrices are outlined in Table 3.5. To try and take advantage of the matrix's structure, each matrix was RCM reordered and stored in CSR format. The reordered matrices can be seen in Appendix B. The ten matrices vary in size, density, and shape to show the scratchpads effects on all types of sparse matrix multiplication. The second study performed with sparse matrices was a scaled problem size experiment. The same memory configurations with core configurations now having up to 16 cores were used with a sparse matrix with varying dimensions. The particular matrix used was an elasticity matrix used to calculate the stiffness for a brick in three dimensions. The matrix's dimensions could be determined and were chosen such that setups with 1 core used a 1000x1000 matrix, 2 cores used a 2000x2000 matrix,

4 cores used a 4000x4000 matrix, 8 cores used a 8000x8000 matrix, and 16 cores used a 16000x16000 matrix. Since the matrix exhibits natural block diagonal data structure, it was not reordered. The matrices used can also be found in Appendix B.

3.4 Application Profiling

Application profiling is essential for effective utilization of the limited SPM space. Data structures that perform well in the regular cache hierarchy should not be placed into the SPM. The dense matrix-vector application is profiled using the algorithm discussed shortly in Section 3.4.2 and the sparse matrix-vector application using Algorithm 1 to understand the specific data vectors required to be placed in the SPM. Since read-only data is to be put in the SPM, the entire dense matrix and the dense vector are candidates for DMV multiplication. The row array ($Rows$), the column array ($Cols$), the value array ($Vals$), and the vector with which the sparse matrix is multiplied (X) can be put into the scratchpad in the SPMV multiplication case. Profiling the algorithms estimates the miss rate of each of the component data structures. The miss rate of a data set is the percentage of the total number of misses to that data structure divided by the total number of access to that data structure. Figure 3.4 shows the results of the miss rates of one of the sparse matrices (fdm2) on a single core. The results of all the other matrices were similar having a higher miss rate for anything that wasn't accessed sequentially or that dealt with the elements of the matrix directly. From Figure 3.4 it can be seen that the sequentially accessed $Rows$ array has a very low miss rate and is thus not a good candidate to be staged in the SPM. The $Rows$ array is left to be handled in cache when available and $Cols$, $Vals$, and X to be handled through a SPM when available. Due to

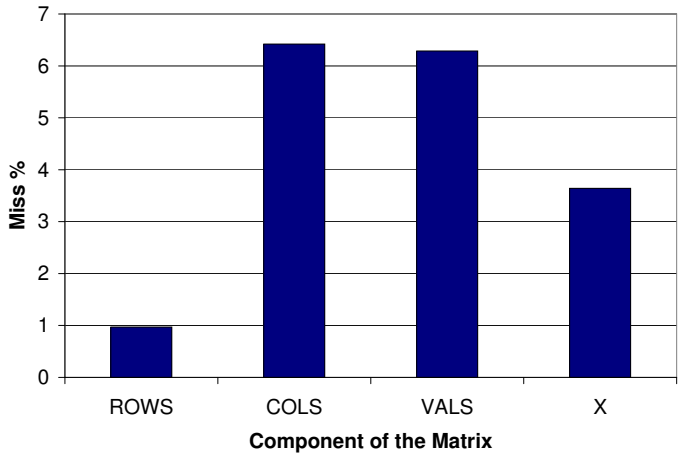


Figure 3.4: Miss rate for the fdm2 matrix.

the algorithm used by the dense matrices, both the matrix itself along with the dense vector make excellent candidates to be placed in the SPM.

3.4.1 Offline Stage Detection

One of the important tasks performed when using a scratchpad memory to perform sparse matrix-vector multiplication is stage detection. A stage is defined as the range of rows that can be multiplied at once when all the elements that the multiplication needs from the $Cols$, $Vals$, and X data sets are present in the scratchpad.

Algorithm 2 shows the stage detection algorithm. The goal of this algorithm is to maximize the total number of rows that can be put into a single stage. By maximizing each stage, the number of DMA transfers that would be required by software can be minimized. In this algorithm, the array $Stage[\ ]$ is filled with the beginning row of all of the stages. The end of each stage can be inferred from the beginning of the next stage. The $Total_{size}$ in the algorithm represents the size required when rows from I to J are being staged in the SPM. The contribution of the column array is $(J - I) * Size_{int}$.

Algorithm 2 *Offline Stage Detection().*

```
1: Stage[1] := 1 /*Array of stage beginnings.*/
2: for I < N_rows do
3:    for J := [I ... N_rows] do
4:       Cols_size := (J − I) * Size_int
5:       Vals_size := (J − I) * Size_double
6:       X_size := (max_col(i,j) − min_col(i,j) ) * Size_double
7:       Total_size := Cols_size + Vals_size + X_size
8:       if Total_size > Size_SPM then
9:          Stage[N_stages] := J − 1
10:         N_stages := N_stages + 1
11:         break out of the inner FOR loop
12:      end if
13:   end for
14:   I := J
15: end for
```

$Vals$ contributes $(J - I) * Size_{double}$ to the stage size and X contributes $(max_{col(i,j)} - min_{col(i,j)}) * Size_{double}$ to the stage size. Note that the computation of finding the maximum and the minimum element of X accessed needs to be performed as there is no assumption that the banded matrix has a regular structure inside the band. The sum of these contributions is compared with the size of the SPM. If it is lower, another row is fit into the SPM. If not, the stage is set and the iteration to determine the next stage begins.

It is important to note that this analysis is independent of the actual X vector. Thus, it needs to be evaluated once offline for each sparse matrix. The offline analysis can be performed in approximately O(N_R) efficiency, where N_R is the number of rows in the input sparse matrix. The size of the SPM ($Size_{SPM}$) is an input to this algorithm. It is also important to note that double point precision is used which limits the number of rows that can be put into each stage when compared to other data types. This code assumes that the number of non-zeros in a single row does not fill the entire SPM, which

is true in all of the benchmark matrices. However, the code can be modified very easily to take into account those situations as well and has not been shown in Algorithm 2 for brevity. The code would simply need to calculate and compare the $Total_{size}$ after every 1 or more row elements. Which row element the next stage should include should be stored between iterations.

3.4.2 Dense Matrix-Vector Multiplication

In order to multiply the dense matrix with the dense vector, the dgemv() subroutine was used. This subroutine is a Basic Linear Algebra Subprogram (BLAS) that is part of the Sun Performance Library. The Sun Performance Library is a set of optimized, high-speed mathematical subroutines for solving linear algebra and other mathematically intensive problems. This particular algorithm was chosen since it is highly optimized for Sun hardware and is widely used in scientific and technical codes. Since the dgemv() subroutine is part of the Sun library, this makes it very difficult to modify the function to take advantage of proper stages in the SPM. Due to this, the penalty of adding staging functionality is based on the other experiments ran and added on to the results assuming one large stage.

To parallelize the DMV multiplication, the workload was divided into threads among the available processors based on the rows of the matrix. Since these are fully populated dense matrices, the amount of work done and the number of SPM loads will be approximately equal. To make effective use of the cache, the threads need to be bound to a core for effective prefetching of data into the SPM through DMA transfers. The number of DMA transfers will be balanced among the cores. Because all of the

Algorithm 3 *SpMV _thread().*

```
1: for K := [N_stages * thread_ID ... N_stages * (thread_ID + 1)] do
2:    for I := rows in Stage_K do
3:       for J := columns of row_I do
4:          result_I += Vals_J * X_Cols_J
5:       end for
6:    end for
7: end for
```

dense matrices used have dimensions that are powers of two, they will divide evenly into threads across the number of cores.

3.4.3 Sparse Matrix-Vector Multiplication

Algorithm 3 shows the pseudo code executed by each thread of the sparse matrix-vector multiplication. The workload is split amongst the threads based on the number of stages such that each thread works on the same number or parts of stages. Due to the possibility of rows having unequal numbers of elements, distributing work to the processors in this way as opposed to the number of rows, in general, results in a more balanced program execution. The first line assigns the stages that the current thread needs to calculate. The data is prefetched into the SPM before a stage is selected. Afterwards, the multiplication proceeds normally for the rows that constitute the stage. This means that the number of SPM loads from the main memory will be the same for each thread. Like with DMV multiplications, threads should be bound to a core for effective prefetching of data. Also note that the number of stages might not be exactly divisible by the number of processors. In that case, one of the threads handles the remaining stages. This has not been shown in Algorithm 3 for brevity.

Chapter 4

Results

This chapter discusses the results to the various experimentations that were performed. It starts out by describing the specific chip layout results, followed by performance results, and then concluded with the energy results.

4.1 Layout

As stated earlier, scratchpads are much smaller than traditional caches so it is only natural that when looking at the total area of the six architectures tested, the ones with scratchpads are the smallest. This is most evident in Figure 4.1. Since the architecture with the most memory replaced with scratchpads is the "ALL SPM" test, it can be confirmed that this one is the least in size. If this is compared this to the baseline, there is a 29.84% decrease in area. From this graph, it can also be extrapolated that the other architectures have decreases in area. Specifically, they are, from left to right, 0%, 0.52%, 1.05%, 14.19%, 28.59%, and 29.84%.

In Figure 4.2, the power density graph derived from the area can be seen. Because each experiment left the size of the processor decrease, there is a smaller chip towards the right. In turn, this makes the power density graph grow larger as the power tradeoff is not as great as the area tradeoff. It is important to point out that this growth is not as

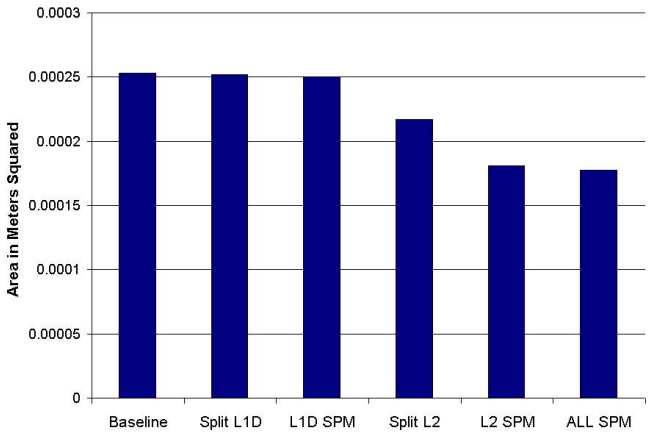


Figure 4.1: Processor Area.

large if the area was reduced with traditional cache. The power density only rises by at most 33.26% after the power requirement of the memory system is reduced by 56.41%.

Figure 4.3 shows the temperatures of each of the processor's components. As can be seen from the graph, some components needed to be added or taken away depending on the architecture being used. In general, the graph makes a clear pattern that as scratchpads are added to the system, the temperature of each component decreases. The components that are directly next to an SPM see the greatest amount of change with the graph having only a few anomalies.

Finally, Figure 4.4 points out the average temperature results for all of the simulations ran. The top red line depicts the average component temperature. The line has a negative slope showing that the average component temperature decreases as SPMs are added. For the "Split L1D" case, the decrease is the largest having a 1.59% decrease in average temperature. This one tends to be the lowest since there is an additional component paired with the extra space on the chip that brings down the average. The second least is the "ALL SPM" case which is able to drop the temperature by 1.22%.

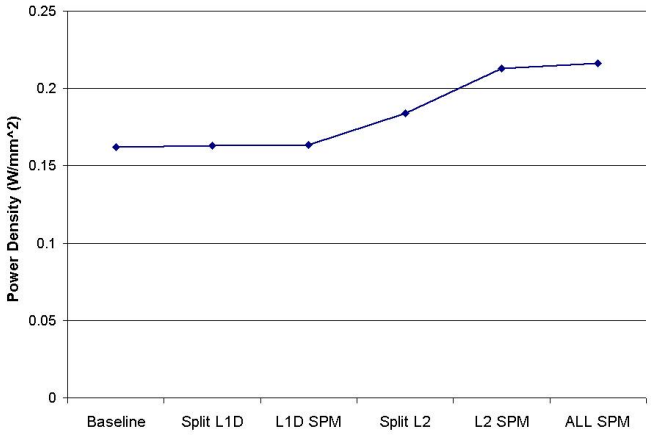


Figure 4.2: Power Density.

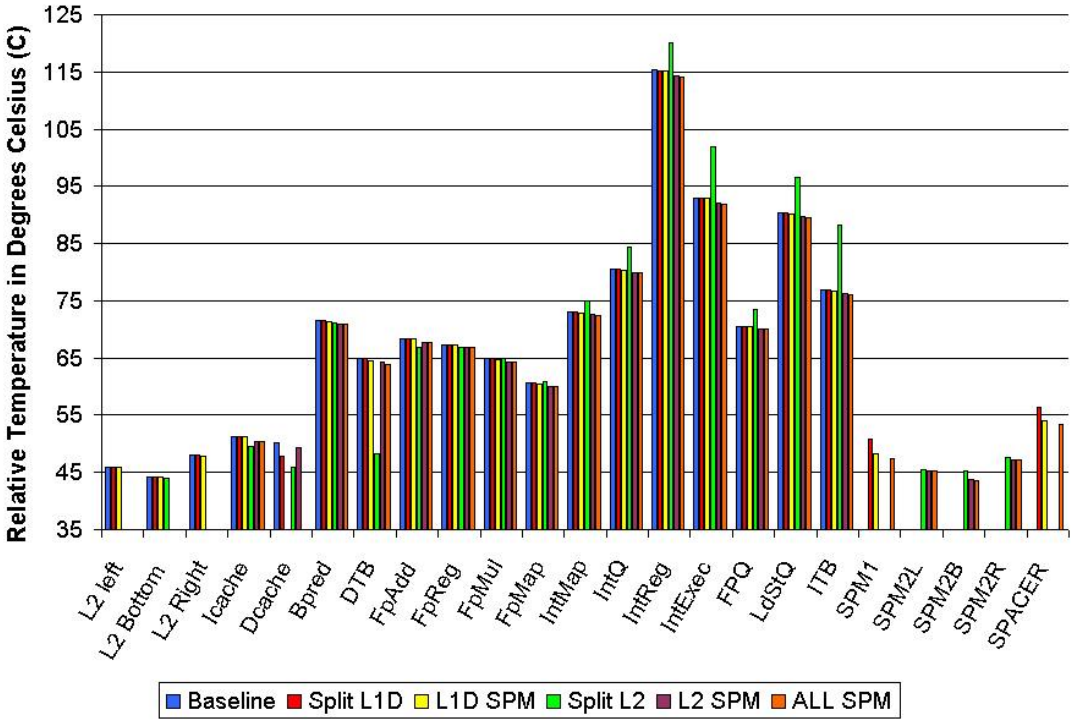


Figure 4.3: Processor Component Temperatures.

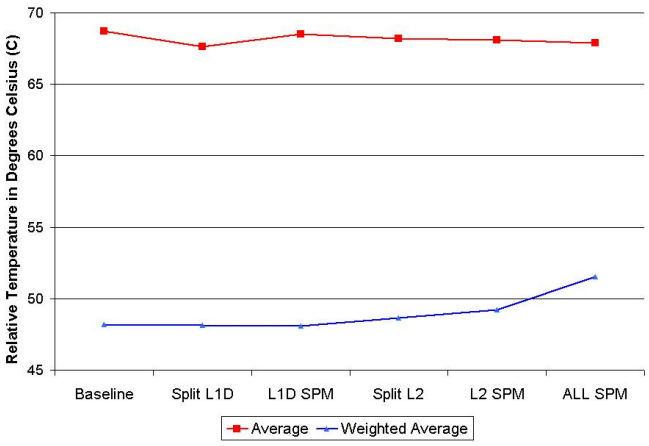


Figure 4.4: Average Processor Temperature.

The other architectures have similar results below 1% of difference. The bottom blue line depicts the weighted average temperature of the chip. This graph behaves very similar to the power density graph since the area of the coolest parts on the chip are decreasing. Since the SPM decreases area and decreases temperature in different ratios, the graph tends to rise as the effects of area are much greater.

4.2 Performance

The *Baseline* for these evaluations is considered to be the architecture with traditional cache as the only memory present at both levels of on-chip memory hierarchy. This and the other configurations tested for performance are outlined in Section 3.2.

Figure 4.5 shows the results of performance for the dense matrix-vector multiplication study. The architectural configurations are the same ones outlined in Table 3.1. The first series in Figure 4.5 is for a fixed problem size while the number of cores changes. For this study, the performance increase as measured in the number of cycles, is greatest on a single core, which is over 60%. The relative amount of performance

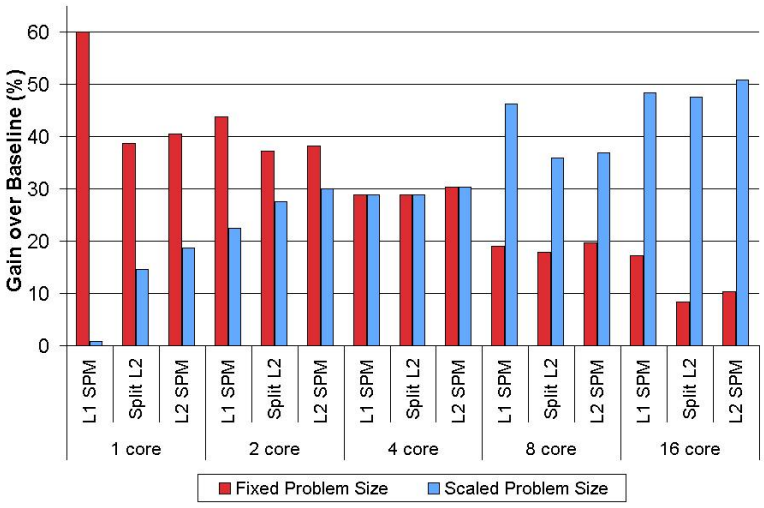


Figure 4.5: DMV multiplication performance (cycles) benefits.

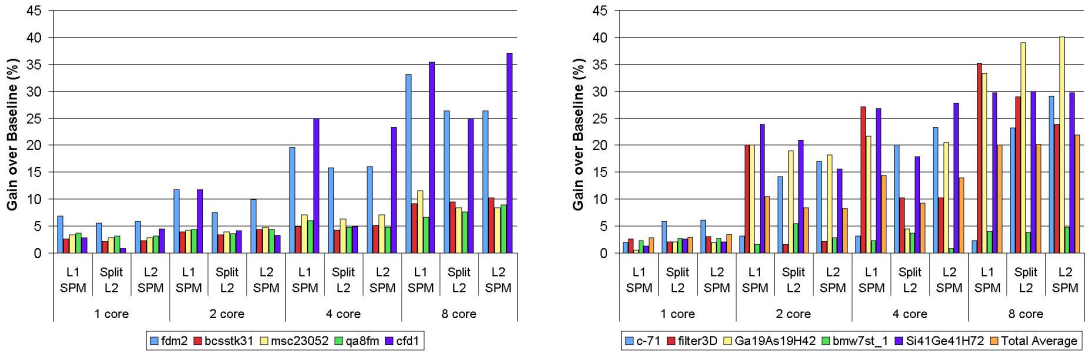
improvement decreases as the number of cores increase. In this study, the least amount it just over 8% for a 16 core system. This can be attributed to the fact that the Baseline also improves greatly as more cores are added to it. In addition, the amount of work per core decreases as more cores are added. Therefore, the total opportunity for an SPM to improve performance decreases as the cores increase. The different memory configurations can be seen to have some effect on the fixed problem size's performance as well. When the SPM is at the Level 1, the performance benefit seems to be greatest. This could be attributed to having the needed data so close to the processing cores and the additional memory it provides for CMPs.

Figure 4.5 also shows the performance results for the dense matrix-vector multiplication with a scaling problem size. The graph shows that as the number of cores increase along with the problem size, it is better to have more cores with a scratchpad configuration. The performance improvement ranges from below 1% on one core up to

over 50% on 16 cores. In addition, the dense algorithm seems to benefit more from having the scratchpad in the Level 2. This produces better results since as the problem size grows in size, less and less will be able to fit in the cache. Also, an SPM at the Level 2 has a reduced access time which can be taken advantage of. It is important to point out how the scaled problem actually grows faster than the number of cores so the isoefficiency is not quite identical. Over the entire dense matrix study, it becomes clear that SPMs provide the most improvement when the problem size is very large compared to the number of cores available.

The performance results for the sparse matrix-vector multiplication fixed problem size study can be found in Figure 4.6. Figure 4.6(a) shows the five smaller sparse matrices while Figure 4.6(b) shows the five larger matrices along with the total average. From these graphs, it can be seen that in general, as the number of cores increase, the performance will also increase. This is because of the complex access patterns of sparse matrices paired with the benefits of multi-core processors. The results show a maximum performance increase of about 40% on an 8 core system, with an average improvement of about 21% on an 8 core system. The graphs in the figure also show the results based on the configurations in Table 3.1. The improvements based on the configurations are not dependent on where the SPM is placed. In general, the results across a core are fairly equal with respect to the configurations. This study also shows that the physical structure of the sparse matrix is very important. The improvements vary greatly depending on the matrix's size, density, and shape.

The performance results for the scaled problem study of sparse matrix-vector multiplication can be found in Figure 4.7. In this scaled problem study, the results show



(a) (b)

Figure 4.6: Fixed problem size SPMV multiplication performance (cycles) benefits.

a relatively flat trend. As the number of cores increase, and so does the problem size, the amount of improvement stays relatively the same, only increasingly slightly. Since in this study the actual problem size reflects almost a true doubling of work, this shows that the SPM can scale nicely with the problem size to provide benefits. The memory configurations also show a similar result to that of the sparse matrix fixed problem size study. Although having the SPM in the Level 2 shows a slight preference, having an SPM in the memory hierarchy will be able to improve the performance of the system.

The performance benefit when the SPM is placed in the Level 1 comes from the predictability of data leading to an overall decrease in the miss rate of Level 1 memory. The ability to load the SPM with the data that is needed allows for a near 100% hit ratio causing little slowdown due to memory. When the SPM is in the Level 2 memory, the benefit not only comes from a decrease in the miss rate of the Level 2 memory but also the faster access times for SPM compared to an equal sized Level 2 cache. Under the memory characteristics, each access miss from Level 1 cache now will have a penalty of 6 cycles in an SPM as opposed to 10 cycles in a cache.

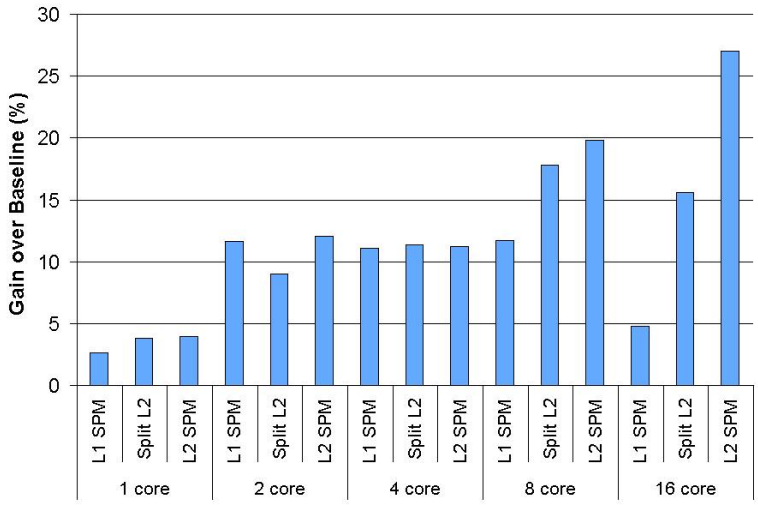
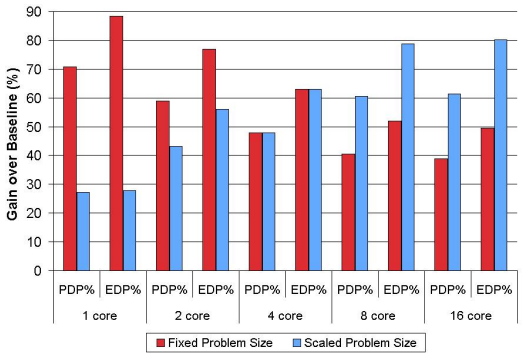


Figure 4.7: Scaled problem size SPMV multiplication performance (cycles) benefits.

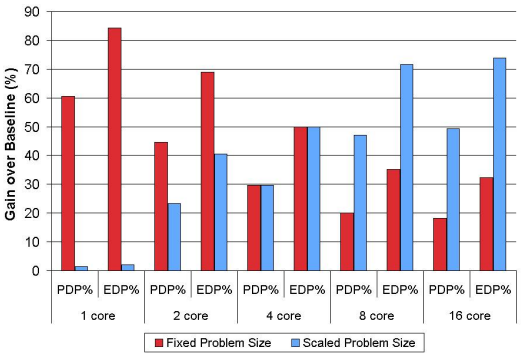
4.3 Energy

In all of the experimental results, the percentage improvement is presented against the *Baseline* described in Chapter 3. All of the configurations examined are detailed in Section 3.2. The measurement of Power-Delay Product (PDP) is the power required by the system multiplied by the execution time. The measurement of Energy-Delay Product (EDP) is the PDP multiplied by the execution time.

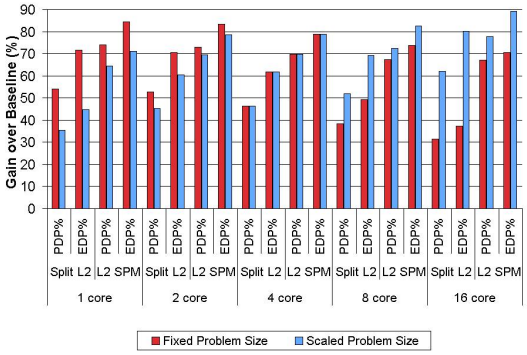
Figure 4.8 shows the PDP and EDP improvements of the dense matrix-vector multiplication study. The results have a natural correlation to the performance improvements discussed in the earlier section. Figure 4.8(a) shows the improvements of the Level 1 memory systems when a scratchpad is present at that level. For a fixed problem size, the PDP can be improved by as much as 71% for 1 core and up to 39% for a 16 core system. The EDP is shown to be improved the most on a single core system with 88% in savings while nearly 50% on a 16 core system. For the scaled problem size, the PDP



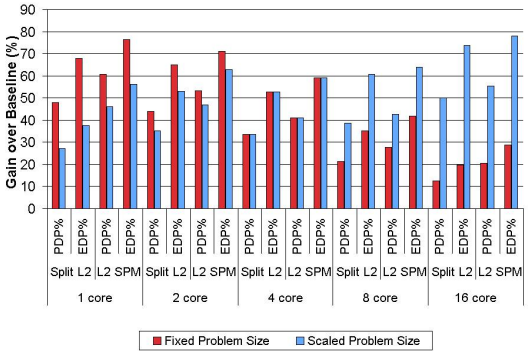
Savings in level 1 memory
(a)



Savings in the entire on-chip system
(b)



Savings in level 2 memory
(c)



Savings in the entire on-chip system
(d)

Figure 4.8: Impacts on the power-delay product (PDP) and the energy-delay product (EDP) for DMV multiplication.

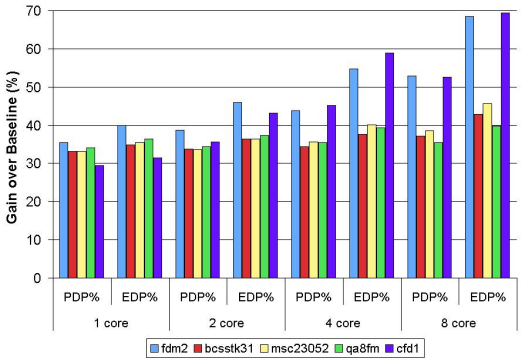
improvement is 27% on a single core and maximized on 16 cores with 62%. The EDP improvements range from 28% to 80% depending on the number of cores.

Figure 4.8(b) shows the PDP and EDP improvements for the entire on-chip system when the SPM is still placed in Level 1. Since the Level 1 memory components only contribute a little to the entire chip's energy requirements, these numbers are reduced. For the fixed problem size study, the PDP improvements now range from 61% to 18% while the EDP improvements range from 84% to 32%. The scaled problem size study's results see a similar degradation. The PDP improvements now range from 1% to 49%, while the EDP improvements now range from 2% to 74%.

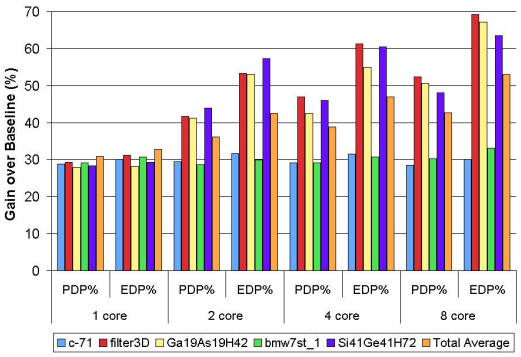
Figure 4.8(c) shows the PDP and the EDP improvements of the Level 2 memory system when an SPM is placed at this level. In the figure, both viable configurations are shown. Like the performance results, more improvement generally comes from replacing the entire cache with an SPM. The fixed problem study shows PDP improvements ranging from 74% to 67% for the all scratchpad based configuration. The scaled problem size study shows PDP results from 65% to 78% improvement.

Figure 4.8(d) shows the entire on-chip system improvements for having an SPM at Level 2. Like the previous configuration, since the Level 2 memory is not the only contribution, these numbers are reduced slightly. For a fixed problem size, the PDP can drop by 61% to 71% for an entire Level 2 SPM. The EDP can be improved by as much as 77%. A scaled problem size will see a 41% to 55% drop in its PDP from an SPM based Level 2. The EDP will drop by a maximum of 78%.

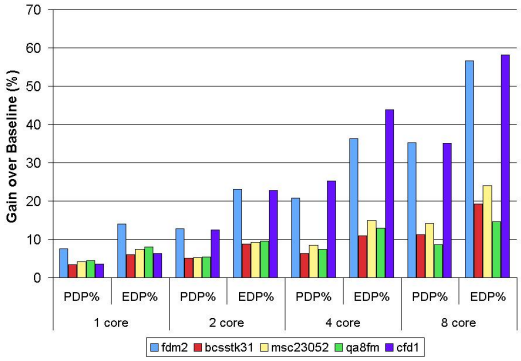
Figures 4.9 and 4.10 show the energy results for the fixed problem size study of SPMV multiplication. Figures 4.9(a) and (b) show energy savings in the Level 1 memory



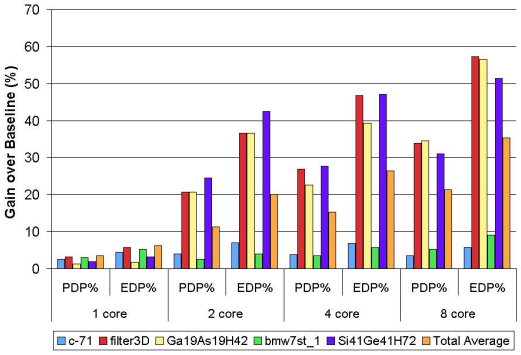
Savings in level 1 memory
(a)



Savings in level 1 memory
(b)

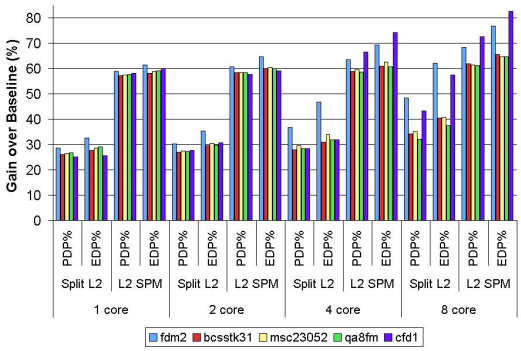


Savings in the entire on-chip system
(c)

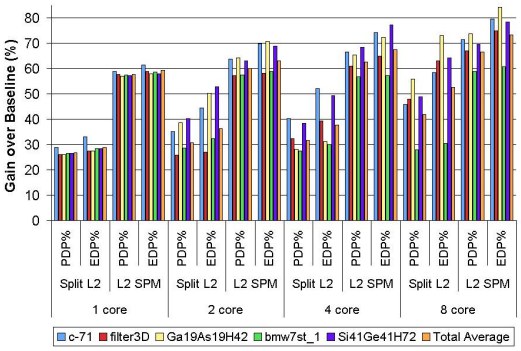


Savings in the entire on-chip system
(d)

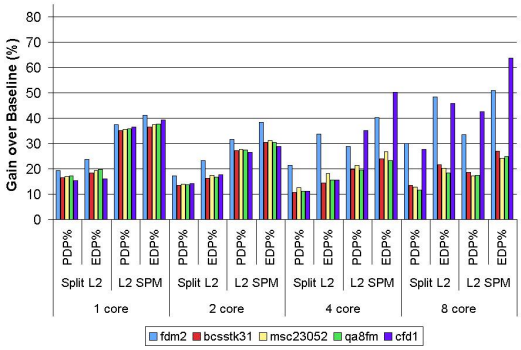
Figure 4.9: Impacts on the Level 1 power-delay product (PDP) and the energy-delay product (EDP) for fixed problem size SPMV multiplication.



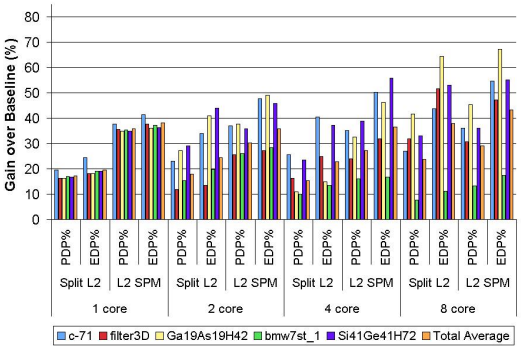
Savings in level 2 memory
(a)



Savings in level 2 memory
(b)



Savings in the entire on-chip system
(c)



Savings in the entire on-chip system
(d)

Figure 4.10: Impacts on the Level 2 power-delay product (PDP) and the energy-delay product (EDP) for fixed problem size SPMV multiplication.

system when an SPM is also at this level. With 1 core, there is an average 31% drop in PDP and a 33% drop in EDP. For an 8 core system, these increase to a 43% improvement in PDP and a 53% improvement in EDP.

Figures 4.9(c) and (d) show the relative savings of the entire on-chip system when an SPM is part of the Level 1 memory hierarchy. Since the Level 1 memory is a small contributor to the total energy required, these numbers are reduced from the ones in (a) and (b) and are mostly performance based. On a single core system, the PDP is improved by an average 4% and 6% for EDP. With 8 cores, these numbers increase to 21% for PDP and 35% for EDP.

Figures 4.10(a) and (b) show the energy savings of the Level 2 of memory when an SPM is present in either configuration. In general, due to its low energy requirements, the "L2 SPM" configuration provides the most improvements. For a single core, the PDP and EDP can be improved by 58% and 59%. On eight cores this increases to 67% and 73%.

Figures 4.10(c) and (d) show the results of the Level 2 configurations for the entire on-chip system. Once again, these are reduced slightly from those of (a) and (b) since the Level 2 memory is not the only part of the chip. For a system with an entire Level 2 of scratchpad memory, the PDP improvements can range from 27% to 36%. The EDP improvements can range from 36% to 43%.

The final figure shown is Figure 4.11 which shows the results of the scaled problem size study of SPMV multiplication. like the other studies, Figure 4.11(a) shows the energy savings of the Level 1 memory system for the "Split L1" configuration. Since the performance for this study varied across the number of cores, so did the energy results.

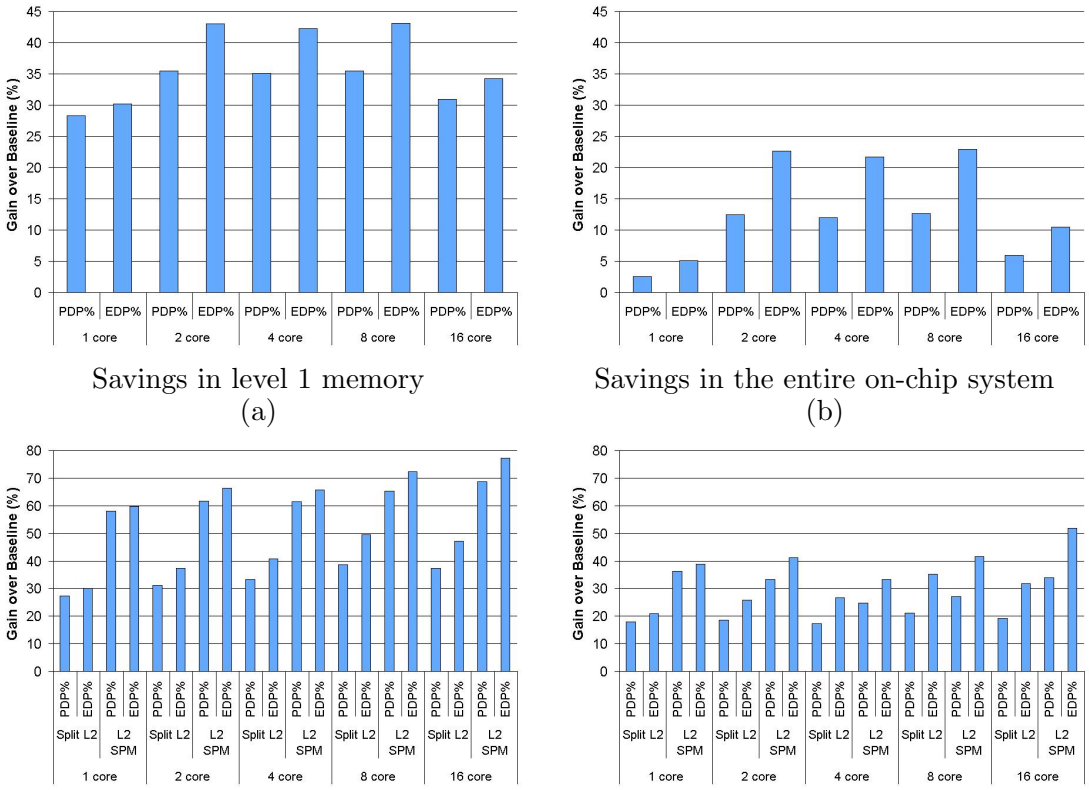
The PDP improvement ranged from 28% to 36%. The EDP varied from 30% to 43% improvement.

Figure 4.11(b) shows the entire on-chip system's energy savings. Since the Level 1 SPM is such a small contributor, these numbers are greatly reduced. The PDP can be improved by 3% to 12%. The EDP can be improved by 5% to 23%.

Figure 4.11(c) shows the energy savings of the Level 2 memory system. Both of the configurations tested are shown and, in general, the "L2 SPM" architecture allows for the most savings. For a single core, the PDP and EDP are improved by as much as 58% and 60%, while on a 16 core system these increase to 69% and 77%.

Figure 4.11(d) shows the reduced improvements when the entire on-chip system is considered. Here the PDP ranges from 25% to 36%. The EDP can be improved by 33% to 52%.

The improvements seen at the Level 1 memory system comes from two sources. First, the dynamic and leakage power consumption of an SPM are lower than that of a traditional cache. Second, the performance gain achieved by the SPM also contributes to reduce the leakage power of the components of a system. The majority of the improvement that is seen by the entire system when an SPM is placed at the Level 1 memory hierarchy is from the performance gain and the reduced leakage power of the scratchpad compared to that of a traditional cache. The improvements seen at the Level 2 memory system occur because each read access will take less energy and have a lower penalty. The greatest contributor though, is the leakage energy required by the Level 2 memory hierarchy. From the power and energy numbers in Table 3.4, it can be seen that the leakage energy is substantially lower for the scratchpad memories. In the case of "L2



Savings in level 1 memory
(a)

Savings in the entire on-chip system
(b)

Savings in level 2 memory
(c)

Savings in the entire on-chip system
(d)

Figure 4.11: Impacts on the power-delay product (PDP) and the energy-delay product (EDP) for scaled problem size SPMV multiplication.

SPM", a 2MB SPM is approximately 56% less than that of a 2MB cache. The total leakage power contribution will decrease as performance improves as well. Much of the improvement of the entire on-chip system results for a Level 2 SPM comes from the reduced leakage power of the scratchpad when compared to a large Level 2 cache. It is important to point out that the processing cores take up an increasingly large portion of the total power and energy requirements, while the Level 2 memory stays constant since it is shared. It is also important to point out that as the number of processors increases, the total on-chip Level 2 cache will increase and not stay constant as assumed in this work. This will improve the percentage of the total on-chip energy saved by using a scratchpad based Level 2 cache. Also the amount of data that can be quickly accessed increases with the use of a larger SPM. The results show that, in general, the most power and energy improvements will come from replacing the second level memory hierarchy of a multi-core system with a large scratchpad. As more cores are added to the system for massively parallel applications, increasing the size of the shared Level 2 SPM or having a scratchpad at the first level of the memory hierarchy for each core will likely continue to produce better energy results. As shown, adding scratchpads to the memory hierarchy will reduce the energy requirements needed by a system to compute matrix-vector multiplication.

Chapter 5

Conclusions and Future Work

5.1 Summary

In this work, I have demonstrated that SPMs can be used to improve the performance, energy consumption, area, and temperature of a processor running matrix-vector multiplication kernels. This work has also been detailed in previous publications by this author including a poster presented at the Supercomputing 2007 conference and a paper presented at the 2008 International Parallel and Distributed Processing Symposium [37; 38]. Scratchpad memories can be placed at all levels of the on-chip memory hierarchy to result in observable improvements. With the help of a compiler, read-only data can be detected and brought into the SPM through DMA transfers with minimal effort from the programmer. This would result in fast access to the data and very low miss rates. My experiments show that by using an SPM, the processor's average temperature can be reduced by as much as 1.6% and the area can be decreased by nearly 30%. Also, I can obtain an increase in performance by up to 60% for dense matrices and 41% for sparse matrices. In addition, the system wide energy-delay product would decrease by up to 84% for dense matrices and 67% for sparse matrices. Dedicating parts of the on-chip memory resources to an SPM instead of a traditional hardware-managed cache will make computer systems very attractive for the scientific computing community, especially as chips require more energy and run hotter.

5.2 Extensions

Future extensions of this work could include working with other scientific applications and kernels. This includes looking at applications that are well suited to be adapted to my architecture and also those that may not be as well suited. Some examples of such applications include n-body problems and image or signal processing kernels. I expect that the results would be similar to the results in this study. As the number of cores continues to increase, the energy and performance benefits of SPM would be clearly observable. Also, an extension could be to look at applications that have very large working data sets which may require scratchpads at both levels of the on-chip memory hierarchy. The trade-offs of on-chip capacity for SPM and cache are other topics. For this study, it would be interesting to go back and perform more scaling studies with scratchpads. In particular, seeing how SPMs benefit scaled sparse matrices when they are greatly increased in size the way dense matrices were in this study. Also, the opposite could be done by looking at not increasing the problem size of dense matrices as much.

Lastly, possible future work related to this topic includes looking at different architectures including reordering the layout of the chip and different architectures. The processor could be optimized for temperature instead of performance by moving some of the components around. Other work involves looking at the new space that was made available by the SPM. More scratchpad memory, traditional cache, or a new hardware feature could be added to this newly acquired space. Other interesting architecture

extensions include looking at a Non-Uniform Cache Architecture on a CMP. In addition, how an SPM benefits multi-core systems with private Level 2 memories could be examined.

Appendix A

DEC Alpha EV6 Architectures

The following figures are the six different performance optimized architectures that were examined for temperature and area improvements.

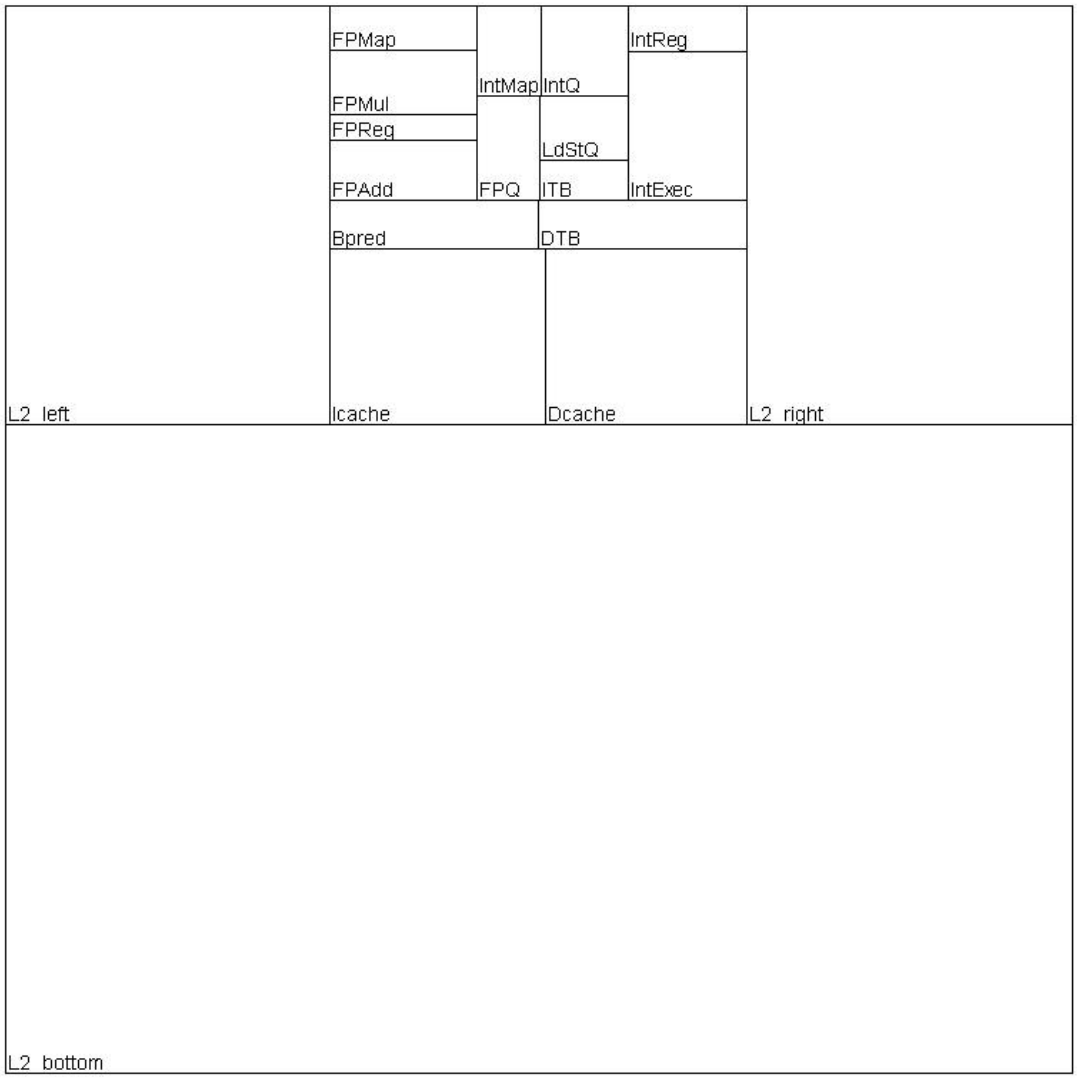


Figure A.1: Baseline Architecture.

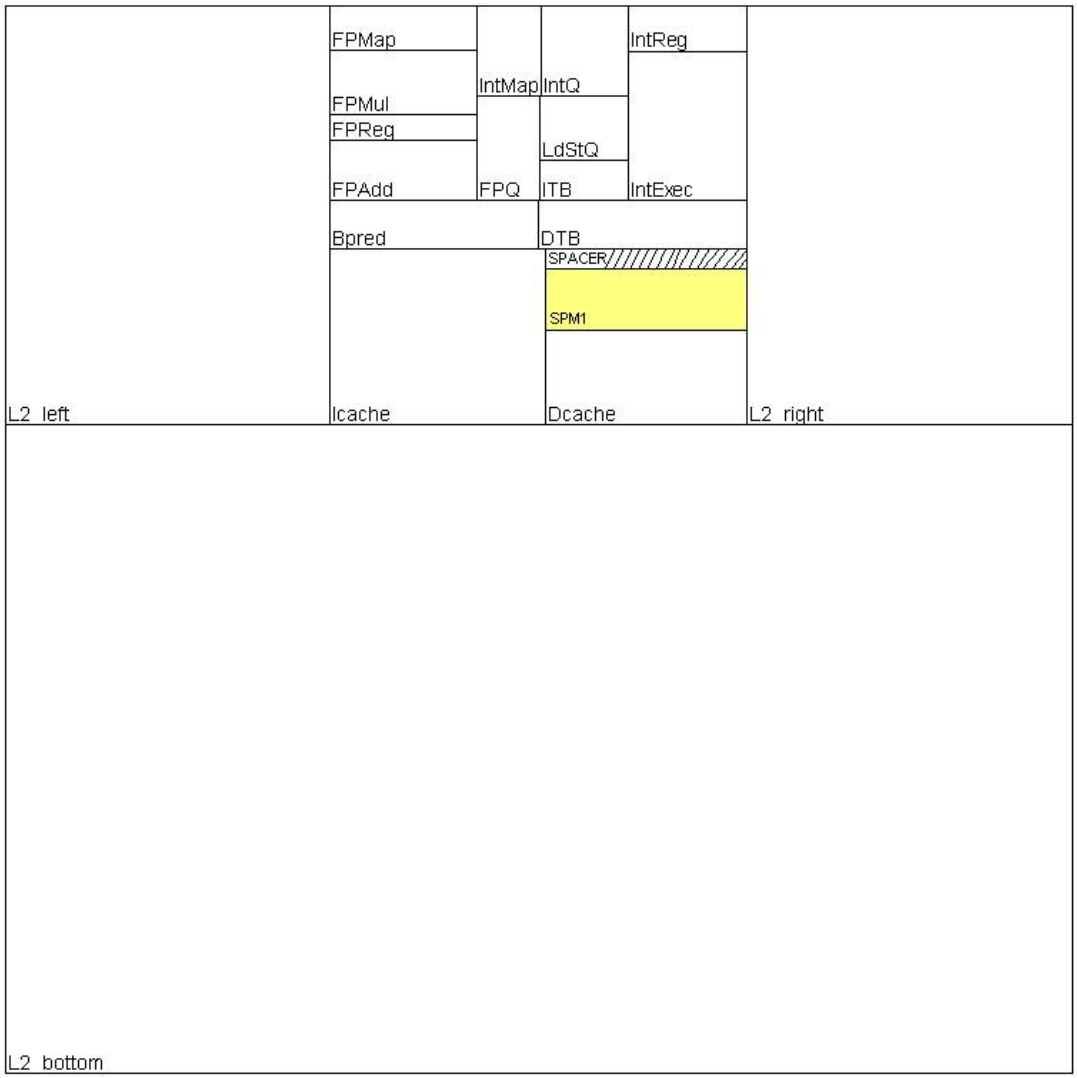


Figure A.2: Split L1D Architecture.

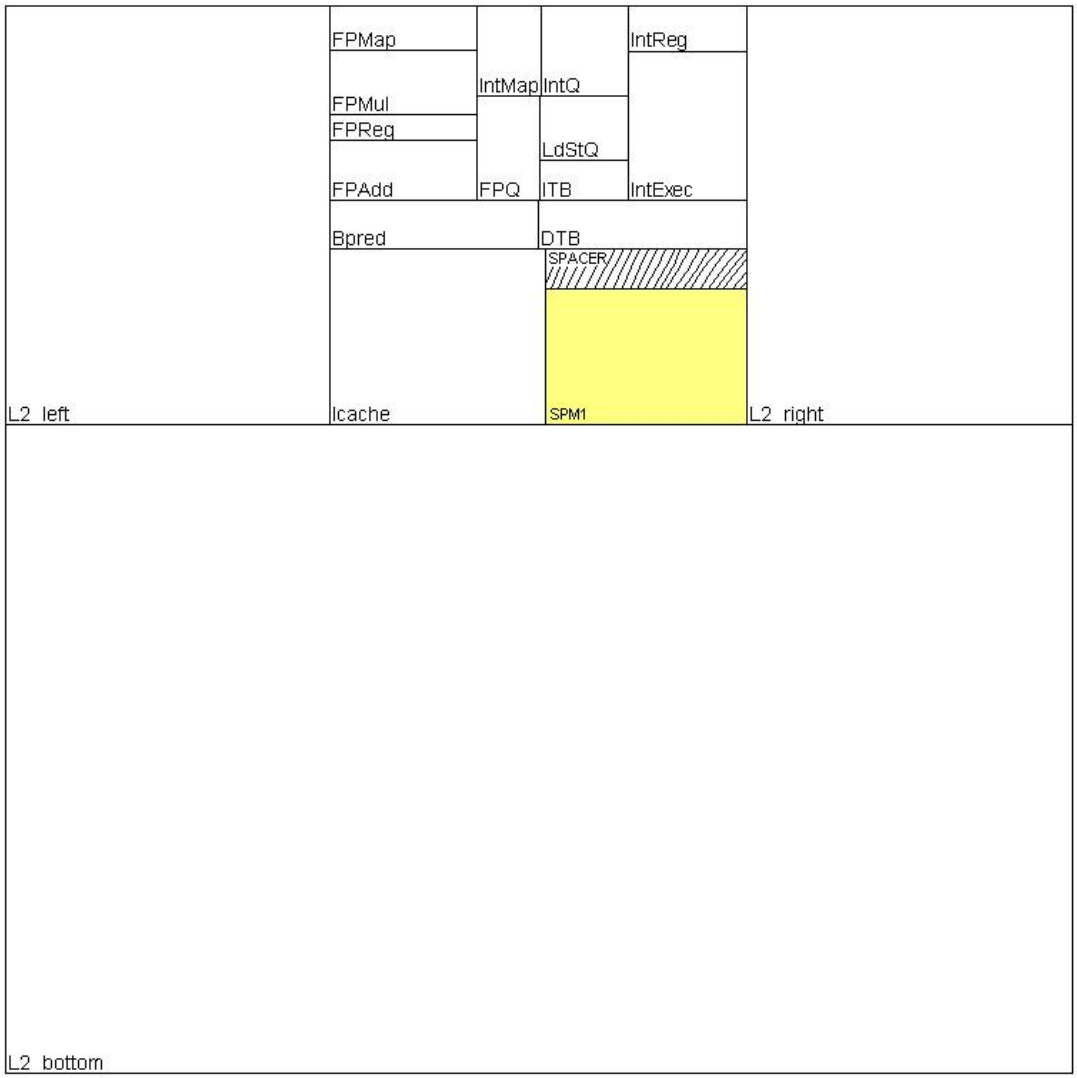


Figure A.3: L1D SPM Architecture.

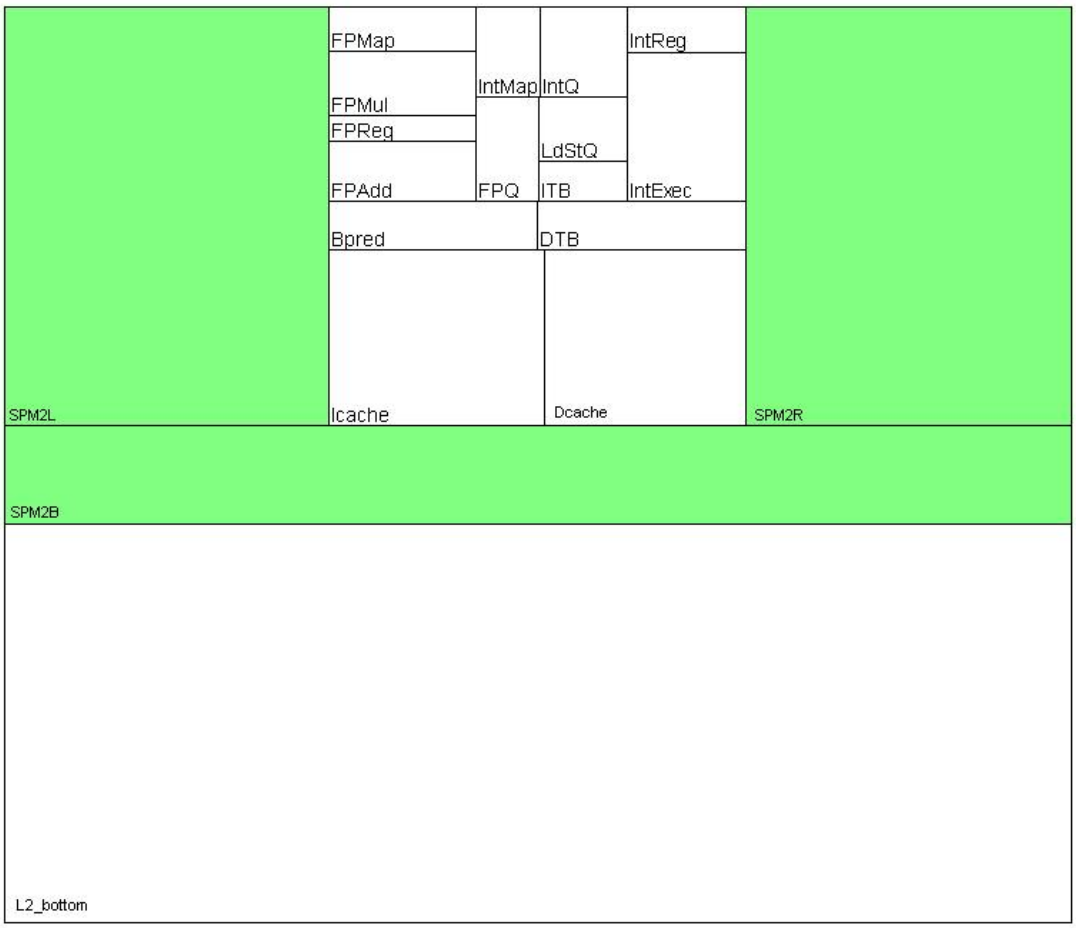


Figure A.4: Split L2 Architecture.

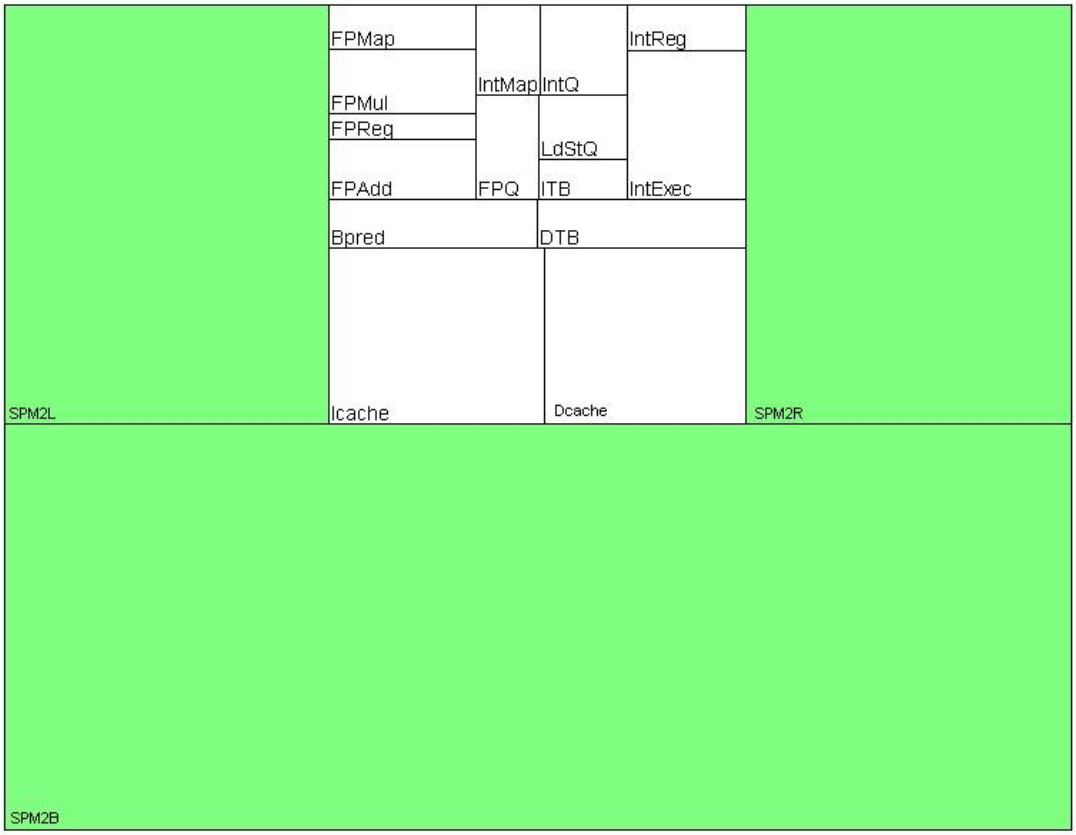


Figure A.5: L2 SPM Architecture.

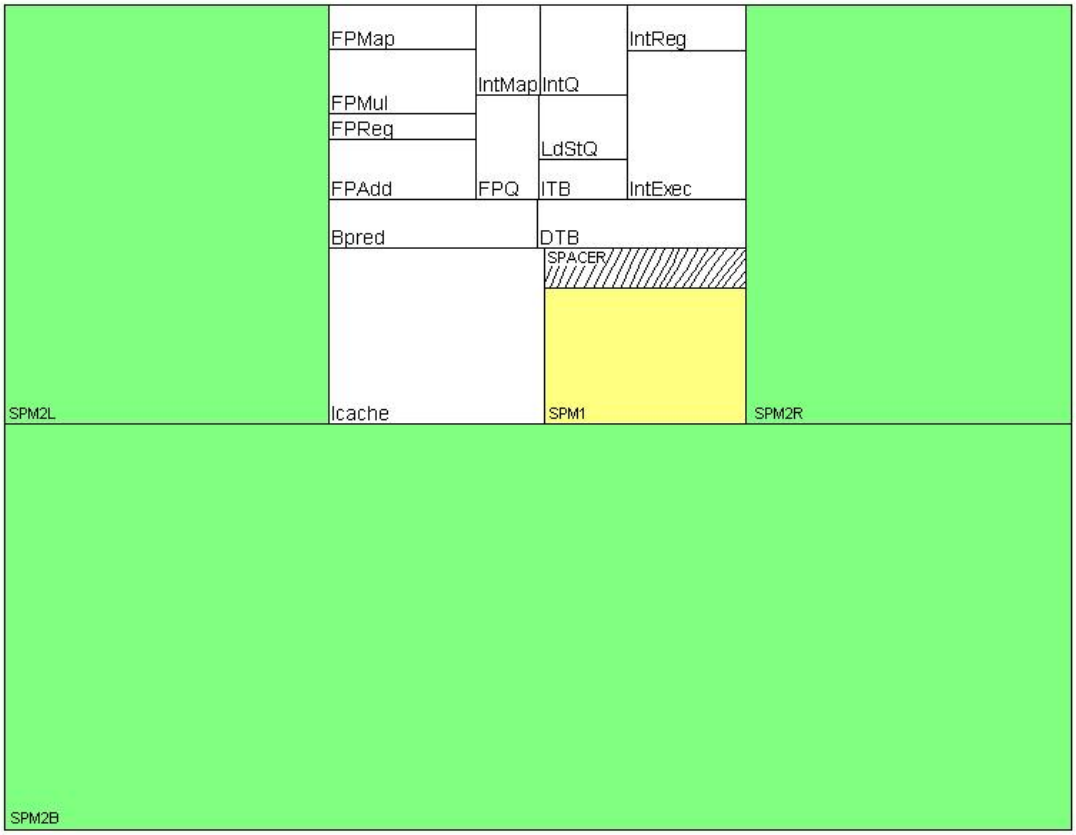


Figure A.6: ALL SPM Architecture.

Appendix B

Sparse Matrices Used

The following figures are visual representations of the matrices that were used in this study. First presented are the ten RCM reordered sparse matrices used for the sparse fixed problem size study followed by the five sparse matrices used for the sparse scaled problem size study.

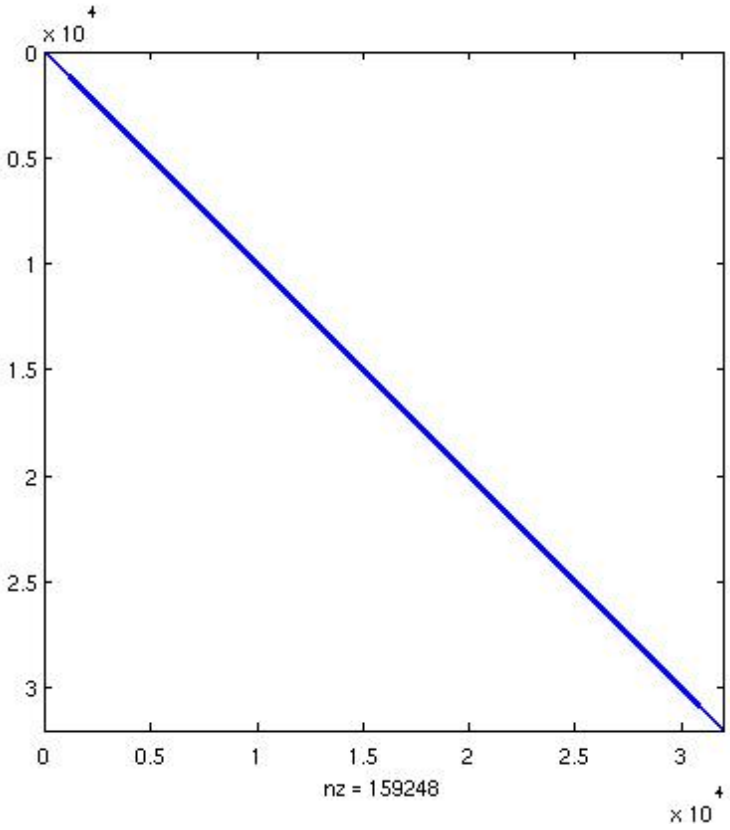


Figure B.1: RCM reordered fdm2 matrix.

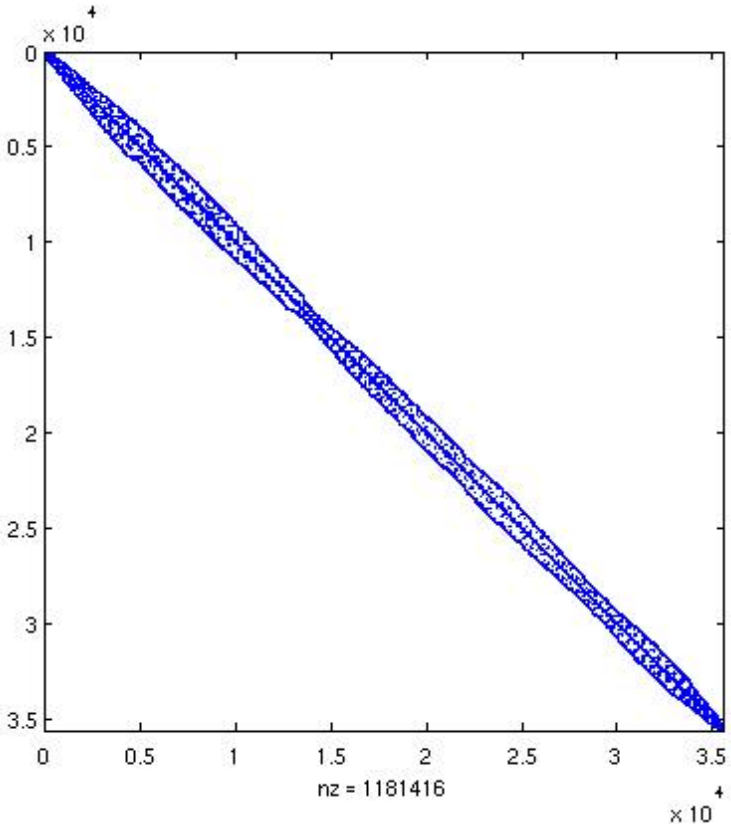


Figure B.2: RCM reordered bcsstk31RCM matrix.

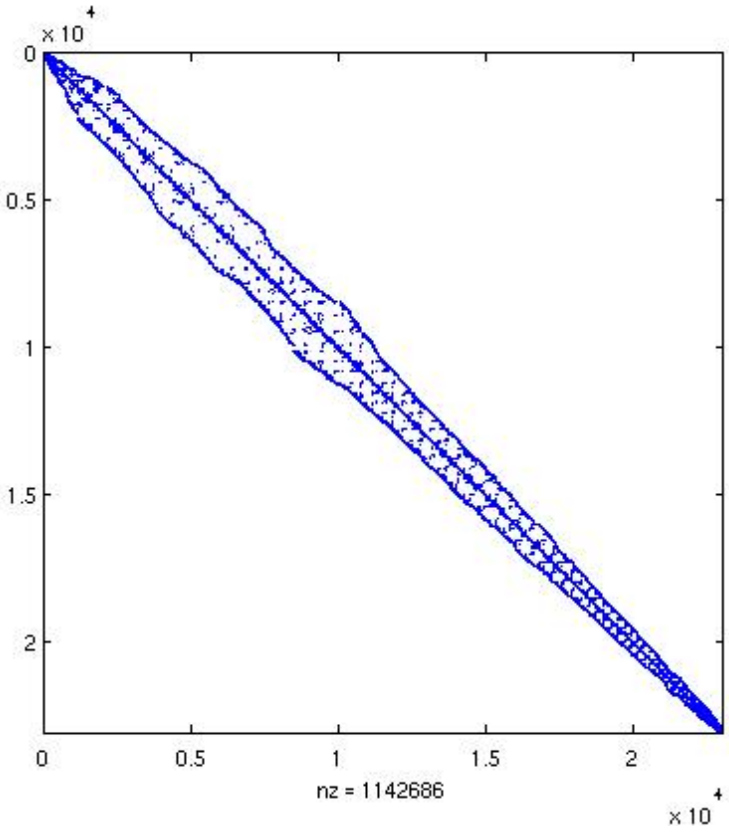


Figure B.3: RCM reordered msc23052 matrix.

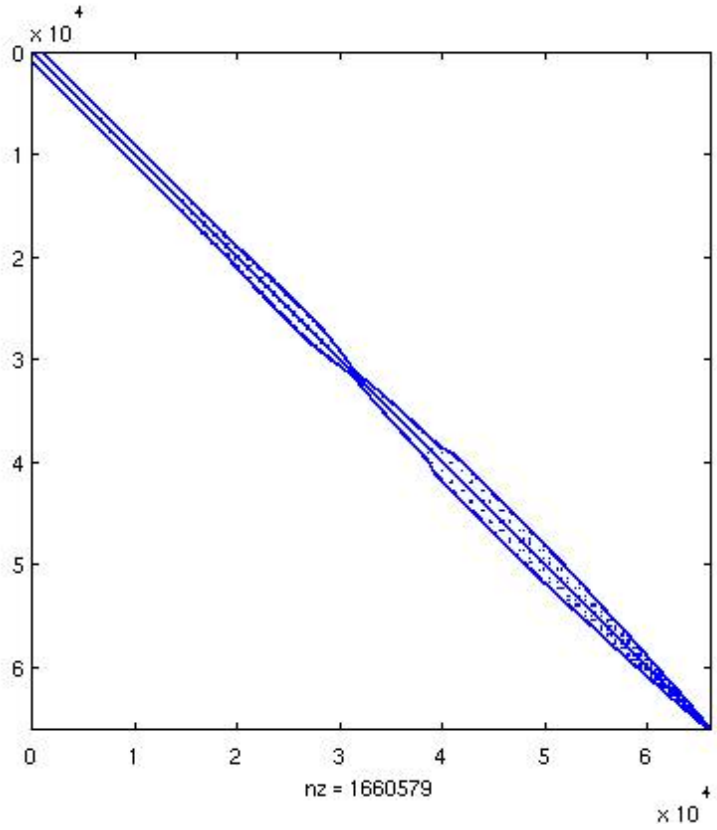


Figure B.4: RCM reordered qa8fm matrix.

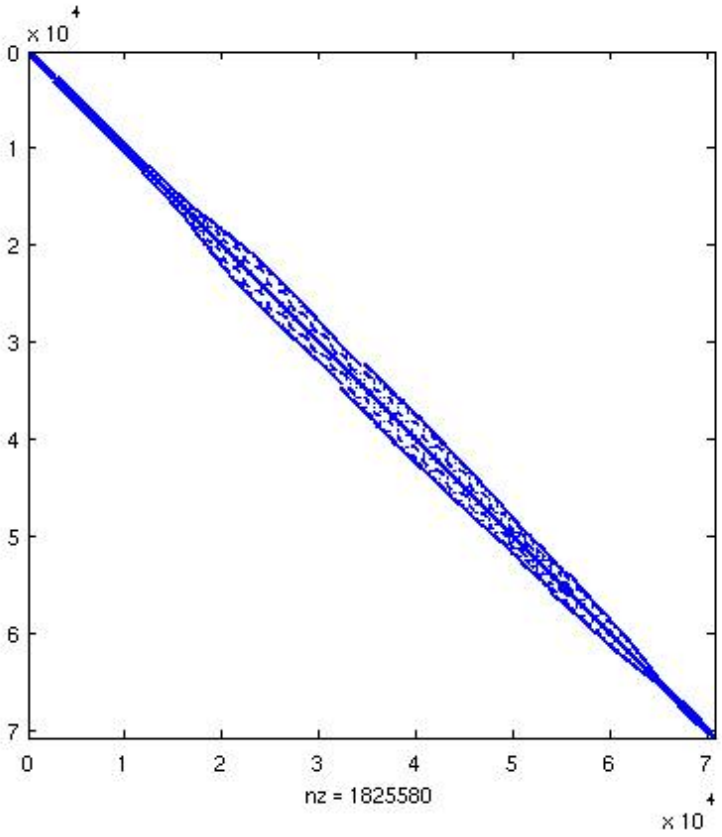


Figure B.5: RCM reordered cfd1 matrix.

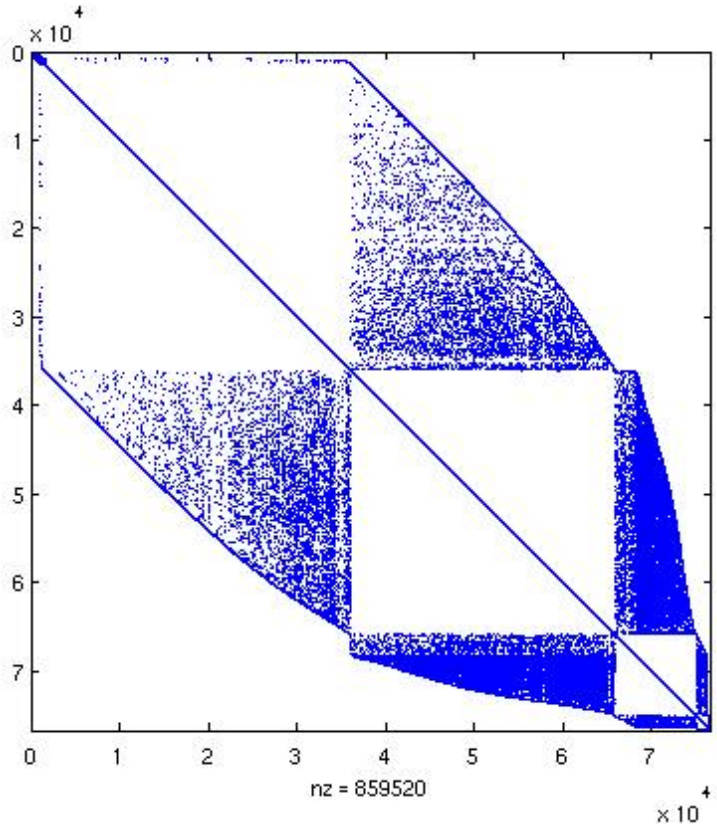


Figure B.6: RCM reordered c-71 matrix.

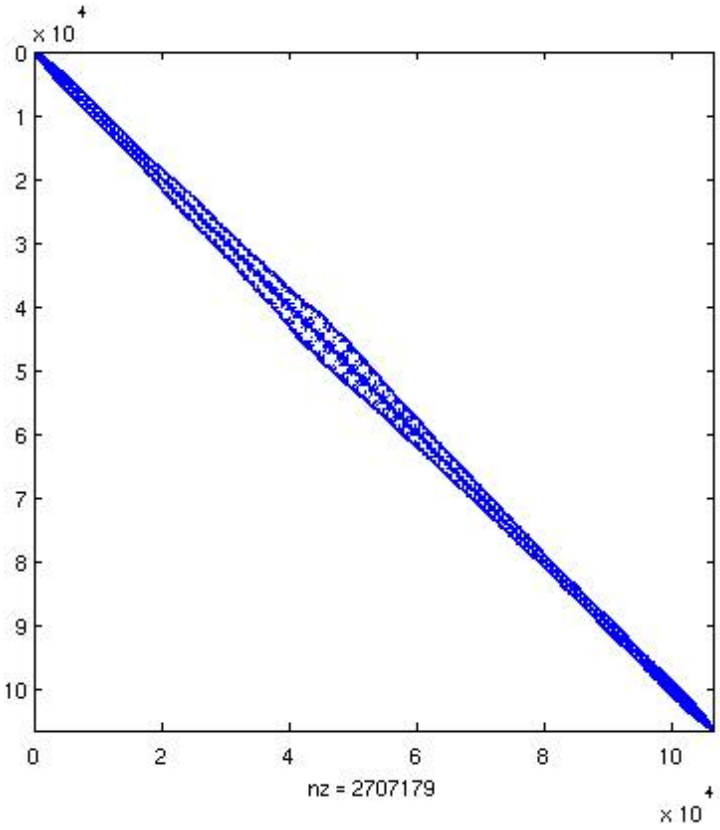


Figure B.7: RCM reordered filter3D matrix.

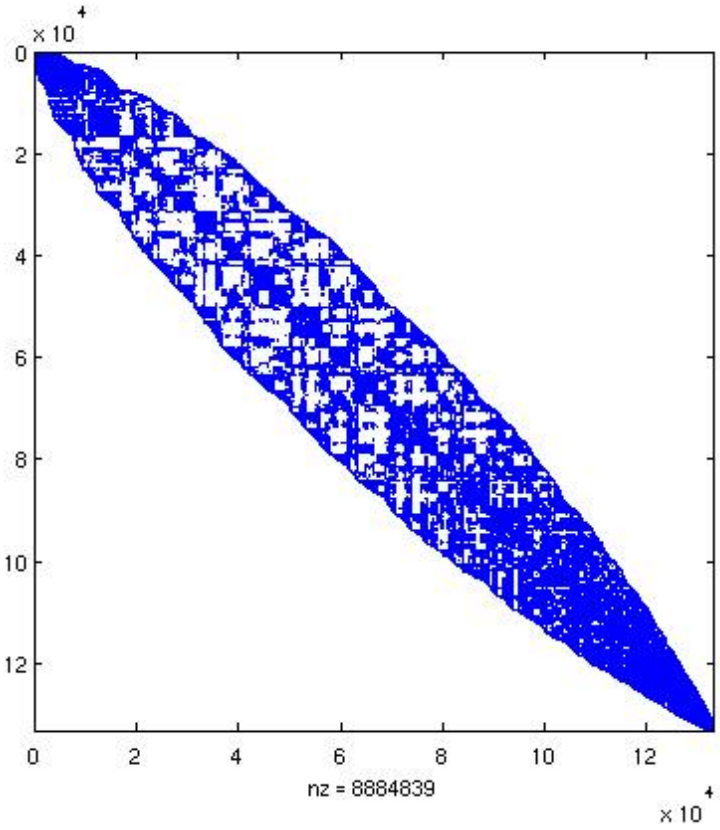


Figure B.8: RCM reordered Ga19As19H42 matrix.

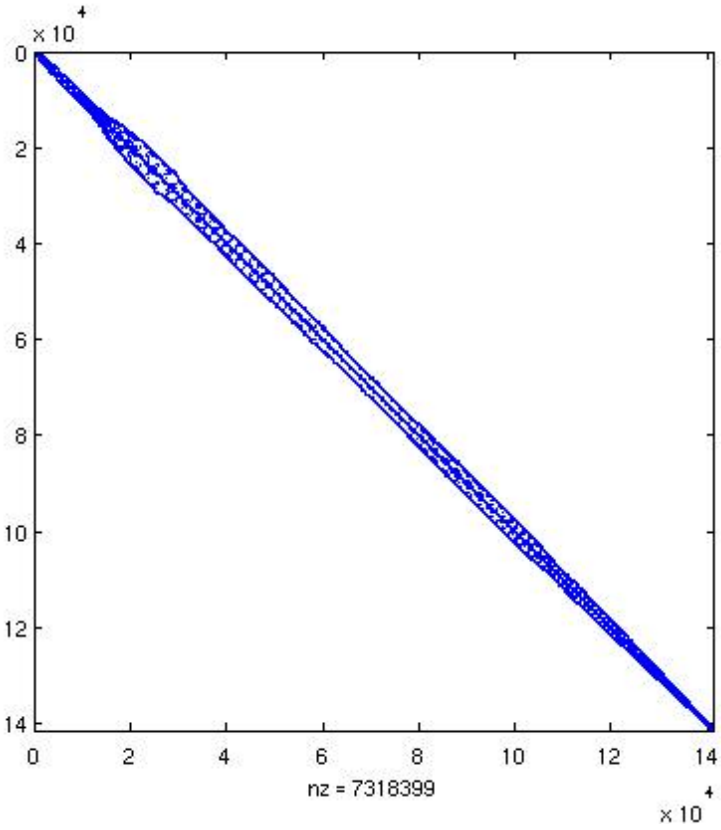


Figure B.9: RCM reordered bmw7st_1 matrix.

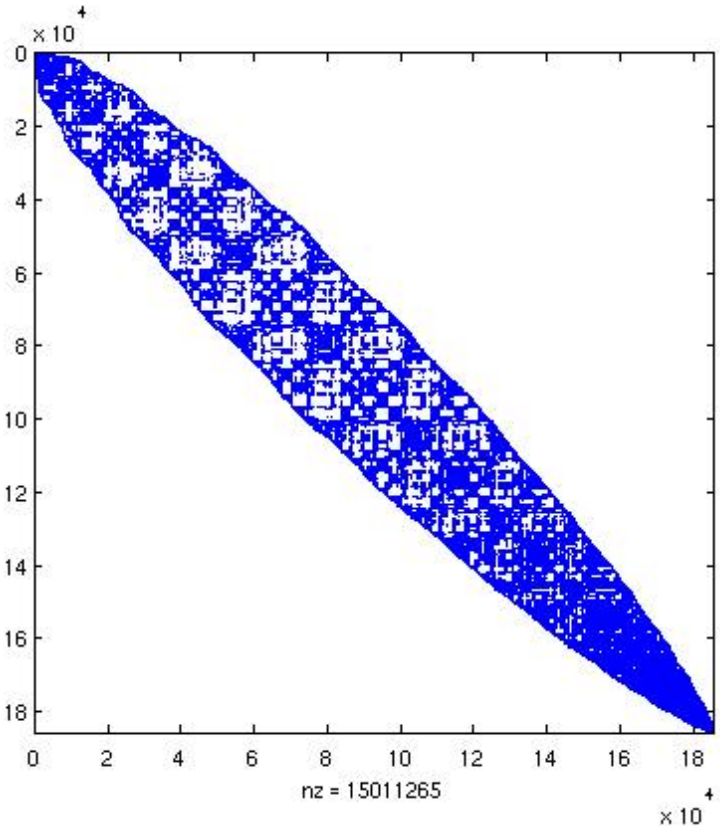


Figure B.10: RCM reordered Si41Ge41H72 matrix.

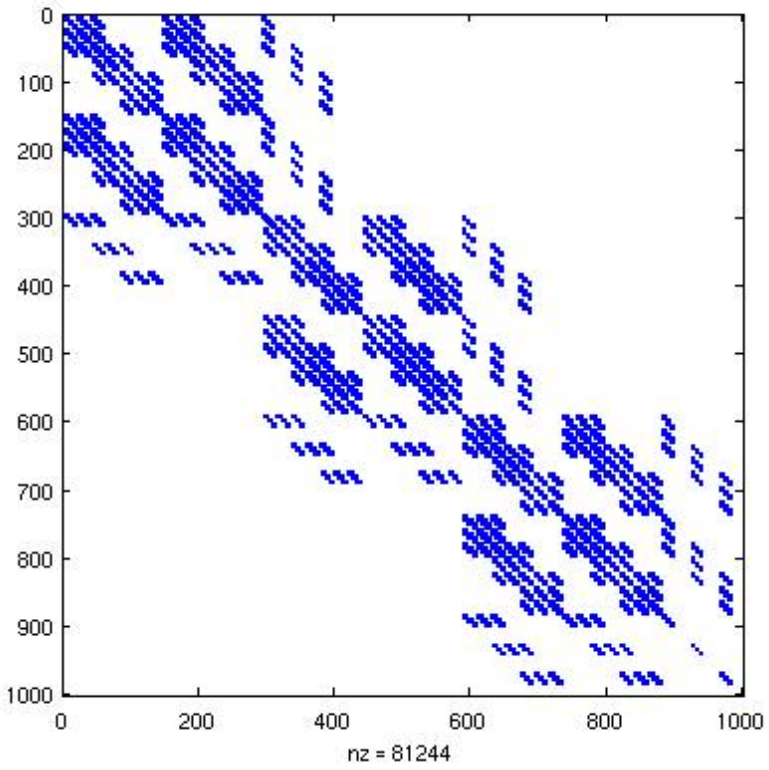


Figure B.11: 1000x1000 elasticity matrix used to calculate the stiffness for a brick in three dimensions.

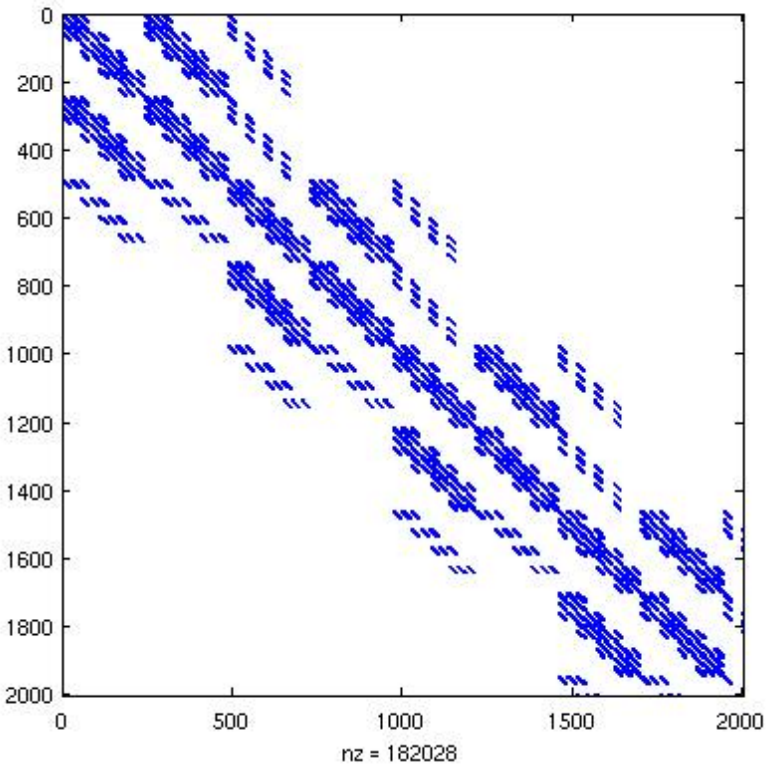


Figure B.12: 2000x2000 elasticity matrix used to calculate the stiffness for a brick in three dimensions.

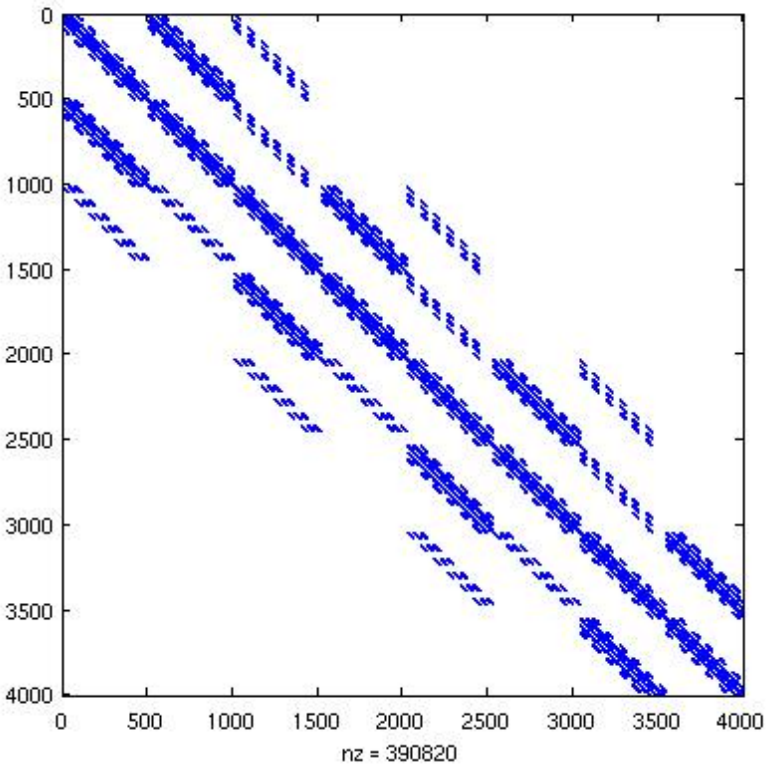


Figure B.13: 4000x4000 elasticity matrix used to calculate the stiffness for a brick in three dimensions.

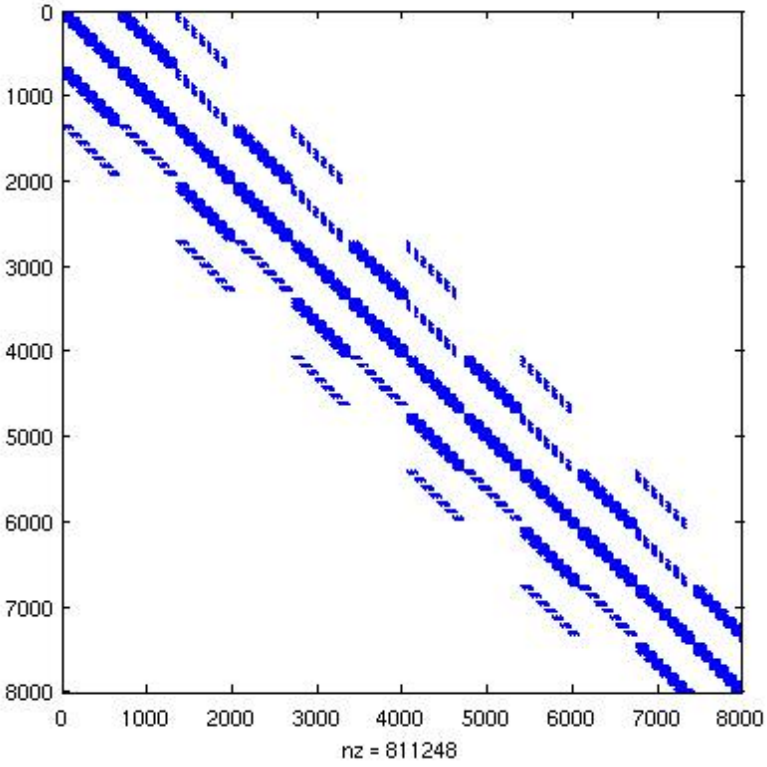


Figure B.14: 8000x8000 elasticity matrix used to calculate the stiffness for a brick in three dimensions.

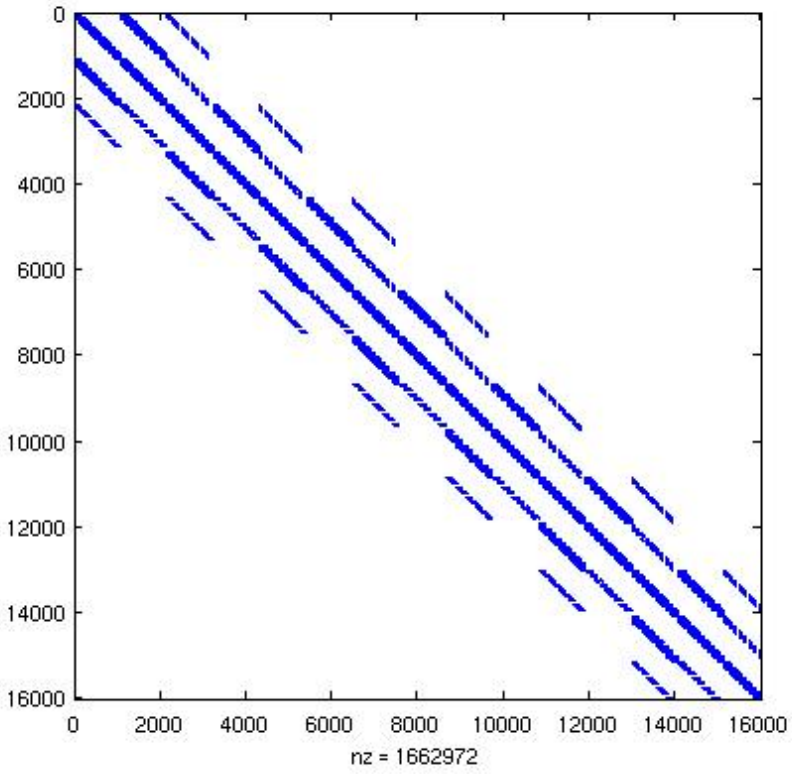


Figure B.15: 16000x16000 elasticity matrix used to calculate the stiffness for a brick in three dimensions.

Bibliography

[1] F. Angiolini, L. Benini, and A. Caprara. Polynomial-time algorithm for on-chip scratchpad memory partitioning. In *CASES '03: Proceedings of the 2003 International Conference on Compilers, rchitecture and Synthesis for Embedded Systems*, pages 318–326, 2003.

[2] R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, and P. Marwedel. Scratchpad memory: design alternative for cache on-chip memory in embedded systems. In *CODES '02: Proceedings of the Tenth International Symposium on Hardware/Software Codesign*, pages 73–78, 2002.

[3] L. Benini, A. Macii, E. Macii, and M. Poncino. Increasing energy efficiency of embedded systems by application-specific memory hierarchy generation. *IEEE Des. Test*, 17(2):74–85, 2000.

[4] G. Chen and M. Kandemir. Dataflow analysis for energy-efficient scratch-pad memory management. In *ISLPED '05: Proceedings of the 2005 International Symposium on Low Power Electronics and Design*, pages 327–330, 2005.

[5] J. Chen, P. Juang, K. Ko, G. Contreras, D. Penry, R. Rangan, A. Stoler, L.-S. Peh, and M. Martonosi. Hardware-modulated parallelism in chip multiprocessors. *SIGARCH Comput. Archit. News*, 33(4):54–63, 2005.

[6] D. Chiou, P. Jain, L. Rudolph, and S. Devadas. Application-specific memory management for embedded systems using software-controlled caches. In *DAC '00: Proceedings of the 37th Conference on Design Automation*, pages 416–419, 2000.

[7] A. Dominguez, S. Udayakumaran, and R. Barua. Heap data allocation to scratch-pad memory in embedded systems. *J. Embedded Comput.*, 1(4):521–540, 2005.

[8] I. Duff, A. Erisman, and J. Reid. *Direct Methods for Sparse Matrices*. Oxford Science Publications, 1986.

[9] B. Egger, C. Kim, C. Jang, Y. Nam, J. Lee, and S. L. Min. A dynamic code placement technique for scratchpad memory using postpass optimization. In *CASES '06: Proceedings of the 2006 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 223–233, 2006.

[10] B. Egger, J. Lee, and H. Shin. Scratchpad memory management for portable systems with a memory management unit. In *EMSOFT '06: Proceedings of the 6th ACM & IEEE International Conference on Embedded software*, pages 321–330, 2006.

[11] V. K. et al. *Introduction to parallel computing. Design and analysis of algorithms*. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

[12] N. E. Gibbs, J. William G. Poole, and P. K. Stockmeyer. A comparison of several bandwidth and profile reduction algorithms. *ACM Trans. Math. Softw.*, 2(4):322–330, 1976.

[13] R. Gupta and C.-H. Chi. Improving instruction cache behavior by reducing cache pollution. In *Supercomputing '90: Proceedings of the 1990 ACM/IEEE conference*

on Supercomputing, pages 82–91, Washington, DC, USA, 1990. IEEE Computer Society.

[14] W. Huang, K. Sankaranarayanan, R. J. Ribando, M. R. Stan, and K. Skadron. An improved block-based thermal model in hotspot-4.0 with granularity considerations. In *ISCA '07: Proceedings of the Workshop on Duplicating, Deconstructing, and Debunking, in conjunction with the 34th International Symposium on Computer Architecture*, 2007.

[15] I. Issenin, E. Brockmeyer, B. Durinck, and N. Dutt. Multiprocessor system-on-chip data reuse analysis for exploring customized memory hierarchies. In *DAC '06: Proceedings of the 43rd annual Conference on Design Automation*, pages 49–52, 2006.

[16] A. Janapsatya, A. Ignjatovi, and S. Parameswaran. A novel instruction scratchpad memory optimization method based on concomitance metric. In *ASP-DAC '06: Proceedings of the 2006 Conference on Asia South Pacific Design Cutomation*, pages 612–617, 2006.

[17] J. Kahle. The cell processor architecture. In *MICRO 38: Proceedings of the 38th Annual IEEE/ACM International Symposium on Microarchitecture*, page 3, 2005.

[18] M. Kandemir, M. J. Irwin, G. Chen, and I. Kolcu. Banked scratch-pad memory management for reducing leakage energy consumption. In *ICCAD '04: Proceedings of the 2004 IEEE/ACM International Conference on Computer-Aided Design*, pages 120–124, 2004.

[19] M. Kandemir, I. Kadayif, and U. Sezer. Exploiting scratch-pad memory using presburger formulas. In *ISSS '01: Proceedings of the 14th International Symposium on Systems Synthesis*, pages 7–12, 2001.

[20] M. Kandemir, J. Ramanujam, and A. Choudhary. Exploiting shared scratch pad memory space in embedded multiprocessor systems. In *DAC '02: Proceedings of the 39th Conference on Design Automation*, pages 219–224, 2002.

[21] M. T. Kandemir. Reducing energy consumption of multiprocessor soc architectures by exploiting memory bank locality. *ACM Trans. Des. Autom. Electron. Syst.*, 11(2):410–441, 2006.

[22] J. Leverich, H. Arakida, A. Solomatnikov, A. Firoozshahian, M. Horowitz, and C. Kozyrakis. Comparing memory systems for chip multiprocessors. *SIGARCH Comput. Archit. News*, 35(2):358–368, 2007.

[23] J. E. Miller and A. Agarwal. Software-based instruction caching for embedded processors. In *ASPLOS-XII: Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 293–302, 2006.

[24] S. S. Mukherjee, J. Emer, and S. K. Reinhardt. The soft error problem: An architectural perspective. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 243–247, Washington, DC, USA, 2005. IEEE Computer Society.

[25] O. Ozturk, M. Kandemir, I. Demirkiran, G. Chen, and M. J. Irwin. Data compression for improving spm behavior. In *DAC '04: Proceedings of the 41st annual Conference on Design Automation*, pages 401–406, 2004.

[26] P. R. Panda, N. D. Dutt, and A. Nicolau. Efficient utilization of scratch-pad memory in embedded processor applications. In *EDTC '97: Proceedings of the 1997 European Conference on Design and Test*, page 7, 1997.

[27] J. Rabaey, A. Chandrakasan, and B. Nikolic. *Digital Integrated Circuits: A Design Perspective*. Prentice Hall, 1993.

[28] Y. Saad. *Iterative methods for Sparse Linear Systems*. SIAM, 2003.

[29] S. Steinke, L. Wehmeyer, B. Lee, and P. Marwedel. Assigning program and data objects to scratchpad for energy reduction. In *DATE '02: Proceedings of the Conference on Design, Automation and Test in Europe*, page 409, 2002.

[30] D. Tarjan and N. P. J. Shyamkumar Thoziyaar. Cacti 4.0. Technical Report HPL-2006-86, Palo Alto, June 2006.

[31] S. Udayakumaran and R. Barua. Compiler-decided dynamic memory allocation for scratch-pad based embedded systems. In *CASES '03: Proceedings of the 2003 International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 276–286, 2003.

[32] M. Verma, S. Steinke, and P. Marwedel. Data partitioning for maximal scratchpad usage. In *ASPDAC: Proceedings of the 2003 conference on Asia South Pacific Design Automation*, pages 77–83, 2003.

[33] M. Verma, L. Wehmeyer, and P. Marwedel. Cache-aware scratchpad allocation algorithm. In *DATE '04: Proceedings of the conference on Design, Automation and Test in Europe*, page 21264, 2004.

[34] C. Weaver, J. Emer, S. S. Mukherjee, and S. K. Reinhardt. Techniques to reduce the soft error rate of a high-performance microprocessor. In *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture*, page 264, Washington, DC, USA, 2004. IEEE Computer Society.

[35] J. Willcock and A. Lumsdaine. Accelerating sparse matrix computations via data compression. In *ICS '06: Proceedings of the 20th annual International Conference on Supercomputing*, pages 307–316, 2006.

[36] S. Williams, J. Shalf, L. Oliker, S. Kamil, P. Husbands, and K. Yelick. The potential of the cell processor for scientific computing. In *CF '06: Proceedings of the 3rd Conference on Computing Frontiers*, pages 9–20, 2006.

[37] A. Yanamandra, B. Cover, P. Raghavan, M. J. Irwin, and M. Kandemir. Evaluating the role of scratchpad memories in chip multiprocessors for sparse matrix computations. In *IPDPS '08: Proceedings of the 2008 IEEE International Parallel and Distributed Processing Symposium*, Washington, DC, USA, 2008. IEEE Computer Society.

[38] A. Yanamandra, B. Cover, P. Raghavan, M. J. Irwin, M. Kandemir, and K. Malkowski. Evaluating the role of scratchpad memories in multi-core for sparse

matrix computations. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, Washington, DC, USA, 2007. IEEE Computer Society.