Matrix Multiplication Research Based on Gem5 Simulation

Junchen Lu [1, *]

[1] Shanghai Jiao Tong University, Shanghai, China
* Corresponding author: lujunchen2020@sjtu.edu.cn

Abstract: Matrix multiplication is critical to modern scientific computation, therefore it is meaningful to improve its performance. Blocking technique is a popular method to enhance locality and reduce cache miss ratio significantly. Several sparse storing formats are possibly better than two dimensional array for sparse matrices. And finding a more suitable replacement policy is another direction for optimization. In this research, a simulator of set associative cache is built based on Gem5 so that cache behaviour can be better observed. Several topics such as ideal parameters setting for blocking technique, better storing formats for sparse matrices and three replacement policies, LRU, MRU, and Bimodal Insertion Policy, are investigated and related empirical evidence has been provided. Based on the statistics collected from the experiments, it can be observed that bigger blocks, CSR or CSC storing format for sparse matrices and a combination of LRU and MRU replacement policies are beneficial for improvement in cache performance and lowering down cache miss ratio.

Keyword： Matrix multiplication, Gem5, Compressed sparse row, Compressed sparse column

1. Introduction

1.1. Research background

Matrix manipulations play a significant role in modern scientific computation. Take MATLAB, a programming and numeric computing platform, as an example. MATLAB stands for "matrix laboratory" and as the name suggests, this platform is designed specifically for arrays and matrices and billions of matrix manipulations are executed all the time [1]. Besides MATLAB, matrix is a critical element that often appears in various algorithms related to Linear Algebra, Artificial Neural Networks, Signal Processing and so on. Therefore, if the performance of matrix manipulations can be improved to some extent, plenty of computing resources and run time can be saved [2].

Also, matrices are not necessarily stored as two dimensional format and some special, interesting and practical storing formats, such as COO(coordinate list), CSR(compressed sparse row) and CSC(compressed sparse column), have been proposed for the storage of sparse matrices. Indeed, it would be meaningless to store plenty of zeros in sparse matrices intuitively speaking and less elements in the storage system means less time required to access the data [3].

Based on this consideration, this paper focuses on improving the performance of matrix manipulations by making matrix multiplication more cache friendly. The storage system of modern computers is always hierarchical with cache acting as a proxy between processors and DRAM. The time it takes CPU to access data stored in cache is approximately between 10 to 100 times faster than DRAM [4]. Therefore, if a matrix multiplication algorithm can be developed so that a greater proportion of hot data, which means data that needs to be accessed repeatedly, can be identified and

stored in cache, cache can be better utilized during the run time, which can be interpreted as higher cache hit ratio and lower cache miss ratio and the predicted run time of multiplication can be reduced significantly.

For those sparse matrices' special storing formats, they can also be possibly tuned for better cache utilization. Also, it is meaningful to figure out the best storing method for a given matrix operation.

1.2. Literature review

Optimization of matrix multiplication in terms of cache utilization has gained the attention of some scholars. Lam et al. Proposed blocking technique which changes the sequence in which elements of matrices are visited [5]. Within the blocking technique, block size is a critical factor which can impose noticeable influence on cache miss rate, so S. Ristov et al. did some research to find the optimized value [6]. The topic of optimizing matrix multiplication is integrated with parallel computing as modern multi-core CPU and GPU architectures prevail.

Among most of the research paper in this field, experiments are operated on certain types of Linux machines instead of a simulator. There are two potential shortcomings to do experiments on real machines. Firstly, there are always multiple applications and processes running on the machine simultaneously which can possibly contaminate the data collected from system [7]. Secondly, these machines are likely to be equipped with some advanced mechanisms to reduce cache miss rate such as predicting and prefetching [8]. Therefore, it is not easy to determine the influence factor of results and data might deviate from theoretical value. In view of the two reasons, a simulator based on Gem5 will be built after which all of the experiments will be operated on the simulator.

1.3. Overview of the paper

In this paper, the second section will mainly focus on the tools and methods to be used in later experiments and in this section, a detailed explanation will be provided as to how to set up the experiment environment and which variables are the research interests [9]. The third section will primarily focus on the design of experiments and the expected results after which the fourth section will report the experiment results with detailed analysis. In the last section, major conclusions will be drawn and several potential directions for future research will be listed [10].

2. Tools and methods

2.1. The chosen simulator: Gem5

Gem5 is an open source project which provides emulation for researches on computer architecture. It can focus on the behaviour of CPU and cache, and it can also simulate the whole computer system with a large number of components and devices. Compared to other types of simulators, Gem5 is still unique and has some outstanding characteristics.

Firstly, the simulated system is described in Python scripts which can be easily modified. Secondly, the codes for each component are open-source, therefore users can modify or define their own components to replace the original one in the system. What's more, Gem5 has provided users with statistics tools which make the running process transparent to the users. Last but not least, Gem5 is maintained by community and updated frequently, due to which most potential bugs will be fixed in time and new features will be added constantly.

2.2. Gem5 syscall emulation(SE) mode

The simulator, Gem5, has two different modes, SE(Syscall Emulation) mode and FS(Full System) mode. SE mode will merely simulate CPU, cache and DRAM, which possesses a simpler structure and faster emulation speed; while FS mode resembles a complete Linux operating system and, thus, can provide a more accurate simulation of the whole system. SE mode is already enough for the research since cache is the primary interests and other components in FS mode night be somewhat distracting.

The run-time logic of Gem5 SE mode can be seen in the following picture [3] which is also the programming model of simulation. As shown in Figure 1.

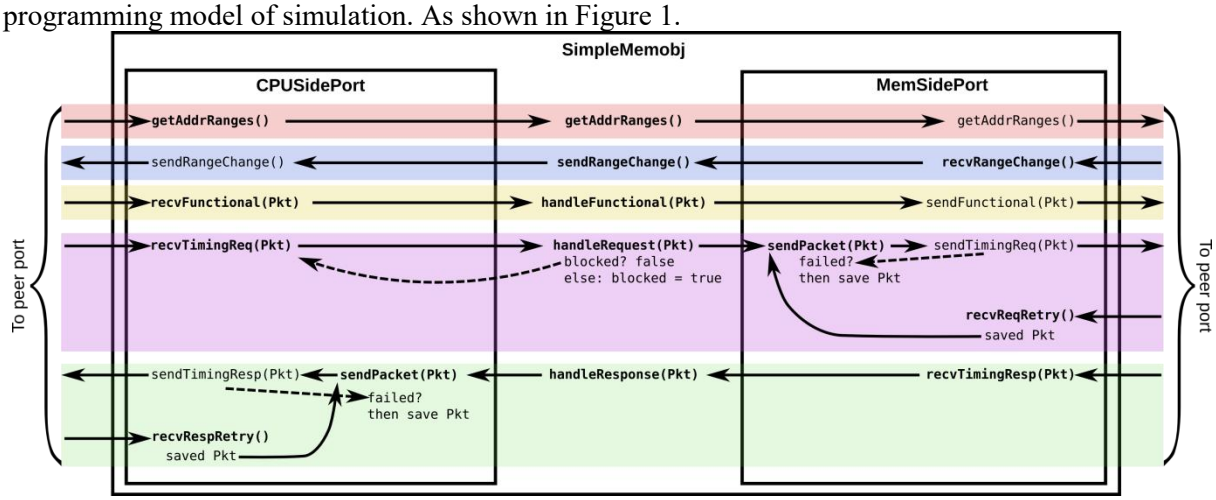

Figure 1. Run-time logic of Gem5 SE mode

2.3. Set associative cache

Actually, Gem5 is a open-source project and it has offered a built-in cache implementation. Nevertheless, this built-in cache is merely represented by an unordered map which is simple and crude. Unordered map cannot simulate the complicated behaviour of cache in modern computers. The basic architecture of set associative cache can be seen in Figure 2.

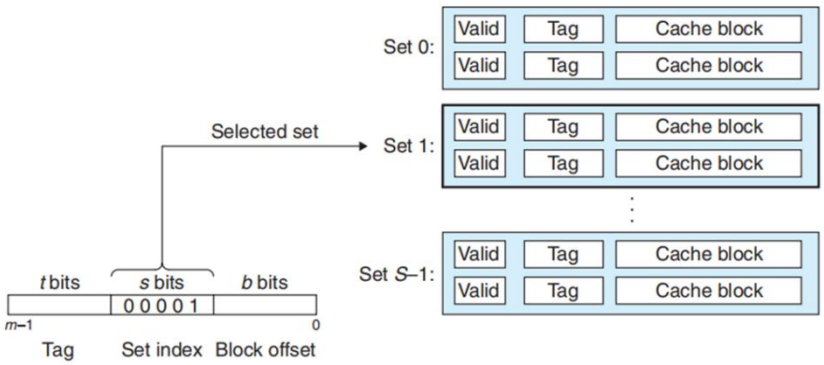

Figure 2. Basic architecture of set associative cache

A given address will be parsed into three segments which are tag, set index and block offset. Cache will locate a certain set based on the ‘set index’ parsed from the address, after which a certain cache line will be picked if its valid bit is set to true and tag is identical with the one parsed from the address. The class diagram of set associative cache can be designed in the following manner according to which it is implemented in C++.As shown in Figure 3.

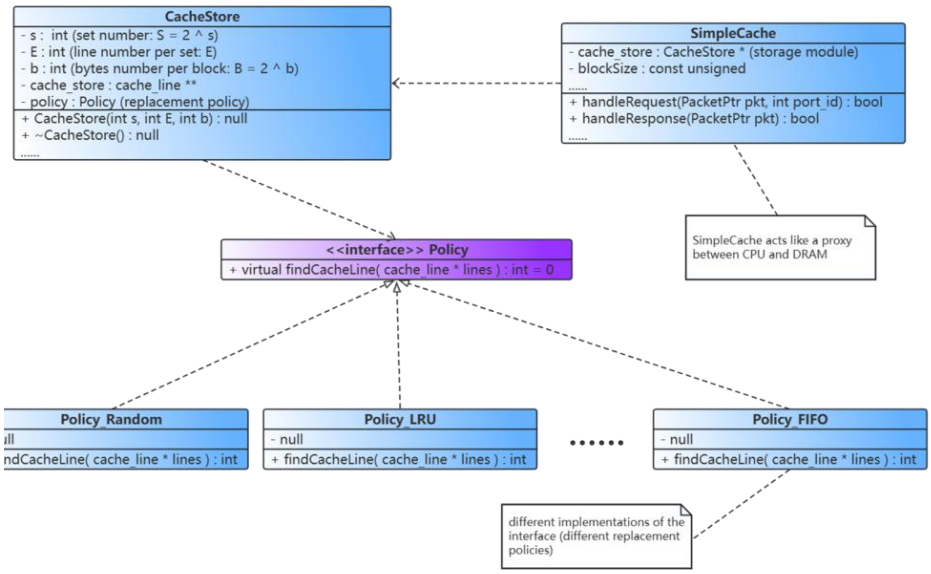

Figure 3. Class diagram of set associative cache

2.4. Replacement policies: LRU, MRU and BIP

Besides the delicate inner structure of cache, replacement policy is another interesting topic and it is the policy based on which the victim cache line is chosen when the set is full. Even if LRU(Least Recently Used) is the most widely used policy, it might not be the best option in every scenario. For example, when scanning every element in the matrix, those elements will evict all of the hot data stored in the cache previously but they will only be visited for once. This special scanning phenomenon will result in the loss of all the hot data and store 'fake' hot data in the cache.

Under this circumstance, Bimodal Insertion Policy(BIP) can relieve this phenomenon and make the cache more scan resistant. BIP is a combination of LRU and MRU(Most Recently Used) and, therefore, it possesses the strengths of both LRU and MRU. In BIP, the possibility that LRU being applied is p while the possibility that MRU being applied is (1-p). MRU brings about a chance that 'fake' hot data from scanning operations will be evicted out of cache immediately so that real hot data will not be affected.

2.5. Blocking in matrix multiplication

After introducing the detailed information about hardware side, matrix multiplication on the software side will become the focus in these two parts. The algorithm is modified to increase the locality of matrices using blocking technique. Instead of traversing all the rows or columns at the same time, blocking technique will load part of rows and update the values of all the elements which need to read values from these rows. Therefore, after these rows are evicted into DRAM, they will never be loaded again. The pseudo code of naive implementation and the implementation with blocking technique can be seen in Figure 4.

```
/* Before */
for (i = 0; i < N; i = i + 1)
      for (j = 0; j < N; j = j + 1)
           {r = 0;
            for (k = 0; k < N; k = k + 1)
                 r = r + y[i][k]*z[k][j];
            x[i][j] = r;
           };
/* After */
for (jj = 0; jj < N; jj = jj + B)
for (kk = 0; kk < N; kk = kk + B)
for (i = 0; i < N; i = i + 1)
      for (j = jj; j < min(jj + B,N); j = j + 1)
          {r = 0;
           for (k = kk; k < min(kk + B,N); k = k + 1)
                r = r + y[i][k]*z[k][j];
           x[i][j] = x[i][j] + r;
          };
```

Figure 4. Implementations with and without blocking technique

2.6. Various storing formats of sparse matrices

The algorithm of blocking technique is totally based on matrices stored in two dimensional format. However, there are various storing formats of sparse matrices which are much more space efficient than two dimensional format. For sparse matrices, considering that most of elements in the matrix are zeros and it would be meaningless to store plenty of zeros. Several typical types of sparse storing formats are demonstrated in Figure 5 and they are COO(coordinate list), CSR(compressed sparse row) and CSC(compressed sparse column).

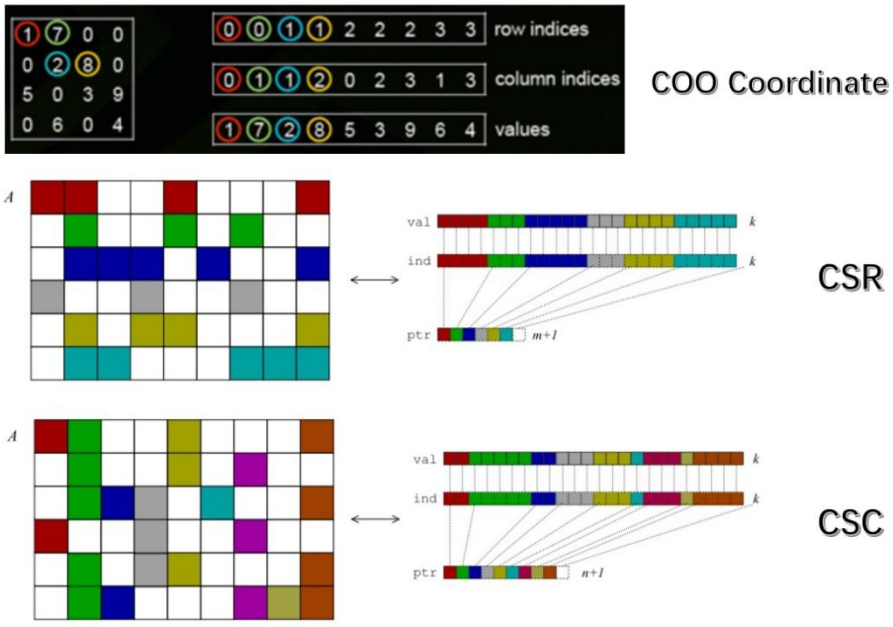

Figure 5. Three types of sparse storing formats(COO, CSR and CSC)

Several interesting topics are all related to the storing formats of matrices. For example, given a certain matrix, a criteria should be developed about whether it should be stored in dense or sparse formats; also, differences in performance among those three sparse storing formats, COO, CSR and CSC, should be experimented on.

3. Experiments designed and hardware environments

3.1. Experiments designed

In this section, a list of experiments will be shown and those sets of experiments are designed based on control variable method. All the matrices to be involved in the experiments are generated randomly and composed of zeros and ones only. Variable 'sparse factor' can be defined as the possibility that a certain element of matrices will be presented as one. Therefore, sparse factor being closer to zero can be interpreted in the way that the matrix is sparser. Here are some variables that will become the focus of following experiments:

- block size in blocking technique
- sparse factor of generated matrices
- storing formats of two dimensional matrices
- replacement policies applied in the cache

Four sets of experiments, at least, can be designed according to the four variables listed above. In each set of experiments, only one factor will be changed in a certain range while three other factors will be kept unchanged so that the influence of that specific factor can be better studied. Four sets of experiments are demonstrated in tables 1.

Table 1. Parameter settings

Set 1	**VALUE OF VARIABLES**
Block size	4/8/16/32/64
Sparse factor	0.5
Storing formats	dense
Replacement policies	Least Recently Used

Set 2	**VALUE OF VARIABLES**
Block size	16
Sparse factor	0.1/0.2/0.3/0.4/0.5
Storing formats	dense
Replacement policies	Least Recently Used

Set 3	**VALUE OF VARIABLES**
Block size	16
Sparse factor	0.5(0.4/0.3/0.2/0.1)
Storing formats	Dense/COO/CSR/CSC
Replacement policies	Least Recently Used

Set 4	**VALUE OF VARIABLES**
Block size	16
Sparse factor	0.5
Storing formats	dense
Replacement policies	LRU/MRU/BIP

Besides those four variables, the size of matrices will be set to 128*128. Also, in order to minimize the influence of unexpected factors, each experiment will be executed for five times and the average number from all the results will be reported in the next section.

3.2. Hardware environments of the simulator

Considering SE mode of Gem5, where merely three components, CPU, cache and DRAM, are simulated, is used in the experiments, hardware environments of the simulator will be mainly about the attributes of CPU, cache and DRAM, and they are listed in table 2.

Table 2. Parameter settings

HARDWARE	**VALUE OF ATTRIBUTES**
CPU model	Timing Simple CPU
CPU frequency	4GHz
DRAM size	2GB
DRAM frequency	1600MHz
L1 dCache size	64KB
L1 iCache size	32KB
dCache line size	64B
dCache associativity	4

4. Experiments results and analysis

In this section, results of the four sets of experiments designed in the previous section will be displayed and the data and observed phenomenon will be analyzed carefully.

4.1. Set 1: Influence of block size

In the first set of experiments, the research goal is to figure out the influence that block size has on the data cache miss rate in blocking technique and other variables are kept fixed. The data collected from repeated experiments are displayed in the following figure.

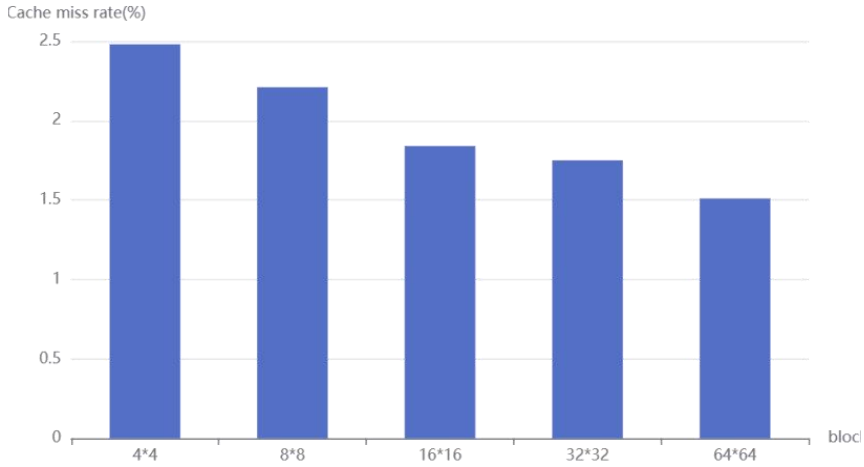

Figure 6. Influence of block size

It can be observed from Figure 6 that in blocking technique, as block size becomes smaller and smaller, data cache miss rate increases and it can be explained in the following manner. Blocking is designed to group several rows and columns together so that the locality can be enhanced, but it would resemble do calculations element by element if blocks are too small. Therefore, it should be encouraged to divide the whole matrix into fewer but bigger blocks rather than more but smaller blocks.

4.2. Set 2 and 3: Influence of sparse factor and storing formats

The results of these two sets of experiments will be reported together since they are closely connected. The sparse factor of generated matrices will be changed at first; for each sparse factor, data cache miss rate will be collected with different storing formats applied and they are dense format, COO, CSR, CSC respectively. The goal of these experiments is to compare the performance of various storing formats and determine whether or not the value of sparse factor has some influences on this comparison. Results of these two sets of experiments can be seen in Figure 7.

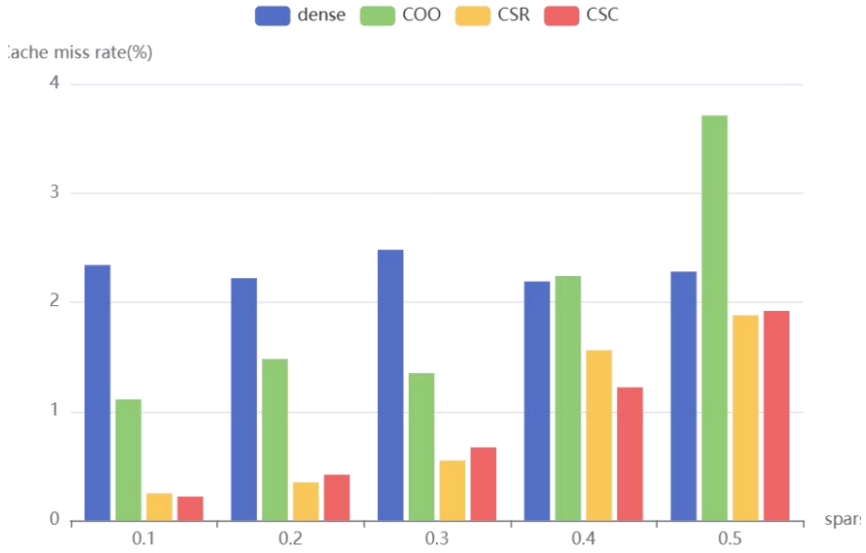

Figure 7. Influence of sparse factor and storing formats

Several phenomenon can be observed from this figure. First of all, sparse factor seems to have no influence on data cache miss rate since in dense two dimensional matrices, every element will be accessed regardless of its value. Therefore, the influence of sparse factor is expected to be trivial for cache miss rate which is identical to the data observed from experiments.

Secondly, if the sparse factor is low, which means there are plenty of zeros in the generated matrices, sparse storing formats is significantly better than dense storing format, probably because less elements need to be stored in cache and memory and the sparse storing formats are compact which is beneficial for high cache hit ratio. However, if the sparse factor is close to 0.5, sparse storing formats become worse than, or at least no better than, dense storing format when more non zero elements need to be stored plus sparse storing formats cannot take the advantage of blocking technique, which all contribute to the loss of better performance.

Moreover, the performance of CSR and CSC storing formats are generally better than COO while CSR and CSC are close to each other. The difference brought by COO format might be due to its specific implementation. CSR and CSC are symmetric and the matrix multiplication needs to scan the first matrix by rows and the second matrix by columns, which is also symmetric. Therefore, the difference between CSR and CSC is minor and insignificant.

4.3. Set 4: Influence of replacement policies

In this experiment, three types of replacement policies are introduced and they are LRU, MRU and BIP. Bimodal Insertion Policy is a combination of LRU and MRU with one configurable parameter p which means the possibility to apply LRU when choosing to evict one cache line. Therefore, the experiments will compare LRU, MRU and BIP with different parameter values. Data collected from this set of experiments is shown in Figure 8.

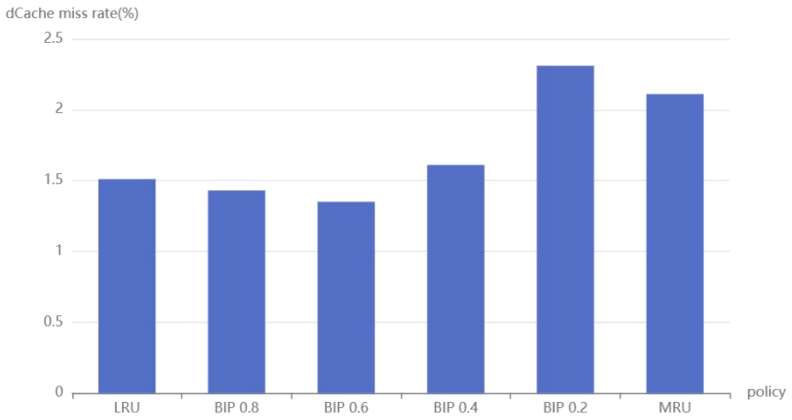

Figure 8. Influence of Replacement Policies

In this figure, cache miss rate is lowest when the configurable parameter of BIP is set to 0.6, which is the case where the possibility of choosing LRU is 60%. It is explainable because LRU is the mechanism to store the real hot data while MRU provides a mechanism of scan resistant to evict fake hot data out of cache immediately after it is accessed. Therefore, the combination of two extreme policies will obtain the optimal performance, which is a great example that sometimes the combination of several policies might make full use of each policy's advantages.

5. Conclusions

In this section, major conclusions will be drawn and stated based on the data shown in the previous section, after which several potential directions for future research will be listed. Several conclusions listed below can possibly provide researchers with some empirical evidences when optimizing and choosing the best setting for matrix multiplication. When using blocking technique, divide matrices into fewer but bigger blocks. When choosing storing format for sparse matrices, try CSR or CSC。 Sometimes the combination of two existing replacement policies will obtain better performance in terms of cache utilization. When considering some possible directions for future research in this field, following ideas might serve as a sort of inspiration. First, it is meaningful to investigate how cache behaviour will change when running matrix multiplication on parallel architecture. Secondly, a special storing format for matrix can be invented which is highly cache friendly. Also, an innovative replacement policy can be created for applications with intense matrix manipulation.

References

[1] Lam, Monica D., Edward E. Rothberg, and Michael E. Wolf. "The cache performance and optimizations of blocked algorithms." ACM SIGOPS Operating Systems Review 25.Special Issue (1991): 63-74.

[2] Ristov, Sasko, Marjan Gusev, and Goran Velkoski. "Optimal block size for matrix multiplication using blocking." 2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). IEEE, 2014.

[3] Binkert, Nathan, et al. "The gem5 simulator." ACM SIGARCH computer architecture news 39.2 (2011): 1-7.

[4] Ananth, M., S. Vishwas, and M. R. Anala. "Cache friendly strategies to optimize matrix multiplication." 2017 IEEE 7th International Advance Computing Conference (IACC). IEEE, 2017.

[5] Srivastava N, Jin H, Liu J, et al. Matraptor: A sparse-sparse matrix multiplication accelerator based on row-wise product[C]//2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020: 766-780.

[6] Pal S, Beaumont J, Park D H, et al. Outerspace: An outer product based sparse matrix multiplication accelerator[C]//2018 IEEE International Symposium on High Performance Computer Architecture (HPCA). IEEE, 2018: 724-736.
[7] Nocua A, Bruguier F, Sassatelli G, et al. ElasticSimMATE: A fast and accurate gem5 trace-driven simulator for multicore systems[C]//2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC). IEEE, 2017: 1-8.
[8] Elnawawy H, Alshboul M, Tuck J, et al. Efficient checkpointing of loop-based codes for non-volatile main memory[C]//2017 26th International Conference on Parallel Architectures and Compilation Techniques (PACT). IEEE, 2017: 318-329.
[9] Banerjee T, Gadou M, Ranka S. A genetic algorithm based approach for multi-objective hardware/software co-optimization[J]. Sustainable Computing: Informatics and Systems, 2016, 10: 36-47.
[10] Alshboul M, Elnawawy H, Elkhouly R, et al. Efficient checkpointing with recompute scheme for non-volatile main memory[J]. ACM Transactions on Architecture and Code Optimization (TACO), 2019, 16(2): 1-27.