Locality Analysis for Characterizing Applications Based on Sparse Matrices

Noboru Tanabe¹, Sonoko Tomimori², Masami Takata², and Kazuki Joe²
¹ Research and Development Center, Toshiba corporation, Kanagawa, Japan
² Department of Advanced Information and Computer Sciences, Nara Women's University, Nara, Japan

Abstract - We propose an adaptability judging method applied to sparse matrices and the target cache memory using two metrics based on spatial locality and temporal locality. For indirect access sequences of sparse matrix-vector multiplications, one metric is the number of valid data within a cache line, and another metric is average reference interval. We also develop a set of analysis tools to generate the above performance metrics, histograms of reference intervals and theoretical cache hit rates. As an experimental result, a cache memory behavior which was difficult to explain from the viewpoint of spatial locality becomes explicable from that of temporal locality. Outputs of the tool show that return on investment is too thin to increase the cache memory capacity unless all the elements of a column vector with the same size as the number of columns of the sparse matrix are stored in the cache memory.

Keywords: Sparse matrix; Cache memory; Spatial locality; Temporal locality; Workload characterization.

1 Introduction

Recently there are serious considerations [1] in Japan in order to put exascale machines to practical use in 2018. We make a prediction that the exascale machines will adopt a complicated memory system because of inevitable memory bandwidth problems. A very high demand for memory bandwidth includes applications of simultaneous linear equations (sparse matrix-vector multiplication) with large sparse matrix coefficient. Since such simultaneous linear equations are used in calculations for important scientific theme in Japan, there is a great need for them.

In the meantime, big data processing such as large-scale graph processing draws global attention. In general, large-scale graphs are represented as large and complicated sparse matrices with small number of non-zero elements. Recommendations/preferences of information retrieval or large-scale ad hoc information search [2] such as PageRank, which gives an importance of the website, needs that a sparse matrix processing expanding the scale and speeding up.

The K-computer that is the world's third fastest supercomputer in the Top 500 list of Nov 2012 has a relatively simple memory system based on two levels of cache memory. However, it is not easy to optimize in spite of its present simplicity. Optimization experts work only for a few selected applications and they do their best of optimization using the specific characteristics of the applications [3]. On the other hand, users of scientists can not take enough time and give any hands for performance tuning for the other applications without optimization experts and the applications with a short life cycle. Just looking at the applications formed with sparse matrices, there is a great variety of placements for non-zero elements as the University of Florida sparse matrix collection [4] shows. For such applications, parallelizing compilers and/or pre-optimized libraries are very important. In the exascale machines with complicated memory systems, the importance of automatic optimization mechanism is widely informed.

The above consideration leads us to start the development of a sparse matrix library that provides a universal adjustment function without the application-specific optimization techniques and auto-tuning new methods for sparse matrices by selecting the access mechanism. As a first step of the development, we propose a characteristic metric about spatial locality of sparse matrices. Furthermore, we propose a characteristic metric about temporal locality as well as spatial locality for their combinatory use. We develop a set of tools to calculate the metrics which measure the characteristics of various sparse matrices and investigate the correlation between the proposed metrics and L1 cache hit rates that have a strong correlation with the GPU processing speed.

The paper is organized as follows. In section 2 we marshal the locality of data accesses. We propose a combination of a characteristic metric based on spatial locality and another characteristic metric based on temporal locality, which compensates the spatial index, for sparse matrices in section 3. We explain the overview of analysis programs based on the proposed metrics and the application of the programs to optimize in section 4 and 5, respectively. In section 6, we describe some experiments using the proposed indices. Related works are given in section 7.

2 Locality of references

This section is organized in terms of locality of reference that cache memory is used to increase the speed of memory accesses.
2.1 Spatial locality

If a particular memory location is referenced at a particular time, then it is likely that the other memory locations around the particular memory will be referenced in the near future. Usually, in order to make effective use of the locality reference characteristic, a block (a cache line) of data addressed near the memory is able to judge cache hit or miss, move the cache line in the case of cache miss, or access the external memory by cache line. A lot of processors including the K-computer and GPUs are equipped with 128 bytes line size caches. The reason of the 128 bytes line size is that data transfer efficiency can be improved by making the burst length longer than a certain length. Furthermore many benchmarks evaluate the trade-off between the magnitude of the penalty per cache miss and the advantage of having the number of lines fitting into the limited cache memory capacity, which is described later as making use of temporal locality. It may be greater in the future because there is a tendency that the larger cache memory capacity is, the larger cache line size is.

However, the Graph500 benchmark and applications with strong random accesses of several sparse matrix-vector multiplications exhibit a lack of spatial locality. Even when just 4 bytes of a cache line is used, the whole 128 bytes including other 124 bytes in the cache line must be transferred from the external memory, and the transferred cache line causes a very inefficient performance as the result. Therefore, it is very important to control the spatial locality in sparse matrix accesses in order to speed up sparse matrix processing.

In particular, it is very difficult for software to control the spatial locality within a cache line. Solving this problem requires the memory system of vector supercomputers without cache memory, some hardware level supports such as the gather function of DIMMnet-2 combined with a cache-based system[5], or Hybrid memory Cube with gather function[6]. When sparse matrices with low temporal locality and low spatial locality are to be processed, the above hardware level supports are a promising solution.

2.2 Temporal locality

If a particular memory location is referenced, then it is likely that the same location will be referenced again in the near future. Usually, in order to make effective use of the locality reference characteristic, cache memory is composed of many lines so that it increases the probability of hitting cache by the accesses except the last access. Theoretical study of temporal locality has a long history. Denning et al. have proposed the concept [11] of working set (t, T) which is defined with the current time t and the window size T in 1968, and a new algorithm [12] to calculate the average working set size from the set of references within one pass.

A tiling method, which divides a matrix into smaller parts of matrices, may be a useful technique to improve the temporal locality of memory references in matrix processing such as dense matrices multiplication. However, the tiling method is not valid for most sparse matrix processing because sparse matrices have various non-zero elemental locations. In this case, by performing the replacement of the row and column numbers to change the order of memory references, the sparse matrix can improve its own temporal locality. That means temporal locality is controllable by software to some extent.

3 Matrix characteristic metrics based on reference locality

3.1 Matrix characteristic metrics based on spatial locality of references

We propose the "spatial locality of column-index sequences" as a new metric on the characteristics of sparse matrix that contributes to the classification of sparse matrix adaptability for cache memory. Fig. 1 shows the conceptual diagram.

![Fig. 1. The proposed metric 1 (spatial locality of column-index sequences)](image)

The following are the definitions used to represent the characteristic value of sparse matrix.

1. Store just the non-zero elements of a sparse matrix in CRS (Compressed Row Storage) format.
2. When read the index array from the top used to load column vectors, count the number of indexes they match except lower 5 bits which come from 32 data per line.
3. If upper bits of the new index do not match with them of the last index, it means cache-miss. Record the counter to count[line_ID], reset the counter to 1, increment the line_ID and continue reading the index array (i.e., go to step 2).
4. The spatial locality of column-index sequences is defined as the average of the count numbers recorded in 3.

Measuring the spatial locality of column-index sequences of sparse matrices with various formats and memory reference orders, we can find the impracticability of a sparse matrix to the target cache memory architecture which is not controllable.
by software. Namely, when the metric is not improved by just changing the orders and/or formats, it requires some hardware support to get better performance.

3.2 Matrix characteristic metrics based on temporal locality of references

In the previous subsection, we explain the spatial locality of references within a cache line and propose a metric for spatial locality of column-index sequences. In this subsection, we propose a combinatiorial use of the temporal locality of references within the same cache line in addition to the spatial locality. To measure the temporal locality by cache line, we use a line identifier line_ID[t], which is obtained by right-shifting the number of column-index sequences by the number of bits corresponding to the number of items (5 bits, which represent 128/4=32, in the case of 128 byte cache line and single-precision floating point) in the cache line, as input address sequences.

The temporal locality is concretely calculated as follows. We apply the Denning’s algorithm [12] that generates a histogram of reference intervals for cache lines, and we measure the resultant temporal locality for each column-index sequence of a sparse matrix to obtain the adaptability of the sparse matrix to the target cache memory. Figure 2 shows the conceptual diagram of reference intervals for cache lines used as the temporal locality of column-index sequences. The reference interval for a cache line is the time interval between time t which is the time when the cache line is accessed and time TIME[line_ID[t]] which means that the last time the cache line has been accessed. In the above algorithm, it records the current time t into TIME [line_ID [t]] before updating the time t.

As a metric about the temporal locality of references for cache lines, we use average reference interval or average working set size that is calculated by multiplying the average reference interval with the cache line size. There is no need to generate a histogram as necessarily in Denning’s algorithm for those calculations. So the histogram should not be generated by default since the calculation time for temporal locality must be short as the front-end part of our target sparse matrix library.

The performance metrics about spatial locality and temporal locality are computable within one pass since they use the same column-index sequences. In particular, it is better to calculate them within one pass when the column-index sequences are in a file rather than calculate them separately to shorten the whole processing time. In the above algorithm to calculate the reference intervals, the cache hit ratio can be obtained by dividing the access numbers where the reference interval does not exceed the number of the cache line by the total access numbers. Since the calculation of the cache hit rate requires an assumption of the FIFO cache line replacement algorithm, the calculation may have some error with different replacement algorithms or different way numbers of set associability.

4 Overview of the programs based on the proposed metrics

4.1 Input

One of the most parts of the calculation time of user applications is sparse matrix-vector multiplications. The access patterns of them are given by index array of sparse matrix. Our programs require sparse matrix files as their inputs to be given by users. These files represent the characteristics of applications for analyzing memory access properties. At this time, the programs accept the format of MatrixMarket. It is one of the most popular formats, and it is also valid for University of Florida Sparse Matrix Collection.

4.2 Output and Component

The outputs and components of the programs are listed below.

(1) Translator for CRS format
   It reads sparse matrix files with the MatrixMarket format to save the data structure with the CRS format in memory.

(2) Translator for GPU related formats
   It converts sparse matrices with the CRS format generated in (1) into the data structure with applying a pre-processing for GPUs, the Fold method [7], which consists of 0-padding, folding and transposition. The pre-processing tends to generate the data structures with completely different access patterns from the CRS format. As a result, a sparse matrix with the CRS format having low access performance is converted to a data structure having high access performance using the pre-processing, and vice versa.

(3) Measurement program for spatial locality
   Giving the output data structures of (1) or (2), it calculates the performance metric for spatial locality proposed in the previous section. In either case, sparse matrices with low (close to 1) calculated values for spatial locality are considered as having low adaptability to the cache memory because the practically available memory bandwidth decreases so much in the case that
the L2 cache memory cannot keep the whole column-index vectors.

(4) Measurement program for temporal locality
Giving L1 and L2 cache size of the target cache memory systems as well as the output data structures of (1) or (2), it calculates the performance metric (average reference interval or average working-set size) for temporal locality and theoretical L1 cache hit rate described in the previous section. A histogram of reference intervals is optionally computable. Analyzing the histogram, it can be estimated how large cache memory capacity is required to get reasonable speed-up of the sparse matrix processing.

5 Qualitative optimization strategy
The average working set size, which is expressed in byte, indicates how much cache capacity it needs in order to be stable for the cache hit rate. Comparing the average working set size with the cache memory capacity of the target platform, it is expected that we know how the cache memory works from the view point of temporal locality and whether the cache memory should be used aggressively or not with automatic optimization. Table 1 qualitatively summarizes the optimization strategy with the performance metrics of spatial locality and temporal locality.

Table 1. Optimization strategy based on dual properties of access locality

<table>
<thead>
<tr>
<th>Low temporal locality</th>
<th>High temporal locality</th>
</tr>
</thead>
<tbody>
<tr>
<td>High spatial locality</td>
<td>Explore the other ordering. Use cache.</td>
</tr>
<tr>
<td>Low spatial locality</td>
<td>Use hardwired gather. Within L1 : Use cache. Overflow from L1 : Use hardwired gather.</td>
</tr>
</tbody>
</table>

From the view point of high spatial locality, a high cache hit rate is expected while extremely low temporal locality may cancel out the performance effect. In this case, since the spatial locality is already high, a memory system with gather functions may not work well and the cache memory benefit is very limited. However, it is possible for the temporal locality to be improved by changing memory reference orders, so the automatic optimization should be performed with ordering changes. Since changing ordering tends to induce the changes of temporal and spatial locality, it is possible to be included in a different category in Table 1.

On the other hand, from the view point of high spatial locality (a high cache hit rate), when the average working-set size is sufficiently-small compared with the cache memory capacity, the temporal locality is extremely high and it cancels out the effect. In this case, the use of cache memory does not really degrade the execution efficiency. Therefore, the automatic optimization should be performed using the cache memory.

In the case of low spatial locality and not extremely high temporal locality, the effect of a memory system with gather functions is very promising, and the automatic optimization should adopt it.

6 Evaluation
6.1 Environments and matrices for experiments
Table 2 and 3 shows the computing environment and matrices used in the experiment, respectively. These matrices are selected from the University of Florida sparse matrix collection, which is a collection of sparse matrices found in real-world applications. These sparse matrices are often used by researchers in numerical linear algebra for the development and performance evaluation of sparse matrix algorithms. In this experiment we chose the sparse matrices to focus on the non-zero elements that look like scattered in irregular shapes (the optimization is difficult for cache memory) on the matrix diagram. They are derived from the sparse matrices of structural analysis, electronic circuit analysis, web analysis, and road network analysis.

Table 2. Experimental environment

<table>
<thead>
<tr>
<th>CPU</th>
<th>Intel®Xeon® X5670 @ 2.93GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPU</td>
<td>Nvidia Tesla C2050 (# of core : 448)</td>
</tr>
<tr>
<td>Device memory</td>
<td>144GB/s, 3GB</td>
</tr>
<tr>
<td>Host I/F</td>
<td>PCI express x16 Gen.2 (8GB/s)</td>
</tr>
<tr>
<td>OS</td>
<td>RedHat Enterprise Linux Client release5.5</td>
</tr>
<tr>
<td>CUDA</td>
<td>Cuda3.2</td>
</tr>
</tbody>
</table>

Table 3. Experimented matrices

<table>
<thead>
<tr>
<th>Name</th>
<th># of non 0 elements</th>
<th># of rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>crankseg_2</td>
<td>7,106,348</td>
<td>63,838</td>
</tr>
<tr>
<td>nd24k</td>
<td>14,393,817</td>
<td>72,000</td>
</tr>
<tr>
<td>thermal2</td>
<td>3,489,300</td>
<td>147,900</td>
</tr>
<tr>
<td>hood</td>
<td>5,494,489</td>
<td>220,542</td>
</tr>
<tr>
<td>F1</td>
<td>15,590,452</td>
<td>343,791</td>
</tr>
<tr>
<td>msdoor</td>
<td>10,328,399</td>
<td>415,863</td>
</tr>
<tr>
<td>rajat29</td>
<td>4,866,270</td>
<td>643,994</td>
</tr>
<tr>
<td>ASIC_680ks</td>
<td>12,329,176</td>
<td>682,712</td>
</tr>
<tr>
<td>apache2</td>
<td>2,766,523</td>
<td>715,176</td>
</tr>
<tr>
<td>ldoor</td>
<td>23,737,339</td>
<td>952,203</td>
</tr>
<tr>
<td>webbase-1M</td>
<td>3,105,536</td>
<td>1,000,005</td>
</tr>
<tr>
<td>delaunay_n20</td>
<td>2,097,124</td>
<td>1,048,576</td>
</tr>
<tr>
<td>roadNET-TX</td>
<td>1,281,106</td>
<td>1,393,383</td>
</tr>
<tr>
<td>Hamrle3</td>
<td>5,514,242</td>
<td>1,447,360</td>
</tr>
<tr>
<td>G3_circuit</td>
<td>4,623,152</td>
<td>1,585,478</td>
</tr>
<tr>
<td>roadNET-CA</td>
<td>1,844,404</td>
<td>1,971,281</td>
</tr>
</tbody>
</table>

* Intel, Xeon are trademarks of Intel Corporation in the U.S. and/or other countries.
6.2 Experiments

We measure the proposed performance metrics of sparse matrices with the CRS format and pre-processed sparse matrices as well as the L1 cache hit rates for the pre-processed sparse matrices. We use the fold method [7] as the pre-process, which converts the access order of index arrays of CRS format sparse matrices into the transposed order for GPUs. The change of the access order affects the cache hit rate. Namely, the converted access order has better compatibility to cache memory rather than the CRS format, the effect of the pre-processing is easily observed.

We examine the correlation between the L1 cache hit rate of a GPU and the proposed metrics for spatial locality. The results are shown in Fig.3. Figure 3 clearly shows that there is a majority group showing positive correlation between the L1 cache hit rate of the GPU and the proposed metrics (the sparse matrices group surrounded by a green ellipse), a minor group showing no correlation (surrounded by a blue ellipse), and an isolated sparse matrix (surrounded by a red circle). It means that only the metric for spatial locality is insufficient for judging the adaptability of a given sparse matrix to cache memory in advance.

Table 4 shows the average inter-reference interval and the average working set size of each sparse matrix. Thermal2 and roadNET-TX obviously have small average working set sizes, and they are less than or equal to the GPU L1 cache size (16KB in the case of Fermi). We think this is the reason why our experiments described in the previous subsection show the relatively high cache hit rates in spite of low spatial locality. This is the notable effect that becomes obvious with the combinational use of temporal locality.

On the other hand, apache2 has a larger average working set size than the other four applications in the blue group from the viewpoint of temporal locality. From the fact that it is significantly larger than the L1 cache memory size, we can explain that the L1 hit rate is observed low in apache2 in spite of the high spatial locality. This is also the notable effect that becomes obvious with the combinatorial use of temporal locality.

Figure 4 shows the relationship between the average reference intervals measured in the experiments and the L1 cache hit rate of a GPU (C2050). The correlation coefficient is -0.682, and any abnormal samples deviating significantly as shown in Fig.3 in the spatial locality are not found although the variance is slightly loosened. A linear approximation formula is expressed as $hit = -0.018d + 41.267$.

![Fig. 3. The relation between L1 cache hit rate of GPU(C2050) and the number of valid data/line (Pre-processed by Fold-method)](image1)

![Fig. 4. The relation between average inter reference distance and L1 cache hit rate of GPU(C2050)](image2)
Figure 5 shows the histograms of reference intervals for each sparse matrix. The right end of each histogram is the sum of the number of reference intervals that are larger than $L$ and the number of the initial cache misses. In this measurement we measure the average interval to assume the window size $L$ corresponding to the L2 cache memory size of 8MB, which is one of the largest ones at this point. The histograms of reference intervals of sparse matrices in Fig.5 show the properties about the adaptability of the sparse matrices to the target cache memory. For example, in the case of apache2 of which cache hit rate is low, there are no sample points but both ends and the cache misses in the right end are not improved by the cache memory with insufficient capacity. Since the spatial locality in this state is extremely high, the effect of the memory system with gather functions is not expected at all. There are two possible optimization for speed-up. One is to change the memory reference orders to improve the cache hit rate and another is to change folded point of pre-processing in order to reduce padded zero which improves inflated spatial locality and degrades the effect of the memory with gather functions.

Although other sparse matrices in Fig.5 show that the histograms are divided in both ends, samples less than several hundreds of the left end are presented in the middle. They can be improved by enlaring the cache memory capacity to several MB that is significantly larger than the average working set size. However, the return on investment is very thin because the number of samples in the middle parts is very small compared with both ends. Therefore we conclude that increment of the cache memory capacity is not cost-effective unless the whole column vector is stored in the cache memory.

7 Related work

On the K-computer, optimization with specific character of two applications of which main calculation is for sparse matrix is performed by human hands. For example, the applications are optimized provided that the maximum number of nonzero elements in a row of the sparse matrix is set to 27 [3]. As general purpose oriented approaches, automatic tuning, which selects suitable software automatically, has been studied. For example, selecting storage schemes of matrices is reported for GPUs [8]. However, as far as we know, there are no reports to measure both spatial and temporal locality of column-index sequences of sparse matrices in advance as the specific characteristics of the sparse matrices or their pre-processing to be used for automatic tuning. Locality in sparse matrix-vector multiplications, which is a similar research to be focused on sparse matrix and memory reference locality, has been investigated by Heras [9]. In this research, three distance functions are proposed as metrics. Since the selection of suitable window size or indices is given by experimental results, it cannot be used for automatic tuning. Our research and Perarnau’s research [10] are focused on the locality of sparse matrix. The target of Perarnau’s research is the AMG method, which makes use of trace data from real machines. On the other hand, the target of our research is sparse matrix-vector multiplication. We use the column-index sequences, which is placement pattern of nonzero elements of the sparse matrix. As for temporal locality, Denning’s working set is widely known [11][12], and we measure performance metrics based on inter-reference interval, which is derived from Denning’s working set. There are many existing researches [14-16] that uses the Reuse distance (Stack distance) proposed by Matson [13]. They are using only the metric for temporal locality. On the other side, we calculate the spatial locality, we can know the efficiency of memory bus and cache and can judge which sparse matrix should be located on the memory with gather functions.
8 Conclusions

We guess that the exascale machines adopt a complicated memory system. Toward the implementation of a sparse matrix library to be possibly used for the exascale machines, in this paper we proposed an adaptability judging method applied to sparse matrices and the target cache memory using metrics of spatial locality within a cache line and temporal locality among cache lines. For indirect access sequences of sparse matrix-vector multiplications, the former metric is the number of valid data within a cache line, and the latter metric is average reference interval and average working set size. We also developed a set of analysis programs to generate the above metrics, histograms of reference intervals and theoretical cache hit rates. We evaluated the proposed metrics based on the analysis programs and the University of Florida sparse matrix collection. As a result, a case that a cache memory behavior was difficult to explain from the view point of spatial locality becomes explicable from the view point of temporal locality. Thus, we could strengthen the basis for deriving more appropriate optimization strategies just by the placement patterns of non-zero elements of sparse matrices.

Now we can estimate how much the cache memory capacity should be increased to reasonably optimize the sparse matrix processing using the reference interval histogram generated by the program. The access characteristics to sparse matrices in the experiments are classified into two groups; small reference intervals where most accesses hit the L1 cache memory and long reference intervals where considerable numbers of accesses do not hit any cache memory. It means that return on investment is too thin to increase the cache memory capacity unless all the elements in a dense column vector are stored in the cache memory.

As described above, just generating a sparse matrix for the target application, the proposed programs make scientists with a poor knowledge of computer architecture judge in advance whether it works efficiently on a cache memory based processor or a special library for sparse matrices judge it at runtime. In addition, we can obtain useful information for future computer designers to investigate appropriate and possible cache memory capacity or consider the use of the memory system with gather functions.

Our future work includes comprehensive survey of sparse matrices and sparse matrix library with auto-tuning mechanisms using the proposed metrics.

Acknowledgements

A part of this work is supported by the Ministry of Internal Affairs and Communications (Soumu-sho).

References


