A Flow-Guided File Layout for Out-Of-Core Streamline Computation

Chun-Ming Chen Lijie Xu Teng-Yok Lee Han-Wei Shen

The Ohio State University, OH 43210 – {chenchu, xul, leeten, hwshen}@cse.ohio-state.edu

ABSTRACT

We present a file reordering method to improve runtime I/O efficiency for out-of-core streamline computation. Because of the increasing discrepancy between the speed of computation and that of I/O on multi-core machines, the cost of I/O becomes a major bottleneck for out-of-core computation. Among techniques that reduce runtime I/O cost, reordering file layout to increase data locality has become popular in recent years. Better layout optimization relies on the knowledge of the data access pattern, which can be acquired from benchmarking. For streamline computation, we observe that the data access pattern is highly dependent on the flow directions. As a disk I/O request can generally be done more efficiently with shorter seek distances, we propose a novel flow-guided file layout method to improve the I/O performance. With a weighted directed graph to model the data access pattern, the file layout problem can be formulated as a linear graph arrangement problem. The goal is to minimized the sum of the disk seek time based on the linear distances between all pairs of adjacent graph nodes. We use a divide-and-conquer algorithm to approximate the optimal layout. The experimental results show that our flow-guided layout outperforms layouts that use space filling curves and some of the more recent cache-oblivious mesh layout methods.

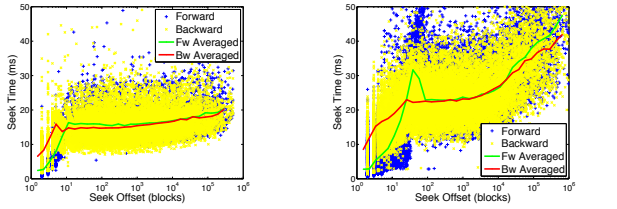
1 INTRODUCTION

As the computation power and storage space in supercomputers continue to increase, the size of data generated from simulations also grows proportionally. Consequently, a key requirement for modern visualization software is to be able to handle very large scale data sets. Although there is a trend to shift as much computation in visualization algorithms to supercomputers as possible, scientists still prefer to look at the data using their desktop computers. Machines of this type, however, usually have a much smaller amount of memory, which means visualization of large data sets need to be done in an out-of-core manner.

The focus of this work is to optimize the layout of three dimensional vector data in disk for out-of-core streamline computation. Although multi-core desktop machines have become pervasive, which allows the computation of streamlines to be done in parallel, the bottleneck is in moving data from disk to memory. Previously, Cox *et al.*[1] show that storing three-dimensional data in blocks increases spatial locality and hence improves I/O performance. However, it is not clear how the data blocks should be stored in a file so as to minimize the I/O cost. It is known that disk I/O can be done more efficiently when reading contiguous data, since smaller disk seek time is involved for each I/O request. In this abstract, we describe a technique that organizes data blocks in the file by following the flow directions so that contiguous data blocks in a file are read when advecting particles.

Re-arranging layouts for better data locality has received much research attention in recent years. Niedermeier *et al.* use Hilbert curve (H-Curve), which has high geometric locality properties, to create efficient mesh indexing [2]. Yoon *et al.* formulate unstructured mesh-reordering as a linear graph arrangement problem [4]. A novel cost function that estimates cache misses is proposed and implemented in an algorithm called OpenCCL.

When tracing particles, two data blocks in a vector field will be accessed continuously if they are neighboring to each other and there exists at least one particle that flows from one block to the other. One important factor to consider is that not all pairs of spatially neighboring data blocks have an equal probability to be accessed together. This is because from a block, randomly seeded particles can go to one of its neighbors more likely than to other neighbors. This is the main difference between traversing data blocks for particle tracing than traversing cells in a mesh. In streamline computation, there may be different access dependencies between neighboring data blocks. However, the access pattern can be estimated after knowing the vectors in the field. With this unique property, the estimated runtime access pattern can be used to create a more I/O friendly layout for the data blocks. We use a weighted directed graph to estimate the probabilities of particles flowing from one block to another. In addition, we construct a cost function from benchmarking to model the disk seek time if two adjacent graph nodes are accessed continuously. With the graph and the cost function for disk seek time between adjacent graph nodes, we can estimate the cost related to non-contiguous disk access during streamline computation. From the cost model, we apply a divide-and-conquer method to determine the data layout that can minimize the data access cost. We show that our layout with the knowledge of flow directions can achieve better runtime I/O efficiency compared to the H-Curve layout and the OpenCCL layout.



(a) Machine-1: 400 GB single disk (b) Machine-2: 1.2 TB RAID-5 disk array

Figure 1: Benchmarks of disk seek time from application point of view. Each sample point represents the response time to the seek distance, by randomly reading a data block in a large file. The block size is around 70 KB, representing an 18 cube of data of floating-point vectors. The time is averaged to construct a mean curve.

2 THE FILE LAYOUT METHOD

2.1 The Disk Seek Time Modeling

Although performance modeling for physical disk seek time has been studied before [3], the actual speed from the application point of view is not always easy to predict because it can be easily affected by operating system related features such as file caching. To address this issue, we create an empirical model by benchmarking the cost for disk seek.

In our benchmark, we randomly accessed a large file. For different file offsets, we measured the response time corresponding to the disk seek distance, and created a log-scale plot where the x axis is the file seek distance and the y axis is the time. The plot divided the x axis in log-scale regions. For the seek distances in each of such region we defined representative seek time by averaging the measured time in the region. Finally by linearly interpolating the av-

IEEE Symposium on Large Data Analysis and Visualization
October 23 - 24, Providence, Rhode Island, USA
978-1-4673-0155-8/11/$26.00 ©2011 IEEE

Table 1: The graph statistics and the layout generation time.

	Nek	Plume	Isabel
File Size	2.28 GB	8.88 GB	11.51 GB
Dimension	512 cube	512x512x2048	1500x1500x300
# Graph Nodes	32,768	131,072	167,884
# Graph Edges	152,806	450,097	485,840
Avg. Graph Node Out-degree	4.7	3.4	2.9
ADG Graph Generation Time	8.03 min	91.3 min	109 min
Layout Generation Time	3.49 secs	37.2 secs	58.4 secs
File Reorder Time	58.4 secs	5.13 min	7.50 min
Total Layout Gen. Time	9.06 min	97.0 min	118 min

eraged time we constructed a mean curve representing the average costs of disk seeks. Figure 1 shows the plots for our experiments on two Linux machines, one is using a single disk and the other a RAID-5 disk array. The mean curve is used as the metric in the following data layout cost function.

2.2 The Data Access Modeling And Layout Algorithm

To model the cost of data access for a block layout, we use a graph-based data dependency model, the *Access Dependency Graph* (ADG), for out-of-core streamline computation. In the graph, each node represents a spatial data block in the flow field, and each directed edge between two nodes represents the flow between the two blocks. The edge weight represents the probability for a particle randomly seeded in the block to travel to the other block. To model the weight, we seed particles evenly on the six surfaces of each block, as illustrated in Figure 2(a). Then the percentage of the particles moving from one block to each of its neighbors are collected as the edge weight, as shown in Figure 2(b). Since the computation is done within each block, it can be easily performed out-of-core and in parallel. To feature the flow direction in the ADG, the block size should be small so that vectors in the same block are in the similar direction. We chose 16 cube as the block size.

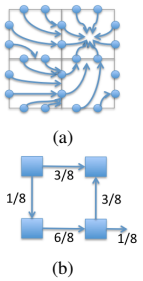


Figure 2: The Construction of ADG

With the ADG and the cost metric from section 2.1, we can estimate the cost of disk seek with respect to a particular ordering of data blocks in a file. We define the cost of each edge as the edge weight multiplied by the cost evaluated from our benchmark given the distance of two blocks in the file. The overall cost is the sum of all the edge costs, assuming each block has an equal chance to be visited. The goal is to find a layout which minimizes the cost.

We use the following divide-and-conquer algorithm to obtain an approximated solution for the layout that will give us a minimum cost: The graph is recursively divided into two sub-graphs by finding a 1/3-cut with the minimum cut-cost. The 1/3-cut is to partition the input graph into two disjoint sub-graphs, within each of which the number of nodes should be at least 1/3 of that of the original graph. When the subgraph is small enough (4 nodes in our implementation), a local permutation search is applied to find the lowest cost.

3 Results and Discussions

We reordered the data files based on the above layout algorithm and compared the runtime I/O performance among four different layouts on different machines, including ours. In the results presented in this section, BENCHMARK is generated from our layout algorithm using the ADG and the disk benchmark curves; HCURVE is the H-Curve layout; CCL is the layout using OpenCCL [4]. Finally, SERIAL is a common ordering where the blocks are in slice- or column-major order. Between our algorithm and OpenCCL, We use the same ADG as the input but ignored the weights and directions in OpenCCL since they are not considered in OpenCCL.

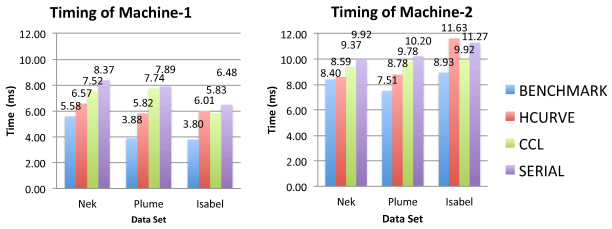


Figure 3: A comparison of average block loading time with different layouts by single-seed particle tracing. BENCHMARK is our layout, HCURVE is using Hilbert-Curve, and CCL is using OpenCCL layout.

The datasets tested were *Nek* (from a fluid dynamics solver NEK5000 from Argonne National Laboratory), *Plume* (from a simulation of the thermal down-flow plumes on the surface layer of the sun) and *Isabel* (from a timestep of a hurricane simulation). The later two datasets are from the National Center for Atmospheric Research. Table 1 shows the data preprocessing time, including graph generation, layout generation and file reordering for different datasets. All the computation ran on a Linux machine with a hyperthreaded 8-core Intel machine, 24 GB of RAM.

To compare the layout performance with the same data request order, we traced one particle each time and summed up the loading time of the data blocks visited by the particle. For each dataset and layout we randomly seeded and averaged the total time from more than 300 tests. All the tests were on the same benchmark machines stated in section 2.1.

Discussion From Figure 3, it can be seen that our layout performs better than H-Curve and CCL layouts in the case of single-seed particle tracing. The advantage of our layout over H-Curve and OpenCCL layout is that ours leverage the knowledge of runtime access pattern, while CCL layout treats all pairs of neighboring blocks equally.

In summary, compared to the Hilbert-Curve layout, our BENCHMARK layout has performance gains of 15%, 33% and 36% with Nek, Plume and Isabel respectively on a single-disk machine, and 2%, 14% and 23% on a RAID-5 machine. The different performance gains may be related to the average node degree in the ADG. As illustrated in table 1, the Nek dataset has the highest average out-degree (4.7), so that it is harder to find a dominant flow direction for the data blocks. In this case the performance is similar to the more general purpose layouts like H-Curve.

Conclusion We have presented a novel algorithm to determine data layout for out-of-core streamline computation. With the knowledge of flow directions and the cost metric for disk seek time from benchmarks, our layout consumes less I/O time than Hilbert-Curve and the more recently published cache-oblivious mesh layouts. The future work is to integrate the file layout with the scheduling of multi-threaded out-of-core streamline computation.

References

[1] M. Cox and D. Ellsworth. Application-Controlled Demand Paging for Out-of-Core Visualization. In *Proceedings of the 8th conference on Visualization '97*, page 235. IEEE Computer Society Press, 1997.

[2] R. Niedermeier and P. Sanders. On the Manhattan-Distance Between Points on Space-Filling Mesh-Indexings. Technical report, 1996.

[3] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *Computer*, 27(3):17–28, 1994.

[4] S.-E. Yoon, P. Lindstrom, V. Pascucci, and D. Manocha. Cache-oblivious mesh layouts. *ACM Trans. Graph.*, 24(3):886, 2005.