Cache-Coherent Layouts

Sung-Eui Yoon
(윤성의)

Course URL:
http://jupiter.kaist.ac.kr/~sungeui/SGA/
Course Administration

- Make progresses on your chosen topic
  - Read papers now, not later!
  - Think about pros and cons of each paper
  - Think about how we can further improve
  - Write down toward the mid-term report, whose deadline is Nov-6
Organization of Report

- **Introduction**
  - State a topic and a problem that you want to address
  - Give motivations
  - Present your main idea (and results)

- **Related work**
  - Identify a few major categories related to the topic
  - Emphasize benefits over your method
Organization of Report

- Overview
  - State the problem in detail
  - Present your idea
  - Why your idea address the problem

Your mid-term report should have introduction, related work, and overview sections
Organization of Report

- **Main body of the papers**
  - Describe your idea/solutions in detail

- **Implementation & results**
  - Describe your implementation and results
  - If you didn’t implement, please provide a rough implementation sketch and the expected results

- **Conclusion**
  - Summary your topic, problem, and your idea
  - Emphasize results/benefits of your idea
  - Lay out future work
At the Previous Classes

- Studied visibility culling and LOD techniques for rasterization and ray tracing
  - Reduced the model complexity
In this Class

- Will study cache-coherent layouts of meshes, graphs, hierarchies
  - Re-organize the data for efficient geometric processing and rendering
Motivation

- Lower growth rate of data access speed

Accumulated growth rate during 1993 – 2004 (log scale)

![Bar chart showing growth rates](chart.png)

1.5X, 20X, 46X, 130X

Disk access speed, RAM access speed, CPU speed, GPU speed during 99 - 04

Memory Hierarchies and Block-Based Caches

Fast memory or cache

CPU

Disk

Access time: $10^{-8}$ sec $10^{-7}$ sec $10^{-2}$ sec

Block transfer
Cache-Coherent Layouts

- Stores related data closely in the 1D layout

- Cache-Aware
  - Optimized for particular cache parameters (e.g., block size)

- Cache-Oblivious
  - Minimizes data access time without any knowledge of cache parameters
  - Directly applicable to various hardware and memory hierarchies
CAD Model – Double Eagle Tanker Model

Irregular distribution of geometry
Isosurface and Scanned Models

Isosurface
100M triangles

St. Matthew
372M triangles
Main Approaches

- Propose novel and practical cache-coherent metrics [Yoon et al. SIG 05, Yoon et al. VIS 06, Yoon et al. Euro 06]
  - Derive metrics given block-based caches
  - Propose efficient cache-coherent layout constructions
  - Apply to different applications
Cache-Coherent Metrics

- Measure the expected number of cache misses of a layout given block-based caches
  - Should correlate well with the observed number of cache misses

- Cache-aware metrics
  - Measure cache-coherence given known cache parameters (e.g., block size)

- Cache-oblivious metrics
  - Consider all possible cache parameters
Run-time Captured Video – View-Dependent Rendering of St. Matthew Model

St. Matthew

372 Million triangles
9 Gigabyte

GPU: GeForce 6800
Related Work

- Computation reordering
- Data layout optimization
Computational Reordering

- **Cache-aware** [Coleman and McKinley 95, Vitter 01, Sen et al. 02]
- **Cache-oblivious** [Frigo et al. 99, Arge et al. 04]
- **Streaming computations** [Isenburg et al. 05, 06]

Focus on specific problems such as sorting and linear algebra computations
Data Layout Optimization

- Rendering sequences (e.g., triangle strips)
  - [Deering 95, Hoppe 99, Bogomjakov and Gotsman 02]
- Processing sequences
  - [Isenburg and Gumhold 03, Isenburg and Lindstrom 05]

Assume that access pattern globally follows the layout order!
Data Layout Optimization

- Graph and matrix layout
  - A survey [Diaz et al. 02]
  - Minimum linear arrangement (MLA)
  - Bandwidth, etc.

Does not necessarily produce good layouts for block-based caches
Data Layout Optimization

- Space-filling curves
  - [Sagan 94, Pascucci and Frank 01, Lindstrom and Pascucci 01, Gopi and Eppstein 04]

Assume geometric regularity!
Outline

● Computation models
● Cache-aware and cache-oblivious metrics
● Results
Outline

- Computation models
- Cache-aware and cache-oblivious metrics
- Results
General Framework of Layout Computation

Input directed graph, G (N, A)

Layout algorithm, $\phi$

Cache-coherent metric

1D layout, $\phi(N)$
Two-Level I/O Model [Aggarwal and Vitter 88]

M cache blocks, whose size is B

Cache

Input directed graph

Layout algorithm, φ

1D layout with block size = 3
Graph Representation

- Directed graph, $G = (N, A)$
  - Represent access patterns between nodes

- Nodes, $N$
  - Data element
  - (e.g., mesh vertex or mesh triangle)

- Directed arcs, $A$
  - Connects two nodes if they are accessed sequentially
Weights of Nodes and Arcs

- Indicate probabilities that each element will be accessed

- Computed in an equilibrium status during infinite random walks
  - Assume that applications infinitely access the data according to the input graph
  - Correspond to eigen-values of the probability transition matrix
Problem Statement

- Vertex layout of $G = (N, A)$
  - One-to-one mapping of vertices to indices in the 1D layout
  \[ \varphi : N \rightarrow \{1, \ldots, |N|\} \]

- Compute a $\varphi$ that minimizes the expected number of cache misses
Cache-Coherence of a Layout given Block-Based Caches

- Expected number of cache misses of a layout
  - Probability accessing a node from another node by traversing an arc
  - Conditional probability that we will have a cache miss given the above access pattern
Specialization to Meshes

- Expected number of cache misses of a layout
  - Probability accessing a node from another node by traversing an arc $= \text{constant}$
  - Conditional probability that we will have a cache miss given the above access pattern

An input mesh

Implicitly derived graph

1. Two opposite directed arcs
2. Uniform distribution to access adjacent nodes given a node
Outline

- Computation models
- **Cache-aware and cache-oblivious metrics**
- Results
Four Different Cases

- **Cache-aware case**
  - single cache block, \( M=1 \)
  - multiple cache blocks, \( M>1 \)

- **Cache-oblivious case**
  - single cache block, \( M=1 \)
  - multiple cache blocks, \( M>1 \)
Cache-Aware: Single Cache Block, M=1

Input directed graph

Straddling arcs

1D layout with block size = 3

Cache, whose block size is B
Cache-Aware: Multiple Cache Blocks, M>1

Input directed graph

Straddling arcs

1D layout with block boundary
Final Cache-Aware Metric

- Counts the number of straddling arcs of the layout given a block size \( B \)

\[
\frac{1}{|A|} \sum_{(i,j) \in A} S(|\phi^B(i) - \phi^B(j)|)
\]

where

- \( \phi^B(i) \): block index containing the node, \( i \)
- \( S(x) \): Unit step function, 1 if \( x > 0 \)
- 0 otherwise.
High Accuracy of Cache-Aware Metric

Tested block size = 4KB

<table>
<thead>
<tr>
<th>Linear correlation [-1, 1]</th>
<th>Observed number of cache misses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With 5 cache blocks</td>
</tr>
<tr>
<td>Cache-aware metric</td>
<td>0.97</td>
</tr>
</tbody>
</table>

Z-curve on a uniform grid

Tested layouts:
- Z-curve, Hilbert curve, H-order,
- minimum linear arrangement layout,
- $\beta\Omega$-layout, geometric CO layout,
- (bi or uni) row-by-row,
- (bi or uni) diagonal layouts
Cache-Aware Layouts

- Optimized with cache-aware metric given a block size $B$
- Computed from the graph partitioning

Input directed graph

Straddling arcs

1D layout with block size = 3
Four Different Cases

- **Cache-aware case**
  - single cache block, \( M=1 \)
  - multiple cache blocks, \( M>1 \)

- **Cache-oblivious case**
  - single cache block, \( M=1 \)
  - multiple cache blocks, \( M>1 \)
Cache-Oblivious: Single Cache Block, M=1

Does not assume a particular block size:
Then, what are good representatives for block sizes?
Two Possible Block Size Progressions

- Arithmetic progression
  - 1, 2, 3, 4, ...

- Geometric progression
  - $2^0$, $2^1$, $2^2$, $2^3$, ...
  - Well reflects current caching architectures
  - E.g., L1: 32B, L2: 64B, Page: 4KB, etc.
Probability that an Arc is a Straddling Arc

Is an arc straddling given a block size?  
Computed as a probability as a function of arc length, $l$

Arc length, $l$, = 2

$\begin{array}{cccc}
n_a & n_d & n_c & n_b \\
\end{array}$

........
Two Cache-Oblivious Metrics

- **Arithmetic cache-oblivious metric**, $COM_a(\varphi)$
  
  MLA metric, Arithmetic mean
  
  $$\frac{1}{|A|} \sum_{(i,j) \in A} l_{ij}$$
  
  Arc length of arc $(i, j)$

- **Geometric cache-oblivious metric**, $COM_g(\varphi)$
  
  $$\frac{1}{|A|} \sum_{(i,j) \in A} \log(l_{ij}) = \log\left(\prod_{(i,j) \in A} l_{ij}\right)$$
  
  Geometric mean of arc lengths
Validation for Cache-Oblivious (CO) Metrics

- Geometric cache-oblivious metric
  - Practical and useful

73% of tested power-of-two block sizes

97% of tested block sizes

The number of cache misses when \( M = 1 \) (log scale)

Geometric CO layout

Arithmetic CO layout
Correlations between Metrics and Observed Number of Cache Misses

Tested block size = 4KB

<table>
<thead>
<tr>
<th>Linear correlation [-1, 1]</th>
<th>Observed number of cache misses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With 1 cache block</td>
</tr>
<tr>
<td>Geometric CO metric</td>
<td>0.98</td>
</tr>
<tr>
<td>Arithmetic CO metric</td>
<td>-0.19</td>
</tr>
</tbody>
</table>

Tested with 10 different layouts on a uniform grid
Cache-Oblivious Layouts

- Geometric cache-oblivious metric
  - Very efficient
  - Can be used in different layout optimization methods
Layout Computation with Geometric Cache-Oblivious Metric

- Multi-level construction method
  - Partition an input mesh into k different sets
  - Layout partitions based on our metric

- Generalized layout method for unstructured meshes

1. Partition
2. Lay out
Evaluating Existing Layouts

An existing layout, $\varphi$

Is it close to the optimal layout?

- No known tight bound
- Compare against the best layout we can construct
  - Employ an efficient sampling method

Use it

Build a new one
Outline

- Computation models
- Cache-aware and cache-oblivious metrics
- Results
Layout Computation Time

- Process 70 million vertices per hour
  - Takes 2.6 hours to lay out St. Matthew model (372 million triangles)
  - 2.4GHz of Pentium 4 PC with 1 GB main memory
Our Layout of Bunny Model

Computed layout
Applications

- Isosurface extraction
- View-dependent rendering
- Collision detection
- Ray tracing
Iso-Surface Extraction

- Uses contour tree [van Kreveld et al. 97]
  - Runtime is dominated by the traversal of isosurface
- Layout graph
  - Use an input tetrahedral mesh

Spx model (140K vertices)
High Correlation with Number of Cache Misses

Tested block size = 4KB

<table>
<thead>
<tr>
<th>Linear correlation [-1, 1]</th>
<th>Observed number of cache misses</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>With 1 cache block</td>
</tr>
<tr>
<td>Geometric CO metric</td>
<td>0.99</td>
</tr>
</tbody>
</table>

Tested with 8 different layouts: our geometric CO, our cache-aware, breadth-first (and depth-first) layouts, spectral [Juvan and Mohar 92], cache-oblivious mesh [Yoon et al. 05], Z-curve [Sagan 94], X-axis sorted layouts
High Correlation with Runtime Performance

<table>
<thead>
<tr>
<th>Linear correlation [-1, 1]</th>
<th>First iso-surface extraction time</th>
<th>Second iso-surface extraction time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Geometric CO metric</td>
<td>0.94</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Disk I/O time is major bottleneck
Memory access time is major bottleneck
Comparison with Other Layouts

The first iso-surface extraction time (sec)

Cache-aware layout  
Geometric CO layout  
Z-curve  
COML  
Depth-first layout  
Spectral layout  
Breadth-first layout  
X-axis layout

8% - 77% improvement and very close to the cache-aware performance
Isocontour Extraction – Puget Sound Model, 134M Triangles

Isocontour

$z(x,y) = 500m$
Comparison – First Extraction of $Z(x,y) = 500m$

Disk access time is bottleneck

Relative Performance over Z-axis sorted layout

- Cache-oblivious layout: 2
- Z-axis sorted: 1
- Y-axis sorted: 21
- Spectral layout: 13

Nearly optimized for particular isocontour
Comparison – Second Extraction of $Z(x,y) = 500m$

Relative Performance over Z-axis sorted layout

Memory and L1/L2 cache access times are bottleneck
View-Dependent Rendering

- Layout vertices and triangles of CHPM [Yoon et al. VIS 04]
  - Reduce misses of GPU vertex cache
# View-Dependent Rendering

Peak performance: 145 M tri / s on GeForce 6800 Ultra

<table>
<thead>
<tr>
<th>Models</th>
<th># of Tri.</th>
<th>Our layout</th>
<th>Simplification layout</th>
</tr>
</thead>
<tbody>
<tr>
<td>St. Matthew</td>
<td>372M</td>
<td>106 M/s</td>
<td>23 M/s</td>
</tr>
<tr>
<td>Isosurface</td>
<td>100M</td>
<td>90 M/s</td>
<td>20 M/s</td>
</tr>
<tr>
<td>Double Eagle Tanker</td>
<td>82M</td>
<td>47 M/s</td>
<td>22 M/s</td>
</tr>
</tbody>
</table>
Comparison with Other Rendering Sequences

- Universal rendering sequences [Bogomjakov and Gotsman 2002]
- Our layout

Graph showing cache miss ratio (misses per triangle) vs. vertex cache size.
Comparison with Other Rendering Sequences

Cache miss ratio (misses per triangle)

- Optimized for 16 vertex cache size with FIFO replacement
- [Hoppe 99]
- Our layout
- Optimized for no particular cache size
Performance during View-Dependent Rendering

Cache miss ratio (given cache size 32)

Optimized for full resolution

Optimized for various resolutions

Our layout

[Hoppe 99]
Cache Miss Ratio on Bunny Model

- GPU vertex cache miss ratio
- Universal rendering seq. [Bogomjakov and Gotsman 02]
- Theoretical lower bound [Bar-Yehuda and Gotsman 96]
- Hoppe [Hoppe 99]
- Geometric CO layout

Vertex cache size
Cache Miss Ratio on Power Plant Model

GPU vertex cache miss ratio

Theoretical lower bound [Bar-Yehuda and Gotsman 96]

Z-curve

COML [Yoon et al. 05]

Hoppe’s rendering seq. [Hoppe 99]

Geometric CO layout

Vertex cache size
Collision Detection

- Use oriented bounding box (OBB) [Gottschalk et al. 96]
  - Breadth-first tree traversal

- Use an input graph representing well the runtime access pattern on the hierarchy
Collision Detection – Robot and Power Plant Models

20k triangles

1M triangles
Collision Detection – Performance Comparison I

41% ~ 500% performance improvement

Working set size (KB)

Collision time (ms/100)

Our cache-oblivious layout

Different layouts

Cache-oblivious layout

Depth-first layout

KAIST
Collision Detection – Performance Comparison II

35% ~ 2600% performance improvement

Different layouts

- Our layout
- So many different layout variations
- Not all the variations make sense
- Our performance comparison

Different layouts

- Working set size (KB)
- Cache-oblivious mesh layout (ms/100)
- Depth-first layout

KAIST
Cache-Oblivious Layout vs Cache-Aware Layout

- Cache-aware layouts
  - Take advantage of block size information (4KB)
- Minor performance degradation
  - 8% compared to cache-aware layouts
Ray Tracing

- Use kd-tree [Wald 04]
  - Depth-first tree traversal
Ray Tracing – Lucy Model

- 28 million triangles
- Pentium IV with 1GB
Ray Tracing – Performance Comparison

77% ~ 180% performance improvement

Different layouts

- Our layout
- van Emde Boas layout
- Breadth-first layout
- Depth-first layout

Render time (sec)

Working set size (MB)
Advantages

● General
  ● Applicable to all kinds of polygonal models
  ● Works well for various applications

● Cache-oblivious
  ● Can have benefit from CPU/ GPU cache to memory and disk

● Robust performance improvement

● No modification of runtime application
  ● Only layout computation
OpenCCL: Cache-Coherent Layouts of Graphs and Meshes

- Source codes for computing a cache-coherent layout
  - Easy to use
  - Google “Cache Coherent Layouts”

```c
CLayoutGraph Graph (NumVertex);

Graph.AddEdge (0, 1);
Graph.AddEdge (0, 2);
Graph.AddEdge (1, 2);

int Order [NumVertex];
Graph.ComputeOrdering (Order);
```
Summary

- Novel cache-aware and cache-oblivious metrics to evaluate layouts
  - Derived metrics based on two-level I/O model
  - Improved the performance of applications without modifying codes
Ongoing and Future Work

- Derive a lower bound on our geometric cache-oblivious metric
- Employ mesh compression to further reduce disk I/O accesses
- Investigate efficient layout method for deforming/dynamic models
- Apply to non-graphics applications
  - e.g., shortest path or other graph computations
- Apply to other representations such as R-tree
At the Next Class

- Will discuss data compression