Cache-Efficient Visualization

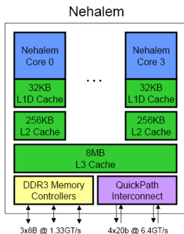
Marc Tchiboukdjian
Vincent Danjean
Bruno Raffin

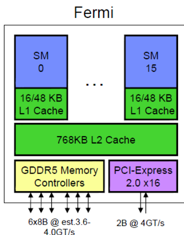
MOAIS

Scientific Visualization

- Precise simulations → **huge** data sets
- Bottleneck for visualization filters: **data transfers**

Speedup Visualization by Efficient Use of Caches





Memory wall

computation speed $\gg$ memory speed

Hardware Solution

- **Caches** hierarchy
- **Complex** architecture with many parameters

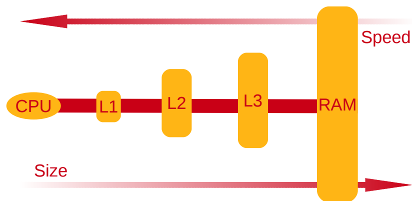
Challenge: efficient use of hardware

- Efficient use of caches (memory access **locality**)
- Without tuning of specific parameters (cache size)
- Performance **guarantee**

Overview

1. Introduction
2. Understanding Caches
3. Techniques to improve locality
4. Cache-Oblivious Mesh Layout
5. Isosurface Extraction using a Coherent Min-Max Tree
6. Conclusion

The Memory Hierarchy



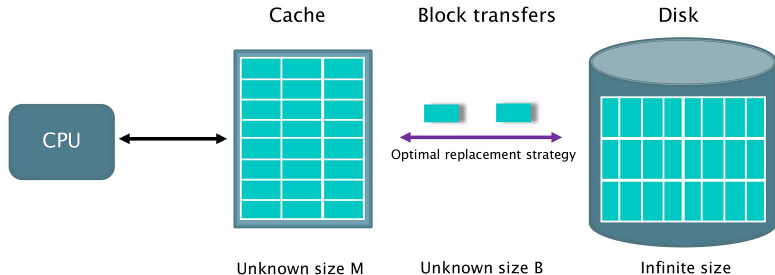
Memory	Size	Latency
L1	16kB	1 cycle
L2	256kB	20 cycles
L3	8MB	50 cycles
RAM	8GB	200 cycles

(Intel Nehalem)

Characteristics

- **Automatically** managed by the CPU
- Transfer by **blocks** or cache lines (generally 64B)
- When a data is not in cache: **cache miss**
- Replacement algorithm (eg. LRU):
  evict a cache line and load the new one

The Cache-Oblivious Model [FLPR99]



- Performance: number of **block transfers** (cache misses)
- Model **locality** of memory accesses
- Architecture **independent**

1 Introduction

2 Understanding Caches

3 Techniques to improve locality

4 Cache-Oblivious Mesh Layout

5 Isosurface Extraction using a Coherent Min-Max Tree

6 Conclusion

Locality of memory accesses

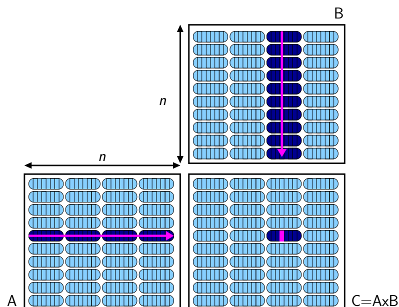
Locality of memory accesses

- **Spatial locality**: use all data contained in a block
- **Temporal locality**: reuse as soon as possible

2 techniques to improve locality

- Computation reordering
- Data reordering

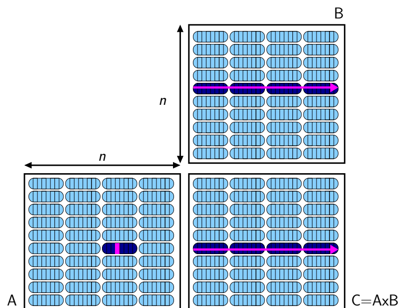
Example: matrix multiplication




```
for ( int i = 0 ; i < n ; ++i )
  for ( int j = 0 ; j < n ; ++j )
    for ( int k = 0 ; k < n ; ++k )
      C[i,j] += A[i,k] * B[k,j]
```

Naive matrix multiplication

- $n^2 \cdot (\frac{n}{B} + n) = O(n^3)$ cache misses

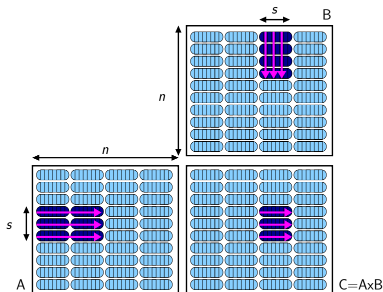



```
for ( int i = 0 ; i < n ; ++i )
  for ( int k = 0 ; k < n ; ++k )
    for ( int j = 0 ; j < n ; ++j )
      C[i,j] += A[i,k] * B[k,j]
```

Modified matrix multiplication

- Improved spatial locality
- $n^2 \cdot 2 \cdot \frac{n}{B} = O(\frac{n^3}{B})$ cache misses

Computation reordering for matrix multiplication




```
for ( int ii = 0 ; i < n/s ; ++ii )
 for ( int jj = 0 ; j < n/s ; ++jj )
  for ( int kk = 0 ; kk < n/s ; ++kk )
   for ( int i = ii*s ; i < (ii+1)*s ; ++i )
    for ( int j = jj*s ; j < (jj+1)*s ; ++j )
     for ( int k = kk*s ; k < (kk+1)*s ; ++k )
      C[i,j] += A[i,k] * B[k,j]
```

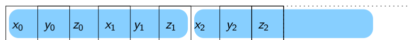
Matrix multiplication by blocks

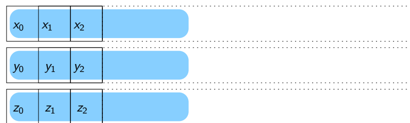
- Improved temporal locality
- Cache misses : $(\frac{n}{s})^2 \cdot \frac{n}{s} \cdot 2 \cdot \frac{s^2}{B} = \frac{3n^3}{sB}$
- 3 submatrices should fit in cache : $3s^2 \leq M \longrightarrow s = \sqrt{\frac{M}{3}}$
- Cache misses : $O(\frac{n^3}{B\sqrt{M}})$

Impact of Data Layout: work on mesh points

```
struct point {
  double x,y,z ;
} ;
point mesh[n] ;
```

```
struct mesh {
  double x[n] ;
  double y[n] ;
  double z[n] ;
}
```






```
for ( int i = 0 ; i < n ; ++i )
  mesh[i].x *= 2 ;
for ( int i = 0 ; i < n ; ++i )
  mesh[i].y *= 2 ;
for ( int i = 0 ; i < n ; ++i )
  mesh[i].z *= 2 ;
```

```
for ( int i = 0 ; i < n ; ++i )
  mesh.x[i] *= 2 ;
for ( int i = 0 ; i < n ; ++i )
  mesh.y[i] *= 2 ;
for ( int i = 0 ; i < n ; ++i )
  mesh.z[i] *= 2 ;
```

```
for ( int i = 0 ; i < n ; ++i ) {
  mesh[i].x *= 2 ;
  mesh[i].y *= 2 ;
  mesh[i].z *= 2 ;
}
```

```
for ( int i = 0 ; i < n ; ++i ) {
  mesh.x[i] *= 2 ;
  mesh.y[i] *= 2 ;
  mesh.z[i] *= 2 ;
}
```

1 Introduction

2 Understanding Caches

3 Techniques to improve locality

4 Cache-Oblivious Mesh Layout

5 Isosurface Extraction using a Coherent Min-Max Tree

6 Conclusion

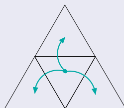
Common Mesh Access Patterns of Visualization Filters

Visualization Filter

A function f to apply to all or a subset of points or cells of the mesh

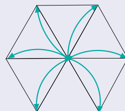
Cell Neighborhood

- ex: vtkConnectivityFilter



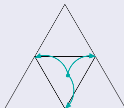
Point Neighborhood

- ex: vtkGradientFilter



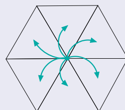
Cell Attributes

- ex: vtkMarchingCube



Point Attributes

- ex: vtkCellDataToPointData



Common Mesh Access Patterns of Visualization Filters

Layout order traversal

- Access points/cells in the order given by the layout
- ex: Marching Cubes

```
for ( int i = 0 ; i < n ; ++i )
  f(i) ;
```

Connectivity traversal

- Access points/cells through neighborhood links
- ex: Raycasting

```
int i = init() ;
while ( not_done(i) ) {
  f(i) ;
  i = neighbor(i) ;
}
```

External data structure traversal

- Access the mesh through an external data structure
- ex: Isosurface extraction with min-max tree

Cache-Oblivious Mesh Layout [TDR10]

Idea

Store elements connected in the mesh close in memory.

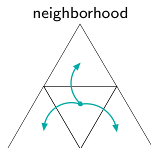
Why?

- Cell/Point neighborhood and connectivity traversal: improved spatial locality
- Layout order traversal + cell/point attributes: improved temporal locality
- External data structure traversal: depends on the data structure

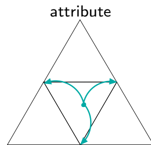
How?

Renumber points/cells so that points/cells connected in the mesh have close indices.

neighborhood



attribute



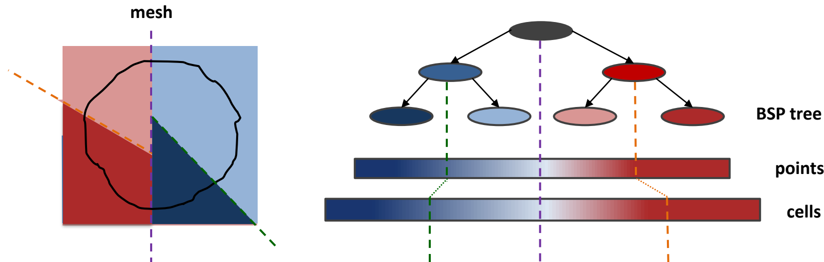
layout order

```
for(int i=0; i<n; ++i)
  f(i) ;
```

connectivity

```
int i = init();
while( not_done(i) ){
  f(i);
  i = neighbor(i);
}
```

Cache-Oblivious Mesh Layout [TDR10]



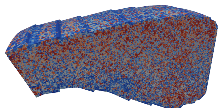
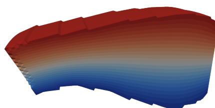
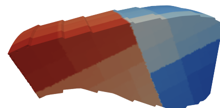
Algorithm

Recursively separate the mesh in two parts while cutting few cells.

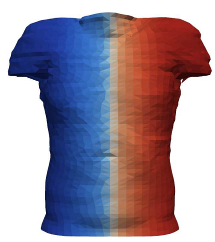
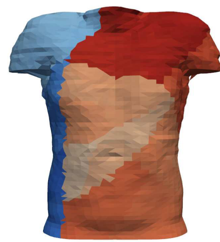
Performance guarantee

- Computing the layout: $O(n \log n)$
- Using the mesh: $O(\frac{n}{B} + \frac{n}{M^{1/3}})$ cache misses

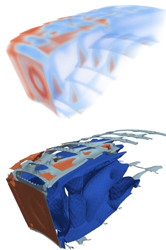
Examples of Generated Layouts

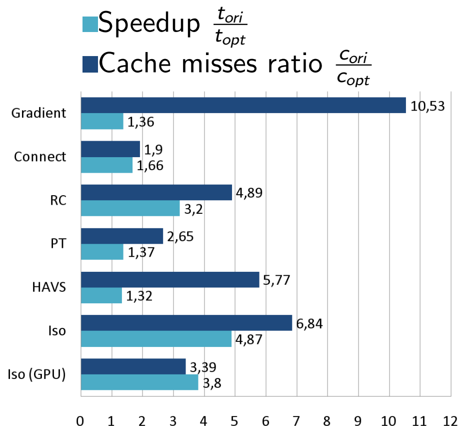




Tetgen | Geometric Sort | Cache-Oblivious

Cache-Oblivious Mesh Layout: experimental results





- Non modified VTK filters except Iso (home-made)
- Speedup of the CO mesh compared to the original mesh

1 Introduction

2 Understanding Caches

3 Techniques to improve locality

4 Cache-Oblivious Mesh Layout

5 Isosurface Extraction using a Coherent Min-Max Tree

6 Conclusion

Isosurface Extraction with a Min-Max Tree

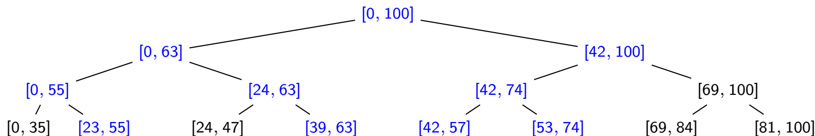
Marching Cube algorithm

- For each cell, interpolate the isosurface inside the cell
- Examine all cells, even the ones not traversed by the isosurface

Min-Max Tree

- Recursively divide the mesh into regions
- Store for each region, the min and max value of the scalar field
- If the isovalue is not in the interval $[\min, \max]$, the region can be discarded

Isovalue $= 54$. Only blue intervals are examined.



Choice of the Min-Max Tree

Divide the mesh into geometric regions

- Cells in the same geometric area often have close scalar values
  $\longrightarrow$ Intervals are small, **more regions can be discarded**
- For each leaf region, store the list of cells in this region
- Cells of the same region could be clustered in the layout
  $\longrightarrow$ **poor locality**

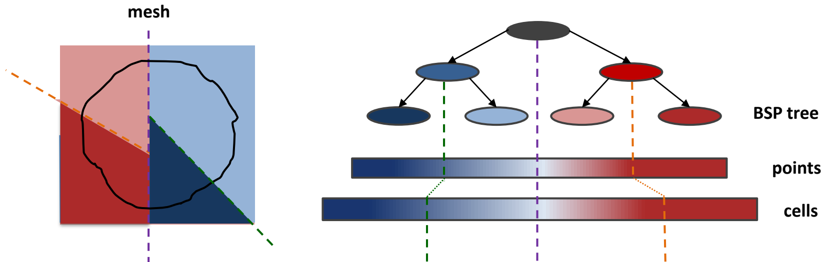
Divide the mesh using the layout

- No need to store the list of cells in a leaf region (cells $[i, j]$)
- Cells in the same region are contiguous in the layout
  $\longrightarrow$ **good locality**
- Cells of the same region could be clustered in the mesh
  $\longrightarrow$ **few discarded regions**
- Strategy used by the vtkSimpleTree

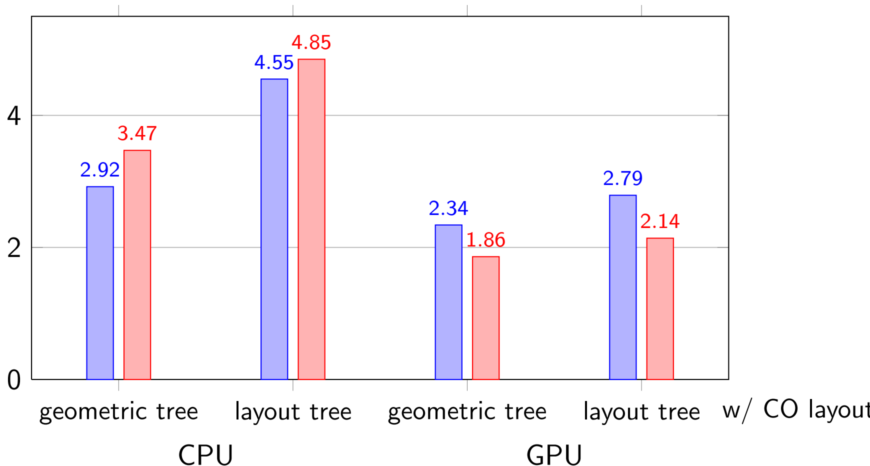
Coherent Min-Max Tree [TDR10]

Combine advantages of both trees

- Divide the mesh into regions that are both contiguous in the layout and based on the geometry
  $\longrightarrow$ **good locality**, **many discarded regions**, **low memory usage**
- Take the tree used to compute the layout



Coherent Min-Max Tree: Experimental results



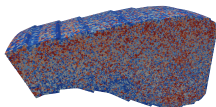
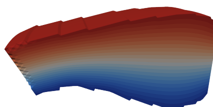
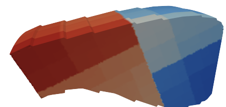
Conclusion

Cache-Efficient Visualization

- Cache-Oblivious mesh layout
- Performance guarantee
- Architecture independent
- Experimental validation on many visualization filters
- Coherent min-max tree for isosurface extraction

Future work

- Locality for parallel programs on multicore processors
- Cache-efficient volume rendering by ray casting

M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran.
Cache-Oblivious Algorithms.
In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, page 285, 1999.

M. Tchiboukdjian, V. Danjean, and B. Raffin.
Binary mesh partitioning for cache-efficient visualization.
Transactions on Visualization and Computer Graphics, (PrePrints), 2010.