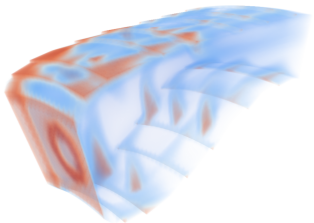
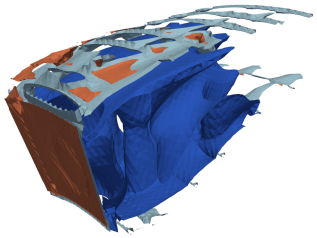
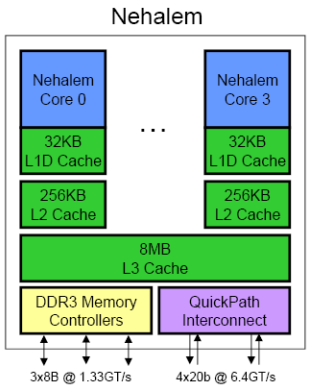
Cache-Efficient Parallel Isosurface Extraction for Shared Cache Multicores

Marc Tchiboukdjian
Vincent Danjean
Bruno Raffin

Memory Issues for Visualization Filters







Visualization filters are often memory bounded

- Visualization filters are memory intensive
- Bottleneck: memory bandwidth/latency

Even worse on multicores

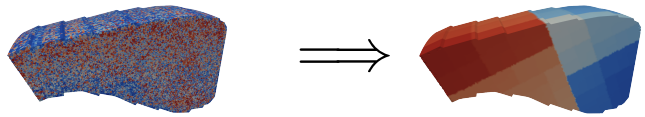
- High number of cores sharing the same memory
- Memory bandwidth and cache size do not scale

Better cache usage $\Longrightarrow$ speedup

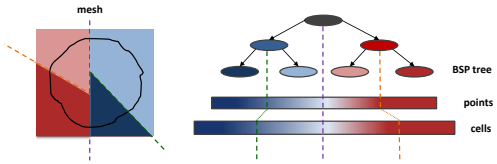
This Talk

Speedup visualization filters by efficient use of caches

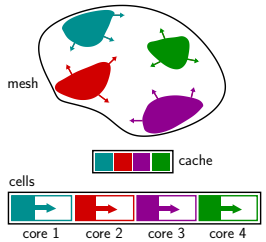
1. Cache-efficient mesh layouts



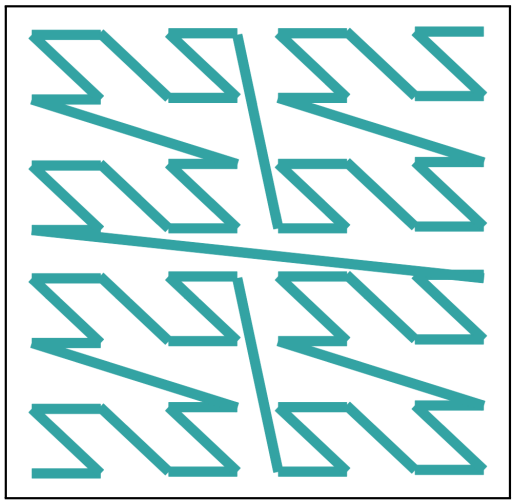
2. Isosurface extraction with a coherent min-max tree



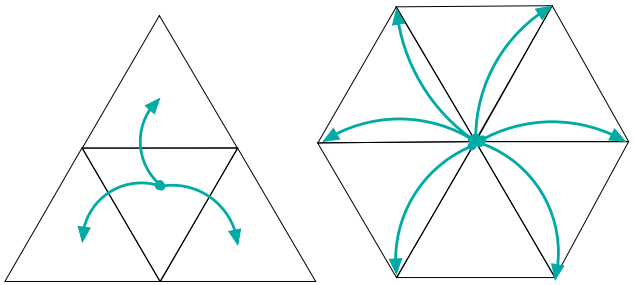
3. Parallel isosurface extraction for multicores



Layout for Regular Grids: Space-filling Curves [Pascucci 01]



Z-curve



Classical access patterns

Space-filling curves

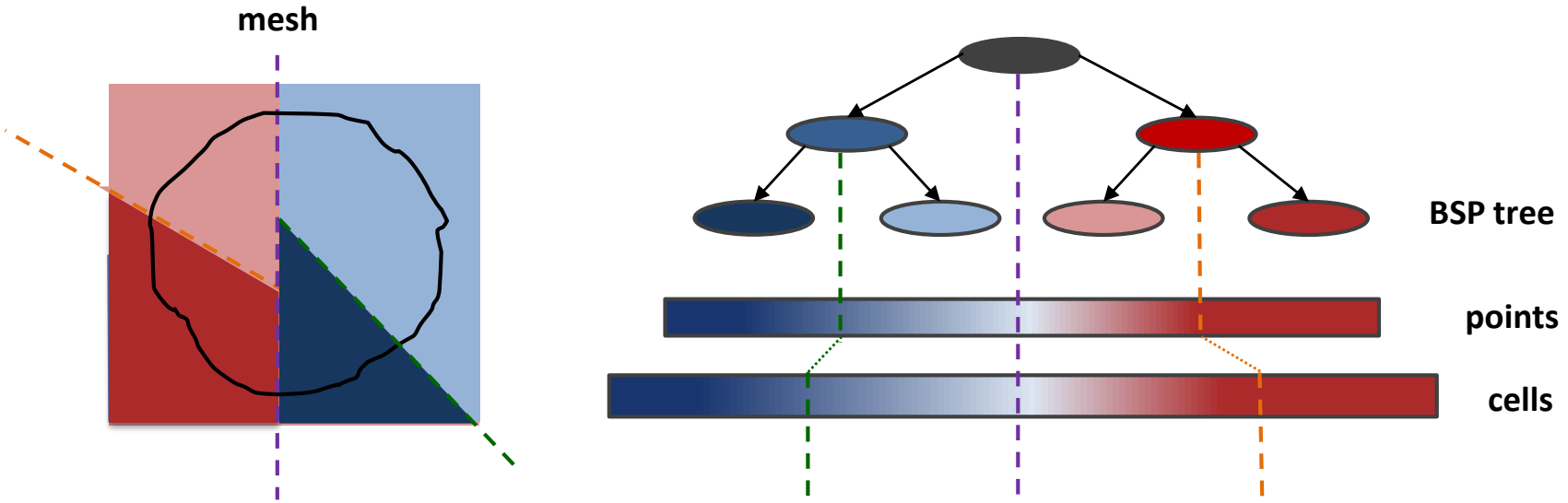
- ex: Z curve, Hilbert curve, etc.
- Map 2D/3D indexes to 1D indexes
- Keep **locality**
  close in 2D/3D $\Rightarrow$ close in 1D

Space-filling curves to index meshes

- Elements close in the mesh are close in memory
- Filters have often spatially coherent access patterns

$\Rightarrow$ Improved cache usage

Layout for Unstructured Meshes: FastCOL



FastCOL Algorithm (TVCG 2010)

- Recursively cut the mesh while minimizing the cut
- Store contiguously elements in the same node of the BSP tree
- Layout computation: $O(n \log n)$

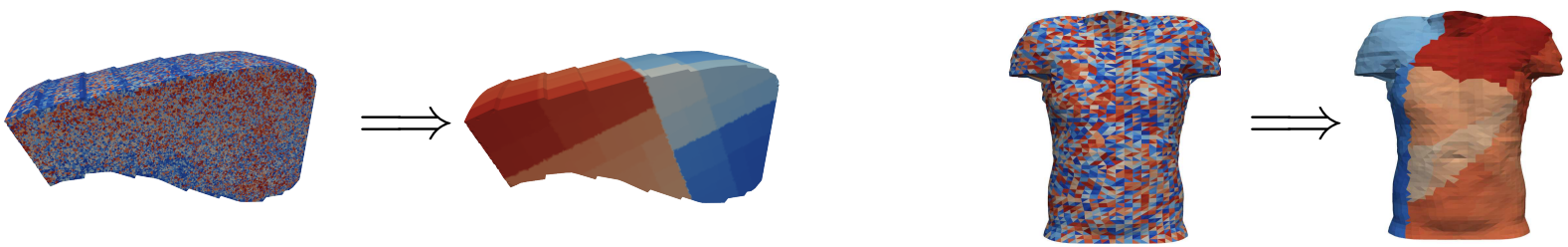
Layout for Unstructured Meshes: FastCOL

The FastCOL layout guarantees a cache-efficient traversal for spatially coherent filters whatever the cache size (cache-oblivious).

$$Q_{B,M}(S) = \frac{S}{B} + O\left(\frac{S}{M^{1/3}}\right)$$

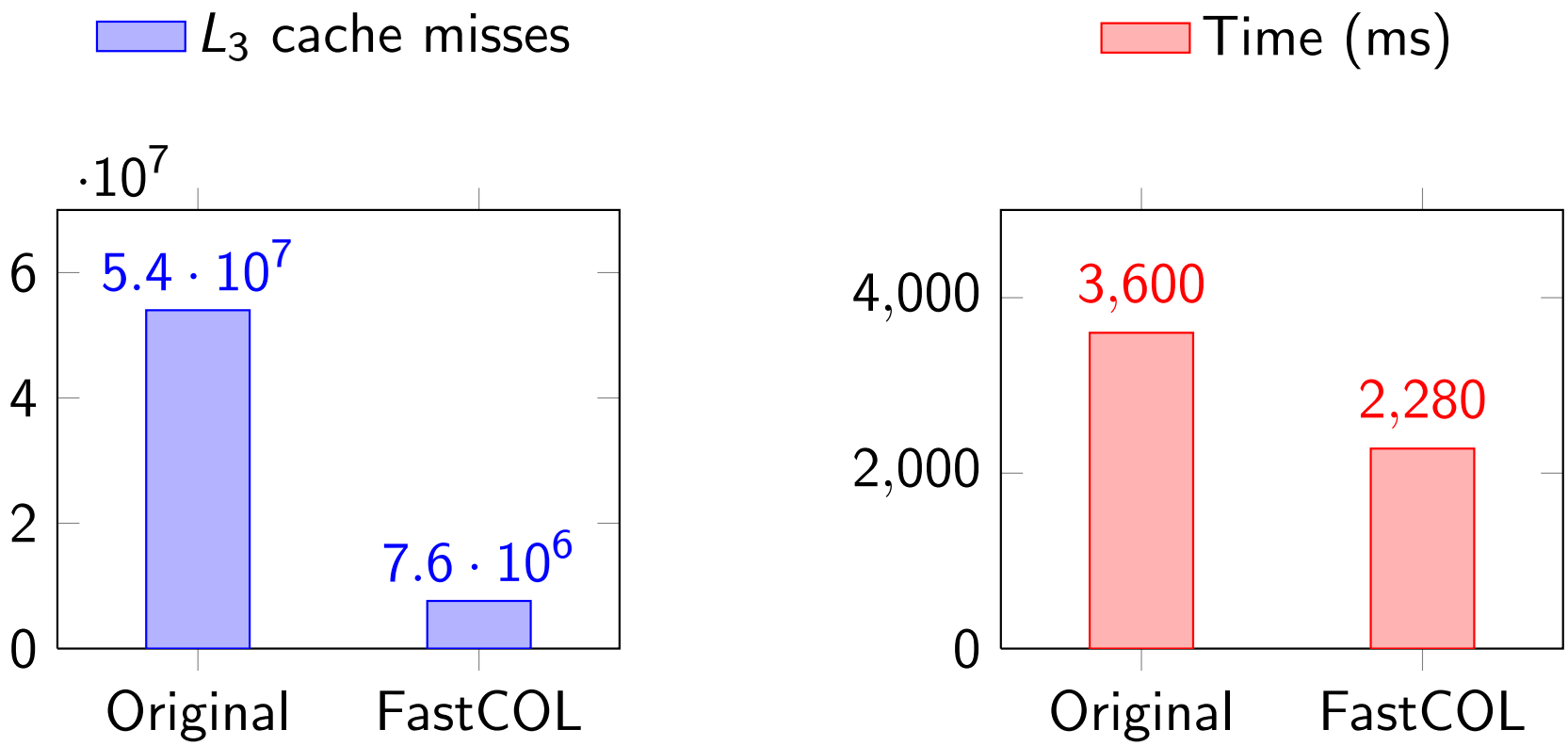
read mesh — overhead

S: mesh size
B: cache line size
M: cache size



Example of layouts
(Cells close in memory have the same color)

Marching Tetrahedra (MT): Original vs FastCOL



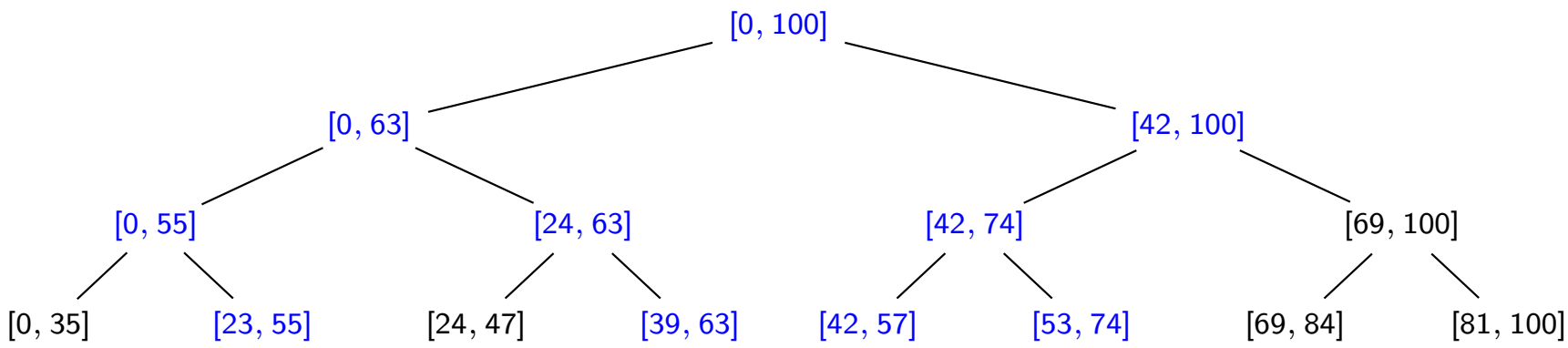
One core of a Nehalem with 8MB of L_3 cache

Isosurface Extraction with a Min-Max Tree

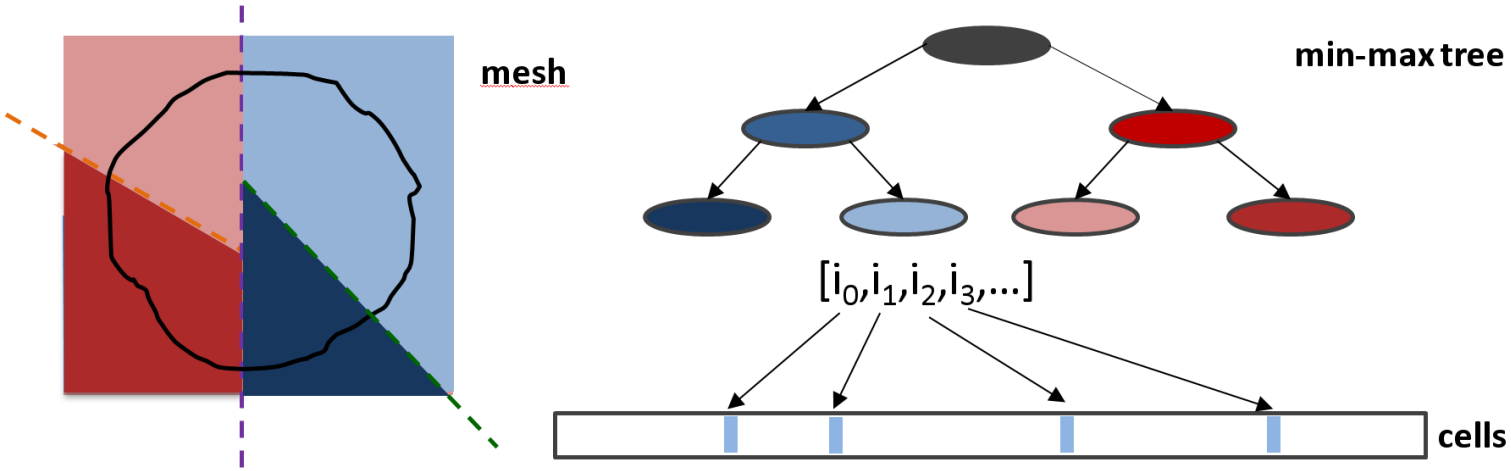
Min-Max Tree

- Recursively divide the mesh into regions
- Store for each region, the min and max value of the scalar field
- Isovalue $\notin [\min, \max] \Rightarrow$ discard the region

If isovalue $= 54$, only blue intervals are examined.



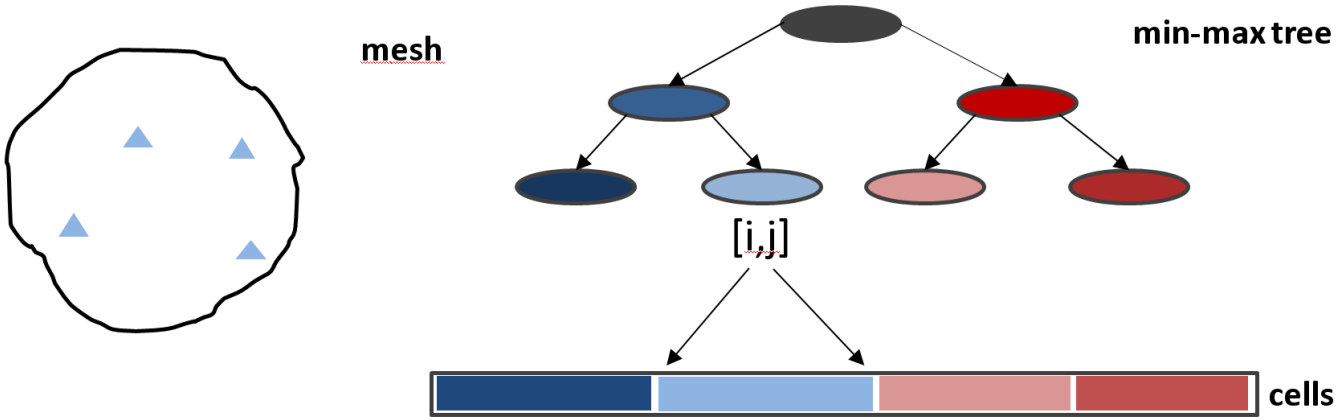
Geometric Min-Max Tree



Divide the mesh into geometric regions

- ex: octree, kd-tree, etc.
- Cells in the same geometric area often have close scalar values
  Many discarded regions
- For each leaf, store the list of cells in this region
  High memory usage
- Cells of the same region could be scattered in the layout
  Not coherent with the layout, poor locality

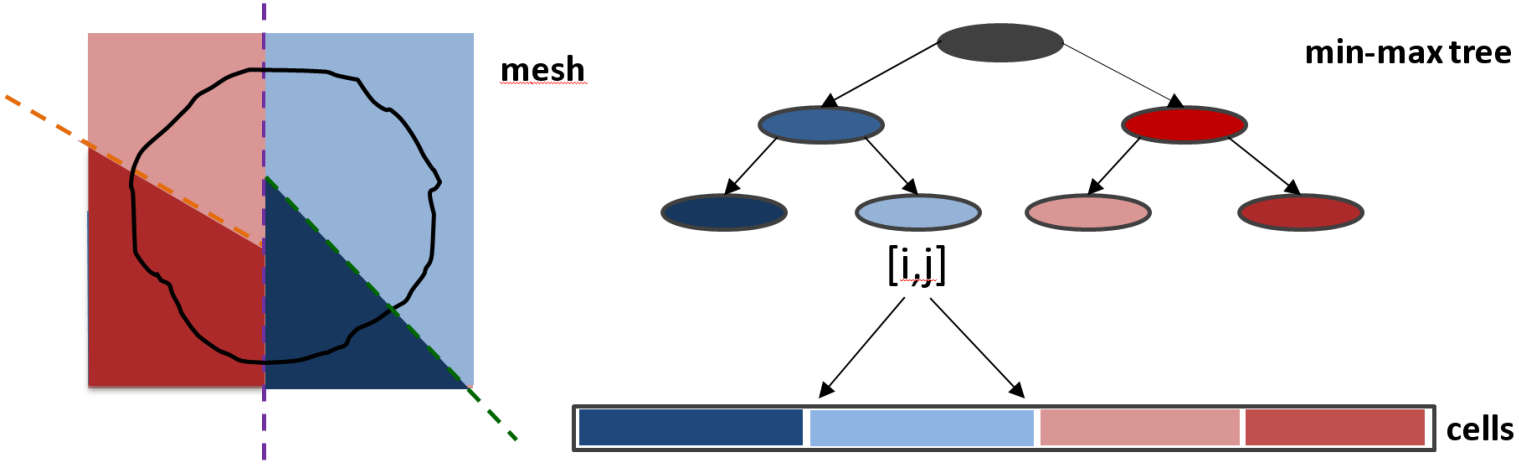
Layout-based Min-Max Tree



Divide the mesh using the layout

- Strategy used by the vtkSimpleTree
- No need to store the list of cells in a leaf (cells $[i, j]$)
  low memory usage
- Cells in the same region are contiguous in the layout
  Coherent with the layout, good locality
- Cells of the same region could be scattered in the mesh
  few discarded regions

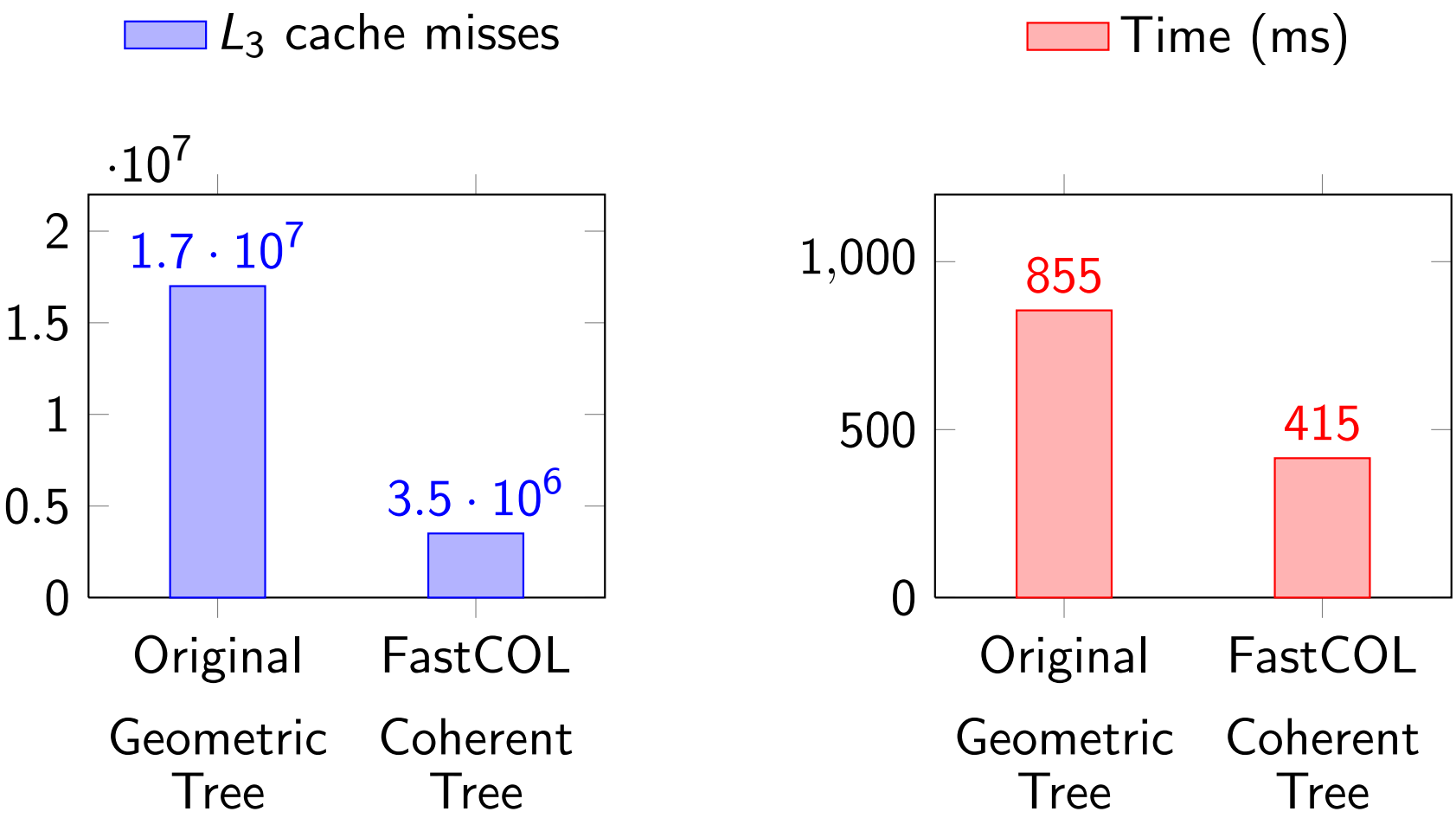
Coherent Min-Max Tree: Combine Both



Take the BSP tree used to compute the FastCOL layout

- Regions are contiguous in the layout and geometry based
  Good locality, many discarded regions, low memory usage
- Low memory usage: for a 150M tets mesh (2.6GB)
  Geometric min-max tree $\rightarrow$ 958MB
  Layout-based and coherent min-max tree $\rightarrow$ 385MB

Isosurface Extraction with Coherent Min-Max Tree

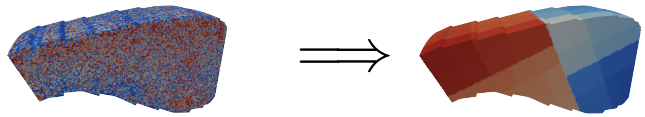


One core of a Nehalem with 8MB of L_3 cache

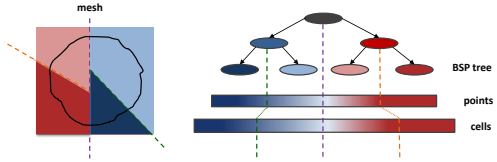
This Talk

Speedup visualization filters by efficient use of caches

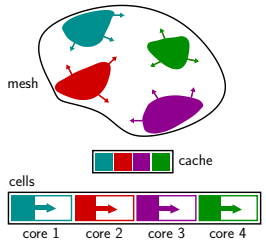
1. Cache-efficient mesh layouts



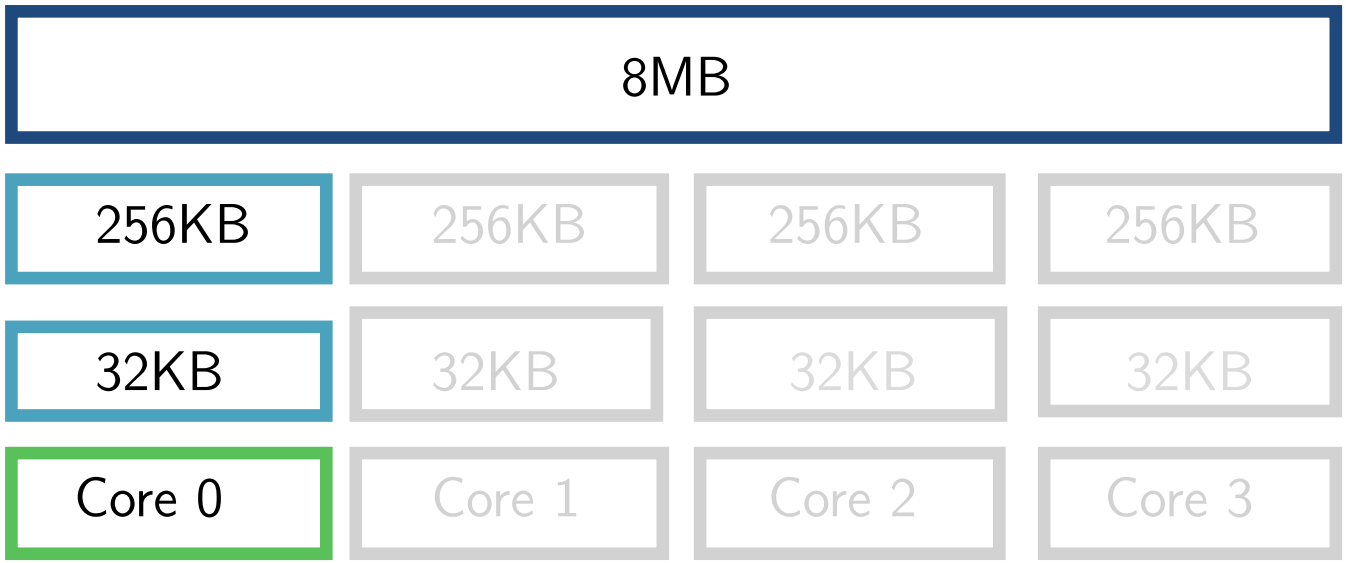
2. Isosurface extraction with a coherent min-max tree



3. Parallel isosurface extraction for multicores

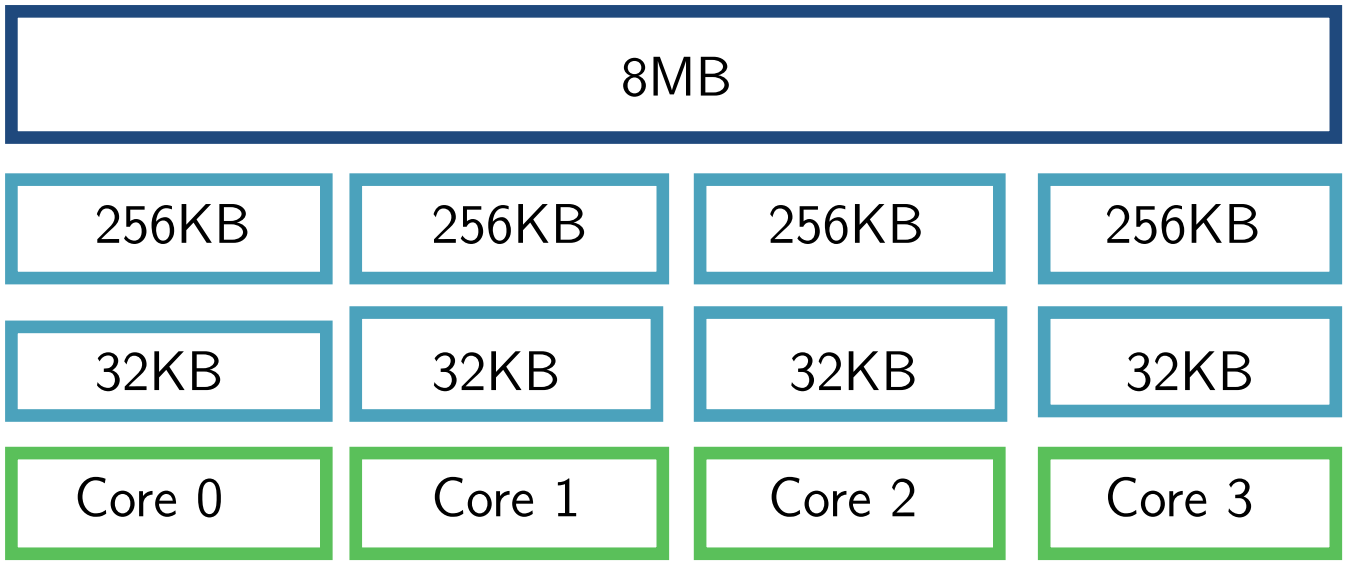


Keep Good Cache Performance in Parallel



Nehalem (Xeon E5530)

Keep Good Cache Performance in Parallel



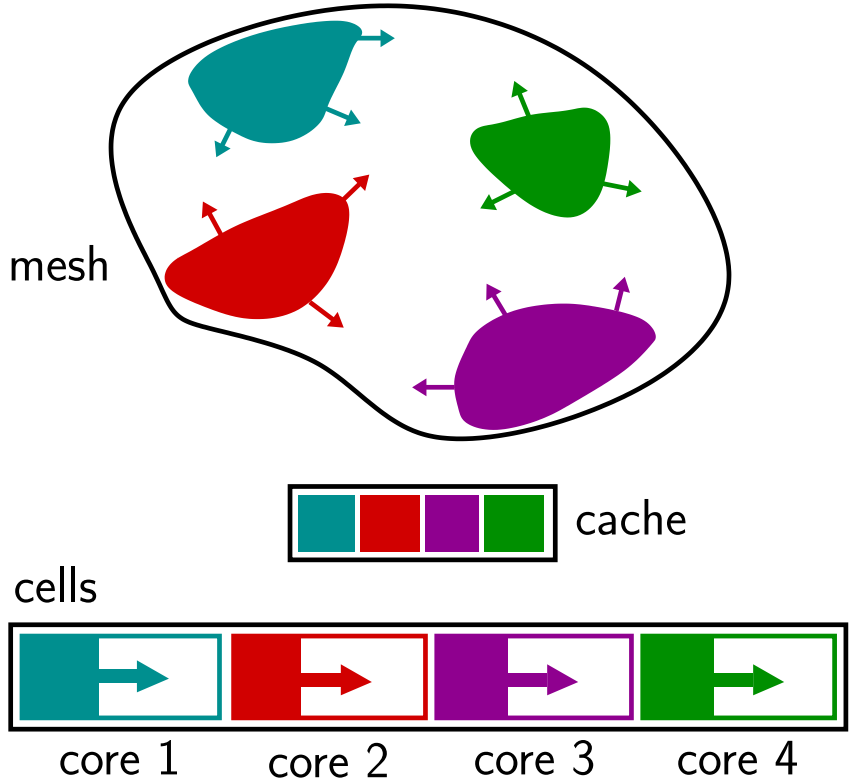
Nehalem (Xeon E5530)

- Take advantage of multiple cores
- No extra cache misses
- **Cores share the last cache level**

Classical Scheme

Split Cache Strategy

- Divide n tets into p chunks
- Cores compete for shared cache space



Performance of Split Cache

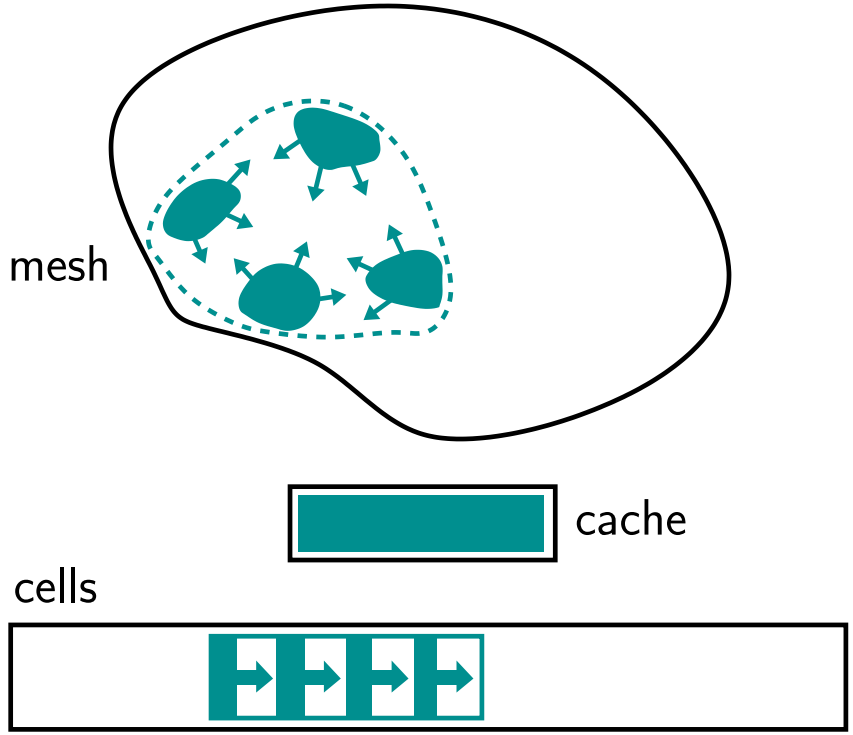
- Cache size: $M \rightarrow \frac{M}{p}$
- Cache misses: $\frac{S}{B} + p^{1/3} \cdot O\left(\frac{S}{M^{1/3}}\right)$

overhead

Shared Cache Aware Scheme

Shared Cache Strategy

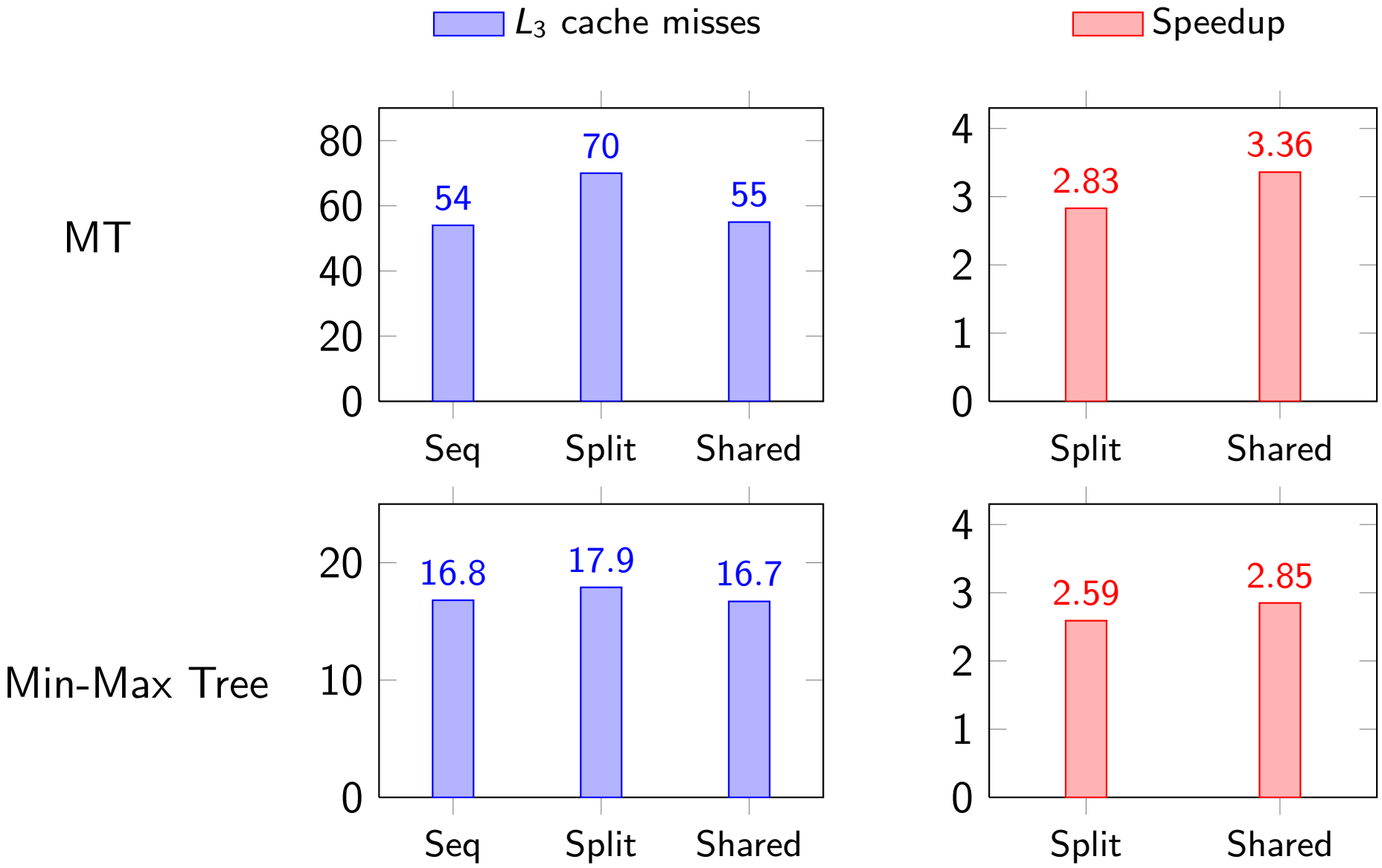
- Divide into chunks fitting in the shared cache
- Cores work in parallel inside a chunk
- Cores benefit from data cached by others



Shared Cache vs Split Cache

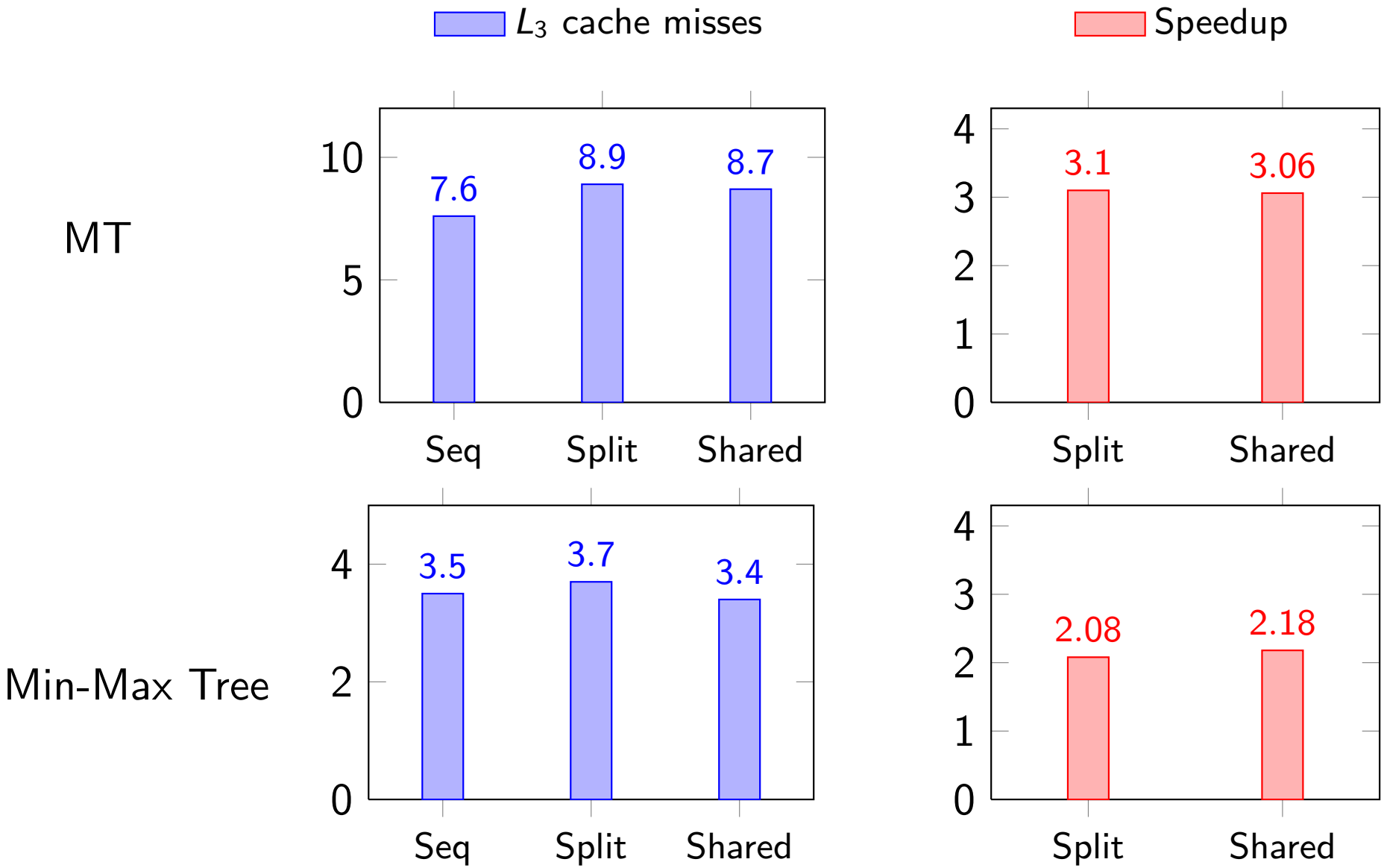
- Less cache misses
  Proof: **no more cache misses than sequential algorithm**
- More synchronizations

Parallel Isosurface Extraction on Original Layout



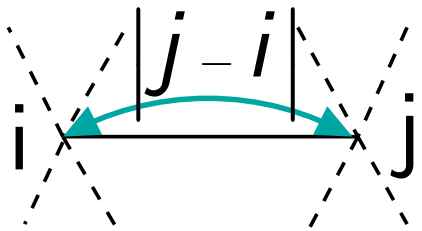
Nehalem quadcore with 8MB of shared L_3 cache

Parallel Isosurface Extraction on FastCOL Layout

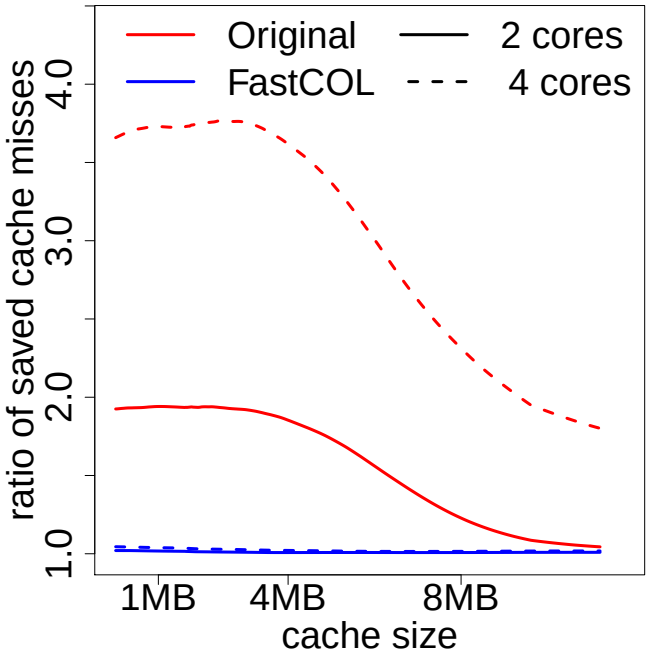


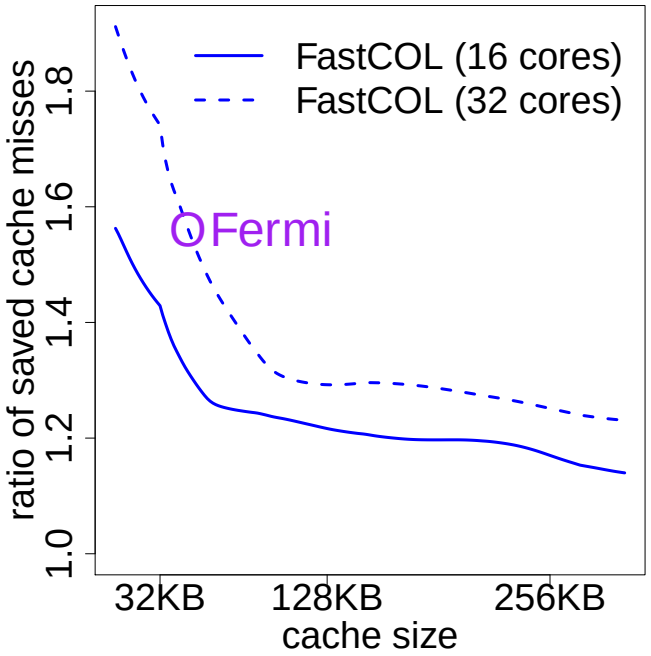
Nehalem quadcore with 8MB of shared L_3 cache

Split Cache vs Shared Cache: Performance Gain Evaluation



ratio of saved cache misses $\approx \dfrac{\sum_{e \in E} \mathbb{1}_{\lambda_e > M}}{\sum_{e \in E} \mathbb{1}_{\lambda_e > \frac{M}{p}}}$ ⟵ shared cache / ⟵ split cache





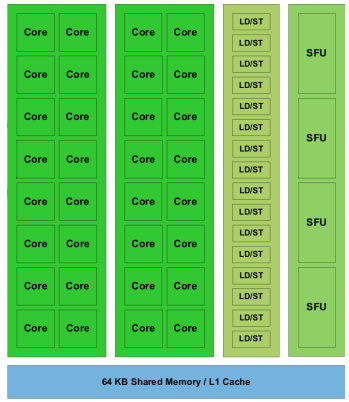
Gain of shared cache over split cache increases

- with the number of cores
- when the cache size decreases

Conclusion

Cache-Efficient Isosurface Extraction

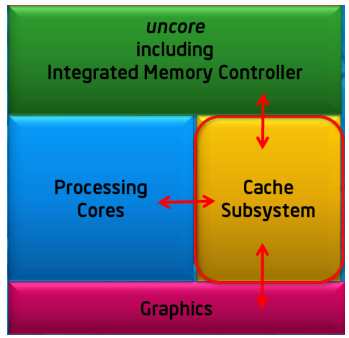
- Coherent min-max tree for isosurface extraction: fast, low memory usage, practical
- Provable performances in sequential and parallel



Fermi SP

Intra-Chip Parallelism

- Shared Cache $>$ Split Cache
  - cooperative vs competitive strategy
  - gain increases with the number of cores
  - gain increases with mesh size / cache size ratio
- Advantage of Shared Cache will likely grow in the future (Nvidia Fermi, Intel Sandy Bridge)



Sandy Bridge