Autotuning (1/2): Cache-oblivious algorithms

Prof. Richard Vuduc

Georgia Institute of Technology

CSE/CS 8803 PNA: Parallel Numerical Algorithms

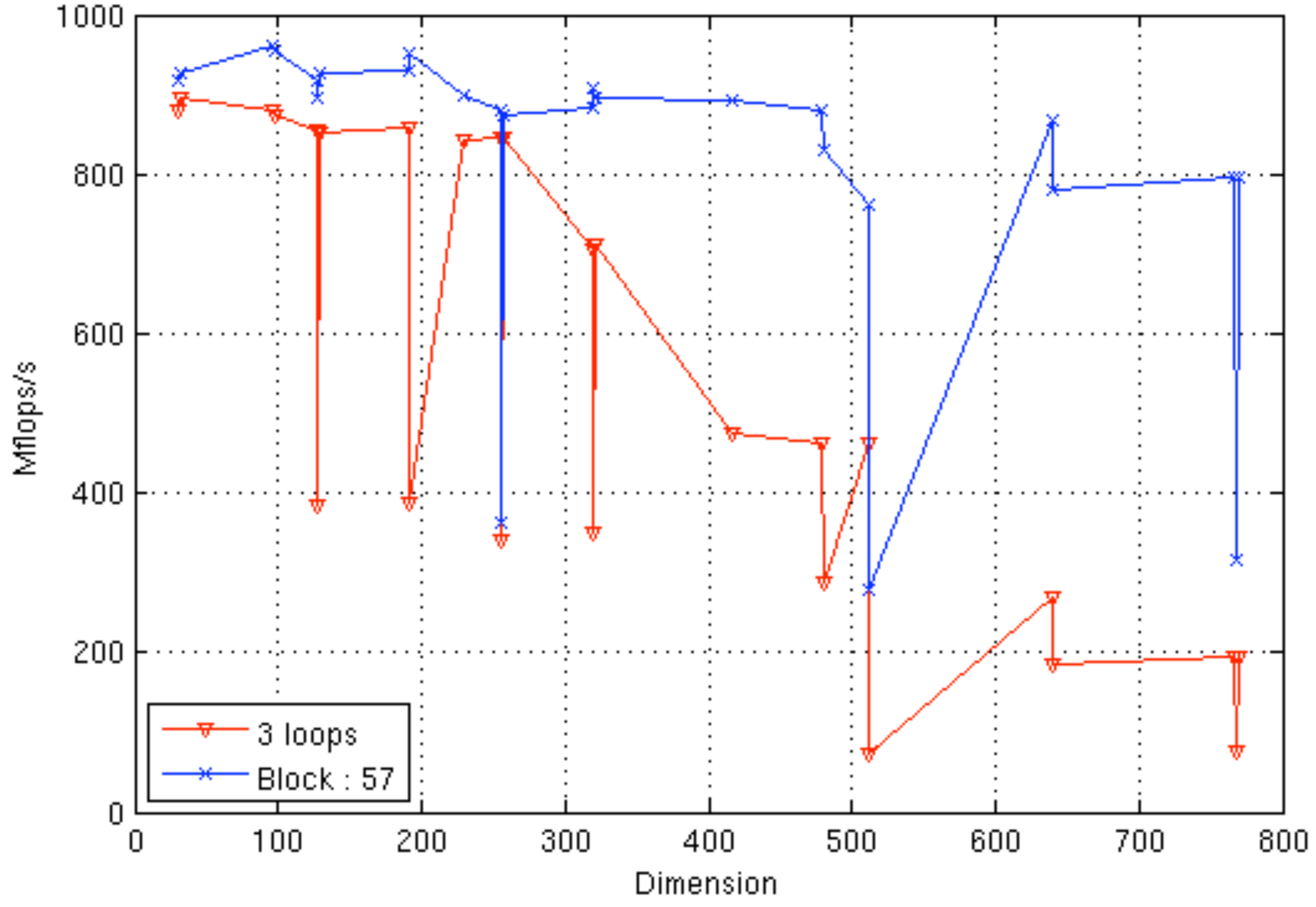
[L.17] Tuesday, March 4, 2008

Today’s sources

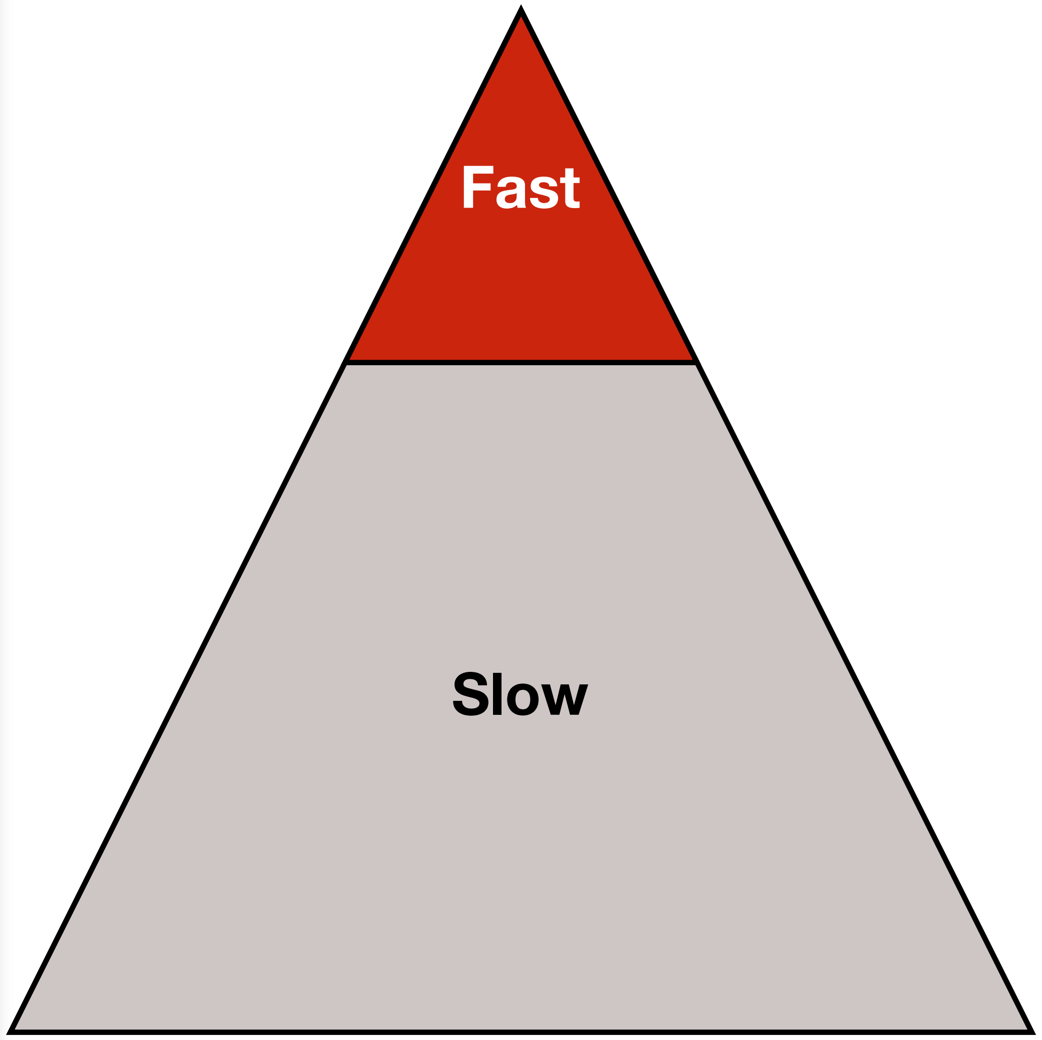
- CS 267 (Demmel & Yelick @ UCB; Spring 2007)
- “*An experimental comparison of cache-oblivious and cache-conscious programs?*” by Yotov, *et al*. (SPAA 2007)
- “*The memory behavior of cache oblivious stencil computations*,” by Frigo & Strumpen (2007)
- Talks by Matteo Frigo and Kaushik Datta at CScADS Autotuning Workshop (2007)
- Demaine’s @ MIT: http://courses.csail.mit.edu/6.897/spring03/scribe_notes

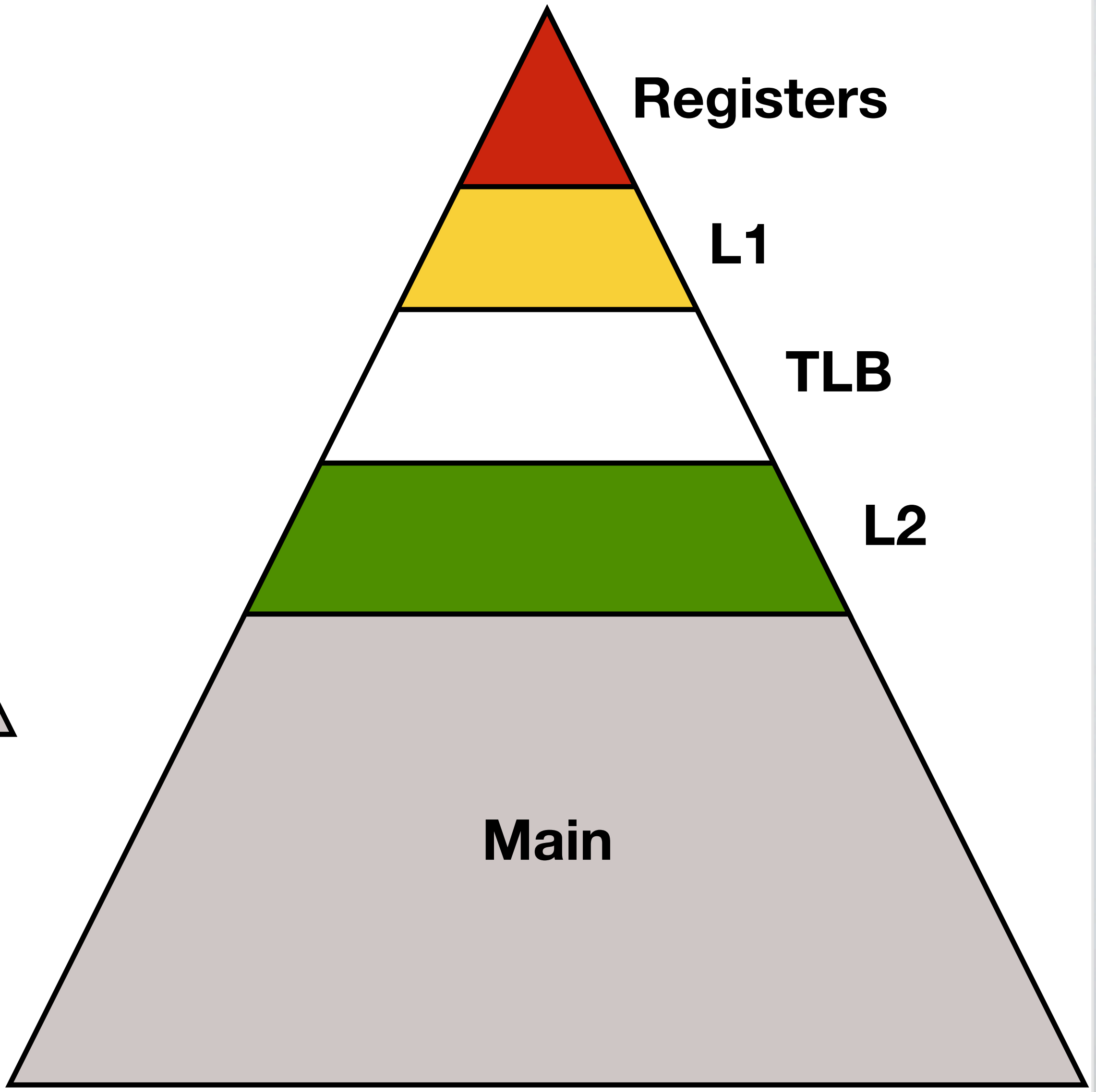
Review:
Tuning matrix multiply

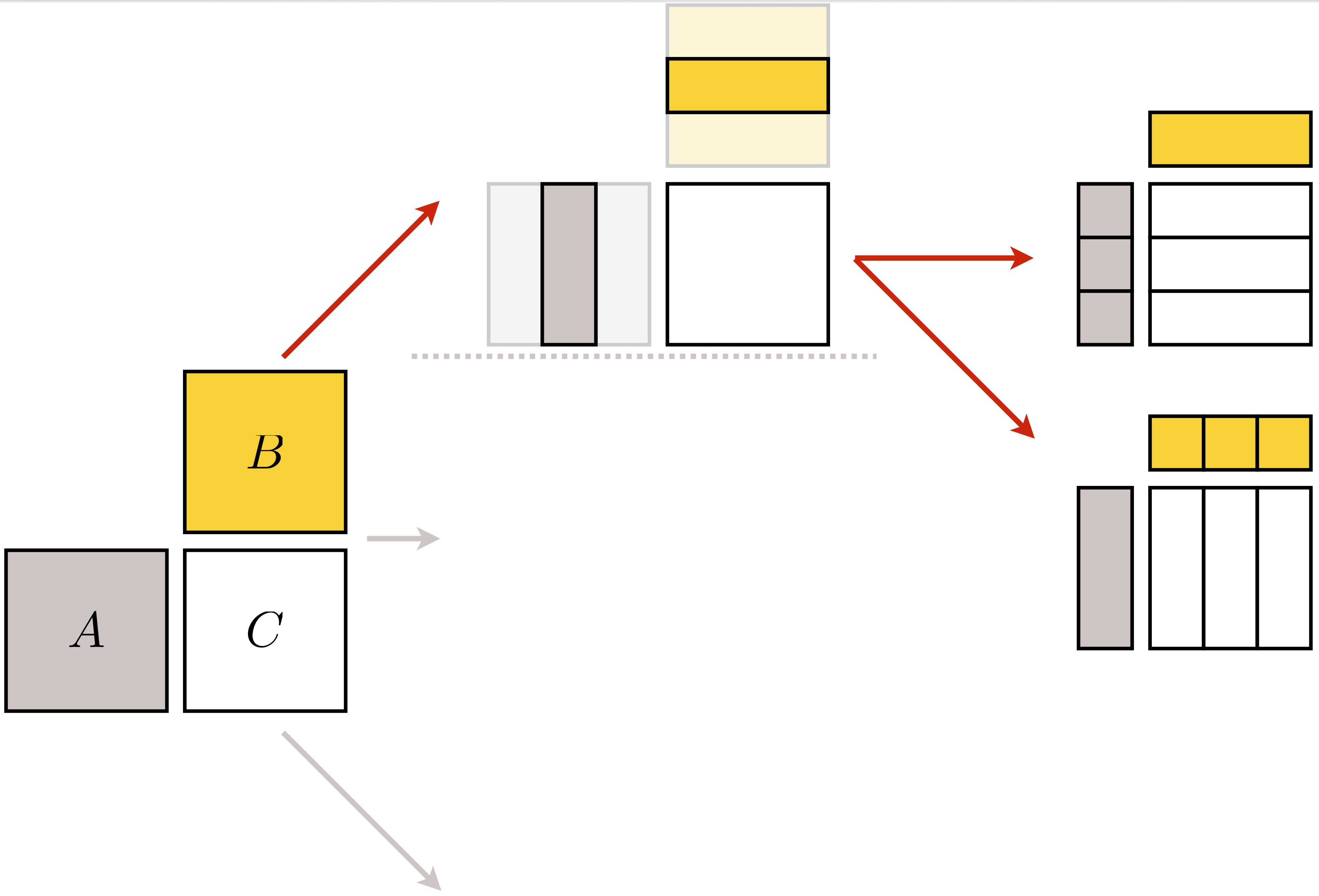
Tiled MM on AMD Opteron 2.2 GHz (4.4 Gflop/s peak), 1 MB L2 cache



< 25% peak! We evidently still have a lot of work to do...

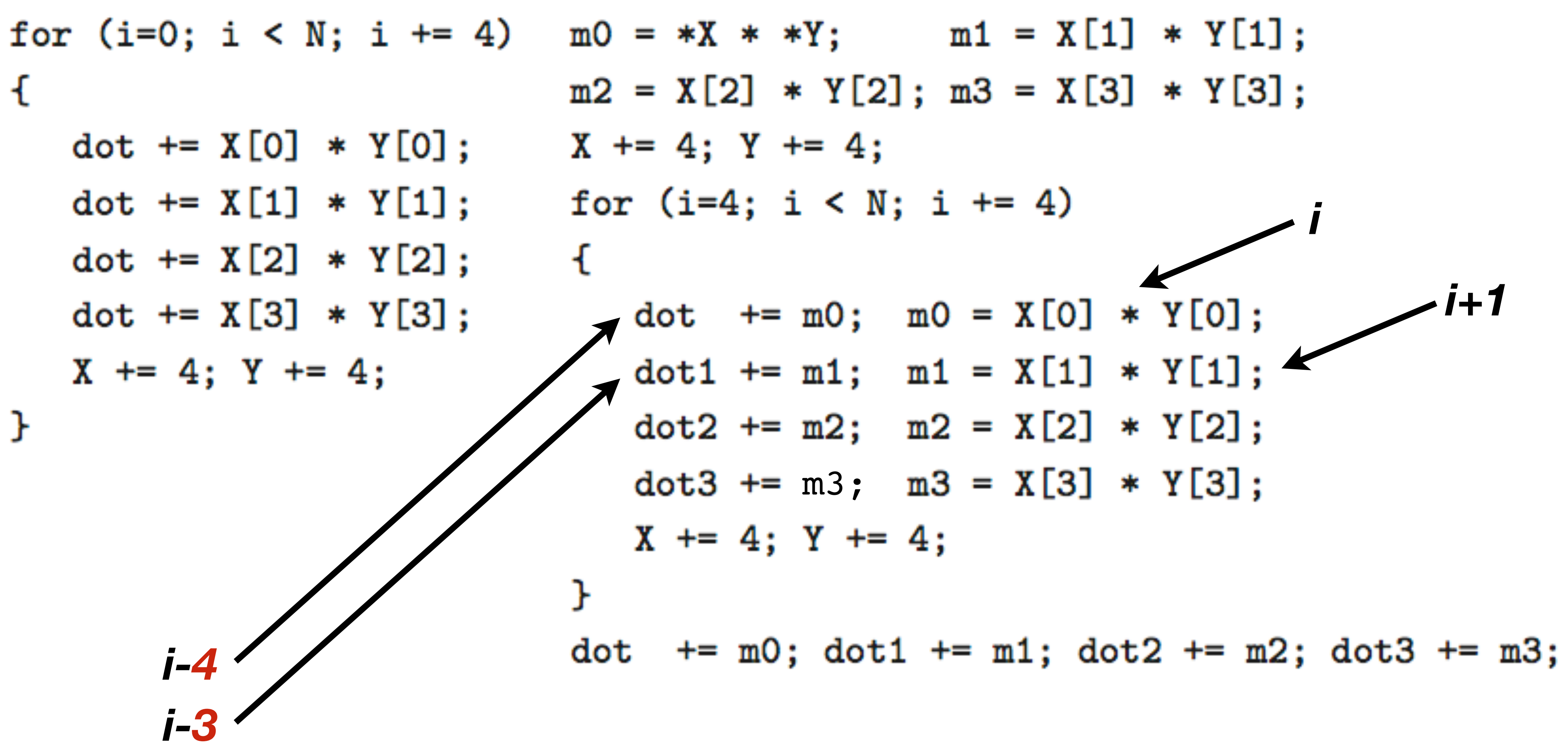

Fast
Slow


Registers
L1
TLB
L2
Main


B
A
C

🟨
⬜
⬜

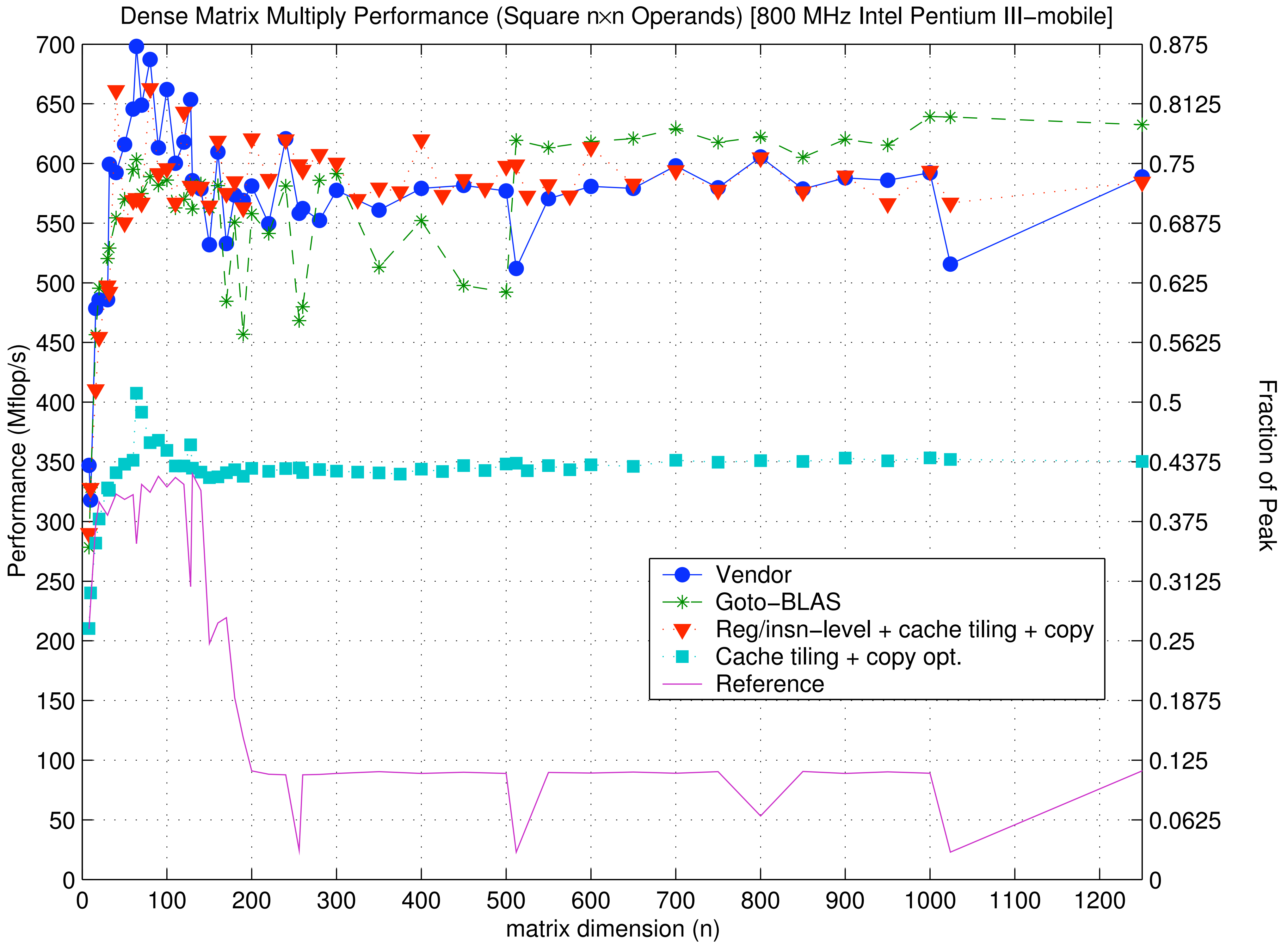
Software pipelining: Interleave iterations to delay dependent instructions



```
for (i=0; i < N; i += 4)
{
   dot += X[0] * Y[0];
   dot += X[1] * Y[1];
   dot += X[2] * Y[2];
   dot += X[3] * Y[3];
   X += 4; Y += 4;
}
```

```
m0 = *X * *Y;      m1 = X[1] * Y[1];
m2 = X[2] * Y[2]; m3 = X[3] * Y[3];
X += 4; Y += 4;
for (i=4; i < N; i += 4)
{
   dot  += m0;  m0 = X[0] * Y[0];
   dot1 += m1;  m1 = X[1] * Y[1];
   dot2 += m2;  m2 = X[2] * Y[2];
   dot3 += m3;  m3 = X[3] * Y[3];
   X += 4; Y += 4;
}
dot  += m0; dot1 += m1; dot2 += m2; dot3 += m3;
```

Source: Clint Whaley's code optimization course (UTSA Spring 2007)



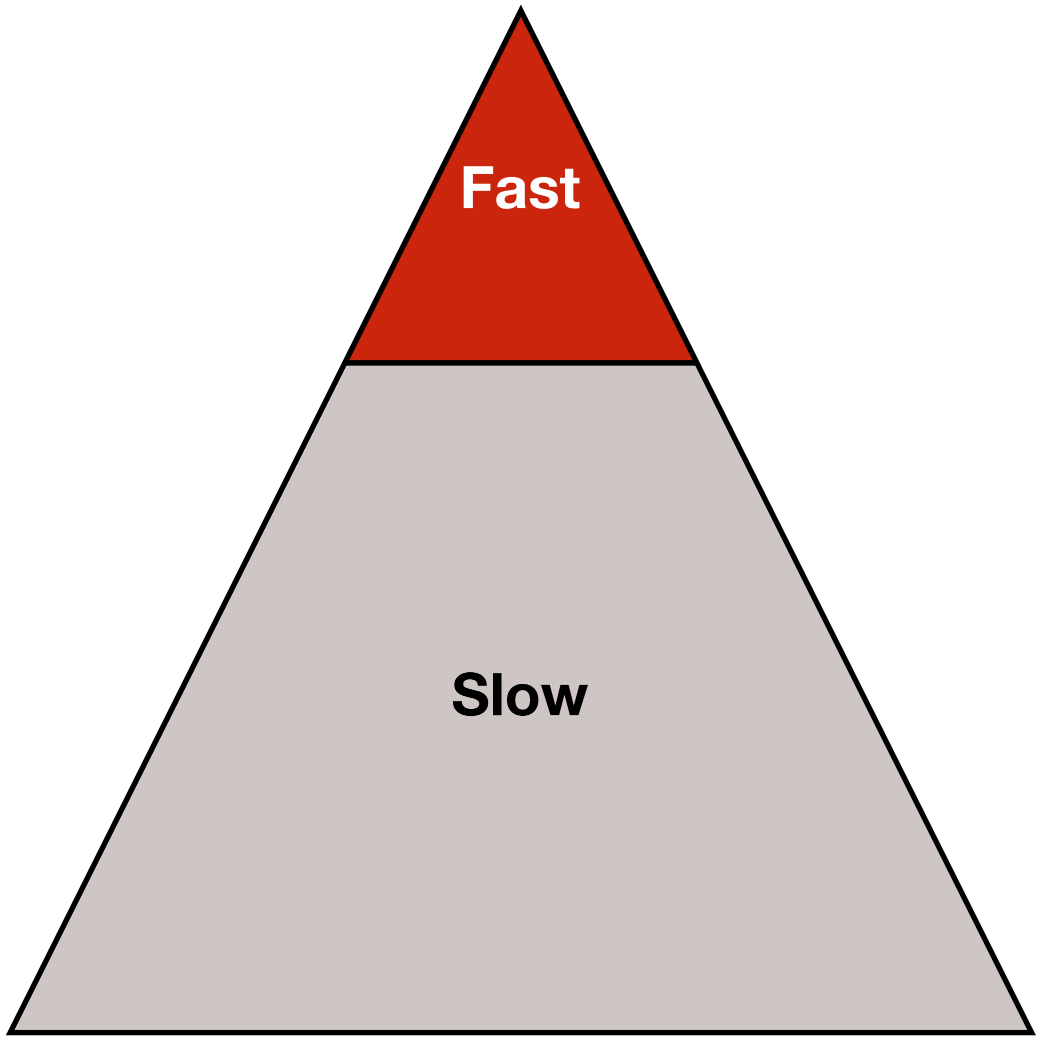
Source: Vuduc, Demmel, Bilmes (IJHPCA 2004)

Cache-oblivious matrix multiply

[Yotov, Roeder, Pingali, Gunnels, Gustavson (SPAA 2007)]
[Talk by M. Frigo at CScADS Autotuning Workshop 2007]

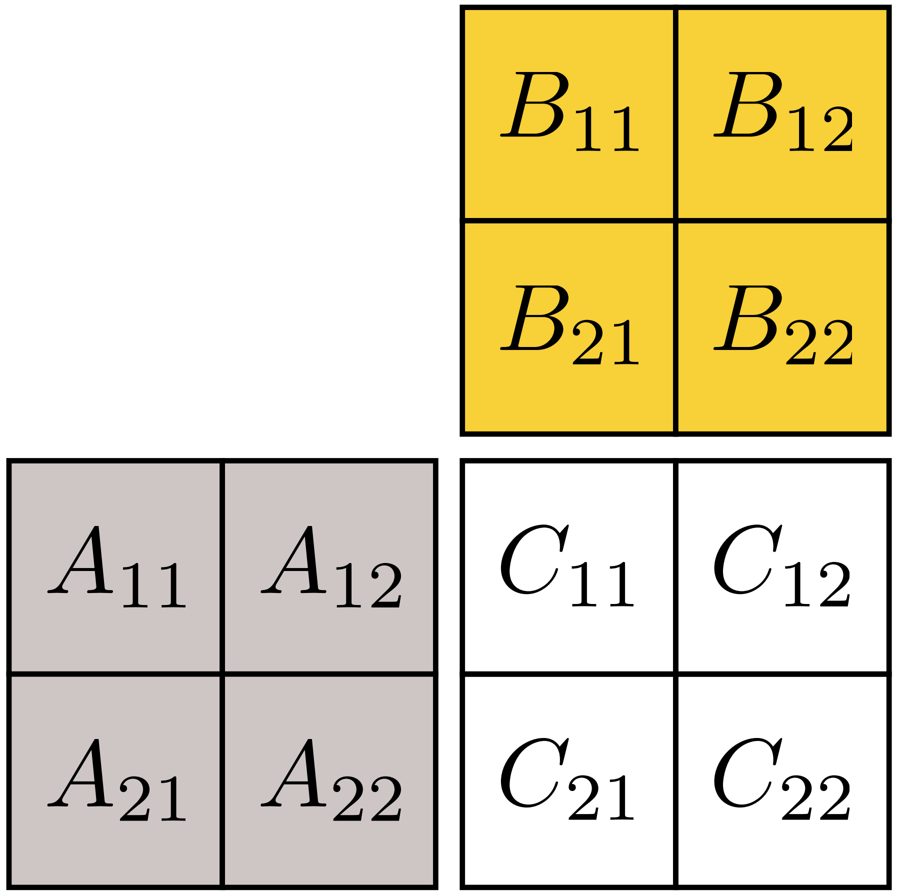
Memory model for analyzing cache-oblivious algorithms

- Two-level memory hierarchy
- M = capacity of cache ("fast")
- L = cache line size
- Fully associative
- Optimal replacement
  - Evicts most distant use
  - Sleator & Tarjan (CACM 1985): LRU, FIFO w/in constant of optimal w/ cache larger by constant factor
- "Tall-cache:" $M \geq O(L^2)$
  - Limits: See Brodal & Fagerberg (STOC 2003)
  - When might this not hold?



A recursive algorithm for matrix-multiply

- Divide all dimensions in half
- Bilardi, *et al*.: Use Gray code ordering

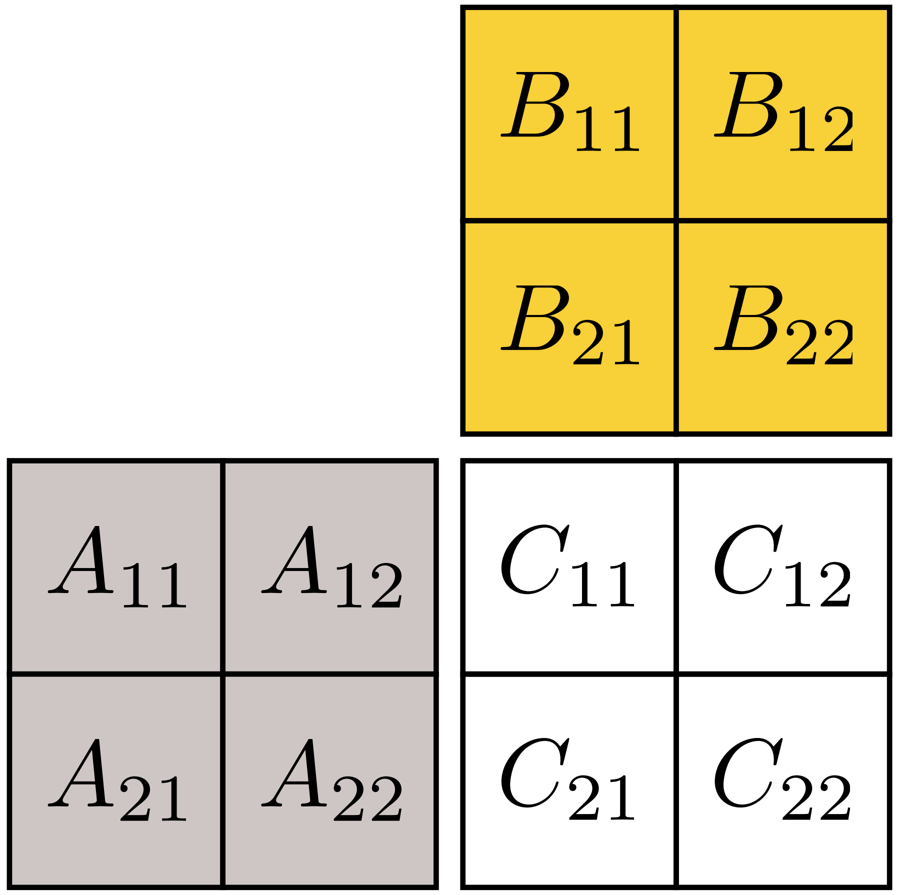


$$\text{Cost (flops)} = T(n) = \begin{cases} 8 \cdot T(\frac{n}{2}) & n > 1 \\ O(1) & n = 1 \end{cases}$$

$$= O(n^3)$$

A recursive algorithm for matrix-multiply

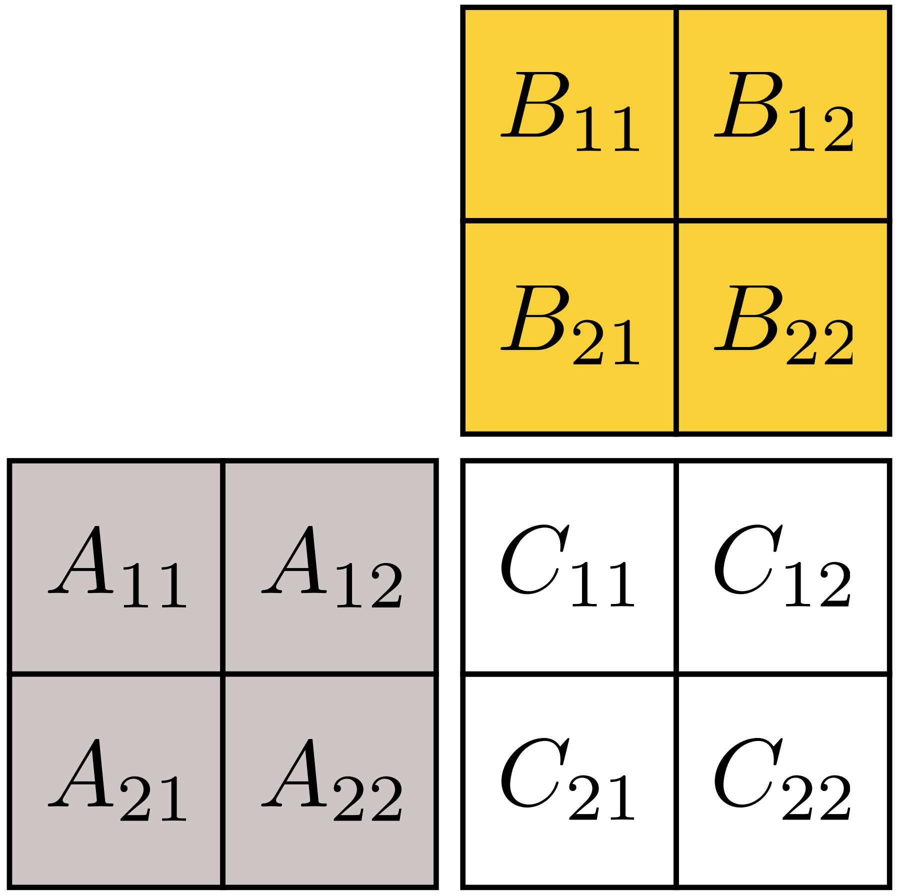
- Divide all dimensions in half
- Bilardi, *et al*.: Use grey-code ordering



I/O Complexity?

A recursive algorithm for matrix-multiply

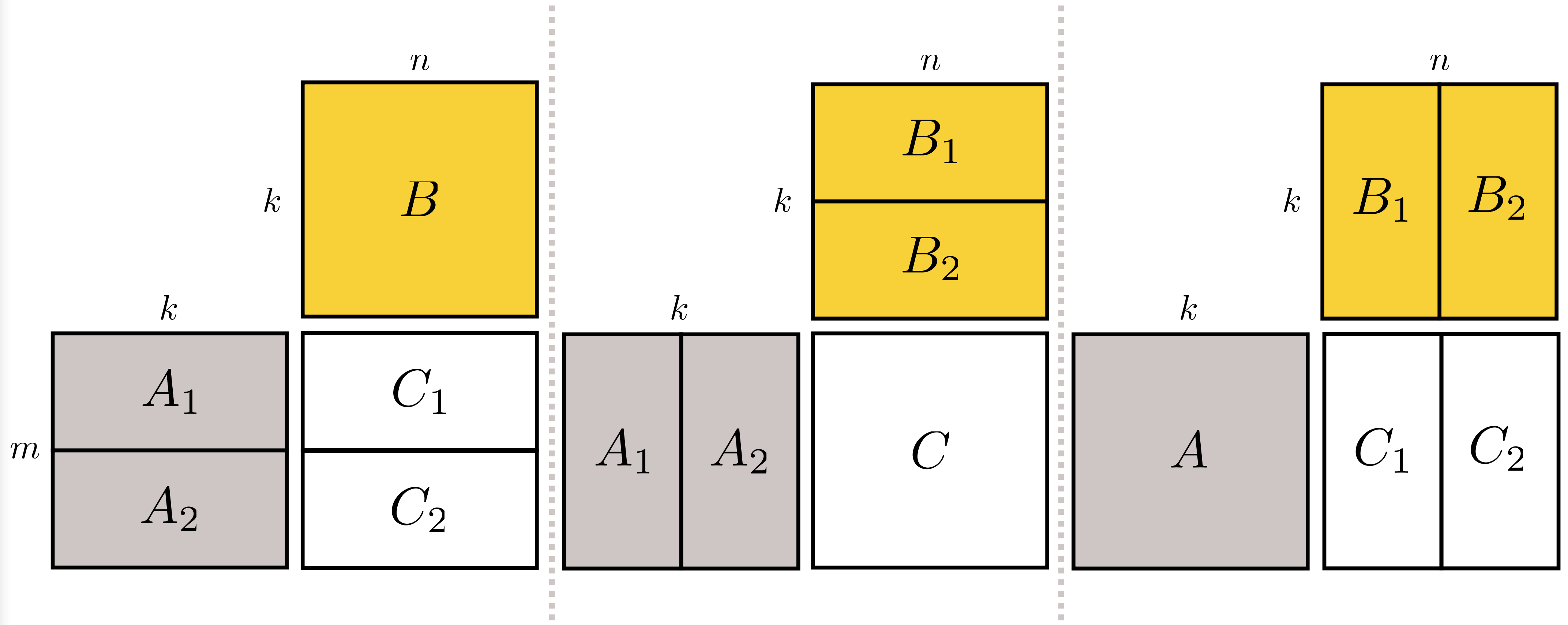
- Divide all dimensions in half
- Bilardi, *et al*.: Use grey-code ordering



No. of misses, with tall-cache assumption:

$$Q(n) = \left\{ \begin{array}{ll} 8 \cdot Q(\frac{n}{2}) & \text{if } n > \sqrt{\frac{M}{3}} \\ \frac{3n^2}{L} & \text{otherwise} \end{array} \right\} \leq \Theta\left(\frac{n^3}{L\sqrt{M}}\right)$$

Alternative: Divide longest dimension (Frigo, *et al.*)

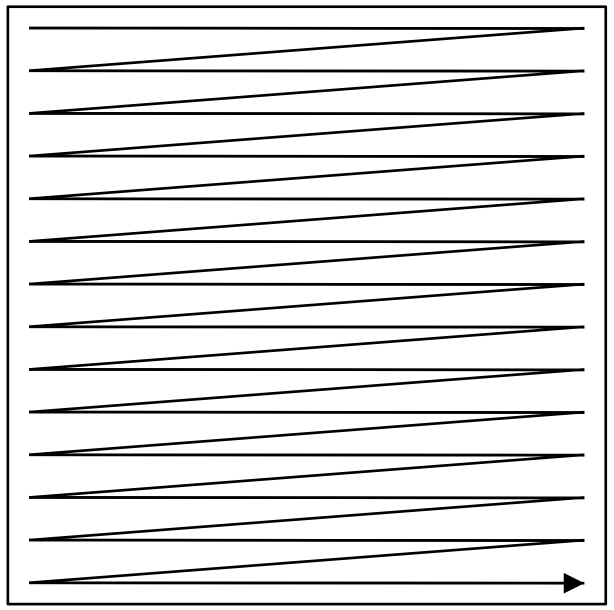


$$\text{Cache misses } Q(m,k,n) \leq \begin{cases} \Theta\left(\frac{mk+kn+mn}{L}\right) & \text{if } mk+kn+mn \leq \alpha M \\ 2Q\left(\frac{m}{2},k,n\right) & \text{if } m \geq k,n \\ 2Q(m,\frac{k}{2},n) & \text{if } k > m, k \geq n \\ 2Q(m,k,\frac{n}{2}) & \text{otherwise} \end{cases}$$

$$= \Theta\left(\frac{mkn}{L\sqrt{M}}\right)$$

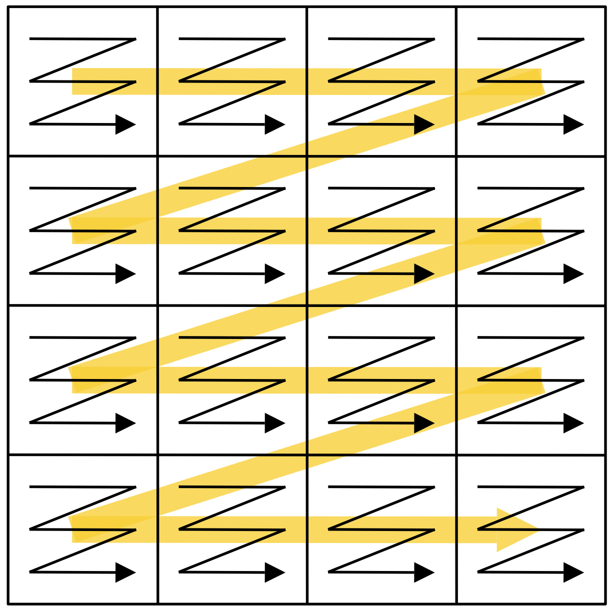
Relax tall-cache assumption using suitable layout

Row-major



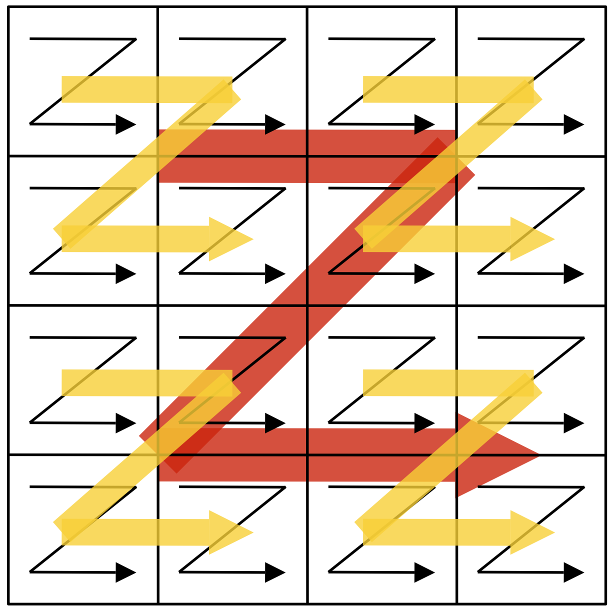
Need tall cache

Row-block-row



$M \geq \Omega(L)$

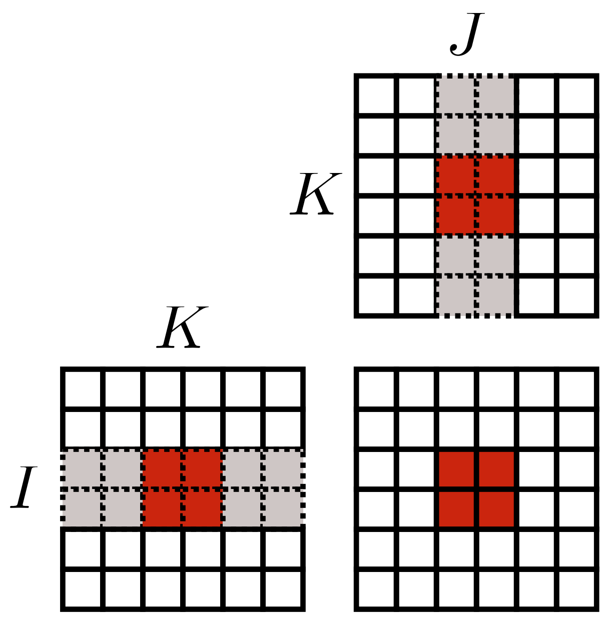
Morton Z



No assumption

Source: Yotov, et al. *(SPAA 2007) and Frigo,* et al. *(FOCS '99)*

Latency-centric vs. bandwidth-centric views of blocking



$$\text{Time per flop} \approx 1 + \frac{\alpha}{\tau} \cdot \frac{1}{b}$$

$$\frac{\alpha}{\tau} \cdot \frac{1}{\kappa} \leq b \leq \sqrt{\frac{M}{3}}$$

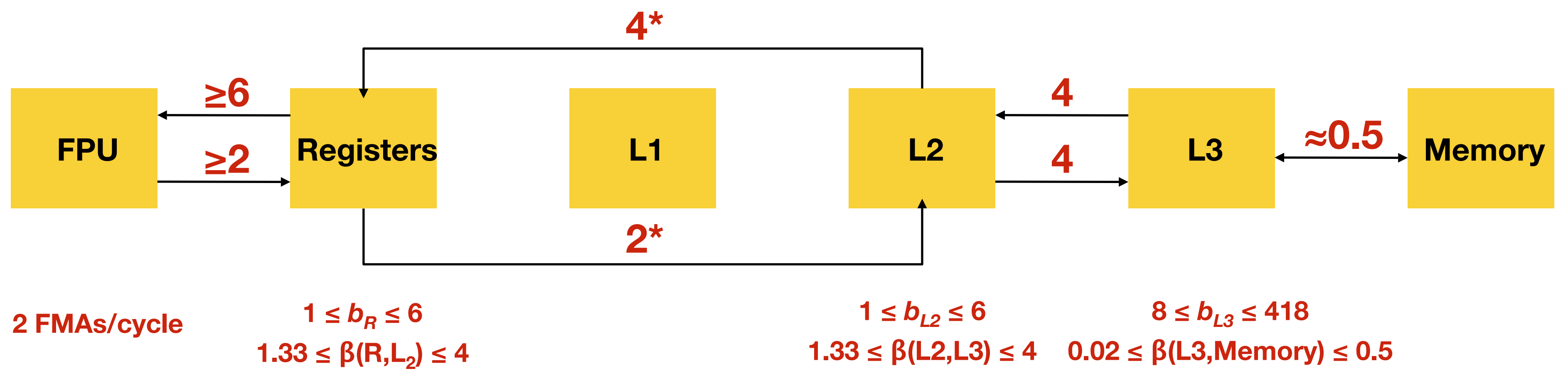
$$\text{Peak flop/cy} \equiv \phi$$

$$\text{Bandwidth, word/cy} \equiv \beta$$

$$\frac{2n^3}{b} \cdot \frac{1}{\beta} \lesssim 2n^3 \cdot \frac{1}{\phi} \implies \frac{\phi}{\beta} \leq b$$

⇐ Assume can perfectly overlap computation & communication

Latency-centric vs. bandwidth-centric views of blocking



- Example platform: Itanium 2
- Consider L3 ←→ memory bandwidth
  - Φ = 4 flops / cycle; β = 0.5 words / cycle
  - L3 capacity = 4 MB (512 kwords)
  - Need 8 ≤ b_{L3} ≤ 418
- Implications: Approximate cache-oblivious blocking works
  - Wide range of block sizes should be OK
  - If upper bound > 2*lower, divide-and-conquer generates block size in range

Source: Yotov, et al. *(SPAA 2007)*

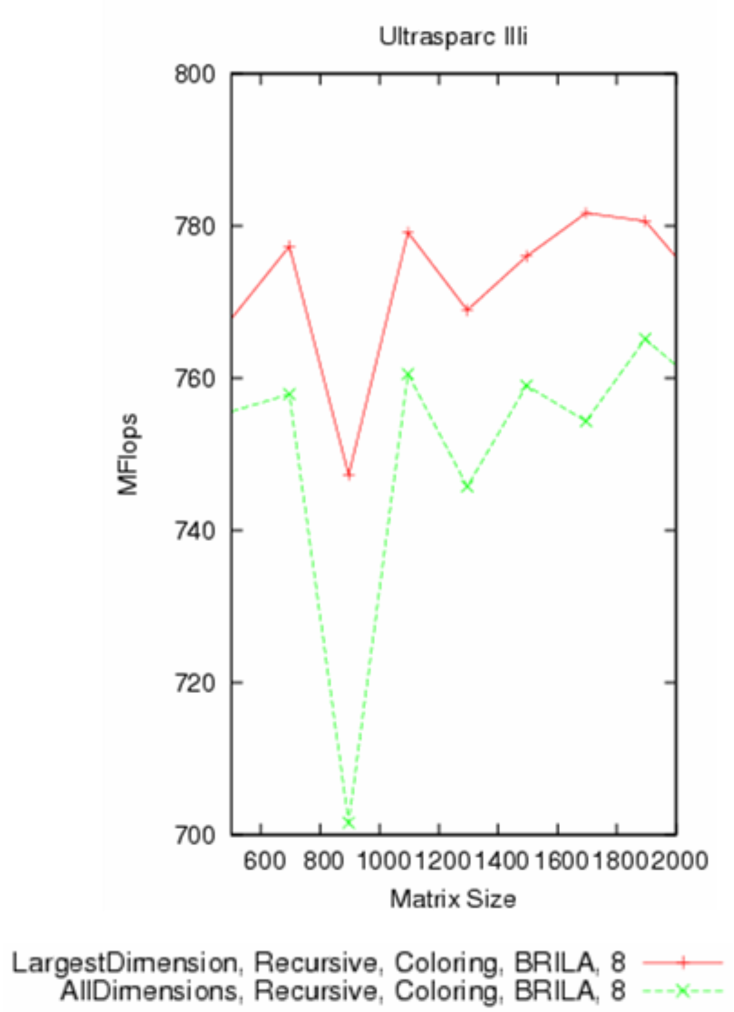
Cache-oblivious vs. cache-aware

- Does cache-oblivious perform as well as cache-aware?
- If not, what can be done?

- Next: Summary of Yotov, *et al*., study (SPAA 2007)
    - Stole slides liberally

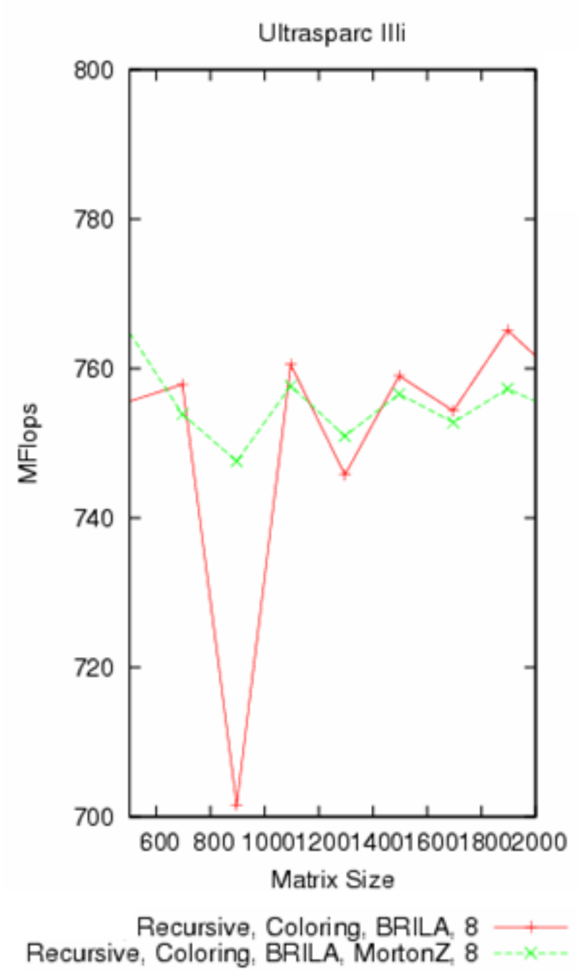
All- vs. largest-dimension

- Similar; assume “all-dim”



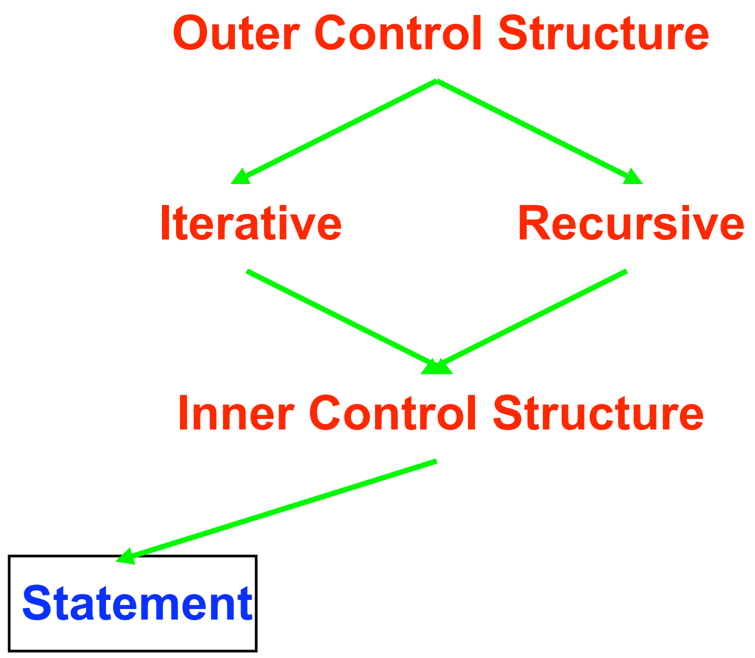
Data structures

- Morton-Z complicated and yields same or worse performance, so assume row-block-row

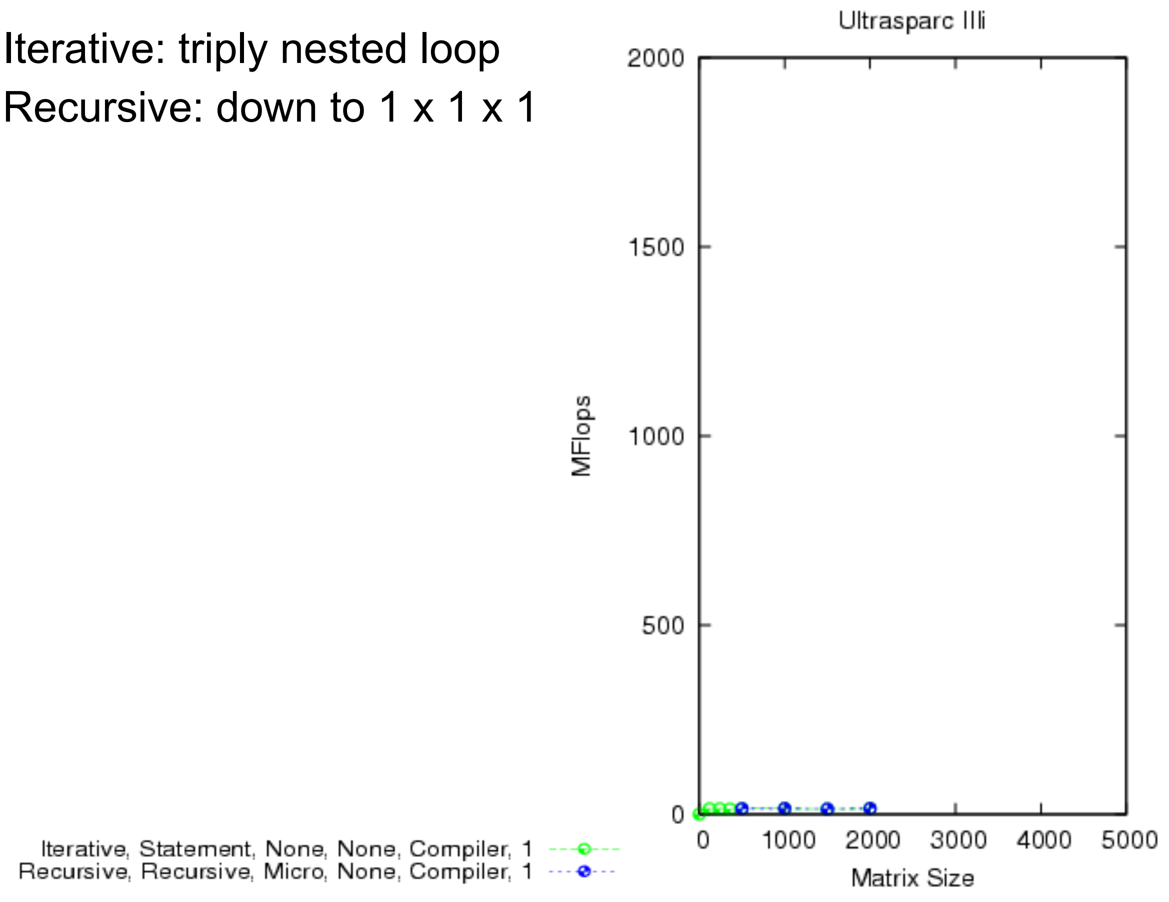


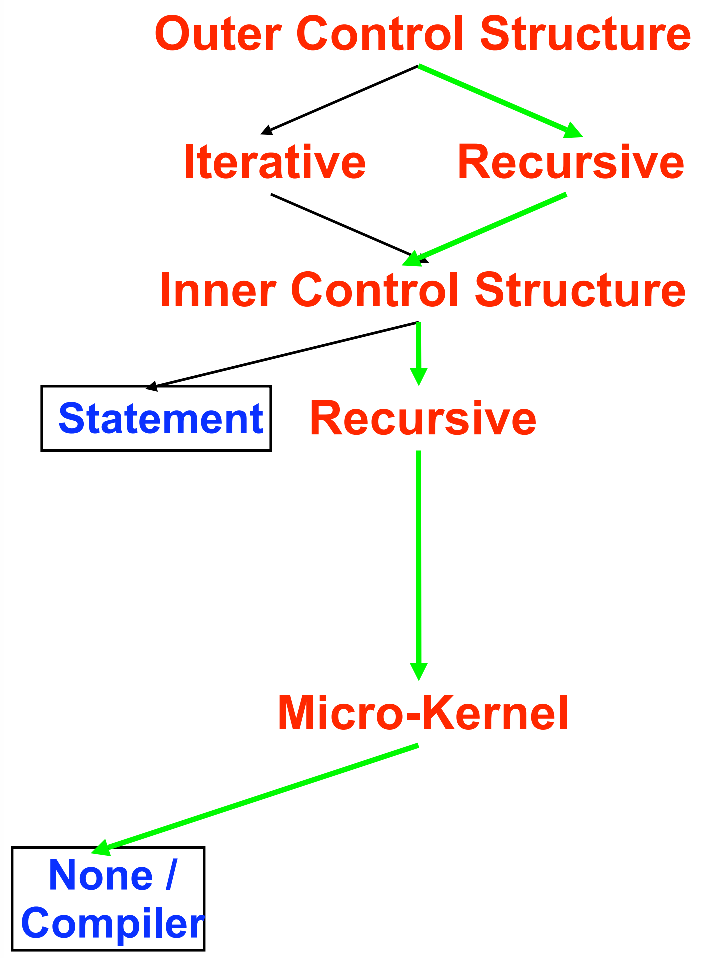
Example 1: Ultra IIIi

- 1 GHz ⇒ 2 Gflop/s peak
- Memory hierarchy
  - 32 registers
  - L1 = 64 KB, 4-way
  - L2 = 1 MB, 4-way
- Sun compiler

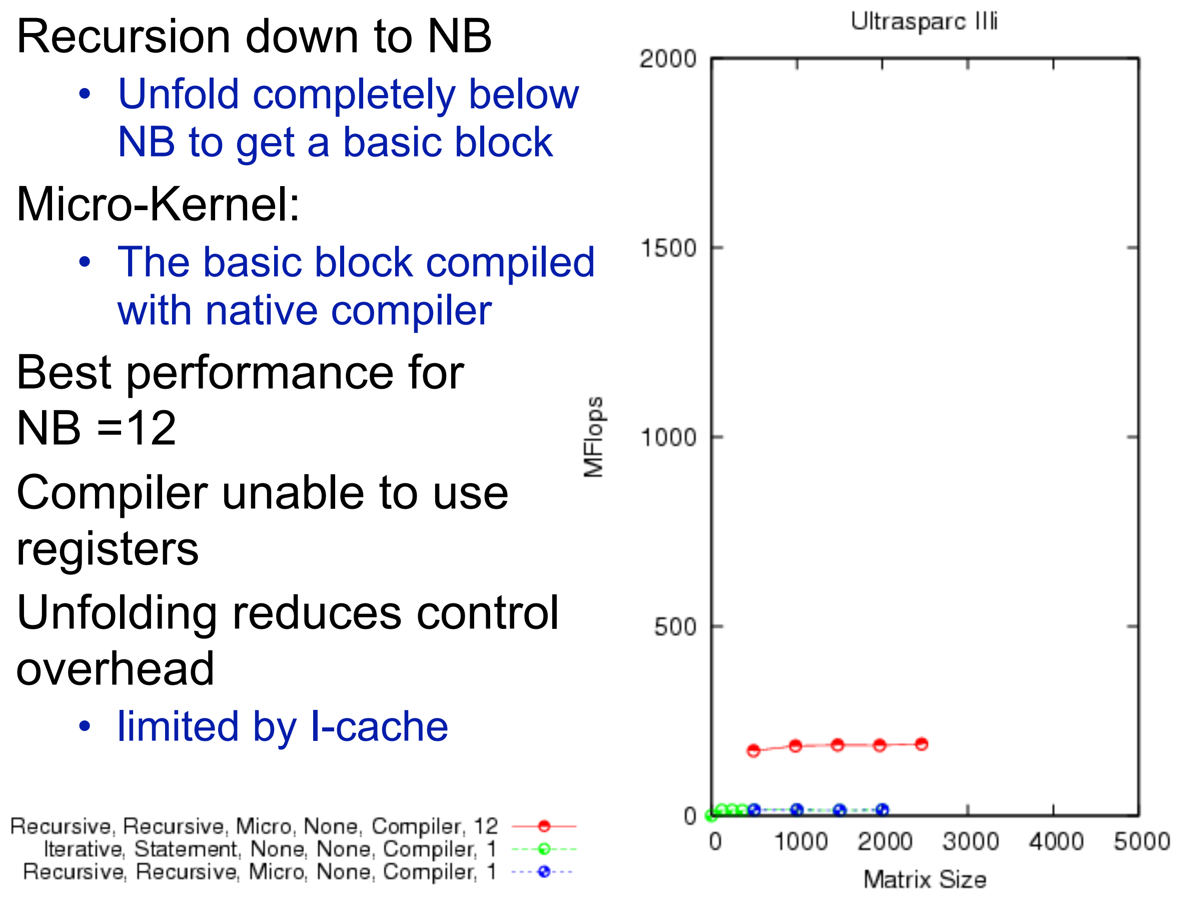


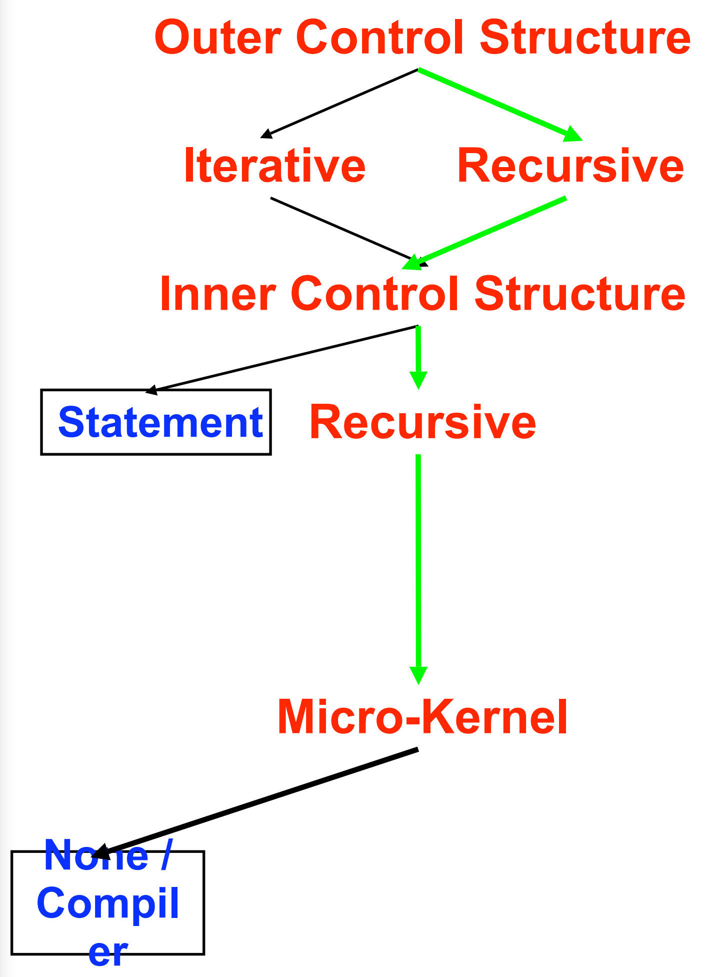
- Iterative: triply nested loop
- Recursive: down to 1 x 1 x 1



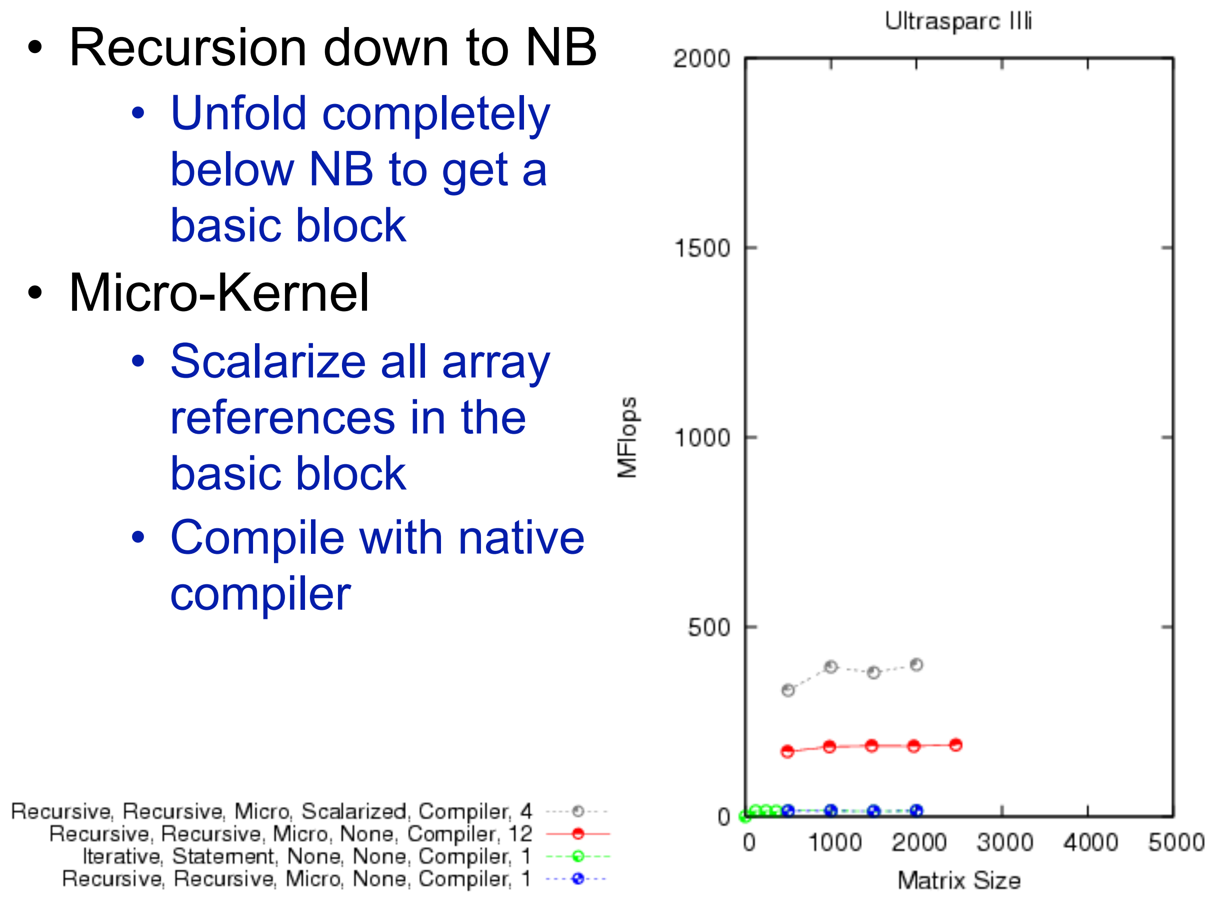


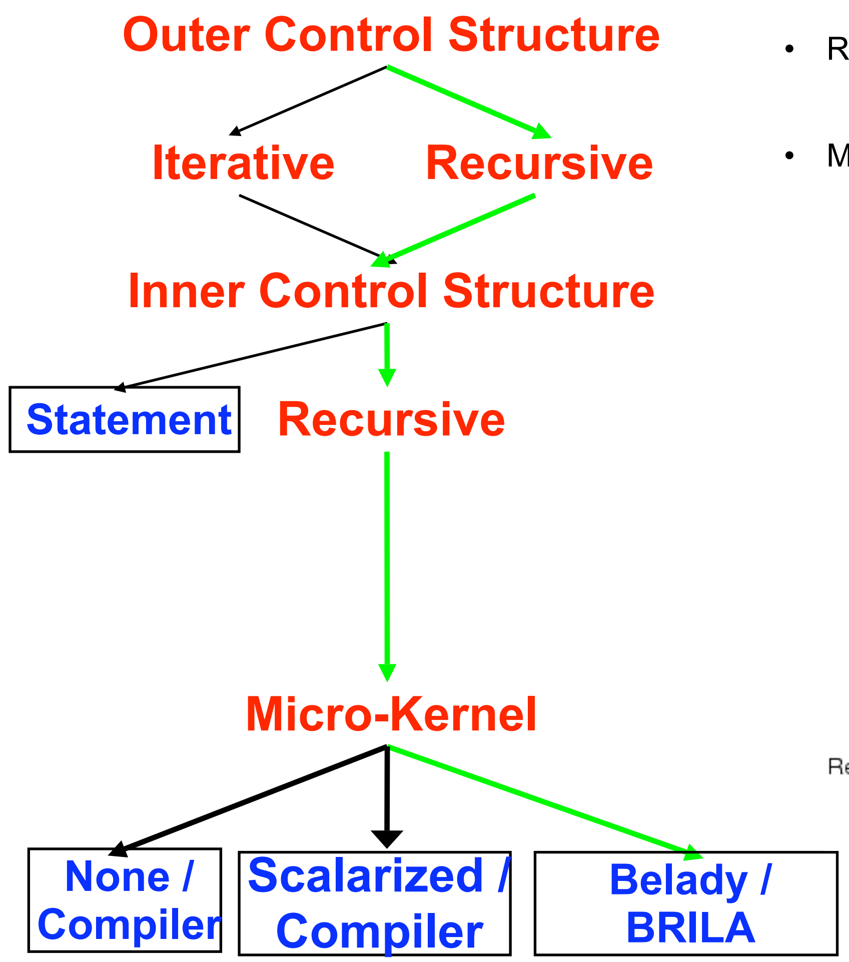
- Recursion down to NB
  - Unfold completely below NB to get a basic block
- Micro-Kernel:
  - The basic block compiled with native compiler
- Best performance for NB =12
- Compiler unable to use registers
- Unfolding reduces control overhead
  - limited by I-cache



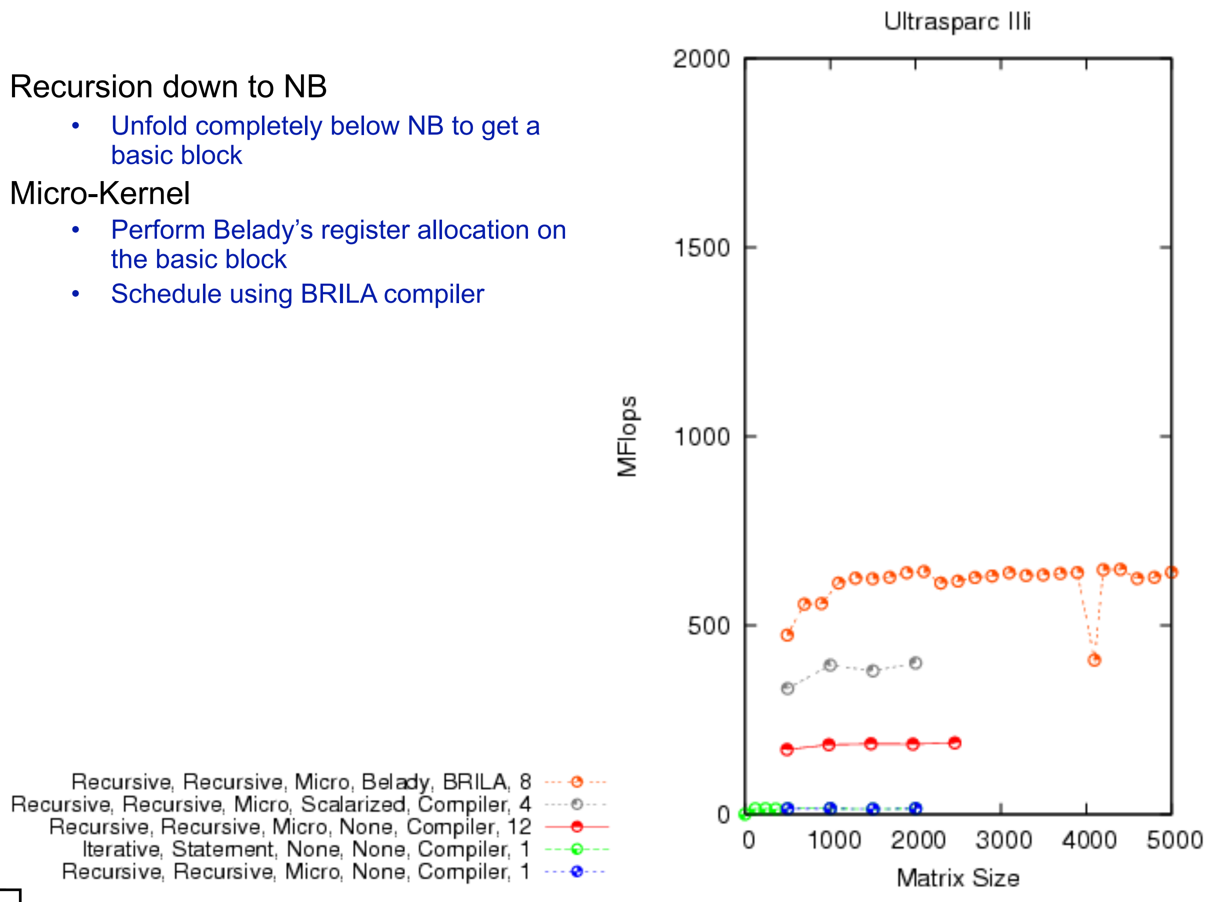


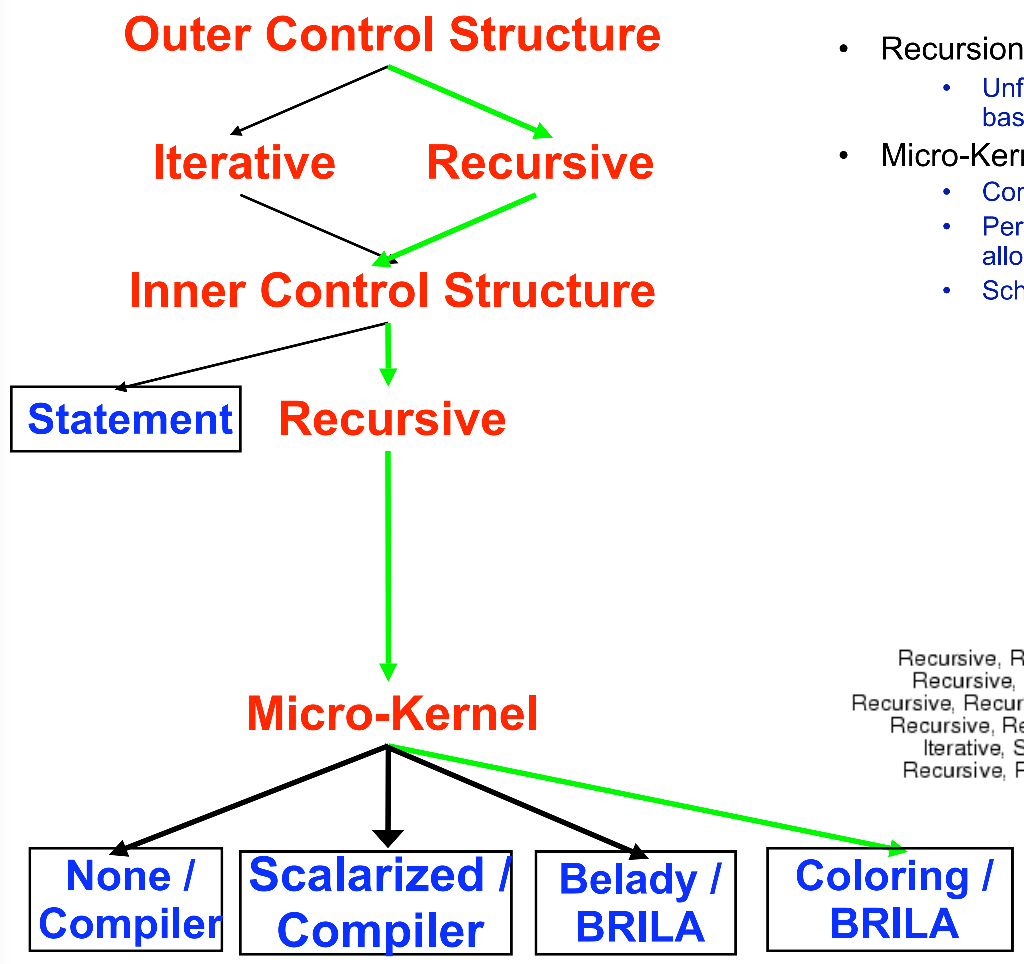
- Recursion down to NB
  - Unfold completely below NB to get a basic block
- Micro-Kernel
  - Scalarize all array references in the basic block
  - Compile with native compiler



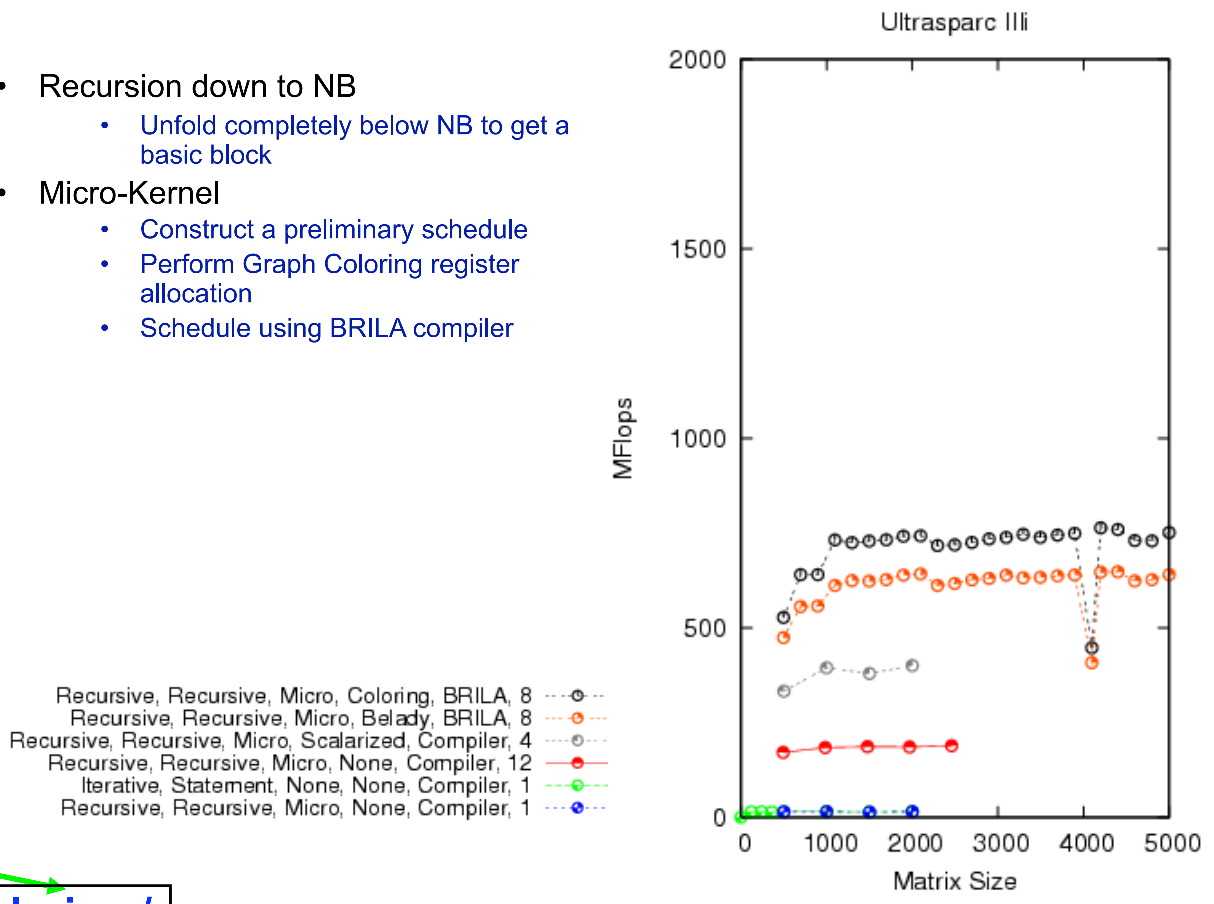


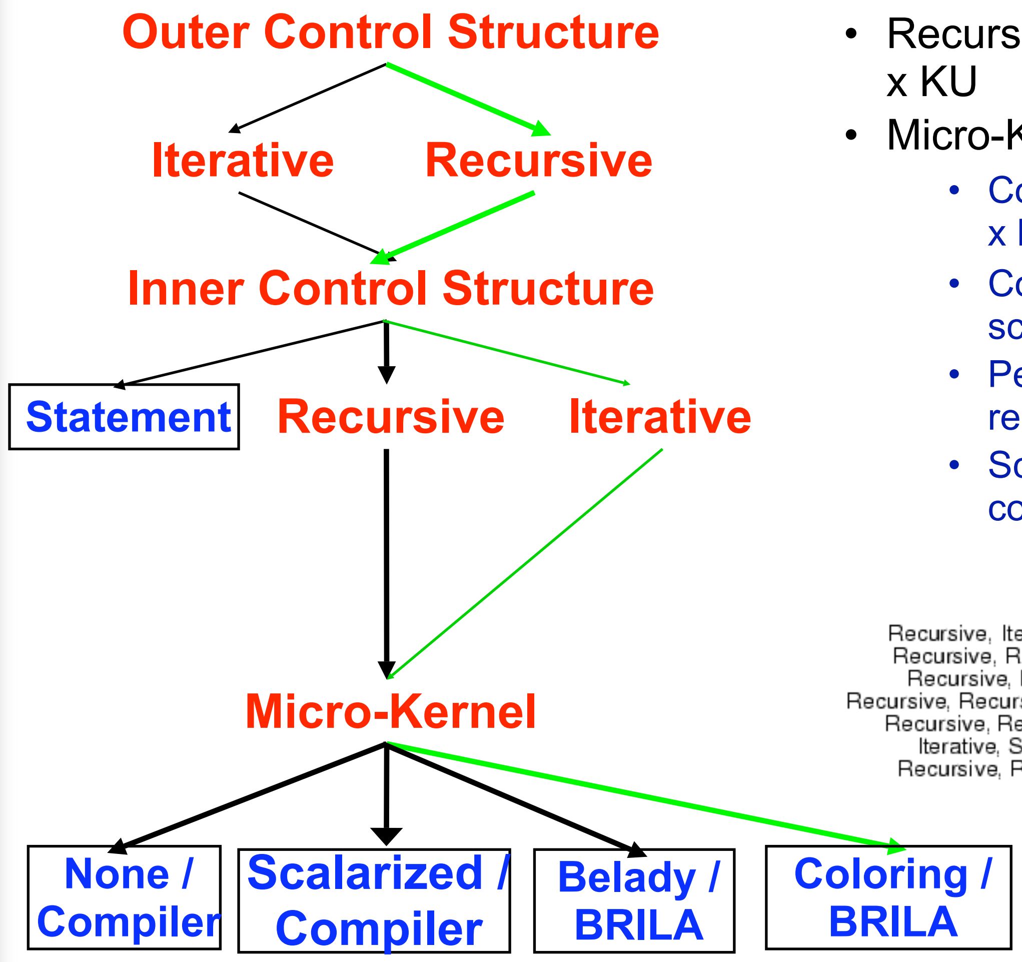
- Recursion down to NB
  - Unfold completely below NB to get a basic block
- Micro-Kernel
  - Perform Belady's register allocation on the basic block
  - Schedule using BRILA compiler



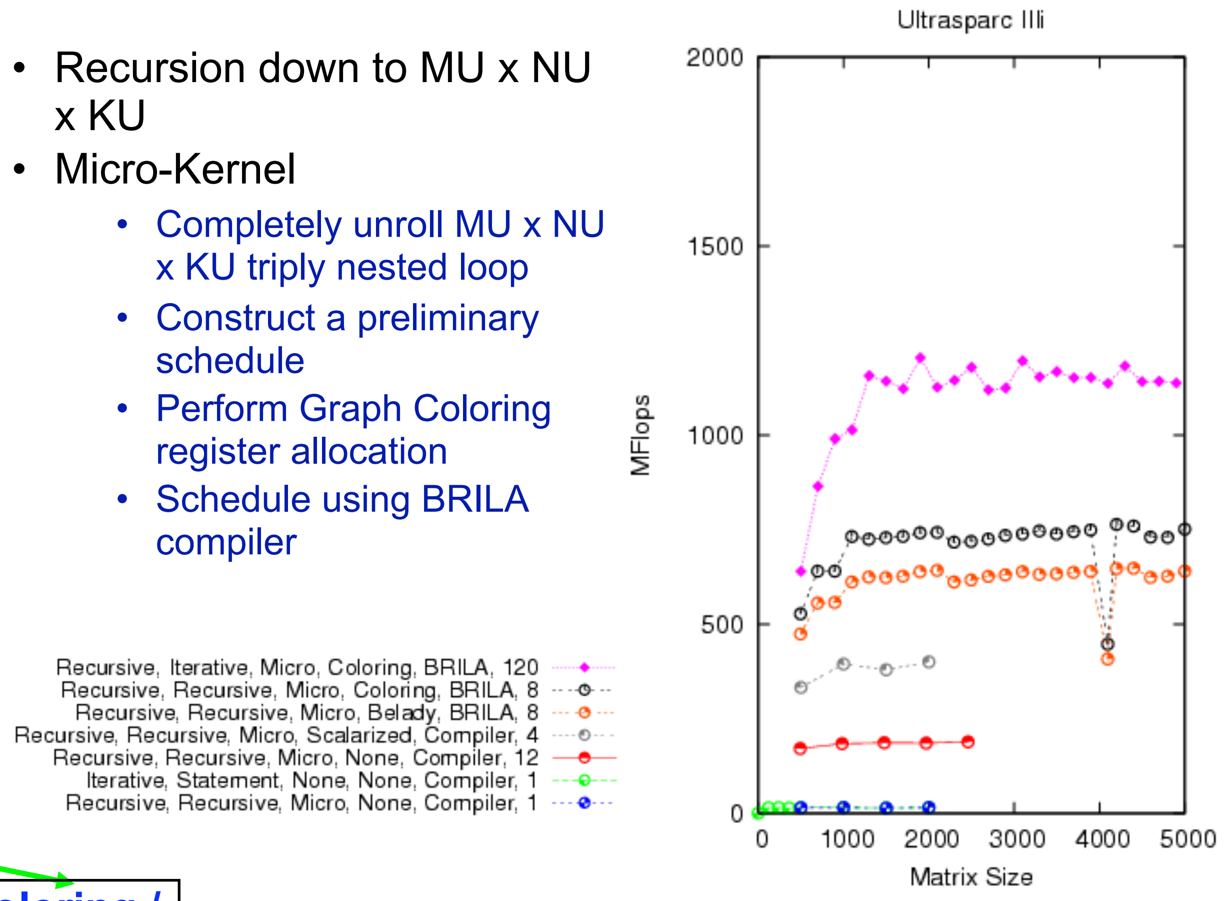


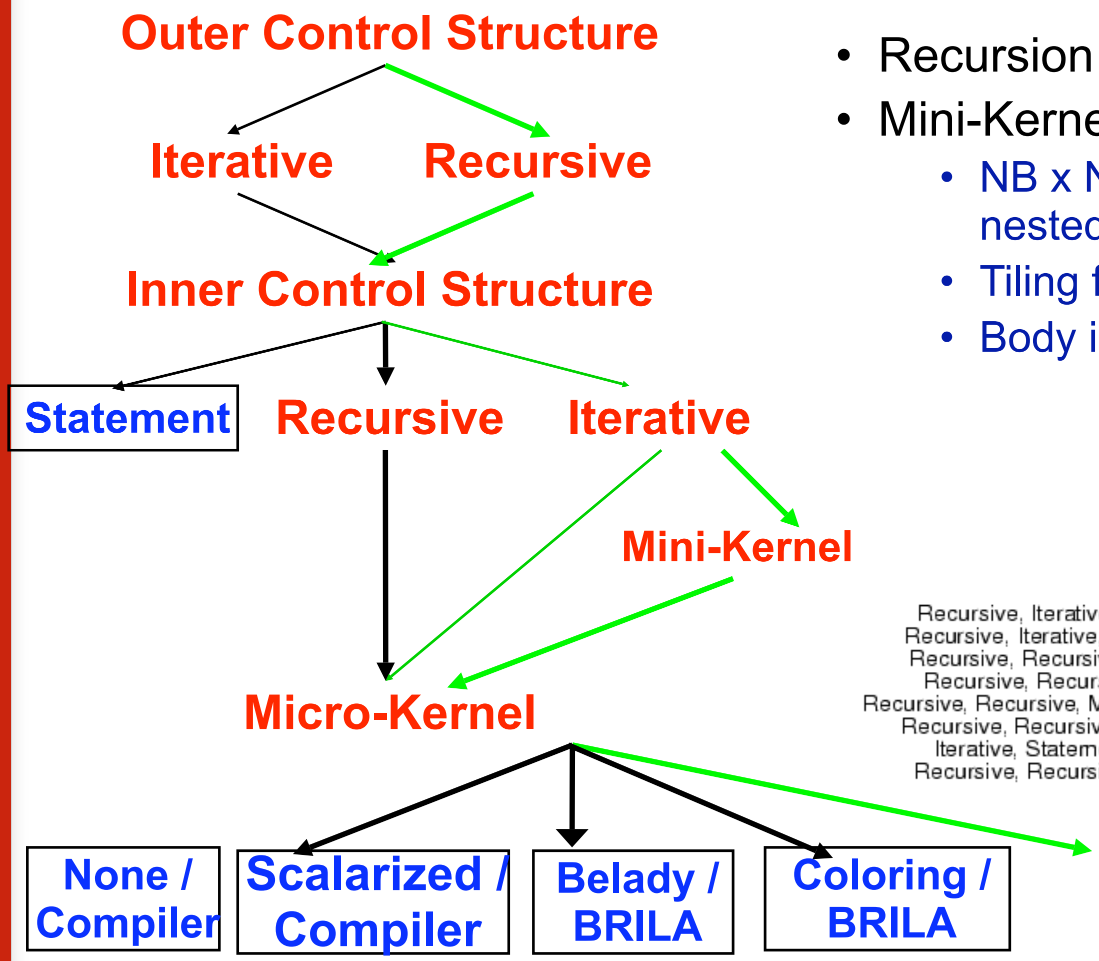
- Recursion down to NB
  - Unfold completely below NB to get a basic block
- Micro-Kernel
  - Construct a preliminary schedule
  - Perform Graph Coloring register allocation
  - Schedule using BRILA compiler



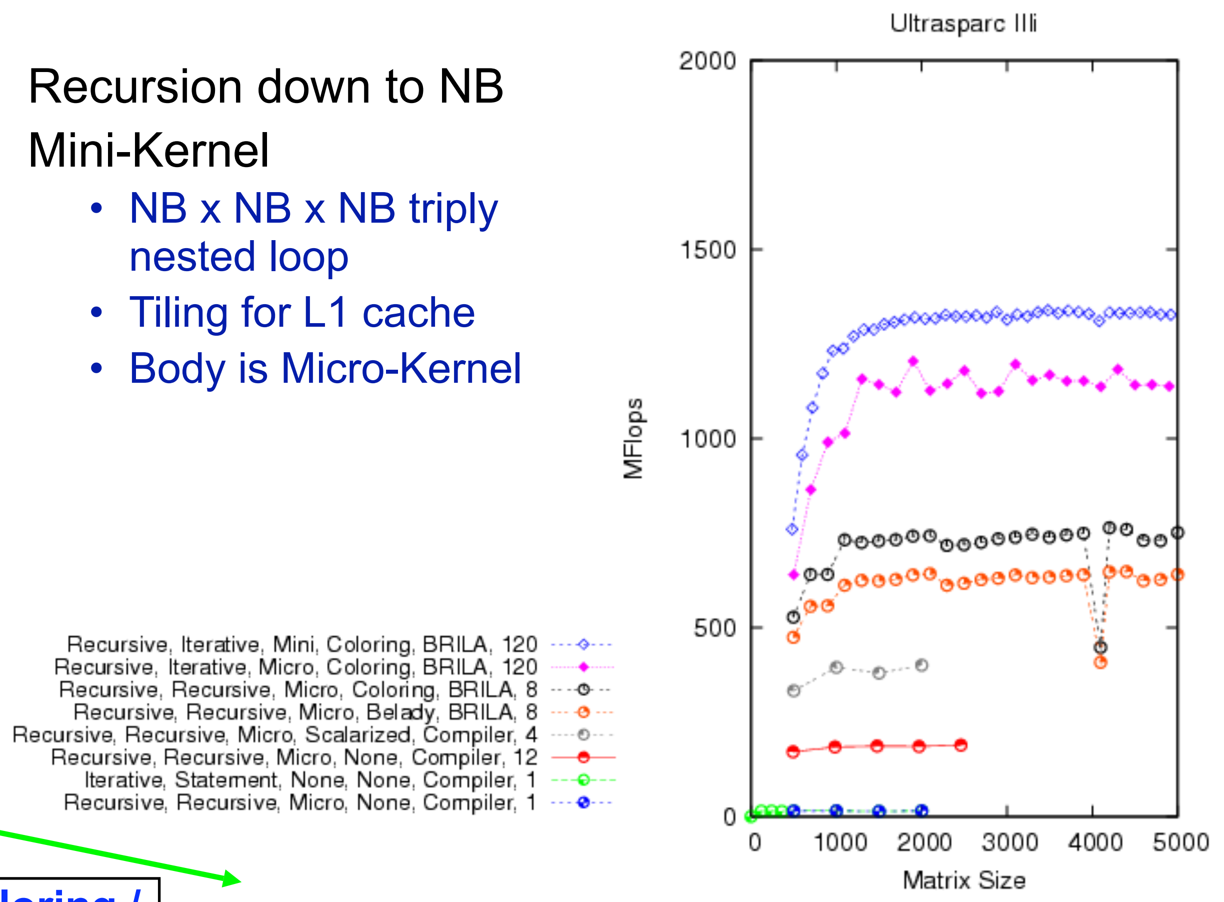


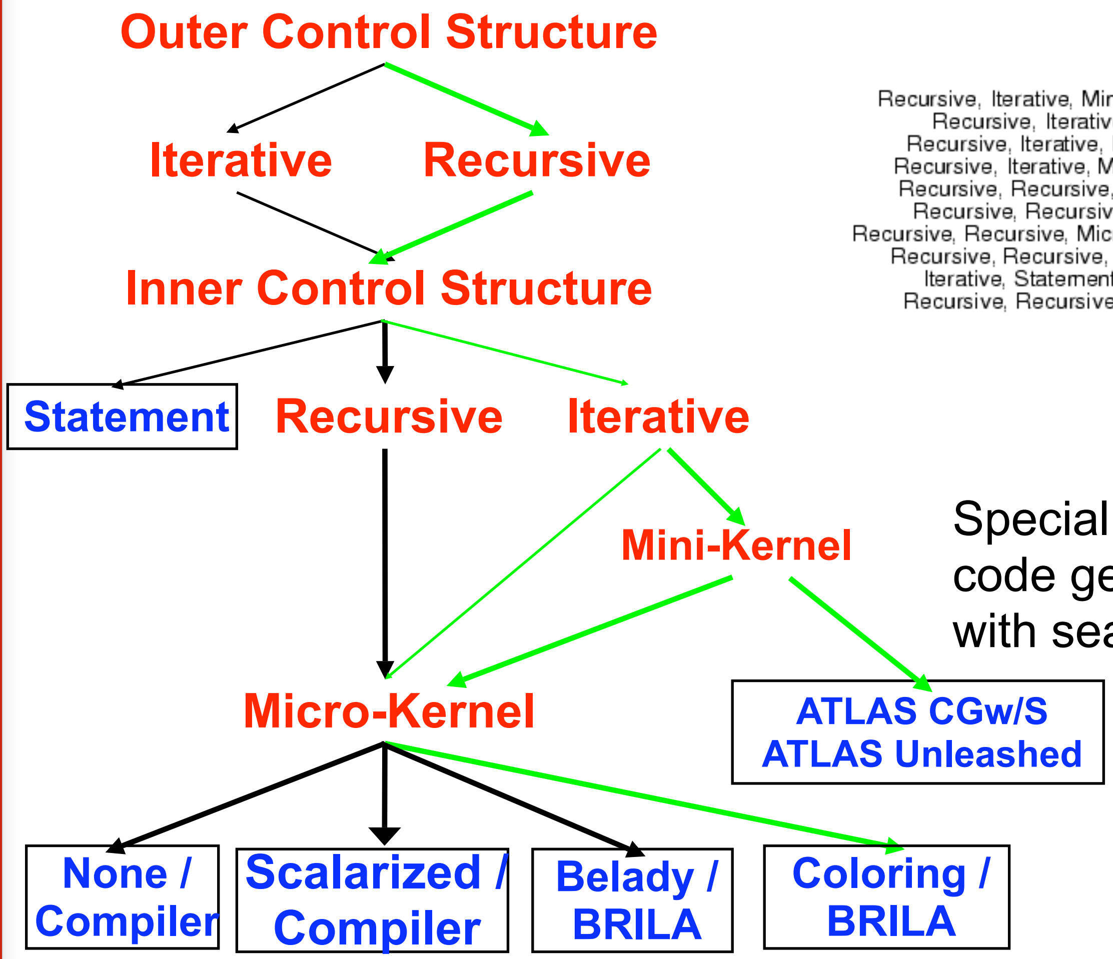
- Recursion down to MU x NU x KU
- Micro-Kernel
  - Completely unroll MU x NU x KU triply nested loop
  - Construct a preliminary schedule
  - Perform Graph Coloring register allocation
  - Schedule using BRILA compiler



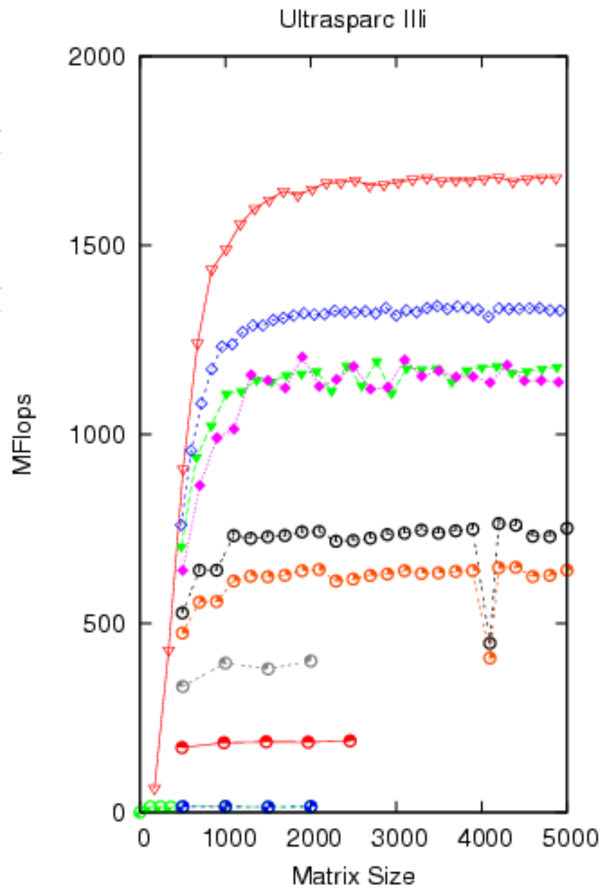


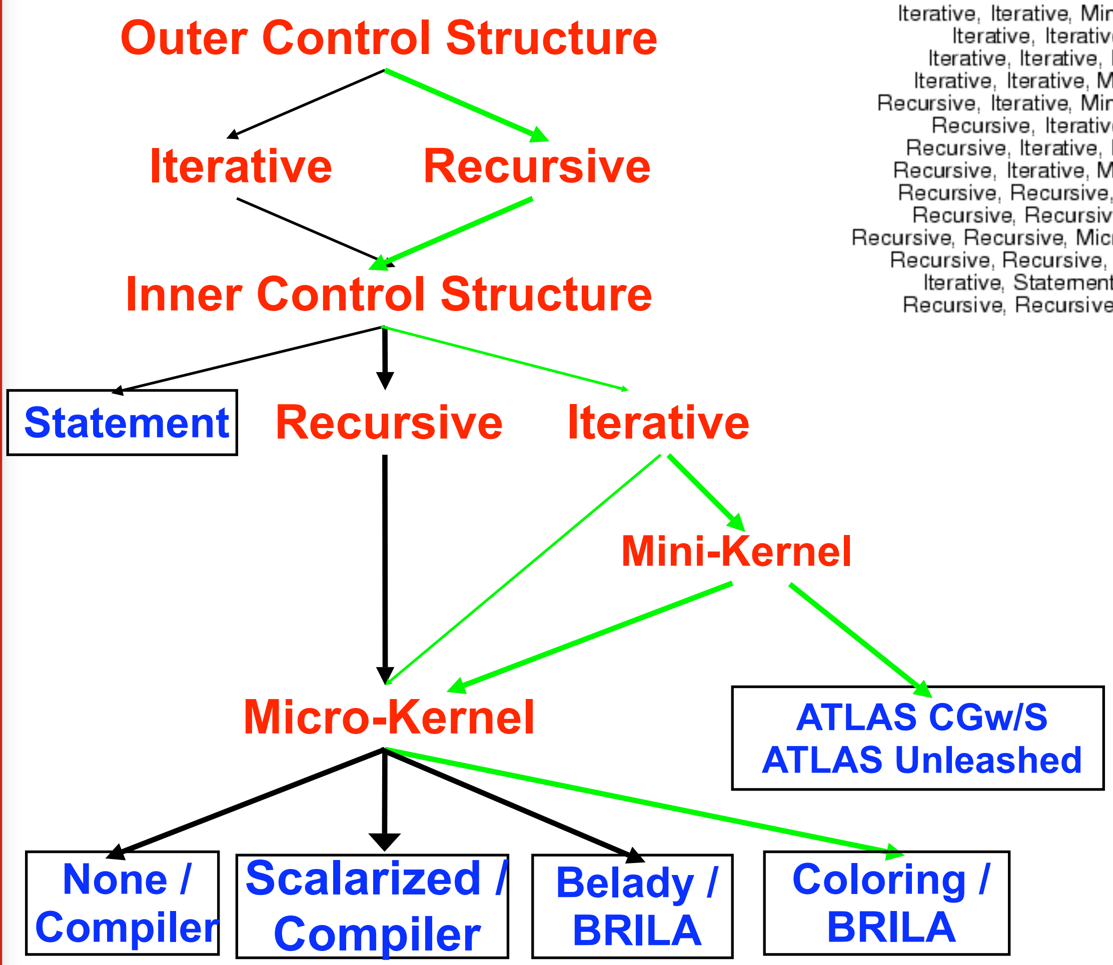
- Recursion down to NB
- Mini-Kernel
  - NB x NB x NB triply nested loop
  - Tiling for L1 cache
  - Body is Micro-Kernel

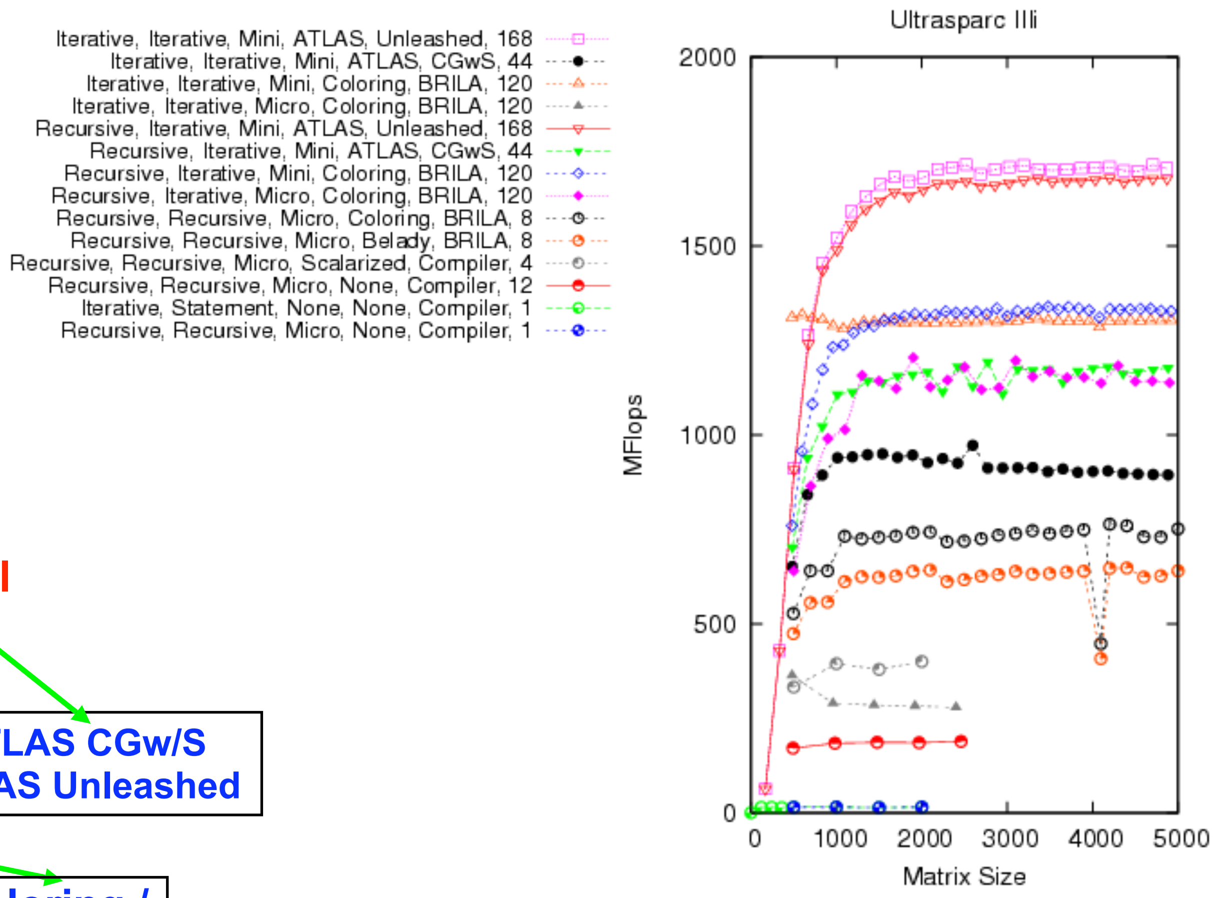




Specialized code generator with search




Outer Control Structure
Iterative
Recursive
Inner Control Structure
Statement
Recursive
Iterative
Mini-Kernel
Micro-Kernel
ATLAS CGw/S
ATLAS Unleashed
None / Compiler
Scalarized / Compiler
Belady / BRILA
Coloring / BRILA


Ultrasparc IIIi
Iterative, Iterative, Mini, ATLAS, Unleashed, 168
Iterative, Iterative, Mini, ATLAS, CGwS, 44
Iterative, Iterative, Mini, Coloring, BRILA, 120
Iterative, Iterative, Micro, Coloring, BRILA, 120
Recursive, Iterative, Mini, ATLAS, Unleashed, 168
Recursive, Iterative, Mini, ATLAS, CGwS, 44
Recursive, Iterative, Mini, Coloring, BRILA, 120
Recursive, Iterative, Micro, Coloring, BRILA, 120
Recursive, Recursive, Micro, Coloring, BRILA, 8
Recursive, Recursive, Micro, Belady, BRILA, 8
Recursive, Recursive, Micro, Scalarized, Compiler, 4
Recursive, Recursive, Micro, None, Compiler, 12
Iterative, Statement, None, None, Compiler, 1
Recursive, Recursive, Micro, None, Compiler, 1
MFlops
0
500
1000
1500
2000
Matrix Size
0
1000
2000
3000
4000
5000

Summary: Engineering considerations

- Need to cut-off recursion
- Careful scheduling/tuning required at “leaves”
- Yotov, et al., report that full-recursion + tuned micro-kernel ≤ 2/3 best
- Open issues
  - Recursively-scheduled kernels worse than iteratively-schedule kernels — why?
  - Prefetching needed, but how best to apply in recursive case?

Administrivia

Upcoming schedule changes

- Some adjustment of topics (TBD)
- Tu 3/11 — Project proposals due
- Th 3/13 — SIAM Parallel Processing (attendance encouraged)
- Tu 4/1 — No class
- Th 4/3 — Attend talk by Doug Post from DoD HPC Modernization Program

Homework 1: Parallel conjugate gradients

- Put name on write-up!
- Grading: 100 pts max
  - Correct implementation — 50 pts
  - Evaluation — 30 pts
    - Tested on two samples matrices — 5
    - Implemented and tested on stencil — 10
    - “Explained” performance (e.g., per proc, load balance, comp. vs. comm) — 15
  - Performance model — 15 pts
  - Write-up “quality” — 5 pts

Projects

- **Proposals due Tu 3/11**
- Your goal should be to do something useful, interesting, and/or publishable!
  - Something you're already working on, suitably adapted for this course
  - Faculty-sponsored/mentored
  - Collaborations encouraged

My criteria for “approving” your project

- “Relevant to this course:” Many themes, so think (and “do”) broadly
    - Parallelism and architectures
    - Numerical algorithms
    - Programming models
    - Performance modeling/analysis

General styles of projects

- Theoretical: Prove something hard (high risk)
- Experimental:
  - Parallelize something
  - Take existing parallel program, and improve it using models & experiments
  - Evaluate algorithm, architecture, or programming model

Examples

- *Anything of interest to a faculty member/project outside CoC*
- Parallel sparse triple product ($R*A*R^T$, used in multigrid)
- Future FFT
- Out-of-core or I/O-intensive data analysis and algorithms
- Block iterative solvers (convergence & performance trade-offs)
- Sparse LU
- Data structures and algorithms (trees, graphs)
- Look at mixed-precision
- Discrete-event approaches to continuous systems simulation
- Automated performance analysis and modeling, tuning
- “Unconventional,” but related
  - Distributed deadlock detection for MPI
  - UPC language extensions (dynamic block sizes)
  - Exact linear algebra

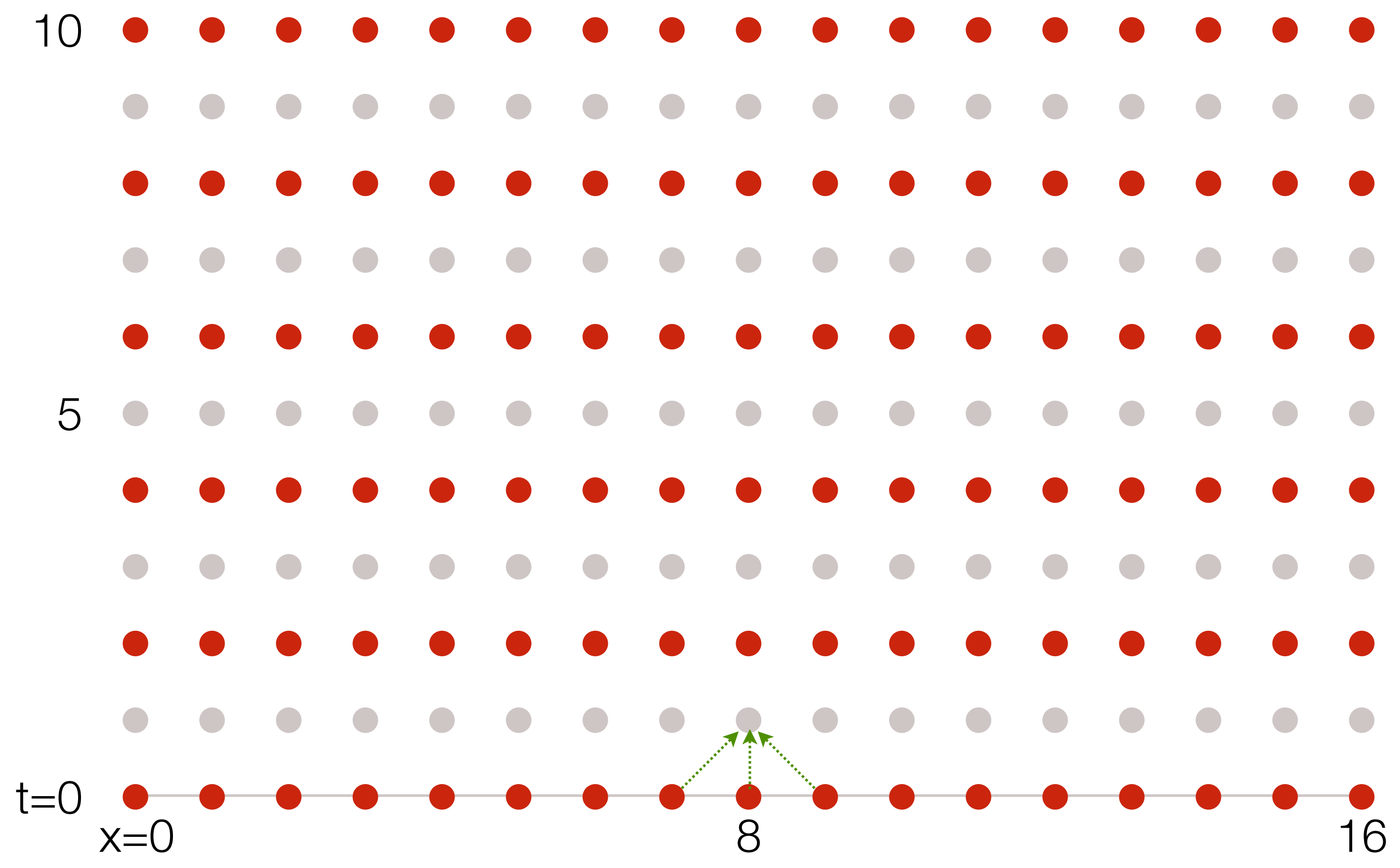
Switch: M. Frigo's talk slides from CScADS 2007 autotuning workshop

http://cscads.rice.edu/workshops/july2007/autotune-workshop-07

Cache-oblivious stencil computations

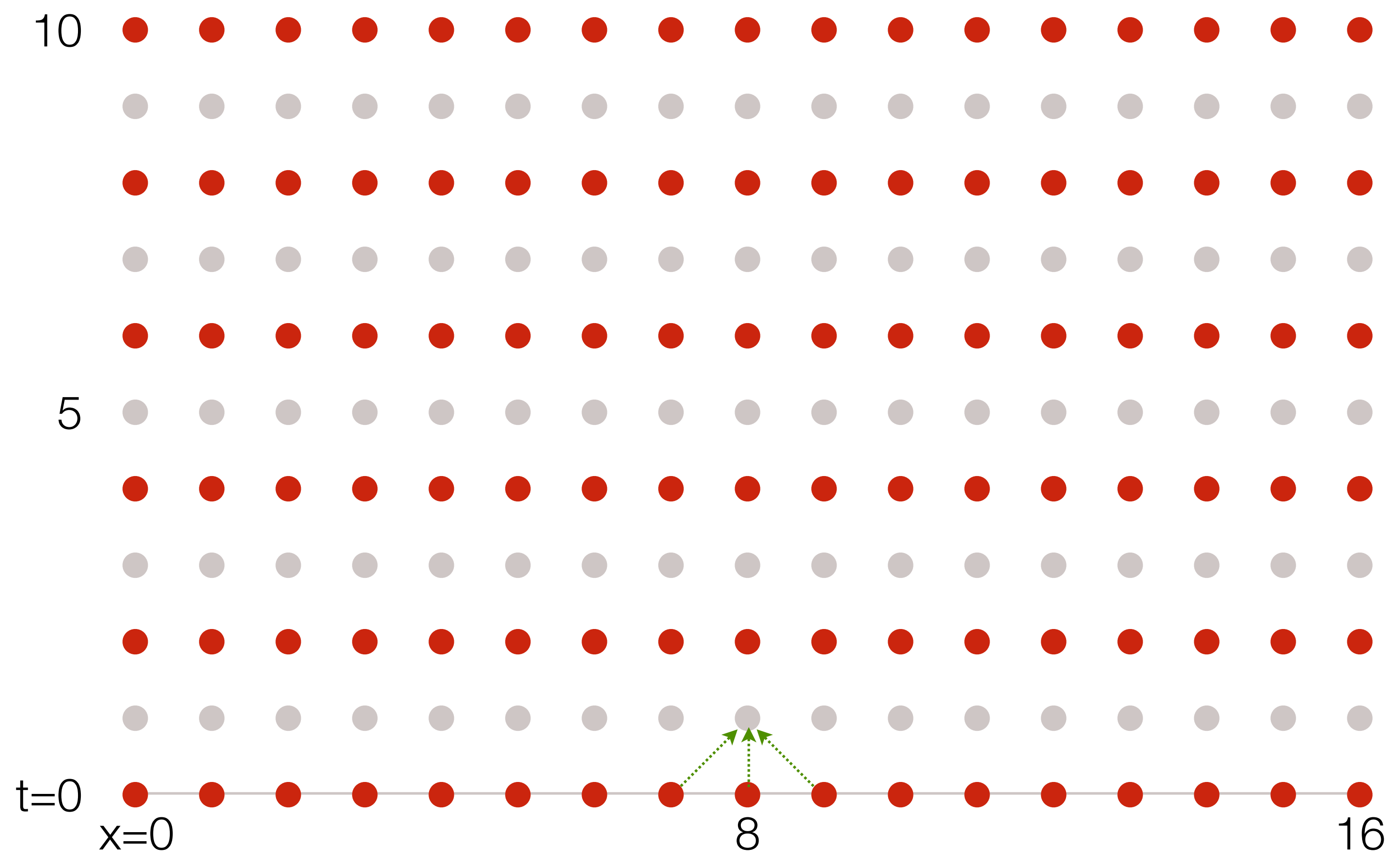
[Frigo and Strumpen (ICS 2005)]
[Datta, *et al*. (2007)]

Cache-oblivious stencil computation



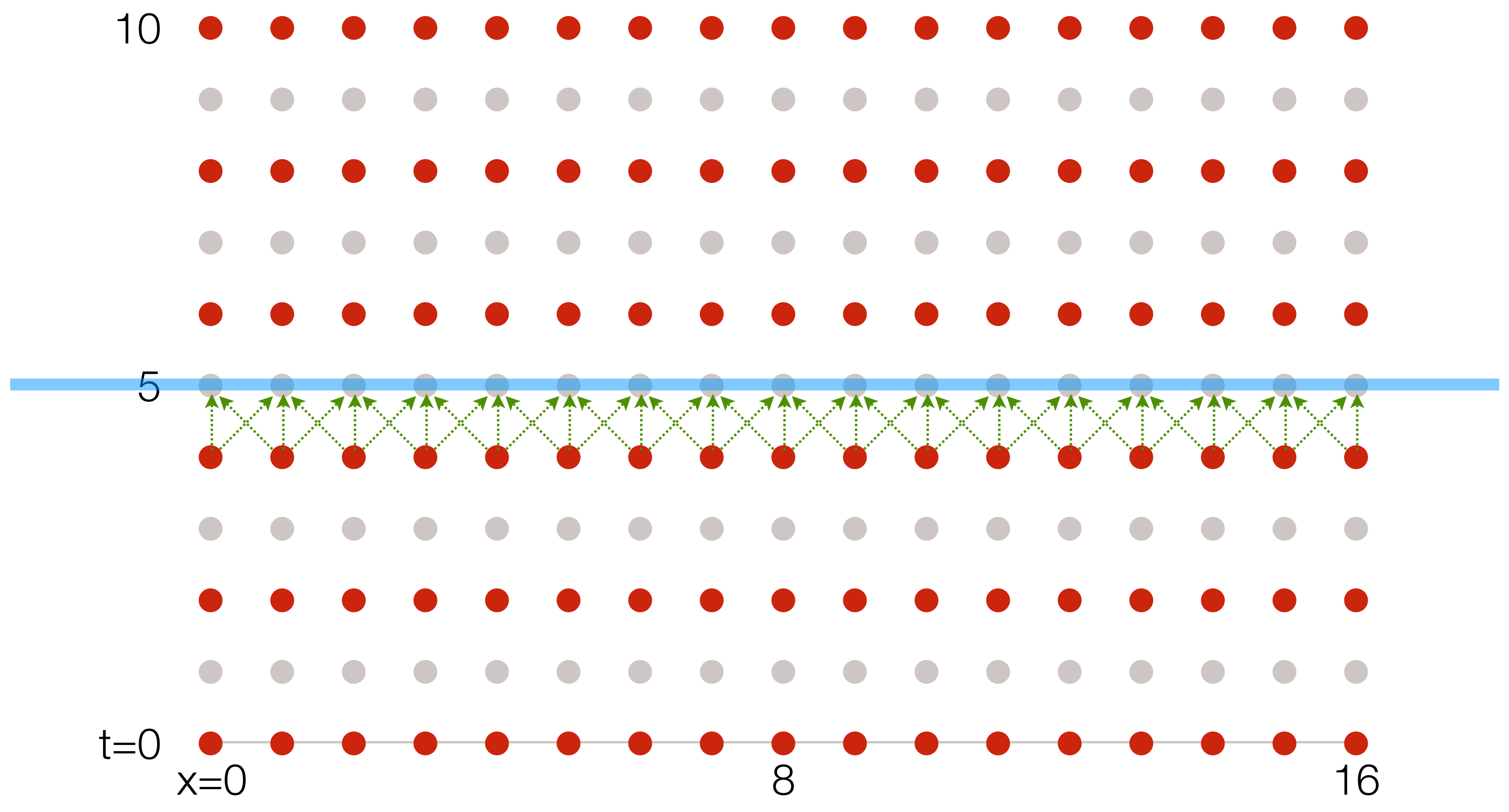
Cache-oblivious stencil computation

w < 2×h:



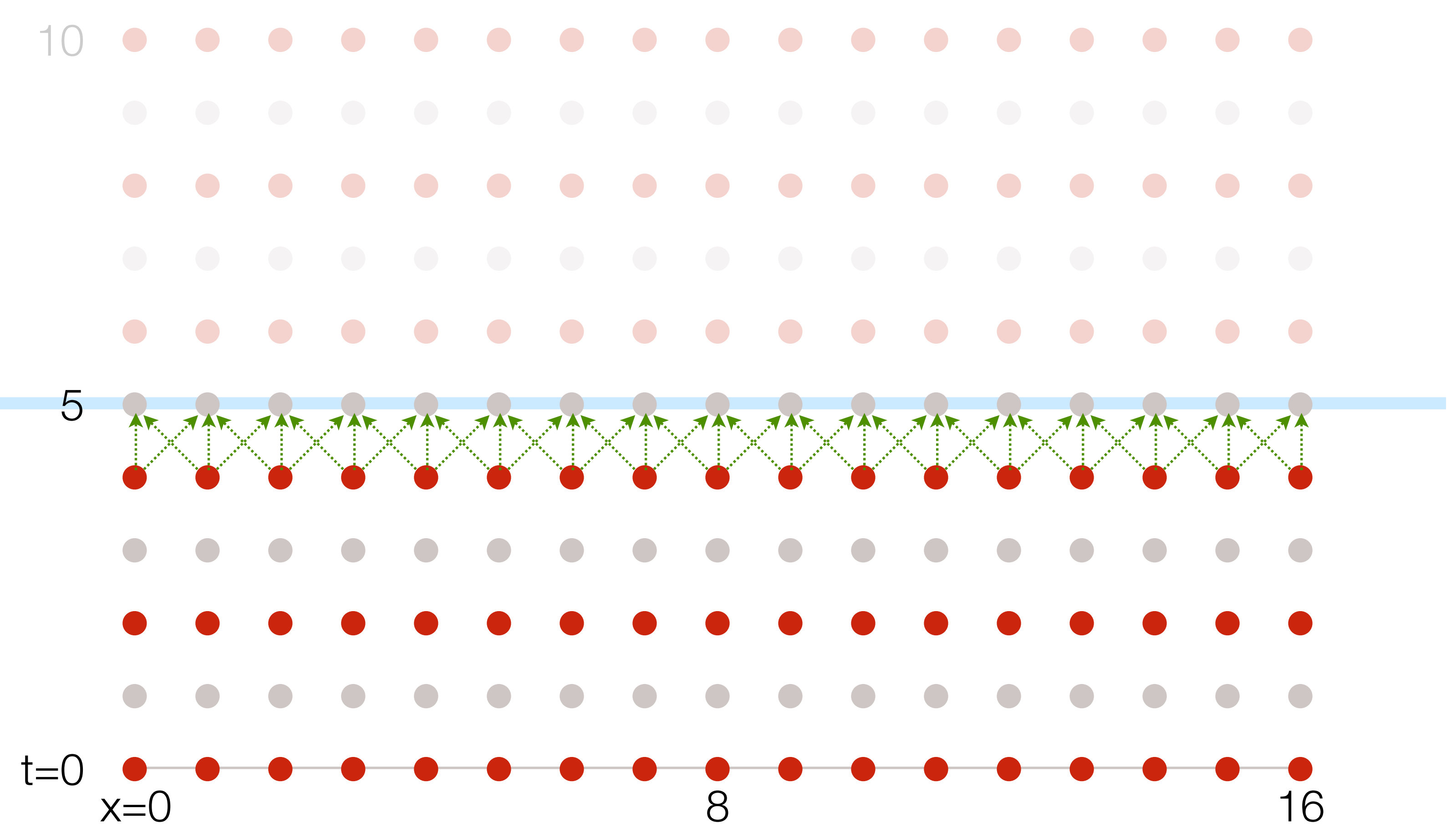
Cache-oblivious stencil computation

w < 2×h ⇒ "Time-cut":



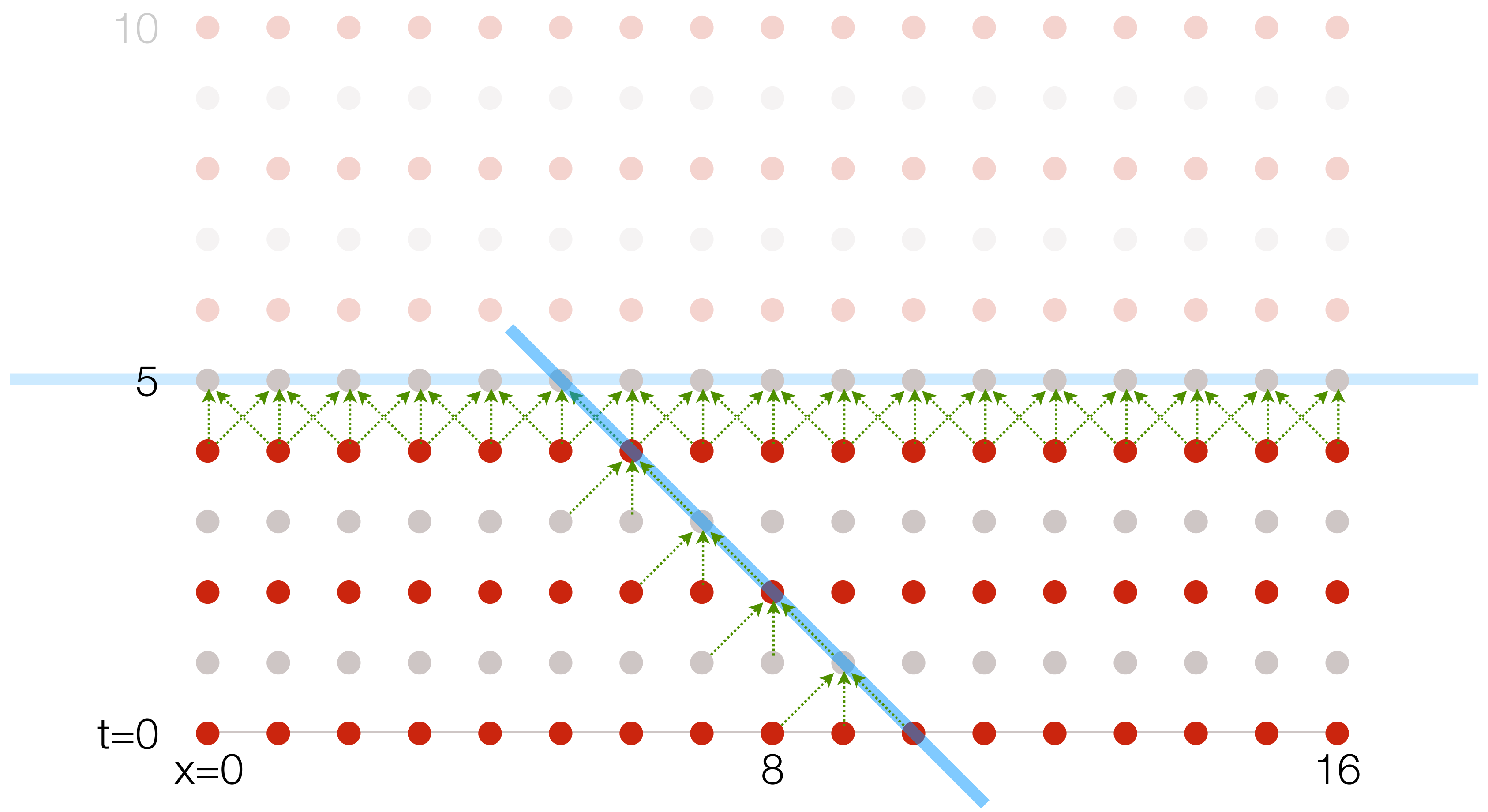
Cache-oblivious stencil computation

w ≥ 2×h:



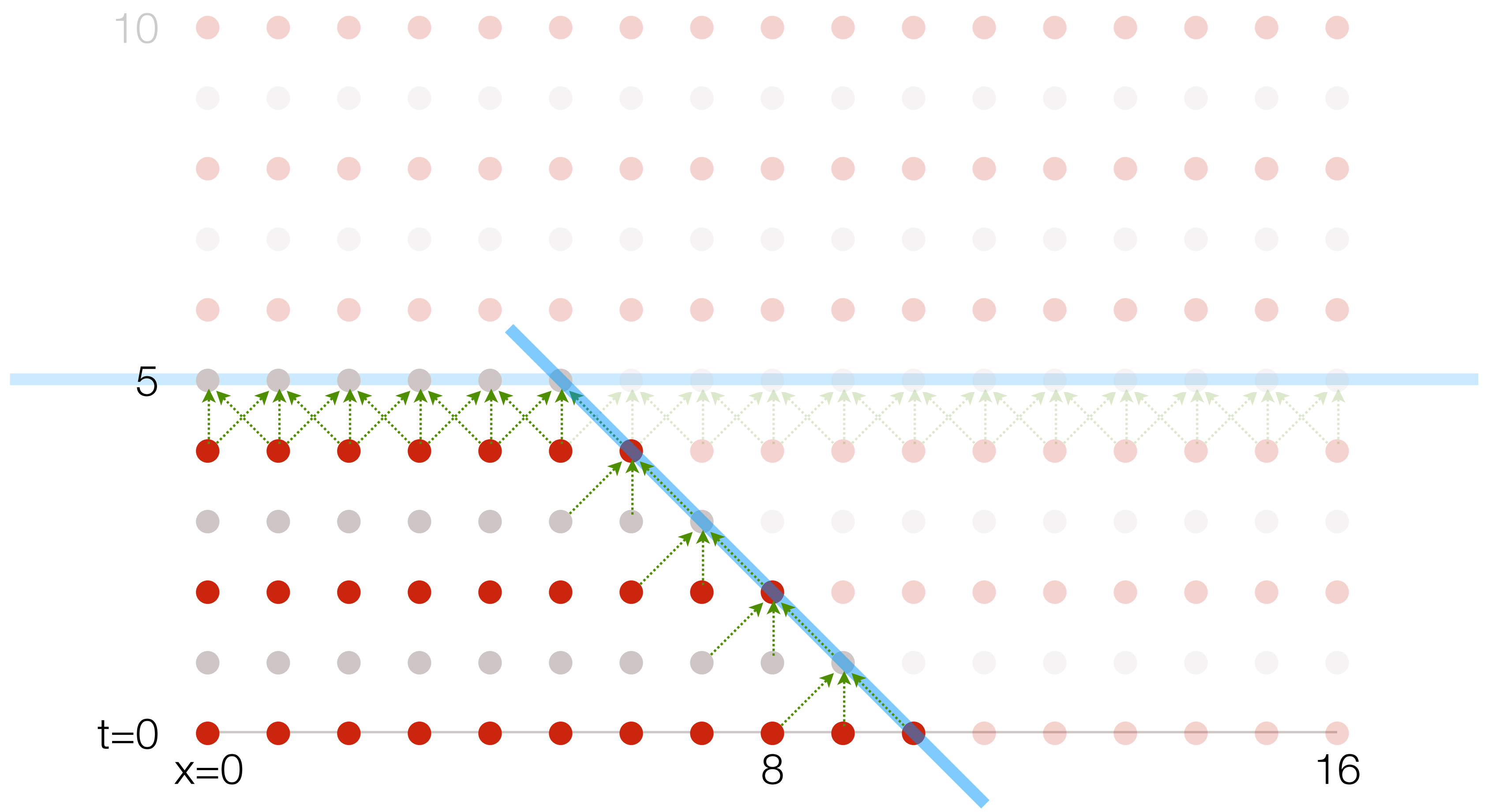
Cache-oblivious stencil computation

w ≥ 2×h ⇒ “Space-cut”:



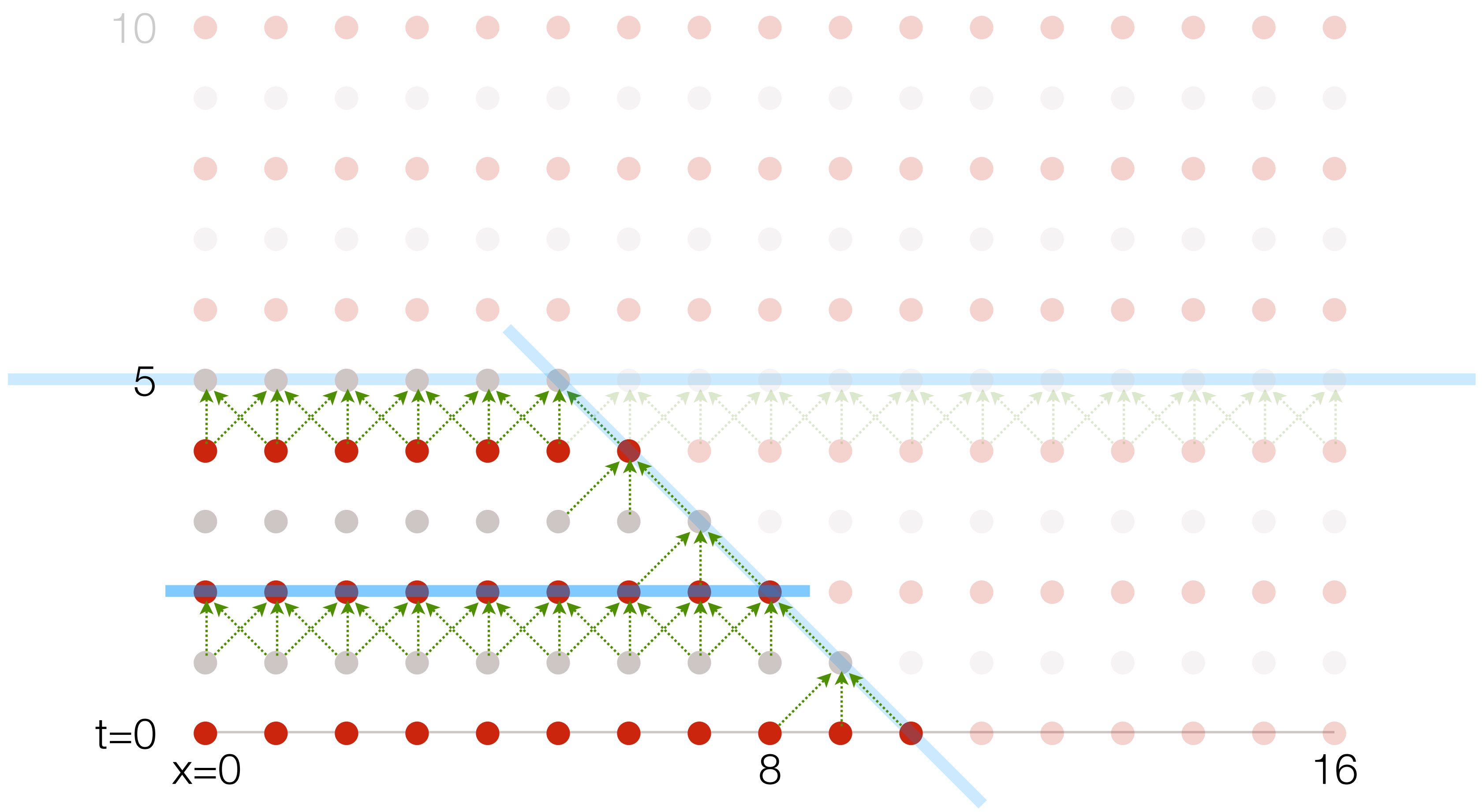
Cache-oblivious stencil computation

w ≥ 2×h ⇒ “Space-cut”:



Cache-oblivious stencil computation

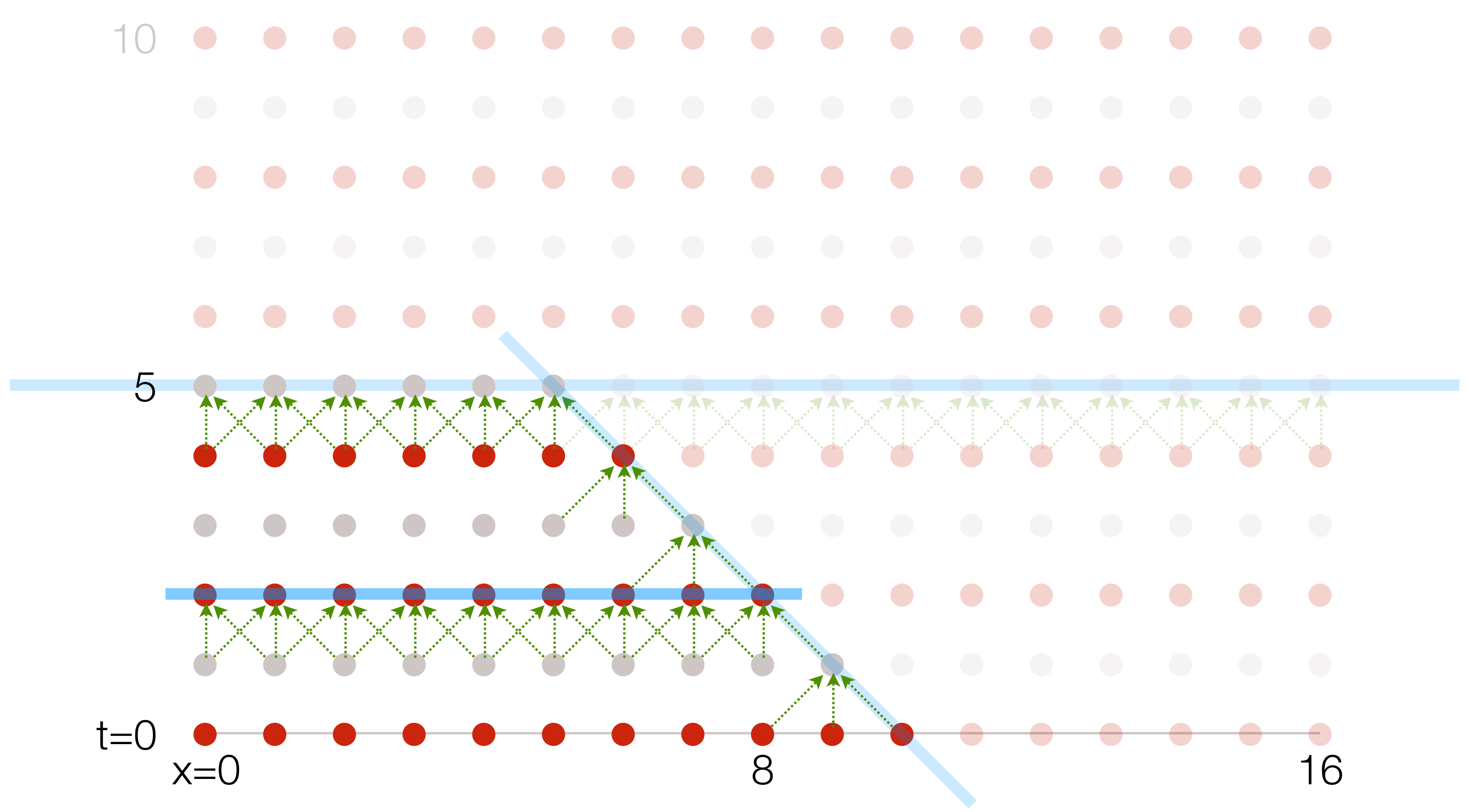
w < 2×h ⇒ "Time-cut":



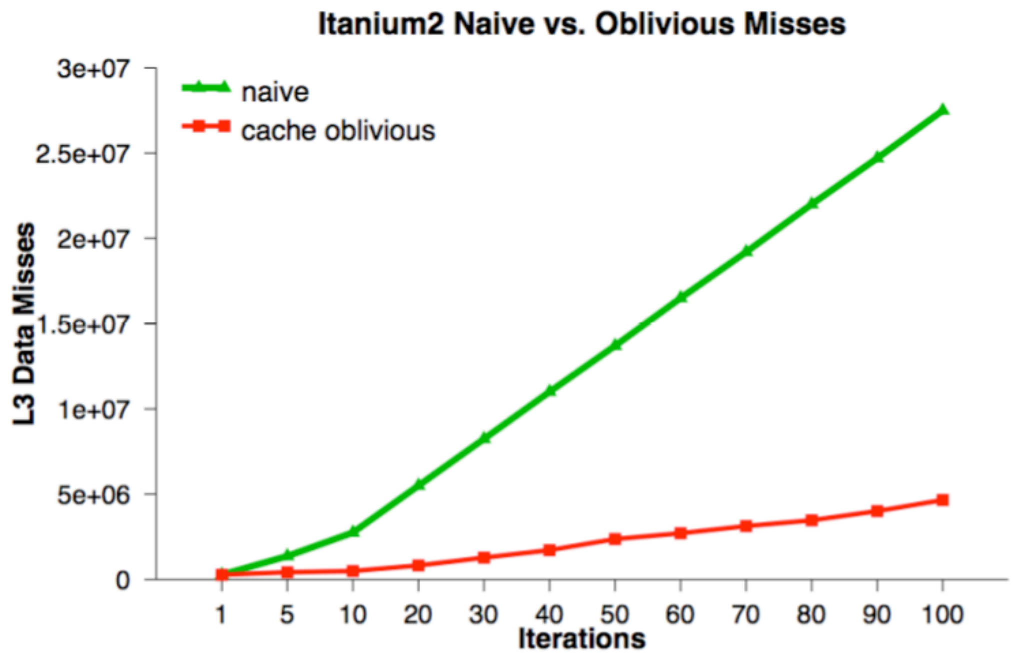
Cache-oblivious stencil computation

Theorem [Frigo & Strumpen (ICS 2005)]:
d **= dimension ⇒**

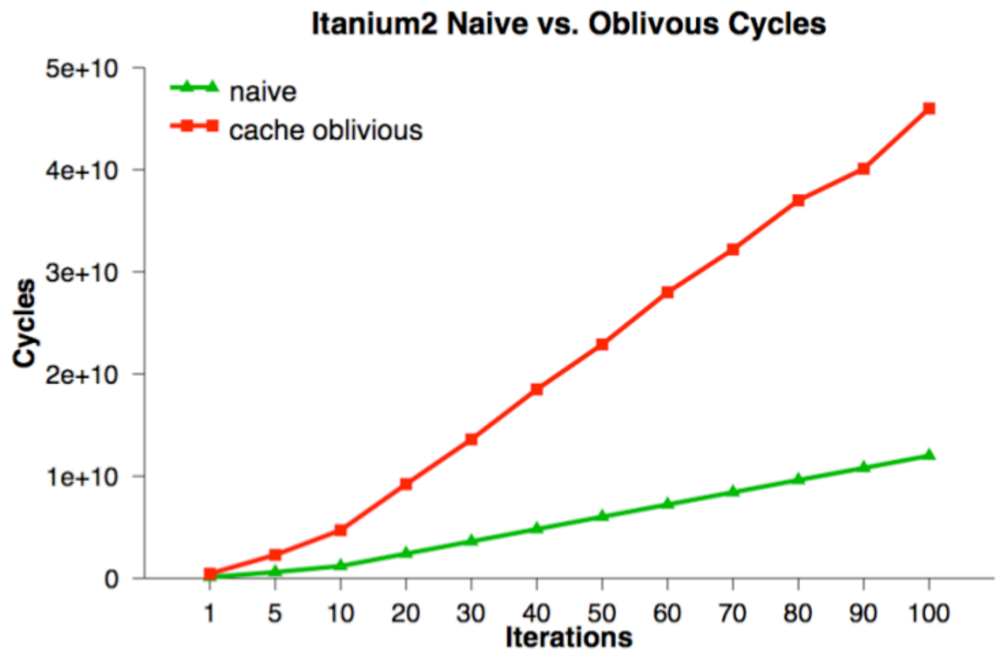
$$Q(n, t; d) \quad = \quad O\left(\frac{n^d \cdot t}{M^{\frac{1}{d}}}\right)$$



Cache-oblivious stencil computation:
Fewer misses but more time



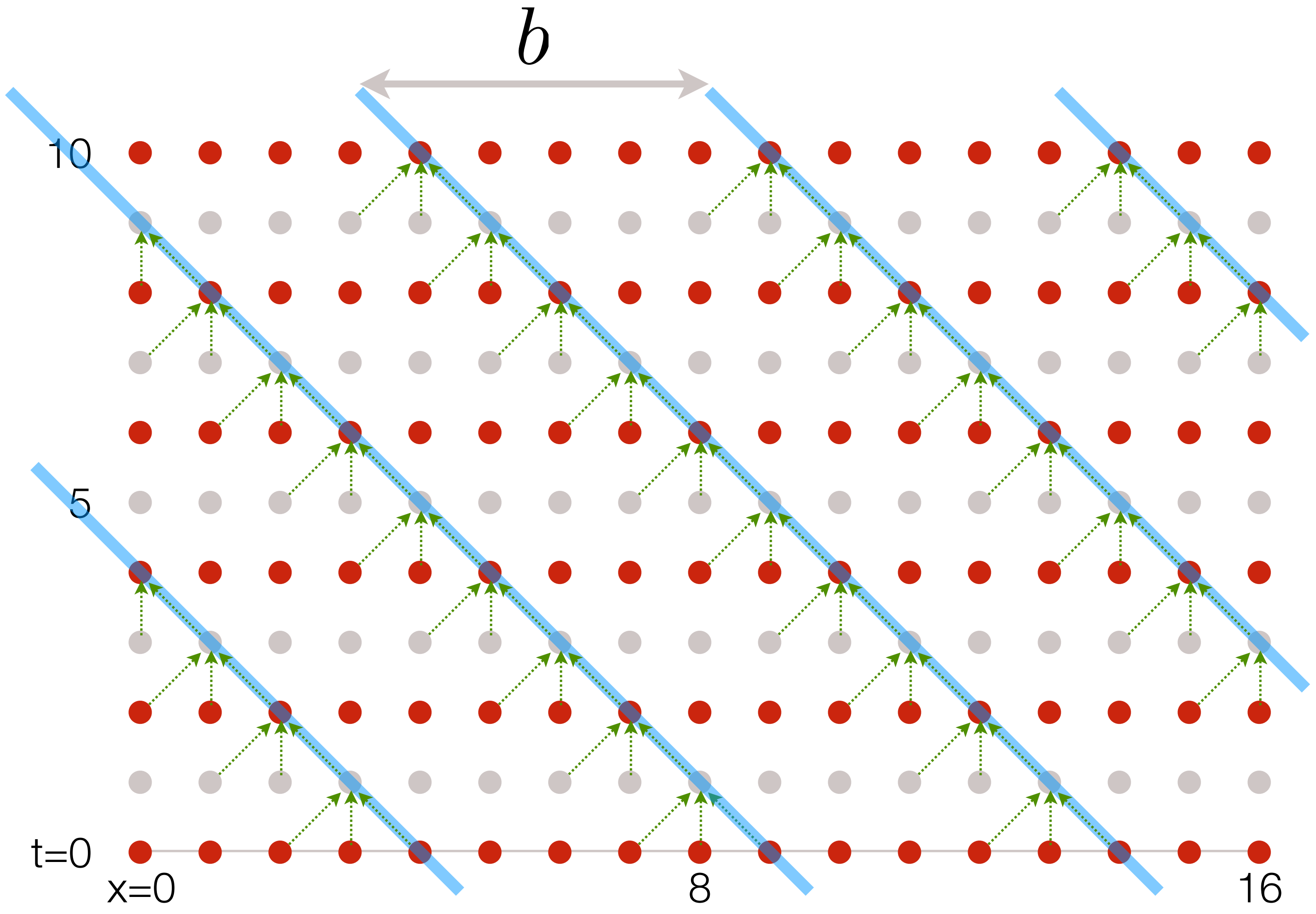
L3 Cache Miss Comparison



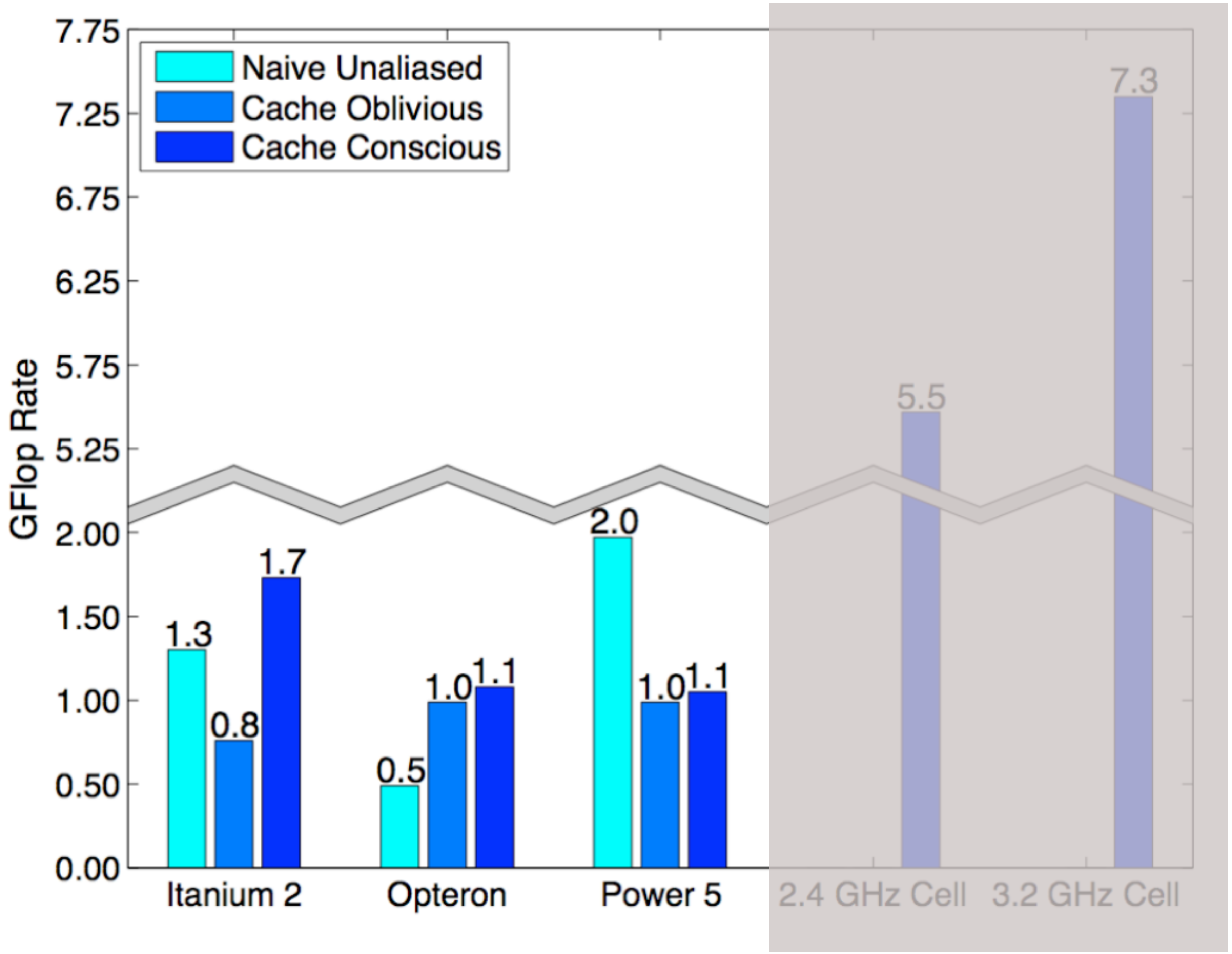
Cycle Comparison

Source: Datta, et al. *(2007)*

Cache-conscious algorithm



Cache-conscious algorithm



Source: Datta, et al. *(2007)*

“In conclusion…”

Backup slides