Georgia Tech College of Computing
Computational Science and Engineering

Numerical algorithms with tunable parallelism

Aparna Chandramowlishwaran, Abhinav Karhu, Ketan Umare,
Richard Vuduc

Georgia Institute of Technology, College of Computing

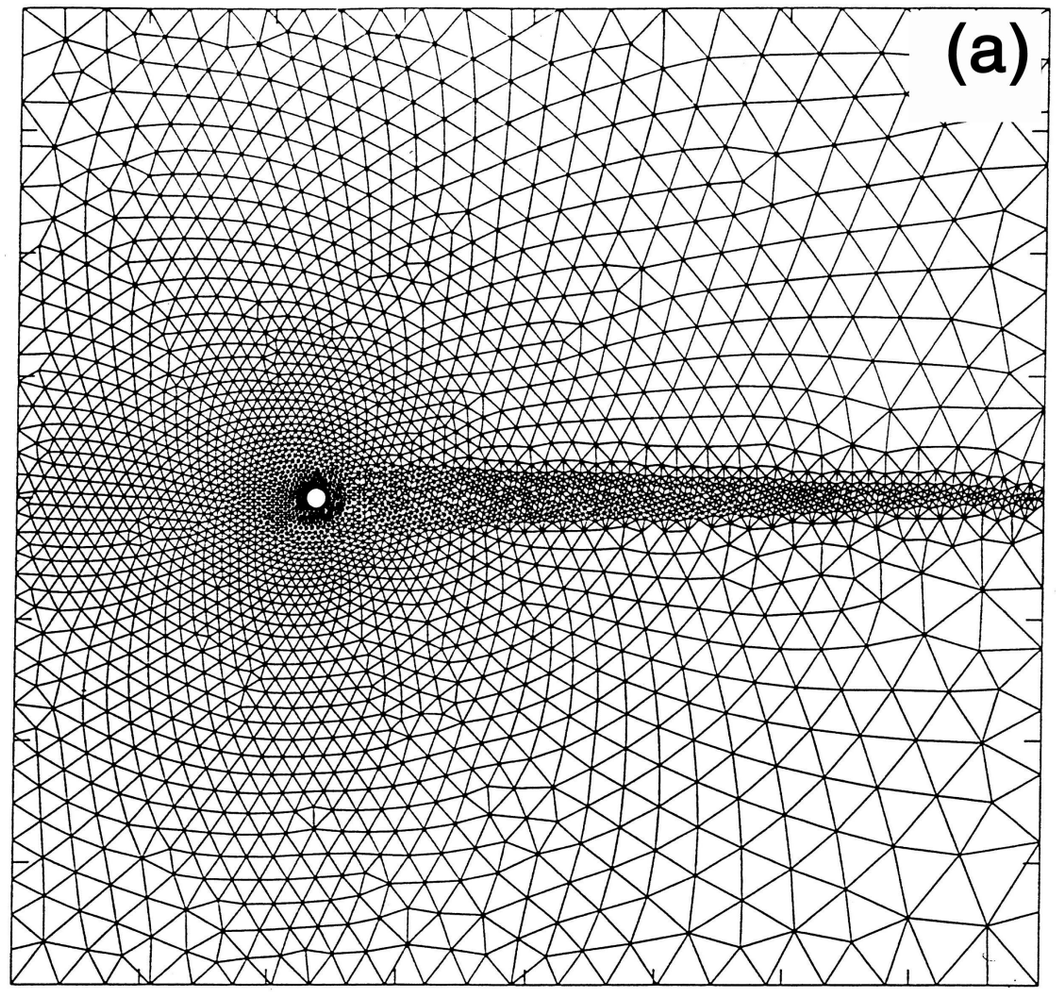
STMCS Workshop (CGO'08)

- **Tunable parallelism**
  - Knob tunes type and degree of parallelism
  - *Example*: data- *vs*. task-/thread-level parallelism

- Trends
  - DAG-based
  - Data structure-centric
  - Wildly asynchronous

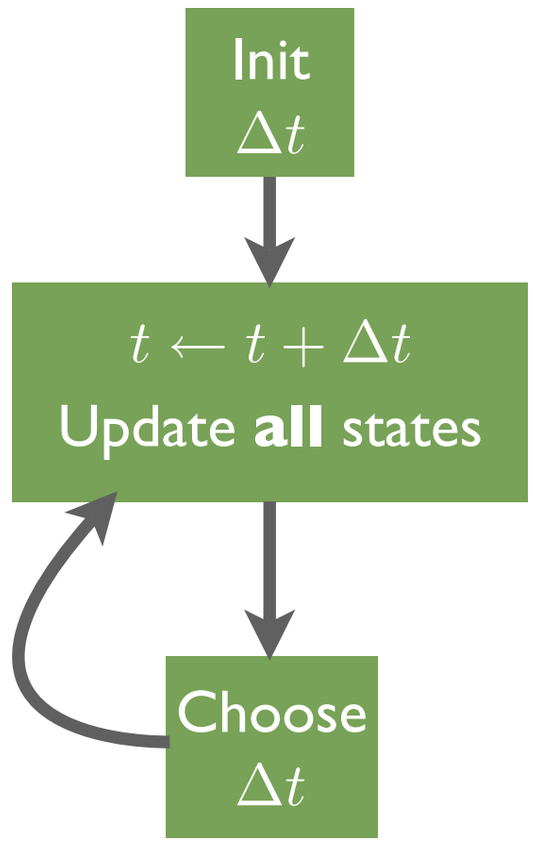
- Questions: Productive interaction?
  - “Enrich” scheduling interfaces?
  - Expose tuning knobs?
  - Optimize software to minimize power?

Motivating problem: Disparate time scales

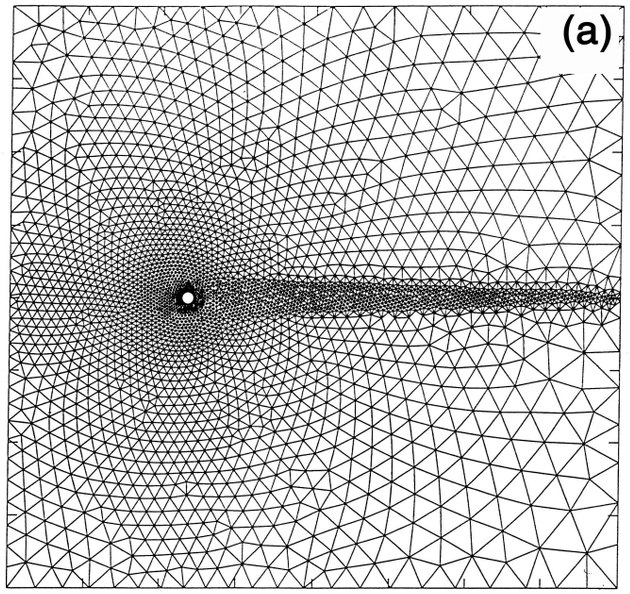


Source: Karimibadi, et al. *(2007)*

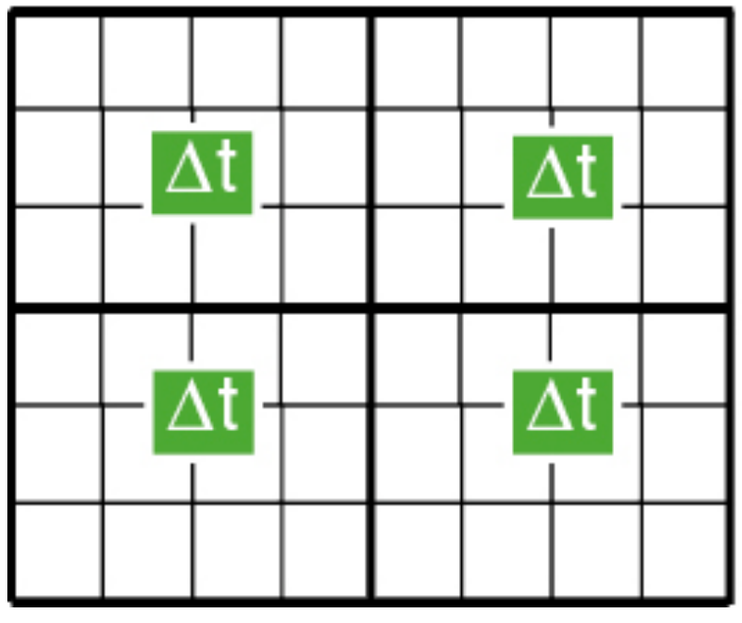
- Disparate time scales
- "Classical" time-stepping



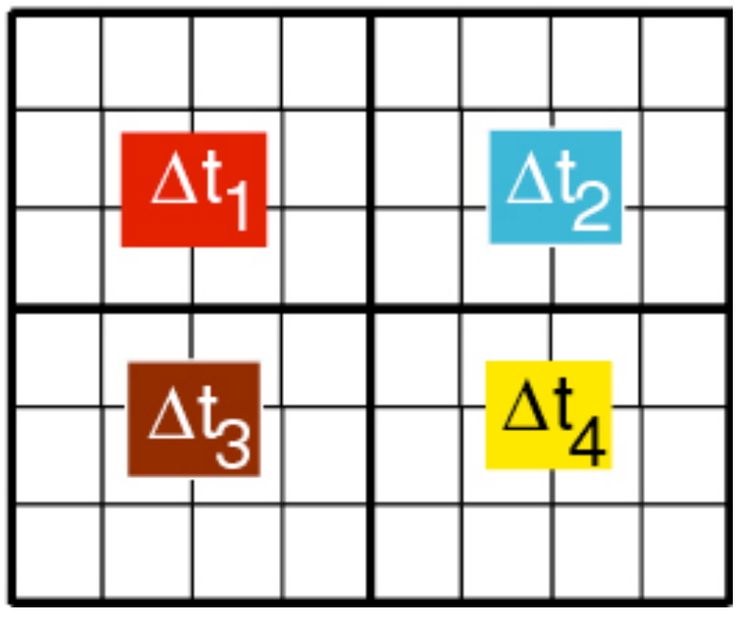
Approach: Asynchronous event-driven algorithms



Time-driven



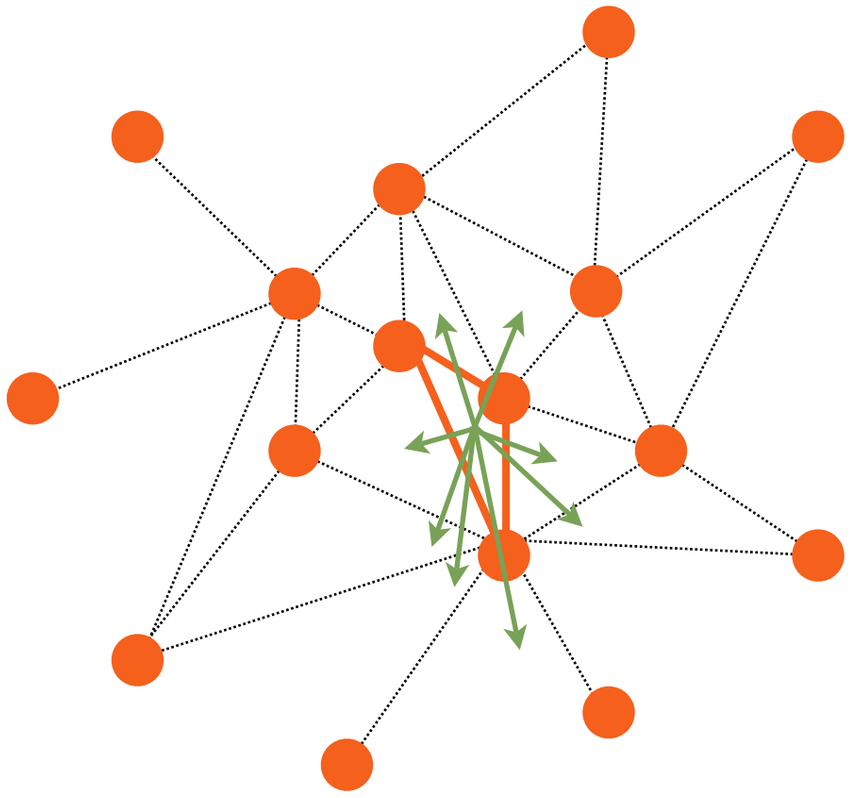
Event-driven

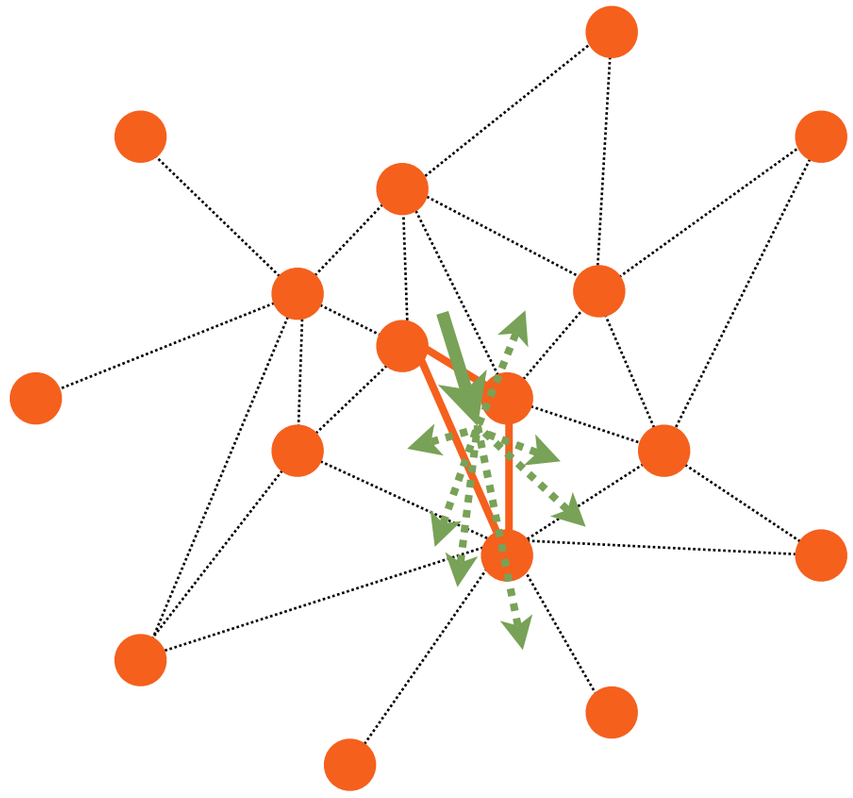


Source: Karimibadi, et al. *(2007)*

Event-driven approach

- Simulate continuous physical system using discrete-event simulation
  - Element updates ⇒ **Time-stamped events**
  - Priority queue for ready events
- Example: **Parallel AVI**
  - "Asynchronous variational integration"
  - Can reduce state updates by ~100x (Lew, *et al.* '03)
  - DAG-based parallel scheduling (Huang, *et al.* '07)

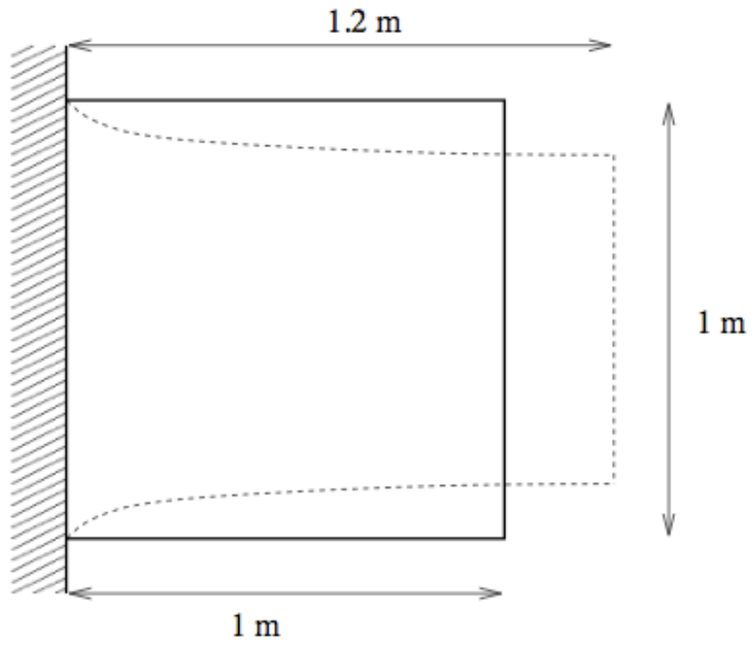


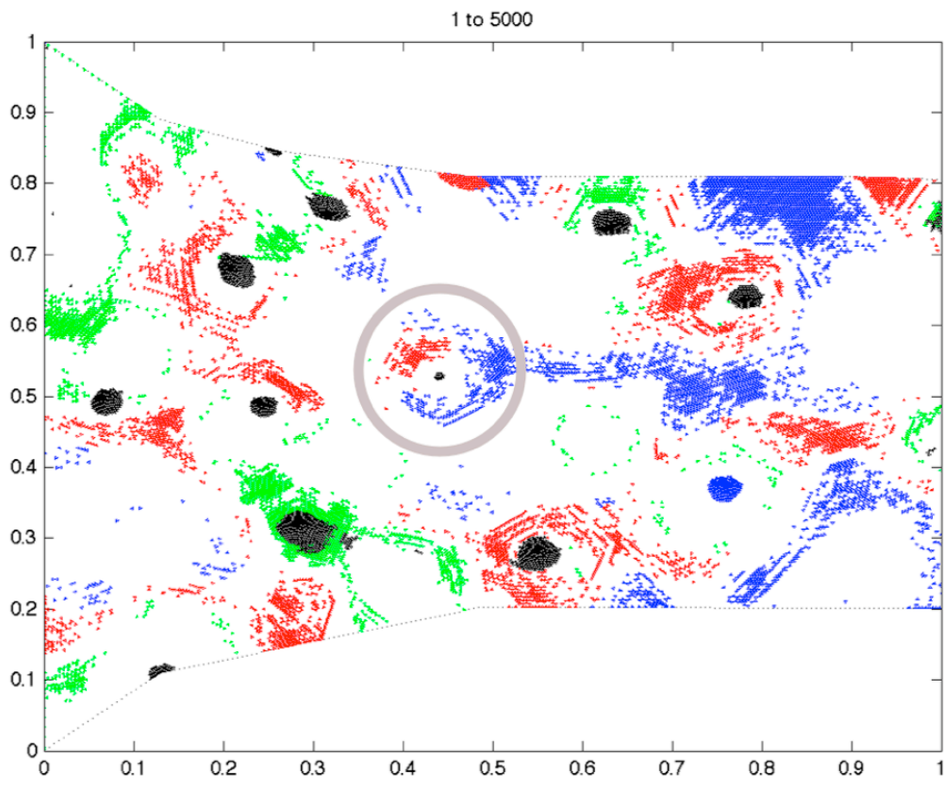


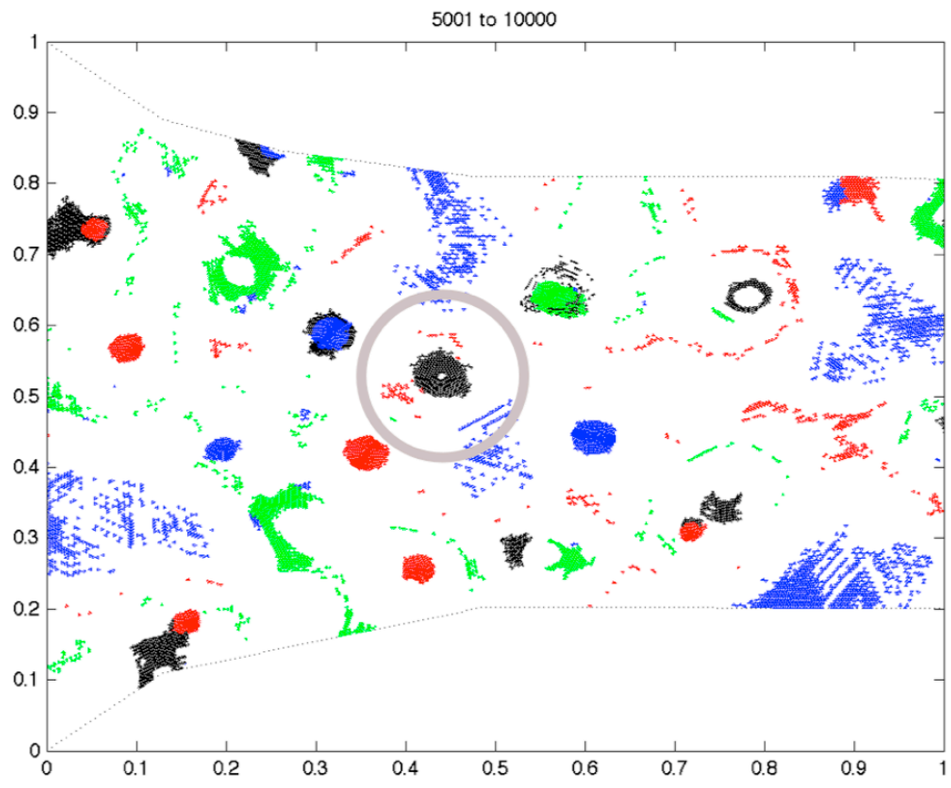
How tunable parallelism arises

- All elements have a minimum time-step (Δt)
- Group elements, assigning min. Δt of the group
  - Within a group: data-parallel
  - Between groups: task-parallel

Theorem (Huang, et al. '07): **Expected parallelism** $\geq \frac{n}{d+1}$

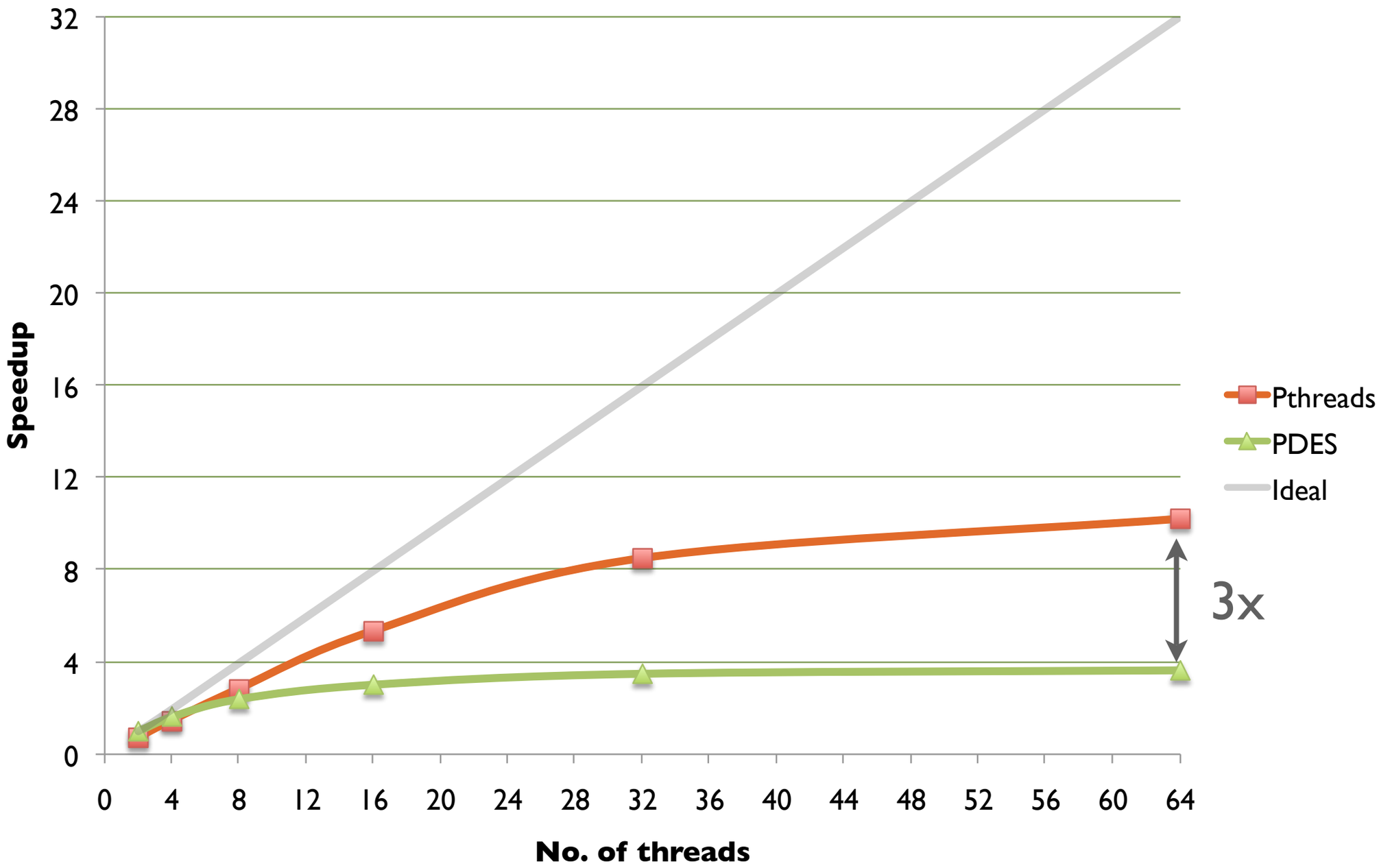


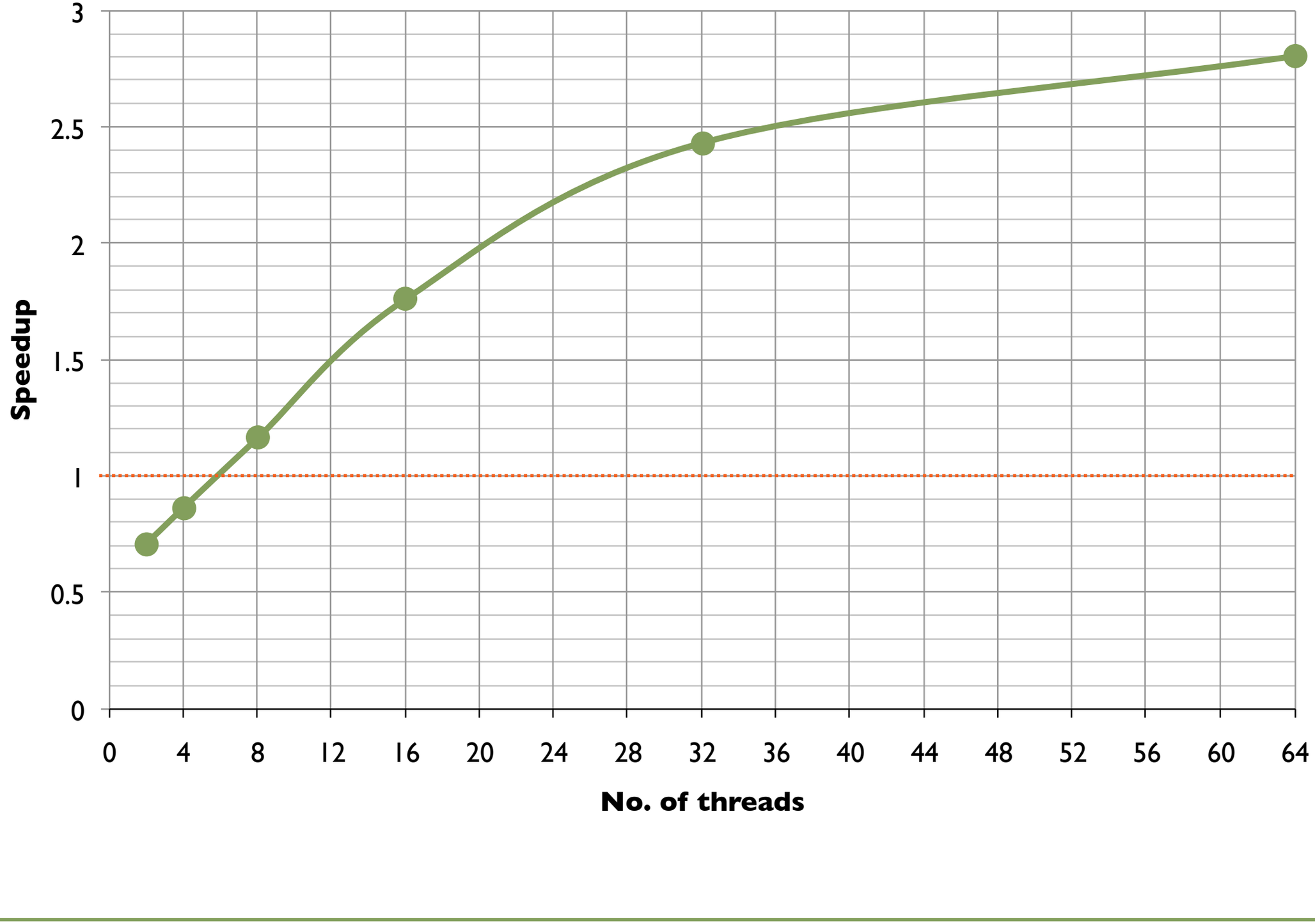




Software library status

- Priority queue-based serial
- DAG-based shared-memory parallel
  - PThreads
  - RSTM
- DES-based: GTW (Georgia Tech TimeWarp)
  - Fujimoto, *et al*. (dev. mid- to late- '90s)
  - Conservative vs. optimistic schemes

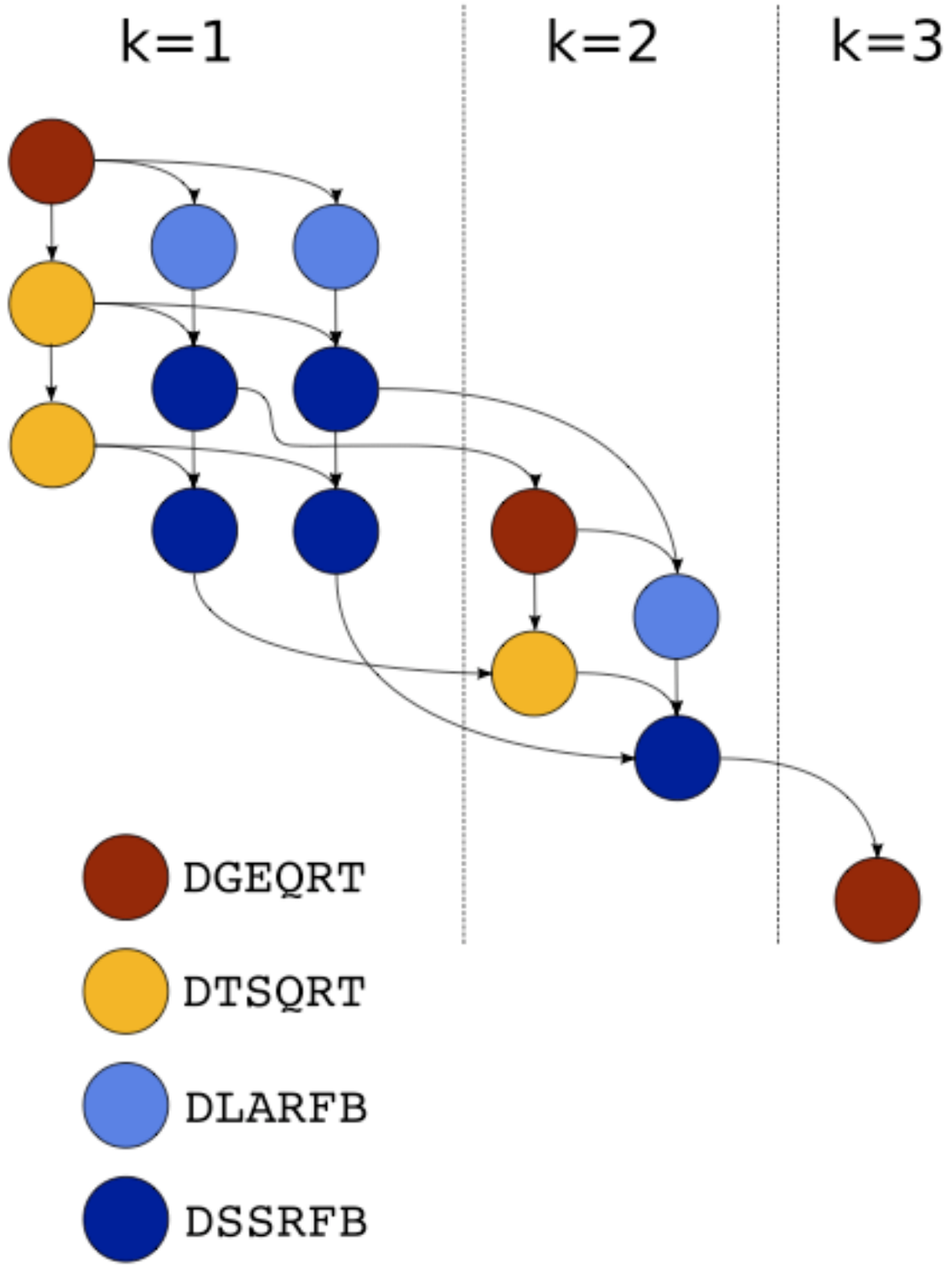

PAVI Speedups on Niagara 2
Speedup
32
28
24
20
16
12
8
4
0
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
No. of threads
Pthreads
PDES
Ideal
3x


PAVI: PThreads speedup over PDES [Niagara 2]
Speedup
3
2.5
2
1.5
1
0.5
0
0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
No. of threads

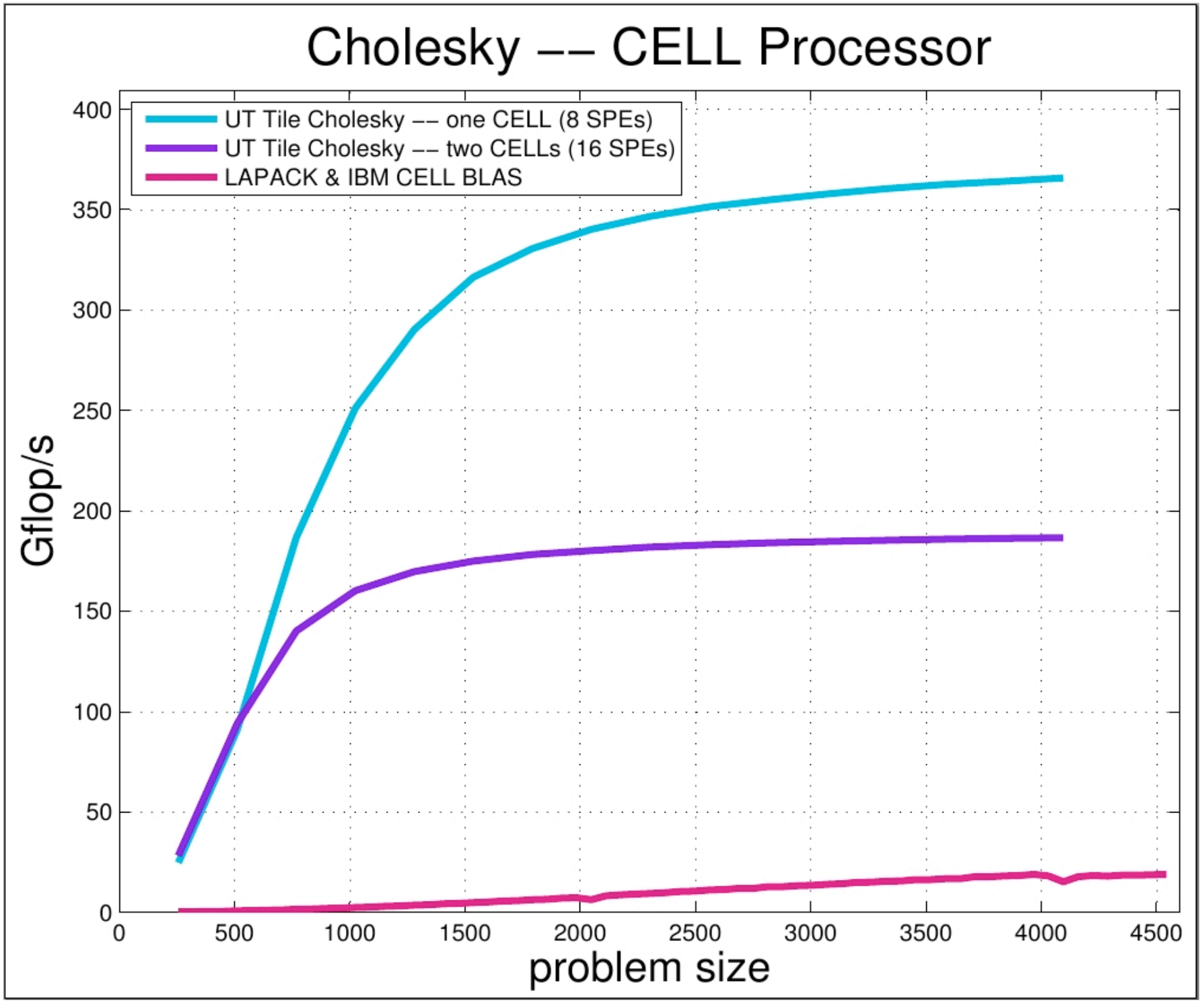
- Building blocks

Algorithm 2 The tiled algorithm for QR

```
1:  for k = 1, 2..., min(p, q) do
2:      DGEQRT(A_kk, V_kk, R_kk, T_kk)
3:      for j = k + 1, k + 2, ..., q do
4:          DLARFB(A_kj, V_kk, T_kk, R_kj)
5:      end for
6:      for i = k + 1, k + 1, ..., p do
7:          DTSQRT(R_kk, A_ik, V_ik, T_ik)
8:          for j = k + 1, k + 2, ..., q do
9:              DSSRFB(R_kj, A_ij, V_ik, T_ik)
10:         end for
11:     end for
12: end for
```

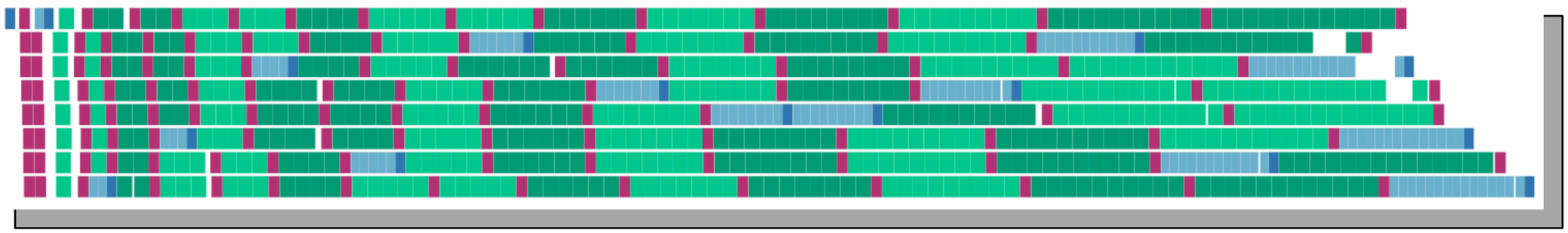


Source: Buttari, Kurzak, Dongarra (2006) [LAWN 191]



Source: Kurzak, Dongarra, Shalf (2008) [SIAM PP 2008; LAWNs 190 & 191]

Cholesky on Cell (1 x 8 core)

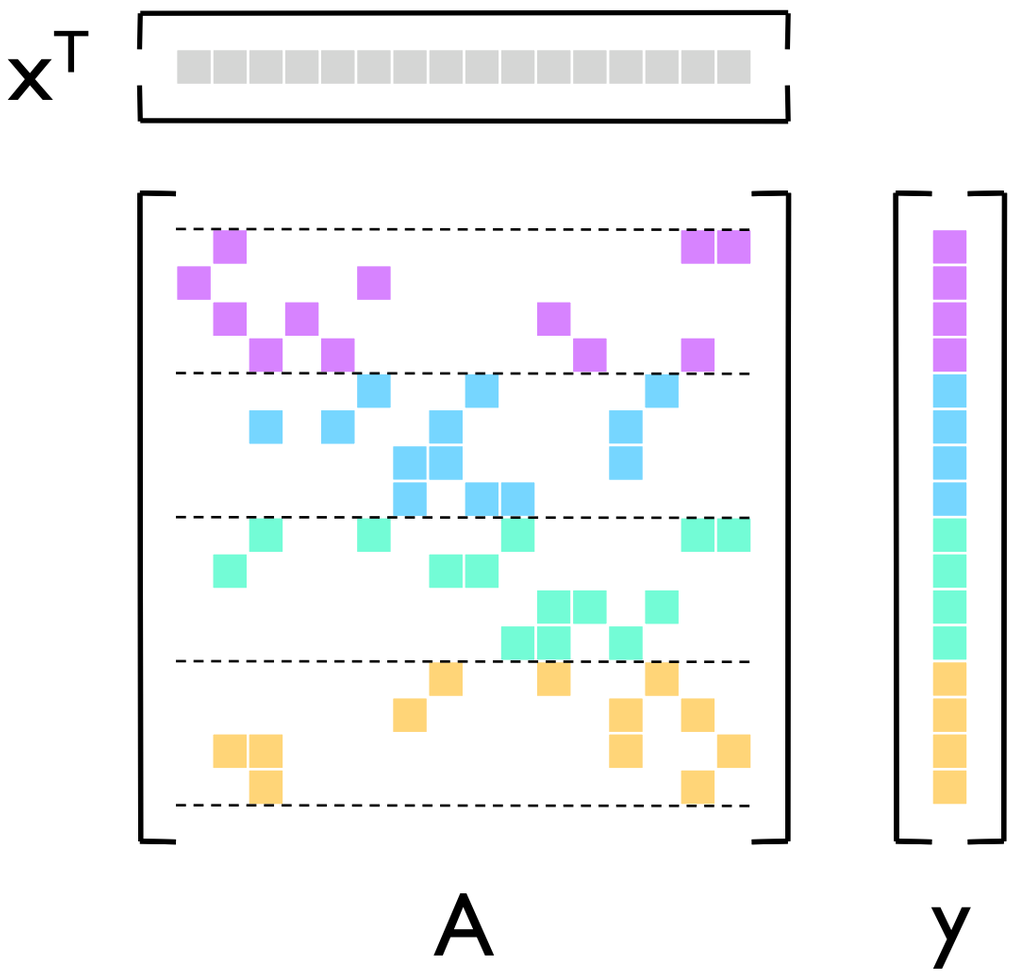


Cholesky on Cell (2 x 8 core)

Source: Kurzak, Dongarra, Shalf (2008) [SIAM PP 2008; LAWNs 190 & 191]

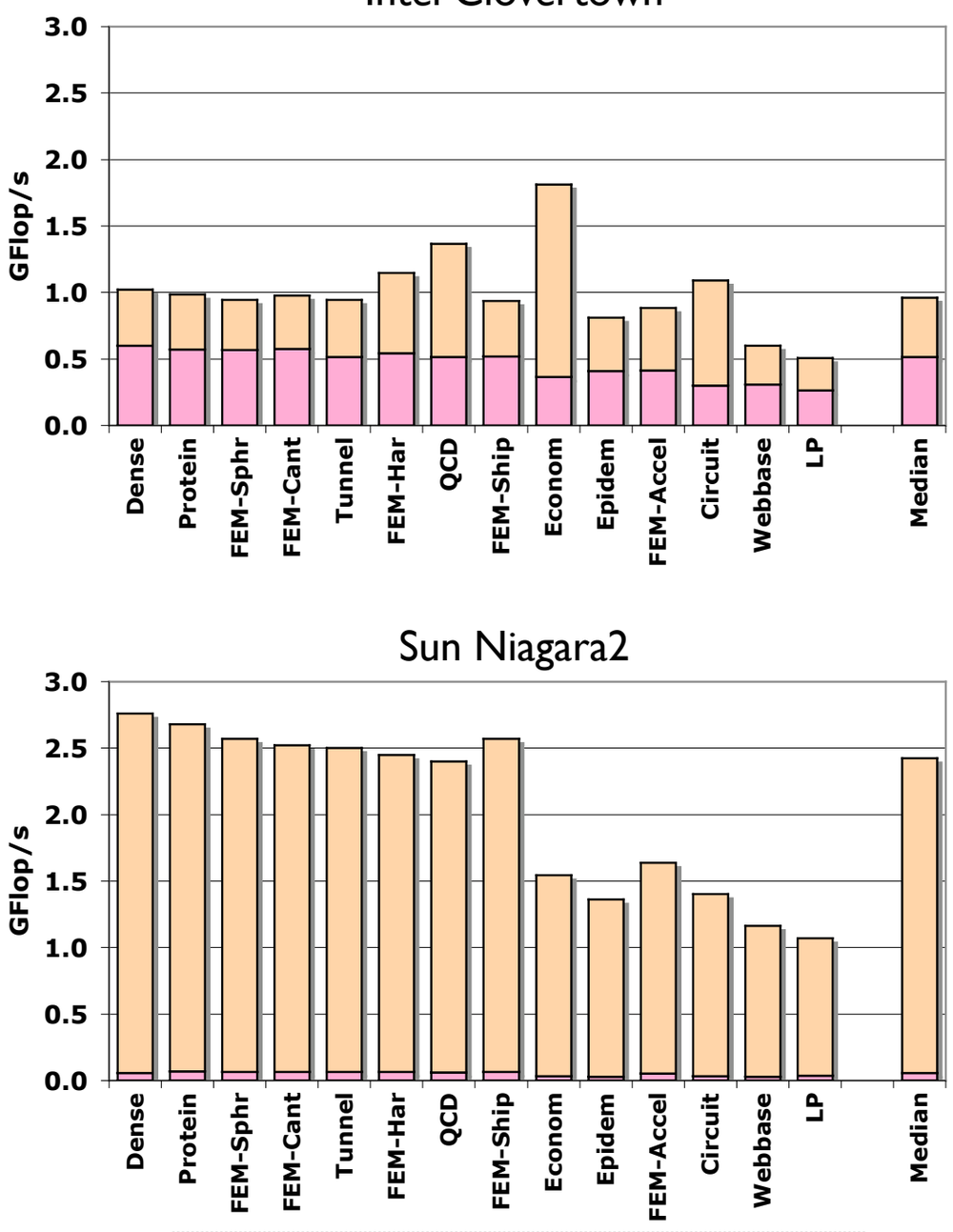
- Data structures
  - Sparse matrix-vector multiply (SpMV)
  - Williams, Vuduc, Oliker, Shalf, Demmel, Yelick (SC'07)

SpMV basics

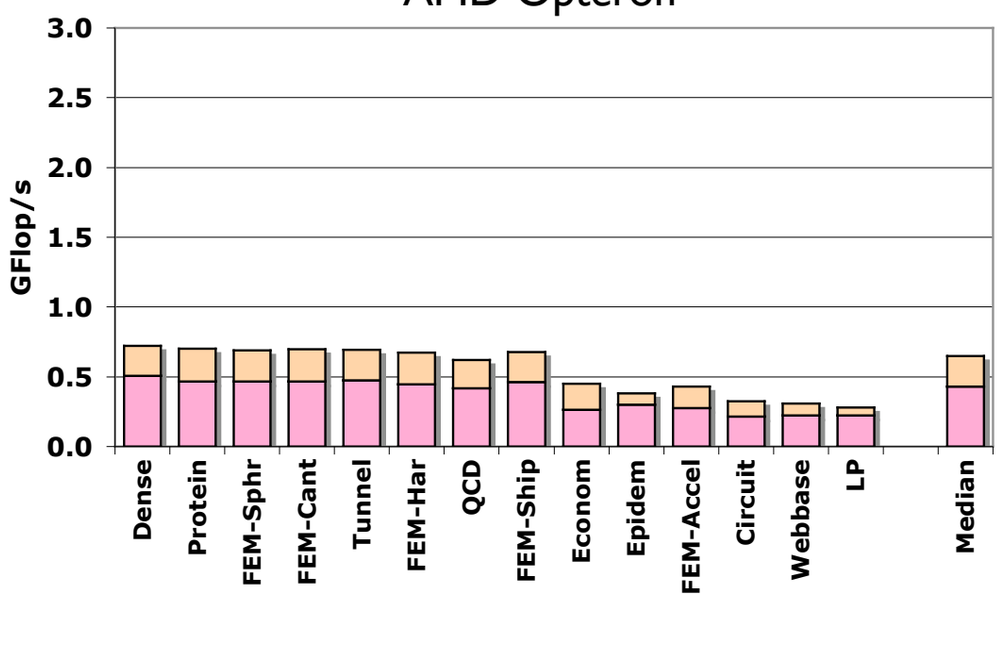


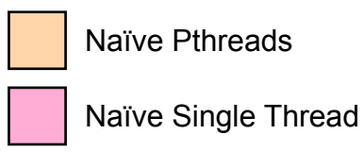
- Sparse vs. dense
  - Low computational intensity
  - Think “adjacency-list”
- Data structure tuning
  - Partitioning
  - Compression

2x4 cores ⇒ 1.9x



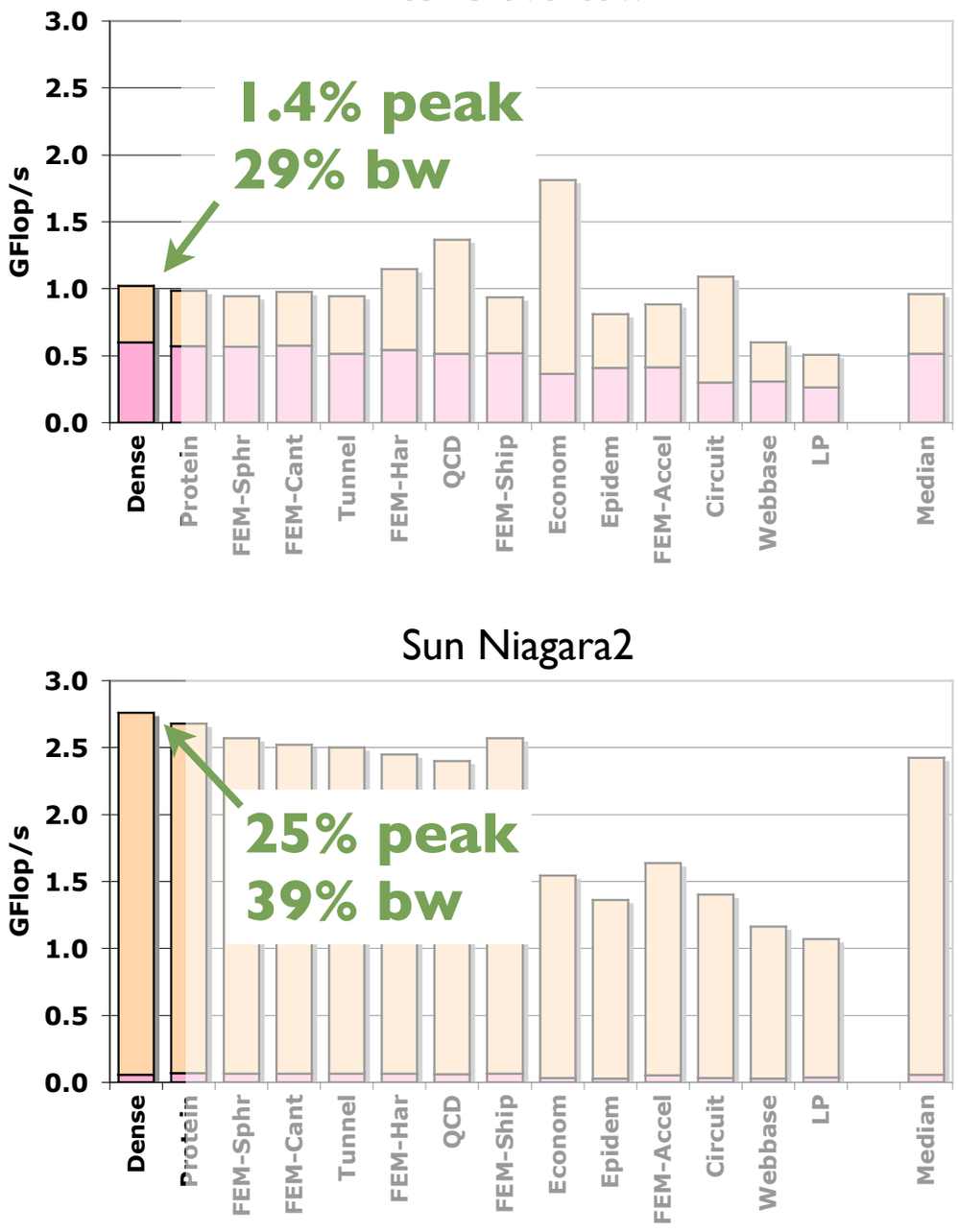
2x2 cores ⇒ 1.5x



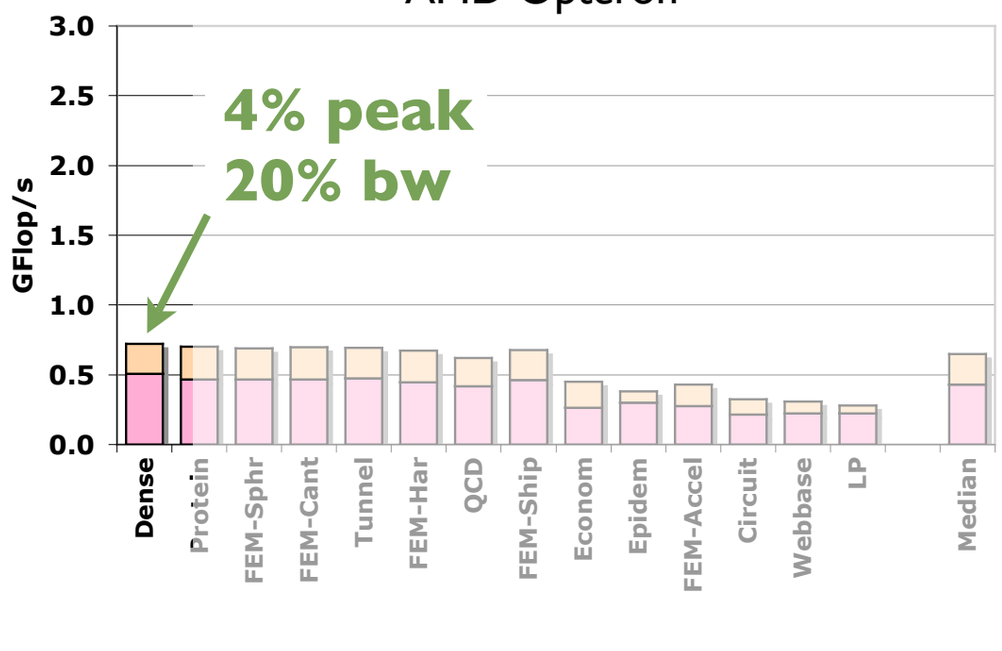


64 threads ⇒ 41x

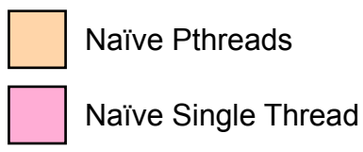
8 cores ⇒ 1.9x



4 cores ⇒ 1.5x



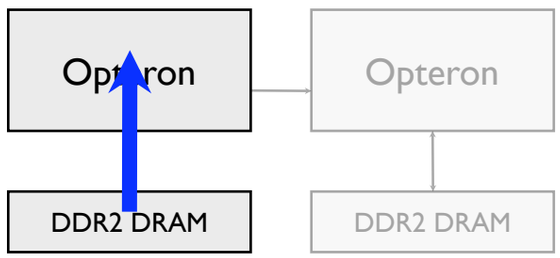
64 threads ⇒ 41x



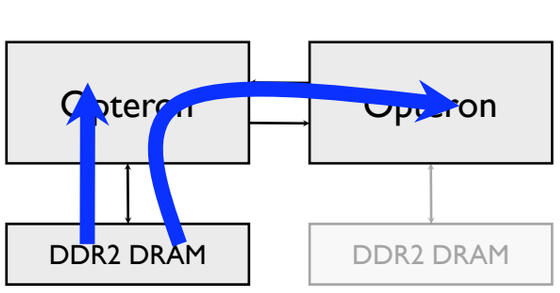
Data-centric tuning

- Bind
- Prefetch
- Compress
- Block
- Buy (more DIMMs)

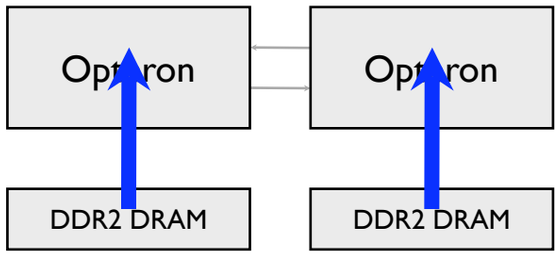
Bind



Single Thread



Multiple Threads,
One memory controller



Multiple Threads,
Both memory controllers

Idea:

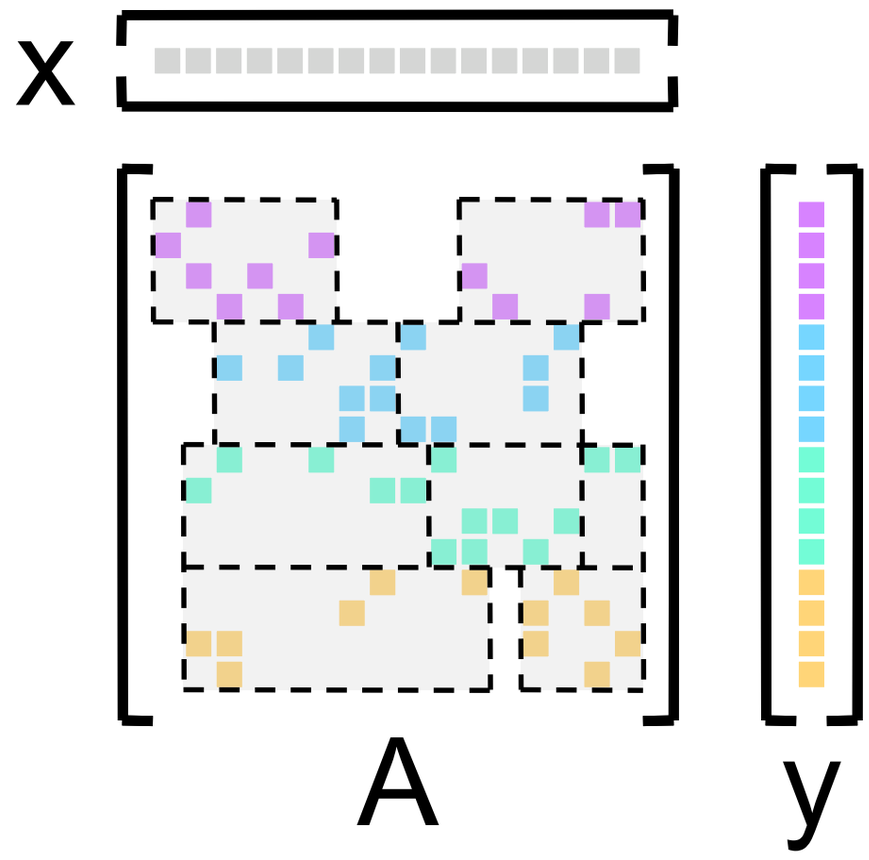
Exploit NUMA, affinity

Bind adjacent blocks to adjacent cores using

Cell: libnuma

Opteron: Custom, based on Linux scheduler

Block (Cache & TLB)



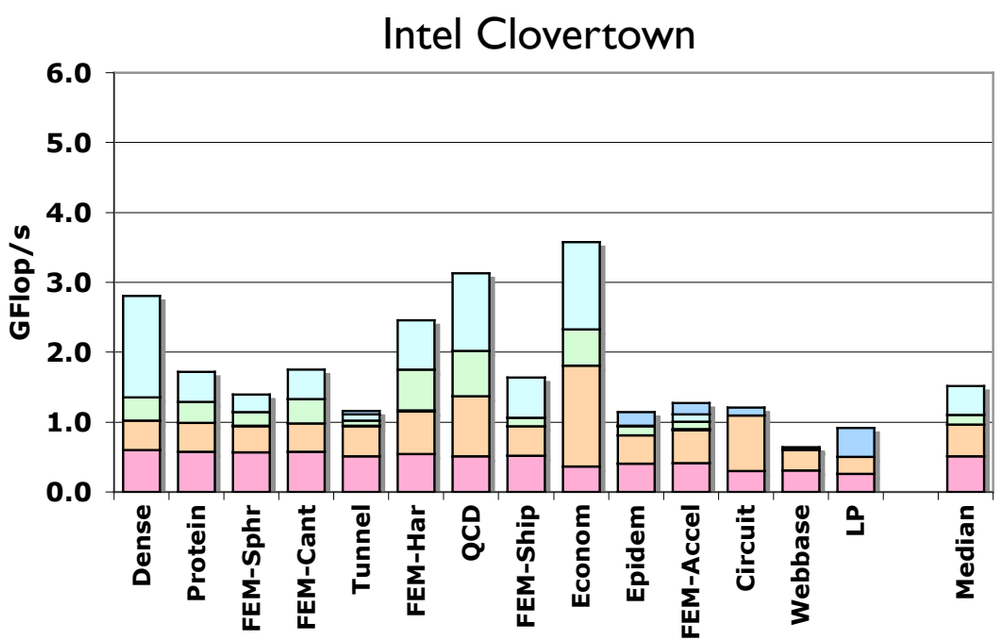
Like dense case,

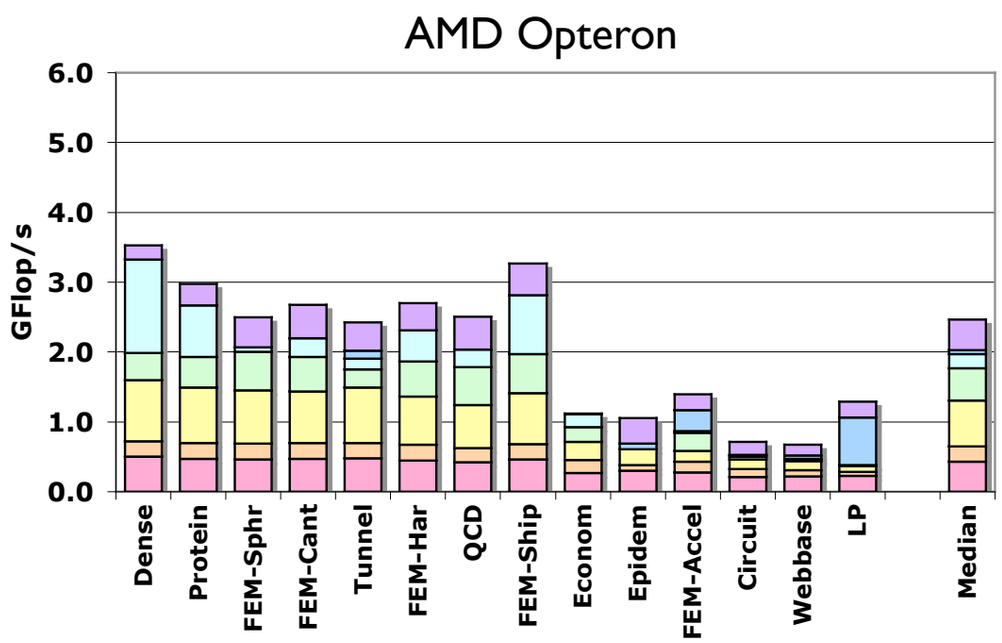
localizes ‘x’ loads for reuse.

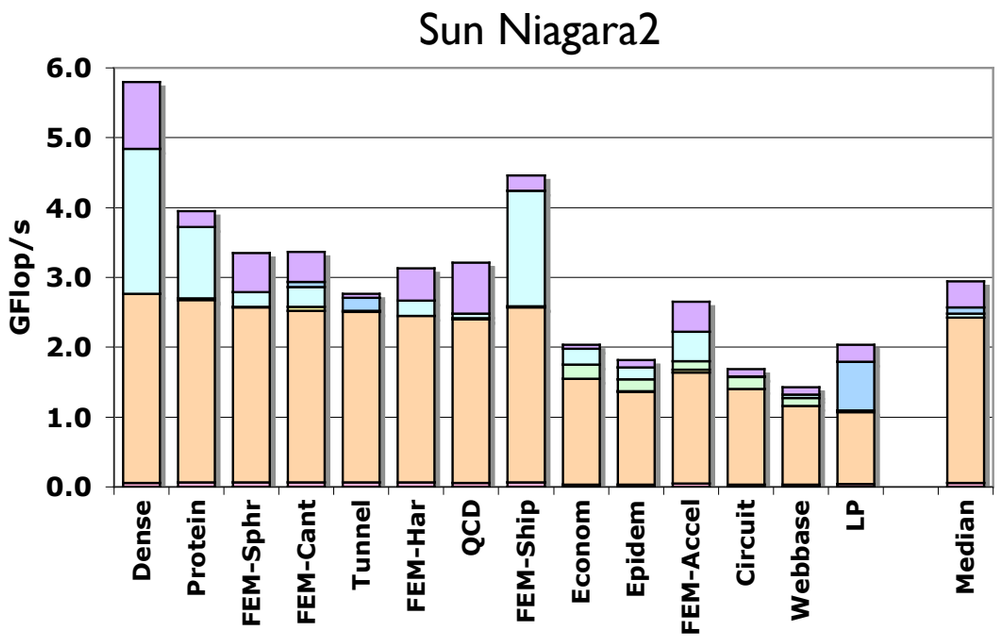
Partition into cache blocks.

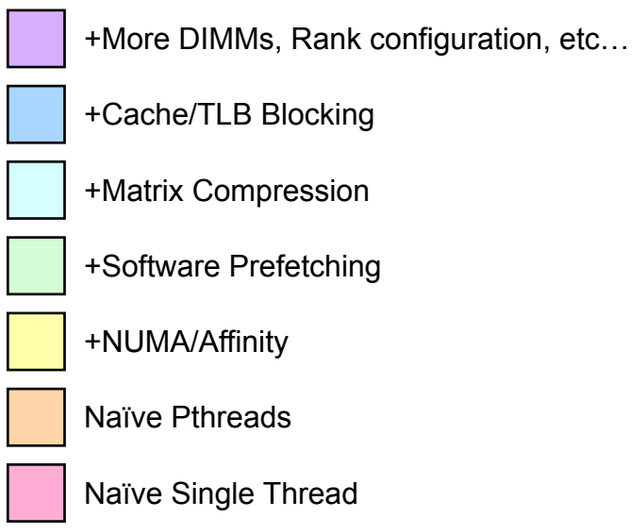
Tune each block.

Data-centric multicore tuning



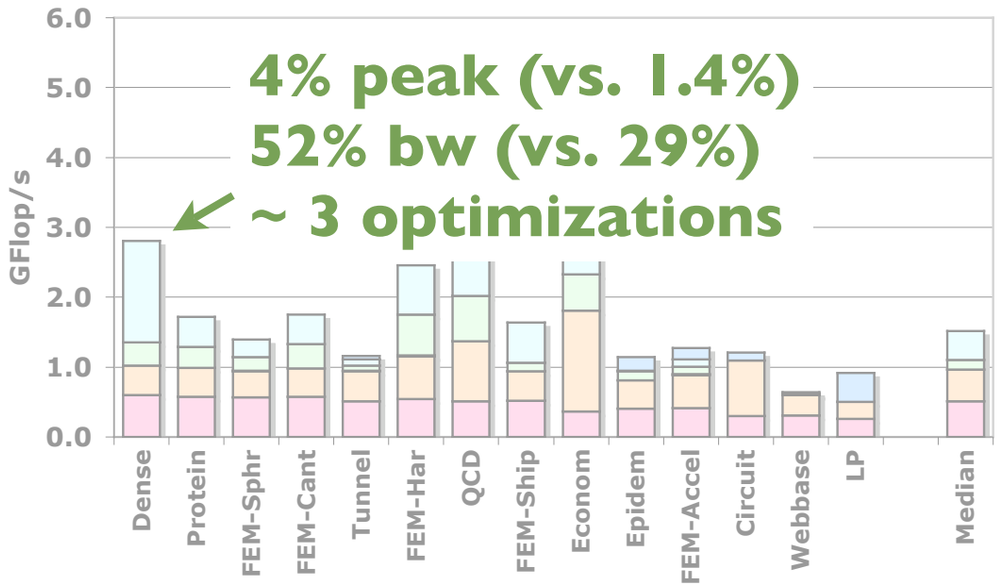




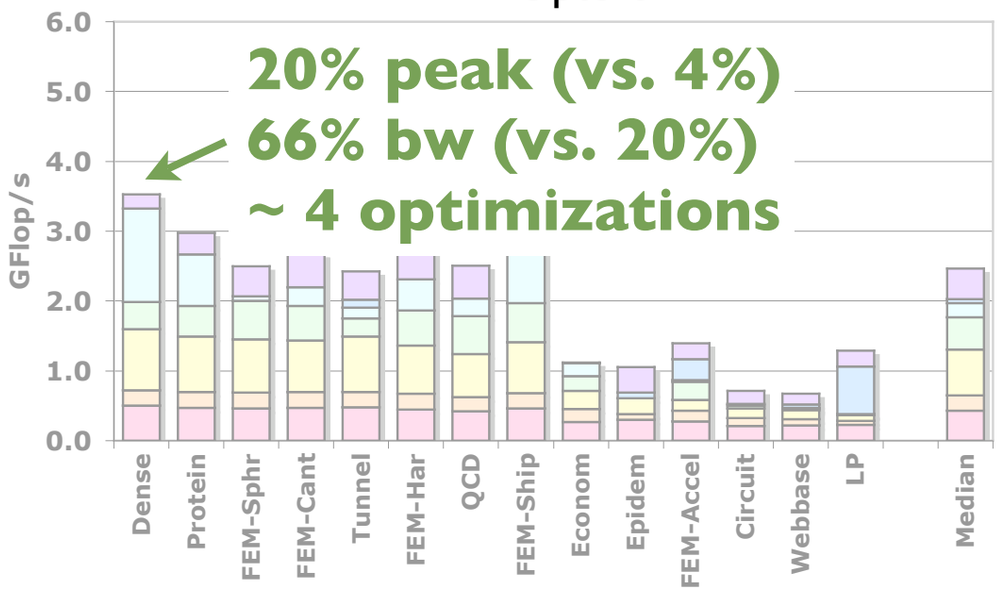


Where do we stand?

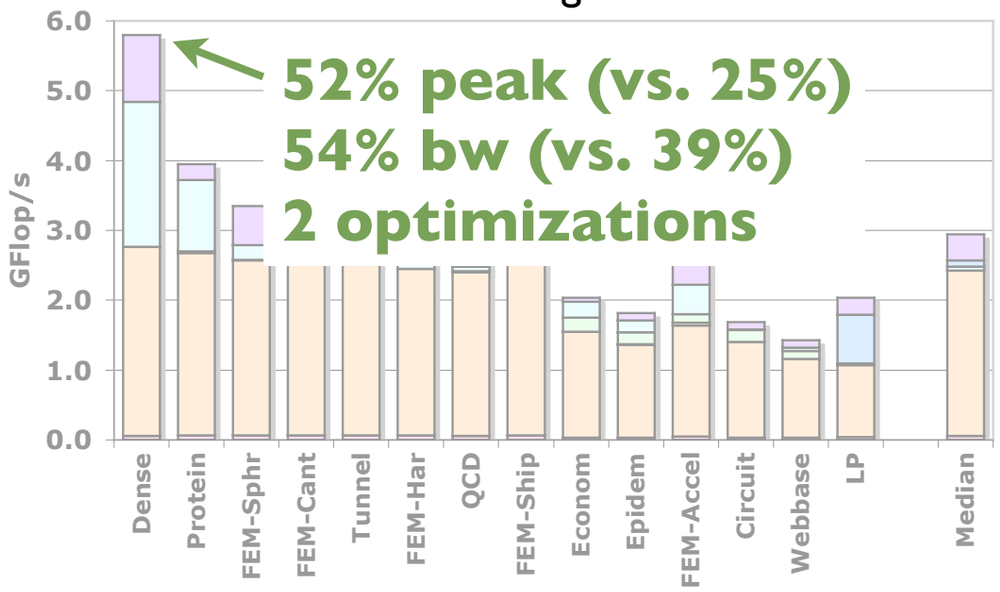
Intel Clovertown

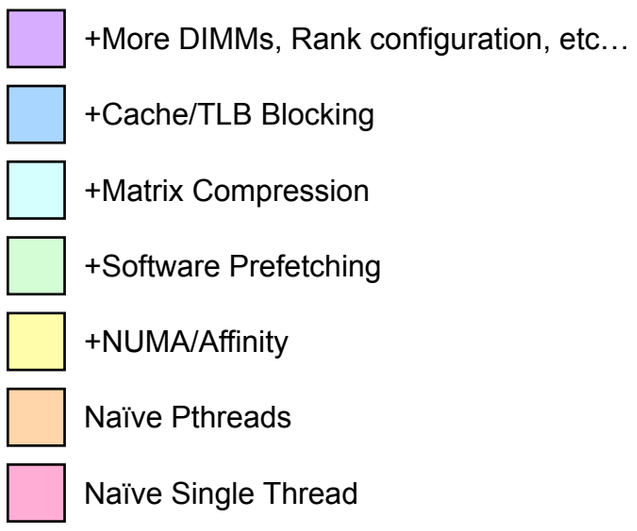


AMD Opteron



Sun Niagara2

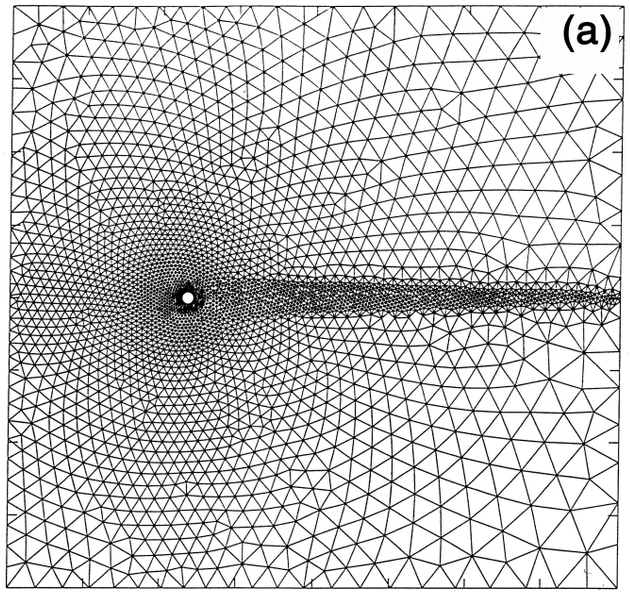


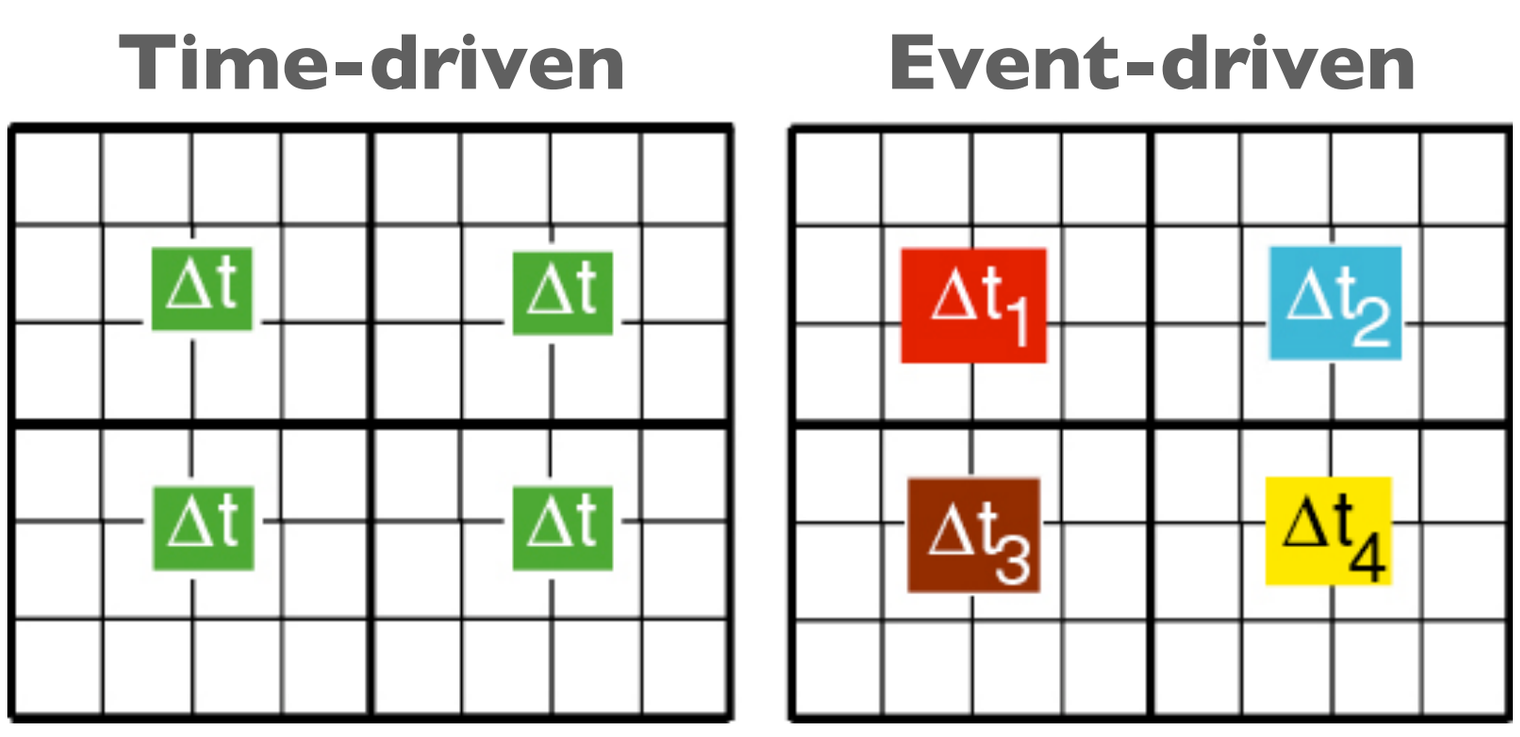


Note: Most complex architecture needed most tuning yet achieved lowest performance

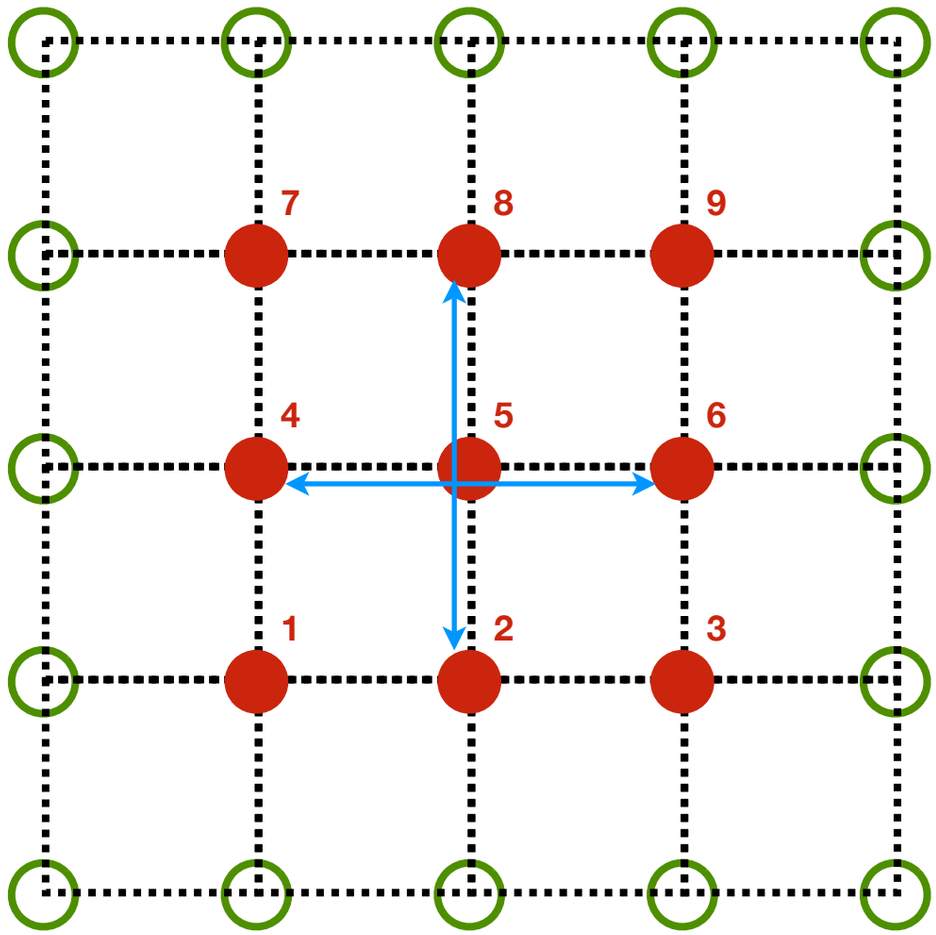
- “Wildly” asynchronous algorithms

Approach: Asynchronous event-driven algorithms



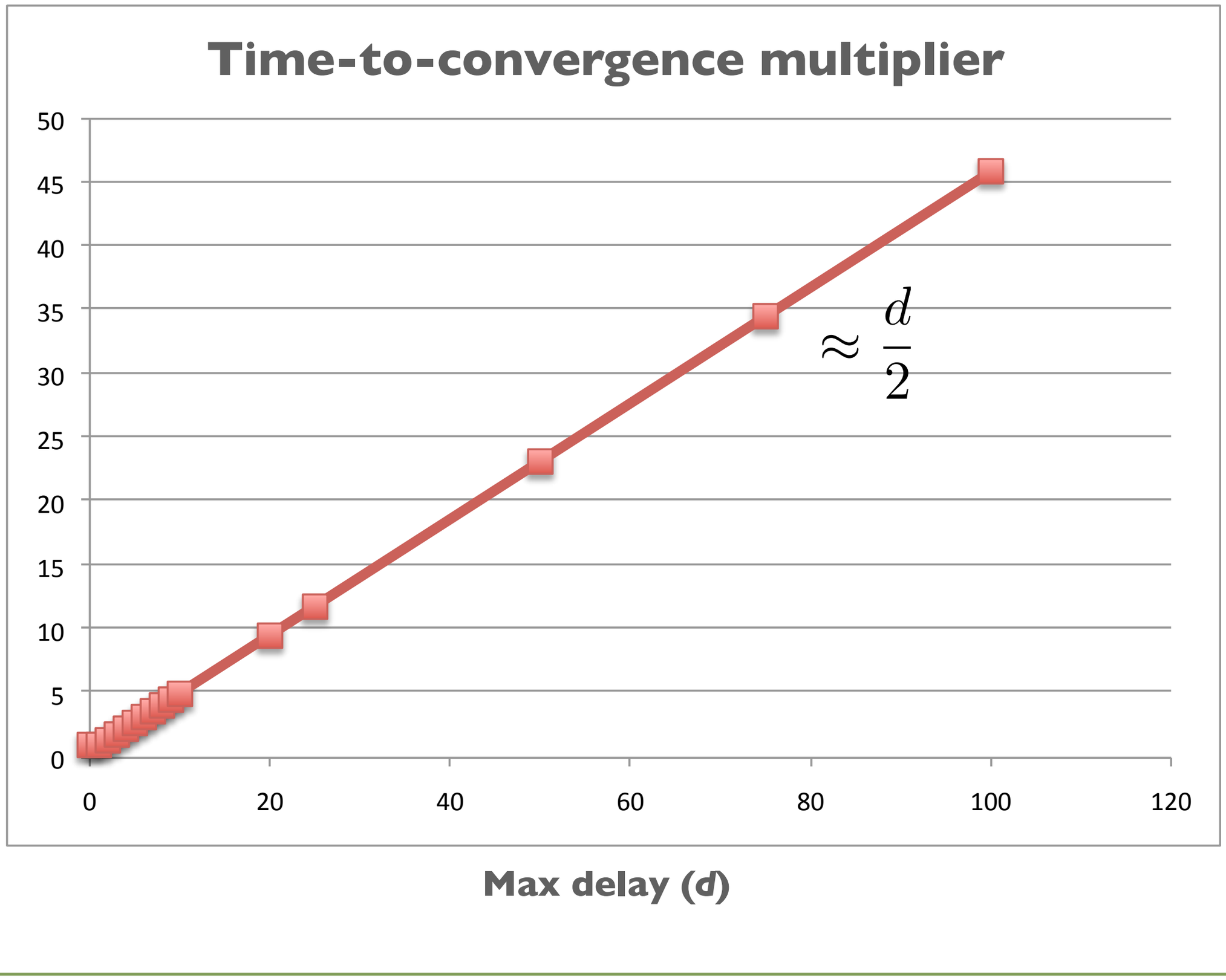


Source: Karimibadi, et al. *(2007)*



Jacobi update (weighted average):

$$u_{i,j}^{t+1} = \frac{1}{4}\left(u_{i-1,j}^{t} + u_{i+1,j}^{t} + u_{i,j-1}^{t} + u_{i,j+1}^{t} + h^2 f_{i,j}\right)$$


Time-to-convergence multiplier
50
45
40
35
30
25
20
15
10
5
0
0
20
40
60
80
100
120
$\approx \frac{d}{2}$
Max delay (d)

- Trends
  - DAG-based
  - Data-centric
  - Wildly asynchronous

- Questions: Productive interaction?
  - “Enrich” scheduling interfaces?
  - Expose tuning knobs?
  - Optimize software to minimize power?