Scaling Dense LU Factorization in Charm++

Jonathan Lifflander

Parallel Programming Laboratory

University of Illinois

Contributors

- PPLers
  - Jonathan Lifflander
  - Phil Miller
  - Ramprasad Venkataraman
  - Anshu Arya
- Oak Ridge National Laboratory
  - Terry Jones

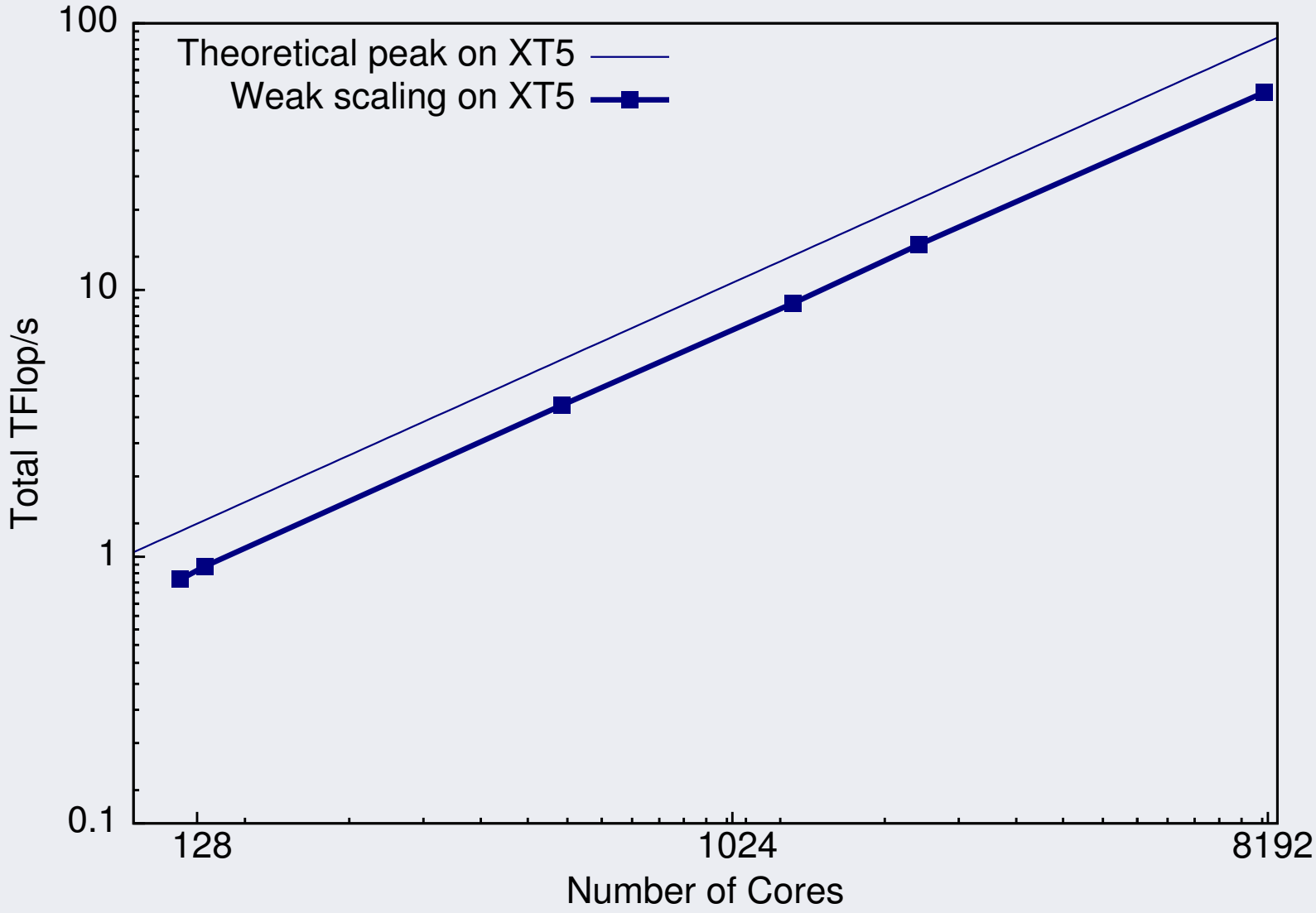
Outline

- Overall results
- The asynchronous "dataflow" parallel algorithm
- Techniques
  - Enabling partial synchrony using *exclusive scheduling classes*
  - Limiting network contention (see paper)
- Mapping
- Theoretical work on schedules that enable deeper lookahead while avoiding deadlock or memory exhaustion (see paper)

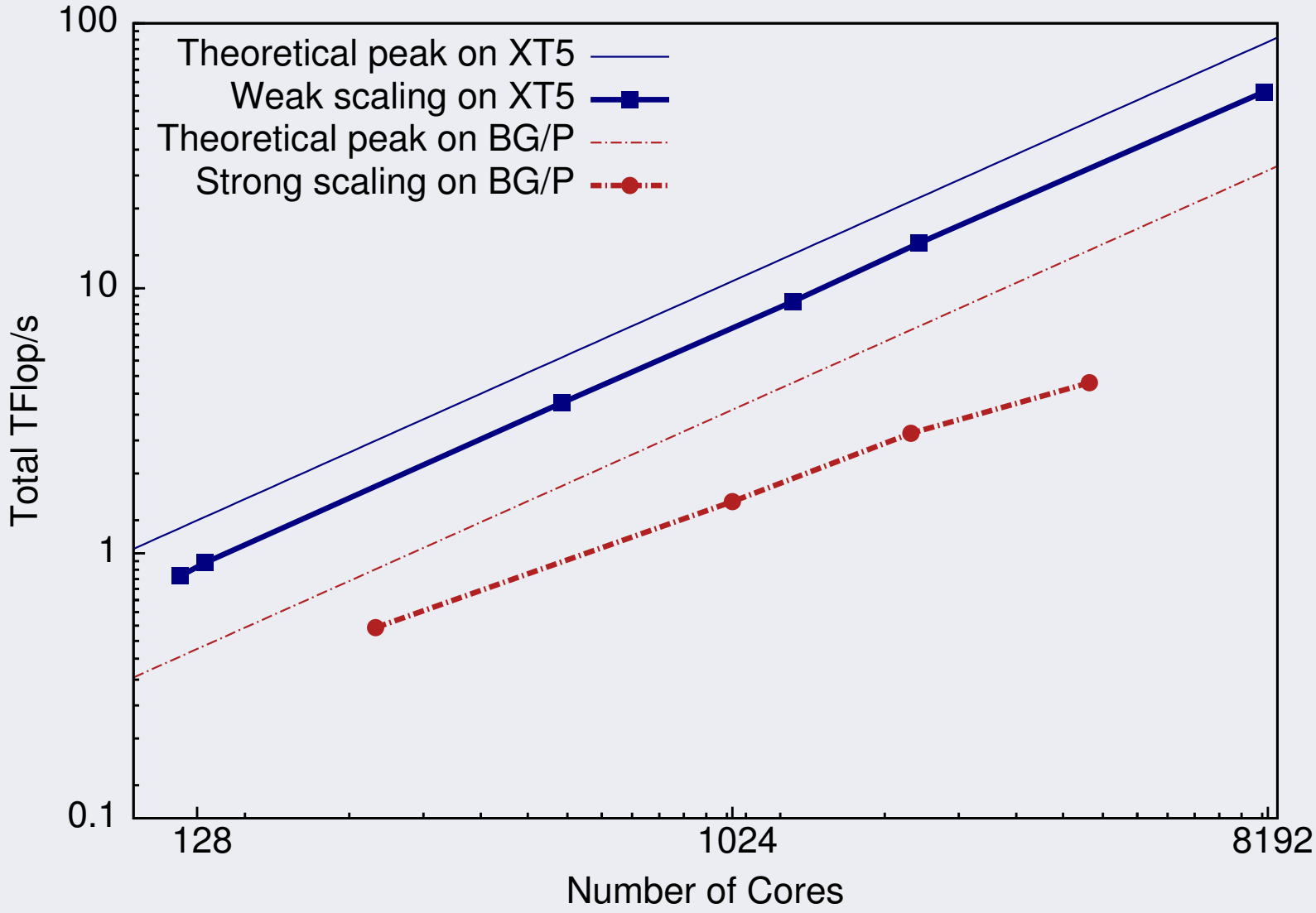
Our Dense LU Solver

- Solves a system of equations
  - Performs LU factorization
    - Crout's algorithm
    - Partial pivoting for numerical stability
  - Validates the solution by calculating the scaled residual

Overall Results



Overall Results



Overall Results

Library	Peak	Cores	n	Architecture
UPC[1]	76.6%	512	229K	XT3
DPLASMA[2]	58.3%	3072	454K	XT5
ScalaPack[3]	59%	3072	454K	XT5
HPCC[4] HPL	65.8%	224220	3936K	XT5
Jaguar top500	75.5%	224162	5474K	XT5
CharmLU	67.4%	2112	528K	XT5

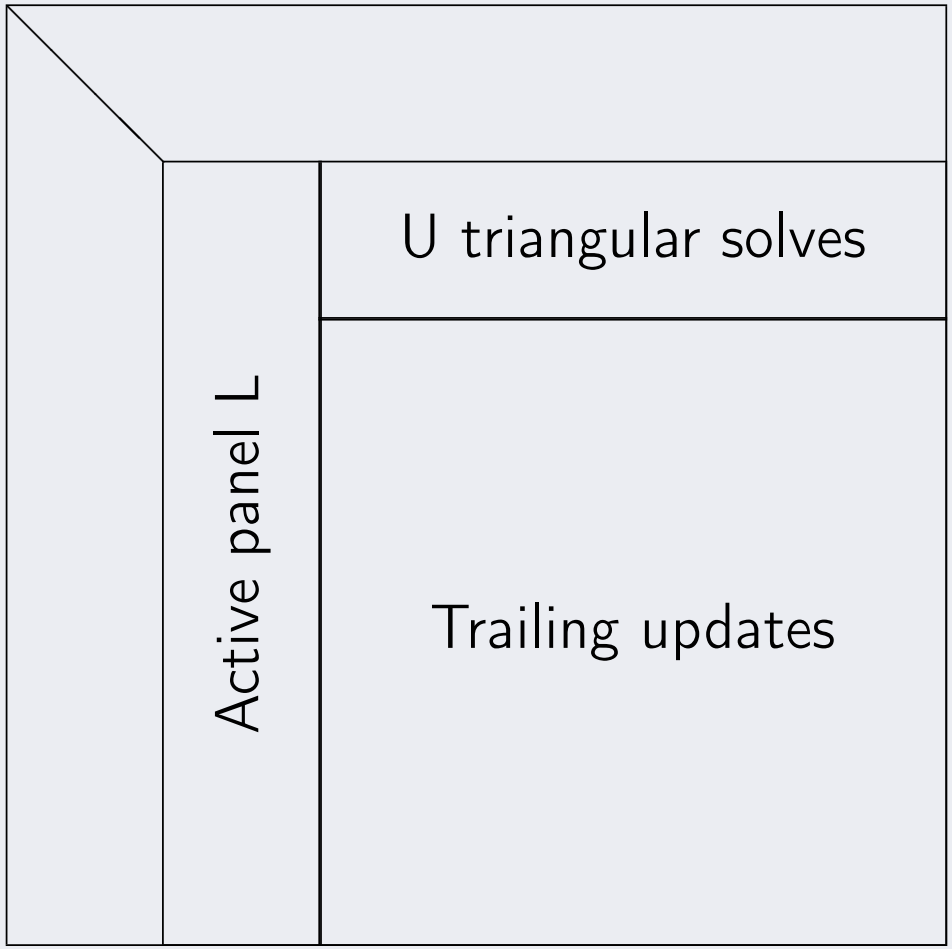
[1] P. Husbands and K. Yelick. Multi-threading and one-sided communication in parallel lu factorization. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, pages 1–10, New York, NY, USA, 2007. ACM

[2] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. Yarkhan, and J. Dongarra. Distibuted [sic.] dense numerical linear algebra algorithms on massively parallel architectures: Dplasma. Technical Report UT-CS-10-660, University of Tennessee, September 2010

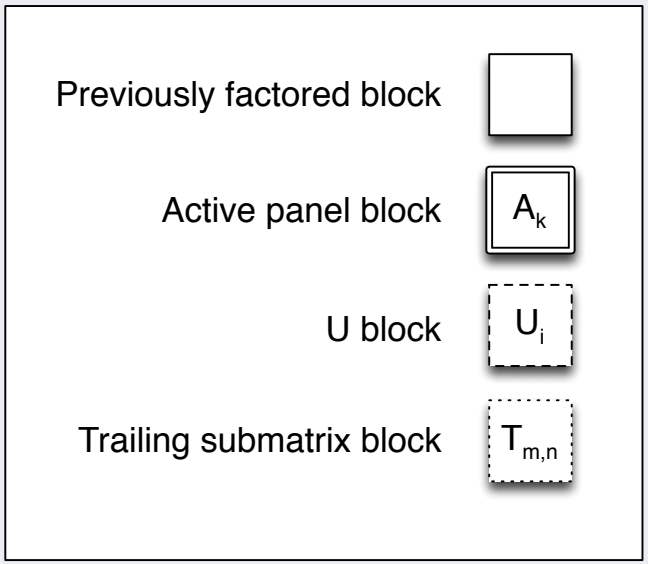
[3] J. Choi, J. Dongarra, and D. Walker. The Design of Scalable Software Libraries for Distributed Memory Concurrent Computers. In H. Siegel, editor, *Proc. Eighth International Parallel Processing Symposium*. IEEE Computer Society Press, April 1994

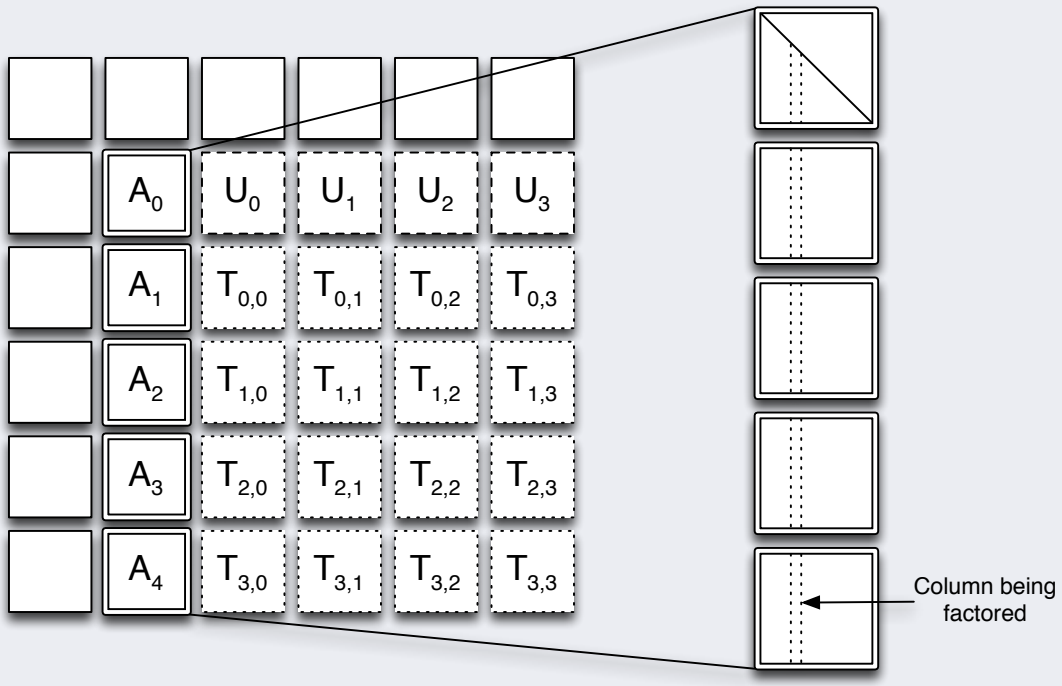
[4] B. Bland. Hpc challenge class i award g-hpl winning submission, 2010

LU Regions

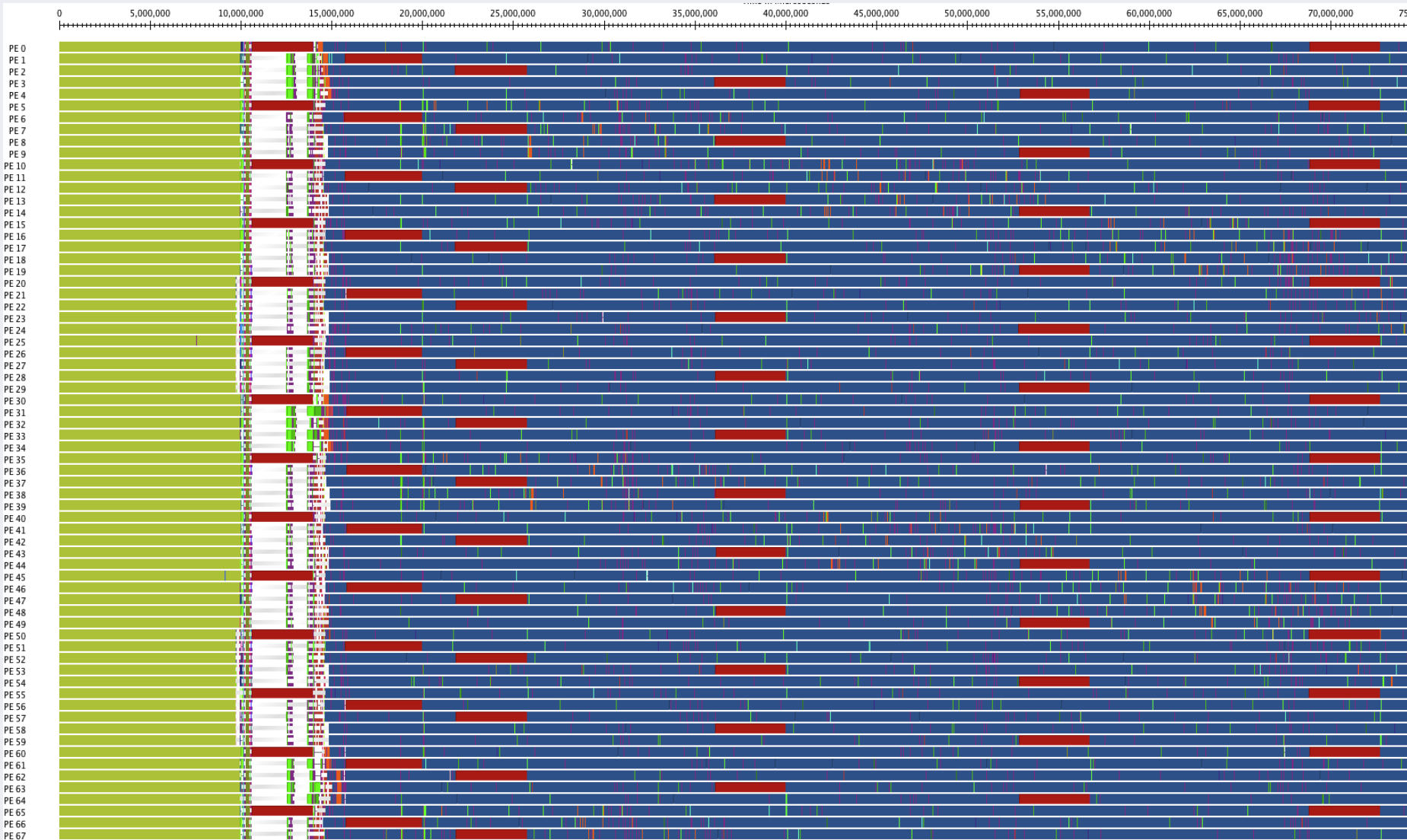


LU "Dataflow" Parallel Algorithm

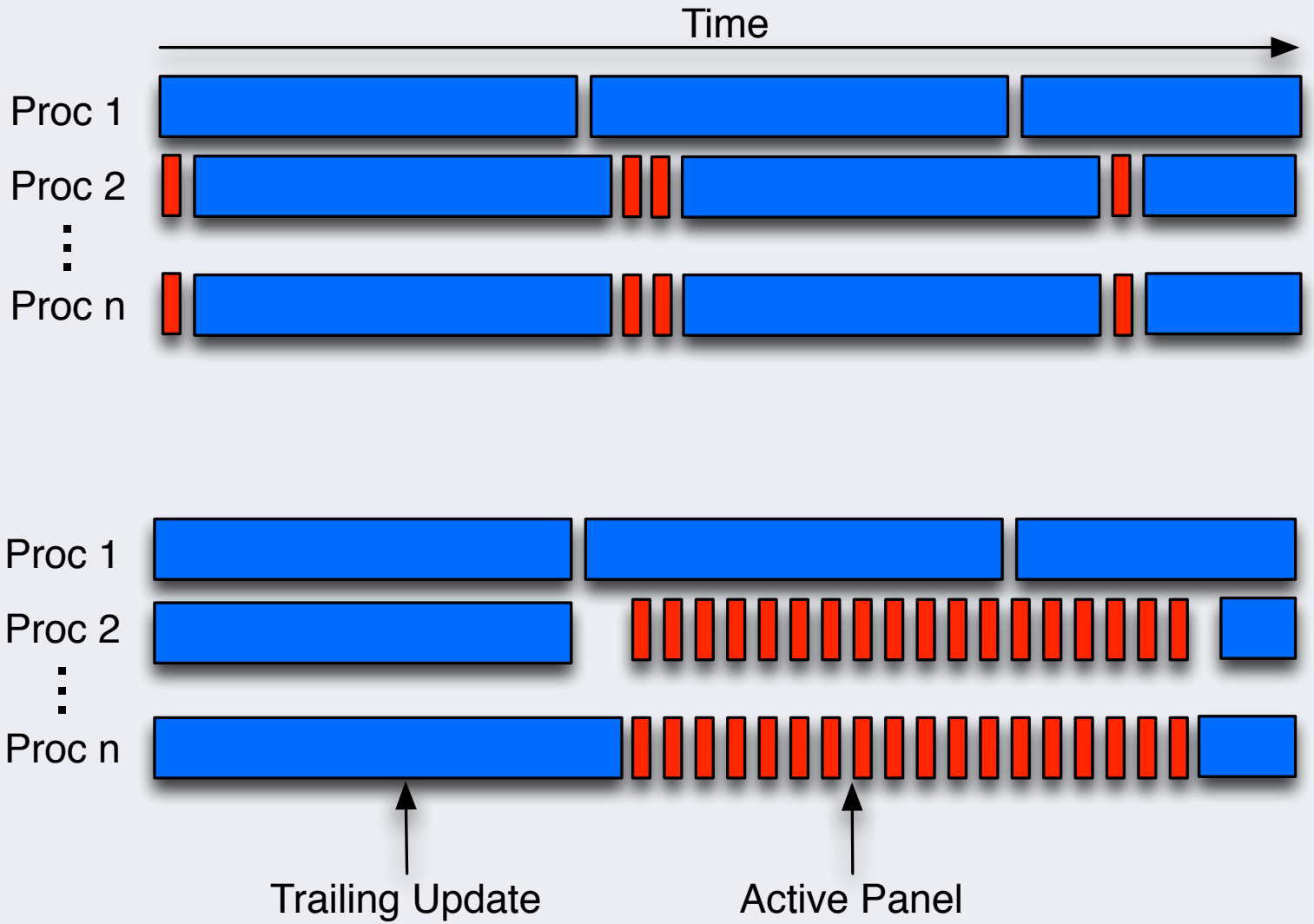




LU Timeline on XT5



Exclusive Scheduling Classes



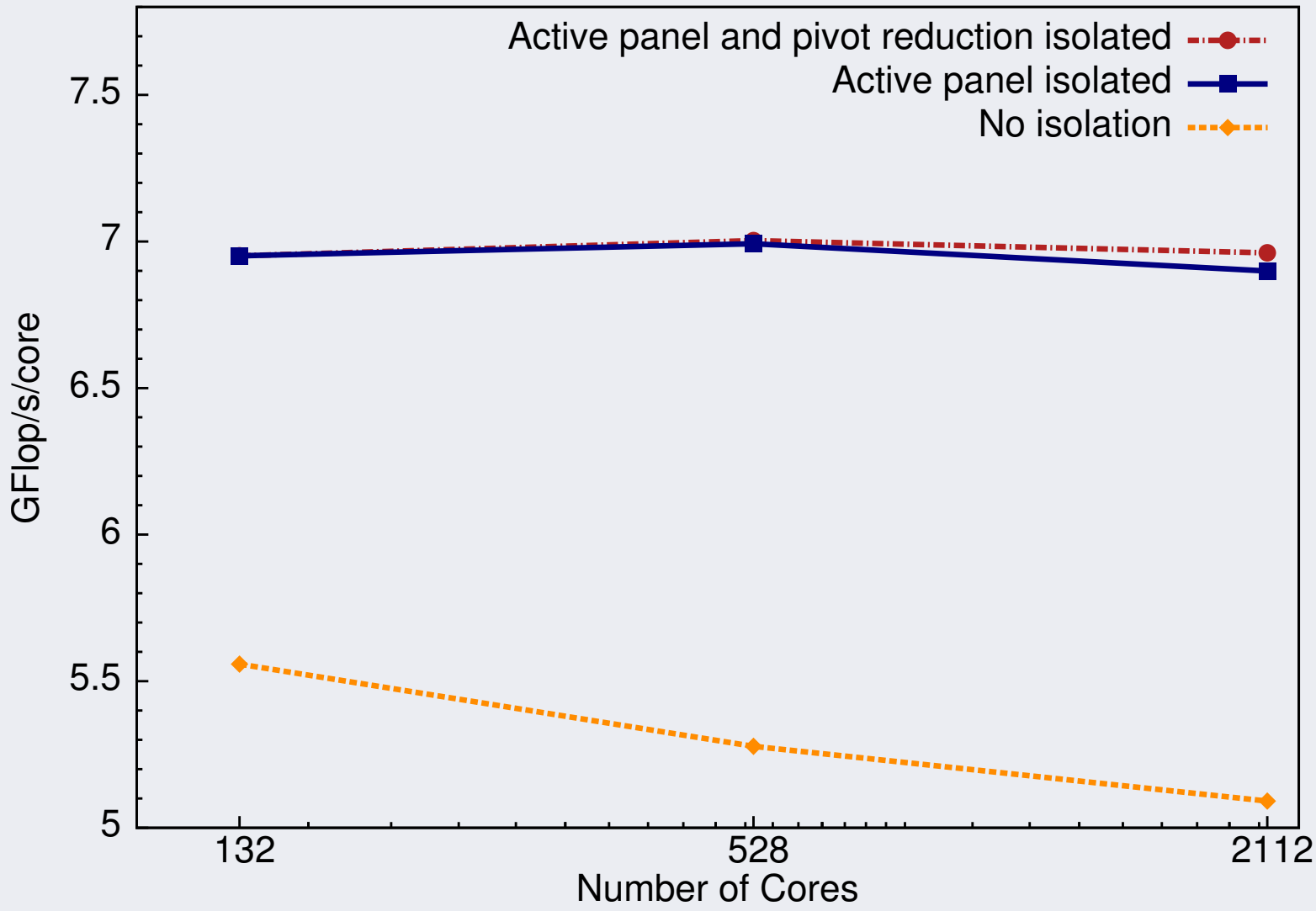
Possible Solutions

- Interrupts/Preemption
- Polling
- RDMA

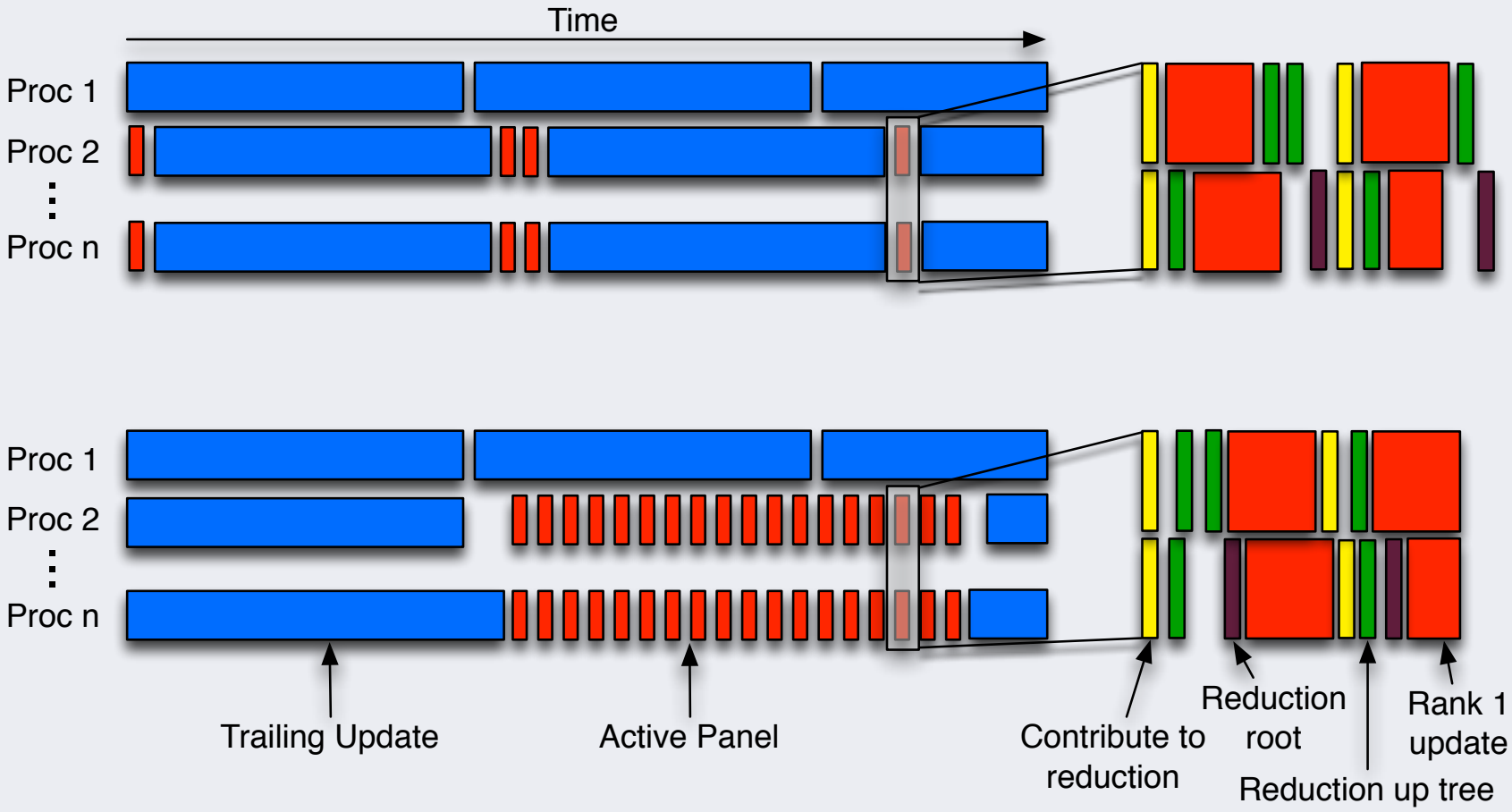
Our Solution

- *Exclusive scheduling classes*
  - During execution the scheduler is set to some exclusive scheduling class
  - All other work units of lower classes in the local queue are held back
  - Work units are stratified by the application into these classes
  - The application determines when to switch scheduling classes
- Implemented in LU
  - Uses application information to know when it is safe to transition to a different class

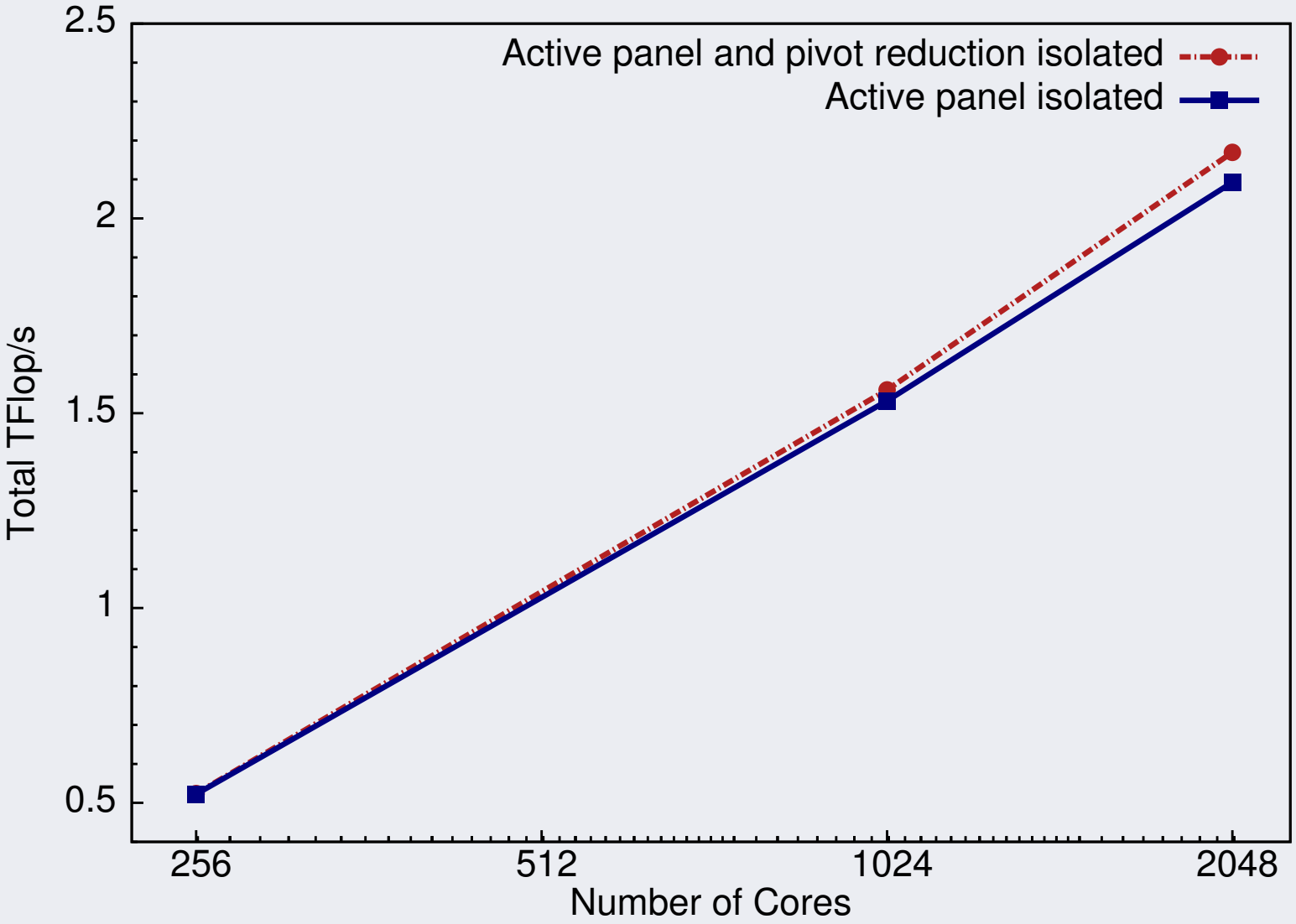
Exclusive Scheduling Classes



Exclusive Scheduling Classes



Exclusive Scheduling Classes



Traditional Block-Cyclic Mapping

$y \rightarrow$

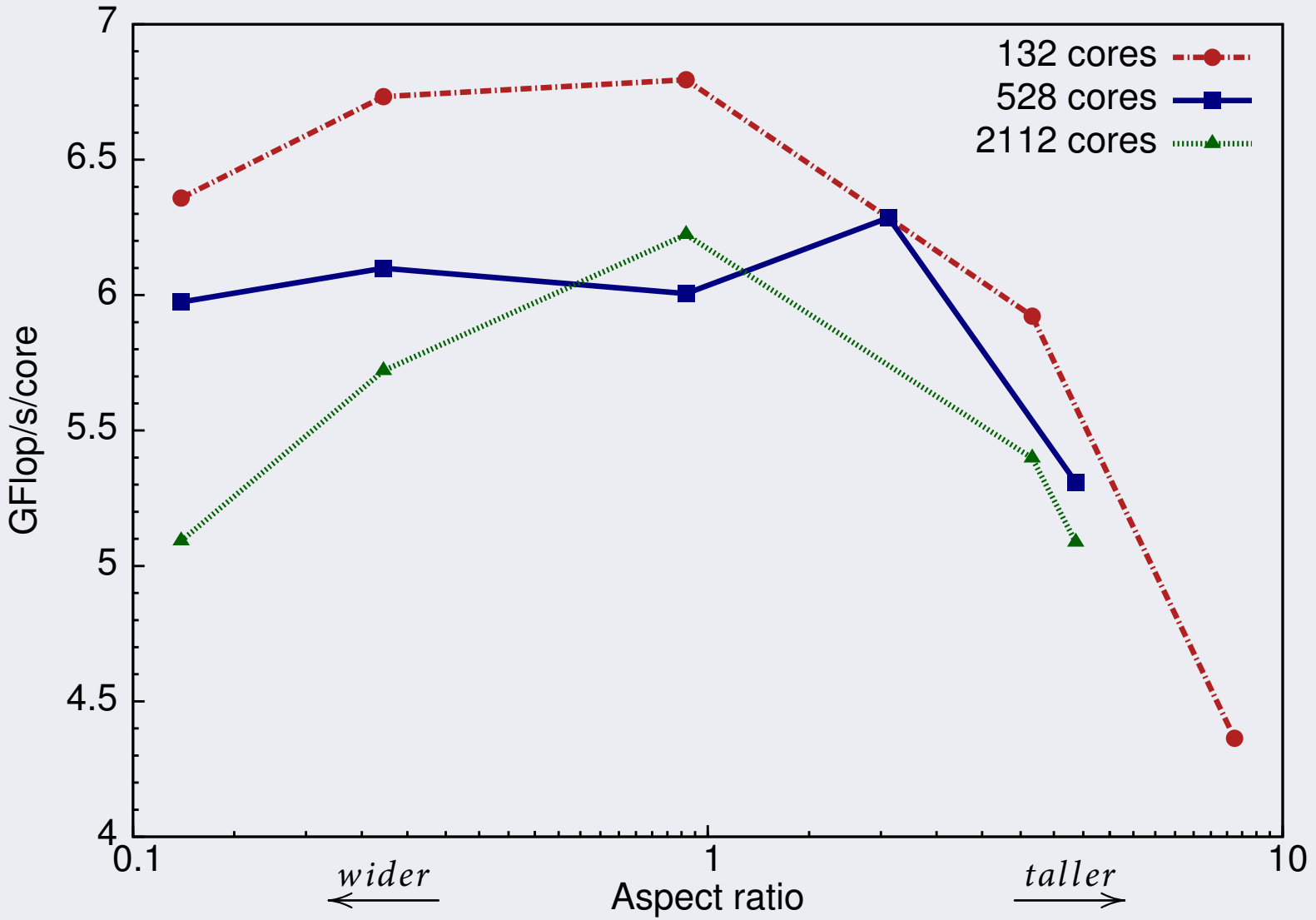
$x \downarrow$

0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8
0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8
0	1	2	0	1	2	0	1	2
3	4	5	3	4	5	3	4	5
6	7	8	6	7	8	6	7	8

Tall tile

0	1	2	3	0	1	2	3	0
4	5	6	7	4	5	6	7	4
8	9	10	11	8	9	10	11	8
12	13	14	15	12	13	14	15	12
16	17	18	19	16	17	18	19	16
20	21	22	23	20	21	22	23	20
0	1	2	3	0	1	2	3	0
4	5	6	7	4	5	6	7	4
8	9	10	11	8	9	10	11	8

Various Aspect Ratios



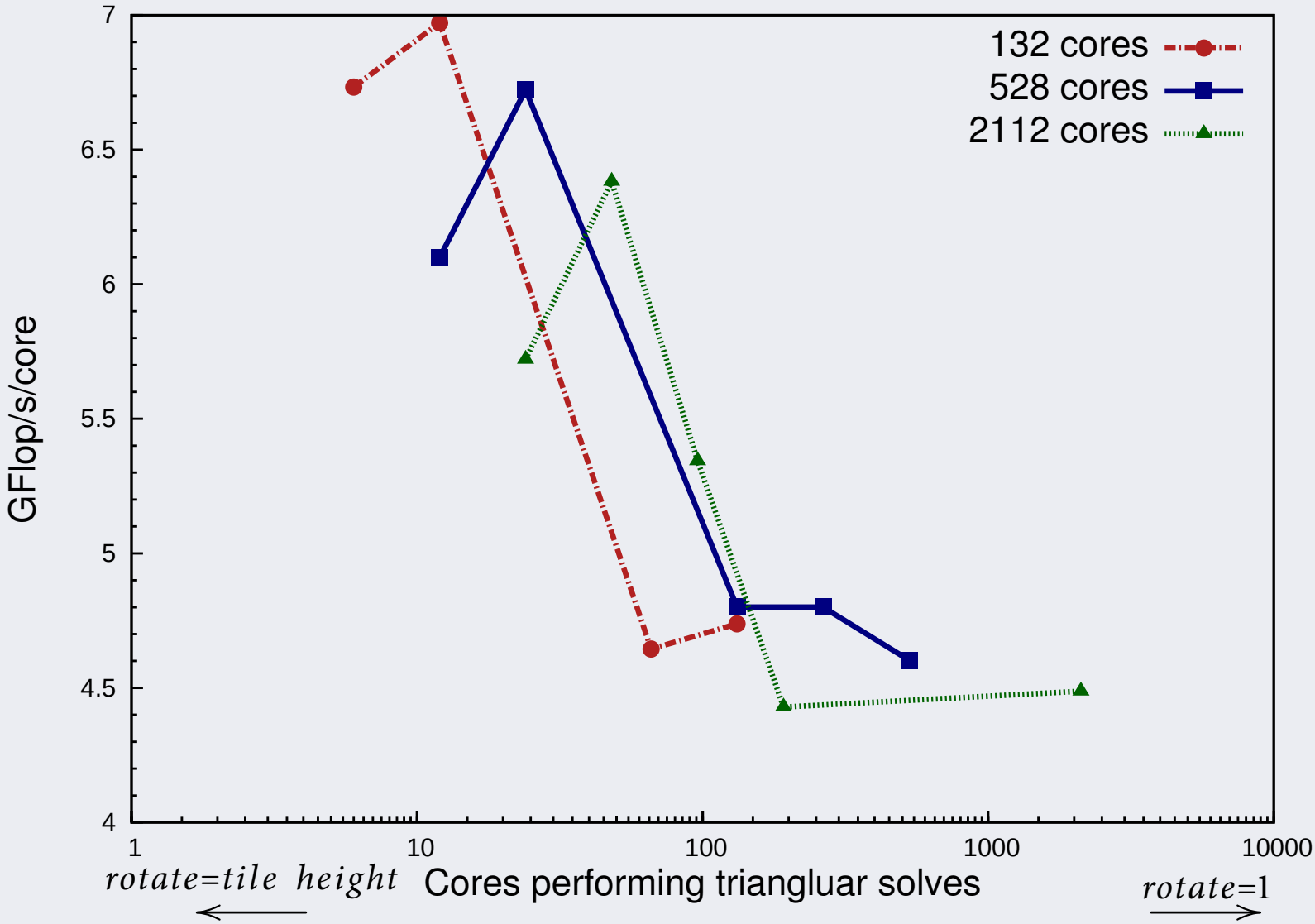
Rotation

- To recover U parallelism decreased by using a tall tile
- Rotate tile in x direction by r rows for each new tile across the matrix in the y direction

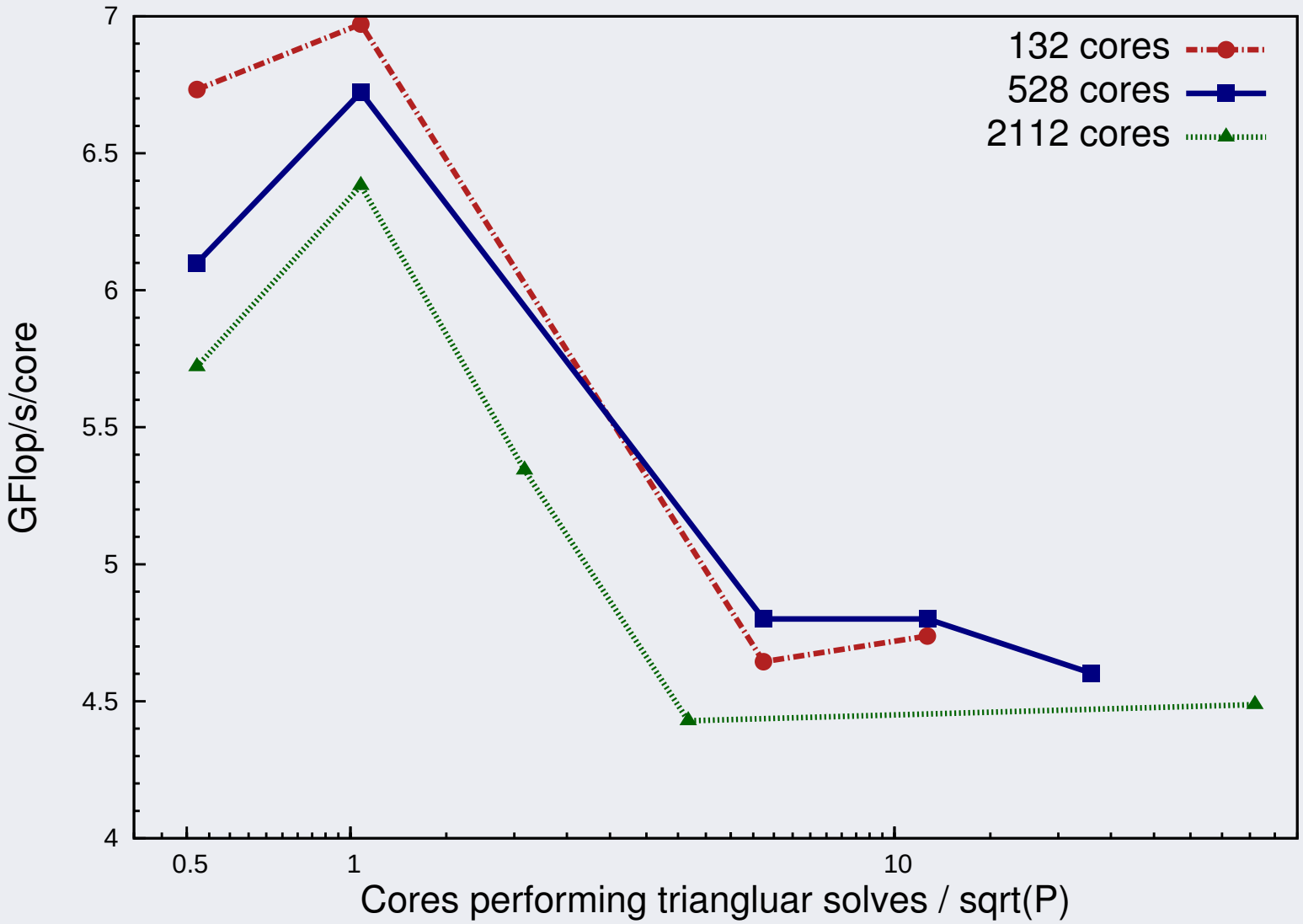
Rotation (rotate = 2)

0	1	2	3	8	9	10	11	16
4	5	6	7	12	13	14	15	20
8	9	10	11	16	17	18	19	0
12	13	14	15	20	21	22	23	4
16	17	18	19	0	1	2	3	8
20	21	22	23	4	5	6	7	12
0	1	2	3	8	9	10	11	16
4	5	6	7	12	13	14	15	20
8	9	10	11	16	17	18	19	0

Rotation


132 cores
528 cores
2112 cores
GFlop/s/core
7
6.5
6
5.5
5
4.5
4
1
10
100
1000
10000
rotate=tile height
Cores performing triangluar solves
rotate=1

Rotation


132 cores
528 cores
2112 cores
GFlop/s/core
7
6.5
6
5.5
5
4.5
4
0.5
1
10
Cores performing triangluar solves / sqrt(P)

Tall tile

0	1	2	3	0	1	2	3	0
4	5	6	7	4	5	6	7	4
8	9	10	11	8	9	10	11	8
12	13	14	15	12	13	14	15	12
16	17	18	19	16	17	18	19	16
20	21	22	23	20	21	22	23	20
0	1	2	3	0	1	2	3	0
4	5	6	7	4	5	6	7	4
8	9	10	11	8	9	10	11	8

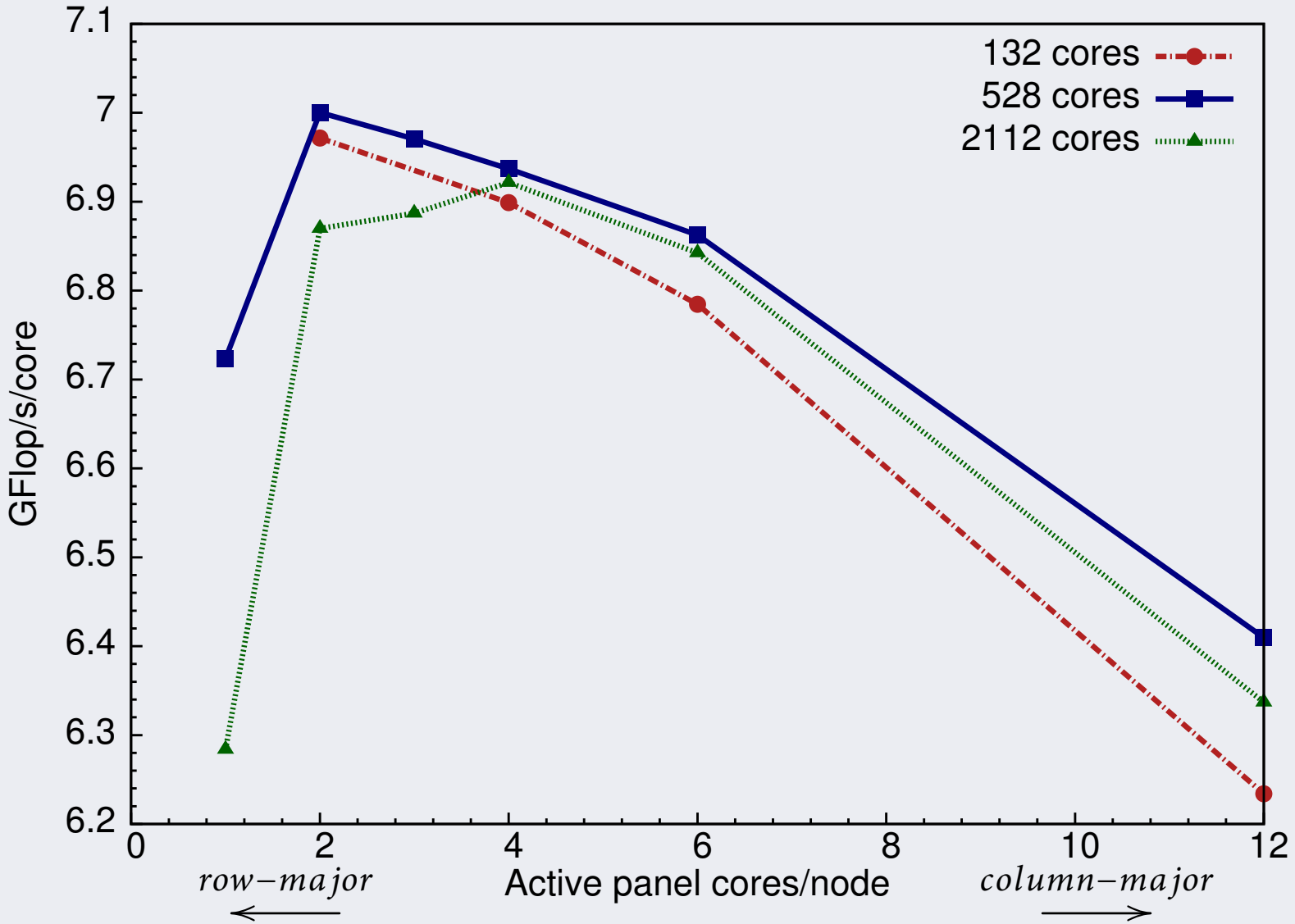
Striding

- Gain the advantages of active panel locality without creating excessive memory contention
- Generalization of controlling whether the tile is column- or row-major

Striding (stride = 2)

0	1	12	13	4	5	16	17	8
2	3	14	15	6	7	18	19	10
4	5	16	17	8	9	20	21	0
6	7	18	19	10	11	22	23	2
8	9	20	21	0	1	12	13	4
10	11	22	23	2	3	14	15	6
0	1	12	13	4	5	16	17	8
2	3	14	15	6	7	18	19	10
4	5	16	17	8	9	20	21	0

Striding


7.1
7
6.9
6.8
6.7
6.6
6.5
6.4
6.3
6.2
GFlop/s/core
132 cores
528 cores
2112 cores
0
2
4
6
8
10
12
row−major
Active panel cores/node
column−major

Outline

- Overall results
- The asynchronous "dataflow" parallel algorithm
- Techniques
  - Enabling partial synchrony using *exclusive scheduling classes*
  - Limiting network contention (see paper)
- Mapping
- Theoretical work on schedules that enable deeper lookahead while avoiding deadlock or memory exhaustion (see paper)

Questions?