A Fast Elliptic Solver based on Hierarchical Schur Complements in Cyclic Reduction

Gustavo Chavez, Hatem Ltaief, George Turkiyyah, Rio Yokota & David Keyes

Extreme Computing Research Center
KAUST

Outline

- **Niche**
  - applications where iterative methods are hard to tune and direct methods have non-optimal complexity
  - emerging fine-grained, SIMD-style architectures
- **Hierarchical matrices**
- **Cyclic reduction**
- **Accelerated cyclic reduction**
- **Exact solver for 2D**
  - numerical experiments
- **Preconditioner for 3D**
  - preliminary numerical experiments
- **Future directions**

Context

- **It is hard to beat**
  - **multigrid**
  - **domain decomposition**

  for weak scaling to a million nodes, SPMD-style
  - **with memory resources (bandwidth and capacity) proportional to the number of nodes**
  - **one MPI process per node**
- **For strong scaling to hundreds of cores *per node* and reduced memory resources *per core*, we may need alternatives**
  - **many-core subdomain solver for stand-alone, for integration in outer DD method, or perhaps for extension to SPMD weak scaling**
  - **need not have lowest flops, but should exploit SIMD styles to have low energy and/or execution time**

Overview

- **Proposed method presented from an algebraic perspective, but its effectiveness is tied to the same principles as solvers and preconditioners based on domain decomposition**
  - **parallel performance from divide-and-conquer to create suitable subproblems**
    - **although typically by recursion, rather than by *a priori* partitioning**
  - **convergence from the decay properties of elliptic Green's functions**
    - **although typical tools are related to rank reduction rather than condition number bounds**
- **Low-rank ideas have not been prominent in our DD meetings**
  - **however, they are rapidly penetrating the linear algebra mainstream**
- **Can compete directly with subspace projection methods in certain niches (Helmholtz, high contrast)**
- **Can plug in to subspace projection methods as component**
- **Tunable on spectrum between direct and iterative**

Key tool: hierarchical matrices

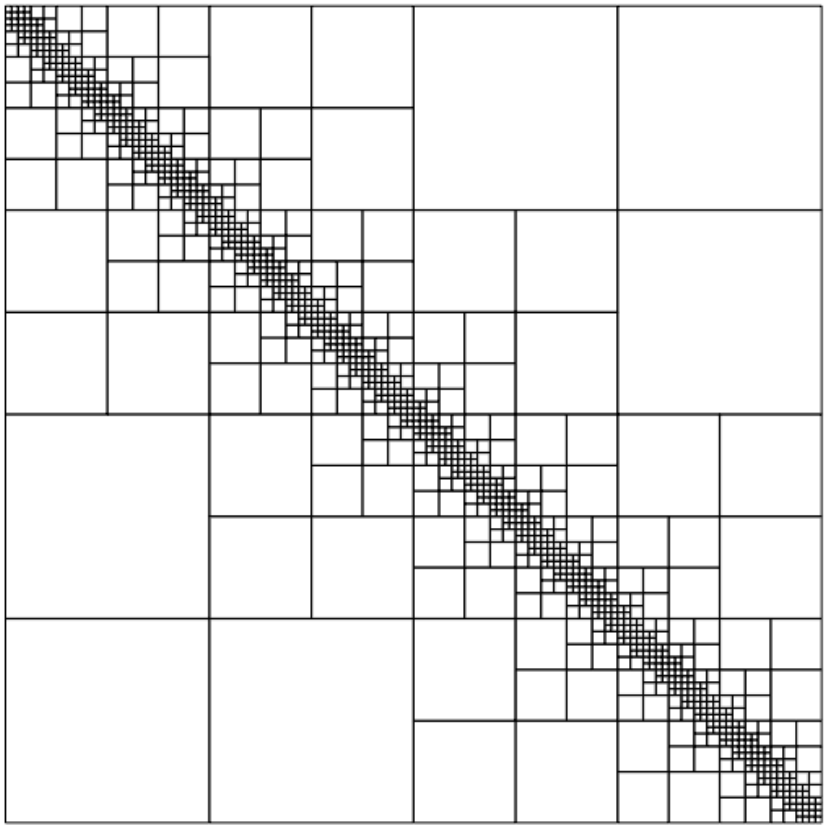
- [Hackbusch, 1999] **: off-diagonal blocks of typical differential and integral operators have low effective rank**
- **By exploiting low rank, k , memory requirements and operation counts approach optimal in matrix dimension n**
  - **polynomial in k**
  - **lin-log in n**
  - **constants carry the day**
- **Such hierarchical representations navigate a compromise**
  - **fewer blocks of larger rank ("weak admissibility") or**
  - **more blocks of smaller rank ("strong admissibility")**

Example: 1D Laplacian

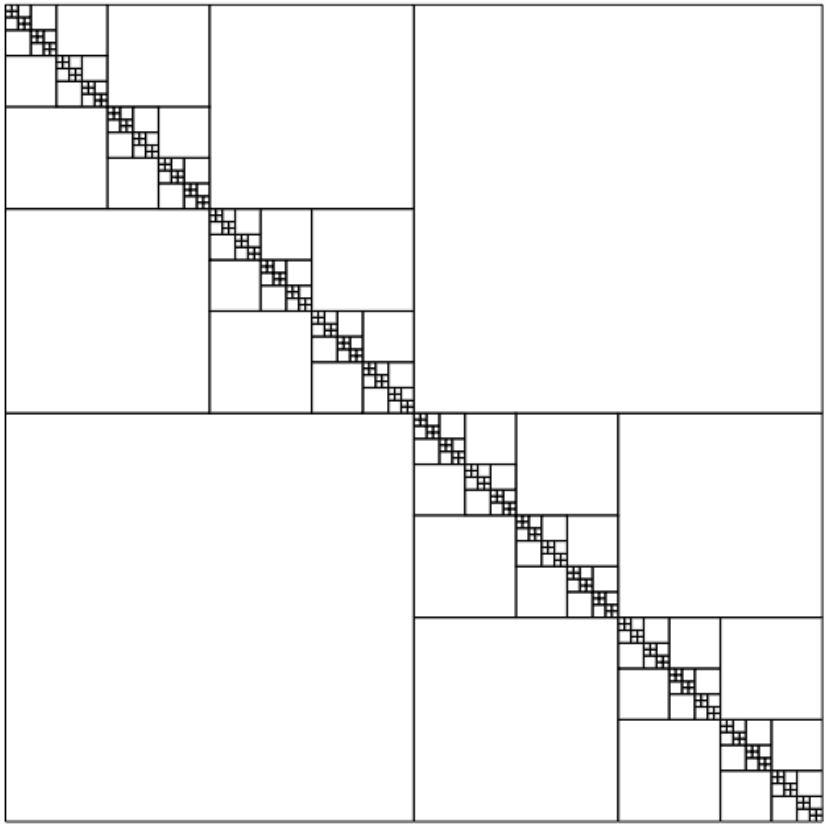
$$A = \left[\begin{array}{ccc|cccc} 2 & -1 & & & & & \\ -1 & 2 & -1 & & & & \\ & -1 & 2 & -1 & & & \\ \hline & & -1 & 2 & -1 & & \\ & & & -1 & 2 & -1 & \\ & & & & -1 & 2 & -1 \\ & & & & & -1 & 2 \end{array}\right] \leftrightarrow = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 0 & 0 \end{bmatrix}$$

$$A^{-1} = \frac{1}{8} \times \left[\begin{array}{ccc|cccc} 7 & 6 & 5 & 4 & 3 & 2 & 1 \\ 6 & 12 & 10 & 8 & 6 & 4 & 2 \\ 5 & 10 & 15 & 12 & 9 & 6 & 3 \\ \hline 4 & 8 & 12 & 16 & 12 & 8 & 4 \\ 3 & 6 & 9 & 12 & 15 & 10 & 5 \\ 2 & 4 & 6 & 8 & 10 & 12 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}\right] \leftrightarrow = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 4 & 3 & 2 & 1 \end{bmatrix}$$

“Standard(strong)” vs. “weak” admissibility



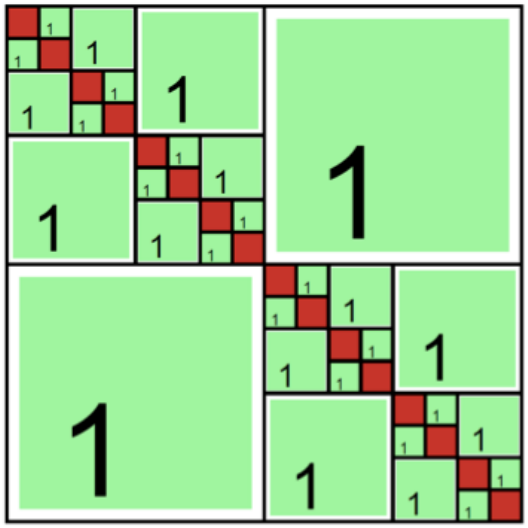
strong admissibility



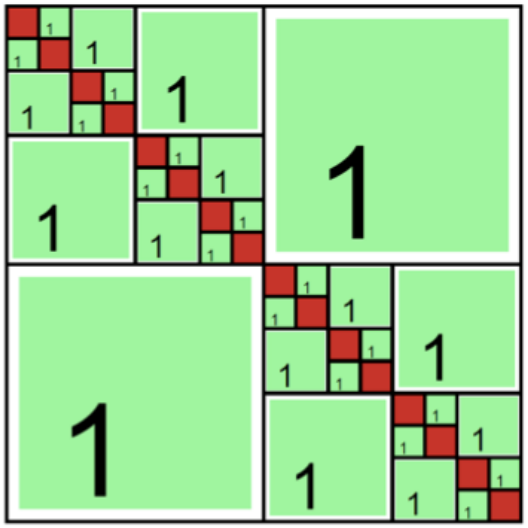
weak admissibility

After [Hackbusch, et al., 2003]

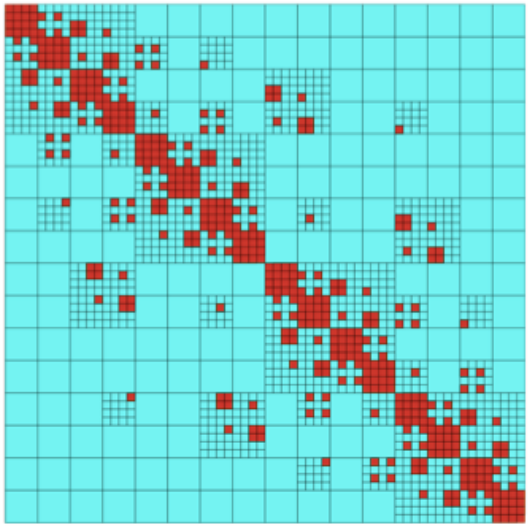
Graphical idiom for hierarchical structure



A



A^{-1}



more general system

Key idea

- **When dense blocks arise in "routine" matrix operations, replace them with hierarchical representations**
  - **e.g., the Schur complements in nested dissection**
- **Use high accuracy (high rank, but typically less than full) to build "exact" solvers**
- **Use low accuracy (low rank) to build preconditioners**

Hierarchical matrix algorithm flavors

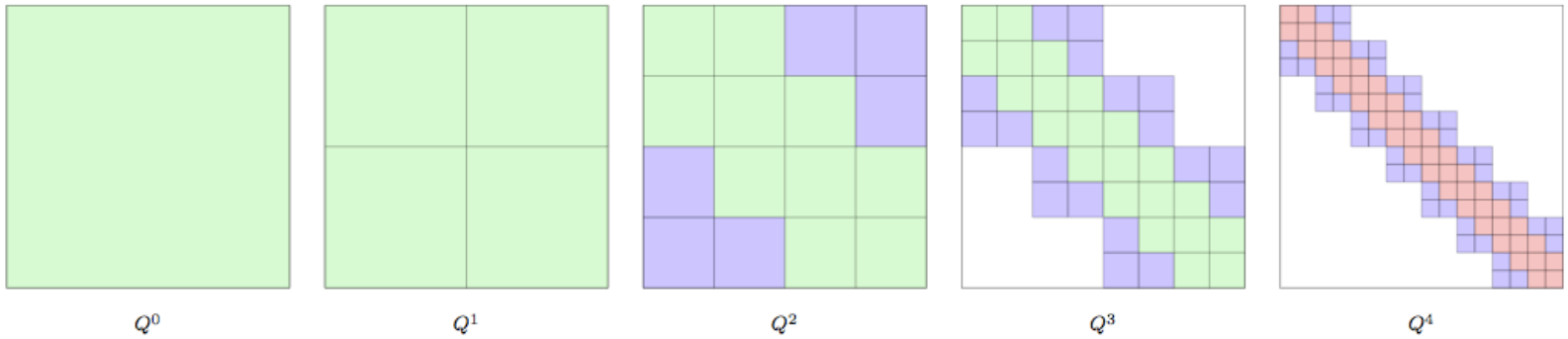
- [Hackbusch, 1999] **: H-matrices**
- [Hackbusch & Khoromskij, 2000] **: H^2-matrices**
- [Li *et al*., 2009] **: hierarchically semi-separable matrices (HSS)**
- [Ho, *et al*., 2012] **: recursive skeletonization**
- [Darve *et al*., 2013] **: hierarchical off-diagonal low-rank matrices (HODLR)**
- [Yokota *et al*., 2014] **: algebraic fast multipole**
- **…**

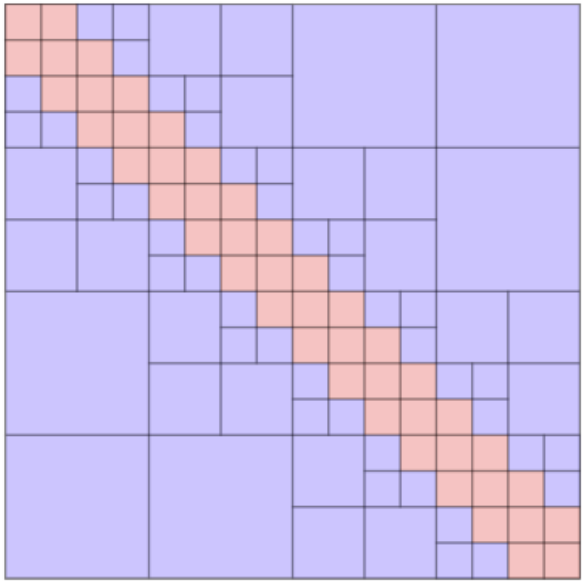
Hierarchical matrix software

Nested dissection, multifrontal, etc.

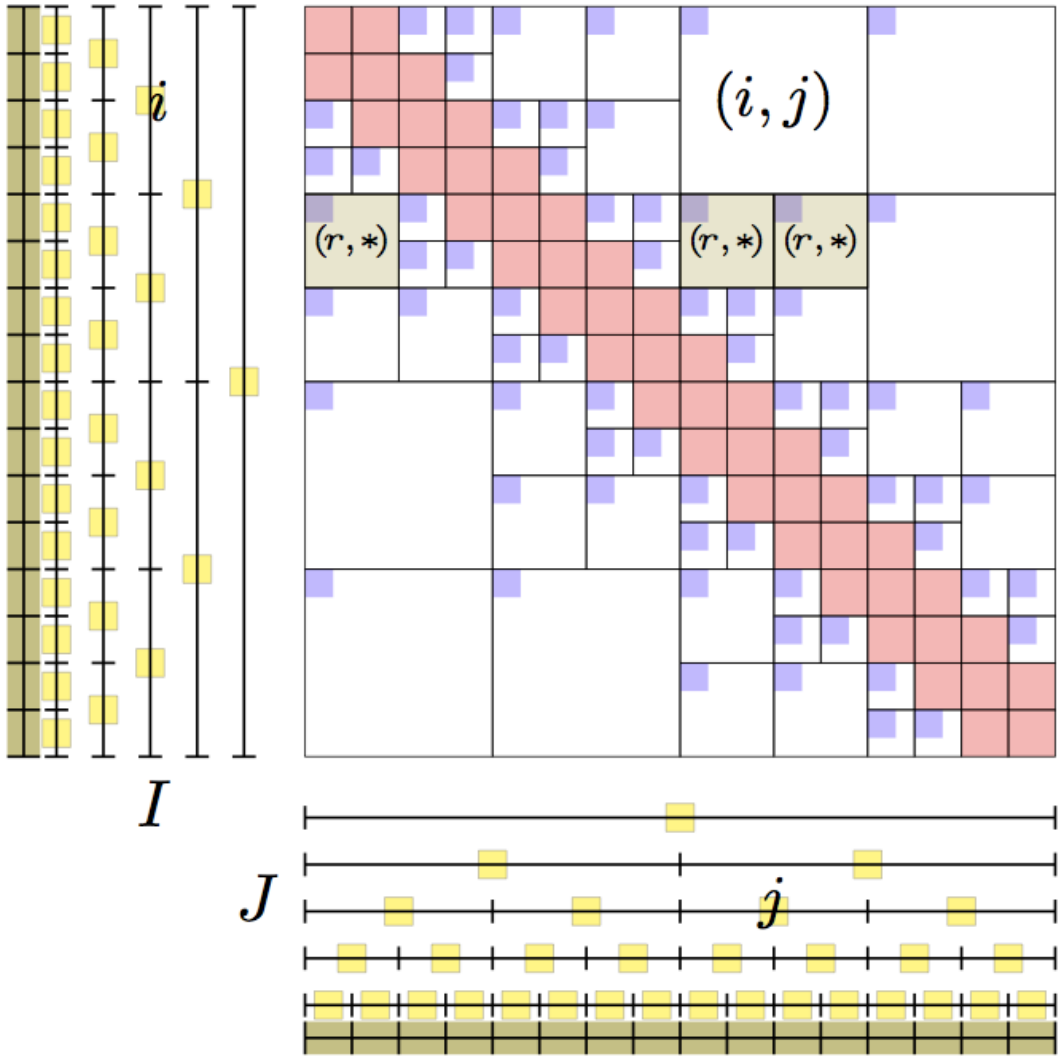
- [Kriemann, *et al*.] **: HLib**
- [Li, *et al*.] **: STRUMPACK**
- [Darve, *et al*.] **: HODLR**
- [Poulson, *et al*.] **: DMHM**
- [Amestoy, *et al*.] **: MUMPS**

Hierarchical structure



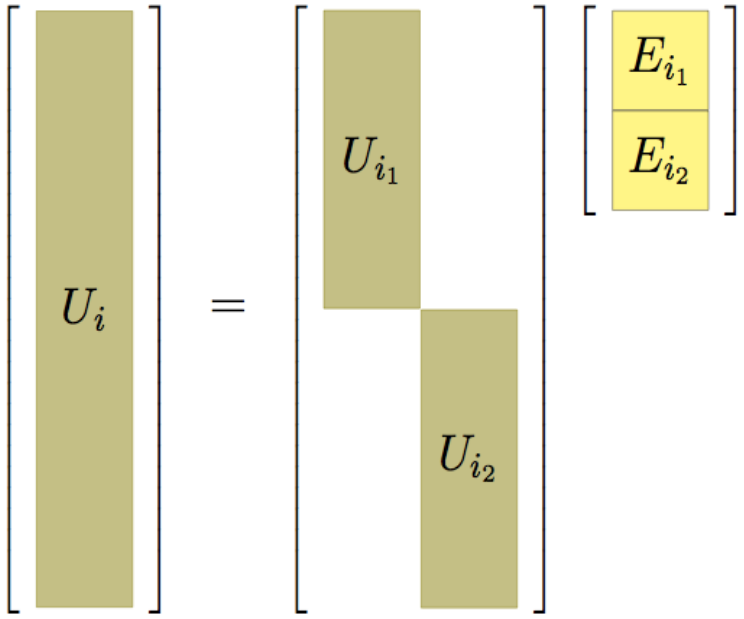


H^2 hierarchical structure



$$A_{ij} = U_i \, S_{ij} \, V_j^t$$

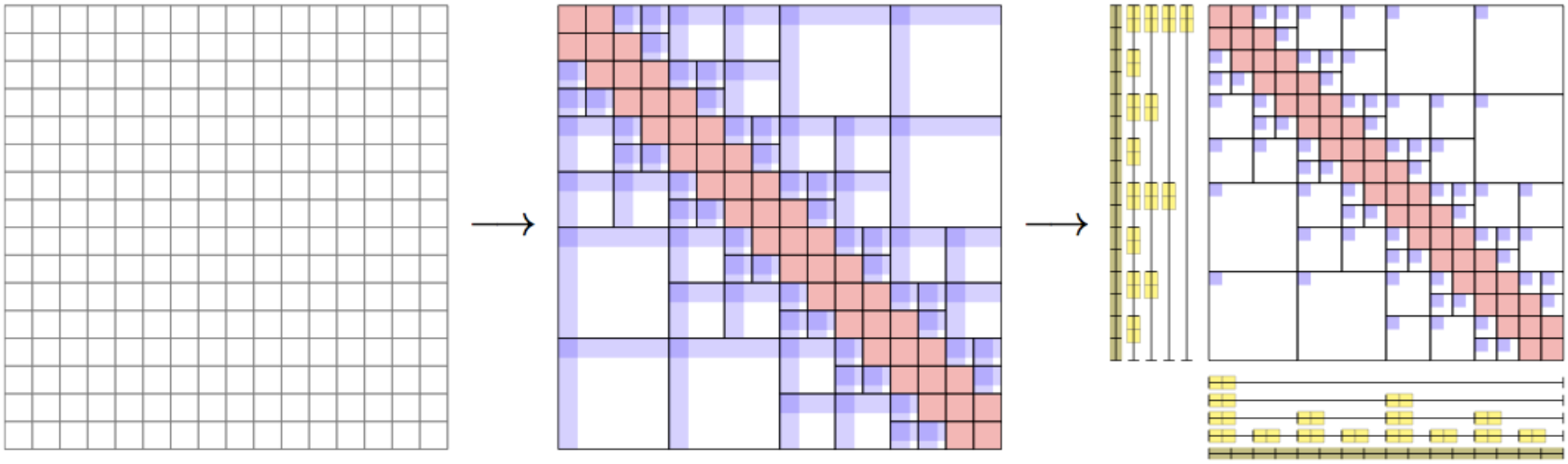
Recursive construction of nested bases



Data structure components:

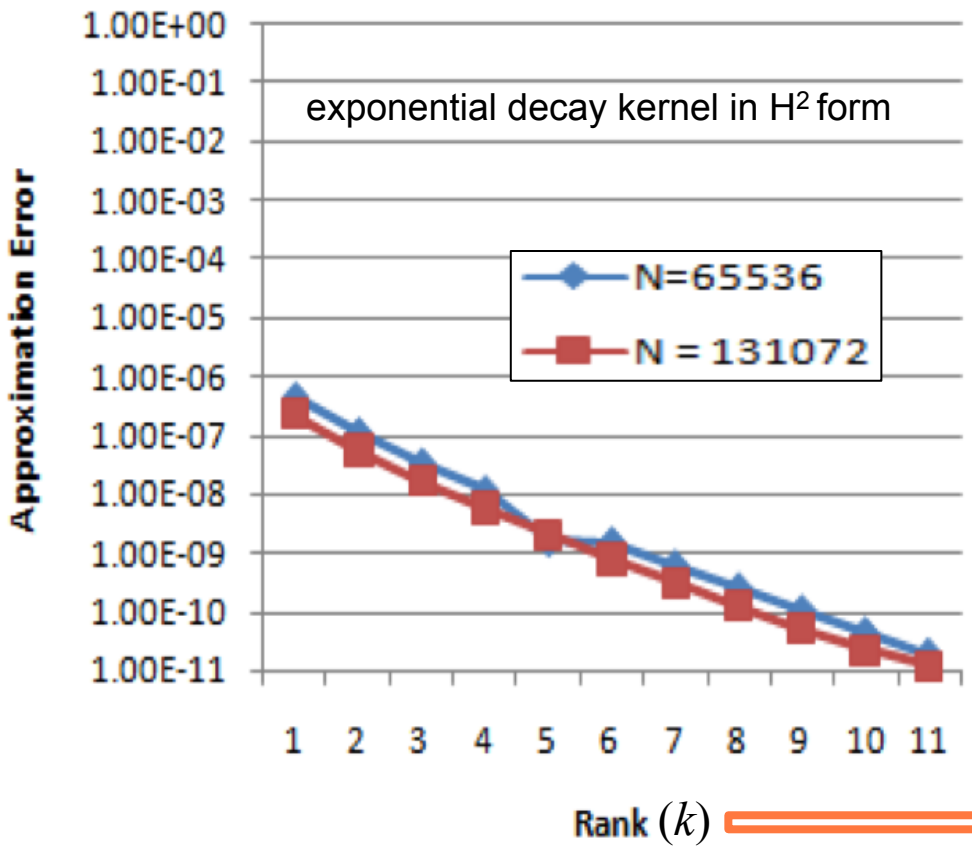
- dense blocks from original matrix
- diagonal blocks of low-rank matrices
- column bases and row bases for low-rank matrices
- maps between bases of different scales

Two-stage compression procedure



Accuracy and complexity with rank

Representation Complexity



Inversion Complexity

	Operations	Storage
Dense	$\mathcal{O}(n^3)$	$\mathcal{O}(n^2)$
$\mathcal{H}$	$\mathcal{O}(k^2 n \log^2 n)$	$\mathcal{O}(kn \log n)$

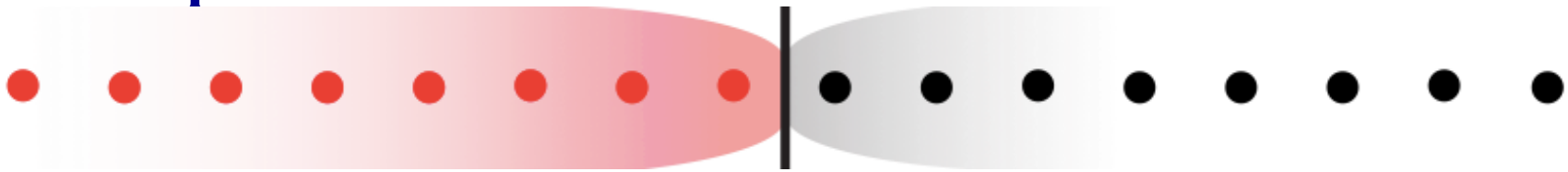
Cyclic reduction

- **"One of the most versatile and powerful methods ever created" – Golub**
- **Originally proposed for tridiagonal systems**
- **Based on two properties, preserved upon recursion**
  - **tridiagonal systems can be even-odd permuted to eliminate half of the unknowns**
  - **the Schur complement of a tridiagonal is still a tridiagonal**
- **In 2D and higher, called "block cyclic reduction"**
- **Though outer tridiagonal structure is preserved, solution of the inner solver governs the complexity**
- **For constant coefficients, FFTs can be used for $O(N \log N)$**
- **Otherwise, CR is a non-optimal form of Gauss elimination**

Cyclic reduction complementary to nested dissection

1D mesh

First step of nested dissection



First step of cyclic reduction

Preservation of tridiagonal structure on Schur complementation

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

$$= \left[\begin{array}{ccc|ccc} x & x & & & & \\ x & x & x & & & \\ & x & x & x & & \\ \hline & & x & x & x & \\ & & & x & x & x \\ & & & & x & x \end{array}\right]$$

$$S = A_{22} - A_{21}A_{11}^{-1}A_{12}$$

$$= \begin{bmatrix} x & x & \\ x & x & x \\ & x & x \end{bmatrix} - \begin{bmatrix} & & x \\ & & \\ & & \end{bmatrix} \begin{bmatrix} x & x & x \\ x & x & x \\ x & x & x \end{bmatrix} \begin{bmatrix} & & \\ & & \\ x & & \end{bmatrix}$$

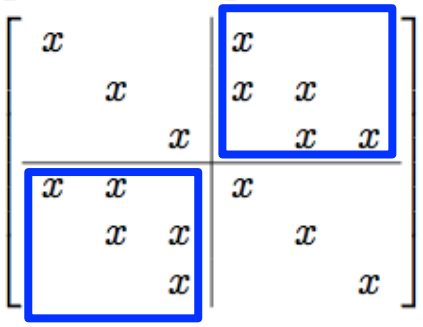
$$= \begin{bmatrix} x & x & \\ x & x & x \\ & x & x \end{bmatrix} - \begin{bmatrix} & & x \\ & & \\ & & \end{bmatrix} \begin{bmatrix} x & & \\ x & & \\ x & & \end{bmatrix}$$

$$= \begin{bmatrix} x & x & \\ x & x & x \\ & x & x \end{bmatrix} - \begin{bmatrix} x & & \\ & & \\ & & \end{bmatrix}$$

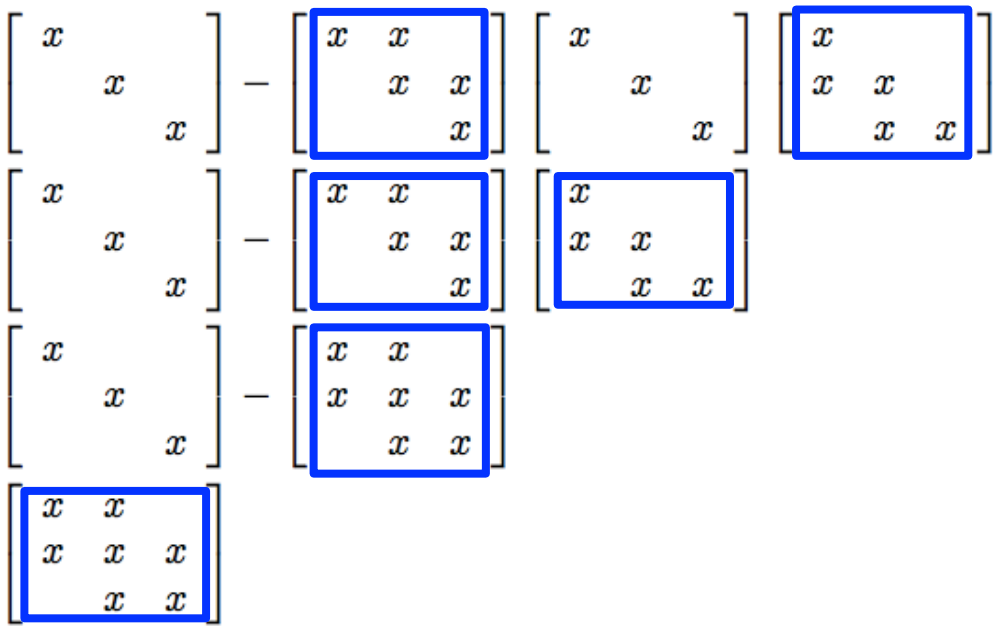
$$= \begin{bmatrix} x & x & \\ x & x & x \\ & x & x \end{bmatrix}$$

Preservation of tridiagonal structure on permuted Schur complementation

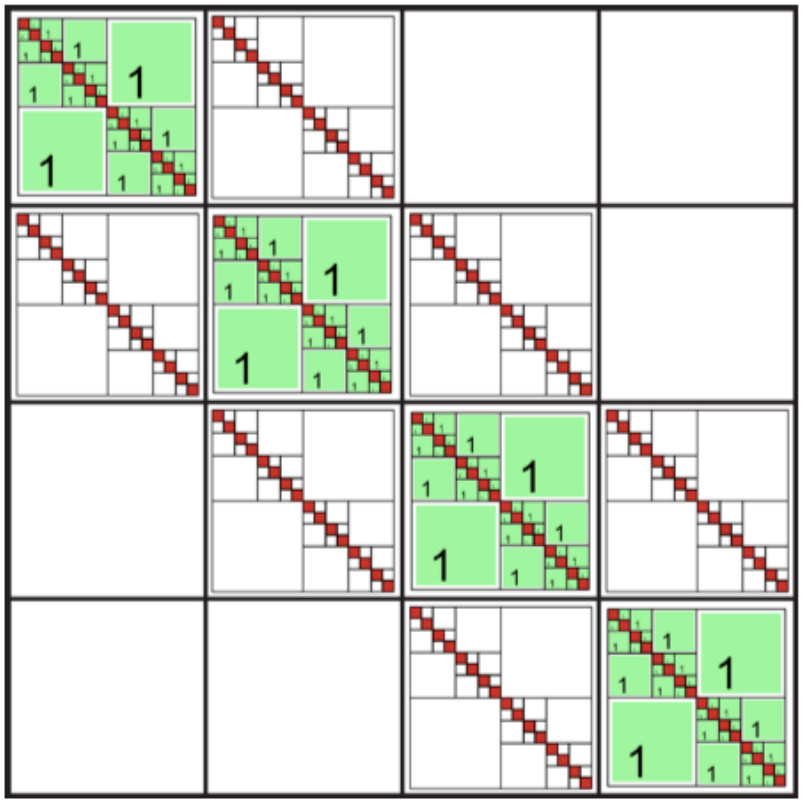
$$P^T A P = \begin{bmatrix} \tilde{A}_{11} & \tilde{A}_{12} \\ \tilde{A}_{21} & \tilde{A}_{22} \end{bmatrix}$$



$$S = \tilde{A}_{22} - \tilde{A}_{21}\tilde{A}_{11}^{-1}\tilde{A}_{12}$$

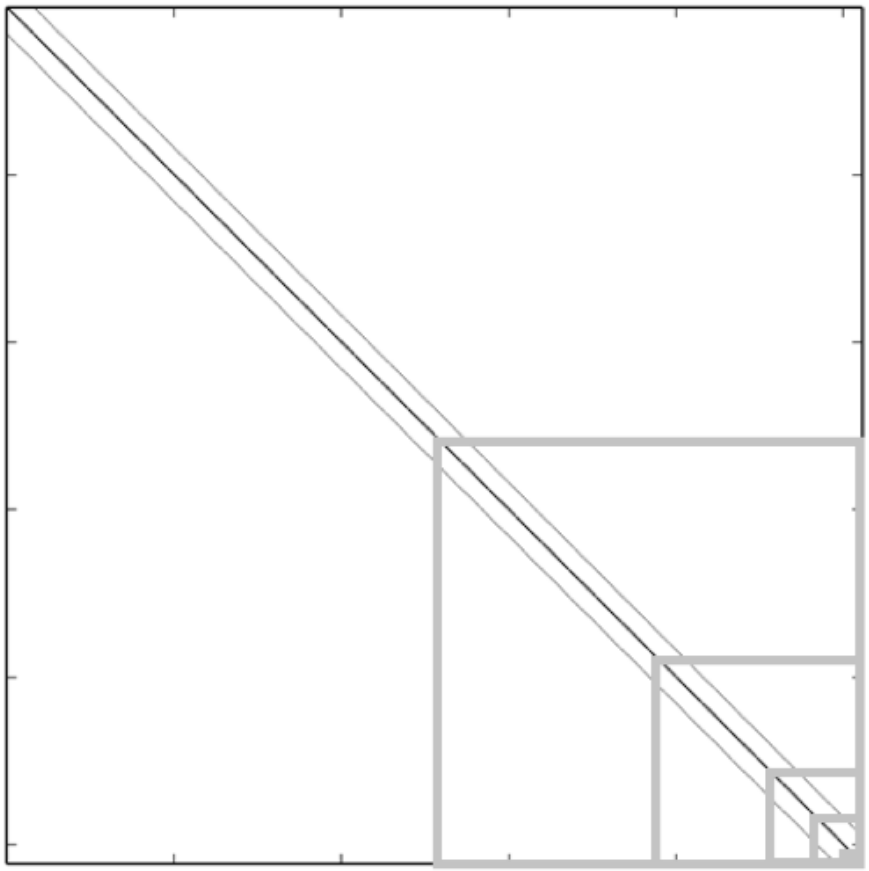


Block structure in 2D

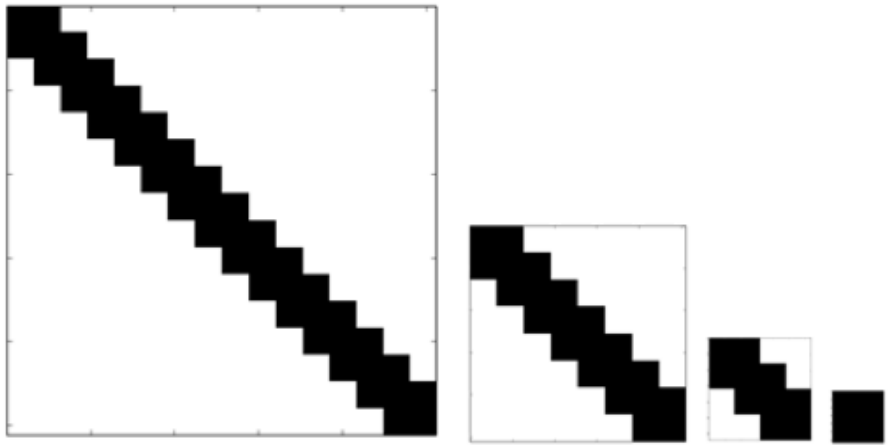


- **Issue: diagonal block inversion is non-optimal when FFT does not apply**
- **Enter hierarchical matrix arithmetic**
  - **how will rank grow?**

Block recursive application



	Operations	Storage
CR	$\mathcal{O}(N^2)$	$\mathcal{O}(N^{1.5} \log N)$
ACR	$\mathcal{O}(N \log^2 N)$	$\mathcal{O}(N \log N)$



Pseudocode

Algorithm 1 Accelerated Cyclic Reduction (ACR)

1: $n = 2^q, q \geq 2$
2: Block-wise low-rank approximation of A: $H^{(0)} = A$
3: Set-up right-hand side f
4: **for** $i = 1$ **to** q **do**
5: $\quad H^{(i)} = PAP^T$
6: $\quad H^{(i+1)} = H_{22}^{(i)} \ominus H_{21}^{(i)} \otimes \mathcal{H}\textbf{inverse}(H_{11}^{(i)}) \otimes H_{12}^{(i)}$
7: $\quad f^{(i+1)} = b_{odd} - H_{21}^{(i)} \otimes \mathcal{H}\textbf{inverse}(H_{11}^{(i)}) \otimes b_{even}$
8: $\quad u_{odd} = \mathcal{H}\textbf{inverse}(H_{11}^{(i+1)}) f^{(i+1)}$
9: **end for**
10: Perform back-substitution
11: Permute solution vector back to natural ordering

Numerical experiments

Second-order finite difference discretization with homogeneous BCs on the unit square

$$-\nabla \cdot \kappa(\vec{x}) \nabla u = f(\vec{x})$$

$$A = \mathsf{tridiag}(E_i, D_i, F_i) = \begin{bmatrix} D_1 & F_1 & & & \\ E_2 & D_2 & F_2 & & \\ & \ddots & \ddots & \ddots & \\ & & E_{n-1} & D_{n-1} & F_{n-1} \\ & & & E_n & D_n \end{bmatrix}$$

Numerical experiments

Six 2D experiments comparing ACR with H-LU and AMG

- Same forcing function
and homogeneous boundary conditions:

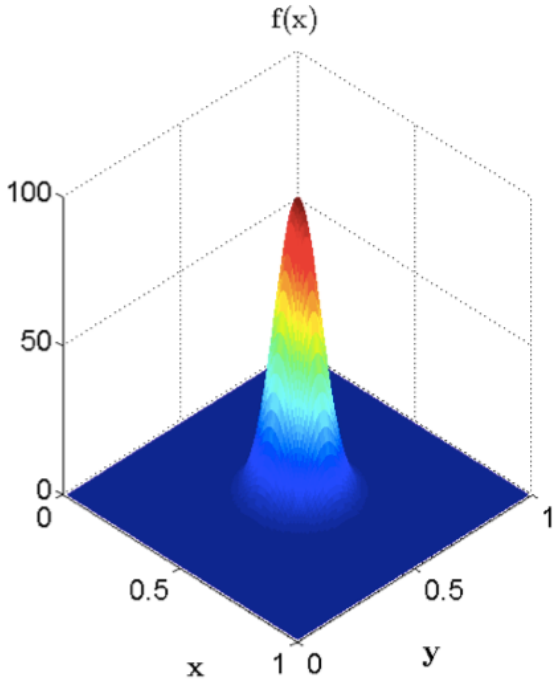
$$f(\mathbf{x}) = 100e^{-100((x-0.5)^2+(y-0.5)^2)}$$

- Both codes
attempt a **direct solution** relative to the discretization error, using rank truncation:

$$\left|\left|u - \hat{u^h}\right|\right|_2 \sim \frac{||Ax-b||_2}{||b||_2}$$

Both codes are
compiled with **icc15** and **O2** optimization enabled. In the same computer system.

- Benchmark is against the $\mathcal{H}$-LU
multi-core implementation of **HLIBPro** by R. Kriemann et al.



Hardware architecture

Metric	Westmere Xeon X5650	Ivy Bridge Xeon E5-2680 v2	Haswell Xeon E5-2699 v3	AMD Opteron 6376
Cores	12	20	36	64
Clock frequency (GHz)	2.66	2.9	2.3	2.6
L3 Total size (KB)	12,288	25,600	46,080	12,288
Cache/core (KB)	1,024	1,280	1,280	192
Bandwith (GB/s)	23.3	57.7	94.46	45.8

Truncation error algebraic convergence tuning and log-lin complexity growth

1 core of Westmere

$$-\nabla^2 u = f(\mathbf{x}). \qquad \mathbf{x} \in \Omega = [0,1]^2 \quad u(\mathbf{x}) = 0, \quad x \in \Gamma$$

$$f(\mathbf{x}) = 100e^{-100((x-0.5)^2+(y-0.5)^2)}$$

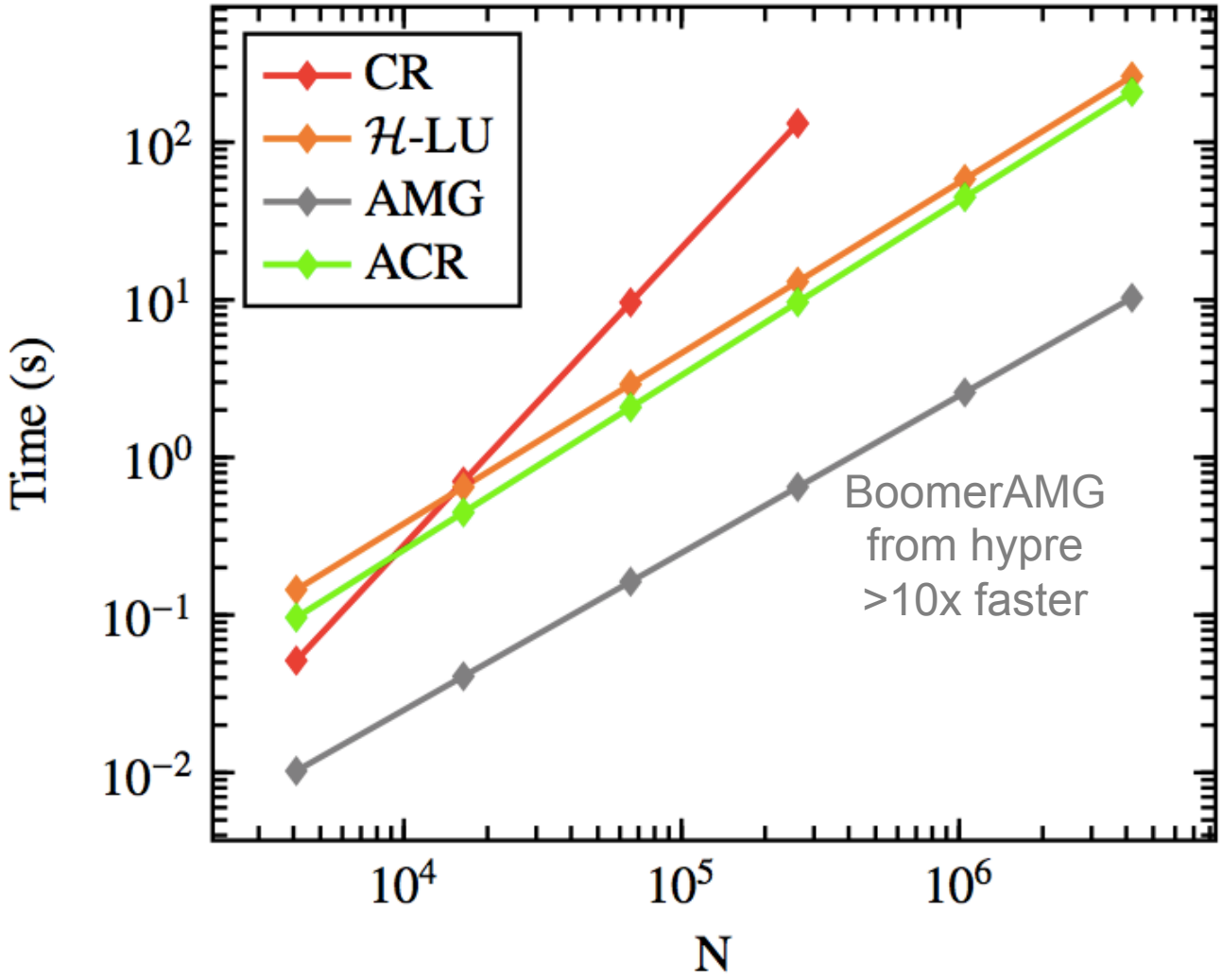
$h=n^{-1}$

$N=n^2$ $\sim h^2$ $\sim N \log N$ $\sim N \log^2 N$

n	ACR_ϵ	Error	Memory (Bytes)	Time (sec)
64	1.0×10^{-5}	1.39×10^{-4}	5.07×10^{6}	0.12
128	5.2×10^{-6}	1.74×10^{-5}	5.50×10^{7}	0.37
256	1.0×10^{-7}	3.30×10^{-6}	2.26×10^{8}	1.94
512	4.0×10^{-9}	7.59×10^{-7}	9.27×10^{8}	8.93
1,024	1.0×10^{-10}	1.34×10^{-7}	3.82×10^{9}	45.79
2,048	1.0×10^{-11}	2.77×10^{-8}	1.57×10^{10}	228.62

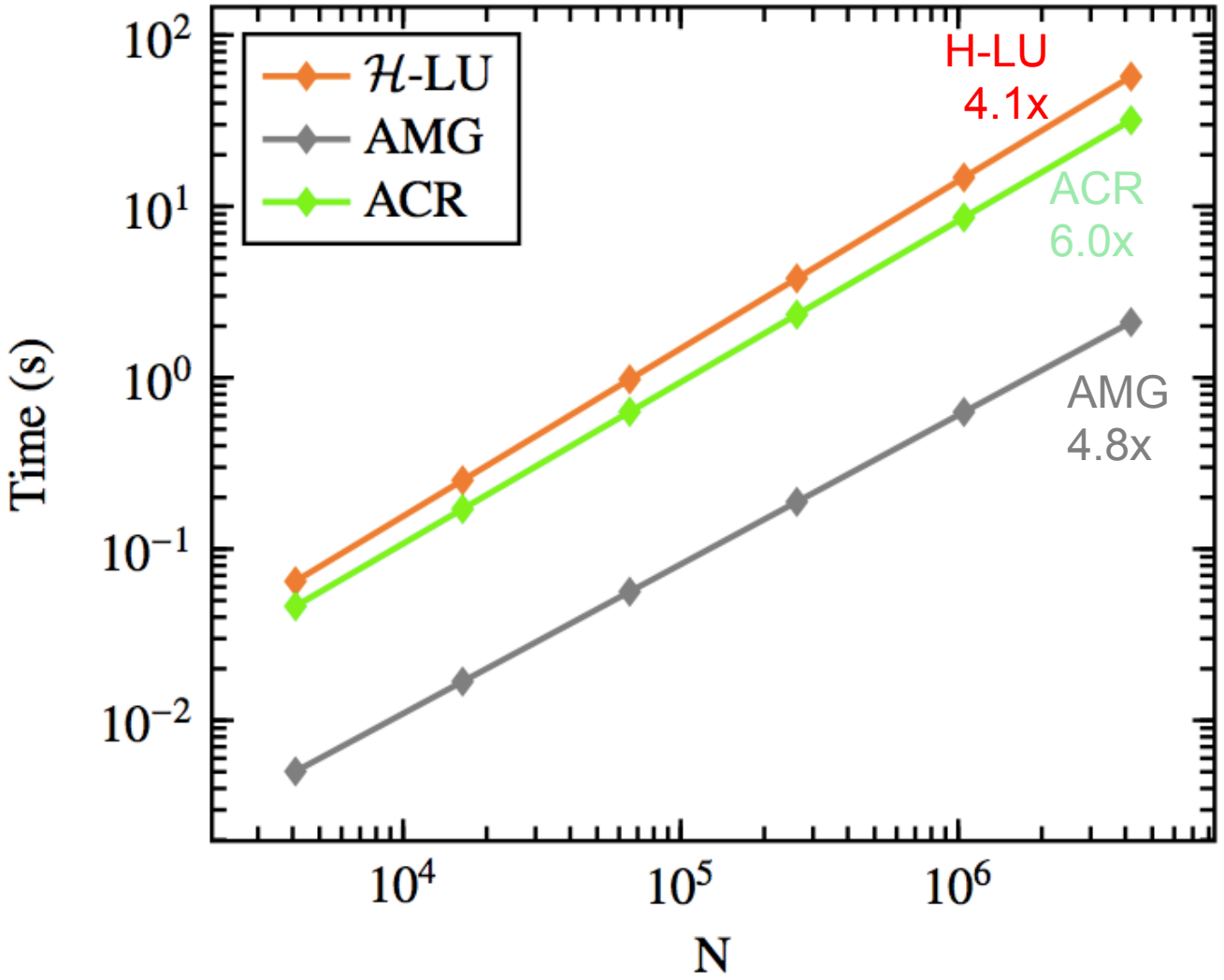
Poisson

1 core of Westmere



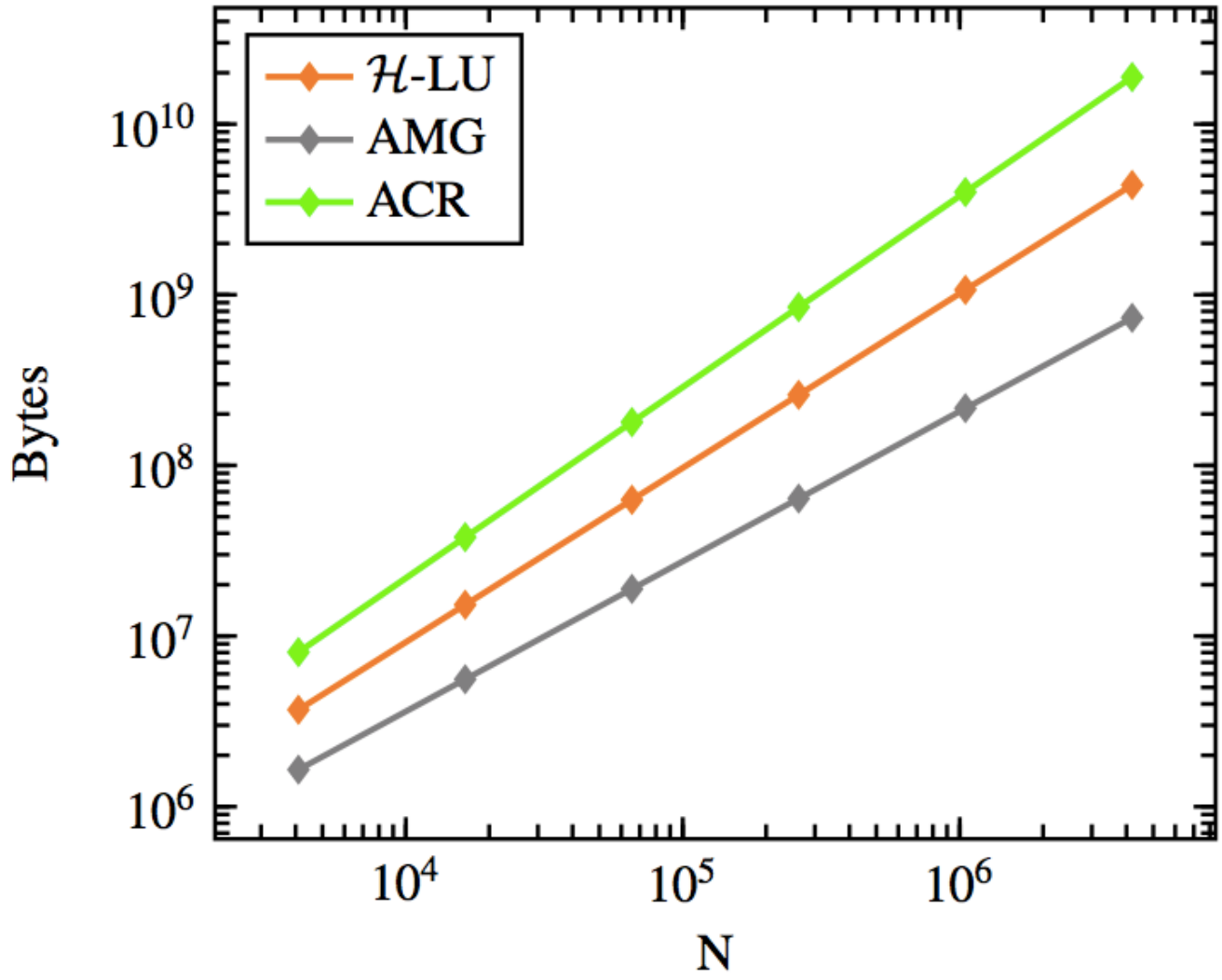
Poisson

12 cores of Westmere



Poisson

Memory consumption

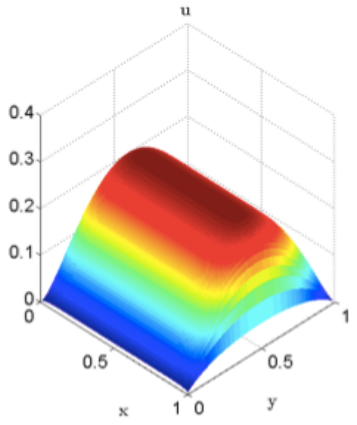


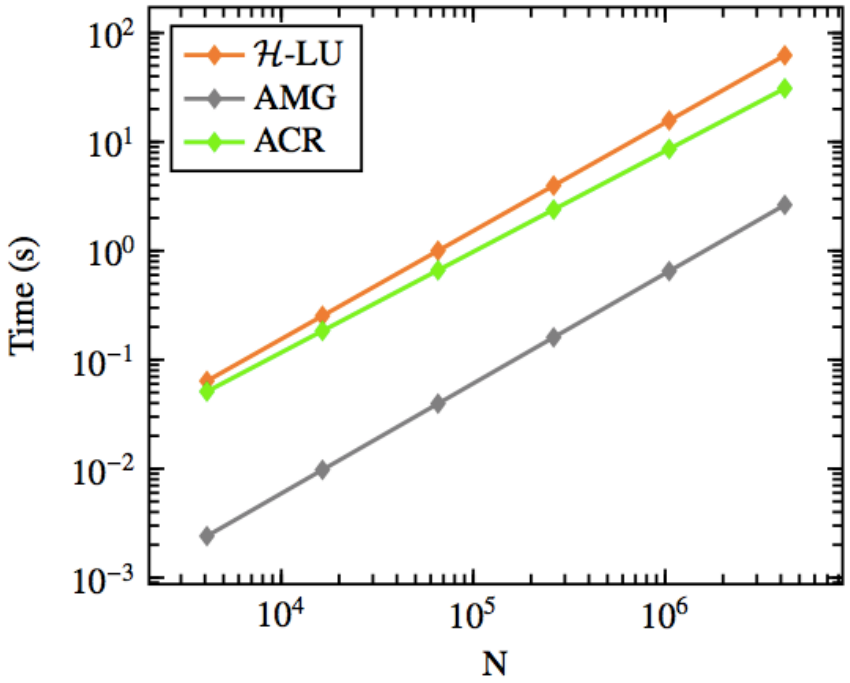
Anisotropy

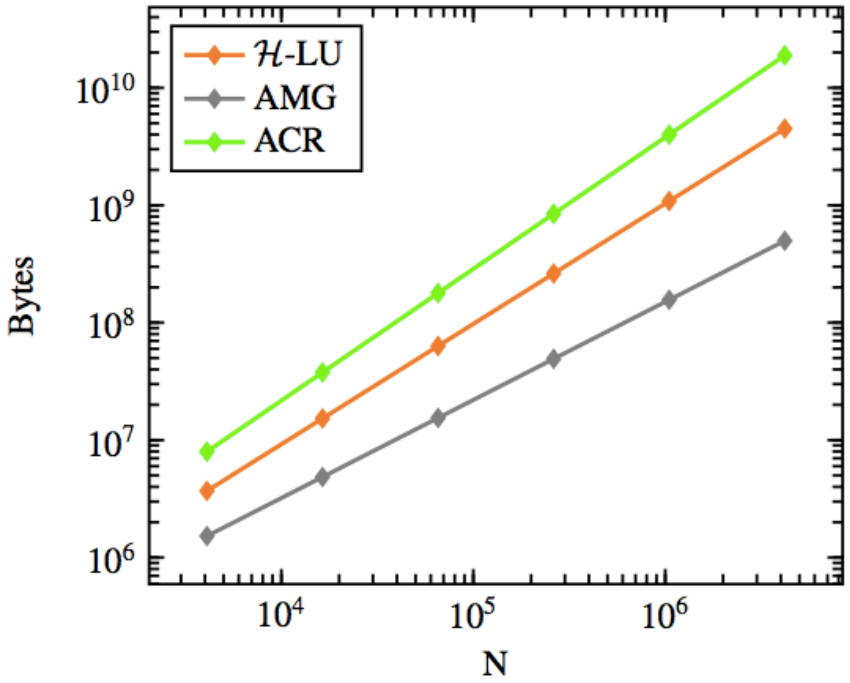
12 cores of Westmere

$$\epsilon U_{xx} + \tfrac{1}{\epsilon} U_{yy} = f(\mathbf{x})$$

$$\epsilon = 10$$





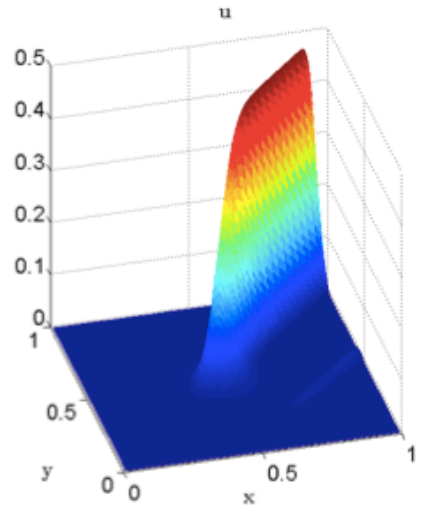


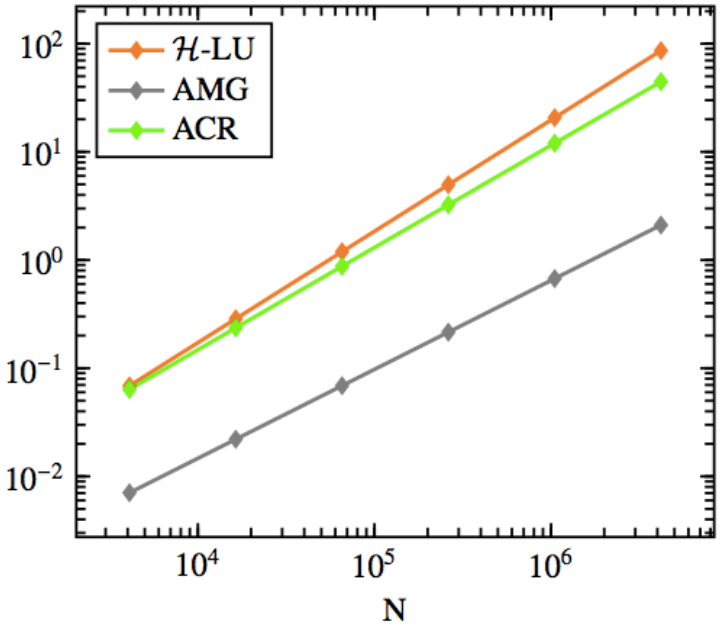
Convection-diffusion [Gillman, 2011]

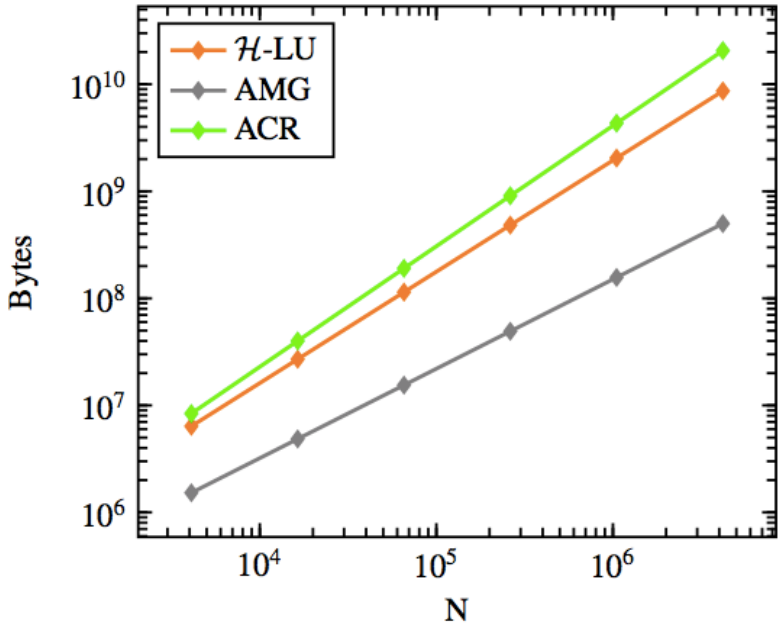
12 cores of Westmere

$$-a(\mathbf{x})\nabla^2 u + b(\mathbf{x}) \cdot \nabla u = f(\mathbf{x})$$

$$a = 125\sin(4\pi\mathbf{x})$$
$$b = 125\cos(4\pi\mathbf{x})$$







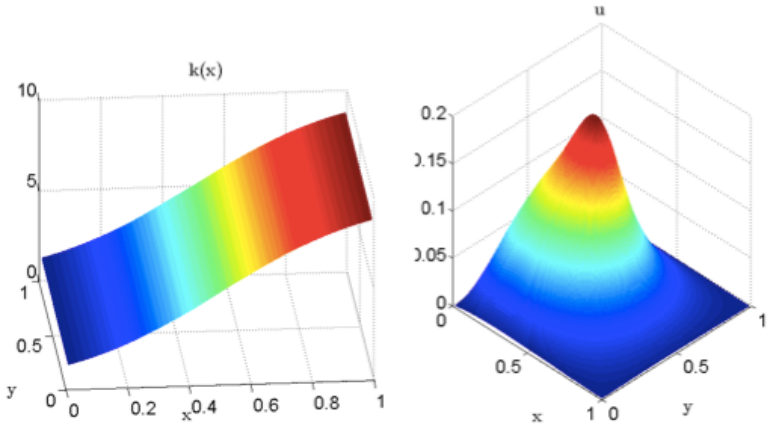
Smoothly varying diffusivity

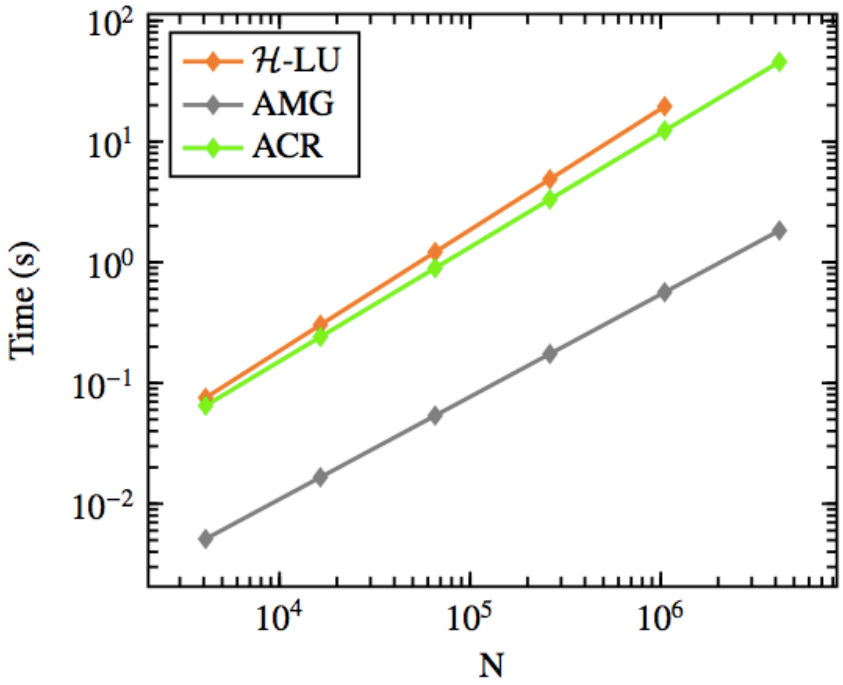
12 cores of Westmere

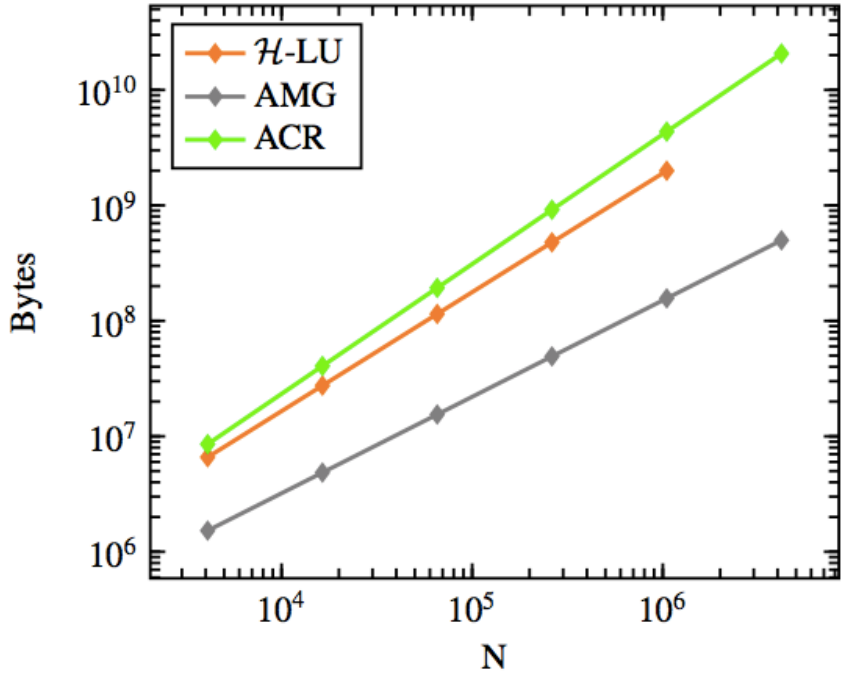
$$-\nabla \cdot \kappa(\mathbf{x}) \nabla u = f(\mathbf{x})$$

$$\kappa(x,y) = \kappa^- + (\kappa^+ + \kappa^-)\frac{1+\tanh(\frac{\mathbf{x}-0.5}{\epsilon})}{2}$$

$$\kappa^- = 0.1, \quad \kappa^+ = 10, \quad \epsilon = 0.5$$







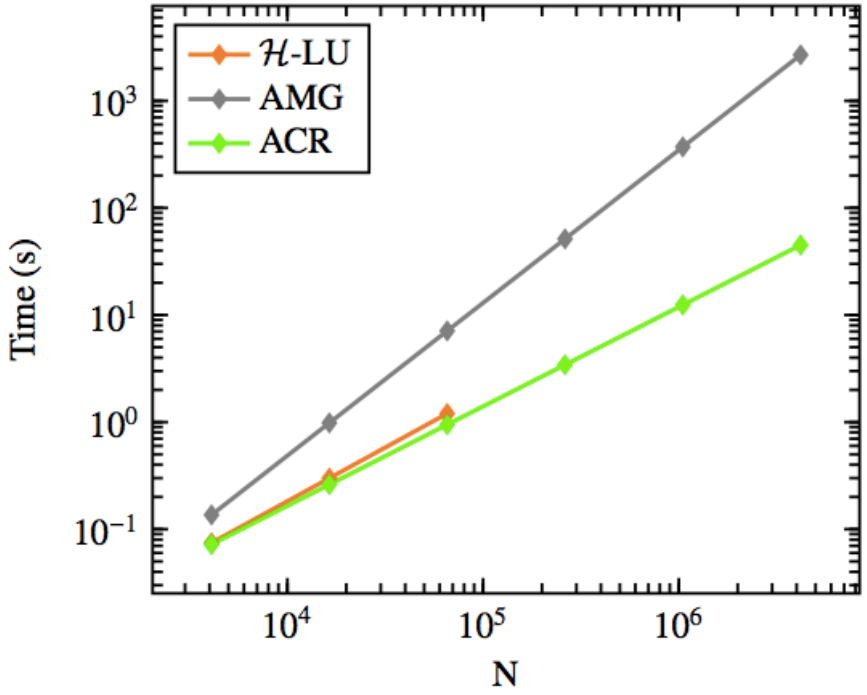
High contrast diffusivity [Ewing et al, 2001]

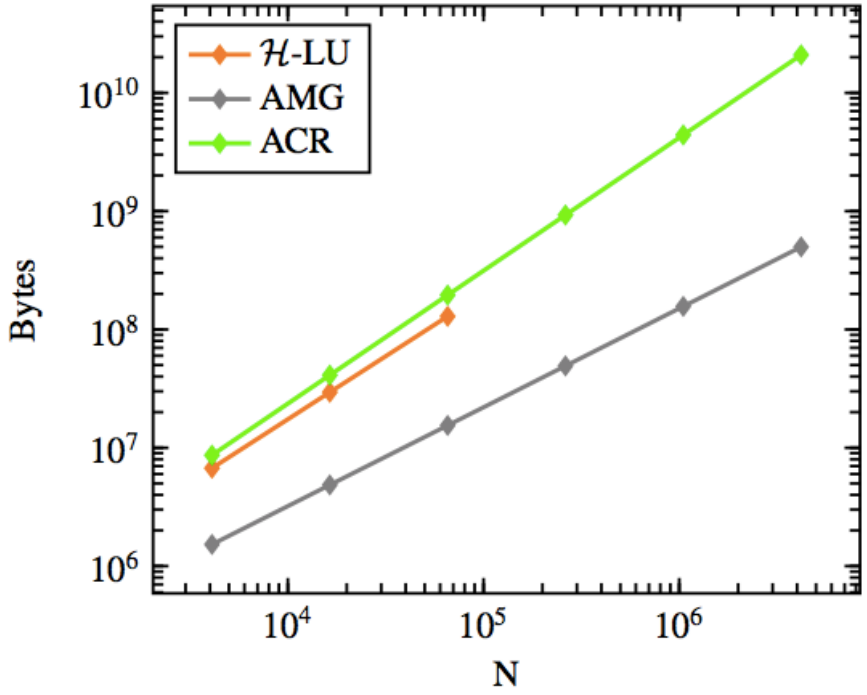
12 cores of Westmere

$$\kappa = \begin{cases} 1 \times 10^{-2} & x \le 0.5, y \le 0.5 \\ 1 & x > 0.5, y \le 0.5 \\ 1 \times 10^{-4} & x < 0.5, y > 0.5 \\ 1 \times 10^{6} & x \ge 0.5, y > 0.5 \end{cases}$$

$$\kappa_{i-\frac{1}{2},j} = \frac{2\kappa_{i,j}\kappa_{i-1,j}}{\kappa_{i,j}+\kappa_{i-1,j}} \qquad \kappa_{i,j-\frac{1}{2}} = \frac{2\kappa_{i,j}\kappa_{i,j-1}}{\kappa_{i,j}+\kappa_{i,j-1}}$$

$$\kappa_{i+\frac{1}{2},j} = \frac{2\kappa_{i,j}\kappa_{i+1,j}}{\kappa_{i,j}+\kappa_{i+1,j}} \qquad \kappa_{i,j+\frac{1}{2}} = \frac{2\kappa_{i,j}\kappa_{i,j+1}}{\kappa_{i,j}+\kappa_{i,j+1}}$$



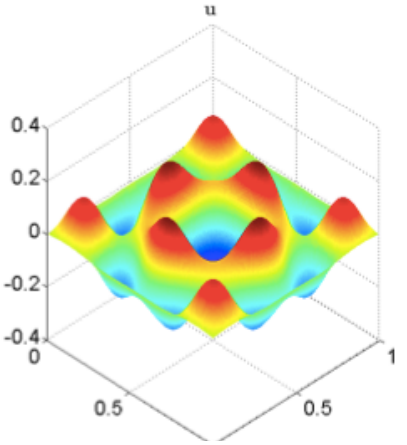


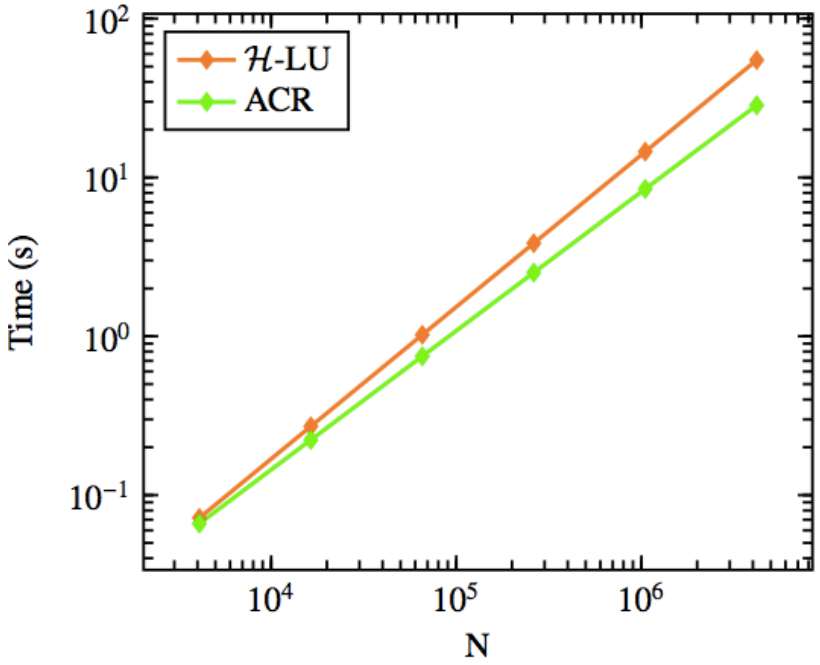
Helmholtz, fixed k

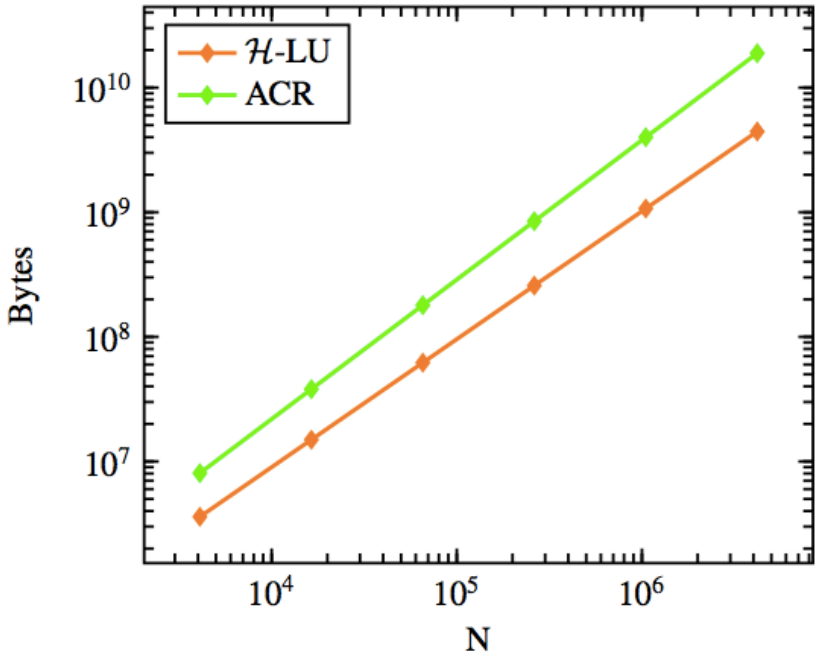
12 cores of Westmere

$$\nabla^2 u + k^2 u = f(\mathbf{x})$$

$$k = 20$$







Helmholtz, increasing k

12 cores of Westmere

ACR

n	k	Error	Memory (Bytes)	Time (sec)
1,024	1	2.41×10^{-6}	3.81×10^{9}	8.22
1,024	25	3.30×10^{-6}	3.81×10^{9}	8.32
1,024	50	1.96×10^{-6}	3.81×10^{9}	8.38
1,024	100	4.14×10^{-6}	3.83×10^{9}	8.62

H-LU

n	k	Error	Memory (Bytes)	Time (sec)
1,024	1	8.26×10^{-6}	1.07×10^{9}	14.35
1,024	25	5.10×10^{-6}	1.04×10^{9}	14.48
1,024	50	9.12×10^{-6}	1.03×10^{9}	14.04
1,024	100	1.41×10^{-6}	1.04×10^{9}	14.52

Helmholtz, fixed kh

12 cores of Westmere

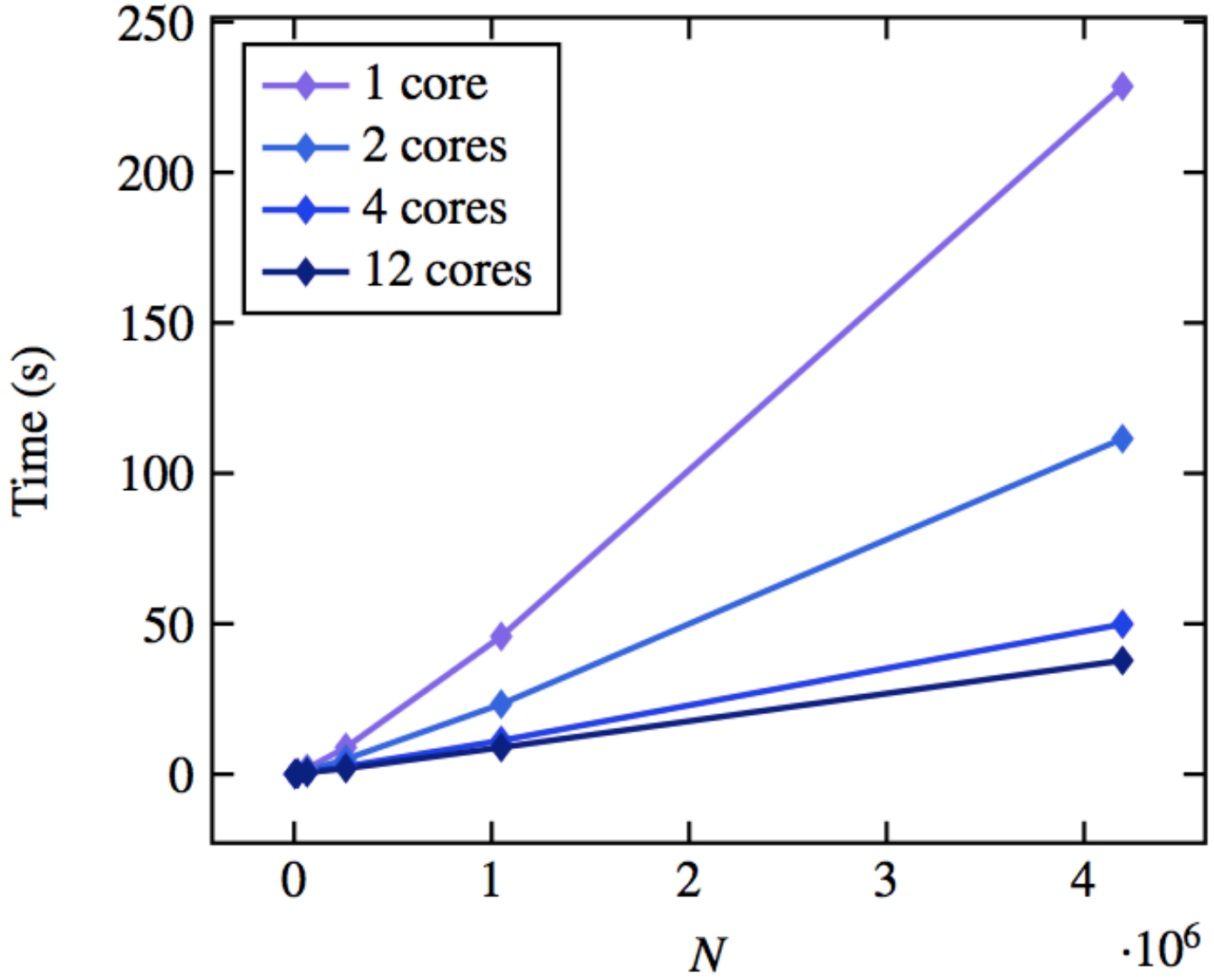
ACR

n	k	Error	Memory (Bytes)	Time (sec)
64	5	5.72×10^{-6}	5.12×10^{6}	0.09
128	10	7.41×10^{-6}	2.21×10^{7}	0.33
256	20	4.71×10^{-6}	9.53×10^{7}	1.07
512	40	7.24×10^{-6}	4.04×10^{8}	3.67
1,024	80	7.87×10^{-6}	1.72×10^{9}	14.37

H-LU

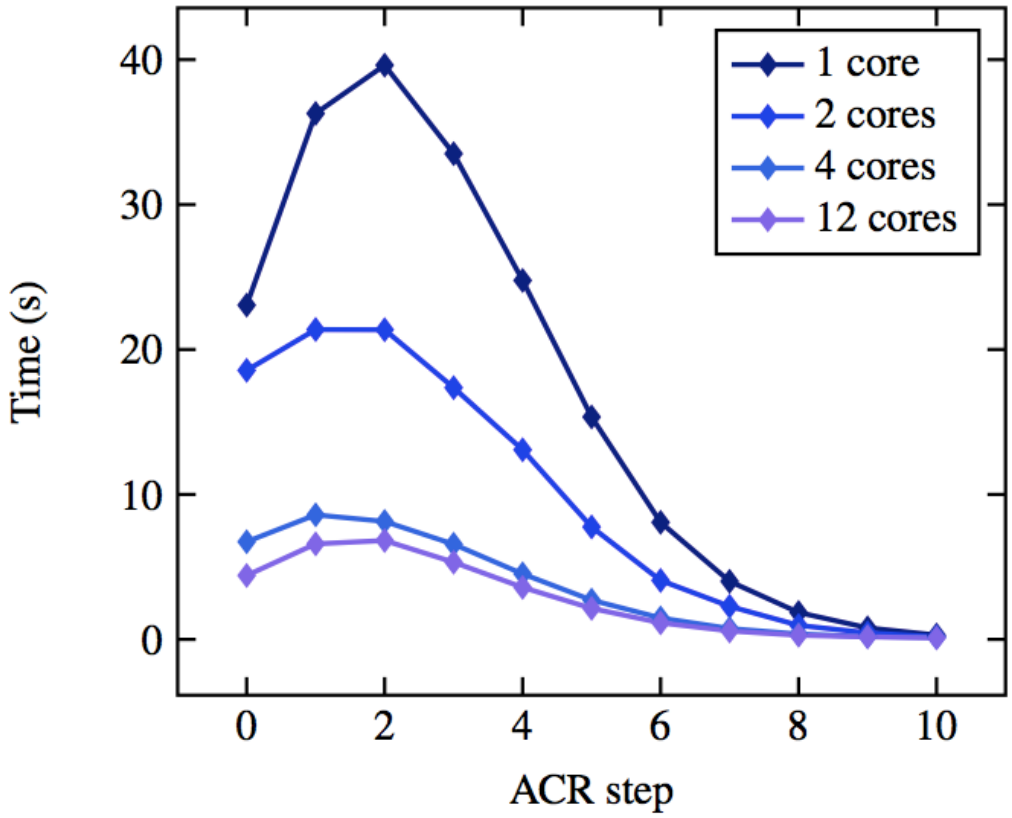
n	k	Error	Memory (Bytes)	Time (sec)
64	5	7.80×10^{-6}	3.64×10^{6}	0.08
128	10	4.03×10^{-6}	1.57×10^{7}	0.21
256	20	4.12×10^{-6}	6.20×10^{7}	0.94
512	40	7.42×10^{-6}	2.56×10^{8}	3.88
1,024	80	9.12×10^{-6}	1.04×10^{9}	14.61

Near linear complexity in problem size

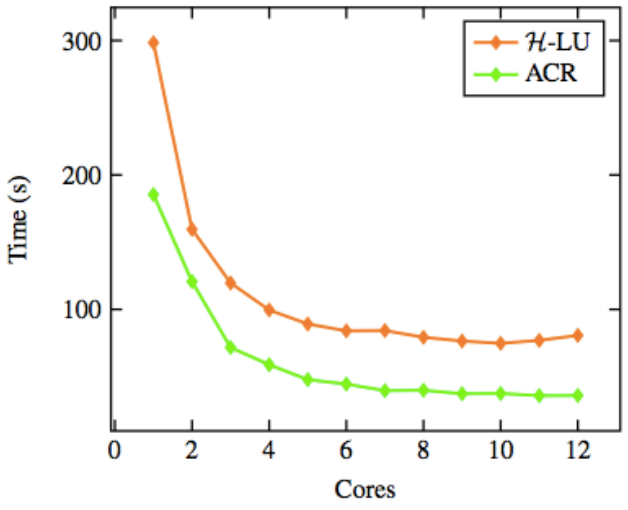


Time vs. log-stage for different core counts

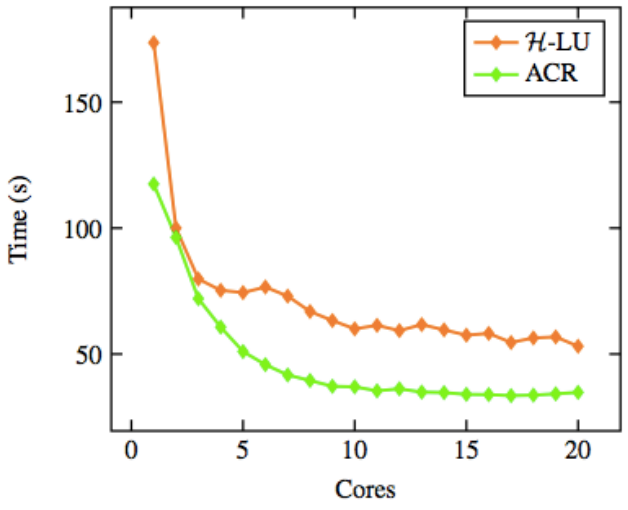
- **Working ranks vary from 1 (first stage, diagonal) up to 10 for later stages**
- **Blocks per stage decrease by 2**



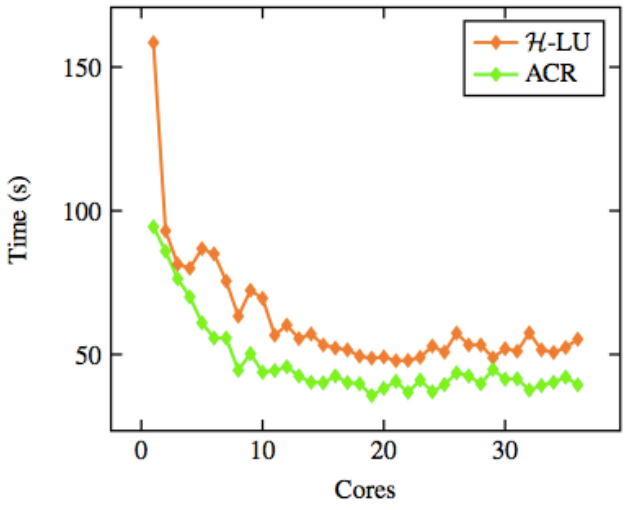
Multicore strong scaling



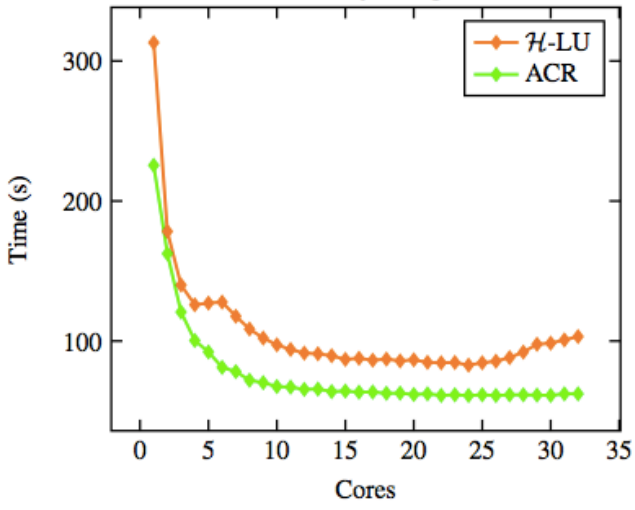
(a) Intel Westmere.



(b) Intel Ivy Bridge.

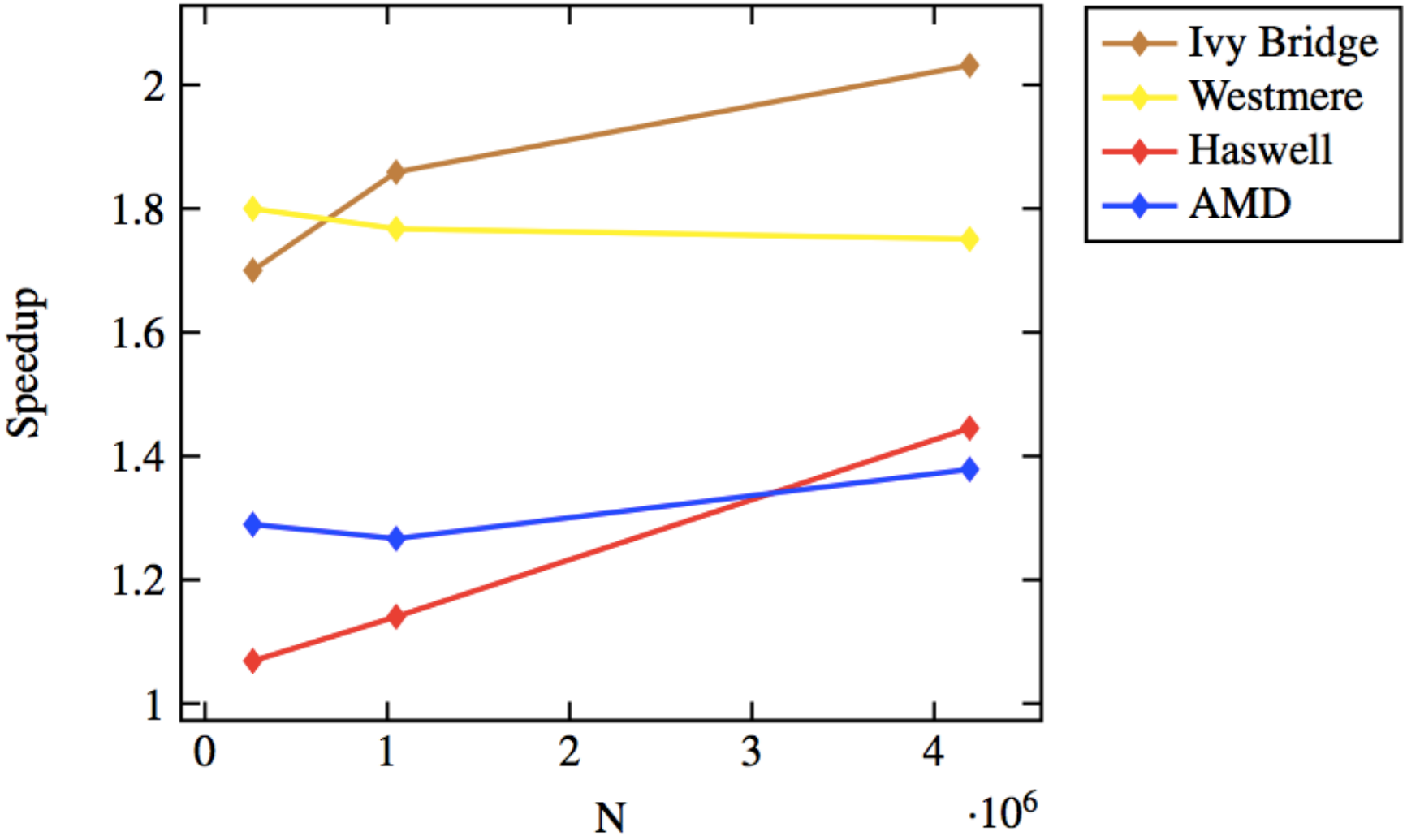


(c) Intel Haswell.

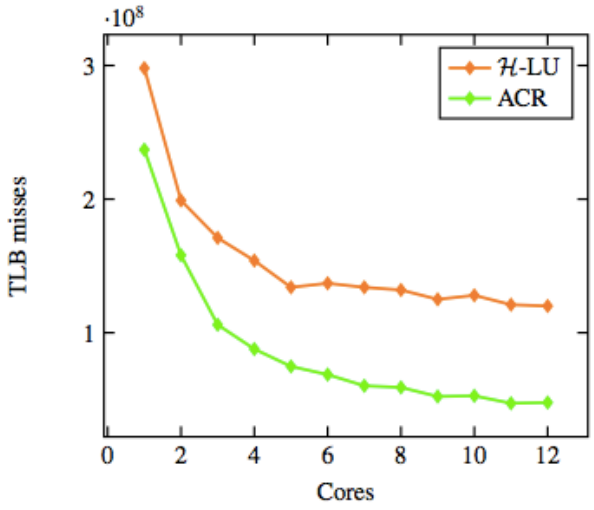


(d) AMD Opteron.

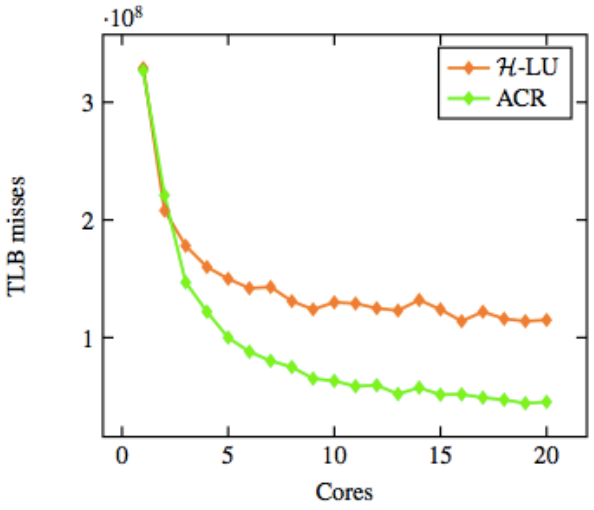
Speedup of ACR/H-LU vs. native H-LU



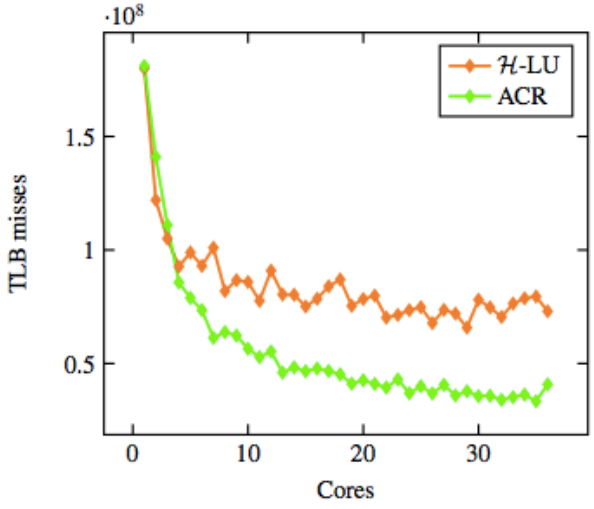
TLB miss comparisons



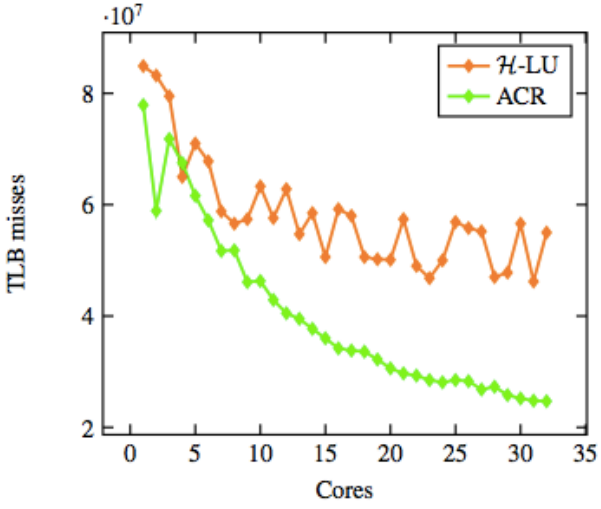
(a) Intel Westmere.



(b) Intel Ivy Bridge.

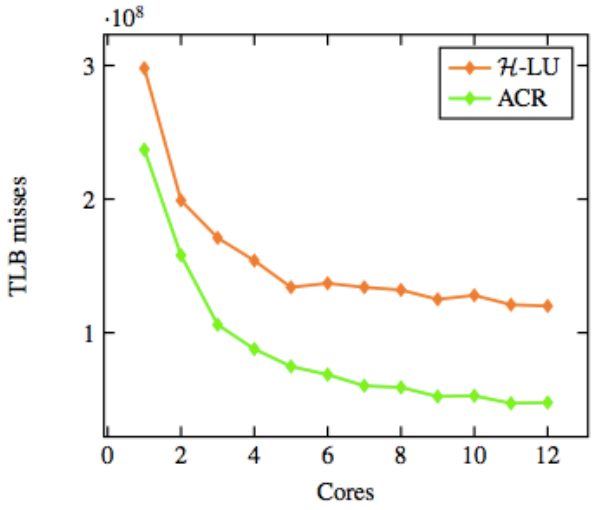


(c) Intel Haswell.

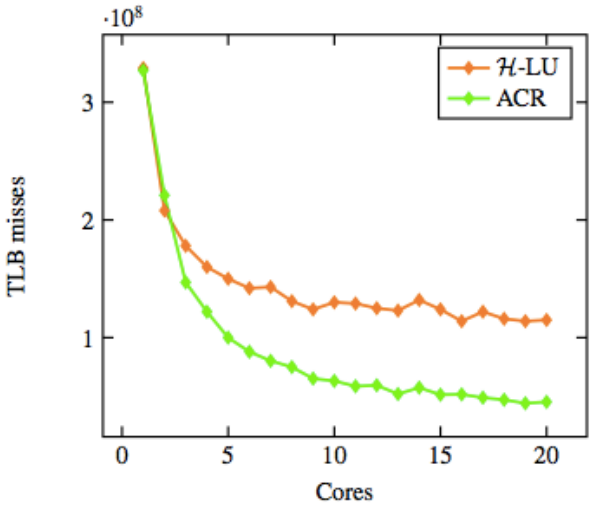


(d) AMD Opteron.

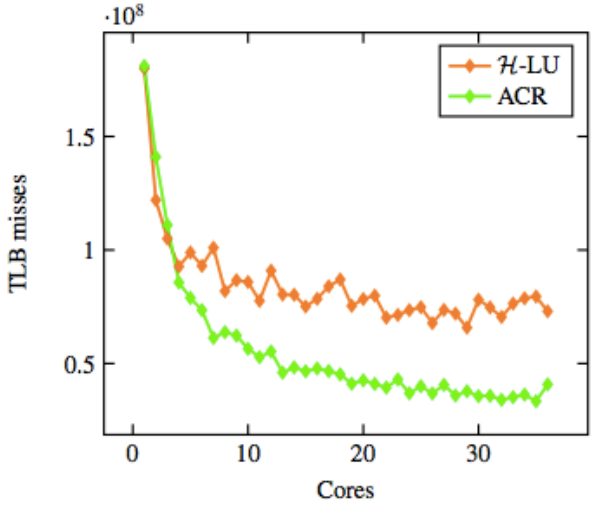
TLB miss comparisons



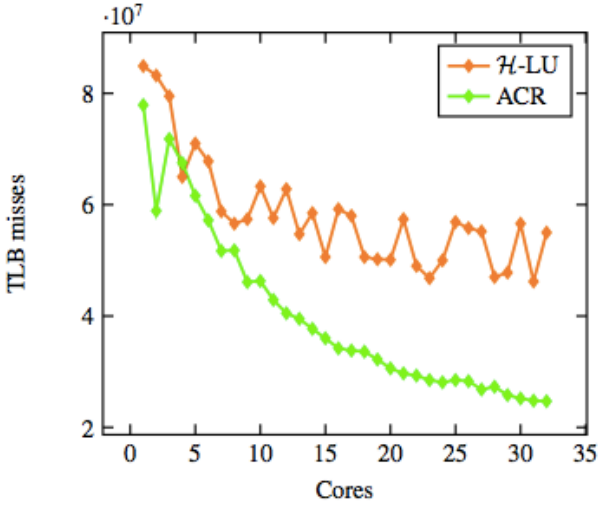
(a) Intel Westmere.



(b) Intel Ivy Bridge.



(c) Intel Haswell.

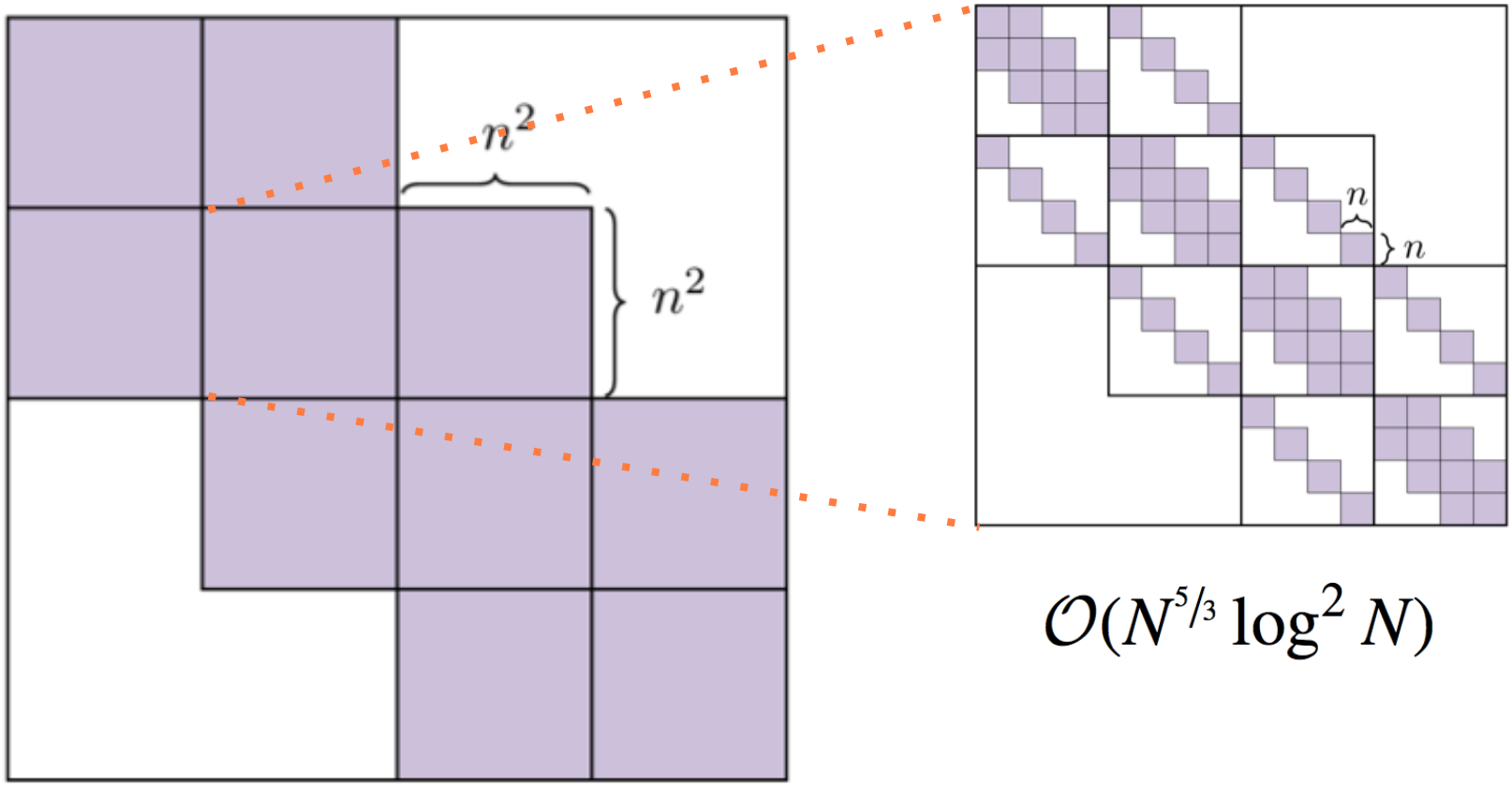


(d) AMD Opteron.

Comments

- **ACR is not in competition with other hierarchical matrix libraries**
  - **algorithmic outer wrapper**
  - **may unlock additional exploitable structure for concurrency and reuse**
- **Internal H arithmetic engine is modular**
  - **here, HLibPro®**
  - **ride emerging software for each emerging hardware environment**

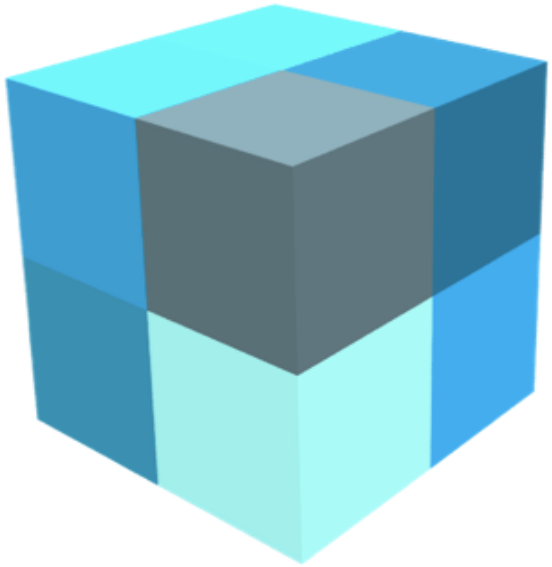
3D problems



- **For natural ordering and weak admissibility, the higher incompressible ranks of off-diagonal blocks make ACR suboptimal as a direct solver**
- **Strong admissibility recovers the formally optimal complexity, but the constant makes it noncompetitive**

Numerical experiment on high-contrast problem

$$-\nabla \cdot \kappa(\mathbf{x}) \nabla u = f(\mathbf{x}).$$



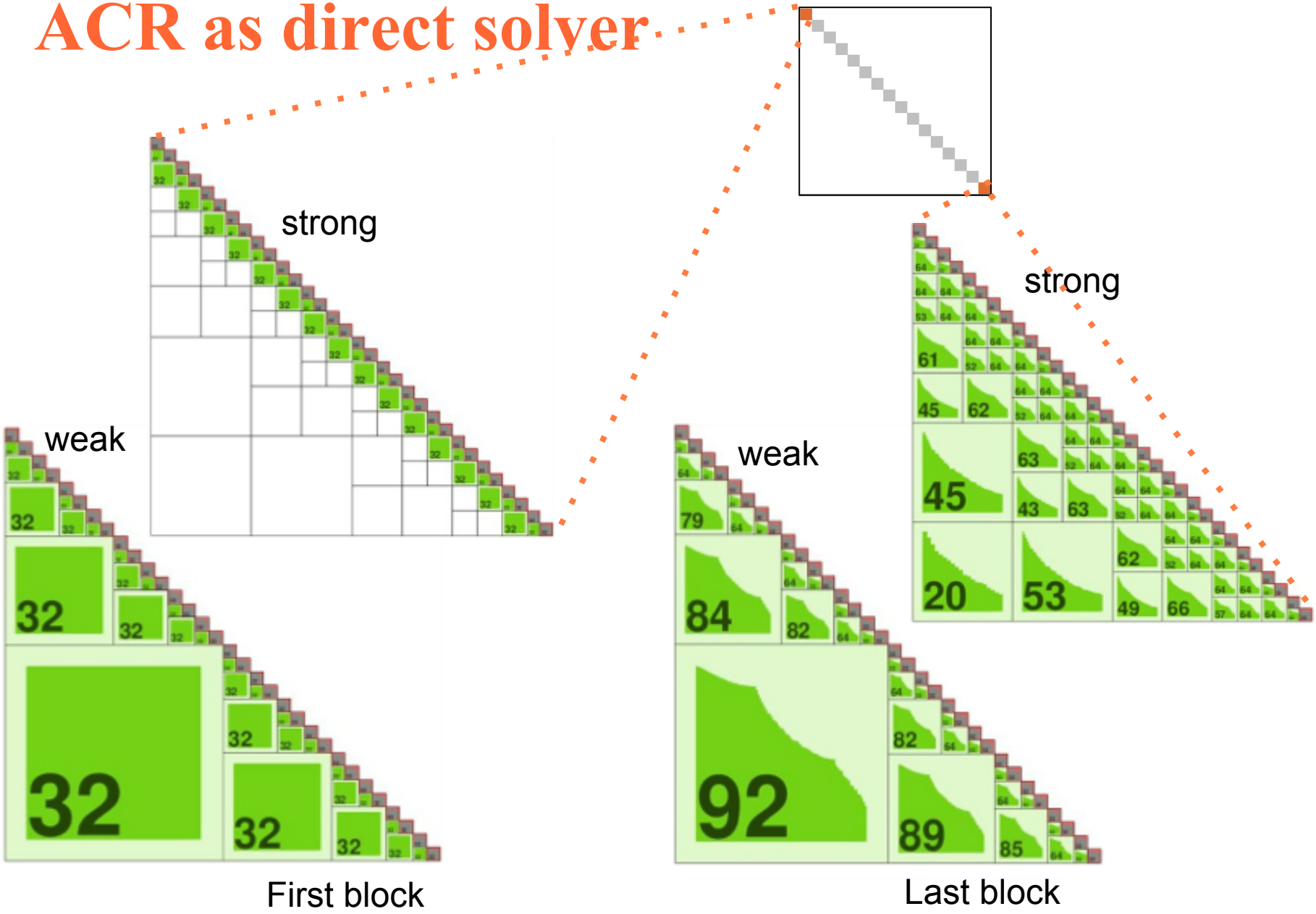
$$\kappa = D_1 \times D_2 \times D_3$$

$$D_1 = \begin{cases} 10^{-2} & 0 < x \leq 0.5 \\ 10^{3} & 0.5 \leq x < 1 \end{cases}$$

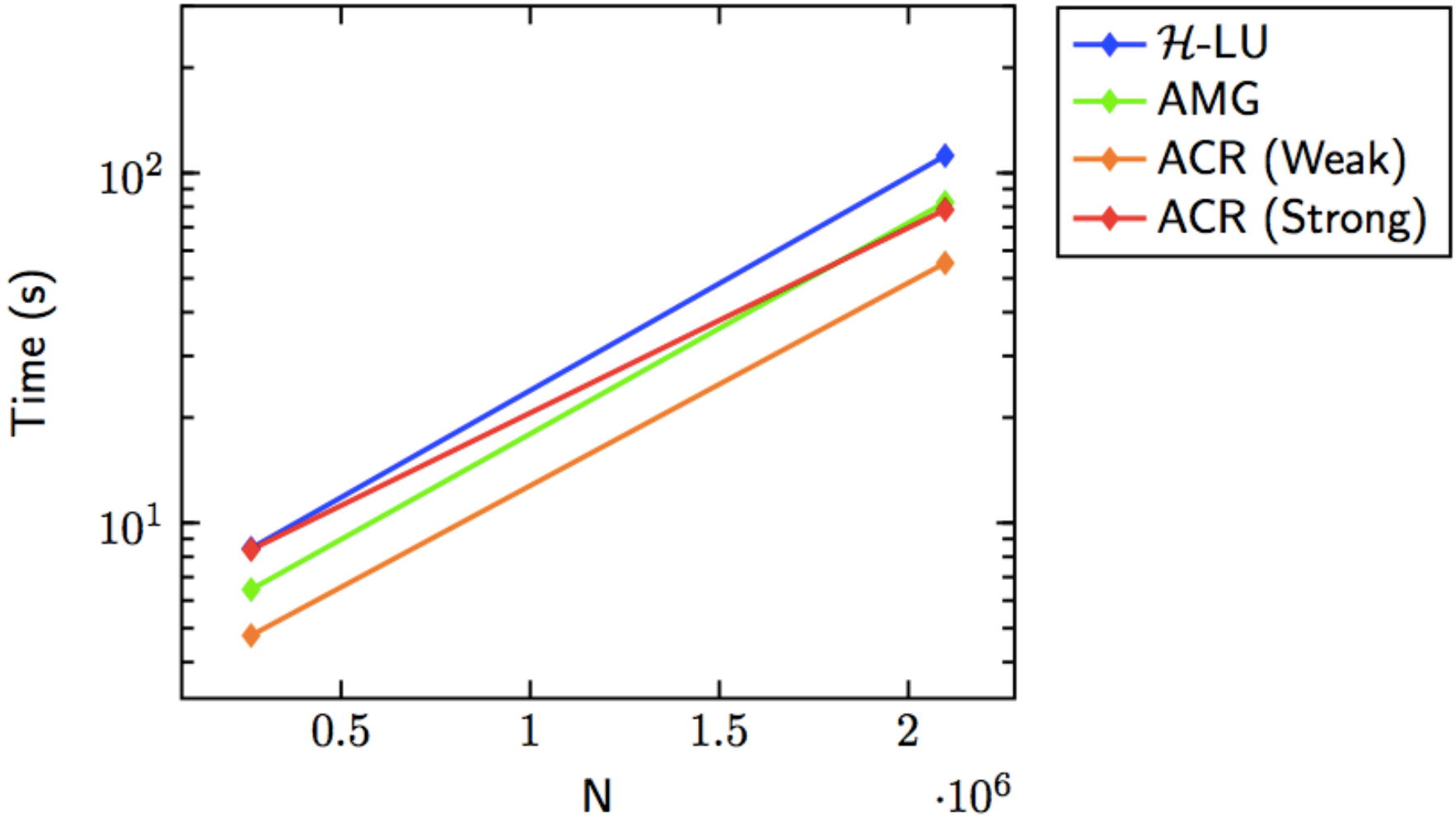
$$D_2 = \begin{cases} 1 & 0 < y \leq 0.5 \\ 10 & 0.5 \leq y < 1 \end{cases}$$

$$D_3 = \begin{cases} 10^{4} & 0 < z \leq 0.5 \\ 1 & 0.5 \leq z < 1 \end{cases}$$

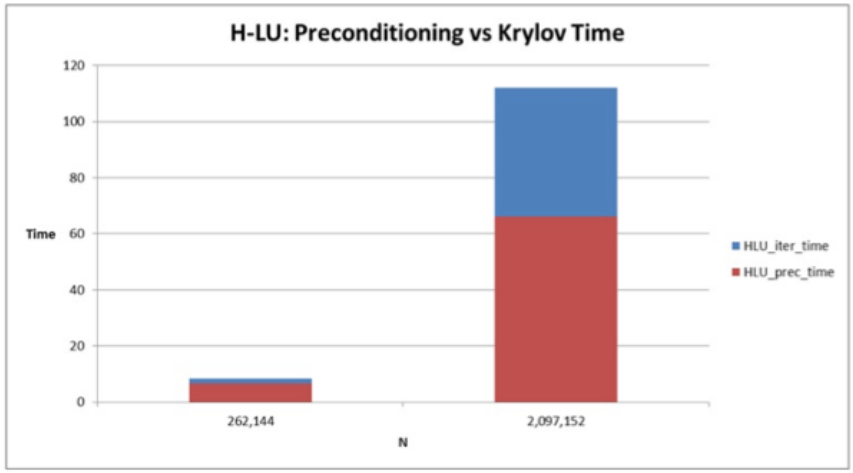
ACR as direct solver

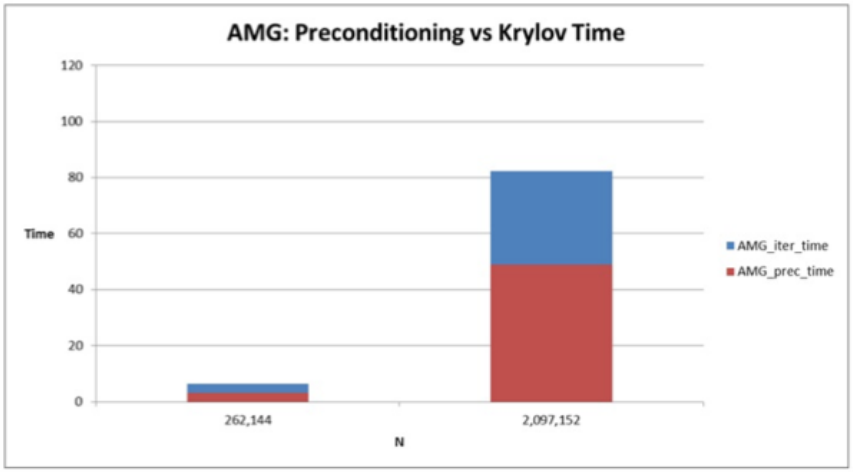


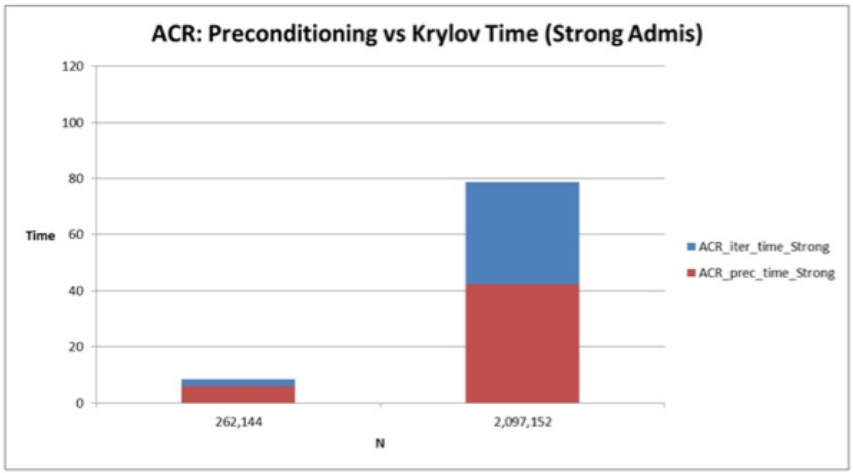
Radically truncated-rank ACR as preconditioner for 3D

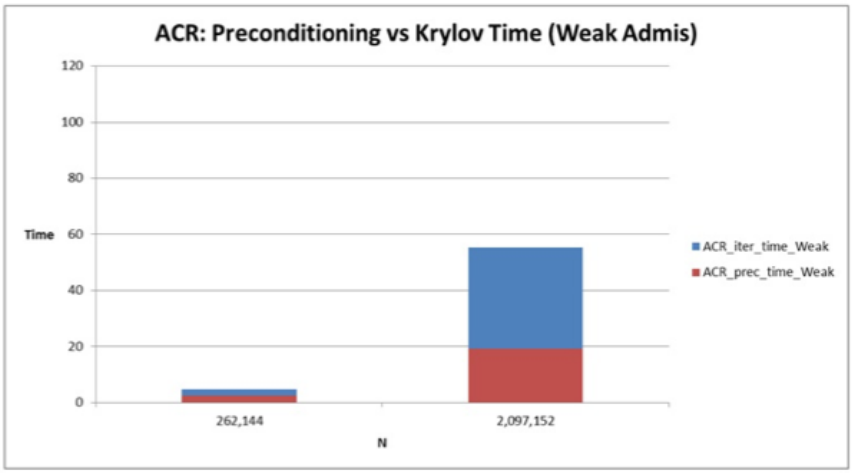


Comparing set up and iteration time for H-LU, ACR, and AMG in 3D





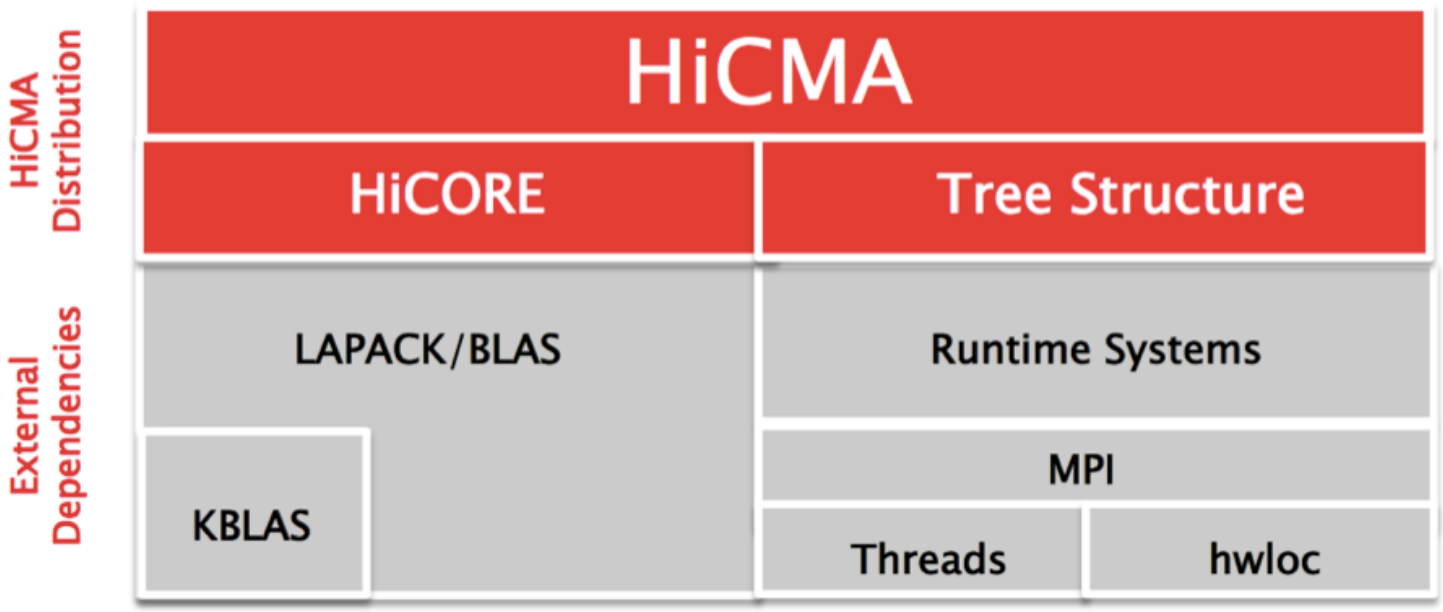




Future work

- **arXiv preprint forthcoming**
- **Adopt directed acyclic graph (DAG)-based approach to scheduling the block-sized tasks**
  - **already tested for H-LU in research codes for GPUs** [Kriemann, 2014]
  - **reduce synchronization idleness**
  - **allow more load-balancing across multiple grain sizes**

Hierarchical Computations on Many-core Architectures (HiCMA)



NVIDIA GPU RESEARCH CENTER

intel Intel Parallel Computing Centers

CRAY THE SUPERCOMPUTER COMPANY
CENTER OF EXCELLENCE

Thank you!