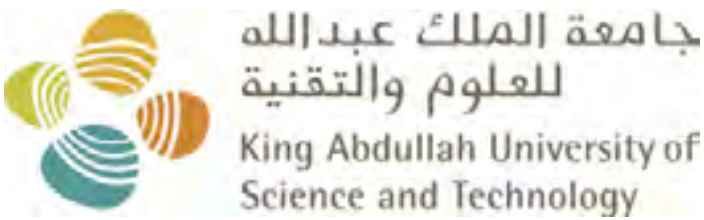
Hierarchical Matrix Computations: GPU Implementations and Applications

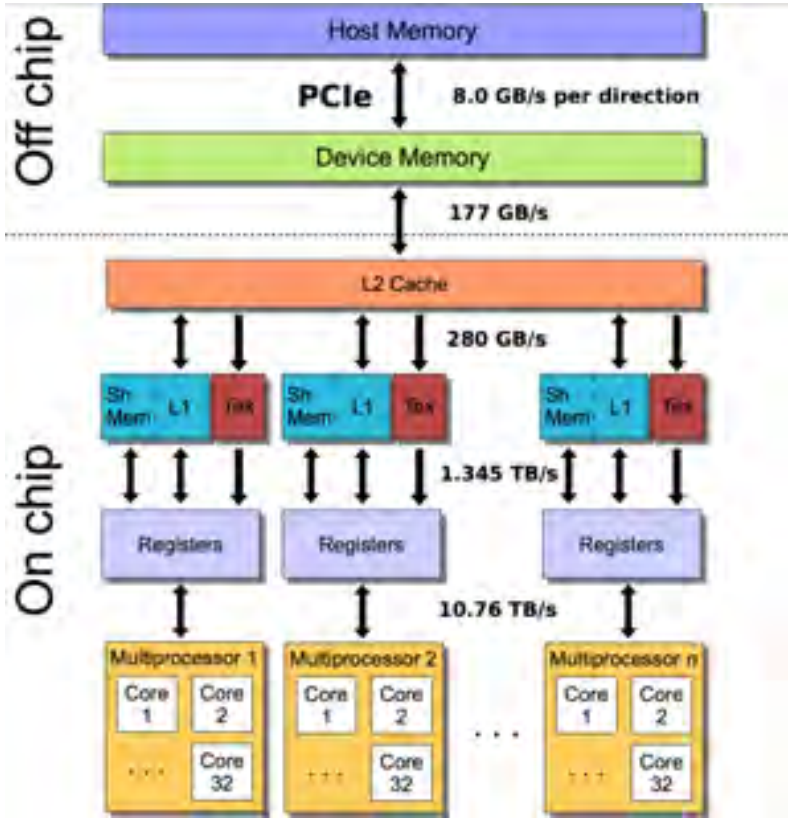
George Turkiyyah
KAUST and American University of Beirut

Workshop on Scalable Hierarchical Algorithms for Extreme Computing
May 9, 2016

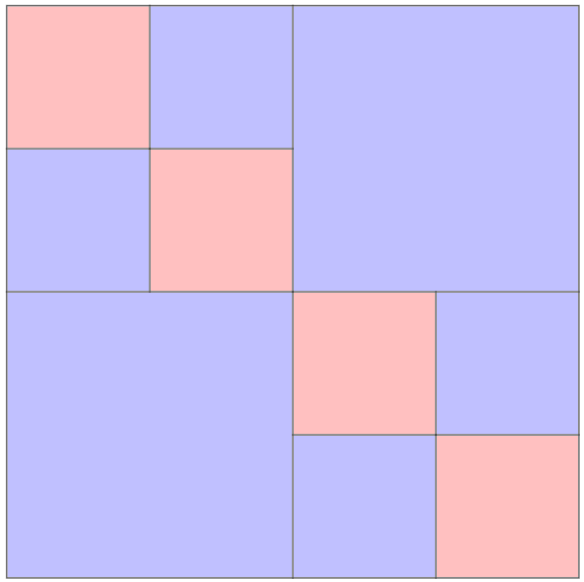


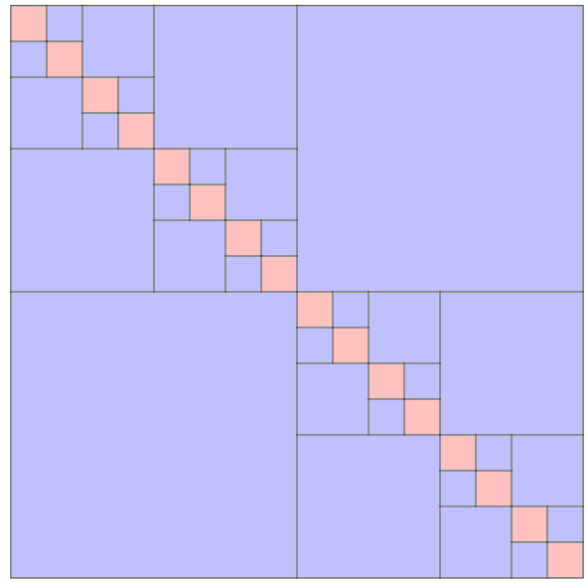
Context

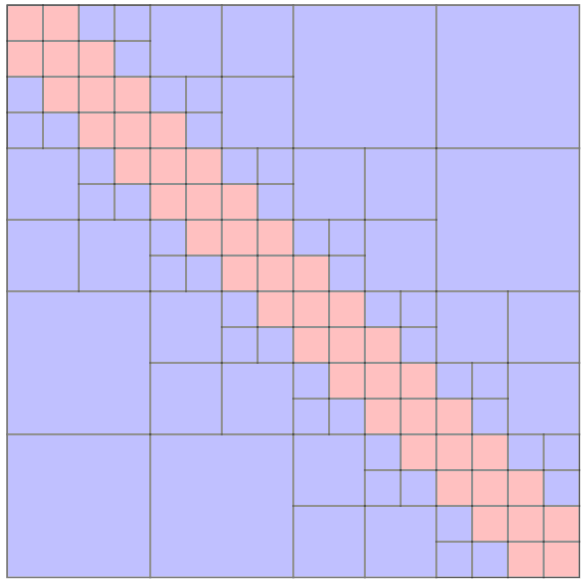
- substantial increase in node performance (manycore, wide vectors) will have to be accomplished with much smaller increase in memory
- we will be expected to work in memory-starved environments, compared to today's machines
- current GPUs are somewhat representative of this environment

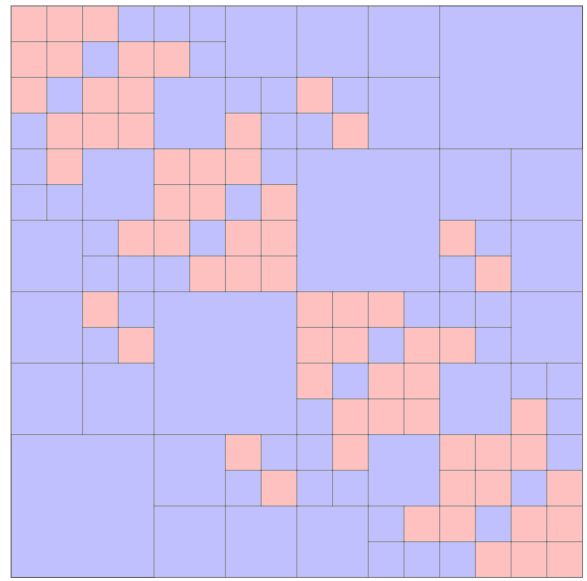


$\mathcal{H}$-matrix blocking

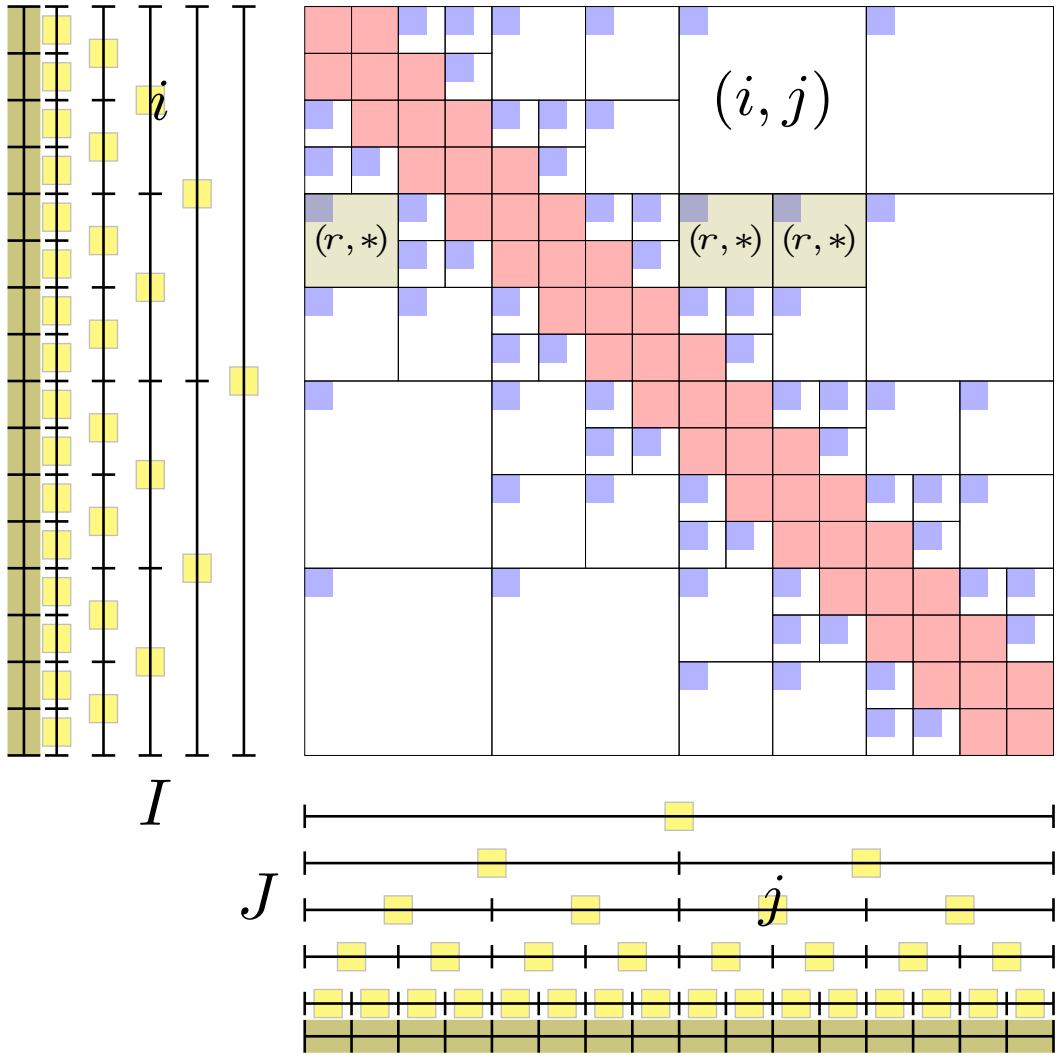






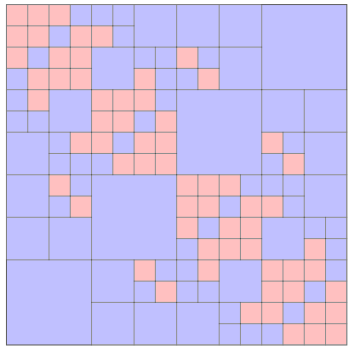


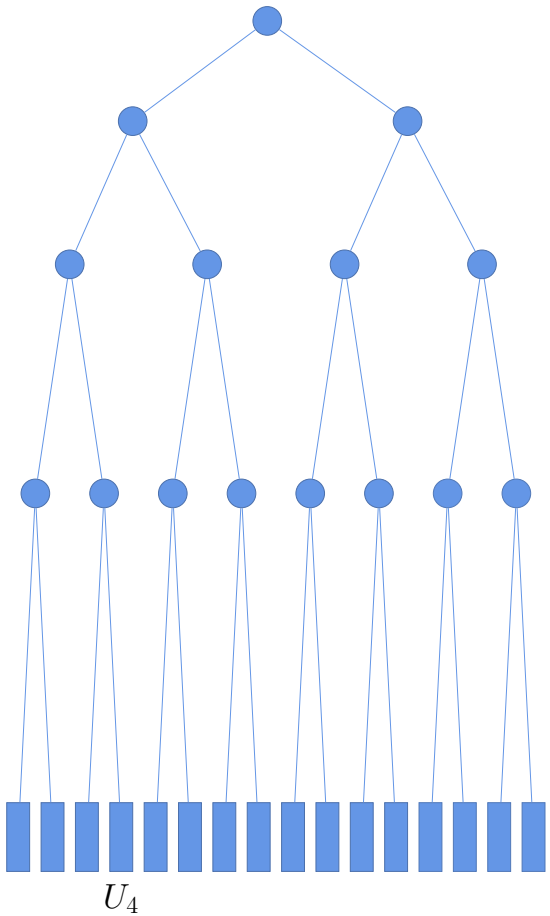
$\mathcal{H}^2$ nested bases

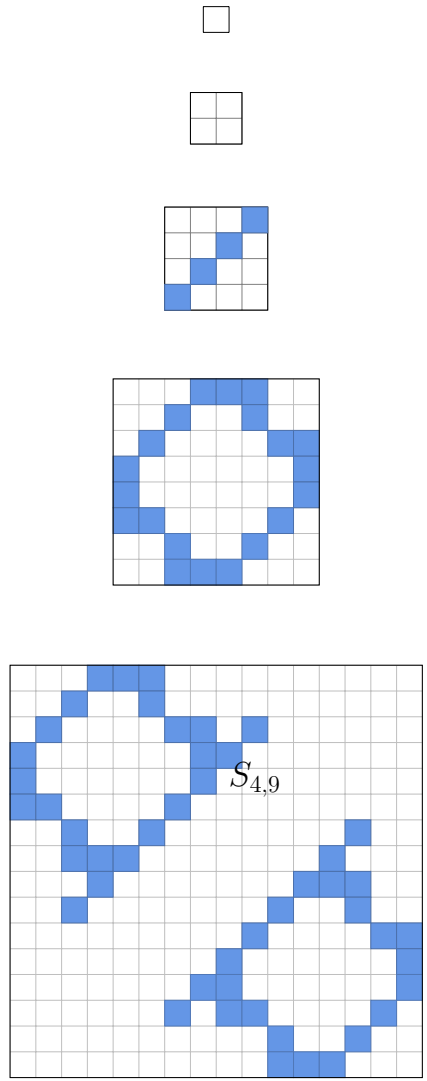


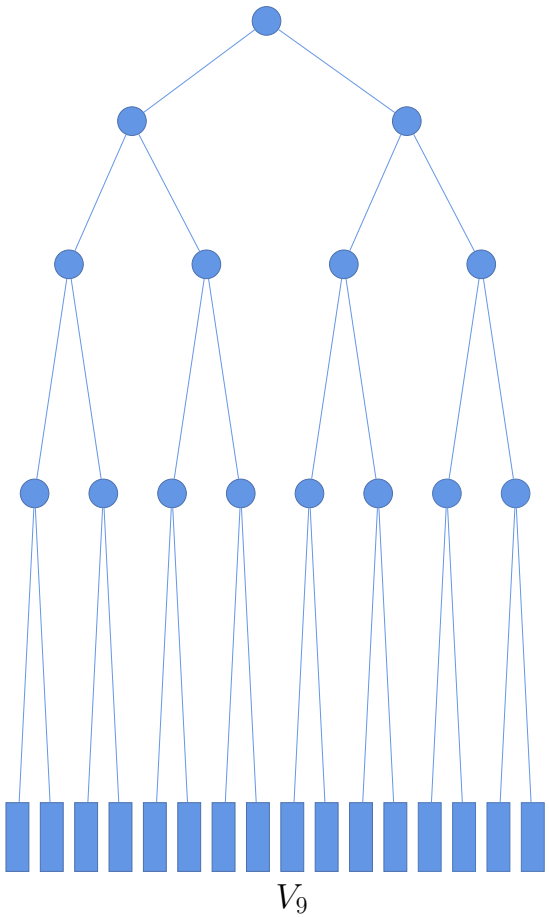
$$A_{ij} = U_i \, S_{ij} \, V_j^t$$

Triplet $(\widehat{U}, \widehat{S}, \widehat{V})$ representation







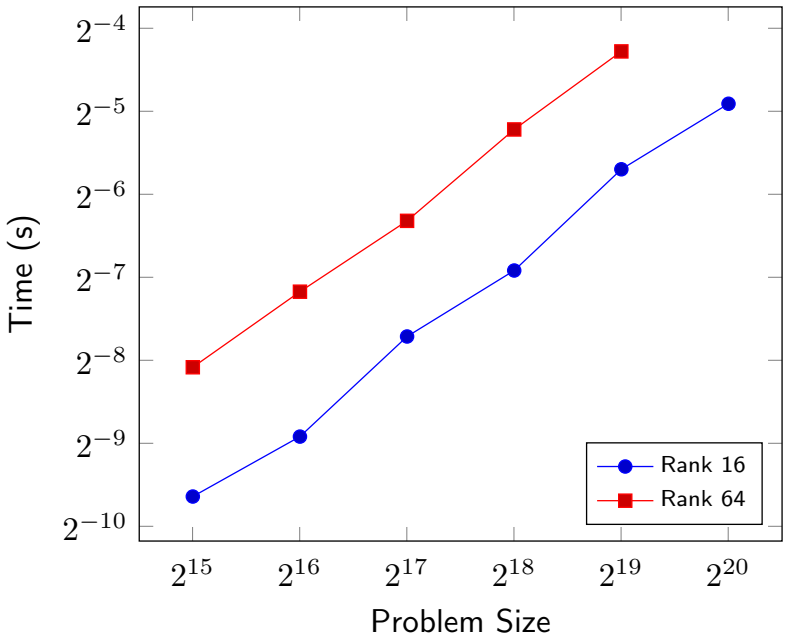


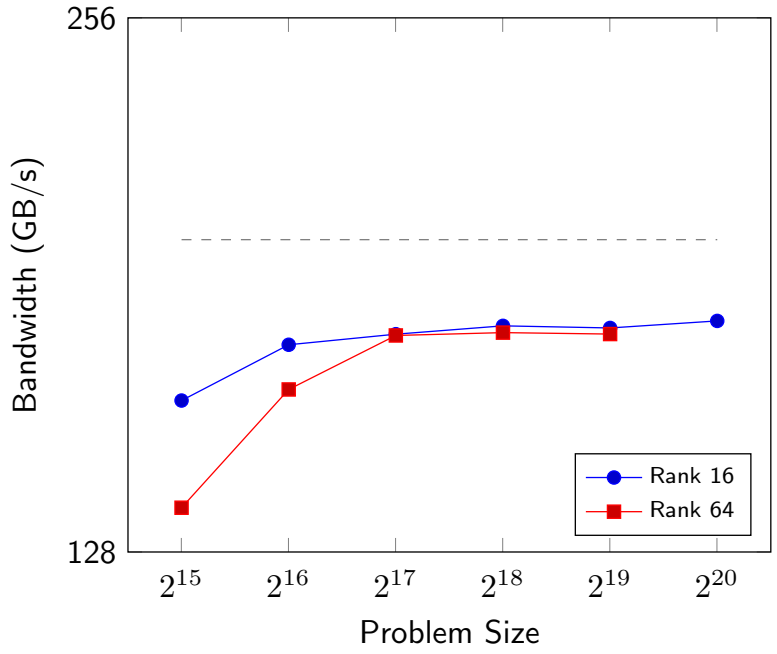
Hierarchical matrix vector multiplication (HMV)

1: **procedure** UPSWEEP(V, F, x, $\widehat{x}$)
2: $q = \text{heightof}(\widehat{x})$ ▷ *leaf level, log(n/m)*
3: $\widehat{x}^{(q)} = \text{gemvBatched}\left(\frac{n}{m}, V^T, (\texttt{batch})\, x\right)$
4: **for** $l = q \to 1$ ▷ *up the* $\widehat{V}$ *tree*
5: $N = n/m/2^{q-l}$
6: $Fx = \text{gemvBatched}(N,\ F^{(l)^T},\ \widehat{x}^{(l)})$
7: $\widehat{x}^{(l-1)} = \text{segmentedAdd}\left(Fx,\ 0\!:\!2\!:\!\frac{N}{2}\right)$

1: **procedure** DOWNSWEEP(U, E, $\widehat{y}$, y)
2: $q = \text{heightof}(\widehat{y})$ ▷ *leaf level, log(n/m)*
3: **for** $l = 1 \to q$ ▷ *down the* $\widehat{U}$ *tree*
4: $N = n/m/2^{q-l}$
5: $yl = \text{segmentedExpand}\left(\widehat{y}^{(l-1)}, [0\!:\!\frac{N}{2} - 1] \otimes [1\ 1]\right)$
6: $\widehat{y}^{(l)} = \text{gemvBatched}(N,\ E^{(l)},\ yl)$
7: $y = \text{gemvBatched}\left(\frac{n}{m}, U, \widehat{y}^{(q)}\right)$

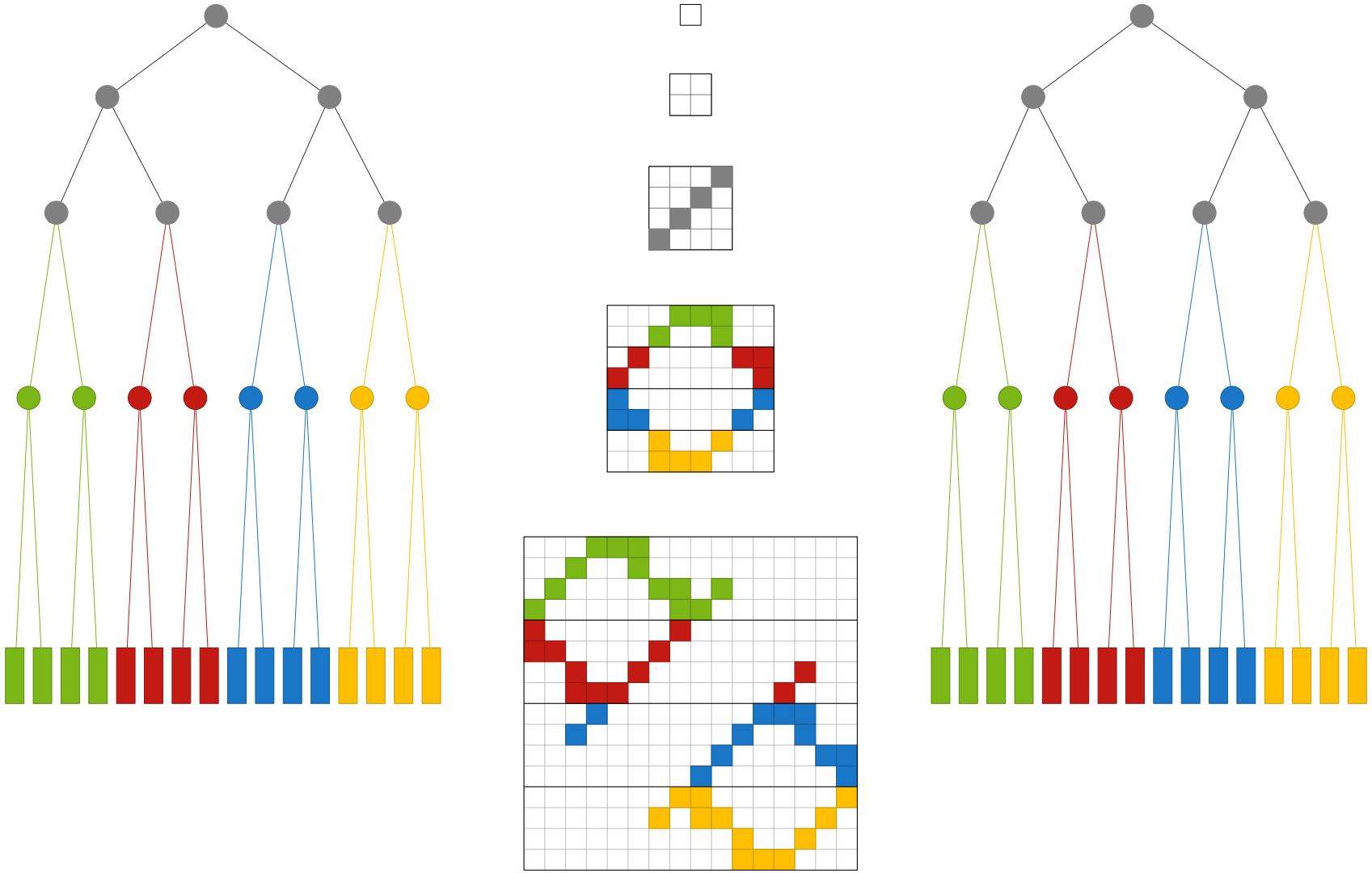
Single GPU performance



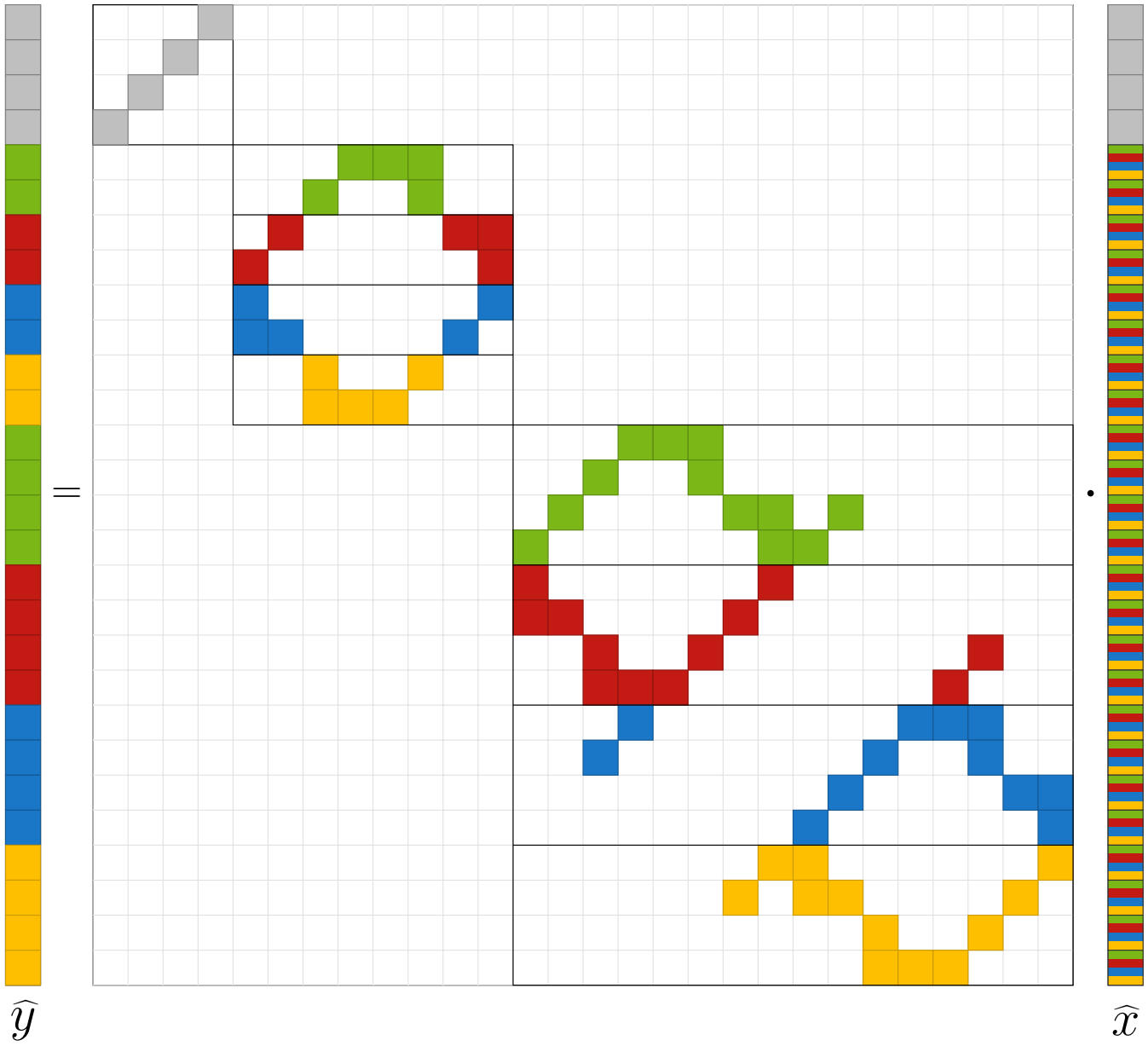


- Kernel matrix from a 2D spatial point distribution
- $\epsilon \approx 10^{-4}$ and $\epsilon \approx 10^{-7}$ for $k = 16$ and 64, repectively
- 90% of peak bandwidth achieved, corresponding to 50 GFLOPS/s on a Tesla K40 GPU
- $O(N)$ growth

MultiGPU representation



MultiGPU HMV



MultiGPU Upsweep

Algorithm Multi-GPU upsweep algorithm for forming $\widehat{x}$

1: **procedure** MULTIUPSWEEP(V, F, x, $\widehat{x}$)
2: $p = \text{rootHeight}(\widehat{x})$ ▷ *levels handled by root GPU*
3: $\text{Scatter}(x, n/\text{nGPUs}, x_i, \text{root})$
4: $\text{upsweep}(V_i, F_i, x_i, \widehat{x}_i)$ ▷ *GPU i handles a subtree*
5: $\text{AllGather}(\widehat{x}_i, \text{sizeof}(\widehat{x}_i), \widehat{x})$
6: **if** $i == \text{root}$
7: **for** $l = p \rightarrow 1$ ▷ *up the root GPU tree*
8: $Fx = \text{gemvBatched}(2^l, F_{\text{root}}^{(l)\,T}, \widehat{x}_{\text{root}}^{(l)})$
9: $\widehat{x}_{\text{root}}^{(l-1)} = \text{segmentedAdd}\big(Fx, 0\!:\!2\!:\!2^l/2\big)$

MultiGPU matrix tree multiplication

Algorithm Multi-GPU Matrix Tree Multiplication for $\widehat{y}$

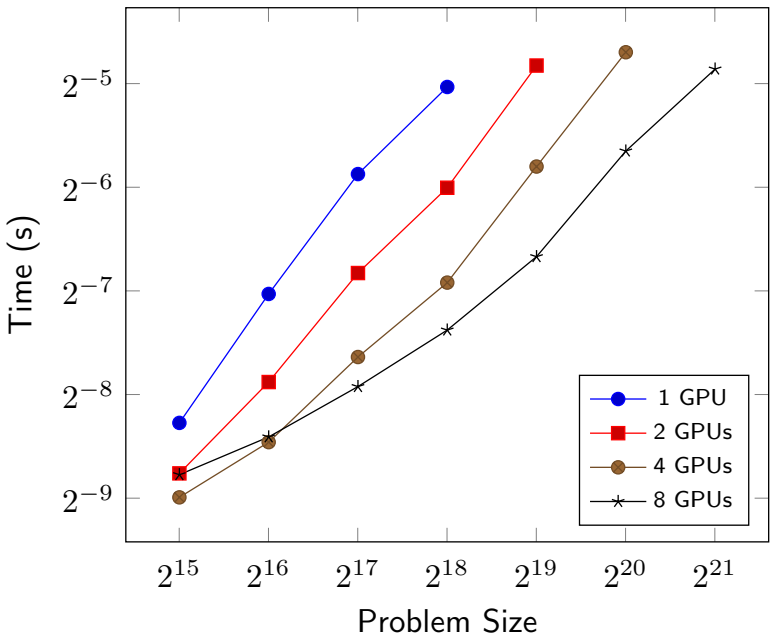
procedure TreeMultiply(S, $\widehat{x}$, $\widehat{y}$)
  $p_i = q_i - (q - p)$ ▷ *levels not handled by GPU i*
  in parallel for $l = q \to p_i$
    blockSparseMV($S_i^{(l)}, \widehat{x}, \widehat{y}_i^{(l)}$)
    if $i ==$ root
      in parallel for $l = p \to 1$
        blockSparseMV($S_{\text{root}}^{(l)}, \widehat{x}, \widehat{y}_{\text{root}}^{(l)}$)

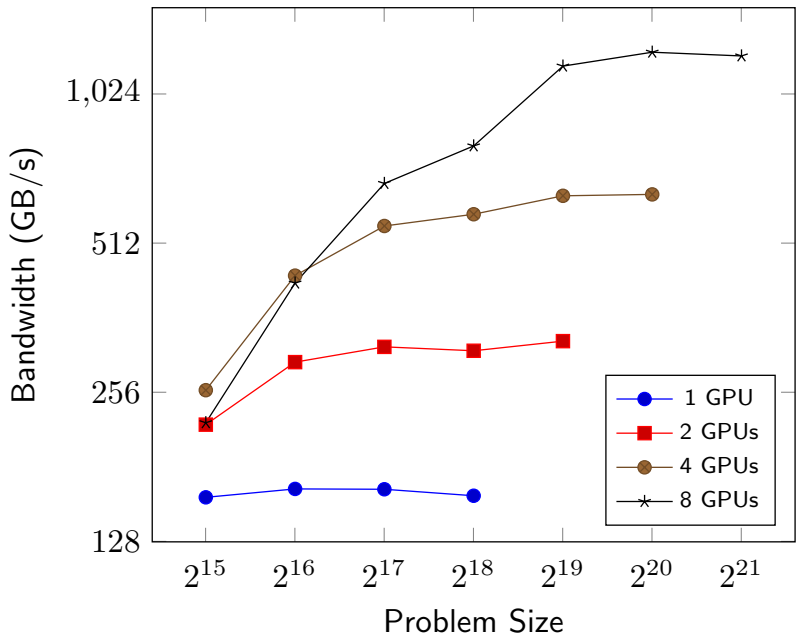
MultiGPU Downsweep

Algorithm Multi-GPU downsweep for computing y

1: **procedure** MULTIDOWNSWEEP(U, E, $\widehat{y}$, y)
2: $p = \text{rootHeight}(\widehat{y})$ ▷ *levels handled by root GPU*
3: **if** $i == \text{root}$
4: **for** $l = 1 \rightarrow p$ ▷ *down the root GPU tree*
5: $yl = \text{segmentedExpand}(\widehat{y}_{\text{root}}^{(l-1)}, [0 : 2^{l-1} - 1] \otimes [1\ 1])$
6: $\widehat{y}_{\text{root}}^{(l)} = \text{gemvBatched}(2^l, E_{\text{root}}^{(l)}, yl)$
7: $\text{Scatter}(\widehat{y}_{\text{root}}^{(p)}, 2^p k^{(p)}/\text{nGPUs}, \widehat{y}_i^{(p_i)}, \text{root})$
8: $\text{downsweep}(U_i, E_i, \widehat{y}_i, y_i)$ ▷ *GPU i handles a subtree*
9: $\text{Gather}(y_i, n/\text{nGPUs}, y, \text{root})$

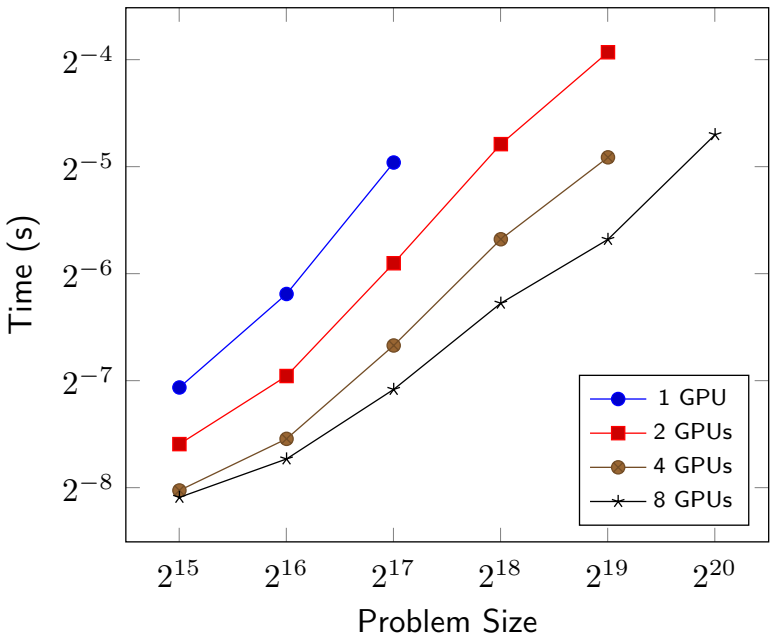
MultiGPU performance

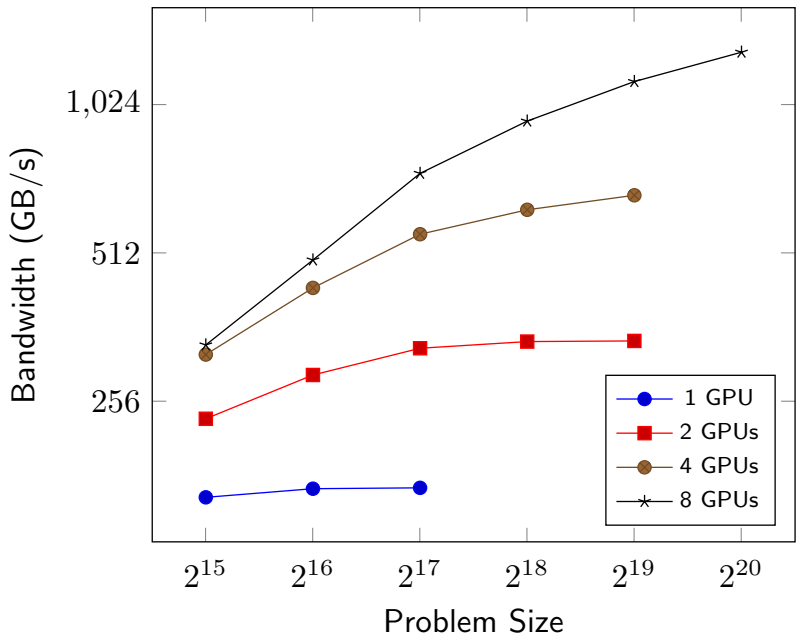




- Kernel matrix from a 3D spatial point distribution
- $\epsilon \approx 10^{-3}$ with $k = 27$ at all levels

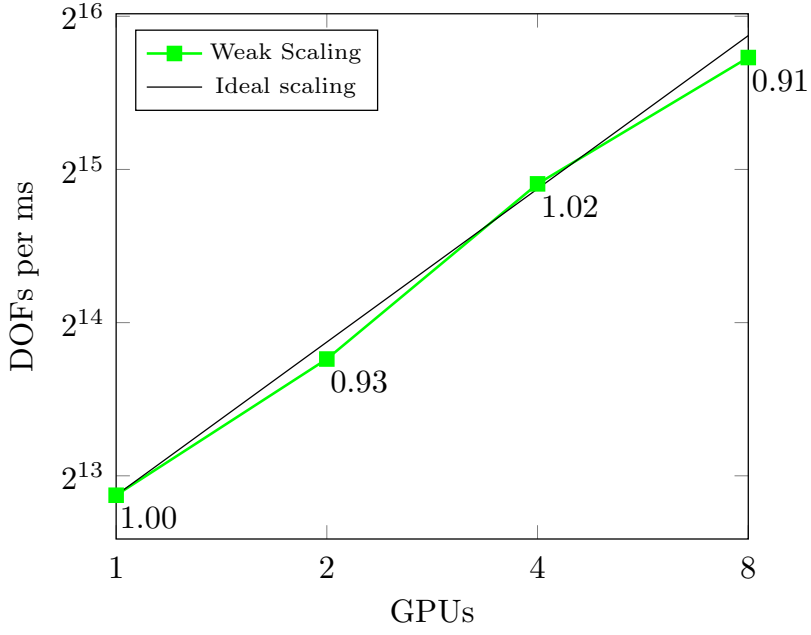
MultiGPU performance (k = 64)

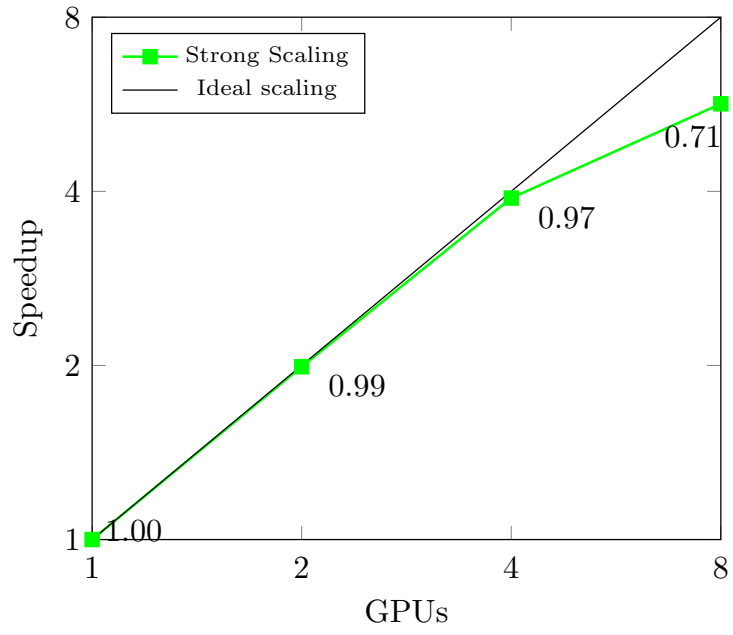




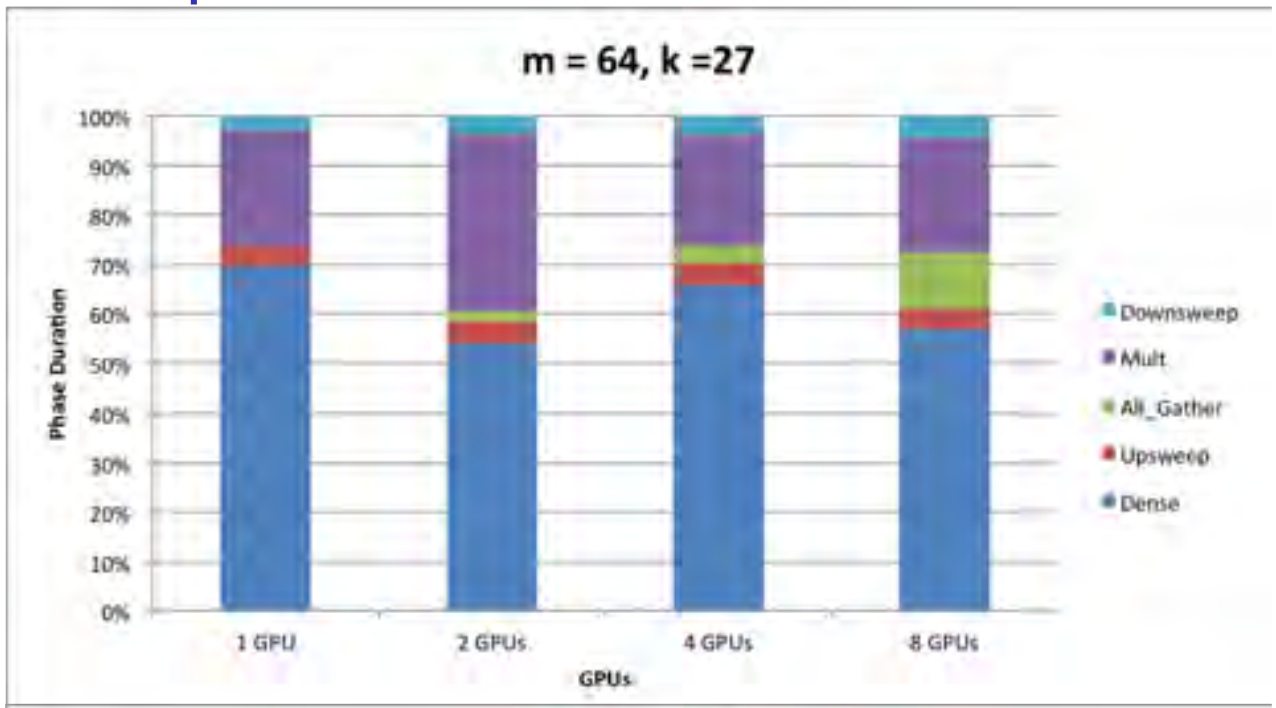
- Kernel matrix from a 3D spatial point distribution
- $\epsilon \approx 10^{-4}$ with $k = 64$ at all levels

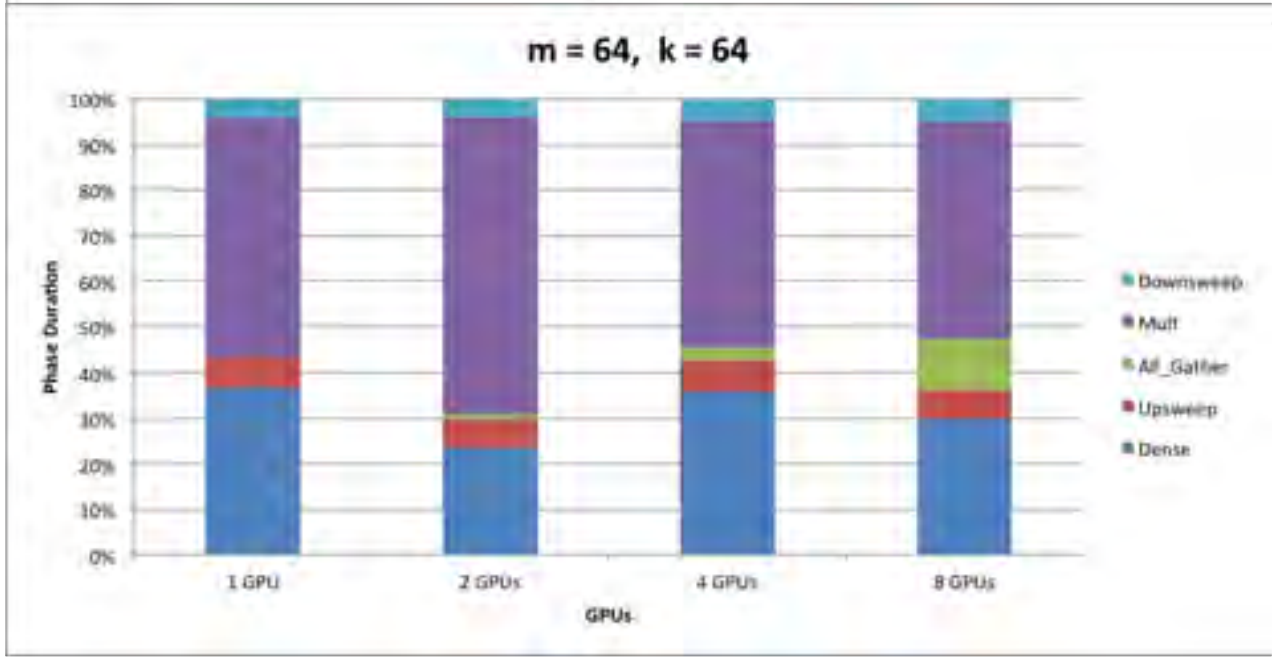
MultiGPU scalability





Computational profile of HMV





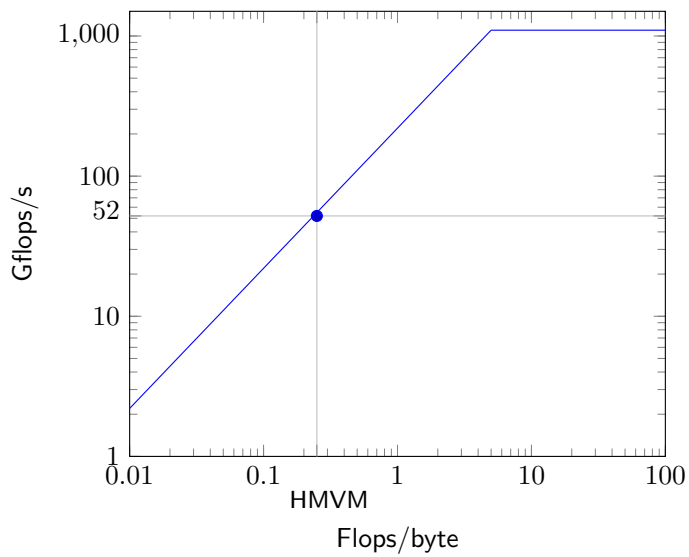
Algebraic compression

Matrix compression involves:

- orthogonalize bases: upsweep with QR of interlevel transfer matrices
- find optimal bases: downsweep with QR of matrix block rows
- truncate bases: upsweep with SVD of interlevel transfer matrices
- projection: batched matrix-matrix multiplications

Higher arithmetic intensity than the HMV sweeps

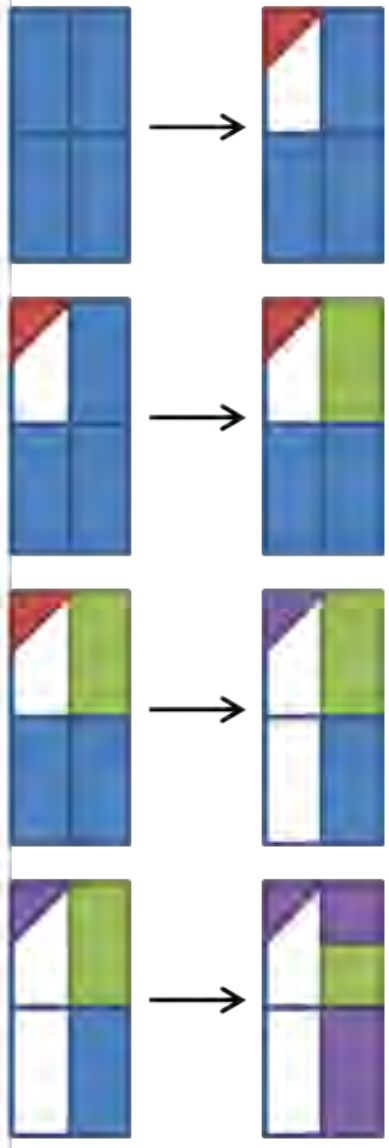
- better FLOPS/s performance expected on GPUs



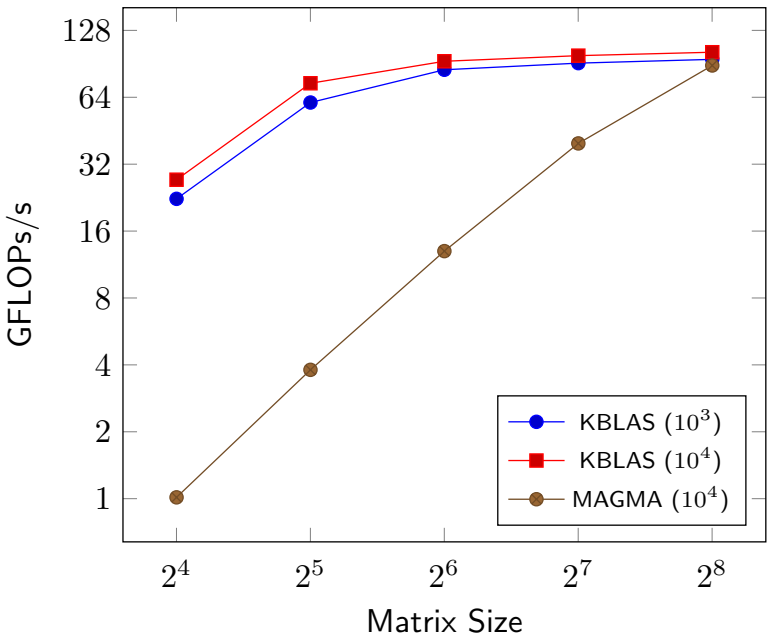
QR on the GPU

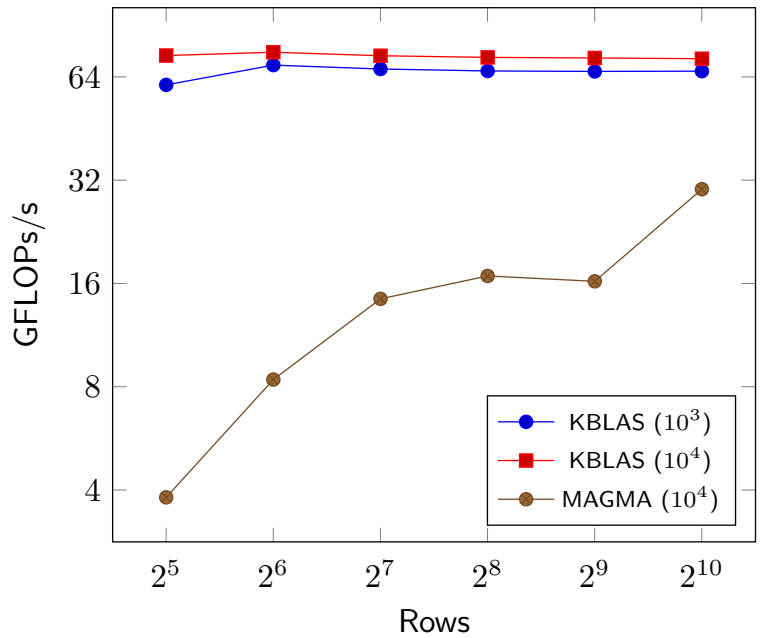
Algorithm 1 The QR algorithm.

procedure QR(A)
  for each panel **do**
    QR decomposition of diagonal sub-panel
    for each trailing sub-panel **do**
      Update using householder reflectors.
    for each lower sub-panel **do**
      QR decomposition of sub-panel with diagonal block.
      for each trailing lower sub-panel **do**
        Update using householder reflectors.



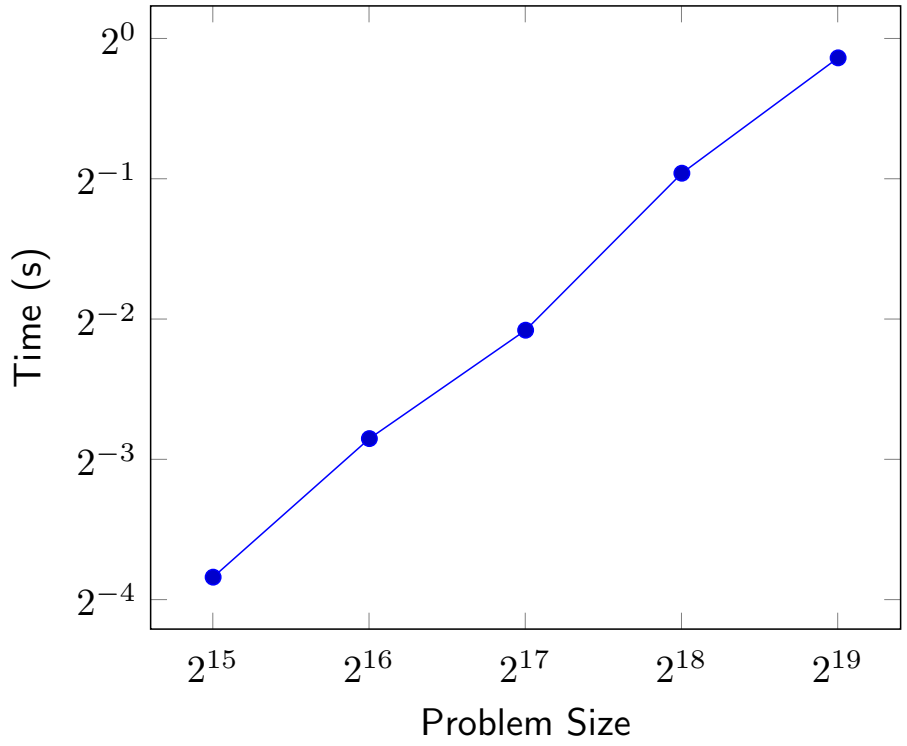
Batched QR performance



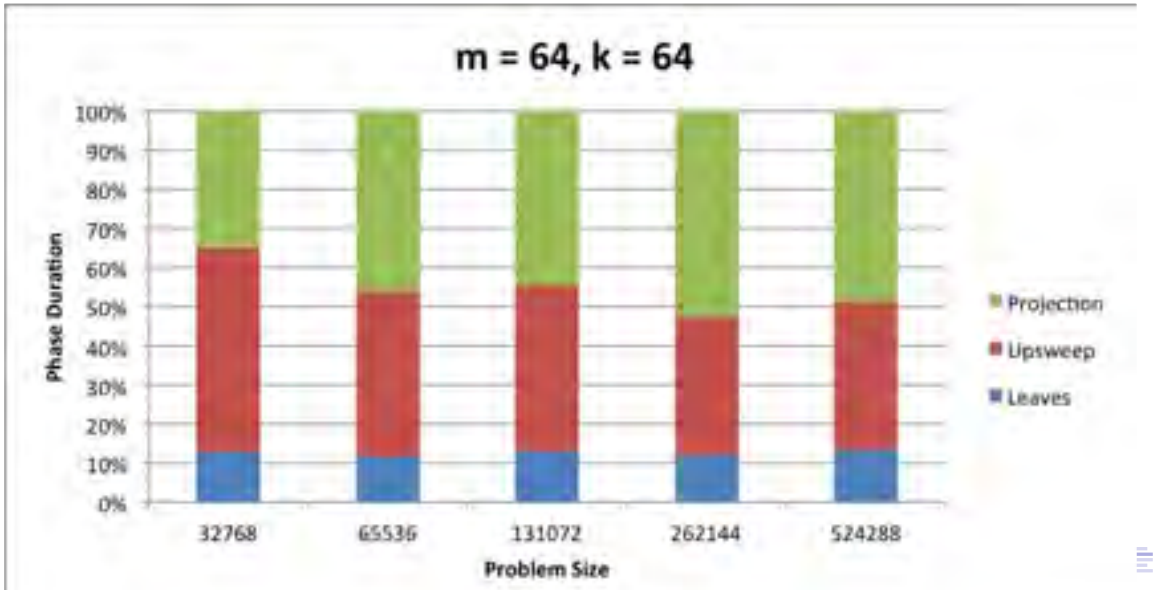


- $>$ 100 GLOPS/s achieved (even on relatively small matrices, e.g., 64×64)

Basis orthogonalization



$\sim 200\,\text{GFLOPS/s}$



High dimensional integration

Multivariate normal integration:

$$\Phi_n(a, b; \Sigma) = \int_a^b \phi_n(x; \Sigma)\, \mathrm{d}x$$
$$= \frac{1}{\sqrt{|\Sigma|(2\pi)^n}} \int_{a_1}^{b_1} \int_{a_2}^{b_2} \cdots \int_{a_n}^{b_n} e^{-\frac{1}{2} x^t \Sigma^{-1} x}\, \mathrm{d}x_n \cdots \mathrm{d}x_1$$

Computational approaches when d is moderate to high:

- Monte Carlo methods
- quasi-Monte Carlo
- sparse grids

Seek to exploit a hierarchical structure of the covariance matrix to compute approximations of the MVN integral efficiently.

Hierarchical decomposition of MVN integrals

Consider a one-level hierarchical Cholesky factorization of $\Sigma = LL^t$:

$$L = \begin{bmatrix} L_1 & \\ UV^t & L_2 \end{bmatrix}$$

Transformation 1:

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} L_1 & \\ UV^t & L_2 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix},$$

$$\Phi_n(a, b, \Sigma) = \frac{1}{\sqrt{(2\pi)^n}|\Sigma|^{\frac{1}{2}}} \int_a^b e^{-\frac{1}{2}x^t\Sigma^{-1}x} \, \mathrm{d}x$$

$$= \frac{1}{\sqrt{(2\pi)^n}} \frac{|L|}{|\Sigma|^{\frac{1}{2}}} \int_{a_1 \le L_1\alpha_1 \le b_1} e^{-\frac{1}{2}\alpha_1^t\alpha_1} \int_{a_2(\alpha_1) \le L_2\alpha_2 \le b_2(\alpha_1)} e^{-\frac{1}{2}\alpha_2^t\alpha_2} \, \mathrm{d}\alpha_2 \, \mathrm{d}\alpha_1$$

Hierarchical decomposition of MVN integrals

Transformation 2:

$$\theta_1 = L_1\alpha_1, \quad \theta_2 = L_2\alpha_2$$

$$\begin{aligned}\Phi_n(a,b,\Sigma) &= \frac{1}{\sqrt{(2\pi)^n}} \frac{1}{|\Sigma_1|^{\frac{1}{2}}} \int_{a_1}^{b_1} e^{-\frac{1}{2}\theta_1^t\Sigma_1^{-1}\theta_1} \frac{1}{|\Sigma_2|^{\frac{1}{2}}} \int_{a_2'}^{b_2'} e^{-\frac{1}{2}\theta_2^t\Sigma_2^{-1}\theta_2}\, \mathrm{d}\theta_2\, \mathrm{d}\theta_1 \\ &= \int_{a_1}^{b_1} \phi_{\frac{n}{2}}(\theta_1,\Sigma_1) \int_{a_2'}^{b_2'} \phi_{\frac{n}{2}}(\theta_2,\Sigma_2)\, \mathrm{d}\theta_2\, \mathrm{d}\theta_1\end{aligned}$$

where

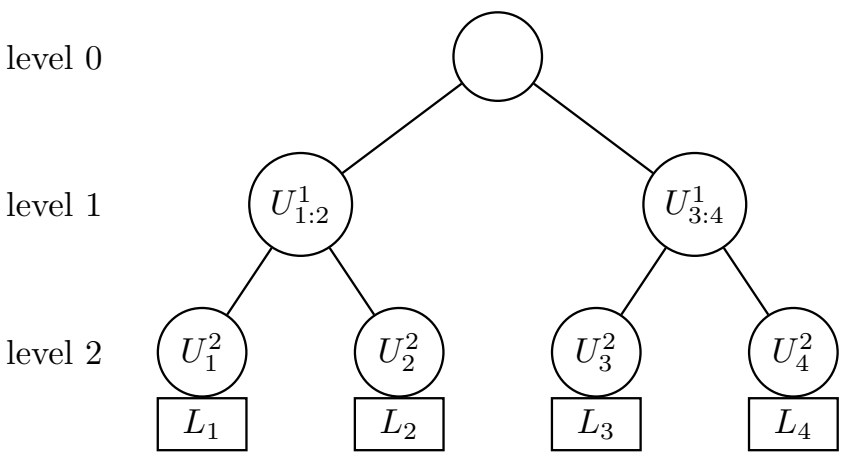
$$a_2' = a_2 - UV^tL_1^{-1}\theta_1 \quad \text{and} \quad b_2' = b_2 - UV^tL_1^{-1}\theta_1.$$

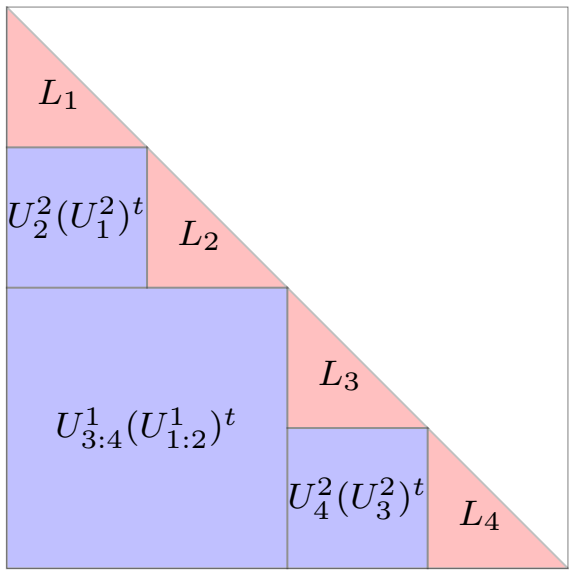
- At the cost of $O(kn)$ operations, the the $n-$dimensional integral is split into the product of two intgerals of half the size.

Continuing the decomposition

- Decompose until sufficiently small-sized blocks ($m = n/2^q$) are reached

$$\Phi_n = \int_{a_1}^{b_1} \phi_m(\theta_1, \Sigma_1) \int_{a_2'}^{b_2'} \phi_m(\theta_2, \Sigma_2) \dots \int_{a_r'}^{b_r'} \phi_m(\theta_r, \Sigma_r)\, \mathrm{d}\theta_r \dots\, \mathrm{d}\theta_2\, \mathrm{d}\theta_1$$



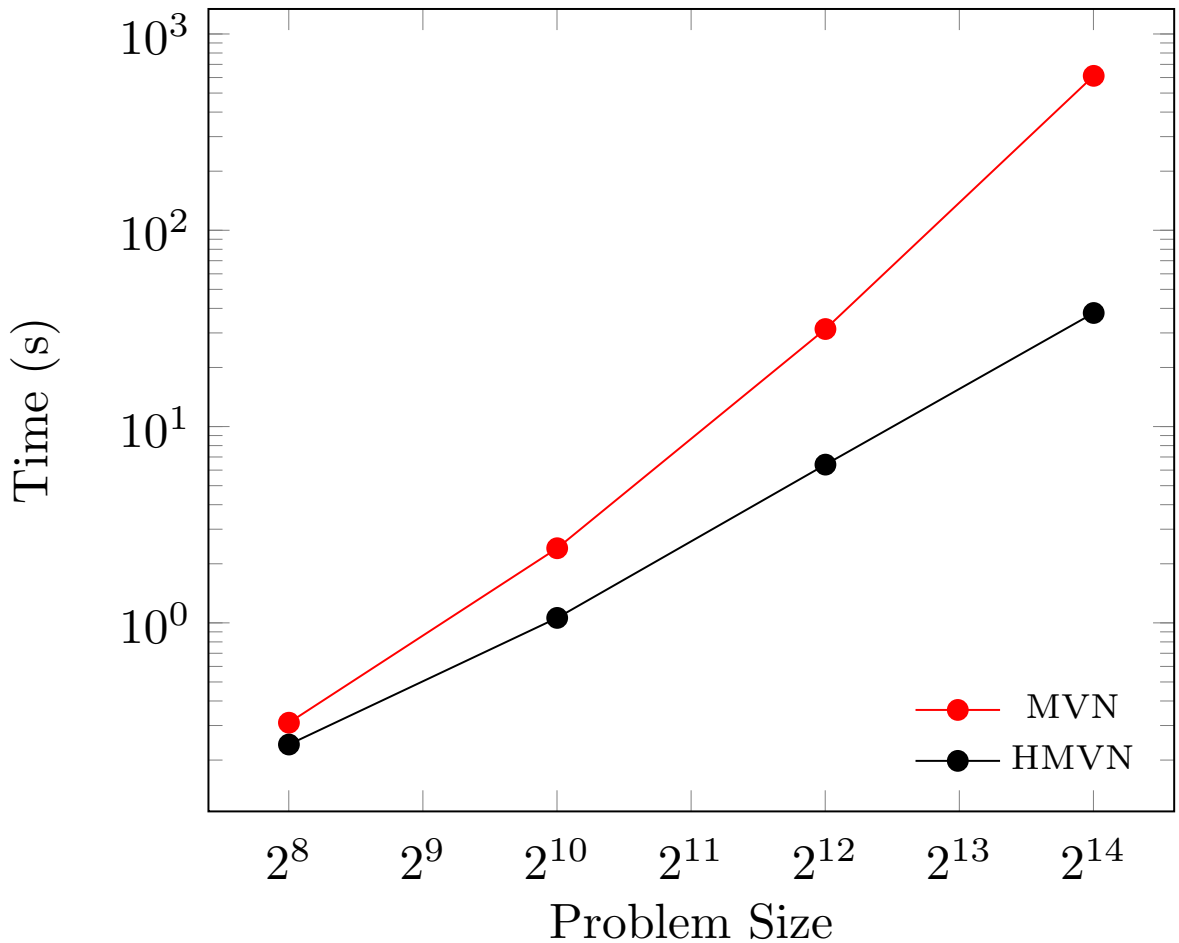


Algorithm

function HMVN(L_H, a, b, N)
  for $s = 1 : M$
    Initialize v and generate a uniform random $\delta \in [0, 1]^n$
    for $t \in \text{inorder}(\text{tree}(L_H))$
      if isleaf(t) ▷ Direct evaluation
        $r =$ range of t
        $[v_t, y_t] = \text{mvn}(L_t, a(r), b(r), N/M, \delta(r))$
        $v = v. * v_t$
        $y(r) = y_t$
      else ▷ Update integration limits
        $p =$ level(t) + 1
        $j =$ range(leftchild(t)), $i =$ range(rightchild(t))
        $\Delta = U_i^p * ((V_j^p)^t * y(j))$
        $a(i) = a(i) - \Delta$, $b_j = b(i) - \Delta$
    $\text{estimate}(s) = \text{mean}(v)$
  result $= \text{mean}(\text{estimate})$
  error $= \alpha$ std(estimate)
  return result, error

Performance

Exponential covariance function for a point distribution in $[0,1]^2$
$\epsilon_{\mathcal{H}} \sim 10^{-4}$, $m = 64$, $k \sim 20$ for finest levels blocks
$\epsilon_{\text{rel}} \sim 10^{-3}$



Fast direct elliptic solvers

Consider the block tridiagonal linear system that arises in the discrete Poisson problem on a Cartesian product mesh.

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

Key idea: Approximate each block in a Hierarchical Matrix format to leverage $\mathcal{H}$-matrix arithmetic operations in the inner steps of Block Cyclic Reduction.

$$A = \begin{bmatrix} D_0 & F_0 & & & \\ E_1 & D_1 & F_1 & & \\ & \ddots & \ddots & \ddots & \\ & & E_{n-2} & D_{n-2} & F_{n-2} \\ & & & E_{n-1} & D_{n-1} \end{bmatrix}$$

Accelerated Cyclic Reduction (ACR)

The fundamental reduction operation is a *Schur Complement*:

$$(A_{22} - A_{21}A_{11}^{-1}A_{12})u_{odd} = f$$

Red-black ordering *decouples* the linear system, exposing concurrency.

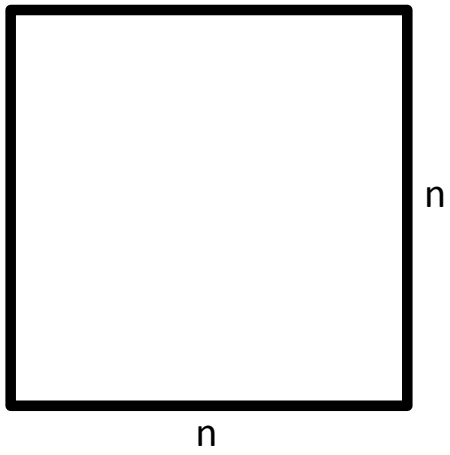
$$PAP^T = \left[\begin{array}{ccccc|ccccc} D_0 & & & & & F_0 & & & & \\ & D_2 & & & & E_2 & F_2 & & & \\ & & D_4 & & & & E_4 & F_4 & & \\ & & & D_6 & & & & E_6 & F_6 & \\ \hline E_1 & F_1 & & & & D_1 & & & & \\ & E_3 & F_3 & & & & D_3 & & & \\ & & E_5 & F_5 & & & & D_5 & & \\ & & & E_7 & & & & & D_7 & \end{array}\right]$$

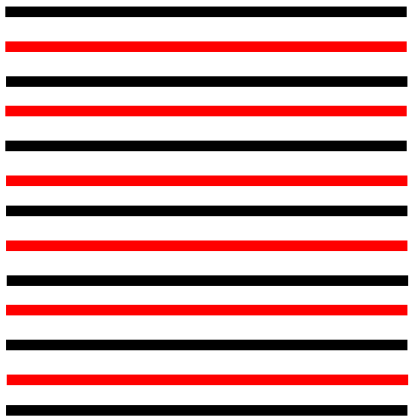
After Schur complementation, system remain block tridiagonal.

$\mathcal{H}$-matrices may be used for block operations.

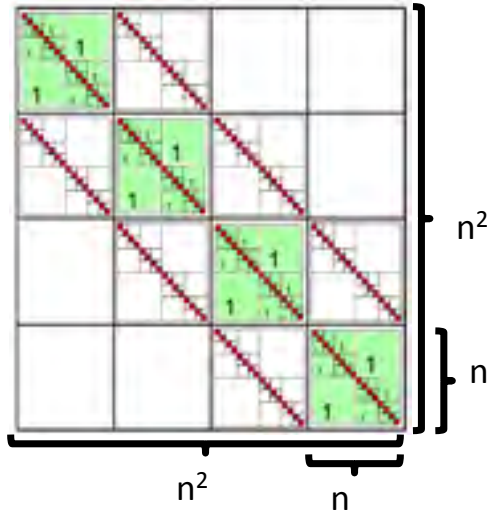
Accelerated Cyclic Reduction (ACR) for 3D elliptic PDEs

$\mathbf{N = n^2}$

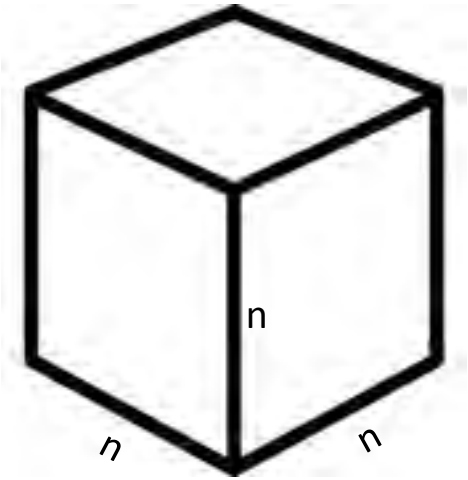




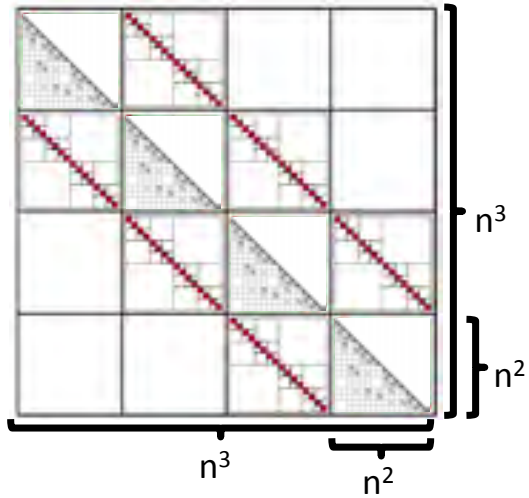
$A =$



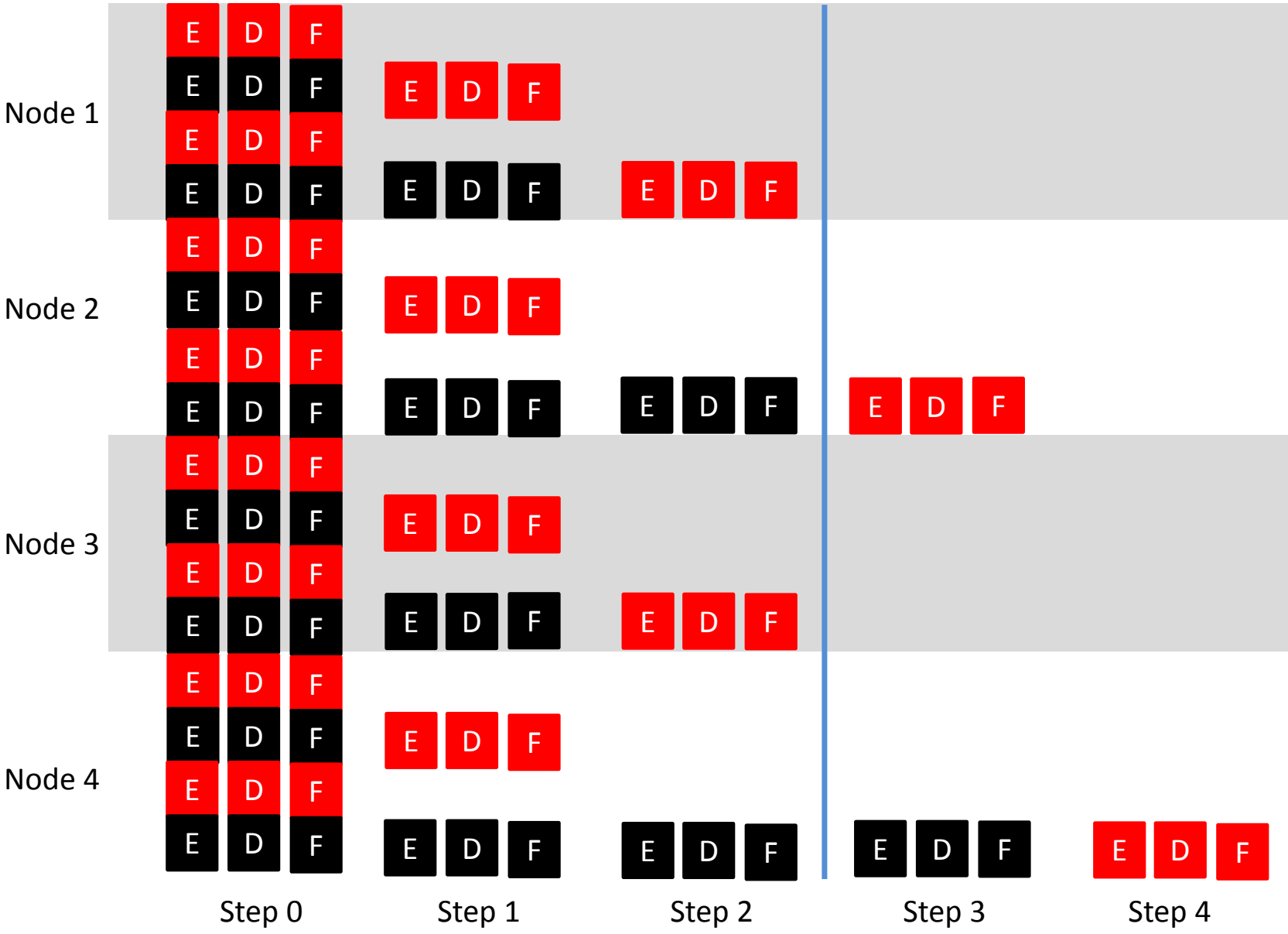
$\mathbf{N = n^3}$



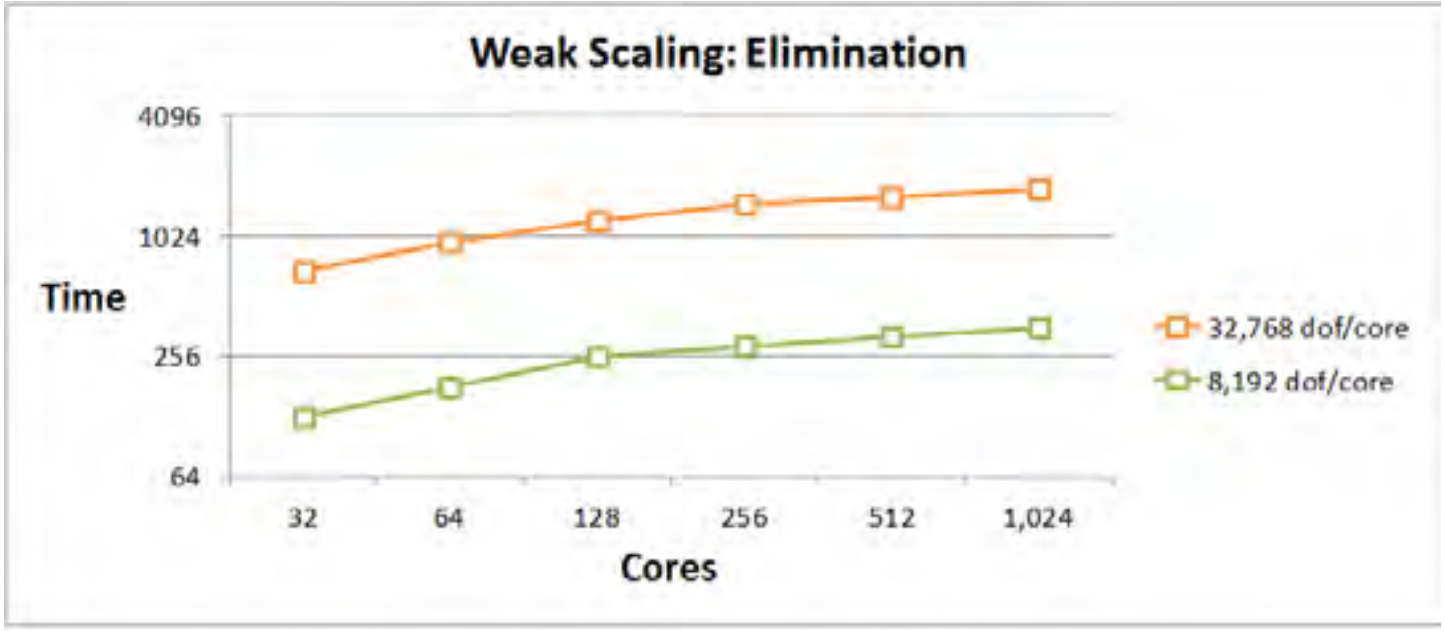
$A =$

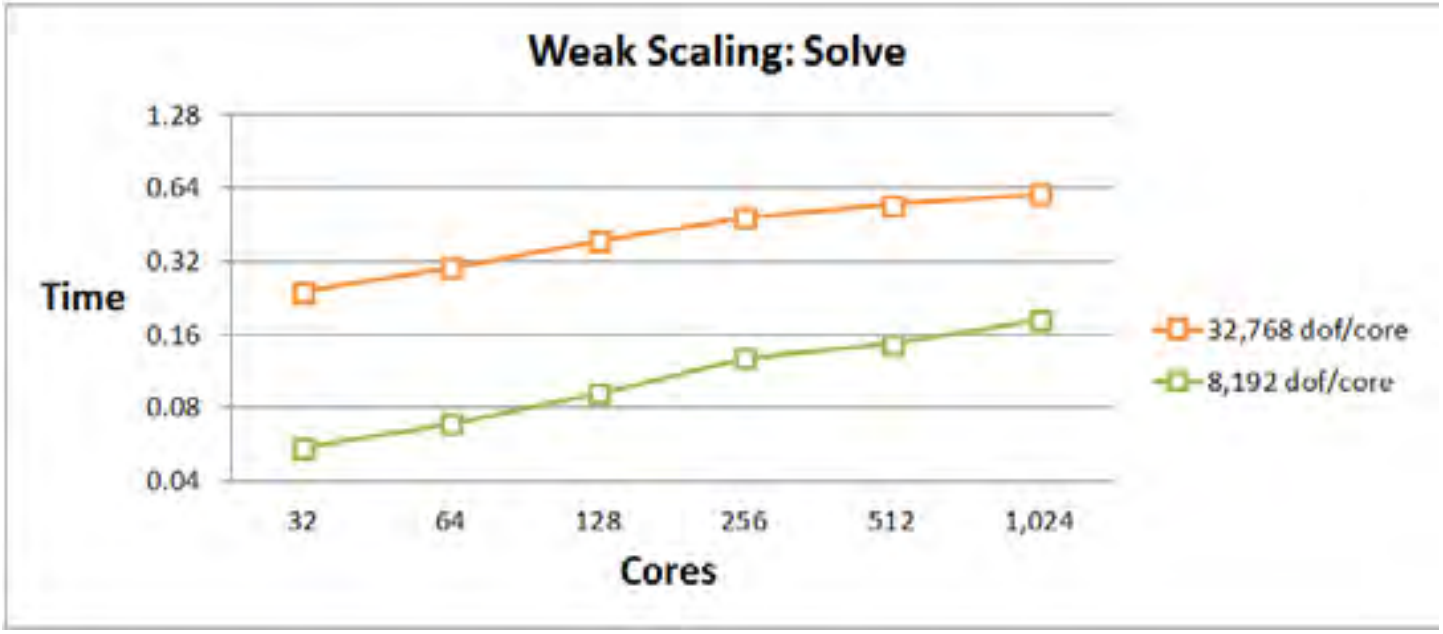


Distributed memory ACR

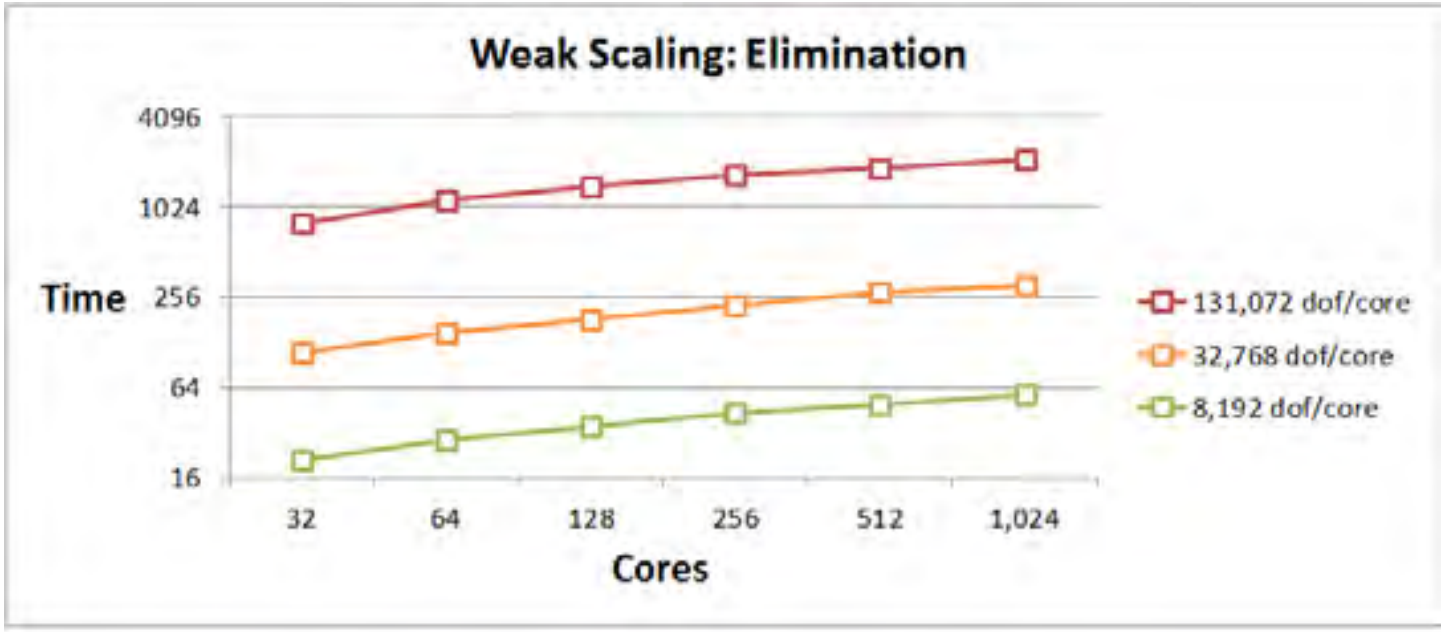


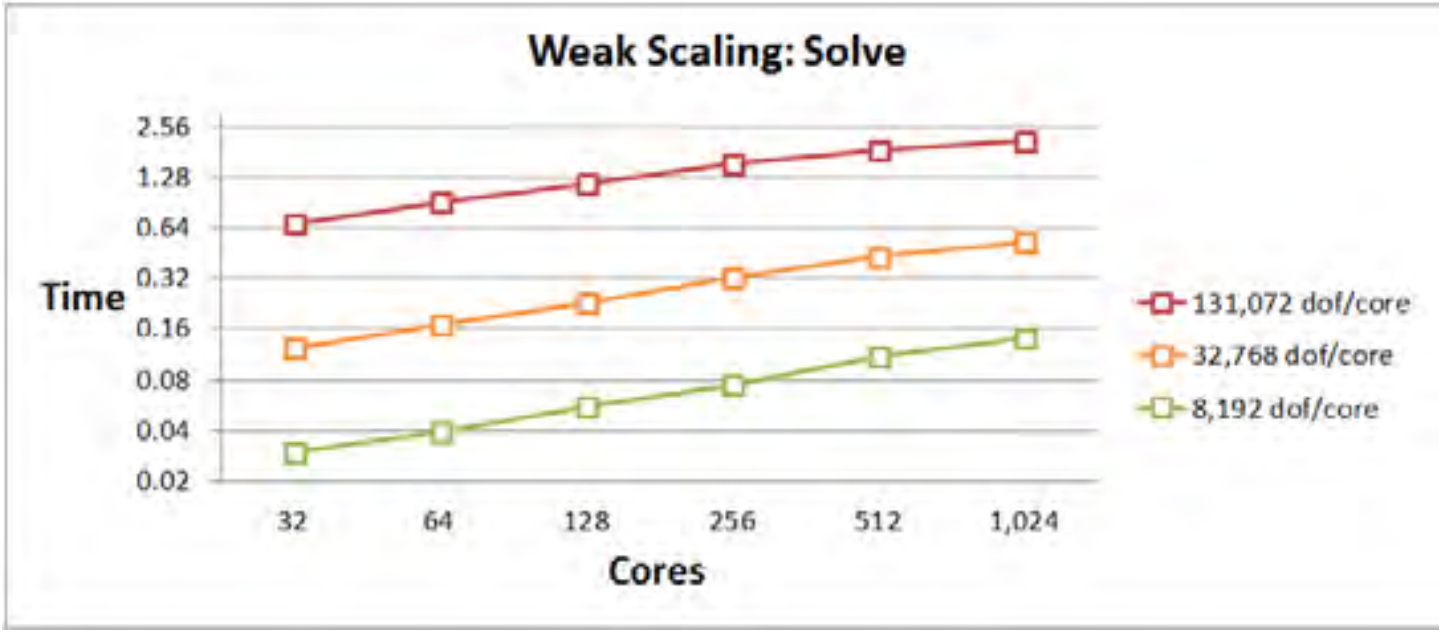
3D Poisson performance (6-digits accurate)





3D Poisson performance (1-digit accurate)

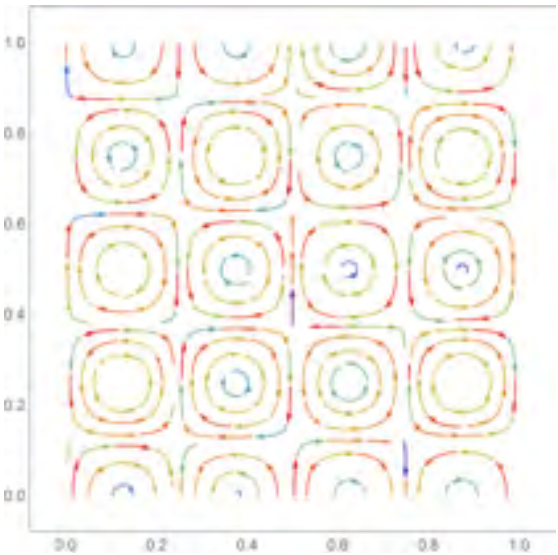


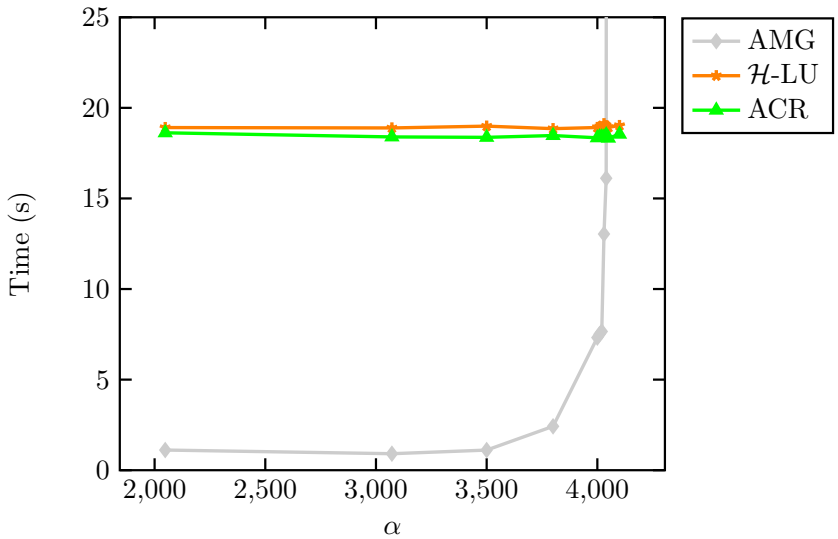


Robustness: convection-diffusion with recirculating flow

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

$$b(\mathbf{x}) = \begin{pmatrix} \sin(4\pi x)\sin(4\pi y) \\ \cos(4\pi x)\cos(4\pi y) \end{pmatrix}$$





Conclusions

- $\mathcal{H}^2$ computations are amenable to efficient processing on GPUs
- open the door for many interesting applications