Sparse Matrix Algorithms

combinatorics + numerical methods + applications

Tim Davis
Texas A&M University

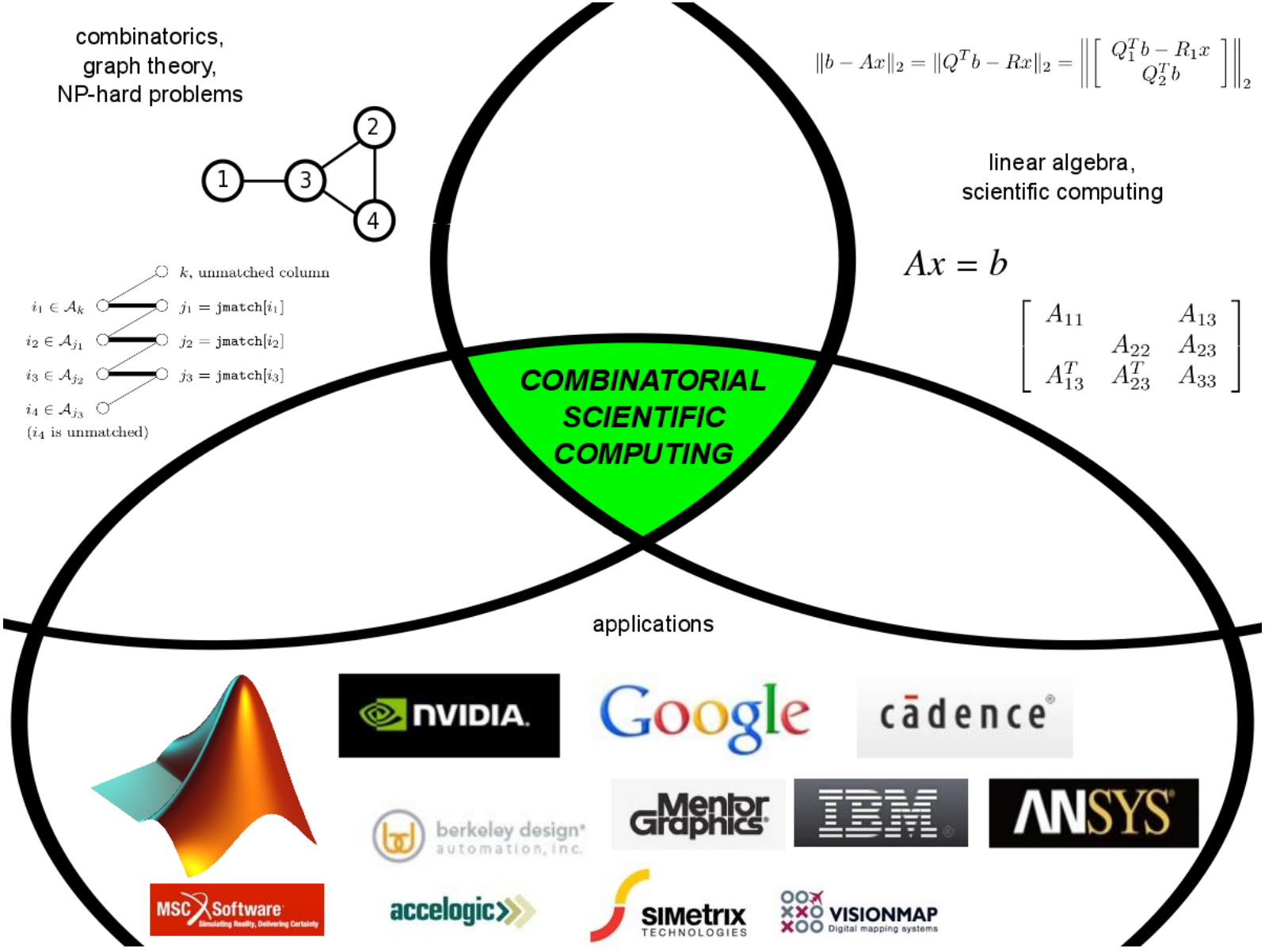
Mar 18, 2016

Univ Tennesee Knoxville, ICL Seminar:

Outline

- **Computer Science + Applied Math =**
  $$\left[\begin{array}{l}\text{high-performance combinatorial scientific computing}\\ \quad +\text{many applications enabled by my contributions}\end{array}\right]$$
- **Sparse matrix algorithms**
- **Contributions to the field**
  - from theory, to algorithms, to reliable software, to applications
  - sparse Cholesky update/downdate (CHOLMOD)
  - approximate minimum degree (AMD)
  - unsymmetric multifrontal LU (UMFPACK)
  - multifrontal QR (SuiteSparseQR)
- **Current work**
  - highly concurrent methods (GPU or massive CPU core)
  - NVIDIA Academic Partner / Texas A&M CUDA Research Center
  - Collaboration with ICL
- **Future vision**

Computer Science + Applied Math = combinatorial scientific computing + applications



Sparse Direct Methods: Algorithms + Code

Impact: new algorithms and useful code

- solve $Ax = b$ when A is sparse
- sparse least-squares problems
- rank estimates, sparse null space bases
- sparse SVD
- solve $Ax = b$, then let A undergo a low-rank change
- fill-reducing orderings, graph partitioning
- all in highly reliable, high performance code
- 3x more reliable than NASA's most extreme effort
- enabling a vast domain of commercial, academic, and government lab applications
- ...

Solve $Lx = b$ with L unit lower triangular; L, x, b are sparse

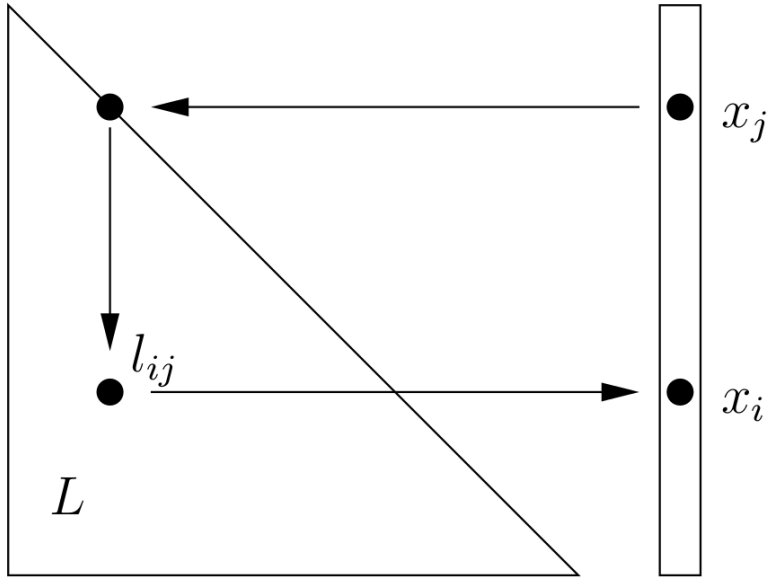
$x = b$
for $j = 0$ to $n - 1$ **do**
  if $x_j \neq 0$
    for each $i > j$ for which $l_{ij} \neq 0$ **do**
      $x_i = x_i - l_{ij} x_j$

- non-optimal time $O(n + |b| + f)$, where f = flop count
- problem: outer loop and the test for $x_j \neq 0$
- solution: suppose we knew $\mathcal{X}$, the nonzero pattern of x
- optimal time $O(|b| + f)$, but how do we find $\mathcal{X}$? (Gilbert/Peierls)

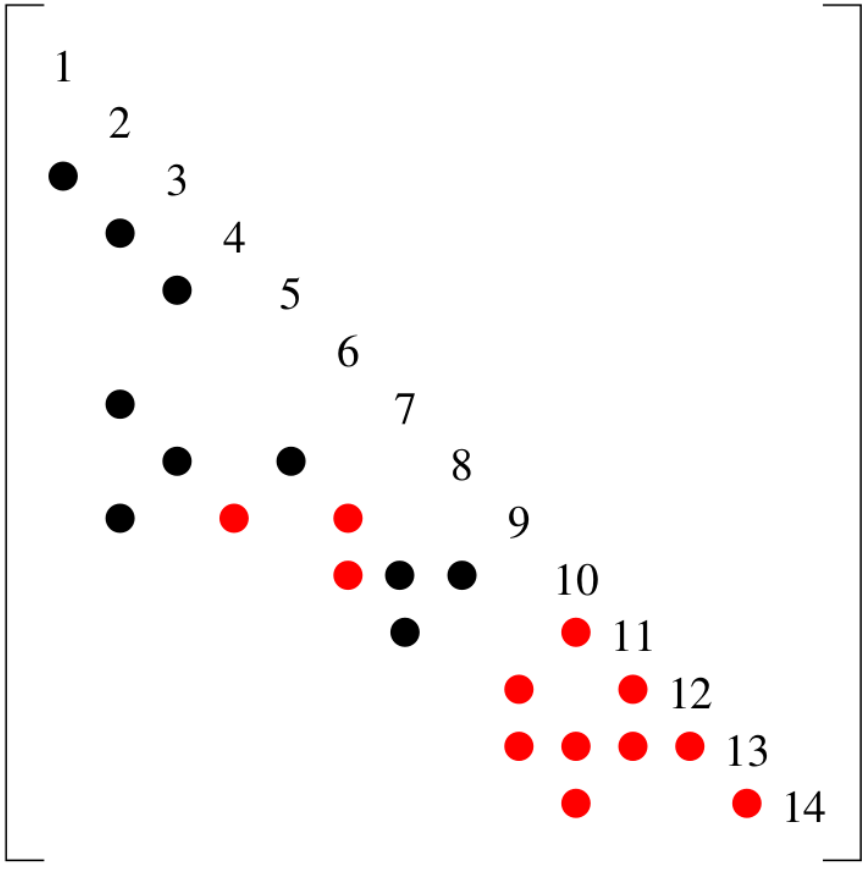
Solve $Lx = b$ with L unit lower triangular; L, x, b are sparse

$x = b$
for $j = 0$ to $n - 1$ **do**
  if $x_j \neq 0$
    for each $i > j$ for which $l_{ij} \neq 0$ **do**
      $x_i = x_i - l_{ij} x_j$

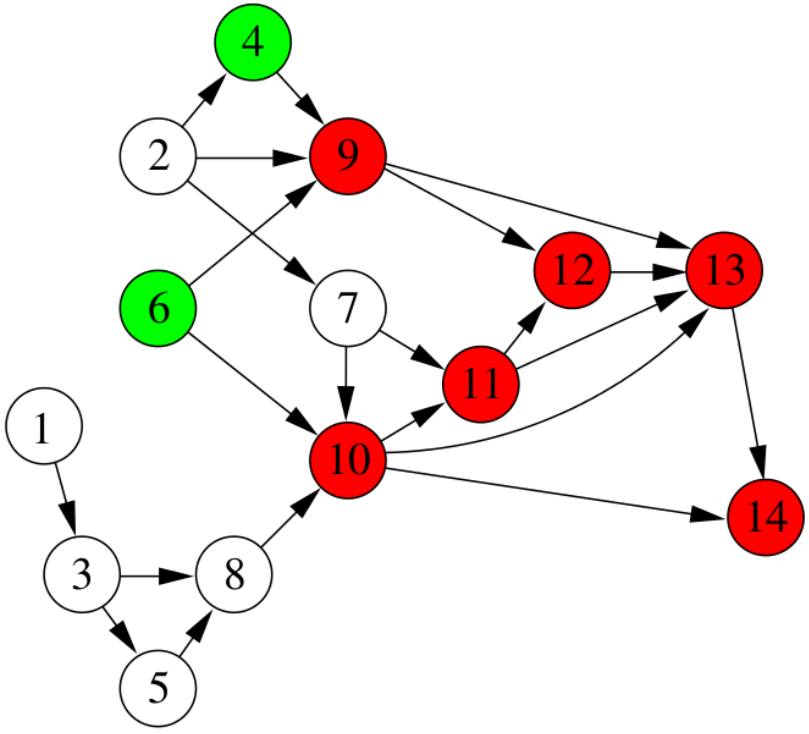
- if $b_i \neq 0$ then $x_i \neq 0$
- if $x_j \neq 0$ and $\exists i(l_{ij} \neq 0)$ then $x_i \neq 0$
- start with pattern $\mathcal{B}$ of b
- graph $\mathcal{L}$: edge (j, i) if $l_{ij} \neq 0$
- $\mathcal{X} = \mathrm{Reach}_{\mathcal{L}}(\mathcal{B})$ (Gilbert/Peierls)



Lower triangular matrix L



Graph G_L



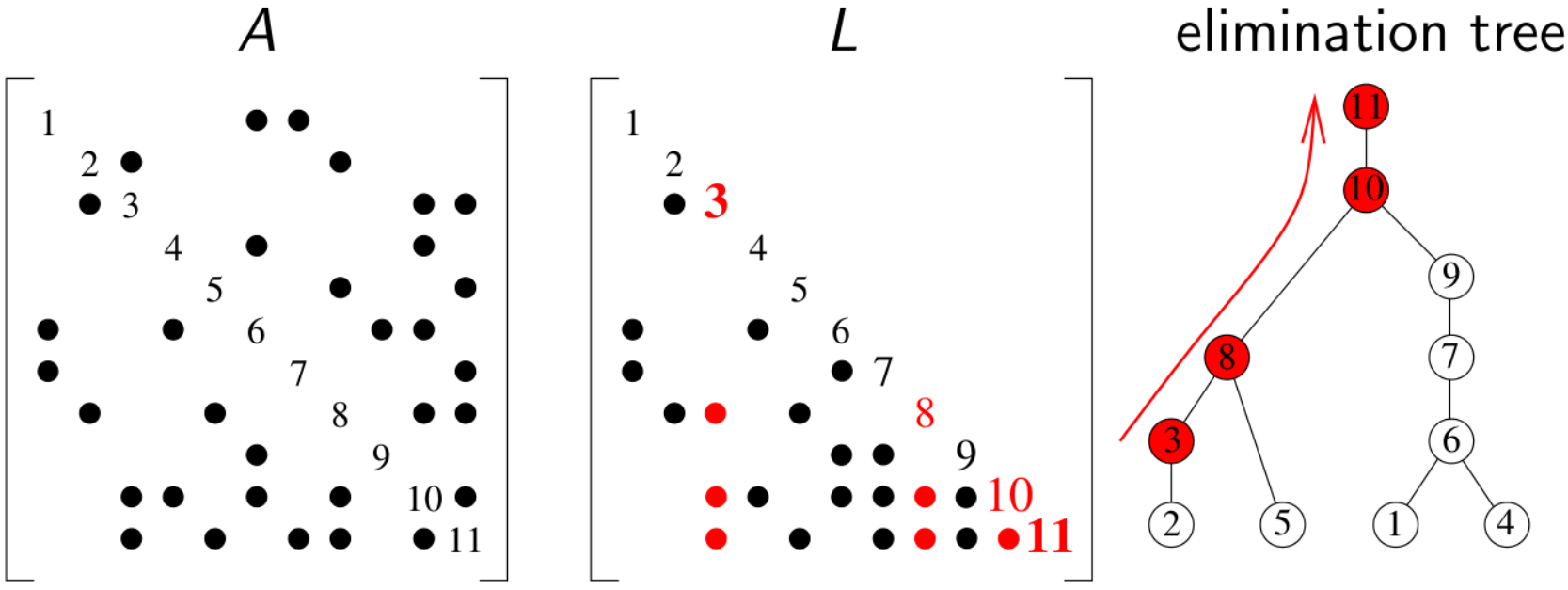
If $\mathcal{B} = \{4, 6\}$ then $\mathcal{X} = \{6, 10, 11, 4, 9, 12, 13, 14\}$

Sparse Cholesky update/downdate

The update/downdate problem:

- Given $A = LL^T$
- A undergoes a low-rank change
- compute $\overline{L}\,\overline{L}^T = A \pm ww^T$
- arises in optimization, crack propagation, robotics, new data in least-squares, short-ciruit power analysis, ...

Sparse Cholesky update/downdate



Key results

- if $\mathcal{L}$ doesn't change: columns in L that change $=$ path from $\min \mathcal{W}$ to root of the etree
- if $\mathcal{L}$ does change, follow the path in etree of $\overline{L}$
- Update/downdate in time proportional to the number of entries in L that change

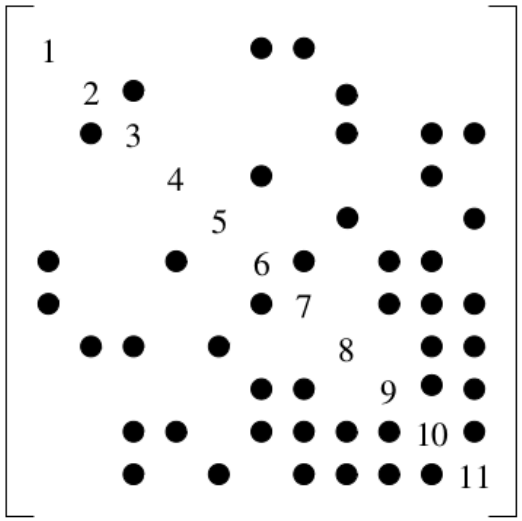
Sparse Cholesky update/downdate

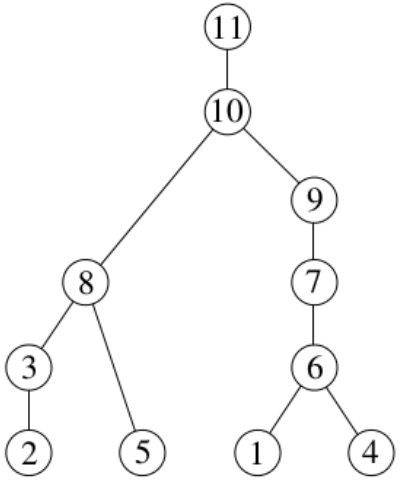
CHOLMOD update/downdate: key results / impact

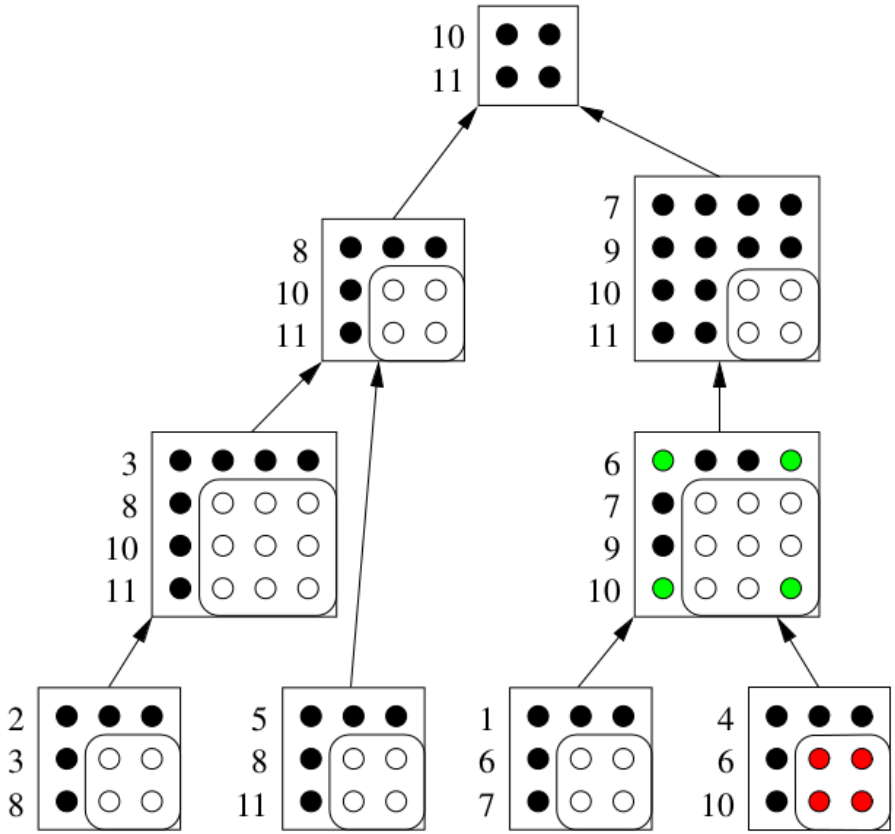
- update/downdate faster than a $Lx = b$ solve for dense b
- example application: LPDASA (Hager and Davis)
  - maintains Cholesky factorization of $A_F A_F^T$ for basis set F
  - update/downdate as basis set changes
- example: g2o (Kümmerle et al)
  - robotics, simultaneous localization and mapping
  - builds a map of its surroundings
  - update/downdate as new images arrive
- example: crack propagation (Pais, Kim, Davis et al)
  - structural engineering problem: crack in aircraft fuselage
  - update/downdate as crack progresses through airframe
- example: short-circuit power analysis (for Mentor Graphics)
  - updating the sparse solve, cut numerics by 100x, total tool: 2x

Multifrontal method

- Classic symmetric multifrontal method (Duff, Reid, others)
- Cliques + elimination tree = sequence of frontal matrices
- Dense factorization within a front; assemble data into parent

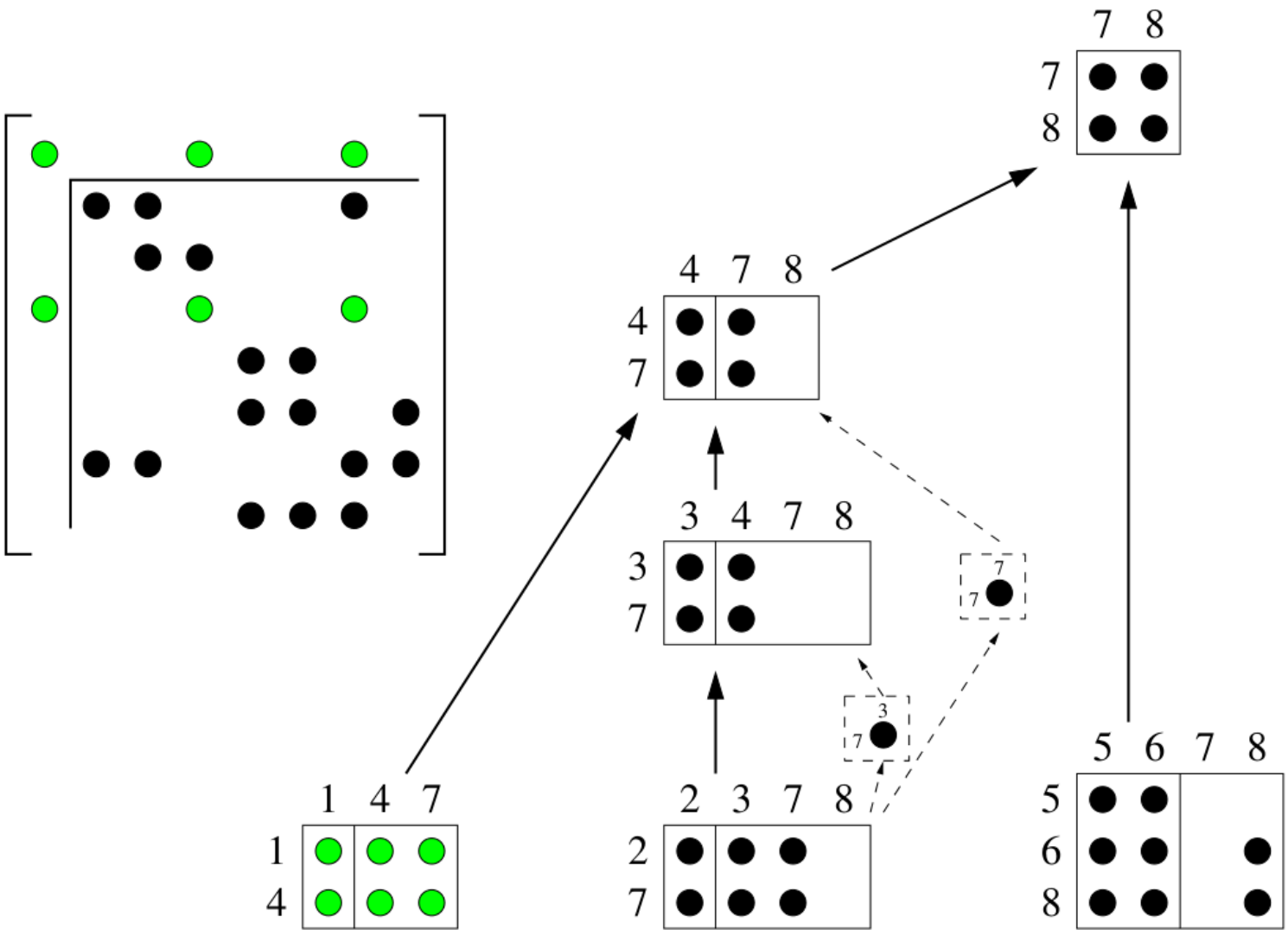






UMFPACK: unsymmetric multifrontal method

- Frontal matrices become rectangular
- Assemble data into ancestors, not just parents



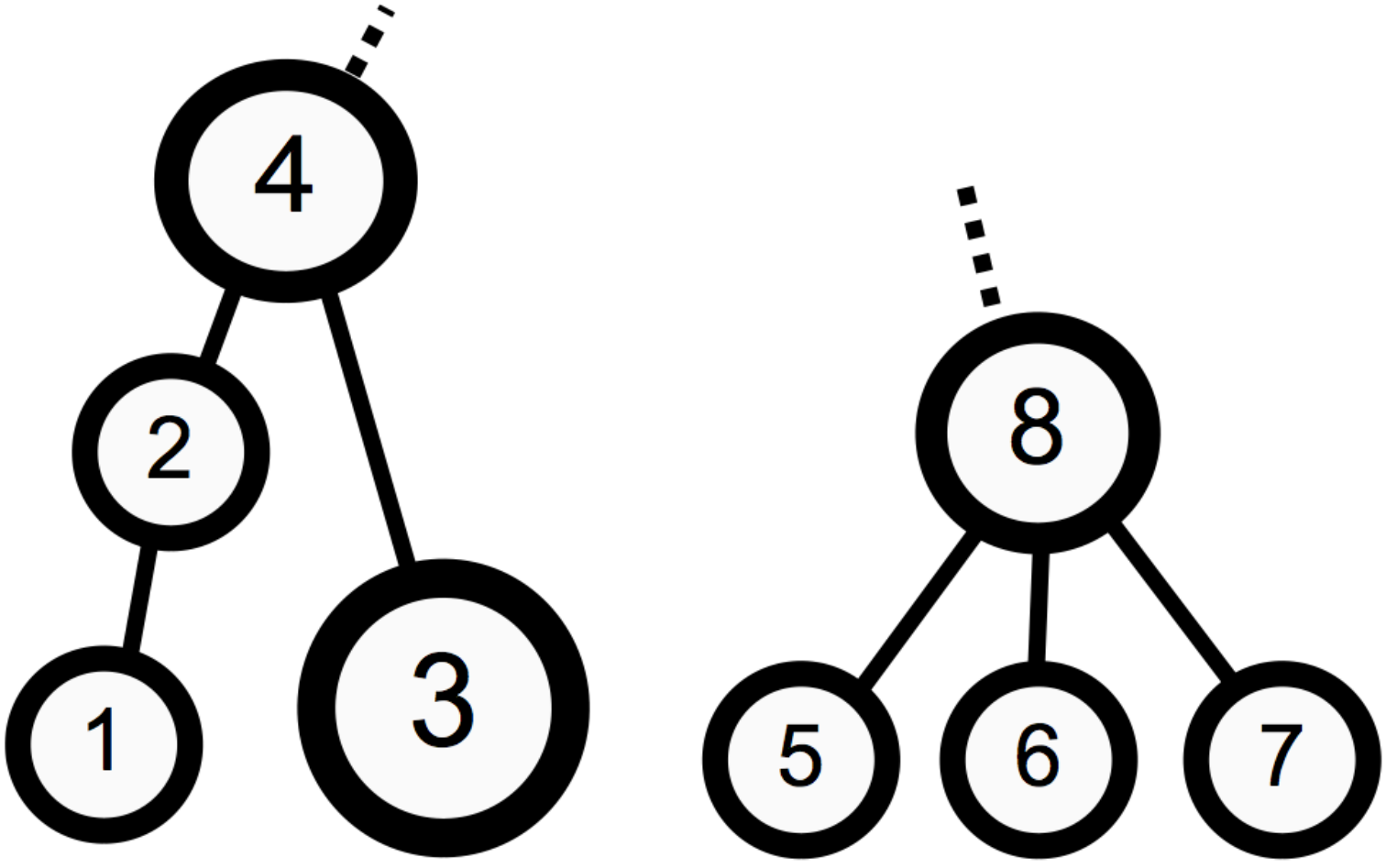
SuiteSparseQR: multifrontal sparse QR factorization

Key results / impact

- rectangular fronts like UMFPACK, but simpler frontal matrix assembly
- multicore parallelism
- amenable to GPU implementation (in progress)
- on multicore CPU: up to 30 Gflops
- sparse `qr` in MATLAB, and `x=A\b`
- on the GPU:
  - novel "Bucket QR" scheduler and custom GPU kernels
  - up to 150 GFlops on the Kepler K20c
  - up to 20x speedup vs CPU algorithm (5x to 10x typical)
  - prototype multi-GPU: another $\sim$ 2x on 2 GPUs

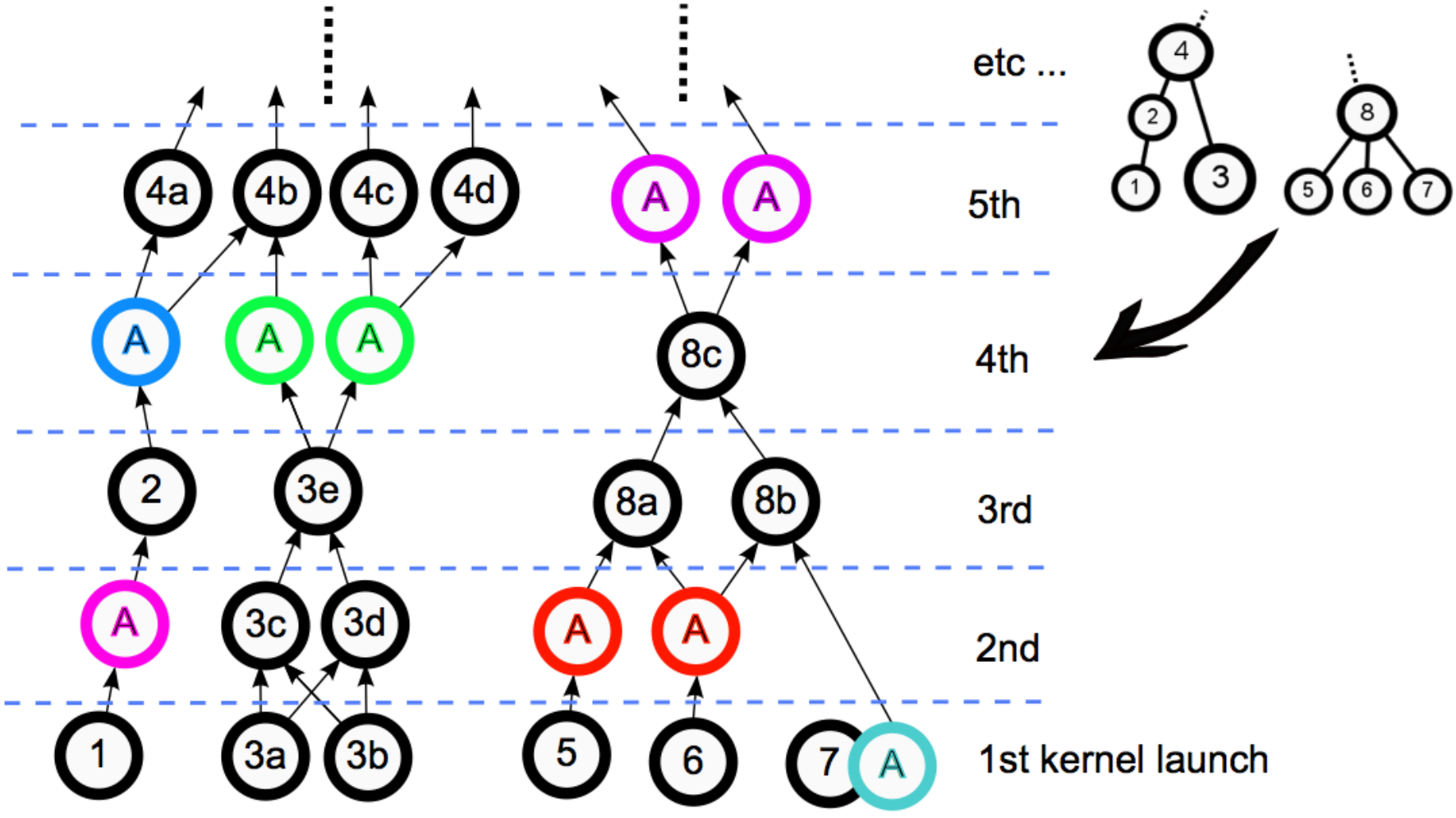
Highly-concurrent heterogeneous parallel computing

Consider a subtree of frontal matrices on the GPU

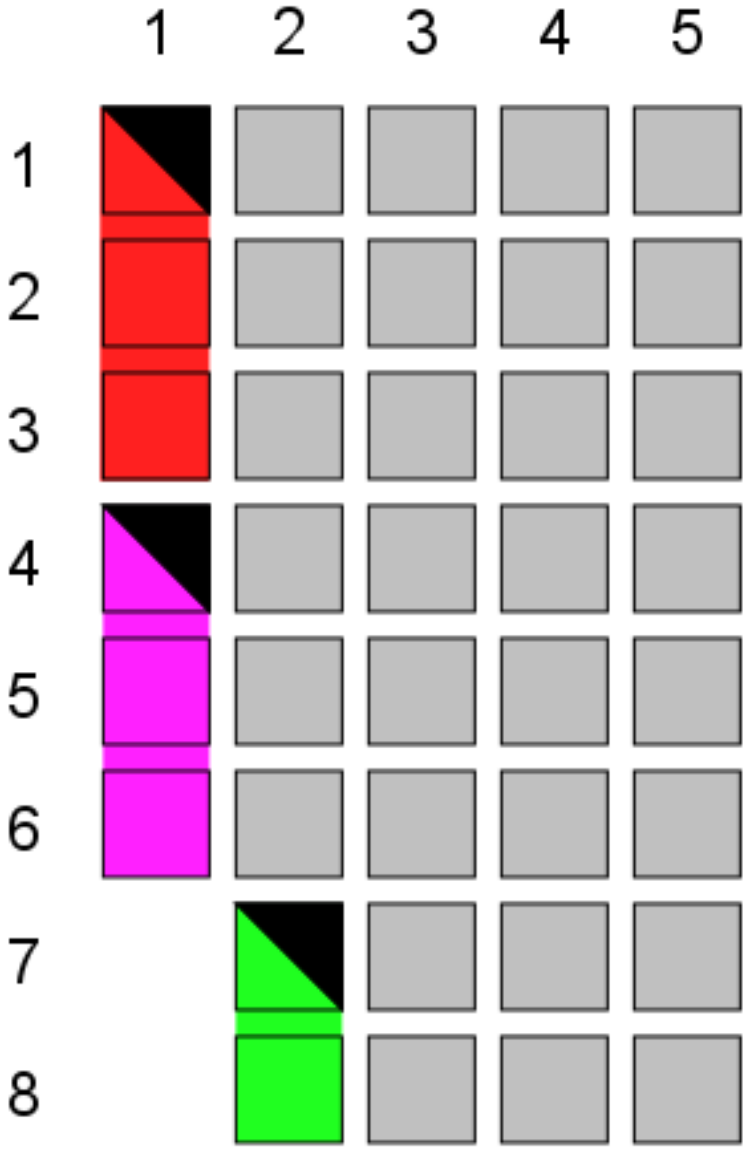


Highly-concurrent heterogeneous parallel computing

Expanded to show GPU kernel launches



GPU-based heterogeneous parallel computing



With no pipelining...

Householder bundles:

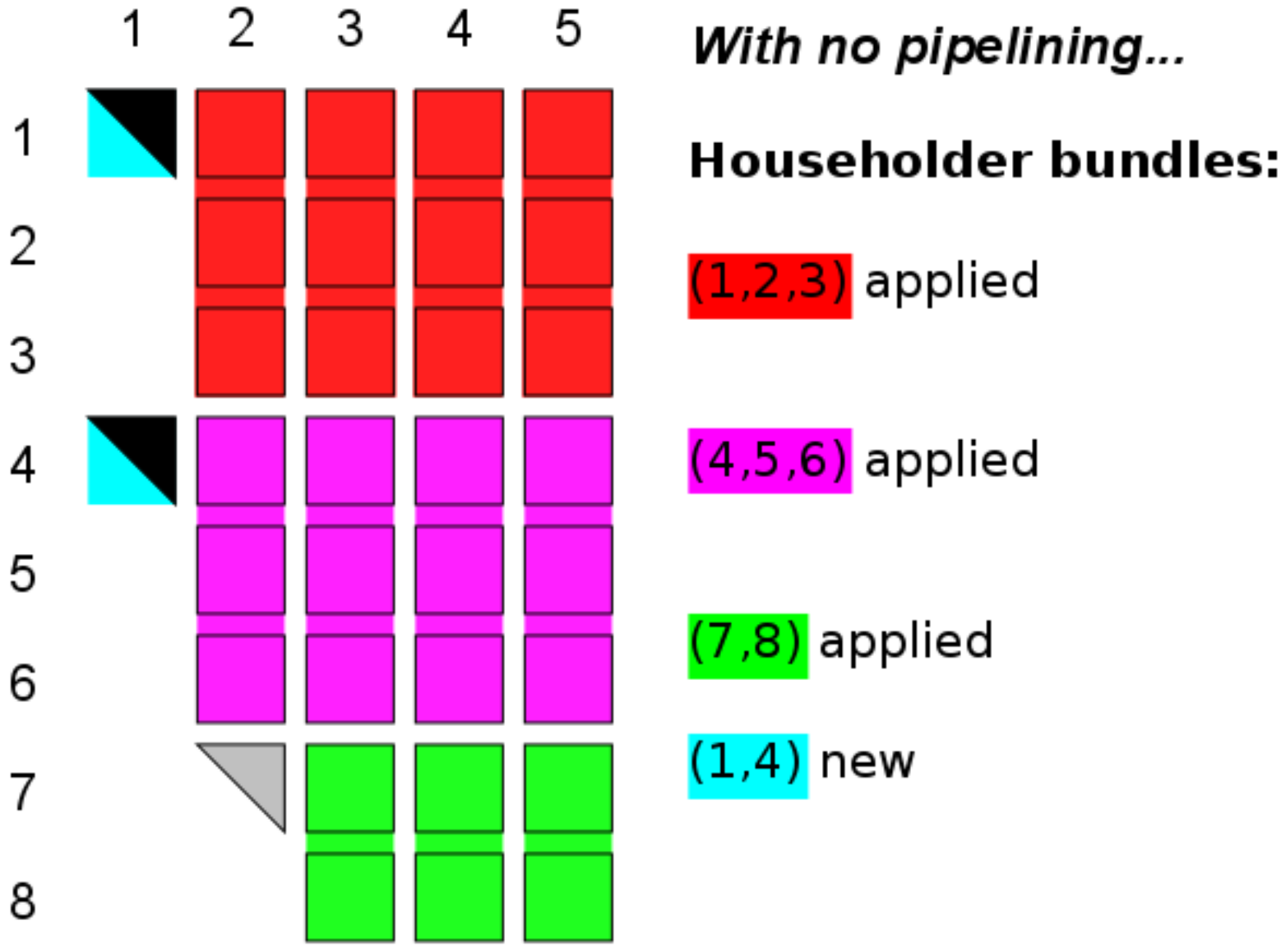
(1,2,3) new

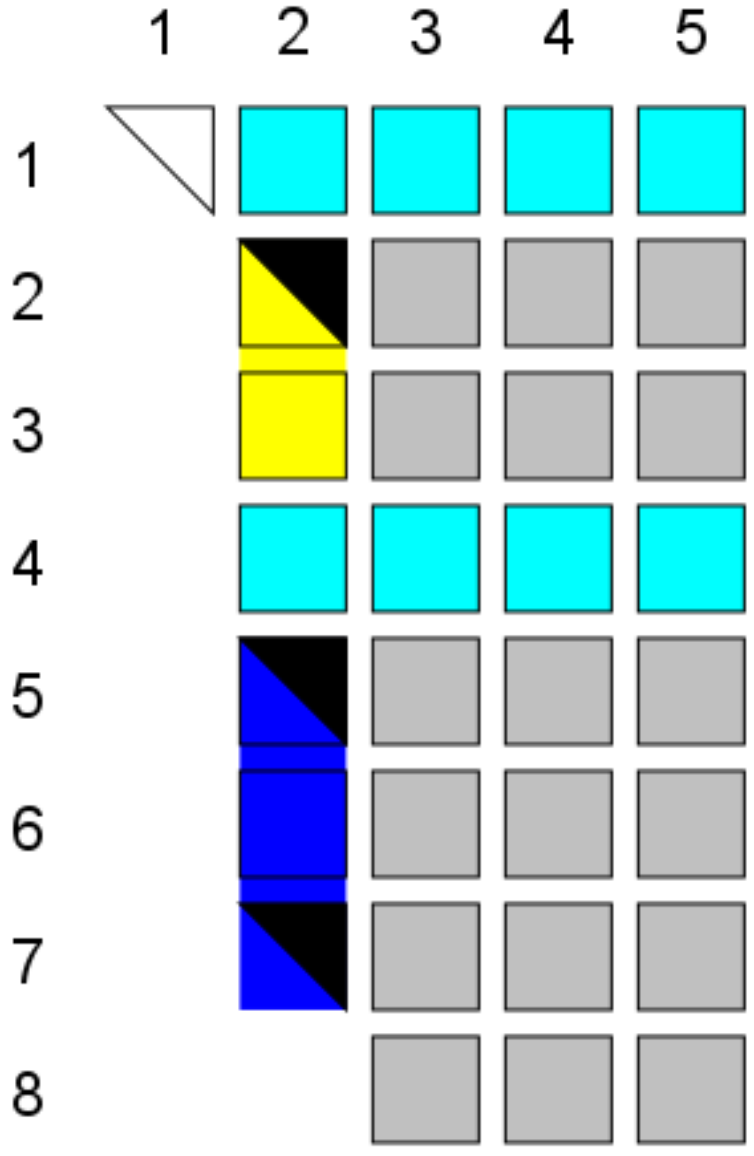
(4,5,6) new

(7,8) new

first kernel launch same as pipelined method

GPU-based heterogeneous parallel computing





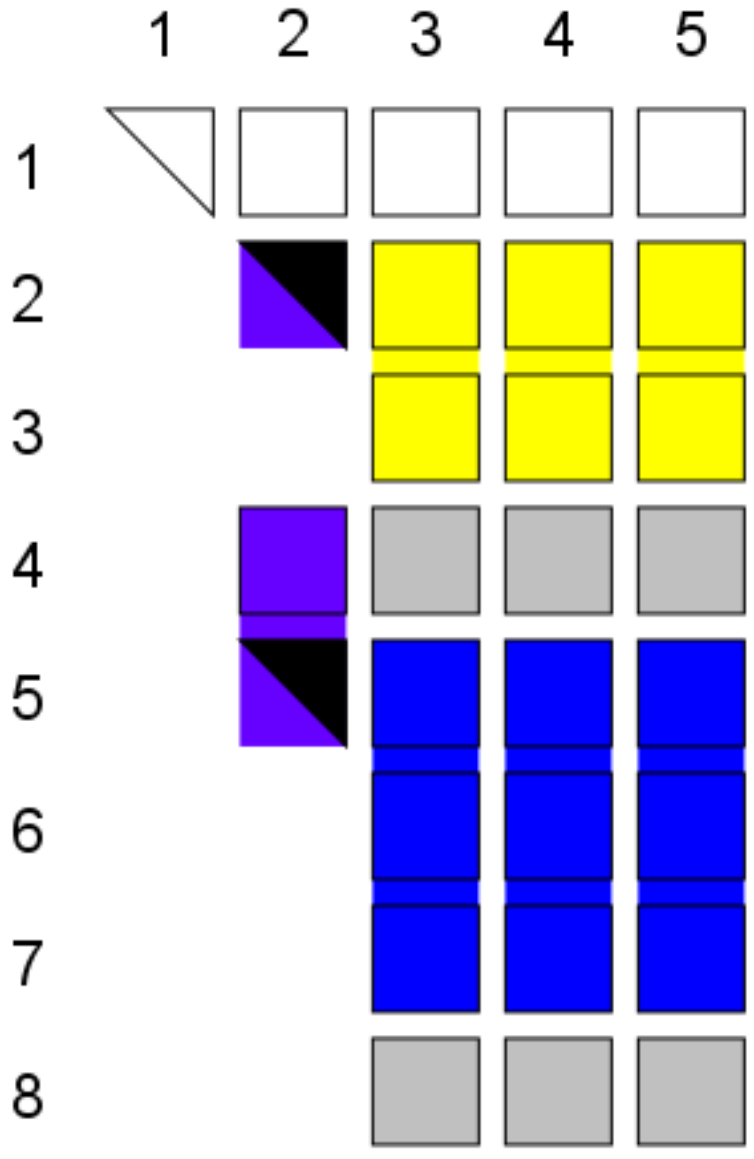
With no pipelining...

Householder bundles:

(2,3) new

(5,6,7) new

(1,4) applied



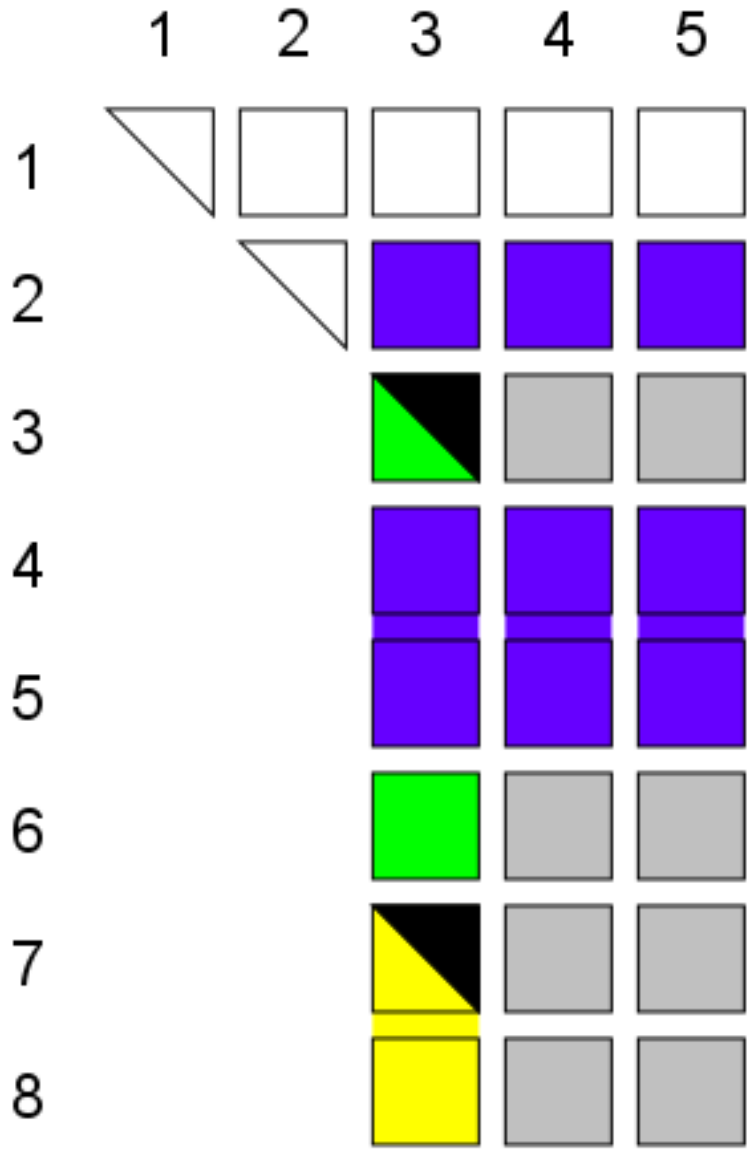
With no pipelining...

Householder bundles:

(2,3) applied

(5,6,7) applied

(2,4,5) new



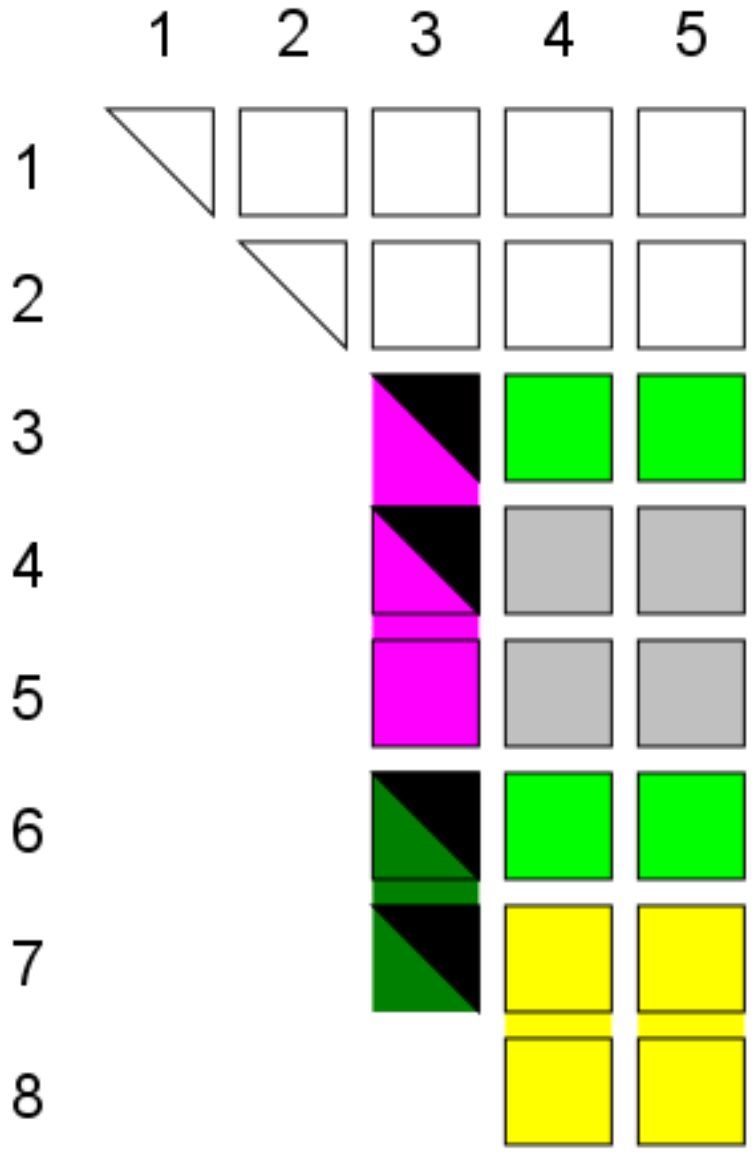
With no pipelining...

Householder bundles:

(3,6) new

(7,8) new

(2,4,5) applied



With no pipelining...

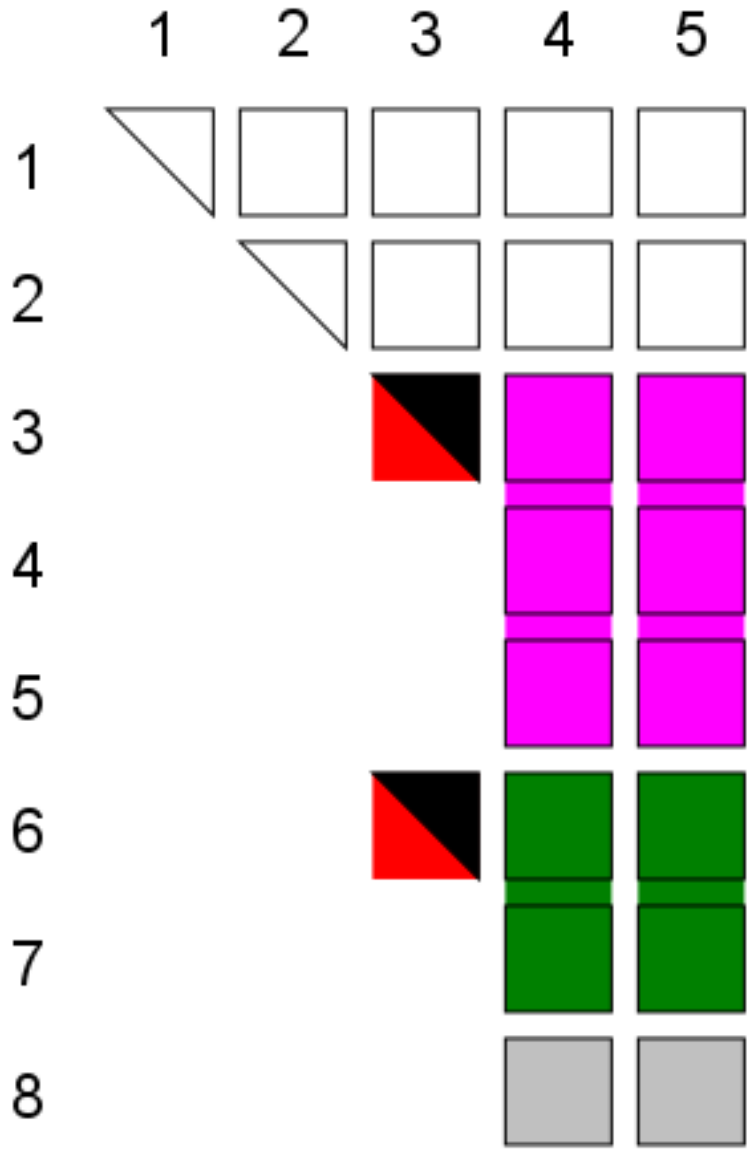
Householder bundles:

(3,6) applied

(7,8) applied

(6,7) new

(3,4,5) new



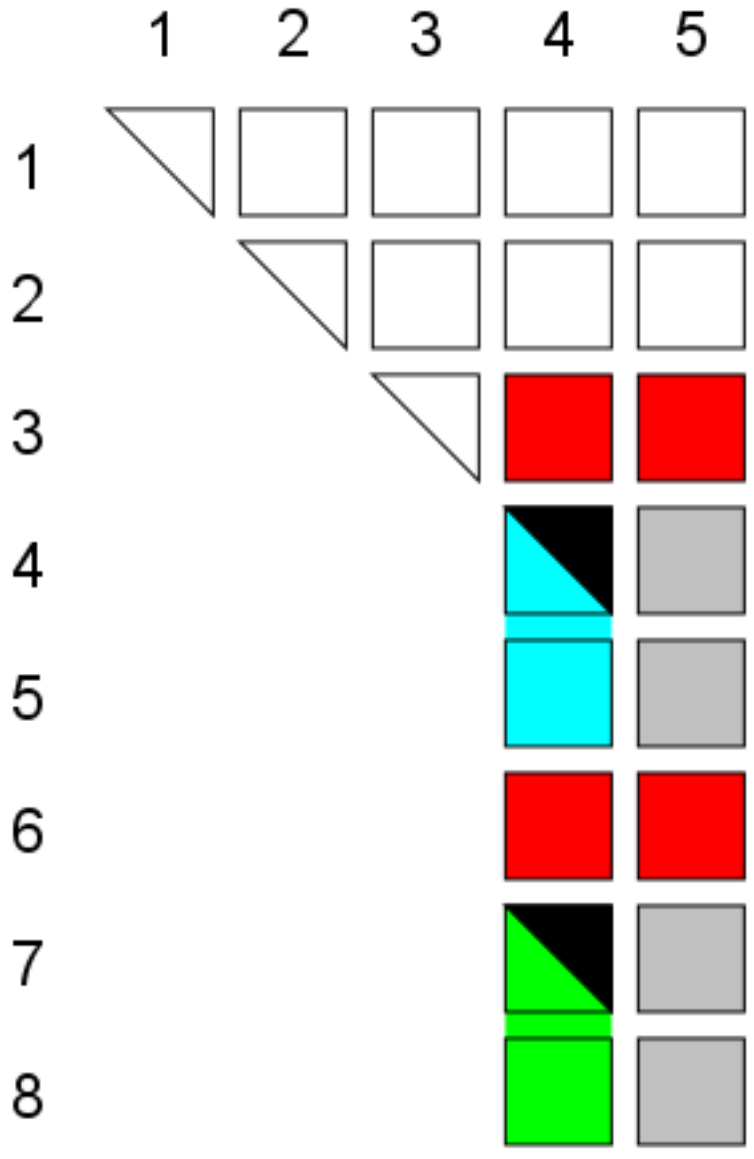
With no pipelining...

Householder bundles:

(3,6) new

(6,7) applied

(4,5) applied



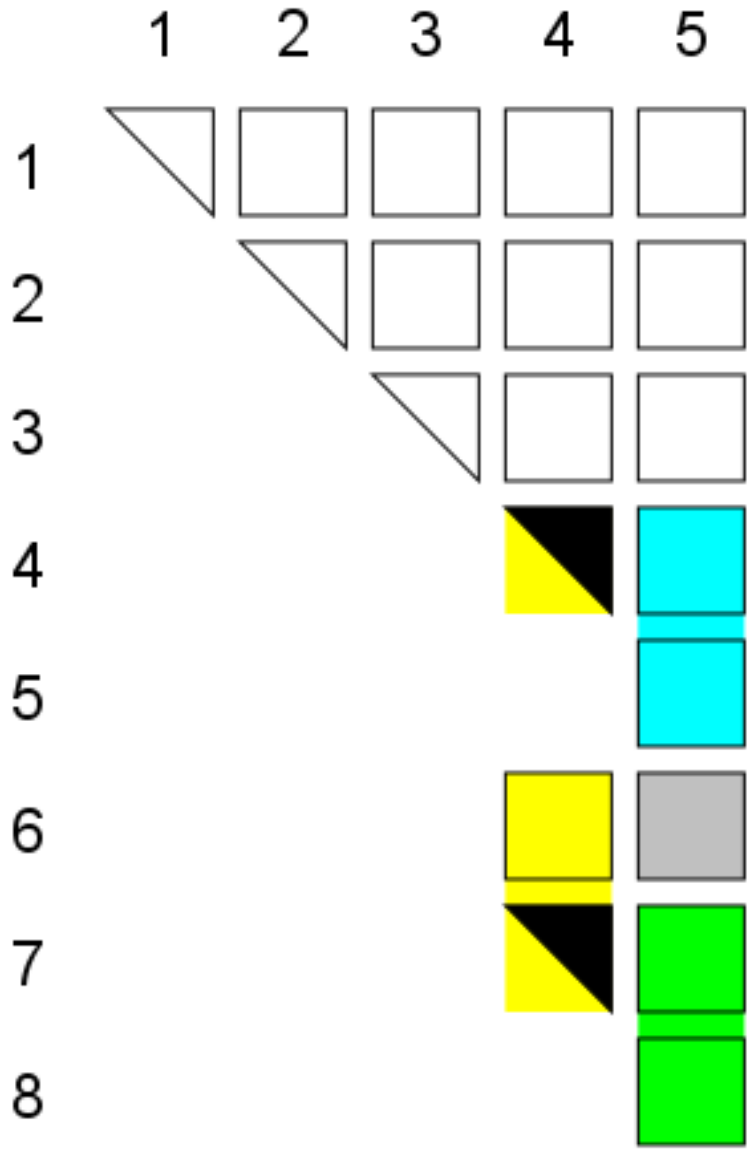
With no pipelining...

Householder bundles:

(3,6) applied

(4,5) new

(7,8) new



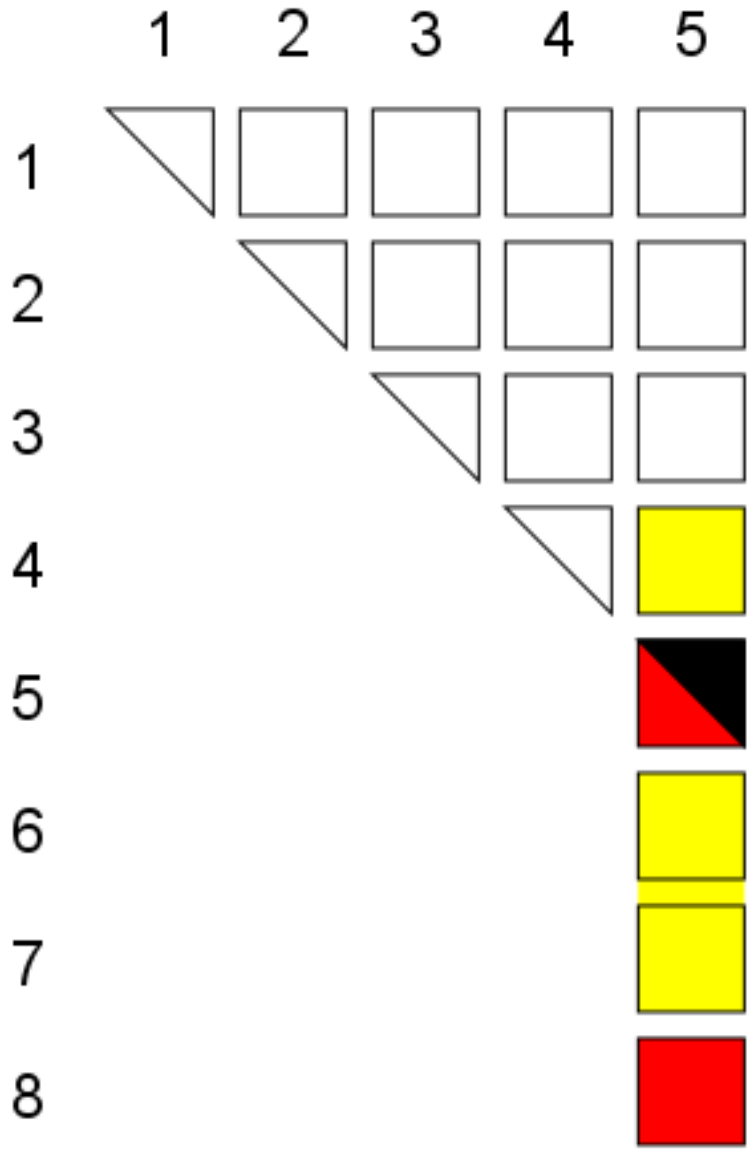
With no pipelining...

Householder bundles:

(4,6,7) new

(4,5) applied

(7,8) applied

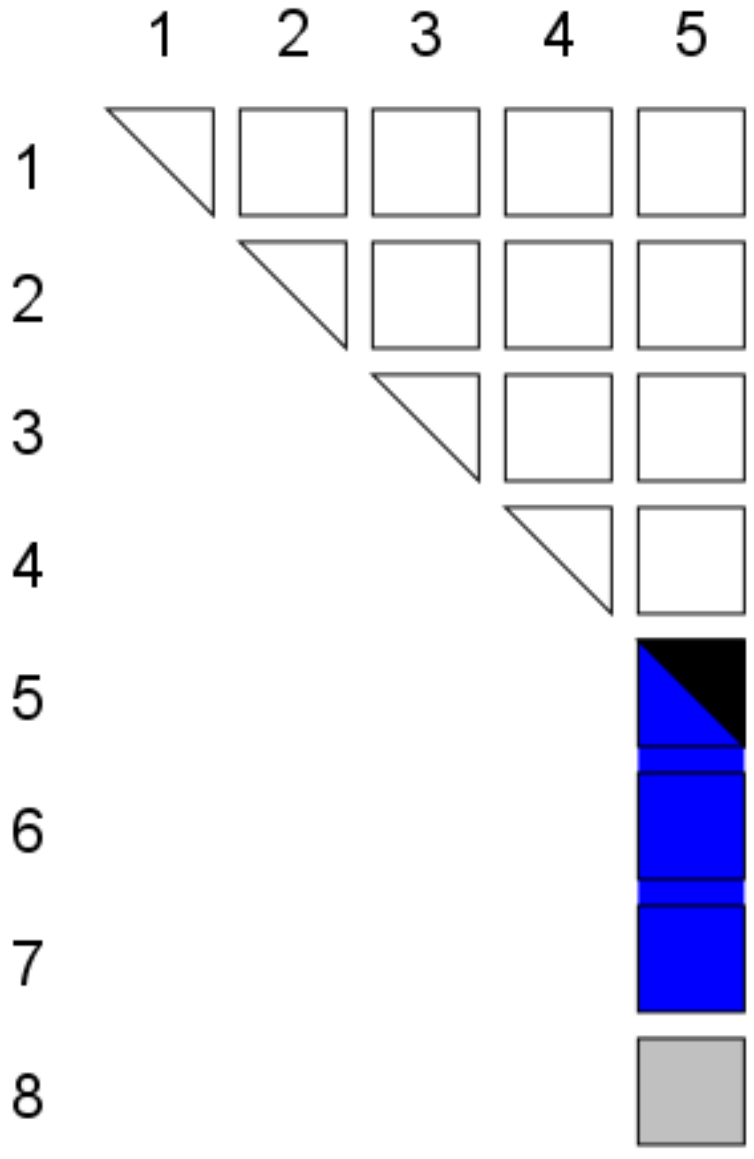


With no pipelining...

Householder bundles:

(4,6,7) applied

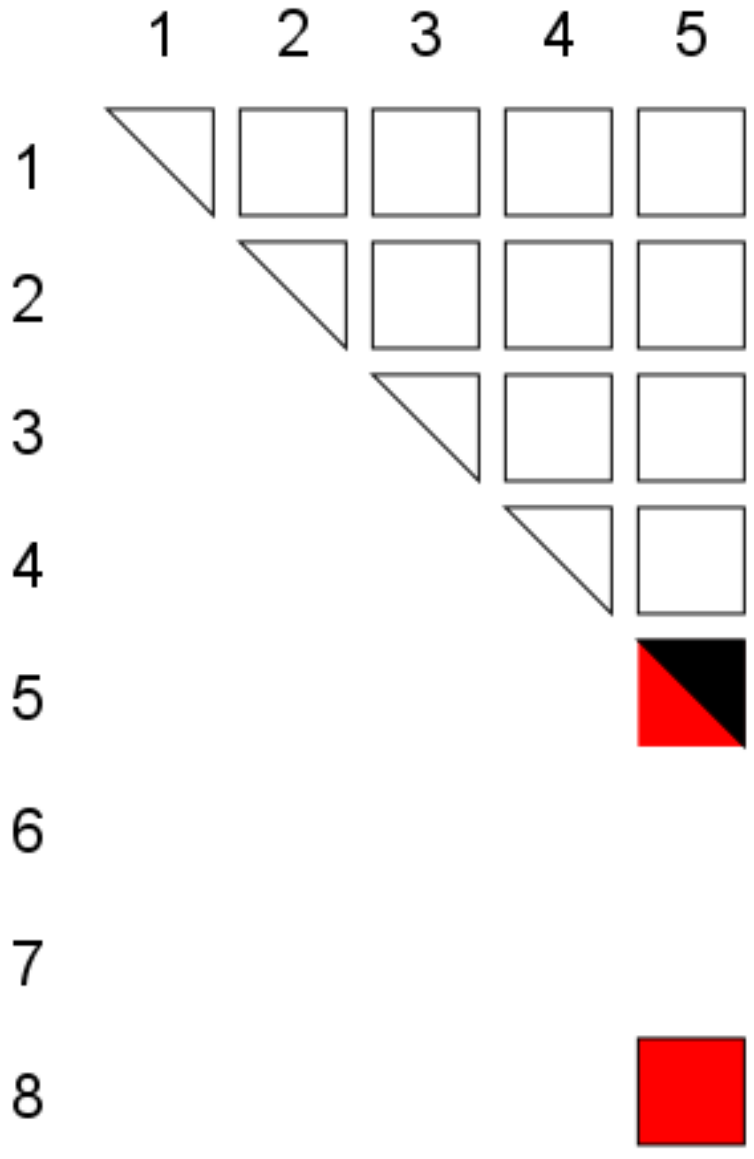
(5,8) new



With no pipelining...

Householder bundles:

(5,6,7) new



With no pipelining...

Householder bundles:

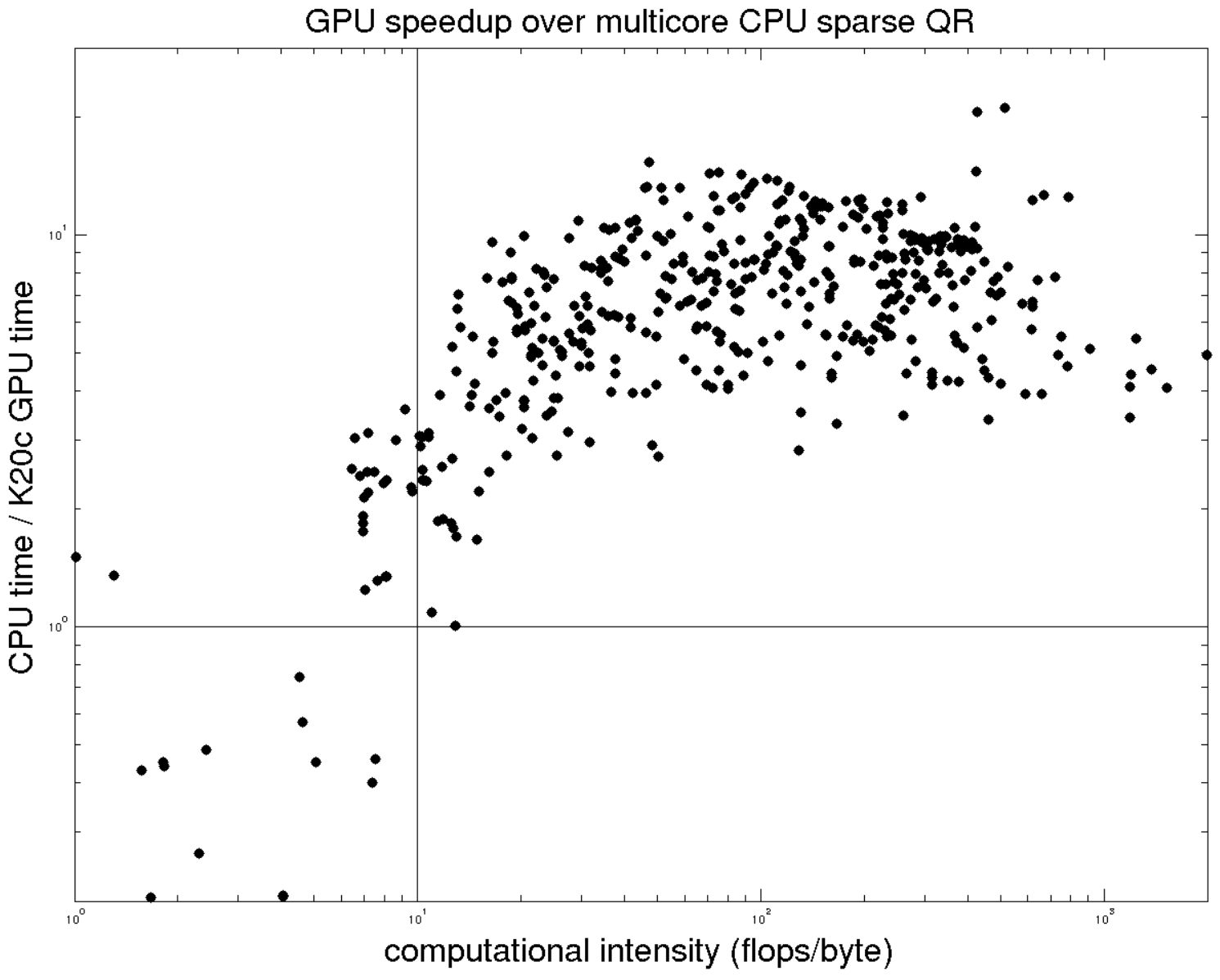
(5,8) new

Highly-concurrent heterogeneous parallel computing

- **Putting it all together ...**

	Fermi	Kepler
GPU kernels:		
apply block Householder	183 Gflops	260 Gflops
factorize 3 tiles	27 Gflops	20 Gflops
dense QR for large front	107 Gflops	120 Gflops
sparse QR on GPU	80 Gflops	150 Gflops
peak speedup over CPU	11x	20x
typical speedup over CPU	5x	10x

Performance on many matrices



Supernodal Sparse Cholesky on the GPU

```
subtrees ← scan elimination tree for A to construct
  appropriately sized subtrees
for all subtrees do
  levels ← arrange subtree supernodes into levels
  copy unfactored A data in subtree to GPU
  for all levels do
    using concurrent, batched kernels:
      initialize buffers and construct all scatter maps
      assemble supernodes (DSYRK)
      assemble lower panels (DGEMM)
      factor supernodes (DPOTRF)
      factor lower panels (DTRSM)
    copy factored supernodes back to CPU
  end for
end for
for all remaining supernodes do
  simultaneously:
    assemble small descendants on CPU
    assemble large descendants on GPU
  block factorization of supernode diagonal block
    on CPU and, depending on size, GPU
  block factorization of lower panel on CPU or
    GPU depending on size
end for
```

Supernodal Sparse Cholesky on the GPU

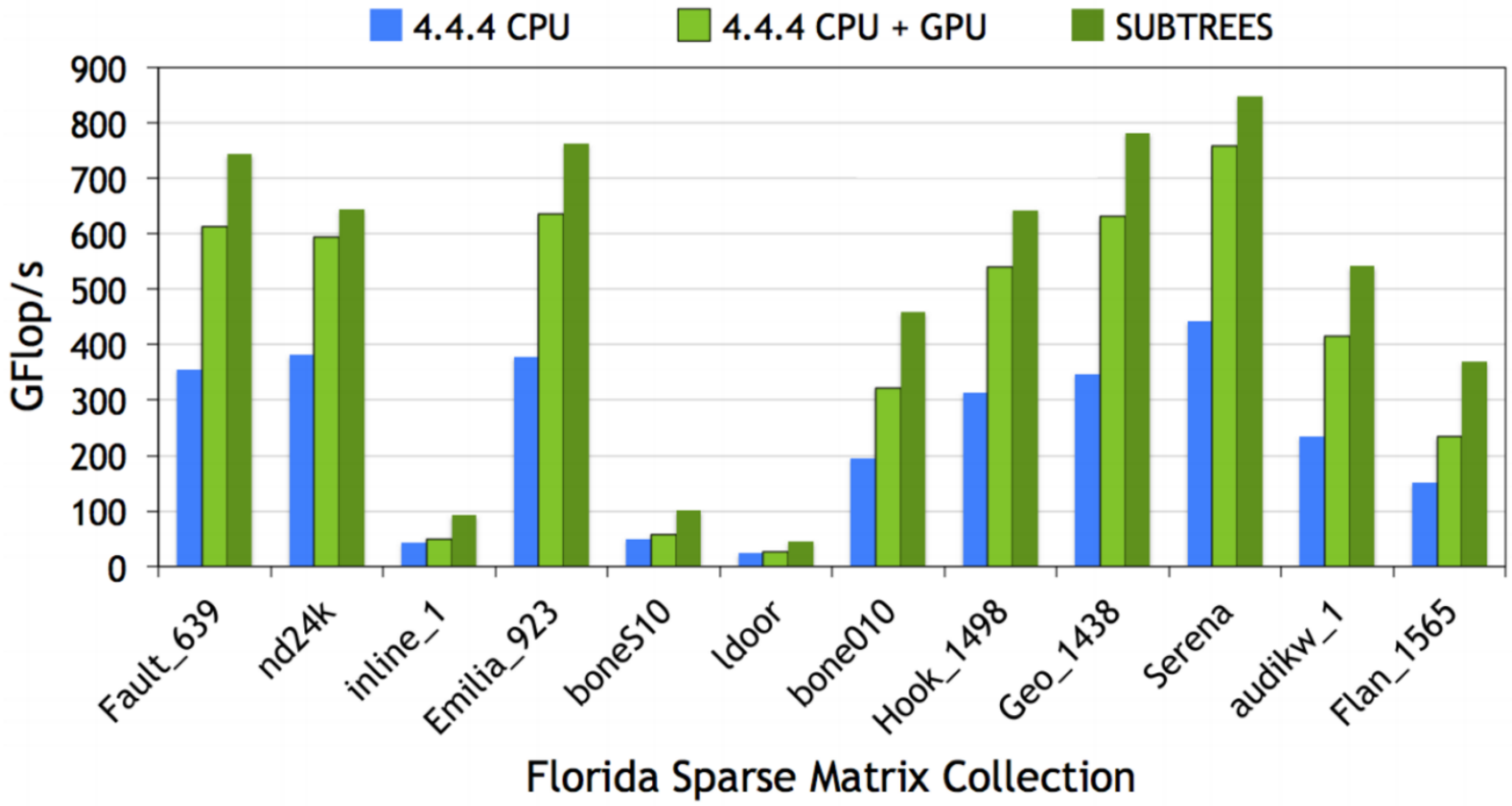
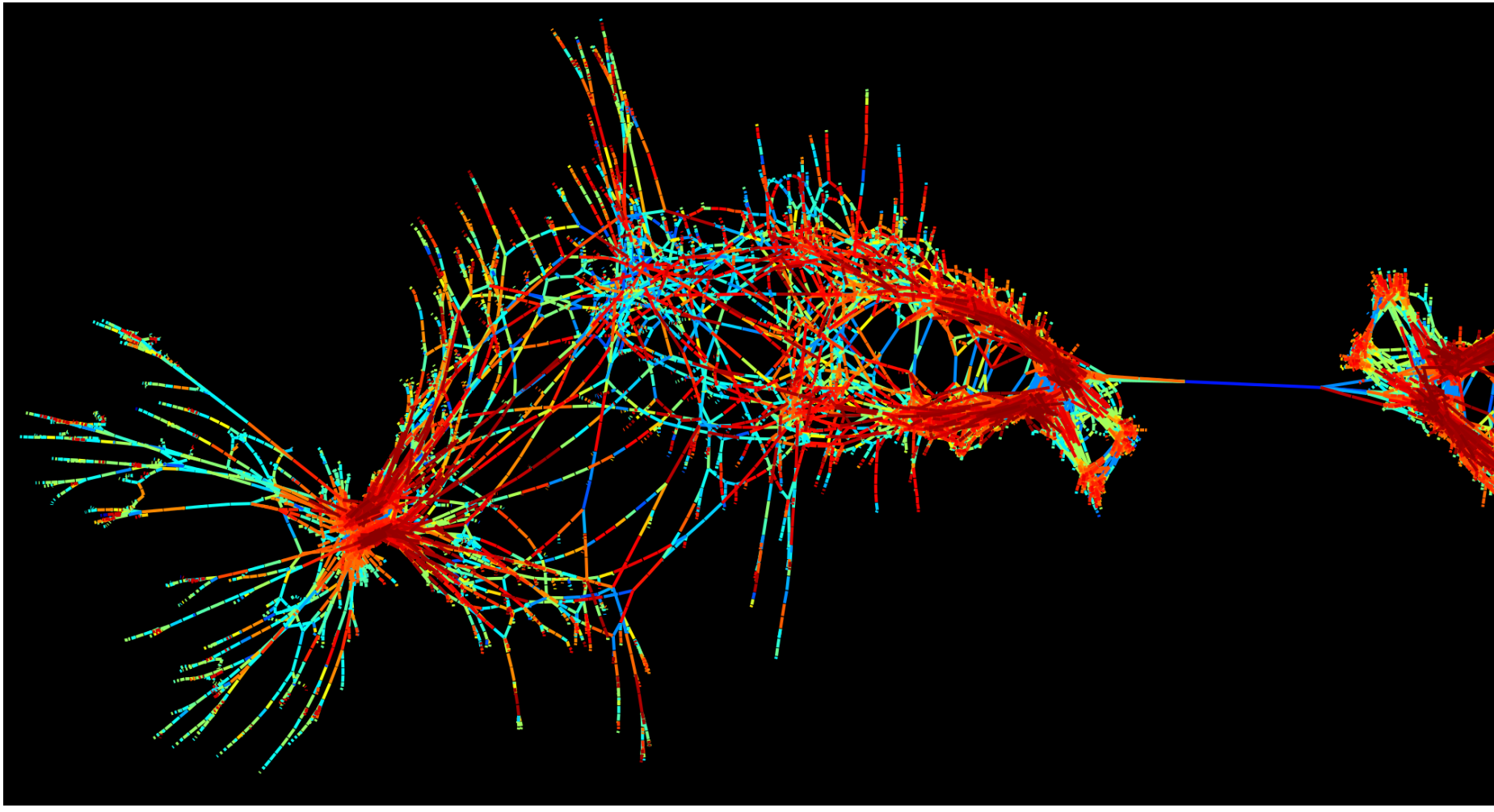


Figure 9: Comparison of Gflops achieved for the sparse factorization between CHOLMOD 4.4.4 CPU-only (blue), CHOLMOD 4.4.4 CPU+GPU (light green) and the current subtree algorithm for the 12 largest real SPD matrices in the Florida Sparse Matrix collection.

Future vision

- **Computer Science + Applied Math**
  (combinatorics + linear algebra + graph algorithms) =
  $$\left[\begin{array}{c}\text{high-performance combinatorial scientific computing} \\ \text{+many applications enabled by my contributions}\end{array}\right]$$
- computational mathematics: the future is heterogeneous; driven by power constraints, need for parallelism
- high impact – getting it out the door
  - novel algorithms: delivered in widely used robust software
  - Collected Algorithms of the ACM
  - enabling academic projects: Julia, R, Octave, FEnICS, ...
  - current collaborations: UTK, UF, NVIDIA, Lawrence Livermore, Harvard, ...
  - growing industrial impact: MathWorks, Google, NVIDIA, Mentor Graphics, Cadence, MSC Software, Berkeley Design Automation, ...
  - applications: optimization, robotics, circuit simulation, computer graphics, computer vision, finite-element methods, geophysics, stellar evolution, financial simulation, ...

Computer Science + Math + Music = Art



- algorithmic translation of music into visual art
- theme artwork, London Electronic Music Festival