Sparse Matrix Algorithms

combinatorics + numerical methods + applications
Math + X

Tim Davis
University of Florida

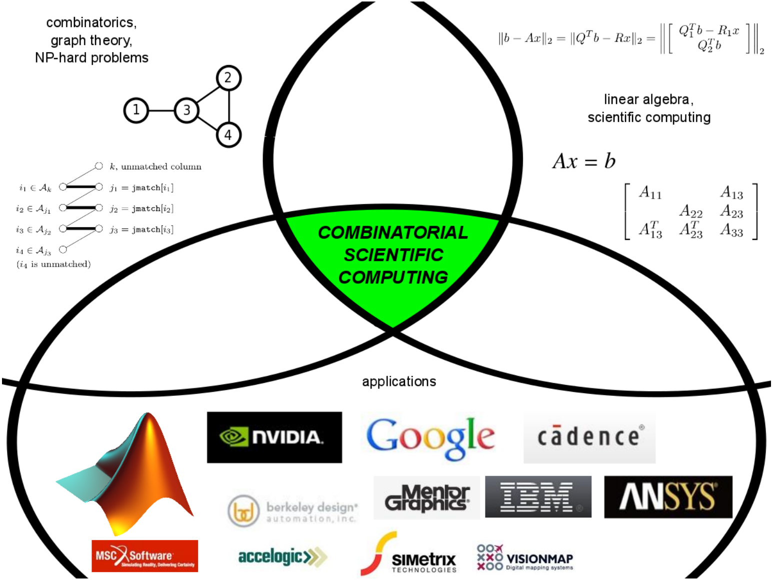
June 2013

contributions to the field
current work
vision for the future

Outline

- **Math+X**
  - $\text{Math} = \left[\ \text{combinatorics} + \text{linear algebra} + \text{graph theory}\ \right]$
  - $X = \left[\begin{array}{c} \text{high-performance combinatorial scientific computing} \\ +\text{many applications enabled by my contributions} \end{array}\right]$
- **Roadmap: past, current, future work**
- **Sparse matrix algorithms**
- **Contributions to the field**
  - from theory, to algorithms, to reliable software, to applications
  - sparse Cholesky update/downdate (CHOLMOD)
  - unsymmetric multifrontal LU (UMFPACK)
  - multifrontal QR (SuiteSparseQR)
- **Current work**
  - highly concurrent methods (GPU or massive CPU core)
  - NVIDIA Academic Partner
- **Future vision**

Math+X $\quad X$ = high-performance combinatorial scientific computing + applications



Roadmap: past, current, future work

completed	simplicial methods	< 1 Gflop: sparse LU for circuit simulation, for extremely sparse problems.
completed	single core, BLAS-based supernodal and multifrontal	~3 Gflop: multifrontal or supernodal LU, QR, and Cholesky (all in x=A\b in MATLAB)
completed	multi-core, BLAS based	~5 to 15 Gflop: multifrontal/supernodal LU, QR, Cholesky x=A\b, large problems, 2D / 3D discretizations
now	single-GPU (100s of cores)	~80 Gflop: communication-avoiding multifrontal sparse QR. Future: need GPU kernels for LU and Cholesky.
near	multi-GPU (1000s of cores) single shared memory cluster	~1 Tflop goal: near horizon: multi-GPU sparse QR ~3 Tflop future: use all CPUs & GPUs in small cluster
future	multi-GPU (10000s of cores) dist. memory cluster	~10 Tflop to ~1 Petaflop goal: farther in future. Need distributed-memory algorithms across clusters, each exploiting multiple CPUs and GPUs.

towards exascale performance in sparse direct solvers

Applications using SuiteSparse

General toolboxes for computational science

Applications using SuiteSparse

Linux distros

Ubuntu

archlinux

fedora

FINK

the GNU-Darwin distribution

debian

openSUSE

Scientific Linux

Finite-element methods and differential equations

ANSYS

MSC Software Simulating Reality, Delivering Certainty

deal.II

Elmer

feat

FEniCS project

NIST

Orcina

SfePy

Applications using SuiteSparse

Circuit / power / semiconductor simulation

ATopTech
berkeley design automation, inc.
cadence
EnerNex
freescale
HRL LABORATORIES
IBM
lumerical illuminating the way
Mentor Graphics
NGSPICE
Sandia National Laboratories
SIMetrix TECHNOLOGIES
Xyce

Computer graphics / computer vision / robotics

Google
CGAL Computational Geometry Algorithms Library
GEGL
ROS.org
TANDENT
USGS science for a changing world Astrogeology Research Program
VISIONMAP Digital mapping systems
OpenSLAM Give your algorithm to the community

Applications using SuiteSparse

Mathematical optimization

CVXOPT
Python Software for Convex Optimization

cvxpy
A Python package for modeling convex optimization problems

CVX RESEARCH

SOL

mosek
http://www.mosek.com

Geophysical modeling

msi kenny

GEOMODELLING SOLUTIONS

GEOGRAF

OpenSees

Financial simulation

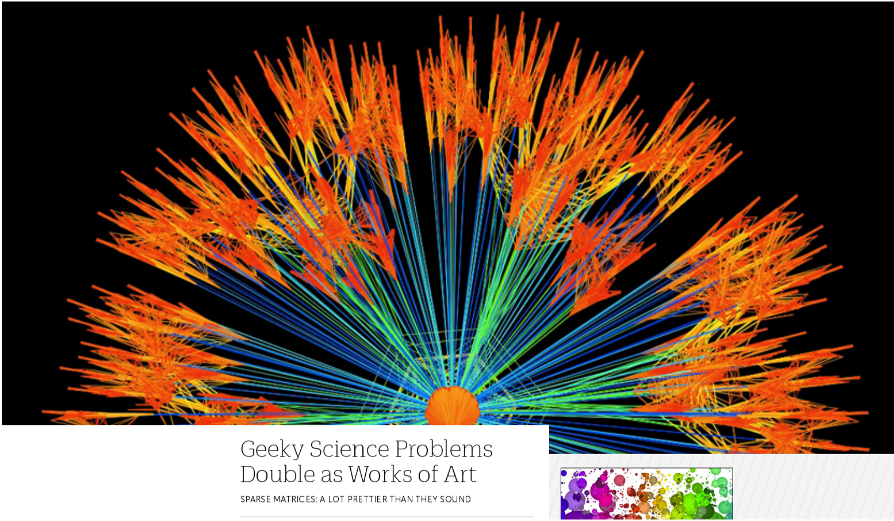
INTEX

Stellar evolution

MESA

Applications using SuiteSparse

- **University of Florida Sparse Matrix Collection (with Hu)**
  - 2500+ sparse matrices from real applications



Sparse matrix algorithms

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

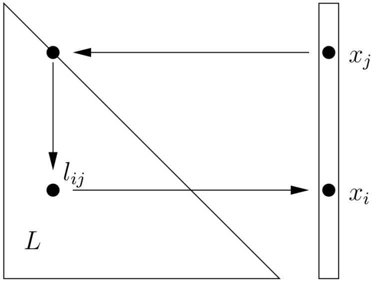
$x = b$
for $j = 0$ to $n - 1$ **do**
  if $x_j \neq 0$
    for each $i > j$ for which $l_{ij} \neq 0$ **do**
      $x_i = x_i - l_{ij} x_j$

- 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

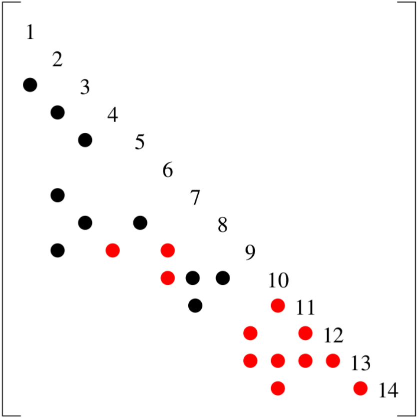
$x = b$
for $j = 0$ to $n-1$ **do**
  if $x_j \neq 0$
    for each $i > j$ for which $l_{ij} \neq 0$ **do**
      $x_i = x_i - l_{ij} x_j$

- 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 $l_{ij} \neq 0$
- $\mathcal{X} = \text{Reach}_{\mathcal{L}}(\mathcal{B})$
  (Gilbert/Peierls)

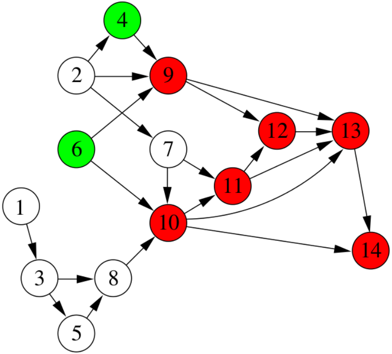


Sparse matrix algorithms

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, ...

Sparse Cholesky update/downdate

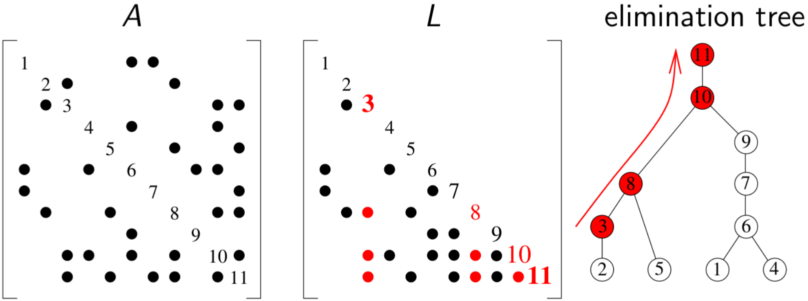
Rank-1 update, $LL^T + ww^T$ (Carlson)

$\overline{\beta} = 1$
for $j = 1$ to n
compute coefficients:
$\alpha = w_j / l_{jj}$
$\beta = \overline{\beta}, \quad \overline{\beta} = \sqrt{\beta^2 + \alpha^2}$
$\gamma = \alpha / (\overline{\beta}\beta), \qquad \delta = \beta / \overline{\beta}$
update diagonal:
$l_{jj} = \delta l_{jj} + \gamma w_j$
$w_j = \alpha$
update below diagonal:
$\mathbf{t} = \mathbf{w}_{j+1:n}$
$\mathbf{w}_{j+1:n} = \mathbf{w}_{j+1:n} - \alpha \mathbf{L}_{j+1:n,j}$
$\overline{\mathbf{L}}_{j+1:n,j} = \delta \mathbf{L}_{j+1:n,j} + \gamma \mathbf{t}$
end

Key observations:

- overwrites w with solution to $L\overline{w} = w$
- `w=L\w` in MATLAB notation
- jth column of L changes only if $\overline{w}_j = (L^{-1}w)_j \neq 0$,
- thus, need pattern of triangular solve
- ... but the Cholesky L can be pruned

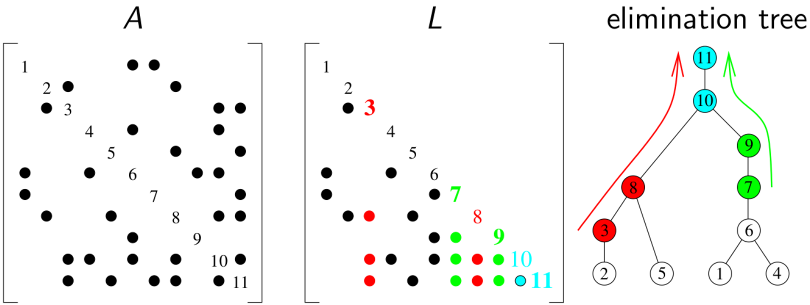
Sparse Cholesky update/downdate



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



Multiple rank, with dynamic supernodes

- multiple rank: $A \pm WW^T$, follow multiple paths in the tree
- supernodes: adjacent columns of L with identical pattern; break apart and merge in update/downdate
- dynamic supernodes: find them as we go (cuts memory traffic)

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, Yeralan)
  - structural engineering problem: crack in aircraft fuselage
  - update/downdate as crack progresses through airframe

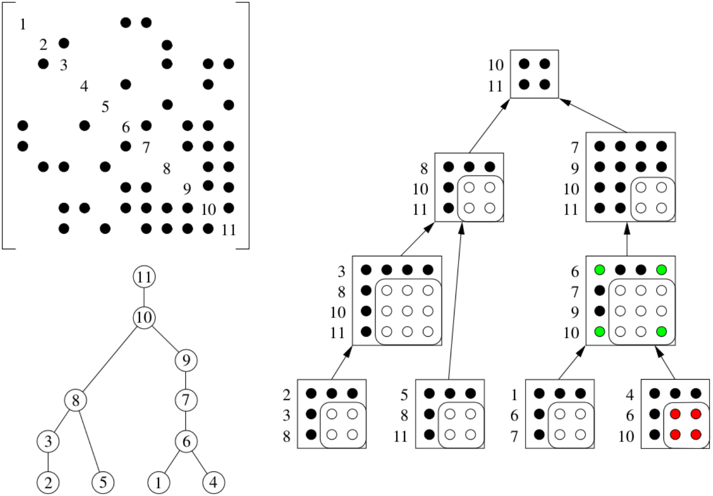
CHOLMOD: Supernodal Sparse Cholesky update/downdate

CHOLMOD: key results / impact

- sparse `chol` in MATLAB, and `x=A\b`
- typical 10x speedup when incorporated into MATLAB
- peak performance over 50% of CPU theoretical peak
- example applications using CHOLMOD:
  - Google Street View, PhotoTours, and 3D Earth
  - Mentor Graphics: circuit simulation
  - Cadence: CAD design
  - VisionMap: satellite image processing
  - CVXOPT: convex optimization
  - ...

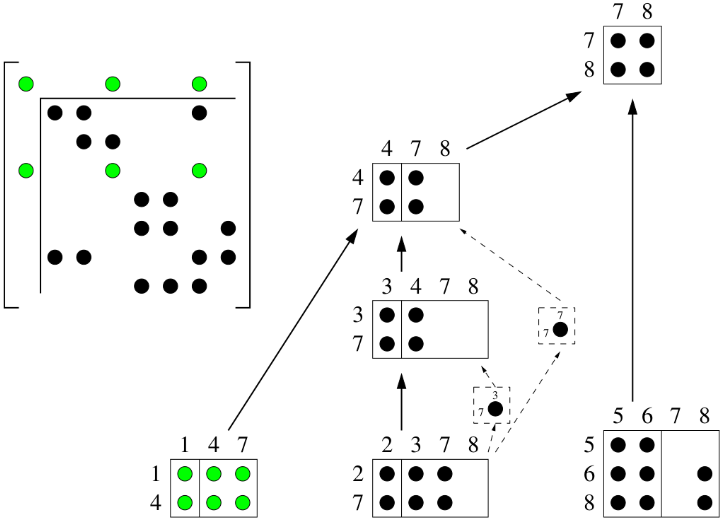
Multifrontal method

- 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



UMFPACK: unsymmetric multifrontal method

Key results / impact

- high-performance via dense matrix kernels within each front
- symbolic preordering and analysis, followed by revised local pivot search with approximate unsymmetric degree update
- could extend to handle threshold rook pivoting
- widely used
  - sparse LU and `x=A\b` in MATLAB
  - Mathematica
  - IBM circuit simulation
  - finite-element solvers: NASTRAN, ANSYS, FEniCS, ...
  - CVXOPT
  - ...

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 14 Gflops
- sparse `qr` in MATLAB, and `x=A\b`

SuiteSparseQR

- Least squares problem: 2 million by 110 thousand

Method	ordering	procs	time
prior `x=A\b`	COLMMD	1	?
prior `x=A\b`	AMD	1	11 days
MA49	AMD	1	3.5 hours
SuiteSparseQR	AMD	1	1.5 hours
SuiteSparseQR	METIS	1	45 minutes
SuiteSparseQR	METIS	16	7.3 minutes

- Algorithmic speedup vs prior `x=A\b`: 375x
- Parallel speedup: 5.75x on 16 cores
- Total: 2,155x (14 Gflops on machine w/ 70 Gflops peak)
- Single core: 2.5 Gflop peak, same as LAPACK dense QR

Highly-concurrent heterogeneous computing

GPU computing: bulk synchronous model

- multiple Streaming Multiprocessors (SMs) on a GPU
- each SM with multiple SIMD threads
- one kernel launch: set of independent tasks

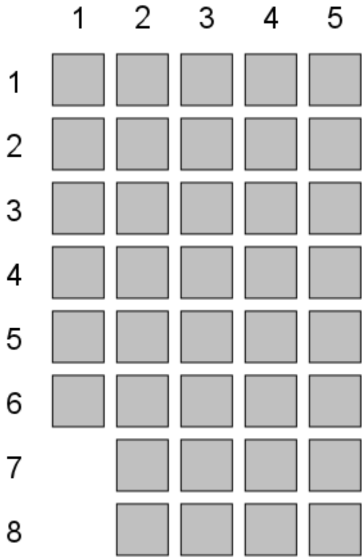
GPU-based sparse multifrontal QR

- symbolic ordering and analysis on the CPU (irregular)
- numerical factorization on the GPU (regular+irregular)

Multifrontal factorization and assembly

- Prior methods
  - one front at a time on the GPU
  - assembly on CPU
  - panel factorization on the CPU, applied on GPU
- Our multifrontal QR
  - many fronts on the GPU (entire subtree)
  - assembly on GPU
  - entire dense QR of each front on the GPU

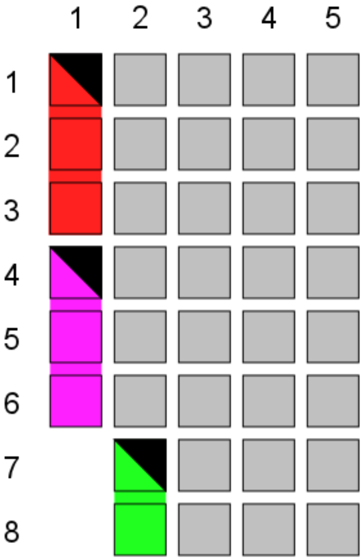
Highly-concurrent heterogeneous computing



Initial matrix:

- each tile a square submatrix
- panelsize: 3-by-1 tiles
- exploits initial staircase

Highly-concurrent heterogeneous computing



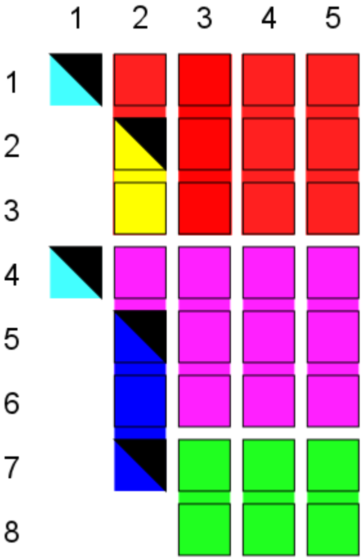
Householder bundles:

(1,2,3) new

(4,5,6) new

(7,8) new

Highly-concurrent heterogeneous computing



Householder bundles:

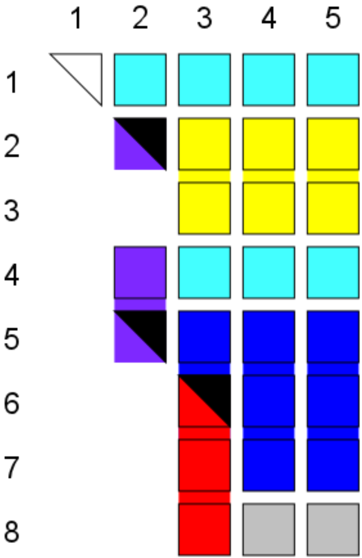
(1,2,3) applied, and
pipelined to (2,3)

(4,5,6) applied, and
pipelined to (5,6,7)

(7,8) applied

(1,4) new

Highly-concurrent heterogeneous computing

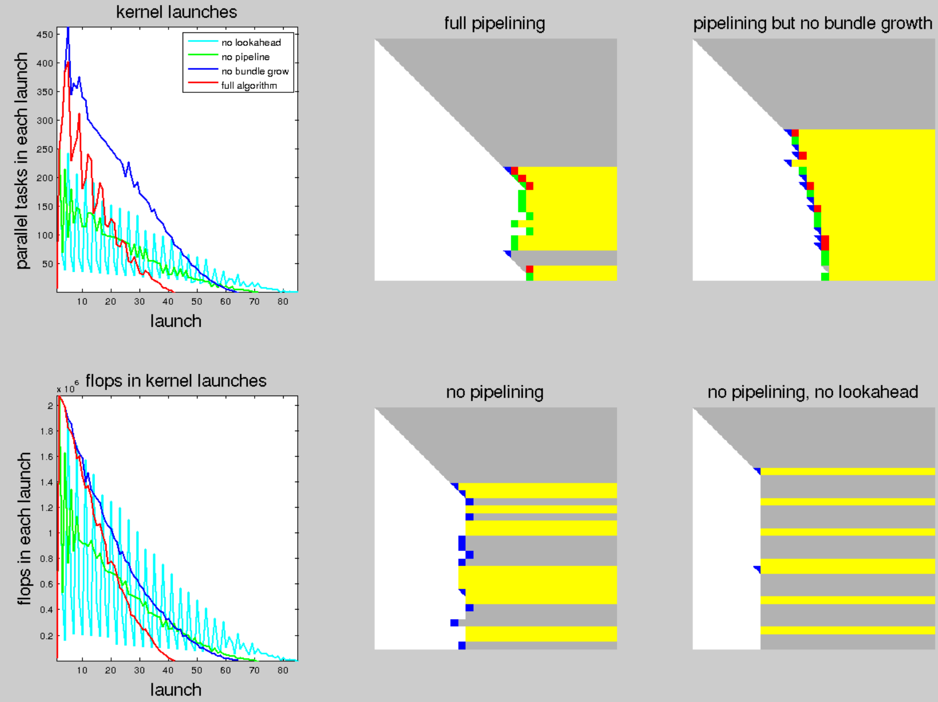


Householder bundles:

(2,3) applied

(5,6,7) applied, and
pipelined to (6,7,8)

(1,4) applied, and
pipelined to (2,4,5)

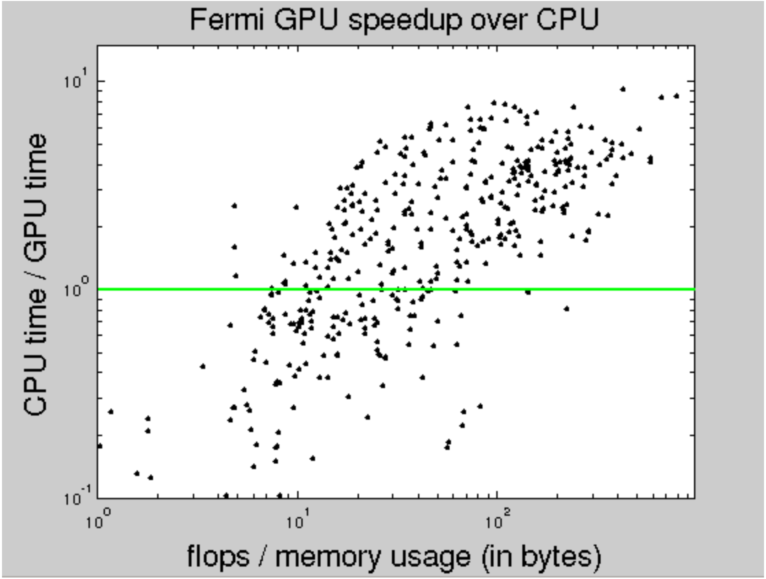

kernel launches
parallel tasks in each launch
no lookahead
no pipeline
no bundle grow
full algorithm
launch
full pipelining
pipelining but no bundle growth
flops in kernel launches
x 10^6
flops in each launch
launch
no pipelining
no pipelining, no lookahead

Highly-concurrent heterogeneous computing

- **Peak performance results on the GPU**

	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	(in progress)
speedup over CPU	9x	(in progress)

Highly-concurrent heterogeneous computing

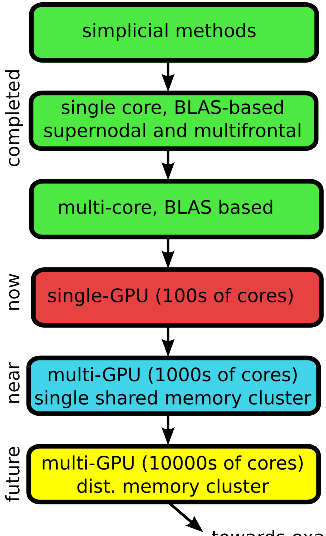


Future vision

Math + X

- Math $= \left[\text{ combinatorics + linear algebra + graph theory } \right]$
- $X = \left[\begin{array}{c} \text{high-performance combinatorial scientific computing} \\ \text{+applications enabled by my contributions} \end{array} \right]$
- 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, ...
  - growing impact on industry, and industrial collaborations: 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, ...

Roadmap: past, current, future work



< 1 Gflop: sparse LU for circuit simulation, for extremely sparse problems.

~3 Gflop: multifrontal or supernodal LU, QR, and Cholesky (all in x=A\b in MATLAB)

~5 to 15 Gflop: multifrontal/supernodal LU, QR, Cholesky x=A\b, large problems, 2D / 3D discretizations

~80 Gflop:
communication-avoiding multifrontal sparse QR.
Future: need GPU kernels for LU and Cholesky.

~1 Tflop goal: near horizon: multi-GPU sparse QR
~3 Tflop future: use all CPUs & GPUs in small cluster

~10 Tflop to ~1 Petaflop goal: farther in future.
Need distributed-memory algorithms across clusters, each exploiting multiple CPUs and GPUs.

towards exascale performance in sparse direct solvers

In Summary

- **Math+X**
  - Math $= \left[\ \text{combinatorics} + \text{linear algebra} + \text{graph theory}\ \right]$
  - $X = \left[\begin{array}{c}\text{high-performance combinatorial scientific computing} \\ +\text{many applications enabled by my contributions}\end{array}\right]$
- **Roadmap: past, current, future work**
- **Sparse matrix algorithms**
- **Contributions to the field**
  - from theory, to algorithms, to reliable software, to applications
  - sparse Cholesky update/downdate (CHOLMOD)
  - unsymmetric multifrontal LU (UMFPACK)
  - multifrontal QR (SuiteSparseQR)
- **Current work**
  - highly concurrent methods (GPU or massive CPU core)
  - NVIDIA Academic Partner
- **Future vision** : individual and collaborative
  - GPU / CPU heterogeneous parallel computing
  - towards exascale
  - continue creating algorithms with deep impact

Math + X, where X = [Poetry + Art]

Sea Fever, by Masefield (1902)

I must go down to the seas again, to the lonely sea and the sky,
And all I ask is a tall ship and a star to steer her by,
And the wheel's kick and the wind's song
    and the white sail's shaking,
And a grey mist on the sea's face and a grey dawn breaking.

C Fever, by T.D. (2010)

I must go code in both C and M, not only C and VI,
And all I ask is a Linux box and a mouse to steer her by,
And the `while`'s `break` and the `if then`
    and the valgrind's shaking,
And a dash `O` so the C's fast and a `switch case break`-ing.

questions?