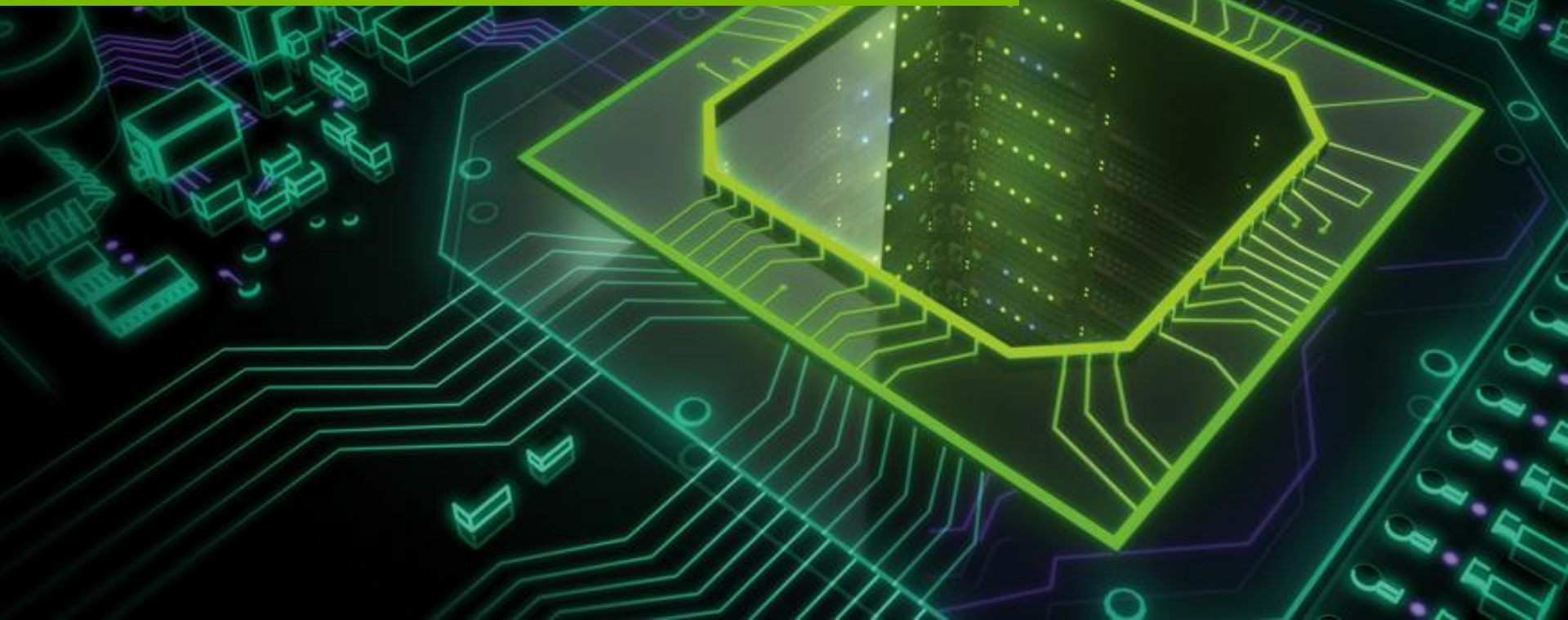
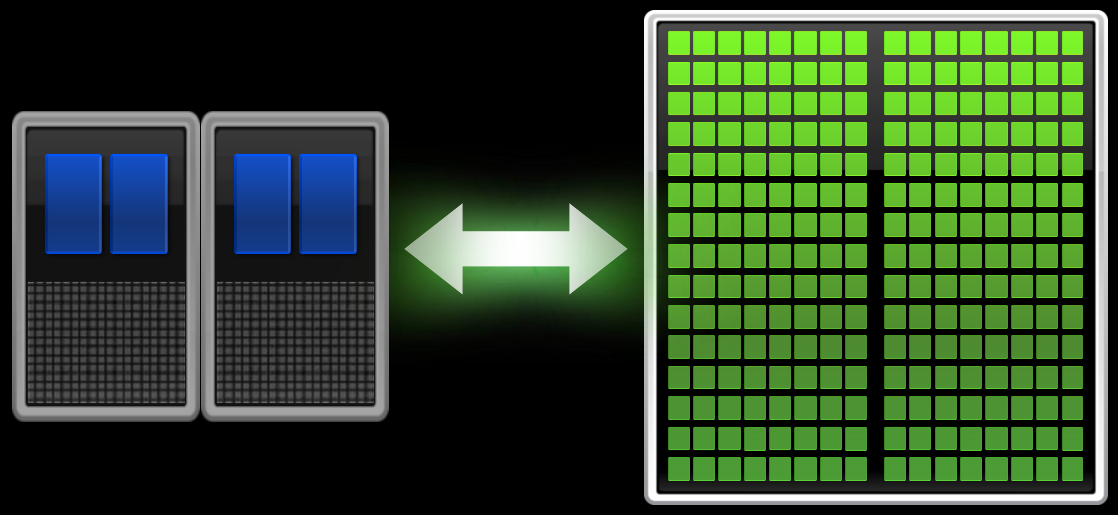
Solving Sparse Problems on GPUs

Michael Garland
NVIDIA Research



Heterogeneous Parallel Computing



Multicore CPU

Fast Serial Processing

Manycore GPU

Scalable Parallel Processing

GPUs are mainstream

Oil & Gas

Schlumberger

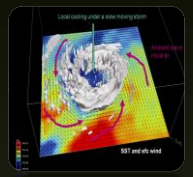
BR PETROBRAS

Total

Chevron

Paradigm

Edu/Research



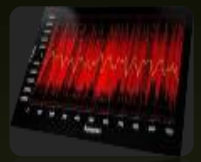
Chinese Academy of Sciences

Georgia Tech

HARVARD School of Engineering and Applied Sciences

Oak Ridge National Laboratory

Government



Air Force Research Laboratory

NASA

Naval Research Laboratory

BAE SYSTEMS

Life Sciences

Boston Scientific

MGH 1811 Mass General Hospital

Max Planck Institute

Beckman Coulter

Finance

Bloomberg

BNP PARIBAS

STANDARD LIFE

SciComp Inc.

NumeriX

Manufacturing

Agilent

ANSYS

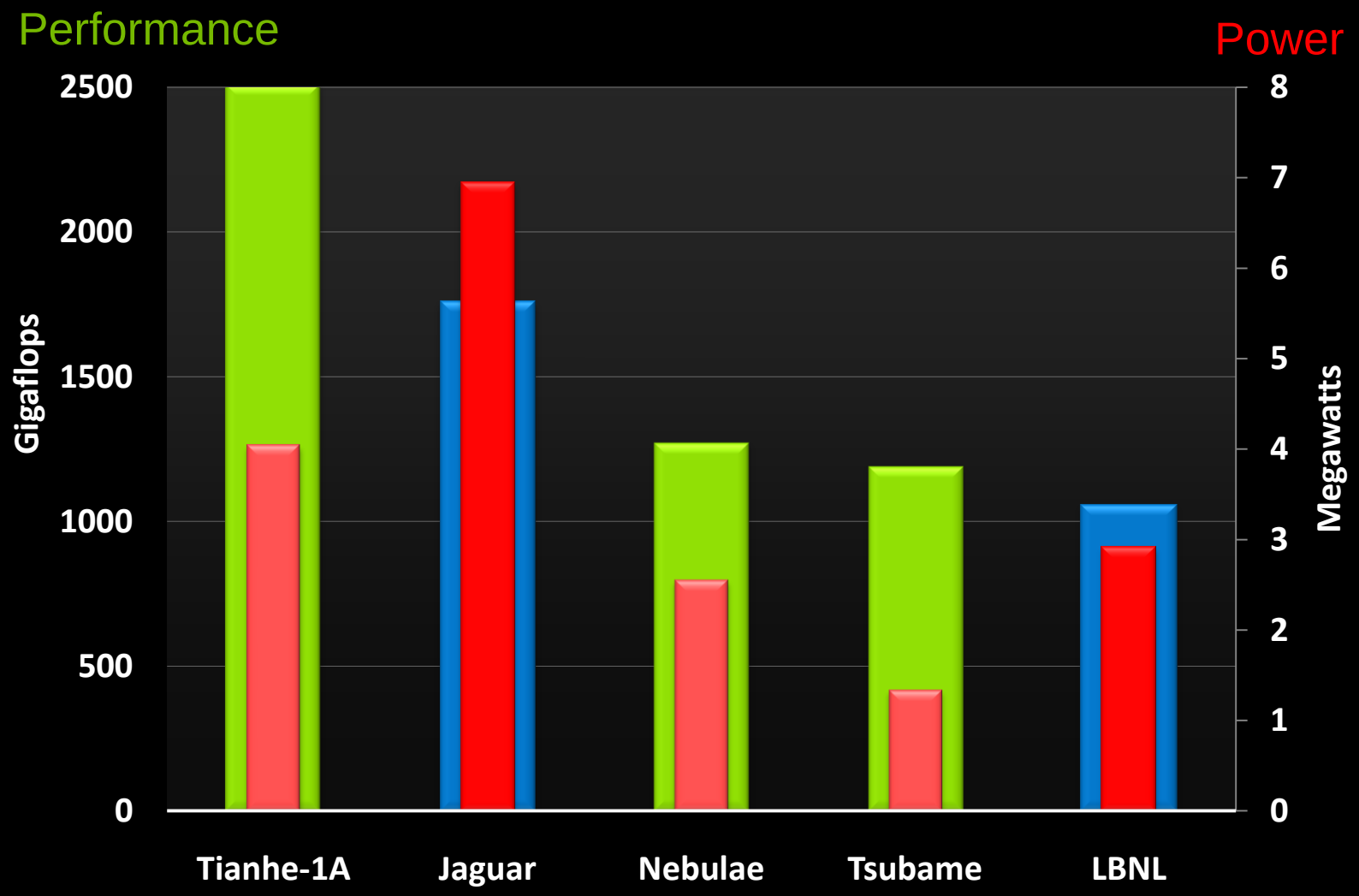
Autodesk

SIMULIA

ACUSIM SOFTWARE

GPUs are power efficient

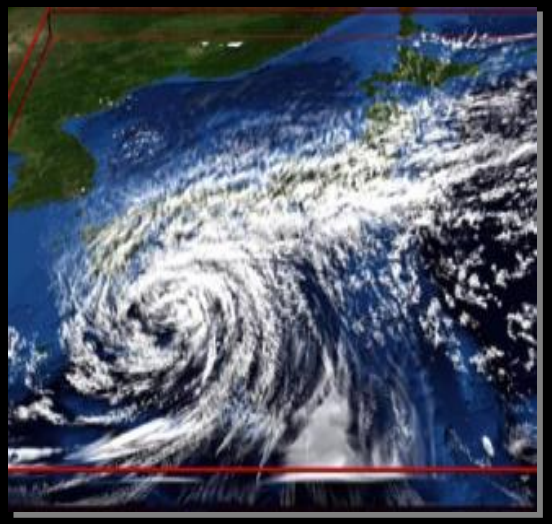
Nov. 2010 Top 500 List



GPU-CPU Supercomputer CPU only Supercomputer Power

GPUs do science

ASUCA Weather Modeling



3990 Tesla GPUs

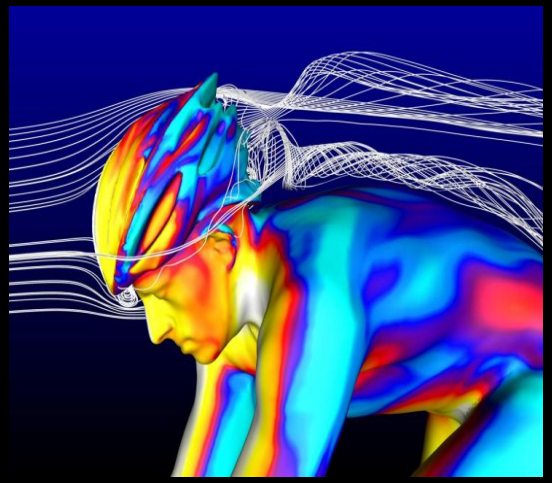
76.1 Tflops

Blood Flow Simulations

4000 Tesla GPUs

600 Tflops

Himeno: Navier Stokes



1024 Tesla GPUs

7.9 Tflops

Large-scale GPU simulations on Tsubame

What about problems that are

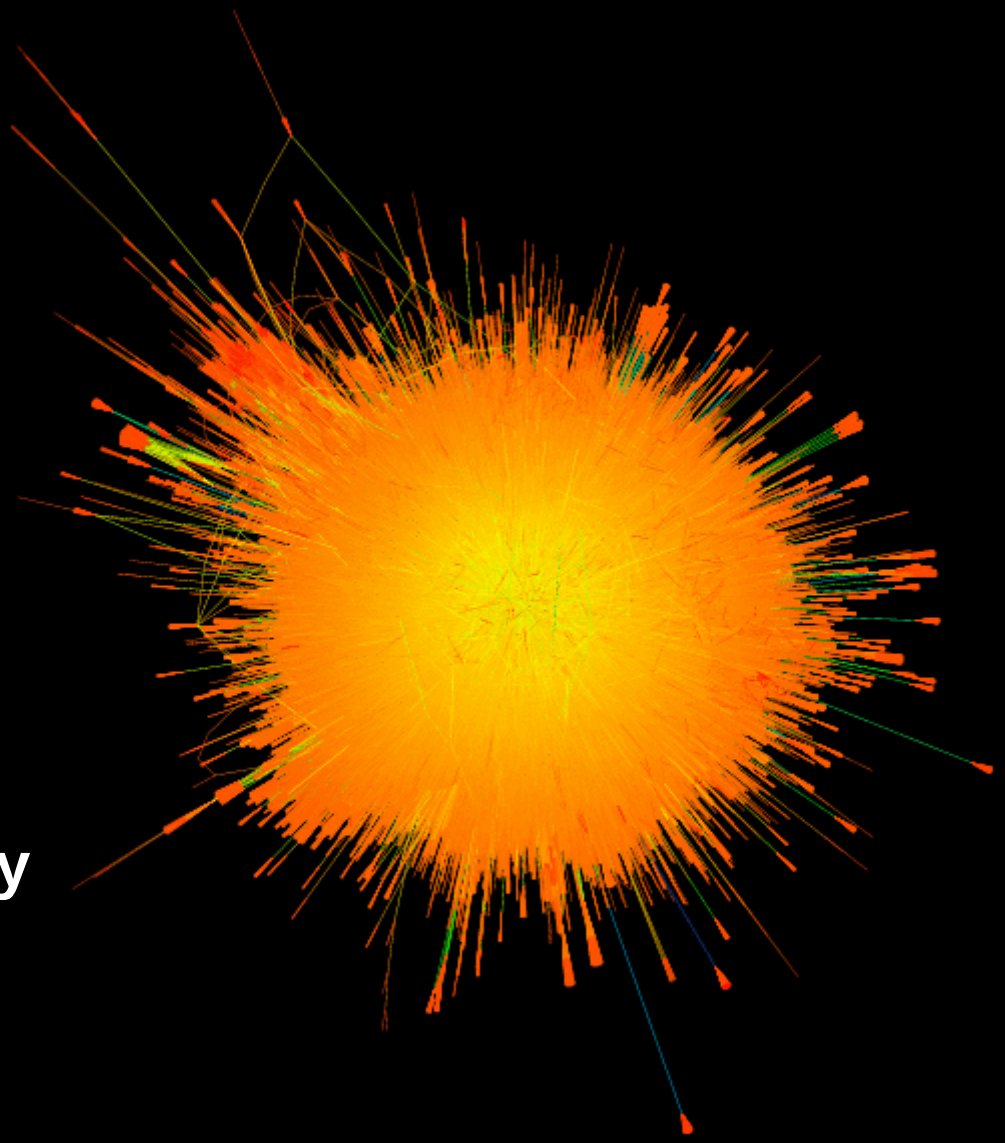
- Sparse
- Irregular
- Unstructured

with Duane Merrill, University of Virginia

BREADTH-FIRST SEARCH

Breadth-First Search on Graphs

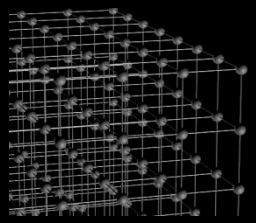
- Pick a source node.
- Rank every vertex by the length of shortest path from source.
- Or label every vertex by predecessor in this traversal order.

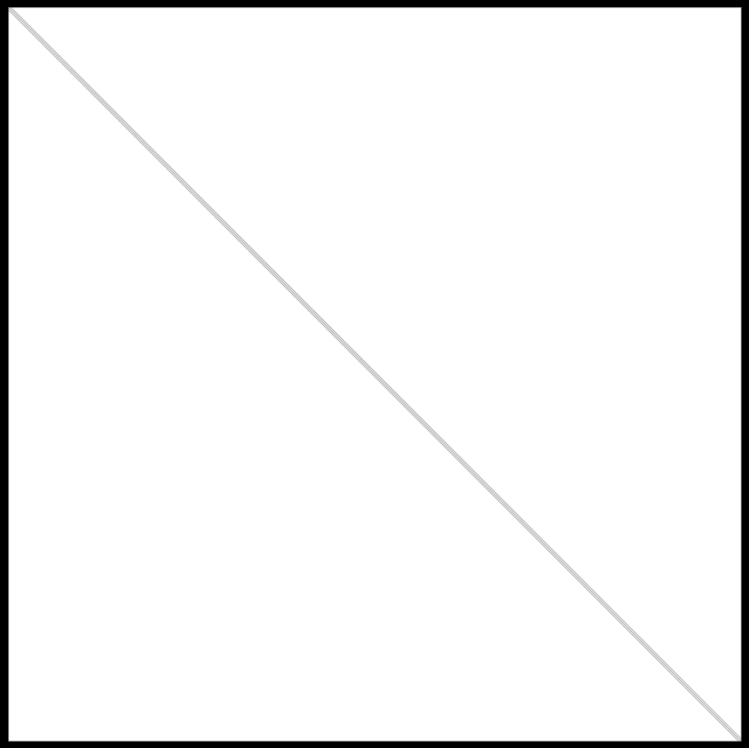


Gleich@wikipedia-20070206. 5797207 nodes, 44930161 edges.

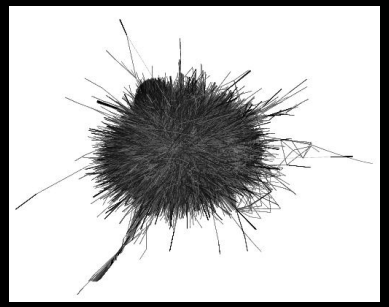
Arbitrary locality of reference

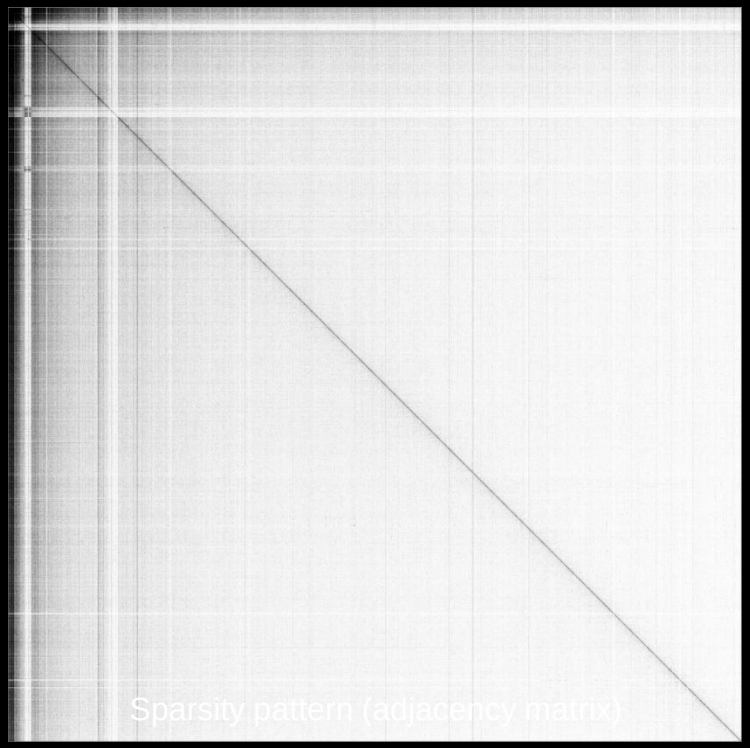
3D 7pt Poisson Lattice





Wikipedia, 2007





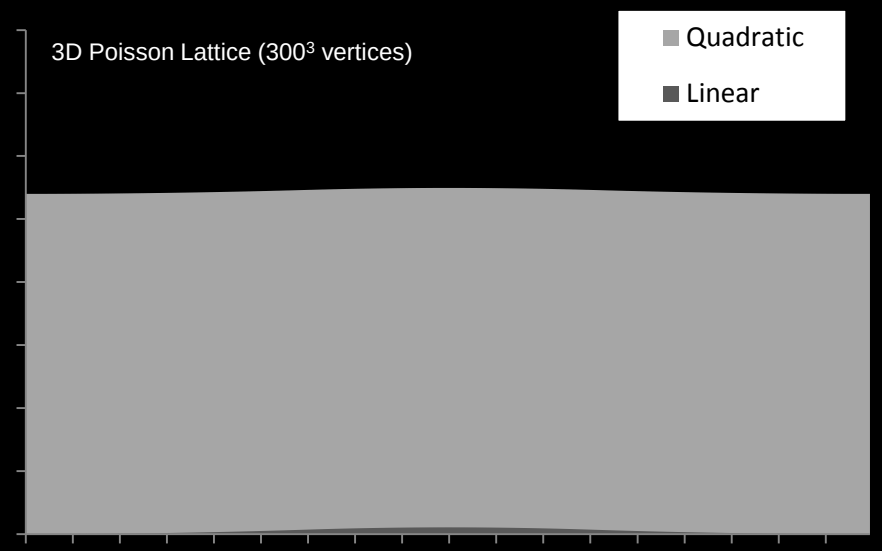
BFS Algorithm: Quadratic vs. Linear

Quadratic

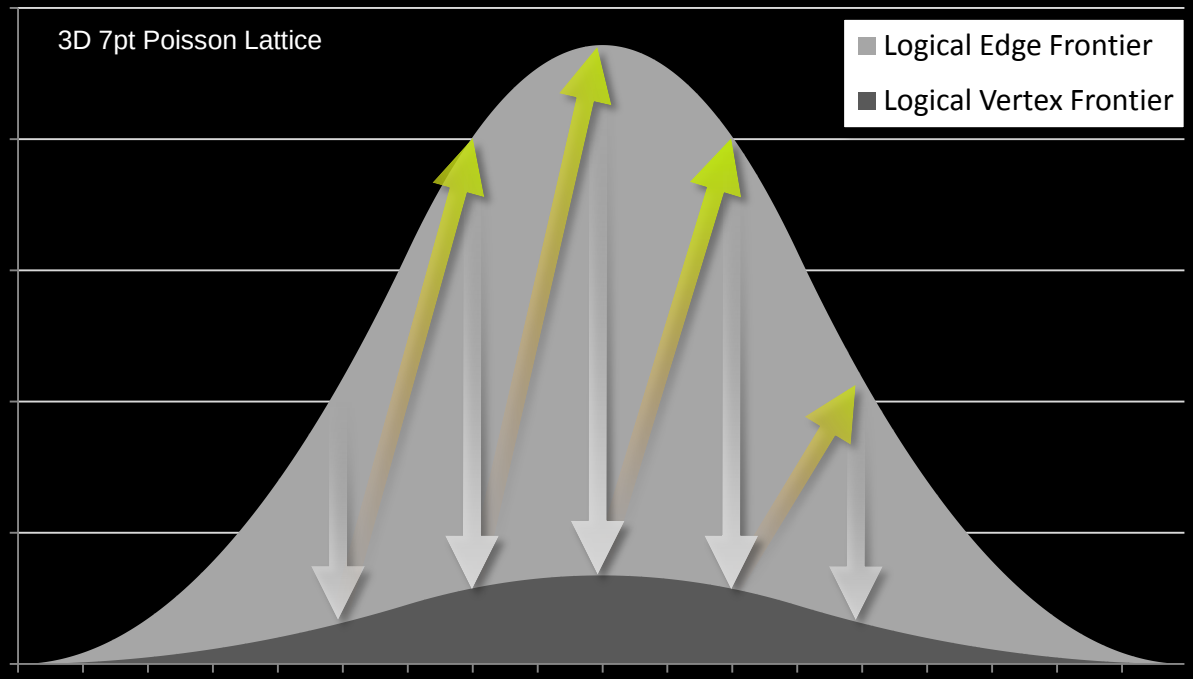
- Inspect every vertex at every BFS iteration to see if it was visited during the previous iteration
- $O(|V|^2 + |E|)$ work
- *"Stencil-oriented"*
  - Trivially matches GPU machine model
- *Unsuitable for diameter > O(5)*

Linear

- Each BFS iteration only inspects neighbors of vertices newly-discovered during the last iteration
- $O(|V| + |E|)$ work
- *"Allocation-oriented"*
  - Considered ill-suited for GPU machine model



Non-uniform & dynamic workloads



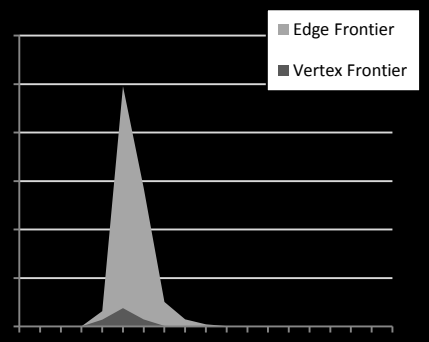
Logical vertex frontier

- **The unique vertices discovered during the current iteration**

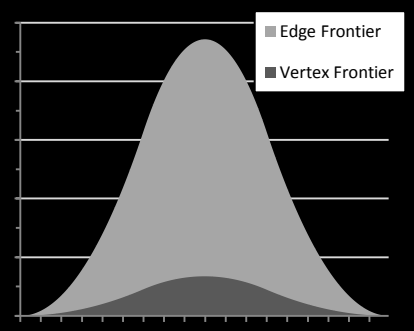
Logical edge frontier

- **The neighbors of the previous iteration's vertex frontier**

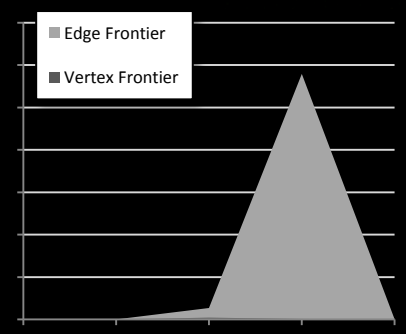
Substantial dataset variety



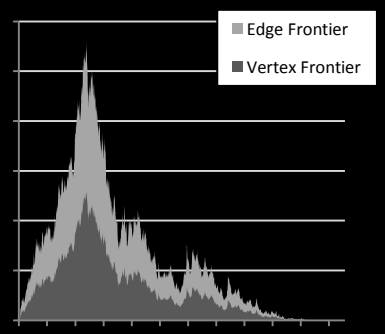
Wikipedia
(social)



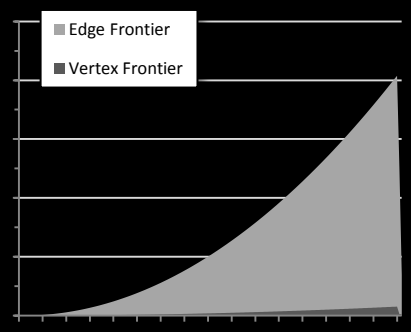
3D Poisson grid
(cubic lattice)



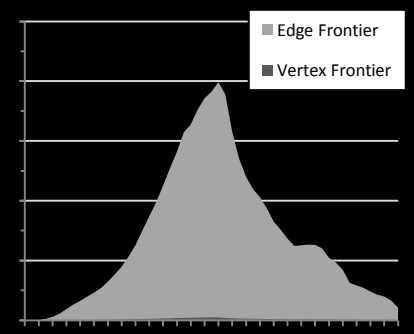
R-MAT
(random, power-law, small-world)



Street map: Europe
(Euclidian space)



PDE-constrained optimization
(non-linear KKT)



Auto transmission manifold
(tetrahedral mesh)

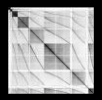
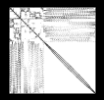
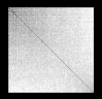
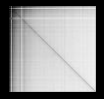
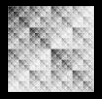
Implementation issues

- **Expose sufficient parallelism**
  - **globally distribute expansion/contraction in DRAM queue**
- **Load imbalance between processing elements**
  - **edge-oriented rather than vertex-oriented algorithm**
  - **“vectorize” tasks across threads**
- **Bandwidth inefficiency due to scattered load/store**
  - **isolate irregular accesses from rest of computation**
  - **filter lookups with bitmask to better leverage cache**

Experimental corpus

	Graph	Vertices (millions)	Edges (millions)
europe.osm	Road network	50.9	108.1
grid5pt.5000	2D Poisson stencil	25.0	125.0
hugebubbles-00020	2D mesh	21.2	63.6
grid7pt.300	3D Poisson stencil	27.0	188.5
nlpkkt160	Constrained optimization problem	8.3	221.2
audikw1	Finite element matrix	0.9	76.7
cage15	Transition prob. matrix	5.2	94.0
kkt_power	Optimization (KKT)	2.1	13.0
coPapersCiteseer	Citation network	0.4	32.1
wikipedia-20070206	Wikipedia page links	3.6	45.0
kron_g500-logn20	Graph500 random graph	1.0	100.7
random.2Mv.128Me	Uniform random graph	2.0	128.0
rmat.2Mv.128Me	RMAT random graph	2.0	128.0

Single-socket performance comparison

Graph	Spy Plot	Avg. Search Depth	**Sequential** Intel Sandybridge†	**Parallel** NVIDIA Tesla C2050		**Parallel** Intel Nehalem
			10^9 TE/s	**10^9 TE/s**	**Parallel speedup**	**10^9 TE/s**
europe.osm		19314	0.03	0.3	11x	
grid5pt.5000		7500	0.08	0.6	7.3x	
hugebubbles		6151	0.03	0.4	15x	
grid7pt.300		679	0.04	1.1	28x	0.12 *(4-core††)*
nlpkkt160		142	0.26	2.5	10x	0.47 *(4-core††)*
audikw1		62	0.65	3.0	4.6x	
cage15		37	0.13	2.2	18x	0.23 *(4-core††)*
kkt_power		37	0.05	1.1	23x	0.11 *(4-core††)*
coPapersCiteseer		26	0.50	3.0	5.9x	
wikipedia-2007		20	0.07	1.6	25x	0.19 *(4-core††)*
kron_g500-logn20		6	0.24	3.1	13x	
random.2Mv.128Me		6	0.10	3.0	29x	0.50 *(8-core†††)*
rmat.2Mv.128Me		6	0.15	3.3	22x	0.70 *(8-core†††)*

† 3.4GHz Core i7 2600K

†† 2.5 GHz Core i7 4-core, Leiserson *et al.*

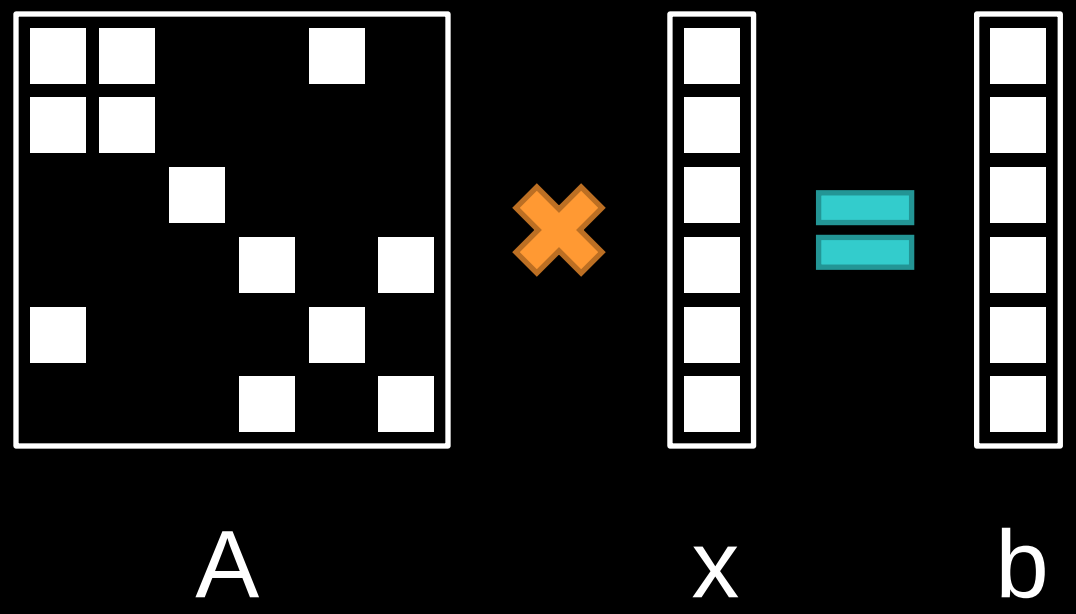
††† 2.7 GHz Xeon X5570 8-core, Agarwal *et al.*

Nathan Bell (NVIDIA), Steven Dalton & Luke Olson (UIUC)

ALGEBRAIC MULTIGRID

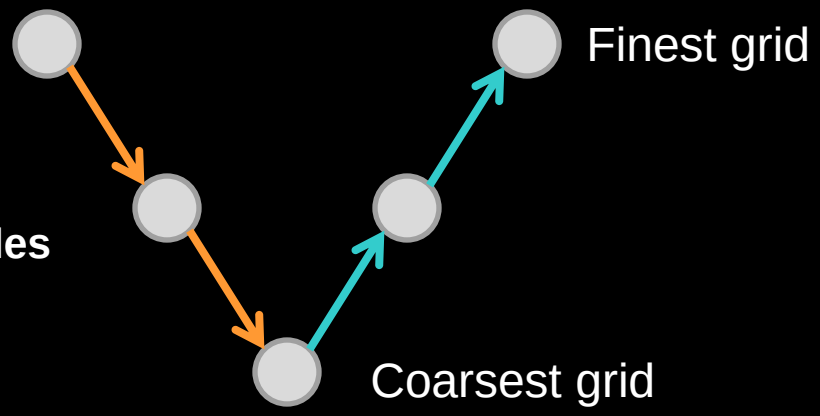
Objective

- **Solve (certain) sparse linear systems *very quickly***
  - **Optimal complexity O(N)**



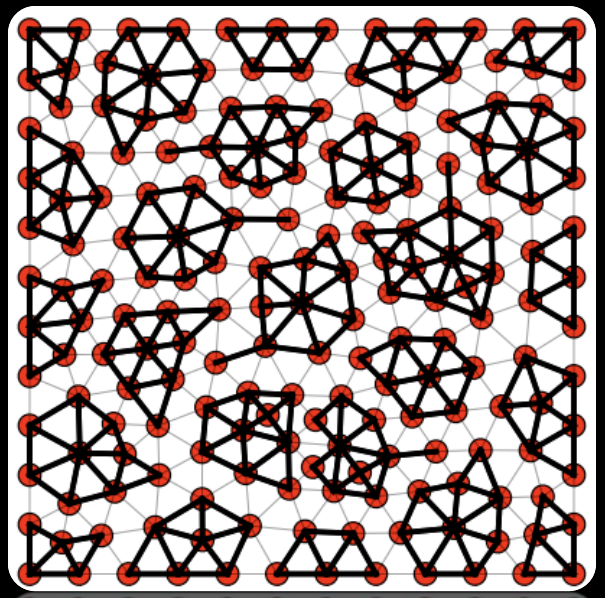
Multigrid in a Nutshell

- **Setup Phase**
  - Construct a hierarchy of grids
  - Sequence of coarser versions of the problem
- **Cycling/Solve Phase**
  - Reduce high-frequency errors with relaxation
  - Restrict remaining low-frequency errors to coarse grid
  - Interpolate coarse-grid solution back to fine grid
- **Applied recursively**
  - Eliminates all error modes
  - Results in V-cycle
  - Converges to solution in multiple cycles



Algebraic Multigrid (AMG)

- Constructs “grids” directly from sparse matrix
  - Hierarchy of sparse matrices
  - Requires no geometric knowledge
- Aggregation-based AMG
  - Coarsens clusters of nodes
  - Works for unstructured meshes



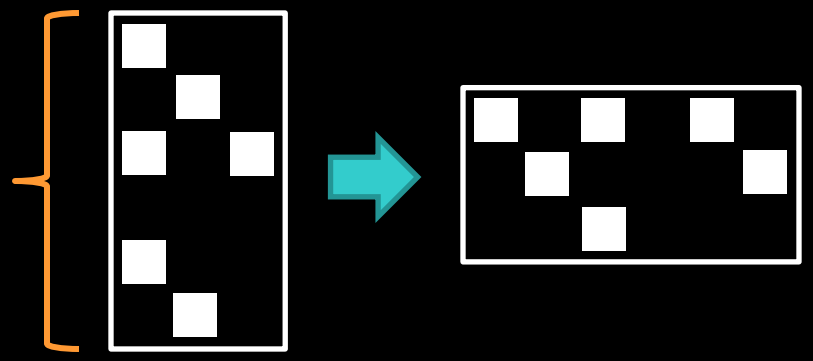
AMG on the GPU

- **Implemented both Setup and Cycling on GPU**
  - **Need to identify *fine-grained* parallelism everywhere**
  - **Distill complex algorithms into *parallel primitives***
- **Setup Phase**
  - **Sparse matrix-matrix multiplication (C = A * B)**
  - **Sparse matrix transpose**
  - **Matrix format conversions, parallel aggregation, etc.**
- **Cycling Phase**
  - **Sparse matrix-vector multiplication (y = A * x)**
  - **Level 1 BLAS operations (e.g. dot product)**

Example: Transpose

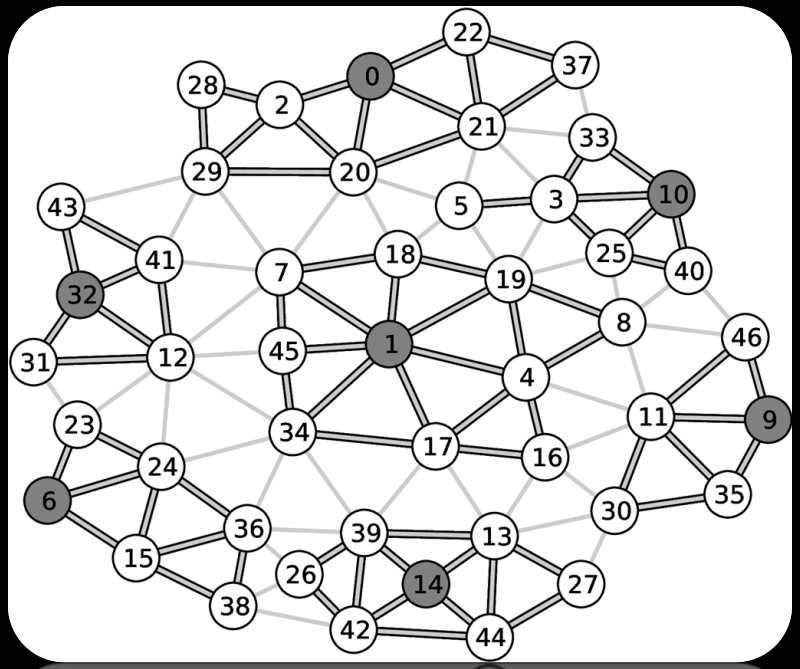
- Matrix in coordinate format (COO)
- Sort rows and values by column index
- Implemented with `thrust::sort_by_key`

0 1 2 2 4 5	Row indices
0 1 0 2 0 1	Column indices
A B C D E F	Nonzero values



Example: Aggregation

- Compute MIS(2) in parallel with extension of Luby's method
- Create aggregates around each MIS(2) node
- Close analog of standard greedy aggregation scheme



Performance Study

- Eight matrix example
  - Isotropic Poisson Problems
  - Structured and Unstructured
- GPU System
  - Tesla C2050 GPU
  - CUDA 4.0
  - Thrust v1.4
  - Cusp v0.2
- CPU System
  - Intel Core i7 950 CPU
  - MKL v10.3
- Reference Solver
  - Trilinos/ML v5.0

Matrix	Notes	Rows	Nonzeros
1a	2D FD, 5-point	~1M	~5M
1b	2D FE, 9-point	~1M	~9M
2a	3D FD, 7-point	~1M	~7M
2b	3D FE, 27-point	~1M	~27M
3a	2D FE, h=0.03	~500K	~4M
3b	2D FE, h=0.02	~1M	~8M
3c	2D FE, h=0.015	~2M	~15M
4	3D FE, h=0.15	~1M	~17M

Individual Components

Algorithm	Speedup
Dot Product	6.62x
Vector Addition	6.35x
Sparse Transpose*	2.90x
Sparse Matrix-Vector Multiply*	6.00x
Sparse Matrix-Matrix Multiply*	1.67x

*Average speedup across eight example matrices

Setup Phase

Matrix	Type	CPU	GPU	Speedup	Trilinos/ML
1a	2D FD	892 ms	518 ms	1.72x	2040 ms
1b	2D FE	1133 ms	649 ms	1.75x	2298 ms
2a	3D FD	1639 ms	944 ms	1.74x	2906 ms
2b	3D FE	2845 ms	2124 ms	1.34x	4420 ms
3a	2D FE	657 ms	335 ms	1.96x	1324 ms
3b	2D FE	1484 ms	648 ms	2.29x	2785 ms
3c	2D FE	2901 ms	1151 ms	2.52x	5236 ms
4	3D FE	3157 ms	1726 ms	1.83x	4967 ms

Cycling Phase

Matrix	Type	CPU	GPU	Speedup	Trilinos/ML
1a	2D FD	1221 ms (20)	423 ms (51)	**7.66x**	14,190 ms (33)
1b	2D FE	1097 ms (16)	461 ms (46)	**7.52x**	10,590 ms (22)
2a	3D FD	1760 ms (23)	295 ms (27)	**6.76x**	14,800 ms (31)
2b	3D FE	1683 ms (14)	482 ms (24)	**5.98x**	13,840 ms (20)
3a	2D FE	1534 ms (42)	286 ms (49)	**5.40x**	14,020 ms (53)
3b	2D FE	3704 ms (47)	633 ms (54)	**5.77x**	34,410 ms (68)
3c	2D FE	7804 ms (53)	1424 ms (65)	**5.75x**	44,530 ms (65)
4	3D FE	4369 ms (43)	1498 ms (50)	**2.96x**	28,380 ms (47)

Notes
- Time to solve Ax=b to 1e-10 relative tolerance
- AMG as a preconditioner to CG solver
- Iteration counts shown in parentheses
- Reporting per-iteration speedup

Summary

- **Fully-Parallelized AMG on GPU**
  - **1.89x Speedup in Setup Phase (average)**
  - **5.89x Speedup in Cycling Phase (average)**

- **References**

"Exposing Fine-Grained Parallelism in Algebraic Multigrid Methods"
Nathan Bell (NVIDIA), Steven Dalton (UIUC), Luke Olson (UIUC),
NVIDIA Technical Report NVR-2011-002, June 2011
http://research.nvidia.com/publications

Cusp Library
http://cusp-library.googlecode.com

Thrust Library
http://thrust.googlecode.com

Final Thoughts

- Sparse problems can be solved by
  - exposing sufficient fine-grained parallelism
  - managing load imbalance
  - efficiently using available bandwidth
- Details available in:
  - technical reports
  - open source implementations
  - see http://research.nvidia.com

Questions?

mgarland@nvidia.com

http://research.nvidia.com