GPU TECHNOLOGY CONFERENCE

ACCELERATING SPARSE CHOLESKY FACTORIZATION ON THE GPU

STEVE RENNICH, SR. ENGINEER, NVIDIA DEVELOPER TECHNOLOGY

DARKO STOSIC, PHD CANDIDATE, UNIV. FEDERAL DE PERNAMBUCO

TIM DAVIS, PROFESSOR, CSE, TEXAS A&M UNIVERSITY

SPARSE MATRIX FACTORIZATION ON GPUs

- Objective:
  - Find methods for GPU acceleration of Sparse Cholesky Factorization
    - Experiment using SuiteSparse 4.4.3 / CHOLMOD
- Outline
  - Sparse Cholesky Factorization
  - Previous work / Issues
  - ‘Branches’ approach

DIRECT SPARSE FACTORIZATION

- Dense block Cholesky

$$\begin{bmatrix} A_{11} & A^t_{21} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & A^*_{22} \end{bmatrix} \begin{bmatrix} L^t_{11} & L^t_{21} \\ 0 & I \end{bmatrix}$$

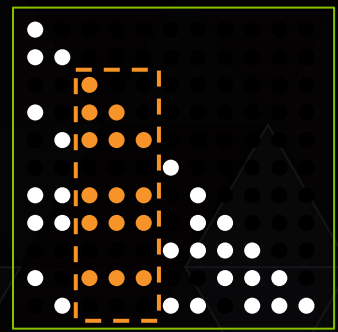
$L_{11} L^t_{11} = A_{11}$ **POTRF** dense Cholesky

$L_{11} L^t_{21} = A^t_{21}$ **TRSM** triangular solve

$A^*_{22} = A_{22} - L_{21} L^t_{21}$ **GEMM** matrix multiplication

Schur complement

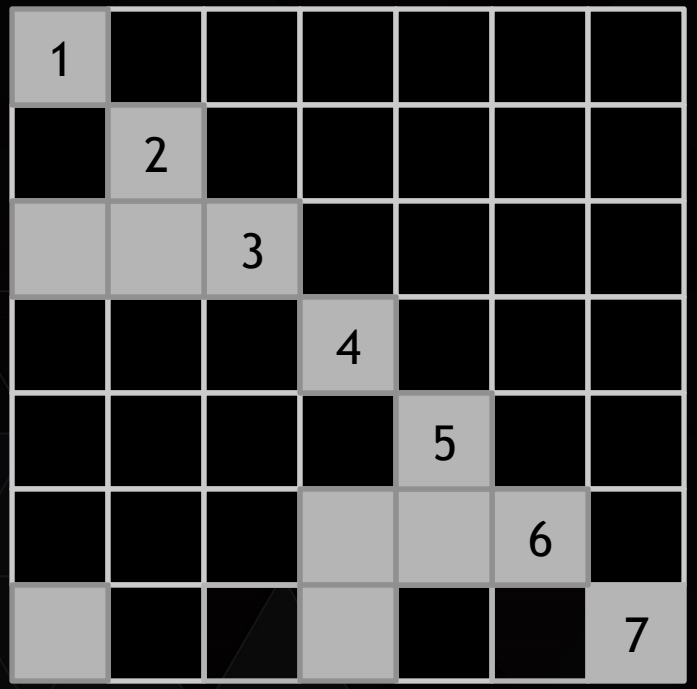
- Supernodes



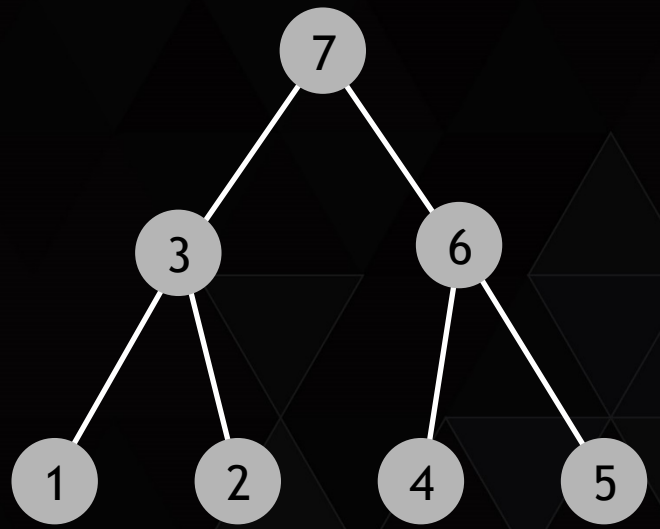
compressed column

DIRECT SPARSE FACTORIZATION

- Apply block Cholesky to supernodes
- ‘Left-looking supernodal’



- Elimination tree

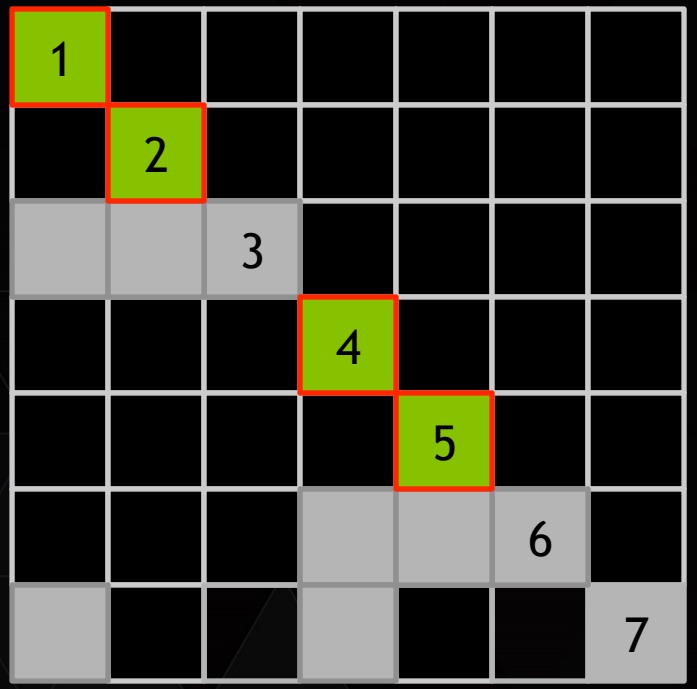


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

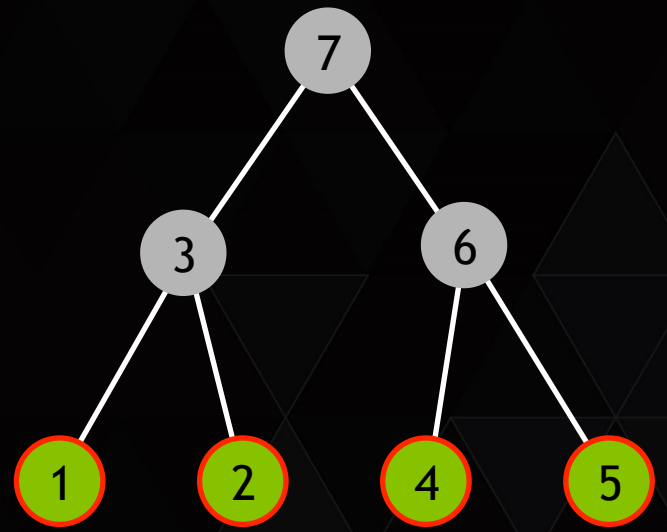
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF

- Elimination tree

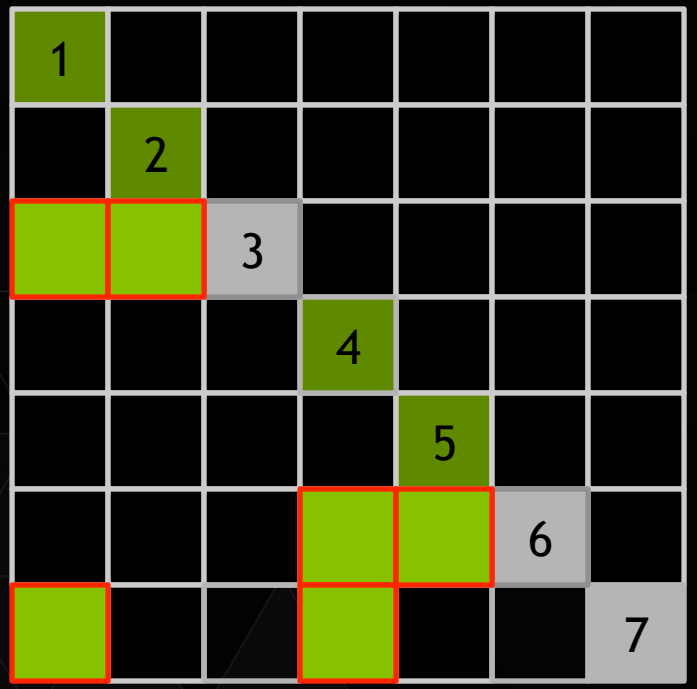


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

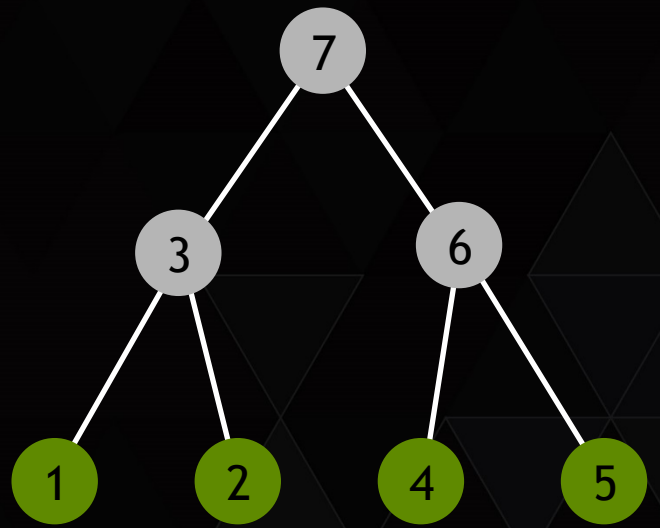
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF
TRSM

- Elimination tree

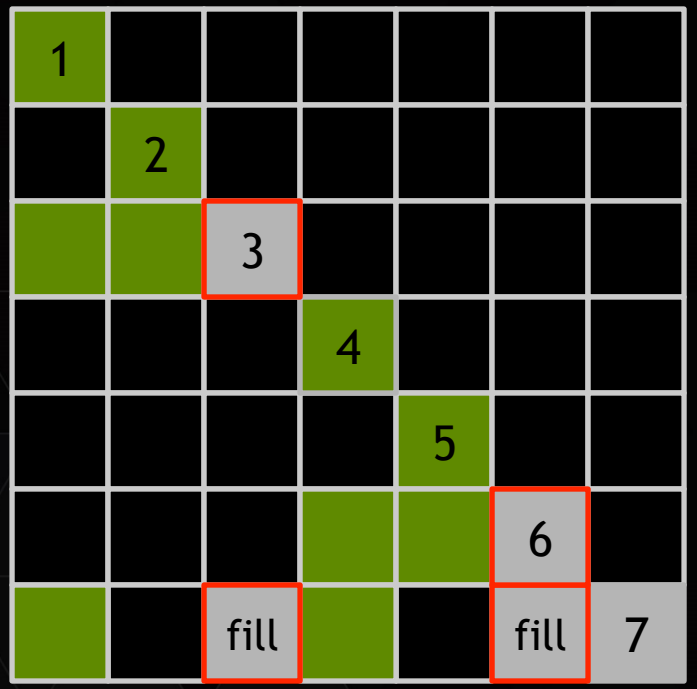


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

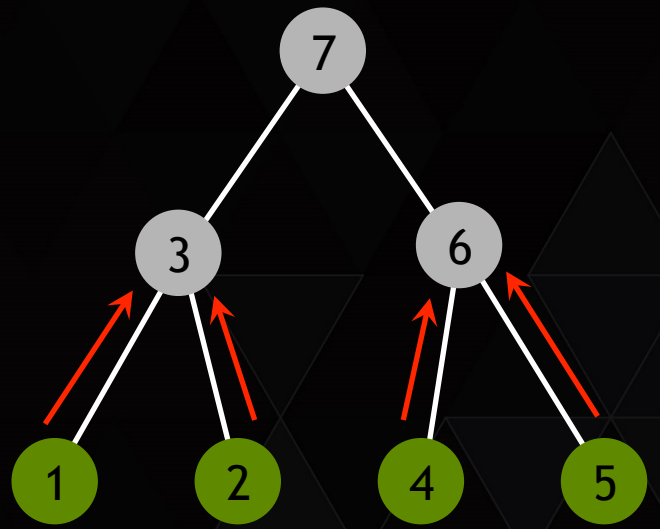
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF
TRSM
GEMM

- Elimination tree

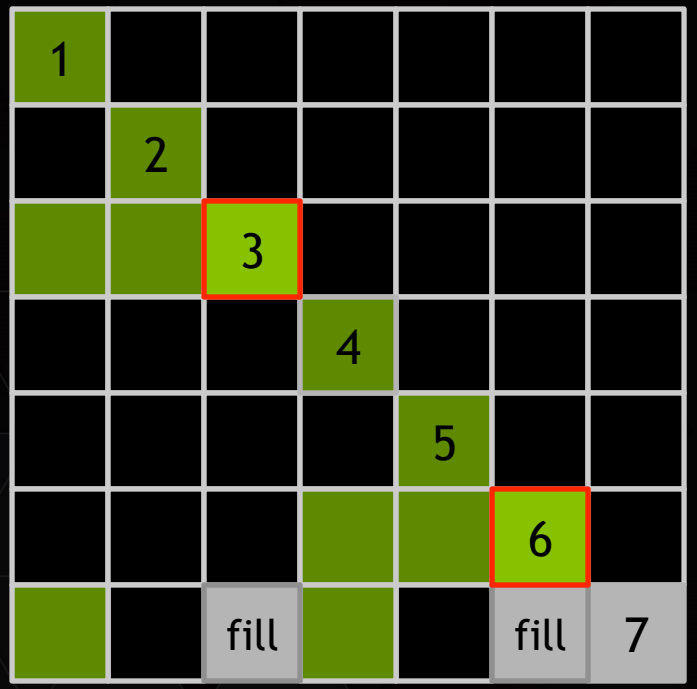


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

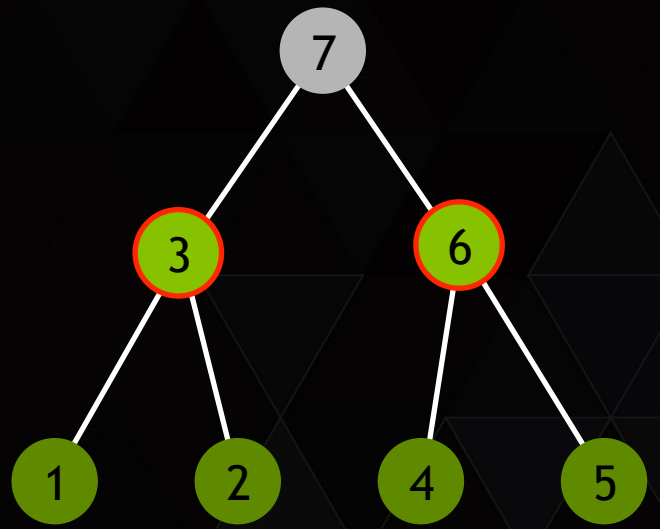
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF
TRSM
GEMM
POTRF

- Elimination tree

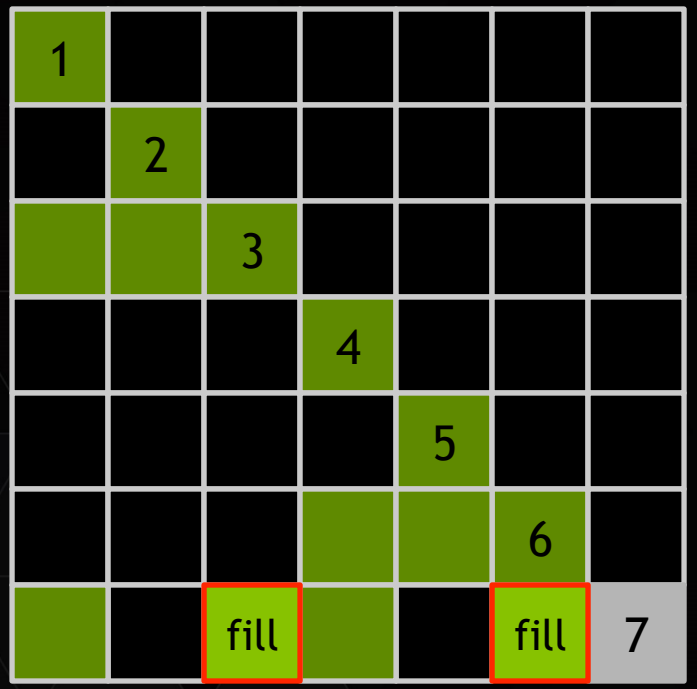


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

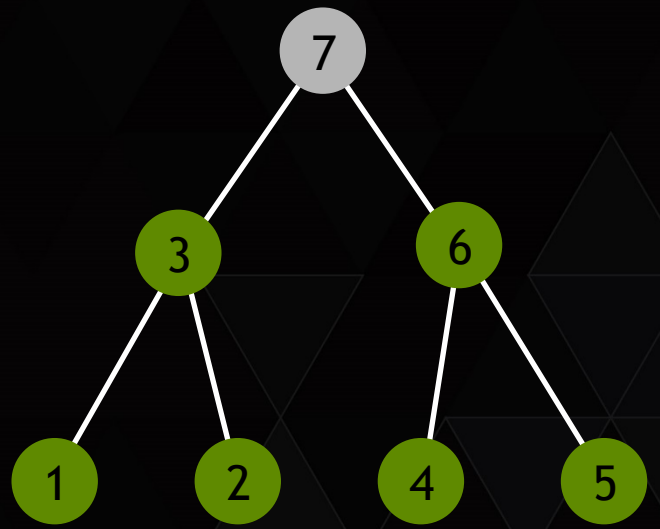
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF
TRSM
GEMM
POTRF
TRSM

- Elimination tree

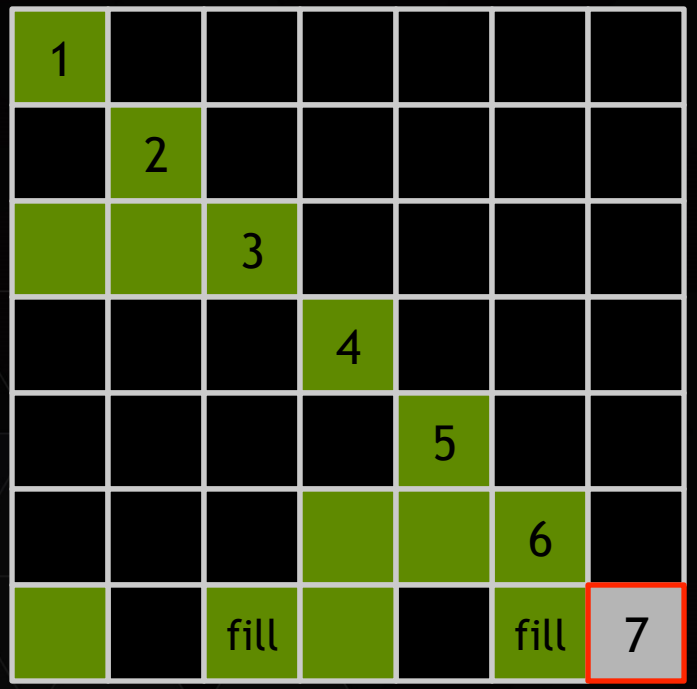


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

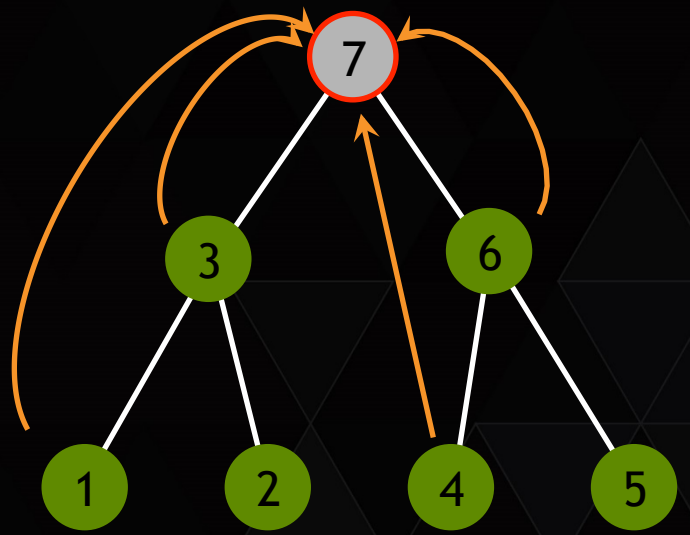
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF
TRSM
GEMM
POTRF
TRSM
GEMM

- Elimination tree

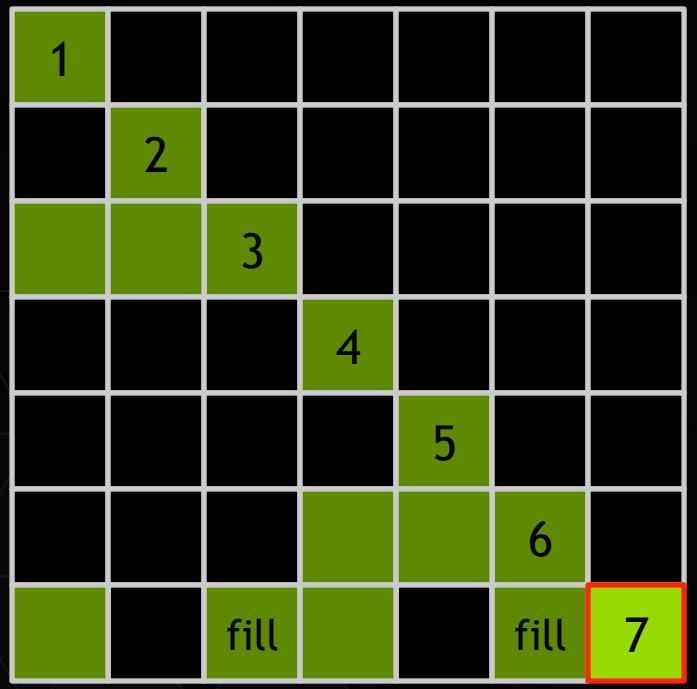


- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

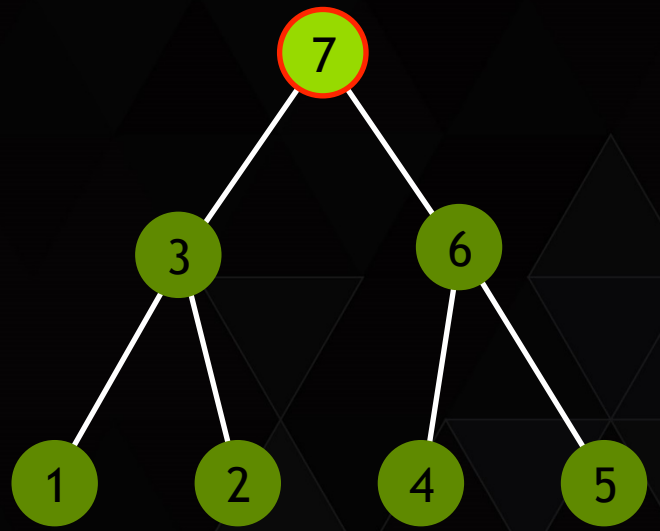
- Apply block Cholesky to supernodes

- ‘Left-looking supernodal’



POTRF
TRSM
GEMM
POTRF
TRSM
GEMM
POTRF

- Elimination tree



- Bulk of work is in assembling supernodes (wide range of descendant sizes)

DIRECT SPARSE FACTORIZATION

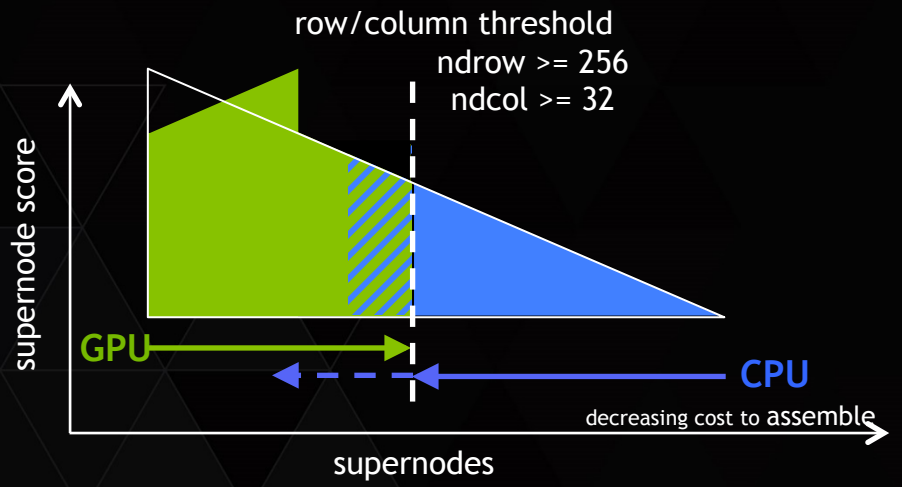
- Lots of ‘small’ math
- Irregular access patterns
- Larger matrices -> more dense math
- Greater connectivity -> more dense math
- Factors can be large (> 128 GB)

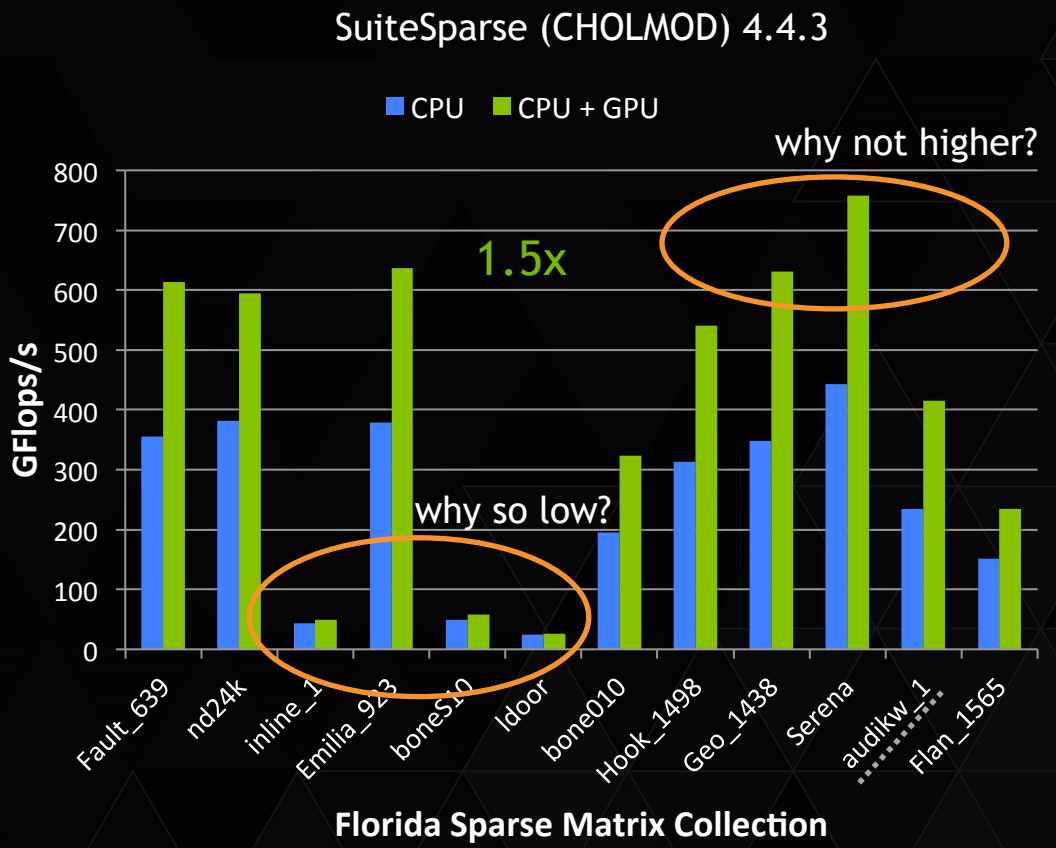
PREVIOUS WORK

- Just send large BLAS-3 to GPU
  - WORKS! For large, dense matrices
  - Not so good for:
    - small matrices
    - large matrices with low connectivity (shells / beams in FEA)
- Find methods for further GPU acceleration of Sparse Factorization

PREVIOUS WORK

- Send appropriately-sized BLAS calls to GPU
- 'hide' PCIe communication
- Assemble supernodes on GPU
- Hybrid computing



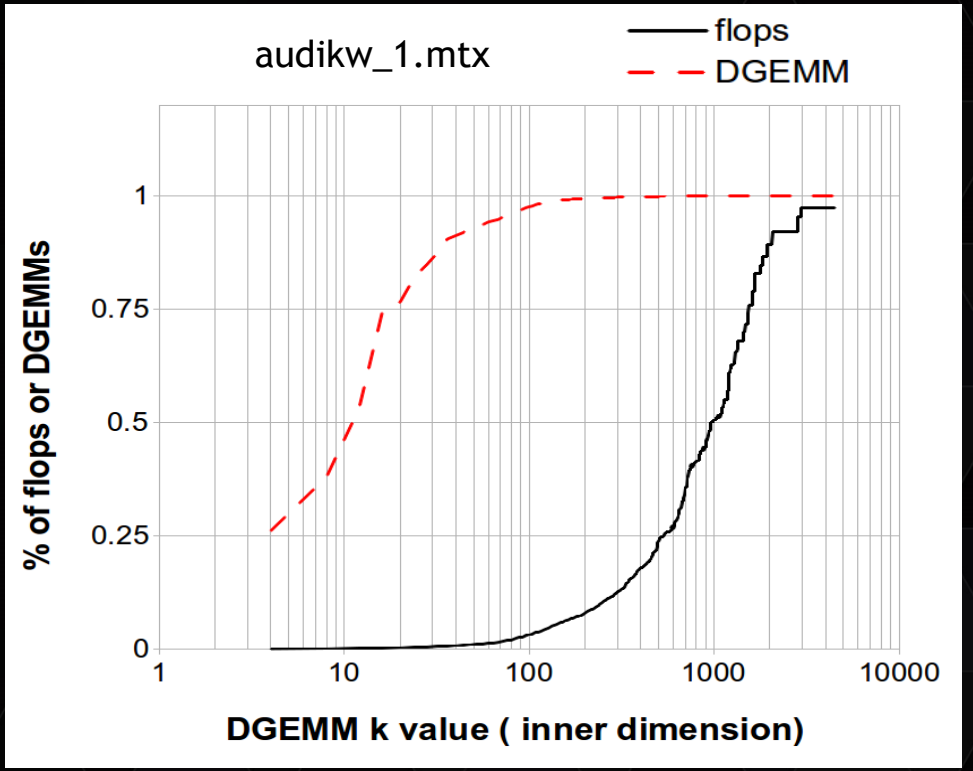


2 x Xeon E5-2698 v3 + K40 (max boost, ECC=off)
http://faculty.cse.tamu.edu/davis/suitesparse.html

ISSUES

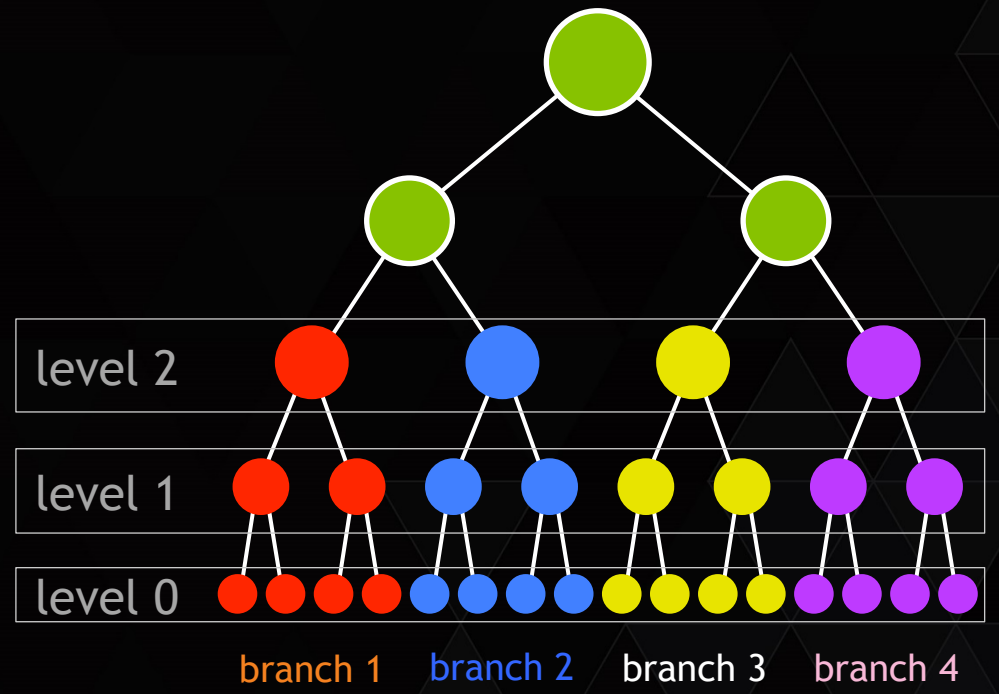
- PCIe communication
  - Limits which BLAS operations can be accelerated on GPU
- Small BLAS
  - Low occupancy
  - Launch overhead
- Most BLAS calls don't get sent to the GPU
- Seek methods which better accelerate factorization of small / minimally-connected matrices

% on CPU



PROPOSED SOLUTION

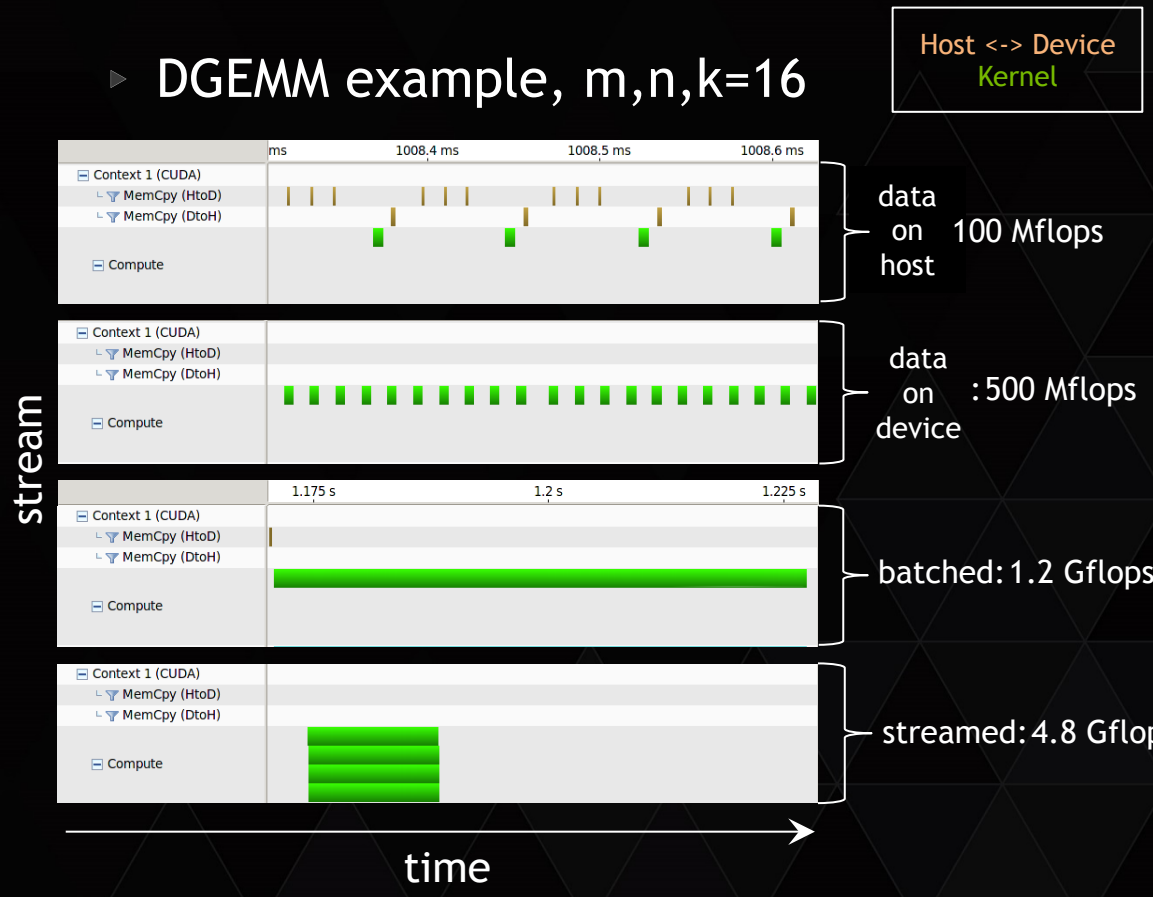
- Factor *branches* on GPU
  - *Use previous methods for root*
  - No use of CPU
  - Eliminates PCIe communication
  - Requires POTRF, TRSM & GEMM on GPU
- *Batch* and *stream* BLAS operations
  - Within levels
  - Amortizes launch overhead
  - Streamed to improve occupancy
- No size restriction
- Maps well to muti-GPU / hybrid computing



BATCHED / STREAMED BLAS

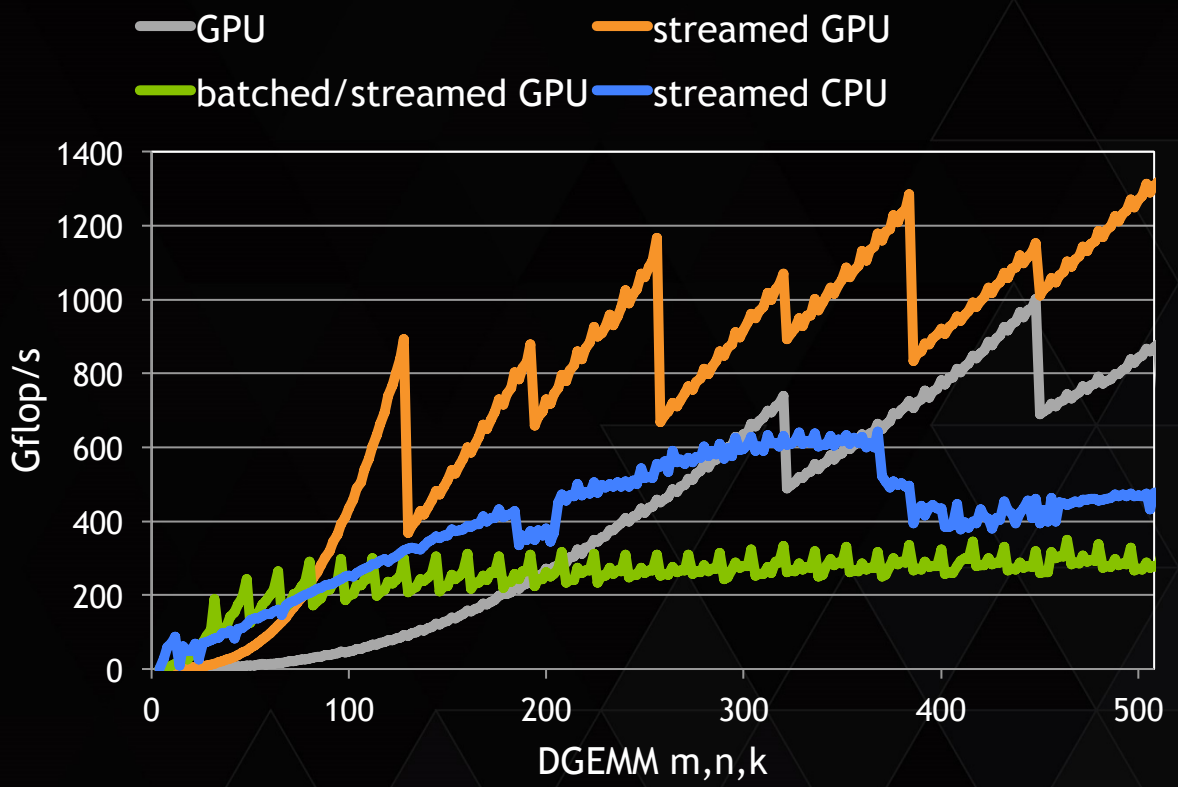
- Batch all BLAS calls to amortize kernel launch latency
- Stream multiple batches to increase occupancy
- Simply wrap cuBLAS subroutine with batch loop
- DGEMM w/ m,n,k=16 -> 40 GF

- DGEMM example, m,n,k=16



BATCHED / STREAMED DGEMM

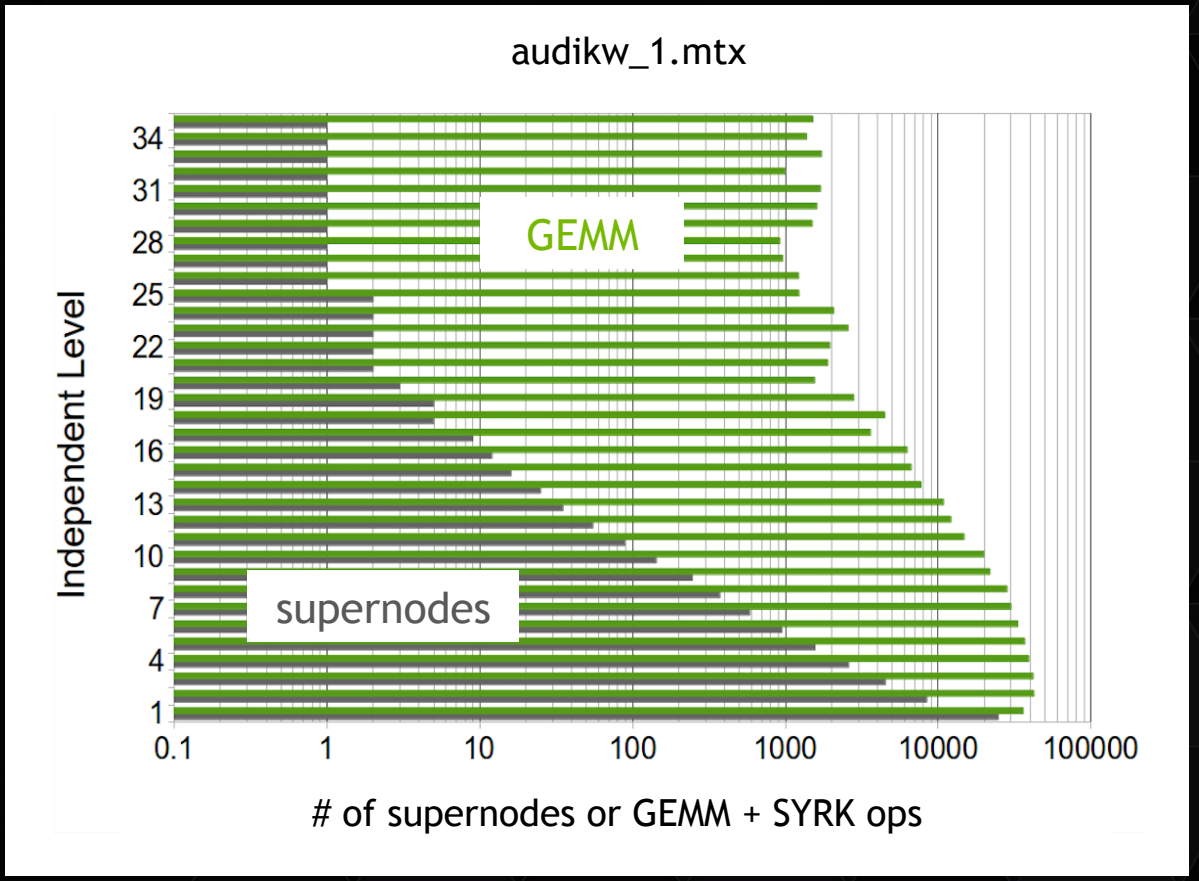
- Square DGEMM
- 64 streams/threads
- Batched / streamed cuBLAS performance matches MKL for small size
- Created by wrapping existing, non-batched routines
  - *passing lists*



2 x Xeon E5-2698 v3 + K40 (max boost, ECC=off)

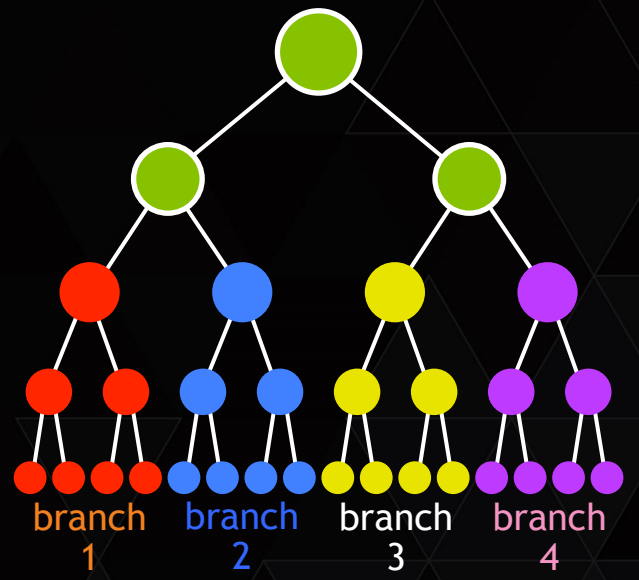
PLENTY OF PARALLELISM

- Lower levels
  - Many supernodes
  - Few descendants
- Upper levels
  - Few supernodes
  - Many descendants



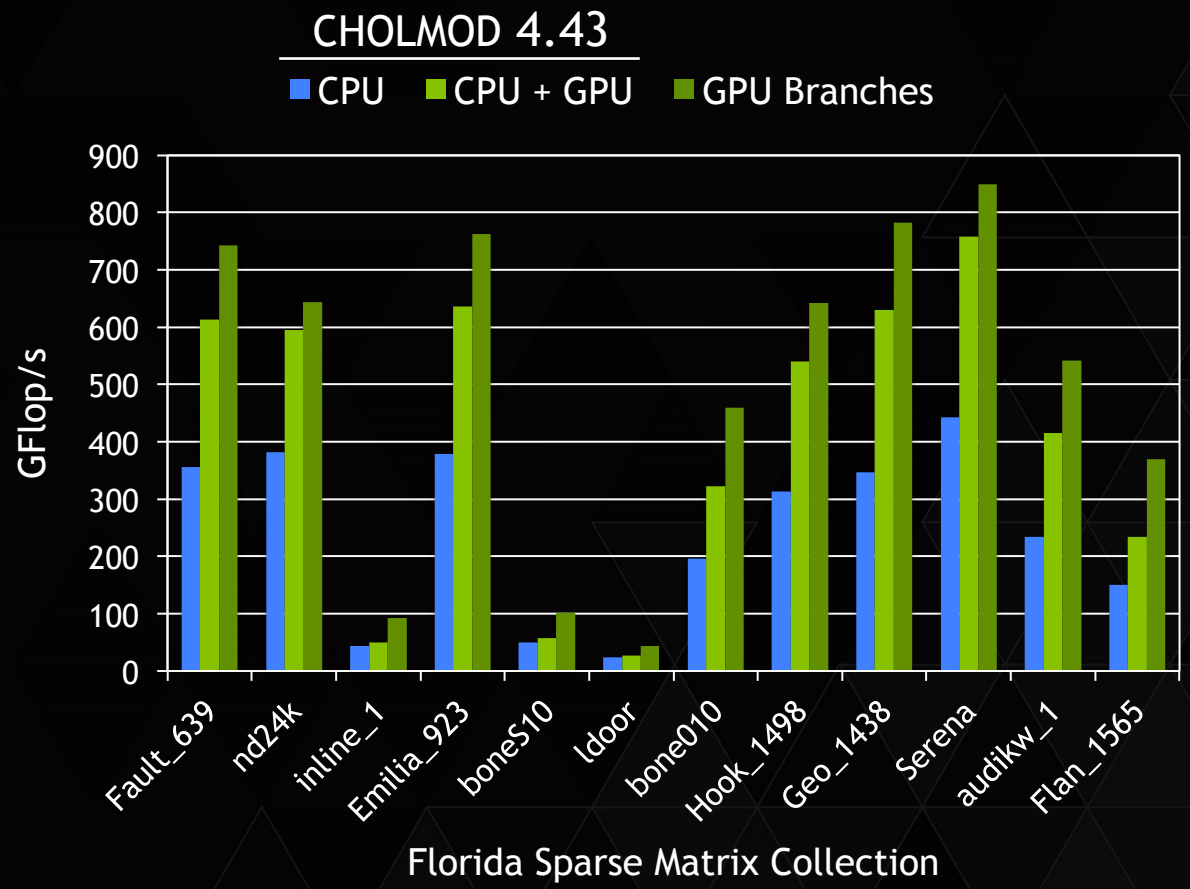
BRANCHES

Matrix	# branches	# levels	# supernodes	# root levels	# root supernodes
Fault_639	2	18-19	14931 - 15794	1	1
nd24k	2	11	302 - 325	1	1
inline_1	4	16-17	3909 - 10633	1	1
Emilia_923	4	17-18	10314 - 11570	3	4
boneS10	4	18-23	7045 - 26182	1	1
ldoor	3	19-20	17413 - 35704	1	1
bone010	6	16-20	1957 - 23610	1	1
Hook_1498	9	1-18	1 - 33608	3	5
Geo_1438	8	17-18	8102 - 9335	5	9
Serena	60	10-17	189 - 4910	10	60
audikw_1	4	17-19	5631 - 22300	1	1
Flan_1564	8	15-17	3937 - 16309	2	2



CHOLMOD RESULTS

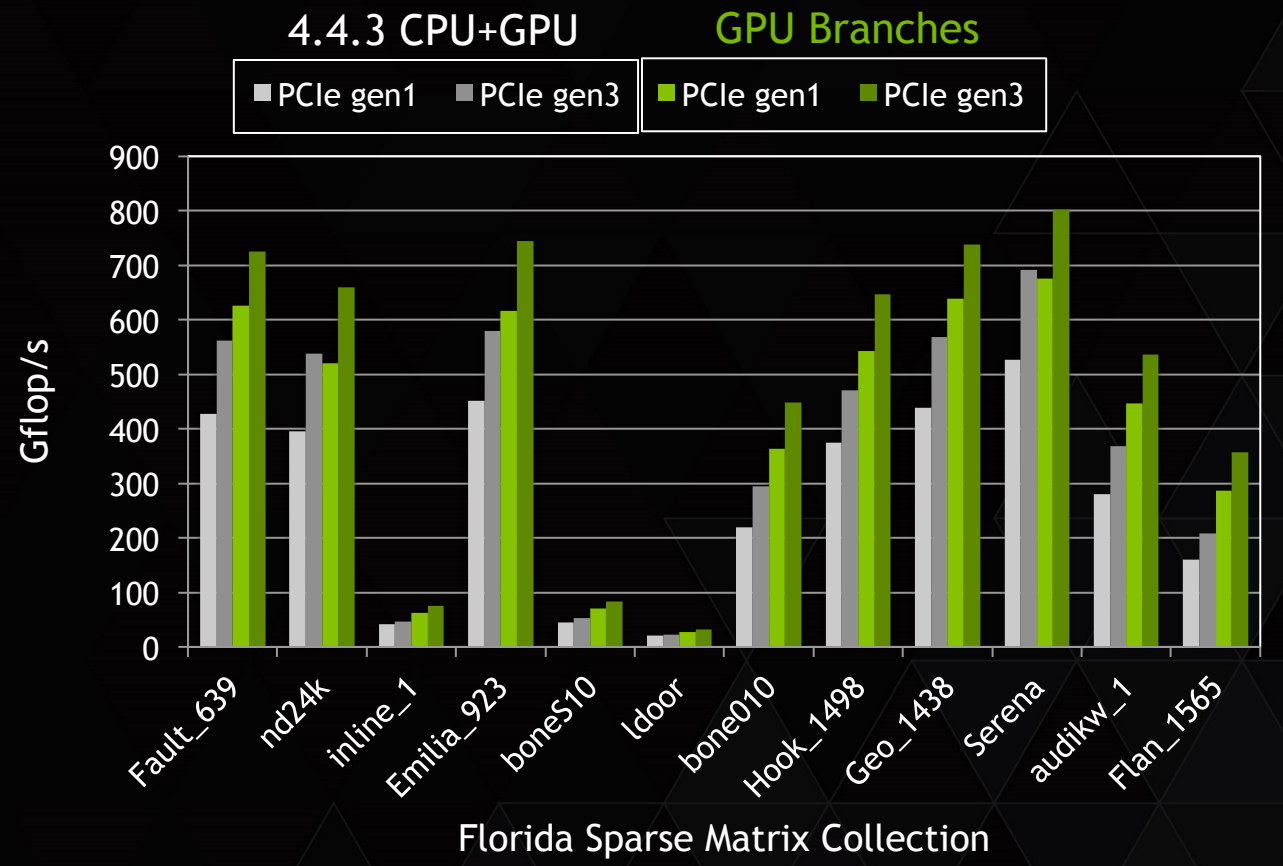
- 1.38x average speedup vs. previous CPU+GPU
- 2x average speedup vs. CPU
- Poorly performing matrices see the greatest speedup



2 x Xeon E5-2698 v3 + K40 (max boost, ECC=off)
http://faculty.cse.tamu.edu/davis/suitesparse.html

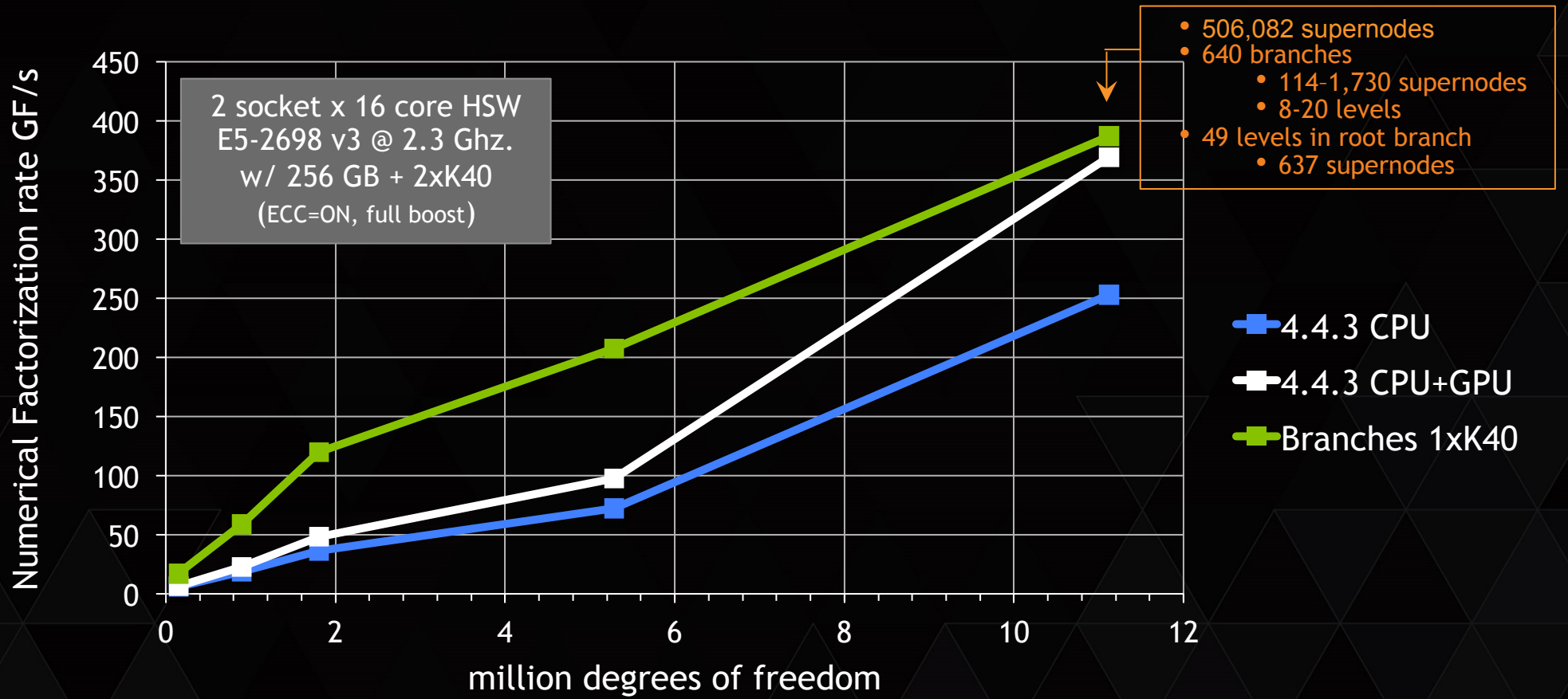
PCIe DEPENDENCE

- PCIe gen3 -> gen1
  - 12 GB/s -> 3 GB/s
  - 75% loss
- CPU+GPU
  - 23% loss
- Branches
  - 17% loss



1 x i7 3930K + K40 (max boost, ECC=on)

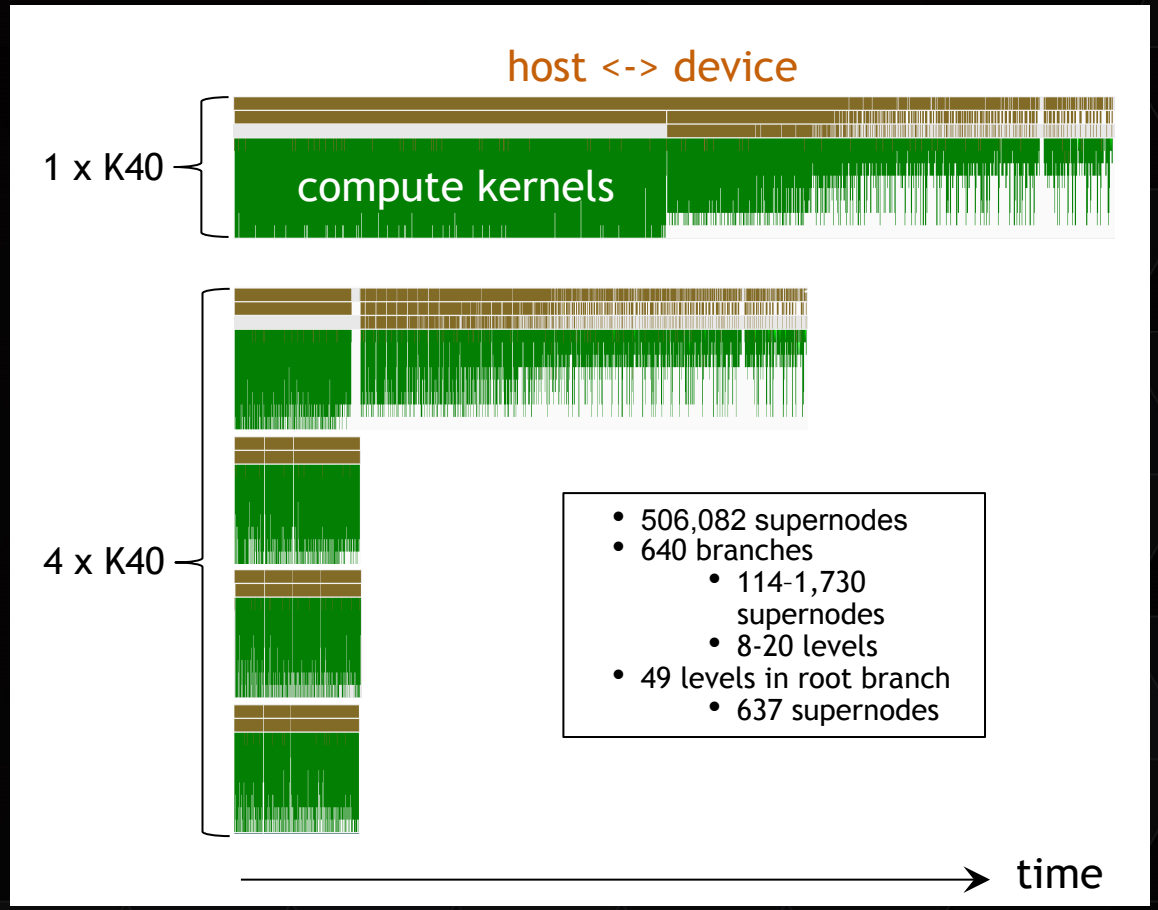
SHELL MODEL PERFORMANCE



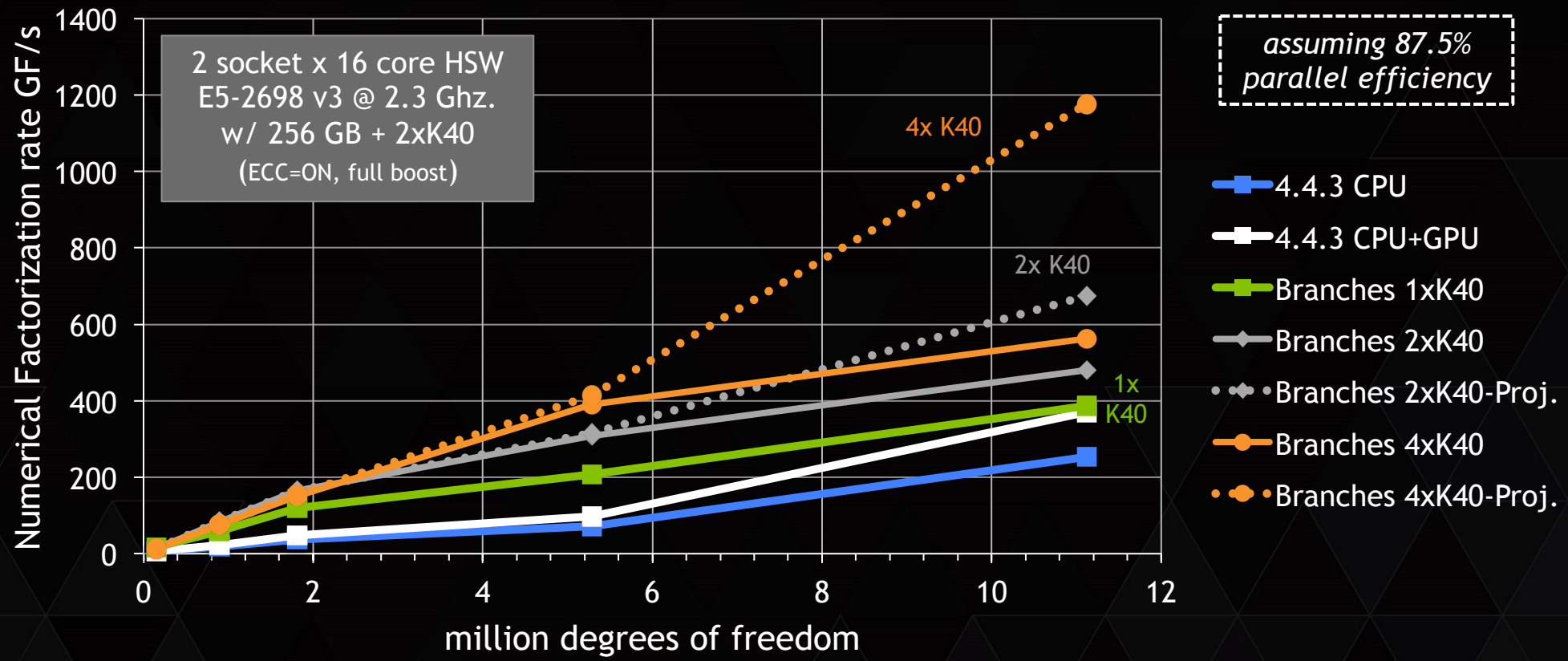
PCB model courtesy of Dr. Serban Georgescu, Fujitsu Laboratories of Europe Ltd

SHELL MODEL PERFORMANCE

- 'Branches' algorithm well-suited for Multi-GPU
- 4 x K40
- Overall 1.5x speedup
- Branches 3.1x speedup
- *We've ported the previous algorithm to multi-GPU*



SHELL MODEL PERFORMANCE



PCB model courtesy of Dr. Serban Georgescu, Fujitsu Laboratories of Europe Ltd

CONCLUSIONS

- Factoring ‘branches’ on GPU avoids PCIe bottleneck
- Batching and streaming permits higher performance on small matrices
- Universally beneficial
- Aspects apply to other factorization methods

- Future
  - Improved performance of batched routines
  - Support hybrid computing
  - Complete multi-GPU support

RELATED WORK

- S5232 - GPU Acceleration of WSMP (Watson Sparse Matrix Package)
  - Natalia Gimelshein, Anshul Gupta
- S5316 - DAG-Scheduled Linear Algebra Using Template-Based Building Blocks
  - Jonathan Hogg
- S5476 - Energy Efficient, High-Performance Solvers through Small Dense Matrix Computations on GPUs
  - Azzam Haidar, Stanimire Tomov
- S5424 - Exploiting Multiple GPUs in Sparse QR: Regular Numerics with Irregular Data Movement
  - Tim Davis
- **S5237 - Jacobi-Davidson Eigensolver in Cusolver Library**
  - Lung-Sheng Chien

GPU TECHNOLOGY CONFERENCE

THANK YOU

JOIN THE CONVERSATION

#GTC15