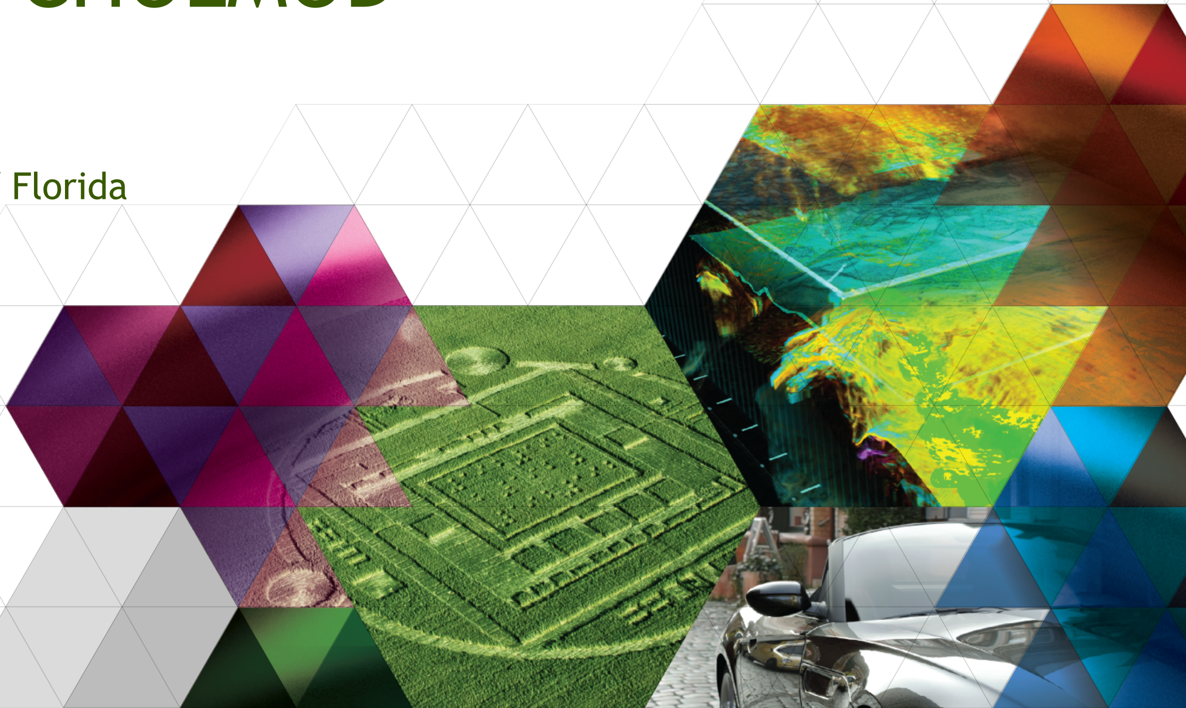
GPU TECHNOLOGY CONFERENCE

GPU ACCELERATION OF SPARSE MATRIX FACTORIZATION IN CHOLMOD

STEVE RENNICH — Nvidia

TIM DAVIS — University of Florida

PHILIPPE VANDERMERSCH — Nvidia



PROBLEM / OBJECTIVE

- Sparse Direct Solvers can be a challenge to accelerate using GPUs
- Tim Davis has been working with NVIDIA to resolve this
  - Describe techniques used
  - Show performance achieved
  - Work in progress
- Would like to suggest that GPUs can be quite good for accelerating sparse direct solves
  - Many optimizations remain

SPECIFIC WORK

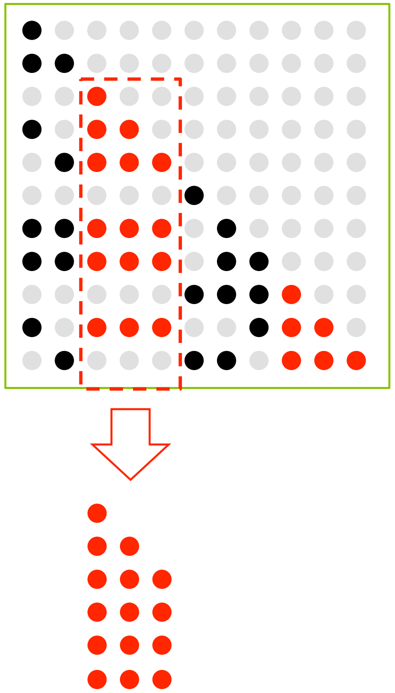
- Cholesky factorization
  - Symmetric Positive Definite (SPD) matrices
- Numerical Factorization
  - Largest component
- CHOLMOD (part of SuiteSparse)
  - High performance
  - Well known
  - Accessible
  - GPU acceleration since v4.0.0

OUTLINE

- Supernodal Cholesky Method
  - Left-Looking Sparse Direct Factorization
- Results
  - CHOLMOD **4.3.0 GPU** *vs.* **4.2.1 GPU** *vs.* **4.3.0 CPU**
- Acceleration Techniques
- Issues / future work

SPARSE DIRECT SOLVERS

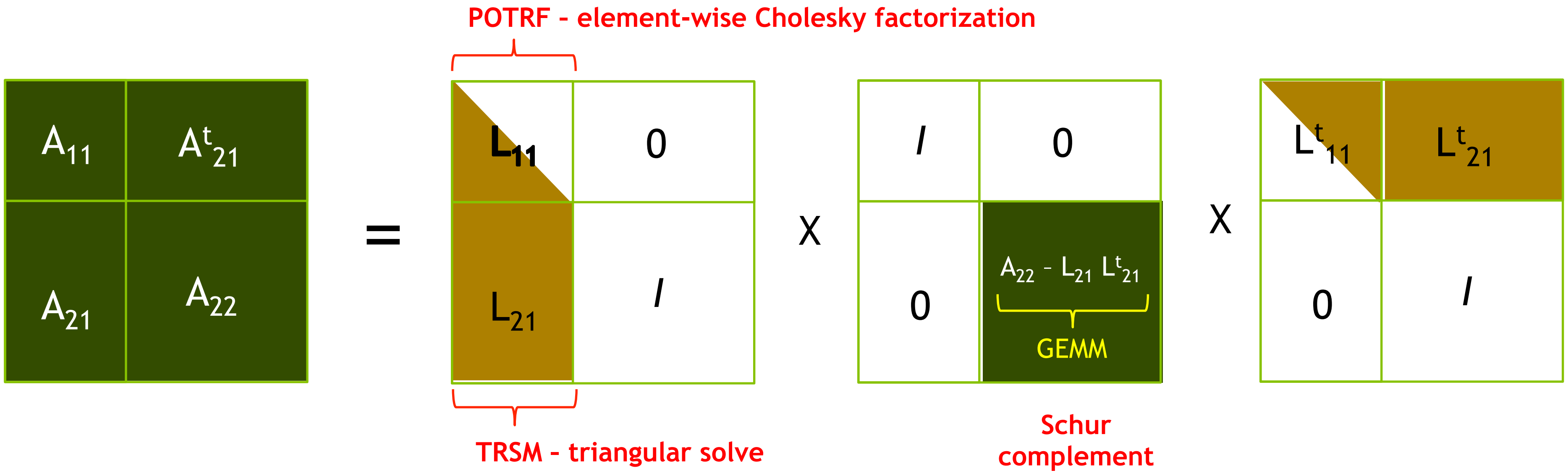
- Many flavors
  - Supernodal / Multi-frontal
  - Left / right looking
- Supernodes
  - collections of similar columns
  - provide opportunity for dense matrix math
  - grow with mesh size due to ‘fill’
  - The larger the model, the larger the supernodes
- Supernodes for solids grow faster than supernodes for shells



DENSE BLOCK CHOLESKY

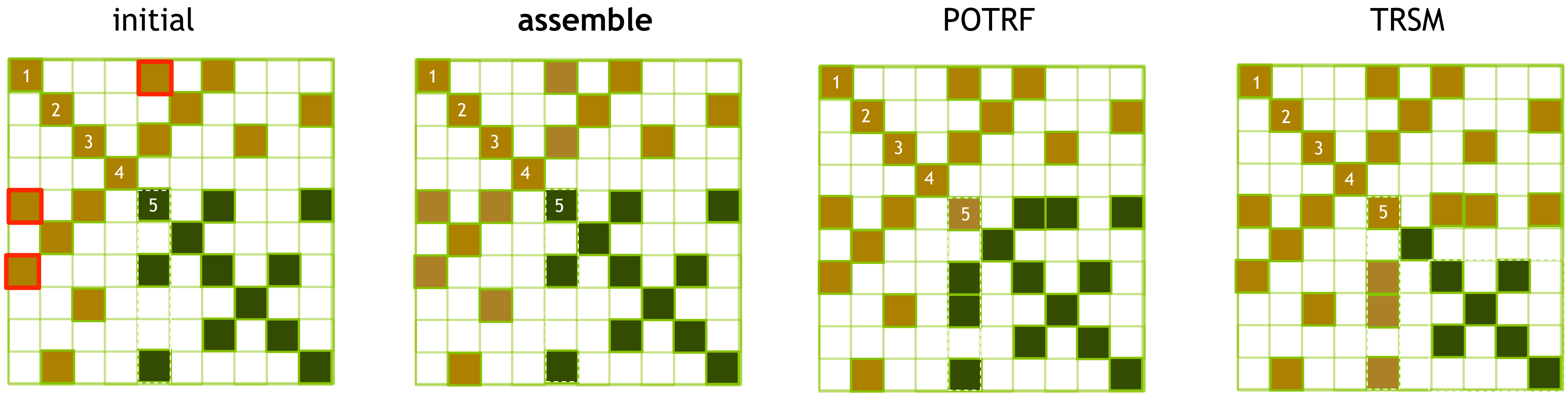
- Basis for sparse direct algorithms
  - **Emphasizes dense math**
  - Dominated by computation of Schur complement

$L_{11} L^t_{11} = A_{11}$	**POTRF**
$L_{11} L^t_{21} = A^t_{21}$	**TRSM**
$A^*_{22} = A_{22} - L_{21} L^t_{21}$	**GEMM**



SUPERNODAL SPARSE CHOLESKY

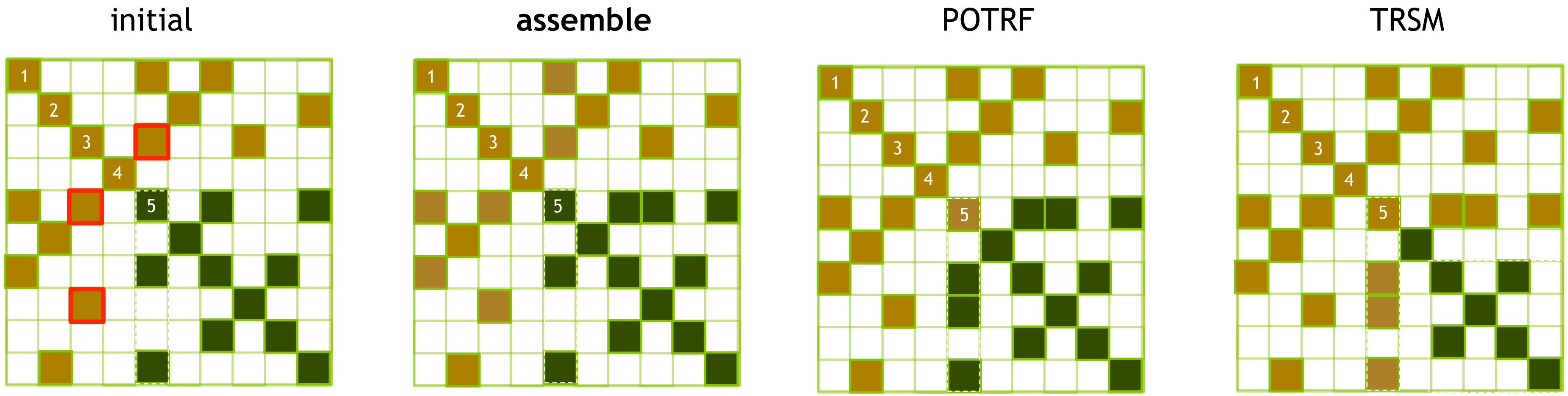
- ‘Left looking’ proceeds left to right by supernodes



Assemble Schur complement from supernode 1 (SYRK / GEMM)

SUPERNODAL SPARSE CHOLESKY

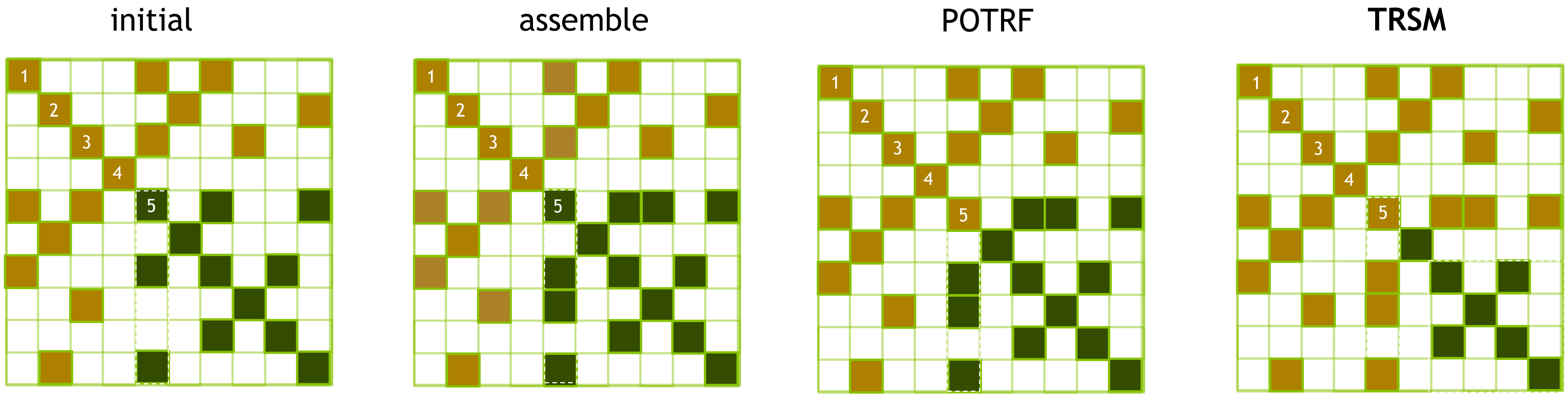
- 'Left looking' proceeds left to right by supernodes



Assemble Schur complement from supernode 3 (SYRK / GEMM)

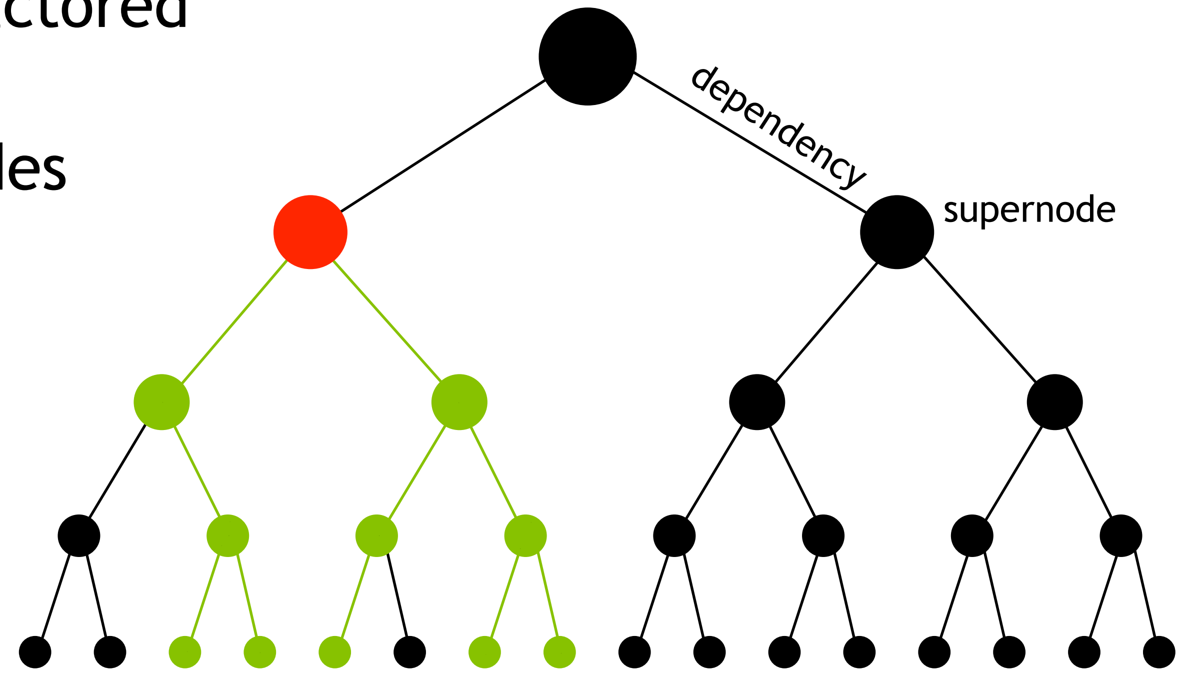
SUPERNODAL SPARSE CHOLESKY

- 'Left looking' proceeds left to right by supernodes



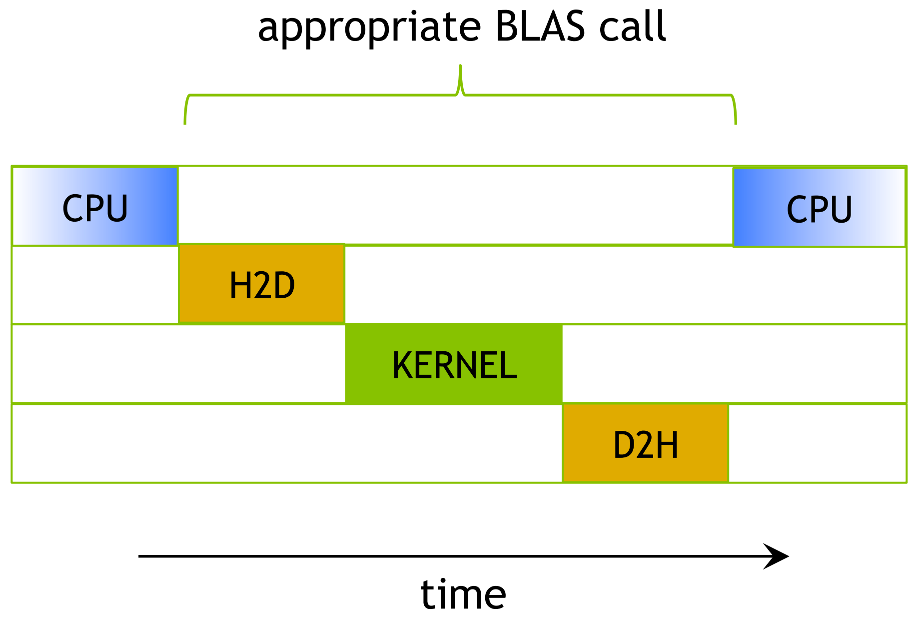
ELIMINATION TREE

- DAG : determines order in which supernodes can be factored
- Descendant supernodes referenced multiple times



SIMPLE ACCELERATION APPROACH

- Large dense math to GPU
  - SYRK, GEMM, TRSM
  - Serial
- Constrained by
  - Serial processing
  - Small supernodes
  - Strong dependence on PCIe bandwidth
  - No hybrid processing
  - Host memory bandwidth



RESULTS - TEST MATRICES

100 SPD matrices from **Florida Sparse Matrix Collection**

Matrix	rows/cols	nnz A	nnz/row	nnz L	fill ratio
nd24k	72,000	14,393,817	199.91	5.12E+08	35.54
Fault_639	638,802	14,626,683	22.90	3.27E+09	223.77
Emilia_923	**923,136**	**20,964,171**	**22.71**	**5.60E+09**	**267.28**
boneS10	**914,898**	**28,191,660**	**30.81**	**3.69E+08**	**13.08**
inline_1	503,712	18,660,027	37.05	2.21E+08	11.82
ldoor	952,203	23,737,339	24.93	1.53E+08	6.46
bone010	986,703	36,326,514	36.82	2.26E+09	62.10
Hook_1498	1,498,023	31,207,734	20.83	3.12E+09	99.92
Geo_1438	1,437,960	32,297,325	22.46	6.68E+09	206.89
Serena	1,391,349	32,961,525	23.69	7.94E+09	240.89
audikw_1	**943,695**	**39,297,771**	**41.64**	**2.33E+09**	**59.20**
Flan_1565	1,564,794	59,485,419	38.01	3.60E+09	60.45

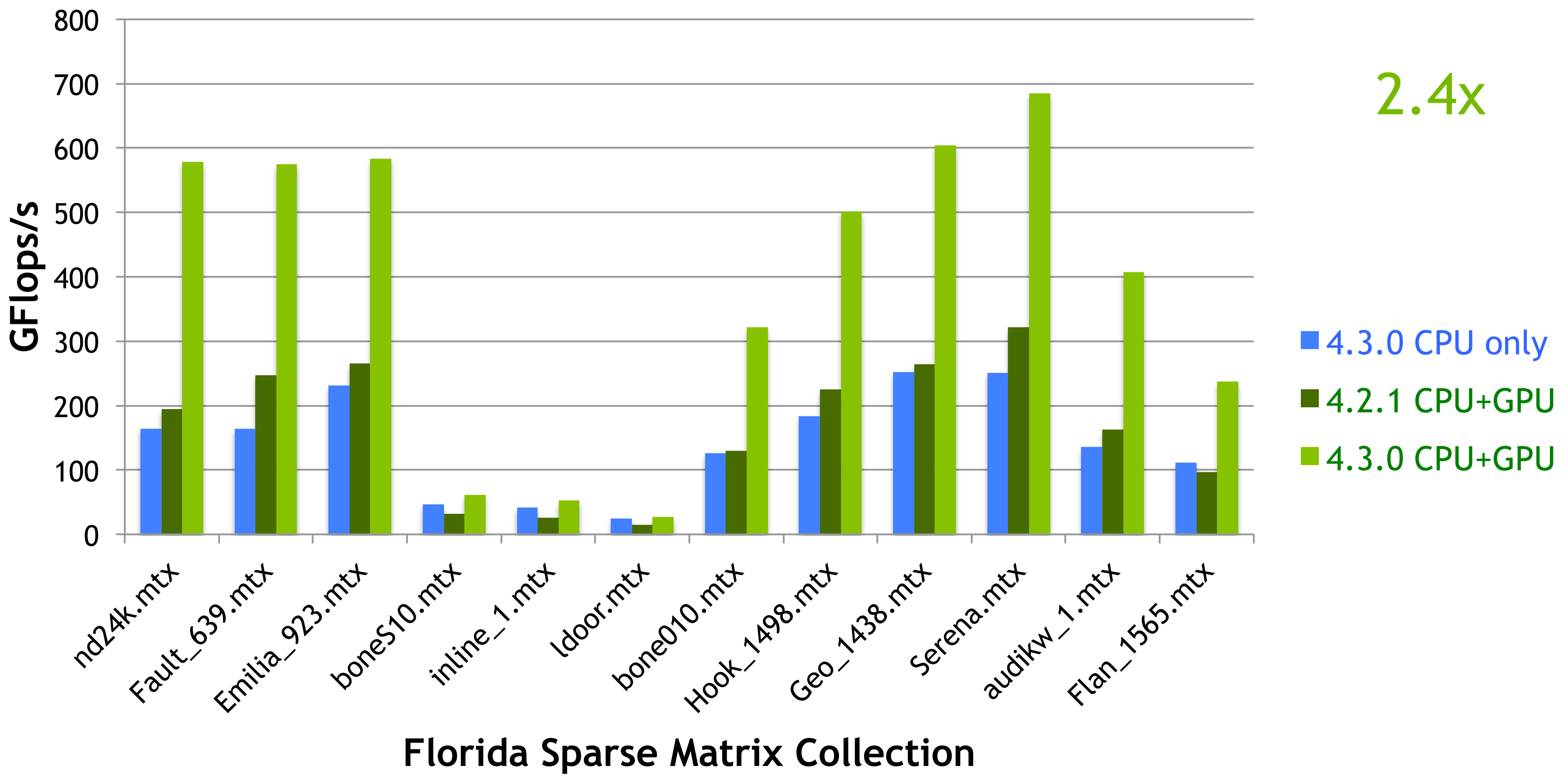
http://www.cise.ufl.edu/research/sparse/matrices/

RESULTS – SYSTEM USED

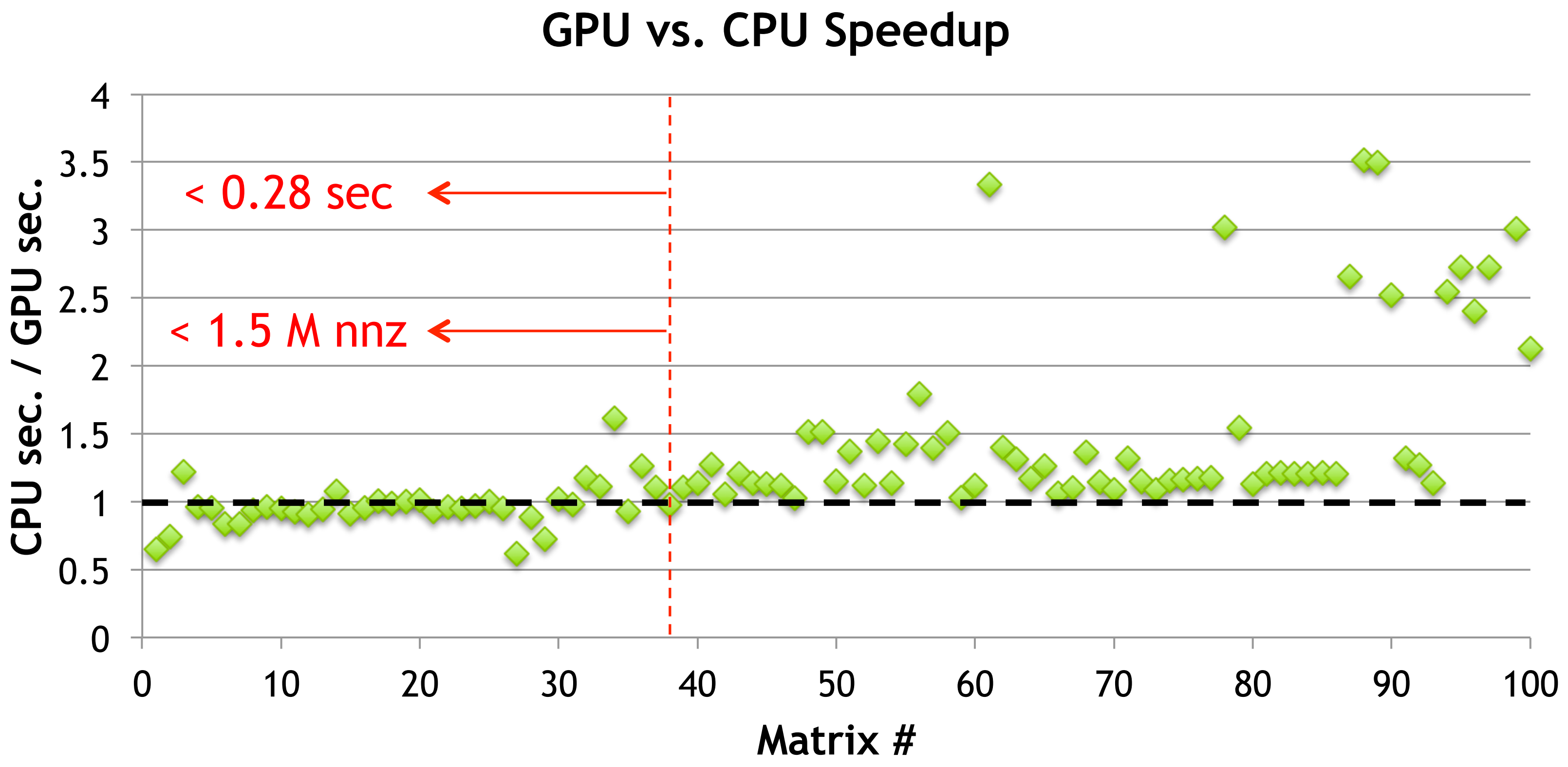
- CHOLMOD (SuiteSparse version 4.3.0)
  - Metis 4.0
- Dual-socket Ivy-Bridge Xeon @ 3.0 Ghz
  - 20 cores total, PCIe gen3, E5-2690 v2
- Tesla K40
  - boost clocks (3004, 875), ECC=OFF, Using 3GB of GPU memory
- Intel Composer XE 2013
  - compiler & MKL

GPU TECHNOLOGY CONFERENCE

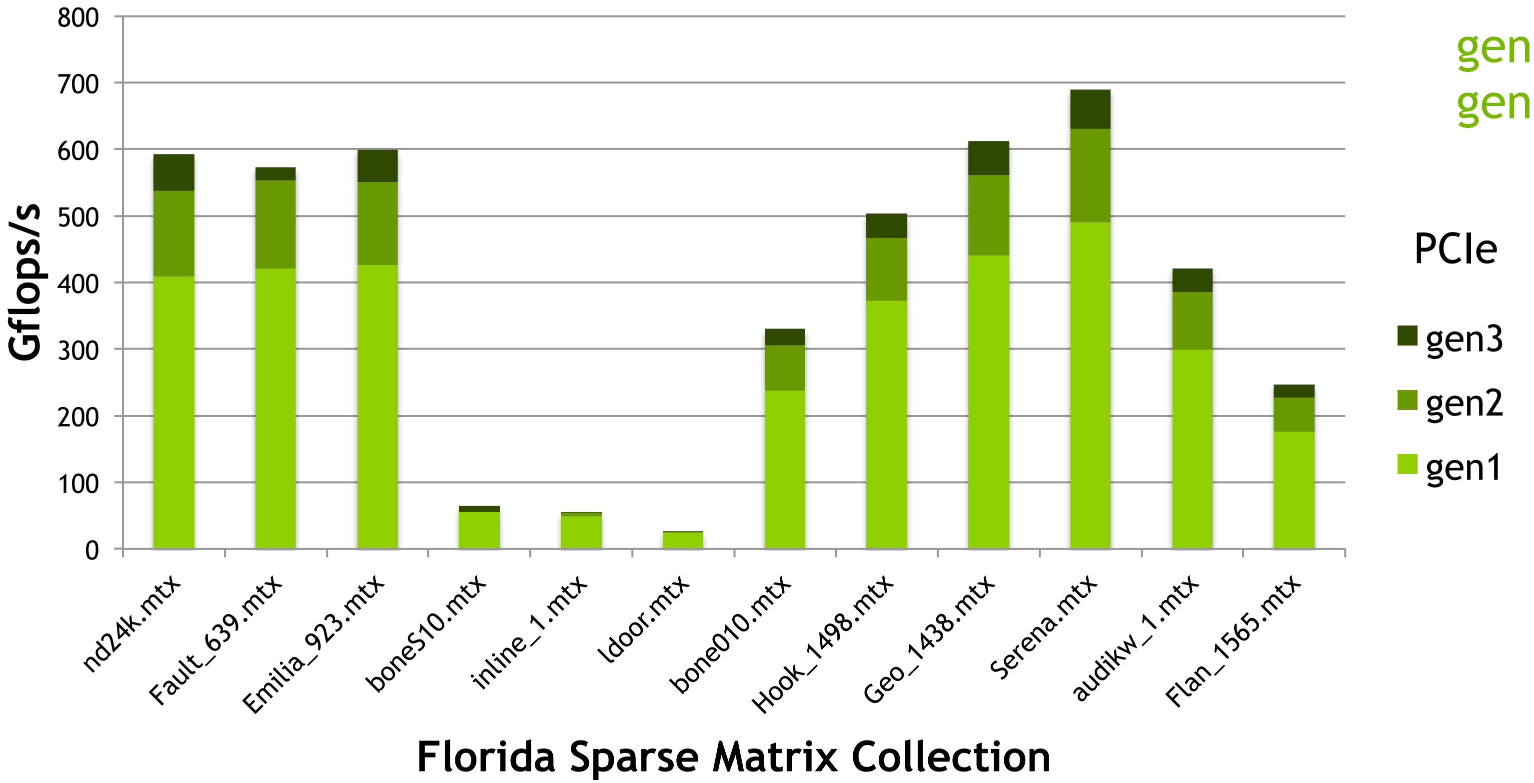
RESULTS - SINGLE K40



RESULTS - SPEEDUP VS. CPU



RESULTS – PCIE DEPENDENCE

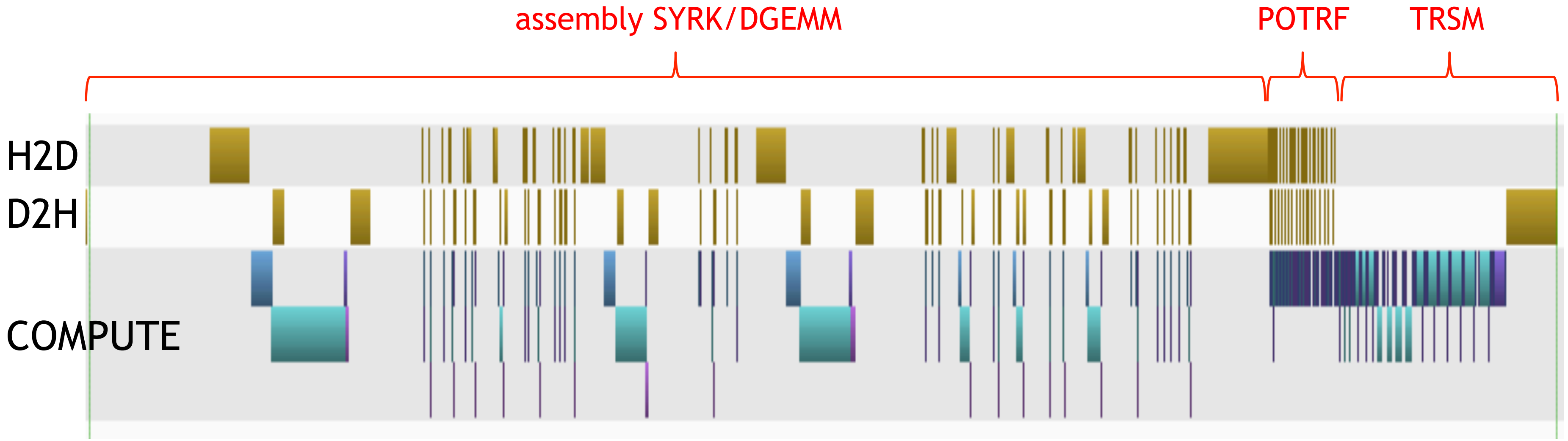


gen2 = 92% gen3
gen1 = 75% gen3

6 core SB i7 @ 3.2 GHz + K40

CHOLMOD-4.2.1 WITH K40

- nvvp
- Second-to-last supernode



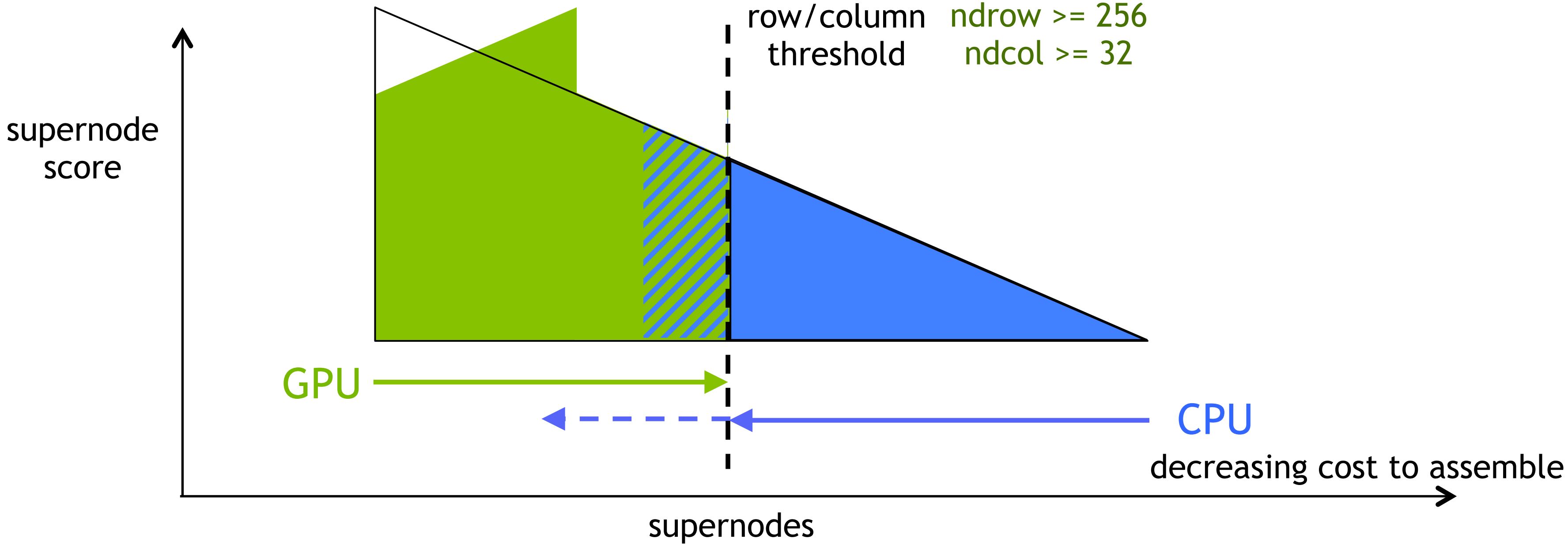
OPTIMIZATION TECHNIQUES

- Reorder Descendants
  - Hide PCIe communication behind computation
- Assemble supernodes on GPU
  - Reduce PCIe & host memory traffic
- Hybrid computing
  - Achieved using fixed GPU and host pinned buffers
- Block factorization of diagonal blocks and lower panel

GPU TECHNOLOGY CONFERENCE

REORDERING DESCENDANTS

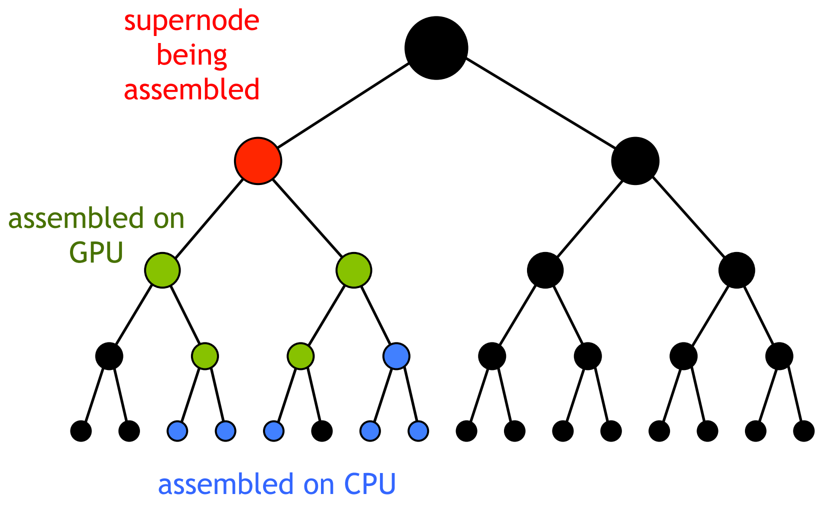
- For each supernode
  - Descendant supernodes are ‘scored’ by their area (ncol*nrow)
- Supernodes are sorted by score to maximize kernel/memcpy overlap



ASSEMBLE SUPERNODES ON GPU

- Large descendants assembled on GPU
  - 2 streams / double buffered
- Small descendants assembled on CPU
  - hybrid computing
- Assembled supernode is sum of the CPU and GPU components

$$A^* = A - \sum_{small} L_{21} L^t_{21} - \sum_{large} L_{21} L^t_{21}$$



SUPERNODE BUFFERS

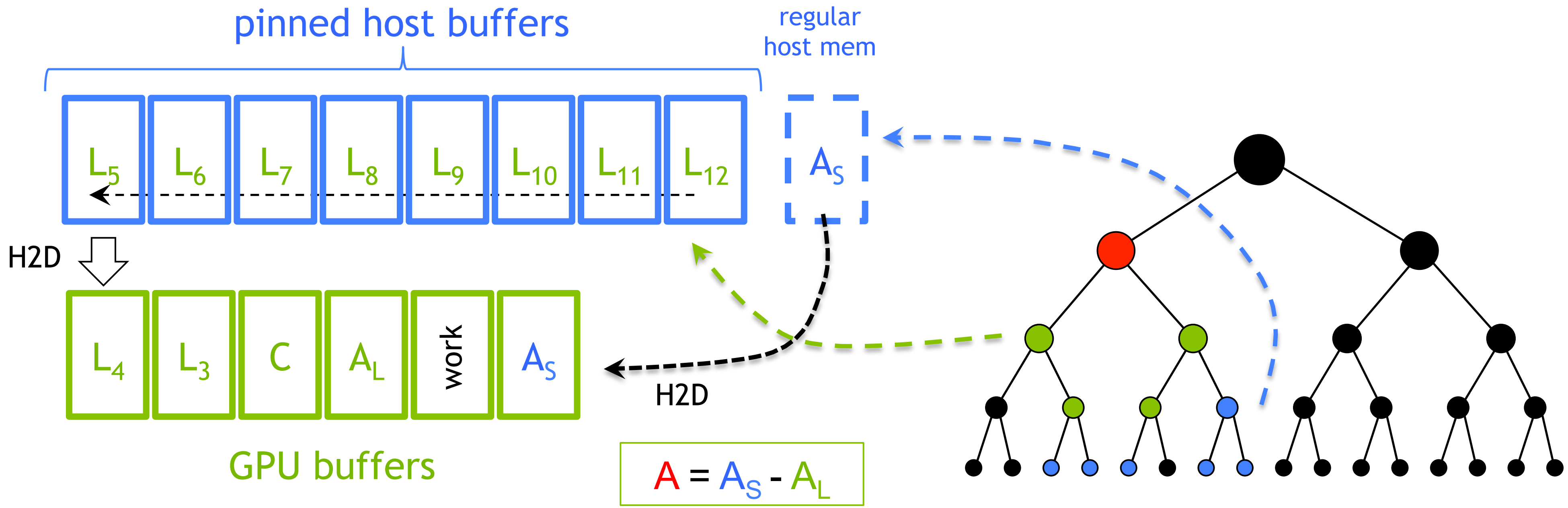
- Single allocation of CPU & GPU memory
  - supports all GPU computing
  - High perf. / asynchronous PCIe requires pinned host memory
  - Allocating pinned host memory is slow (~1.4 sec. for 4 GB)
    ! This time is not included in benchmarks presented here !
- All buffers are reused
  - Independent of matrix being factored
- Symbolic Factorization
  - **LIMIT supernode size such that they all fit in the pre-defined buffers**

SUPERNODE BUFFERS

- 6 Device Buffers (0.5 GB each)
  - 2 to hold incoming descendant supernodes: L_{21}
  - 1 to hold Schur complement update: $C = L_{21} L^t_{21}$
  - 2 to hold partial assemblies (1 from CPU): A -= C
  - 1 for everything else: scatter maps
- 8 Host buffers (0.5 GB each)
  - Hold descendant supernodes ready for async transfer to GPU
  - CPU fills buffers and issues/queues GPU operations

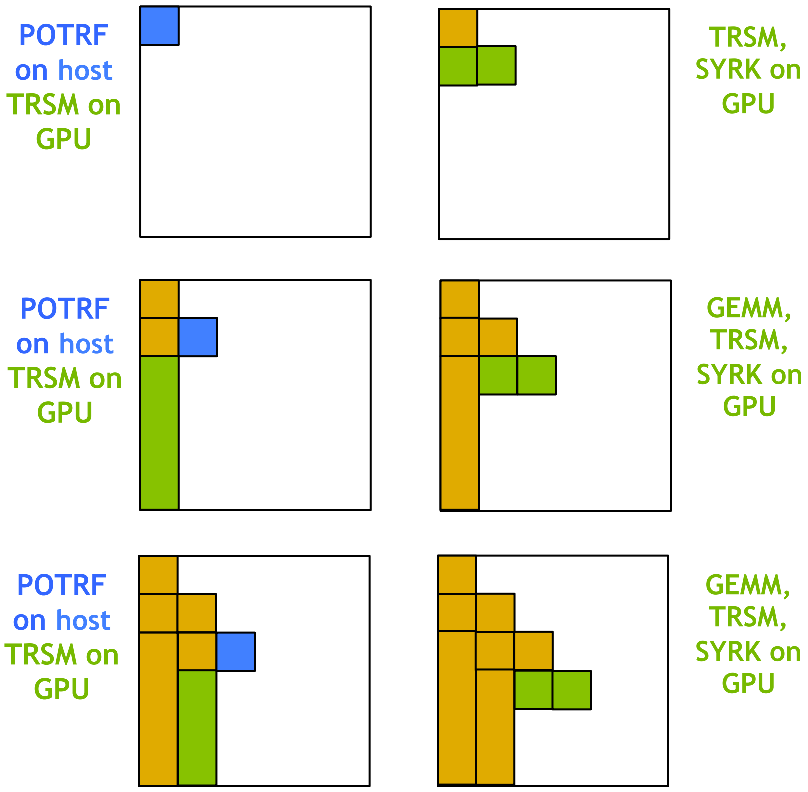
SUPERNODE BUFFERS

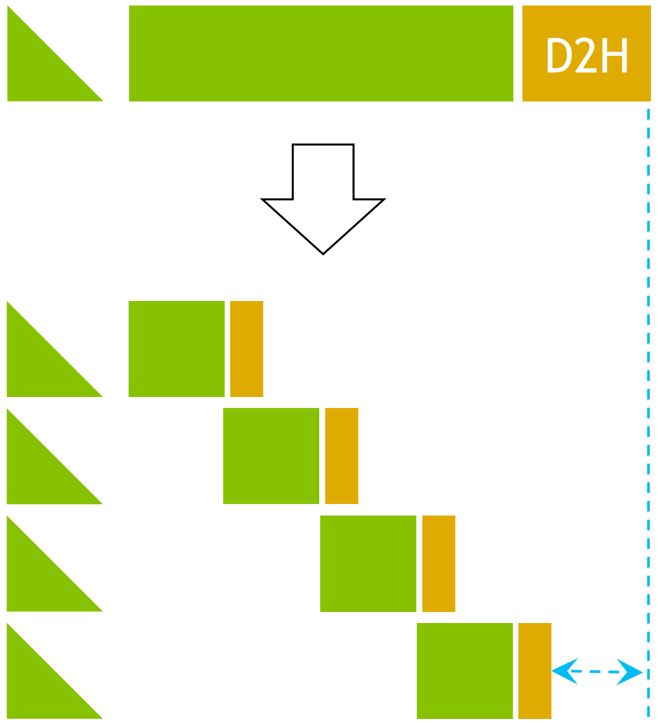
- While a host buffer is available
  - copy largest remaining descendant and queue factorization commands on GPU
- CPU assembles 3 smallest remaining descendants



BLOCKED POTRF AND TRSM

- POTRF – element Cholesky
- TRSM – triangular solve

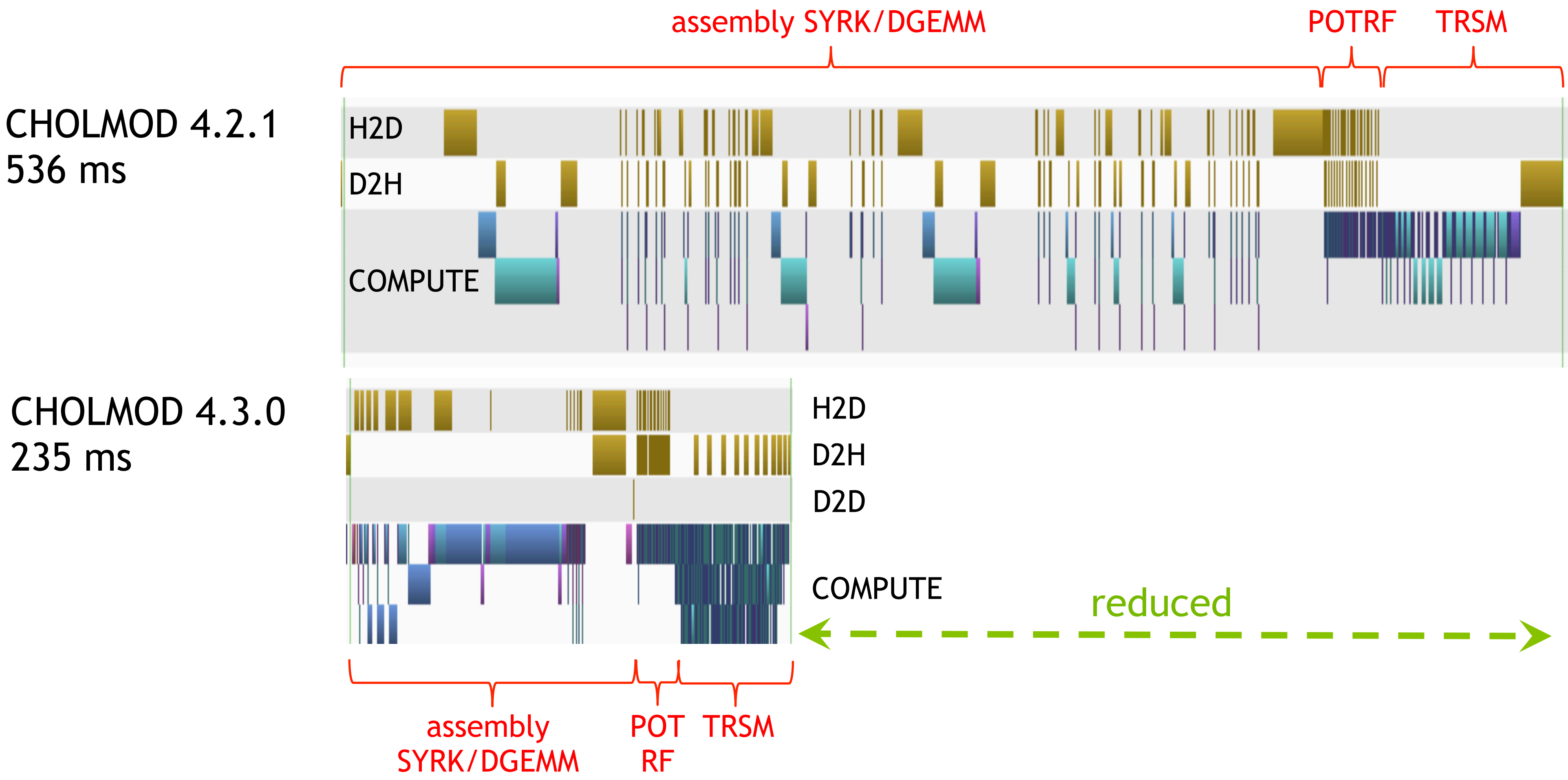




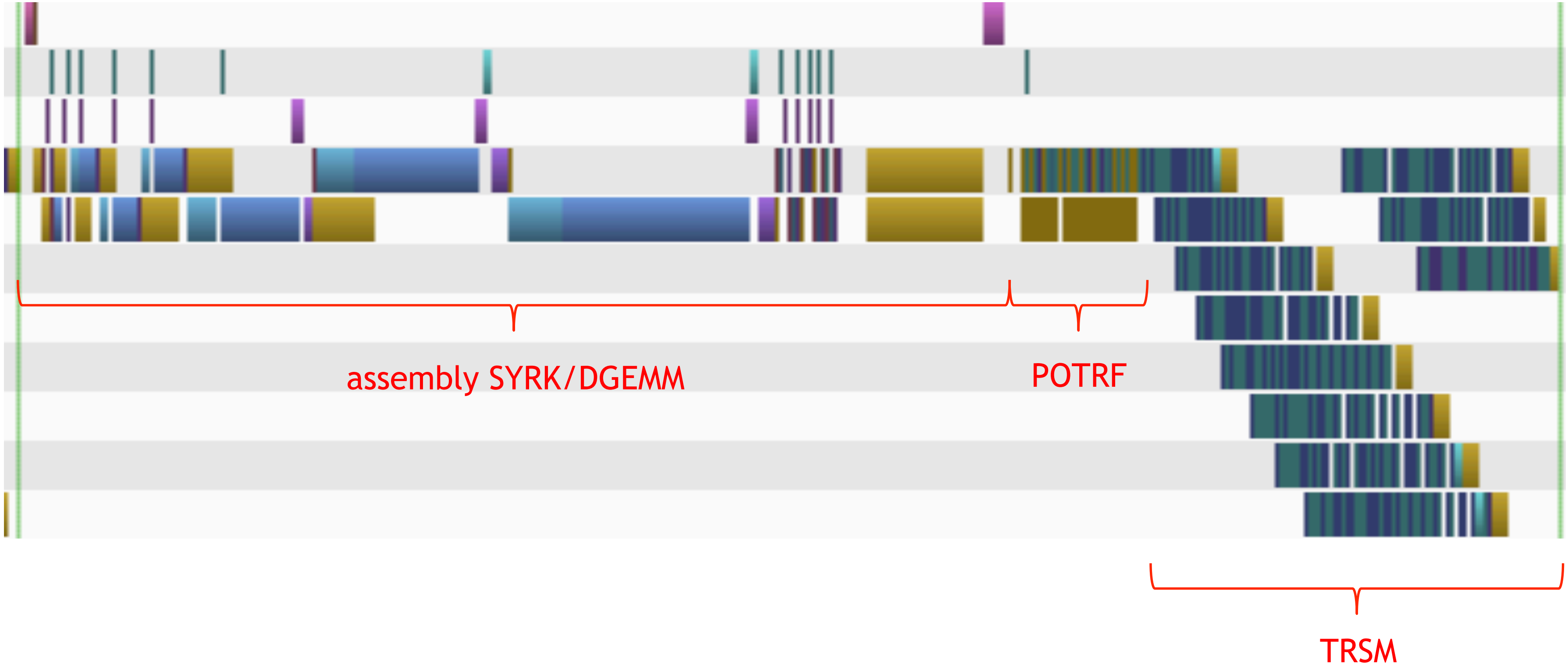
ONLY 3 CUSTOM KERNELS

- Create map
  - Map rows of current supernode
- Create relative map
  - Map rows of current descendant to current supernode
- Scatter update
  - Use maps to scatter descendants contribution to the partial assembly
- Very simple, very fast

CHOLMOD 4.2.1 VS. 4.3.0



CHOLMOD V4.3.0



USING GPU ACCELERATION IN CHOLMOD

- Programmatically

```
cholmod_start ( Common );
Common->useGPU = 1;
Common->maxGpuMemBytes = 3000000000;
```

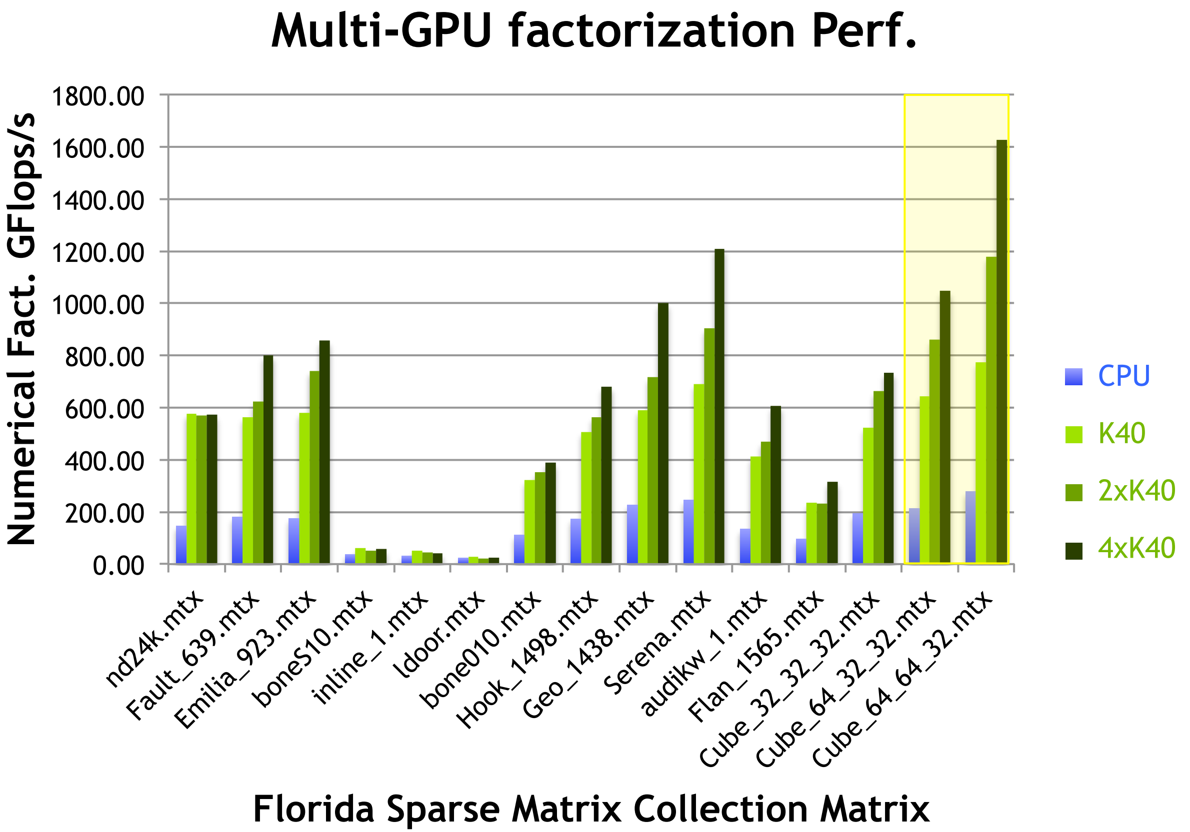
1	= Use GPU
0	= Don't use GPU
-1	= query environment (default)

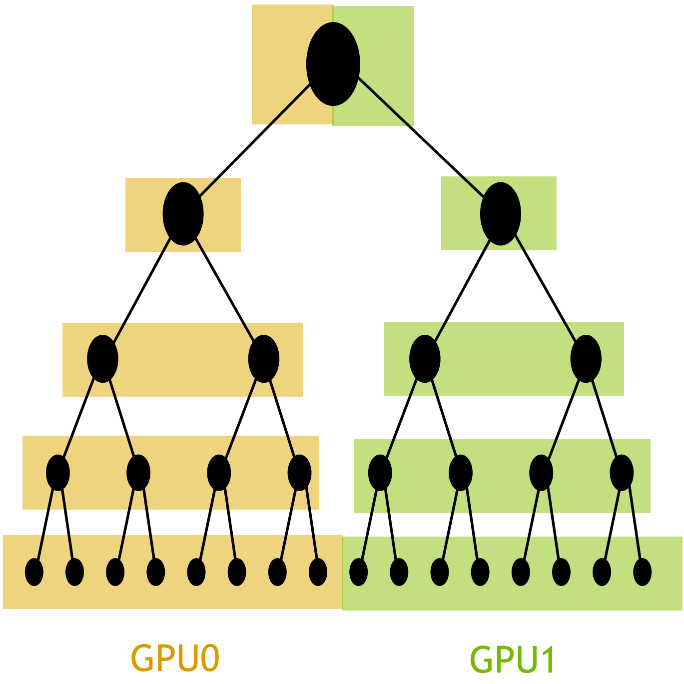
- Environmentally

```
>export CHOLMOD_USE_GPU = 1
>export CHOLMOD_GPU_MEM_BYTES = 3000000000
```

GPU TECHNOLOGY CONFERENCE

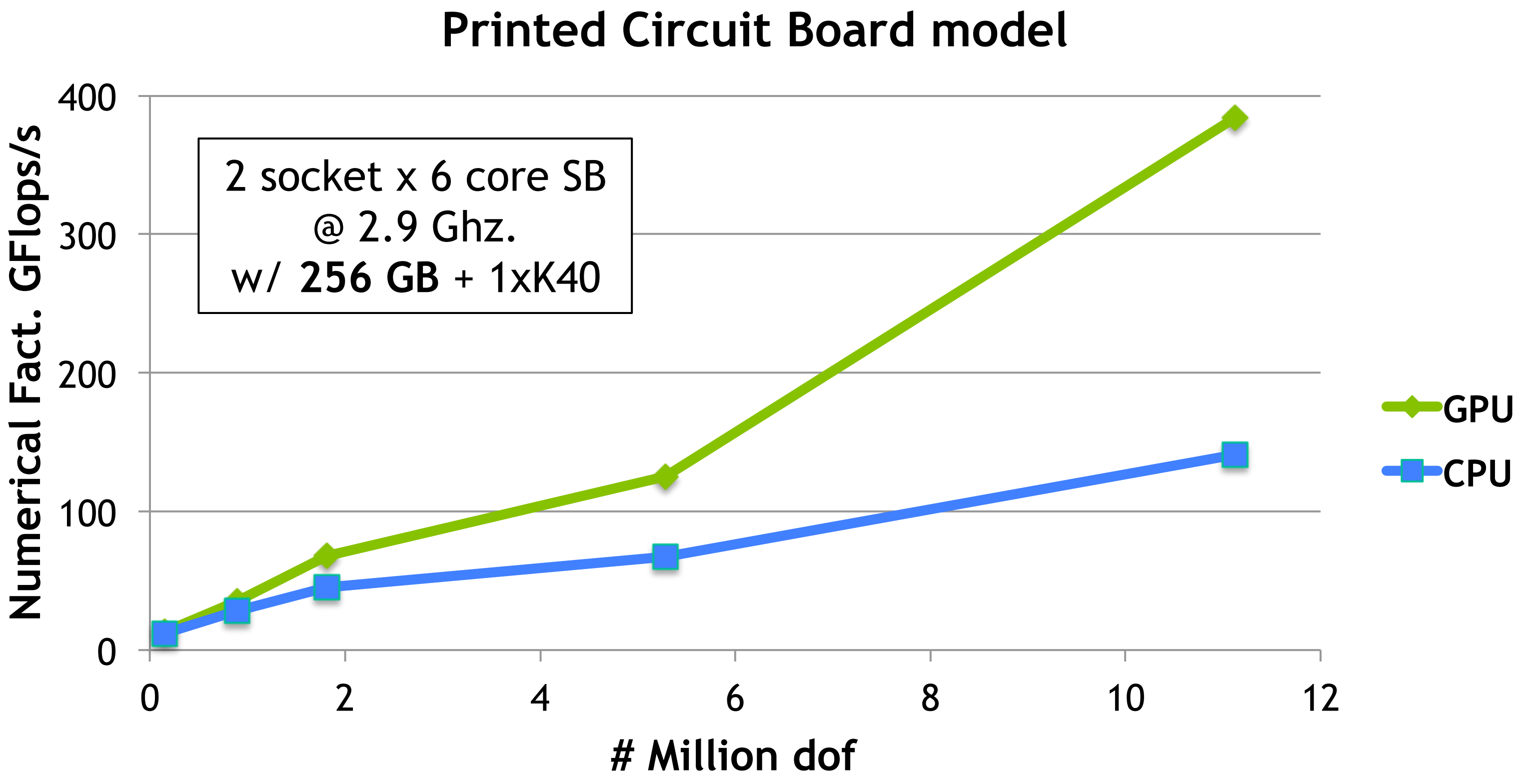
FUTURE – LEVERAGE MULTI-GPU





thanks to Wajih Boukaram, KAUST

SHELL MODEL PERFORMANCE

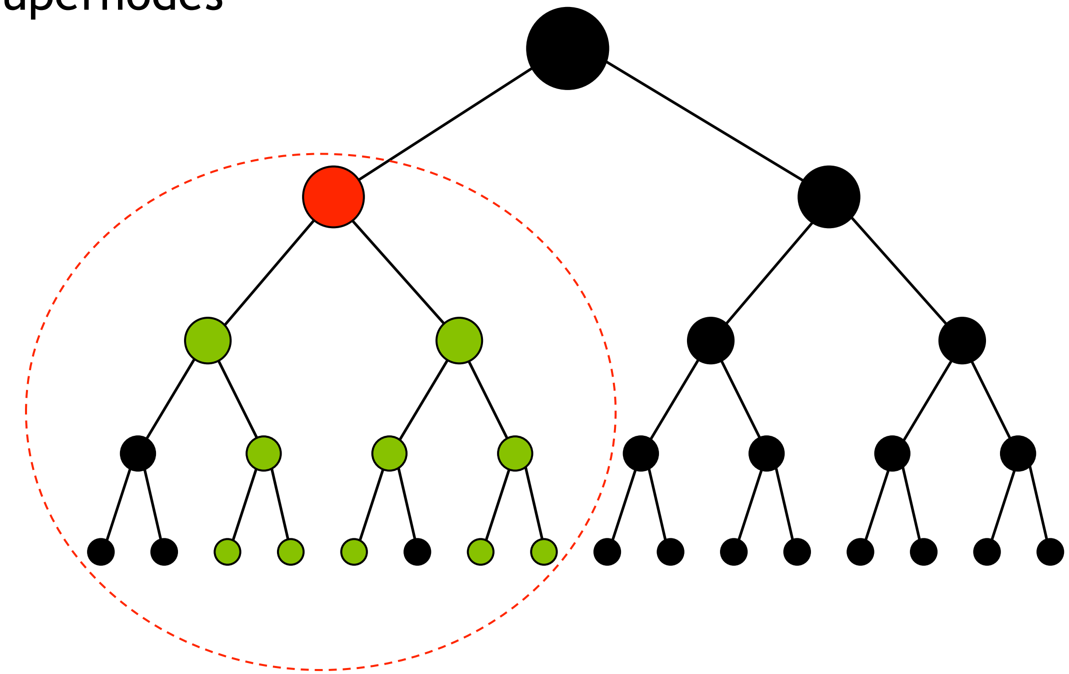


PCB model courtesy of Dr. Serban Georgescu, Fujitsu Laboratories of Europe Ltd

FUTURE – 'BRANCHES ON GPU'

- Move branches of the elimination tree to the GPU
  - Requires POTRF on GPU
  - Eliminates substantial PCIe overhead
  - Accelerates small supernodes

matrix data for these nodes is transferred to GPU and entire factor is computed on GPU



THANK YOU

- Try it out!
  - Download SuiteSparse 4.3.0 w/ CHOLMOD 3.0.0
  - See exactly what was done and how it performs

http://www.cise.ufl.edu/research/sparse/SuiteSparse/