USC Viterbi
School of Engineering

Accelerating MCAE with GPUs

Information Sciences Institute

Information Sciences Institute
USC

15 Sept 2010

Bob Lucas, Gene Wagenbreth, Dan Davis, Roger Grimes
{rflucas,genew,ddavis}@isi.edu and grimes@lstc.com

Outline

MCAE Sparse Solver Bottleneck

Review of Multifrontal Method

Adding a GPU

Performance Results

Future Directions

MCAE

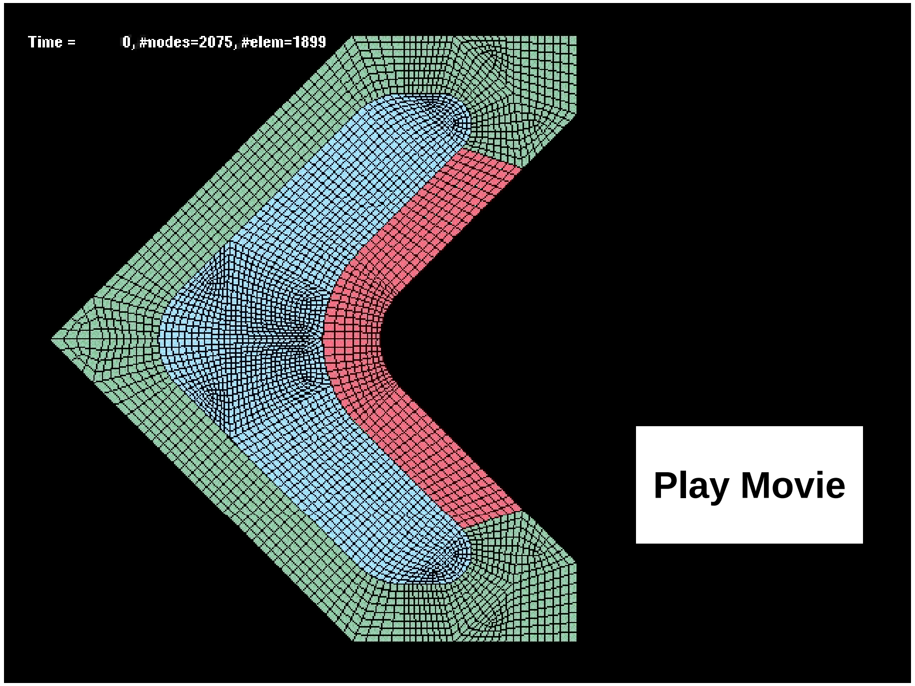
Mechanical Computer Aided Engineering

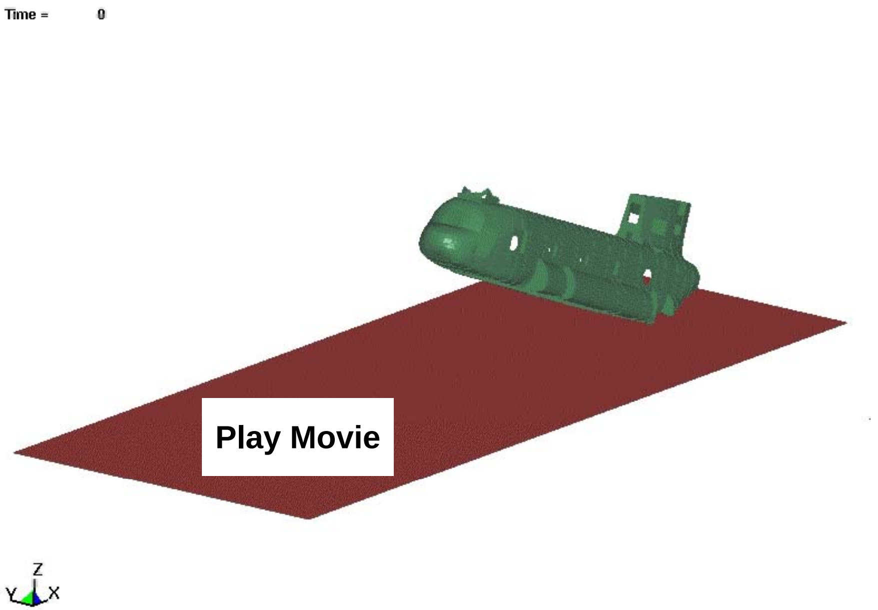
- ISVs ABAQUS, ANSYS, LS-DYNA, & NASTRAN
- GOTS Alegra, ALE3D, CTH, & ParaDYN

Broad range of capabilities

- Static analysis
- Vibration analysis
- Crash analysis

Defense Examples



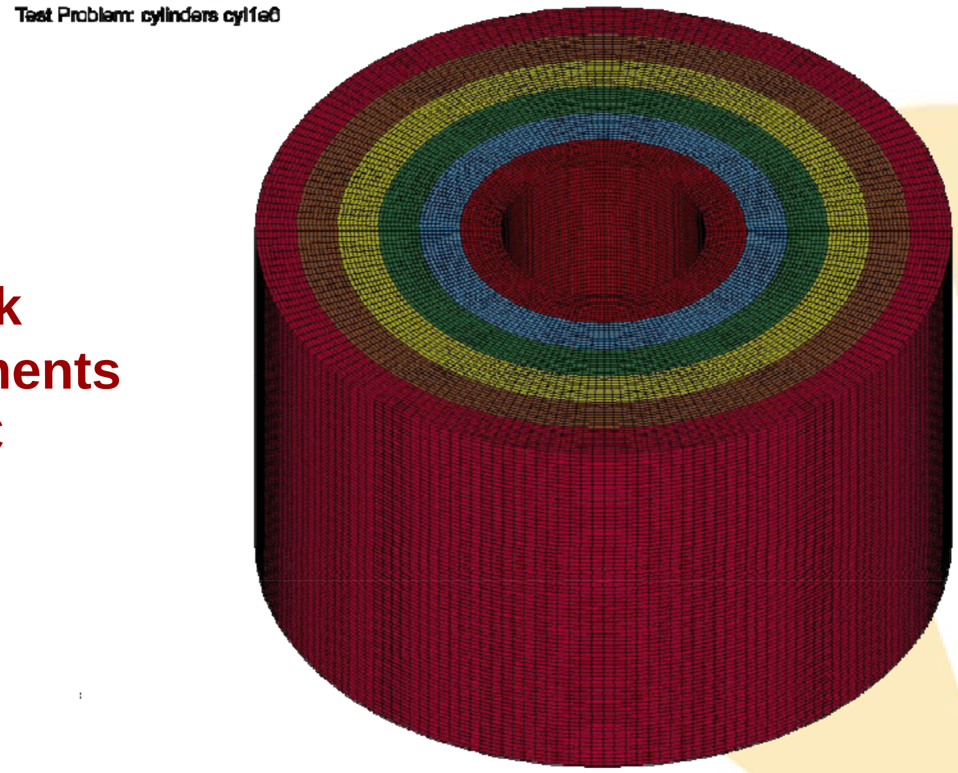


Shaped charge
Courtesy FEA Info & LSTC

CH47 Landing
Courtesy FEA Info & Boeing

Computational Bottleneck

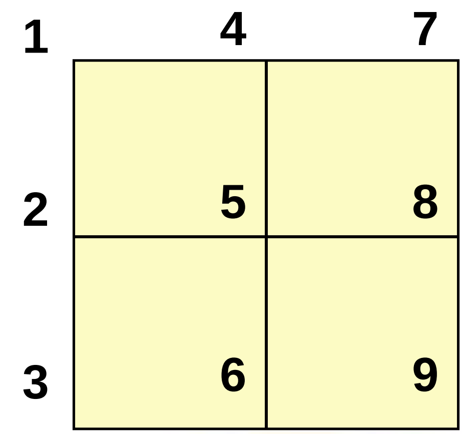
Total time	2057 sec.	
Linear solver	1995 sec.	97%
Factorization	1981 sec.	96%



AWE benchmark
230K 3D Finite Elements
Courtesy LSTC

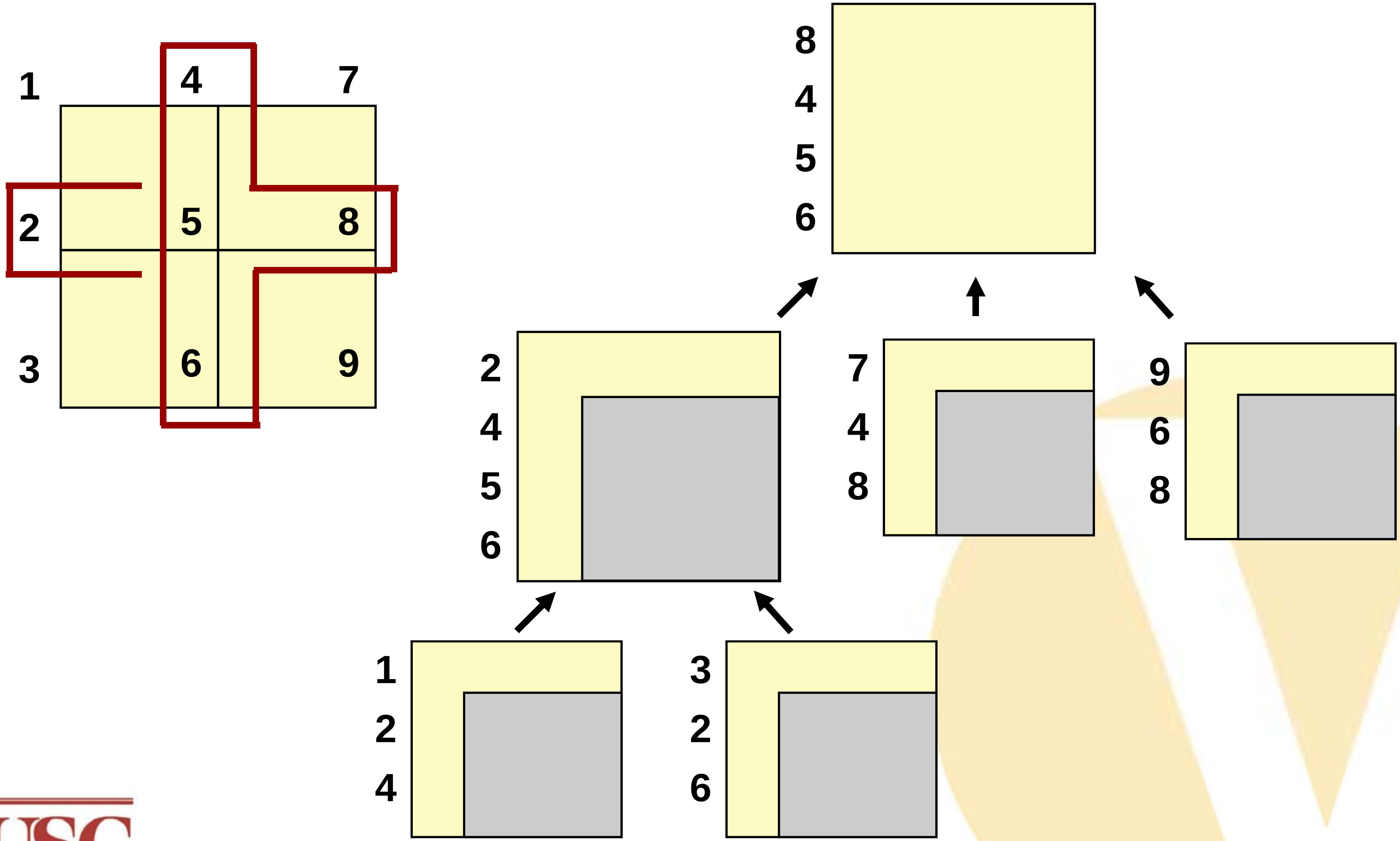
Toy Sparse Matrix


```
      do 4 k = 1, 9
        do 1 i = k + 1, 9
          a(i, k) = a(i,k) / a(k,k)
1       continue
        do 3 j = k + 1, 9
          do 2 i = k + 1, 9
             a(i,j) = a(i,j) -
     1                a(i,k) *
     2                a(k,j)
2         continue
3       continue
4     continue
```




```
1 X X   X
3  XX     X
2 XXX   *X*
7    X XX
9     XX  X
8    XXX*X*
4 X *X *XX*
5   X  XXXX
6  X* X**XX
```

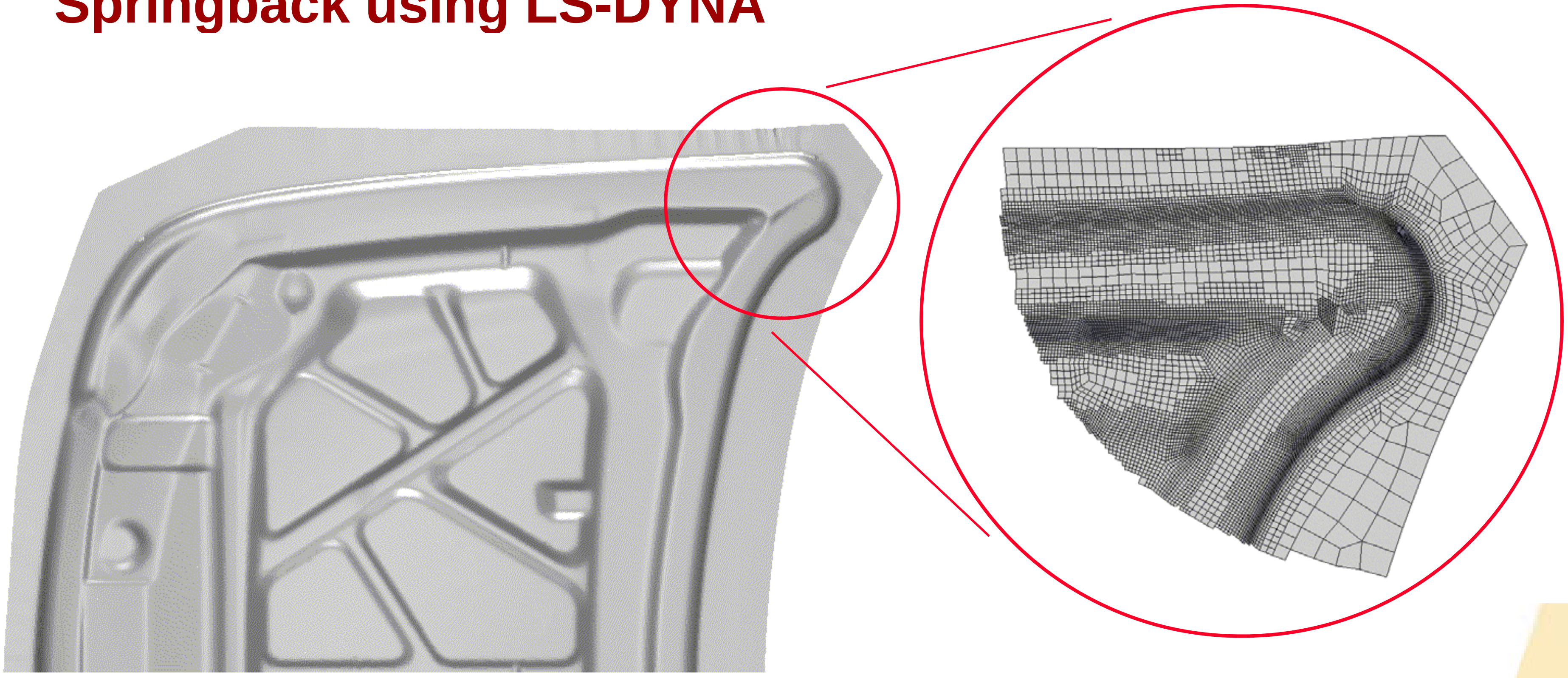
Multifrontal View of the Toy Matrix



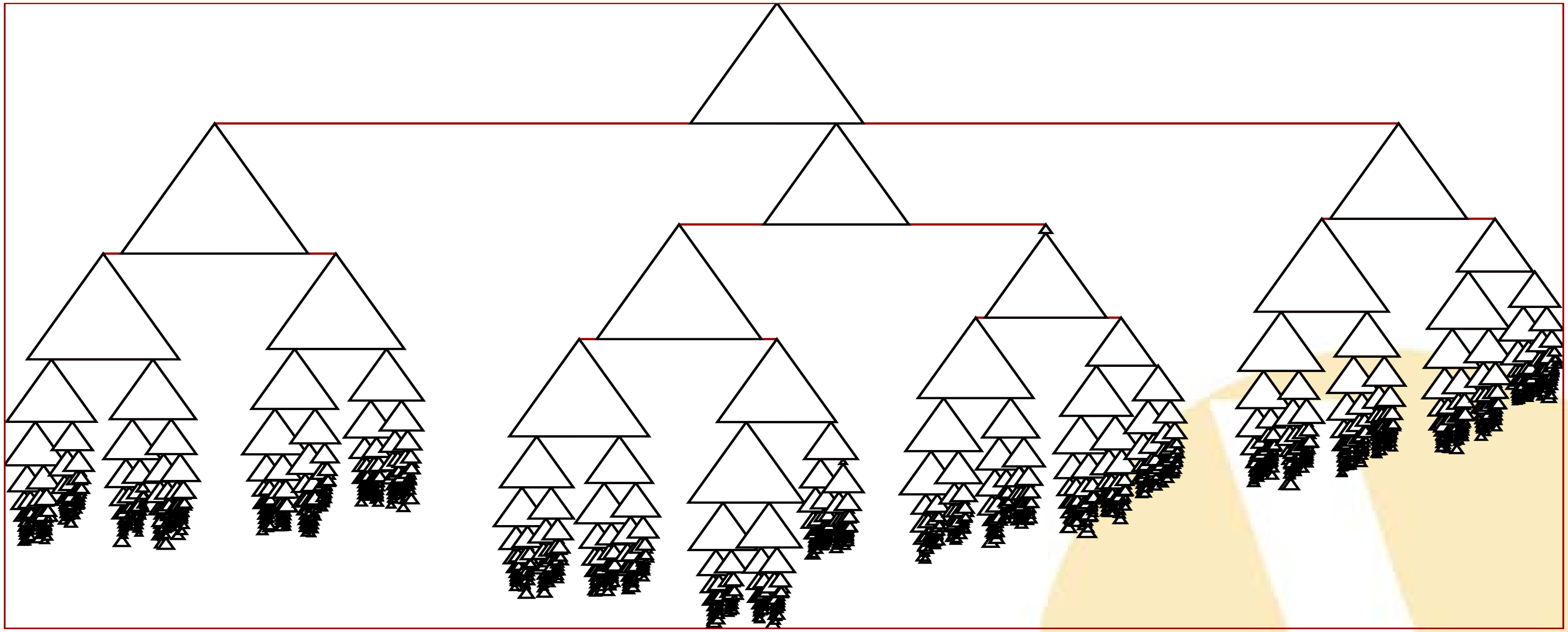
Duff and Reid, ACM TOMS 1983

A Real Problem : "Hood"

Automotive Hood Inner Panel Springback using LS-DYNA



“Hood” Elimination Tree



Each frontal matrix’s triangle scaled by operations required to factor it.

Two Sources of Concurrency

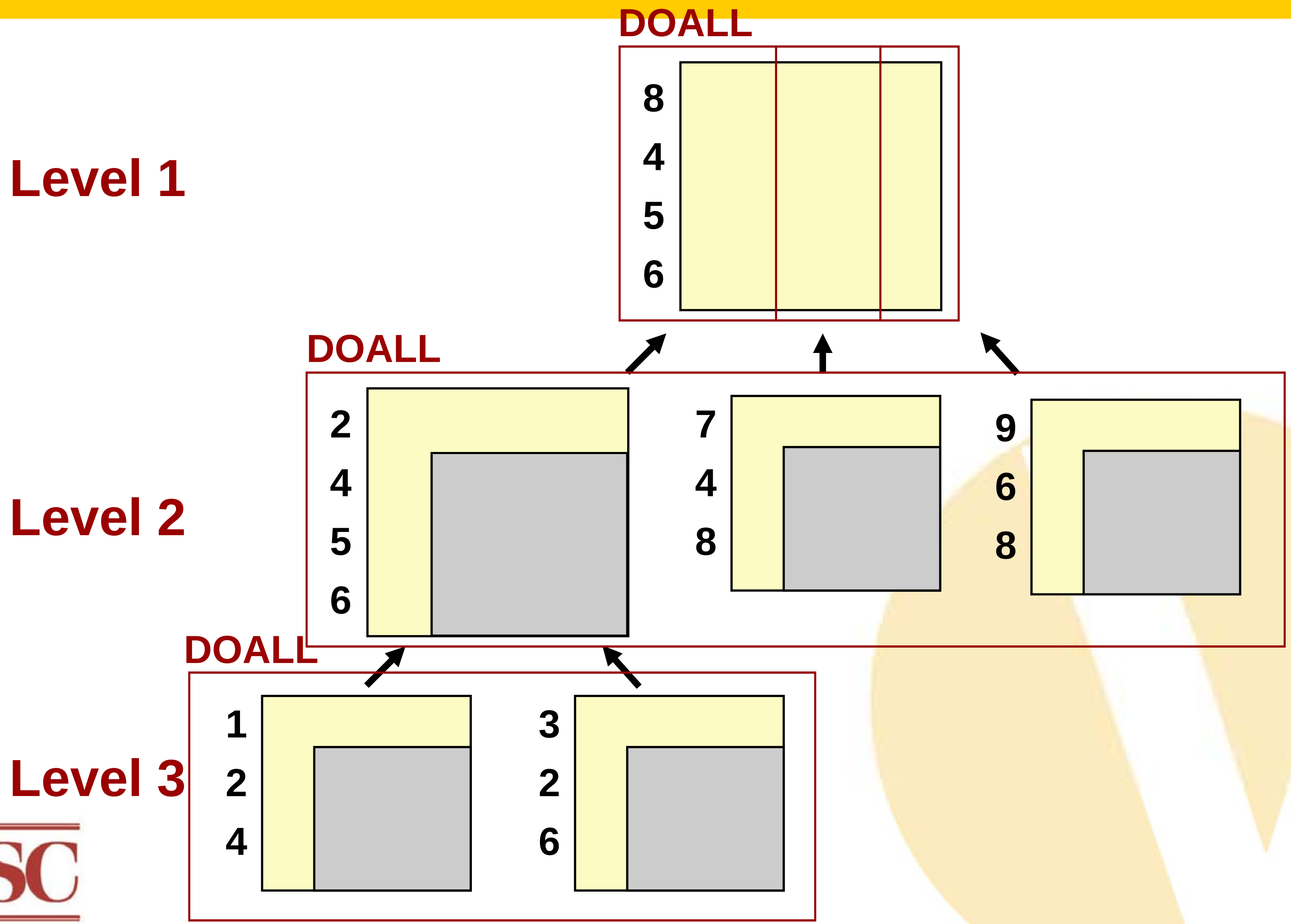
Concurrency within frontal matrices

- Small P => column wrap
- Large P => 2D (ala LINPACK benchmark)

Concurrency across elimination tree

- Frontal matrices only dependent on children
- “Subtree – subcube” typically used
- Limits communication

Shared Memory Concurrency



Why Explore GPUs?

USC Viterbi
School of Engineering

Information Sciences Institute

Ubiquitous, cheap, high performance!

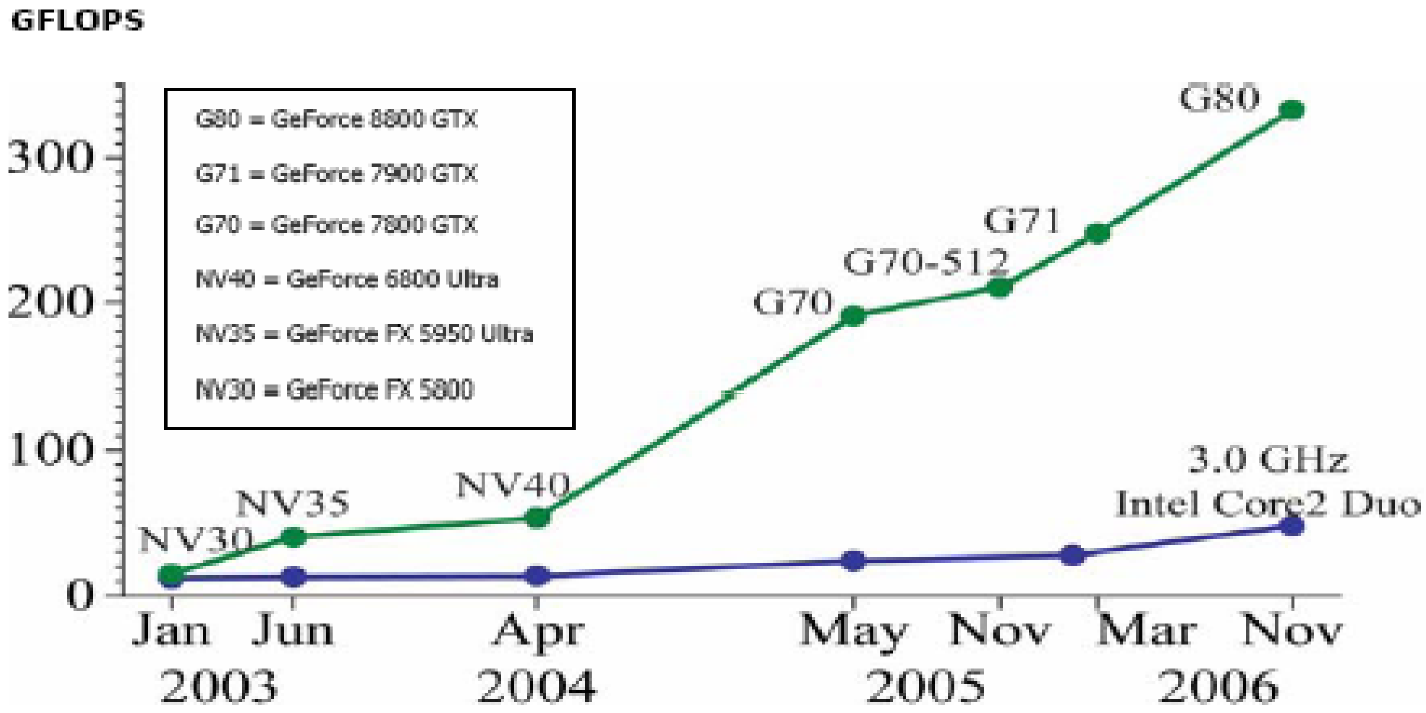


Figure 1-1. Floating-Point Operations per Second for the CPU and GPU

Courtesy NVIDIA

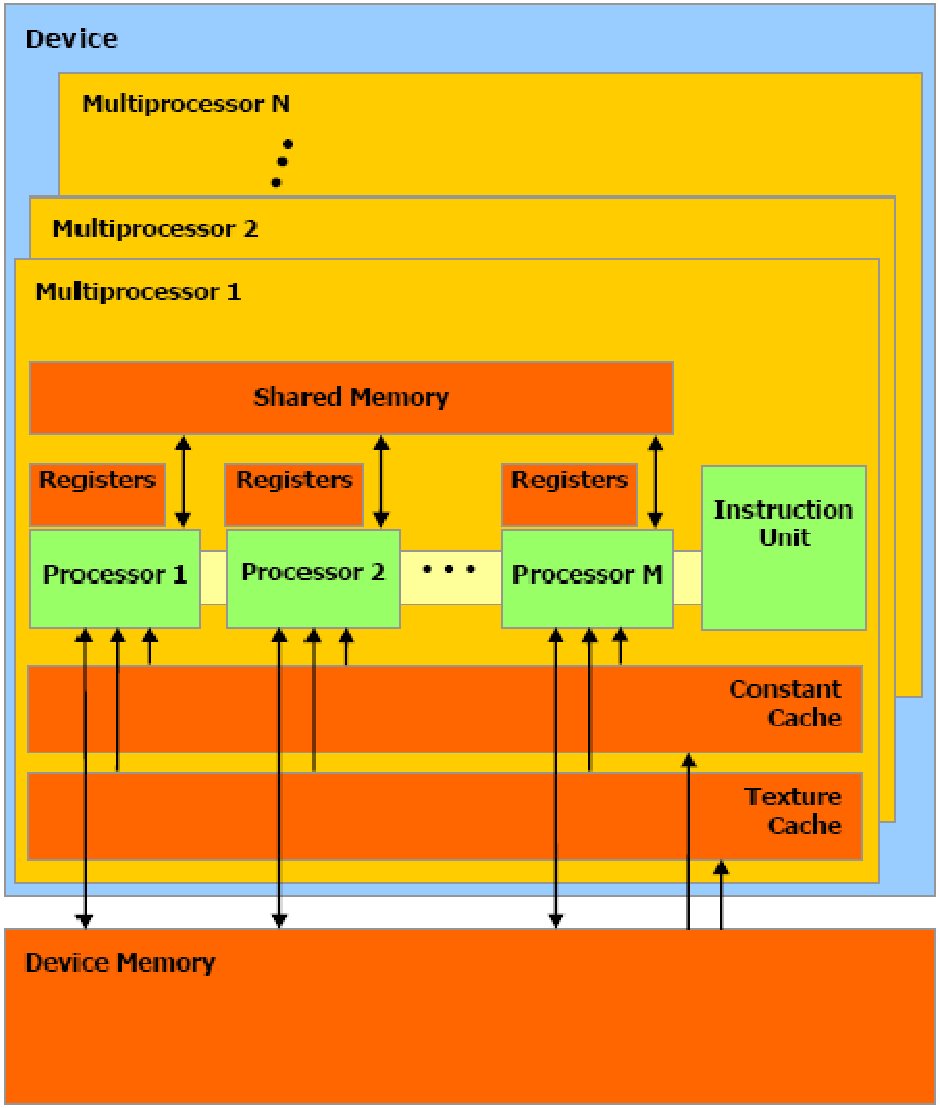
GPU Architecture

Multiple SIMD cores

Multithreaded
O(1000) per GPU

Banked shared memory
16 Kbytes C1060
48 Kbytes C2050

Simple thread model
Only sync at host



A set of SIMD multiprocessors with on-chip shared memory.

Figure 3-1. Hardware Model

Courtesy NVIDIA

Fortran vs CUDA

```
do j = jl, jr
  do i = jr + 1, ld
    x = 0.0
    do k = jl, j - 1
      x = x + s(i, k) * s(k, j)
    end do
    s(i, j) = s(i, j) - x
  end do
end do
```

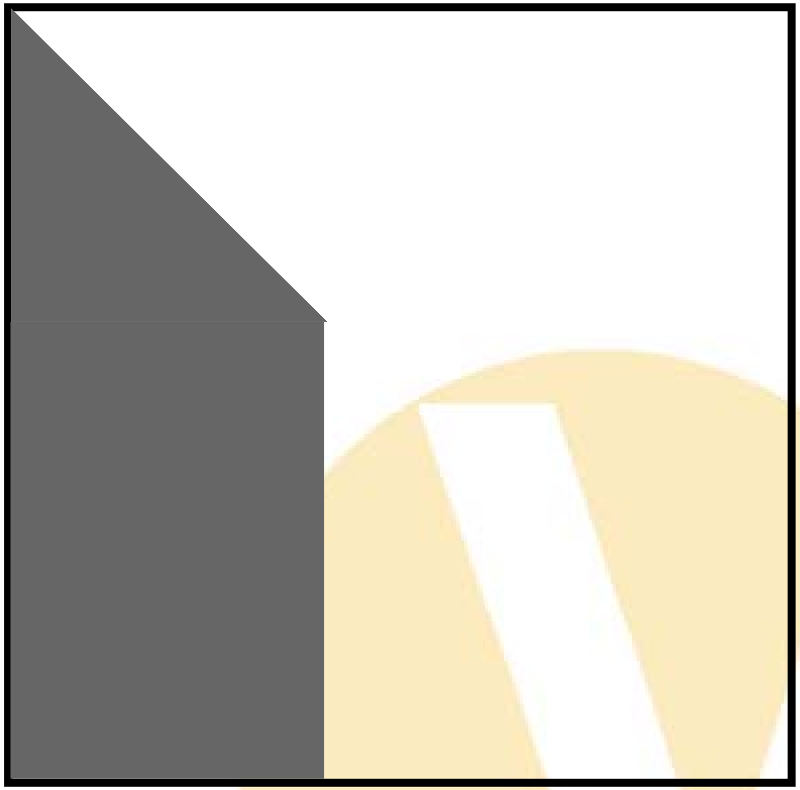
```
ip=0;
for (j = jl; j <= jr; j++) {
  if(ltid <= (j-1)-jl){
    gpulskj(ip+ltid) = s[IDXS(jl+ltid,j)];
    }
  ip = ip + (j - 1) - jl + 1;
  }

__syncthreads();

for (i  = jr + 1 + tid; i <= ld;
     i += GPUL_THREAD_COUNT) {
  for (j = jl; j <= jr; j++) {
    gpuls(j-jl,ltid) = s[IDXS(i,j)];
    }
  ip=0;
  for (j = jl; j <= jr; j++) {
    x = 0.0f;
    for (k = jl; k <= (j-1); k++) {
      x  = x  + gpuls(k-jl,ltid) * gpulskj(ip);
      ip = ip + 1;
      }
      gpuls(j-jl,ltid) -= x;
    }
  for (j = jl; j <= jr; j++) {
    s[IDXS(i,j)] = gpuls(j-jl,ltid);
    }
  }
```

Initial Experiment

Assemble frontal matrix on host CPU

Initialize by sending panel of assembled frontal matrix

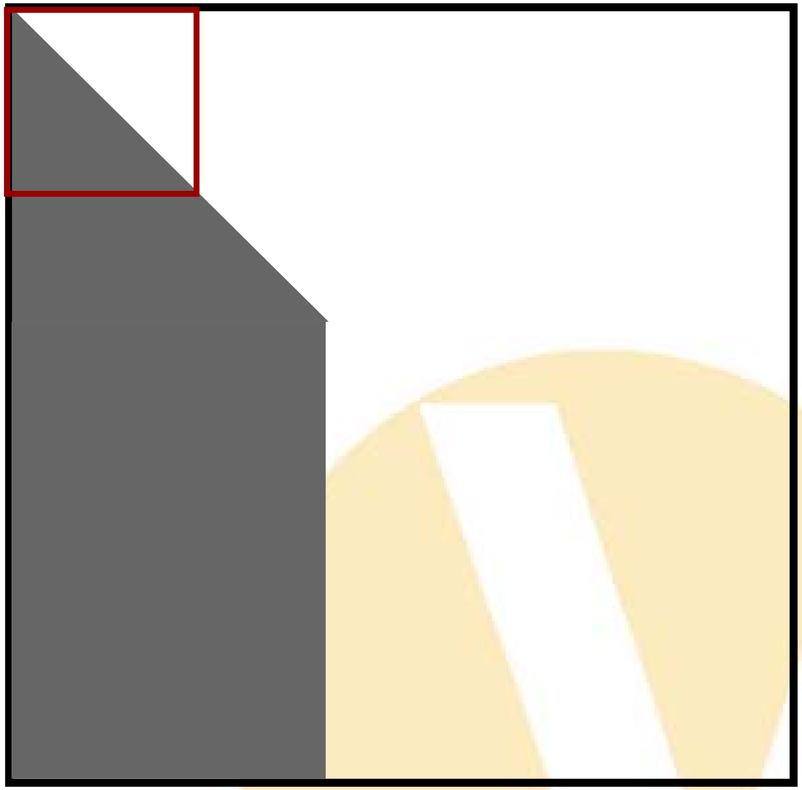
Only large frontal matrices due to high cost of sending data to and from GPU



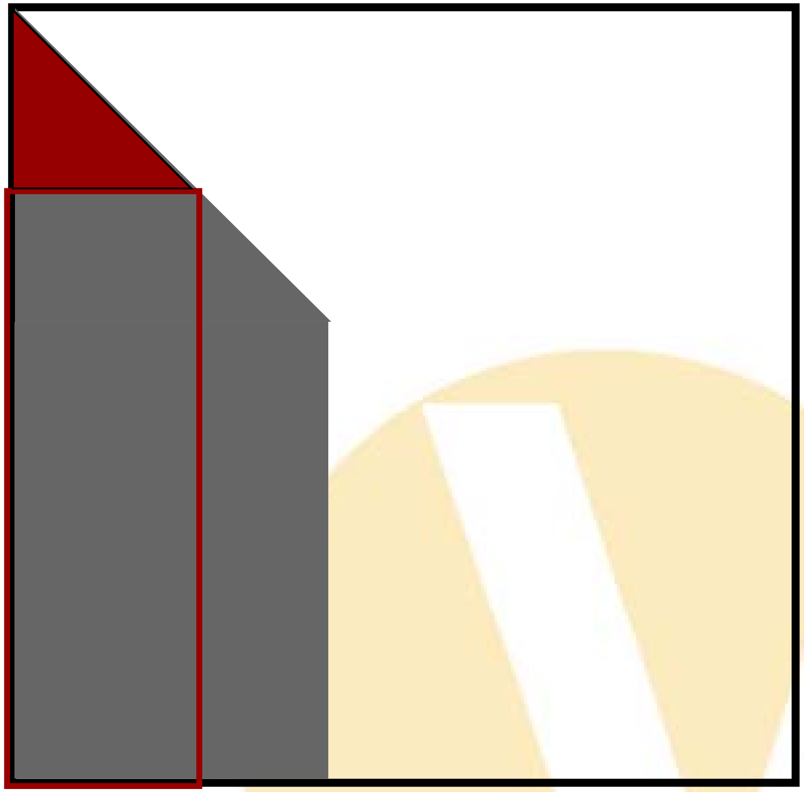
Eliminate panels

Factor diagonal block

Note: host is faster, but its better to avoid data transfer



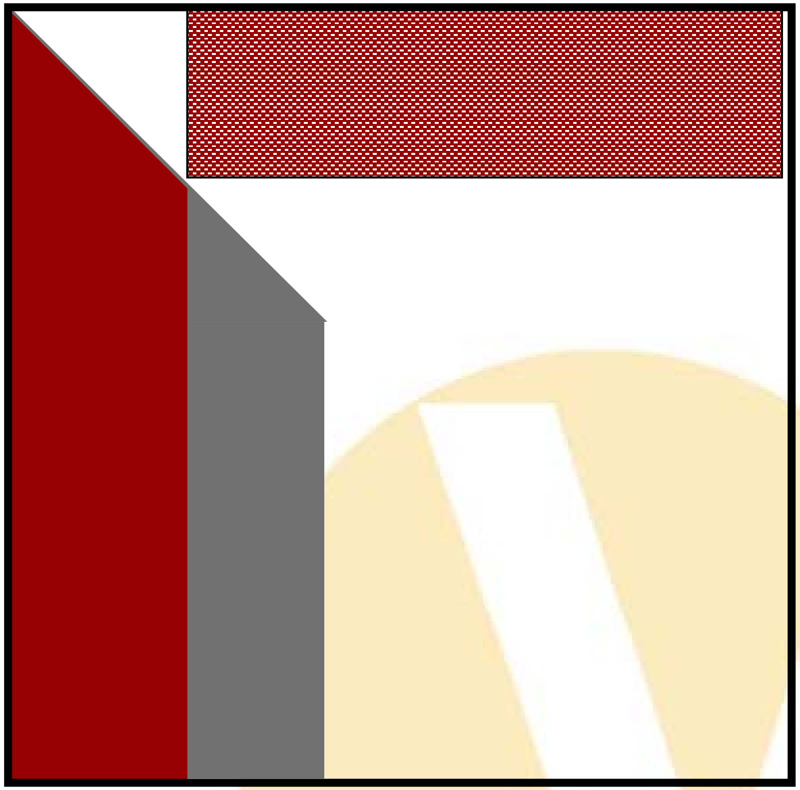
Eliminate panels



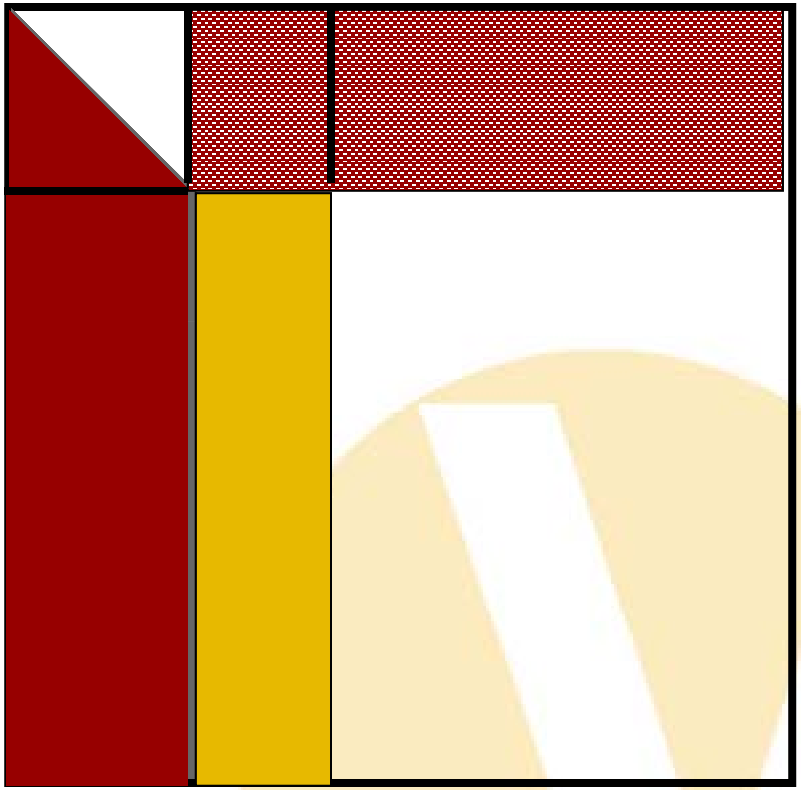
Eliminate off-diagonal panel

Earlier CUDA code

Fill Upper Triangle



Update Schur Complement



Update panels with DGEMM

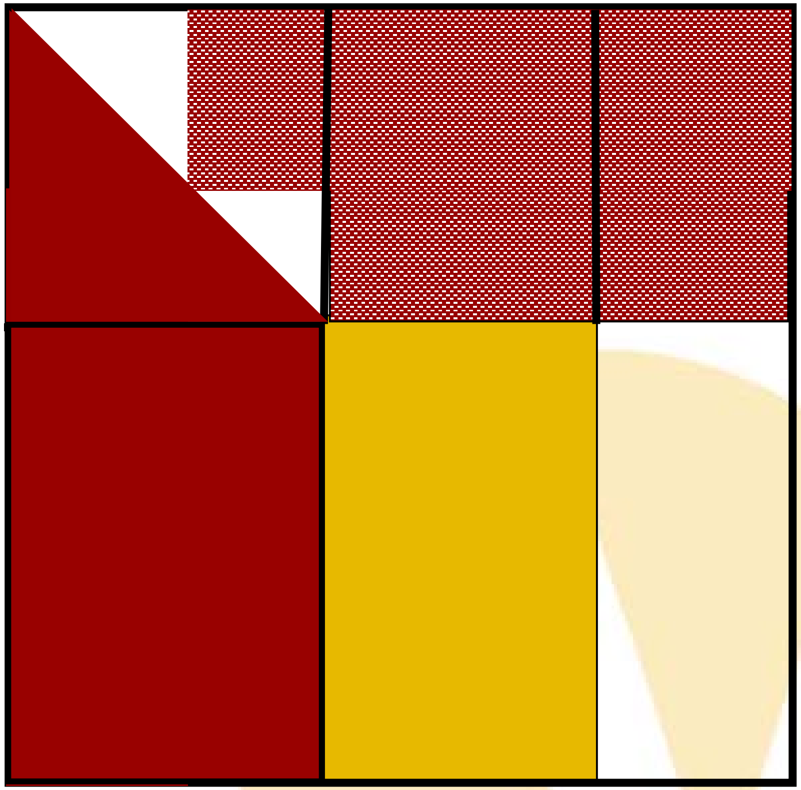
DGEMM is extremely fast!

We've observed >100 GFlop/s
Tesla C2050 (i4r8)

Update Schur Complement

Wider panels in Schur complement

DGEMM is even faster

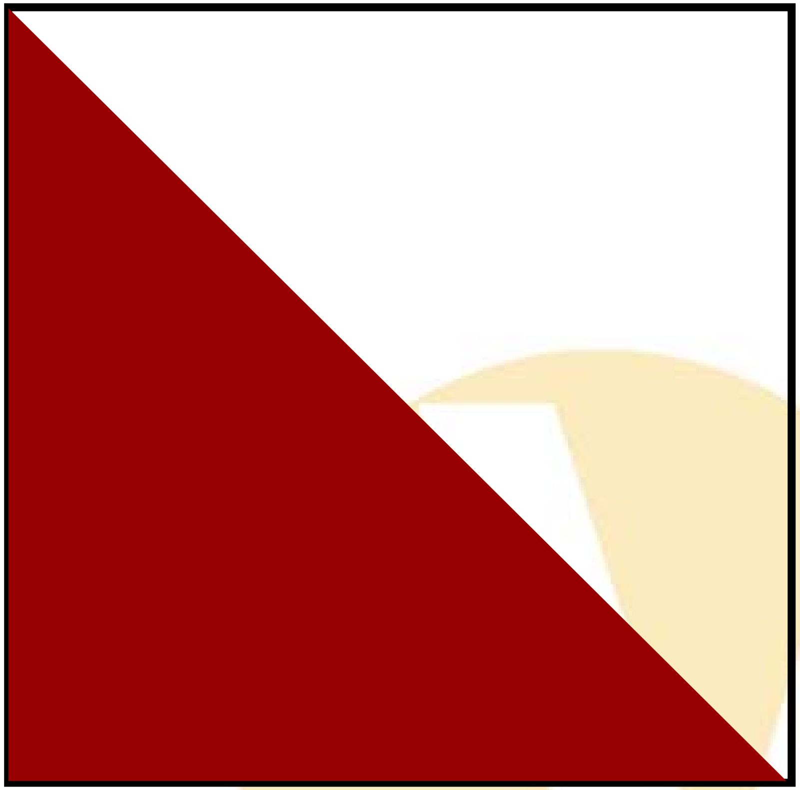


Return Entire Frontal Matrix

Return error if diagonal of 0.0 encountered or pivot threshold exceeded

Otherwise complete frontal matrix is returned

Schur complement added to initial values on host CPU



Factoring a Frontal Matrix Timing on C1060 (i4r4)

Method Name	GPU msec	%GPU time
Copy data to and from GPU	201.0	32.9%
Factor 32x32 diagonal blocks	42.6	7.0%
Eliminate off diagonal panels	37.0	6.1%
Update with SGEMM	330.6	54.1%
Total time	611.4	100.0%

Calibrating Expectations
Dense Kernel Performance

Intel Nehalem Host

2 sockets * 4 cores * {4,2} ALUs * 2.6 GHz

We get ~80 GFlop/s (r4) and 53 GFlop/s (r8)

NVIDIA Tesla C1060

30 processors * {8,1} ALUs * 1.3 GHz

We get 170 GFlop/s (r4)

NVIDIA Tesla C2050 (aka, Fermi)

28 processors * {16,8} ALUs * 1.15 GHz

We get 97 GFlop/s (r8)

Kernel Performance (i4r8) C2050 vs 8 Nehalem Cores

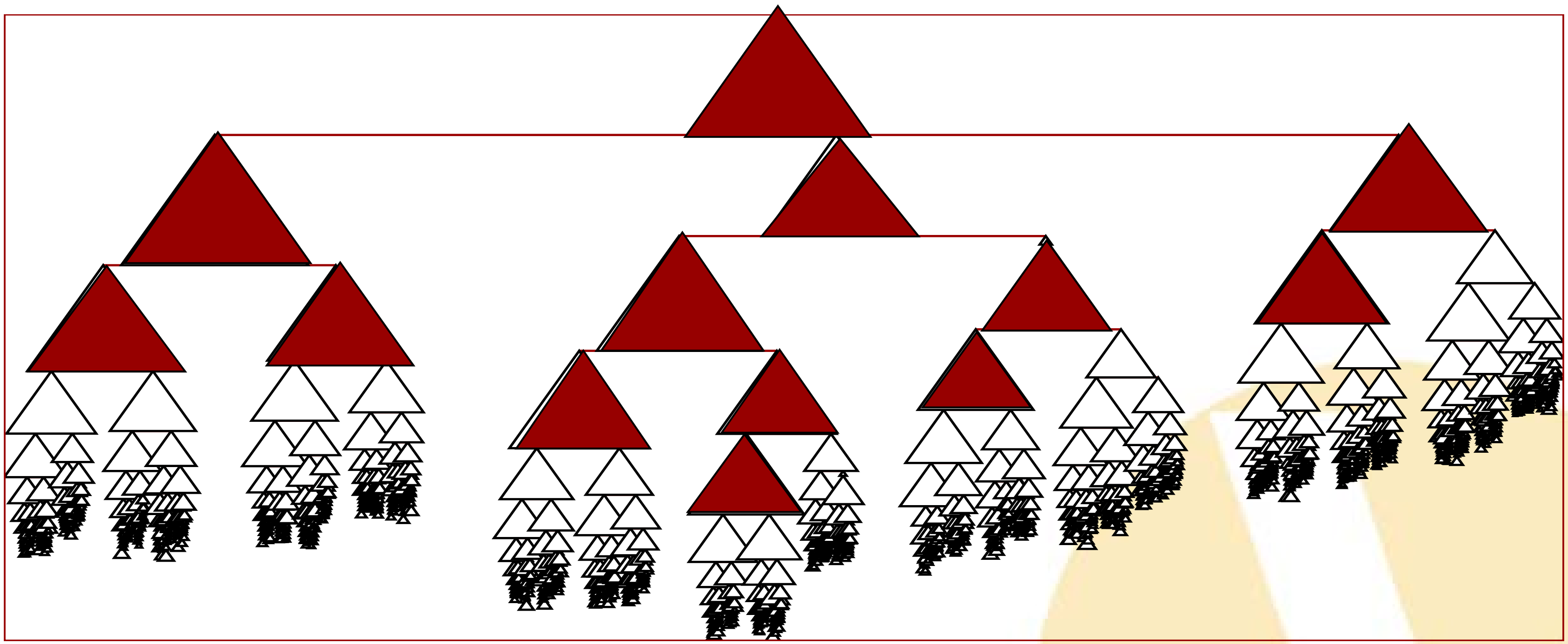
USC Viterbi School of Engineering

Information Sciences Institute

Upper GPU, lower CPU - red means GPU is faster

		Update	Order	
Degree	1024	2048	3072	4096
512	N/A 22.8	23.5 47.0	32.3 49.9	42.0 51.5
1024	22.3 43.2	42.5 48.1	57.0 50.5	66.7 51.8
1536	36.2 42.2	55.5 49.0	68.8 49.9	77.3 52.0
2048	47.9 46.8	66.6 49.8	78.2 51.2	86.1 52.2
2560	57.0 48.0	73.9 50.3	83.6 51.5	91.5 52.0
3072	65.6 49.0	80.1 50.8	89.0 51.4	97.4 52.6

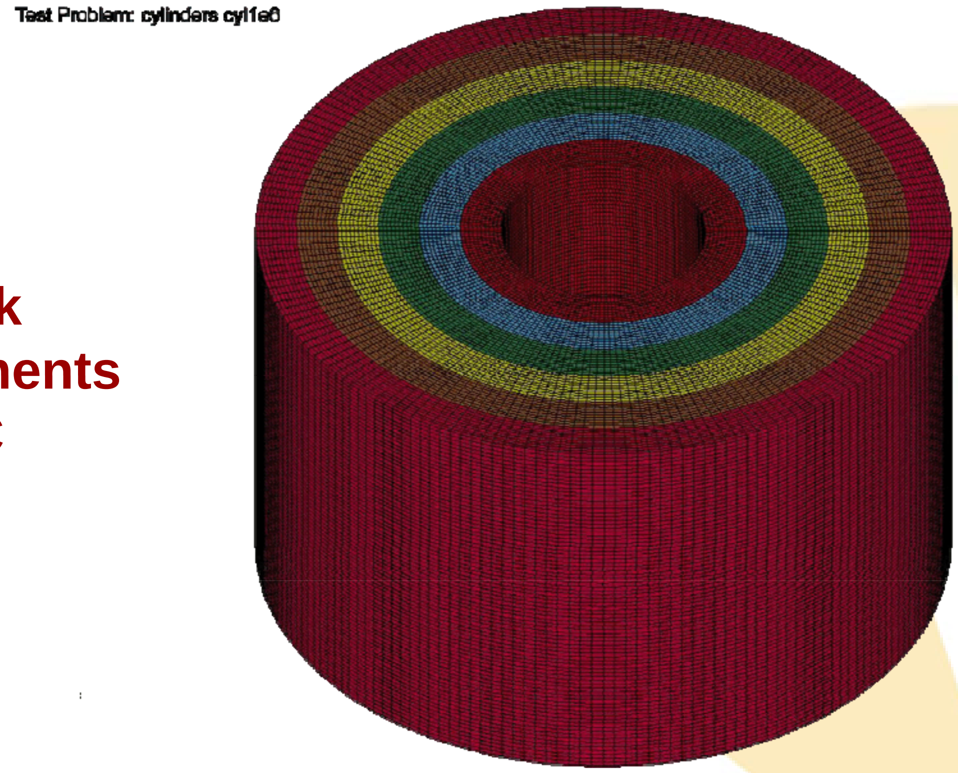
What goes on GPU?



Handful of large supernodes near the root of the tree

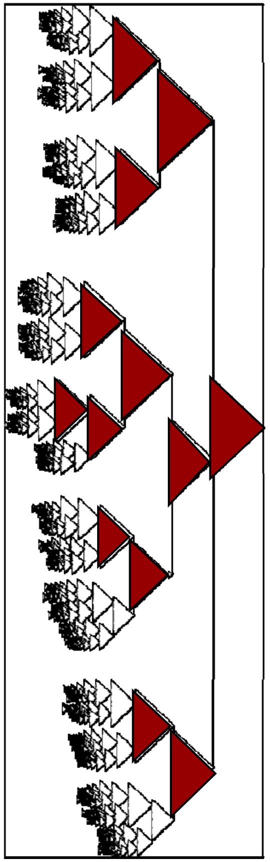
Computational Bottleneck

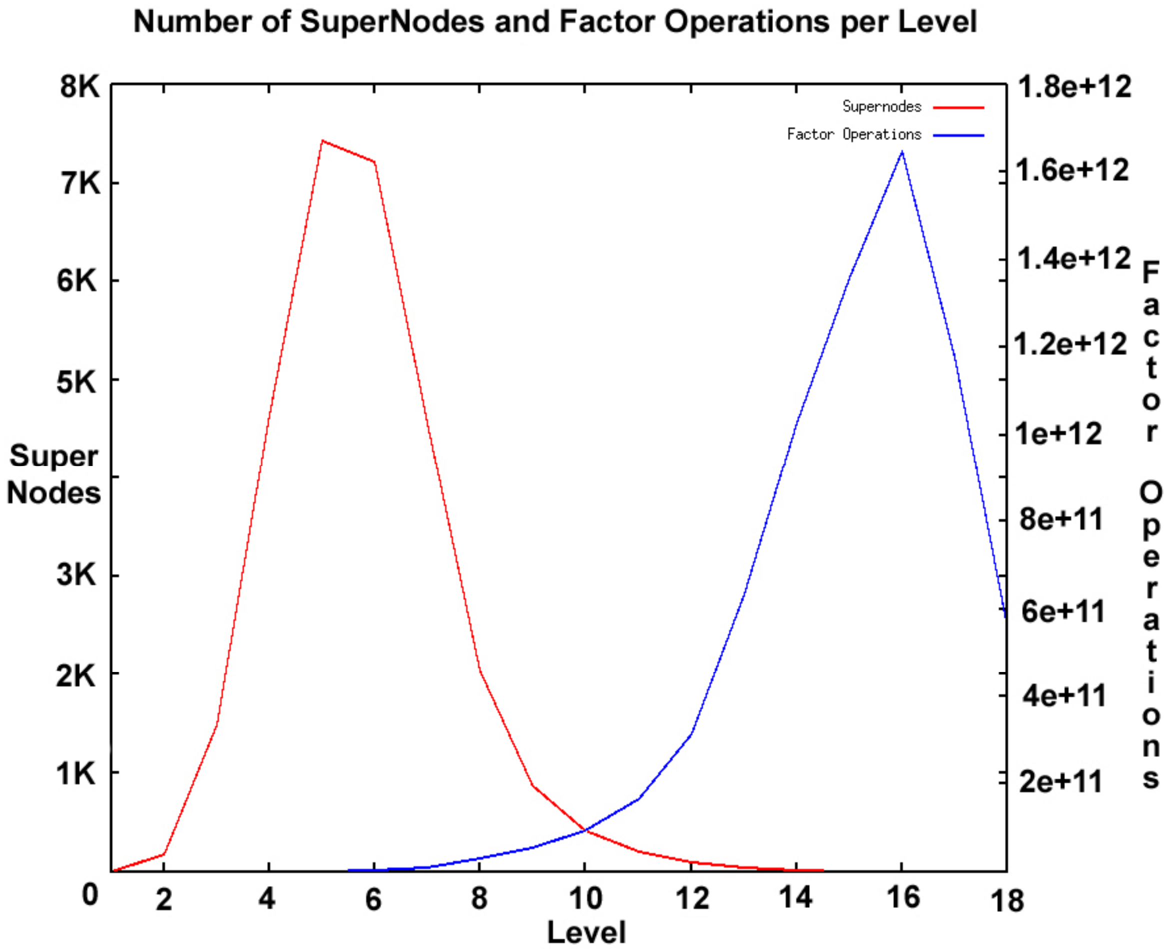
Total time	2057 sec.	
Linear solver	1995 sec.	97%
Factorization	1981 sec.	96%
Suitable for GPU?		88%



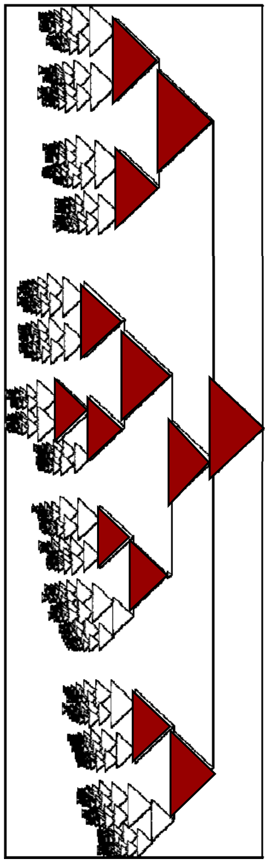
AWE benchmark
230K 3D Finite Elements
Courtesy LSTC

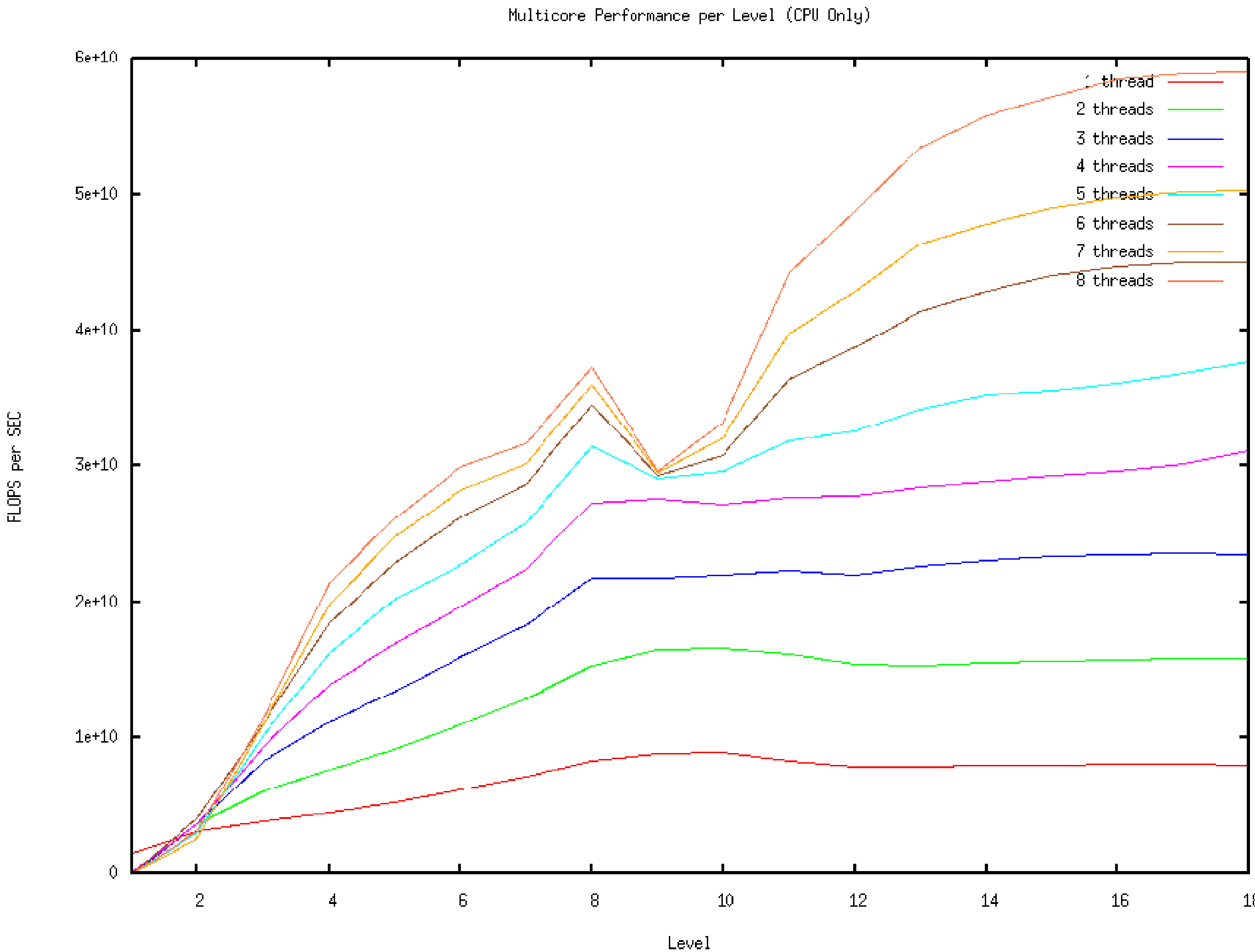
Number of Supernodes & Factor Operations in Tree



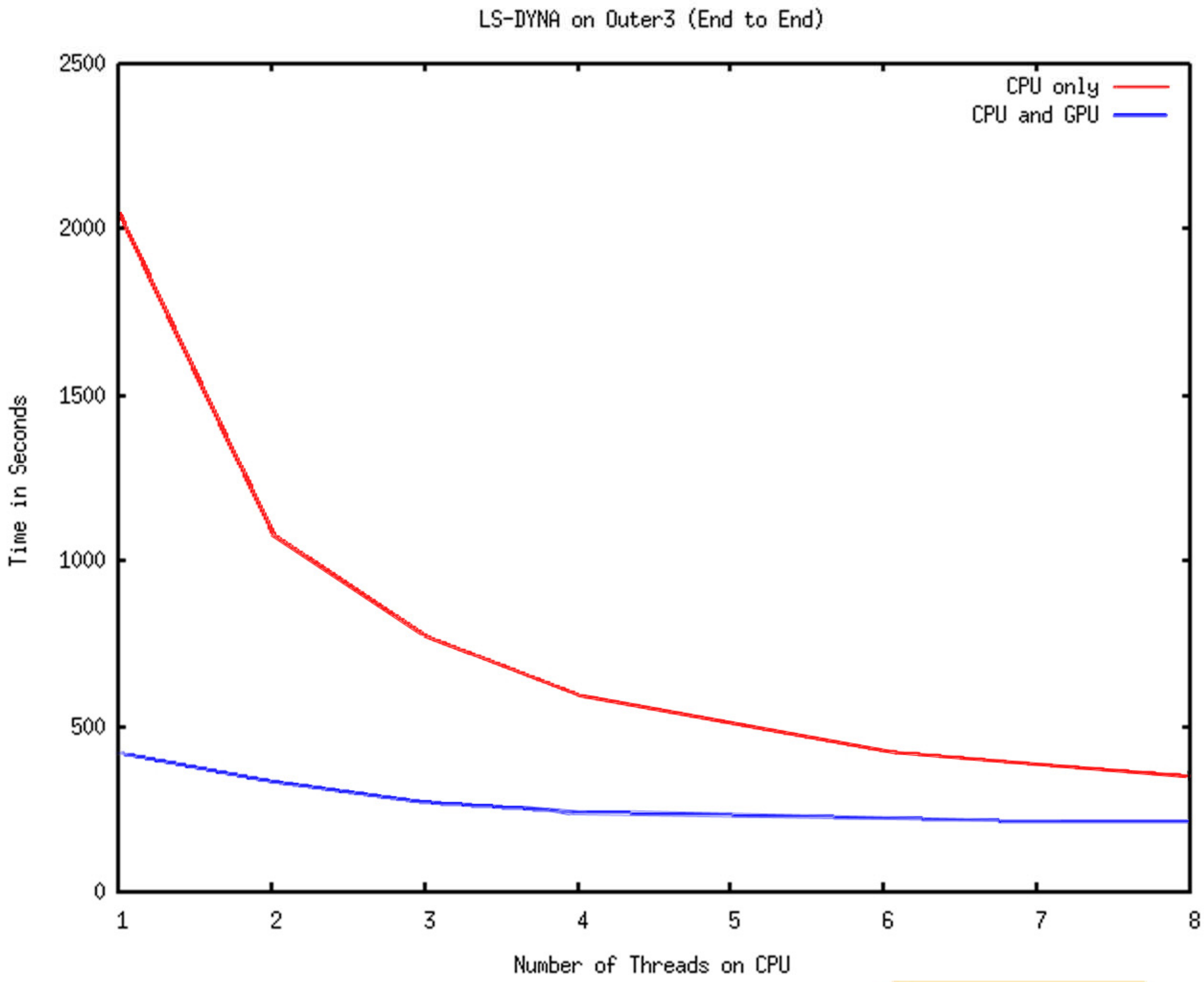


Multicore Performance (i4r4) vs. the Elimination Tree





LS-DYNA Implicit
CPU vs. CPU & GPU (i8r8)



Near-term Future
Bigger Problems

- Problems that don't fit in GPU memory
  - Out-of-core to host memory?
- Performance Optimization
  - Better NVIDIA libraries
  - Re-optimize our CUDA kernel
  - Overlap computation & communication
- Pivoting for numerical stability
- Distributed memory (e.g., MPI)
  - One GPU per Supernode
  - Kernel with MPI and GPUs

CUBLAS 3.2 is Faster

CUBLAS 3.2 based on UTK's MAGMA

We've seen:

- **SGEMM 398 Gflop/s**
- **DGEMM 231 Gflop/s**

Longer-term Future
Smaller Problems

- Factor smaller frontal matrices on GPU
  - Maintain real stack on GPU
  - Assemble initial values on GPU
- If the entire matrix fits on the GPU
  - Forward and back solves
  - Exploit GDRAM memory B/W

Summary

- Factoring large frontal matrices on Nvidia C2050
  - Sped up LS-DYNA implicit
  - Another factor of 2X likely
  - Explicit will be much harder
- Similar results for other implicit MCAE codes
  - BCSLIB-GPU too
- ISVs slowly to come to market
  - Modest speedup
  - Support and pricing issues

Research Partially Funded by JFCOM and AFRL

This material is based on research sponsored by the U.S. Joint Forces Command via a contract with the Lockheed Martin Corporation and SimIS, Inc., and on research sponsored by the Air Force Research Laboratory under agreement numbers F30602-02-C-0213 and FA8750-05-2-0204. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the U.S. Government. Approved for public release; distribution is unlimited.