CUDA Accelerated Linpack on Clusters

E. Phillips, NVIDIA Corporation

Outline

- Linpack benchmark
- CUDA Acceleration Strategy
- Fermi DGEMM Optimization / Performance
- Linpack Results
- Conclusions

LINPACK Benchmark

The LINPACK benchmark is very popular in the HPC space, because it is used as a performance measure for ranking supercomputers in the TOP500 list.

The most widely used implementation is the HPL software package from the Innovative Computing Laboratory at the University of Tennessee:

It solves a random dense linear system in double precision arithmetic on distributed-memory computers.

LINPACK Benchmark

Solve a dense NxN linear system:

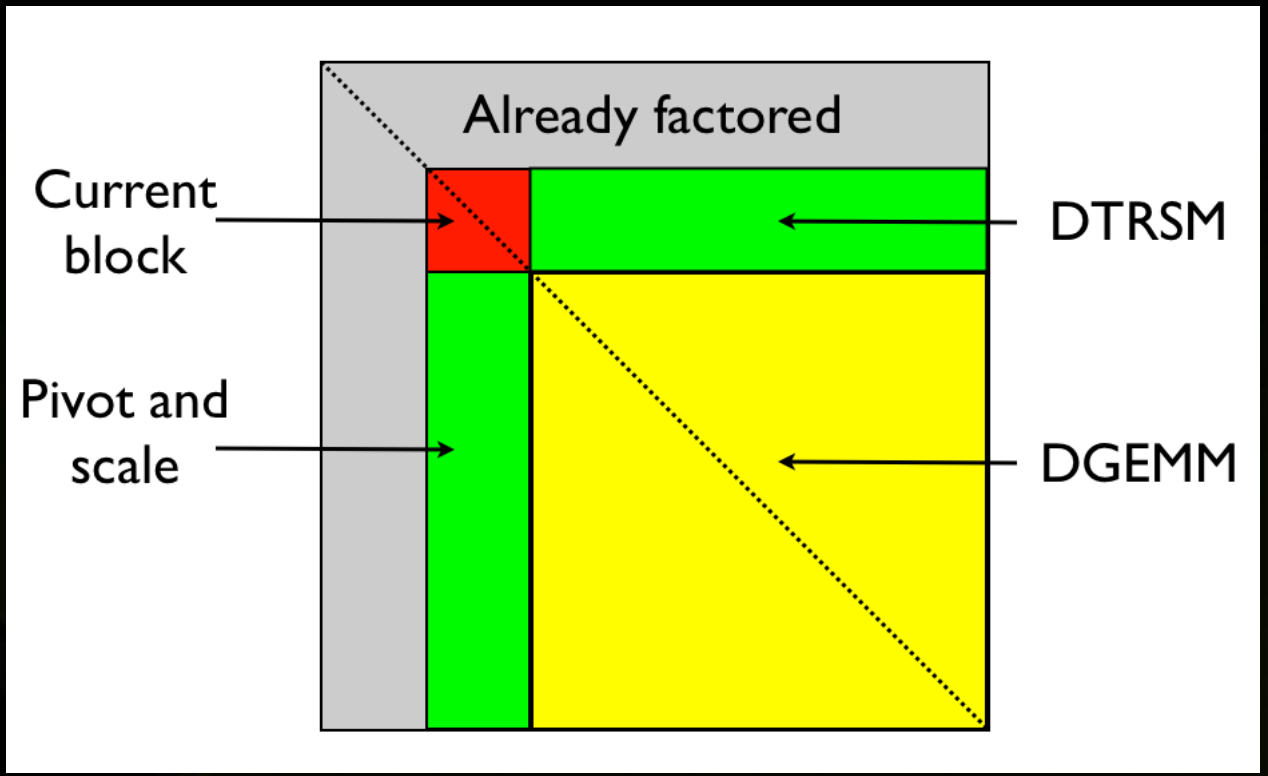
$$Ax=b$$

Solution is obtained by Gaussian elimination with partial pivoting

Floating point workload:

2/3 N^3 + 2 N^2

(LU decomposition) (back solve)



Factorize the current block (red), update the green and yellow parts when done

The bigger the problem size N is, the more time is spent in the update (DGEMM)

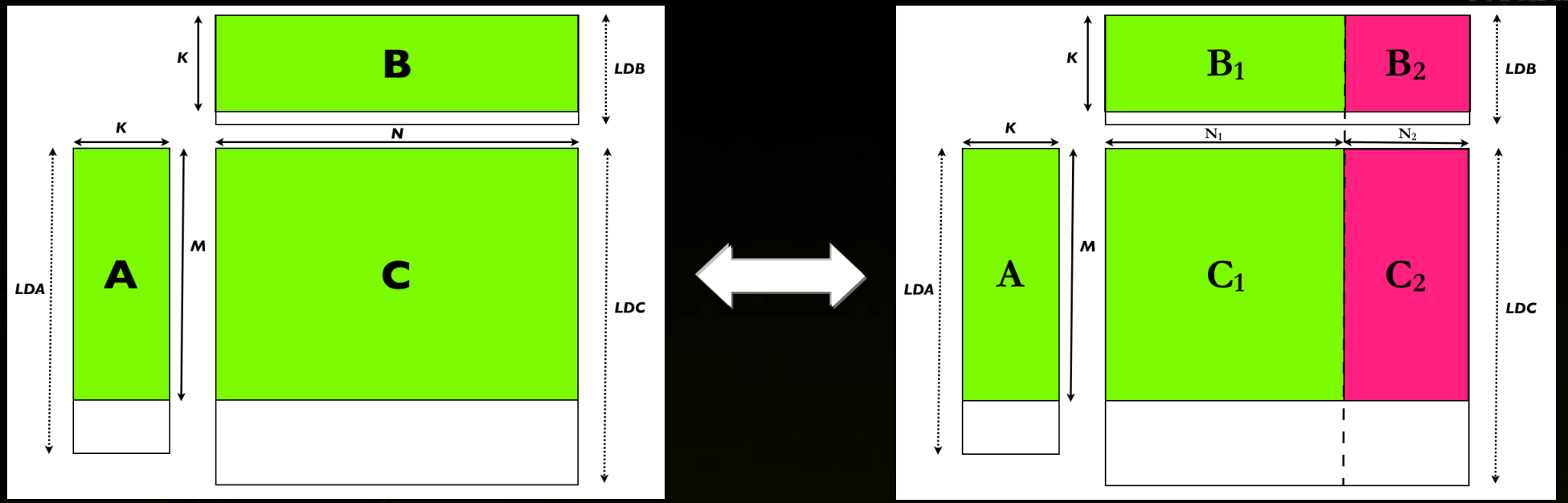
CUDA Accelerated LINPACK

Both CPU cores and GPUs are used in synergy with minor or no modifications to the original source code (HPL 2.0):

- An host library intercepts the calls to DGEMM and DTRSM and executes them simultaneously on the GPUs and CPU cores.

- Use of pinned memory for asynchronous PCI-e transfers, up to 6.3 GB/s on x16 gen2 slots. Only changes to the HPL source

DGEMM: C = alpha A B + beta C



DGEMM(A,B,C) = DGEMM(A,B1,C1) U DGEMM(A,B2,C2)

(CPU)

The idea can be extended to multi-GPU configuration and to handle huge matrices

Find the optimal split, knowing the relative performances of the GPU and CPU cores on DGEMM

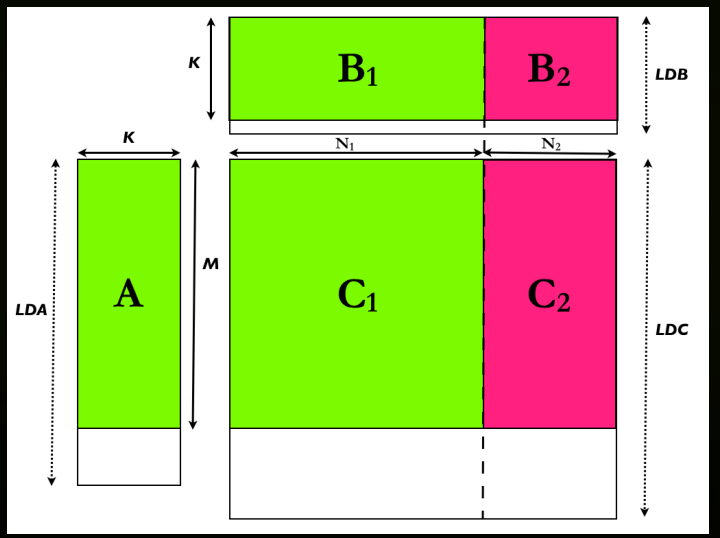
Optimal split

If A(M,K), B(K,N) and C(M,N), a DGEMM call performs 2*M*K*N operations

$T_{CPU}(M,K,N2) = T_{GPU}(M,k,N1)$ $N=N1+N2$

If G_{CPU} denotes the DGEMM performance of the CPU in Gflops and G_{GPU} the one of the GPU, The optimal split is

$\eta = G_{GPU} / (G_{CPU}+G_{GPU})$



NVIDIA Tesla GPU Computing Products

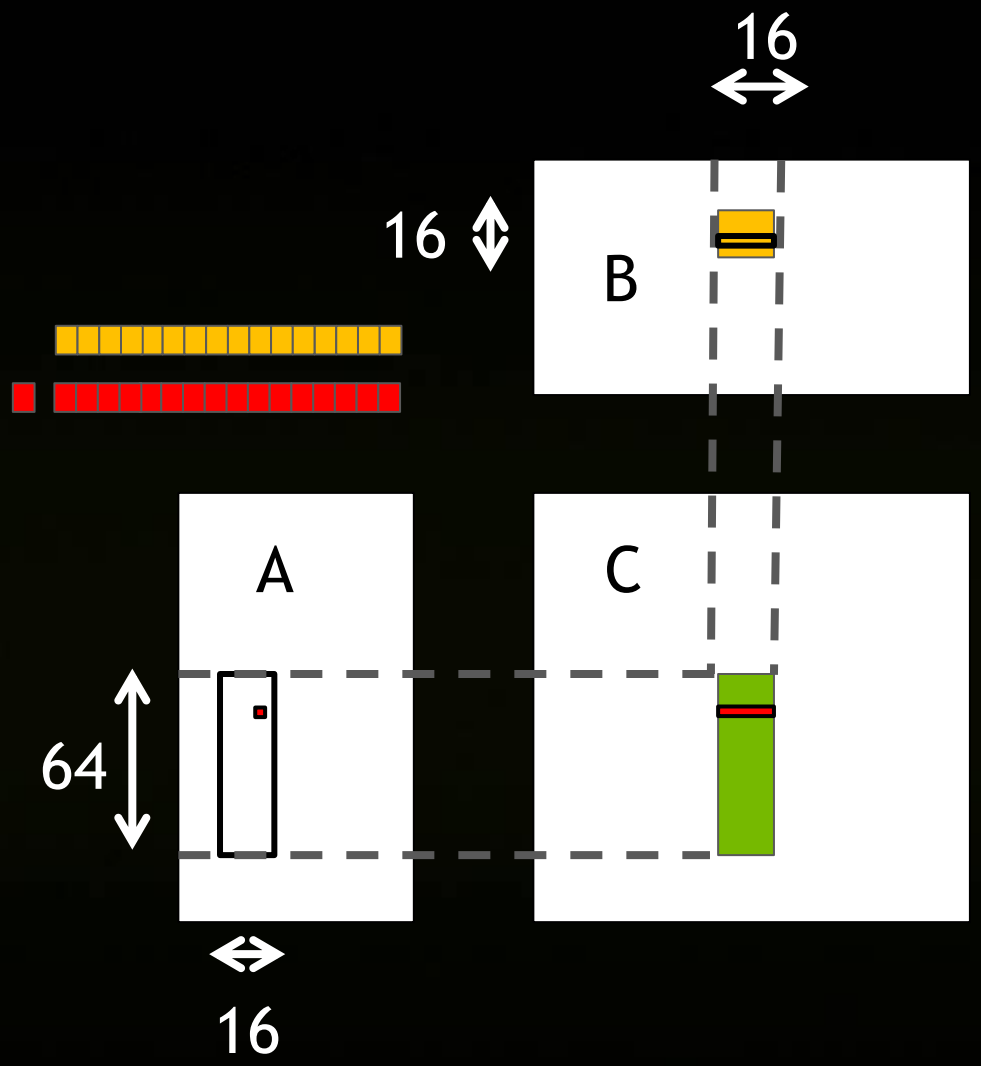
	Server Module		1U Systems		Workstation Boards	
	 Tesla M2070 / Tesla M2050	 Tesla M1060	 Tesla S2050	 Tesla S1070	 Tesla C2070 / Tesla C2050	 Tesla C1060
GPUs	1 T20 GPU	1 T10 GPU	4 T20 GPUs	4 T10 GPUs	1 T20 GPU	1 T10 GPU
Single Precision	1030 GFlops	933 GFlops	4120 GFlops	4140 GFlops	1030 Gflops	933 GFlops
Double Precision	515 Gflops	78 GFlops	2060 GFlops	346 GFlops	515 Gflops	78 GFlops
Memory	6 GB / 3 GB	4 GB	12 GB (S2050)	16 GB 4 GB / GPU	6 GB / 3 GB	4 GB
Mem BW	148.4 GB/s	102 GB/s	148.4 GB/s	102 GB/s	144 GB/s	102 GB/s

T20 (Fermi) Architecture

- Increased DP Throughput
  - 515 GFLOPS *PEAK* vs 85 GFLOPS T10
  - Theoretical 75% DGEMM : 515 x .75 = **386 GFLOPS**
- Cache Heirarchy
  - 64KB configurable L1 + shared memory (48/16 or 16/48)
  - 768 KB unified L2
  - Load/Store architecture
- Dual Copy Engines
  - Overlap both download and upload of data with compute
  - Support Asynchronous 2D data transfers

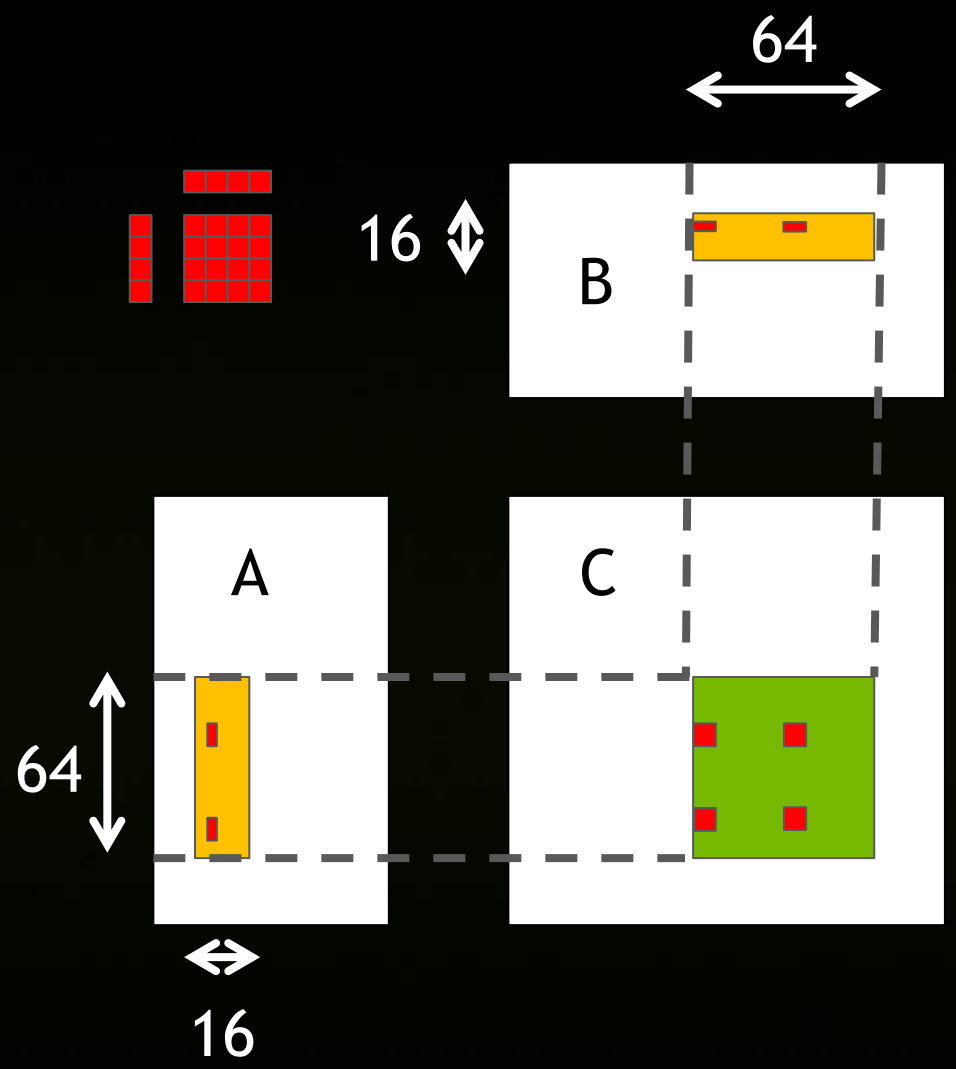
T20 DGEMM Optimization

- T10 DGEMM algorithm (Volkov)
  - 64 Threads 64x16 of C
  - B reuse in shared memory
  - A loaded from global memory
  - DP limited on T10
    - DP 8x slower than other instructions
- T20 DP full speed
  - T10 DGEMM runs 175 GFLOPS
  - Different Bottlnecks



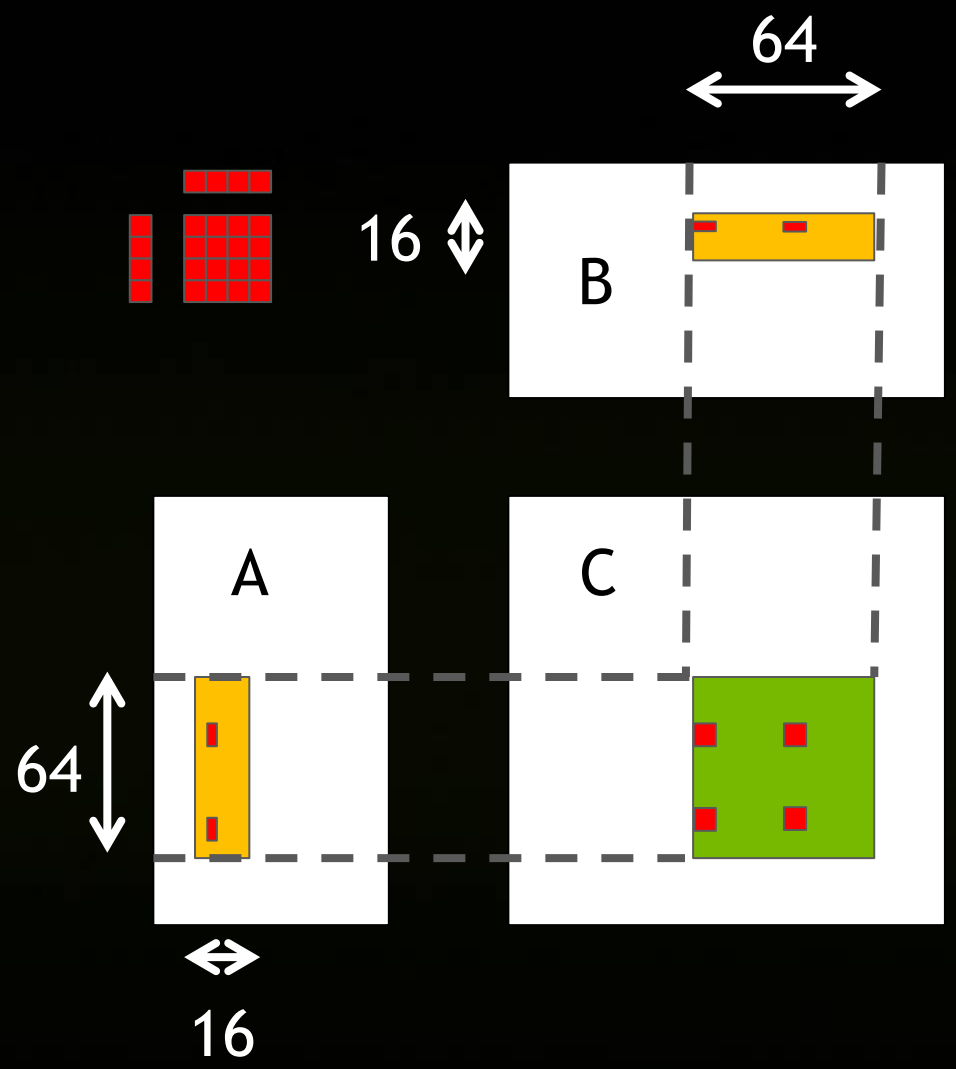
T20 DGEMM Optimization

- 16x16 Threads update 64x64 of C
- Instruction throughput bottleneck
  - Maximize Density of DFMA instrucitons
  - Register Blocking
  - Use wide vector Loads (128 bits)
  - DFMA dual issues with Texture
    - Texture fetch A and B
- Hide Latencies
  - Double Buffer Shared Memory



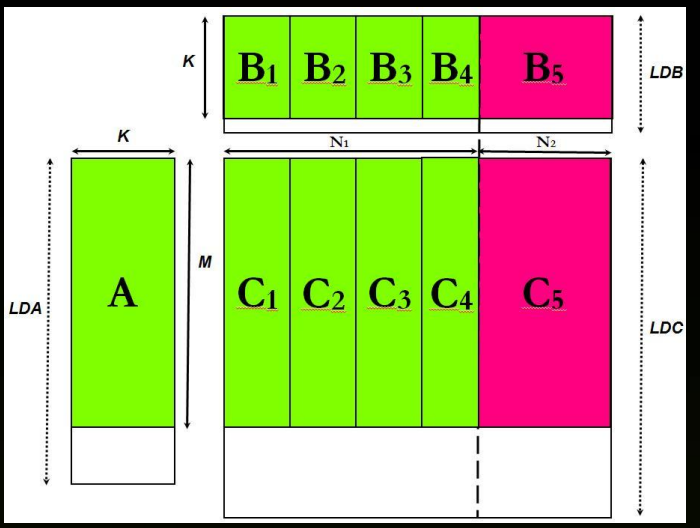
T20 DGEMM Performance

- Inner Loop
  - Multiply 4x4 elements per thread
  - 4 Loads (128-bit), 16 DFMA = 80%
- Outer Loop (16 iterations)
  - 256 DFMA / 341 instruction = 75 %
- CUDA code ~ 301 GFLOPS
  - General - all configs - all sizes
- Linpack version ~360 GFLOPS
  - NT config, M%64, N%64, K%16
  - 360/515 = 70% 360/386 = 93%



Fermi DGEMM Strategy

- Slice Matrix into several pieces
- Use Stream API
  - Overlap COPY + Compute



Copy A H2D

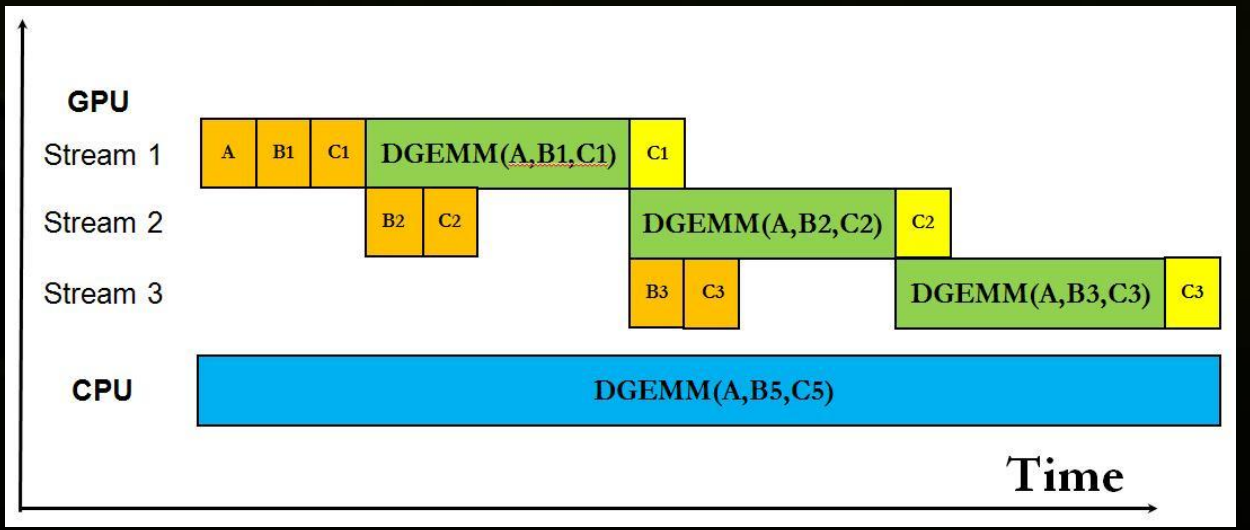
Loop over pieces:

Copy Bi,Ci H2D

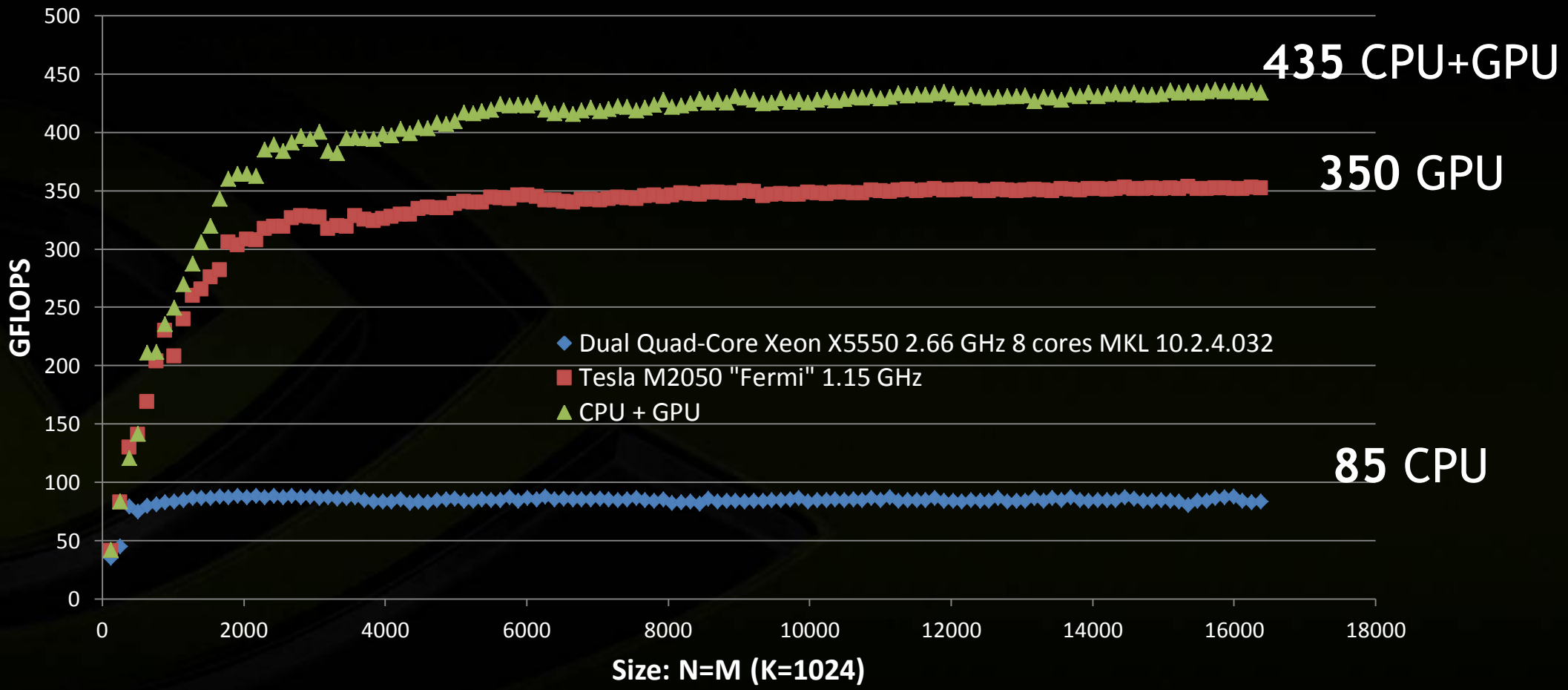
DGEMM A,Bi,Ci

Copy Ci D2H

CPU DGEMM A,B_last,C_last



Fermi DGEMM Performance



Optimizations - Auto Split

- Keep track of CPU and GPU performance and adjust split
  - Wallclock() CPU time
  - Cuda event record GPU time
  - Compute optimal split for next iteration

```
cudaEventRecord(GPU_start,0);
Loop: launch GPU copys + Kernels
cudaEventRecord(GPU_stop,0);
CPU_start = wallclock();
Call CPU_DGEMM
CPU_stop = wallclock();
cudaEventSynchronize(GPU_stop);
```

```
GPU_GFLOPS = GPU_FLOPS/GPU_TIME
CPU_GFLOPS = CPU_FLOPS/CPU_TIME
SPLIT = GPU_GFLOPS/(GPU_GFLOPS+CPU_GFLOPS)
```

Results on single node

Supermicro 6016GT-TF: Dual Intel Xeon X5560 Nehalem 2.8 GHz
96GB memory, 1 or 2 Tesla M2050 1.15Ghz cards

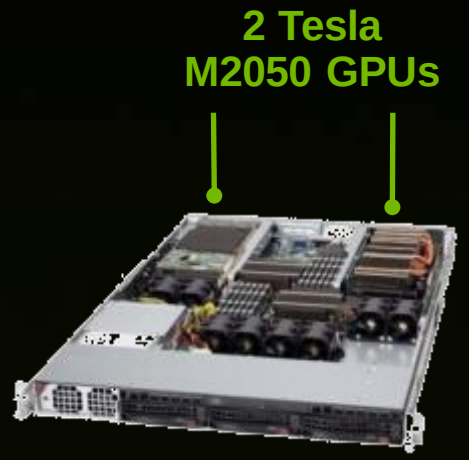
-Peak Synthetic = 89 + 515 = 604 GFLOPS
-Peak DGEMM = 89 + 0.75*515 = 475 GFLOPS

```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR10L2L2       108032   768     1     1            2011.42             4.179e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0039415 ...... PASSED
================================================================================
```

417.9 GFLOPS = 69% of *PEAK* or 88% of *PEAKDGEMM*

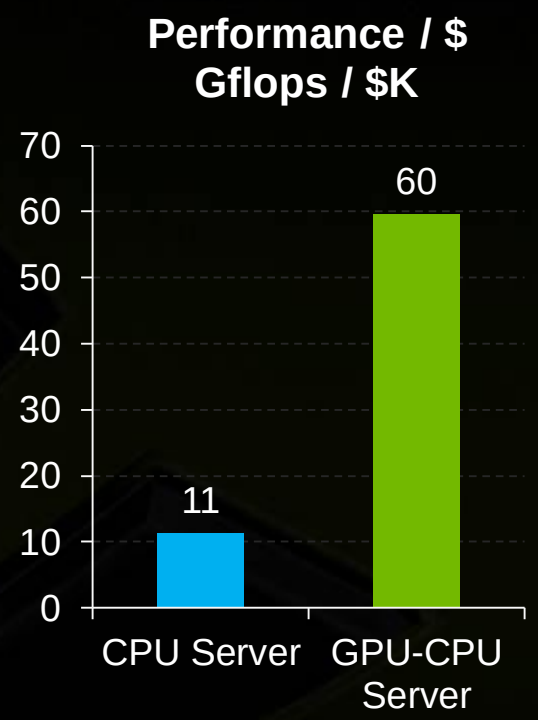
```
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR10L2L2       108032   768     1     2            1192.13             7.051e+02
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0040532 ...... PASSED
================================================================================
```

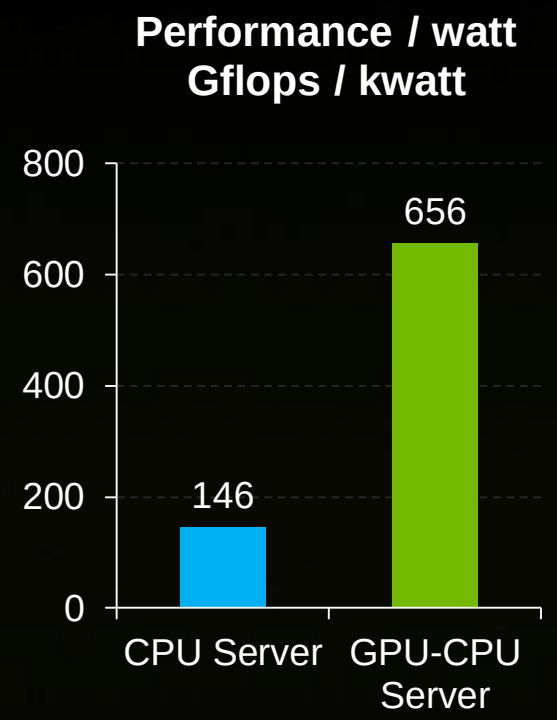
705.1 GFLOPS = 63% of *PEAK* or 81% of *PEAKDGEMM*



SuperServer 6016GT-TF
2 CPUs + 2 GPUs in 1U

Single Node Linpack Performance



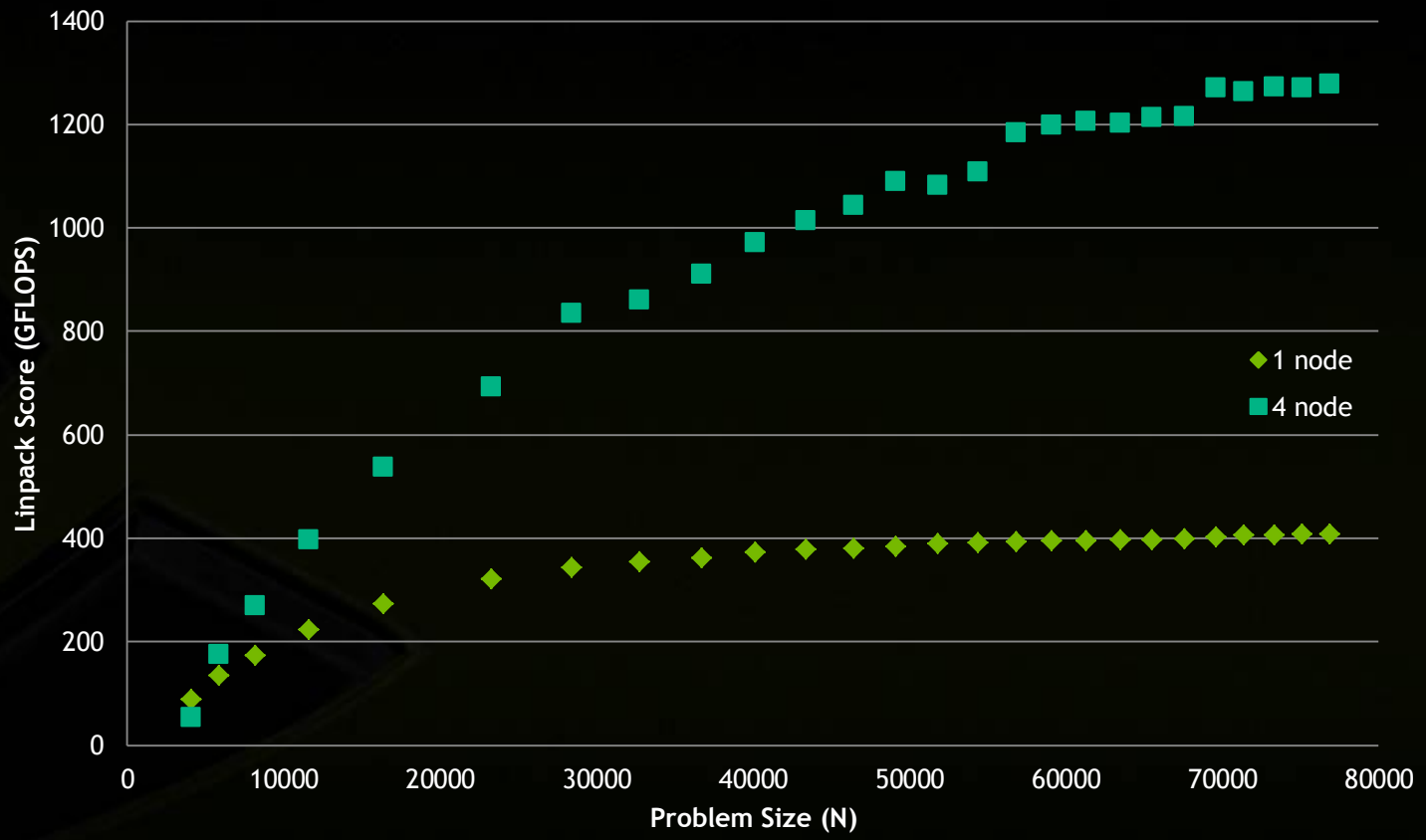


CPU 1U Server: 2x Intel Xeon X5550 (Nehalem) 2.66 GHz, 48 GB memory, $7K, 0.55 kw

GPU-CPU 1U Server: 2x Tesla C2050 + 2x Intel Xeon X5550, 48 GB memory, $11K, 1.0 kw

Effect of Problem Size (memory size)

- Memory use ~ 8 bytes*N*N
- More memory increases performance and efficiency

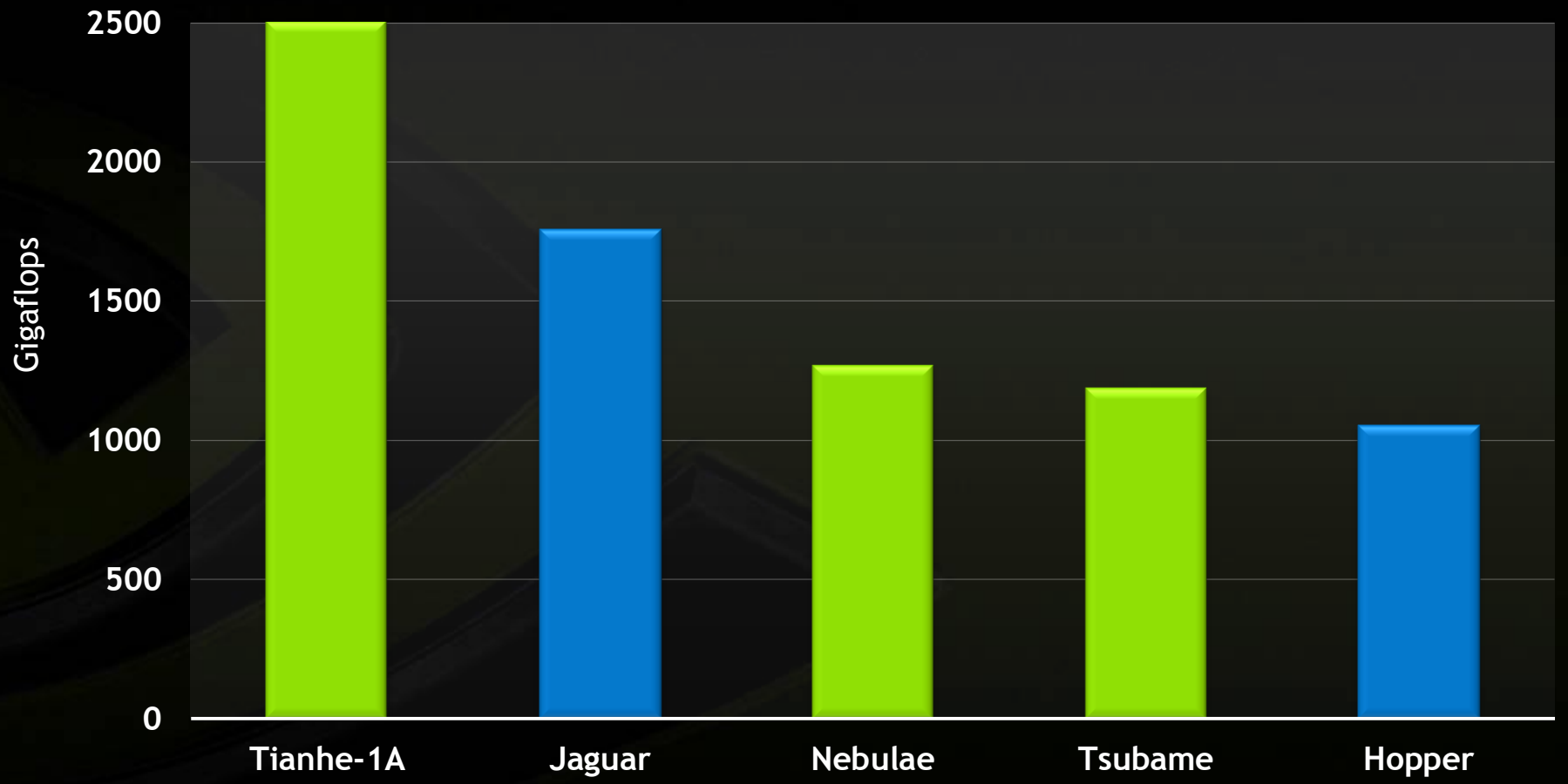


Cluster Performance

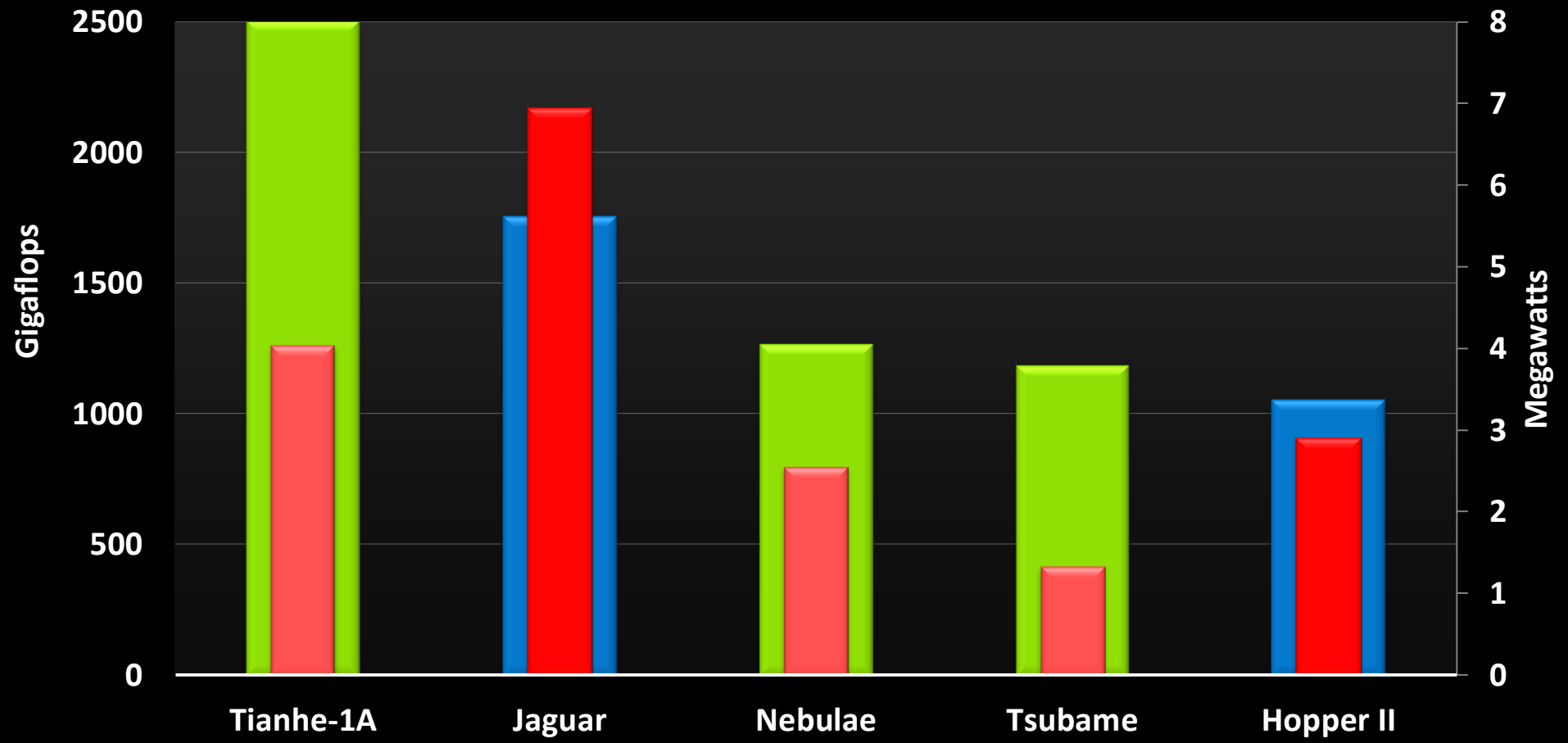
CASPUR Jazz Cluster			
Node	16x HP DL360 G7	**Interconnect**	Infiniband QDR
CPU	2x X5650 @ 2.67 GHz (6 cores)	**Switch**	HP (Voltaire) 36P Managed Switch
RAM	48 GB	**HCA**	HP InfiniBand 4X QDR ConnectX-2 PCIe G2 Dual Port
Peak Perf.	1152 GFLOPS (DP)	**GPU**	NVIDIA S2050 (half x node)
xHPL Perf.	892 GFLOPS (DP)		(2x GPUs C2050 @ 1.15 GHz & 3 GB DDR5 RAM)
Notes	MKL_NUM_THREADS = 6 /CPU		OMPL_NUM_THREADS = 6 /CPU
	Infiniband QDR as OpenIB no RDMA		**GFLOPS: best results per # Procs**
	mpirun -mca tbl openib,self -mca btl_openib_flags 1 -bysocket -bind-to-socket -hostfile ./myhostfile -np #Procs ./run_linpack		

						GFLOPS			
#Nodes	**#Procs**	**PxQ**	**N**	**MemTot(MB)**	**MemNode(MB)**	**Nb=768**	**x Node**	**% xHPL**	**% peak**
1	1	1x1	70000	41870	41870	410	410	80.6%	64.4%
1	2	1x2	70000	41870	41870	640	640	71.7%	55.6%
2	4	2x2	105000	94208	47104	1319	660	73.9%	57.2%
4	8	2x4	150000	192261	48065	2551	638	71.5%	55.4%
6	12	3x4	180000	276855	46143	3814	636	71.3%	55.2%
8	16	4x4	210000	376831	47104	5092	637	71.4%	55.3%
10	20	4x5	235000	471893	47189	6295	630	70.6%	54.6%
12	24	4x6	256000	560000	46667	7498	625	70.0%	54.2%
14	28	4x7	280000	669922	47852	8771	627	70.2%	54.4%
15	30	5x6	290000	718628	47909	9147	610	68.4%	52.9%
16	32	4x8	300000	769043	48065	10050	628	70.4%	54.5%

Large-Scale Performance



Power Efficiency



Future Challenges

- Compute growing faster than bandwidths
  - System Memory, PCIe, Network
- Optimize HPL code to reduce GPU idle time
  - Pipeline swap / update
- Dynamic Task Scheduling / Load Balancing
  - PLASMA / MAGMA / StarPU
  - Task Dependency Graph / Performance Models

Conclusions

- Easy to accelerate the Linpack performance of workstations and clusters using Tesla and CUDA
- Increasing the performance per node reduces the cost of high performance interconnects on clusters
- Improved Power Efficiency
- Code is available from NVIDIA