

The background of the slide features several overlapping, curved, metallic-looking shapes in shades of gray and black, set against a dark, fine-grained texture. These shapes resemble stylized architectural elements or perhaps the folds of a material.

# *Accelerating Linpack with CUDA on heterogeneous clusters*

*Massimiliano Fatica*  
*[mfatica@nvidia.com](mailto:mfatica@nvidia.com)*



# Outline



- **Linpack benchmark**
- **CUBLAS and DGEMM**
- **Results:**
  - **Accelerated Linpack on workstation**
  - **Accelerated Linpack on heterogeneous cluster**



# Linpack



# 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:

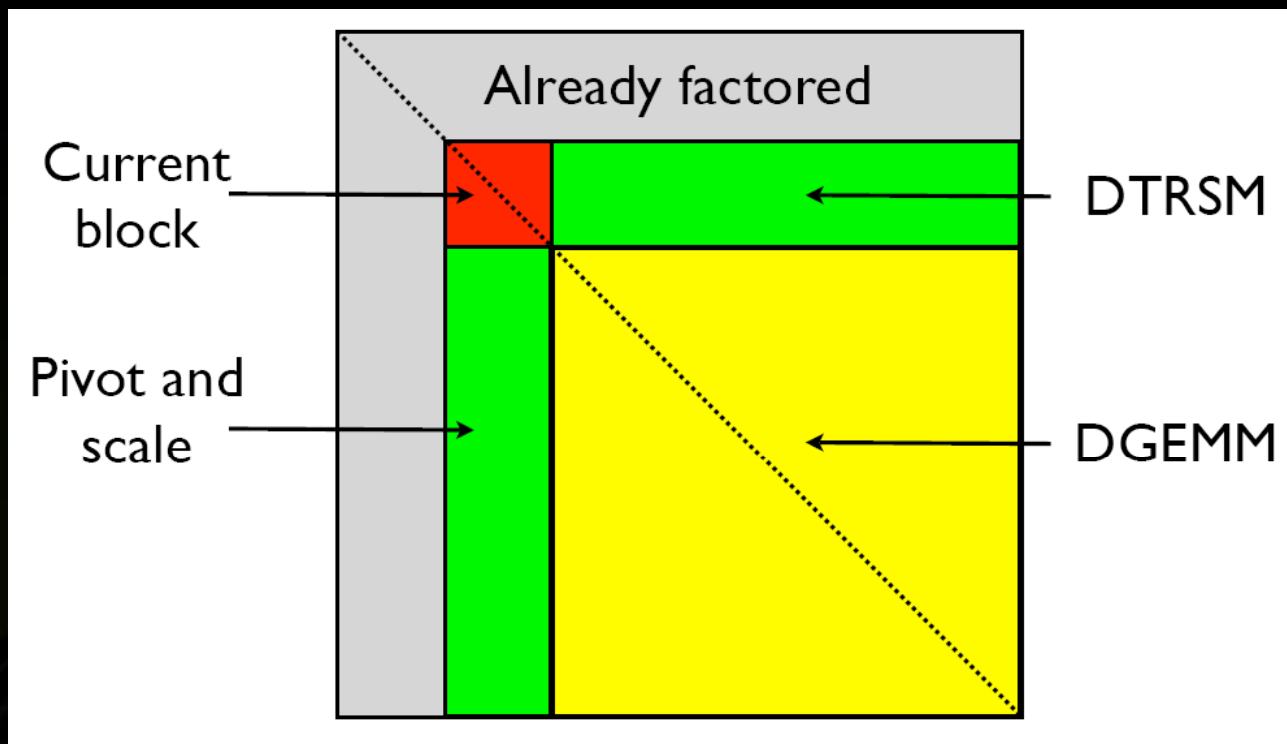
$$Ax=b$$

Solution is obtained by Gaussian elimination with partial pivoting

Floating point workload:

$$\frac{2}{3} N^3 + 2 N^2$$

(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 of the trailing matrices (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. Library is implemented with CUBLAS
- Use of pinned memory for fast PCI-e transfers, up to 5.7GB/s on x16 gen2 slots. Only changes to the HPL source



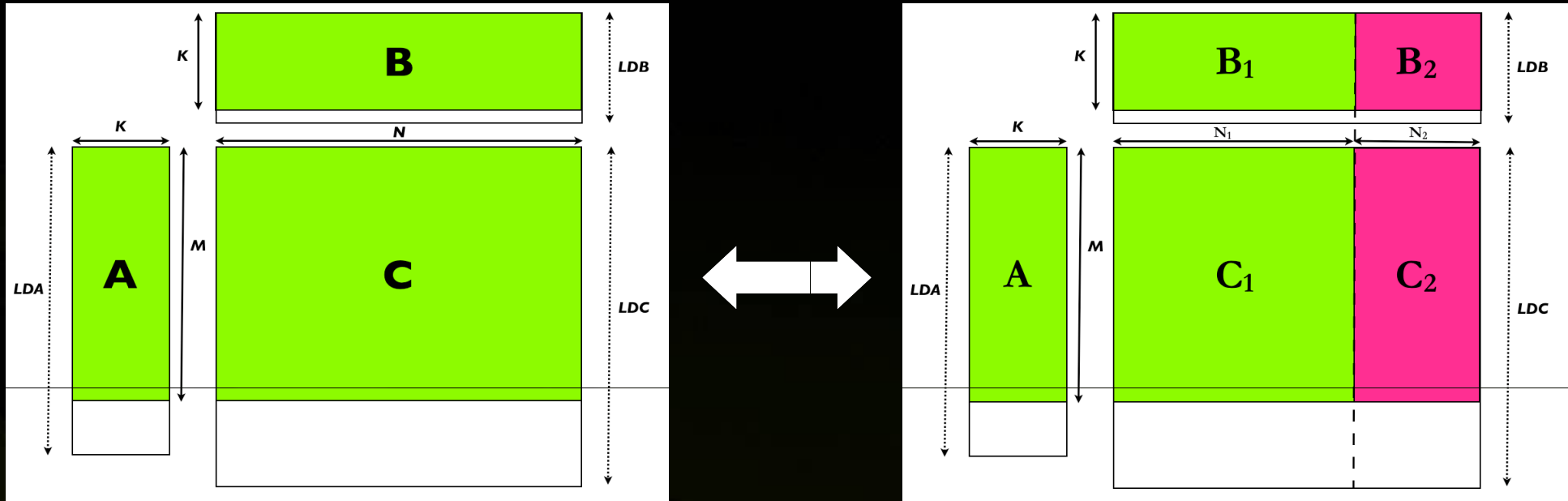
# CUBLAS and DGEMM

# CUBLAS

- **Implementation of BLAS (Basic Linear Algebra Subprograms) on top of CUDA driver**
  - Self-contained at the API level, no direct interaction with CUDA driver
- **Basic model for use**
  - Create matrix and vector objects in GPU memory space
  - Fill objects with data
  - Call sequence of CUBLAS functions
  - Retrieve data from GPU
- **CUBLAS library contains helper functions**
  - Creating and destroying objects in GPU space
  - Writing data to and retrieving data from objects
  - Error handling
- **Function naming convention**
  - cublas + BLAS name: eg., cublasDGEMM
  - Easier to mix CPU and GPU BLAS calls



# DGEMM: $C = \alpha A B + \beta C$



$$\text{DGEMM}(A, B, C) = \text{DGEMM}(A, B_1, C_1) \cup \text{DGEMM}(A, B_2, C_2)$$

(GPU)                      (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

# Overlap DGEMM on CPU and GPU

Copy A from CPU memory to GPU memory devA

```
status = cublasSetMatrix (m, k, sizeof(A[0]), A, lda, devA, m_gpu);
```

Copy B1 from CPU memory to GPU memory devB

```
status = cublasSetMatrix (k, n_gpu, sizeof(B[0]), B, ldb, devB, k_gpu);
```

Copy C1 from CPU memory to GPU memory devC

```
status = cublasSetMatrix (m, n_gpu, sizeof(C[0]), C, ldc, devC, m_gpu);
```

Perform DGEMM(devA,devB,devC) on GPU

Control immediately return to CPU

```
cublasDgemm('n', 'n', m, n_gpu, k, alpha, devA, m, devB, k, beta, devC, m);
```

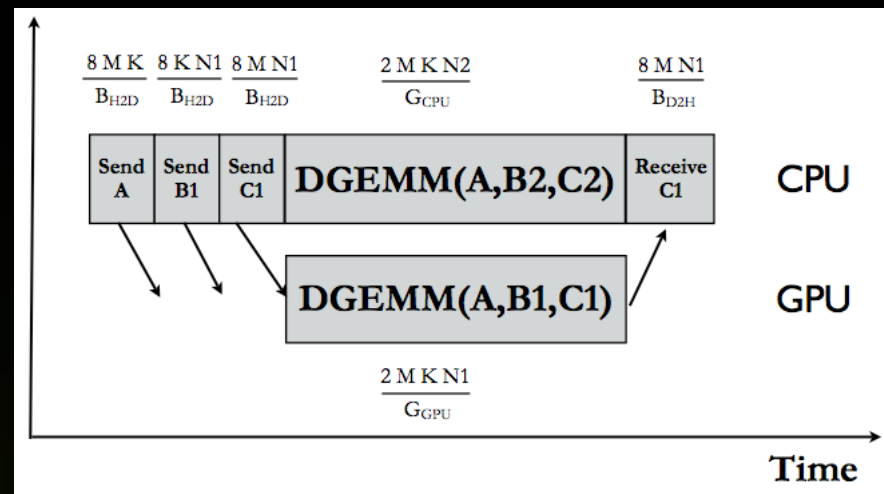
Perform DGEMM(A,B2,C2) on CPU

```
dgemm('n','n',m,n_cpu,k, alpha, A, lda,B+ldb*n_gpu, ldb, beta,C+ldc*n_gpu, ldc);
```

Copy devC from GPU memory to CPU memory C1

```
status = cublasGetMatrix (m, n, sizeof(C[0]), devC, m, C, *ldc);
```

Using CUBLAS, it is very easy to express the workflow in the diagram



# DGEMM performance on GPU



A DGEMM call in CUBLAS maps to several different kernels depending on the size of the matrices. With the combined CPU/GPU approach, we can always send optimal work to the GPU.

M	K	N	M%64	K%16	N%16	Gflops
448	400	12320	Y	Y	Y	82.4
12320	400	1600	N	Y	Y	75.2
12320	300	448	N	N	Y	55.9
12320	300	300	N	N	N	55.9

Tesla T10 1.44Ghz, data resident in GPU memory. Optimal kernel achieves 95% of peak

# Optimal split

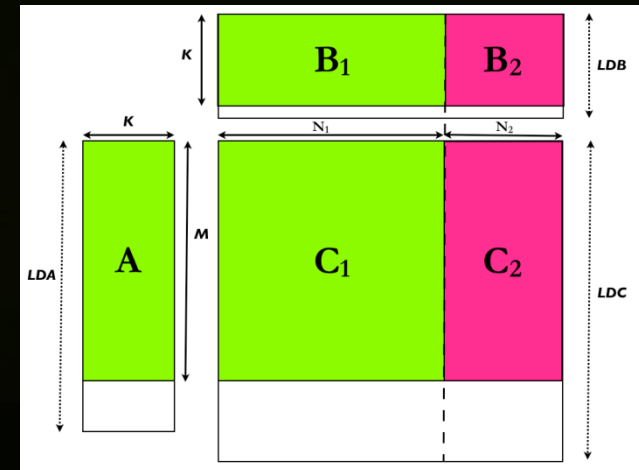


If  $A(M,K)$ ,  $B(K,N)$  and  $C(M,N)$ , a DGEMM call performs  $2*M*K*N$  operations

$$T_{\text{CPU}}(M,K,N_2) = T_{\text{GPU}}(M,k,N_1) \quad N=N_1+N_2$$

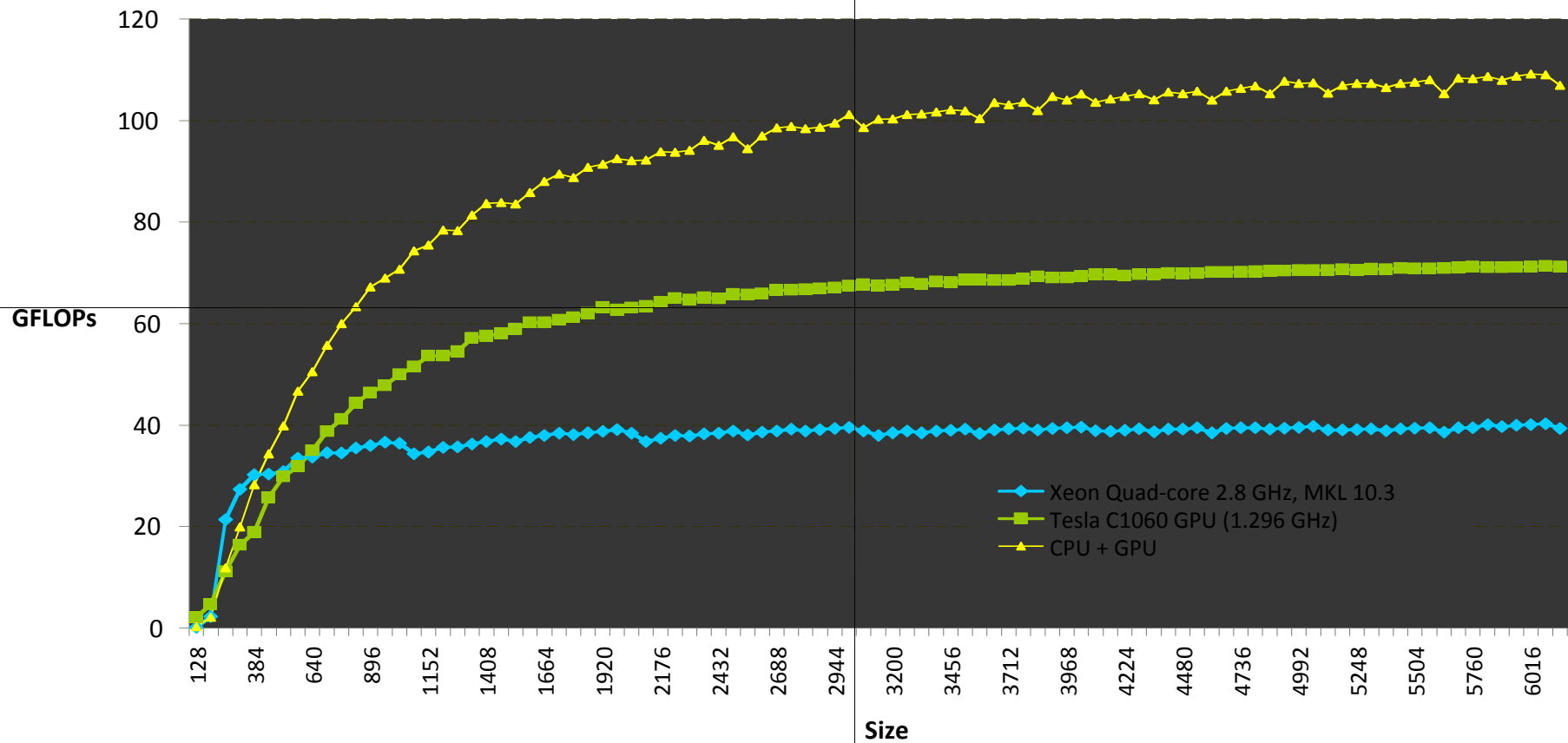
If  $G_{\text{CPU}}$  denotes the DGEMM performance of the CPU in Gflops and  $G_{\text{GPU}}$  the one of the GPU,  
The optimal split is

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





# DGEMM Performance



# Results



## Results on workstation

SUN Ultra 24 workstation with an Intel Core2 Extreme Q6850 (3.0Ghz) CPU, 8GB of memory plus a Tesla C1060 (1.296Ghz) card.

- Peak DP CPU performance of 48 GFlops
- Peak DP GPU performance of 77 GFlops (60\*clock)

T/V	N	NB	P	Q	Time	Gflops
WR00L2L2	23040	960	1	1	97.91	<b>8.328e+01</b>
$\ Ax-b\ _{\infty}/(\epsilon * (\ A\ _{\infty} * \ x\ _{\infty} + \ b\ _{\infty}) * N) =$						0.0048141 ..... PASSED

83.2 Gflops sustained on a problem size using less than 4GB of memory: 66% of efficiency

T/V	N	NB	P	Q	Time	Gflops
WR00C2R2	32320	1152	1	1	246.86	<b>9.118e+01</b>
$\ Ax-b\ _{\infty}/(\epsilon * (\ A\ _{\infty} * \ x\ _{\infty} + \ b\ _{\infty}) * N) =$						0.0042150 ..... PASSED

91.1 Gflops sustained on a problem size using 8GB of memory: 73% of efficiency

# PCI-e transfer speed

	SUN Ultra 24 PCI-e x16 gen2		Supermicro 6015TW PCI-e x16 gen2	
	Pageable Memory	Pinned Memory	Pageable Memory	Pinned Memory
Host to Device	<b>2132 MB/s</b>	<b>5212 MB/s</b>	<b>2524 MB/s</b>	<b>5651 MB/s</b>
Device to Host	<b>1882 MB/s</b>	<b>5471 MB/s</b>	<b>2084 MB/s</b>	<b>5301 MB/s</b>

CUDA offers a fast PCI-e transfer when host memory is allocated with `cudaMallocHost` instead of regular `malloc`. Limited to 4GB in CUDA 2.1, no limitations in upcoming CUDA 2.2





T/V	N	NB	P	Q	Time	Gflops
WR00L2L2	23040	960	1	1	97.91	<b>8.328e+01</b>
Ax-b  _oo/(eps*(  A  _oo*  x  _oo+  b  _oo)*N)=					0.0048141 ..... PASSED	

T/V	N	NB	P	Q	Time	Gflops
WR00L2L2	23040	960	1	1	117.06	<b>69.66e+01</b>
Ax-b  _oo/(eps*(  A  _oo*  x  _oo+  b  _oo)*N)=					0.0048141 ..... PASSED	

# Results on cluster

Cluster with 8 nodes, each node connected to half of a Tesla S1070-500 system:

- Each node has 2 Intel Xeon E5462 ( 2.8Ghz with 1600Mhz FSB) , 16GB of memory and 2 GPUs (1.44Ghz clock).
- The nodes are connected with SDR Infiniband.

T/V	N	NB	P	Q	Time	Gflops
WR10L2R4	92164	960	4	4	479.35	1.089e+03
$\ Ax-b\ _{\infty}/(\epsilon \cdot (\ A\ _{\infty} \cdot \ x\ _{\infty} + \ b\ _{\infty}) \cdot N) =$					0.0026439 .....	PASSED

1.089 Tflop/s sustained using less than 4GB of memory per MPI process.

The first system to break the Teraflop barrier was ASCI Red (1.068 Tflop/s) in June 1997 with 7264 Pentium Pro processors. The GPU accelerated nodes occupies 8U ( 11U including Ethernet and Infiniband switches).

## Results on cluster

Increasing the problem size ( 8GB per MPI process, CUDA 2.2 beta):

T/V	N	NB	P	Q	Time	Gflops
WR11R2L2	118144	960	4	4	874.26	1.258e+03
$\ Ax-b\ _{\infty}/(\text{eps}*(\ A\ _{\infty}*\ x\ _{\infty}+\ b\ _{\infty})*N)=$					0.0031157 .....	PASSED

# Results on cluster

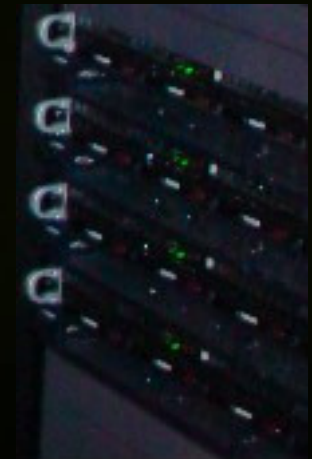


11/30/2008

71

Computer (Full Precision)	Number of Procs or Cores	$R_{max}$ GFlop/s	$N_{max}$ Order	$N_{1/2}$ Order	$R_{Peak}$ GFlop/s
MVS-5000BM Cluster IBM JS20 (dual IBM PowerPC 970 - 1.6 GHz w/Myrinet)	330	1401	280000	45000	2112
Cray X-1 (800 MHz)	120	1400.4	230400	26496	1536.0
IBM xSeries 335 cluster (dual 3.06GHz Xeon w/InfiniBand)	384	1389	120000		2350
IBM eServer pSeries 690 Turbo(1.3 GHz Power 4 w/Colony)	512	1384.0	200000		2662
NEC SX-7/160M5(1.81ns)	160	1378	200000	15200	1412.8
Dell PowerEdge 1850 (Xeon 64 3.2GHz w/Topspin InfiniBand)	256	1349	220440	110220	1638
Intel ASCI Option Red (200 MHz Pentium Pro)	9152	1338.	235000	63000	1830
Legend DeepComp 1800 (2GHz Pentium 4 w/Myrinet)	512	1297	172000		2048
Self Made Pentium4 Xeon (80-3.06 GHz, 72-2.8 GHz, 112-2.4 GHz, 256-2.2 GHz w/GigE)	520	1283	260000		2557
Intel EM64T (2 way 3.2 GHz Intel EM64T w/Myrinet D)	256	1269	241920		1638
HP Integrity rx2600 Itanium2 (1.3 GHz w/Myrinet)	304	1253	256000		1580
IBM eServer (Opteron 2.2 GHz w/Infiniband)	400	1246	200000		1760

1258



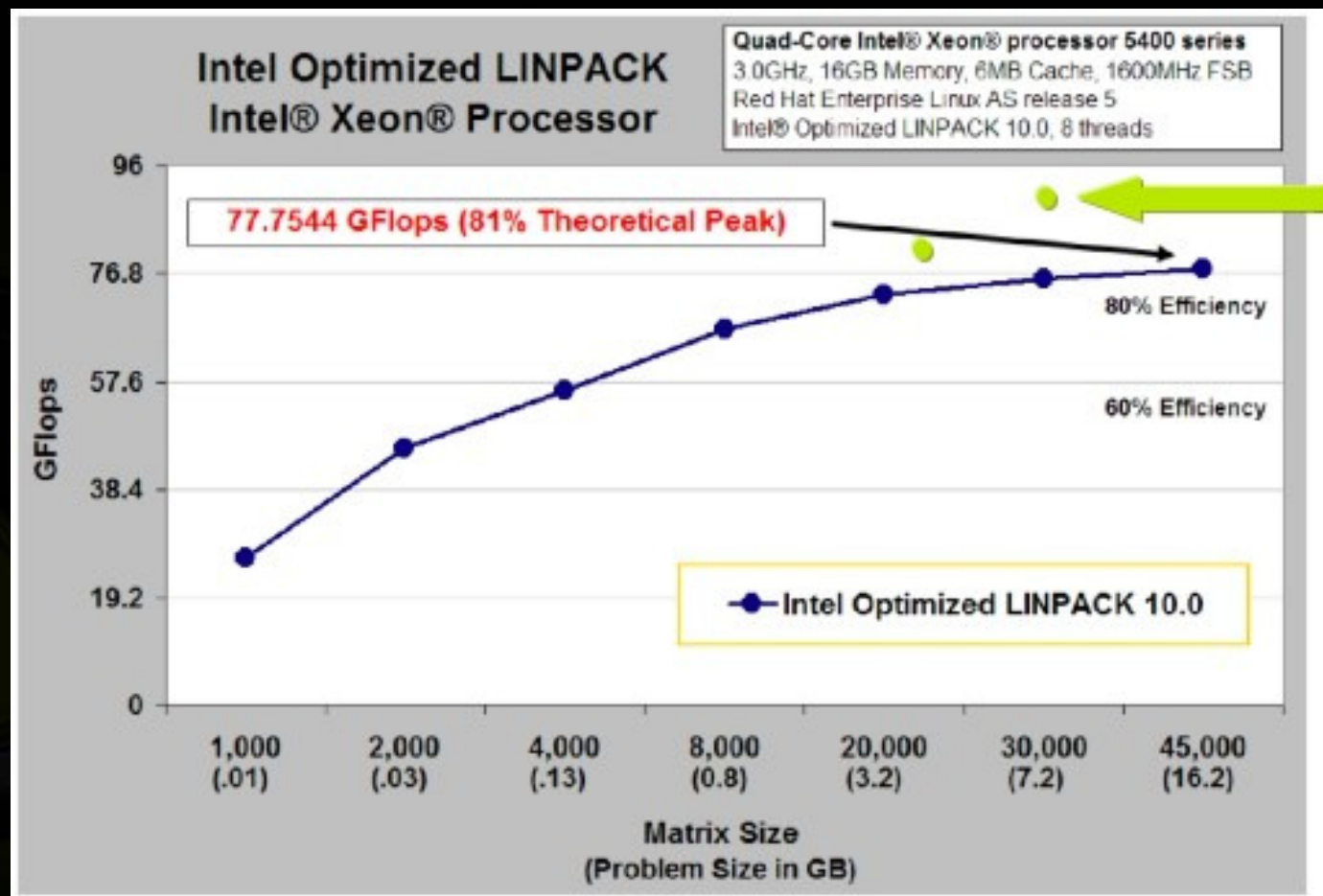


# Conclusions



- 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
- Code is available from NVIDIA

# Results on workstation



1 QC Core 2 and  
1 Tesla C1060 faster  
then 2 Xeon