ICHEC
Irish Centre for High-End Computing

The Practical Reality of Heterogeneous Super Computing

Rob Farber

Visiting scientist at the NVIDIA CUDA Research Center at the Irish Center for High-End Computing (ICHEC)

Science Foundation Ireland

Ireland's EU Structural Funds Programmes 2007 - 2013

Co-funded by the Irish Government and the European Union

EUROPEAN REGIONAL DEVELOPMENT FUND

An Roinn Oideachais agus Eolaíochta | Department of Education and Science

HEA
Higher Education Authority
An tÚdarás um Ard-Oideachas

Outline of the talk

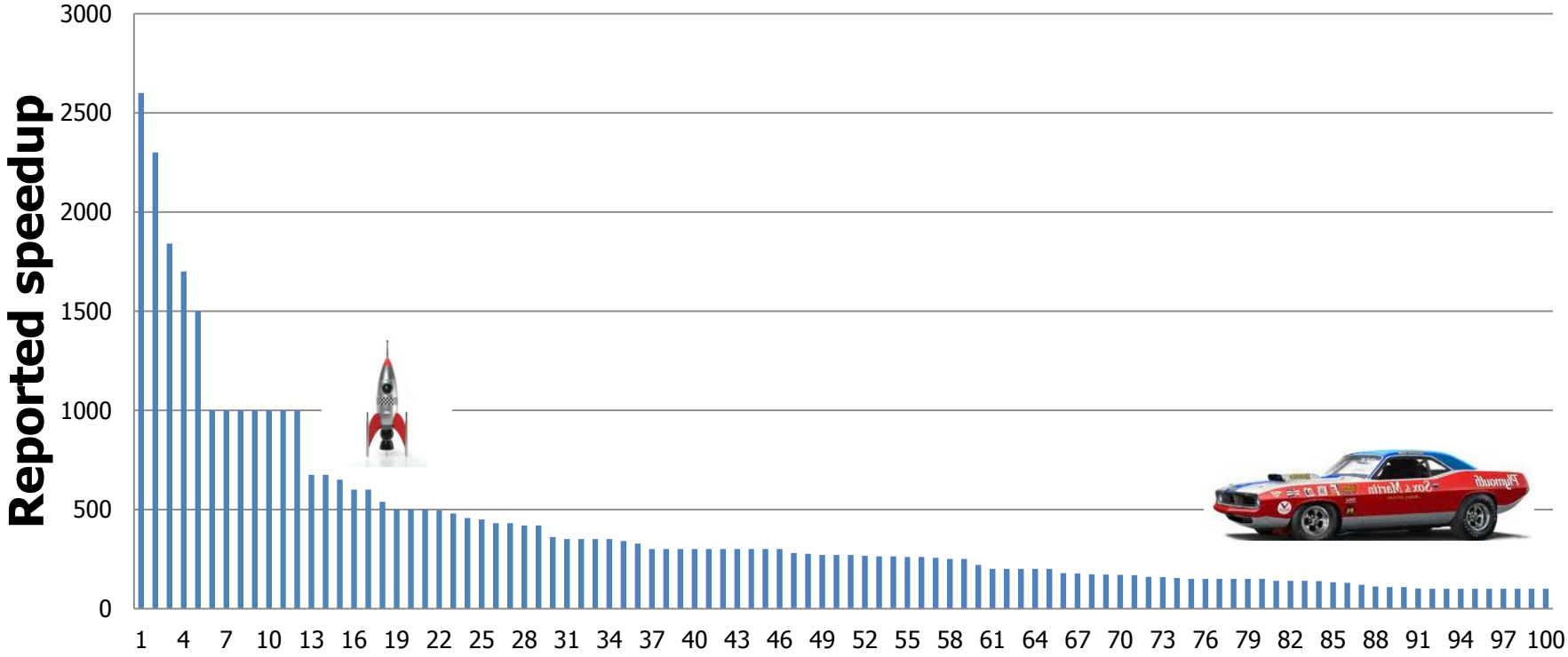
- CUDA is now a language *for all application development* just like C/C++ and Java!
- Strategies for embracing heterogeneous computing
  - Opportunities enabled by CUDA x86
  - Practical ideas for balancing CPU & GPU
  - Practical tips on running CUDA Kernels on CPU cores

The growth of CUDA

- First introduced in February 2007
  - Now taught at 454 institutions world-wide

Performance is the reason for GPUs

Top 100 NVIDIA CUDA application showcase speedups as of May, 9, 2011
(Min 100, Max 2600, Median 1350)



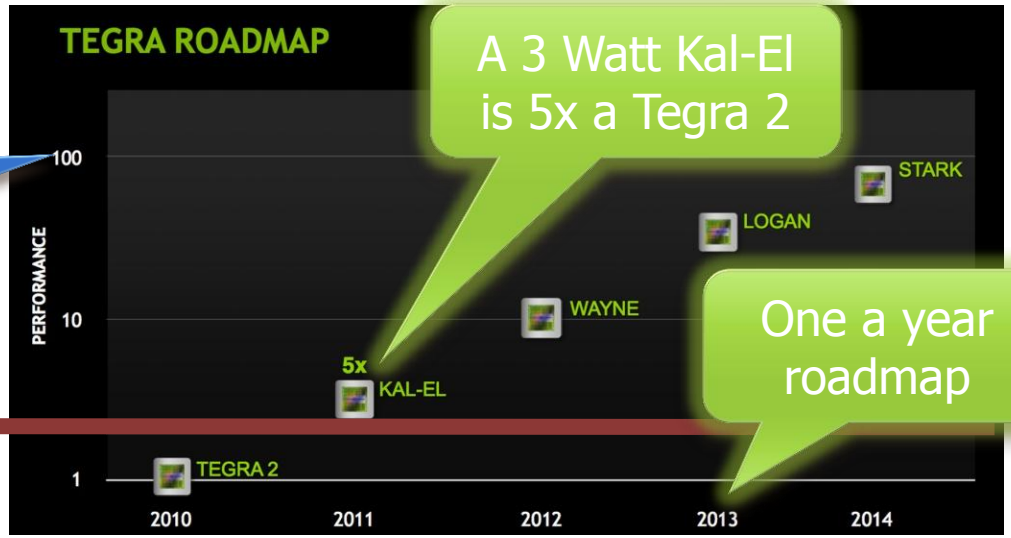
Ranked from highest to lowest speedup

http://developer.nvidia.com/cuda-action-research-apps

Why x86? (Why ARM?) (Why …?)

- Market accessibility:
  - Over ¼ BILLION CUDA-enabled GPUs sold (300M)
  - Small compared to the number of x86 systems.
- ARM is the power behind many super phones
  - What a market segment! (cellphones, tablets, …)

Performance is on a Log scale

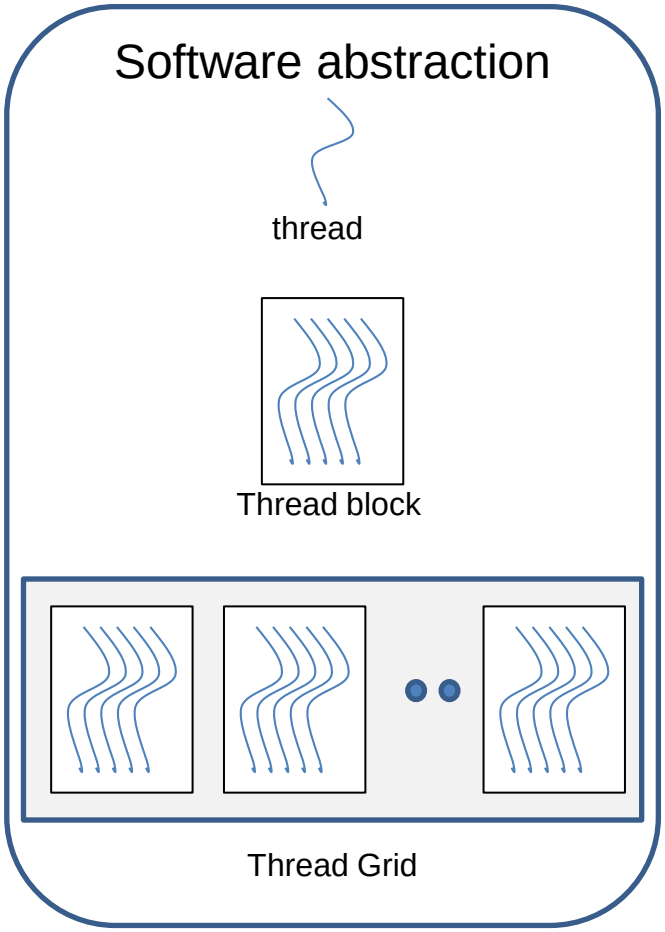


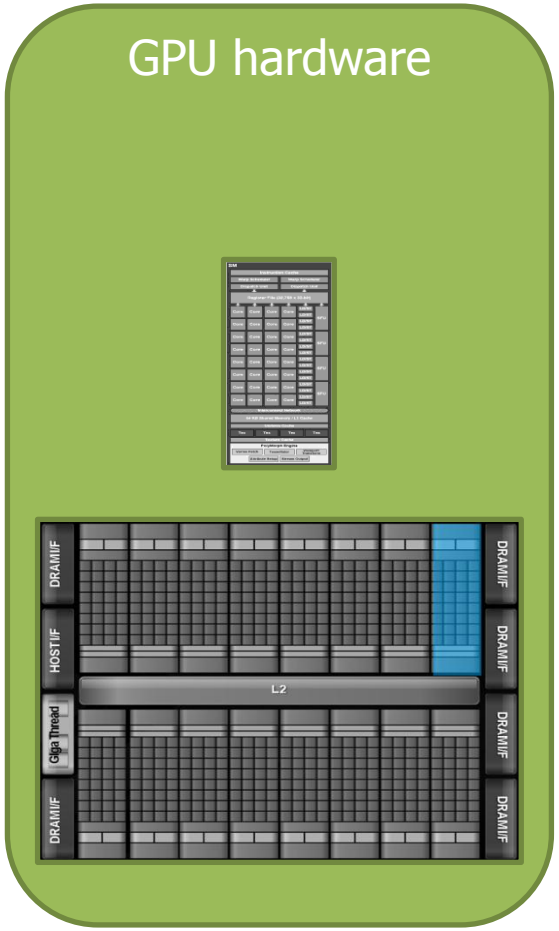
Core2Duo

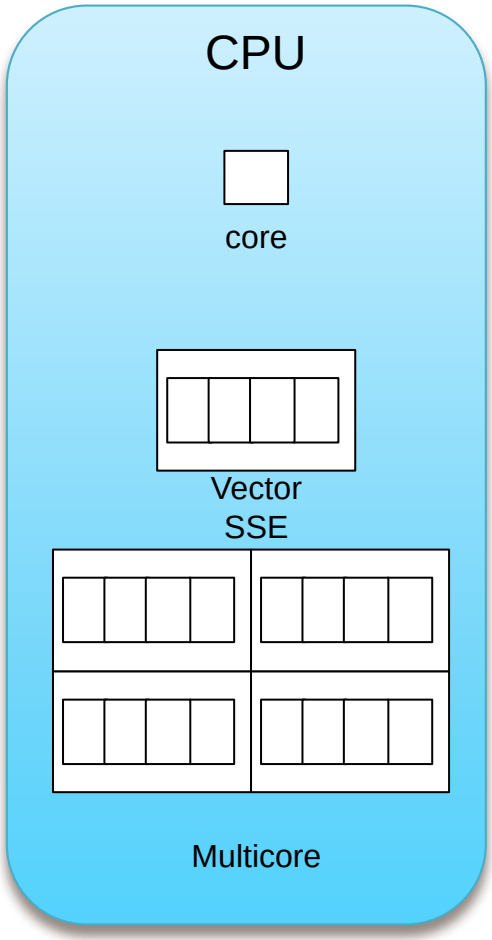
"CUDA is for GPUs and CPUs! "

"One source tree to hold them all and on the GPU accelerate them!" (A parody of J.R.R. Tolkien)

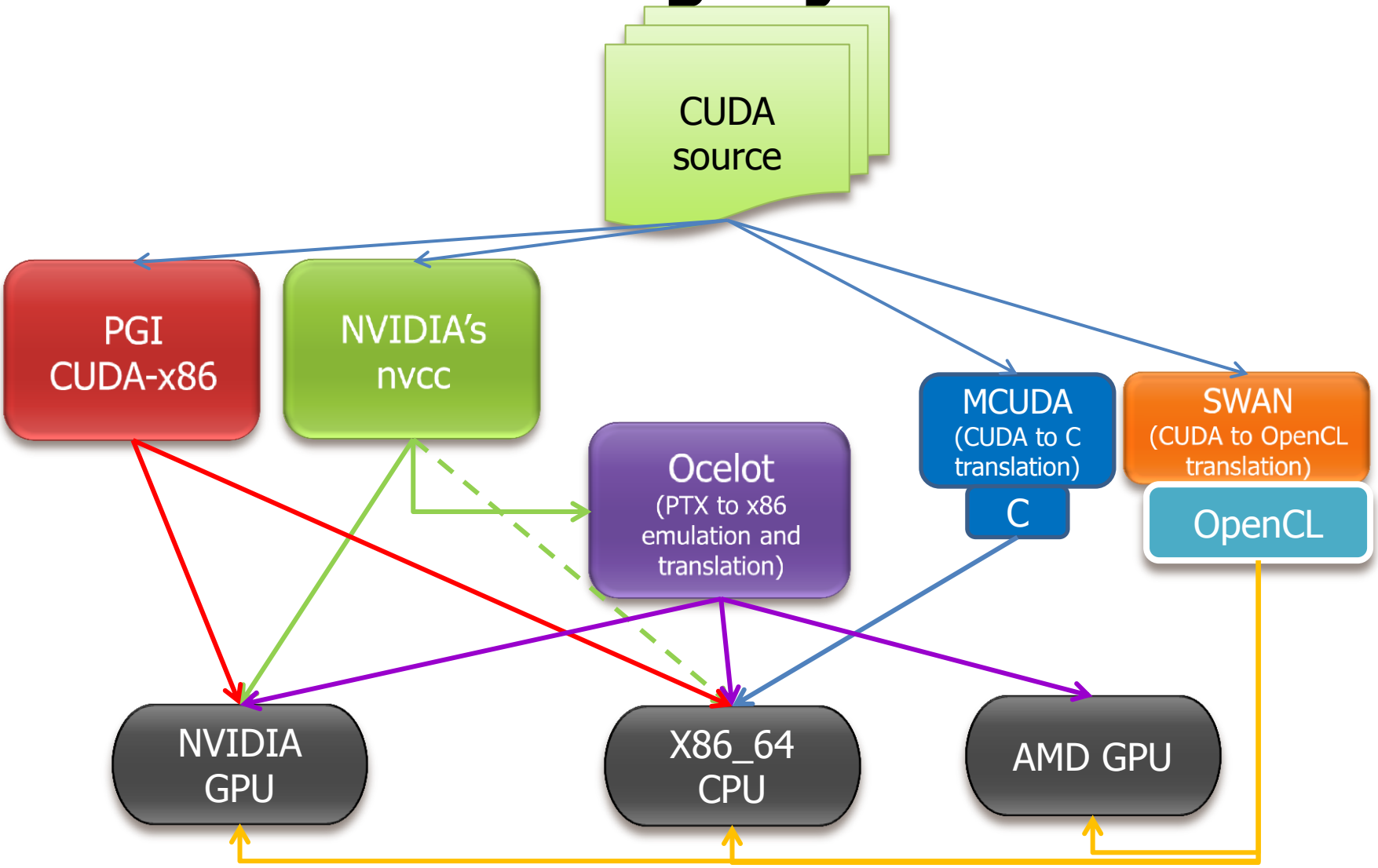
A convergence of concepts (CPU 2-6 cores/GPU hundreds of cores)







CUDA is no longer just for GPUs



PGI deviceQuery on a Xeon e5560

```
CUDA Device Query (Runtime API) version (CUDART static linking)

There is 1 device supporting CUDA

Device 0: "DEVICE EMULATION MODE"
  CUDA Driver Version:                           99.99
  CUDA Runtime Version:                          99.99
  CUDA Capability Major revision number:         9998
  CUDA Capability Minor revision number:         9998
  Total amount of global memory:                 128000000 bytes
  Number of multiprocessors:                     1
  Number of cores:                               0
  Total amount of constant memory:               1021585952 bytes
  Total amount of shared memory per block:       1021586048 bytes
  Total number of registers available per block: 1021585904
  Warp size:                                     1
  Maximum number of threads per block:           1021585920
  Maximum sizes of each dimension of a block:    32767 x 2 x 0
  Maximum sizes of each dimension of a grid:     1021586032 x 32767 x 1021586048
  Maximum memory pitch:                          4206313 bytes
  Texture alignment:                             1021585952 bytes
  Clock rate:                                    0.00 GHz
  Concurrent copy and execution:                 Yes
  Run time limit on kernels:                     Yes
  Integrated:                                    No
  Support host page-locked memory mapping:       Yes
  Compute mode:                                  Unknown
  Concurrent kernel execution:                   Yes
  Device has ECC support enabled:                Yes

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 99.99, CUDA Runtime Version =
99.99, NumDevs = 1, Device = DEVICE EMULATION MODE


PASSED

Press <Enter> to Quit...
------------------------------------------------------------
```

PGI running *arrayReversal_multiblock_fast.cu* from Part 3 of my DDJ tutorials

```
$ pgCC arrayReversal_multiblock_fast.cu
$ ./a.out
Correct!
```

It just compiles and runs:

- Boring from a presentation point of view.
- **Exciting from an opportunity point of view.**

PGI to ship a unified binary in 2012

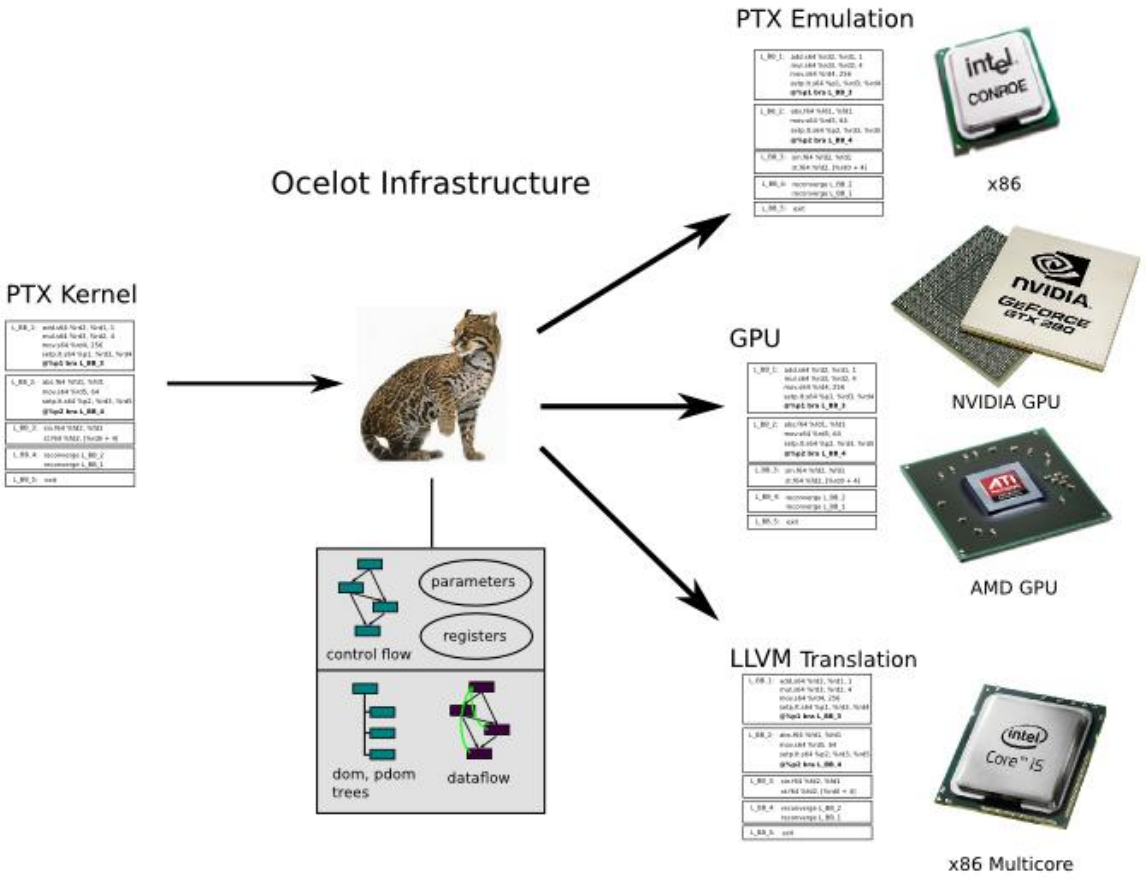
Developer

Customer

Unified binary

X86

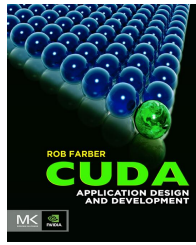
Ocelot runs CUDA binaries



- Must install Ocelot
- Offers a lot more than just x86
  - Profiling/hot-spotting
- Not a turn-key system!

Thrust: CUDA made simple

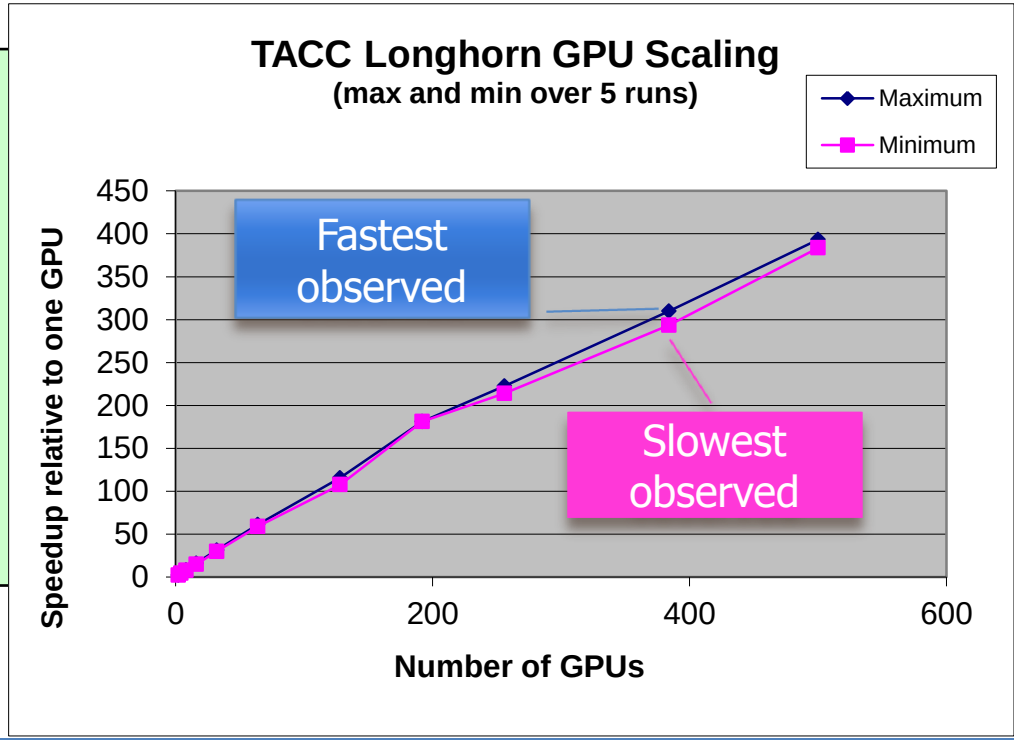
- Most of the actual code from an example that scales to 500 GPUs and delivers 341-times speedup over a single-core (32-bit) Xeon CPU




```
…

FcnOfInterest objFcn(input);

energy = thrust::transform_reduce(
        thrust::counting_iterator<int>(0),
        thrust::counting_iterator<int>(nExamples),
                        objFcn,
                        0.0f,
                        thrust::plus<Real>());
```



Functors can run on both the host and device

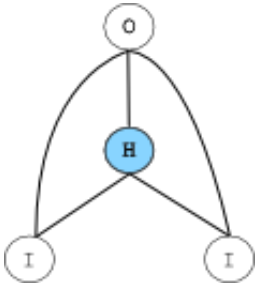
__device__ __host__

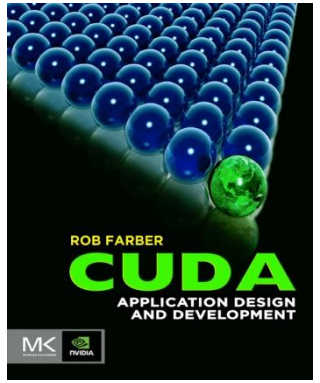
```
Real operator()(unsigned int tid)
{
  const register Real* in = &examples[tid * exLen];
  register int index=0;
  register Real h1 = p[index++];
  register Real o = p[index++];

  h1 += in[0] * p[index++];
  h1 += in[1] * p[index++];
  h1 = G(h1);

  o += in[0] * p[index++];
  o += in[1] * p[index++];
  o += h1 * p[index++];

  // calculate the square of the diffs
  o -= in[nInput];
  return o * o;
}
```





Thrust can use an OpenMP backend

```
nvcc -O2 -o monte_carlo monte_carlo.cu -Xcompiler -fopenmp \
                -DTHRUST_DEVICE_BACKEND=THRUST_DEVICE_BACKEND_OMP \
                -lcudart -lgomp
```

Device	seconds
GPU	0.222
4 OpenMP threads	2.090
2 OpenMP threads	4.168
1 OpenMP thread	8.333

- Timing reported on the Thrust website show that the performance is acceptable
- **Be aware that Thrust is not optimized to produce the best x86 runtimes**

Strategies for embracing heterogeneous computing.

- Opportunities enabled by CUDA x86
- Practical ideas for balancing CPU & GPU
- Practical tips on running CUDA Kernels on CPU cores

The *one CUDA source tree* rationale (aside from saving development $)

- **Fast**: A compiler can perform optimizations that a PTX-based system like Ocelot will miss
  - Please prove me wrong!
- **Transparent**: Both NVIDIA and PGI state that even CUDA applications utilizing proprietary features of the GPU texture units will exhibit identical behavior on both x86 and GPU hardware
  - Please don't prove me wrong!
- **Convenient**: ship one binary to customers for GPU and x86

Reasons for CUDA for all apps

1. **Not much of a change for many applications and organizations**
   a. CUDA is based on standard C and C++
   b. Both of these languages have a solid history of application development spanning decades
2. **Makes applications faster**
   a. CUDA gives the programmer the ability to better exploit parallelism
   b. Exploit the SIMD parallelism in the AVX or SSE instruction in each x86 core

Reasons for CUDA for all apps

3. **Helps to avoid parallel bugs**:
   a. The CUDA execution model precludes common parallel programming errors including race conditions and deadlock
      - Programmer still has to update shared memory correctly
4. **A growing tool ecosystem**
   a. cuda-gdb/Parallel Nsight can debug massively parallel apps with large # of concurrent operations
   b. NVIDIA: Parallel Nsight, computefprof
   c. Others: TAU/PAPI profiler, Ocelot

Reasons for CUDA for all apps

5. **Scalability of the model**:
   a. 100k threads = no big deal, (1M threads = …), (…)
   b. Save future software development dollars and allow fast penetration into new markets and technology platforms
6. **GPU acceleration comes for free**
   a. Opens the door for order of magnitude application acceleration
   b. Expands market reach to the ¼ billion CUDA-enabled GPUs that have been sold worldwide
   c. Future-proofs applications

Reasons for CUDA for all apps

7. There are many CUDA developers

a. This developer base is rapidly expanding

b. CUDA is currently taught at over 454 universities and colleges worldwide -> also rapidly expanding

- ICHEC is in the final stages of becoming a CUDA Teaching Center

Strategies for embracing heterogeneous computing

- Opportunities enabled by CUDA x86
- Practical ideas for balancing CPU & GPUs
- Practical tips on running CUDA Kernels on CPU cores

Three rules for fast GPU codes

1. **Get the data on the GPU (and keep it there!)**
   - PCIe x16 v2.0 bus: 8 GiB/s in a single direction
   - Compute 2.0/2.1 GPUs: 140-200 GiB/s

2. **Give the GPU enough work to do**
   - Assume 10 μs latency and 1 TF device
   - Can waste ($10^{-6} * 10^{12}$) = 1M operations

3. **Reuse and locate data to avoid global memory bandwidth bottlenecks**
   - 10^{12} flop hardware delivers 10^{10} flop when global memory limited
   - Can cause a 100x slowdown!

Corollary: Avoid malloc/free!

Data movement still happens on x86
PGI bandwidthTest on a Xeon e5560

```
Running on...

 Device 0: DEVICE EMULATION MODE
 Quick Mode

 Host to Device Bandwidth, 1 Device(s), Paged memory
   Transfer Size (Bytes)      Bandwidth(MB/s)
   33554432                   4152.5

 Device to Host Bandwidth, 1 Device(s), Paged memory
   Transfer Size (Bytes)      Bandwidth(MB/s)
   33554432                   4257.0

 Device to Device Bandwidth, 1 Device(s)
   Transfer Size (Bytes)      Bandwidth(MB/s)
   33554432                   8459.2


[bandwidthTest] - Test results:
PASSED
```

Happens with straight compilation of CUDA codes

- PGI allows x86 programs to just do a pointer assignment

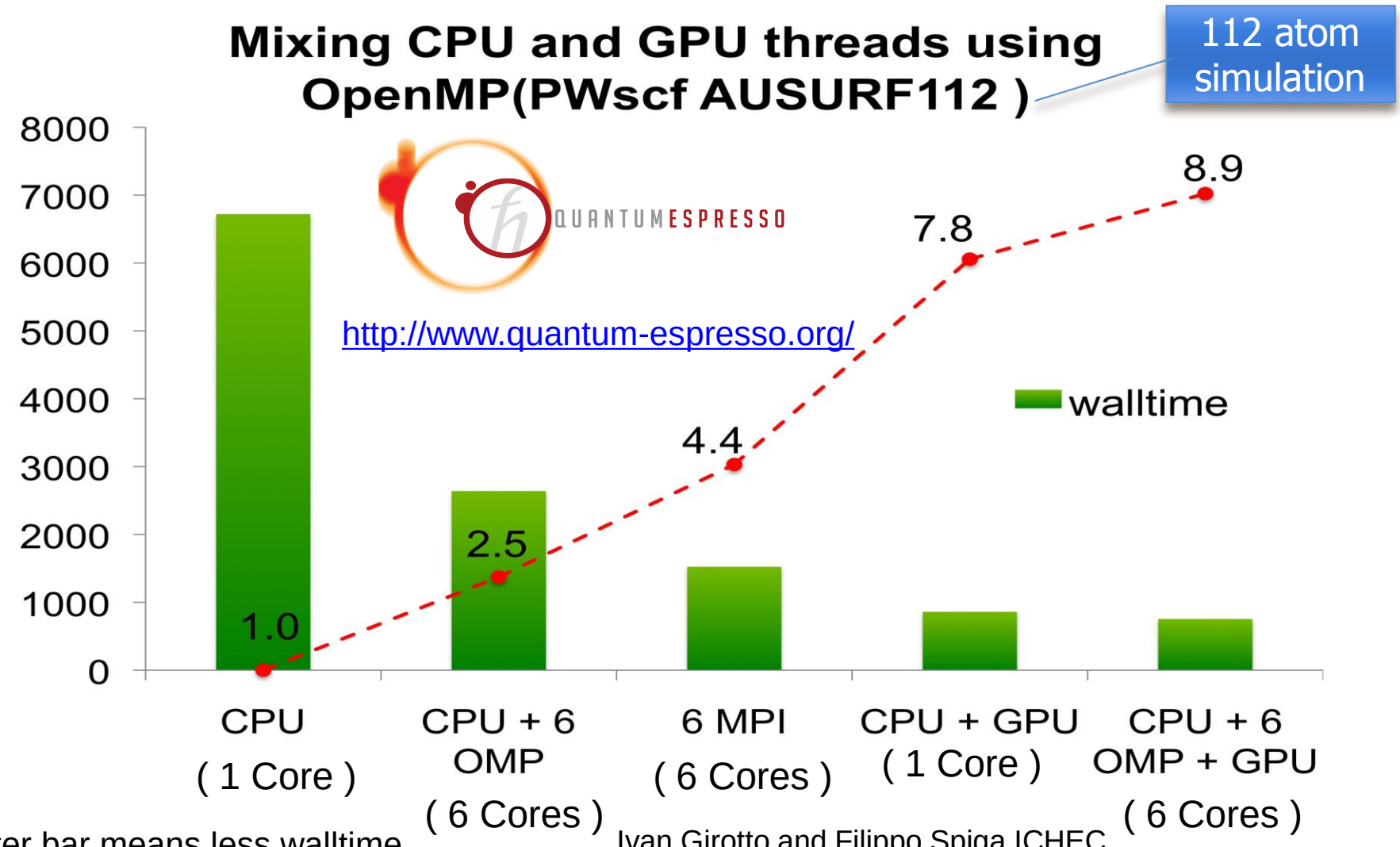
Heterogeneous apps with CUDA libraries

Ivan Girotto and Filippo Spiga ICHEC

- An important example: electronic-structure calculations and materials modeling at the nanoscale
- SCF* calculation on plane wave (64-bit calculations)
- Main bottlenecks
  - 3D FFT (-> CUFFT)
  - Linear Algebra
    - Matrix Matrix Multiplication (-> CUBLAS)
    - Eigenvalues and Eigenvectors (work in-progress)
- First GPU-enabled beta released on May 2011

* **Plane**-**Wave** Gaussian Self-Consistent Field Method

A good start: 8-times speedup over serial



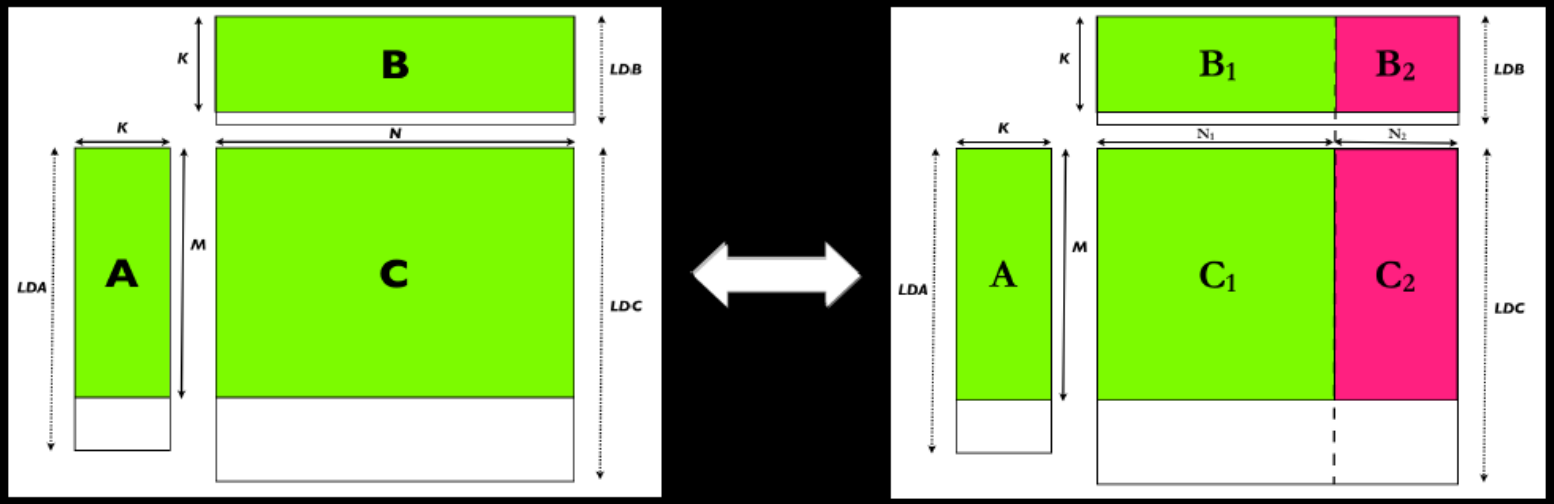
* Shorter bar means less walltime

Existing work Dgemm for Linpack HPL

E. Phillips, et.al.: http://www.nvidia.com/content/GTC-2010/pdfs/2057_GTC2010.pdf

GPU TECHNOLOGY CONFERENCE

DGEMM: C = alpha A B + beta C



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

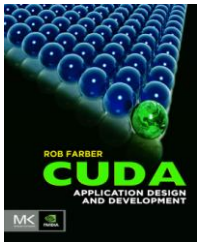
The idea can be extended to multi-GPU configuration and to handle huge matrices

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

PRESENTED BY NVIDIA.

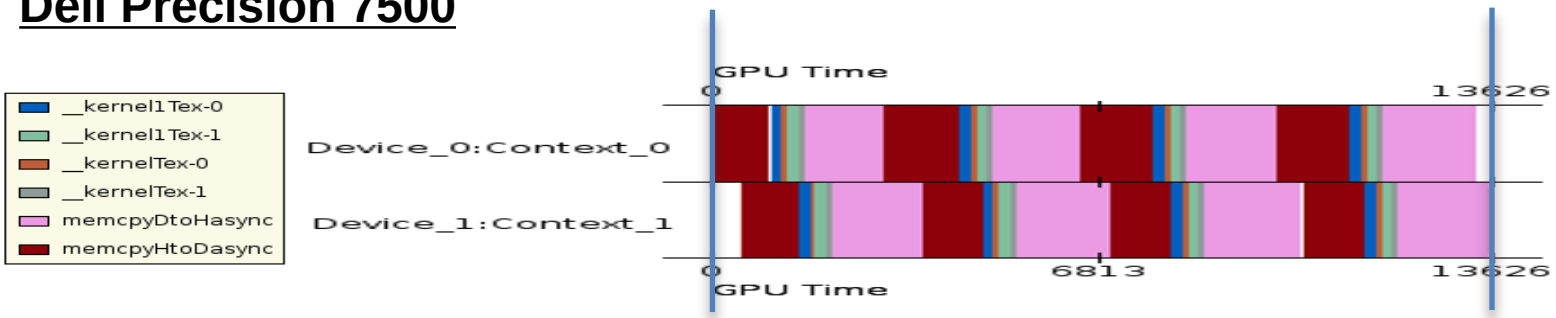
Watch out for PCIe configuration!

(and benchmarkman's ship!)

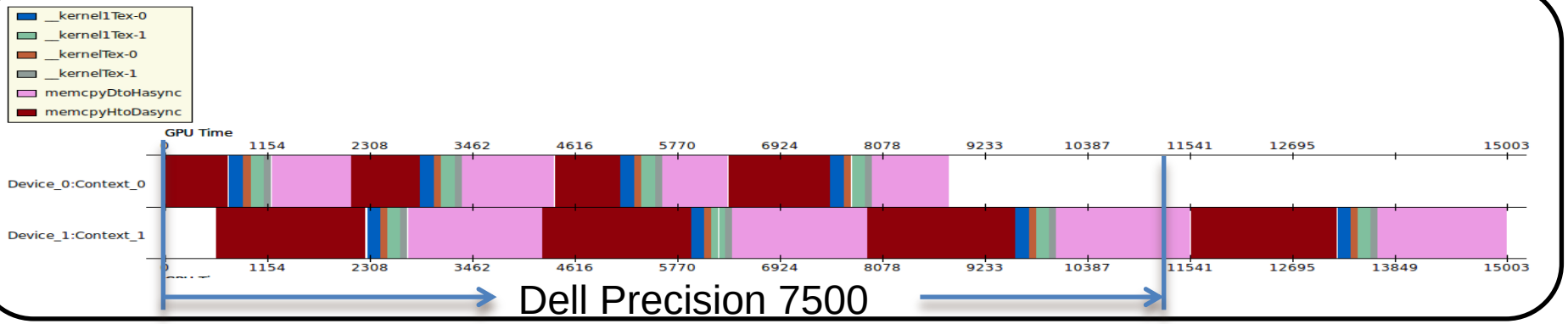


Two GPU CUFFT run (some benchmarks use individual PCIe buses)

Dell Precision 7500



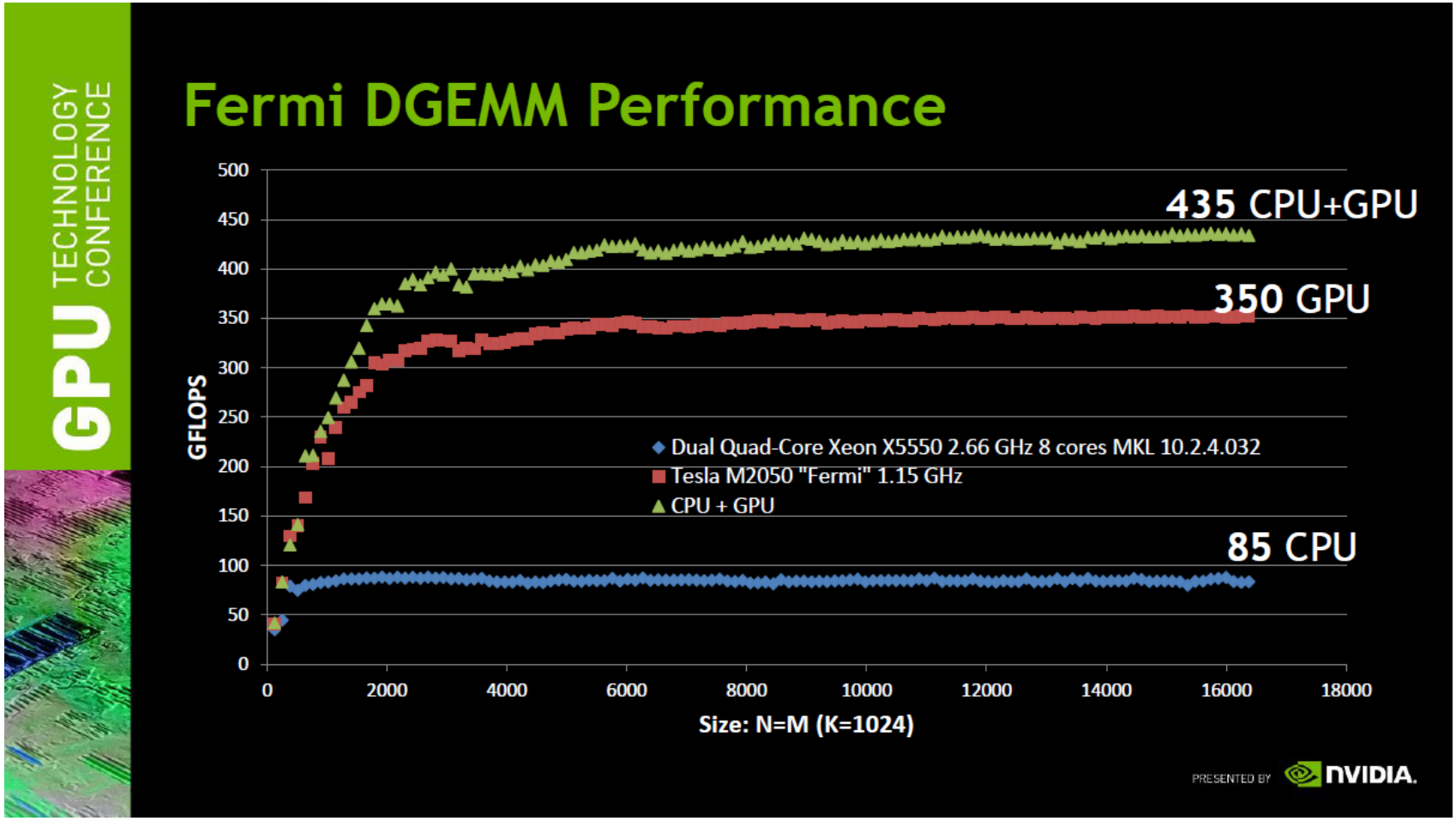
HP z800



Current leader: CPU+GPU 435 GF/s

CPU

Phillips & Fatica.: http://www.nvidia.com/content/GTC-2010/pdfs/2057_GTC2010.pdf



The phiGEMM library from ICHEC

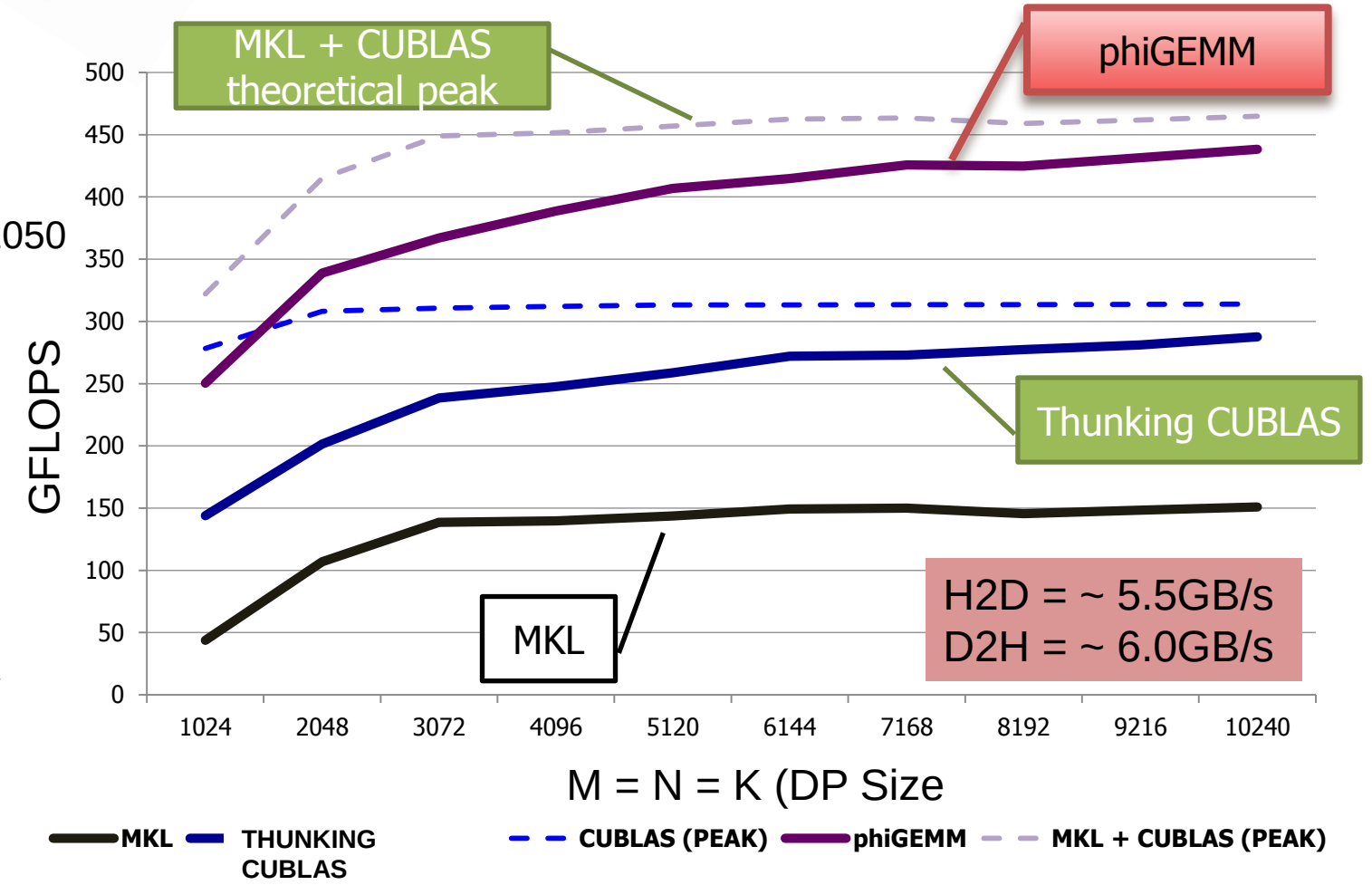
- A library that you use like CUBLAS
  - Transparently manages the thunking operations
  - Supports Sgemm(), Dgemm(), and Zgemm()
  - Asynchronous data transfer (via PINNED Memory)
  - MultiGPU management through single process (CUDA 4.0)
- Evolving: Possible improvements via multi-stream out-of-order execution (see http://www.nvidia.com/content/GTC-2010/pdfs/2057_GTC2010.pdf)
- Written by Girotto and Spiga. Freely downloadable from http://qe-forge.org/projects/phigemm/

phiGEMM matches 1 GPU performance

CPU

2 x Intel
Xeon X5680
3.33GHz
+
NVIDIA Tesla C2050

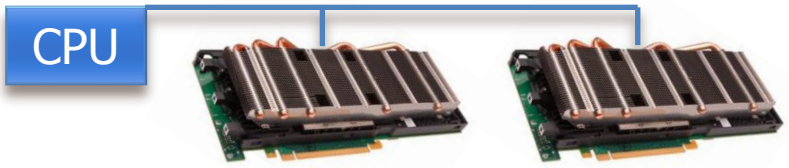
http://qe-forge.org/projects/phigemm/



System provided by

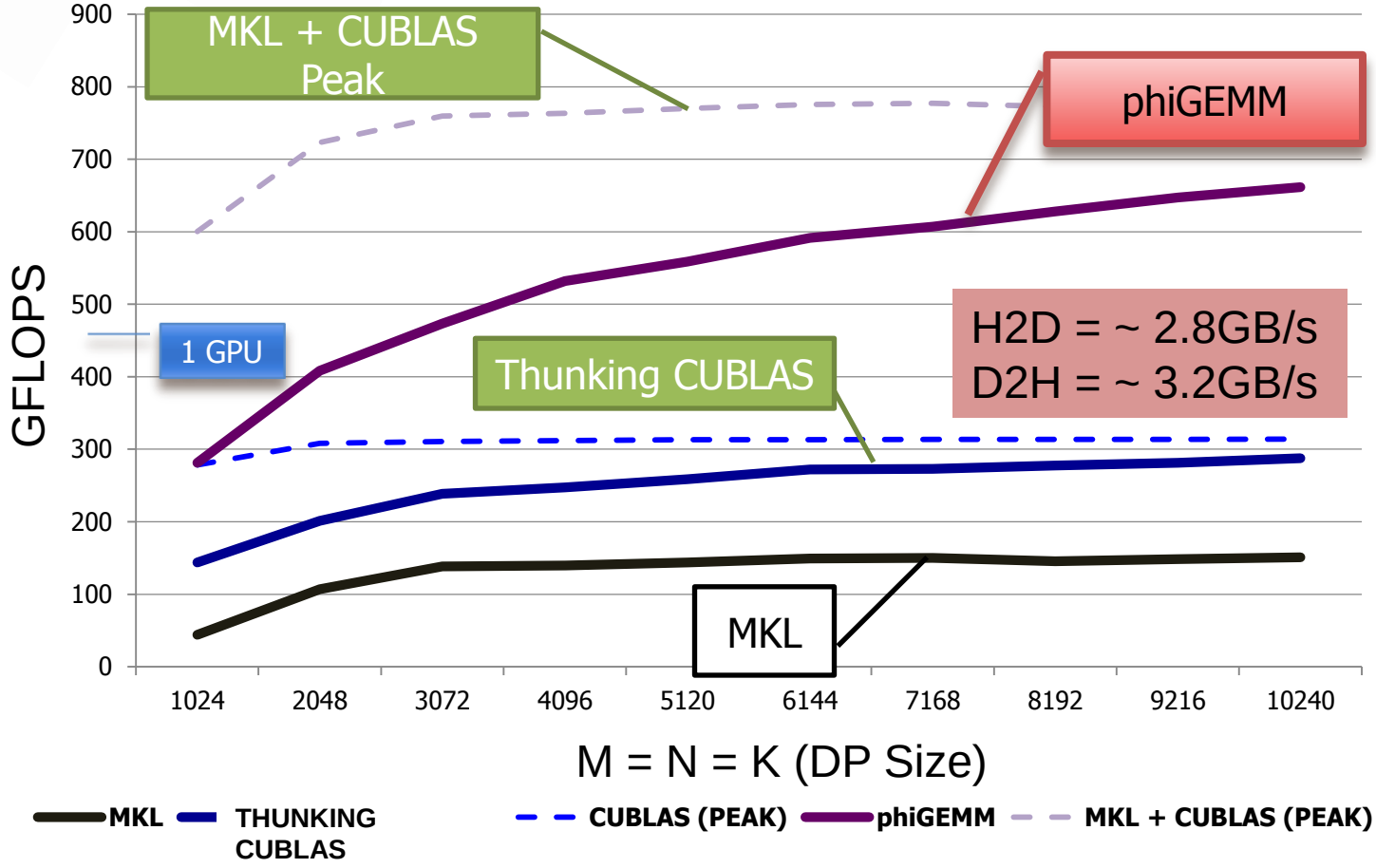
EL COMPUTER ENGINEERING

phiGEMM dual GPU/single bus



http://qe-forge.org/projects/phigemm/

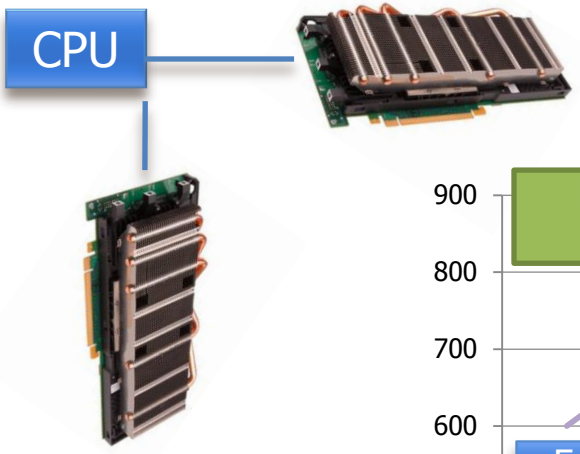
2 x Intel
Xeon X5680
3.33GHz
+
2 NVIDIA Tesla
C2050



System provided by

EL COMPUTER ENGINEERING

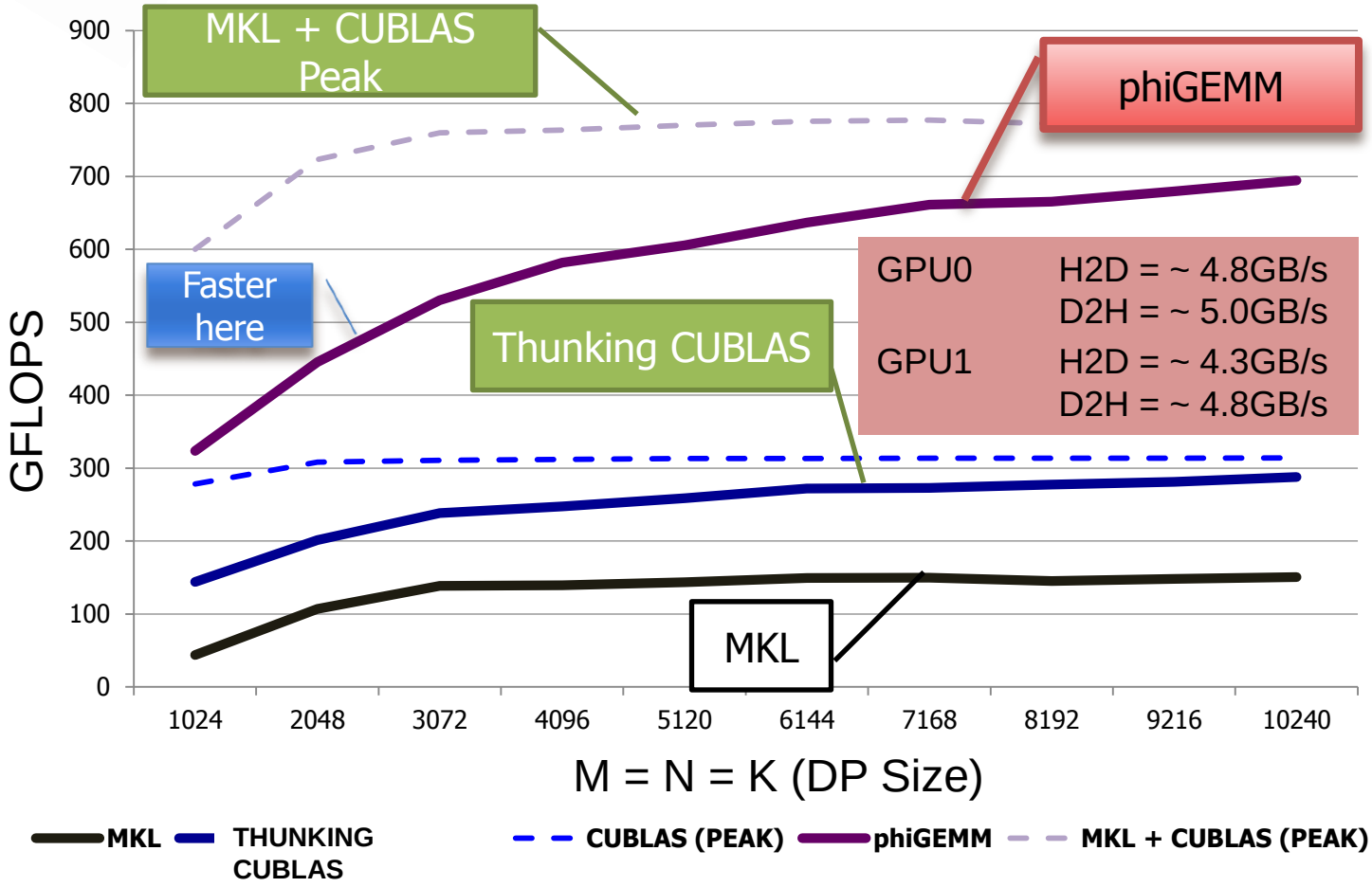
phiGEMM dual GPU/dual bus



2 x Intel
Xeon X5680
3.33GHz
+
2 NVIDIA Tesla
C2050

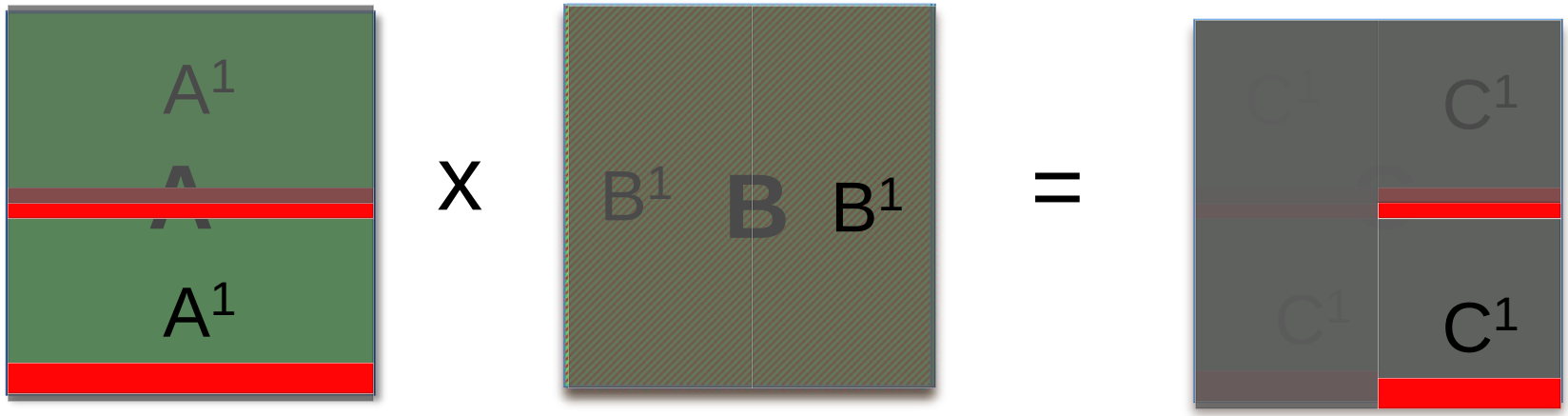
System
provided
by

http://qe-forge.org/projects/phigemm/

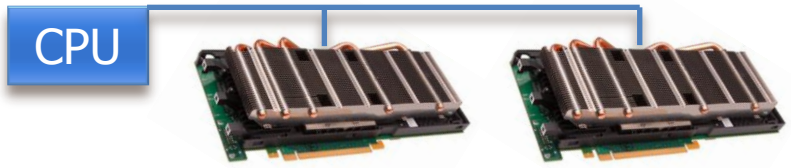


Performance is dependent on problem size

- phiGEMM can run GEMM on matrices that do not fit on a single GPU
- Recursive call to phiGEMM with smaller sub-matrix

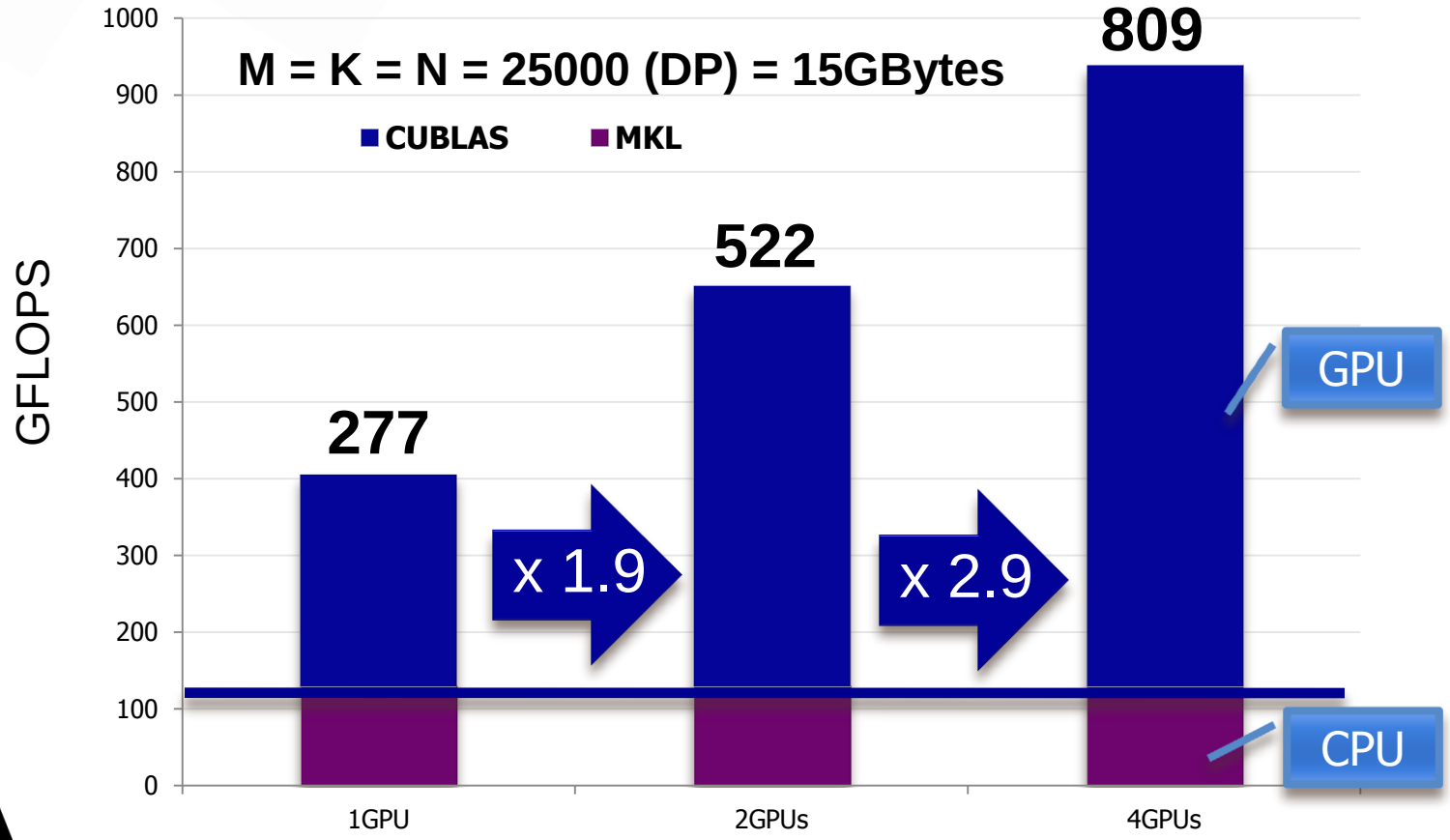


BIG phiGEMM multi GPU/single bus



http://qe-forge.org/projects/phigemm/

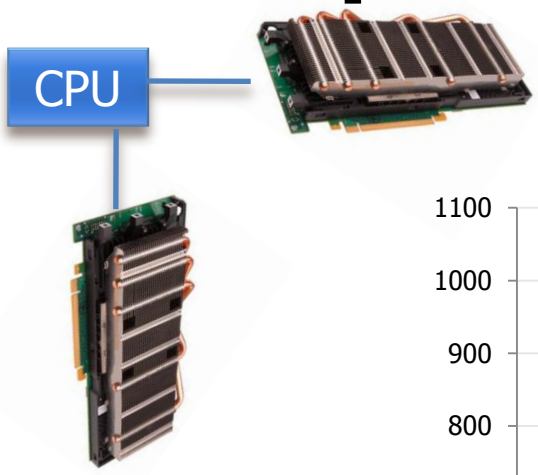
2 x Intel
Xeon X5670
2.93GHz
+
4 NVIDIA Tesla
C2050



System
provided
by

NVIDIA®

BIG phiGEMM multi GPU/dual bus

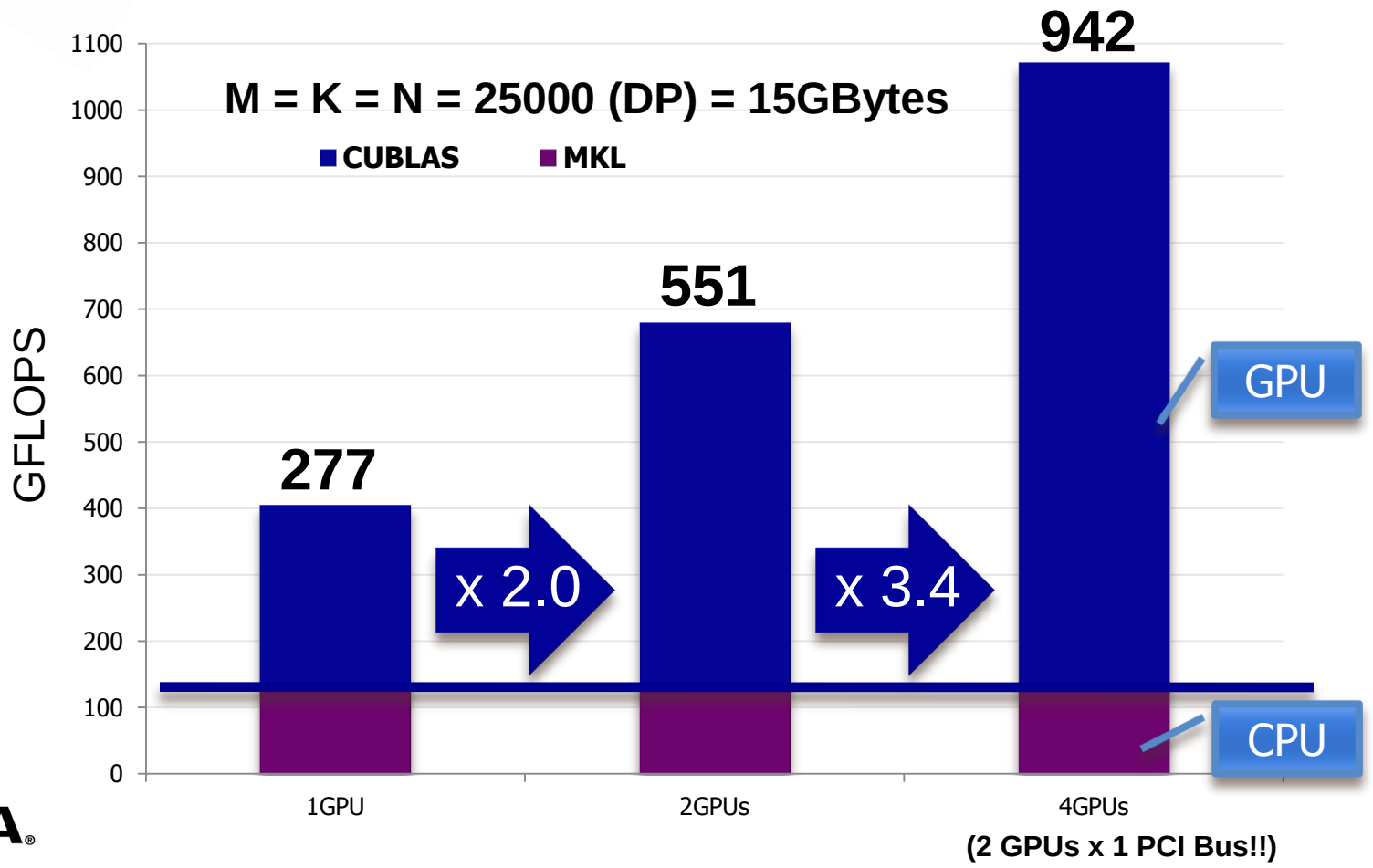


2 x Intel
Xeon X5670
2.93GHz
+
4 NVIDIA Tesla
C2050

System
provided
by

NVIDIA

http://qe-forge.org/projects/phigemm/

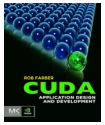


*Gemm operations are compute intensive

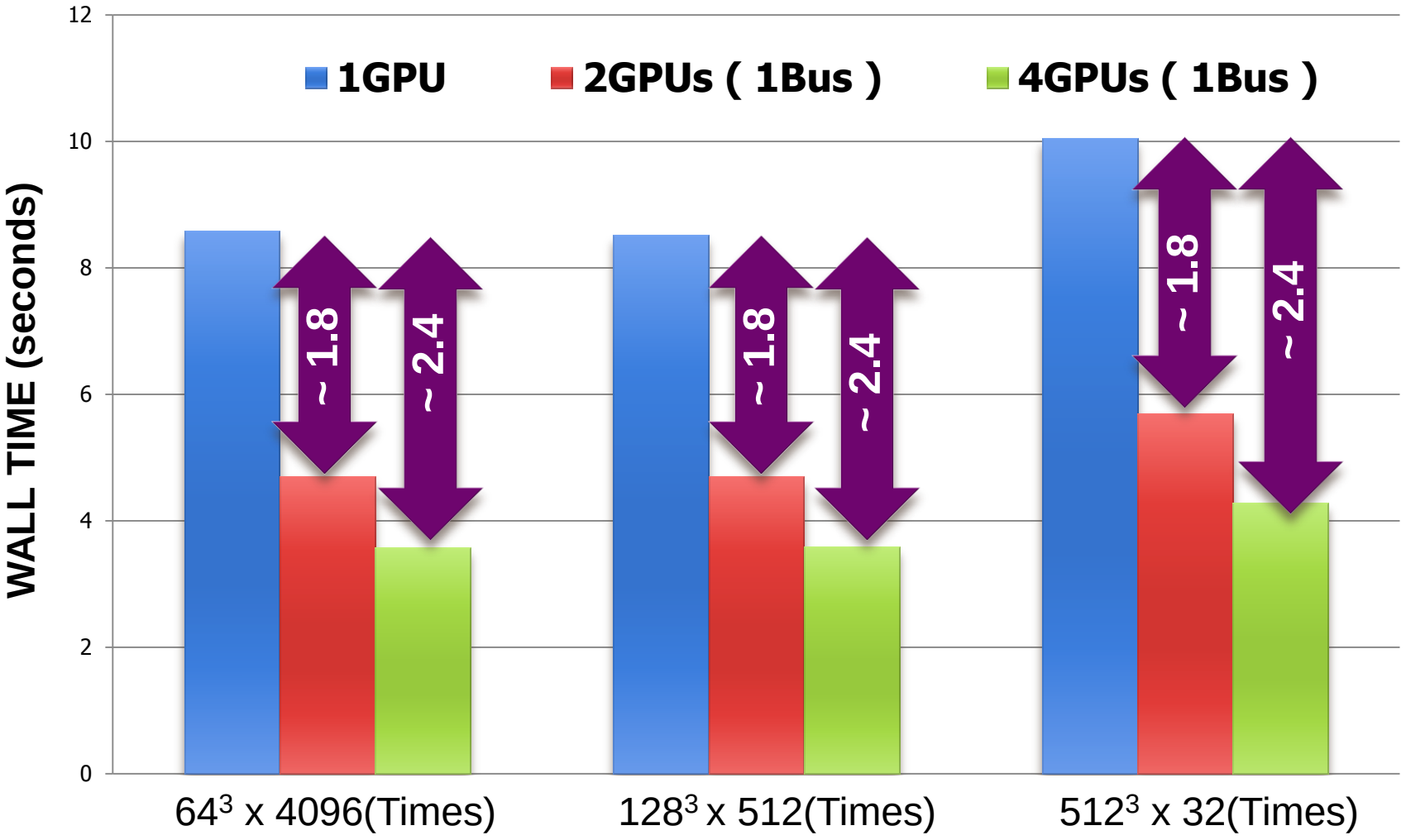
*Gemm is a Level 3 BLAS operation: Work per datum transferred is high **O(N)**

BLAS level	Data	Work	Work per Datum
1	O(N)	O(N)	O(1)
2	$O(N^2)$	$O(N^2)$	O(1)
3	**$O(N^2)$**	**$O(N^3)$**	**O(N)**

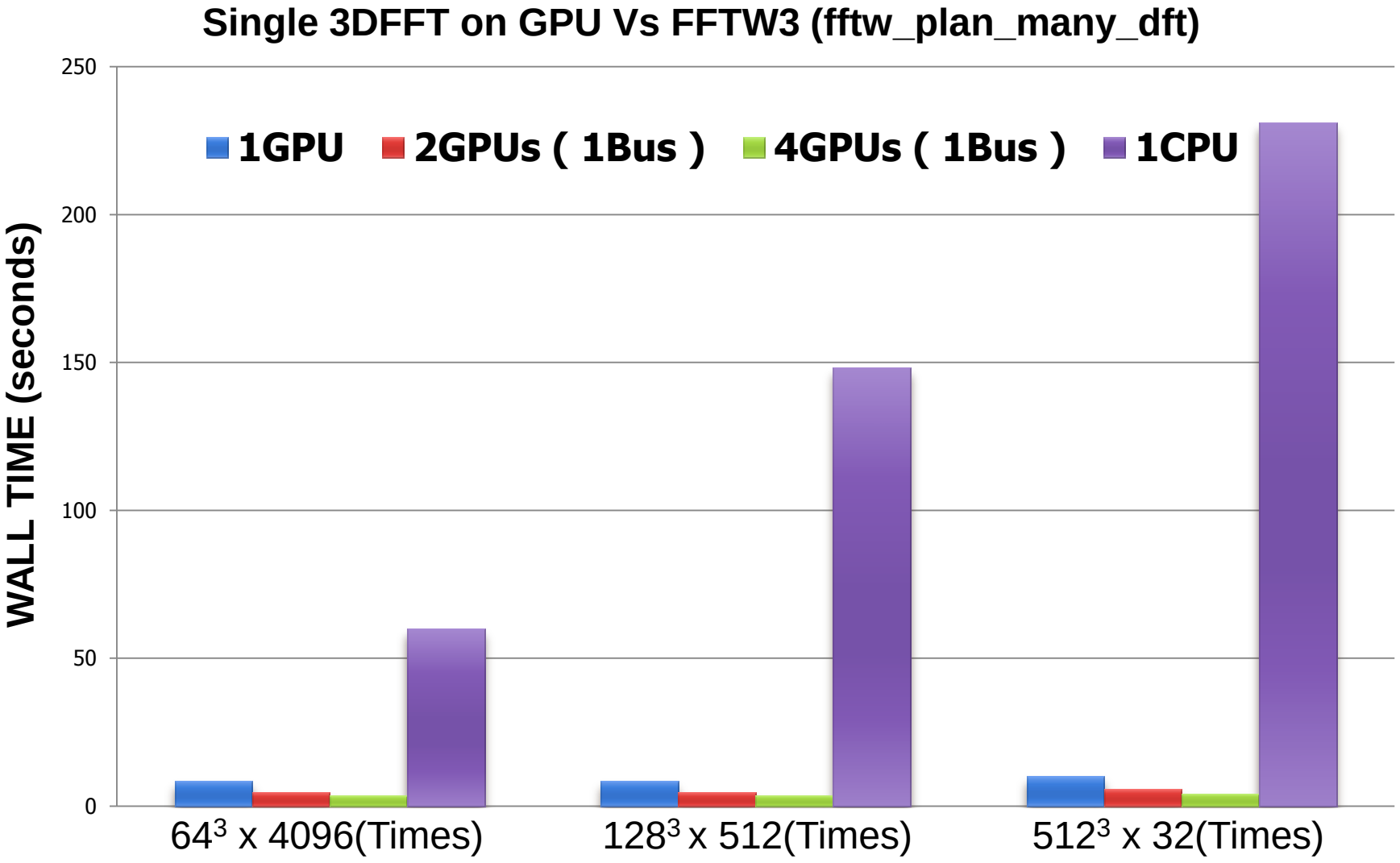
Let's look at a problem that is more dependent on data transfers: 3D FFTs



Performance 3DFFT on multi-GPU



Performance 3DFFT on multi-GPU



Lessons learned

- Watch out for shortcuts with the PCIe bus!
- Thunking can deliver high performance
- Libraries like phiGEMM can make multiGPU/hybrid application development transparent and compatible with libraries like CUBLAS
- I envision a multi/hybrid "smart pointer" to create a non-thunking interface
  - Rule 1: Get the data on the GPU and keep it there

ICHEC contribution to MAGMA

- Like MAGMA, phiGEMM aims "*to design linear algebra algorithms and frameworks for hybrid manycore and GPUs systems that can enable applications to fully exploit the power that each of the hybrid components offers*."
  - Quote from the MAGMA website (http://icl.cs.utk.edu/magma/)
- phiGEMM is under consideration for inclusion in the MAGMA library

Really Exciting! Hybrid Codes

- MAGMA (Matrix Algebra on GPU and Multicore Architectures)
  - "A dense linear algebra library similar to LAPACK but for heterogeneous/hybrid architectures, starting with current "Multicore+GPU" systems." http://icl.cs.utk.edu/magma/
- The MAGMA team has made the conclusion that **dense linear algebra methods are now a better fit on GPU architectures instead of traditional multicore architectures**
  - (Nath, Stanimire, & Dongarra, 2010)
- MAGMA BLAS libraries up to **838 Gflop/s**
  - 33% occupancy and 2 thread blocks per SM (Volkov, 2010)

Strategies for embracing heterogeneous computing.

- Opportunities enabled by CUDA x86
- Practical ideas for balancing CPU & GPU
- Practical tips on running CUDA Kernels on CPU cores

A brand new area where everyone is learning

Do I foresee this as an important topic in the future?

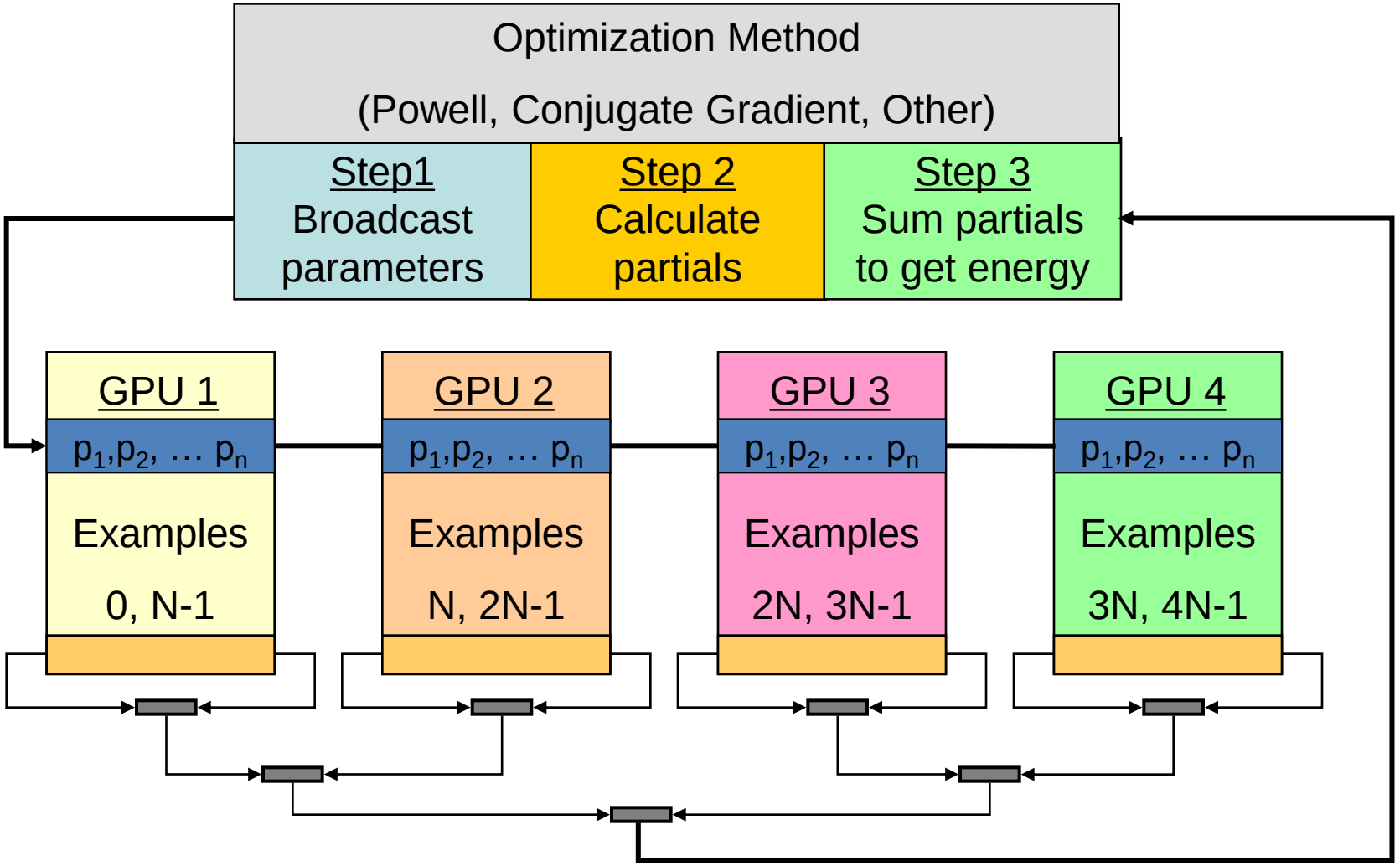
Items of note (slide 1)

- The **size of a warp will be different** from the expected 32 threads per warp for a GPU.
  - For x86 computing a warp might be the size of the SIMD units on the x86 core (either four or eight) or one thread per warp when SIMD execution is not utilized
- **Synchronization is different**: The compiler will remove explicit synchronization of the thread processors when it can determine that it is safe to split loops where the synchronization calls occur

Items of note (slide 2)

- **Still have explicit movement** of data between host and device memory and global to shared memory
  - The PGI compiler allows pointer swapping on x86 systems.
  - Perhaps a wrapper around cudaMemcpy()?
- Watch out for PCIe configuration!
  - Especially for benchmarks that hide poor configurations

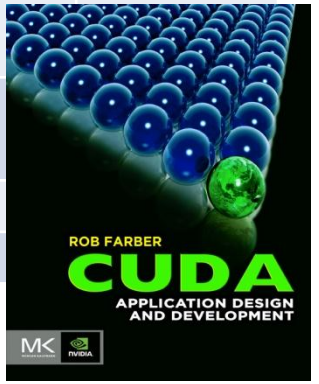
Find a mapping that reuses data *energy = objFunc(p_1, p_2, … p_n)*



Speedup over a quad core

OS	Machine	Opt method	Precision	Ave obj func time	% func time	Speedup over quad-core	Speedup over single-core
Linux	NVIDIA C2070	Nelder-Mead	32	0.00532	100.0	85	341
Win7	NVIDIA C2070	Nelder-Mead	32	0.00566	100.0	81	323
Linux	NVIDIA GTX280	Nelder-Mead	32	0.01109	99.2	41	163
Linux	NVIDIA C2070	Nelder-Mead	64	0.01364	100.0	40	158
Win7	NVIDIA C2070	Nelder-Mead	64	0.01612	100.0	22	87
Linux	NVIDIA C2070	Levenberg-Marquardt	32	0.04313	2.7	10	38
Linux	NVIDIA C2070	Levenberg-Marquardt	64	0.08480	4.4	6	23
Linux	Intel e5630	Levenberg-					
Linux	Intel e5630						
Linux	Intel e5630						
Linux	Intel e5630						

```
#pragma omp parallel for reduction(+ : sum)
    for(int i=0; i < nExamples; ++i) {
      Real d = getError(i);
      sum += d;
    }
```

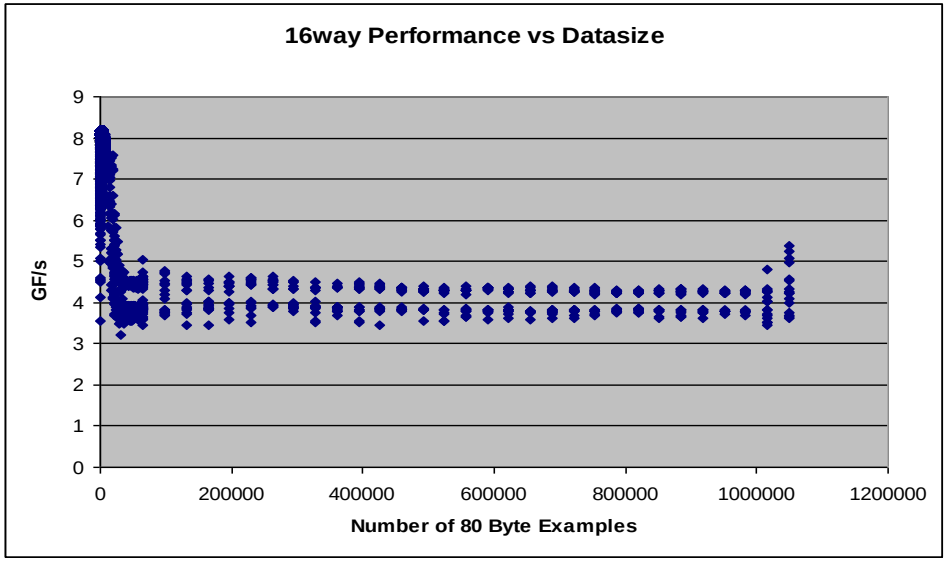


The CUDA execution model

- Loose coupling between SM translates to strong scaling (even on CPU cores) – very good news!
- *On x86 :beware SMP scaling limits caused by cache coherency* (AMD Barcelona example on TACC Ranger)

```
#pragma omp parallel for reduction(+ : sum)
    for(int i=0; i < nExamples; ++i) {
      Real d = getError(i);
      sum += d;
    }
```

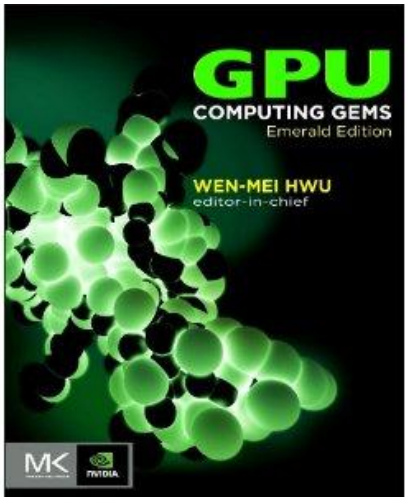
Likely cause: some AMD cache coherency messages take two hops



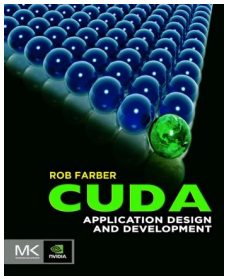
Task parallelism

- Asynchronous kernel launches will become more important (task vs. data parallelism)
  - x86 great for task parallelism
- Interesting to see how prevalent use will affect CUDA
  - Reduction to a single value does not naturally fit in the CUDA model as it requires:
    - Atomic operations (scalability issues!)
    - Separate kernels (rule 2: startup overhead)
    - Transfer to the host for the final step

Map irregular data structures to the CPU



GPU Computing Gems is an excellent resource



A gather operation

```
for(int i=0; i < n; i++)
  a[i] = b[index[i]]
```

The GPU L2 cache cannot help with large data

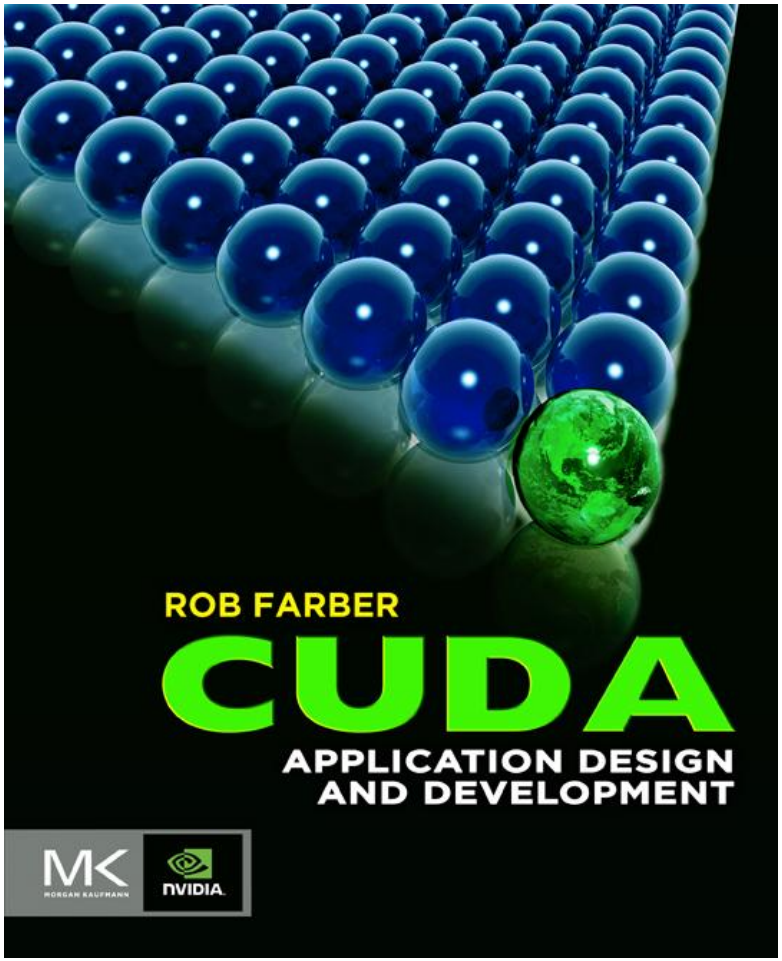
Size	Op	nTests	Time	Slowdown relative to sequential performance
0.01M	Sequential	1000	3.37E-06	
0.01M	Sorted	1000	3.44E-06	1.0
0.01M	Random	1000	7.46E-06	2.2
0.1M	Sequential	1000	1.39E-05	
0.1M	Sorted	1000	1.42E-05	1.0
0.1M	Random	1000	6.94E-05	5.0
1M	Sequential	1000	0.000107	
1M	Sorted	1000	0.000106	1.0
1M	Random	1000	0.000972	9.1
10M	Sequential	1000	0.001077	
10M	Sorted	1000	0.00105	1.0
10M	Random	1000	0.011418	10.6
100M	Sequential	1000	0.011553	
100M	Sorted	1000	0.013233	1.1
100M	Random	1000	0.132465	11.5

There is certainly much, much more

Thank you!

CUDA Application Design and Development is now available for preorder

http://www.amazon.com/CUDA-Application-Design-Development-Farber/dp/0123884268



Acknowledgements

Supported by Science Foundation Ireland under grant 08/HEC/I1450 and by HEA's PRTLI-C4.

sfi Science Foundation Ireland

Ireland's EU Structural Funds Programmes 2007 - 2013

Co-funded by the Irish Government and the European Union

EUROPEAN REGIONAL DEVELOPMENT FUND

AN RÚNAÍ OIDEACHAIS AGUS EOLAÍOCHTA | DEPARTMENT OF EDUCATION AND SCIENCE

HEA
Higher Education Authority
An tÚdarás um Ard-Oideachas