Software e Sviluppi per le GPU

Stato e Prospettive del Calcolo Scientifico

16-18 Febbraio 2011

Laboratori Nazionali di Legnaro (INFN)

Luca Ferraro

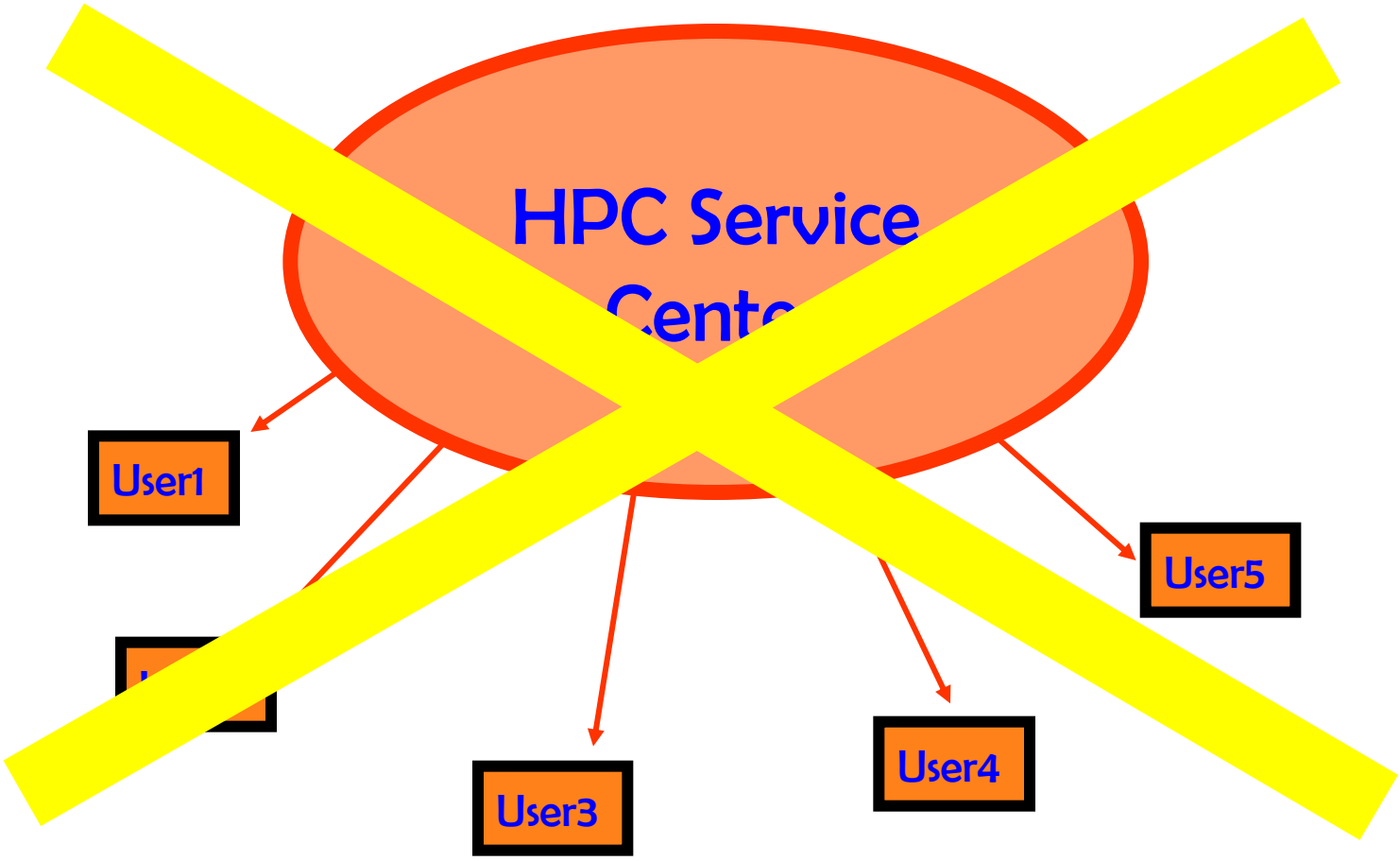
(l.ferraro@caspur.it)

What is CASPUR

Some Partnerships

- AIA
- AlphaData
- ARPA
- CIRA
- CNMCA
- CNR-DPM
- CNR-DSV
- CNR-IAC
- CNR-ICB
- CNR-ICRM
- CNR-IMCB
- CNR-IMIP
- CNR-INFM
- CNR-INMM
- CNR-IPCF
- CNR-IRC
- CNR-ISAC
- CNR-ISM
- CNR-ISTM
- CNR-MDM
- ENEA
- HP
- IASMA – Fond. Mach
- IFO Regina Elena
- IIT
- INAF
- INGV
- Ist. Naz. Spallanzani
- INSEAN
- ISPRA
- ISS
- IZSLT
- Microsoft
- NuMIDIA
- NVIDIA
- Policlinico Umberto I
- SCIRE
- Sigma-Tau
- SNS di Pisa
- SISSA
- Telethon
- Ylichron

HPC: the Wrong Model



What We Provide

- Code optimization
- Parallelization
- Scalable algorithms
- Scalable data management
- High performance data bases
- Workflow tuning
- Highly tuned HPC codes
  - SCElib (CUDA Zone!), LBM/BGK, CMPTool, NEMO
- **Training**
  - Training on the job
  - CASPUR's Summer School
  - 24+ intensive courses on a regular schedule (corsihpc.caspur.it)
  - Attended by ~450 people per year

- covered disciplines:
  - Applied Mathematics
  - Astrophysics
  - Bioinformatics
  - Chemistry
  - Data Analysis
  - Finance
  - Fluid Dynamics
  - Materials Science
  - Optimization Theory
  - Statistics
  - …

CASPUR & GPUs

- Budget is drastically reduced over years
- Dominant costs of HPC:
  - High-speed, low-latency interconnect
  - Scalable storage and file systems
  - System housing and cooling infrastructure
- Among all tested alternatives (FPGAs, exotic accelerators,…), GPUs:
  - while being reasonably easy to program
  - enabled a jump from 177 MFlops/W to 800+ MFlops/W
  - and enough widespread to limit investment risks

Jazz Fermi GPU Cluster

192 cores Intel X5650 @ 2.67 GHz

14336 cores on 32 Fermi C2050 GPUs

QDR IB Interconnect (4GB/s)

14.3 TFlops

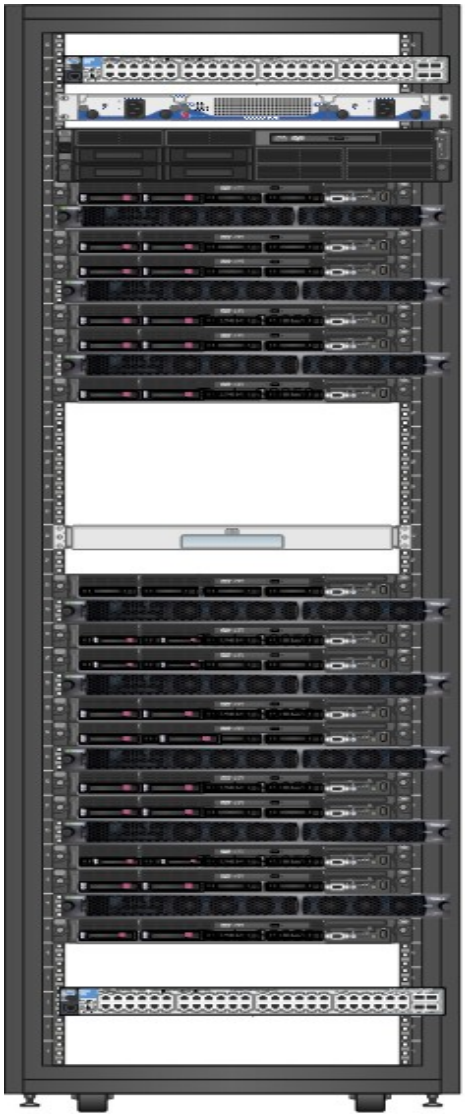
Peak performance

10.1 TFlops

Sustained Linpack

785 MFlops/W

Peak green performance



GPU-ready Applications

Many CUDA-enabled applications are installed and ready to use on the cluster:

Computational Chemistry

1. NAMD
2. Amber
3. GROMACS
4. DL-POLY
5. CP2K

Fluidodynamics

1. OpenFOAM
2. ACUSIM

Numerical Analysis

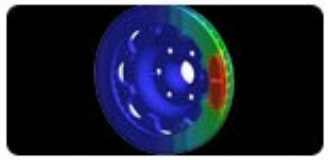
1. Matlab+GPU

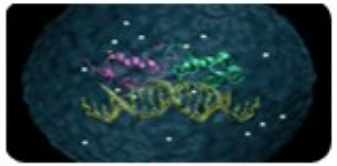
Finance

1. QuantLib

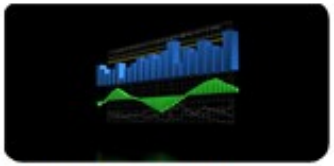
Bioinformatics

1. CUDA-BLASTP
2. GPU-HMMER
3. CUDASW++


Computational Structural Mechanics


Bio-Informatics and Life Sciences

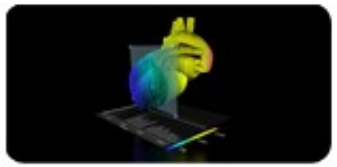

Computational Electromagnetics and Electrodynamics

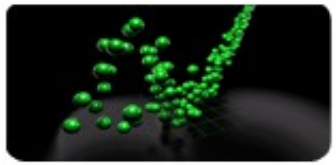

Computational Finance

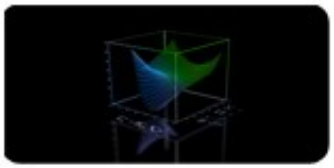

Computational Fluid Dynamics

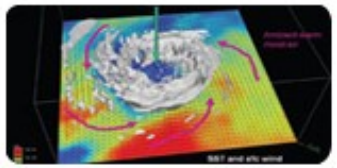

Data Mining, Analytics, and Databases


Imaging and Computer Vision


Medical Imaging

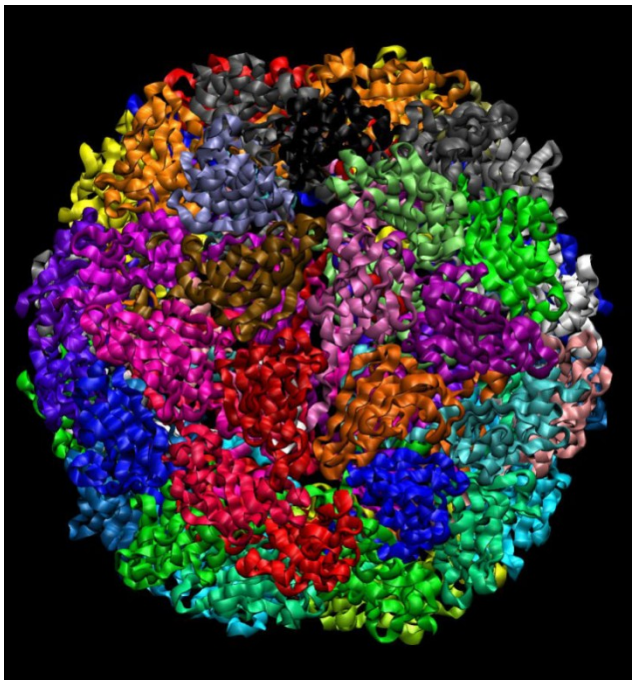

Molecular Dynamics


Numerical Analytics


Weather, Atmospheric, Ocean Modeling, and Space Sciences

NAMD Performance on CPUs/GPU

Satellite Tobacco Mosaic Virus (STMV) with NAMD



This benchmark consists of 1,066,628 atoms

Parameters affecting performances:
- real space cutoff: 12 angstroms
- PME electrostatics
- constant temperature and pressure (NPT)
- time step = 1 fs

Performances:

Serial:	166 days/ns (14.26 s/step)
Serial + GPU:	26 days/ns (2.25 s/step)
24 CPUs + 4 GPUs:	3.33 days/ns (0.29 s/step)
192 CPUs + 32 GPUs:	0.58 days/ns (0.05 s/step)

Porting in progress ...

- ... some GPU-ready applications are not as ready as declared
  - a lot of “experimental” work in progress
  - not driven by wide experience
- many fields yet to cover
  - some do not rely on community codes
    efforts are “wasted” multiple times
  - some might not benefit much from this revolution

One Year of GPU Porting Activities

- SCElib (lepton-molecule scattering)
  available in CUDA Zone
- BGK (Fluid Dynamics)
- Particle and polimers transport in water
  (mixed MD+Boltzman)
- Pricing Derivatives with Market Model (Finance)
- Climatology (ENEA - Casaccia)
- Earth Quake modelling (INGV)

  ....

- and very other BIG committors (NDAs)

First in Italy

Looking for collaborations

research groups intereset in this new technology are heartly invited to submit their ideas and needs

...we are paid for that!

- we are thinking also about a collaborative acquisition of larger cluster based on GPUs
  - aggregate resources
  - rely on our experience in both mantaining activities and in know-how to use them efficiently

Our Experience with GPUs
... and results

GPU: an awaited help

- GPGPU is a new promising technology to boost scientific computing code performances:
- ☺ *pros:*
  - High number of processor elements (up to 500)
  - specialized for *intense data-parallel computations*
    - same algorithm on many element (SPMD)
  - very light thread creation/content-switch to hide latencies
    - best performance when thousand of threads are cast
  - impressive peak performance (> 1 TFlops)
    ... and often highly sustained (50-80%)
- ☹ *cons:*
  - pre-existend code has to be modified
  - still a new evolving device
  - ... yet far from standardization

GPU Programming Model

- GPU is seen as a coprocessor
  - with its own memory
  - able to execute and control thousands of threads
- *computational-intensive data-parallel* regions of an application are delegated to the GPU device
  - the same kernel is executed by all threads in parallel on different data
- GPU threads are very different from CPU threads:
  - GPU threads are extremely lightweighted
    - no *content-switch overload*
  - GPU requires thousands of threads to hide latencies and to reach peak performance
    - a multi-core CPU can handle few of threads per core

Available GPU Programming Toolkit

- NVIDIA CUDA
  - Proprietary
  - Rapidly evolving in reaction to user needs
  - Lots of available sources and examples
- ATI Stream
  - Proprietary
  - Still waiting for it…
- Microsof DirectCompute
  - OS Proprietary, device independent
  - Big market force behind

- OpenCL (Khronos Group)
  - open standard with the backing of:
    Apple, AMD/ATI, Intel, Nvidia
  - device independent
  - slow evolving standard

all present similar features

... but different programming interfaces

OpenCL or not OpenCL

- Optimizing a code with proprietary toolkit is very sensitive to the formulation of the kernel
  - high performance can be reached:
    - rethinking the algorithm mapping the features exposed by the hardware
- Optimizing the same code with OpenCL for the same device is possible
  - with a hard work you get almost the same performances (10-30% less)
- Optimizing the same code for several architecture is almost impossible
  - portability is at cost of performance
  - OpenCL compilers still lacks of counterpart smartness in producing a competitive intermediate code

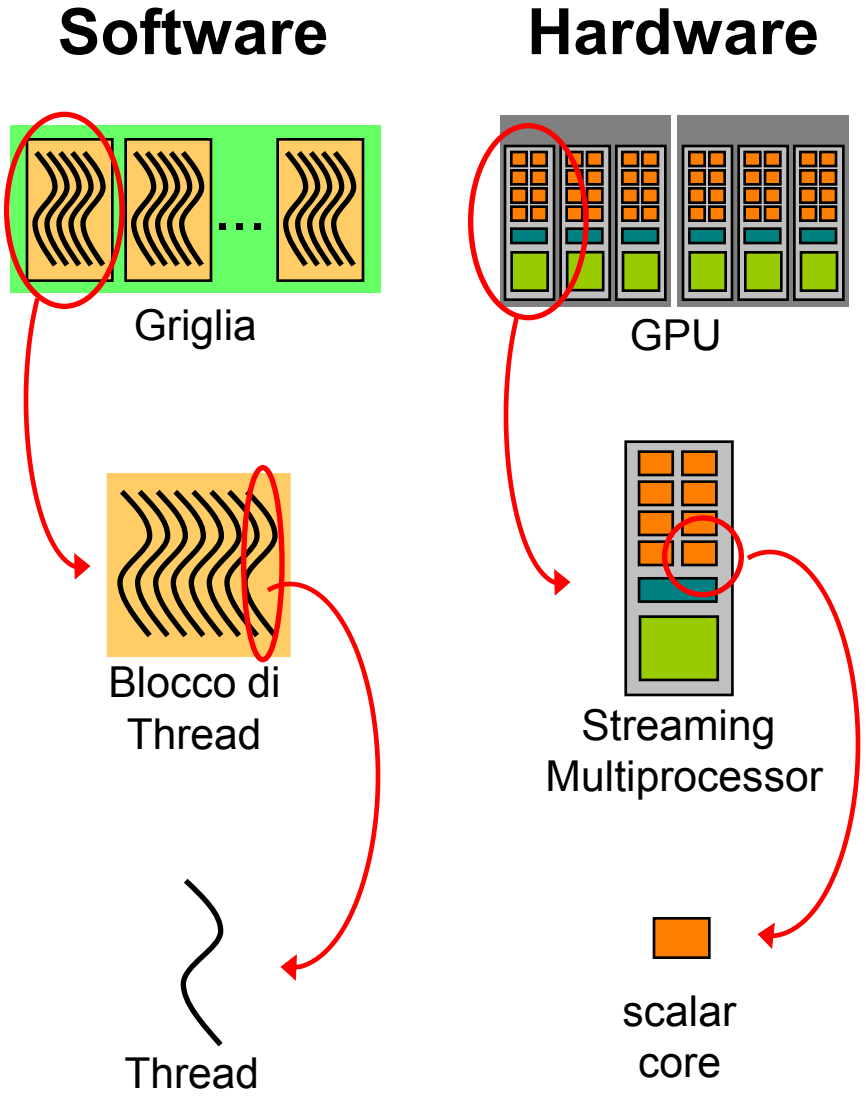
Directive based approach

- Directive based approach are coming out for those who really don't want/have time to rewrite the code (PGI Accelerator, HPMM, ...)
- ☺
  - "reduced" number of code modification
  - incremental approach
  - reducing time for code validation & verification
- ☹
  - there is not a standard
  - performance depends on compiler
  - performance "often" lower then using CUDA, OpenCL, …

Looming on the Horizon

- Accelerator support in OpenMP
  - AMD, Intel, NVIDIA, TI, most system vendors and many HPC centers (including CASPUR) working on it
  - A BIG customer asking for it
  - Aimed at more detailed control than PGI Directives
- Heterogeneous architecture support in OpenMP
  - Includes GPUs, of course
  - Part of EU STREP proposal just submitted (BSC, CASPUR, EPCC, NAG, RWTH)
  - Aimed at ease of programming
- Bringing to GPUs models from different platforms
  - Like those developed for IBM Cell
  - Part of EU STREP proposal just submitted (BSC, CASPUR, ICHEC, NVIDIA, PETAPATH)

Underneath Whatever Model/API



- It all boils down to:
  - taking a lot of concurrent computations
  - (typically the iteration space of a loop)
  - and spreading them among block of threads
- With Accelerator Directives, this is semi-automated
  - The programmer annotates loops
  - The compiler moves work to GPU
- With direct programming interfaces, everything must be done manually

Our Experience with CUDA

- We have extensively used NVIDIA CUDA C/Fortran
  - few extensions to C/Fortran language to get ready
  - very easy and fast learning curve even for computational scientists focused on research goals
  - many reference sources and samples
  - very easy to port to OpenCL
    - just few differences

Many Ready-to-use Libraries

- CUDA toolkit includes many mathematical libraries: http://developer.nvidia.com/object/cuda_3_2_downloads.html
  - CUBLAS: Basic Linear Algerba Subroutine
  - CUFFT : Fast Fourier Transform
  - CUSPARSE: sparse matrix algebra
  - CURAND: pseudorandom and quasirandom number generator
- more libraries available :
  - MAGMA (Matrix Algebra on GPU and Multicore Architectures) a LAPACK port for GPU http://icl.cs.utk.edu/magma/
  - CULA: Lapack port (commercial version) http://www.culatools.com/contact/cuda-training/
  - THRUST (CUDA library for parallel algorythms in C++) http://code.google.com/p/thrust/
  - CUDPP (Data Parallel Primitives): parallel prefix-sum, sort, reduction http://code.google.com/p/cudpp/

GPU vs. CPU

- How fast is a GPU compared to CPU?
- Which speedup should we expect?
  - A factor 1?
  - A factor 10?
  - A factor 100?
- But let's ask a question first:
- What are we comparing to what?

Let's Do Some Theory

- Intel Xeon X5650@2.67GHz esacore (Westmere)
  - CPU Peak Performance
    - Double precision 64 GFlops
    - Single precsion 128 GFlops
- GPU NVIDIA Fermi Tesla S2050
  - GPU Peak performance:
    - Double precision 500 GFlops
    - Single precision 1000 GFlops
- Fermi GPU & Westmere CPU are **current state of the art** ...
- IF the code reaches the same efficiency on both
  - i.e. percentage of peak performance, let's say 10%
  - (which is quite improbable, by the way)
- THEN the **GPU should be 7.8x faster than the CPU**

And Face Some Practice

- In our experience:
  - using real codes (not toys)
  - using CUDA or PGI Directives
  - keeping a similar numerical scheme/algorithm as those in the CPU version
  - **honest speedups range from 2x to 12x**
- And when, in the same scenario, the speedup approaches 100x:
  - the CPU code was inefficiently written
  - or the choice of compiler options was poor
- However:
  - Some kernels are intrinsically inefficient on a conventional CPU while fly on a GPU
  - Some kernels are intrinsically inefficient on a GPU,
    but the source of inefficiencies will be addressed along the roadmap
  - The game on a GPU is very, very different:
    a complete rethinking of the approach can sky-rocket performances

Rethinking the Approach: Application Profile

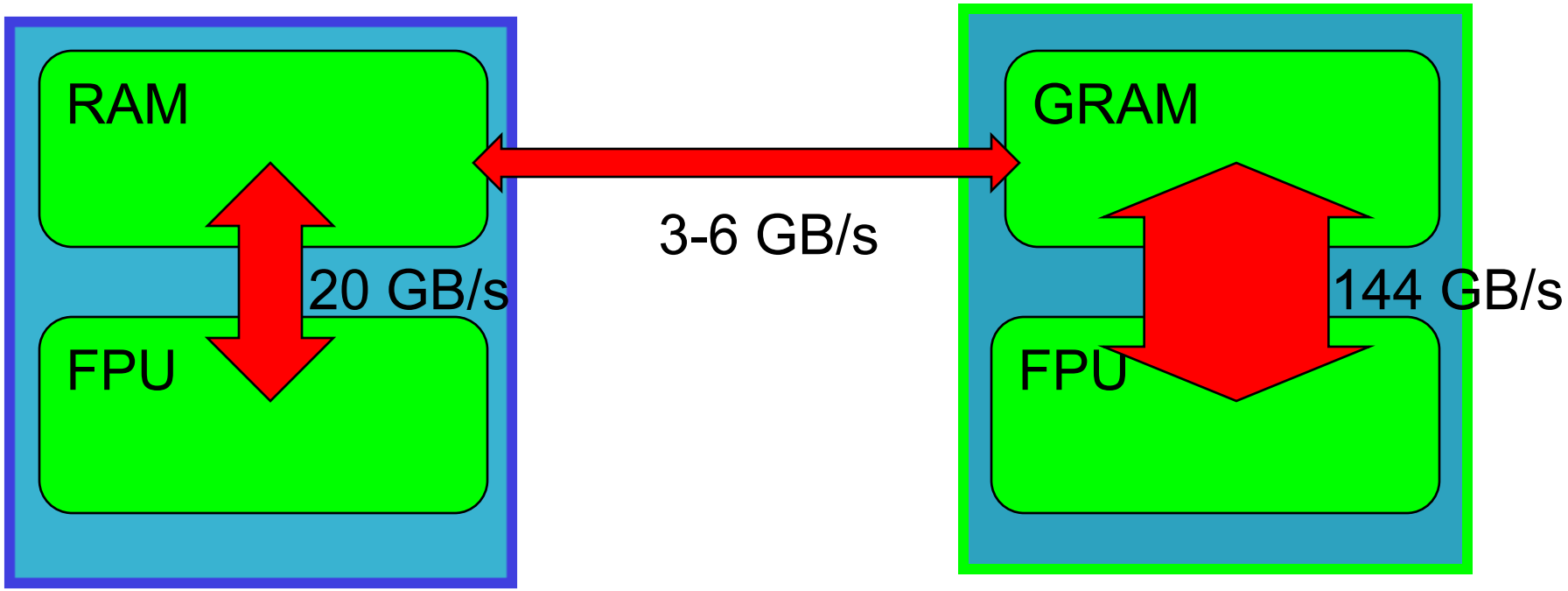
- A real application is made of many computational kernels
- E.g., for a typical “PDE on a mesh” code:
  1. Source terms
  2. Spatial scheme
  3. Time-stepping scheme
  4. Boundary conditions
- Usually, 1 and 4 will show much less concurrency
- Never expect the kernels to have the same relative weights on the GPU as on the CPU

The Bottleneck Kernel

- What to do with a kernel that brakes your GPU?
- Be objective!
  - 50% of the GPU runtime vs. 4% of the CPU runtime
  - But a 20x overall speedup
  - Wait, maybe the ARM cores in future GPUs will fix
- Be creative!
  - Find a different formulation, more GPU friendly
  - Reimplement it: sometimes some dependencies are artifacts to spare memory
- If you can’t beat them, Hide them!
  - delegate it to host CPU, if data must be frequently exchanged
  - It will come for free

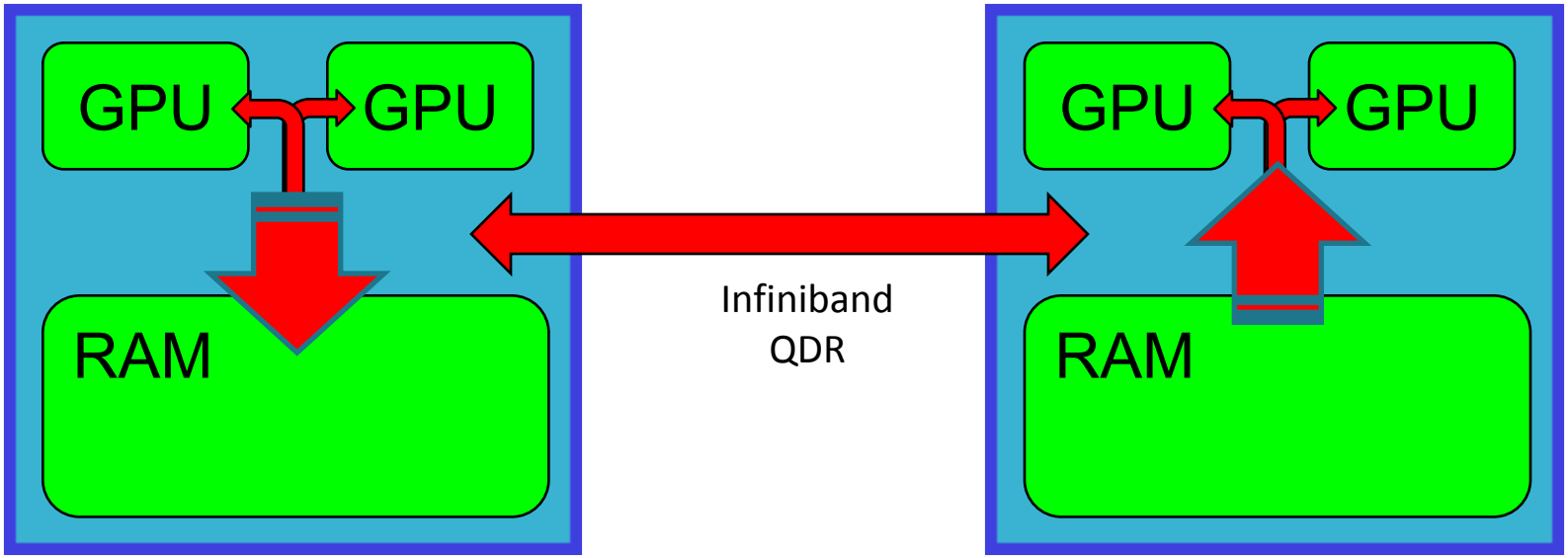
The Data Transfer Bottleneck: Single GPU

- Real GPU bottleneck is bandwidth for data communication from/to GPU to/from CPU
  - CPU memory bandwidth: 10-20 GB/s
  - GPU memory bandwidth: peak 150 GB/s (90GB/s real)
  - PCIe 16x gen2: peak 8GB/s per direction (3-6GB/s real)



The Data Transfer Bottleneck: multi-GPUs

- using many GPUs:
  - on the same node: might share PCIs bandwidth
    - degraded performances during transfers in the same direction
  - on different nodes using Infiniband QDR 4x
    - bandwidth 4GB/s similar to internal bandwidth (MPI might interfere)
    - transfer times is **summed up**:
      GPU-CPU + CPU-CPU + CPU-GPU



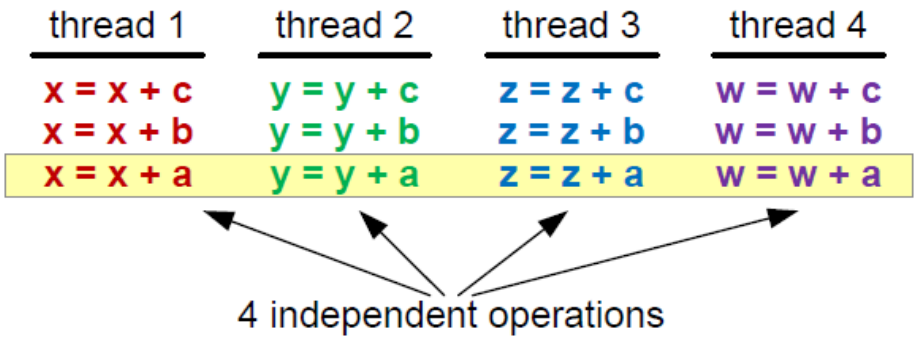
Who's Afraid of the Data Transfer Bottlenecks?

- Think of your host+GPU system like a cluster
  - The host is already a cluster, in some respects
- And use the same approaches as in MPI
  - Multiple buffering
  - Overlapping communications and computations
  - Hiding latency with computations
- In particular in multi-GPU, multi host applications, it's nothing more than having a greater latency

Rethinking the Approach: Fight All Latencies

- Fundamental GPU programming principle
  - Memory latency is huge (400 800 cycles), but bandwidth is wide
  - Hide memory latency behind parallelism
  - Spawn a lot of threads:
    if some stalls, others will work!
- Often forgotten truth
  - Dependent instructions will stall
  - Give enough independent instructions to your thread!

TLP vs. ILP

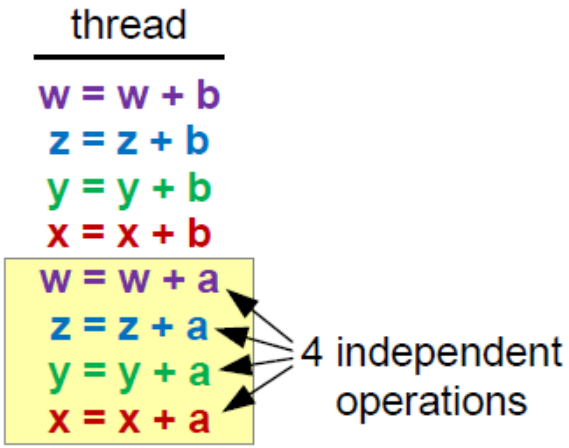


Thread Level Parallelism

- 192 threads/SM on G80
- 562 threads/SM for GF100

Instruction Level Parallelism

Let the thread work on many independent operations while reusing data as much as possible



BEWARE: if you are short on independent data/operations TLP and ILP strive, find your sweet-spot by experiment

TLP, ILP and Resources

- Find your right balance
  - Increasing TLP will reduce available resources per thread
  - Increasing ILP may make your thread more resource hungry
  - Reusing data may lessen resource usage
- Limiting the number of registers per thread at compile time changes the generated code
  - Sometimes with very pleasing results in terms of performance

How to get the best from GPU

1. Find out if your problem is caracterized by *intense data-parallel computations*
   - if this is not the case, GPU might not be the solution
2. Choose the model to use:
   1. directive based *vs* direct programming approach
3. If not using directive based approach:
   1. rethink your algorithm in a GPU friendly way
   2. hide latencies as much as possible
   3. tune for your target HW for peak performances
   4. cooperate with your CPU (eterogeneous system!)

Conclusions

- GPUs do really boost *intense data-parallel computations* in scientific computing codes
  - at lower costs, with few efforts
- we are facing a real change
  - **we must surf a true eterogeneous system revolution**
- too many competitive models , one to spot for the future
  - portability at the cost of performances
  - if you eager for performance, use your vendor's toolkit
- what any, rethink your algorithm and recast your problem into a GPU friendly way
- partecipate to CASPUR GPU Programming Courses

  http://corsihpc.caspur.it

References

- NVIDIA, NVIDIA CUDA - Programming and Best Practice Guides
- A. Munshi (Ed.) - The OpenCL Specification, Khronos OpenCL
- INTEL “Debunking the 100X GPU vs CPU Myth: An Evaluation of Throughput Computing on CPU and GPU”
- P. Micikevicius, “Fundamental and Analysis-Driven Optimization”, GPU Technology Conference 2010 (GTC 2010)
- V. Volkov, “Better performance at lower occupancy”, GPU Technology Conference 2010 (GTC 2010)
- J. Dongarra *et al.* “An Improved MAGMA GEMM for Fermi GPUs”
- J. Dongarra *et al.* “From CUDA to OpenCL: Toward a Performance-portable Solution for Multi-platform GPU Programming”