GPGPU
in Theoretical Physics

F. Di Renzo
University of Parma and I.N.F.N.

Napoli, Jan 25th 2010

Not really a review. Keep in mind instead that

we like novels more than dictionaries

Available online at www.sciencedirect.com
ScienceDirect
Computer Physics Communications 177 (2007) 631–639
Computer Physics Communications
www.elsevier.com/locate/cpc

One of the pioneering papers

Lattice QCD as a video game

Győző I. Egri [a], Zoltán Fodor [a,b,c,*], Christian Hoelbling [b], Sándor D. Katz [a,b], Dániel Nógrádi [b], Kálmán K. Szabó [b]

[a] *Institute for Theoretical Physics, Eötvös University, Budapest, Hungary*
[b] *Department of Physics, University of Wuppertal, Germany*
[c] *Department of Physics, University of California, San Diego, USA*

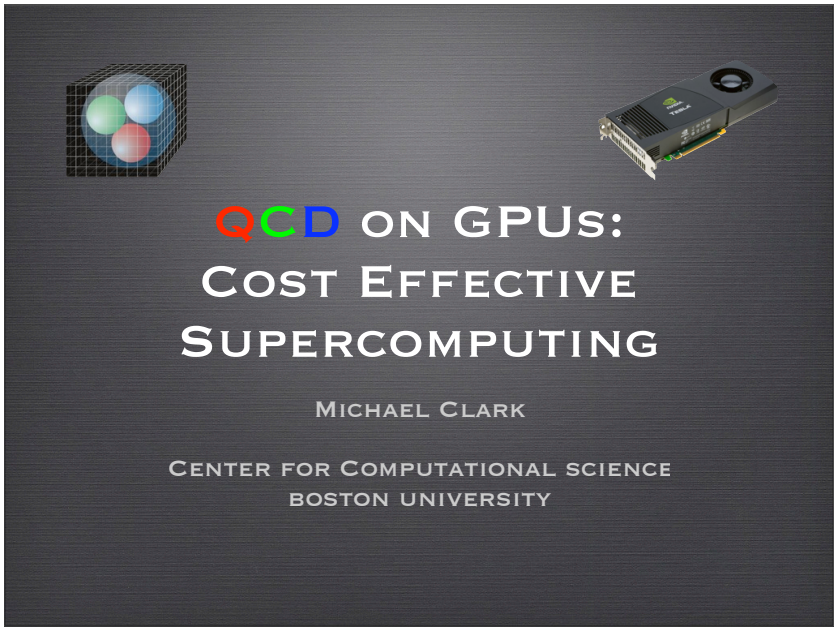
Received 2 February 2007; received in revised form 29 May 2007; accepted 7 June 2007

Available online 15 June 2007

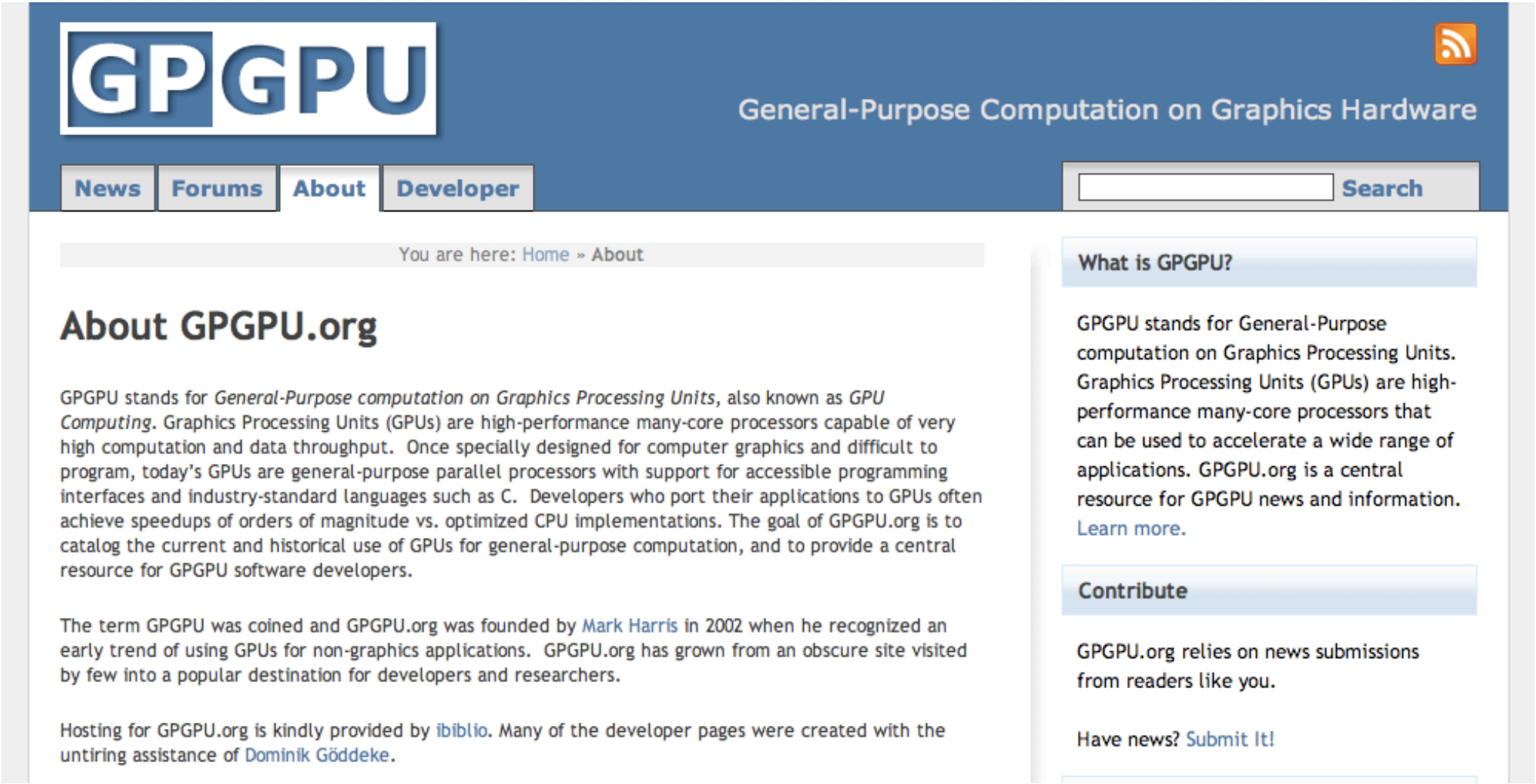
Abstract

The speed, bandwidth and cost characteristics of today's PC graphics cards make them an attractive target as general purpose computational platforms. High performance can be achieved also for lattice simulations but the actual implementation can be cumbersome. This paper outlines the architecture and programming model of modern graphics cards for the lattice practitioner with the goal of exploiting these chips for Monte Carlo simulations. Sample code is also given.

A review talk at the 2009 Lattice Conference



Nowadays proof of existence (according to my friend Lele Tripiccione):
is there a web page? http://gpgpu.org



We can try to look for Science, Physics, Theoretical Physics ...

GPGPU

General-Purpose Computation on Graphics Hardware

News | Forums | About | Developer

Search

GPU Simulations of Gravitational Many-body Problem and GPU Octrees

January 20th, 2010

This undergraduate thesis and poster by Kajuki Fujiwara and Naohito Nakasato from the University of Aizu approach a common problem in astrophysics: the many-body problem, with both brute-force and hierarchical data structures for solving it on ATI GPUs. Abstracts:

Fast Simulations of Gravitational Many-body Problem on RV770 GPU
Kazuki Fujiwara, Naohito Nakasato (University of Aizu)
Abstract:

The gravitational many-body problem is a problem concerning the movement of bodies, which are interacting through gravity. However, solving the gravitational many-body problem with a CPU takes a lot of time due to O(N^2) computational complexity. In this paper, we show how to speed-up the gravitational many-body problem by using GPU. After extensive optimizations, the peak performance obtained so far is about 1 Tflops.

Oct-tree Method on GPU
N.Nakasato
Abstract:

The kd-tree is a fundamental tool in computer science. Among others, an application of the kd-tree search (oct-tree method) to fast evaluation of particle interactions and neighbor search is highly important since computational complexity of these problems are reduced from O(N^2) with a brute force method to O(N log N) with the tree method where N is a number of particles. In this paper, we present a parallel implementation of the tree method running on a graphic processor unit (GPU). We successfully run a simulation of structure formation in the universe very efficiently. On our system, which costs roughly $900, the run with N ~ 2.87×10^6 particles took 5.79 hours and executed 1.2×10^13 force evaluations in total. We obtained the sustained computing speed of 21.8 Gflops and the cost per Gflops of 41.6/Gflops that is two and half times better than the previous record in 2006.

Posted in Research | Tags: Astrophysics, Data Structures, Papers, Posters | Write a comment

What is GPGPU?

GPGPU stands for General-Purpose computation on Graphics Processing Units. Graphics Processing Units (GPUs) are high-performance many-core processors that can be used to accelerate a wide range of applications. GPGPU.org is a central resource for GPGPU news and information. Learn more.

Contribute

GPGPU.org relies on news submissions from readers like you.

Have news? Submit It!

Subscribe

Entries RSS
Comments RSS
follow me Twitter feed

Categories

Business (51)
Developer Resources (170)
Events (98)
Press (39)

Again on gpgpu.org, let's have a look at programming environments

GPGPU Programming

The GPGPU programming landscape has rapidly evolved over the past several years, so that now there are several approaches to programming GPUs. Recently, convergence towards standardization has begun. Readers are recommended to browse the available material to make their own decisions on which approach to use.

NVIDIA CUDA, AMD Stream and OpenCL

GPU Computing really took off when CUDA and Stream arrived in late 2006. These are programming interfaces and languages, designed by the GPU vendors in close proximity with the hardware, that constitute a tremendous step towards a usable, suitable, scalable and manageable future-proof programming model. Learn more about AMD Stream. Learn more about NVIDIA CUDA.

The Open Compute Language (OpenCL) is designed to provide a unified API for heterogeneous computing on several kinds of parallel devices, including GPUs, multicore CPUs and the Cell Broadband Engine. Learn more about OpenCL.

Sh and Brook for GPUs

High-level languages and programming environments for GPUs, in particular *BrookGPU* from Stanford University and *Sh* from the University of Waterloo were precursors to todays solutions like CUDA and OpenCL. Sh has been commercialized by its developers into RapidMind and BrookGPU has served as the basis for AMD's Stream. Learn more about BrookGPU and Sh.

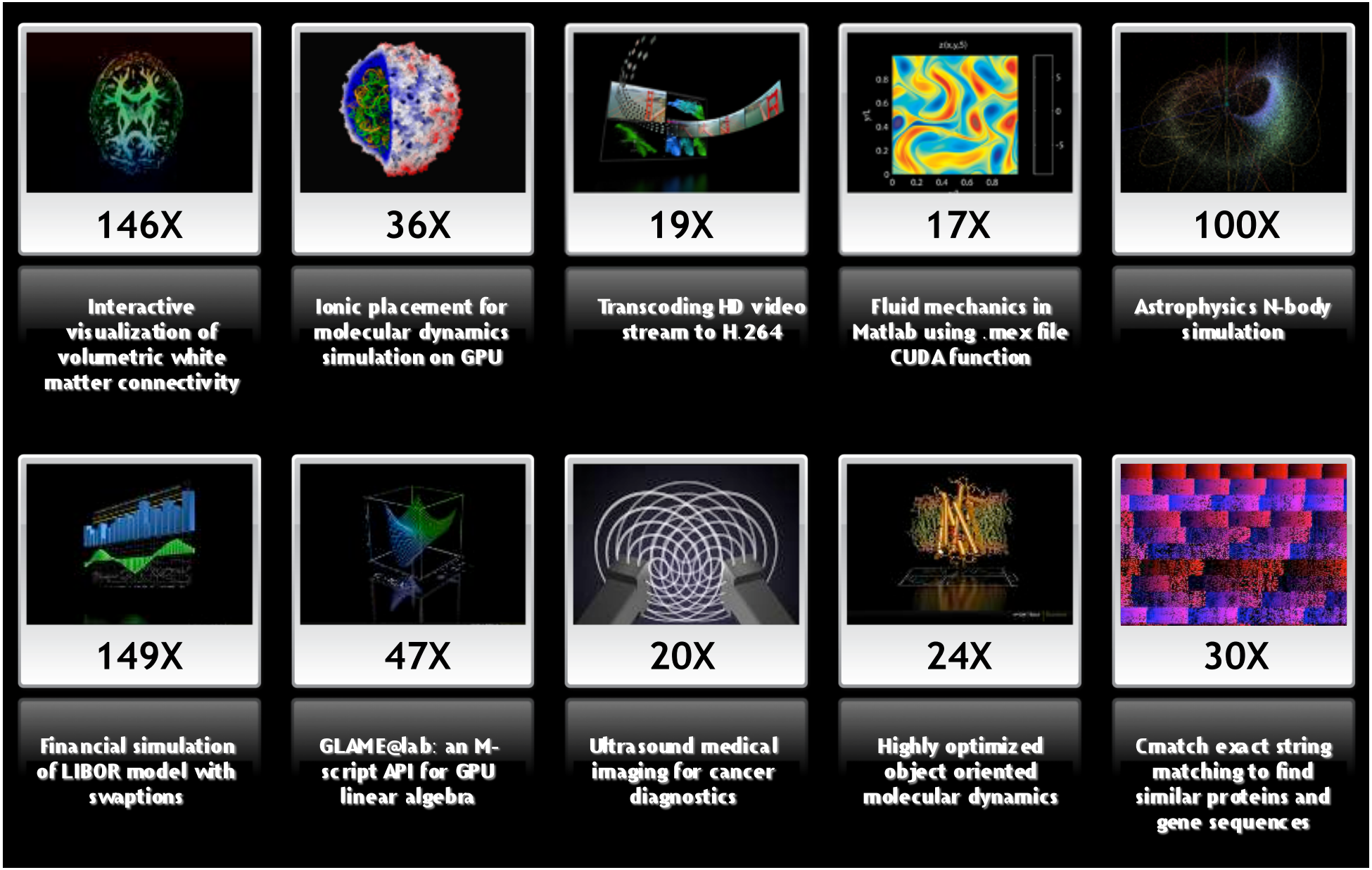
Legacy GPGPU: Graphics APIs

In the early days, GPGPU programming was a bit hacky. Algorithms had to be cast in terms of graphics APIs such as OpenGL and Direct3D; the underlying hardware was not fully exposed or documented; and the programming was sometimes unproductive. Despite all this, a lot of ground-breaking research has been accomplished that helped pave the way to what GPU computing is now. Despite their legacy status, these older tutorials and sample applications still have some value. Learn more about legacy GPGPU.

GPGPU Tutorials

Supercomputing 2009 CUDA Tutorial
PPAM 2009 GPU and OpenCL Tutorial
ISC 2009 CUDA Tutorial
ASPLOS 2008 CUDA Tutorial
SUPERCOMPUTING 2008 Tutorial
SUPERCOMPUTING 2007 Tutorial
SIGGRAPH 2007 GPGPU Course
SUPERCOMPUTING 2006 Tutorial
IEEE Visualization 2005 Tutorial
SIGGRAPH 2005 GPGPU Course
IEEE Visualization 2004 Tutorial
SIGGRAPH 2004 GPGPU Course

Move to NVIDIA web pages: examples of CUDA applications



As I told you, the style is not really that of a review

I will basically concentrate on one type of applications (lattice), but I will try to make quite general comments

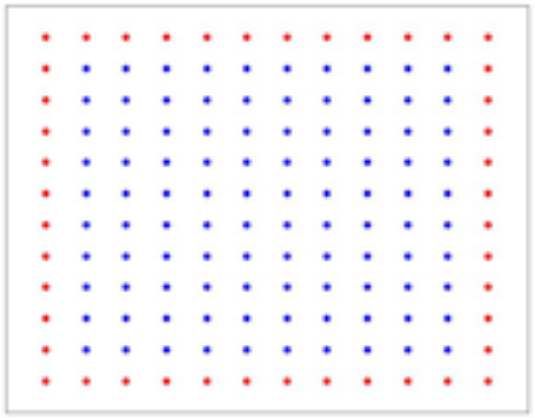
- It's a SIMD business!
- An example from Statistical Mechanics: Hybrid Monte Carlo for a spin model (keeping an eye on Lattice Gauge Theories)
- It's a SIMD business, but one does not deal in general with embarassing parallelism (synchronizing is an issue)
- New GPGPU programming environments vs legacy GPGPU
- What can we learn from Lattice QCD?
- A single mention of a completely non-lattice application
- Conclusions

We said it's a SIMD business

- Single Instruction Multiple Data: problems with a huge number of degrees of freedom, each roughly undergoing the same "dynamics".
- A rough (and personally biased) distinction: lattice vs non-lattice applications (canonical example of the latter type: Molecular Dynamics).
- SIMD/MIMD classification emphasyses the two main characters on the stage: FLOPS and BYTES. There are in general a given Flops/Bandwith ratio and a desired one!
- Who is the director in your program? Notice that these devices do not run an OS, so in general programming paradigm is that of number-crunching accelerators.
- As a byproduct there are in general 2 bandwiths to take into account (inside the device and CPU-GPU).

And here we go with the example: 2d XY spin model

$$H = -J \sum_{ij} \vec{s_i} \cdot \vec{s_j} = -J \sum_{ij} \cos(\theta_i - \theta_j).$$



Il modello XY come un video gioco

Candidato Giovanni Conti

Relatore Dott. Francesco Di Renzo
Universitá di Parma
https://www.fis.unipr.it

28 Febbraio 2008

It's a basic prototype of a LATTICE problem

Typical problem: compute by Monte Carlo

$$\frac{1}{N}\sum_{i=1}^{N\to\infty} O(X_i) \to \langle O \rangle_B$$

A good proptotype computation: Hybrid Monte Carlo. Configurations are generated

via a fictitious hamiltonian system

$$\mathcal{H} = \sum_i \frac{\pi_i^2}{2} + S(\theta_i)$$

Fictitious momenta are gaussian

$$P(\pi) = \mathcal{N}_0 e^{-\frac{1}{2}\sum \pi_i^2}.$$

$$S(\theta_i) = \beta H(\theta_i),$$

$$H = -J\sum_{ij} \vec{s_i}\cdot\vec{s_j} = -J\sum_{ij}\cos(\theta_i - \theta_j).$$

Given the spin (angles) configuration, one extracts momenta and then obtains a new (extended) configuration $\{\theta_i', \pi_i'\}$

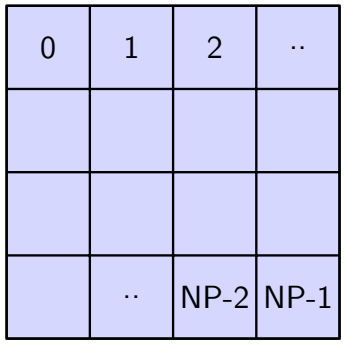
via a Hamiltonian flux

$$\dot{\theta}_i = \frac{\partial \mathcal{H}[\theta,\pi]}{\partial \pi_i}, \qquad \dot{\pi}_i = -\frac{\partial \mathcal{H}[\theta,\pi]}{\partial \theta_i}.$$

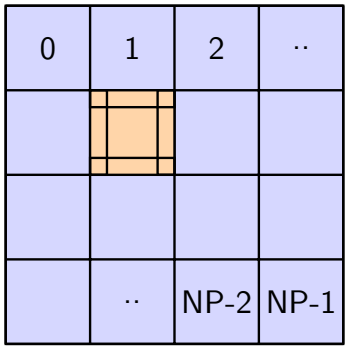
New (angles) configuration will be eventually accepted/rejected.

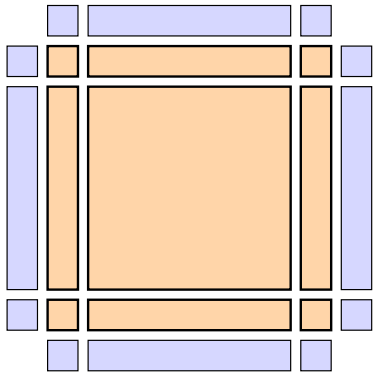
The equations for angles are LOCAL, those for angles are NOT LOCAL!

So, how do we SIMD-solve a prototype LATTICE problem?

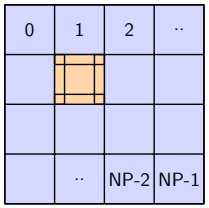


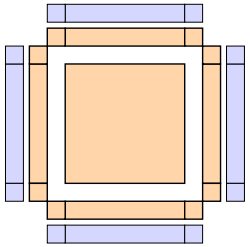
First step: a collection of sublattices



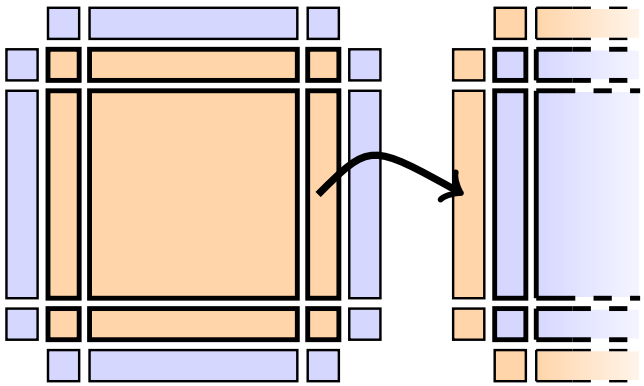


But there is a nn interaction! Borders





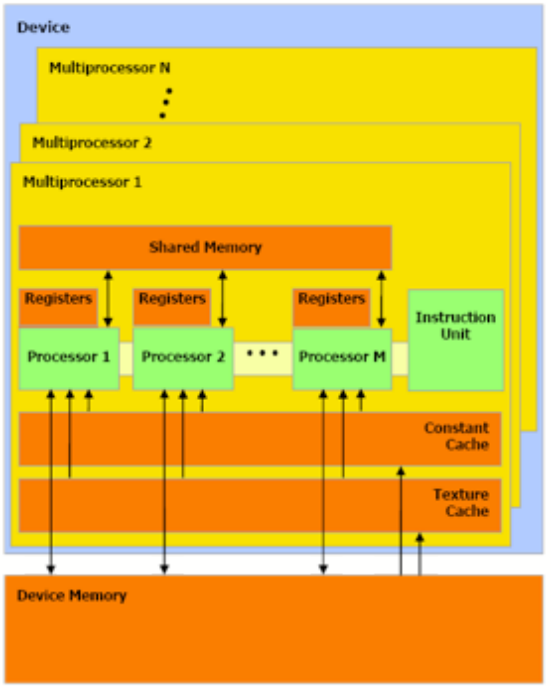
By the way: usually one wants to optimize memory



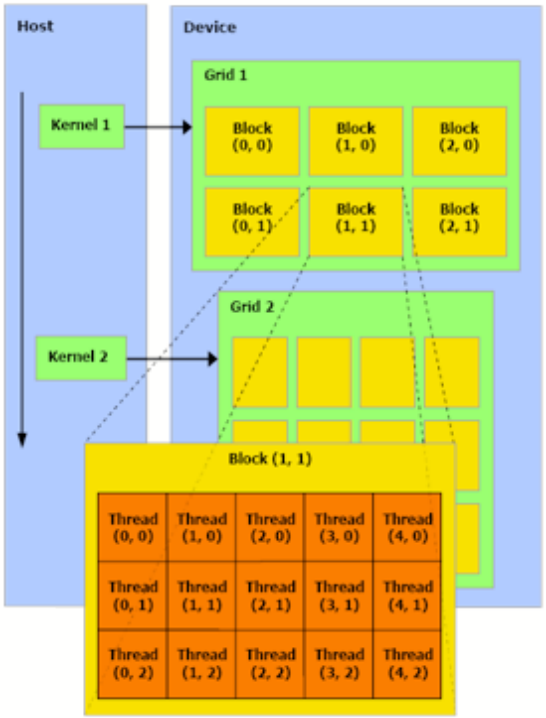
Borders exchange is a sinchronization issue!
Typically, one wants to hide latencies with computations.

The typical CUDA solution (I)

The CUDA environment by nVIDIA is a quite natural exploitation of the typical GPU architecture



Notice the memory hierarchy! Shared memory enables nn communications and synchronization ...



CUDA basics: grid of blocks and blocks of threads

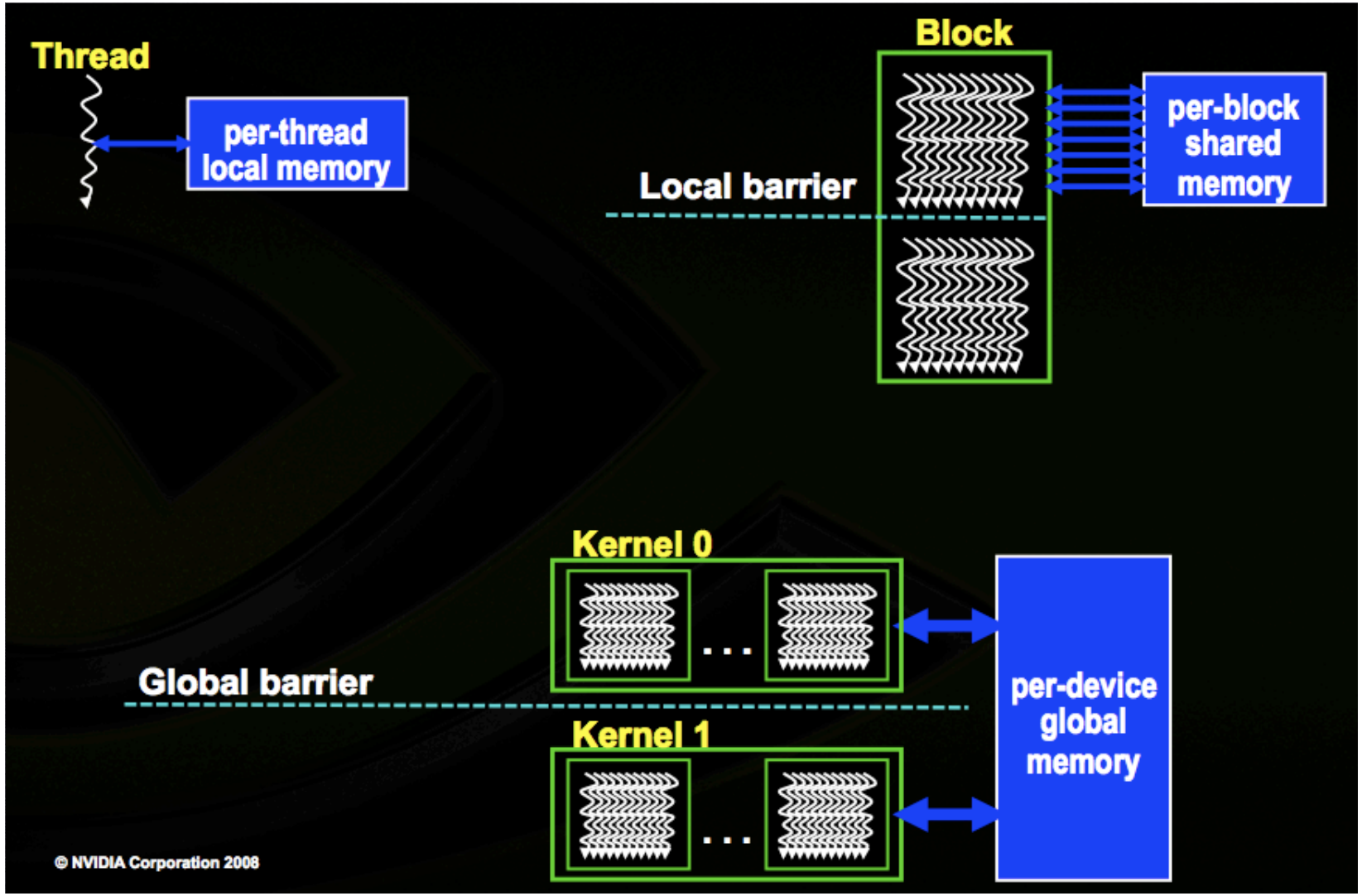
Assign spins to threads and sublattices to blocks

Since a block is assigned to a given multiprocessor one is done with local data sharing and synchronization

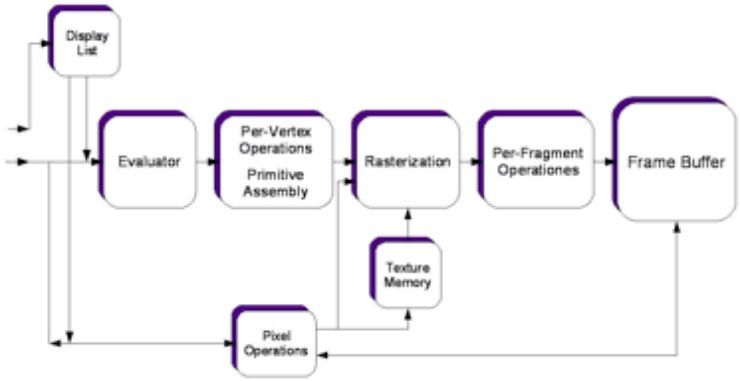
Resources are finite and blocks are sliced into warps

The typical CUDA solution (II)

In general for any given (not only lattice!) problem, we can better understand the issue of data-sharing and synchronization by inspecting nVIDIA picture for that (we can now think for example of a MD problem)

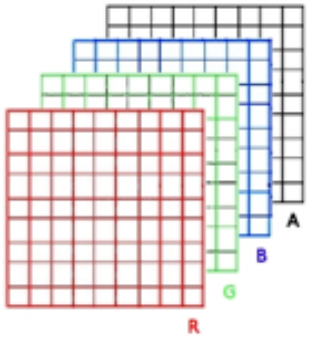


Legacy GPGPU: talk to GPU as if you were drawing



In the end, why are GPU efficient? Because computer graphic needs a lot of work! Efficiency is nowadays obtained by a HW graphic pipeline.

There is a lot of work involved in getting from a 3d image to a 2d bitmap. Roughly speaking, in the end one needs a bunch of bits on a display and that's why basic datatype are textures (4 components for colors)



In the OpenGL environment one can make use of the GSGL shading language

a) a kernel is "enabled" to enter the graphic pipeline

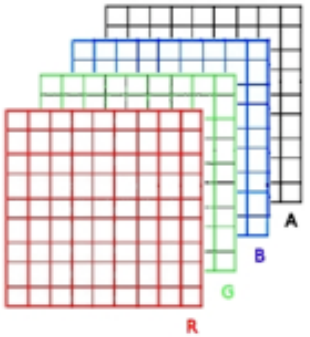
b) input means binding textures to texture units, output means binding target texture to a FBO

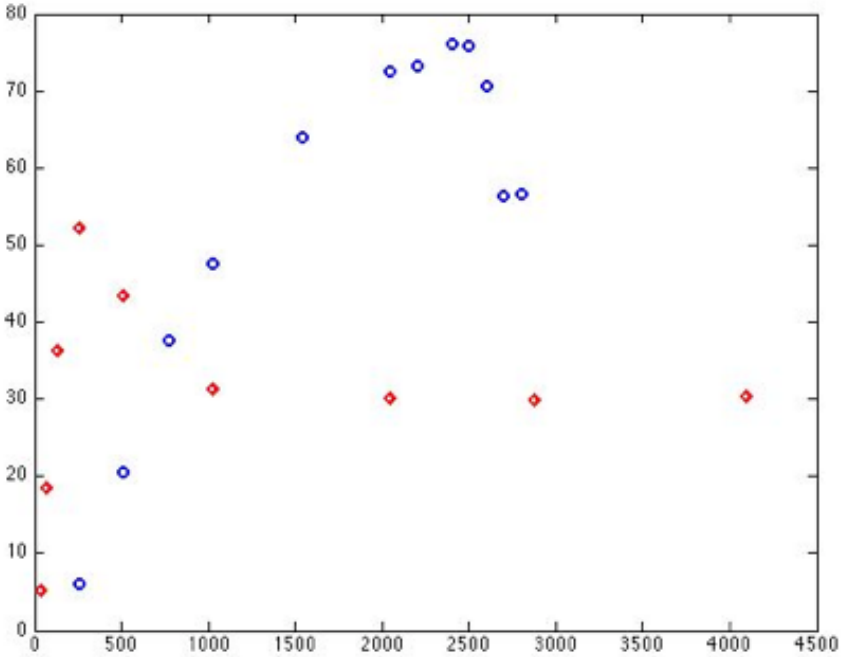
c) one pipeline stage is input for the following: RW restrictions! PING PONG

Theoretical Physics in OpenGL...

Texels can be your spins and texels make a texture like spins make a lattice.

An RGBA texture accomodate 4 replicas of a lattice and nn (which are 4 in 2d) can be mapped to a texture as well.





Efficiency is not at all bad!

x-axis is lattice size, y-axis is gain with respect to a given serial code.

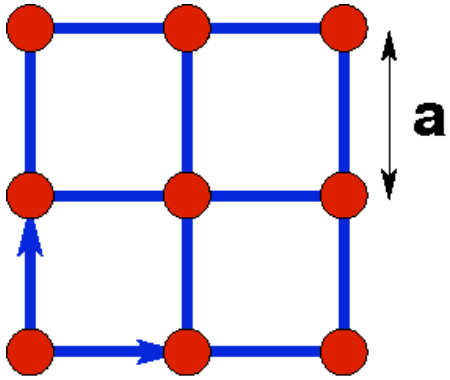
Blue point are OpenGL simulations

Red point CUDA simulations

What do we learn in general from QCD? (I)

The spin example was made of not so many degrees of freedom... A typical problem in LQCD is the application of the Wilson-Dirac operator

$$\psi'_{\alpha i}(x) = \sum_{\mu=1}^{4}\{ U_\mu^{ij}(x)\,(1+\gamma_\mu^{\alpha\beta})\,\psi_{\beta j}(x+\hat{\mu}) + U_\mu^{\dagger ij}(x-\hat{\mu})\,(1-\gamma_\mu^{\alpha\beta})\,\psi_{\beta j}(x-\hat{\mu})\ \}$$



$U_\mu^{ij}(x)$ 3x3 complex matrices, residing on links

$\gamma_\mu^{\alpha\beta}$ 4x4 complex matrices, sparse (4 complex)

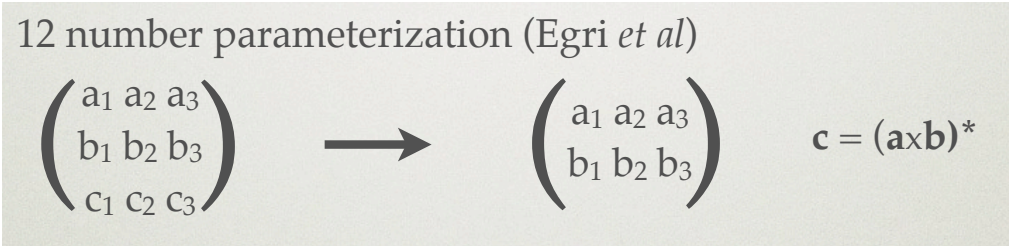
$\psi_{\alpha i}(x)$ 4x3 complex (spin-color), residing on sites

All toghether, 1320 FP on (9x12+8x9 complex) 360 FP words per site.

What do we learn in general from QCD? (II)

LQCD is a good place to point out a couple of strategies often useful in our problems

- We have to rely on a HUGE Flops/Bandwith ratio, so a typical trick is to get slimmer data trading data for computations



Bytes 1440 --> 1152

Flops += 384

- SP/DP ... precision is an issue! GPU slow down quite a lot

MULTIPRECISION schemes ...

a short Summary

- Only prototypes computations presented, to set up a picture ...
- ... so first of all we add: not only lattice and non-lattice; also completely different problems (Feynman graphs!)

- Mainly very efficient number-crunching accelerators
- Notice that we did not refer to scaling to “many-GPU”
- World is not finished with CUDA (AMD Stream; OpenCL; Larrabee is coming!). But also OpenGL is probably not dead...