JÜLICH
FORSCHUNGSZENTRUM

Massively parallelized replica-exchange simulations of polymers on GPUs

11/25/10 CompPhys10 | Jonathan Gross

Outline

1. Part 1: General Purpose Computing on GPUs
2. Part 2: GPU Simulations of an Elastic Polymer Model
3. Part 3: Physical and Technical Results

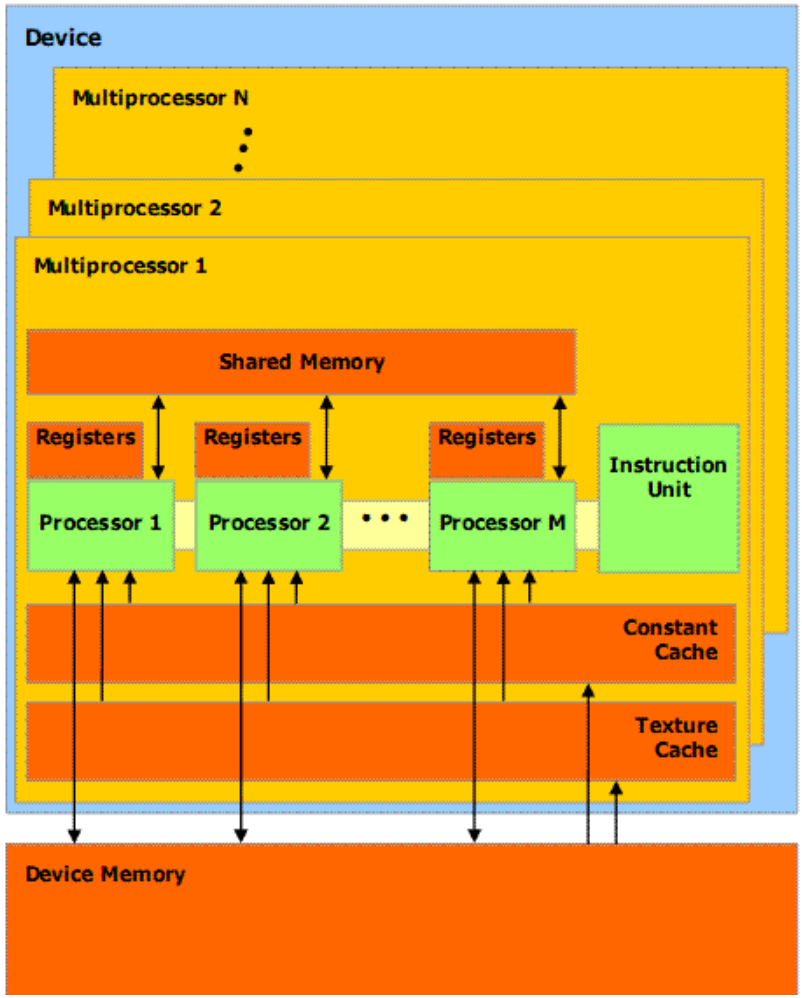
Part 1: General Purpose Computing on GPUs

Parallel Programming Paradigms

1. distributed memory systems
   - cluster computers
   - explicit communication through message passing
   - need fast network
2. shared memory systems
   - multicore CPUs
   - implicit communication through shared memory (fast)
   - multithreaded programming
   - MPI also possible

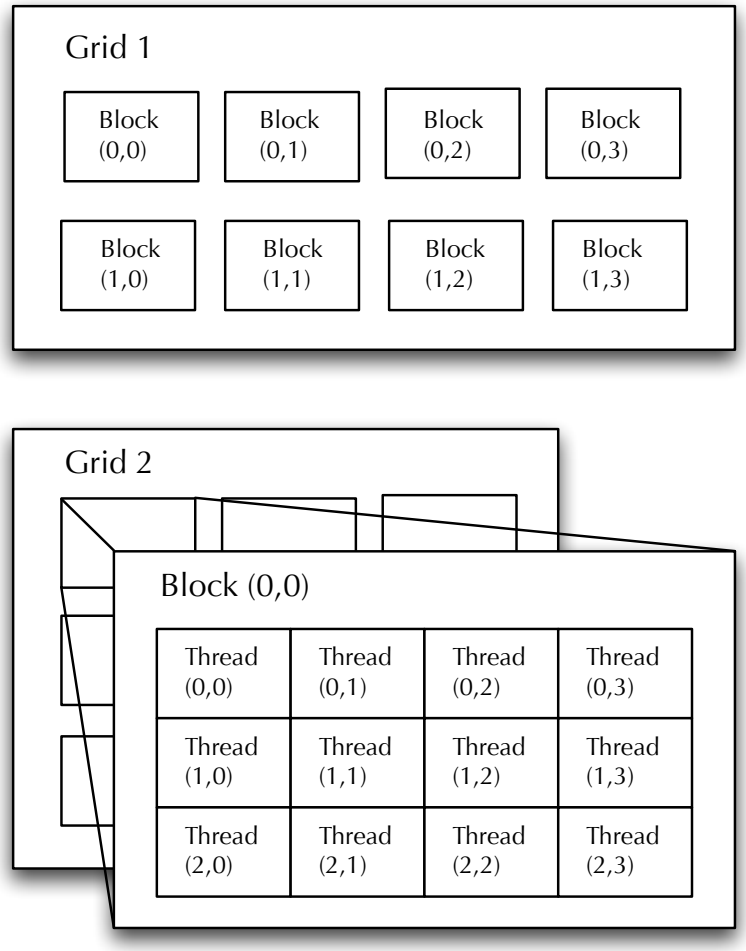
GPU Architecture

- modern GPUs have many cores (≈ 500)
- each capable of processing a vast number of threads
- device memory (up to 4GB) is globally accessible
- each multiprocessor has its own very fast cache memory



Grids and Threads

- max grid dimension 2D
- thread blocks can be arranged in up to 3 dimensions
- threads have a unique id
- 512x512x64 but max, 512 threads per block on GT200
- 1024x1024x64 and 1024 max. threads on Fermi

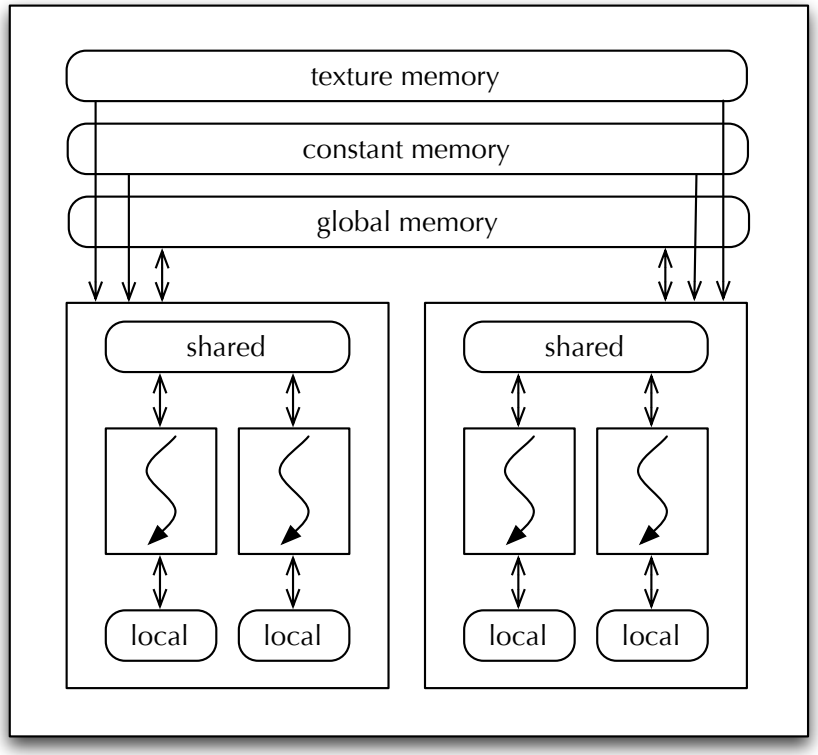


Thread blocks and Warps

- 32 threads of *one* thread block $=$ *warp*
- transparent scalability (scheduling of warps to multiprocessors)
- up to 48 simultaneous active warps on one multiprocessor (equals 1536 threads)
- possibility to gather warps from 8 different thread blocks

GPU memory layout

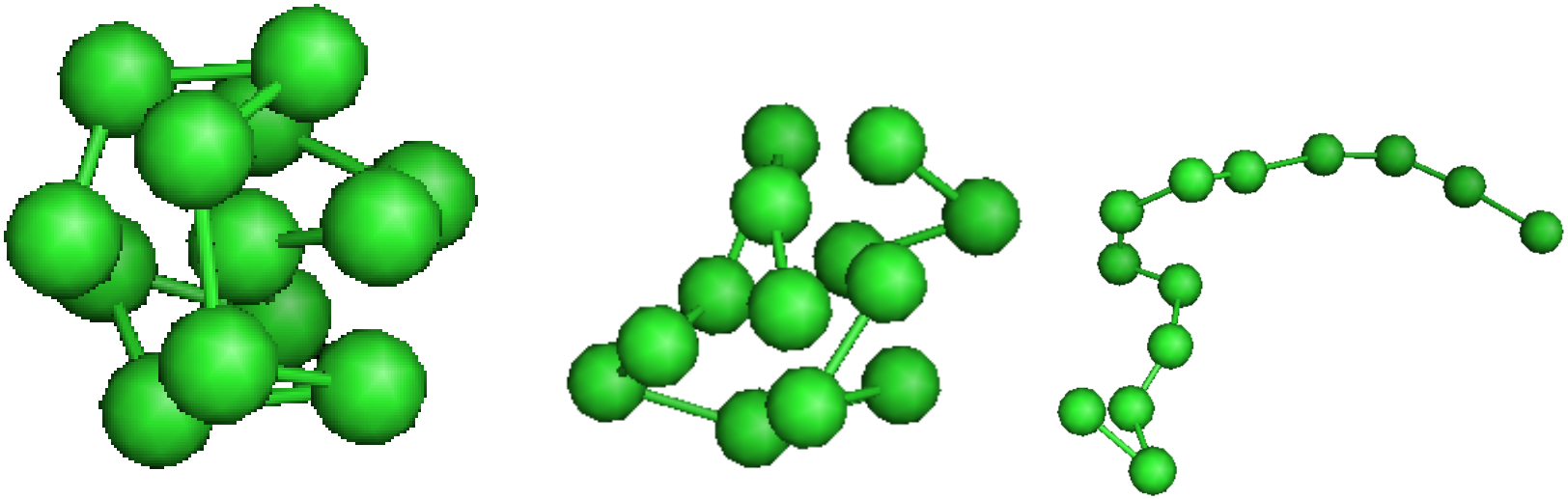
- global memory (read/write access from every thread)
- constant and texture memory (read-only per grid)
- shared memory (read/write access per thread-block)
- local memory (read/write access per thread only)



What is CUDA?

- API for nVIDIA GPUs
- extension to C/C++ (Fortran compiler avail.)
  - function callers
  - memory management
  - intrinsic functions
- program sequence
  1. allocate memory on device
  2. copy data to device
  3. run kernel function
  4. copy data back to host

Part 2: GPU Simulations of an Elastic Polymer Model



A Simple Model for a Flexible Homopolymer

- Lennard-Jones potential for interaction of all monomers
- FENE potential between bonded monomers

$$E_{\mathrm{LJ}}^{\mathrm{mod}}(r_{ij}) = E_{\mathrm{LJ}}(\min(r_{ij}, r_c)) - E_{\mathrm{LJ}}(r_c)$$

$$E_{\mathrm{LJ}}(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^{6} \right]$$

$$E_{\mathrm{FENE}}(r_{ii+1}) = -\frac{K}{2} R^2 \log \left[1 - \left(\frac{r_{ii+1} - r_0}{R} \right)^2 \right]$$

$$E(C) = \sum_{i<j}^{N} E_{\mathrm{LJ}}^{\mathrm{mod}}(r_{ij}) + \sum_{i}^{N-1} E_{\mathrm{FENE}}(r_{ii+1}).$$

$\epsilon = 1$, $r_0 = 0.7$, $\sigma = 2^{-1/6} r_0$, $r_c = 2.5\sigma$

Metropolis Update

1. calculate energy of conformation $\rightarrow E_{old}$
2. pick random monomer
3. perform a random shift of each cartesian coordinate within the interval $[-0.01, 0.01]$
4. calculate energy of conformation $\rightarrow E_{new}$
5. accept update with probability
   - $w(C^{old} \rightarrow C^{new}) = min(1, e^{-\beta(E_{new} - E_{old})})$

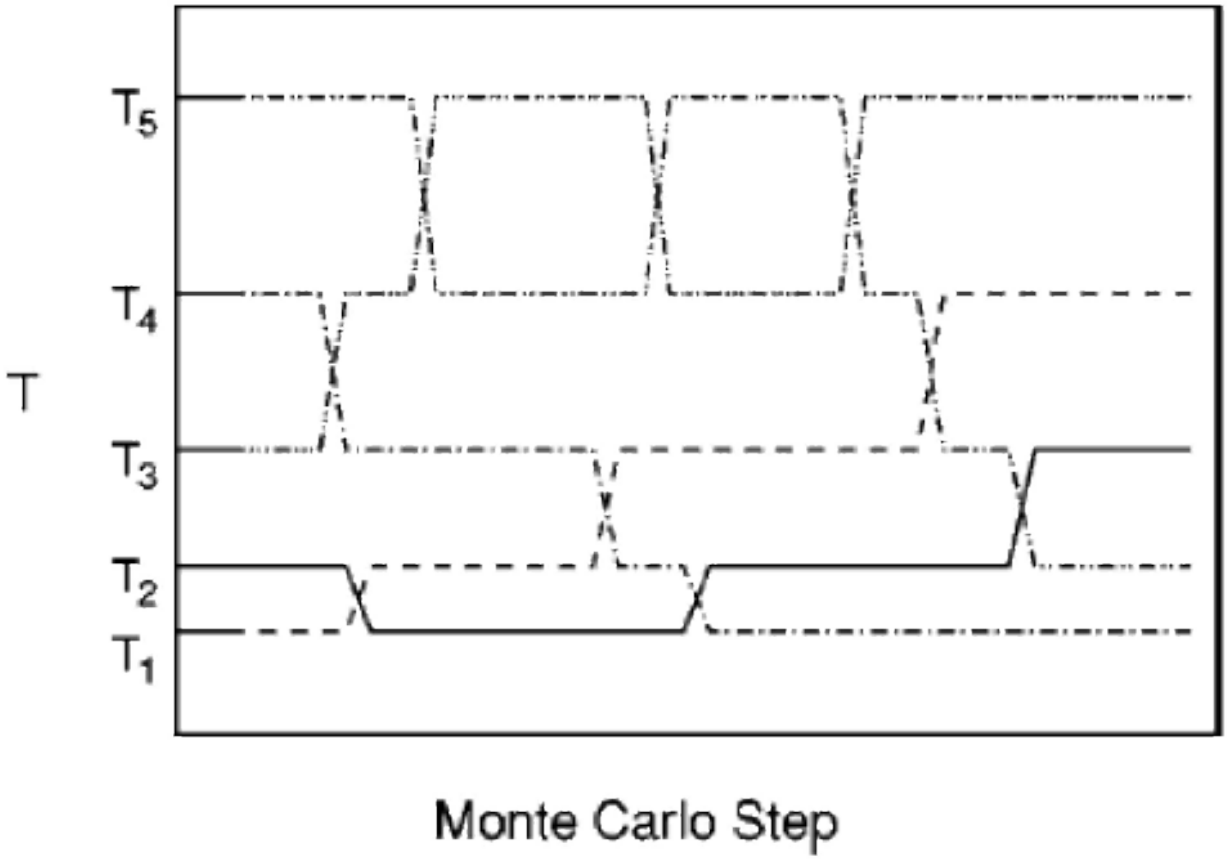
Parallel Tempering

n copies of the system: $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$

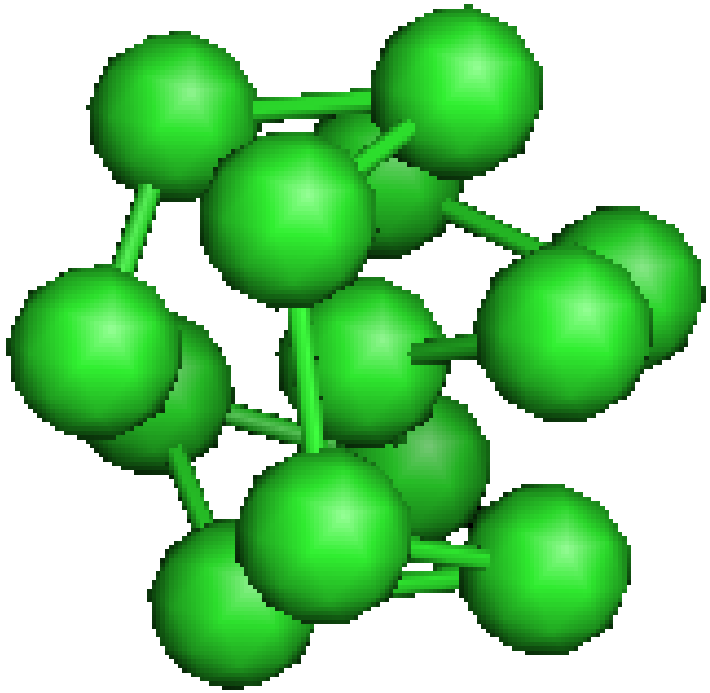
1. Metropolis update like before:
   - $w(\mathcal{C}^{old} \rightarrow \mathcal{C}^{new}) = w(C_i^{old} \rightarrow C_i^{new}) = min(1, e^{-\beta_i \Delta E_i})$

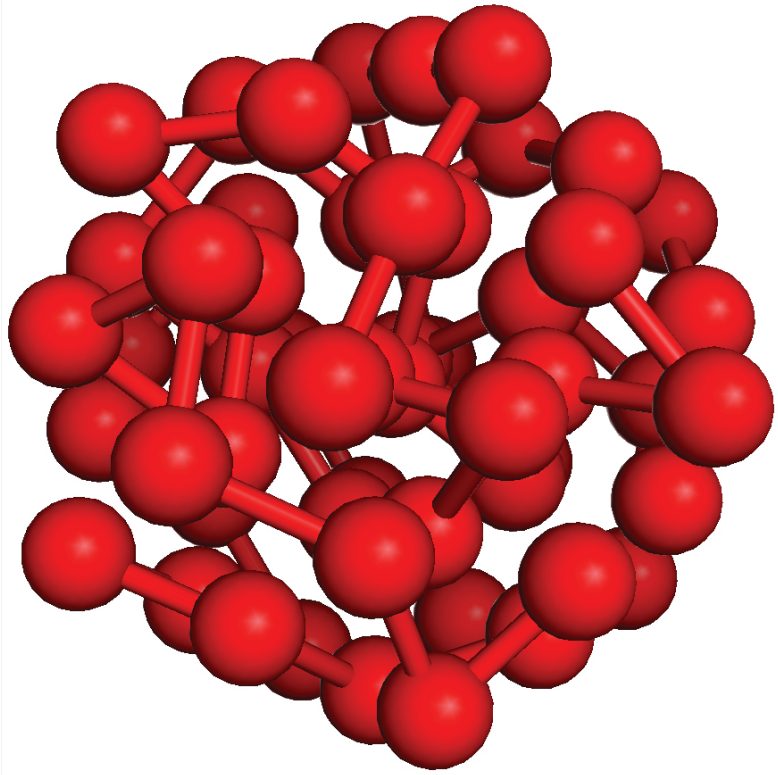
2. exchange conformations between copies i and j
   - $w(\mathcal{C}^{old} \rightarrow \mathcal{C}^{new}) = min(1, e^{\Delta\beta\Delta E})$
   - with $\Delta\beta = \beta_j - \beta_i$ and $\Delta E = E(C_j) - E(C_i)$

Time Series of Temperatures



Part 3: Physical and Technical Results





Simulation Parameters

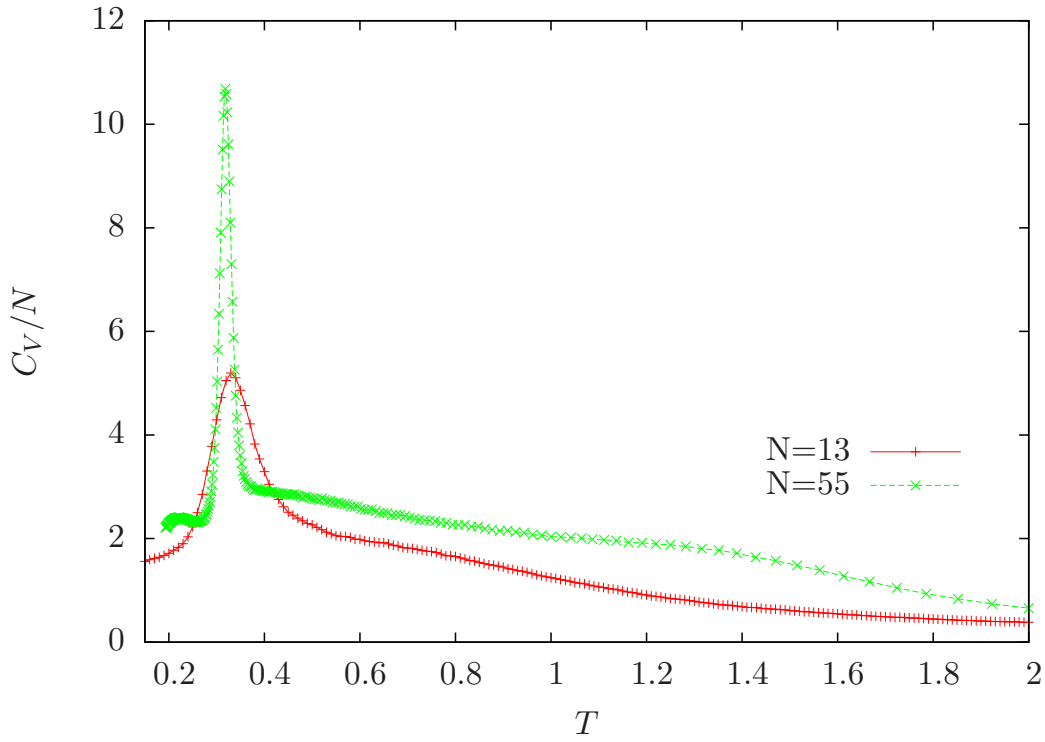
- simulated two system sizes: $N = 13$ and $N = 55$
- 240 different temperatures
- replica-exchange every 1000 sweeps

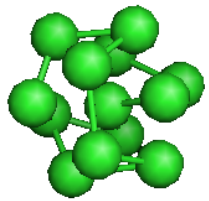
	reference CPU	GPU1	GPU2	GPU3
name	Xeon E5620	Tesla C1060	GTX285	GTX480
# processors	1	30	30	15
cores/processor	4 (only 1 used)	8	8	32
RAM	16384MB	4096MB	1024MB	1536MB
clock speed	2.4GHz	1.3GHz	1.48GHz	1.4GHz
shared mem.	-	16kB	16KB	48kB

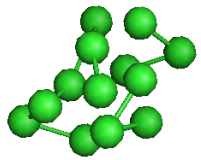
Table: Used hardware

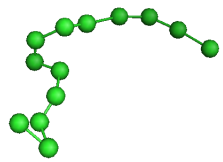
Results

Specific Heat





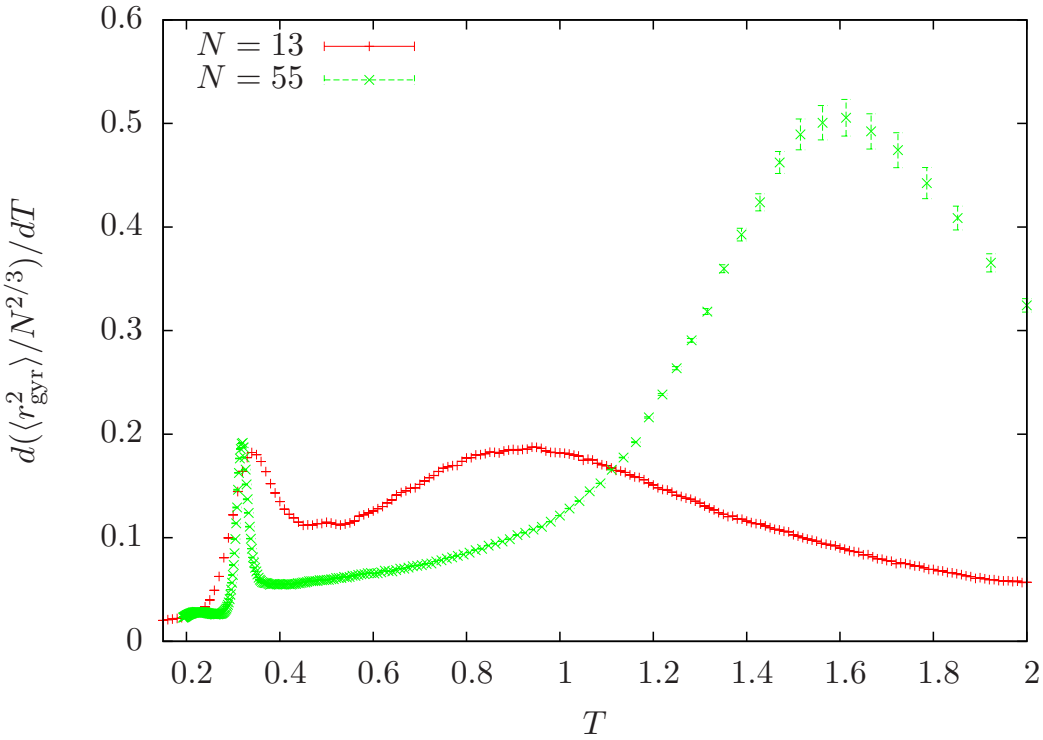




$$\frac{C_V}{N} = \frac{1}{N}\frac{\partial\langle E\rangle}{\partial T} = \frac{\beta^2}{N}(\langle E^2\rangle - \langle E\rangle^2)$$

Results

Radius of Gyration



$$\frac{\partial \langle r^2_{gyr} \rangle}{\partial T} = \beta^2 (\langle r^2_{gyr} \cdot E \rangle - \langle r^2_{gyr} \rangle \cdot \langle E \rangle)$$

Speed-up and Runtime Measurement

- runtimes measured with **cutil**-timers
- same accuracy for CPU and GPU times
- CPU: only actual calculation measured
- GPU: calculation + memory transactions

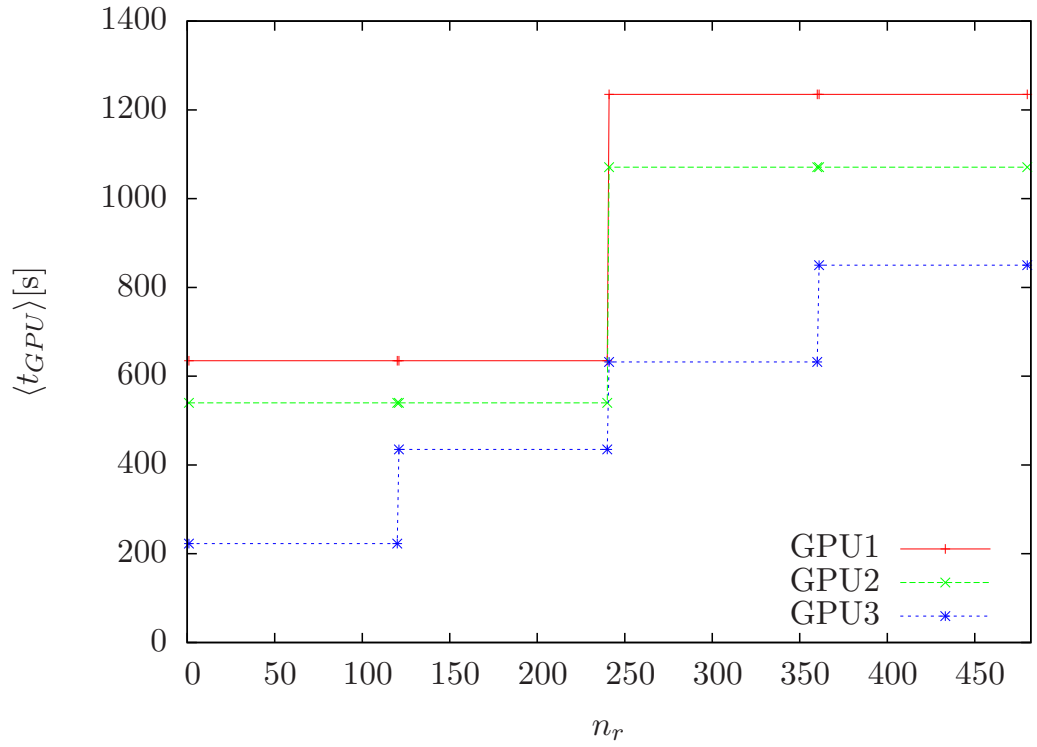
definition: speed-up factor

$$S_p = \frac{t_{\mathrm{CPU}}}{t_{\mathrm{GPU}}}$$

Runtimes on GPUs

first naive implementation

GPU1 = Tesla C1060, GPU2 = GTX285, GPU3 = GTX480

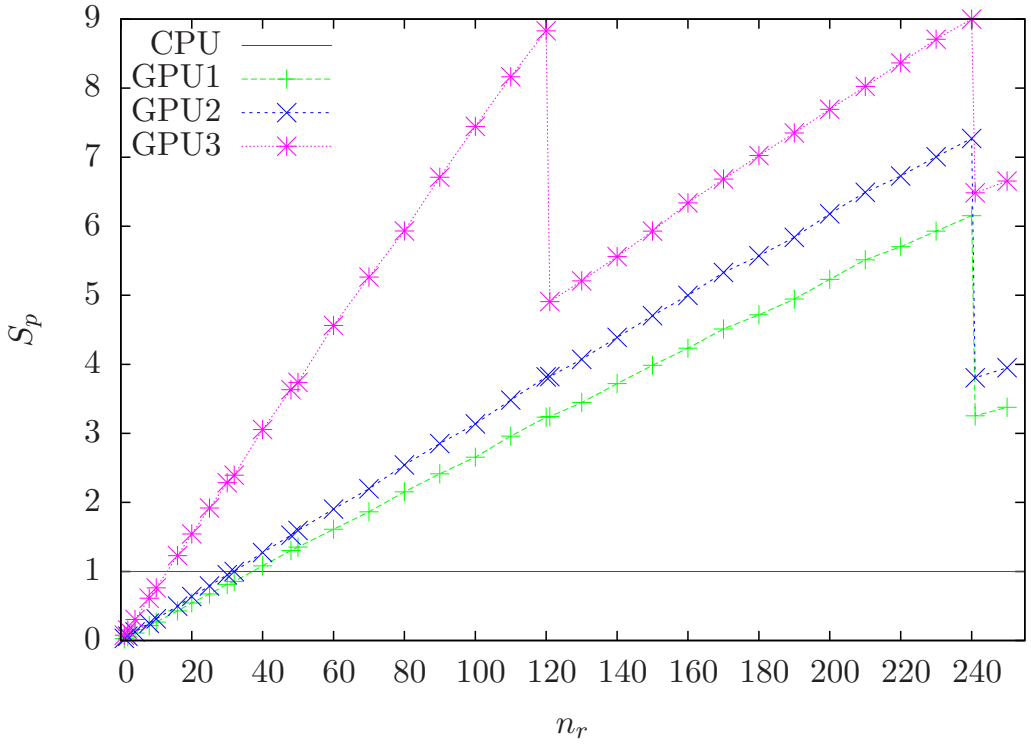


- 1 thread per thread block
- 8 blocks are grouped together

Speed-up

first naive implementation

GPU1 = Tesla C1060, GPU2 = GTX285, GPU3 = GTX480, CPU = Core i7 2.4GHz



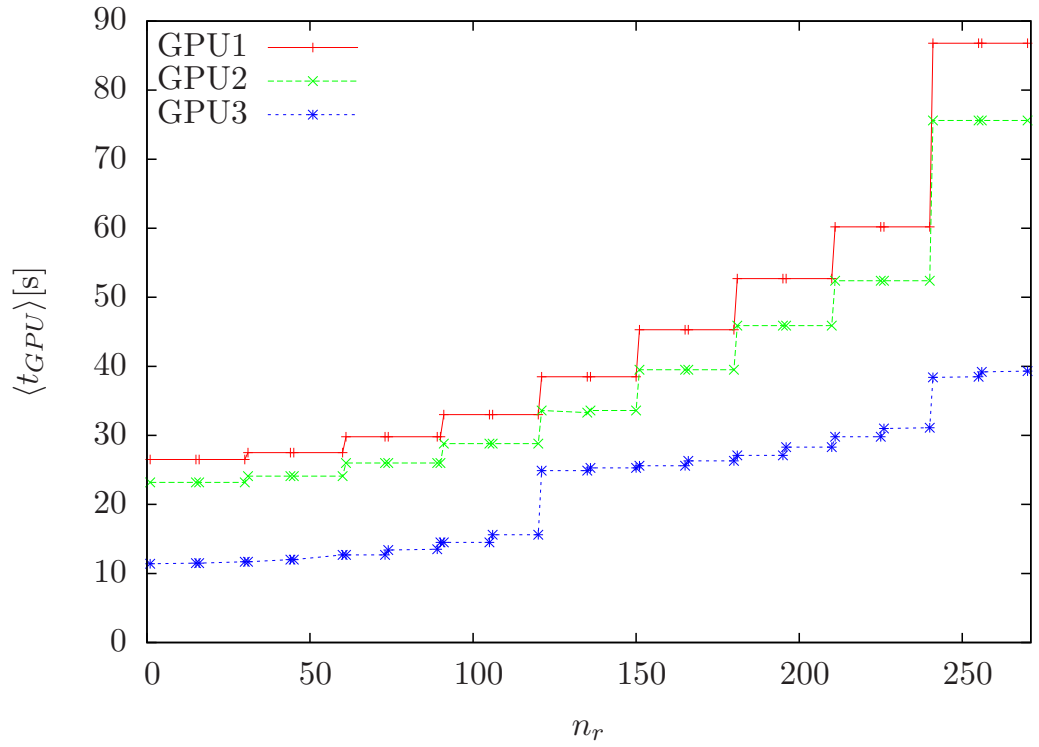
max. speed-up factors:

- C1060: 6.1×
- GTX285: 7.2×
- GTX480: 9×

Runtimes on GPUs

improved implementation

GPU1 = Tesla C1060, GPU2 = GTX285, GPU3 = GTX480

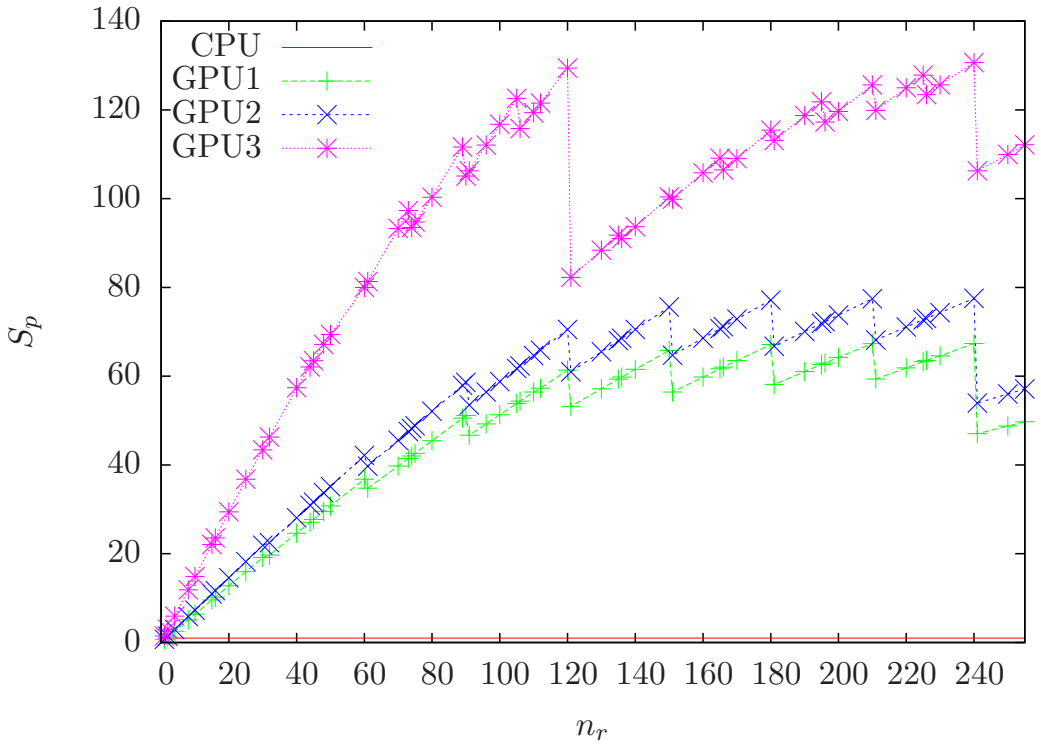


- 64 threads per thread block
- 8 blocks are grouped together
- 512 threads per multiprocessor

Speed-up

improved implementation

GPU1 = Tesla C1060, GPU2 = GTX285, GPU3 = GTX480, CPU = Core i7 2.4GHz



max. speed-up factors:

- C1060: 68×
- GTX285: 78×
- GTX480: 130×

Speed-up Summary

GPU1 = Tesla C1060, GPU2 = GTX285, GPU3 = GTX480

	naive	improved
GPU1	6.1×	68×
GPU2	7.2×	78×
GPU3	9×	**130**×

Table: maximum achieved speed-ups – $\max\left(S_p(n_r)\right)$ – for the two different GPU implementations, compared to the single core CPU implementation.

Thanks to Michael Bachmann and Wolfhard Janke
and thank you for your attention!

Get started with CUDA:
www.nvidia.com/cuda