Characterizing scientific application performance on GPUs

Presenter: Base Paul

Date: 19 December 2007

Motivation

- GPUs have evolved significantly
  - Highly parallel, and programmable too.
  - 10x raw computational performance over CPUs
  - Great potential to be a co-processor
- Most scientific computations are
  - about number crunching – GPUs excel there
  - data parallel – GPUs excel there too.
- But, GPU Programming is not easy
  - Programming model designed primarily for graphics
  - GPU cant directly access data from main memory
  - Accuracy of computation has been low

The project

- Is about exploring the use of GPUs for scientific workloads
  - Programming environments
  - Application work loads – Ocean and Barnes

- Characterizing the application performance
  - Depending on memory access patterns
  - Depending on amount of resources available

The project

- Understand the suitability of GPU for an application (or the other way)
- Identify a programming environment
- Create some benchmarking code based on well-known workloads
- Understand
  - raw performance
  - “end-to-end” performance
  - suitability as a “co-processor”
  - accuracy of the computation

Programming Environment

- Most existing models are difficult
  - not for generic programming and require graphics programming background
- Nvidia's Cuda is different
  - No graphics/shader programming skills required
  - Write a 'kernel' program in C, and run it on the data over and over on the device
  - Fairly good control on tweaking parameters (threads, memory, etc)
  - No synchronization between programs running on GPU and CPU, and no recursion ☹

Benchmarking

- Applications picked based on memory access patterns
  - Ocean – localized memory access
  - Barnes – distributed memory access
- Raw performance
  - Pick a data set that fits in cache. Compare between CPU and GPU
- End-to-end performance
  - Pick a data set that does not fit in cache. Compare between CPU and GPU
  - Vary data set size on GPU – from “fits in memory” to “does not fit in memory” – characterize the performance on either side of the breaking point

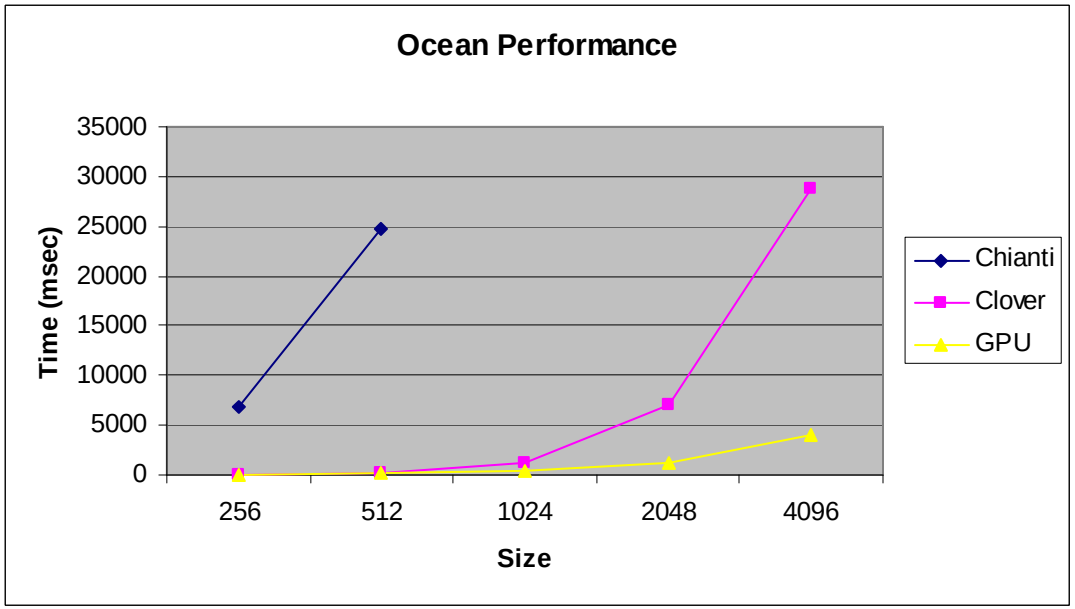
Benchmarking Contd..

- Suitability as co-processor
  - Measure CPU load during the computation on GPU, compare against CPU load during the computation on CPU
  - Measure system memory usage during the computation, compare against that during the computation on CPU
- Accuracy
  - Compare results of floating point arithmetic operations and those of functions like sin, log, exp, sqrt, etc to that on CPU

Results

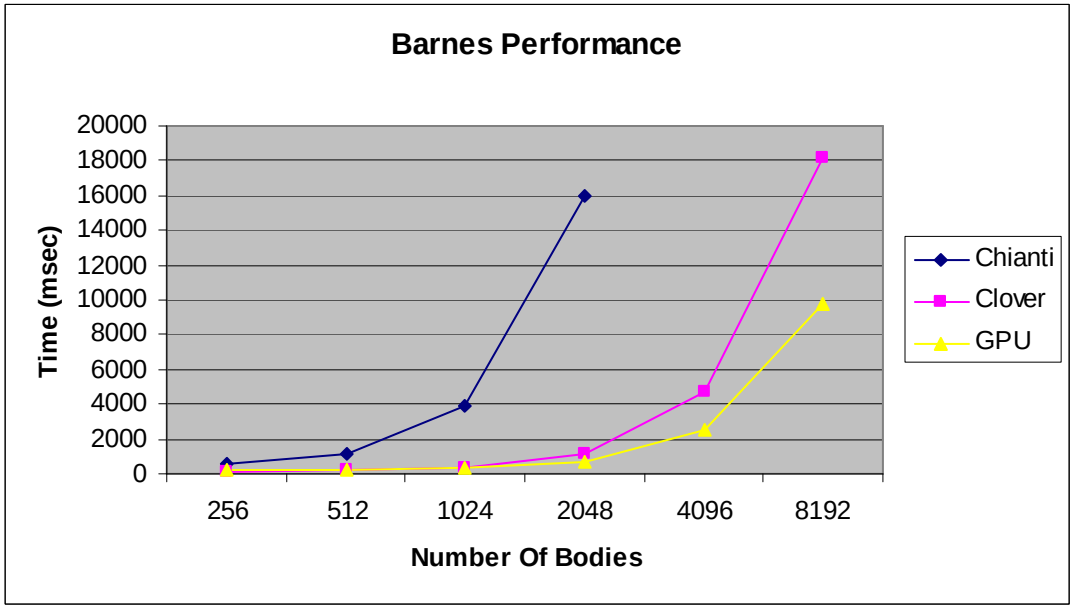
- Cuda was used as the programming environment
  - Easy to program; focus only on application
  - Used Dual Xeon 2.8GHz with nVidia GeForce 8800GTX (768MB)
- Ocean and Barnes implemented
  - Better performance than 8-core Clover nodes
  - Leaves all but one CPU cores idle
  - Accuracy close to CPU (within 6 decimal places)

Results – Contd..

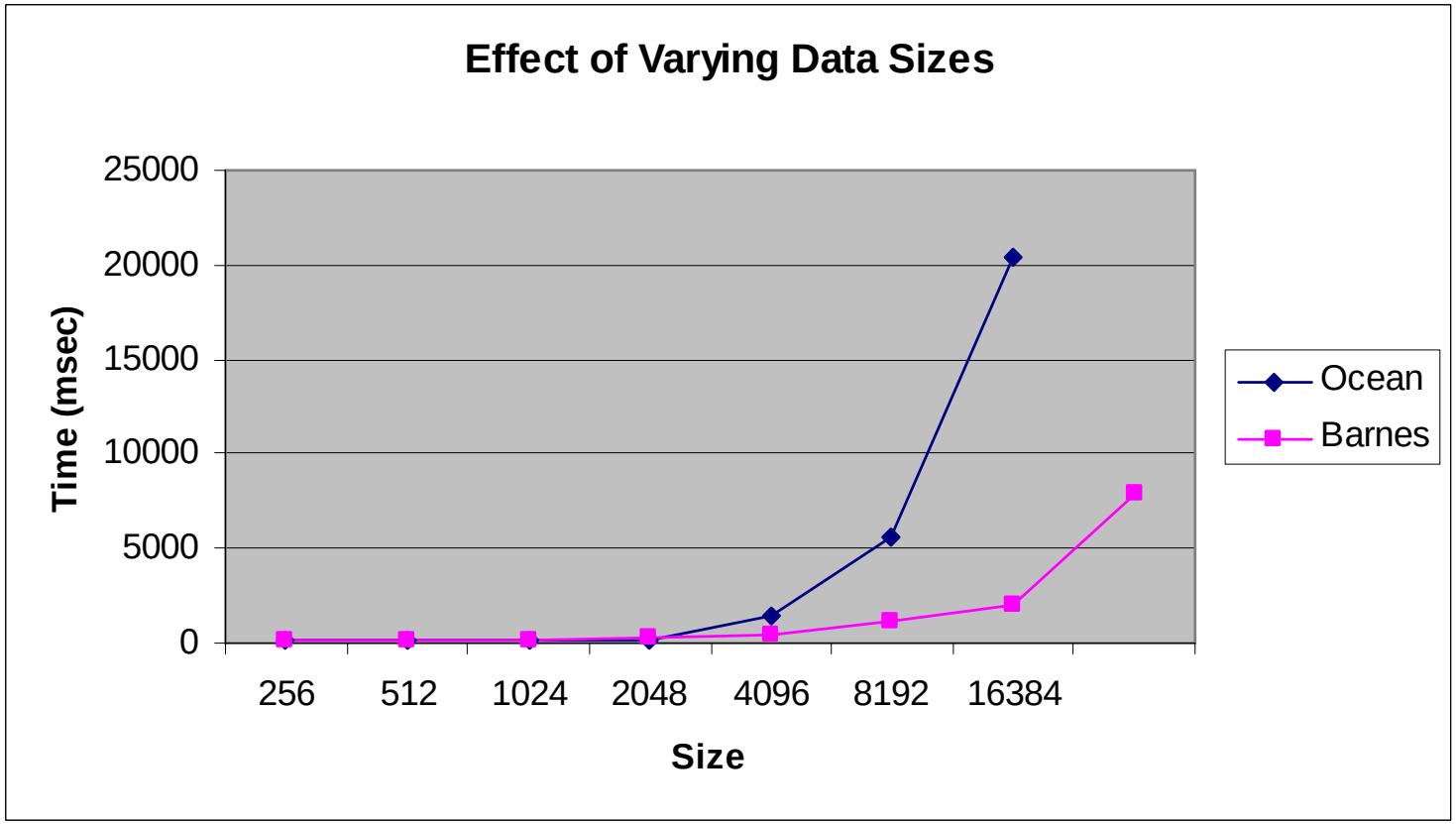


- GPU 7x faster for 4096x4096 ocean – use of "shared memory"

- GPU only 2x faster – memory access pattern



Results – Contd..



- Performance start degrading when the memory limit is hit
- Ocean hits the limit faster (because of O(N^2) space requirements)
- Slopes are misleading – 4x size increase in Ocean => 4x time, while Barnes has 2x size increase => 4x time.. Thanks to memory access pattern

Results Contd.

- Raw performance of GPU found to be only ~2x of Clover node, and ~5x of Chianti for these applications
- CPU load consistent during GPU operations – 1 core ~100%, all others idle
- Memory used depends on data set size – if data can fit in GPU memory, memory can be freed on CPU side.
- Accuracy is within 6 decimal places for the applications tested (for 'float' type)
  - Some operations (like trigonometric, exponents) do not seem to have as good accuracy (within 3 decimal places)

Conclusions

- GPUs hold promise
  - Competes with high-end CPUs at fraction of the cost
  - Accuracy still a question mark for some operations
- Cuda has proved that CPU-like programming flexibility is possible on GPUs..
  - More options are now “work in progress” – Intel Larrabee for e.g.
- GPU is still not for all applications – you must know the applications well (algorithm, inter-dependency between data, memory access pattern, etc)
  - but then, it is a lot better than a year ago when you had to be a graphics wizard too.

Questions?