High Performance Computing Modernization Program (HPCMP) Summer 2011 Puerto Rico Workshop on Intermediate Parallel Programming & Cluster Computing

in conjunction with
the National Computational Science Institute (NCSI)/ SC11 Conference

S.V. Providence

Jointly hosted at
Polytechnic U of Puerto Rico
and U Oklahoma
and available live via videoconferencing
(streaming video recordings coming soon)

Sponsored by
DOD HPCMP,
SC11/ACM,
NCSI and
OK EPSCoR

HPCMP

PUPR-IPPCC 2011
LN2: CUDA Overview

S.V. Providence

Department of Computer Science
Hampton University
Hampton, Virginia 23668
stephen.providence@hamptonu.edu
(757)728-6406 (voice mail)

Polytechnic University of Puerto Rico
Intermediate Parallel Programming & Cluster Computing,
Thursday, Aug. 4th, 2011

Outline

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - O(n^2) version
  - Fast O($n \log n$) version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - O(n^2) version
  - Fast O($n \log n$) version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - N-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - O(n^2) version
  - Fast O($n \log n$) version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - N-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - N-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

Outline

- Introduction
  - *N*-body problem
- Core Methods
- Algorithms and Implementations
  - $O(n^2)$ version
  - Fast $O(n \log n)$ version [Barnes-Hut]
- Conclusion
  - Inputs
  - System
  - Compilers
  - Metric
  - Validation

- P2P - point to point: taken pairwise is order n^2
  - start with the first of n points and form pairs with the other $n - 1$ points
  - perform this step for all n point
  - asymptotically n^2 pairings are formed - dipoles
  - this does not consider tri-poles, quad-poles upto multi-poles
- there are basically two categories:
  - macro: massive objects $>10^{10}$, galaxies and cosmological phenomena - Einstein (relativity; gravity)
  - micro: small objects $<10^{-10}$, quarks, and nano-scale phenomena - Dirac (quantum mechanics; strong, weak & EM forces)

- P2P - point to point: taken pairwise is order n^2
  - start with the first of n points and form pairs with the other $n-1$ points
  - perform this step for all n point
  - asymptotically n^2 pairings are formed - dipoles
  - this does not consider tri-poles, quad-poles upto multi-poles
- there are basically two categories:
  - macro: massive objects $>10^{10}$, galaxies and cosmological phenomena - Einstein (relativity; gravity)
  - micro: small objects $<10^{-10}$, quarks, and nano-scale phenomena - Dirac (quantum mechanics; strong, weak & EM forces)

- P2P - point to point: taken pairwise is order n^2
  - start with the first of n points and form pairs with the other $n-1$ points
  - perform this step for all n point
  - asymptotically n^2 pairings are formed - dipoles
  - this does not consider tri-poles, quad-poles upto multi-poles
- there are basically two categories:
  - macro: massive objects $>10^{10}$, galaxies and cosmological phenomena - Einstein (relativity; gravity)
  - micro: small objects $<10^{-10}$, quarks, and nano-scale phenomena - Dirac (quantum mechanics; strong, weak & EM forces)

- O(n^2) requires straightforward translation of simple data structures to arrays for BLAS 1, 2 computations
- O($n \log n$) is challenging
  1. CON: repeatedly builds and traverses (in-order) an irregular tree-based data structure
  2. CON: performs a great deal of pointer chasing memory operations
  3. CON: the Barnes-Hut approach is typically expressed recursively
  4. PRO: this approach makes interesting problem sizes computationally tractable
  5. PRO: GPUs can be used ti accelerate irregular problems

- $O(n^2)$ requires straightforward translation of simple data structures to arrays for BLAS 1, 2 computations
- $O(n \log n)$ is challenging
  1. CON: repeatedly builds and traverses (in-order) an irregular tree-based data structure
  2. CON: performs a great deal of pointer chasing memory operations
  3. CON: the Barnes-Hut approach is typically expressed recursively
  4. PRO: this approach makes interesting problem sizes computationally tractable
  5. PRO: GPUs can be used ti accelerate irregular problems

- $O(n^2)$ requires straightforward translation of simple data structures to arrays for BLAS 1, 2 computations
- $O(n \log n)$ is challenging
    1. CON: repeatedly builds and traverses (in-order) an irregular tree-based data structure
    2. CON: performs a great deal of pointer chasing memory operations
    3. CON: the Barnes-Hut approach is typically expressed recursively
    4. PRO: this approach makes interesting problem sizes computationally tractable
    5. PRO: GPUs can be used ti accelerate irregular problems

Introduction
approaches

- $O(n^2)$ requires straightforward translation of simple data structures to arrays for BLAS 1, 2 computations
- $O(n \log n)$ is challenging
  1. CON: repeatedly builds and traverses (in-order) an irregular tree-based data structure
  2. CON: performs a great deal of pointer chasing memory operations
  3. CON: the Barnes-Hut approach is typically expressed recursively
  4. PRO: this approach makes interesting problem sizes computationally tractable
  5. PRO: GPUs can be used ti accelerate irregular problems

Core Methods
Barnes-Hut

- widely used, starts with all N bodies in a computational box
- hierarchically decomposes the space around the bodies into successively smaller boxes called cels
- hierarchical decomposition is recorded in octrees (3D equivalent to binary tree), resembles tic-tac-toe grid

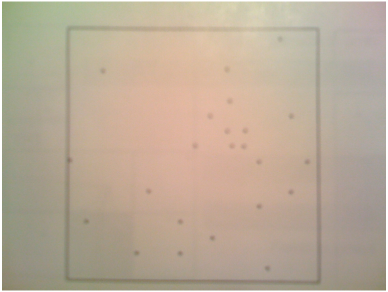

Figure: GPU Gems: Emerald Edition

Core Methods
Barnes-Hut

- widely used, starts with all *N* bodies in a computational box
- hierarchically decomposes the space around the bodies into successively smaller boxes called cels
- hierarchical decomposition is recorded in octrees (3D equivalent to binary tree), resembles tic-tac-toe grid

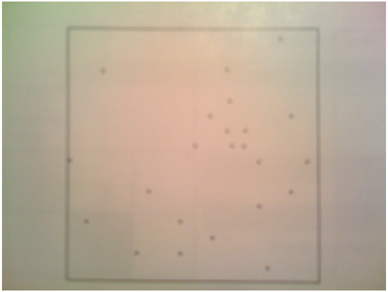

Figure: GPU Gems: Emerald Edition

Core Methods
pseudo code

1. read input data and transfer to GPU

   for each time step do {

   1. compute computational box around all bodies
   2. build hierarchical decomposition by inserting each body into octree
   3. summarize body information in each internal octree node
   4. approximately sort the bodies by spatial distance
   5. compute forces acting on each body with help of ochre
   6. update body positions and velocity

   }
2. transfer result to CPU and output

Core Methods

pseudo code

1. read input data and transfer to GPU
   for each time step do {
   1. compute computational box around all bodies
   2. build hierarchical decomposition by inserting each body into octree
   3. summarize body information in each internal octree node
   4. approximately sort the bodies by spatial distance
   5. compute forces acting on each body with help of ochre
   6. update body positions and velocity

   }
2. transfer result to CPU and output

Core Methods
pseudo code

1. read input data and transfer to GPU
   for each time step do {
   1. compute computational box around all bodies
   2. build hierarchical decomposition by inserting each body into octree
   3. summarize body information in each internal octree node
   4. approximately sort the bodies by spatial distance
   5. compute forces acting on each body with help of ochre
   6. update body positions and velocity

   }
2. transfer result to CPU and output

1. read input data and transfer to GPU
   for each time step do {
   1. compute computational box around all bodies
   2. build hierarchical decomposition by inserting each body into octree
   3. summarize body information in each internal octree node
   4. approximately sort the bodies by spatial distance
   5. compute forces acting on each body with help of ochre
   6. update body positions and velocity

   }
2. transfer result to CPU and output

Core Methods
hierarchical decomposition

- 2D view

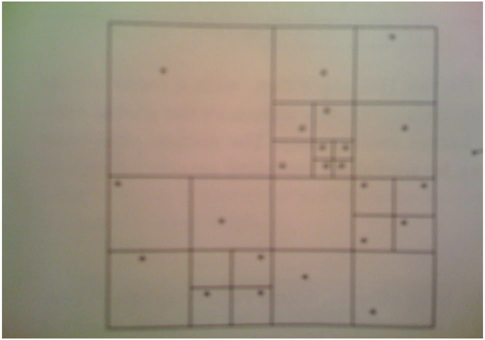

Figure: GPU Gems: Emerald Edition

- 2D view

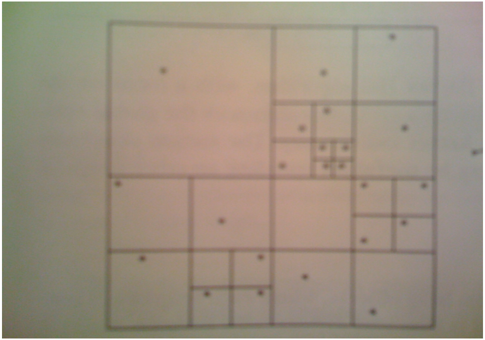

Figure: GPU Gems: Emerald Edition

- tree view

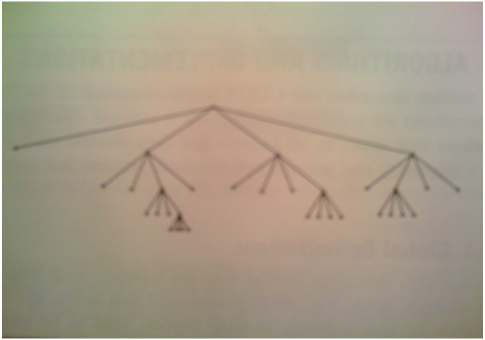

Figure: GPU Gems: Emerald Edition

- tree view

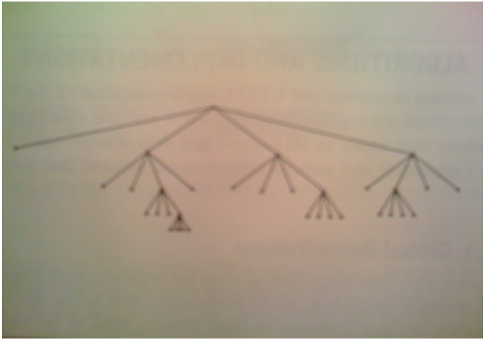

Figure: GPU Gems: Emerald Edition

Core Methods
hierarchical decomposition

- center of gravity (or forces)
- force calculation depicted

Figure: GPU Gems: Emerald Edition

- center of gravity (or forces)
- force calculation depicted

Figure: GPU Gems: Emerald Edition

- runtime per simulated tome step in milliseconds of GPU $O(n^2)$ vs. GPU Barnes-Hut vs. CPU Barnes-Hut

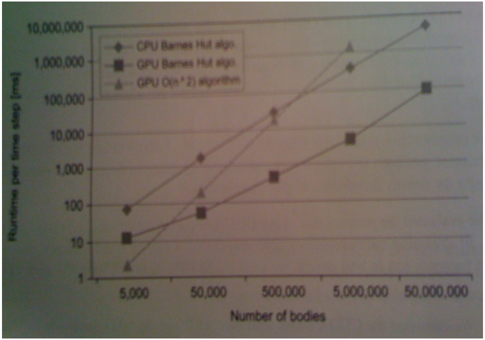

Figure: GPU Gems: Emerald Edition

1. the following expressions for the potential and force, resp. on a body i, and $r_{ij} = x_j - x_i$ is a vec from body i to body j

$$\Phi_i = m_i \sum_{j=1}^{N} \frac{m_j}{r_{ij}}, \quad \mathbf{F}_i = -\nabla \Phi_i$$

this results in $O(n^2)$ computational complexity

2. here the sum for the potential is factored into a near-field and a far-field expansion as follows:

$$\Phi_i = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} m_i r_j^{-n-1} Y_n^m(\theta_i, \phi_i) \sum_{j=1}^{N} \underbrace{m_j \rho_j^n Y_n^{-m}(\alpha_j, \beta_j)}_{M_n^m}.$$

M_n^m clusters particles to far field, Y_n^m is spherical harmonic funct and (r, θ, ϕ); (ρ, α, β) are dist vecs from center of the expansion to bodies i, and j resp.

1. the following expressions for the potential and force, resp. on a body i, and $r_{ij} = x_j - x_i$ is a vec from body i to body j

$$\Phi_i = m_i \sum_{j=1}^{N} \frac{m_j}{r_{ij}}, \quad \mathbf{F}_i = -\nabla \Phi_i$$

   this results in $O(n^2)$ computational complexity
2. here the sum for the potential is factored into a near-field and a far-field expansion as follows:

$$\Phi_i = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} m_i r_j^{-n-1} Y_n^m(\theta_i, \phi_i) \sum_{j=1}^{N} \underbrace{m_j \rho_j^n Y_n^{-m}(\alpha_j, \beta_j)}_{M_n^m}.$$

   M_n^m clusters particles to far field, Y_n^m is spherical harmonic funct and (r, θ, ϕ); (ρ, α, β) are dist vecs from center of the expansion to bodies i, and j resp.

1. here the sum for the potential is factored into a far-field and a near-field expansion as follows:

$$\Phi_i = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} m_i r_i^{-n-1} Y_n^m(\theta_i, \phi_i) \sum_{j=1}^{N} \underbrace{m_j \rho_j^n Y_n^{-m}(\alpha_j, \beta_j)}_{M_n^m} .$$

M_n^m clusters particles to far field, Y_n^m is spherical harmonic funct and (r, θ, ϕ); (ρ, α, β) are dist vecs from center of the expansion to bodies i, and j resp.

$$\Phi_i = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} m_i r_i^n Y_n^m(\theta_i, \phi_i) \sum_{j=1}^{N} \underbrace{m_j \rho_j^{-n-1} Y_n^{-m}(\alpha_j, \beta_j)}_{L_n^m} .$$

L_n^m clusters particles to near field, use of FMM, O(N): fast multipole method and the tree structure list of log N cells interacting with N particles, yields O($N \log N$) complexity.

- flow of tree code and FMM calculation

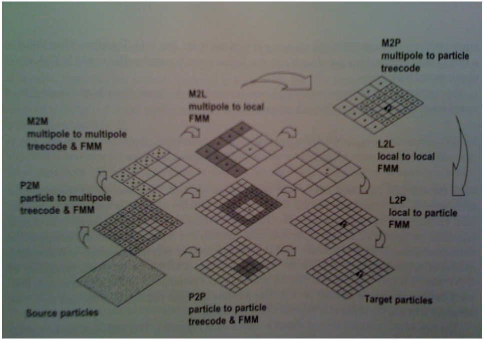

Figure: GPU Gems: Emerald Edition

- flow of tree code and FMM calculation

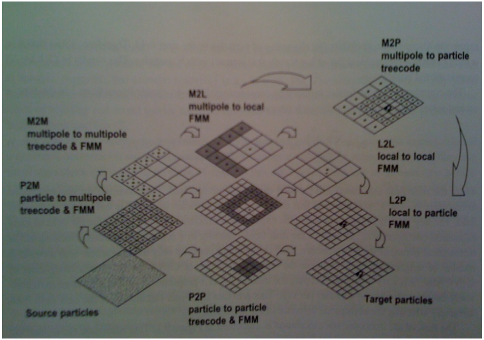

Figure: GPU Gems: Emerald Edition

Conclusion
modest evaluation

- No implementation of an entire N-body algorithm runs on the GPU
- Inputs: 5,000 to 50,000,000 galaxies
- System: 2.53 GHz Xeon E5540 CPU with 12GB RAM per node and !.66 GHz TESLA GPU with 240 cores
- Compilers: CUDA codes with nvcc v.4.1 and the "-O3 -arch=sm_13", "-O2" is used with icc.
- Metric: runtimes appear close together
- Validation: $O(n^2)$ is more accurate than Barnes-Hut algorithms because the CPUs floating point arithmetic is more precise than the GPUs

Conclusion
modest evaluation

- No implementation of an entire *N*-body algorithm runs on the GPU
- Inputs: 5,000 to 50,000,000 galaxies
- System: 2.53 GHz Xeon E5540 CPU with 12GB RAM per node and !.66 GHz TESLA GPU with 240 cores
- Compilers: CUDA codes with nvcc v.4.1 and the "-O3 -arch=sm_13", "-O2" is used with icc.
- Metric: runtimes appear close together
- Validation: O(n^2) is more accurate than Barnes-Hut algorithms because the CPUs floating point arithmetic is more precise than the GPUs

Conclusion
modest evaluation

- No implementation of an entire N-body algorithm runs on the GPU
- Inputs: 5,000 to 50,000,000 galaxies
- System: 2.53 GHz Xeon E5540 CPU with 12GB RAM per node and !.66 GHz TESLA GPU with 240 cores
- Compilers: CUDA codes with nvcc v.4.1 and the "-O3 -arch=sm_13", "-O2" is used with icc.
- Metric: runtimes appear close together
- Validation: $O(n^2)$ is more accurate than Barnes-Hut algorithms because the CPUs floating point arithmetic is more precise than the GPUs

- No implementation of an entire *N*-body algorithm runs on the GPU
- Inputs: 5,000 to 50,000,000 galaxies
- System: 2.53 GHz Xeon E5540 CPU with 12GB RAM per node and !.66 GHz TESLA GPU with 240 cores
- Compilers: CUDA codes with nvcc v.4.1 and the "-O3 -arch=sm_13", "-O2" is used with icc.
- Metric: runtimes appear close together
- Validation: O(n^2) is more accurate than Barnes-Hut algorithms because the CPUs floating point arithmetic is more precise than the GPUs

For Further Reading I

- Michael J. Quinn,
  Parallel Programming in C with MPI and OpenMP
  McGraw-Hill, 2004
- J. Sanders, E. Kandrot,
  CUDA By Example: An Introduction to General-Purpose GPU Programming,
  Nvidia, 2011
- Board of Trustees of the University of Illinois, 2011
  NCSA News,
  http://www.ncsa.ilinois.edu/BlueWaters/systems.html
- B. Sinharoy,et al.
  IBM POWER7 Multicore Server Processor
  IBM J. Res. & Dev. Vol. 55 No. 3 Paper 1 May/June 2011

For Further Reading II

- Jeffrey Vetter, Dick Glassbrook, Jack Dongarra, Richard Fujimoto, Thomas Schulthess, Karsten Schwan
  Keeneland - Enabling Heterogenous Computing for the Open Science Community
  Supercomputing Conference 2010, New Orleans, Louisiana
- C. Zeller, Nvidia Corporation
  C. Zeller - CUDA C Basics
  Supercomputing Conference 2010, New Orleans, Louisiana