Computational Numerical Integration for Spherical Quadratures

Verified by the Boltzmann Equation

Huston Rogers.[1]
Glenn Brook, Mentor[2]
Greg Peterson, Mentor[2]

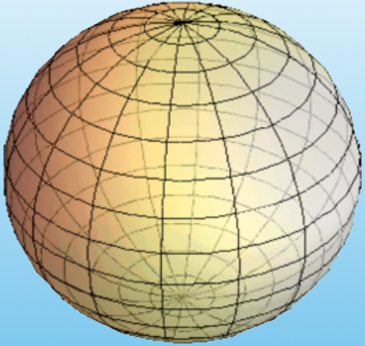
[1]The University of Alabama

[2]Joint Institute for Computational Science
University of Tennessee

Outline

The Basic Problem.
Spherical Quadrature

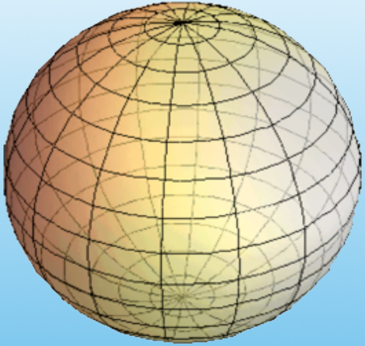
- Spherical Quadratures are more natural to use with the Boltzmann Integrals
- Create grid of pieces of a sphere
- Built around cartesian and spherical integration relationship: $\int \mathrm{d}V_{cart} = \int \rho^2 sin(\theta) \mathrm{d}V_{sphere}$ where $\mathrm{d}V_{cart} = \mathrm{d}x\mathrm{d}y\mathrm{d}z$ and $\mathrm{d}V_{sphere} = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$



The Basic Problem.

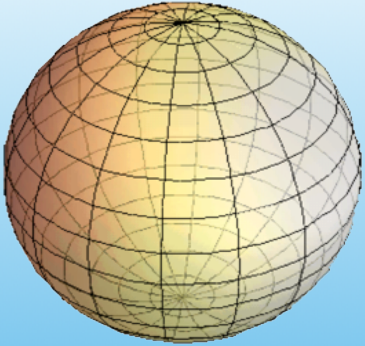
Spherical Quadrature

- Spherical Quadratures are more natural to use with the Boltzmann Integrals
- Create grid of pieces of a sphere
- Built around cartesian and spherical integration relationship: $\int \mathrm{d}V_{cart} = \int \rho^2 sin(\theta) \mathrm{d}V_{sphere}$ where $\mathrm{d}V_{cart} = \mathrm{d}x\mathrm{d}y\mathrm{d}z$ and $\mathrm{d}V_{sphere} = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$



The Basic Problem.
Spherical Quadrature

- Spherical Quadratures are more natural to use with the Boltzmann Integrals
- Create grid of pieces of a sphere
- Built around cartesian and spherical integration relationship:
  $\int \mathrm{d}V_{cart} = \int \rho^2 sin(\theta) \mathrm{d}V_{sphere}$
  where $\mathrm{d}V_{cart} = \mathrm{d}x\mathrm{d}y\mathrm{d}z$
  and $\mathrm{d}V_{sphere} = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$



Numerical quadrature implementation

- Using the trapezoidal rule the Boltzmann integrals are computed; known values, from verified experiment, can be used to check the accuracy of the program.
- For the chosen verified experimental values,the Maxwellian distribution function, in the Boltzmann integrals, is known.
- Due to inter-dependancy the integration was separated into two portions; values from the first were utilized in the second.
- Then we can integrate, using a parallel implementation of the chosen quadrature, to get back the original bulk values
- Grid Refinement can then be used to improve upon the accuracy of the program and to measure convergence of the numerical quadrature method

$$\iiint_S \mathbf{f}\rho^2 sin(\theta)\mathrm{d}V = Density \qquad \frac{\iiint_S \mathbf{f}c^2\rho^2 sin(\theta)\mathrm{d}V}{3{*}Density} = Temperature$$

$$\frac{\iiint_S \mathbf{f}V_i\rho^2 sin(\theta)\mathrm{d}V}{Density} = U_i \qquad c^2 = (V_x - U_x)^2 + (V_y - U_y)^2 + (V_z - U_z)^2$$

$$\mathrm{d}V = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$$

Numerical quadrature implementation

- Using the trapezoidal rule the Boltzmann integrals are computed; known values, from verified experiment, can be used to check the accuracy of the program.
- For the chosen verified experimental values,the Maxwellian distribution function, in the Boltzmann integrals, is known.
- Due to inter-dependancy the integration was separated into two portions; values from the first were utilized in the second.
- Then we can integrate, using a parallel implementation of the chosen quadrature, to get back the original bulk values
- Grid Refinement can then be used to improve upon the accuracy of the program and to measure convergence of the numerical quadrature method

$$\iiint_S \mathbf{f}\rho^2 sin(\theta)\mathrm{d}V = Density$$

$$\frac{\iiint_S \mathbf{f}c^2\rho^2 sin(\theta)\mathrm{d}V}{3*Density} = Temperature$$

$$\frac{\iiint_S \mathbf{f}V_i\rho^2 sin(\theta)\mathrm{d}V}{Density} = U_i$$

$$c^2 = (V_x - U_x)^2 + (V_y - U_y)^2 + (V_z - U_z)^2$$

$$\mathrm{d}V = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$$

Numerical quadrature implementation

- Using the trapezoidal rule the Boltzmann integrals are computed; known values, from verified experiment, can be used to check the accuracy of the program.
- For the chosen verified experimental values,the Maxwellian distribution function, in the Boltzmann integrals, is known.
- Due to inter-dependancy the integration was separated into two portions; values from the first were utilized in the second.
- Then we can integrate, using a parallel implementation of the chosen quadrature, to get back the original bulk values
- Grid Refinement can then be used to improve upon the accuracy of the program and to measure convergence of the numerical quadrature method

$$\iiint_S \mathbf{f}\rho^2 sin(\theta)\mathrm{d}V = Density$$

$$\frac{\iiint_S \mathbf{f}c^2\rho^2 sin(\theta)\mathrm{d}V}{3{*}Density} = Temperature$$

$$\frac{\iiint_S \mathbf{f}V_i\rho^2 sin(\theta)\mathrm{d}V}{Density} = U_i$$

$$c^2 = (V_x - U_x)^2 + (V_y - U_y)^2 + (V_z - U_z)^2$$

$$\mathrm{d}V = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$$

Numerical quadrature implementation

- Using the trapezoidal rule the Boltzmann integrals are computed; known values, from verified experiment, can be used to check the accuracy of the program.
- For the chosen verified experimental values,the Maxwellian distribution function, in the Boltzmann integrals, is known.
- Due to inter-dependancy the integration was separated into two portions; values from the first were utilized in the second.
- Then we can integrate, using a parallel implementation of the chosen quadrature, to get back the original bulk values
- Grid Refinement can then be used to improve upon the accuracy of the program and to measure convergence of the numerical quadrature method

$$\iiint_S \mathbf{f}\rho^2 sin(\theta)\mathrm{d}V = Density \qquad \frac{\iiint_S \mathbf{f}c^2\rho^2 sin(\theta)\mathrm{d}V}{3*Density} = Temperature$$

$$\frac{\iiint_S \mathbf{f}V_i\rho^2 sin(\theta)\mathrm{d}V}{Density} = U_i \qquad c^2 = (V_x - U_x)^2 + (V_y - U_y)^2 + (V_z - U_z)^2$$

$$\mathrm{d}V = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$$

Numerical quadrature implementation

- Using the trapezoidal rule the Boltzmann integrals are computed; known values, from verified experiment, can be used to check the accuracy of the program.
- For the chosen verified experimental values,the Maxwellian distribution function, in the Boltzmann integrals, is known.
- Due to inter-dependancy the integration was separated into two portions; values from the first were utilized in the second.
- Then we can integrate, using a parallel implementation of the chosen quadrature, to get back the original bulk values
- Grid Refinement can then be used to improve upon the accuracy of the program and to measure convergence of the numerical quadrature method

$$\iiint_S \mathbf{f}\rho^2 sin(\theta)\mathrm{d}V = Density \qquad \frac{\iiint_S \mathbf{f}c^2\rho^2 sin(\theta)\mathrm{d}V}{3{*}Density} = Temperature$$

$$\frac{\iiint_S \mathbf{f}V_i\rho^2 sin(\theta)\mathrm{d}V}{Density} = U_i \qquad c^2 = (V_x - U_x)^2 + (V_y - U_y)^2 + (V_z - U_z)^2$$

$$\mathrm{d}V = \mathrm{d}\rho\mathrm{d}\theta\mathrm{d}\phi$$

Code Characteristics

- Code designed to run on Beacon: A next-generation Green 500 supercomputer based on the Intel®Xeon Phi™coprocessor architecture
- Step 1 : Parallelize on the Xeon host processors
  - 8 OpenMP threads, mapped to each of the 8 cores of the Xeon Processor
  - MPI used to distribute processes to multiple processors
  - micmpiexec used with -np flag to specify number of processors
  - MPI_Bcast, MPI_Reduce, and MPI_Allreduce used as necessary to sum needed values and pass them around
- Step 2: Migrate code for use with Intel®Many Integrated Core™architectures

Code Characteristics

- Code designed to run on Beacon: A next-generation Green 500 supercomputer based on the Intel®Xeon Phi™coprocessor architecture
- Step 1 : Parallelize on the Xeon host processors
  - 8 OpenMP threads, mapped to each of the 8 cores of the Xeon Processor
  - MPI used to distribute processes to multiple processors
  - micmpiexec used with -np flag to specify number of processors
  - MPI_Bcast, MPI_Reduce, and MPI_Allreduce used as necessary to sum needed values and pass them around
- Step 2: Migrate code for use with Intel®Many Integrated Core™architectures

Code Characteristics

- Code designed to run on Beacon: A next-generation Green 500 supercomputer based on the Intel®Xeon Phi™coprocessor architecture
- Step 1 : Parallelize on the Xeon host processors
  - 8 OpenMP threads, mapped to each of the 8 cores of the Xeon Processor
  - MPI used to distribute processes to multiple processors
  - micmpiexec used with -np flag to specify number of processors
  - MPI_Bcast, MPI_Reduce, and MPI_Allreduce used as necessary to sum needed values and pass them around
- Step 2: Migrate code for use with Intel®Many Integrated Core™architectures

Pseudocode

- Initialize distribution function, parameters and the underlying mesh
- Assign density and moment integration tasks to MPI ranks, via the mesh decomposition, and begin computations
- Sum and reduce density and moment integral computations, share results among MPI ranks.
- Utilize first stage results to compute the temperature integral on each MPI rank; reduce and sum the result from all ranks.
- Finalize and End

Pseudocode

- Initialize distribution function, parameters and the underlying mesh
- Assign density and moment integration tasks to MPI ranks, via the mesh decomposition, and begin computations
- Sum and reduce density and moment integral computations, share results among MPI ranks.
- Utilize first stage results to compute the temperature integral on each MPI rank; reduce and sum the result from all ranks.
- Finalize and End

Pseudocode

- Initialize distribution function, parameters and the underlying mesh
- Assign density and moment integration tasks to MPI ranks, via the mesh decomposition, and begin computations
- Sum and reduce density and moment integral computations, share results among MPI ranks.
- Utilize first stage results to compute the temperature integral on each MPI rank; reduce and sum the result from all ranks.
- Finalize and End

Pseudocode

- Initialize distribution function, parameters and the underlying mesh
- Assign density and moment integration tasks to MPI ranks, via the mesh decomposition, and begin computations
- Sum and reduce density and moment integral computations, share results among MPI ranks.
- Utilize first stage results to compute the temperature integral on each MPI rank; reduce and sum the result from all ranks.
- Finalize and End

Pseudocode

- Initialize distribution function, parameters and the underlying mesh
- Assign density and moment integration tasks to MPI ranks, via the mesh decomposition, and begin computations
- Sum and reduce density and moment integral computations, share results among MPI ranks.
- Utilize first stage results to compute the temperature integral on each MPI rank; reduce and sum the result from all ranks.
- Finalize and End

Defining Error

- The next step of this project was determining an appropriate method for evaluating error.
- The error in each of the three recordable values (ρ, θ, ϕ) was recorded
- The evaluated error was taken to be the absolute maximum of the error of the three values
- The evaluated error was then analyzed against computer time and relative fineness of grid

Defining Error

- The next step of this project was determining an appropriate method for evaluating error.
- The error in each of the three recordable values (ρ, θ, ϕ) was recorded
- The evaluated error was taken to be the absolute maximum of the error of the three values
- The evaluated error was then analyzed against computer time and relative fineness of grid

Defining Error

- The next step of this project was determining an appropriate method for evaluating error.
- The error in each of the three recordable values (ρ, θ, ϕ) was recorded
- The evaluated error was taken to be the absolute maximum of the error of the three values
- The evaluated error was then analyzed against computer time and relative fineness of grid

Defining Error

- The next step of this project was determining an appropriate method for evaluating error.
- The error in each of the three recordable values (ρ, θ, ϕ) was recorded
- The evaluated error was taken to be the absolute maximum of the error of the three values
- The evaluated error was then analyzed against computer time and relative fineness of grid

Experimental Approach

- We began with an initial grid, then refined in all three directions simultaneously by a factor of 2
- Identified a number of gridpoints, beyond which there was a non-significant decrease in error
- Using this as an upper bound, began refinement in each direction, attempting to find a "sweet spot" for overall error with the lowest grid complexity

Experimental Approach

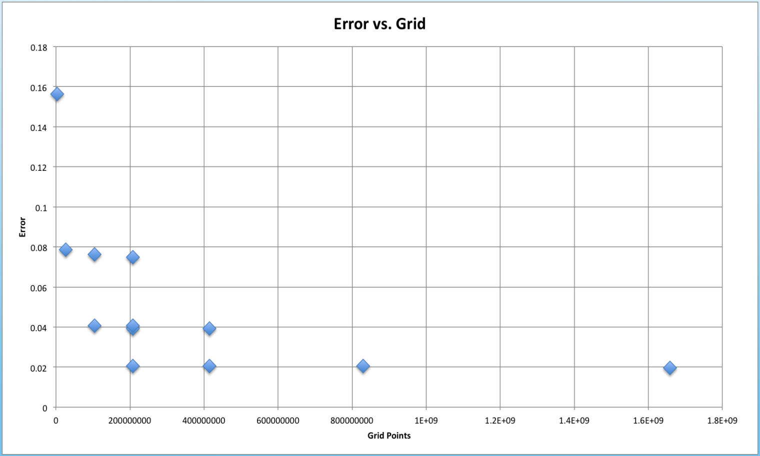
- We began with an initial grid, then refined in all three directions simultaneously by a factor of 2
- Identified a number of gridpoints, beyond which there was a non-significant decrease in error
- Using this as an upper bound, began refinement in each direction, attempting to find a "sweet spot" for overall error with the lowest grid complexity

Experimental Approach

- We began with an initial grid, then refined in all three directions simultaneously by a factor of 2
- Identified a number of gridpoints, beyond which there was a non-significant decrease in error
- Using this as an upper bound, began refinement in each direction, attempting to find a "sweet spot" for overall error with the lowest grid complexity

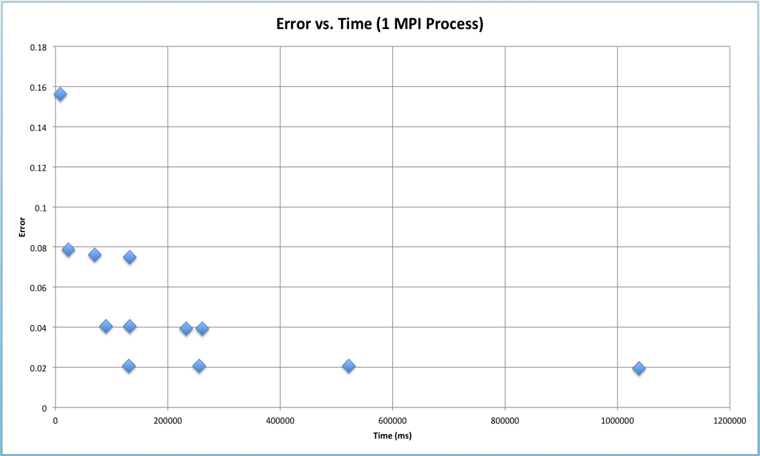
Accuracy vs. Fineness of Grid

Verification through determination of appropriate grid



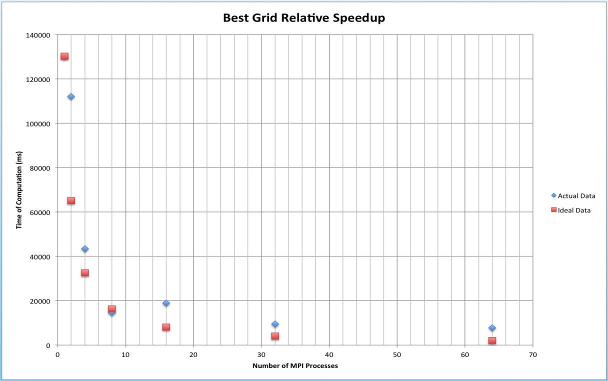
Accuracy vs. Time of Computation

Verification through determination of appropriate grid



Relative MPI Speedup

- The test-case sphere had a radius of 10
- In the best case, the number of gridpoints was (ρ, θ, ϕ) : (100,720,2880)
- The ideal speed up is for every time the number of MPI processes double, the time taken for the computation should halve.



The data shows the point where number of communications made between processes affects time for computation

Further Goals and Applications

- Utilize offload statements to effectively use the Many Integrated Core Architecture of Beacon in the computation
- Further optimize to reduce amount of resources or number of communications required
- Apply code to other style problems that require a spherical quadrature

References and Acknowledgements

- R. G. Brook, ”A parallel, matrix-free newton method for solving approximate boltzmann equations on unstructured topologies,” Ph.D. dissertation, University of Tennessee at Chattanooga, 2008.
- Work was conducted using the National Institute for Computational Sciences compute resources
- Supported in part by The University of Tennessee Knoxville, The Joint Institute for Computational Sciences, Oak Ridge National Lab, and The National Science Foundation