GPU Acceleration of the Longwave Rapid Radiative Transfer Model in WRF using CUDA Fortran

G. Ruetsch, M. Fatica, E. Phillips, N. Juffa
Outline

- WRF and RRTM
- Previous Work
- CUDA Fortran Features
- RRTM in CUDA Fortran
- Results

Hot of the Press

- Batched Solver for Small Matrix Problems
WRF and RRTM

WRF (Weather Research and Forecast) is a mesoscale numerical weather prediction code designed for both operational forecasting and the research community with a broad spectrum of applications and multiple physics options.

Longwave RRTM (Rapid Radiative Transport Model) is an optional model that computes the energy transfer through the atmosphere due to electromagnetic radiation:
- Uses look-up tables for efficiency
- Separates calculation into 16 spectral bands
RRTM - Previous Work

- RRTM proposed as benchmark kernel
  - Contains only CPU code

- Initial GPU port to CUDA Fortran on PGI’s website
  - [http://www.pgroup.com/resources/accel_files/list.htm](http://www.pgroup.com/resources/accel_files/list.htm)
  - One of several WRF components on PGI Accelerator Files site
  - Contains CUDA Fortran source code and white paper
  - Based on C1060 and early version (10.1) of the CUDA Fortran compiler
CUDA and CUDA Fortran

CUDA programming model
- Heterogenous programming model
  - Use both CPU and GPU, which have different memory spaces
  - Allows for incremental development
- Scalable programming model
  - Programs runs on any number of processors without recompiling
  - Write a program for one thread, instantiate on many parallel threads

CUDA Fortran is the Fortran analog of CUDA C
- Implemented in PGI’s Fortran compiler
- Program host and device code similar to CUDA C
- Host code is based on Runtime API
- Fortran language extensions to simplify data management
RRTM Components

RRTM has the following steps

- INIRAD: computes the ozone mixing ratio distributions
- MM5ATM: provides atmospheric profiles
- SETCOEF: calculates various quantities needed for the radiative transfer algorithm
- GASABS: calculates gaseous optical depths and Planck functions
  - Computed in 16 spectral bands
- RTRN: calculate the radiative transfer for both clear and cloudy columns

Radiation only depends on data in the same vertical column

- GPU exploits this parallelism
Data Layout

CPU Layout

GPU Layout

A(i,k,j)

Outer loops over (i,j)

Extract 1D arrays in k and send to RRTM

Reorder to A(i,j,k) after transfer

For rtrn(): launch nx*ny threads, each thread calculates one column

All other routines: launch nx*ny*nz threads
Recent Changes

- **Fermi architecture**
  - `-Mcuda=cc20`
  - L1 cache
    - 64KB allotment set to 48KB L1 cache and 16KB shared memory
  - Lookup tables changed from `constant` to `device` arrays
    - `constant` still used for physical constants (scalar values)

- **CUDA Fortran additions**
  - Pinned host memory used for large arrays (faster transfer, enables asynchronous transfers)
    - add `pinned` attribute to variable declarations
Results

Performed on system with
- Two quad-core Xeon X5550 CPUs (2.67 GHz)
- Tesla M2050 (Fermi) GPU (448 cores, 1.15 GHz, 3GB memory)
- Tesla M2090 (Fermi) GPU (512 cores, 1.3 GHz, 6GB memory)
- PGI 11.8 compilers
  - Utilizes a single CPU core

Input data is on a \((nx,nz,ny)\) mesh with \(nx=73, nz=28, ny=60\)
## Results

<table>
<thead>
<tr>
<th></th>
<th>Previous Study</th>
<th>Current Results</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X5440</td>
<td>Tesla C1060</td>
</tr>
<tr>
<td></td>
<td>-fast</td>
<td>Baseline</td>
</tr>
<tr>
<td>Overall time (ms)</td>
<td>703</td>
<td>83</td>
</tr>
<tr>
<td>% time in gasabs()</td>
<td>56%</td>
<td>31%</td>
</tr>
<tr>
<td>% time in rtrn()</td>
<td>28%</td>
<td>46%</td>
</tr>
</tbody>
</table>

Tesla M2050/M2090 Baseline: code from previous study, recompiled with 11.8
Tesla M2050/M2090 Modified: items in Recent Changes implemented
Conclusions and Future Work

Conclusions
- Good speedup for a small problem
- Data reordering is key
  - Leverage separate memory space (host data unchanged)
  - Reordering fast on device

Future Work
- Add asynchronous data transfers to hide communication
- Textures are coming to CUDA Fortran
  - Possibility for lookup tables
- Utilize CPU and GPU
Batched Solver for Small Matrices

- Motivated by chemical simulation containing ~20 species
  - Independent system is solved at each grid point
- LU decomposition, Gaussian and Gauss-Jordan elimination
  - Dense matrices
  - Partial pivoting
  - Routine chosen based on matrix size and hardware
- Double precision (and double-complex)
- Code will be available at http://nvdeveloper.nvidia.com
Batched Solver Implementation

- Each system is mapped to a thread block
- Each thread in a thread block can work on multiple matrix elements
  - One matrix element per thread most efficient for small matrices
- Pivoting via two-stage process
  - Configurable number of threads finds maximum of elements assigned to them, and write these intermediate values to shared memory
  - All threads redundantly find the maximum of the intermediate values
  - Can be turned off for diagonally dominant matrices
Batched Solver Performance

Double Precision Solve Performance (Batch of 100,000 Matrices)

- Tesla M2090
- Tesla M2050
- CPU (2 x X5550, 8-way Custom)
- CPU (2 x X5550, 8-way Batched MKL)

GFLOPS vs. N (for NxN matrix)
GPU Acceleration of the Longwave Rapid Radiative Transfer Model in WRF using CUDA Fortran

G. Ruetsch, M. Fatica, E. Phillips, N. Juffa