Porting MURaM (Max Planck University of Chicago Radiative MHD) to GPUs Using OpenACC

Rich Loft (Director TDD in CISL, NCAR)
Eric Wright (PhD student) & Sunita Chandrasekaran (Assistant Professor)
University of Delaware
loft@ucar.edu, {efwright, schandra}@udel.edu

Project in collaboration with NCAR, Max Planck for Solar System Research and University of Delaware

March 19, 2019 GTC 2019
Anatomy of the Sun

- Core: 16,000,000 K
- Prominence: 10,000 K
- Corona: 1,000,000 K
- Chromosphere: 10,000 K
- Photosphere: 6,000 K
- Flare: 20,000,000 K

K: Temperatures in Kelvin
MURaM (Max Planck University of Chicago Radiative MHD)

- The primary solar model used for simulations of the upper convection zone, photosphere and corona.
- Jointly developed and used by HAO, the Max Planck Institute for Solar System Research (MPS) and the Lockheed Martin Solar and Astrophysics Laboratory (LMSAL).
- MURaM has contributed substantially to our understanding of solar phenomena.
- MURaM also plays a key role in interpreting high resolution solar observations.

The Daniel K. Inouye Solar Telescope (DKIST), a ~$300M NSF investment, is expected to advance the resolution of ground based observational solar physics by an order of magnitude.
High Resolution Simulations of the Solar Photosphere

- Forward modeling of DKIST observables will require simulations with grid spacing of 4 km on a regular basis.
- Requires at least 10-100x increase in computing power compared to current baseline.
From data inspired to data driven simulations of solar eruptions

- Realistic simulations of the coupled solar atmosphere are an important tool to understand and even predict solar eruptions.
- Current models run about ~100x slower than real-time
- Data driven simulations of solar events would allow for analysis and prediction of ongoing solar events
- Future data assimilation applications will require ensemble runs (~10x)

Comprehensive model of entire life cycle of a solar flare (Cheung et al 2018)
Realistic Simulations of the Chromosphere

- The chromosphere is a difficult region to observe and model
- New instruments (DKIST 4m telescope and sunrise balloon observatory) will allow for unprecedented observations
- However, the modeling still must be brought up-to-date to these observational advancements.
Toward Predicting Solar Eruptions

• How do we get MURaM 100x faster? GPU-accelerated RT is the key.
  – RT solver speed up on GPUs (~85x) a CPU core via wavefront algorithms (Chandrasekaran et al.)
  – RT iteration counts are reduced (~2x) on GPU’s bigger subdomains.
  – But, we can rewrite RT solver to do required wavelengths (these are embarrassingly parallel).
  – Last two points play to GPU’s strength: data parallelism.
  – Estimate 450 GPUs could achieve breakeven (1 simulated hour/hour)

• Thousands of GPUs would be required to do full data assimilation.
  – Expensive but not unthinkable (Summit has 27,600!)

• Requirements play to strengths of GPUs, and trend in their design.
Planned MURaM Development

• Porting of the MURaM code to GPUs using OpenACC (collaboration between HAO, CISL, University of Delaware and MPS)
• Implementation of a more sophisticated chromosphere (HAO, MPS)
• Implementation of boundary conditions that allow data driven simulations of solar events (HAO, CU Boulder, LMSAL)
• New IO modules that allow for data compression during the IO process and runtime visualization.
Time to dive deeper into the computational science side of things! 😊
Roadmap

• Profiling, parallelization, re-profiling
• Optimizing CPU-GPU data transfer/management
• Focusing on the most important loop – Radiation Transport (RT)
  – Long term science goals
• Re-designing Radiation Transport Algorithm
• More profiling info to find & address performance challenges
Profiling Tools

- Several profiling techniques were used to obtain an initial, high-level view of the code
- Function call map
- Arm MAP for generalized performance metrics and MPI
- NVProf for GPU performance profiling
Experimental Setup

• **NVIDIA PSG Cluster (hardware)**
  - CPU: *Intel* Xeon E5-2698 v3 (16-core) and Xeon E5-2690 v2 (10-core)
  - GPU: *NVIDIA* Tesla V100 (4 GPUs per node)

• **Software Used**
  - PGI 18.4 (CUDA 9.1 and 9.2) and PGI 18.10
    • Results in this talk use PGI 18.4
  - icc 17.0.1/18.0.1

Thanks to NVIDIA for giving us access to their PSG system for our experiments!!!
# Routine Descriptions

<table>
<thead>
<tr>
<th>Name</th>
<th>Routine Summary:</th>
<th>Broadwell (v4) core (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TVD Diffusion</td>
<td>Update diffusion scheme - using TVD slope + flux limiting.</td>
<td>7.36812</td>
</tr>
<tr>
<td>Magnetohydrodynamics</td>
<td>Calculate right hand side of MHD equations.</td>
<td>6.26662</td>
</tr>
<tr>
<td>Radiation Transport</td>
<td>Calculate radiation field and determine heating term (Qtot) required in MHD.</td>
<td>5.55416</td>
</tr>
<tr>
<td>Equation of State</td>
<td>Calculate primitive variables from conservative variables. Interpolate the equation of state tables.</td>
<td>2.26398</td>
</tr>
<tr>
<td>Time Integration</td>
<td>Performs one time integration.</td>
<td>1.47858</td>
</tr>
<tr>
<td>DivB Cleaner</td>
<td>Clean any errors due to non-zero div(B).</td>
<td>0.279718</td>
</tr>
<tr>
<td>Boundary Conditions</td>
<td>Update vertical boundary conditions.</td>
<td>0.0855162</td>
</tr>
<tr>
<td>Grid Exchange</td>
<td>Grid exchanges (only those in Solver)</td>
<td>0.0667914</td>
</tr>
<tr>
<td>Alfven Speed Limiter</td>
<td>Limit Maximum Alfven Velocity</td>
<td>0.0394724</td>
</tr>
<tr>
<td>Synchronize timestep</td>
<td>Pick minimum of the radiation, MHD and diffusive timesteps.</td>
<td>4.48E-05</td>
</tr>
</tbody>
</table>
Flowchart describes one timestep of the MHD equations: Function Description

- **Initialization**
  - Save Cons (int_time)

- **Runge Kutta Time Update all stages**
  - Cons to Prim (eos_time)
  - MHD Residual (mhd_time)
  - Sync Timestep (sync_time)
  - Source Integrate (int_time)
  - Grid Exchange (dst_time)
  - Boundary Conditions (bnd_time)

- **Stage 1 Special**
  - Adjust Va Max (vlim_time)
  - Optically Thin Losses (int_time)
  - Radiation Transport (rt_time)
  - Diagnostic, EOS, Slice and DEM Outputs (io_time)

- **Integration**
  - TVD diffusion (tvd_time)
  - DivB Clean (divB_time)
  - TCheck Limits (int_time)
  - Grid Exchange (dst_time)
  - Boundary Conditions (bnd_time)
  - Timestep Update
  - Output + Analysis (io_time)

If time < Tmax
Profile driven parallelization

• Based on information gathered from profiling, implement simple development cycle:
  – Identify – which loops are currently the most impactful
  – Parallelize – the loop(s) for GPU execution
  – Verify – that our test cases pass with the new change
  – Reprofile/Optimize - until happy enough to move
GPU Profile using nvprof
CUDA Occupancy Report

240x160x160 Dataset

<table>
<thead>
<tr>
<th>Kernel Name</th>
<th>Theoretical Occupancy</th>
<th>Achieved Occupancy</th>
<th>Runtime % (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHD</td>
<td>25%</td>
<td>24.9%</td>
<td>32.4%</td>
</tr>
<tr>
<td>TVD</td>
<td>31%</td>
<td>31.2%</td>
<td>31.6%</td>
</tr>
<tr>
<td>CONS</td>
<td>25%</td>
<td>24.9%</td>
<td>6.3%</td>
</tr>
<tr>
<td>Source_Tcheck</td>
<td>25%</td>
<td>24.9%</td>
<td>5.2%</td>
</tr>
</tbody>
</table>

Radiation Transport

<table>
<thead>
<tr>
<th>Kernel Name</th>
<th>Theoretical Occupancy</th>
<th>Achieved Occupancy</th>
<th>Runtime % (GPU)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver</td>
<td>100%</td>
<td>10.2%</td>
<td>15.2%</td>
</tr>
<tr>
<td>Interpol</td>
<td>56%</td>
<td>59.9%</td>
<td>4.9%</td>
</tr>
<tr>
<td>Flux</td>
<td>100%</td>
<td>79%</td>
<td>1.5%</td>
</tr>
</tbody>
</table>
What did we learn so far?

- What is MURaM?
- What is the state of the project?
- What are the challenges identified?
- What problems do we still have to overcome?
  - Optimizing RTS
  - Learning from CUDA Occupancy Report
Toward Predicting Solar Eruptions

• **How do we get MURaM 100x faster? GPU-accelerated RT is the key.**
  – RT solver speed up on GPUs (~85x) a CPU core via wavefront algorithms (Chandrasekaran et al.)
  – RT iteration counts are reduced (~2x) on GPU’s bigger subdomains.
  – But, we can rewrite RT solver to do required wavelengths (these are embarrassingly parallel).
  – Last two points play to GPU’s strength: data parallelism.
  – **Estimate 450 GPUs could achieve breakeven (1 simulated hour/hour)**

• **Thousands of GPUs would be required to do full data assimilation.**
  – Expensive but not unthinkable (Summit has 27,600!)

• **Requirements play to strengths of GPUs, and trend in their design.**
Data Dependency Along Rays:

- Data dependency is along a plane for each octant, angle combo.
- Depends on resolution ratio, not known until run-time.
- Number of rays per plane can vary.

Solving RTS Data Dependency

• We can deconstruct the 3D grid into a series of 2D slices
• The direction of the slices is dependent on the X,Y,Z direction of the ray
• Parallelize within the slice, but run the slices themselves serially in predetermined order
More profile driven optimizations
More profile driven optimizations
More profile driven optimizations
More profile driven optimizations
More profile driven optimizations
Get inputs from MHD: (density, temperature and pressure)
Interpolate onto (offset) RTS Grid.
Calculate Radiative properties ($\kappa$, S).
Determine number of rays.
Loop over octant: (up/down, left/right, fwd/back) And angle: $(0, N_{\mu})$

Ray By Ray Process:
Interpolate onto Ray: xyz -> along rays.
Calculate Radiation coefficients.
Load lower BC's
Integrate along ray.
Write Upper BC's
Exchange BC's
Calculate error in I
Converged?
Add contribution to J

All Rays Finished?
Yes
No

From J, Calculate Q.
Send Back to MHD

Wrapper Process:
MHD Variable Calculation:

RT function
Get inputs from MHD: (x,y,z)

Interpolate onto (offset) RTS Grid.

Calculate Radiative properties (κ, S).

Split problem in 3: xy, yz, xz planes.

Determine number of rays per plane.

Interpolate onto Ray: xyz -> along rays.

Calculate Radiation coefficients.

Load lower BC's

Integrate along ray.

Write Upper BC's

Exchange BC's

Add contribution to J

Calculate error in J

Is J Converged

From J, Calculate Q

Send Back to MHD

Alternative approach:
Less communication => more parallelism
More computation

RT function
What did we learn so far?

• What is MURaM?
• What is the state of the project?
• What are the challenges identified?
• What problems do we still have to overcome?
  – Optimizing RTS
• Learning from CUDA Occupancy Report
NVIDIA V100 specification

- 64 warps per multiprocessor (32 threads per warp)
- 65536 registers per multiprocessor
- 96 KB shared memory per multiprocessor
- Occupancy = # Used Warps / # Possible Warps
- How many warps we can use is dependent on how many registers and how much shared memory we use per thread
CUDA Occupancy Report

• Using the CUDA occupancy calculator
• We can see why our theoretical occupancy for MHD, TVD, etc is so low
• From this graph, we know that threads-per-block is not the problem
CUDA Occupancy Report

- We can also see that our shared memory usage is much lower than it could be.
- One idea we have is to start moving some of our data to shared memory to get better usage.
CUDA Occupancy Report

• Finally, we see that our problem is register usage per thread
• To get 100% occupancy, we would need to reduce register usage to 32 registers per thread
MURaM: Parallel processing using MPI

Introduction

- Three dimensional uniform cartesian grid.
- Grid is divided into smaller pieces based on MPI ranks and are processed independently.
- Halo information is communicated between the MPI ranks at a regular intervals.
- Large percentage of halo exchange happens in Radiative transfer.

Optimization strategies

- Overlap computation and communication in RTS.
- Perform a GPU to GPU direct communication without involving host.
- Optimize the packing and unpacking of communication data.
MURaM: MPI profiling using Intel Trace Analyzer

Communication in RTS for 24 rays
Results

Currently:
- 40% runtime is GPU
- 40% runtime is still CPU
- 20% is data movement
- As we finish porting the rest of the code to GPU, and optimizing the parts discussed today, we expect these results to improve significantly

<table>
<thead>
<tr>
<th>Speedup of NVIDIA Volta V100 over -&gt;</th>
<th>Singlecore</th>
<th>Full MPI node (32 cores)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RTS</td>
<td>27x</td>
<td>1.8x</td>
</tr>
<tr>
<td>TVD</td>
<td>36x</td>
<td>2x</td>
</tr>
<tr>
<td>MHD</td>
<td>18x</td>
<td>0.78x</td>
</tr>
<tr>
<td>Overall</td>
<td>15x</td>
<td>0.9x</td>
</tr>
</tbody>
</table>
Summary

• Accelerating the most significant routine – Radiation Transport, among other routines
  – Exploring how to optimize further
• Profiling and re-profiling – a Must
• Using directives enable maintenance of a single source code for both multicore and accelerators
  – Enables *new science*

Rich Loft, Eric Wright, Sunita Chandrasekaran
loft@ucar.edu, {efwright, schandra}@udel.edu