Porting the ICON Non-hydrostatic Dynamics and Physics to GPUs

William Sawyer (CSCS/ETH), Christian Conti (ETH), Xavier Lapillonne (C2SM/ETH)

Programming weather, climate, and earth-system models on heterogeneous multi-core platforms

Sep. 7-8, 2011, NOAA, Boulder USA
The ICON Model

- ICOsahedral Non-hydrostatic model
- Multi-resolution grid (not supported here)
- Triangular cells
- Conservation laws
- ‘Bandwidth limited’
- Extensive use of indexing arrays
- Developers: MPI-M, DWD
ICON-GPU Project

- CSCS/C2SM offered its assistance with GPUs
- Goal: compare GPU paradigms in terms of efficiency, usability and developer friendliness
- Non-hydrostatic solver (~5K l.o.c.), and physical parameterizations
- Paradigms chosen: OpenCL, CUDAFortran for dynamics, accelerator directives (PGI/Cray) for physics
- OpenCL NH solver: 6 weeks, by PhD student (Conti)
- CUDAFortran NH solver: ~8 weeks (Sawyer)
- Lapillonne: microphysics, radiation, turbulence with directives
Roofline of Various GPUs

PCIe Theoretical Bandwidth - 8 GB/s (GPU as Accelerator)
Porting NH solver to GPUs

Fortran

OpenCL

CUDA Fortran
# OpenCL/CUDAFortran Approaches

<table>
<thead>
<tr>
<th>OpenCL</th>
<th>PGI CUDAFortran</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Minimal refactoring</td>
<td>• Refactored to remove intermediate arrays</td>
</tr>
<tr>
<td>• Extensive use of local (shared) memory</td>
<td>• More use of registers</td>
</tr>
<tr>
<td>• Iteration space: 1D or 2D</td>
<td>• 1-D grid of thread blocks, each with 2D distribution</td>
</tr>
<tr>
<td>• Blocking factor: nproma=1 optimal</td>
<td>• nproma=8/16 optimal</td>
</tr>
<tr>
<td>• Simpler but more kernels, fewer IFs in kernels</td>
<td>• Fewer kernels, more IFs</td>
</tr>
</tbody>
</table>
Implicit Vertical Solver

- Implicit solver requires a tridiagonal solution for each vertical column
- All 2-D arrays except one ($z_q$) can be replaced with registers; CUDAFortran version makes use of this
OpenCL Kernels

Performance (GFLOP/s)

Operational Intensity (FLOP/B)

Kernels with loops

M2050 > Cayman

R2B3 to R2B4

Kernels acting on small arrays
## CUDAFortran Time Distribution

<table>
<thead>
<tr>
<th>call</th>
<th>t_min</th>
<th>t_average</th>
<th>t_max</th>
<th>t_total</th>
</tr>
</thead>
<tbody>
<tr>
<td>total</td>
<td>57.547s</td>
<td>57.547s</td>
<td>57.547s</td>
<td>57.547s</td>
</tr>
<tr>
<td>solve_nh</td>
<td>0.05679s</td>
<td>0.06114s</td>
<td>0.06780s</td>
<td>56.780s</td>
</tr>
<tr>
<td>nh_driver</td>
<td>5.755s</td>
<td>5.755s</td>
<td>5.755s</td>
<td>57.547s</td>
</tr>
<tr>
<td>intp</td>
<td>0.01501s</td>
<td>0.01501s</td>
<td>0.01501s</td>
<td>0.01501s</td>
</tr>
<tr>
<td>vel tendencies</td>
<td>0.00007s</td>
<td>0.00007s</td>
<td>0.00007s</td>
<td>0.00007s</td>
</tr>
<tr>
<td>exner value</td>
<td>0.00015s</td>
<td>0.00015s</td>
<td>0.00015s</td>
<td>0.00015s</td>
</tr>
<tr>
<td>rho and ddz_exner</td>
<td>0.00101s</td>
<td>0.00101s</td>
<td>0.00101s</td>
<td>0.00101s</td>
</tr>
<tr>
<td>horizontal calcs</td>
<td>0.00187s</td>
<td>0.00187s</td>
<td>0.00187s</td>
<td>0.00187s</td>
</tr>
<tr>
<td>rbf vt calc</td>
<td>0.00090s</td>
<td>0.00090s</td>
<td>0.00090s</td>
<td>0.00090s</td>
</tr>
<tr>
<td>post calcs</td>
<td>0.00345s</td>
<td>0.00345s</td>
<td>0.00345s</td>
<td>0.00345s</td>
</tr>
<tr>
<td>device copies</td>
<td>0.17517s</td>
<td>0.17517s</td>
<td>0.17517s</td>
<td>0.17517s</td>
</tr>
</tbody>
</table>

• More optimizations possible!

• “vel tendencies” consists of 13 kernels, “vertical calcs” 5 kernels, “vn vt covariant” also 5, but still they seem to contain bottlenecks

• Device copies and tridiagonal solver appear not to be a problem
Aggregated NH Performance (DP)

- Fermi M2050 (CUDAFortran):
  - R2B3: 18.8 GFLOP/s
  - R2B4: 33.0 GFLOP/s
- Cayman (OpenCL):
  - R2B4: 21.2 GFLOP/s
Physics Parameterizations

- To be shared between ICON and COSMO (European regional model)
- Currently ported to GPUs:
  - PGI: microphysics (hydci_pp), radiation (fesft), turbulence (only turbdiff yet)
  - OMP – acc (Cray): microphysics, radiation
- GPU optimization: loop reordering, replacement of arrays with scalars
- Note: hydci_pp, fesft and turbdiff subroutines represent respectively 6.7%, 8% and 7.3% of the total execution time of a typical cosmo-2 run.
Physics Performance

- Peak performance of Fermi card for double precision is 515 GFlop/s, i.e., 5%, 4.5% and 2.5% of peak performance for the microphysics, radiation and turbulence schemes, respectively.

- Parallel CPU code run on 12 cores AMD magny-cours CPU – however there are no mpi- communications in these standalone test codes.

- Note the peak performance of Fermi card is 5 times that of the magny cours processor. Overhead of data transfer for microphysics and turbulence is very large.
Lessons learned

- Never underestimate the potential of a smart, motivated graduate student!
- CUDA/OpenCL programming not that difficult, but highly error-prone; debugging options limited
- CUDA Fortran is much more ‘appealing’ to developers, but OpenCL is a portable paradigm
- Optimizations to both versions still possible
- Future: use domain-specific language to describe solver; library to implement kernel operations
- Physics: we must learn how to combine directive-based Fortran codes with CUDA/C++ codes (e.g., COSMO dycore)
Thank you for your attention!
wsawyer@cscs.ch
Physical Parameterizations

- 2D data fields inside the physics packages with one horizontal and one vertical dimensions: $f(nproma,ke)$, with $nproma = \text{ie} \times \text{je} / \text{nblock}$.

- 2D data fields inside the physics packages with one horizontal and one vertical dimensions: $f(nproma,ke)$, with $nproma = \text{ie} \times \text{je} / \text{nblock}$.

- Goals:
  - Parameterizations to be shared with COSMO (regional) model
  - Blocking strategy: all physics parametrization could be computed while data remains in the cache

```fortran
  call init_radiation
  call init_turbulence
  ... 
  do ib=1,nblock 
    call copy_to_block 
    call organize_radiation 
    ... 
    call organize_turbulence 
    call copy_back 
  end do 
```
THE ROOFLINE MODEL


OPERATIONAL INTENSITY $R = \frac{\text{FLOPS}}{\text{MEMORY TRAFFIC (BYTES)}}$

PERFORMANCE MODEL FOR BOTH GPU AND CPU

"HOW GOOD IS MY CODE?"

$\text{f} = r \cdot b_{\text{max}}$

GREEN COMPUTING:
- computationally bound: reduce bus clock/s
- memory bound: reduce processor clock/s

4x Quad-Core AMD Opteron 8380 @ 2.5GHz - 1 Thread - C++

STREAM BENCHMARK

4.7 GB/S

MAXIMUM ACHIEVABLE PERFORMANCE!

C++ MICROBENCHMARK

3 GFLOP/S
GPU Bandwidths

![Graph showing GPU bandwidths with different models and size values.](image-url)