Code restructuring to improve performance in WRF model physics on Intel Xeon Phi

John Michalakes

NOAA Affiliate
Environmental Modeling Center

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

September 20, 2013
WRF and Accelerators

  - WSM5 Microphysics
  - Advection
  - Reactive chemistry
  - Radiation

- Whole code porting and performance studies
WSM5 Microphysics

- Predict five hydrometrics with thermodynamic feedback to model
- Largest physics component of model cost (next is radiation)
  - ~25% of Jan. 2000 30km workload
  - ~9% of CONUS 12km workload
- Logically column wise but called for a west-east strip of columns at a time
- Standalone kernel adapted to GPUs and now Xeon Phi
  - [www.mmm.ucar.edu/wrf/WG2/GPU](http://www.mmm.ucar.edu/wrf/WG2/GPU)
  - Michalakes and Vachharajani, 2009
  - Mielikainen et al. 2012

Fig. 1. Flowchart of the microphysics processes in the WSM5 scheme.

*$OMP PARALLEL DO
DO <over tiles>
CALL WSM5
SUBROUTINE WSM5
DO J = jts, jte
  CALL WSM52D
SUBROUTINE WSM52D
  DO k = kts,kte
    DO i = its,ite
      ...

WSM5 Microphysics

Optimization for Xeon Phi

- Performance analysis showed
  - Insufficient thread parallelism
  - Inadequate vectorization from misalignment and loop peeling
  - Memory latency bound and not saturating memory bandwidth

- Optimizations
  1. Collapse i and j loops over 16-cell chunks. More thread parallelism; smaller footprint per thread.

Note: could have used OMP COLLAPSE except the nested i-loops are in a called subroutine and there is often a k-loop in between

Original

```fortran
!$OMP PARALLEL DO
DO <over tiles>
  CALL WSM5
SUBROUTINE WSM5
  DO J = jts, jte
    CALL WSM52D( t, q, … )
  END DO
SUBROUTINE WSM52D( t, q, … )
  DO k = kts,kte
    DO i = its,ite
      DO k = kts,kte
        DO i = its,ite
          …
        END DO
      END DO
    END DO
  END DO
END SUBROUTINE WSM52D
```

Modified

```fortran
CALL WSM5
SUBROUTINE WSM5
  !$OMP PARALLEL DO
  !$OMP COLLAPSE
  DO ichunk = 1,((1+(ite-its+1)/CHUNK)*CHUNK)*(jte-jts+1),CHUNK
    CALL WSM52D ! for one 16 column-wide chunk
  END DO
SUBROUTINE WSM52D
  #define its 1
  #define ite 16
  DO k = kts,kte
    DO i = its,ite
      DO k = kts,kte
        DO i = its,ite
          …
        END DO
      END DO
    END DO
  END DO
END SUBROUTINE WSM52D
```
WSM5 Microphysics

Optimization for Xeon Phi

- Performance analysis showed
  - Insufficient thread parallelism
  - Inadequate vectorization from misalignment and loop peeling
  - Memory latency bound and not saturating memory bandwidth

- Optimizations
  1. Collapse i and j loops over 16-cell chunks. More thread parallelism; smaller footprint per thread.
  2. Compute using thread-private statically sized arrays. Improve vectorization.
WSM5 Microphysics

Optimization for Xeon Phi

- Performance analysis showed
  - Insufficient thread parallelism
  - Inadequate vectorization from misalignment and loop peeling
  - Memory latency bound and not saturating memory bandwidth

- Optimizations
  1. Collapse i and j loops over 16-cell chunks. More thread parallelism; smaller footprint per thread.
  2. Compute using thread-private statically sized arrays. Improve vectorization.
  3. Fuse loops, combine/eliminate temporaries to reduce footprint from 100KB → 60KB thread. More threads/core, hide memory latency.
WSM5 Microphysics

Optimization for Xeon Phi

- Performance analysis showed
  - Insufficient thread parallelism
  - Inadequate vectorization from misalignment and loop peeling
  - Memory latency bound and not saturating memory bandwidth

- Optimizations
  1. Collapse i and j loops over 16-cell chunks. More thread parallelism; smaller footprint per thread.
  2. Compute using thread-private statically sized arrays. Improve vectorization.
  3. Fuse loops, combine/eliminate temporaries to reduce footprint from 100KB → 60KB thread. More threads/core, hide memory latency.

Effort optimizing for Phi benefits host

Allows all four threads per core to fit in L2
Additional restructuring for vectorization

- Inspect compiler vectorization report (-vec-report3)
  - Find loops that don’t vectorize and fix if possible
  - Example: inside loop that won’t vectorize because of conditionals

```cpp
if(qrs(i,k,2).gt.qcmin.and.qci(i,k,1).gt.qmin) then
  psacw(i,1) = min(pacrc*n0sfac(i,k)*rslope3_v(i,2)
                  *rslopes_v(i,2)*qci(i,k,1)*denfac(i,k)
                  ,qci(i,k,1)*rdtcld)
eendif
pidep(i,1) = 0.
pdsdep(i,1) = 0.
if(supcol .gt. 0) then
Deposition/Sublimation rate of ice [HDC 9]
(T<T0: V->I or I->V)
```
## Vtune Analysis on KNC

<table>
<thead>
<tr>
<th></th>
<th>ORIG</th>
<th>Some Optim.</th>
<th>Full Optim.</th>
<th>Desired **</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avg. CPI per Thread</td>
<td>5.71</td>
<td>4.21</td>
<td>3.6</td>
<td>&lt; 4</td>
</tr>
<tr>
<td>%VECTOR INSTR.</td>
<td>36.84</td>
<td>35.87</td>
<td>49.73</td>
<td></td>
</tr>
<tr>
<td>Vectorization Intensity</td>
<td>5.9</td>
<td>5.44</td>
<td>10.34</td>
<td>&gt; 16</td>
</tr>
<tr>
<td>%SW PF INSTR.</td>
<td>9.38</td>
<td>12.39</td>
<td>13.1</td>
<td></td>
</tr>
<tr>
<td>L1 Compute to Data Access Ratio</td>
<td>6.98</td>
<td>6.96</td>
<td>17.34</td>
<td>&gt; Vec. Intens.</td>
</tr>
<tr>
<td>L2 Compute to Data Access Ratio</td>
<td>74.23</td>
<td>71.53</td>
<td>414.81</td>
<td>&gt; 100x L1 Ratio</td>
</tr>
<tr>
<td>L1 MISSES</td>
<td>208</td>
<td>179</td>
<td>73</td>
<td>(tending downward)</td>
</tr>
<tr>
<td>L1 Hit Rate</td>
<td>88</td>
<td>88.69</td>
<td>91.75</td>
<td>&gt; 95</td>
</tr>
<tr>
<td>Estimated Latency Impact</td>
<td>178.93</td>
<td>140.44</td>
<td>242.97</td>
<td>&lt; 145</td>
</tr>
<tr>
<td>L2 CACHE READ MISS RATIO</td>
<td>20.32</td>
<td>14.71</td>
<td>9.15</td>
<td>(tending downward)</td>
</tr>
<tr>
<td>L1 TLB MISS RATIO</td>
<td>2.02</td>
<td>1.14</td>
<td>1.36</td>
<td>&lt; 1</td>
</tr>
<tr>
<td>PEAK RD BW (MB/s)</td>
<td>70102.35</td>
<td>69451.53</td>
<td>33542.19</td>
<td>?</td>
</tr>
<tr>
<td>PEAK WR BW (MB/s)</td>
<td>40923.18</td>
<td>31611.32</td>
<td>19080.70</td>
<td>?</td>
</tr>
<tr>
<td>PEAK TOTAL BW (MB/s)</td>
<td>111025.50</td>
<td>101062.90</td>
<td>54622.89</td>
<td>?</td>
</tr>
<tr>
<td>PERFORMANCE (GF/s/ % peak)***</td>
<td>46/2.0%</td>
<td>94/4.1%</td>
<td>176/7.6%</td>
<td></td>
</tr>
</tbody>
</table>

** Optimization and Performance Tuning for Intel® Xeon Phi Coprocessors, Part 2: Understanding and Using Hardware Events**  
Shannon Cepeda, Nov. 2012  

*** Assumes 2.3TF/s Peak SP
# Additional Optimizations

- I. Gokhale, L. Meadows, R. Sasanka, Intel Corp.

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline aka Original Optimized version (as of 6/25/13)</td>
<td>37.1 msec</td>
</tr>
<tr>
<td>Loop changes from WHERE to “do if else” with VECTOR ALIGNED pragma (~5 loops)</td>
<td>33.5 msec</td>
</tr>
<tr>
<td>OMP Scheduling to “guided”</td>
<td>31.0 msec</td>
</tr>
<tr>
<td>Copy intrinsic (pre-fetch)</td>
<td>29.7 msec</td>
</tr>
</tbody>
</table>
Idiosyncratic code generation

- Looking at generated code, Indraneil and Ruchira found scatter-gathers being generated in place of masked vector operations.
- Fixed by changing WHERE statements back to explicit DO loops with conditions (go figure).
- Reported to compiler team....
WSM5 Microphysics

WSM5 CONUS 12KM Workload, 6.53 GF/call (Intel SDE)
(www.mmm.ucar.edu/wrf/WG2/bench)

* Improved GPU/CUDA Based Parallel Weather and Research Forecast (WRF) Single Moment 5-Class (WSM5) Cloud Microphysics.
Considerations

• Some optimizations affect results
  
  Changed –fp-model precise to -fp-model fast=1
  Added –ftz -no-prec-div -no-prec-sqrt -fimf-precision=low

  – Verification using global RMS differences suggest within roundoff; however:
    • WSM5, like many physics schemes, contains sharp on-off transitions
    • Therefore, differences may show up more dramatically in point-wise comparisons
    • Meteorological validation may be needed

• Symmetric execution mode is appealing, but
  – Requires two executables
  – Raises another auto-tuning problem
Summary

- NWP codes have ample parallelism, both thread and fine-grained that must be exposed and exploited
- Set size works against floating point utilization by constraining vector length and number of threads in flight
- Restructuring techniques
  - Compiler and runtime supported
    - Automatic vectorization
    - Data alignment
    - Decomposition tuning
  - Programmer supported
    - Blocking, loop fusion [eventually functional fusion]
    - Conditionals to vector mask operations when possible
    - Reuse or eliminate temporary storage
Contributors

• Intel Corp.
  – Michael Greenfield
  – Lawrence Meadows
  – Alexander Knyazev
  – Indraneil Gokhale
  – Ruchira Sasanka

• NCAR
  – WRF Developers Committee

• U. Wisconsin/SSEC
  – Jarno Mielikainen
  – Bormin Huang

• Alexandria University, Egypt
  – Ahmed Saeed et al.

• U. Colorado
  – Jeremy Siek

• NREL
  – Jim Albin