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

John Michalakes
UCAR Visiting Scientist
Naval Research Laboratory, Marine Meteorology Division

Los Alamos National Laboratory
1 June 2016

ESMF webinar, 12/14/2016
Outline

• Hardware and software
  – Knights Landing overview
  – Roofline and next-generation HPC

• Optimizing for KNL
  – WSM5 Microphysics
  – RRTMG Radiative Transfer

• Summary
Hardware: Xeon Multi/Many-core Computing Platforms

- Intel Xeon Phi 7250 (Knights Landing) announced at ISC’16 in June
  - 14 nanometer feature size, > 8 billion transistors
  - 68 cores, 1.4 GHz modified “Silvermont” with out-of-order instruction execution
  - Two 512-bit wide Vector Processing Units per core
  - Peak ~3 TF/s double precision, ~6 TF/s single precision
  - 16 GB MCDRAM (on-chip) memory, > 400 GB/s bandwidth
  - “Hostless” – no separate host processor and no “offload” programming
  - Binary compatible ISA (instruction set architecture)
**Optimizing for Intel Xeon Phi**

- Most work in MIC programming involves *optimization* to achieve best share of peak performance
  - **Parallelism:**
    - KNL has up to 288 hardware threads (4 per core) and a total of more than 2000 floating point units on the chip
    - Exposing coarse, medium and especially fine-grain (vector) parallelism in application to efficiently use Xeon Phi
  - **Locality:**
    - Reorganizing and restructuring data structures and computation to improve data reuse in cache and reduce floating point units idle-time waiting for data: *memory latency*
    - Kernels that do not have high data reuse or that do not fit in cache require high *memory bandwidth*
- The combination of these factors affecting performance on the Xeon Phi (or any contemporary processor) can be characterized in terms of *computational intensity*, and conceptualized using The Roofline Model
Optimizing for Intel Xeon Phi

- **Roofline Model of Processor Performance**
  - Bounds application performance as a function of **computational intensity** – the number of floating point operations per byte moved between the processor and a level of the memory hierarchy.

**Empirical Roofline Toolkit**
https://crd.lbl.gov/departments/computer-science/PAR/research/roofline/

*Thanks: Doug Doerfler, LBNL*

*Sometimes referred to as:*
- **Arithmetic intensity** (registers→L1): *largely algorithmic*
- **Operational intensity** (LLC→DRAM): *improvable*
Optimization Toolbox

• Code and data restructuring
  – Static array/loop dimensions
  – Tiling + thread-local arrays (statically sized)
  – Modify loops to tile over more smaller pieces of domain
  – Fusing loops (beyond what compiler is able to do)
  – Thread task interleaving
  – Recompute rather than table lookups
  – Functional Fusion, Pipelining, Concurrency
Optimization Toolbox

- Code and data restructuring
  - Static array/loop dimensions
  - Tile to increase concurrency and locality
    - Recall: WRF tiles each MPI patch for threads and smaller working sets
Optimization Toolbox

• Code and data restructuring
  – Static array/loop dimensions
  – Tile to increase concurrency and locality
    • Recall: WRF tiles each MPI patch for threads and smaller working sets
    • But original design didn’t go far enough – threads compute shared data
    • Non-contiguous vectors, false-sharing, etc.
      - Hybrid MPI/OpenMP was rarely a win over straight MPI.
Optimization Toolbox

- **Code and data restructuring**
  - Static array/loop dimensions
  - Tiling + **thread-local arrays (statically sized)**
    - Continuous vectors within thread-local arrays
    - Helps shrink and align working set for cache and vector
    - Avoids false sharing between threads
    - Assumes enough work within routine to offset copying into and out of tile-sized arrays

- Fusing loops (beyond what compiler is able to do)
- Increase concurrency by tiling over more smaller pieces of domain
- Thread task interleaving
- Recompute rather than table lookups

Different algorithms (hardest)
Optimization Toolbox

- Code and data restructuring
  - Static array/loop dimensions
  - Tiling + thread-local arrays (statically sized)
  - Collapse and tile outer loops for more thread parallelism
    - May require “lowering” the outer loop from higher in call tree

!(OMP PARALLEL DO
  DO J = jts, jte
  subroutine( j, )
  DO i = its,ite
    ...
  subroutine( ichunk, )
  DO i = 1, CHUNK
    ...
)

1D parallel
2D parallel

(Note: there is also an OMP COLLAPSE directive that can be used if loop nesting is local to the subroutine)
Optimization Toolbox

• Code and data restructuring
  – Static array/loop dimensions
  – Tiling + thread-local arrays (statically sized)
  – Modify loops to tile over more smaller pieces of domain
  – Fusing loops (beyond what compiler is able to do)
    • Foster reuse of arrays touched repeatedly in outer (k) loop
    • Remove temporary arrays, reduce memory footprint
    • Improved cache and register utilization

SUBROUTINE ( ... )
  DO k = kts,kte
    DO i = its,ite
    DO k = kts,kte
    DO i = its,ite
...

SUBROUTINE ( ... )
  DO k = kts,kte
    DO i = its,ite
    DO i = its,ite
    DO I = its,ite
...
Optimization Toolbox

• Code and data restructuring
  – Static array/loop dimensions
  – Tiling + thread-local arrays (statically sized)
  – Modify loops to tile over more smaller pieces of domain
  – Fusing loops (beyond what compiler is able to do)
  – Thread task interleaving
  – Recompute rather than table lookups
Example 1: 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 columnwise but called for a west-east strip of columns at a time
- Standalone kernel adapted to GPUs and Xeon Phi
  - Michalakes and Vachharajani, 2009
  - Mielikainen et al. 2012
  - Gokhale and Meadows, 2016:

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 Xeon Phi
  - Michalakes and Vachharajani, 2009
  - Mielikainen et al. 2012

```c
!$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
      DO k = kts,kte
      DO i = its,ite
    ...
```
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

---

![Graph showing performance comparison between Xeon Sandybridge-EP (2x8 core) and Xeon Phi (60 C, 4 T/C)]

- Host processor: Intel Xeon Sandybridge 8 cores
- Phi

Higher is better in terms of performance.
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. Lower OpenMP loop; collapse i and j loops over 16-cell chunks.

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

Original
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. Lower OpenMP loop; collapse i and j loops over 16-cell chunks. More thread parallelism; smaller footprint per thread.
  2. Compute using thread-private statically sized arrays. Improved vectorization.
  3. 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. Lower OpenMP loop; 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. Lower OpenMP loop; 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.

Effort optimizing for Phi benefits host
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. Lower OpenMP loop; 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 \( \rightarrow \) 60KB thread. Better fit to L2 cache.
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. Lower OpenMP loop; 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. Better fit to L2 cache.

Effort optimizing for Phi benefits host
Loop nesting and Architecture Agnosticism

- CPU wants fine-grained (vector) parallelism over innermost loops

```plaintext
do k
  !dir$ vector
  do i
    enddo
endo
do k
  !dir$ vector
  do i
    enddo
endo
do k
  !dir$ vector
  do i
    enddo
endo
```

- GPU wants fine-grained (thread) parallelism over outermost loops

```plaintext
!$acc parallel loop
do i
  do k
    enddo
endo
do k
  enddo
endo
do k
  enddo
endo
!$acc end parallel loop
```

- Iteration over i-dimension may not be visible within subroutine
- There may be dependencies that prevent simple inversion of loops
- Scalar temporaries may need to be promoted to vectors
Rapid Radiative Transfer Model (RRTMG*)

- Accurate calculation of fluxes and cooling rates from incoming (shortwave) and outgoing (longwave) radiation
  - Significant computational cost
  - Load imbalance (day/night and cloud fraction)
  - Coded as 1-D vertical columns but this dimension does not vectorize
- Used in many weather and climate models
  - NCAR WRF
  - NCAR CAM5 and CESM1
  - NASA GEOS-5
  - NOAA NCEP GFS, CFS, RUC
  - ECMWF IFA and ERA40
  - ECHAM5


https://www.aer.com/science-research/atmosphere/radiative-transfer

One column of a weather or climate model domain
Restructuring RRTMG in NMM-B

- Concurrency and locality
  - Original RRTMG called in OpenMP threaded loop over South-North dimension

- Vectorization
  - Originally vertical pencils
Restructuring RRTMG in NMM-B

- **Concurrency and locality**
  - Original RRTMG called in OpenMP threaded loop over South-North dimension
  - Rewrite loop to iterate over tiles in two dimensions
  - Dynamic thread scheduling

- **Vectorization**
  - Originally vertical pencils
  - Extend inner dimension of lowest-level tiles to width of SIMD unit on KNC
  - Static definition of VECLEN
Performance results: RRTM Kernel on Xeon Phi and host Xeon (SNB)

1.2x on Xeon
3.5x on KNC

GF/s vs Thread Subscription

- 2xSNB dp (best)
- MIC dp (best)
- 2xSNB dp orig
- MIC orig
Effect of Optimizations on RRTMG Kernel

- Improvement
  - 2.8x Overall
    - 5.3x in SWRAD
    - 0.75x in LWRAD (degraded)

- Increasing vector length results in
  - 2.5x increase in working set size from 407KB to 1034KB per thread
  - 4x increase in L2 misses which Task Interleaving reduced by 30% in SWRAD

- Memory traffic
  - Increased from 59 to 124 GB/s, short of saturation
  - Key bottlenecks
    - Memory latency (not bandwidth)
    - Instruction latency on KNC

Higher-Level Transformations

- Functional Fusion
- Pipelineing
- Concurrency

Figure 3: (a) Order of iteration of grid point state vectors for physics routines. (b) Optimized execution order that requires breaking modularity of physics routines.

Figure 2: Overview of data dependences amongst routines in a simplified WRF.

“Taking the Weather Research and Forecasting (WRF) Model to Petascale”
Summary

• Restructuring for performance
  – Parallelism – provide as much work as possible to > 2000 fp units per processor
    • Increase available thread-level parallelism
      – Thread over multiple dimensions
      – Nested OpenMP regions (teams)
    • Expose available fine-grained parallelism
      – Reordering loop nesting, through subroutine boundaries if needed
  – Computational Intensity – perform as much work as possible on operands brought from memory
    • Tiling, local statically defined data structures
    • Array of Structures (AoS) versus Structure of Arrays (SoA)
    • Loop unrolling, fusion
    • Functional fusion and pipelining

• Tradeoff between software “ilities” and maximum performance