Performance Optimization Techniques for Accelerating WRF Physics Codes on Micro-architectures

Presented by:
T.A.J. Ouermi, Mike Kirby, Martin Berzins
Motivation

➢ Faster weather physics for operational Navy Environmental Prediction system Utilizing the NUMA core (NEPTUNE)

➢ Target architectures: Micro-architectures
   ➢ Intel Knights Landing (KNL),
   ➢ Intel Haswell

➢ Portability with OpenMP
Physics Optimization Challenges

➢ Large loops with many conditional not favorable for parallelism.

➢ Difficult to optimize with transition between many regimes.

WRF single-moment 6-class Microphysics Scheme (WSM6)
Vertical Physics Advantage

- Dependencies within columns.
- No dependencies between columns.
Methodology

- Identify bottlenecks
  - Wall Clock, Vtune.
  - Advisor, optrpt.
- Standalone experiments
- Apply findings to physics
Structures of Arrays (SOA)

Chunk size = multiple of vector length
$$\text{Transpose}$$

$$\begin{align*}
&\text{!$OMP DO} \\
&\text{do } j=1,km \\
&\text{do } i=1,im \\
&\quad a_{i,k} = b_{i,k} - c_{i,k} \\
&\text{end do} \\
&\text{end do}
\end{align*}$$

$$\begin{align*}
&\text{!$OMP DO} \\
&\text{do } i=1,im \\
&\text{do } j=1,km \\
&\quad a_{k,i} = b_{k,i} - c_{k,i} \\
&\text{end do} \\
&\text{end do}
\end{align*}$$

Thread id = j

Thread id = i
Vectorization

!$OMP DO
do j=1,km
!$OMP SIMD
do i=1,im
  a(i,k)=b(i,k)-c(i,k)
end do
end do

!$OMP DO
do i=1,im
!$OMP SIMD
do j=1,km
  a(k,i)=b(k,i)-c(k,i)
end do
end do

Thread id = j

Thread id = i
## Architectures

<table>
<thead>
<tr>
<th>Intel Knights Landing (KNL)</th>
<th>Intel Xeon CPU E-7-8890 (Haswell)</th>
</tr>
</thead>
<tbody>
<tr>
<td>➢ 1 socket</td>
<td>➢ 4 sockets</td>
</tr>
<tr>
<td>➢ 64 cores</td>
<td>➢ 18 cores per socket</td>
</tr>
<tr>
<td>➢ 4 threads per core</td>
<td>➢ 2 threads per core</td>
</tr>
<tr>
<td>➢ 2VPU per core (AVX-512)</td>
<td>➢ VPU (AVX-2)</td>
</tr>
<tr>
<td>➢ Clock of 1.5 Ghz</td>
<td>➢ Clock of 2.5 Ghz</td>
</tr>
<tr>
<td>➢ L1 32k</td>
<td>➢ L1 32k</td>
</tr>
<tr>
<td>➢ L2 1024k</td>
<td>➢ L2 256K</td>
</tr>
<tr>
<td>➢ MCDRAM 16GB</td>
<td>➢ L3 46MB</td>
</tr>
</tbody>
</table>
Experiments

➢ Transpose data.
➢ Thread-local SOA with different chunk sizes.
➢ Scheduling: dynamics vs static.
➢ Thread configurations.
➢ Simplify complex code by removing conditionals and nested code for vectorization.
Transpose vs SOA

➢ Identify suitable chunk size.

➢ Thread-local SOA 2x faster than transpose.
WSM6 Chunk Size

➢ Chunk = 32 for haswell.

➢ Chunk = 64 for KNL.
WSM6 Results

➢ Dynamic scheduling better in both cases.

➢ 70x on KNL and 26x on Haswell.

➢ FLAT better results than CACHE on KNL.

➢ Haswell peak at 32 threads and KNL at 64 threads.
GFS Phys. Chunk Size

- Chunk = 8, 12 for haswell.
- Chunk = 16, 8 for KNL.
GFS Phys. Results

➢ Scale up to 18x with 72 threads on Haswell.

➢ Scale up to 27x with 128 threads on KNL.

➢ Static scheduling performs better than dynamics.
GFS Rad. Chunks

- **Chunk = 8, 12 for haswell.**

- **Chunk = 16, 8 for KNL.**
GFS Rad. Results

➢ Scale up to **30x 72** threads on Haswell.

➢ Scale up to **23x 64** threads on KNL.

➢ Dynamics scheduling performs better.
Better runtimes with haswell because more cores and faster clock.
Discussion

<table>
<thead>
<tr>
<th>physics schemes</th>
<th>WSM6</th>
<th>GFS physics</th>
<th>GFS radiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>KNL</td>
<td>23.0</td>
<td>4.8</td>
<td>190.0</td>
</tr>
<tr>
<td>speed-up threads</td>
<td>70</td>
<td>27</td>
<td>23</td>
</tr>
<tr>
<td>configuration</td>
<td>64 dynamic+flat</td>
<td>128 static+flat</td>
<td>64 dynamic+flat</td>
</tr>
</tbody>
</table>

| Haswell           | 17.0  | 2.0         | 29.0          |
| speed-up threads  | 26    | 18          | 30            |
| configuration     | 32 dynamic | 72 static  | 72 dynamic    |

➢ Better runtimes with haswell because more cores and faster clock.

➢ Better speed-ups with KNL because better utilization of threads.
Conclusion and Future Work

➢ Code modification to use thread-local SOA.
➢ Identifying the appropriate chunk size to maximize work per thread and locality.

Future Directions

➢ Better understanding of how to improve peak performance.
➢ Study of MPI+OpenMP on larger test cases in context of NEPTUNE.
Acknowledgements:

- Intel Parallel Computing Center.
- Alex Reinecke, Kevin Viner (NRL), John Michelakes (UCAR)

Thank you!!

Questions?
E-mail: touermi@sci.utah.edu
Structured Arrays (SOA)

SOA chunk size. Corresponds to parts of i loop.

Simple example of SOA.

Figure to the right shows actual SOA used in WSM6 optimization.

Chunk size is chosen to be multiple of vector unit length.

Top down optimization approach = From “high-level” to “low-level”
Complex Loop Parallelization

- No conditional 9.7x
- No function calls 30x
- Vectorization 41x

```fortran
do k=kte,kts-1
  do i=its,ite
    ...
    if(t(i,k).gt.t0c)then
      ...
      w(i,k) = venfac(p(i,k), t(i,k), den(i,k))
      if(qrs(i,k,2).gt.0)then
        ...
        psmlt(i,k)=xka(t(i,k), den(i,k))
      end if
    if(qrs(i,k,2).gt.0)then
      psmlg(i,k)=xka(t(i,k), den(i,k))
    end if
  end if
end do
end do
```

Loop 12 from WSM6
1D Arrays Experiments

1D case

Number of threads

Speed-up

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>5</td>
</tr>
<tr>
<td>32</td>
<td>10</td>
</tr>
<tr>
<td>64</td>
<td>15</td>
</tr>
<tr>
<td>128</td>
<td>20</td>
</tr>
<tr>
<td>256</td>
<td>25</td>
</tr>
</tbody>
</table>

```
OMP SIMD

do j=2,je-1
  a(j)=0.1+c(j)/d(j)
  b(j)=(0.2+c(j-1)-c(j))/(c(j)-c(j-1)+0.5)
end do
```

1D case
1D Arrays Experiments

1D case with large array sizes

```cpp
!$OMP SIMD

do j=2, je-1
    a(j) = 0.1 + c(j) / d(j)
    b(j) = (0.2 + c(j-1) - c(j)) / (c(j) - c(j-1) + 0.5)
end do
```

Speed-up

<table>
<thead>
<tr>
<th>Number of threads</th>
<th>Original</th>
<th>Transpose</th>
<th>SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>128</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>256</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

26
2D Arrays Experiments

```
do j=2,je-1
    !$OMP SIMD
    do i=1,ie
        a(i,j)=0.1+c(i,j)/d(i,j)
        b(i,j)=(0.2+c(i,j-1)-c(i,j))/(c(i,j)-c(i,j-1)+0.5)
    end do
end do
```
2D Arrays Experiments

![Speed-up Diagram](chart.png)

```fortran
   do j=2,je-1
      !$OMP SIMD
      do i=1,ie
         a(i,j)=0.1+c(i,j)/d(i,j)
         b(i,j)=(0.2+c(i,j-1)-c(i,j))/(c(i,j)-c(i,j-1)+0.5)
      end do
   end do
```

**2D case with large array sizes**
Chunk Size

![Graph showing time versus number of threads for different chunk sizes.](image-url)
KNL Architecture

- MCDRAM: 16GB, High BW
- Peak 3 teraflops double precision
- 512 bit vectors
MCDRAM & Configurations

● **Cache Mode**
  ○ No source changes needed
  ○ Misses are expensive (higher latency)

● **Flat Mode**
  ○ MCDRAM mapped to physical address
    ■ use numactl -- for configuration
  ○ Exposed as NUMA node

● **Hybrid Mode**
  ○ Combination of flat and cache mode
    ■ eg: 8GB cache and 8GB flat