Performance of Cubed-sphere Atmospheric Dynamical Core on Intel Xeon and Xeon Phi Architectures

Hoang Vu Nguyen (KAUST)
Chris Kerr (GFDL)
Zhi Liang (GFDL)
Cubed-sphere Dynamical Core

• Used in GFDL global model
• Production code
• Run on several platforms (Xeon, BlueGene)
• Fortran 90 and later
• MPI with some OpenMP directives
Project motivation

New hardware such as the Intel Xeon Phi processor

• OpenMP on large number of cores

• Long vector

• Somewhat limited memory

• “Exascale”
## Hardware Platforms

<table>
<thead>
<tr>
<th></th>
<th>Intel Xeon E5650 (Westmere)</th>
<th>Intel Xeon E5-2660 (Sandy Bridge)</th>
<th>Intel Xeon Phi 5110P (Knights Corner)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number of sockets</strong></td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td><strong>Cores per socket</strong></td>
<td>6</td>
<td>8</td>
<td>60</td>
</tr>
<tr>
<td><strong>Core frequency</strong></td>
<td>2.66 GHz</td>
<td>2.20 GHz</td>
<td>1.05 GHz</td>
</tr>
<tr>
<td><strong>RAM</strong></td>
<td>48 GB DDR3</td>
<td>64 GB DDR3</td>
<td>8 GB GDDR5</td>
</tr>
<tr>
<td><strong>Vector width</strong></td>
<td>128 bits</td>
<td>256 bits</td>
<td>512 bits</td>
</tr>
<tr>
<td><strong>Power consumption (max)</strong></td>
<td>95W</td>
<td>95W</td>
<td>225W</td>
</tr>
</tbody>
</table>
Fyppm kernel

• A finite volume advection kernel
• Singled out from the dynamical core
• Significant portion of runtime in production run
• Test bed for blocking. Can only block horizontally, no vertical blocking due to dependency
Blocking scheme

- Using Array of Structures of Tiled-Array
- Avoid dependency using ghost cells
- Block to fit into cache
- Still suitable for vectorization
Thread parallelism

- Original OMP loop over all blocks
- Collapsed loop to improve scaling
- Needs more tasks than threads

```fortran
!$OMP do collapse(2)
    do n=1, num_block
        do k=1,npz
            call fyppm( Var(n)%c(:,:k), Var(n)%q(:,:k), &
                        Var(n)%flux(:,:k), &
                        Var(n)%dya, jord, ppm_limiter, Block(n), Grid)
        enddo
    enddo
enddo
```

```fortran
end do
!$omp end parallel
```
Vectorization

- Using directives
- Data alignment
- Needs large enough blocks

```fortran
do j=jfirst,jlast+1!
   !dec$ simd
   !dir$ vector aligned
do i=ifirst,ilast
   if( c(i,j)>0. ) then
      flux(i,j) = q(i,j-1) + (1.-c(i,j))*(br(i,j-1)-c(i,j))*(bl(i,j-1)+br(i,j-1))
   else
      flux(i,j) = q(i,j) + (1.+c(i,j))*(bl(i,j)+c(i,j))*(bl(i,j)+br(i,j))
   endif
endo
dendo
```
## Performance results

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>Intel Xeon E5650 (Westmere)</th>
<th>Intel Xeon E5-2660 (Sandy Bridge)</th>
<th>Intel Xeon Phi 5110P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original time</td>
<td>1.72</td>
<td>0.95</td>
<td>3.34</td>
</tr>
<tr>
<td>After optimization</td>
<td>1.57</td>
<td>0.55</td>
<td>0.58</td>
</tr>
<tr>
<td>Speedup</td>
<td>1.1</td>
<td>1.73</td>
<td>5.76</td>
</tr>
</tbody>
</table>
• Speedup depends on number of cores and vector length
• Good speedup even when doing strong scaling
• Optimizations improve performance on both Sandy Bridge and Xeon Phi. Same codebase
• Using only directives
Ongoing work

• Sub-blocking: breaking the i loop of each block into chunks same as vector length. Significant improvement but more manual code changes.
• Fyppm as basis for modifying the dynamical core (Kareem’s talk)
Some lessons

• Certain parts of the code is hard to vectorized. Cannot be avoided
• Threading should be scalable. Tasks instead of threads
• Vectorization is essential. OpenMP 4.0 is coming with more support.
Acknowledgment

Intel team

• Micheal Greenfield
• Gokhale Indraneil
• Larry Meadows
• Chuck Yount
References
