Cubed Sphere Dynamics on the Intel Xeon Phi

Early Experience and Progress

Kareem Sorathia
NASA/SSAI

Collaborators:
Chris Kerr (GFDL)
Craig Pelissier (NASA)
Bill Putman (NASA)
Chuck Yount (Intel)
Hoang Vu Nguyen (KAUST)
Background and Project Goals

- **Target:** Dynamical Core (FV-CS)
  - Solves fluid equations on cubed sphere geometry using a finite-volume method
  - Shared component between GFDL/NASA

- **Goal:** Port the FV-CS core to the Intel Xeon Phi

- **Constraints:**
  - Maintain one codebase for MIC/CPU
  - No CPU performance degradation
  - Minimize code invasiveness
  - Retain usability/readability for scientists
Plan of Action

- Start with solo dynamical core
- Native mode on MIC
- Requirements for performance
  - Concurrency
    - Need to generate significant OMP concurrency (>240)
  - Vectorization
    - Make use of wide (512b/8 dbl) registers
    - Ensure compiler is vectorizing workload
    - Take advantage of alignment
Solution

• Blocking
  • Additional layer of decomposition managed by OMP
  • Only decomposed in x/y
  • Blocks include halos

• Grid vs. Block
  • G: Var(is:ie, js:je, 1:npz)
  • B: Block(n)%Var(isb:ieb, jsb:jeb, 1:npz)

• High Level
  • Only changed top-level routines
  • Majority of code is not “block aware”
Cubed Sphere Dynamical Core

<table>
<thead>
<tr>
<th>Routine</th>
<th>Fraction of $fv_{\text{dyn}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>c_sw</td>
<td>10.1%</td>
</tr>
<tr>
<td>d_sw</td>
<td>36.2%</td>
</tr>
<tr>
<td>geopk</td>
<td>5.5%</td>
</tr>
<tr>
<td>tracer</td>
<td>7.8%</td>
</tr>
<tr>
<td>remap</td>
<td>2.9%</td>
</tr>
<tr>
<td>halo</td>
<td>29%</td>
</tr>
</tbody>
</table>

Will focus on:
- d_sw
- c_sw
- geopk
Structure of Slab Modules

Original

```
!$OMP DO PARALLEL
do k=1,npz
    call slab_routine(A(:,:,k)) !reference to (i,j) in region
enddo
```

Concurrency: npz

Blocked

Concurrency: Bx*By*npz
Slab Example

Original

```fortran
!$omp parallel do default(shared)
  do k=1,npz
    call c_sw(delpc(isd,jsd,k), delp(isd,jsd,k), ptc(isd,jsd,k), &
           pt(isd,jsd,k), u(isd,jsd,k), v(isd,jsd,k), &
           w(isd,jsd,k), uc(isd,jsd,k), vc(isd,jsd,k), &
           ua(isd,jsd,k), va(isd,jsd,k), omga(isd,jsd,k), &
           ut(isd,jsd,k), vt(isd,jsd,k), divgd(isd,jsd,k), &
           flagstruct%nord, dt2, hydrostatic, .true., bd, &
           gridstruct, flagstruct)
  enddo
```

Blocked

```fortran
!$OMP parallel do default(shared) &
!$OMP& private(k,n)
  do nk=1,num_blocks*npz
    k = MOD((nk-1),npz) + 1
    n = MOD(INT((nk-1)/npz),num_blocks) + 1
    call c_sw(Coreb(n)%delpc(:,;),k, Atmb(n)%delp(:,;),k, Coreb(n)%ptc(:,;),k, &
              Atmb(n)%pt(:,;),k, Atmb(n)%u(:,;),k, Atmb(n)%v(:,;),k, &
              Atmb(n)%w(:,;),k, Atmb(n)%uc(:,;),k, Atmb(n)%vc(:,;),k, &
              Atmb(n)%ua(:,;),k, Atmb(n)%va(:,;),k, Atmb(n)%omga(:,;),k, &
              Coreb(n)%ut(:,;),k, Coreb(n)%vt(:,;),k, Coreb(n)%divgd(:,;),k, &
              flagstruct%nord, dt2, hydrostatic, .true., bd(n), &
              Atmb(n)%gridstruct, flagstruct)
  enddo
```
**Structure of Column Modules**

```fortran
subroutine column_routine
    !$OMP DO PARALLEL
    do j=js,je
        do k=1,npz
            do i=is,ie
                ! references to (i,j,k) and (i,k+1,j) in region
                enddo
            enddo
        enddo
    enddo
end subroutine
```

**Concurrency:** npy

**Original**

**Blocked**

**Concurrency:** Bx*npy
Column Example

Original

```fortran
call geopk(ptop, pe, peln, delpc, pkc, gz, phis, ptc, pkz, 
           npz, akap, .true., gridstruct%nested, .false., 
           & npx, npy, flagstruct%a2b_ord, bd)
```

Intermediate

```fortran
!$omp parallel do default(shared)
    do n=1,num_blocks
        call geopk(ptop, Atmb(n)%pe, Atmb(n)%peln, Coreb(n)%delpc, Atmb(n)%pkc, 
                   & Coreb(n)%gz, Atmb(n)%phis, Coreb(n)%ptc, Atmb(n)%pkz, npz, akap, 
                   & .true., gridstruct%nested, .false., npx, npy, flagstruct%a2b_ord, bdb(n))
    enddo
```

Blocked

```fortran
!$OMP parallel do default(shared) & !$OMP& private(n,j,ifirst,ilast)
    do nj=1,nj_blocks_tf
        n   = bnj_tf(nj,1)
        j   = bnj_tf(nj,2)
        ifirst = bnj_tf(nj,3)
        ilast = bnj_tf(nj,4)
        call geopkb(Atmb(n)%pe, Atmb(n)%peln, Coreb(n)%delpc, Atmb(n)%pkc, 
                    & Coreb(n)%gz, Atmb(n)%phis, Coreb(n)%ptc, Atmb(n)%pkz, 
                    & ptop, npz, akap, .true., bdb(n), ifirst, ilast, j )
    end do
```
Vectorization

- Beginning this phase of optimization
- Vtune
  - Can use “vectorization intensity” to prioritize routines
  - Many metrics are difficult to interpret
- Anecdote (Geopk)
  - 3 SIMD directives ➔ 50% imp. on MIC, 25% on SNB
- Sub-blocking
  - Requires refactoring loops
  - 30% improvement on FYPPM (Vu’s talk)
  - How do we use this?
- Alignment
  - Can we benefit from this?
Overhead

- Intrinsic: (G)rid (B)lock
  - $G \rightarrow B$ & $B \rightarrow G$, each call to dynamics
  - $\sim (1-3)\%$

- Scaffolding
  - MPI routines/diagnostics require $G$
    - Extra $G \rightarrow B$ & $B \rightarrow G$ for each call
  - Memory footprint ($G$ memory)
Status

- Completed
  - Fully blocked dynamical core
    - Caveats: Scaffolding
  - OMP over n/k & n/j
  - Concurrency w/ minimal code changes
    - Slab routines unchanged, refactoring to column routines

- In progress
  - Vectorization (Increased intensity/starting alignment)
  - Minimizing independent // regions
Results

Run Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>C96</td>
</tr>
<tr>
<td>Vertical Levels</td>
<td>71</td>
</tr>
<tr>
<td>Test Case</td>
<td>Held-Suarez</td>
</tr>
<tr>
<td>Model Time</td>
<td>6hrs</td>
</tr>
<tr>
<td>Precision</td>
<td>64b</td>
</tr>
</tbody>
</table>

Hardware

<table>
<thead>
<tr>
<th>Hardware</th>
<th>6xSNB Nodes</th>
<th>6xKNC (5110P)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPI/Node</td>
<td>16</td>
<td>1</td>
</tr>
<tr>
<td>OMP/MPI</td>
<td>0</td>
<td>240 (Max)</td>
</tr>
</tbody>
</table>

Key Routines

- $b_x=1$, $b_y=3$ (Slab Best)
- $b_x=2$, $b_y=4$ (Column Best)

- Approx. SNB performance (vs. unblocked)
- Poor response to finer blocking, esp. Bx
- Need to understand this, as column routines depend on this
Results - Overall

- Real codes have many routines that are neither negligible nor dominant
- Want to make an end to end comparison

<table>
<thead>
<tr>
<th>Routine</th>
<th>SNB</th>
<th>KNC (1x3)</th>
<th>Slowdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>FV_RESTART</td>
<td>1.66</td>
<td>296.79</td>
<td>178.5x</td>
</tr>
<tr>
<td>HALO</td>
<td>2.26</td>
<td>106.52</td>
<td>22x</td>
</tr>
<tr>
<td>TRACER</td>
<td>0.17</td>
<td>1.88</td>
<td>10.6x</td>
</tr>
<tr>
<td>REMAP</td>
<td>0.27</td>
<td>1.04</td>
<td>3.8x</td>
</tr>
<tr>
<td>1GRAD_P</td>
<td>0.64</td>
<td>2.0</td>
<td>3.1x</td>
</tr>
</tbody>
</table>

Initialization (No OMP)

C_SW/D_SW/GEOPK (SNB) : 1.18/4.55/1.44
C_SW/D_SW/GEOPK (KNC) : 1.39/4.05/1.18
Conclusions

- Time and expertise required for development
- Success requires collaboration (vendors/software/science)
- Need restructuring on MIC for concurrency, vectorization, and alignment (2 months/3 ppl)
- After restructuring, SNB and MIC have similar performance
- Work harder to achieve performance on MIC, but
  - Time spent yields improvement on SNB
  - Prepares for the future (wider registers/multi-core/larger-scale)
### Runtime environment

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIC_OMP_STACKSIZE</td>
<td>2M</td>
</tr>
<tr>
<td>MIC_USE_2MB_BUFFERS</td>
<td>64K</td>
</tr>
<tr>
<td>MIC_KMP_AFFINITY</td>
<td>“granularity=thread,balanced”</td>
</tr>
<tr>
<td>FFLAGS_OPT</td>
<td>-O3 -debug minimal -fp-model precise -override-limits -align array64byte</td>
</tr>
<tr>
<td>Compiler</td>
<td>intel-13.1.1.163</td>
</tr>
<tr>
<td>MPI</td>
<td>impi-4.1.0.036</td>
</tr>
</tbody>
</table>