Using GPUs for ICON: An MPI and OpenACC Implementation

William Sawyer, Jeff Poznanovic, Leonidas Linardakis (MPI-M), Christian Conti (ETH), Gilles Fourestey

Earth System Models on Heterogeneous Multicore
Sep. 20, 2013, CSCS, Boulder, USA
ICON NWP/Climate Model

- ICOsahedral Non-hydrostatic model
- Dynamical core: conservation laws
- Triangular cells
- Nested grid
- Memory bandwidth limited
- Extensive use of indexing arrays
- Developers: MPI-M, DWD
Goal: Implement a GPU-capable version of the ICON Non-hydrostatic dynamical core (NHDC) currently under development at the Max Planck Institute for Meteorology (MPI-M) and German Weather Service (DWD)

- **Completed:** OpenCL single-node NHCD implementation
- **Completed:** CUDAFortran single-node NHDC implementation
- **Presented:** results of single-node versions (e.g. Boulder, Sep. 2011)
- **Completed:** Refactored multi-node NHDC (based on r0211 DSL testbed Jun. 2012) in preparation for subsequent GPU implementation
- **Completed:** GPU-capable multi-node NHDC using OpenACC directives within the ICON domain-specific language (DSL) testbed
- **Planned:** implementation in main development trunk
Recap: Port to single-node GPU

**Fortran**

```
DO jb = i_startblk, i_endblk
   CALL get_indices_c(p_patch, jb, i_startblk, i_endblk, &
        i_startidx, i_endidx, rl_start, rl_end)
DO jk = 1, nlev
   DO jc = i_startidx, i_endidx
```

**CUDAFortran**

```
jb = blockIdx%x + ( i_startblk - 1 )
je = threadIdx%x
jk = threadIdx%y
IF ( ( i_startblk < jb .and. jb < i_endblk ) .or. &
    ( i_startblk == jb .and. i_startidx <= je ) .or. &
    ( i_endblk == jb .and. je <= i_endidx ) ) THEN
```

**OpenCL**

```
const int jb = i_startblk + get_global_id(0);
const int jc = localStart[get_global_id(0)] + get_global_id(2);
const int jk = get_global_id(1);
if (jk < nlev && jb < i_endblk && jc < localEnd[get_global_id(0)])
{
   const int id = jc + jk*nproma + jb*nproma*nlev;
```
Aggregated NH Performance (DP)

- Fermi M2050 (CUDAFortran):
  - R2B3 (5120 cells x 35 lev): 18.8 GFLOP/s
  - R2B4 (20480 cells x 35 lev): 33.0 GFLOP/s
- Cayman (OpenCL):
  - R2B4: 21.2 GFLOP/s
Single Node Prototype Results

<table>
<thead>
<tr>
<th></th>
<th>R2B03 (s.)</th>
<th>R2B04 (s.)</th>
<th>R2B05</th>
</tr>
</thead>
<tbody>
<tr>
<td>17.2</td>
<td>47.6</td>
<td>168</td>
<td></td>
</tr>
<tr>
<td>53.8</td>
<td>82.4</td>
<td>201</td>
<td></td>
</tr>
<tr>
<td>27.6</td>
<td>101</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>13.3</td>
<td>38.6</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>36.3</td>
<td>120</td>
<td>450</td>
<td></td>
</tr>
<tr>
<td>27.8</td>
<td>60.4</td>
<td>192</td>
<td></td>
</tr>
<tr>
<td>21.5</td>
<td>65.7</td>
<td>241</td>
<td></td>
</tr>
<tr>
<td>31.6</td>
<td>56.8</td>
<td>159</td>
<td></td>
</tr>
<tr>
<td>11.3</td>
<td>47.5</td>
<td>202</td>
<td></td>
</tr>
<tr>
<td>13.9</td>
<td>59.3</td>
<td>240</td>
<td></td>
</tr>
<tr>
<td>66.3</td>
<td>273</td>
<td>1089</td>
<td></td>
</tr>
<tr>
<td>21.1</td>
<td>89.7</td>
<td>326</td>
<td></td>
</tr>
<tr>
<td>26.9</td>
<td>114</td>
<td>457</td>
<td></td>
</tr>
<tr>
<td>81.1</td>
<td>327</td>
<td>1492</td>
<td></td>
</tr>
<tr>
<td>23.1</td>
<td>112</td>
<td>472</td>
<td></td>
</tr>
<tr>
<td>39.8</td>
<td>158</td>
<td>639</td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>579</td>
<td>2424</td>
<td></td>
</tr>
</tbody>
</table>

- **Interlagos**
  - Core (1 Thread)
  - Socket (16 Threads)
  - Node (26-30 Threads)

- **Westmere**
  - Core (1 Thread)
  - Socket (6 Threads)
  - Node (12 Threads)

- **Sandybridge**
  - Core (1 Thread)
  - Socket (8 Threads)
  - Node (16 Threads)

- **Kepler K20X**
  - (OpenCL)
  - (CUDA Fortran)

- **GTX 480**
  - (OpenCL)
  - (CUDA Fortran)

- **Tesla M2090**
  - (CUDA Fortran)

- **GTX 480**
  - (CUDA Fortran)

- **Kepler K20X**
  - (CUDA Fortran)

- **GTX 480**
  - (CUDA Fortran)

- **GTX 480**
  - (CUDA Fortran)

- **GTX 480**
  - (CUDA Fortran)

- **GTX 480**
  - (CUDA Fortran)
Single-node prototype NHDC: lessons learned

- Never underestimate the potential of a smart, motivated graduate student (in this case Christian Conti)!
- CUDA/OpenCL programming not that difficult, but highly error-prone; debugging options limited; code validation crucial
- CUDAFortran is much more ‘appealing’ to developers; OpenCL is the more portable paradigm (but OpenCL 1.2/2.0 not supported by NVIDIA!!)
- Optimizations to both versions still possible
- Feedback from ICON developers: OpenCL and CUDAFortran not viable for production version
- Only valid option for multi-node version: OpenACC ‘standard’
ICON NHDC Example:
mean normal, tangent winds

!ICON_OMP_DO_STD PRIVATE(jb,i_startidx,i_endidx,jk,je, iqidx_1,iqblk_1,...)
   DO jb = i_startblk, i_endblk
!ICON_OMP_TASK_STD PRIVATE(i_startidx,i_endidx,jk,je, iqidx_1, iqblk_1,...) firstprivate(jb)
   CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, 
                        i_startidx, i_endidx, rl_start, rl_end)
   DO je = i_startidx, i_endidx
      iqidx_1 = iqidx(je,jb,1)
      DO jk = 1, nlev
         ! Average normal wind components
         ptr_vn(je,jk,jb) = p_int%e_flx_avg(je,1,jb)*p_nh%prog(nnew)%vn(je,jk,jb)&
                           + p_int%e_flx_avg(je,2,jb)*p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
         ! RBF reconstruction of tangential wind component
         p_nh%diag%vt(je,jk,jb) = p_int%rbf_vec_coeff_e(1,je,jb) &
                        * p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
      ENDDO
   ENDDO
!ICON_OMP_END_TASK
ENDDO
!ICON_OMP_END_DO
!ICON_OMP_WAIT_TASKS

ICON DSL primitives  Private indices  First/last block correction
Block number  Block size (usually 4 or 8)  Derived types
ACC copies outside time loop

i_qidx_d = p_patch(1)%edges%quad_idx
i_qblk_d = p_patch(1)%edges%quad_blk
e_flx_avg_d = p_int_state(1)%e_flx_avg
prog_vn_now_d = p_nh_state(1)%prog(nnow(1))%vn
rbf_vec_coeff_e_d = p_int_state(1)%rbf_vec_coeff_e

 !$ACC DATA COPY(i_qidx_d, i_qblk_d, ..., e_flx_avg_d, prog_vn_now_d, rbf_vec_coeff_e_d, ...)

TIME_LOOP: DO jstep = 1, nsteps
  ! dynamics stepping
  CALL integrate_nh(p_nh_state, p_patch, p_int_state, datetime, ... )
ENDDO TIME_LOOP

 !$ACC END DATA

Invocation (inside non-hydrostatic solver):

rl_start = 3
rl_end = min_rledge_int - 2
i_startblk = p_patch%edges%start_blk(rl_start,1)
i_endblk = p_patch%edges%end_blk(rl_end,i_nchdom)
e_startidx = GET_STARTIDX_E(rl_start,1)
e_endidx = GET_ENDIDX_E(rl_end, MAX(1,p_patch%n_childdom))

#include "vn_and_vt_alt.inc"
ICON DSL OpenACC Implementation

!$ACC PARALLEL &
!$ACC PRESENT( iqidx_d, ..., ptr_vn_d, e_flx_avg_d, vn_d, vt_d, rbf_vec_coeff_e_d )
!$ACC LOOP GANG PRIVATE( i_startidx, i_endidx, jb )
  DO jb = i_startblk, i_endblk
    IF ( i_startblk == jb ) THEN; i_startidx = e_startidx; ELSE; i_startidx = 1; ENDIF
    IF ( i_endblk == jb ) THEN; i_endidx = e_endidx; ELSE; i_endidx = nproma; ENDIF
  !$ACC LOOP VECTOR
  !DIR$ loop_info max_trips(MAX_NPROMA)
  DO je = i_startidx, i_endidx
    iqidx_1 = iqidx_d(je,jb,1)
    DO jk = 1, nlev
      ! Average normal wind components
      ptr_vn_d(je,jk,jb) = e_flx_avg_d(je,1,jb)*vn_now_d(je,jk,jb) &
        + e_flx_avg_d(je,2,jb)*vn_now_d(iiqidx_1,jk,iqblk_1) &
        : 
      ! RBF reconstruction of tangential wind component
      vt_now_d(je,jk,jb) = rbf_vec_coeff_e_d(1,je,jb) &
        * vn_now_d(iqidx_1,jk,iqblk_1) &
        :
      ENDDO
  ENDDO
!$ACC END PARALLEL

Block size (usually 128-512)
GPU implementation of communication

**ORIGINAL:**
```
DO i = 1, p_pat%n_send
    send_buf(1:ndim2,i) = send_ptr(p_pat%send_src_idx(i),1:ndim2, &
    p_pat%send_src_blk(i)-lbound3+1)
ENDDO
```

**ACCELERATED:**
```
!/ACC DATA CREATE( send_buf, recv_buf )
!/ACC PARALLEL &
!/ACC PRESENT ( p_pat%send_src_idx, p_pat%send_src_blk, sendrecv )
!/ACC LOOP
    DO i = 1, n_send
        send_buf(1:ndim2,i) = sendrecv(p_pat%send_src_idx(i),1:ndim2, &
        p_pat%send_src_blk(i))
    ENDDO
!/ACC END PARALLEL
!/ACC UPDATE HOST( send_buf )
```
OpenACC:
Issues with derived types

Cray CCE allows specification of individual members of derived types, e.g.,

```fortran
!$ACC DATA COPY(p_nh_state(1)%prog(1)%theta_v)
```

PGI: This yields a syntax error. PGI supports only:

```fortran
!$ACC DATA COPY(p_nh_state(1))
```

This will copy the instance p_nh_state(1), but it will not make a deep copy of
of the member array (pointers) onto the GPU

In order to fully support (nested) ICON, selective deep copy needed

```fortran
!$ACC DATA COPY(p_nh_state(:)%prog(:)%theta_v)

!$ACC PARALLEL PRESENT( p_nh_state(grid_no)%prog(nnow)%theta_v)
```

We are communicating this to vendors and to the OpenACC consortium
MPI+OpenACC: How long did it take?

Creation of shadow arrays for all fields 15 days
Moving data region outside main time loop 10 days
Validation infrastructure (needed for debugging) 25 days
Merging in latest software releases 10 days
Insertion of directives (NHDC solver) 2 days
Insertion of directives (Communication) 3 days
Tweaking of directives, compiler workarounds 5 days
Optimization of directives for best performance 10 days
(many thanks to Cray’s Vince Graziano)

Perhaps a full code rewrite is not prohibitive
MPI+OpenACC XK7 results: Interlagos Socket vs. K20x

• On Cray XK7: compare nodes with only Interlagos socket with Interlagos-Kepler K20x nodes (Cray CCE)

- R2B05, R2B06, R2B07, 35 levels
- Accelerated code >2x faster
- For OpenMP, GNU yields much better performance than CCE
- R2B05: MPI+OpenACC performance tails off at high node numbers

• unfair comparison
  * non-optimal CPU runs?
  * CPU socket in GPU version essentially idle
MPI+OpenACC first results: Sandybridge node vs. K20x

- Compare original (GNU) on Cray XC30 (2x Sandybridge sockets) vs. XK7 node with Kepler K20x (Cray CCE)
  - Fairer comparison
  - OpenACC faster for cases where memory is fully exploited
  - Weak scaling comparable, CPU strong scaling better
  - OpenACC version can be further optimized (compare to single-node prototypes)

- After optimizations: MPI+OpenACC factor 2x for cases of interest
Roofline performance model revisited

ICON is memory bandwidth-bound, consider linear regime of roofline model based on stream benchmark

\[ t(lev, iter, b_{\text{max}}) = \frac{0.00735 \times iter \times 4^{lev}}{0.38 \times b_{\text{max}}} \]

- 0.00735 G-ops (DP) per iteration
- 0.38 average operations (DP) per byte memory access
- \( lev \): level of refinement
- \( iter \): number of iterations
- \( B_{\text{max}} \): memory bandwidth from stream benchmark e.g., 2xSandy: ~70 GB/s, K20x: ~170 GB/s per node
Performance model vs. observed

2x Sandybridge: Perf. Model vs. Observed

Kepler K20x Model vs. Observed
Final Challenge: Re-integration into ICON community

- Final Integration into the Domain Specific Language (DSL) testbed version of ICON (discussions with Leonidas Linardakis and Alex Koehler at MPI-M)

- Continuation: Integration into main ICON development trunk (with Guenther Zaengl, DWD). Goals:
  - Minimal modifications other than directives
  - Support nested grids
  - Performance comparable to testbed, e.g., 2x vs. Dual-Sandy

First attempts made -- significant challenges with derived types
OpenACC Coding Objective:
mean normal, tangent winds

!$ACC PARALLEL &
!$ACC PRESENT( iqidx, ..., ptr_vn, p_int%e_flx_avg, p_nh%prog(nnew)%vn, &
!$ACC p_nh%diag%vt, p_int%rbf_vec_coeff_e )
!$ACC LOOP GANG PRIVATE( i_startidx, i_endidx, jb )
  DO jb = i_startblk, i_endblk
    CALL get_indices_e(p_patch, jb, i_startblk, i_endblk, &
    i_startidx, i_endidx, rl_start, rl_end)
  !$ACC LOOP VECTOR
    DO je = i_startidx, i_endidx
      iqidx_1 = iqidx(je,jb,1)
      DO jk = 1, nlev
        ! Average normal wind components
        ptr_vn(je,jk,jb) = p_int%e_flx_avg(je,1,jb)*p_nh%prog(nnew)%vn(je,jk,jb) &
        + p_int%e_flx_avg(je,2,jb)*p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
        ! RBF reconstruction of tangential wind component
        p_nh%diag%vt(je,jk,jb) = p_int%rbf_vec_coeff_e(1,je,jb) &
        * p_nh%prog(nnew)%vn(iqidx_1,jk,iqblk_1) &
      ENDDO
    ENDDO
  ENDDO
!$ACC END PARALLEL

OpenACC directives    Private indices    First/last block correction
Block number          Block size (usually 128-512) Derived types
OpenACC coding objective: selective deep copies

!$ACC DATA COPY(  p_patch(1)%edges%vertex_blk,                  p_patch(1)%edges%vertex_idx,          &
  p_patch(1)%comm_pat_v%n_send,                  p_patch(1)%comm_pat_v%n_pnts,          &
  p_patch(1)%comm_pat_v%send_src_idx,            p_patch(1)%comm_pat_v%send_src_blk,     &
  p_nh_state(1)%prog(nnow(1))%vn,                p_nh_state(1)%prog(nnew(1))%vn,        &
  p_nh_state(1)%diag%vn_ie,                      p_nh_state(1)%diag%vt,                 &
  :
  TIMELOOP: DO jstep = 1, nsteps

      ! Lots of stuff we won't put on the GPU at this time

      CALL integrate_nh(datetime, 1, jstep, dtime, dtime_adv, 1)

      :

  ENDDO TIME_LOOP

!$ACC END DATA
Community needs OpenACC support for derived types

The ICON development trunk is different than the testbed in that there is far less flexibility to rewrite the code.

I think the support of derived type members is a key aspect for the usefulness of OpenACC for modern codes (not only ICON), because working around that would really break our coding structure.

Guenther Zaengl, ICON lead developer
OpenACC: Reflections

- OpenACC is the right idea: try to consolidate accelerator functionality into standardized directives
- OpenACC is not yet mature; significant functionality missing, vendors may interpret and implement standard differently, e.g. derived types
- Inserting directives may be quick, but refactoring and optimizing code for GPU are not
- Ultimately, ICON should offer an OpenACC implementation as it does OpenMP; this mitigates vendor dependencies
• Funding: PRACE 2IP WP8, Grant RI-283493
• ICON team (MPI-M/DWD): collaborative effort
• CSCS Future Systems and User Support staff
• Technical support from:
  ‣ Cray: Vince Graziano, others...
  ‣ PGI: Michael Wolfe, Mat Colgrove, others...
  ‣ NVIDIA: Peter Messmer, others...
• Thanks to you for listening!
Other possible optimizations

• Improve strong scalability of MPI+GPU version: use GPUdirect message passing within MVAPICH2

• Optimize OpenACC implementation: further refactoring of vertical solver and geopotential (theta) kernels
ORNL Caveats

- Jeff Poznanovic, our consortium representative has left
- Our priority is our users, i.e., the final application. OpenACC is the means to that end
- Our wish list will seem asking for the moon
- We are generally not concerned about special cases -- we might not understand the implication and complexity of the extensions we suggest. Reasonable limitations to the new functionality are probably OK
- I am not a compiler developer; the OpenACC forum discussion humbles me; I’m just learning the lingo
Essentially what was sent to Jeff Sandoval, compressed into one slide.
Wish list

- We would like to achieve the following aliasing solution
- without explicitly adding the aliases:
• Our derived types do not have to be indivisible
• Just learned that deep copies would be straightforward in Fortran code, thanks to the dope vectors. This might be sufficient for us?