Porting Work and Optimizations provided by NEC for ICON

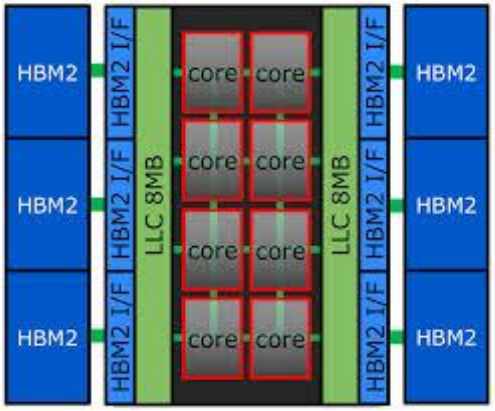
COSMO GM 2020

Günther Zängl

Overview

- A few technical details on DWD's upcoming HPC system
- Optimizations implemented and provided by NEC in the context of the procurement
- Performance comparison Cray XC 40 – NEC Aurora for ICON
- Differences in performance behaviour w.r.t. NEC SX-9
- A COSMO example
- Vector vs. GPU

Technical Details



➔A computing node consists of

- ➔a vector host: 24-core AMD Rome (2.8 GHz; 256 GB memory)
- ➔8 vector engines: SX-Aurora 1 TSUBASA Typ 10AE
- ➔Every vector engine has
  - ➔8 cores (1.584 GHz; 304.1 GF/s (DP); 608.3 GF/s (SP) per core)
  - ➔48 GB HBM2 3D-stacked memory (6 GB/core; 1.35 TB/s)
  - ➔and is direct liquid cooled

➔Number of nodes / engines

Phase	Operations	Experiments
0	178 / 1424	232 / 1856
1	224 / 1792	292 / 2336
2	224+101/3408	292+132 / 4448

Technical Details (cont'd)

- ➔ Infiniband HDR
- ➔ Peak Performance (DP, op. system): 0: 3383 TF/s; 1: 4260 TF/s; 2: 8332 TF/s
- ➔ Power efficiency is competitive with GPUs:
- ➔ typical power usage in Phase 2: 777 kW (operations); 981 kW (research); this is about 70 % compared to Intel and about 80 % compared to AMD for 75% of the computing power offered by NEC.

- ➔ The infiniband network is not as fast as the Aries network of our current Cray, in particular, the latencies are much larger
- ➔ This reflects the fact that the emphasis of our procurement was on ‚capacity', i.e. the ability to calculate in parallel as many ensemble members as possible

Optimizations implemented and provided by NEC

- Optimization of MPI communication during the computation of the domain decomposition (DD)
- Communication is blocked into vectors rather than doing it grid point by grid point → greatly reduces number of communication calls
- Currently #ifdef‘d by `__BLOCK__GET__`
- Reduces total computing time for DD by a factor of two on the NEC, but not much difference on the Cray due to very small MPI latencies
- Nevertheless, there might be other platforms than our NEC taking benefit from the blocked communication

Optimizations implemented and provided by NEC

- Task parallelism for reading atmospheric input data
- Using OpenMP sections, reading data, distributing data, and computing input statistics can be parallelized
- Can be activated with `use_omp_input` in `parallel_nml`
- Speeds up reading input data by about 20% on the NEC, beneficial impact on Cray only if hyperthreading is turned off

Optimizations implemented and provided by NEC

- ➔ Hybrid mode: offload I/O tasks to vector hosts; this involves two binaries running in parallel that communicate via MPI
- ➔ Motivation: I/O is faster on vector hosts rather than vector engines, particularly for GRIB2 files
- ➔ Easy to accomplish for asynchronous tasks (output, latbc-prefetch), but requires excluding PE0 from domain decomposition (would otherwise slow down vector PEs during runtime)
- ➔ Implemented via namelist switch `proc0_shift` in `parallel_nml`; this allows technical tests on any platform
- ➔ Speeds up setup phase by a factor of 2 and total runtime by 20-30% when writing routine-equivalent output

Optimizations implemented and provided by NEC

- ➔ Reduction of number of mtime - iscurrenteventactive calls
- ➔ Motivation: mtime calls, in particular of the function ‚iscurrenteventactive' are extremely expensive
- ➔ Replaced by vector-host offloading with subsequent broadcast
- ➔ Reduces total computing time by 1% - 3% depending on domain size per core (i.e. scaling level)
- ➔ Remark: mtime is also quite expensive on x86-CPUs: in the strong-scaling limit, the mtime overhead reaches a few per cent even on our Cray

Optimizations implemented and provided by NEC

Bits and pieces

- ➔ Replacement of old SX9 directives by Aurora directives
  - ➔ (!$NEC instead of !CDIR)
- ➔ Vectorization modifications / vectorized variants for some loops / routines (not all of them were correct)
- ➔ Workarounds for optimization / vectorization bugs of the compiler (will hopefully be removed soon)

Further remarks

- ➔ Fortunately, modifications for metadata communication to asynchronous output PEs (which were quite ugly) turned out to be unnecessary if PE0 runs on vector host as well
- ➔ Another modification in `mo_name_list_output` to reduce the number of `MPI_win_lock`‘s could be replaced by a bug fix that prevents the namelist variable for the chunk size from being overwritten

Performance Comparison Cray-NEC

- Model configuration: global deterministic R3B7N8, 7.5-day forecast needs to be completed in about 50 min
- Cray: 2952 Broadwell cores (82 nodes, 8.4% of routine system)
- NEC: 352 VE + 14 VH cores (5.5 nodes; 3.1% of phase 0)

- Runtime fractions of main components (NEC / Cray)
  - dycore incl. diffusion: 45% / 55%
  - transport: 14% / 16%
  - physics: 36% / 25%
- This is qualitatively as expected due to the higher memory bandwidth per flop, but not as pronounced as on the SX-9
- MPI communication generally consumes a larger fraction of the computing time than on the Cray

Differences in performance behaviour compared to SX-9

- ➔ Index lists are no longer as beneficial as they were on the SX-9
- ➔ Example: graupel microphysics scheme
- ➔ ftrace shows 91 GF/core for the standard variant vs. 36 GF/core for the index-list-based variant (which will be removed again after successful verification tests); this overcompensates the larger number of calculations done without index lists

- ➔ Miura scheme in dycore can use the same code as conventional CPUs; on the SX-9, precomputed back-trajectory fields were much faster

Vector vs. GPU Optimization

An example from the COSMO-Model

COSMO GM 2020

Ulrich Schättler

Problematic Construct: DO WHILE loops

- ➔ Porting the COSMO-Model to the SX-Aurora brought up one issue, where we had to use different implementations to get an optimized code on SX and on GPUs
- ➔ Computation of the Lightning Potential Index LPI:
  - ➔ Implementation in the COSMO-Model uses a Newton-Method to find a zero of a function (has to be done per grid point).
  - ➔ Uses a DO WHILE loop until a convergence criterion is met or a maximum iteration count is reached.

SX- and GPU implementation

```
DO <horizontal loops over ij>
   some initializations
ENDDO
DO WHILE (<MAX criteria>)
  DO <horizontal loops over ij>
    IF (criteria per ij)
       … computations …
    ENDIF
    compute MAX criteria for all ij
  ENDDO
ENDDO ! DO WHILE
```

➔ Vectorization has to be over horizontal loops ij (not possible over iteration count in DO WHILE)

➔ DO WHILE must be the outermost loop

```
DO <horizontal loops over ij>
   some initializations

   DO WHILE (criteria per ij)
       … computations …
       compute criteria per ij
   ENDDO   ! DO WHILE

ENDDO <horizontal loops over ij>
```

➔ Computation of a „MAX criteria“ would be unnecessary and expensive on the GPU

➔ DO WHILE should be the innermost loop

Vector vs. GPU Optimization

A Look to the Future

Problematic for Vector

- ➔ Innermost loop over vertical levels
- ➔ Which seems to be necessary for the SNOWPOLINO multi-layer snow scheme
  - ➔ No problem any more for COSMO, because we will sure not run it any more on the NEC SX
  - ➔ What to do in ICON?