Achieving Performance Portability for the Math Library in ICON with C++/Kokkos

Pradipta Samanta *Deutsches Klimarechenzentrum (DKRZ), Hamburg* *samanta@dkrz.de* **km-Scale Summit 2026**

The Portability Challenge in ICON

ICON runs on CPUs and GPUs from one Fortran code base — but the two want the work organized in opposite ways:

CPU

many small blocks of grid columns
one block per core — fits into fast cache memory
cores work on their blocks in parallel

GPU

one huge block with all grid columns
thousands of GPU threads work on it at once
small blocks would leave the GPU idle

- Steering the GPU needs extra directives in the code (OpenACC) — in practice these work reliably **only on NVIDIA hardware**
- Moving ICON to the AMD GPUs of LUMI took **~5 person-years** — and only a single compiler can build ICON there

How do we avoid repeating this effort for every new machine?

Why C++/Kokkos?

- **Write once, run everywhere**
  the same loop code runs on NVIDIA GPUs, AMD GPUs, or CPUs — chosen when ICON is built
- **Open and vendor-neutral**
  open source from US national labs, widely used in science — evaluated by DKRZ & MPI-M
- **Works with ICON's arrays as they are**
  Kokkos arrays adopt Fortran's memory layout directly — no conversions, no copies

Our strategy — incremental rewriting: an all-at-once rewrite would halt science and risk operations; we port one component at a time, validate it against the Fortran baseline, and run both side by side.

ICONMath: The Math Library

ICON's math library — 39 Fortran routines, rewritten in C++/Kokkos:

22 **horizontal**
divergence · gradient
rotation · Laplacian

13 **interpolation**
cells↔verts↔edges
Radial Basis Function

4 **support**
constants · geometry
Random Number Generator

- Plugs into ICON as an external library — the model code is unchanged
- The Fortran version stays as the baseline: the rewrite must reproduce its results exactly

What the Rewrite Looks Like

One interpolation loop, before and after — the only code on this poster:

Original Fortran (with OpenACC)

```
!$ACC PARALLEL PRESENT(...)
!$ACC LOOP GANG VECTOR COLLAPSE(2)
DO jk = slev, elev
  DO jv = i_startidx, i_endidx
    p_out(jv,jk) = c(jv,1) * p_in(...)
  END DO
END DO
!$ACC END PARALLEL
```

C++/Kokkos

```
// how to run: 2D range, GPU or CPU
Kokkos::MDRangePolicy<Rank<2>> policy(
    {slev, i_start}, {elev, i_end});
Kokkos::parallel_for(policy,
  KOKKOS_LAMBDA(int jv, int jk) {
    // what to run: the same formula
    p_out(jv,jk) = c(jv,0) * p_in(...);
});
```

The red directives are the problem:
they steer the parallelization by hand — and they are what breaks on new hardware.

The loop is written once:
Kokkos decides at build time how to run it — as GPU threads or on CPU cores.

The numerics are untouched — only how the loop is launched changes.

The Fortran ↔ C++ Bridge

The new C++ routines slot in behind the existing Fortran interfaces:

ICON (Fortran)
calls each math routine exactly as before

↓ same call, same arguments

Thin bridge layer
translates array sizes and index conventions automatically

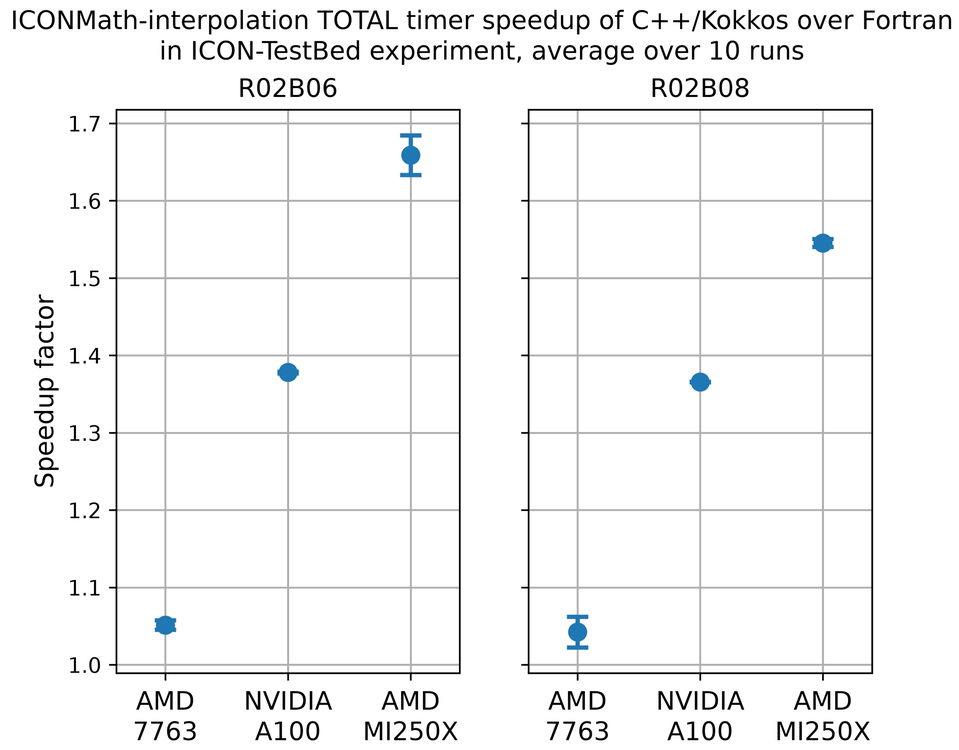
↓ hands the arrays over — no copies

ICONMath (C++/Kokkos)
one portable implementation of all 39 routines

CPU · **NVIDIA GPU** · **AMD GPU**

Performance: Total Interpolation Time

Speed-up of C++/Kokkos over the original Fortran/OpenACC (higher = faster) · mean of 10 runs
Grids: R2B6 (40 km) & R2B8 (10 km) · Levante: NVIDIA A100 GPU, AMD CPU · LUMI: AMD MI250x GPU



- One source, faster on **both GPU vendors**: **1.38×** NVIDIA A100 · **1.66×** AMD MI250x
- Just as fast on CPUs (**1.04–1.05×**) — portability costs nothing
- Biggest gains in the **RBF interpolation kernels**: up to **3.3×** faster

What This Means for ICON Users

✓ **Nothing changes in how you run ICON**
same build, same namelists, same workflows — the new library drops in underneath

✓ **Same results**
every ported routine is validated against the original Fortran

✓ **Ready for the next machines**
a new architecture means a rebuild — not a multi-year porting project

Summary & What's Next

✓ All 39 math routines: one C++/Kokkos source for NVIDIA, AMD, and CPUs

✓ GPU: **1.38–1.66×** faster than Fortran/OpenACC · CPU on par · results bit-identical

→ Next: apply the same approach across more of ICON towards **a Fortran-free model**

One portable C++/Kokkos source — legacy Fortran and new code coexist, no disruption

Acknowledgements — DKRZ: J. Behrens · C. Frauen · D. Kierans · G. Mania · R. Müller · A. Sedighi
MPI-M: H. Balasubramanian · O. Heidmann · S. Kosukhin | TUM: Y.-C. Chen · G. Lucas | CSC: J. Enkovaara
Part of **WarmWorld Faster** · Compute: Levante (DKRZ), LUMI (CSC/EuroHPC) | *samanta@dkrz.de* · ICON is open source: *icon-model.org*

With funding from the: Federal Ministry of Research, Technology and Space