E3SM All Hands: NGD Nonhydrostatic Atmosphere: Performance-Portable Physics Parameterizations

Andrew M. Bradley, Jim Foucar, SCREAM Team
Outline

1. Machines
2. Methods
3. Mini-app
Outline

1. Machines

2. Methods

3. Mini-app
Summit

- 4608 IBM Power9 nodes
- 27,648 16GB Nvidia V100 GPUs
- 2560 double precision cores/GPU (80 SM × 32 cores/SM)
- 32 threads/core

Nonhydrostatic 3km atmosphere

- \( n_e = 1024 \)
  - ★ 6,291,456 horizontal elements
  - ★ 56,623,106 columns
- 128 levels

Implied minimum parallelism in each column

- \(~2048\) columns/GPU
- Must use \(\geq 40\) threads/column
- In practice, 2–4 cores controlled by a thread block (Cuda) = team (Kokkos)
- Must parallelize in each column.

1/4-degree model \((n_e = 120, 72\) levels) with 2 (3) cores/column can occupy \(~608\) \((\sim973,\) accounting for unused 2 cores/SM when team size is 3) GPUs.
Outline

1. Machines
2. Methods
3. Mini-app
Methods: Expose all Parallelism

- C++/Kokkos
- Hierarchical parallelism
- Team per column
- Parallel map (for), reduction, scan within team
- Kokkos tutorial and documentation:
  - github.com/kokkos/kokkos/tree/master/doc
  - github.com/kokkos/kokkos-tutorials
  - github.com/kokkos/kokkos/tree/master/example/tutorial
  - github.com/kokkos/kokkos-tutorials/tree/master/Intro-Full/Slides
Vector processing units (VPU) are important on current CPU/KNL.

Adding VPUs to an architecture is an efficient means to increase FLOPS and optimize use of available memory bandwidth.

Thus, I predict VPUs will never go away.

V100 already supports limited vector intrinsics.

Implementation:

- Fortran auto-vectorizes well if code is written carefully.
- C++ does not.
- But C++ easily supports Pack and Mask types.
  - Bonus: Vectorization is roughly independent of compiler.

SCREAM solution:

- Pack: Multiple scalars packed together, respecting memory alignment and vector width.
- Mask: Conditionals (e.g., if statements).
Methods: Manage Memory and Temporaries

- Moving data is expensive . . .
- . . . and becomes relatively more expensive with time.
- Must handle
  - Communication between devices
  - Global data on a device
  - Local data
    - Per team
    - Per thread

Temporaries implementation:

- Need reusable workspace shared among threads in a team.
- Minimize global memory footprint.
- SCREAM solution: WorkspaceManager.
  - Request and release column-friendly temporary arrays.
  - User-friendly and aggressive performance API options.
    - Start with user-friendly API.
    - When everything works, optimize with aggressive performance API.
Conversion strategy:

- Expose all Parallelism.
- Develop Vectorization strategy.
- Develop Memory, temporaries, MPI strategy.

Also:

- Maintain bit-for-bit against reference Fortran with a set of compiler flags and code configuration.
  - Cuda: `-fmad=false`
  - GCC: `-ffp-contract=off`
  - Intel: `-fp-model strict`

- Unit test everything.
  - `catch2`
Outline

1. Machines

2. Methods

3. Mini-app
To work through PVM for SCREAM, we made a mini-app implementing P3 rain sedimentation.

- Many intermediate versions.
- Final repo `master` has just a few.
- Docs to explain pieces.
P3 mini-app performance

- **PVM:**
  - Expose all Parallelism
  - Vectorize on Skylake and KNL
  - Memory and temporaries

- Performance and programmer convenience. Roughly,
  - Kokkos provides these for P.
  - Packs provide these for V.
  - Workspace provides these for M.

- **Initial (performant) Kokkos: P**
  - Also assure that V and M constructs don’t decrease performance.

- **Final Kokkos: PVM**

- **Comments:**
  - On KNL, mostly V gives Final Kokkos a 2.9–3.4× speedup over the Fortran reference.
  - On SKX, mostly V gives a 1.4–1.5× speedup over the Fortran reference.
  - \( n_e = 1024 \) on full Summit: \( \sim 2048 \) columns/GPU.
Next steps

SHOC has two algorithmic pieces plus a lot of code that has the same patterns as P3. These two algorithmic pieces will likely be of use to others:

- **Tridiagonal solve**
  - Diagonally dominant $\Rightarrow$ no pivoting (great).
  - Two systems, one with 2 RHS, one with 3 + num_tracer. (But #RHS still much $<$ than number of threads in team: at most $\sim 43$ vs 128.)
  - GPU: Combinations of two cyclic reduction variants and Thomas algorithm (standard elimination): thread across rows and RHS.
  - Non-GPU: Thomas algorithm, ideally with tracers packed along i (not k).

- **Linear interpolation**
  - Many applications with the same grids $\Rightarrow$
    - Set up, probably $O(n \log n)$ and fully parallel.
    - Application, probably $O(n)$ and fully parallel.