Performance Comparison of the Multi-zone Scalar Pentadiagonal (SP-MZ) NAS Parallel Benchmark on Many-core Parallel Platforms

Christopher Stone[1] and Bracy Elton[2]

[1] *Computational Science & Engineering, LLC, Chicago, IL 60622, U.S.A.*

[2] *Engility Corporation at the U.S. Air Force Research Laboratory, Wright-Patterson Air Force Base, OH 45433, U.S.A.*

SP-MZ NAS Benchmark

- **NAS Parallel Benchmark Scalar-Pentadiagonal Multi-Zone (SP-MZ) v3.3.1**
  - Multi-zone structured mesh with Pulliam-Chaussee ADI scheme
  - Variant of common ADI scheme: five linearly independent scalar pentadiagonal matrices in each sweep … implicit line solver
  - Solves steady-state PDE system with five components on uniform mesh using 2nd-order central with 4th-order dissipation
- **Mimics major CFD codes used for aerospace research:**
  - OVERFLOW-2 (NASA)
  - FDL3DI (AFRL)
- **A major component is the solution of the scalar pentadiagonal matrices in the X/Y/Z sweeps**
- **Uses variant of Thomas Algorithm (TMA) to solve the SP matrices**
  - TMA is inherently a sequential.
  - But, SP-MZ is line-implicit so each grid line can be computed concurrently.
    - `X_solve: Ny * Nz` lines
    - `Y_solve: Nx * Nz` lines
    - `Z_solve: Nx * Ny` lines

- **Important details:**
  - Algorithm designed for cache-based HPC systems.
  - Uniform zone sizes with (nearly) cubic zones.
  - Zones can not be divided or merged to improve load balancing.
  - Zones are independent within each iteration.
  - Fixed number of iterations across all zones.
- **Parallelism:**
  - MPI across zones (coarse-grained)
  - OpenMP threading within zones (fine-grained)

CLASS	Number of Zones	Zone Sizes (X x Y x Z)	Points per Zone	Total grid size (X x Y x Z)	Total Points
A	16 (4x4)	32x32x16	16k	128x128x16	256k
C	256 (16x16)	30x20x28	16k	480x320x28	4.3m

```
For All Iterations:
  Exchange     … [MPI + BCs]
  For All ZonesPerTask:
    ADI
      RHS      … [Finite Diff]
      Txinvr   … [SpMV]
      Xsolve   … [Line Solver]
      Ninvr    … [SpMV]
      Ysolve   … [Line Solver]
      Pinvr    … [SpMV]
      Zsolve   … [Line Solver]
      Tzetar   … [SpMV]
      Add      … [Vector Add]
```

Accelerating with Directives

The focus of this project is to address the following questions: (i) Can we successfully accelerate SP-MZ-like CFD codes on many-core accelerators such as Kepler GPU's and Xeon Phi's? (ii) Can we avoid coding directly in low-level, platform-specific languages such as CUDA? And (iii), what performance can be achieve?

OpenACC and OpenMP directives are portable (platform-independent) and are generally easier to implement than languages such as CUDA. And both support Fortran, the language of most legacy CFD applications. We explore using OpenACC for NVIDIA Kepler GPUs and OpenMP (with MPI) for the Intel Xeon Phi accelerators.

Generic Concept	CUDA on GPU	OpenMP on Xeon Phi
Coarse-grained, asynchronous tasks across accelerator cores	ThreadBlock	Thread
Fine-grained, synchronous SIMD operations within core	Thread	SIMD: vector operations within each thread

OpenACC

```
!$ACC PARALLEL LOOP GANG COLLAPSE(2)
!$ACC+   VECTOR_LENGTH(32)
      do k = 1, Nz
        do j = 1, Ny
          do i = 1, Nx
```

- **Outer `jk` iterations collapsed into a single iteration space and mapped to OpenACC gangs (CUDA ThreadBlock).**
  - Number of OpenACC gangs determined at run-time: `Ny` x `Nz`
- **Inner `i` iterations implicitly vectorized.**
  - Vector width set to 32 to match CUDA warp size.
  - Vector iterations mapped to GPU Threads on CUDA GPUs.

OpenMP

```
!$OMP PARALLEL DO COLLAPSE(2)
      do k = 1, Nz
        do j = 1, Ny
!$OMP     SIMD SAFELEN(16)
          do i = 1, Nx
```

- **Outer `jk` iterations collapsed into a single iteration space and mapped to OpenMP threads.**
  - Number of OpenMP threads determined specified by user (generally 1 per core).
- **Inner `i` iterations explicitly vectorized.**
  - SIMD vector width can be ≤ 16.

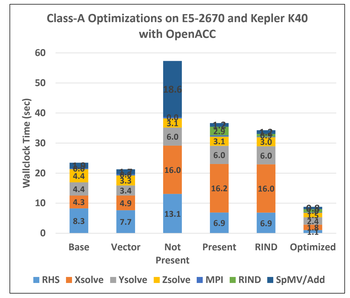
Single GPU OpenACC Optimization

Offloading with OpenACC is straightforward but still requires optimization. To the right is a sequence of optimizations with OpenACC.

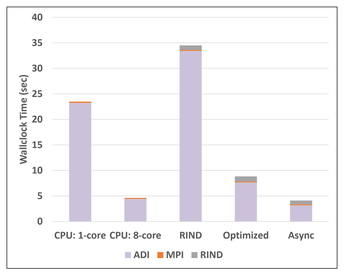
- Base: baseline SP-MZ on E5-2670 CPU core
- Vector: transposed array syntax from `U(:,I,J,K)` to `U(I,J,K,:)` on CPU core
- Not Present: H-D data transfer each routine
- Present: data persists across iterations
- RIND: only zone boundaries transferred
- Optimized: each routine fully optimized

The optimizations were:

- Restructuring loops to improve thread vectorization.
- Collapsing outer loops to increase number of thread blocks.
- Reordering array syntax to insure unit-stride access.



OpenACC optimizations improved Class-A performance by 75% (from RIND).



Each SP-MZ zone is small. Running many zones in parallel improves performance on a single device. Launch all zones asynchronously using OpenACC stream doubles the performance over the full optimized implementation.

OpenACC SP-MZ with asynchronous kernels outperforms 8-core Sandy Bridge CPU by 12%.

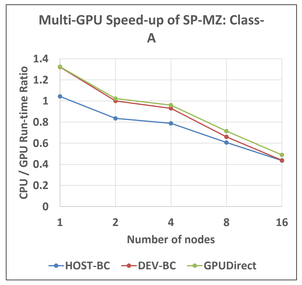
Multiple GPU Optimization

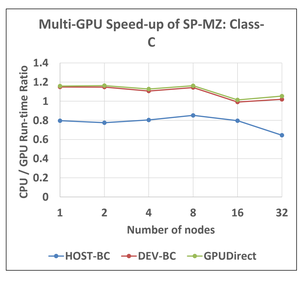
Major feature of the SP-MZ code is the ability to run across many compute nodes. We performed scaling studies across 32 nodes on the Shepard supercomputer at the Navy DoD Supercomputing Resource Center (DSRC). Each node has 1 E5-2670v2 CPU (10-core) and 1 K40c Kepler GPU. Prior OpenACC implementation set all boundary conditions (RIND) on the host requiring H-D communication for all zones each iteration (HOST-BC). Two methods tested in multi-GPU configuration:

- **DEV-BC: neighbor zones on device directly copy data instead of transferring to host. MPI used on host to communication remote zones.**
- **GPUDirect: local zones copy RIND and remote zones are fetched directly from remote device via direct GPU-GPU path.**

DEV-BC increases performance by 45%.

GPUDirect adds extra 11% boost for larger number of nodes.



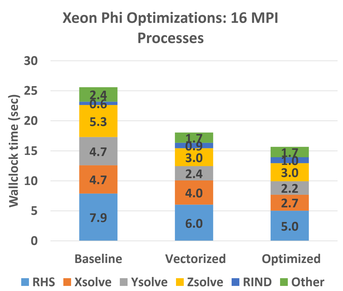


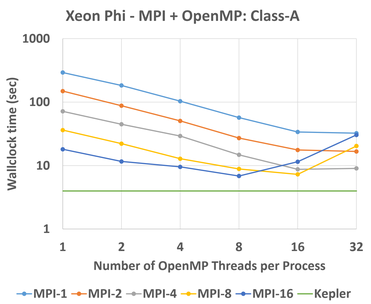
MIC Optimizations

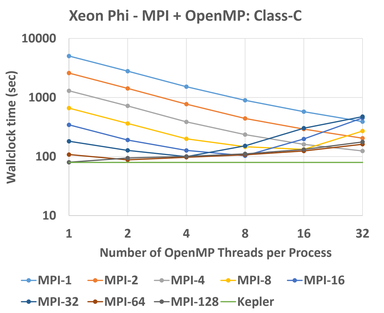
Intel Xeon Phi can use existing MPI + OpenMP CFD code in *native* mode. Combine MPI (coarse), OpenMP (medium), and inner-loop vectorization (fine parallelism) for optimal performance.

Single-device benchmark shows Xeon Phi with "vectorized" (structure-of-arrays) SP-MZ performs close to fully optimized Kepler OpenACC code on Class-C using MPI only. Class-A works best with 8 threads per zone and 16 MPI tasks (64 total) to use all the available cores.

Similar optimizations need for Xeon Phi as GPU: unit stride indexing and loop reordering / fusing.





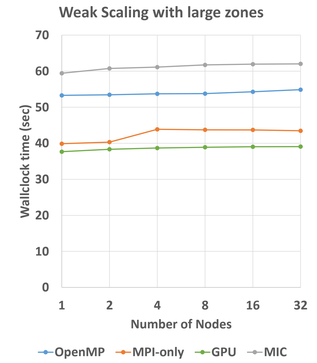


Head-to-head Performance Comparison

SP-MZ zone sizes are smaller than those commonly used on modern HPC platforms. Created new benchmark that is more realistic:

- 1 million points per zone (100^3).
- Weak scaling study with new zone sizes conducted on the Navy DoD Supercomputing Resource Center Shepard system.
- Compared hybrid MPI+OpenMP (1 100^3 zone, 1 MPI task; 10 OMP threads) and pure MPI (10 MPI tasks with 1 50x20x100 zone each) on the host to MPI+OpenACC (1 100^3 zone per GPU) and MPI+OpenMP on the MIC (1 MPI task and 100 OMP threads per device).

MPI+OpenACC 29% faster than MPI+OpenMP on host and is faster than pure MPI on host even while ignoring the slower convergence of pure MPI. MIC performs 11% slower than host.



Future Directions

SP-MZ shows promise on accelerators using directives. Several aspects need to be explored further in future work:

- Standard SP-MZ uses only 2-d zone partitioning: explore 3-d zone distributions.
- SP-MZ algorithm requires on 1 RIND layer to be exchanged. High-order CFD applications need 2 or 3 layers which adds communication costs: explore communication (MPI and host-device) impact of thicker RIND layers.
- Xeon Phi implementation uses native mode with MPI + OpenMP: explore offloading directives to enable host and accelerator symmetric (hybrid) computing.
- Xeon Phi implementation uses only outer loop threading: explore tiling or blocking of inner loops to increase thread parallelism and cache data locality.

Acknowledgements

This study was supported by the United States Dept. of Defense High Performance Computing Modernization Program (HPCMP) User Productivity Enhancement, Technology Transfer, and Training (PETTT) activity (GSA Contract No. GS04T09DBC0017 through High Performance Technologies, Inc.).