Optimizing Turbomachinery CFD applications for Modern Multi-core and Accelerator HPC Systems

Christopher P. Stone[1], Daryl Y. Lee[2], and Roger L. Davis[2]

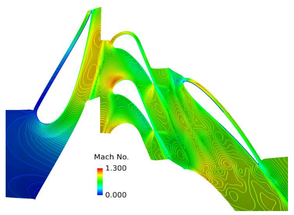
[1] *Computational Science & Engineering, LLC, Chicago, IL, U.S.A.*

[2] *Dept. of Aerospace and Mechanical Engineering, U. of California Davis, Davis CA, U.S.A.*

MBFLO3

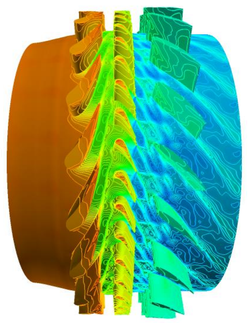
MBFLO3 is a general-purpose, multi-disciplinary conjugate heat transfer solution procedure with primary applications in turbomachinery. It's numerical features include:

- **Multi-block, structured grids with fine-grain decomposition for massive parallelization**
- **Control-volume based numerical method**
- **Explicit, multi-grid for steady flow**
- **Explicit time-marching or point-implicit dual-time for unsteady flow**
- **Sliding-grid method for inter-blade-row treatment for unsteady turbomachinery simulations**
- **Turbulence models include Wilcox k-ω (RANS) and detached-eddy simulation (DES)**



Mach number contours through 2d slice of HIT turbine vane model simulation

Temperature contours through HIT turbine vane model



Transitioning to Hybrid HPC Architectures

The focus of this project is to investigate the performance of a turbomachinery simulation code (MBFLO3) using hybrid MPI + X programming methods. Transitioning MBFLO3 to modern HPC platforms requires refactoring the code to exploit multiple levels of parallelism on all platforms:

Coarse-grain: block parallel domain decomposition via MPI (processes)

Medium-grain: loop parallelism via OpenMP (threads) / OpenACC (gangs) directives

Fine-grain: data parallelism for inner-most loop iterations via OpenMP (SIMD) / OpenACC (vector) directives

OpenMP

```
!$OMP PARALLEL DO COLLAPSE(2)
      DO K = 1, KMAX
        DO J = 1, JMAX
!$OMP      SIMD SAFELEN(16)
!$OMP+          ALIGNED(X:32)
           DO I = 1, IMAX
             X(I,J,K) = …
```

- **Outer `JK` iterations collapsed into a single iteration space and mapped to OpenMP threads.**
  - Number of OpenMP threads determined specified by user
  - 1 per-core on host CPU; 2+ per-core on Xeon Phi.
- **Inner `I` iterations explicitly vectorized.**
  - SIMD vector width can be ≤ 16.
  - Array `X` is aligned to 32-byte boundary.

OpenACC

```
!$ACC PARALLEL LOOP GANG
!$ACC+    NUM_WORKERS(4)
!$ACC+    VECTOR_LENGTH(64)
      DO K = 1, KMAX
        DO J = 1, JMAX
          DO I = 1, IMAX
```

- **Outer `K` iterations mapped to OpenACC `gangs` (CUDA `ThreadBlocks`).**
- **Middle `J` iterations mapped to OpenACC `workers` (CUDA `warps`)**
- **Inner `I` iterations implicitly vectorized across two workers.**
  - Equivalent to vector width of 32.
  - Vector iterations mapped to GPU Threads on CUDA GPUs.

Host SIMD Optimizations

The MBFLO3 code was originally programmed in Fortran90 using a traditional, cache-optimized data layout where the dependent variables were the left-most (inner-most) index to take advantage of data locality.

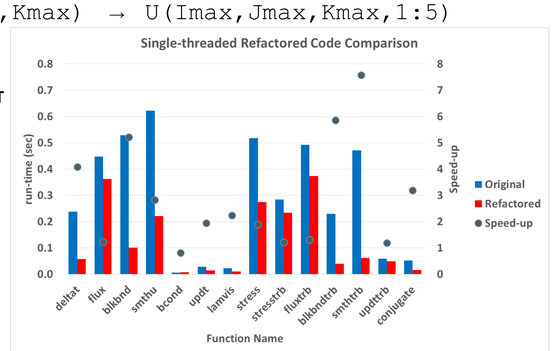
The first effort in the project has been to refactor the MBFLO3 code using Fortran data-types in which the flow/heat transfer variable is now the right-most index to take advantage of vector processing such as on accelerator processors. This effort has been followed by incorporation of OpenMP data parallel SIMD directives to improve single-core performance on AVX / AVX2 CPU cores.

`U(1:5,Imax,Jmax,Kmax)` → `U(Imax,Jmax,Kmax,1:5)`

Average run-time (right) of major MBFLO3 functions for 40 iterations of the steady-state HIT turbine vane model with 16 blocks, 1.6 million point mesh.

Run on one Intel E5-2697 (Haswell) CPU core on *Thunder* (SGI ICE-X) at the Air Force Research Laboratory (AFRL).

Showing speed-up due to SIMD data parallelism and memory optimizations over original implementation.



Overall speed-up is 2.2x over original data layout and loop design.

Certain computationally-intensive functions improving by more than 7x.

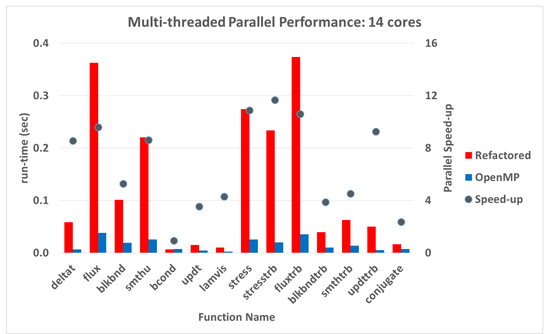
Host OpenMP Optimizations

The second effort is to include loop-level thread parallelism. This increases the available parallelism due to limits with block-level domain decomposition (MPI-only).

Average run-time (right) of major MBFLO3 functions for 40 iterations of the steady-state HIT turbine vane model with 16 blocks, 1.6 million point mesh.

Run on all 14 cores of an Intel E5-2697 (Haswell) CPU on *Thunder* (SGI ICE-X) at the Air Force Research Laboratory (AFRL).

Showing parallel speed-up over single-threaded, refactored MBFLO3 code.



Overall multi-core parallel efficiency is 59% on 14 cores.

Thread parallel performance hindered by data-intensive inter-block communication routines that are difficult to parallelize.

Most computationally-intensive functions attaining ≥ 70% efficiency.

MIC Accelerator Performance

The data layout refactoring and OpenMP thread and data parallel optimizations were tested on an Intel Xeon Phi in native mode to assess their impact on the many-core environment. The Xeon Phi is heavily dependent upon SIMD vector parallelism and 100's of concurrent threads.

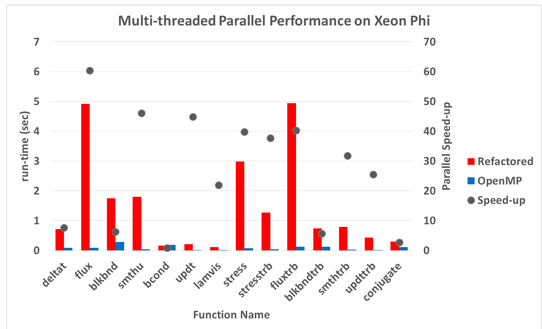
Data layout and SIMD optimizations gave 4.3x speed-up over original MBFLO3 implementation on a single Xeon Phi core. This is twice the speed-up on the Haswell CPU core and matches the SIMD width differences.

Overall parallel efficiency only 28% (17x speed-up) on 60 Phi cores relative to single-core Phi run.

Performance limited by functions with global reduction operation (`deltaT`) and block-interface routines (`blkbnd*`) with irregular data movement.

Significant improvement needed on these routines in order to compete with host CPU.

Computationally-intensive routines performing well on massively parallel device.



Average run-time (right) of major MBFLO3 functions for 40 iterations of the steady-state HIT turbine vane model with 16 blocks, 1.6 million point mesh. Run on an Intel Xeon Phi (5120D) in native mode with 60 cores using 120 OpenMP threads on *Shepard* (Cray XC-30) at the Navy DoD Supercomputing Resource Center.

OpenACC Performance

Two routines have been ported to OpenACC and tested on an NVIDIA Kepler K40c GPU on *Shepard* (Cray XC-30) at the Navy DoD Supercomputing Resource Center (NAVO) using the PGI 15.7 compiler. The table below compares the run-time of the functions executed on the GPU compared to 14 host cores using OpenMP on Thunder.

GPU times are divided into *total* (transfer + computation) and *kernel* (computation-only). Current implementation requires all variables to be transferred. Final implementation will avoid most transfers and will approach the kernel time.

Time-step (`deltaT`) computation requires global min/max reduction which is costly on the GPU. Kernel is 43% slower than host-only OpenMP.

Viscosity (`lamvis`) computation is computationally-intensive and requires no reduction operations. Kernels is 30% faster than host-only OpenMP.

Function	OpenMP: 14 cores	OpenACC: Kepler K40c
`deltaT`	`6.8` ms	`73.3` ms (total) `9.7` ms (kernel-only)
`lamvis`	`2.4` ms	`37.8` ms (total) `1.7` ms (kernel-only)

Future Directions

Future focus will include:

- **Further optimization of shared-memory parallelization on multi-cores, GPUs, and Intel Phi's (reduction operations and irregular block interface routines).**
- **Load balancing all available compute devices (multi-core CPUs, GPUs, and/or Intel Xeon Phi's) by adapting thread parallelism per mesh block or per routine.**
- **Comparison of Kokkos C++ platform-independent implementation performance to manual code refactoring.**
- **Further validation of overall procedure.**

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.).