Concurrent parallel processing on Graphics and Multicore Processors with OpenACC and OpenMP

Christopher P. Stone, DoD HPCMP PETTT Program

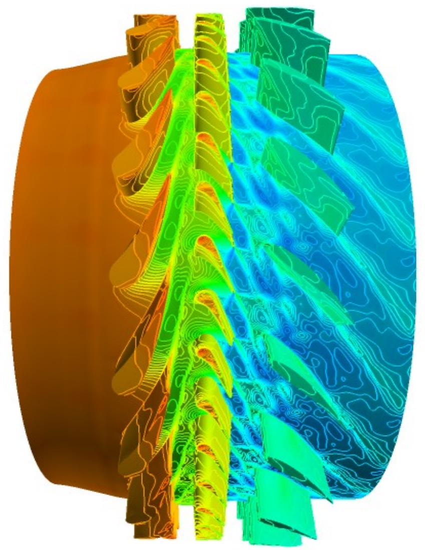
Prof. Roger L. Davis and Daryl Lee, U. of California Davis

Acknowledgments

- This material is based upon work supported by, or in part by, the Department of Defense High Performance Computing Modernization Program (HPCMP) under User Productivity, Technology Transfer and Training (PETTT) contract number GS04T09DBC0017.

Background

- **MBFLO3 is a general-purpose, three-dimensional, multi-block structured-grid, finite-volume, multidisciplinary solution procedure**
  - Primary application of MBFLO3 is modeling turbomachinery (e.g., compressors and turbines) for aerospace.
  - Explicit Lax-Wendroff control-volume, time-marching scheme
  - Multi-block structured-grids with both point-matched and overset interface capabilities
  - Multigrid acceleration for steady flows; point-implicit dual time-step for unsteady flows.
  - Conjugate heat transfer modeling via coupled fluid-thermal solvers.



Motivation

- **MBFLO3 originally used only MPI for parallelism across CPU cores.**
  - One MPI rank per mesh block.
  - Blocks partitioned with STAGE3 preprocessor / grid-generator.
  - Arbitrary mesh block subdivision not available for point-matched, multi-block grids.
- **Fixed mesh partitioning limits strong scaling, especially as we transition to many-core devices.**
  - On-chip (cache) memory per-core has been stagnant for multi-core devices and memory per-core on newer many-core devices is lower.

Goals

- **Redesign / refactor MBFLO3 to harness**
  - single instruction multiple data (SIMD) vectorization … i.e., improve fine-grain data parallelism.
  - device-level parallel … i.e., implement coarse-grain thread / task parallelism.
- **Implement data and thread parallelism in platform-independent manner so as to apply to variety of heterogeneous multi-core and many-devices.**
  - That is, support multi-core CPUs and many-core devices such as GPUs and Intel Xeon Phi (KNC / KNL).
  - Use standards-based approaches, OpenMP and OpenACC, to provide platform independence for Fortran90 code base.

MBFLO3 Refactoring

- **Original MBFLO3 used a cache-friendly M/I/J/K indexing scheme.** `U(5,Imax,Jmax,Kmax)`
  - This is suitable for scalar CPUs with strong cache dependencies.
  - CPUs are becoming hybrid scalar-vector processors so we needed to reorder arrays to promote inner loop vectorization.
  - Accelerators (MICs and GPUs) rely on vectorization already: effective vector widths of 8-32 words per operation (e.g., AVX-512).
- **Transposed indices and created BLK data structure at the same time for most MBFLO3 routines.** `BLK(n)%U(Imax,Jmax,Kmax,5)`
  - Improved performance largely due to improved vectorization facilitated by vector-friendly layout.

MBFLO3 Refactoring

- **Added OpenMP thread parallelism to the outer J/K-loops for multi-core CPUs in all computationally-intensive MBFLO3 routines.**
  - Nested I/J/K loops collapsed to increase the thread parallelism using OpenMP collapse feature.
- **Added OpenMP SIMD directives (where needed) to the inner i-loop for CPUs and MIC accelerators.**
  - Inner loop vectorization necessary for all accelerators and CPUs.
  - Works in concert with J/K outer loop collapsing.

MBFLO3 Refactoring

- **Added OpenACC gang parallelism to the outer J/K-loops in all computationally-intensive MBFLO3 routines.**
  - Nested I/J/K loops collapsed to increase the gang parallelism using OpenACC collapse feature.
- **Added OpenACC vector loop directive to the inner i-loop for GPU accelerators.**
  - Inner loop vectorization necessary for all accelerators and CPUs.
  - Works in concert with J/K outer loop collapsing.
- **Added OpenACC ENTER/EXIT DATA and UPDATE directives to manage data on host / device**
- **Added OpenACC WAIT directive to ensure synchronization.**

Thread / Gang / Task Parallelism

- **Adding OpenMP / OpenACC thread parallelism to the outer `j`/`k`-loops for multi-core CPUs and many-core devices.**

```
!$omp parallel do collapse(2)
do k = 1,kmax
  do j = 1,jmax
    do i = 1,imax
      <ijk kernel>
```

Behaves like …

```
!$omp parallel do
do jk = 1,jmax*kmax
  k = jk / jmax
  j = jk - k*jmax
  do i = 1, imax
    <ijk kernel>
```

  - OpenMP/OpenACC will split up the outer (k) loop across the available cores: i.e., *worksharing* loop iterations.
  - If `kmax` < number of processing elements, then we waste the resources.
  - *collapse(2)* merges (or flattens) the j and k loops. Signals to OpenMP/OpenACC to split up all `Jmax*Kmax` iterations … but with some overhead.
  - No OpenMP equivalent for OpenACC tile(8,8). Useful for cache blocking.

Data (Vector) Parallelism

- **Add SIMD / VECTOR directives to the inner `i`-loop to promote / enforce vectorization along fastest array index.**
  - ~4-8x potential (64b precision) speed-up on SB, IVB, HW cores.
  - ~8-16x potential speed-up on AVX512 cores.
  - ~32-way data (thread) parallelism on CUDA GPUs.
  - Works in concert with OpenMP / OpenACC collapsed outer j/k loops.

```
!$omp parallel do collapse(2) private( xvel,… )
!$acc parallel loop gang collapse(2) vector_length(64)
!$acc&   async( block_idx ) pcopyin( x,u … ) pcopy( du, … )
      Do k = 1, kmax
         Do j = 1, jmax
!$omp       simd safelen(8)
!$acc       loop vector
            Do i = 1, imax
               xvel = u(i,j,k,2) / u(i,j,k,1)
               du(i,j,k,1) = du(i,j,k,1) + xvel**2
               ⋮
```

Task Parallelism

- **Small boundary condition and block-interface (BLKBND) routines cannot be efficiently threaded with loop-level parallelism.**
- **BLKBND routines:**
  - only touch surface mesh nodes (2d v. 3d … cheap)
  - vectorize poorly (non-unit stride and often use indirect addressing)
  - unstructured control loops (unknown loop counts).
  - different BC's take different amount of time (non-uniform workload)
- **Used OpenMP Tasks to parallelize complex, unstructured BC function calls.**
  - With sufficient # of tasks, automatically load balances across the threads.
  - Avoids serialized bottleneck and poor scaling on many-core devices such as KNL.

Task Parallelism

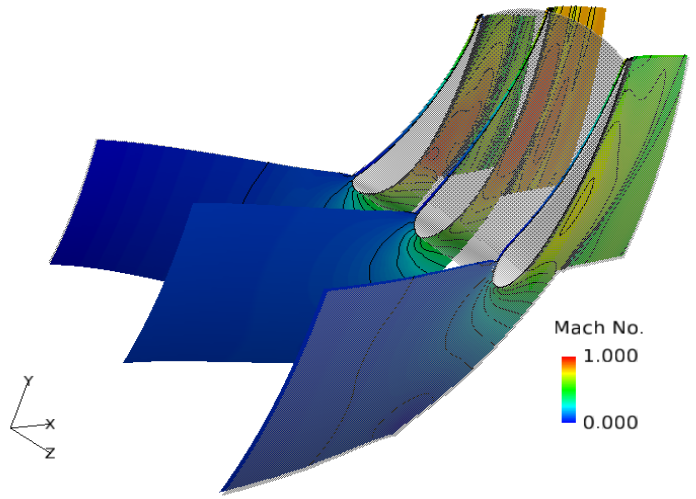
```
!$omp parallel private (n,blkid,f,p) shared (irecv,buf)
!$omp single
! -- Loop over all blocks, faces, and subface patches.
      Do n = 1, n_blocks
         blkid = block_list(n)
         Do f = 1, 6
            Do p = 1, n_face_patches(blkid, f)
               If ( has_remote_neighbor( blkid,f,p ) ) Then
! -- Enqueue task to unpack shared buffer filled by MPI_Recv.
                  irecv = irecv + 1
!$omp             task firstprivate( irecv,blkid,f,p )
                  call unpack_recv_buf( buf(irecv),blkid,f,p,… )
!$omp             end task
               Else
! -- Enqueue task to set conditions on a subface patch
!$omp             task firstprivate( blkid,f,p )
                  call set_phys_bc( blkid,f,p,… )
!$omp             end task
               Endif
            Enddo
         Enddo
      Enddo
!$omp end single nowait
!$omp end parallel
```

Heterogeneous Design

- **OpenMP is well suited for multi-core parallelism with multi-threading.**
- **OpenACC is well suited for many-core offloading.**
- **Ideally, blend the two to use both the multi-core host concurrently with the many-core accelerator.**
- **Both OpenMP and OpenACC can reside within the same source code but cannot compile both.**
- **Our approach is to compile twice, once with OpenACC and once with OpenMP enabled, and rename the functions for ACC or OMP.**
- **At run-time, mesh blocks are allocated to either the accelerator or the host and these use the appropriate computational routines.**
  - Host-Device transfers required to satisfy block-interface boundary conditions.
  - All block-interface data transfers occur on the host; no GPU-Direct used (at this time).

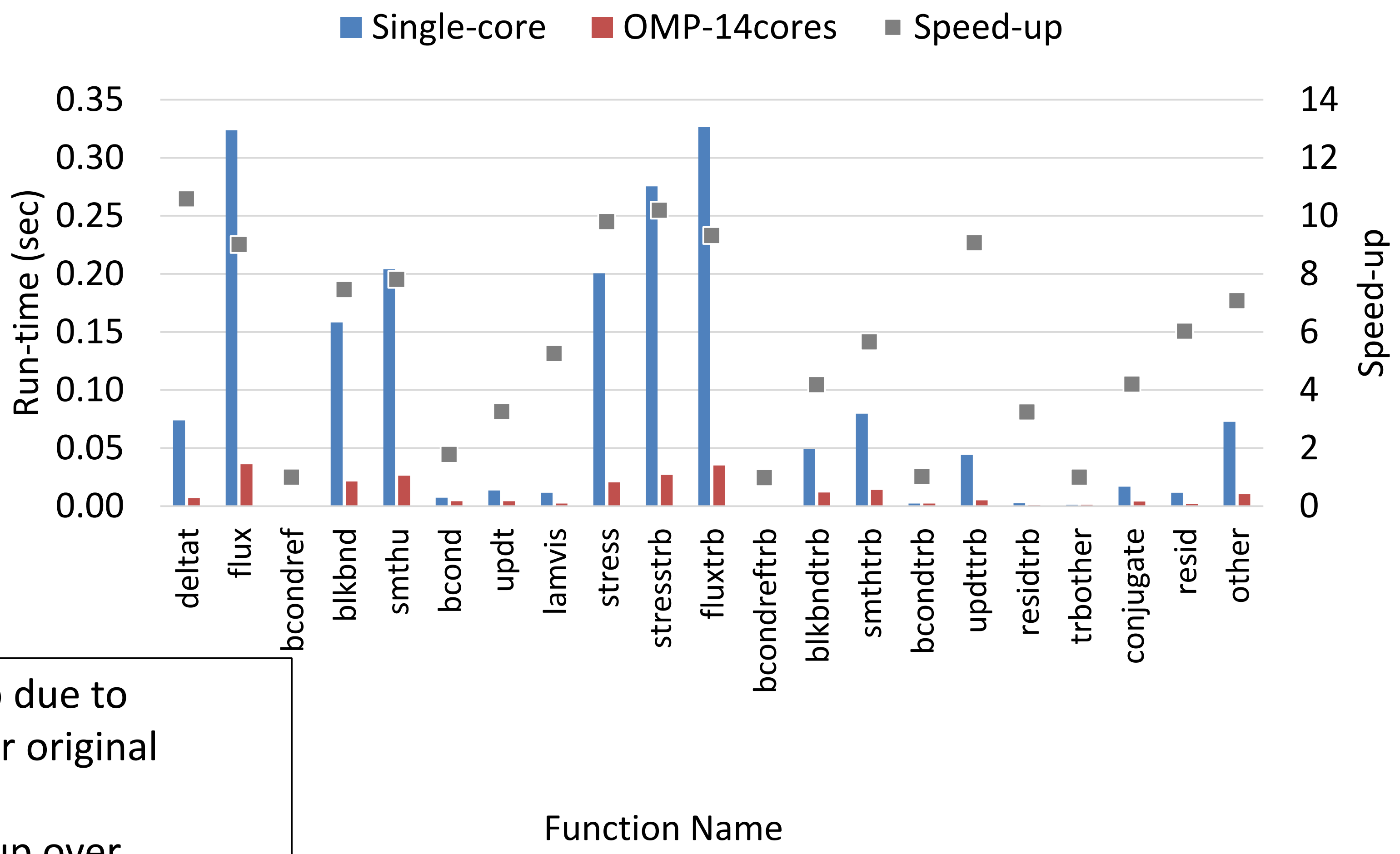
Benchmark Conditions / Platforms

- **Reporting average run-time for 60 iterations of the steady-state turbine vane model with 16 blocks and 1.6 million points (total).**



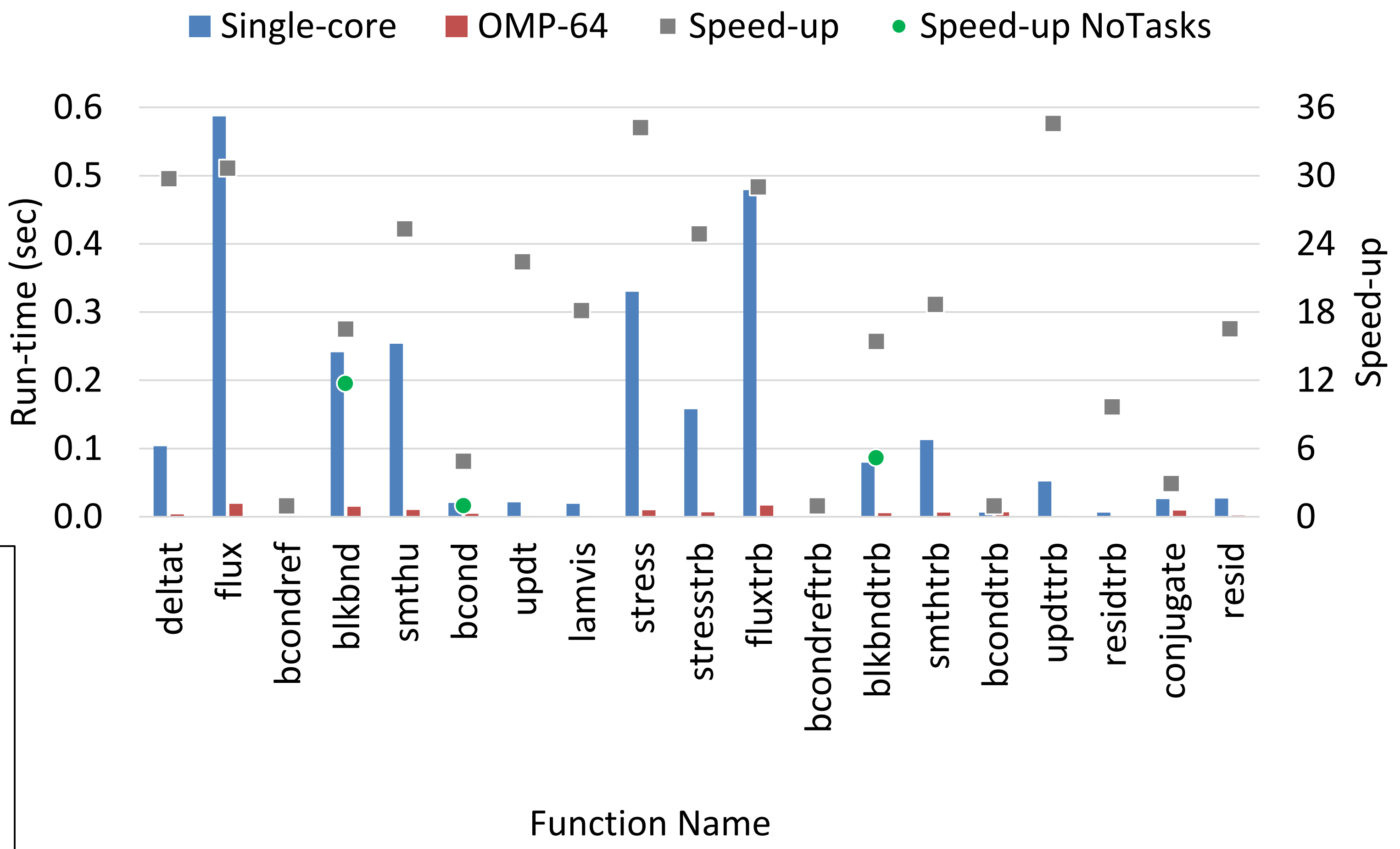
- **Benchmarks run on**
  - Thunder: (SGI ICE-X) at the Air Force Research Lab (AFRL) DSRC (Dayton, OH)
    - PGI 16.7 compiler; 2 Haswell (E5-2697v3) CPUs with 14 cores per CPU and 2 NVIDIA K40m GPUs
  - ARL-KNL Testbed: the Army Research Laboratory (ARL) DSRC (Aberdeen, MD)
    - Intel 17.2 compiler; 64-core Intel Xeon Phi 7230
  - Hokule'a: (IBM Power8 + 4 P100 GPUs) at the Maui HPC Center (MHPCC).
    - PGI 17.7 compiler; 2 IBM Power8 with 4 P100-SXM2 (16 GB)

Host (HSW) OpenMP Parallelism



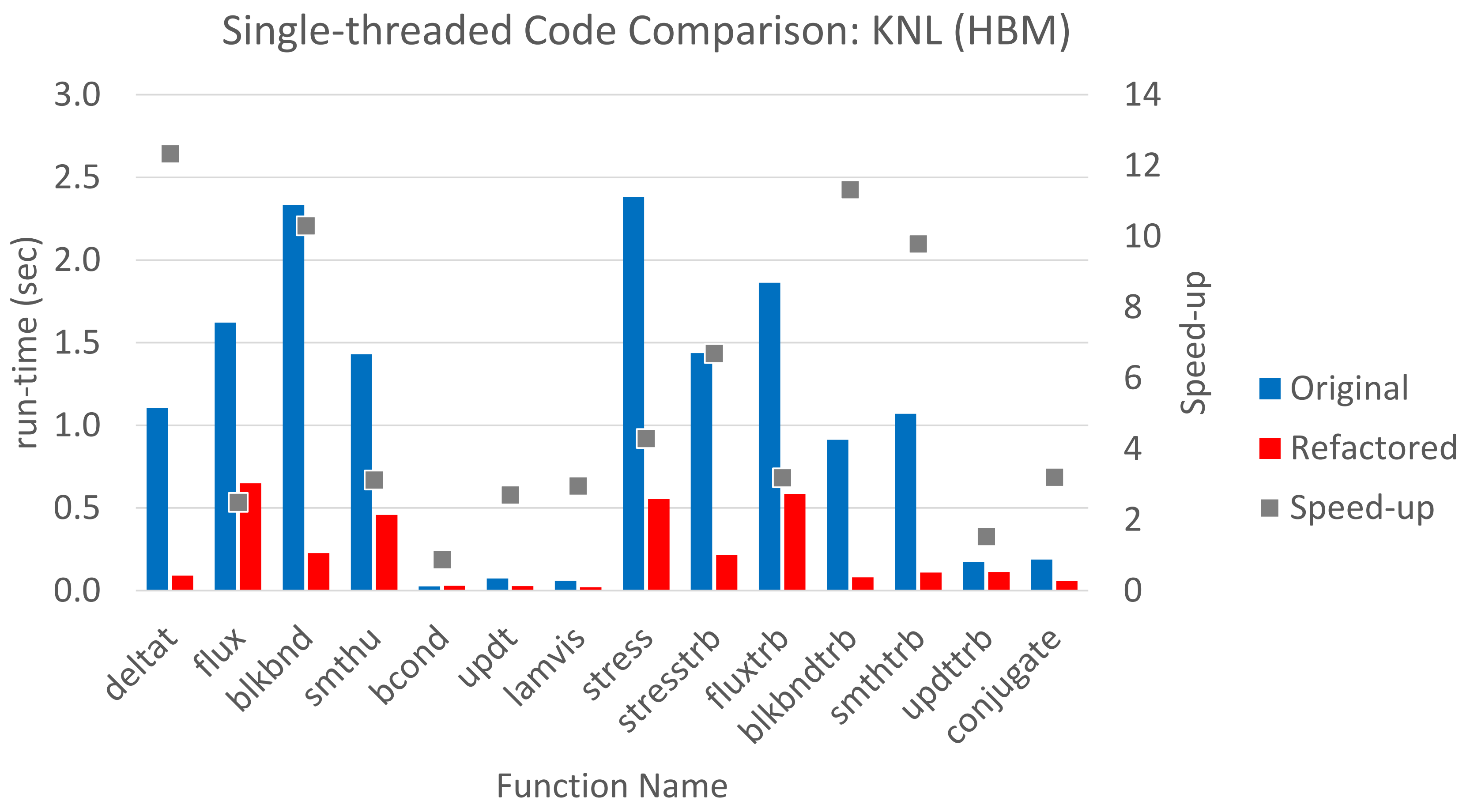
- 2.2x net speed-up due to vectorization (over original single-threaded)
- 8.5x OMP speed-up over refactored single-threaded. (61%)

KNL OpenMP Parallelism

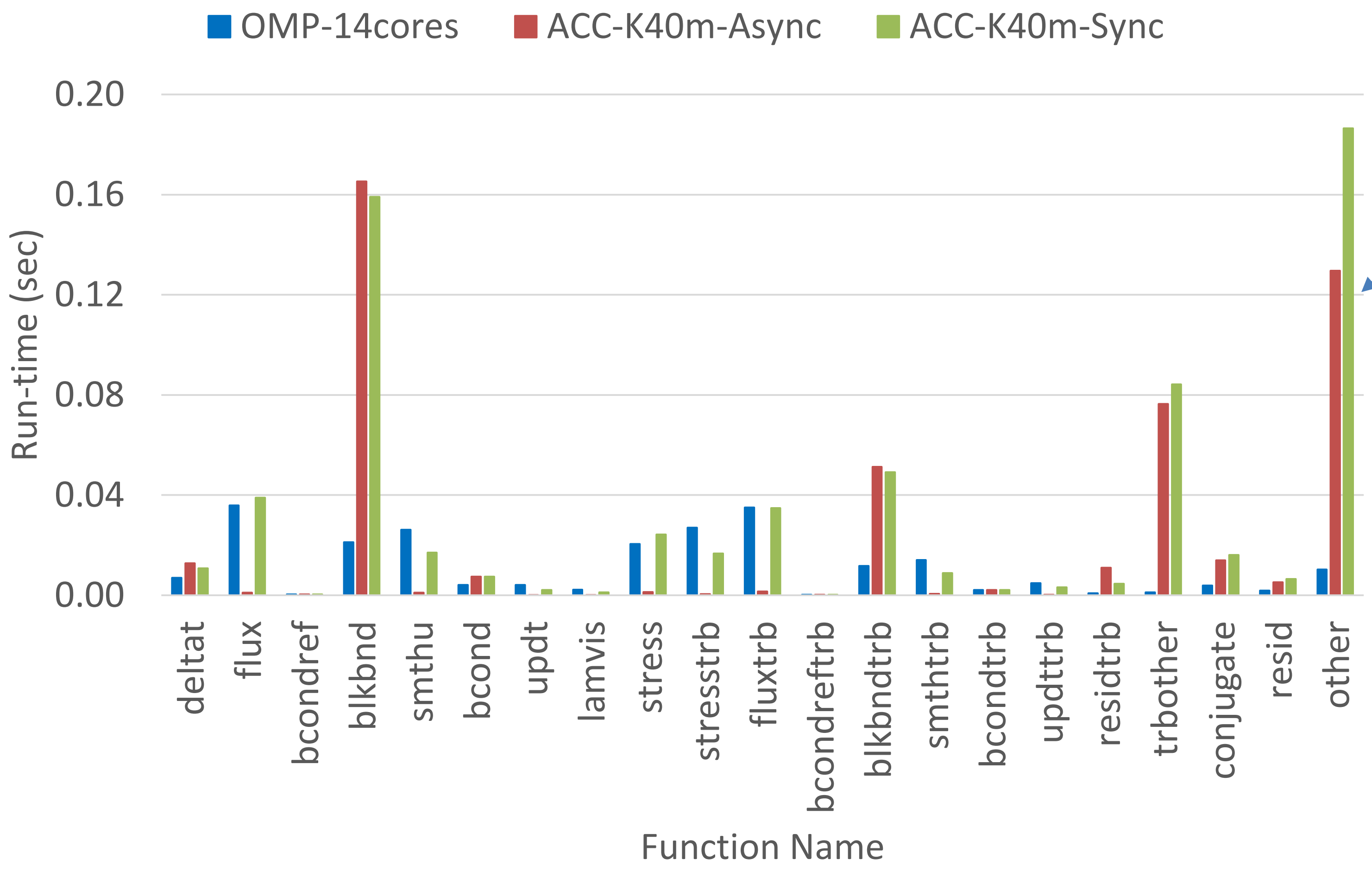


Task parallelism for BC routines improved net speed-up from 17x to 29x on KNL with 64 threads.

KNL Data Parallelism

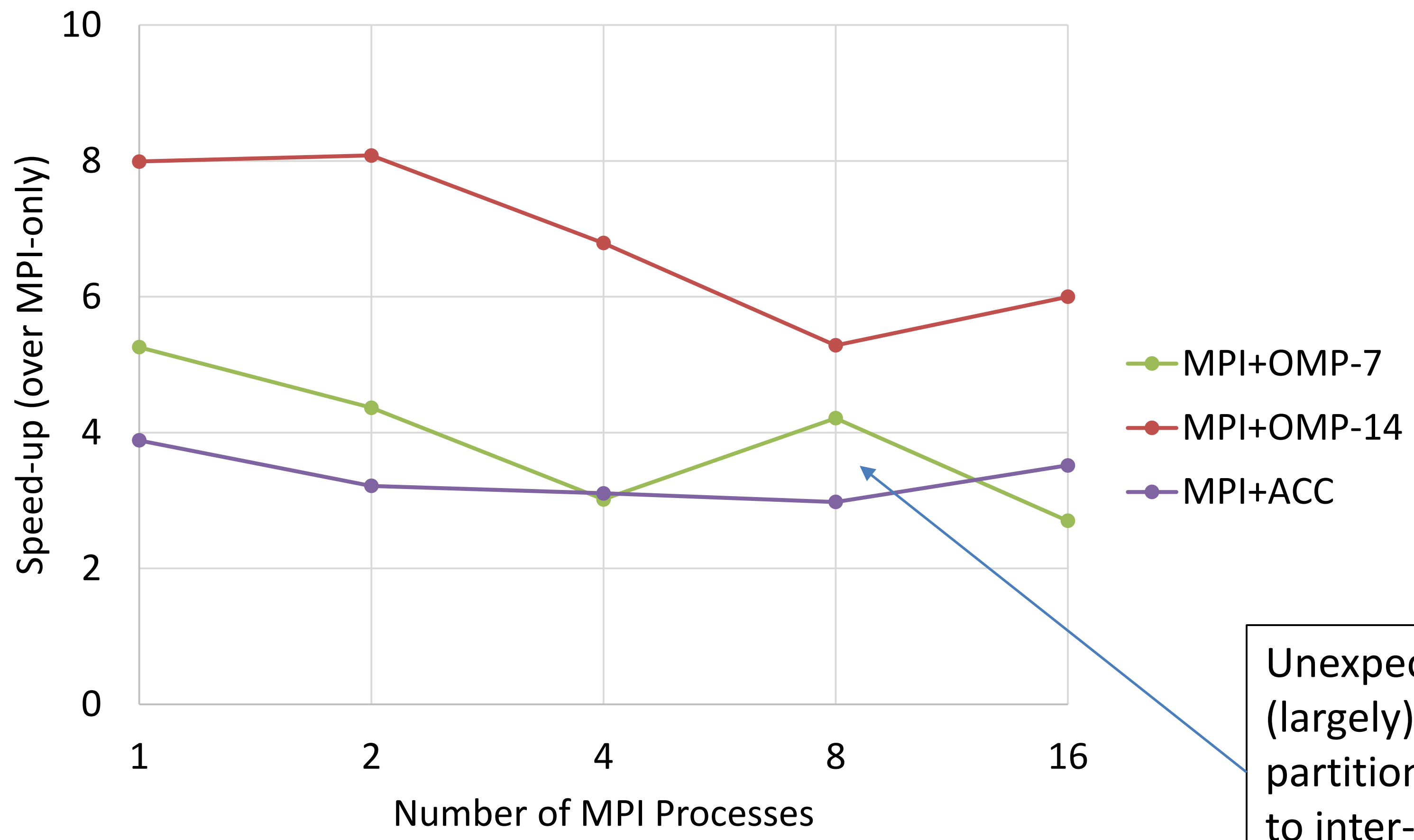


Offload GPU Parallelism



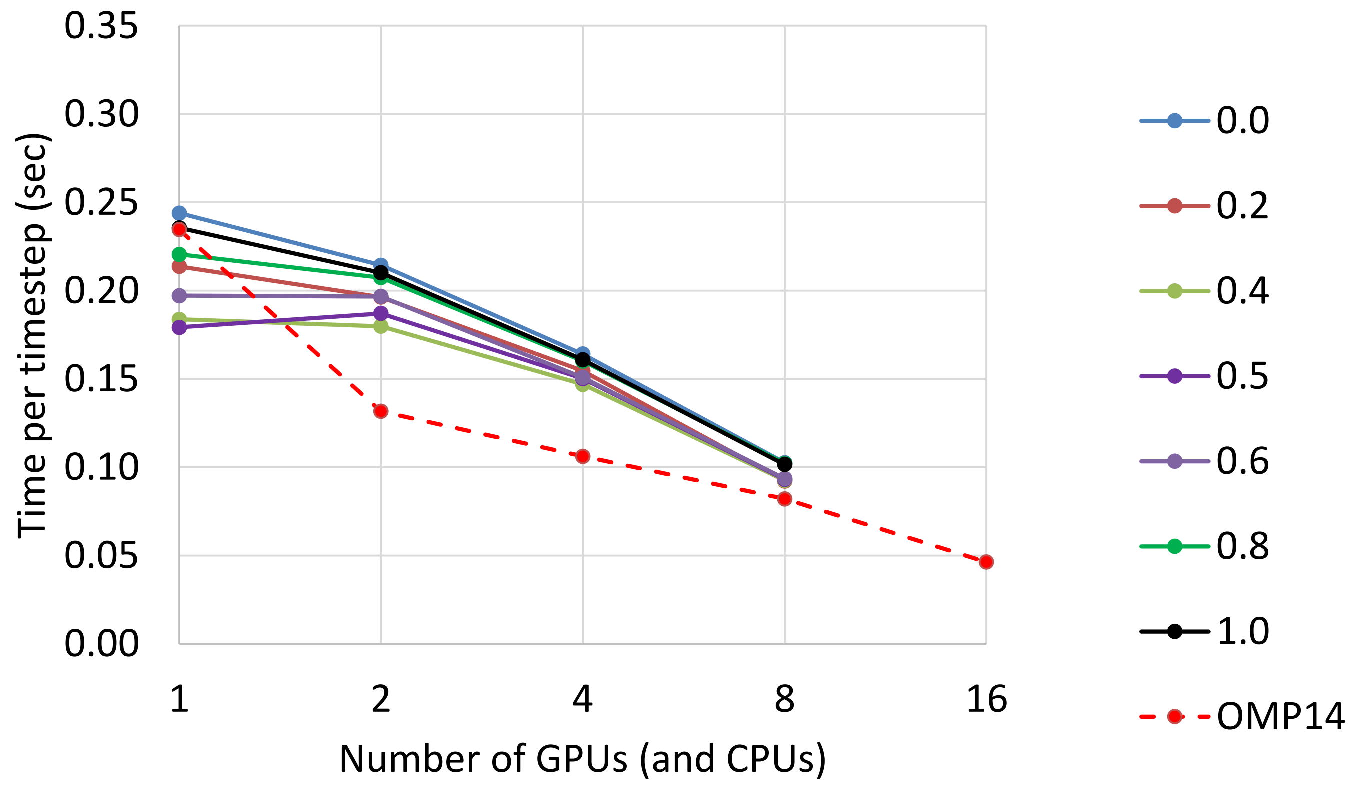
Host-Device Communication costs

Multi-device Performance Comparison: Intel Haswell + NVIDIA Kepler K40m

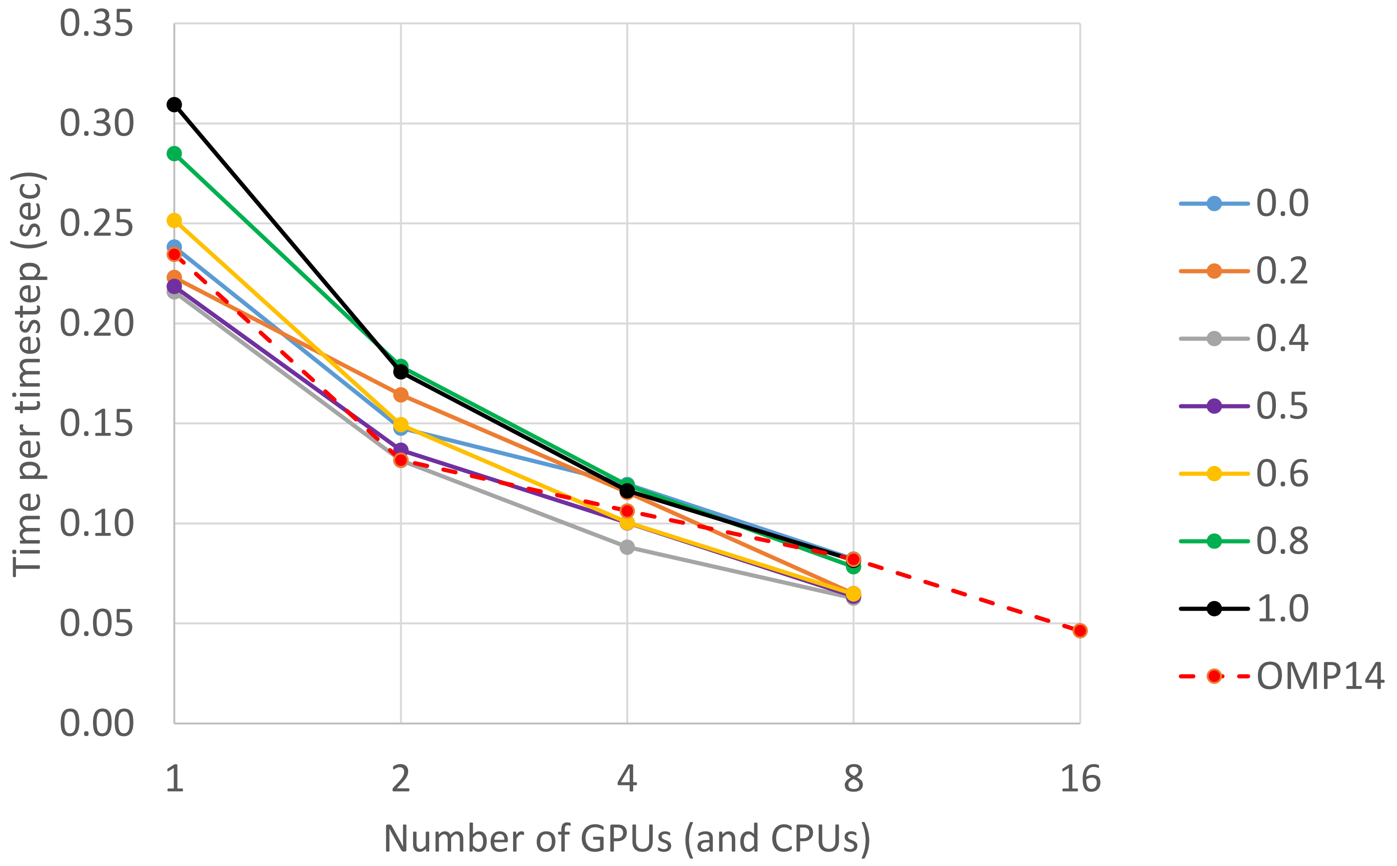


Unexpected behavior (largely) due to block-level partitioning changing intra- to inter-node communication

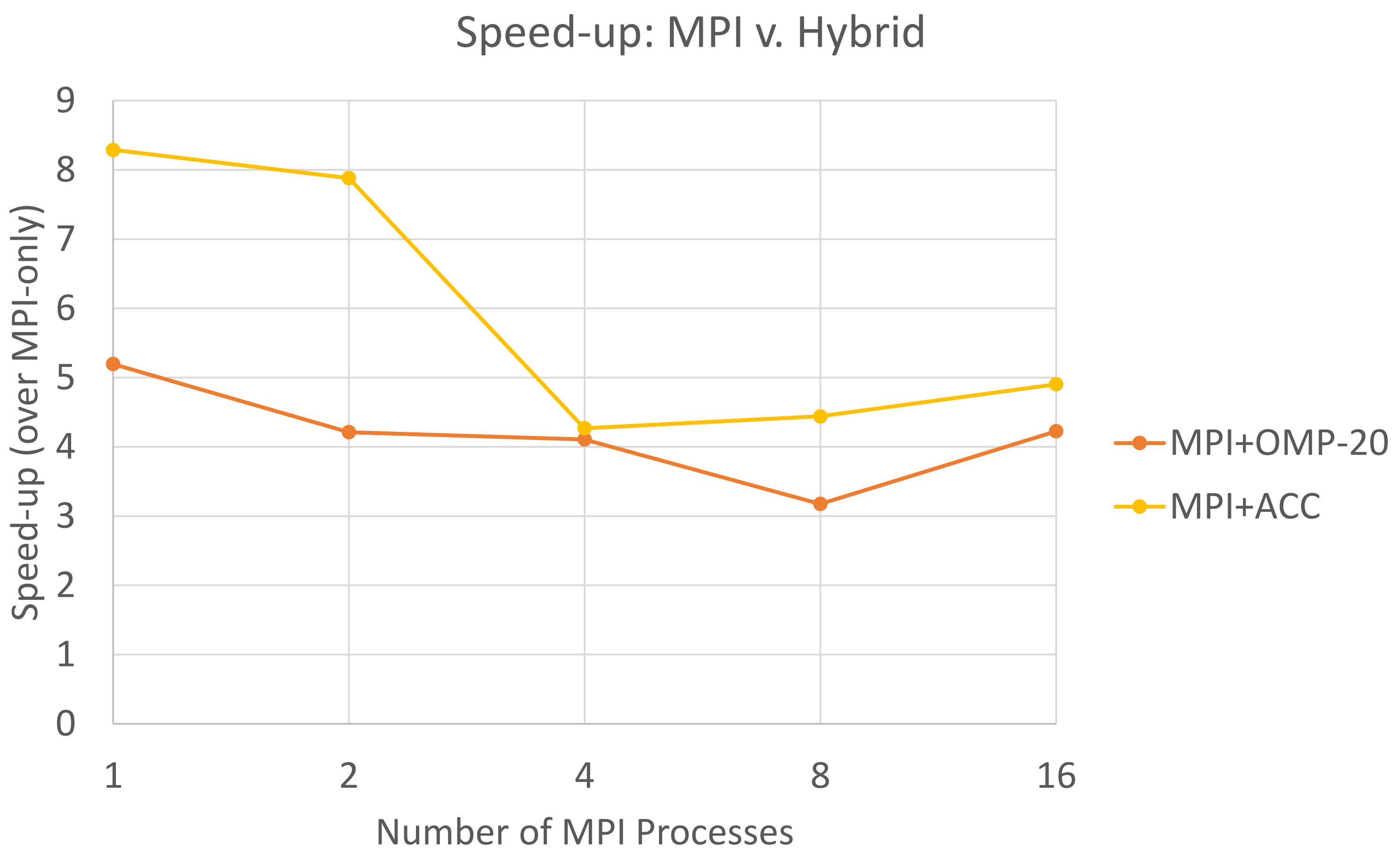
Heterogeneous CPU/GPU Parallelism: OMP-to-ACC workload ratio *with pinned memory*



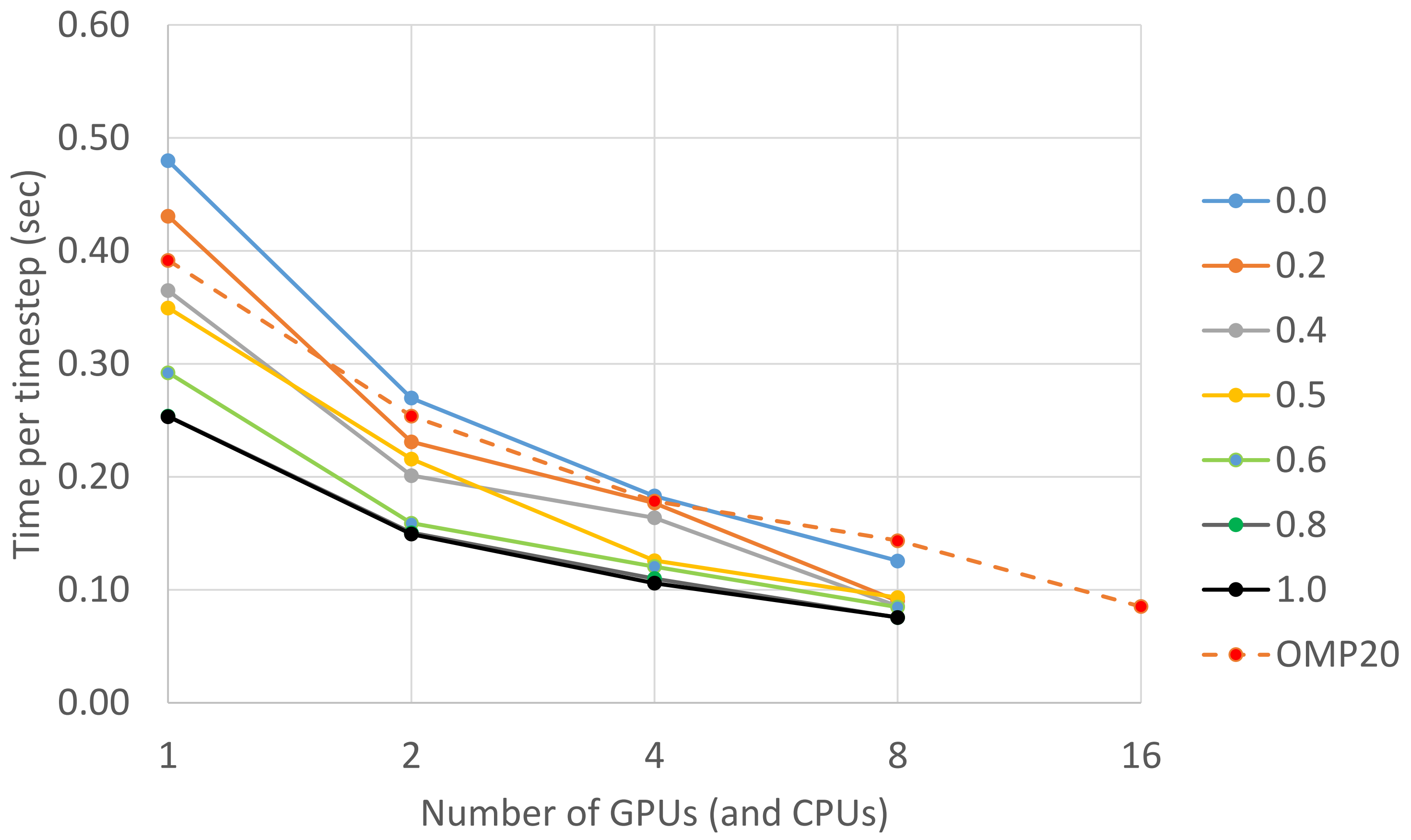
Heterogeneous CPU/GPU Parallelism: OMP-to-ACC workload ratio *without* *pinned memory*



Multi-device Performance Comparison: IBM Power8 + NVIDIA Pascal P100



Heterogeneous CPU/GPU Parallelism: OMP-to-ACC workload ratio on PWR8+P100



Conclusions

- **Refactoring legacy CFD applications requires 3-pronged approach:**
  - Coarse-grained multi-threading (tasks / threads / gangs)
  - Fine-grained data parallelism (SIMD / SIMT vectorization)
  - Data layout optimization to promote cache / vector efficiency (e.g., unit stride)
- **OpenMP task-level parallelism helpful to improve non-uniform, unstructured code regions, especially on many-core environments.**
  - 70% performance improvement observed in KNL by avoiding serialized code sections.
- **Heterogeneous CPU-GPU requires balancing workload across devices.**

Conclusions

- **Single-GPU performance improved with pinned memory but found to be detrimental in multi-device setting.**
  - 30% slower with non-pinned memory on one K40m but 25% faster in multi-GPU environment.
  - When combined with host-side multi-core parallelism, non-pinned is 32% faster with workload of 40% (ACC-to-OMP).
- **Newer P100 provided 46% performance improvement over K40m with no code modifications.**
- **Higher density PWR8 GPU system required high ACC-to-OMP workload ratio (100%) to reach maximum performance.**
  - HSW-Kepler system provided 17% higher overall performance but used twice as many nodes in heterogeneous environment: 8 K40m + 8 HSW CPUs v. 8 P100's + 4 PWR8 CPUs.
- **Current application spends majority of time in communication in heterogeneous environment.**
  - Must incorporate GPU-aware MPI methods to reduce host-device transfer costs.