GPU Acceleration of Multiphysics CFD Software for Propulsion and Power Flow Systems (RAPTOR)

Joseph C. Oefelein
Georgia Institute of Technology, Atlanta, GA

Ramanan Sankaran
Oak Ridge National Laboratory, Oak Ridge, TN

RAPTOR is a massively parallel flow solver designed to treat turbulent reacting flows in propulsion and power systems. These flows involve a multitude of strongly coupled fluid dynamic, thermodynamic, transport, chemical, multiphase, and heat transfer processes that are intrinsically coupled and must be considered simultaneously in the complex domains associated with; e.g., gas-turbine and rocket engines. In preparation for the Summit system currently being installed at the Oak Ridge Leadership Computing Facility (OLCF), a performance portable and GPU ready version of RAPTOR has been developed. A combination of programming models has been used to convert the original distributed memory parallel code to a hybrid parallel code with multiple levels of parallelism. Major performance critical kernels have been reimplemented in C++ using the Kokkos programming model, and the main flow solver has been accelerated using OpenMP 4.5 compiler directives. This poster presents our approach, recent progress, and performance characteristics of RAPTOR under the OLCF Center for Accelerated Application Readiness (CAAR) program.

Simulation of propulsion and power flow processes involves many challenges

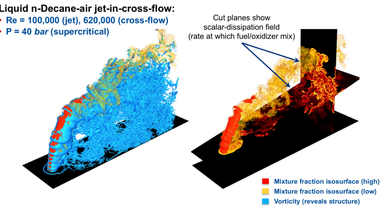
- Need for predictive models is well recognized (i.e., ultimate multiscale/multiphysics problem)
  - High Reynolds number turbulence and scalar mixing processes (Re > 100,000)
  - High-pressure, multiregime combustion
  - Compressible, acoustically active flow
  - Complex geometries and heat transfer
  - Complex fuels, multiphase flow
- Significant challenges still exist (i.e., current models are neither robust or predictive)
  - Simulations treat only a limited ranges of scales
  - Experiments provide limited information
  - Many sources of uncertainty
  - Costs can be prohibitive
- Advanced research requires a synergistic combination of unique capabilities
  - Coupling simulations and experiments
  - High-performance massively-parallel computing

Advanced treatment requires LES, detailed physics, high-quality numerics

- Theoretical framework (Comprehensive)
  - Fully-coupled, compressible conservation equations
  - Real-fluid equation(s) of state for non-ideal gas/liquid systems (high-pressure phenomena)
  - Detailed thermodynamics, transport, and chemistry
  - Multiphase flow, sprays (Lagrangian-Eulerian formulation)
  - Generalized subfilter model framework for treatment of turbulence
  - Dynamic subfilter modeling (no tuned constants)
- Numerical framework (High-quality)
  - Structured multiblock topology in generalized curvilinear coordinates with unstructured connectivity between blocks
  - Staggered finite-volume differencing (non-dissipative, discretely conservative)
  - Dual-time stepping with generalized preconditioning (all-Mach-number)
  - Detailed treatment of geometry, wall phenomena, transient BC's

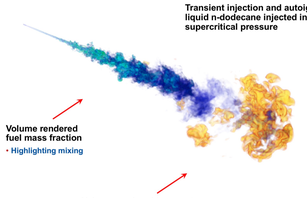
Example science application … injection and combustion of supercritical fuels

Liquid n-Decane-air jet-in-cross-flow:
- Re = 100,000 (jet), 620,000 (cross-flow)
- P = 40 *bar* (supercritical)

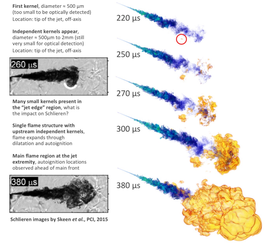


A. Ruiz, G. Lacaze and J. C. Oefelein. Flow topologies and turbulence scales in a jet-in-cross-flow. *Physics of Fluids*, **27**, 045101, 2015.

Example science application … injection and combustion of supercritical fuels

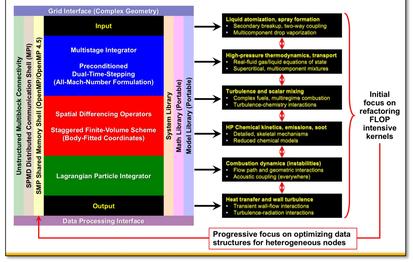


Results provide quantitative insights not available elsewhere



One-to-one correspondence with experiments allows us to extract data that complements the experimental measurements and imaging

Hybrid parallelization strategy



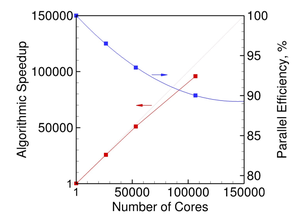
Parametric scaling studies

Grid	Cells across d_j	Grid size (in million cells)
G1	8	6.29
G2	16	50.33
G3	32	402.65
G4	64	3221.23

Topology	Blocks	G1	G2	G3	G4
T1	24	64^3	128^3	256^3	512^3
T2	192	32^3	64^3	128^3	256^3
T3	1536	16^3	32^3	64^3	128^3
T4	12288	8^3	16^3	32^3	64^3
T5	98304		8^3	16^3	32^3

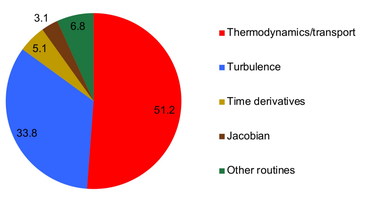
- Grids sizes of G1 to G4 for resolution (fine/coarse grain) tests
- Grid topologies T1 to T5 for performance analysis

Original MPI only version of RAPTOR has been well optimized



e.g., strong scaling on Titan

Performance profile on Titan



- Evaluation of EOS, thermodynamics, transport properties are most FLOP intensive
- Turbulence closure models are next most expensive (e.g., test-filtering operation)
- Complex chemistry, and Lagrangian spray solver are also FLOP intensive

Hybrid parallelization strategy

- Rewrite kernels in C++
  - Use Kokkos performance portable programming model
  - Provide interfaces for coupling with the Fortran solver
- Hybridize the main solver
  - Refactor from pure MPI to hybrid MPI+OpenMP
  - Port to GPUs using OpenMP 4.5

Rewrite kernels in C++ … Step 1: Vectorization of loops

```
subroutine convertYtoX(ys, xs, ws, ws_bar)
real, dimension(ns) :: ys, xs
real, dimension(ns) :: ws
real :: ws_bar

do m=1, ns
  xs(m)=ys(m)/ws(m)/ws_bar;
end do
```

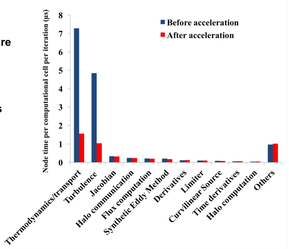
```
subroutine convertYtoX(ys, xs, ws, ws_bar)
real, dimension(nc, ns) :: ys, xs
real, dimension(ns) :: ws
real, dimension(nc) :: ws_bar

do i=1, nc
do m=1, ns
  xs(i,m)=ys(i,m)/ws(m)/ws_bar(i);
end do
end do
```

Rewrite kernels in C++ … Step 2: Rewrite as Kokkos kernel

```
struct convertYtoX {
  VectorFieldType ys;
  VectorFieldType xs;
  ScalarFieldType ws_bar;

  // constructor
  convertYtoX(
      VectorFieldType ys_, VectorFieldType xs_,
      ScalarFieldType ws_bar_)
      : ys(ys_), xs(xs_), ws_bar(ws_bar_) {};

  KOKKOS_INLINE_FUNCTION
  void operator()(const size_type i) const {
    for(int m=0; m<ns; m++){
      xs(i,m)=ys(i,m)/ws(m)/ws_bar(i);
    }
  }
};

Kokkos::parallel_for(nCells, convertYtoX(ys, xs, ws_bar));
```

Profile after kernel acceleration

- Kokkos kernels are coupled to the Fortran solver
- Performance profiled on Titan with 16 MPI ranks per node



Hybridization of the main solver in RAPTOR

- Code uses semi-implicit multistage scheme with dual-time integrator to advance the state (i.e., RHS operator) in time
- RHS operator targeted for CPU threading as first step
  - Scoping of private and shared variables
  - Loop analysis for potential data races
  - Created high level OMP regions with multiple parallel loops
  - Identification of wait and nowait based on data dependencies
  - Rewriting the loops and functions to remove false sharing
- Hybridization went better than expected due to favorable algorithmic structure
  - Loops are fine grained parallel and easily vectorized
  - No data races or dependencies across iterations
  - Two categories of data structures
    - Allocated and initialized once at start, do not change for duration of run
    - Allocated at start and periodically updated on host/device

Data Offloading with OpenMP 4.5

- OpenMP 4.5 target directives are used to map the host memory to device memory
- Two categories of data structures
  - Allocated and initialized once at the start and do not change for the duration of the program (such as grid metrics)

```
!$omp target enter data &
!$omp map(to: sxu, syv, szw)
```

  - Allocated at the start and need to be updated on the host/device periodically

```
!$omp target enter data &
!$omp map(alloc: qh, qv, dq)
........
!$omp target update to (qh)
........
!$omp target update from (dq)
```

Offloading loops to device (1)

Arrays mapped to target device already allocated at start using !$omp map (alloc: ...)

```
!$omp target
!$omp map    (      UF, VF, WF, dq) &
!$omp depend (in:   UF, VF, WF    ) &
!$omp depend (inout:             dq) &
!$omp nowait
!$omp teams distribute parallel &
!$omp do collapse(4) default(none) &
!$omp shared (    UF, VF, WF, dq) &
!$omp shared (nx, ny, nz, ne)
!$omp teams distribute parallel do collapse(4)
do l=1,ne
do k=1,nz-1
do j=1,ny-1
do i=1,nx-1
   dq(i,j,k,l) = dq(i,j,k,l)     - ( UF(i+1,j,k,l) - UF(i,j,k,l) &
                                 +   VF(i,j+1,k,l) - VF(i,j,k,l) &
                                 +   WF(i,j,k+1,l) - WF(i,j,k,l) )
end do; end do; end do; end do
!$omp end target
```

Offloading loops to device (2)

Dependencies on data are marked, data updates to/from target are similarly marked
!$omp target update to(dq)&
!$omp depend (out:dq)

```
!$omp target
!$omp map    (      UF, VF, WF, dq) &
!$omp depend (in:   UF, VF, WF    ) &
!$omp depend (inout:             dq) &
!$omp nowait
!$omp teams distribute parallel &
!$omp do collapse(4) default(none) &
!$omp shared (    UF, VF, WF, dq) &
!$omp shared (nx, ny, nz, ne)
!$omp teams distribute parallel do collapse(4)
do l=1,ne
do k=1,nz-1
do j=1,ny-1
do i=1,nx-1
   dq(i,j,k,l) = dq(i,j,k,l)     - ( UF(i+1,j,k,l) - UF(i,j,k,l) &
                                 +   VF(i,j+1,k,l) - VF(i,j,k,l) &
                                 +   WF(i,j,k+1,l) - WF(i,j,k,l) )
end do; end do; end do; end do
!$omp end target
```

Offloading loops to device (3)

Most kernels are marked "nowait," dependencies are marked explicitly

```
!$omp target
!$omp map    (      UF, VF, WF, dq) &
!$omp depend (in:   UF, VF, WF    ) &
!$omp depend (inout:             dq) &
!$omp nowait
!$omp teams distribute parallel &
!$omp do collapse(4) default(none) &
!$omp shared (    UF, VF, WF, dq) &
!$omp shared (nx, ny, nz, ne)
!$omp teams distribute parallel do collapse(4)
do l=1,ne
do k=1,nz-1
do j=1,ny-1
do i=1,nx-1
   dq(i,j,k,l) = dq(i,j,k,l)     - ( UF(i+1,j,k,l) - UF(i,j,k,l) &
                                 +   VF(i,j+1,k,l) - VF(i,j,k,l) &
                                 +   WF(i,j,k+1,l) - WF(i,j,k,l) )
end do; end do; end do; end do
!$omp end target
```

Offloading loops to device (4)

Loops are collapsed and parallelized; i.e., Raptor loops are fine grained parallel with no data races or dependencies across iterations

```
!$omp target
!$omp map    (      UF, VF, WF, dq) &
!$omp depend (in:   UF, VF, WF    ) &
!$omp depend (inout:             dq) &
!$omp nowait
!$omp teams distribute parallel &
!$omp do collapse(4) default(none) &
!$omp shared (    UF, VF, WF, dq) &
!$omp shared (nx, ny, nz, ne)
!$omp teams distribute parallel do collapse(4)
do l=1,ne
do k=1,nz-1
do j=1,ny-1
do i=1,nx-1
   dq(i,j,k,l) = dq(i,j,k,l)     - ( UF(i+1,j,k,l) - UF(i,j,k,l) &
                                 +   VF(i,j+1,k,l) - VF(i,j,k,l) &
                                 +   WF(i,j,k+1,l) - WF(i,j,k,l) )
end do; end do; end do; end do
!$omp end target
```

Offloading loops to device (5)

Explicit data sharing clauses, default set to none, loop indices are auto private

```
!$omp target
!$omp map    (      UF, VF, WF, dq) &
!$omp depend (in:   UF, VF, WF    ) &
!$omp depend (inout:             dq) &
!$omp nowait
!$omp teams distribute parallel &
!$omp do collapse(4) default(none) &
!$omp shared (    UF, VF, WF, dq) &
!$omp shared (nx, ny, nz, ne)
!$omp teams distribute parallel do collapse(4)
do l=1,ne
do k=1,nz-1
do j=1,ny-1
do i=1,nx-1
   dq(i,j,k,l) = dq(i,j,k,l)     - ( UF(i+1,j,k,l) - UF(i,j,k,l) &
                                 +   VF(i,j+1,k,l) - VF(i,j,k,l) &
                                 +   WF(i,j,k+1,l) - WF(i,j,k,l) )
end do; end do; end do; end do
!$omp end target
```

Strong and weak scaling on Titan

CPU + GPU yields factor of approximately 4x while maintaining good parallel efficiency across nodes

Accelerated performance on SummitDev using 2 P100 GPUs is 4.1x

	CPU	CPU + GPU	Speedup
Titan	42.3	11.1	3.8x
SummitDev	14.0	3.4	4.1x

- RAPTOR was benchmarked on SummitDev with accelerated Kokkos kernels
  - 240 blocks, 12 nodes, 20 MPI ranks per node
  - 10 MPI ranks per P8 socket
  - 5 MPI ranks share a P100 GPU through MPS
  - Units: microseconds/grid-points/time-step/node
- Better performance anticipated on Summit (i.e., using 9 CPUs, 6 NVIDIA V100 GPUs, w/NVLink)

Summary

- RAPTOR is being readied for Summit through a MPI+OpenMP+Kokkos programming model
- Code acceleration using Kokkos C++ library provides a performance portable path for transition to systems with heterogeneous architectures
- Core solver was hybridized with OpenMP to allow the performance space to be explored with MPI and threads independently
- Porting the regions to GPU targets through OpenMP 4.5 compiler support is currently in progress
- MPI task mapping to improve scalability is also being pursued

This research used resources of the **Oak Ridge Leadership Computing Facility** at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC05-00OR22725. Support provided by the **Center for Accelerated Application Readiness** program at Oak Ridge National Laboratory is gratefully acknowledged.