EARLY EXPERIENCE OF APPLICATION DEVELOPERS WITH OPENMP OFFLOADING
MOTIVATION FOR THIS BOF

- The current HPC environment is diverse and complex
  - Variety of hardware and multiple vendors providing their own programming interfaces and runtimes

- Critical for application developers to consider portable (and even better performance portable) solutions which can target different platforms across vendors
  - OpenMP is an open standard supported by nearly every vendor, and a promising solution

- Goals
  - Present examples about how developers are using OpenMP
  - Discuss lessons learned or best practices discovered
  - Provide feedback on any of the current OpenMP implementations (any new features desired?)
AUTHORS

- Hadia Ahmed
- Colleen Bertoni
- Reuben Budiardja
- Barbara Chapman
- Christopher Daley
- Jack Deslippe
- Johannes Doerfert
- Douglas Doerfler
- Brian Friesen
- Rahul Gayatri
- Yasaman Ghadar
- Pieter Ghysels
- Austin Harris
- Oscar Hernandez
- Brian Homerding
- Gustav Jansen
- Vivek Kale
- Thorsten Kurth
- JaeHyuk Kwack
- Raymond Loy
- Ye Luo
- Piotr Luszczek
- Brian Mackie-Mason
- Bronson Messer
- Jose Monsalve Diaz
- Matthew Norman
- John Tramm
- Tim Williams
- Charlene Yang
AUTHORS

- Hadia Ahmed
- Colleen Bertoni
- Reuben Budiardja
- Barbara Chapman
- Christopher Daley
- Jack Deslippe
- Johannes Doerfert
- Douglas Doerfler
- Brian Friesen
- Rahul Gayatri
- Yasaman Ghadar
- Pieter Ghysels
- Austin Harris
- Oscar Hernandez
- Brian Homerding
- Gustav Jansen
- Vivek Kale
- Thorsten Kurth
- JaeHyuk Kwack
- Raymond Loy
- Ye Luo
- Piotr Luszczek
- Brian Mackie-Mason
- Bronson Messer
- Jose Monsalve Diaz
- Matthew Norman
- John Tramm
- Tim Williams
- Charlene Yang
<table>
<thead>
<tr>
<th>Session</th>
<th>Minutes</th>
<th>Presenter or Moderator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fortran Session</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GenASiS</td>
<td>4</td>
<td>Reuben Budiardja</td>
</tr>
<tr>
<td>GESTS</td>
<td>4</td>
<td>Oscar Hernandez</td>
</tr>
<tr>
<td>Thornado/Exastar</td>
<td>4</td>
<td>Austin Harris</td>
</tr>
<tr>
<td>E3SM</td>
<td>4</td>
<td>Matt Norman</td>
</tr>
<tr>
<td>NUCCOR</td>
<td>4</td>
<td>Gustav R. Jansen</td>
</tr>
<tr>
<td>GAMESS</td>
<td>4</td>
<td>JaeHyuk Kwack</td>
</tr>
<tr>
<td>XGC</td>
<td>4</td>
<td>Brian MacKie-Mason</td>
</tr>
<tr>
<td><strong>Fortran Session Q&amp;A</strong></td>
<td><strong>15</strong></td>
<td>Tim Williams, all presenters</td>
</tr>
<tr>
<td><strong>C/C++ Session</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LAMMPS SNAP mini-app</td>
<td>4</td>
<td>Rahul Gayatri</td>
</tr>
<tr>
<td>MILC SU3 matrix-multiply mini-app</td>
<td>4</td>
<td>Doug Doerfler/Chris Daley</td>
</tr>
<tr>
<td>HPGMG</td>
<td>4</td>
<td>Hadia Ahmed/Chris Daley</td>
</tr>
<tr>
<td>QMCPack</td>
<td>4</td>
<td>Ye Luo/Thomas Applencourt</td>
</tr>
<tr>
<td>XSBench</td>
<td>4</td>
<td>John Tramm</td>
</tr>
<tr>
<td>Numerical Libraries</td>
<td>4</td>
<td>Piotr Luszczek</td>
</tr>
<tr>
<td><strong>C/C++ Session Q&amp;A</strong></td>
<td><strong>15</strong></td>
<td>Oscar Hernandez, all presenters</td>
</tr>
</tbody>
</table>
OPENMP RESOURCES

- OpenMP website
  - https://www.openmp.org

- OpenMP Validation and Verification
  - https://crpl.cis.udel.edu/ompvvsollve/

- OpenMP YouTube Channel
  - https://www.youtube.com/user/OpenMPARB/

- At ECP Annual Meeting:
  - OpenMP 4.5 and 5.0 Tutorial (Offload)
  - Wed Feb 5, 2020, 2:30 PM - 6:00 PM in Discovery A
OPENMP BOF - EARLY EXPERIENCE

TARGETING GPUS USING OPENMP DIRECTIVES ON SUMMIT WITH GENASIS

Reuben D. Budiardja
Oak Ridge Leadership Computing Facility

February 4, 2020
**APPLICATION**

**General Astrophysics Simulation System (GenASiS)**
- Current target: 3D position space + 1D momentum space the simulations of core-collapse supernovae; Towards 3D + 3D (sustained exascale)
- Previous results: studied of fluid instabilities in supernova dynamics, discovered exponential magnetic field amplification in progenitor star

**Code characteristics:**
- Modular, object-oriented design, extensible
- OpenMP 4.5 for offloading (recent work)
STORAGEFORM CLASS

A class for **data** and **metadata**; the ‘**heart**’ of data storage facility in GenASiS.

StorageForm % Value ( nCells, nVariables )
e.g. Pressure => StorageForm % Value ( :, 1 ),
    Density  => StorageForm % Value ( :, 2 ), ...

StorageForm % AllocateDevice ( )

Tells OpenMP data location on GPU
→ avoid (implicit) allocation & transfer
Offloading a Computational Kernel

Persistent allocation and association

```fortran
1 subroutine AddKernel ( A, B, C )
2 real ( KDR ), dimension ( : ), intent ( in ) :: A, B
3 real ( KDR ), dimension ( : ), intent ( out ) :: C
4
5 integer ( KDI ) :: i
6
7 !$OMP target teams distribute parallel do schedule ( static, 1 )
8 do i = 1, size ( C )
9 C ( i ) = A ( i ) + B ( i )
10 end do
11 !$OMP end target teams distribute parallel do
12
13 end subroutine AddKernel
```

call F % Initialize & ([nCells, nVariables])
call F % AllocateDevice ( )
call F % UpdateDevice ( )
call AddKernel &
( F % Value ( :, 1 ),
 F % Value ( :, 2 ),
 F % Value ( :, 3 )
)

No implicit data transfer, no explicit map()
Explicit management of data movement is necessary for overall performance
- use “pinned memory” → no mechanism in OpenMP 4.5, but in 5.x
  - resorted to wrapping CUDA calls for now
- persistent host ⇋ GPU data association done via class StorageForm method

Avoid complicated data structure in kernels
- pass “simple” arrays as subroutine arguments

Use “associate” construct to help match what OpenMP maps
- associate ( v_1 => F_Value ( :, F % Velocity(1) ) )

Speedups about ~15X on per Summit node (6 GPU to 42 CPU cores) for hydrodynamics solvers
PROBLEM ENCOUNTERED

With IBM XL, “-qoffload -qsmp=omp” flag on module with derived-type (DT) definition implies offloading the DT
- Unresolved linking errors if DT has type-bound procedure calling host-only subroutines
- IBM PMR → !$omp declare target device_type(host)
- Our solution: separate out “kernel” subroutines using F2008 submodules
  • Reduce # files compiled with those flags → faster compilation overall

- With OpenMP 4.5, no good way to have the same code / kernel for both OpenMP threads and offload
  - Workaround: code duplications with different OpenMP pragmas
  - Future: OpenMP 5 meta-directives?
OPENMP BOF - EARLY EXPERIENCE

FOURIER TRANSFORM CODE

KIRAN RAVIKUMAR (GEORGIA TECH)
PK YEUNG (GEORGIA TECH)
OSCAR HERNANDEZ (ORNL)

February 4, 2020
INTRODUCTION

- Multidimensional FFT provides a case study of wide relevance
  - Fluid dynamics, weather forecasting, signal processing are some applications that require Fourier transforms
  - Scalability often ultimately limited by communication costs
- State-of-the-art heterogeneous machines provide new opportunities
  - Computations accelerated using GPUs
  - Overlap operations to hide GPU compute, data transfers and network communication

- Languages, programming models, and dependency on libraries
  - Original code in CUDA Fortran
  - Port to Fortran using OpenMP to target GPUs for portability
  - CUDA FFT library for FFTs on GPU
  - MPI library for all-to-all transpose
BRIEF DESCRIPTION AND MOTIVATION

- Successfully developed a batched asynchronous algorithm to solve large problem sizes without being limited by GPU memory
  - CUDA Fortran code with asynchronous execution using CUDA Events and Streams
  - Presented results collected on Summit at SC19:

- Motivation for current work
  - Future systems may not support CUDA
  - Use OpenMP for a more portable code

- Target systems:
  - Summit: Two P9 CPUs and 6 NVIDIA NV100 GPUs using XLF compilers
  - Frontier: AMD CPUs/GPUs with Cray Fortran and gfortran compilers
CURRENT APPROACH

First pencil is copied in before entering the following loop.

```
do ip=1,np
   next=mod(ip+1,3); current=mod(ip,3); previous=mod(ip−1,3); comm=mod(ip−2,3);
   ! if any of above are zero, set to 3, negative cases are not considered
   ! Host→Device copy of pencil ip+1 (NEXT) using cudaMemcpy in transfer stream
   ierr =cudaEventRecord(HtoD(next),trans_stream)
   ierr =cudaStreamWaitEvent(comp_stream,HtoD(current),0)
   ! Compute on ip pencil (CURRENT) in compute stream eg., call cuFFT library
   ierr =cudaEventRecord(compute(current),comp_stream)
   ierr =cudaStreamWaitEvent(trans_stream,compute(previous),0)
   ! Device→Host copy of pencil ip−1 (PREVIOUS) using cudaMemcpy in transfer stream
   ierr =cudaEventRecord(DtoH(previous),trans_stream)
   ierr =cudaEventSynchronize(DtoH(comm))
   ! MPI Alltoall on pencil ip−2 copied out in the previous iteration
```

Last pencil is copied out after the loop. MPI on last 2 pencils are performed.

- 1 slab of data per MPI task
- Vertically divided among multiple GPUs (3 in above figure)
- 4 pencils per slab: different operations running in parallel on each pencil
1D FFT EXAMPLE USING OPENMP

```c
!$OMP PARALLEL
!$OMP SINGLE
!$OMP TARGET DATA MAP(to:a) MAP(from:b)

!$OMP TASK DEPEND(OUT:var)
!$OMP TARGET DATA USE_DEVICE_PTR(a)
call cufftExecC2C (...,CUFFT_FORWARD)
call cufftExecC2C (...,CUFFT_INVERSE)
!$OMP END TARGET DATA
!$OMP END TASK

!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO &
!$OMP PRIVATE(x,z,y) DEPEND(IN:var) NOWAIT
  Compute b(x,z,y)=a(x,z,y)*1.0/nx
!$OMP END TARGET TEAMS DISTRIBUTE PARALLEL DO
!$OMP TASKWAIT
!$OMP END TARGET DATA
!$OMP END SINGLE
!$OMP END PARALLEL
```

- Forward & Inverse Transform
- Transformed Data scaled and compared with initial data using OpenMP Target Loop
- Expect scalar multiplication to be performed after FFT finishes
- Gives incorrect results!
- Task dependency enforced on host
- Task performing FFT is considered complete when thread finishes launching cuFFT kernel
- Dependency not enforced based on when cuFFT kernel finishes on GPU
LESSONS LEARNED AND NEXT STEP

- Lessons learned
  - Use cudaEventRecord and cudaEventSynchronize to ensure host waits for cuFFT kernel to complete
  - Above work around by mixing CUDA Fortran and OpenMP: but not a permanent solution

- Next step
  - Explore TASK DETACH and other OpenMP 5.x features that might help when compiler support is available
THANK YOU!
OPENMP BOF - EARLY EXPERIENCE

THORNADO (EXASTAR)

J. AUSTIN HARRIS (ORNL)
M. PAUL LAIU (ORNL)
RAN CHU (UTK)
EIRIK ENDEVE (ORNL)
EXASTAR

- Component-based multi-physics AMR-based toolkit for simulating stellar explosions (e.g., supernovae, neutron-star mergers)
  - Couple physics in AMR-based Flash/Castro codes

- Key physics modules
  - Spectral neutrino transport (compiler directives and linear algebra libraries)
  - Nuclear burning (compiler directives and linear algebra libraries)
  - Magnetohydrodynamics (compiler directives)
  - Gravity (compiler directives)
  - Equation of state (compiler directives)
THORNADO

ExaStar proxy application for spectral neutrino transport

- High-order discontinuous Galerkin (DG) methods
- Implicit-Explicit (IMEX) time integration
  - Explicit neutrino advection operator
  - Implicit neutrino collision operator
    - Nonlinear solver
    - Tabulated microphysics
- Designed to be incorporated into AMR-based codes
  - Focus on node-level performance
- GPU port of original CPU code via compiler directives and linear algebra libraries
THORNADO GPU STRATEGY

COMPILER DIRECTIVES

LINEAR ALGEBRA LIBRARIES

SUBROUTINE MatrixMatrixMultiply ( ... )

# if defined ( THORNADOOMP )
#omp target teams distribute parallel do simd collapse (7)
# else defined ( THORNADOACC )
#acc parallel loop gang vector collapse (7)
# endif

do
i = iZ + 1, iZ + 1 + 1 ; ... ; do iNode = 1, nDOF
uCrK ( iNode, iZ1, iZ2, iZ3, iM, iS, iZ4 ) = U ( iNode, iZ1, iZ2, iZ3, iZ4, iM, iS )
end do ; ... ; end do

# if defined ( WEAKLIBOMP )
#omp target teams distribute if ( do.gpu ) private( iT, dT, dIX, dX )
# else defined ( WEAKLIBOACC )
#acc parallel loop gang async( async_flag ) if ( do.gpgu ) private( iT, dT, dIX, dX )
# endif

do
k = 1, size ( logT )

iT = ... ; dT = ...

ith = ... ; iX = ...
end do ; ... ; end do

# if defined ( WEAKLIBOMP )
#omp parallel do simd private( i0, j0, i, j )
# else defined ( WEAKLIBOACC )
#acc loop vector private( i0, j0, i, j )
# endif

do

j = j0 + 1, sizeE + ( sizeE + 1 ) / 2 ! Fused triangular loop: do j = 1, sizeE : do i = 1, j

j0 = mod ( ( i - 1 ) / ( sizeE + 1 ), sizeE + 1 ) + 1

i0 = mod ( ( j - 1 ) / ( sizeE + 1 ), sizeE + 1 ) + 1

if ( i > j ) then

j = sizeE - j0 + 1 ; i = sizeE - i0 + 2
else

j = j0 ; i = i0
end if

interpolant ( i, j, k ) = bilinear &
( table ( i, j, iT, dIX ) ), table ( i, j, iT + 1, IX ), &

table ( i, i, IT, IX + 1), table ( i, iT, IX + 1, IT + 1, IX + 1, dT, dX )
end do ; ... ; end do

SUBROUTINE MatrixMatrixMultiply &
( transa, transb, m, n, k, alpha, a, lda, b, ldb, beta, c, ldc )

CHARACTER

transa, transb :: char

INTEGER

m, n, k, lda, ldb, ldc

REAL ( DP )

alpha, beta

REAL ( DP ), DIMENSION ( lda, * ), TARGET :: a

REAL ( DP ), DIMENSION ( ldb, * ), TARGET :: b

REAL ( DP ), DIMENSION ( ldc, * ), TARGET :: c

END SUBROUTINE

CALL MatrixMatrixMultiply &
( 'N', 'N', nDOF, XI, NF, nDOF, One, L-X3-Up, nDOF, X3, &

uCrK ( iZB1, iZB2, iZB3, 1, iZB4 ), nDOF, Zero, &

uCrL ( iZB1, iZB2, iZB3, 1, iZB4 ), nDOF, XI )
DELEPTONIZATION WAVE TEST PROBLEM

- Proxy for CCSN problem
- Evolve one “block” with _thornado_
  - Energy discretization: 32 points
  - Spatial discretization: $12^3$ points
- ~100x speedup relative to serial CPU code
PROBLEMS ENCOUNTERED (IF ANY)

- Compilers
  - Very long compile time with IBM XL Fortran using OpenMP OL
  - Poor kernel launch configuration for some kernels with OpenMP OL (XL)

- Migration from other programming models
  - Challenging to port iteration kernels due to register pressure (e.g., Newton-Raphson)
    - Solved by packing “active” elements and breaking up iteration into smaller kernels

- Interoperability/libraries
  - Customized MAGMA routines for pre-allocated work arrays
  - Complicated interfaces for batched linear algebra routines with Fortran (e.g., array of pointers on device)

- Tools
  - Arm MAP useful for low-overhead, easy-to-use sampling for initial profiling
LESSONS LEARNED AND NEXT STEP

- Lessons learned
  - Write "portable" CPU code (e.g., tight loop nests, libraries when possible)
  - Often faster for CPU as well

- Next step
  - Benchmarking and scaling with FLASH+thornado in AMR framework
  - GPU porting of additional microphysics and solvers
  - Performance implications for higher-order DG
  - Optimization/tuning
    - Asynchronous GPU kernels

This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.
THANK YOU!
E3SM EARLY OPENMP EXPERIENCES

MATTHEW NORMAN (ORNL)
ENERGY EXASCALE EARTH SYSTEM MODEL (E3SM)

- Coupled high-resolution climate model
  - Atmospheric component is the most expensive
  - Better approximated clouds with embedded Cloud Resolving Models (CRMs)

- Cloud Resolving Model (CRM) Code is in Fortran 90
  - About 10K Lines of Code, but consumes > 90% of the runtime
  - No MPI communication in the CRM code
ENERGY EXASCALE EARTH SYSTEM MODEL (E3SM)

- Past Work
  - First ported to GPUs with OpenACC
  - Exposed parallelism by pushing loops down callstack and promoting variables
  - Had to remove pointers and derived types from all kernels (PGI bugs)

- Motivation for current work
  - PGI bugs in OpenACC on Summit
  - PGI unlikely to be available on Frontier
  - Greater portability
CURRENT APPROACH

- Make the CRM code as portable as possible
  - Not allowing any derived types or classes (no “%” in data statements or kernels)
  - Allocate model with a custom C++ allocator that uses CUDA managed memory
    - Uses cudaMallocManaged and cudaMemPrefetchAsync for performance
    - Also maps data in OpenACC and OpenMP runtimes via:
      - acc_map_data() in OpenACC
      - omp_target_associate_ptr() in OpenMP offload
    - Forces offload runtimes to ignore data transfers and leave it to the CUDA runtime instead
    - Now, you no longer need data statements in either offload approach
  - In this manner, OpenMP and OpenACC APIs are nearly one-to-one
    - Can switch between the two with preprocessor directives
  - Examples: https://github.com/mrnorman/YAKL | https://github.com/mrnorman/gator

- Previous approach has been demonstrated in a smaller app
- Currently working to port all of the CRM code to OpenMP in this manner
PROBLEMS ENCOUNTERED (IF ANY)

- Compilers
  - GNU OpenMP is pretty far behind in performance and coverage
  - Not easy to access Cray compiler except through ECP test machines
  - Cray and IBM XL have different definitions of simd and parallel do on the GPU
    * Fairly easily handled with preprocessors
  - Cray compiler fails in the omp_target_associate_ptr() approach
    * But they are aware of it and working on it
LESSONS LEARNED AND NEXT STEP

- Lessons learned
  - Simplify data used in kernels as much as possible
    - Shield kernels and data statements from classes via parameter passing
  - For large codes with hundreds of kernels and variables, it’s best not to have to use data statements at all
    - Achievable through data mapping and C++ allocations with CUDA Managed Memory

- Next Steps
  - Continued answer bug and performance bug reports to vendors
OPENMP BOF EARLY EXPERIENCE - NUCCOR

Gustav R. Jansen <jansengr@ornl.gov>
NUCCOR (NUCLEAR COUPLED CLUSTER OAK RIDGE)

Science

- Computing properties of atomic nuclei from forces between protons and neutrons:
  - Binding energies (mass)
  - Radii
  - Density distributions
  - Excitation spectra
  - Radioactive properties

- Developed at ORNL for the last 20 years
- Fortran 2003/2008/20XX
- Based on NTCL (Nuclear Tensor Contraction library)
- Many (>1M) dense tensor contractions
TYPICAL TENSOR CONTRACTION – SIMPLE VERSION

```c
!$omp target data map(t2,t3,v,f)
!$omp target teams distribute private(b,m,temp1,temp2) collapse(3)
do  ij = 1, p%nj
  do  bidx = 1, p%nc
    do  a = 1, p%np
      temp1 = 0.0d0
      do  midx = 1, p%nk
        m = p%kmap(midx)
        temp2 = temp2 + t3(ef, bidx, ij, midx)*v(ef, a, m)
      end do
    end do
  end do
!$omp end parallel do simd
  temp1 = temp1 + temp2*f(midx, a, bidx)
end do
b = p%cmap(bidx)
t2(a, b, ij) = t2(a, b, ij) + temp1
end do
!$omp end target teams distribute
!$omp end target data
```

- Sizes vary many orders of magnitude
- $ef \ll a, b \ll ij \ll m$
- Different compilers takes different directives
- Performance not stable between compilers
- Performance not there compared to CUDA and HIP implementations
WHAT WE REALLY WANT TO DO!

Transpose ➔ Transpose ➔ Gemm ➔ Transpose

Transpose ➔ Transpose ➔ Gemm ➔ Transpose

Transpose ➔ Transpose ➔ Gemm ➔ Transpose

Transpose ➔ Transpose ➔ Gemm ➔ Transpose

OpenMP

Library
SUMMARY

• Performance not quite there
• Fortran compilers need to catch up
• Must be able to combine OpenMP with accelerated libraries

Thank you!
OPENMP BOF - EARLY EXPERIENCE

GAMESS RI-MP2 FORTRAN KERNEL W/ OPENMP OFFLOADING

JAEHYUK KWACK (ANL)
BUU PHAM (IOWA STATE UNIVERSITY)
COLLEEN BERTONI (ANL)

February 4, 2020
GAMESS

- A general quantum chemistry and *ab initio* electronic structure code
  - *ab initio* SCF energies (e.g. RHF and MCSCF)
  - Force fields (e.g., the Effective Fragment Potential)
  - Perturbative corrections to Hartree-Fock (e.g., MP2 and RI-MP2)
  - Near-linear scaling fragmentation methods (e.g., Fragment Molecular Orbital method)
  - *ab initio* gradients, hessians, and geometry optimizations

- Languages, programming models, and dependency on libraries
  - Mainly written in Fortran
  - An optional C++ library with re-implementations of certain methods
  - BLAS (e.g., MKL, ESSL, CUBLAS)
  - An MPI parallelization library (DDI library) written in C
  - MPI + X
    - OpenMP for CPU cores
    - CUDA for NVIDIA GPU accelerators.
GAMESS RI-MP2 FORTRAN KERNEL

- Written in Fortran; therefore it has limited possible options for GPU computing.
- Need a performance portable model on GPUs from multiple vendors (Intel, NVIDIA, AMD)
- FMO/RI-MP2 is one of the quantum chemistry algorithms of interest
  - Algorithm of interest in GAMESS ECP problem

RI-MP2 equations:

\[
E^{(2)} = \sum_{i \leq j}^{occ} (2 - \delta_{ij}) \sum_{ab}^{vir} \frac{(ia|jb)[2(ja|jb) - (ib|ja)]}{\epsilon_i + \epsilon_j - (\epsilon_a + \epsilon_b)}
\]

\[
(ia|jb) = \sum_{n}^{aux} B_{an}^i B_{bn}^j
\]
**SCALE-UP**

- Restructured the RI-MP2 kernel to increase concurrency on GPUs
- Testing multiple BLAS kernels
  - CUBLAS, CUBLASXT and NVBLAS
  - Intel oneMKL on Aurora testbed (in progress)

<table>
<thead>
<tr>
<th>Input: c60.kern</th>
<th>WTime(s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Serial w/ 1 core of P9</td>
<td>342.697</td>
<td>0.036 x</td>
</tr>
<tr>
<td>OpenMP threading + ESSL w/ 42 threads on 2 P9</td>
<td>12.231</td>
<td>1 x</td>
</tr>
<tr>
<td>OpenMP offloading + NVBLAS on 1 V100</td>
<td>1.734</td>
<td>7.05 x</td>
</tr>
<tr>
<td>OpenMP offloading + CUBLAS on 1 V100</td>
<td>1.983</td>
<td>6.17 x</td>
</tr>
<tr>
<td>OpenMP offloading + CUBLASXT on 1 V100</td>
<td>1.728</td>
<td>7.08 x</td>
</tr>
</tbody>
</table>

Similar to the peak ratio of one V100 GPU over two IBM P9 CPUs (i.e., 7.22x)
**SCALE-OUT**

- Testing MPI + OpenMP offloading on multiple GPUs and multiple compute nodes on Summit

The peak ratio of six V100 GPUs over two IBM P9 (i.e., 43.33x)
PROBLEMS ENCOUNTERED (IF ANY)

- **Compilers**
  - IBM XLF on OLCF Summit works fine
  - Intel IFORT on ALCF JLSE works fine

- **Migration from other programming models**
  - Smoothly migrated *in a week* during ECP OpenMP Hackathon

- **Interoperability/libraries**
  - CUBLAS and CUBLASXT make Fortran kernel messy, but functional.
  - NVBLAS uses standard BLAS calls; it helps code keep clean.
    - However, **NVBLAS and CPU math library** (e.g., ESSL, MKL, and ArmPL) use the *same symbol* (e.g., DGEMM), and it may result in *unexpected errors* or *lower performance on heterogeneous architecture*.
      - OpenMP 5 *declare variant* directive may figure out this symbolic conflict

- **Tools**
  - NVPROF works fine
LESSONS LEARNED AND NEXT STEP

Lessons learned
- On a single NVIDIA V100 GPU, the OpenMP offloading kernels show
  • more than 7x speedup over 42 threaded code on IBM P9 processors,
  • around 200 x speedup over the serial run on IBM P9 processors.
- On the same number of Summit nodes, the MPI+OpenMP offloading kernels show
  • More than 40x speedup over the MPI + OpenMP threading kernels.

Next step
- Near term
  • Actively working on Aurora testbed for Intel GPUs
  • Extending OpenMP offloading implementations to other kernels of GAMESS
- Long term
  • Running GAMESS on NERSC Perlmutter, ALCF Aurora and OLCF Frontier at scale

Anything else
THANK YOU!
EARLY OPENMP EXPERIENCE WITH COLLISION KERNEL

BRIAN MACKIE-MASON
XGC TEAM

February 4, 2020
XGC INTRODUCTION

- Collision operator used after particles pushed
- Uses Landau interaction tensor, advection/diffusion tensors, elliptic integrals
  - E.S. Yoon, et. al., Physics of Plasmas 21 (2014), 032503
- Pre-exascale physics simulating two different species (deuteron-electron)
- Exascale goal is to simulate top ten ion impurity species of ITER
PROGRAMMING MODEL CHOICES

- Cori/Theta (MIC) – OpenMP 3.0
- Titan/Summit – OpenACC
- Perlmutter – OpenMP 4.5+
- Aurora – OpenMP 4.5+
- Frontier – OpenMP 4.5+
- Etc…
COMPILATION

- IBM XL compilers (xlf2008_r)
- FC = mpifort
- FFLAGS = -llapack -lblas -qsmp=omp -qoffload -O2
- XGCFLAGS = -D_OPENMP_OFFLOAD

- PGI compilers (pgfortran 19.4)
- FC = mpifort
- FFLAGS = -llapack -lblas -mp -fast -O3
- ACC_FLAGS = -acc -ta=tesla:cc70,ptxinfo,managed -Minfo=accel -Mnostack_arrays -DUSE_ASYNC
OPENACC->OPENMP OFFLOAD (MEMORY)

allocate(M_s( col_f_nvr-1, 5, (col_f_nvr-1)*(col_f_nvz-1), &
col_f_sp_s:col_f_sp_e, &
isize ), stat=alloc_stat)

#ifdef _OPENACC
!$acc enter data pcreate(M_s)
#elif _OPENMP_OFFLOAD
!$omp target enter data map(alloc:M_s)
#endif

deallocate(M_s, stat=alloc_stat)

#ifdef _OPENACC
!$acc exit data delete(M_s)
#elif _OPENMP_OFFLOAD
!$omp target exit data map(release:M_s)
#endif

Result: Appears to work based on other observations
OPENACC->OPENMP OFFLOAD (SUCCESS)

Result:
~4.5x speed-up
OPENACC -> OPENMP OFFLOAD (W.I.P.)

Result: Incorrect result. Two directives diverge with external module call.
THANK YOU!
BMACKIE@ANL.GOV
FORTRAN SESSION Q&A
C/C++ SESSION
OPENMP BOF - EARLY EXPERIENCE

SNAP ON GPU USING OPENMP~4.5

RAHUL GAYATRI
AIDAN THOMPSON
STAN MOORE

February 4, 2020
EXAALT – ECP PROJECT

- ECP EXAALT project seeks to extend the accuracy, length, and time scales of material science simulations for fission/fusion reactors using LAMMPS MD

- Primary KPP target is MD of nuclear fusion materials that uses the SNAP interatomic potential in LAMMPS
  - Performance directly depends on the single node performance of SNAP

- C++, Math Libraries (math.h)
TESTSNAP – A PROXY APP FOR LAMMPS/SNAP

- TestSNAP - an independent standalone application for the SNAP module in LAMMPS
- Testbed for various parallelization strategies and optimizations
- Successful optimizations are merged into LAMMPS

for(num_atoms) // loop over atoms
{
    build_neighborlist(); // build neighborlist for each atom
    compute_ui();
    compute_yi();
    for(num_nbors) // loop over neighbors
    {
        compute_duidrj();
        compute_dbidrj();

        update_force(); // update force for (atom,nbor) pair
    }
}
for(num_atoms)
{
  build_neighborlist();
  compute_ui();
  compute_yi();
  for(num_nbors)
  {
    compute_duidrj();
    compute_dbidrj();
  }
  update_force();
}
OPENMP OFFLOAD – MAP DATA TO DEVICE

#pragma omp target enter data map(to:this[0:1])
#pragma omp target enter data map(to:this->U[0:U.size], ...)
OPENMP OFFLOAD – DISTRIBUTE WORK ACROSS TEAMS AND THREADS

```c
void compute_ui
{
    #pragma omp target teams distribute parallel for
    for(num_atoms)
    {
        ......
    }
}

#pragma omp target enter data map(to:this[0:1])
#pragma omp target enter data map(to:this->U[0:U.size], ...)
```
OPENMP OFFLOAD – COLLAPSE ATOM AND NEIGHBOR LOOPS

```c
void compute_ui
{
    #pragma omp target teams distribute parallel for
    for(num_atoms)
    {
        ....
    }
}

void compute_duidrj
{
    #pragma omp target teams distribute parallel for\
    collapse(2)
    for(num_nbor)
    {
        for(num_atoms)
        {
            ....
        }
    }
}

#pragma omp target enter data map(to:this[0:1])
#pragma omp target enter data map(to:this->U[0:U.size], ...)
```
OPENMP OFFLOAD – MAP THE OUTPUT BACK TO HOST

```c
#pragma omp target enter data map(from:this[0:1])
#pragma omp target enter data map(to:this->U[0:U.size], ...)

void compute_ui {
    #pragma omp target teams distribute parallel for
    for(num_atoms) {
        ......
    }
}

void compute_duidrj {
    #pragma omp target teams distribute parallel for\
    collapse(2)
    for(num_nbor) {
        for(num_atoms) {
            ......
        }
    }
}

#pragma omp target exit data map(from:this->deidrj[0:deidrj.size])
```
PERFORMANCE AND LESSONS LEARNT

- Optimized Kokkos implementation
  - Within 90% of the equivalent optimized kokkos implementation

- Lessons learnt
  - Optimize on data movements
  - Column major data access pattern for GPUs (row major: 3x slower for OpenMP~4.5 SNAP implementation)
  - No performance differences with `#pragma omp alloc` vs `#pragma omp map to/from` if its done same number of times
  - For multiple synchronization points between host-device `#pragma alloc` and `#pragma update` to is a preferable

Compilers

- XL – summit (xl/16.1)
- Clang – Cori-GPU (llvm-10.0)
THANK YOU!
SU(3) BENCHMARK (QCD PROXY)

CHRIS DALEY (LBNL)
DOUG DOERFLER (LBNL)
OVERVIEW

- The SU(3) benchmark calculates a matrix-matrix multiply of complex number matrices
  - Derived from the MILC Lattice QCD code
- The benchmark performs ~4 million 3x3 matrix-matrix multiplies
  - Operations on 3x3 matrices are a fundamental building block of LQCD applications
- It is written in C++ and has no library dependencies
- Doug Doerfler has created GPU implementations in CUDA, OpenCL, OpenMP target offload, OpenACC, and SYCL: https://bitbucket.org/dwdoerf/su3_bench
OPENMP TARGET OFFLOAD APPROACH

```c
#pragma omp target teams distribute
for(int i=0; i<1048576; ++i) {
    #pragma omp parallel for collapse(3)
    for(int j=0; j<4; ++j) {
        for(int k=0; k<3; k++) {
            for(int l=0; l<3; l++) {
                Complx cc;
                for(int m=0; m<3; m++) {
                    cc += d_a[i].link[j].e[k][m] * d_b[j].e[m][l];
                }
                d_c[i].link[j].e[k][l] = cc;
            }
        }
    }
}
```

Use teams parallelism for the ~1 million sites

Use thread parallelism for the matrices associated with the 4 “links” per site
MIXED OPENMP TARGET OFFLOAD PERFORMANCE ON CORI-GPU AND SUMMIT

Analytical roofline on NVIDIA V100 GPUs

- Benchmark is memory bandwidth bound but is sensitive to overheads

- Terrible performance in LLVM/Clang because of excess GPU memory traffic (implicit OpenMP flushes)
#pragma omp target teams num_teams(NTEAMS) thread_limit(NTHREADS) {
  #pragma omp parallel
  {
    int total_teams = omp_get_num_teams();
    int team_id = omp_get_team_num();
    int sites_per_team = (total_sites + total_teams - 1) / total_teams;
    int istart = team_id * sites_per_team;
    if (istart > total_sites) istart = total_sites;
    int iend = istart + sites_per_team;
    if (iend > total_sites) iend = total_sites;

    // This is the total_sites loop manually chopped up
    for (int i = istart; i < iend; ++i) {
      #pragma omp for collapse(3)
      for (int j=0; j<4; ++j) {
        // No code between teams and parallel OpenMP directives
      }
    }
  }
}

Performance improves by 33x to 401 GFLOP/s
LESSONS LEARNED AND NEXT STEP

- All compilers successfully ran the benchmark but performance was varied
- We needed to manually SPMDize the code to get good performance with LLVM/Clang
  - This avoided $O(1 \text{ million})$ implicit memory flushes
  - Future versions of LLVM/Clang will hopefully include the TRegion enhancement [1] to
    make manual optimization unnecessary
- We plan to profile the benchmark to identify the remaining sources of differences between
  OpenMP compilers

and Compiler Optimizations for OpenMP Target Regions." In International Workshop on
THANK YOU!
OPENMP BOF - EARLY EXPERIENCE

HPGMG-FV

CHRISTOPHER DALEY - LBNL
HADIA AHMED - LBNL
MAT COLGROVE - NVIDIA
HPGMG-FV OVERVIEW

- HPGMG-FV is a finite-volume based geometric multigrid solver
  - It is a standalone benchmark used to produce an alternative Top-500 list

- The selected configuration is a Full Multigrid (FMG) which is 4th order accurate and uses the out-of-place GSRB smoother

- The upstream version of HPGMG-FV is written in C99, MPI, and OpenMP
HPGMG-FV FOR GPUS

- There is already a CUDA version of HPGMG-FV for GPUs: https://bitbucket.org/nsakharnykh/hpgmg-cuda
  - This version depends on managed memory (cudaMallocManaged)

- We have developed OpenACC and OpenMP target offload versions of HPGMG-FV to improve the chance that this benchmark is included in the SPEChpc® 2020 benchmark suite
  - 1:1 translation of CUDA kernels to OpenACC and OpenMP target offload
  - Data is still allocated using cudaMallocManaged
    • Programmer-controlled data movement is work in progress
# ADDING DIRECTIVES WAS SIMPLE – BIGGEST CHALLENGE IS CORRECTNESS

<table>
<thead>
<tr>
<th>Operation</th>
<th>IBM 16.1.1</th>
<th>PGI 19.9 (OpenACC)</th>
<th>LLVM/Clang 10.0</th>
<th>Cray 9.1.0 - LLVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>increment_block</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>copy_block</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>extrapolate_betas</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>residual</td>
<td>✓</td>
<td>✓</td>
<td>VE (success at –O0)</td>
<td>VE (success at –O0)</td>
</tr>
<tr>
<td>rebuild</td>
<td>✓</td>
<td>✓</td>
<td>VE (success at –O0)</td>
<td>VE (success at –O0)</td>
</tr>
<tr>
<td>color_vector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>smooth</td>
<td>✓</td>
<td>✓</td>
<td>VE (success at –O0)</td>
<td>VE (success at –O0)</td>
</tr>
<tr>
<td>restriction</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>apply_BCs_v1</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>apply_BCs_v2</td>
<td>✓</td>
<td>VE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>apply_BCs_v4</td>
<td>✓</td>
<td>VE</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>interpolation_v4</td>
<td>✓</td>
<td>CE (can workaround)</td>
<td>VE (success at –O0)</td>
<td>VE (success at –O0)</td>
</tr>
<tr>
<td>Interpolation_v2</td>
<td>✓</td>
<td>CE (can workaround)</td>
<td>VE (success at –O0)</td>
<td>VE (success at –O0)</td>
</tr>
<tr>
<td>zero_vector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>scale_vector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>shift_vector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>mul_vector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>add_vector</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>sum</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>max_abs</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

20 GPU kernels:

CE = Compilation Error

VE = Verification Error
INITIAL OPENMP OFFLOAD PERFORMANCE WITH IBM COMPILER IS ENCOURAGING

- No performance results for LLVM/Clang or Cray because of –O0 requirement
- We are currently working with LLVM/Clang developers to identify and fix the issue: [https://bugs.llvm.org/show_bug.cgi?id=44390](https://bugs.llvm.org/show_bug.cgi?id=44390)
LESSONS LEARNED

- Easy to translate from CUDA launch configuration to OpenACC / OpenMP loops
  - However, we found several compiler bugs when using strict correctness tests

- All the tested compilers (IBM, PGI, Clang, Cray) successfully interoperate with CUDA
  - CUDA allocated data can be used in OpenACC / OpenMP kernels
  - CUDA kernels can be used instead of incorrect OpenACC / OpenMP kernels

- Poorly documented compiler options made benchmark validation harder, e.g. “-Xllvm2ptx -nvvm-compile-options=-fma=0” to turn off fused-multiply-add in IBM compiler
THANK YOU!
REAL-WORLD OPENMP 4.5 EXPERIENCE WITH QMCPACK

YE LUO, CPS & ALCF, ARGONNE
THOMAS APPLENCOURT, ALCF, ARGONNE

February 4, 2020
QMCPACK

- QMCPACK, is a modern high-performance open-source Quantum Monte Carlo (QMC) simulation code for electronic structure calculations of molecular, quasi-2D and solid-state systems.
- The code is C/C++ and adopts MPI+X(OpenMP/CUDA)
- Monte Carlo: massive Markov chains (walkers) evolving in parallel. 1st level concurrency. Good for MPI and coarse level threads.
- Quantum: The computation in each walker can be heavy when solving many body systems (electrons). 2nd level concurrency. Good for fine level threads and SIMD.
- Math libraries: BLAS/LAPACK, HDF5, FFTW
GPU STATUS

- Legacy CUDA implementation efficient for small problem sizes
  - Limited functionality
  - Not portable to non NVIDIA GPUs.
- Need a performance portable design to cover all current and Exascale systems
  - Refactor with a more flexible code architecture (C++)
  - Using portable programming model (OpenMP, Kokkos) for accelerators
- Three sets of code for different needs
  - Standalone fake kernels for testing OpenMP compilers
  - MiniQMC containing OpenMP offload having QMCPACK kernels with fake input
  - QMCPACK full application with OpenMP offload to feel the real-world.
FAKE KERNELS VALIDATING COMPILERS

- OpenMP V&V improves a lot but still has blind spots
- Fake GEMV kernels to validate OpenMP usage in QMCPACK
- Correctly running on Nvidia, AMD, Intel GPUs recently
- Cover many compilers
- Still need to try PGI on NVIDIA cards and GCC on AMD cards

https://github.com/ye-luo/openmp-target/wiki/GEMV-tests
MINIQMC IS A CHALLENGE

- Functionality bugs
  - Basic math functions
  - Declare persistent data on device
  - Static linking fat binaries
- Needed for performance
  - Asynchronous tasking
  - Multi GPU stream/queue support

XL is the only survival
Other compilers need further improvements
QMCPACK RUNS

- Runs on Summit with XL compiler
- Runs on AMD Radeon VII card with AOMP compiler
- QMCPACK developers will keep improve the adoption of OpenMP
- Compiler developers can improve the quality of compiler frontend, openmp runtime library to help boosting performance.

Let us make OpenMP shine on all exascale systems
THANK YOU!
OPENMP BOF - EARLY EXPERIENCE

MONTE CARLO PARTICLE TRANSPORT CASE STUDY: XSBENCH

JOHN TRAMM
Background

- Mini-app representing key kernel of Monte Carlo particle transport simulation
- Features stochastic access pattern of large data table structures

Code Details

- C
- No library dependencies
- Versions available: OpenMP (threading), OpenMP (offloading), OpenCL, CUDA, SYCL
XSBench

- XSBench is a mini-app representing a key kernel from the full application OpenMC

- Experience with performance and limitations for different programming models will inform model selection decision for OpenMC
RESULTS

XSbench FOM Performance on V100
(Higher is Better)

<table>
<thead>
<tr>
<th></th>
<th>FOM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA</td>
<td>2.4</td>
</tr>
<tr>
<td>CUDA (Optimized)</td>
<td>3.2</td>
</tr>
<tr>
<td>OpenCL</td>
<td>2.1</td>
</tr>
<tr>
<td>hipSYCL</td>
<td>3.1</td>
</tr>
<tr>
<td>OpenMP Offload (Xeon host, clang-ykt)</td>
<td>1.0</td>
</tr>
<tr>
<td>OpenMP Offload (Power9 host, xlc_r)</td>
<td>2.5</td>
</tr>
</tbody>
</table>
LESSONS LEARNED AND NEXT STEPS

- Lessons learned
  - OpenMP offloading relatively easy to implement
  - Available compilers worked well – no bugs.
  - IBM XL compiler generated fast code (better than naive CUDA implementation!)
  - However, clang-ykt was 2.5x slower than IBM XL

- Next steps
  - For full application, OpenMC, we are planning to use OpenMP offloading
  - Reasons:
    - Development team already familiar with OpenMP threading model, pragma verbiage
    - Performance was very good
    - Appears it will be portable to a variety of exascale systems
THANK YOU!
OPENMP BOF - EARLY EXPERIENCE

OPENMP 4+ IN NUMERICAL LIBRARIES: MAGMA, PLASMA, AND SLATE

MARK GATES (SLATE)
PIOTR LUSZCZEK (PLASMA)
STANIMIRE TOMOV (MAGMA)

February 4, 2020
PARALLEL NUMERICAL LINEAR ALGEBRA

- Numerical linear algebra: scope
  - Linear systems: Ax=b
  - Least squares: min||Ax-b||
  - Eigenvalue problems: Ax=λx
  - Generalized eigenvalue problems: Ax=λBx
  - Singular value problem: Av=σu
  - Decompositions: Cholesky, LU, LDLᵀ, QR, Schur, UΣVT
  - Sparse, batched, DNN tensors

- Languages, programming models, and dependency on libraries
  - C, C++, Fortran bindings, CUDA (Summit), HIP (Frontier), OpenCL (Aurora)
  - OpenMP, MPI
  - Dependency: BLAS and LAPACK
    - CPU: Accelerate, ARM Perf. Lib., ESSL, libSci, MKL, OpenBLAS
    - GPU: cuBLAS, hipBLAS, rocBLAS
MAGMA, PLASMA, SLATE: CPU & GPU WORK

- OpenMP features on CPU
  - Basic work sharing loops
  - Tasks and task loops
  - Data clauses: in, out, inout
  - Named critical regions

- Previous or current GPU work
  - MAGMA: CUDA, OpenCL (clMAGMA)
  - PLASMA: Offload targets: MAGMA backend (magmaBLAS) or OpenMP offload
  - SLATE: targets are CPU, CPU-nested, GPU, GPU-batched

- Motivation for current work
  - Summit: OpenMP + cuBLAS
  - Aurora: OpenCL + MKL + L0
  - Frontier: hipMAGMA + rocBLAS
PROBLEMS ENCOUNTERED

- Compilers
  - Older XL: unsupported named critical region across compilation units
  - Older GCC: no support for data-dependent tasks
    - Still widely available and installed by default
- Migration from other programming models
  - Abandoned in-house dataflow tasking runtime called QUARK
- Interoperability/libraries
  - The middle ground: must support both threading in lower-level BLAS/LAPACK and in upper-level applications
- Tools
  - Looking into using the OMPT layer for auto-generation of tracing layer
- Reproducibility support
LESSONS LEARNED AND NEXT STEPS

- Lessons learned
  - Using conservative subset of modern OpenMP helps us with portability

- Next steps with OpenMP 5
  - Memory allocators
  - Native kernels across vendor hardware and math libraries

- Portability testing across compilers
  - Apple LLVM
  - GCC 6+
  - IBM XL C/C++
  - Intel + MKL (iomp and gomp)
  - PGI
THANK YOU!

MARK GATES (SLATE)
MGATES3@ICL.UTK.EDU

PIOTR LUSZCZEK (PLASMA)
LUSZCZEK@ICL.UTK.EDU

STANIMIRE TOMOV (MAGMA)
TOMOV@ICL.UTK.EDU
THANKS!