Experiences Porting Mini-applications to OpenACC and OpenMP on Heterogeneous Systems

Verónica G. Vergara Larrea
Reuben D. Budiardja
Rahul Gayatri
Christopher Daley
Oscar Hernandez
Wayne Joubert
Outline

• Objective

• Application porting experiences
  – Minisweep
  – GenASIS
  – GPP
  – FF

• Conclusions

• Future work
Objectives

• Port four mini-applications to OpenACC and OpenMP 4.5 programming models.

• Evaluate the performance of individual versions on different architectures.
  – Titan, Summitdev, Summit, Cori, and Cori-GPU.

• Document challenges and issues encountered.
Target Systems

- **Titan**: Cray XK7
  - 18,688 compute nodes
  - 16-core AMD Opteron + 1 NVIDIA K20X GPU per node
- **Summit**: IBM AC922
  - 4,608 compute nodes
  - Two 22-core POWER9 + 6 NVIDIA V100 GPUs per node
- **Summitdev**: IBM P8+
  - 54 compute nodes
  - Two 10-core POWER8 + 4 NVIDIA P100 GPUs per node

- **Cori**: Cray XC40
  - 2,388 multi-core nodes + 9,688 many-core nodes
  - Dual-socket 16-core Intel Haswell per multi-core node
  - One 68-core Intel KNL per many-core node
- **Cori-GPU**: Cray CS-Storm
  - 18 nodes development system
  - Dual-socket 20-core Intel Xeon Gold ‘Skylake’ + 8 NVIDIA V100 GPUs per node
Minisweep

• Part of the Profugus radiation transport proxy application project.
  – Simulates the sweep pattern used in Denovo $S_n$ radiation transport application.
  – Used for nuclear reactor core analysis, nuclear forensics, radiation shielding, and radiation detection

• Original code developed at ORNL (W. Joubert).
  – Written in C with CUDA and OpenMP 3.1 support.

• OpenACC port available from University of Delaware (R. Searles et al.).
Minisweep compute kernels

- Sweep kernel accounts of >80% of the runtime in Denovo.
- Two main functions that are offloaded to the GPU:
  - `Sweep_sweep()`
  - `Sweep_in_gridcell()`
- Fairly straightforward transformations

<table>
<thead>
<tr>
<th>OpenACC</th>
<th>OpenMP 4.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc loop gang</td>
<td>omp teams distribute</td>
</tr>
<tr>
<td>acc loop vector</td>
<td>omp parallel for</td>
</tr>
<tr>
<td>acc loop seq</td>
<td>None</td>
</tr>
<tr>
<td>acc parallel</td>
<td>omp target</td>
</tr>
<tr>
<td>acc wait</td>
<td>omp taskwait</td>
</tr>
<tr>
<td>acc data copyout</td>
<td>omp data map(from:&lt;var&gt;)</td>
</tr>
<tr>
<td>acc data copyin</td>
<td>omp data map(to:&lt;var&gt;)</td>
</tr>
<tr>
<td>acc data create</td>
<td>omp data map(alloc:&lt;var&gt;)</td>
</tr>
</tbody>
</table>
Minisweep results (Lower is better)

NX x NY x NZ = 32 x 32 x 32; NM =16
GenASIS

• **General Astrophysical Simulation System**
• Object-oriented, modular design in modern Fortran (2003, 2008)
• Three major subdivisions → allow for unit testings, mini-apps development
  
  – **Basics**: Utilitarians functionalities
    • I/O, *StorageForm* class, Devices (OpenMP & CUDA library wrappers), MPI facades, …
    • Solvers & physics are implemented within test program

  – **Mathematics**: Object-oriented Manifolds, Operations, and Solvers
    • Meshing (distributed mesh, manifolds, charts), PDE solvers, Time integration
    • Physics are implemented within test program

  – **Physics**: Stress-Energy, Universes, Spaces
    • Newtonian, GR, Fluids, Radiation
    • Full applications use all three subdivisions
GenASIS: Target Problem

View movie at: https://tinyurl.com/yya2bqvw
Lower-Level GenASiS Functionality

- Fortran wrappers to OpenMP / OpenACC APIs
  - call AllocateDevice(Value, D_Value)
    → `omp_target_alloc()`, `acc_malloc()`
  
  call AssociateHost(D_Value, Value)
  → `omp_target_associate_ptr()`, `acc_map_data()`

  call UpdateDevice(Value, D_Value),
  call UpdateHost(Value, D_Value)
  → `omp_target_memcpy()`, `acc_memcpy_{to,from}_device()`

- Affirmative control of data movement
- Persistent memory allocation on the device
Higher-level GenASiS Functionality

- **StorageForm**: 
  - a class for data and metadata; the ‘heart’ of data storage facility in GenASiS
  - metadata includes units, variable names (for I/O, visualization)
  - used to group together a set of related physical variables (e.g. Fluid)
  - render more generic and simplified code for I/O, ghost exchange, prolongation & restriction (AMR mesh)

- **Data**: StorageForm % Value ( nCells, nVariables )

- **Methods**: 
  - call StorageForm % Initialize ( [shape] ) ← allocate data on host
  - call StorageForm % AllocateDevice ( ) ← allocate data on GPU
  - call StorageForm % Update{Device,Host} ( ) ← transfer data
Offloading Computational Kernel

1 subroutine AddKernel ( A, B, D_A, D_B, D_C, C )
2    real ( KDR ), dimension ( : ), intent ( in ) :: A, B
3    type ( c_ptr ), intent ( in ) :: D_A, D_B, D_C
4    real ( KDR ), dimension ( : ), intent ( out ) :: C
5
6    integer ( KDI ) :: i
7
8    call AssociateHost ( D_A, A )
9    call AssociateHost ( D_B, B )
10   call AssociateHost ( D_C, C )
11
12 !$OMP target teams distribute parallel do schedule ( static, 1 )
13    do i = 1, size ( C )
14        C ( i ) = A ( i ) + B ( i )
15    end do
16 !$OMP end target teams distribute parallel do
17    call DisassociateHost ( C )
18    call DisassociateHost ( B )
19    call DisassociateHost ( A )
20
21 call F % Initialize &
22 ( [nCells, nVariables] )
23 call F % AllocateDevice ( )
24 call F % UpdateDevice ( )
25 call AddKernel &
26 ( F % Value ( :, 1 ),
27   F % Value ( :, 2 ), &
28   F % D_Value ( 1 ),
29   F % D_Value ( 2 ), &
30   F % D_Value ( 3 ),
31   F % Value ( :, 3 ) )
32 end subroutine AddKernel
**Offloading Computational Kernel**

```fortran
1 subroutine AddKernel ( A, B, D_A, D_B, D_C, C )
2
3   real ( KDR ), dimension ( : ), intent ( in ) :: A, B
4   type ( c_ptr ), intent ( in ) :: D_A, D_B, D_C
5   real ( KDR ), dimension ( : ), intent ( out ) :: C
6
7   integer ( KDI ) :: i
8
9   call AssociateHost ( D_A, A )
10  call AssociateHost ( D_B, B )
11  call AssociateHost ( D_C, C )
12
13  !$OMP target teams distribute parallel do schedule ( static, 1 )
14  do i = 1, size ( C )
15      C ( i ) = A ( i ) + B ( i )
16  end do
17  !$OMP end target teams distribute parallel do
18
19  call DisassociateHost ( C )
20  call DisassociateHost ( B )
21  call DisassociateHost ( A )
22
23 end subroutine AddKernel
```

*call F % Initialize & AssociateHost (variables]*)

*call F % UpdateDevice ( )
call F % UpdateDevice ( )
call AddKernel &
( F % Value ( :, 1 ),
  F % Value ( :, 2 ), &
  F % D_Value ( 1 ),
  F % D_Value ( 2 ), &
  F % D_Value ( 3 ),
  F % Value ( :, 3 )
)

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

subroutine AddKernel ( A, B, D_A, D_B, D_C, C )
  real ( KDR ), dimension ( : ), intent ( in ) :: A, B
  type ( c_ptr ), intent ( in ) :: D_A, D_B, D_C
  real ( KDR ), dimension ( : ), intent ( out ) :: C

  !$OMP target teams distribute parallel do [collapse(n)] &
  !$OMP& schedule (static, 1)
  !$ACC loop gang vector [collapse(n)]

  !$OMP target teams distribute parallel do schedule ( static, 1 )
  do i = 1, size ( C )
    C ( i ) = A ( i ) + B ( i )
  end do
  !$OMP end target teams distribute parallel do

  call AddKernel &
  ( F % Value ( :, 1 ),
    F % Value ( :, 2 ), &
    F % D_Value ( 1 ),
    F % D_Value ( 2 ), &
    F % D_Value ( 3 ),
    F % Value ( :, 3 ) )

  call AddKernel &
  ( F % Value ( :, 1 ),
    F % Value ( :, 2 ), &
    F % D_Value ( 1 ),
    F % D_Value ( 2 ), &
    F % D_Value ( 3 ),
    F % Value ( :, 3 ) )

  !$OMP target teams distribute parallel do schedule ( static, 1 )
  do i = 1, size ( C )
    C ( i ) = A ( i ) + B ( i )
  end do
  !$OMP end target teams distribute parallel do

  call AddKernel &
  ( F % Value ( :, 1 ),
    F % Value ( :, 2 ), &
    F % D_Value ( 1 ),
    F % D_Value ( 2 ), &
    F % D_Value ( 3 ),
    F % Value ( :, 3 ) )

end subroutine AddKernel
GenASiS: Test Setup

- **Compilers:**
  - IBM XL (OpenMP)
  - GCC-8 (OpenMP, OpenACC from MentorGraphics)
  - Cray CCE (OpenMP, OpenACC)

- **Runtime:**
  - $128^3$ cells per MPI process, 1 MPI with OpenMP threads per GPU
  - Summit: 7 CPU threads vs. 1 GPU
  - Titan: 8 CPU threads vs 1 GPU
  - Cori-GPU: 8 CPU threads vs 1 GPU
GenASiS: CPU Timings (Lower is Better)
GenASiS: GPU Timings (Lower is Better)
GenASiS: Speed-up / Slow-down Relative to Multithreaded CPU
GPP

- Mini-application from the BerkeleyGW suite written in C++.
- Computes electron self-energy using the General Plasmon Pole (GPP) approximation.
- Kernel is comprised of 4-nested loops with a reduction in the innermost loop.
GPP Results

GPP implementations

- OpenMP(Haswell)
- OpenMP(KNL)
- OpenACC(V100)
- OpenMP(V100)

Time [secs]
FF

• Also part of the BerkeleyGW suite and written in C++.

• Represents the Full-Frequency (FF) Self-Energy Summations.

• Comprised of 3 kernels with different loop structures.
FF results

FF implementations

Time [secs]

- OpenMP 3.0(Haswell)
- OpenMP 3.0(Xeon-Phi)
- OpenMP 4.5(V100)
- OpenACC(V100)

achsDtemp, asxDtemp, achDtemp-cor
Summary and Conclusion

- We shared experiences to port (mini)-applications to OpenMP / OpenACC

- Performance portability is not observed
  - Different level of maturity for compilers
  - Some workarounds required

- Code available for further experimentation
  - GenASiS: https://github.com/GenASiS/GenASiS_Basics/releases
  - Minisweep: https://github.com/olcf/minisweep
  - GPP/FF: https://gitlab.com/rgayatri/berkeleygw-kernels
Future work

- Perform in-depth analysis for cases that exhibit performance degradation.
- Identify root cause for remaining issues encountered in each code.
- Report bugs to the different compilers.
- Perform multi-node/large scale experiments.
Acknowledgements

• 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.
• This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.