Using OpenMP to Harness GPUs for Core-Collapse Supernova Simulations with GenASiS

OpenMP ARB Webinar
February 17, 2022

Reuben D. Budiardja
Computational Scientist
Oak Ridge Leadership Computing Facility
Oak Ridge National Laboratory
Core-Collapse Supernovae (CCSN)

- The death throes of massive star (\(M > \sim 10\) Solar M)
  - The birth of neutron stars and black holes
- Among the most powerful explosions in the universe
  - \(\sim 10^{53}\) ergs of energy released as mostly neutrino
  - \(\sim 10^{51}\) ergs visible electromagnetic radiation
    \(\sim 10^{28}\) megatonnes of TNT
- Observables:
  - Gamma-ray burst, gravitational wave, neutrino
- Occur about twice per century in our galaxy

Cassiopeia A Supernova Remnant (Chandra Observatory)
Why Simulate Supernova?

Answers to questions relating to our origins in the universe.
The Path to Explosion

Supernova 1987A
The Path to Explosion
Understanding detail explosion mechanism requires high-fidelity simulations.
Supernova: Multi-Physics & Multi-Scale Problem

- General relativistic gravity
- Compressible Fluid
- Magnetic fields
- Convection
- Nuclear kinetic
- Dense equation of state
- Radiation transport

Daunting software integration tasks

- Collapse: \( \sim 10^6 \) density increase
- Proto-neutron star: < ~1 km scale
- Outer layer: \( \sim 10^3 \) km
- 3D → > \( 10^9 \) cells
- \( 10^{-6} \)s timestep to 1s total time
- Turbulent cascade
- Magnetorotational instability

Pushing boundaries of computational hardware and software

Compute power requirements
**GenASiS**

- **General Astrophysics Simulation System**
  - Current target: 3D position space + 1D momentum space the simulations of core-collapse supernovae; Towards 3D + 3D (sustained exascale)
  - Earlier versions have been used to study of fluid instabilities in supernova dynamics, discover exponential magnetic field amplification in progenitor star

- **Code characteristics:**
  - Modern Fortran (mostly F2008, some F2018)
  - Modular, object-oriented design, extensible
  - OpenMP for threading + offloading
GenASiS Structure

Basics: Utilitarian infrastructure, data management, I/O, Units, Devices, MPI facades

Mathematics: manifolds / meshes (cartesian, curvilinear), solvers (time-dependent ODE, PDE, elliptic)

Physics: fluid type, equation of states, stress-energy tensors, space-type (newtonian, relativistic, …)
GenASiS Storage 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 )
  - use as, e.g. Pressure => StorageForm % Value ( :, 1 ), Density => StorageForm % Value ( :, 2 )

- **Methods**: 
  - call S % Initialize ( ) ← allocate data on host
  - call S % AllocateDevice ( ) ← allocate and associate data on GPU
  - call S % Update{Device,Host} ( ) ← transfer data
GPU Data Allocation and Mapping

- call StorageForm % Initialize &
  ( Shape = [6, 6], &
  VariableOption = [“Pressure”, “Density”, &
  “Energy”, ... ] )
Two ways to associate (“map” in OpenMP) the host and device memory:

1. One mapping for each variable (column-wise) → $n\text{Variables}$ mapping
2. One mapping for the whole (contiguous) block

Which mapping to use to use depends on how a kernel is written (more on this later).
real ( KDR ), dimension ( :, : ), pointer :: Scratch

type ( c_ptr ) :: D_Value

call AllocateDevice ( S % nValues * S % nVariables, D_Value )
call c_f_pointer ( D_Value, Scratch, [ S % nValues, S % nVariables ] )
do iV = 1, S % nVariables
  D_Value = c_loc ( Scratch ( :, iV ) )
  Variable => S % Value ( :, iV )
call AssociateHost ( D_Value, Variable )
end do

Tells OpenMP data location on GPU
→ avoid (implicit) allocation & transfer
real ( KDR ), dimension ( :, : ), pointer :: Scratch

Tells OpenMP data location on GPU →avoid (implicit) allocation & transfer

call AllocateDevice ( S % nValues * S % nVariables, D_Value )

call AssociateHost ( D_Value, S % Value )
One can go back-and-forth between the two ways of mapping with:

```plaintext
call S % ReassociateHost ( AssociateVariables = [true, false] )
```
Lower-Level GenASiS Functionality

• Fortran wrappers to OpenMP APIs

  - call AllocateDevice(Value, D_Value) → omp_target Alloc()
    call AssociateHost(D_Value, Value) → omp_target associate_ptr()
    call UpdateDevice(Value, D_Value),
    call UpdateHost(Value, D_Value) → omp_target memcpy()
    call ReassociateHost (...)
      → omp_target disassociate_ptr (...), ...
      omp_target associate_ptr (h_ptr, d_ptr, totalSize, ...)
Offloading Computational Kernel

Persistent allocation and association

1 subroutine AddKernel ( A, B, C )
2
3 real ( KDR ), dimension ( : ), intent ( in ) :: A, B
4 real ( KDR ), dimension ( : ), intent ( out ) :: C
5
6 integer ( KDI ) :: i
7
8 !$OMP target teams distribute parallel do
9 do i = 1, size ( C )
10 C ( i ) = A ( i ) + B ( i )
11 end do
12 !$OMP end target teams distribute parallel do
13
14 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()
Offloading Computational Kernel

!-- F1, F2, F3 are StorageForm object

call [F1, F2, F3] &
    % ReassociateHost ( AssociateVariablesOption = .false. )
call SumVariableKernel (F1, F2, F3 )

```fortran
subroutine SumVariableKernel ( V_1, V_2, V_3 )
    real ( KDR ), dimension ( :, : ), intent ( inout ) :: V_1
    real ( KDR ), dimension ( :, : ), intent ( in ) :: V_2, V_3
    integer ( KDI ) :: iV, iS

    !$OMP target teams distribute parallel do collapse ( 2 )
    do iS = 1, size ( V_1, dim = 1 )
        do iV = 1, size ( V_1, dim = 2 )
            V_1 ( iS, iV ) = V_2 ( iS, iV ) + V_3 ( iS, iV )
        end do
    end do

end subroutine SumVariableKernel
```
Example of Kernel with Pointer Remapping

```fortran
1 subroutine ComputeDifferences_X ( V, dV )
2
3   real ( KDR ), dimension ( -1:, -1:, -1: ), &
4       intent ( in ) :: &
5
6   V
7   real ( KDR ), dimension ( -1:, -1:, -1: ), &
8       intent ( out ) :: &
9   dV
10
11   integer ( KDI ) :: i, j, k
12
13   !$OMP target teams distribute parallel do collapse ( 3 ) schedule ( static, 1 )
14   do k = 1, nZ
15     do j = 1, nY
16       do i = 0, nX + 2
17         dV ( i, j, k ) &
18           = V ( i, j, k ) &
19             - V ( i - 1, j, k )
20       end do
21     end do
22   end do
23   !$OMP end target teams distribute parallel do
24
25 end subroutine ComputeDifferences_X
```

**Caller, using pointer remapping feature:**

```fortran
real ( KDR ), dimension ( :, :, : ), &
pointers :: V, dV

V ( -1:nX+2, -1:nY+2, -1:nZ+2 ) &
=> F % Value ( : , iV )

dV ( -1:nX+2 , -1:nY+2 , -1:nZ+2 ) &
=> dF % Value ( : , iV )

call ComputeDifferences_X ( V, dV )
```
Pointer Remapping + ReassociateHost ( )

$$\text{call } \text{S} \% \text{ReassociateHost ( .false. )}$$
$$\text{call } \text{Flux}_I \% \text{ReassociateHost ( .false. )}$$

$$\text{S ( -1:nX+2, -1:nY+2, -1:nZ+2, 1:nF ) \&}$$
$$\Rightarrow \text{S % Value ( :, 1:nF )}$$
$$\text{F}_I ( -1:nX+2, -1:nY+2, -1:nZ+2, 1:nF ) \&$$
$$\Rightarrow \text{Flux}_I \% \text{Value ( :, 1:nF )}$$
$$\text{A}_I ( -1:nX+2, -1:nY+2, -1:nZ+2 ) \&$$
$$\Rightarrow \text{G % Value ( :, AREA_INNER )}$$
$$\text{V ( -1:nX+2, -1:nY+2, -1:nZ+2 ) \&}$$
$$\Rightarrow \text{G % Value ( :, VOLUME )}$$

!$OMP target teams distribute parallel do collapse ( 4 ) &
!$OMP private ( iaVP )
do is = 1, nS
  do kv = lv ( 3 ), uV ( 3 )
    do jv = lv ( 2 ), uV ( 2 )
      do iv = lv ( 1 ), uV ( 1 )

        iaVP = [ iv, jv, kv ] + iaS

        S ( iv, jv, kv, is ) &
        = S ( iv, jv, kv, iS ) &
          - ( A_I ( iaVP ( 1 ), iaVP ( 2 ), iaVP ( 3 ) ) &
              * F_I ( iaVP ( 1 ), iaVP ( 2 ), iaVP ( 3 ), iS ) &
                - A_I ( iv, jv, kv ) &
                * F_I ( iv, jv, kv, iS ) ) ) &
          / V ( iv, jv, kv )

      end do !-- iv
  end do !-- jv
end do !-- is
GenASiS Structure

https://github.com/GenASiS

Basics RiemannProblem:

- An example problem using only Basics classes with simplified fluid solvers representative of higher-level solvers in (Mathematics, Physics)

- Useful for testings & experimentations with only a handful of computational kernels and simplified mesh (distributed cartesian)
Basics Riemann Problem

- A benchmark problem for hydrodynamics solvers.
- Similar solvers are used for the explicit part of radiation transport in an IMEX scheme.
- Workhorse proxy-application for experimentation and testing, including (currently):
  - metadirectives
  - requires unified_shared_memory
  - new platforms / compilers
- Kernels for this problem have also been ported to CUDA / HIP for comparisons.

Initial (left) and final (right) density of 1D and 3D Riemann Problem
Weak Scaling & Speedups

~12-15X speedup from 7 CPU threads to GPU
GenASiS Basics: RiemannProblem 3D, 256^3 per GPU, 1 process, 50 cycles

Kernel and data transfer timings: lower is better

- Summit XL OpenMP V100
- Summit CUDA V100
- Spock CCE OpenMP MI100
- Spock ROCm HIP MI100

Graph showing walltime (s) for various kernels:
GenASiS Basics: RiemannProblem 3D, 256^3 per GPU, 1 process, 50 cycles

Kernel and data transfer timings: lower is better

- Summit XL OpenMP V100
- Summit CUDA V100
- Spock CCE OpenMP MI100
- Spock ROCm HIP MI100

Walltime (s)

ComputeTimeStep  RK Step  Update  Difference  Reconstruction  RiemannSolverEq...  RawFluxes  Fluxes  ComputePrimitive  ComputeConserv...  ComputeAuxiliary  ApplyBoundaryCond...  HostToDevice  DeviceToHost
Ongoing Work: Metadirective

- Example use case: evolve fluid on host + radiation on GPU + load balancing
- Currently code duplication needed for compiler to generate both device and host versions.
- Better: metadirective with user-selector and dynamic condition

```c
if ( UseDevice ) then
    !$OMP target teams distribute parallel do &
    !$OMP private ( KE )
    do iV = 1, size ( E )
        [kernel code ...]
    end do
    !$OMP end target teams distribute parallel do
else
    !$OMP parallel do &
    !$OMP private ( KE )
    do iV = 1, size ( E )
        [duplicated kernel code ...]
    end do
    !$OMP end parallel do
end if
```

Awaiting full compiler support for metadirective.
Ongoing Work: Remove Compatibility Interfaces

• Currently uses [cuda/hip]HostMalloc() to allocate page-locked (pinned) memory
  – Replacing to use OpenMP 5 allocator + allocate directive
  – Pre-defined (extension) allocator **OMPX_PINNED_MEM_ALLOC** is available in some compilers. Needs consensus & standardization.

• In OpenMP 4.5 some library routines only have C binding. Replacing these as they become available from compilers supporting OpenMP 5.x
  – For e.g. omp_target_alloc(), omp_target_[associate/disassociate]_ptr()
Ongoing Work: Optimizations

• What is the best scheduling policy (static, X), guided, auto, runtime)? This could be compiler dependent.

• Best mapping to hardware (teams, parallel do, simd). Compilers map to hardware differently.

• More work per target region vs. multiple (concurrent) target launch (with nowait).
  – could be compiler dependent
  – could be kernel dependent
  – could be hardware dependent

• Exploiting GPU “shared memory” via allocate clause in target
Stochastic Behavior of SASI-dominated Models
Ongoing Science Investigation

• Using OpenMP offload to exploit GPU, we now have the computational capability to perform ensemble CC-SN simulations with more realistic setups, including with radiation transport.

• Bridging the gap of two traditional classes of CC-SN sims:
  – A handful of cutting edge sophistication, vs.
  – Large ensembles of simplified phenomenological simulations
  – Broad survey with intermediate sophistication to target scrutiny of selected region with increased sophistication.
Thank You

Reuben D. Budiardja, reubendb@ornl.gov