

OpenMP experiences with thornado

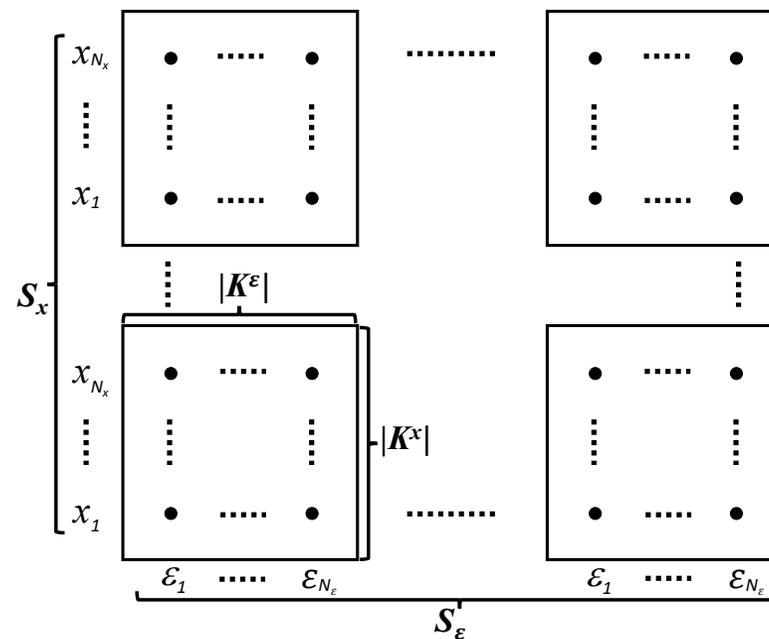
J. Austin Harris (ORNL)
M. Paul Laiu (ORNL)
Ran Chu (UTK)
Eirik Endeve (ORNL)

ORNL is managed by UT-Battelle LLC for the US Department of Energy

thornado

(toolkit for high-order neutrino rad-hydroo)

- ExaStar proxy application for spectral neutrino transport in stellar astrophysics simulations
 - Designed to be incorporated into AMR-based codes (e.g., Flash-X)
 - Focus on single MPI rank+GPU performance
- Tabulated nuclear equation of state and neutrino opacity tables from WeakLib
- GPU port of original CPU code via compiler directives and linear algebra libraries
- Discontinuous Galerkin (DG) method for phase-space discretization
 - 1 AMR block $\cong 8^3$ spatial elements = 4096 nodes
 - 16 energy elements = 32 nodes



thornado time integration

- Positive, diffusion accurate, two-stage implicit-explicit (PD-IMEX) time integration
 - Explicit neutrino advection operator
 - Implicit neutrino collision operator
 - Nonlinear solver
 - Tabulated microphysics
 - Numerical limiters to maintain realizability and conservation

Enter from Flash-X

1. **Explicit Update (Advection)**
2. **Apply Limiters**
3. **Implicit Update (Collisions)**
4. **Apply Limiters**
5. **Explicit Update (Advection)**
6. **Apply Limiters**
7. **Implicit Update (Collisions)**
8. **Apply Limiters**

Exit to Flash-X

OpenMP features

- Ubiquitous use of:
 - `target teams distribute parallel do simd collapse`
 - `target enter data, target exit data`
 - `target data use_device_ptr`
 - `declare target`
- In the future:
 - `metadirective` (for conditional GPU execution at runtime)
 - `interop` (for asynchronous OpenMP and linear algebra interoperability)
 - `collapse` of non-rectangular loop nest (for triangular loops)

GPU Strategy

Compiler Directives

```
#if defined(THORNADO.OMP.OL)
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(7)
#elif defined(THORNADO.OACC)
!$ACC PARALLEL LOOP GANG VECTOR COLLAPSE(7)
#elif defined(THORNADO.OMP)
!$OMP PARALLEL DO SIMD COLLAPSE(7)
#endif
#endef
DO iZ4 = iZB4-1, iZE4+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(WEAKLIB.OMP.OL)
!$OMP TARGET TEAMS DISTRIBUTE IF( do-gpu ) PRIVATE( iT, dT, iX, dX )
#elif defined(WEAKLIB.OACC)
!$ACC PARALLEL LOOP GANG ASYNC( async-flag ) IF( do-gpu ) PRIVATE( iT, dT, iX, dX )
#endif
#endef
DO k = 1, SIZE( LogT )
  iT = ... ; dT = ...
  iX = ... ; dX = ...
#if defined(WEAKLIB.OMP.OL)
!$OMP PARALLEL DO SIMD PRIVATE( i0, j0, i, j )
#elif defined(WEAKLIB.OACC)
!$ACC LOOP VECTOR PRIVATE( i0, j0, i, j )
#endif
#endef
DO ij = 1, SizeE*(SizeE+1)/2 ! Fused triangular loop: DO j = 1, SizeE ; DO i = 1, j
  j0 = MOD( (ij-1) / ( SizeE + 1 ), SizeE + 1 ) + 1
  i0 = MOD( (ij-1) , SizeE + 1 ) + 1
  IF ( i0 > j0 ) 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, iX ), Table(i, j, iT+1, iX ), &
      Table(i, j, iT, iX+1), Table(i, j, iT+1, iX+1), dT, dX )
END DO
END DO
```

Linear Algebra Libraries

```
SUBROUTINE MatrixMatrixMultiply (... )
...
#if defined(THORNADO.OMP.OL)
!$OMP TARGET DATA USE_DEVICE_PTR( pa, pb, pc )
#elif defined(THORNADO.OACC)
!$ACC HOST.DATA USE_DEVICE( pa, pb, pc )
#endif
da = CLOC( pa ) ; db = CLOC( pb ) ; dc = CLOC( pc )
#if defined(THORNADO.OMP.OL)
!$OMP END TARGET DATA
#elif defined(THORNADO.OACC)
!$ACC END HOST.DATA
#endif
#if defined(THORNADO.LA.CUBLAS)
ierr = cublasDgemv.v2( ... , da, lda, ... )
#elif defined(THORNADO.LA.MAGMA)
CALL magma_dgemm( ... , da, lda, ... )
#else
CALL DGEMM( ... , a, lda, ... )
#endif
END SUBROUTINE

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

...
CALL MatrixMatrixMultiply &
  ( 'N', 'N', nDOF_X3, nF, nDOF, One, L-X3-Up, nDOF_X3, &
    uCRK(1, iZB1, iZB2, iZB3, 1, 1, iZB4-1), nDOF, Zero, &
    uCR.L(1, iZB1, iZB2, iZB3, 1, 1, iZB4 ), nDOF_X3 )
```

Example #1

LogInterpolateSingleVariable_2D2D_Custom_Aligned

- ~40% of runtime in this kernel
- 2D interp. of opacity table for each spatial point and (E1,E2) pair
- Called multiple times per nonlinear solver iteration in implicit step

```
!$OMP TARGET TEAMS DISTRIBUTE PRIVATE( iT, dT, iX, dX )
DO k = 1, SIZE( LogT ) ! Loop over every spatial point
CALL GetIndexAndDelta_Lin( LogT(k), LogTs, iT, dT )
CALL GetIndexAndDelta_Lin( LogX(k), LogXs, iX, dX )
!$OMP PARALLEL DO SIMD PRIVATE( i0, j0, i, j )
DO ij = 1, SizeE*(SizeE+1)/2 ! Collapsed triangular loop over energy
  j0 = MOD( (ij-1) / ( SizeE + 1 ), SizeE + 1 ) + 1
  i0 = MOD( (ij-1) , SizeE + 1 ) + 1
  IF ( i0 > j0 ) THEN
    j = SizeE - j0 + 1 ; i = SizeE - i0 + 2
  ELSE
    j = j0 ; i = i0
  END IF
  Interpolant(i,j,k) = LinearInterp2D_4DArray_2DAligned_Point &
    (i,j,iT,iX,dT,dX,OS,Table)
END DO
END DO

FUNCTION LinearInterp2D_4DArray_2DAligned_Point &
( iX1, iX2, iY1, iY2, dY1, dY2, OS, Table) RESULT(Interpolant)
!$OMP DECLARE TARGET
INTEGER, INTENT(in) :: iX1, iX2, iY1, iY2
REAL(dp), INTENT(in) :: dY1, dY2, OS, Table(:, :, :, :)
REAL(dp) :: p00, p10, p01, p11, Interpolant
p00 = Table(iX1,iX2,iY1,iY2 ) ; p10 = Table(iX1,iX2,iY1+1,iY2 )
p01 = Table(iX1,iX2,iY1,iY2+1) ; p11 = Table(iX1,iX2,iY1+1,iY2+1)
Interpolant = 10.0d0** (Bilinear(p00,p10,p01,p11,dY1,dY2)) - OS
END FUNCTION LinearInterp2D_4DArray_2DAligned_Point

REAL(dp) FUNCTION Bilinear( p00, p10, p01, p11, dX1, dX2 )
!$OMP DECLARE TARGET
REAL(dp), INTENT(in) :: p00, p10, p01, p11, dX1, dX2
REAL(dp) :: ddX1, ddX2
ddX1 = One - dX1 ; ddX2 = One - dX2
Bilinear = ddX1*(ddX2*p00+dX2*p01) + dX1*(ddX2*p10+dX2*p11)
END FUNCTION Bilinear
```

Example #2

ApplyPositivityLimiter_TwoMoment

- ~20% of runtime in this routine
- Check that each point is a realizable physical solution
- Called 5 times per call to thornado
- Contains bisection iteration
- Very low arithmetic intensity

```
CALL ComputePointValuesZ(iZ_B0,iZ_E0,U_Q_N,U_P_N) ! 4 short & wide DGEMMs
CALL ComputeCellAverage(iZ_B0,iZ_E0,Tau_Q,U_Q_N,U_K_N) ! called 4 times
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(5) &
!$OMP PRIVATE( Gam, Min_Gam, Theta_2, Theta_P, iP_X ) &
!$OMP REDUCTION( min: MinTheta_2 )
DO iS=1,nS; DO iZ4=ZB4,ZE4; DO iZ3=ZB3,ZE3; DO iZ2=ZB2,ZE2; DO iZ1=ZB1,ZE1
Min_Gam = Min_2 ; Theta_2 = One
  DO iP_Z = 1, nPT_Z
    iP_X = PointZ2X(iP_Z)
    Gam = GammaFun(<some scalar inputs>)
    Min_Gam = MIN( Min_Gam, Gam )
    IF ( Gam < Min_2 ) THEN
      ! Use bisection to find zero for f(x) = GammaFun(x)-1
      CALL SolveTheta_Bisection(<some scalar inputs>, Theta_P, ...)
      Theta_2 = MIN( Theta_2, Theta_P )
    END IF
  END DO
  IF ( Min_Gam < Min_2 ) THEN
    U_Q_N(:,...) = U_Q_N(:,...)*Theta_2 + U_K_N(...)*(One-Theta_2)
    MinTheta_2 = MIN( Theta_2, MinTheta_2 )
  END IF
END DO ; END DO ; END DO ; END DO ; END DO

SUBROUTINE ComputeCellAverage( iZ_B0, iZ_E0, Tau_Q, U_Q, U_K )
INTEGER, INTENT(in) :: iZ_B0(4), iZ_E0(4)
REAL(DP), INTENT(in) :: Tau_Q(:, :, :, :, :), U_Q(:, :, :, :, :)
REAL(DP), INTENT(out) :: U_K(:, :, :, :, :)
INTEGER :: iZ1, iZ2, iZ3, iZ4, iS
REAL(DP) :: S1, S2
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(5) PRIVATE(S1,S2)
DO iS=1,nS; DO iZ4=ZB4,ZE4; DO iZ3=ZB3,ZE3; DO iZ2=ZB2,ZE2; DO iZ1=ZB1,ZE1
  S1 = SUM( Weights_q*Tau_Q(:,iZ1,iZ2,iZ3,iZ4)*U_Q(:,iZ1,iZ2,iZ3,iZ4,iS) )
  S2 = SUM( Weights_q*Tau_Q(:,iZ1,iZ2,iZ3,iZ4) )
  U_K(iZ1,iZ2,iZ3,iZ4,iS) = S1/S2
END DO ; END DO ; END DO ; END DO ; END DO
END SUBROUTINE ComputeCellAverage
```

Example #3

ComputeIncrement_Divergence_X{1,2,3}

- ~10% of runtime in these routines
- Computes IMEX explicit update
- Called twice per call to thornado
- Healthy mix of linear algebra libraries and compiler directives
- Arrays permuted for better striding

```
! Calculate X1 Flux
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(6) &
!$OMP PRIVATE( EF, FF, uPR_K, Tau, Flux_X1_K )
DO iZ2=ZB2,ZE2; DO iS=1,nS; DO iZ4=ZB4,ZE4; DO iZ3=ZB3,ZE3; DO iZ1=ZB1,ZE1
DO iNode=1,nDOF
  uPR_K(:) = uCR_K(iNode,iZ1,iZ3,iZ4,:,iS,iZ2)
  FF = FluxFactor( uPR_K, G_K )
  EF = EddingtonFactor( uPR_K(iPR_D), FF )
  Flux_X1_K(:) = Flux_X1( uPR_K, FF, EF, G_K )
  DO iCR = 1, nCR
    Tau = One * G_K(iNode,iGF_SqrtGm,iZ3,iZ4,iZ2)
    Flux_X1_q(iNode,iCR,iZ1,iZ3,iZ4,iS,iZ2) &
      = dZ3*dZ4*Tau*Weights_q(iNode)*Flux_X1_K(iCR)*G_K(iNode)
  END DO
END DO
END DO ; END DO ; END DO ; END DO ; END DO

! nDOF x nK ~ 16 x 200k
CALL MatrixMatrixMultiply &
  ('T', 'N', nDOF, nK, nDOF, One, dLdX1_q, nDOF, Flux_X1_q, nDOF, One, dU_X1, nDOF)

! Add X1 contribution
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(7)
DO iS = 1, nS ; DO iCR = 1, nCR
DO iZ4=ZB4,ZE4; DO iZ3=ZB3,ZE3; DO iZ2=ZB2,ZE2; DO iZ1=ZB1,ZE1
DO iNode = 1, nDOF
  dU(iNode,iZ1,iZ2,iZ3,iZ4,iCR,iS) &
    = dU(iNode,iZ1,iZ2,iZ3,iZ4,iCR,iS) &
      + dU_X1(iNode,iCR,iZ1,iZ3,iZ4,iS,iZ2)
END DO
END DO ; END DO ; END DO ; END DO
END DO ; END DO
```

Example #4

SolveMatterEquations_FP_Coupled

- High level driver routine for non-linear solver
- Composed of many kernels, including opacity interpolations (Example #1)
- Called twice per call to thornado
- Only need to calculate for non-converged points:
 - With MASK array
 - Packing/unpacking (scatter/gather)

```
! from ComputeNumberDensity_FP
!$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD COLLAPSE(2) &
!$OMP PRIVATE( Eta, Eta_T, Chi_T )
DO iN_X = 1, nX_G ; DO iN_E = 1, nE_G
  IF ( MASK(iN_X) ) THEN ! use MASK to skip converged points
    Eta = Chi_1(iN_E,iN_X) * J0_1(iN_E,iN_X)
    Eta_T = Eta + Eta_NES_1(iN_E,iN_X) + Eta_Pair_1(iN_E,iN_X)
    Chi_T = Chi_1(iN_E,iN_X) + Chi_NES_1(iN_E,iN_X) + Chi_Pair_1(iN_E,iN_X)
    Jnew_1(iN_E,iN_X) = ( Jold_1(iN_E,iN_X) + dt*Eta_T ) / ( One + dt*Chi_T )

    Eta = Chi_2(iN_E,iN_X) * J0_2(iN_E,iN_X)
    Eta_T = Eta + Eta_NES_2(iN_E,iN_X) + Eta_Pair_2(iN_E,iN_X)
    Chi_T = Chi_2(iN_E,iN_X) + Chi_NES_2(iN_E,iN_X) + Chi_Pair_2(iN_E,iN_X)
    Jnew_2(iN_E,iN_X) = ( Jold_2(iN_E,iN_X) + dt*Eta_T ) / ( One + dt*Chi_T )
  END IF
END DO ; END DO
```

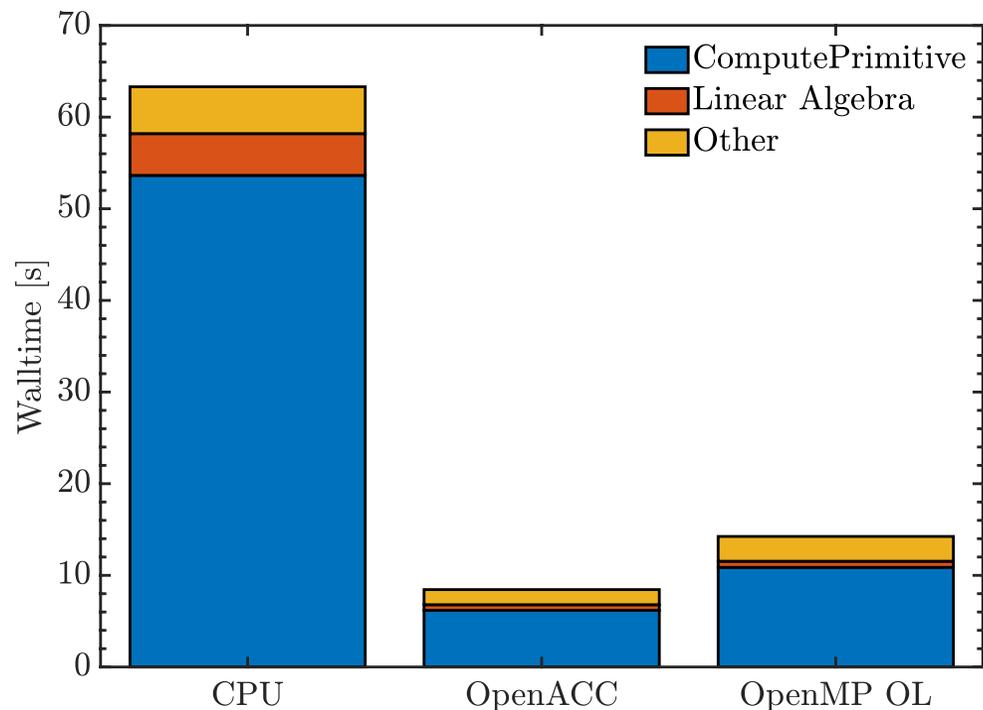
```
! from ComputeOpacities_Packed
D_P => P1D(1:nX,iP1D_D)
CALL ArrayPack( nX, UnpackIndex, D, D_P )
J0_1_P => P2D(:,1:nX,iP2D_J0_1)
CALL ComputeEquilibriumDistributions(...,D_P,...,J0_1_P,...)
CALL ArrayUnpack( nX, MASK, PackIndex, J0_1_P, J0_1 )
CALL ArrayUnpack( nX, MASK, PackIndex, J0_1_P, J0_1 )
```

```
SUBROUTINE ArrayPack1D_1( nP, UnpackIndex, X1, X1_P )
  INTEGER, INTENT(in) :: nP, UnpackIndex(1:)
  REAL(DP), INTENT(in) :: X1(1:)
  REAL(DP), INTENT(inout) :: X1_P(1:)
  INTEGER :: iPack
  IF ( nP < SIZE(X1,1) ) THEN
    !$OMP TARGET TEAMS DISTRIBUTE PARALLEL DO SIMD PRIVATE(i)
    DO iPack = 1, nP
      X1_P(iPack) = X1(UnpackIndex(iPack))
    END DO
  ELSE
    CALL ArrayCopy( X1, X1_P )
  END IF
END SUBROUTINE ArrayPack1D_1
```

Compiler status

- Can thornado compile and run with _____
 - NVIDIA HPC SDK (Summit/Ascent/CoriGPU)?
 - **Yes** (Preferred development compiler for OpenMP and OpenACC)
 - IBM XL(Summit/Ascent)?
 - **Yes** (but very long compile times)
 - Intel oneAPI (Iris/Arcticus)?
 - **Somewhat** (internal compiler errors); collaboration with Intel and ANL COE
 - Can compile and test stand-alone kernels
 - CCE (Crusher/Birch)?
 - **Somewhat** (internal compiler errors)
 - Can compile and run small number of tests

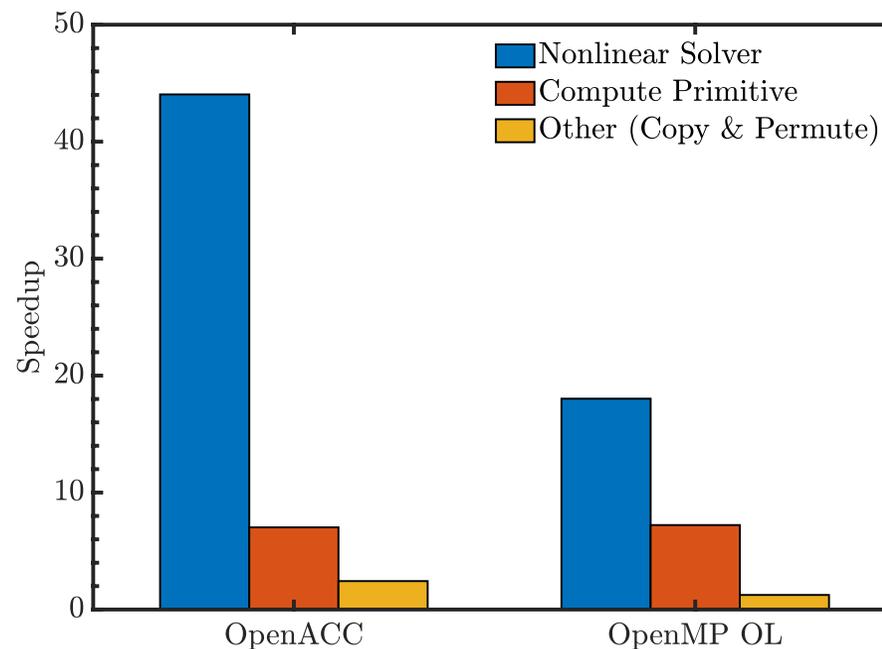
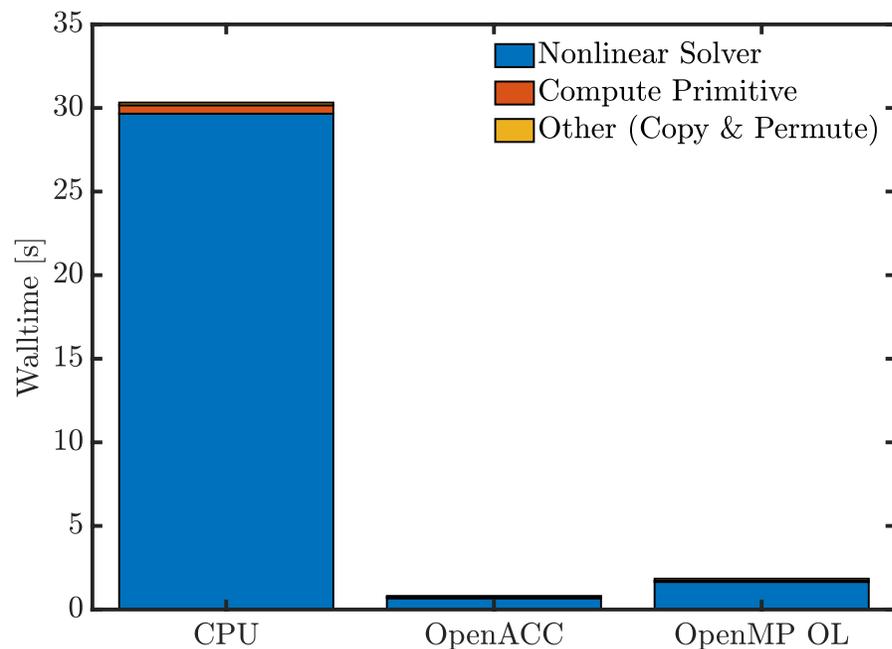
Streaming Sine Wave Test Problem



- Verification problem to test explicit operator
 - Run on single V100 and 7 P9 cores
- Evolve one “block” with **thornado**
 - Energy discretization: 32 points
 - Spatial discretization: 8^3 points
- Performance data from Summit
 - PGI 19.9 (CPU and OpenACC)
 - XL 16.1.1-5 (OpenMP OL)
- ~7x speedup relative to threaded CPU code

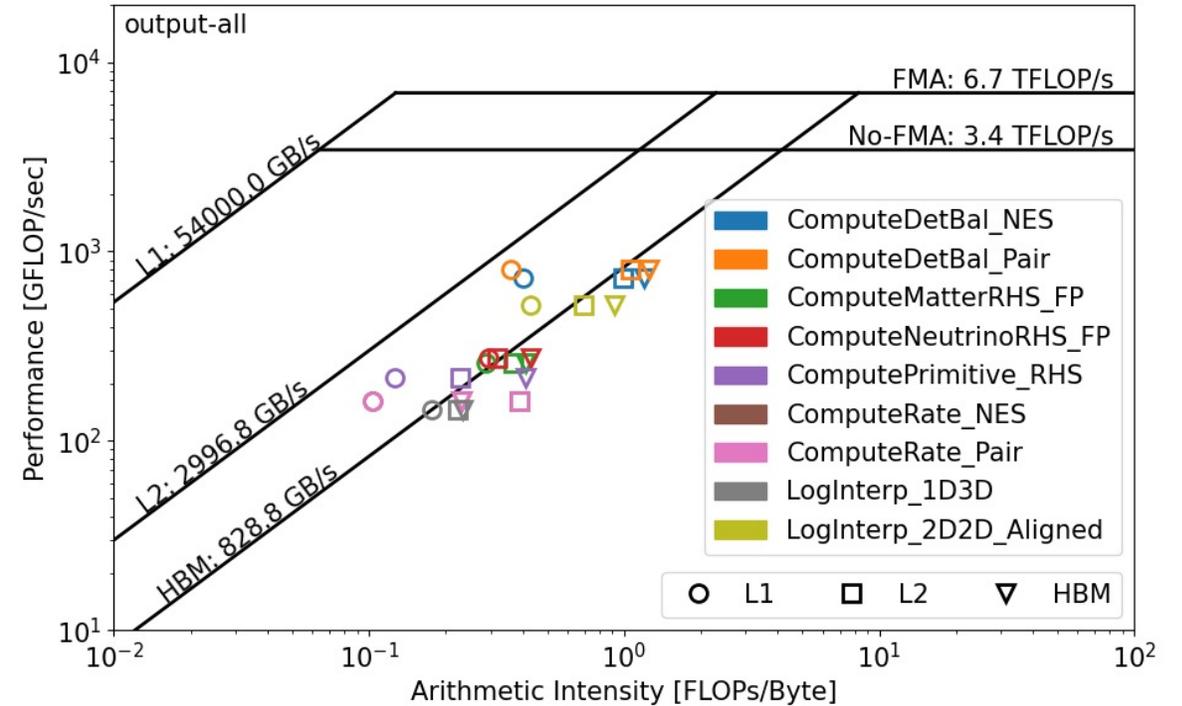
Relaxation Test Problem

- Verification problem to test implicit operator
 - Run on single V100 and 7 P9 cores
- Evolve one “block” with **thornado**
 - Energy discretization: 32 points
 - Spatial discretization: 8^3 points
- Performance data from Summit
 - NVHPC 21.3 (CPU and OpenACC)
 - XL 16.1.1-10 (OpenMP OL)
- ~20x speedup relative to threaded CPU cod



V100 Roofline Analysis

- Generated hierarchical roofline model of bottleneck kernels in O(v/c) Relaxation benchmark
- Key takeaways
 - thornado is memory (HBM) bound
 - Most kernels are near the roofline (good!)
 - L1 and L2 cache reuse is so-so
 - Need to think about how to increase arithmetic intensity (e.g., change vector length?)
- Need analysis on other systems for comparison



Final Thoughts

- Write “portable” CPU code (e.g., tight loop nests, libraries when possible)
 - Will also be faster for CPU
- Challenging to port iteration kernels due to register pressure (e.g., Newton-Raphson, Bisection)
 - Need to pack points and break up iteration into separate, smaller kernels
- Managing asynchronous kernels between libraries and directives is a challenge
- Need more compilers with **robust** Fortran+OpenMP 5.x support (worry about performance later)
- Looking forward to more comparisons with upcoming systems