OAK RIDGE National Laboratory

KokkACC: Enhancing Kokkos with OpenACC

PhD. Pedro Valero-Lara,

Computer Scientist at Programming Systems Group
valerolarap@ornl.gov

OpenACC Webinar

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

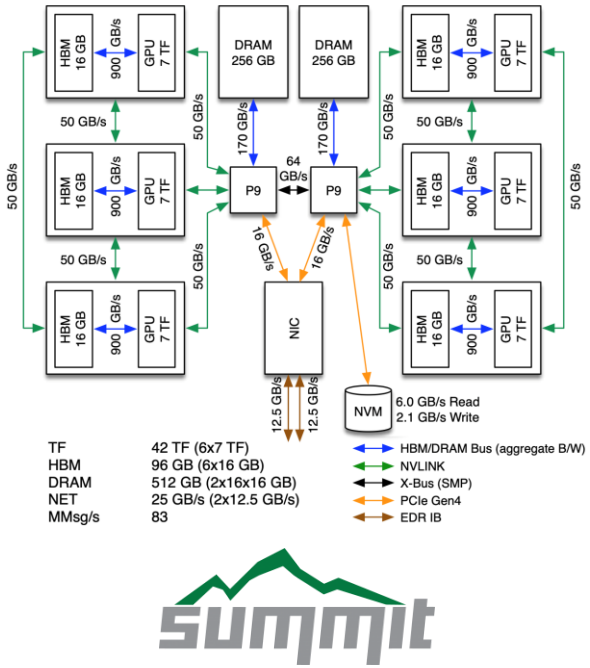
OpenACC kokkos

EXASCALE COMPUTING PROJECT

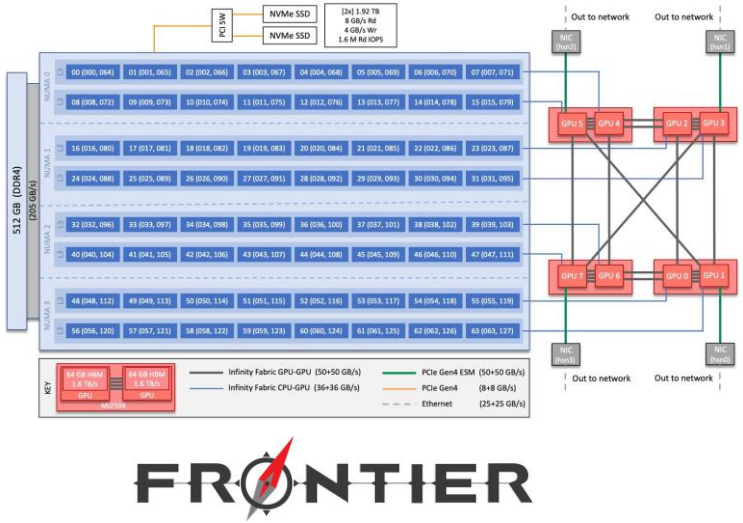
U.S. DEPARTMENT OF ENERGY

Motivation

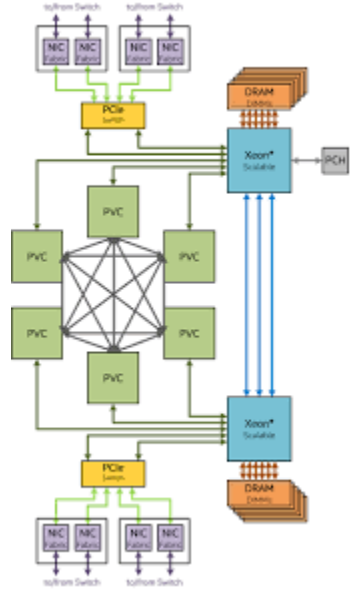
- Programming productivity!



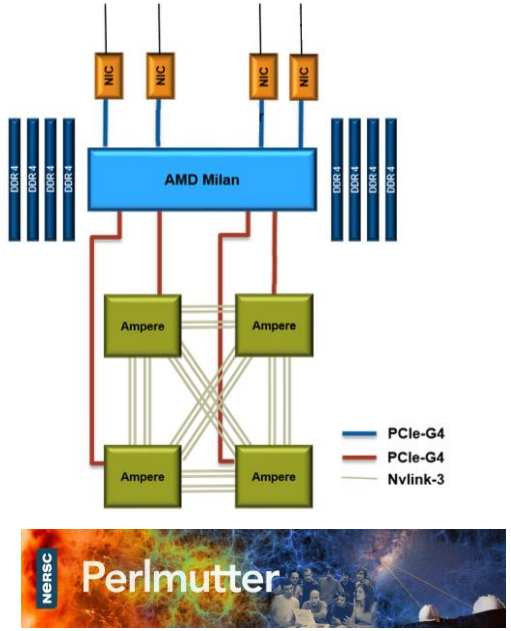
summit



FRONTIER



Aurora



OAK RIDGE National Laboratory

Descriptive (Agnostic) VS Prescriptive (Device Specific)

A software paradigm in which no particular programming model is promoted

‘‘Teaching principles rather than programing models features’’

“Interoperable across the systems and there are no prejudices towards using a specific technology, model, methodology or data”

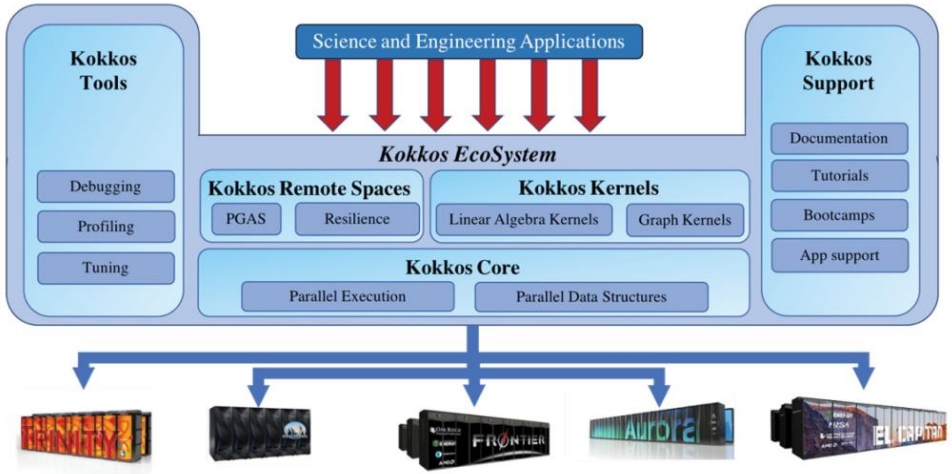
“A technology that can be used with any type of system, regardless of underlying systems' technology or architecture”

Less complicated programs are often ***more*** performing

Proactive, not reactive

Kokkos

- Open-source performance portability C++ template and metaprogramming
- It is implemented as a template library on top of:
  - CUDA, HIP, OpenMP, OpenMP Target, HPX, SYCL, etc, **and now OpenACC too!!**
- Target back end must be defined at compilation time
  - (KOKKOS_DEVICE=OpenACC)
- It can only use one back end/device at a time



https://github.com/kokkos/kokkos

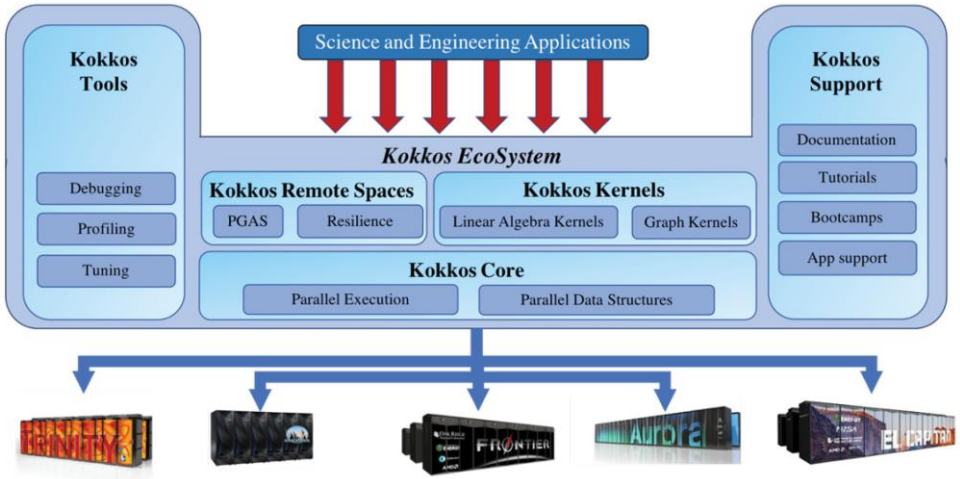
Kokkos Programing Model

- **Memory management is composed by:**
  - Kokkos_malloc and Kokkos views
- **Data parallel execution:**
  - parallel_for, parallel_reduce and parallel_scan
  - 3 different APIs
    - Single Range, Multi-Dimensional Range and Hierarchical Parallelism
  - Each Kokkos construct has:
    - Number of iterations
    - A C++ Lambda that acts like a function

```
auto X  = static_cast<double*>(Kokkos::kokkos_malloc<>(N * sizeof(double)));
auto Y  = static_cast<double*>(Kokkos::kokkos_malloc<>(N * sizeof(double)));

Kokkos::parallel_for( "axpy_init", N, KOKKOS_LAMBDA ( int n )
{
 X[n] = InitValue;
 Y[n] = InitValue;
});

Kokkos::parallel_for( "axpy_computation", N, KOKKOS_LAMBDA ( int n )
{
  double alpha = ALPHA;
  Y[n] += alpha * X[n];
});
```



KokkACC Implementation

```
auto X  = static_cast<double*>(Kokkos::kokkos_malloc<>(N * sizeof(double)));
auto Y  = static_cast<double*>(Kokkos::kokkos_malloc<>(N * sizeof(double)));

Kokkos::parallel_for( "axpy_init", N, KOKKOS_LAMBDA ( int n )
{
 X[n] = InitValue;
 Y[n] = InitValue;
});

Kokkos::parallel_for( "axpy_computation", N, KOKKOS_LAMBDA ( int n )
{
  double alpha = ALPHA;
  Y[n] += alpha * X[n];
});
```

kokkos

```
template <class TagType> inline void execute_impl() const
{
  OpenACCExec::verify_is_process("Kokkos::Experimental::OpenACC parallel_for");
  OpenACCExec::verify_initialized("Kokkos::Experimental::OpenACC parallel_for");

  const auto begin = m_policy.begin();
  const auto end   = m_policy.end();
  if (end <= begin) return;
  const FunctorType a_functor(m_functor);

  #pragma acc parallel loop gang vector copyin(a_functor)
  for (auto i = begin; i < end; i++)
    a_functor(i);
}
```

OpenACC

```
SIZE = M * N * sizeof(double);
Kokkos::View <double**> X("X", M , N);
Kokkos::View <double**> Y("Y", M , N);

typedef Kokkos::MDRangePolicy< Kokkos::Rank<2> >mdrange_policy;

Kokkos::parallel_for( "axpy_init", mdrange_policy( {0, 0}, {M, N} ), KOKKOS_LAMBDA ( int m, int n  )
{
  X(m, n) = InitValue;
  Y(m, n) = InitValue;
} );

Kokkos::parallel_for( "axpy_computation", mdrange_policy( {0, 0}, {M, N} ), KOKKOS_LAMBDA ( int m, int n )
{
  double alpha = ALPHA;
  Y(m, n) += alpha * X(m, n);
} );
```

kokkos

```
template <class TagType, int Rank> inline typename std::enable_if<Rank == 2>::type execute_functor( const
↪  FunctorType& functor, const Policy& policy ) const
{
  const FunctorType a_functor(functor);
  int begin1 = policy.m_lower[0];
  int end1 = policy.m_upper[0];
  int begin2 = policy.m_lower[1];
  int end2 = policy.m_upper[1];

  #pragma acc parallel loop gang vector collapse(2) copyin(a_functor)
  for (auto i0 = begin1; i0 < end1; i0++) {
    for (auto i1 = begin2; i1 < end2; i1++) {
      a_functor(i0, i1);
  }
 }
}
```

OpenACC

```
SIZE = M * N * sizeof(double);
auto X  = static_cast<double*>(Kokkos::kokkos_malloc<>(SIZE));
auto Y  = static_cast<double*>(Kokkos::kokkos_malloc<>(SIZE));

typedef Kokkos::TeamPolicy<>               team_policy;
typedef Kokkos::TeamPolicy<>::member_type  member_type;

Kokkos::parallel_for( "axpy_init", team_policy( M, Kokkos::AUTO ), KOKKOS_LAMBDA ( const member_type
↪  &teamMember )
{
  const int i = teamMember.league_rank();
  Kokkos::parallel_for( Kokkos::TeamThreadRange( teamMember, N ), [&] ( const int j )
  {
    X[i * N + j] = InitValue;
    Y[i * N + j] = InitValue;
  } );
} );

Kokkos::parallel_for( "axpy_computation", team_policy( M, Kokkos::AUTO ), KOKKOS_LAMBDA ( const
↪  member_type &teamMember )
{
  const int i = teamMember.league_rank();
  Kokkos::parallel_for( Kokkos::TeamThreadRange( teamMember, N ), [&] ( const int j )
  {
    double alpha = ALPHA;
    Y[i * N + j] += alpha * X[i * N + j];
  } );
} );
```

kokkos

```
template <class TagType> inline void execute_impl() const {
 OpenACCExec::verify_is_process( "Kokkos::Experimental::OpenACC parallel_for");
 OpenACCExec::verify_initialized("Kokkos::Experimental::OpenACC parallel_for");
 auto league_size   = m_policy.league_size();
 auto team_size     = m_policy.team_size();
 auto vector_length = m_policy.impl_vector_length();
 const FunctorType a_functor(m_functor);

 #pragma acc parallel loop gang copyin(a_functor)
 for ( int i = 0; i < league_size; i++ ) {
   int league_id = i;
   typename Policy::member_type team( league_id, league_size, team_size, vector_length );
   a_functor(team);
 }
}
#pragma acc routine worker
template <typename iType, class Lambda>
KOKKOS_INLINE_FUNCTION void parallel_for( const Impl::TeamThreadRangeBoundariesStruct<iType,
↪  Impl::OpenACCExecTeamMember>& loop_boundaries, const Lambda& lambda ) {
  #pragma acc loop worker
  for (iType j = loop_boundaries.start; j < loop_boundaries.end; j++) {
    lambda(j);
  }
}
```

OpenACC

- Both models attempts to be architecture agnostic
- Strong connection between Kokkos front-end and OpenACC specification
- All this makes easy the implementation, maintainability and sustainability of the OpenACC back end

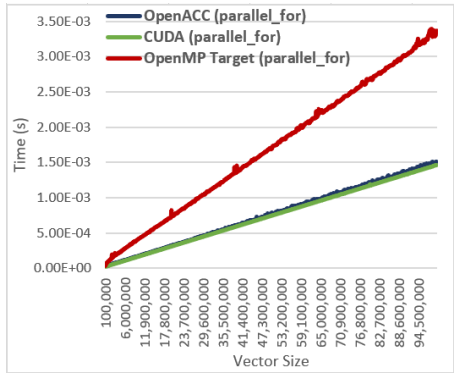
```
#pragma acc routine seq
inline unsigned int atomic_fetch_add( volatile
↪  unsigned int *const dest, const unsigned int
↪  &val ) {
  unsigned int retval;
  unsigned int *ptr = const_cast<unsigned int
  ↪  *>(dest);
  #pragma acc atomic capture {
    retval = ptr[0];
    ptr[0] += val;
  }
  return retval;
}
```

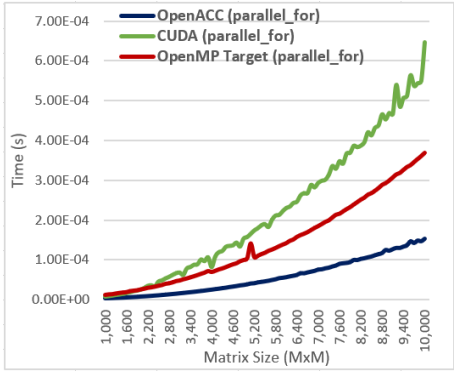
https://github.com/kokkos/kokkos/tree/develop/core/src/OpenACC

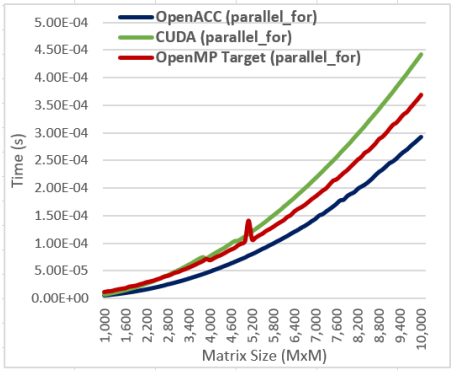
Performance Evaluation on SUMMIT

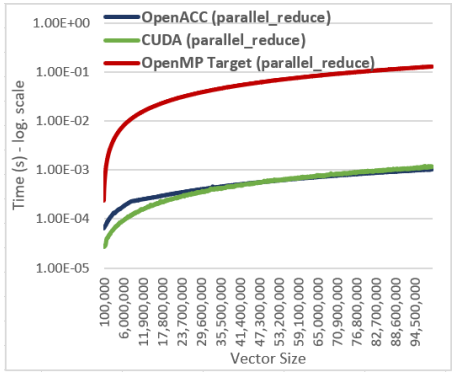
- ORNL SUMMIT
  - 1x NVIDIA C100 GPU (16GB)
  - CUDA (CUDA 11.0.3)
  - OpenMP Target (LLVM 15.0.0)
  - OpenACC (NVHPC 21.3)

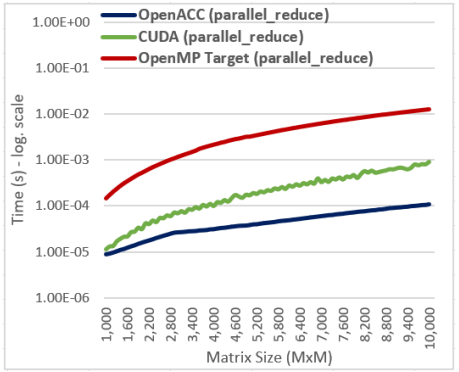
- Mini-benchmarks

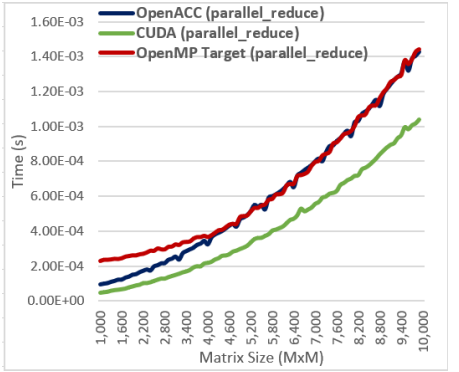




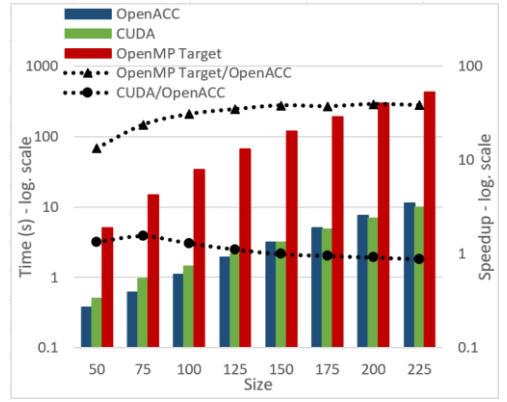




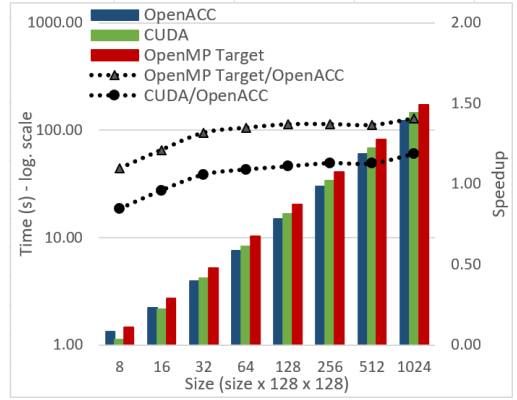




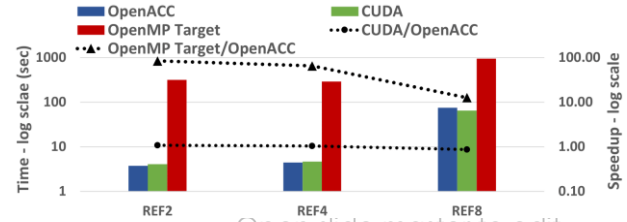
- LULESH



- MiniFE



- LAMMPS-SNAP



“Low-Level” Performance Evaluation on SUMMIT

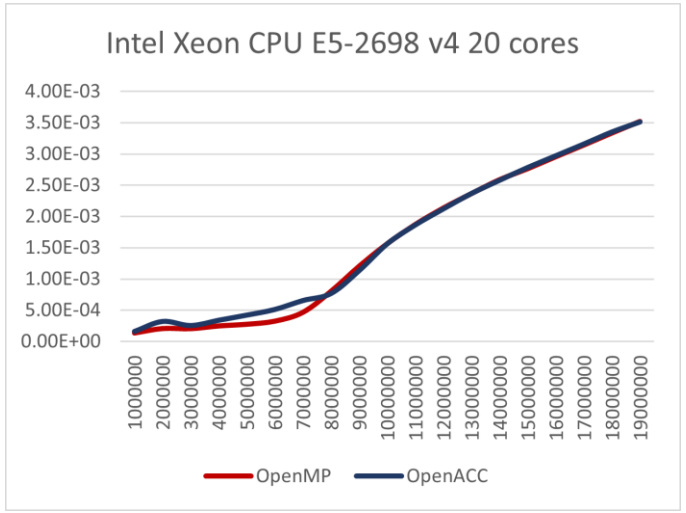
- Time
- Occupancy
- #Instructions
- Block size
- Warps
- Bandwidth
  - Global memory (RAM)
  - L2
  - L1
  - Shared memory

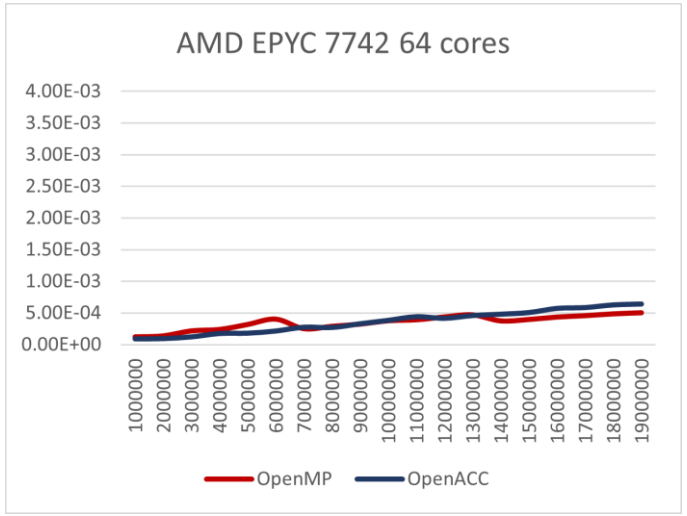
—Mini-benchmarks—						
—AXPY—						
–Kokkos execution policy–	CUDA		OpenACC		OpenMP target	
SR	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**
axpy_init	1,286,228/8.18	1/128/5.23	922,648/94.83	1/128/60.69	2,717,105/22.59	1/128/14.46
axpy_comp	1,445,109/87.72	1/128/56.14	1,468,756/97.51	1/128/62.41	3,764,011/21.57	1/128/13.81
Memory operations	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**
HtD/axpy_init	13,471/8	69/24/37/13	7,359/4	97/34/52/23	9,538/5	69/24/37/13
DtH/axpy_comp	12,225/6	92/32/24/12	4,322/2	92/32/24/13	8,927/4	92/32/24/12
Memset	4,023/2	-	0/-	-	0/-	-
MD	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**
axpy_init	5,142,724/3.55	1/32/2.27	922,583/94.77	1/128/60.65	3,800,173/22.08	1/128/14.13
axpy_comp	5,141,252/10.07	1/32/6.45	1,520,085/97.09	1/128/62.14	2,817,873/22.82	1/128/14.61
Memory operations	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**
HtD/axpy_init	13,477/8	34/12/18/9	7,296/4	97/34/52/40	8,926/5	29/10/37/59
DtH/axpy_comp	13,282/6	52/18/13/9	4,317/2	91/32/24/25	7,805/4	33/11/28/45
Memset	7,819/4		0/-		0/-	
HR	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**
axpy_init	916,152/94.95	1/128/60.77	909,530/72.64	1/256/46.49	1,046,552/95.10	1/256/60.86
axpy_comp	1,690,036/96.88	1/128/62.01	1,562,549/95.99	1/256/61.44	1,912,053/97.01	1/256/62.09
Memory operations	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**
HtD/axpy_init	13,345/8	96/34/52/6	7,456/4	86/30/69/41	11,937/7	83/29/44/17
DtH/axpy_comp	12,255/6	79/29/34/5	4,545/2	85/30/35/22	12,606/6	69/24/21/10
Memset	4,088/2		0/-		0/-	

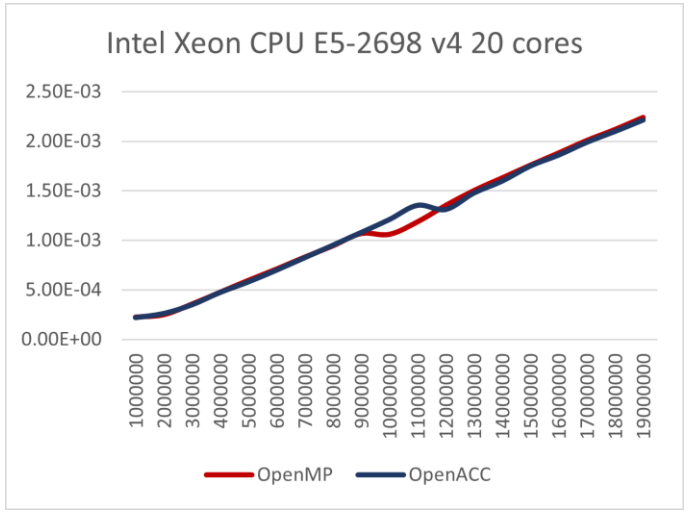
—DOT product—						
–Kokkos execution policy–	CUDA		OpenACC		OpenMP target	
SR	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**
dot_init	1,286,199/8.57	1/128/5.49	927,511/94.71	1/128/60.62	2,717,455/21.57	1/128/13.81
dot_comp	1,199,189/40.02	1/256/25.61	920,312/97.62	1/128/62.3	133,444,166/24.93	1/128/15.96
			91,164/12.43	2/256/7.95		
Memory operations	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**
HtD/dot_init	13,375/8	69/24/37/13	6,976/4	98/34/52/24	11,136/6	34/12/40/59
DtH/dot_comp	12,226/6	73/25/20/5	6,943/3	97/34/26/22	10,399/5	0.6/0.7/1.8/2.7
Memset	3,991/2		1,344/1	0.3/0.1/6.9/0	0/-	
MD	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**
dot_init	5,119,158/3.49	1/32/2.23	917,207/94.74	1/128/60.63	3,170,832/22.08	1/128/14.13
dot_comp	9,121,816/40.0	1/256/25.60	981,304/97.18	1/128/62.20	10,3469,313/24.90	1/128/15.94
Memset			89,723/12.43	2/256/7.95		
Memory operations	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**
HtD/dot_init	13,631/8	34/12/18/9	7,357/4	98/34/52/40	10,881/6	29/10/37/59
DtH/dot_comp	13,407/6	19/5/9/7	6,465/3	95/33/25/42	9,954/5	0.8/0.9/2.3/3.6
Memset	8,724/4		1,376/1	0.3/0.1/6/0	0/-	
HR	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**	**Time/Occup.**	**Inst./BS/Warps**
dot_init	917,848/95.01	1/128/60.81	910,710/72.61	1/256/46.47	1,060,792/94.97	1/256/60.78
dot_comp	1,057,367/96.88	1/128/62.01	1,382,709/74.00	1/256/47.36	1,463,991/48.03	1/256/30.74
			15,484/12.07	2/256/7.72		
Memory operations	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**	**Time/Inst.**	**RAM/L2/L1/SM**
HtD/dot_init	10,780/8	96/34/51/6	6,144/4	85/30/69/41	12,642/6	85/29/45/18
DtH/dot_comp	10,463/6	85/30/25/11	5,184/3	82/28/20/12	11,837/7	61/21/20/28
Memset	2,809/2		990/1	0.2/0.2/6.2/0	0/-	

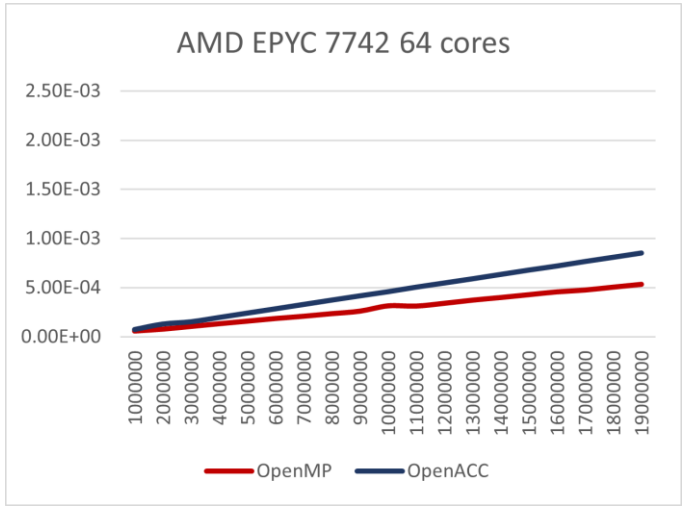
Performance on CPUs

- Intel and AMD CPUs
- Mini-benchmarks (AXPY on top, DOT on bottom)





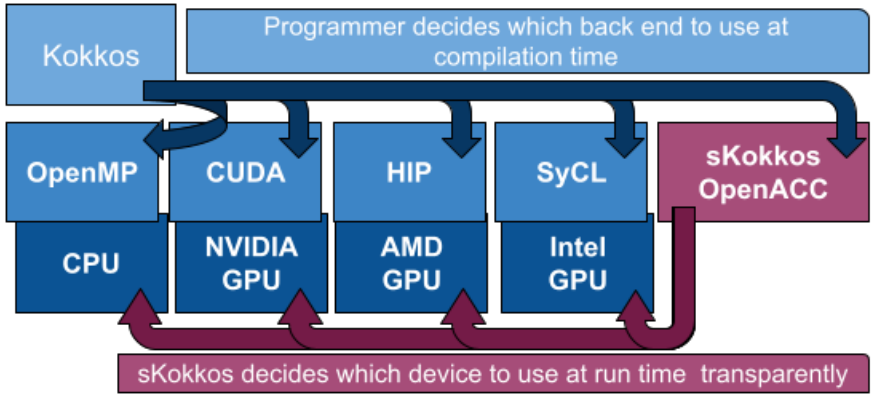


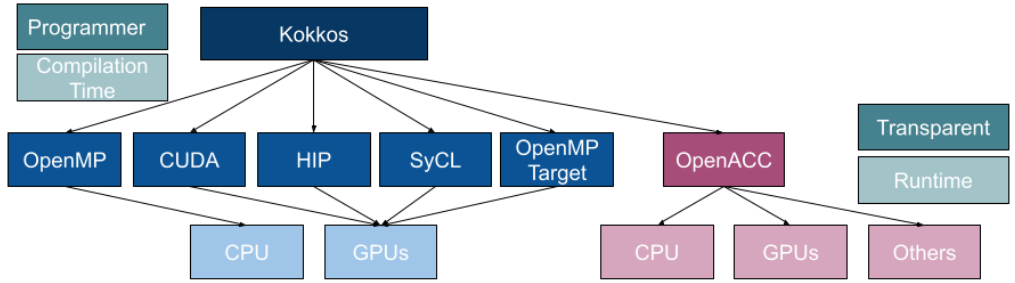


OAK RIDGE National Laboratory

sKokkos: Enabling Kokkos with Transparent Device Selection on Heterogeneous Systems

- sKokkos decides which device (CPU or GPU) is better at runtime (at the very beginning)






```
auto X  = static_cast<double*>(Kokkos::kokkos_malloc<>(N * sizeof(double)));
auto Y  = static_cast<double*>(Kokkos::kokkos_malloc<>(N * sizeof(double)));

Kokkos::parallel_for( "axpy_init", N, KOKKOS_LAMBDA ( int n )
{
 X[n] = InitValue;
 Y[n] = InitValue;
});

Kokkos::parallel_for( "axpy_computation", N, KOKKOS_LAMBDA ( int n )
{
  double alpha = ALPHA;
  Y[n] += alpha * X[n];
});
```

kokkos

```
SIZE = M * N * sizeof(double);
Kokkos::View <double**> X("X", M , N);
Kokkos::View <double**> Y("Y", M , N);

typedef Kokkos::MDRangePolicy< Kokkos::Rank<2> >mdrange_policy;

Kokkos::parallel_for( "axpy_init", mdrange_policy( {0, 0}, {M, N} ), KOKKOS_LAMBDA ( int m, int n  )
{
  X(m, n) = InitValue;
  Y(m, n) = InitValue;
} );

Kokkos::parallel_for( "axpy_computation", mdrange_policy( {0, 0}, {M, N} ), KOKKOS_LAMBDA ( int m, int
{
  double alpha = ALPHA;
  Y(m, n) += alpha * X(m, n);
} );
```

kokkos

```
template <class FunctorType, class... Traits>
class ParallelFor< FunctorType,
    Kokkos::RangePolicy<Traits...>,
    Kokkos::Experimental::OpenACC > {
 private:
  using Policy    = Kokkos::RangePolicy<Traits...>;
  using WorkTag   = typename Policy::work_tag;
  using WorkRange = typename Policy::WorkRange;
  using Member    = typename Policy::member_type;
  const FunctorType m_functor;
  const Policy m_policy;
 public:
  inline void execute() const
  {      execute_impl<WorkTag>(); }
  template <class TagType>
  inline void execute_impl() const {
    OpenACCExec::verify_is_process(
        "Kokkos::Experimental::OpenACC parallel_for");
    OpenACCExec::verify_initialized(
        "Kokkos::Experimental::OpenACC parallel_for");
    const auto begin = m_policy.begin();
    const auto end   = m_policy.end();
    if (end <= begin) return;
    const FunctorType a_functor(m_functor);
    #pragma acc parallel loop gang vector copyin(a_functor)
    for (auto i = begin; i < end; i++) {
        a_functor(i);
    }
  }
  ...
};
```

OpenACC

```
template <class TagType, int Rank>
inline typename std::enable_if<Rank == 2>::type
 execute_functor( const FunctorType& functor,
 const Policy& policy ) const {
 const FunctorType a_functor(functor);
 int begin0 = policy.m_lower[0];
 int end0 = policy.m_upper[0];
 int begin1 = policy.m_lower[1];
 int end1 = policy.m_upper[1];

 acc_device_t target_dev;
 target_dev = acc_get_device_type();

 if ( target_dev == acc_device_nvidia ) {
  #pragma acc parallel loop gang vector \
              collapse(2) copyin(a_functor)
  for (auto i1 = begin1; i1 < end1; i1++) {
   for (auto i0 = begin0; i0 < end0; i0++) {
    a_functor(i0, i1);
   }
  }
 }
 else if (target_dev == acc_device_host) {
  #pragma acc parallel loop gang vector \
              copyin(a_functor)
  for (auto i0 = begin0; i0 < end0; i0++) {
   for (auto i1 = begin1; i1 < end1; i1++) {
    a_functor(i0, i1);
   }
  }
 }
}
```

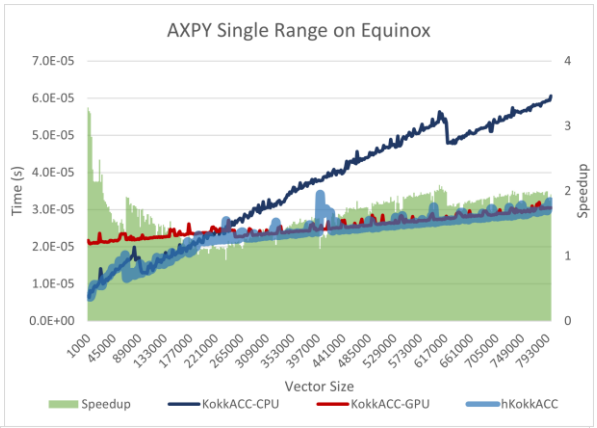
OpenACC

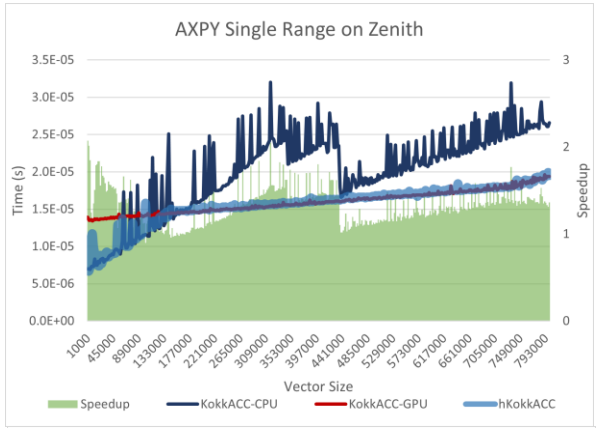
- Kokkos::set_device(tunning factor)
  - Tuning factor: number of operations, nnz elements, size of the grid, etc.
  - CPU performance = Tuning factor/CPU flops
  - GPU performance = Tuning factor/GPU flops + GPU overhead

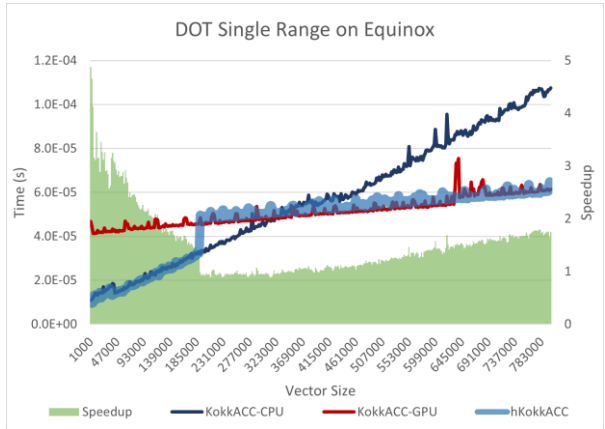
https://code.ornl.gov/5pv/skokkos

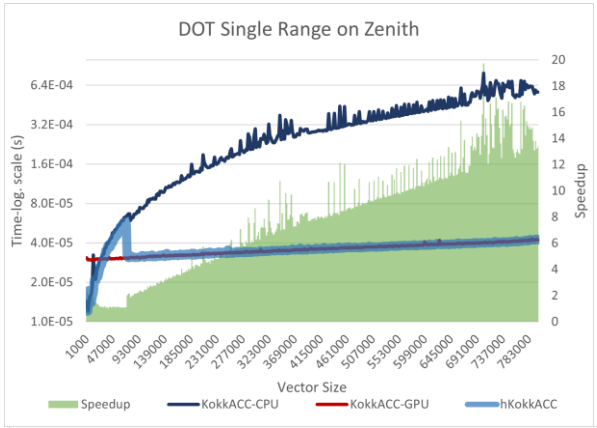
sKokkos Performance: Mini-benchmarks

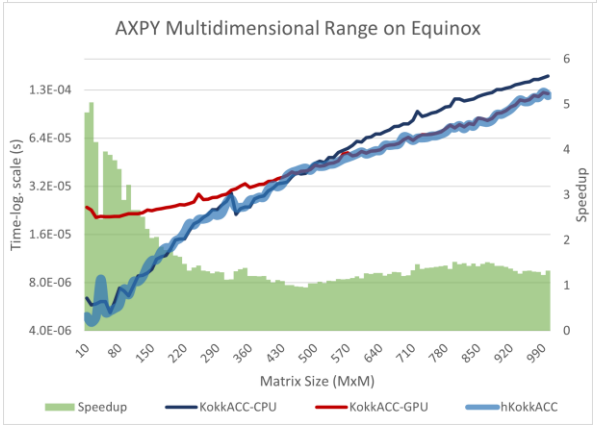
- Two different heterogeneous systems (ExCL ORNL):
  - Equinox: 1x Intel Xeon E5-2698 v4 20-Core CPU + 1x NVIDIA V100 GPU
  - Zenith: 1x AMD Ryzen 3970X 32-Core CPU + 1x NVIDIA GeForce RTX 3090 GPU
- Mini-benchmarks:

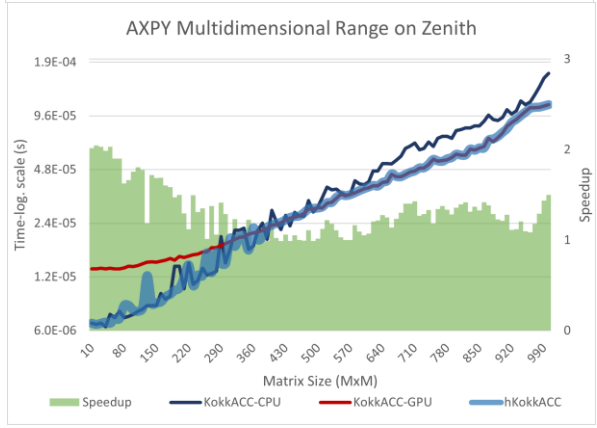


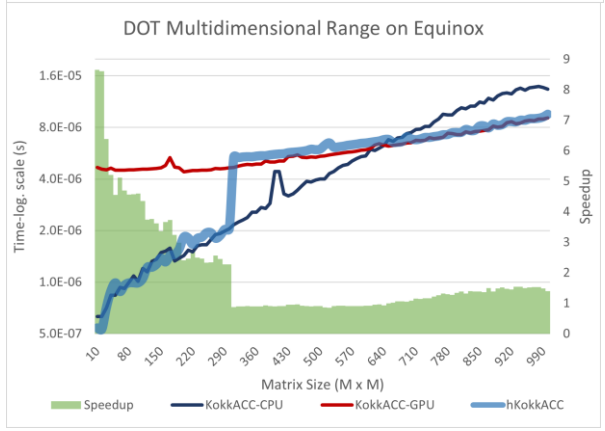


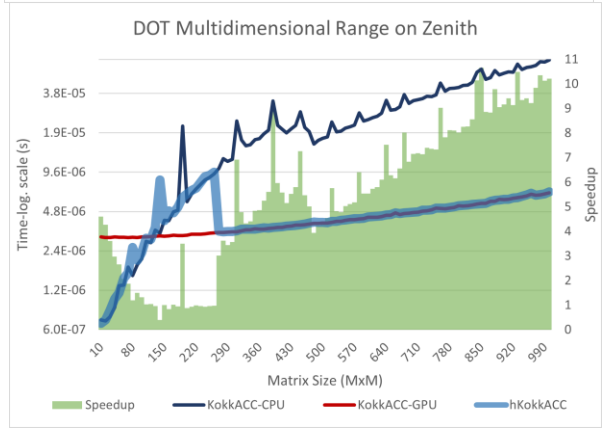








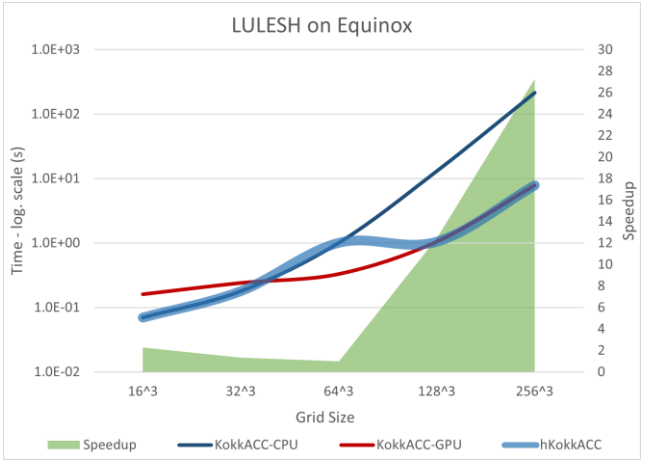


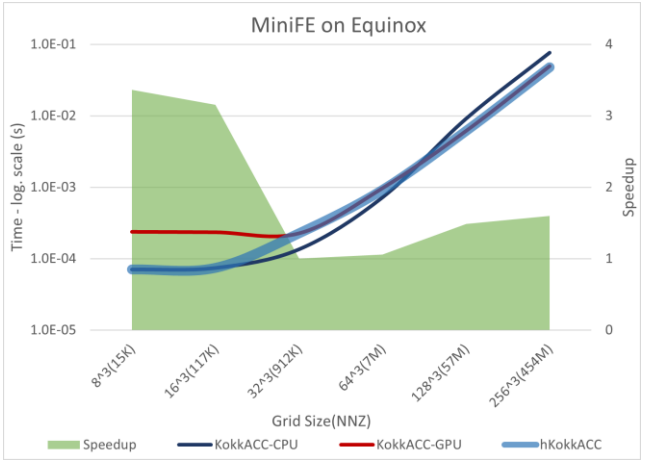


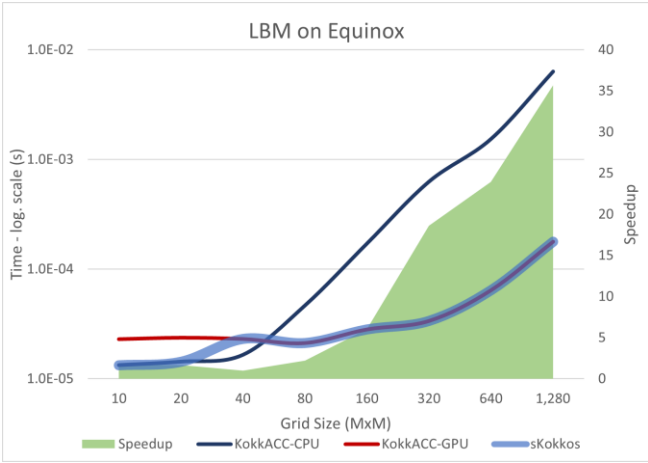
OAK RIDGE National Laboratory

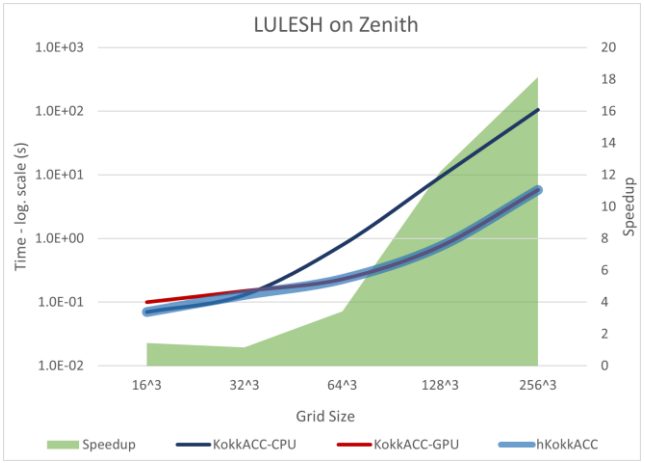
sKokkos Performance: Mini-apps

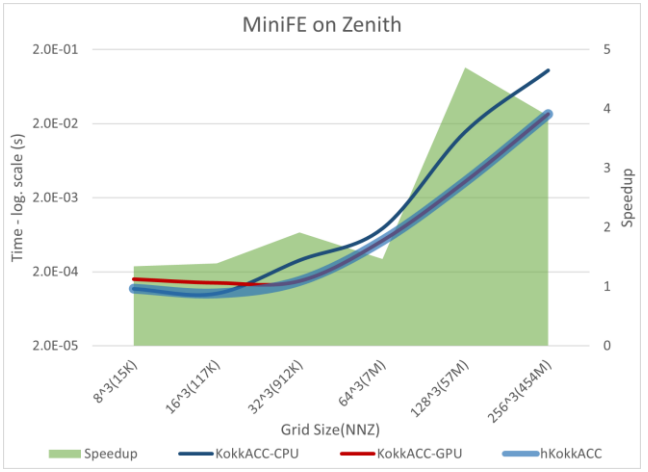
- Two different mini-apps (tunning factors):
  - Lulesh: Stencil computation on the 3D domain (size of the domain)
  - MiniFE: Conjugate Gradient (#nnz)
  - LBM: Lattice-Boltzmann Method (#operations)

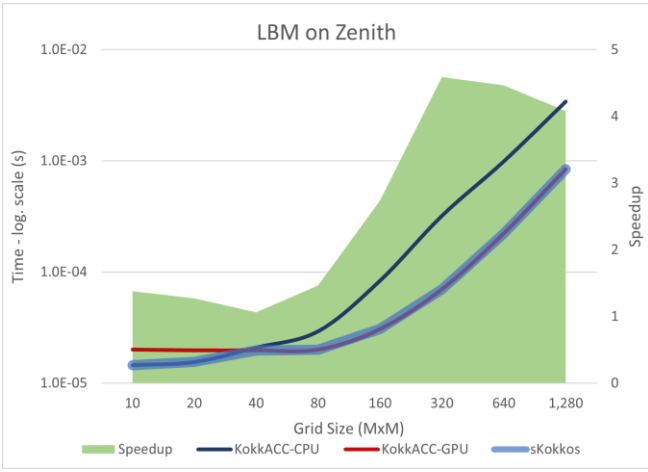












Descriptive (Agnostic) VS Prescriptive (Device Specific)

- Next, we highlight why it is possible to provide competitive or even better performance using a high-level and high programming productivity descriptive (pragma-based) model (OpenACC) than using a low-level prescriptive (device-specific) model (CUDA) for C++ Metaprogramming solutions (Kokkos).
- C++ Metaprogramming solutions, like Kokkos, relay on C++ lambdas. C++ lambdas are defined by application programmers and can express any operation.
- Device-specific solutions like CUDA weren't designed to work at lambda level originally. CUDA Kokkos back-end **relays on CUDA developers**, who don't know which operations will be computed by GPU kernels, but they must take decisions about size of CUDA blocks, memory usage, synchronization, etc. This makes the optimization of these solutions extremely difficult or even impossible.
- OpenACC backend **relays on compiler**, which can work at "lambda" level and take the best decisions depending on the operations defined by C++ lambdas and application developers and **increasing the programming productivity**

Conclusions and Future Work

- OpenACC vs CUDA (NVIDIA GPU):
  - Competitive performance for Single Range
  - Better performance for Multi-Dimensional
  - Competitive performance for Hierarchical Parallelism parallel_for and worse performance for parallel_reduce
  - Competitive/better performance on mini-apps (LULESH, miniFE, LAMPS-SNAP)
- OpenACC vs OpenMP Target (NVIDIA GPU):
  - Better performance in most of the cases tested.
- OpenACC vs OpenMP (Intel and AMD CPUs):
  - Similar performance on Intel and AMD CPUs than OpenMP
- sKokkos:
  - Enabling Kokkos with Transparent Device Selection on Heterogeneous Systems
  - Transparent device selection on two different heterogeneous systems and three mini-apps (LULESH, miniFE, LBM)
- Future/Ongoing Efforts:
  - Support for multi-GPU and distributed memory

Acknowledgments

- KokkACC team: Seyong Lee, Joel Denny, Marc Gonzalez-Tellada, Jeffrey Vetter, Pedro Valero-Lara.
- Best paper award at WACCPD@SC'22

- ECP Proteas-TUNE project (https://www.exascaleproject.org/research-project/proteas-tune/)
- This research used resources of the Oak Ridge Leadership Computing Facility and the Experimental Computing Laboratory at the Oak Ridge National Laboratory, which is supported by DOE's Office of Science under Contract No. DE-AC05-00OR22725. This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the DOE's Office of Science andthe National Nuclear Security Administration. This work has been authored by UT-Battelle LLC under Contract No. DE-AC05-00OR22725 with the DOE.

OAK RIDGE National Laboratory

KokkACC: Enhancing Kokkos with OpenACC

Thanks!!
Questions??

PhD. Pedro Valero-Lara,

Computer Scientist at Programming Systems Group
valerolarap@ornl.gov

OpenACC Webinar

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

OpenACC kokkos

ECP EXASCALE COMPUTING PROJECT

U.S. DEPARTMENT OF ENERGY