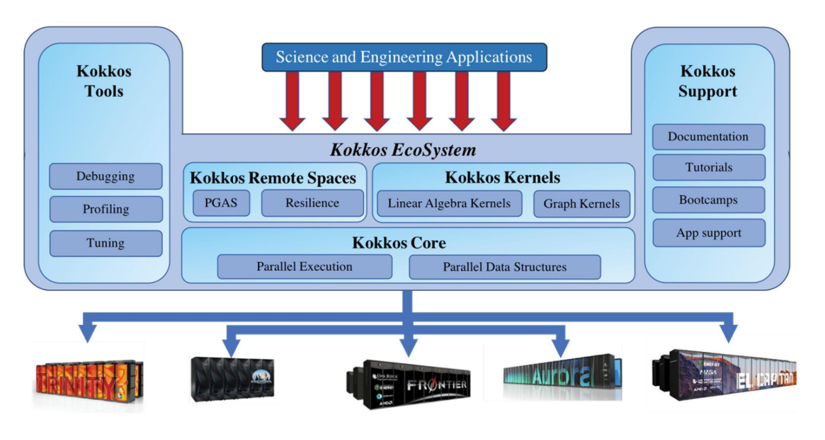
KokkACC: Enhancing Kokkos with OpenACC

BoF: OpenACC User Experience: Relevance, Hackathons, and Roadmaps

Pedro Valero-Lara, Computer Scientist,
Oak Ridge National Laboratory, valerolarap@ornl.gov
Seyong Lee, Marc Gonzalez-Tallada, Joel Denny, and Jeffrey S. Vetter

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




```cpp
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

- **Single Range:**

```
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
Directives for Accelerators

- **Multi-dimensional Range:**

```
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
Directives for Accelerators

- **Hierarchical Parallelism**

```
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
Directives for Accelerators

- **Atomics**

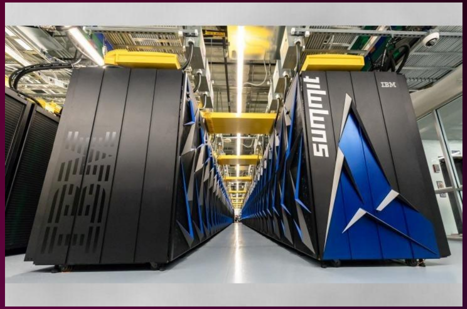
```
#pragma acc routine seq
inline unsigned int atomic_fetch_add( volatile
↪   unsigned int *const dest, const unsigned int
↪   &val ) {
  unsigned int retval;
  unsigned int *ptr = const_cast<unsigned int
   ↪   *>(dest);
  #pragma acc atomic capture {
    retval = ptr[0];
    ptr[0] += val;
  }
  return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

- 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

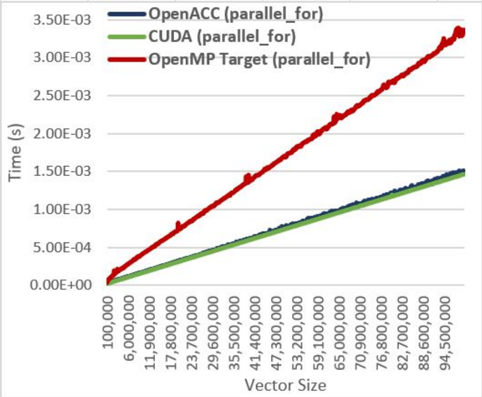
https://github.com/kokkos/kokkos/tree/develop/core/src/OpenACC

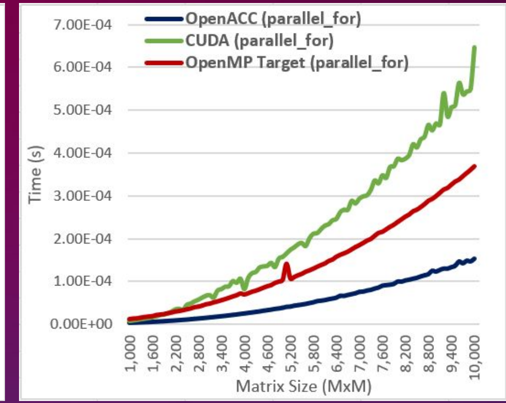
Performance Evaluation on GPUs

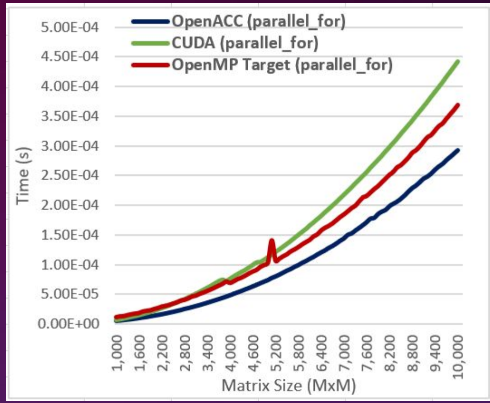
ORNL SUMMIT
- 1x NVIDIA Volta V100 GPU (16 GB)
- CUDA back-end (CUDA 11.0.3)
- OpenMP Target back-end (LLVM 15.0.0)
- OpenACC back-end (NVHPC 21.3)

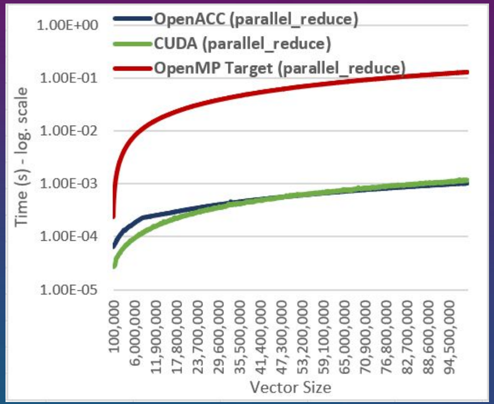


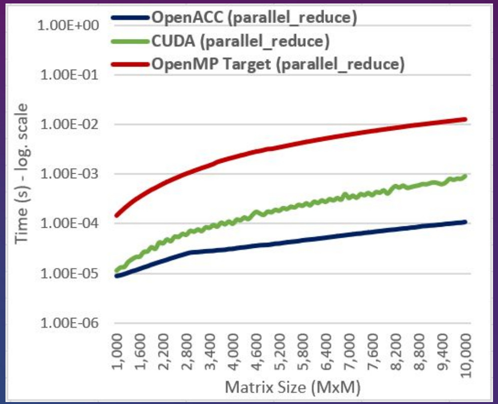
- Mini-benchmarks

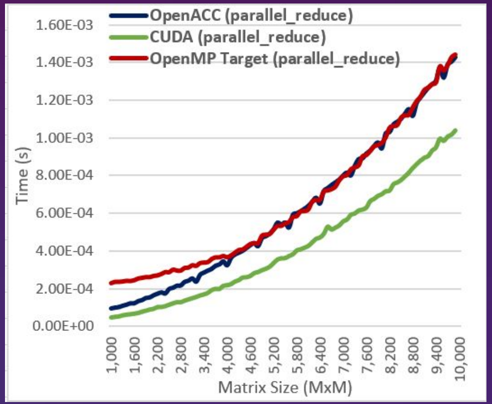




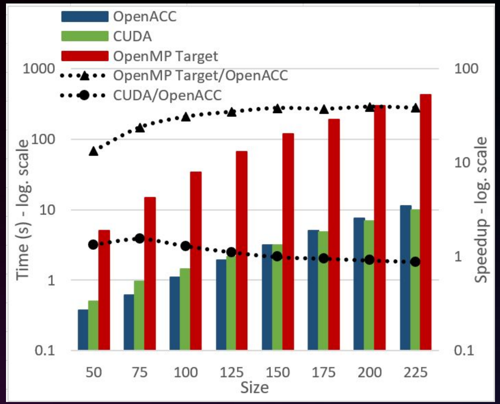




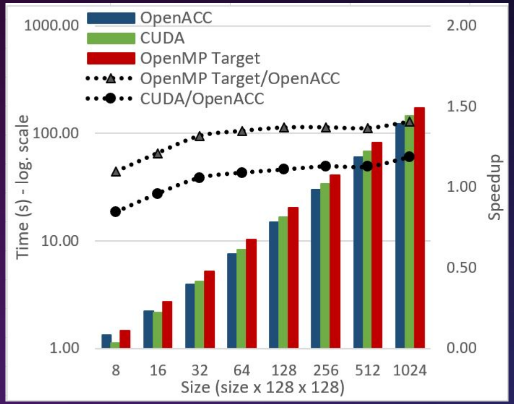




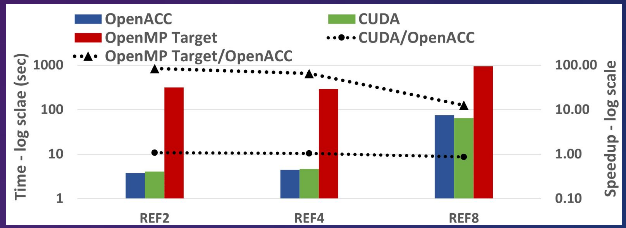
- LULESH



- MiniFE

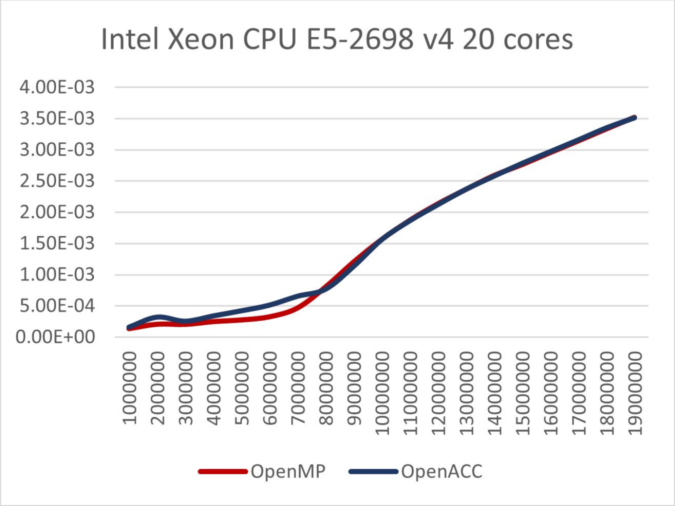


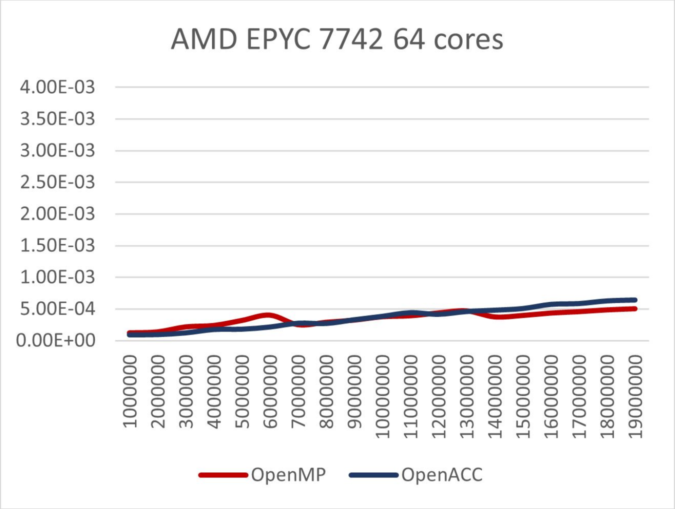
- LAMMPS-SNAP

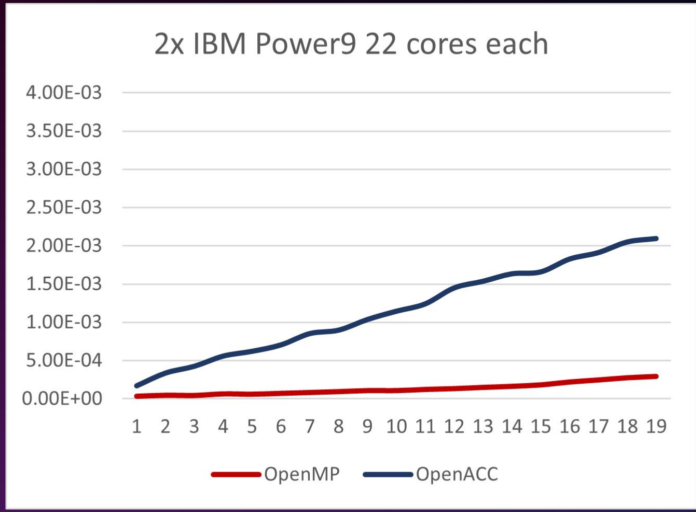


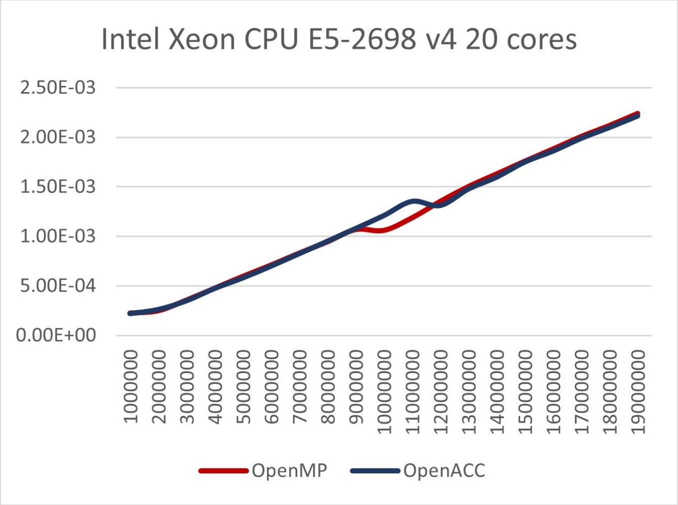
Performance Evaluation on CPUs

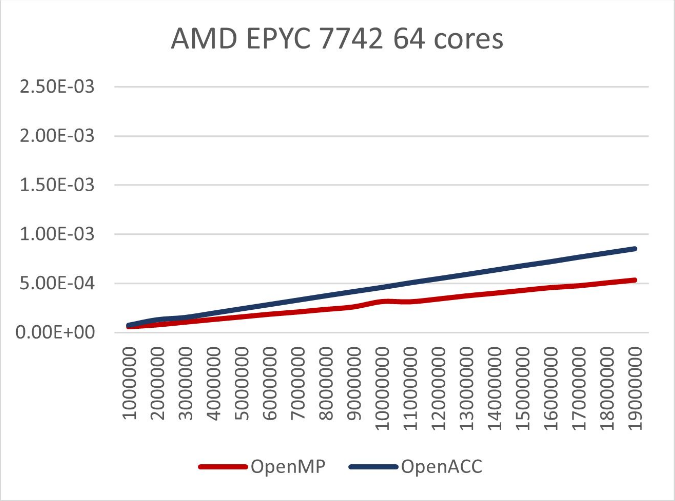
- Intel, AMD, and IBM CPUs
- Mini-benchmarks

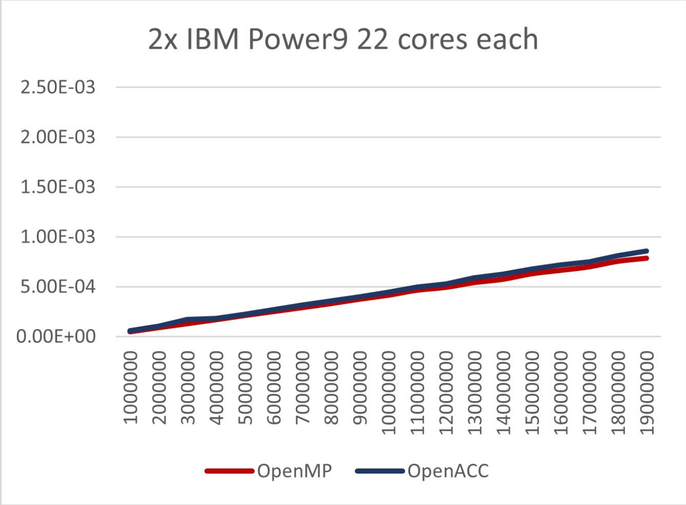












Descriptive VS Prescriptive (Device Specific)

- 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, AMD, and IBM CPUs):
  - Similar performance on Intel, AMD and IBM except for AXPY (parallel_for) on IBM
- KokkACC is aligned with other important efforts:
  - Analysis, codesign and development of the OpenACC capacity for C++.
  - Enhancing C++ [for HPC] using the capacity of OpenACC.
  - Design of new OpenACC capabilities.
- Future Efforts:
  - Implement all Kokkos front-end features in OpenACC back end
  - Explore novel optimizations

Thanks!

Questions??

KokkACC: Enhancing Kokkos with OpenACC

BoF: OpenACC User Experience: Relevance, Hackathons, and Roadmaps

Pedro Valero-Lara, Computer Scientist,
Oak Ridge National Laboratory, valerolarap@ornl.gov
Seyong Lee, Marc Gonzalez-Tallada, Joel Denny, and Jeffrey S. Vetter