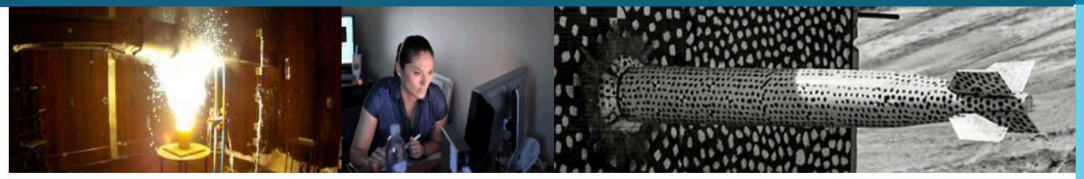
SAND2020-3071PE

PORTING SPHYNX TO GPU-ENABLED SYSTEMS



Seher Acer, ExaGraph TelCon, 03/09/2020

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Outline

- SPHYNX – Trilinos aspects
- A Kokkos example – GEMM for tall & skinny matrices

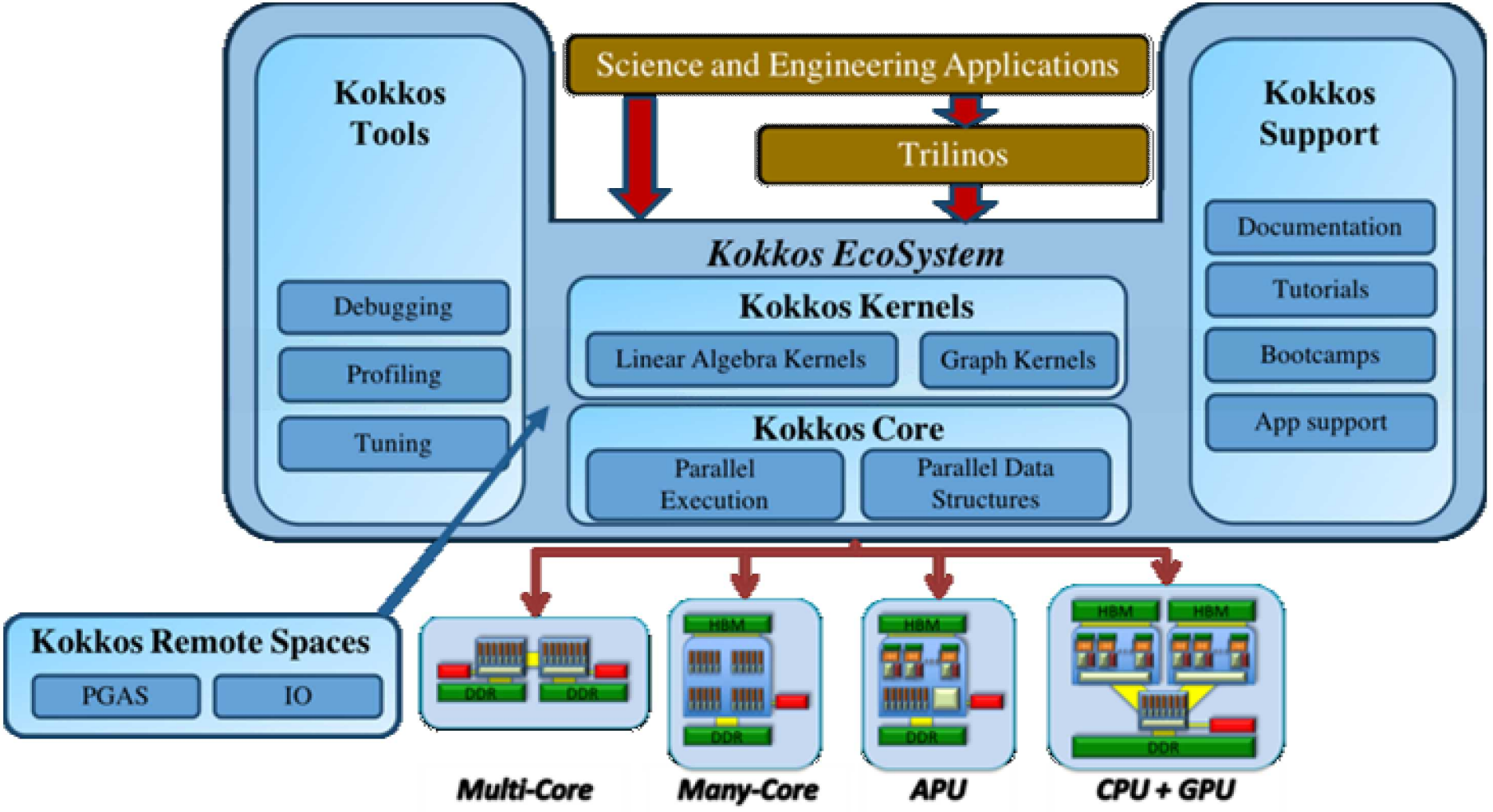
SPHYNX

- SPHYNX: Spectral Partitioning on HYbrid aNd aXelerator-enabled systems
  - paper accepted by AsHES @ IPDPS 2020
- SPHYNX uses Trilinos packages
- Trilinos uses Kokkos & Kokkos Kernels framework
  - provides optimized functions and kernels
  - Building on top of Kokkos means that SPHYNX will run on all the ECP platforms from Day 1
    - Perlmutter (CUDA), Aurora (SYCL), Frontier (HIP, OpenMP)

SPHYNX

- SPHYNX uses Tpetra
  - A Trilinos package -- distributed linear algebra objects
    - e. g., sparse matrices, dense vectors, …
  - Supports hybrid parallelism: MPI+X
- Tpetra uses Kokkos & Kokkos Kernels
  - Data-management and parallelization on the node
  - X: Serial, OpenMP, Pthreads, CUDA

KOKKOS ECOSYSTEM



A KOKKOS EXAMPLE - GEMM

- `C = beta C + alpha A^T B`
- `A` and `B` are tall & skinny dense matrices with `n` rows
- Serial pseudocode:
  - `for i=1 to numCrows`
    - `for j=1 to numCcols`
      - `C(i,j)= beta C(i,j) + alpha < A(:,i),B(:,j)>`

A KOKKOS EXAMPLE - GEMM

- $C = beta\ C + alpha\ A^T\ B$

```
409
410     // Determine the local length for the dot product
411     chunkSize = dotSize / numDivPerDot;
412     if(numDivPerDot > 1)
413       chunkSize++;
414
415     // Initialize C matrix if beta != 1
416     if(beta == CVT::zero()) {
417       Kokkos::MDRangePolicy<TagZero, ExecSpace, Kokkos::Rank<2>> policyInit({0,0}, {numCrows, numCcols});
418       Kokkos::parallel_for("Initialize C for Dot Product Based GEMM", policyInit, *this);
419     }
420     else if(beta != CVT::one()) {
421       Kokkos::MDRangePolicy<TagInit, ExecSpace, Kokkos::Rank<2>> policyInit({0,0}, {numCrows, numCcols});
422       Kokkos::parallel_for("Initialize C for Dot Product Based GEMM", policyInit, *this);
423     }
424
425     // Multiply alpha*A^TB and add it to beta*C
426     if(conjugateTranspose) {
427       Kokkos::TeamPolicy<TagMultCT, ExecSpace> policyMult(numTeams, Kokkos::AUTO);
428       Kokkos::parallel_for("Perform Dot Product Based GEMM", policyMult, *this);
429     }
430     else{
431       Kokkos::TeamPolicy<TagMult, ExecSpace> policyMult(numTeams, Kokkos::AUTO);
432       Kokkos::parallel_for("Perform Dot Product Based GEMM", policyMult, *this);
433     }
434   }
435
436   KOKKOS_INLINE_FUNCTION
437   void operator() (const TagZero&, const size_C &rowId, const size_C &colId ) const {
438     C(rowId, colId) = CVT::zero();
439   }
```

A KOKKOS EXAMPLE - GEMM

- Initialization: `C = beta C`

```
// Initialize C matrix if beta != 1
if(beta == CVT::zero()) {
  Kokkos::MDRangePolicy<TagZero, ExecSpace, Kokkos::Rank<2>> policyInit({0,0}, {numCrows, numCcols});
  Kokkos::parallel_for("Initialize C for Dot Product Based GEMM", policyInit, *this);
}
else if(beta != CVT::one()) {
  Kokkos::MDRangePolicy<TagInit, ExecSpace, Kokkos::Rank<2>> policyInit({0,0}, {numCrows, numCcols});
  Kokkos::parallel_for("Initialize C for Dot Product Based GEMM", policyInit, *this);
}
```

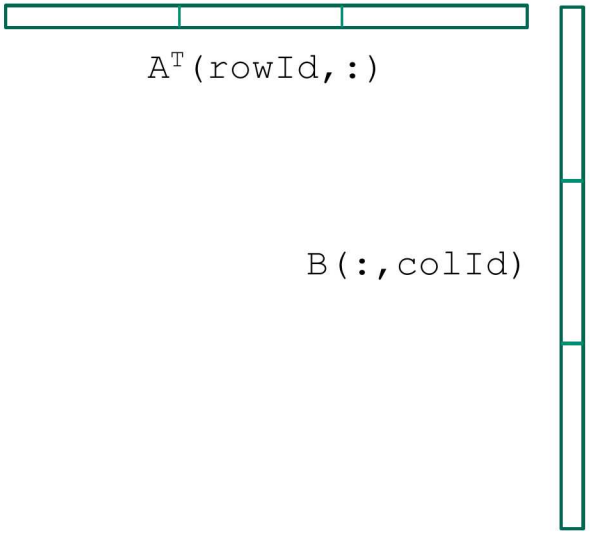
```
KOKKOS_INLINE_FUNCTION
void operator() (const TagZero&, const size_C &rowId, const size_C &colId ) const {
  C(rowId, colId) = CVT::zero();
}

KOKKOS_INLINE_FUNCTION
void operator() (const TagInit&, const size_C &rowId, const size_C &colId ) const {
  C(rowId, colId) = beta * C(rowId, colId);
}
```

A KOKKOS EXAMPLE - GEMM

- Multiplication $C = C + alpha\ A^T\ B$

```
// Multiply alpha*A^TB and add it to beta*C
if(conjugateTranspose) {
  Kokkos::TeamPolicy<TagMultCT, ExecSpace> policyMult(numTeams, Kokkos::AUTO);
  Kokkos::parallel_for("Perform Dot Product Based GEMM", policyMult, *this);
}
else{
  Kokkos::TeamPolicy<TagMult, ExecSpace> policyMult(numTeams, Kokkos::AUTO);
  Kokkos::parallel_for("Perform Dot Product Based GEMM", policyMult, *this);
}
```



- Each dot product is performed by `numDivPerDot` teams
- Total number of teams:

```
numCrows*numCcols*numDivPerDot
```

A KOKKOS EXAMPLE - GEMM

- Multiplication $C = C + alpha\ A^T\ B$

```
KOKKOS_INLINE_FUNCTION
void operator() (const TagMult&, const typename Kokkos::TeamPolicy<>::member_type& teamMember) const {

  const size_C globalRank = teamMember.league_rank();
  const size_C localRank = globalRank % numDivPerDot;
  const size_C i = globalRank / numDivPerDot;
  const size_C rowId = i / numCcols;
  const size_C colId = i % numCcols;

  scalar_C result = CVT::zero();
  const size_A baseInd = chunkSize*localRank;
  Kokkos::parallel_reduce( Kokkos::TeamThreadRange(teamMember, chunkSize), [&]( const size_A k, scalar_C &update ) {
      if(baseInd + k < dotSize)
        update += alpha * A(baseInd+k, rowId) * B(baseInd+k, colId);
    }, result );

  Kokkos::single(Kokkos::PerTeam(teamMember), [&] () {
    Kokkos::atomic_add(&C(rowId, colId), result);
    });
}
```