SAND2018-13126C

Early Experience with NVSHMEM

Extending the Kokkos Programming Model with PGAS Semantics

Christian Trott

Sandia National Laboratories

www.github.com/kokkos

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and 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-NA-0003525.

EXASCALE COMPUTING PROJECT

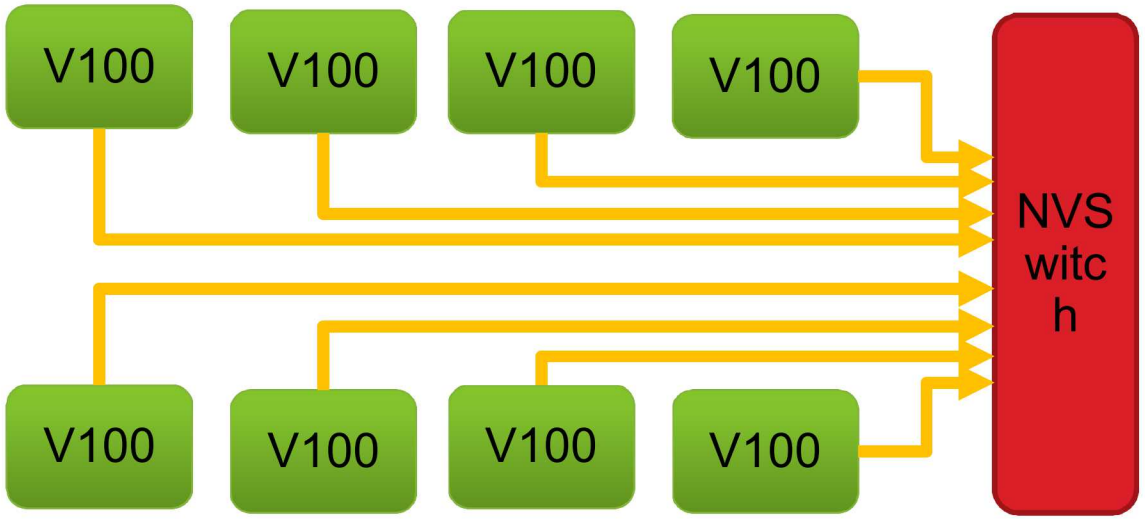
National Nuclear Security Administration

U.S. DEPARTMENT OF ENERGY | Office of Science

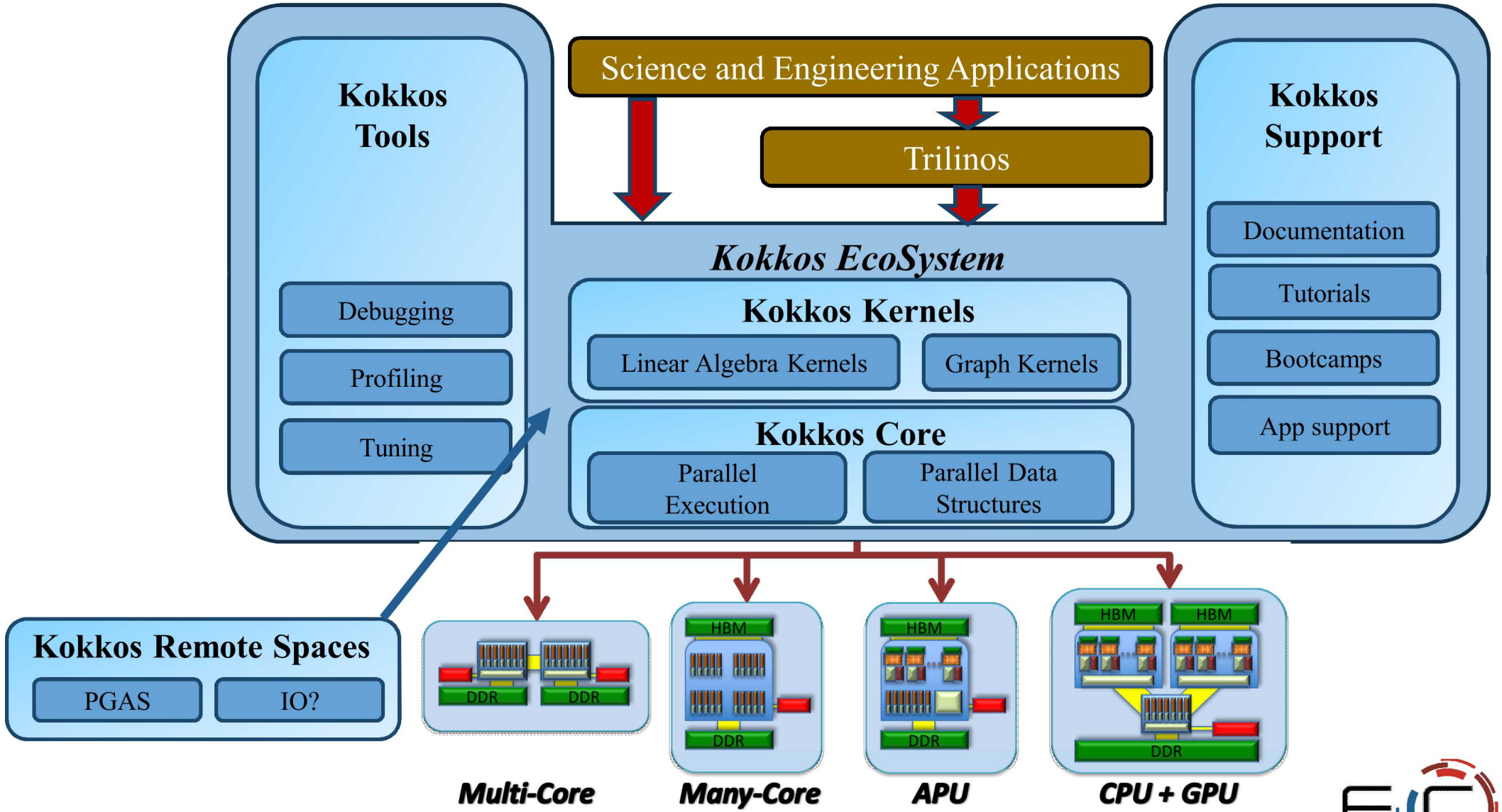
SAND2018-XXXX PE

Why PGAS Models May Now Work

- Networks become more memory fabric like
- Hierarchical system designs with “SuperNodes”
  - Summit: 6GPUs per Node less than 5000 Nodes for full machine
- DGX2: 16 GPUs interconnected with 300GB/s NVLINK



The Kokkos EcoSystem



Kokkos Users DOE

- We don't actually know who all is using (or trying) Kokkos.
- Labs included: SNL, ORNL, LANL, PNNL, NREL, LBL, ANL. Partial ECP Project List:

Application	State
SNL ATDM Apps	Base Code (SPARC, EMPIRE, Nimble, SPARTA, …)
LANL ATDM Apps	In Parts
EXAALT	Base Code
QMCPack	Evaluation
ExaWind	Base Code
ExaAM	Experimenting
LatticeQCD	Experimenting
ProxyApp	Base Code (in parts)
COPA	Base Code
ExaGraph	Base Code (in parts)
ExaLearn	Committed (in parts)

Software Technology	State
SNL ATDM PMR	This is Kokkos ;-)
LANL ATDM PMR	Experimenting
KokkosSupport	
SNL ATDM DevTools	Base Code (in parts)
ExaPapi	Integration with KokkosTools
SNL ATDM Math	BaseCode
ForTrilinos	BaseCode
PEEKS	Base Code

Additionally: Many ASC applications at Sandia are porting or using Kokkos in their base code.

Kokkos Users In the Greater HPC Community

- Many Institutions outside of DOE started experimenting with Kokkos or have projects which are already committed
- This does not include projects who only use Kokkos indirectly via Trilinos solvers

Extending Kokkos With PGAS Semantics

- Kokkos View: Multi-Dimensional Array with Layout and MemorySpace
  - View<double**[3], LayoutLeft, CudaSpace> a(“A”,N,M);
  - a(i,j,1) = 5.0;
  - Can return meta-object: e.g. View<double*,MemoryTraits<Atomic>>
- Idea: Add new memory spaces which return data handles with shmem semantics
  - View<double**[3], LayoutLeft, NVShmemSpace> a(“A”,N,M);
  - Operator a(i,j,k) returns:

```
template<>
struct NVShmemElement<double> {
  NVShmemElement(int pe_, double* ptr_):pe(pe_),ptr(ptr_) {}
  int pe; double* ptr;
  void operator = (double val) { shmem_double_p(ptr,val,pe); }
};
```

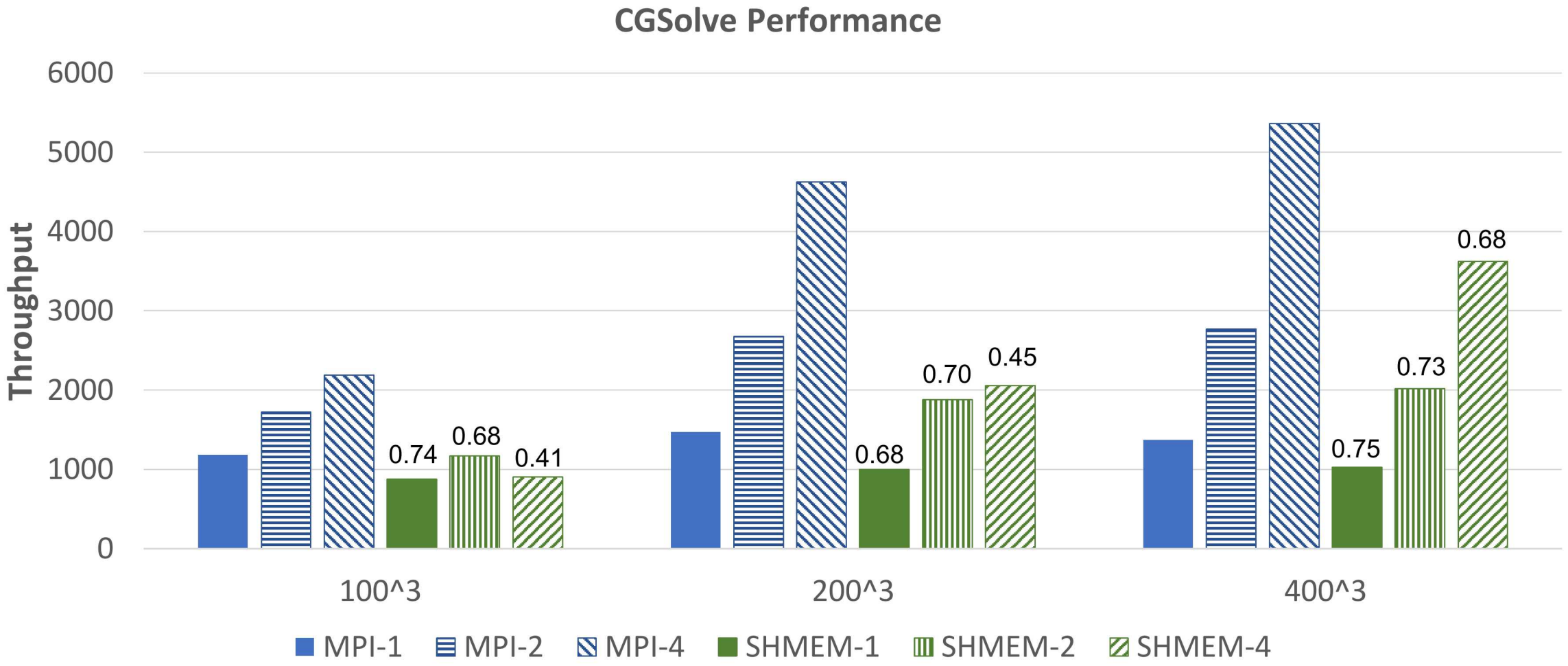
A Test Case: CG-Solve

- Using MiniFE Input Matrix
  - Compare against MPI MiniFE/Cuda implementation
    - Well optimized: communication hiding algorithm + ELL Matrix Format instead of CSR
- A series of sparse-matrix vector multiply (SPMV), dot product, and vector adds
  - Only remote accesses are inside SPMV: y = A * x;
  - No change from serial kernel.

```
parallel_for("SPMV", Kokkos::TeamPolicy<>((nrows+rows_per_team-1)/rows_per_team,team_size,8),
  KOKKOS_LAMBDA (const Kokkos::TeamPolicy<>::member_type& team) {
  const int64_t first_row = team.league_rank()*rows_per_team;
  const int64_t last_row = first_row+rows_per_team<nrows ? first_row+rows_per_team:nrows;
  parallel_for(TeamThreadRange(team,first_row,last_row), [&] (const int64_t row) {
    int64_t row_start=A.row_ptr(row);
    int64_t row_length=A.row_ptr(row+1)-row_start;

    double y_row;
    parallel_reduce(ThreadVectorRange(team,row_length), [&] (const int64_t i,double& sum) {
      int64_t idx = A.col_idx(i+row_start);
      sum += A.values(i+row_start)*x(idx);
    },y_row);
    y(row) = y_row;
  });
});
```

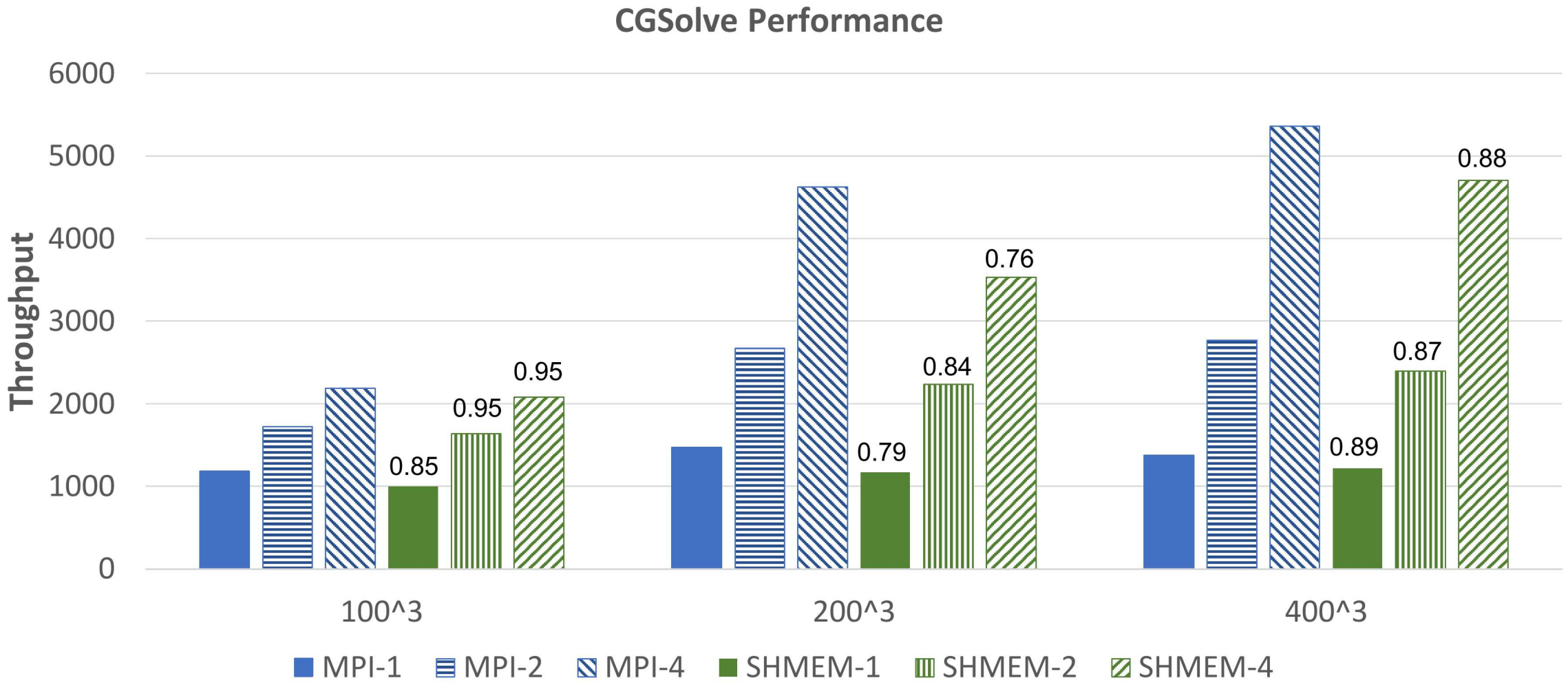
Performance



Optimize Away Non-Inline Function Calls

- Store array of remote pointers in the View
  - Implemented via specialization of the pointer type
  - Obtained via shmem_ptr(pe,ptr) call during view construction
- Return type of View is now simple scalar reference (e.g. double&)
- Relies on direct accessible of memory

Performance Results – Inline Functions



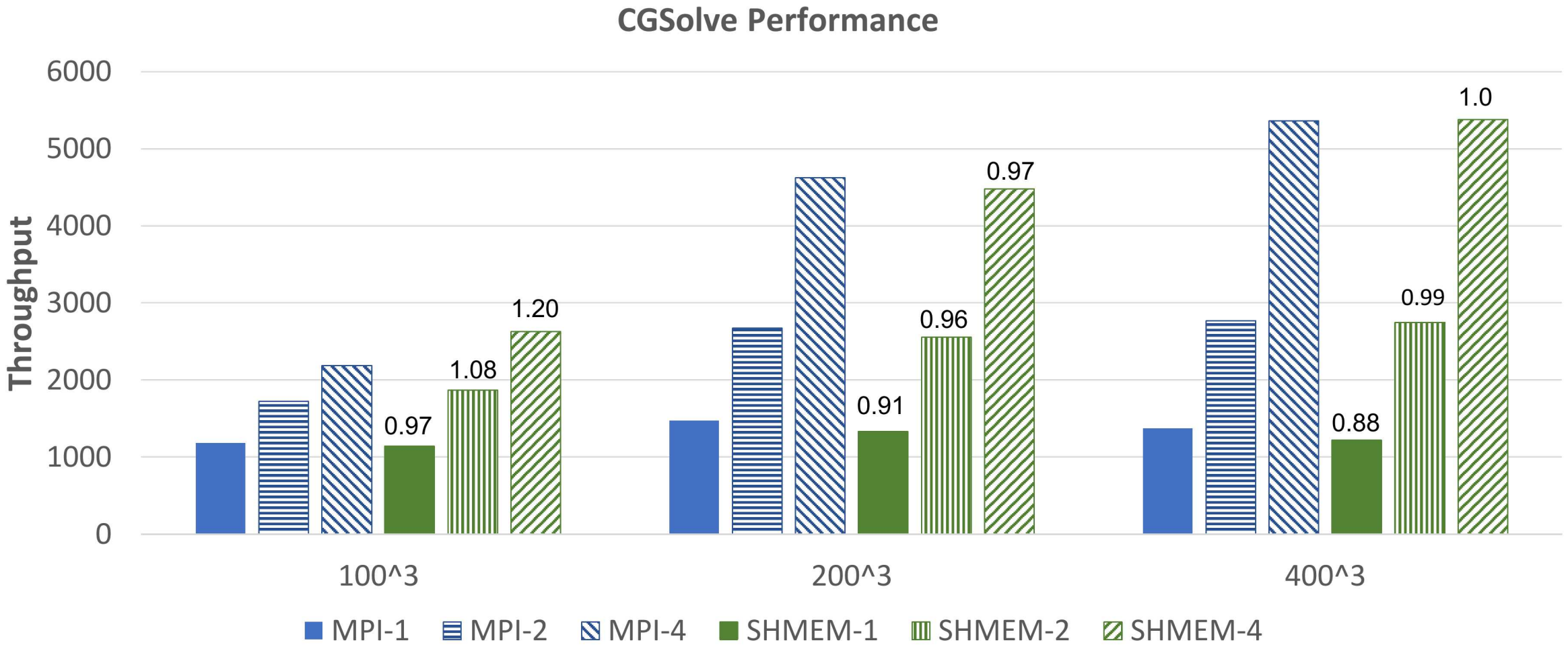
Performance vs Convenience

- Previous implementation did runtime div/mod operation in data access
- Performance can be improved by using explicit Rank / Offset access
  - Not too bad: encode rank in the Matrix Ordinal as:
    - idx/local_length * MASK + idx%local_length
    - Index as: x(idx/MASK, idx%MASK)
    - With MASK compile time power of two: index manipulation are shift operations

```
parallel_for("SPMV", Kokkos::TeamPolicy<>((nrows+rows_per_team-1)/rows_per_team,team_size,8),
  KOKKOS_LAMBDA (const Kokkos::TeamPolicy<>::member_type& team) {
  const int64_t first_row = team.league_rank()*rows_per_team;
  const int64_t last_row = first_row+rows_per_team<nrows ? first_row+rows_per_team:nrows;
  parallel_for(TeamThreadRange(team,first_row,last_row), [&] (const int64_t row) {
    int64_t row_start=A.row_ptr(row);
    int64_t row_length=A.row_ptr(row+1)-row_start;

    double y_row;
    parallel_reduce(ThreadVectorRange(team,row_length), [&] (const int64_t i,double& sum) {
      int64_t idx = A.col_idx(i+row_start);
      sum += A.values(i+row_start)*x(idx/MASK, idx % MASK);
    },y_row);
    y(row) = y_row;
  });
});
```

Performance Results - Explicit Rank Split



Summary

- NVSHMEM even in its prototype stage works and can provide an efficient vehicle for Multi-GPU access in the same node
- Much simpler code than MPI but competitive with highly optimized implementations
- Kokkos Remote Space integration makes it trivial to use for Kokkos applications

Questions:

- crtrott@sandia.gov
- kokkosteam.slack.com
- Issues @ github.com/kokkos/kokkos