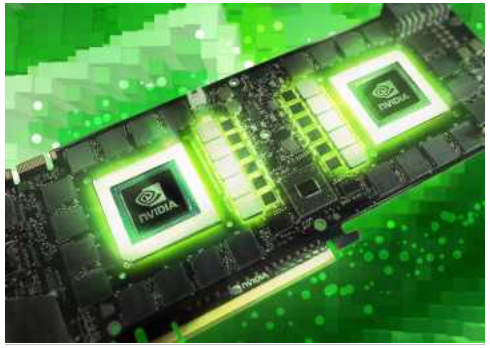
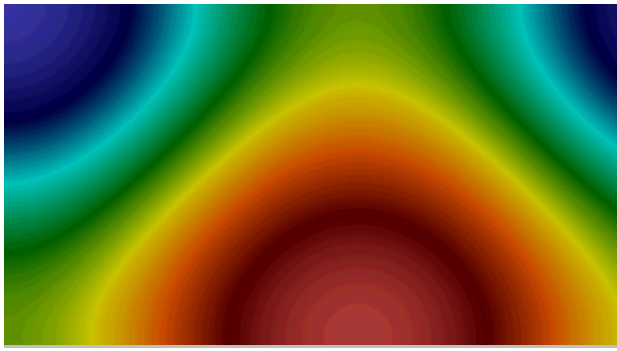
SAND2018-1534C



ACS Technical Exchange: Threaded Assembly in Aria Expressions

Jonathan Clausen, Victor Brunini, Chris Forster, David Noble, Mark Hoemmen, Si Hammond, and Christian Trott

TF Team, STK Team

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

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.

Motivation

Trinity (ATS-I) Phase I
"Haswell" CPUs
2 socket x 16 core x 2 ht
4 wide vector instructions

Trinity Phase II
"Knight's Landing"
1 socket x 68 cores x 4 ht
8 wide vector instructions

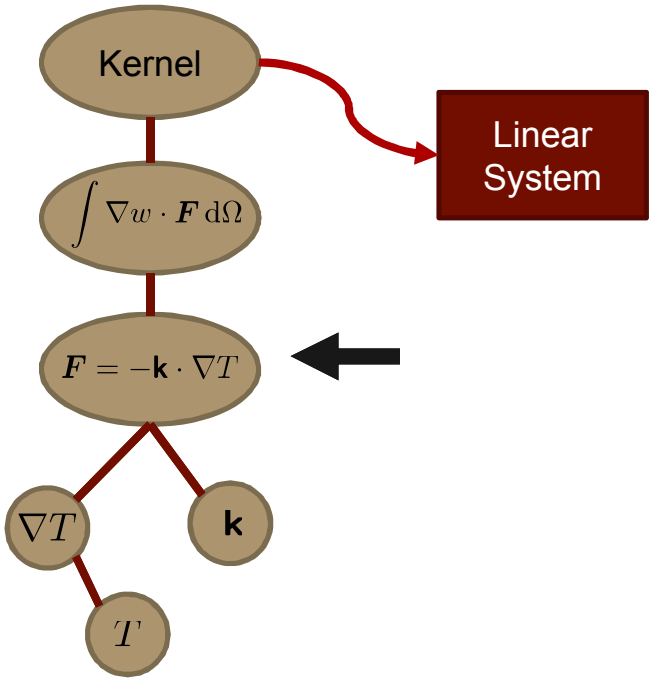
Sierra (ATS-II)
Power9 CPUs
NVIDIA Volta GPUs
120-150 PetaFlops

Technologies

Sandia National Laboratories

- Technologies
  - Threading
  - Vectorization
- Software Abstractions
  - Kokkos
  - stk::simd
- Prototype work in Ariamini

Expression Subsystem



$$\boldsymbol{F} = -\mathbf{k} \cdot \nabla T$$

public:
Expression()
…
void compute_values()
void compute_sensitivities()

private:
Expression_Handle gradT
Expression_Handle k
values
sensitivities //wrt each dof

Expression Subsystem

$$\boldsymbol{F} = -\mathbf{k} \cdot \nabla T$$

public:
Expression()
…
void compute_values()
void compute_sensitivities()

private:
Expression_Handle gradT
Expression_Handle k
values
sens

compute_values()

for each element:
  for each integration point:
    for each dim:
      values(elm, pt, dim)
        = -k(elm,pt)*gradT(elm, pt, dim)

compute_sensitivities()

for each element:
  for each integration point:
    for each dim:
      for each dof:
      sens(elm, pt, dim, dof)
        += -k.sens(…, dof)*gradT(…)
         - k(elem, pt)*gradT.sens(…, dof)

Threading

```
for(bucket : element_buckets) {
  for(workset : bucket) {
    set_mesh_object_couriers(workset_elems);
    evaluate_expressions();
    evaluate_kernels(rhs, lhs);
    sum_into_linear_system(rhs, lhs);
  }
}

//-------------------------
void evaluate_expressions() {
  for(expr : ordered_expressions) {
    expr.prepare_to_recompute(); //resizes and zeros storage if needed
    expr.compute_values();
    expr.compute_sensitivities();
  }
}
```

Threading

```
resize_expression_storage(num_concurrent_worksets);
Kokkos::parallel_for(bucket : element_buckets) {
  for(workset : bucket) {
    start = workset_index * workset_size;
    num_elems = workset_size;
    ElementRange scratch_range(
       start, start + num_elems);
    set_mesh_object_couriers(scratch_range, workset_elems);
    evaluate_expressions(scratch_range);
    evaluate_kernels(scratch_range, rhs, lhs);
    atomic_sum_into_linear_system(rhs, lhs);
  }
}

//------------------------
void evaluate_expressions(ElementRange elem_range) {
  for(expr : ordered_expressions) {
    expr.prepare_to_recompute(elem_range); //resizes and zeros storage if needed
    expr.compute_values(elem_range);
    expr.compute_sensitivities(elem_range);
  }
}
```

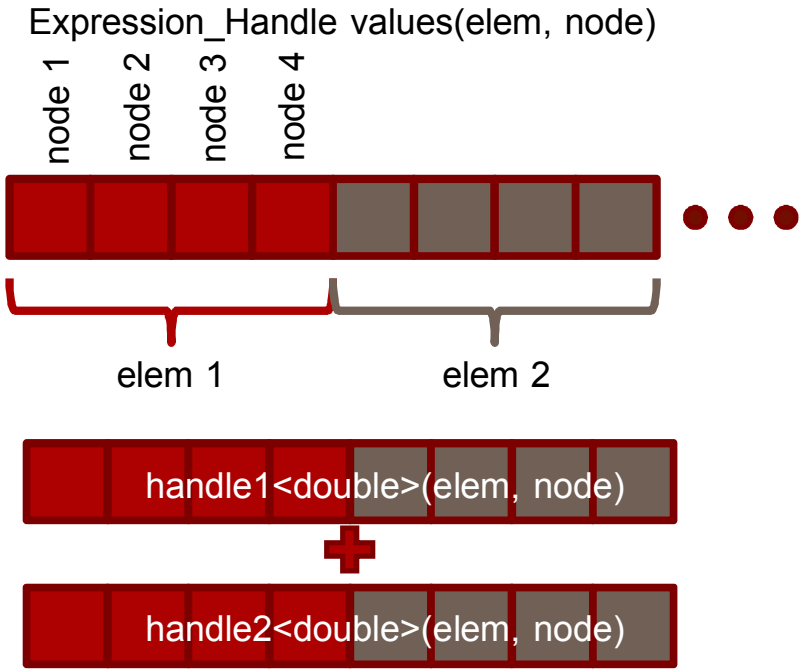
Threading

```
void Scalar_Vector_Product_Expression::compute_values() {
  for(int elem = 0; elem < nelem; ++elem) {
    for(int ip = 0; ip < nip; ++ip) {
      for(int dim = 0; dim < ndim; ++dim) {
        values(elem, ip, dim) =
          scalar(elem, ip) * vector(elem, ip, dim);
      }
    }
  }
}
//------
void Scalar_Vector_Product_Expression::compute_sensitivities()
{
  //...
}
```

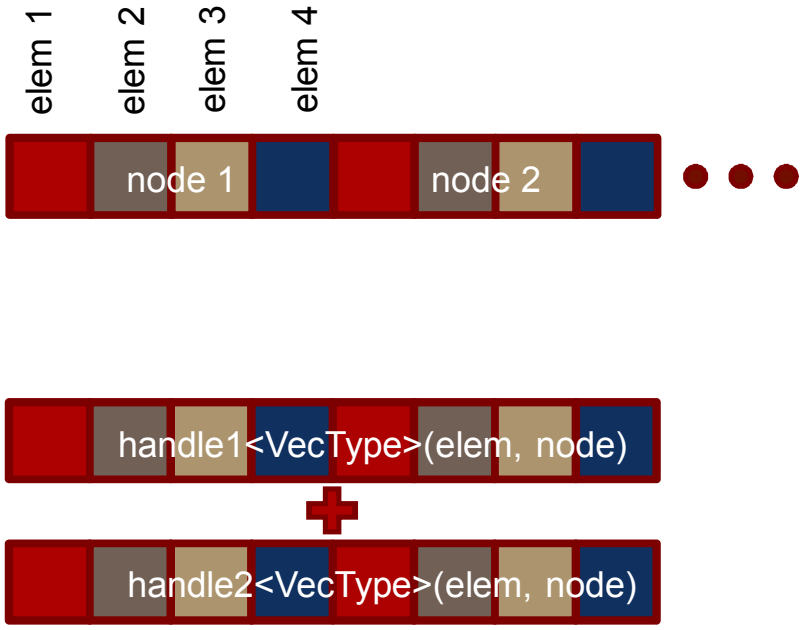
Threading

```
void Scalar_Vector_Product_Expression::compute_values(ElemRange elem_range) {
  for(auto elem : elem_range) {
    for(auto ip : my_sizes.get_points()) {
      for(auto dim : my_sizes.get_dims()) {
        values(elem, ip, dim) =
          scalar(elem, ip) * vector(elem, ip, dim);
      }
    }
  }
}
//------
void Scalar_Vector_Product_Expression::compute_sensitivities(ElemRange elem_range)
{
  //...
}
```

Vectorization



4 elem x 4 node = 16 scalar additions



4-wide intrinsic: 4 elem x 4 node = 4 vector additions

Vectorization

```
resize_expression_storage(num_concurrent_worksets);
Kokkos::parallel_for(bucket : element_buckets) {
  for(workset : bucket) {
    start = workset_index * workset_size;
    num_elems = workset_size;
    ElementRange scratch_range(
       start, start + num_elems);
    set_mesh_object_couriers(scratch_range, workset_elems);
    evaluate_expressions(scratch_range);
    evaluate_kernels(scratch_range, rhs, lhs);
    atomic_sum_into_linear_system(rhs, lhs);
  }
}

//------------------------
void evaluate_expressions(ElementRange elem_range) {
//...
```

Vectorization

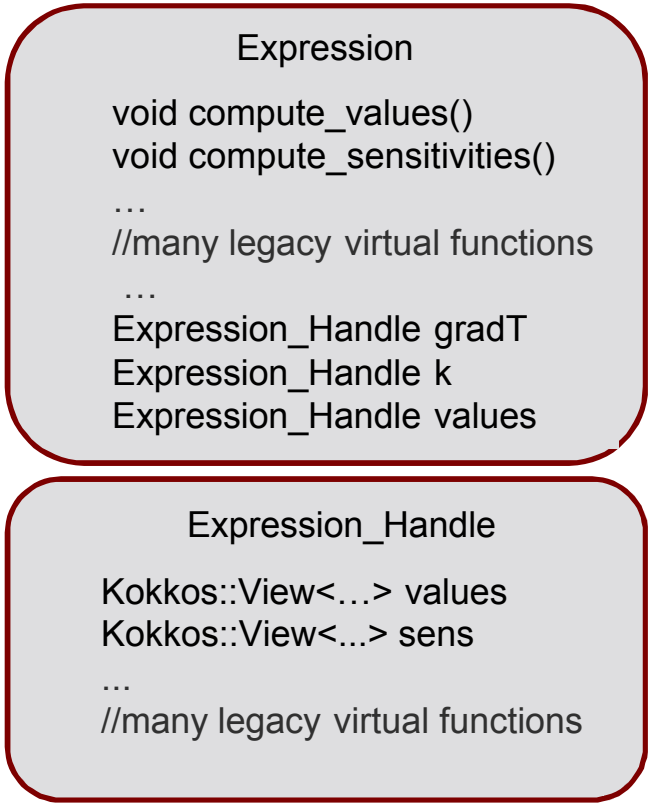
```
resize_expression_storage(num_concurrent_worksets);
Kokkos::parallel_for(bucket : element_buckets) {
  for(workset : bucket) {
    start = workset_index * workset_size / VecLength;
    remainder = worset_size % VecLength;
    num_simd_elems = workset_size / VecLength;
    if (remainder) num_simd_elems++; //Pad if remainder
    ElementRange scratch_range(
       start, start + num_simd_elems);
    set_mesh_object_couriers(scratch_range, workset_elems);
    evaluate_expressions(scratch_range);
    evaluate_kernels(scratch_range, rhs, lhs);
    atomic_sum_into_linear_system(rhs, lhs);
  }
}

//------------------------
void evaluate_expressions(ElementRange elem_range) {
//...
```

Intrusiveness

- Vectorization
  - Most expressions remained unchanged
  - Gather from stk fields
  - Scatter into linear system
  - Interface with TPLs
- Threading
  - Expressions receive ElementRange chunk of elements
  - Expression_Handles now wrap Kokkos::Views

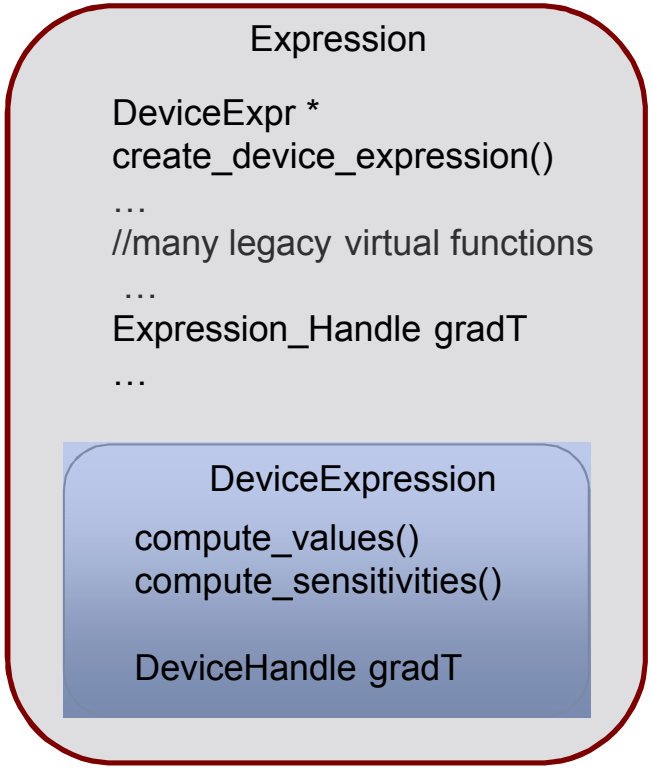
Code Changes for GPU

Sandia National Laboratories



- Nested parallelism
- Virtual dispatch requires objects allocated on device
- Device classes must be copyable (relocatable-device code)
- Caching of member variables to local (places in register)
- Nested classes providing device-specific version of class

Code Changes for GPU



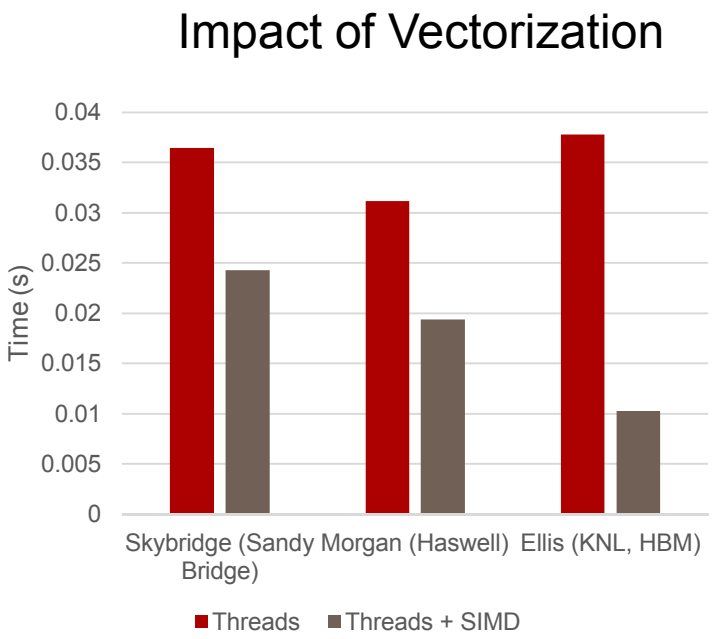
DeviceExpression Allocation

```
ExprType * p
  = static_cast<ExprType *>
    (Kokkos::kokkos_malloc<>(sizeof(ExprType)));
Kokkos::parallel_for(
    Kokkos::RangePolicy<>(0, 1),[=] DEVICE(const int i)
{
  new (p) ExprType(rhs);
});
Kokkos::fence();
```

constructor for DeviceExpression sets up DeviceHandles
DeviceHandles and DeviceExpression UVM accessible

Ariamini

- **prototype iterations**
- **vectorization crucial**
- **threading less clear on KNL**
- **GPU**
- SAND2017-9807PE



Aria Performance

Sandia National Laboratories

Milestone problems (from NGP performance test suite)

8M element conduction
- conduction only
- one block
- volume and surface assembly
- Dirichlet BCs

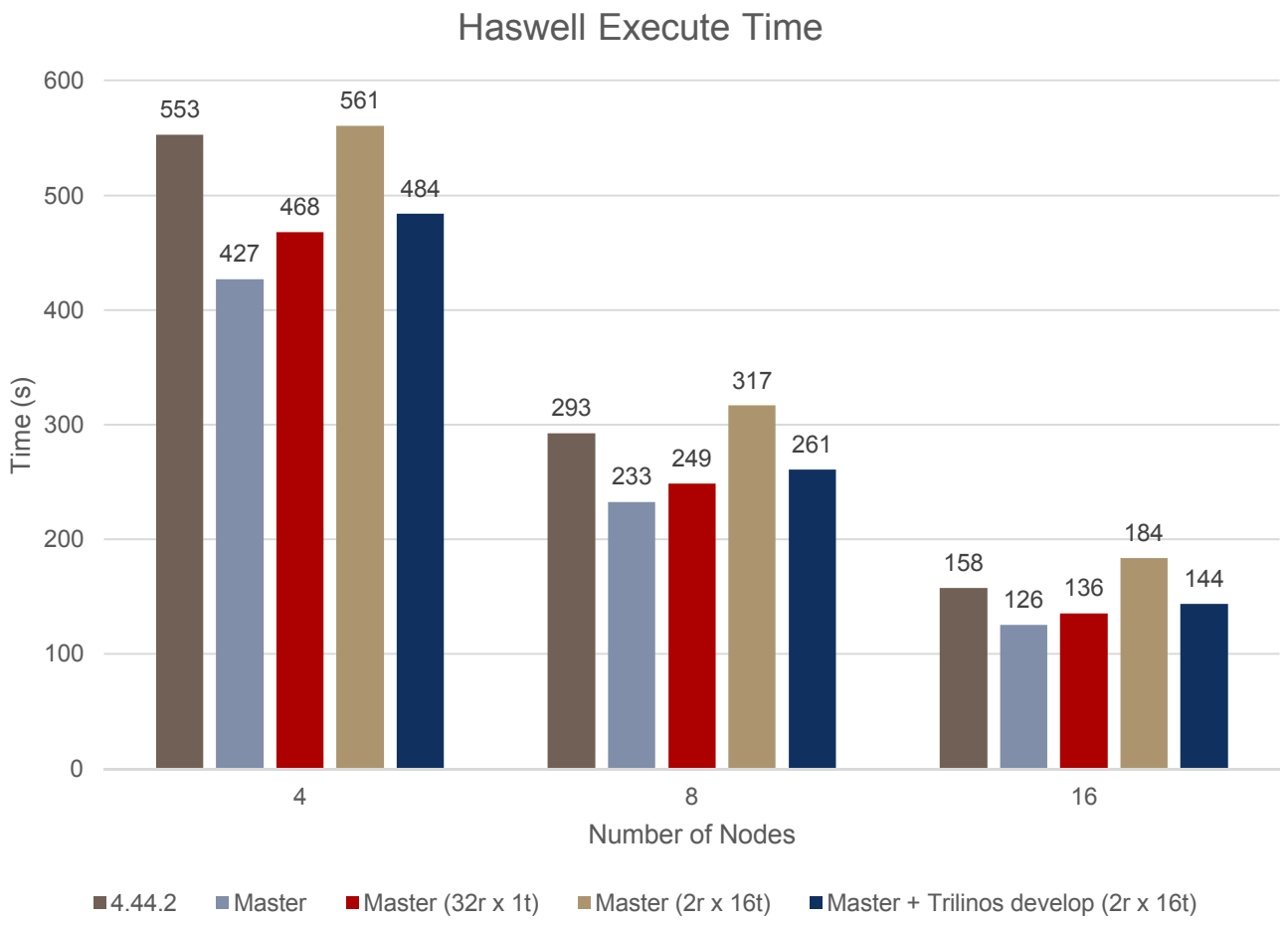
AFF performance test
- 3.5M elements
- 318 blocks
- conduction and radiation
- volume and surface assembly
- Dirichlet BCs

Aria vs. Ariamini

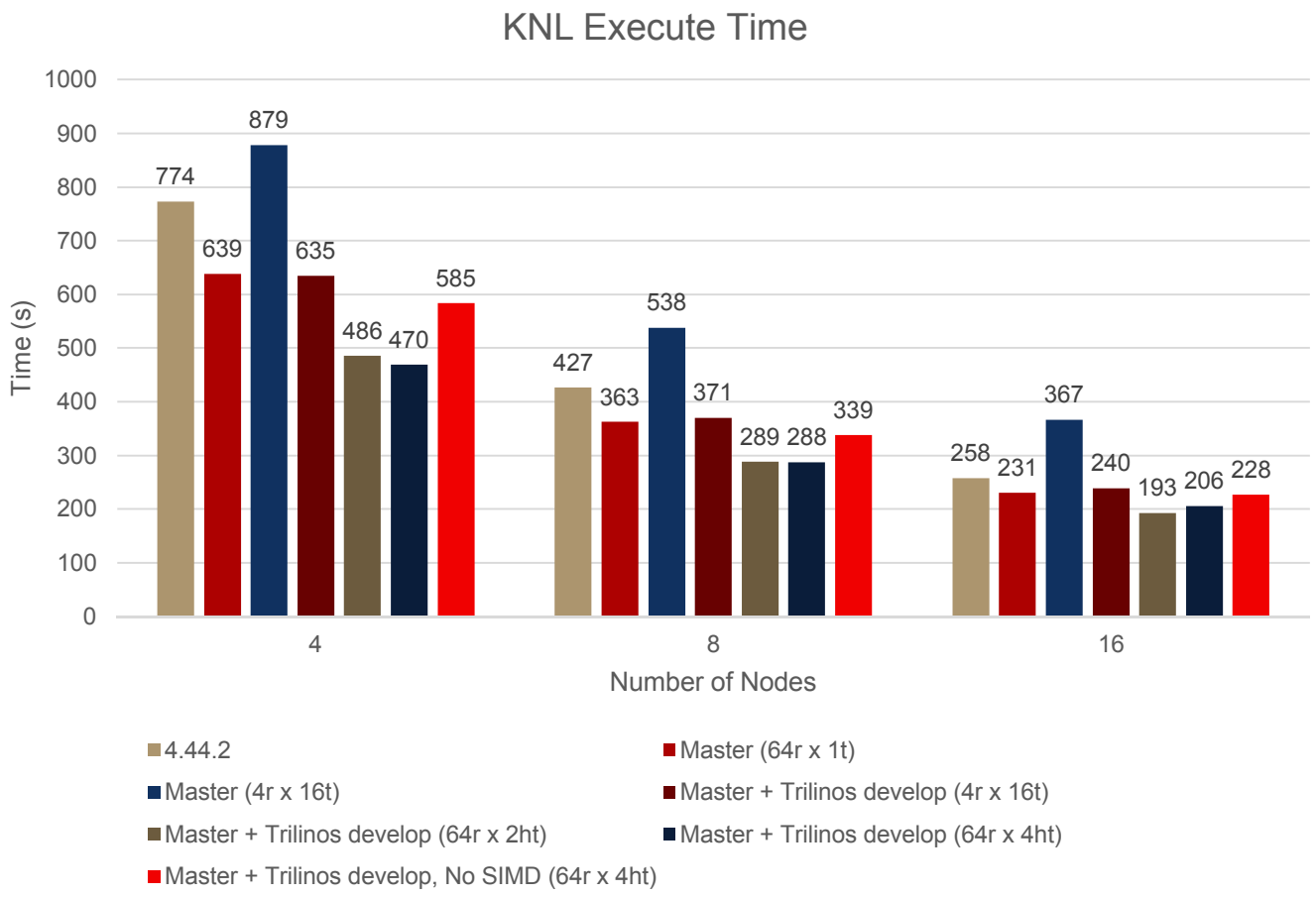
Sandia National Laboratories

- optimization not included in Aria
  - master element
  - compile-time sizes everywhere
- application code much more complex
- ”real” problems and load balancing
  - volume algorithm now threads over all blocks
- load complete
- Dirichlet BCs

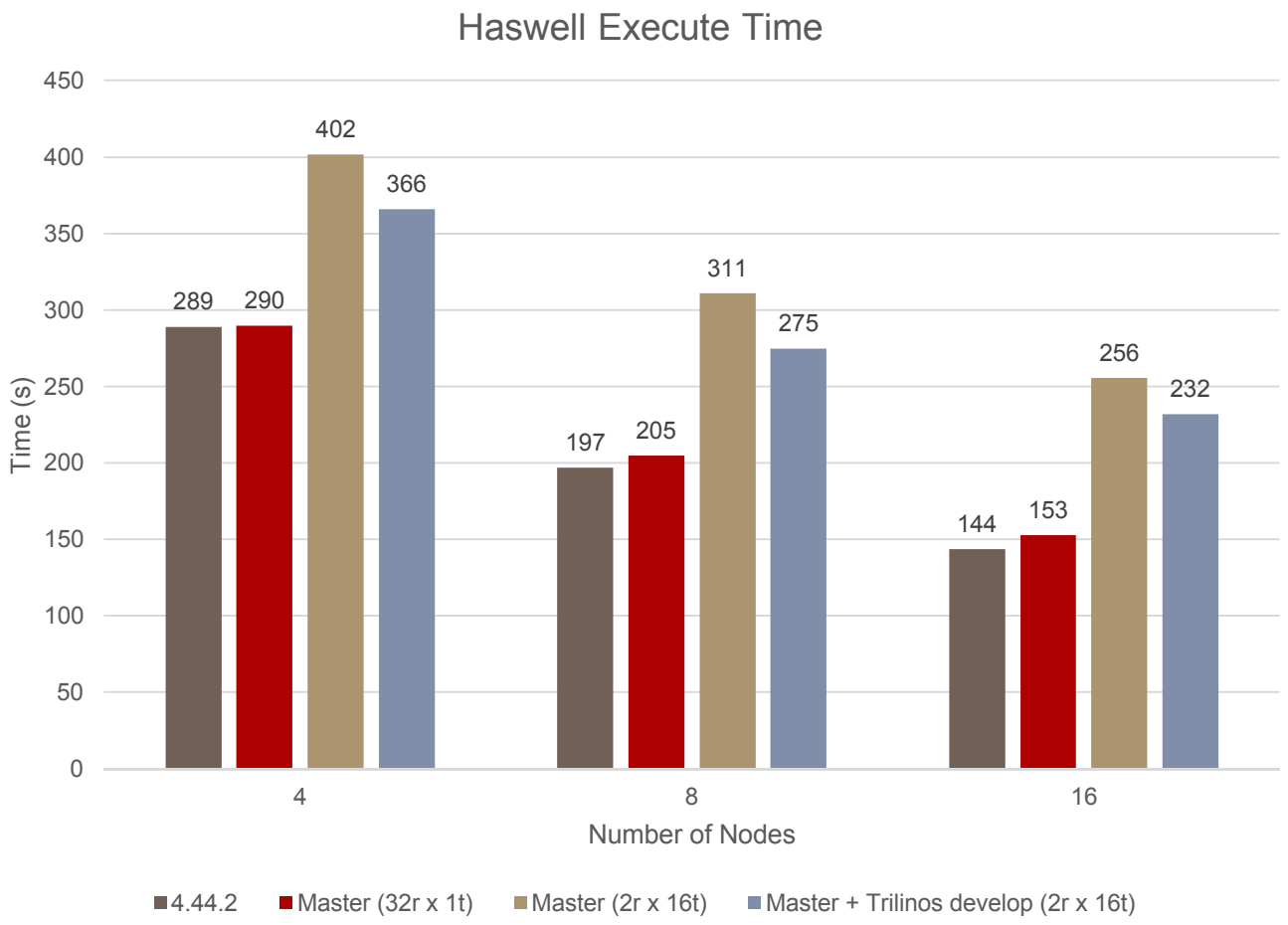
8M Element Conduction



8M Element Conduction

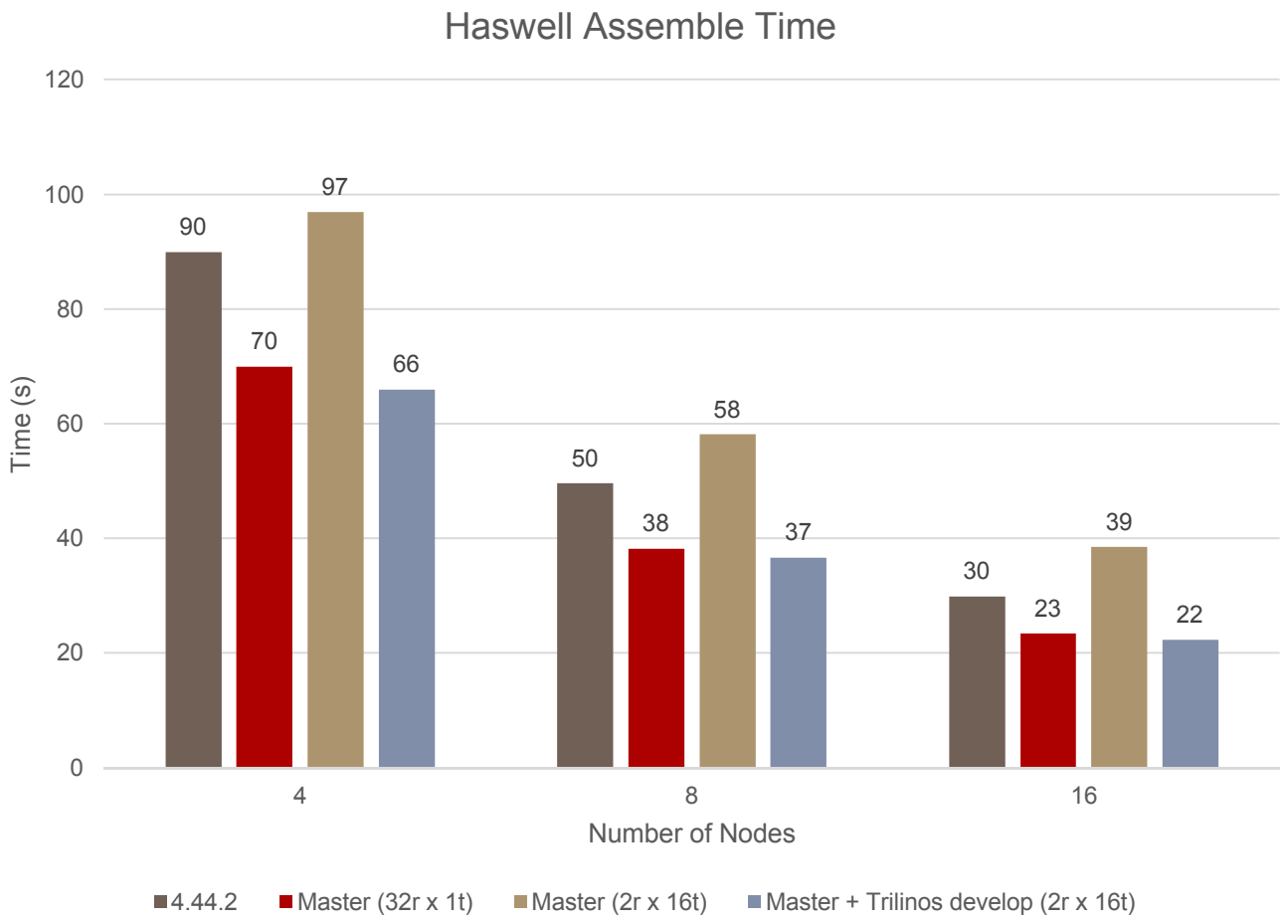


AFF Test Case



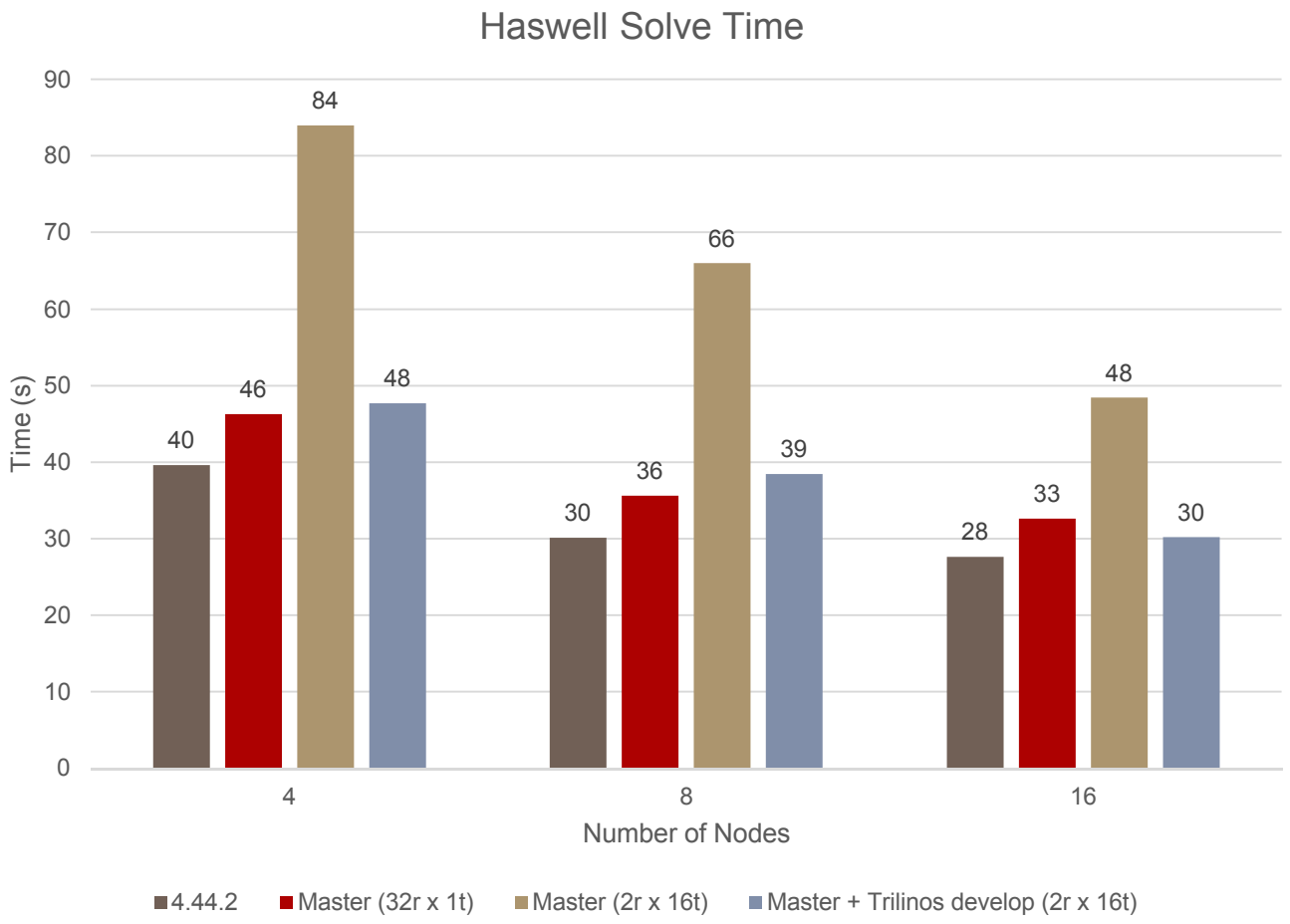
Includes non-threaded enclosure radiation

AFF Test Case



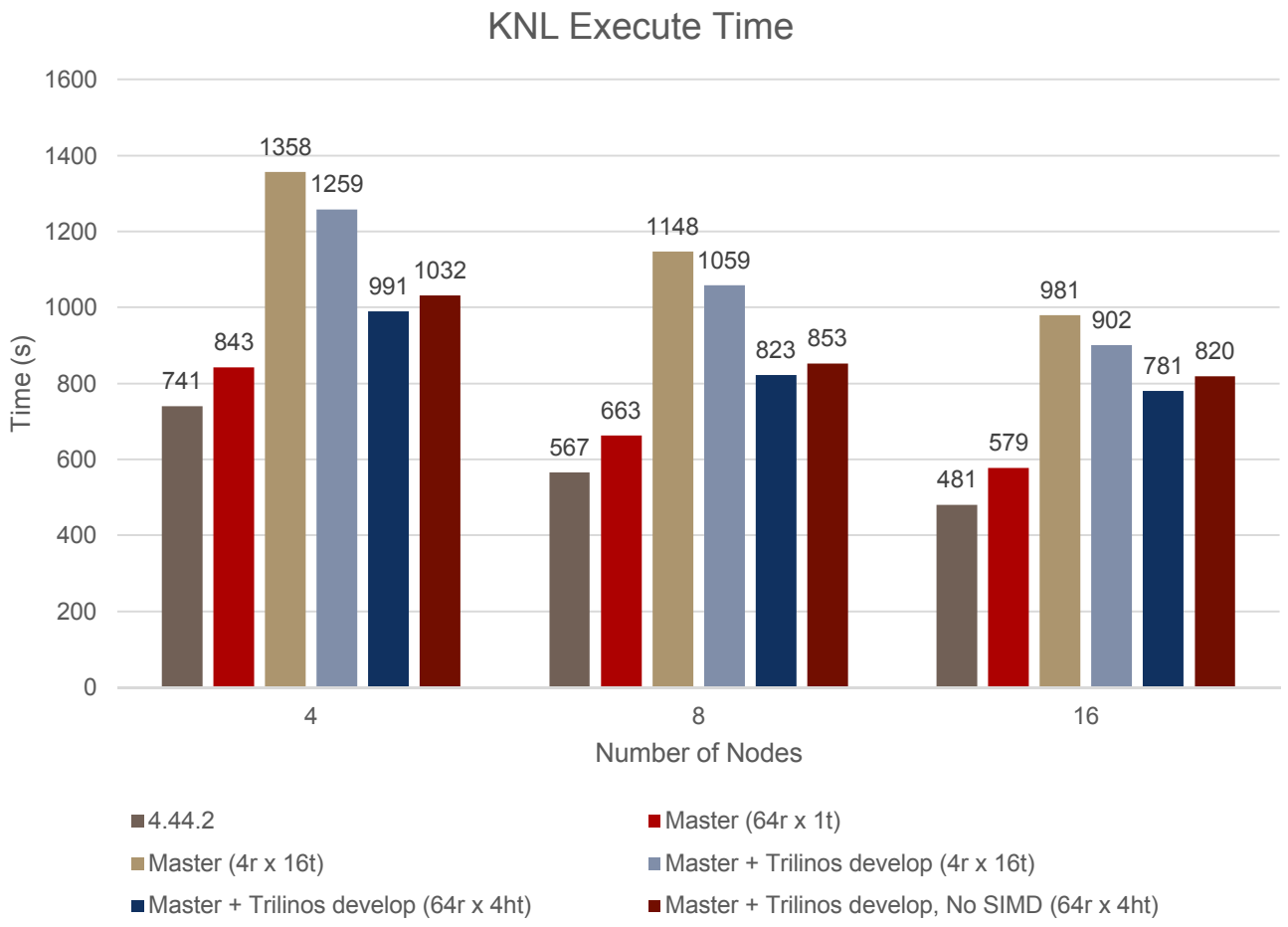
Includes non-threaded enclosure radiation

AFF Test Case



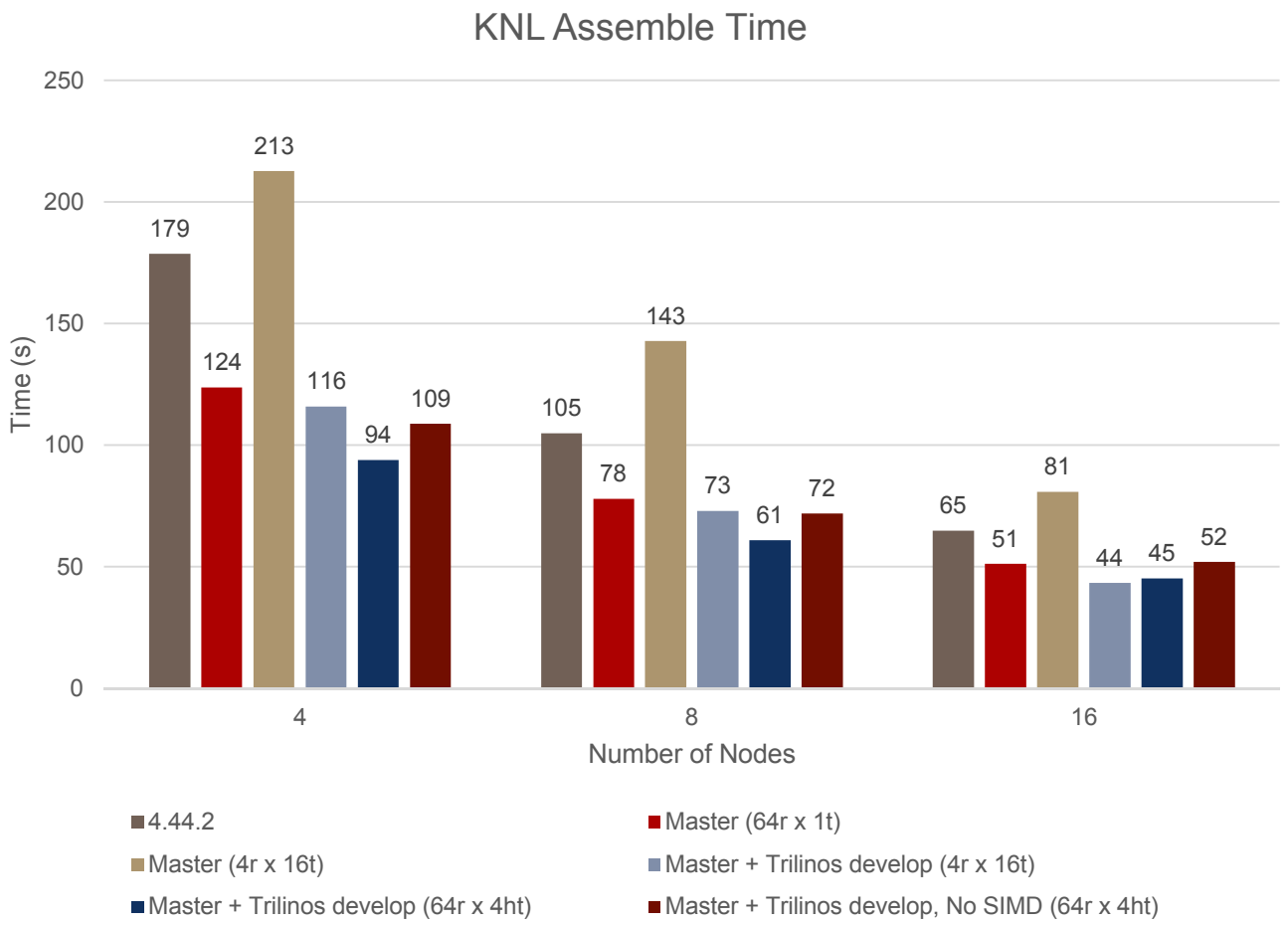
Includes non-threaded enclosure radiation

AFF Test Case

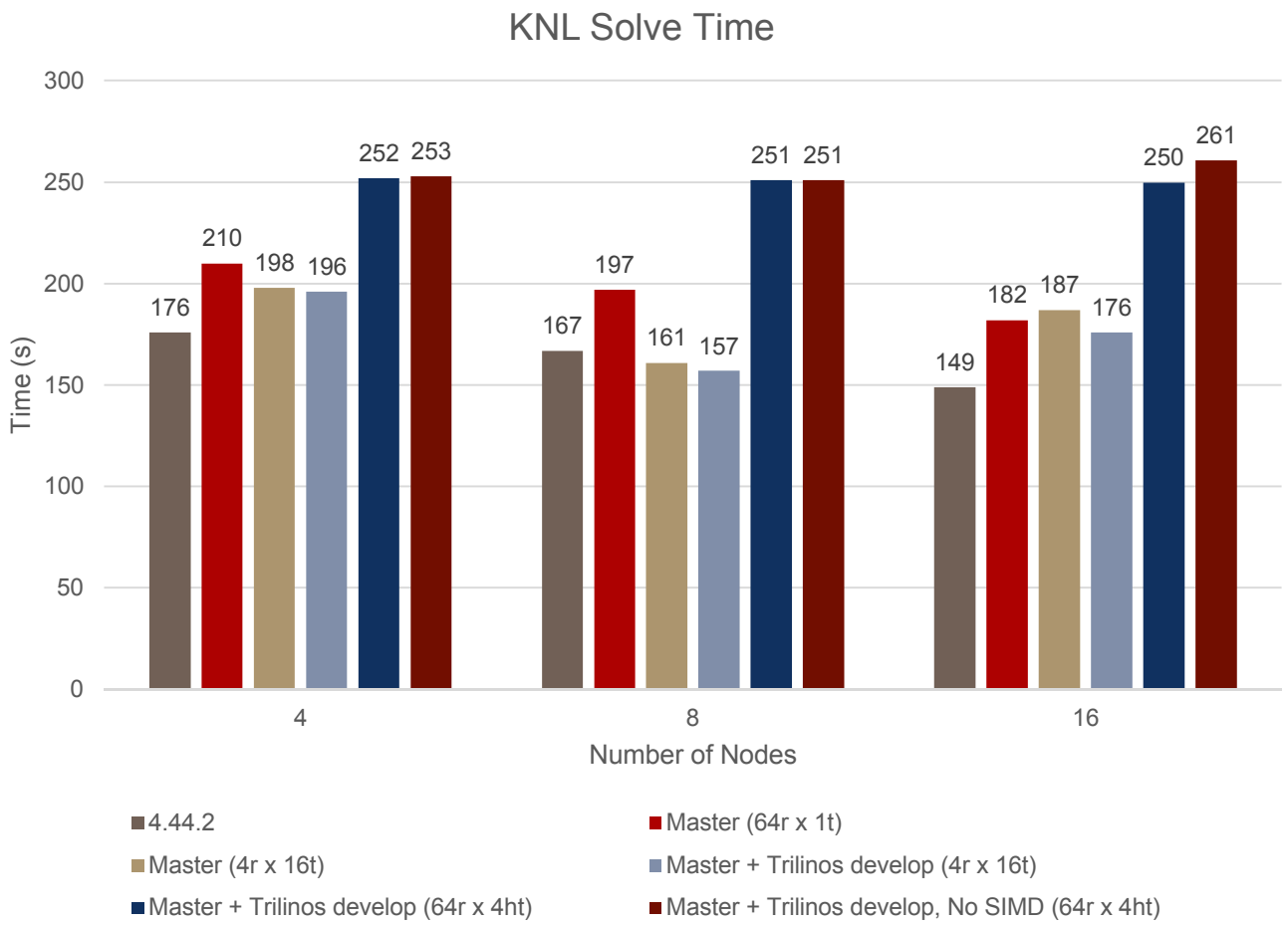


Includes non-threaded enclosure radiation

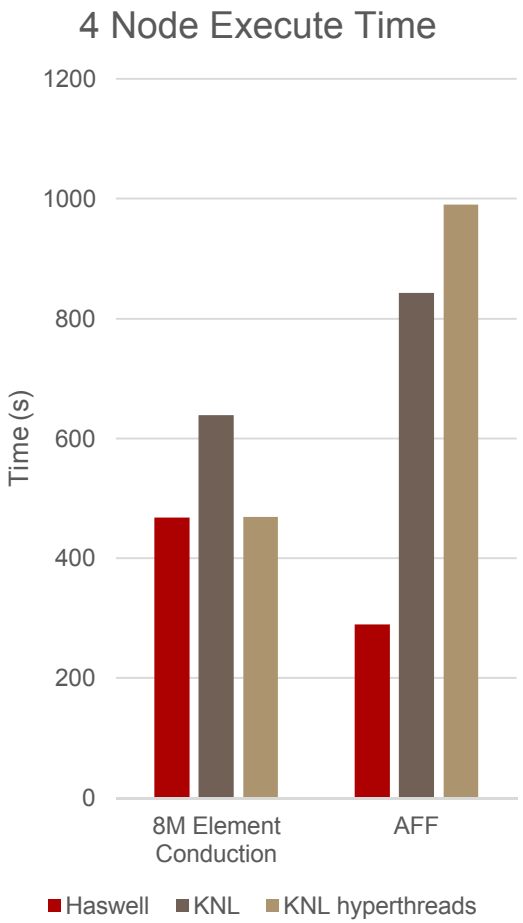
AFF Test Case

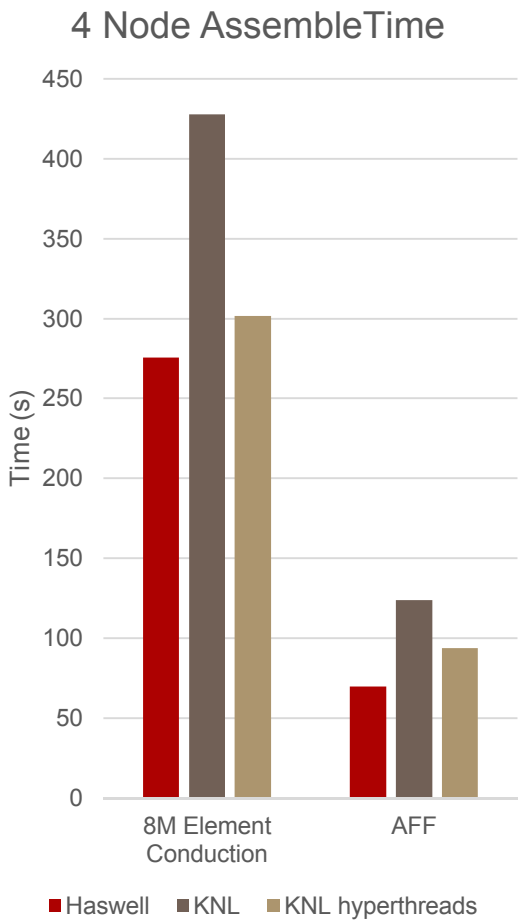


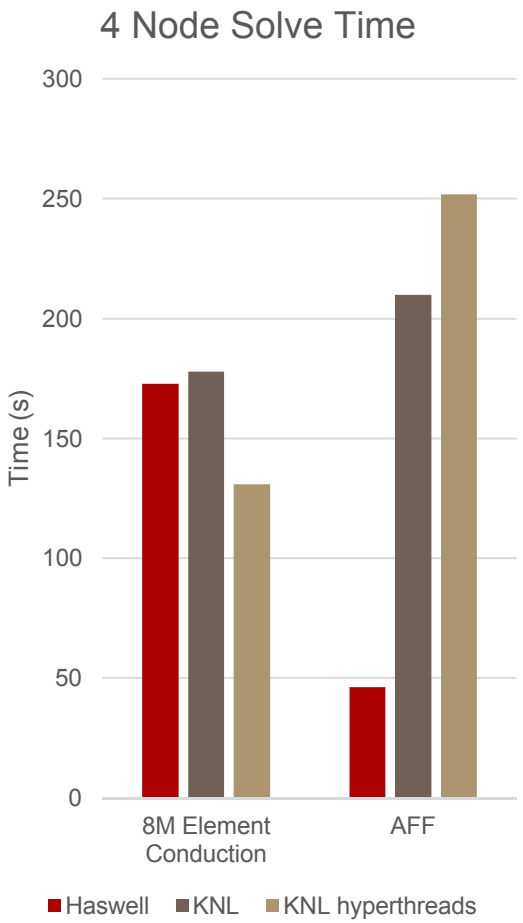
AFF Test Case



Haswell vs KNL comparisons







Haswell: 32r x 1t; KNL: 64r x 1t; KNL hyperthreads: 64r x 4ht

Conclusions

- Ariamini
- Aria transition
- Speedup vs 4.44
- Vectorization
- Threading
  - Haswell
  - KNL

For more information: SAND2017-9807PE

Questions?

Sandia National Laboratories

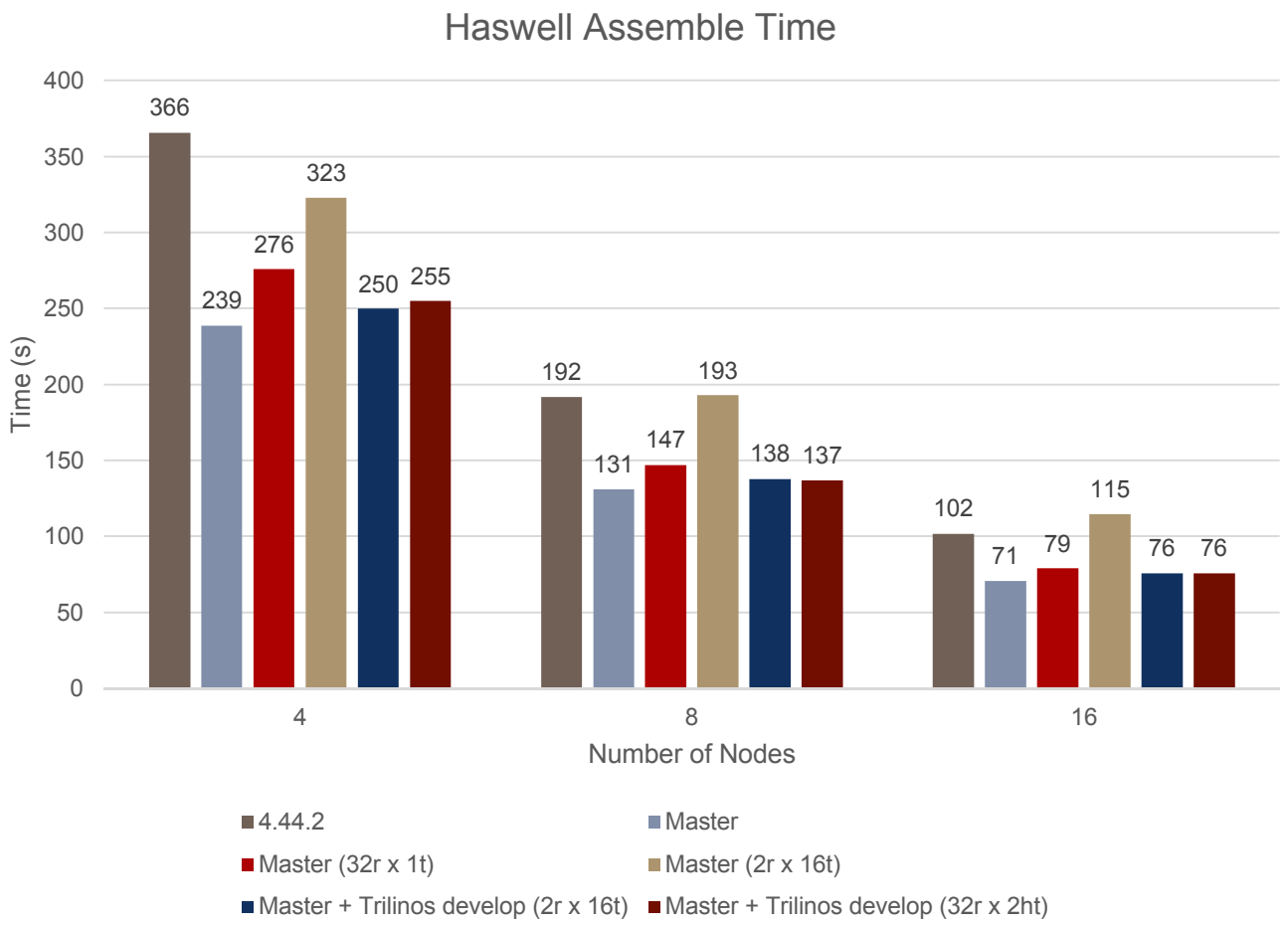
Questions?

Sandia National Laboratories

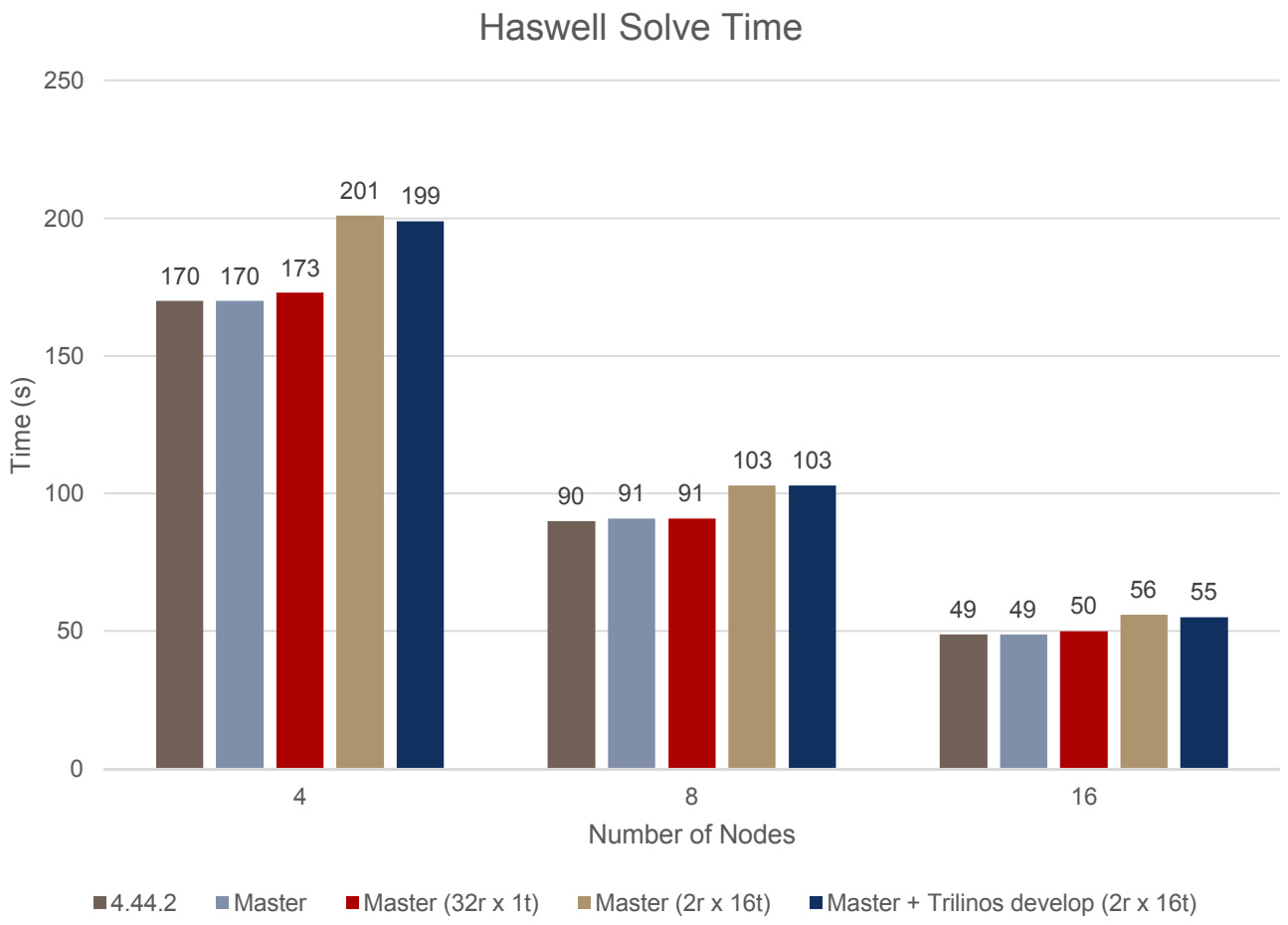
Sandia National Laboratories

Backup Slides

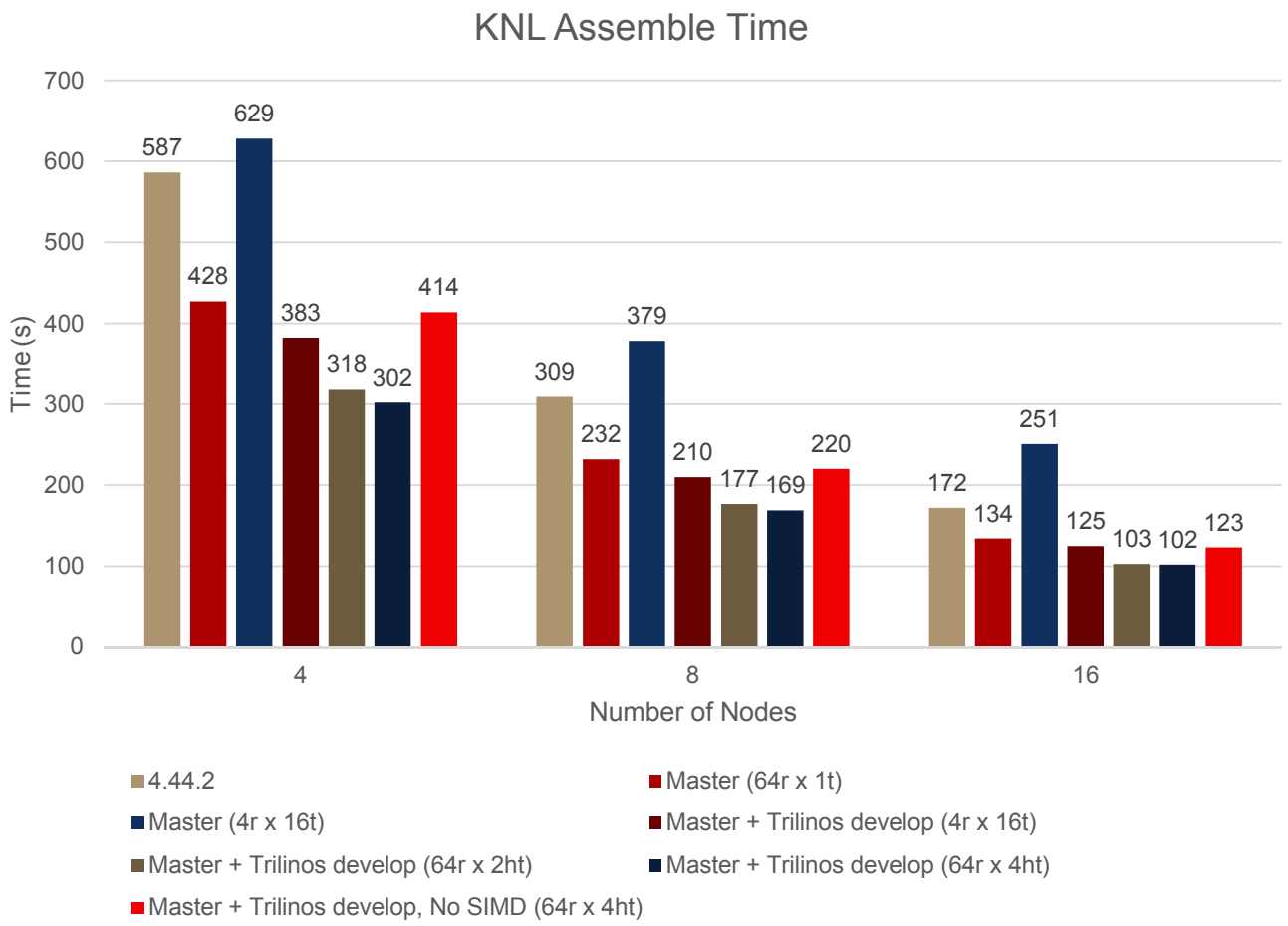
8M Element Conduction



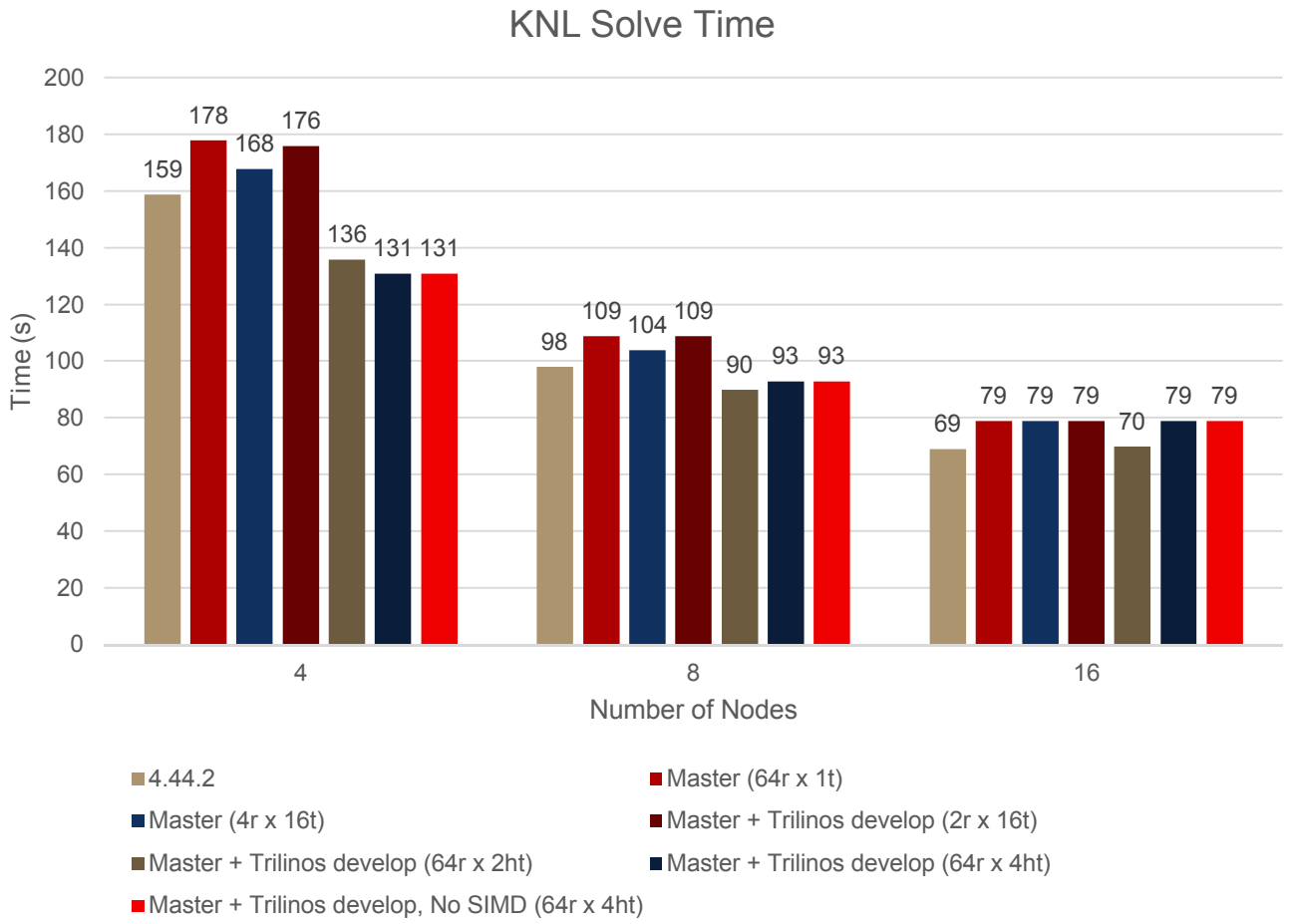
8M Element Conduction



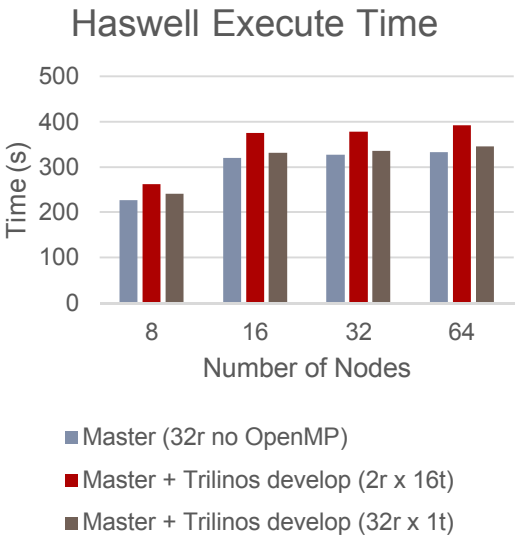
8M Element Conduction

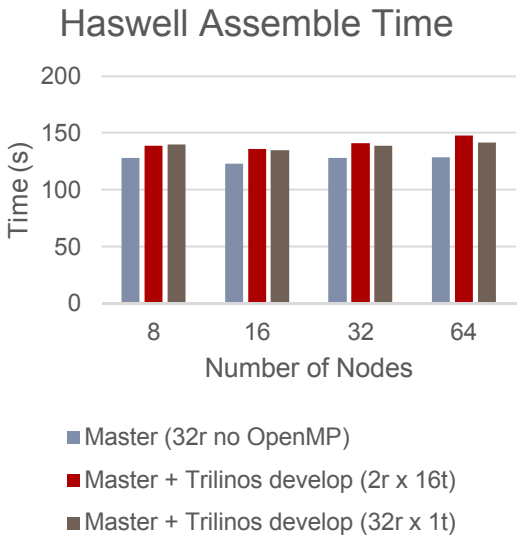


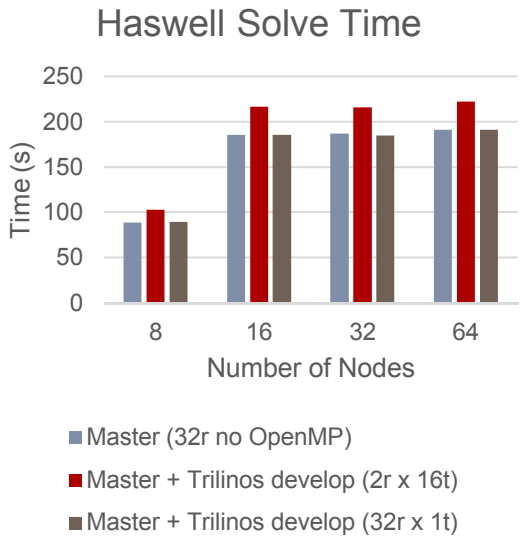
8M Element Conduction

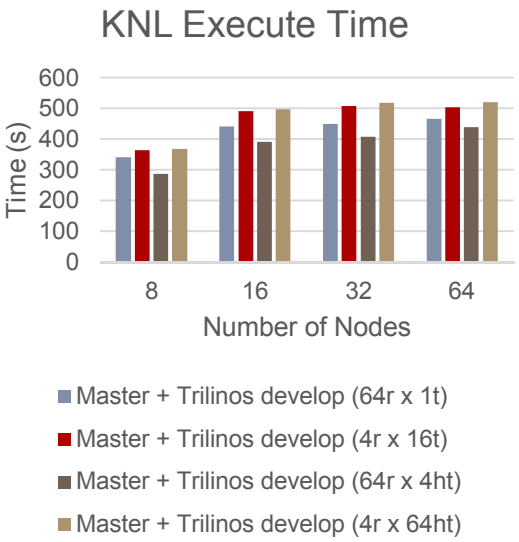


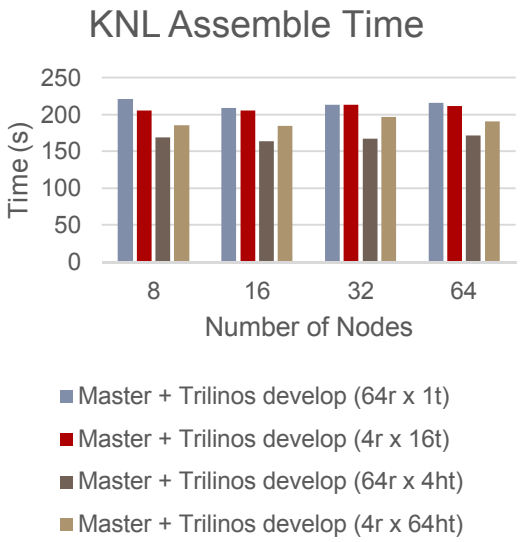
Weak Scaling

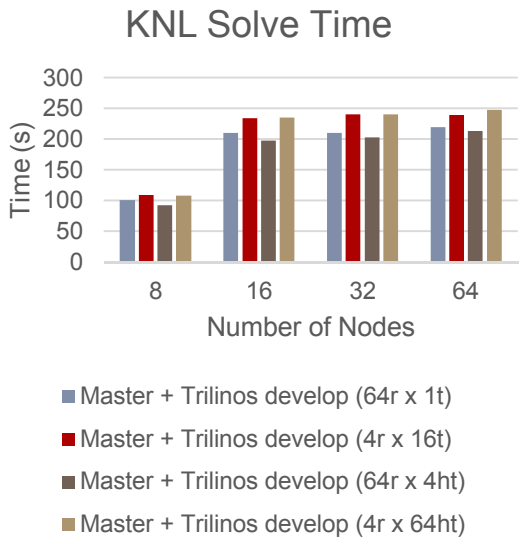












Efficiency

Sandia National Laboratories

$$\text{eff} = \frac{t_{mpi}}{t_{threads}}$$