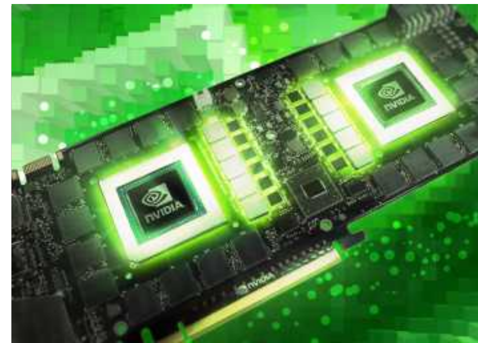
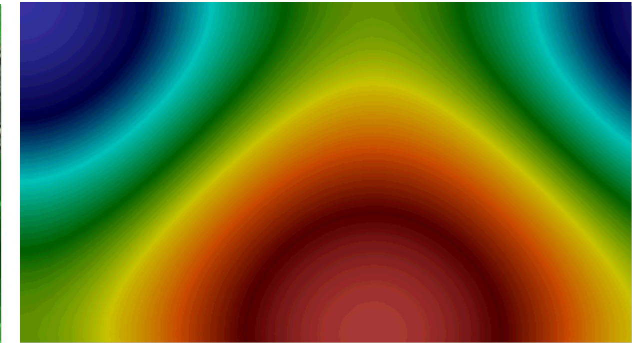
SAND2017-7241C





Prototyping the Next Generation of Aria

Jonathan Clausen, Victor Brunini, Chris Forster , David Noble ,
Christian Trott , Si Hammond , Mark Hoemenn , Paul Lin

14th US National Congress in Computational Mechanics
July 17th-20th Montreal, Canada

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.

Outline

- Motivation
- Aria and Ariamini
- Threading and Vectorization
- GPU capable expressions
- Performance data
- Transitioning Aria

Motivation

Sandia National Laboratories

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)
Power CPUs
Nvidia Volta GPUs
120-150 PetaFlops

Technologies

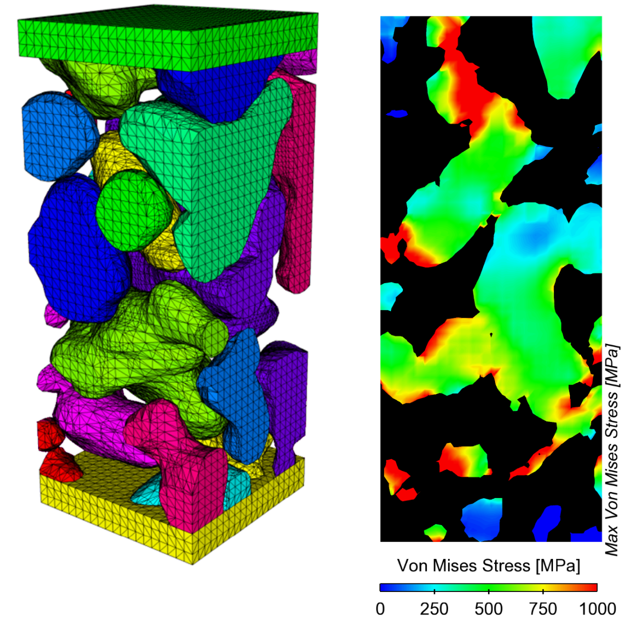
Sandia National Laboratories

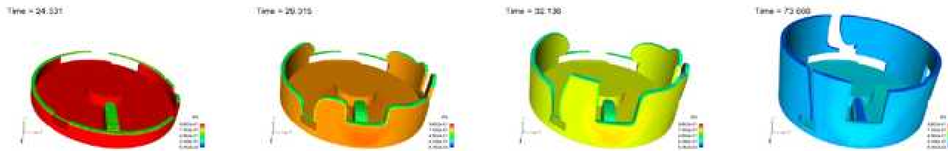
- Threading
- Vectorization
- GPUs
- Kokkos
- stk::simd

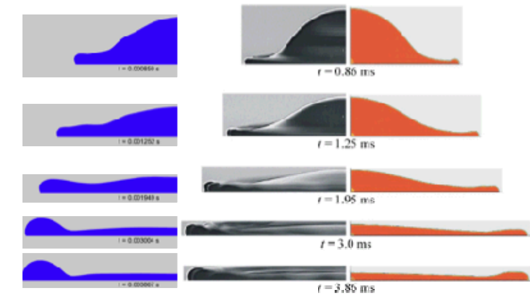
Aria

Sandia National Laboratories

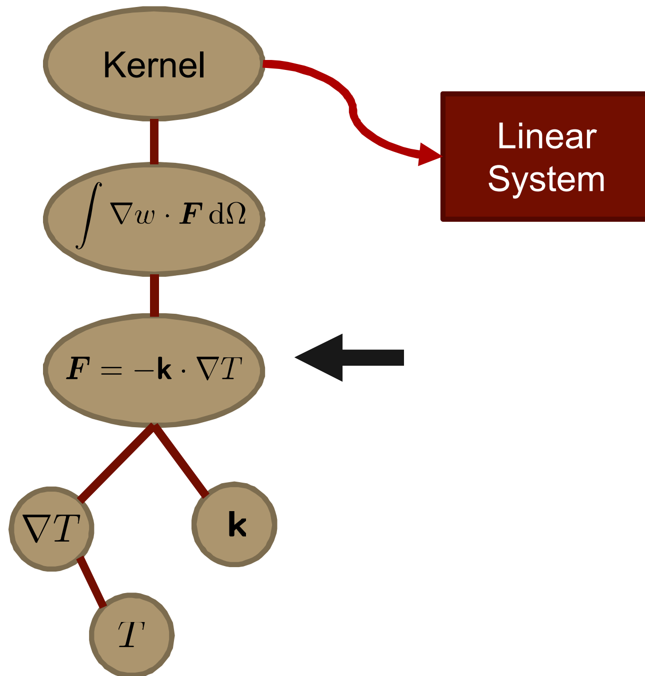
- nonlinear finite element
- full Newton sensitivities
- solvers (Trilinos)
- MPI
- monolithic or segregated coupling
- many equations (energy, mass, momentum, species transport, generalized chemistry, voltage)
- expression based







What is ariamini?



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

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

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

What is an expression?

Sandia National Laboratories

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

```
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

Sandia National Laboratories

```
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) {
//...
```

Threading

```
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

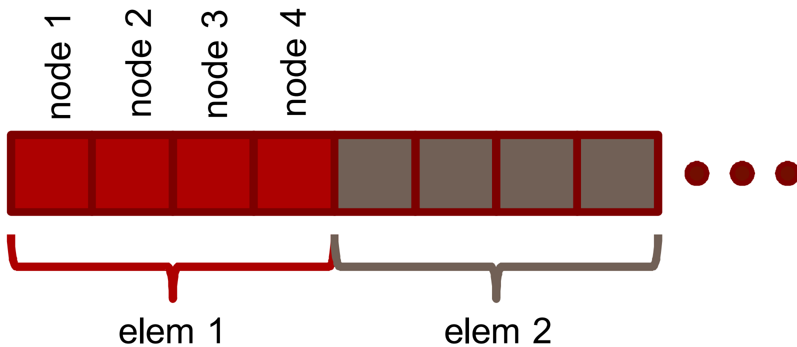
```
void Scalar_Vector_Product_Expression::compute_values() {
  for(int elem = 0; elem < nelem; ++elem) {
    for(int ip = 0; ip < nip; ++ip) {
      for(int dim = 0; dim < ndim; ++dim) {
        values(elem, ip, dim) =
          scalar(elem, ip) * vector(elem, ip, dim);
      }
    }
  }
}
//------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

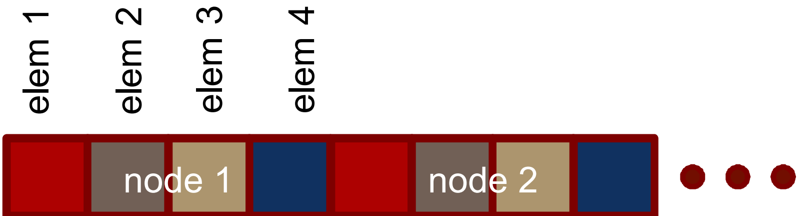
Threading

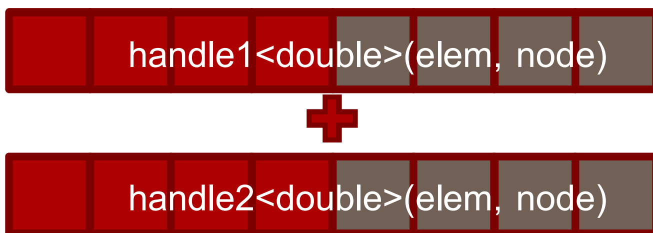
```
void Scalar_Vector_Product_Expression::compute_values(ElemRange elem_range) {
  for(auto elem : elem_range) {
    for(auto ip : my_sizes.get_points()) {
      for(auto dim : my_sizes.get_dims()) {
        values(elem, ip, dim) =
          scalar(elem, ip) * vector(elem, ip, dim);
      }
    }
 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

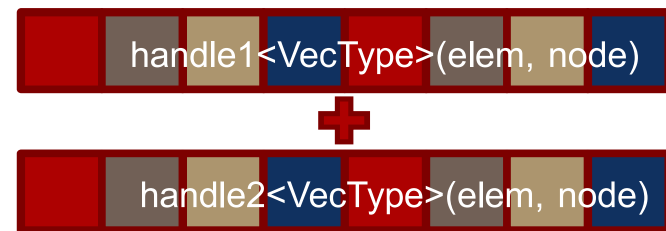
Vectorization

Expression_Handle values(elem, node)









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

Sandia National Laboratories

```
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

Sandia National Laboratories

- Vectorization
  - 95% of expressions remain unchanged
  - Gather from stk fields
  - Scatter into linear system
- Threading
  - ElementRange passed into compute functions
  - Expression_Handles now wrap Kokkos::Views

Other Optimizations

Sandia National Laboratories

- Rewritten master element
  - simplification of expressions
  - elimination of Fortran interface
- Compile time sizes throughout all expressions
  - spatial dimension
  - number nodes
  - number integration points

Code Changes for GPU

Sandia National Laboratories

Expression

```
void compute_values()
void compute_sensitivities()
…
//many legacy virtual functions
…
Expression_Handle gradT
Expression_Handle k
Expression_Handle values
```

Expression_Handle

```
Kokkos::View<…> values
Kokkos::View<...> sens
...
//many legacy virtual functions
```

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

Code Changes for GPU

Expression

DeviceExpr *
create_device_expression()
…
//many legacy virtual functions
…
Expression_Handle gradT
…

DeviceExpression

compute_values()
compute_sensitivities()

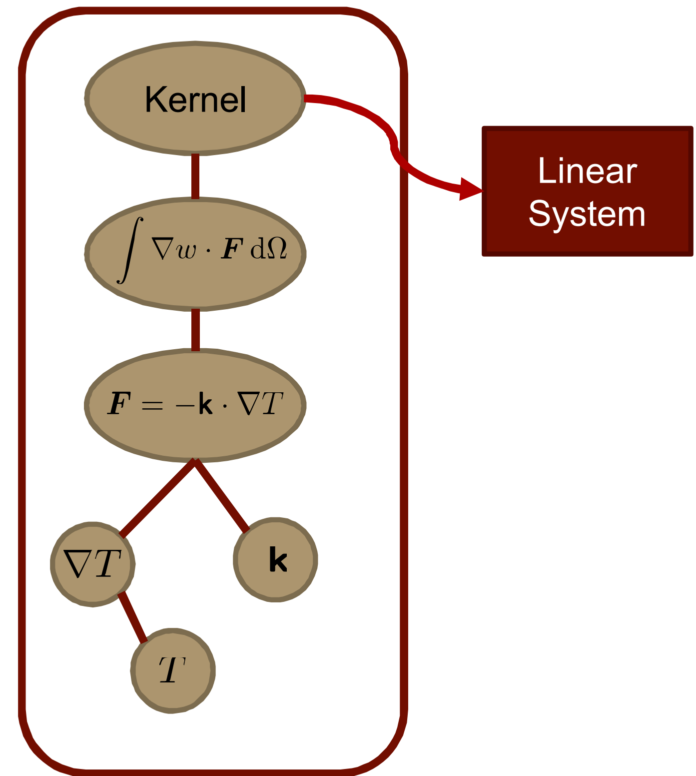
DeviceHandle gradT

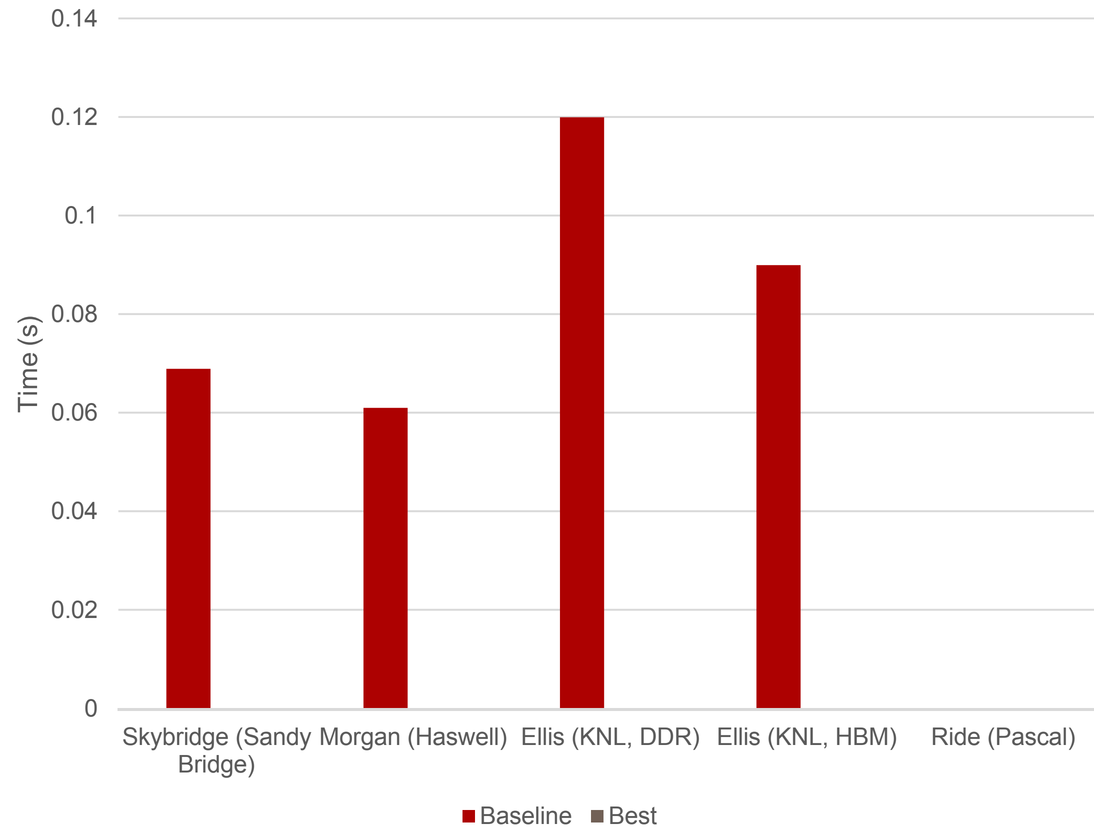
```
ExprType * p
  = static_cast<ExprType *>
    (Kokkos::kokkos_malloc<>(sizeof(ExprType)));
Kokkos::parallel_for(
    Kokkos::RangePolicy<>(0, 1),[=] DEVICE(const int i)
{
  new (p) ExprType(rhs);
});
Kokkos::fence();
```

constructor for DeviceExpression sets up DeviceHandles
DeviceHandles and DeviceExpression UVM accessible

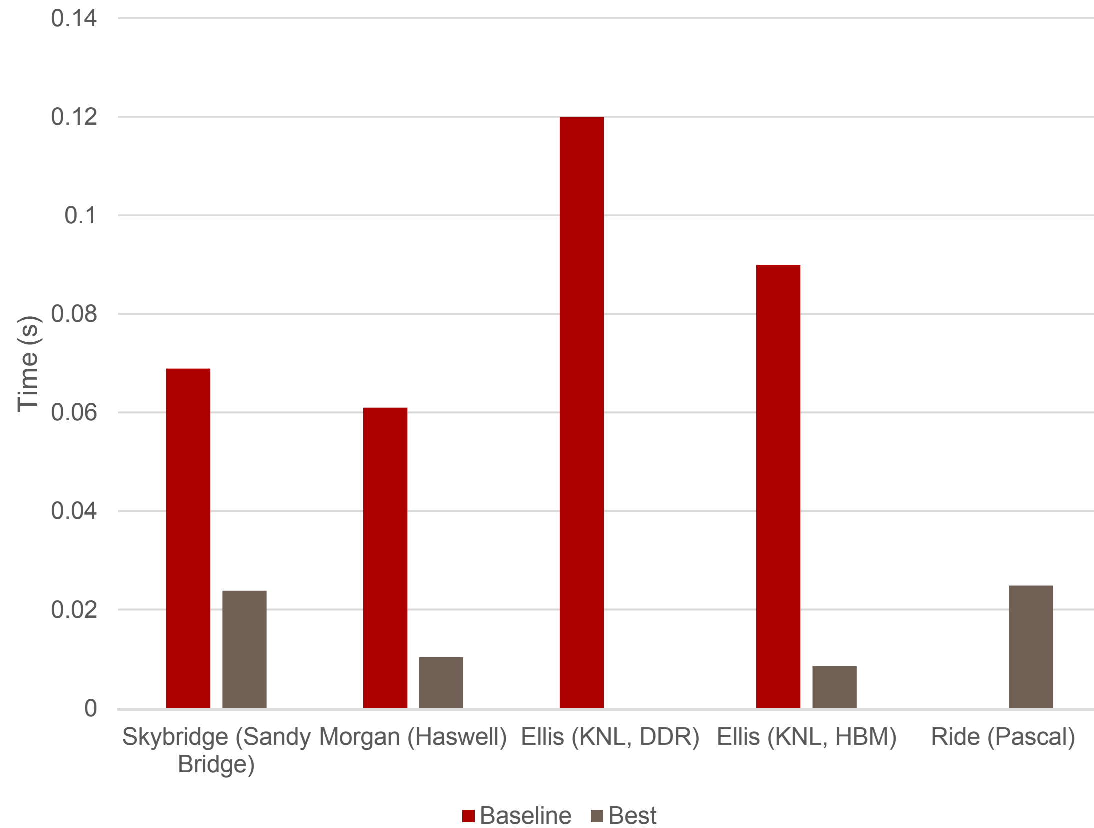
Performance Data

- **60^3 heat conduction**
- **Single-node performance**

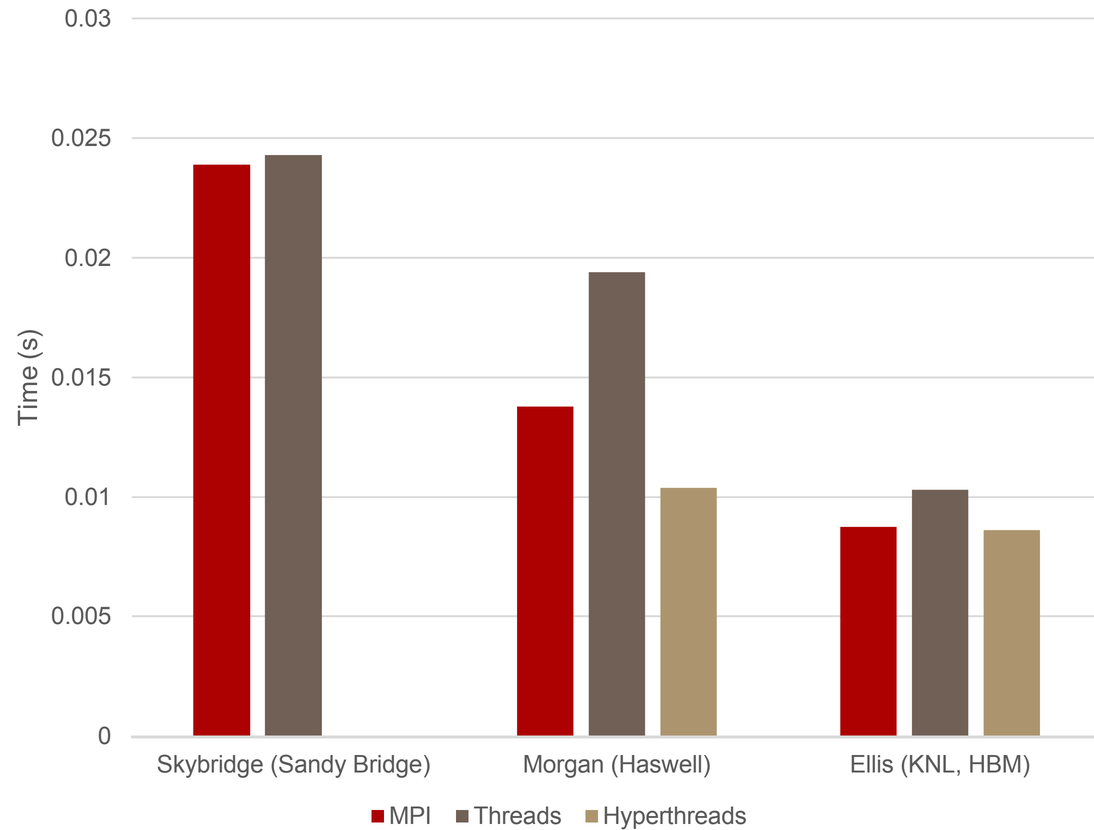




MPI (no threading)

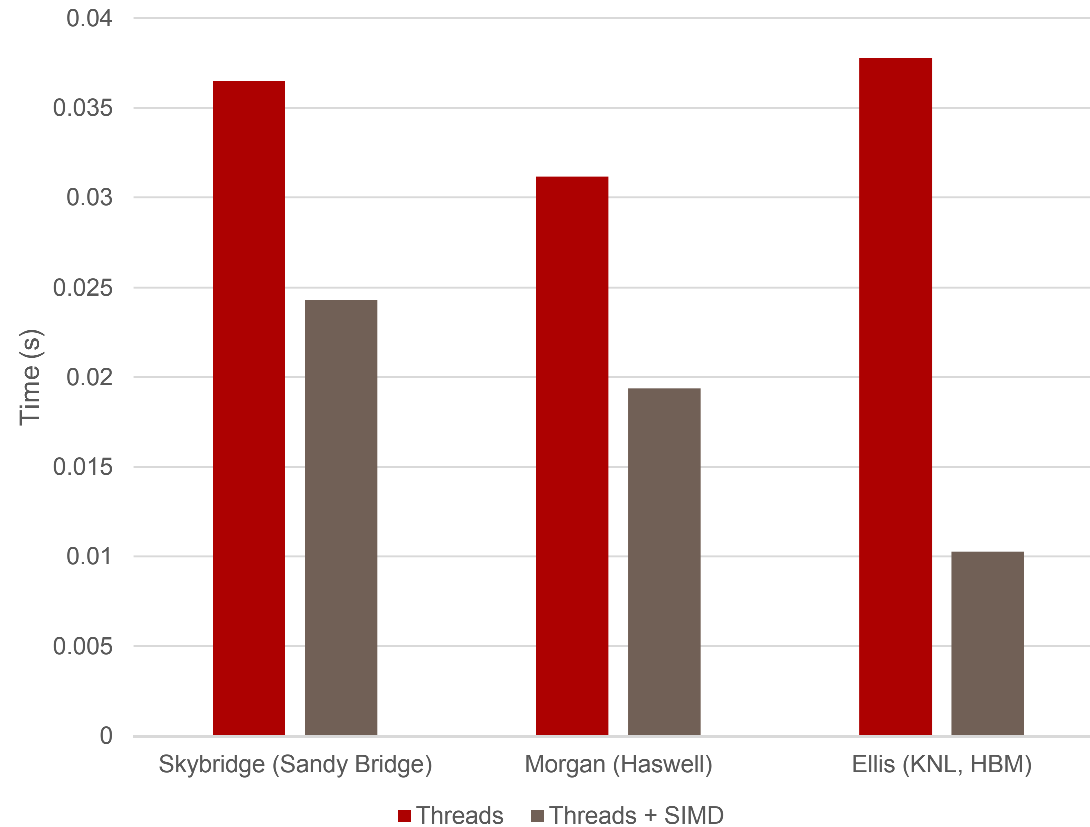


MPI (no threading)



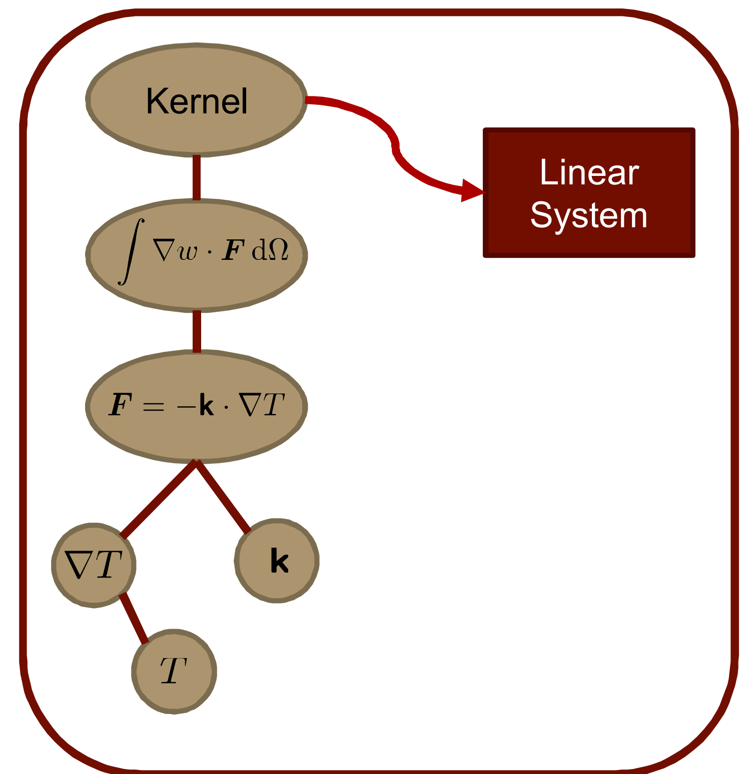
Threads:
- 2R x 8t (Sandy)
- 2R x 16t (Haswell)
- 16R x 4t (KNL)

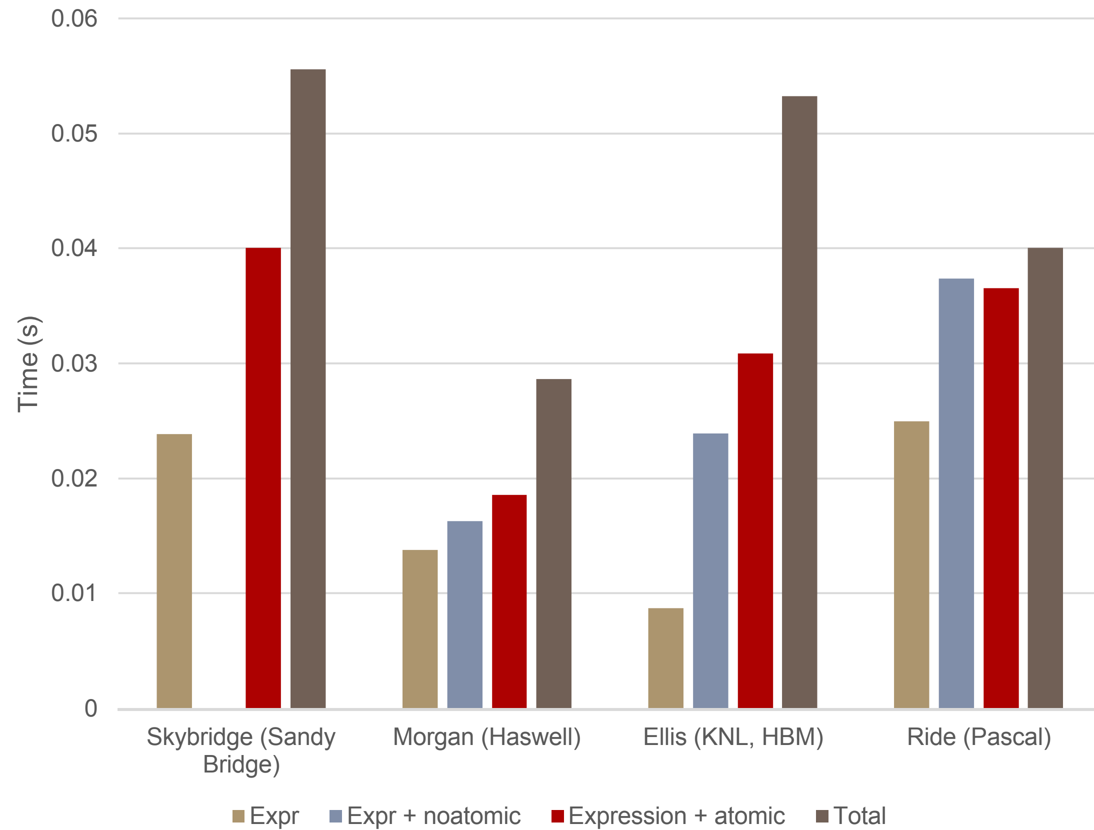
Hyperthreads:
- 16R x 2t (Haswell)
- 64R x 2t (KNL)

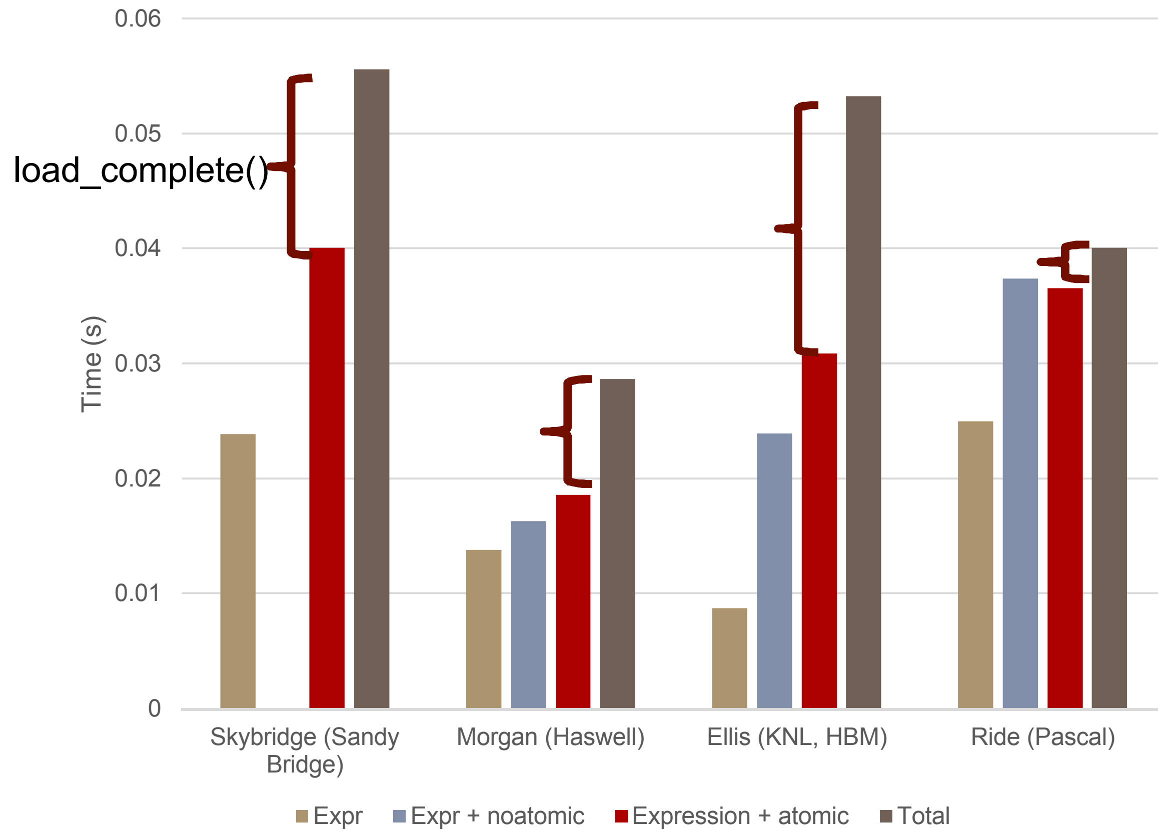

Influence of Vectorization
60³ Heat Conduction
0.04
0.035
0.03
0.025
0.02
0.015
0.01
0.005
0
Time (s)
Skybridge (Sandy Bridge)
Morgan (Haswell)
Ellis (KNL, HBM)
Threads
Threads + SIMD

Performance Data

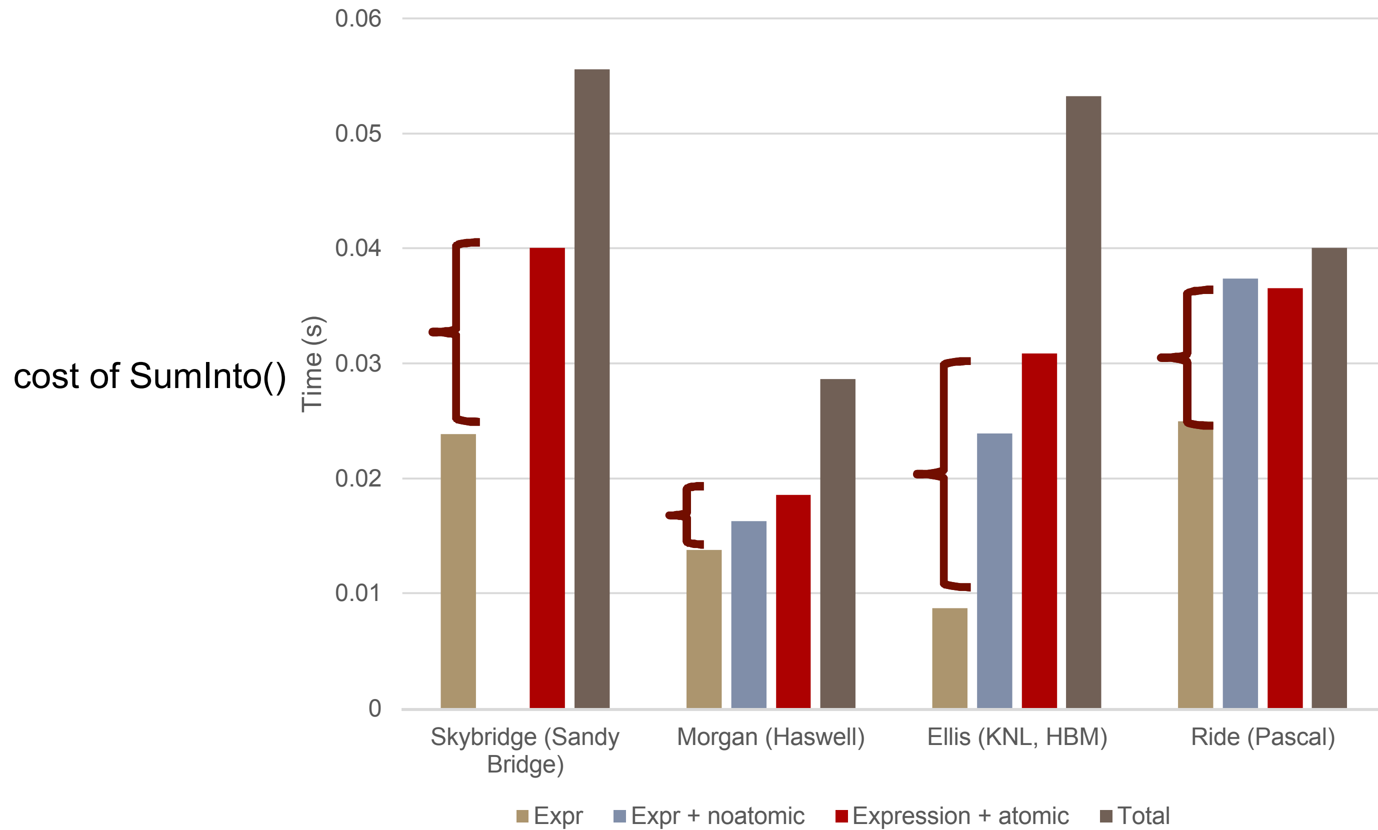
- **60^3 Heat Conduction with scatter into linear system**
- **Using stk::simd**
- **MPI everywhere**

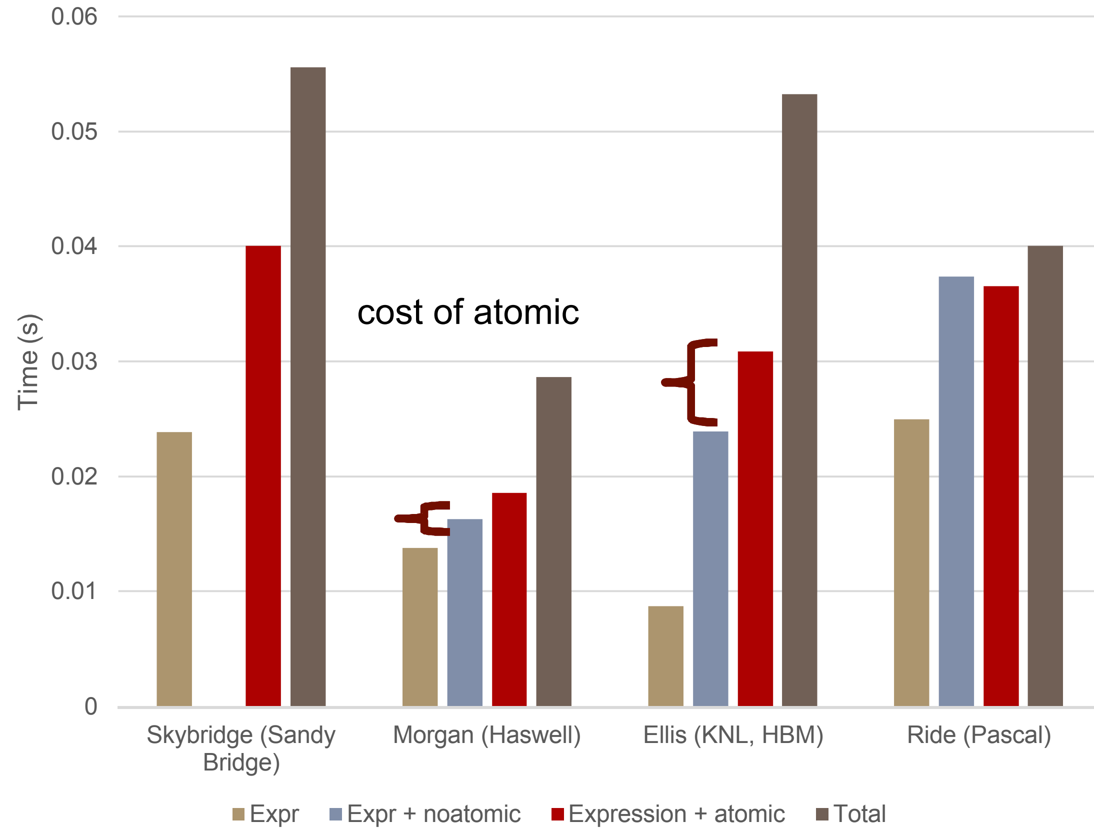



Assembly into Linear System
60³ Heat Conduction
Time (s)
0.06
0.05
0.04
0.03
0.02
0.01
0
Skybridge (Sandy Bridge)
Morgan (Haswell)
Ellis (KNL, HBM)
Ride (Pascal)
Expr
Expr + noatomic
Expression + atomic
Total

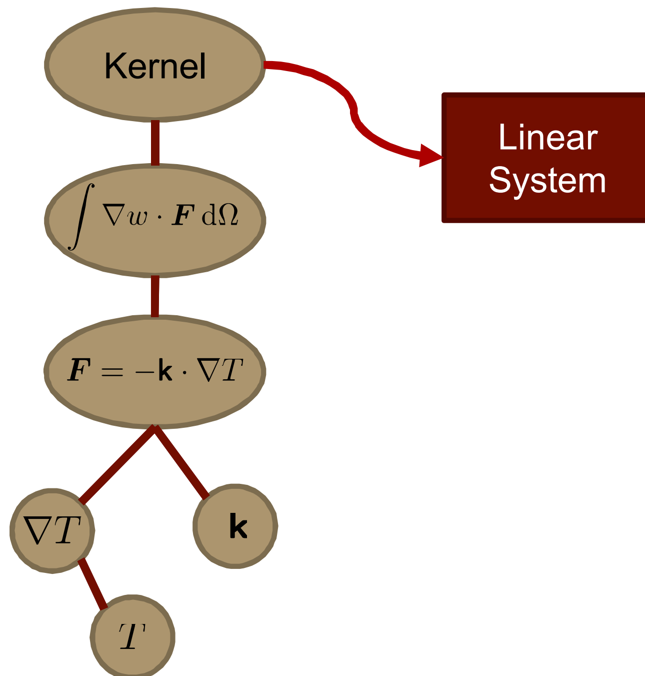

Assembly into Linear System
60³ Heat Conduction
0.06
0.05
0.04
0.03
0.02
0.01
0
Time (s)
load_complete()
Skybridge (Sandy Bridge)
Morgan (Haswell)
Ellis (KNL, HBM)
Ride (Pascal)
Expr
Expr + noatomic
Expression + atomic
Total

Sandia National Laboratories

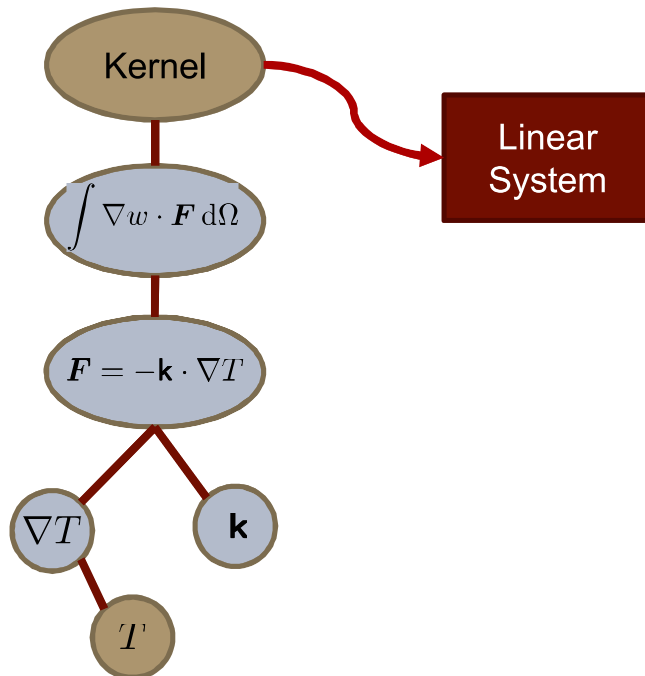



Assembly into Linear System
60³ Heat Conduction
0.06
0.05
0.04
0.03
0.02
0.01
0
Time (s)
cost of atomic
Skybridge (Sandy Bridge)
Morgan (Haswell)
Ellis (KNL, HBM)
Ride (Pascal)
Expr
Expr + noatomic
Expression + atomic
Total

Expression Fusion

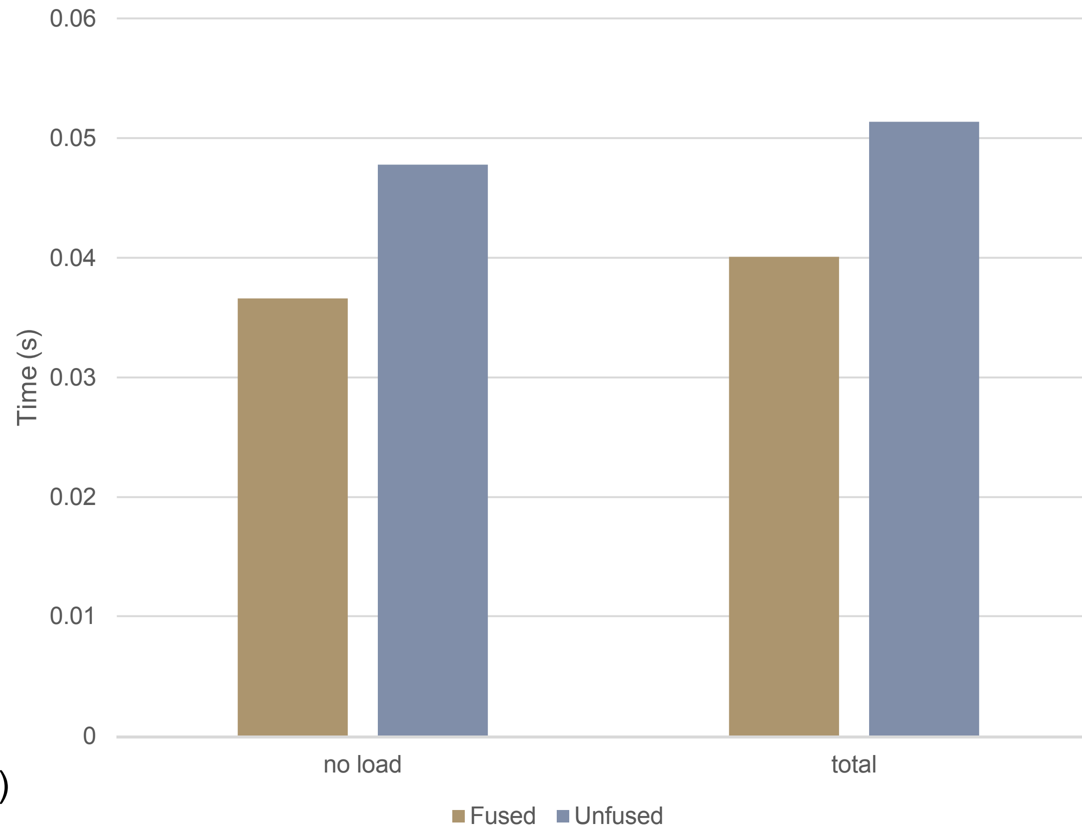


Expression Fusion

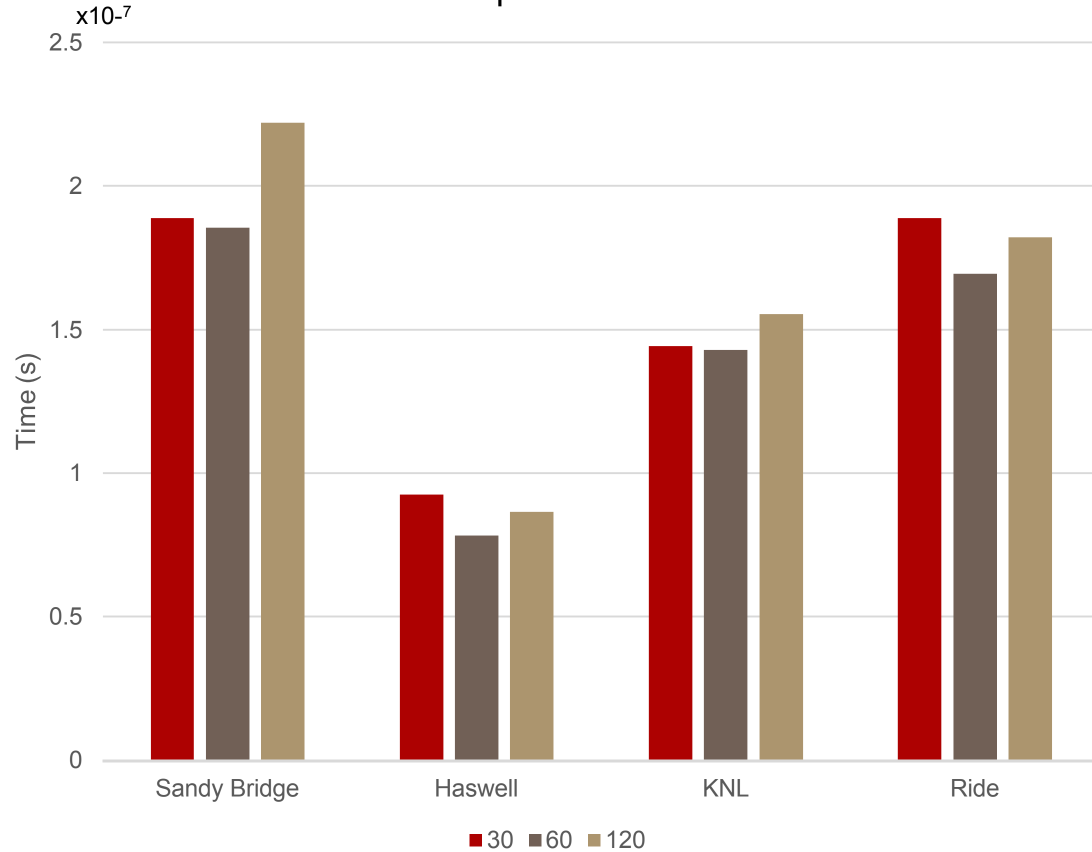


combine expressions into one
loss of generality
memory bandwidth
CPU cache vs GPU

Sandia National Laboratories



Results on Ride (Pascal)
1 MPI Rank



Sandy Bridge (MPI)
Haswell (Hyperthreads)
KNL (MPI)

Migration in Aria

Sandia National Laboratories

- O(1000) expressions to change
- Legacy expressions
- Must be incremental
- Interoperability of old and new expression handles
- Technical debt
- Defer GPU capable expressions later

Conclusions

Sandia National Laboratories

- Pleased with performance optimizations
  - vectorization
  - changes unrelated to threading
- Threading may be unnecessary for Trinity I/II
  - need more data at scale
  - nice to have additional knob
- GPU requires extensive changes
  - nested expressions
  - rewrite master element library

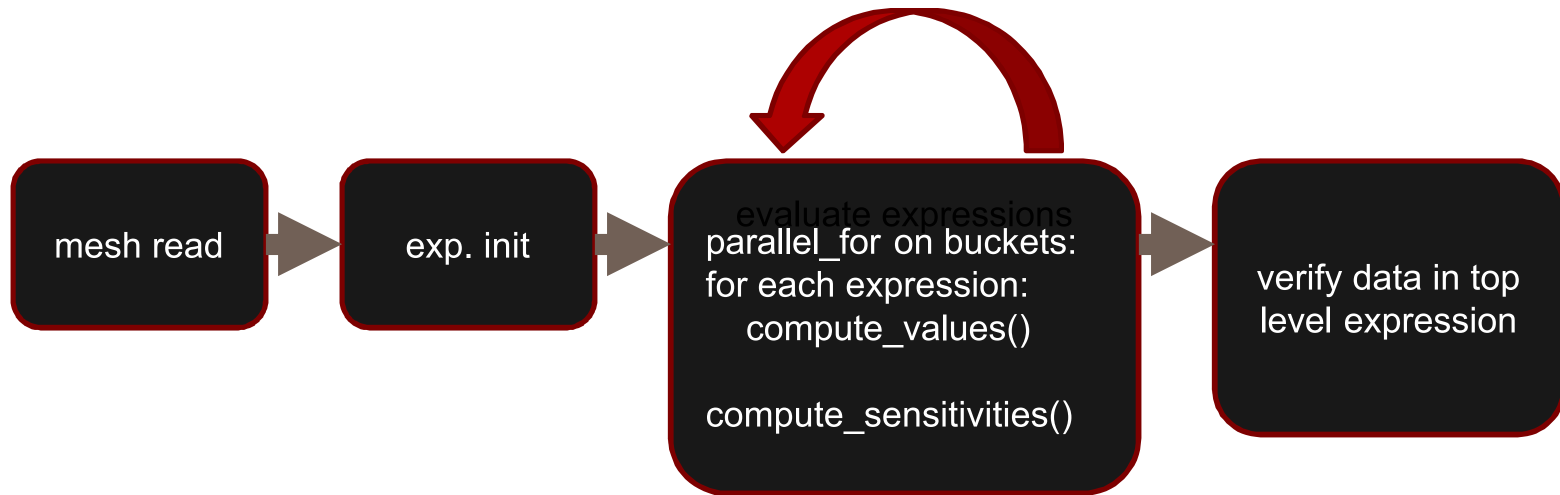
Sandia National Laboratories

Questions?

Sandia National Laboratories

Backup Slides

What is ariamini?



What is an expression?

Sandia National Laboratories

```
template <typename SIZE_TRAITS>
void Scalar_Diffusion_Kernel_Expression::fast_compute_values(const ElementRange elem_range)
{
  const SIZE_TRAITS sz(my_tensor_dimension, num_points, num_dof_coeffs);
  const Int elem_begin = elem_range.begin();
 const Int nPoint = sz.nPoint;
//---nDim, nCoeff, nPoint---
  for (Int elem = elem_begin; elem < elem_end; ++elem) {
    for (auto && flux_contrib : flux_vec) {
      for (Int gp = 0; gp < nPoint; ++gp) {
        const auto c = my_multiplier * detj(elem, gp) * intg_weight(elem, gp) * h3_scale(elem, gp);
        for (Int in = 0; in < nCoeff; ++in)
        {
          typename Scalar_Expression_Handle::value_type dot_product = 0.0;
          for (Int dim = 0; dim < nDim; ++dim)
          {
            dot_product += gradwfcn(elem, gp, in, dim) * flux_contrib(elem, gp, dim);
          }
          values(elem, in) += c * dot_product;
        }
      }
    }
  }
}
```

What is an expression?

```
template <typename SIZE_TRAITS>
void Scalar_Diffusion_Kernel_Expression::fast_compute_sensitivities(const ElementRange elem_range)
{
  const SIZE_TRAITS sz(my_tensor_dimension, num_points, num_dof_coeffs);
  const Int elem_begin = elem_range.begin();
 const Int nPoint = sz.nPoint;
  //----
  for (auto && flux : flux_vec)
  {
  for (auto && sens : flux.get_kokkos_sens_handles())
    {
      const Int num_sens_dofs = sens.num_dofs;
      auto & dself_vals = values.get_sens_handle(sens.sens_id());

      SensContributionCaller<DiffusionFluxVecSensContribution> caller;
      caller.apply_sens_contributions(sz, num_sens_dofs, elem_range, my_multiplier,
          detj, intg_weight, h3_scale, gradwfcn, sens, dself_vals);
    }
  }
}
```

What is an expression?

```
class DiffusionFluxVecSensContribution
{
public:
  template <typename SENS_TRAITS, typename SIZE_TRAITS>
  void operator()(const SIZE_TRAITS sz, //...
//-----
 const auto sens_view = sens.get_sens_view(sz, sens_sz);
  auto dself_view = dself.get_sens_view(sz, sens_sz);
//-----
 for (Int elem = elem_begin; elem < elem_end; ++elem) {
      for (Int gp = 0; gp < nPoint; ++gp) {
        const auto c = my_multiplier * detj_view(elem, gp) * intg_weight_view(elem, gp) *
            h3_scale_view(elem, gp);
        for (Int in = 0; in < nCoeff; ++in) {
          for (Int dim = 0; dim < nDim; ++dim) {
            const auto gradw = gradwfcn_view(elem, gp, in, dim);
            for (Int dof = 0; dof < nDof; ++dof) {
              dself_view(elem, in, dof) += gradw * sens_view(elem, gp, dim, dof) * c;
            }
          }
        }
      }
   }
 }
//---
```

What is an expression?

```
//-----
 const auto sens_view = sens.get_sens_view(sz, sens_sz);
  auto dself_view = dself.get_sens_view(sz, sens_sz);
//-----
```

```
template <typename SIZE_TRAITS, typename SENS_TRAITS>
decltype(auto) get_sens_view(const SIZE_TRAITS & sz, const SENS_TRAITS & sens_sz) const
//    -> decltype(Traits::get_compile_time_sens_view(this->my_view, sz, sens_sz))
{
  return Traits::get_compile_time_sens_view(this->my_view, sz, sens_sz);
}

template <typename SIZE_TRAITS, typename SENS_TRAITS>
static decltype(auto) get_compile_time_sens_view(const SensViewType & rt_view, const SIZE_TRAITS & sz, const
SENS_TRAITS & sens_sz)
    //-> decltype(TempView<DoubleType * [SIZE_TRAITS::nPoint_val][SENS_TRAITS::nDof_val]>(rt_view))
{
  return TempView<DoubleType * [SIZE_TRAITS::nPoint_val][SENS_TRAITS::nDof_val]>(rt_view);
}
```

Assembly

```
void do_assembly()
{
  //...
  stk::mesh::BucketVector const & buckets = mesh().get_buckets(iterate_rank, selector);
  auto team_policy = get_team_policy(buckets.size());
  //...
  Kokkos::parallel_for(kernel_name, team_policy, [&](const DeviceTeam & team) {
  // body (next slide)...
});
}
```

Assembly

Sandia National Laboratories

```
const int ib = team.league_rank();

int my_workset_index = Kokkos::DefaultExecutionSpace::hardware_thread_id();
auto & b = *buckets[ib];
const auto bucket_length = b.size();
for (stk::mesh::Bucket::size_type offset = 0; offset < bucket_length; offset += workset_size)
{
  //need simd padded elems
  const UInt elems_this_pass = std::min(workset_size, bucket_length - offset);
  const UInt remainder = elems_this_pass % DoubleTypeLength;
  const UInt simd_chunks = remainder ? elems_this_pass / DoubleTypeLength + 1 : elems_this_pass / 
DoubleTypeLength;
  const Int start = (my_workset_index * workset_size) / DoubleTypeLength;

  ElementRange elem_range(start, start + simd_chunks, elems_this_pass, my_workset_index);
  mock_manager().set_mesh_object_couriers(...);

  mock_manager().evaluate_expressions(elem_range);
  // Apply lhs + rhs
}
```

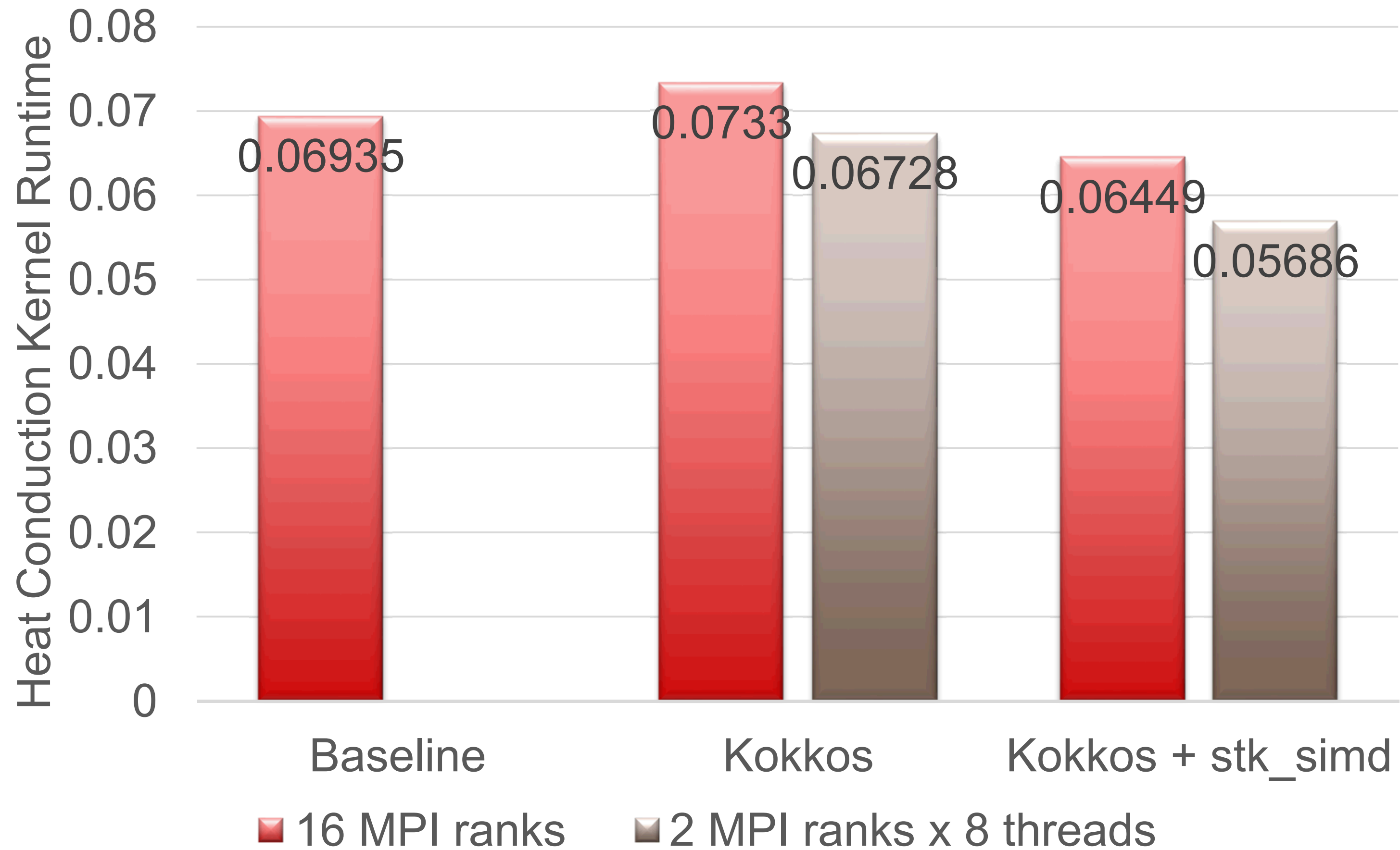
Gather

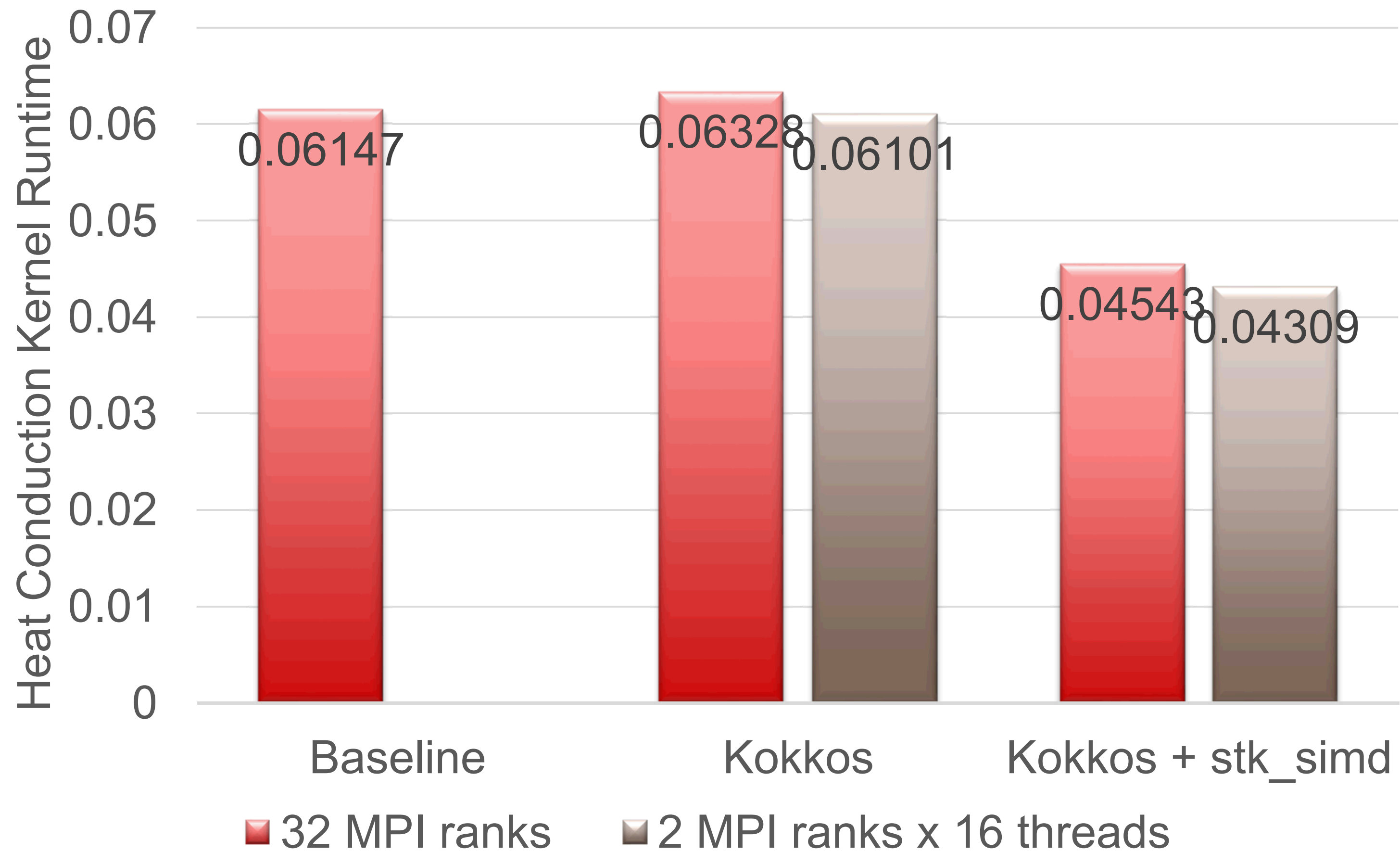
```
for (Int elem = elem_begin; elem < elem_end; ++elem) {
  for (Int id = 0; id < DoubleTypeLength; ++id) {

    Int elemIndex = elem * DoubleTypeLength + id;
    const tftk::mesh::FastMeshIndex * entity_node_indices =
        mesh_indices.node_indices(mesh.local_id(bucket[elemIndex]));

    for (UInt nd = 0; nd < field_nodes_per_entity; ++nd) {
      typename Functor::ObjDataType node_data =
          tftk::mesh::field_data<Real>(field, entity_node_indices[nd], field_length);

      stk::simd::set_data(node_data, values, id);
    }
  }
}
```


Skybridge (Sandy Bridge)
Ariamini Version Comparison
Heat Conduction Kernel Runtime
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0.06935
0.0733
0.06728
0.06449
0.05686
Baseline
Kokkos
Kokkos + stk_simd
16 MPI ranks
2 MPI ranks x 8 threads


Morgan (Haswell)
Ariamini Version Comparison
Heat Conduction Kernel Runtime
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
0.06147
0.06328
0.06101
0.04543
0.04309
Baseline
Kokkos
Kokkos + stk_simd
32 MPI ranks
2 MPI ranks x 16 threads

Sandia National Laboratories

Workset Sizing