Template Task Graph

An emerging practical dataflow programming paradigm for scientific simulation

Robert J. Harrison, Mahdi Javanmard, Poornima Nookala

Institute for Advanced Computational Science
Stony Brook University

Edward Valeev

Virginia Tech

George Bosilca and Thomas Herault

Innovative Computing Laboratory
University of Tennessee, Knoxville

robert.harrison@stonybrook.edu

Outline

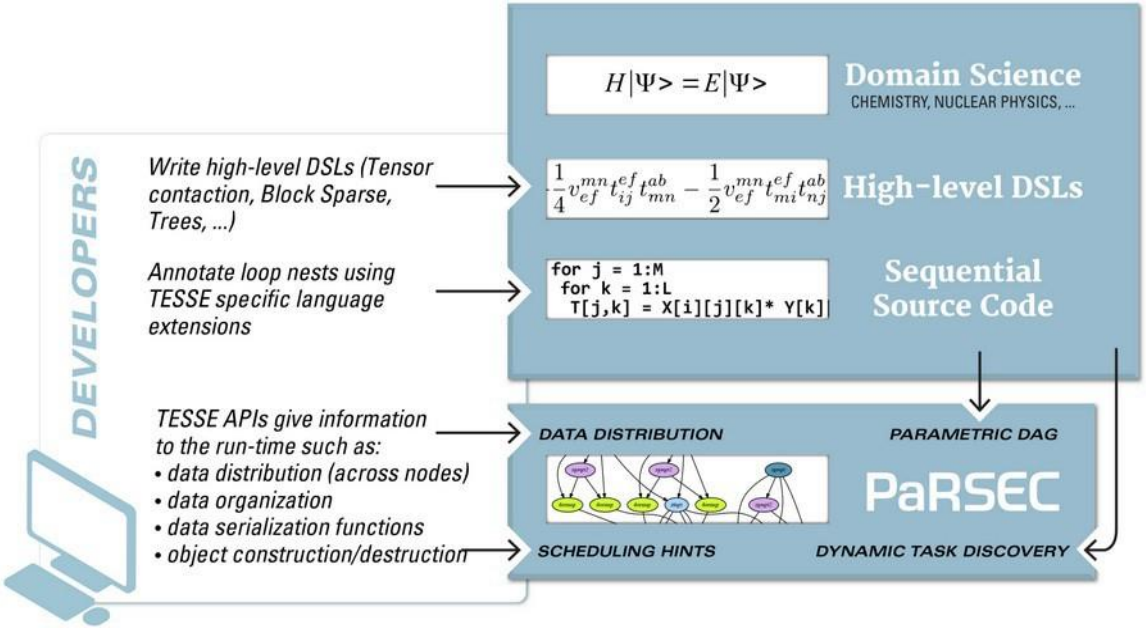
- What are TESSE and TTG?
- Motivation
- Exploring a mini-app
- Future work and opportunities for collaboration
- Ookami

Task-based environment for scientific simulation at extreme scale (TESSE)

<u>Stony Brook University</u>
- Harrison

<u>University of Tennessee</u>
- Bosilca and Herault

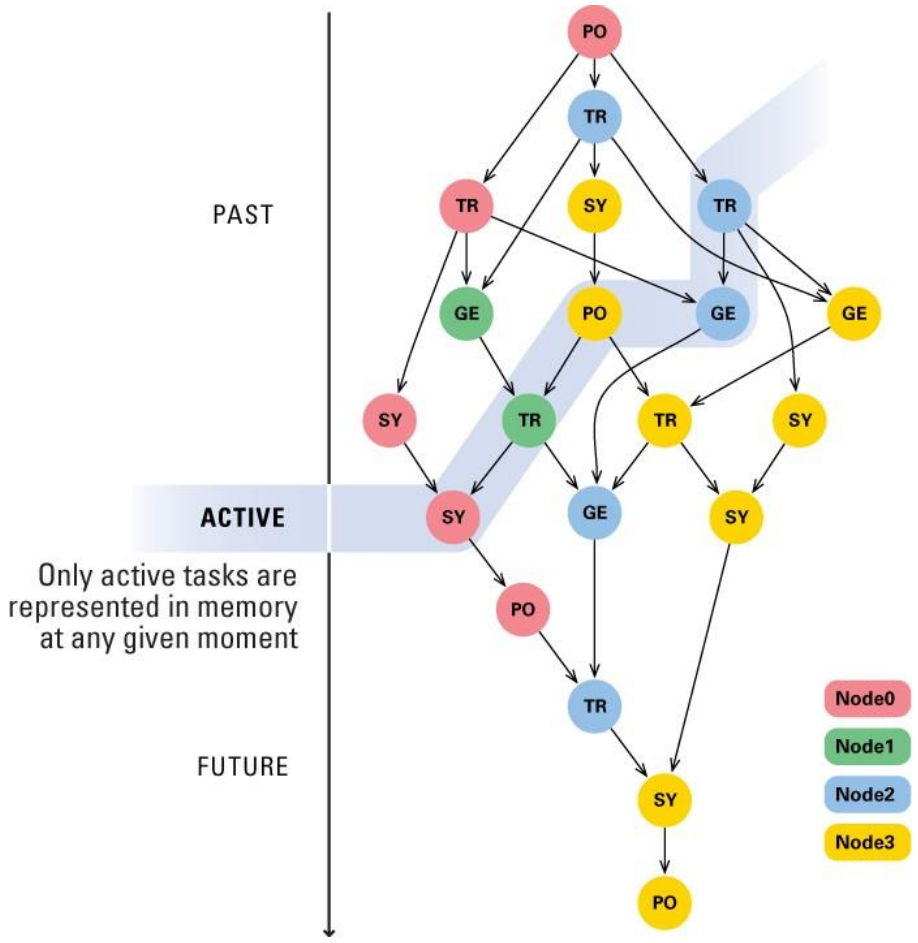
<u>Virginia Tech</u>
- Valeev



An extensible, robust and scalable directed acyclic graph (DAG) execution model supported by an intelligent and dynamic runtime that can adapt to changing requirements presented by the evolving numerical theories and HPC platforms.

- TTG (template task graph) is the main initial C++ API.
- Multiple runtimes supported
- Plans for multiple interoperable DSLs and algorithms on distributed data stru3ctures

TESSE extends PaRSEC to irregular apps



- Extends Parallel Scheduling and Execution Controller (PaRSEC) to larger classes of dynamic (data-dependent) computation; data distribution; compostion and execution of multiple DAGs
- Addresses
  - heterogeneous hardware by runtime selection between multiple implementations
  - heterogeneous data distribution by separate specification of data and algorithm, and runtime management of data motion
  - heterogeneous task duration through lightweight scheduling policies
- Automatic latency hiding enabled by knowledge of the dataflow of the program to enable all communications to occur in the background of the execution itself

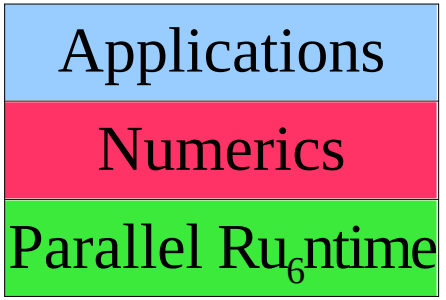
PaRSEC: http://icl.cs.utk.edu/parsec/

TESSE C++ API and dataflow model

- Driven by
  - MADNESS: Algorithms on unbalanced, deep spatial trees in 1-6 dimensions
  - TiledArray: Block-sparse and low-rank algorithms for tensors with 2-6 or more dimensions
  - Sparse linear algebra: element/block/rank sparsity
  - Dense linear algebra: original use of PaRSEC
- Related projects – CnC, Legion, Charm++, …

MADNESS

- A general purpose numerical environment for reliable and fast scientific simulation
  - Chemistry, nuclear physics, atomic physics, material science, nanoscience, climate, fusion, ...
- Want robust and fast algorithms that scale correctly with system size and are easy to write
- Semantic gap
  - Why are equations O(100) lines but codes O(1M) ?
- Facile path from laptop to exaflop



https://github.com/m-a-d-n-e-s-s/madness

Let

$$\Omega = [-20, 20]^3$$

$$r = x \to \sqrt{x_0^2 + x_1^2 + x_2^2}$$

$$g = x \to \exp(-2 * r(x))$$

$$v = x \to -\frac{2}{r(x)}$$

In

$$\nu = \mathcal{F}\, v$$

$$\phi = \mathcal{F}\, g$$

$$\lambda = -1.0$$

for $i \in [0, 10]$

$$\phi = \phi * \|\phi\|^{-1}$$

$$V = \nu - \nabla^{-2}\left(4 * \pi * \phi^2\right)$$

$$\psi = -2 * \left(-2 * \lambda - \nabla^2\right)^{-1} (V * \phi)$$

$$\lambda = \lambda + \frac{\langle V * \phi | \psi - \phi \rangle}{\langle \psi | \psi \rangle}$$

$$\phi = \psi$$

print "iter", i, "norm", $\|\phi\|$, "eval", λ

end

End

Highest-level DSL

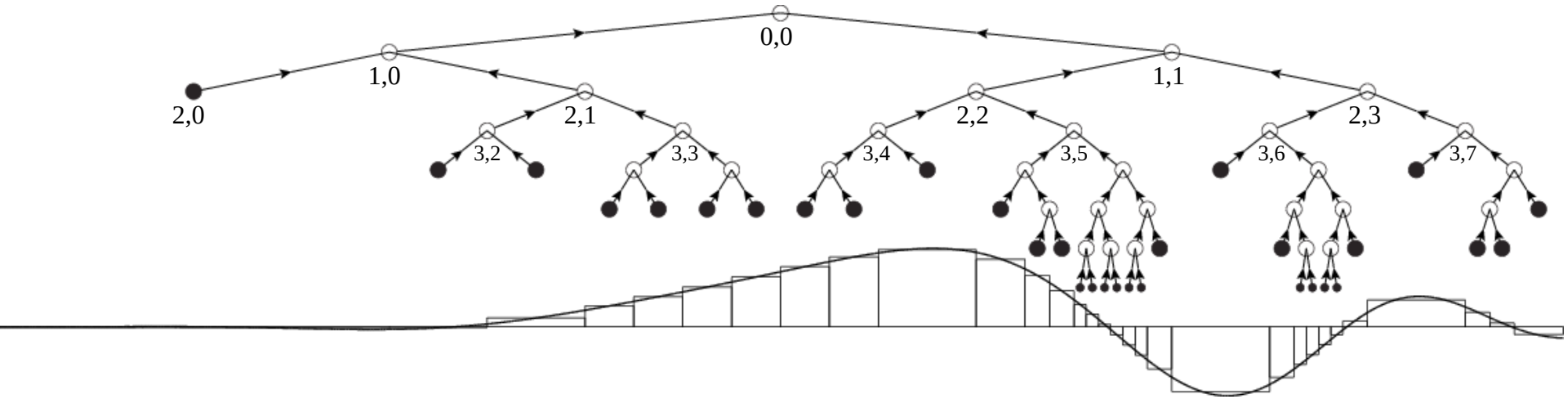
Compose directly in terms of functions and operators

This is a Latex rendering of a program to solve the Hartree-Fock equations for the helium atom

The compiler also outputs a C++ code that can be compiled without modification and run in parallel

Example adaptive tree in Haar basis

- Haar basis is a piecewise constant (like a histogram)
  - Not useful for calculation but easy to visualize and of fundamental interest
- Adaptive local refinement until local error measure is satisfied
  - Smaller boxes where rate of change is high and value not negligible
- Conventional adaptive mesh corresponds to boxes
- Construct tree connecting fine-scale to coarser-scale boxes
- Boxes labeled with level (n=0,1,...) and translation (l=0,1,...,2^n-1)
- Tree-based algorithms for fast computation (in math sense)



A Key Component

- Trade precision for speed – everywhere
  - Don’t do anything exactly
  - Perform everything to O(ε)
  - Require
    - Robustness
    - Speed, and
    - Guaranteed, arbitrary, finite precision
- Leads to very irregular and dynamic data structures and computation on unbalanced trees
  - Trees in 3D routinely go down 30+ levels

TILEDARRAY

- Generic massively parallel framework for dense and sparse tensor algebra
- State of the art application to electronic structure of chemistry and materials in Massively Parallel Quantum Chemistry (MPQC) package
  - Prototyping platform for DOE Exascale Chemistry App
  - Experimental use by research codes, e.g. ChronusQuantum (Xiaosong Li/UW)
- Reduces communication and load imbalance of sparse tensor algebra using data-driven MADNESS runtime
- Development supported in part by the NSF SSI program (OCI-1047696)

TESSE will allow:

- Vastly improved functionality by making it easier to compose complex algorithms
- Improved portability (e.g. accelerators)
- Improved resource management
- Leverage CS community expertise

TiledArray: http://github.com/ValeevGroup/tiledarray/

TILEDARRAY: HIGH-LEVEL DSL

Math

$$R_{iajb} = G_{iajb} + F_{ac}T_{icjb} + F_{bc}T_{iajc} - F_{ik}T_{kajb} - F_{jk}T_{iakb}$$
$$E = (G_{iajb} + R_{iajb})(2T_{iajb} - T_{ibja})$$

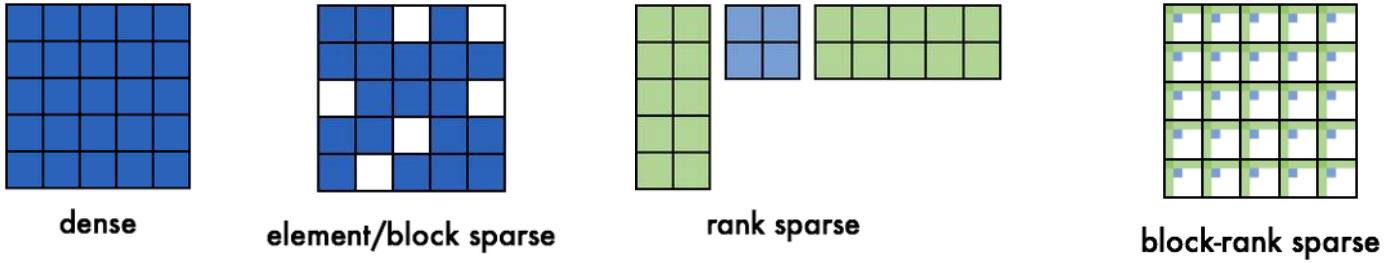
C++

```
TA::TArrayD R(world, ovov);

R("i,a,j,b") = G("i,a,j,b") + Fv("a,c") * T("i,c,j,b") +
               Fv("b,c") * T("i,a,j,c") - Fo("i,k") * T("k,a,j,b") -
               Fo("j,k") * T("i,a,k,b");

double energy =
     (G("i,a,j,b") + R("i,a,j,b")).dot(2 * T("i,a,j,b") - T("i,b,j,a"));
```

expression trees are parsed at compile time,
transformed (op conversion and fusion) and evaluated lazily at runtime

TILEDARRAY: FEATURES

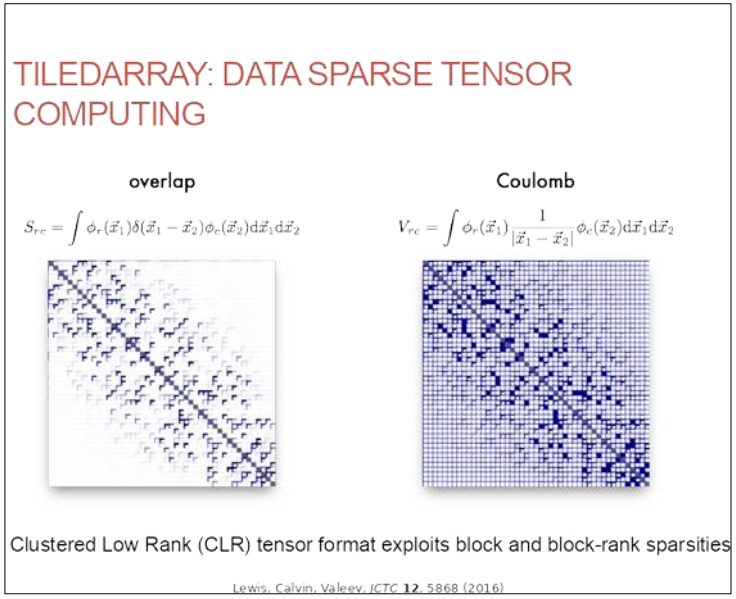
- Structure



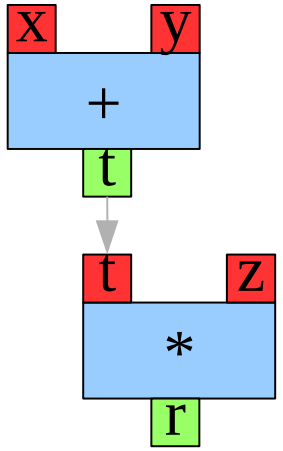
  - including slicing (`.block()`)

- Arithmetic
  - +, -, * (Hadamard), scale
  - Contract (tensor analog of GEMM)
  - Permutation (tensor analog of transposition)

- Algebra
  - Conjugate gradient
  - Direct Inversion of Iterative Subspace (DIIS)
  - In progress: decompositions (CP, Tucker)



Key concepts of TESSE dataflow model



- Template task graph (TTG)
  - Parameterize each task that will execute the operation by a `TaskId`, e.g.,
    - a loop index (i) making a separate task for each iteration
    - the label of each node in a tree being traversed
    - a pair of indices labelling a matrix sub-block
  - Avoids describing the entire task DAG which is huge
  - The user provides a map from `TaskId` to where the task will execute
- All data also labeled with `TaskId` to match with consuming task
  - All inputs (arguments) of a task have the same `TaskId`
  - Outputs may have different `TaskId` to send to different successors
- Through each output terminal a task can
  - send data to a specific successor (one key), or
  - broadcast to multiple successors (many keys), or
  - reduce to an accumulator

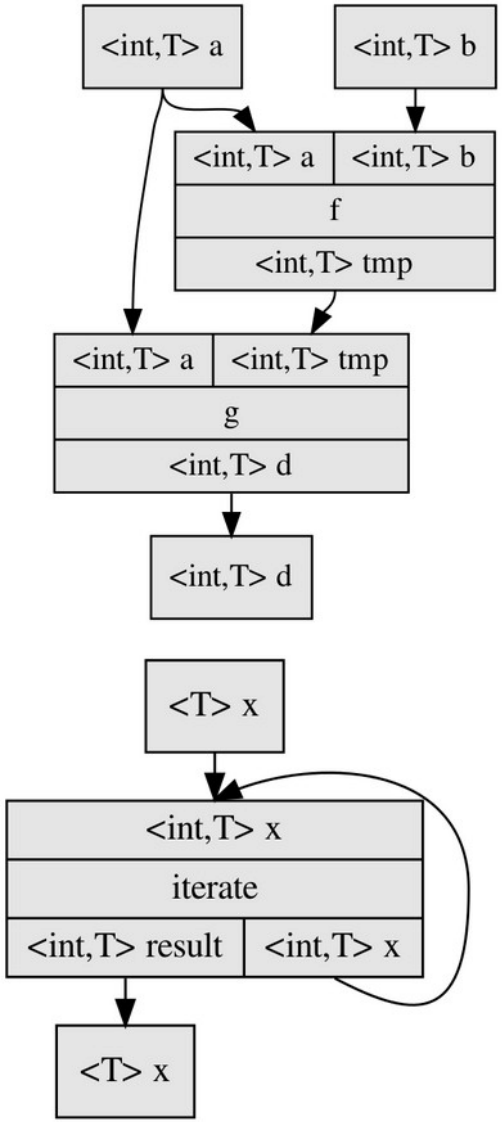
TTG Examples

- Data-parallel loop on vectors

```
for i : range {
    T tmp = f(a[i],b[i]);
    d[i] = g(a[i],tmp);
}
```

- Iterative algorithm

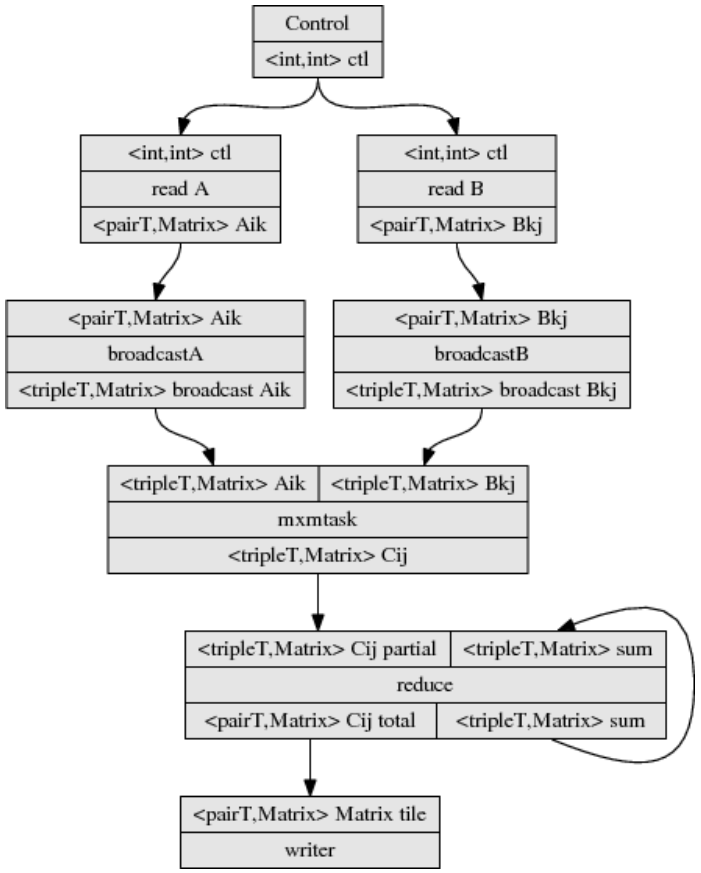
```
iter = 0;
while (tmp<10 && iter<100) {
    iter++;  x = f(x);
 }
```



Reading the TTG

- Input terminals at top of operation
- Name of operation in middle
- Output terminals at bottom of operation
- Connect output terminals to input terminals of successor tasks
- Computation/data flows from top to bottom
  - Pull terminal modifies this
- Recursion represented by backward edges and cycles
  - The TTG is not a DAG
  - The executed task graph is a DAG

Naïve SUMMA MxM in TTG



TemplateTasks and Tasks

- A `TemplateTask` describes a set of tasks parameterized by a `TaskId`
- The `Terminals` of a `TemplateTask` are placeholders for the inputs/outputs of a task

```
void f(TaskId, Arg0, Arg1, ..., OutputTerminals)
```

TemplateTasks and Tasks

- A `TemplateTask` describes a set of tasks parameterized by a `TaskId`
- The `Terminals` of a `TemplateTask` are placeholders for the inputs/outputs of a task

```
void f(TaskId, Arg0, Arg1, ..., OutputTerminals)
```

- The most general task can contain multiple send/broadcast operations that conditionally propagate results to successors
  - In general, this requires copying data and a split task implementation for GPUs

TemplateTasks and Tasks

- A `TemplateTask` describes a set of tasks parameterized by a `TaskId`
- The `Terminals` of a `TemplateTask` are placeholders for the inputs/outputs of a task

  ```
  void f(TaskId, Arg0, Arg1, ..., OutputTerminals)
  ```

- The most general task can contain multiple send/broadcast operations that conditionally propagate results to successors
  - In general, this requires copying data and a split task implementation for GPUs
- A "pure" task sends only at most once to each output terminal and defers this motion to the runtime by returning results with predicates

  ```
  std::tuple<R0,R1> f(TaskId, Arg0, Arg1, ...)
  ```

  where `R0=std::pair<bool,Result0>`, `Result0` is the actual type of result 0, and the bool determines if the data is to be sent
  - Exposes all info to the runtime, avoids copying data, and simplifies use of GPUs
  - Closest to PaRSEC PTG --- adds runtime computed predicates, general types

Terminals

- Correspond to inputs/outputs of a task
- Compose the TTG by connecting inputs and outputs
  - Either via providing an `Edge` to the constructor or by manually connecting

Terminals

- Correspond to inputs/outputs of a task
- Compose the TTG by connecting inputs and outputs
  - Either via providing an `Edge` to the constructor or by manually connecting
- By default data are pushed to a successor and terminals act as single assignment variables
  - The first input argument to arrive causes the task to be instantiated
  - Once all arguments have arrived, the task is marked ready

Terminals

- Correspond to inputs/outputs of a task
- Compose the TTG by connecting inputs and outputs
  - Either via providing an `Edge` to the constructor or by manually connecting
- By default data are pushed to a successor and terminals act as single assignment variables
  - The first input argument to arrive causes the task to be instantiated
  - Once all arguments have arrived, the task is marked ready
- Terminals can be marked as pull
  - A logically preceding task is instantiated to send the data
    - E.g., reading from memory, generating data on the fly, recursive construction
  - Important optimizations
    - Directly call the operation to avoid task overhead
    - Greedy (at task creation) or lazy (when all other inputs available) pulling of data to control resource utilization

Terminals

- Correspond to inputs/outputs of a task
- Compose the TTG by connecting inputs and outputs
  - Either via providing an `Edge` to the constructor or by manually connecting
- By default data are pushed to a successor and terminals act as single assignment variables
  - The first input argument to arrive causes the task to be instantiated
  - Once all arguments have arrived, the task is marked ready
- Terminals can be marked as pull
  - A logically preceding task is instantiated to send the data
    - E.g., reading from memory, generating data on the fly, recursive construction
  - Important optimizations
    - Directly call the operation to avoid task overhead
    - Greedy (at task creation) or lazy (when all other inputs available) pulling of data to control resource utilization
- Terminals are programmable
  - E.g., general reduction operations, computation on streaming data

Examples of send and broadcast

```
void taskfn(const TaskID& task_id, const MatrixTile& input,
            tuple<Out<TaskID, MatrixTile>,
                  Out<TaskID, MatrixTile>,
                  Out<TaskID, MatrixTile>>& out)
{
  MatrixTile output = compute_output_tile(input);
  send<0>(task_id, output); // new copy required
  send<1>(task_id, move(output)); // no new copy due to move
  send<2>(task_id, input); // no new copy since input is const
}
```

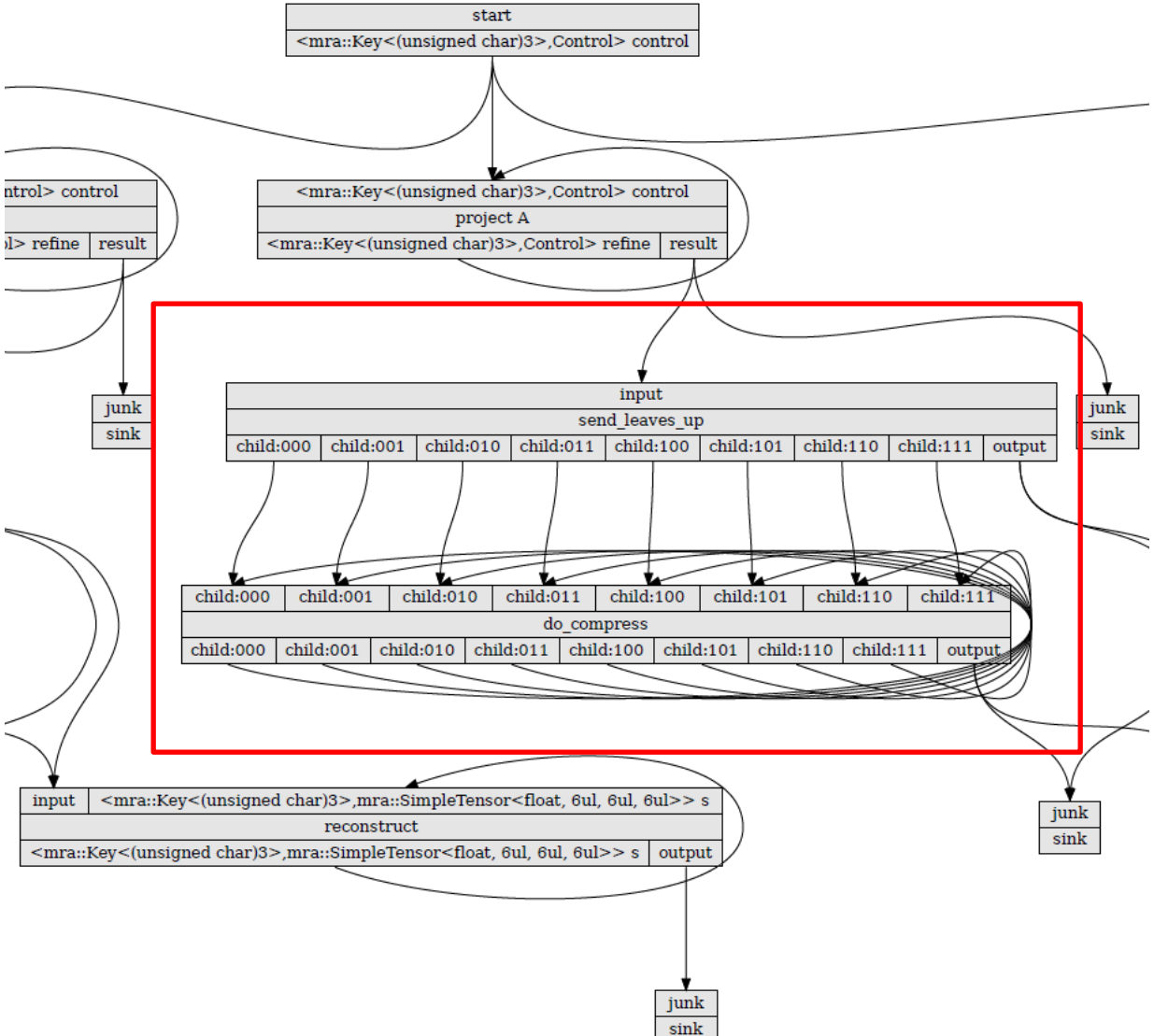
```
  /* broadcast the result to 4 output terminals:
   * 0: to final output task writing back the tile;
   * 1: to the SYRK kernel;
   * 2: to the gemm tasks on in row I;
   * 3: to the gemm tasks in column K; */
  ttg::broadcast<0, 1, 2, 3>(
    std::make_tuple(id, Int2(I, K), row_ids, col_ids),
    std::move(tile_mk), out);
```

Streaming Terminal Example

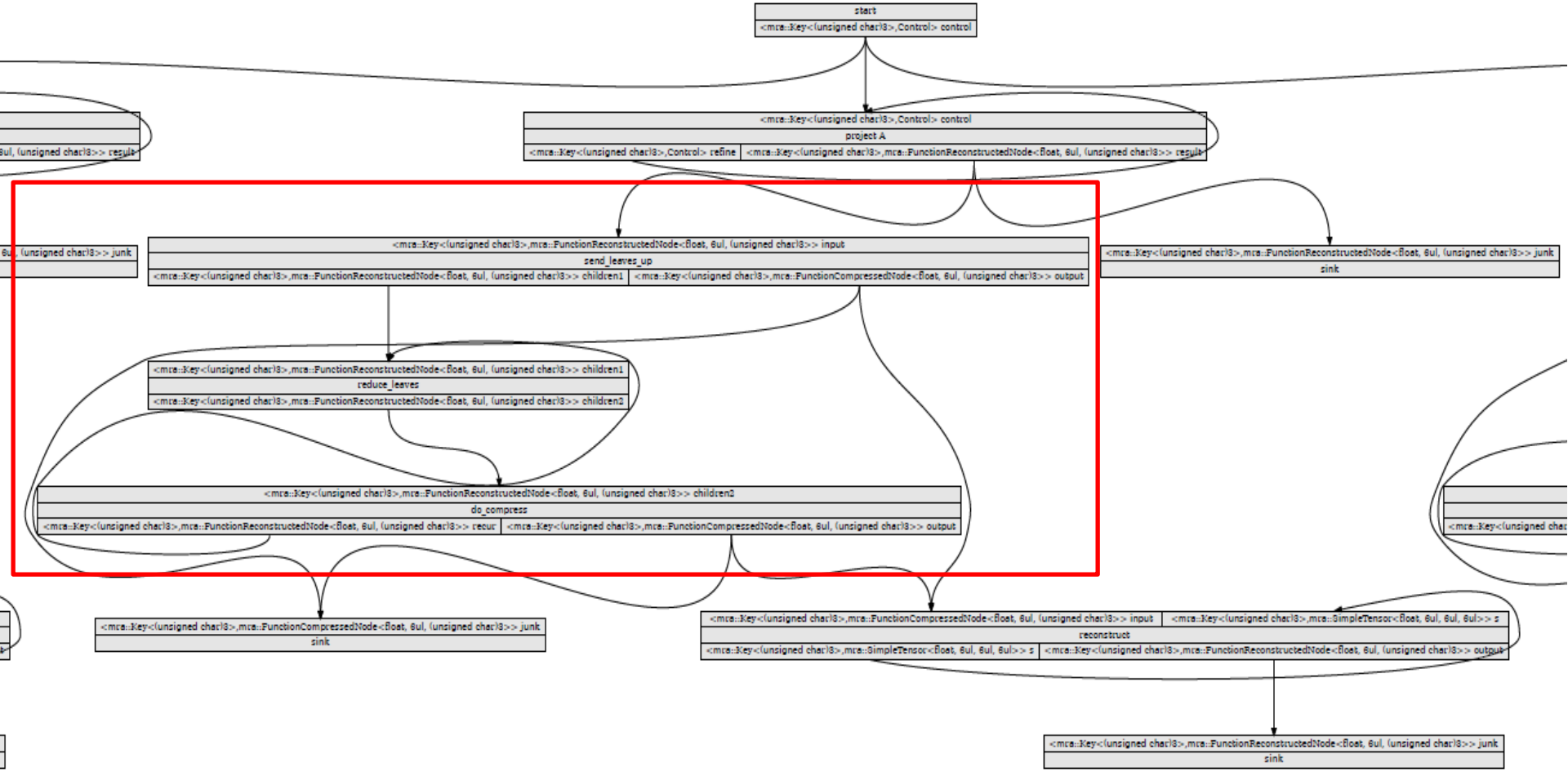
- MRA works on a 3D spatial tree data structure. There are 3 steps of computation and work/data flow down the tree in the projection and reconstruction steps and flows up the tree for the compression step.

```
reduce_leaves_op->template set_input_reducer<0>(
  /* the reduction operator */
  [](FunctionReconstructedNode<T,K,NDIM> &&a,
     FunctionReconstructedNode<T,K,NDIM> &&b)
  {
    a.neighbor_coeffs[a.key.childindex()] = a.coeffs;
    a.is_neighbor_leaf[a.key.childindex()] = a.is_leaf;
    a.neighbor_sum[a.key.childindex()] = a.sum;
    a.neighbor_coeffs[b.key.childindex()] = b.coeffs;
    a.is_neighbor_leaf[b.key.childindex()] = b.is_leaf;
    a.neighbor_sum[b.key.childindex()] = b.sum;
    return a;
  },
  1 << NDIM /* the number of reductions to perform */
);
```

MRA – TTG (prior to streaming terminals)

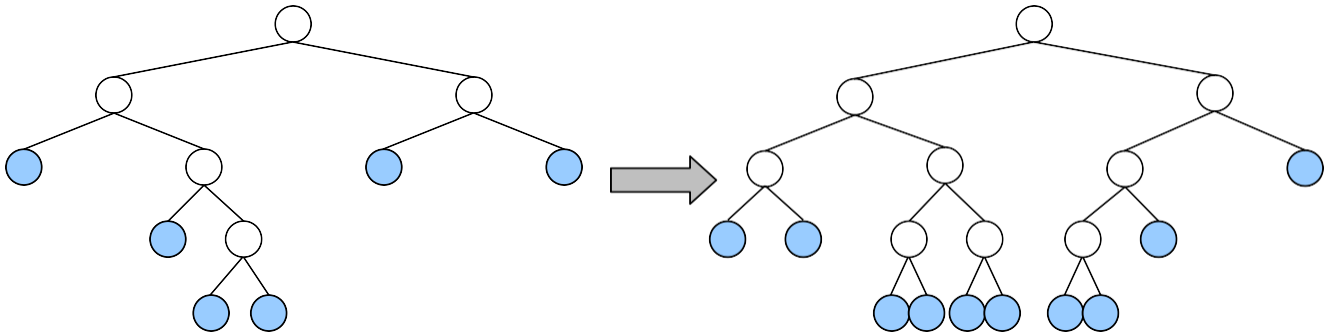


MRA – TTG (with streaming terminals)



Optimizing data motion

- PaRSEC internally tracks contiguous blocks of memory holding data
  - a.k.a. data copies or DCs
- Data allocated within a DC can be intelligently tracked through tasks
  - Constness deduced from C++ API
  - Objects can be moved into send operations to eliminate copying
  - Maintain copies of data in different memory regions
  - “Pure” tasks expose full data motion to the runtime



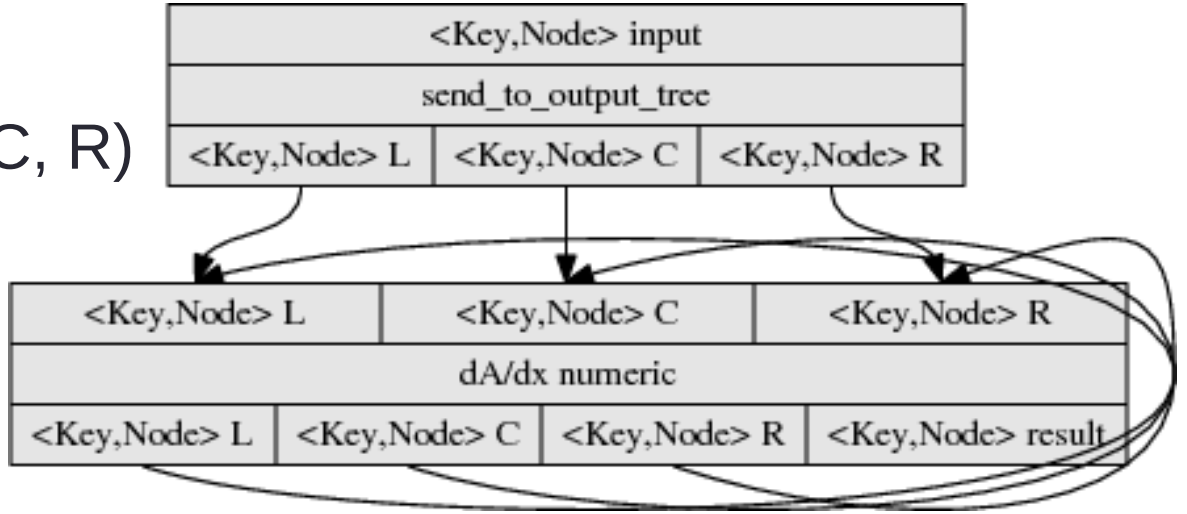
Differentiation (for simplicity here using central differences and Dirichlet boundary conditions) is applied in the scaling function basis. To compute the derivative of the function in the box corresponding to a leaf node, we require the coefficients from the neighboring boxes at the same level.

- If the neighboring leaf nodes exist, compute.
- If it exists at a higher level,we can make the coefficients by recurring down from the parent using the two-scale relation.
- If the neighbor exists at a finer scale, we must recur down until both neighbors are at the same level.

Hence, phrased as parallel computation on all leaf nodes , differentiation must search for neighbors in the tree at the same and higher levels, and may initiate computation at lower levels. It can also be phrased as a recursive descent of the tree, which can have advantages in reducing the amount of probes up the tree for parents of neighbors (esp. in higher dimensions).

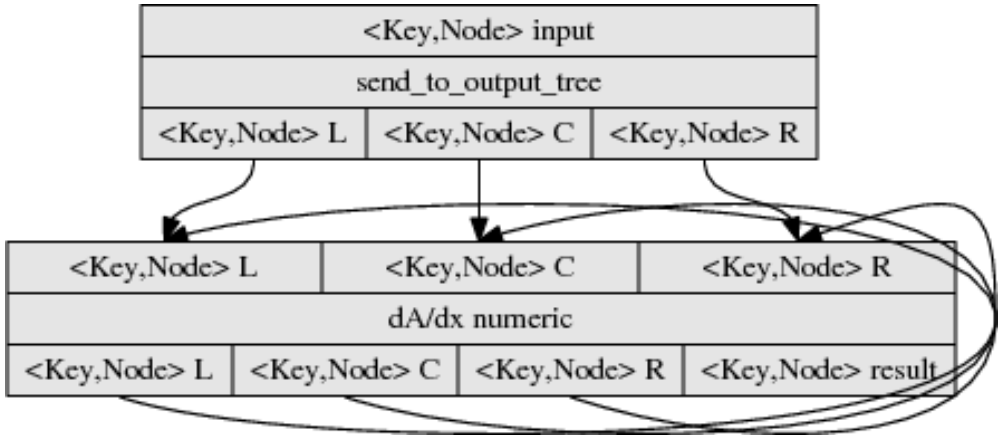
Central dif erence derivatives

- Logically we take an input function (tree) and produce an output function
  - 3-point stencil uses values on left, center, right
- Internally
  - Send input nodes to corresponding L, C, R nodes of output tree
  - Recur down output tree until all 3 (L, C, R) are available then compute result



Actual code for 1D Haar derivative – plumbing

```
using nodeEdge = Edge<Key,Node>;
using doubleEdge =
Edge<Key,double>;

using nodeOut = Out<Key,Node>;
using doubleOut =
Out<Key,double>;
```




```
auto make_diff(nodeEdge& input, nodeEdge& result)
{
    nodeEdge L, C, R;
    return make_tuple(make_tt(&send_to_output_tree, edges(input),
                      edges(L,C,R)), make_tt(&diff, edges(L,C,R),
                      edges(L,C,R,result)))
}
```

Actual code for 1D Haar derivative – computation & data flow

```
void send_to_output_tree(const Key& key, const Node& node, tuple<nodeOut,nodeOut,nodeOut>& out)
{
    send<0>(key.right(), node, out);
    send<1>(key, node, out);
    send<2>(key.left(), node, out);
}
void diff(const Key& key, const Node& left, const Node& center, const Node& right,
          tuple<nodeOut,nodeOut,nodeOut,nodeOut>& out)
{
    auto& [L,C,R,result] = out;
    if (!(left.has_children || center.has_children || right.has_children)) {
        double derivative = (right.s - left.s)/(4.0*::L*pow2(-key.n));
        result.send(key,Node(key,derivative,0.0,false));
    }
    else {
        result.send(key,Node(key,0.0,0.0,true));
        if (!left.has_children) L.send(key.left_child(), left);
        if (!center.has_children) {
            auto children = {key.left_child(),key.right_child()};
            L.send(key.right_child(),center);
            C.broadcast(children,center);
            R.send(key.left_child(), center);
        }
        if (!right.has_children) R.send(key.right_child(),right);
    }
}
```

Aim to have this auto generated from a high-level spec

TTG Status

- Prototype running on PaRSEC and MADNESS runtimes
- Currently
  - Completing design by identifying use cases and developing mini-apps
  - Tuning performance of irregularly-tiled, block-sparse tensor operations on multi-GPU nodes (e.g., Summit)
- Next steps
  - Complete first full implementation of specification along with an intermediate API to facilitate use of multiple runtimes
  - Demonstrate performance on components of full applications
  - Support additional runtimes including UCX, native C++, cloud stacks
  - Provide allocators over PaRSEC Dcs with optimized local and distributed data structures
- Seeking collaborators
  - New applications, new use cases, new runtime targets, developers, tool integration

Related Funding

- NSF OAC-1931387: Production quality Ecosystem for Programming and Executing eXtreme-scale Applications (EPEXA)
- NSF OAC-1927880: Ookami: A high-productivity path to frontiers of scientific discovery enabled by exascale system technologies
- NSF ACI-1450300: Task-Based Environment for Scientific Simulation at Extreme Scale (TESSE)
- NSF ACI-1141509: Dependence Programming and Optimization of Scalable Irregular Numerical Applications
- NSF (under subcontract from VT): Molecular Sciences Software Institute
- DOE Exascale Computing Project: NWChemEx