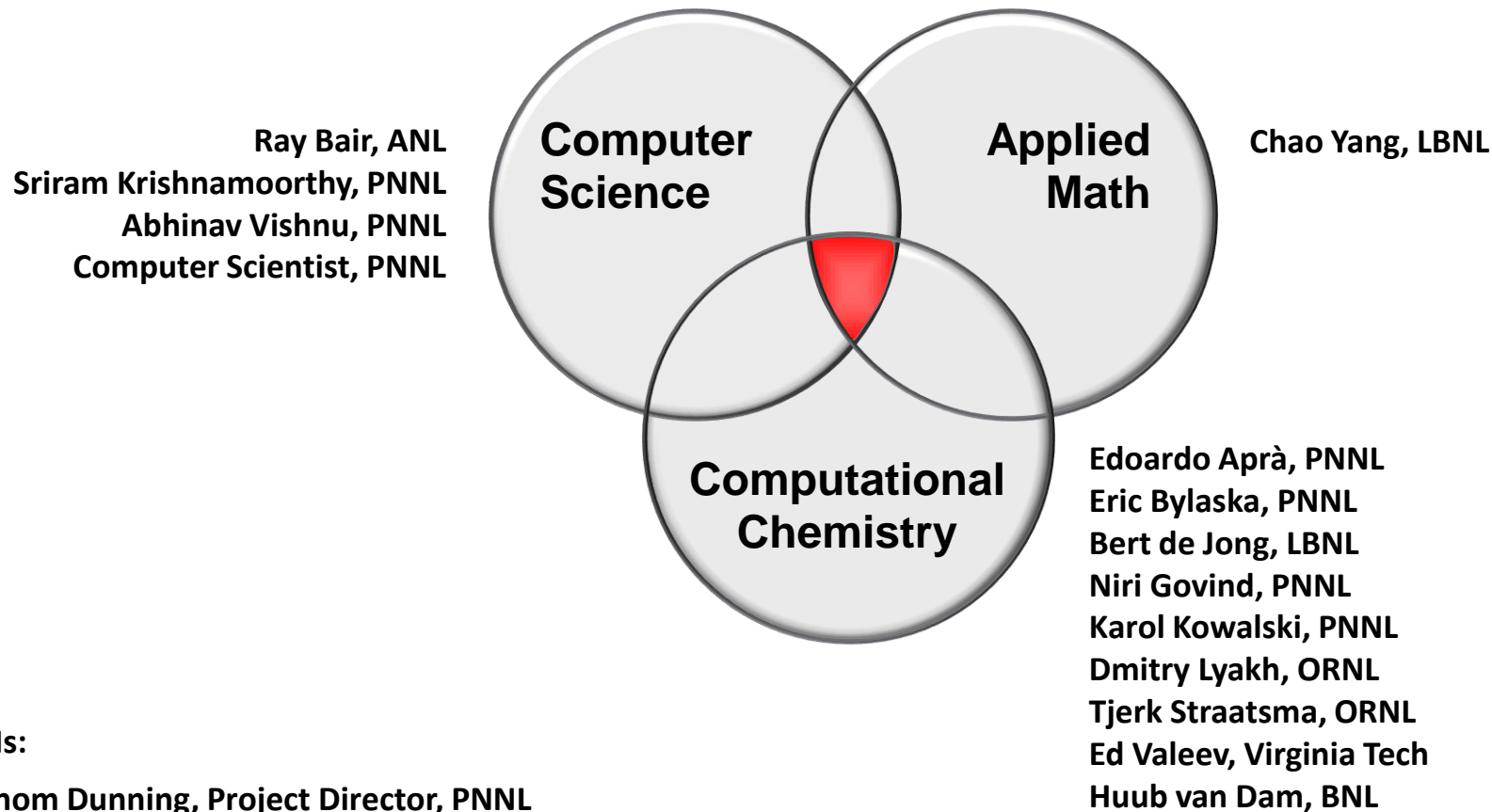


NWChemEx

Moving Computational Chemistry to the Exascale

Full NWChemEx Team:

Six national laboratories (Ames, ANL, BNL, LBNL, ORNL, PNNL) + Virginia Tech



PIs:

Thom Dunning, Project Director, PNNL

Theresa Windus, Deputy Project Director, Ames

Robert Harrison, Chief Architect, BNL

NWChemEx

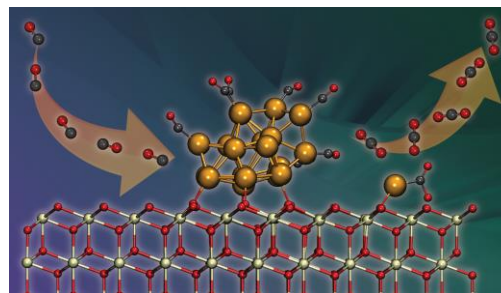
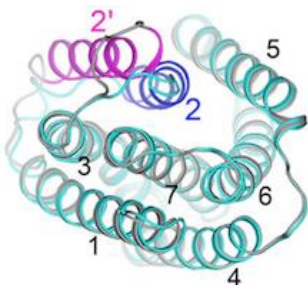
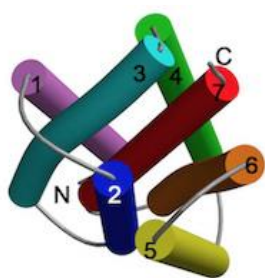
NWChemEx Target Problems

▶ DOE 2014-2018 Strategic Plan

- Development of high performance computing models that demonstrate that biomass can be a viable, sustainable feedstock for biofuels, hydrogen and other products

▶ Two Inter-related Challenges

- Efficient production of biomass
 - Development of detailed molecular model of transport processes across cellular membranes that control stress responses to aid in the development of stress-resistant crops (Q. Liu, BNL)
- Efficient conversion of biomass to biofuels
 - Development of detailed molecular model of catalytic conversion of biomass-derived alcohols to biofuels to aid in the discovery of low-temperature conversion processes (P. Sushko, PNNL)



Exascale Challenge

- ▶ Quantitative thermochemical information – reaction energies and barrier heights
- ▶ $O(10^3-10^4)$ Quantum mechanical atoms
 - Methods nominally scale as $N^6 - N^8$ where N is the number of basis functions
- ▶ $O(10^4-10^5)$ Environmental atoms (DFT embedding)
 - Methods nominally scale as $\sim N^3$
- ▶ Need excellent scaling software
 - Strong scaling w.r.t. parallel computer size
 - Reduced scaling w.r.t. system size (N)
- ▶ New mathematical techniques



NWChemEx features required to meet exascale science objectives

- ▶ High-performance programming paradigms and software ecosystem that:
 - Enables performance portable, where feasible, software implementations on exascale computing systems, and
 - Extends the programmer productivity and performance portability demonstrated by the TCE many-body methods to other NWChemEx modules.
- ▶ Strong scaling on anticipated exascale platforms for all algorithms in order to meet wall time constraints especially for molecular dynamics.
- ▶ Reduced-scaling, with respect to system size, algorithms for all levels of theory with robust error control.
- ▶ Elimination or controlled reduction of basis error for all levels of theory.
- ▶ A more extensible and less monolithic S/W arch. compared to NWChem.
- ▶ Facile and efficient support for anticipated exascale science workflows.



Initial priorities

- ▶ *Programming Model*: pervades everything else and will be our highest priority, noting there will not be a one-size fits all solution.
- ▶ *Performance and Productivity*: strive for portable performance and excellent strong parallel scaling
- ▶ *Reduced-scaling algorithms with strong error control*: are essential for enabling new science at the exascale, and are feasible at this scale
 - $1000^{1/1} = 1000$
 - $1000^{1/2} = 31.6$
 - $1000^{1/3} = 10$
 - $1000^{1/7} = 2.7$
- ▶ *Design space exploration and reducing technical uncertainty*: using a combination of theoretical investigations and proof-of-principle prototyping.



Slices through the NWChemEx software stack



Figure 2. Sandbox for many-body methods.



Figure 3. Sandbox for one-electron methods.

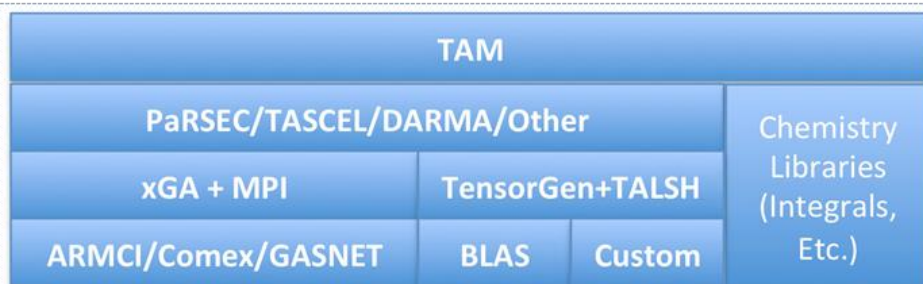


Figure 4. Sandbox for the next generation tensor contraction engine (TAM).



Figure 5. Sandbox for data management.



Example TAMD input (Tensor Algebra for Many-body Methods)

```
t1 {  
  
  index h1,h2,h3,h4,h5,h6,h7,h8 = 0;  
  index p1,p2,p3,p4,p5,p6,p7 = V;  
  
  array i0[V][O];  
  array f[N][N]: irrep_f;  
  array v[N,N][N,N]: irrep_v;  
  array t_vo[V][O]: irrep_t;  
  array t_vvoo[V,V][O,O]: irrep_t;  
  array t1_2_1[O][O];  
  array t1_2_2_1[O][V];  
  array t1_3_1[V][V];  
  array t1_5_1[O][V];  
  array t1_6_1[O,O][O,V];  
  
  t1_1:      i0[p2,h1] += 1 * f[p2,h1];  
  t1_2_1:    t1_2_1[h7,h1] += 1 * f[h7,h1];  
  t1_2_2_1:  t1_2_2_1[h7,p3] += 1 * f[h7,p3];  
  t1_2_2_2:  t1_2_2_1[h7,p3] += -1 * t_vo[p5,h6] * v[h6,h7,p3,p5];  
  t1_2_2:    t1_2_1[h7,h1] += 1 * t_vo[p3,h1] * t1_2_2_1[h7,p3];  
  t1_2_3:    t1_2_1[h7,h1] += -1 * t_vo[p4,h5] * v[h5,h7,h1,p4];  
  t1_2_4:    t1_2_1[h7,h1] += -1/2 * t_vvoo[p3,p4,h1,h5] * v[h5,h7,p3,p4];  
  t1_2:      i0[p2,h1] += -1 * t_vo[p2,h7] * t1_2_1[h7,h1];  
  t1_3_1:    t1_3_1[p2,p3] += 1 * f[p2,p3];  
  t1_3_2:    t1_3_1[p2,p3] += -1 * t_vo[p4,h5] * v[h5,p2,p3,p4];  
  t1_3:      i0[p2,h1] += 1 * t_vo[p3,h1] * t1_3_1[p2,p3];  
  t1_4:      i0[p2,h1] += -1 * t_vo[p3,h4] * v[h4,p2,h1,p3];  
  t1_5_1:    t1_5_1[h8,p7] += 1 * f[h8,p7];  
  t1_5_2:    t1_5_1[h8,p7] += 1 * t_vo[p5,h6] * v[h6,h8,p5,p7];  
  t1_5:      i0[p2,h1] += 1 * t_vvoo[p2,p7,h1,h8] * t1_5_1[h8,p7];  
  t1_6_1:    t1_6_1[h4,h5,h1,p3] += 1 * v[h4,h5,h1,p3];  
  t1_6_2:    t1_6_1[h4,h5,h1,p3] += -1 * t_vo[p6,h1] * v[h4,h5,p3,p6];  
  t1_6:      i0[p2,h1] += -1/2 * t_vvoo[p2,p3,h4,h5] * t1_6_1[h4,h5,h1,p3];  
  t1_7:      i0[p2,h1] += -1/2 * t_vvoo[p3,p4,h1,h5] * v[h5,p2,p3,p4];  
  
}
```

End of “serial” computing

- Parallelism is now the only path to increased performance
 - Simultaneously executing multiple operations and entire tasks
- How much is necessary for peak performance?

Computer	Ops/cycle
Mythical serial computer	1
2013 Intel desktop chip	32
2015 Intel desktop chip	64-128
2020 Intel desktop chip	1024
2012 Intel MIC chip	1024
2013 Supercomputer	10^7
2022 Supercomputer	10^9

I.e., this affects *everyone*, not just supercomputer users.

If your code is serial, it will run slower by up to this factor.

Growth in parallelism mostly on chip (vectors, cores, ...)

- Actually started in mid 1990s but we collectively ignored it

Preparing for the computing future

- Science and computer science students commonly not prepared for this future
 - Taught sequential not parallel programming
 - Little awareness of performance or architecture
- Many computer programs written now will live for at least 10 years into the future
 - But designed to run on computers from 10 years past
- Individual research groups, small institutions, disciplines new to computing, most companies, do not have the awareness, skills, resources, to navigate this transition



The Molecular Sciences Software Institute

T. Daniel Crawford (PI), Cecilia Clementi, Robert Harrison,
Teresa Head-Gordon, Shantenu Jha, Anna Krylov,
Vijay Pande, and Theresa Windus

The Molecular Sciences Software Institute (MoSSI)

Education, Outreach, and Training

- MoSSI will serve as an education and outreach nexus for the worldwide CMS community.
- MoSSI will organize summer schools, targeted workshops, high-school and undergraduate training programs, and on-line resources and classes to provide current and future CMS students with a modern and complete set of programming skills.
- MoSSI will reach beyond the traditional student cohort to computer scientists and mathematicians seeking interdisciplinary applications.
- MoSSI will deploy a Professional Master's program in Molecular Simulation and Software Engineering.

The Molecular Sciences Software Institute (MolSSI)

Community Engagement and Leadership

- MolSSI will enable the CMS community to establish its own standards for interoperability, best practices, and curation tools.
- Through a “**grass roots**” approach, MolSSI will engage the community broadly using interoperability workshops and focus groups – and ultimately the formation of a Molecular Sciences Consortium – to catalyze the consensus needed for standardization of data structures, APIs, and frameworks for the entire CMS software ecosystem.

The Molecular Sciences Software Institute (MoISSI)

Expertise

- MoISSI will work with CMS research groups nationwide and internationally to design, develop, test, deploy, and maintain key code infrastructure and frameworks for the entire community.
- MoISSI will interact with partners in industry, NSF supercomputing centers, national laboratories, and international facilities to identify and act on emerging hardware trends, access leading-edge computing architectures, further educational goals, set software priorities, and identify future workforce career paths.

MADNESS - parallel runtime, and the task and data-flow programming paradigms

Robert J. Harrison

Institute for Advanced Computational Science
Stony Brook University

and

Center for Scientific Computing
Brookhaven National Laboratory

robert.harrison@stonybrook.edu



National Science Foundation
WHERE DISCOVERIES BEGIN



Stony Brook University



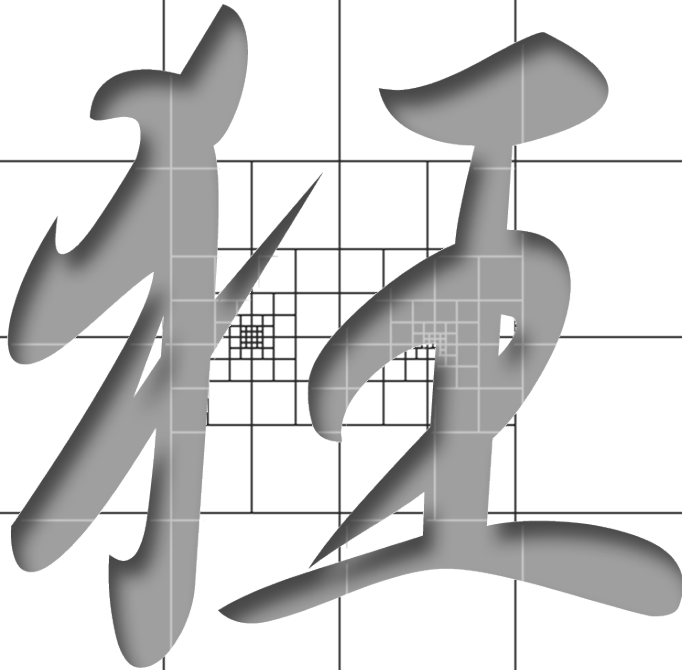
iACS
INSTITUTE FOR ADVANCED
COMPUTATIONAL SCIENCE

M

A

D

N



T

S



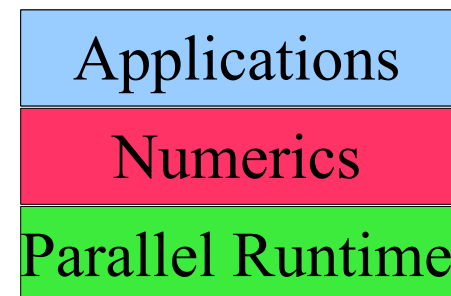
Multiresolution
Adaptive
Numerical
Scientific
Simulation

S

What is 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>



E.g., with guaranteed precision of $1e-6$ form a numerical representation of a Gaussian in the cube $[-20,20]^3$, solve Poisson's equation, and plot the resulting potential
(all running in parallel with threads+MPI)

Let

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

$$\epsilon = 1e - 6$$

$$g = x \rightarrow \exp(- (x_0^2 + x_1^2 + x_2^2)) * \pi^{-1.5}$$

In

$$f = \mathcal{F} g$$

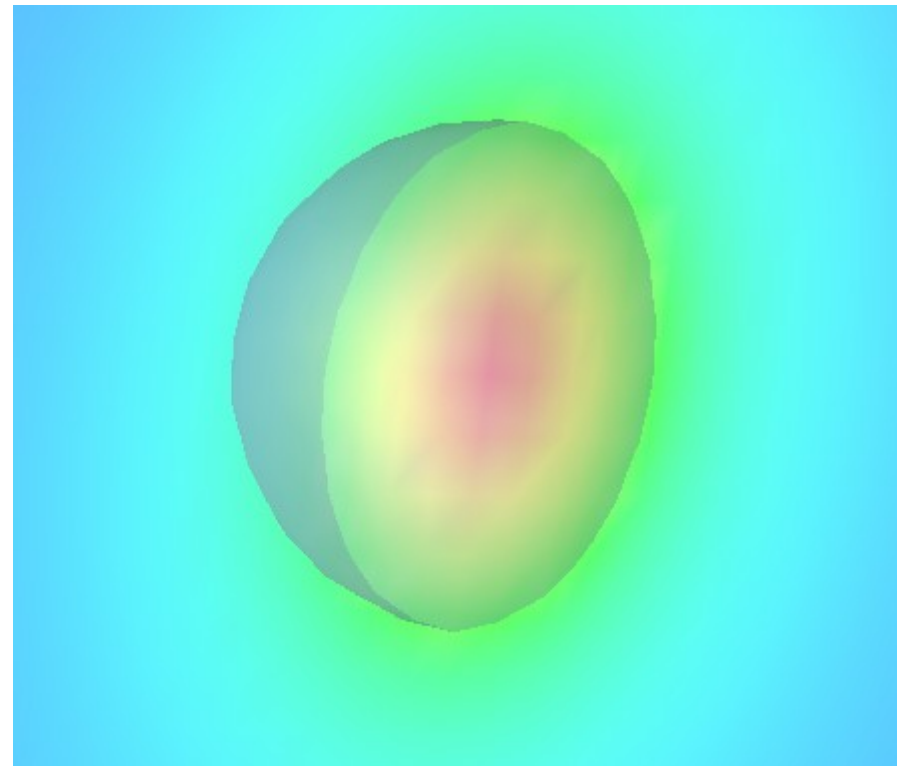
$$u = \nabla^{-2} (-4 * \pi * f)$$

```
print "norm of f", <f>, "energy", <f|u> * 0.5
```

```
plot u
```

End

output: norm of f 1.00000000e+00 energy 3.98920526e-01



There are only two lines doing real work. First the Gaussian (g) is projected into the adaptive basis to the default precision. Second, the Green's function is applied. The exact results are norm=1.0 and energy=0.3989422804.

Let

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

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

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

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

In

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

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

$$\lambda = -1.0$$

for $i \in [0, 10]$

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

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

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

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

$$\phi = \psi$$

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

end

End

He atom Hartree-Fock

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 output a C++ code that can be compiled without modification and run in parallel

The math behind the MADNESS

- Multiresolution

$$V_0 \subset V_1 \subset \dots \subset V_n$$
$$V_n = V_0 + (V_1 - V_0) + \dots + (V_n - V_{n-1})$$

- Low-separation rank

$$f(x_1, \dots, x_n) = \sum_{l=1}^M \sigma_l \prod_{i=1}^d f_i^{(l)}(x_i) + O(\epsilon)$$
$$\|f_i^{(l)}\|_2 = 1 \quad \sigma_l > 0$$

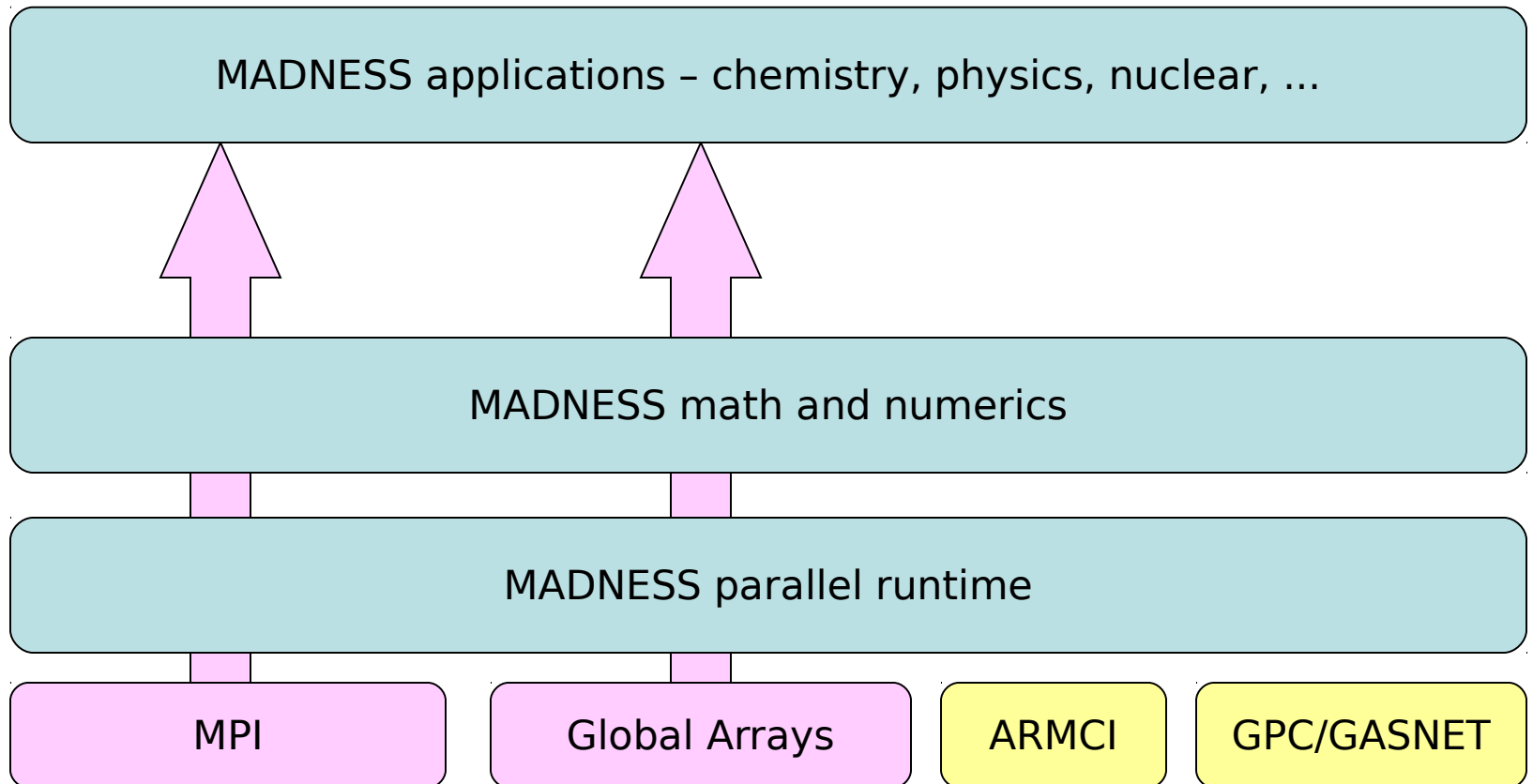
- Low-operator rank

$$A = \sum_{\mu=1}^r u_{\mu} \sigma_{\mu} v_{\mu}^T + O(\epsilon)$$
$$\sigma_{\mu} > 0 \quad v_{\mu}^T v_{\lambda} = u_{\mu}^T u_{\lambda} = \delta_{\mu\nu}$$

A Key Component

- Trade precision for speed – everywhere
 - Don't do anything exactly
 - Perform everything to $O(\epsilon)$
 - Require
 - Robustness
 - Speed, and
 - Guaranteed, arbitrary, *finite* precision
- Leads to very irregular and dynamic data structures and computation
 - Trees can routinely go down 30+ levels

MADNESS architecture



Intranode runtime – original thread pool, Intel TBB, PaRSEC

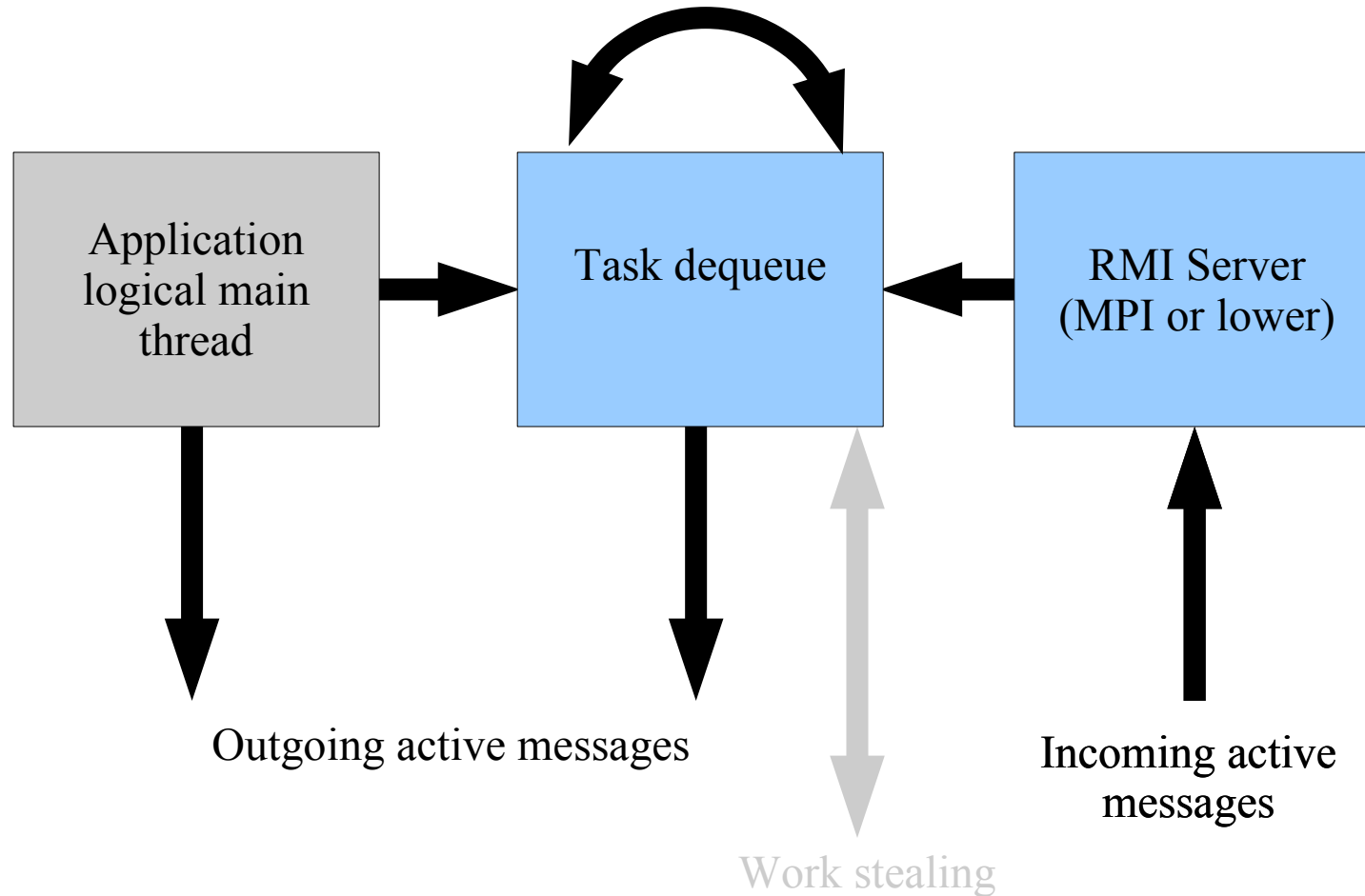
Why a new runtime?

- MADNESS computation is irregular & dynamic
 - 1000s of dynamically-refined meshes changing frequently & independently (to guarantee precision)
- Because we wanted to make MADNESS itself easier to write not just the applications using it
 - We explored implementations with MPI, Global Arrays, and Charm++ and all were inadequate
- MADNESS helped drive
 - One-sided operations in MPI-3, DOE projects in fault tolerance, ...

Key runtime elements

- Futures for hiding latency and automating dependency management
- Global names and name spaces
- Non-process centric computing
 - One-sided messaging between objects
 - Retain place=process for MPI/GA legacy compatibility
- Dynamic load balancing
 - Data redistribution, work stealing, randomization

Multi-threaded architecture



Tasks

- Basic API takes pointer to object derived from

```
class TaskInterface {
```

```
public:
```

```
    virtual void run();
```

```
}
```

- User implements run function and adds task to queue (that takes ownership of ptr) using

- Can add priority

```
world.taskq.add(pointer_to_task)
```

C++ templates automate many things

- Wrapping calls to functions or lambdas or object member functions inside a task
- Managing dependencies by connecting futures and tasks through the dependency interface (callbacks)
- Assisting programmers to avoid explicit use of futures as arguments to functions
- (de)Serializing arguments to remote tasks
- And templates help avoid execution overhead (compile time)

Success with tasks

- Can readily compose parallel versions of many complex and irregular algorithms
- Can obtain good performance
 - On multi-threaded CPUs
 - With attention to details
 - Especially effective for irregular work loads and complex, even dynamic, dependencies

Problems with tasks without compiler+intelligent runtime

- Manual continuation passing - everywhere code may block need to partition into separate tasks (zillions of them!)
- Rigid decisions made about granularity and decomposition
- Hard to automatically aggregate many small tasks
- Makes portability from CPU to GPU worse
- Resource management – bounding buffers and not drowning/starving in parallelism: partially manage with task generators, data flow, task priorities
- Efficient execution challenging esp. use of memory hierarchy, scheduling of critical path
- Makes easy things harder (e.g., parallel for)
- Task specification often distant from task use - very effective for code obfuscation – fixed in modern C++
- Interoperability with just about everything else is painful

Who (what) should be using task-based composition?

- Not people with substantial codes unless have
 - Compiler support
 - To keep the easy stuff easy
 - To automate partitioning and capturing of state
 - To automate/facilitate adjustment of granularity
 - To assist in CPU/GPU portability
 - Intelligent runtime
 - To assist in scheduling/placement, efficient use of memory hierarchy, ...
 - Code generation from DSLs
 - Since the magical compiler is a myth

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

Stony Brook University

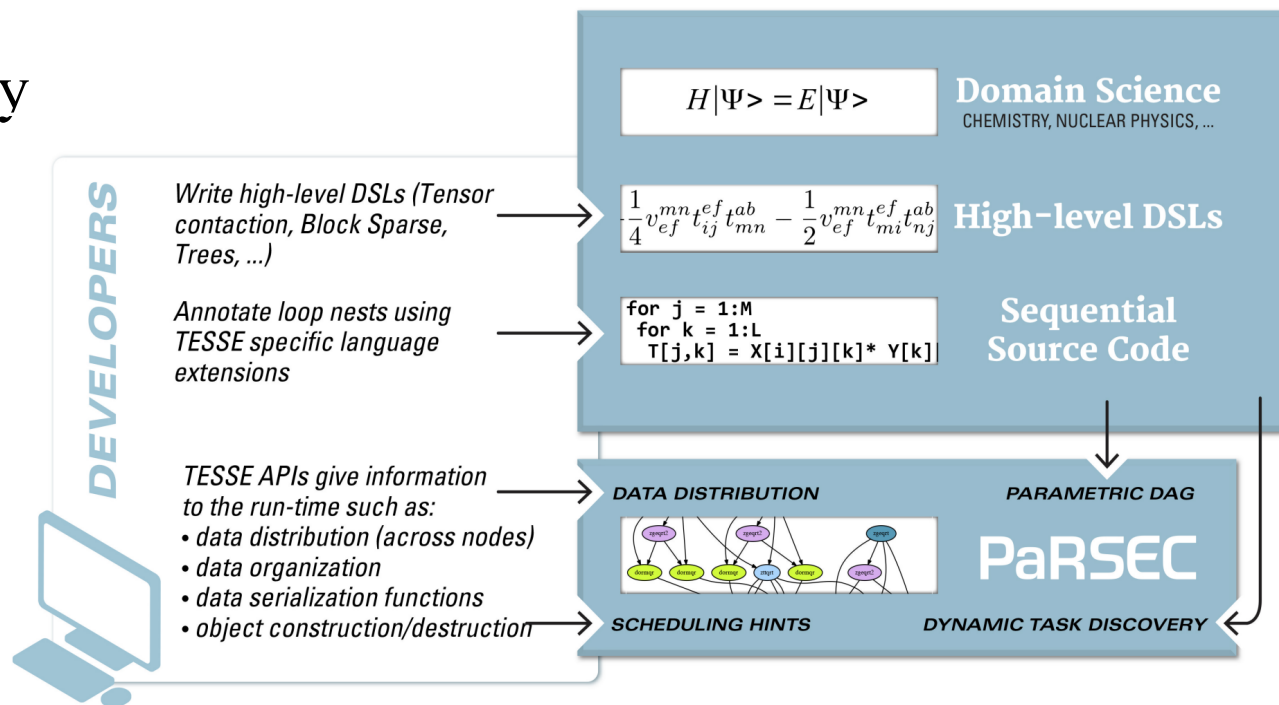
- Harrison

University of Tennessee

- Bosilca and Herault

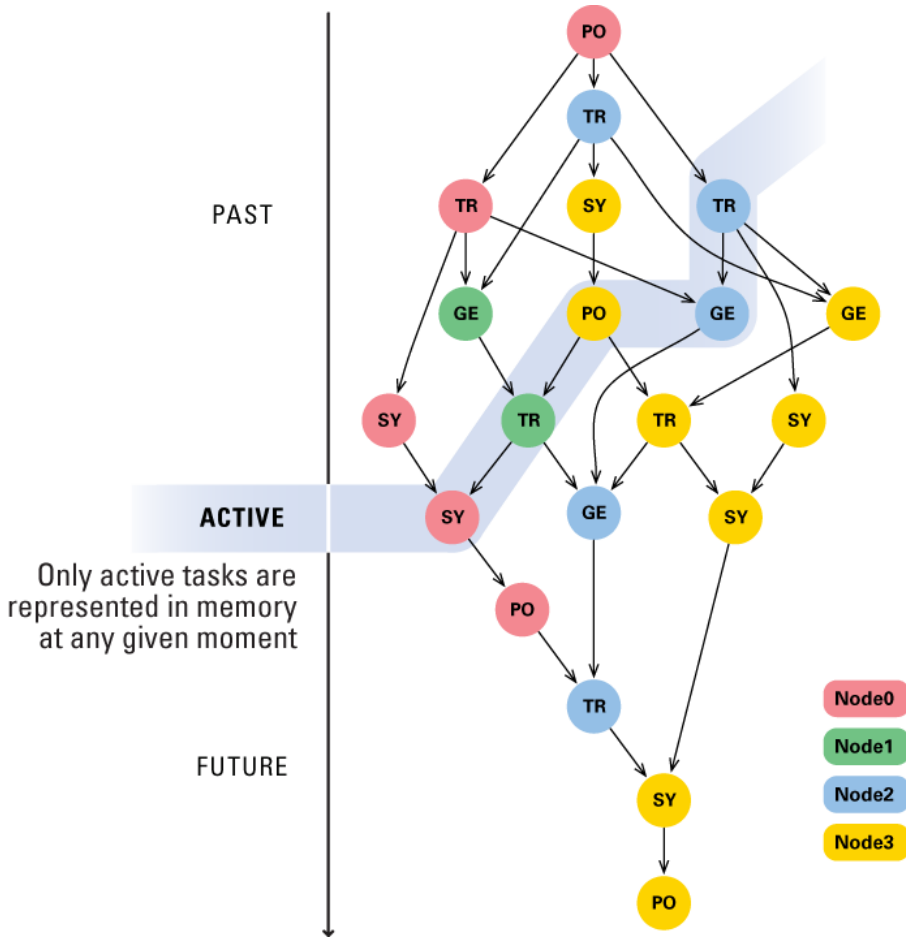
Virginia Tech

- Valeev



Application-driven design of a general-purpose and production quality software framework addressing programmer productivity and portable performance for advanced scientific applications on massively-parallel, hybrid, many-core systems of today and tomorrow.

TESSE



- Extends Parallel Scheduling and Execution Controller (PaRSEC) to larger classes of dynamic (data-dependent) computation; data distribution; composition and execution of multiple DAGs
- Address
 - 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

Emerging TESSE data flow model

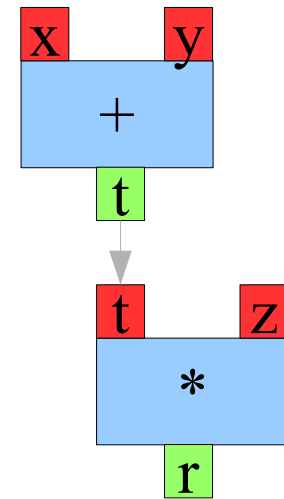
- Overall objectives
 - High-level expression of irregular algorithms
 - Data flow model enables readily parallelized implementation due to lack of side effects
 - Permit side effects for generality and optimization
 - Runtime needs to understand associated data structures
 - Extend powerful PaRSEC runtime to manage dependencies and scheduling
 - Eliminate all non-essential barriers
 - Express all available parallelism without drowning

Emerging TESSE data flow model

- Driven by
 - MADNESS: Algorithms on unbalanced, deep spatial trees in 1-6 dimensions
 - TiledArray: Block-sparse algorithms for tensors with 2-6 or more dimensions
 - Sparse linear algebra: element/block sparsity for matrices
 - Dense linear algebra: original users of PaRSEC
- Related projects
 - CnC, Legion, Charm++, DARMA, ...
 - Mini app. being ported in collab. with these teams

Elements of TESSE data flow model

- We are describing the computation as a graph flowing data through a sequence of operations
 - E.g., consider
for (i=0; i<n; i++) r[i] = (x[i]+y[i])*z[i];
- Data is flowing into each operation via its arguments and out via its results
- Making the graph is nothing more than connecting inputs with outputs
- Each task that will execute the operation is labelled by a key
 - E.g., a loop index (i) making a separate task for each iteration
 - The key labels where the task will execute

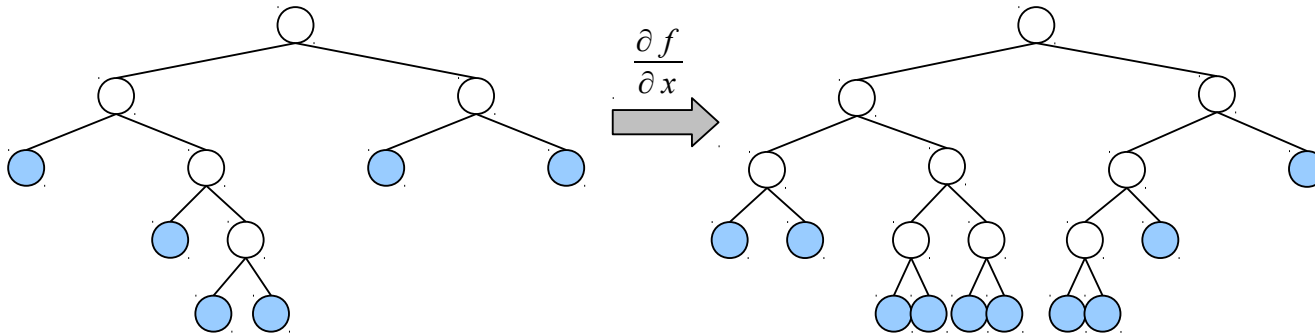


Current status

- Mini-app done in Python, CnC, MADNESS, OpenMP and in progress for Legion
- In prototype Flow API have demonstrated
 - All example 1-D MADNESS tree algorithms
 - Simple dense SUMMA
 - Simple block-sparse SUMMA
- Next steps
 - In progress, fully native implementation with PaRSEC
 - n-D MADNESS numerical algorithms
 - TiledArray
 - Integration with MADNESS distributed data structures

Summary

- MADNESS parallel runtime designed to support very irregular and dynamic computation
 - Task-based composition very powerful but is not a magical solution and has real negatives
 - Task-based execution also very powerful but demands a combination of static and runtime tools
- Need an entire ecosystem of interoperable programming models, tools and runtimes
 - Leverage success of data flow model in modern linear algebra community (with compiler & runtime support)



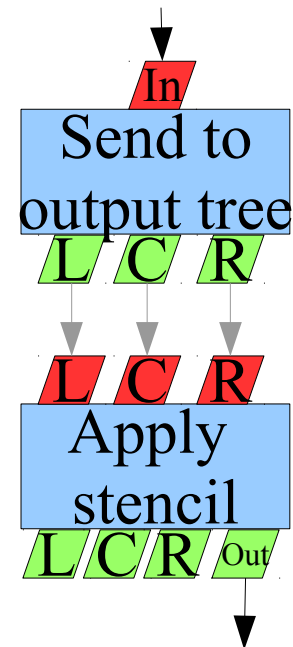
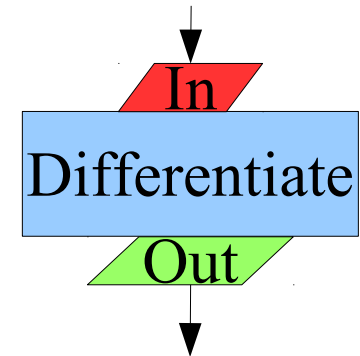
Differentiation (for simplicity here using central differences and Dirichlet boundary conditions) is applied in the scaling function basis. To compute the derivative of the function in the box corresponding to a leaf node, we require the coefficients from the neighboring boxes at the same level.

- If the neighboring leaf nodes exist, all is easy.
- 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).

Differentiation

- Logically we take an input function (tree) and produce an output
- Internally
 - We must send the input tree nodes to the corresponding L, C, R nodes of the output tree
 - Recur down the output tree until all 3 (L, C, R) are available then compute the output



Actual implementation in 1D

```
void send_to_output_tree(const Key& key, const Node& in,
                        Flows<nodeFlow,nodeFlow,nodeFlow>& out)
{
    nodeFlow L, C, R; std::tie(L, C, R) = out.all(); // Unpack output flows
    L.send(key.right(), in);
    C.send(key, in);
    R.send(key.left(), in);
}

void apply(const Key& key, const Node& left, const Node& center, const Node& right,
           Flows<nodeFlow,nodeFlow,nodeFlow,nodeFlow>& outputs)
{
    nodeFlow L, C, R, out; std::tie(L,C,R,out) = outputs.all();
    if (!(left.has_children || center.has_children || right.has_children)) {
        double derivative = (right.s - left.s)/(4.0*::L*pow2(-key.n));
        out.send(key,Node(key,derivative,0.0,false));
    }
    else {
        out.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);
    }
}

auto make_diff(nodeFlow in, nodeFlow out) {
    nodeFlow L, C, R;
    return std::make_tuple(make_op_wrapper(&send_to_output_tree, in, make_flows(L,C,R)),
                          make_op_wrapper(&diff, make_flows(L,C,R), make_flows(L,C,R,out)));
}
```

Aim to have this
auto generated from
a high-level spec in
a simple declarative
language.