HIGH-PERFORMANCE WORKFLOWS WITH SWIFT/T

JUSTIN M WOZNIAK
MCS Division
`wozniak@mcs.anl.gov`

Blue Waters Webinar — September 20, 2017

EXTREME-SCALE WORKFLOWS

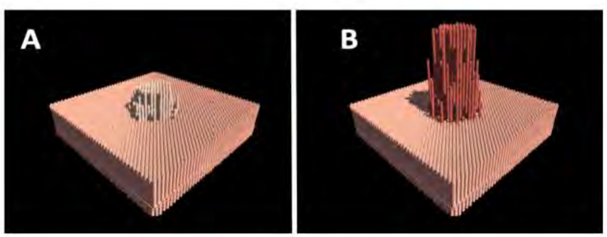
U. CHICAGO HOSPITALS: CANCER ENSEMBLES

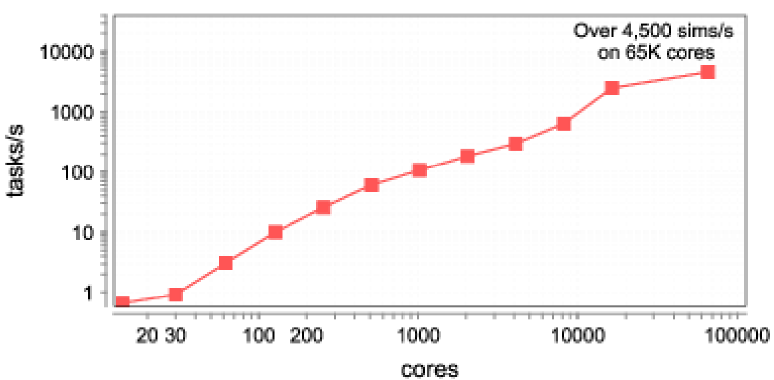
Best paper at SC Cancer Workshop 2016

- Parameter fitting for biological phenomenon (DNA repair rate) via massive scale evolutionary algorithm in Swift/T framework

GIOABM – Integration into SEGMEnT

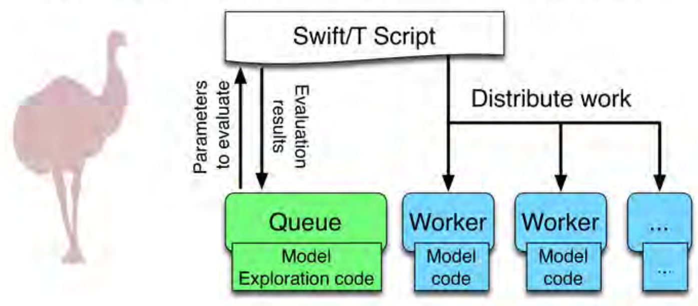
- A cancerous cell has three features: immortality, invasiveness, and ability to proliferate unnaturally
- GIOABM call functionality overlaps with SEGMEnT at four locations:
  - B-catenin: proliferation
  - PI3K: Proliferation/Apoptosis
  - TGF-B/SMAD: Proliferation/Apoptosis
  - P53: Gene repair/Apoptosis
- Added E-cadherin protein mutation to SEGMEnT representing invasiveness





Extreme-scale Model Exploration with Swift (EMEWS)

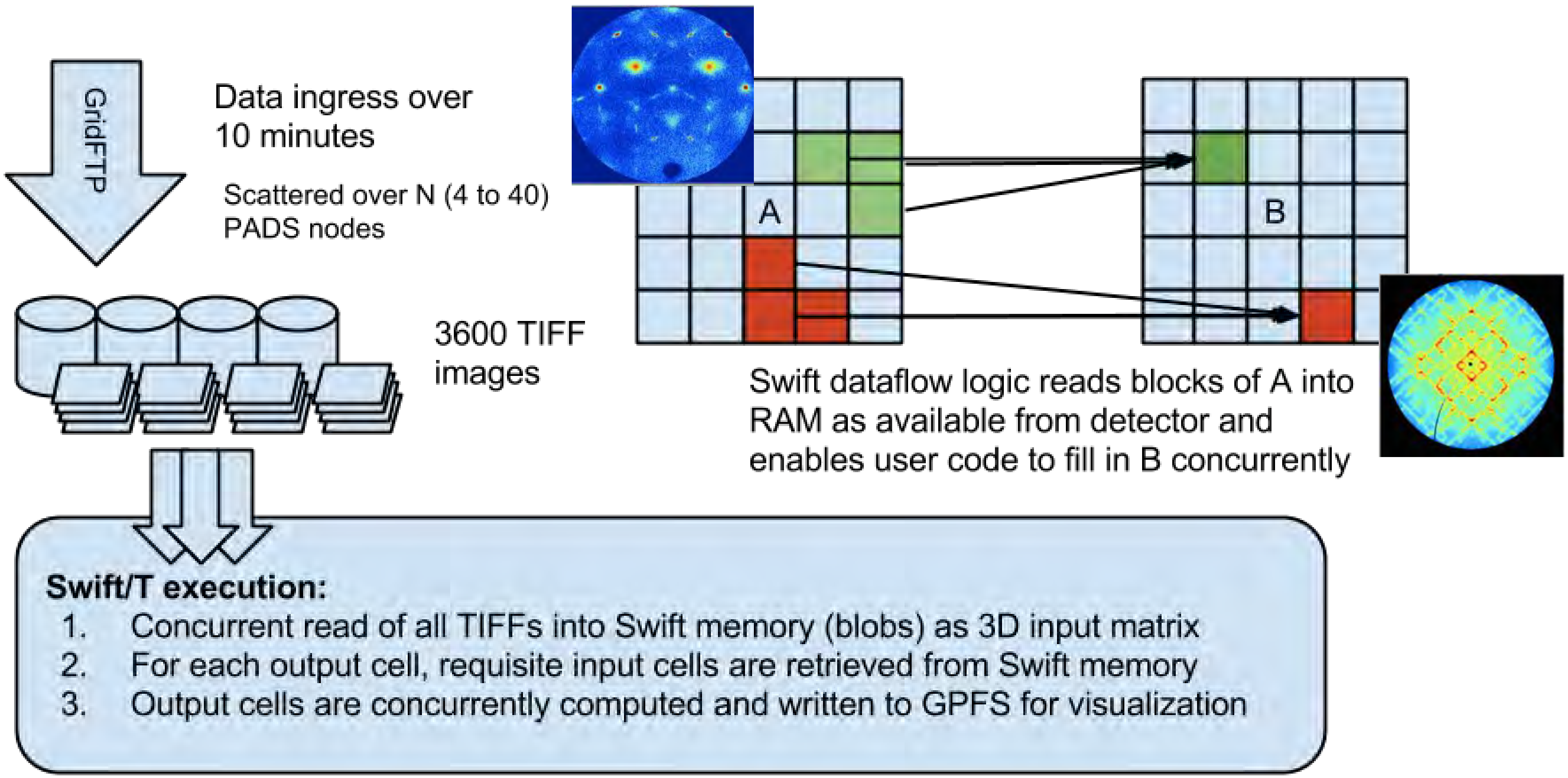
- EMEWS offers:
  - the capability to run very large, highly concurrent ensembles of simulations of varying types
  - supports a wide class of ME algorithms, including those increasingly available to the community via Python and R libraries
- EMEWS design goal: to ease software integration while providing scalability to the largest scale (petascale plus) supercomputers, running millions of models



- **Anatomic-scale cancer modeling using the Extreme-scale Model Exploration with Swift (EMEWS) framework.** Proc. CACW @ SC, 2016

APS: CRYSTAL COORDINATE TRANSFORMATION WORKFLOW

MapReduce-like pattern expressed elegantly in Swift



SWIFT/T SYSTEM OVERVIEW

LANGUAGE GOALS

Hierarchical, naturally parallel, script-like programming

- Make it easy to run large batteries of external program or library executions
- Provide rich programming language at the top level – fully generic
- Support implicit concurrency and conventional programming constructs
- Enable complex tasks based in other scripting languages (e.g., Python) or parallel MPI tasks
- Other powerful features- rich data types, resource management, …

THE SWIFT PROGRAMMING MODEL

All progress driven by concurrent dataflow

```
(int r) myproc (int i, int j)
{
    int x = F(i);
    int y = G(j);
    r = x + y;
}
```

- `F()` and `G()` implemented in native code or external programs
- `F()` and `G()` run in concurrently in different processes
- `r` is computed when they are both done
- This parallelism is *automatic*
- Works recursively throughout the program’s call graph

SWIFT SYNTAX

- Data types

```
int i = 4;
string s = "hello world";
file image<"snapshot.jpg">;
```

- Shell access

```
app (file o) myapp(file f, int i)
{ mysim  "-s" i @f @o; }
```

- Structured data

```
typedef image file;
image A[];
type protein_run {
  file pdb_in; file sim_out;
}
bag<blob>[] B;
```

- Conventional expressions

```
if (x == 3) {
    y = x+2;
    s = strcat("y: ", y);
}
```

- Parallel loops

```
foreach f,i in A {
    B[i] = convert(A[i]);
}
```

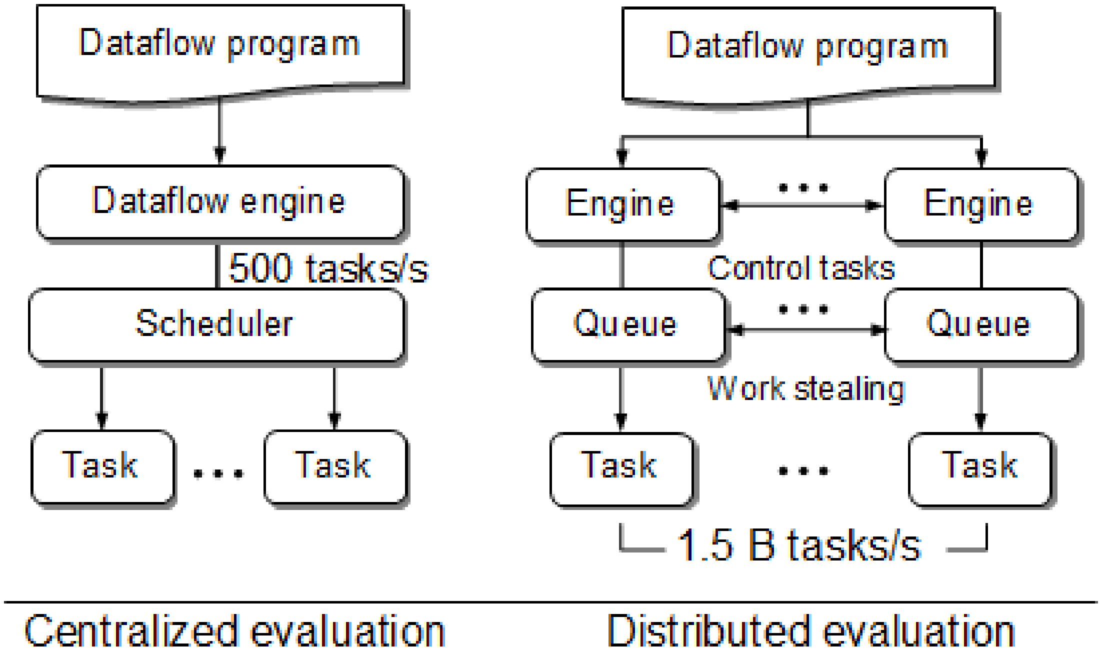
- Data flow

```
merge(analyze(B[0], B[1]),
      analyze(B[2], B[3]));
```


- **Swift: A language for distributed parallel scripting.** J. Parallel Computing, 2011
- **Compiler techniques for massively scalable implicit task parallelism.** Proc. SC, 2014

CENTRALIZED EVALUATION IS A BOTTLENECK AT EXTREME SCALES

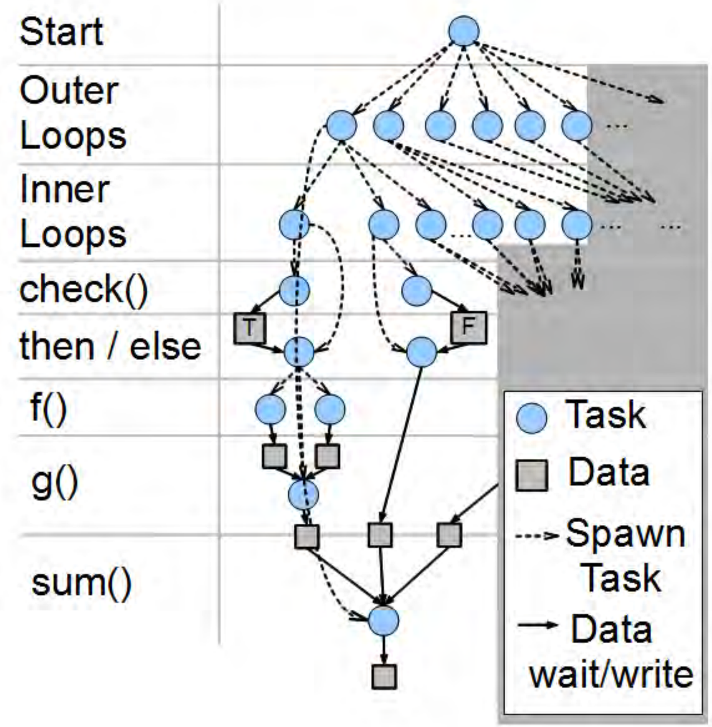
Had this (Swift/K): Now have this (Swift/T):



- **Turbine: A distributed-memory dataflow engine for high performance many-task applications.** Fundamenta Informaticae 28(3), 2013

SWIFT/T: FULLY PARALLEL EVALUATION OF COMPLEX SCRIPTS

```
int X = 100, Y = 100;
int A[][];
int B[];
foreach x in [0:X-1] {
  foreach y in [0:Y-1] {
    if (check(x, y)) {
      A[x][y] = g(f(x), f(y));
    } else {
      A[x][y] = 0;
    }
  }
  B[x] = sum(A[x]);
}
```



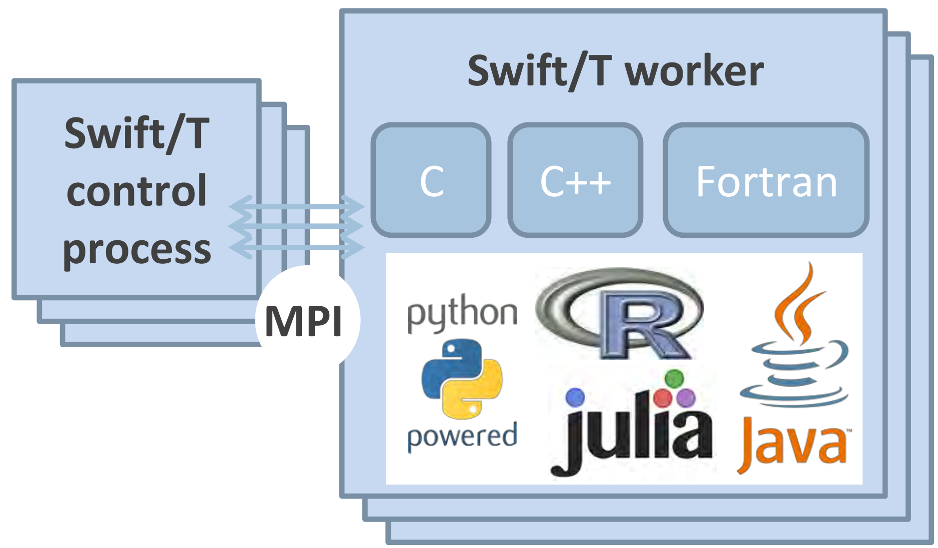
- **Swift/T: Scalable data flow programming for distributed-memory task-parallel applications .** Proc. CCGrid, 2013.

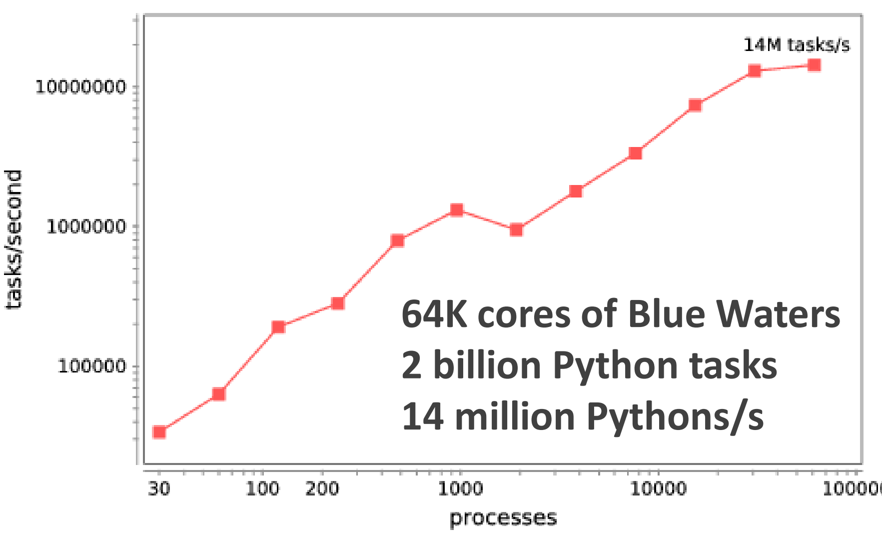
Argonne NATIONAL LABORATORY

SWIFT/T: ENABLING HIGH-PERFORMANCE SCRIPTED WORKFLOWS

Supports tasks written in many languages

- Write site-independent scripts
- Automatic parallelization and data movement
- Run native code, script fragments as applications
- Rapidly subdivide large partitions for MPI jobs
- Move work to data locations

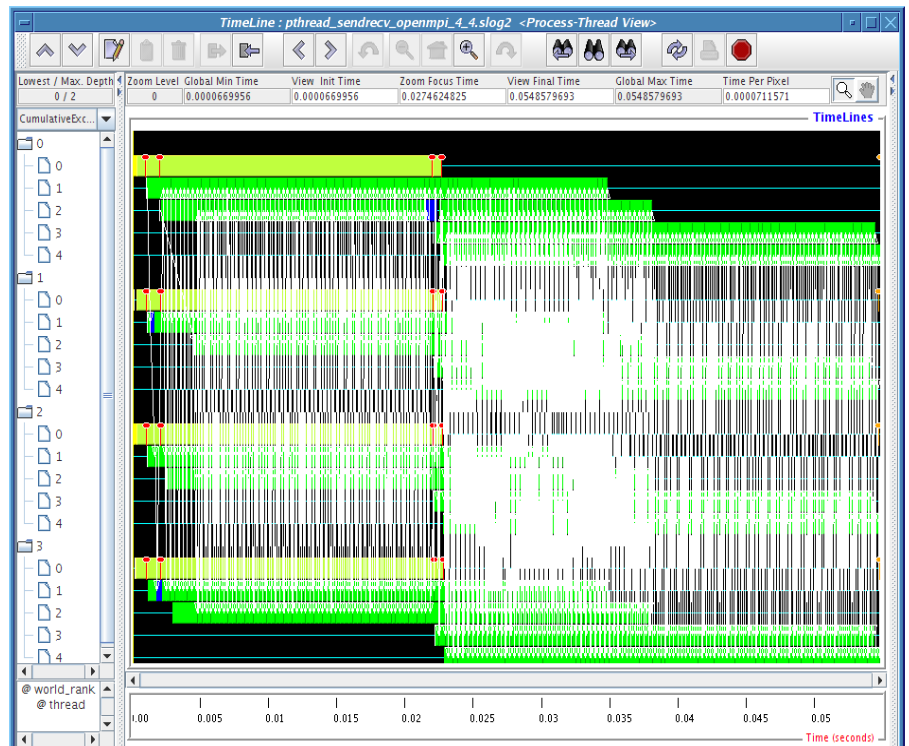




- **Interlanguage parallel scripting for distributed-memory scientific computing.** Proc. WORKS @ SC 2015

MPI: THE MESSAGE PASSING INTERFACE

- Programming model used on large supercomputers
- Can run on many networks, including sockets, or shared memory
- Standard API for C and Fortran; other languages have working implementations
- Contains communication calls for
  - Point-to-point (send/recv)
  - Collectives (broadcast, reduce, etc.)
- Interesting concepts
  - Communicators: collections of communicating processing and a context
  - Data types: Language-independent data marshaling scheme

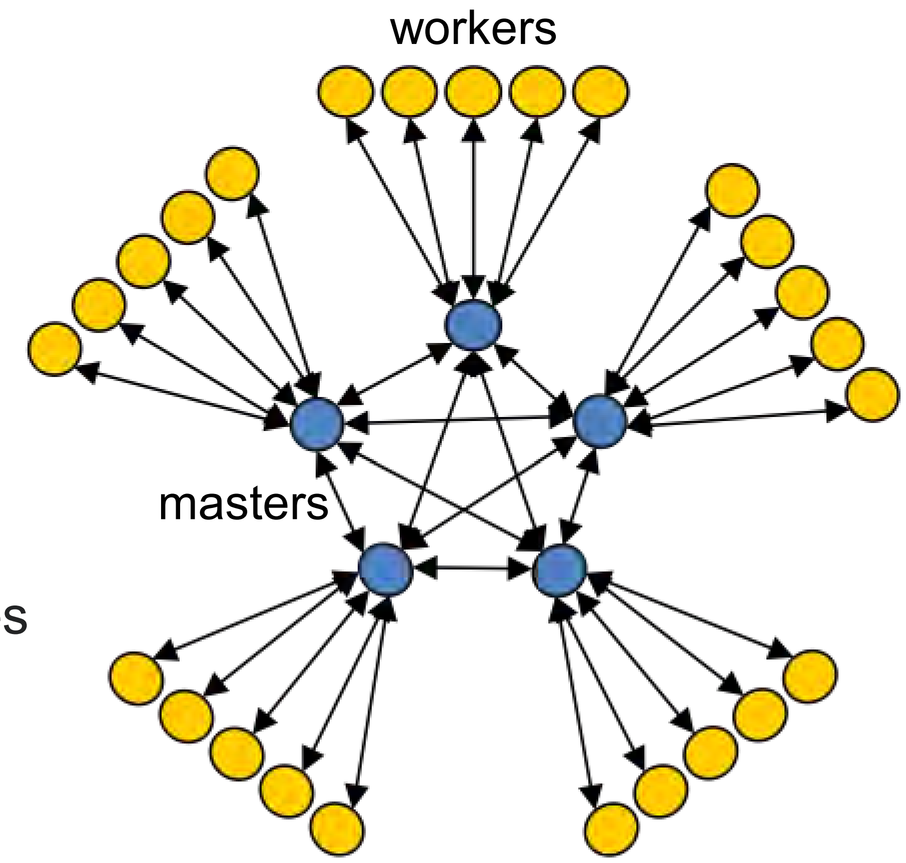


ASYNCHRONOUS DYNAMIC LOAD BALANCER

ADLB for short

- An MPI library for master-worker workloads in C
- Uses a variable-size, scalable network of servers
- Servers implement work-stealing
- The work unit is a byte array
- Optional work priorities, targets, types

- For Swift/T, we added:
  - Server-stored data
  - Data-dependent execution

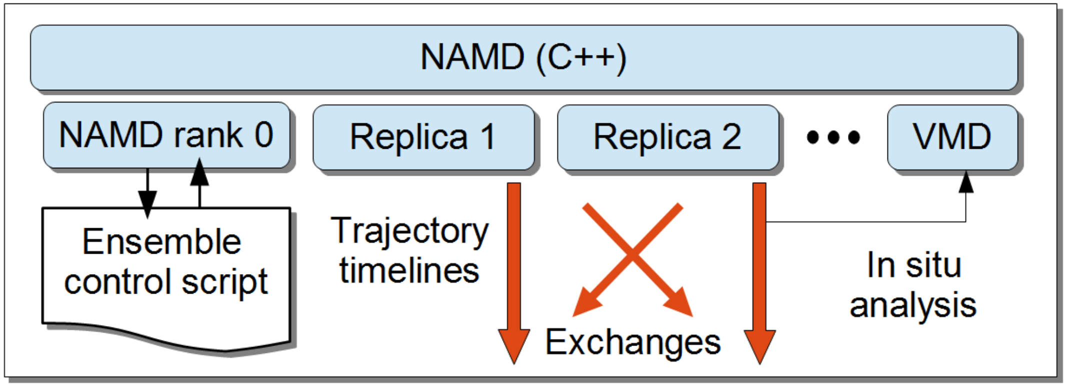


- Lusk et al. **More scalability, less pain: A simple programming model and its implementation for extreme computing.** SciDAC Review 17, 2010

APPLICATION CASE STUDIES

NAMD REPLICA EXCHANGE LIMITATIONS

- One-to-one replicas to Charm++ partitions:
  - Available hardware must match science
  - Batch job size must match science
  - Replica count fixed at job startup
  - No hiding of inter-replica communication latency
  - No hiding of replica performance divergence
- Can a different programming model help?



Swift job

SWIFT INTEGRATION INTO NAMD AND VMD

http://www.ks.uiuc.edu/Research/swift

www.ks.uiuc.edu/Research/swift/

Apps | git-svn | RTM | BigBoard | Maps | Wikipedia | Swift, Inc | ProxyIt! | Other Bookmarks

NIH CENTER FOR MACROMOLECULAR MODELING & BIOINFORMATICS | UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Type Keywords SEARCH

THEORETICAL and COMPUTATIONAL BIOPHYSICS GROUP

Home | Research | Publications | Software | Instruction | News | Galleries | Facilities | About Us

- Home
- Overview
- Publications
- Research
  - Membrane Biology
  - Mechanobiology
  - Nanoengineering
  - Bioenergetics
  - SMD/IMD
  - Quantum Biology
  - Neurobiology
  - X-ray/Cryo-EM Modeling with MDFF
  - Driving Biomedical Projects
  - Collaborations
  - Other Topics
- Software
- Outreach

Integrating NAMD and VMD with Swift/T

NAMD and VMD have recently been successfully coupled to the **Swift/T high performance parallel scripting language** developed as part of the **ExM project**, a collaboration led by Argonne National Laboratory with University of Chicago and University of British Columbia, as a part of the **Department of Energy ASCR X-Stack** program. Swift/T is now supported as part of the **Swift project** under the **NSF SI2 program**. Standard NAMD 2.10 and VMD 1.9.2 binaries can be launched across the nodes of a parallel computer and efficiently execute Swift/T dataflow programs with functions implemented in the embedded Tcl scripting language. The NAMD and VMD user communities are already familiar with Tcl, and Tcl allows access to the two programs' complete functionality. The NAMD integration with Swift/T has been used to demonstrate n:m multiplexing of n replicas across a smaller arbitrary number m of NAMD processes, a very complex capability to implement with normal NAMD scripting that can be expressed naturally in under 100 lines of Swift/T code.

All example files: **directory**, **tar archive**

VMD Swift/T Hello World

VMD and Turbine must be built with compatible Tcl libraries so that VMD can dynamically load libtclturbine.so.

- Example command: `mpiexec -n 8 vmdwrapper -e vmdswift.tcl`
- Wrapper script to run standard VMD under MPI: **vmdwrapper**
- Tcl package and Swift startup for VMD: **vmdswift.tcl**
- Swift program source code: **hello.swift**
- Swift compiler Tcl output: **hello.tcl**

NAMD Swift/T Replica Exchange

NAMD and Turbine must be built with compatible Tcl libraries so that NAMD can dynamically load libtclturbine.so.

- Example command: `mpiexec -n 8 namdwrapper namdswift.tcl apoa1.namd --run 0 --source $cwd/replica.tcl < /dev/null &`
- Wrapper script to run multicore NAMD under MPI: **namdwrapper**
- Tcl package and Swift startup for NAMD: **namdswift.tcl**
- Swift program source code: **replica.swift**
- Swift compiler Tcl output: **replica.tcl**

NAMD Swift/T MPI Tight Binding

Charm++ and NAMD must be built from source code. An MPI-based Charm++ must be used. Apply the patches below to Charm++ and NAMD, respectively, to allow Turbine to access the Charm++ inter-partition communicator. Charm++, NAMD, and Turbine must be built with compatible Tcl and MPI libraries so that NAMD can dynamically load libtclturbine.so.

- Example command: `mpiexec -n 32 Linux-x86_64-g++.mpi/namd2 namdswift.tcl apoa1.namd --run 0 --source $cwd/replica.tcl +replicas 8 +stdout /var/tmp/stdout.%d.log < /dev/null &`
- Patch for Charm++ source code: **charmswift.patch**
- Patch for NAMD source code: **namdswift.patch**

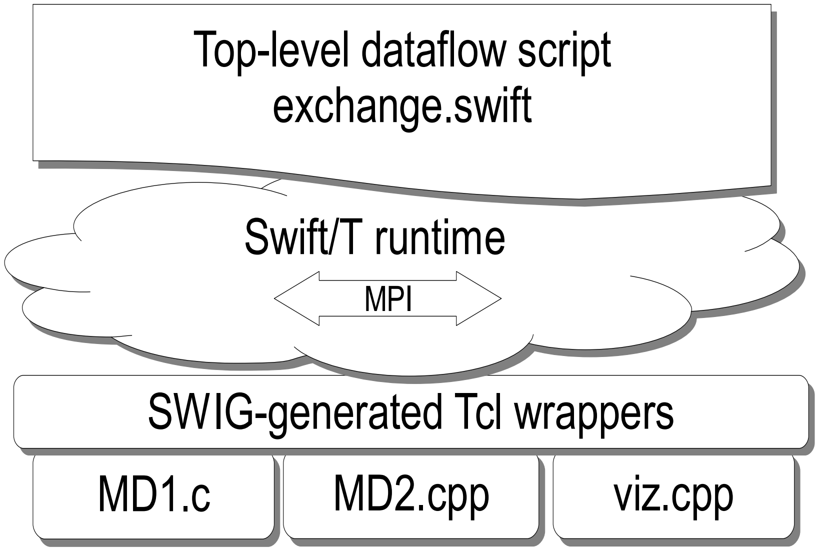
Funded by a grant from the National Institute of General Medical Sciences of the National Institutes of Health

Center for the Physics of Living Cells

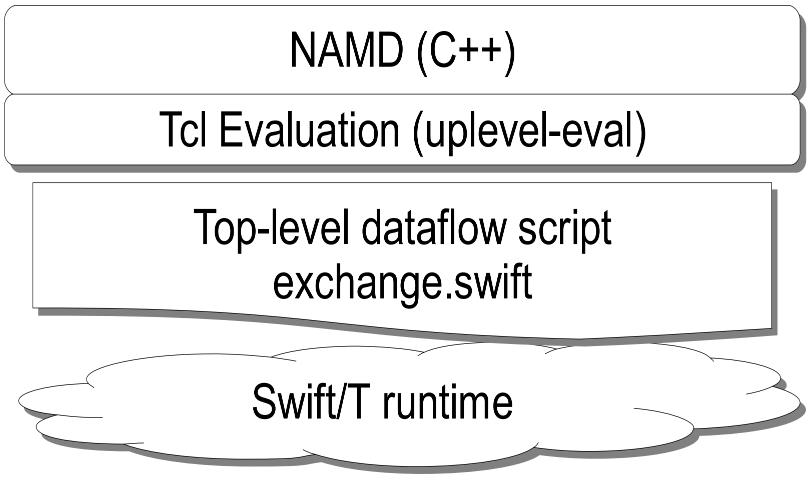
Beckman Institute for Advanced Science and Technology // National Institutes of Health // National Science Foundation // Physics, Computer Science, and Biophysics at University of Illinois at Urbana-Champaign

NAMD/VMD AND SWIFT/T

- Typical Swift/T Structure



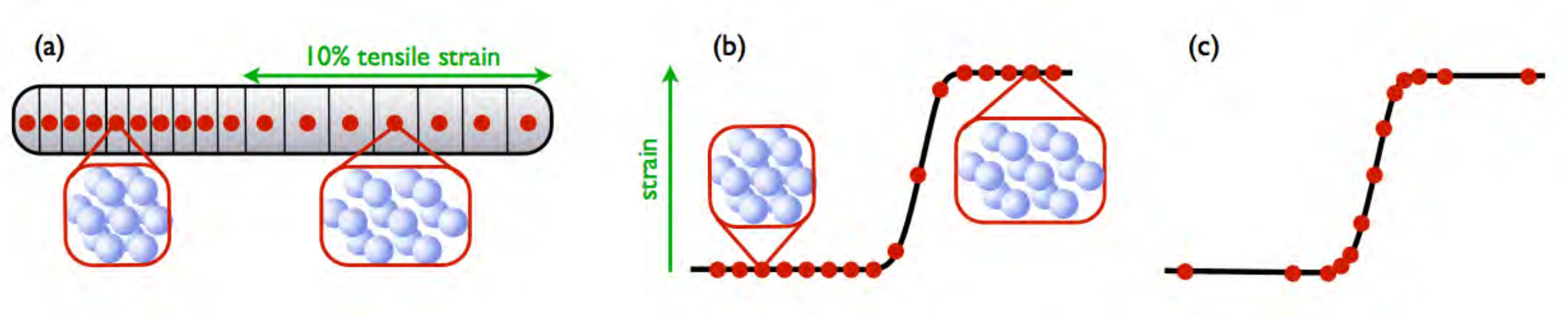
- NAMD/VMD Structure



- Phillips et al. **Petascale Tcl with NAMD, VMD, and Swift/T.** Proc. High Performance Technical Computing in Dynamic Languages @ SC, 2014.

EXMATEX: CO-DESIGN FOR MATERIALS RESEARCH

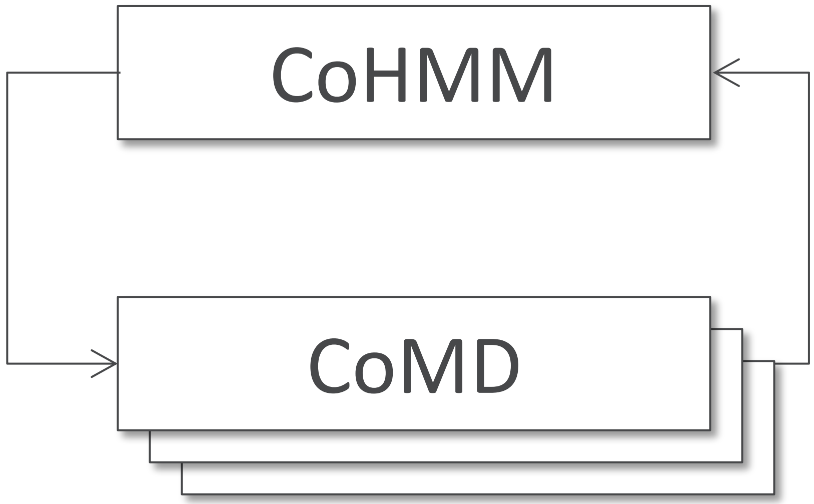
Multi-scale materials modeling



- CoHMM: Heterogeneous Multiscale Method
- CoMD: Molecular Dynamics
- Coarse-grain strain evolution using basic conservation laws
- Fine-grain molecular dynamics as necessary for physical coefficients

- **Rapid development of highly concurrent multi-scale simulators with Swift.** ExMatEx all-hands meeting 2013.
- **Swift: Parallel scripting for simulation ensembles.** ExMatEx all-hands meeting 2015.

COHMM/SWIFT

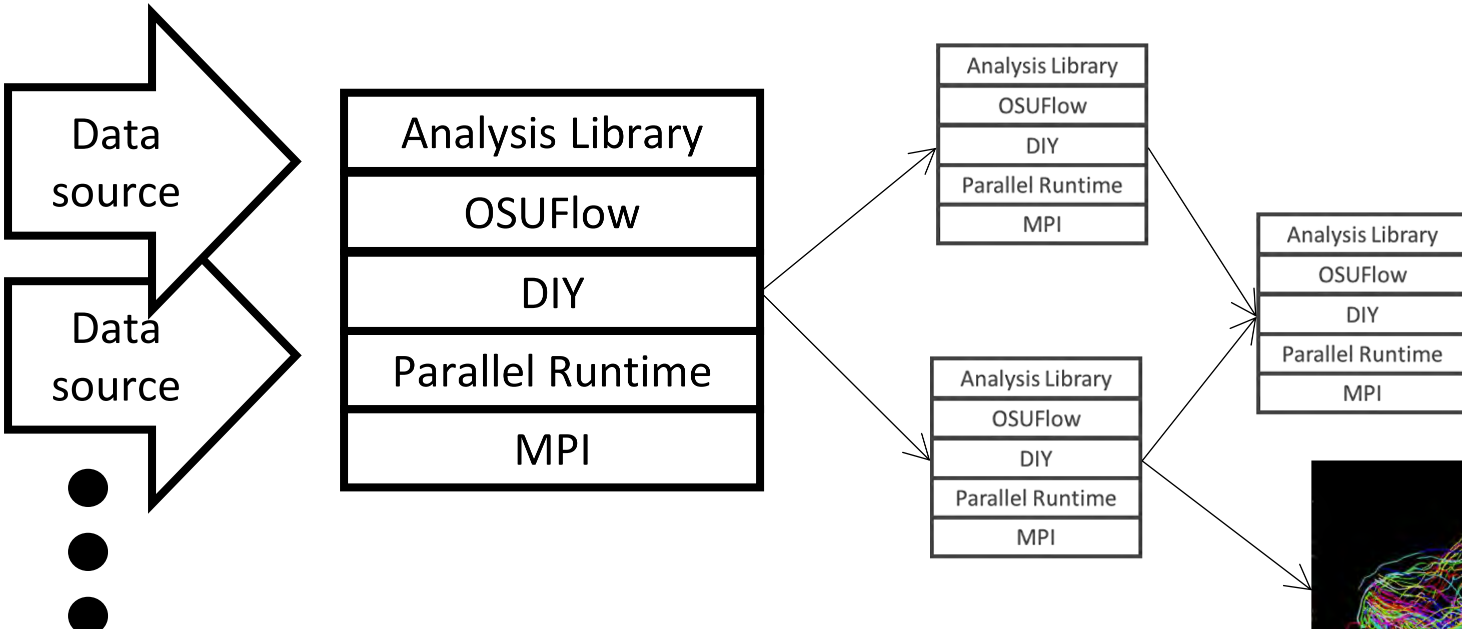


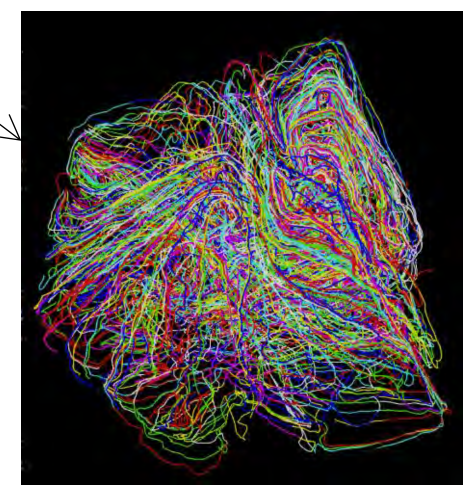
- Concurrency gained primarily by calls to CoMD

- 300 lines of sequential C
- Coordinates multiple sequential calls to CoMD
- We rewrote this in Swift

- 1000's lines of sequential C
- Simplified MD simulator
- Typically called as standalone program
- We exposed CoMD as a Swift function – no exec()

DATAFLOW+DATA-PARALLEL ANALYSIS/VISUALIZATION



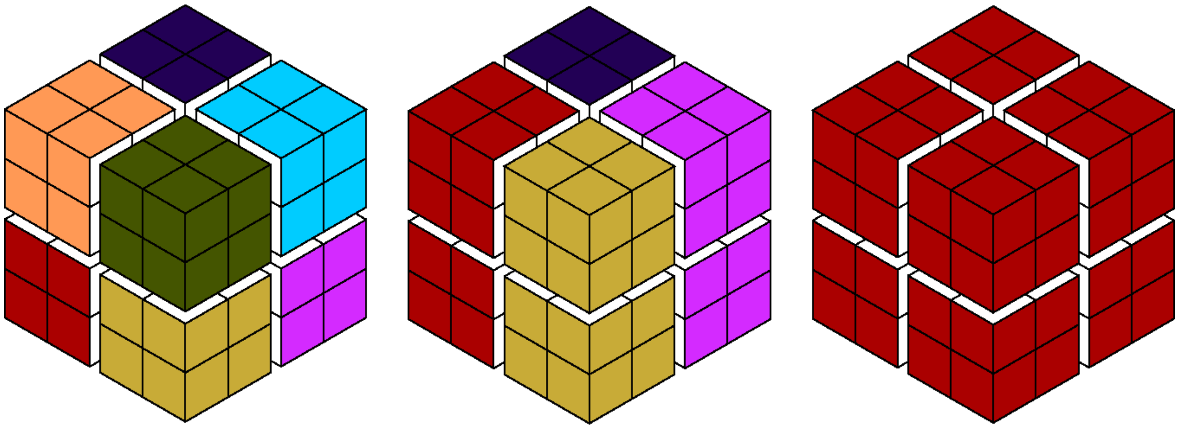


Dataflow-structured analysis framework based on OSUFlow/DIY

- **Dataflow coordination of data-parallel tasks via MPI 3.0**
  Proc. EuroMPI, 2013

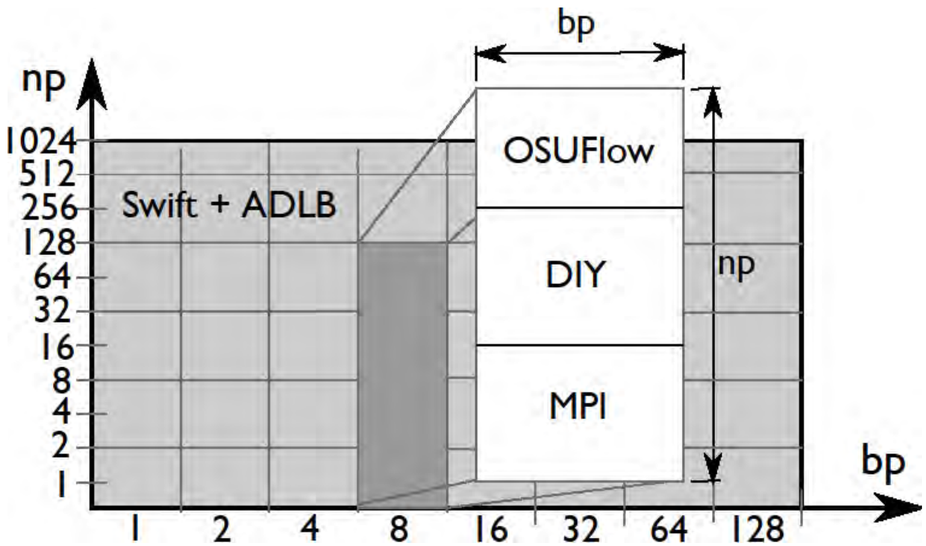
PARAMETER OPTIMIZATION FOR DATA-PARALLEL ANALYSIS

Process configurations



8 processes
1 block per process

4 processes
2 blocks per process

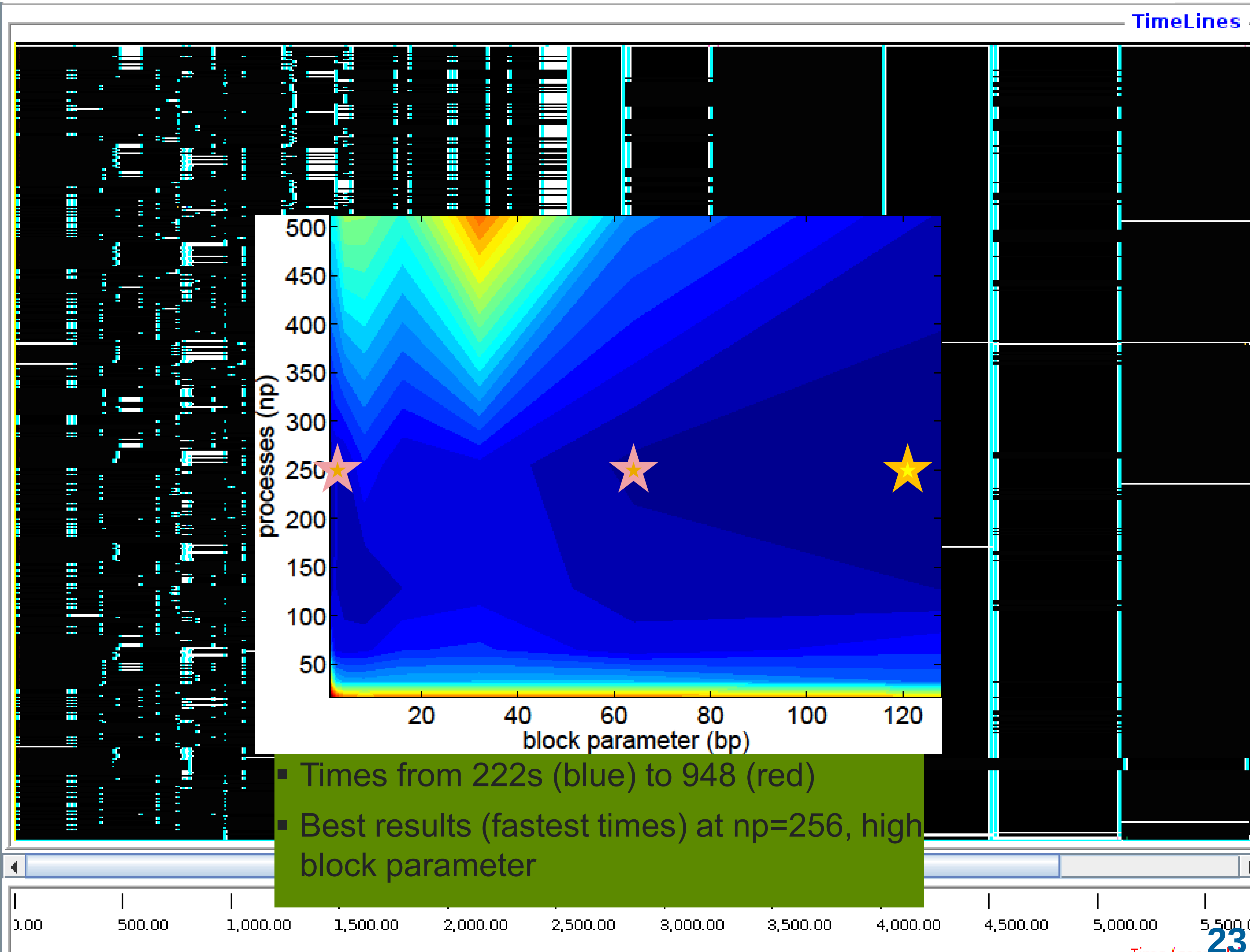


OSUFLOW APPLICATION

Complete code!

```
// Define call to OSUFlow feature MpiDraw
@par (float t) mpidraw(int bf) "mpidraw";

foreach b in [0:7] {
  // Block factor: 1-128
  bf = round(2**b);
  foreach n in [4:9] {
    // Number of processes/task: 16-512
    np = round(2**n);
    t = @par=np mpidraw(bf);
    printf("RESULT: bf=%i np=%i -> time=%0.3f",
                    bf,   np,       t);
  }
}
```



- Times from 222s (blue) to 948 (red)
- Best results (fastest times) at np=256, high block parameter

LAMMPS PARALLEL TASKS

LAMMPS provides a convenient C++ API

```
foreach i in [0:20] {
  t = 300+i;
  sed_command = sprintf("s/_TEMPERATURE_/%i/g", t);
  lammps_file_name = sprintf("input-%i.inp", t);
  lammps_args = "-i " + lammps_file_name;
  file lammps_input<lammps_file_name> =
    sed(filter, sed_command) =>
    @par=8 lammps(lammps_args);
}
```

This example can be found on GitHub:
https://github.com/b240/Workflows/tree/master/demo/LAMMPS-1
See the README.md file for more information.

CAN WE BUILD A MAKEFILE IN SWIFT?

- User wants to test a variety of compiler optimizations
- Compile set of codes under wide range of possible configurations
- Run each compiled code to obtain performance numbers
- Run this at large scale on a supercomputer (Cray XE6)

- **In Make you say:**

```
CFLAGS = ...
f.o : f.c
    gcc $(CFLAGS) f.c -o f.o
```

In Swift you say:

```
string cflags[] = ...;
f_o = gcc(f_c, cflags);
```

SWIFT FOR REALLY PARALLEL BUILDS

Plus language features- typed files, arrays, string processing

App definitions

```
app (object_file o) gcc(c_file c, string cflags[])
{
// Example:
//  gcc   -c   -O2    -o  f.o f.c
   "gcc" "-c" cflags "-o" o   c;
}

app (x_file x) ld(object_file o[], string ldflags[])
{
// Example:
//  gcc           -o  f.x f1.o f2.o ...
   "gcc" ldflags "-o" x   o;
}

app (output_file o) run(x_file x)
{
  "sh" "-c" x @stdout=o;
}

app (timing_file t) extract(output_file o)
{
  "tail" "-1" o "|" "cut" "-f" "2" "-d" " " @stdout=t;
}
```

Swift code

```
string program_name = "programs/program1.c";
c_file c = input(program_name);

foreach O_level in [0:3]
{
  // Construct the compiler flags
  string O_flag = sprintf("-O%i", O_level);
  string cflags[] = [ "-fPIC", O_flag ];

  object_file o<my_object> = gcc(c, cflags);
  object_file objects[] = [ o ];
  string ldflags[] = [];
  // Link the program
  x_file x<my_executable> = ld(objects, ldflags);
  // Run the program
  output_file out<my_output> = run(x);
  // Extract the run time from the program output
  timing_file t<my_time> = extract(out);
}
```

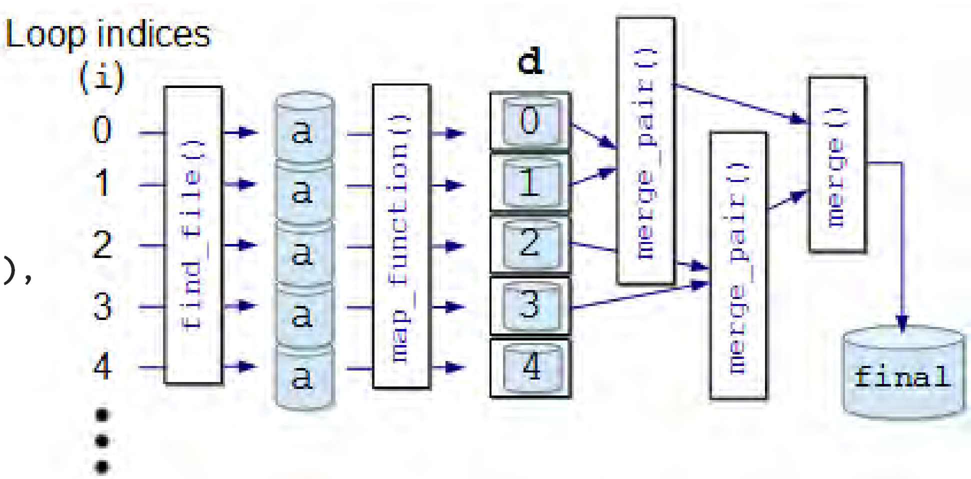
SPECIAL FEATURES FOR SCALABILITY

MAPREDUCE IN SWIFT/T

```
file d[];
int N = string2int(argv("N"));
// Map phase
foreach i in [0:N-1] {
  file a = find_file(i);
  d[i] = map_function(a);
}
// Reduce phase
file final <"final.data"> =
                     merge(d, 0, tasks-1);

(file o) merge(file d[], int start, int stop)
{
  if (stop-start == 1) {
    // Base case: merge pair
    o = merge_pair(d[start], d[stop]);
  } else {
    // Merge pair of recursive calls
    n = stop-start;
    s = n % 2;
    o = merge_pair(merge(d, start,     start+s),
                   merge(d, start+s+1, stop));
  }
}
```

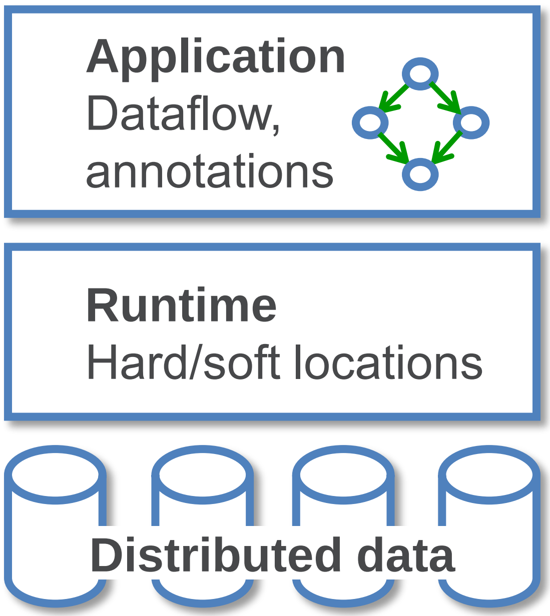
- The user needs to implement `map_function()` and `merge()`
- These may be implemented in native code, Python, etc.
- Could add **annotations**
- Could add additional custom **application logic**



- **Big data staging with MPI-IO for interactive X-ray science.** Proc. Big Data Computing, 2014

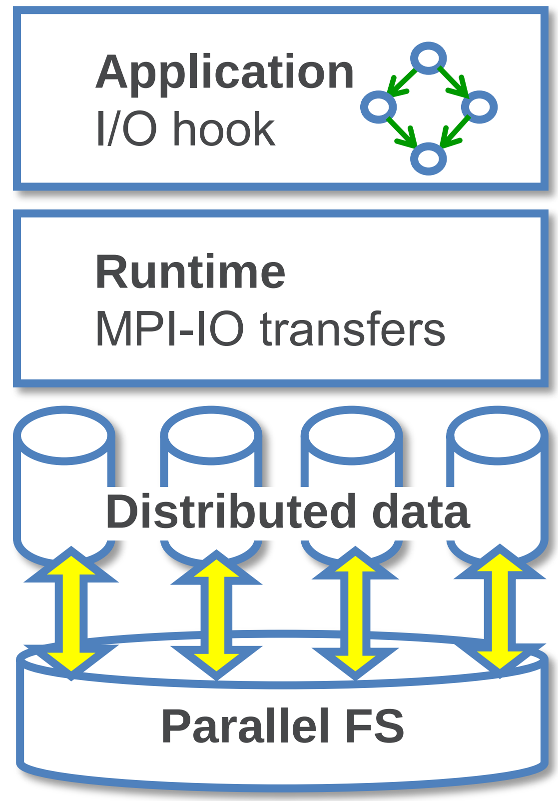
FEATURES FOR BIG DATA ANALYSIS

- **Location-aware scheduling**
  User and runtime coordinate data/task locations



- F. Duro et al**. Flexible data-aware scheduling for workflows over an in-memory object store**. Proc. CCGrid, 2016.

- **Collective I/O**
  User and runtime coordinate data/task locations



TASK LOCATIONS

- User-written annotation on function call
- Swift/T provides a **hostmap** library that maps host names to MPI ranks
- User annotation sends function to rank:

```
foreach i in 0:N-1 {
  location L = hostmap_lookup("file"+i);
  @location=L f(i);
}
```

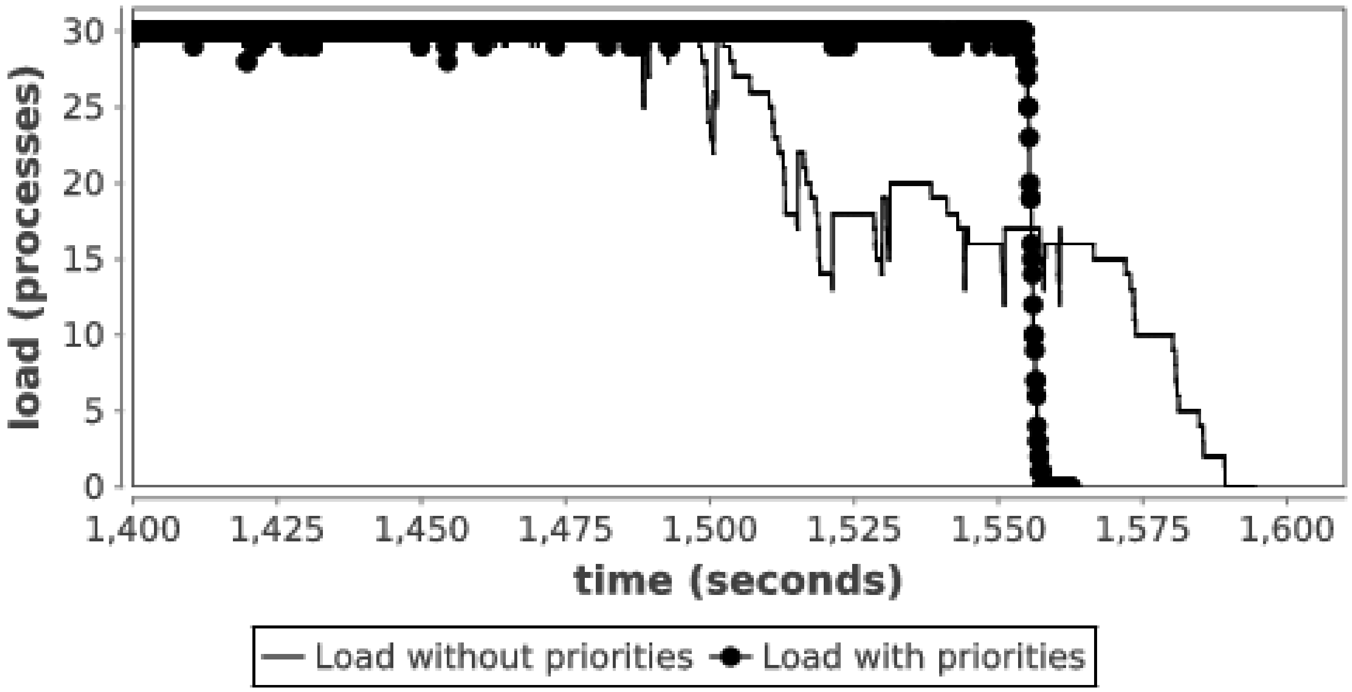
- Useful for data-intensive applications or leaf functions with state
- Soft locations: allow queued tasks to be stolen and execute anywhere

```
foreach i in 0:N-1 {
  location L = hostmap_lookup("file"+i);
  @location=(L, SOFT) f(i);
}
```

- Want to automate this in some cases with the `@heavy` syntax

SWIFT/T: PRIORITIZE LONG-RUNNING TASKS

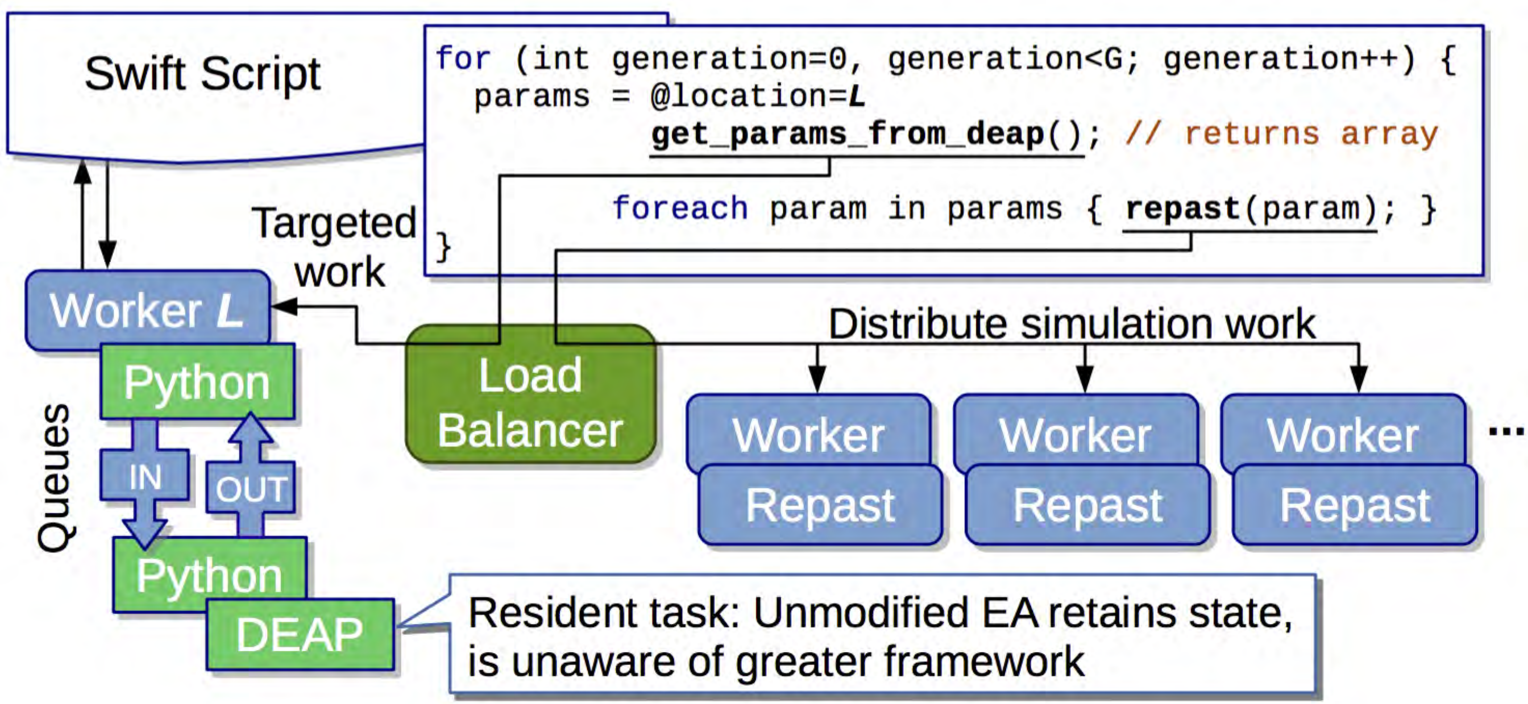
- Variable-sized tasks produce trailing tasks: addressed by exposing ADLB task priorities at language level



MODEL EXPLORATION

EMEWS: EXTREME-SCALE MODEL EXPLORATION WORKFLOWS IN SWIFT/T

- To query the state of the EA, we designate one worker on location L for exclusive use by DEAP.



- http://www.mcs.anl.gov/~emews/tutorial

QUESTIONS?

LINKS

- Swift/T Home: http://swift-lang.org/Swift-T
- Swift/T Guide: http://swift-lang.github.io/swift-t/guide.html
- Swift/T Sites Guide: http://swift-lang.github.io/swift-t/sites.html
- Swift/T GitHub: https://github.com/swift-lang/swift-t
- This tutorial: https://github.com/swift-lang/tutorial-NCSA-2017
- Support: https://groups.google.com/forum/#!forum/swift-t-user
- Parsl: https://github.com/Parsl/parsl

Argonne NATIONAL LABORATORY