Cloud Programming and Software Environments

The Swift parallel scripting language

Slides courtesy of Michael Wilde

Ioan Raicu

Computer Science Department
Illinois Institute of Technology

Computation Institute

CS554: Data-Intensive Computing

September 30th, 2013

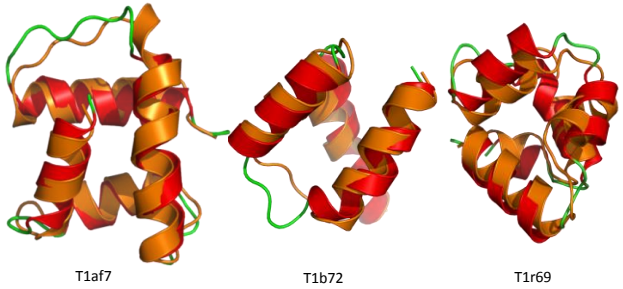
Argonne NATIONAL LABORATORY

- **Swift is a parallel scripting language** for multicores, clusters, grids, clouds, and **supercomputers**
  - *for loosely-coupled applications - application and utility programs linked by exchanging files*
  - *debug on a laptop, then run on a Cray*
- **Swift is easy to write**
  - *it's a simple high-level functional language with C-like syntax*
  - *Small Swift scripts can do large-scale work*
- **Swift is easy to run**: contains all services for running Grid workflow - in one Java application
  - *untar and run – Swift acts as a self-contained grid or cloud client*
  - ***Swift automatically runs scripts in parallel** – usually with no user input*
- **Swift is fast**: based on a powerful, efficient, scalable and flexible Java execution engine
  - *scales readily to millions of tasks*
- Swift **usage** is growing:
  - *applications in neuroscience, proteomics, molecular dynamics, biochemistry, economics, statistics, earth systems science, and more.*

When do you need Swift?

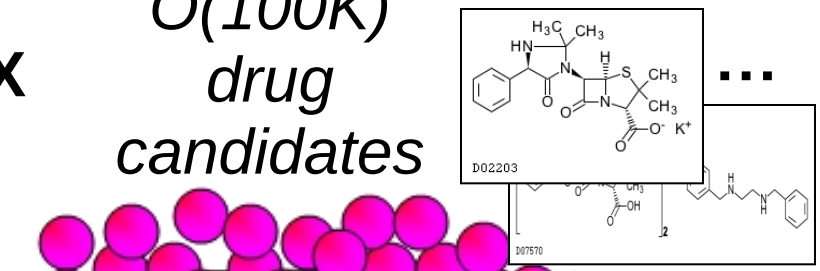
Typical application: protein-ligand docking for drug screening

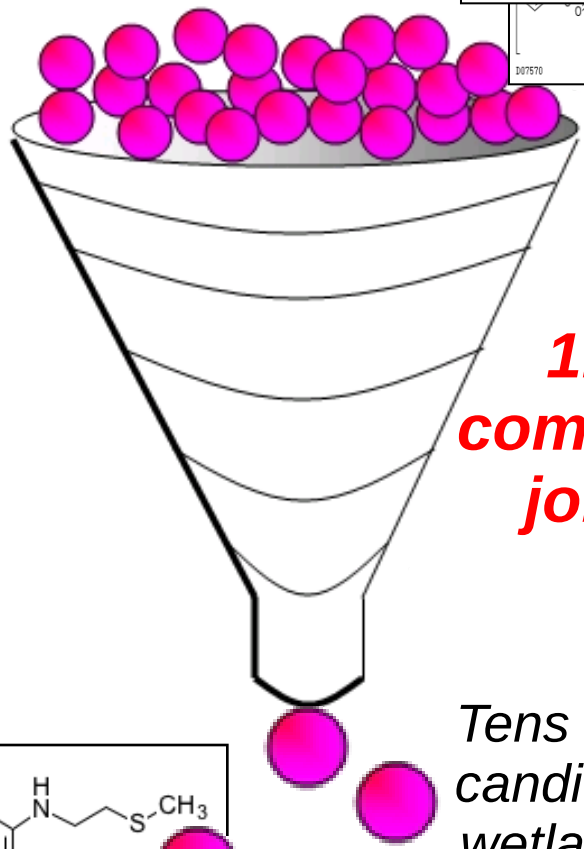
O(10) proteins implicated in a disease



X

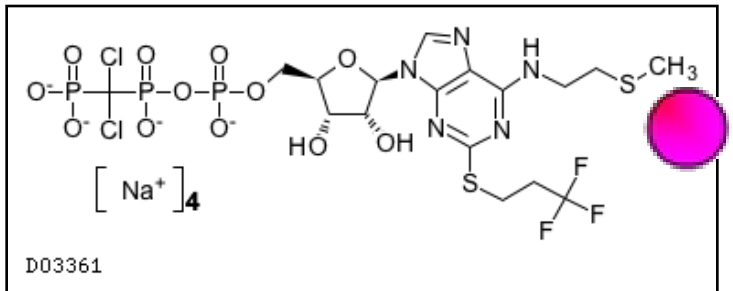
O(100K) drug candidates



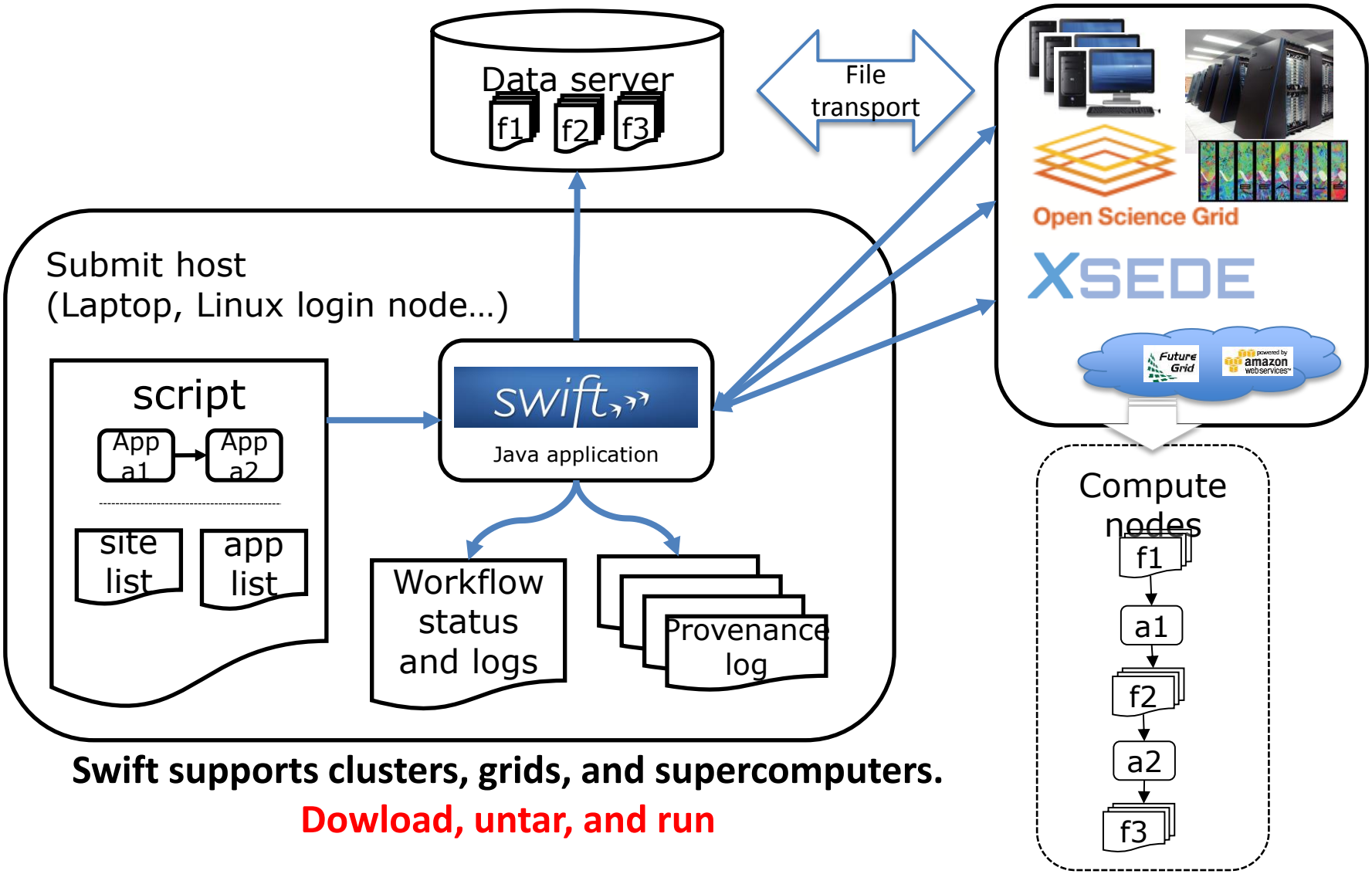


1M compute jobs

Tens of fruitful candidates for wetlab & APS



Solution: parallel scripting for high level parallellism



Swift supports clusters, grids, and supercomputers.

Dowload, untar, and run

A simple Swift script: functions run programs

```
1  type image;  // Declare a “file” type.
2
3  app (image output) rotate (image input) {
4  {
5     convert  "-rotate"  180  @input  @output ;
6  }
7
8  image oldimg <"orion.2008.0117.jpg">;
9  image newimg <"output.jpg">;
10
11 newimg = rotate(oldimg);   // runs the “convert” app
```

A simple Swift script: functions run programs

"application" wrapping function

```
1  /* declare a "file" type.
2
3  app (image output) rotate (image input) {
4  {
5     convert  "-rotate"  180  @input  @output ;
6  }
7
8  image oldimg <"orion.2008.0117.jpg">;
9  image newimg <"output.jpg">;
10
11 newimg = rotate(oldimg);   // runs the "convert" app
```

A simple Swift script: functions run programs

```
1  type image;  // De                e" type.
2
3  app (image output) rotate (image input) {
4  {
5     convert  "-rotate"  180  @input  @output ;
6  }
7
8  image oldimg <"orion.2008.0117.jpg">;
9  image newimg <"output.jpg">;
10
11 newimg = rotate(oldimg);   // runs the "convert" app
```

Output file

Input file

Actual files to use

A simple Swift script: functions run programs

```
1  type image;  // Declare a “file” type.
2
3  app (image output) rotate (image input) {
4  {
5     convert  "-rotate"  180  @input  @output ;
6  }
7
8  image oldimg <"orion.2008.0117.jp
9  image newimg <"output.jpg">;
10
11 newimg = rotate(oldimg);  // runs the “convert” app
```

Invoke the “rotate” function to run the “convert” application

Execution is driven by data flow

```
1  (int r) myproc (int i)
2  {
3      j = f(i);
4      k = g(i);
5      r = j + k;
6  }
```

7 f() and g() are computed in parallel
8 myproc() returns r when they are done

9 This parallelism is AUTOMATIC
10 Works recursively down the program's call graph

Parallelism via foreach { }

```
type image;

(image output) flip(image input) {
  app {
    convert "-rotate" "180" @input @output;
  }
}

image observations[ ]
image flipped[ ]
```

Map inputs from local directory

```
<simple_mapper; prefix="orion">;
<simple_mapper; prefix="flipped">;
```

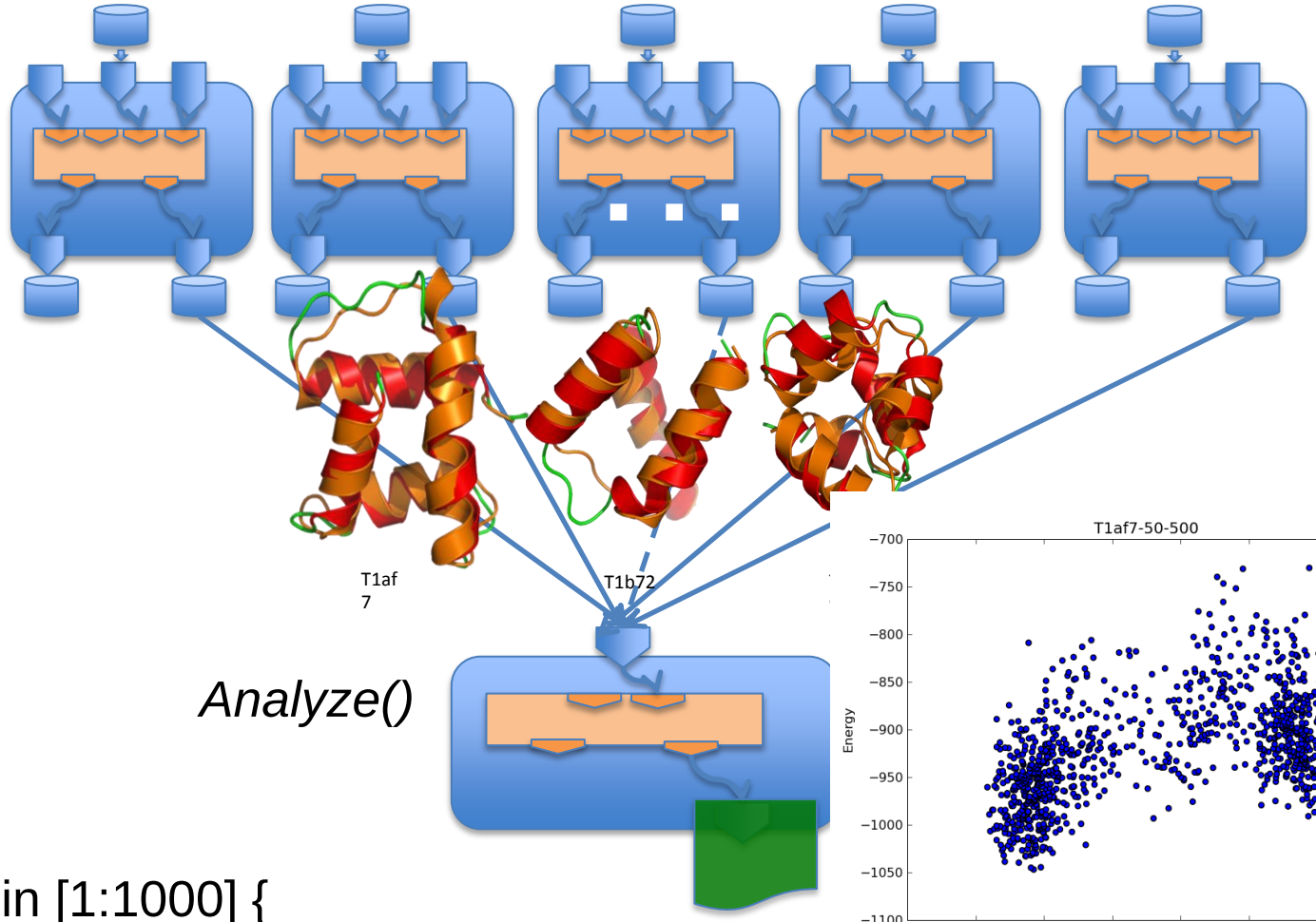
Name outputs based on index

```
foreach obs,i in observations {
  flipped[i] = flip(obs);
}
```

Process all dataset members in parallel

Large scale parallelization with simple loops

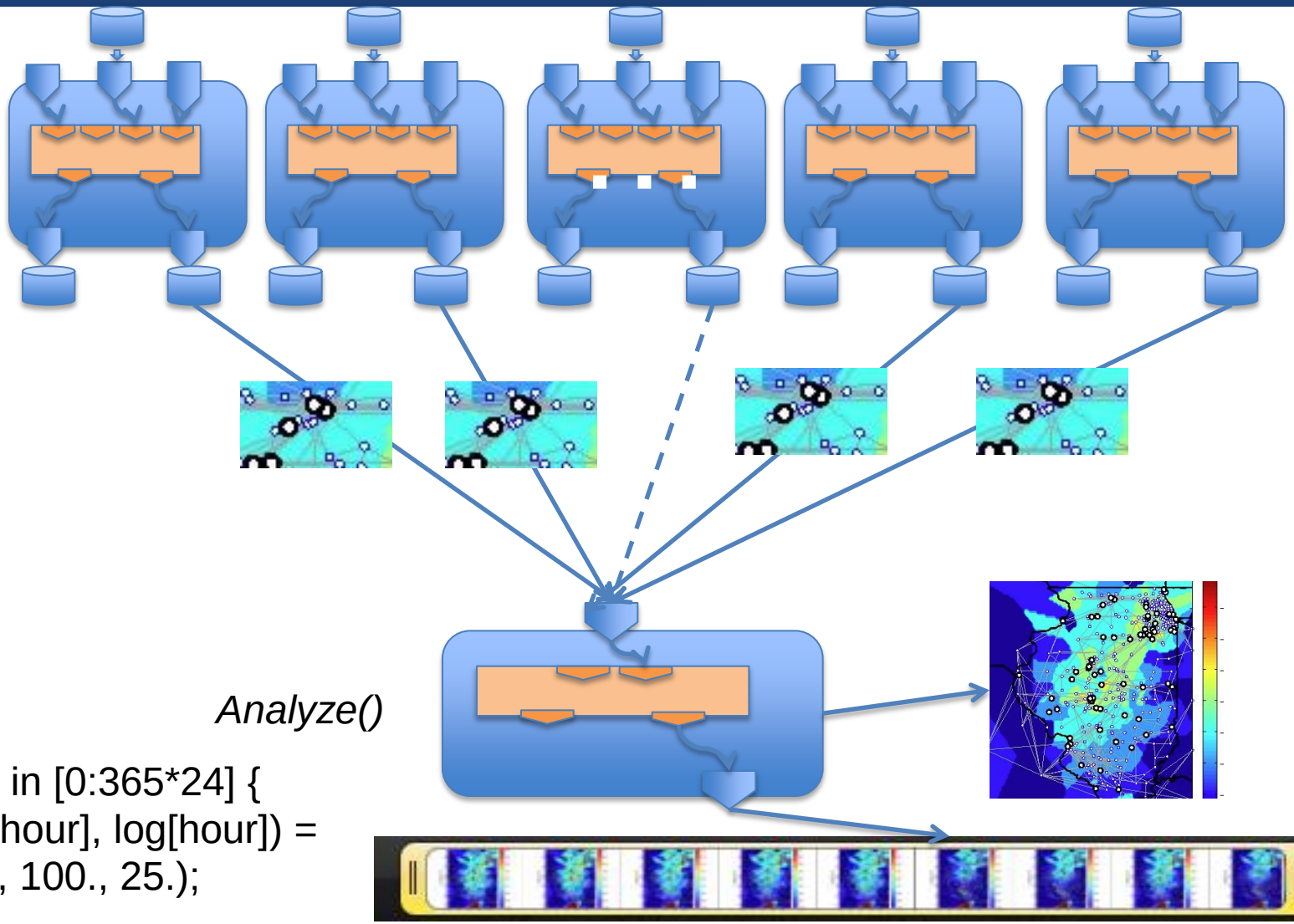
1000 Runs of the "predict" application




```
foreach sim in [1:1000] {
    (structure[sim], log[sim]) = predict(p, 100., 25.);
}
result = analyze(structure)
```

Similar patterns handle diverse app domains

8760 Runs of the "AMPL" modeling application for DOE power grid transmission simulation



Analyze()

```
foreach hour in [0:365*24] {
    (structure[hour], log[hour]) =
       model(p, 100., 25.);
}
result = analyze(structure)
```

Nested loops generate massive parallelism

```
1.  Sweep( )
2.  {
3.    int nSim = 1000;
4.    int maxRounds = 3;
5.    Protein pSet[ ] <ext; exec="Protein.map">;
6.    float startTemp[ ] = [ 100.0, 200.0 ];
7.    float delT[ ] = [ 1.0, 1.5, 2.0, 5.0, 10.0 ];
8.    foreach p, pn in pSet {
9.      foreach t in startTemp {
10.       foreach d in delT {
11.         ItFix(p, nSim, maxRounds, t, d);
12.       }
13.     }
14.   }
15. }
16.
17. Sweep();
```

10 proteins x 1000 simulations x
3 rounds x 2 temps x 5 deltas
= 300K tasks

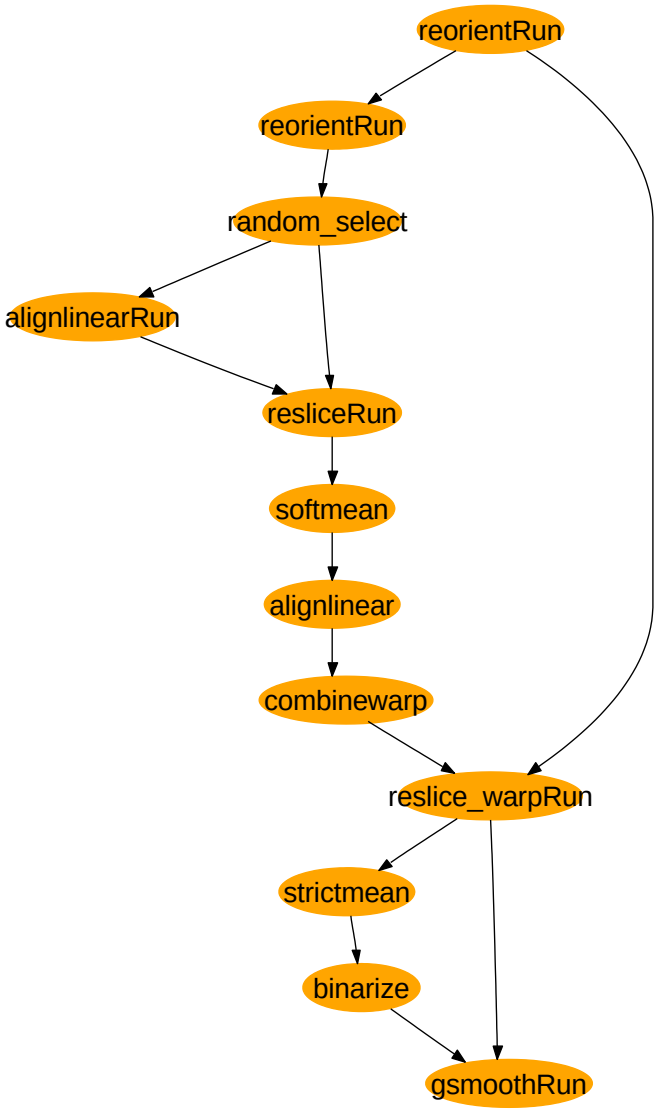
Complex parallel workflows can be easily expressed

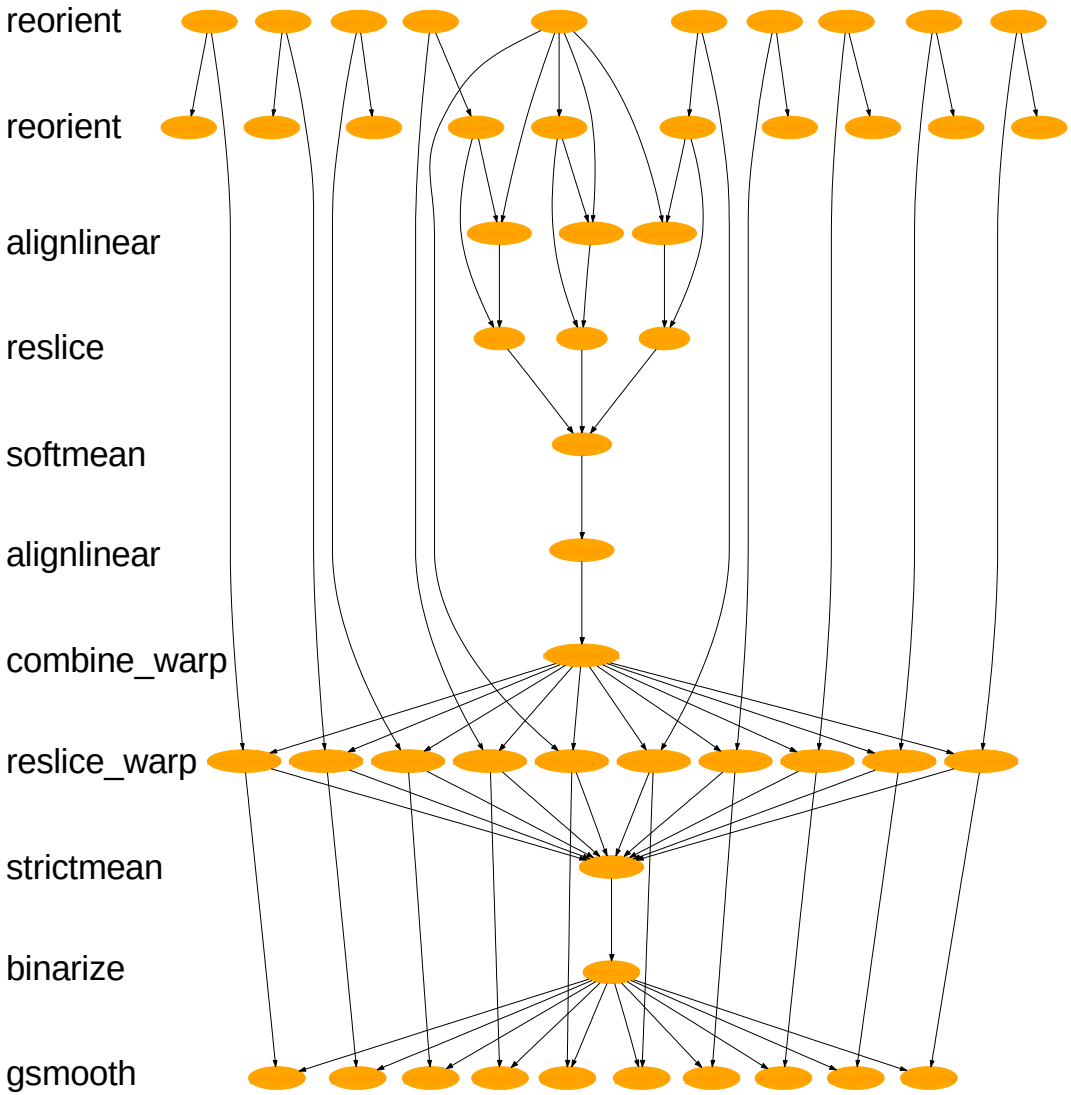
Example fMRI preprocessing script below is automatically parallelized

```
(Run snr) functional ( Run r, NormAnat a,
                       Air shrink )
{
    Run yroRun = reorientRun( r , "y" );
    Run roRun = reorientRun( yroRun , "x" );
    Volume std = roRun[0];
    Run rndr = random_select( roRun, 0.1 );
    AirVector rndAirVec = align_linearRun( rndr, std, 12, 1000, 1000, "81 3 3" );
    Run reslicedRndr = resliceRun( rndr, rndAirVec, "o", "k" );
    Volume meanRand = softmean( reslicedRndr, "y", "null" );
    Air mnQAAir = alignlinear( a.nHires, meanRand, 6, 1000, 4, "81 3 3" );
    Warp boldNormWarp = combinewarp( shrink, a.aWarp, mnQAAir );
    Run nr = reslice_warp_run( boldNormWarp, roRun );
    Volume meanAll = strictmean( nr, "y", "null" )
    Volume boldMask = binarize( meanAll, "y" );
    snr = gsmoothRun( nr, boldMask, "6 6 6" );
}
```

```
(Run or) reorientRun (Run ir,
                      string direction) {
    foreach Volume iv, i in ir.v {
        or.v[i] = reorient(iv, direction);
    }
}
```

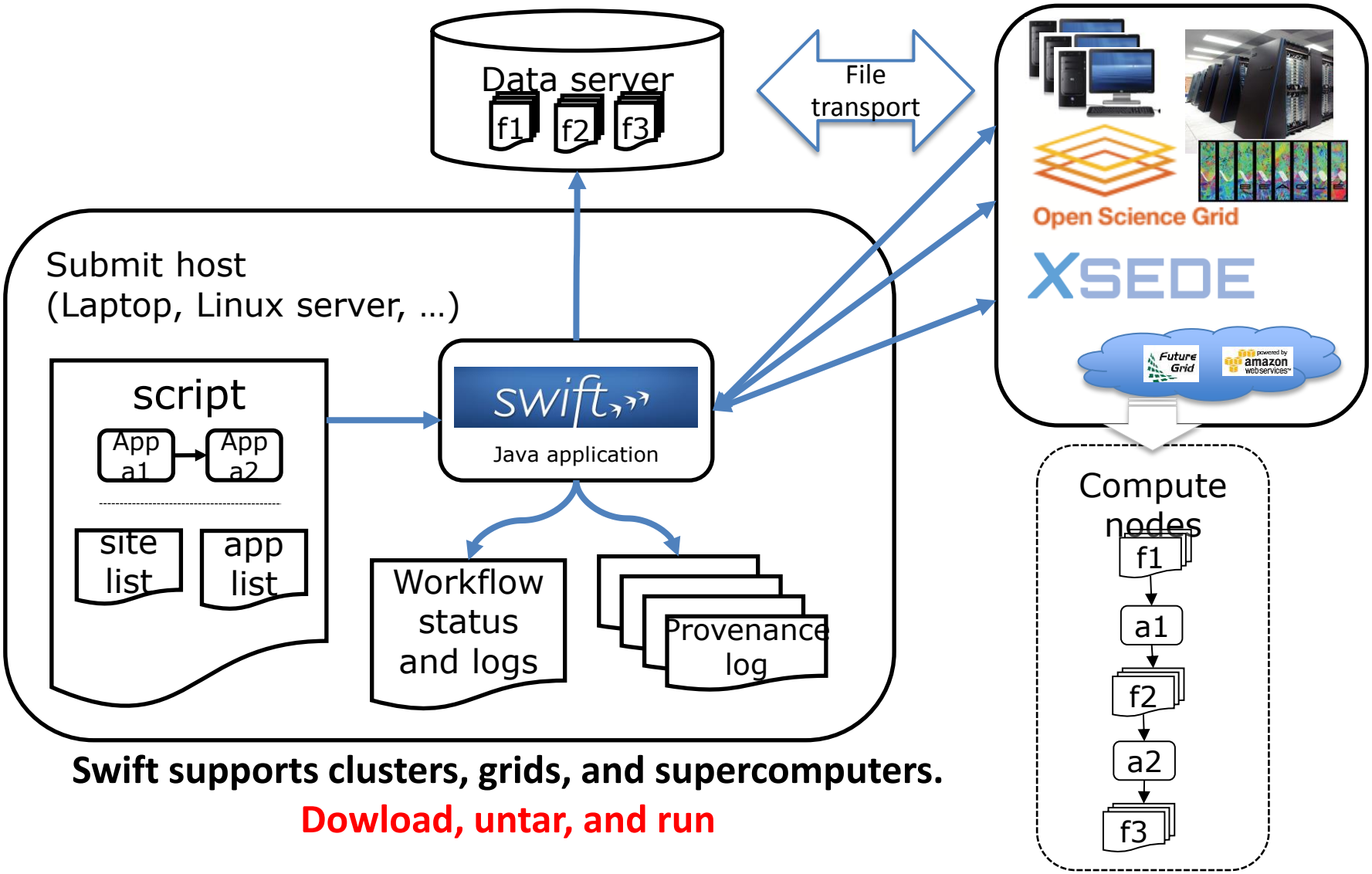
Spatial normalization of functional run





Expanded (10 volume) workflow

Running Swift scripts



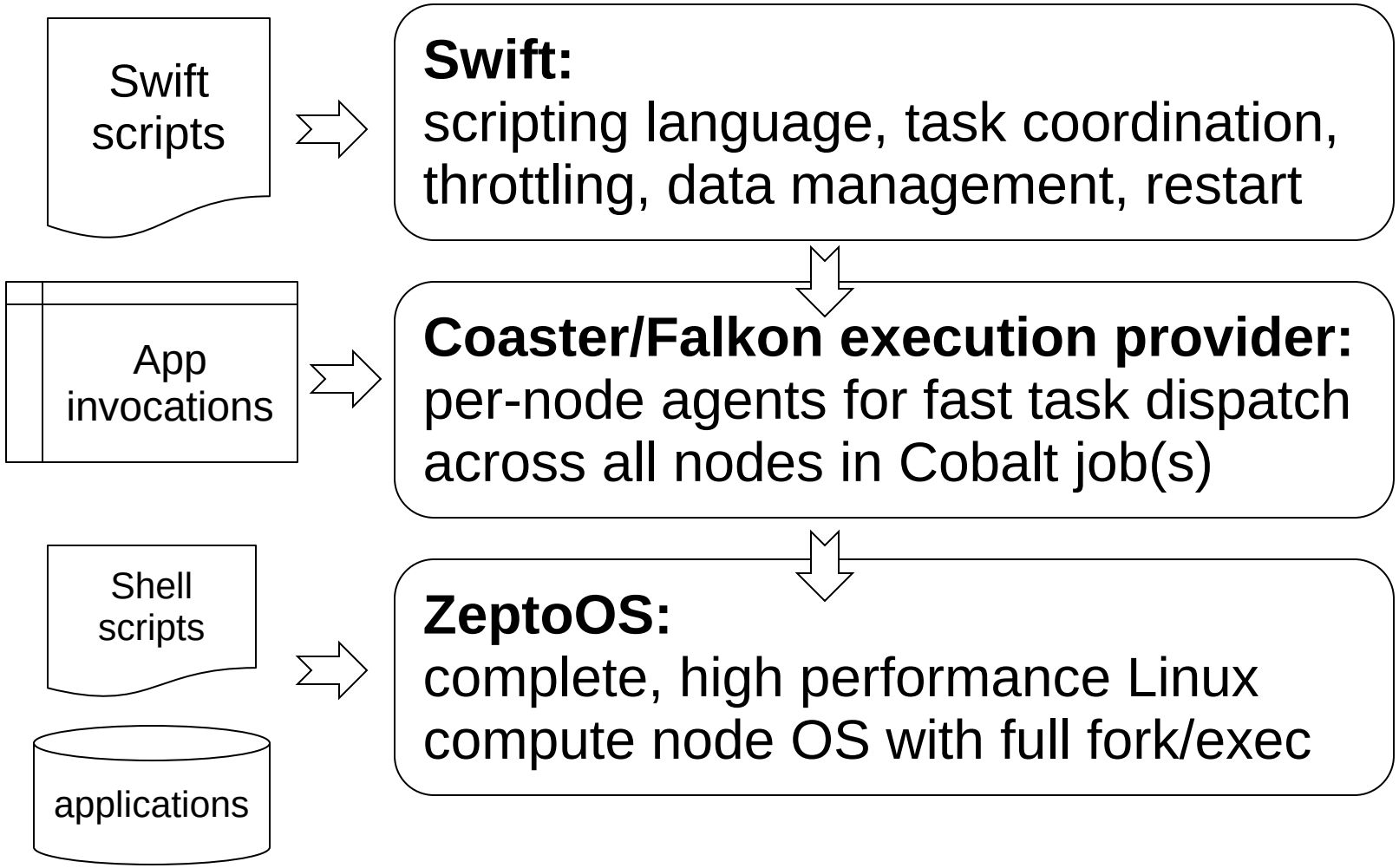
Swift supports clusters, grids, and supercomputers.

Dowload, untar, and run

Running Swift at the ALCF

- Fully contained Java grid client
- Run swift on a BG/P login host
  - Soon can run Swift from an external host
- Uses (and needs) ZeptoOS
  - Code must be compiled for ZeptoOS compatibility
  - Typically we use standard GNU toolchain
- "Jets" allows multiple socket-based MPICH2 apps to run across worker nodes
  - Trying to extend this to enable higher performance MPI stacks to run in same manner

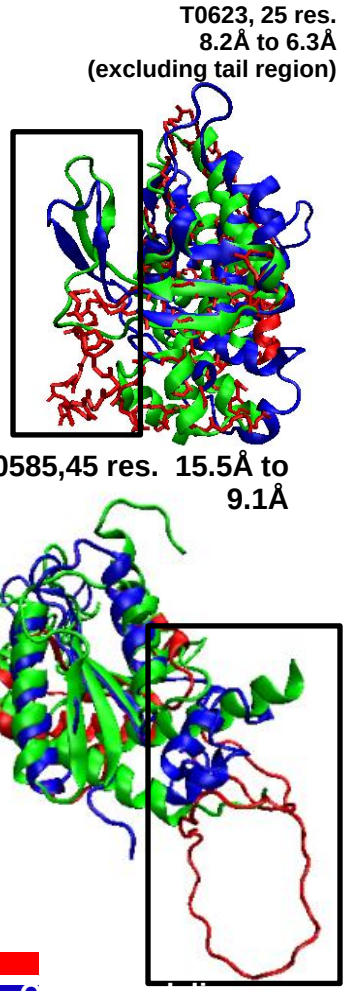
Software stack for Swift on ALCF BG/Ps

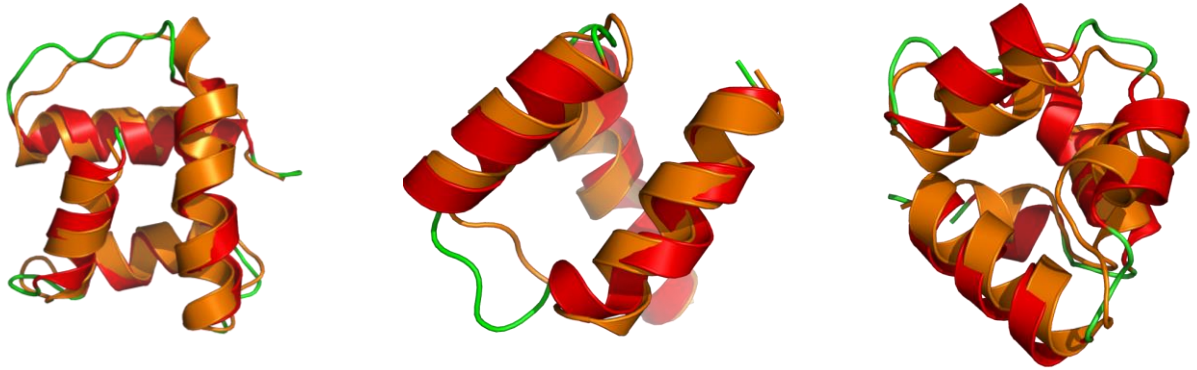


Swift app: homology-free protein structure prediction

The laboratories of Karl Freed and Tobin Sosnick use Beagle to develop and validate methods to predict protein structure using homology-free approaches.

A. Adhikari (under K. Freed and T. Sosnick) has developed new structure prediction
techniques based on Monte Carlo simulated annealing which employ novel, compact molecular representations and innovative “moves” of the protein backbone to achieve accurate prediction with far less computation then previous methods. One of the applications of the method involves rebuilding local regions in protein structures, called “loop modeling”, a problem which the group tackled with considerable success in the CASP protein folding tournament(shown in right).They are now testing further algorithmic innovations using the computational power of Beagle.

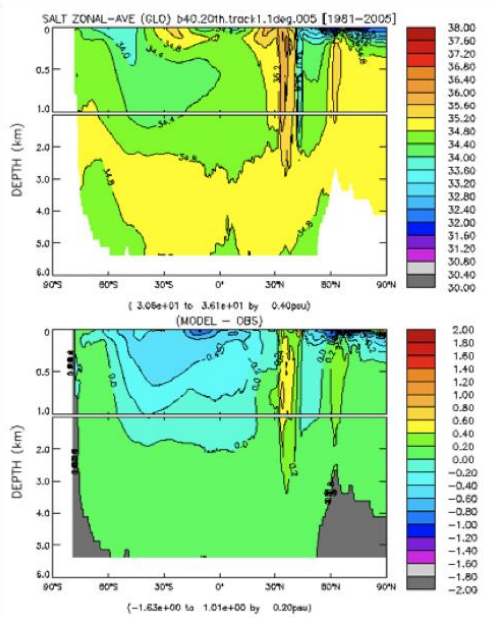


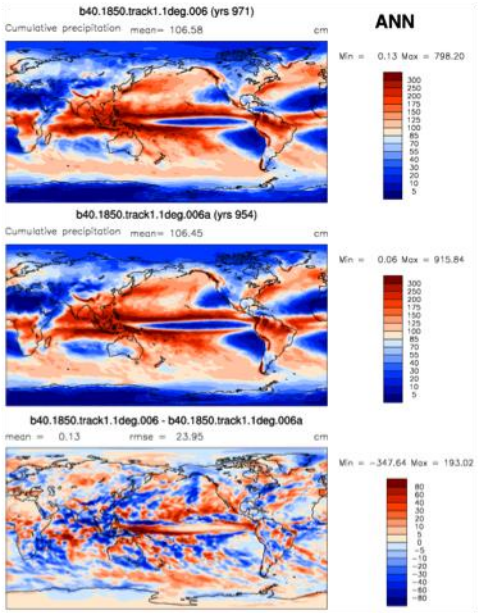


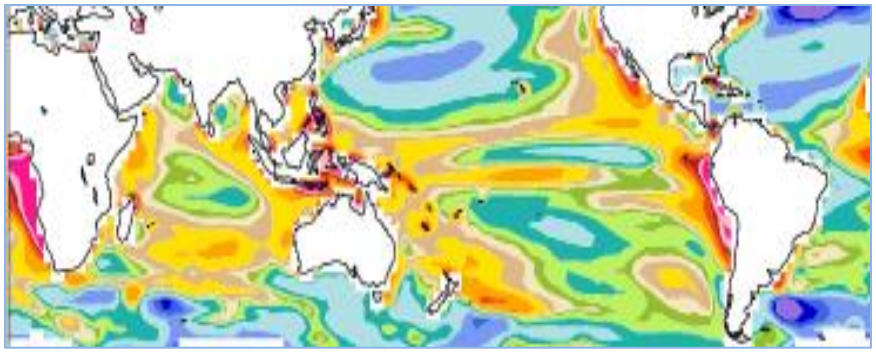
Swift app: analysis & visualization of high-res climate models

parVis

agoda







Climate models are continuing to increase both their resolution and the number of variables resulting in multi-terabyte model outputs. This large volume of data overwhelms the series of processing steps used to derive climate averages and produce visualizations. Since many of the tasks in the post-processing sequence are independent, we have applied task-parallel scripting to speed up the post-processing. We have re-written portions of the complex shell script that process output from the Community Atmosphere Model in Swift, a high-level implicitly-parallel scripting langauge that uses data dependencies to automatically parallelize a workflow. This has resulted in valuable speedups in model analysis for this heavily-used procedure.

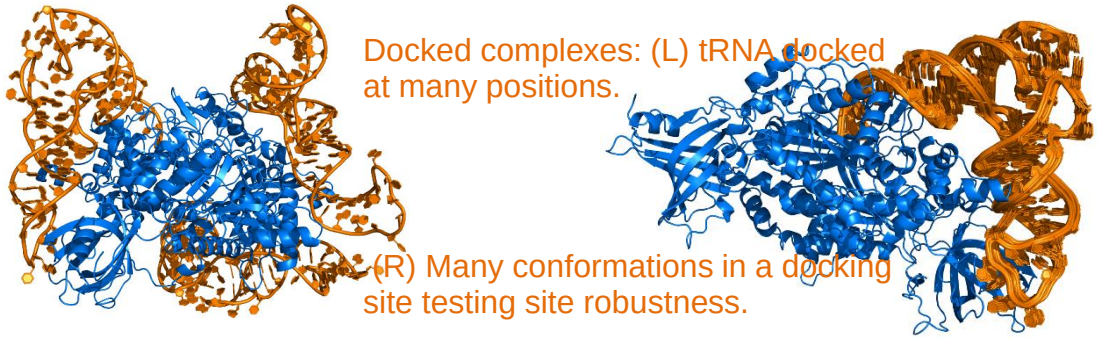
Work of: J Dennis, M Woitasek, S Mickelson, R Jacob, J Wozniak K Schuchardt

Swift app: Protein-nucleic acid interaction modeling

M. Parisien (with T. Sosnick, T. Pan, and K. Freed) used Beagle to develop a first-generation algorithm for the prediction of the RNA-protein interactome.

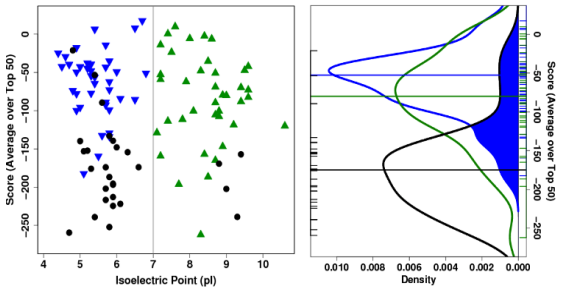
Non-coding RNAs often function in cells through specific interactions with their protein partners. Experiments alone cannot provide a complete picture of the RNA-protein interactome. To complement experimental methods, computational approaches are highly desirable. No existing method, however, can provide genome-wide predictions of docked RNA-protein complexes.
the application of computational predictions, together with experimental methods, will provide a more complete understanding on cellular networks and function of RNPs. The approach makes use of a rigid-body docking algorithm and a scoring function custom- tailored for protein-tRNA interactions. Using Swift, Beagle screened about 300 proteins per day on 80 nodes of 24 cores (11% of the total XE6's power).

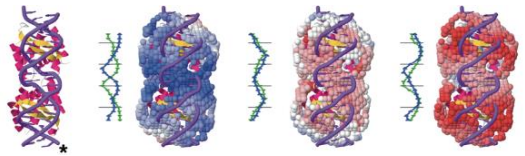
Results: the scoring function can identify the native docking conformation in large sets of decoys (100,000) for many known protein-tRNA complexes (4TRA shown here). (left) Scores for true positive complexes (•)(N=28) are compared to true negative ones of low (▾)(N=40) and high (▴) (N=40) isoelectric points. (right) Because the density curve of the true positives, which have pI < 7, has minimal overlap with the curve of the low pI true negatives (blue area), the scoring function has the specificity to identify tRNA-binding proteins. Protein-DNA interactions are being similarly studied.



Docked complexes: (L) tRNA docked at many positions.

(R) Many conformations in a docking site testing site robustness.





Systematic prediction and validation of RNA-protein interactome. Parisien M, Sosnick TR, Pan T. Poster; Kyoto, June 12-19, 2011, RNA Society. Manuscript in progress.

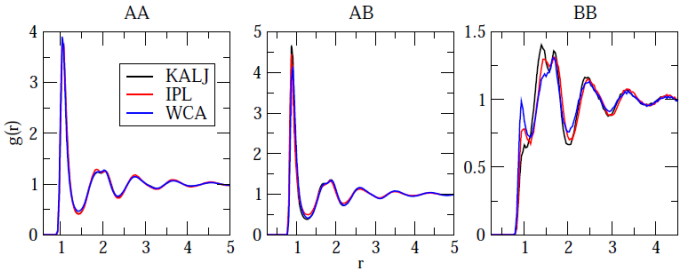
Swift app: glass structure modeling (theoretical chemistry)

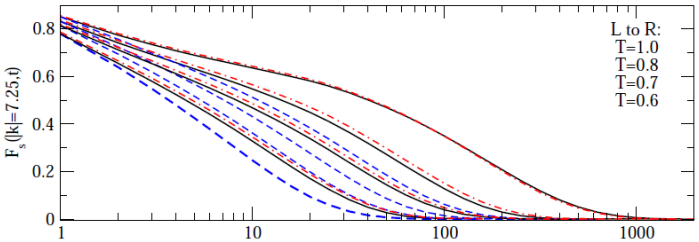
This project models of aspects of glass structure at a theoretical chemistry level. (Hocky/Reichman)

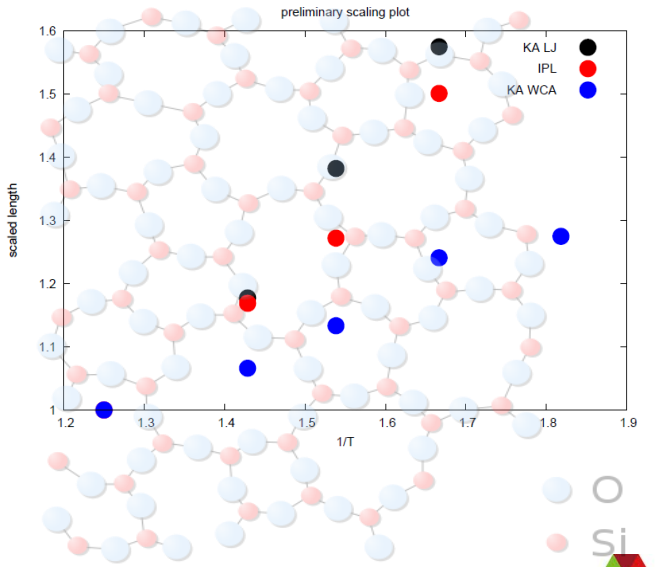
Recent studies of the glass transition in model systems have focused on calculating from theory or simulation what is known as the “mosaic length”. This project evaluated a new “cavity method” for measuring this length scale. Correlation functions are calculated at the interior of cavities of varying sizes and averaged over many independent simulations to determine a thermodynamic length. Using Swift on Beagle, Hocky investigated whether this thermodynamic length causes variations among seemingly identical systems. ~1M Beagle CPU hours were used.

Results: Three simple models of glassy behavior were studied. All appear the same (top, abc) but only two of which have particles relaxing at the same rate for the same temperature (top, d). This would imply that the glass structure does not dictate the dynamics. A new computational technique was used to extract a length scale on which the liquid is ordered in an otherwise undetectable way. Results (bottom) showed that using this length we can distinguish the two systems which have the same dynamics as separate from the third which has faster dynamics than the other two.

A manuscript is in preparation for Physical Review Letters.







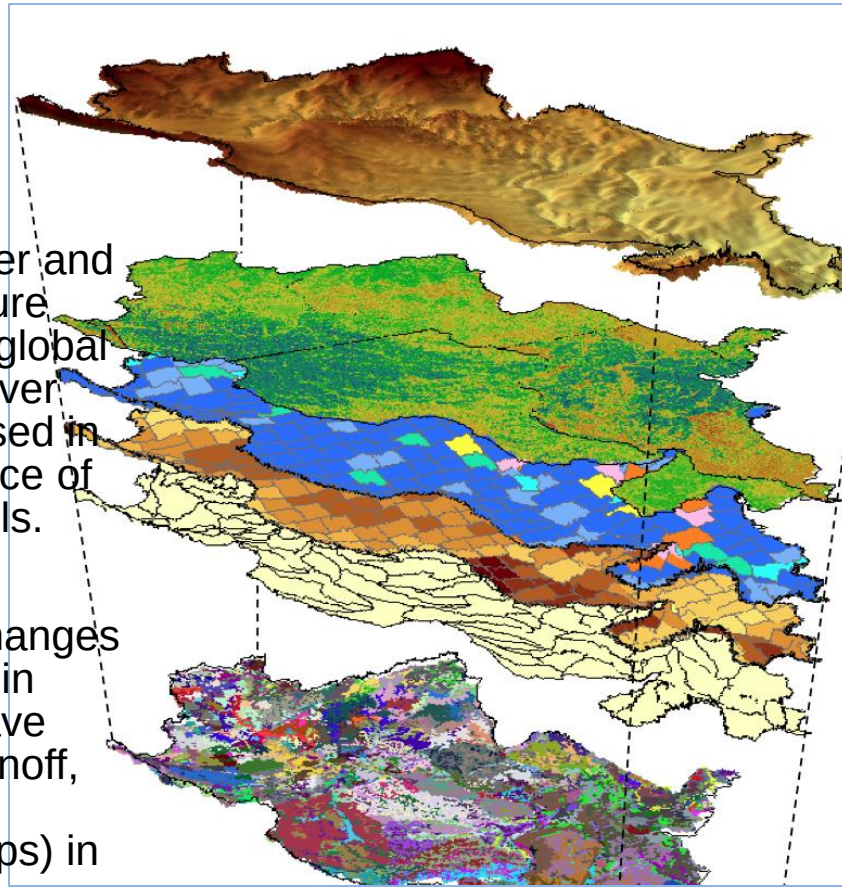
Swift app: modeling climate impact on watershed hydrology

Projecting biofuel production impact on hydrology (E. Yan, Y. Demisie)

SWAT – model – Soil and Water Assessment Tool

This project studies the impact of global temperature increase on the Upper Mississippi River Basin on water and plant productivity. It is in the process of combining future climate data obtained from a statistically downscaled global circulation model (GCM) into the Upper Mississippi River Basin model. The results from these models will be used in the proposed study to evaluate the relative performance of the proposed coupling of climate and hydrology models.

Results of this research demonstrate that plausible changes in temperature and precipitation caused by increases in atmospheric greenhouse gas concentrations could have major impacts on both the timing and magnitude of runoff, soil moisture, water quality,
water availability, and crop yield (including energy crops) in important agricultural areas.



Visualization of multiple layers of SWAT hydrology model. Courtesy E. Yan.

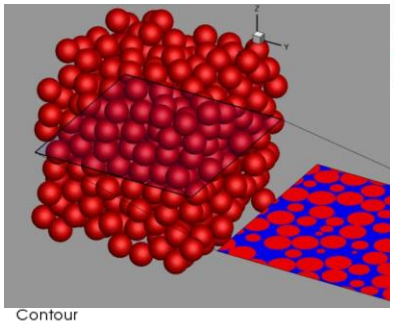
Swift app: hybrid multiscale subsurface modeling

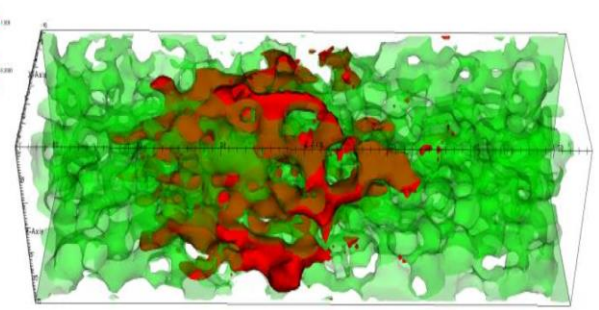
Multiscale subsurface modeling using the STOMP application

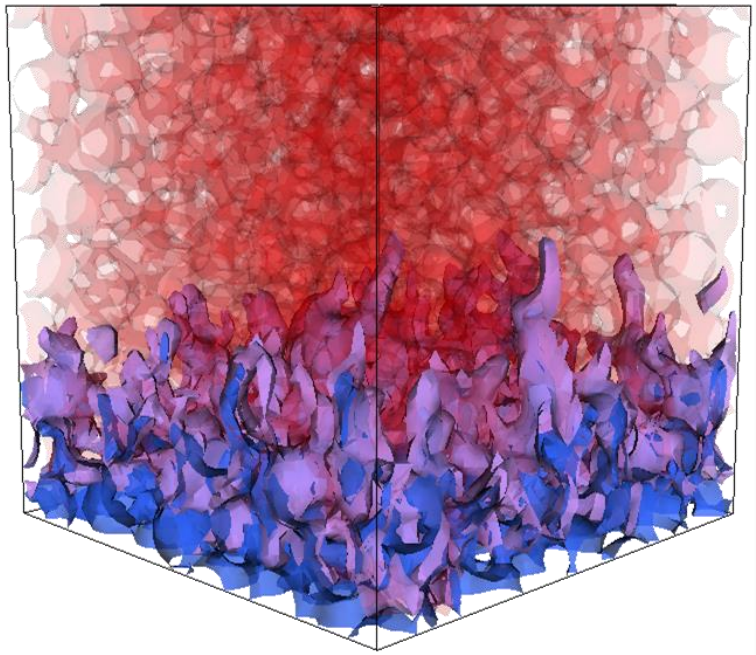
Integrates STOMP and SPH application codes

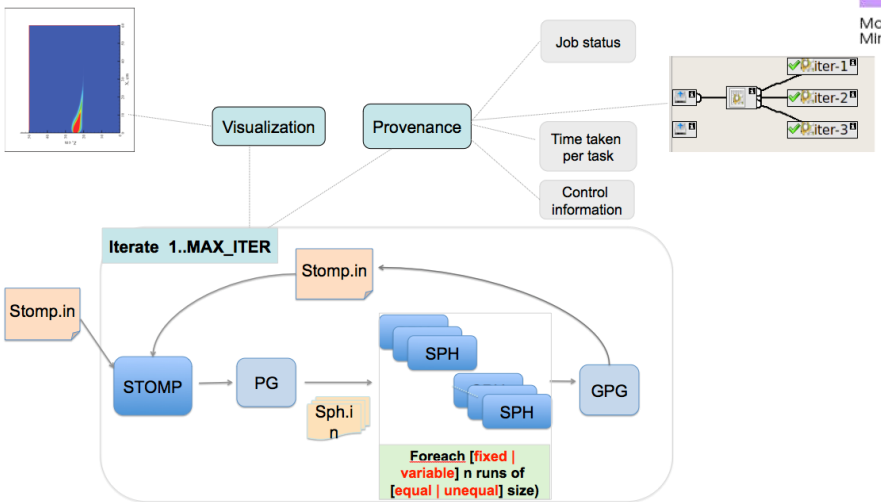
Executes MPI, serial, and viz apps form Swift on NERSC Cray resources Franklin and Hopper

Design and Implementation of Many Parallel Task Hybrid Subsurface Model, (K Agarwal, J Chase, K Schuchardt, T Scheibe, B Palmer, T Elsethagen, PNNL, **MTAGS 2011 at SC11.**









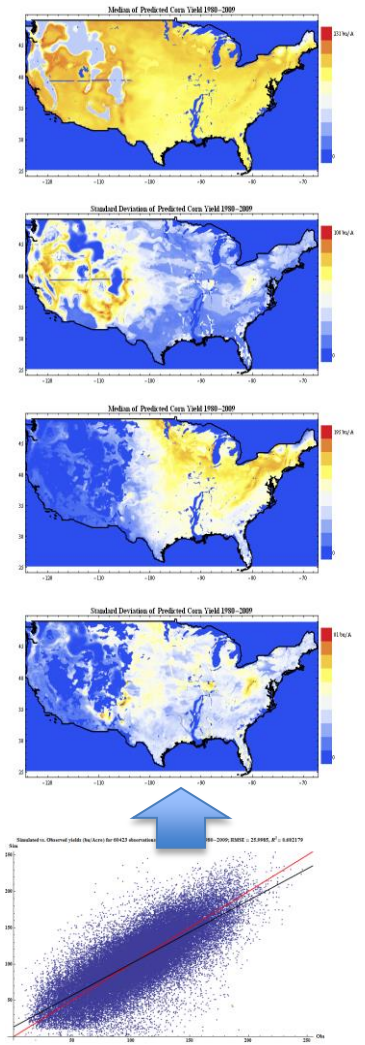
Credit: Karen Schuchardt , Bruce Palmer, Khushbu Agarwal, Tim Scheibe, PNNL

Swift app: econ/land use models for CIM-EARTH and RDCEP

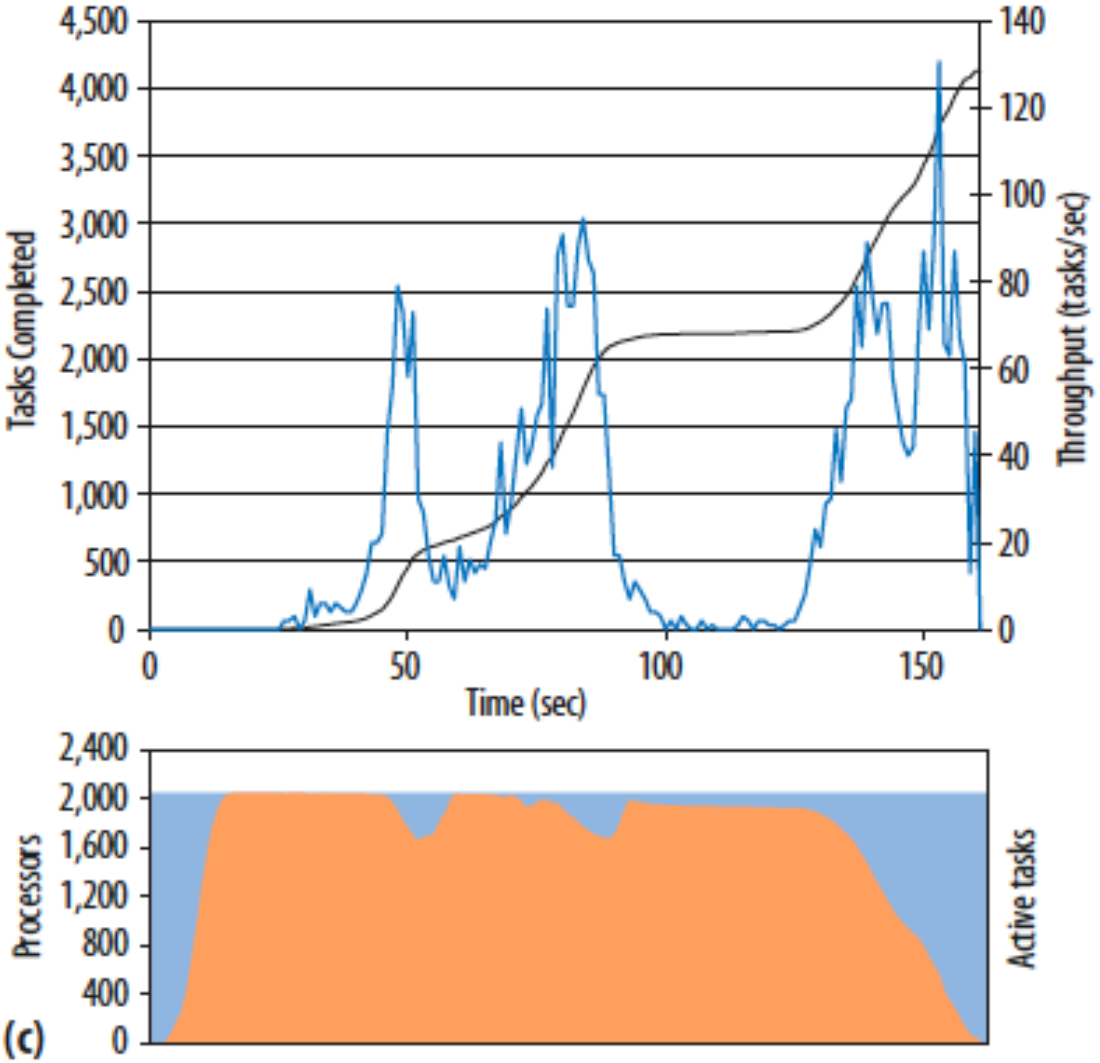
The CIM-EARTH project develops a large-scale integrated modeling framework for decision makers in climate and energy policy. (Foster, Elliott)

Beagle is being used to study land use, land cover, and the impacts of climate change on agriculture and the global food supply. Using a DSSAT 4.0 ("Decision Support System for Agrotechnology Transfer") crop systems model **ported from Windows**, a parallel simulation framework was implemented using Swift. Benchmarks of this framework have been performed on a prototype simulation campaign, measuring yield and climate impact for a single crop (maize) across the conterminous USA with daily weather data and climate model output spanning 120 years (1981-2100) and 16 different configurations of local management (fertilizer and irrigation) and cultivar choice.

Preliminary results of parallel DSSAT on Beagle have been presented in an NSF/advisory board meeting of the CIM-EARTH project. At right, top 2 maps: Preliminary results of parallel DSSAT: maize yields across the USA with intensive nitrogen application and full irrigation; bottom 2 maps show results with no irrigation. Each model run is ~120,000 DSSAT invocations.

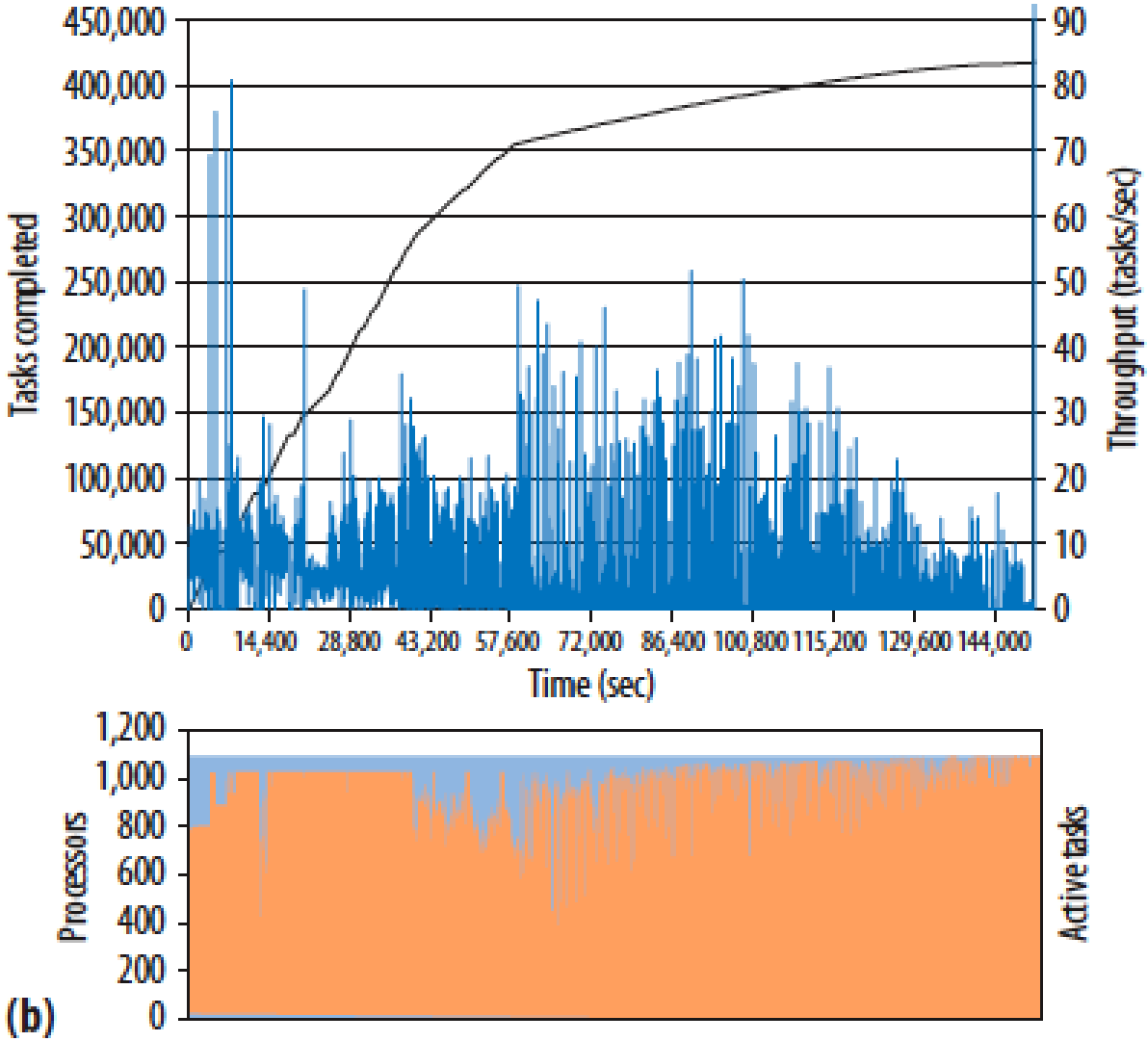


Performance: Proteomics on BG/P



4,127 PTMap jobs with Swift/Falkon on BG/P in 3 minutes

Performance: SEM for fMRI on Sun Constellation "Ranger"



Executing 418K SEM models in 41 hours running Swift with coasters on Ranger

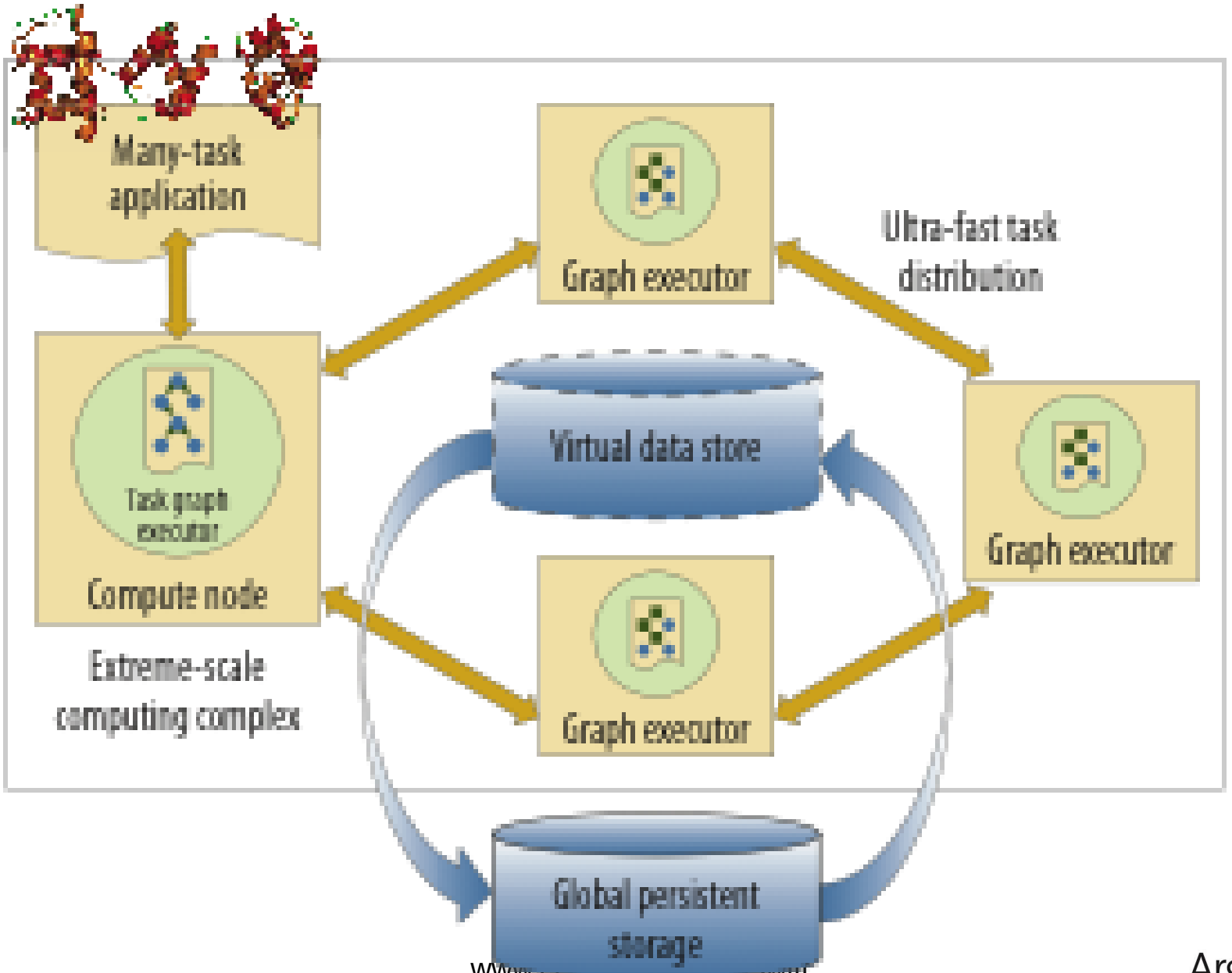
Swift work in progress

- ExM: extending *Swift* to the exascale realm
  - Jets: Running multiple MPI jobs under Swift agents
  - Turbine: Scaling up Swift with fully parallel evaluation
  - Collective data management: adapting Swift data management to best use storage hardware (broadcast, local RAM disk, gather() primitives)
- GPSI web portal: enables *Swift* use without programming
- Integration with Globus Online
  - Swift as an execution service under GO
  - Swift to use GO as a data transport service

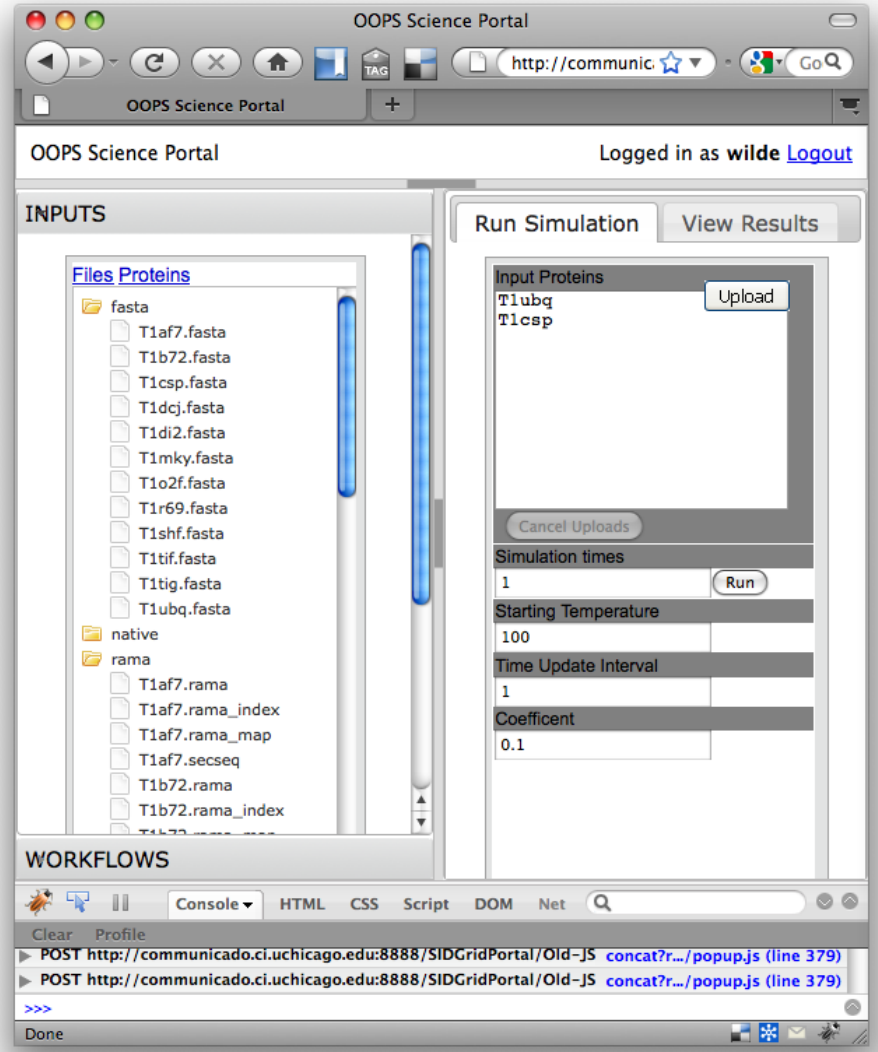
ExM project: scaling many-task computing to exascale

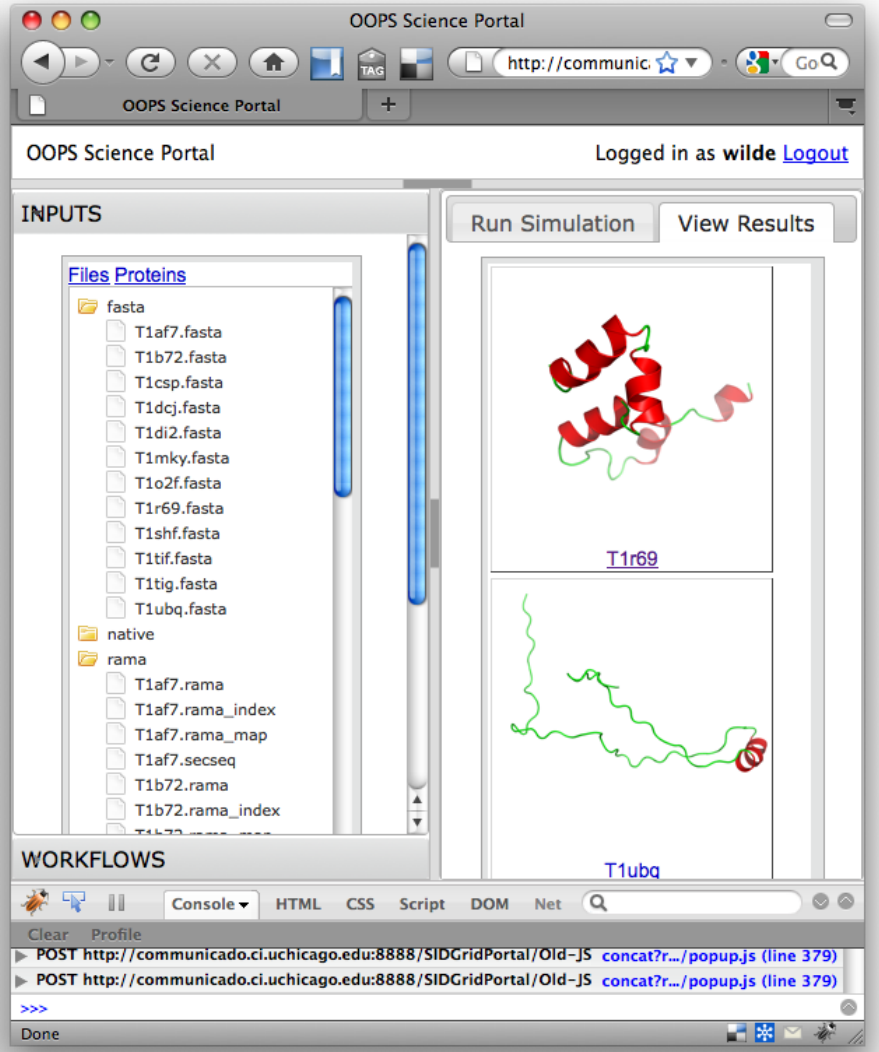
- Sponsored under DOE ASCR X-Stack program
- Extend Swift: tasks can be lightweight functions
  - Use Swift for the high-level logic of exascale applications
  - Retain functional semantics of input-process-output
- Highly distributed program evaluation
  - Re-building Swift based on an *intermediate representation* ("TIC") that lends itself to highly parallel evaluation
  - Scales to massive computing complexes
  - Distributed future store accessible in the manner of global arrays
  - Highly distributed program evaluation
  - Optimizations to reduce access to global future store
- Transparent distributed local storagemanagement
  - MosaStore aggregates local/RAM filesystems, make access more transparent through DHT methods

ExM: Scaling the many-task model to exascale



GPSI science portal for *Swift* workflow





- **Swift is a parallel scripting language** for multicores, clusters, grids, clouds, and **supercomputers**
  - *for loosely-coupled applications - application and utility programs linked by exchanging files*
  - *debug on a laptop, then run on a Cray*
- **Swift is easy to write**
  - *it's a simple high-level functional language with C-like syntax*
  - *Small Swift scripts can do large-scale work*
- **Swift is easy to run**: contains all services for running Grid workflow - in one Java application
  - *untar and run – Swift acts as a self-contained grid or cloud client*
  - ***Swift automatically runs scripts in parallel** – usually with no user input*
- **Swift is fast**: based on a powerful, efficient, scalable and flexible Java execution engine
  - *scales readily to millions of tasks*
- Swift **usage** is growing:
  - *applications in neuroscience, proteomics, molecular dynamics, biochemistry, economics, statistics, earth systems science, and more.*

Parallel Computing 37 (2011) 633–652

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Swift: A language for distributed parallel scripting

Michael Wilde [a,b,*], Mihael Hategan [a], Justin M. Wozniak [b], Ben Clifford [d], Daniel S. Katz [a], Ian Foster [a,b,c]

[a] Computation Institute, University of Chicago and Argonne National Laboratory, United States
[b] Mathematics and Computer Science Division, Argonne National Laboratory, United States
[c] Department of Computer Science, University of Chicago, United States
[d] Department of Astronomy and Astrophysics, University of Chicago, United States

ARTICLE INFO

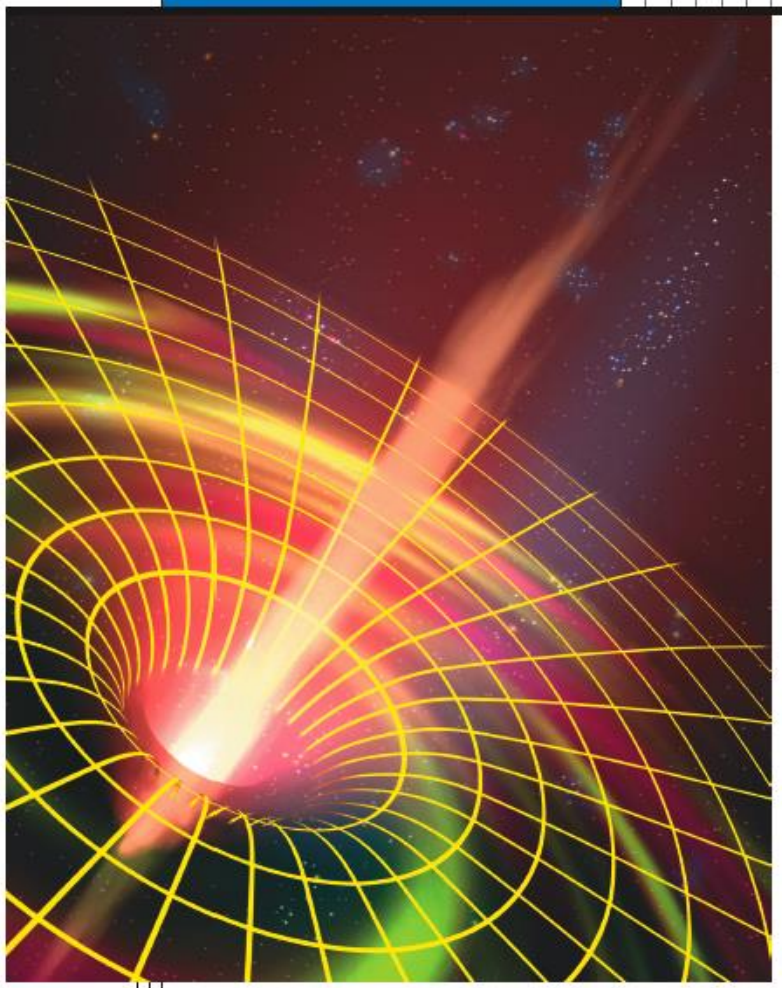
Article history:
Available online 12 July 2011

Keywords:
Swift
Parallel programming
Scripting
Dataflow

ABSTRACT

Scientists, engineers, and statisticians must execute domain-specific application programs many times on large collections of file-based data. This activity requires complex orchestration and data management as data is passed to, from, and among application invocations. Distributed and parallel computing resources can accelerate such processing, but their use further increases programming complexity. The Swift parallel scripting language reduces these complexities by making file system structures accessible via language constructs and by allowing ordinary application programs to be composed into powerful parallel scripts that can efficiently utilize parallel and distributed resources. We present Swift's implicitly parallel and deterministic programming model, which applies external applications to file collections using a functional style that abstracts and simplifies distributed parallel execution.

COVER FEATURE



PARALLEL SCRIPTING FOR APPLICATIONS AT THE PETASCALE AND BEYOND

Michael Wilde, Ian Foster, Kamil Iskra, and Pete Beckman,
University of Chicago and Argonne National Laboratory
Zhao Zhang, Allan Espinosa, Mihael Hategan, and Ben Clifford, *University of Chicago*
Ioan Raicu, *Northwestern University*

IEEE COMPUTER, Nov 2009