Workflow for extreme-scale systems

Presenter: Michael Wilde  wilde@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory
University of Chicago/Argonne Computation Institute
Outline

- Overview – context of workflow for science and engineering
- Workflow environments
- Expressing workflows – tools and programming models
- Workflow issues for extreme scale
- IO performance envelopes for workflow
- Expressing workflows in Swift
- Hands-on workflow examples and exercises using Swift
  - Language basics using Swift/K
  - Running Swift/T on Blue Gene/Q
  - Running Swift/K on Tukey
Definitions

- **Workflow:** the execution of a set of application programs
  - Often for a diverse set of application programs
  - Often with logical and physical dependencies
    - Logical: data dependencies
    - Physical: resource dependencies (space, processor, solution priorities)
  - Scripting is one way to implement workflows (Ad-hoc, Parallel libraries, Swift)
  - Generation of engine-specific input is another (DAGMan, Pegasus, Galaxy, Kepler)

- **Scripting:** higher-level dynamic programming
  - J. Ousterhout: “Scripting: Higher level programming for the 21st century”

- **High throughput computing (HTC)**

- **Many-task computing (MTC)**

- **Dataflow**

- **Data parallel vs. task parallel**
  - Workflow is almost always task-parallel at its outer levels
  - SPMD: typified by MPI
  - MPMD: multiple programs, multiple data – more typical of workflow
Many-task Applications

- Many-task Computing applications assemble existing parallel or sequential programs
- Those programs read and write data to a filesystem (but this limitation is being overcome...)
- Applications often have multiple stages
- Task dependencies between stages are in the form of file production and consumption
- Can have very high rates (e.g., hundreds per second) of very short tasks (minutes down to sub-second)

Slide courtesy of Zhao Zhang
When do you need workflow?
Typical application: protein-ligand docking for drug screening

- $O(10)$ proteins implicated in a disease
- $O(100K)$ drug candidates
- Tens of fruitful candidates for wetlab & APS
- 1M compute jobs

Work of M. Kubal, T.A. Binkowski, And B. Roux

www.ci.uchicago.edu/swift  www.mcs.anl.gov/exm
Parallel BLAST as a workflow


Based on script of D. Matthog by Z.Zhang, L. Gahelha

<table>
<thead>
<tr>
<th>Stage</th>
<th># Tasks</th>
<th># In</th>
<th># Out</th>
<th>In (MB)</th>
<th>Out (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>fastasplit</td>
<td>1</td>
<td>1</td>
<td>N</td>
<td>4039</td>
<td>4039</td>
</tr>
<tr>
<td>formatdb</td>
<td>N</td>
<td>N</td>
<td>3N</td>
<td>4039</td>
<td>4400</td>
</tr>
<tr>
<td>blastp</td>
<td>N*M</td>
<td>N+M</td>
<td>N*M</td>
<td>73<em>N</em>M</td>
<td>2.4<em>N</em>M</td>
</tr>
<tr>
<td>merge</td>
<td>M</td>
<td>N*M</td>
<td>M</td>
<td>2.4<em>N</em>M</td>
<td>4.8*M</td>
</tr>
</tbody>
</table>
Can workflow scale?
BLAST workflow lags MPI BLAST by \( \sim 5\% \)

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 language 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

http://swift-lang.org
Spatial normalization of functional MRI runs

Dataset-level workflow

Expanded (10 volume) workflow

http://swift-lang.org
Numerous many-task workflow applications

- Simulation of super-cooled glass materials
- Protein folding using homology-free approaches
- Climate model analysis and decision making in energy policy
- Simulation of RNA-protein interaction
- Multiscale subsurface flow modeling
- Modeling of power grid for OE applications
TARGET RESOURCES

- System types
  - Clouds
  - Clusters (campus, department)
  - Petascale HPC systems
  - Grids (OSG, LCG, ...)
  - Multi/many-cores – 256 core nodes!

- Patterns
  - A single big HPC machine
  - HPC Machine with attached resources
  - Extend campus cluster with cloud
  - Many HPC machines
  - Many combinations of above

Clouds: Amazon EC2, XSEDE Wispy, ...
Diffuse scattering workflows

- **Sample**
- **Material composition**: La 60% Sr 40% → Simulated structure → Simulated scattering
- **Evolutionary optimization**
- **Knowledge base**: Past experiments; simulations; literature; expert knowledge
- **Knowledge-driven decision making**
- **Select experiments (mins—hours)**
- **Detect errors (secs—mins)**
- **Contribute to knowledge base**
- **Simulations driven by experiments (mins—days)**
- **Experimental scattering**
Architecture realization for APS experiments

- **Experiment hutch**: Data Capture Host
  - **S1dserv**: (200 TB)
  - **sector dservers**

- **User**: Swift
  - Swift-enabled Galaxy Portal

- **Data Transfer**
  - **Catalog**
  - **Globus SaaS services**
  - Swift and Globus paths for jobs, data, and metadata

- **GridFTP parallel data path**

- **1 PB**
  - **GridFTP**
  - APS Data Archive
    - petrel (1 PB)

- **GridFTP**
  - APS Experiment Data Server
    - clutch

- **5 TB**
  - **Swift**
  - Fast Skim Cluster
    - orthros (320)

- **External Compute resources**
  - ALCF
    - mira (768K), tukey (1,000)
  - LCRC
    - blues, fusion (10K)
  - Cloud
    - pads (160), EC2 (1,000s)
Workflow patterns and issues

- Parameter sweeps
- Ensembles
- Data analysis
- Scaling studies
- Specialized patterns: uncertainty quantification, branch and bound
- Programming an application from libraries of applications
- Dataflow vs control flow
  - Ultimately, workflow is essentially dataflow
  - The difference is who writes and thinks about the dataflow
- Pipelining and concurrency (and how dataflow is good at this)
- Workflow manager drives application (outer workflow, inner scripts)
- Workflow manager embedded in application (outer scripts, inner workflow)
PROGRAMMING MODELS

- MPI, OpenMP, Hybrid
- Map reduce
- Record processing (with functions) vs file processing (with apps)
- Generating workflows for other engines
- Dynamically interpret the workflow
- Script mode (for Blue Gene, Cray systems)
- Dependent job processing
A partial sampler of workflow tools

- **High throughput tools**
  - Condor
  - Cluster schedulers / local resource managers (PBS, SGE, Cobalt, LSF, LL, SLURM,..)

- **Workflow task dependency managers**
  - DAGMan
  - Schedulers with job dependencies

- **Integrated dependency and data management**
  - Pegasus

- **Dataflow languages**
  - Dryad, Ciel, Swift

- **Big data solutions**
  - Hadoop, Spark, Zookeeper, Uzi

- **Multicore tools**
  - GNU Parallel, iPython parallel support

- **Languages with parallel support**
  - Py_nnn, Java_nnn, Haskell, R, MATLAB => PSOM, Parallel BASH (Walker)
A *sampler of workflow tools* (con’t)

- Interactive workflow frameworks
  - Galaxy
  - Taverna
  - Kepler
  - LONI Pipeline (neuroscience)
  - Microsoft Workflow manager
  - Airivata

- Science gateways
Galaxy workflow portal

Data Management

Data Analysis

http://swift-lang.org
Two fundamental problems in scaling workflow

- **Task rate**
  - 60,000 cores / 60 sec/task = 1,000 tasks per second!

- **Data management**
  - 1K tasks / sec may generate 5GB/sec – not so bad if blocked efficiently
  - 1K tasks / sec may generate 2,000 files / sec – not so easy
Multi-level scheduling: pilot jobs can improve task rate performance

- Pilot jobs are long-running meta-jobs
  - allocate compute resources and run many smaller jobs
- PANDA
  - Widely used on OSG and LCG by the ATLAS physics collaboration
- GWMS using Condor Glide-Ins
  - A generalized solution widely deployed on OSG
- SAGA and Bigjob
  - Obtaining good results on XSEDE resources
- Java CoG Coasters
  - Allocates/frees resources based on demand
    - Peaks at 600 tasks per second
- Falkon
  - Research system reached 3,000 tasks per second and 1B tasks
Workflow patterns and data exchange

Filesystem Access Patterns:
- File Creation
- File Open
- 1-to-1 Read
- N-to-1 Read
- Few-to-1 Read
- 1-to-1 Write

Some engineering problems and research challenges for extreme workflow

- **Engineering**
  - Diversity of interfaces, hard to tame and test, hard to abstract
  - Inter-language bindings and data interchange – challenge to usability
  - Integration with extreme-scale networks, runtimes and language stacks

- **Research**
  - Economics and policy-based scheduling
  - Retry/recovery of large distributed task and data graphs
  - Power management
  - Load balancing
  - Programming models: integration of dataflow and big-data techniques and tools
Summary: Challenges of workflow at extreme scale

- Inter-resource coordination
- Hybrid programming tools
- The challenges of data motion
  - Data management strategies and system envelopes
- The challenges of task scheduling and dispatch
  - Task rates and task distribution
  - Resource utilization vs. time to solution
- Workflow expression and separation of concerns
- Provenance: tracking what was done
The Swift parallel scripting language

Presenter: Michael Wilde  wilde@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory
University of Chicago/Argonne Computation Institute
- Parallel scripting language for clusters, clouds & grids
  - For writing loosely-coupled scripts of application programs and utilities linked by exchanging files
  - Can call scripts in shell, python, R, Octave, MATLAB, ...

- Swift does 3 important things for you:
  - Makes parallelism transparent – with functional dataflow
  - Makes basic failure recovery transparent
  - *Makes computing location transparent* – can run your script on multiple distributed sites and diverse computing resources (from desktop to petascale)
Swift programming model: all progress driven by concurrent dataflow

```swift
(int r) myproc (int i, int j) {
    int f = F(i);
    int g = G(j);
    r = f + g;
}
```

- \( F() \) 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 programming model

- **Data types**
  - `int i = 4;`
  - `int A[];`
  - `string s = "hello world";`

- **Mapped data types**
  - `file image<"snapshot.jpg">;`

- **Structured data**
  - `image A[]<array_mapper...>;`
  - `type protein {
  file pdb;
  file docking_pocket;
}
  protein p<ext; exec=protein.map>;`

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

---

*Swift: A language for distributed parallel scripting, J. Parallel Computing, 2011*
Language-driven: *Swift* parallel scripting

Swift runs parallel scripts on a broad range of parallel computing resources.
Programming model:
all execution driven by parallel data flow

```c
(int r) myproc (int i)
{
    j = f(i);
    k = g(i);
    r = j + k;
}
```

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

- This parallelism is *automatic*
- Works recursively throughout the program’s call graph
Encapsulation enables distributed parallelism

Encapsulation is the key to transparent distribution, parallelization, and automatic provenance capture
app( ) functions specify cmd line argument passing

To run:
psim -s 1ubq.fas -pdb p -t 100.0 -d 25.0 >log

In Swift code:

```swift
app (PDB pg, File log) predict (Protein seq, Float t, Float dt)
{
    psim "-c" "-s" @pseq.fasta "-pdb" @pg "–t" temp "-d" dt;
}

Protein p <ext; exec="Pmap", id="1ubq">;
PDB structure;
File log;

(structure, log) = predict(p, 100., 25.);
```
Large scale parallelization with simple loops

1000 Runs of the “predict” application

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

Analyze()
Nested parallel prediction loops in Swift

```swift
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. Sweep();
```

10 proteins x 1000 simulations x
3 rounds x 2 temps x 5 deltas x
= 300K tasks
Spatial normalization of functional run

Dataset-level workflow

Expanded (10 volume) workflow

www.ci.uchicago.edu/swift  www.mcs.anl.gov/exm
Complex scripts can be well-structured
programming in the large: fMRI spatial normalization script example

(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" );
}
Dataset mapping example: fMRI datasets

```swift
// On-Disk Data Layout

// Dataset mapping example: fMRI datasets

// Swift’s in-memory data model

type Study {
    Group g[ ];
}

type Group {
    Subject s[ ];
}

type Subject {
    Volume anat;
    Run run[ ];
}

type Run {
    Volume v[ ];
}

type Volume {
    Image img;
    Header hdr;
}
```

Mapping function or script
Nested loops can generate massive parallelism

Protein folding example:

Sweep( )
{
    int nSim = 1000;
    int maxRounds = 3;
    Protein pSet[] <ext; exec="Protein.map">;
    float startTemp[] = [ 100.0, 200.0 ];
    float delT[] = [ 1.0, 1.5, 2.0, 5.0, 10.0 ];
    foreach p, pn in pSet {
        foreach t in startTemp {
            foreach d in delT {
                ItFix(p, nSim, maxRounds, t, d);
            }
        }
    }
}

Sweep();

10 proteins x 1000 simulations x
3 rounds x 2 temps x 5 deltas
= 300K tasks
Flexible worker-node agents for execution and data transport

- Main role is to efficiently run Swift tasks on allocated compute nodes, local and remote
- Handles short tasks efficiently
- Runs over Grid infrastructure: Condor, GRAM
- Also runs with few infrastructure dependencies
- Can optionally perform data staging
- Provisioning can be automatic or external (manually launched or externally scripted)

http://swift-lang.org
Worker architecture handles diverse environments

Submit site

Swift

```swift
file a = compute(b, c);
```

Karajan

```xml
<execute task="compute"/>
```

API

Coaster Client

socket

Remote site

Coaster Service

Worker

Worker

Worker

Worker

sockets

sockets
Implementation: The job packing problem (II) (also not to scale)

- Commit jobs to blocks and adjust as necessary based on actual walltime

- The actual packing problem is NP-complete
- Solved using a greedy algorithm: always pick the largest job that will fit in a block first
Swift is a parallel scripting system for grids, clouds and clusters
- for loosely-coupled applications - application and utility programs linked by exchanging files

Swift is easy to write: simple high-level C-like functional language
- 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 – acts as a self-contained Grid client

Swift is fast: uses efficient, scalable and flexible “Karajan” execution engine.
- Scaling close to 1M tasks – .5M in live science work, and growing

Swift usage is growing:
- applications in neuroscience, proteomics, molecular dynamics, biochemistry, economics, statistics, and more.

Try Swift! http://swift-lang.org (Swift/K) and www.mcs.anl.gov/exm (Swift/T)