TOWARD MPI-BASED WORKFLOW EXECUTION: APPLICATIONS AND STANDARDIZATION

JUSTIN M WOZNIAK
Data Science & Learning Division
Argonne National Laboratory
woz@anl.gov

ECP
EXASCALE COMPUTING PROJECT

Joint Lab  April 17, 2019  Knoxville, TN
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 cell functionality overlaps with SEGMEnt at four locations:
  - β-catenin/proliferation
  - PI3K: Proliferation/Apoptosis
  - TGF-B/SMAD: Proliferation/Apoptosis
  - F53: 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. Cancer Workshop @ SC, 2016
Set record for data collection at Cornell High Energy Synchrotron Source
Argonne group sets record for largest x-ray dataset ever at CHESS, CHESS News 2015.
WHAT HAPPENS WHEN THE TASK IS MPI

Outline

- Background: Overview of the Swift/T language

- Advanced features
  - Big data
  - Model exploration
  - Parallel tasks

- Example application: Cancer Deep Learning Environment
  - Parallel tasks

- Toward MPI-based workflows
  - Requirements
  - Standardization
SWIFT/T BASICS
GOALS OF THE SWIFT LANGUAGE

Swift was designed to handle many aspects of the computing campaign

- Make it easy to run large batteries of external program or library executions
- Ability to integrate many application components into a new workflow application
- Enable complex tasks based in other scripting languages (e.g., Python) or parallel MPI tasks
- Provide rich programming language at the top level – fully generic
- Data structures for complex data organization
- Portability- separate site-specific configuration from application logic
- Logging, provenance, and plotting features
- Support implicit concurrency and conventional programming constructs
THE SWIFT PROGRAMMING MODEL

All progress driven by concurrent dataflow

\[
\text{(int } r \text{) myproc (int } i, \text{ int } j) \\
\{ \\
\quad \text{int } x = F(i); \\
\quad \text{int } y = G(j); \\
\quad r = x + y; \\
\} \\
\]

- \( F() \) 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 \textit{automatic}
- Works recursively throughout the program’s call graph
SWIFT SYNTAX

- **Data types**
  
  ```swift
  int i = 4;
  string s = "hello world";
  file image<"snapshot.jpg">;
  ```

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

- **Structured data**
  
  ```swift
  typedef image file;
  image A[];
  type protein_run {
    file pdb_in; file sim_out;
  }
  bag<blob>[] B;
  ```

- **Conventional expressions**
  
  ```swift
  if (x == 3) {
    y = x+2;
    s = strcat("y: ", y);
  }
  ```

- **Parallel loops**
  
  ```swift
  foreach f,i in A {
    B[i] = convert(A[i]);
  }
  ```

- **Data flow**
  
  ```swift
  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):

```
Dataflow program
  `→ Dataflow engine
      `→ Scheduler
          `→ Task  ... Task
```

500 tasks/s

Now have this (Swift/T):

```
Dataflow program
  `→ Engine
      `→ Queue
          `→ Task  ... Task
```

1.5 B tasks/s

---

- Turbine: A distributed-memory dataflow engine for high performance many-task applications. Fundamenta Informaticae 28(3), 2013
SWIFT/T: ENABLING HIGH-PERFORMANCE SCRIPTED WORKFLOWS

Supports tasks written in many languages

- Write site-independent scripts, translates to MPI
- Automatic task parallelization and data movement
- Invoke native code, script fragments
- Rapidly subdivide large partitions for MPI jobs in multiple ways

Swift/T control process

Swift/T worker

C
C++
Fortran

MPI

64K cores of Blue Waters
2 billion Python tasks
14 million Pythons/s

14M tasks/s

$ spack install stc

Swift/T: Scalable data flow programming for
distributed-memory task-parallel applications
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

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
ADVANCED FEATURES
FEATURES FOR BIG DATA ANALYSIS

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

  Application
  Dataflow, annotations

  Runtime
  Hard/soft locations

  Distributed data

• Collective I/O
  User and runtime coordinate data/task locations

  Application
  I/O hook

  Runtime
  MPI-IO transfers

  Distributed data

  Parallel FS

PARALLEL TASKS IN SWIFT/T

- Swift expression: $z = @\text{par}=8 \ f(x,y)$;
- When $x$, $y$ are stored, Turbine releases task $f$ with parallelism=8
- Performs ADLB_Put($f$, parallelism=8)
- Each worker performs ADLB_Get(&task, &comm)
- ADLB server finds 8 available workers
- Workers receive ranks from server
  - Perform MPI_Comm_create_group()
- ADLB_Get() returns: task=$f$, size(comm)=8
- Workers perform user task
  - communicate on comm
- comm is released by Turbine
- Can hand the communicator to LAMMPS, NAMD, DIY, CODES/ROSS, etc.

The core novel contributions of EMEWS are shown in green, these allow the Swift script to access a running Model Exploration (ME) algorithm, and create an inversion of control (IoC) workflow.

- Both green and blue boxes accept existing multi-language code.
- [http://emews.org](http://emews.org)
EMEWS: EXTREME-SCALE MODEL EXPLORATION WORKFLOWS IN SWIFT/T

- How do we couple complex model exploration algorithms with workflows?
  - Optimization, active learning, uncertainty quantification…

- http://emews.org

- From desktop to large-scale model exploration with Swift/T
ECP CODAR: SWIFT-CONTROLLED ADIOS TRANSFERS

- Enable Swift to dynamically lay out tasks
- Control large simulation/redistribute/analysis ensembles
- Highly flexible, programmable use of MPI subjobs

Model exploration algorithm

Swift workflow control and load balancing
EXAMPLE APPLICATION: CANDLE
CANDLE

- CANDLE (PI: Rick Stevens) is an Argonne led multi-DOE lab collaboration developing a suite of software to support scalable deep learning for cancer applications on DOE supercomputing resources
- Funded by the DOE Exascale Computing Project
- Developing implementations of deep neural networks on targeted problems related to the three core DOE-NCI Joint Design of Advanced Computing Solutions for Cancer (JDACS4C) pilot projects
- GitHub: http://github.com/ECP-CANDLE
WHAT IS HYPERPARAMETER OPTIMIZATION?

Hyperparameter optimization = HPO

- Neural networks have a large number of possible configuration parameters, called hyperparameters
  - Avoids collision with NN weights, which are sometimes called parameters
- Applying optimization can automate part of the design of the neural network

- In the cancer Pilot 1 autoencoder shown, the system can determine
  - How many neurons to put in each layer
  - What activation function to use
  - What batch size to use
  - Etc.
PARALLELISM STRATEGIES

10,000 x 10-1000 x 10-100 = 1M – 1000M processing elements

Hyperparameter Search: up to ~10,000x
Depends on search strategy

Data Parallel: 10x-1000x

Model Parallel 10x-100x  Model Parallel 10x-100x  ...  Model Parallel 10x-100x

Data Parallel 10x-1000x

Model Parallel 10x-100x  Model Parallel 10x-100x  ...  Model Parallel 10x-100x
WORKFLOW SUPPORT FOR ML FRAMEWORKS

- **Concurrency:**
  - Scalable task distributor
  - Intranode concurrency, accelerators left up to the framework
  - Multinode ML tasks are parallel tasks

- **Data management:**
  - Could reuse input staging methods
  - Intermediate caches via DataSpaces/Mochi

- **Software integration:**
  - Usually launch frameworks in separate process
  - Launching within process is a configuration challenge
  - Search methods launched within process
CANDLE HYPERPARAMETER LEARNING

• Search trajectory of mlrMBO (R model-based optimization) algorithm
• Each iteration does 300 evaluations (batch size)
• Minimum and average performance on validation data set decreases as the ME algorithm learns

CANDLE PERFORMANCE

- Delivers 1+ petaflop!

<table>
<thead>
<tr>
<th>Machine</th>
<th>Architecture (Nov 2017 Top 500 ranking)</th>
<th>Scheduler</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLCF Titan at Oak Ridge National Laboratory</td>
<td>Cray XK7, AMD Opteron / K20X (5)</td>
<td>PBS</td>
</tr>
<tr>
<td>NERSC Cori at Lawrence Berkeley National Laboratory</td>
<td>Cray XC40, Intel Xeon Phi / E5 (8)</td>
<td>SLURM</td>
</tr>
<tr>
<td>ALCF Mira at Argonne National Laboratory</td>
<td>BlueGene/Q, PowerPC A2 (11)</td>
<td>Cobalt</td>
</tr>
<tr>
<td>ALCF Theta at Argonne National Laboratory</td>
<td>Cray XC40, Intel Xeon Phi (18)</td>
<td>Cobalt</td>
</tr>
<tr>
<td>NIH Biowulf</td>
<td>HP Apollo XL1x0r, Intel Xeon E5 (66)</td>
<td>SLURM</td>
</tr>
<tr>
<td>Blues, Bebop at Argonne National Laboratory</td>
<td>Intel Xeon Phi / E5 (148)</td>
<td>PBS/SLURM</td>
</tr>
<tr>
<td>Beagle2 at University of Chicago</td>
<td>Cray XE6, AMD Operton (NA)</td>
<td>PBS</td>
</tr>
<tr>
<td>Midway, Midway2 at University of Chicago</td>
<td>Intel various / AMD Opteron (NA)</td>
<td>SLURM</td>
</tr>
</tbody>
</table>
PARALLEL TASKS IN CANDLE WORKFLOWS

Complex concurrency structures

- Model parallelism: running the same network across nodes

- **Library approach:**
  - Use Swift/T @par syntax
  - Uses MPI 3 to dynamically create communicator from group
  - User task library accepts communicator via function input
  - Approach developed for other scientific computing cases, LAMMPS, NAMD, DIY, etc.
  - Have patched Horovod library
  - Should work with LBANN

- **MPI_Launch approach**
  - Use Swift/T launch() function
  - Creates MPI 3 group
  - Launches mpiexec on those resources, creating a new MPI_COMM_WORLD and separate processes (fault tolerance)
  - Works on clusters and Cray
  - Would work with unmodified Horovod or LBANN
**ECP INTERACTION: CODAR, CANDLE**

- **CANDLE** workflows produce a great number of medium-sized ML models
- **Goal:** Cache these on compute node storage for *possible* later use. Need to flush to global FS before end of run, but many models will be discarded
- **Approach:** Integrated Swift/T workflow system used in CANDLE with DataSpaces client
- Provide an opportunity for workflow-based data analysis and I/O reduction
- Demonstrate the utility of node-local storage for complex workflows

- Scaling deep learning for cancer with advanced workflow storage integration. Proc. MLHPC @ SC 2018.
TOWARD MPI_COMM_LAUNCH()
NEW MPI_LAUNCH FEATURE

- Allows Swift/T to run external parallel programs on subcommunicators inside a large allocation on a big machine

- Swift/T syntax:

  ```
  @par=8 launch("./my-program", args[], envs[]);
  ```

- Provides:
  - Scalable, in-place job launch
  - Handles cases where called program crashes
  - Can pack many such variably-sized programs within a large workflow

- Based on a proposed MPI function (available on clusters and Cray):
  ```
  int MPIX_Comm_launch(const char *cmd, char **argv,
                      MPI_Info info, MPI_Comm comm,
                      int *exit_code);
  ```
CONTEXT

- Want to run scientific ensembles controlled by C/C++/Fortran programs:
  - Parameter sweeps, searches, optimizations
  - Tests under varying parameters and process counts
  - Workflows and code coupling cases where jobs can exit or fail

- State of the art:
  - MPI_Comm_spawn() has limited availability
    • in part due to complexity of implementing?
  - MPI_Comm_spawn() does not support job exit detection and failures
  - Users write complex shell scripts against vendor-specific job launchers

- Proposing an alternate function: MPI_Comm_launch()

- Implemented on clusters (via MPICH/mpiexec hack) and by a vendor (internal)
MPI_COMM_LAUNCH(): DETAILS

- \texttt{int MPIX\_Comm\_launch(const char *cmd, char **argv, }
  \texttt{MPI\_Info info, MPI\_Comm comm, }
  \texttt{int* exit\_code);}

- Runs “in-place” on given communicator- no interaction with scheduler, etc.
- Parent is blocked
- No communication between parent and child
- Parent gets exit code- easy recovery from child failures
MPI_COMM_LAUNCH(): BENEFITS

- Allows user to use familiar communicator management to setup subjob
  - We have done things with MPI_Comm_split() and MPI_Comm_create_group()

- Easy to work with unmodified child codes

- Allows for the development of MPI-based workload management systems
  - Write simple test harnesses or parameter sweeps in C or Fortran + MPI
  - Or use an MPI-based system like ADLB, Swift/T, or MPI-Bash (all implemented!)
  - Possibly work with other parallel programming systems?
    - Collaboration opportunity?
SWIFT/T EXAMPLE

When file A is created, launch N sub jobs of varying size

```swift
define file B[]; // Define an array of file variables
A => {
    foreach i in [0:N-1] {
        file B_i="B-%i.txt"%i;
        string args_B[] = [ int2string(i),
                          filename(A), filename(B_i) ];
        @par=i launch("./child.x", args_B) => B_i = touch();
        B[i] = B_i;
    }
}
```

- Child tasks are load-balanced, MPI_Comm_create_group() is done automatically!
MPI-BASH EXAMPLE

Author: Scott Pakin (LANL)

- Forked and extended for `MPI_Comm_launch()` by Wozniak

```bash
#!/usr/bin/env mpibash
enable -f mpibash.so
mpi_init mpi_init
mpi_comm_rank rank

mpi_barrier
if [ $rank -eq 0 ]; then
  iter=0
  while [ $iter -lt $niter ]; do
    mpi_send 1 X
    mpi_recv 1 msginfo
    let iter++
  done

# Run with:
$ mpirun -np 16 ./my-script.sh
```

```bash
#!/usr/bin/env mpibash
enable -f mpibash.so
mpi_init mpi_init
mpi_comm_rank rank
mpi_comm_split $rank $rank newcomm

echo newcomm: $newcomm

mpi_comm_set $newcomm
mpi_comm_rank rank

mpi_comm_launch hostname
echo exit_code $?
```

# Run with:
$ mpirun -np 16 ./my-script.sh
```
NEWER MPI_LAUNCH_MULTI FEATURE

- Allows Swift/T to run external parallel program groups on subcommunicators inside a large allocation on a big machine
  - Call these groups Functional Online Bundles (FOBs)

- Swift/T syntax:

  ```swift
  @par=8 launch_multi(procs[], programs[], args[][], envs[][], <colors>);
  ```

- Provides:
  - Scalable, in-place simultaneous job launch
  - The programs are able to find each other and communicate with ADIOS (or other techniques)
  - Job layout can be controlled with the optional colors argument
  - A variety of other controls are available via special environment variables
WORKFLOW ABSTRACTIONS FOR IN SITU

- Need to provide workflow structure to capture in situ communication
- Default load-balanced scheduling results in disordered layout
- L: LAMMPS  A: ADIOS  V: Voro++

MPI Jobs within MPI Jobs
Under review for FGCS
LINKS

- Swift/T Home: http://swift-lang.org/Swift-T
- Swift/T Sites Guide: http://swift-lang.github.io/swift-t/sites.html
- Swift/T GitHub: https://github.com/swift-lang/swift-t
- Support: https://groups.google.com/forum/#!forum/swift-t-user
- Other papers: http://swift-lang.github.io/swift-t/pubs.html
This research was supported by the Exascale Computing Project (17-SC-20-SC), a joint project of the U.S. Department of Energy’s Office of Science and National Nuclear Security Administration, responsible for delivering a capable exascale ecosystem, including software, applications, and hardware technology, to support the nation’s exascale computing imperative.
MATHEMATICAL EXPRESSION FOR HPO

- For a given problem:
  - A loss function $F$ is determined on a given NN (usually accuracy)
  - The hyperparameter optimization problem is to minimize $F(p)$,
    - for all hyperparameter sets $p$ in the valid parameter space $P$,
    - however, $P$ is large and $F$ is expensive.
    - $P$ is the cross product of all valid network settings,
      - some of which may be categorical, some integer, some continuous.
    - Evaluating $F$ involves training the network on a training data set and applying it to the validation set

- We can use a generic, previously developed method to optimize $F$!

- These methods require and can use large compute resources
BASIC STRATEGIES

- Grid search
- Random search

- Generic optimization
  - Stochastic gradient descent
  - Evolutionary algorithms
  - Model-based optimization (mlrMBO in R)

- NN hyperparameter-specific optimization
  - Hyperopt, NEAT, Optunity, …
SUMMARY OF KEY CANDLE POINTS

- What about Swift/T enables CANDLE?
  - A workflow system that is actually a hierarchical programming language
  - Runs entirely on the compute nodes
  - Uses standard APIs for HPC (MPI), allows for minimal OS environment
  - Very scalable
  - Supports MPI tasks, embedded Python, R interpreters

- What about EMEWS enables CANDLE?
  - Allows user to focus on two sequential codes
    - The optimizer
    - Their objective function code
  - Everything else is managed by the system
DISTRIBUTED DATAFLOW PROCESSING

- **Code**
  \[
  \]
  \[
  \]

- **Engines:** evaluate dataflow operations
  - **Perform** `getenv()`
  - **Submit** \( f \)
  - **Subscribe to** \( A[2] \)
  - **Submit** \( g \)

- **Workers:** execute tasks
  - **Task put**
    - **Process** \( f \)
    - **Store** \( A[2] \)
  - **Notification**
    - **Process** \( g \)
    - **Store** \( A[3] \)

SWIFT/T COMPILER AND RUNTIME

- STC translates high-level Swift expressions into low-level Turbine operations:
  - Create/Store/Retrieve typed data
  - Manage arrays
  - Manage data-dependent tasks
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]);