Motivation: Many-Task Applications

Simple in some dimensions:
- Coarse-grained task parallelism: tasks are function
calls, command-line executables, with serial or
fine-grained parallelism inside
- Can express high-level logic with single-assignment
variables and structured control flow

Challenging in others:
- Irregular parallelism: needs load balancing & task priorities
- Extreme scale (10,000+ cores) with distributed memory
- File system often used for input, output &
intermediate data
- Legacy or closed-source code in many languages
- Limited time budget, no parallel programming gurus

Swift Programming Language

Mix of functional and imperative ideas
- Close correspondence between imperative script and Swift
- Single-assignment variables, deterministic by default

Hierarchical programming model
- Wrap C functions, command-line apps as Swift functions
- First-class file, Binary Large OJect variables

Implicit, global-view parallelism
- All statements in block can execute asynchronously
- Asynchronous tasks executed in data dependency order
- Transparent task & data movement between cluster nodes
- First-class file, Binary Large OJect variables
- Wrap C functions, command-line apps as Swift functions
- Implicit, global-view parallelism

Example Applications

- Motivation: Many-Task Applications
- Illustrative optimization
- Branch and bound

Example Applications

- Parameter sweeps
- Iterative optimization
- Branch and bound

ExM Language Stack

STC Optimizing Compiler [2]
- Compile-time error checking
- Custom intermediate representation for dataflow programs
- Standard optimization techniques reduce communication

Turbine Dataflow Engine [3]
- Shared data store
- Single-assignment variables
- Data structures (e.g. hash tables)
- Data-dependent task launching
- Commutative data operations for language-level determinism

ExM Language Stack

Application | Measured | Required
--- | --- | ---
Tasks | Task Dur. | Tasks | Task Rate
Power Grid Simulation | 10,000 | 15 s | 20  | 6.6 X 10^9 s
DSSAT | 500,000 | 12 s | 10^3 | 10^3 X 10^9 s
SciColSim | 10,800,000 | 10 s | 10^7 | 10^7 X 10^9 s
SWAT | 2,200 | 120 s | 10^2 | 10^2 X 10^9 s
ModFTDock: dock | 12,000 | 1,000 s | 10^3 | 10^3 X 10^9 s
ModFTDock: score | 12,000 | 6,000 s | 10^4 | 10^4 X 10^9 s
Quantitative description of applications and required performance on 1 million cores

ModFTDock: Protein Docking in Swift

```swift
dock_score scores[];
foreach p1, i in proteins {
    docked[] = modftdock(p1, p2);
}
```

M. Parisien, T. Sosnick, T. Pan, K. Maheshwari

MosaStore File System

- Parallel FS’s: unsuited for small reads/writes, many files
- POSIX “intermediate file system” uses aggregated memory of cluster nodes to store data [5]
- Data caching, batching of small operations
- Cross-layer optimization with hints [6][7]: file placement, replication, data-aware task scheduling, block-size, etc

Project Status

- Simple benchmarks on 10,000+ cores with high utilization [2]
- Language stack working end-to-end with real Swift programs: simulated annealing, branch-and-bound Sudoku solver
- Compiler optimization reduces runtime ops. 5x-10x [2]
- MosaStore with cross-layer optimization gives speedups of 20-40% on data-intensive workloads
- Work on FS/language integration in progress
- Many language features missing
- Much tuning, optimization, etc, remains to be done
- Fault tolerance, energy-awareness to be explored further

Further reading


ExM is supported by the United States Department of Energy, Office of Advanced Scientific Computing Research, under X-Stack Grant DE-SC0005380.

http://exm.xstack.org