ExM: High-level Dataflow Programming for Extreme-scale

Tim Armstrong, Justin M. Wozniak, Michael Wilde, Ketan Maheshwari, Daniel S. Katz, Matei Ripeanu, Ewing Lusk, Ian T. Foster

TA, DK, IF: University of Chicago *JW, MW, KM, DK, EW, IF: Argonne National Laboratory* *MR: University of British Columbia*

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 OBject variables

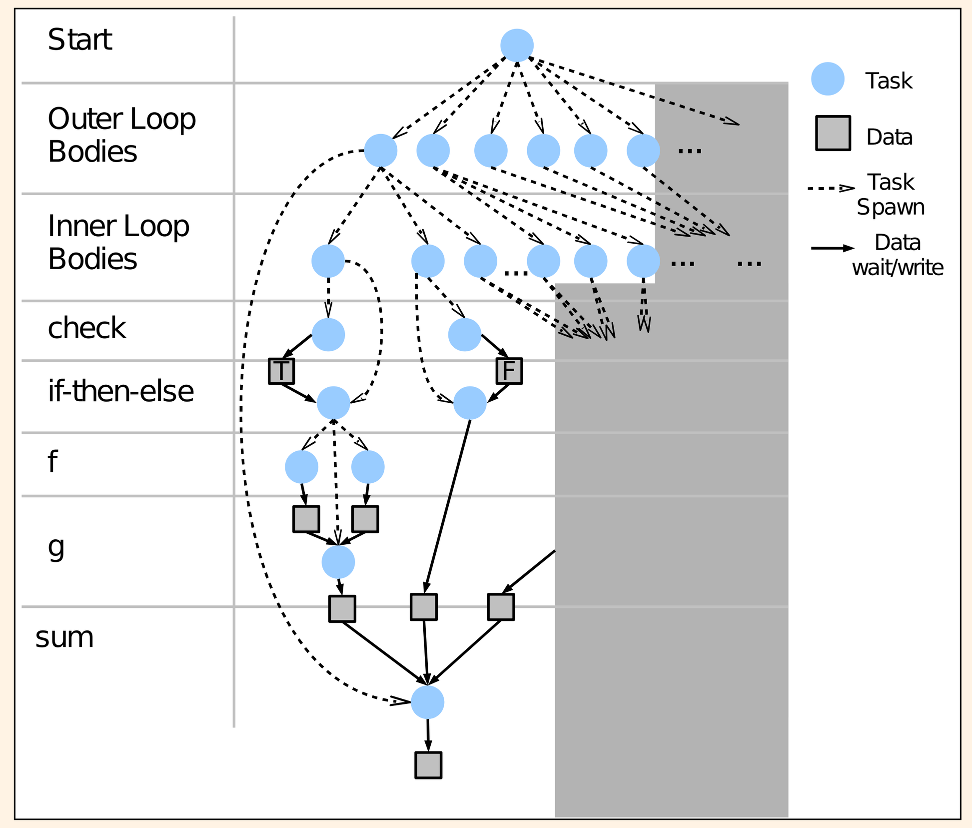
Implicit, global-view parallelism

- All statements in block can execute asynchronously
- Asynchronous tasks executed in data dependency order
- Transparent task & data movement between cluster nodes

[1] describes original Swift language and implementation
[2] describes ground-up ExM reimplementation

```
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]);
}
```

Structured control flow in Swift



Execution trace of script (arrays omitted)

Example Applications

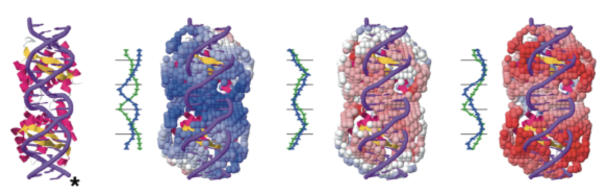
- Parameter sweeps
- Iterative optimization
- Branch and bound

Application	Measured		Required	
	Tasks	Task Dur.	Tasks	Task Rate
Power-grid Distribution	10,000	15 s	10^9	6.6×10^4/s
DSSAT	500,000	12 s	10^9	8.3×10^4/s
SciColSim	10,800,000	10 s	10^9	10^5/s
SWAT	2,200	120 s	10^5	8.3×10^3/s
ModFTDock stages: dock	1,200,000	1,000 s	10^9	10^3/s
modmerge	12,000	5 s	10^7	2×10^5/s
score	12,000	6,000 s	10^7	166/s

Quantitative description of applications and required performance on 1 million cores

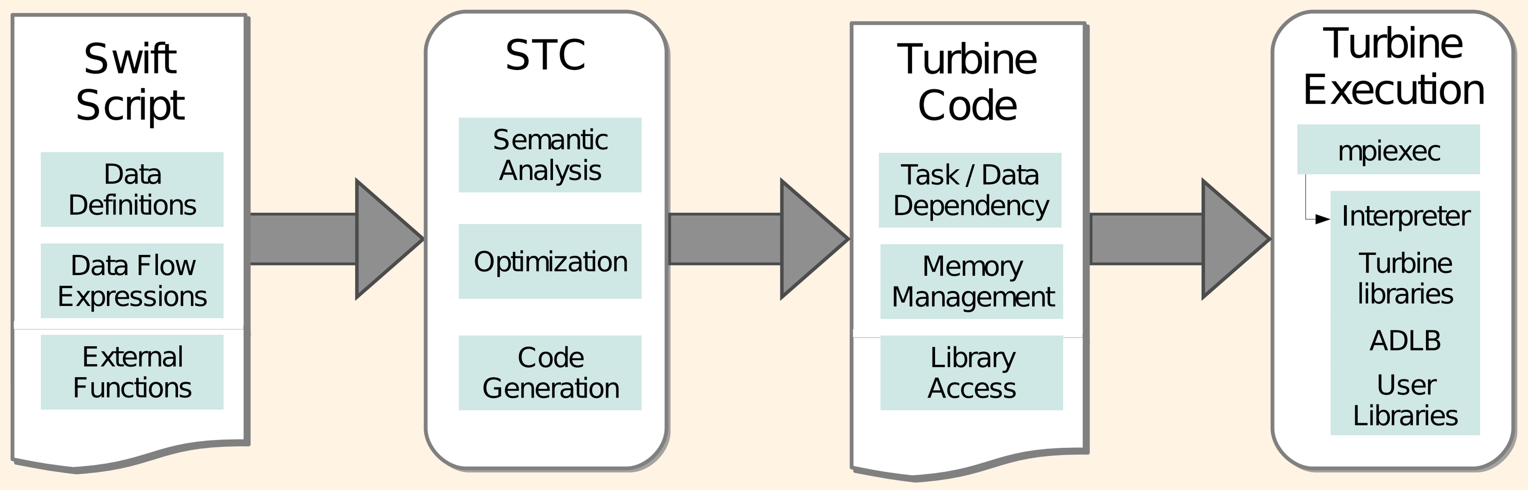
ModFTDock: Protein Docking in Swift

```
dock_score scores[];
foreach p1, i in proteins {
  dock_result docked[];
  foreach (p2, j in proteins) {
    if (i < j) {
      docked[j] = modftdock(p1, p2);
    }
  }
  scores[i] = score(merge(docked));
}
```



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

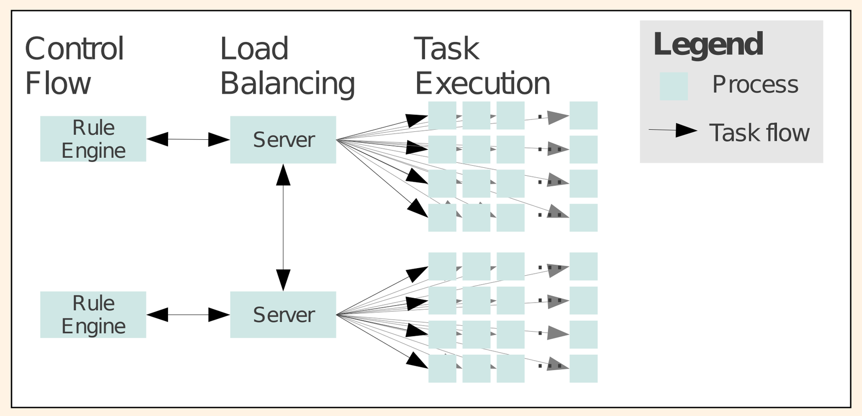
ExM Language Stack



STC Optimizing Compiler [2]

- Compile-time error checking
- Custom intermediate representation for dataflow programs
- Standard optimization techniques reduce communication

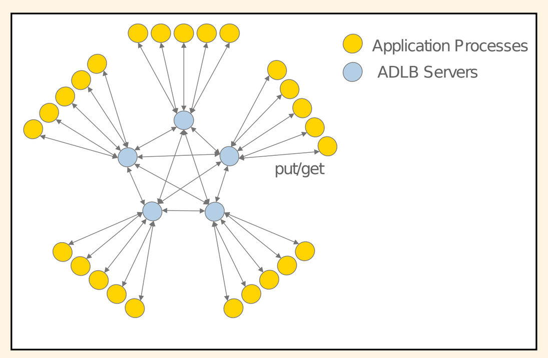
Turbine Dataflow Engine [3]



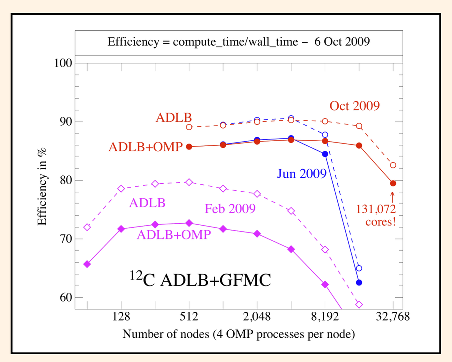
- Shared data store
- Single-assignment variables
- Data structures (e.g. hash tables)
- Data-dependent task launching
- Commutative data operations for language-level determinism

ADLB Load Balancer [4]

- MPI-based
- Highly scalable: 100k+ cores
- Task priorities



ADLB architecture



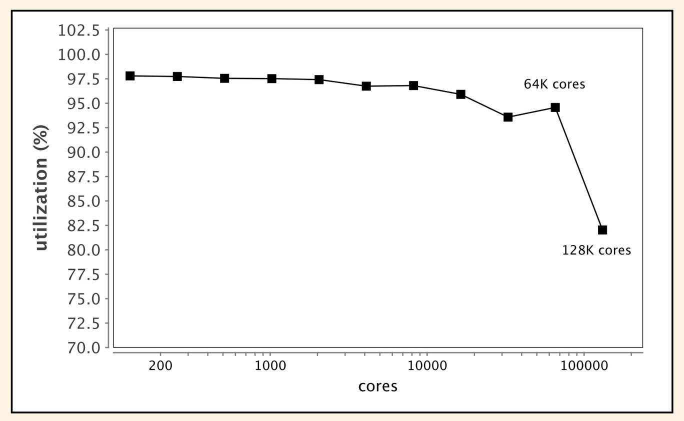
Evolution of ADLB scalability

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



System utilization for batch of 100s independent tasks on Blue Gene/P Intrepid [2]

Optimizations	Rule	Store	Load	Subscribe	Insert	Lookup
Unoptimized	52422	42646	78470	113905	5871	11445
+ Cfp + DCElim	52422	41629	77454	112857	5871	11445
+ Const share	52422	30174	77454	112852	5871	11445
+ Fwd data-flow	4114	4681	12272	15437	5871	10645
+ Unroll loops	4014	4643	12111	15213	5871	10595

Runtime operation counts in simulated annealing run by optimization level. Each row includes prior optimizations [2]

THE UNIVERSITY OF CHICAGO

Computation Institute

Argonne NATIONAL LABORATORY

UBC a place of mind THE UNIVERSITY OF BRITISH COLUMBIA

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

Further reading

http://exm.xstack.org

[1] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, I. T. Foster, "Swift: A language for distributed parallel scripting," Parallel Computing 2011
[2] J. M. Wozniak, T. Armstrong, M. Wilde, D. S. Katz, E. Lusk, I. T. Foster, "Swift/T: scalable data flow programming for many-task applications," in submission, SC'12
[3] J. M. Wozniak, T. Armstrong, E. L. Lusk, D. S. Katz, M. Wilde, and I. T. Foster, “Turbine: A distributed memory data flow engine for many-task applications,” SWEET’12
[4] E. L. Lusk, S. C. Pieper, and R. M. Butler, “More scalability, less pain: A simple programming model and its implementation for extreme computing" SciDAC Rev. 2010
[5] S. Al-Kiswany, A. Gharaibeh, and M. Ripeanu, “The case for a versatile storage system,” SIGOPS '10
[6] E. Varanaithan, S. Al-Kiswany, L. Costa, M. Ripeanu, Z. Zhang, D. Katz, M. Wilde, "A workflow aware storage system: an opportunity study," Proc. CCGrid 2012
[7] S. Al-Kiswany, E. Vairavanathan, A. Barros, et. al. "The case for cross-layer optimizations in storage: a workflow-aware storage system." In submission, SC '12