X10 at Petascale

Olivier Tardieu, Benjamin Herta, David Cunningham, David Grove, Prabhanjan Kambadur, Vijay A. Saraswat, Avraham Shinnar, Mikio Takeuchi, Mandana Vaziri

IBM Research

ABSTRACT

X10 is a high-performance, high-productivity programming language aimed at large-scale distributed and shared-memory parallel applications. It is based on the Asynchronous Partitioned Global Address Space (APGAS) programming model, supporting the same fine-grained concurrency mechanisms within and across nodes.

We demonstrate that X10 delivers solid performance at petascale by running (weak scaling) eight application kernels on an IBM Power 775 supercomputer utilizing up to 55680 Power7 cores (1.7 Pflop/s). We sketch advances in distributed termination detection, distributed load balancing, and use of high-performance interconnects that enable X10 to scale out to thousands of nodes.

1. OVERVIEW

X10 is a high-performance, high-productivity programming language developed at IBM. It is a class-based, strongly typed, garbage-collected, object-oriented language [8, 7]. To support concurrency and distribution, X10 uses the Asynchronous Partitioned Global Address Space programming model (APGAS [6]). This model introduces two key concepts – places and asynchronous tasks – and a few mechanisms for coordination. With these, APGAS can express both regular and irregular parallelism, message-passing-style and active-message-style computations, fork-join and bulk-synchronous parallelism. In contrast to hybrid models like MPI+OpenMP, the same constructs underpin both intra- and inter-place concurrency.

We present experimental results for eight kernels.[1] We implement the four HPC Class 2 Challenge benchmarks: *HPL*, *FFT*, *RandomAccess*, and *Stream Triad* [3], as well as *Smith-Waterman* [10], *Betweenness Centrality* [1], *K-Means* [4], and *Unbalanced Tree Search* (UTS) [5]. We run them on a large Power 775 system with a theoretical peak performance of 1.7 Pflop/s. For the HPC Challenge benchmarks, X10 today achieves 41% to 87% of the system's potential at scale as reported by IBM's optimized runs entry to the HPC Class 1 Challenge in Nov. 2012 [2]. To the best of our knowledge, our UTS implementation is the first to scale linearly to petaflop systems for geometric trees. Our K-Means and Smith-Waterman codes also scale linearly. Although still statically load-balanced, our Betweenness Centrality code can process 245 Billion edges per second using 47040 cores.

These results have been made possible with our solutions to the distributed termination detection problem at scale, the effective use of high-performance interconnects, and a refined distributed load balancing scheme for UTS.

[1]The X10 tool chain and the benchmark codes are publicly available at `http://x10-lang.org`.

2. OPTIMIZING FOR PETASCALE

The X10 programming model relies heavily on termination detection. Detection scopes can be nested, distributed, and collocated. The reference implementation has to handle arbitrary distributed task graphs and cope with arbitrary network latencies. It does not scale. We identify a series of common patterns of distributed termination detection and provide specialized scalable implementations for these, relying on a combination of static analysis, dynamic analysis, and user input (pragmas) to guide selection.

Supercomputers such as Power 775 are built upon high-performance interconnects and provide network acceleration mechanisms such as collectives and RDMAs. We augment X10's communication APIs to expose these primitives when available and emulate them when not (e.g., over ethernet).

The UTS benchmark measures the rate of traversal of a tree generated on the fly using a splittable random number generator. The tree is highly irregular and unpredictable. Dynamic distributed load balancing is therefore indispensable. We start from the state-of-the-art lifeline-based load balancer described in [9], which only scales well to hundreds of Power 775 nodes, and make it scale to thousands of nodes. The reference algorithm cleverly combines random work-stealing with organized work-sharing to efficiently distribute work dynamically. We improve the work queue implementation by compactly representing intervals of sibling nodes. We reduce termination detection overheads by separating work-stealing and work-sharing termination scopes. We optimize routes by taming the random victim selection and exploiting indirect routes in termination detection.

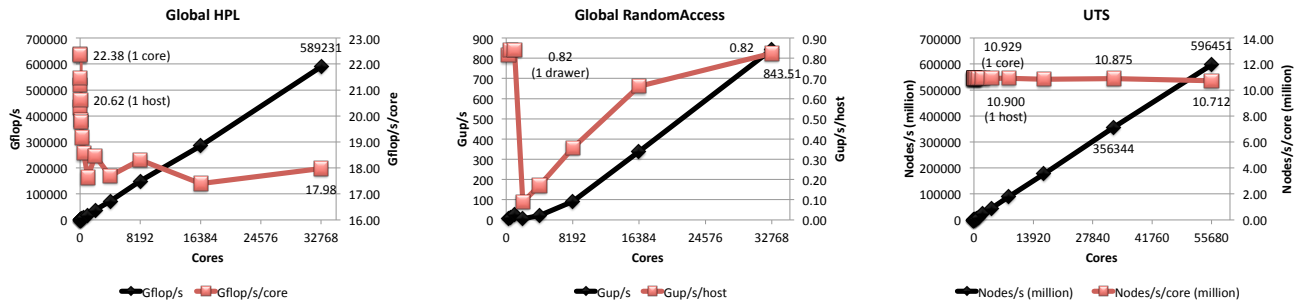

Figure 1: Performance of the X10 Implementations of HPL, RandomAccess, and UTS.

Benchmark	X10 performance		Hosts	Efficiency	Performance relative
	with a single host	at scale	used	at Scale	to Class 1 at scale
Global HPL	20.62 Gflop/s/core	17.98 Gflop/s/core	1024	87%	85%
Global RandomAccess	0.82 Gup/s/host	0.82 Gup/s/host	1024	100%	81%
Global FFT	0.88 Gflop/s/core	0.88 Gflop/s/core	1024	100%	41%
EP Stream (Triad)	7.23 GB/s/core	7.12 GB/s/core	1740	98%	87%
UTS	10.900 M nodes/s/core	10.712 M nodes/s/core	1740	98%	n/a
K-Means	6.16s run time	6.27s run time	1470	98%	n/a
Smith-Waterman	12.68s run time	12.87s run time	1470	98%	n/a
Betweenness Centrality	11.59 M edges/s/core	5.21 M edges/s/core	1470	45%	n/a

Table 1: Performance at Scale versus Single-Host Performance and versus HPC Class 1 Optimized Runs.

3. BENCHMARKS

Each host of the Power 775 system has 32 Power7 cores and 982 Gflop/s of theoretical peak performance. We bind one single-threaded X10 place to each core. We run (weak scaling) every benchmarks with at least 1024 hosts. We rely on ESSL, FFTW, and SHA1 libraries for sequential kernels.

The HPC Challenge benchmarks provide an opportunity to compare programming models for concurrency and distribution [3]. In 2012, IBM entered both the Class 1 – best performance – and Class 2 – most productivity – competitions for this Power 775 system. The Class 1 entry used low-level implementations intended to achieve the highest possible performance [2]. For the Class 2 entry, we demonstrated X10 implementations of the benchmarks [11].

In the last column of Table 1, we compare the per-host performance of the two implementations at scale. In short, X10 delivers between 41% and 87% of the Class 1 codes hand- and auto-tuned by IBM's best experts. Our FFT code is primarily handicapped by untuned sequential code. Our RandomAccess code performs just as well as the UPC code. Both are limited by the network stack, which the Class 1 code bypasses. The sharp performance drop for mid-size runs is characteristic of the Power 775 network topology.

In the *efficiency at scale* column, we compare the per-host performance at scale to the single-host performance (both X10). Six benchmarks, including UTS, scale perfectly with 98% efficiency and above. Global HPL is at 87%. Betweenness Centrality efficiency is lower at 45% in part because we do not use dynamic distributed load balancing for it yet.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency under its Agreement No. HR0011-07-9-0002.

4. REFERENCES

[1] U. Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25:163–177, 2001.

[2] HPC Challenge Benchmark Record 495. `http://icl.cs.utk.edu/hpcc/hpcc_record.cgi?id=495`, Nov. 2012.

[3] HPC Challenge. `http://icl.cs.utk.edu/hpcc/`.

[4] J. MacQueen. Some methods for classification and analysis of multivariate observations. Proc. 5th Berkeley Symp. Math. Stat. Probab., Univ. Calif. 1, 281–297, 1967.

[5] S. Olivier, J. Huan, J. Liu, J. Prins, J. Dinan, P. Sadayappan, and C.-W. Tseng. UTS: An unbalanced tree search benchmark. In *LCPC'06*, pages 235–250, 2007.

[6] V. Saraswat, G. Almasi, G. Bikshandi, C. Cascaval, D. Cunningham, D. Grove, S. Kodali, I. Peshansky, and O. Tardieu. The Asynchronous Partitioned Global Address Space Model. In *AMP'10*, June 2010.

[7] V. Saraswat, B. Bloom, I. Peshansky, O. Tardieu, and D. Grove. The X10 language specification, v2.2.3. Aug. 2012.

[8] V. Saraswat and R. Jagadeesan. Concurrent clustered programming. In *Concur'05*, pages 353–367, 2005.

[9] V. A. Saraswat, P. Kambadur, S. Kodali, D. Grove, and S. Krishnamoorthy. Lifeline-based global load balancing. In *PPoPP '11*, pages 201–212, 2011.

[10] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[11] O. Tardieu, D. Grove, B. Bloom, D. Cunningham, B. Herta, P. Kambadur, V. A. Saraswat, A. Shinnar, M. Takeuchi, and M. Vaziri. X10 for Productivity and Performance at Scale, A Submission to the 2012 HPC Class II Challenge, Nov. 2012.