PES
Software Engineering
for Embedded Systems

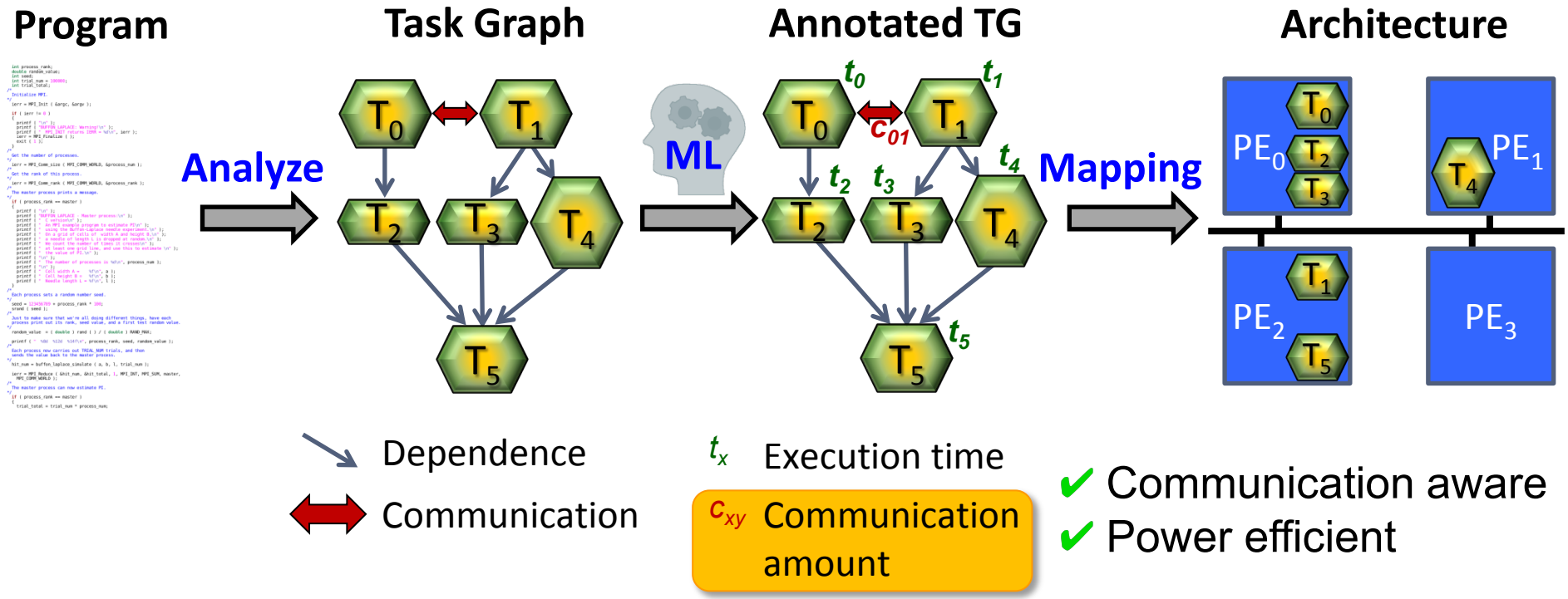
TU berlin

Towards Predicting Recursion Depth Using Machine Learning to Improve Task Mapping

Dirk Tetzlaff, Sabine Glesner
Berlin Institute of Technology (TU Berlin)

4th Workshop on Mapping of Applications to MPSoCs
St. Goar, June 29th, 2011

ML-based Task Mapping[Map2MPSoCs'10]



Dependence

Communication

t_x Execution time

c_{xy} Communication amount

✔ Communication aware

✔ Power efficient

- Communication within loop bodies
  - ✔ Predict loop iteration count[CiSE'10]
- Communication within recursive functions
  - Predict recursion depth

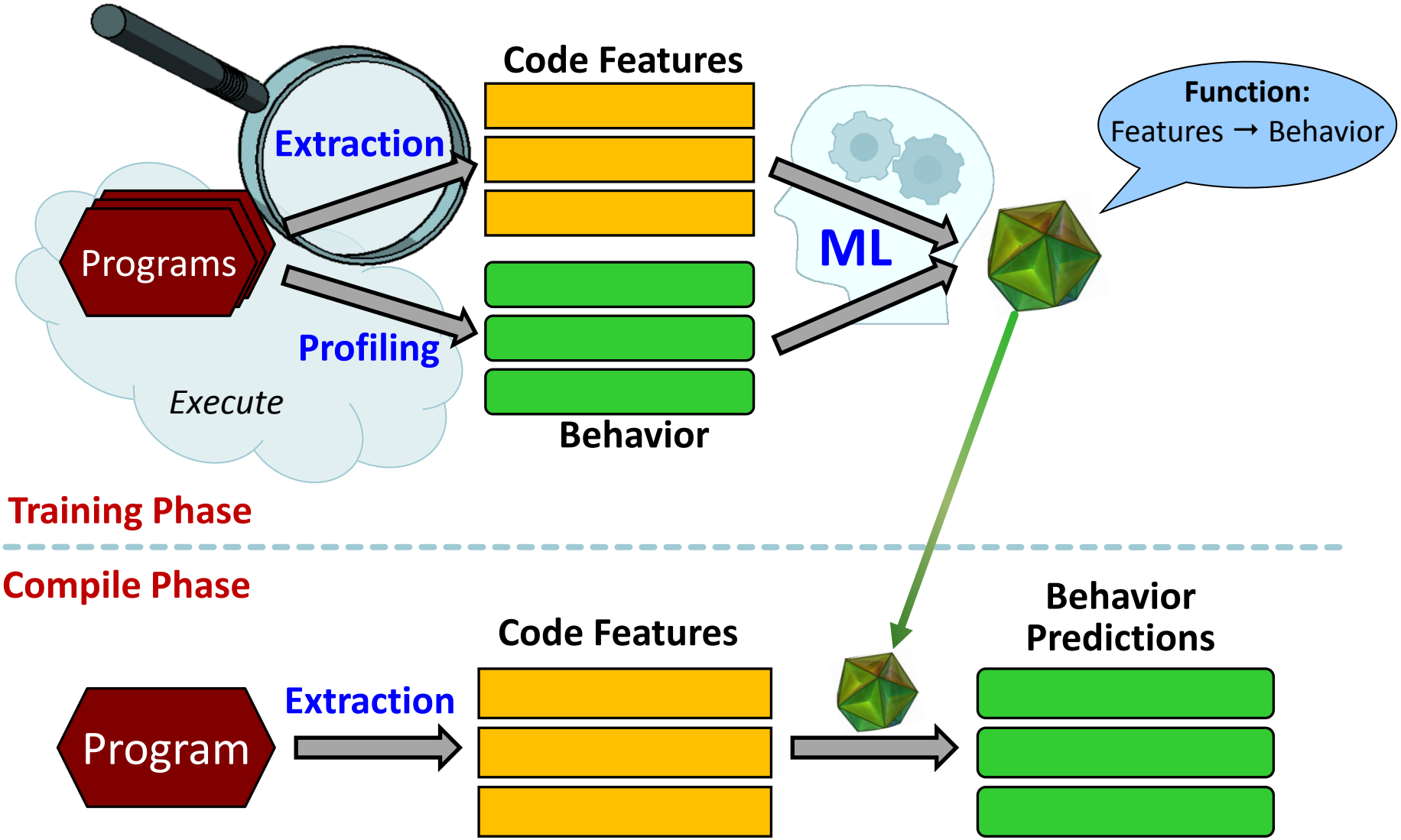
Predicting Recursion Depth

- ✗ Static analyses: over-approximate
- ✗ Profiling: strongly input data dependent and expensive
- ✗ WCET prediction, memory/stack analysis, HW-synthesis tools disallow recursion or require annotations[GE+06], [P99], [R06], [BG+06]
- Related Work
  - *Recursion Flattening*[SV08]:
    - ✗ Requires constant initialization and monotonic update
  - *Recursion Depth Analysis for Special Tree Traversal Algorithms*[KP84]
    - ✗ Size of planted planar tree must be known
- ➡ Use **Machine Learning** (ML)
  - More precise with knowledge of runtime behavior
  - Automatically generated and compile time not extended

Outline

- Learning Recursion Depth
  - Recap: ML-based Compilation
- Benchmarks
- Experiments
  - Results
- Conclusion

Recap: ML-based Compilation



Learning Recursion Depth

- Supervised classification learning
  - Runtime behavior discretized into classes
- 20 code features
  - Parameters, return values, variables
    - Structure (*tree* vs. *list* vs. *array*)
    - Size (*stack frame*)
  - Function body
    - Static number of self-calls *(fib* vs. *fac)*
    - Arithmetic operations (>> vs. / vs. –)
    - File-IO (*char* vs. *string*)

  weighted with static execution probability

Initial Benchmark Suites

- 153 programs from 12 benchmark suites
  - Used for learning loop iteration count[CiSE'10]
  - SPEC CPU{95,2000,2006}[1], NAS Parallel Benchmarks[2], SWEET WCET[3], BioBench[4], MediaBench II[5], cBench[6], LLCbench[7], LMbench[8], X Bench[9], Ptrdist[10]
- Comprise only 347 recursive functions (of >26.000)
  - Compared to 16.500 loops

Extended Benchmark Collection

- 395 programs from 31 benchmark suites
  - CSiBE[11], Bit Stream Benchmarks[12], Fhourstones Benchmark[13], FreeBench[14], GCbench[15], Heaplayers[16], llvm[17], MallocBench[18], McCat[19], MediaBench[20], MiBench[21], OldenBench[22], Phoronix Test Suite[23], Prolangs-C[24], Shoot[25], Splash2[26], Trimaran[27], UnixBench[28], Versabench[29]
- Now 890 recursive functions (of >70.000)

Empirical Study of Recursion

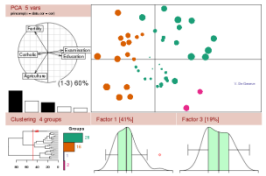
- Loops preferred programming style vs. recursion
  - 67.800 loops vs. 890 recursive functions (of >70.000)
- Recursion used for
  - media de-/encoding: *h263, h264, jpg, speech recognition*
  - compression: *bzip2*
  - string/tree traversal: *parser, interpreter, rsynth, anagram*
  - biology: *DNA/protein analysis, sequence alignment*
  - academic : *Fibonacci, Ackermann, FOL prover, N-body group motion, sorting, searching*
  - gaming: *chess, go*

Outline

- Learning Recursion Depth
  - Recap: ML-based Compilation
- Benchmarks
- Experiments
  - Results
- Conclusion

Experiments

- Compiler framework: *CoSy*
  - Feature extraction, static branch prediction, path profiling
- Machine Learning: *R Project*
  - *rpart* for predictor construction

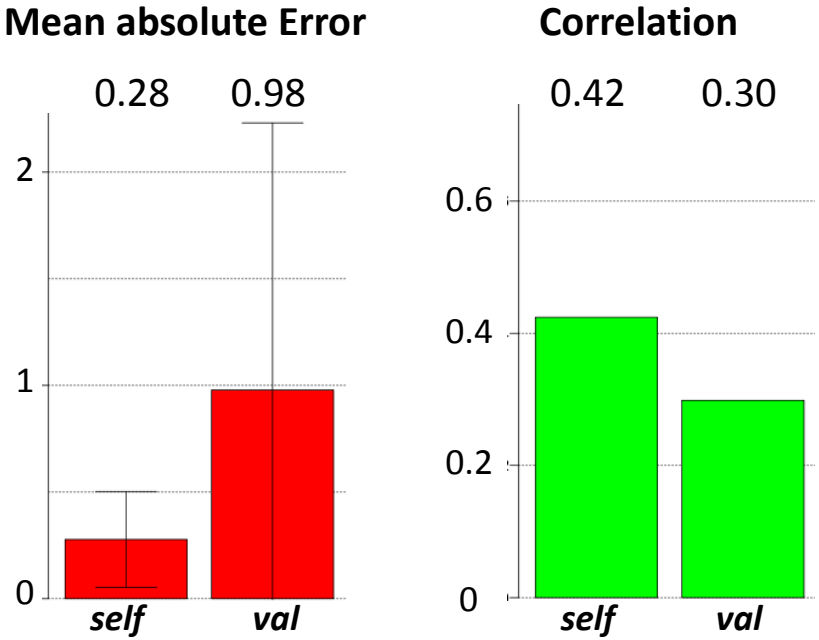


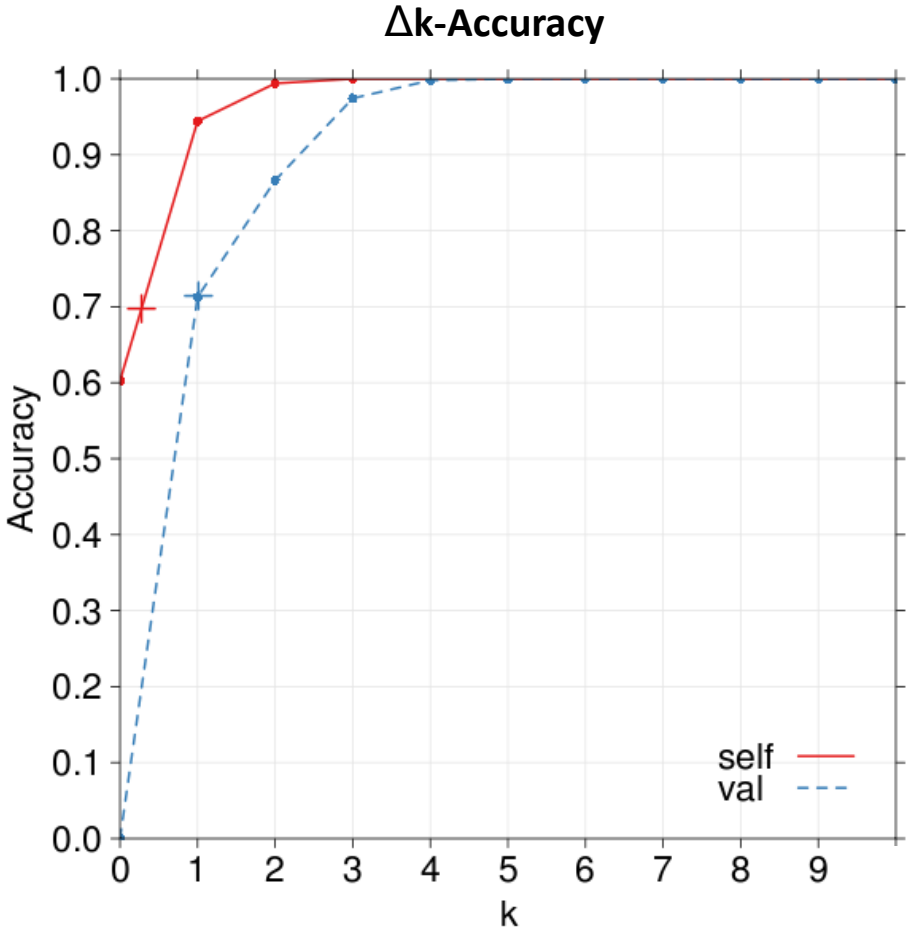
- 890 recursive functions analyzed
  - 424 actually recurse at least once
- Observed recursion depths: 1 – 300 million
- Recursion depth classified using truncated $\log_{10}$
  - 1 .. 9 ⇝ class 1
  - 10 .. 99 ⇝ class 2

  …
  - 100 million .. 999.999.999 ⇝ class 9

Experimental Results

- Self evaluation (*self*)
- Validation (*val*)
  - 331 functions used for learning
  - 93 functions used for prediction



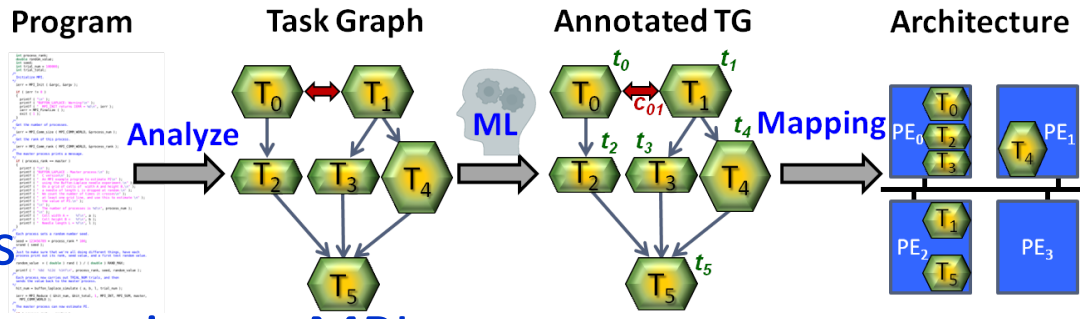


Outline

- Learning Recursion Depth
  - Recap: ML-based Compilation
- Benchmarks
- Experiments
  - Results
- Conclusion

Conclusion

- Predicting recursion depth via ML
- Huge collection of 31 benchmark suites
  - About 400 programs
  - Empirical study of recursion
- Experimental results
  - ✔ Precise prediction of runtime behavior (error < 1 class)
- Future Work
  - Task graph extraction
  - Learn execution times
  - Apply ML-based task mapping to MPI programs



Benchmark References

[1] Standard Performance Evaluation Corporation, 2011, http://www.spec.org/benchmarks.html#cpu.

[2] R. F. V. der Wijngaard, “NAS parallel benchmarks version 2.4,” NASA Ames Research Center, Moffett Field, CA, NAS Technical Report NAS-02-007, October 2002.

[3] J. Gustafsson, “The WCET tool challenge 2006,” in 2nd International Symposium on Leveraging Applications of Formal Methods (ISOLA’06), November 2007, pp. 248–249.

[4] K. Albayraktaroglu, A. Jaleel, X. Wu, M. Franklin, B. Jacob, C.-W. Tseng, and D. Yeung, “Biobench: A benchmark suite of bioinformatics applications,” in ISPASS 2005: IEEE International Symposium on Performance Analysis of Systems and Software, March 2005, pp. 2 –9.

[5] J. E. Fritts, F. W. Steiling, J. A. Tucek, and W. Wolf, “Mediabench II video: Expediting the next generation of video systems research,” Microprocess. Microsyst., vol. 33, no. 4, pp. 301–318, 2009.

[6] G. Fursin, “Collective tuning initiative: automating and accelerating development and optimization of computing systems,” in Proceedings of the GCC Developers’ Summit, June 2009.

[7] P. Mucci and K. S. London, “Low level architectural characterization benchmarks for parallel computers,” UT Computer Science, Tech. Rep. 394, July 1998.

[8] L. McVoy and C. Staelin, “lmbench: Portable tools for performance analysis,” in ATEC ’96: Proceedings of the 1996 annual conference on USENIX Annual Technical Conference. Berkeley, CA, USA: USENIX Association.

[9] T. M. Austin and SimpleScalar LLC, “X benchmarks,” 2004, http://www.simplescalar.com/.

[10] T. Austin, et al., The pointer-intensive benchmark suite, September 1995, http://pages.cs.wisc.edu/~austin/ptr-dist.html.

Benchmark References (2)

[11] Arṕ d Beszź des and P. Carlini, “CSiBE benchmark: One year perspective and plans,” Proceedings of GCC Developers’ Summit Ottawa, 2004.

[12] P. Gustafsson, “Bit stream benchmarks,” 2010, http://user.it.uu.se/~pergu/bitbench/index.html.

[13] J. Tromp, “The fhourstones benchmark,” 2010, http://homepages.cwi.nl/~tromp/c4/fhour.html.

[14] P. Rundberg and F. Warg, “The freebench v1.0 benchmark suite,” 2002, http://www.elsniwiki.de/index.php/Main/FreeBench.

[15] H. Boehm, “An artificial garbage collection benchmark,” 2010, http://www.hpl.hp.com/personal/Hans_Boehm/gc/gc_bench.html.

[16] E. D. Berger, B. G. Zorn, and K. S. McKinley, “Composing high-performance memory allocators,” in SIGPLAN Conference on Programming Language Design and Implementation, 2001, pp. 114–124.

[17] LLVM Team, “LLVM Testing Infrastructure,” 2011, http://llvm.org/docs/TestingGuide.html.

[18] D. Grunwald, B. Zorn, and R. Henderson, “Improving the cache locality of memory allocation,” in Proceedings of the ACM SIGPLAN 1993 conference on Programming language design and implementation, ser. PLDI ’93. New York, NY, USA: ACM, 1993, pp. 177–186.

[19] M. Hind and A. Pioli, “Assessing the effects of flow-sensitivity on pointer alias analyses,” in Proc. of the 5th International Symposium on Static Analysis. London, UK: Springer-Verlag, 1998, pp. 57–81.

[20] C. Lee, M. Potkonjak, and W. H. Mangione-Smith, “Mediabench: a tool for evaluating and synthesizing multimedia and communicatons systems,” in 30th ACM/IEEE symposium on Microarchitecture, ser. MICRO 30. Washington, DC, USA: IEEE Computer Society, 1997, pp. 330–335.

Benchmark References (3)

[21] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, “Mibench: A free, commercially representative embedded benchmark suite,” in IEEE International Workshop on Workload Characterization, 2001, pp. 3–14.

[22] M. C. Carlisle and A. Rogers, “Software caching and computation migration in Olden,” in Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming, ser. PPOPP ’95. New York, NY, USA: ACM, 1995, pp. 29–38.

[23] M. Larabel, “Phoronix Test Suite 3.0 Iveland, An Automated, Open-Source Testing Framework,” 2011, http://www.phoronix-test-suite.com/.

[24] W. Landi and B. G. Ryder, “Programming Languages Research Group, The State University of New Jersey,” 1997, http://prolangs.cs.vt.edu/rutgers/software.php.

[25] B. Fulgham, “Revived version of Bagley’s Great Computer Language Shootout benchmarks,” 2004, http://shootout.alioth.debian.org/.

[26] S. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in Proc. of the 22nd Int’l Symposium on Computer Architecture, June 1995.

[27] R. Rabbah and N. Clark, “An infrastructure for research in backend compilation and architecture exploration,” 2007, CCCP group at Georgia Institute of Technology and IBM Research, http://www.trimaran.org/index.shtml.

[28] I. Smith, “Revised BYTE UNIX benchmark suite,” 2007, http://code.google.com/p/byte-unixbench/.

[29] R. M. Rabbah, I. Bratt, K. Asanovic, and A. Agarwal, “Versatility and versabench: A new metric and a benchmark suite for flexible architectures,” Computer Science and Artifical Intelligence Laboratory Massachusetts Institute of Technology Cambridge, MA 02139, Tech. Rep., 2004.

References

[SV08] Greg Stitt and Jason Villarreal, *"Recursion Flattening"*, Proceedings of the 18th ACM Great Lakes symposium on VLSI, ACM, 2008, p. 131-134

[KP84] P. Kirschenhofer and H. Prodinger. *"Recursion Depth Analysis for Special Tree Traversal Algorithms"*, Automata, Languages and Programming (ICALP), LNCS, 11th Colloquium Antwerp, Belgium, July 16–20, 1984

[GE+06] J. Gustafsson, A. Ermedahl, C. Sandberg, and B. Lisper. *"Automatic Derivation of Loop Bounds and Infeasible Paths for WCET Analysis Using Abstract Execution"*, IEEE International Real-Time Systems Symposium, IEEE Computer Society, 2006, pp. 57-66

[P99] Patrik Persson. *"Live memory analysis for garbage collection in embedded systems"*, Proceedings of the ACM SIGPLAN 1999 workshop on Languages, compilers, and tools for embedded systems, ACM, 1999, pp. 45-54

[R06] D. Ramakrishna Rao, *"Efficient stack sizing for very large software systems"* International Conference on Computing Informatics (ICOCI'06), 2006, 1 -10

[BG+06] B. Buyukkurt, Z. Guo, and W. Najjar, *"Impact of Loop Unrolling on Area, Throughput and Clock Frequency in ROCCC: C to VHDL Compiler for FPGAs"* Reconfigurable Computing: Architectures and Applications, Springer Berlin / Heidelberg, 2006, 3985, 401-412