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]

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

![Diagram of task mapping process](image)
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 \cite{GE06, P99, R06, BG06}

**Related Work**

- **Recursion Flattening** \cite{SV08}:
  - Requires constant initialization and monotonic update
- **Recursion Depth Analysis for Special Tree Traversal Algorithms** \cite{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

**Training Phase**
- **Programs**
- **Extraction**
- **Profiling**
- **Execute**

**Compile Phase**
- **Function:** Features → Behavior
- **ML**
- **Behavior Predictions**

**Code Features**
- Features
- Behavior
- Predictions
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\textsuperscript{[CiSE’10]}
  - SPEC CPU\{95,2000,2006\}\textsuperscript{[1]}, NAS Parallel Benchmarks\textsuperscript{[2]}, SWEET WCET\textsuperscript{[3]}, BioBench\textsuperscript{[4]}, MediaBench II\textsuperscript{[5]}, cBench\textsuperscript{[6]}, LLCbench\textsuperscript{[7]}, LMbench\textsuperscript{[8]}, X Bench\textsuperscript{[9]}, Ptrdist\textsuperscript{[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

<table>
<thead>
<tr>
<th>Mean absolute Error</th>
<th>Correlation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.28</td>
<td>0.42</td>
</tr>
<tr>
<td>0.98</td>
<td>0.30</td>
</tr>
</tbody>
</table>

Δk-Accuracy

Accuracy vs. k

Self (solid red) vs. Val (dashed blue)
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

Benchmark References (2)


Benchmark References (3)


References


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