Detecting Variable-Length Phases in Utility Programs

Xipeng Shen, Chen Ding, Sandhya Dwarkadas, and Michael L. Scott
Department of Computer Science, University of Rochester

UNIVERSITY OF ROCHESTER
COMPUTER SCIENCE

INTRODUCTION

Utility Programs

- Take a sequence of requests as inputs.
- The service to the requests composes the kernel computation; the behavior highly depends on the request.
- The behavior variations give challenges to behavior characterizations.
- Examples:
  - GCC (each input function is a request)
  - Compress, Interpreter, etc.

Goals

- To recognize behavior patterns of utility programs and then detect phase boundaries (i.e. the boundaries of behavior patterns).

Motivations

- Program and hardware adaptation is becoming increasingly important.
- The benefit of the adaptation depends on the accuracy of phase partitioning and behavior prediction.
- Utility programs pose big challenges to previous techniques.
  - The irregular behavior due to their high dependence on the inputs.
  - Previous online methods cannot afford sophisticated analysis.
  - Previous offline methods don't apply to utility programs due to the irregular behavior (Shen etc. ASPLOS'04).

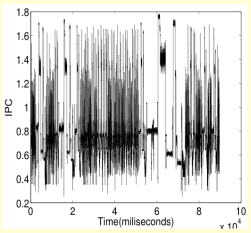
TECHNIQUES

Technique Overview

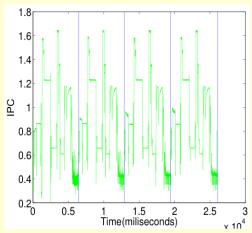
- Through *active profiling*, we induce perfect behavior repetitions in utility programs.
- Then we find the phase boundaries through the *filtering* of dynamic basic block traces based on frequencies and distances.

Adaptive Profiling: Convert Challenges to Opportunities

- Taking advantage of input-dependent behavior, we use regular inputs to induce regular behavior repetitions.
- Example: feed GCC with a C program containing 4 identical functions with different function names. Figure (a) shows the irregular IPC curve of a ref execution; Figure (b) shows the IPC curve of the execution on a regular input with vertical blue lines as the boundaries of outermost phases.



(a) IPC Curve of GCC on a ref input (*scilab*)



(b) IPC Curve of GCC on a regular input

Finding Phase Markers

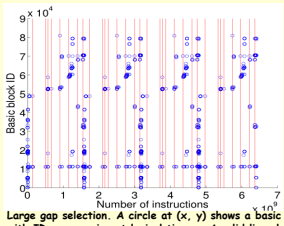
- Checking for regularity (filtering on frequency and recurring patterns)
  - Profile execution on the regular input with f identical requests.
  - Only record those basic blocks that appear exactly f times.
  - Remove noisy blocks based on recurring patterns.
    - A tuple to represent the recurring patterns of a basic block.

$$(\bar{r}_b ,\ \delta_{r_b})$$

$\bar{r}_b$: the average recur-distance of block b.

δ_{r_b} : the standard deviation of the recur-distance of block b.

- The outliers of the tuple set are considered noisy blocks.
- Checking for consistency
  - Rerun the application on one or more real (irregular) inputs.
  - Record the blocks appearing as many times as the number of requests.
  - Take the intersection between this set of blocks and the one from the regularity check.
  - Find the boundaries of the large gaps in the temporal sequence of the remaining blocks as phase boundaries: selecting the outliers of the gap size.



Large gap selection. A circle at (x, y) shows a basic block with ID y appearing at logical time x. A solid line shows the boundary of a large gap, whose size is an outlier of all the gaps'

EVALUATION

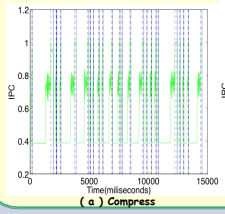
Methodology

- Benchmarks

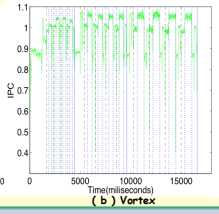
Benchmark	Description	Source
Compress	common UNIX compression utility	SPEC95Int
GCC	GNU C compiler 2.5.3	SPEC2kInt
LI	Xlisp interpreter	SPEC95Int
Parser	natural language parser	SPEC2kInt
Vortex	object oriented database	SPEC2kInt

- In the analysis stage, we use ATOM for instrumentation on a Digital Alpha machine.
- For evaluation, we use PMAPI on an IBM POWER4 machine for hardware performance measurement.

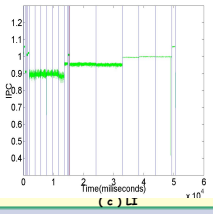
Evaluation on Other Benchmarks



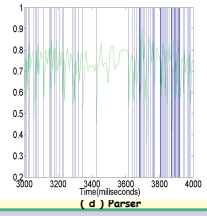
(a) Compress



(b) Vortex



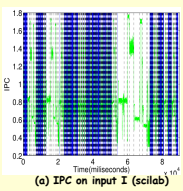
(c) LI



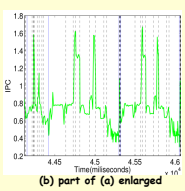
(d) Parser

Evaluation on GCC

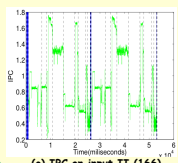
- The source code has 120 files and 222182 lines of C code.
- The outermost phase boundary is found at the start of the compilation of an input function.
- Eight inner phases show different compilation stages.
- Some phase boundaries do not coincide with loop or function boundaries.
- With phase markers, regularity emerges from visually irregular behavior curve.



(a) IPC on input I (scilab)



(b) part of (a) enlarged

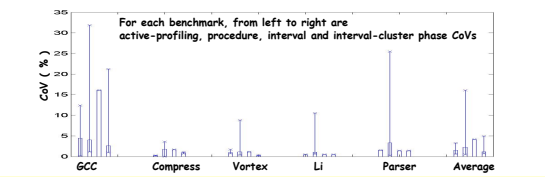


(c) IPC on input II (166)

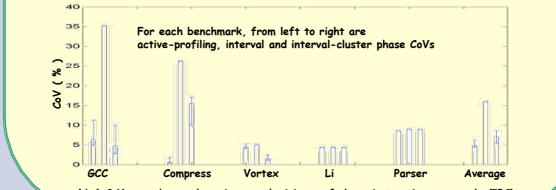
- Blue solid vertical lines show outermost phase boundaries; blue broken vertical lines show inner phase boundaries.
- Phase behavior regularity emerges as the consistency of phase IPC patterns across phase instances inside and across the figures.

Comparison with Procedure and Interval Phases

- Coefficient of Variance (CoV) is the standard deviation divided by the mean.
- Use the IPC and cache miss CoVs of the instances of a phase to measure the consistency of the phase behavior.
- Comparison of Different Approaches:
  - Active-profiling : our method.
  - Procedure : taking important subroutines as phases [Maglis etc. ISCA'03].
  - Interval : fixed-length intervals as phase instances.
  - Interval-cluster : the upper-bound of interval-based methods, which clusters intervals according to cache hit rate or IPC directly.



(a) CoV mean bar and maximum and minimum of phase cache hit rates



(b) CoV mean bar and maximum and minimum of phase instructions per cycle (IPC)

Uses in Memory Management

- A phase, especially the outermost phase, often represents a memory usage cycle, in which temporary data are allocated and then collected.
- Monitor and predict memory demand trend by measuring memory usage at phase boundaries.
- Categorize objects as phase local or global, helping leak detection.
- Optimize garbage collection by invoking GC at phase boundaries.
  - Reduce execution time by 44% on Xlisp.

Conclusions

- A two-step technique for detecting input-dependent phase behavior in utility programs.
  - Active-profiling induces repeating behavior.
  - Examines all program statements to identify phase boundaries that recur regularly in frequency and distance in the regular input and recur consistently in real inputs.
- Compared with procedure-based and interval-based methods, the technique often captures and predicts program behavior at larger granularity with higher accuracy, using NO thresholds.