Dynamic Optimization of Interpreters – "JIT's for Free!"

Gregory T. Sullivan, Saman Amarasinghe & Iris Baron

Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

http://www.ai.mit.edu

The Problem: Programming language interpreters (which include virtual machines) are easier to implement than source-to-native compilers, and also offer advantages of compactness and portability. Unfortunately, interpreters suffer from seemingly unavoidable runtime overhead. Getting rid of that overhead has traditionally involved large, expensive reimplementation efforts, affordable by only the largest development organizations. Our research is aimed at providing much of the benefit of "Just In Time" (JIT) native compilation with only a small development cost.

Motivation: Suppose your interpreter compiles source programs into some intermediate representation, such as a vector of byte codes. To run an application, after it has been compiled to byte codes, the interpreter must fetch each byte code in turn and then dispatch to the appropriate handler for that byte code. This so-called "interpretive overhead" can swamp the actual work being done for the individual instructions. There have been many clever techniques (e.g., [4]) to speed up bytecode processing, but the difference between interpreted and "native" code is still huge.

Writing a source-to-native compiler to provide better performance than an interpreter is a daunting task for a small implementation group, and is made significantly more difficult when trying to implement the dynamic, object-oriented features the Dynamic Languages Group (`http://www.ai.mit.edu/projects/dynlangs/`) is interested in.

Previous Work: The most promising approach to improving the performance of interpreted languages is called "Just In Time" (JIT) compilation, exemplified by [1] and [2]. The idea is that when a sequence of bytecodes is selected for execution (or identified as "hot" by profiling), the bytecodes are translated to native instructions, which are cached for later reference, and then executed. This approach maintains the compactness and portability aspects of bytecode interpreted languages, while approaching the speed of natively compiled applications.

Unfortunately, writing a JIT compiler is extremely difficult, and requires resources typically only available to a few organizations, such as IBM, Sun, or Microsoft.

Approach: The *Dynamo* [3] system from Hewlett-Packard Research Labs is a native dynamic optimization tool. Dynamo intercepts native code before it executes, optionally rewrites it, instruments the block exits to return to Dynamo or jump to other cached blocks, caches the code block, and jumps to it. By "straightening" control flow of common paths, Dynamo can actually improve the performance of some applications. Figure 1 gives an overview of the Dynamo system.

Unfortunately, Dynamo's optimization strategy of finding a common path through hot loops is foiled by the structure of the main loop in an interpreter. By design, the fetch-decode-dispatch loop in an interpreter has many different paths through it (typically one per bytecode opcode).

However, consider the execution of an interpreted application in terms of an *abstract PC* consisting of the pair (native PC, logical PC). In our example, the logical PC is an offset into the bytecode vector. A given *logical trace* will include many trips to and from the interpreter's fetch-decode-dispatch loop. Once the trace is recorded, starting at some abstract PC, it can be statically optimized. In particular, each sequential bytecode dereference can be replaced by a constant, and constant propagation extended to partial evaluation can remove an enormous amount of the original interpreter overhead.

Impact: There are many, many interpreted language implementations, some of which are quite popular (Perl, Python, etc.). Indeed, many applications other than programming language implementations can be viewed ab-

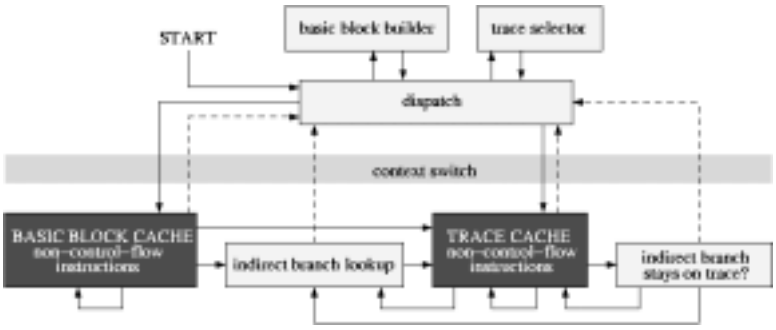

Figure 1: Flow Chart of the Dynamo System

stractly as interpreters: they interpret data (one kind of data is bytecodes) by performing different actions based on different data. All of these applications can benefit from some variant of the JIT compilation approach.

Future Work: We are in the initial phases of this research, and over the next year we will apply it to ever more realistic interpreters. We hope to have some concrete results by late Fall 2002.

Research Support: This research is supported by a grant from the HP-MIT Alliance.

References:

[1] The java hotspot ™ virtual machine technical white paper, 2002. Online at java.sun.com / products / hotspot / .

[2] Matthew Arnold, Stephen J. Fink, David Grove, Michael Hind, and Peter F. Sweeney. Adaptive optimization in the jalapeno JVM. In *Conference on Object-Oriented Prorgramming and Systems*, pages 47–65, 2000.

[3] Vasanth Bala, Evelyn Duesterwald, and Sanjeev Banerjia. Dynamo: a transparent dynamic optimization system. *ACM SIGPLAN Notices*, 35(5):1–12, 2000.

[4] M. Anton Ertl. A portable Forth engine. In *EuroFORTH '93 conference proceedings*, Mariánské Láznè (Marienbad), 1993.