A Good Way to Exploit Predictable Control Flow? Vectorize Programs at Runtime!

Sriram Vajapeyam
Indian Institute of Science
Bangalore, INDIA 560012
sriram@csa.iisc.ernet.in; http://www.csa.iisc.ernet.in/~sriram

The Problem

Current processors rely on extensive compiler optimizations and significant hardware-based runtime optimizations of programs for achieving high performance. The significant potential benefits of exploiting program behavior information available only at runtime is driving research in hardware and software schemes for dynamic optimization. Hardware-based approaches can suffer from hardware complexity and a relatively small window of program execution visible for optimization. Software-based runtime schemes can suffer from significant overheads that might swamp any gains from optimizations. At the same time, research continues to show large amounts of program locality, even with regard to second-level program characteristics such as the *degree of* predictability (or, prediction *confidence levels*) of instruction dependences, instruction operand & result values, etc. Thus, low overhead, low complexity approaches to exploiting dynamic program behaviour seem critical for future high-performance processors.

The Solution

We propose the novel use of a highly successful and mature hardware approach, *vector processing*, to exploit runtime control-flow locality that may not be visible or predictable at compile time. Previously, only programs amenable to static, compile-time vectorization could take advantage of vector processing. We observe that history-based branch prediction schemes achieve high prediction accuracies even for programs that are not easily statically vectorizeable. Intuition suggests that the execution profiles of such programs are likely to reveal repetitive execution of a single flow of control in loops, even for loops that contain spaghetti static code with numerous control paths. Indeed, examination of the SPECInt92 benchmarks shows that a significant fraction of the dynamic instructions of several integer programs can be converted to vector form by a dynamic scheme.

We propose simple Dynamic Vectorization (DV) hardware that can dynamically detect a repetitively-executing code sequence and capture it in vector form. Relatively simple and mature register renaming hardware is adapted to map operands of vectorized instructions to vector queues. Simple extensions to this hardware correctly captures cross-iteration instruction dependences (recurrences) as well as dependences between the vectorized code and the rest of the program. Support for speculative handling of the memory dependences of vectorized instructions seems possible with hardware of the same complexity and size as in superscalar processors.

Tbe Benefits of Dynamic Vectorization

All the traditional benefits of vectors accrue with dynamic vectorization. Important benefits include:

(i) Overlap of post-loop code processing with the vectorized processing of the loop, thus boosting ILP. Several ILP limit studies have emphasized the presence of larger amounts of ILP across far-apart instructions in program execution.

(ii) Capture of a large logical program segment in a relatively small hardware instruction window of the processor, since multiple iterations of a loop are captured by a single physical copy of a vector code sequence. This alleviates several well known problems with large physical instruction windows.

(iii) From the real-world viewpoint, dynamic vectorization provides binary compatibility for performance. Also, application domains such as multimedia that currently rely on hand-vectorization can benefit from automatic dynamic vectorization.

(iv) From the processor implementation viewpoint, vector instruction processing is a well-understood paradigm. This allows processor implementations to leverage previous experience and knowledge base when it comes to implementing DV.