Link-Time Optimization of IA64 Binaries

Bertrand Anckaert, Frederik Vandeputte, Bruno De Bus, Bjorn De Sutter, and Koen De Bosschere

Ghent University, Electronics and Information Systems Department
Sint-Pietersnieuwstraat 41 9000 Gent, Belgium
{banckaer, fgvdeput, bdebus, brdsutte, kdb}@elis.UGent.be

Abstract. The features of the IA64 architecture create new opportunities for link-time optimization. At the same time they complicate the design of a link-time optimizer. This paper examines how to exploit some of the opportunities for link-time optimization and how to deal with the complications. The prototype link-time optimizer that implements the discussed techniques is able to reduce the code size of statically linked programs with 19% and achieves a speedup of 5.4% on average.

1 Introduction

On the EPIC (Explicitly Parallel Instruction Computer) platform, the compiler determines which instructions should be executed in parallel. This responsibility corresponds to the belief that better performance can be achieved by shifting the parallelism extraction task from hardware (as in superscalar out-of-order processors) to the compiler: the hardware becomes less complex and the compiler can exploit its much wider view on the code [8].

Unfortunately compilers only have a fragmented program view: most compilers compile and optimize all source code files independently of each other. Even when all source code is compiled together, the libraries are still compiled separately, and hence not optimized for any specific program. The resulting lack of compile-time whole-program optimization is particularly bad for address computations: As the linker decides on the final program layout in memory, code and data addresses are not known at compile time. The compiler therefore has to generate *relocatable* code, which is most often far from optimal.

Optimizing linkers try to overcome these problems by adding a link-time optimization pass in the tool chain. Optimizing linkers take compiled object files and precompiled code libraries as input, and optimize them together to produce smaller or faster binaries. In this paper we present our link-time optimizer for the IA64 architecture. Our main contributions are:

- We extend the existing work on Global Offset Table optimizations by creating a second global pointer at link-time.
- We show how existing link-time liveness analysis can be adapted to deal with the rather peculiar register files of the IA64 architecture.
- We demonstrate how the set of branch registers can be exploited more effectively with whole-program optimization.