Function Level Parallelism Driven by Data Dependencies

Sean Rul Hans Vandierendonck Koen De Bosschere

Department of Electronics and Information Systems (ELIS),
Ghent University, Sint-Pietersnieuwstraat 41, 9000 Gent, Belgium
Email: {srul, hvdieren, kdbosche}@elis.ugent.be

Abstract

With the rise of Chip multiprocessors (CMPs), the amount of parallel computing power will increase significantly in the near future. However, most programs are sequential in nature and have not been explicitly parallelized, so they cannot exploit these parallel resources. Automatic parallelization of sequential, non-regular codes is very hard, as illustrated by the lack of solutions after more than 30 years of research on the topic. The question remains if there is parallelism in sequential programs that can be detected automatically and if so, how much parallelism there is.

In this paper, we propose a framework for extracting potential parallelism from programs. Applying this framework to sequential programs can teach us how much parallelism is present in a program, but also tells us what the most appropriate parallel construct for a program is, e.g. a pipeline, master/slave work distribution, etc.

Our framework is profile-based, implying that it is not safe. It builds two new graph representations of the profile-data: the interprocedural data flow graph and the data sharing graph. This graphs show the data-flow between functions and the data structures facilitating this data-flow, respectively.

We apply our framework on the SPECcpu2000 bzip2 benchmark, achieving a speedup of 3.74 of the compression part and a global speedup of 2.45 on a quad processor system.

1 Introduction

Today we are at the dawn of a new era in computer architecture. While during the past decade the computer scene was mainly dominated by complex uniprocessors with deep pipelines, we now see the rise of CMPs containing several slimmer cores. This transition was fueled by several incentives. Firstly, the instruction level parallelism (ILP) has been exploited to its full extent, such that extracting more ILP becomes overly complex. Secondly, power dissipation kept increasing alarmingly, caused by implementing a single large core with long wires and clocked at high frequencies.

While this transition was necessary, we are also confronted with new issues. A big question is how we can exploit all this parallel processing power in the new processor generation? Although there is a group of programs, such as scientific and media applications, that inherently have a lot of easily exploitable thread-level parallelism (TLP), another majority of programs are inherently sequential. These programs, exemplified by the SPECcpu integer benchmark suite, have remained out of scope of research on parallelization up to the last few years.

Automatically parallelizing inherently sequential programs is hard due to the difficulty of correct static analysis of both control flow and data flow. To extract large amounts of thread-level parallelism, it is necessary to look past these limiting control flow and data flow restrictions. In this paper, we develop a framework for extracting thread-level parallelism from sequential programs that assumes perfect knowledge of these dependencies. As such, it is able to discover large amounts of TLP.

Our goal is to discover nonspeculative parallelism in the first place, without being restricted by the unpredictability of control flow and data flow. Hereto, we measure the control flow and data flow exhibited during a particular run of the program. Analysis of this control flow shows opportunities for parallelization. However, by measuring dependencies during one or more particular runs of a program, the parallelism extracted by our framework may be unsafe, i.e., particular dependencies may arise depending on the input data set of the program. This situation can be handled using, e.g. thread-level speculation (TLS) systems [9, 14] that allow dependencies to be violated, but detects and corrects them with the aid of hardware support. It is also possible to use our framework as an analysis tool for a parallel programmer, who can use it to detect parallelism, but still needs to validate their correctness.

Our analysis focusses on memory dependencies, since register dependencies can be predicted [15] or precom-