

Compiler Generated Multithreading to Alleviate Memory Latency

Kristof E. Beyls and Erik H. D'Hollander
(Dept. of Electronics and Information Systems
University of Ghent, Belgium
{kbeyls,dhollander}@elis.rug.ac.be)

Abstract: Since the era of vector and pipelined computing, the computational speed is limited by the memory access time. Faster caches and more cache levels are used to bridge the growing gap between the memory and processor speeds. With the advent of multithreaded processors, it becomes feasible to concurrently fetch data and compute in two cooperating threads. A technique is presented to generate these threads at compile time, taking into account the characteristics of both the program and the underlying architecture. The results have been evaluated for an explicitly parallel processor. With a number of common programs the data-fetch thread allows to continue the computation without cache miss stalls.

Key Words: data locality, multithreading, run-time data relocation, compiler optimization, cache optimization, prefetching, tiling

1 Introduction

The well known Von Neumann bottleneck has hampered the unbridled development of shared memory multiprocessors. New parallel programming paradigms, architectures and interconnection networks with huge bandwidth have increased the computing power, but the memory latency remains the limiting factor of the computation. Whereas the processor performance increases by about 60%/year, the memory performance increases only by about 7%, leading to a relative performance drop of 2 every 21 months. In the beginning, a single cache level sufficed to dampen the effect of slow memory access, but the widening gap has led to a hierarchy of caches. This cache hierarchy limits the damage, but clearly the memory latency is important when cache misses occur. It is not uncommon that a processor stalls for memory accesses more often than doing useful computations. With more data-hungry superscalar and multithreaded processors, this trend is not likely to change.

To tighten the processor-memory gap, several hardware and software techniques have been proposed to use the cache hierarchy more effectively and to hide the memory latency:

1. Software techniques which reorder the program loop structure in order to promote data locality and reuse, such as loop permutation, reversal, fusion, distribution[McKinley *et al.*1996] and tiling[Lam *et al.*1991].

The remainder of this paper is not included as this paper is copyrighted material. If you wish to obtain an electronic version of this paper, please send an email to bib@elis.UGent.be with a request for publication P100.120.pdf.
