Chapter 4

A Multidimensional Software Cache for Scratchpad-Based Systems

Arnaldo Azevedo
Delft University of Technology, The Netherlands

Ben Juurlink
Technische Universität Berlin, Germany

ABSTRACT

In many kernels of multimedia applications, the working set is predictable, making it possible to schedule the data transfers before the computation. Many other kernels, however, process data that is known just before it is needed or have working sets that do not fit in the scratchpad memory. Furthermore, multimedia kernels often access two or higher dimensional data structures and conventional software caches have difficulties to exploit the data locality exhibited by these kernels. For such kernels, the authors present a Multidimensional Software Cache (MDSC), which stores 1-4 dimensional blocks to mimic in cache the organization of the data structure. Furthermore, it indexes the cache using the matrix indices rather than linear memory addresses. MDSC also makes use of the lower overhead of Direct Memory Access (DMA) list transfers and allows exploiting known data access patterns to reduce the number of accesses to the cache. The MDSC is evaluated using GLCM, providing an 8% performance improvement compared to the IBM software cache. For MC, several optimizations are presented that reduce the number of accesses to the MDSC.

INTRODUCTION

Most processors use a cache to overcome the memory latency. Some processors, however, employ software-controlled high-speed internal memories or scratchpad memories to exploit locality. Processors based on scratchpad memories are very efficient in terms of power and performance (Banakar et al., 2002). The power efficiency is due to the simple structure of the memory compared to caches. Scratchpad memories also have predictable latencies. These characteristics
make scratchpad memories a common choice for embedded processors.

Many kernels (e.g., multimedia kernels) have a working set that is predictable, which makes it possible to transfer data before the computation. It is often also possible to overlap computation with data transfers by means of a double buffering technique, where the data in one buffer is processed while the data for the next processing stage is fetched in another buffer. In scratchpad-based systems these data transfers usually need to be explicitly programmed using Direct Memory Access (DMA) requests. There are also many multimedia kernels, however, that process data that is known just before it is needed. This is the case, for instance, in the Motion Compensation (MC) kernel of H.264 video decoding. Only after Motion Vector Prediction it is possible to fetch the data necessary to reconstruct the frame. Other kernels have working sets that exceed the capacity of the scratchpad memory. This is the case in the Gray Level Co-occurrence Matrix (GLCM) kernel. It features a relatively random access that renders DMA requests for each individual access impractical. MC is an interesting kernel as its memory access pattern is similar to other important multimedia kernels such as texture mapping. GLCM features a fine-grain random access pattern that is representative of other tabulation algorithms, such as histogram.

Both kernels exhibit data locality that could be exploited by a cache. In MC the motion vectors are often closely related so that data that is (logically) adjacent to the reference area is needed to decode the next macroblock (MB). In GLCM the difference of adjacent pixels is often small so that the kernel accesses small parts of the GLCM matrix. In a scratchpad memory a cache can be emulated. This is often referred to as a software cache. Software caches, however, incur high overhead, representing up to approximately 50% (Gonzalez et al., 2008) of the total application execution time. Such high overheads could harm performance compared to hand-programmed, just-in-time DMA transfers. It is therefore necessary to reduce the number of cache accesses as much as possible. An additional feature of these as well as many other multimedia kernels is that they access 2- or higher-dimensional data structures and adjacent sub-rows are not consecutive in memory.

For such kernels we propose a Multidimensional Software Cache (MDSC). The MDSC stores 1- to 4-dimensional blocks (sub-matrices) and the cache is indexed by the matrix indices rather than a linear memory address. This approach both minimizes the memory transfer time and the number of cache accesses. The first is achieved by grouping memory requests, thereby reducing the overhead associated with memory requests. The latter is achieved by exploiting the multidimensional access behavior of the application.

Our experimental platform is the Cell processor. Implementing a software cache for the Cell processor is an active research topic (Balart et al., 2007; Lee et al., 2008; Chen et al., 2008). Balart et al. (2007) propose a compile time software cache with support for asynchronous transfers. The compiler uses asynchronous transfers to overlap memory transfers with computation. They report a speedup of 1.26 to 1.66 over synchronous transfers. Chen et al. (2008) propose a similar approach with support for runtime prefetching based on the access patterns. These works are complementary to our work since the MDSC can be used as the software cache implementation for the compiler.

The current version of the MDSC does not feature cache coherency. Currently it is not needed because either only read-only data is cached or efficient multicore kernel implementation avoids the need for coherency. However, cache coherency is an important feature for caches in a multicore environment. Lee et al. (2008) and Seo et al. (2009) propose a coherent shared memory interface for the Cell BE using software caches. It employs a software cache in the local store for page-level caching. It guarantees coherency at the page level and uses centralized lazy release coherency.