

A Prefetching Memory System for Mediaprocessors

Stefan G. Berg¹, Donglok Kim² and Yongmin Kim³

¹Department of Computer Science and Engineering

²SVision LLC, Bellevue, WA

³Departments of Electrical Engineering and Bioengineering

9:35 a.m. HUB209A

In this talk, I will describe a high-performance prefetching memory system designed for mediaprocessors. This is work I did together with my advisor Dr. Yongmin Kim and Dr. Donglok Kim, a former faculty member from Electrical Engineering who is now working at SVision.

Outline

- Mediaprocessors
 - Overview
 - Direct memory access (DMA) controllers
 - Cache controllers
- Program-directed prefetcher
- No-write-allocate write-miss cache policy
- Results
- Conclusions

Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

2

I will begin with a brief overview of mediaprocessors, specifically focusing on their memory system that is usually made up of a DMA controller or a cache controller. Then I will introduce the program-directed prefetcher that forms a central component of our high-performance memory system. In addition, we use a no-write-allocate write-miss cache policy that I will briefly explain. And finally, I have results and conclusions.

Mediaprocessor Overview

- Programmable processor
 - Flexible
 - Cost-effective (~ US\$50)
 - High-performance
- Multimedia processing
 - Set-top boxes, DVD players, printers, camcorders, HDTV, security devices, camera on a chip, etc.
- Examples: Hitachi/Equator Technology MAP-CA, Texas Instruments TMS320C64x, TriMedia Technologies TM-64
- Likely to be a dominant choice for future embedded market

Feb. 26th, 2002

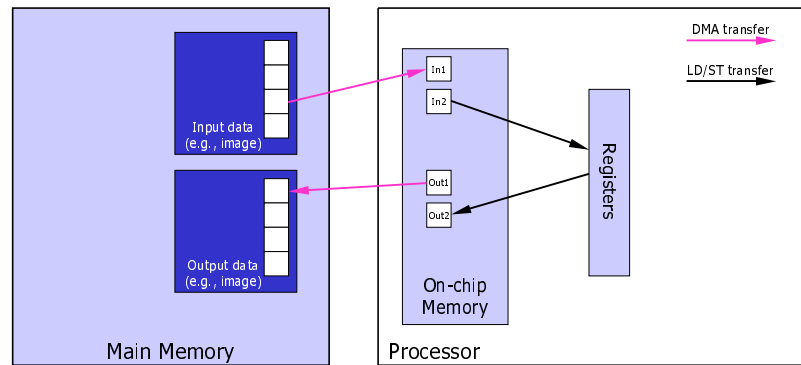
A Prefetching Memory System for Mediaprocessors

3

What is a mediaprocessor? We see mediaprocessors as being able to become a dominant choice for the embedded market when high performance is required. For example, in DVD players, camcorders, and high definition television sets. They can be used in security cameras, for example performing facial recognition at public events. Mediaprocessors are programmable and therefore intended to be more flexible than ASICs. Compared to general-purpose processors, they are more cost-effective and can provide higher performance on multimedia applications. Mediaprocessors, such as Hitachi and Equator's MAP-CA and Texas Instruments C64, typically contain multiple wide execution units organized in a very long instruction word. They contain partition function units to be able to process multiple sets of data simultaneously and a high-bandwidth memory systems built using either a DMA controller or a cache controller, sometimes both.

Since the focus of my talk is the memory system, I want to begin with a brief introduction of how DMA controllers are used to efficiently transfer data between main memory and processor.

Double Buffering with DMA



Feb. 26th, 2002

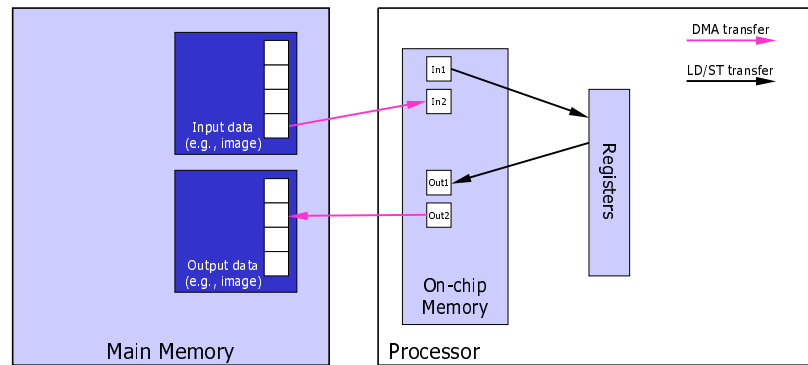
A Prefetching Memory System for Mediaprocessors

4

DMA controllers are typically used in combination with an addressable on-chip memory, although it is also possible to combine them with an on-chip cache. When using an addressable on-chip memory, the processor's load store units are used for transferring data between on-chip memory and registers as indicated by the black arrows on the slide. Whereas in a cache-based system, we rely on the cache controller to transparently transfer data between main memory and on-chip cache (called servicing a cache miss), in a DMA-based system the programmer must explicitly write a DMA program that instructs the DMA controller to transfer data between main memory and on-chip memory as indicated with the pink arrows on the slide.

While this might sound like an impossible task for someone used to writing traditional programs on a general-purpose processor, it is greatly simplified by the fact that the data flow of most multimedia functions is predictable and regular. Because of that, a very popular approach of utilizing a DMA controller is to use double buffering with a 2D block or row-by-row transfer. On the slide we give an example of a 2D block transfer mode for an image processing function that has a single input and output image. To support double buffering, for each image transfer two on-chip buffers are allocated. At any one time, one set of buffers, In2 and Out2 in this example, are used by the processor for doing its computing while at the same time the DMA controller uses the other set of buffers for loading the next input image block and storing the previous results.

Double Buffering with DMA



Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

5

Once the computation and data transfer finishes, the role of the buffers is switched. The processor uses data from In1 and Out1, while the DMA controller transfers new data to In2 and writes Out2 back to main memory.

Invert8 Tight Loop

```
#include <eti/mm.h>
#include <hmpv_icl.h>
#include <invert8.h>

void invert8_tight_loop( n64 *restrict src64_ptr,
                       n64 *restrict dst64_ptr,
                       n32 size)
{
    int i, j;
    n64 const64_ff;

    const64_ff = hmpv_combine_64(0xffffffff, 0xffffffff);

    precondition_amount(16);

    for(i=0, j=0; i<size; i = i + NUM_PIXELS_INVERT8_TIGHT_LOOP, j++)
        dst64_ptr[j] = hmpv_sub_pu8(const64_ff, src64_ptr[j]);
}
```

Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

6

I always like to bring up these next two slides to give a sense of the complexity of writing DMA programs. This is the tight loop C code for 8-bit image invert. [switch to previous slide] So, one execution of the tight loop would take one input block and generate a single output block. [switch back] You can probably recognize what it does quite easily. The statement inside the loop uses a partitioned instruction to perform eight 8-bit subtractions in a single operation. On the MAP-CA, two such operations can be executed per cycle.

Mediaprocessor Programming

Cache-based

- Performance:
 - Register pressure and data dependences limit ability to overlap computation and data transfers
 - Little control to efficiently utilize main memory
- Programmability:
 - + Cache is transparent to programmer, no DMA program needed

DMA-based

- Performance:
 - + Computation and data transfer effectively overlapped
 - + Blocked memory transfers typically effectively utilize DRAM's page-mode access, thus achieving high throughput
- Programmability:
 - DMA programs can be difficult to write and debug
 - Lack of standardized API for DMA programming complicates porting of functions to other architectures

Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

8

So, comparing cache-based and DMA-based programming on a mediaprocessor, one observation is that DMA programming can be complex. In comparison, in a cache-based function, because caches are transparent to the programmer, the tight loop [switch to tight loop slide] is all that we ever have to write. [switch back] The reason DMA-based programming is so popular is because it significantly outperforms cache-based programming for most functions. I will quantify this difference on my results slide later. This difference arises in part for two reasons. First, using double buffering with a DMA controller, we can very effectively overlap computation and data transfer, whereas this is much harder to achieve with a cache-based program. Secondly, DMA controllers typically access memory in sequential blocks which effectively utilizes the DRAM's page-mode access. This is not naturally achieved with a cache-based program.

Program-directed Prefetcher

- Allows simple cache-based programming
 - Programmer must only specify a set of prefetch hints per region of data, which are stored in hardware registers
- Mimics data flow of DMA controller to achieve DMA-like performance
 - Prefetching of larger blocks of data possible to efficiently access main memory
 - Can prefetch far ahead for effective overlapping of computation and memory transfer
 - Multiple prefetch regions used to maximize prefetch accuracy for each type of data flow present in function

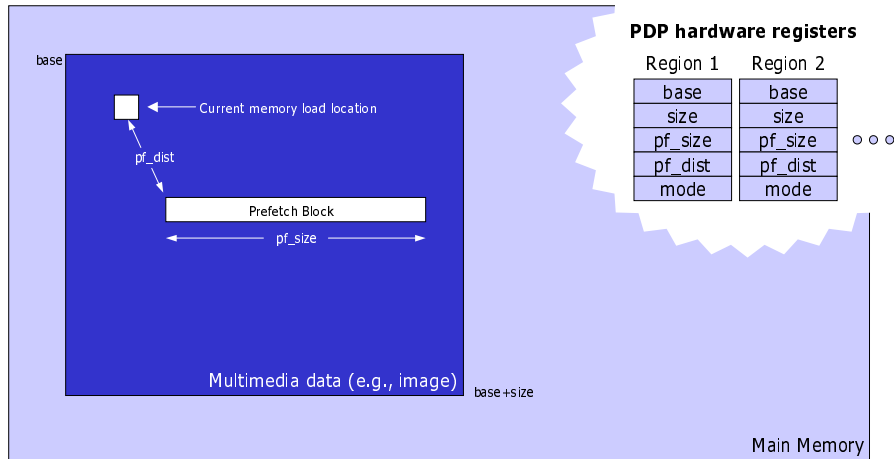
Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

9

The motivation for our work, therefore, has been to find a mediaprocessor memory model that can provide us the performance of a DMA controller with the simple programming of a cache. The first part of our solution is a cache prefetcher for a cache-based memory system. We wanted a new kind of prefetcher that was specifically optimized to deal with multimedia data flows, including the ability to prefetch large, sequential blocks from main memory to the cache to minimize page misses. We also wanted control over the prefetch distance to permit effective overlapping of computation and memory transfers.

Program-directed Prefetcher



Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

10

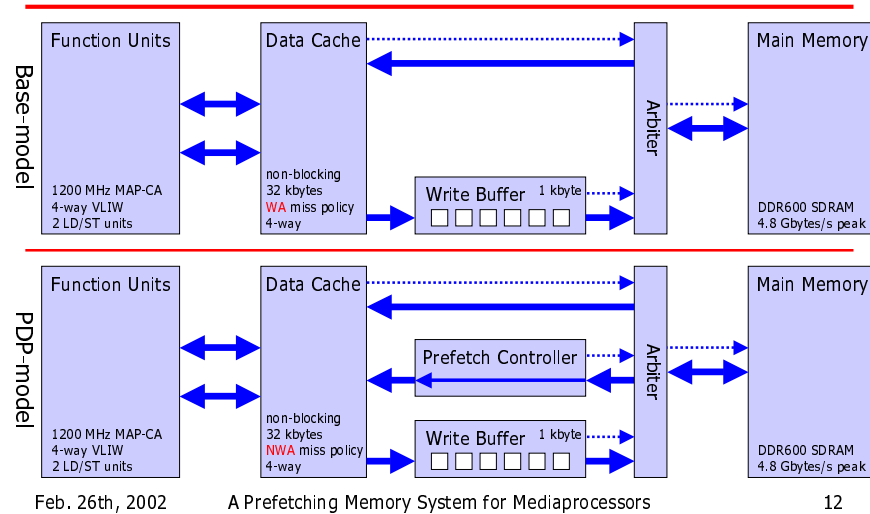
Our prefetcher is driven by prefetching hints provided by the programmer, thus the name program-directed prefetcher. These hints, stored in hardware registers, define basic prefetching parameters. We allow for multiple sets of prefetch parameters, each restricted to a specific region of memory, to support several distinct multimedia data flows simultaneously. The first two parameters in a set are therefore a base address and size that define the extent of a prefetch region in main memory. For example, for image invert, we could define region one to be the input image as indicated here by the dark blue region. Every memory reference by the processor inside this region will generate a prefetch block. Two additional prefetch parameters define the size of the prefetch block and its distance from the memory reference. Prefetch blocks are used as a basis for prefetching data to the processor.

No-write-allocate (NWA) Write-miss Cache Policy

- Write-allocate (WA) policy popular, but not efficient for writing output data
- NWA eliminates allocation of output data in cache when output data is not modified
- For typical image processing functions with one input image and one output image, WA has 50% greater memory traffic than NWA

The second part of our memory system is the use of a no-write-allocate write-miss cache policy instead of the more popular write-allocate policy. In the no-write-allocate policy, data will not be allocated in the cache on a write miss. We found this approach greatly reduces the total memory traffic for multimedia functions because it eliminates the allocation of the output data in the cache.

Simulation Models



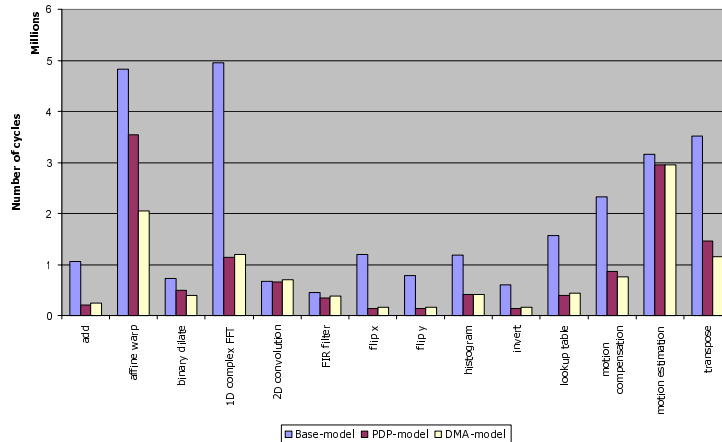
Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

12

We measured the performance of several functions on three different models. Our first model, the base model, is a traditional cache-based system very similar to that found on the MAP-CA, although we work with a 1200 MHz processor frequency and a double-data-rate SDRAM main memory with a peak throughput of 4.8 Gbytes per second. The cache is non-blocking, uses the write-allocate policy, and has a size of 32 kbytes with an additional 1-kbyte write buffer. Our second model uses the program-directed prefetcher that is different to the base model only in that it adds the prefetch controller. We also use the no-write-allocate policy on this model. The prefetch controller competes with the write buffer and cache refill path for access to the main memory. The arbiter gives the highest priority to cache refills and lowest priority to cache prefetches. Our third model, no shown here, uses a DMA controller capable of transferring data between main memory and an on-chip memory.

Execution Time Results



Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

13

With the limited time, I will only show execution time results from the three models. We use fourteen signal, image, and video processing functions to compare the models. The results were obtained through trace simulation.

The reductions in execution time with the DMA-model compared to the base-model ranges from 86% in flip x to a slightly worse execution time for the DMA-model in 2D convolution. On average, the execution time is reduced by 56%. Our cache-based memory model that uses the program-directed prefetcher and a no-write-allocate cache, achieves a reduction of 54% on average, so very closely matching the DMA result. In some cases, the DMA-model has a slightly worse execution time than our PDP-model due to the additional overhead associated with setting up the DMA program. In some other cases the greater flexibility of the DMA controller gives it an edge, such as in affine warp. In affine warp in particular, the DMA controller is quite efficient because it can be used to automatically generate zero pixels for pixels references that fall outside the input image. This operation takes more time in a cache-based version, whether with or without PDP, because it has to be done by the processor.

Conclusions

- DMA-based programming: best performance
- Cache-based programming: fast development
- Program-directed prefetcher:
 - Runs cache-based programs annotated with prefetching hints
 - Achieves almost DMA-based performance
 - Less flexible than DMA controller
 - Can simplify porting of functions across platforms
- PDP is intended to significantly simplify software development process without sacrificing performance

Feb. 26th, 2002

A Prefetching Memory System for Mediaprocessors

14

DMA controllers allow for extremely high performance. On the other hand, they are hard to program, something that has not only been a difficult challenge for students in the Image Computing Systems Laboratory in the Department of Electrical Engineering, who provided me with the benchmark functions, but also for our industry contacts with the lab. The cache-based memory system I introduced today, whose main parts are the program-directed prefetcher and a no-write-allocate cache, is intended as an alternative to using a DMA controller, but that is easier to program. Although DMA controllers provide more flexibility, which can be an advantage, I demonstrated that most functions running on our memory system achieve DMA-like performance.