Accelerating Blocked Matrix-Matrix Multiplication using a Software-Managed Memory Hierarchy with DMA

Roland E. Wunderlich Markus Püschel James C. Hoe

Department of Electrical and Computer Engineering
Carnegie Mellon University, Pittsburgh, PA 15213-3890
{rolandw, pueschel, jhoe}@ece.cmu.edu

Abstract

The optimization of matrix-matrix multiplication (MMM) performance has been well studied on general-purpose desktop and server processors. Classic solutions exploit common microarchitectural features including superscalar execution and the cache and TLB hierarchy to achieve near-peak performance. Typical digital signal processors (DSPs) do not have these features, and instead use in-order execution, configurable memory hierarchies, and programmable I/O interfaces.

We investigate the methods needed to achieve high performance MMM on the Texas Instruments C6713 floating-point DSP. This processor has two components that can be used to accelerate MMM: a software-managed memory hierarchy, and a direct memory access (DMA) engine that can perform block copies from main memory to into the memory hierarchy. Our MMM implementation overlaps computation with DMA block transfers. For matrices larger than the data caches, we observed a 46% performance increase over a blocked MMM implementation, and a 190% increase over the Texas Instruments DSP library.

Introduction

The availability of a high performance MMM implementation is of critical importance for a large range of numerical computation problems. MMM is both a common standalone function and a ubiquitous kernel of more complex computations.

Texas Instruments (TI) provides an optimized single precision floating point MMM implementation for their C67x processors, the DSPF_sp_mat_mul() function. This assembly-coded function is optimal for matrices that can fit within the L1 data cache. Its innermost loop attains 100% of the peak performance of the C6713 with minimal overhead for the outer loop control code. Unfortunately, TI's triple loop MMM implementation has poor data locality that leads to frequent cache misses for larger matrices.

MMM temporal data locality is improved by partitioning the computation so that it operates on cache resident submatrices (called blocks). The computation that is performed on these cache resident blocks is called the mini-MMM. This blocked MMM algorithm is used by all fast MMM implementations for general purpose processors including the Goto BLAS library [4], the Intel Math Kernel Library [1], and ATLAS library generator [5].

While blocked MMM also improves performance on DSPs, further performance gains can be achieved by using their software-managed memory hierarchies. Specifically, the DMA engine can efficiently copy sub-matrices directly from main memory into a higher level of the memory hierarchy. This block copying can also be overlapped with mini-MMM computation. This approach has been applied before for MMM on the PlayStation 2's vector units [2].

We develop a fast MMM implementation for a widely used DSP, the TI C6713. Our implementation uses: (1) blocked MMM computation, and (2) DMA block copies from main memory to the scratch pad in parallel with the mini-MMM computation. This approach achieves the optimal FLOP rate of in-cache mini-MMM computation for large non-cache-resident matrices.

DSP microarchitecture features

The TMS320C6713 DSP [3] is the latest implementation in the C67x family of high-end floating-point (FP) DSP chips from Texas Instruments and is available at speeds up to 300 MHz. The C6713 has two single precision FP adders, two FP comparators, and two FP multipliers that give it a peak performance of 1800 MFLOPS. The peak theoretical performance for MMM is 1200 MFLOPS because only the adders and multipliers are used.

The C6713 has two levels of cache and a 192 KB scratch pad SRAM, referred to by TI as "L2 mapped RAM." Data in main memory are cached normally by the L1 caches and the L2 cache. Data explicitly allocated or copied into the scratch pad are also cached upon usage in the L1 caches. The 4 KB L1 data cache has a 4 cycle latency, and the 64 KB unified L2 cache has an 8 cycle latency and can also be used as scratch pad memory.

The C6713 DMA engine allows for background memory block transfers. The C6713 DMA engine can also perform memory transfers to and from the scratch pad. This allows data to be loaded directly into memory hierarchy. Figure 1 illustrates the major components of the C6713 DSP.

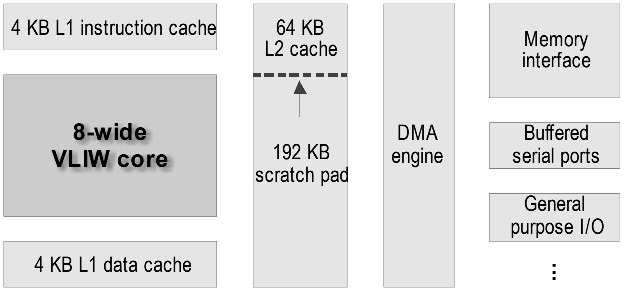

Figure 1: The TI C6713 DSP microarchitecture

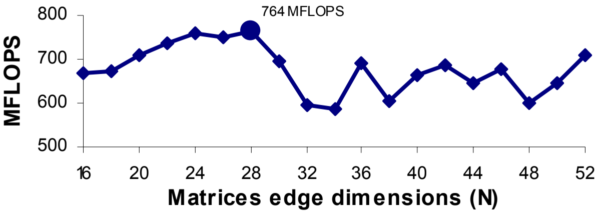

Figure 2: Search for optimal block size with TI MMM

Measurement methodology

All performance results are presented as MFLOPS as measured on the TI Code Composer Studio 3.0 cycle accurate simulator. Before each MMM implementation is measured, we touch the values of the input matrices to recreate the cache state as if the matrices had been produced by a preceding function. The L2 cache was configured at its largest possible size of 64 KB. The TI TMS320C6x C/C++ Compiler 5.0 was used to compile our C code implementation. Currently, our MMM implementations only support matrices that are even multiples of the block size. Preliminary implementations without this limitation see only minor losses of performance.

Blocked MMM

The first optimization we perform is to implement a blocked MMM to address the main shortcoming of the TI MMM for matrices larger than the data caches.

We use two approaches to determine the best block size for our blocked MMM, a model-driven optimization by Yotov et al. [6], and a search similar to ATLAS [5]. The model-driven approach predicts that a block size of 28×28 is best for the 4 KB L1 data cache of the C6713. The results of the empirical search match the model's prediction, as plotted in Figure 2.

Our blocked MMM function uses the TI MMM function for the mini-MMM computation, and a block size of 28×28. The performance of this implementation is plotted in Figure 3 for matrices up to 336×336. The performance of the TI MMM degrades as the input matrices no longer fit in the data caches, while the blocked MMM suffers far less due to its superior data locality. The blocked MMM is slower than the TI MMM for small matrices because it performs memory block copies.

Blocked MMM with scratch pad and DMA

The block copy operations that are required by a blocked MMM implementation can be performed faster by using the DMA engine of the C6713 instead of explicitly reading and then writing floating point values. The DMA engine can load matrix data faster than the CPU because it issues cache-line sized requests whereas the CPU must transfer data one word at a time.

Allocating MMM blocks in the scratch pad is a second part of effectively using the architecture of the C6713. Figure 3 plots the performance of our implementation that replaces conventional block copy operations with DMA transfers to blocks allocated in the scratch pad. The overhead for the setup of DMA transfers results in worse performance than the non-DMA blocked MMM for the smallest of matrices. However for large matrices, the DMA transfers improve performance by 37% on average over the non-DMA blocked MMM.

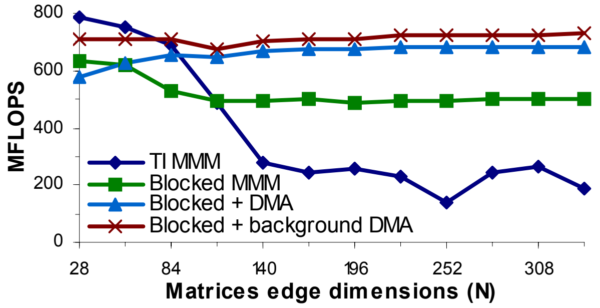

Figure 3: Performance of our MMM implementations

Blocked MMM with background DMA

Because the DMA operations take place with little interaction with the CPU, it is possible to perform at least some of the DMA transfers in the background with the mini-MMM computation. We modify our blocked MMM to overlap DMA block transfers with each mini-MMM computation.

Our final MMM implementation with background DMA achieves a 7% performance gain over the serial DMA version. Figure 3 plots the performance of the background DMA implementation. For large matrices, our MMM implementation comes within 4% of the peak performance of the TI MMM for cache resident matrices. This means that the block operations and loop overhead are only 4% of the runtime of our blocked MMM implementation.

Conclusions

The blocked MMM is an effective algorithm for high performance on general purpose processors. On DSP chips such as the C6713, the scratch pad and DMA enable further performance gains. By performing DMA transfers in the background with the mini-MMM, we achieve performance limited only by the mini-MMM.

References

[1] Intel Math Kernel Library web site [Online]. Available: http://www.intel.com/software/products/mkl/

[2] Scientific Computation on PlayStation 2: Using the Vector Units [Online]. Available: http://arrakis.ncsa.uiuc.edu/ps2/using_vector_units.php

[3] Texas Instruments DSP products web site [Online]. Available: http://dspvillage.ti.com/

[4] K. Goto and R. van de Geijn. "On reducing TLB misses in matrix multiplication." Dept. Comput. Sci., Univ. Texas, Austin, Tech. Rep. TR-2002-55, 2002.

[5] R. C. Whaley, A. Petitet, and J. J. Dongarra. "Automated empirical optimization of software and the ATLAS project." *Parallel Comput.*, vol. 27, no. 1-2, pp. 3-35, 2001.

[6] K. Yotov, X. Li, G. Ren, M. J. Garzarán, D. Padua, K. Pingali, and P. Stodghill. "Is search really necessary to generate high-performance BLAS?" Proc. IEEE, vol. 93, no. 2, pp. 358-386, 2005.