General Matrix-Matrix Multiplication Using SIMD Features of the PIII

Douglas Aberdeen Douglas.Aberdeen@anu.edu.au
Jonathan Baxter Jonathan.Baxter@anu.edu.au

Research School of Information Sciences and Engineering
Australian National University

Abstract. Generalised matrix-matrix multiplication forms the kernel of many mathematical algorithms. A faster matrix-matrix multiply immediately benefits these algorithms. In this paper we implement efficient matrix multiplication for large matrices using the floating point Intel SIMD (Single Instruction Multiple Data) architecture. A description of the issues and our solution is presented, paying attention to all levels of the memory hierarchy. Our results demonstrate an average performance of 2.09 times faster than the leading public domain matrix-matrix multiply routines.

1 Introduction

A range of applications such as artificial neural networks benefit from GEMM (generalised matrix-matrix) multiply routines that run as fast as possible. The challenge is to use the CPU's peak floating point performance when memory access is fundamentally slow. The SSE (SIMD Streaming Extensions) instructions of the Intel Pentium III chips allow four 32-bit (single precision) floating point operations to be performed simultaneously. Consequently, efficient use of the memory hierarchy is critical to being able to supply data fast enough to keep the CPU fully utilised. In this paper we focus on the implementation of an efficient algorithm for the Pentium SIMD architecture to achieve large, fast, matrix-matrix multiplies. Our code has been nicknamed Emmerald.

Without resorting to the complexities associated with implementing Strassen's algorithm on deep-memory hierarchy machines [5], dense matrix-matrix multiplication requires $2MNK$ floating point operations where $A : M \times K$ and $B : K \times N$ define the dimensions of the two matrices. Although this complexity is fixed, skillful use of the memory hierarchy can dramatically reduce overheads not directly associated with floating point operations. It is the optimization of memory hierarchy combined with the SSE that gives Emmerald its performance.

Emmerald implements the SGEMM interface of Level-3 BLAS, and so may be used immediately to improve the performance of single-precision libraries based on BLAS (such as LAPACK [4]). There have been several recent attempts at automatic optimization of GEMM for deep-memory hierarchy machines, most notable are PHiPAC [3] and the more recent ATLAS [6]. ATLAS in particular achieves performance close to optimized commercial GEMMs. Neither ATLAS nor PhiPAC make use of the SSE instructions on the PIII for their implementation of SGEMM.

Our experiments showed that ATLAS achieves a peak of 375 MFlops/s for single-precision multiplies on a PIII @ 450 MHz, or $0.83\times$ *clock rate*. Our matrix-matrix multiply using SIMD instructions achieves a peak of 890 MFlops/s, or $1.98\times$ *clock rate*. We also report a price/performance ratio under USD $1/MFlop/s for training Neural Networks for Japanese OCR. The following section will describe our novel use of the SSE for Emmerald, followed by a description of optimizations designed to improve use of the memory hierarchy. The paper concludes with a comparison between ATLAS and Emmerald.

2 SIMD Parallelization

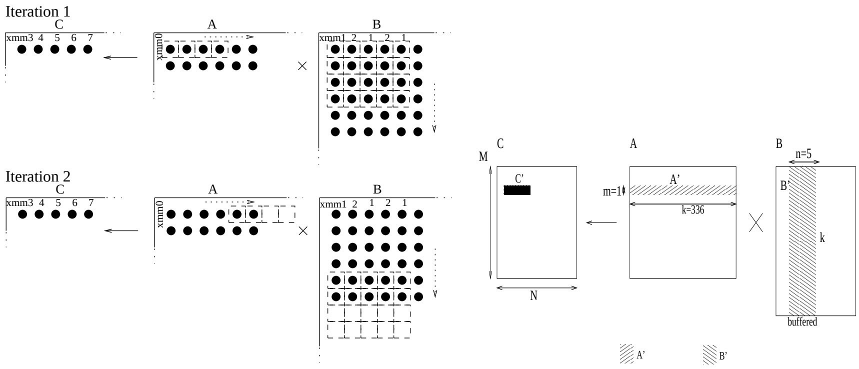
Two core strategies are employed to minimise the ratio of memory accesses to floating point operations: accumulate results in registers for as long as possible to reduce write backs and re-use values in registers as much as possible. In [2] several dot-products were performed in parallel as the innermost loop of the GEMM. We took the same approach and found experimentally that 5 dot-products in the inner loop gave the best performance. Figure 1(a) shows how these 5 dot products utilise SIMD parallelism.

Four values from a row of A are loaded into an SSE register. This is re-used five times by doing SIMD multiplication with four values from each of the five columns of B used in the inner loop. Two SSE registers are allocated to loading values from B. Results are accumulated into the remaining five SSE registers. When the dot-product ends each SSE result register contains four partial dot-product sums. These are summed with each other then written back to memory. For the best efficiency, the dot product length is maximised with the constraint that all data must fit into L1 cache.

3 Memory Hierarchy Optimizations

A number of standard techniques are used in Emmerald to improve performance. Briefly, they include:

- *L1 blocking*: Emmerald uses matrix blocking [2, 3, 6] to ensure the inner loop is operating on data in L1 cache. Figure 1(b) shows the L1 blocking scheme. The block dimensions m and n are determined by the configuration of dot-products in the inner loop (Section 2) and k was determined experimentally.
- *Unrolling*: The innermost loop is completely unrolled for all possible lengths of k in L1 cache blocks while avoiding overflowing the instruction cache.
- *Re-buffering*: Since B' is large (336×5) compared to A' (1×336), we deliberately buffer B' into L1 cache. By also re-ordering B to enforce optimal memory access patterns we minimise translation look-aside buffer misses [6].



(a) Five dot products using eight SSE registers (`xmm[0-7]`). Each circle is an element in the matrix. Each dashed square represents one floating point value in an SSE register.

(b) L1 blocking for Emmerald: $C' \leftarrow A'B'$ where A' and B' are in L1 and C' is accumulated in registers.

- *Pre-fetching*: Values from A' are not buffered in L1. We make use of SSE prefetch assembler instructions to bring A' values into L1 cache when needed.
- *L2 Blocking*: Efficient L2 cache blocking ensures that peak rates can be maintained as long as A, B and C fit into main memory.

4 Results

The performance of Emmerald was measured by timing matrix multiply calls with size $M, N, K = \{16, 17 \ldots 700\}$. To ensure a conservative performance estimate we use wall clock time on an unloaded machine rather than CPU time; the stride of the matrices (which determines the separation in memory between each row of matrix data) is fixed to the largest matrix size (700) and caches are flushed between calls to `sgemm()`. Timings were performed using a PIII 450 MHz with 128 MB RAM, 512 KB L2 cache and a 100 MHz bus.

Figure 2 shows Emmerald's performance compared to ATLAS and a naive three-loop matrix multiply. The average MFlop/s rate of Emmerald after size 100 is 1.69 times the clock rate of the processor and 2.09 times faster than ATLAS. A peak rate of 890 MFlops/s is achieved when $m = n = k = stride = 320$. This represents 1.97 times the clock rate. The largest tested size was $m = n = k = stride = 4032$ on a 550 MHz machine which ran at 1016 MFlops/s.

We have used Emmerald in distributed training of large Neural Networks with more than one million adjustable parameters and a similar number of training examples [1]. By distributing training over 196 Intel Pentium III 550 MHz processors, and using Emmerald as the kernel of the training procedure, we achieved a sustained performance of 152 GFlops/s for a price performance ratio of 98¢ USD/MFlop/s (single precision).

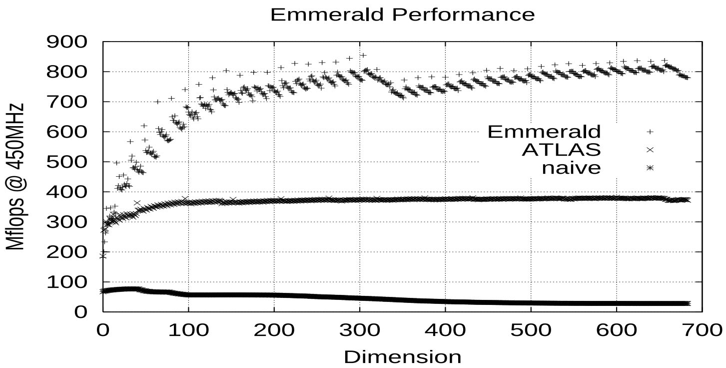


Fig. 2. Performance of Emmerald on a PIII running at 450 MHz compared to ATLAS SGEMM and a naive 3-loop matrix multiply. Note that ATLAS does not make use of the PIII SSE instructions.

5 Conclusion

This paper has presented Emmerald, a version of SGEMM that utilises the SIMD instructions and cache hierarchy of the Intel PIII architecture. An application demonstrating the cost-effectiveness of such work was also reported. This code and the full version of this paper is available from `http://beaker.anu.edu.au/research.html`.

References

1. D. Aberdeen, J. Baxter, and R. Edwards. 98¢ /mflop, ultra-large-scale neural-network training on a PIII cluster. Sumbitted to SC2000, May 2000.
2. B. Greer and G. Henry. High performance software on Intel Pentium Pro processors or Micro-Ops to TeraFLOPS. Technical report, Intel, August 1997. http://www.cs.utk.edu/ ~ghenry/sc97/paper.htm.
3. J.Bilmes, K.Asanovic, J.Demmel, D.Lam, and C.W.Chin. PHiPAC: A portable, high-performace, ANSI C coding methodology and its application to matrix multiply. Technical report, University of Tennessee, August 1996. http://www.icsi.berkeley.edu/ ~bilmes/phipac.
4. Netlib. *Basic Linear Algebra Subroutines*, November 1998. http://www.netlib.org/blas/index.html.
5. M. Thottethodi, S. Chatterjee, and A. R. Lebeck. Tuning strassen's matrix multiplication for memory efficiency. In *Super Computing*, 1998.
6. R. C. Whaley, A. Petitet, and J. J. Dongarra. Automated empirical optimizations of software and the ATLAS project. Technical report, Dept. of Computer Sciences, Univ. of TN, Knoxville, March 2000. http://www.cs.utk.edu/ ~rwhaley/ATLAS/atlas.html.