Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 21 (2005) 743–748
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

Development of efficient computational kernels and linear algebra routines for out-of-order superscalar processors

O. Bessonov[a,*], D. Fougère[b], B. Roux[b]

[a] *Institute for Problems in Mechanics of the Russian Academy of Sciences, 101, Vernadsky ave., 119526 Moscow, Russia*
[b] *Laboratoire de Modélisation et Simulation Numérique en Mécanique (L3M), L3M–IMT, La Jetée, Technopôle de Château-Gombert, 13451 Marseille Cedex 20, France*

Available online 25 June 2004

Abstract

We present methods for developing high performance computational kernels and dense linear algebra routines. The microarchitecture of AMD processors is analyzed with the goal to achieve peak computational rates. Approaches for implementing matrix multiplication algorithms are suggested for hierarchical memory computers. Block versions of matrix multiplication and LU-decomposition algorithms are considered. The obtained performance results for AMD processors are discussed in comparison with other approaches.

Keywords: Instruction level parallelism; Out-of-order processor; Cache memory; Performance measurement; LINPACK benchmark

1. Introduction

Modern computers achieve on typical applications only 15–25% of their "theoretical speed" due to memory limitations. For dense linear algebra [4], much higher levels can be reached (50–95%, depending on CPU or system architecture). Development of such algorithms can be considered as approaching "the speed of sound" of particular architecture and demonstrating its potential. To achieve this performance, investigation of a system architecture is necessary in order to exploit fully the intrinsic instruction level parallelism (ILP), with the following development of efficient and robust computational algorithms.

The reason of the efficiency of linear algebra algorithms is that they perform $O(n^3)$ arithmetic operations on $O(n^2)$ data elements. Therefore, blocking strategies with caching can be employed. In the previous paper [2] we proposed a new approach based on multiplication of a block-vector by matrix, as opposed to vector–matrix (BLAS 2) and matrix–matrix (BLAS 3) ones. This approach (BLAS 2.5) combines the efficiency of BLAS 3 with the flexibility and scalability of BLAS 2, because it depends less on the shape and size of matrices. Linear equation solvers based on this new algorithm have demonstrated the record level of LINPACK benchmark [3] performance for some RISC microprocessors.

* Corresponding author.
E-mail address: bess@ipmnet.ru (O. Bessonov).

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2004.05.016

The ILP of RISC processors is revealed easily due to their regular structure with uniform instruction sets, large number of addressable registers and deterministic execution. This is not true for superscalar CISC processors of 86× architecture (Intel Pentium-4 and AMD Athlon). However, irregularities can be partly compensated by their highly asynchronous microarchitecture, with deep out-of-order execution and register renaming. To achieve a good performance on CISC processors, computational cores must be based on long independent chains of instructions, rather than on cycle-by-cycle scheduling (as for RISC).

The goal of this work is to analyze the architecture of AMD processors and to employ the previous experience for building efficient computational cores and linear algebra kernels. We describe the new method (as applied to AMD processors) and compare it to the "Cache-contained matrix multiply" approach (ATLAS Project [6]). Our method demonstrates superiority for a range of matrix sizes (small to medium). The record results of LINPACK-1000 benchmark are achieved for Athlon and Duron CPUs and registered appropriately. These results are comparable to the performance of RISC processors.

2. Microarchitecture of processors and development of computational cores

AMD Athlon/Duron are very fast CISC (Complex Instruction Set Computer) processors [1]. They are characterized by three-way superscalar architecture with multiple functional units, deep out-of-order execution and register renaming; pipelined 87× Floating Point Unit that executes up to two 64-bit operations per cycle; eight floating point registers organized as a stack; large physical register files used for renaming (remapping) architectural registers in asynchronous execution; 64 kB level 1 data cache and 256 kB L2 cache.

Some of these properties may reduce the performance (e.g. legacy 86× instruction set), others compensate limitations (e.g. memory prefetch and asynchronous execution). Appropriate coding is required in order to avoid bottlenecks and tolerate limitations.

The core of most linear algebra routines is the matrix multiplication algorithm. Different forms of this algorithm were investigated in [2]. The scalar product

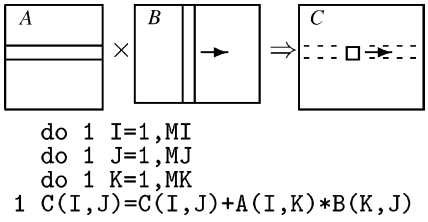

Fig. 1. Scalar product form of the matrix multiplication algorithm.

form (Fig. 1) is chosen for AMD processor architecture as the most efficient. To achieve 80–90% of the peak computational speed (for 64-bit FP arithmetic), the following method is used:

- multiply a block row in the matrix *A* by the matrix *B*, using the L1 cache as a pool of vector registers (to keep this block row);
- rely on asynchronous (out-of-order) execution to hide instruction and memory latencies, since the static instruction scheduling is not applicable to 87× stack architecture;
- pack every three instructions into an aligned 8 B bundle for higher decoding rate;
- set the width of a block row in *A* to 4 or 6 (limited by the depth of 87× register stack);
- group every four block rows in *A* (of the width 4 or 6) into a "wide" block row (of the width 16 or 24) to tolerate the limited memory throughput (Fig. 2);
- employ data prefetch for the next column of the matrix *B* to hide memory latencies.

A group of four block rows in *A* is copied preliminarily into a dense work array. Owing to the cache replacement algorithm, this array becomes effectively "locked" in the L1 cache after the first iteration. The typical length of this group (256 for the block width 4) corresponds to the array size 32 kB, i.e. half of the L1

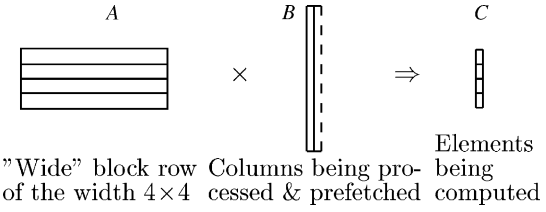

Fig. 2. Grouping block rows in a computational core.

```
fldl    (%edi,%eax)
fldl    (%edx,%eax,4)
fmul    %st(1),%st      # 1
faddp   %st,%st(5)
fldl    8(%edx,%eax,4)
fmul    %st(1),%st      # 2
faddp   %st,%st(4)
fldl    16(%edx,%eax,4)
fmul    %st(1),%st      # 3
faddp   %st,%st(3)
fmull   24(%edx,%eax,4)
faddp   %st,%st(1)      # 4
```

```
do K=1,NK
  W=W+V(1,K)*B(K,J)
  X=X+V(2,K)*B(K,J)
  Y=Y+V(3,K)*B(K,J)
  Z=Z+V(4,K)*B(K,J)
enddo
```

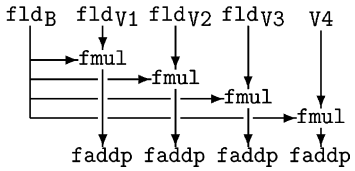

Fig. 3. Basic block in assembler, its FORTRAN notation and the dependency graph.

cache. All other memory accesses are implicitly served through another half (cache line) of the L1 cache.

The scalar product results are accumulated in the 87× register stack. Four block rows are multiplied in sequence by the same column of the matrix B with prefetching the next column in B into the cache. A basic block of the algorithm (Fig. 3) consists of four aligned 8 B bundles (12 instructions, 8 FP operations) and is executed ideally in four clock cycles.

The dependency graph consists of four independent instruction chains, each ∼12 clock cycles long, with deep overlap due to asynchronous (out-of-order) execution and register renaming. Execution speed of this inner loop (with loop control and prefetch instructions) achieves 90% of the peak computational rate, i.e. 1.8 FP instructions per clock cycle. This corresponds to the performance 2750 MFLOPS for the processor frequency 1530 MHz.

3. Matrix multiplication and solving linear systems

Generally, two-level blocking strategy is used for matrix multiplication algorithms [2]: strip-mining (vertical splitting of the matrix A) to fit a part of a block-row in A to L1 cache, and tiling (vertical splitting of the matrix B) to fit a rectangular block to L2 cache.

For AMD processors, tiling is no more necessary because the new algorithm with "wide" block-rows tolerates the limited memory throughput (about 1300 MB/s on Athlon MP1800+/1530 MHz and 1000 MB/s on Duron/900 MHz). Therefore, only strip-mining can be applied, that makes the algorithm more simple and

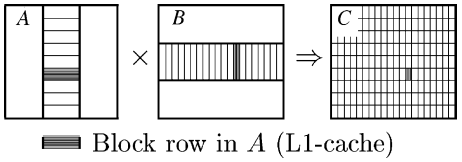

Fig. 4. Blocking strategy for the matrix multiplication (with strip-mining).

straightforward (Fig. 4). The width of strips is determined by the size of the L1 cache and is equal to 256.

For a linear equation solver, the top-looking variant with partial pivoting and strip-mining is used [5,2] (Fig. 5). The LU-decomposition algorithm consists of two basic steps: processing a trapezoidal strip (this is similar to the multiplication of a "wide" block row in L1 cache by a matrix strip), and solving a subsystem within a "wide" block with pivoting.

The first step is performed using the matrix multiplication algorithm as above. The second step and the solution of triangular systems (using the computed factors U and L) are performed separately. They are not time-consuming as the first step. Nevertheless, the resulting performance of the linear is lower than that of the matrix multiplication.

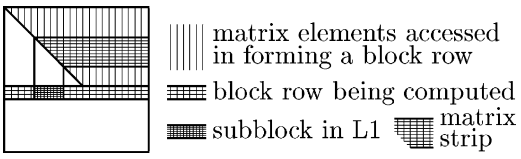

Fig. 5. Top-looking variant of LU-decomposition for solving linear systems.

4. Results and comparisons with other approaches

Performance results of the new algorithms are presented in Fig. 6 for two AMD processors. On Athlon (1530 MHz), the matrix multiplication algorithm reaches about 80% of the peak speed (3060 MFLOPS) for small matrices that fit into the L2 cache (256 kB), and reduces to 72% for bigger matrices. For solving linear systems, performance increases with the matrix size. Duron is a cheaper processor, with smaller L2 cache (64 kB), simpler memory subsystem and lower frequency. Its performance behavior is similar.

These results are compared in Fig. 6 with the results of the Athlon-optimized ATLAS software implementation [6]. The idea of the ATLAS approach is "Cache-contained matrix multiply" when matrices are split into square submatrices of small size. Several submatrices can fit into L1 cache, and the amount of memory accesses can be reduced. For the Athlon-optimized implementation, this size is 30×30 (i.e. about 7 kB).

For the comparison, the results for AMD ACML and Intel MKL libraries are also presented. It is seen that the MKL is not adequate for AMD processors. On the other hand, the results of Athlon-optimized ACML library are closer to that of the leaders.

The new algorithm demonstrates competitive performance and wins on small and medium-size matrices. In particular, it outperforms the ATLAS for the LINPACK-1000 benchmark (solving a linear system of the size 1000). For Athlon (1530 MHz) and Duron (900 MHz) processors the new results are 1705 and 977 MFLOPS respectively. These results are registered as record values (for above processors) in the database [3]. The best LINPACK-1000 results for ATLAS and ACML libraries are shown in the table in Fig. 6.

The new results for Athlon processors are comparable to that of some modern RISC computers. For example, the best LINPACK-1000 result for the fastest Alpha processor (1250 MHz) is 1945 MFLOPS. This means that modern inexpensive commodity microprocessors (like AMD Athlon) have become a very attractive alternative for performing scientific computations and building low-cost computer systems and clusters.

Performance of the new algorithm will be even higher on the new AMD64 CPUs (Opteron and Athlon64). These processors have the extended 86–64× architecture with some performance enhancements like SSE2 FPU instructions and bigger register sets. Due to this, more efficient implementations will become possible in a future.

Below are the general properties of the BLAS 2.5 as compared to the ATLAS approach:

- it is more simple in implementation;
- it is more flexible for application to complicated matrix shapes (e.g. in LU-decomposition);
- it minimizes the required memory access rate below some reasonable (and sufficient) level;
- it behaves better for small and medium-size matrices;

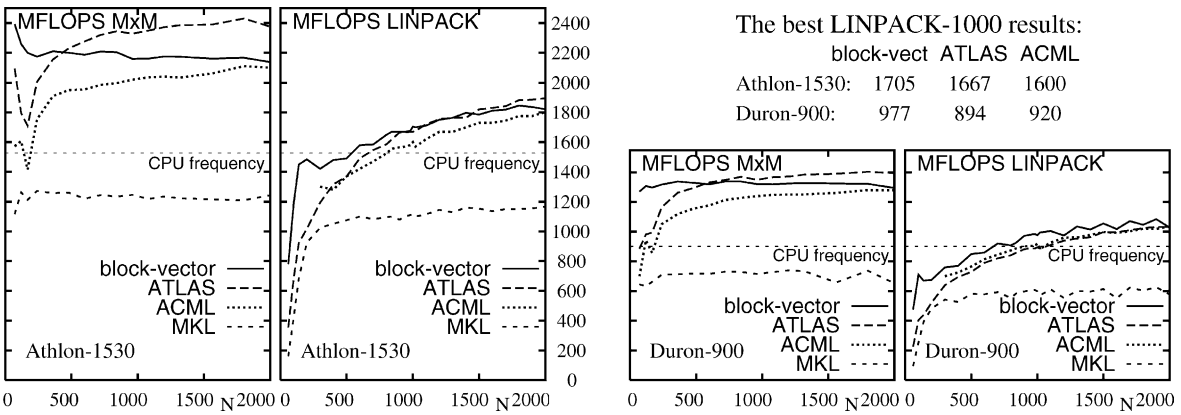

Fig. 6. Performance of different algorithms for matrix multiplication (left plots) and solving linear systems (right plots) on Athlon-1530 and Duron-900 processors.

- on the other hand, the ATLAS behaves better for very large matrices.

The further optimization of new algorithms will be based on the results of this comparison. The combination of two competing approaches looks very attractive for implementing adaptive linear algebra kernels, to be used in parallel software for clusters and MPPs.

5. Conclusion

The new methods and algorithms, described in this paper, combine the efficiency of a large block approach with the flexibility and scalability of vector–matrix operations. They demonstrate the record level of performance for several processors, can be easily adapted to new architectures and incorporated into other serial and parallel libraries.

Implementation for 86×/87× CISC CPUs (AMD Athlon/Duron) tolerates low memory bandwidth and does not depend much on blocking strategies and outer cache levels. The record LINPACK-1000 results are obtained with this algorithm and registered in [3].

In comparison with ATLAS [6], our algorithm shows competitive performance and wins on some matrix sizes. It is more flexible for processing narrow matrices and for solving linear systems. To achieve better results, these two approaches may be combined.

As a result, the ability to achieve multi-gigaflops performance on a single inexpensive commodity microprocessor increases attractiveness of 86× CISC architectures for building high-performance computer systems and clusters.

Acknowledgements

This work was partially supported by the Russian Foundation for Basic Research (grants 01-01-00745 and 02-01-00210).

References

[1] AMD Athlon™ Processor 86× Code Optimization Guide, Publication No. 22007, Advanced Micro Devices, February 2002.

[2] O. Bessonov, D. Fougère, K. Dang Quoc, B. Roux, Methods for Achieving Peak Computational Rates for Linear Algebra Operations on Superscalar RISC Processors, in: Proceedings of the PaCT-99, LNCS, vol. 1662, Springer, Berlin, 1999, pp. 180–185.

[3] J. Dongarra, Performance of Various Computers Using Standard Linear Equations Software, Report CS-89–85, University of Tennessee, Knoxville, TN, 2003.

[4] J. Dongarra, D. Walker, The Design of Linear Algebra Libraries for High Performance Computers, Lapack Working Note 58, University of Tennessee, Knoxville, TN, 1993.

[5] J.M. Ortega, Introduction to Parallel and Vector Solution of Linear Systems, Plenum Press, New York, 1988.

[6] R.C. Whaley, A. Petitet, J. Dongarra, Automated Empirical Optimization of Software and the ATLAS Project, Parallel Computing 27 (1–2) (2001) 3–35.

Oleg Bessonov received his diploma degree in nuclear physics and system programming in 1976 at the Moscow Institute of Physics and Technology (MIPT). Then he worked at the Institute for High Energy Physics as a system programmer and a head of system programming team. Since 1988, he is a senior research scientist at the Institute for Problems in Mechanics. In his work he closely collaborates with Dr. Bernard Roux and his colleagues from the laboratory L3M at Marseille, France. His current research activities include design of high performance algorithms, parallel and distributed computing, development of efficient methods for incompressible hydrodynamics and numerical investigation of fluid flows in crystal growth applications.

Dominique Fougère received computer science engineer degree at the Research National Institute of Computer Science and Automatics (INRIA) in 1976 where she worked as a system and network engineer-developer. From 1983 to 1989 she had the head position of the network and system team at the National university center for technical and scientific information in Lyon. Since 1990 she has the head position for computer science in the group "High Performance Computing in Mechanics" at the laboratory L3M in Marseille. Her current research activities include design of high performance computing, cluster management, HPC computing grids, parallel and distributed computing.

Bernard Roux received his PhD degree in 1966 and then his state doctorate degree in sciences in 1971 at the Marseille University. He got a position in 1967 at the Institute of Fluids Mechanics and created the Computational Fluid Dynamics team. He developed CFD in several areas: fluid dynamics, heat/mass transfer, coupled mechanisms, vibrational convection, etc. His research activities include mathematical and physical modeling of crystal growth processes, space-related physical sciences, high performance computing. Presently he is head of the group "High Performance Computing in Mechanics" at the laboratory L3M in Marseille, manager of several research networks and research-training networks, co-editor of the Series "Notes on Numerical Fluid Mechanics" (Springer, Germany), member of Editorial Advisory Board of the journal "Microgravity Quarterly" (Pergamon Press, USA). Awards: Silver Medal of CNRS, France (1987); Docteur Honoris Causa of Perm State University, Russia (1995).