A Parallel Symbolic Computation Environment: Structures and Mechanics

'Mantŝika Matooane and Arthur Norman

University of Cambridge Computer Laboratory,
New Museums Site, Pembroke Street,
Cambridge CB2 3QG, United Kingdom,
mam32@cam.ac.uk, acn1@cam.ac.uk

Abstract. We describe a set of representations for polynomials and sparse matrices suited for use with fine-grain parallelism on a distributed memory multiprocessor system. Our aim is to support use of supercomputers with this style of architecture to perform computations that would exceed the main memory capacity of more traditional computers: although such systems have very high performance communication networks it is still essential to avoid letting any one part of the network become a bottleneck. We use randomised data placement both to avoid hot-spots in the communication patterns and to balance (in a probabilistic sense) the memory load placed upon each processing element. The expected application areas for such a system will be those where intermediate expression swell means that the huge primary memory available on MPP systems will be needed if the smaller final result is to be successfully computed.

1 Introduction

We study very fine grained parallelism in computing with small numbers of huge polynomials. The fine granularity is a result of the need to distribute each polynomial across several processing elements (PE's) taking advantage of the memory available and enabling concurrent computation on multiple arguments on each processing element. Granularity and communication cost are inversely related [9], therefore the fine granularity in our algorithms incurs large communication costs.

The emphasis in this paper, on the low level details of data structures and the communication flow, is motivated by the need to compare the efficiency of different representations in parallel and sequential implementations. Secondly, we would like to compare any differences between symbolic and other algorithms regarding network use [1].

To ensure that each polynomial is uniformly distributed we follow the universal hashing mechanism introduced in CABAL [7]. In this paper we elaborate on the strategy and evaluate it performance and progress compared to the earlier version. We also monitor the traffic generated by two computations: multiplication and solution of large sparse systems of equations. An analysis of the

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 1492–1495, 1999.

communication patterns generated is conducted. These experiments reveal the need to address the bandwidth requirements while attempting to balance peak memory demand.

The parallel environment consists of processing elements with tightly-coupled memory, and cooperation through message passing. Although we have been able to carry out small scale testing and debugging on networks of workstations, the target for serious applications is a Hitachi SR2201[4] computer with 256 nodes, each with 256 Mbytes of memory giving a total 64 Gbytes. We note that this amount of memory should eventually allow us to address problems that are significantly beyond the capabilities of large individual workstations.

The speed of the communication network enables us to consider algorithms that generate large numbers of messages. However, this does not remove the need for bandwidth and synchronization consideration in the algorithm.

For the work described here the message passing uses MPI[12, 8] rather than PVM, which had been used in an earlier exploration of these ideas. This change was to fit in better with the preferred and best supported software environment on the target computer, and reflects the growing maturity of MPI [2] as a widely available low-level substrate for parallel processing.

2 Polynomial and Matrix Representations

The data distribution problem for numerical matrix algorithms is a static one and is solved in various ways: block distribution, cyclic, block-cyclic, cyclic-cyclic [5, 6]. The choice between these is made on the basis of how the distribution interacts with the algorithm being performed, and an attempt is normally made to minimize communication requirements.

Symbolic computations often exhibit intermediate expression swell [13]. This recurring hazard requires that we implement dynamic distribution which balances storage requirements on each processing element during computation. The growth of terms is not easy to predict and therefore a policy such as block distribution may result in heavy storage use on a single processor and possibly overflow.

We count the need to distribute parts of large intermediate results across many separate memory modules as the driving force in this work.

The basic strategy used is to represent polynomials in a distributed format, using a fixed collection of indeterminates so that the associated exponents could be kept in packed vectors of bytes. No a-priori ordering is imposed on the terms making up a polynomial[3]. The justification for this is that the asymptotic cost of multiplication for polynomials where the terms are kept sorted is $O(n^2 \log n)$ and for very large formula the logarithmic factor would be especially important. A system-wide hash table supports the unordered merge operations involved in polynomial arithmetic and allows polynomial multiplication of n-term polynomials to complete in expected time $O(n^2)$. Whenever a monomial is created a hash value was computed for it; part of this hash value is used to determine which PE the monomial belongs in, while the rest determines its placement in the hash

table maintained within that PE. This scheme results is every polynomial having its terms pseudo-randomly spread among all the PE's . The hashing leads to even distribution so the memory load will be balanced, although of course there could be pathological cases where hashing attempted to map almost all terms to just one of the PE's. Each PE holds copies of a few index values (as well as the hash table) that provide access to the head of each polynomial stored there.

Our application area is symbolic linear algebra, and specifically the calculation of determinants of sparse symbolic matrices [11, 10]. Because the structures needed to represent the sparse matrices are fairly compact we keep a copy of them in each PE so that this information at least does not need repeated broadcast.

The most communication-intensive part of our computation occurs when two large polynomials need to be multiplied. Every possible pair of terms will have to be combined, and the resulting product term will then have to be sent to its proper PE for storage. We arrange that each PE in turn acts in an active state, broadcasting the terms that it holds. Some degree of global synchronization (and hence cost) is needed in the handshakes that control which PE's take this active role and which remain passive. To improve overlap between communication and computation we make use of the MPI non-blocking persistent communication and through that arrange our own buffering of messages.

There are two communication levels; one-to-all communication of the current work terms, and asynchronous one-to-one communication of storage messages that have hashed to a particular PE. All messages are the same size, and bandwidth requirements are dependent on the frequency and number of messages to each PE. While the communication of current term from master to slaves may be implemented in a broadcast with synchronization side-effect, the hashed terms are random 'events'. The second type of message requires anticipation of an asynchronous event and a decision of when to handle it. We expect requests for memory balancing to be handled as soon as possible to free the receive buffer for a possible next message from the same source.

3 Evaluation and Conclusions

We have been measuring the communication traffic generated by our randomized distribution algorithm. To verify the fine-grain patterns in which messages interleave and also to assist with debugging we have a version of the code where each processor writes a time stamped message to its output file indicating a sent message, and its destination. All messages generated are the same size therefore bandwidth requirements are dependent only on the frequency and number of messages to each destination. This testing framework also makes it easy to insert arbitrary extra delays into selected parts of the processing to explode deadlock and buffer overflow potential. Sample traces will be included as part of the conference presentation of this work, as will be timings for the system running on the SR2201 and showing that although move to a distributed memory system leads to an significant initial overhead compared with the costs of

running on a single processor, for large polynomial calculations our scheme can show speed-up as well as the ability to perform calculations that would otherwise be too big to complete.

We have described structures that support efficient parallel algorithms for symbolic computation on distributed memory architectures. The hashing mechanisms generate communication patterns that counter intermediate expression swell, but introduce some extra synchronization and event-handling requirements. Recent work on solution of sparse systems of equations indicates their viability and they can be refined further for matrix computations. The structures and mechanisms discussed here are also amenable to vector operations and future work would explore their combination with pseudo-vector capabilities of the Hitachi SR2201 in manipulation of symbolic data.

References

[1] Dinda, P. A., Garcia, B. M., and Leung, D.-S. The measured network traffic of compiler-parallelized programs. Tech. rep., Carnegie Mellon University School of Computer Science, 1998. Techincal Report CMU-CS-98-144.

[2] Geist, G., Kohl, J., and Papadopoulos, P. Pvm vs mpi: A comparison of features. In *Calculateurs Paralleles* (1996).

[3] Gustavson, F., and Yun, D. Y. Y. Arithmetic complexity of unordered sparse polynomials. In *SYMSAC'76* (August 1976), R. D. Jenks, Ed., SIGSAM, ACM, pp. 149–153.

[4] Hitachi Ltd. *A tutorial for parallel programming*, 1995.

[5] Kumar, V., Grama, A., Gupta, A., and Karypis, G. *Introduction to Parallel Computing:Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1994.

[6] Miller, R., and Stout, Q. F. *Parallel Algorithms for Regular Architectures: Meshes and Pyramids*. The MIT Press, Cambridge, Massachusetts, 1996.

[7] Norman, A., and Fitch, J. Cabal: Polynomial and power series algebra on a parallel computer. In *Proceedings of PASCO 1997* (1997).

[8] Pacheco, P. S. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997.

[9] Sasaki, T., and Kanada, Y. Parallelism in algebraic computaion and parallel algorithms for symbolic linear systems. In *Proceedings of 1981 ACM Symposium on Symbolic and Algebraic Computation* (1981), P. S. Wang, Ed., pp. 160–167.

[10] Smit, J. New recursive minor expansion algorithms: A presentation in a comparative context. In *Proceedings of International Symposium on Symbolic and Algebraic Manipulation* (1979), E. W. Ng, Ed., Lecture Notes in Computer Science 72, Springer-Verlag, pp. 74–87.

[11] Smit, J. A cancellation free algorithm, with factoring capabilities, for the efficient solution of large sets of equations. In *Proceedings of the 1981 ACM Symposium on Symbolic and Algebraic Computation* (1981), P. S. Wang, Ed.

[12] Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., and Dongarra, J. *MPI:The Complete Reference*. The MIT Press, Cambridge, Massachusetts, 1996.

[13] Tobey, R. Experience with formac algorithm design. In *Communications of the ACM* (1966), pp. 589–597.