Automatic Data and Computation Partitioning on Scalable Shared Memory Multiprocessors

Sudarsan Tandri and Tarek S. Abdelrahman

Department of Electrical and Computer Engineering
The University of Toronto
Toronto, Ontario, Canada, M5S 3G4
e-mail: {tandri,tsa}@eecg.toronto.edu

1 Introduction

Scalable Shared Memory Multiprocessors (SSMMs) are becoming increasingly popular as platforms for parallel scientific computing. Recent commercial systems such as the Convex Exemplar and the Cray T3E offer not only scalability previously exclusive to distributed memory multiprocessors, but also the convenience of a single coherent view of memory. The presense of shared memory initially suggests that parallelizing compilers for SSMMs need not be concerned with the data management issues that compilers for distributed memory must contend with. However, the non-uniformity of memory accesses and limited operating system data management policies suggest that compilers should play a more active role in data management on SSMMs. A data partitioning based approach to data management can improve application performance on SSMMs [1].

In this paper, we address the problem of automatically deriving data and computation partitions for scientific applications on SSMMs. The problem of deriving optimal partitions is NP-hard [2]. Additionally, the suitability of data and/or computation partitions depends not only on data access patterns, but also on the characteristics of the target multiprocessor. A number of approaches have been proposed to tackle this problem. Bixby, et al. [3] formulate a 0–1 integer programming problem to represent all possible data partitions and their associated costs. The 0–1 problem is then solved to obtain optimal partitions. Anderson and Lam [2] propose a more efficient algebraic framework, but require run-time profiling to determine commonly executed loops. In both cases, the optimality of partitions is based on communication cost; partitions that result in less communication are considered superior.

We present a framework for deriving data and computation partitions on SSMMs. We show that communication cost alone is not adequate to assess the appropriateness of data and computation partitions; shared memory effects such as cache affinity, false sharing, synchronization and contention must also be taken into account. Furthermore, the presense of shared memory hardware makes the use of the owner-computes rule unnecessary; the performance of some applications benefit from relaxing this rule. We describe an algorithm for deriving data and computation partitions on SSMMs taking shared memory effects into account. The algorithm is computationally efficient compared to previous approaches and does not rely on run-time profiling. Experimental results from a prototype implementation of the algorithm demonstrate its effectiveness in parallelizing standard benchmarks and the necessity of taking shared memory effects into account. The results also demonstrate the computational efficiency of our framework.

2 The Algorithm

Partitions are specified by the selection of a processor geometry, the assignment of a distribution attribute to dimensions of arrays (and to parallel loops), and a mapping between array dimensions (and parallel loops) to the dimensions of the processor geometry. We introduce a new set of distribution attributes that extend well-known attributes (such as HPF's [4]) to resolve alignment conflicts between data and computations. Furthermore, explicit association between the dimensions of arrays and the dimensions of the processor geometry is used to obtain new data partitions that reduce contention and synchronization in programs on SSMMs [5].

The algorithm derives computation and data partitions for programs in which parallel loops have been identified. First, computation partitions are derived based on data access patterns in the program using the NUMA-CAG. Each node in this graph represents either an array dimension or a parallel loop. An unweighted edge connects an array dimension node to a loop node if a subscript expression in the array dimension contains the loop iterator. Array access patterns are used to assign an initial distribution attribute to each node in the graph. An iterative algorithm that utilizes our new distribution attributes is used to ensure that any two connected nodes in the NUMA-CAG have the same distribution attribute. In addition, the algorithm also maps connected nodes onto the same dimension of the processor geometry. This enhances locality and minimizes communication by allocating data and computation to the same processor.

The above use of the NUMA-CAG results in static array partitions for the entire program. Such partitions may be not be the best choice for arrays that require different partitions in different loop nests. The partitioning of *only* these arrays is re-evaluated by considering all possible partitions for each array individually using depth-first search with pruning. For example, if an array `A` requires two different partitions `D1` and `D2` in two loop nests, possible partitions evaluated are: replication, a static `D1` partition, a static `D2` partition, the static partition determined by the NUMA-CAG, and dynamic partitions obtained by re-partitioning the array between the two loop nests. The cost of a given data partition is computed using knowledge of cache, local and remote memory accesses costs as well as the costs of contention and synchronization (incurred when a loop is executed in wavefront due to data dependencies or to avoid contention). These costs are determined by a combination of analytical models and empirical evaluation of the target machine. The number of processors along each dimension of the processor geometry is selected to enhance cache affinity by examining possible processor geometry combinations for a given number of processors and the selected dimensionality of the processor geometry.

The computational complexity of the algorithm is reduced by selecting a dimensionality of the processor geometry based on the parallelism in the program, by only re-evaluating the partitions of the subset of arrays that may benefit from re-evaluation, and by using pruning to limit the number of partitions examined in each re-evaluation.

3 Results and Comparisons

The speedup of applications (only three applications are used here due to space limitations) on three SSMMs is shown in Figure 1. The speedup of the applications with data partitions is higher to the the speedup of the applications when data management is delegated to the operating system.

The impact of shared memory effects on the execution time of the well-known ADI benchmark is shown in Figure 2 for the Hector multiprocessor. A static distribution

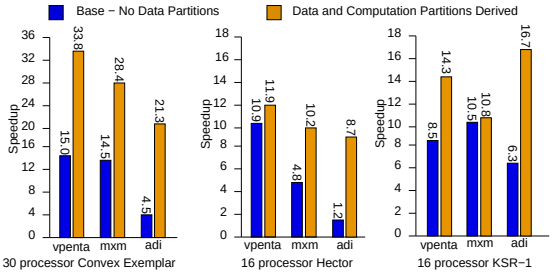

Fig. 1. Speedup of applications on three SSMMs.

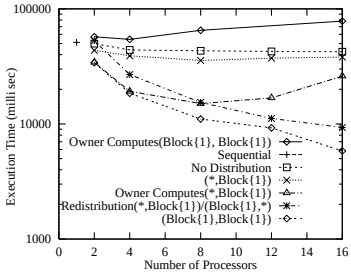

Fig. 2. Execution time of `ADI`.

dictated by either the first or second set of loop nests of the application results in contention and wavefront synchronization. The cost of data repartitioning between the two sets does not justify its use either. The best performance is obtained using our new data partition and a relaxed computation rule.

The computational efficiency of our framework is favorable compared to that of Bixby et al.'s approach. For example, data and computation partitions for the `vpenta` and `mxm` benchmarks are obtained directly from the NUMA-CAG (i.e., without re-evaluation), and are obtained for `ADI` after examining 116 possible partitions. This compares to solving 0–1 integer programming problems of sizes 2595×338, 170×59, 175×65, respectively. The partitions are obtained without run-time profiling.

4 Concluding Remarks

This paper presented an algorithm to automatically derive data and computation partitions taking into consideration shared memory effects. Experimental results indicate that these effects must be taken into account, and that the partitions derived by our algorithm improve the performance of the applications. A complete description of the algorithm and results can be obtained at *http://www.eecg.toronto.edu/~tsa/ca.ps*.

References

1. T. Abdelrahman and T. Wong, "Distributed array data management on NUMA multiprocessors," *Proc. of SHPCC*, pages 551–559, 1994.
2. J. Anderson and M. Lam. "Global optimization for parallelism and locality on scalable parallel machines," *Proc. of Conf. on Prog. Lang. Design and Implementation*, pp. 112–125, 1993.
3. R. Bixby, K. Kennedy, and U. Kremer. "Automatic data layout using 0–1 integer programming," *Proc. of PACT*, pp. 111–122, 1994.
4. C. Koelbel et al. *The High Performance Fortran Handbook*, The MIT Press, Cambridge, MA, 1994.
5. S. Tandri and T. Abdelrahman. "Computation and data partitioning on scalable shared memory multiprocessors," *Proc. of PDPTA*, pp. 41–50, 1995.

This article was processed using the LaTeX macro package with LLNCS style