MATHEMATICAL PROGRAMMING APPROACH FOR STATIC LOAD BALANCING OF PARALLEL PDE SOLVER

SHUICHI ICHIKAWA
Department of Knowledge-based Information Engineering,
Toyohashi University of Technology
Toyohashi 441-8580, JAPAN
ichikawa@tutkie.tut.ac.jp

TAKAMITSU KAWAI , TOSHIO SHIMADA
Department of Information Electronics,
School of Engineering, Nagoya University
Nagoya 464-8603, JAPAN
{kawai,shimada}@nuee.nagoya-u.ac.jp

Abstract

A static load-balancing scheme is discussed for numerical simulation system NSL, which automatically generates parallel solver of partial differential equations, PDE, from high level description of problem. NSL partitions computational domain into multiple blocks, and allocates processors optimally for each block in accordance with computation and communication cost. This allocation problem is formulated as a combinatorial optimization problem, and solved by branch-and-bound method. Though it is impractical to solve large problems by this method because of combinatorial explosion, this paper also describes an effective method to derive sub-optimal solution in practical time by limiting search space. The error of this approximation is less than 15% under reasonable condition. Elapsed time for combinatorial optimization is measured in numerical simulations to induce the estimation equation. The method presented here is widely applicable by adapting evaluation function for each purpose.

1. INTRODUCTION

Authors have been developing a numerical simulation language NSL [1], by which parallel simulation programs can be automatically generated from high level description of PDE (Partial Differential Equations) problems. There are many parallel processing systems for PDE (e.g. [2][3][4]) but only sub-optimal static load balancing has been implemented, because it is a very difficult combinatorial optimization problem [5][6].

There are researches on parallelizing compilers that optimize execution time considering both computation and communication. In particular, PARADIGM compiler [7] optimizes execution time by using convex programming [8][9]. However, the estimation model in PARADIGM is still too much simplified and the derived solution is sub-optimal, because optimization problem is relaxed to continuous domain to make the problem easy to solve. Contrary, a combinatorial optimization problem is solved in our research to derive the optimal solution in discrete domain. Such method is strongly desired for maximal performance and for evaluation of approximation algorithms on an absolute scale.

The purpose of this research is to optimize the execution time of parallel PDE solver, taking both computation and communication into consideration. In the following sections, domain partitioning and processor allocation problems are formalized as a combinatorial optimization problem, and solved by branch-and-bound method.

2. MODEL OF COMPUTATION

Two notable features of NSL are boundary-fitted coordinate system and multi-block method to describe physical domain. Physical domain with complicated boundary is divided into multiple components, each of which is mapped to a rectangular computational domain (*block*) topologically connected each other (Fig. 1). By this mapping, boundary condition can be precisely given and local grid control gets easier. From the computational point of view, rectangular computational domain is preferable for high-performance processing with vector and parallel processors. To utilize high degree of data-parallelism, explicit scheme is used in NSL to solve PDE. In explicit scheme, calculation of each node can be performed in parallel, followed by communication to exchange the values.

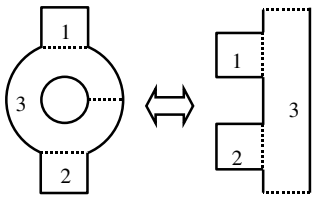

FIG.1 PHYSICAL AND COMPUTATIONAL DOMAIN

Each block is partitioned into congruent rectangular sub-blocks (see Fig. 2), which are distributed among processors for parallel computation, causing communications between adjacent sub-blocks on their borders. This communication latency can be overlapped with calculation by properly arranging the order of nodes in calculation. Processor first calculates border part, then sends them out by non-blocking communication, and then calculates the rest of nodes.

Presented in the 16th IASTED International Conference APPLIED INFORMATICS (23 Feb. 1998 at Garmisch-Partenkirchen, Germany)

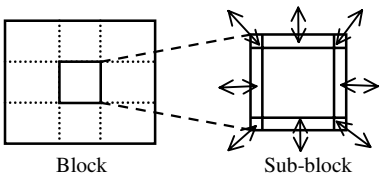

FIG. 2 PARTITIONING A BLOCK INTO SUB-BLOCKS

Communication latency is hard to estimate, because it depends on such factors as network implementation, processor allocation, and network contention. In this paper, parallel computer with uniform processors and uniform network is assumed to keep optimization problem simple. Hence physical allocation of processors is temporary ignored. Dynamic aspects of network (e.g. contention) are also ignored here. With these simplifications, the problem is formulated as finding the best allocation of *n* equivalent processors among *m* distinguishable blocks to minimize execution time. This simple optimization problem still involves nCm combinations, i.e. $O(n^m)$ search space.

Note that NSL can keep the number of blocks (m) relatively small by using boundary-fitted coordinate system. On the other hand, we are usually interested in large-scale parallel computer (big n). Therefore, the relationship $m << n$ is implicitly assumed in the following discussion.

3. ESTIMATION OF EXECUTION TIME

This section outlines how to estimate execution time when a processor allocation is given. The evaluation function T is the execution time for iteration. Let T_i be the execution time of the *i*-th block B_i, and n_i be the number of processors allocated to B_i. B_i is partitioned into n_i congruent rectangular sub-blocks (Bs_i), one for each of n_i processors. Remind that Bs_i is not uniquely determined, because n_i has generally many way of factorization. We have to choose the best factorization of n_i that makes T_i minimal.

Under these conditions, T_i is given by

$$T_i = Tb_i + Ts_i + \max_i(Ta_i, Tc_i)$$

where Tb_i is the calculation time for border part of Bs_i, Ta_i is the calculation time for the rest of Bs_i, Ts_i is the setup time for communication of Bs_i, and Tc_i is communication time of Bs_i. In this paper, all of Tb_i, Ta_i, and Ts_i are simply assumed to be the linear function of the number of corresponding nodes. Tc_i depends on various factors and can appear as various kind of function, but this paper only deals with an illustrative example such as latency is monotonically increasing function of n_i and bandwidth is constant. Other kind of Tc_i is also possible and feasible for our method.

With these notations, the whole allocation problem is formulated as follows.

Minimize $\quad T = \max_i T_i \quad (0 \le i \le m-1)$

Subject to $\quad n \ge \sum_{i=0}^{m-1} n_i$

4. BRANCH-AND-BOUND

As in many combinatorial optimization problems, branch-and-bound method is adopted to limit search space. Suppose that we have a temporal feasible solution $\overline{T}$. T_i takes the minimum value at intermediate n_i, because Ta_i monotonically decreases and Tc_i monotonically increases when n_i gets larger (Fig. 3). Therefore, lower and upper bounds of each block are derived from $\overline{T}$. The real T_i is discrete and jaggy, but instead we can use the lower bound of T_i that is continuous and smooth, which is derived by relaxation.

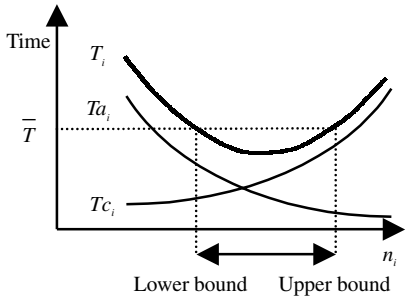

FIG. 3 FIND UPPER AND LOWER BOUNDS FROM SUBOPTIMUL SOLUTION $\overline{T}$

It is important for branch-and-bound method to get a good temporal solution earlier, because better temporal solution induces tighter bound that makes search space drastically small. So, it is a good idea to use an approximation algorithm at the first stage of optimization to cut hopeless branch roughly. The next section describes a simple and effective approximation algorithm.

5. APPROXIMATION ALGORITHM

It is essential for approximation algorithm to find a feasible sub-optimal solution in very short time. It is also desirable to give high-quality solutions, hopefully with performance guarantee. This section presents an approximation strategy, in which the upper limit $\overline{n}_i$ is set for search of n_i. The basic idea is to reserve the share of processors for each block, and permit to discard surplus. Let H_i and W_i be the height and width of B_i, then $\overline{n}_i$ is defined by the following equation.

$$\overline{n}_i = \left\lfloor (n-m)H_iW_i \Big/ \sum_{i=0}^{m-1} H_iW_i \right\rfloor + 1$$

Searching between 1 and $\overline{n}_i$, the approximated solution of n_i is that makes T_i minimal. Repeating this procedure for all blocks, a feasible sub-optimal allocation is derived in $O(n)$ time. This is negligible compared to $O(n^m)$ search time for optimal allocation, hence applicable to large problems. The quality of derived sub-optimal solution is presented in the next section.

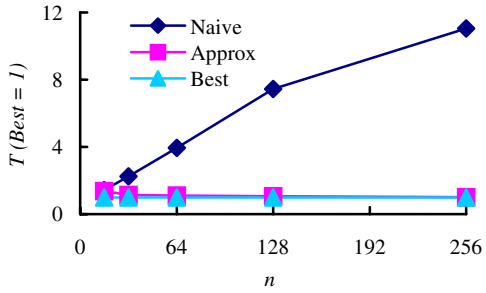

FIG.4 EXECUTION TIME ON PARALLEL PROCESSORS

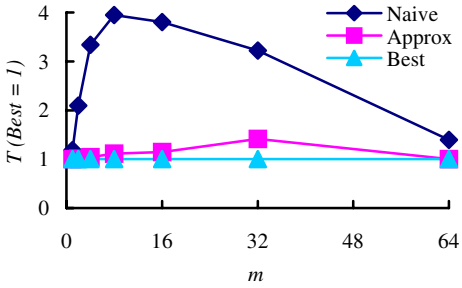

FIG. 5 EXECUTION TIME ON 64 PROCESSORS

6. EVALUATION

This section shows the results of numerical simulations. In simulations, three static load-balancing schemes are evaluated. The first is Best, which is the optimal allocation derived by combinatorial optimization. The second is Approx, which is the approximation derived by the method in the previous section. The last is Naive, in which every block is equally partitioned and distributed to all n processors. Naive is the best to balance the calculation, though accompanied by massive communication among processors. Computational domains were randomly generated, and the result is the arithmetic mean of 100 trials, normalized by the Best value. Simulation parameters are set to typical values in commercial parallel processors.

Fig. 4 shows the effect of using larger parallel computers in case m is 8. Approx and Best is far better than Naive, because Naive incurs heavy communication latency. The error of Approx against Best is less than 14.8% ($n \geq 32$). Fig. 5 shows the effect of using more blocks on 64 processors. In most cases, Best and Approx is a few times better than Naive. When m is small, Best and Naive is similar because each block is distributed to many processors, which causes heavy network traffic. When m is comparable to n, Best scores bad because it has only limited choices for load balancing. The error of Approx is less than 14.6% ($m \leq 16$). In both cases, $n \geq 4m$ seems satisfactory condition for Approx algorithm.

Elapsed time for optimization (T_{ex}) was measured in numerical simulations to induce the estimation equation shown below.

$$\log_{10} T_{ex} = (0.084m + 1.18)\log_{10} n - 5.24$$

Unit is second, and the platform is Intel Pentium-Pro 200 MHz with FreeBSD. T_{ex} is almost proportional to n^m as predicted. From this equation, we can see the extent of m and n that can be solved in realistic time.

7. CONCLUSION

Our research enabled the optimal processor allocation for practical computation. Although it is still impractical to solve very large optimization problem because of combinatorial explosion, our approximation algorithm can find good sub-optimal allocation in negligible time. One of the obvious merits of our approach is that the quality of such approximation algorithm can be verified quantitatively by comparing to the optimal solution. Another merit is wide applicability to problems by adapting evaluation function. Our evaluation function is not necessarily to be convex or contiguous. Measurement results can be used for evaluation function in optimization.

Evaluations on real parallel computer platforms are now under way to confirm and to improve our optimization scheme. Physical allocation of processors should be considered in optimization process. Dynamic aspects including network contention and collision should also be considered. It is also desirable to handle distributed computational environment, which consists of non-uniform processors and relatively slow networks.

References

[1] T. Kawai, S. Ichikawa, and T. Shimada, NSL: High Level Language for Parallel Numerical Simulation, *Transactions of Information Processing Society of Japan*, Vol. 38, No. 5, pp. 1058-1067 (1997).

[2] E. N. Houstis and J. R. Rice, Parallel ELLPACK: A Development and Problem Solving Environment for High Performance Computing Machines, In Gaffney, P. W. and Houstis, E. N. (eds.), *Programming Environments for High-Level Scientific Problem Solving,* Elsevier Science Publishers B. V., pp. 229-243 (1992).

[3] K. Suzuki, et al., DISTRAN system Implementation on Parallel Computers, In *Joint Symposium on Parallel Processing '91*, pp. 301-308 (1991).

[4] T. Okochi, C. Konno, and M. Igai, High Level Numerical Simulation Language DEQSOL for Parallel Computers, *Transactions of Information Processing Society of Japan*, Vol. 35, No. 6, pp. 977-985 (1994).

[5] N. Chrisochoides, E. Houstis, and J. Rice, Mapping Algorithms and Software Environment for Data Parallel PDE Iterative Solvers, *Special Issue of the Journal of Parallel and Distributed Computing on Data-Parallel Algorithms and Programming*, Vol. 21, No. 1, pp. 75-95 (1994).

[6] N. Chrisochoides, N. Mansour, and G. Fox, Comparison of optimization heuristics for the data distribution problem, *Journal of Concurrency Practice and Experience*, Vol. 9, No. 5, pp. 319-344 (1997).

[7] P. Banerjee et al., The PARADIGM Compiler for Distributed-Memory Multicomputers, *IEEE Computer*, Vol. 28, No. 10, pp. 37-47 (1995).

[8] S. Ramaswamy, S. Sapatnekar, and P. Banerjee, A Convex Programming Approach for Exploiting Data and Functional Parallelism on Distributed Memory Multicomputers, In *Proceedings of the 23rd International Conference on Parallel Processing*, St. Charles, IL, pp. II:116-125 (1994).

[9] S. Ramaswamy, *Simultaneous Exploitation of Task and Data Parallelism in Regular Scientific Applications*, Ph.D. Thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, IL (1996).