COMBINED MPI/OPENMP IMPLEMENTATIONS FOR A STOCHASTIC PROGRAMMING SOLVER

D. ROTIROTI*, C. TRIKI** AND L. GRANDINETTI*

* *University of Calabria*
Department of Electronics, Informatics and Systems
87030 Rende (CS) - ITALY

** *University of Lecce - Department of Mathematics*
Via Arnesano 73100 Lecce - ITALY
E-mail: chefi.triki@unile.it

Stochastic Linear Programming (SLP) is an effective tool to deal with problems for which some of the input data are uncertain. These problems are typically characterized by a very high number of variables and constraints and the use of conventional computational resources is usually inappropriate for their solution. Parallel systems are required in order to achieve high level of efficiency in reasonable time. In this paper we propose several MPI and OpenMP implementations on the basis of which we develop a two-level parallel solver that take advantage from the features offered by both the standards. Experimental results show that the two-level solver could be faster than any other basic algorithm.

1 Introduction

In several real world applications there is the need for solving problems with input data which are not known with certainty; these data are usually modelled as random variables, with some probability distribution. Stochastic Linear Programming (SLP) is an effective tool to deal with this kind of problems giving the optimal solution across the different events that could be observed.

We focus on two-stage SLP problems, with a finite and discrete distribution of the random variables. The realization of the random variables consists in the occurrence of one of the N possible events, known as *scenarios*. In this formulation, decision variables are divided into two groupes: *anticipative* and *adaptive* (or first- and second-stage) variables. The anticipative decisions, denoted with x, are taken before knowing the random values, while the adaptive ones are determined after the realization of the random event. For each scenario $l = 1, \ldots, N$ a corresponding second-stage vector y_l is calculated containing the relative decisions.

The two-stage SLP problem with N scenarios can be modelled in the following general form (for detailed description of the model the reader can refer, for example to[1]) :

$$
\begin{aligned}
\min \;\; & c_0^T + \sum_{l=1}^{N} p_l \, c_l^T y_l \\
s.t. \;\; & A_0 x = b_0 \\
& T_l x + W_l y_l = h_l \qquad\qquad l = 1, \ldots, N \\
& x, y_l \geq 0 \qquad\qquad\qquad\;\; l = 1, \ldots, N.
\end{aligned} \tag{1}
$$

As can be easily noted, the number of variables and constraints increases considerably as the number of scenarios N increases. For most of the real-world applications N is very big so the resulting problem can not be solved using conventional sequential systems. Moreover, for many nowadays problems a real-time solution still remains a challenge. The use of parallel machines is necessary to achieve high level of efficiency in reasonable time.

Not many works have been published on the parallelization of SLP techniques since this field is still considered in its infancy. The main interest has been concentrated on parallel algorithms deriving from the implementation of primal-dual Path Following (PF) interior point methods. Within the PF algorithm it is easy to split the overall problem in order to handle N independent sub-blocks of the constraint matrix each block corresponding to one scenario. De Silva and Abramson have implemented the PF algorithm on a Fujitsu AP1000 with 128 distributed memory processors[2]. Jessup, Yang and Zenios have parallelized the technique of Birge-Qi[3] factorization(BQ) on an Intel iPSC/860 hypercube[4]. Their effort was continued by the implementation done by Yang and Zenios on a Connection Machine CM-5e in order to develop a parallel PF solver[5]. All these implementations are based on the message passing paradigm. More recently, Beraldi, Grandinetti, Musmanno and Triki have proposed both (an efficient) PVM and (a poorly tuned) OpenMP implementations of the BQ factorization on an Origin2000 machine[1].

This paper will continue these efforts towards developing efficient message-passing (MPI) and shared-memory (OpenMP) implementations of the BQ factorization. Moreover, we propose novel two-level parallel implementations in which we combine the more attractive features of the two standards in order to reduce computation time.

It is worthwhile noting that the benifit of the two-level parallelism has been recently recognized in other applications like in CGWAVE[6]. But no tentatives have been published in this direction in the stochastic programming field.

In this paper we will describe first the basic MPI and OpenMP implementations. The second section will be devoted to the two-level implementations

and the discussion of their results with special regard to the issues of scalability and portability. A bref summary will conclude the paper.

2 Basic Parallel Implementations

In this section various parallel implementations of the PF algorithm are presented. All the versions are based on the parallelization scheme of the BQ factorization as described in our previous paper[1]. In this scheme most of the computation tasks, corresponding to the independent scenarios, are carried out in parallel on the available processors. Some communication and synchronization points are necessary in order to form the scenarios-coupled matrices. (Interested readers are invited to refer to the above mentioned paper for details.)

2.1 MPI Implementations

A first MPI version of the solver has been developed by using a static load balancing policy. The work is divided assigning a block of scenarios to each worker, so that the difference between the size of blocks is as small as possible. Each worker can easily recognize its position within the group and accordingly its slice of work without any communication.

Every processor computes its part of scenarios, whereas the first-stage stuff is done redundantly by all the processors. This is more efficient than using a single processor to operate on first-stage computation and then broadcast the results to all the others. A correct implementation of the algorithm requires also a reduction in order to form the scenarios-coupled matrices by using the sum operator.

Figure 1 shows the speedups of the solution of test problem `scagr7` with 864 scenarios selected from the SLP library of Holmes[7]. Detailed wall-clock times of other test problems are shown in table 1. The failure in the solution of the `scagr7.936` problem and the deterioration of performance in the case of `scsd8.504` are due to computational difficulties caused by the increase of the problem size after the replication of the first-stage variables.

Even though these results were satisfactory, we made the tentative of developing another version of an MPI implementation based on the master-worker model. In this scheme the workload is balanced dynamically through the introduction of an additional task (Boss) that starts by the distribution of the input data on the available workers. When a worker is idle it sends a request to the boss who will assign it the next free block to be computed. In order to maintain a high level of efficiency in this scheme particular attention

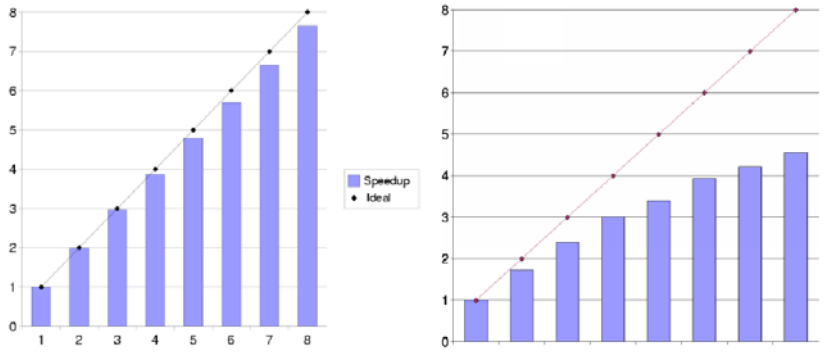

Figure 1. Speedups with MPI (left) and with fine grained OpenMP (right)

Table 1. Computation time (seconds), MPI version

Problem	**Variables**	**Constraints**	1 cpu	2 cpu	4 cpu	8 cpu
scagr7.864	69180	67407	9.64	4.84	2.49	1.26
scagr7.936	74940	73023	10.3	5.16	2.65	-
scsd8.432	121170	69130	22.60	11.53	6.03	3.94
scsd8.504	141330	80650	25.40	13.00	6.93	45
sctap1.480	92304	74910	29.34	14.92	7.60	4.07

should be addressed to the critical zone of synchronization. During the reduction each task caches the corresponding data handled during the first part of the iteration and then continues to operate on the same scenarios after the reduction. With this cache friendly execution communication overhead is kept as low as possible.

The dynamic implementation ensures a good load balancing and achieves encouraging results specially on non dedicated and heterogeneous systems. However, using our parallel machine and running in a dedicated environment the static version performs with a slight advantage with respect to the dynamic implementation.

2.2 OpenMP Implementations

Developing a shared-memory version by simply introducing the parallel directives of OpenMP in the sequential code seems to be an easy and fast task. However, optimising the code and using the incremental level of parallelizing

Table 2. Computation time (seconds) with the coarse grained OpenMP implementation

Problem	**Variables**	**Constraints**	1 cpu	2 cpu	4 cpu	8 cpu
scagr7.864	69180	67407	9.60	5.07	2.53	1.44
scagr7.936	74940	73023	10.18	5.43	2.69	1.48
scsd8.432	121170	69130	22.73	12.33	6.44	6.23
scsd8.504	141330	80650	25.36	13.73	7.11	6.94
sctap1.480	92304	74910	29.33	15.08	7.79	4.19

features offered by OpenMP in order to ensure high performance necessitate a long and painful implementation effort.

In a first version parallelization was introduced with respect to the second-stage components by splitting the workload at the inner loops level. At each update step of y in the iterative BQ procedure, a component-wise division of the vector is performed among the available processors. The distribution of input vectors is done statically among the threads in order to maximize the reuse of the data.

Intuitively, this fine grained version is expected to allow better integration with the MPI layer in a combined implementation.

The experimental results for the test problem `scsd8` with 508 scenarios depicted in Figure 1 show that as the number of threads increases the performance deteriorates. The same trend is observed for the other test problems. This is mainly due to the high number of the synchronization points needed at the end of each parallel loop when data dependency between two consecutive loops is met.

A possible way to overcome this deficiency is to insert OpenMP directives at the outer iteration level, i.e. scenario-wise rather than component-wise parallelism. This coarse grained version tracks the same parallel scheme as the one described in the MPI implementation and presents the same advantages, but at the same time suffers from the same bottleneck caused by the reduction operation.

A collection of computational results of this version is shown in Table 2. The same results corresponding to the problem `scagr7` with 936 scenarios are shown in Figure 2 as well.

3 Two-level parallelism implementations

Combining the MPI and OpenMP implementations have the objective of taking a full advantage from parallel systems based on a distributed shared mem-

Table 3. Execution time (seconds) with two-level implementations

	MPI/OpenMP (coarse grained)				MPI/OpenMP (fine grained)			
Problem	**8 - 1**	**4 - 2**	**2 - 4**	**1 - 8**	**8 - 1**	**4 - 2**	**2 - 4**	**1 - 8**
scagr7.864	1.26	1.27	1.32	1.44	1.26	1.51	2.01	2.95
scagr7.936	-	1.39	1.40	1.48	-	1.62	2.17	3.49
scsd8.432	3.94	3.45	3.41	6.23	3.94	5.59	8.14	13.30
scsd8.504	45	3.99	3.96	6.94	45	6.32	9.23	15.65
sctap1.480	4.07	4.12	4.15	4.35	4.07	5.91	7.85	11.48

ory architecture.

Two versions have been developed: MPI with the fine grained OpenMP version from one side, and MPI with the coarse grained version from the other side.

In the first version the MPI layer implements a scenario-wise parallelism, whereas the OpenMP directives are introduced at a component-wise level parallelism.

This version of the solver has been ran for each test problem by varying simultaneously the number of MPI tasks and OpenMP threads whereas the number of processors remains unaltered (i.e. 8 processors).

As can be shown in Table 3, the execution time gets longer as the number of OpenMP threads increases. It is clear that this two-level version suffers from the same defects of the fine grained OpenMP implementation.

More interesting results have been obtained by combining the MPI implementation with the coarse grained OpenMP version. In this case the whole number of scenarios is divided among the MPI tasks and each task splits its slice among the OpenMP threads. The most critical region, i.e. the reduction, is done in two different phases: an inter-threads reduction via OpenMP directives and then an MPI call among the tasks.

Table 3 reports the solution time by using this second version of the two-level implementation. These results show that the two-level version outperforms the pure OpenMP implementation for all the test problems. Moreover, for some test problems the solution time can be even lower than that of the MPI code. This is the case of well structured problems like `scsd8` with 432 scenarios (depicted in Figure 2) in which the best execution time is measured with two MPI tasks and four OpenMP threads each.

Another remarkable advantage is that both the two-level versions are able to overcome the failure observed in the solution of the `scagr7.936`. This

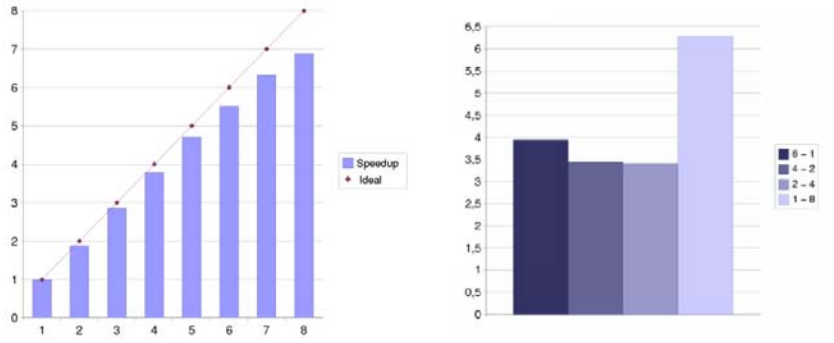

Figure 2. Speedups with coarse grained OpenMP (left) and execution time in seconds with the two-level parallelism (right)

advantage is expected to persist even in the solution of bigger problems on larger parallel systems.

3.1 Portability

In order to ensure a perfect portability of our implementations only standard features of the MPI and OpenMP libraries have been used. Particular attention was devoted in order to avoid the use of the extensions offered by the SGI implementation of the OpenMP on the Origin2000. These extensions, such as data placement directives, even though can improve the performance of the shared-memory versions have the disadvantage of limiting the portability of the code to Origin-like systems.

3.2 Scalability

As mentioned above, the two-level implementations are more scalable with respect to both the pure MPI and OpenMP codes. While the MPI version suffers from the explosion of the problem size, OpenMP implementations are very sensitive to the number of synchronization points.

On the other hand, some interesting remarks on the scalability of each of the two-level versions can be drawn. More specifically, the second version (MPI-coarse grained OpenMP) not only performs better but also it is expected to scale better in the case of problems with high number of scenarios. On the contrary, the MPI-fine grained version is expected to scale better for problems

characterized by high number of second-stage variables and few scenarios. Indeed, while the second version can use not more than N computational units the first two-level parallelism code is able to use $N \cdot n_2$ processors, where n_2 is the size of each vector $y_l, l = 1, \ldots, N$.

4 Conclusions

In this paper we propose new implementations using two-level parallelism to solve SLP problems. Several versions have been developed as a result of the possible combinations of MPI static and dynamic implementations from one side and OpenMP fine and coarse grained parallelism from the other side. For the test problems we considered, and using the multiprocessor machine Origin2000, the best results have been obtained combining the static MPI version with the coarse grained OpenMP implementation. By using other systems such as cluster of multiprocessor machines different conclusions could be expected. This may be the subject of further investigations in this field.

References

1. P. Beraldi, L. Grandinetti, R. Musmanno, and C. Triki. Parallel algorithms to solve stochastic linear programs with robustness constraints. *Parallel Computing*, 26:1889–1908, 2000.
2. A. De Silva and D. Abramson. Parallel algorithms for solving stochastic linear programs. In A. Zomaya, editor, *Handbook of Parallel and Distributed Computing*, pages 1097–1115. McGraw Hill, 1996.
3. J. R. Birge and L. Qi. Computing block-angular Karmarkar projections with applications to stochastic programming. *Management Science*, 34(12), 1990.
4. E. R. Jessup, D. Yang, and S. A. Zenios. Parallel factorization of structured matrices arising in stochastic programming. *SIAM Journal on Optimization*, 4:833–846, 1994.
5. D. Yang and S. A. Zenios. A scalable parallel interior point algorithm for stochastic linear programming and robust optimization. *Computational Optimization and Applications*, 7:143–158, 1997.
6. S. W. Bova e altri. Dual level parallel analysis of harbor wave response using mpi and openmp. *The International Journal of High Performance Computing*, 14(1):384–392, 2000.
7. D. Holmes. A collection of stochastic programming problems. Technical Report 94–11, Department of Industrial and Operations Engineering, University of Michigan, 1994.