CENTER FOR
PARALLEL OPTIMIZATION

SERIAL AND MASSIVELY PARALLEL SOR ALGORITHMS FOR LARGE-SCALE ENGINEERING PROBLEMS

by

R. De Leone and M. A. Tork Roth

Computer Sciences Technical Report #1073

January 1992

Serial and Massively Parallel SOR Algorithms for Large–Scale Engineering Problems*

R. DE LEONE† M. A. TORK ROTH†

Abstract

In this paper we discuss the solution of several engineering problems using serial and parallel successive overrelaxation (SOR) methods. The sparsity structure of the problems considered here allowed us to efficiently implement the SOR algorithm on a Connection Machine CM–2, a massively parallel Single–Instruction–Multiple–Data machine. Computational results are reported for three classes of problems: obstacle problems, elastic–plastic torsion problems, and journal bearing problems. Problems with up to 4 million variables have been solved in a few minutes on a CM–2 with 16,384 processors.

Keywords: Quadratic Programs, Successive Overrelaxation, Massively Parallel Algorithms.

1 Introduction

Our concern here is the quadratic programming problem with simple bounds:

$$\begin{aligned} &\text{minimize} && \tfrac{1}{2}x^T M x + q^T x \\ &\text{subject to} && l \leq x \leq u \end{aligned} \tag{1}$$

where M is a symmetric, positive semidefinite nxn real matrix with positive diagonal entries and q, l, u, and x are n–dimensional vectors. Problems of this form arise in many physical and engineering applications such as contact and friction problems in rigid body mechanics, elastic–plastic torsion problems, and journal bearing lubrication [1, 2].

Various types of algorithms have been proposed and studied for the solution of these problems; for example, algorithms based on active set strategies and, more recently, interior point algorithms. An effective active set strategy for very large problems must add or drop many constraints at the same time. The recent algorithm proposed by Moré and Toraldo

*This material is based on research supported by the Air Force Office of Scientific Research Grant AFOSR-89-0410

†Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin Madison, 1210 West Dayton Street, Madison, WI 53706

[10] combines a conjugate gradient method to explore a face of the feasible polytope and a projected line search strategy to select a new face. The interior point algorithm proposed by Han, Pardalos and Ye [4] is (at least from a theoretical point of view) one of the most effective, requiring a total of $O(\sqrt{n}L)$ iterations, where L is the size of the input data of the problem. A review of the recent algorithms based on active set strategies can be found in [10, 9, 3]. We refer the reader to [4] for an outline of the results for interior point based algorithms.

In this paper we show that important large–scale engineering problems can be solved using both serial and parallel successive overrelaxation (SOR) methods [6, 7]. The effectiveness of the SOR algorithm in solving large sparse linear complementarity problems and linear programs derives in part from the fact that original problem data are never modified and the sparsity structure of the matrix M is preserved throughout the algorithm.

For all the problems considered here, the matrix M is pentadiagonal. This special sparsity structure of the matrix allowed us the effectively implement the SOR algorithm on the massively parallel Single–Instruction–Multiple–Data (SIMD) Connection Machine CM–2.

In a previous paper [3], we concentrated on the obstacle problem and results on the Connection Machine CM–2 and the MasPar MP–1 were reported. Two different implementations of the algorithm in Fortran 90 and C* were discussed. For larger problems the C* implementation was more efficient, reducing the computing time in some cases to 40% less than the Fortran 90 implementation. We attributed this to lower inter-processor communication costs in the C* implementation (see [3]).

In this paper, we report computational results for three classes of problems: obstacle problems, elastic–plastic torsion problems, and journal bearing problems. We compare the results of our algorithm with the results for the Han, Pardalos and Ye algorithm [4] and with the results reported by McKenna, Mesirov and Zenios in [11] for their implementation of the Moré and Toraldo [10] algorithm on the Connection Machine CM–2.

We briefly describe our notation now. Given the vectors l and u (with $l \leq u$) and a vector x all in $\mathbb{R}^n$, $x_\#$ will denote the vector with components $(x_\#)_i = \min\{u_i, \max\{l_i, x_i\}\}$. The scalar product of two vectors x and y in $\mathbb{R}^n$ will be denoted by $x^T y$. The symbol $:=$ denotes definition of the term on the left side of the symbol.

2 The serial SOR algorithm

In this section we will discuss a serial implementation of the SOR algorithm for the quadratic program with simple bounds (1).

A point $x \in \mathbb{R}^n$ solves the quadratic program (1) if and only if ([6])

$$x = (x - \omega E(Mx + q))_\#$$

for some positive diagonal matrix E and some $\omega > 0$. The above formula is the basis of the SOR algorithm. The successive overrelaxation algorithm constructs a sequence of iterates $\{x^k\}$ as follows:

SOR Algorithm

For any initial feasible $x^0 \in [l, u]$, generate the sequence $\{x^k\}, k = 0, 1, 2, \ldots$, as follows:

$$x_i^{k+1} := \left(x_i^k - \omega E_{ii} \left(\sum_{j<i} M_{ij} x_j^{k+1} + \sum_{j \geq i} M_{ij} x_j^k + q_i \right) \right)_{\#} \tag{2}$$

for $i = 1, 2, \ldots, n$.

In our implementation we used $E_{ii} = M_{ii}^{-1}$. Convergence of the algorithm can be established if the relaxation parameter ω is chosen in the interval $(0, 2)$ [5].

We refer the reader to [3] for more information on the serial implementation of the SOR algorithm including the data structure used for storing the matrix M and the choice of the relaxation parameter ω. For all the problems considered here, the value of ω was fixed to 1.95.

A simple but very effective "fixing–strategy" for the problem variables was implemented. During the updating process, the algorithm recorded if (and how many times) a particular variable remained fixed at its upper or lower bound. Based on this information, certain variables were held fixed for a number of iterations. A resetting step was performed every 30 iterations to verify the appropriateness of fixing these variables. We estimate that a 15% reduction in solution time was achieved by this strategy.

3 The massively parallel SOR algorithm

Although the SOR algorithm is inherently serial, two distinct components x_i and x_j (with $i < j$) can be concurrently updated using the above formula (2) provided that $M_{jr} = 0$ for all $r = i, i+1, \ldots, j-1$. This observation is the basis of our massively parallel implementation of the algorithm.

The quadratic programming problems considered in our parallel implementations arise as finite element approximations to elliptic variational inequalities. These approximations are obtained by triangulating the unit square, giving rise to a grid. In this context it is more convenient to describe our parallel implementation in terms of an $m \times m$ matrix X. Then $n = m^2$ and

$$X := \begin{bmatrix} X_{1m} & X_{2m} & \ldots & X_{mm} \\ & \ldots & \ldots & \\ X_{12} & X_{22} & \ldots & X_{m2} \\ X_{11} & X_{21} & \ldots & X_{m1} \end{bmatrix}, \quad x := [X_{11}, X_{21}, \ldots, X_{m1}, X_{12}, \ldots, X_{m2}, \ldots, X_{1m}, \ldots, X_{mm}].$$

At each grid point the value of a piecewise linear function at the vertices of the triangulation of the problem domain is stored (see also [10]).

To update a particular component X_{ij}, the SOR algorithm requires the value X_{ij}, along with its north, south, east and west neighbors in X. This special structure allowed us to take advantage of the NEWS (North–East–West–South) communication grid of the Connection Machine CM–2.

In our C* implementation, the processors were organized in an $p \times p$ grid and assigned a $k \times k$ submatrix of X, where $n = p^2 \times k^2$. A standard red–black coloring [8] was imposed on X so that each processor would be active at every time step; all red components were updated simultaneously, followed by all black components.

4 Performance of the serial and parallel algorithms

The test problems considered are instances of three classes of problems: the obstacle problem, the elastic–plastic torsion problem, and the journal bearing problem. These three problems can be posed as the following constrained variational problem:

$$\min\{q(v) : v \in K\}.$$

For the obstacle problem and the elastic-plastic torsion problem, the objective function is given by

$$q(v) = 1/2 \int_{\mathcal{D}} \|\nabla v\|^2 \, d\mathcal{D} - f \int_{\mathcal{D}} v d\mathcal{D},$$
$$\mathcal{D} = (0,1) \times (0,1),$$

where ∇ is the Laplacian operator, and K is the subset of all functions v with compact support on $\mathcal{D}$ such that v and $\|\nabla v\|^2$ belong to the square integrable class $L^2(\mathcal{D})$.

For the obstacle problem, v varies between the bounds v_l and v_u given in Table 1 and the force $f = 1$.

Problem	v_l	v_u
1	$(\sin(9.2x_1)\sin(9.3x_2))^3$	$(\sin(9.2x_1)\sin(9.3x_2))^2 + 0.02$
2	$\sin(3.2x_1)\sin(3.3x_2)$	2000.0

TABLE 1: *Lower and upper bounds for the obstacle problems.*

For the elastic–plastic torsion problem, $f = c$ for some constant c, and the bounds on v are given by

$$\{|v(x)| \leq dist(x, \partial\mathcal{D}), x \in \mathcal{D}\}.$$

where $dist(\cdot, \partial\mathcal{D})$ is the distance function to the boundary $\partial\mathcal{D}$.

We considered the three cases $c = 5$, $c = 10$, and $c = 20$.

The journal bearing problem has an objective function given by

$$q(v) = 1/2 \int_{\mathcal{D}} (1 + \epsilon \cos\theta)^3 \|\nabla v\|^2 \, d\mathcal{D} - \epsilon \int_{\mathcal{D}} \sin\theta v d\mathcal{D},$$

where

$$\mathcal{D} = \{(\theta, y) : 0 < \theta < 2\pi, 0 < y < 2b\},$$

$$\{v \geq 0\}.$$

In our tests, we set $b = 10$, and used $\epsilon = 0.1$ and $\epsilon = 0.5$.

Finite element approximations give rise to a quadratic minimization problem with a finite number of variables. For all three classes of problems, the matrix M is a pentadiagonal matrix. For the obstacle problem and elastic–plastic torsion problem, M has diagonal entries of 4 and off–diagonal entries of -1. While the matrix M for the journal bearing problem has the same pentadiagonal structure, diagonal and off–diagonal entries are more complicated to compute.

In tables 2 to 11 the computational results for the serial and parallel version of the SOR algorithm are reported. The serial results were obtained on an IBM RISC 6000 POWERstation 550 while the massively parallel results were obtained on a Connection Machine CM–2 using 8K and 16K processors. All numerical computation was carried out in double precision.

The results for the obstacle problem are reported in Tables 2 to 5. Table 2 shows the number of iterations, solution time (in seconds) and number of free variables at the optimum (i.e., the number of variables neither at the at the lower or upper bound) for the first of two obstacle problems. The number of variables ranges from 10,000 to 490,000 which corresponds to a grid with the number of points varying from 100x100 to 700x700. The fifth column contains solution accuracy defined as:

$$\text{Accuracy} := \left\| x^* - (x^* - (Mx^* + q))_{\#} \right\|_{\infty}.$$

The column labeled HPY reports solution times for the same problems with the Han–Pardalos–Ye interior point algorithm [4] on a IBM 3090–600S supercomputer with Vector Facilities. In their implementation they took full advantage of the special pentadiagonal structure of the matrix M to solve the system of linear equations arising at every iteration.

For the problems we tested, the solution time for our serial SOR algorithm was lower than the time required by the Han–Pardalos–Ye algorithm. The solution time ratio for the two algorithms is shown in the last column. The time per iteration for the SOR algorithm grows linearly with the number of variables while the number of iterations grows sublinearly. For Problem 1, about 1/5 of the variables were at their lower or upper bound at the optimum, and for Problem 2, almost 60% of the variables were at bound at the optimum.

In Table 4 we compare the massively parallel results obtained with our Fortran 90 implementation with the results obtained by McKenna, Mesirov and Zenios for their implementation of the Moré and Toraldo [10] algorithm. Their results are reported in the column labeled MMZ.

Table 5 reports the solution time for the faster C* implementation of SOR for both types of obstacle problems using 8K and 16K processors. The number of variables ranged from 16,384 to 1,048,576. The final solution accuracy for all was at least 10^{-7}. In all instances, the speedup efficiency was always very high: always above 84% and in many cases over 90%. We also note that, using 16K processors, we were able to reduce the solution time by a factor of 30 over the serial algorithm on the IBM RISC 6000 POWERstation 550.

Tables 6 to 9 report the serial and parallel results for the elastic–plastic torsion problems for different values of c. Two starting points are considered: $x^0 = 0$ and $x^0 = u$. Tables 6 and 7 show the serial results for the two different starting points. All problems were solved within an accuracy of at least 10^{-7}.

Once again in Table 6 we compare our results with the results reported for the Han–Pardalos–Ye interior point algorithm. In all instances, the solution time for our serial SOR algorithm is substantially lower than the time required by the Han–Pardalos–Ye method. The time ratio is bigger for the "easier" problems. For the case $c = 20$, the solution time was reduced 10–fold when $n = 490,000$, $x^0 = u$, while a maximum reduction of only 2.8 was achieved for the case $c = 5$.

The results for $x^0 = 0$ are reported in Table 7. The same starting point was also used in [4]. However, no results for problems in this class with more than 160,000 variables were reported and their algorithm failed to converge for the case $n = 160,000$ and $c = 20$.

The next two tables report the results for our massively parallel implementation. Due to memory limitation we were unable to run large problems with 8K processors. Using 16K processors a 30–fold time reduction was achieved over the serial counterpart. Problems with over 4,000,000 variables were solved with solution times varying from 7 minutes ($c = 20, x^0 = u$) to less than 50 minutes ($c = 5, x^0 = u$).

Finally Tables 10 and 11 report the solution times for the journal bearing problem for the serial and parallel implementation, respectively. Two problems were considered here with $\epsilon = 0.1$ and $\epsilon = 0.5$. Results for these problems were reported in [10] for n varying from 5625 to 15625. However only the number of iterations and number of function evaluations per iteration were reported. Once again on the IBM RISC 6000 POWERstation 550, we solved problems with up to 490,000 variables and on the 16K Connection Machine CM–2, problems with over a million variables. Due to memory limitation we were unable to run problems with more than 1,048,576 variables even with 16K processors.

5 Conclusions

We have implemented both serial and massively parallel SOR algorithms to solve several large scale engineering problems. On a 16,384-processor Connection Machine CM–2, we were able to solve problems with over 4,000,000 variables in less than 50 minutes. To the best of our knowledge, these are among the largest problems in this class ever attempted.

References

[1] G. CIMATTI, *On a problem of the theory of lubrication governed by a variational inequality*, Applications of Mathematical Optimization, 3 (1977), pp. 227–242.

[2] G. CIMATTI AND O. MENCHI, *On the numerical solution of a variational inequality connected with the hydrodynamic lubrication of a complete journal bearing*, Calcolo, 15 (1978), pp. 249–258.

[3] R DE LEONE AND M.A. TORK ROTH, *Massively parallel solution of quadratic program via successive overrelaxation*, Tech. Report TR # 1041, University of Wisconsin, Computer Sciences Dept., August 1991.

[4] C. HAN, P.M. PARDALOS, AND Y. YE, *Solving some engineering problems using an interior-point algorithm*, Tech. Report CS-91-04, Department of Computer Science, The Pennsylvania State University, Pennsylvania, 1991.

[5] Z.-Q. LUO AND P. TSENG, *On the convergence of a matrix splitting algorithm for the symmetric monotone linear complementarity problem*, Tech. Report LIDS-P-1884, Laboratory for Information and Decision System, Massachusetts Institute of Technology, Cambridge, 1990. to appear *SIAM Journal on Control and Optimization.*

[6] O.L. MANGASARIAN, *Solution of symmetric linear complementarity problems by iterative methods*, Journal of Optimization Theory and Applications, 22 (1977), pp. 465–485.

[7] ——, *Sparsity-preserving sor algorithms for separable quadratic and linear programming*, Computer and Operations Research, 11 (1984), pp. 105–112.

[8] J.J. MODI, *Parallel Algorithms and Matrix Computation*, Clarendon Press, Oxford, England, 1988.

[9] J.J. MORÈ, *On the performance of algorithms for large-scale bound constrained problems*, Tech. Report MCS-P140-0290, Argonne National Laboratory, Argonne, Illinois, 1990.

[10] J.J. MORÈ AND G. TORALDO, *On the solution of large quadratic programming problems with bound constraints*, SIAM Journal on Optimization, 1 (1991), pp. 93–113.

[11] MC KENNA M.P., MESIROV J.P., AND S.A. ZENIOS, *Data parallel quadratic programming on box–constrained problems*, Tech. Report 91-04-01, Decison Sciences Department, The Warton School, Philadelphia, PA 19104, October 1991.

n	# iter	# free	time	accuracy	HPY	time ratio
10,000	306	7,588	6.30	$0.1848 * 10^{-7}$	16.3	2.59
40,000	323	31,239	26.86	$0.1643 * 10^{-7}$	131.1	4.88
90,000	361	70,896	68.10	$0.2027 * 10^{-7}$	437.6	6.43
115,600	393	91,363	95.89	$0.1900 * 10^{-6}$	700.3	7.30
160,000	449	126,904	148.54	$0.6603 * 10^{-8}$	1035.8	6.97
250,000	608	198,759	314.58	$0.5813 * 10^{-7}$	2110.5	6.71
360,000	872	286,712	750.58	$0.1171 * 10^{-7}$	4090.3	5.45
490,000	851	390,736	848.79	$0.9143 * 10^{-7}$	8977.8	10.58

TABLE 2: *Comparison of serial SOR algorithm on the IBM RISC 6000 POWERstation 550 and the HPY algorithm on the IBM 3090-600S. Obstacle Problem 1.*

n	# iter	# free	time	accuracy	HPY	time ratio
10,000	191	3,843	3.53	$0.2056 * 10^{-6}$	25.4	7.20
40,000	417	15,880	32.03	$0.1244 * 10^{-7}$	203.9	6.37
90,000	459	36,160	80.44	$0.1288 * 10^{-6}$	699.9	8.70
115,600	535	46,528	124.15	$0.7085 * 10^{-7}$	1018.7	8.21
160,000	780	64,639	241.42	$0.6745 * 10^{-8}$	1534.7	6.36
250,000	1114	101,242	560.28	$0.5935 * 10^{-7}$	3141.9	5.61
360,000	1594	146,167	1357.48	$0.5659 * 10^{-7}$	5312.4	3.91

TABLE 3: *Comparison of serial SOR algorithm on the IBM RISC 6000 POWERstation 550 and the HPY algorithm on the IBM 3090-600S. Obstacle Problem 2.*

n	# iter	time	MMZ
90,000	480	8.3	19.27
160,000	620	18.6	40.31
250,000	980	57.4	73.05
360,000	1460	87.0	110.55
490,000	2000	139.7	–
640,000	2620	264.2	–
810,000	3330	546.9	–

TABLE 4: *Comparison of parallel Fortran-90 SOR algorithm and MMZ algorithm on the Connection Machine CM-2 with 8K processors. Obstacle Problem 2.*

n	Problem 1			Problem 2		
	# iter	8K	16K	# iter	8K	16K
16,384	320	3.39	2.00	360	3.66	2.16
65,536	320	3.40	2.00	440	4.51	2.68
262,144	540	17.58	10.14	1040	32.04	18.57
1,048,576	2280	253.77	143.36	4240	472.47	249.86

TABLE 5: *Solution time in seconds for the parallel C* SOR algorithm on the Connection Machine CM-2 with 8K and 16K processors. Obstacle Problems 1 and 2.*

n	$c = 5$			$c = 10$			$c = 20$		
	# iter	time	HPY	# iter	time	HPY	# iter	time	HPY
10,000	475	9.09	12.4	469	6.54	8.8	467	5.14	5.4
40,000	672	50.68	104.9	548	29.71	71.0	539	22.74	43.0
90,000	1763	291.68	354.5	654	77.98	236.7	656	58.99	146.1
160,000	1526	501.61	911.5	724	149.87	597.5	703	109.72	381.1
250,000	2288	1034.80	1810.5	563	190.78	1128.6	300	99.46	722.5
360,000	3157	2404.78	3338.4	815	439.45	2141.1	373	190.04	1280.7
490,000	2302	1964.54	5528.3	819	508.49	3405.8	292	190.56	2052.4

TABLE 6: *Comparison of serial SOR algorithm on the IBM RISC 6000 POWERstation 550 and the HPY algorithm, $x^0 = u$. Elastic-Plastic Torsion Problem.*

n	$c = 5$			$c = 10$			$c = 20$		
	# free	# iter	time	# free	# iter	time	# free	# iter	time
10,000	7016	464	8.96	3632	469	6.54	1768	467	5.15
40,000	28112	832	63.82	18416	570	31.24	7432	561	23.53
90,000	63336	2109	362.77	35328	698	88.39	16976	651	60.71
160,000	112630	2125	668.14	59736	1040	232.04	30384	766	125.48
250,000	176040	3224	1582.09	93540	1164	436.74	47696	550	173.41
360,000	253540	4512	3189.77	134804	1676	1055.16	68888	649	357.98
490,000	345114	5981	5787.66	183640	2263	1640.34	93952	827	531.67

TABLE 7: *Solution time in seconds for the serial SOR algorithm on the IBM RISC 6000 POWERstation 550, $x^0 = 0$. Elastic-Plastic Torsion Problem.*

n	$c = 5$			$c = 10$			$c = 20$		
	# iter	8K	16K	# iter	8K	16K	# iter	8K	16K
16,384	300	2.98	1.61	300	2.86	1.58	280	2.52	1.67
65,536	400	3.78	2.13	320	2.70	1.71	280	2.52	1.48
262,144	1860	49.94	28.73	400	10.69	6.14	260	6.95	3.99
1,048,576	6320	NEM	324.75	1700	NEM	87.28	360	NEM	18.43
4,194,304	9980	NEM	2836.75	5700	NEM	1678.15	1540	NEM	432.14

TABLE 8: *Solution time in seconds with 8K and 16K processors on the Connection Machine CM-2, accuracy* $= 10^{-7}$, $x^0 = u$. *Elastic-Plastic Torsion Problem.*

n	$c = 5$			$c = 10$			$c = 20$		
	# iter	8K	16K	# iter	8K	16K	# iter	8K	16K
16,384	320	3.30	1.69	320	3.05	2.07	320	2.88	1.68
65,536	320	6.62	4.21	360	3.39	2.01	340	3.06	1.82
262,144	2700	73.57	41.43	920	24.75	14.20	480	12.83	7.37
1,048,576	9680	NEM	520.44	3740	NEM	193.51	1440	NEM	75.13

TABLE 9: *Solution time in seconds with 8K and 16K processors on the Connection Machine CM-2, accuracy* $= 10^{-7}$, $x^0 = 0$. *Elastic-Plastic Torsion Problem.*

n	$\epsilon = 0.1$			$\epsilon = 0.5$		
	# free	# iter	time	# free	# iter	time
5625	3808	255	2.87	3359	289	3.24
10000	6768	264	5.33	6040	290	5.81
15625	10564	255	8.01	9426	288	8.94
40000	27082	378	30.58	24174	290	23.27
90000	60940	908	254.55	54402	582	107.16
160000	108438	1573	436.24	96742	1066	300.13
250000	169472	2371	987.66	151156	1648	672.42
360000	244032	3292	1928.65	217684	2328	1310.18
490000	332137	4328	3445.18	296314	3102	2319.78

TABLE 10: *Solution time in seconds for the serial SOR algorithm on the IBM RISC 6000 POWERstation 550, accuracy* $= 10^{-7}$. *Journal Bearing Problem.*

n	$\epsilon = 0.1$				$\epsilon = 0.5$			
	# free	# iter	8K	16K	# free	# iter	8K	16K
16,384	11072	300	3.53	2.87	9892	320	3.95	2.18
65,536	44410	480	5.67	3.49	39598	340	4.01	2.32
262,144	177682	2000	72.53	40.91	158438	1380	49.62	28.33
1,048,576		6920	NEM	485.09		5080	NEM	354.06

TABLE 11: *Solution time in seconds with 8K and 16K processors on the Connection Machine CM-2, accuracy* $= 10^{-7}$. *Journal Bearing Problem.*