pPCx: Parallel Software for Linear Programming*

Thomas F. Coleman† Joseph Czyzyk‡ Chunguang Sun§ Michael Wagner¶
Stephen J. Wright‖

Abstract

We describe pPCx, a parallel variant of the PCx interior-point code for linear programming. We outline the major computational operation—parallel multifrontal Cholesky factorization—and present computational results on the IBM-SP multiprocessor.

1 Introduction

PCx is a linear programming solver developed at the Optimization Technology Center at Argonne National Laboratory and Northwestern University. It implements a variant of Mehrotra's predictor-corrector algorithm [5], a primal-dual interior-point approach that has proved to be highly efficient on large-scale linear programming problems. In this paper we describe pPCx, a parallel variant of PCx developed at Cornell University, and present computational results from the IBM-SP multiprocessor system.

PCx typically requires between ten and one hundred iterations to find a primal-dual solution to a feasible linear program. At each iteration, two linear systems with an identical coefficient matrix must be solved. Most of the computational effort is spent in factoring this matrix—which is large, sparse, positive definite, and sometimes ill conditioned—and in performing triangular substitutions with the factors. Efficiency of the overall algorithm depends on three main factors:

(i) identification of a good starting point,

(ii) efficiency of the sparse matrix formation, factorization, and triangular substitution procedures, and

(iii) effectiveness of the algorithmic heuristics.

Much of the speedup in pPCx is obtained by replacing the sparse Cholesky procedure [4, 2] used in PCx with the parallel sparse Cholesky factorization code of Sun [8].

*To appear: Proceedings of the Eighth SIAM Conference on Parallel Processing in Scientific Computing, Minneapolis, March, 1997. Work partially supported by the U.S. Department of Energy: the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, Contract W-31-109-Eng-38, and the Mathematical Sciences Research Program (KC-04-02) under grant DE-FG02-90ER25013.

†Computer Science Department and Center for Applied Mathematics, Cornell University, Ithaca, NY, 14853.

‡Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, U.S.A.

§Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY 14853.

¶School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY 14853.

‖Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, Illinois 60439, U.S.A.

Linear programs are input to PCx via a C data structure or MPS file. This problem is transformed by introducing shifts, slack and surplus variables, and by splitting free variables into positive and negative parts, to obtain the following simplified form:

$$\min_{x \in R^n} c^T x \qquad \text{subject to} \quad Ax = b, \;\; 0 \leq x \leq u, \tag{1}$$

where u is a vector of upper bounds. The dual problem associated with (1) is

$$\max_{\pi \in R^m, r \in R^n, s \in R^n} b^T \pi - r^T u \qquad \text{subject to} \quad A^T \pi + s - r = c, \;\; (r, s) \geq 0, \tag{2}$$

where π, s, and r represent the Lagrange multipliers for the constraints $Ax = b$, $x \geq 0$, and $x \leq u$, respectively. The Karush-Kuhn-Tucker (KKT) conditions—conditions that are necessary and sufficient for (π, x, s, r, w) to be a primal-dual solution of (1) and (2)—are as follows:

$$\begin{aligned}
A\pi + s - r &= c, && && (3)\\
Ax &= b, && && (4)\\
x + w &= u, && && (5)\\
x_i s_i &= 0, && i = 1, 2, \ldots, n, && (6)\\
w_i r_i &= 0, && i = 1, 2, \ldots, n, && (7)\\
(x, s, r, w) &\geq 0. && && (8)
\end{aligned}$$

Note that we have introduced $w \in R^n$ as a slack variable for the upper bound constraint $x \leq u$.

PCx generates a sequence of iterates

$$(\pi^k, x^k, s^k, r^k, w^k), \qquad k = 0, 1, 2, \ldots,$$

that satisfy the strict positivity condition $(x^k, s^k, r^k, w^k) > 0$. These iterates do not in general satisfy the equality conditions (3),(4),(5) except in the limit as $k \to \infty$. Compliance with the complementarity conditions (6),(7) is measured by the *duality measure* τ, defined by

$$\tau = \frac{x^T s + w^T r}{2n}. \tag{9}$$

Note that τ is the average value of all the pairwise products $x_i s_i$, $i = 1, 2, \ldots, n$, and $w_i r_i$, $i = 1, 2, \ldots, n$.

Primal-dual algorithms such as Mehrotra's algorithm apply a modified Newton's method to the equality conditions in (3)–(8). Steps are obtained by solving linear systems of the form

$$\begin{bmatrix} 0 & A & 0 & 0 & 0 \\ A^T & 0 & I & 0 & -I \\ 0 & I & 0 & I & 0 \\ 0 & S & X & 0 & 0 \\ 0 & 0 & 0 & R & W \end{bmatrix} \begin{bmatrix} \Delta\pi \\ \Delta x \\ \Delta s \\ \Delta w \\ \Delta r \end{bmatrix} = \begin{bmatrix} -r_b \\ -r_c \\ -r_u \\ -r_{xs} \\ -r_{wr} \end{bmatrix}. \tag{10}$$

Here, the matrices X, S, W, and R are positive diagonal matrices formed from the components of the positive vectors x, s, w, and r, respectively, so that the coefficient matrix in (10) is the Jacobian of the system of equations (3)–(8). The first three components of the right-hand side are the residuals of the linear constraints (3), (4), and (5), that is,

$$r_u = x + w - u, \qquad r_c = A^T \pi + s - r - c, \qquad r_b = Ax - b. \tag{11}$$

Our algorithms first calculates the affine-scaling search direction—also known as the *predictor* direction—by setting the remaining right-hand side components to the residuals of (7) and (8), that is,

$$r_{xs} = XSe, \qquad r_{rw} = RWe, \tag{12}$$

For these choices of the right-hand side, the solution $(\Delta\pi^{\text{aff}}, \Delta x^{\text{aff}}, \Delta s^{\text{aff}}, \Delta r^{\text{aff}}, \Delta w^{\text{aff}})$ of (10) is the pure Newton direction for the equality conditions in (3)–(8). Second, a *corrector* direction is calculated by re-solving the system (10) for the following right-hand side components: $r_u = 0, \quad r_c = 0, \quad r_b = 0$, and

$$r_{xs} = \Delta X^{\text{aff}} \Delta S^{\text{aff}} e - \sigma\tau e, \qquad r_{rw} = \Delta R^{\text{aff}} \Delta W^{\text{aff}} e - \sigma\tau e, \tag{13}$$

where τ is defined in (9) and ΔX^{aff}, ΔS^{aff}, ΔR^{aff}, and ΔW^{aff} are the diagonal matrices constructed from the affine scaling step components Δx^{aff}, Δs^{aff}, Δr^{aff}, and Δw^{aff}, respectively. The scalar $\sigma \in [0,1]$ in (13) is chosen by a simple heuristic; for details, consult the PCx User Guide [1].

The search direction $(\Delta\pi, \Delta x, \Delta s, \Delta r, \Delta w)$ for PCx is obtained by simply adding the predictor and corrector directions. The step taken by the algorithm is then a fraction of the maximum steps $\alpha^{\max,\text{P}}$, $\alpha^{\max,\text{D}}$ to the boundary in the primal and dual variables, respectively. We calculate

$$\alpha^{\max,\text{P}} = \inf\{\alpha \in [0,1] \mid (x,w) + \alpha(\Delta x, \Delta w) \geq 0\}, \tag{14}$$

$$\alpha^{\max,\text{D}} = \inf\{\alpha \in [0,1] \mid (s,r) + \alpha(\Delta s, \Delta r) \geq 0\}, \tag{15}$$

and set

$$\alpha^{\text{P}} = \gamma_{\text{P}} * \alpha^{\max,\text{P}}, \qquad \alpha^{\text{D}} = \gamma_{\text{D}} * \alpha^{\max,\text{D}}, \tag{16}$$

where γ_{P} and γ_{D} are two scaling factors [5, p. 588].

Since the matrix A is generally large and sparse, the coefficient matrix in the step equations (10) is sparse and highly structured. By performing simple block elimination steps, we obtain the following equivalent procedure. First, solve for π:

$$AD^2A^T\Delta\pi = -r_b + AD^2(-r_c - W^{-1}r_{wr} + X^{-1}r_{xs} + W^{-1}Rr_u), \tag{17}$$

where D is the positive diagonal matrix defined by $D = \left(S^{-1}X + W^{-1}R\right)^{1/2}$. Second, the component Δx can be recovered from

$$\Delta x = D^2(A^T\Delta\pi - (-r_c - W^{-1}r_{wr} + X^{-1}r_{xs} + W^{-1}Rr_u)). \tag{18}$$

Finally, the remaining components Δw, Δr, and Δs can be recovered as follows:

$$\Delta w = -r_u - \Delta x, \quad \Delta r = -W^{-1}(R\Delta w + r_{wr}), \quad \Delta s = -X^{-1}(S\Delta x + r_{xs}). \tag{19}$$

Most of the computational effort is tied up in solving the system (17). Provided that A has full row rank, the matrix AD^2A^T is symmetric positive definite. We assume too that it is sparse, since any dense columns in A can be omitted from the product and taken account of via the Sherman-Morrison-Woodbury formula. In pPCx, the formation of the matrix AD^2A^T, the Cholesky factorization of AD^2A^T, and the sparse triangular substitutions are performed in parallel.

For further details on Mehrotra's algorithm and primal-dual software for linear programming, see the book by Wright [9] and its associated Web site at the following

URL: http://www.mcs.anl.gov/home/wright/IPPD/. A user guide for PCx, which applies for the most part to pPCx as well, can be obtained from the following URL: http://www.mcs.anl.gov/otc/Library/PCx/. The efficiency of PCx on serial architectures is evident in benchmarks compiled by Hans Mittelmann at the following URL: ftp://plato.la.asu.edu/pub/lpbench.txt.

2 Parallel Multifrontal Cholesky Factorization

The solution of system (17) is the key computational step in an interior-point iteration. Let $M = AD^2A^T$. We solve system (17) by computing the sparse Cholesky factorization $PMP^T = LL^T$, where L is an $m \times m$ lower triangular matrix and P is an $m \times m$ permutation matrix. In this section, we describe a parallel multifrontal algorithm for performing this factorization.

Like most sparse Cholesky algorithms, our algorithm consists of a symbolic phase and a numeric phase. The symbolic phase determines a permutation matrix P, chosen to reduce fill, and the corresponding sparsity structure for the Cholesky factor L. The numeric phase computes the numerical values of L. Since the sparsity structure of M remains the same from one iteration to the next, the symbolic phase is performed only once. Details on our parallel multifrontal Cholesky factorization and parallel sparse triangular solution are given by Sun [8]. The strategy for handling small pivots during the Cholesky factorization of M is described in the PCx User Guide [1] and analyzed by Wright [10].

The *elimination tree* of L is defined to be the structure with m nodes $\{1, 2, \cdots, m\}$ such that node j is the parent of node i if and only if $j = \min\{k > i \mid L_{ki} \neq 0\}$. Node j in the elimination tree corresponds to the j-th column of L. Let $\mathcal{L}_j$ denote the sparsity structure of the j-th column of L—i.e., the set of row indices of nonzeros in the j-th column of L. Assume that the elimination tree is postordered. A *supernode* is defined to be a maximal set of contiguous columns $\{i, i+1, \cdots, j\}$ in L such that $\mathcal{L}_l = \{l\} \cup \mathcal{L}_{l+1}$ and l is the only child of $l+1$ in the elimination tree for $i \leq l < j$.

Let $S = \{i, i+1, \cdots, j\}$ be a supernode of L. The supernode S is associated with an $h \times h$ lower triangular matrix F_S called the *frontal matrix* of S, where $h = |\mathcal{L}_i|$. The first $|S| = j - i + 1$ columns of F_S are columns $i, i+1, \cdots, j$ of L. The lower triangular matrix formed by the last $h - |S|$ columns of F_S is referred to as the *update matrix* of S.

The computational task associated with the supernode S consists of the assembly of the frontal matrix F_S from the update matrices of the children of S and the partial Cholesky factorization of the assembled frontal matrix F_S. Based on the supernodal elimination tree of L, a multifrontal sparse Cholesky factorization is decomposed into a set of computational tasks. Computation starts with the leaves of the tree and progresses toward the root of the tree. Disjoints subtrees can be processed independently.

In a parallel setting, the computational tasks are mapped onto node processors by a proportional mapping scheme proposed in [6]. If a task associated with a supernode S is partitioned among a set of processors, then the frontal matrix F_S is distributed among the set of processors. In our implementation, the columns of F_S are assigned to the set of processors in a wrap-around manner. During numerical factorization, a processor μ traverses the supernodal elimination tree in postorder. If the frontal matrix F_S is entirely mapped to μ, then the processor μ initializes F_S with the numerical values of the original matrix M, assembles the update matrices of the children of S into F_S, and computes the partial Cholesky factorization of F_S. If the frontal matrix F_S is partitioned among a set of processors, then the initialization, assembly, and partial Cholesky factorization of F_S are performed in parallel.

```
k = 1;
while (k ≤ |S|) do
    if μ = map[S, k] then
        cdiv(k);
        send column k to {map[S, j] : k < j ≤ h};
        cmod(j, k) if k < j ≤ h and μ = map[S, j];
    else
        if ∃j k < j ≤ h and μ = map[S, j] then
            receive column k;
            if k + 1 ≤ |S| then
                if μ = map[S, k + 1] then
                    cmod(k + 1, k);
                    cdiv(k + 1);
                    send column k + 1 to {map[S, j] : k + 1 < j ≤ h};
                    cmod(j, k) and cmod(j, k + 1)
                        if k + 1 < j ≤ h and μ = map[S, j];
                    k = k + 1;
                else
                    cmod(j, k) if k + 1 < j ≤ h and μ = map[S, j];
                end if
            else
                cmod(j, k) if k < j ≤ h and μ = map[S, j];
            end if
        end if
    end if
    k = k + 1;
end while
```

FIG. 1. *Parallel partial Cholesky factorization of a frontal matrix F_S on processor μ.*

The key components of our parallel multifrontal Cholesky factorization are the parallel assembly of a frontal matrix and the parallel partial Cholesky factorization of an assembled frontal matrix F_S. We describe only the latter component. Let $map[S, j]$ denote the processor to which the j-th column of F_S is assigned. The columns of F_S are numbered as $1, 2, \cdots, h$. F_S is distributed among a set of processors $S = \{l, l+1, \cdots, l+q-1\}$ by columns with a wrap-around scheme–i.e., $map[S, j] = l + (j-1) \bmod q$. The partial Cholesky factorization of F_S is to compute the first $|S|$ columns of the Cholesky factor of F_S, and accumulate the effects on the remaining $h - |S|$ columns which constitute the update matrix of S. Let $cdiv(k)$ and $cmod(j, k)$ represent scaling column k and subtracting a multiple of column k from column j, respectively. The parallel algorithm for partial Cholesky factorization of F_S is shown in Fig. 1.

Since the matrix $M = AD^2A^T$ must be computed in every iteration, it is important to form M efficiently. Let $S_\mu = \{(i,j) | M_{ij} (i \geq j) \text{ is assigned to processor } \mu\}$, which is the set of nonzero entries of M allocated to processor μ. Since $M = AD^2A^T$, $M_{ij} = \sum_{k=1}^{n} A_{ik}A_{jk}d_k^2$. A processor μ computes M_{ij} if $(i,j) \in S_\mu$. The proportional mapping scheme [6] used in our parallel multifrontal code ensures that the distribution of

M among processors is roughly balanced. In the present implementation of pPCx, the constraint matrix A and the diagonal matrix D are made available on every processor, so no communication is required for forming M.

3 Computational Results

TABLE 1
Results for PDS–20, one of the larger Netlib problems

PDS–20					
size of A	32287×106180				
ordering package	WGPP				
number of processors	**1**	**2**	**4**	**8**	**16**
total running time	9489	7095	5365	3000	2016
average factorization time	143	106	79	42	26
average triangular solve time	1.63	1.34	1.32	1.06	1.55
loop time	9267	6924	5227	2912	1950
in % of total	98%	98%	97%	97%	97%
forming $M = AD^2A^T$	177	117	73	46	25
in % of loop	1.9%	1.7%	1.4%	1.6%	1.3%
pred. & corr. steps	294	259	257	225	286
in % of loop	3.2%	3.8%	4.9%	7.7%	14.7%
factorization	8723	6479	4824	2568	1565
in % of loop	94%	94%	92%	88%	80%
# of iterations	60	60	60	60	60

We report pPCx results obtained on six of the larger Netlib LP test problems. The runs were performed in batch mode on the IBM SP2 at the Cornell Theory Center on "thin" nodes consisting of processors roughly equivalent to a RS/6000 model 390 with 64 KB data cache, 64 bit memory bus and 128 MB of main memory. pPCx was run with the PCx default parameters. For further details, consult the PCx User Guide [1].

In the current implementation the matrix $M = AD^2A^T$ is formed in parallel in the sense that each processor only computes the matrix entries it will need for the Cholesky factorization. As described in the previous section, the numerical factorization is also performed in parallel. A significant portion of the code is serial. All pre- and post-processing, the line searches and other relatively minor computations are done simultaneously on all processors.

The matrix ordering has a great impact on the execution time of a sparse system solution routine, since it determines the fill in the Cholesky factor and the workload balance during the numerical factorization phase. Minimizing fill and imbalance can be conflicting goals, so it is interesting to look at the behaviour of different ordering algorithms. We ran pPCx with a well-known ordering scheme— the multiple minimum degree ordering [4]—as well as a more recent ordering, WGPP [3], which is based on graph partitioning ideas. The results differ significantly. Although there does not seem to be a clear winner, it seems that graph partitioning algorithms have a significant advantage for larger problems and larger numbers of processors.

A typical example of a run is given in Table 1. All times are given in wallclock-seconds. The total running time includes pre- and post-processing, but not the time required for reading in the data and converting it to the internal data structures. The "average

TABLE 2

Results for six larger Netlib problems

name	size of A			1	2	4	8	16
CRE–B	9648×72447	total running time	MMD	589	544	285	248	182
			WGPP	622	619	391	311	196
		avg. num. fact. time	MMD	2.25	2.55	1.68	1.10	0.81
			WGPP	3.23	4.57	2.87	1.97	1.20
		avg. triangular solve time	MMD	0.22	0.27	0.18	0.19	0.25
			WGPP	0.24	0.28	0.27	0.30	0.35
CRE–D	8926×69980	total running time	MMD	425	313	234	181	145
			WGPP	455	425	328	229	196
		avg. num. fact. time	MMD	2.06	1.76	1.50	1.12	0.83
			WGPP	2.70	3.42	2.64	1.74	1.38
		avg. triangular solve time	MMD	0.17	0.18	0.14	0.16	0.20
			WGPP	0.18	0.23	0.22	0.23	0.35
KEN–18	105127×154699	total running time	MMD	1422	888	647	489	421
			WGPP	1610	975	673	533	480
		avg. num. fact. time	MMD	14.3	7.99	4.81	3.18	2.38
			WGPP	19.63	10.38	5.94	3.85	2.64
		avg. triangular solve time	MMD	2.19	1.29	0.80	0.58	0.47
			WGPP	2.42	1.50	1.04	0.91	0.72
MAROS–R7	3136×9408	total running time	MMD	273	163	108	92.4	82.2
			WGPP	304	182	137	100	70.6
		avg. num. fact. time	MMD	6.74	4.01	2.55	2.56	2.23
			WGPP	8.64	4.88	3.53	2.68	2.05
		avg. triangular solve time	MMD	0.11	0.13	0.22	0.22	0.27
			WGPP	0.11	0.11	0.19	0.20	0.20
PDS–10	16558×49932	total running time	MMD	3279	2863	2373	1690	1393
			WGPP	1554	926	768	516	399
		avg. num. fact. time	MMD	57.6	50.1	40.4	26.1	19.0
			WGPP	27.6	15.7	12.7	7.76	4.50
		avg. triangular solve time	MMD	0.77	0.93	0.96	1.17	1.79
			WGPP	0.66	0.54	0.56	0.50	0.63
PDS–20	33874×105728	total running time	MMD	*	*	*	31590	9813
			WGPP	9489	7095	5365	3000	2016
		avg. num. fact. time	MMD	*	*	*	410	110
			WGPP	143	106	79	42	26
		avg. triangular solve time	MMD	*	*	*	14.6	2.53
			WGPP	1.63	1.34	1.32	1.06	1.55

triangular solve time" is the average time required for both a forward- and backward-substitution with the computed Cholesky factor (that is, for *two* triangular solves). The loop time is the total time for the iterative process, that is, for the repeated systems solutions, line searches and updates.

In Table 2, we see that MMD is a little more successful on the first four problems and WGPP performs significantly better on the last two. Notice also that in general the triangular solve process does not scale well and, since the predictor and corrector step computation rely heavily on this subroutine, it becomes more and more of a bottleneck as the number of processors increases. The speedup of the factorization is between 3 and

6 on 16 processors. Overall speedup factors of up to 5 are observed. Due to memory requirements PDS-20 could not be solved with the MMD ordering on $1, 2$ or 4 processors.

4 Conclusions and Future Work

There is a need for efficient parallel LP solvers capable of handling very large problems. The code pPCx, currently implemented on the IBM-SP multiprocessor system, is a step in this direction. The numerical results presented here indicate that large problem instances can be solved with good efficiency, and they motivate further development.

Several improvements and investigations are underway to further enhance performance of PCx and its range of applicability. This work includes:

- Distribution of the matrix A among processors. The current implementation requires that A be available on every processor (although the Cholesky factor of M is distributed), a memory limitation that can be severe for very large problems.
- Further investigate ordering strategies, especially the graph partitioning algorithms (for example, those of Rothberg and Hendrickson [7]).
- Investigate algorithmic variants that allow concurrent solution of triangular systems with multiple right-hand-sides.

References

[1] J. Czyzyk, S. Mehrotra, and S. J. Wright, *PCx User Guide*, Technical Report OTC 96/01, Optimization Technology Center, Argonne National Laboratory and Northwestern University, October 1996.

[2] J. R. Gilbert, E. Ng, and B. W. Peyton, *An efficient algorithm to compute row and column counts for sparse Cholesky factorization*, SIAM Journal on Matrix Analysis and Applications, 15 (1994), pp. 1075–1091.

[3] A. Gupta, *WGPP: Watson Graph Partitioning (and sparse matrix ordering) Package*, IBM Research Report RC 20453 (90427), May 1996.

[4] J. W.-H. Liu, *Modification of the minimum degree algorithm by multiple elimination*, ACM Transactions on Mathematical Software, 11 (1985), pp. 141–153.

[5] S. Mehrotra, *On the implementation of a primal-dual interior point method*, SIAM Journal on Optimization, 2 (1992), pp. 575–601.

[6] A. Pothen and C. Sun, *A mapping algorithm for parallel sparse Cholesky factorization*, SIAM Journal on Scientific Computing, 14 (1993), pp. 1253–1257.

[7] E. Rothberg and B. Hendrickson, *Sparse matrix ordering methods for interior point linear programming*, Sandia National Laboratories Technical Report 96–0475, January, 1996.

[8] C. Sun, *Efficient parallel solutions of large sparse SPD systems on distributed-memory multiprocessors*, Technical Report CTC92TR102, Advanced Computing Research Institute, Cornell Theory Center, Cornell University, Ithaca, NY, August 1992.

[9] S. J. Wright, *Primal-Dual Interior-Point Methods*, SIAM Publications, Philadelphia, Pa, 1996.

[10] S. J. Wright, *The Cholesky factorization in interior-point and barrier methods*, Preprint MCS-P600-0598, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., May 1996.