XV International Conference on Computational Plasticity. Fundamentals and Applications
COMPLAS 2019
E. Oñate, D.R.J. Owen, D. Peric, M. Chiumenti & Eduardo de Souza Neto (Eds)

HIGH PERFORMANCE INTERIOR POINT METHODS FOR THREE-DIMENSIONAL FINITE ELEMENT LIMIT ANALYSIS

NATHAN C. PODLICH*, ANDREI V. LYAMIN† AND SCOTT W. SLOAN†

*University of Newcastle
Callaghan, NSW 2308, Australia
email nathan.podlich@newcastle.edu.au

† Centre for Geotechnical Science and Engineering
University of Newcastle
Callaghan, NSW 2308, Australia

Key words: finite element limit analysis, interior point methods, Cholesky factorization, parallelization

Abstract. The ability to obtain rigorous upper and lower bounds on collapse loads of various structures makes finite element limit analysis an attractive design tool. The increasingly high cost of computing those bounds, however, has limited its application on problems in three dimensions. This work reports on a high-performance homogeneous self-dual primal-dual interior point method developed for three-dimensional finite element limit analysis. This implementation achieves convergence times over 4.5× faster than the leading commercial solver across a set of three-dimensional finite element limit analysis test problems, making investigation of three dimensional limit loads viable. A comparison between a range of iterative linear solvers and direct methods used to determine the search direction is also provided, demonstrating the superiority of direct methods for this application.

The components of the interior point solver considered include the elimination of and options for handling remaining free variables, multifrontal and supernodal Cholesky comparison for computing the search direction, differences between approximate minimum degree [1] and nested dissection [13] orderings, dealing with dense columns and fixed variables, and accelerating the linear system solver through parallelization. Each of these areas resulted in an improvement on at least one of the problems in the test set, with many achieving gains across the whole set. The serial implementation achieved runtime performance 1.7× faster than the commercial solver `Mosek` [5]. Compared with the parallel version of `Mosek`, the use of parallel BLAS routines in the supernodal solver saw a 1.9× speedup, and with a modified version of the GPU-enabled `CHOLMOD` [11] and a single NVIDIA Tesla K20c this speedup increased to 4.65×.

1 INTRODUCTION

One of the most crucial aspects in the design of ground-based structures is the stability of the supporting material, the soil. The upper and lower bound theorems of limit analysis provide a useful methodology to address the stability of the supporting body [7]. A lower bound on the true collapse load can be identified by finding a stress distribution which satisfies the equilibrium equations and stress boundary conditions, and does not violate the yield criterion at any point (a statically admissible stress field). An upper bound to the true collapse load can be determined by equating the external rate of work to the internal power dissipation through an assumed velocity field, and ensuring that the velocity boundary conditions, and the strain and velocity compatibility conditions are satisfied (a kinematically admissible velocity field). The availability of such a precise measure of the error sets limit analysis apart from many other forms of numerical analysis and makes it a very useful tool in predicting soil stability. Both the lower and upper bound problems can be formulated as a convex optimization problem [14]:

$$\begin{aligned} &\text{maximize } \alpha \\ &\text{subject to } \mathbf{B}^T\sigma = \alpha\mathbf{p} + \mathbf{p_0} \\ &\qquad\qquad f\left(\sigma^{(\mathbf{e})}\right) \le 0 \ \forall \ e = 1, 2, ..., n_e, \end{aligned} \tag{1}$$

where α is the load multiplier, $\mathbf{B}$ is the standard finite element strain-displacement matrix, $\mathbf{p}$ are the optimized loads, $\mathbf{p_0}$ are the prescribed tractions, $\sigma^{(e)}$ are the elemental stresses, f is the convex yield function, and n_e is the number of elements. If the yield function can be cast in terms of a conic constraint then the problem can be solved directly by an interior point method for conic optimization problems. The relevant conic constraints here are the second-order cone and the semidefinite cone which can be used to represent the Drucker-Prager and Mohr-Coulomb yield criteria in three dimensions, respectively. The second-order cone constraint on the n-vector $\mathbf{x}$ restricts the values to those that satisfy $x_0 \ge \sqrt{x_1^2 + x_2^2 + ... + x_3^2}$. The semidefinite constraint on the $n \times n$ symmetric matrix $\mathbf{X}$ requires $\mathbf{z}^T\mathbf{X}\mathbf{z} \ge 0 \ \forall \ \mathbf{z} \in \Re^n$. This paper is restricted to the former case using the Drucker-Prager yield criterion in three dimensions with second-order conic constraints.

The bulk of the computational effort spent by interior point methods lies in determining the search direction at each iteration by solving a set of symmetric indefinite linear equations known as the augmented equations

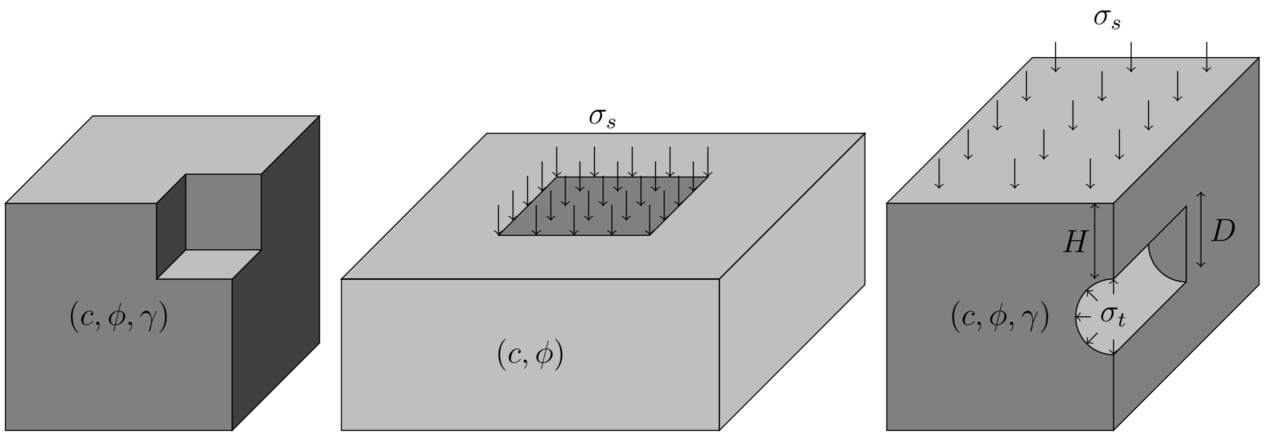
$$\begin{bmatrix} -\mathbf{D} & \mathbf{A}^T \\ \mathbf{A} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x} \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p} \\ \mathbf{q} \end{Bmatrix}, \tag{2}$$

(note that $\mathbf{A} = \mathbf{B}^T$) which is often reduced to the symmetric positive definite Schur complement equation

$$\mathbf{A}\mathbf{D}^{-1}\mathbf{A}^T\mathbf{y} = \mathbf{A}\mathbf{D}^{-1}\mathbf{p} + \mathbf{q}. \tag{3}$$

In this case, the coefficient matrix defing the search direction is a very sparse and very large positive definite matrix that is increasingly ill-conditioned as an optimum point is approached. Thus, the majority of the effort in enhancing the performance of the IPM is

Figure 1: Test problems.



(a) Square excavation (b) Square footing (c) Tunnel heading

focussed on Cholesky and Cholesky-like factorization routines. This paper steps through the major choices in developing a state-of-the-art interior point solver for finite element limit analysis using a simplified homogeneous self-dual embedded interior point method [5]. The first section addresses the choice of linear solver, the second compares the two main ordering approaches, the third section looks at improving the linear system defining the search direction, and the fourth section compares the parallel performance against the top commercial solver.

Figure 1 shows the three problems considered in the study. See Table 1 for details. The problems are a square excavation in cohesive-frictional material ($c = 1$, $\phi = 1°$ and $\gamma = 1$), a square footing on weightless cohesive material ($c = 1$), and a tunnel heading in cohesive-frictional material ($c = 1$, $\phi = 10°$, $\gamma = 0.5$, cover of $H = 4$, diameter of $D = 2$ and length $L = 6$). An angle of $\theta = 25°$ is used to determine α and k for the Drucker-Prager criterion from c and ϕ. The UB2 problems use quadratic elements in their formulation.

2 CHOICE OF LINEAR SOLVER

Because of the increasing ill-conditioning of the Schur complement as the IPM converges, the use of a Cholesky solver generally requires some way of dealing with non-positive pivots before taking the square root [18]. This can be achieved by allowing for 2×2 block pivots as in a symmetric indefinite factorization or by using basic pivot modification strategies. For the comparisons here, a left-looking supernodal Cholesky routine based on `CHOLMOD` [11] with a modified BLAS DPOTF2 subroutine that substitutes a large value (10^{32} is used) for any diagonal entry that is less than or equal to zero. `MA57` [9] was used for the multifrontal method and both options used the `MC50` implementation of the approximate minimum degree (AMD) ordering [9]. Free variables were split into the difference of two linear variables. All sequential tests were run on an Apple MacBook Pro with an Intel Core i7-3740QM 2.70GHz CPU with 16GB RAM.

Table 1: Test problem details. The columns from left to right show the number of finite elements, the number of velocity nodes, the number of stress nodes, n_F is the number of free variables, m is the number of equations in $\mathbf{B}^T$, n is the number of columns in $\mathbf{B}^T$, and $nz_{\mathbf{B}}$ is the number of non-zero entries in $\mathbf{B}$.

Problem	**Elements**	**Velocity**	**Stress**	n_F	m	n	$nz_{\mathbf{B}}$
sqexLB	265,632	265,632	155,136	271,488	1,034,703	1,202,304	6,123,526
sqexUB	337,920	196,608	337,920	944,064	1,517,567	2,971,584	9,477,630
sqexUB2	49,152	73,809	196,608	429,158	639,461	1,608,806	22,665,659
sqftLB	357,264	357,264	207,360	208,656	1,227,168	1,452,816	7,493,136
sqftUB	357,264	207,360	357,264	375,408	980,208	2,518,992	8,059,824
sqftUB2	51,840	77,257	207,360	219,750	439,757	1,463,910	22,945,686
tunhLB	561,240	561,240	326,016	335,736	1,998,455	2,291,832	15,051,585
tunhUB	561,240	326,016	561,240	600,984	1,549,007	3,968,424	15,794,780

The multifrontal method required a noticeable amount less storage for the factors than did the supernodal factorization, albeit at a significant time cost. On a smaller set of the same problems used here without the lower bound tunnel heading (which both options failed to solve to the required accuracy), the multifrontal method had an average factor non-zero count of 32 million entries and an average IPM convergence time of 188.7s compared with the supernodal methods 38 million entries and average solution time of 78.8s. The IPM using the supernodal solver was 27s-435s faster with a median improvement of 46s. This is due to the better arrangement of level 3 dense BLAS operations in the supernodal solver. The iteration counts were all the same ± 2 except for the upper bound tunnel heading where the IPM using the multifrontal solver took eight more iterations. Based on these results, the supernodal factorization appears to be superior in terms of computational speed and does not suffer from any severe robustness issues, although multiple problems did encounter numerical indefiniteness (i.e. very small or negative pivots were encountered). For the remainder of the study, the supernodal factorization was used.

2.1 A note on iterative linear solvers

Iterative linear or Krylov-subspace solvers were tested on both the Schur complement and augmented equations with a range of preconditioners [17]. The best approaches tested used a robust incomplete Cholesky factorization preconditioner with the preconditioned conjugate gradients solver. These approaches lacked robustness, failing to solve the lower bound problems in most cases (and some of the upper bound problems) in the small test set. For the problems in which the approaches were successful, the number of non-zero entries in the preconditioner ranged from 44% to 145% (averages of 60% to 75%) of the size of the factor computed in the approach developed below, with a total IPM runtime 8× to 181× slower (with an average around 70× slower). This improved slightly as the problems became larger, although no improvement in the ability to solve the lower bound problems was found [17]. The modest reduction in storage requirements did not overcome the lack of robustness coupled with the significant increase in runtime to warrant further investigation of this approach.

3 SPARSITY-PRESERVING ORDERINGS

The two resources required by any factorization are storage and time. Both of these are driven by the amount of fill-in in the factor and so good quality orderings that reduce fill-in can yield significant savings. The two main approaches in use for ordering symmetric systems are the approximate minimum degree (AMD) and nested dissection (ND) orderings. These two were compared using the supernodal Cholesky solver and split free variables. The AMD ordering was computed by `MC50` [9] while the ND ordering is computed by `METIS` [13]. The number of IPM iterations, the total IPM solution time and the number of non-zero entries in the Cholesky factor **L** are shown in Table 2.

The number of iterations was very similar, being identical for most problems and only showing differences in a small number of problems where convergence had slowed. The computation time of the ND ordering was, in general, longer than that of AMD, but this could be amortized over the duration of the IPM and was more than compensated for in the significant reduction in fill-in. The number of non-zeros in the Cholesky factor ranged from 65% more to 181% more with the AMD ordering. On average, the ND ordering saved 111% over AMD. This led to the IPM with the AMD ordering running 2.4× to 7.7×, with an average of 4.7×, that of the IPM with the ND ordering. The only problems that did not converge to tolerance was the lower bound tunnel headings. From these results it is clear that the nested dissection ordering is superior to AMD for these problems.

Table 2: Performance of solver improvements. *nit* is the number of IPM iterations, t_T is the total IPM solution time, and $nz_{\mathbf{L}}$ is the number of non-zero entries in the Cholesky factor. PREG is the solver with all improvements described.

	AMD			ND			PREG			`Mosek`		
Problem	*nit*	t_T	$nz_{\mathbf{L}}$	*nit*	t_T	$nz_{\mathbf{L}}$	*nit*	t_T	$nz_{\mathbf{L}}$	*nit*	t_T	$nz_{\mathbf{L}}$
sqexLB	24	5,058	835	24	657	297	21	507	267	20	785	237
sqexUB	23	5,350	961	23	1,332	444	18	788	339	17	1,109	293
sqexUB2	55	6,901	556	61	1,942	270	16	248	139	24	523	128
sqftLB	24	7,834	1,079	24	1,636	503	24	1,588	485	20	3,432	564
sqftUB	25	5,947	886	25	1,721	467	21	1,114	383	23	2,031	336
sqftUB2	25	1,436	317	25	599	192	19	379	168	18	556	153
tunhLB	22	12,277	1,670	18	1,618	733	25	2,367	720	22	3,482	660
tunhUB	39	12,999	1,284	35	3,250	673	17	1,350	590	19	2,364	524
Total	469	63,429		492	14,838		328	9,496		329	16,083	

4 EXPLOITING PROBLEM STRUCTURE

4.1 Elimination of free variables

While most authors mention the difficulties posed by free variables in the FELA problem formulation, few provide beneficial methods of dealing with them. Makrodimopoulos and Martin [15] describe how the free variables may be eliminated from a lower bound formulation, but only achieve around 10% improvement in performance. This is likely to be a result of `Mosek` being able to efficiently handle the free variables when not being removed. In the following, an outline is given of the efficient elimination of some (possibly all) free variables from the problem before solving the optimization problem using purely algebraic conditions. Note that a post-solve is also required if free variables are eliminated in order to return a solution suitable for interpretation of the results by the calling program and is included in performance comparisons.

The most general approach seeks to find a full rank row-set of as many free variables as possible on which it is possible to block-pivot without causing too much fill-in and eliminating as many free variables as possible. Because it may not be possible to find a suitable sparsity-preserving and stable block pivot for all free variables, the partitioning leads to

$$\mathbf{B}^T = \begin{bmatrix} \tilde{\mathbf{B}} & \mathbf{E} \\ \mathbf{N} & \bar{\mathbf{A}} \end{bmatrix} \tag{4}$$

where $\tilde{\mathbf{B}}$ is the non-singular block to be pivoted on. The constraint matrix used in the IPM is thus reduced to $\mathbf{A} = \bar{\mathbf{A}} - \mathbf{N}\tilde{\mathbf{B}}^{-1}\mathbf{E}$. This can be achieved by using a truncated LU factorization with the Markowitz criterion (row count less one multiplied by the column count less one) to rank potential pivots [16]. Note that this approach does not consider the actual fill-in, but instead uses an upper bound on the fill-in as a heuristic. An alternative interpretation is that it minimizes the number of updates to the active submatrix when the rank-one outer product update is made to the active submatrix upon eliminating the free variable. The use of doubly linked lists for both rows and columns provides a simple way to find the next pivot of lowest or low Markowitz-count subject to the stability threshold at each step and also allows simple insertion and deletion operations [8]. The process continues until a suitable pivot can no longer be found. The approach considered here uses an approximate minimum local fill-count at each step. A limited number of columns are searched in order of increasing column count [19] for a pivot satisfying the stability threshold for some scalar which produces a minimum fill score upon elimination of this pivot. The fill score is calculated as the amount of fill-in produced by using the pivot less the number of entries that would be eliminated from the active submatrix. An efficient procedure for finding such a pivot checks for stability of the potential pivot, and then computes the amount of fill-in caused by the stable potential pivot.

The elimination of free variables will often lead to a “dense” column in the constraint matrix, which will cause the Schur complement and its factor to contain many more non-zeroes. This can be dealt with as described next.

4.2 Handling dense columns

A column with a large number of non-zeros in the constraint matrix, relative to the other columns, leads to a considerably more dense Schur complement system than would be the case if the column did not contribute to the Schur complement system. For this reason, various approaches for reducing the impact of dense columns have been developed [3, 6]. The approach used here is based on Andersen's [6] modified Schur complement method but differs to allow dense columns associated with free variables to be treated explicitly, without any perturbations or modification to a conic variable. In contrast, Andersen's method will require any dense columns associated with free variables to be to be duplicated, increasing the amount of required work and memory requirements, as well as introducing the numerical difficulties known to be caused by splitting free variables.

From the augmented equations (2), partitioning $\mathbf{A} = [\mathbf{A}_D \mathbf{A}_S]$ along with $\mathbf{x}$ and $\mathbf{D}$ to match, and eliminating $\mathbf{x}_S$ gives

$$\begin{bmatrix} -\mathbf{D}_D & \mathbf{A}_D^T \\ \mathbf{A}_D & \mathbf{A}_S\mathbf{D}_S^{-1}\mathbf{A}_S \end{bmatrix} \begin{Bmatrix} \mathbf{x}_D \\ \mathbf{y} \end{Bmatrix} = \begin{Bmatrix} \mathbf{p}_D \\ \mathbf{q} + \mathbf{A}_S\mathbf{D}_S^{-1}\mathbf{p}_S \end{Bmatrix} \tag{5}$$

To determine $\mathbf{x}$ and $\mathbf{y}$ the process is as follows:

1. compute the Cholesky factorization $\mathbf{L}\mathbf{L}^T = \mathbf{A}_S\mathbf{D}_S^{-1}\mathbf{A}_S$;

2. solve $\mathbf{L}\mathbf{V} = \mathbf{A}_D$ for $\mathbf{V}$;

3. solve $\mathbf{L}\mathbf{r} = (\mathbf{q} + \mathbf{A}_S\mathbf{D}^{-1}\mathbf{p}_S)$ for $\mathbf{r}$;

4. solve $\left(\mathbf{D}_D + \mathbf{V}^T\mathbf{V}\right)\mathbf{x}_D = \mathbf{V}^T\mathbf{r} - \mathbf{q}_D$ for $\mathbf{x}_D$;

5. solve $\mathbf{L}^T\mathbf{y} = (\mathbf{r} - \mathbf{V}\mathbf{x}_D)$ for $\mathbf{y}$; and

6. solve $-\mathbf{D}_S\mathbf{x}_S = \mathbf{q}_S + \mathbf{A}_S^T\mathbf{y}$ for $\mathbf{x}_S$.

From the above it is apparent that an additional solve with the triangular factor plus the solution with the coefficient system is required. Note also that there is no guarantee that the factorization exists, but there is no more than a single dense column as a result of the elimination of free variables for these problems, with no dense columns in the original constraint matrices and no numerical difficulties were encountered in factorizing the matrix without modification.

4.3 Eliminating fixed variables subject to a second-order cone constraint

In some problems, fixed variables subject to second-order cone constraint occur. Obviously, if a fixed variable is free or linear, it may be substituted out of the problem immediately (if they are linear and fixed to be negative, the primal problem is infeasible). If, however, the fixed variable is part of a second-order cone constraint, then they may not be substituted out so easily. If the first variable associated with the kth second-order cone constraint is fixed, i.e. $a_{ij}x_j = b_i$ (that is, the ith row of $\mathbf{A}$ only has an entry in column j) and b_i is non-zero, then the Schur complement may be reduced in size.

Because the values of the unfixed variables associated with the kth second-order cone are not known, the variable may not be removed from the problem. The fixed variables constrained by a second-order cone may still be exploited, however, to reduce the computational effort required to compute the search direction [5]. For each of the fixed variables, they may be easily eliminated from every other constraint. Then symmetrically permuting (2) gives

$$\begin{bmatrix} -\mathbf{D}_{11} & -\mathbf{D}_{12} & \mathbf{0} & \mathbf{I} \\ -\mathbf{D}_{11} & -\mathbf{D}_{22} & \mathbf{A}_{12}^T & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{12} & \mathbf{0} & \mathbf{0} \\ \mathbf{I} & \mathbf{0} & \mathbf{0} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{y}_1 \\ \mathbf{y}_2 \end{Bmatrix} = \begin{Bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \mathbf{q}_1 \\ \mathbf{q}_2 \end{Bmatrix} \tag{6}$$

where $\mathbf{A}_{12}$ are the constraints with no fixed variables (possibly containing dense columns). From the fourth block equation, it is obvious that $\mathbf{x}_1 = \mathbf{q}_2$. This can be substituted into the second and third block equations, leading to a system very similar to (2) which can then be solved for $\mathbf{x}_2$ and $\mathbf{y}_1$ (there is a well-known explicit inverse for $\mathbf{D}_{22}$ that can be computed just as cheaply as for $\mathbf{D}$ [5]). Finally, $\mathbf{y}_2$ can be found from the first block equation.

4.4 Effect of exploiting problem structure

The performance of the IPM with the free variable elimination, dense column handling and fixed variable routine is shown in Table 2. It should be noted that this option uses regularization instead of splitting free variables (see [17] for details). For comparison, the top commercial solver `Mosek` is included as a benchmark.

The reduction in runtime over all of the problems is 36%. This is, in part, due to the removal of most of the free variables in many of the problems which would otherwise hamper the IPM's performance and exhibited in the reduction in iteration counts on all problems except the larger tunnel heading lower bound problem, where the problem is solved to required tolerances for the first time. The square footing and tunnel heading problems had all but one of the free variables removed, with the one remaining being associated with a dense column in the constraint matrix. The other way in which the solver has benefitted is through a smaller factorization in terms of dimension and non-zero entries. The number of non-zero entries was reduced from 2% to 49% with an average of 16%, and the most benefit occuring on both forms of the upper bound problems. The reduction in dimension resulted from both the free variable elimination and from exploiting the fixed variables. In the square footing problems, a 17% reduction in the dimension of the lower bound problem and reductions of 44% and 50% on the upper bound and quadratic formulation upper bound problems, respectively.

In addition, the solver now compares favourably with `Mosek`. The solver spends, on average, an almost identical number of iterations as the commercial offering and is noticeably quicker in runtime. The number of non-zero entries in the factorization is still considerably better in `Mosek` across the set, suggesting that further improvement in the ordering may be possible, and possibly further free variable eliminations in some of the problems.

5 PARALLELIZATION OF THE LINEAR SOLVER

In the breakdown of the IPM solution time, over 75% of the total time is still spent in the factorization routine. While other areas are able to be parallelized and are likely to yield some improvement [4], the benefit is not expected to be significant for FELA problems. The main hurdle to be overcome for sparse linear equation solvers is ensuring that the dense subproblems are large enough to fully exploit the available performance of the hardware. Fortunately, many of the supernodes, especially as one moves towards the root of the elimination tree, are large enough to expect a major improvement if a machine's parallel computing resources can be exploited. This approach also enables pre-compiled and highly optimized BLAS libraries to be used on single machines. These parallel libraries have been tuned for the hardware and provide a high fraction of peak hardware performance.

The parallel Intel MKL 11.0.5 was used for the dense BLAS operations in the supernodal Cholesky factorization. To exploit the highly parallel capability of recent GPU hardware, a modified version of `CHOLMOD` (version 3.0.3) was used [11]. `CHOLMOD` was modified by discarding the functionality to restart the factorization when a non-positive pivot is encountered and the same modified DPOTF2 routine used above is called in place of the BLAS library routine. To benchmark the improvement in performance, the problem set was solved using `Mosek` with multiple threads. Note that it is not known what specific areas in `Mosek` have been parallelized, while the two parallel options considered are only exploiting parallelism in the factorization of the Schur complement system and subsequent solves, as well as any calls to the BLAS library. The results presented in this chapter summarize simulations performed on an Intel Xeon E5-1620 @ 3.60 GHz with 64GB RAM and an NVIDIA Tesla K20c GPU. The results are shown in Figure 2 and Table 3, while Figure 3 shows the growth in factor size and total solution time with problem dimension for the upper bound tunnel heading problem.

The two options compute the same objective values, with slight differences caused by rounding in the solver with dynamically scheduled accumulation in the GPU solver when computing the contribution from the child supernodes. As expected, the iteration counts are almost identical between the two solvers and `Mosek` computes similar objective values for the problems. The parallel version on a quad-core processor is approximately twice as fast as the sequential version which is consistent with the behaviour of `Mosek` (the parallel test machine yields a 10%-15% edge over the machine used for the sequential tests). The GPU option is more than twice as fast as the parallel CPU-only version, even with the additional time necessary to initialize and finalize the GPU functionality. Both solvers are faster than `Mosek` across the set, with the GPU version achieving 4.65$\times$ speedup. As can be seen in Figure 3(b), the GPU version's performance is better as the problems become larger, where the computation is able to better saturate the hardware's capabilities.

6 CONCLUSIONS

The development of a state-of-the-art interior point method has been described, achieving significant gains over the best available solvers. The judicious choice of available linear

Figure 2: Parallel performance.

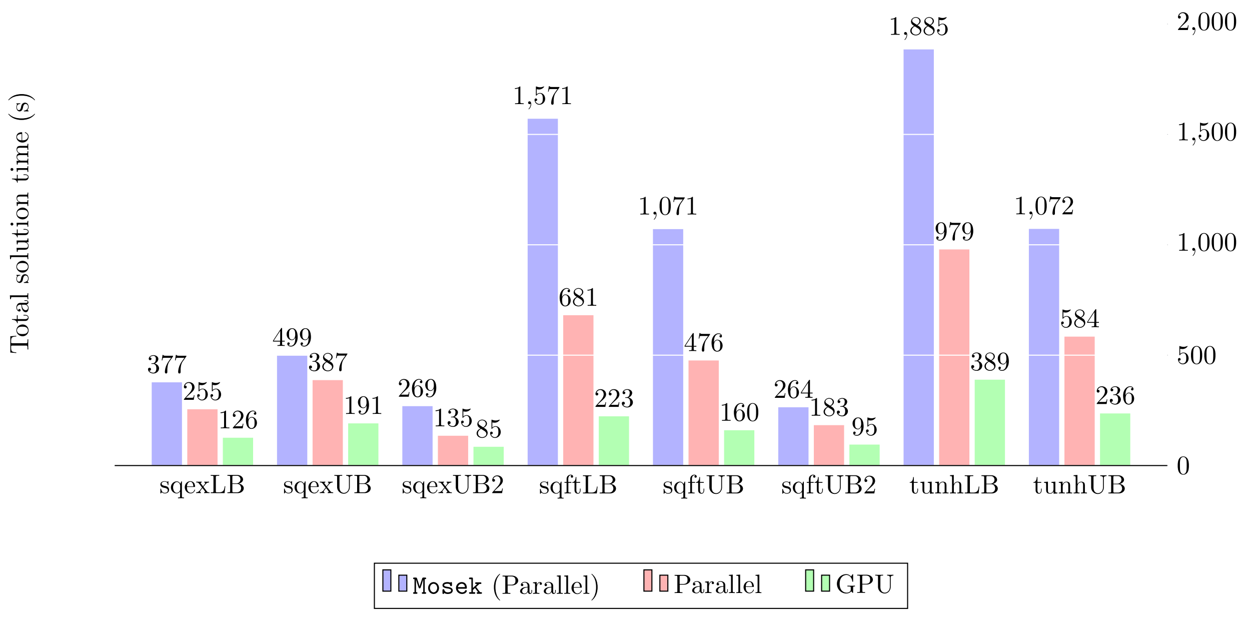
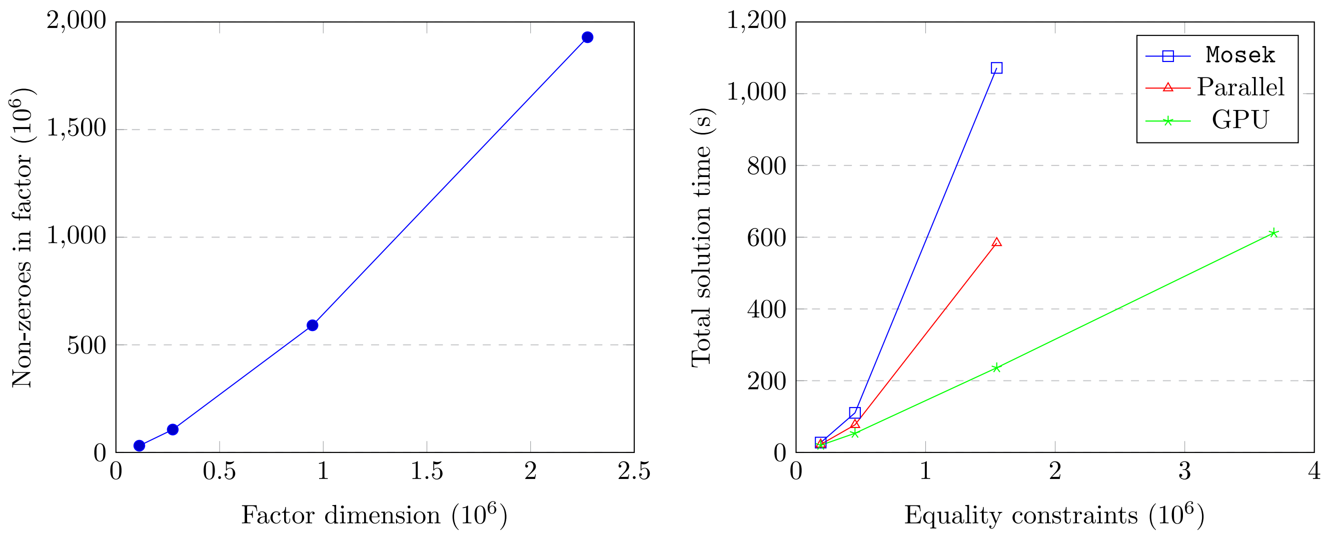


Figure 3: Upper bound tunnel heading problem with varying problem size.



(a) Number of non-zeroes in factorization (b) Total solution time

Table 3: Performance of parallel solvers. t_P is the initialization, ordering and presolve time spent before the IPM starts.

	`Mosek` (Parallel)			Parallel			GPU		
Problem	*nit*	t_T	t_P	*nit*	t_T	t_P	*nit*	t_T	t_P
sqexLB	20	377	23	21	255	4	21	126	6
sqexUB	17	499	14	18	387	2	18	191	4
sqexUB2	24	269	14	16	135	2	16	85	3
sqftLB	20	1,571	51	24	681	6	24	223	7
sqftUB	23	1,071	27	21	476	4	21	160	6
sqftUB2	18	264	12	19	183	2	19	95	3
tunhLB	22	1,885	56	24	979	9	27	389	11
tunhUB	19	1,072	25	17	584	4	19	236	6
Total	332	8,011	285	327	4,305	44	332	1,914	68

equation solver methods and attention to the nature of finite element limit analysis problems with large numbers of free and fixed variables has led to an improvement in the reliability and efficiency with which these problems can be solved. This progress enables finite element limit analysis to remain a viable and attractive design tool as larger and more sophisticated problems are considered.

To continue improving the applicability of the solver, extending the solver to handle semidefinite cones will enable large three-dimensional Mohr-Coulomb analyses to be conducted. Elastoplastic analysis could also benefit from the development of a solver specifically for the problem formulation while utilizing the developments made here. Use of greater parallelism in the linear equation solver through multiple GPUs and/or clusters should also be considered for extremely large problems.

REFERENCES

[1] Amestoy, P.R., Davis, T.A. and Duff, I.S. An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.* (1996) **17**:886–905.

[2] Andersen, E.D and Andersen, K.D. Presolving in linear programming. *Math. Program.* (1995) **71**:221–245.

[3] Andersen, E.D., Gondzio, J., Meszaros, C. and Xu, X. Implementation of interior point methods for large scale linear programming in Terlaky, T. (Ed.) *Interior Point Methods of Mathematical Programming.* Kluwer Academic Publishers, (1996) 189–252.

[4] Andersen, E.D. and Andersen, K.D. The MOSEK interior point optimizer for linear programming: an implementation of the homogeneous algorithm in Frenk, H., *et al.* (Eds.) *High Performance Optimization.* Springer, (2000) 197–232.

[5] Andersen, E.D., Roos, C. and Terlaky, T. On implementing a primal-dual interior point method for conic quadratic otimization. *Math. Program.* (2003) **95**:249–277.

[6] Andersen, K.D. A modified Schur-complement method for handling dense columns in interior point methods for linear programming. *ACM Trans. Math. Softw.* (1996) **22**:348–356.

[7] Drucker, D.C., Greenberg, H.J. and Prager, W. Extended limit design theorems for continuous media. *Q. Appl. Math.* (1952) **9**:381–389.

[8] Duff, I.S. MA28 - A set of Fortran subroutines for sparse unsymmetric linear equations. Report AERE R8730, HMSO, London.

[9] Duff, I.S. and Rutherford, C. MA57 - A new code for the solution of sparse symmetric definite and indefinite systems. *ACM Trans. Math. Softw.* (2004) **30**:118–144.

[10] Chen, W.F. *Limit Analysis and Soil Plasticity.* Elsevier, (1975).

[11] Chen, Y., Davis, T.A., Hager, W.W. and Rajamanickam, S. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.* (2008) **35**.

[12] Gondzio, J. Presolve analysis of linear programs prior to applying an interior point method. *INFORMS J. Comput.* (1997) **9**:1–110.

[13] Karypis, G. and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.* (1998) **20**:359–392.

[14] Krabbenhøft, K., Lyamin, A.V. and Sloan, S.W. Formulation and solution of some plasticity problems as conic programs. *Int. J. Solids Struct.* (2007) **44**:1533–1549.

[15] Makrodimopoulos, A. and Martin, C.M. Lower bound limit analysis of cohesive-frictional materials using second-order cone programming. *Int. J. Numer. Anal. Methods Eng.* (2006) **66**:604-634.

[16] Mészáros, C. On free variables in interior point methods. *Optim. Methods Softw.* (1998) **4**:121–139.

[17] Podlich, N.C. *The Development of Efficient Algorithms for Large-Scale Finite Element Limit Analysis.* Ph.D. dissertation. University of Newcastle, (2018)

[18] Wright, S.J. *Primal-Dual Interior Point Methods.* SIAM, (1997).

[19] Zlatev, Z. On some pivotal strategies in Gaussian elimination by sparse technique. *SIAM J. Numer. Anal.* (1980) **17**:18–30.