©1999, CIVIL-COMP Ltd., Edinburgh, Scotla
B.H.V. Topping and B. Kumar. (Editors).
Developments in Analysis and Design
using Finite Element Methods.
Civil-Comp Press. Edinburgh. 85-94.

# TRANSFERRING A NON-LINEAR FINITE ELEMENT CODE TO THE OXFORD SUPERCOMPUTER, OSCAR

**A.G. Bloodworth, C.E. Augarde and G.T. Houlsby**
**Department of Engineering Science, Oxford University, Oxford, England**

## Abstract

A complex three-dimensional finite element model has been developed to study the effects of tunnel construction on adjacent structures. The model uses an in-house finite element code in which complex simulations and non-linear material models have been developed. This paper describes the transfer of this code to a large parallel computer, and the improvements in performance that resulted from the move. The examination of the code, which was necessary for the transfer, also led to improvements in the serial version.

## 1 Introduction

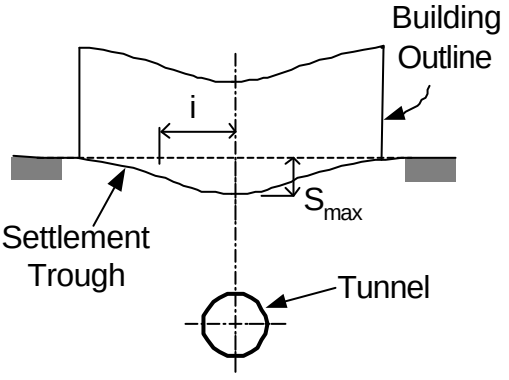
Numerical modelling of geotechnical engineering problems has been carried out in the Civil Engineering Research Group at Oxford since 1980. Finite element analysis is performed using the in-house finite element program, OXFEM, written in Fortran 90 and running on Unix workstations. A particular area of interest of the group, since 1992, has been the modelling of soft ground tunnelling and the interaction between the effects of tunnel construction and surface structures. A complex model, including simulation of tunnel construction and a surface structure has been developed (Augarde [1], Liu [2]) and is currently being improved and validated against field data (Augarde *et al.* [3], Bloodworth & Houlsby [4]).

A significant aspect of this research is the use of three-dimensional analysis. Two-dimensional simulations of tunnel-building interaction are only satisfactory for very simple geometries. A second feature is the use of non-linear material models. An overconsolidated clay deposit is modelled using a weakly non-linear, elasto-plastic formulation. Masonry, for building facades, is modelled using a strongly non-linear elastic-no-tension formulation. The combination of large numbers of degrees of freedom, and the use of an incremental solution technique, to deal with the non-linearities present, leads to large analyses, in terms of memory and run-times. The recent acquisition of a supercomputer at Oxford, with a limited number of users, has allowed more complex analyses to be undertaken.

This paper describes the transfer of the finite element code to the new parallel computer. The paper begins with a detailed background to the current research into tunnelling, to demonstrate the need for three-dimensional modelling. Some results are presented of analyses on workstations and the limitations are highlighted. This is followed by a description of the Oxford Supercomputer, OSCAR. The transfer of the existing finite element code, OXFEM, to the parallel computer is detailed and the improvements in performance are discussed. Apart from making more complex analyses feasible, the process of transfer has highlighted some improvements of the serial code, which have also been implemented.

## 2 Numerical modelling of tunnelling-induced settlement damage to structures

Tunnelling is a popular solution for the expansion of infrastructure in urban areas, since the impact of the final scheme on the environment is usually small compared to surface alternatives. The locations of many major cities are low-lying or coastal, where the near-surface geology typically involves a significant depth of unlithified deposits, leading to soft ground tunnelling conditions. In these conditions it is necessary to provide permanent ground support to the faces of a tunnel in the form of a lining.




Trough parameters: $i$ = distance from centreline to point of inflexion; $S_{max}$ = maximum settlement

Figure 1: Section through circular tunnel driven beneath a building showing assumed "greenfield" settlement trough.

The installation of a lining involves the excavation of a void of a larger diameter than the finished tunnel. An annular void is therefore created, between the lining and the excavated faces, into which the surrounding soil is able to move. In addition, soil is able to deform and flow towards the advancing tunnel face. In soft ground, the effect of these movements will be seen at the surface as settlements, which may have significant and potentially damaging effects on surface structures of historical or national importance. Much effort and expense has therefore been devoted to limiting ground movements in tunnelling schemes. Expensive accommodation works, such as compensation grouting, have been used, on schemes such as the Jubilee Line Extension in London, to prevent tunnelling settlements affecting particular structures.

The prediction of the magnitude of tunnelling settlements is relatively straightforward at "greenfield" sites, through empirical rules calibrated against past field data. The transverse settlement trough is assumed, in this case, to be an inverted Gaussian distribution [5], as indicated in Figure 1. There is, however, little guidance on the effect of a surface structure above the tunnel on a settlement trough. Consequently, little is known of the potential damage to a structure due to tunnelling-induced settlement, where this interaction occurs.

### 2.1 Damage prediction methods

Current practice usually relies on the approaches of Burland and Wroth [6] or Boscardin and Cording [7]. Both assume the settlement of a building to follow a profile predicted for a "greenfield" site (Fig. 1). Measures of damage based on the maximum tensile strain in the building are then used. The building is idealised as a linear elastic deep beam. This approach, neglecting the possible effects of the building weight and stiffness, is usually conservative and provides little indication of the localised damage seen in real structures affected by settlements.

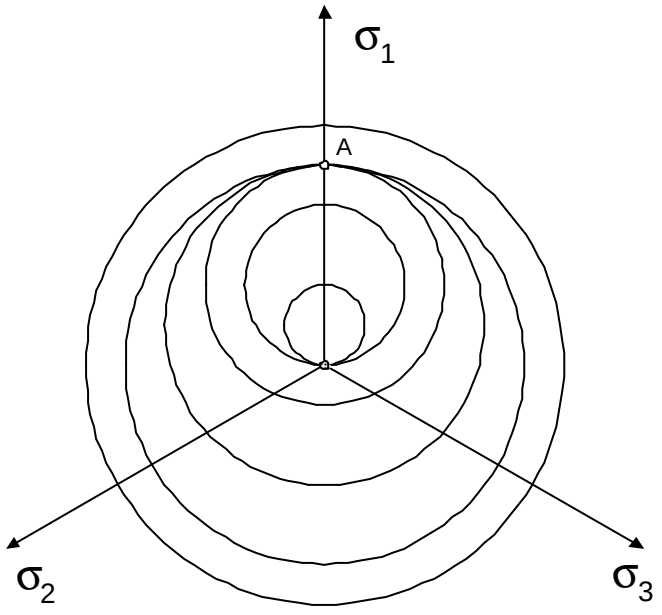



Figure 2: A plot in principal stress space of nested yield surfaces after a stress path from the origin to A and back.

Improvements to the approach outlined above have been sought, in recent years, to produce reliable predictive tools that include soil-structure interaction. Two-dimensional finite element modelling is used by Potts and Addenbrooke [8] to generate modification factors to apply to the "greenfield" settlement trough to allow for building stiffness. This approach, while giving a useful indication of some of the interaction effects, is limited in its application to problems which may be reasonably represented in two-dimensions, and where the building weight is not a significant factor. This study also provides no information on localised building damage.

### 2.2 A three-dimensional finite element model of tunnelling.

A research programme has been in progress since 1992 at Oxford to develop and use an improved numerical model of tunnelling for the prediction of settlement damage. This research has concentrated on masonry buildings located above overconsolidated clay soil [9].

The approach taken in this research is new, mainly because the problem is modelled in three dimensions rather than two. This is judged essential to represent adequately the geometry of a tunnel constructed beneath a building. Three-dimensional modelling allows the unambiguous representation of the true orientation of the building relative to the tunnel, and allows the incremental advance of the tunnel heading to be simulated.

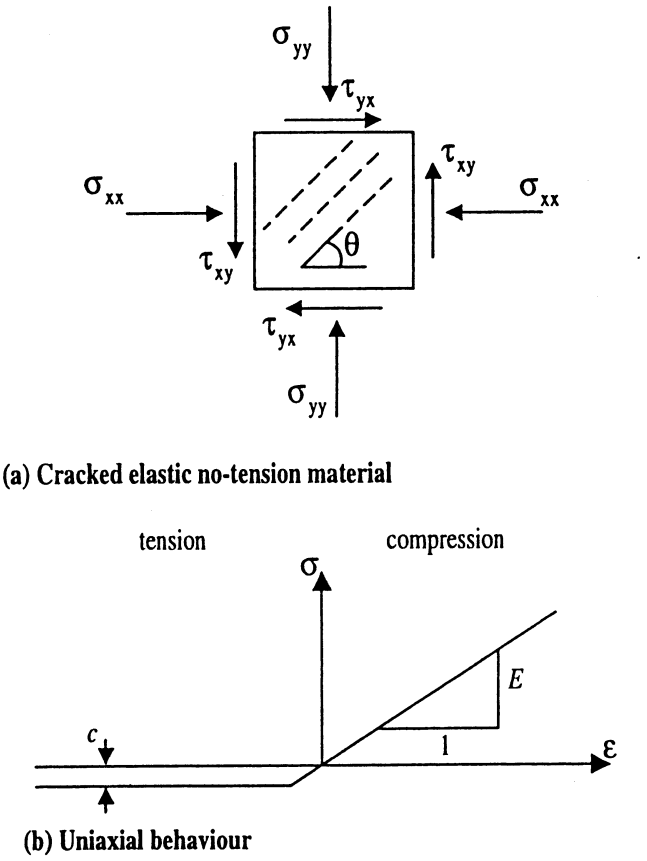



Figure 3: Elastic no-tension material model for masonry

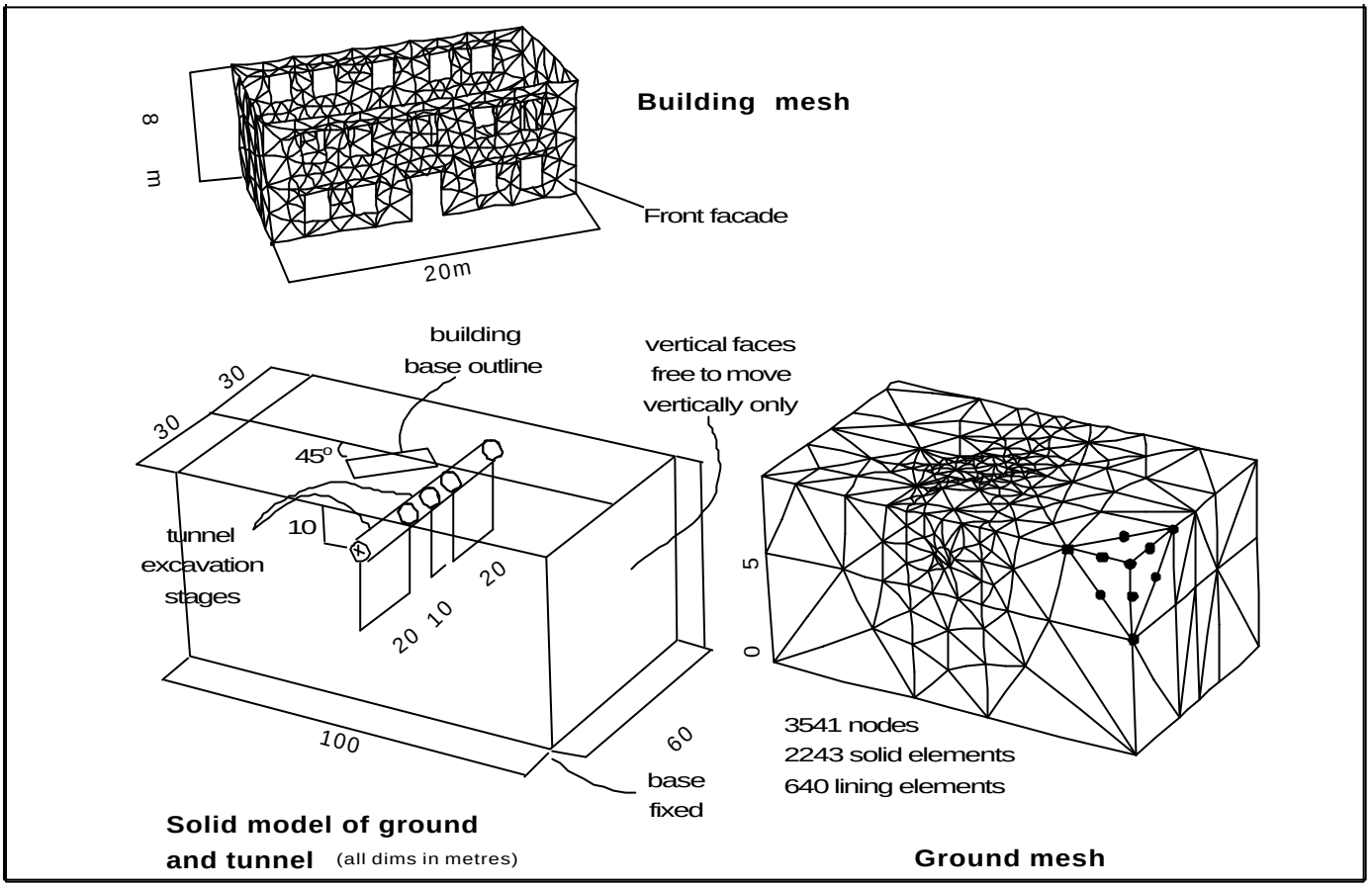



Figure 4: Demonstration model for 3-D analysis of a tunnel, ground and masonry building

The numerical model developed incorporates procedures for modelling excavation of the tunnel volume and lining installation [1]. The primary source of surface settlements, the volume loss due to the over-excavation of the real tunnel (as described above), is simulated by shrinking the lining. The lining is modelled with faceted shell elements [10, 11], where bending stiffness arises from a novel overlapping scheme, rather than through shape functions involving rotational terms, as in conventional shell elements.

The choice of material models for the ground and structure is vital for the accuracy of the final solution. The types of non-linearities that occur in the problem have been identified. Overconsolidated clay, in the undrained condition to approximate London Clay, exhibits a gradual change in stiffness at small strains. This is modelled by a nested-surface work-hardening plasticity model described in detail by Houlsby [12]. This type of constitutive model is able to capture the stress-strain history of an overconsolidated soil, now recognised to be of crucial importance in the determination of its subsequent behaviour. This is accomplished with nested von Mises surfaces as shown in Figure 2.

Building masonry on the other hand is a brittle material, experiencing an abrupt change of state in tension, moving from an uncracked elastic state to a cracked state with almost complete loss of stiffness normal to the crack direction. A material model has been implemented which models cracking and subsequent crack reclosure [2]. The stress-strain behaviour of this model is illustrated in Figure 3.

A final innovative feature of this model is a numerical scheme to "tie" a two-dimensional finite element mesh to one in three-dimensions [2, 13]. This is used to attach two-dimensional meshes, representing building facades, onto a three-dimensional mesh of the ground in which tunnel excavation is simulated. Openings in the facades may be represented either as holes in the mesh or as regions of reduced stiffness [4].

The complete model was demonstrated by Augarde [1] for a simple tunnelling scheme, involving a masonry building at a skewed angle to a tunnel. Figure 4 shows some details of this analysis and the meshes used for the ground and the building. Currently, work is underway to collect field data of buildings subjected to tunnelling settlement and to conduct verification analyses with the model [4]. A detail of the finite element mesh of one of these analyses is shown in Figure 5. In this case, a shaft is excavated close to the corner of a masonry church building. The structure is rectangular in plan with a number of openings in the facades. The predicted settlements on the surface of the combined model of ground, shaft and building are shown in Figure 6. The effect of the building in distorting the shape of the settlement bowl around the shaft is evident from this plot.

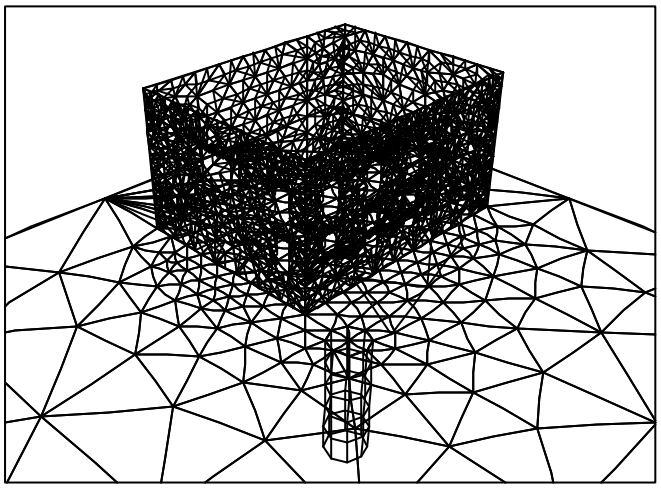


Figure 5: Detail of finite element mesh for shaft and church at Maddox Street

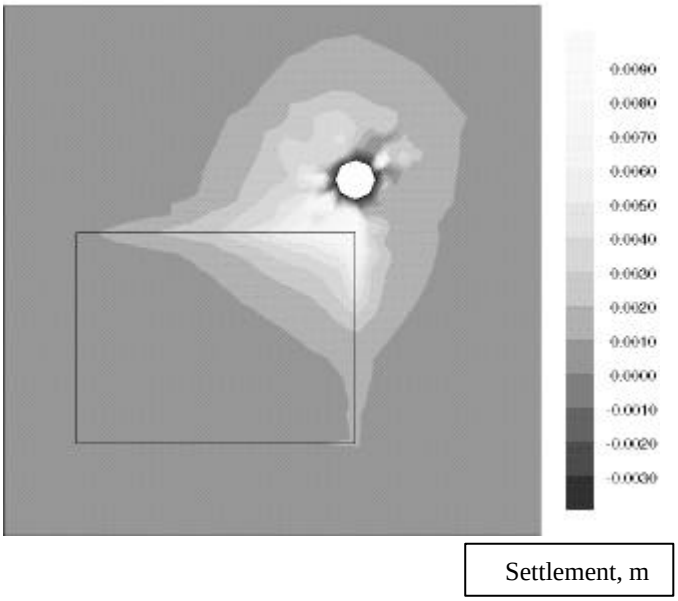



Figure 6: Surface settlements obtained from analysis of model shown in Figure 5.

# 3 Serial analysis

### 3.1 Computational procedures

The use of the model of tunnelling described above, is complex and includes considerable pre- and post-processing stages. Mesh generation is carried out using the commercial analysis package I-DEAS. Unstructured meshing is used for the soil mesh, with partitioning to provide blocks of elements for excavation simulation. Unstructured meshing enables greater control over local element lengths where more detail is required, around the tunnel and the building footprint.

A variety of Unix workstations have been used to conduct this research, prior to and since the arrival of the Oxford Supercomputer. At present, the fastest serial platform available to the Group is a Sun Microsystems Ultra 2, having two 300 MHz processors and 512Mb of RAM.

The solution procedure adopted in the finite element program OXFEM is a highly optimised implementation of the Frontal method. While this method has proved adequate for this research to date, iterative solution techniques are also under investigation. prompted by the considerable debate, at present, as to the viability of iterative solution techniques with geotechnical material models [14]. These techniques, however, appear to provide the most promising way to conduct the very large analyses planned for the future.

### 3.2 Performance

The demonstration analyses, such as that shown in Figure 4, require many degrees of freedom, even with relatively coarse meshes. The mix of non-linearities, described above, leads to a large number of incremental load steps. The combination of these two features leads to very long run-times. The verification analyses [4], for comparison to field data, use even larger and more complex models than shown in Figure 4. This is necessary to model adequately the geometry and level of detail of a practical site. For models with over 30,000 degrees of freedom, and up to 500 load steps to model a tunnel advance, run-times of one to two weeks are required on the fastest serial platform. One cause of these excessive run-times is the need to use memory well in excess of the RAM during execution.

Figure 7 shows the ground mesh for a recent verification analyses. A model with 40,000 degrees of freedom is necessary to model the construction of an underground railway tunnel beneath the Mansion House in London (construction completed in 1991). The viability of this analysis on the serial platform is dubious since run-time for a single load step, of which there are 500 in the whole analysis, is 1.5 hours.

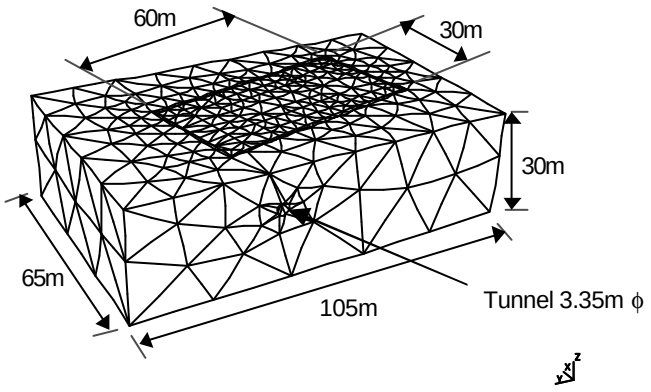



Figure 7: Verification analysis - model of tunnelling beneath Mansion House, London

# 4 The OSCAR Supercomputer

## 4.1 Background and history

The Oxford Supercomputing Centre (OSC) was established in 1998, with a mission to promote multi-disciplinary research in the application of high performance computing, particularly in parallel analysis techniques. Funding for the centre came from an approximately £1.4m grant under the UK Higher Education Funding Council (HEFCE) Joint Research Equipment Initiative, with additional support from Silicon Graphics Inc. (SGI). The main resource is the Oxford Supercomputer, OSCAR commissioned in June 1998.

Nineteen research groups at Oxford, covering a wide range of disciplines in science and engineering have access to the supercomputer. As well as civil engineering, research on OSCAR is in other branches of engineering science and in biochemistry, bioinformatics, biophysics, computing, earth sciences, materials, physics, physical and theoretical chemistry and physiology. The research groups involved share similar interests in large-scale, often mesh-based, numerical modelling.

The OSC is funded for a three-year period initially, from April 1998. In addition to maintaining and managing the system, including running a batch queue system for maximum utilisation of the resource, the OSC provides training and advice on parallel computing methods, including arranging outside speaker meetings, and facilitates contact and dissemination of knowledge between the member research groups.

OSCAR is an SGI Cray Origin 2000 supercomputer, similar in appearance to Figure 8, currently having 84 No. R10000 MIPS processors (upgradable to 128), 21GB of RAM and 256GB of disc storage. A group may make use of any number of processors at one time, with the CPU and memory usage charged to a notional account. The processors are physically arranged in pairs with RAM and access to disk space, to form modules, which in turn communicate via the Craylink interconnector (Fig. 9).

The peak processor speed is 195Mhz, 780 MIPS or 390 MFLOPS (2 integer plus 2 floating point execute and one load/store per cycle). The peak computing power is 32 GFLOPS. The architecture is CC-NUMA (cache coherent, non-uniform memory architecture). This means that the memory, although physically distributed on the nodes of the machines, behaves as one large block of shared memory. Access times to memory vary from 1 cycle to 100 cycles,

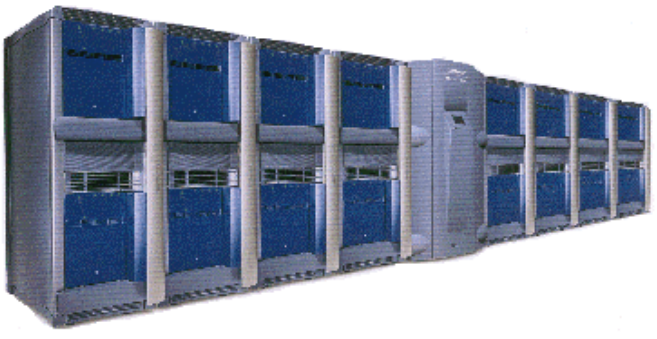


Figure 8: An SGI Origin 2000 supercomputer (from www.sgi.com)

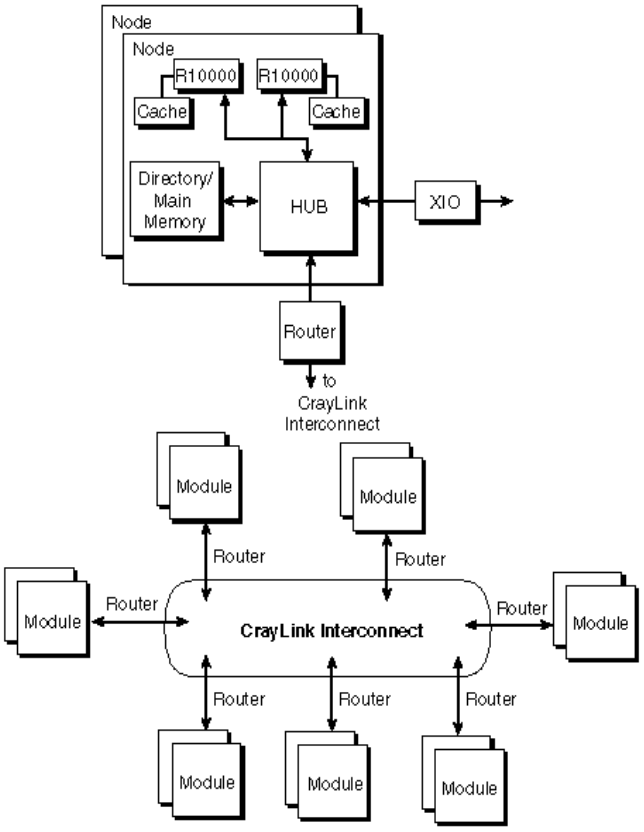



Figure 9: Physical structure of OSCAR Supercomputer (from www.sgi.com)

depending on whether the data is stored in the local cache of the processor or in the memory or cache of a processor in a remote module.

## 4.2 Parallel programming paradigms

The FORTRAN 77, FORTRAN 90, C and C++ programming languages are available on OSCAR via its SGI MIPSpro compilers. Programs may be compiled and run in serial form, but the preference is for OSCAR to be used for parallel programming, for which a number of models are available.
The first is automatic parallelisation, in which the compiler parallelises the source code without further user intervention. This may be suitable for simple programs but is less efficient for more complex code and is not often used.

Shared memory parallelisation uses directives inserted by the user in the source code, which appear as comments to a serial compiler. These directives instruct the compiler where to compile the code for parallel execution. OSCAR supports the OpenMP standard for such directives. This parallelisation method takes full advantage of the shared memory architecture; the user is not concerned about how the memory is allocated across different processors, nor how and when communication between processors occurs. It also has the advantage that the same source code may be compiled and run on a serial machine. This feature has practical advantages for program development with this project, since serial running continues side-by-side with the use of OSCAR. Extensions to the OpenMP directive set allow some user intervention over memory allocation if desired.

The most interventionist form of parallelisation is termed explicit distributed memory parallel programming. Each processor remains associated with its own local memory during program execution. The programming paradigms BSP (Bulk Synchronous Processing) and MPI (Message Passing Interface) [15], both of which are supported on OSCAR, enable the user to control the transfer of data between processors during program execution. A greater level of understanding of the system architecture is required; in particular the times taken for transfers between different parts of memory. The approach may utilise the parallel resource in the most efficient manner, but the same source code may not be compiled and run on a serial machine, so the overhead in program development is greater.

# 5 Transferring the serial FE code to OSCAR

To exploit this new computing resource, it was first necessary to move the OXFEM finite element program to OSCAR, and adapt it to run in parallel. OpenMP parallel directives for shared memory programming were used since this appeared to be a simple procedure that maintained a single code for use on serial or parallel platforms.

Parallelising an existing serial program first involves identifying the sections of the code that can be parallelised while maintaining correct functioning. The serial code is then optimised. Ideally, only then should parallel directives be inserted, and the parallel code then optimised for maximum speed-up and efficiency, which are defined as follows:

- Speed-up, $S_p = T_1 / T_p$

  where $T_1$ is the run-time on a single processor and $T_p$ the run-time on $p$ processors.

- Efficiency, $E_p = S_p / p \times 100\,\%$

## 5.1 Profiling

The first stage of the move to OSCAR involved examining the serial code. A profiling program, *speedshop* was used for this purpose. This program provides a number of utilities for performance tuning. Different 'experiments' may be run, giving information, for example, on the CPU time spent in each subroutine in the program or, alternatively, the number of cache misses and floating point exceptions.

The CPU timings for the individual subroutines in the original serial version of OXFEM, running a medium-sized analysis are given in Figure 10. This Figure is taken from *speedshop* output and gives the CPU seconds spent in the subroutine named in the final column. The preceding columns give the time in seconds and the percentage overall. Listing is in descending order. The output indicates that 98.3% of the analysis time is spent exclusively in the subroutine *frontl*. (This routine implements the Frontal solution method, as described above). The profiling therefore quickly indicated where effort should be put into optimisation and parallelisation.

By breaking a subroutine down into smaller subroutines it is possible to concentrate on the sections of code where most analysis time is spent. In the case of the *frontl* subroutine attention focussed on the few lines which carry out the elimination of the degrees of freedom from the Front matrix. (Once all elements that contribute element stiffness terms to a degree of freedom in the mesh have been assembled, the coefficients in the structure stiffness matrix for that term may be eliminated. Operations then continue on a much smaller Front matrix of active degrees of freedom). As an initial experiment, OpenMP parallel directives were placed around these lines. Figure 11 shows a code fragment including the OpenMP directives.

```
-------------------------------------------------------------------------------
Function list, in descending order by exclusive time
-------------------------------------------------------------------------------
 [index] excl.secs excl.%  cum.%  incl.secs incl.%   samples  procedure (dso: file, line)

   [2]   301.440  98.3%   98.3%   301.470  98.4%     10049  frontl (oxfem_seq: frontl.f90, 1)
   [6]     0.570   0.2%   98.5%     0.570   0.2%        19  ms_mult_d (oxfem_seq: matrix.f90, 853)
   [8]     0.510   0.2%   98.7%     0.510   0.2%        17  input (oxfem_seq: input.f90, 1)
   [4]     0.330   0.1%   98.8%     1.200   0.4%        40  force_shell (oxfem_seq: force_shell_sub.f90, 1)
  [16]     0.150   0.0%   98.9%     0.150   0.0%         5  report_alloc_realm (oxfem_seq: alloc_report.f90, 42)
  [18]     0.120   0.0%   98.9%     0.120   0.0%         4  m_mult_d (oxfem_seq: matrix.f90, 557)
  [19]     0.120   0.0%   98.9%     0.120   0.0%         4  ms_mult_iw (oxfem_seq: matrix.f90, 809)
  [11]     0.090   0.0%   99.0%     0.300   0.1%        10  initia (oxfem_seq: initia.f90, 1)
  [22]     0.090   0.0%   99.0%     0.090   0.0%         3  mincws (oxfem_seq: matrix.f90, 125)
  [20]     0.090   0.0%   99.0%     0.120   0.0%         4  mdet (oxfem_seq: matrix.f90, 42)
  [25]     0.060   0.0%   99.0%     0.060   0.0%         2  codes (oxfem_seq: codes.f90, 1)
  [26]     0.060   0.0%   99.1%     0.060   0.0%         2  g6029 (oxfem_seq: gauss.f90, 2194)
  [27]     0.060   0.0%   99.1%     0.060   0.0%         2  bshell (oxfem_seq: bshell.f90, 1)
  [28]     0.060   0.0%   99.1%     0.060   0.0%         2  mincwm (oxfem_seq: matrix.f90, 108)
```

Figure 10: CPU times obtained from *speedshop* experiment on unoptimised serial code

### 5.2 Initial parallel code results

The results obtained for speed-up and efficiency, following this relatively crude attempt to parallelise the code are shown in Figures 12(a) and 12(b) for between 2 and 8 processors. Two results sets are given in Figure 12(a): one in which $T_1$, used in the determination of the speed-up, $S_p$ is the run-time for a single processor on OSCAR and the second where $T_1$ is the run-time on the fastest serial platform. A maximum speed-up of just over four times over the serial platform was obtained, with the optimum arrangement being 5 processors. It is clear from these plots that a single processor on OSCAR is much faster than the serial platform. This is ascribed to the more advanced compiler optimisation available on OSCAR.

Figure 12(b) shows that, although a speed up was obtained on OSCAR, the actual efficiency was well below optimal, and decreased rapidly with increasing numbers of processors used. This indicated that there were probably still significant improvements to be gained. These results were, however, gratifying given the minimal effort involved.

### 5.3 A second stage of optimisation

The code fragment in Figure 11, the heart of the Frontal solver, shows that elimination across the row of the Frontal matrix is split into two loops, one each side of the pivot. It is possible to bind these two loops into one, to give improved parallel performance.

In addition, the addresses of array storage in FORTRAN 90 run column by column rather than row by row. It is, therefore, quicker to access consecutive array entries if operations proceed down columns, rather than along rows. By storing the information in a transpose of the Front matrix, and then interchanging the loops in the code shown in Figure 11, a significant speed up of around 3 was obtained on the serial machine, and around 7 on OSCAR with 8 processors.

### 5.4 Variables and scheduling

Further optimisation of the parallel sections of code is possible by considering the declaration of variables used in the loops, and the method by which the calculation load was shared between the processors, the latter being termed scheduling.

The OpenMP standard allows the declaration of variables occurring in a parallel region of code as either PRIVATE, meaning that each processor keeps its own copy of the variable, or SHARED; the latter being the default. When a particular processor accesses a shared variable, a lock is put on address in memory to prevent it being accessed or updated by another. Other processors must wait until the lock is released before using the variable, which has a time penalty. Often the compiler can determine automatically whether the variable should logically be PRIVATE (for example the loop indices I and J in the code fragment in Figure 11). In optimising the OXFEM code, it was apparent, however, that it was necessary to explicitly declare the factor MULT in Figure 10 as PRIVATE for greater efficiency.

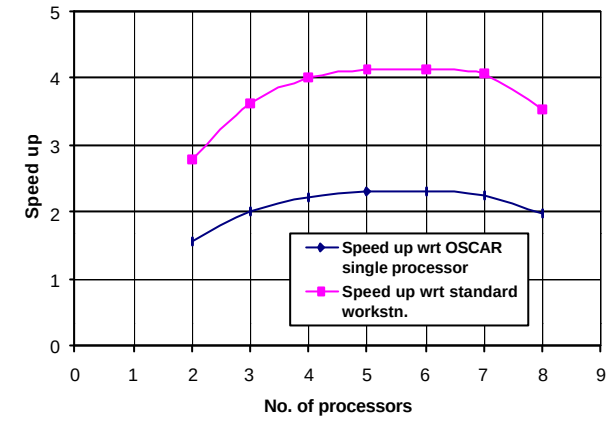
A number of types of scheduling are available in OpenMP. In static scheduling, each processor is given an equal number of iterations of the parallelised 'DO' loop to execute. In dynamic scheduling, a processor is given a smaller parcel of the total calculation load, and on completion of each, requests a new parcel. Dynamic scheduling is often more efficient when the amount of calculation load varies between loop increments, but incurs an overhead each time a processor requests new work.

The elimination carried out in the *frontl* subroutine, where attention has been focussed, involves the same number of floating-point operations per loop. The loop-counter changes between eliminations but this happens outside the parallel region. Static scheduling is therefore most appropriate and likely to yield improvements in performance in this case.

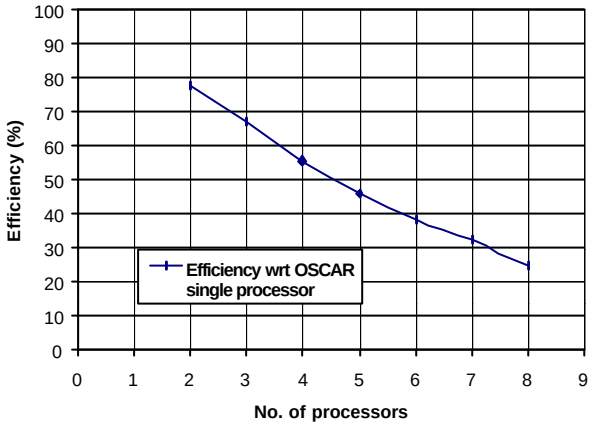
Figure 13 shows the same section of the *frontl* subroutine following implementation of the variable declarations and

```
<snip>
RPIV = 1.0_dp / FRONT(LL,LL)
!$OMP PARALLEL DO
do L = 1, LL - 1
  MULT = FRONT(L,LL)*RPIV
  FRONT(L,1:LWFRON) = FRONT(L,1:LWFRON) - FRONT(LL,1:LWFRON)*MULT
  GLOAD(L) = GLOAD(L) - GLOAD(LL)*MULT
end do
!$OMP END PARALLEL DO
!$OMP PARALLEL DO
do L = LL + 1, LWFRON
  MULT = FRONT(L,LL)*RPIV
  FRONT(L,1:LWFRON) = FRONT(L,1:LWFRON) - FRONT(LL,1:LWFRON)*MULT
  GLOAD(L) = GLOAD(L) - GLOAD(LL)*MULT
end do
!$OMP END PARALLEL DO
<snip>
```

Figure 11: First stage parallelisation of Frontal solver routine




(a)




(b)

Figure 12: Speed-up and efficiency on OSCAR after the first stage of parallelisation

scheduling described above. Figure 14 shows the change in performance of the machine relative to that with the code in Figure 11, expressed in terms of the time taken to execute one load step of the large Mansion House model (Fig. 7). The Figures show that the time taken per step has been reduced from about 20 minutes to less than 4 minutes (with 8 processors). The importance of this is that a full analysis with 500 steps becomes viable; finishing in about 33 hours, as compared to nearly 1 week before these relatively simple changes were added.

A slight reduction in wall clock time is obtained by increasing to 16 or 32 processors, as Figure 14 indicates, but the efficiency reduces. All experience with OSCAR and the analyses described here has shown this problem to be most suitable for between 8 and 16 processors. The reasons for this are thought to be the nature of the algorithms employed in OXFEM, although further work is necessary to be certain of this. Storage of the Frontal matrix in cache local to the processor, using the OpenMP directives appears to be a profitable next step to take in optimisation.

# 6 Future work

The development of the finite element code described here is a constant process. Porting the code to OSCAR is one aspect of this work. The next logical step in parallelisation of the code is the use of MPI or BSP distributed memory paradigms. Since this approach is likely to need a full re-write of the code, it is thought that the extra investment required in learning these new techniques is unlikely to be balanced by proportional gains in speed-ups and efficiency or comparable to those found in this first stage of parallelisation. This is probably due to the relative simplicity of the algorithms used here.

As indicated above, future development of OXFEM, to cope with non-linear analyses with over 50,000 degrees of freedom, is likely to include investigation of iterative solution techniques, which require less memory during execution [16, 17]. Multi-frontal methods are also available [18] for implementation and use on OSCAR. The use of OXFEM for soil/structure interaction problems other than tunnelling is also likely and this will, doubtless prompt the development of new simulation techniques within the finite element code.

```
<snip>
RPIV = 1.0_dp / FRONT_T(LL,LL)
GLOAD_LL = GLOAD(LL)
ELIM_COLUMN = FRONT_T(1:LWFRON, LL)
!$OMP PARALLEL PRIVATE(MULT, J, L)
!$OMP& SHARED(LWFRON,FRONT_T,ELIM_COLUMN,RPIV,GLOAD,GLOAD_LL,LL)
!$OMP DO SCHEDULE(STATIC)
do L = 1, LWFRON
  MULT = FRONT_T(LL,L)*RPIV
  do J = 1, LWFRON
    FRONT_T(J,L) = FRONT_T(J,L) - ELIM_COLUMN(J) * MULT
  end do
  GLOAD(L) = GLOAD(L) - GLOAD_LL*MULT
end do
!$OMP END DO
!$OMP END PARALLEL
<snip>
```

Figure 13: Second stage parallelisation of Frontal solver routine

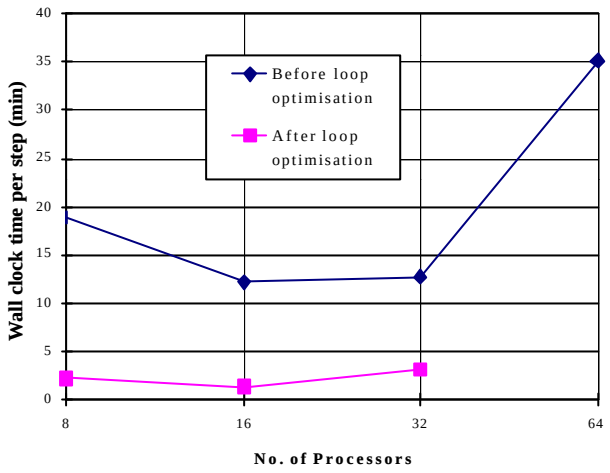



Figure 14: Execution time for Mansion House model after second stage parallelisation

# 7 Concluding Remarks

The move across to the OSCAR Supercomputer has made feasible the analysis of significantly larger 3D non-linear finite element models of tunnelling. These models are being used to develop improved predictive methods for the effects of tunnelling settlement damage on surface structures.

The process of optimising the code for parallel execution has also stimulated improvements to the serial version of the code. The move to parallel analyses was relatively straightforward because of the algorithm used and the choice of parallel paradigm. Profiling reveals that most of the computational effort, in this finite element code, is concentrated in a few lines. Once identified, various additions in this area of the code, using OpenMP directives have led to major reductions in the run-times.

# Acknowledgements

This research programme has been supported by the Engineering and Physical Sciences Research Council, the SBFSS Foundation, Howard Humphreys Consulting Engineers, the Royal Commission for the Exhibition of 1851 and Oxford University. Calculations described in this paper were performed at the Oxford Supercomputing Centre.

# References

[1] Augarde, C.E. *"Numerical modelling of tunnelling processes for assessment of damage to buildings"*, DPhil Thesis, Oxford University, (1997)

[2] Liu, G., *"Numerical modelling of damage to masonry buildings due to tunnelling"*, DPhil Thesis, Oxford University, (1997)

[3] Augarde, C.E., C. Wisser & H.J. Burd. *"Numerical modelling of tunnel installation procedures"*. Proc. 7th Int. Symp. Num. Meth. Geomech., NUMOG VII, Graz, September (in press) (1999).

[4] Bloodworth, A.G. & G.T. Houlsby. *"Three-dimensional analysis of building settlement caused by shaft construction"*. Proc. Int. Symp. Geotech. Aspects of Underground Construction in soft ground, Tokyo, July (in press) (1999).

[5] Peck, R.B., *"Deep excavations and tunnelling in soft ground"*, Proc. VII Int. Conf. Soil Mechanics and Foundation Eng., Mexico City, 226-290, (1969).

[6] Burland, J.B. and C.P. Wroth, *"Settlement of buildings and associated damage"*, Review Paper, Conference on Settlement of Structures, Cambridge, Pentech Press. (1975)

[7] Boscardin, M.D. and E.J. Cording, *"Building response to excavation-induced settlement"*, ASCE J. Geotechnical Eng., 115, 1-21 (1989).

[8] Potts, D.M. and T.I. Addenbrooke, *"A structure's influence on tunnelling-induced ground movements"*, Proc. ICE Geotechnical Eng., Vol. 125, No.2, April, 109-125, (1997)

[9] Houlsby, G.T., H.J. Burd and C.E. Augarde, *"Analysis of tunnel-induced settlement damage to surface structures"*, Proc. XII European Conference on Soil Mechanics and Geotechnical Engineering, Amsterdam, 7-10 June, in press, (1999)

[10] Phaal, R. and C.R. Calladine, "A simple class of finite elements for plate and shell problems II: an element for thin shells, with only translational degrees of freedom", Int. J. Numer. Meth. Eng., 35, 979-996, (1992)

[11] Augarde, C.E., H.J. Burd and G.T. Houlsby, *"Some experiences of modelling tunnelling in soft ground using three-dimensional finite elements"*, Proc. 4th European Conf. on Numerical Methods in Geotechnical Engineering, Udine, 14-16 October, 603-612 , (1998)

[12] Houlsby, G.T., *"A model for the variable stiffness of undrained clay"*, Proc. Int. Symp. On Pre-Failure Deformation Characteristics of Geomaterials, Torino, Sept., in press, (1999).

[13] Houlsby, G.T., G. Liu and C.E. Augarde, *"A tying scheme for imposing displacement constraints in finite element analysis"*, (in preparation).

[14] Smith, I.M., *"Parallel coupled analyses in geotechnical engineering"*, Proc. 4th European Conf. on Numerical Methods in Geotechnical Engineering, Udine, 14-16 October, 25-34, (1998)

[15] Oxford Supercomputing Centre, Training Course Notes, September , (1998).

[16] Dickinson, J.K. and P.A. Forsyth. *"Preconditioned conjugate gradient methods for three-dimensional linear elasticity"*. Int. J. Numer. Meth. Eng, **37**(13), 2211-2234, (1994)

[17] Hladik, , I., M.B. Reed and G. Swoboda *"Robust preconditioners for linear elasticity FEM analyses"*. Int. J. Numer. Meth. Eng, **40**(11), 2109-2127, (1995).

[18] Ingle, N.K. and T.J. Mountziaris, "Multifrontal algorithm for the solution of large systems of equations using network-based parallel computing". Computers and Chemical Eng., **19**(6-7), 671-681, (1995).