Technical report

Mark Filipiak[1], Francesc Levrero-Florencio[2,3], Lee Margetts[4], Pankaj Pankaj[2]

[1]Edinburgh Parallel Computing Centre, The University of Edinburgh

[2]Institute for Bioengineering, School of Engineering, The University of Edinburgh

[3]Department of Computer Science, University of Oxford

[4]Mechanical and Aeronautical Engineering Division, School of Earth and Environmental Sciences, University of Manchester

April 28, 2017

1. Abstract

The ParaFEM finite element analysis package (parafem.org.uk) has been modified to use the preconditioners and linear solvers from PETSc (www.mcs.anl.gov/petsc). ParaFEM currently possesses two different iterative solvers: conjugate gradients and BiCGSTAB(l), both with Jacobi preconditioning. The addition of PETSc solvers provides a wide range of solvers and preconditioners, including parallel direct solvers available via PETSc's interface to external packages. Performance comparisons between the corresponding ParaFEM and PETSc solvers show that they take similar times and have similar time scaling but that the PETSc solvers have poorer memory scaling, requiring three times more memory at large process counts – ParaFEM uses an unassembled stiffness matrix but PETSc uses an assembled stiffness matrix, and thus requires extra memory for temporary storage during the assembly. Other solvers and preconditioners available using PETSc were tested but none, even parallel sparse direct solvers, showed better overall time performance than conjugate gradient with Jacobi. Algebraic multigrid preconditioning was not used – PETSc uses smoothed aggregation, which for vector problems (e.g. solid mechanics) needs the full set of degrees-of-freedom to be kept at each node and ParaFEM removes restrained DOF to get a reduced set of equations.

2. Introduction

ParaFEM (parafem.org.uk) [2] is a freely available and portable library of subroutines for parallel finite element (FE) analysis, which is written in FORTRAN 90 and uses Message Passing Interface (MPI) as the parallelisation method. ParaFEM can be used to solve various types of problems, including stress analysis, heat flow, eigenvalue and forced vibrations. It has been ported to many high performance computing (HPC) systems. The ParaFEM library is accompanied by a large set of driver programs for different FE problems documented in the book *Programming the Finite Element Method* [2].

ParaFEM currently possesses two different iterative solvers, which have already proven to be highly scalable with the number of MPI processes. These solvers are the conju-

gate gradient (CG) and BiCGSTAB(l), both with Jacobi preconditioning. The CG solver solves positive-definite matrices and BiCGSTAB(l) solves general, unsymmetric, matrices. BiCGSTAB(l) is considerably slower than CG since it does many more operations to solve a linear algebraic system.

Although the existing solvers scale well, a wider range of solvers was desired to deal with any problem size and the wide range of possible types of linear algebraic systems found in FE analyses. Some of the reasons why the addition of the solvers of PETSc [1] was considered are:

- Direct solvers can be faster than iterative solvers for small systems, i.e. fewer than ~2.5M degrees of freedom (DOF).

- When using FE, the same large strain solid mechanics problem can give rise to different types of linear algebraic systems. For example, a system may be elastic at low loads, and positive-definite (CG can be used), but when reaching buckling or a limit point, it becomes positive-semidefinite and it may become indefinite in the post-buckling regime (MINRES can be used, for instance). If damage or non-associative plasticity is considered, the resulting linear algebraic system will become unsymmetric after yield (BiCGSTAB(l) or GMRES can be used). In these cases, there needs to be a choice of solvers available.

- There are many other preconditioners that can be used instead of Jacobi, for example incomplete LU factorisation (ILU) and algebraic multigrid (AMG). These will reduce the number of iterations of the solver but each iteration will be slower.

ParaFEM also possesses ARNOLDI and LANCZOS algorithms to solve eigenvalue problems, but in this eCSE project we have not extended the range of eigenvalue problem solvers (which would require a package such as SLEPc).

With a wide range of available solvers and preconditioners, a general interface is needed to choose each. There are several open source linear solver toolkits available that can provide the wider range of solvers desired in ParaFEM (e.g. PETSc, Trilinos, hypre...). PETSc (www.mcs.anl.gov/petsc) [1] was chosen because it has a comprehensive set of scalable parallel solvers and preconditioners (including parallel direct

solvers via the external packages MUMPs and SuperLU), and up-to-date Fortran interface, and is actively maintained and developed. PETSc also provides nonlinear solvers (of the Newton type) and also load/time-stepping procedures; and these could also be added to the ParaFEM-PETSc interface.
The problems that were used to test the efficiency of PETSc's solvers are:

- A linear elastic solid system in a small strain regime. This is an example of a positive-definite case.
- Steady-state cavity driven flow of an incompressible Navier-Stokes fluid case. This is an example of an unsymmetric case.
- Large strain solid mechanics modelling of trabecular bone. This is an example of a case which starts as positive-definite and becomes indefinite.

3. The ParaFEM-PETSc interface

The PETSc library can be used directly in a ParaFEM driver program but the structure of most ParaFEM programs makes it possible to have a simple interface to the PETSc linear solvers and preconditioners. The design of the ParaFEM-PETSc interface depends on the characteristics of the two libraries, which are:

3.1. ParaFEM

- Two linear solvers are already available (CG and BiCGStab) and these were already converted into subroutines (they are in-line code in the examples in [2]), and solver choice is by conditional statements in the driver program.
- The solver sequence is always one of the following:
  - Linear problems:

```
Create stiffness matrix K and vectors x, f
Set the entries in K
Solve Ku = f
```

- Nonlinear problems:

```
Create stiffness matrix K and vectors x, f
DO Newton-Raphson iterations
  Set the entries in K
  Solve Ku = f
END DO
```

- Load/time-stepping problems:

```
Create stiffness matrix K and vectors x, f
DO load/time step
  Do Newton-Raphson iteration
    Set the entries in K
    Solve Ku = f
  END DO
END DO
```

In all cases, there is only one matrix, one solution vector and one load vector. The matrix structure is fixed (i.e. the mesh connectivity does not change throughout the problem) but the matrix entries may change in nonlinear problems.

- The matrix is stored in unassembled form – each process has an array holding all the element matrices for the elements on that process. The solution and load vectors are held in assembled form – each process has its part of the global array holding the vector. The load and solution vectors are converted between the global and per-element form (used for the matrix-vector products) with the ParaFEM subroutines `gather` and `scatter`. These subroutines use global data structures set up by ParaFEM from the mesh connectivity data read at the beginning of the corresponding driver program.

- The driver programs use a control file to set problem data, solver tolerances, etc...

- Equations corresponding to restrained degrees of freedom (i.e. homogeneous Dirichlet BCs) are removed from the system.

3.2. PETSc

- Matrices and vectors are stored as objects (C structures), `Mat` and `Vec`.
- Matrices and vectors are stored in assembled form – this means that there is an extra step when assembling a matrix. It is still possible to use unassembled matrices in PETSc but this would require creating a new matrix type and providing all the operations needed; for example the matrix-vector product for the iterative solvers. However, all the preconditioners provided by the PETSc expect an assembled matrix.
- The memory needed to store the matrix needs to be allocated before the actual assembly, otherwise the assembly becomes around 50 times slower.
- PETSc can use options set in a control file to set the solver and preconditioner used, solvers tolerances, etc...The options are described in the PETSc documentation [1].
- Tasks such as setting up matrices, vectors and solvers require calling several PETSc subroutines in sequence in each case.

3.3. Interface design

- The structure of the driver programs is kept as close as possible to that in the examples in [2].
- The driver programs can be written to use the solvers in ParaFEM, or the solvers in PETSc, or can choose between them at run time to do comparisons (subroutine `get_solvers`).
- The ParaFEM matrix and vectors are converted to PETSc `Mat` and `Vec` structures when using the PETSc solvers.
- The solve sequences in ParaFEM allow the PETSc subroutine calls to be wrapped up in one interface call for each task. As an example, the solvers sequence for

load/time-stepping problems when using PETSc is:

```
Initialise PETSc
Create PETSc matrix K and vectors x, f
DO load/time step
  Do Newton-Raphson iteration
    Clear the global matrix
    DO element
      Calculate element stiffness matrix k_m
      Add k_m to global matrix K
    END DO
    Complete the setup of K
    Solve Ku = f
  END DO
END DO
```

Table 1 lists the subroutines that provide the ParaFEM-PETSc interface.

- The standardised data structures in ParaFEM allow the PETSc data structures to be held in one global data structure. This makes the interface even simpler – just subroutine calls.
- Control of PETSc is by standard PETSc control file, not by specifying options in the ParaFEM control file and translating them to PETSc – this would increase the maintenance as PETSc is continually changing (e.g. addition and deletion of solvers and preconditioners, changes in the options, even the API...). Several solvers and preconditioners can be listed in the control file and chosen during the program execution using the `p_use_solver` subroutine.
- Direct calls to PETSc are still possible.
- PETSc nonlinear and load/time-stepping subroutines are not yet used.

Table 1: PETSc interface subroutines

Name	Description
`get_solvers`	Get the name of the solvers to use from the command line
`p_initialize`	Initialise PETSc
`p_create`	Create the PETSc matrix and vectors
`p_zero_matrix`	Zero the PETSc global matrix
`p_add_element`	Add an element matrix to the PETSc global matrix
`p_assemble`	Assemble the PETSc global matrix
`p_zero_rows`	Modify the PETSc global matrix and the load vector by the fixed freedoms. The resulting matrix is not symmetric.
`p_use_solver`	Choose one of the PETSc solvers specified in the `.petsc` configuration file
`p_solve`	Solve using PETSc
`p_shutdown`	Destroy the PETSc data structures and finalise PETSc

4. Matrix pre-allocation

After a PETSc `Mat` structure is created, the memory needed to store the matrix needs to be allocated. If sufficient memory is not allocated then, as PETSc adds entries to a matrix, it will reallocate memory as required – this can increase time to add all the entries by a factor of around 50. The amount of memory can be overestimated but any excess memory is not freed after the assembly step – this means that more memory per process is required and thus more computing nodes are required than are necessary for the actual calculation. The amount of memory needed is approximated by estimating the number of non-zero entries in each row of the matrix (and these are split into on/off process entries). Three methods for estimating the number of non-zeroes were tested:

1. For each of the element types used in ParaFEM (Appendix B of [2]), the node connectivity of a regular (almost regular for tetrahedral) mesh was calculated. For example, for linear hexahedra, each node is connected to 26 others; for linear tetra-

hedra, each node is connected to approximately 14 others on average. For higher order elements the connectivity of vertex nodes was used – this overestimates the connectivity. To allow for mesh irregularity, an extra factor of 1.3 was applied – this could be adjusted by the user. With this method, the fraction of non-zeroes that are on-off-process cannot be estimated, so this effectively applied a further factor of 2. This method is fast but overestimates the memory required by a factor of 3 in total for the xx18 test case.

2. ParaFEM calculates the element-to-equation mapping (or pairing) in order to gather and scatter vector data during the matrix-vector multiplication by the unassembled matrix. The mapping was inverted to get the equation-to-element mapping. The composition of these two mappings gives an equation-to-equation mapping from which the non-zero entries in the assembled matrix can be extracted. The partitioning of elements and equations is also defined by ParaFEM, and thus the number of on-process and off-process non-zeroes in each row can be found. This gives an exact estimate of the memory to be pre-allocated. The element-to-equation mapping is distributed and, before it is inverted, all the elements that contain equations on a particular process need to have their part of the mapping sent to that process (this is effectively setting up element haloes). The inversion of the mapping is done by sorting an array of element-equation pairs by equation. The composition of the mappings is combined with the counting of non-zero entries. The slow steps are the sorting to invert the mapping, and the composition and counting.

3. PETSc 3.7 (installed after method 2 was developed) has a `MATPREALLOCATOR` matrix type which can be used to calculate the pre-allocation for the actual matrix by doing a matrix assemble but not actually setting aside space for the values. The stash of off-processor matrix elements (see below in the discussion of the peak memory use in PETSc) still needs to be created and distributed, so use `MATPREALLOCATOR` is slower than the hand calculation. The pre-allocator matrix also uses up more peak memory than the actual matrix (about 30% more at large process counts).

Method 2 is currently used because it is faster and uses less peak memory but method 3 would have been used instead if it had been available earlier.

5. Tests and performance

The PETSc solvers can be used in all the ParaFEM driver programs (except for eigenvalue problems). Three types of problems have been tested:

- Linear elasticity (xx18)
- Steady-state incompressible fluid flow (xx17)
- Nonlinear elasto-plasticity (xx15)

All the tests were done on the ARCHER Test and Development System (TDS). This has the same hardware and software as ARCHER but consists of one cabinet with 43 nodes (1,032 cores) and is lightly loaded. The communications bandwidth on ARCHER would be about half that on the TDS (ARCHER scheduling does not put all requested nodes in one cabinet, and ARCHER is heavily loaded) so the performance results should be treated as an upper bound for the performance on ARCHER.

5.1. Program xx18

This is Program 12.1 of [2] with the addition of the PETSc interface and the ParaFEM subroutine `pcg_ver1`. This program is a three-dimensional analysis of a linear elastic solid, in this case a cuboid with a uniformly loaded patch at the centre of one face, made up of 20-node hexahedral finite elements. A range of problem sizes can be generated, from a small 5^3 elements grid for regression tests up to a 440^3 elements grid for scaling tests. The results shown in Fig 1 are for a 100^3 elements grid (~12M DOF). The solver was CG with Jacobi preconditioning for ParaFEM and for PETSc.

Note that the PETSc error tolerance is for the preconditioned residual, while the ParaFEM error tolerance is for the un-preconditioned ("true") residual – the same tolerance was used for both solvers after the appropriate conversion. The converged displace-

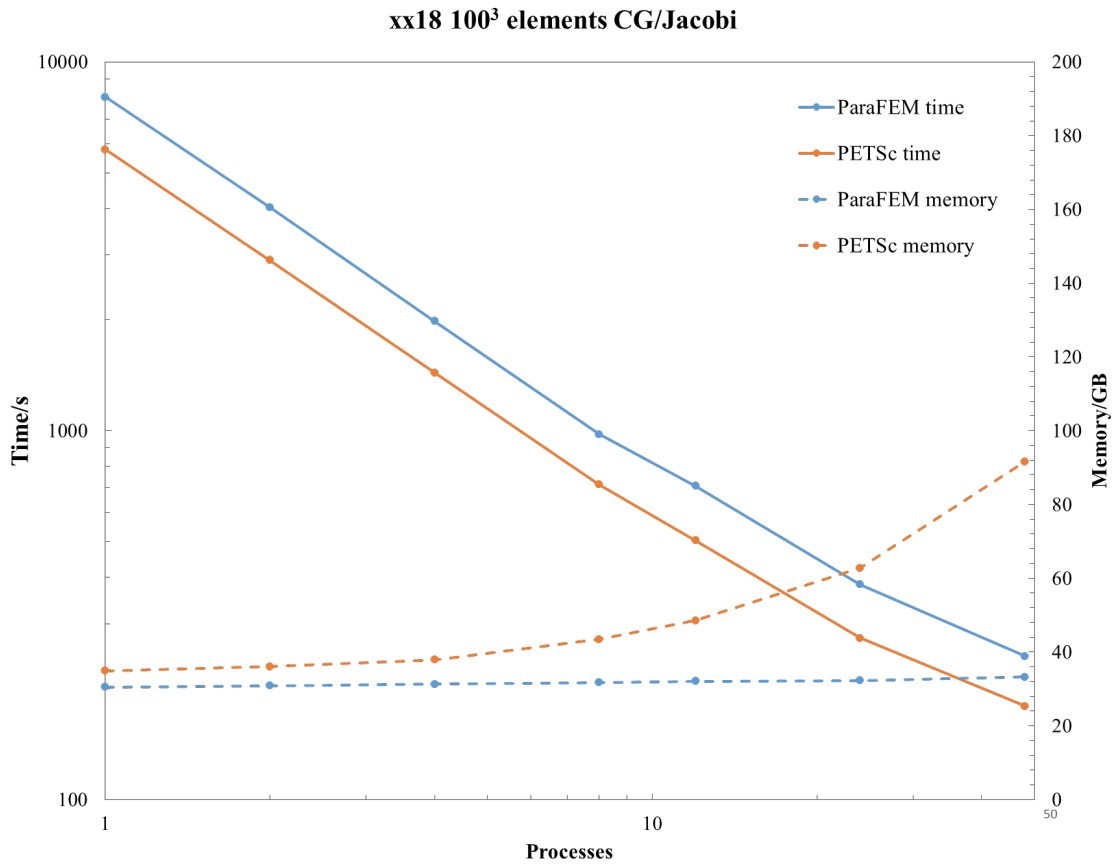

Figure 1: Time and memory scaling of CG/Jacobi for xx18 with 100^3 elements. Time is the sum of the matrix assembly, preconditioner setup and solver times. Memory is the peak memory use. The used system is a Cray XC30, 8 Intel E5-2650 v2 processors (4 nodes) with 12 cores per processor. The processes were distributed equally across 8 nodes, so 1 processes uses 1 node, 2 processes use 2 nodes, . . . , 16 processes use 8 nodes with 2 processes per node, etc.

ment fields from the PETSc and ParaFEM linear solvers matched within the specified tolerance.

The PETSc version of the CG/Jacobi is ~30% faster than the ParaFEM version for this case but both have the same time scaling. Note that the slowdown seen above eight processes is an artefact of the distribution of the processes across nodes: as the number of processes per node increases, the memory bandwidth per process is reduced (and FE codes are generally memory bound); and Turbo Boost is enabled on ARCHER processes,

so that the clock speed will increase when fewer cores on a processor are active.

PETSc requires much more memory as the number of processes increases. The peak memory occurs during the assembly of the stiffness matrix – off-process entries are stored temporarily as elements are added with `p_add_element` and then distributed with `p_assemble`. The relative amount of temporary storage depends on how the mesh is partitioned among the processes but will always increase with increasing numbers of processes. On distributed-memory computers (e.g. clusters), the total memory available will usually increase faster than the peak memory requirement.

Some tests were made of the performance of other solvers and preconditioners available in PETSc with a small test case of 20^3 elements (~100K equations) – all of the following were slower than CG/Jacobi:

- CG/block-Jacobi
- MINRES/Jacobi
- GMRES/ILU – the only parallel ILU (incomplete ILU) preconditioner available in PETSc is hypre's PILUT, which does not preserve symmetry so GMRES was used as the solver.
- MUMPS and SuperLU_DIST parallel sparse direct solvers – these are also slower than CG/Jacobi, even when the latter uses a very small convergence tolerance

The algebraic multigrid preconditioner in PETSc uses smoothed aggregation. This can be used for scalar problems (e.g. heat conduction) in ParaFEM but not vector problems (e.g. solid mechanics) because the full set of degrees-of-freedom needs to be kept at each node and ParaFEM removes restrained DOF to get a reduced set of equations.

5.2. Program xx17

This is Program 12.6 of [2] with the addition of the PETSc interface and the ParaFEM subroutine `bicgstabl_p`. This program models three-dimensional steady-state flow of an incompressible fluid in a driven cavity, discretised with Taylor-Hood elements (20-node hexahedra for velocity, 8-node hexahedra for pressure). A range of problem sizes

can be generated, from a small 10^3 elements grid for regression testing to a 100^3 elements grid for scaling tests. This problem is nonlinear and is solved using a Newton-Raphson scheme. The matrix for the linear solver in each Newton-Raphson iteration is unsymmetric and the solver used was BiCGStab(l = 4) with no preconditioning (Jacobi preconditioning gives poorer, or none, convergence for this unsymmetric, saddle-point problem).

For this case there are non-zero restraints (the velocity of the driven boundary). These are handled in the same way by ParaFEM and PETSc – the corresponding rows of the matrix are zeroed and the effect of the restraints is added to the RHS vector.

The small, medium and large cases were tested. In the small case the results are the same for ParaFEM and PETSc within the solver tolerance, but in the medium and large cases there are differences larger than the solver tolerance. Tests with ordered reduce operations (using the Cray MPI environment variables `MPICH_ALLREDUCE_NO_SMP` and `MPICH_REDUCE_NO_SMP`) and with an ordered version of the ParaFEM scatter operation showed similar differences between ParaFEM and PETSc. The BiCGStab(l) algorithm in PETSc is a modified version, which may explain the differences – when using PETSc the solver diverges at a particular Newton-Raphson iteration. This causes more Newton-Raphson iterations and PETSc is two times slower than ParaFEM overall (note that the number of iterations reported by PETSc count the l iterations at each step, so the value is l times the corresponding value reported by ParaFEM). Further work is needed to resolve the numerical differences and the cause of the solver divergence.

5.3. Program xx15

This is an experimental program, still under development, that will become a new program example in [2]. xx15 is a large strain elasto-plasticity driver program which is currently being used for multiscale modelling of trabecular bone. xx15 uses a plasticity algorithm and implements a UMAT-like subroutine, which means that most (with a few restrictions) user-defined elasto-plastic constitutive law can be implemented. Three constitutive laws are currently available: linear elasticity, linear elasticity with an eccentric-ellipsoid yield surface with linear hardening, and linear elasticity with a von Mises yield

surface with nonlinear hardening.

Elasto-plastic simulations in solid mechanics are inherently nonlinear and solved in (pseudo-)time; thus, several load (or displacement) increments must be performed to track the solution. Since implicit time integration is used, an iterative solution (currently Newton-Raphson) for nonlinear equations has to be used within each load increment in order to obtain a converged solution. Consequently the number of times an algebraic system needs to be solved equals the number of load increments multiplied by the corresponding number of Newton-Raphson iterations required for each increment. The problems considered comprised of a fixed number of load increments (typically 50 or 100) and each load increment typically requires from two to five Newton-Raphson iterations, depending on the nonlinearity of the solution at that stage.

xx15 is used in solving mechanical systems involving buckling and elasto-plastic problems which may present softening. For example, bone is known to suffer from microcracking (which can produce softening) and buckling/structural failure (or in mathematical terms at least one eigenvalue of the stiffness matrix becomes zero). Systems with a softening response will have both positive and negative eigenvalues. The resulting linear algebraic systems are indefinite systems, which become unsymmetric when a damage model or non-associative plasticity is used. The addition of the PETSc gives a range of solvers so that the fastest solvers can be used at each stage: CG can be used during the linear stage (matrix is positive-definite), SYMMLQ or MINRES during buckling (indefinite), and eventually BiCGStab or GMRES if the matrix becomes unsymmetric.

In xx15 fixed displacements (rather than loads) are incremented and non-zero restraints are used to represent these displacements. Unlike xx17, these non-zero restraints are handled separately, which means that a small ancillary array is needed when using PETSc (otherwise a full ParaFEM-type stiffness matrix would need to be kept as well as the PETSc stiffness matrix).

Three small cases were used for validation against the ParaFEM solvers and one large case (simulation of the mechanical behaviour of trabecular bone) was used for scaling tests. The small cases were:

- One element in compression and tension – eccentric-ellipsoid yield surface

- Several elements constant strain case – eccentric-ellipsoid yield surface
- Necking of a cylinder – von Mises yield surface

The results using ParaFEM and PETSc versions of CG/Jacobi matched to within the solver tolerance.

The orthopaedic engineering research group at The University of Edinburgh uses ParaFEM to simulate the mechanical behaviour of trabecular bone. The detailed geometry generated through X-ray microtomography is used to assess the response of bone to mechanical loads at the microstructural level. A set of images describing the micro-architecture of a small cylindrical sample of 20 mm of height and 10 mm of diameter can contain ~100M degrees of freedom. The test case used here is a smaller sample, containing ~7.5M elements (8-node hexahedra) and ~26M DOF. The load was kept in the range where the stiffness matrix remained positive-definite, so CG/Jacobi could be used. Figure 2 shows the time and memory scaling, which is similar to that seen for the xx18 program. In this case the PETSc solver is around 10% slower than the ParaFEM solver at large process counts but both scale up to 1,000 processes. The peak memory needed by PETSc is again about three times that needed by ParaFEM.

6. Discussion

The addition of PETSc was eased by the PETSc Fortran interface, simple control structure in ParaFEM, and use of the PETSc control file. It was made more difficult by the need to pre-allocate memory for the matrix and the handling of non-zero restraints.

ParaFEM's unassembled matrix solvers are competitive with PETSc's assembled matrix solvers – importantly, the ParaFEM solvers use much less peak memory than the PETSc solvers. An unassembled matrix type could be added to PETSc and used with the existing solvers (the matrix-vector product would have to be written) but most of PETSc's preconditioners need an assembled matrix to set up and apply the preconditioner (the exception is Jacobi – it is easy to extract the diagonal). The test cases all used Jacobi preconditioning – other preconditioners (or using direct solvers) were slower overall. The

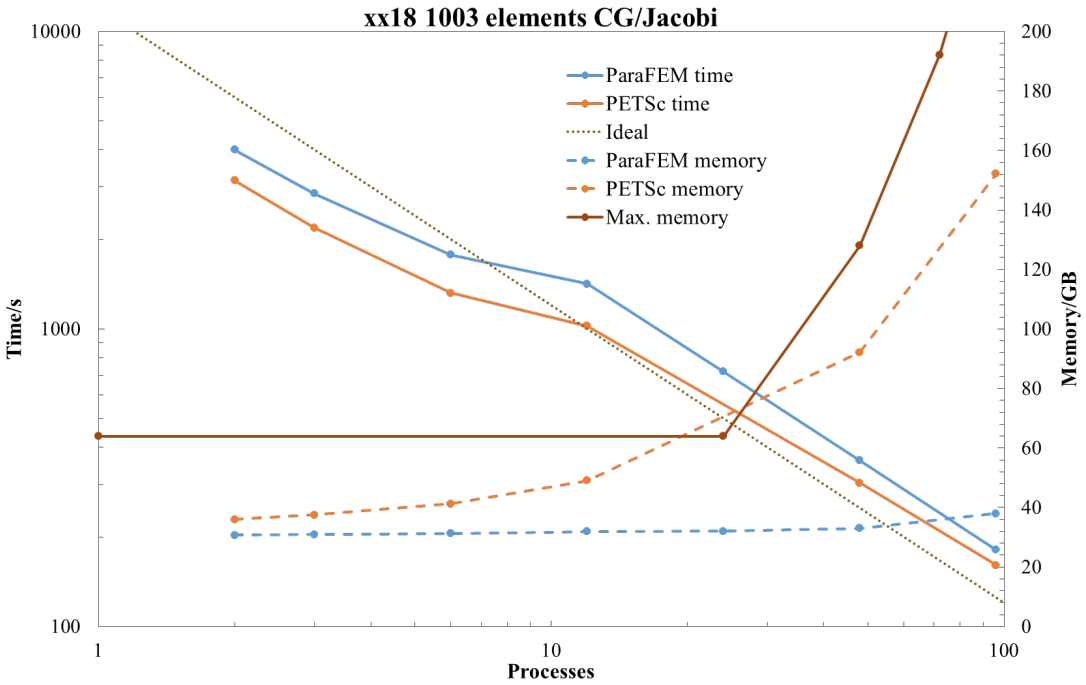

Figure 2: Time and memory scaling of CG/Jacobi for xx15 with ~7.5M elements. Time is the sum of the matrix assembly, preconditioner setup and solver times. Memory is the peak memory use. Cray XC30, Intel E5-2650 v2 processors with 12 cores per processor.

algebraic multigrid preconditioner in PETSc was not tested – it uses smoothed aggregation, which needs the full set of DOF to be kept at each node, and ParaFEM removes restrained DOF to get a reduced set of equations. Using algebraic multigrid should give improved performance but would require a radical change in the ParaFEM design.

The techniques developed here may be useful for interfacing PETSc to other FEM programs. The main points are:

- Use PETSc's pre-allocation calculator – this is slower than the algorithm developed here but is much easier to use
- Use PETSc's control file to choose solvers and preconditioners
- Wrap up PETSc as much as possible to look like the FEM program code but leave the ability to use PETSc directly

- PETSc provides load/time-stepping and nonlinear solvers – these could be used but they currently do not include arc-length methods, which are interesting from a structural point of view

7. Getting the ParaFEM library with the PETSc interface

It is recommended to first register on the ParaFEM website (http://parafem.org.uk). The ParaFEM code is available on SourceForge in the Subversion repository https://sourceforge.net/projects/parafem. The ParaFEM-PETSc interface is in the PETSc branch – this will eventually be merged into the trunk. The current stable revision of the PETSc branch is 2237.

8. Acknowledgements

This work was funded under the embedded CSE programme of the ARCHER UK National Supercomputing Service (http://www.archer.ac.uk).

References

[1] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc Web page. http://www.mcs.anl.gov/petsc, 2016. URL http://www.mcs.anl.gov/petsc.

[2] I. M. Smith, D. V. Griffits, and L. Margetts. *Programming the Finite Element Method*. Wiley, 2013.