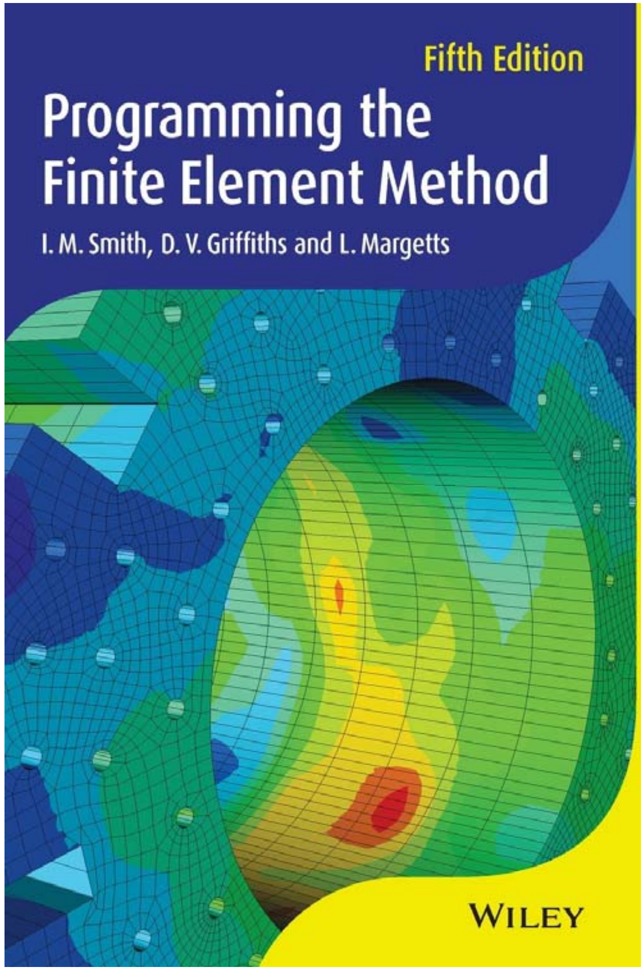
Implementation of generic solving capabilities in ParaFEM

Mark Filipiak, EPCC, University of Edinburgh m.filipiak@epcc.ed.ac.uk
Francesc Levrero Florencio, University of Oxford
Lee Margetts, University of Manchester
Pankaj Pankaj, University of Edinburgh

archer eCSE

ParaFEM


Fifth Edition
Programming the Finite Element Method
I. M. Smith, D. V. Griffiths and L. Margetts
WILEY

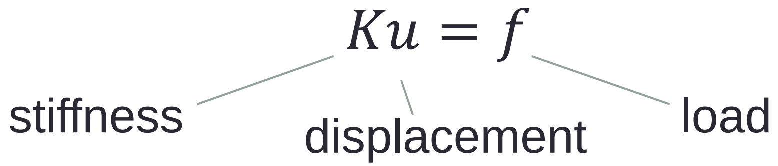
- Finite Element Method
- Open source library + ~70 mini Apps
- Parallel, scaling to 64,000 cores
- >1 billion degrees of freedom
- Used for teaching and research
- 1000+ registered on website
- ~1400 citations of text book

http://parafem.org.uk

http://www.amazon.com/Programming-Finite-Element-Method-Smith/dp/1119973341

FEM has a linear solver at its core

- In finite element modelling programs such as ParaFEM, the step that takes the most time is the linear solve of the large, sparse matrix equation



$$Ku = f$$

stiffness

displacement

load

- A typical program in ParaFEM would be

```
Read in grid, material properties, BCs, loads
Create stiffness matrix K and vectors u, f
DO time step or load step
  DO Newton-Raphson iteration
        Set the entries in K and f
        Solve Ku = f
  END DO
END DO
```

Linear solvers

ParaFEM currently implements two parallel iterative linear solvers:

- Conjugate Gradient (CG), for elastic models
- BiCGStab(*l*), for plastic and flow models – slower than CG

both with Jacobi preconditioning

Adding a range of other solvers and preconditioners can give a better match to the wide range of possible types of algebraic systems found in FEA and shorter time to solution, e.g.,

- Parallel direct solvers can be faster than iterative solvers for smaller systems
- The stiffness matrix in large strain solid mechanics simulations may become symmetric indefinite in the post-buckling regime and MINRES can be used instead of BiCGStab(*l*)
- Many other preconditioners can be used instead of Jacobi, for example algebraic multigrid (AMG). These will reduce the number of iterations of the solver but the each iteration will be slower – overall the result is usually shorter time to solution

Solver libraries

PETSc (http://www.mcs.anl.gov/petsc)

- Wide range of iterative linear solvers, preconditioners, but lots more as well: non-linear solvers, time integration, mesh handling
- High level interface, e.g. basic objects include matrices (Mat) and vectors (Vec)
- C with Fortran interface
- Interface to other libraries (Trilinos algebraic multigrid, hypre algebraic multigrid and ILU, MUMPS and Super_LU direct solvers)

Trilinos (http://trilinos.org)

- Similar to PETSc in its range of solvers and tools
- C++; Fortran interface out-of-date, being re-written
- Interface to other libraries (PETSc, MUMPS, Super_LU)

Using PETSc directly

‘Create stiffness matrix K’

```
Mat :: K
CALL MatCreate(PETSC_COMM_WORLD,K,ierr)
CALL MatSetSizes(K,PETSC_DETERMINE,PETSC_DETERMINE,neq,neq,
CALL MatSetType(K,MATAIJ,ierr)
!calculate nnz = estimate of number of entries in a row
CALL MatSeqAIJSetPreallocation(K,nnz,PETSC_NULL_INTEGER,ier
CALL MatMPIAIJSetPreallocation(K,nnz,PETSC_NULL_INTEGER, &
                               nnz,PETSC_NULL_INTEGER,ierr)
```

‘Set the entries in K ’

```
CALL MatZeroEntries(K,ierr)
DO iel=1,nels
  !calculate values = element stiffness matrix
  !calculate rows = cols = local to global index mapping
  CALL MatSetValues(K,nrows,rows,ncols,cols,values,ADD_VALU
END DO
CALL MatAssemblyBegin(K,MAT_FINAL_ASSEMBLY,ierr)
CALL MatAssemblyEnd(K,MAT_FINAL_ASSEMBLY,ierr)
```

ParaFEM-PETSc interface

- Wrap up the PETSc tasks to be equivalent to the ParaFEM tasks
- but still have a way to choose between ParaFEM and PETSc solvers, and the various solvers and preconditioners.

ParaFEM characteristics

- Two ParaFEM solvers as subroutines
- The most complicated control sequence is for time/load stepping of a non-linear solution

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

- One matrix, one solution vector and one load vector, all stored as Fortran arrays. The matrix structure is fixed but the matrix entries may change in non-linear problems
- The matrix is stored in unassembled form
- Global mappings between elements and degrees-of-freedom (~nodes) are set up by ParaFEM from mesh connectivity data
- The driver programs use a control file to set solver tolerances, etc.
- Equations corresponding to restrained degrees of freedom (i.e., homogeneous Dirichlet BCs) are removed from the system

PETSc characteristics

- Matrices and vectors are stored as C structures, Mat and Vec
- Matrices are stored in assembled form – this means that there is an extra step when assembling a matrix
- The memory needed to store the matrix needs to be allocated before the actual assembly, otherwise the assembly becomes 50 times slower.
    - PETSc can calculate the amount of memory needed
- PETSc can use a control file to set the solver and preconditioner used, solver tolerances, etc.

Interface design

- The driver program can be written to use the ParaFEM solvers, or the PETSc solvers, or can choose between them at run time
- The ParaFEM matrix and vectors are converted to PETSc Mat and Vec structures when using the PETSc solvers. The standardized data structures in ParaFEM allow the PETSc data structures to be held in one global data structure.
- Control of PETSc is by the standard PETSc control file, not by specifying options in the ParaFEM control file and translating to PETSc
  - Reduces maintenance
  - Several solvers and preconditioners can be listed in the control file and chosen during program execution
- Direct calls to PETSc are still possible.
- PETSc non-linear and time-/load-stepping routines are not used.

Interface design: example

- An example of the solver sequence for time-stepping (or load-stepping) problems when using PETSc is

```
Initialise PETSc (p_initialize)
Create PETSc matrix K and vectors x,f (p_create)
DO time/load step
  DO Newton-Raphson iteration
    Clear the global matrix (p_zero_matrix)
    DO element
      Calculate element stiffness matrix km
      Add km to global matrix K (p_add_element)
    END DO
    Complete the setup of K (p_assemble)
    Solve Ku = f (p_solve)
  END DO
END DO
```

Interface design: wrappers

Name	Description
get_solvers	Get the name of the solvers to use from the command line
p_initialize	Initialize 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 matrix
p_assemble	Assemble the PETSc matrix
p_zero_rows	Modify the PETSc global matrix and the load vector by the fixed freedoms. The resulting matrix in not symmetric.
p_use_solver	Choose one of the PETSc solvers specified in the .petsc configuration files
p_solve	Solve using PETSc
p_shutdown	Destroy the PETSc data structures and finalize PETSc.

Interface design: PETSc control file

```
-nsolvers 2
-prefix_push solver_1_ # call p_use_solver(1,...)
  -ksp_type cg
  -ksp_rtol 1.0e-5
  -ksp_max_it 2000
  -pc_type jacobi
-prefix_pop
-prefix_push solver_2_ # call p_use_solver(2,...)
  -ksp_type minres
  -ksp_rtol 1.0e-5
  -ksp_max_it 2000
  -pc_type jacobi
  -pc_jacobi_abs
-prefix_pop
```

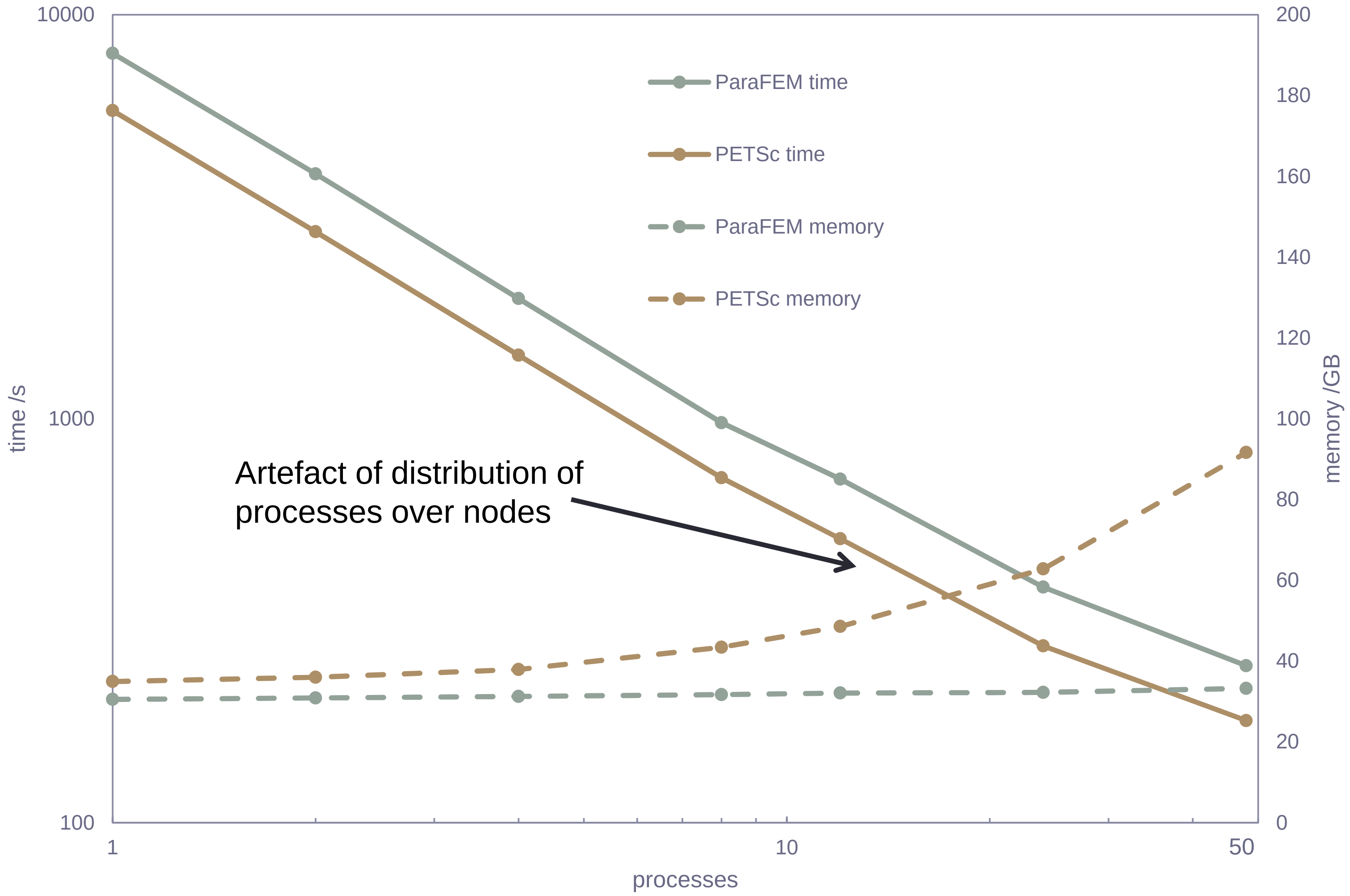
PETSc vs ParaFEM

- The main reason to add PETSc to ParaFEM is to increase the range of solvers but we also compared the PETSc and ParaFEM solvers for 3 driver programs
  - xx18: a linear elastic solid system in a small strain regime. This is an example of a symmetric positive definite case (CG/Jacobi)
  - xx17: steady state cavity driven flow of an incompressible Navier – Stokes fluid. This is an example of an unsymmetric case (BiCGStab(*l*))
  - xx15: large strain solid mechanical modelling of trabecular bone. This is an example of a symmetric case that starts as positive definite and becomes indefinite at in the post-buckling regime – this was modelled only in the pre-buckling regime (CG/Jacobi)
- Tested on TDS – a single-cabinet (1000 core) version of Archer, with about 2x communications B/W of Archer

xx18 description

- Three-dimensional analysis of a linear elastic solid
  - cuboid with a uniformly loaded patch at the centre of one face
  - 20-node hexahedral finite elements
  - 100^3 element grid (12M equations)
  - Conjugate gradient with Jacobi preconditioning
- Processes were distributed equally across 8 nodes
  - slowdown is an artefact
    - as the number of processes per node increases, the memory bandwidth per process is reduced (and FE codes are generally memory bound)
    - Turbo Boost is enabled on ARCHER processors, so that the clock speed will increase when fewer cores on a processor are active.

xx18 scaling



Artefact of distribution of processes over nodes

xx18 discussion

- PETSc is about 30% faster than ParaFEM in this case. Both have similar scaling.
- PETSc requires much more memory as the number of processes increases. During the assembly of the matrix off-process entries are stored temporarily before they are distributed with p_assemble.
  - On distributed-memory computers, the total memory available will usually increase faster than the peak memory requirement
  - The number of off-process entries depends on how well the problem has been partitioned across the processes

xx17

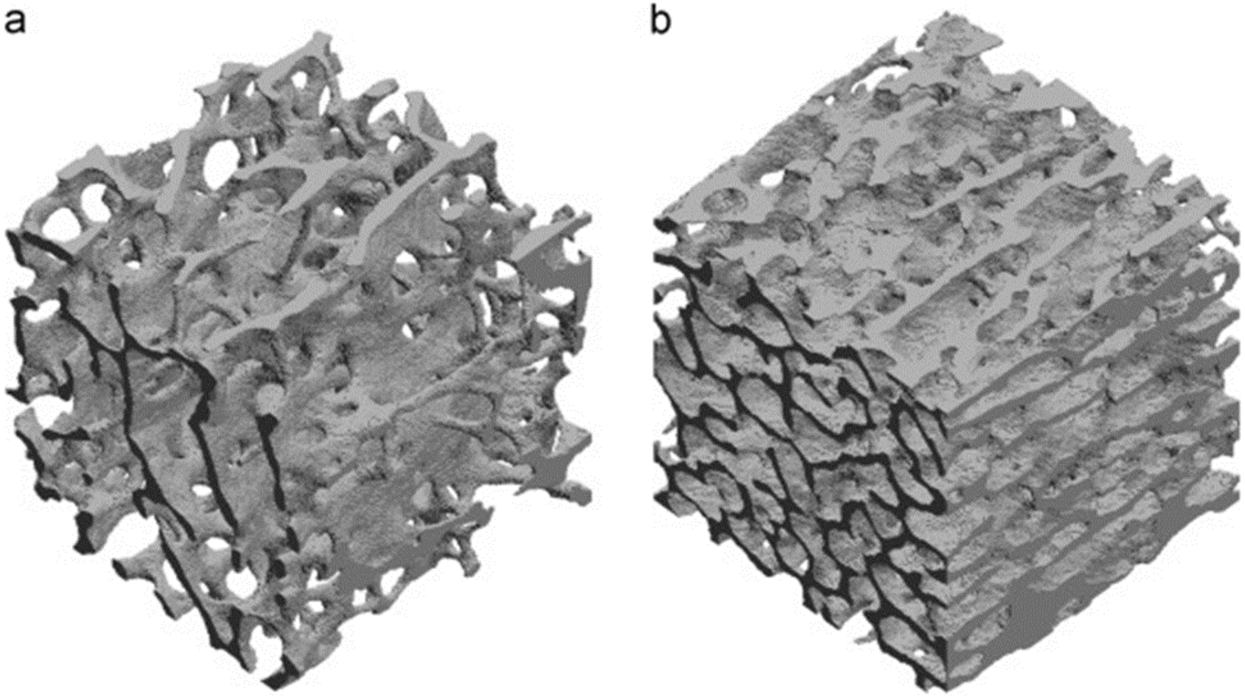
- 3D steady-state flow of an incompressible fluid in a driven cavity, discretized with Taylor-Hood elements (20-node hexahedra for velocity, 8-node hexahedra for pressure)
- Non-linear, solved using a Newton-Raphson solver
- Unsymmetric matrix, solved using BiCGStab(*l*=4) with no preconditioning
- Non-zero restraints (the velocity of the driven boundary) – uses the interface routine p_zero_rows
- Small, medium and large cases were tested
  - In the small case, the results are the same for ParaFEM and PETSc within the solver tolerance
  - In the medium and large cases there are differences larger than the solver tolerance, even if reduce and scatter operations are ordered. The PETSc BiCGStab(*l*) diverges at a particular N-R iteration causing extra N-R iterations to achieve convergence if the N-R solver
  - The BiCGStab(*l*) algorithm in PETSc is a modified version, which may explain the differences
- Further work is needed to resolve the numerical differences and the cause of the solver divergence

xx15 description

- Large strain plasticity driver program used in solving mechanical systems involving buckling and elastoplastic softening problems
- Non-linear, solved using a Newton-Raphson solver
- The orthopaedic engineering research group at the University of Edinburgh uses xx15 to simulate the mechanical behaviour of trabecular bone

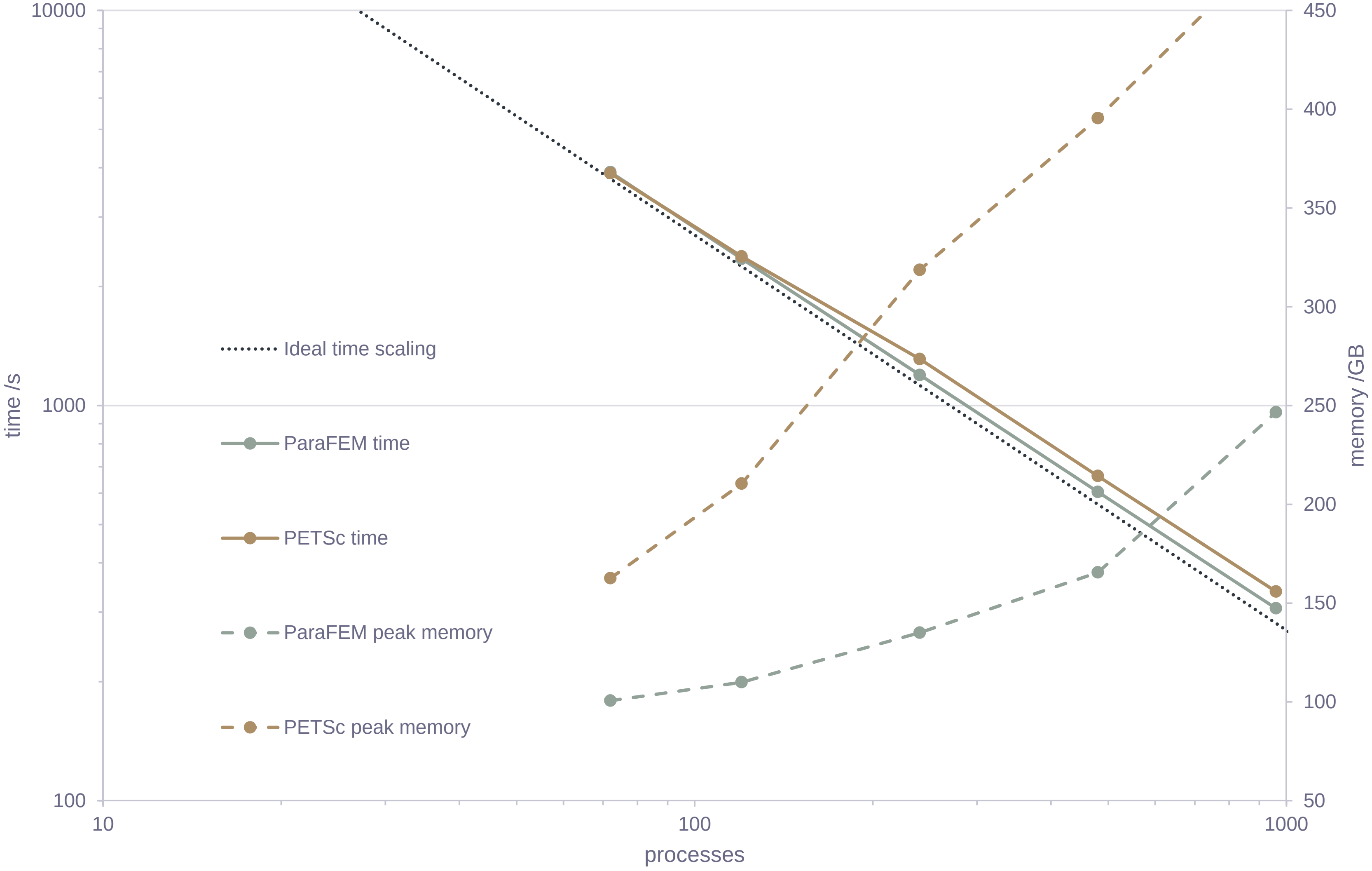
xx15: bone modelling test case

- The detailed geometry of the bone structure is generated through X-ray microtomography
- 7.5M elements (8-node hexahedra), 26M dofs



- The load was kept in the range where the stiffness matrix remained symmetric positive definite, so CG/Jacobi could be used

xx15 scaling



PETSc vs ParaFEM: summary

- PETSc and ParaFEM have similar time performance for the same solvers
- PETSc requires much more memory that ParaFEM
- The main value in adding PETSc to ParaFEM is to make a wide range of solvers available in ParaFEM so that users can choose the most suitable solver for their problem and even change this during the simulation

Getting ParaFEM with PETSc

- Register on the ParaFEM website http://parafem.org.uk.
- ParaFEM Subversion repository is on SourceForge https://sourceforge.net/projects/parafem
- 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
  - Build script for ARCHER
  - User guide in same format as Chapter 12 of ‘Programming the Finite Element Method’ by I.M. Smith, D.V. Griffiths and L. Margetts, 5th edition, 2014
  - Example programs, PETSc control files, Archer job scripts, and test case generation tools for xx18, xx17 and xx15

Conclusion

- ParaFEM now has a wide range of solvers and preconditioners available through the PETSc interface

Reusing this material

Note that this presentation contains images owned by others. Please seek their permission before reusing these images.

Assembled matrix, or …

In some finite element programs and in most linear solver toolkits, the global stiffness matrix K is assembled before use:

$$K = \sum_{i}^{\text{elements}} S_i K_i$$

where K_i are the element stiffness matrices and S_i are the mappings from element numbering to global numbering (*scatter* in ParaFEM)

- Most preconditioners in PETSc expect assembled matrices. An unassembled Mat type could be defined but it would need to include any operations required by the preconditioner (e.g. get the diagonal values for Jacobi)

… unassembled matrix

ParaFEM uses unassembled matrices

- An iterative solver only needs to evaluate the matrix-vector product, and that is achieved using *scatter* and *gather* on the vectors:

$$\sum_{i}^{\text{elements}} S_i K_i S_i^T u = f$$

where S_i^T are the mappings from global numbering to element numbering (*gather* in ParaFEM)

- No assembly is needed but a scatter is needed at every iteration
- (For sparse matrices, unassembled and assembled matrices *both* have a gather every iteration)