Manchester Computing
Supercomputing, Visualization & e-Science

Finite Element Analysis on the Cray MTA

Jon Gibson and Mike Pettipher

Date 15th May 2003
Event: Cray User Group
Venue: OSC, Columbus, Ohio

THE UNIVERSITY of MANCHESTER

Acknowledgements

- Acknowledgments to:
  - Apologies to Jon Gibson – we were not prepared to pay him to come and present this work.
  - Cray Inc. for giving us access to an MTA system.
  - Simon Kahan (in particular) for quickly identifying (and fixing) some obscure problems.

Contents from another presentation

- Parallel Finite Element Analysis at the University of Manchester
  - Program philosophy, application areas
  - Parallelisation strategy
- Performance
  - Cache issues
  - Communication
  - Scalability
- The Simple Approach – Cray MTA-2
- Applications / projects
  - DNS Navier Stokes
  - DNS Magnetohydrodynamics
  - Geotechnical Engineering, University of Manchester
  - Advanced Virtual Prototyping Research Centre (AVPRC) - consortium of four UK universities.
- [Objective – to show that engineers with little or no experience of HPC can fairly easily use parallel versions of codes with which they are familiar.]

Finite Element Analysis – Serial Programs

- Template serial programs
  - Written by Professor Ian Smith.
    - Author of a number of books on FEA and Fortran 90.
    - NAG Finite Element Analysis Library based on these codes.
  - Written in Fortran 90.
  - Modular structure – ‘building blocks’ for general FEA modelling – users adapt these codes for their own requirements.
  - Element by element method using iterative solvers, PCG, BiCGStab(l), Lanczos.
  - Widely used by engineers worldwide.

Template Serial Codes

- All the usual FEA application areas covered
  - Material behaviour, elasticity, plasticity
  - Heat, fluid flow
  - Dynamics, forced vibrations
  - Coupled physical processes, such as magnetohydrodynamics

Finite Element Analysis - Parallel Programs (MPI)

- Serial syntax/structure preserved as much as possible.
- Philosophy of serial code templates preserved – users adapt codes for own requirements.
- Parallel coding hidden away in libraries:
  - MPI routines for communication
  - Other ‘utility’ routines to manage data distribution etc.
- Intended that users with minimal parallel computing knowledge can adapt the parallel code templates for their own particular problems.

Element by Element (EBE) – Inherent Parallelism

- Global (sparse) matrix never assembled.
- All codes dominated by loops over the elements (nels):
  - Set up Global arrays
    - elements_1: do i = 1, nels
  - Element stiffness integration, storage and preconditioner
    - elements_2: do i = 1, nels
  - Start load increments
  - Start plastic iterations
  - Perform conjugate gradient iterations
    - elements_3: do i = 1, nels
  - Go round Gauss points
    - elements_4: do i = 1, nels
  - End plastic iterations
  - End load increments.
- Can parallelise all loops over elements

PCG Solver

- Time dominated by PCG solver, which is dominated by matrix-vector computation section:

- DO i = 1, nels

```
  …
  pmul = p(g)                    ! gather
  utemp = matmul(km,pmul)        ! matrix-vector
  u(g) = u(g) + utemp            ! Scatter
  …
END DO
```

Elasto-plastic analysis

- km matrix can be made the same for all elements, so can use matrix-matrix:
-
```
DO iel = 1, nels
  DO i= 1, ndof                          ! ndof = 60
    pmul(i, iel) = p(g(i,iel))           ! gather
  END DO
END DO

CALL DGEMM(…)                            ! matrix-matrix

DO iel = 1, nels
  DO i= 1, ndof
    u(g((i, iel)) = u(g(i,iel)) + utemp(i,iel)
  END DO
END DO
```

MPI Implementation

- Gather/scatter require significant communication.
- Wrote our own routines to identify all communications, packing and sending data once for all gathers and once for all scatters - small number of large messages.
- Took substantial time to develop (debug!) code.

Elasto-plastic analysis – MPI results

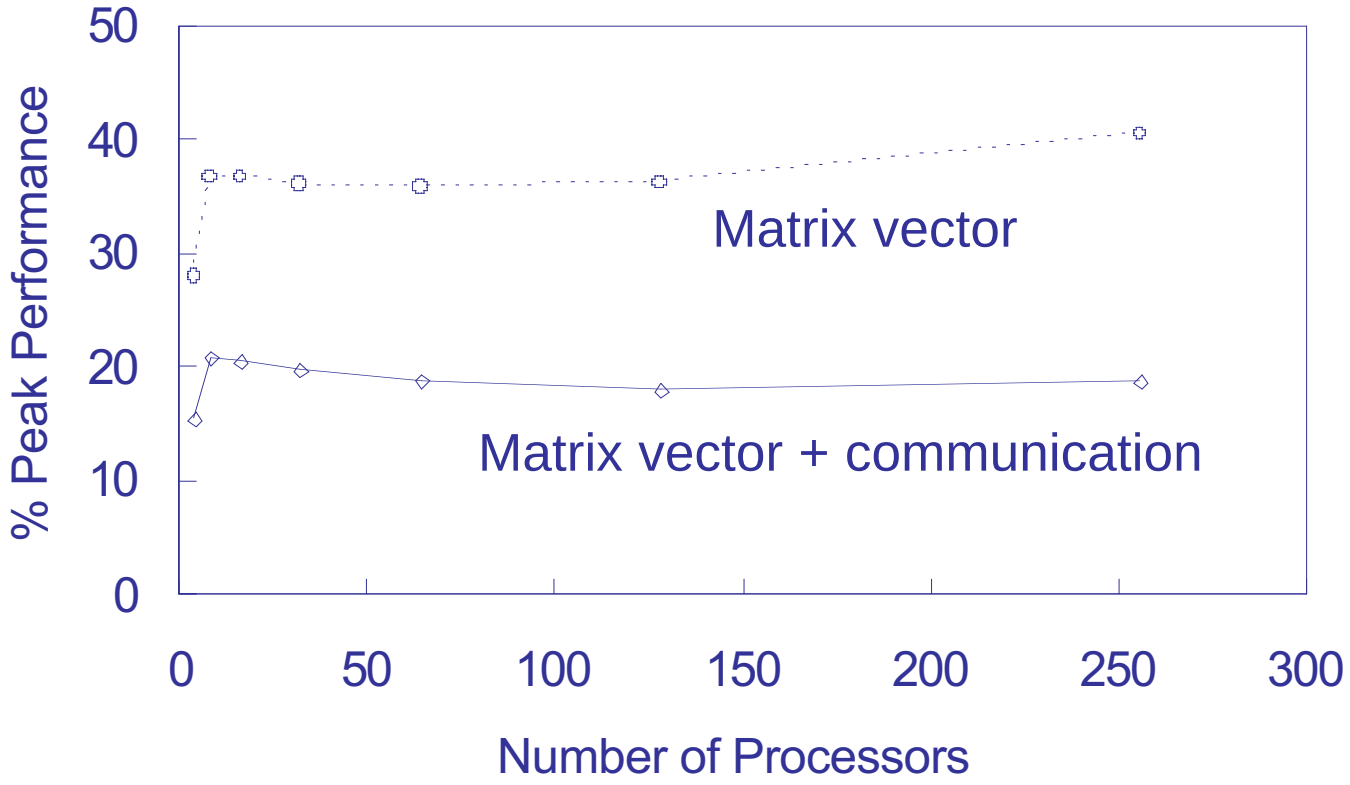
- Stiffness matrix can be the same for each element – giving rise to matrix-matrix rather than matrix-vector computations.
- Results from Cray T3E, 64000 elements:

Processors	8	512
PCG elapsed time	104.9	3.1
PCG speedup	1.0	33.8 (out of 64)
PCG % peak	22%	12%

Magnetohydrodynamics example

- Discretisation using the finite element (EBE) method
  - 20 node quadrilateral bricks
  - Navier Stokes solver
  - Iterative solution algorithm BiCGStab(l)
  - BiCGStab(l) necessary to deal with unsymmetric stiffness matrix
- Different stiffness matrix for each element => large stiffness matrix storage and results, by default, in poor cache re-use, and consequently in poor performance.
- Can find common structure in stiffness matrices, and with recoding, improve cache and overall performance …

MPI implementation - Percentage Peak Performance



~20% peak sustained on 256 processors on Origin 3800 for whole code

Factors limiting performance

- There are two major factors affecting the performance in the MPI implementation
  - Time for gather and scatter of data is significant, although it does not affect scaling.
  - Cache re-use in matrix-vector operations required special coding to avoid performance problems.
- Thus ‘good’ performance has been achieved, but had to spend significant time writing routines for gather/scatter and cache re-use.

- Can the Cray MTA-2 do better?

The Cray MTA-2

- **M**ulti-**T**hreaded Multiple active threads (up to 128) on each processor.
  - Used to hide latency.
  - 16 to 256 processors.
- Scalable uniform access to global shared memory.
  - 2.4GB/s bandwidth.
  - 4GB of memory per processor.

MTA Ease of Programming

- No data cache
- No stride sensitivity
- No message passing
- Loop-based parallelism
- Uniform access to global memory.
- Dynamic scheduling of tasks
- Memory-based synchronization
- Low application development costs

Programming the MTA-2
What is Irrelevant?

- Cache misses
- Poor data locality
- Static assignment of tasks to processors
- Poor communication to computation ratio
- Parallelism that is too fine or too coarse to exploit effectively

FEA on the MTA

- Memory structure of Cray MTA-2 should avoid or reduce the problems encountered in the MPI implementations of the FEA codes…
  - Flat memory should remove cost of distributed memory gather and reduce that of the scatter.
  - Memory latency hidden by using multiple threads so cache use, etc, not an issue.
  - With the ‘EBE’ method, the programs require very few changes to the serial codes – just some Open MP style directives.
  - Ideal for this suite of general finite element analysis codes.

MTA v MPI on an Origin

- Using Elasto-plastic code, 64000 elements
- SGI Origin 3800, 400MHz, MPI:

Processors	1	2	4
Conj Grad loop	787.1	393.5	204.7
Matrix mult	378.6	192.2	98.0
Gather/scatter	254.2	132.2	72.0

- MTA-2, 200MHz:

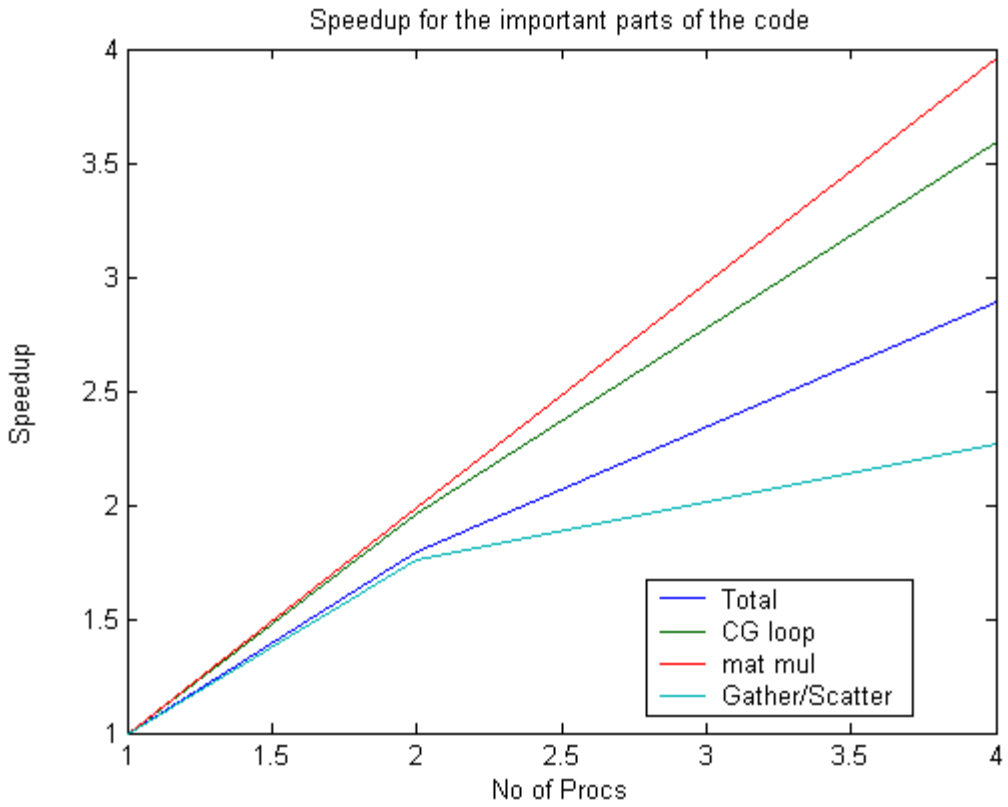
Processor	1	2	4
Conj Grad loop	841.3	429.8	234.1
Matrix mult	664.5	334.7	167.7
Gather/scatter	76.7	43.7	33.8

MPI versus MTA

- Time in gather/scatter much smaller, so even with much slower processor, MTA gives only slightly slower times than Origin/MPI.
- Coding of MTA version took a few hours from serial version (probably less than one hour for a programmer familiar with the MTA).
- MPI version took months!

- What about scaling?
- What about OpenMP – should also be simple to code?

Scaling on the MTA



MTA versus OpenMP on an Origin

- Using Elasto-plastic code, 64000 elements.
- SGI Origin 3800, 400MHz, OpenMP:

Processors	1	2	4
Conj Grad loop	678.4	473.2	359.0
Matrix mult	381.6	200.2	103.4
Gather/scatter	142.2	110.5	100.4

- MTA-2, 200MHz:

Processor	1	2	4
Conj Grad loop	841.3	429.8	234.1
Matrix mult	664.5	334.7	167.7
Gather/scatter	76.7	43.7	33.8

Gather/scatter

- MTA wins because gather/scatter worse with OpenMP.

- Scaling of gather/scatter on MTA not perfect – why?
- Time in gather insignificant – about 0.02s. Scatter does involve multiple updates to specific locations (potentially 8 elements share a single node), so there is contention.
- Scatter does not scale well – why?
- A little work by Simon Kahan identified a feature of the original serial code that was removed from the MPI version. A simple fix improved the scaling. (This fix introduces an overhead, but it should be possible to remove it):

Scaled scatter

- Using Elasto-plastic code, 64000 elements.
- MTA-2, original

Processors	1	4
Conj Grad loop	841.3	234.1
Matrix mult	664.5	167.7
Gather/scatter	76.7	33.8

- MTA-2, updated scatter:

Processor	1	4
Conj Grad loop	952.1	289.6
Matrix mult	670.5	167.9
Gather/scatter	101.6	25.7

Summary

- The MTA gives comparable performance to an MPI implementation and better performance than an OpenMP implementation, on a system with twice the clock rate.
- The coding time on the MTA is insignificant compared with MPI.
- The code used is the basis of a suite of FEA codes, serial versions of which are widely used.
- If engineers (without HPC expertise) were not impressed by our MPI templates, they may (should) be by the simpler approach on the MTA.

Manchester Computing
Supercomputing, Visualization & e-Science

SVE @ Manchester Computing

World Leading Supercomputing Service,
Support and Research

Bringing Science and Supercomputers Together

www.man.ac.uk/sve
sve@man.ac.uk

THE UNIVERSITY of MANCHESTER