1 Objective

The aim of the work was to install the **Trilinos** software library on the **HPCx** service and then to investigate its performance using test problems from the IFISS suite.

Trilinos is a collection of compatible software packages that support parallel linear algebra computations, solution of linear, non-linear and eigen systems of equations and related capabilities. The majority of packages are written in C++ using object-oriented techniques and use MPI. **Trilinos** is produced by Sandia National Labs.

HPCx is the UK science community's newest capability computing service, operated by EPCC and CCLRC on behalf of the EPSRC. At the time of use, it comprised 96 IBM eServer 575 LPARs available for computation, each with IBM Power5 processors. The Power5 is a 64 bit RISC processor with a 1.5 GHz clock rate. Each eServer node contains 16 processors and 32 Gbytes of main memory. At the time of use, the maximum number of processors that could be used by a single job was 1024. (`http://www.hpcx.ac.uk`)

IFISS is a set of test problems from [Elman, Silvester, Wathen]. Convection-diffusion problems CD2 and CD4 are the two of interest here.

2 Installation

2.1 Download

The latest version of the software available (7.0.3 in this case) needs to be downloaded as a tar file from the **Trilinos** website

`http://software/sandia.gov/Trilinos`

to **HPCx** and then the commands

```
gunzip trilinos-7.0.3.tar.gz
tar xf trilinos-7.0.3.tar
```

unpack the software into a directory named `trilinos-7.0.3`.

2.2 Build

Create a subdirectory named `HPCx`, i.e. `trilinos-7.0.3/HPCx`, and in this directory create a file named `conf_script` containing the necessary *build flags*.

2.2.1 Parallel Library

The following version of `conf_script` builds a parallel library containing the packages needed for the two IFISS problems.

```
../configure \
--disable-default-packages \
--enable-galeri \
--enable-aztecoo \
--enable-ml \
--enable-epetraext \
--enable-amesos \
--enable-anasazi \
--disable-tests \
--disable-examples \
CC='xlc_r -q64 -O3 -qarch=pwr5 -qtune=pwr5' \
CXX='xlC_r -q64 -qrtti=all -O3 -qarch=pwr5 -qtune=pwr5' \
F77='xlf_r -q64 -O3 -qarch=pwr5 -qtune=pwr5' \
--prefix=$HOME/trilinos7-mpi-libs \
--enable-mpi \
--with-mpi-incdir=/usr/lpp/ppe.poe/include \
--with-mpi-libs=-lmpi_r \
--with-mpi-libdir=/usr/lpp/ppe.poe/lib \
--with-blas="-lessl -llapack -L/usr/local/lib -L/usr/lib" \
--with-lapack=/usr/local/lib/liblapack.a
```

2.2.2 Configure and Make

Now, from the `HPCx` directory, initiate the (lengthy) configuration stage with the command

```
./conf_script
```

Ensure that the configuration has completed successfully. Next, the environmental variable `OBJECT_MODE` needs to be set before starting the `make`.

```
export OBJECT_MODE=64
make
```

Once `make` has completed successfully then

```
make install
```

completes the process. A directory `trilinos7-mpi-libs` should now exist in the home directory.

2.2.3 Serial Library

An equivalent `conf_script` file to build a sequential version of the library is

```
../configure \
--disable-default-packages \
--enable-galeri \
--enable-aztecoo \
--enable-ml \
--enable-epetraext \
--enable-amesos \
--enable-anasazi \
--disable-tests \
--disable-examples \
CC='xlc_r -q64' \
CXX='xlC_r -q64 -qrtti=all' \
F77='xlf_r -q64' \
--prefix=$HOME/trilinos7-seq-libs \
--disable-mpi \
--with-blas="-lessl -llapack -L/usr/local/lib -L/usr/lib" \
--with-lapack=/usr/local/lib/liblapack.a
```

2.2.4 Notes

- The most common build flags are documented in the **Trilinos** *Users Guide*.
- **Trilinos** comprises a number of separate packages, and it is more efficient to build only the ones that are needed. Do this by using the `disable-default-packages` flag, and then enabling those that are wanted. Any other packages required by the selected ones will be included automatically.
- The script above uses the commands `xlc_r`, `xlC_r` and `xlf_r` which are, in fact, the ones used for compiling sequential (non-MPI) code. There is a good reason for this : the configuration process for **Trilinos** involves compiling and running simple executables, but if these are produced with the 'MPI code compilers' `mpcc_r`, `mpCC_r` or `mpxlf_r`, `configure` will fail. This is because on **HPCx** MPI jobs *must* be handled by the `Loadleveler` job scheduler, but `configure` does not do this. To avoid this problem, specify the 'sequential' compiler commands and then explicitly link to the MPI libraries – this is actually equivalent to specifying the MPI commands in the first place.

3 Hold On a Minute

3.1 Bug Fixes

The following five problems need fixing at this point.

1 :> **File** `trilinos7-mpi-libs/include/Galeri_FiniteElements.h` **L**31–57.

```
#include "FiniteElements/Galeri_Workspace.h"
// Finite element includes:
// - abstract files
#include "FiniteElements/Galeri_AbstractGrid.h"
#include "FiniteElements/Galeri_AbstractQuadrature.h"
#include "FiniteElements/Galeri_AbstractProblem.h"
#include "FiniteElements/Galeri_AbstractVariational.h"
// - grid files;
#include "FiniteElements/Galeri_FileGrid.h"
#include "FiniteElements/Galeri_TriangleRectangleGrid.h"
#include "FiniteElements/Galeri_QuadRectangleGrid.h"
#include "FiniteElements/Galeri_TetCubeGrid.h"
#include "FiniteElements/Galeri_HexCubeGrid.h"
// - quadrature files;
#include "FiniteElements/Galeri_AbstractQuadrature.h"
#include "FiniteElements/Galeri_QuadRectangleGrid.h"
#include "FiniteElements/Galeri_TriangleQuadrature.h"
#include "FiniteElements/Galeri_QuadQuadrature.h"
#include "FiniteElements/Galeri_TetQuadrature.h"
#include "FiniteElements/Galeri_HexQuadrature.h"
// - variational files
#include "FiniteElements/Galeri_SUPGVariational.h"
#include "FiniteElements/Galeri_GalerkinVariational.h"
// - problem files
#include "FiniteElements/Galeri_LinearProblem.h"
// - other files
#include "FiniteElements/Galeri_MEDITInterface.h"
```

Fix : remove all occurrences of the string `'FiniteElements/'` (20 occurrences).

2 :> **File** `trilinos7-mpi-libs/include/Galeri_SUPGVariational.h` **L**288–294.

```
// SUPG stabilization
res += tau_ * ((conv_x(z, y, z) * PsiX +
                conv_y(x, y, z) * PsiY +
                conv_z(x, y, z) * PsiZ) *
               (conv_x(z, y, z) * PhiX +
                conv_y(x, y, z) * PhiY +
                conv_z(x, y, z) * PhiZ));
```

Fix : `(x, y, z)` not `(z, y, z)` in 2 places (L198 and L201).

3 :> **File** `trilinos7-mpi-libs/include/Galeri_SUPGVariational.h` **L**303–308.

```
// Galerkin contribution
res = force(x,y,z)*Psi;
// SUPG stabilization
res += tau_ * (conv_x(x, y, z) * PsiX +
               conv_y(x, y, z) * PsiY +
               conv_z(x, y, z) * PsiZ);
```

Fix :

```
// Galerkin contribution and SUPG stabilization
res = force(x,y,z) * (Psi + tau_
                  * (conv_x(x, y, z) * PsiX +
                     conv_y(x, y, z) * PsiY +
                     conv_z(x, y, z) * PsiZ) );
```

4 :> **File** `trilinos7-mpi-libs/include/Galeri_SUPGVariational.h` **L**326–328.

```
double cx = conv_x(z, y, z);
double cy = conv_x(z, y, z);
double cz = conv_x(z, y, z);
```

Fix :

```
double cx = conv_x(x, y, z);
double cy = conv_y(x, y, z);
double cz = conv_z(x, y, z);
```

5 :> **File** `trilinos7-mpi-libs/include/Galeri_QuadQuadrature.h` **L**203–207.

```
/* transformation from the actual to the reference */
J_(0,0) =   divide_by * ijacobian[0][0];
J_(1,0) = - divide_by * ijacobian[0][1];
J_(0,1) = - divide_by * ijacobian[1][0];
J_(1,1) =   divide_by * ijacobian[1][1];
```

Fix :

```
/* transformation from the actual to the reference */
J_(1,1) =   divide_by * ijacobian[0][0];
J_(0,1) = - divide_by * ijacobian[0][1];
J_(1,0) = - divide_by * ijacobian[1][0];
J_(0,0) =   divide_by * ijacobian[1][1];
```

3.2 Test Problems

The following build flags will build only the test and example programs for the **Galeri** package.

```
--disable-tests \
--enable-galeri-tests \
--disable-examples \
--enable-galeri-examples
```

The commands `make run-tests` and `make run-examples`, issued from the directory `trilinos-7.0.3/HPCx`, will run the problems (for a sequential build) and check the output.

3.2.1 Notes

- The compiler flag `-qrtti=all` is needed for the tests to be passed.

4 Driver Program

4.1 AdvDiff2D

An example program provided with the **Galeri** package, `AdvDiff2D.cpp`, is a good starting point for an attempt to solve problem CD4. This is found in directory

`trilinos-7.0.3/packages/galeri/examples/FiniteElements`

JB produced a program, `AdvDiff2D_ifiss4.cpp`, compatible with version 6 of **Trilinos** which is being used here. Some further changes had to be made to it for **Trilinos** version 7 because of the introduction of the **Galeri** package.

The finite element setup routines are from the **Galeri** package and the GMRES preconditioner is smoothed aggregation AMG from *ML* in conjunction with *Aztecoo*'s GMRES solver.

Galeri provides a routine, `QuadRectangleGrid`, for uniformly discretising a rectangular grid with quadrilateral elements. JB wrote a variation of this, `QuadRectangleStretchedGrid`, which creates a grid of quadrilateral elements stretched in the `X` and `Y` directions. This needs to be explicitly placed in the directory `trilinos7-mpi-libs/include`.

Timings of some sections of the code were done using `MPI_Wtime();`.

4.2 Make files

4.2.1 Parallel Executable

To create an MPI executable, use the following make file.

```
TRILINOS_HOME= /hpcx/home/e22/e22/amd/trilinos7-mpi-libs

CXX= mpCC_r

LINKER= mpCC_r

CFLAGS = -DHAVE_CONFIG_H -q64 -c -qrtti=all -qarch=pwr5 -qtune=pwr5 -O3

INCLS=  -I$(TRILINOS_HOME)/include/ -I./ -I/usr/lpp/ppe.poe/include

LIBS= -L/usr/lpp/ppe.poe/lib \
    -L$(TRILINOS_HOME)/lib \
    -lgaleri -lml -lamesos -lanasazi -laztecoo \
    -lepetraext -ltriutils -lepetra -lteuchos \
    -L/usr/lpp/ppe.poe/lib -L/usr/local/lib -lessl -llapack -lblas -lmpi_r \
    -lxlf90 -L/usr/lpp/xlf/lib -lxlopt -lxlf -lxlomp_ser -lpthreads -lm  \
    -L/usr/lib

LFLAGS = -q64 -qarch=pwr5 -qtune=pwr5 -O3

PROG = AdvDiff2D_ifiss4

$(PROG): $(PROG).cpp
    $(CXX) $(CFLAGS) $(PROG).cpp $(INCLS)
    $(LINKER) $(LFLAGS) $(PROG).o $(LIBS) -o $(PROG).exe

clean:
    rm -f *.o *.exe
```

4.2.2 Sequential Executable

The make file for a sequential executable differs slightly.

```
TRILINOS_HOME=/hpcx/home/e22/e22/amd/trilinos7-seq-libs

CXX= xlC_r

LINKER= xlC_r

CFLAGS = -DHAVE_CONFIG_H -q64 -c -qrtti=all

INCLS=  -I$(TRILINOS_HOME)/include/  \
```

```
         -I$(TRILINOS_HOME)/ -I.

LIBS = -L$(TRILINOS_HOME)/lib \
       -lgaleri -lml -lamesos -lanasazi -laztecoo \
       -lifpack -lepetraext -ltriutils -lepetra -lteuchos \
       -L/usr/local/lib -lessl -llapack -lblas -L/usr/lib  \
       -lxlf90 -L/usr/lpp/xlf/lib -lxlopt -lxlf -lxlomp_ser -lpthreads -lm

LFLAGS = -q64

PROG = AdvDiff2D_ifiss4

$(PROG): $(PROG).cpp
    $(CXX) $(CFLAGS) $(PROG).cpp $(INCLS)
    $(LINKER) $(LFLAGS) $(PROG).o $(LIBS) -o $(PROG).exe

clean:
    rm -f *.o *.exe
```

4.2.3 Notes

- It is important to link to the the ESSL library before the BLAS because ESSL contains tuned versions of the BLAS routines.
- The *User's Guide to the HPCX Service* suggests trying the compiler option `-qhot` to optimise code performance. Here, it always caused the program to fail with an `Internal AztecOO Error; Error Code -5` which seems to be a general, non-specific error according to the source code in `AztecOO.cpp`.

5 Run

Having created an executable using the make files above, there are four different ways in which a job may need to be run : sequentially or in parallel; interactively (small development jobs only) or in batch.

Sequential jobs (built with a sequential version of the library and no MPI code) were initially necessary not only for testing, but because there were problems in getting the code to run at all. Running a sequential version at least eliminated the possibility that the problems were related to message passing.

5.1 Sequential code; interactive execution.

Execute with the command

```
./AdvDiff2D_ifiss4.exe
```

5.2 Sequential code; batch execution.

Create a Loadleveler script, in this case named `t10seq.ll`, containing the following lines.

```
#@ shell = /bin/ksh
#
#@ job_name = ifiss4seq
#
#@ job_type = serial
#@ node_usage = shared
#
#@ wall_clock_limit = 0:10:00
#@ account_no = e22
#
#@ output = $(job_name).$(jobid).out
#@  error = $(job_name).$(jobid).err
#@ notification = never
#
#@ queue
#
./AdvDiff2D_ifiss4.exe
```

and submit the job with the command

```
llsubmit t10seq.ll
```

5.3 Parallel code; interactive execution.

This is only for small test jobs in development. Create a Loadleveler script, in this case named `t5intp4.ll`, containing the following lines.

```
#@ job_name = intp4
#
#@ job_type = parallel
#@ cpus=4
#@ node_usage = shared
#
#@ wall_clock_limit = 0:05:00
#@ account_no = e22
#
#@ notification = never
#
#@class = inter32_1
#
#@ queue
```

```
#
```

Submit the job with the command

```
poe ./AdvDiff2D_ifiss4.exe -llfile ./t5intp4.ll
```

or, if parameters are to be provided to the `AdvDiff2D_ifiss4` program, then use for example

```
poe ./AdvDiff2D_ifiss4.exe --mx=2 --my=2 -llfile ./t5intp4.ll
```

`mx` and `my` define the number of processors allocated to the `x` and `y` directions by the code. However, if `mx * my` is not equal to the number of processors requested for the job, it will fail. Other parameters can be specified in a similar way.

5.4 Parallel code; batch execution.

Create a Loadleveler script, in this case named `t20p16.ll`, containing the following lines.

```
#@ shell = /bin/ksh
#
#@ job_name = ifiss4p16
#
#@ job_type = parallel
#@ cpus=16
#@ node_usage = not_shared
#
#@ bulkxfer = yes
#
#@ wall_clock_limit = 0:20:00
#@ account_no = e22
#
#@ output = $(job_name).$(jobid).out
#@  error = $(job_name).$(jobid).err
#@ notification = never
#
#@ queue
#
export MP_EAGER_LIMIT=65536
export MP_SHARED_MEMORY=yes
export MEMORY_AFFINITY=MCM
export MP_TASK_AFFINITY=MCM
poe ./AdvDiff2D_ifiss4.exe --mx=4 --my=4 --nx=2048 --ny=2048
```

```
/usr/local/packages/bin/accounting_g128
```

Submit the job with the command

```
llsubmit t20p16.ll
```

5.5 Notes

- `/usr/local/packages/bin/accounting_g128` prints the cost (in allocation units) of a batch job.
- `MP_EAGER_LIMIT` is an environment variable affecting the buffering of message passing. If the program fails when `MP_EAGER_LIMIT=0` then it may well contain a message passing error. The program did fail (hang in fact), although setting the value to 65536 instead enabled it to complete 'successfully'. There wasn't time to investigate further.
- When more than 16 processors were requested, i.e. more than one LPAR, the program failed also. There was not enough time to find the cause of this. Therefore, no results using more than 16 processors were possible. (Although such jobs would have been very expensive in allocation units anyway, compared to the total available to the project.)

6 Results

6.1 Symmetric Gauss Seidel; diffusion = 0.001

All runs have aggregation damping factor = 1.0 and smoother damping factor = 0.67.

number of iterations

	uniform grid			stretched grid		
grid size	1 proc	4 procs	16 procs	1 proc	4 procs	16 procs
512×512	21	20	19	19	18	18
1024×1024		14	15		15	15
2048×2048			11			8

solution time

	uniform grid			stretched grid		
grid size	1 proc	4 procs	16 procs	1 proc	4 procs	16 procs
512×512	15.8	3.7	1.1	14.6	3.5	1.2
1024×1024		12.2	3.7		12.6	3.7
2048×2048			14.0			10.7

6.2 Jacobi; diffusion = 0.001

All runs have aggregation damping factor = 1.33 and smoother damping factor = 0.67.

number of iterations

	uniform grid			stretched grid		
grid size	1 proc	4 procs	16 procs	1 proc	4 procs	16 procs
512×512	74	57	52	37	33	32
1024×1024		30	34		26	26
2048×2048			20			21

solution time

	uniform grid			stretched grid		
grid size	1 proc	4 procs	16 procs	1 proc	4 procs	16 procs
512×512	18.7	3.3	1.1	10.2	2.2	0.7
1024×1024		10.4	3.7		8.9	3.0
2048×2048			11.4			14.4

7 What Didn't Get Done

7.1 IFISS : CD2

While CD4 has purely Dirichlet boundary conditions, CD2 uses a combination of Neumann and Dirichlet. Having implemented CD2, **Trilinos** ended with an exception (value -1) and the message `'Still to check'` which is from `Galeri_LinearProblem.h`. It seemed to be a problem in dealing with the Neumann boundary condition.

7.2 Hypre

Hypre is a software library of high performance preconditioners and solvers for the solution of large, sparse linear systems of equations on massively parallel computers. **Hypre** is produced by Lawrence Livermore National Labs.

Using **Trilinos** to set up problem CD4 and then the **Hypre** routines to solve it, didn't work straight away. In the event, the problems in getting this running successfully weren't solvable in the time available.