



## **Evaluation of the Matlab Distributed Computing Engine and Distributed Computing Toolbox**

**White Rose Grid e-Science Centre**

Michael Griffiths, University of Sheffield  
Deniz Savas, University of Sheffield

**03-04-2008**

### **Abstract:**

The Matlab distributed computing engine is tested and analysed on the White Rose Grid. Its installation and use are described with reference to the possibility of deployment onto the National Grid Service. Results from performance and scalability tests are presented and demonstrate cases where point to point messaging and collective messaging are most beneficial. For computation using large matrices the transport tests demonstrated that collective messaging scales much better than point to point messaging.

**UK e-Science Technical Report Series**

**Report UKeS-2008-02**

Available from [http://www.nesc.ac.uk/technical\\_papers/UKeS-2008-02.pdf](http://www.nesc.ac.uk/technical_papers/UKeS-2008-02.pdf)

Copyright © 2008 The University of Sheffield. All rights reserved.

## 1 Introduction

The UK's Engineering Task Force (ETF) has evaluated several grid middleware solutions. These products from GridSystems, the Open Middleware Infrastructure Institute (OMII), gLite (from EGEE) and GT4 (from the Globus Alliance) will be deployed and tested by representatives of the ETF in order to evaluate their deployability on the resources within the National Grid Service (NGS) and the wider UK e-Science community.

The main purpose of this evaluation is to examine the strengths and weaknesses of each software product and identify the issues that would need to be considered before deployment in a production environment. By formally identifying a set of criteria, a systematic evaluation will take place across all the software evaluations enabling useful comparisons between different products.

This evaluation report describes work that was undertaken to evaluate the Matlab distributed computing engine (MDCE) and the associated distributed computing toolbox (DCT). The evaluation was undertaken using Matlab v7.3.0.298 (R2006b).

## 2 General Information

The Matlab distributed computing engine and the distributed computing toolbox provide tools for running Matlab operations over a cluster of compute nodes. This results in the speed up of Matlab jobs and can enable more effective utilisation of a compute resource. The basic structure of this component of Matlab is a client application that submits tasks to the Matlab distributed computing engine. The Matlab distributed computing engine is basically a job manager that coordinates the submission of task across a collection of worker nodes. Client applications are developed using the Matlab distributed computing toolbox. Every machine hosting a worker or job manager must run the Matlab distributed computing engine (MDCE) service. This application architecture provides a number of benefits:

- Can be run across a heterogeneous collection of nodes
- MDCE service enables communication between processes on different nodes
- MDCE daemon recovers work and job manager sessions when a host node crashes

The Matlab distributed computing toolbox provides functionality for resource discovery, job creation and job submission. Work is undertaken on a collection of processors that has been reserved by the system scheduler. The toolbox also provides functions for calculating functions across a collection of processors.

### 2.1 Provider

MDCE and the MDCT are developed by Mathworks Ltd., founded 1984. Mathworks has an excellent reputation with a well established development process that has enabled the successful distribution of Matlab and its associated toolboxes to a wide range of users in academia and industry worldwide.

The Matlab distributed computing engine has been available since early 2005. Although growth of the Matlab user community is levelling out it is highly probable that within the next three years it will continue to sustain the existing community.

With the evolving maturity of the Matlab distributed computing engine it is anticipated that there will be growth of new communities of Matlab users utilising the high throughput computing capabilities, distributed computing capabilities and co-ordination of workflows. In particular this may enable users to exploit the new generation of multi-core processors.

## 2.2 Licensing

Matlab uses the FLEX licensing system and provides a range of licenses appropriate for individuals, groups or organisations. The Matlab distributed computing engine introduces a new type of license. The MDCE can be purchased with a so called license pack; in packs that reflect the size of the cluster on which the MDCE will be used. This effectively enables multiple instances of Matlab to be initiated on many worker nodes on a compute cluster.

## 2.3 Supported platforms

The current evaluation has been undertaken on a cluster of 160 AMD 64 bit opteron processors with redhat scientific linux. The system uses the scheduler Sun Grid Engine version 6 and includes a gnu compiled version of openmpi for parallel application development.

Mathworks provide support for the MDCE for a range of schedulers including its own Mathworks job manager and third party schedulers such as PBS, Platform LSF and windows css.

## 2.4 Support

Support provided by Mathworks for the installation of this product was provided by a Mathworks engineer, but the installation was completed using just the documentation provided with the MDCE. Support from the Mathworks engineer was required for testing the operational modes of the MDCE and the MDCT. Matlab have an online support request facility and a well supported user community website known as Matlab Central.

# 3 Systems Management

## 3.1 Documentation for System Manager

Accurate documentation for installation and configuration of the MDCE is provided in pdf format from the Matlab web site. The documentation includes a quick start guide.

## 3.2 Installation

The software is installed on a cluster head node with storage mounted across the full cluster. The MDCE installation manages the creation of required environment variables: for the evaluation discussed here, no administrative intervention was required. The complexity of the installation is governed by the interaction of the MDCE with the system schedulers and its implementation of the MPI libraries. The installation procedure is therefore dependent on the particular cluster configuration. Task submission to a scheduler via the MDCE is achieved by a number of wrapper scripts in the MDCT and it may be necessary for an administrator to configure these. The MDCE/MDCT is designed for use on a high performance compute cluster and

there was no need for firewall reconfiguration. However if the MDCT is accessing a compute cluster from outside a firewall, it may be necessary to open port 22 for access by the secure shell protocol.

The MDCE defines abstracted communications with a range of different scheduling mechanisms. This abstraction is achieved through a collection of functions within the Matlab distributed computing toolbox which are used to call job submit functions for the scheduler. This enables the distributed computing toolbox to be configured on a Matlab desktop external to the cluster: the desktop is then able to submit Matlab tasks directly to the cluster. This currently requires customisation, by the user, of the Matlab job submission wrappers and also the preparation of public key exchange for enabling secure shell access to the remote cluster. This assumes that appropriate firewall permissions for the secure shell protocol have been established.

The MDCE possesses its own integrated job manager for submitting work. However, the DC toolbox provides a generic scheduler interface allowing the user to interact with third party schedulers. Examples are provided for condor, SGE and PBS: we describe below the procedure for configuring the generic interface to use Sun Grid Engine

### 3.2.1 Configuring the SGE scheduler for Running Distributed Jobs

For running distributed jobs using the generic scheduler interface, the following files are needed:

- `sgeSubmitFcn.m` - the submit function, which submits individual SGE jobs to the scheduler using "qsub". The submitted script is always "sgeWrapper.sh".
- `sgeWrapper.sh` - simply launches MATLAB on the cluster, and forwards the correct environment variables needed by the decode function.
- `sgeDecodeFunc.m` - the SGE decode function for distributed jobs - this must be on the MATLAB path of the worker machines. One way to achieve this is to put this file into `$MATLABROOT/toolbox/local`.

These files are normally found in the directory `$MATLABROOT/toolbox/local`.

### 3.2.2 Configuring the SGE scheduler for Running Parallel Jobs

For running parallel jobs, an SGE parallel environment is required. The "matlab" parallel environment is based on the "MPI" example shipped with SGE. To use this parallel environment, the "matlabpe.template" must be used. This template must be customised to match the number of slots available, and to match where the "startmatlabpe.sh" and "stopmatlabpe.sh" scripts are installed on the cluster.

The following stages are needed to create the parallel environment, and then make it runnable on a particular queue:

1. Add the "matlab" parallel environment, using a shell command like:

```
$ qconf -Ap matlabpe.template
```

## 2. Make the "matlab" PE runnable on all queues:

```
$ qconf -mq all.q
```

This will bring up a text editor for you to make changes: search for the line "pe\_list", and add "matlab"

## 3. Ensure you can submit a trivial job to the PE:

```
$ echo "hostname" | qsub -pe matlab 1
```

Check that the job runs correctly using "qstat", and check the output file which will have a default name of "~/STDIN.o####" where "####" is the SGE job number.

Once SGE is correctly configured, the following files are then used:

sgeParallelSubmitFcn.m - the parallel submit function, which submits jobs to the "matlab" PE.

sgeParallelWrapper.sh – the parallel wrapper, which launches SMPD/MPD on all the hosts allocated by SGE then uses mpiexec to launch matlab (SMPD and MPD are the MPICH2 process managers on windows and UNIX respectively)

sgeParallelDecode.m - the parallel decode function - this must be on the MATLAB path of the worker machines. One way to achieve this is to put this file into \$MATLABROOT/toolbox/local

By default, the configuration will use the standard tcp/ip interconnect for the compute cluster: for the evaluation, this was a giga-bit ethernet. Further effort is required to enable the installation to use a specialised high performance interconnect such as infiniband or myrinet. The distributed computing engine will work with any MPI that is binary compatible with MPICH2, and Matlab provides documentation on how to configure the MPI libraries for the MDCE. OpenMPI is not compatible with MPICH2 and so this cannot be used. Also, for myricoms implementation of MPI over myrinet we were unable to compile an MPICH2 compatible library. The work for this evaluation was therefore undertaken using the giga-bit ethernet for the test cluster.

### 3.2.3 Customising the scheduler configuration

Configuring the MDCE with a cluster providing a centralized high performance computing service to a wide community requires careful configuration of the scheduling system. This was particularly important for enabling the use of the interactive parallel mode, called, "pmode".

For Sun Grid Engine the following method using subordinate queues is recommended. A number of nodes are allocated for interactive work. A parmatlab queue, configured using the parallel environment, is set up for these selected nodes. All of the queues on these workers are made subordinate to the parmatlab queue. This configuration means that whenever parallel Matlab work is run on these nodes other work becomes suspended. These suspended jobs are restarted when the interactive work has finished or slots become available. In this way a number of users are able to start an interactive session for using Matlab in "pmode"; this enables efficient use of

the compute cluster without preventing interactive use of Matlab in “pmode”.

### 3.3 Distributed Computing Toolbox Deployment

In order for users to utilise the MDCE they must use the functionality provided by the matlab distributed computing toolbox and follow the documentation for the DCT.

### 3.4 Reliability

Much of the functionality of the MDCE is dependent on the scheduler configuration and its policies. It was found that most users wanted to use the MDCE in interactive parallel mode with the system fully loaded: such sessions were difficult to obtain. Reliability of the product is sensitive to MPI and scheduler configuration across a cluster.

### 3.5 External Services

User provided toolboxes are easily integrated into Matlab and used in conjunction with the distributed computing toolbox.

### 3.6 Scalability

Scalability of the MDCE is limited by the licensing constraints of a particular installation. It is anticipated that the MDCE will easily scale with future upgrades/expansions of a particular compute cluster. The documentation claims that the MDCE will support a mixed platform environment. In principle, it is therefore scalable over a campus grid with appropriate licensing.

## 4 User Experience

### 4.1 Documentation for Users

Overall the user documentation is very good. Although some examples are provided in the online documentation, users were still requesting, for example, codes. Also, users developing parallel applications requested further examples.

### 4.2 Migration between platforms

If code is restricted to Matlab m-scripts or p code then porting the applications is very easy. Porting would be subject to the availability of toolboxes and licenses on the new system.

### 4.3 Usability

System interaction with the MDCE is through the Matlab work space using the Matlab distributed computing toolbox. The usability of the interactive parallel mode for Matlab increased the demand for interactive parallel compute queues on the cluster. The toolbox enabled easy configuration of job ensembles and workflows.

The distributed computing toolbox enables users to manage the submission of task arrays or to submit parallel jobs to the distributed computing engine. An attractive feature of the toolbox is the so-called “pmode” operation in which the user can run an interactive parallel session. This mode is dependent on the availability of resources for parallel interactive tasks on a given compute cluster. “pmode” is invaluable for the development and testing of parallel applications. When working in “pmode” it was found that editing an m-script and re-running it did not make the updated version available to the worker nodes. We suspect that this is a problem caused by the workers not being able to detect the changes to the script and continuing to use the older version residing in the local toolbox cache of the worker. With the version used for the evaluation, 7.3.0.298 (R2006b), it was found that the user had to exit “pmode” and then restart “pmode” in order to test a given application.

For a standalone version of Matlab, using version 7.5.0 (R2007b) it is possible to start “pmode” with up to 4 labs using the local processor alone. This is a particularly important feature, enabling users to test their applications before running them on a compute cluster. The evaluation demonstrated that the best approach for developing a distributed application was by using “pmode” on the user’s local desktop. This reduces the amount of testing a user performs using “pmode” on the cluster. It should be noted that in pmode distributed matrices may be used directly; however the message passing capability can only be used within functions being developed by the user.

The generic scheduler interface provides a useful means for submitting tasks to resources managed by using different scheduling systems. This kind of abstraction could be particularly beneficial for users running resources on a range of systems with differing scheduling systems. One drawback is that the user is unable to specify exact job requirements, such as memory and cpu time required to run a job. Through the generic scheduler interface it is possible to customise the job submission scripts to enable the user to specify such requirements as memory and required cpu time. Table 5 shows the command required to start “pmode”, i.e. interactive Matlab with Sun

Grid Engine. The second row shows the command used to invoke “pmode” for a standalone Matlab distribution. Table 6 shows the main elements of script for starting an array of tasks such as setting the submission functions for a specified compute resource. It also shows how the user may create an array of tasks and submit to a scheduler, particularly the use of the WaitforState function for enabling control of the flow of work between arrays of tasks. Finally, table 7 shows a similar script for submitting a parallel job. The key features are the correct specification of the parallel submit function and the specification of the parallel Matlab code including the input and output arguments for the task. The data returned from each processor in a parallel task is returned in a cell array.

By default the MDCE provides a lot of output generated by its interaction with the scheduler: this is sometimes useful for diagnostic purposes.

On a production service compute cluster, development is limited by the availability of interactive shells for parallel Matlab. The recommended procedure is therefore for users to exploit the use of “pmode” on their local desktop using release 2007b or later: this greatly speeds up the development of parallel applications. Although the distributed computing toolbox provides message passing calls that are similar to MPI, the overall philosophy behind the development of distributed applications within Matlab is matrix oriented parallelism where parallel application development is focused on mechanisms for scattering and gathering matrices over an array of processors. It is noted that very often developers would utilise the parfor mechanism for creating a parallel for loop: this approach can lead to the rapid generation of a parallelised application.

Developing applications using the Matlab DCT is quite different from using parallel development languages such as MPI and PVM. One of the most encouraging features was the ease with which applications can be developed and tested.

#### 4.4 Verification

The MDCE provides status information generated by the scheduling system on the cluster. This may be queried using query operations provided by the distributed computing toolbox.

#### 4.5 Performance of Applications Developed Using the Distributed computing toolbox

A number of tests were performed to investigate the scaling of performance as the number of processors was increased. In the first test we investigated the solution of an ordinary differential equation (ODE) over a range of values. Each processor solves the ODE over a different part of the range. At the start of the run the solution vector is therefore scattered across the processors and at the end the solution is gathered. Such a problem has a potentially low communications overhead. To perform this test, three applications were developed:

- Using point to point messaging functions in the Matlab DCT
- Using collective messaging in the Matlab DCT
- The problem was written as a c program and MPI point to point messaging was used.

For comparison against the C MPI application we ensured that the C program was compiled using MPICH with the gnu compilers and using the same interconnect as the Matlab distributed computing engine. The results are displayed by graphs illustrated in figure 1 and the data tables shown by table 1. The results show that for low levels of parallelism the performance scaling compares favourably with the MPI using C. It may be observed from figure 1 that the best scaling is achieved using point to point messaging.

Figure 2 illustrates the scaling of performance for the analysis of a collection of 20 130MB data files. In this case each processor loads one of the files from the series as a matrix, some analysis is performed and the next file is processed. Figure 2 shows the performance scaling expected when the analysis problem is distributed over a number of processors. The scaling test was performed for file collections on locally mounted storage, NFS mounted storage and fuse mounted storage using an sshfs. For this problem there is minimal communication overhead and this is illustrated by the performance scaling. Here, scaling is not influenced by the level of file I/O. Tabulated data is shown in figure 2. The large peak for fuse-sshfs with 6 processors may be explained, as the fuse tests were made on the network carrying other traffic and this was susceptible to high levels of network traffic during the tests. These tests were run at a time when it was anticipated that there would be a low level of network traffic.

Further tests were undertaken to compare performance scaling when moderately sized (480x480) and large sized matrices (4800x4800) are distributed across an array of processors. Performance scaling was compared for point to point messaging techniques and collective messaging techniques for both Matlab and the C implementation of MPI. Figures 4, 5, 6 and 7 show the scaled time for distributing a matrix: the scaled time is relative to the time with two processors. Figures 4 and 5 show results comparing the performance scaling for distributing and gathering a 480x480 matrix. Figures 6 and 7 show results comparing the performance scaling for distributing and gathering a 4800x4800 matrix. It is also important to note that figures 4 and 6 are results for tests with collective messaging whilst figures 5 and 7 apply to point to point messaging. The corresponding data with the raw timings (in ms) is shown in tables 3 and 4.

The following observations may be made in these results.

- There are two orders of magnitude difference in performance for the C implementation of the tests.
- For all tests the performance scaling displays a jump above 4 processors.
- For Matlab there is a marked increase in the scaled time for the large 4800x4800 matrix when the number of processors is greater than 4 and point to point messaging is used.

The above observations may lead to the following conclusions about messaging using the Matlab distributed computing engine:

- Collective messaging tools have much better performance scaling, and this becomes more important when a network interconnect is used.
- For relatively low levels of parallelism, with boxes housing multi processors with multiple cores, good performance scaling effects are expected. For multi core tests we will expect effects due to some memory bottlenecks.

#### 4.6 Languages and Tool Support

Matlab includes an integral profiler and debugger; however, tools for distributed debugging and profiling are not yet available with 2006b. The 2007b release has additional features such as a parallel profiler and user configuration management and selection tool. This allows users to set and select different configurations for distributed computing. The way in which configurations are stored is different to the method used in release 2006b and an import tool is available on Matlab central file exchange. The import function is called `importDistcompUserConfig`.

### 5 Conclusions

Using the Matlab distributed computing engine provides users with a number of benefits:

- It is easy to develop models that may be run on compute clusters provided by other institutions
- Run multiple instances of parametric models using task arrays
- Generate workflows of executed task arrays
- Develop models that may be distributed over a cluster using tools based on standards such as MPI.

Although it is feasible for users to develop parallel applications using a message passing approach, the interactive parallel mode Matlab is intended to be a more intuitive matrix oriented approach to parallel computing.

Experience with developing applications using the MDCE demonstrated that the best approach is to develop an application using “pmode” on the user’s local desktop. This reduces the amount of testing a user performs using “pmode” on the cluster. It should be noted that in pmode distributed matrices may be used directly; however the message passing capability can only be used within functions being developed by the user.

The performance results demonstrated that for low levels of parallelism the point to point messaging demonstrated better scaling than the collective messaging functions. However, for large matrices the transport tests demonstrated that collective messaging scales much better than point to point messaging.

An interesting application of the distributed computing toolbox is as a desktop client for a running Matlab tasks under the control of the Matlab distributed computing engine. This currently requires public key exchange enabled access to the remote cluster using the secure shell protocol. An exciting benefit of the DCT on a desktop workstation is its ability to tap into the resources of a remote HPC cluster running the MDCE as a service. This can be done with Matlab alone without any need for terminal access or file transfer software.

The MDCE and DCT are currently the focus of a lot of development work by Mathworks, and it is likely that future versions of the MDCE and DCT will look quite different to the release evaluated here. For example, in release 2008a the distributed

computing toolbox is renamed parallel computing toolbox, and there is some function renaming and enhancement to the job creation functions. Release 2008a includes parallel computing support for the optimization toolbox.

## 6 Figures

### Scaling Test for ODE Solver

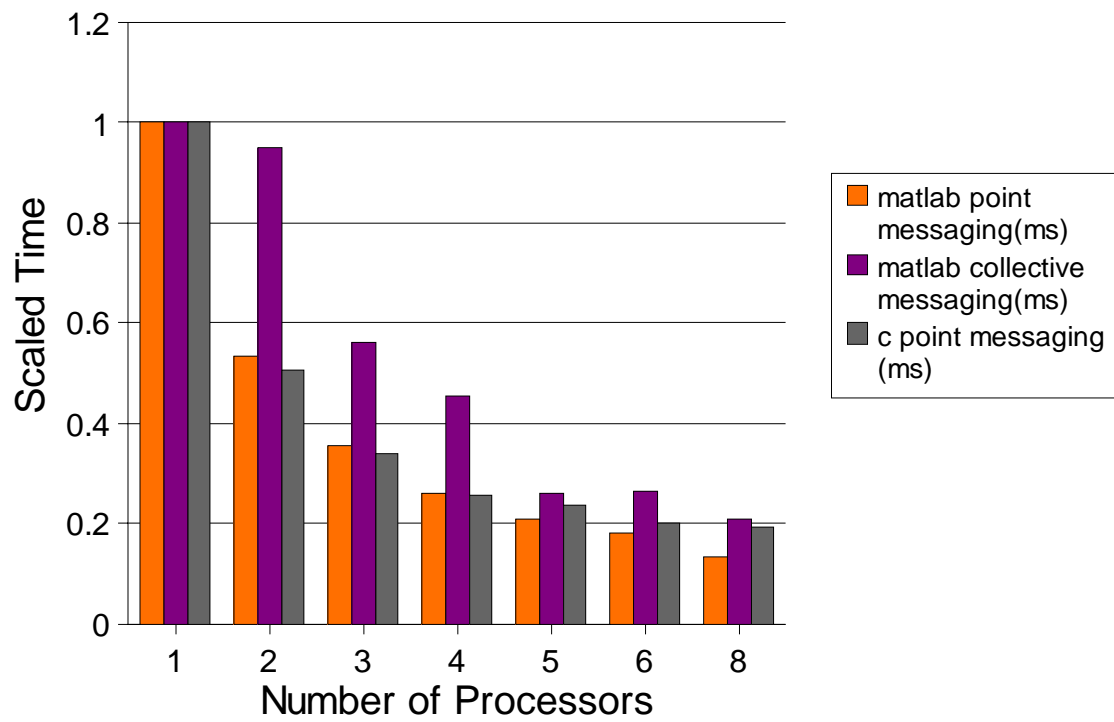


Figure 1. Comparison of performance scaling for an ODE solver using the Matlab distributed computing engine and MPI.

### File Transfer Scaling tests

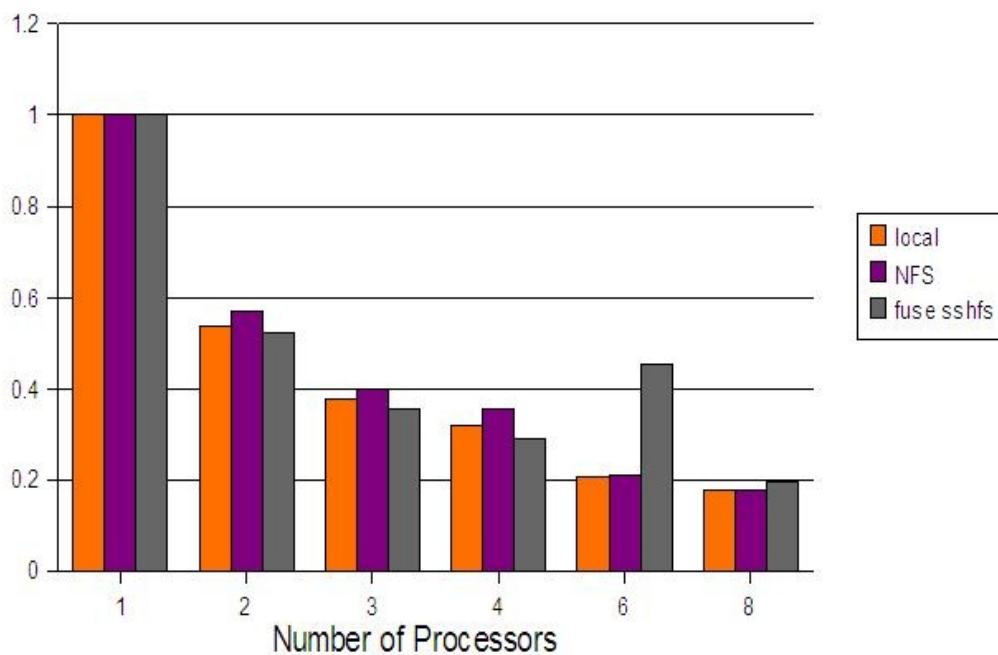


Figure 2. Comparison of scaling performance for distribution tests using a set of 20x130MB files.

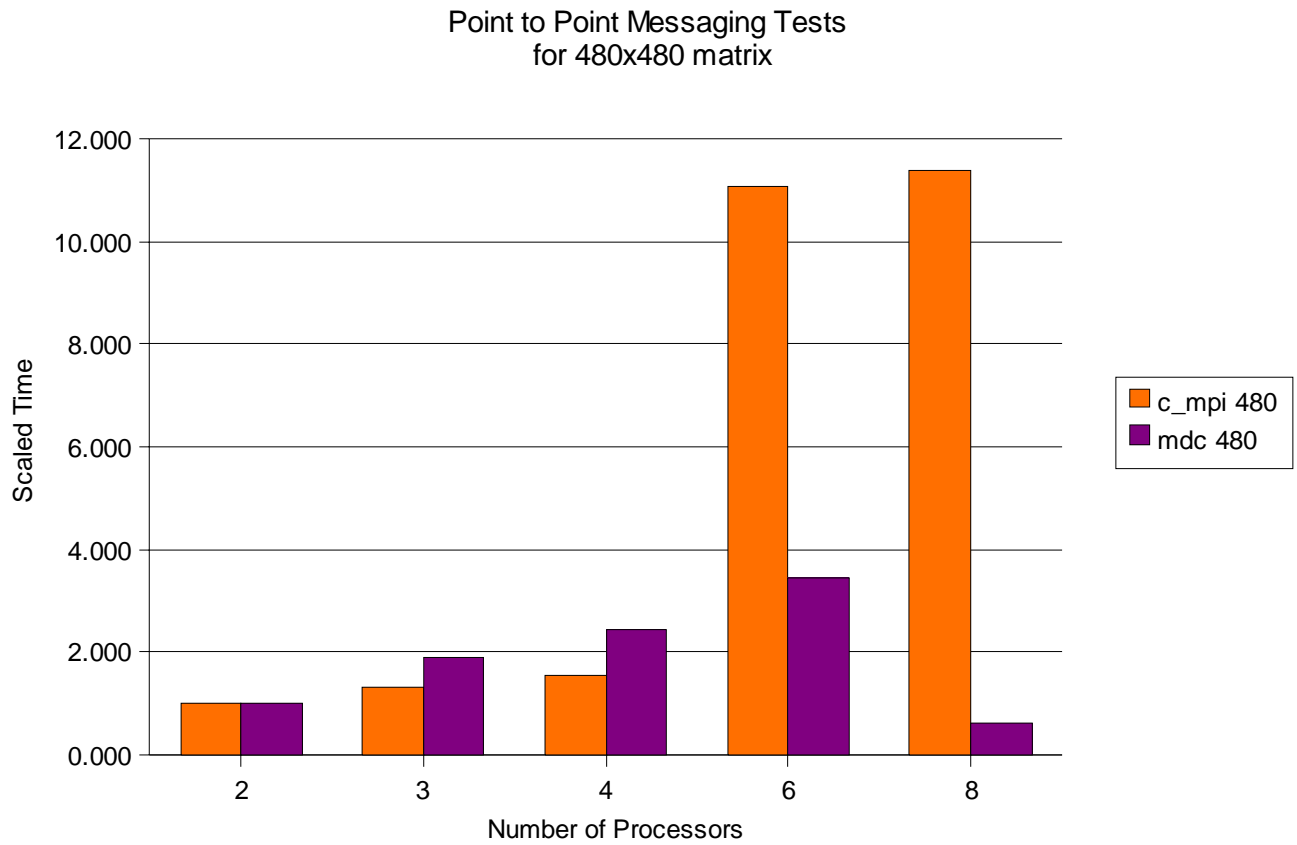


Figure 3. Comparison of scaling for point to point messaging, distributing and gathering a 480x480 matrix over the processors

## Collective Messaging Tests for 480x480 Matrix

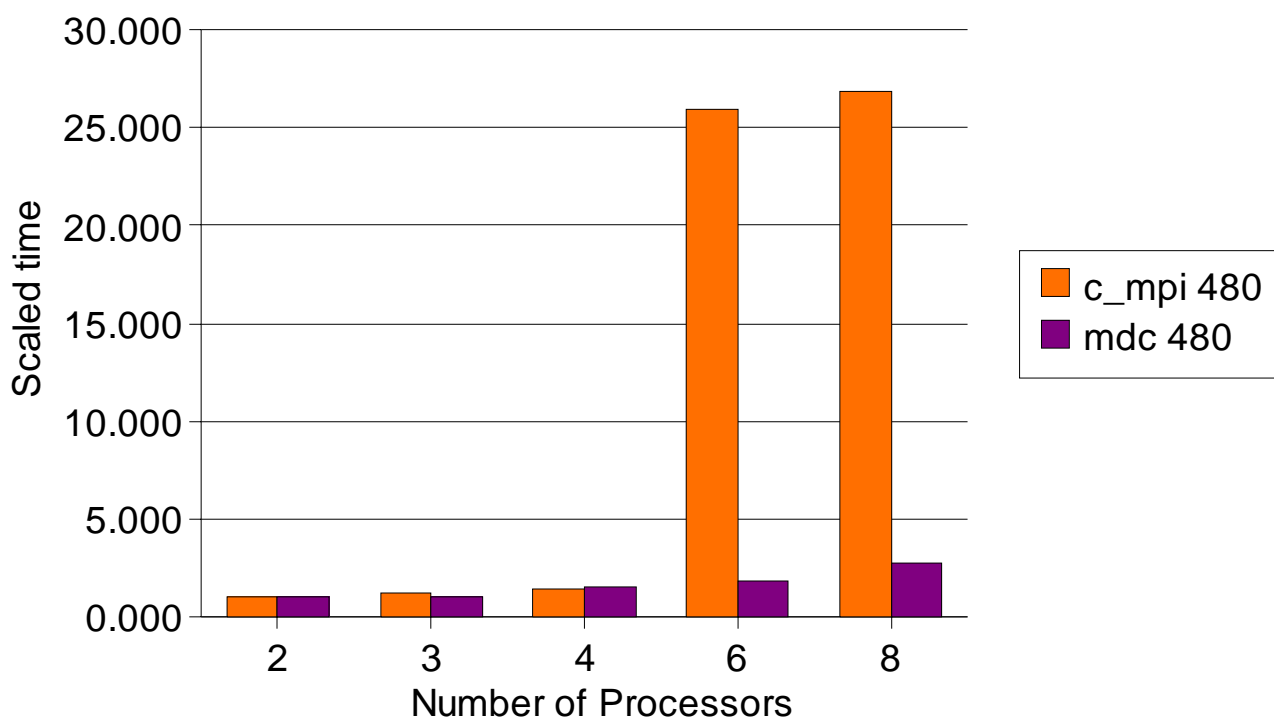


Figure 4. Comparison of scaling for collective messaging, distributing and gathering a 480x480 matrix over the processors

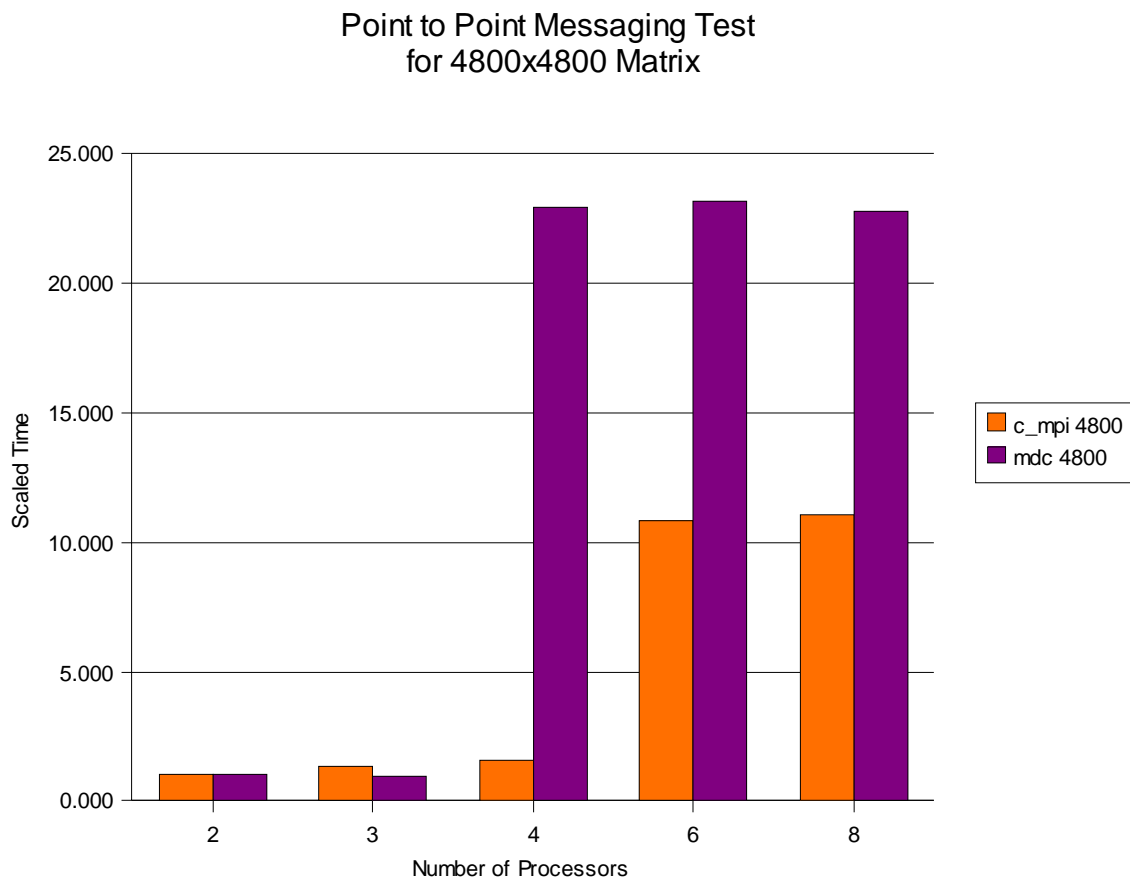


Figure 5. Comparison of scaling for point to point messaging, distributing and gathering a 4800x4800 matrix over the processors

## Collective Messaging Tests for 4800x4800 Matrix

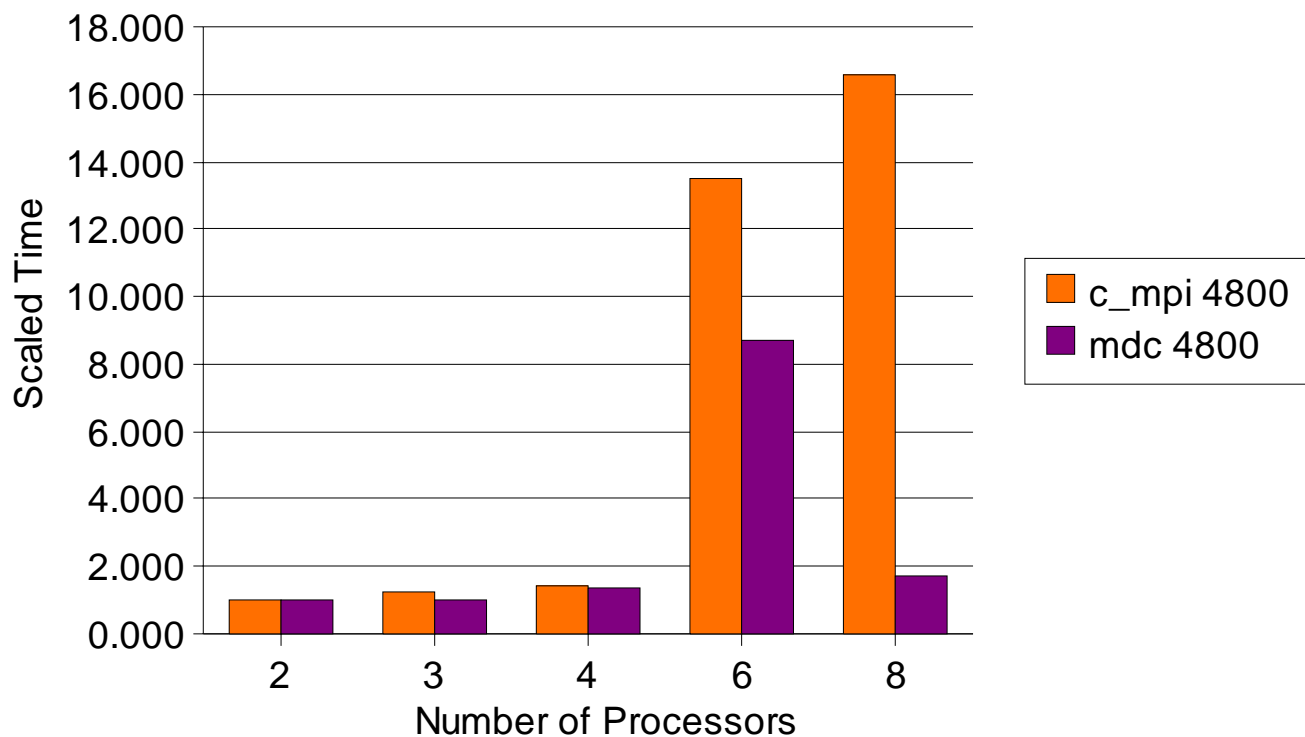


Figure 6. Comparison of scaling for collective messaging, distributing and gathering a 4800x4800 matrix over the processors

## 7 Data Tables

The data tables show show timings obtained with varying numbers of processors.

	matlab point messaging(ms)	matlab collective messaging(ms)	c point messaging (ms)
1	9413.2	10266	350.22
2	5010.2	9753.7	177.9
3	3363.4	5783.7	118.8
4	2447.2	4670.5	89.4
5	1978.1	2690.1	82.7
6	1689.6	2711.9	71
8	1267.8	2147	67.9

Table 1. Results for Comparison of performance scaling for an ODE solver using the Matlab distributed computing engine and MPI.

	local	NFS	fuse sshfs
1	205.24	206.59	214.32
2	110.16	117.72	112.3
3	77.12	82	76.08
4	65.12	73.49	61.78
6	42.07	43.55	97.28
8	36.31	36.79	42

Table 2. Results for Comparison of scaling performance for distribution tests using a set of 20x130MB files.

	mdc 480X480 matrix	c_mpi mpich gnu gm 480X480 matrix	mdc 4800X4800 matrix	c_mpi mpich gnu gm 4800X4800 matrix
1	12.5	0	1140	0
2	158	3.15	4378.6	283.62
3	299.3	3.78	4073.9	378.36
4	386.9	3.9	100356.4	419.38
6	546.6	6.13	101372.5	584.57
8	94.2	6.63	99620.9	618.92

Table 3. Time in ms for using point topoint messaging to distribute a matrix over an array of processors.

	mdc 480X480 matrix	c_mpi mpich gnu gm 480X480 matrix	mdc 4800X4800 matrix	c_mpi mpich gnu gm 4800X4800 matrix
1	25.4	4.27	1429.3	533
2	374.8	7.52	8787	822
3	391	10.1	8946.1	1098
4	552.6	11.48	12000.1	1169.6
6	696.2	21.48	76481.7	2401
8	1019.7	34.17	15041.2	4198

Table 4. . Time in ms for using collective messaging to distribute a matrix over an array of processors.

```
“pmode” start sge 4
```

```
“pmode” start local 4
```

Table 5 Starting interactive Matlab with Sun Grid Engine and on a local desktop

```
resource=findResource('scheduler','type','generic');
set(resource, 'configuration', 'sge');

set(resource, 'SubmitFcn', @sgeSubmitFcn);
set(resource,... 'ClusterMatlabRoot','/usr/local/packages/matlab_mdcdem_r06b');

job=createJob(resource);

createTask(job, @rand, 1, {3,3});
createTask(job, @rand, 1, {4,4});
createTask(job, @rand, 1, {5,5});
createTask(job, @rand, 1, {6,6});

%submit all the tasks
submit(job);

%wait for job to complete before continuing
waitForState(job);

results=getAllOutputArguments(job)
```

Table 6 Elements of script for starting an array of tasks

```
%simple demo script for matlab dce
%test by Mike Griffiths 22nd January 2007

resource=findResource('scheduler','configuration','sge');
set(resource, 'configuration', 'sge');

set(resource, 'SubmitFcn', @sgeSubmitFcn);
set(resource, 'ParallelSubmitFcn', @sgeParallelSubmitFcn);
set(resource, 'ClusterMatlabRoot',... '/usr/local/packages/matlab_mdcdem_r06b');

j=resource.createParallelJob...
('MaximumNumberOfWorkers',4,'Configuration','generic');
createTask(j,'mparfunc',1,{1});
submit(j);

waitForState(j);
parout = getAllOutputArguments(j);
```

Table 7. Submitting a parallel job