Performance Studies with COMSOL Multiphysics via Scripting and Batch Processing

Noemi Petra and Matthias K. Gobbert

Department of Mathematics and Statistics, University of Maryland, Baltimore County

{znoemi1,gobbert}@math.umbc.edu

Abstract

COMSOL Multiphysics is an extremely powerful and versatile finite element package for the solution of partial differential equations. While the graphical user interface (GUI) offers a friendly environment for solving small problems, for the solution of a more demanding problem with correspondingly larger memory requirements and longer run times, it is often desirable to explore the script-based modeling capabilities, as well as the benefits of running COMSOL in parallel. This work gives step-by-step instructions on how to use m-files in conjunction with MATLAB as scripting tool and COMSOL's own binary format for batch processing under Linux. We also investigate how to run COMSOL in parallel and report on the shared-memory parallel performance of COMSOL, that is, using all cores available on a compute node. The results show that the speedup is not in proportion to the number of cores used for any of the linear solvers. The results also show that the PARDISO linear solver outperforms all other solvers for our particular test problem.

1 Introduction

COMSOL Multiphysics is an excellent, state-of-the-art software for the solution of many types of partial differential equations (PDEs), both stationary and time-dependent, by numerical techniques based on the finite element method for the spatial discretization. Some of its key features include its CAD capabilities for the creation of complicated 2-D or 3-D domains and its sophisticated meshing capabilities. These highly visible features are accessible through the Java-based graphical user interface (GUI). Beginning users will start to learn the software by using this GUI, and this is suitable for the immediate solution of a problem. However, for the solution of larger problems with correspondingly larger memory requirements and longer run times, other features of COMSOL become crucial. In these cases, it is often useful to run COMSOL on a different machine than the desktop computer, such as a remote Linux machine with larger memory or faster processors like hpc.rs in the UMBC High Performance Computing Facility (HPCF; `www.umbc.edu/hpcf`). Moreover, to ensure reproducibility of one's research results, or to perform parameter studies, the use of the GUI is not ideal. These problems are addressed by COMSOL's capabilities for scripting and batch processing. These features are not easy to use, and in fact, it is even quite confusing what features all exist and what they mean; this is a result of the extreme versatility and power of COMSOL. This report will attempt to structure and explain the options for the use of COMSOL that exist and show them at work, including specifically how to use m-files in conjunction with MATLAB as scripting tool and COMSOL's own binary format for batch processing. Furthermore we also test and demonstrate the parallelism capabilities for various linear solvers available in COMSOL.

As model problem, we consider the classical elliptic test problem, given by the Poisson equation with homogeneous Dirichlet boundary conditions

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega, \end{aligned} \tag{1.1}$$

on the unit square where $\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2$. Here, $\partial\Omega$ denotes the boundary of the domain Ω and the Laplace operator Δ is defined as $\Delta u = u_{xx} + u_{yy}$. To test the numerical method, we consider the elliptic problem (1.1) with right-hand side function

$$f(x,y) = (-2\pi^2) \left(\cos(2\pi x)\, \sin^2(\pi y) + \sin^2(\pi x)\, \cos(2\pi y)\right), \tag{1.2}$$

for which the problem admits the true solution

$$u(x,y) = \sin^2(\pi x)\, \sin^2(\pi y). \tag{1.3}$$

After solving this boundary value problem via COMSOL, the attention typically shifts to the goal of gaining confidence in the correctness and accuracy of the computed solution. We make use of the availability of the true solution and compute the error between the FEM solution u_h and the true solution in the $L^2(\Omega)$-norm, which is defined by

$$\|u - u_h\|_{L^2(\Omega)} = \left(\iint_\Omega (u - u_h)^2 \, dx\, dy \right)^{1/2} \tag{1.4}$$

for a mesh spacing h which denotes the maximum side length of the elements in the mesh used to discretize the domain. By applying these results repeatedly for progressively smaller mesh spacings obtained by regularly refining a coarse initial mesh several times, one can assess if the sequence of solutions is converging as expected based on the theory and understand the quality of the solution [2].

Our primary goals in this paper are to show how to run COMSOL 3.5a on hpc and to report on the shared-memory parallel performance of various linear solvers in COMSOL. In [4], the authors reported on performance studies for multithreading in MATLAB. Their results demonstrate that the use of more than one thread is often not very beneficial for MATLAB code. This outcome motivated us to be cautious in testing the parallel performance of COMSOL. To avoid any influence of MATLAB in our performance report, we run COMSOL in two ways: (i) COMSOL with MATLAB and (ii) COMSOL in batch mode (standalone COMSOL). Our results suggest that the speedup is not in proportion to the number of cores used, independent of the way COMSOL is run; this independence is expected as MATLAB is only used as scripting tool and both ways to run COMSOL call the same solver functions. Furthermore, we present all the necessary ingredients one needs to follow in order to be able to reproduce or repeat our tests and detailed usage instructions on hpc.

The numerical studies in this report were performed on the distributed-memory Linux cluster hpc in the UMBC High Performance Computing Facility (HPCF; `www.umbc.edu/hpcf`), which has 33 compute nodes, each with two dual-core AMD Opteron 2.66 GHz processor (1 MB cache per core) and 13 GB memory, for a total of up to four parallel processes to be run simultaneously per node. The files needed to reproduce our results are posted along with the PDF file of this report on the HPCF web page under Technical Reports.

The outline of this paper is as follows. Section 2 explores the various ways of running COMSOL on a desktop computer. It includes not only step-by step instructions on how to create and solve the test problem but also has the purpose of preparing the reader to run COMSOL with or without MATLAB properly in background, which is a necessary condition before running an application on a compute node. For clarity and reproducibility we use the head node of hpc as 'desktop computer' here. If you are familiar with COMSOL and its scripting capabilities, you could skip this section and go straight to Section 3, which shows how to run COMSOL with MATLAB or COMSOL in batch mode properly on hpc, that is, by using a compute node. This section also goes through the parallel capabilities of COMSOL of multi-threading on a single node and of using several nodes, and it presents also a parallel performance study by the number of cores on a compute node. Finally, our conclusions are summarized in Section 4, which also gives a list of future work plans.

2 Running COMSOL

As we already mentioned, the starting point in learning how to use COMSOL is the GUI which is suitable for the immediate solution of a problem. However, to ensure reproducibility of one's research results, or to perform parameter studies, the use of the GUI is not ideal. Moreover, for the solution of larger problems with correspondingly larger memory requirements and longer run times one needs to make use of other features of COMSOL such as scripting and batch processing and try to use a machine, preferably remotely, with larger memory or faster processors like hpc.rs in the UMBC High Performance Computing Facility (HPCF; `www.umbc.edu/hpcf`). In this section we explain in detail the options for the use of COMSOL that exist and show them at work. We use here the head node of hpc as the 'desktop computer' to ensure that the reader (with access to hpc) can reproduce the results exactly; usually you would run COMSOL as documented in this section on your actual desktop, and the information here should apply to other Linux computers without change.

We start by mentioning that help on running COMSOL under Linux is available by saying `comsol -h` for the general help and by `comsol -h matlab` for help on a particular application mode, in this example on MATLAB. More information can be found in the section "*Running COMSOL*" in the COMSOL Installation and Operations Guide, Version 3.5a.

2.1 COMSOL Multiphysics (the Graphical User Interface)

This section explains how to use the graphical user interface (GUI) of COMSOL Multiphysics to create an m-file `poisson2d.m` that solves the PDE under consideration with one regular mesh refinement. We intend to use this refinement later to compare the solution on both meshes to check the quality of the numerical solution. Whether graphical visualization or analysis of the errors is intended, it is desired to develop a method that not only is efficient but also allows reproducibility of results. Therefore, our goal is to solve the problem once in the GUI, save the solution process as an m-file for reproducibility, and for editing. A detailed description of how to create a MATLAB script file is given in [2], which we repeat here for clarity.

To solve the model problem (1.1)–(1.2), start the GUI of COMSOL Multiphysics, by typing `comsol` at the Linux prompt, which is shorthand for `comsol multiphysics`. In the Model Navigator for the problem dimension select 2D, then select COMSOL Multiphysics → PDE Modes → PDE, Coefficient Form → Stationary analysis. In the draw mode of the GUI, draw the desired domain $\Omega = (0,1) \times (0,1)$. To set the source term to (1.2), go to the Physics menu, choose Subdomain Settings and enter `(-2*pi^2)*(cos(2*pi*x)*sin(pi*y)^2+sin(pi*x)^2*cos(2*pi*y))` in the field for the Source term f from (1.2); notice that there must not be any spaces in this expression. For our model problem (1.1)–(1.2), all other PDE and boundary coefficients are at their default values.

To ensure that linear Lagrange elements are used, select Physics → Subdomain Settings, highlight the subdomain and then select the Element tab and here select the Lagrange linear elements. Mesh (or re-mesh if you already have a mesh) the domain with a coarse initial mesh, so that we can use several refinement levels later, as follows: Under the Mesh menu select the Free Mesh Parameters, then choose the Predefined mesh size as Extremely coarse. When we re-meshed for the unit square $\Omega = (0,1) \times (0,1)$ in this way, we obtained a mesh with 26 triangular elements. Select Mesh → Refine Mesh once at this point, which gave us 104 triangular elements; notice that regular mesh refinement is the default in 2-D in COMSOL, hence the number of elements quadrupled in the refinement. Using the mesh refinement adds the `meshrefine` command to the m-file that we will export later. You can check the mesh statistics by going to Mesh → Mesh Statistics which confirms that we have 65 degrees of freedom (DOF) as well as 65 mesh points, thus confirming the

linear Lagrange elements, which have one DOF per node. Notice that this is the value for the DOF in Table 1 for refinement level $r = 1$.

Now, have COMSOL solve the problem by selecting the solution icon (the = sign) or by selecting Solve → Solve Problem. This also causes COMSOL to display the solution. The solution and error between the FEM solution and the true solution are shown in Figure 1, using default settings for the 3-D view in both cases. To produce the error plot, go to Postprocessing → Plot Parameters; in the Surface tab, enter as Expression for both Surface Data and Height Data the difference between the FEM solution `u` and the true solution, that is, enter `u-sin(pi*x)^2*sin(pi*y)^2`. To compute the L^2-norm of the error between the FEM solution and the true solution given by (1.4), have COMSOL first compute the square of the norm by going to the Postprocessing → Subdomain Integration, where you enter the expression `(u-sin(pi*x)^2*sin(pi*y)^2)^2`, which is the square of the error. The value of integral is given in the report window located at the bottom of the GUI, which in our case is 7.0201e-4. To obtain the L^2-norm error, take the square root of this value, which should give you $\|u - u_h\|_{L^2(\Omega)} = 2.6496\text{e-}2$.

At this point, create the m-file by saving the entire interactive session in the GUI up to this point using File→ Save As. After selecting Model m-file in the Files of Type field, navigate to a desired directory and provide the File Name `poisson2d.m`. To use the MATLAB scripting features and actually have access to options like Save as m-file, you must have COMSOL installed with the MATLAB interface functions enabled during installation; otherwise, your selection under Files of Type will not include Model m-file. A detailed step-by-step instruction on how to use the COMSOL Multiphysics with its scripting capabilities is given in the next subsection. You can also check the COMSOL manual, COMSOL Multiphysics MATLAB Interface Guide, Version 3.5a, for a complete description on how to model using COMSOL Multiphysics in conjunction with MATLAB and also the section "*Running COMSOL Multiphysics with MATLAB*," available in the COMSOL Installation and Operations Guide, which guides you through the first steps to run COMSOL with MATLAB. We wish to emphasize that all instructions in this report apply for COMSOL 3.5a run under Linux. It is worth also mentioning at this point that the MATLAB version we are testing COMSOL with is R2008b.

The m-file `poisson2d.m` is posted along with the PDF file of this report on the HPCF web page under Technical Reports.

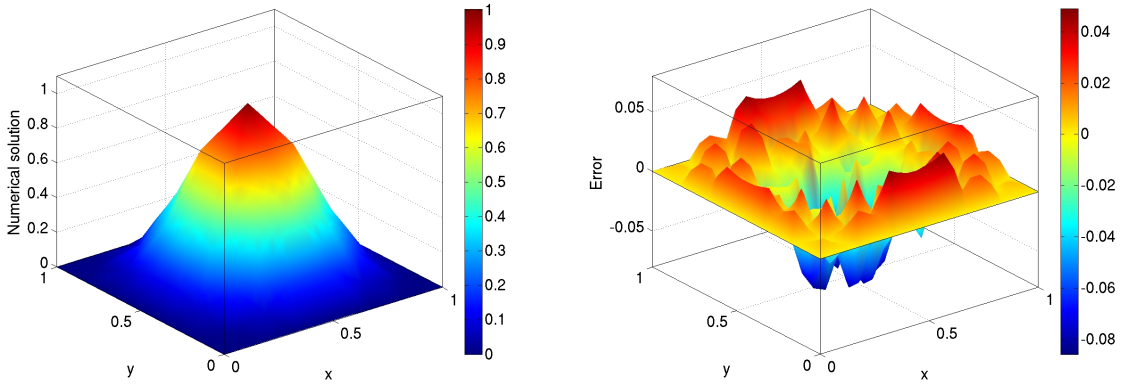

Figure 1: Numerical solution (left) and the error (right) for (1.1)–(1.2).

2.2 COMSOL with MATLAB Interactively with Plots

We now modify the script file that was obtained in the previous subsection to obtain an m-file that solves the problem for a desired refinement level $r = 0, 1, ...$ At the same time, we add a few more capabilities to our function to output desired quantities which will assure us that the code works properly. These include the wall clock time for the solution process, the L^2-norm of the true error, and a plot of the solution printed to a file in jpeg format.

To start MATLAB with paths to COMSOL functions type

```
comsol -verbose matlab path -ml -nodesktop
```

or append `multiphysics` to also start the COMSOL Multiphysics GUI. Here, and in what follows, the COMSOL option `-verbose` is optional and shown for demonstration and debugging purposes. Now, edit the `poisson2d.m` from the previous subsection as `poisson2d_graphics.m` and change the content of the file as follows:

- insert `nrefines = 1` to the beginning of the `poisson2d_graphics.m` file;
- search for the call to `meshrefine` and enclose it in a for-loop that counts nrefines many refinement levels;
- insert the `tic` and `toc` functions before and after the solve command, respectively, to measure the elapsed solution time;
- insert `error_norm_L2=sqrt(I1)` right after the `postint` command;
- insert `print -djpeg100 poisson2d.jpg` after the `postplot` command to print plot to file.

Now you can run this m-file from the MATLAB prompt or use File → Open from the COMSOL GUI; notice that you will only have the option for an m-file in the Files of Type list here, if you have started COMSOL with MATLAB. Another option is to run COMSOL with MATLAB in the background, but still with an open `ssh` and X11 connection (!), by saying

```
comsol -verbose matlab path -ml -nodesktop < poisson2d_graphics.m
    >& poisson2d_graphics.log &
```

at the Linux prompt.

The m-file `poisson2d_graphics.m` is posted along with the PDF file of this report.

2.3 Running COMSOL with MATLAB in background

In this section we focus on running COMSOL without the GUI interface, in conjunction with MATLAB. In particular, we give specific steps one needs to follow in order to prepare the code for a background running. You can now start COMSOL with MATLAB and run the script that was just saved to confirm that you get exactly the same result as from the GUI.

Since graphics capabilities are not available for background runs, we need to suppress all screen output from both COMSOL and MATLAB. Hence, edit the m-file `poisson2d_graphics.m` from the previous subsection as `poisson2d_matlab.m` and apply the following modifications:

- insert `flreport('off')` at the beginning of the m-file to suppress the progress report that reports on the progress of the solver on the screen by default;
- then delete (or comment out) the unneeded commands at the end of the m-file, namely the line fem0=fem and all plot commands such as `postplot`;

- instead of the plots, insert `flsave('poisson2d_matlab.mph',fem)` to the end of the function which saves the `fem` struct.

Here, the final bullet saves the final result in the `fem` structure to a file in COMSOL's binary format called `poisson2d_matlab.mph` for future post-processing.

For testing purposes, we run this COMSOL with MATLAB script in the background on the head node first, where we can test that the m-file `poisson2d_matlab.m` works properly. This is a debugging step and just intended to avoid creating a hung job on a compute node, where it is more difficult to kill for a user.

To run the COMSOL MATLAB script in background on the head node, type

```
nohup comsol matlab path -ml -nodesktop -ml -nodisplay
    < poisson2d_matlab.m >& poisson2d_matlab.log &
```

To use the solution from the file `poisson2d_matlab.mph`, you have two options: (i) If using the COMSOL GUI, load the file by File → Open and proceed with interactive post-processing using the Postprocessing pull-down menu. (ii) If using COMSOL with MATLAB interactively (see above), load the file with the command `flload('poisson2d_matlab.mph')`, then proceed with the plot commands seen in `poisson2d_graphics.m` at the MATLAB prompt. In both cases, the solution is not recomputed but just loaded; the idea is that you would do this on your desktop computer, where you have efficient access to the screen.

The m-file `poisson2d_matlab.m` is posted along with the PDF file of this report.

2.4 Convergence Study for the Model Problem

This is a good moment to check that the COMSOL script we developed is working properly and that the results are correct. For this purpose, we compute numerical estimates that can confirm if the finite element method performs as predicted by the theory which will validate our script.

In order to quantify the correctness of the numerical solution we analyze the error that incurs against the true solution u of the PDE (1.1). This error can be quantified by bounding the norm of the error $\|u - u_h\|_{L^2}$ in terms of the mesh spacing h of the finite element mesh. Such estimates have the form

$$\|u - u_h\|_{L^2} \leq C\, h^q, \quad \text{as } h \to 0, \tag{2.1}$$

where C is a problem-dependent constant independent of h, and q indicates the order of convergence of the FEM, as the mesh spacing h decreases. We see from this form of the error estimate that we need $q > 0$ for convergence as $h \to 0$. For the case of linear (degree $p = 1$) Lagrange elements, we have the well-known a priori bound for quadratic convergence, that is, $q = 2$, given by

$$\|u - u_h\|_{L^2} \leq C\, h^2, \quad \text{as } h \to 0, \tag{2.2}$$

where C is independent of h; see , e.g., [1, Section II.7]. The theory suggests that for the test problem with linear Lagrange elements, we should obtain quadratic convergence. Therefore, in order to confirm that the problem and method behave correctly we compute the convergence order as shown in [2]. Let us consider a sequence of FEM solutions u_h on meshes with progressively smaller h, that is, starting from some initial mesh, we will refine it regularly repeatedly, which subdivides every triangle into four triangles. If h measures the maximum side length of all triangles, this procedure halves the value of h in each refinement. Then assuming that $\|u - u_h\|_{L^2} = Ch^2$, the error for the next coarser mesh with mesh spacing $2h$ is $\|u - u_{2h}\|_{L^2} = C(2h)^2 = 4Ch^2$. If r denotes the number

Table 1: Convergence study listing the mesh refinement level r, the number of elements N in the mesh, the number of degrees of freedom (DOF), the norm of the finite element error $E_r = \|u - u_h\|_{L^2}$, the ratio $R_r = E_{r-1}/E_r$, the solution time in seconds, and the observed memory usage in MB.

r	N	DOF	E_r	R_r	time (s)	memory (MB)
0	26	20	1.0762e-01		< 1	1,713
1	104	65	2.6496e-02	4.06	< 1	1,678
2	416	233	6.7034e-03	3.95	< 1	1,699
3	1,664	881	1.6823e-03	3.98	< 1	1,739
4	6,656	3,425	4.2103e-04	4.00	< 1	1,739
5	26,624	13,505	1.0529e-04	4.00	1.36	1,736
6	106,496	53,633	2.6324e-05	4.00	1.52	1,684
7	425,984	213,761	6.5811e-06	4.00	6.78	1,929
8	1,703,936	853,505	1.6453e-06	4.00	33.20	2,854
9	6,815,744	3,410,945	4.1133e-07	4.00	170.02	5,754

of refinement levels from the initial mesh and $E_r := \|u - u_h\|_{L^2}$, then $E_{r-1} = \|u - u_{2h}\|_{L^2}$ and we can write $R_r = E_{r-1}/E_r$ for their ratio. Thus, this ratio should approach 4 for the quadratic convergence of linear Lagrange elements.

Table 1 lists the number of degrees of freedom (DOF) along with the quantities E_r, R_r, elapsed time in seconds, and the observed memory usage for the model problem. The results show that the norms of the finite element errors decrease by a factor of 4 each time the mesh is refined by a factor 2. This confirms that the finite element method is quadratically convergent, as predicted by theory. We remind the reader that to obtain results such as in Table 1 for any PDE, for which the true solution u is not known, one classical technique is to use a numerical solution computed on a finer mesh as reference solution [2]. Also, for a numerical demonstration of the finite element convergence for higher order Lagrange elements in COMSOL Multiphysics; see [3].

2.5 COMSOL batch processing

The COMSOL batch mode can be used to solve models or to run parametric sweeps (see also "*The COMSOL Command*" on page 54 of the COMSOL Installation and Operations Guide, Version 3.5a for a description of this running mode). To run COMSOL Multiphysics in batch mode first you need to prepare an input file which is saved from the GUI with all problem information, but only with solution parameters without actually solving the problem. More specifically, one needs to go through the steps we described in Section 2.1, but without solving the problem, then use File → Save as to save the session as a COMSOL mph-file. If saved as a file by the name `poisson2d_infile.mph`, you can then run COMSOL in batch mode by saying at the Linux prompt

```
comsol -verbose batch -input poisson2d_infile.mph -output poisson2d_output.mph
```

This command starts COMSOL batch, solves the model with the given input file name using the solver settings in the model, and stores the solution in the `poisson2d_output.mph`.

The mph-file `poisson2d_infile.mph` is posted along with the PDF file of this report.

3 Computing with COMSOL on hpc

The first two subsections below contain usage instructions for how to run COMSOL properly on hpc. That means that you would not use the head node but have the job scheduler reserve one of the compute nodes for you. For COMSOL, this is only possible for jobs that do not produce any screen output of any kind. It can be done both for COMSOL with MATLAB as explained in Section 2.3 as well as for COMSOL in batch mode as explained in Section 2.5. Section 2.3 explains specifically how to ensure that no screen output is generated; notice that COMSOL shows a log window on the screen by default, but also that needs to be turned off, in addition to avoiding graphics output. Please see these subsections above for more details and instructions to test your setup on the head node before proceeding.

Parallel computing offers opportunities to decrease run times as well as to decrease memory usage per process by spreading the problem over the parallel processes. However, at this point we do not have a clear understanding of how COMSOL behaves in parallel. Although the documentation suggests that nearly all direct and iterative solvers as well as smoothers are parallelized, in this report we wish to test this feature and show it at work. In particular, we study the parallel performance of the linear solvers listed in Table 2 for COMSOL in batch mode and with MATLAB.

Notice that as we go to finer meshes, see Table 1, the run times and the memory usage become a big concern. To solve real problems we need several refinements in order to obtain a desired accuracy, in which case the number of degrees of freedom increases drastically hence the computations become very demanding in terms of memory usage and run time. Then, it is essential to run COMSOL on a computer like hpc with larger memory and/or faster processors than your desktop computer might have. After giving specific instructions for running COMSOL with MATLAB and in batch mode on a compute node on hpc, we run COMSOL in parallel in shared-memory mode with and without MATLAB and compare the run time for the two modes.

Table 2: Linear solvers and preconditioners tested for parallel performance.

Linear System Solver	Preconditioner
Direct Method	
UMFPACK	N/A
SPOOLES	N/A
PARDISO	N/A
Iterative Method	
CG	Incomplete Cholesky (IChol)

3.1 COMSOL with MATLAB

In order to submit a `comsol matlab` job to a compute node on hpc one first needs to prepare a submission script for the job scheduler, such as the `qsub-COMSOL_with_MATLAB` script that we list below and that can be downloaded from the hpc web page. The default qsub script includes the lines

```
#!/bin/bash
#PBS -N 'COMSOLwMAT'
#PBS -o 'qsub-poisson2d_matlab.log'
#PBS -j oe
#PBS -W umask=007
#PBS -q low_priority
#PBS -l nodes=1:ppn=1
cd $PBS_O_WORKDIR
comsol matlab path -ml -nodesktop -ml -nodisplay < poisson2d_matlab.m
```

The line `#!/bin/bash` is always the first line in the the script which tells Linux to use the shell `/bin/bash` to execute the script. The second line `-N` creates a name for the job which shows up in the output of qstat; see below. The option `-o` sends the output from stdout to the file `qsub-poisson2d_matlab.log`. With `-j oe`, the output from stderr is joined to the stdout file. The line `-W umask=007` sets the permissions of new files properly; without this line, only the user who ran the job would have any permissions on these files. The `-q low_priority` specifies which queue to submit your job to; the queue `low_priority` is the default on on hpc. Later in the script, `$PBS_O_WORKDIR` is used to change the current working directory to the directory from which you ran qsub. Finally, the last line of the qsub script

```
comsol matlab path -ml -nodesktop -ml -nodisplay < poisson2d_matlab.m
```

runs COMSOL with MATLAB and re-directs the standard input from `poisson2d_matlab.m`. The line `-l nodes=1:ppn=1` requests one computational core on a compute node for the job, which is the default for both COMSOL and MATLAB on hpc and appropriate for single-threaded runs with both applications. This default setup is justified by the results observed in this report, where we observe that it is most effective for throughput of jobs to use only one core per node. If a job is expected to need more than the 3 GB of memory available per core or if tests for a particular job demonstrate that it is advantageous to use more than one core, then the user needs to change the line with the `ppn` value to `-l nodes=1:ppn=2` or `-l nodes=1:ppn=4` to reserve 2 or 4 cores, respectively.

Then, to run the `poisson2d_matlab.m` script we prepared in Section 2.3 on a compute node type

```
qsub qsub-COMSOL_with_MATLAB
```

After submitting, you can use `qstat` or `qstat -n` to see if your job is running; see the HPCF web page for more information. After the job has completed, the `qsub` script produces the same `poisson2d_matlab.log` file that resulted as an output file from the background running on the head node.

The scheduler submission script `qsub-COMSOL_with_MATLAB` is posted along with the PDF file of this report.

3.2 COMSOL in Batch Mode

Similarly to the previous subsection, to run COMSOL in batch mode on a compute node, one needs to prepare first the scheduler submission script. For your convenience the `qsub-COMSOL` script for our tests can be downloaded from the same location on the hpc web page. This submission script is very similar to the one we created earlier. However, there are a few differences. Firstly, the second line that creates a name for the COMSOL job which shows up in the output of qstat was replaced by `#PBS -N 'qsub-COMSOL'`. Also, the name of the output file was changed to a more suggestive one for this particular case, i.e., `qsub-poisson2d_batch.log`. Finally the last line was replaced by the command

```
comsol batch -input poisson2d_infile.mph -output poisson2d_outfile.mph
```

which is the same as the one given in Section 2.5.

After you download the submission script you can run COMSOL in batch mode on a compute node by typing

```
qsub qsub-COMSOL
```

As before, when the job is finished, the `qsub` script produces a `qsub-poisson2d_batch.log` which will contain the output from stdout and stderr. This log file contains the same report that resulted from the batch mode running on the head node.

The scheduler submission script `qsub-COMSOL` is posted along with the PDF file of this report.

3.3 Multi-Threading in COMSOL

The parallel operation mode based on the shared-memory parallelism is performed on the available cores on one node. On Linux, COMSOL is single-threaded by default. To run COMSOL multi-threaded, use

```
comsol -np Np
```

with the number of threads `Np` greater than 1. More specifically, if you wish to run COMSOL Multiphysics with two computational threads, for example, start COMSOL with the option

```
comsol -np 2
```

which is short for `comsol -np 2 multiphysics`. See `comsol -h` for the options available to the `comsol` command.

3.3.1 Shared-Memory Parallel COMSOL in Batch Mode

For COMSOL batch you will have to use the same command as before with the `-np` option, i.e.,

```
comsol -np Np batch -input poisson2d_infile.mph -output poisson2d_outfile.mph
```

Here `Np` represents the number of threads, which for a machine with two dual-cores can vary between 1 and 4. To run COMSOL batch on a compute node with the `-np` option activated, insert the above command into the last line of the scheduler submission script.

3.3.2 Shared-Memory Parallel COMSOL with MATLAB

To activate `Np` threads for COMSOL with MATLAB with interactive usage with plots, start COMSOL with the `-np` option and the `matlab` argument as follows

```
comsol -np Np matlab
```

To run COMSOL with MATLAB in background you will have to use the same command as shown in Section 2.3 but with the `-np` option, i.e.,

```
nohup comsol -np Np matlab path -ml -nodesktop -ml -nodisplay
    < poisson2d_matlab.m >& poisson2d_matlab.log &
```

Finally, to run COMSOL with MATLAB on a compute node with the `-np` option activated, insert

```
comsol -np Np matlab path -ml -nodesktop -ml -nodisplay < poisson2d_matlab.m
```

into the scheduler submission script. In addition, you need to activate the multi-thread capabilities of MATLAB. This can be done by adding the line `maxNumCompThreads(numThreads)` at the beginning of the `poisson2d_matlab.m` file. This command sets the maximum number of computational threads to `numThreads`. For example, if you wish to activate two computational threads, set the maximum number of computational threads to 2 by inserting the call `maxNumCompThreads(2)` at the beginning of the m-file.

To observe memory when running a job on a compute node of hpc, use first the command `qstat -n` to list the nodes on which your job is running. Then log into the node using `ssh node#` and type `top` on the compute node to observe the memory usage. Please note that you can login to a compute node only while your job is running.

3.3.3 Shared-Memory Parallel Performance Results

To avoid any influence of MATLAB in our performance report, we run COMSOL in two ways: (i) COMSOL in batch mode (standalone COMSOL) and (ii) COMSOL with MATLAB. In Table 3, we list the parallel performance results by the number of cores on one compute node for COMSOL in batch mode and COMSOL with MATLAB. Each column corresponds to the number of threads run by using the `-np` option. Each row lists the results we obtained by running COMSOL in batch mode or COMSOL with MATLAB for the four linear solvers listed in Table 2. (i) Reading along the rows of Table 3, we conclude that the speedup is not in proportion to the number of cores used, whether running COMSOL in standalone mode or with MATLAB. (ii) Comparing Tables 3 (a) and (b) confirms broadly that COMSOL in batch mode and COMSOL with MATLAB take the same amount of time to solve a problem. Both of these observations should be expected since MATLAB is only used as scripting tool and both ways to run COMSOL call the same solver functions. However, it appears that for unknown reasons the PARDISO linear solver does run slightly faster in COMSOL with MATLAB than in standalone mode. (iii) Our results also show that for our particular test problem the PARDISO linear solver outperforms all other solvers tested here. We note that COMSOL in batch mode failed to provide the mesh for the 9th refinement level in the memory available, hence only results of COMSOL with MATLAB are available for this case in Table 3 (c). The results in this table confirm otherwise the conclusions drawn so far, which confirms that the speedup is not affected by the relative smaller size of the problem for the 8th refinement.

Table 3: Wall clock times in seconds by the number of threads on one compute node for the four linear solvers listed in Table 2. The mesh sizes correspond to the indicated refinement levels $r = 8$ for (a) and (b) and $r = 9$ for (c) in Table 1.

(a) Run times (s) for COMSOL batch for $r = 8$

Solver	1 thread	2 threads	3 threads	4 threads
UMFPACK	58.94	45.49	42.68	43.26
SPOOLES	78.35	66.18	64.37	63.67
PARDISO	41.40	34.52	31.44	30.46
PCG–IChol	165.92	170.91	165.13	171.35

(b) Run times (s) for COMSOL with MATLAB for $r = 8$

Solver	1 thread	2 threads	3 threads	4 threads
UMFPACK	54.36	41.08	36.62	36.38
SPOOLES	78.70	61.39	64.14	53.28
PARDISO	32.19	26.76	24.92	23.36
PCG–IChol	168.53	192.47	159.05	184.08

(c) Run times (s) for COMSOL with MATLAB for $r = 9$

Solver	1 thread	2 threads	3 threads	4 threads
UMFPACK	356.48	257.32	212.00	189.09
SPOOLES	456.00	393.44	326.52	318.82
PARDISO	167.38	129.09	114.42	104.28
PCG–IChol	1637.38	1549.55	1512.67	1796.10

3.4 Distributed-Memory Model

The parallel operation mode based on the distributed-memory model is performed on multiple nodes in a cluster. When operating in this mode, COMSOL uses MPI for communicating between the processes in the distributed environment. The distributed mode can be activated by

```
comsol -nn Nn -np Np
```

where `-nn` directs COMSOL to start up `Nn` computational nodes and `-np` directs each computational node to use `Np` threads, so that the total number of parallel MPI processes is the product of the number of computational nodes multiplied by the number of threads run on each computational node. For additional information, see the subsection "*Distributed-Memory Parallel Comsol*" on page 72 of the COMSOL Installation and Operations Guide, Version 3.5a. Although everything points to the suggestion that COMSOL supports distributed parallelism, unfortunately after contacting COMSOL support we found that this capability is not available for this COMSOL version.

4 Conclusions

This work shows how to run COMSOL Multiphysics as a standalone product, or in conjunction with MATLAB on the cluster hpc in the UMBC High Performance Computing Facility. We give step-by-step instructions for running COMSOL in batch mode and with MATLAB in background as well as provide specific instructions on how to write a qsub scrip which allows you to submit computational jobs on hpc nodes. We remind the reader that all necessary files can be downloaded from `www.umbc.edu/hpcf`.

We also looked at how to activate the parallel capabilities for COMSOL and reported on the shared-memory parallel performance of COMSOL. The results presented in Table 3 show that the speedup is not in proportion to the number of cores used. The same results show that the poor parallel performance behavior is not induced by MATLAB.

The outlook for our work is to confirm with COMSOL support that what our tests show is the real behavior of COMSOL and is not induced by an improper installation of COMSOL or communication with the needed libraries. Then we wish to run the same tests on the new cluster scheduled to be available in Summer 2009. Finally, it will be worth testing the parallel capabilities of the new COMSOL version, scheduled for release later this year.

Acknowledgments

The authors would like to thank Samuel Trahan for his assistance and for contacting COMSOL support and obtaining the confirmation that COMSOL does not run in full distributed memory parallel mode. The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

References

[1] Dietrich Braess. *Finite Elements*. Cambridge University Press, third edition, 2007.

[2] Matthias K. Gobbert. A technique for the quantitative assessment of the solution quality on particular finite elements in COMSOL Multiphysics. In Vineet Dravid, editor, *Proceedings of the COMSOL Conference 2007, Boston, MA*, pp. 267–272, 2007.

[3] Matthias K. Gobbert and Shiming Yang. Numerical demonstration of finite element convergence for Lagrange elements in COMSOL Multiphysics. In Vineet Dravid, editor, *Proceedings of the COMSOL Conference 2008, Boston, MA*, 2008.

[4] Neeraj Sharma and Matthias K. Gobbert. Performance studies for multithreading in Matlab with usage instructions on hpc. Technical Report HPCF–2009–1, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2009.