Parallel Performance Studies for COMSOL Multiphysics Using Scripting and Batch Processing

Noemi Petra and Matthias K. Gobbert

Department of Mathematics and Statistics, University of Maryland, Baltimore County, 1000 Hilltop Circle, Baltimore, MD 21250, {znoemi1,gobbert}@math.umbc.edu

Abstract: The graphical user interface (GUI) of COMSOL Multiphysics offers an effective environment to get started solving problems. For reproducibility of the results, it is often desirable to explore the script-based modeling capabilities of COMSOL with MATLAB. There are also potential benefits of running COMSOL in parallel, specifically by running several computational threads in shared-memory parallelism mode. We use the scripting abilities of COMSOL with MATLAB to study the shared-memory parallel performance of COMSOL, that is, the solution time required by using multi-threading on one multiprocessor, multi-core computer with shared memory among the processors. The performance results show that using more than one thread saves time, but the speedup is not in proportion to the number of cores used.

Key words: Poisson equation, COMSOL with MATLAB, shared-memory parallelism.

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. 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. For a classical model problem specified in Section 2, this paper will show concretely in Section 3 how to transition from COMSOL's GUI to m-files in conjunction with MATLAB as scripting tool under Linux. We use the scripts in Section 4 to investigate and report on the parallel performance of COMSOL. For such a test, it is important to be able to perform several runs easily and reproducibly. We see that this is easily possible by changing just a few lines in an m-file saved from COMSOL's GUI.

In [3], the authors reported on performance studies for multi-threading in MATLAB. Their results demonstrate that the use of more than one thread is often not very beneficial for MATLAB code. In [2], the authors run COMSOL in two ways: (i) COMSOL with MATLAB and (ii) COMSOL in batch mode (standalone COMSOL) to avoid any influence of MATLAB in their performance report. The results reported there suggest that the speedup is not in proportion to the number of cores used, independent of the way COMSOL is run. In this paper, our primary goal is to extend the studies from [2] to Lagrange finite elements of higher degrees. We focus on the linear solver PARDISO, since it performed the best in [2] and it is expected to profit most from multi-threading on a shared memory node. Specifically, our studies tests if the speedup measured for PARDISO improves with increasing polynomial degree of the Lagrange finite elements in COMSOL. Our findings suggest that it saves time to use more than one thread, with the most improvement from one to two threads, which is expected on a cluster node with two dual-core processors (as opposed to one quad-core processor, for instance). However, the performance improvement does not get better with increasing the order of the polynomial degrees.

The numerical studies in this report were performed on one cluster node of the distributed-memory Linux cluster hpc in the UMBC High Performance Computing Facility (HPCF; `www.umbc.edu/hpcf`). The node used has two dual-core AMD Opteron 2.66 GHz processor (1 MB cache per core) and 17 GB memory.

2 Test Problem

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

$$\begin{aligned} -\Delta u &= f \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \partial\Omega, \end{aligned} \tag{2.1}$$

on the unit square $\Omega = (0,1) \times (0,1) \subset \mathbb{R}^2$. Here, $\partial\Omega$ denotes the boundary of the domain Ω, and the Laplace operator Δ is defined as $\Delta u = u_{xx} + u_{yy}$ in two spatial dimensions. To test the numerical method, we consider the elliptic problem (2.1) with right-hand side function

$$\begin{aligned} f(x,y) = &- 2\pi^2 \cos(2\pi x)\, \sin^2(\pi y) \\ &- 2\pi^2 \sin^2(\pi x)\, \cos(2\pi y), \end{aligned}$$

for which the problem admits the known true solution

$$u(x,y) = \sin^2(\pi x)\, \sin^2(\pi y). \tag{2.2}$$

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

$$\|u - u_h\|_{L^2(\Omega)} = \left(\iint_\Omega (u - u_h)^2 \, dx\, dy \right)^{1/2} \tag{2.3}$$

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 [1].

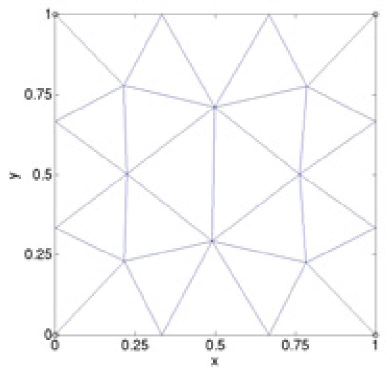
3 Use of COMSOL

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 and linear Lagrange elements. We intend to generalize this script after wards with all possible polynomial degrees 1 through 5, as well as for several meshes, obtained by regular refinement from a coarse initial mesh, and compare the timing results. More details on how to create a MATLAB script file for numerical convergence studies are available in [1].

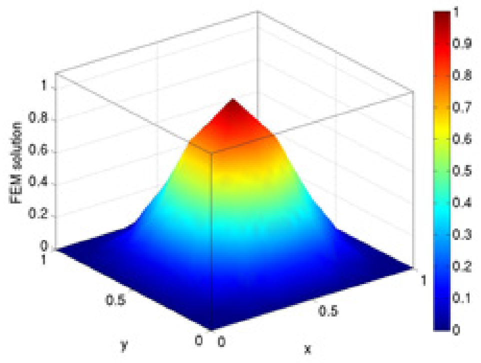
To solve the model problem, start the GUI of COMSOL Multiphysics, by typing `comsol` at the Linux prompt, which is shorthand for `comsol multiphysics`. In the Model Navigator for the problem dimension select 2D, then select COMSOL Multiphysics → PDE Modes → PDE, Coefficient Form → Stationary analysis. For our testing purposes it is important that linear Lagrange elements are selected at this stage. Then, in the draw mode of the GUI, draw the desired domain $\Omega = (0,1) \times (0,1)$. To set up the source, set f in the Physics → Subdomain Settings window to `(-2*pi^2)*(cos(2*pi*x)*sin(pi*y)^2+ sin(pi*x)^2*cos(2*pi*y))`. For our model problem, all other PDE and boundary coefficients are at their default values.

Mesh the domain with an extremely coarse initial mesh, so that we can use several regular refinement levels later. Thus, in the Mesh → Free Mesh Parameters dialog window, we select the Predefined mesh size as Extremely coarse; notice that the Refinement method in 2D is Regular by default. This gives the initial mesh with 26 triangular elements and 20 vertices shown in Figure 1 (a). In order to have the command for mesh refinements included in the script, we select the mesh refinement button in the GUI once now; this subdivides each triangular mesh element into four congruent triangles to obtain a mesh with 104 elements and 65 vertices (not shown).

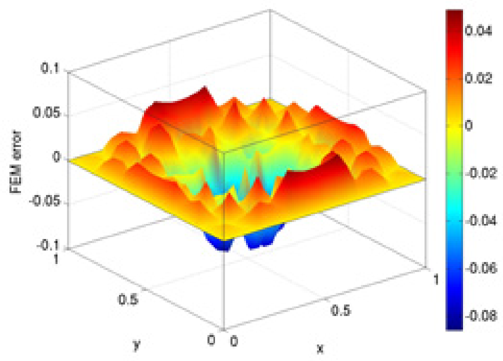
Since we expect the linear solver PARDISO to have the best chance for performing well on a multi-core processor, we select under Solve → Solve Parameters the Linear System Solver "Direct (PARDISO)". Then have COMSOL solve the problem using this linear solver on the mesh with one regular refinement.



(a)



(b)



(c)

Figure 1: (a) Initial mesh of the unit square domain. (b) FEM solution and (c) FEM error using linear Lagrange elements on a mesh with one regular mesh refinement.

The solution on this mesh, computed using the linear Lagrange elements chosen in the Model Navigator originally, is shown in Figure 1 (b). The plot shown is the 3D Surface Plot of the solution, without title, axes limits controlled manually, with plot box, and axes labels added. Using the known true solution (2.2), we can also plot the error by replacing the default expression `u` by the expression for the error `u-sin(pi*x)^2*sin(pi*y)^2` in the Postprocessing → Plot Parameters dialog windows; specifically, in the Surface tab, we replace the expression both for the Surface Data and for the Height Data with this expression. In the 3D Surface Plot, without title, axes limits controlled manually, with plot box, and with labels added, this gives the plot in Figure 1 (c).

To compute the L^2-norm of the error between the FEM solution and the true solution given by (2.3), have COMSOL first compute the square of the norm by entering in the Postprocessing → Subdomain Integration window the expression `(u-sin(pi*x)^2*sin(pi*y)^2)^2`. COMSOL then computes the integral in (2.3), that is, the square of the error norm. The value of integral is reported in the report window located at the bottom of the GUI, which in our case is 7.020117e-4. To obtain the L^2-norm error, take the square root of this value (using a calculator or similar), which gives you $\|u - u_h\|_{L^2(\Omega)} \approx$ 2.6496e-2 for the solution on this mesh.

At this point, create the m-file by saving the entire interactive session in the GUI up to this point as `poisson2d.m` using File → Save As. We remind the reader that 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.

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 $r = 0, 1, \ldots$ and for different Lagrange polynomial degrees. At the same time, we add a few more capabilities to our function to output desired quantities such as the elapsed wall clock time for the solution process. Thus, edit the `poisson2d.m` and save as `poisson2d_matlab.m` as follows:

- In the first line of the file, insert the function header

  ```
  function [fem, tsec] = ...
       poisson2d_matlab(p, r)
  ```

  This means that the function will accept the degree of Lagrange finite elements p and the number of regular mesh refinements r as input variables. The function returns the COMSOL FEM object along with the elapsed wall clock time in seconds (see below) to the calling driver routine.

- Insert `flreport('off')` at the beginning of the m-file, which suppresses the solution progress report from COMSOL to make this function suitable for running in the background.

- Search for the call to `meshrefine` and enclose it in a for-loop of the number of refinements r input in the function call:

  ```
  for i = 1 : r
    fem.mesh = ...
      meshrefine(fem, ...
                 'mcase',0, ...
                 'rmethod','regular');
  end;
  ```

- Search for the choice of the finite elements used and replace the line `prop.elemdefault='Lag1'` by

  ```
  prop.elemdefault=sprintf('Lag\%1d',p);
  ```

  This `sprintf` command appends the number p to the string "Lag" and thus allows us to choose the degree of the Lagrange elements as input to this function.

- Insert the `tic` and `toc` functions before and after the solve command `femstatic`, respectively, to measure the elapsed wall clock time.

- Delete (or comment out) all plot commands such as `postplot` and `postint`.

This completes the conversion of the COMSOL script `poisson2d.m` to the function `poisson2d_matlab.m`. Since this is a function, we can now write a script `driver_poisson2d.m` that calls the function `poisson2d_matlab.m` repeatedly for all desired values of Lagrange degree p and refinement level r, using for-loops such as

```
for p = 1 : 5
  ...
  for r = nrefmin : nrefmax
    [fem, tsec] = ...
      poisson2d_matlab (p, r);

    I1=postint(fem, ...
      '(u-sin(pi*x)^2*sin(pi*y)^2)^2', ...
      'unit','', ...
      'recover','off', ...
      'dl',1);
    errnorm = sqrt(I1);
    ...
  end;
end;
```

It is worth mentioning here that the use of higher Lagrange elements increases the number of degrees of freedoms and hence the dimension of the problem. Therefore, so that all data structures fit in the available memory of the computer, we choose different values of `nrefmin` and `nrefmax` for different values of p; this control logic between the two for-loops is not shown above. We show here also how the `postint` command, saved as part of the original script `poisson2d.m`, can be moved here and then used to compute the L^2-norm by `sqrt(I1)`. The remainder of the script (not shown) outputs various other quantities that will appear in the table of results in the next section.

4 Results

The driver script `driver_poisson2d.m` developed in the previous section computes the solution to the model problem using the PARDISO linear solver for Lagrange finite elements with all possible degrees $p = 1, \ldots, 5$ and for several meshes, each obtained by refining a coarse initial mesh r times regularly. The script records the elapsed wall clock time for each run as measure of performance. It also obtains several pieces of information about the mesh that characterize the complexity of the FEM problem. Specifically, the number of mesh elements N_e and the number of vertices N_p in the mesh are printed, and the number of degrees of freedom DOF. The DOF is the number of unknowns that the linear solver has to obtain, and this is the true indication of computational complexity of each problem. For completeness, we also compute the L^2-norm of the FEM error between

FEM solution and the known true solution to confirm the correctness of the computed solution.

To study the performance as a function of the number of computational threads, the script is run for each possible value of threads on our cluster node with two dual-core processors. For instance in the case of 2 threads, we start COMSOL with MATLAB from the Linux command-line by

```
  comsol -np 2 matlab path
```

The quick help from `comsol -h` explains the option `-np` as setting the "number of processors". More precisely, the COMSOL Installation and Operations Guide explains the meaning as shared-memory parallelism, which is most precisely termed multi-threading on a node with shared memory among the processors. See also `comsol -h matlab` for more information on starting COMSOL with MATLAB under Linux.

For consistency with the use of Matlab, we also set its number of threads to the same number, in this example to 2, by setting

```
>> maxNumCompThreads(2);
```

at the Matlab prompt. We note that Matlab is merely used to interpret the command line and that we only time the `femstatic` command in COMSOL, hence this setting is likely not relevant.

The numerical tests are performed with 1, 2, 3, and 4 threads in place of the 2 in the commands quoted above. Table 1 lists the shared-memory parallel performance results for each Lagrange finite element and for four refinement levels for each. Notice that the refinement levels vary for different Lagrange elements, such that the DOF are similar in magnitude for each subtable. Lagrange finite elements with higher degrees have more degrees of freedom on each mesh element, thus their DOF are higher. On the one hand, this leads them to run out of memory on coarser meshes than Lagrange elements with lower degrees, exactly corresponding to the number of DOFs. We mention that the maximum observed memory usage was 14.5 GB for the case $p = 5$ and $r = 7$. On the other hand, as the printed errors readily show, higher degree elements (for a PDE problem with a sufficiently smooth solution) can give much smaller errors for equivalent work; for instance, the DOF work associated with $p = 1$ and $r = 9$ are identical to the ones with $p = 2$ and $r = 8$, but the error is actually three orders of magnitude better for a similar amount of work.

However, the structure of the system matrix is different for Lagrange elements with different degrees, even if the numbers of DOF are exactly the same. More precisely, for higher degrees p one expects slightly less sparsity and a tighter clustering of non-zero terms in the system matrix. On the one hand, this makes the solution slightly more expensive for higher degrees, as born out by the comparisons of the case $p = 1$ and $r = 9$ to the case $p = 2$ and $r = 8$, for instance. But on the other hand, we might expect potentially better speedup when going from 1 to 2 or more threads.

As it turns out, the results in Table 1 for our tests using the PARDISO linear solver for this test problem are more uniform, though. The elapsed wall clock time increases with the degrees of freedom of the finite element method, with higher degree Lagrange elements being slightly more expensive, if the DOFs are equal. For the most computationally intensive refinement for each Lagrange element, using 2 threads instead of 1 thread saves about 25% of wall clock time, which is less than the theoretically expected 50%. The improvement resulting from using 3 or 4 threads is less significant, which might be expected, as 3 or 4 threads require the use of both dual-core processors on a cluster node.

References

[1] 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*, pages 267–272, 2007.

[2] Noemi Petra and Matthias K. Gobbert. Performance studies with COMSOL Multiphysics via scripting and batch processing. Technical Report HPCF–2009–4, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2009.

[3] 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.

Table 1: Elapsed wall clock times in seconds for the linear solver PARDISO using 1, 2, 3, 4 threads on one cluster node. The mesh sizes correspond to the indicated refinement levels r.

r	N_e	N_p	DOF	E_r	1 thread	2 threads	3 threads	4 threads
6	106496	53633	53633	2.6324e-005	1.45	1.24	1.10	1.09
7	425984	213761	213761	6.5811e-006	6.54	5.34	4.89	4.64
8	1703936	853505	853505	1.6453e-006	32.29	26.16	23.25	21.81
9	6815744	3410945	3410945	4.1133e-007	171.85	126.20	112.25	102.08

(a) Lagrange elements with $p = 1$.

r	N_e	N_p	DOF	E_r	1 thread	2 threads	3 threads	4 threads
5	26624	13505	53633	2.8487e-007	1.53	1.36	1.31	1.21
6	106496	53633	213761	3.5635e-008	6.76	5.88	5.43	5.20
7	425984	213761	853505	4.4560e-009	33.79	27.46	25.39	24.23
8	1703936	853505	3410945	5.7108e-010	181.02	137.85	122.83	117.54

(b) Lagrange elements with $p = 2$.

r	N_e	N_p	DOF	E_r	1 thread	2 threads	3 threads	4 threads
5	26624	13505	120385	2.0616e-009	4.85	3.45	3.21	3.11
6	106496	53633	480385	1.2891e-010	20.81	16.27	14.80	13.86
7	425984	213761	1919233	3.1744e-011	95.44	73.54	66.77	63.78
8	1703936	853505	7672321	1.1865e-010	539.28	390.09	339.73	315.73

(c) Lagrange elements with $p = 3$.

r	N_e	N_p	DOF	E_r	1 thread	2 threads	3 threads	4 threads
4	6656	3425	53633	1.2633e-010	1.98	1.64	1.54	1.56
5	26624	13505	213761	3.9761e-012	8.63	7.07	6.41	6.25
6	106496	53633	853505	1.0879e-012	39.11	32.40	29.05	28.16
7	425984	213761	3410945	3.0417e-012	200.35	159.83	141.43	130.76

(d) Lagrange elements with $p = 4$.

r	N_e	N_p	DOF	E_r	1 thread	2 threads	3 threads	4 threads
4	6656	3425	83681	1.3803e-012	3.75	2.97	2.80	2.67
5	26624	13505	333761	1.1108e-012	15.21	12.61	11.66	11.01
6	106496	53633	1333121	5.1919e-012	73.40	57.05	50.41	48.72
7	425984	213761	5328641	1.9951e-011	362.94	263.55	241.03	219.44

(e) Lagrange elements with $p = 5$.