

CSE 598D Supplementary Handout

Algorithms as if Architecture Mattered

Parallel Computing Exercise

distributed 17 March 1999

This exercise has four parts, which require increasing levels of effort and understanding. It is recommended to undertake them in order.

0) PETSc.

Login to the class account systems and find PETSc at the location announced by the instructor, with relative pathname `~keyes/cse598d/petsc`.

Become familiar with it in a cursory way from the on-line html facility at www.mcs.anl.gov/petsc/docs/manualpages/manualpages.html and Chapter 1 of the downloadable user manual.

Define the two PETSc environment variables `PETSC_DIR` and `PETSC_ARCH` using the `setenv` command in Unix, according to the computational environment you are in. (The architectures could be `rs6000` or `solaris`, for instance.)

Create a directory (so you have write permission) and copy into it `ex8.c`, `common8and9.c`, `ex8and9.h`, and `makefile` from `$PETSC_DIR/src/snes/examples/tutorials`.

Customize the makefile to your personal taste (if you want to examine one that works properly in your local environment, see the instructor's at `~keyes/cse598d/petsc`), and build `ex8` with

```
make BOPT=g ex8
```

Run with

```
mpirun -np 1 ex8 -mx 32 -my 32 -grashof 1000. -contours -draw_pause 5  
-snes_monitor -ksp_monitor
```

Observe the rapid convergence of the nonlinear residual norm and the solution plots with legends for two velocity components, vorticity, and temperature in a square domain — the lid- and thermally-driven cavity problem. (If you do not get the plots, you will have to use the `setenv DISPLAY` and `xhost +` commands.)

Run again with the switches `-log_info` and `-log_summary` and try to make sense of the plethora of information you get, along with the convergence plots. You may want to pipe the output into a file for study in an editor, and you may want to suppress graphics (by leaving off the `-contours` option):

```
mpirun -np 1 ex8 -mx 32 -my 32 -grashof 1000. -log_info -log_summary  
-snes_monitor -ksp_monitor > ex8_log.txt
```

1) The Lid- and Thermally-driven Cavity.

The code in `ex8.c` approximates the solution of the following system of four elliptic partial differential equations in two dimensions (x, y) :

$$-\nabla^2 u - \frac{\partial \omega}{\partial y} = 0, \quad (1)$$

$$-\nabla^2 v + \frac{\partial \omega}{\partial x} = 0, \quad (2)$$

$$-\nabla^2 \omega + u \frac{\partial \omega}{\partial x} + v \frac{\partial \omega}{\partial y} - \text{Gr} \frac{\partial T}{\partial x} = 0, \quad (3)$$

$$-\nabla^2 T + \text{Pr} \left(u \frac{\partial T}{\partial x} + v \frac{\partial T}{\partial y} \right) = 0, \quad (4)$$

where $(u(x, y), v(x, y))$ are the velocity fields in the (x, y) directions, $\omega(x, y) = -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$ is the component of the vorticity normal to the xy -plane (representing the in-plane rotation of an infinitesimal element), and $T(x, y)$ is the temperature.

These equations are subject to boundary conditions:

- along the bottom ($0 < x < 1, y = 0$): $u = v = 0, \frac{\partial T}{\partial y} = 0$
- along the top ($0 < x < 1, y = 1$): $u = V_{lid}, v = 0, \frac{\partial T}{\partial y} = 0$
- along the left ($x = 0, 0 < y < 1$): $u = v = 0, T = 0$
- along the right ($x = 1, 0 < y < 1$): $u = v = 0, T = 1$ if $\text{Gr} > 0$ or 0 otherwise

On each boundary, ω is given by its definition, $\omega(x, y) = -\frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$, based on the normal gradients of either u and v (the tangential gradients being zero, according to the velocity boundary conditions).

Play with the three dimensionless parameters switchable with `-grashof`, `-prandtl`, and `-lidvelocity` and develop an intuition for how the difficulty of converging the problem varies with their relative magnitudes. (N.B.: For those who would rather see a Peclet number than a Prandtl number in the energy equation, Reynolds number is unity, so Pe and Pr are identical.) Pay attention to the initial residual norms. Make a table (with `-np 1 -mx 32 -my 32`) of at least 27 interesting triples of `-grashof -prandtl -lidvelocity` (three values for each, varied independently), and tabulate:

- the initial residual norm
- whether you can achieve convergence without monkeying with the linear solver (default KSP and PC options) or the nonlinear solver (default SLES options), and, if so, in how many iterations.

Repeat runs for a case in which the lid velocity is zero, Prandtl is unity, and Grashof is not too large, for varying `-mx -my`. (You may keep these two equal to each other, or you can try varying them separately.) Track the maximum value of the vorticity in the converged solution over at least five doublings of your mesh parameter.

Now run the code in parallel; that is, play with `-np` (or more directly with `-Nx -Ny`) to create rectilinearly domain decomposed problems. Use the default Schwarz options at first; then try extra overlap (see the `man` pages). Try extra levels of ILU fill, and extra vectors for the restarted GMRES. Try at least one SLES accelerator besides GMRES. For each parameter (number of subdomains, overlap, ILU strength, GMRES restart size, alternative accelerator) note at least one trend you find interesting, and document the set of command lines that led you to it.

2) Mesh Sequencing.

Use PETSc's mesh sequencing feature, as implemented for you in `ex9`, by starting to solve the problem on an exceedingly coarse grid, and then recursively doubling the grid density using an interpolate of the solution on the previous grid as the initial iterate on the new grid. To illustrate, try

```
mpirun -np 1 ex9 -mx 5 -my 5 -ncycles 5 -printg
-snes_smonitor -lidvelocity 0. -grashof 1000.
```

Compare the time taken to solve initially on a given fine grid (e.g., 256×256) with the *total time* required to solve on all grids (e.g., 4×4 , 8×8 , 16×16 , 32×32 , 64×64 , 128×128 , and 256×256) through mesh sequencing — sometimes called “one-way multigrid.” (Note that because PETSc's `-mx` parameter measures the number of points into which the x -interval is discretized, you must set `-mx 5` in order to get 4 subintervals, `-mx 9` in order to get 8 subintervals, etc.)

Compare the initial residual norm on the finest grid from a cold start to the initial residual norm on the finest grid from the interpolate of the previous.

Also, compare the initial residual norms of each member of the sequence of recursively doubled problems with each other.

3) The Salinity-driven Cavity.

Copy `ex8.c` into `ex10.c` and augment the test suite as follows. Invent a new property; call it $S(x, y)$ for “salinity”. By augmenting the unknown vector and imitating the energy equation, add a fifth component to the problem, with an equation for S like the energy equation. You will need to define a new parameter to take the place of Prandtl — call it A ; and a new parameter to take the place of Grashof — call it B . Hook these parameters into the command line by imitating the way the other parameters are input.

Set up boundary conditions for S as follows:

- along the bottom ($0 < x < 1, y = 0$): $\frac{\partial S}{\partial y} = 0$ (bottom is impermeable)
- along the top ($0 < x < 1, y = 1$): $S(x, 1) = -\cos[2\pi(x - \frac{1}{2})]$ (S is injected near the walls and removed in the middle)
- along the sides ($x = 0, 1, 0 < y < 1$): $\frac{\partial S}{\partial x} = 0$ (side walls are impermeable)

Set up the initial iterate for $S(x, y)$ to linearly interpolate in y the given $S(x, 1)$ at the top to zero at the bottom for each x .

Choose unity for A and run the simulation with

```
-lidvelocity 0. -prandtl 1. -grashof 1000.
```

Capture the contour plot of S and print it in color postscript (or save it in a publicly readable file).

Since the S -equation does not couple back into any of the other equations, the other four fields should not change as you vary A . Verify that this is true.

Now, put a source term like the Grashof term into the vorticity equation, using parameter B and an x -gradient of S . Document in your write-up exactly what equation you are modeling, from the continuous differential equation perspective. Play with B (both signs), keeping the other parameters as in the previous tests and interpret your results.

The due date on this assignment will be sometime in April.

As the short course progresses, you will understand progressively more of the background of this assignment in each way — the application (slight), the tunable algorithmic family (more), and what to expect in terms of performance (the main purpose).

You do not have to write any code in order to get started immediately with a parameterized parallel computation in which the number of processors (generically p) and the problem size (generically n) can be varied widely. This makes the code an experimental testbed for performance.

In addition, you can tune each component of the solver (the Newton-like nonlinear solver, the Krylov linear solver, and the Schwarz-style preconditioner and its subdomain components) from the command line, making the code an experimental testbed for algorithms.

In addition, you can play with a few different types of nonlinearity and (indirectly) with the conditioning of the problem. (The conditioning gets worse as the mesh size gets finer, that is, as n increases.) This allows you to “stress” both the nonlinear and linear aspects of the solution algorithms.

Finally, this code may be a reasonable template for some other parallel PDE coding projects you may have (or elect to integrate) into your other courses at Notre Dame.

Have fun! Student-authored additions could be incorporated into future releases.