PETSc

Portable, Extensible Toolkit for Scientific Computation

Karl Rupp
rupp@mcs.anl.gov

Mathematics and Computer Science Division
Argonne National Laboratory

Tutorial at Segundo Encuentro Nacional de Computación de Alto Rendimiento para Aplicaciones Científicas

May 9th, 2013

Step 1: Hello World

Initialize Environment

```
$> export PETSC_DIR=/opt/petsc-20130502
$> git clone https://github.com/karlrupp/whpc13.git

$> cd whpc13
$> module load mpi
$> cp scripts/submit_job.sh .
$> make
$> salloc --gres=gpu:1 ./submit_job.sh
```

Options

Provided via `petscrc`

Step 2: Poisson Equation

Poisson Equation

$$-\Delta u = 0$$

```
$> git checkout -f step-2
$> make
$> salloc --gres=gpu:1 ./submit_job
```

Quiz

First correct answer wins chocolate!

Question

Which preconditioner (PC) type is used?

Hints

- Use options from petscrc
- Read and try to understand output

Step 3: Poisson Equation with Nonlinearity

Poisson Equation with Nonlinearity

$$-\Delta u - \lambda e^{u} = 0$$

```
$> git checkout -f step-3
$> make
$> salloc --gres=gpu:1 ./submit_job
```

Options

Provided via `petscrc`

Quiz

Preparation

Use the following options in `petscrc`:

```
-da_grid_x  80
-da_grid_y  80
-ksp_max_it 500
```

Question

Determine critical parameter λ_{crit} (divergence)

Accuracy: One decimal after comma, e.g. 4.2

Step 4: $\mathfrak{p}$-Bratu Equation

$\mathfrak{p}$-Bratu Equation

$$-\nabla \cdot (\eta \nabla u) - \lambda e^{u} - f = 0$$

$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{\mathfrak{p}-2}{2}} \qquad \gamma(u) = \frac{1}{2}|\nabla u|^2$$

```
$> git checkout -f step-4
$> make
$> salloc --gres=gpu:1 ./submit_job
```

Options

Provided via `petscrc`

Quiz

$$-\nabla \cdot (\eta \nabla u) - \lambda e^{u} - f = 0$$

$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{p-2}{2}} \qquad \gamma(u) = \frac{1}{2}|\nabla u|^2$$

Preparation

Use the following options in `petscrc`:

```
-da_grid_x 40
-da_grid_y 40
```

Three Questions

a) $(p > 1, \lambda \geq 0, \varepsilon \in [10^{-9}, 10^{-2}])$:
convergence within at most 2 Newton iterations

b) $p = 1.1$: Find $(\lambda, \varepsilon \in [10^{-9}, 10^{-2}]))$ for convergence

c) $(p > 2, \lambda \geq 0, \varepsilon \in [10^{-9}, 10^{-2}]))$:
convergence with *more* than 30 Newton iterations

Step 5: Providing Jacobian

$\mathfrak{p}$-Bratu Equation

$$-\nabla \cdot (\eta \nabla u) - \lambda e^{u} - f = 0$$

$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{\mathfrak{p}-2}{2}} \qquad \gamma(u) = \frac{1}{2}|\nabla u|^2$$

```
$> git checkout -f step-5
$> make
$> salloc --gres=gpu:1 ./submit_job
```

Use 'simpler' Jacobians for Newton

- Just use $-\Delta w - e^{u} w$ (simplest)
- More detail: Include η, but not η'

Play with Parameters

Check out the following options (one at a time):

```
-snes_mf
-snes_fd
-jtype 1 -myJ
-jtype 2 -myJ
```

Look at Scalability

Adjust

```
-da_grid_x 40
-da_grid_y 40
```

Go up to (don’t use `-snes_fd`)

```
-da_grid_x 160
-da_grid_y 160
```

Step 6: Full Implementation

$\mathfrak{p}$-Bratu Equation

$$-\nabla \cdot (\eta \nabla u) - \lambda e^{u} - f = 0$$

$$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{\mathfrak{p}-2}{2}} \qquad \gamma(u) = \frac{1}{2}|\nabla u|^2$$

```
$> git checkout -f step-6
$> make
$> salloc --gres=gpu:1 ./submit_job
```

Two more implementations for Jacobian

`-jtype 3`: Jacobian on 5-star stencil

`-jtype 4`: Full Jacobian

Final Quiz

Preparation

Use the following options in `petscrc`:

```
-da_grid_x 320
-da_grid_y 320
-p         3
-lambda    2
```

Task

Find the fastest set of parameters (see petscrc)

Use `-log_summary`

Experiment with

- CPU vs. GPU
- Jacobian Matrices (`-jtype`)
- Linear solvers (KSP)
- Preconditioners (PC)

Conclusions

PETSc can help You

- solve algebraic and DAE problems in your application area
- rapidly develop efficient parallel code, can start from examples
- develop new solution methods and data structures
- debug and analyze performance
- advice on software design, solution algorithms, and performance

`petsc-{users,dev,maint}@mcs.anl.gov`

You can help PETSc

- report bugs and inconsistencies, or if you think there is a better way
- tell us if the documentation is inconsistent or unclear
- consider developing new algebraic methods as plugins, contribute if your idea works