PETSc
Portable, Extensible Toolkit for Scientific Computation

Karl Rupp
me@karlrupp.net

Freelance Computational Scientist

Workshop on Modern Scientific Computing
DTU Compute, Technical University of Denmark

March 2-3, 2016
Step 1: Setup

Initialize Environment

```bash
$> git clone https://github.com/karlrupp/dtu2016_2.git
$> cd dtu2016_2
$> make
$> ./pbratu
```

Options

Provided via petsrcr
Poisson Equation

\[-\Delta u = 0\]

$>$ git checkout -f step-2
$>$ make
$>$ ./pbratu
Step 3: Poisson Equation with Nonlinearity

Poisson Equation with Nonlinearity

\[-\Delta u - \lambda e^u = 0\]

Options
Provided via petscrc
Quiz

Preparation

Use the following options in `petscrc`:

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

Determine critical parameter $\lambda_{\text{crit}}$ (divergence) with accuracy: One decimal after comma, e.g., 4.2.
Quiz

Preparation

Use the following options in petsrcrc:

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

Question

Determine critical parameter $\lambda_{\text{crit}}$ (divergence)

Accuracy: One decimal after comma, e.g. 4.2
Step 4: \( p \)-Bratu Equation

\( p \)-Bratu Equation

\[-\nabla \cdot (\eta \nabla u) - \lambda e^u - f = 0\]

\[\eta(\gamma) = (\epsilon^2 + \gamma) \frac{p-2}{2} \quad \gamma(u) = \frac{1}{2} |\nabla u|^2\]

\begin{verbatim}
$> \text{git checkout -f step-4}
$> \text{make}
$> \text{./pbratu}
\end{verbatim}

Options
Provided via petscrc
Step 5: Providing Jacobian

\( p \)-Bratu Equation

\[-\nabla \cdot (\eta \nabla u) - \lambda e^u - f = 0\]

\[\eta(\gamma) = (e^2 + \gamma)^{\frac{p-2}{2}} \quad \gamma(u) = \frac{1}{2} |\nabla u|^2\]

\begin{verbatim}
$> git checkout -f step-5
$> make
$> ./pbratu
\end{verbatim}

Use 'simpler' Jacobians for Newton

Just use \(-\Delta w - e^u w\) (simplest)

More detail: Include \(\eta\), but not \(\eta'\)
Play with Parameters

Check out the following options (one at a time, \texttt{-snes_monitor}): 
\begin{verbatim}
  -snes_mf
  -snes_fd
  -jtype 1  -myJ
  -jtype 2  -myJ
\end{verbatim}

Look at Scalability

Adjust
\begin{verbatim}
  -da_grid_x  40
  -da_grid_y  40
\end{verbatim}

Go up to (don’t use \texttt{-snes_fd})
\begin{verbatim}
  -da_grid_x 160
  -da_grid_y 160
\end{verbatim}
Step 6: Full Implementation

\[ -\nabla \cdot (\eta \nabla u) - \lambda e^u - f = 0 \]

\[ \eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{p-2}{2}} \quad \gamma(u) = \frac{1}{2} |\nabla u|^2 \]

$> \text{git checkout -f step-6}$
$> \text{make}$
$> \text{./pbratu}$

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`:

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>-da_grid_x</td>
<td>320</td>
</tr>
<tr>
<td>-da_grid_y</td>
<td>320</td>
</tr>
<tr>
<td>-p</td>
<td>3</td>
</tr>
<tr>
<td>-lambda</td>
<td>2</td>
</tr>
</tbody>
</table>

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)
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

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

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