PetIGA
A Framework for High Performance Isogeometric Analysis

Lisandro Dalcin$^{1,3}$, Nathaniel Collier$^{2}$, Adriano Côrtes$^{3}$, Victor M. Calo$^{3}$

$^{1}$Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET)
Santa Fe, Argentina

$^{2}$Oak Ridge National Laboratory (ORNL), Knoxville, United States

$^{3}$King Abdullah University of Science and Technology (KAUST)
Thuwal, Saudi Arabia

Third International Workshop on Multiphysics, Multiscale, and Optimization Problems
Bilbao, Spain – May 22-23, 2014
What is PetIGA?

PetIGA is an implementation of isogeometric analysis built on top of PETSc (the Portable, Extensible Toolkit for Scientific Computation).

$$\text{PetIGA} = \text{Pet} \ (\text{PETSc}) + \text{IGA} \ (\text{isogeometric analysis})$$
IGA: FEM + B-Spline/NURBS

\[
p = 2 \ C^0
\]

\[
p = 2 \ C^1
\]

\[
p = 3 \ C^0
\]

\[
p = 3 \ C^2
\]
Portable, Extensible Toolkit for Scientific Computation

We base our framework on PETSc

- Eases development of large-scale scientific codes
- Provides a rich environment for prototyping
- Grants parallelism with little interaction with MPI

Solving time-dependent, nonlinear PDE-based problems

- Requires specification of two user-defined functions:
  1. Residual (or Function) evaluation (Vec)
  2. Tangent (or Jacobian) evaluation (Mat)
- Rest is handled inside a hierarchy of solvers:
  - Timestepping loop (TS)
  - Nonlinear loop (SNES)
  - Linear loop (KSP)
  - Preconditioner (PC)
Main Routine

Timestepping Solvers (TS)

Nonlinear Solvers (SNES)

Linear Solvers (KSP)

Preconditioners (PC)

Application

Initialization

Function Evaluation

Jacobian Evaluation

Postprocessing

PETSc
Parallel implementation

How it works …

Structured grid

Processors

\[ P_0 \quad P_1 \]
\[ P_2 \quad P_3 \]
Parallel implementation

Natural vs. Global numbering

<table>
<thead>
<tr>
<th>Natural numbering</th>
<th>Global numbering</th>
</tr>
</thead>
<tbody>
<tr>
<td>30 31 32</td>
<td>24 25 26</td>
</tr>
<tr>
<td>24 25 26</td>
<td>27 28 29</td>
</tr>
<tr>
<td>18 19 20</td>
<td>21 22 23</td>
</tr>
<tr>
<td>12 13 14</td>
<td>15 16 17</td>
</tr>
<tr>
<td>6 7 8</td>
<td>9 10 11</td>
</tr>
<tr>
<td>0 1 2</td>
<td>3 4 5</td>
</tr>
<tr>
<td></td>
<td>33 34 35</td>
</tr>
<tr>
<td></td>
<td>21 22 23</td>
</tr>
<tr>
<td></td>
<td>27 28 29</td>
</tr>
<tr>
<td></td>
<td>15 16 17</td>
</tr>
<tr>
<td></td>
<td>12 13 14</td>
</tr>
<tr>
<td></td>
<td>9 10 11</td>
</tr>
<tr>
<td></td>
<td>0 1 2</td>
</tr>
</tbody>
</table>
Parallel implementation

Global vs. Local numbering: $C^0$ case

<table>
<thead>
<tr>
<th>24</th>
<th>25</th>
<th>26</th>
<th>33</th>
<th>34</th>
<th>35</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>22</td>
<td>23</td>
<td>30</td>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>18</td>
<td>19</td>
<td>20</td>
<td>27</td>
<td>28</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>6</th>
<th>7</th>
<th>8</th>
<th>15</th>
<th>16</th>
<th>17</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>12</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Global numbering

<table>
<thead>
<tr>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>X</td>
</tr>
</tbody>
</table>

Local numbering

<table>
<thead>
<tr>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>11</th>
<th>X</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>3</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>
Parallel implementation

Global vs. Local numbering: $C^1$ case

<table>
<thead>
<tr>
<th>24 25 26</th>
<th>33 34 35</th>
</tr>
</thead>
<tbody>
<tr>
<td>21 22 23</td>
<td>30 31 32</td>
</tr>
<tr>
<td>18 19 20</td>
<td>27 28 29</td>
</tr>
<tr>
<td>6 7 8</td>
<td>15 16 17</td>
</tr>
<tr>
<td>3 4 5</td>
<td>12 13 14</td>
</tr>
<tr>
<td>0 1 2</td>
<td>9 10 11</td>
</tr>
</tbody>
</table>

Global numbering

<table>
<thead>
<tr>
<th>X X X X</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 21 22</td>
</tr>
<tr>
<td>15 16 17</td>
</tr>
<tr>
<td>10 11 12</td>
</tr>
<tr>
<td>5 6 7</td>
</tr>
<tr>
<td>0 1 2</td>
</tr>
</tbody>
</table>

Local numbering
PetIGA: basic usage

- Initialize the IGA context

```c
IGA iga;
IGACreate(PETSC_COMM_WORLD,&iga);
IGASetDof(iga,1); // scalar problem
IGASetDim(iga,2); // in 2D
IGASetFromOptions(iga);
IGASetUp(iga);
```
PetIGA: basic usage

- Set boundary conditions

  ```c
  PetscInt axis, side;
  for (axis=0; axis<2; axis++)
    for (side=0; side<2; side++)
      IGASetBoundaryValue(iga, axis, side, 0, 0.0);
  ```

- Specify routine to evaluate at quadrature points.

  ```c
  IGASetFormSystem(iga, Poisson, NULL);
  ```
int Poisson(IGAPoint p, double K[], double F[], void *ctx) {
    int a, b, nen = p->nen;
    double *N0 = (typeof(N0)) p->shape[0];
    double (*N1)[2] = (typeof(N1)) p->shape[1];

    for (a=0; a<nen; a++) {
        for (b=0; b<nen; b++)
            K[a*nen+b] = N1[a][0]*N1[b][0] +
                         N1[a][1]*N1[b][1];
        F[a] = N0[a] * 1.0; // unit body force
    }
    return 0;
}
Get matrix and vectors from the IGA context

```c
Mat A;
IGACreateMat(iga,&A);
Vec x,b;
IGACreateVec(iga,&x);
IGACreateVec(iga,&b);
```

Assemble LHS matrix and RHS vector

```c
IGAComputeSystem(iga,A,b);
```
PetIGA: basic usage

- Create and setup linear solver

  ```c
  KSP  ksp;
  IGACreateKSP(iga,&ksp);
  KSPSetOperators(ksp,A,A);
  KSPSetFromOptions(ksp);
  ```

- Solve linear system

  ```c
  KSPSolve(ksp,b,x);
  ```

- Write geometry and solution to disk

  ```c
  IGAWrite(iga,"PoissonGeometry.dat");
  IGAWriteVec(iga,x,"PoissonSolution.dat");
  ```
$ mpiexec -n 4 ./CahnHilliard2D \ 
  -ts_type alpha -ts_max_time 0.1 \ 
  -snes_type ls -snes_rtol 1e-6 \ 
  -ksp_type gmres -ksp_rtol 1e-6 -pc_type asm \ 
  -ts_monitor -snes_monitor -ksp_monitor

...
Runtime monitoring solution evolution

PETSc has basic 2D visualization capabilities to aid debugging

$ mpiexec -n 4 ./NSK2D -ts_monitor_solution

Plot control variables on control mesh (no spline interpolation)
PETSc can approximate Jacobians

1. Code residual function (Gauss point evaluation)
2. Approximate Jacobian with \textit{finite differences}
   2.1 Use a \textbf{matrix free} approach (\texttt{-snes mf})
      \begin{itemize}
      \item Allows checking bug-free residual function implementation
      \item Potentially slow, no black-box preconditioner available
      \end{itemize}
   2.2 Use \textbf{colored finite differences} (\texttt{-snes_fd_color})
      \begin{itemize}
      \item Computes sparse matrix with approximate Jacobian explicitly
      \item This alternative allows black-box preconditioning
      \item Still slow for large problems
      \end{itemize}
Practical approach to nonlinear problems

PETSc can check hand-coded Jacobians

1. Code residual function (Gauss point evaluation)
2. Code Jacobian function (Gauss point evaluation)
3. Check Jacobian correctness using built-in facilities
   ▶ Approximate Jacobian with matrix-free (-snes_mf_operator)
   ▶ Invert computed Jacobian as a preconditioner (-pc_type lu)

If Jacobian is correct, KSP converges in one iteration
Geometry handling

Creation of initial (simple?) geometries is a nontrivial task

▶ Volume NURBS representations are cumbersome and are (mostly) developed manually
▶ Bridging the CAD/CAE gap is on going

Geometry handling made easier by:

1. Running in parametric mode if possible – avoid mapping cubes into cubes

2. Creating Python interfaces to low-level NURBS routines (knot insertion, degree elevation) – Python scripting is very flexible

3. Using binary files that PETSc reads/writes in parallel
   ▶ Sidesteps issue of parallel I/O
   ▶ No need to manually partition the domain
from igakit.cad import *

C0 = circle(radius=1)
C1 = circle(radius=2)

annulus = ruled(C0, C1)

pipe = extrude(annulus,
    displ=3.0, axis=2,
).reverse(2)

elbow = revolve(annulus,
    point=(3,0,0),
    axis=(0,-1,0),
    angle=Pi/2)

bentpipe = join(pipe, elbow, axis=2)
Solver Scalability

- Incompressible Navier–Stokes with VMS turbulence modeling
- 10 time steps × 2 Newton steps × 30 GMRES iterations
- B-spline space: \( p = 2, \ C^1 \); geometrical mapping: identity

### Parallel efficiency, single node (Lonestar, TACC)

<table>
<thead>
<tr>
<th>mesh</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>(32^3)</td>
<td>100%</td>
<td>98%</td>
<td>91%</td>
<td>84%</td>
<td>85%</td>
<td>77%</td>
<td>81%</td>
</tr>
<tr>
<td>(64^3)</td>
<td>100%</td>
<td>93%</td>
<td>85%</td>
<td>77%</td>
<td>79%</td>
<td>77%</td>
<td></td>
</tr>
</tbody>
</table>

### Parallel efficiency, multiple node (Lonestar, TACC)

<table>
<thead>
<tr>
<th>mesh</th>
<th>64</th>
<th>216</th>
<th>512</th>
<th>1000</th>
<th>1728</th>
<th>4104</th>
</tr>
</thead>
<tbody>
<tr>
<td>(120^3)</td>
<td>100%</td>
<td>102%</td>
<td>100%</td>
<td>97%</td>
<td>87%</td>
<td>74%</td>
</tr>
<tr>
<td>(168^3)</td>
<td>100%</td>
<td>100%</td>
<td>90%</td>
<td>91%</td>
<td>93%</td>
<td>74%</td>
</tr>
</tbody>
</table>
Applications: Elasticity
Applications: Navier-Stokes + VMS
Applications: Cahn-Hilliard equation
Applications: Navier-Stokes-Korteweg equation
Applications: Phase-Field Crystal equation
Conclusions

- Propose framework for solving problems using IGA
- Reuse PETSc data structures to simplify parallelism
- Acceptable scaling up to 4K processors
- Ongoing work: support for H-div/H-curl conforming spaces
- To do: extend to multipatch domains
Source code:

- https://bitbucket.org/dalcinl/petiga
- https://bitbucket.org/dalcinl/igakit

Tutorial (in development):

- https://petiga-igakit.readthedocs.org/

Questions & Comments:

- dalcinl@gmail.com
- nathaniel.collier@gmail.com
- adrimacortes@gmail.com