ViennaCL and PETSc Tutorial

Karl Rupp
rupp@mcs.anl.gov

Mathematics and Computer Science Division
Argonne National Laboratory

FEMTEC 2013

May 23th, 2013

Part 1

ViennaCL

Vienna Computing Library

http://viennacl.sourceforge.net/

Consider Existing CPU Code (Boost.uBLAS)

```
using namespace boost::numeric::ublas;


matrix<double> A(1000, 1000);
vector<double> x(1000), y(1000);

/* Fill A, x, y here */

double val = inner_prod(x, y);
y += 2.0 * x;
A += val * outer_prod(x, y);

x = solve(A, y, upper_tag()); // Upper tri. solver

std::cout << "  2-norm: " << norm_2(x) << std::endl;
std::cout << "sup-norm: " << norm_inf(x) << std::endl;
```

High-level code with syntactic sugar

Previous Code Snippet Rewritten with ViennaCL

```
using namespace viennacl;
using namespace viennacl::linalg;

matrix<double> A(1000, 1000);
vector<double> x(1000), y(1000);

/* Fill A, x, y here */

double val = inner_prod(x, y);
y += 2.0 * x;
A += val * outer_prod(x, y);

x = solve(A, y, upper_tag()); // Upper tri. solver

std::cout << "  2-norm: " << norm_2(x) << std::endl;
std::cout << "sup-norm: " << norm_inf(x) << std::endl;
```

High-level code with syntactic sugar

ViennaCL in Addition Provides Iterative Solvers

```
using namespace viennacl;
using namespace viennacl::linalg;

compressed_matrix<double> A(1000, 1000);
vector<double> x(1000), y(1000);

/* Fill A, x, y here */

x = solve(A, y, cg_tag());       // Conjugate Gradients
x = solve(A, y, bicgstab_tag()); // BiCGStab solver
x = solve(A, y, gmres_tag());    // GMRES solver
```

No Iterative Solvers Available in Boost.uBLAS...

From Boost.uBLAS to ViennaCL

Thanks to Interface Compatibility

```
using namespace boost::numeric::ublas;
using namespace viennacl::linalg;

compressed_matrix<double> A(1000, 1000);
vector<double> x(1000), y(1000);

/* Fill A, x, y here */

x = solve(A, y, cg_tag());       // Conjugate Gradients
x = solve(A, y, bicgstab_tag()); // BiCGStab solver
x = solve(A, y, gmres_tag());    // GMRES solver
```

Code Reuse Beyond GPU Borders

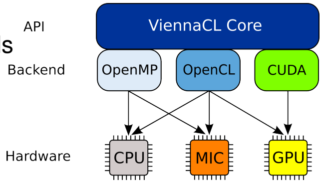
Eigen `http://eigen.tuxfamily.org/`

MTL 4 `http://www.mtl4.org/`

About ViennaCL

About

- High-level linear algebra C++ library
- OpenMP, OpenCL, and CUDA backends
- Header-only
- Multi-platform



Dissemination

- Free Open-Source MIT (X11) License
- http://viennacl.sourceforge.net/
- 50-100 downloads per week

Design Rules

- Reasonable default values
- Compatible to Boost.uBLAS whenever possible
- In doubt: clean design over performance

Basic Types

scalar, vector

matrix, compressed_matrix, coordinate_matrix, ell_matrix, hyb_matrix

Data Initialization

```
     std::vector<double>      std_x(100);
   ublas::vector<double>    ublas_x(100);
viennacl::vector<double>      vcl_x(100);

for (size_t i=0; i<100; ++i)
  //   std_x[i] = rand();  // (1)
  // ublas_x[i] = rand();  // (2)
       vcl_x[i] = rand();  // (3)
```

(3) is fastest, right?

Basic Types

scalar, vector

matrix, compressed_matrix, coordinate_matrix, ell_matrix, hyb_matrix

Data Initialization

Using viennacl::copy()

```
     std::vector<double>      std_x(100);
   ublas::vector<double>    ublas_x(100);
viennacl::vector<double>      vcl_x(100);

/* setup of std_x and ublas_x omitted */

viennacl::copy(std_x.begin(), std_x.end(),
               vcl_x.begin());   //to GPU
viennacl::copy(vcl_x.begin(), vcl_x.end(),
               ublas_x.begin()); //to CPU
```

Basic Types

scalar, vector

matrix, compressed_matrix, coordinate_matrix, ell_matrix, hyb_matrix

Data Initialization

Using viennacl::copy()

```
     std::vector<std::vector<double> >    std_A;
   ublas::matrix<double>                ublas_A;
viennacl::matrix<double>                  vcl_A;

/* setup of std_A and ublas_A omitted */

viennacl::copy(std_A, vcl_A);     // CPU to GPU
viennacl::copy(vcl_A, ublas_A);   // GPU to CPU
```

Iterator concept doesn't quite work on accelerators

Vector Addition

```
x = y + z;
```

Temporaries are costly (particularly on GPUs)

Expression Templates

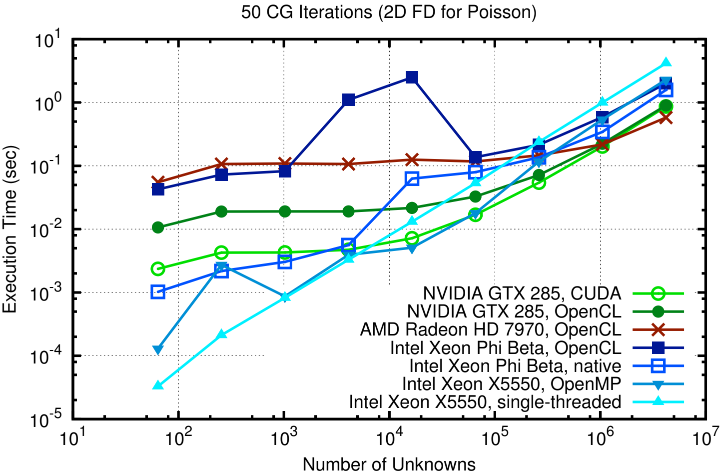
Limited expansion

Map to a set of predefined kernels

```
vector_expression<vector<T>, op_plus, vector<T> >
operator+(vector<T> & v, vector<T> & w) { ... }

vector::operator=(vector_expression<...> const & e) {
  viennacl::linalg::avbv(*this, 1,e.lhs(), 1,e.rhs());
}
```

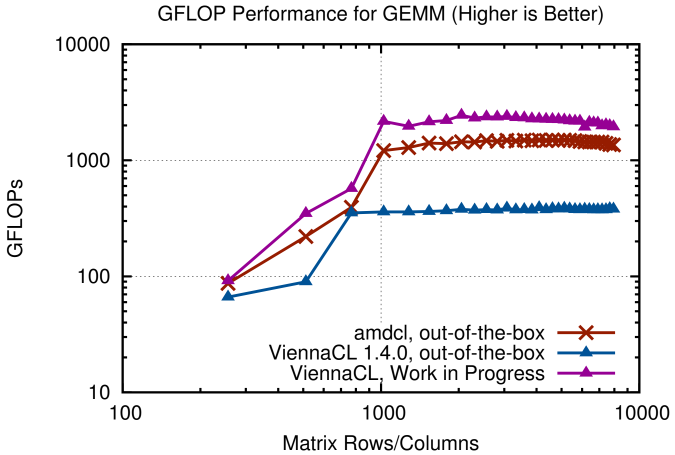
Benchmarks



50 CG Iterations (2D FD for Poisson)
Execution Time (sec)
Number of Unknowns
10^{1}
10^{0}
10^{-1}
10^{-2}
10^{-3}
10^{-4}
10^{-5}
10^{1}
10^{2}
10^{3}
10^{4}
10^{5}
10^{6}
10^{7}
NVIDIA GTX 285, CUDA
NVIDIA GTX 285, OpenCL
AMD Radeon HD 7970, OpenCL
Intel Xeon Phi Beta, OpenCL
Intel Xeon Phi Beta, native
Intel Xeon X5550, OpenMP
Intel Xeon X5550, single-threaded

Matrix-Matrix Multiplication

Autotuning environment



(AMD Radeon HD 7970, single precision)

Acknowledgements

Contributors

Thomas Bertani
Evan Bollig
Philipp Grabenweger
Volodymyr Kysenko
Nikolay Lukash
Günther Mader
Vittorio Patriarca
Florian Rudolf
Astrid Rupp
Philippe Tillet
Markus Wagner
Josef Weinbub
Michael Wild

Summary

High-Level C++ Approach of ViennaCL

- Convenience of single-threaded high-level libraries (Boost.uBLAS)
- Header-only library for simple integration into existing code
- MIT (X11) license

http://viennacl.sourceforge.net/

Selected Features

- Backends: OpenMP, OpenCL, CUDA
- Iterative Solvers: CG, BiCGStab, GMRES
- Preconditioners: AMG, SPAI, ILU, Jacobi
- BLAS: Levels 1-3

PETSc

Portable Extensible Toolkit for Scientific Computing

Obtaining PETSc

- Linux Package Managers
- Web: http://mcs.anl.gov/petsc, download tarball
- Git: https://bitbucket.org/petsc/petsc
- Mercurial: https://bitbucket.org/petsc/petsc-hg

Installing PETSc

```
$> cd /path/to/petsc/workdir
$> git clone \
     https://bitbucket.org/petsc/petsc.git \
     --branch master --depth 1
$> cd petsc
```

```
$> export PETSC_DIR=$PWD PETSC_ARCH=mpich-gcc-dbg
$> ./configure --with-cc=gcc --with-fc=gfortran
                 --download-f-blas-lapack
                 --download-{mpich,ml,hypre}
```

PETSc

Portable Extensible Toolkit for Scientific Computing

Architecture

- tightly coupled (e.g. XT5, BG/P, Earth Simulator)
- loosely coupled such as network of workstations
- GPU clusters (many vector and sparse matrix kernels)

Software Environment

- Operating systems (Linux, Mac, Windows, BSD, proprietary Unix)
- Any compiler
- Usable from C, C++, Fortran 77/90, Python, and MATLAB
- Real/complex, single/double/quad precision, 32/64-bit int

System Size

- 500B unknowns, 75% weak scalability on Jaguar (225k cores) and Jugene (295k cores)
- Same code runs performantly on a laptop

Free to everyone (BSD-style license), open development

Portable **Extensible** Toolkit for Scientific Computing

Philosophy: Everything has a plugin architecture

- Vectors, Matrices, Coloring/ordering/partitioning algorithms
- Preconditioners, Krylov accelerators
- Nonlinear solvers, Time integrators
- Spatial discretizations/topology

Example

- Vendor supplies matrix format and associated preconditioner, distributes compiled shared library.
- Application user loads plugin at runtime, no source code in sight.

Portable Extensible **Toolkit** for Scientific Computing

Toolset

- algorithms
- (parallel) debugging aids
- low-overhead profiling

Composability

- try new algorithms by choosing from product space
- composing existing algorithms (multilevel, domain decomposition, splitting)

Experimentation

- Impossible to pick the solver *a priori*
- PETSc's response: expose an algebra of composition
- keep solvers decoupled from physics and discretization

PETSc

Portable Extensible Toolkit for **Scientific Computing**

Computational Scientists

PyLith (CIG), Underworld (Monash), Magma Dynamics (LDEO, Columbia), PFLOTRAN (DOE), SHARP/UNIC (DOE)

Algorithm Developers (iterative methods and preconditioning)

Package Developers

SLEPc, TAO, Deal.II, Libmesh, FEniCS, PETSc-FEM, MagPar, OOFEM, FreeCFD, OpenFVM

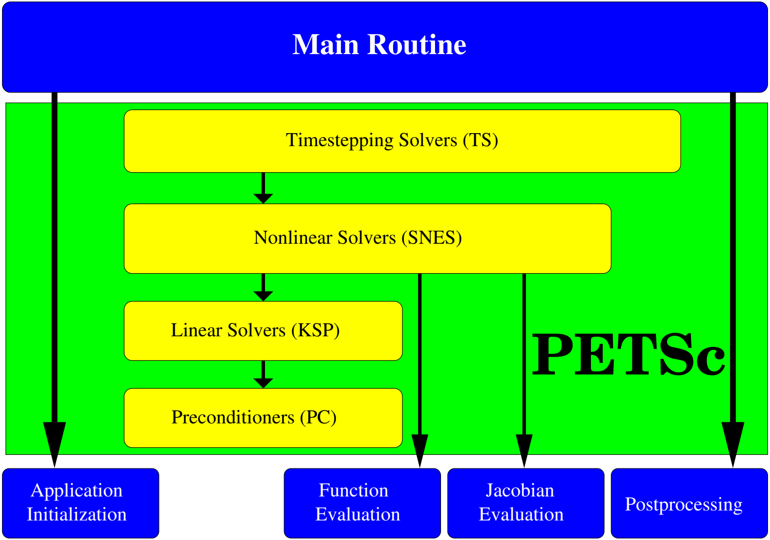
Funding

- Department of Energy
  - SciDAC, ASCR ISICLES, MICS Program, INL Reactor Program
- National Science Foundation
  - CIG, CISE, Multidisciplinary Challenge Program

Documentation and Support

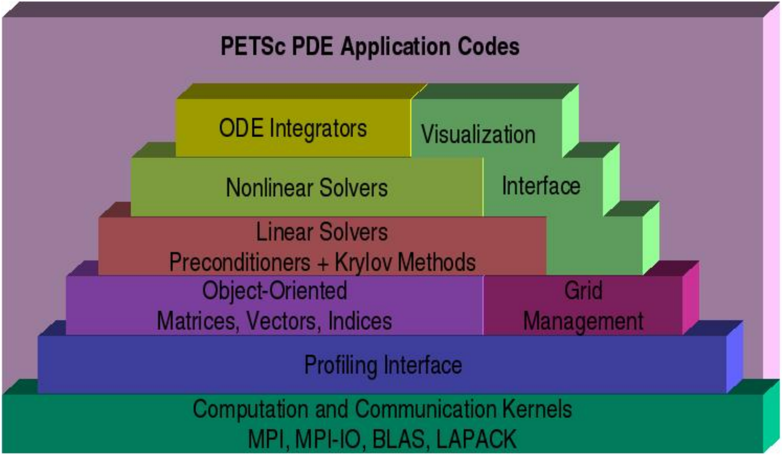
- Hundreds of tutorial-style examples
- Hyperlinked manual, examples, and manual pages for all routines
- Support from `petsc-maint@mcs.anl.gov`

Flow Control for a PETSc Application



PETSc Pyramid

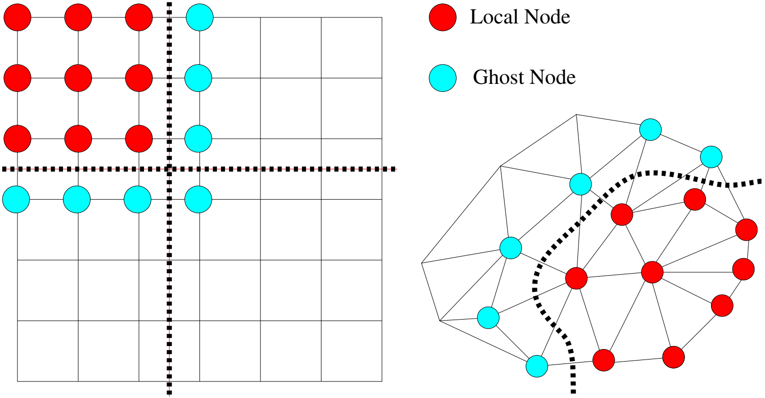
PETSc Structure



Ghost Values

To evaluate a local function $f(x)$, each process requires

- its local portion of the vector x
- its ghost values, bordering portions of x owned by neighboring processes



DMDA Global Numberings

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

DMDA Global vs. Local Numbering

Global: Each vertex has a unique id, belongs on a unique process

Local: Numbering includes vertices from neighboring processes

These are called ghost vertices

Proc 2			Proc 3	
X	X	X	X	X
X	X	X	X	X
12	13	14	15	X
8	9	10	11	X
4	5	6	7	X
0	1	2	3	X
Proc 0			Proc 1	

Local numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

Global numbering

Working with the Local Form

Wouldn't it be nice if we could just write our code for the natural numbering?

Proc 2			Proc 3	
25	26	27	28	29
20	21	22	23	24
15	16	17	18	19
10	11	12	13	14
5	6	7	8	9
0	1	2	3	4
Proc 0			Proc 1	

Natural numbering

Proc 2			Proc 3	
21	22	23	28	29
18	19	20	26	27
15	16	17	24	25
6	7	8	13	14
3	4	5	11	12
0	1	2	9	10
Proc 0			Proc 1	

PETSc numbering

Working with the Local Form

Wouldn't it be nice if we could just write our code for the natural numbering?

- Yes, that's what `DMDAVecGetArray()` is for.

DMDA offers *local* callback functions

- `FormFunctionLocal()`, set by `DMDASetLocalFunction()`
- `FormJacobianLocal()`, set by `DMDASetLocalJacobian()`

Evaluating the nonlinear residual $F(x)$

- Each process evaluates the local residual
- PETSc assembles the global residual automatically
  - Uses `DMLocalToGlobal()` method

$\mathfrak{p}$-Bratu Equation

2-dimensional model problem

$$-\nabla \cdot \left(|\nabla u|^{\mathfrak{p}-2}\nabla u\right) - \lambda e^u - f = 0, \qquad 1 \leq \mathfrak{p} \leq \infty, \quad \lambda < \lambda_{\text{crit}}(\mathfrak{p})$$

Singular or degenerate when $\nabla u = 0$, turning point at λ_{crit}.

$\mathfrak{p}$-Bratu Equation

2-dimensional model problem

$$-\nabla \cdot \left(|\nabla u|^{\mathfrak{p}-2} \nabla u \right) - \lambda e^u - f = 0, \qquad 1 \leq \mathfrak{p} \leq \infty, \quad \lambda < \lambda_{\text{crit}}(\mathfrak{p})$$

Singular or degenerate when $\nabla u = 0$, turning point at λ_{crit}.

Regularized Variant

Remove singularity of η using a parameter ε:

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

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

Physical interpretation: diffusivity tensor flattened in direction ∇u

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