Highly Productive Application Development with ViennaCL for Accelerators

K. Rupp, J. Weinbub, and F. Rudolf
1 Mathematics and Computer Science Division, Argonne National Laboratory
2 Institute for Microelectronics, Technische Universität Wien
rupp@mcs.anl.gov, {weinbub|rudolf}@iue.tuwien.ac.at
http://viennacl.sourceforge.net/

Motivation

• Common linear algebra operations appear naturally in geophysics.
• Both dense linear algebra and sparse linear algebra operations required.
• Large sparse linear equation systems are solved by iterative methods, which require preconditioners to obtain good convergence rates.
• Novel computing hardware is no longer single-threaded, but requires fine-grained parallelism in order to be used efficiently.
• Various programming models for parallelism: MPI, OpenMP, OpenCL, CUDA, etc.
• Library-centric software development required in order to cope with the increased programming complexity.

About ViennaCL

• Scientific computing library written in C++ (header-only).
• Multiple computing backends: Host-based, OpenCL, and CUDA.
• High-level programming interface, compatible with Boost.uBLAS.
• Focus on iterative solvers: CG, BiCGStab, GMRES.
• Various preconditioners: Jacobi, ICHOL, ILU0, ILUT, AMG, SPAI.
• BLAS level 1, 2, and 3 operations for dense linear algebra.
• MIT (X11) free open-source license.

API

<table>
<thead>
<tr>
<th>Hardware</th>
<th>Host-based</th>
<th>OpenCL</th>
<th>CUDA</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MIC</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Code Example


```cpp
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);

// Upper triangular solver: Ax = y
x = solve(A, y, upper_tag());

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

With ViennaCL: Only a change of namespaces required. GPU-accelerated.

```cpp
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);

// Upper triangular solver: Ax = y
x = solve(A, y, upper_tag());

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

Benchmark Results

• Out-of-the-box performance gain of high-level implementation by up to a factor three over LAPACK for QR factorizations.
• Up to 500 GFLOPs for single-precision matrix-matrix-multiplications on NVIDIA GTX 580.
• Up to ten-fold performance gain for iterative solvers.
• Memory-bandwidth is the limiting factor for iterative solvers.

Acknowledgments

• Hardware: NVIDIA for providing a Tesla 2050.
• Students: Google for providing students slots within the Google Summer of Code 2011 and 2012.
• Funding: Austrian Science Fund; European Research Council; Department of Energy ASCR SciDAC Project – Frameworks, Algorithms, and Scalable Technologies (FASTMath).