Computational Science on Many-Core Architectures

360.252

Karl Rupp

Institute for Microelectronics
Vienna University of Technology
http://www.iue.tuwien.ac.at

Zoom Channel 95028746244
Wednesday, December 9, 2020

Agenda for Today

Exercise 7 Recap

Libraries for GPUs

Exercise 8

Fine Friday Feast

Exercise 7 Recap

Kernel

- How was your experience?

Exercise 7 Recap

Kernel

- How was your experience?
- Points for Exercise 6 will be provided within 24 hours.

Exercise 7 Recap

Kernel

- How was your experience?
- Points for Exercise 6 will be provided within 24 hours.

```
STRINGIFY( CUCL_KERNEL void dotProduct(
  CUCL_GLOBALMEM double *x, CUCL_GLOBALMEM double *y,
  CUCL_GLOBALMEM double * partial_result, unsigned int N) {
  CUCL_LOCALMEM shared_buf[512]; double thread_sum =0;
  for (int i = CUCL_GLOBALID0;i<N;i += CUCL_GLOBALSIZE0)
    thread_sum += x[i]* y[i];

  shared_buf[CUCL_LOCALID0] = thread_sum;
  for (int stride = CUCL_LOCALSIZE0 / 2; stride>0; stride/=2) {
    CUCL_BARRIER;
    if ( CUCL_LOCALID0 < stride )
      shared_buf[CUCL_LOCALID0]+=shared_buf[CUCL_LOCALID0+stride];
  }

  CUCL_BARRIER;
  if (CUCL_LOCALID0 == 0)
    partial_result[ CUCL_GROUPID0] = shared_buf [0];
} )
```

Exercise 7 Recap

CUDA Use

```
#define STRINGIFY(ARG)  ARG
#define CUCL_KERNEL         __global__
#define CUCL_GLOBMEM
#define CUCL_GLOBALID0      blockDim.x * blockIdx.x + threadIdx.x
#define CUCL_GLOBALSIZE0    gridDim.x * blockDim.x
#include "dot.cucl"
```

Exercise 7 Recap

CUDA Use

```
#define STRINGIFY(ARG)  ARG
#define CUCL_KERNEL         __global__
#define CUCL_GLOBMEM
#define CUCL_GLOBALID0      blockDim.x * blockIdx.x + threadIdx.x
#define CUCL_GLOBALSIZE0    gridDim.x * blockDim.x
#include "dot.cucl"
```

OpenCL Use

```
 #define STRINGIFY(ANS)    #ANS
const char *opencl_kernel_sources =
"#define CUCL_KERNEL __kernel\n"
/* other preprocessor defines for the OpenCL sources here */
...
"#pragma OPENCL EXTENSION cl_khr_fp64 : enable \n"
#include "dot.cucl"
;
```

Libraries for GPUs

Fixed Function

- Libraries that expose a predefined set of routines
- Broad range of different applications covered, possibly well tuned
- Example: CUDA toolkit libraries, clSparse, MAGMA

Libraries for GPUs

Fixed Function

- Libraries that expose a predefined set of routines
- Broad range of different applications covered, possibly well tuned
- Example: CUDA toolkit libraries, clSparse, MAGMA

Variable Function

- Libraries that assist the user with application-specific operations
- Examples: Boost.Compute (OpenCL), VexCL (OpenCL), ViennaCL (CUDA, OpenCL), Thrust (CUDA)

Libraries for GPUs

Fixed Function

- Libraries that expose a predefined set of routines
- Broad range of different applications covered, possibly well tuned
- Example: CUDA toolkit libraries, clSparse, MAGMA

Variable Function

- Libraries that assist the user with application-specific operations
- Examples: Boost.Compute (OpenCL), VexCL (OpenCL), ViennaCL (CUDA, OpenCL), Thrust (CUDA)

Wrappers

- Provide GPU functionality in languages other than C or C++
- Examples: PyOpenCL, PyCUDA, gpuR, etc.

OpenCL Library Ecosystem

Abacus	CLFORTRAN	GPUVerify	OpenCloo	VisionSkelCL
ACML	clMAGMA	Halide	OpenCV-CL	SnuCL
Accelerate	clpp	Harlan	OpenHMPP	SpeedIT 2.4
amgCL	clSpMV	Haskell	Paralution	streamscan
Aparapi	CLTune	HOpenCL	Pardiso	SuperLU
AQUAgpusph	Clyther	JOCL	Pencil	s-u/OpenCL
ArrayFire	Concord	libCL	PETSc	TM-Task
ASL	COPRTHR	Libra SDK	PyOpenCL	Trilinos
Barracuda	Data Layout	Lua	RaijinCL	VexCL
Bolt	DelphiOpenCL	M3	Rivertrail	ViennaCL
Boost.Compute	ForOpenCL	MUMPS	RNG	ViNN
Bullet Physics	fortranCL	Octave	ROpenCL	VirtualCL
C++ AMP	FSCL.Compiler	OpenFortranP.	RoseACC-	VOBLA
CALDGEMM	GMAC	OpenCL.jl	OpenCL	VOCL
CF4OCL	Go-OpenCL	OpenCL.NET	Rose Compiler	VSI/Pro
clBLAS	GPULib	OpenCLIPP	Rust-OpenCL	WAMS
clFFT	gpumatrix	OpenCLLink	ScalaCL	

92 libraries listed on iwocl.org (as of 12/2020)

Selected GPU Libraries

A Brief Look at Four Selected GPU Libraries

- ViennaCL: CUDA, OpenCL, OpenMP
- Thrust: CUDA, OpenMP
- VexCL: OpenCL
- Boost.Compute: OpenCL

Selected GPU Libraries

A Brief Look at Four Selected GPU Libraries

- ViennaCL: CUDA, OpenCL, OpenMP
- Thrust: CUDA, OpenMP
- VexCL: OpenCL
- Boost.Compute: OpenCL

General Advice

- Avoid starting from scratch if possible
- Check application-specific GPU libraries
- NEVER implement dense matrix-matrix multiplication yourself!

  (Unless you are paid for doing it and no proper implementation is available)

From Boost.uBLAS to ViennaCL

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

From Boost.uBLAS to ViennaCL

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

From Boost.uBLAS to ViennaCL

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
```

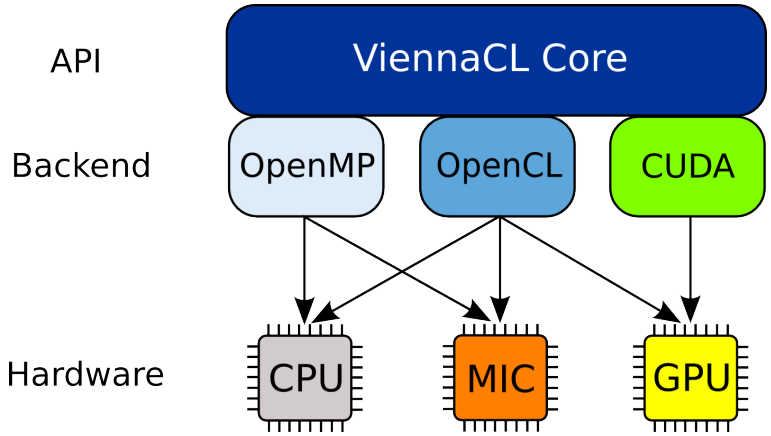
Code Reuse Beyond GPU Borders

- Armadillo `http://arma.sourceforge.net/`
- 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

About ViennaCL

Basic Types

- scalar
- vector
- matrix
- compressed_matrix, coordinate_matrix, (sliced_)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);


for (size_t i=0; i<100; ++i){
    std_x[i] = rand();
  ublas_x[i] = rand();
    vcl_x[i] = rand();  //possible, inefficient
}
```

About ViennaCL

Basic Types

- scalar
- vector
- matrix
- compressed_matrix, coordinate_matrix, (sliced_)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
```

About ViennaCL

Basic Types

- scalar
- vector
- matrix
- compressed_matrix, coordinate_matrix, (sliced_)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
```

Internals

Vector Addition

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

Naive Operator Overloading

```
vector<T> operator+(vector<T> & v, vector<T> & w);
```

- $t \leftarrow y + z$, $x \leftarrow t$
- Temporaries are extremely expensive!

Expression Templates

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

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

Internals

Vector Addition

```
// x = y + z
void avbv(...) {
  switch (active_handle_id(x))
  {
    case MAIN_MEMORY:
      host_based::avbv(...);
      break;
    case OPENCL_MEMORY:
      opencl::avbv(...);
      break;
    case CUDA_MEMORY:
      cuda::avbv(...);
      break;
    default:
      raise_error();
  }
}
```

- Memory buffers can switch memory domain at runtime

Generalizing Compute Kernels

```
  // x = y + z
  __kernel void avbv(
    double * x,

    double * y,

    double * z, uint size)
{
  i = get_global_id(0);
  for (size_t i=0; i<size; i += get_global_size(0))
    x[i] = y[i] + z[i];

}
```

Generalizing Compute Kernels

```
  // x = a * y + b * z
  __kernel void avbv(
    double * x,
    double a,
    double * y,
    double b,
    double * z, uint size)
{
  i = get_global_id(0);
  for (size_t i=0; i<size; i += get_global_size(0))
    x[i] = a * y[i] + b * z[i];

}
```

Generalizing Compute Kernels

```
  // x[4:8] = a * y[2:6] + b * z[3:7]
  __kernel void avbv(
    double * x, uint off_x,
    double a,
    double * y, uint off_y,
    double b,
    double * z, uint off_z, uint size)
{
  i = get_global_id(0);
  for (size_t i=0; i<size; i += get_global_size(0))
    x[off_x + i] = a * y[off_y + i] + b * z[off_z + i];

}
```

Generalizing Compute Kernels

```
  // x[4:2:8] = a * y[2:2:6] + b * z[3:2:7]
  __kernel void avbv(
    double * x, uint off_x, uint inc_x,
    double a,
    double * y, uint off_y, uint inc_y,
    double b,
    double * z, uint off_z, uint inc_z, uint size)
{
  i = get_global_id(0);
  for (size_t i=0; i<size; i += get_global_size(0))
    x[off_x + i * inc_x] =  a * y[off_y + i * inc_y]
                          + b * z[off_z + i * inc_z];
}
```

- No penalty on GPUs because FLOPs are for free

Thrust

About

- C++ Template Library
- Bundled with CUDA Toolkit

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sort.h>
#include <cstdlib>

int main(void) {
  // generate 32M random numbers on the host
  thrust::host_vector<int> h_vec(32 << 20);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer data to the device
  thrust::device_vector<int> d_vec = h_vec;
  // sort data on the device
  thrust::sort(d_vec.begin(), d_vec.end());
  // transfer data back to host
  thrust::copy(d_vec.begin(), d_vec.end(), h_vec.begin());

  return 0;
}
```

VexCL

About

- C++ Template Library on top of OpenCL
-

```
#include <iostream>
#include <stdexcept>
#include <vexcl/vexcl.hpp>

int main() {
    vex::Context ctx( vex::Filter::GPU && vex::Filter::
        DoublePrecision );

    if (!ctx) throw std::runtime_error("No devices available.");

    // Print out list of selected devices:
    std::cout << ctx << std::endl;

    ...
}
```

VexCL

About

- C++ Template Library on top of OpenCL
- Easy installation, good support

```
#include <iostream>
#include <stdexcept>
#include <vexcl/vexcl.hpp>

int main() {
 vex::Context ctx(vex::Filter::GPU&&vex::Filter::DoublePrecision);
  std::cout << ctx << std::endl; // print list of selected devices

 size_t N = 1000;
 std::vector<double> a(N, 1.0), b(N, 2.0);

 vex::vector<double> A(ctx, a);
 vex::vector<double> B(ctx, b);

 vex::vector<double> C = A + B;

 std::cout << C[0] << ", " << C[1] << ", ..." << std::endl;
 return 0; }
```

Boost.Compute

About

- C++ Template Library on top of OpenCL
- Part of the Boost Libraries (for better or worse)

```
// get default device and setup context
compute::device device = compute::system::default_device();
compute::context context(device);
compute::command_queue queue(context, device);

// generate random data on the host
std::vector<float> host_vector(10000);
std::generate(host_vector.begin(), host_vector.end(), rand);

// create a vector on the device
compute::vector<float> device_vector(host_vector.size(), context);

...
```

Boost.Compute

About

- C++ Template Library on top of OpenCL
- Part of the Boost Libraries (for better or worse)

```
...

// transfer data from the host to the device
compute::copy(host_vector.begin(), host_vector.end(),
              device_vector.begin(), queue);

// calculate the square-root of each element in-place
compute::transform(device_vector.begin(), device_vector.end(),
                   device_vector.begin(), compute::sqrt<float>(),
                   queue);

// copy values back to the host
compute::copy(device_vector.begin(), device_vector.end(),
              host_vector.begin(), queue);
```

Exercises

Environment

- `https://gtx1080.360252.org/2020/ex8/`
- (Might receive visual updates and additional hints over the next days)
- Due: Tuesday, December 15, 2020 at 23:59pm

Hints and Suggestions

- Consider version control for locally developed code
- Please let me know of any bugs or issues

Fine Friday Feast

Opportunity for Informal Chatting

- When? Friday, December 11, 17:00-18:00
- Where? ~~Wieden Bräu~~ This Zoom channel
- What? Preserving mental sanity during Christmas Shopping Spree

Hints and Suggestions

- Consider bringing a drink
- Will not change your course evaluation
- **Completely optional and no obligation to show up**