High-Level API for GPGPU using Meta-programming

Joel Falcou

University Paris-Sud LRI

December 15, 2015
The Hardware/Software Trade-Off

Single Core Era
- Performance
- Expressiveness
- C/Fort.
- C++
- Java

Multi-Core/SIMD Era
- Performance
- Expressiveness
- Sequential
- Threads
- SIMD

Heterogenous Era
- Performance
- Expressiveness
- Sequential
- SIMD
- Threads
- GPU
- Phi
- Distributed

2 of 23
The Hardware/Software Trade-Off

How to have performance AND expressiveness?
Designing tools for Scientific Computing

Challenges

1. Be non-disruptive
2. Domain driven optimizations
3. Provide intuitive API for the user
4. Support a wide architectural landscape
Designing tools for Scientific Computing

Challenges

1. Be non-disruptive
2. Domain driven optimizations
3. Provide intuitive API for the user
4. Support a wide architectural landscape

Our Approach

- Design tools as **C++ libraries** (1)
- Design these libraries as **Domain Specific Embedded Language (DSEL)** (2+3)
- Use **Generative Programming** to deliver performance (4)
Outline

Meta-Programming Tools

The NT$^2$ Library

Applications to Linear algebra

Conclusion
Outline

Meta-Programming Tools

The NT\(^2\) Library

Applications to Linear algebra

Conclusion
Generative Programming and DEMRAL

Domain Specific Application Description

Generative Component

Translator

Parametric Sub-components

Concrete Application
Generative Programming as a Tool

Available techniques

- Dedicated compilers
- External pre-processing tools
- Languages supporting meta-programming

Definition of Meta-programming

Meta-programming is the writing of computer programs that analyse, transform and generate other programs (or themselves) as their data.
Generative Programming as a Tool

Available techniques

- Dedicated compilers
- External pre-processing tools
- Languages supporting meta-programming
Generative Programming as a Tool

Available techniques

- Dedicated compilers
- External pre-processing tools
- Languages supporting meta-programming

Definition of Meta-programming

Meta-programming is the writing of computer programs that analyse, transform and generate other programs (or themselves) as their data.
From Generative to Meta-programming

Meta-programmable languages

- Template Haskell
- meta0caml
- C++
From Generative to Meta-programming

Meta-programmable languages

- Template Haskell
- meta0caml
- C++
From Generative to Meta-programming

Meta-programmable languages

- **template Haskell**
- meta0caml
- C++

C++ meta-programming

- Relies on the C++ `template` sub-language
- Handles `types` and `integral constants` at compile-time
- Proved to be Turing-complete
Domain Specific Embedded Languages

What’s a DSEL?
- DSL = Domain Specific Language
- Declarative language, easy-to-use, fitting the domain
- DSEL = DSL within a general purpose language

DSEL in C++
- Relies on operator overload abuse (Expression Templates)
- Carry semantic information around code fragment
- Generic implementation become self-aware of optimizations

Exploiting static AST
- At the expression level: code generation
- At the function level: inter-procedural optimization
Expression Templates

```cpp
matrix x(h,w), a(h,w), b(h,w);
x = cos(a) + (b*a);

expr<assign,
    expr<matrix&>,
    expr<plus,
        expr<cos,
            expr<matrix&>>, 
        expr<multiples,
            expr<matrix&>,
            expr<matrix&>> >
    >(x, a, b);

#pragma omp parallel for
for(int j=0; j<h; ++j)
{
    for(int i=0; i<w; ++i)
    {
        x(j,i) = cos(a(j,i))
            + (b(j,i)
            * a(j,i)
        )
    }
}
```

Arbitrary Transforms applied on the meta-AST
Outline

Meta-Programming Tools

The NT² Library

Applications to Linear algebra

Conclusion
NT²: The Numerical Template Toolbox

A Scientific Computing Library

- Provide a simple, MATLAB-like interface for users
- Provide high-performance computing entities and primitives
- Easily extendable
$NT^2$ : The Numerical Template Toolbox

A Scientific Computing Library

- Provide a simple, MATLAB-like interface for users
- Provide high-performance computing entities and primitives
- Easily extendable

Components

- Use Boost.SIMD for in-core optimizations
- Use recursive parallel skeletons for threading
- Code is made independent of architecture and runtime
MATLAB

A1 = 1:1000;
A2 = A1 + randn(size(A1));
X = lu(A1*A1');
rms = sqrt(sum((A1(:) - A2(:)).^2) / numel(A1));

NT²

table<double> A1 = _(1,1000);
table<double> A2 = A1 + randn(size(A1));
table<double> X = lu(mtimes(A1, trans(A1)));
table<double> rms = sqrt(sum(sqr(A1(_)-A2(_))) / numel(A1));
Outline

Meta-Programming Tools

The $NT^2$ Library

Applications to Linear algebra

Conclusion
Exploiting configuration spaces

Configuration space

- Based on iterative compilation techniques
- Combine the various configurations available in patterns
- Applications are assembled as combination of such patterns
Exploiting configuration spaces

Configuration space

- Based on iterative compilation techniques
- Combine the various configurations available in patterns
- Applications are assembled as combination of such patterns

Functionnal point of view

- Optimizations on binaries: Atlas,…
- Performance Analysis at runtime: StarPU,…
- Library level Exploration: NT2, Eigen, Armadillo,…
Configuration space

Patterns

- Identify statical properties of a linear system
- Combine them with hardware configuration
Configuration space

Patterns

- Identify statical properties of a linear system
- Combine them with hardware configuration

Configuration space parameters levels

<table>
<thead>
<tr>
<th>0 - Matrix type</th>
<th>general</th>
<th>band</th>
<th>diagonal</th>
<th>symmetric</th>
<th>positive definite</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 - Data type</td>
<td>float</td>
<td>double</td>
<td>single/double complex</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 - Precision</td>
<td>fixed</td>
<td>mixed-precision</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 - Conditioning</td>
<td>no information</td>
<td>ill-conditioned</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 - Storage scheme</td>
<td>general</td>
<td>packed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 - Architecture</td>
<td>CPU</td>
<td>GPU</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unified architectural model

Objectives

- Apply DSEL generation techniques for different kind of hardware
- High performance implementations on CPU and GPU
- Implementation of Linsolve in reference to MATLAB
Unified architectural model

Objectives

- Apply DSEL generation techniques for different kind of hardware
- High performance implementations on CPU and GPU
- Implementation of Linsolve in reference to MATLAB

Means

- Architecture aware binding between NT2 and LAPACK/MAGMA
- Memory container model for CPU and GPU memory
- Extend DEMRAL to AA-DEMRAL
Architecture Aware Generative Programming
Benefits to a generative solver

Overview

- Support to different factorization (QR, LU, SVD, Cholesky)
- There is no overhead compared to direct calls
- Prototyping a solver is not harder than with MATLAB

Mixed Precision Semi-Normal Equation Solver

- QR-based solver with least squares method
- Use of mixed-precision to improve performance
- No GPU implementation in the litterature
table<double> mcsne(table<double> const & A, table<double> const & B)
{
    // Compute the infinity norm of the system
    double anrm = lange(A,'I');
    double cte = anrm*Eps<double>()*nt2::sqrt(width(a));

    table<float> SA = cast<float>(A);

    table<float, upper_triangular_> SR = triu(qr(SA,no_pivot_));
    // The matrix SA is not actually transposed
    table<float> SX = mtimes(trans(SA),cast<float>(B));

    SX = linsolve(trans(SR),SX);
    SX = linsolve(SR    ,SX);

    table<double> X = cast<double>(SX);
    table<double> E  = B - mtimes(A,X);
}
// Apply iterative refinement steps to improve accuracy
do 
{
    SX = cast<float>(mtimes(trans(A),cast<float>(E)));
    SX = linsolve(trans(SR),SX);
    SX = linsolve(SR ,SX);

    E = cast<double>(SX)

double RNRM = maximum(abs(E(_)));

    X += E;
    double XNRM = maximum(abs(X(_)));

    E = B - mtimes(A,X);
    i++;
} while( !(RNRM < XNRM*cte) && (i<max_iter));
}
Mixed Precision Semi-Normal Equation Method

Performance - CPU

![Graph showing performance metrics for different methods and precisions. The x-axis represents the input size (10000/1000 to 20000/10000), and the y-axis represents Gflop/s (Giga floating-point operations per second). The legend includes different methods such as QR_single_prec, CSNE_mixed_prec, QR_double_prec, sgemm, plasma_sgemm, and CSNE_m_Armadillo and CSNE_m_Eigen.]
Mixed Precision Semi-Normal Equation Method

Performance - GPU

![Graph showing performance comparison between QR_single_prec, CSNE_mixed_prec, and QR_double_prec methods. The x-axis represents data sizes, and the y-axis represents Gflops/s.](image-url)
Outline

Meta-Programming Tools

The NT^2 Library

Applications to Linear algebra

Conclusion
Let’s round this up!

Parallel Computing for Scientist

- Software Libraries built as Generic and Generative components can solve a large chunk of parallelism related problems while being easy to use.
- Like regular language, a DSEL needs informations about the hardware system
- Integrating hardware descriptions as Generic components increases tools portability and re-targetability
Let’s round this up!

Parallel Computing for Scientist

- Software Libraries built as Generic and Generative components can solve a large chunk of parallelism related problems while being easy to use.
- Like regular language, a DSEL needs informations about the hardware system.
- Integrating hardware descriptions as Generic components increases tools portability and re-targetability.

More information

- Prototype for single source GPU support
- Toward a global generic approach to parallelism
- Looking for SYCL!
Thanks for your attention