The Numerical Template Toolbox
An Architecture-aware EDSL for Scientific Computing

Joel Falcou
Pierre Esterie Antoine Tran tan
Jean-Thierry Lapresté
Mathias Gaunard

LRI - University Paris Sud XI - CNRS
Institut Pascal - Université Clermont II
MetaScale SAS

DSLDI - 01/07/2013
Context

In Scientific Computing ... 

- there is Scientific
In Scientific Computing ...

- there is *Scientific*
  - Applications are domain driven
  - Users ≠ Developers
  - Users are reluctant to changes
In Scientific Computing ...

- there is Scientific
  - Applications are domain driven
  - Users ≠ Developers
  - Users are reluctant to changes

- there is Computing
Context

In Scientific Computing ...

- there is Scientific
  - Applications are domain driven
  - Users ≠ Developers
  - Users are reluctant to changes

- there is Computing
  - Computing requires performance ...
  - ... which implies architectures specific tuning
  - ... which requires expertise
  - ... which may or may not be available
Context

In Scientific Computing ...

- there is Scientific
  - Applications are domain driven
  - Users ≠ Developers
  - Users are reluctant to changes

- there is Computing
  - Computing requires performance ...
    - ... which implies architectures specific tuning
    - ... which requires expertise
    - ... which may or may not be available

The Problem

People *using* computers to do science want to do *science* first.
Talk Layout

Introduction

NT2

EDSL in C++

Architecture Aware EDSL

Conclusion
What’s NT²?

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

A Research Platform
- Simple framework to add new optimization schemes
- Test bench for EDSL development methodologies
- Test bench for Generic Programming in real life projects
The NT² API

Principles

- table<T,S> is a simple, multidimensional array object that exactly mimics MATLAB array behavior and functionalities
- 300+ functions usable directly either on table or on any scalar values as in MATLAB
The NT\textsuperscript{2} API

Principles

- \texttt{table<T,S>} is a simple, multidimensional array object that exactly mimics \texttt{MATLAB} array behavior and functionalities
- 300+ functions usable directly either on \texttt{table} or on any scalar values as in \texttt{MATLAB}

How does it works

- Take a \texttt{.m} file, copy to a \texttt{.cpp} file
The NT$^2$ API

Principles

- `table<T,S>` is a simple, multidimensional array object that exactly mimics MATLAB array behavior and functionalities
- 300+ functions usable directly either on `table` or on any scalar values as in MATLAB

How does it work

- Take a .m file, copy to a .cpp file
- Add `#include <nt2/nt2.hpp>` and do cosmetic changes
The NT² API

Principles

- `table<T,S>` is a simple, multidimensional array object that exactly mimics Matlab array behavior and functionalities
- 300+ functions usable directly either on table or on any scalar values as in Matlab

How does it work

- Take a .m file, copy to a .cpp file
- Add `#include <nt2/nt2.hpp>` and do cosmetic changes
- Compile the file and link with `libnt2.a`
MATLAB you said?

1. \( R = I(:, :, 1); \)
2. \( G = I(:, :, 2); \)
3. \( B = I(:, :, 3); \)
4. \( Y = \min(\text{abs}(0.299 \times R + 0.587 \times G + 0.114 \times B), 235); \)
5. \( U = \min(\text{abs}(-0.169 \times R - 0.331 \times G + 0.5 \times B), 240); \)
6. \( V = \min(\text{abs}(0.5 \times R - 0.419 \times G - 0.081 \times B), 240); \)
Now with NT²

```cpp
auto R = I(_, _, 1);
auto G = I(_, _, 2);
auto B = I(_, _, 3);
table<float> Y, U, V;

Y = min(abs(0.299*R+0.587*G+0.114*B), 235);
U = min(abs(-0.169*R-0.331*G+0.5*B), 240);
V = min(abs(0.5*R-0.419*G-0.081*B), 240);
```
Embedded Domain Specific Languages

EDSL in C++
- Relies on operator overload abuse
- Carry semantic information around code fragment
- Generic implementation become self-aware of optimizations

Advantages
- Allow introduction of DSLs without disrupting dev. chain
- Semantic defined as type informations means compile-time resolution
- Access to a large selection of runtime binding
Expression Templates

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

#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
What’s Boost.Proto

- EDSL for defining EDSLs in C++
- Generalize Expression Templates
- Easy way to define and test EDSL
- EDSL = some Grammar Rules + some Semantic Actions

Boost.Proto Benefits

- Fast development process
- Compiler SDK: Grammar + Semantic + Code Generation process
- Easily extensible through user-defined AST Transforms
Generative Programming

- Domain Specific Application Description
- Generative Component
- Concrete Application
- Translator
- Parametric Sub-components
Architecture Aware Generative Programming
Multipass EDSL Code Generation

Optimize

- AST Pattern Matching at expression construction
- E.g: \(a+b*c\) to gemm or fma, \(x = \text{inv}(a)*b\) to \(x = \text{linsolve}(a,b)\)
Multipass EDSL Code Generation

Optimize

- AST Pattern Matching at expression construction
- E.g: \( a+b*c \) to gemm or fma, \( x = \text{inv}(a)*b \) to \( x = \text{linsolve}(a,b) \)

Schedule

- Use AST node categorization to segment AST into forest
- Each sub-AST are tied to a Parallel Skeleton
- \( NT^2 \) skeletons can be hierarchically nested
Multipass EDSL Code Generation

Optimize

- AST Pattern Matching at expression construction
- E.g: \( a+b\times c \) to gemm or fma, \( x = \text{inv}(a)\times b \) to \( x = \text{linsolve}(a,b) \)

Schedule

- Use AST node categorization to segment AST into forest
- Each sub-AST are tied to a **Parallel Skeleton**
- \( NT^2 \) skeletons can be hierarchically nested

Run

- Generate code for each AST in the scheduled forest
- Potentially generate runtime calls for runtime optimization
- Classical EDSL code generation happens here on the correct hardware
NT2 before AADEMRAL

EDSL Core
- Code base was 11 KLOC
- 8KLOC was dedicated to the various Expression Template glue
- 3KLOC of actual smart stuff

Architectural support
- Altivec and SSE2 extension support
- Some vague pthread support
- Efforts were stagnant: adding a simple feature required multiple KLOC of code to change
NT2 after AADEMRAL

State of the code base
- Expression Template handling  1-2KLOC
- Skeleton handling : 1KLOC
- Actual features and function implementation : 10KLOC

Architectural support
- Support for everything SIMD from SSE to AVX, NEON, etc..
- Support for OpenMP, Intel TBB, openCL
- Time to complete new architecture support : 1 week to 1 month
# Some Performances

## RGB2YUV timing (in cycles/pixels)

<table>
<thead>
<tr>
<th>Size</th>
<th>128x128</th>
<th>256x256</th>
<th>512x512</th>
<th>1024x1024</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MATLAB 2010a (2 cores)</strong></td>
<td>85</td>
<td>89</td>
<td>97</td>
<td>102</td>
</tr>
<tr>
<td><strong>C (1 core)</strong></td>
<td>23.6</td>
<td>23.8</td>
<td>23.9</td>
<td>24.0</td>
</tr>
<tr>
<td><strong>NT² (1 core)</strong></td>
<td>22.3</td>
<td>22.4</td>
<td>22.2</td>
<td>24.1</td>
</tr>
<tr>
<td><strong>NT² OpenMP (2 cores)</strong></td>
<td>11.4</td>
<td>11.4</td>
<td>11.5</td>
<td>12.2</td>
</tr>
<tr>
<td><strong>NT² SIMD</strong></td>
<td>5.5</td>
<td>5.6</td>
<td>5.6</td>
<td>5.9</td>
</tr>
<tr>
<td><strong>NT² SIMD+OpenMP</strong></td>
<td>2.9</td>
<td>2.9</td>
<td>2.9</td>
<td>3.0</td>
</tr>
<tr>
<td><strong>NT² SIMD speed-up</strong></td>
<td>3.9</td>
<td>3.9</td>
<td>3.9</td>
<td>4.0</td>
</tr>
<tr>
<td><strong>NT² OpenMP speed-up</strong></td>
<td>1.92</td>
<td>1.96</td>
<td>1.98</td>
<td>1.95</td>
</tr>
<tr>
<td><strong>NT² vs MATLAB speed-up</strong></td>
<td>29.3</td>
<td>30.7</td>
<td>33.5</td>
<td>34</td>
</tr>
</tbody>
</table>
Some Performances

Linear System Resolution (in GFLOPS)

\[ nt2::tie(x,r) = \text{linsolve}(a,b) \]

<table>
<thead>
<tr>
<th>GFLOPS</th>
<th>C++(float)</th>
<th>NT(^2)(float)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAPACK GESV 1024*1024</td>
<td>85.3</td>
<td>83.1</td>
</tr>
<tr>
<td>LAPACK GESV 2048*2048</td>
<td>350.7</td>
<td>348.2</td>
</tr>
<tr>
<td>LAPACK GESV 12000*12000</td>
<td>735.7</td>
<td>738.1</td>
</tr>
<tr>
<td>MAGMA GESV 1024*1024</td>
<td>85.3</td>
<td>85.8</td>
</tr>
<tr>
<td>MAGMA GESV 2048*2048</td>
<td>235.8</td>
<td>238.6</td>
</tr>
<tr>
<td>MAGMA GESV 12000*12000</td>
<td>1297.8</td>
<td>1300.6</td>
</tr>
</tbody>
</table>
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, EDSL 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, EDSL needs informations about the hardware system
- Integrating hardware descriptions as Generic components increases tools portability and re-targetability

Recent activity

- Available at http://www.github.com/MetaScale/nt2
- A Matlab to NT2 compiler has been designed
- Prototype for single source GPU support
- Toward a global generic approach to parallelism
Thanks for your attention