Generic and Generative Programming for HPC

Joel Falcou

LRI - INRIA

04/06/2014

Context

In Scientific Computing ...

- there is *Scientific*
  - Applications are domain driven
  - Users $\neq$ 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.

The Problem – and how we want to solve it

The Facts

- The *”Library to bind them all”* doesn’t exist (*or we should have it already*)
- All those users want to take advantage of new architectures
- Few of them want to actually handle all the dirty work

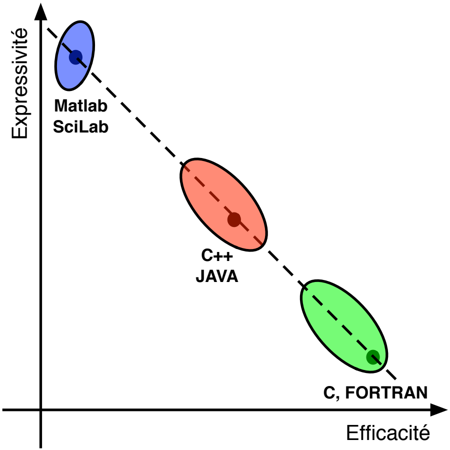
The Ends

- Provide a ”familiar” interface that let users benefit from parallelism
- Helps compilers to generate better parallel code
- Increase sustainability by decreasing amount of code to write

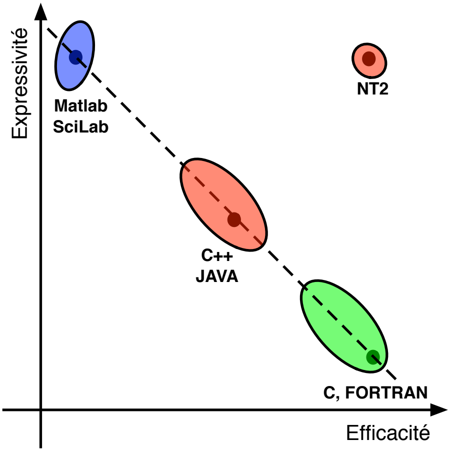
The Means

- Parallel Abstractions: Skeletons
- Efficient Implementation: DSEL
- The Holy Glue: Generative Programming

Efficient or Expressive – Choose one



Efficient or Expressive – Choose one



Talk Layout

Talk Layout

Introduction

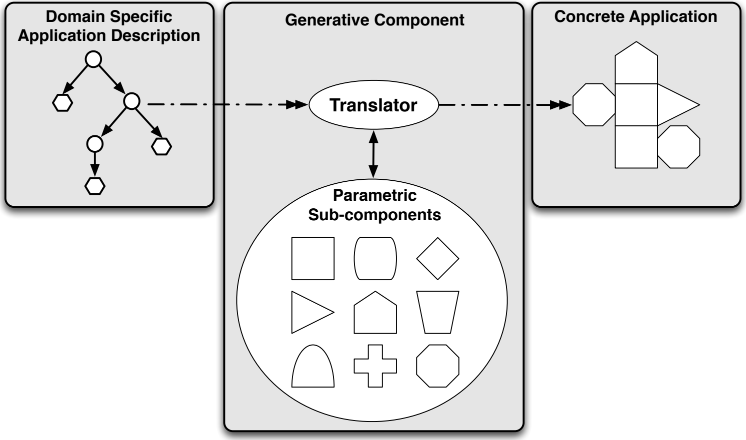
Efficiency

Abstractions

Tools

Conclusion

Generative Programming



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

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
- `metaOcaml`
- C++

From Generative to Meta-programming

Meta-programmable languages

- TEMPLATE HASKELL
- `metaOcaml`
- C++

From Generative to Meta-programming

Meta-programmable languages

- TEMPLATE HASKELL
- `metaOcaml`
- 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 an DSEL ?

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

EDSL 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

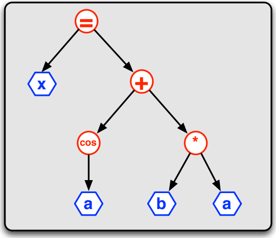
- At the expression level: code generation
- At the function level: inter-procedural optimization

Expression Templates

```
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<multiplies
               ,expr<matrix&>
               ,expr<matrix&>
               >
         >(x,a,b);
```

Arbitrary Transforms applied on the meta-AST




```
#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)
           );
  }
}
```

Embedded Domain Specific Languages

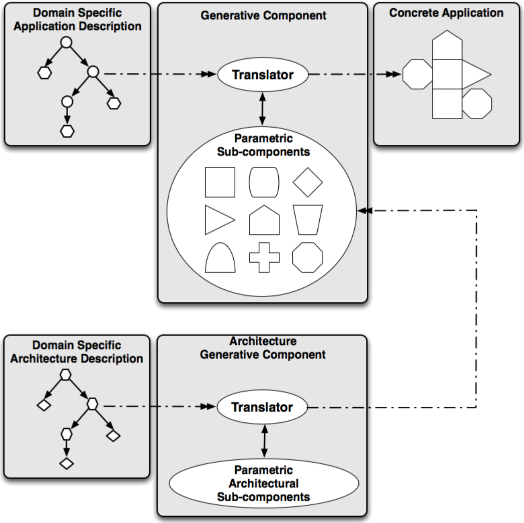
EDSL in C++

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

Advantages

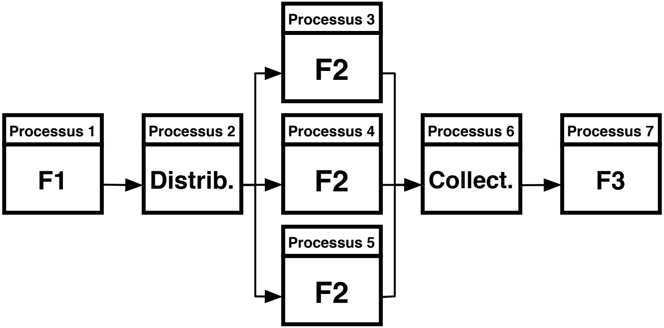
- Allow introduction of DSLs without disrupting dev. chain
- Semantic defined as type informations means compile-time resolution
- Access to a large selection of runtime binding

Architecture Aware Generative Programming

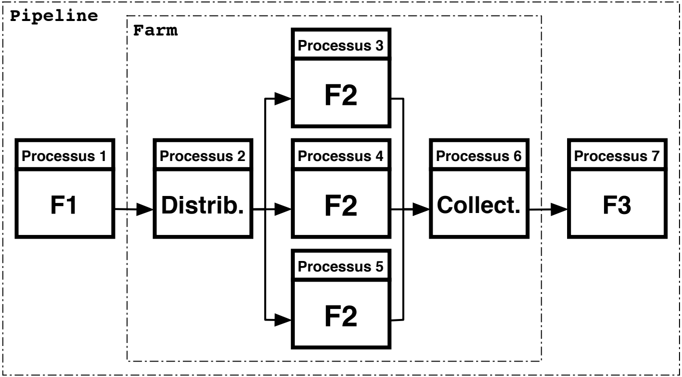


Talk Layout

Spotting abstraction when you see one



Spotting abstraction when you see one



Parallel Skeletons in a nutshell

Basic Principles [COLE 89]

- There are patterns in parallel applications
- Those patterns can be generalized in *Skeletons*
- Applications are assembled as combination of such patterns

Parallel Skeletons in a nutshell

Basic Principles [COLE 89]

- There are patterns in parallel applications
- Those patterns can be generalized in *Skeletons*
- Applications are assembled as combination of such patterns

Functionnal point of view

- Skeletons are *Higher-Order Functions*
- Skeletons support a compositionnal semantic
- Applications become composition of state-less functions

Classic Parallel Skeletons

Data Parallel Skeletons

- `map`: Apply a n-ary function in SIMD mode over subset of data
- `fold`: Perform n-ary reduction over subset of data
- `scan`: Perform n-ary prefix reduction over subset of data

Classic Parallel Skeletons

Data Parallel Skeletons

- `map`: Apply a n-ary function in SIMD mode over subset of data
- `fold`: Perform n-ary reduction over subset of data
- `scan`: Perform n-ary prefix reduction over subset of data

Task Parallel Skeletons

- `par`: Independant task execution
- `pipe`: Task dependency over time
- `farm`: Load-balancing

Why using Parallel Skeletons

Software Abstraction

- Write without bothering with parallel details
- Code is scalable and easy to maintain
- Debuggable, Provable, Certifiable

Why using Parallel Skeletons

Software Abstraction

- Write without bothering with parallel details
- Code is scalable and easy to maintain
- Debuggable, Provable, Certifiable

Hardware Abstraction

- Semantic is set, implementation is free
- Composability $\Rightarrow$ Hierarchical architecture

Talk Layout

Different Strokes

Objectives

- Apply DSEL generation techniques for different kind of hardware
- Demonstrate low cost of abstractions
- Demonstrate applicability of skeletons

NT^2

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 independant of architecture and runtime

The Numerical Template Toolbox

Comparison to other libraries

Feature	Armadillo	Blaze	Eigen	MTL	uBlas	NT^2
MATLAB-like API	✓	–	–	–	–	✓
BLAS/LAPACK binding	✓	✓	✓	✓	✓	✓
MAGMA binding	–	–	–	–	–	✓
SSE2+ support	✓	✓	✓	–	–	✓
AVX support	✓	✓	–	–	–	✓
AVX2 support	–	–	–	–	–	✓
Xeon Phi support	–	–	–	–	–	✓
Altivec support	–	–	✓	–	–	✓
ARM support	–	–	✓	–	–	✓
Threading support	–	–	–	–	–	✓
CUDA support	–	–	–	–	–	✓

The Numerical Template Toolbox

Principles

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

The Numerical Template Toolbox

Principles

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

How does it works

- Take a `.m` file, copy to a `.cpp` file

The Numerical Template Toolbox

Principles

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

How does it works

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

The Numerical Template Toolbox

Principles

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

How does it works

- 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`

NT2 - From MATLAB ...

```
A1   = 1:1000;
A2   = A1 + randn(size(A1));

X = lu(A1*A1');

rms = sqrt( sum(sqr(A1(:) - A2(:))) / numel(A1) );
```

NT2 - ... to C++

```
table<double> A1 = _(1.,1000.);
table<double> A2 = A1 + randn(size(A1));

table<double> X = lu( mtimes(A1, trans(A1) );

double rms = sqrt( sum(sqr(A1(_) - A2(_))) / numel(A1) );
```

Sigma-Delta Motion Detection

Context

- Mono-modal algorithm based on background substraction
- Use local gaussian model of lightness variation to detect motion
- Target applications: robotic, video survey and analytics, defence
- **Challenge**: Very low arithmetic density
- **Challenge**: Integer-based implementation with small range

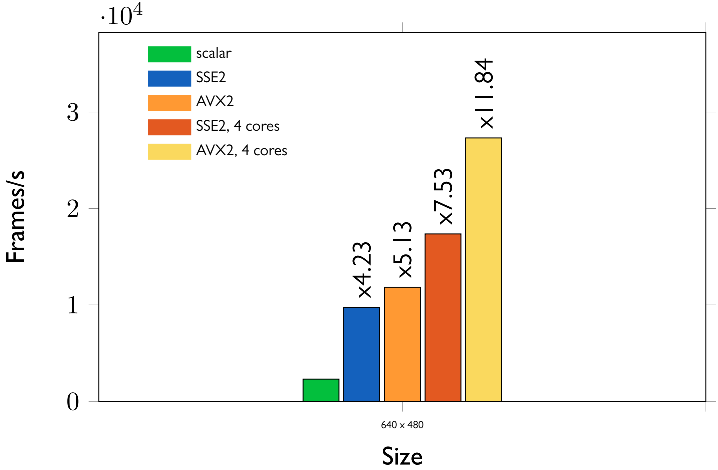
Motion Detection

NT^2 Code

```
table<char> sigma_delta( table<char>& background
                       , table<char> const& frame
                       , table<char>& variance
                       )
{
  // Estimate Raw Movement
  background = selinc( background < frame
                 , seldec(background > frame, background)
                 );

  table<char> diff = dist(background, frame);

  // Compute Local Variance
  table<char> sig3 = muls(diff,3);

  var = if_else( diff != 0
               , selinc( variance < sig3
                       , seldec( var > sig3, variance)
                       )
               , variance
               );

  // Generate Movement Label
  return if_zero_else_one( diff < variance );
}
```

Motion Detection

Performance



Context

- Mathematical model of a financial market containing certain derivative investment instruments.

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0$$

- Implementation of European-style option call and pricing
- Target applications: finance, insurance
- **Challenge**: Sensitive to data locality
- **Challenge**: Use complex statistical functions

Black and Scholes Option Pricing

NT^2 Code

```
table<float> blackscholes( table<float> const& Sa, table<float> const& Xa
                         , table<float> const& Ta
                         , table<float> const& ra, table<float> const& va
                         )
{
  table<float> da = sqrt(Ta);
  table<float> d1 = log(Sa/Xa) + (sqr(va)*0.5f+ra)*Ta/(va*da);
  table<float> d2 = d1-va*da;

  return Sa*normcdf(d1)- Xa*exp(-ra*Ta)*normcdf(d2);
}
```

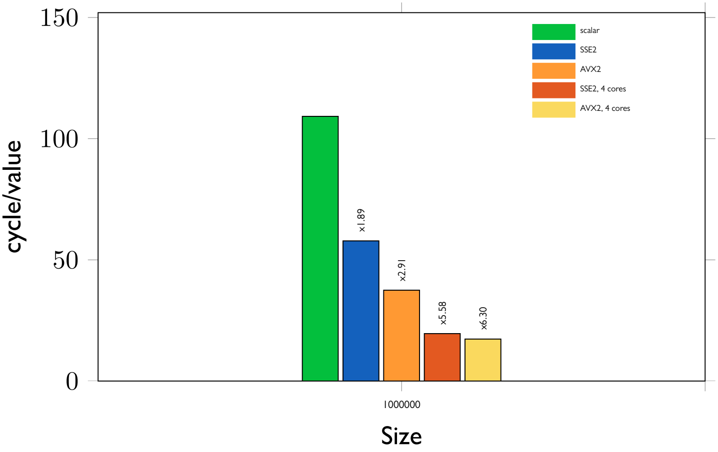
Black and Scholes Option Pricing

NT^2 Code with loop fusion

```
table<float> blackscholes( table<float> const& Sa, table<float> const& Xa
                         , table<float> const& Ta
                         , table<float> const& ra, table<float> const& va
                         )
{
  // Preallocate temporary tables
  table<float> da(extent(Ta)), d1(extent(Ta)), d2(extent(Ta)), R(extent(Ta));

  // tie merge loop nest and increase cache locality
  tie(da,d1,d2,R) = tie( sqrt(Ta)
                       , log(Sa/Xa) + (sqr(va)*0.5f+ra)*Ta/(va*da)
                       , d1-va*da
                       , Sa*normcdf(d1)- Xa*exp(-ra*Ta)*normcdf(d2)
                       );

  return R;
}
```

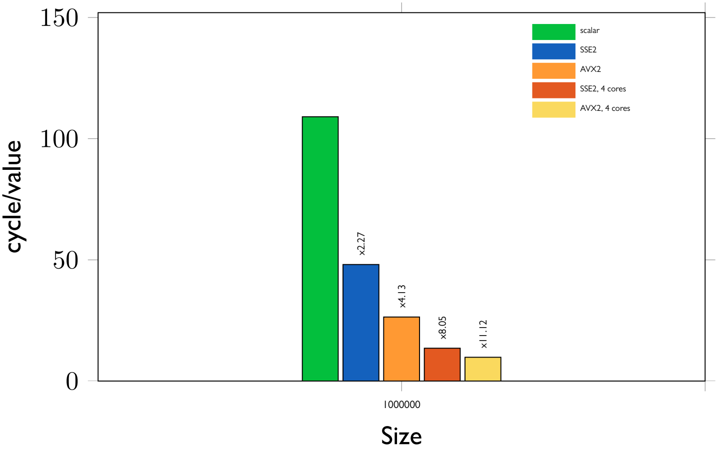
Black and Scholes Option Pricing

Performance



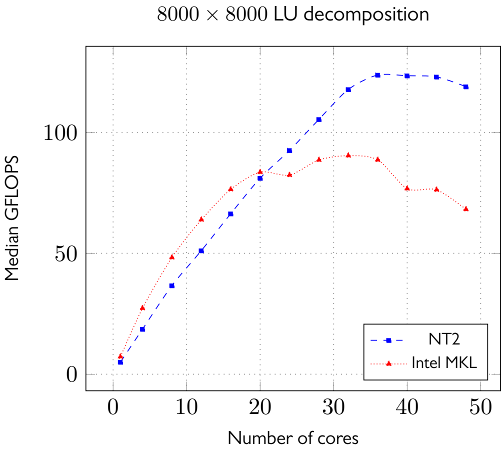
Black and Scholes Option Pricing

Performance with loop fusion



LU Decomposition

Performance



Talk Layout

Introduction

Efficiency

Abstractions

Tools

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, 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

- Follow us on `http://www.github.com/MetaScale/nt2`
- Prototype for single source GPU support
- Toward a global generic approach to parallelism

Thanks for your attention