Domain Specific Embedded Languages in C++

Contributions to HPC

Joel Falcou

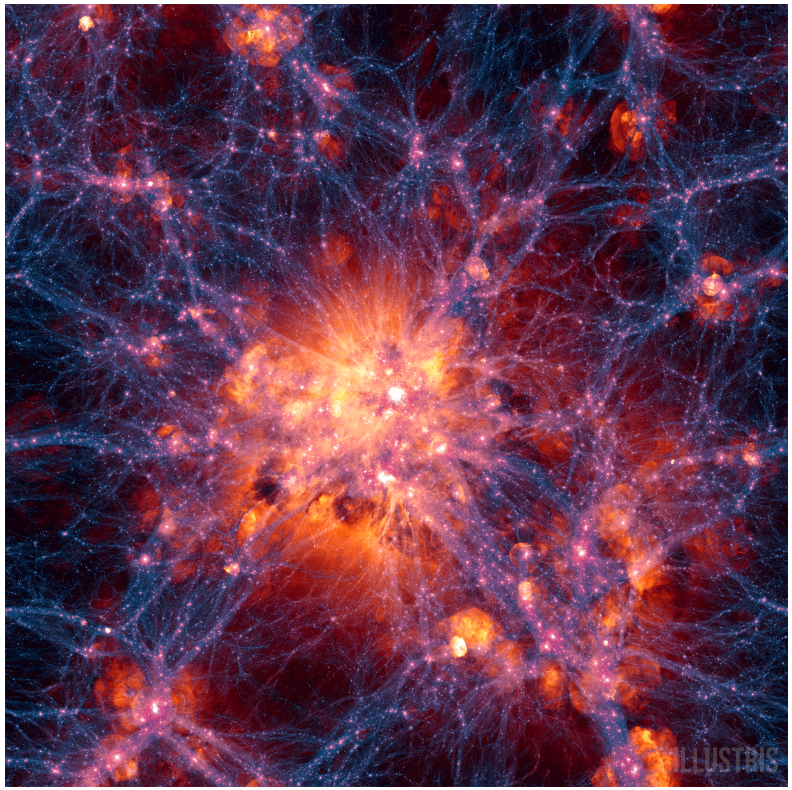
NumScale

May 27, 2016

The Paradigm Change in Science

From Experiments to Simulations

- Simulations is now an integral part of the *Scientific Method*
- Scientific Computing enables larger, faster, more accurate Research
- **Fast Simulation is Time Travel** as scientific results are now more readily available



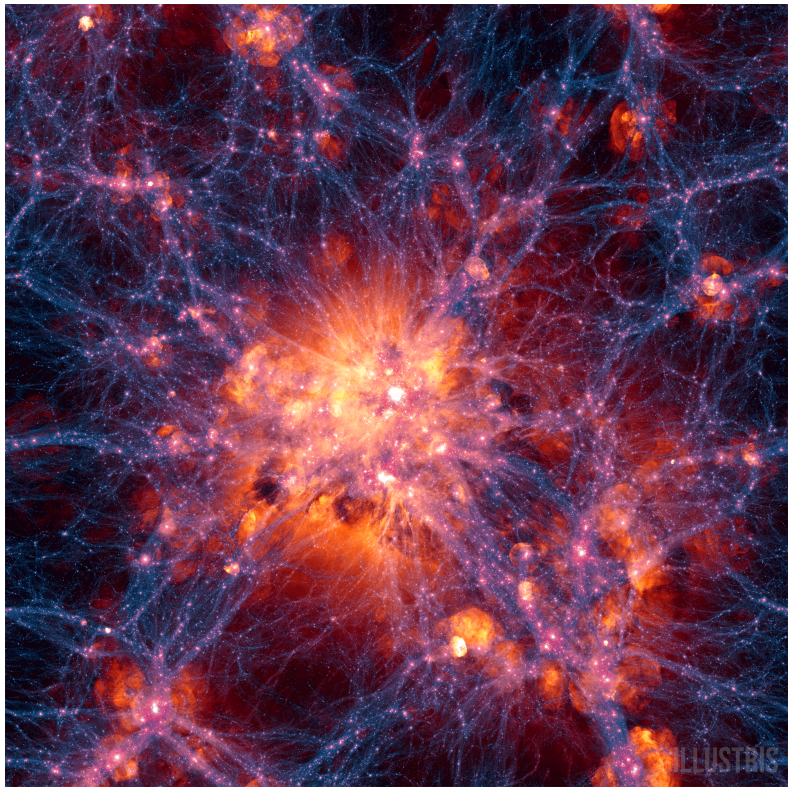
Local Galaxy Cluster Simulation - Illustris project

Computing is first and foremost a mainstream science tool

The Paradigm Change in Science

The Parallel Hell

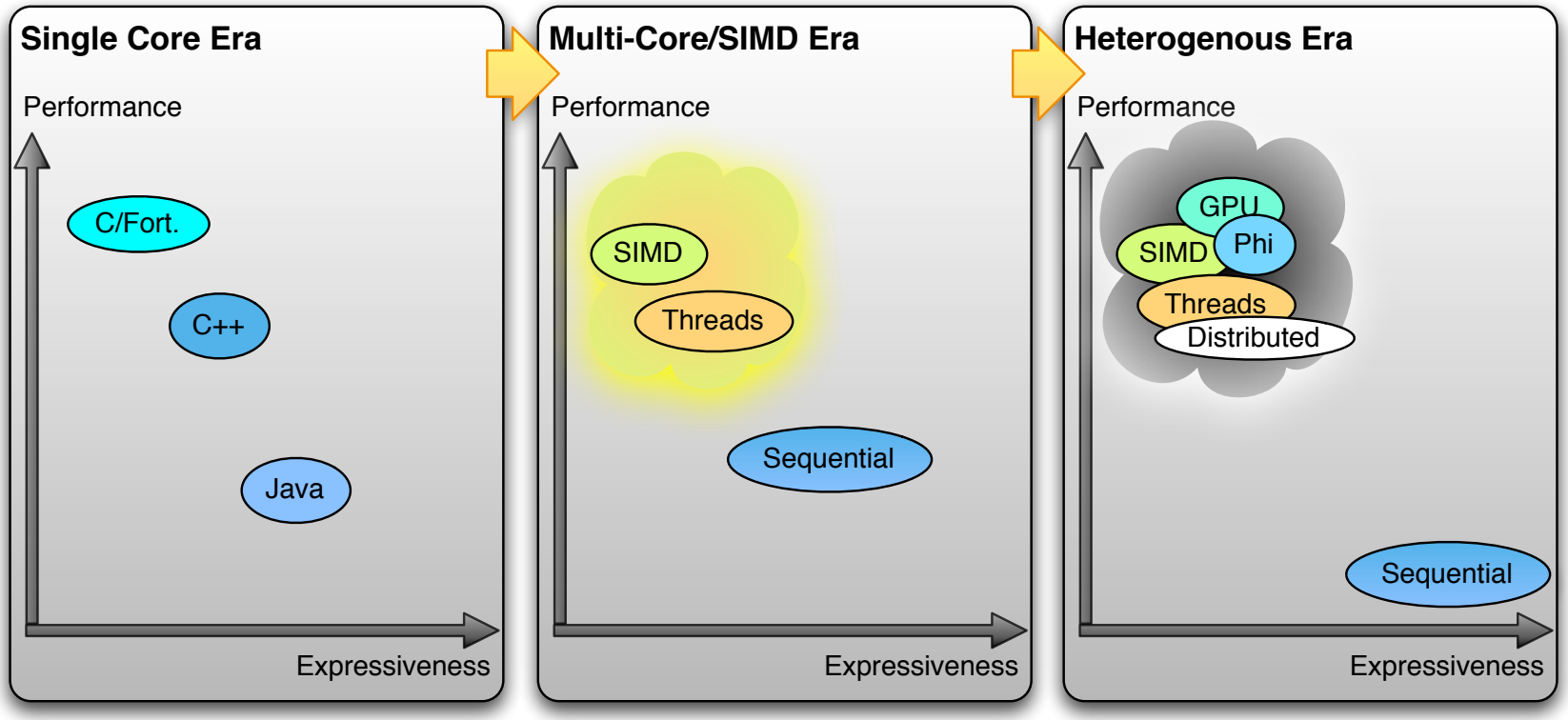
- Heat Wall: Growing cores instead of GHz
- Hierarchical and heterogeneous parallel systems are the norm
- *The Free Lunch* is over as hardware complexity rises faster than the average developer skills



Local Galaxy Cluster Simulation - Illustris project

The real challenge in HPC is the Expressiveness/Efficiency War

The Expressiveness/Efficiency War



As parallel systems complexity grows, the expressiveness gap turns into an ocean

Designing tools for Scientific Computing

Objectives

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

Designing tools for Scientific Computing

Objectives

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

Our Approach

- Design tools as C++ libraries (1)
- Design these libraries as Domain Specific Embedded Languages (DSEL) (2+3)
- Use Parallel Programming Abstractions as parallel components (4)
- Use Generative Programming to deliver performance (5)

Talk Layout

Why Parallel Programming Models ?

Limits of regular tools

- Unstructured parallelism is error-prone
- Low level parallel tools are non-composable
- Contribute to the Expressiveness Gap

Why Parallel Programming Models ?

Limits of regular tools

- Unstructured parallelism is error-prone
- Low level parallel tools are non-composable
- Contribute to the Expressiveness Gap

Available Models

- Performance centric: P-RAM, LOG-P, BSP
- Pattern centric: Futures, Skeletons
- Data centric: HTA, PGAS

Why Parallel Programming Models ?

Limits of regular tools

- Unstructured parallelism is error-prone
- Low level parallel tools are non-composable
- Contribute to the Expressiveness Gap

Available Models

- Performance centric: P-RAM, LOG-P, BSP
- Pattern centric: Futures, Skeletons
- Data centric: HTA, PGAS

Parallel Skeletons [Cole 89]

Principles

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

Functional point of view

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

Parallel Skeletons [Cole 89]

Principles

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

Classical Skeletons

- Data parallel: `map`, `fold`, `scan`
- Task parallel: `par`, `pipe`, `farm`
- More complex: Distribuable Homomorphism, Divide & Conquer, …

Domain Specific Embedded Languages

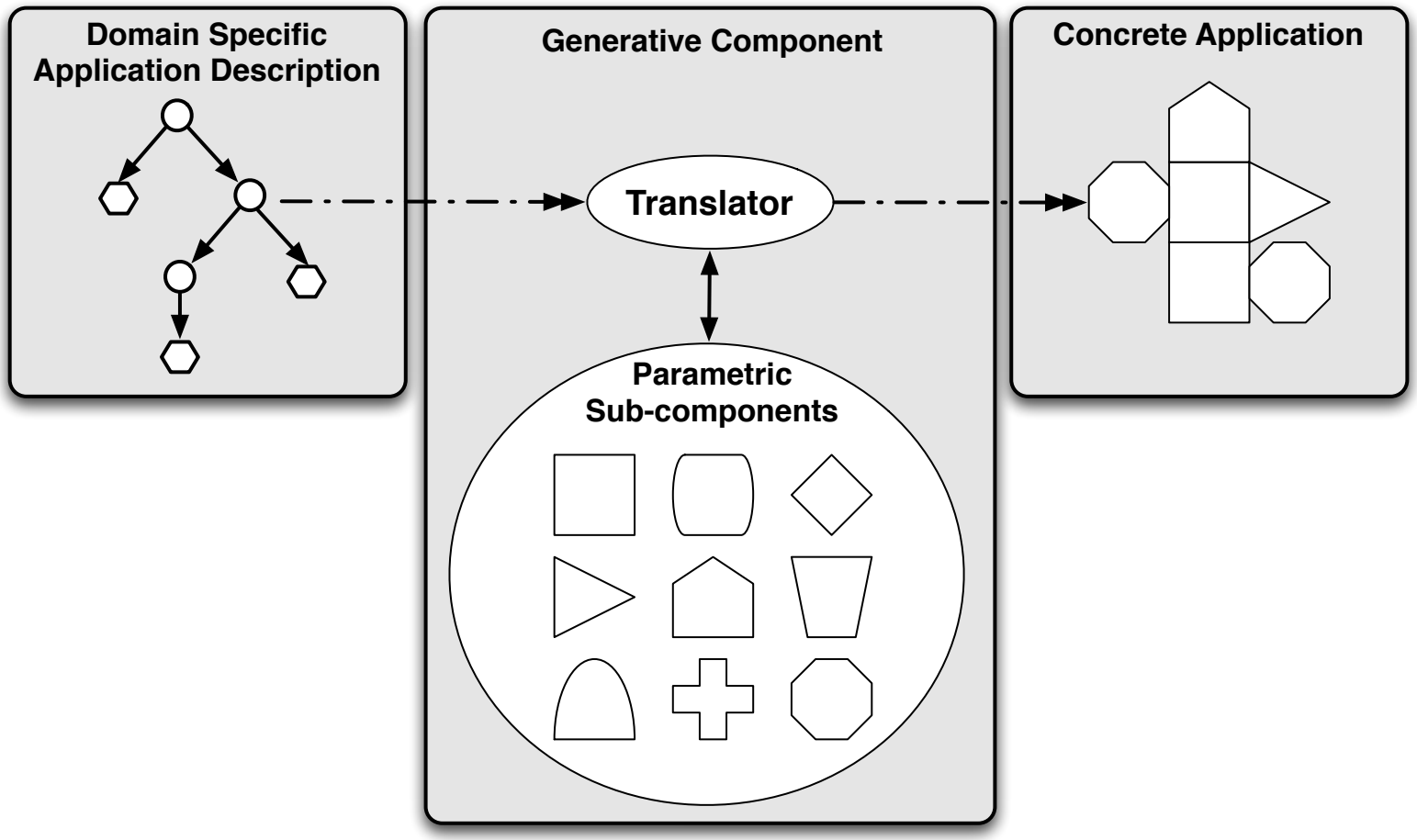
Domain Specific Languages

- Non-Turing complete declarative languages
- Solve a single type of problems
- Express what to do instead of how to do it
- E.g: SQL, MAKE, MATLAB, …

From DSL to DSEL [Abrahams 2004]

- A DSL incorporates domain-specific notation, constructs, and abstractions as fundamental design considerations.
- A Domain Specific Embedded Languages (DSEL) is simply a library that meets the same criteria
- Generative Programming is one way to design such libraries

Generative Programming [Eisenecker 97]



Meta-programming as a Tool

Definition

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

Meta-programmable Languages

- metaOCAML : runtime code generation via code quoting
- Template Haskell : compile-time code generation via templates
- C++ : compile-time code generation via templates

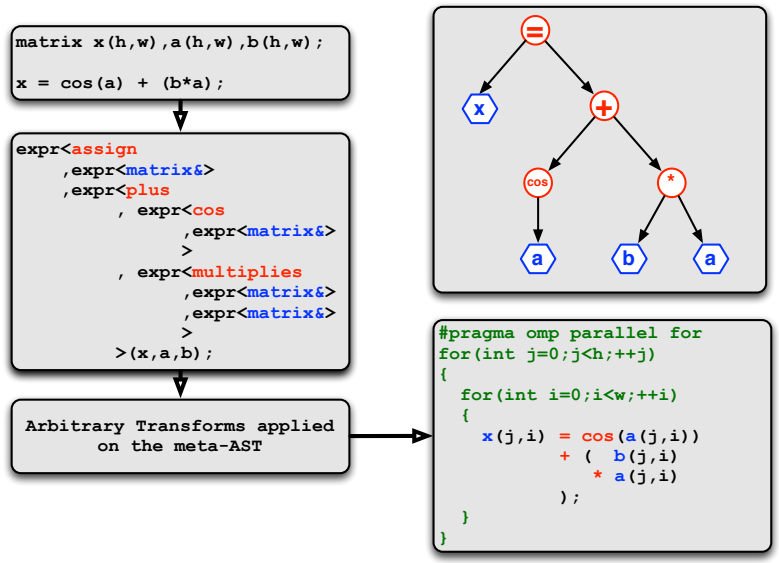
C++ meta-programming

- Relies on the Turing-complete C++ TEMPLATE sub-language
- Handles types and integral constants at compile-time
- TEMPLATE classes and functions act as code quoting

The Expression Templates Idiom

Principles

- Relies on extensive operator overloading
- Carries semantic information around code fragment
- Introduces DSLs without disrupting dev. chain

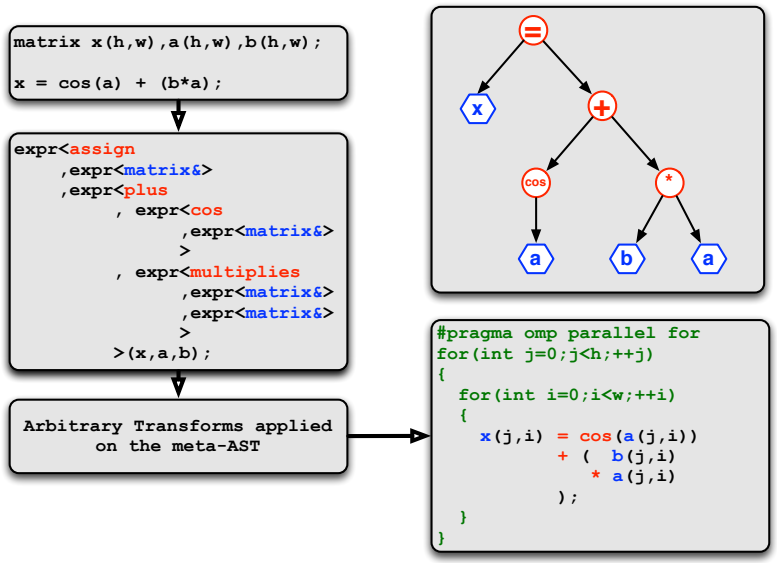


General Principles of Expression Templates

The Expression Templates Idiom

Advantages

- Generic implementation becomes self-aware of optimizations
- API abstraction level is arbitrary high
- Accessible through high-level tools like Boost.Proto



General Principles of Expression Templates

Our Contributions

Our Strategy

- Applies DSEL generation techniques to parallel programming
- Maintains low cost of abstractions through meta-programming
- Maintains abstraction level via modern library design

Our contributions

Tools	Pub.	Scope	Applications
Quaff	ParCo'06	MPI Skeletons	Real-time 3D reconstruction
SkelIBE	PACT'08	Skeleton on Cell BE	Real-time Image processing
BSP++	IJPP'12	MPI/OpenMP BSP	Bioinformatics, Model Checking
NT^2	JPDC'14	Data Parallel Matlab	Fluid Dynamics, Vision

Second Look at our Contributions

Development Limitations

- DSELs are mostly tied to the domain model
- Architecture support is often an afterthought
- Extensibility is difficult as many refactoring are required per architecture
- Example : No proper way to support GPUs with those implementation techniques

Second Look at our Contributions

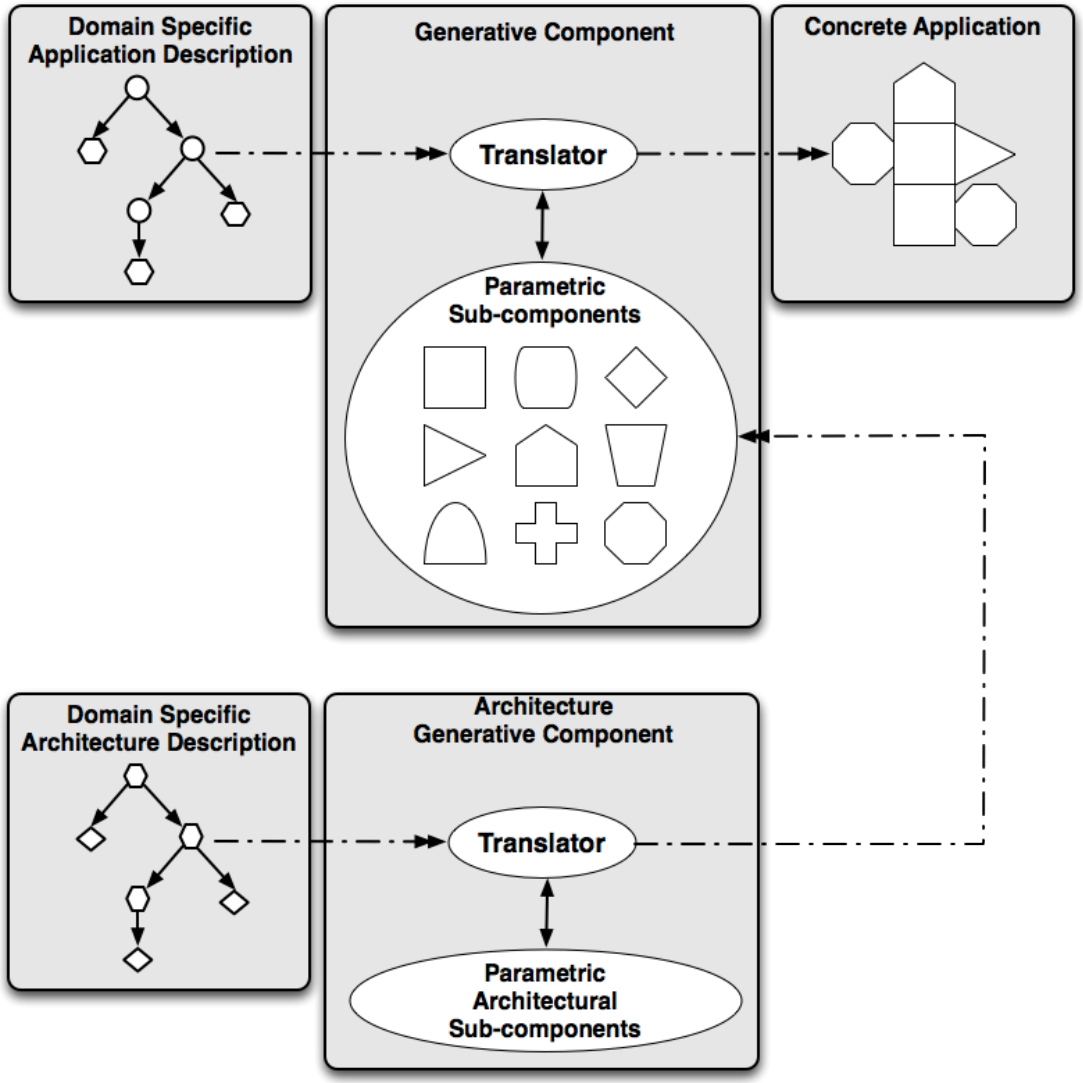
Development Limitations

- DSELs are mostly tied to the domain model
- Architecture support is often an afterthought
- Extensibility is difficult as many refactoring are required per architecture
- Example : No proper way to support GPUs with those implementation techniques

Proposed Method

- Extends Generative Programming to take this architecture into account
- Provides an architecture description DSEL
- Integrates this description in the code generation process

Architecture Aware Generative Programming



Software refactoring

Tools	Issues	Changes
~~Quaff~~	Raw skeletons API	Re-engineered as part of NT^2
~~SkellBE~~	Too architecture specific	Re-engineered as part of NT^2
BSP++	Integration issues	Integrate hybrid code generation
NT^2	Not easily extendable	Integrate Quaff Skeleton models
Boost.SIMD	-	Side product of NT^2 restructuration

Conclusion

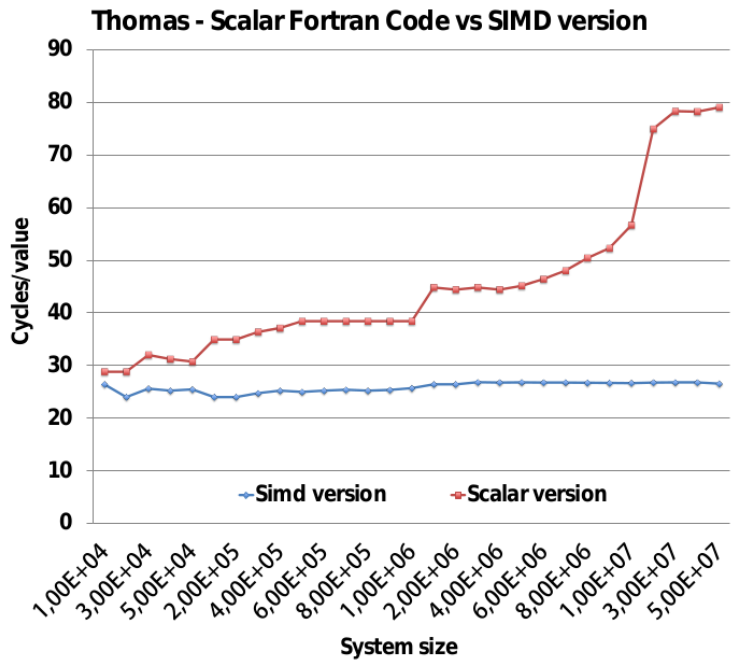
- Skeletons are fine as parallel middleware
- Model based abstractions are not high level enough
- For low level architectures, the simplest model is often the best

Boost.SIMD

Pierre Estérie PHD 2010-2014

Principles

- Provides simple C++ API over SIMD extensions
- Supports every Intel and PPC instructions sets
- Fully integrates with modern C++ idioms



Sparse Tridiagonal Solver - collaboration with M. Baboulin and Y. wang

Talk Layout

The Numerical Template Toolbox

Pierre Estérie PHD 2010-2014

NT^2 as a Scientific Computing Library

- Provides a simple, MATLAB-like interface for users
- Provides high-performance computing entities and primitives
- Is easily extendable

Components

- Uses Boost.SIMD for in-core optimizations
- Uses recursive parallel skeletons
- Supports task parallelism through Futures

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 to C++

MATLAB code

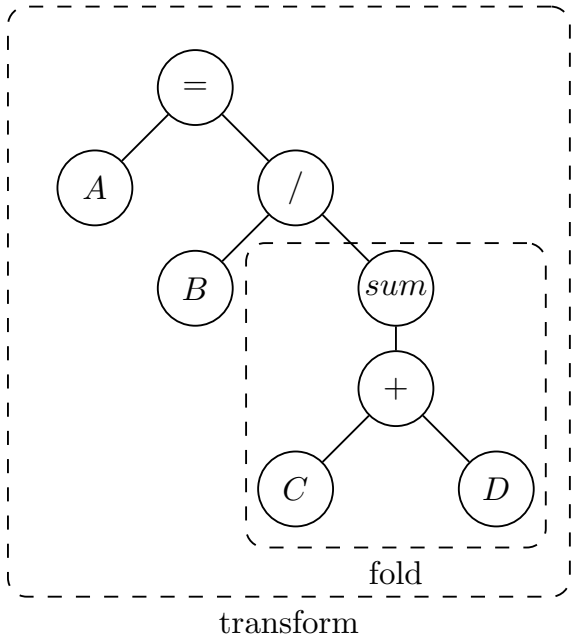
```
A1  = 1:1000;
A2  = A1 + randn(size(A1));
X = lu(A1*A1');

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

NT^2 code

```
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) );
```

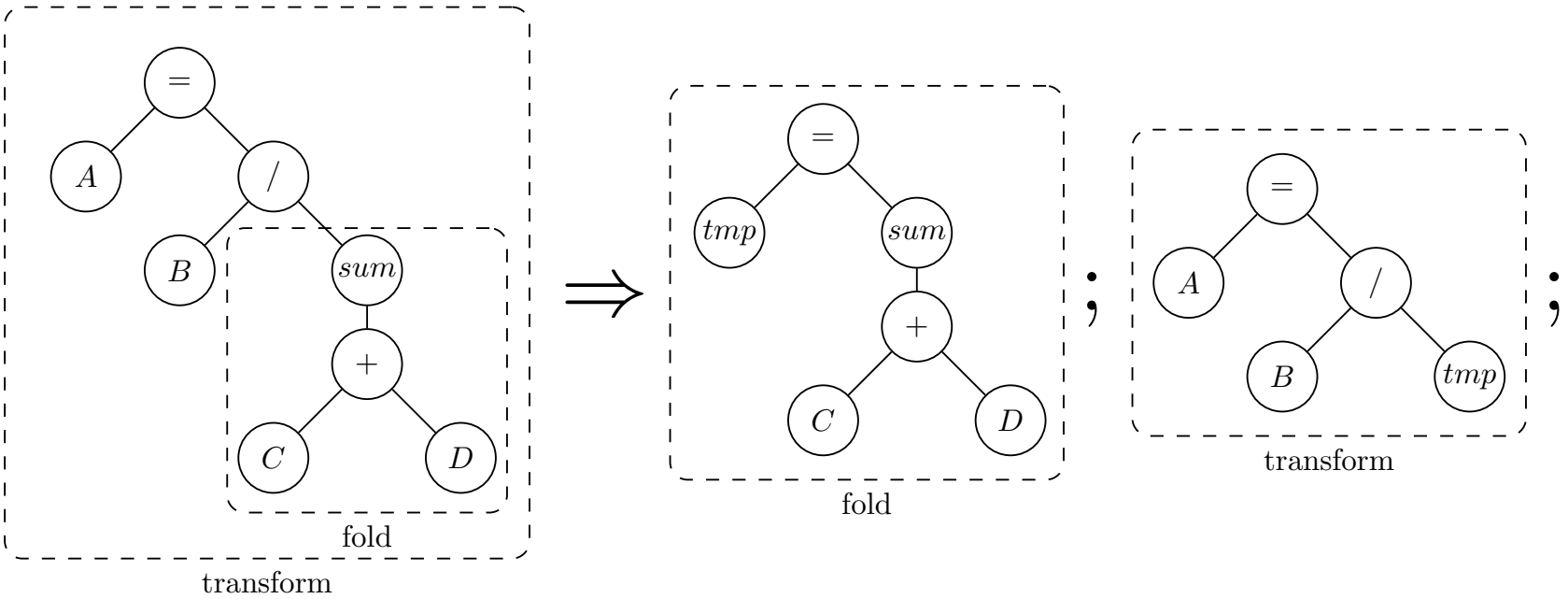
Parallel Skeletons extraction process

```
A = B / sum(C+D);
```



Parallel Skeletons extraction process

```
A = B / sum(C+D);
```



From data to task parallelism

Antoine Tran Tan PHD, 2012-2015

Limits of the fork-join model

- Synchronization cost due to implicit barriers
- Under-exploitation of potential parallelism
- Poor data locality and no inter-statement optimization

From data to task parallelism

Antoine Tran Tan PHD, 2012-2015

Limits of the fork-join model

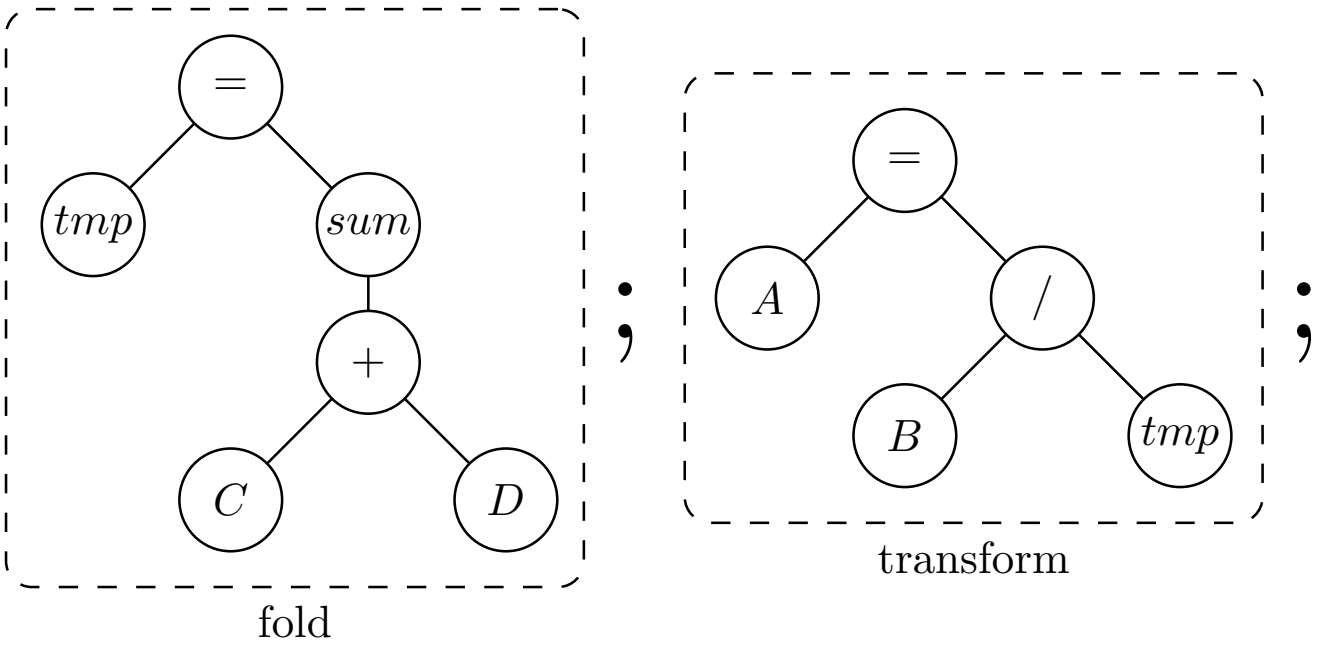
- Synchronization cost due to implicit barriers
- Under-exploitation of potential parallelism
- Poor data locality and no inter-statement optimization

Skeletons from the Future

- Adapt current skeletons for taskification
- Use Futures (STD or HPX) to automatically pipeline
- Derive a dependency graph between statements

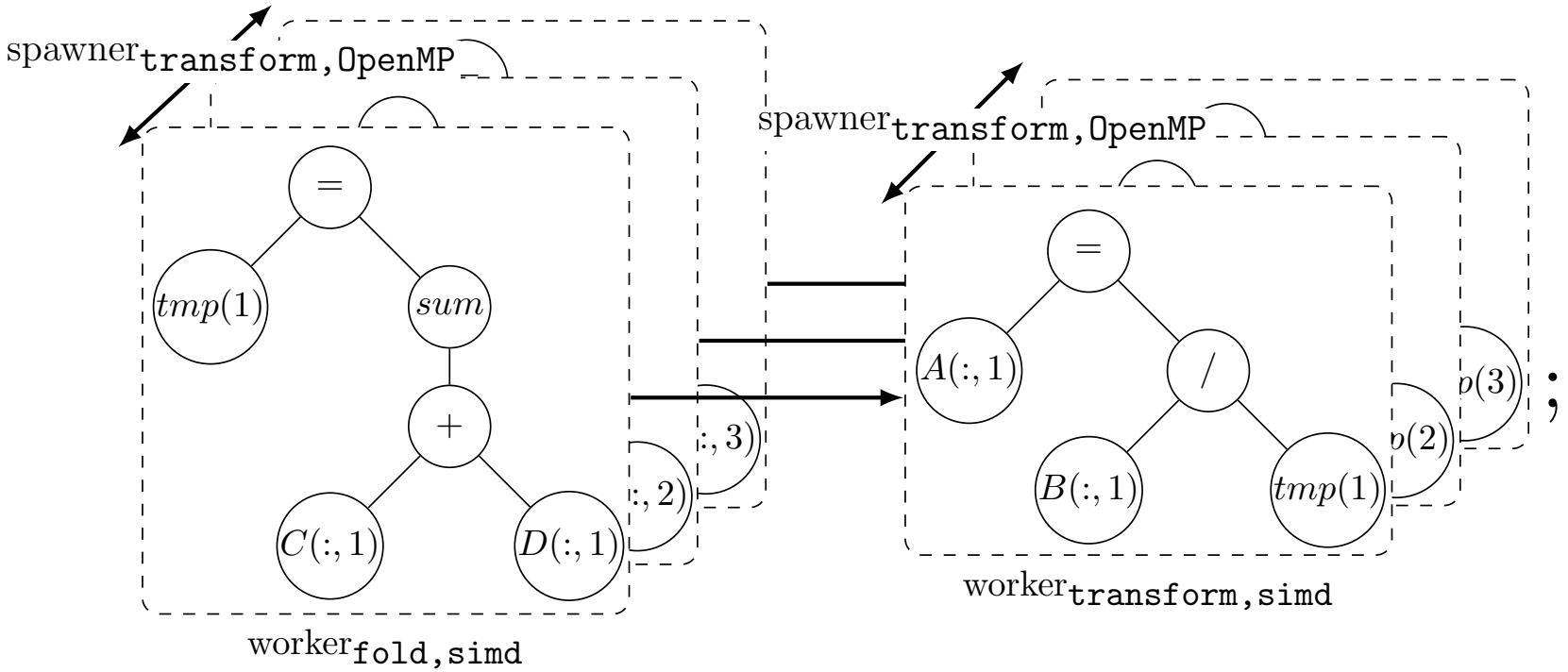
Parallel Skeletons extraction process - Take 2

```
A = B / sum(C+D);
```



Parallel Skeletons extraction process - Take 2

```
A = B / sum(C+D);
```



Motion Detection

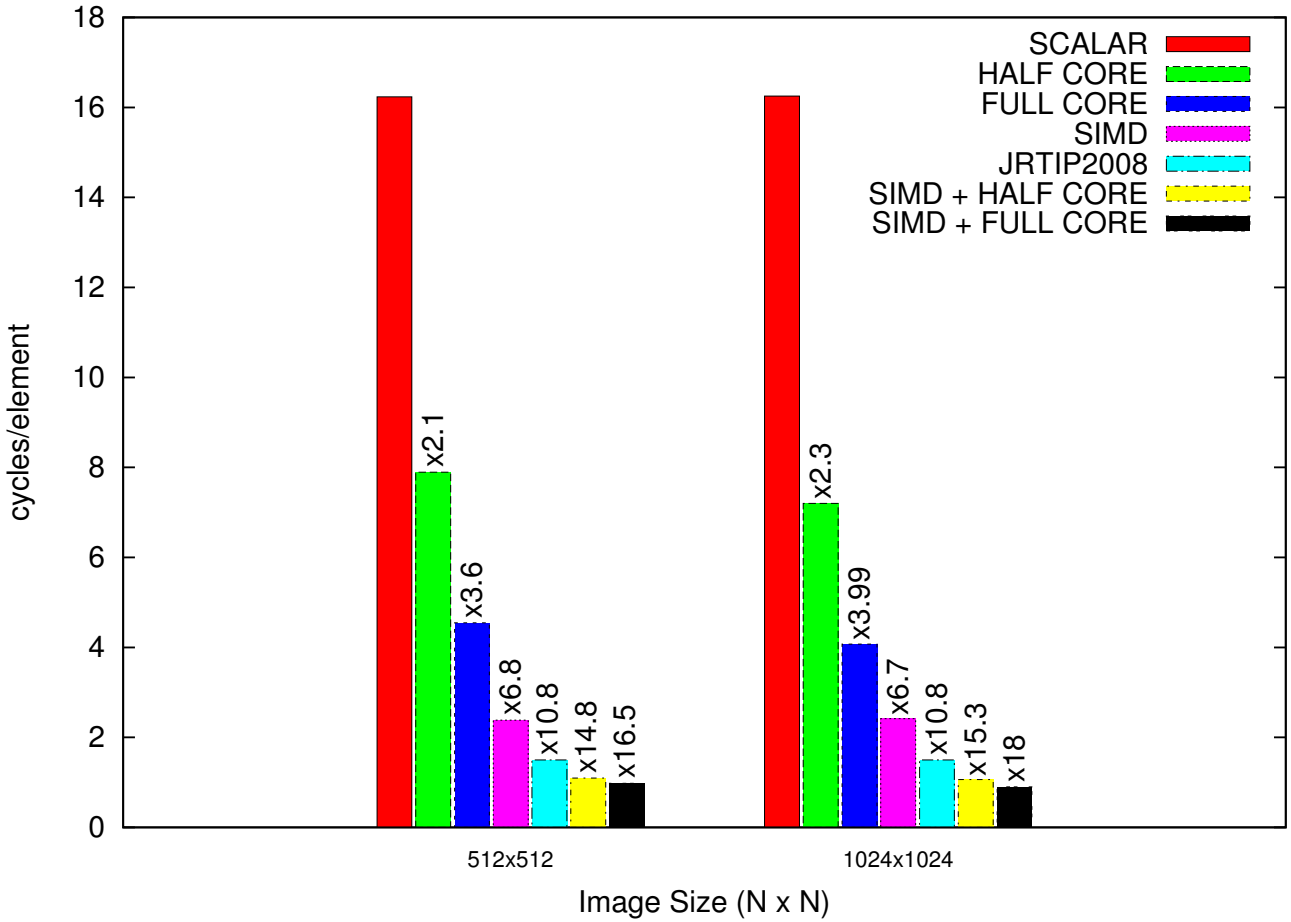
Lacassagne et al., ICIP 2009

- Sigma-Delta algorithm based on background substraction
- Use local gaussian model of lightness variation to detect motion
- **Challenge**: Very low arithmetic density
- **Challenge**: Integer-based implementation with small range

Motion Detection

```
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



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);
}
```

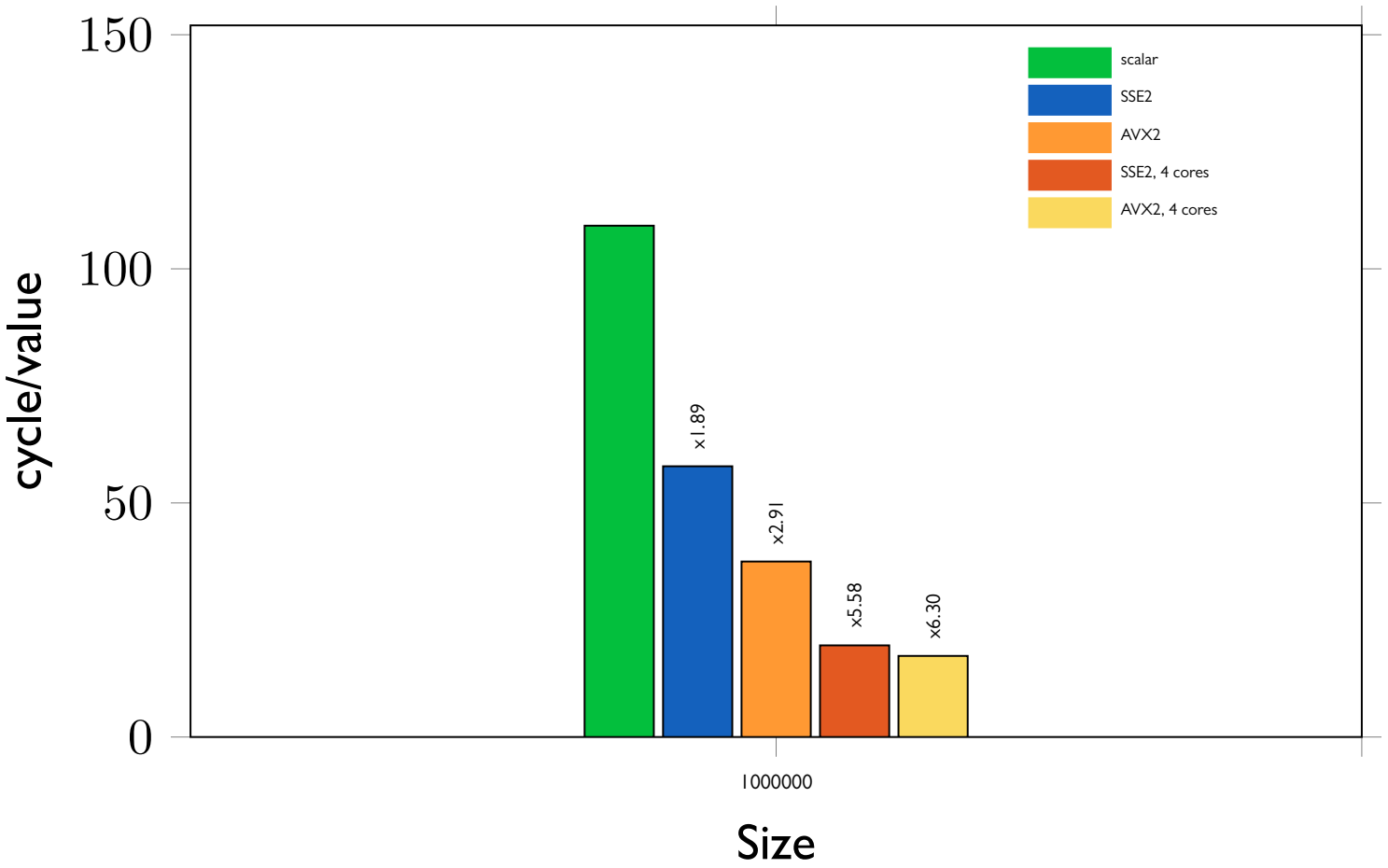
Black and Scholes Option Pricing

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;
}
```

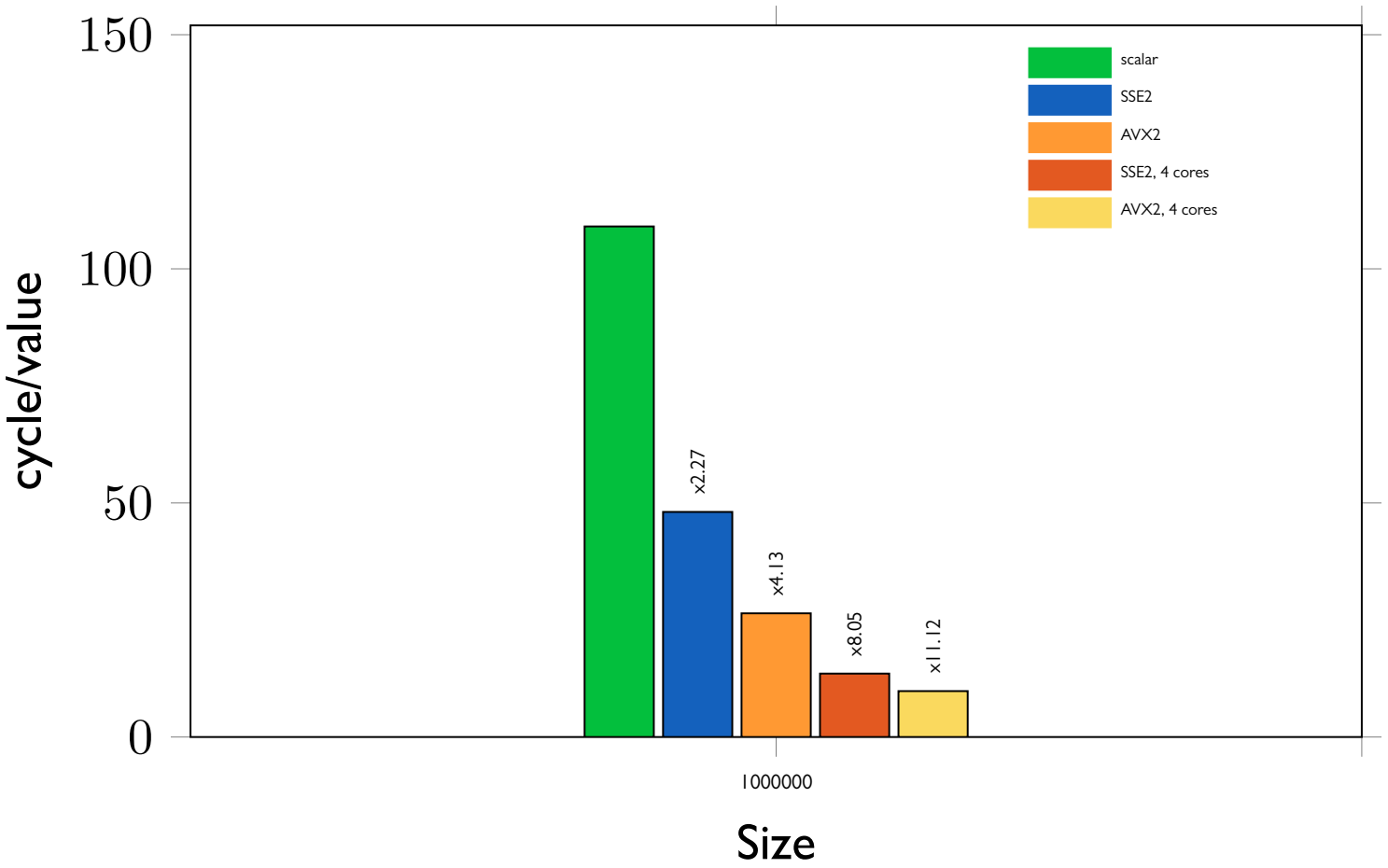
Black and Scholes Option Pricing

Performance



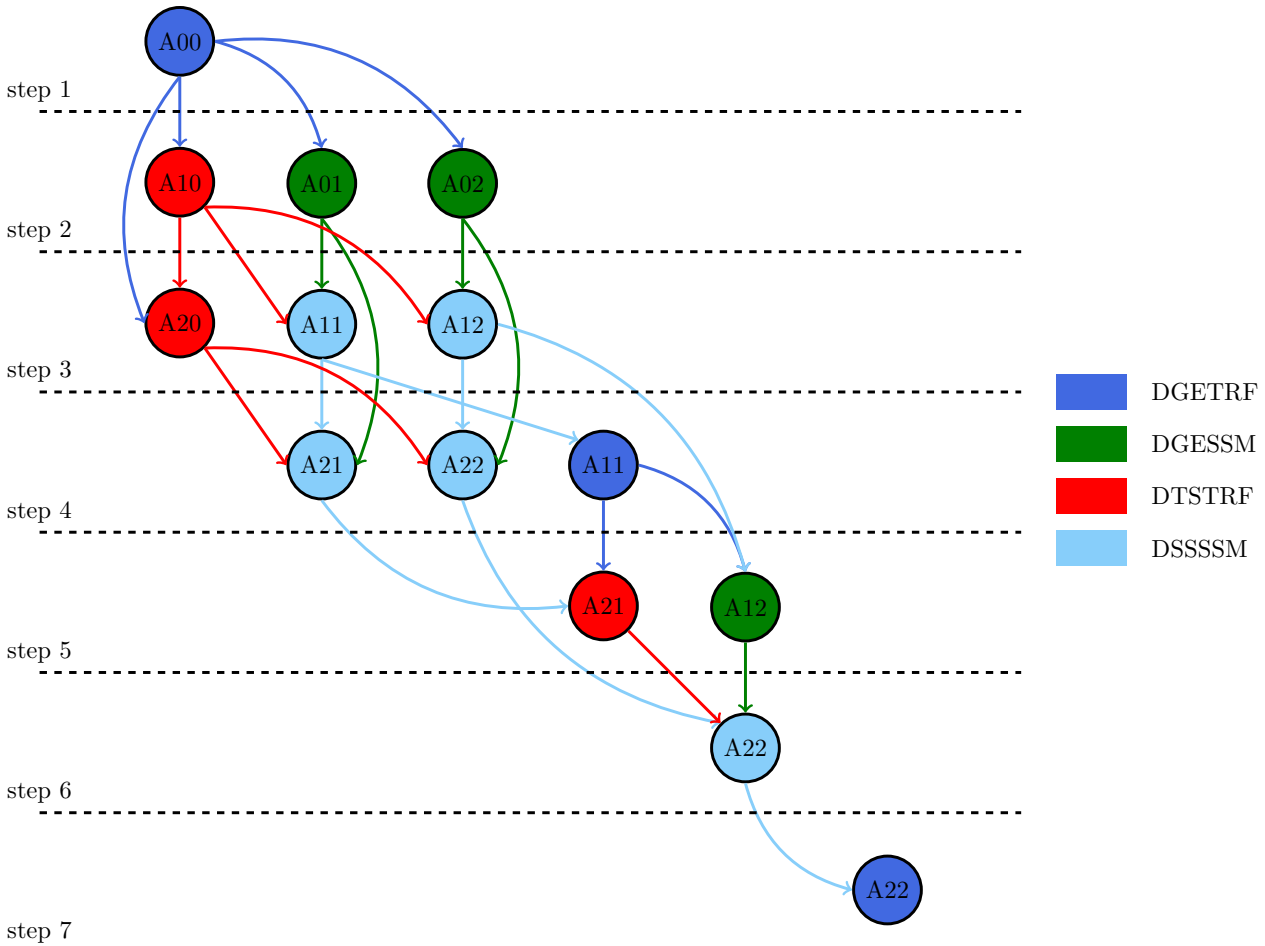
Black and Scholes Option Pricing

Performance with loop fusion/futurisation



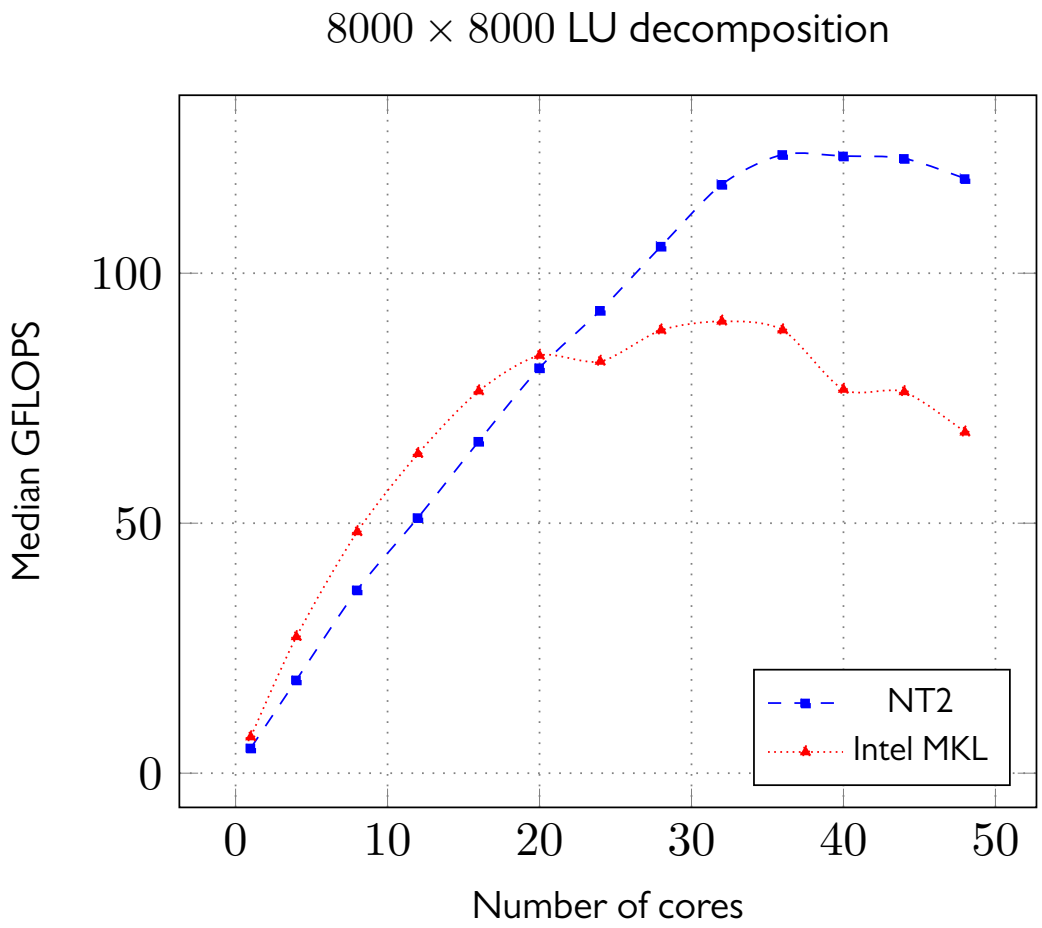
LU Decomposition

Algorithm



LU Decomposition

Performance



Talk Layout

Conclusion

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, DSEL needs informations about the hardware system
- Integrating hardware descriptions as Generic components increases tools portability and re-targetability

Conclusion

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, DSEL needs informations about the hardware system
- Integrating hardware descriptions as Generic components increases tools portability and re-targetability

Our Achievements

- A new method for parallel software development
- Efficient libraries working on large subset of hardware
- High level of performances across a wide application spectrum

Perspectives

DSEL as C++ first class idiom

- Build partial evaluation into the language
- Ease transition between regular and meta C++
- Mid-term Prospect: metaOCAML like quoting for C++

DSEL and compilers relationship

- C++ DSEL hits a limit on their applicability
- Compilers often lack high level informations for proper optimization
- Mid-term Prospect: Hybrid library/compiler approaches for DSEL

Thanks for your attention