HPC and Parallel Architectures for Big Data

Joel Falcou

PARSYS - POSTALE Team
LRI - INRIA

08/07/2014

Who are we

Parsys/Postale Team

- Joint LRI/INRIA team
- 6 permanents researchers
- 8 PHDs

Who are we

Parsys/Postale Team

- Joint LRI/INRIA team
- 6 permanents researchers
- 8 PHDs

Research interests

- Algorithms for Computer Vision, Linear Algebra (LAPACK)
- High-Level parallel programming tools (Boost.SIMD, NT2)
- Hardware Exploration

The hardware landscape

Decades of hardware improvements

- Scientific Computing now drives most hardware innovations
- Current Solution: Parallel architectures
- Machines become more and more complex

The hardware landscape

Decades of hardware improvements

- Scientific Computing now drives most hardware innovations
- Current Solution: Parallel architectures
- Machines become more and more complex

The Game Changing Data Size

- One example: the Large Hadron Collider
  - 1Gb of data per events
  - 1000+ events per experiments
  - dozens of experiments
- Moving from on-site HPC clusters to Cloud computing

Challenges

From HPC to Big Data

- HPC Hardware is not adapted to Big Data issues
- Design efficient Big Data software tools

Challenges

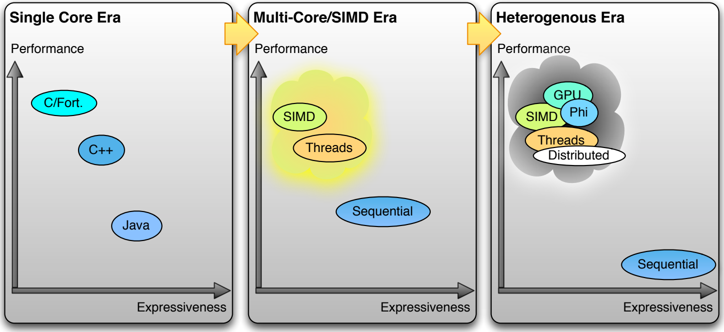
From HPC to Big Data

- HPC Hardware is not adapted to Big Data issues
- Design efficient Big Data software tools

From Big Data to HPC

- From experiments to simulation to analytics
- Increasing size of HPC simulations

The Usability Challenges of HPC



Designing tools for Scientific Computing

Challenges

Designing tools for Scientific Computing

Challenges

1. Be non-disruptive

Designing tools for Scientific Computing

Challenges

1. Be non-disruptive
2. Domain driven optimizations

Designing tools for Scientific Computing

Challenges

1. Be non-disruptive
2. Domain driven optimizations
3. Provide intuitive API for the user

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
5. Be efficient

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
5. Be efficient

Our Approach

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
5. Be efficient

Our Approach

- Design tools as C++ libraries (1)

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
5. Be efficient

Our Approach

- Design tools as C++ libraries (1)
- Design these libraries as Domain Specific Embedded Language (DSEL) (2+3)

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
5. Be efficient

Our Approach

- Design tools as C++ libraries (1)
- Design these libraries as Domain Specific Embedded Language (DSEL) (2+3)
- Use Parallel Skeletons as parallel components (4)

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
5. Be efficient

Our Approach

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

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

DSEL in practice

- Relies on operator overload abuse
- Carry semantic information around code fragment
- Library like code is self-aware of optimizations

Exploiting DSEL

- At the expression level: code generation for arbitrary hardware
- At the function level: inter-procedural optimizations

Parallel DSEL in practice

Objectives

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

Parallel DSEL in practice

Objectives

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

Our contribution

- BSP++ : Generic C++ BSP for shared/distributed memory
- Quaff: DSEL for skeleton programming
- Boost.SIMD: DSEL for portable SIMD programming
- NT2: Matlab like DSEL for scientific computing

Parallel DSEL in practice

Objectives

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

Our contribution

- BSP++ : Generic C++ BSP for shared/distributed memory
- Quaff: DSEL for skeleton programming
- Boost.SIMD: DSEL for portable SIMD programming
- NT2: Matlab like DSEL for scientific computing

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 SIMD for in-core optimizations
- Use recursive parallel skeletons
- Code is made independant of architecture and runtime

Parallel Skeletons in a nutshell

Basic Principles [COLE 1989]

- 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 1989]

- 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

Skeleton in Big Data

The MapReduce case

- MapReduce : abstraction of parallel processing
- Implemented on a large subset of settings (Hadoop, Intel DAH)
- Successful as it shields users from the details of parallelism

Skeleton in Big Data

The MapReduce case

- MapReduce : abstraction of parallel processing
- Implemented on a large subset of settings (Hadoop, Intel DAH)
- Successful as it shields users from the details of parallelism

That's quite innovative I guess ?

- MapReduce is a small application of Parallel Skeletons
- Hadoop is a smart implementation of those skeletons

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) );
```

Some results: Smith-Waterman Algorithm

Platform	Size	Hardware	Best Speedup	GCUPs
State of the Art				
cluster (MPI)	24,894,250	64 cores	33x	1.45
2 clusters (MPI)	816,394	20 procs	14x	0.37
cluster (MPI)	1,100,000	60 procs	39x	0.25
cluster (MPI/OpenMP)	2,000	24 cores	14x	4.38
Our results				
cluster (MPI)	1,072,950	128 cores	73x	6.53
cluster (MPI/OpenMP)	1,072,950	128 cores	116x	10.41
OpenMP	1,072,950	16 cores	16x	0.40
Hopper (MPI)	5,303,436	3072 cores	260x	3.09
Hopper (MPI+OpenMP)	24,894,269	6144 cores	5664x	15,5

Hardware Challenges for Big Data

Hardware challenges

- Modern machines relies on cache and data locality
- HPC systems are over-provisioned in CPUs, under-provisioned in I/O
- How to design data-friendly HPC systems ?
- How to apply HPC techniques to highly irregular problems ?

Hardware Challenges for Big Data

Hardware challenges

- Modern machines relies on cache and data locality
- HPC systems are over-provisioned in CPUs, under-provisioned in I/O
- How to design data-friendly HPC systems ?
- How to apply HPC techniques to highly irregular problems ?

Hardware Challenges for Big Data

Hardware challenges

- Modern machines relies on cache and data locality
- HPC systems are over-provisioned in CPUs, under-provisioned in I/O
- How to design data-friendly HPC systems ?
- How to apply HPC techniques to highly irregular problems ?

Hardware Challenges for Big Data

Hardware challenges

- Modern machines relies on cache and data locality
- HPC systems are over-provisioned in CPUs, under-provisioned in I/O
- How to design data-friendly HPC systems ?
- How to apply HPC techniques to highly irregular problems ?

Software challenges for Big Data

- HPC softwares aggregates 20+ years of expertise
- Lots of efficient implementations of classical tools
- Shouldn't we do the same for Big Data ?

Hardware Challenges for Big Data

Hardware challenges

- Modern machines relies on cache and data locality
- HPC systems are over-provisioned in CPUs, under-provisioned in I/O
- How to design data-friendly HPC systems ?
- How to apply HPC techniques to highly irregular problems ?

Software challenges for Big Data

- HPC softwares aggregates 20+ years of expertise
- Lots of efficient implementations of classical tools
- Shouldn't we do the same for Big Data ?

Conclusion

Common Challenges

- Simplify access to HPC systems to data scientists
- Explore new memory oblivious algorithms
- Apply HPC tools design to create Big Data tools
- Brigde the gap between the communities

Conclusion

Common Challenges

- Simplify access to HPC systems to data scientists
- Explore new memory oblivious algorithms
- Apply HPC tools design to create Big Data tools
- Brigde the gap between the communities

Conclusion

Common Challenges

- Simplify access to HPC systems to data scientists
- Explore new memory oblivious algorithms
- Apply HPC tools design to create Big Data tools
- Brigde the gap between the communities

Conclusion

Common Challenges

- Simplify access to HPC systems to data scientists
- Explore new memory oblivious algorithms
- Apply HPC tools design to create Big Data tools
- Brigde the gap between the communities

Impact of Big Data on HPC

- Isolate new parallel paradigms
- Replace brute force HPC with data-inspired algorithms
- Drive the design of new hardware systems

Thanks for your attention