High-Level Parallel Programming EDSL
A BOOST libraries use case

Joel Falcou

LRI, Université Paris SUD, Orsay

BOOST'CON 09 – Aspen, Colorado – May, 2009

Few words about me

Who am I ?

- Assistant professor at University Paris South XI

Few words about me

Who am I ?

- Assistant professor at University Paris South XI
- Works in the PARALL team of the LRI
  - Fault Tolerance
  - Grid and Cluster Computing
  - Parallel Programming tools

Few words about me

Who am I ?

- Assistant professor at University Paris South XI
- Works in the PARALL team of the LRI
  - Fault Tolerance
  - Grid and Cluster Computing
  - Parallel Programming tools
- Focus :
  - Integrating High Level Models into common languages
  - Architecture/Algorithm Tuning
  - Application of Meta-Programming to parallelism

Few words about me

Who am I ?

- Assistant professor at University Paris South XI
- Works in the PARALL team of the LRI
  - Fault Tolerance
  - Grid and Cluster Computing
  - Parallel Programming tools
- Focus :
  - Integrating High Level Models into common languages
  - Architecture/Algorithm Tuning
  - Application of Meta-Programming to parallelism

Presentation Layout

1. Context
    - Living Moore's Law Ending
    - The Desktop Super-Computer Era
    - Classic Tools Design
2. Parallel Skeletons in a (somehow large) Nutshell
3. From Theory to C++ Practice : the QUAFF library
4. Experimental Results
5. Conclusion

"Where a calculator on the ENIAC is equipped with 19,000 vacuum tubes and weighs 30 tons, computers in the future may have only 1,000 vacuum tubes and perhaps only weigh 1.5 tons."

- Popular Mechanics, March 1949

Living in a Super-Computer World

Moore's Law is ending

CPU performances stalls despite continuous progress in transistor integration

Living in a Super-Computer World

Moore’s Law is ending

CPU performances stalls despite continuous progress in transistor integration

- Thermal dissipation
- The ’Die Desert’ Syndrom
- Memory bandwidth

Living in a Super-Computer World

Moore's Law is ending

CPU performances stalls despite continuous progress in transistor integration

- Thermal dissipation
- The 'Die Desert' Syndrom
- Memory bandwidth

Switching gears in the GHZ race

Living in a Super-Computer World

Moore's Law is ending

CPU performances stalls despite continuous progress in transistor integration

- Thermal dissipation
- The 'Die Desert' Syndrom
- Memory bandwidth

Switching gears in the GHZ race

- 1998 : Multimedia extension (AltiVec, SSEx)

Living in a Super-Computer World

Moore's Law is ending

CPU performances stalls despite continuous progress in transistor integration

- Thermal dissipation
- The 'Die Desert' Syndrom
- Memory bandwidth

Switching gears in the GHZ race

- 1998 : Multimedia extension (AltiVec, SSEx)
- 2000 : Multi-processors
- 2003 : Multi-cores (2X, 4X and soon 8X cores)

Living in a Super-Computer World

Moore's Law is ending

CPU performances stalls despite continuous progress in transistor integration

- Thermal dissipation
- The 'Die Desert' Syndrom
- Memory bandwidth

Switching gears in the GHZ race

- 1998 : Multimedia extension (AltiVec, SSEx)
- 2000 : Multi-processors
- 2003 : Multi-cores (2X, 4X and soon 8X cores)
- 2005 : Programmable GPUs (NVIDIA, ATI)

Living in a Super-Computer World

Moore's Law is ending

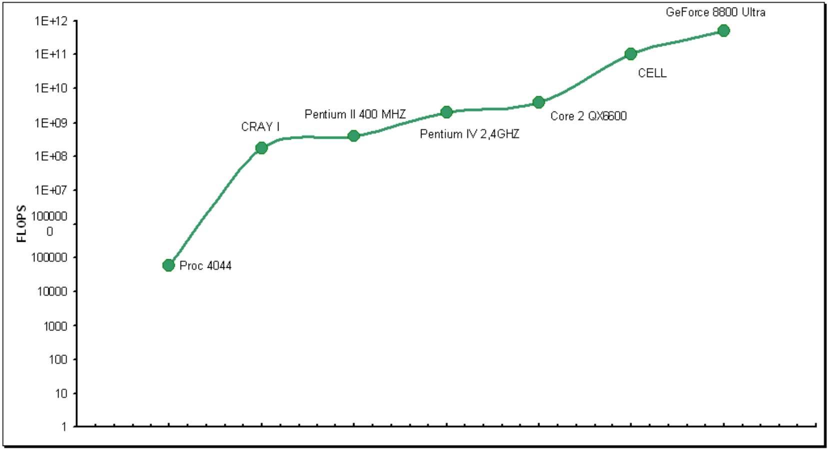
CPU performances stalls despite continuous progress in transistor integration

- Thermal dissipation
- The 'Die Desert' Syndrom
- Memory bandwidth

Switching gears in the GHZ race

- 1998 : Multimedia extension (AltiVec, SSEx)
- 2000 : Multi-processors
- 2003 : Multi-cores (2X, 4X and soon 8X cores)
- 2005 : Programmable GPUs (NVIDIA, ATI)
- 20xx : On-Chip Heterogeneous Architectures (CELL, Larabee)

The Infamous GFLOPS Slide



Is the Free Lunch over yet ?

The hidden cost of new architectures

The more parallel, the more complex it becomes to work with:

- Many programming model
- Many technologic choices
- Inter-operability of heterogeneous machines

Is the Free Lunch over yet ?

The hidden cost of new architectures

The more parallel, the more complex it becomes to work with:

- Many programming model
- Many technologic choices
- Inter-operability of heterogeneous machines

How can a non-expert in parallelism take full advantage of these ?

Is the Free Lunch over yet ?

The hidden cost of new architectures

The more parallel, the more complex it becomes to work with:

- Many programming model
- Many technologic choices
- Inter-operability of heterogeneous machines

How can a non-expert in parallelism take full advantage of these ?

Our Stance

Tools for programming upcoming new parallel architectures are bound to become strategic for both academics and industrials

```
//------------------------------------------------
//detection of most easy points to track
//------------------------------------------------

local_detected_pts_number=0;  //reset detection

top1 = mtime();
if(!first_iteration)
{
  Detection(INT_img_pred,DETBORDER_left,DETBORDER_up,DETBORDER_right,DETBORDER_bottom,
            local_Detected_point_x,local_Detected_point_y,&local_detected_pts_number,local_wanted_pts_number);
}
top2 = mtime();
timer[3] = top2-top1;

//------------------------------------------------
//tracking of selected points
//------------------------------------------------
top1 = mtime();

TrackingPoints(local_Detected_point_x,local_Detected_point_y,local_Tracked_point_x,local_Tracked_point_y,local_detected_pts_number,
img_pred.data,img_curr.data,
LR2_img_pred.data,LR2_img_curr.data,
LR4_img_pred.data,LR4_img_curr.data,
TX,TY);

SwapImages();

MPI_Barrier(MPI_COMM_WORLD);

MPI_Gather( local_Detected_point_x, local_wanted_pts_number, MPI_DOUBLE, Detected_point_x, local_wanted_pts_number, MPI_DOUBLE,MASTER,
MPI_Gather( local_Detected_point_y, local_wanted_pts_number, MPI_DOUBLE, Detected_point_y, local_wanted_pts_number, MPI_DOUBLE,MASTER,
MPI_Gather( local_Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE,MASTER, MP
MPI_Gather( local_Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE,MASTER, MP

MPI_Reduce(&local_detected_pts_number, detected_pts_number,1, MPI_INT, MPI_SUM, MASTER, MPI_COMM_WORLD);

MPI_Allgather( local_Detected_point_x, local_wanted_pts_number, MPI_DOUBLE, Detected_point_x, local_wanted_pts_number, MPI_DOUBLE, MPI_
MPI_Allgather( local_Detected_point_y, local_wanted_pts_number, MPI_DOUBLE, Detected_point_y, local_wanted_pts_number, MPI_DOUBLE, MPI_
MPI_Allgather( local_Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_x, local_wanted_pts_number, MPI_DOUBLE, MPI_CO
MPI_Allgather( local_Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE, Tracked_point_y, local_wanted_pts_number, MPI_DOUBLE, MPI_CO

MPI_Allreduce(&local_detected_pts_number, detected_pts_number,1, MPI_INT, MPI_SUM, MPI_COMM_WORLD);
MPI_Allreduce(timer, timer, 10, MPI_FLOAT, MPI_MAX, MPI_COMM_WORLD);


//---------------------------------------------------
// movement matrix extraction (median square value)
//---------------------------------------------------

top1 = mtime();
```

The Parallel Gordian Node

For a given task at hand, we want

- Easy to read/maintain code
- Efficiency
- Abstraction of underlying architectures

The Parallel Gordian Node

For a given task at hand, we want

- Easy to read/maintain code
- Efficiency
- Abstraction of underlying architectures

What to do then ?

- New Languages with parallel semantic
- Software libraries for existing languages

Compiler-based Tools

Principles

Existing compilers are modified to add either new languages constructs or unerlying parallelised code generator

Compiler-based Tools

Principles

Existing compilers are modified to add either new languages constructs or unerlying parallelised code generator

Limitations

- One compiler per architectures ?
- Time to market
- Long term support
- Acceptance by actual users

Library-based Tools

Principles

- Gather data structures and related functions
- Favor binary level reuse
- Capitalize on Algorithmic Expertise

Library-based Tools

Principles

- Gather data structures and related functions
- Favor binary level reuse
- Capitalize on Algorithmic Expertise

Limitations

- Few to none inter-procedural optimization
- Combinatorial explosion of variants
- Internal dependances on data structures choice

Cutting the Gordian Node

Best of both worlds

- Compilers :

Cutting the Gordian Node

Best of both worlds

- Compilers :
  - Generated code is efficient
  - Well-known theory and practice

Cutting the Gordian Node

Best of both worlds

- Compilers :
  - Generated code is efficient
  - Well-known theory and practice
- Libraries :

Cutting the Gordian Node

Best of both worlds

- Compilers :
  - Generated code is efficient
  - Well-known theory and practice
- Libraries :
  - Domain specific interface
  - User-level abstraction
  - Easy to use and deploy

Cutting the Gordian Node

Best of both worlds

- Compilers :
  - Generated code is efficient
  - Well-known theory and practice
- Libraries :
  - Domain specific interface
  - User-level abstraction
  - Easy to use and deploy

Why not using Embedded DSL ?

Presentation Layout

"When we had no computers, we had no programming problem either.
When we had a few computers,we had a mild programming problem.
Confronted with machines a million times as powerful, we are faced with a gigantic programming problem."

- Edsger Dijkstra

Parallel Skeletons in a few words

Basic Principles

- Co-proposed by Cole and Darlington in 1989
- There is patterns in parallel applications
- Those patterns can be generalized
- Application = Combination of such patterns

Parallel Skeletons in a few words

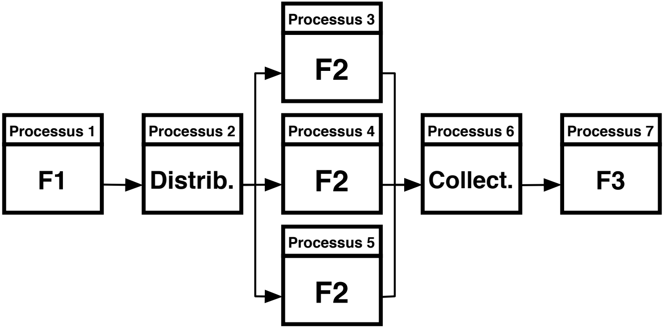
Basic Principles

- Co-proposed by Cole and Darlington in 1989
- There is patterns in parallel applications
- Those patterns can be generalized
- Application = Combination of such patterns

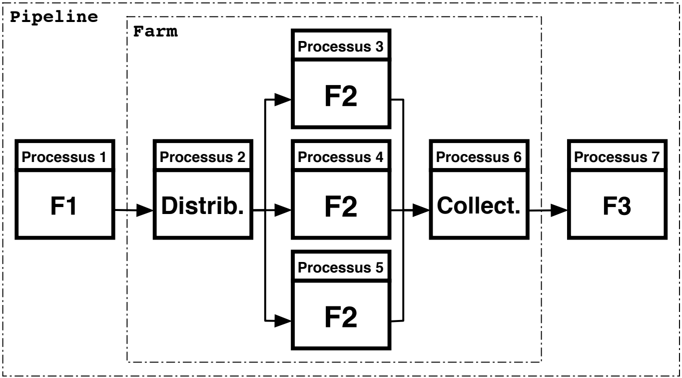
Functionnal point of view

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

Spotting skeletons when you see one



Spotting skeletons when you see one



State of the Art in Parallel Skeleton

Dedicated Languages

Mostly functional oriented

- P3L [DANELUTTO 95]
- Skipper [SEROT 95]

State of the Art in Parallel Skeleton

Dedicated Languages

Mostly functional oriented

- P3L [DANELUTTO 95]
- Skipper [SEROT 95]

Software Libraries

Mostly imperative

- C : eSkel [COLE 89]
- C++ : Muesli [KUCHEN 03]
- JAVA : Lithium [DANELUTTO 02]

State of the Art in Parallel Skeleton

Dedicated Languages

Mostly functional oriented

- P3L [DANELUTTO 95]
- Skipper [SEROT 95]

Software Libraries

Mostly imperative

- C : eSkel [COLE 89]
- C++ : Muesli [KUCHEN 03]
- JAVA : Lithium [DANELUTTO 02]

Applications

- Google MapReduce
- MalBa for non-linear optimisation

Definitions

A Small SKeleton Grammar

- $\Sigma \ ::= \ \text{Seq}\ f \mid \text{Pipe}(\Sigma_1, \ldots, \Sigma_n) \mid \text{Farm}(n, \Sigma) \mid \text{Scm}(n, f_s, \Sigma, f_m)$
- $f, f_s, f_m \ ::= \ $ sequential, application-specific user-defined functions
- $n \ ::= \ \textit{integer} \geq 1$

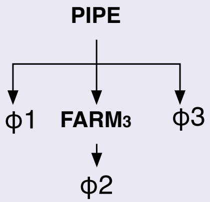
Definitions

A Small SKeleton Grammar

- $\Sigma \;::=\; \text{Seq } f \mid \text{Pipe}(\Sigma_1, \ldots, \Sigma_n) \mid \text{Farm}(n, \Sigma) \mid \text{Scm}(n, f_s, \Sigma, f_m)$
- $f, f_s, f_m \;::=\;$ sequential, application-specific user-defined functions
- $n \;::=\; \textit{integer} \geq 1$

Example

$$\textbf{Pipe}(\textbf{Seq } \phi_1,\ \textbf{Farm}(3,\ \textbf{Seq } \phi_2),\ \textbf{Seq } \phi_3)$$



Process network structure

The *CSP* model

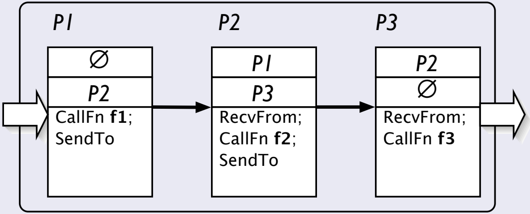
- Set of processes communicating by channels
- Each process keeps tracks of its *predecessors* and *successors*
- Each process executes a sequence of *instructions*

Process network structure

The *CSP* model

- Set of processes communicating by channels
- Each process keeps tracks of its *predecessors* and *successors*
- Each process executes a sequence of *instructions*

Example : process network for **Pipeline**



Process network algebra

Definition

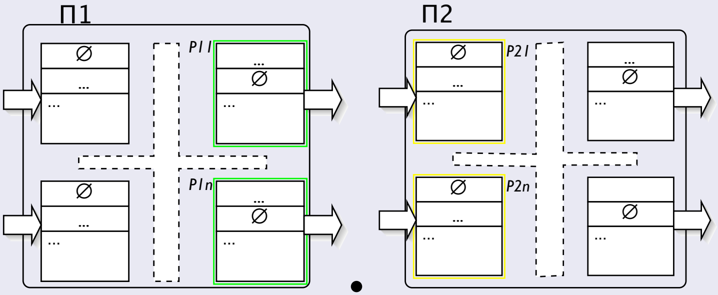
- Operators incrementally build process networks
- Operators are formally defined using production rules

Example : the • operator

$$\frac{\pi_i = \langle P_i, I_i, O_i \rangle \; (i = 1, 2) \qquad |O_1| = |I_2| = m}{\pi_1 \bullet \pi_2 = \langle (P_1 \cup P_2)[(o_1^j, \sigma) \leftarrow \phi_d((o_1^j, \sigma), i_2^j)]_{j=1\ldots m}[(i_2^j, \sigma) \leftarrow \phi_s((i_2^j, \sigma), o_1^j)]_{j=1\ldots m}, I_1, O_2 \rangle}$$

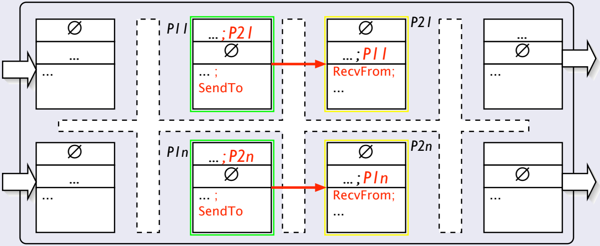
Process network algebra

Informal description of the • operator



Process network algebra

Informal description of the • operator



$\Pi 1 \bullet \Pi 2$

Skeleton as PN Algebra operators

Definition

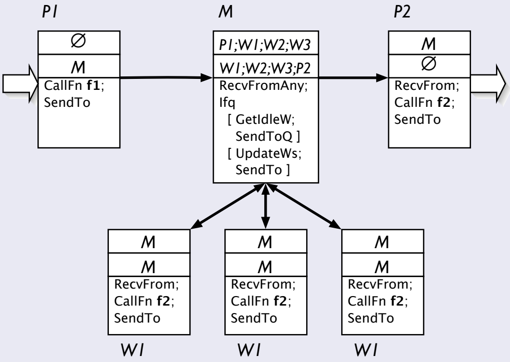
We define a conversion function $\mathcal{C}[[]]$ that transforms a skeleton into the equivalent process nextwork.

$$
\begin{aligned}
\mathcal{C}[[\text{Seq } f]] &\equiv \lceil f \rceil \\
\mathcal{C}[[\text{Pipe } \Sigma_1 \ldots \Sigma_n]] &\equiv \mathcal{C}[[\Sigma_1]] \bullet \ldots \bullet \mathcal{C}[[\Sigma_n]] \\
\mathcal{C}[[\text{Farm } n\ \Sigma]] &\equiv \lceil \text{FarmM} \rceil \bowtie (\mathcal{C}[[\Sigma]]_1 \parallel \ldots \parallel \mathcal{C}[[\Sigma]]_n) \\
\mathcal{C}[[\text{Scm } m\ f_s\ \Sigma\ f_m]] &\equiv \lceil f_s \rceil \triangleleft (\mathcal{C}[[\Sigma]]_1 \parallel \ldots \parallel \mathcal{C}[[\Sigma]]_m) \triangleright \lceil f_m \rceil
\end{aligned}
$$

Skeletons to process network transformation

Exampel

$$\textbf{Pipe}(\text{Seq } \phi_1,\ \textbf{Farm}(3,\ \text{Seq } \phi_2),\ \text{Seq } \phi_3)$$



Presentation Layout

Objectives

Proposing a C++ skeletons library

- Limit runtime overhead
- Use the formal model skeleton as guidelines

Objectives

Proposing a C++ skeletons library

- Limit runtime overhead
- Use the formal model skeleton as guidelines

What we want to write

```
run( pipeline( seq(F1), seq(F2) ) )
```

Objectives

Proposing a C++ skeletons library

- Limit runtime overhead
- Use the formal model skeleton as guidelines

What we want to run

```
if( rank == 0 )
{
  do { out = F1();
       MPI_Send(&out,1,MPI_INT,1,0,MPI_COMM_WORLD);
     } while( isValid(out) );
}
if( rank == 1 )
{
  do { MPI_Recv(&in,1,MPI_INT,0,0,MPI_COMM_WORLD,&s);
       out = F2(in);
       MPI_Send(&in,1,MPI_INT,2,0,MPI_COMM_WORLD);
     } while ( isValid(out) )
}
if( rank == 2 )
{
  do { MPI_Recv(&in,1,MPI_INT,1,0,MPI_COMM_WORLD,&s);
       F2(in);
     } while ( isValid(in) )
}
```

Design Rationale

Analysis

- Once defined, a skeleton application structures is immutable
- No need for dynamic add/remove of processus
- Represent Processes as statically polymorphic objects

Design choice

- Compile-time constructs represent processes
- Meta-programming helps handling those construct
- Proto bridges the gap between user interface and code generation

Challenges

User Interface

- Seamless integration of legacy code
- Support C and C++ style functions/functors
- Extract send/receive schemes from function prototypes
- BE usable by ’Joe Average’ developper

Code Generation

- Turn skeleton AST into Proto usable elements
- Implement meta semantic rules
- Optimize out process network
- Generate proper MPI commands

Presentation Layout

3D tracking of pedestrians

Objectives

Detect and track pedestrians in a stereoscopic video stream while keeping up with stream frequency (25-30 FPS)

3D tracking of pedestrians

Objectives

Detect and track pedestrians in a stereoscopic video stream while keeping up with stream frequency (25-30 FPS)

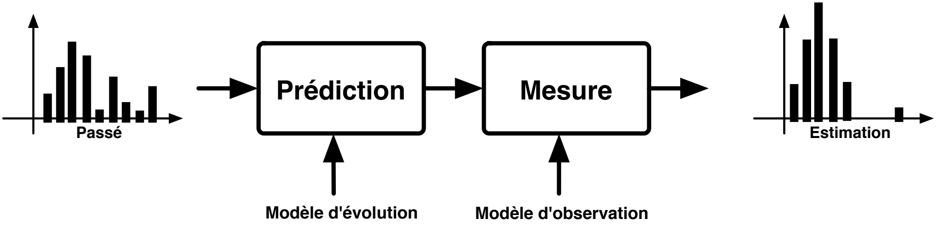
Difficulties

- Lightning, textures or pose may vary
- Partial or global occultations
- Dynamic background

Trackign with particle filter [BLAKE 96]

Principles

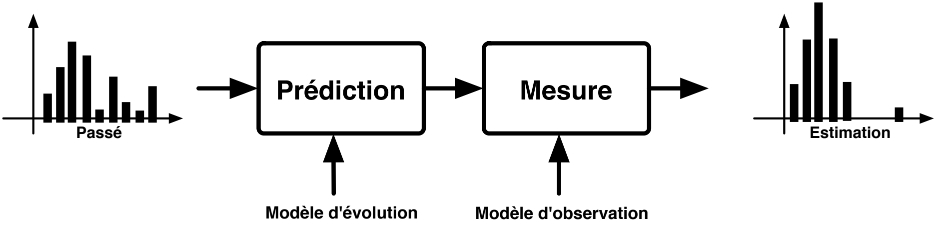
- Predict
- Measure
- Estimate



Trackign with particle filter [BLAKE 96]

Principles

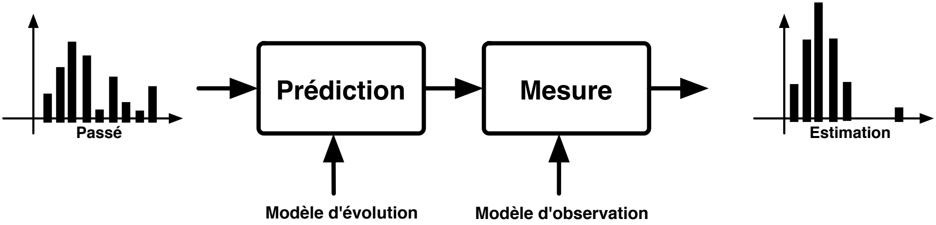
- Predict
- Measure
- Estimate



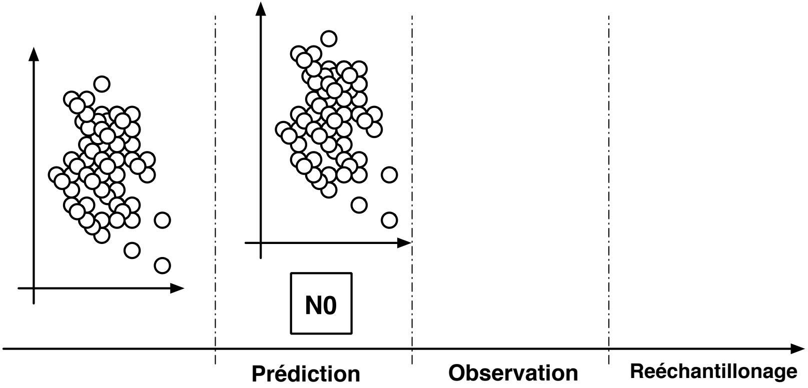
Trackign with particle filter [BLAKE 96]

Principles

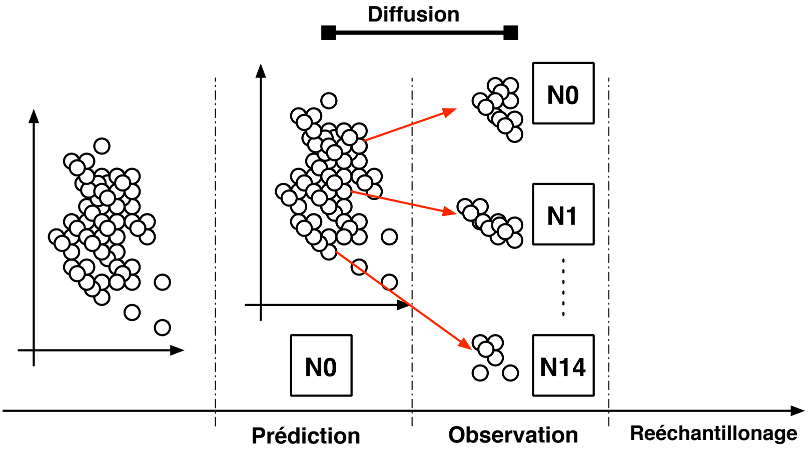
- Predict
- Measure
- Estimate



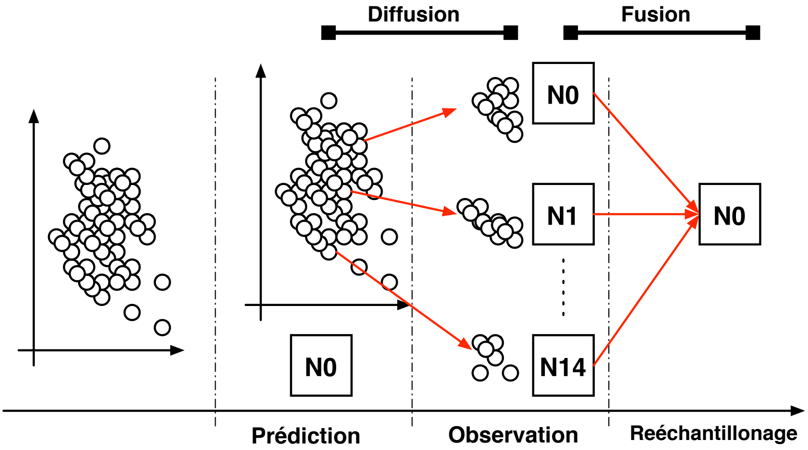
Particle filter parallelisation



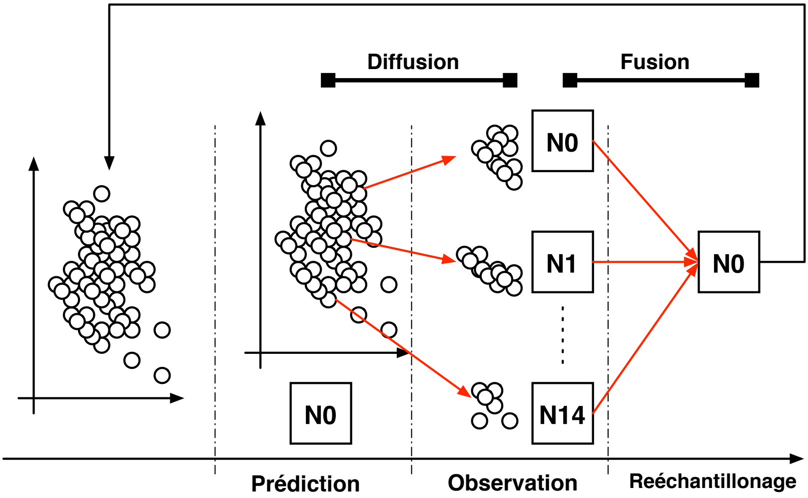
Particle filter parallelisation



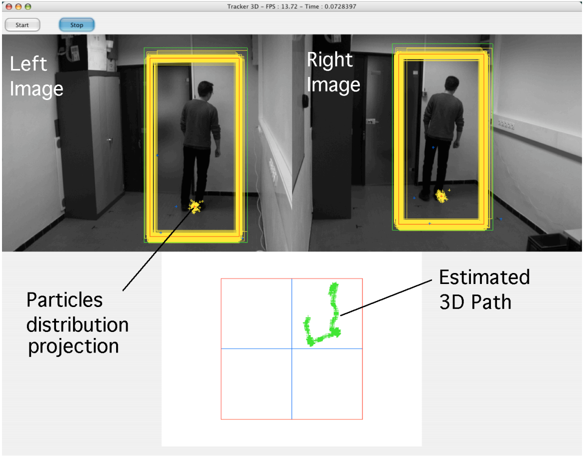
Particle filter parallelisation



Particle filter parallelisation



Sample application



Runtime Performances

	100	500	1000	5000
Sequential	$0.061s$	$0.299s$	$0.704s$	11.358
FPS	16.38	3.34	1.42	0.09
Parallel	$0.014s$	$0.0195s$	$0.0334s$	$0.105s$
FPS	70.8	51.3	29.96	9.4
Speed-up	4.31	15.38	21.1	107.35

Runtime Performances

	100	500	1000	5000
Sequential	$0.061s$	$0.299s$	$0.704s$	11.358
FPS	16.38	3.34	1.42	0.09
Parallel	$0.014s$	$0.0195s$	$0.0334s$	$0.105s$
FPS	70.8	51.3	29.96	9.4
Speed-up	4.31	15.38	21.1	107.35

Results (CIMCV/ECCV 06)

- Real-time obtained for more than 1000 particles

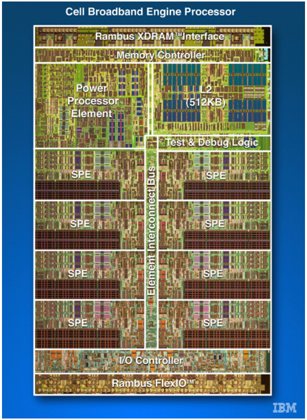
Runtime Performances

	100	500	1000	5000
Sequential	$0.061s$	$0.299s$	$0.704s$	11.358
FPS	16.38	3.34	1.42	0.09
Parallel	$0.014s$	$0.0195s$	$0.0334s$	$0.105s$
FPS	70.8	51.3	29.96	9.4
Speed-up	4.31	15.38	21.1	107.35

Results (CIMCV/ECCV 06)

- Real-time obtained for more than 1000 particles
- Maximum speed-up greater than $\times 100$.

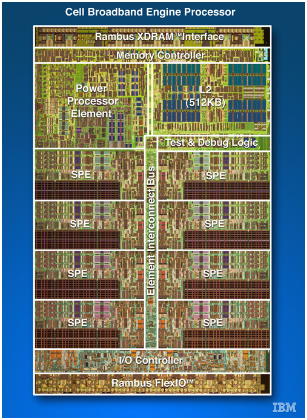
Architecture of the CELL processor



Features

- 9 Cores :1 PPE + 8 SPE
- 200 GB/s Bus
- Multi-thread support
- SIMD support (Altivec)
- Up to 400 GFLOPS

Architecture of the CELL processor



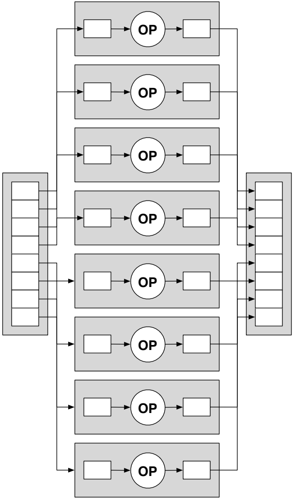
Features

- 9 Cores :1 PPE + 8 SPE
- 200 GB/s Bus
- Multi-thread support
- SIMD support (Altivec)
- Up to 400 GFLOPS

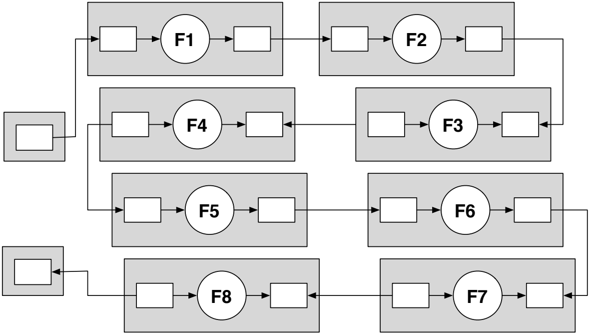
Developping on the CELL

How should I map an application on this thing ?

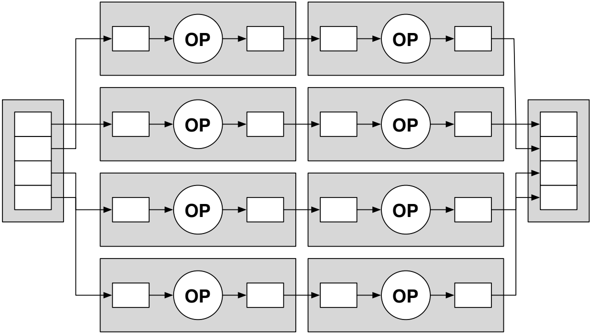
Various mapping ...



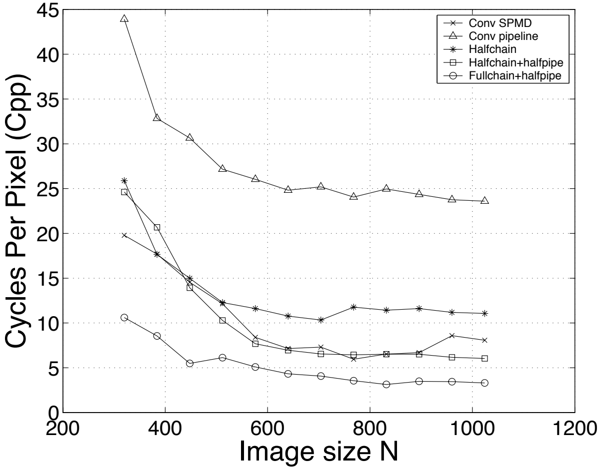
Various mapping ...



Various mapping ...



... lead to various results



Modification to the skeletons grammar and semantic

Identified Skeletons

- *Pipe* : parallel function composition over SPEs
- *Pardo* : replicate skeleton over SPEs
- *Seq* : skeleton integration

Modification to the skeletons grammar and semantic

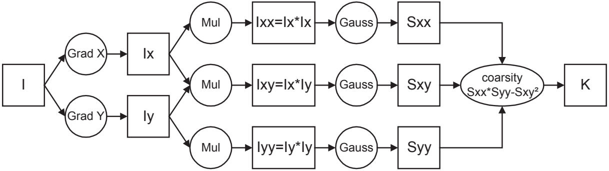
Identified Skeletons

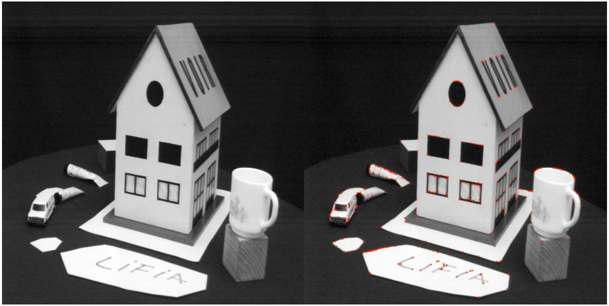
- *Pipe* : parallel function composition over SPEs
- *Pardo* : replicate skeleton over SPEs
- *Seq* : skeleton integration

Associated Grammar

$$\begin{array}{lcl} \mathcal{A} & ::= & \text{run } \Sigma \\ \Sigma & ::= & \Gamma \mid \textit{Pardo } n \; \Gamma \\ \Gamma & ::= & \textit{Seq } f \mid \textit{Pipe } \Gamma_1 \ldots \Gamma_n \\ f & ::= & \textit{C++ user-defined function} \\ n & ::= & \textit{integer} \geq 1 \end{array}$$

Application: Harris and Stephen Corner Detector





Application: Harris and Stephen Corner Detector

PPE code

```
#include <quaff/quaff.hpp>
QUAFF_REGISTER_KERNEL(harris);

int main(int ,char**)
{
  tile<float>  in(1,512,1,512);
  tile<float>  out(1,512,1,512);

  harris(in,out);

  return 0;
}
```

Application: Harris and Stephen Corner Detector

SPE code - classic Pipeline

```
#include <quaff/quaff.hpp>
using namespace quaff;

void sobel    ( tile<float> const& in,tile<float>& sx,tile<float>& sy);
void mul      ( tile<float> const& sx,tile<float> const& sy
              , tile<float>& sxx,tile<float>& sxy,tile<float>& syy);
void gauss    ( tile<float> const& sxx,tile<float> const& sxy,tile<float> const& syy
              , tile<float>& gxx,tile<float>& gxy,tile<float>& gyy);
void coarsity( tile<float> const& gxx,tile<float> const& gxy,tile<float> const& gyy
              , tile<float> const& out );

BEGIN_SKELL_KERNEL(harris);

void harris( tile<float> const&, tile<float>& )
{
  run( pardo<2>( seq(Sobel) | seq(Mul)) | seq(Gauss) | seq(Coarsity) ) );
}
```

Application: Harris and Stephen Corner Detector

SPE code - Half-chain

```
#include <quaff/quaff.hpp>
using namespace quaff;

void sobel    ( tile<float> const& in,tile<float>& sx,tile<float>& sy);
void mul      ( tile<float> const& sx,tile<float> const& sy
              , tile<float>& sxx,tile<float>& sxy,tile<float>& syy);
void gauss    ( tile<float> const& sxx,tile<float> const& sxy,tile<float> const& syy
              , tile<float>& gxx,tile<float>& gxy,tile<float>& gyy);
void coarsity( tile<float> const& gxx,tile<float> const& gxy,tile<float> const& gyy
              , tile<float> const& out );

BEGIN_SKELL_KERNEL(harris);

void harris( tile<float> const&, tile<float>& )
{
  run( pardo<4>( (seq(Sobel),seq(Mul)) | (seq(Gauss),seq(Coarsity)) ) );
}
```

Application: Harris and Stephen Corner Detector

SPE code - Full chain

```
#include <quaff/quaff.hpp>
using namespace quaff;

void sobel    ( tile<float> const& in,tile<float>& sx,tile<float>& sy);
void mul      ( tile<float> const& sx,tile<float> const& sy
              , tile<float>& sxx,tile<float>& sxy,tile<float>& syy);
void gauss    ( tile<float> const& sxx,tile<float> const& sxy,tile<float> const& syy
              , tile<float>& gxx,tile<float>& gxy,tile<float>& gyy);
void coarsity( tile<float> const& gxx,tile<float> const& gxy,tile<float> const& gyy
              , tile<float> const& out );

BEGIN_SKELL_KERNEL(harris);

void harris( tile<float> const&, tile<float>& )
{
  run( pardo<8>( (seq(Sobel), seq(Mul)), seq(Gauss), seq(Coarsity) )  ) );
}
```

Application: Harris and Stephen Corner Detector

Runtime performance

Mode	Classic	Half chain	Full chain
Hand-written	27.67 cpp	11.26 cpp	8.34 cpp
Quaff	28.07 cpp	11.36 cpp	8.42 cpp
Overhead	1.46 %	0.8%	0.9%

Application: Harris and Stephen Corner Detector

Runtime performance

Mode	Classic	Half chain	Full chain
Hand-written	27.67 cpp	11.26 cpp	8.34 cpp
Quaff	28.07 cpp	11.36 cpp	8.42 cpp
Overhead	1.46 %	0.8%	0.9%

Code-bloat impact

- Marshalling code $\approx$ 2Ko
- Functions interface $\approx$ 1Ko par opérateur
- Manual implementation : 8Kb per SPE
- Quaff implementation : 70Kb per SPE

Presentation Layout

BOOST as libraries building blocks

Boost.Proto

- Acted as the main enabling library
- Made the transition between model and code easy
- Cutted down the amount of code written by 50%

BOOST as libraries building blocks

Boost.Proto

- Acted as the main enabling library
- Made the transition between model and code easy
- Cutted down the amount of code written by 50%

Boost.Fusion and MPL

- Ease the transition between the Proto transform and actual runtime elements
- Work like peas in pod :)

BOOST as libraries building blocks

Boost.Proto

- Acted as the main enabling library
- Made the transition between model and code easy
- Cutted down the amount of code written by 50%

Boost.Fusion and MPL

- Ease the transition between the Proto transform and actual runtime elements
- Work like peas in pod :)

Boost.MPI

One word : serialization

Similar works in progress

NT2 - EDSL for linear algebra

- Matrix container with algebra and numeric operations
- Support SIMD extensions, OpenMP, pThread and soon GPUs
- Uses Proto, Fusion, Thread and MPL

Similar works in progress

NT2 - EDSL for linear algebra

- Matrix container with algebra and numeric operations
- Support SIMD extensions, OpenMP, pThread and soon GPUs
- Uses Proto, Fusion, Thread and MPL

BSP++ - Bulk Synchronous Parallelism in C++

- A BSP programs = sequence of parallel super-step
- BSP programs performances can be predicted with an analytic model
- Uses Lambda, Fusion and MPI

Similar works in progress

NT2 - EDSL for linear algebra

- Matrix container with algebra and numeric operations
- Support SIMD extensions, OpenMP, pThread and soon GPUs
- Uses Proto, Fusion, Thread and MPL

BSP++ - Bulk Synchronous Parallelism in C++

- A BSP programs = sequence of parallel super-step
- BSP programs performances can be predicted with an analytic model
- Uses Lambda, Fusion and MPI

Toward BOOST-based multi-stage programming

- Develop a variant of Kelly's TaskGraph library
- Planned to use meta-programming instead of macros

Future works

BOOST and parallel computing

- Parallelism became a recurring topic on the Dev. List
- Low-level support : *threads*, *coroutine*
- Structured support : *futures*

Future works

BOOST and parallel computing

- Parallelism became a recurring topic on the Dev. List
- Low-level support : *threads*, *coroutine*
- Structured support : *futures*

Possible Contributions

- BOOST support for CELL processor
- Boost.Simd proposal

PROTO contributions

Improvements

- Improving compilation time
- Support for van Wijngaarden grammars

Extending PROTO scope

- Adding support for generic template virtual machines
- Higher-level code transformation algorithms

Thanks for your attention.