Programmation générative et Architecture parallèle

Une approche guidée par la sémantique

Joel Falcou - LRI, Université Paris Sud, Orsay

Séminaire LRDE

24 Septembre 2008

Plan de l'exposé

Contexte

La fin de la loi de Moore

Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

Contexte

La fin de la loi de Moore

Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

- La consommation électrique
- La dissipation thermique.
- Le mur de la mémoire.

Contexte

La fin de la loi de Moore

Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

- La consommation électrique
- La dissipation thermique.
- Le mur de la mémoire.

La fin de la course au GHz : des sauts technologiques majeurs

Contexte

La fin de la loi de Moore

Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

- La consommation électrique
- La dissipation thermique.
- Le mur de la mémoire.

La fin de la course au GHz : des sauts technologiques majeurs

- 1998 : Extension multimédia : AltiVec

Contexte

La fin de la loi de Moore

Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

- La consommation électrique
- La dissipation thermique.
- Le mur de la mémoire.

La fin de la course au GHz : des sauts technologiques majeurs

- 1998 : Extension multimédia : AltiVec
- 2000 : Machine *multi-processeurs*.
- 2003 : Machine *multi-cœurs* : bi, quadri et bientôt octo-cœurs.

Contexte

La fin de la loi de Moore

Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

- La consommation électrique
- La dissipation thermique.
- Le mur de la mémoire.

La fin de la course au GHz : des sauts technologiques majeurs

- 1998 : Extension multimédia : AltiVec
- 2000 : Machine *multi-processeurs*.
- 2003 : Machine *multi-cœurs* : bi, quadri et bientôt octo-cœurs.
- 2005 : Carte graphique programmable (CUDA, Brook GPU)

Contexte

La fin de la loi de Moore

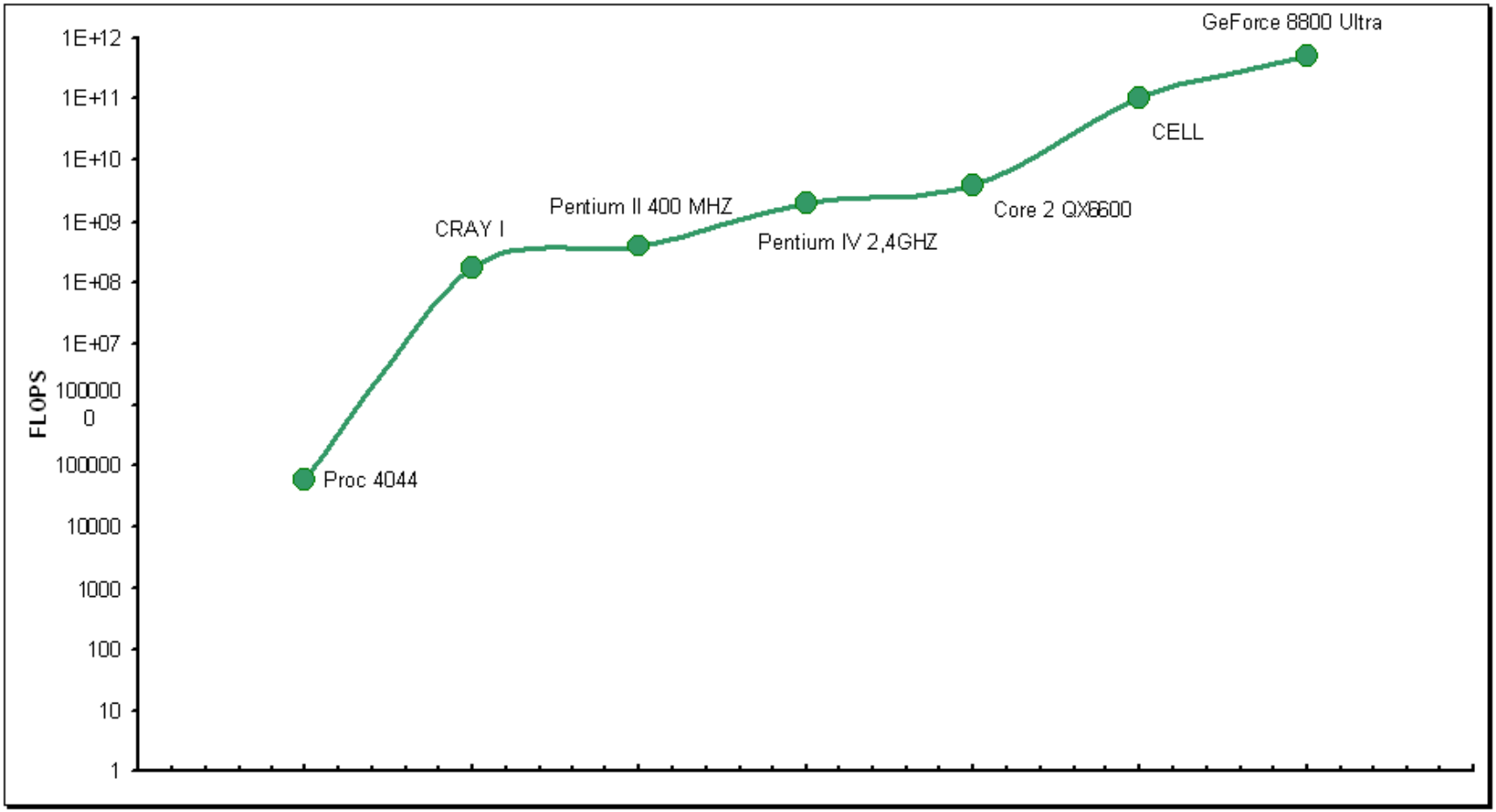
Si le nombre de transistors par unité de surface continue d'augmenter, la puissance des machines reste limité par :

- La consommation électrique
- La dissipation thermique.
- Le mur de la mémoire.

La fin de la course au GHz : des sauts technologiques majeurs

- 1998 : Extension multimédia : AltiVec
- 2000 : Machine *multi-processeurs*.
- 2003 : Machine *multi-cœurs* : bi, quadri et bientôt octo-cœurs.
- 2005 : Carte graphique programmable (CUDA, Brook GPU)
- 200x : Machine hétérogène sur puce : CELL, TILE64, Polaris 80.

L'avènement des super-calculateurs de bureau



L'avènement des super-calculateurs de bureau

Le coût caché des nouvelles technologies

Développer pour de telles architectures est complexe :

- Multiplicité des modèles de programmation.
- Multiplicité des technologies.
- Interopérabilité.

L'avènement des super-calculateurs de bureau

Le coût caché des nouvelles technologies

Développer pour de telles architectures est complexe :

- Multiplicité des modèles de programmation.
- Multiplicité des technologies.
- Interopérabilité.

Comment permettre à un non-expert de profiter de cette puissance ?

L'avènement des super-calculateurs de bureau

Le coût caché des nouvelles technologies

Développer pour de telles architectures est complexe :

- Multiplicité des modèles de programmation.
- Multiplicité des technologies.
- Interopérabilité.

Comment permettre à un non-expert de profiter de cette puissance ?

Constat

La création d'outils pour la programmation de ces machines complexes est un verrou technologique majeur et d'une importance stratégique tant académique qu'industriel.

Plan de l'exposé

"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

Le nœud gordien du programmeur

Pour un problème donné, on désire ...

- Un code simple à coder/maintenir
- Un code efficace
- Un support architectural transparent

Le nœud gordien du programmeur

Pour un problème donné, on désire ...

- Un code simple à coder/maintenir
- Un code efficace
- Un support architectural transparent

Solutions disponibles

- Langage + interpreteur ?
- Langage + compilateur ?
- Une bibliothèque ?

Approche à base de compilateurs

Principes

Un compilateur existant est modifié afin de prendre en compte les spécificités architecturales d'une cible donnée au sein des différentes étapes de la compilation.

Approche à base de compilateurs

Principes

Un compilateur existant est modifié afin de prendre en compte les spécificités architecturales d'une cible donnée au sein des différentes étapes de la compilation.

Limitations

- 1 cible = 1 compilateur ?
- Délai de maturité.
- Support et pérennité.
- Acceptation au sein de la communauté.

Approche à base de bibliothèques

Principes

- Collection de SdD et de fonctions associées.
- Favorise la réutilisation de modules compilés.
- Capitalise de l'Expertise Algorithmique.

Approche à base de bibliothèques

Principes

- Collection de SdD et de fonctions associées.
- Favorise la réutilisation de modules compilés.
- Capitalise de l'Expertise Algorithmique.

Limitations

- Pas (ou peu) d'optimisation inter-procédurale
- Explosion combinatoire des versions
- Code dépendant des TDA sous-jacents

Comment couper ce nœud ? ?

Le meilleur des deux mondes

- Compilateurs :

Comment couper ce nœud ? ?

Le meilleur des deux mondes

- Compilateurs :
  - Performances du code généré.
  - Modèles et algorithmes d'optimisation bien connus.

Comment couper ce nœud ? ?

Le meilleur des deux mondes

- Compilateurs :
  - Performances du code généré.
  - Modèles et algorithmes d'optimisation bien connus.
- Bibliothèques :

Comment couper ce nœud ? ?

Le meilleur des deux mondes

- Compilateurs :
  - Performances du code généré.
  - Modèles et algorithmes d'optimisation bien connus.
- Bibliothèques :
  - Interface adaptée au domaine considéré.
  - Facilité d'intégration.

Comment couper ce nœud ? ?

Le meilleur des deux mondes

- Compilateurs :
  - Performances du code généré.
  - Modèles et algorithmes d'optimisation bien connus.
- Bibliothèques :
  - Interface adaptée au domaine considéré.
  - Facilité d'intégration.

Mais des challenges subsistent

La prise en compte de l'architecture doit être effectué au plus tôt afin de réduire l'aspect combinatoire de l'implantation de ces outils.

La solution *Embedded Domain Specific Language*

Définition

Langage déclaratif dédié enfoui dans un langage impératif généraliste.

- *Ruby on Rails* pour Ruby
- Framework *JUnit* pour JAVA
- Les expressions régulières en PERL,C++, C ou .NET

La solution *Embedded Domain Specific Language*

Définition

Langage déclaratif dédié enfoui dans un langage impératif généraliste.

- *Ruby on Rails* pour Ruby
- Framework *JUnit* pour JAVA
- Les expressions régulières en PERL,C++, C ou .NET

Avantages

- Problèmes et solutions exprimés par un seul jeu de concepts
- Code facile à écrire, lire et à maintenir
- Chaîne de développement peu perturbée
- Efficace grâce à l'évaluation partielle

L'évaluation partielle en trois points

Un programme est une fonction Φ qui traite des données connues soit à la compilation soit à l'exécution et génère un résultat à l'exécution.

$$\Phi \ : \ \left(I_{static},\ I_{dynamic}\right) \rightarrow O$$

L'évaluation partielle en trois points

Un programme est une fonction Φ qui traite des données connues soit à la compilation soit à l'exécution et génère un résultat à l'exécution.

$$\Phi \ : \ (I_{static}, \ I_{dynamic}) \rightarrow O$$

Un évaluateur partiel Γ transforme $(\Phi, \ I_{static})$ en un programme résiduel Φ^* dans lequel I_{static} a été calculé à la compilation.

$$\Gamma \ : \ (\Phi, \ I_{static}) \rightarrow \Phi^*$$

L'évaluation partielle en trois points

Un programme est une fonction Φ qui traite des données connues soit à la compilation soit à l'exécution et génère un résultat à l'exécution.

$$\Phi \ : \ (I_{static}, \ I_{dynamic}) \rightarrow O$$

Un évaluateur partiel Γ transforme $(\Phi, \ I_{static})$ en un programme résiduel Φ^* dans lequel I_{static} a été calculé à la compilation.

$$\Gamma \ : \ (\Phi, \ I_{static}) \rightarrow \Phi^*$$

Φ^* et Φ sont fonctionnellement équivalent mais Φ^* a, en général, de meilleures performances à l'exécution que Φ.

$$\Phi \ \equiv \ \Phi^* \ : \ (I_{dynamic}) \rightarrow O$$

De l'évaluation partielle à la méta-programmation

Définition de la Méta-programmation

Écriture de programme qui analyse, transforme et/ou génère d'autres programmes (ou eux-mêmes)

De l'évaluation partielle à la méta-programmation

Définition de la Méta-programmation

Écriture de programme qui analyse, transforme et/ou génère d'autres programmes (ou eux-mêmes)

Langages disponibles

- *template* HASKELL
- `metaOcaml`
- C++

De l'évaluation partielle à la méta-programmation

Définition de la Méta-programmation

Écriture de programme qui analyse, transforme et/ou génère d'autres programmes (ou eux-mêmes)

Langages disponibles

- *template* HASKELL
- `metaOcaml`
- C++

De l'évaluation partielle à la méta-programmation

Définition de la Méta-programmation

Écriture de programme qui analyse, transforme et/ou génère d'autres programmes (ou eux-mêmes)

Langages disponibles

- *template* HASKELL
- `metaOcaml`
- C++

La méta-programmation template C++

- Templates et surcharge d'opérateurs approximent la syntaxe d'un DSL au sein de C++
- Les templates sont Turing-complet.

Plan de l'exposé

Dans *Domain Specific Language*, ...

... il y a *Language*

- Notion de grammaire explicite
- Nécessité de renforcer la syntaxe
- S'inspirer du fonctionnement classique d'un compilateur

Dans *Domain Specific Language*, ...

... il y a *Language*

- Notion de grammaire explicite
- Nécessité de renforcer la syntaxe
- S'inspirer du fonctionnement classique d'un compilateur

.. il y a *Domain* aussi !

- Sémantique forte des entités du domaine
- Écriture de règles de production
- S'inspirer des années de recherche en Théorie des Langages

Dans *Domain Specific Language*, ...

... il y a *Language*

- Notion de grammaire explicite
- Nécessité de renforcer la syntaxe
- S'inspirer du fonctionnement classique d'un compilateur

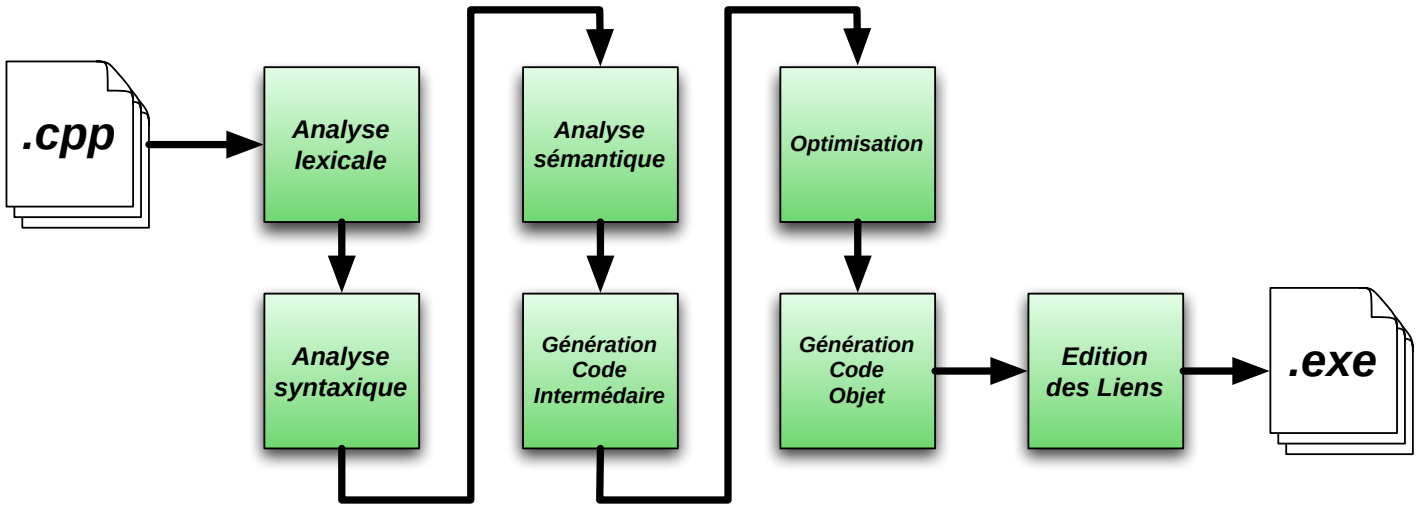
.. il y a *Domain* aussi !

- Sémantique forte des entités du domaine
- Écriture de règles de production
- S'inspirer des années de recherche en Théorie des Langages

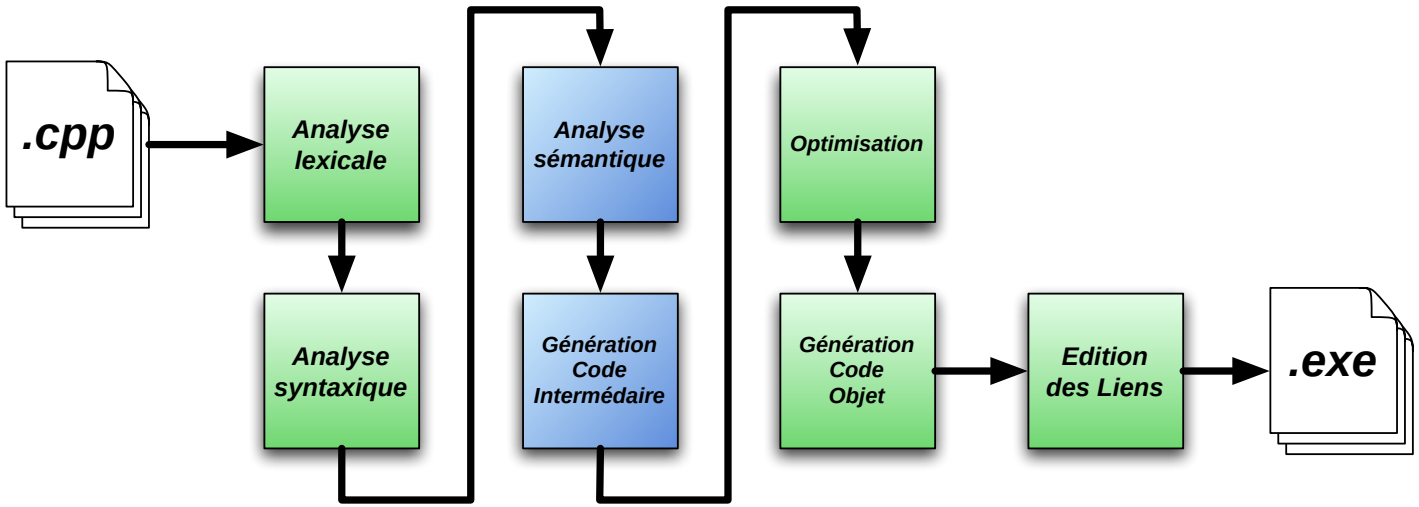
Constat

L'écriture de *EDSL* doit se défaire de son aura d' «astuce technique»

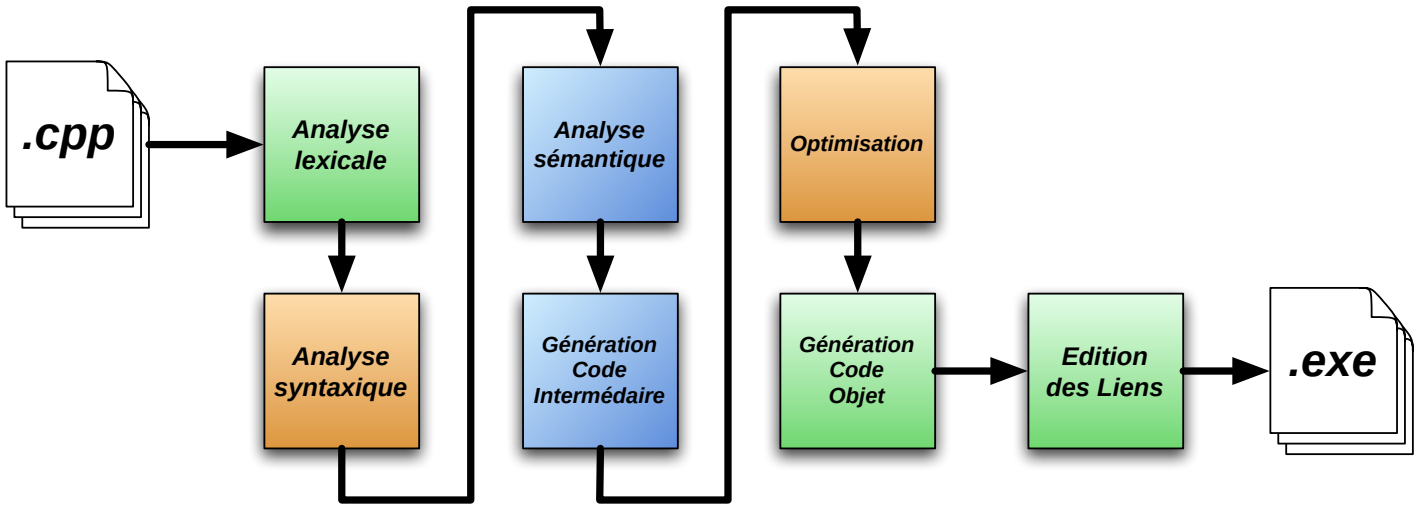
Du compilateur à l' *EDSL*



Du compilateur à l' *EDSL*



Du compilateur à l' *EDSL*



L'introspection en C++

Principes

Extraire des informations sur les *statements* du langage sous une forme directement manipulable au sein de ce dernier.

L'introspection en C++

Principes

Extraire des informations sur les *statements* du langage sous une forme directement manipulable au sein de ce dernier.

Mise en œuvre : Les Expression Templates

Utiliser la surcharge d'opérateurs et de fonction pour générer un objet intermédiaire dont le *type* encode l'*arbre de syntaxe abstraite* de la déclaration.

L'introspection en C++

Principes

Extraire des informations sur les *statements* du langage sous une forme directement manipulable au sein de ce dernier.

Mise en œuvre : Les Expression Templates

Utiliser la surcharge d'opérateurs et de fonction pour générer un objet intermédiaire dont le *type* encode l'*arbre de syntaxe abstraite* de la déclaration.

Notre approche

Revenir aux sources :

- 1 DSL = 1 modèle formel d'implantation
- 1 modèle = 1 jeu de méta-programmes

Des modèles formels au méta-programmes

Principes

- Les *éléments* d'un modèle formel sont représenté par des *types* au niveau implantation
- Les règles de sémantiques et les actions associées deviennent des *meta-programmes*

Des modèles formels au méta-programmes

Principes

- Les *éléments* d'un modèle formel sont représenté par des *types* au niveau implantation
- Les règles de sémantiques et les actions associées deviennent des *meta-programmes*

QUAFF

- DSL de programmation parallèles par squelettes
- Un squelette = structure parallèle récurrente
- Définition d'un modèle formel pour l'implantation de ces squelettes

Formal semantics applied to the implementation of a skeleton-based parallel programming library
J. Falcou and J. Sérot, ParCo 2007

Modèle formel de QUAFF

Définition

- $\Sigma ::= \text{Seq } f \mid \text{Pipe}(\Sigma_1, \ldots, \Sigma_n) \mid \text{Farm}(n, \Sigma) \mid \text{Scm}(n, f_s, \Sigma, f_m)$
- $f, f_s, f_m ::=$ fonction séquentielle
- $n ::= \textit{entier} \geq 1$

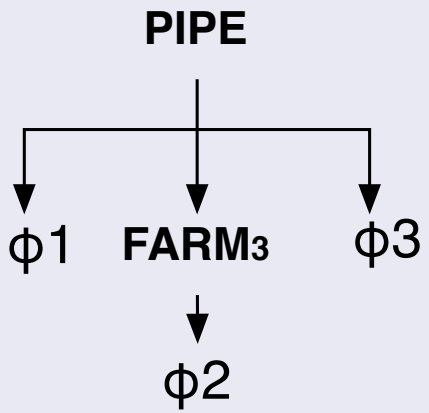
Modèle formel de QUAFF

Définition

- $\Sigma ::= \text{Seq } f \mid \text{Pipe}(\Sigma_1, \ldots, \Sigma_n) \mid \text{Farm}(n, \Sigma) \mid \text{Scm}(n, f_s, \Sigma, f_m)$
- $f, f_s, f_m ::=$ fonction séquentielle
- $n ::= \textit{entier} \geq 1$

Exemple

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



Modèle formel de QUAFF

Structure du réseau de processus

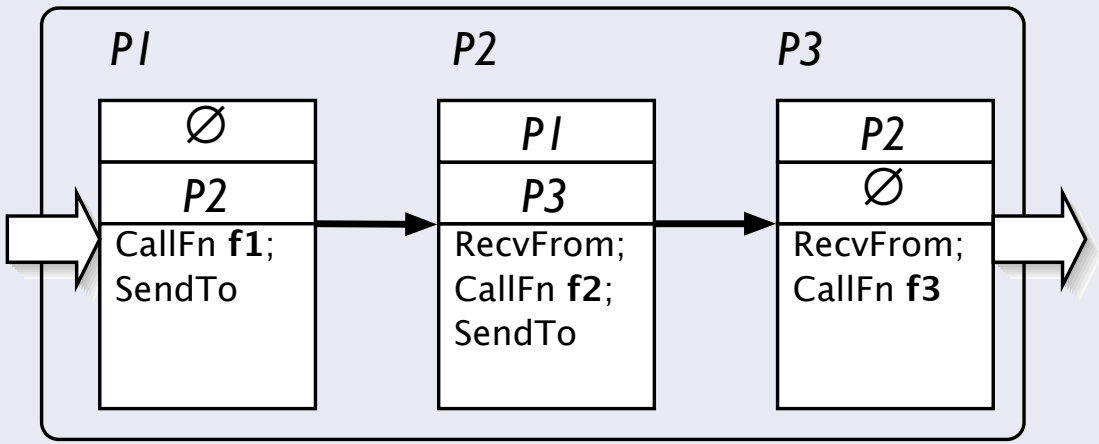
- Ensemble de processus communicant par des canaux dédiés
- Chaque processus connait ses *prédécesseurs* et *successeurs*
- Chaque processus exécute un séquence d'*instructions*

Modèle formel de QUAFF

Structure du réseau de processus

- Ensemble de processus communicant par des canaux dédiés
- Chaque processus connait ses *prédécesseurs* et *successeurs*
- Chaque processus exécute un séquence d'*instructions*

Exemple : RSPC pour **Pipeline**



Modèle formel de QUAFF

Un algèbre des processus

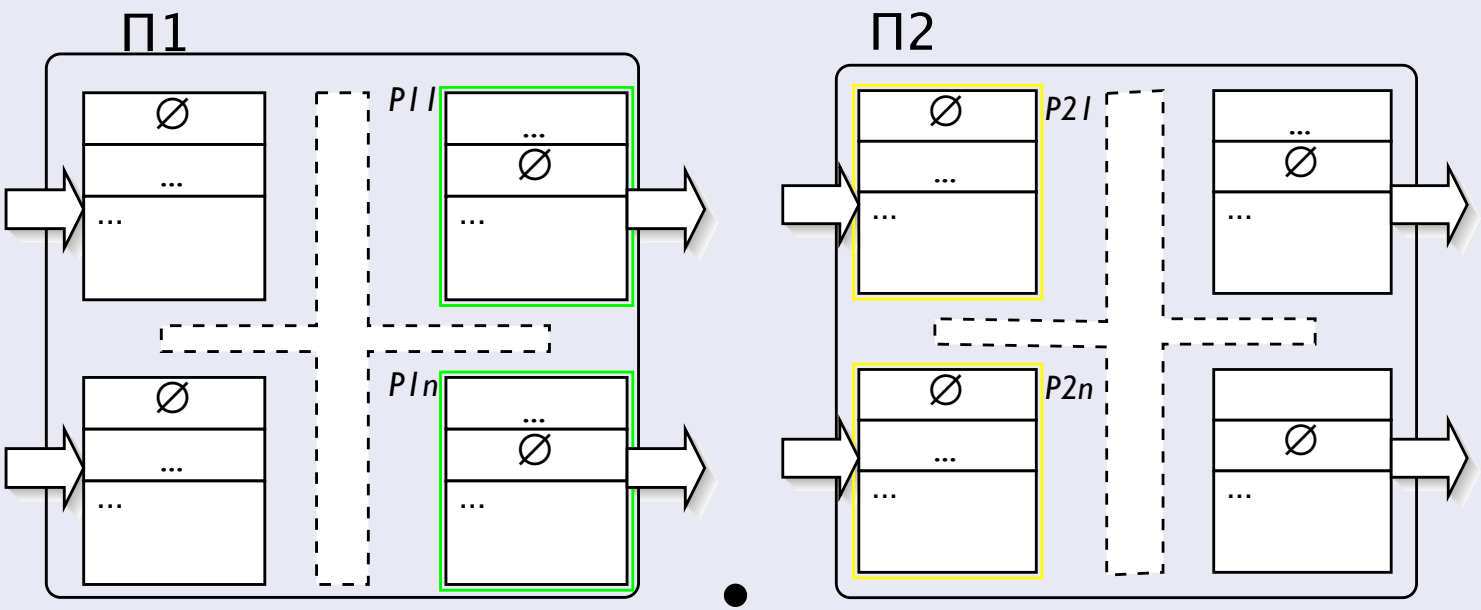
- Le RSPC est construit itérativement par une suite d'opérateurs
- Ces opérateurs sont définis par des règles de sémantique

Exemple : l'opérateur •

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

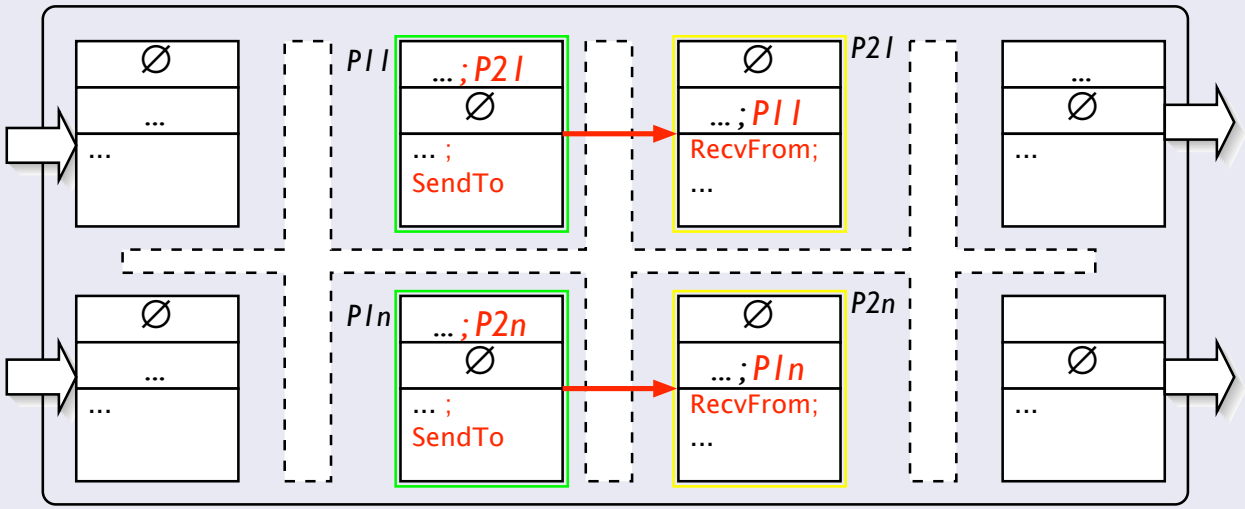
Modèle formel de QUAFF

Description informelle de •



Modèle formel de QUAFF

Description informelle de •



Π1 • Π2

Modèle formel de QUAFF

Du Squelette à un RSPC

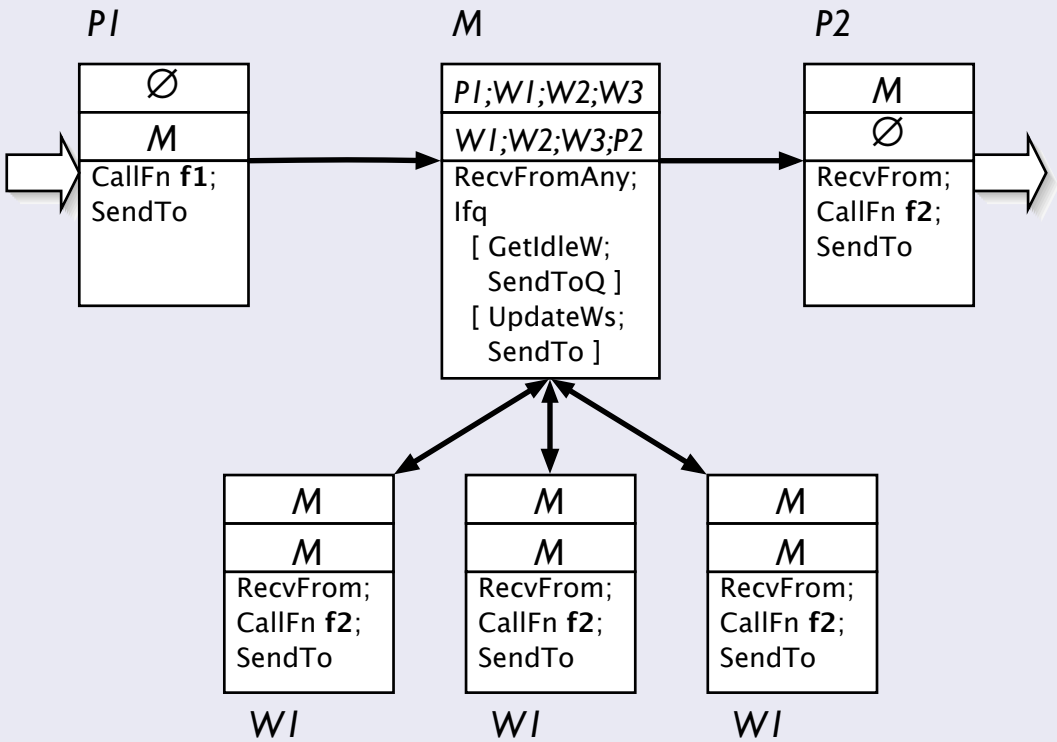
On définit une fonction de conversion $\mathcal{C}[[]]$ qui transforme un squelette en RSPC équivalent.

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

Modèle formel de QUAFF

Exemple de conversion

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



Du modèle formel au méta-programme

Comment méta-implanter ces règles ?

- Interface de construction des squelettes ;
- Génération d'un RSPC ;
- Génération du code C+MPI résiduel.

Du modèle formel au méta-programme

Comment méta-implanter ces règles ?

- Interface de construction des squelettes ;
- Génération d'un RSPC ;
- Génération du code C+MPI résiduel.

Qu'avons nous à disposition ?

- Les *Expression Templates* ;
- Spécialisation partielles des *templates* ;
- Injection de fragment de code *template* .

Du modèle formel au méta-programme

Comment méta-implanter ces règles ?

- Interface de construction des squelettes ;
- Génération d'un RSPC ;
- Génération du code C+MPI résiduel.

Qu'avons nous à disposition ?

- Les *Expression Templates* ;
- Spécialisation partielles des *templates* ;
- Injection de fragment de code *template* .

Du modèle formel au méta-programme

Comment méta-implanter ces règles ?

- Interface de construction des squelettes ;
- Génération d'un RSPC ;
- Génération du code C+MPI résiduel.

Qu'avons nous à disposition ?

- Les *Expression Templates* ;
- Spécialisation partielles des *templates* ;
- Injection de fragment de code *template* .

Du modèle formel au méta-programme

Comment méta-implanter ces règles ?

- Interface de construction des squelettes ;
- Génération d'un RSPC ;
- Génération du code C+MPI résiduel.

Qu'avons nous à disposition ?

- Les *Expression Templates* ;
- Spécialisation partielles des *templates* ;
- Injection de fragment de code *template* .

Génération du RSPC

Principes

Processus, descripteur et RSPC sont définis sous forme de classe *template* .

Exemple :

```
template<class P,class I,class O>
struct process_network
{
  typedef P   process;
  typedef I   inputs;
  typedef O   outputs;

};
```

Génération du RSPC

Principes

Processus, descripteur et RSPC sont définis sous forme de classe *template* .

Exemple :

```
template<class ID,class DESC, class IT, class OT>
struct process
{
  typedef ID       pid;
  typedef DESC     descriptor;
  typedef IT       input_type;
  typedef OT       output_type;
};
```

Génération du RSPC

Principes

Processus, descripteur et RSPC sont définis sous forme de classe *template* .

Exemple :

```
template<class IPID,class OPID,class CODE, class KIND>
struct descriptor
{
  typedef IPID  i_pids;
  typedef OPID  o_pids;
  typedef CODE  instrs;
  typedef KIND  kind;
};
```

Génération du RSPC

Génération du RSPC *template*

La fonction `run` utilise le pattern matching *template* sur l'AST afin de générer une instance du type *template* correspondant au RSPC.

La fonction *template* `run`

```
template<class SKL>
void run( const SKL& )
{
  typedef typename convert<SKL,
                           mpl::int_<0>
                          >::type network;
  network::Run();
}
```

Génération du RSPC

Génération du RSPC *template*

La fonction `run` utilise le pattern matching *template* sur l'AST afin de générer une instance du type *template* correspondant au RSPC.

La méta-fonction `convert` pour le `pipeline`

```
template<class S0,class S1,class ID>
struct convert<Serial<S0,S1>,ID>
{
  typedef Serial<S1,mpl::void_>                        tail;
  typedef typename convert<S0,ID>::type                proc1;
  typedef typename convert<S0,ID>::new_id              next_id;
  typedef typename convert<tail,next_id>::new_id       new_id;
  typedef typename convert<tail,next_id>::type         proc2;
  typedef typename rule_serial<proc1,proc2>::type      type;
};
```

Génération du RSPC

Génération du RSPC *template*

La fonction `run` utilise le pattern matching *template* sur l'AST afin de générer une instance du type *template* correspondant au RSPC.

La méta-fonction `rule_serial`

```
template<class P1, class P2> struct rule_serial
{
  typedef typename P1::process                                          proc1;
  typedef typename P2::process                                          proc2;
  typedef typename P1::inputs                                           i1;
  typedef typename P2::inputs                                           i2;
  typedef typename P1::outputs                                          o1;
  typedef typename P2::outputs                                          o2;
  typedef typename mpl::transform< proc1, phi_d<_1,o1,i2> >::type       np1;
  typedef typename mpl::transform< proc2, phi_s<_1,i2,o1> >::type       np2;
  typedef typename mpl::copy<np2, mpl::back_inserter<np1> >::type       process;
  typedef process_network<process,i1,o2>                                type;
};
```

Un exemple ...

Detection de contours

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

int main()
{
  run( seq(function(gui))
     & ( seq(load)
         | scm<4>( seq(b_split), seq(smooth), seq(b_merge) )
         | scm<4>( seq(b_split), seq(edges) , seq(b_merge) )
         | seq(save)
         )
     );
}
```

Plan de l'exposé

Contexte

Traitement d'images et parallélisme

- Nombreuses applications en temps interactif
- Plusieurs niveaux de parallélismes exploitables
- Performances limitées par la mémoire

Contexte

Traitement d'images et parallélisme

- Nombreuses applications en temps interactif
- Plusieurs niveaux de parallélismes exploitables
- Performances limitées par la mémoire

Architectures pour le Traitement d'images

- Machines multi-cœurs + extension multimédia SIMD
- Cartes graphiques programmables
- Machines hétérogènes sur puce

Contexte

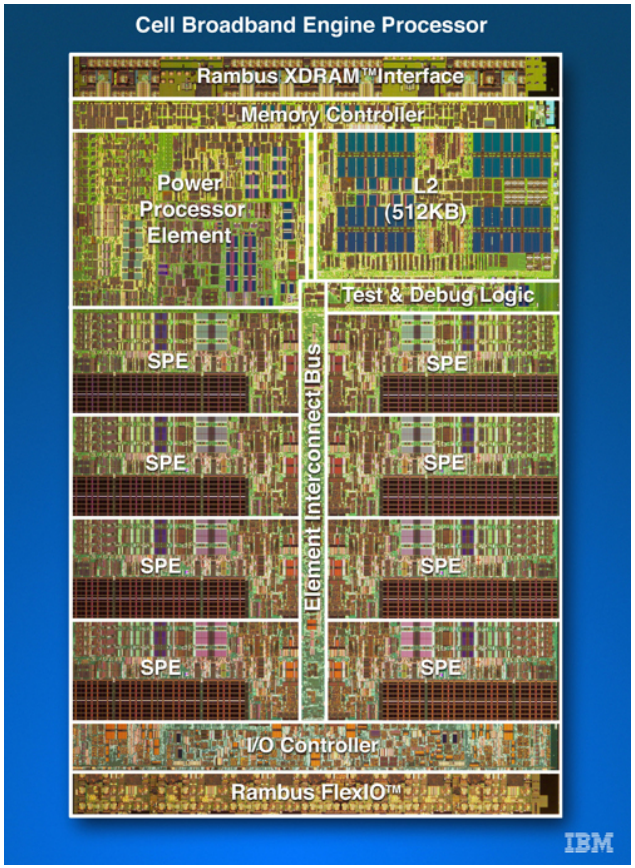
Traitement d'images et parallélisme

- Nombreuses applications en temps interactif
- Plusieurs niveaux de parallélismes exploitables
- Performances limitées par la mémoire

Architectures pour le Traitement d'images

- Machines multi-cœurs + extension multimédia SIMD
- Cartes graphiques programmables
- Machines hétérogènes sur puce

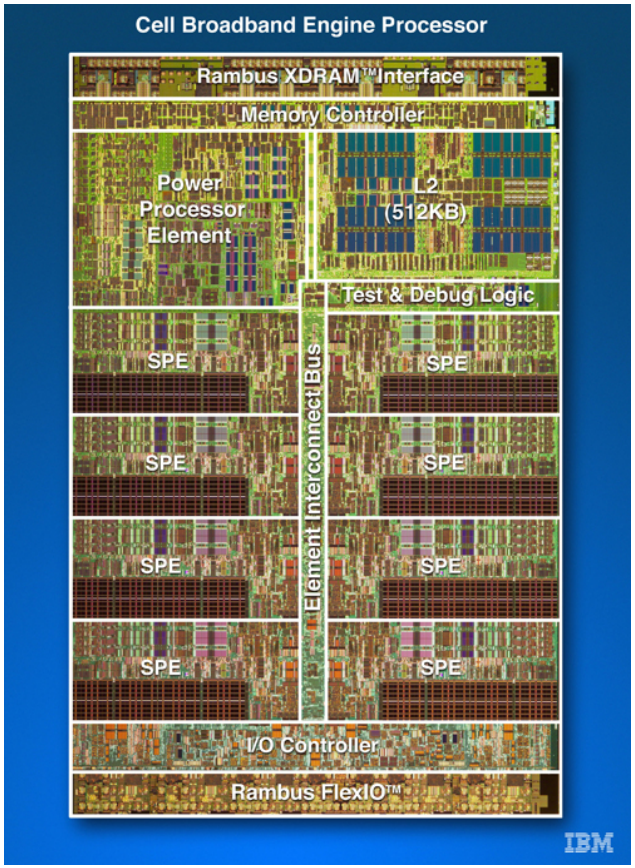
Le processeur CELL - Architecture



Spécificités

- 9 Processeurs :
  1 PPE + 8 SPE
- Bus d'inter-connection à 200 GB/s
- Support multi-thread
- Support SIMD

Le processeur CELL - Architecture



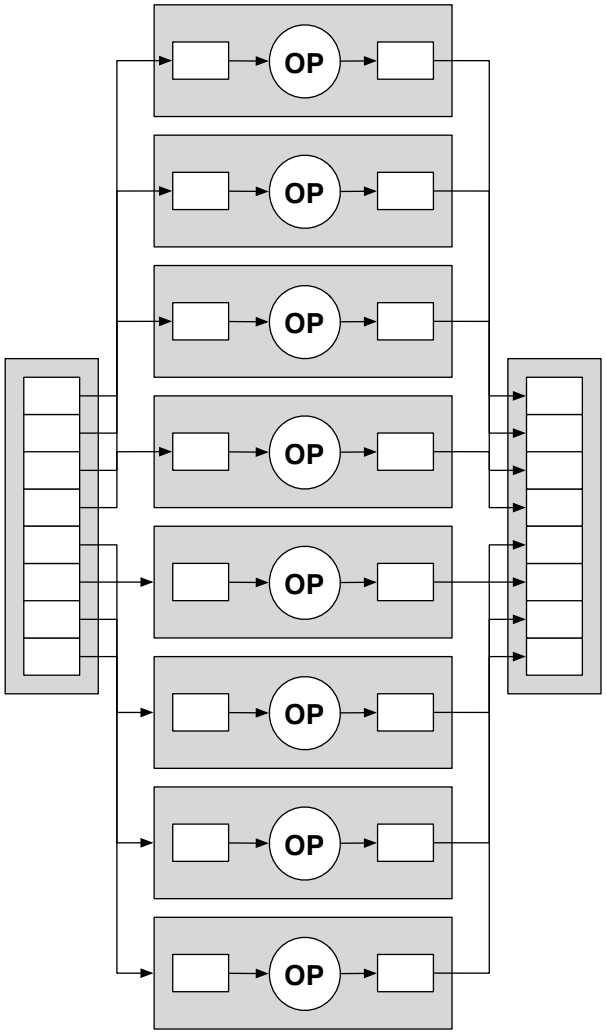
Spécificités

- 9 Processeurs : 1 PPE + 8 SPE
- Bus d'inter-connection à 200 GB/s
- Support multi-thread
- Support SIMD

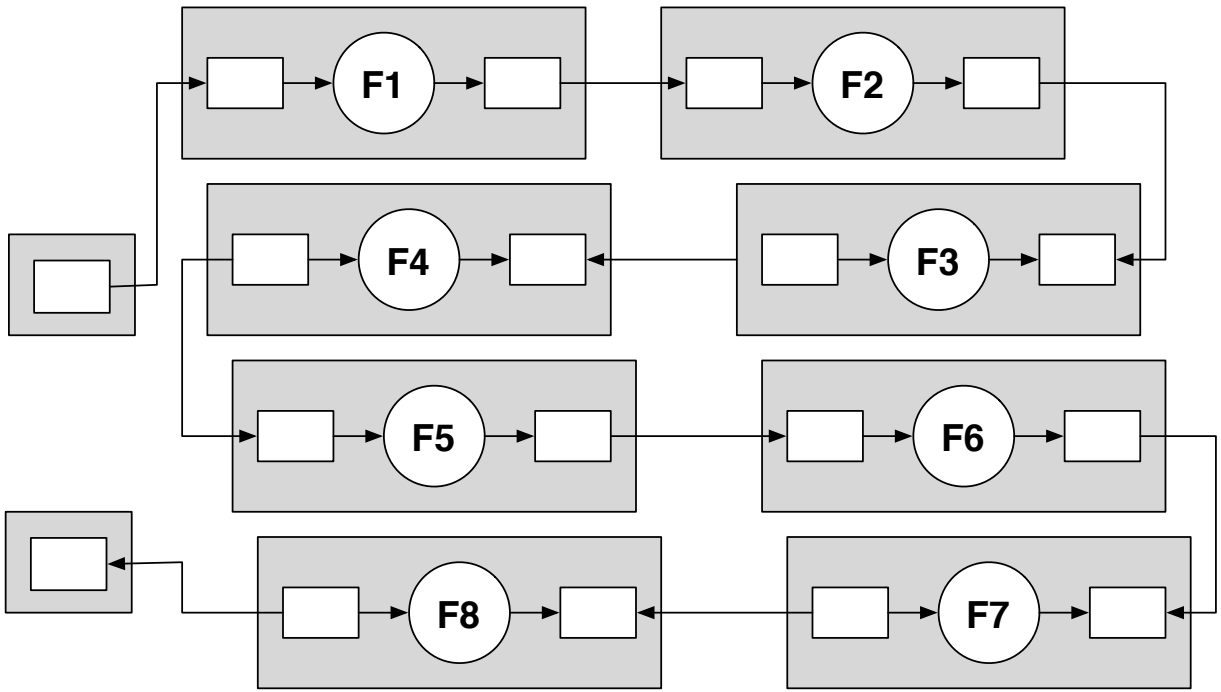
Développer pour le CELL

Comment déterminer le placement optimal d'une application ?

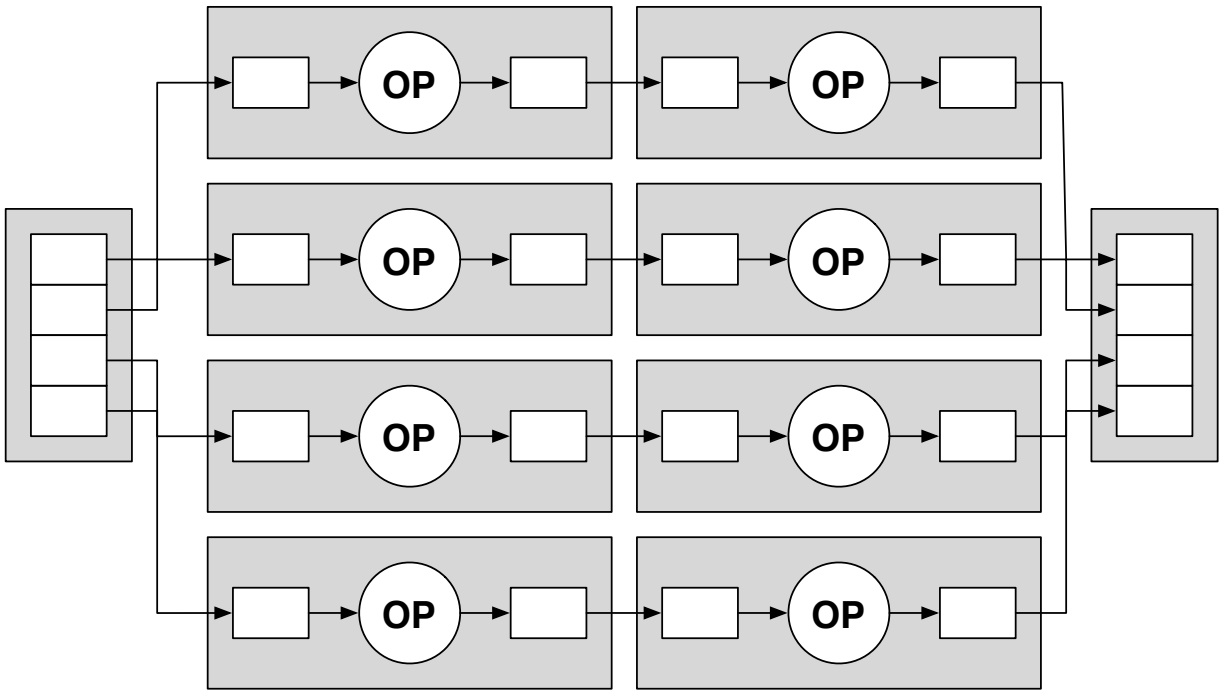
Développer pour le CELL - Quel modéle ?



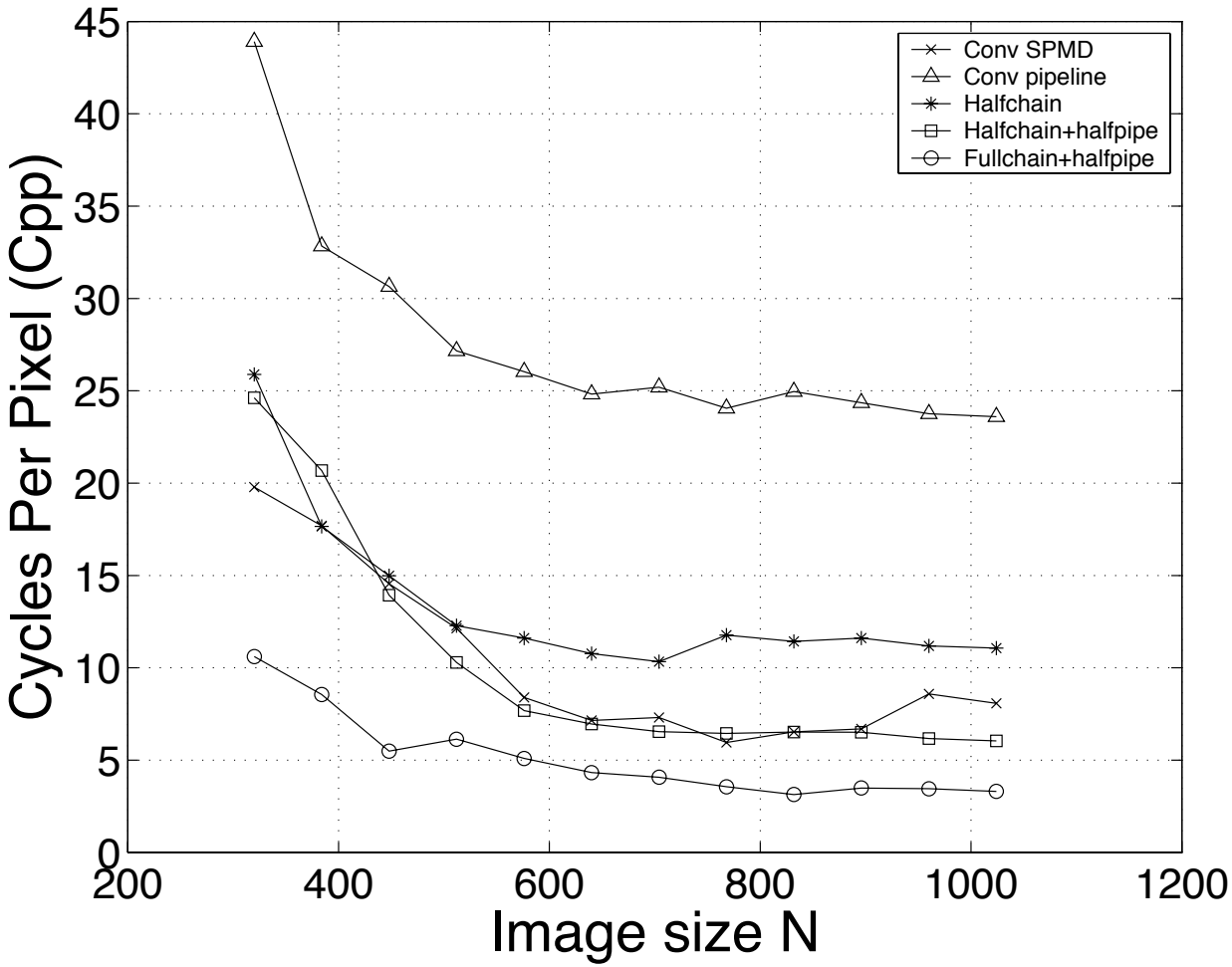
Développer pour le CELL - Quel modéle ?



Développer pour le CELL - Quel modéle ?



Développer pour le CELL - Quel résultat ?



Parallelization Schemes for Memory Optimization on the Cell Processor
T. Saidani, J. Falcou, L. Lacassagne, S. Bouaziz, HiPEAC Trans.

Définition de l'*EDSL* SKELL-BE

Squelettes identifiés

- *Pipe* : structure de type parallélisme de contrôle
- *Farm* : structure de type parallélisme de données
- *Seq* : structure de liaison

Définition de l'*EDSL* SKELL-BE

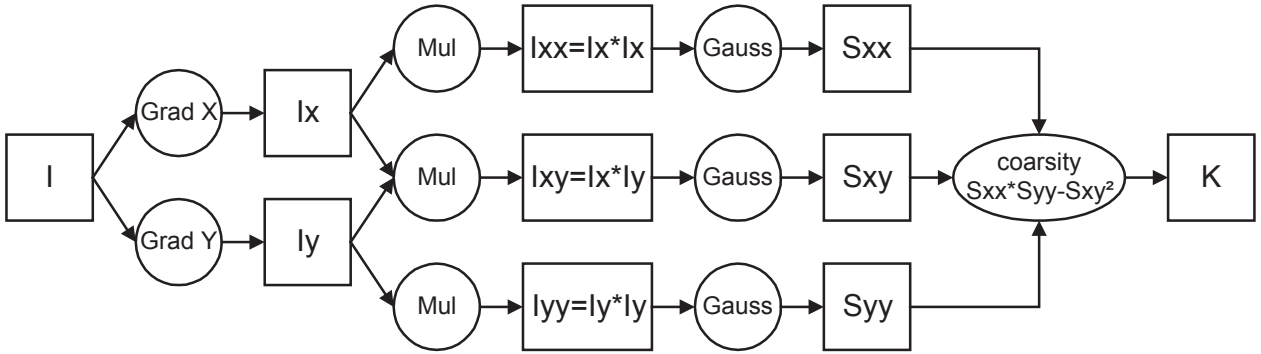
Squelettes identifiés

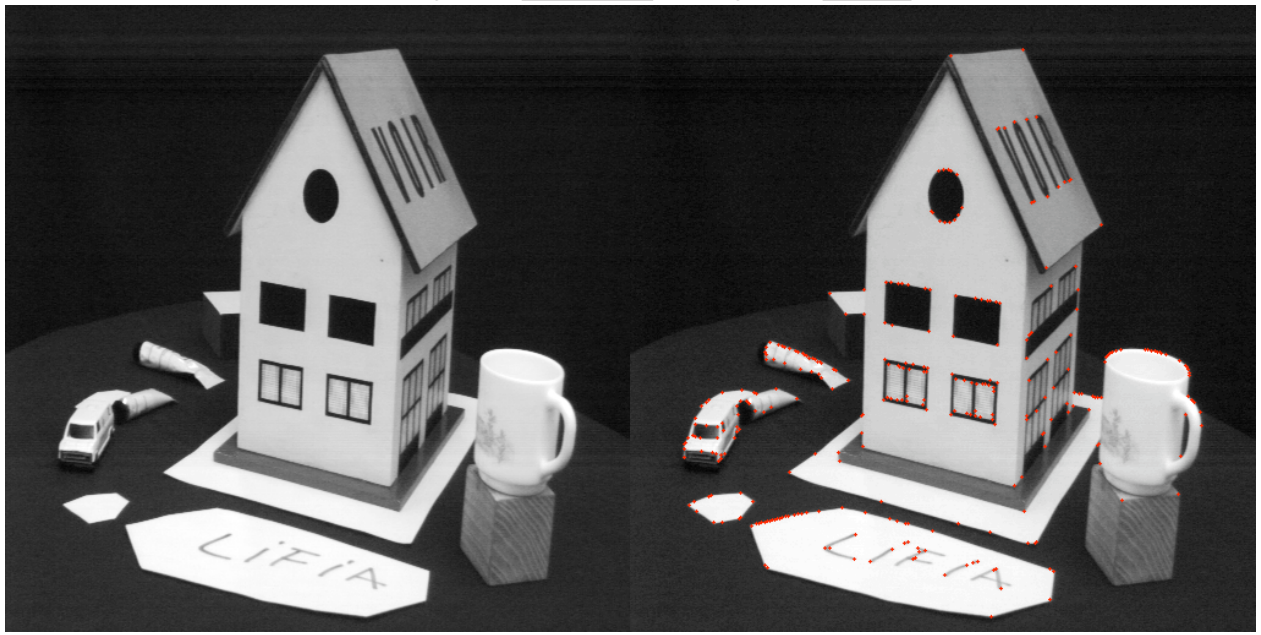
- *Pipe* : structure de type parallélisme de contrôle
- *Farm* : structure de type parallélisme de données
- *Seq* : structure de liaison

Grammaire associée

$$
\begin{array}{lcl}
\mathcal{A} & ::= & \text{run } \Sigma \\
\Sigma & ::= & \Gamma \mid \textit{Farm } n\ \Gamma \\
\Gamma & ::= & \textit{Seq } f \mid \textit{Pipe } \Gamma_1 \ldots \Gamma_n \\
f & ::= & \textit{fonctions C++ définies par l'utilisateur} \\
n & ::= & \textit{entier} \geq 1
\end{array}
$$

Détection de point d'intérêt de Harris





Détection de point d'intérêt de Harris

Code PPE

```
#include <quaff/quaff.hpp>
using namespace quaff;
QUAFF_REGISTER_KERNEL(harris_kernel);

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

  run(harris_kernel)(in)(out);
}
```

Détection de point d'intérêt de Harris

Code SPE - Pipeline classique

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

QUAFF_BEGIN_KERNEL(harris_kernel)
{
  run( farm<2>( seq(Sobel) | seq(Mul) | seq(Gauss) | seq(Coarsity)) );
}
QUAFF_END_KERNEL(harris_kernel)
```

Détection de point d'intérêt de Harris

Code SPE - Half-chain

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

QUAFF_BEGIN_KERNEL(harris_kernel)
{
  run( farm<4>( (seq(Sobel),seq(Mul)) | (seq(Gauss),seq(Coarsity))) );
}
QUAFF_END_KERNEL(harris_kernel)
```

Détection de point d'intérêt de Harris

Code SPE - Full chain

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

QUAFF_BEGIN_KERNEL(harris_kernel)
{
  run( farm<8>( (seq(Sobel),seq(Mul),seq(Gauss),seq(Coarsity)) );
}
QUAFF_END_KERNEL(harris_kernel)
```

Détection de point d'intérêt de Harris

Performances temporelles

Mode	Classique	Half chain	Full chain
Manuel	27.67 cpp	11.26 cpp	8.34 cpp
Automatique	28.07 cpp	11.36 cpp	8.42 cpp
Surcoût	1.46 %	0.8%	0.9%

Programmation par squelettes algorithmiques pour le processeur CELL
J. Falcou, T. Saidani, L. Lacassagne, D. Etiemble, SYMPA08

Détection de point d'intérêt de Harris

Performances temporelles

Mode	Classique	Half chain	Full chain
Manuel	27.67 cpp	11.26 cpp	8.34 cpp
Automatique	28.07 cpp	11.36 cpp	8.42 cpp
Surcoût	1.46 %	0.8%	0.9%

Impact sur la taille de l'exécutable

- Code de gestion $\approx$ 2Ko
- Code d'interfaçage $\approx$ 1Ko par opérateur
- Implantation manuelle : exécutable de 8Ko sur chaque SPE
- Implantation automatique : exécutable de 70Ko sur chaque SPE

Programmation par squelettes algorithmiques pour le processeur CELL
J. Falcou, T. Saidani, L. Lacassagne, D. Etiemble, SYMPA08

Plan de l'exposé

"I always knew C++ templates were the work of the Devil, and now I'm sure..."

- Cliff Click, *Lead Architect* de la JVM Sun

Synthèse

Vers le *Teraflops Off The Shelves*

- Besoin d'outils simples
- Besoin d'outils efficaces

Synthèse

Vers le *Teraflops Off The Shelves*

- Besoin d'outils simples
- Besoin d'outils efficaces

Une solution : les *EDSLs*

- Adaptée aux utilisateurs
- Adaptable aux architectures à venir
- Une méthodologie est nécessaire

Synthèse

Vers le *Teraflops Off The Shelves*

- Besoin d'outils simples
- Besoin d'outils efficaces

Une solution : les *EDSLs*

- Adaptée aux utilisateurs
- Adaptable aux architectures à venir
- Une méthodologie est nécessaire

Contributions

- Formalisation des stratégies de parallélisation
- Adaptation aux spécificités du CELL
- Démonstration de l'efficacité de cette approche

Perspectives

Un *DSL* pour les règles de sémantiques ?

- Version statique de *Spirit* ?
- Comment méta-définir des TDA ?
- Existe-il une sémantique des règles de sémantique ?

Perspectives

Un *DSL* pour les règles de sémantiques ?

- Version statique de *Spirit* ?
- Comment méta-définir des TDA ?
- Existe-il une sémantique des règles de sémantique ?

Un *DSL* pour les architectures ?

- Définir une architecture par ses composants
- Définir la nature d'une architecture
- Profil UML MARTES (Thales Research) ? ?

Perspectives

Un *DSL* pour les règles de sémantiques ?

- Version statique de *Spirit* ?
- Comment méta-définir des TDA ?
- Existe-il une sémantique des règles de sémantique ?

Un *DSL* pour les architectures ?

- Définir une architecture par ses composants
- Définir la nature d'une architecture
- Profil UML MARTES (Thales Research) ? ?

Des solutions plus exotiques ?

- PIPS (http ://www.cri.ensmp.fr/ pips/) ? ?
- Le langage D ? ?

Merci de votre attention.