Modèle de Compilation d'HPF pour la Machine MIMD à Bancs Mémoire et Réseau Distribué Programmable Phénix

Pierre FIORINI

DSIT/DAPSA/PTO &
Établissement Technique Central de l'Armement
`fiorini@etca.fr`

François IRIGOIN
Ronan KERYELL

Centre de Recherche en Informatique de l'École des Mines de Paris
`{irigoin,keryell}@cri.ensmp.fr`

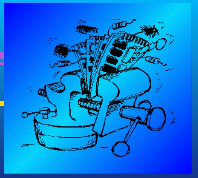
22 mai 1996

Plan

- Machine PHÉNIX
- Modèle d'exécution
- Modèle de compilation

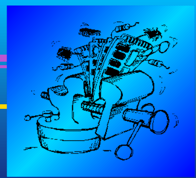
Introduction

- Applications de type traitement du signal lourd
- Parallélisme pour la puissance de calcul
- Débit entrées-sorties important
- Débit mémoire important
- Processeurs et composants si possible du commerce
- Environnement de programmation & compilation



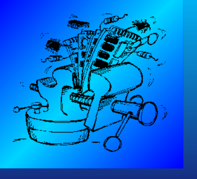
Qualité de la mémoire

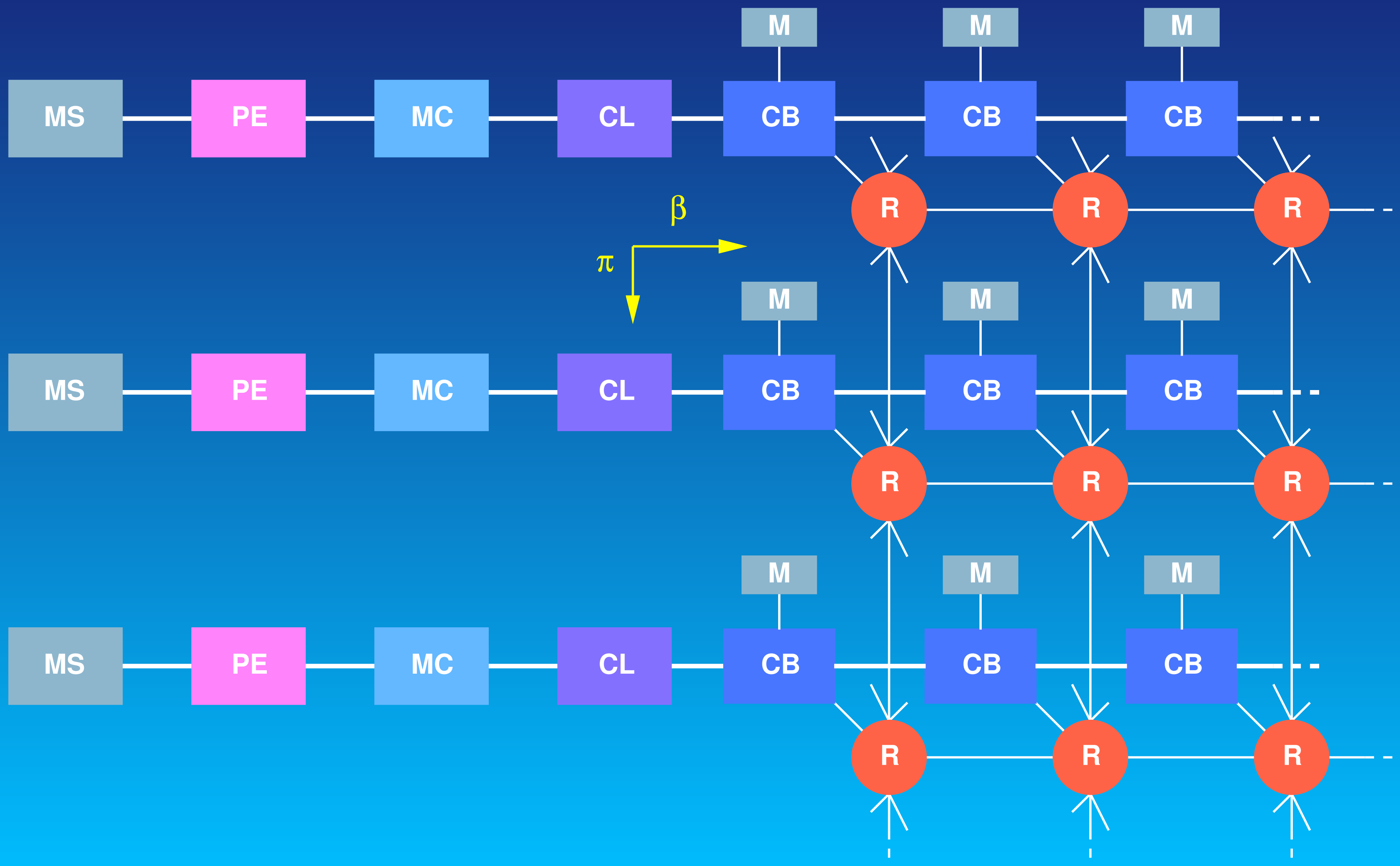
- Modèle Von NEUMANN : transferts *processeur* ⇆ *mémoire*
- Augmentation du débit mémoire :
  - Quantité ≡ débit de données
    - Cache (cohérence spatio-temporelle)
    - Fort débit par bloc sur mémoire et réseau
    - Bancs mémoire
  - Qualité ≡ débit d'adresses
    - Efforts de compilation & programmation
    - Bancs mémoire & matériel assurant le débit d'adresse (Cray...)
    - Calculer les adresses en local et réindexer en local

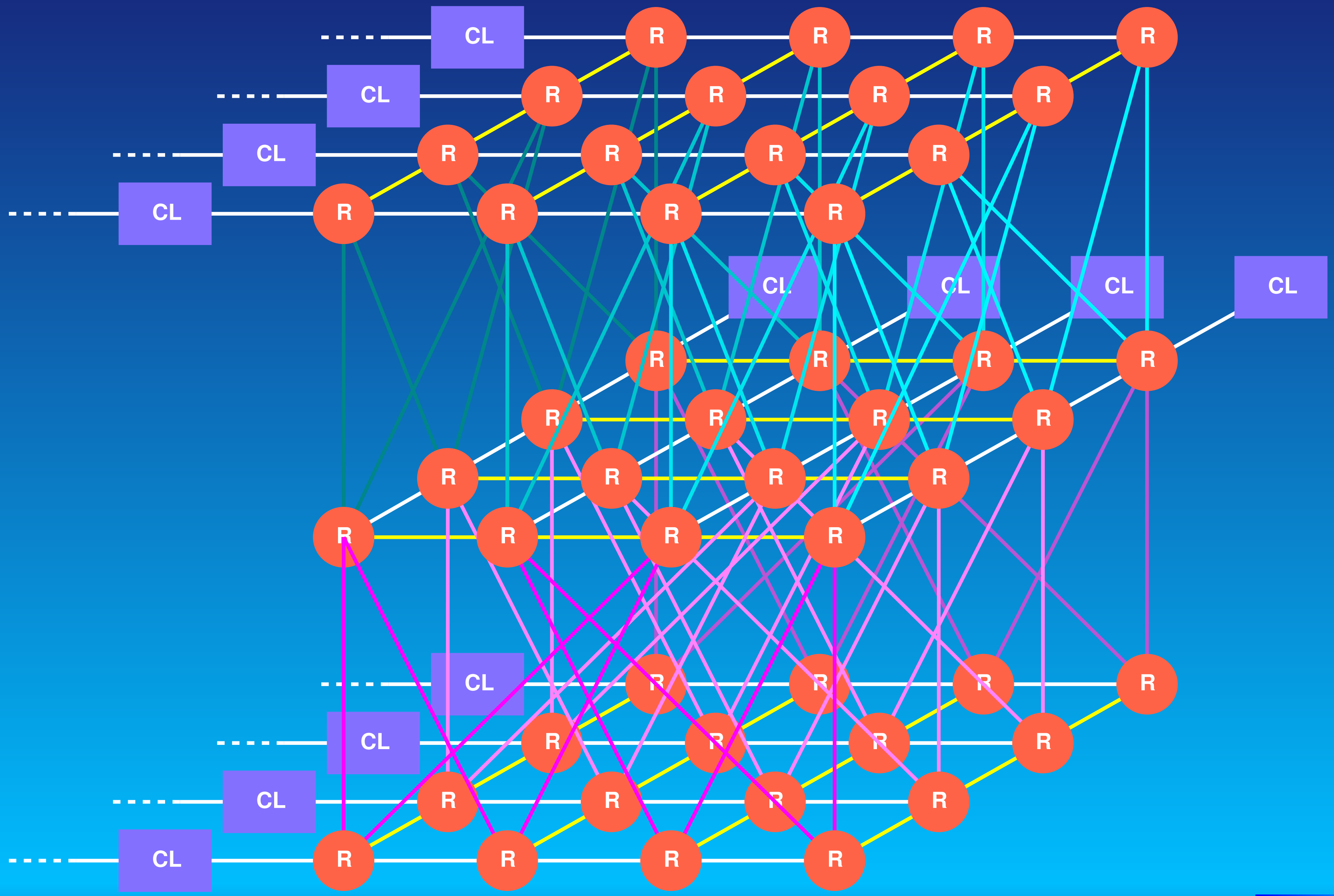


La machine PHÉNIX

- ||isme à mémoire distribué
- Processeurs du commerce PE avec mémoire scalaire MS et cache MC
- Rajout de matériel pour tout aligner dans le cache :
  - MC liée via un contrôleur de ligne CL à des bancs mémoire M
  - Chaque contrôleur de banc CB génère des adresses
  - Chaque CB a un contrôleur réseau R qui génère des adresses
  - Bus pipeline
- Réseau grille 2D
- Réseau Ω-pyramide entre plans
- Tout est programmable...




M
M
M
MS
PE
MC
CL
CB
CB
CB
R
R
R
β
π
M
M
M
MS
PE
MC
CL
CB
CB
CB
R
R
R
M
M
M
MS
PE
MC
CL
CB
CB
CB
R
R
R


CL
CL
CL
CL
R
CL
CL
CL
CL
CL
CL
CL
CL
CL
CL
CL
CL

- Modèle d'exécution
- Modèle de compilation
- Générer du code pour PE, CL, CB, R
- Choisir un langage
- Expression du programmeur (conflicts de bancs)
- Rétroaction sur le développement matériel $\leadsto$ co-développement

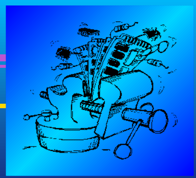
Langage

- Parallèle
- Informations de placement
- Adapté à PHÉNIX

¿Yet Another Language?

⇝HPF

- ▶ Partial... Compilateur développé au CRI
- ▶ Nids de boucle, alignements, distributions
- ▶ Bancs mémoire ≡ une dimension de *processors* virtuels



Hypothèses de compilation

- Nids de boucles parfaitement imbriqués à bornes affines
  - Compilation + simple
  - Souvent majorité du (data-)||isme disponible
  - Transformations de programme pour normaliser
- *Owner computes rule*
  - Distribue les calculs en fonction des destinations
  - Contournable par temporaire

`!HPF$INDEPENDENT(`i`)`

`forall(`$Li \leq b_0(n)$`)`

`X(`$S_{\mathtt{X}} i + a_{\mathtt{X}0}(n)$`) = f(Y(`$S_{\mathtt{Y}} i + a_{\mathtt{Y}0}(n)$`),`

`Z(`$S_{\mathtt{Z}} i + a_{\mathtt{Z}0}(n)$`),`

`...)`

`real A(0:24), B(0:24) !` $0 \leq a_{\mathtt{A}} \leq 24,\ 0 \leq a_{\mathtt{B}} \leq 24$

`!HPF$ template T(0:80) !` $0 \leq t \leq 80$

`!HPF$ processors P(0:3) !` $0 \leq p \leq 3$

`!HPF$ align A(i) with T(3*i) !` $a_{\mathtt{A}} = 3\ t$

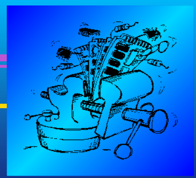
`!HPF$ align B(i) with A(i) !` $a_{\mathtt{A}} = a_{\mathtt{B}}$

`!HPF$ distribute T(cyclic(4)) onto P !` $t = 16c + 4p + \ell$

`!` $0 \leq \ell < 4$

`A(0:U:3) = A(0:U:3) + B(1:U+1:3) !` $i = 3i',\ 0 \leq i \leq \mathtt{U}$

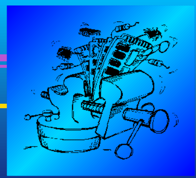
`!` $a = i$



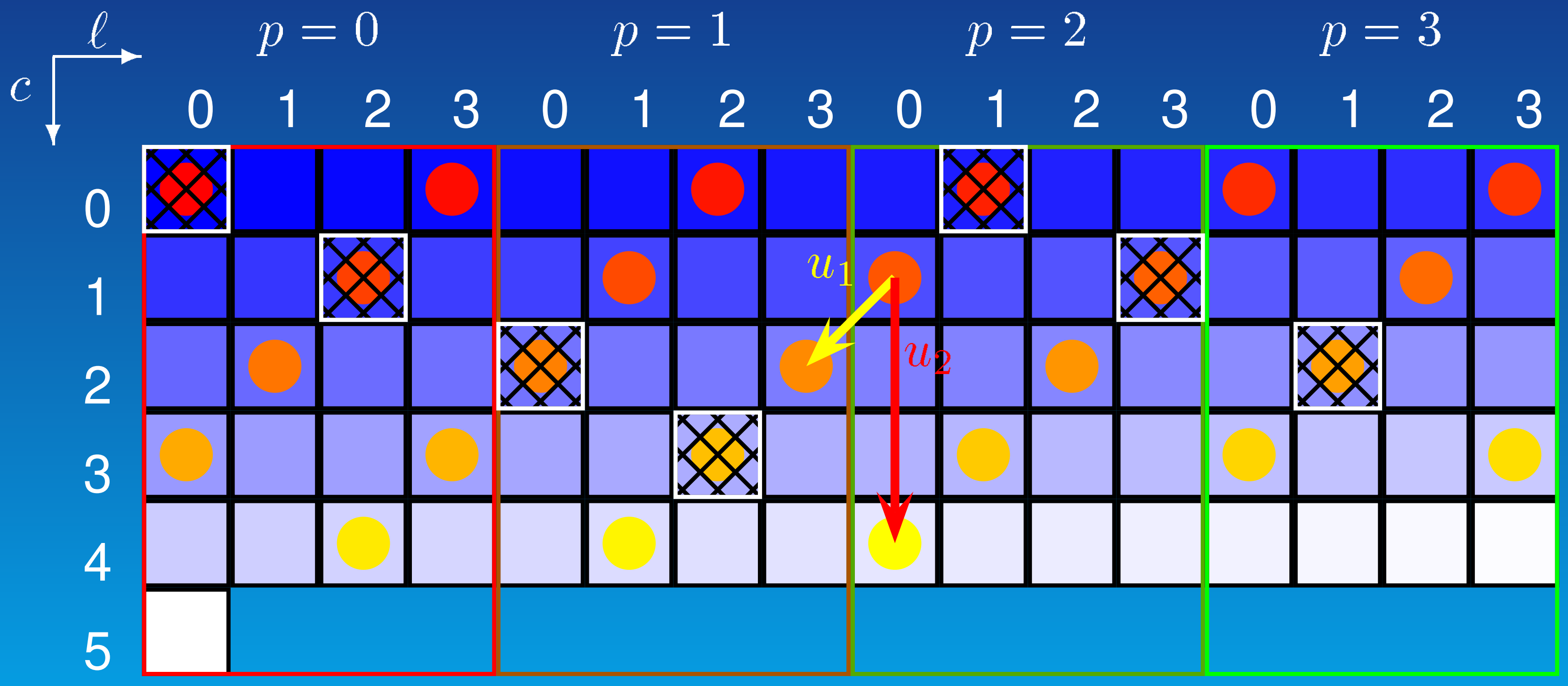
HPF dans un formalisme linéaire

- Algèbre linéaire : formalisme pratique
- Les directives d'HPF rentrent dans le formalisme :

Alignement	$Rt = Aa + s_0$
Distribution	$\Pi t = CPc + Cp + \ell$
Référence affine	$a = Si + a_0(n)$
Taille des blocs	$0 \leq \ell < C.1$
Taille des tableaux	$0 \leq a < D.1$
Taille des *processors*	$0 \leq p < P.1$
Taille des *templates*	$0 \leq t < T.1$



Par exemple $\mathbb{U} = 20$:



: template, : tableau, : calcul

- Éléments de tableau possédés :

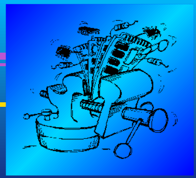
$$
\begin{aligned}
\mathit{Own}_{\mathtt{X}}(p) = \{a \mid \exists t, \exists c, \exists \ell :\ & R_{\mathtt{X}} t = A_{\mathtt{X}} a + t_{\mathtt{X}0} \\
& \wedge\ \Pi t = C_{\mathtt{X}} P c + C_{\mathtt{X}} p + \ell_{\mathtt{X}} \wedge\ 0 \leq a < D_{\mathtt{X}} \\
& \wedge\ 0 \leq p < P \wedge\ 0 \leq \ell < C_{\mathtt{X}} \\
& \wedge\ 0 \leq t < T_{\mathtt{X}}\}
\end{aligned}
$$

- Itérations calculées :

$$\mathit{Compute}(p) = \{i \mid S_{\mathtt{X}} i + a_{\mathtt{X}0} \in \mathit{Own}_{\mathtt{X}}(p)\}$$

- Éléments nécessaires au calcul :

$$
\begin{aligned}
\mathit{View}_{\mathtt{Y}}(p) = \{a \mid \exists i \in \mathit{Compute}(p) :& \\
& a = S_{\mathtt{Y}} i + a_{\mathtt{Y}0}\}
\end{aligned}
$$



Définition de quelques ensembles

- Émissions & réceptions :

$$\mathit{Send}_\Upsilon(p) = \{(p', a) \mid a \in \mathit{Own}_\Upsilon(p) \cap \mathit{View}_\Upsilon(p')\}$$

$$\mathit{Receive}_\Upsilon(p) = \{(p', a) \mid a \in \mathit{View}_\Upsilon(p) \cap \mathit{Own}_\Upsilon(p')\}$$

(Banc β, ligne de processeurs $\pi_1, \ldots, \pi_{n-1}) \equiv p = (p_1, p_2, \ldots, p_n)$

$$
\begin{array}{l}
\texttt{array X}'((c,\ell) \in \mathit{Own}_{\texttt{X}}(p))\texttt{,}\\
\qquad\quad \texttt{Y}'((c,\ell) \in \mathit{View}_{\texttt{Y}}(p))\texttt{,}\\
\qquad\quad \texttt{Z}'((c,\ell) \in \mathit{View}_{\texttt{Z}}(p))\\
\texttt{array X}''(\alpha \in \mathit{Flush}_{\texttt{X}}(p))\texttt{,}\\
\qquad\quad \texttt{Y}''(\alpha \in \mathit{Fill}_{\texttt{Y}}(p))\texttt{,}\\
\qquad\quad \texttt{Z}''(\alpha \in \mathit{Fill}_{\texttt{Z}}(p))\\
\texttt{forall(U} \in \{\texttt{Y}', \texttt{Z}', \ldots\}\texttt{)} \qquad\qquad 1^{\circ}\\
\qquad\quad \texttt{forall(}(p,p'), p \neq p', \mathit{Send}_{\texttt{U}}(p,p') \neq \varnothing\texttt{)}\\
\qquad\qquad\quad \texttt{forall(}(\ell,c) \in \mathit{Send}_{\texttt{U}}(p,p')\texttt{)}\\
\qquad\qquad\qquad\quad \texttt{send(}p', \texttt{U}(\ell,c)\texttt{)}\\
\texttt{forall(U} \in \{\texttt{Y}', \texttt{Z}', \ldots\}\texttt{)} \qquad\qquad 2^{\circ}\\
\qquad\quad \texttt{forall(}(p,p'), p \neq p', \mathit{Receive}_{\texttt{U}}(p,p') \neq \varnothing\texttt{)}\\
\qquad\qquad\quad \texttt{forall(}(\ell,c) \in \mathit{Receive}_{\texttt{U}}(p,p')\texttt{)}\\
\qquad\qquad\qquad\quad \texttt{U}(\ell,c) \texttt{ = receive(}p'\texttt{)}
\end{array}
$$

Schéma de compilation

$\texttt{forall}(\mathtt{U} \in \{\mathtt{Y}', \mathtt{Z}', \ldots\})$ 3°

$\quad\quad \texttt{forall}((\ell, c) \in \mathit{View}_{\mathtt{U}}(p))$

$\quad\quad\quad\quad \texttt{send\_to\_cache}(\mathtt{U}(\ell, c))$

$\texttt{forall}(\mathtt{U} \in \{\mathtt{Y}'', \mathtt{Z}'', \ldots\})$ 4°

$\quad\quad \texttt{forall}(\alpha \in \mathit{Fill}_{\mathtt{U}}(\pi))$

$\quad\quad\quad\quad \texttt{receive\_from\_memory}(\mathtt{U}(\alpha))$

$\texttt{forall}(\alpha \in \mathit{Compute}(\pi))$ 5°

$\quad\quad \mathtt{X}''(S_{\mathtt{X}''}(\alpha)) = \mathtt{f}(\mathtt{Y}''(S_{\mathtt{Y}''}(\alpha)),$

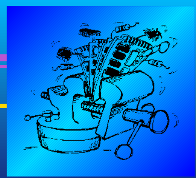
$\quad\quad\quad\quad\quad\quad\quad\quad \mathtt{Z}''(S_{\mathtt{Z}''}(\alpha)), \ldots)$

$\texttt{forall}(\alpha \in \mathit{Flush}_{\mathtt{X}}(\pi))$ 6°

$\quad\quad \texttt{send\_to\_memory}(\mathtt{X}''(\alpha))$

$\texttt{forall}((\ell, c) \in \mathit{Write}_{\mathtt{X}}(p))$ 7°

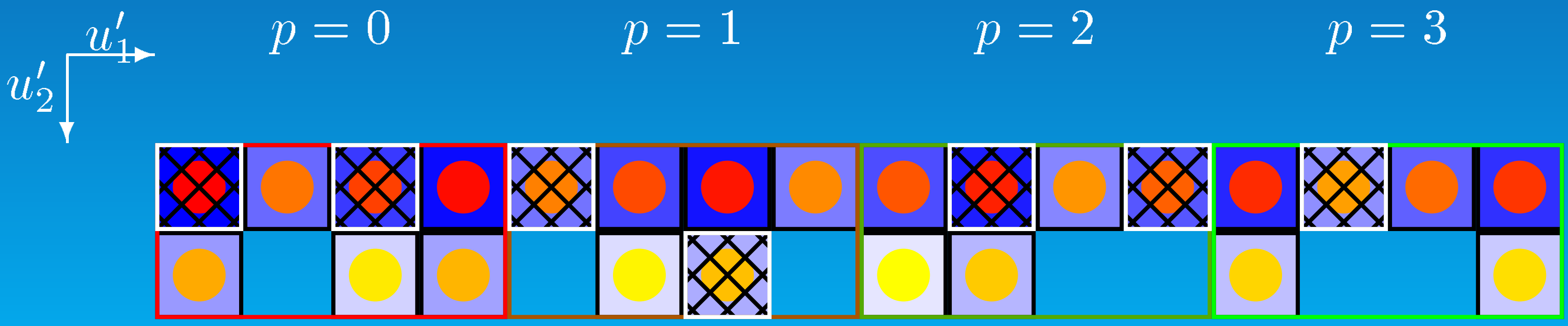
$\quad\quad \texttt{receive\_from\_cache}(\mathtt{X}'(\ell, c))$



- Description implicite $\rightsquigarrow$ paramétrique des ensembles
- Énumérer par des nids de boucle, processeurs physiques en paramètre
- Optimisations :
  - Diminuer les communications
  - Économiser la mémoire
  - Caches de taille finie

Allocation dans les bancs mémoire

- HPF : autorise des trous
- Compromis temps-espace
- Méthode :
  - Base (c, ℓ)
  - Base du cristal (u_1, u_2) via HERMITE
  - « Fortranisation » du tableau en (u'_1, u'_2) par division entière



Allocation dans le cache

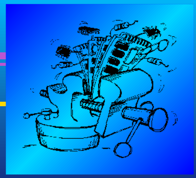
- Méthode similaire à la mémoire des processeurs : 1 ligne de cache = 1 ligne de c par exemple
- Dépend des capacités d'adressage du contrôleur de ligne :
  - Lecture/écriture consécutives entrelacées sur les bancs
  - Lecture/écriture entrelacées sur les bancs à des adresses arbitraires dans le cache

Contrôleur de ligne simple

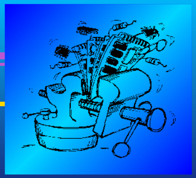
- Lecture/écriture consécutives entrelacées sur les bancs
- Cache $\equiv$ registres vectoriels gérés par *Fill* & *Flush*
- Énumération de *View* et *Write* dans la même base $\rightsquigarrow$ α = numéro d'ordre de transfert d'un élément
- PE $\equiv$ coprocesseur vectoriel
- Relation (rapide) entre (u'_1, u'_2) et α pour éviter des copies multiples :
  `A(0:U - 1) = A(0:U - 1) + A(1:U)`
  Transmettre les trous $\rightsquigarrow$ bijection $\alpha = U'_1 B u'_2 + B u'_1 + \beta$. Épluchage de boucle pour gérer le cycle c

Contrôleur de ligne avancé

- Lecture/écriture entrelacées sur les bancs à des adresses arbitraires dans le cache
- Évite de transférer des trous
- Énumère *Fill* & *Flush* ou fait suivre les adresses (!)
- Trous nécessaires pour gestion rapide des *overlaps*
- Gain néanmoins faible
- Mauvais rapport performance/coût



- 1° & 2° : $p \neq p'$ & $\pi \neq \pi'$ $\Longrightarrow$ communications au sein d'une même ligne
- Supprimer la communication avant chargement dans le cache
- Avoir un contrôleur de ligne
  - programmable pour remplacer la communication : adressage simple
  - simple mais adressage plus compliqué par le PE



Pipeline de l'usage du cache

- Modèle simple : tout doit loger dans le cache...
- Pipeline $($*Flush*$^{n-1}$, *Fill*$^{n+1}) \parallel$ *Compute*^{n} sur 2 zone de cache
- Boucle `independent` $\Longrightarrow$ liberté du pipeline
- Pipeline suivant :
  - Autres dimensions HPF si possible
  - Suivant c (bon facteur de forme pour le cache) si possible :

    $$c = \Gamma c' + c'',\ 0 \leq c'' < \Gamma$$

    Boucle $c \rightsquigarrow$ 2 boucles sur c' (pipeline) et c''
  - Suivant ℓ si pas de cyclique. Idem méthode c
- Choix du grain Γ du pipeline selon majoration du volume de variables nécessaire dans le cache

Exemple de code généré

```
      real A_hpf(0:3,0:1)
c     A(0:U:3) = ...
      do p = 0, 3
         do u3 = 0, min(4, divide(3*u, 16))
            do u4 = max(0, divide(8 + 16*u3 + 4*p, 9)),
     &           min(8, divide(u, 3), divide(3 + 16*u3 + 4*p, 9))
               c = u3
               i = 3*u4
               l = 9*u4 - 16*u3 - 4*p
               w1 = p
               w2 = l
               w3 = divide(c, 3)
               A_hpf(w2,w3) = ...
            enddo
         enddo
      enddo
```

Conclusion

- HPF pour décrire des distributions sur mémoire banquée
- Pas de langage supplémentaire
- Algèbre linéaire comme fondement du système de compilation
- Domaines polyhédriques quelconques
- Codéveloppement matériel-logiciel
- Différentes optimisations
- À valider sur des applications réelles
- `http://www.cri.ensmp.fr/pips`