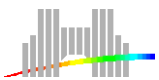
Ordonnancement et placement

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

6 novembre 2008



ORDONNANCEMENT

Ordonnancement

- Soit E un ensemble de tâches, opérations, instructions ... à exécuter à l'aide d'un certain nombre de machines, opérateurs, processeurs ...
- Ordonnancer = fixer pour chaque opération la date de départ et l'opérateur utilisé.
- Ordonnancement concret : le problème est donné sous forme de tables (des tâches, des durées, des dépendances) et il s'agit de tabuler les dates de lancement
- Ordonnancement symbolique : le problème est spécifié par des formules (l'ensemble des opérations, les dépendances) et on demande de trouver une formule donnant les dates de lancement.

Ordonnancement d'un nid de boucles

- Comme le nombre d'opérations peut être très grand (ou inconnu, ou infini), on doit utiliser l'ordonnancement symbolique
- Opération : $\langle S, i \rangle$, $i \in D_S$.
- Ordonnancement affine :

$$\theta(S, i) = h_S.i + k_S,$$

où h_S est un vecteur de même dimension que i, et où k_s est un scalaire, en général entiers.

- Première contrainte : $\theta(S, i) \geq 0$.
- NB : on peut utiliser des ordonnancements plus compliqués, par exemple affines par morceaux, quasi-affines (avec divisions entières)

Contrainte de causalité

L'ordre associé à l'ordonnancement doit respecter les dépendances :

$$\langle S, i \rangle \ \delta \ \langle T, j \rangle \Rightarrow \theta(S, i) + d(S) \leq \theta(T, j).$$

- Il s'agit ici de la relation de dépendance détaillée :

$$\begin{array}{rcl} i \in D_S, & & j \in D_T \\ f_S(i) & = & g_T(j), \\ i & \ll_p & j \end{array}$$

  - f_S, g_T : fonctions d'indice ;
  - $\ll_p$: ordre lexicographique à la profondeur p.

Un exemple I/III

Boucle de Lamport :

```
for(i=1; i<=n; i++)
    for(j=1; j<=n; j++)
        a[i][j] = 0.5*(a[i-1][j] + a[i][j-1]);
```

- Ordonnancement : $\theta(i,j) = \alpha i + \beta j$. Il n'y a pas besoin de terme constant.
- Durée de l'opération : 1.
- Dépendance : $(i,j) \rightarrow (i+1,j)$ et $(i,j) \rightarrow (i,j+1)$.
- Noter qu'aucune des boucles n'est parallèle.

Un exemple II/III

Résolution

$$\begin{aligned} \alpha i + \beta j + 1 &\leq \alpha(i+1) + \beta j, \\ \alpha i + \beta j + 1 &\leq \alpha i + \beta(j+1) \end{aligned}$$

- Soit $\alpha \geq 1, \beta \geq 1$.
- On a intérêt à prendre les plus petites valeurs possibles, donc : $\theta(i,j) = i + j$.

Un exemple III/III

Génération du code parallèle

- Rappel : la valeur de $i + j$ est la date à laquelle l'opération (i, j) doit être exécutée.
- Le temps doit avancer de sa valeur minimale, 2, à sa valeur maximale, $2n$.
- A chaque instant, on exécute en parallèle toutes les opérations telles que $i + j = t$.

```
for(t=2; t<=2*n; t++)
    //for(j=max(1, t-n); j <= min(n, t-1); j++)
        a[t-j][j] = 0.5 * (a[t-j-1][j] + a[t-j][j-1]);
```

- Vérifier : pas de dépendance dans la boucle parallèle.

Le programme objet

- Calculer les bornes du temps :

$$\begin{aligned} T_{\min} &= \min\{\theta(u) \mid u \in E\}, \\ T_{\max} &= \max\{\theta(u) \mid u \in E\} \end{aligned}$$

- Construire les *fronts* :

$$\mathtt{F}(t) = \{u \in E \mid \theta(u) = t\}.$$

```
for(t = Tmin; t <= Tmax; t++)
  forall F(t);
```

- On verra plus loin comment expliciter le `forall`
- Nécessite une architecture synchrone ou des instructions de synchronisation (`barrier`)
- Presque vectoriel

Dépendances uniformes

- L'exemple de Lamport est le cas d'un nid de boucle parfait à dépendances uniformes : $\langle S, i\rangle \longrightarrow \langle S, i + d_k\rangle,\ k = 1, n$.
- Conditions de causalité : $h_S.d_k \geq 1,\ k = 1, n$.
- Il y a toujours une solution, parce que les vecteurs d_k sont lexicopositifs. On peut trouver la meilleure par programmation linéaire.

Theorem

Un nid de boucle parfait de profondeur ≥ 2 à dépendances uniformes contient toujours du parallélisme.

Démonstration.

Ranger les vecteurs de dépendance par ordre lexicographique croissant, et construire l'ordonnancement par étape. □

Cas général

- La condition de causalité peut représenter jusqu'à $\mathrm{Card}\ D_S \times \mathrm{Card}\ D_T$ contraintes linéaires sur les coefficients des ordonnancements.
- Ce nombre peut être très grand, ou inconnu, ou même infini.
- Il faut trouver un moyen pour *résumer* ces contraintes en un ensemble fini.
- C'est possible parce que les contraintes sont linéaires.

Méthode des sommets

Patrice Quinton, circa. 1987

- La relation de dépendance détaillée définit un polyèdre dans le produit cartésien $D_S \times D_T$.

$$\begin{array}{rcl} i \in D_S, & & j \in D_T \\ f_S(i) & = & g_T(j), \\ i & \ll_p & j \end{array}$$

- Théorème de Minkowski : tout polyèdre est combinaison convexe d'un nombre fini de points, ses sommets.
- Pour qu'une fonction affine soit non négative dans un polyèdre, il faut et il suffit qu'elle soit non négative en ses sommets.
- Trouver les sommets du polyèdre des dépendances (algorithme de Chernikova).
- Ecrire la condition de causalité en ces points.
- Résoudre par programmation linéaire.

Méthode de Farkas I/II

Paul Feautrier, 1992

- Lemme de Farkas : pour qu'une fonction affine soit non négative dans un polyèdre, il faut et il suffit qu'elle soit combinaison affine positive des contraintes définissant le polyèdre :

$$Ax + b \geq 0 \Rightarrow cx + d \geq 0$$

est équivalent à :

$$cx + d = \lambda_0 + \lambda.(Ax + b) \quad \lambda_0, \lambda \geq 0.$$

- antécédent : les contraintes définissant le polyèdre des dépendances.
- conséquent : le délai $h_T.j + k_T - h_S.i - k_S - d_S$.

Méthode de Farkas II/II

- Ecrire l'identité de Farkas.
- Identifier les coefficients de i et j et les termes constants.
- On obtient un système d'équations linéaires dont les inconnues sont les coefficents des ordonnancements h_S, k_S et les multiplicateurs de Farkas.
- Comme les inconnues sont positives, on doit résoudre par une méthode de programmation linéaire.
- On peut ajouter une fonction objectif (par exemple, la latence totale).

Latence

En général, le programme dépend de paramètres positifs et grands (exemple de la méthode de Gauss : la taille de la matrice à inverser).

- La latence est la valeur maximum de l'ordonnancement dans le domaine d'itération, donc une fonction affine des paramétres :

$$L = H.n + K, \quad \forall i \in D_S : h_S.i + k_S \leq L.$$

- Cette contrainte peut être également traitée par l'algorithme de Farkas
- On ajoute le résultat aux autres contraintes et on résoud par le simplex, avec pour objectif de minimiser les coefficients de H.

Exemple

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
M:      c[i] += a[i][j]*b[j];
```

- Ordonnancement : $\theta(M, i, j) = \alpha i + \beta j.$
- Dépendance :

$$i \geq 0, j \geq 0,\ i' \geq 0, j' \geq 0$$

$$i = i',\ j' \geq j + 1.$$

$$\alpha i' + \beta j' - \alpha i - \beta j - 1 = \lambda_0 + \lambda_1 i + \lambda_2 j + \lambda_3 i' + \lambda_4 j' + \lambda_5 (j' - j - 1) + \mu (i' - i).$$

$$\alpha = \lambda_3 - \mu, \quad -\alpha = \lambda_1 + \mu,$$

$$\beta = \lambda_4 + \lambda_5, \quad -\beta = \lambda_2 - \lambda_5,$$

$$-1 = \lambda_0 - \lambda_5.$$

$$\lambda_0 = \lambda_1 = \lambda_2 = \lambda_3 = \lambda_4 = 0, \quad \lambda_5 = \beta = 1 \quad \mu = \alpha = 0.$$

Ordonnancements à délai borné

De la même façon, on peut rechercher des ordonnancements qui minimisent le délai associé à une dépendance :

$$\forall u, v : u \ \delta \ v \Rightarrow H_\delta.n + K_\delta \geq \theta(v) - \theta(u)$$

et application du lemme de Farkas.
De tels ordonnancements améliorent la localité (Bondughula et. al.)

Ordonnancement fractionnaires I/II

Il arrive que la résolution de la condition de causalité donne un ordonnancement à valeurs fractionnaires :

```
for(i=7; i<n; i++)
  a[i] = f(a[i-7]);
```

$$\theta(i) = \alpha i$$

$$\theta(i+7) \geq \theta(i) + 1$$

$$7\alpha \geq 1 : \alpha = 1/7 : \theta(i) = i/7.$$

Signification ?

Ordonnancements fractionnaires II/II

Theorem (Patrice Quinton)

Si θ est un ordonnancement causal fractionnaire, alors $\lfloor\theta\rfloor$ est un ordonnancement causal entier.

Démonstration.

Soit $u\ \delta\ v \Rightarrow \theta(u) + 1 \leq \theta(v)$. Comme la fonction $\lfloor x \rfloor$ est monotone croissante :

$$\lfloor\theta(u) + 1\rfloor = \lfloor theta(u)\rfloor + 1 \leq \lfloor\theta(v)\rfloor$$

□

Retour à l'exemple : $\theta(i) = i \div 7$. On voit que les itérations 0 à 6 s'exécutent toutes, en parallèle, à l'instant 0. Le programme correspondant (simplifié) :

```
for(i=7; i<n; i+=7)
  //for(j=0; j<7; j++)
     a[i+j] = f(a[i+j-7]);
```

a un degré de parallélisme de 7.

Echec et Mat ...

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
M:      s += a[i][j];
```

- Ordonnancement : $\theta(M, i, j) = \alpha i + \beta j$.
- Dépendance :

$$i \geq 0, j \geq 0,\ i' \geq 0, j' \geq 0$$

$$i' \geq i + 1,$$

$$i = i',\ j' \geq j + 1.$$

$$\alpha i' + \beta j' - \alpha i - \beta j - 1 = \lambda_0 + \lambda_1 i + \lambda_2 j + \lambda_3 i' + \lambda_4 j' + \lambda_5 (i' - i - 1).$$

$$\beta j' - \beta j - 1 = \mu_0 + \mu_1 i + \mu_2 j + \mu_3 j' + \mu_4 (j' - j - 1)$$

La première identité entraine $\beta = 0$ alors que la seconde entraine $\beta \geq 1$. Il y a contradiction.
Il n'y a pas d'ordonnancement affine légal.

Explication

- Un programme dont les boucles sont de taille n et qui a un ordonnancement affine, s'exécute en temps $O(n)$ sur un nombre illimité de processeurs.
- Or le programme de l'exemple est intrinséquement séquentiel. Il doit prendre $O(n^2)$ cycles quelque soit le nombre de processeurs.
- Son ordonnancement doit être quadratique ou *bidimensionnel*.

Ordonnancement multidimensionnel

- La valeur de l'ordonnancement (le temps) est maintenant un vecteur :

$$\theta(S, i) = (\theta_1(S, i), \cdots, \theta_d(S, i))$$

.

- L'ordre de succession des instants est l'ordre lexicographique (comme pour les chiffres d'une montre digitale).
- Il est facile de voir que la durée d'un ordonnancement de dimension d est $O(n^d)$.

Condition de causalité, I/II

- Soit u et v deux opérations en dépendances.
- On doit avoir $\theta(u) \ll \theta(v)$, soit :

$$\theta_1(u) + 1 \leq \theta_1(v) \vee (\theta_1(u) = \theta_1(v) \ \& \ \cdots)$$

- On introduit une variable $\epsilon \in \{0, 1\}$ et on écrit la contrainte :

$$\theta_1(u) + \epsilon \leq \theta_1(v),$$

on utilise l'algorithme de Farkas et on résoud en maximisant $\sum \epsilon$.

- Il n'est pas nécessaire d'utiliser la PLNE.

Condition de causalité, II/II

- Les dépendances pour lesquelles $\epsilon = 1$ sont satisfaites.
- Pour déterminer θ_2, on recommence en ne prenant en compte que les dépendances pour lesquelles $\epsilon = 0$.
- On s'arrète quand toutes les dépendances sont satisfaites.
- On démontre que l'algorithme converge toujours, au pire vers l'ordonnancement séquentiel.
- On démontre que l'algorithme construit l'ordonnancement de plus petite dimension (donc le plus efficace) compatible avec les dépendances (Frédéric Vivien).

Retour sur l'exemple

$$\alpha i' + \beta j' - \alpha i - \beta j - \epsilon_1 = \lambda_0 + \lambda_1 i + \lambda_2 j + \lambda_3 i' + \lambda_4 j' + \lambda_5 (i' - i - 1).$$

$$\beta j' - \beta j - \epsilon_2 = \mu_0 + \mu_1 i + \mu_2 j + \mu_3 j' + \mu_4 (j' - j - 1)$$

$$\alpha = \lambda_3 + \lambda_5,\ \beta = \lambda_4,\ -\alpha = \lambda_1 - \lambda_5,\ -\beta = \lambda_2,\ -\epsilon_1 = \lambda_0 - \lambda_5.$$

$$\beta = 0,\ \ \alpha \geq \epsilon_1.$$

$$\beta = \mu_3 + \mu_4,\ -\beta = \mu_2 - \mu_4,\ 0 = \mu_1,\ -\epsilon_2 = \mu_0 - \mu_4.$$

$$\beta \geq \epsilon_2.$$

Il en résulte que $\epsilon_2 = 0, \epsilon_1 = 1$. Pour la deuxième composante du temps, on n'utilise que la deuxième contrainte, et on peut prendre $\epsilon_2 = 1$.

$$\theta(i, j) = (i, j).$$

C'est l'ordonnancement séquentiel.

Optimisation

On peut ordonnancer composante fortement connexe par composante fortement connexe, dans l'ordre du graphe réduit.

- L'ordonnancement des sommets initiaux ne dépend pas des autres sommets
- Quand une cfc a été ordonnancée, on peut substituer son ordonnancement dans les cfc suivantes
- Comme dans l'algorithme de Kennedy et Allen, quand on descend d'un niveau, les cfc peuvent être déconnectées.

Lemme de comptage

Theorem

Si θ est un ordonnancement multidimensionnel causal :

$$u \ \delta \ v \Rightarrow \theta(u) \ll \theta(v)$$

alors $t(u) = \mathrm{Card}\ \{\theta(x) \mid \theta(x) \ll \theta(u)\}$ est un ordonnancement monodimensionnel causal.

Démonstration.

$$\begin{aligned} u \ \delta \ v &\Rightarrow \theta(u) \ll \theta(v) \\ &\Rightarrow \{\theta(x) \mid \theta(x) \ll \theta(u)\} \subset \{\theta(x) \mid \theta(x) \ll \theta(u)\} \\ &\Rightarrow t(u) < t(v) \end{aligned}$$

L'inclusion est stricte car $\theta(v)$ fait partie du deuxième ensemble mais pas du premier.

Conclusion

- La méthode la plus puissante connue à ce jour pour trouver du parallélisme.
- Peut s'appliquer après passage en assignation unique (simplification des conditions).
- Cependant, contre-exemple.

```
for(i=0, i<=2*n; i++)
    a[i] = a[2*n-i];
```

- Le problème de l'ordonnancement sous contraintes de ressource est ouvert.

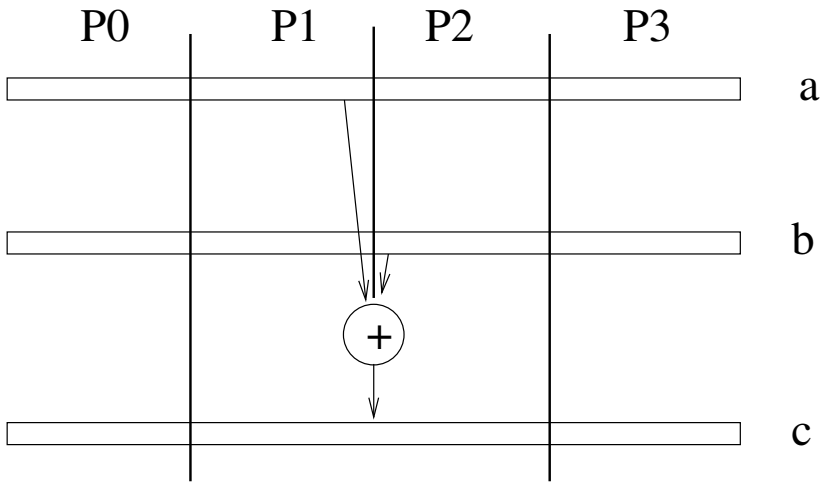
PLACEMENT

Critique

- Le programme obtenu est *à grain fin* : les boucles parallèles sont à l'intérieur des boucles séquentielles
- Le programme obtenu manque de localité : chaque opération d'un front écrit dans une cellule de mémoire différente
- Le programme obtenu ne convient pas pour un ordinateur à mémoire distribuée : la question de savoir comment les données sont réparties entre les mémoires et quel processeur exécute les calculs n'est pas résolue.

Un exemple trivial

```
for(i=0; i<n; i++)
S:   a[i] = b[i] + c[i];
```



- On dispose de 4 processeurs
- Si l'on met chaque tableau sur un processeur différent et si on fait les calculs sur le quatrième, il y aura beaucoup de communications et pas de parallélisme
- Si on coupe chaque tableau en quatre et si on distribue sur quatre processeurs, il n'y aura pas de communications et beaucoup de parallélisme

Formalisation

Hypothèse :

- Une grappe composée de processeurs identiques interconnectés par un réseau
- Chaque processeur a sa mémoire
- Les processeurs sont numérotés de 0 à $P-1$
- Extension : les processeurs forment une grille à d dimensions et sont numérotés par un vecteur.

Fonction de placement :

- Une fonction qui donne le numéro du processeur qui exécute chaque opération : l'opération $\langle S, x \rangle, x \in D_S$ est exécutée par le processeur $\varpi(S, x) \in [0, P-1]$
- Une fonction qui donne le numéro de la mémoire qui héberge chaque cellule de tableau : la cellule $A[i]$ est hébergée par la mémoire $\varpi(A, i) \in [0, P-1]$.

Equations de coupure

Soit $\langle S, x \rangle$ une opération qui accède (en lecture ou en écriture) au tableau A avec l'indice $f(x)$. Pour qu'il n'y ait pas de communication, il faut que :

$$\forall x \in D_S : \varpi(S, x) = \varpi(A, f(x)).$$

Comment résoudre ?

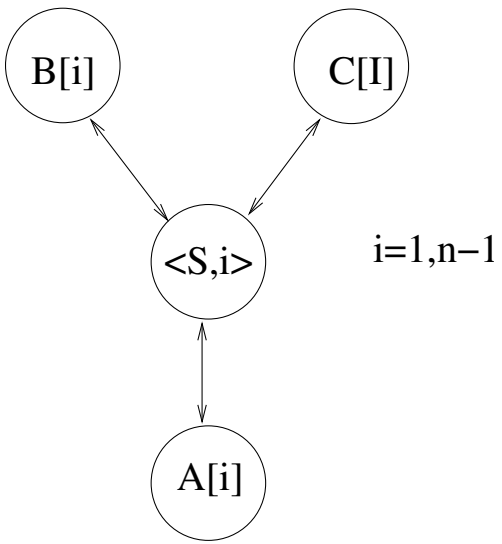
- On observe que si un placement ϖ satisfait les équations de coupure, toute fonction $\lambda u.\phi(\varpi(u))$ les satisfait aussi.
- Utilité 1) Si ϕ est une permutation, cela correspond à un renommage des processeurs
- Utilité 2) Si ϕ n'est pas biunivoque, cela permet de réduire le nombre de processeurs, et en particulier de garantir $\phi(\varpi(u)) \in [O, P-1]$
- Application pratique : $\phi(x) = x \bmod P$.

Solution formelle

On considère le graphe de communication (biparti, non orienté) ayant pour sommet les opérations et les cellules de mémoire.

- Si l'opération $\langle S, x \rangle$ accède à la cellule $A[f(x)]$, alors il y a un arc entre $\langle S, x \rangle$ et $A[f(x)]$.
- Si les équations de coupures sont satisfaites, alors :
  - Deux sommets voisins ont le même placement,
  - Deux sommets qui peuvent être joints par un chemin ont le même placement
  - La fonction de placement est constante sur une composante connexe du graphe de communication.
- En particulier, si le graphe de communication est connexe, la fonction de placement est constante : il y a *collapsus* sur un seul processeur !

Retour sur l'exemple



- Il y a autant de composantes connexes que de valeur de i
- On peut donc prendre

$$\varpi(A,x) = \varpi(B,x) = \varpi(C,x) = \varpi(S,x) = x$$

- Si on prend $\phi(x) = \lfloor x/D \rfloor$ avec $D = \lceil n/P \rceil$ on retrouve la solution ci-dessus (partitionnement par bloc)
- Mais on peut également prendre $\phi(x) = x \bmod P$ (partitionnement cyclique).

La matrice de communication, I/II

La méthode ci-dessus n'est effective que dans les cas simples.

- On choisit une fonction de placement affine :

$$\varpi(U, x) = h_U.x + k_U,$$

  et le problème est de déterminer les valeurs des vecteurs h_U et des constantes k_U.
- Hypothèse : les domaines d'itérations sont de dimension pleine
- Il suffit alors d'écrire les équations de coupures en $d + 1$ points pour qu'elles soient vérifiées partout

La matrice de communication, II/II

Hypothèses : Une opération $\langle S, i \rangle$ qui accède à un tableau `A` au moyen de la fonction d'indice $f_{SA}(i) = F_{SA}i + G_{SA}$.

- Le placement de S : $\varpi(S, i) = h_S.i + k_S$
- Le placement de A : $\varpi(A, x) = h_A.x + k_A$
- L'équation de coupure :

$$h_S.i + k_S = h_A.(F_{SA}i + G_{SA}) + k_A$$

- On fait $i = 0$: $k_S = h_A G_{SA} + k_A$
- On fait successivement $i = e_k$ (les vecteurs unitaires) : $h_S = h_A F_{SA}$.

On a obtenu un système d'équations linéaires et homogènes pour les inconnues h_S, k_S, h_A, k_A. La matrice de ce système est la matrice de communication du programme.

Résolution

Observation : Si la matrice de communication est de rang plein, la seule solution est la solution triviale. Il n'est pas en général possible de couper toutes les communications.

On introduit le vecteur h obtenu en concaténant les h_A, k_A, h_S, k_S. Pour chaque référence SA, la méthode précédente construit un système $C_{SA}h = 0$.

$C := \emptyset$;
foreach SA **do**
    former la matrice $C' := C \cup C_{SA}$;
    if $\ker C' \neq \{0\}$ **then**
        $C := C'$
    end
end

Résolution, remarques

- Il ne suffit pas de vérifier que le système C' a des solutions non triviales. Il faut vérifier que chaque h_S est non nul pour avoir du parallélisme
- Il existe une méthode pour calculer le noyau de C' de façon incrémentale
- L'ordre dans lesquelles les références sont traitées est important. Il faut commencer par les références qui engendrent le plus de traffic
- Il est usuel de toujours couper en priorité les références en écriture (*owner computes rule*).

Le programme objet, I/II

On suppose un ordinateur à mémoire distribuée. On pose :

- P le nombre de processeurs physiques
- Par exemple, $\Pi(S, i) = \varpi(S, i) \bmod P$
- $p_{\min}(t) = \min\{\Pi(u) \mid \theta(u) = t\}$, $p_{\max}(t) = \max\{\Pi(u) \mid \theta(u) = t\}$
- $\mathcal{F}(t, p) = \{u \in E \mid \theta(u) = t, \Pi(u) = p\}$

```
for(t=Tmin; t<=Tmax; t++){
  //for(p=pmin(t); p <= pmax(t); p++){
      acquérir les données distantes;
      F(t,p);
      envoyer les résultats distants;
  }
  barrier;
}
```

Les accés à distance ne concernent que les références dont les équations de coupure n'ont pu être satisfaites.

Le programme objet, II/II

On peut également n'utiliser que le placement :

- $p_{\min} = \min\{\Pi(u) \mid u \in E\}, p_{\max} = \max\{\Pi(u) \mid u \in E\}$
- Les opérations $\{u \in E \mid \Pi(u) = p, p_{\min} \leq p \leq p_{\max}\}$ constituent un *thread* que l'on exécute sans l'ordre séquentiel
- A chaque équation de coupure non satisfaite correspond un échange de message qui assure en même temps une synchronisation
- Le programme peut être optimisé de diverses façons (regroupement de messages et de synchronisations).

Conclusion

- Le problème du calcul de l'ordonnancement libre est maintenant bien connu
- Par contre, le problème de l'ordonnancement sous contraintes de ressources reste ouvert. Le problème est en général de *ralentir* un ordonnancement ayant trop de parallélisme
- Approches possibles :
  - Ajouter des écritures dans un tableau fictif. Le nombre d'opérations simultanées ne peut dépasser la taille du tableau
  - Ajouter des dépendances fictives (qui doivent être non uniformes)
  - Déterminer une fonction d'allocation (une fonction qui respecte les dépendances mais qui ne les satisfait pas nécessairement) et la tuiler