Optimisation du parallélisme à grain fin

Christine Eisenbeis, INRIA,
Rocquencourt, France

Plan

- Architectures
- Modélisation
- Ordonnancement
- Un algo optimal d'allocation de ressources, application au processeur i860 d'Intel
- Une expérience avec les compilateurs FORTRAN sur le SuperSPARC et le DEC alpha 21064
- Conclusion

Architectures

Parallélisme à bas niveau, grain fin
ILP Instruction Level Parallelism

- unités fonctionnelles multiples (ex : 2 ALU)
- unités fonctionnelles pipelinées (émission de l'instruction même si la précédente n'est pas terminée)
- plusieurs instructions en parallèle (émission de plusieurs instructions en même temps)

Exemples

- **LIV/VLIW** FPS-,ST-100, ...
  WARP, PCS ...
  MULTIFLOW, CYDROME ...
  Synchrones, requièrent des compilateurs spécialisés

- **RISC** MIPS, SPARC ...
  Peu de parallélisme

- **Superscalaires** IBM RS-6000, i960 ...
  i860 ...
  PowerPC, DEC-alpha 20164,21164, MIPS R8000-R10000, UltraSparc, P6 Intel *Plusieurs instructions par cycle*

- **Modèles** CRAY-2
  Plusieurs instructions vectorielles par cycle

Contrôle du parallélisme

Contrôle statique Famille “VLIW”

Résolution des conflits de données et ressources à la *compilation*

Hardware simplifié

Contrôle dynamique Famille “Superscalaire”

Résolution des conflits de données et ressources à l'*exécution*

- Pipeline interlock
- Exécution désordonnée
- Exécution spéculative
- Renommage de registres

Évolution

CISC Jeu d'instructions optimisées

Optimisation = recherche du plus petit nombre d'instructions

RISC Simplification du jeu d'instructions

Optimisation = Ordonnancement (bonus)

Superscalaires Lancement en parallèle d'instructions, contrôle dynamique

Optimisation = Ordonnancement (important)

VLIW Lancement en parallèle d'instructions, contrôle statique

Ordonnancement nécessaire

Étape de modélisation

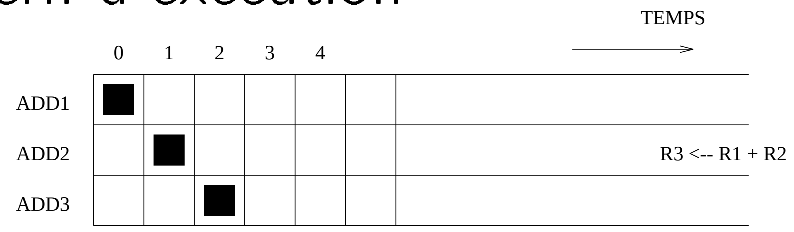
Mécanismes dynamiques

$$\Downarrow$$

Modèle statique

Génération de code

- Pattern d'exécution



- Résolution des conflits de ressource

Hardware	Software
.	.
.	.
.	.
.	$R3 \leftarrow R1 + R2$
$R3 \leftarrow R1 + R2$	NOP
STORE $R3$	NOP
.	STORE $R3$
.	.
.	.
.	.

- Ensemble de registres indifférenciés

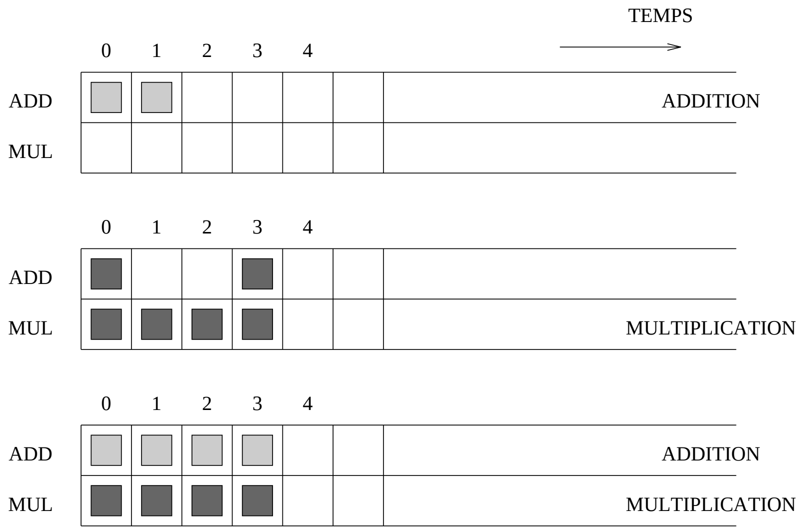
Modèle d'exécution

Synchrone

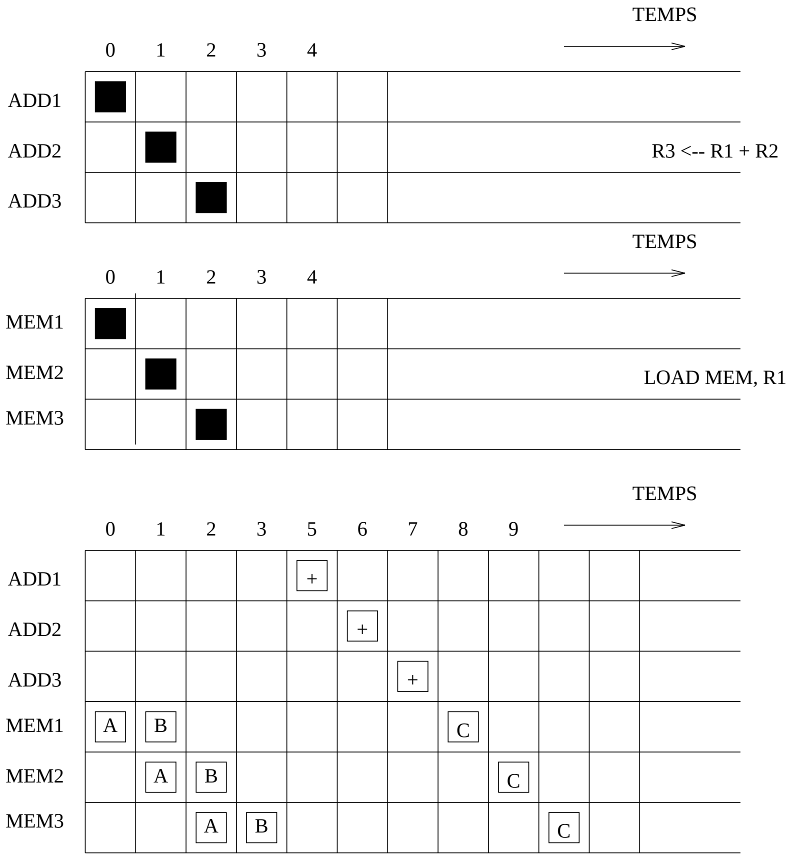
Formalisme des tables de réservation

Des bâtis (templates) modélisent l'utilisation des ressources par les instructions

Exemple: **MIPS-RS2000**



Tables de réservation



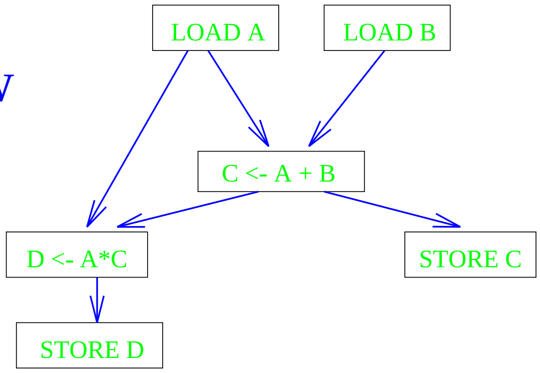
$$C = A + B$$

Optimisation de code (1)

- Optimisation locale

graphe data-flow

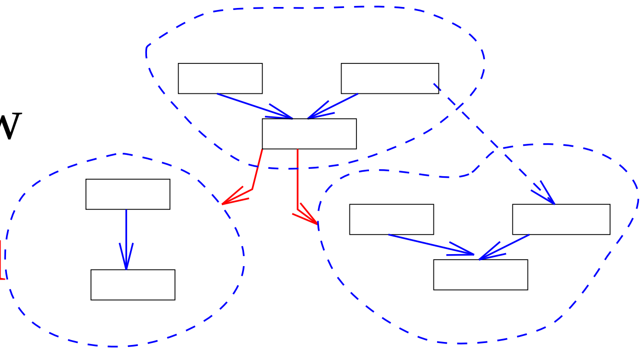
bloc de base



- Optimisation globale

data / control flow

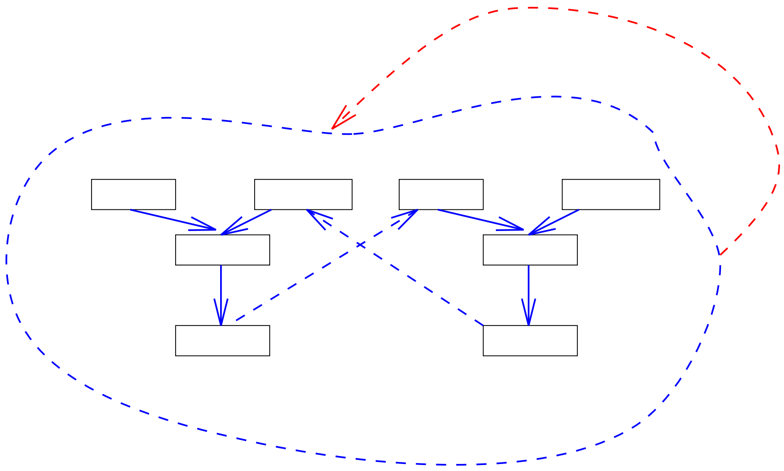
programme général



Optimisation de code (2)

- Optimisation des boucles

boucle



Ordonnancement Local - Ressources statiques

DAG	Architecture (unités fonctionnelles)
min (temps d'exécution)	

Problème NP-complet

- **en pratique:** bonne heuristique : ordonnancement par liste de priorité

- **en théorie:** quelques cas polynomiaux

Arbre binaire	1 unité mémoire 1 unité de calcul
DAG	1 unité pipelinée à 1 ou 2 étages

Ordonnancement Local - Registres

DAG	Architecture (registres)
min (temps d'exécution)	

Problème NP-complet

en pratique : pas d'algo satisfaisant

en théorie : le problème reste NP-complet **même pour**

Arbre binaire
Ensemble de chaînes

une formulation par programme linéaire en nombres entiers

Ordonnancement Global

Technique d'ordonnancement par traces

- Extraction d'une trace
- Ordonnancement local de la trace
- Correction du code

Explosion de la taille du code

Dans le cas des boucles: dépliage

Boucles

Problème plus facile en général car il s'agit d'optimiser un *débit* (problème "en moyenne")

Problème polynomial dans les cas suivants :

- graphe de dépendances de données sans cycle
- pas de contrainte de ressources

Boucles - Pipeline logiciel

Exemple:

```
Do i =  1,N
      r1 <-- A(i)
      r2 <-- B(i)
      r3 <-- r1+r2
      C(i) <-- r3
Enddo
```

Hypothèse:

- lecture et écriture mémoire avec adressage auto-incrementé
- support matériel pour branchement de fin de boucle

<u>Modèle de machine</u>

Unité Fonctionnelle	latence	occupation
Accès Mémoire	m	1
Additionneur	a	1

Boucles : Ordonnancement local

LOOP (i=i+1)

(i=1)	1	2	3	...	m	m+1	m+2	m+2	...	m+a+1	m+a+2
MEM 1	A(i)	B(i)									C(i)
MEM 2		A(i)	B(i)								
...			...	...	...	...	...	...			
...				...	...	...	...	...			
MEM m					A(i)	B(i)					
ADD 1							+(i)		...		
ADD 2								+(i)			
...									...		
ADD a										+(i)	

1 itération tous les $a + m + 2$ cycles

$$L = a + m + 2$$

$$U = 1$$

Boucles : Optimisation par dépliage fini

LOOP (i=i+4)

(i=1)	1	2	3	...	m	m+1	m+2	m+2	...	m+a+1	m+a+2	m+a+3	m+a+4	m+a+5	m+a+6	m+a+7	m+a+8
MEM 1	A(i)	B(i)	A(i+1)	B(i+1)	A(i+2)	B(i+2)	A(i+3)	B(i+3)			C(i)		C(i+1)		C(i+2)		C(i+3)
MEM 2		A(i)	B(i)	A(i+1)	B(i+1)	A(i+2)	B(i+2)	A(i+3)									
...			...	...	...	...	...	...									
...				...	...	...	...	...									
MEM m					A(i)	B(i)	A(i+1)	B(i+1)									
ADD 1							+(i)		...		+(i+2)		+(i+3)				
ADD 2								+(i)		+(i+1)		+(i+2)		+(i+3)			
...									...		+(i+1)		+(i+2)		+(i+3)		
ADD a										+(i)		+(i+1)		+(i+2)		+(i+3)	

4 itérations tous les $a + m + 8$ cycles

$$L = \frac{m+a+8}{4}$$

$$U = 4$$

Boucles : Optimisation par dépliage "infini"

(i=8) LOOP (i=i+6)

	1	2		21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
ADD 1	A(1)	B(1)		B(i)	A(i+1)	C(i-2)	B(i+1)	A(i+2)	B(i+2)	C(i-1)	A(i+3)	B(i+3)	C(i)	A(i+4)	B(i+4)	C(i+1)	A(i+5)	C(i+2)	B(i+5)	A(i+6)	C(i+3)
MEM1						+(i-1)			+(i)			+(i+1)		+(i+2)			+(i+3)			+(i+4)	

6 itérations tous les 18 cycles

$$L = 3$$

$$U = 6$$

Ordonnancement cyclique

LOOP (i=i+1)

(i=1)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
MEM 1	A(1)	B(1)		A(2)	B(2)		A(3)	B(3)		A(i)	B(i)	C(i-3)			C(i-2)			C(i-1)			C(i)
MEM 2		A(1)	B(1)		A(2)	B(2)		A(3)	B(3)												
...			:::	...																	
...				...	...																
MEM m					A(1)	B(1)		A(2)	B(2)		A(i-1)	B(i-1)									
ADD 1							+(1)			+(i-2)			+(i-1)			+(i)					
ADD 2																					
...																					
ADD a																					

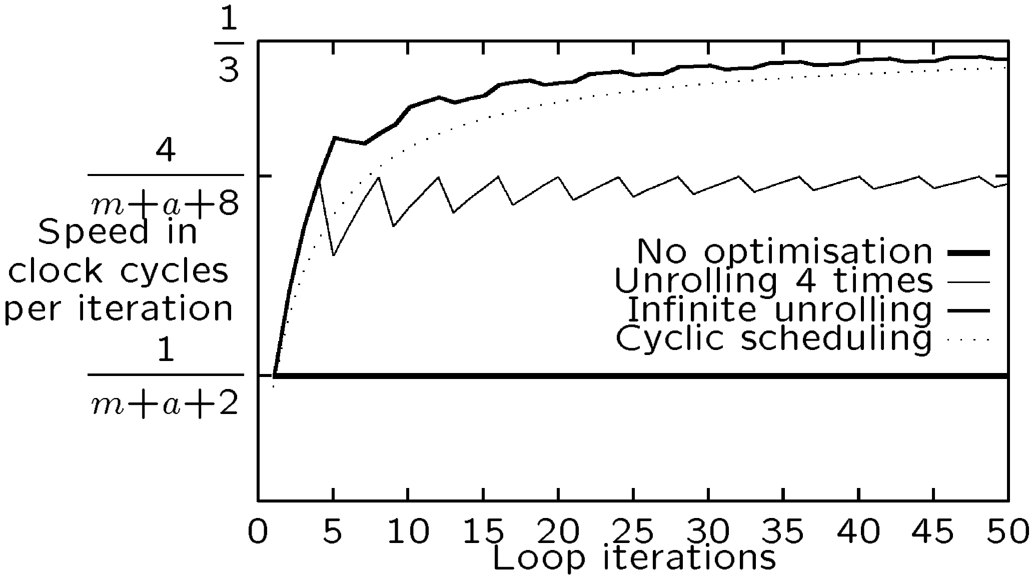
1 itération tous les 3 cycles

$$L = 3$$

$$U = 1$$

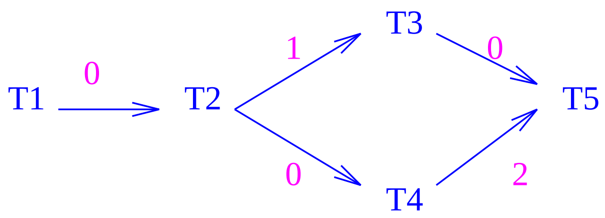
INRIA

Performances



Graphe acyclique - exemple

```
          DO i = 1,N
T1(i)        A(I) = B(I) + C(I)
T2(i)        D(I+1) = A(I) * 2
T3(i)        F(I) = D(I) + E(I)
T4(i)        G(I) = F(I) * H(I)
T5(i)        H(I+2) = D(I+1) + 5
```

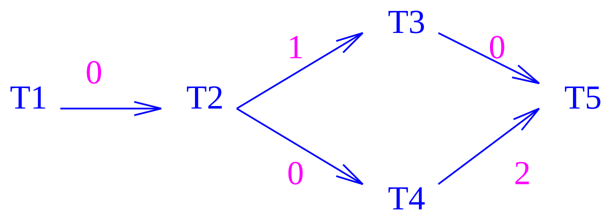


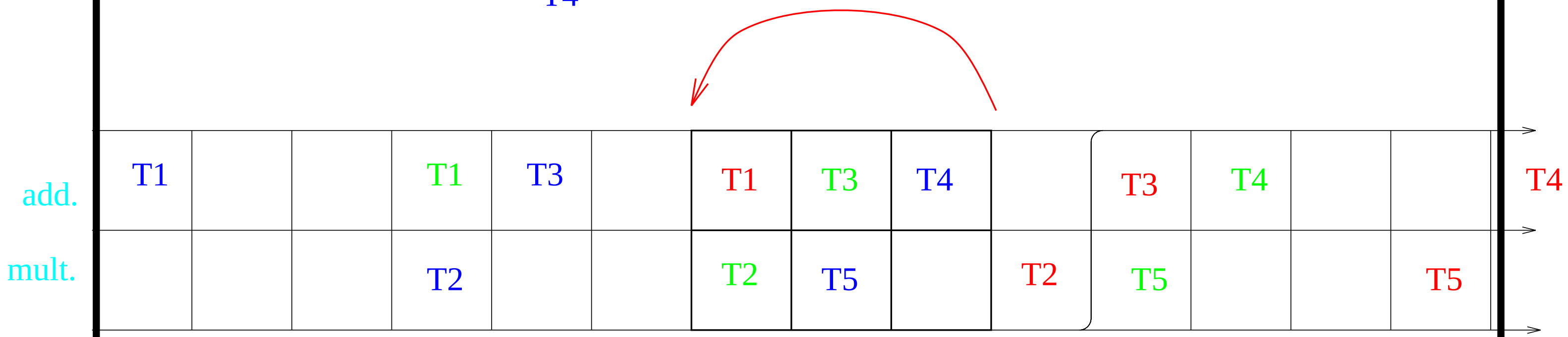
distance de dépendance

time →

add.	T1	T3	T4
mult.	T2	T5	

Assignation des itérations (problème de plus long chemin)





Code résultant:

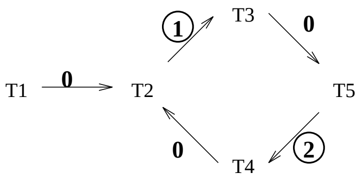
T1(1)
T1(2) T2(1)
T3(1)
T1(i+2) T2(i+1)
i=i+1 T3(i+1) T5(i)
T4(i)
T2(N)
T3(N) T5(N-1)
T4(N-1)
T5(N)

Modélisation de la boucle, paramètres (1)

- Contraintes sémantiques: dépendances de données
  Hauteurs h_{ij}

$T1(i) \longrightarrow T2(i)$
$T3(i) \longrightarrow T5(i)$
$T4(i) \longrightarrow T2(i)$

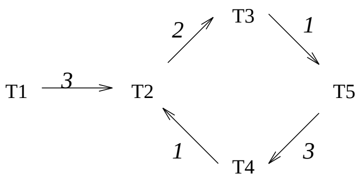
$T2(i) \longrightarrow T3(i+1)$
$T5(i) \longrightarrow T4(i+2)$



- Contraintes matériel
  Valuations v_{ij}

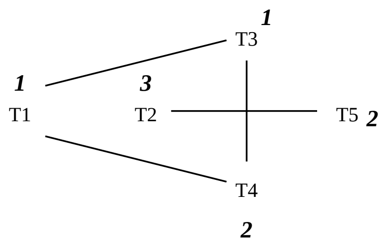
$T1(i) \xrightarrow{3} T2(i)$
$T3(i) \xrightarrow{1} T5(i)$
$T4(i) \xrightarrow{1} T2(i)$

$T2(i) \xrightarrow{2} T3(i+1)$
$T5(i) \xrightarrow{3} T4(i+2)$

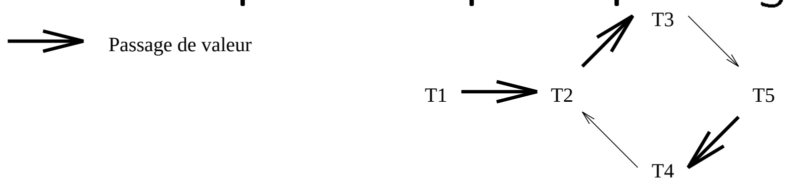


Modélisation de la boucle, paramètres (2)

- Contraintes matériel: nombre fini de processeurs
  Contraintes de disjonction, durées $d_i + m_p$ processeurs de type P_p



- Contraintes matériel: nombre fini de registres: R_{max}
  Flèches spéciales pour passage de valeurs



Ordonnancement des boucles: bornes

Débit asymptotique : $\boxed{\max(\frac{1}{L})}$

Latence moyenne : $\boxed{\min(L)}$

- (Contrainte due au programme)

$$(L_1)\ \boxed{L \geq \frac{\sum_C v_{ij}}{\sum_C h_{ij}}}$$

pour chaque circuit C du graphe

- (Contrainte due à la machine) m_1 processeurs de type P_1, m_2 de type P_2, ..., m_k de type P_k

$$(L_2)\ \boxed{L \geq \frac{\sum_T d_T}{m_i}}$$

pour chaque type de processeur P_i

Problèmes

Il peut s'avérer nécessaire de *déplier* la boucle pour les raisons suivantes:

1. présence d'un cycle de dépendance critique de valeur non entière

$$U_{dep}$$

2. contraintes de resources

$$U_{res}$$

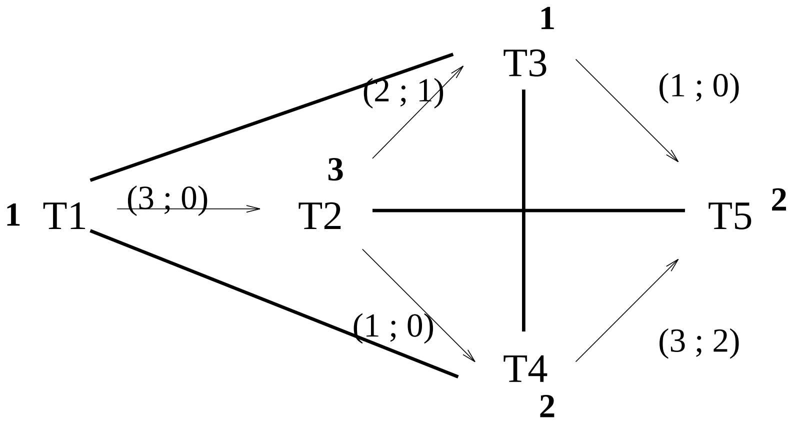
3. occupation des registres

$$U_{reg}$$

Le déroulage global est donnée par

$$U = \mathrm{lcm}\,(U_{dep}, U_{res}, U_{reg})$$

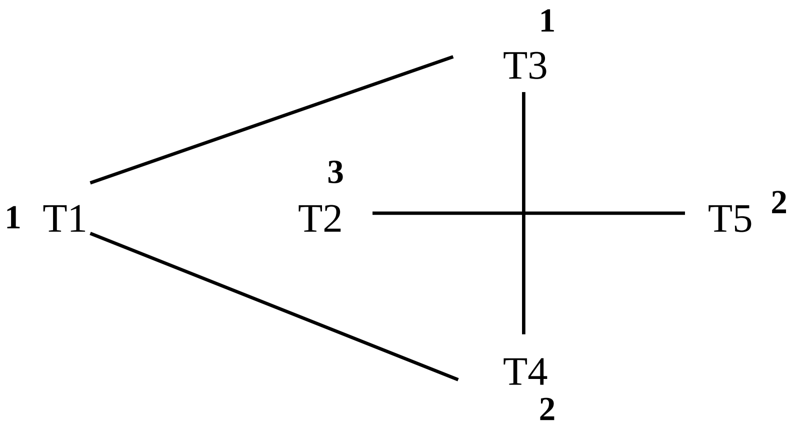
Graphe acyclique



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P1	1																
	2																
	m1																
P2	1																
	2																
	3																
	m2																

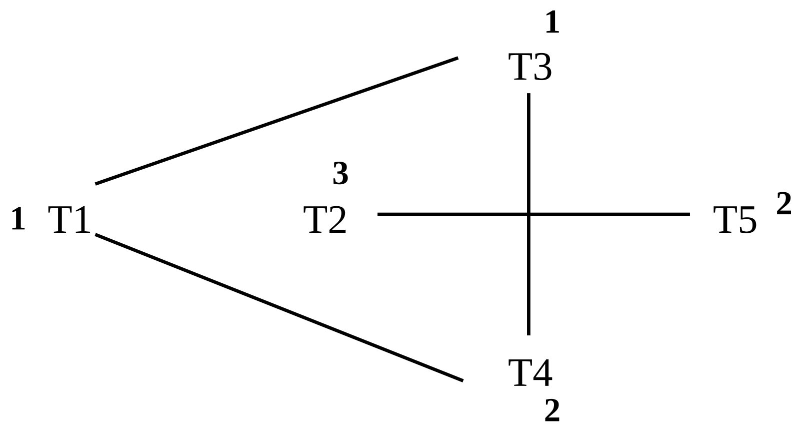
$$L_1 = 0,\ L_2 = max(4/3, 5/4) = 1.333..$$

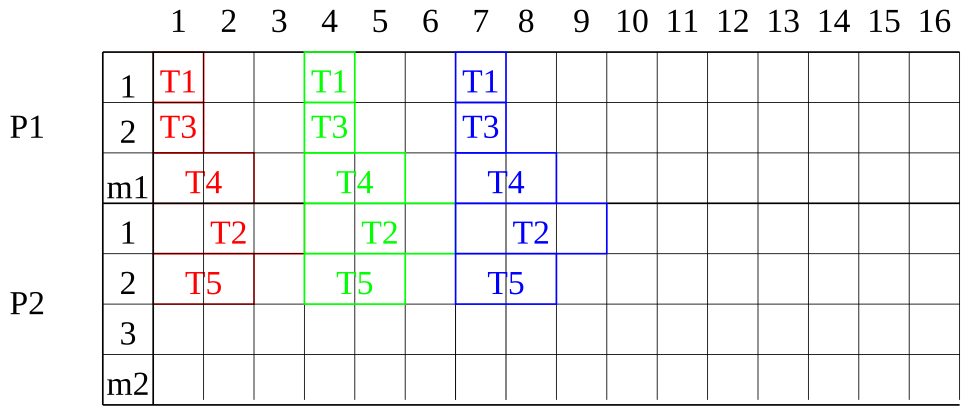
Méthode: Relaxer les contraintes de sémantique



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P1	1																
	2																
	m1																
P2	1																
	2																
	3																
	m2																

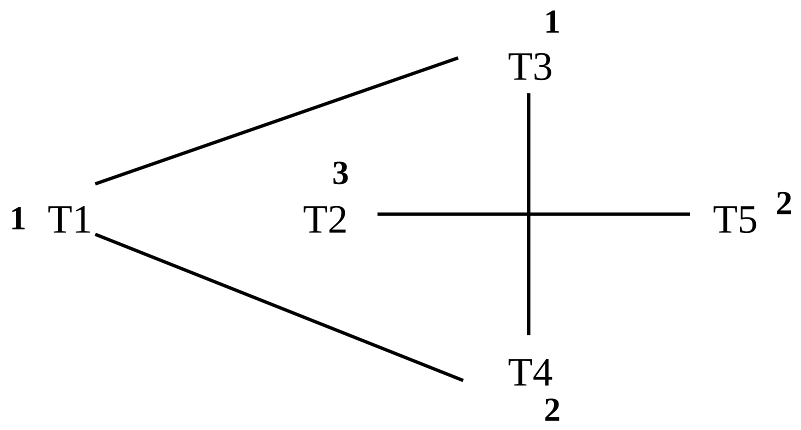
Mauvaise utilisation des ressources ?





L=3, U=1

Déroulage

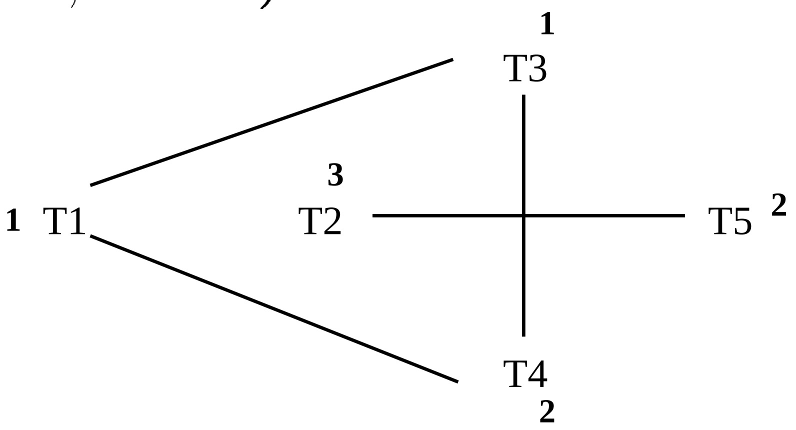


		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
P1	1	T1	T4														
	2	T3		T3													
	m1	T4		T1													
P2	1		T2														
	2	T5															
	3		T2														
	m2	T5															

L=3/2, U=2

Utilisation optimale

Borne supérieuresur le déroulage minimal nécessaire à la saturation des ressources :
$ppcm(m1, m2, \ldots Mk)$



		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	1	T1	T3	T4		T1	T3	T4		T1	T3	T4		T1	T3	T4	
P1	2	T1	T3	T4		T1	T3	T4		T1	T3	T4		T1	T3	T4	
	m1	T1	T3	T4		T1	T3	T4		T1	T3	T4		T1	T3	T4	
	1		T2		T5			T2		T5			T2		T5		
P2	2		T2		T5			T2		T5			T2		T5		
	3		T2		T5			T2		T5			T2		T5		
	m2		T2		T5			T2		T5			T2		T5		

L=4/3, U=12

Résultats :

1 type de processeur en m exemplaires •

n tâches de même durée d : U valide ssi $U \geq \frac{m}{n}$ et U multiple de $\frac{m}{pgcd(nd,m)}$

- $n1$ tâches de durée $d1$, $n2$ tâches de durée $d2$: U valide ssi UL est multiple de $pgcd(d1, d2)$ et

$$m\lceil\frac{UaL}{d2}\rceil \leq U(bn1 - an2) \leq m\lfloor\frac{UbL}{d1}\rfloor$$

- plus de 3 durées différentes ?

cas général Approximation par de bin packing ou utilisation de de la borne supérieure

Une classe d'algorithmes de pipeline logiciel

Deux étapes:

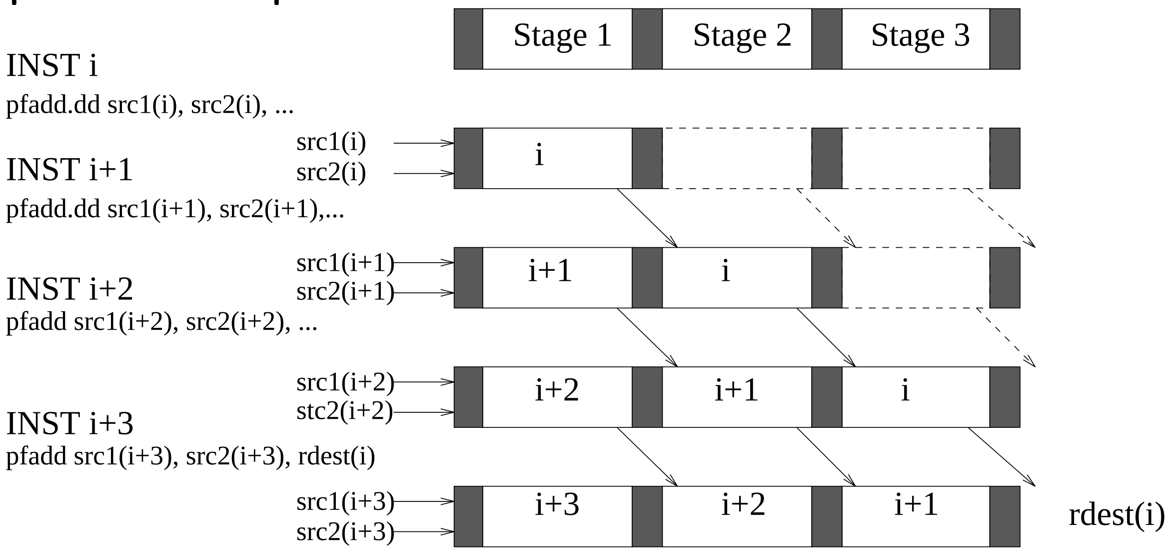
1. construction de la table de réservation avec l'hypothèse que toutes les instructions sont indépendantes
2. calcul du numéro d'itération de chaque instruction

Intérêts

- **Résultat théorique**
  complexité du problème d'ordonnancement cyclique avec contraintes de ressources
  - borne supérieure pour α_{res} dépendant uniquement de l'architecture
  - La complexité est polynomiale pour une architecture donnée
- **Résultat pratique**
  permet de traiter des contraintes de ressources complexes
  Exemple: pipeline explicit du processeur i860

Application au processeur Intel i860

Pipelines explicites:



Exemple (1)

Code source:

```
        DO 1 i = 1, N
                Y(i) = A * X(i) + B
1       CONTINUE
```

Code assembleur I860:

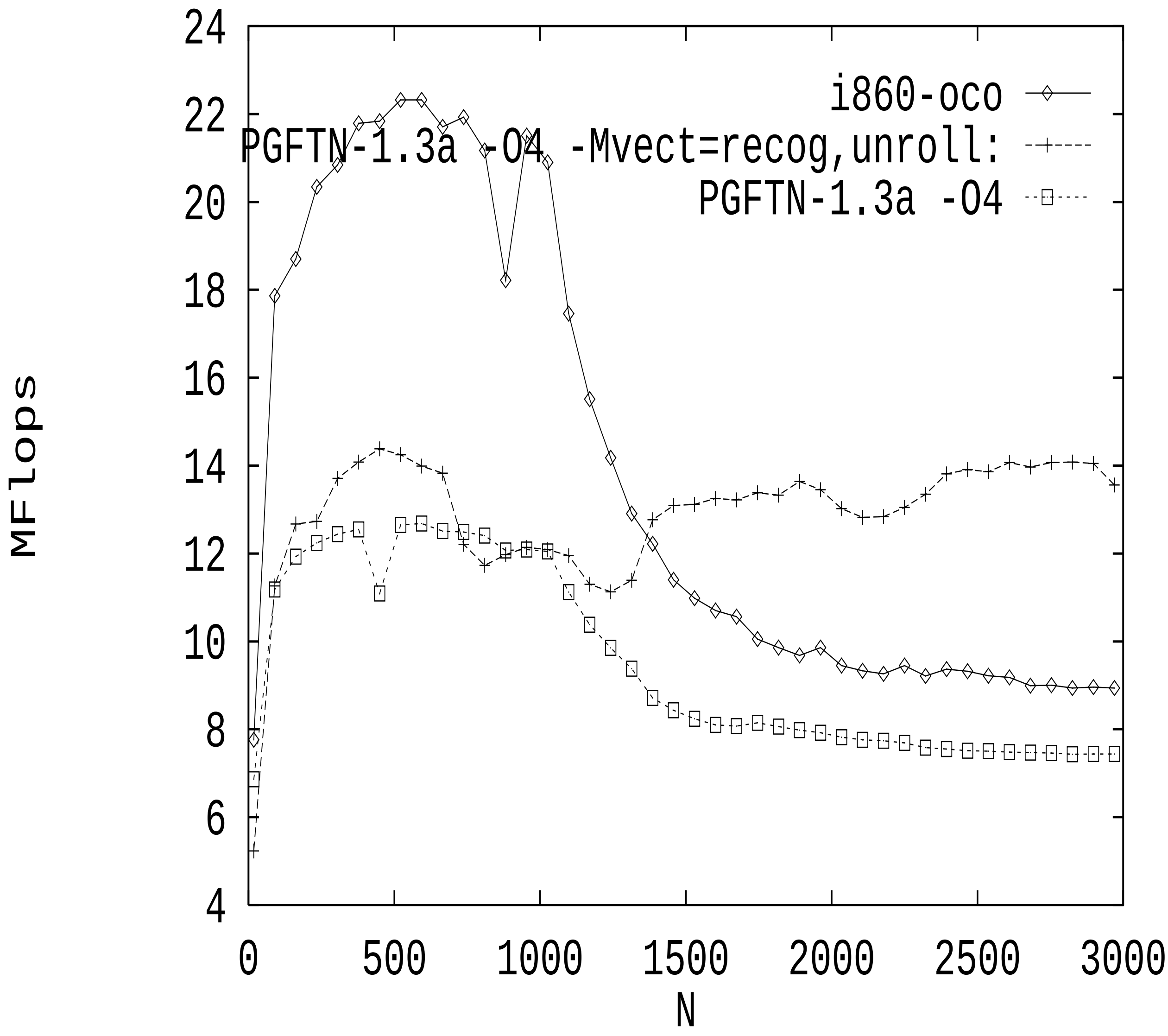
```
      .xxx LOOP BEGIN
Loop:
      .xxx mem "READ_FROM" 1
      fld.d 8(r7)++, v90           // @X++;load X(i)
      fmul.dd v90, f8, v92         // A*X(i)
      fadd.dd f10, v92, v94        // A*X(i) + B
      bla r4, r5, Loop
      .xxx mem "WRITE_TO" 2
      fst.d v94, 8(r6)++           // @Y++;store Y(i)
      .xxx LOOP END
```

Exemple (2)

Cycle	0	1	2
core_issue	L(i)	B	S(i-6)
bus			
fld_1			
fld_2			
fld_3			
fld_res			L(i)
fpu_issue	*(i-1)	+(i-3)	
fpu_mul_1		*(i-1)	*(i-1)
fpu_mul_2		*(i-2)	*(i-2)
fpu_mul_res		*(i-3)	
fpu_add_1			+(i-3)
fpu_add_2			+(i-4)
fpu_add_3			+(i-5)
fpu_add_res			+(i-6)

Issue		result
0	L	2
3	*	10
10	+	20
20	S	

Exemple (3)



24
22
20
18
16
14
12
10
8
6
4
MFlops
i860-oco
PGFTN-1.3a -O4 -Mvect=recog,unroll:
PGFTN-1.3a -O4
0
500
1000
1500
2000
2500
3000
N

Commentaires

Code	Pipeline Logiciel	Lecture Mémoire pipeliné	Vectorisation
i860-oco fld.d	$\surd$		
PGFTN-1.3a -04	$\surd$		
PGFTN-1.3a -04 -Mvect=recog,unroll	$\surd$	$\surd$	$\surd$

<u>Code Vectoriel</u>

Code source

```
      DO 1 k = 1,N
1        Y(k)= A * X(k) + B
```

Code generé par PGFTN + vect (retranscrit en FORTRAN)

```
      double precision vcache(1024)
      DO 1 ii = 1,N,1024
         DO 10 i = ii,MIN(N, ii + 1024),1
10          vcache(i-ii+1) = A * X(i)
         DO 11 i = ii,MIN(N, ii + 1024),1
11          Y(i) = vcache(i-ii+1) + B
1 CONTINUE
```

- cache est géré comme des registres vectoriels
- les lectures mémoire pipelinées sont efficaces en raison de la bonne localité spatiale (pas d'accès de 1) et de la fréquence élevée des requêtes

Des mesures de vitesse

Boucle FORTRAN :

```
       DO 200 I = 1, N
         C(I) = A(I)+B(I)
 200   CONTINUE
```

Différents niveaux d'optimisation

Modification 1

(Loads en avance d'une itération)

```
      DO 20 I = 3, NN,2
        R1=A(I)
        R2=B(I)
        R6=R4+R5
        C(I-2)=R3
C
        R4=A(I+1)
        R5=B(I+1)
        R3=R1+R2
        C(I-1)=R6
 20   CONTINUE
```

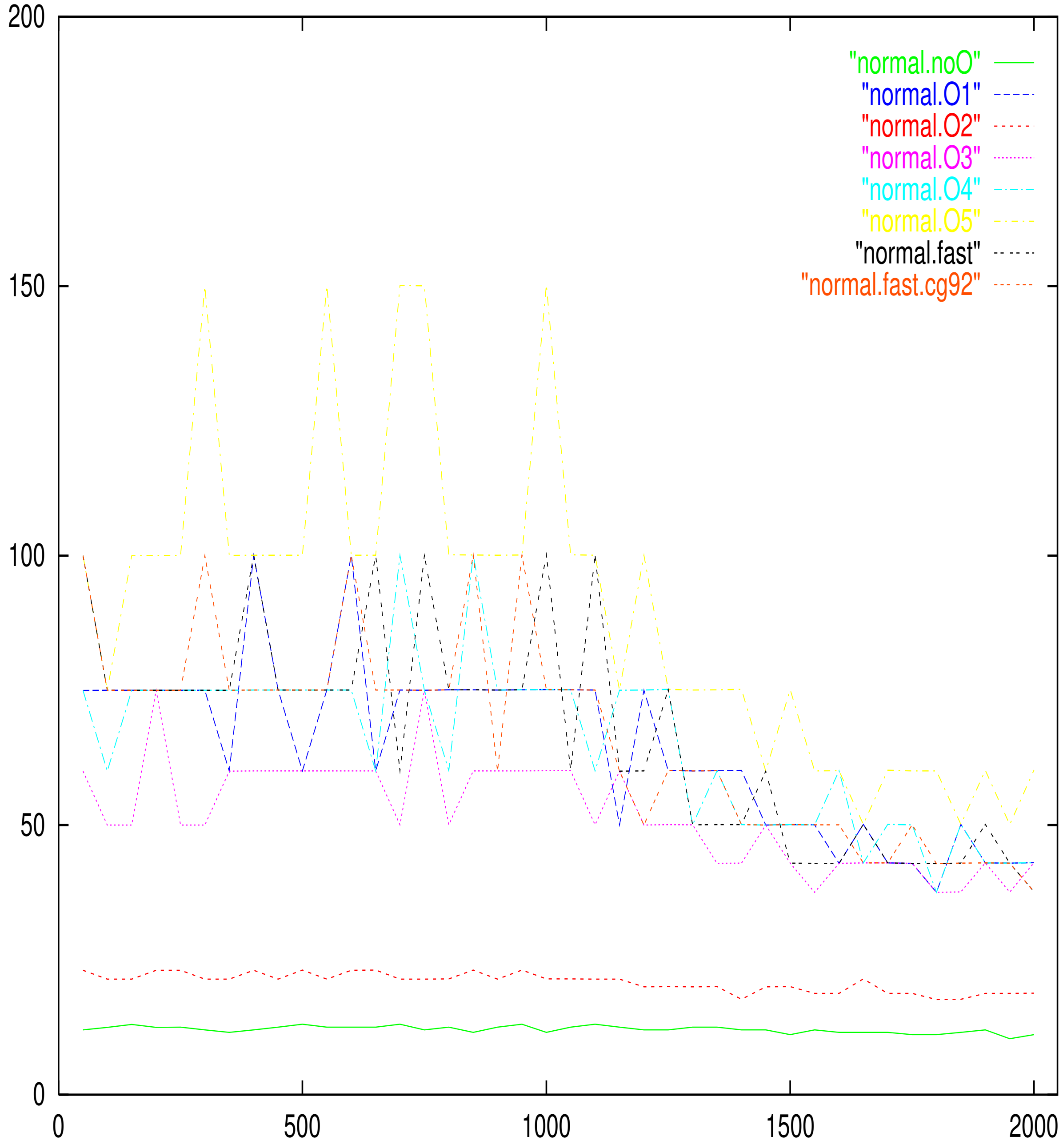
Modification 2

(Loads en avance de deux itérations)

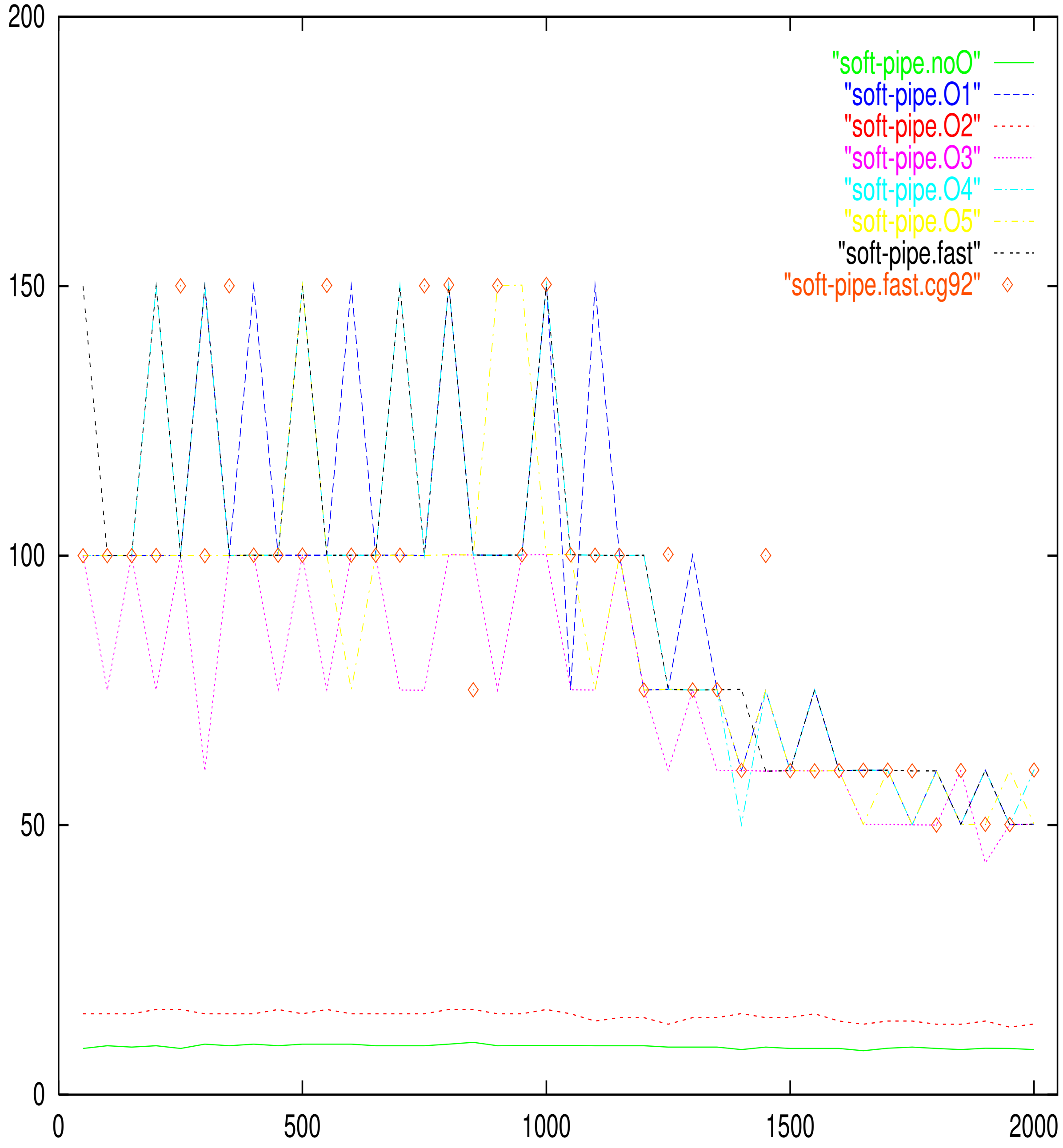
```
      DO 200 I = 1,NN - 2,3
      R6 = R4 + R5
      R1 = A(I)
      R2 = B(I)
      C(I-3) = R3

      R9 = R7 + R8
      R4 = A(I+1)
      R5 = B(I+1)
      C(I-2) = R6

      R3 = R1 + R2
      R7 = A(I+2)
      R8 = B(I+2)
      C(I-1) = R9
200   CONTINUE
```

INRIA

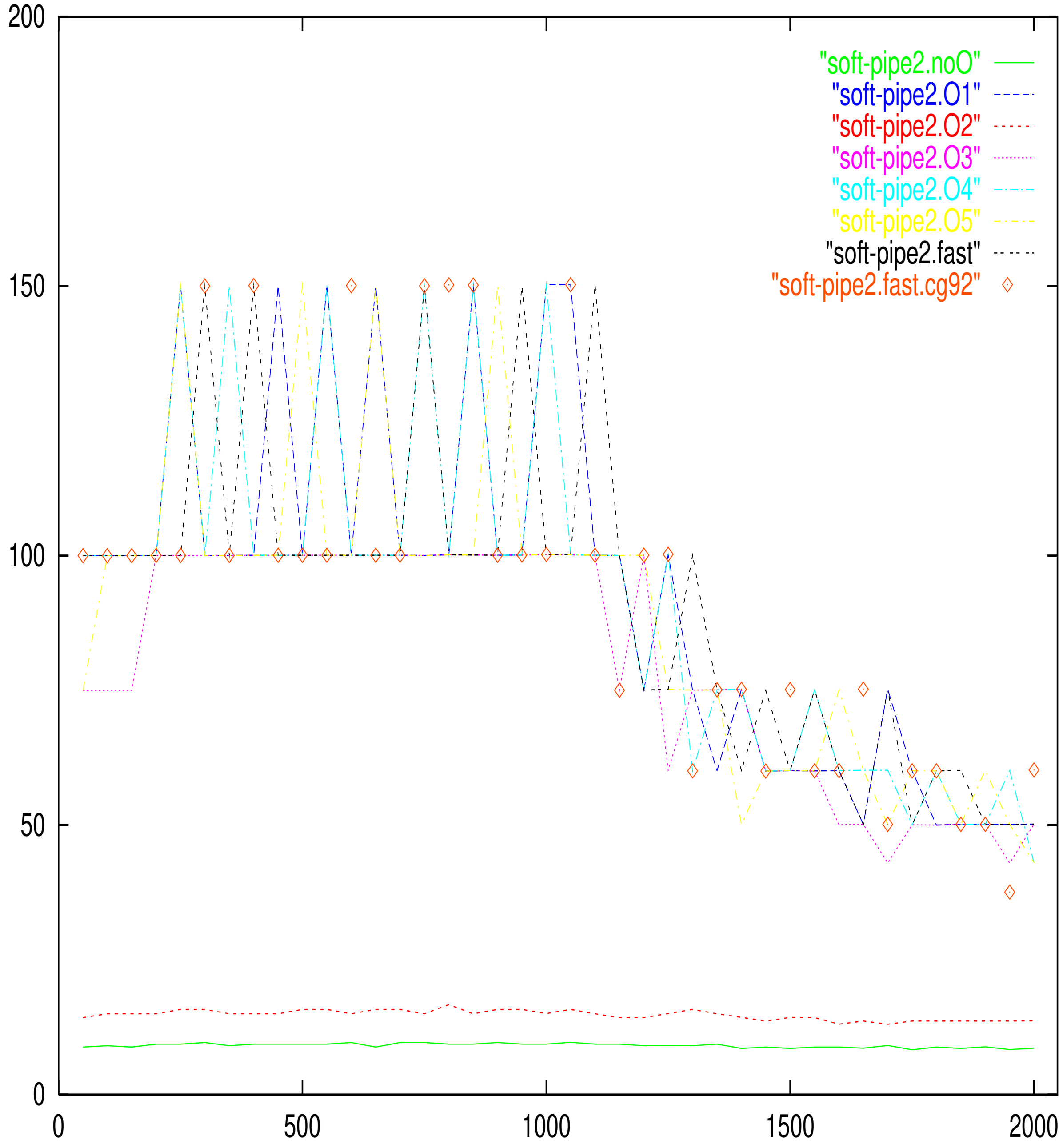
SuperSPARC



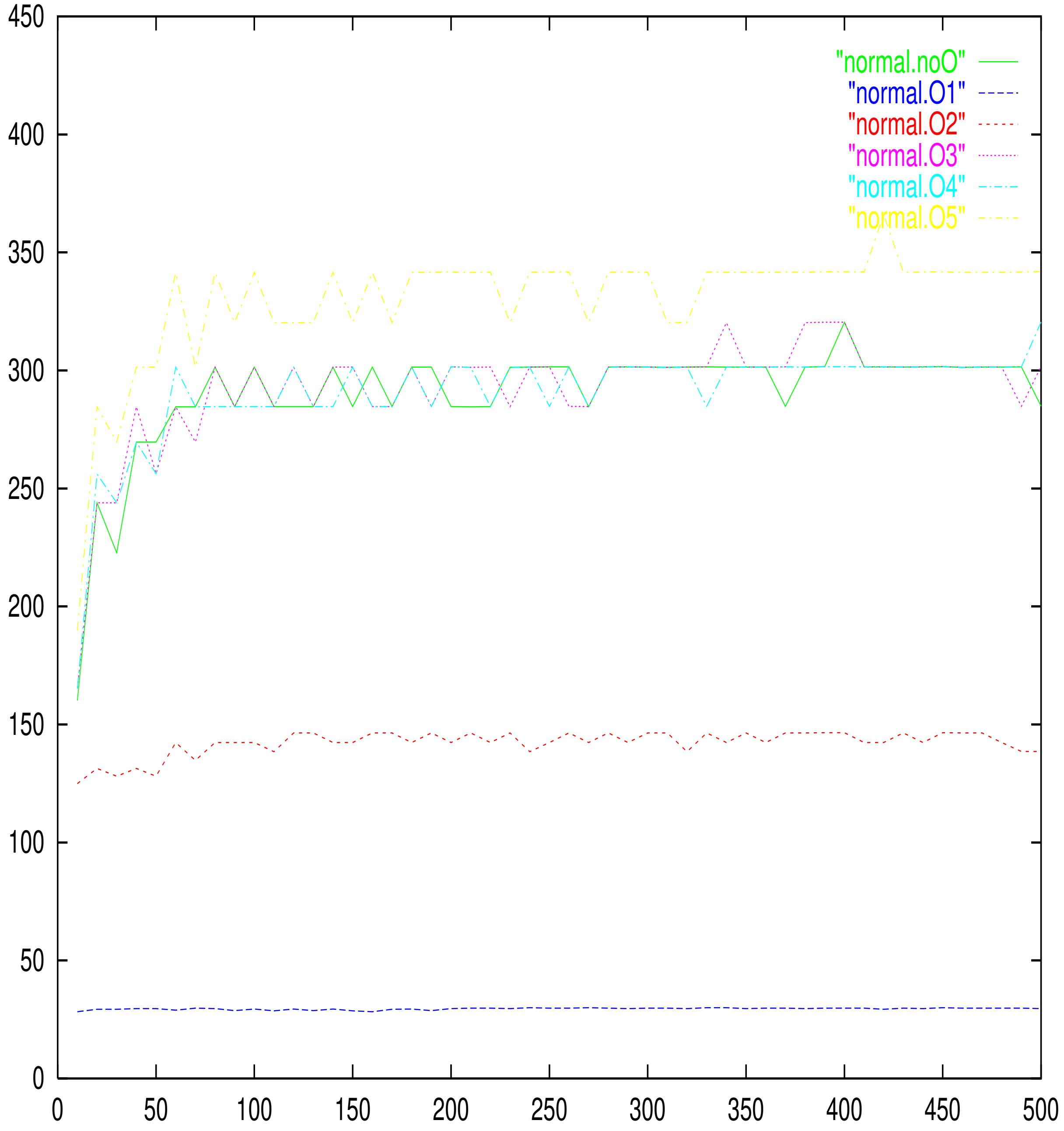
SuperSPARC - modif 1



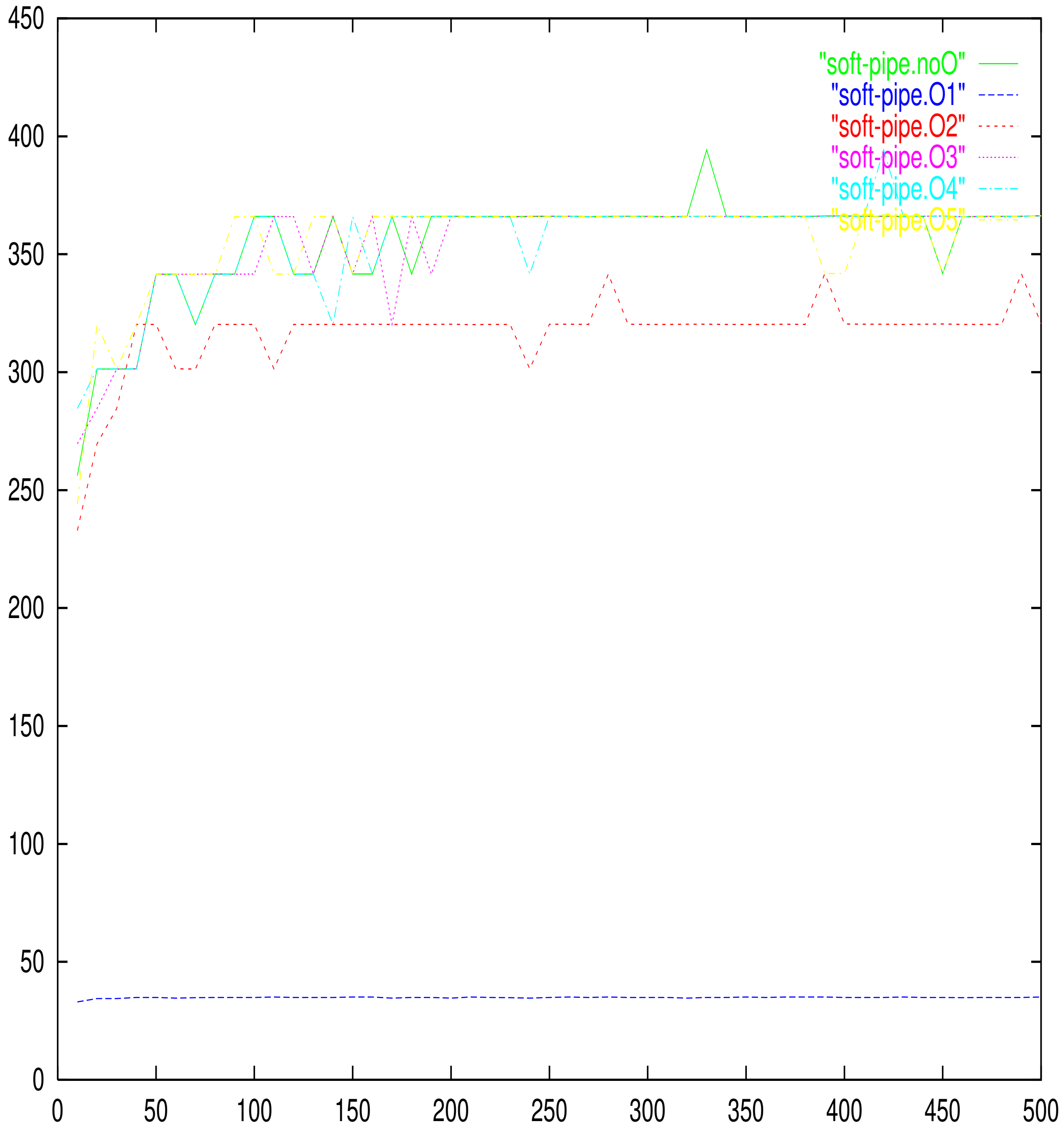
SuperSPARC - modif 2



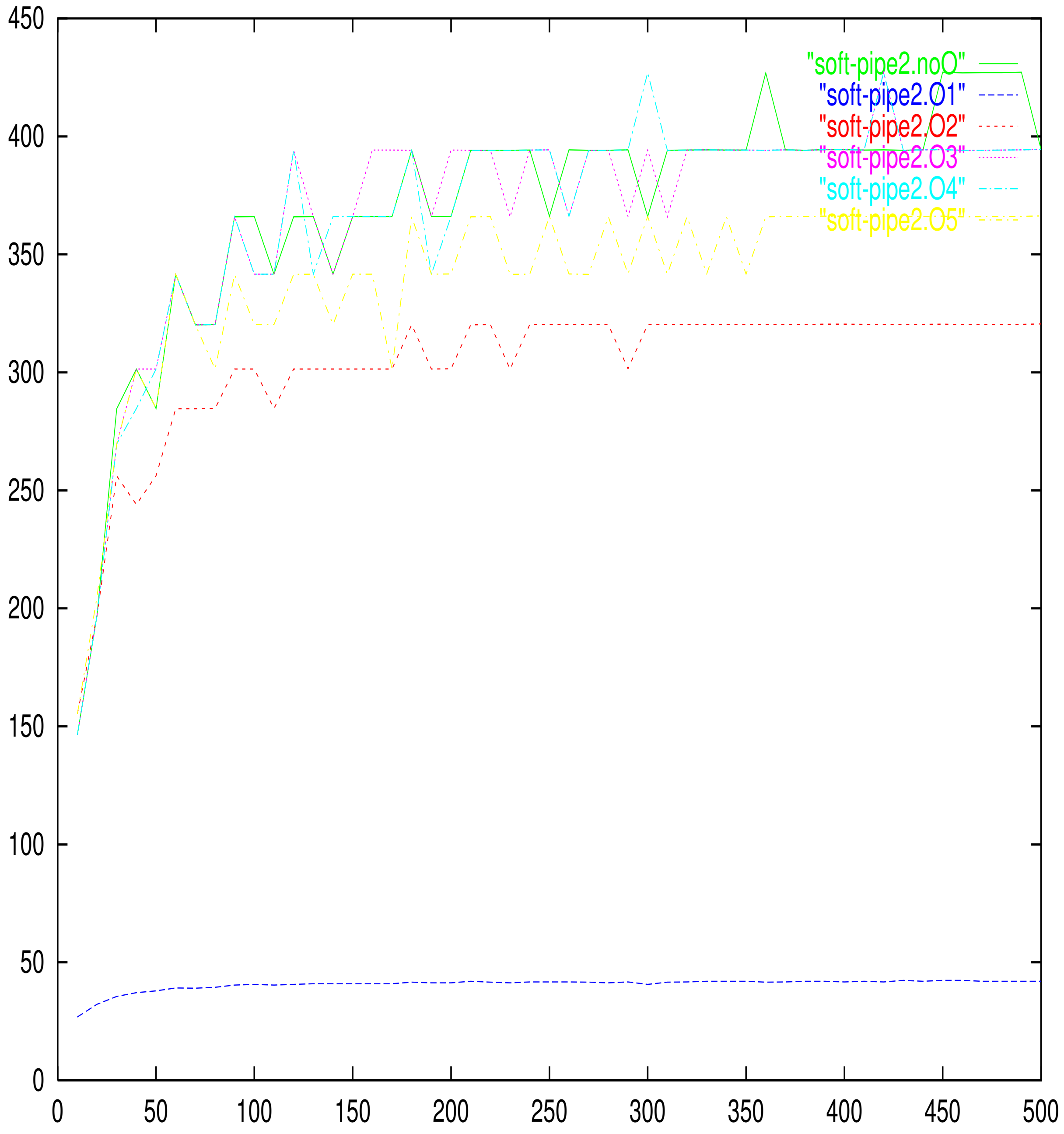
DEC-Alpha, taille 500



DEC-Alpha, taille 500 - modif 1

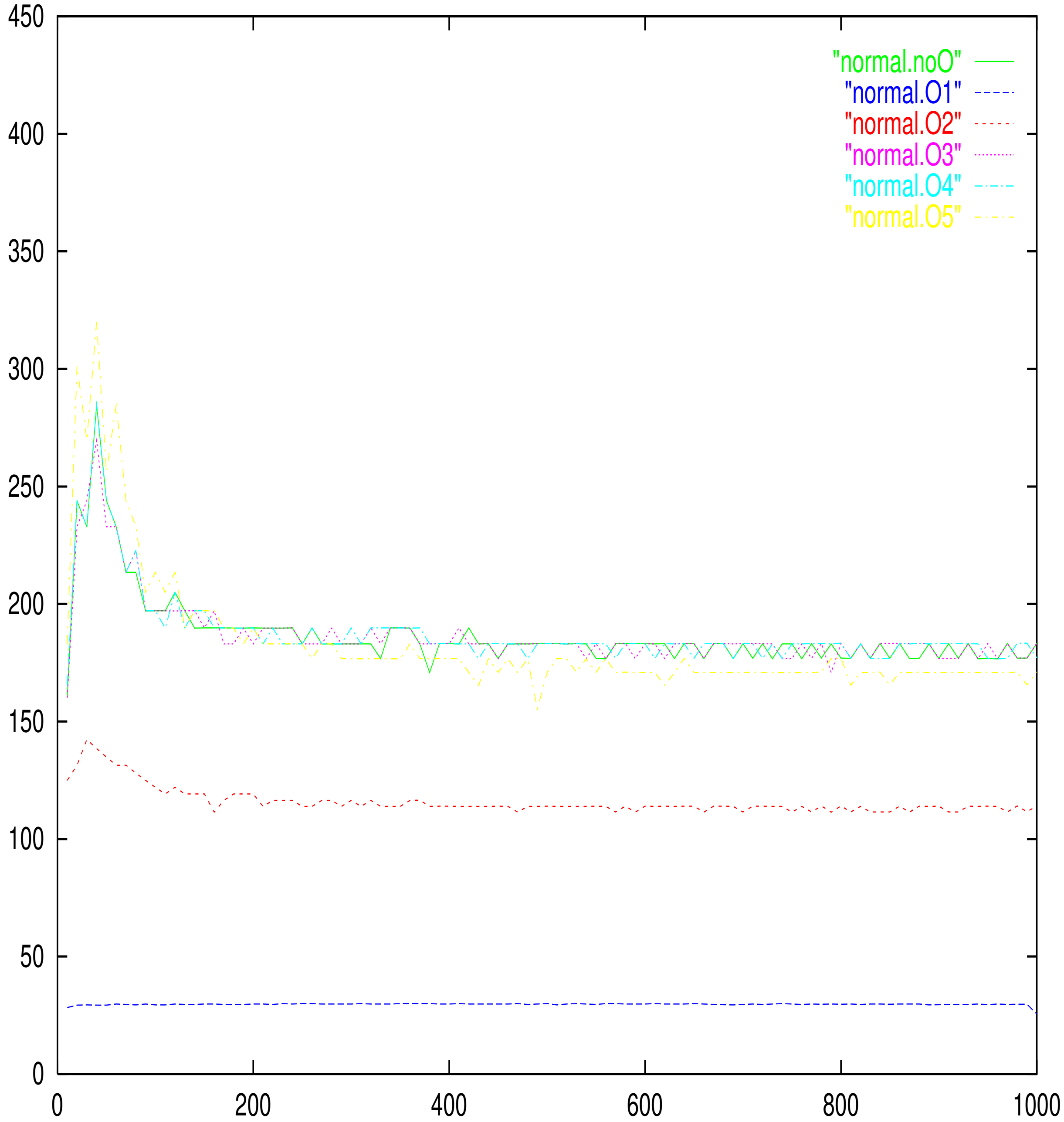


DEC-Alpha, taille 500 - modif 2

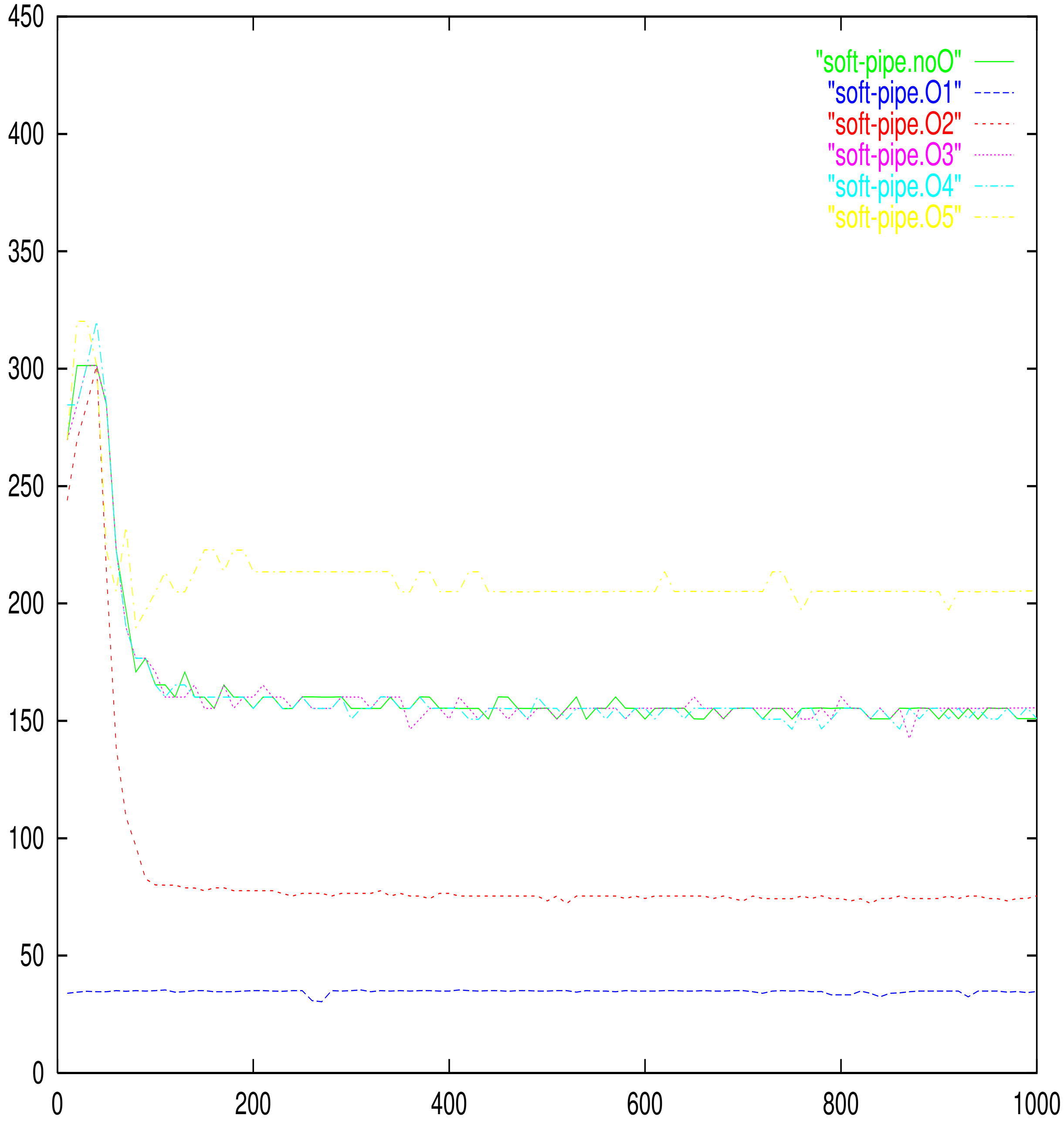


INRIA

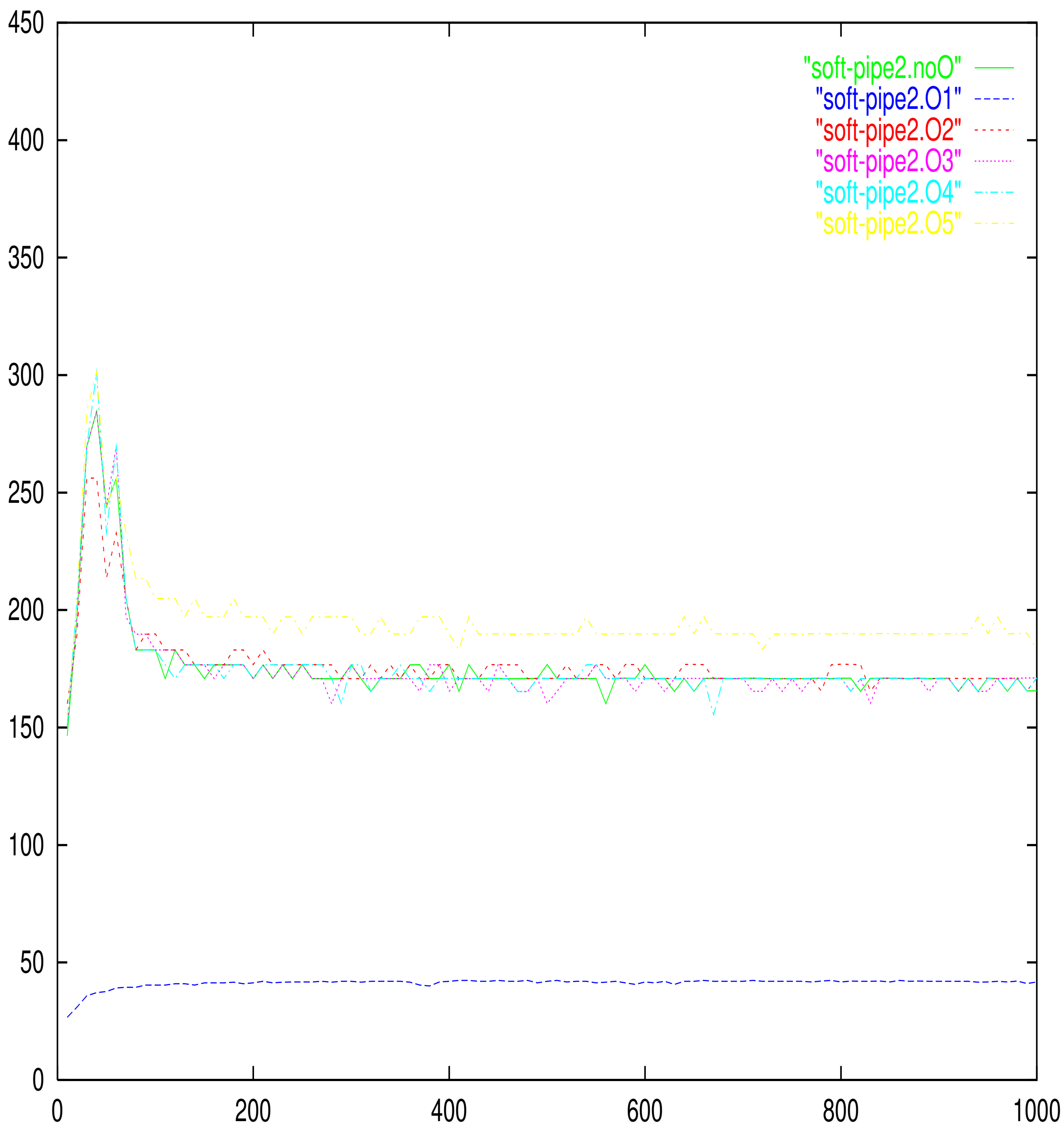
DEC-Alpha, taille 1000



DEC-Alpha, taille 1000 - modif 1



DEC-Alpha, taille 1000 - modif 2



Conclusion

L'optimisation du parallélisme à grain fin est

- bien connue en théorie (par séparation des problèmes)
- peu implantée en pratique dans les compilateurs
  - ingéniérie des compilateurs
  - interaction avec d'autres problèmes (hiérarchie mémoire, parallélisation)
- primordiale pour les processeurs à venir

Le processeur connaît le présent, le compilateur connaît le futur ... !