

---

## Optimisation de programmes : Principes généraux

Daniel Etiemble  
de@lri.fr

---

## Résumé

- Mesures de performance
- IA-32 et RISC
- Parallélisme d'instructions : scalaire, superpipeline et superscalaire
- Dépendances de données, suspensions, déroulage de boucle et pipeline logiciel
- Caches
- Instructions SIMD

## Programmeur et performance

---

- Programmes compilés par un compilateur optimisant
  - Niveau d'optimisation
- Performances intrinsèques
  - Structure interne du processeur
    - Jeu d'instructions et microarchitecture
      - IA-32 ou autres (RISC, IA-64...)
      - Pipelines, prédiction de branchement
  - Hiérarchie mémoire
    - Taille et organisation des caches, TLB, Mémoire virtuelle
- Impact du programmeur : optimisations
  - Hiérarchie mémoire
    - Structure des accès
    - Instructions de préchargement
  - Parallélisme SIMD
    - Instructions « multimédia »
  - Parallélisation OpenMP
    - Processeurs « multithread »

## Evaluation de performance

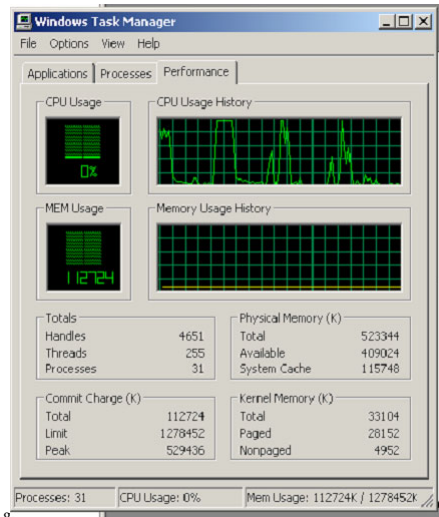
---

- Temps d'exécution du programme
- Détection des goulots d'étranglement
  - Détection des fonctions consommatrices de temps
  - Comprendre la cause du goulot d'étranglement
- Quelques outils
  - Perfmon
  - Compteurs matériels
  - PROF et GPROF
  - PAPI
  - VTune

## Outils élémentaires d'évaluation de performance

PERFMON  
Gestionnaire de tâches Windows

Information qualitative  
Petit ensemble d'évènements

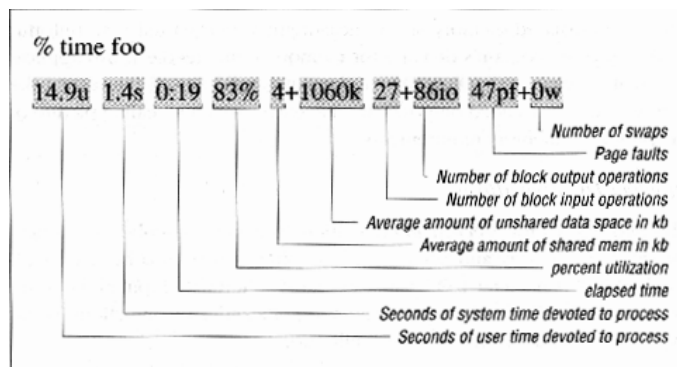


Master Pro  
2005

Optimisations pour g  
D. Etienne

## Mesurer le temps d'exécution (Linux)

- Commande Time



Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etienne

6

## Mesurer le temps d'exécution

---

- <time.h> (Windows)
  - Déclaration
    - clock\_t: begin, end
  - Mesure
    - begin=clock(); // début
      - Temps d'exécution à mesurer
    - end=clock();
    - printf("« temps d'exécution= %f\n", (double) (end-begin));
  - Précis pour des temps d'exécution supérieurs à la ms

## Mesure du temps d'exécution

---

- RDTSC (Read-time stamp counter)
  - Déclaration
    - #define CPUID \_\_asm \_\_emit 0fh \_\_asm \_\_emit 0a2h
    - #define RDTSC \_\_asm \_\_emit 0fh \_\_asm \_\_emit 031h
  - Fonction
    - unsigned \_\_int64 timercall();
  - Mesure

```
temp_cycles1=timercall(); temp_cycles2=timercall();
base= temp_cycles2 - temp_cycles1;
temp_cycles1=timercall();
//CODE A MESURER
temp_cycles2=timercall();
elapsed= temp_cycles2 - temp_cycles1 - base;
printf("te(cycles)= %f\n" ,(double)(elapsed));
```

## Instruction RDTSC (IA-32 sous Windows)

```
unsigned __int64 timercall(){
    unsigned cycles_low, cycles_high;
    __asm {
        pushad
        CPUID
        RDTSC
        mov     cycles_high, edx
        mov     cycles_low, eax
        popad }

    return ((unsigned __int64)cycles_high << 32)
           | cycles_low;}

```

Mesure: cycles d'horloge

## Instruction RDTSC (IA-32 sous Linux)

```
long long readTSC(void)
{
    long long t;
    asm volatile (".byte 0x0f,0x31" : "=A" (t));
    return t;
}

double dtime()
{
    return (double) readTSC();
}

```

Mesure: cycles d'horloge

## Outils de profilage

---

- PROF (Linux)
  - Temps passé dans chaque fonction ou sous-programme
  - Utilisation
    - `gcc -o file -p file.c` (compile and link with -p option)
    - `file` (run program and produces 'mon.out')
    - `prof file > file.prof`
  - Exemple
- VTune (Windows)

## Sortie Prof

---

Liste de fonctions, triée en ordre décroissant

[index]	secs	%	cum.%	samples	function (dso: file, line)
[1]	23.380	43.0%	43.0%	2338	scatter (montecarlo: montecarlo.c, 266)
[2]	8.410	15.5%	58.5%	841	ran3 (montecarlo: montecarlo.c, 439)
[3]	6.050	11.1%	69.6%	605	__fastm_cos (libfastm.so: cos.s, 73)
[4]	5.730	10.5%	80.2%	573	path (montecarlo: montecarlo.c, 216)
[5]	5.560	10.2%	90.4%	556	__fastm_log (libfastm.so: log.s, 100)
[6]	2.660	4.9%	95.3%	266	getRandomNumber (montecarlo: montecarlo.c, 481)
[7]	2.470	4.5%	99.9%	247	doAnisotropicScatter (montecarlo: montecarlo.c, 119)
[8]	0.040	0.1%	99.9%	4	__fastm_exp (libfastm.so: exp.s, 92)
[9]	0.020	0.0%	100.0%	2	__atan (libm.so: atan.c, 145)
[10]	0.010	0.0%	100.0%	1	__write (libc.so.1: write.s, 20)
	0.010	0.0%	100.0%	1	**OTHER** (includes excluded DSOs, rld, etc.)

## Profilage via graphe d'appels

- GPROF (Linux)

- gcc -o file -pg file.c (compile and link with -pg option)
- file ( run program and produces 'gmon.out')
- gprof file > file.gprof

### Exemple

called/total index	%time	parents		called+self called/total	name	index
		self	descendents			
		3.74	27.14	7/7	.main [1]	
[3]	46.2	3.74	27.14	7	.doAnisotropicScatter [3]	
		9.90	27621579/27621579		.scatter [4]	
		3.35	4.61 27691579/27691579		.path [9]	
		0.02	0.03 70000/70000		.initPhoton [16]	
		0.03	0.00 70000/70000		.score [18]	
		0.00	0.00 21/53		.printf [55]	
		0.00	0.00 7/7		.printTissueParams [93]	
		0.00	0.00 7/7		.clearIntensityArray [91]	
		0.00	0.00 7/7		.dump1 [92]	

## Compteurs matériels

- Pratiquement tout processeur disponible a des compteurs matériels
- Les interfaces et la documentation sont pauvres ou inexistantes
- Le matériel et la sémantique diffèrent selon les CPU
- Utile pour les mesures, l'analyse, l'optimisation, la modélisation et l'évaluation de performance.

## Données qui peuvent être disponibles

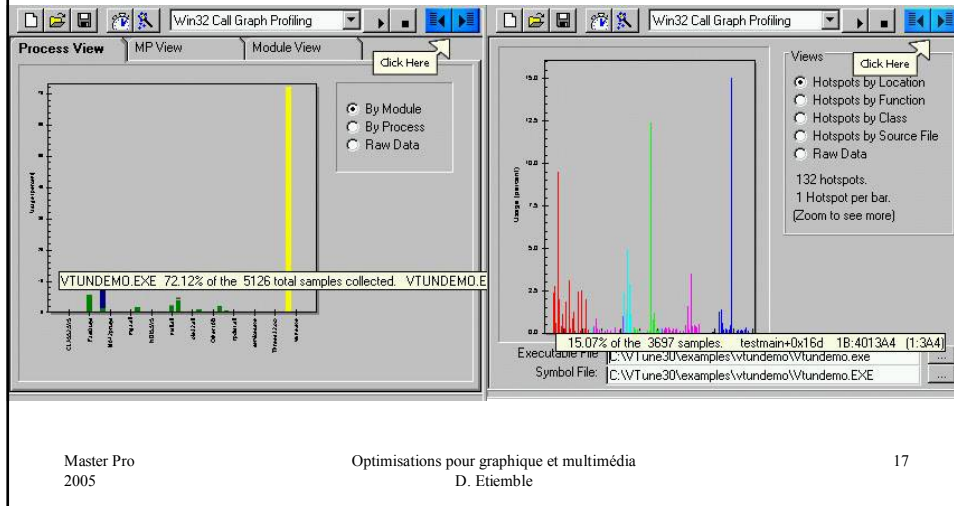
- Nombre de cycles
- Nb d'instructions flottantes
- Nb d'instructions entières
- Nb de chargements/rangements
- Nb de branchements pris/non pris
- Mauvaises prédictions de branchement
- Suspensions pipeline dues au système mémoire
- Suspensions pipeline dues aux conflits de ressources
- Echecs caches I/D pour différents niveaux
- Invalidations cache
- Echecs TLB
- Invalidation TLB

## VTune (Intel)

- L'analyseur VTune fournit trois types de techniques d'analyse pour différents aspects de la performance de l'application
  - Analyse basée sur le temps/événements pour exécuter et échantillonner le programme et identifier les goulots d'étranglement dans le code
  - Analyse du code statique pour analyser les fonctions statiques et indiquer les problèmes de performance
  - Profilage du graphe d'appel pour suivre chaque thread et séquence d'appels qui prennent le plus de temps

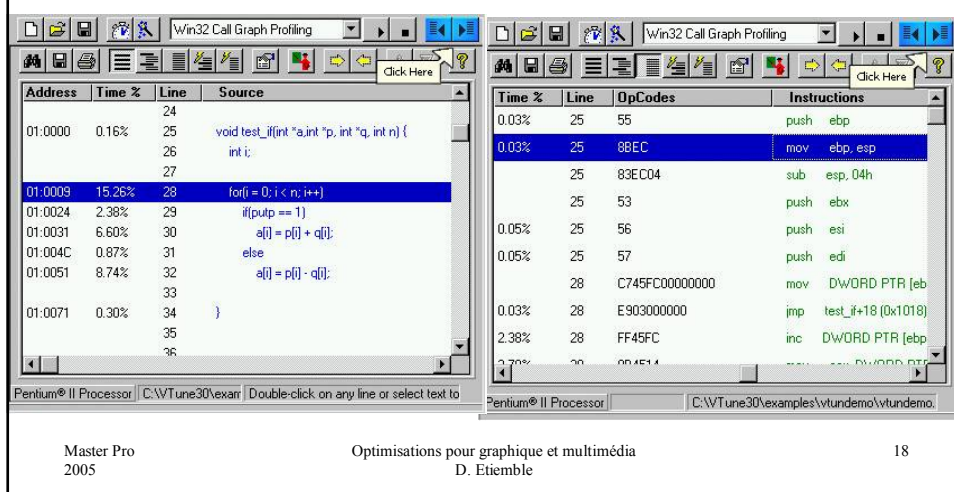
# VTune

## Détection des goulots d'étranglement



# VTune (Intel)

## Localisation des goulots d'étranglement



# VTune

## Grappe d'appel

The screenshot displays the VTune interface. On the left, a tree view shows the execution context: VTUNDEMO, CallGraph, Chronologies, OS Chronologies, Processor Chronologies, Code Analysis, VTUNDEMO.EXE, Dynamic Analysis, Options, Sampling Sessions, and Systems. The 'Functions' pane lists various functions with their offsets and lengths. The main area shows a call graph starting with 'Appmain\_init', which calls 'MandPanel.<init>', 'MandControls.<init>', and 'Container.add'. 'MandPanel.<init>' calls 'Panel.<init>', which in turn calls 'Container.add'. 'MandControls.<init>' calls 'Button.<init>', which calls 'Component.<init>'. 'Container.add' calls 'Frame.addImpl', which calls 'Window.addImpl', which calls 'Container.addImpl', which calls 'BorderLayout.addLayoutComponent', which finally calls 'BorderLayout.addLayoutComponent'.

Function Name	Offset	Length
divd_rout	01.00000128	82
test_i	01.00000000	118
test_i1	01.00000076	123
test_memset	01.000000F1	58
test_oror1	01.000001DA	93
test_oror2	01.0000017D	93
testmain	01.00000237	664
WinMain	01.000006CB	264
WndProc	01.000004CF	508

# Modèles RISC et IA-32

- Modèles d'exécution (n,m)
  - n : nombre d'opérandes par instruction
  - m : nombre d'opérandes mémoire par instruction
- RISC : (3,0)
  - Instructions de longueur fixe
  - Load et Store : seules instructions mémoire
- IA-32 : (2,1)
- Pile (0,0)
  - Tous les opérandes sont accédés via la pile

## Caractéristiques IA-32

- Instructions de longueur variable

Op code	Reg. et M	Déplacement	Immédiat	
1 or 2	1 or 2	0, 1, 2 or 4	0, 1, 2 or 4	octets

– Inst dest, source

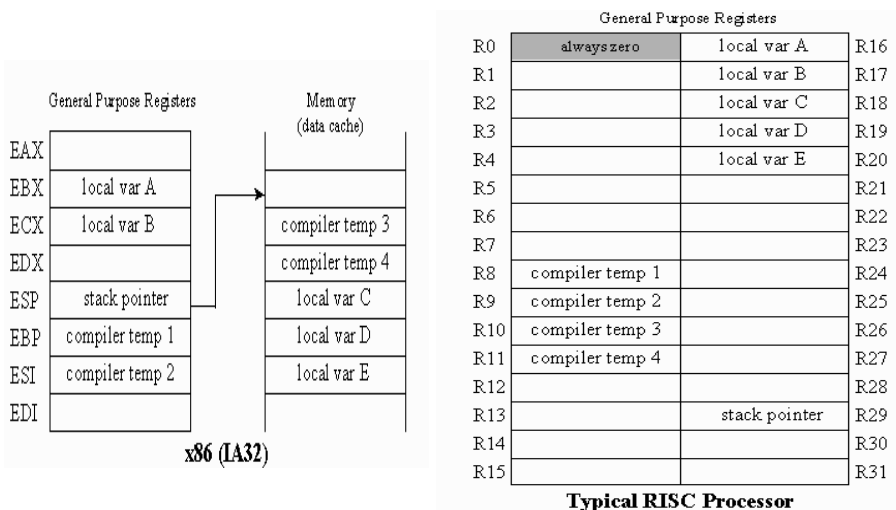
<b>REG</b>	<b>REG</b>
<b>REG</b>	<b>MEM</b>
<b>REG</b>	<b>IMM</b>
<b>MEM</b>	<b>REG</b>
<b>MEM</b>	<b>IMM</b>

Lecture mémoire,  
Exécution,  
Ecriture mémoire

- Instructions complexes
  - Rep
- Modes d'adressage complexes

$$\text{Adresse mémoire} = \text{Rb} + \text{RI} \times \text{f} + \text{déplacement}$$

## Registres: organisation x86 et RISC



## Traduction des instructions x86

---

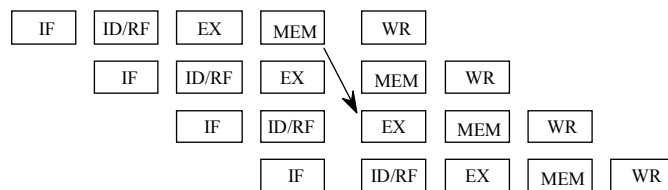
Pentium Pro, PII, PIII, P4

Instructions x86	Opérations RISC
add EAX, [EBP +d8]	load temp, [EBP + d8] add EAX, temp
add [EBP +d8], EAX	load temp, [EBP + d8] add EAX, temp store EAX, [EBP+8]
cmp EAX, imm32	cmp EAX, imm32
push ECX	sub ESP, 4 store [ESP], ECX

## Les délais liés au pipeline

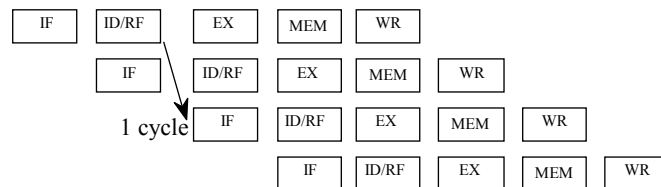
---

### DELAI DE CHARGEMENT



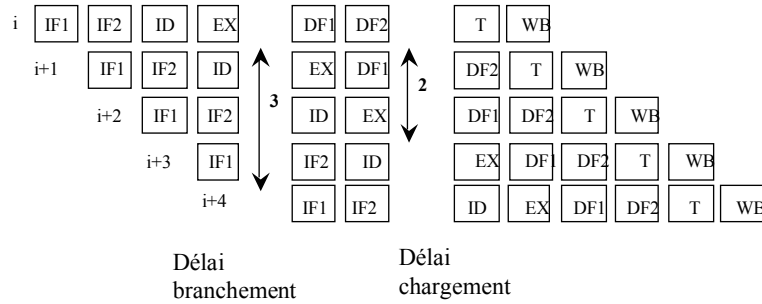
### DELAI DE BRANCHEMENT

1 cycle



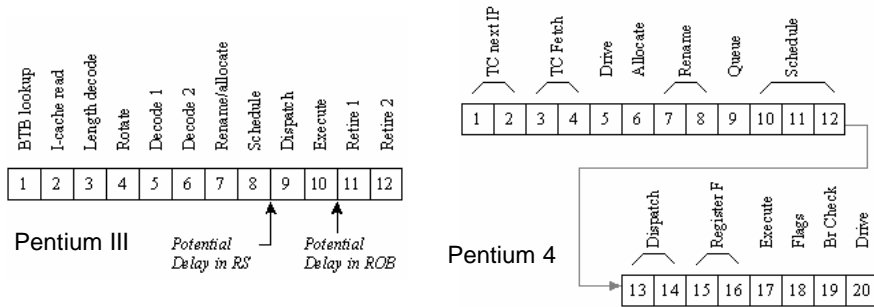
## Superpipelines : chargements et branchements

### Superpipeline MIPS R4000



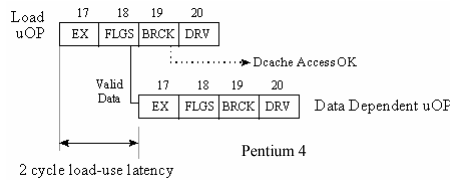
## Pipelines Pentium III et Pentium 4

- Superpipeline : technique permettant d'utiliser des fréquences d'horloge élevée (2 à 3 GHz en 2003)



## Latences chargement/branchement

### Latence de chargement



### Pénalité de mauvaise prédiction

Basic Pentium® III Processor Misprediction Pipeline									
1	2	3	4	5	6	7	8	9	10
Fetch	Fetch	Decode	Decode	Decode	Rename	ROB Rd	Rdy/Sch	Dispatch	Exec

Basic Pentium® 4 Processor Misprediction Pipeline																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
TC/Not IP	TC/Fetch	Drive/Alloc	Rename	Que	Sch	Sch	Disp/Disp	RF	RF	Ex	Flgs	Br Ck	Drive						

- Optimisation matérielle
  - Prédiction de branchement
- Optimisation programmeur/compilateur
  - Ordonnancement des instructions
  - Conversion SI
    - Utilisation des instructions de transfert conditionnel pour supprimer des branchements conditionnels

## Les opérations multicycles

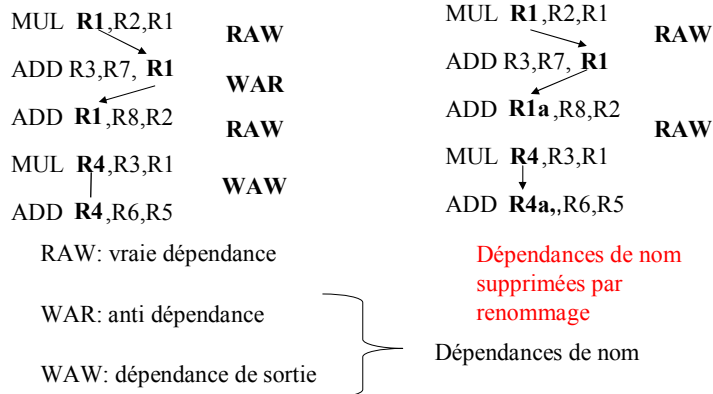
- Latence et débit des opérations flottantes double précision

	Latence					Débit				
	+	*	*+	/	SQR	+	*	*+	/	SQR
21064	6	6	-	61	-	1	1		61	
21164	4	4	-	22/60	-	1	1		22/60	
21264	4	4	-	15	32	1	1		13	30
US-1	3	3	-	22	22	1	1		22	22
US-3	4	4	-	17	24	1	1		17	24
R10000	2	2	-	19	33	1	1		21	35
PA-8000	3	3	3	31	31	1	1	1	31	31
P2SC	2	2	2	?	?	1	1	1	?	?
P4-x87	5	7	-	38	38	1	2	-	38	38
P4-SSE2	4	6	-	35	-	2	2	-	35	-

## Les dépendances de données

---

**Exemple:**



## Traitement des dépendances

---

- Dépendances et parallélisme
  - Les dépendances producteur - consommateur doivent être respectées (sémantique du programme)
  - Des opérations/instructions/programmes ne peuvent être exécutés en parallèles que s'ils sont *indépendants*
- Parallélisme d'instructions dans un processeur
  - Contrôle des dépendances par matériel
    - Mais introduit des suspensions
  - Suppression des dépendances de nom par matériel
  - Techniques logicielles pour supprimer les suspensions
    - Déroulage de boucle
    - Pipeline logiciel

## Multiplication de matrices : SAXPY

### IJK

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++){
    x[i][j]=0;
    for (k=0; k<N; k++)
      x[i][j]+=y[i][k]*z[k][j];}
```

Produit scalaire

```
for (k=0; k<N; k++)
  r+=Y[k]*Z[k];
```

### IKJ

```
for (i=0; i<N; i++)
  for (k=0; k<N; k++)
    for (j=0; j<N; j++)
      x[i][j]+=y[i][k]*z[k][j];
```

SAXPY ou DAXPY

```
for (j=0; j<N; j++)
  X[j]+=Y*Z[j];
```

## Dépendances de données et suspensions

### EXEMPLE

```
Do for i = 0 to N-1 step 1
  Begin
    Y(i) = A x X(i) + Y(i)
  End
```

DAXPY

Latence des instructions (cycles)

<i>Source</i>				
<i>Dest.</i>	UAL	LD/ST (données)	Opérations FP	
UAL	2	2		
LD/ST (adresses)	2	2		
ST (données)	1	1	4	
Opérations flottantes		2	5	

## Performance DAXPY

### Boucle non optimisée

Boucle	1	LD F1, (R1)	; charge X(i)
	2	LD F2, (R2)	; charge Y(i)
	3	MULTD F1, F0, F1	; a * X(i)
	4		
	5		
	6		
	7		
	8	ADDD F2, F2, F1	; a * X(i) + Y(i)
	9	SUB R6, R7, R1	; compare i et N-1
	10	ADDI R1, R1, 8	; adresse X(i+1)
	11	ADDI R2, R2, 8	; adresse Y(i+1)
	12	SD F2, -8(R2)	; range Y(i)
	13	BNEQ R6, Boucle	; si I<N-1, branchement

13 cycles (4 cycles sont perdus)

## Déroutage de boucle

Loop	1	LD F1, (R1)	; charge X(i)	14	ADDD F2, F2, F1	; a * X(i) + Y(i)
	2	LD F3, 8(R1)	; charge X(i+1)	15	ADDD F4, F4, F3	; a * X(i+1) + Y(i+1)
	3	LD F5, 16(R1)	; charge X(i+2)	16	ADDD F6, F6, F5	; a * X(i+2) + Y(i+2)
	4	LD F7, 24(R1)	; charge X(i+3)	17	ADDD F8, F8, F7	; a * X(i+3) + Y(i+3)
	5	LD F2, (R2)	; charge Y(i)	18	SD F2, (R2)	; range Y(i)
	6	LD F4, 8(R2)	; charge Y(i+1)	19	SD F4, 8(R2)	; range Y(i+1)
	7	LD F4, 16(R2)	; charge Y(i+2)	20	SD F6, 168(R2)	; range Y(i+2)
	8	LD F4, 24(R2)	; charge Y(i+2)	21	SD F8, 24(R2)	; range Y(i+3)
	9	MULTD F1, F0, F1	; a * X(i)	22	ADDI R1, R1, 32	; adresse X(i+4)
	10	MULTD F3, F0, F3	; a * X(i+1)	23	ADDI R2, R2, 32	; adresse Y(i+4)
	11	MULTD F5, F0, F5	; a * X(i+2)	24	BNEQ R6, Loop	; si I<N-4, branch
	12	MULTD F7, F0, F7	; a * X(i+3)			
	13	SUB R6, R7, R1	; compare i et N-4			

**6 cycles/itération  
au lieu de 13**

```
For (i=0; i<N; i+=4) {
    Y[i]+=A*X[i];
    Y[i+1]+=A*X[i+1];
    Y[i+2]+=A*X[i+2];
    Y[i+3]+=A*X[i+3];
}
```

\* Plus de suspensions (4 cycles)  
\* 1 cycle de gestion de boucle  
(4 inst./4) au lieu de 4 (-3 cycles)

## Pipeline logiciel

### DAXPY    PROLOGUE

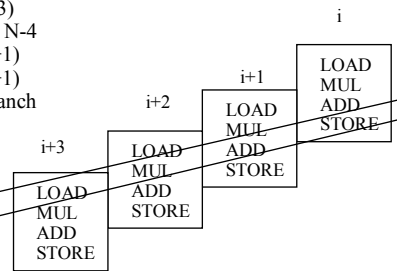
9 cycles (pas de suspension)  
+ surcoût (prologue + épilogue)

```

LOOP  1   SD F4, 0(R2)      ; range Y(i)
      2   ADDD F4,F2,F3    ; a * X(i+1) + Y(i+1)
      3   MULTD F3, F0, F1 ; a * X(i+2)
      4   LD F1, 24(R1)    ; charge X(i+3)
      5   LD F2, 24(R2)    ; charge Y(i+3)
      6   SUB R6,R7,R1     ; compare i et N-4
      7   ADDI R1,R1,8     ; adresse X(i+1)
      8   ADDI R2,R2,8     ; adresse Y(i+1)
      9   BNEQ R6, LOOP    ; si I<N-4, branch
    
```

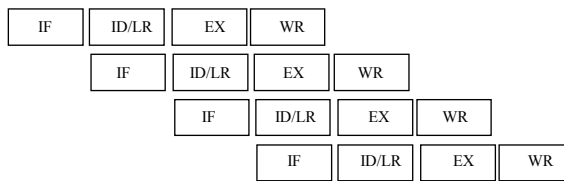
### EPILOGUE

**Boucle  
pipelinée  
par logiciel**

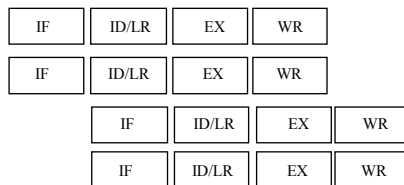


## Principes des superscalaires

### pipeline

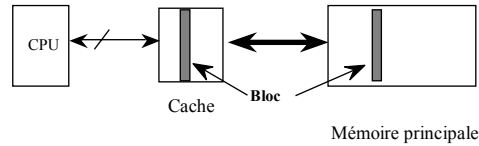


### superscalaire



Superscalaire de degré n :  
n instructions par cycle

## Equation de performance mémoire



$$T_{ex} = NI * (CPI_{cpu} + CPI_{mémoire}) * T_c$$

$$CPI_{mémoire} = ma * m * p$$

ma: accès mémoire/instruction

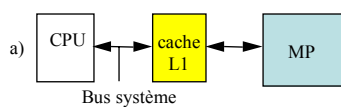
m: taux d'échec

p : pénalité d'échec (cycles d'horloge)

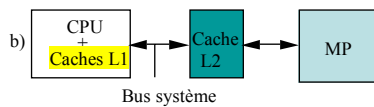
Réduire le taux d'échec  
Réduire la pénalité d'échec

## Hiérarchies de caches : 1985-2000

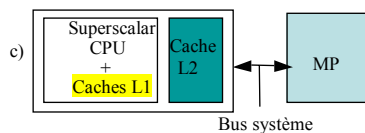
**486**



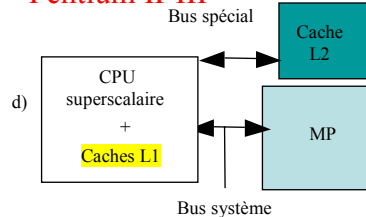
**Pentium**



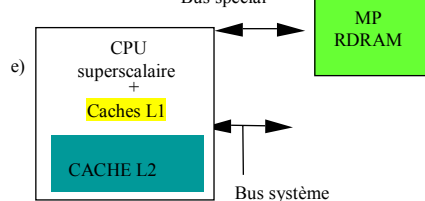
**Pentium Pro**



**Pentium II-III**



**Pentium 4**



## Amélioration des performances cache

- Réduire le taux d'échec
  - Taille de cache, de ligne et degré d'associativité *HW*
  - Optimisations du compilateur *SW*
  - Préchargement *HW SW*
- Réduire le temps de l'accès réussi
  - Prédiction d'ensemble *HW*
  - Eviter les traductions d'adresse *HW*
- Réduire la pénalité d'échec
  - Caches non bloquants *HW*
  - Caches de second niveau *HW*

## Caches données des processeurs Intel

### CACHE PRIMAIRE

	Taille	Assoc.	Lignes	Latence	Ecriture
Pentium Pro	8 Ko	2 voies	32 o	3	Réécriture
Pentium III	16 Ko	4 voies	32 o	3	Réécriture
Pentium 4	8 Ko	4 voies	64 o	2/6	Simultanée

### CACHE SECONDAIRE (UNIFIE)

	Taille	Assoc.	Lignes	Latence	Ecriture
Pentium Pro	512 Ko	4 voies	32 o		Réécriture
Pentium III*	256 Ko	8 voies	32 o	6	Réécriture
Pentium 4	256 Ko	8 voies	128 o	7/7	Réécriture


## L'allocation mémoire des tableaux C

- Les tableaux C sont alloués dans l'ordre ligne d'abord
  - Ligne dans des cases mémoires contigues
- Balayage à travers les colonnes dans une ligne :
  - `for (i = 0; i < N; i++)`  
`sum += a[0][i];`
  - Accède aux éléments successifs
  - Exploite la localité spatiale :
    - Taux d'échecs obligatoires :  $1/B$  (avec  $B$  éléments par bloc de cache)
- Balayage à travers les lignes dans une colonne :
  - `for (i = 0; i < n; i++)`  
`sum += a[i][0];`
  - Accèdent à des éléments distants
  - Aucune localité spatiale !
    - Taux d'échecs obligatoires = 1 (i.e. 100%)

## Optimisations du compilateur

### ECHANGE DE BOUCLES *Améliore la localité spatiale*

```
for (j=0; j<100; j++)  
  for (i=0; i<5000; i++)  
    x[i][j] = 2*x[i][j];
```



```
for (i=0; i<5000; i++)  
  for (j=0; j<100; j++)  
    x[i][j] = 2*x[i][j];
```

### FUSION DE TABLEAUX

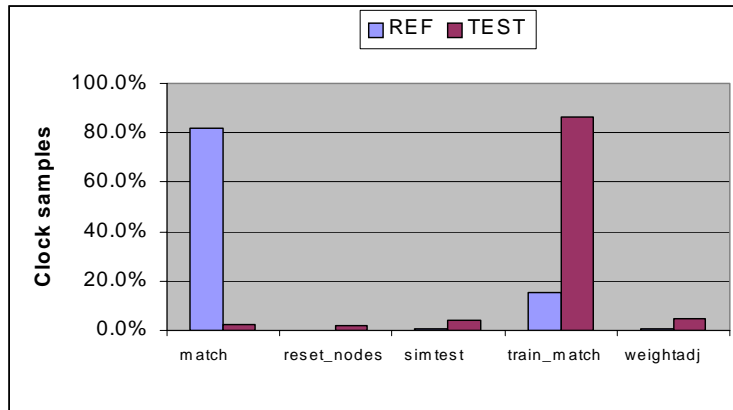
```
int val[SIZE];  
int key[SIZE];
```



```
struct merge {  
    int val;  
    int key; }  
struct merge merge-array [SIZE];
```



## Goulets d'étranglement ART



### Fonctions Match et train match

## Code critique

```
if ( !Y[tj].reset )  
    for (ti=0;ti<numfls;ti++)  
        Y[tj].y += fl_layer[ti].P * bus[ti][tj];
```



*Tableau de structures Pas non unitaire*

- bus[][] et tds[][] interviennent dans différentes fonctions dans le programme
- f\_layer est la structure principale

## IMPACT DE LA STRUCTURE

```
typedef struct {  
    double *I;  
    double W;  
    double X;  
    double V;  
    double U;  
    double P;  
    double Q;  
    double R;  
} fl_neuron;
```



11 (entrée)

+ 7

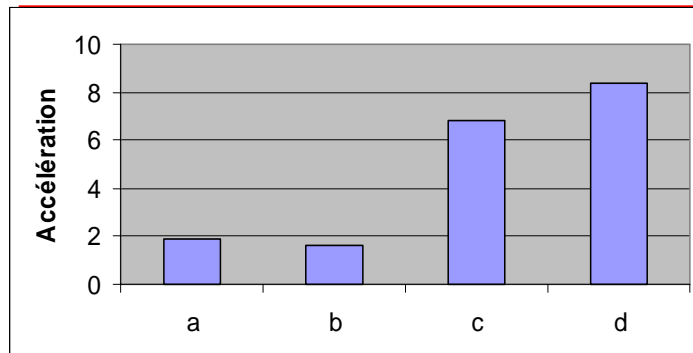
= 18 doubles par  
occurrence de la structure

```
for (ti=0;ti<numfls;ti++)  
    Y[tj].y += fl_layer[ti].P * bus[ti][tj];
```

```
fl_neuron *fl_layer;
```

**1 ECHEC/ITERATION !!!!!**  
**10 000 ITERATIONS**

## Optimisation de ART



- a) Remplacer la structure par des tableaux d'éléments
- b) Changer l'ordre des itérations pour Bus[ ][ ] et tds[ ][ ]
- c) Les deux optimisations a) et b)
- d) Les deux optimisations plus division remplacée par multiplication

## Division/Multiplication

```
tnorm = sqrt((double) tnorm);  
for (tj=0;tj<numfls;tj++)  
    fl_layerU[tj] = fl_layerV[tj] / tnorm;  
for (tj=0;tj<numfls;tj++)  
    fl_layerX[tj] = fl_layerW[tj] / tnorm;
```

```
itnorm = 1/sqrt((double) tnorm);  
for (tj=0;tj<numfls;tj++)  
    fl_layerU[tj] = fl_layerV[tj] * itnorm;  
for (tj=0;tj<numfls;tj++)  
    fl_layerX[tj] = fl_layerW[tj] * itnorm;
```



## Optimisations du compilateur

*Améliore la localité temporelle*

### FUSION DE BOUCLES

```
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
        a[i][j] = 1/b[i][j]*c[i][j];  
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
        d[i][j] = a[i][j]+c[i][j];
```



```
for (i=0; i<N; i++)  
    for (j=0; j<N; j++)  
    {  
        a[i][j] = 1/b[i][j]*c[i][j];  
        d[i][j] = a[i][j]+c[i][j];  
    }
```

## Optimisations du compilateur

### BLOCCAGE

### AMELIORE LA LOCALITE TEMPORELLE

```
for (i=0; i<N; i++)
  for (j=0; j<N; j++)
    {r=0;
     for (k=0; k<N; k++)
       r+=y[i][k]*z[k][j];
     x[i][j]=r;}
```

```
for (jj=0; jj<N; jj+=B)
  for (kk=0; kk<N; kk+=B)
    for (i=0; i<N; i++)
      for (j=jj; j<(min(jj+B-1,N); j++)
        {r=0;
         for (k=kk; k<(min(kk+B-1,N); k++)
           r+=y[i][k]*z[k][j];
         x[i][j]+=r;}
```

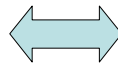
B : facteur de blocage

## Compatibilités entre optimisations

- Pas unitaire est efficace pour les caches et pour la vectorisation
- Mais contre-exemples

```
for (j=0; j<2; j++)
  for (i=0; i<1000; i++)
    x[i][j] = 0.0;
```

Le tableau tient dans le cache  
Réutilisation totale



```
for (i=0; i<1000; i++)
  for (j=0; j<2; j++)
    x[i][j] = 0.0;
```

Boucle interne trop courte

```
for (j=1; j<m; j++)
  for (i=0; i<n; i++)
    x[i][j] = x[i][j] + x[i][j-1];
```

```
for (i=0; i<n; i++)
  for (j=1; j<m; j++)
    x[i][j] = x[i][j] + x[i][j-1];
```

Bon comportement du cache, mais récursion sur j

## Préchargement logiciel

- Charger à l'avance les lignes de cache avant les accès mémoire qui les demanderont
- Questions
  - Où charger : quel niveau de cache ?
  - Quand ?
- Les instructions disponibles
  - Instructions de préchargement (IA-32)
    - PREFETCHT0 m8 : tous les niveaux de cache
    - PREFETCHT1 m8 : tous les niveaux sauf L0
    - PREFETCHT2 m8 : tous les niveaux sauf L0 et L1
    - PREFETCHNTA m8 : préchargement dans une structure non temporelle
  - Indications de préchargement
    - Dépendent des implémentations

## Préchargement dans les PIII et P4

Type de préchargement

PrefetchNTA	Précharge 32 octets Précharge dans L1	Précharge 128 octets Précharge dans 1 voie de L2
PrefetchT0	Précharge 32 octets Précharge dans L1 et L2	Précharge 128 octets Précharge dans L2
PrefetchT1, PrefetchT2	Précharge 32 octets Précharge dans L2	Précharge 128 octets Précharge dans L2

## Utilisation du préchargement (Exemples)

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

```
for (i=0; i<n; i++) {  
  prefetch b[i+16];  
  a+ = b[i];  
}
```

Préchargement

```
for (i=0; i<n; i++) {  
  if ((i%16) == 0) prefetch b[i+16];  
  a+ = b[i];  
}
```

1 préchargement par ligne de cache,  
Mais surcoût du test IF

```
for (i=0; i<n; i+=2) {  
  prefetch b[i+16];  
  a+ = b[i];  
  a+ = b[i+1];  
}
```

Déroulage supplémentaire pour limiter  
le nombre de préchargements redondants

## Concaténation de préchargements

```
for (i=0; i<n; i++) {  
  for (j=0; j<m; j++) {  
    prefetch a[i][j+8];  
    computation a[i][j];  
  }  
}
```

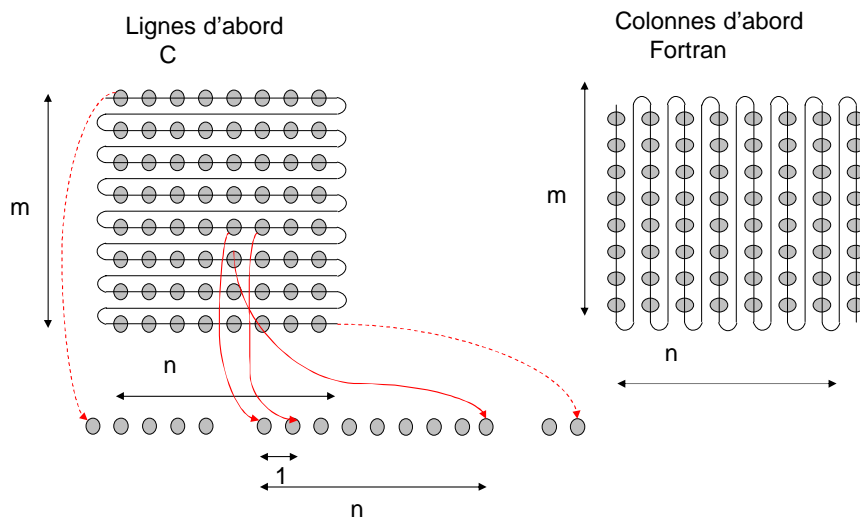
a[i][0] n'est jamais préchargé

```
for (i=0; i<n; i++) {  
  for (j=0; j<m-1; j++) {  
    prefetch a[i][j+8];  
    computation a[i][j];  
  }  
  prefetch a[i+1][0];  
  computation a[i][j];  
}
```

## Accès aux données et caches

- Images
  - Traitement sur un tableau [lignes][colonnes]
  - Un certain nombre de filtres implique un accès horizontal (selon les lignes) et un accès vertical (colonnes)
  - Minimiser les échecs cache
- Accès aux données
  - Placement canonique
  - Placement hiérarchique (4D, Morton)

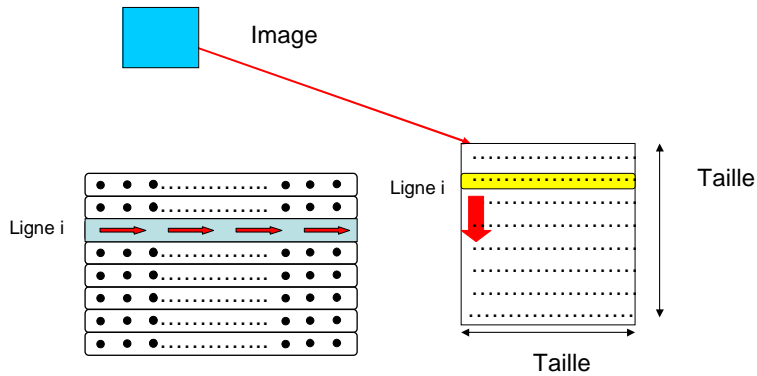
## Placements canoniques



## Ligne d'abord : filtrage horizontal

Ligne d'abord  
Filtrage horizontal

Exploitation de la  
localité spatiale



Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

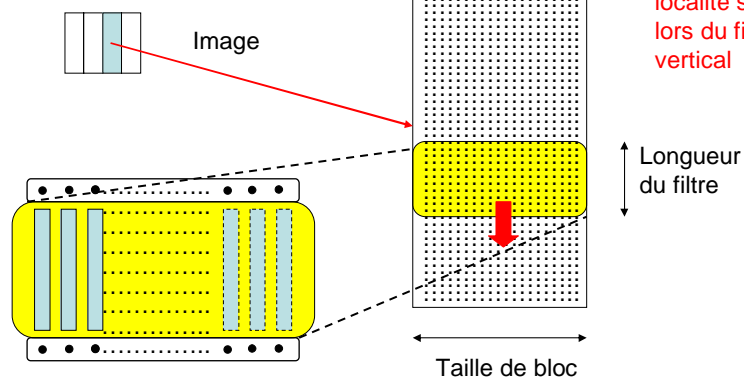
57

## Ligne d'abord : filtrage vertical

Ligne d'abord  
Filtrage vertical

AGGREGATION

Meilleure  
exploitation de la  
localité spatiale  
lors du filtrage  
vertical

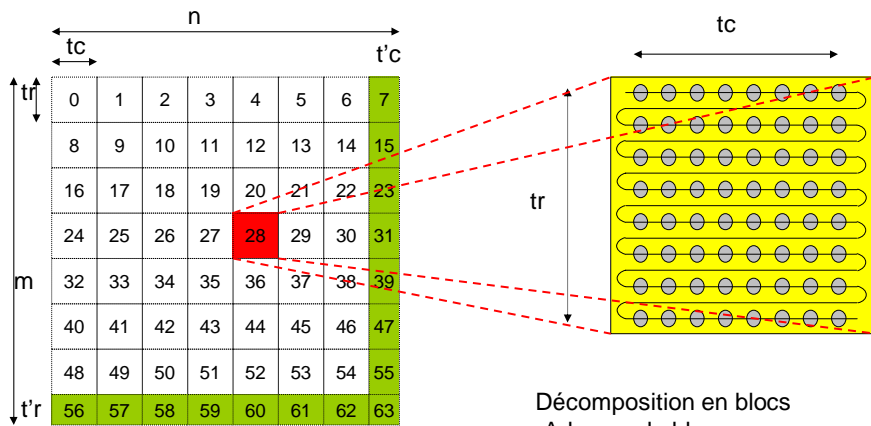


Master Pro  
2005

Optimisations pour graphique et multimédia  
D. Etiemble

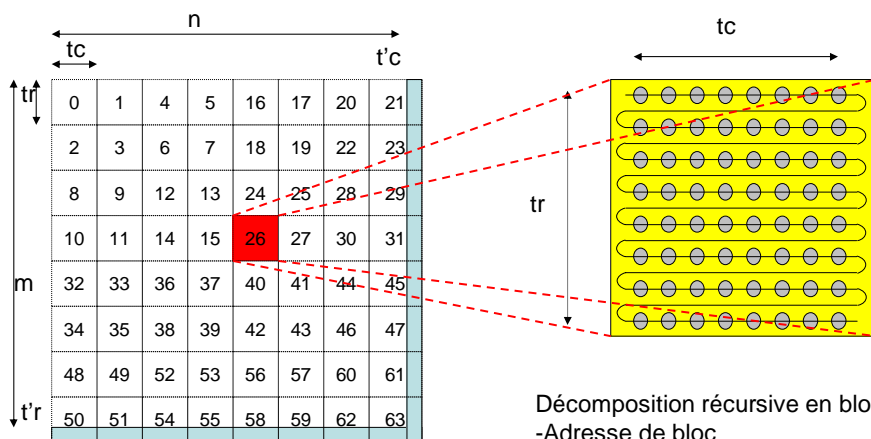
58

## Placement 4D



Décomposition en blocs  
-Adresse de bloc  
-Adresse dans le bloc

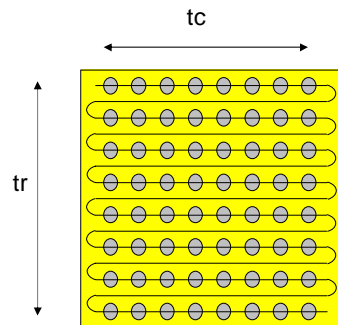
## Placement de Morton



Décomposition récursive en blocs  
-Adresse de bloc  
-Adresse dans le bloc

## Filtrage 1D sur un bloc de $tr \times tc$ éléments

- Filtrage vertical
  - Pour chaque colonne
    - Pour chaque ligne
      - Pour chaque coefficient
        - » Filtre
- Filtrage horizontal
  - Pour chaque ligne
    - Pour chaque colonne
      - Pour chaque coefficient
        - » Filtre
- Filtrage N
  - Pour chaque colonne
    - Pour chaque coefficient
      - Pour chaque ligne
        - » Filtre
- Filtrage Z
  - Pour chaque ligne
    - Pour chaque coefficient
      - Pour chaque colonne
        - » Filtre



## Instructions SIMD

- A suivre