## Plus loin, plus vite. . .

Un exemple : produit matriciel $C = A \times B$

$$C_{i,j} = \sum_{k=1}^{N} A_{i,k} \times B_{k,j}$$

Opération très courante

En C ? naif.c

En langage d'assemblage LC3 ?

Toutes les boucles se valent-elles ? Matériel !



## Plus loin, plus vite. . . Boucles

Avec des flottants (`double`) ⇝ opération longue et pipelinable

Problématiques pipeline :

- Débit
- Latence (et dépendances)

Dans boucle :

- Calcul
- M. à j. indice
- Test de saut
- Saut éventuel (voir fin du cours)



## Plus loin, plus vite. . . Boucles

Directement ⇝ 1 test + 1 saut par calcul

*Déroulage* : plusieurs calculs dans boucle

*Ordre* : nb. de calculs par passage

- Moins d'administration (grâce à ordre. . . )
- Corps plus gros (meilleure gestion de latence. . . )

Exemple en assembleur ARM



## Plus loin, plus vite. . . Boucles

Retour au produit matriciel

`naif.c` ⇝ produit scalaire

Boucle interne :

$$C = g(C, f(k))$$

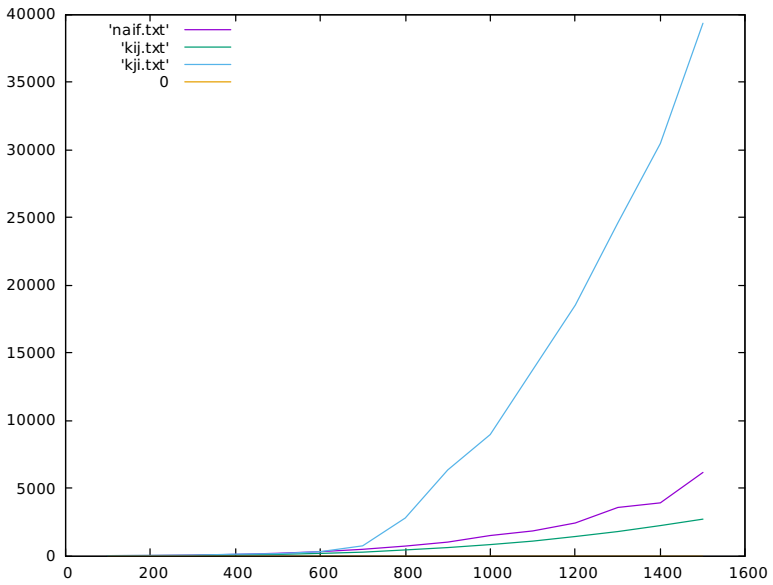
Si $g$ avec latence. . .

Deux façons de changer :

- KIJ $C_{i,j} = C_{i,j} + A_{i,k} \times B_{k,j}$
- KJI $C_{i,j} = C_{i,j} + A_{i,k} \times B_{k,j}$

De la forme AXPY (A fois X Plus Y) ⇝ **D**AXPY ou **S**AXPY

Test et explication






## Plus loin, plus vite. . .



Tout ça : localité spatiale

Localité temporelle ? $\rightsquigarrow$ découpage en blocs

Idée :

$C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$

$C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$

$\vdots$

Idéalement : L2 contient les blocs MAIS du coup dépend du matériel



## Plus généralement



*Nid de boucles* : imbrications

- À pas constants
- Tableaux linéaires
- Arithmétique + affectation

Techniques de réorganisation ?



## Boucles



Échange (déjà vu)

```
for (i=0; i<N; i=i+1)
  for (j=0; j<N; j=j+1)
    B[i][j] = A[j][i];
```

$\rightsquigarrow$

```
for (j=0; j<N; j=j+1)
  for (i=0; i<N; i=i+1)
    B[i][j] = A[j][i];
```

Déroulage (déjà vu)

```
for (i=0; i<N; i=i+1)
    A[i] = A[i-1] + ...;
```

$\rightsquigarrow$

```
for (bi=0; bi<N; bi=bi+S)
  for (j=0; j<bi+S; j=j+1)
    A[j] = A[j-1] + ...;
```

Opération inverse : Coalescing

## Boucles — Réorganisation

Boucles séparées mais même plage d'indice ?

Fusion

```
for (i=0; i<N; i=i+1) {
  for (j=0; j<N; j=j+1)
    B[i] += A[i][j];
  for (j=0; j<N; j=j+1)
    C[i][j] = 0;
}
```

⇝

```
for (i=0; i<N; i=i+1)
  for (j=0; j<N; j=j+1) {
    B[i] += A[i][j];
    C[i][j] = 0;
  }
```

Opération inverse fission

Décalage, inversion...

VALIDITÉ !



## Instructions de contrôle

10% à 25% des inst exécutées

- BR inconditionnels
- RET et JMP routines
- BR conditionnels ~75%



## Instructions de contrôle — BR conditionnels

Ne rien faire... pénalités

Retarder

- Inst. précédente
- Back jump (si annulation)

Prédire

- Automates
- Multicouche historique + automate

Supprimer

- Instructions à prédicats

Exemple ARM



## Instructions de contrôle — BR conditionnels

Supprimer : instructions à prédicats

cf. infocenter.arm.com

```
gcd:  CMP   r0 r1
      BEQ   end
      BLT   less
      SUBS  r0 r0 r1
      B     gcd
less: SUBS  r1 r1 r0
      B     gcd
end:  ...
```

⇝

```
gcd:  CMP   r0 r1
      SUBGT r0 r0 r1
      SUBLT r1 r1 r0
      BNE   gcd
      ...
```

`SUBGT` et `SUBLT` exécutées en fonction du registre de signaux mis à jour par `CMP`

Branchements `BEQ`, `BLT`, `BNE` en fonction registre signaux
(et pas en fonction de `r0`, `r1`)

Exemple exécution 1 2 : 13 cycles contre 10