

Parallélisation automatique de boucles pour processeurs vectoriels

Félix-Antoine Ouellet

Université de Sherbrooke

18 Septembre 2014

- 1 Motivation
- 2 Exemple
- 3 Procédure
- 4 Problèmes ouverts
- 5 Conclusion

Plan

- 1 Motivation
 - Fin de la loi de Moore
 - Processeurs modernes
- 2 Exemple
- 3 Procédure
- 4 Problèmes ouverts
- 5 Conclusion

Loi de Moore

"Le nombre de transistors dans les microprocesseurs double tous les 18 mois."

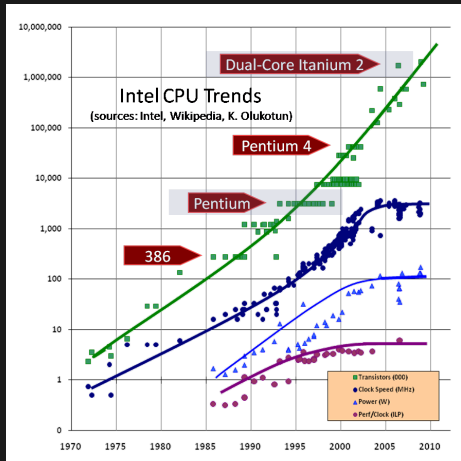
- Loi de Moore

Constat de l'industrie

"The free lunch is over"

- Herb Sutter

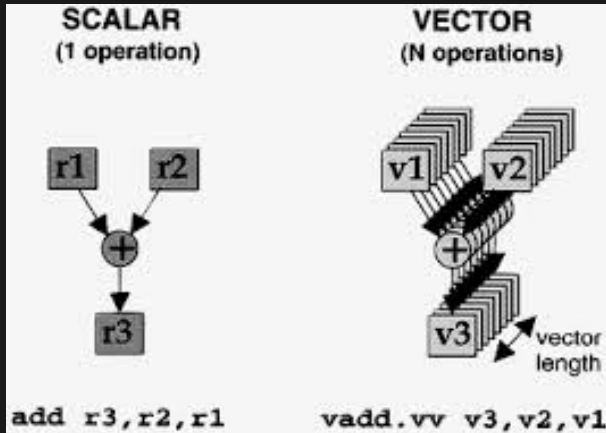
Constat de l'industrie



Avenues possible

- Processeurs multi-coeurs
- Accélérateurs
- Processeurs vectoriels

Processeurs vectoriels



Plan

- 1 Motivation
- 2 **Exemple**
- 3 Procédure
- 4 Problèmes ouverts
- 5 Conclusion

Code Séquentiel

Somme des éléments de vecteurs

```
int somme = 0;
for (int i = 0; i < 100; ++i) {
    somme += A[i];
}
```

Code Vectoriel

Somme des éléments de vecteurs

```
int sommeTab[] = { 0, 0 };
for (int i = 0; i < 100; i+=8) {
    sommeTab[0] += A[i:i+3];
    sommeTab[1] += A[i+4:i+7];
}
int somme = 0;
for (int i = 0; i < 2; ++i) {
    somme += sommeTab[i];
}
for (int i = 96; i < 100; ++i) {
    somme += A[i];
}
```

Plan

- 1 Motivation
- 2 Exemple
- 3 **Procédure**
 - Notions de base
 - Analyse
 - Transformations
- 4 Problèmes ouverts
- 5 Conclusion

Notions de base

Dépendence mémoire

Situation dans laquelle deux instructions accèdent à la même donnée.

Notions de base

Classification des dépendances mémoire

```
void fct() {  
    int a = 20;  
    int b = a;  
    /* ... */  
    int c = d;  
    int d = e;  
}
```

Notions de base

Classification des dépendances mémoire

```
void fct() {  
    int a = 20;  
    int b = a;  
    /* ... */  
    int c = d;  
    int d = e;  
}
```

Vraie dépendance
(Lecture après écriture)

Notions de base

Classification des dépendances mémoire

```
void fct() {  
    int a = 20;  
    int b = a;  
    /* ... */  
    int c = d;  
    int d = e;  
}
```

Vraie dépendance
(Lecture après écriture)

Anti-dépendance
(Écriture après lecture)

Notions de base

Classification des dépendances de boucles

```
for (int i = 0; i < 100; ++i) {  
    A[i] = B[i] + C[i];  
    D[i] = A[i] + 10;  
}
```

```
for (int i = 0; i < 100; ++i) {  
    A[i+1] = A[i] + B[i];  
}
```

Notions de base

Classification des dépendances de boucles

```
for (int i = 0; i < 100; ++i) {  
    A[i] = B[i] + C[i];  
    D[i] = A[i] + 10;  
}
```

Dépendance indépendante
de la boucle

```
for (int i = 0; i < 100; ++i) {  
    A[i+1] = A[i] + B[i];  
}
```

Notions de base

Classification des dépendances de boucles

```
for (int i = 0; i < 100; ++i) {  
    A[i] = B[i] + C[i];  
    D[i] = A[i] + 10;  
}
```

Dépendance indépendante
de la boucle

```
for (int i = 0; i < 100; ++i) {  
    A[i+1] = A[i] + B[i];  
}
```

Dépendance portée par
la boucle

Légalité

Parallélisation des instructions

```
int somme = 0;
for (int i = 0; i < 100;
    ++i) {
    somme += A[i];
}
```

Légalité

Parallélisation des instructions

✓ Pas d'appels de fonctions

```
int somme = 0;
for (int i = 0; i < 100;
    ++i) {
    somme += A[i];
}
```

Légalité

Parallélisation des instructions

```
int somme = 0;  
for (int i = 0; i < 100;  
    ++i) {  
    somme += A[i];  
}
```

- ✓ Pas d'appels de fonctions
- ✓ Opération parallélisable

Légalité

Parallélisation des instructions

```
int somme = 0;
for (int i = 0; i < 100;
    ++i) {
    somme += A[i];
}
```

- ✓ Pas d'appels de fonctions
- ✓ Opération parallélisable
- ✓ Types des paramètres parallélisables

Légalité

Parallélisation des instructions

```
int somme = 0;  
for (int i = 0; i < 100;  
    ++i) {  
    somme += A[i];  
}
```

- ✓ Pas d'appels de fonctions
- ✓ Opération parallélisable
- ✓ Types des paramètres parallélisables
- ✓ Type de retour parallélisable

Légalité

Parallélisation de la mémoire

```
int somme = 0;
for (int i = 0; i < 100; ++i) {
    somme += A[i];
}
```

Légalité

Parallélisation de la mémoire

```
int somme = 0;  
for (int i = 0; i < 100; ++i) {  
    somme += A[i];  
}
```

✓ Pas de chevauchement
d'accès mémoire

Légalité

Parallélisation de la mémoire

```
int somme = 0;  
for (int i = 0; i < 100; ++i) {  
    somme += A[i];  
}
```

- ✓ Pas de chevauchement d'accès mémoire
- × Pas de dépendences mémoire

Profitabilité

- Lié à l'architecture physique
- Coût version séquentielle VS Coût version vectorielle

Séquentielle	Vectorielle
Coût add i64	Coût add $\langle 2 \times i64 \rangle$
Coût load i64	Coût load $\langle 2 \times i64 \rangle$

- Meilleur facteur de déroulement

Reconnaissance d'idiomes

Théorie

- But: Agir en présence d'une situation connue
- Exemples:
 - Induction
 - Réduction

Reconnaissance d'idiomes

Pratique

```
int somme = 0;
int sommeTab[2];
for (int i = 0; i < 2; ++i) {
    sommeTab[i] = 0;
    for (int j = i; j < 100; j+=2) {
        sommeTab[i] += A[j];
    }
    somme += sommeTab[i];
}
```

Distribution de boucle

Théorie

- But: Regrouper les calculs similaires
- Moyen: Produire plusieurs boucles à partir de la boucle originale

Distribution de boucle

Pratique

```
int sommeTab[] = { 0, 0 };  
for (int i = 0; i < 2; ++i) {  
    for (int j = i; j < 100; j+=2) {  
        sommeTab[i] += A[j];  
    }  
}  
  
int somme = 0;  
for (int i = 0; i < 2; ++i) {  
    somme += sommeTab[i];  
}
```


Inter-échange de boucles

Théorie

- But: Optimiser les accès mémoire et exposer du parallélisme
- Moyen : Échanger les variables d'induction des boucles ciblées

Inter-échange de boucles

Pratique

```
int sommeTab[] = { 0, 0 };  
for (int j = 0; j < 100; j+=2) {  
    for (int i = j; i < min(j+2, 100); ++i) {  
        sommeTab[i-j] += A[j+i];  
    }  
}  
  
int somme = 0;  
for (int i = 0; i < 2; ++i) {  
    somme += sommeTab[i];  
}
```

Vectorization

Théorie

- But: Exploiter les registres et opérations vectoriels disponibles
- Moyen : Générer du code vectoriel

Vectorization

Pratique

```
int sommeTab[] = { 0, 0 };  
for (int j = 0; j < 100; j+=4) {  
    for (int i = j; i < min(j+2, 100); ++i) {  
        sommeTab[i-j] += A[j:j+3];  
    }  
}  
  
int somme = 0;  
for (int i = 0; i < 2; ++i) {  
    somme += sommeTab[i];  
}
```

Déroutage de boucle

Théorie

- But: Réduire le temps d'exécution d'une boucle
- Moyen : Expliciter les calculs dans une boucle
- Attention, on choisit de prendre plus de mémoire pour gagner en vitesse d'exécution

Déroutage de boucle

Pratique

```
int sommeTab[] = { 0, 0 };
for (int i = 0; i < 100; i+=8) {
    sommeTab[0] += A[i:i+3];
    sommeTab[1] += A[i+4:i+7];
}
int somme = 0;
for (int i = 0; i < 2; ++i) {
    somme += sommeTab[i];
}
for (int i = 96; i < 100; ++i) {
    somme += A[i];
}
```

Plan

- 1 Motivation
- 2 Exemple
- 3 Procédure
- 4 Problèmes ouverts**
- 5 Conclusion

Pointeurs

```
void bar(float *A, float *B, float K, int n) {  
    for (int i = 0; i < n; ++i)  
        A[i] *= B[i] + K;  
}
```


Superword Level Parallelism

```
void foo(int a1, int a2,  
         int b1, int b2, int *A) {  
    A[0] = a1*(a1 + b1)/b1 + 50*b1/a1;  
    A[1] = a2*(a2 + b2)/b2 + 50*b2/a2;  
}
```

Plan

- 1 Motivation
- 2 Exemple
- 3 Procédure
- 4 Problèmes ouverts
- 5 Conclusion

Conclusion

- Trois grandes étapes

Conclusion

- Trois grandes étapes
- L'autovectorization c'est bien, mais c'est limité

Conclusion

- Trois grandes étapes
- L'autovectorization c'est bien, mais c'est limité
- L'architecture change donc la programmation doit changer