Optimisations applicatives et transformations algorithmiques

Lionel Lacassagne

LIP6

Sorbonne University (UPMC) / LIP6 / ALSOC

https://www.lip6.fr/

lionel.lacassagne@lip6.fr

Les domaines applicatifs

pour faire simple

- le HPC classique (*High Performance Computing*)
  - algèbre linéaire → opérations ponctuelles avec réduction (produit scalaire)
  - équations aux dérivées partielles → éléments finis, différences finies → *stencil* + dépendance de données
- Les systèmes embarqués
  - traitement du signal: filtrage non récursif (convolution), filtrage récursif (dépendance de données)
  - traitement des images: traitement du signal en 2D + autres algorithmes
- Points communs
  - 2 types d'opérateurs : ponctuels ou avec voisinage (stencil = convolution)
  - avec/sans dépendance de données
  - en 1D, 2D ou 3D (généralement, mais 4D et + existent)
- Problèmes communs
  - optimisation pour la persistance des données en cache
  - transformation des boucles et du *memory layout*
- Problèmes supplémentaire pour les systèmes embarqués
  - en HPC, si pas assez de puissance, achat de noeuds de calcul (augmentation de la consommation)
  - en SE, si pas assez de puissance, plus d'optimisation et compromis vitesse/qualité

Comment aller plus vite ?

- Optimiser la durée des boucles
  - car c'est là que se concentre la majorité des calculs
  - $\Rightarrow$ identifier les boucles consommant le plus de temps
  - alors comment aller plus vite ?

- Réduire la durée des calculs
  - en enchainant mieux les calculs (archi & pipeline)
  - en diminuant leur nombre (algo & factorisation)
  - en les remplaçant par des calculs équivalents plus rapides (algo & maths)

- Réduire la durée des accès mémoire
  - en enchainant mieux les accès mémoire (archi & mémoire cache)
  - en diminuant leur nombre (algo)
  - c'est aujourd'hui *la* priorité

- Les architectures actuelles
  - ont une énorme puissance de calcul (SIMD $\times$ multicoeurs)
  - qui n'est utilisable que si les données sont proches des unités de calcul

Les métriques d'analyse et de performance

- Les Giga
  - GFlops/s: milliards d'opérations (flottante) par seconde
  - GB/s: milliards d'octets transférés par seconde
- Les métriques du microprocesseur
  - ipc: instruction par cycle: le débit moyen d'instruction
  - cpi: cycle par instruction, quand ça va mal ($cpi = 1/ipc$) et que le code est lent
- La métrique normalisée
  - cpp: cycle par point : temps normalisé pour produire un point
- Les métriques embarquées
  - MFlops/Watt: la puissance de calcul efficace
  - nJoules/point: l'énergie normalisée
- L'intensité arithmétique
  - caractérise un algorithme: nb de calculs / nb d'accès mémoire
  - par extension le ratio *compute/bandwidth* pour caractériser un processeur

L'intensité arithmétique (IA) #1

- A l'origine
  - popularisée par Nvidia pour guider le développeur
- Lien entre IA et performance (atteindre la performance crête)
  - avoir une IA élevée
  - avoir du ré-use afin d'avoir des accès mémoire rapides
- Exemple #1: addition de deux tableaux
  - $y(n) = x_1(n) + x_2(n)$
  - $IA = 1/3 = 0.3$
  - in-accélérable (ira vite car simple, mais aucune implémentation efficace)
- Exemple #2: somme de 3 points (stencil)
  - $y(n) = x(n-1) + x(n) + x(n+1)$
  - $IA = 2/4 = 0.5$ accélérable car ré-use
- Exemple #3: produit scalaire (réduction)
  - $dp = \Sigma x(i) \cdot y(i)$
  - $IA = 1/2 = 0.5$ in-accélérable car pas de ré-use
- Exemple #4: produit matriciel
  - $C(i,j) = C(i,j) + A(i,j) \times B(k,j)$
  - n^2 produits scalaires
  - $IA = 2/4 = 0.5$ très accélérable, car ré-use

Stencils et convolutions

définitions, notation et vocabulaire

- *stencil* = pochoir = traitement sur un petit voisinage d'éléments
- convolution: le fait d'appliquer un filtre à un signal
  - Soit x un signal en entrée, h un filtre de m coefficients et y le signal de sortie, alors $\boxed{\text{y = x * h}}$ (le symbole de la convolution est l'étoile)
  - Pour tout point n du signal

$$\boxed{y(n) = \sum_{k=0}^{k=m-1} h(k)x(n-k)}$$

  - Soit: $y(n) = h_0x(n) + h_1x(n-1) + \cdots + h_{m-1}x(n-m+1)$
  - Ce calcul est répété pour les n points de la sortie y
  - Filtrer un signal, convoluer un signal par un filtre, ce n'est que réaliser une somme pondérée, calculer le produit scalaire des coefficients du filtre par les points du signal.

Stencils et convolutions

définitions, notation et vocabulaire

- Traitement du Signal (TS)
  - filtre causal = ne dépend que du présent et du passé = filtre à gauche
  - exemple si $h = [3, 2, 1]$ alors $y(n) = 3x(n) + 2x(n-1) + x(n-2)$

- Traitement des Images (TI)
  - passé et futur ont moins de sens qu'en TI
  - filtre non causal, par exemple filtre centré
  - exemple si $h = [3, 2, 1]$ alors $y(n) = 3x(n+1) + 2x(n) + x(n-1)$
  - en général, les filtres centrés sont symétriques et on oublie de "retourner" les coefficients lorsque le filtre n'est plus symétrique (ce qui n'a pas d'impact sur les autres traitements)
  - exemple $h = [1, 2, 1]/4$ soit $y(n) = \frac{1}{4}[x(n-1) + 2x(n) + x(n+1)]$
  - on peut aussi avoir une écriture plus compacte: $y_n = (x_{n-1} + 2x_n + x_{n+1})/4$

- Notations
  - TSI et algo: parenthèse ou indice soit $x(n)$ ou x_n
  - codage et implémentation en C: crochet, minuscule pour variable, majuscule pour tableau soit `x=X[i]`

Convolutions et gestion des bords

- Soit filtre à gauche $h = [3, 2, 1]/6 \Rightarrow y(n) = [3x(n) + 2x(n-1) + x(n-2)]/6$
- En convoluant x par h on obtient:
  - $(3 \times 6 + 2 \times 5 + 1 \times 4)/6 = 32/6 \simeq 5.3$
  - $(3 \times 7 + 2 \times 6 + 1 \times 5)/6 = 38/6 \simeq 6.3$

n	0	1	2	3
$x(n)$	4	5	6	7
$y(n)$	?	?	5.3	6.3

  - il manque les premières sorties: $y(0)$ et $y(1)$

- Plusieurs stratégies possibles :
  - recopie des premières valeurs de l'entrée (*pré-duplication*)

n	0	1	2	3
$x(n)$	4	5	6	7
$y(n)$	4	5	5.3	6.3

  - duplication de *la* première valeur de sortie (*post-duplication*)

n	0	1	2	3
$x(n)$	4	5	6	7
$y(n)$	5.3	5.3	5.3	6.3

Convolutions et gestion des bords

- Une autre stratégie: *pré-duplication avec bords*
  - ajout d'un bord au signal et recopie de la première valeur dans le bord
  - puis application du filtre: le premier point calculable est en 0
  - $(3 \times 4 + 2 \times 4 + 1 \times 4)/6 = 24/6 = 4.0$
  - $(3 \times 5 + 2 \times 4 + 1 \times 4)/6 = 27/6 \simeq 4.7$

n	-2	-1	0	1	2	3
$x(n)$	4	4	4	5	6	7
$y(n)$			4	4.7	5.3	6.3

- Avantage: les valeurs de sortie sont progressives et non constantes, contrairement aux cas précédents
- Une dernière stratégie: la rotation de registres ...

Filtre non récursif (aka *stencil* ou convolution)

Rotation de Registres

- Le plus simple des filtres et des stencils: la somme de 3 points
  - $y(n) = x(n) + x(n-1) + x(n-2)$
- Implémentation naive
  - `for(int i=0;i<n;i++) Y[i]=X[i]+X[i-1]+X[i-2];`
  - $IA = 2/4 = 0.5$
  - le cas des bords n'est pas traité (on suppose qu'il n'y a pas de problème)
- Rotation de Registres (de variables en fait)
  - notation: `xk` contient $x(n-k)$ avec n le point courant

```
x1 = x2 = X[0]; // une solution au probleme des bords
for(i=0;i<n;i++) {
  x0   = X[i];           // LOAD:1
  y    = x0 + x1 + x2;   // CALC:2
  Y[i] = y;              // STORE:1
  x2   = x1; x1 = x0;    // RR
}
```

  - $IA = 2/(1+1) = 1$ en progression
  - les accès mémoire ont été remplacés par des copies entre variables (*move's*)

Filtre non récursif

Déroulage de boucle

- Déroulage mais de quel ordre ?
  - coloriage des éléments : une élément, une couleur (toujours la même)

$$y(n+0) = x(n+0) + x(n-1) + x(n-2)$$
$$y(n+1) = x(n+1) + x(n+0) + x(n-1)$$
$$y(n+2) = x(n+2) + x(n+1) + x(n+0)$$

- Comment colorier les nouveaux éléments ?
  - les indices sont équivalents, modulo l'ordre de déroulage
  - $\Rightarrow (n-2) \equiv (n+1)$ et $(n-1) \equiv (n+2) \mod 3$
  - il faut retomber sur la configuration initiale $\Rightarrow$ écrire les autres itérations

$$y(n+0) = x(n+0) + x(n-1) + x(n-2)$$
$$y(n+1) = x(n+1) + x(n+0) + x(n-1)$$
$$y(n+2) = x(n+2) + x(n+1) + x(n+0)$$
$$y(n+3) = x(n+3) + x(n+2) + x(n+1)$$

Filtre non récursif

Déroulage de boucle

$$y(n+0) = x(n+0) + x(n-1) + x(n-2)$$
$$y(n+1) = x(n+1) + x(n+0) + x(n-1)$$
$$y(n+2) = x(n+2) + x(n+1) + x(n+0)$$

- Reste à scalariser, écrire prologue et épilogue
  - posons `xk` $\equiv x(n-k) \mod 3$, point distant de k du point courant

```
x1=X[0]; x2=X[0]; // prologue
for(i=0; i<n-r; i+=3) {
  x0=X[i+0]; Y[i+0] = x0+x1+x2;
  x2=X[i+1]; Y[i+1] = x2+x0+x1;
  x1=X[i+2]; Y[i+2] = x1+x2+x0;
}// epilogue restant a ecrire
```

- Remarques
  - au prochain tour de boucle `x1` joue bien de nouveau le rôle de $x(n-1)$ et `x2` le rôle de $x(n-2)$, n ayant été augmenté de 3
  - il est possible d'écrire 3 fois la même somme, cela est moins "lisible" ne cela montrerait pas les intentions de l'auteur du code: il est important de les comprendre

Filtre non récursif (aka *stencil* ou convolution)

Rotation de Registres et Déroulage de boucle

- Filtre centré, $y(n) = \frac{1}{4}[x(n-1) + 2x(n) + x(n+1)]$
  - toujours noté $\frac{1}{4}[1, 2, 1]$
- Implémentation naïve
  - `for(int i=0;i<n;i++) Y[i]=(X[i-1]+2*X[i]+X[i+1])/4;`
  - $IA = 4/4 = 1.0$
- A faire pour s'entrainer

Filtre récursif 1

Rotation de Registres

- Le plus simple des filtres récursifs: moyenne récursive
  - $y(n) = \frac{1}{2}[x(n) + y(n-1)]$
- Implémentation naïve
  - `for(int i=0;i<n;i++) Y[i]=(X[i]+Y[i-1])/2;`
  - $IA = 2/3 = 0.7$
  - le cas des bords n'est pas traité (on suppose qu'il n'y a pas de problème)
- Rotation de Registres (de variables en fait)
  - notation: `xk` contient $x(n-k)$, `yk` contient $y(n-k)$ avec n le point courant

```
y1 = X[0]; // une solution au probleme des bords
for(i=0;i<n;i++) {
  x0   = X[i];          // LOAD:1
  y0   = (x0 + y1)/2;   // CALC:2
  Y[i] = y0;            // STORE:1
  y1   = y0;            // RR
}
```

  - $IA = 2/(1+1) = 1$ en progression
  - les accès mémoire ont été remplacés par une copie entre variable (*move*)
  - le vrai problème est la latence générée par la dépendance (invisible ici)
  - le code du filtre récursif est plus lent que celui du filtre non récursif ...

Filtre récursif 1

Déroulage de boucle

$$y(n+0) = [x(n+0) + y(n-1)]/2$$
$$y(n+1) = [x(n+1) + y(n+0)]/2$$
$$y(n+2) = [x(n+2) + y(n+1)]/2$$

- Ordre de déroulage ?
  - facile car que 2 cas: n et $n-1 \Rightarrow$ déroulage d'ordre 2
  - posons $x_k \equiv x(n+1)$, et $y_k \equiv y(n-k) \mod 2$

$$y_0 = [x_0 + y_1]/2$$
$$y_1 = [x_1 + y_0]/2$$

```
y1 = X[0]; // une solution au probleme des bords
for(i=0;i<n-r;i+=2) {
  x0 = X[i+0]; x1 = X[i+1]; // LOAD:2
  y0 = (x0 + y1)/2;         // CALC:2
  y1 = (x1 + y0)/2;         // CALC:2
  Y[i+0] = y0; Y[i+1] = y1; // STORE:2
}
```

- Analyse
  - $IA = 4/4 = 1$ pareil que RR, mais sans les copies

Filtre récursif 2

- L'un des plus utilisés
  - $y(n) = b_0 x(n) + a_1 y(n-1) + a_2 y(n-2)$
- Implémentation naïve
  - `for(int i=0;i<n;i++) Y[i]=b0*X[i]+a1*Y[i-1]+a2*Y[i-2];`
  - $IA = 5/4 = 1.25$
- Rotation de Registres (de variables en fait)
  - notation: `xk` contient $x(n-k)$, `yk` contient $y(n-k)$ avec n le point courant

```
y2 = y1 = X[0]; // une solution au probleme des bords
for(i=0;i<n;i++) {
  x0    = X[i];                        // LOAD:1
  y0    = b0*x0 + a1*y1 + a2*y2;       // CALC:5
  Y[i]  = y0;                          // STORE:1
  y2    = y1;                          // RR
  y1    = y0;                          // RR
}
```

  - $IA = 5/2 = 1.5$ plus le filtre est complexe et plus l'impact des optimisations augmente
  - le vrai problème est la latence générée par la dépendance (invisible ici)

Filtre récursif 2

Déroulage de boucle

A FAIRE