LES ALGORITHMES LIGNES EN TRAITEMENT D'IMAGES

J.L. Basille, P. Fernandez, J.Y. Latil

Laboratoire CERFIA, IUT Département Informatique
50 A, Chemin des Maraîchers, 31077 TOULOUSE CEDEX

RESUME

La parallélisation SIMD des algorithmes de Traitement d'Images opérant au niveau du pixel est particulièrement aisée à effectuer sur un Processeur Ligne. Elle permet d'obtenir des performances qui, comparées aux performances sur Processor Arrays bi-dimensionnels, sont, proportionnellement au matériel mis en oeuvre, meilleures dans le cas des algorithmes indépendants du balayage, ou bien même, dans le cas des algorithmes de propagation, meilleures en temps absolus. Nous développons actuellement un tel Processeur Ligne en collaboration avec le DEIN du CEA.

SUMMARY

Pixel-level Image Processing algorithms may be easily processed in parallel on a Line Processor. When comparing performances obtained with such a structure to performances obtained with Processor Arrays, the Line Processor structure gives better execution efficiency in proportion to the number of Processing Elements involved for scanning independant algorithms and better absolute execution times for propagation algorithms. We are currently developing such a Line Processor in collaboration with the DEIN/CEA.

I. INTRODUCTION

Les algorithmes de traitement d'images peuvent être regroupés en trois grandes catégories. Nous trouvons tout d'abord les algorithmes qualifiés de globaux et qui comprennent en particulier toutes les tranformations orthogonales. La seconde catégorie correspond aux algorithmes qui traitent séparément des régions de l'image et dont la parallélisation peut conduire à des structures de type MIMD. Nous trouvons enfin dans la dernière catégorie les algorithmes qui opérent au niveau du pixel.

C'est à ces algorithmes et à leur parallélisation SIMD de type Processor Array que nous nous intéressons ici. Nous pouvons distinguer les algorithmes indépendants du balayage et ceux qui sont liés au balayage. Nous allons illustrer ces deux types d'algorithmes avec des exemples pour montrer la facilité de mise en oeuvre de ceux-ci sur une structure Processeur Ligne. Nous pourrons alors constater l'intérêt de cette structure en ce qui concerne les performances obtenues et comparer celles-ci avec des structures plus performantes mais également plus complexes, en tenant compte des rendements respectifs. Nous indiquerons les performances obtenues en simulation avec le projet SYMPATI [BASI85] pour lequel nous intégrons actuellement un Processeur Elémentaire en collaboration avec le DEIN du CEA.

II. ALGORITHMES INDEPENDANTS DU BALAYAGE

Parmi les algorithmes opérant au niveau du pixel , ceux qui sont indépendants de tout balayage constituent la catégorie la plus nombreuse. Nous trouvons en particulier la plupart des algorithmes de filtrage , à l'exception de certains filtres récursifs comme nous le verrons au paragraphe suivant. Cette double caractéristique d'indépendance du balayage et de traitement local sur un certain voisinage de chaque pixel autorise une parallélisation sur une architecture de type Processor Array. Une telle parallélisation peut être considérée comme étant maximale. Mais elle requiert des systèmes à la fois complexes et coûteux [BATC80, HUNT81, DUFF82].

Au lieu d'etre appliqué directement dans son principe bi-dimensionnel , le concept de Processor Array peut être considéré dans une optique mono-dimensionnelle. Nous obtenons alors une stucture de Processeur Ligne dont nous montrons ici l'intérêt.

Dans le cas des algorithmes indépendants du balayage , la mise en oeuvre sur un Processeur Ligne est très proche de celle correspondant à un Processor Array bi-dimensionnel classique. Il est d'ailleurs facilement envisageable et souhaitable de pouvoir utiliser un Processeur Ligne en émulateur de Processor Array bi-dimensionnel.

Pour illuster la facilité de cette mise en oeuvre , nous pouvons prendre l'exemple typique de l'algorithme de SOBEL qui correspond bien à la catégorie d'algorithmes indépendants du balayage.

Mise en oeuvre

Nous allons montrer à partir de cet exemple comment programmer un tel algorithme sur un Processeur Ligne. Rappelons son traitement.

```
POUR tous les pixels aij de l'image FAIRE

 Gx  <--- (V3 + 2V2 + V1) - (V5 + 2V6 + V7)
 Gy  <--- (V3 + 2V4 + V5) - (V1 + 2V0 + V7)
 Aij <---   Gx   +    Gy

FIN
```

```
V3   V2   V1
V4   aij  V0
V5   V6   V7
```

voisinage du point . codage de Freeman .

La mise en oeuvre de cet algorithme est présentée ci-dessous. Elle correspond à l'utilisation d'un Processeur Ligne dont la structure des Processeurs Elémentaires comprend essentiellement un ensemble de registres Ri et une Unité Arithmétique et Logique. Un jeu d'indicateurs permet l'utilisation en mode SIMD mais n'est pas utilisé dans l'exemple présent. Les chemins de données fournis par les interconnexions avec les Processeurs Elémentaires voisins permettent d'accéder directement aux points de la fenêtre 3X3 , et , dans le cas de voisinages de taille quelconque , ils permettent d'accélérer les accès.

Si nous décrivons plus précisément pour un point de l'image le traitement séquentiel , nous obtenons :

```
<< calcul de V3 + 2V2 + V1 >>

 - lecture  du point V3 et stockage dans R0.
              ( 1 cycle )
 - lecture  du point V2 et stockage dans R1.
              ( 1 cycle )
 -  multiplication de R1 par 2.
              ( 1 cycle )
 -  R1 <--- R1 + R0
              ( 1 cycle )
 - lecture du point V1 et stockage dans R2.
              ( 1  cycle )
 -  R1 <--- R1 + R2
              ( 1 cycle )

 % soit au total 6 cycles %


 % R2 est utilisé ci-dessous dans le calcul%
 % de V1 + 2V0 + V7                        %

 << calcul de V5 + 2V6 + V7 >>

       ..........
      ( 6 cycles )

<< production de   Gx    >>

       ..........
      ( 3 cycles )

 << calcul de V3 + 2V4 + V5  >>

       ..........
      ( 4 cycles )
 << calcul de V1 + 2V0 + V7  >>

       ..........
      ( 4 cycles )

 << production de   Gy     >>

       ..........
      ( 3 cycles )

 << résultat final et écriture mémoire >>

       ..........
      ( 2 cycles )
```

Remarque :

L'écriture du point considéré ne peut pas se faire sur l'image de départ puisque le traitement est sans propagation. Nous devons donc avoir deux images : une image de départ et une image d'arrivée.

Le parallélisme est totalement transparent pour l'utilisateur qui programme l'algorithme comme sur une machine séquentielle sans avoir besoin de gérer les boucles d'itération pour balayer la totalité de l'image. La simulation d'autres algorithmes fournit les performances indiquées dans le tableau de la figure 1. Le processeur Ligne développé actuellement possède un temps de cycle de 100 ns et 32 Processeurs Elémentaires.

	# cycles par pixel	temps total image 512X512
binarisation	4	3.2 ms
max fen 3X3	13	10.4 ms
SOBEL	28	22.4 ms
TUKEY	106	85 ms
Pseudo-TUKEY	18	14.7 ms
Convolution	30	24.6 ms

Figure 1

Exemples d'algorithmes sans propagation

III. ALGORITHMES DE PROPAGATION

Les algorithmes liés au balayage sont en général des algorithmes de propagation. Cela signifie que dans le calcul de la valeur f(x,V(x)) d'un pixel x pour un certain voisinage V(x) , il faut partitionner ce voisinage en un sous-ensemble Vi(x) de voisins dont la valeur considérée est la valeur initiale , et un sous-ensemble Vr(x) de voisins dont la valeur considérée est la nouvelle valeur , résultat du calcul effectué précédemment sur ces pixels.

C'est ce type de propagation que nous allons considérer sur la structure de Processeur Ligne en adaptant l'écriture séquentielle classique. La propagation utilisée sur un Processor Array est d'un type différent puisque tous les pixels sont traités simultanément. Le sous-ensemble Vr(x) est donc vide et V(x) Vi(x). Ce type de propagation peut conduire à des rendements très défavorables comme nous le verrons plus loin.

Le premier algorithme que nous pouvons citer pour illustrer ces algorithmes est le calcul de l'Ordre des points d'une image, c'est à dire la distance de chaque pixel à la frontière de la forme à laquelle il appartient.

Cet algorithme [BASI86] nécessite 2 balayages, ascendant et descendant. Au cours de chaque balayage, chaque pixel est comparé avec ses 3 voisins de la ligne précédente, dans le sens du balayage, et ses 2 voisins gauche et droit.

Le deuxième exemple que nous citerons est un algorithme de Filtrage récursif [SHEN86] dont les résultats sont particulièrement satisfaisants. Ce filtrage qui correspond à l'utilisation d'une fenêtre de taille infinie, est très facile à mettre en oeuvre sur un Processeur Ligne. Il suffit

d'effectuer un balayage aller-retour horizontal et un balayage aller-retour vertical. Au cours de chaque balayage, la valeur f(i,j) de chaque pixel est remplacée, dans le cas horizontal gauche-droite par exemple, par :

$$g(i,j) = (1-a).g(i-1,j) + a.f(i,j)$$

où a est une constante dont la valeur optimale est de l'ordre de 0.65.

Ce traitement demande 28 cycles par pixel soit 22.4 ms pour une image 512x512 sur un Processeur Ligne de 32 Processeurs Elémentaires.

Nous terminerons avec, pour troisième exemple, l'algorithme de Labellisation. Quelle que soit la méthode employée pour labelliser les formes connexes d'une image, il est nécessaire de procéder à une propagation.

Le temps total d'exécution dépendra à la fois de la longueur du plus long chemin contenu dans la forme et de la "topologie" de celle-ci. Le cas le plus défavorable est celui d'une spirale telle que celle de la figure 2. Il est nécessaire avec cet exemple d'effectuer 10 balayages, 6 pour la labellisation proprement dite et 4 pour détecter la fin de la propagation.

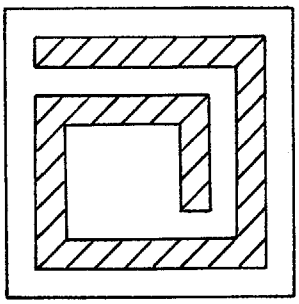

Figure 2

Spirale de longueur 2000 nécessitant 10 balayages pour la labellisation

Les temps d'exécution de ces algorithmes sont présentés dans le tableau de la figure 3. Ils montrent, pour un Processeur Ligne de 32 Processeurs Elémentaires seulement, des performances qui sont très satisfaisantes dans l'absolu et que nous allons comparer dans le paragraphe suivant avec les performances obtenues sur des Processor Arrays bi-dimensionnels.

	# cycles par pixel	temps total image 512x512
Ordre des points	12 x 2	19.6 ms
Filtre récursif	28	22.4 ms
Labellisation	6 pour chaque balayage	4.9 ms

Figure 3

Exemples d'algorithmes de propagation.

IV. CRITERES DE COMPARAISON

Si nous voulons comparer les performances ci-dessus avec des résultats obtenus sur d'autres architectures , il est nécessaire de prendre en compte un critère permettant de confronter des structures entre elles, de façon indépendante de leur mise en oeuvre et en particulier du nombre de Processeurs Elémentaires utilisés.

Nous pourrons alors mieux apprécier l'apport de chaque architecture en fonction des types d'algorithmes considérés. Le critère souvent utilisé est le facteur d'accélération , ou speed-up , défini comme le rapport du temps d'exécution sur mono-processeur par le temps d'exécution sur la structure multiprocesseur concernée :

$$S = \text{temps mono} / \text{temps multi}$$

Mais ce critère ne tient pas compte du nombre de processeurs mis en oeuvre sur la structure considérée. Il est donc préférable de définir un speed-up normalisé $\overline{S}$ défini par :

$$\overline{S} = S / \text{nombre de processeurs}$$

Ce critère nécessairement inférieur à 1, mesure donc le taux de parallélisme et il est d'autant plus proche de 1 que le parallélisme mis en oeuvre est important. Il faut toutefois être prudent dans l'utilisation de ce critère dont la mesure peut conduire à des valeurs supérieures à 1 à cause des calculs d'adresse nécessaires sur monoprocesseur mais inutiles sur un Processor Array par exemple. Aussi semble-t-il préférable de prendre tout simplement les temps d'exécution mais de normaliser ceux-ci en les multipliant par le nombre de Processeurs Elémentaires utilisés.

Nous obtenons alors des temps normalisés qui seront identiques, pour un algorithme donné, sur, par exemple, des Processor Arrays constitués des mêmes Processeurs Elémentaires mais correspondant à des matrices de tailles différentes.

Si nous appliquons ce critère aux trois algorithmes de Binarisation , de Convolution 3X3 et de Filtre pseudo-médian 3X3 , nous obtenons les résultats indiqués dans le tableau de la figure 4. Nous avons présenté dans ce même tableau les temps absolus et normalisés pour MPP, obtenus à partir des performances de ce système [POTT83]. Les temps normalisés de MPP ont été divisés par 8 pour tenir compte du fait que les Processeurs Elémentaires utilisés ne travaillent pas au niveau de l'octet mais sont binaires. Il faudrait, en tout état de cause, ramener ces temps aux nombres de transistors employés. Néanmoins, ces chiffres montrent que, en temps absolus, la supériorité de MPP est d'autant plus faible que l'algorithme considéré est moins élémentaire. Ainsi pour la convolution, MPP n'est que 8 fois plus rapide alors qu'il met en oeuvre 512 fois plus de Processeurs Elémentaires binaires, soit l'équivalent de 64 fois plus de Processeurs Elémentaires d'un octet.

Par contre, si nous prenons les temps normalisés, nous constatons alors que MPP est à peine plus performant pour la Binarisation mais que le rapport s'inverse avec le Filtre Pseudo-médian et qu'il est nettement moins performant pour la Convolution, le rapport étant de 1 à 8.8 en faveur de SYMPATI.

	Temps Total Image 512x512			
	Absolu		Normalisé	
	MPP	SYMPATI	MPP	SYMPATI
Binarisation	37 µs	3.2 ms	75 ms	102 ms
Pseudo-TUKEY	283 µs	14.7 ms	580 ms	470 ms
Convolution	3400 µs	24.6 ms	6922 ms	787 ms

Figure 4

Exemples de temps absolus et normalisés

Nous obtiendrions des résultats encore plus favorables à la structure de Processeur Ligne avec des algorithmes de propagation comme ceux que nous avons cités au paragraphe précédent. Or le principe de la propagation est parfois incontournable. C'est le cas, comme nous l'avons dit, avec la Labellisation et c'est également vrai pour le calcul de l'Ordre des points. Les temps considérés sont des temps que nous avons estimés à partir des données précédentes [POTT83].

Si nous prenons le cas de la spirale de la figure 2, dont la longueur est 2000, elle nécessite 10 balayages sur SYMPATI, soit un temps total d'exécution de 49 ms. Elle nécessiterait 2000 pas de propagation sur un Processor Array bi-dimensionnel. Or nous pouvons estimer le temps de chaque pas de propagation à environ 300 µs, c'est à dire l'ordre de grandeur du temps d'exécution du Filtre Pseudo-médian. Le temps total d'exécution, absolu, serait donc de 600 ms, soit 12 fois plus qu'avec un Processeur Ligne de 32 Processeurs Elémentaires seulement. Le rapport des temps normalisés serait alors évidemment encore plus favorable aux Processeurs Ligne.

Nous trouverions des résultats semblales avec le calcul de l'Ordre qui nécessite des pas de propagation similaires à ceux de la Labellisation. Le traitement d'un carré de "rayon" 100 par exemple, conduirait à un temps absolu de 30 ms sur MPP alors qu'il ne requiert que 19 ms de temps absolu sur SYMPATI. De plus le temps d'exécution de cet algorithme est indépendant des formes traitées sur un Processeur Ligne.

V. CONCLUSION

Nous avons pu constater l'intérêt de la structure de Processeur Ligne sous 2 aspects. Tout d'abord l'écriture parallèle d'algorithmes , de propagation ou non , est proche de l'écriture séquentielle classique. De plus l'émulation d'un Processor Array bi-dimensionnel sur une telle structure facilite la mise en oeuvre de cette parallélisation. Ensuite, les performances obtenues montrent que les temps absolus des algorithmes indépendants du balayage sont d'autant plus proches des temps obtenus sur des structures de type Processor Array, que les algorithmes considérés sont moins élémentaires. Et la structure de Processeur Ligne se montre plus performante si nous considérons les temps normalisés, mettant en évidence un meilleur rendement. Enfin, le cas des algorithmes de propagation est très favorable aux Processeurs Ligne, en temps absolus et évidemment en temps normalisés. Nous développons actuellement un tel Processeur Ligne en collaboration avec le DEIN du CEA.

Références bibliographiques

[BASI85]
BASILLE J.L. : Structures Parallèles et Traitement d'Images, thèse d'Etat ,Toulouse, 9 déc. 1985.

[BASI86]
BASILLE J.L., DALLE P., CASTAN S.: Iconic and Symbolic Use of a Line Processor in Multilevel Structures, in Intermediate-Level Image Processing, M.J.B. Duff ed., Ac. Press, 1986.

[BATC80]
BATCHER K.E. : Design of a Massively Parallel Processor, IEEE Trans. on Comp., Sept. 1980,pp.1-9.

[DUFF82]
DUFF M.J.B. : CLIP 4, in Special Computer Architectures for Pattern Recognition, K.S. Fu & T.Ichikawa eds., CRC Press, 1982.

[HUNT81]
HUNT D.J. : The ICL DAP and its application to Image Processing, in Languages and Architectures for Image Processing, M.J.B Duff & S. Levialdi eds, Ac.Press, 1981.

[POTT83]
POTTER J.L. : Image Processing on the Massively Parallel Processor, Computer, Jan. 1983, pp. 62-67.

[SHEN86]
SHEN J. : Filtrage Rapide en Traitement d'Image et Vision Tridimensionnelle par Ordinateur, Thèse d'Etat, Toulouse, 15 mai 1986.