Exécution temps réel des détecteurs de contours de Deriche par des processeurs RISC

Thomas EA[1], Lionel LACASSAGNE[1,2], Patrick GARDA[1]

[1]Laboratoire Instruments et Systèmes
Université Pierre et Marie Curie
4 place Jussieu - B.C. 252
75252 Paris Cedex 05

eathomas | lionel | garda @lis.jussieu.fr

[2]Electronique Informatique Applications
Bâtiment 4, Burospace
route de Gisy
91571 Bièvres Cedex

eia@wanadoo.fr

Résumé - Cet article étudie les performances des processeurs RISC pour l'exécution des détecteurs de contours de Deriche. Différentes techniques d'optimisation sur le plan des algorithmes, de l'architecture et de la programmation sont présentées puis réalisées. Les performances de quatre familles de RISC sont évaluées expérimentalement et comparées à l'état de l'art.
Mots clés - détection de contours, filtrage optimal de Deriche, optimisation de code, processeur RISC, déroulement de boucle, analyse de performance.

Abstract - This paper benchmarks RISC processors for the execution of Deriche edge detectors. Different optimizations technics for the algorithms, the architectures and their implementation are presented and applied. A benchmark analysis of four classes of RISC processors is done experimentaly and the results are compared to the state of the art.
Keywords - edge detection, optimal Deriche filters, code optimizations, RISC processors, benchmarks.

1 Introduction

Les opérateurs de Deriche se sont imposés pour la détection de contours dans de nombreux domaines d'applications en traitement d'images et en vision par ordinateur.
Leur principal inconvénient est la lenteur de leur exécution due au grand volume de calculs qu'ils impliquent. Il a conduit à étudier différentes solutions pour accélérer leur exécution s'appuyant sur des processeurs spécialisés (machines parallèles, processeurs de traitement du signal) ou des réalisations électroniques (FPGA, ASIC).
D'un autre côté, les performances des processeurs RISC remettent en cause la nécessité des architectures spécialisées. Cependant il n'est pas évident de choisir entre une architecture spécialisée et un processeur RISC pour une application temps réel.
Cet article donne des éléments quantitatifs pour faire ce choix. Il évalue les performances des processeurs RISC pour l'exécution des opérateurs de Deriche et de Garcia-Lorca et étudie l'optimisation de ces algorithmes pour prendre en compte les spécificités de l'architecture du processeur.

2 Opérateurs de Deriche pour la détection de contours

2.1 Présentation des opérateurs de Deriche

Les opérateurs de Deriche sont utilisés dans deux grandes méthodes de détection de contours : l'une basée sur les maxima locaux du gradient, l'autre sur les passages par zéro du laplacien. Les méthodes récentes combinent le gradient et le laplacien. Les opérateurs implémentés sont ceux proposés par Rachid Deriche dans [3].

2.2 Laplacien de Deriche

Le laplacien s'exprime sous la forme d'un opérateur bidimensionnel approximant une dérivée seconde qui est la différence de deux fonctions de transfert séparables.

$$\Delta(x,y) = e^{-a|x|}.e^{-a|y|} - ka|x|e^{-a|x|}.a|y|e^{-a|y|}$$

Le premier filtre est un lisseur, le second, un dérivateur. Ces filtres sont eux mêmes composés de deux filtres mono-dimensionnels séparables.

2.2.1 Laplacien - Lisseur

$$y_1(n) = x(n) + e^{-a} y_1(n-1)$$
$$y_2(n) = e^{-a}\left[x(n+1) + y_2(n+1)\right]$$
$$y(n) = y_1(n) + y_2(n)$$

2.2.2 Laplacien - Dérivateur

$$y_1(n) = x(n-1) + 2e^{-a} y_1(n-1) - e^{-2a} y_1(n-2)$$
$$y_2(n) = x(n+1) + 2e^{-a} y_2(n+1) - e^{-2a} y_2(n+2)$$

$$y(n) = k\left[y_1(n) + y_2(n)\right]$$
$$k = \frac{1-e^{-2a}}{2}$$

2.3 Gradient de Deriche

La dérivée directionnelle selon x est le résultat d'un lissage suivant la direction y, suivi d'une dérivation suivant x. Pour la dérivée suivant y, on permute les directions.

2.3.1 Gradient Lisseur

$$y_1(n) = k\left[x(n) + e^{-a}(a-1)x(n-1)\right] + 2e^{-a} y_1(n-1) - e^{-2a} y_1(n-2)$$
$$y_2(n) = k\left[e^{-a}(a+1)x(n+1) - e^{-2a} x(n+2)\right] + 2e^{-a} y_2(n+1) - e^{-2a} y_2(n+2)$$
$$y(n) = y_1(n) + y_2(n)$$
$$k = \frac{\left(1-e^{-a}\right)^2}{1+2ae^{-a}-e^{-2a}}$$

2.3.2 Gradient Dérivateur

$$y_1(n) = -kx(n-1) + 2e^{-a} y_1(n-1) - e^{-2a} y_1(n-2)$$

$$y_2(n) = kx(n+1) + 2e^{-a} y_2(n+1) - e^{-2a} y_2(n+2)$$

$$y(n) = y_1(n) + y_2(n)$$
$$k = \left(1-e^{-a}\right)^2$$

3. Opérateurs de Garcia Lorca pour la détection de contours

3.1 Présentation des opérateurs de Garcia Lorca

Dans sa thèse [5], Federico Garcia Lorca a étudié la réalisation des filtres de Deriche. Pour cela, il a introduit une nouvelle démarche algorithmique pour la construction des filtres lisseurs de Deriche et il a proposé de nouveaux opérateurs monodimensionnels d'ordre un plus simples qui s'utilisent en cascade.

3.2 Filtre de Garcia Lorca

Les équations des nouveaux filtres lisseurs cascades sont :

filtre causal : $y(n) = (1-g)x(n) + gy(n-1)$

filtre anticausal : $y(n) = (1-g)x(n) + gy(n+1)$

avec $g = e^{-a}$

En appliquant ce filtre cascade une fois, on obtient une bonne approximation du filtre monodimensionnel de Shen [9], et en l'appliquant deux fois, une approximation du filtre de Deriche.

Les filtres bidimensionnels sont obtenus par l'application horizontale et verticale des filtres monodimensionnels.

3.3 Gradient

Après le lissage, le gradient s'obtient par l'application des noyaux

$$\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \text{ et } \begin{bmatrix} -1 & -1 \\ +1 & +1 \end{bmatrix}.$$

3.4 Laplacien

Le laplacien est obtenu par l'application, après le lissage, du noyau 8-connexe

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}.$$

4 Optimisations

4.1 Introduction

L'objectif de cette étude est de réaliser l'exécution en temps réel des détecteurs de contours de Deriche. Dans cette section nous introduisons les techniques d'optimisations qui permettent d'obtenir une telle exécution sur des processeurs RISC programmés dans un langage

de haut niveau. Tous les calculs sont effectués en flottant.

4.2 Optimisation algorithmique

La première des optimisations est de modifier le filtre pour diminuer sa complexité, c'est-à-dire le nombre d'opérations réalisées par pixel. Le pas principal a été fait par Garcia Lorca dans [5]. Ces filtres étant appliqués deux fois pour Deriche, on peut encore gagner une multiplication en utilisant un opérateur du second ordre qui est le carré de l'opérateur du premier ordre :

$$y_1(n) = (1-g)^2 x(n) + 2gy(n-1) - g^2 y(n-2)$$

$$y_2(n) = (1-g)^2 x(n) + 2gy(n+1) + g^2 y(n+2)$$

La complexité des opérateurs est donnée pour les trois filtres (Deriche, FGL ordre 1 et FGL ordre 2) dans le tableau suivant :

	Deriche		FGL ordre 1		FGL ordre 2	
	MUL	ADD	MUL	ADD	MUL	ADD
gradient	26	24	16	14	12	14
laplacien	14	16	17	16	13	16
ensemble	40	40	17	22	13	24

Tableau 1 : complexité des opérateurs

La complexité de ces filtres est donc plus de deux fois plus petite que celle de Deriche.

4.3 Architecture des processeurs RISC

Pour accélérer le traitement, il faut alimenter au mieux le pipeline et les unités de traitement, limiter les accès mémoire, utiliser efficacement le cache, et employer les registres pour stocker les calculs.

4.4 Optimisations des accès mémoire

Lorsque l'image est parcourue verticalement, deux points contigus d'une colonne ne sont pas côte à côte en mémoire. Cela peut provoquer des défauts de cache en lecture. Pour éliminer ce problème, le filtre vertical est remplacé par un filtre horizontal, appliqué sur la transposée de l'image.

4.5 Optimisation des traitements

4.5.1 Vectorisation des calculs

Le déroulement de boucle est une technique d'optimisation très efficace. L'accélération apportée par cette technique résulte d'une meilleure utilisation du pipeline du processeur [6]. Plutôt que d'exécuter n fois les instructions se trouvant dans le corps de la boucle, on les duplique k fois et on exécute $\left\lfloor \frac{n}{k} \right\rfloor$ fois la boucle.

4.5.2 Utilisation des registres

Les filtres utilisés étant récursifs, en stockant les valeurs de sortie dans les registres du processeur, on diminue les transferts de données avec le cache.

4.5.3 Exemples

Les exemples suivants illustrent le codage des filtres d'ordre 1 et 2 de Garcia Lorca.

Filtre du premier ordre

$$y(n) = b_0 x(n) + a_1 y(n-1)$$

Le corps de boucle est déroulé quatre fois, y_0 contient la dernière sortie, le corps de boucle est :

$$x_0 \leftarrow X[i],\ x_1 \leftarrow X[i+1]$$
$$x_2 \leftarrow X[i+2],\ x_3 \leftarrow X[i+3]$$
$$y_0 \leftarrow b_0 x_0 + a_1 y_0$$
$$y_1 \leftarrow b_0 x_1 + a_1 y_0$$
$$y_2 \leftarrow b_0 x_2 + a_1 y_1$$
$$y_3 \leftarrow b_0 x_3 + a_1 y_2$$
$$Y[i] \leftarrow y_0,\ Y[i+1] \leftarrow y_1$$
$$Y[i+2] \leftarrow y_2,\ Y[i+3] \leftarrow y_3$$
$$y_0 \leftarrow y_3$$

Filtre du second ordre

$$y(n) = b_0 x(n) + a_1 y(n-1) + a_2 y(n-2)$$

Ici, le corps de boucle est déroulé trois fois, au début du corps de boucle y_k contient Y[i-k].

$$x_0 \leftarrow X[i],\ x_1 \leftarrow X[i+1],\ x_2 \leftarrow X[i+2]$$
$$y_0 \leftarrow b_0 x_0 + a_1 y_1 + a_2 y_2$$
$$y_2 \leftarrow b_0 x_1 + a_1 y_0 + a_2 y_1$$
$$y_1 \leftarrow b_0 x_2 + a_1 y_2 + a_2 y_0$$
$$Y[i] \leftarrow y_0,\ Y[i+1] \leftarrow y_2,\ Y[i+2] \leftarrow y_1$$

5 Estimation de performances

5.1 Mode opératoire

Les performances de 4 familles de processeurs RISC (DEC, HP, Intel - AMD, Sun)ont été évaluées pour des images de taille 128 à 640. Chaque mesure a été faite 10 fois. Une régression linéaire a ensuite été appliquée, rendant la mesure plus fiable.

Deux versions de chaque filtre (Deriche et Garcia Lorca) ont été implémentées : une sans aucune optimisation, la seconde avec les optimisations décrites dans la section 4.

Les stations tournent sous Unix. Pour les stations Sun et HP, le compilateur utilisé est gcc 2.7, pour les stations DEC, c'est le compilateur de Digital.

Les processeurs équipant les PC représentent le plus grand des parcs informatiques, il est important d'approfondir leurs analyses. Les processeurs d'Intel ont été comparé à l'AMD K6. Deux compilateurs C également été testé : Microsoft Visual C 4 et Watcom 11. Les Pentiums tournent sous NT 4.0 sauf le Pentium II, le Pentium Pro et le K6 qui tournent sous 95.
Pour le Pentium MMX et le Pentium II, les fonctionnalités multimédia n'ont pas été utilisées, car il aurait été nécessaire d'écrire les routines en assembleur, alors que le but de cet article est d'estimer les performances en C. Par contre l'intérêt du Pentium MMX et du Pentium II est que leurs caches de niveaux 1 sont deux fois plus grands que ceux des Pentiums non MMX.
Les caractéristiques des processeurs testés sont résumées dans le tableau 2.

Machines et Processeurs	Fréquence (MHz)	Cache niveau 1 (Ko)	Cache niveau 2/3 (Ko)	RAM (Mo)
PA 7100 LC	100	256+256	0	32
PA 7200	100	256+256	0	64
PA 8000	180	2 Mo	0	64
SuperSparc	50	18+18	0	32
Micro Sparc 2	110	24	0	32
Ultra Sparc 1	143	16+16	512 / 0	64
Ultra Sparc 2	300	16+16	512 / 0	64
Pentium	150 et 166	8+8	256	32
Pentium Pro	180	8+8	256	64
Pentium MMX	200	16+16	512	48
Pentium II	300	16+16	512	32
K6	233	32+32	512	32
Alpha 21164	500	8+8	96 / 8 Mo	128
Alpha 21164	625	8+8	96 / 4 Mo	128

Tableau 2 : caractéristiques des machines testées

5.2 Résultats

Les tableaux donnent pour les processeurs les plus significatifs de chaque famille et pour les tailles d'image 128, 256 et 512, le temps en millisecondes pour réaliser le calcul du gradient, du laplacien et du traitement complet (gradient + laplacien). Le dernier tableau indique la taille maximale de l'image pour laquelle le traitement peut être exécuté en temps réel (durée inférieure à 40 ms).

PA 8000	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	25.9	10.8	18.4	13.0	44.2	16.6
256	114.4	43.8	82.0	54.5	195.6	70.2
512	690.5	301.8	504.5	324.2	1187	399.0

Ultra 1	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	20.1	11.0	16.5	10.0	37.4	14.7
256	94.5	47.4	78.1	41.9	173.5	64.9
512	562.1	253.0	469.9	234.5	1032	324.4

Ultra2	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	11.6	6.1	9.2	5.8	21	9.1
256	58.7	28.4	47.5	23.6	107	37.9
512	370.4	169.7	331.6	151.3	699.2	214

Ppro 180	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	30.2	12.2	25.9	8.4	55.4	16.3
256	192.4	72.4	176	51.2	370.1	85.5
512	805.6	304.1	703.5	224.8	1511	348.9

MMX 200	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	29.1	11.9	23.5	11.5	53.1	14.8
256	149.8	63.5	121	46.6	271.5	73.6
512	660.6	259.9	530	225.4	1194	340.6

PII 300	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	13.6	5.1	13.6	4.3	29.7	6.3
256	89.3	33.6	80.5	22.2	171.6	37.6
512	473.5	175.9	417.1	148.1	893	207.6

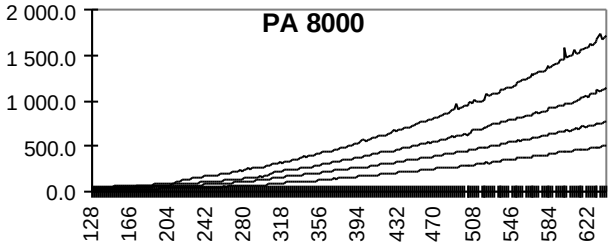
K6 233	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	36.8	17.9	31.3	17.5	74.4	22.8
256	194	91.8	151.2	63.7	347.2	98.7
512	1083.6	399.8	884.2	302.8	1978.8	433.6

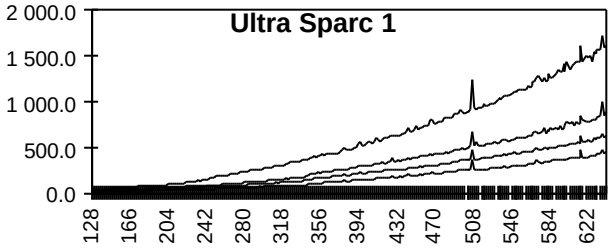
Alpha 500	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	10.5	4.7	8.8	4.7	17.7	5.9
256	46.2	13.3	39.3	17.1	86.9	22.4
512	262.2	71.7	230.7	73.8	519.5	94.6

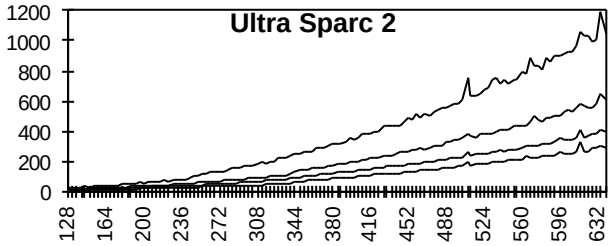
Alpha 625	Gradient		Laplacien		Total	
Taille	Deriche	FGL	Deriche	FGL	Deriche	FGL
128	7.6	3.6	6.4	3.4	13.4	4.3
256	34.2	16.2	28.6	13.4	63.8	19.8
512	190.0	73.9	174.3	62.3	367.6	99.9

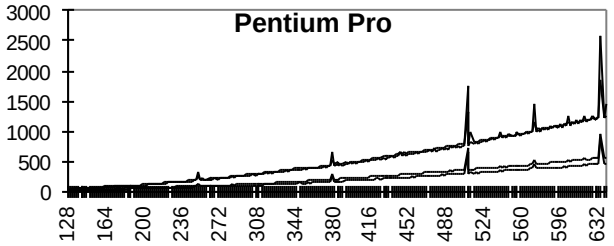
	Gradient		Laplacien		Ensemble	
Machines	Deriche	FGL	Deriche	FGL	Deriche	FGL
PA 7100 LC	-	178	130	190	-	160
PA 7200	132	182	138	206	-	162
PA 8000	158	246	182	226	-	198
Super Sparc	-	-	-	-	-	-
Micro Sparc 2	-	130	-	142	-	-
Ultra Sparc 1	172	234	188	240	128	202
Ultra Sparc 2	220	292	232	324	168	256
P 166	-	182	142	220	-	172
P 150	132	194	146	220	-	180
P Pro 180	139	196	140	226	-	184
P 200 MMX	144	216	156	244	-	200
PII 300	196	280	196	312	152	260
K6 233	136	176	152	236	-	176
Alpha 500	248	392	276	388	188	344
Alpha 625	292	404	320	432	208	348

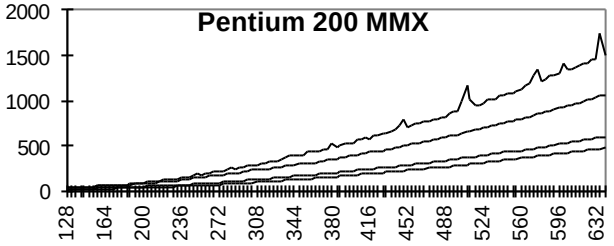
Les courbes suivantes donnent les temps d'exécution du gradient, pour les algorithmes non optimisés et optimisés de Deriche et de Garcia Lorca.

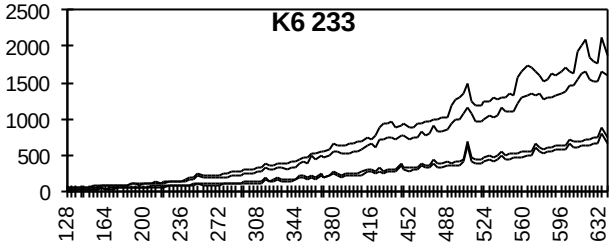


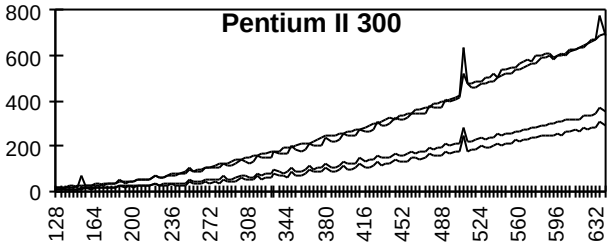


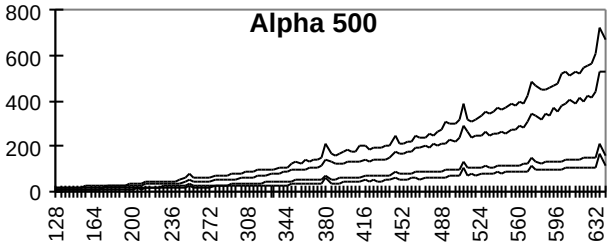


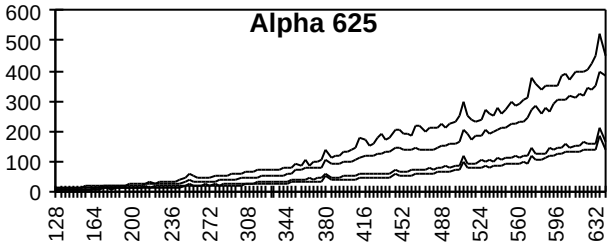












5.3 Analyse des résultats

5.3.1 Comparaisons des algorithmes

Les courbes représentant les trois traitement ont des allures très voisines. Le gain entre les versions non-optimisées et optimisées est plus grand pour les filtres de Deriche (50%) que pour les filtres de Garcia Lorca (10%). Cela doit être dû à la qualité des compilateurs et à l'architecture des processeurs : les filtres de Garcia-Lorca étant plus simples, ils sont plus facilement optimisés par le compilateur. Par contre le rapport entre les durées des versions optimisées de Garcia-Lorca et de Deriche est de l'ordre de 2 à 3 pour le gradient et

le laplacien (voire 5 pour l'Alpha) et de 3 à 5 pour le traitement complet (voire 10 pour l'Alpha).

5.3.2 Comparaisons des architectures

Les temps de PA8000, de l'Ultra2 et du Pentium II sont très proches. Par contre, les deux Alpha sont nettement plus rapides que ces trois machines : de 2 à 3 fois pour Deriche et de 2 à 5 fois pour Garcia-Lorca.

Les défauts de caches apparaissent pour tous les opérateurs, sur toutes les machines et pour les mêmes tailles d'image. Pour ces tailles critiques, le nombre de défauts de cache est liés à la taille des caches : le Pentium Pro en génère énormément, alors que le PA8000 n'en génère quasiment pas.

5.3.3 Comparaisons des PC

Le Pentium MMX 200 est plus rapide qu'un Pentium Pro 180 : si sa FPU est moins performante, ses caches, deux fois plus grands contre-balancent les performances. Ramené à fréquences égales, ils sont aussi rapides l'un que l'autre.

Le K6 est nettement en deçà des Pentiums, à fréquence égale (plus de 50 % plus lent que le MMX 200). Cela s'explique par le fait que la FPU du K6 n'est pas entièrement pipelinée.

Le Pentium II est deux fois plus rapide que les autres Pentiums parce qu'il combine trois avantages : la taille du cache a été doublée, il possède la FPU et l'architecture DIB (Dual Instruction Bus) du Pentium Pro et surtout, il a une horloge à 300 Mhz. Avec l'augmentation de la fréquence d'horloge seule, il n'aurait été que 1.5 fois plus rapide que le MMX 200 ou le Ppro 180. L'utilisation de cartes mères à 100 Mhz (et non plus seulement à 66 Mhz) permettra de tirer pleinement parti de l'efficacité de la SDRAM (dans ces tests, le K6 était pourvu de SDRAM à 66 Mhz mais cela n'a pas d'impact significatif).

Le compilateur Visual C4 est de 20 à 45 % plus rapide que Watcom 11.

6 Comparaison des détecteurs à l'état de l'art

6.1 Introduction

Cette section présente les réalisations matérielles et logicielles des filtres récursifs pour la détection de contours, puis une comparaison avec les résultats de la section 5.

6.2 Réalisations matérielles

Les réalisations matérielles effectuent rarement la détection de contours complètement. De ce fait nous présentons dans le tableau 3 (temps en ms) les performances obtenues pour le calcul du gradient où les calculs sont effectués en virgule fixe.

Laboratoire	Type	Filtre	Temps 256^2	Temps 512^2
ETIS [5]	ASIC	Shen Deriche	3.9	15.4
	FGPA	Shen Deriche	6.6	26.2
LIESIB [8]	ASIC	BPT	3.3	13.1
	ASIC	BPT	6.6	26.6
LIRMM [10]	ASIC	BPT	1.6	6.6

Tableau 3 : réalisations matérielles

6.3 Réalisations parallèles.

Laboratoire	Machine	Temps 256^2	Temps 512^2
LASMEA	Transvision	1544	-
Université Washington	DSP C80	-	152

Tableau 4 : réalisations logicielles

La Transvision [2] est composée de 4 transputers T800. Le traitement se fait en pipeline : le calcul du gradient se fait en 2 étapes et une troisième permet d'extraire les maxima locaux. Les calculs sont faits en flottant 32 bits. Il existe aussi des versions de la Transvision à base de C40 ou de T9000 sur lesquelles l'implémentation des algorithmes est en cours. Les performances seront meilleures, mais il sera difficile d'atteindre le temps réel.

Le TMS320C80 [7] est un processeur de traitement du signal. Le gradient est calculé par la méthode de Canny, les maxima locaux sont ensuite extraits. Ces calculs sont faits en entier.

6.4 Comparaison avec l'état de l'art

La comparaison entre les réalisations matérielles et les réalisations logicielles peut se faire de deux manières : en comparant le temps d'exécution ou le temps de réalisation. Le tableau 5 indique le gain en terme de temps d'exécution des meilleures

réalisations matérielles par rapport au meilleur des processeurs RISC évalués dans la section 5.

	gain 256^2	gain 512^2
FPGA	2.0	2.7
semi-custom	3.4	4.7
full-custom	8.3	10.9

Tableau 5 : gain par rapport aux RISC

D'un autre coté, les réalisations matérielles nécessitent de longs développements (VHDL et CAO), alors que les RISC se programment en langage C. De plus, elles font les calculs en entier alors que les RISC les font en flottant.
Des travaux en cours entre EIA et le LIS permettent d'atteindre les latences des FPGA, en programmant en assembleur le C80 et le C62, ce qui représente un gain de 4 à 7 par rapport à l'opérateur de Canny de la bibliothèque de Washington.

7 Conclusion

Dans cet article, nous avons étudié les performances de processeurs RISC pour l'exécution des opérateurs de Deriche et des nouveaux opérateurs proposés par Garcia Lorca.
Cette étude a montré que l'optimisation du code dans un langage de haut niveau apporte des gains de 10 à 50 % sur le temps d'exécution. Les filtres de Garcia Lorca permettent un gain de l'ordre de 2 à 5 par rapport aux filtres de Deriche.
Les résultats obtenus situent les possibilités des processeurs RISC pour une exécution des différents opérateurs de détection de contours à la cadence vidéo (25 images secondes soit 40 ms par image).
D'une part les dernières générations de processeurs (Pentium II, et Ultra Sparc 2) peuvent traiter des images 256×256 en temps réel, lorsqu'on ne calcule que le gradient ou le laplacien par Garcia Lorca. Mais l'Alpha effectue le traitement complet en temps réel : il s'agit, à notre connaissance, de la première exécution logicielle de Deriche en temps réel sur des images 256×256 pour des processeurs RISC.
D'autre part, la plus rapide des stations (l'Alpha) met 70 ms pour des images 512×512 : le gain à gagner pour atteindre la cadence vidéo est inférieur à deux !
Or les futures générations de processeurs (Deschutes, PA8500, Merced, Alpha 21264) apporteront un gain de 2, du seul fait du changement d'horloge. Ces machines devraient donc être capables d'égaler la vitesse des FPGA et de traiter des images 512×512 en temps réel.

8 Remerciements

Nous remercions toutes les sociétés qui ont mis à disposition des machines, nous permettant ainsi de réaliser cette étude comparative.

Processeur Alpha :
CCR M. Racine.
DEC MM. Decamps, Verdun.
ESC M. Bostroem.
Processeur HP :
HP MM. Missélis, Coslovi, Robson.
Processeur Intel :
Gateway MM. Gontier, Arikessavane.
Intel M. Colin.
Processeur Sun :
IEF M. Mérigot.

Bibliographie

[1] J.P. Cocquerez et S. Philipp
Analyse d'image : filtrage et segmentation, Masson

[2] J.P. Derutin, B. Besserer, T. Tixien, A. Klickel
A Parallel Vision Machine : Transvision
CAMP91, Computer and Machine Perception, Décembre 1991

[3] R. Deriche
Fast Algorithms for low level-vision
IEEE Transaction on Pattern Analysis and Machine Intelligence, vol 12, n°1, January 1990

[4] B. El-Bay
Conception et implémentation d'un détecteur de contours optimisé sous forme d'un circuit ASIC
Thèse de l'Université de Bourgogne, 1994

[5] F. Garcia Lorca
Filtres récursifs temps réel pour la détection de contours : optimisations algorithmiques et architecturale.
Thèse de l'Université d'Orsay, novembre 1996.

[6] John L. Hennessy, David A. Patterson
Architecture des ordinateurs
traduit par Daniel Etiemble, Thomson Publishing

[7] Y. Kim, kim@ee.washington.edu
Université de Washington
http : //icsl.ee.washington.edu/projects/iclib

[8] Sarifuddin
Implémentation sous forme de circuit spécialisé d'un algorithme de détection de contours multi-échelles
Thèse de l'Université de Bourgogne, 1995

[9] J. Shen, S. Castan
An optimal linear operator for step edge detection
CVGIP : graphical models and image processing
Vol 64, n°2, March 1992

[10] L. Torres
Intégration de filtres numériques pour le traitement d'images : du silicium au système reconfigurable.
Thèse de l'Université de Montpellier II, 1996