Apport des Instructions Multimédia
—
Quelques Techniques de Compilation

Ronan.Keryell@enstb.org

—

Département Informatique

École Nationale Supérieure des Télécommunications de Bretagne

Master Recherche 2ème année Informatique de Rennes 1 — ENSTBr
ISIA — ENSMP
octobre 2006–février 2007
Version 1.9

Résumé

On présentera sommairement les instructions dites multimédia introduites dans les dernières générations de processeurs, telles que les Pentium d'Intel avec instructions MMX. Elles consistent à segmenter les ALU pour traiter par exemple de manière SIMD 8 données sur 8 bits stockées dans les registres 64 bits du processeur afin d'accélérer les applications nécessitant beaucoup de calculs sur de petits types de données. Quelques exemples d'applications sont abordés et une méthode de compilation est introduite.

Introduction 1

- Explosion du marché du multimédia et des télécommunications
  - ► Affichage et synthèse d'image 2D & 3D
  - ► Son (stéréo et plus)
  - ► Télécommunications rapides (modem)
  - ► Compression & décompression (MPEG-2 $\approx$ 1 GOPS)
- Point commun $\rightsquigarrow$ beaucoup de calculs
- Processeurs génériques de plus en plus puissants +60 %/an (11,7 à 19,8 SPECint95)
- Mémoire $\nearrow$ +60 %/an
- Baisse des coûts $\rightsquigarrow$ intégration & factorisation de ces moyens de calcul

Processeur générique 2

- Puissant : UAL 64 bits, extraction matérielle du parallélisme
- Pas d'optimisations spécifiques
- Optimisé pour des types de données « classiques » (C) : entiers 32 bits, nombres flottants (grand public ?)
- Pas adapté aux applications intensives sur des petites données :
  - ► image GIF : 8 bits
  - ► image RVBα en vrai couleur 4×8 bits
  - ► son téléphone encodage A ou μ : 8 bits
  - ► son qualité CD : 2×16 bits
- $\rightsquigarrow$ transistors sous-utilisés pour ces applications (multiplieur double précision...)

- Rajouter les opérations les plus utiles qui manquent
- Économie sur l'emballage des opérations (monoprocesseur)
- Opérations à parallélisme explicite SIMD
- Réutilisation de parties d'opérateurs (multiplieur flottant)
- Transparent au système d'exploitation : données dans registres flottants)

- Demander au compilateur de faire des efforts
  `gcc -march=pentium4 -mfpmath=sse`
- Bibliothèques optimisées du constructeur (Direct3D)
  - ► Gain immédiat et facile sans recompilation
  - ► Surcoût d'interface
  - ► Pas toujours les fonctions critiques qui sont optimisées
- Réécriture de portions du code en assembleur avec instructions multimédia
  - ► Flexibilité
  - ► Difficile & sujet à erreurs
  - ► Pas évident de prévoir le gain vue la complexité des micro-architectures (Intel...)

- Codage langage de haut niveau avec des macros constructeurs pour accéder aux primitives multimédia
  - ► Abstraction de type fonction représentant une instruction multimédia
  - ► Laisse au compilateur gestion des registres & déroulage
  - ► Intégration plus fluide au code
  - ► Portabilité via une redéfinition des macros
  - ► Classes C++ pour des opérations SIMD (`F64vec2`, `Is16vec8`,...)
- Compilation automatique (recherche...)
  - ► Optimisation pour les flux importants de données
  - ► Exploitation micro-parallélisme (inférence des types)
  - ► Vectorisation automatique (mais C et non Fortran en général...)

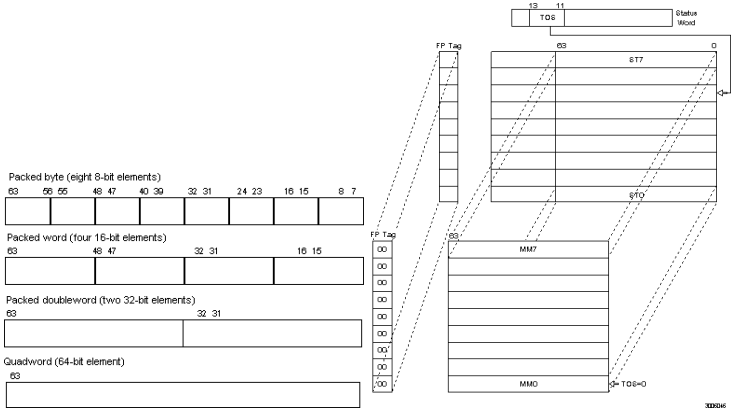
- Exploitation recherche en compilation pour ordinateurs parallèles

- Processeurs avec instructions multimédia
- Exemples d'applications
- Techniques de compilation

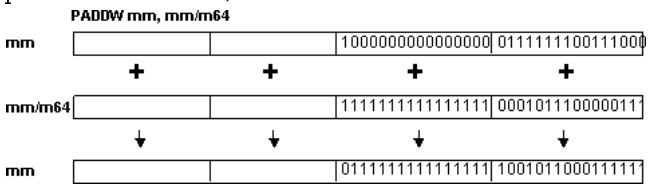
- Intel : i860, Pentium (Pro, II) avec MMX (MultiMedia eXtensions)
- SGI/MIPS : MIPS-V (R10000) MDMX
- HP : PA-RISC MAX-2 (Multimedia Acceleration eXtensions)
- Digital : MVI dans Alpha 21164PC (Motion Video Instructions)
- Moins génériques : Philips TriMedia,...

... et du vieux avec du neuf !

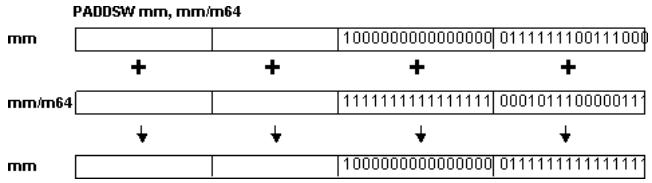
- Registres vectoriels (8 dans MMX) dans registres flottants



- Instructions vectorielles : `paddb %mm0, %mm1` ≡ 8 additions sur 8 bits, `paddw` 4 sur 16 bits,...



- Pas de scatter-gather ou d'indirection locale
- Opérations de (dé)compactage
- Éventuellement registres de masque (Sun VIS)
- Éventuellement registres d'accumulation (MDMX)
- Gestion des débordements avec saturation



- Superscalaire : 2 opérations MMX/cycle (4,8 GOPS 8 bits à 300 MHz)

Visual Instruction Set

- 32 registres (flottants)
- Opérations partitionnées SIMD
- Multiplications partitionnées style $8 \times 8 \rightarrow 8$ ou $8 \times 8 \rightarrow 16$
- Opérations de comparaison générant un masque
- Instructions de réalignement
- Instructions de compactage/décompactage/fusion
- Conversion adresse 3D (3×11b,11b) $\rightarrow$ 1D
- Écriture suivant masque
- Calcul de distance pixel ($\mathcal{L}_1$) (MPEG-2, H.261)

MIPS Digital Media eXtension

- 32 registres (flottants)
- Opérations partitionnées SIMD en 8 ou 16 bits
- Accumulateur privé de 192 bits partitionné en données signées de taille $3\times$ type de l'opération
- Possibilité de promouvoir des scalaires : `add.qh $v10,$v9,25` et `adda.ob $v4,$v6[7]`
- Instructions de chargement acceptant le non-alignement
- Instructions de compactage/décompactage/fusion
- Comparaison générant une condition vectorielle
- Min/max
- Multiplication d'un vecteur par le signe d'un autre
- Multiplications partitionnées avec saturation ou accumulation
- Opération d'homothétie sur l'accumulateur avec arrondi

Multimedia Acceleration eXtensions

- 32 registres (flottants)
- Opérations partitionnées SIMD avec ou sans saturation
- Moyennage arithmétique entre 2 registres
- Permutations, compactages/décompactages
- 2 opérations + 2 accès mémoires simultanés dans un PA-8000
- Occupe 0,1 % de la surface d'un PA-8000

MultiMedia eXtensions

- 8 registres (flottants)...
- Opérations partitionnées SIMD avec ou sans saturation
- Multiplication-addition $8 \times 8 + 8 \times 8 \rightarrow 16$
- Opérations de comparaison générant un masque
- Compactages/décompactages
- Opération de nettoyage des registres MMX (50 cycles)
- 2 opérations simultanées (+ accès mémoire sur Pentium II)
- Pas de chargement de constantes
- Puissance d'expression entre MAX-2 et VIS


- Modèle CISC remontant au 4004...
- MMX rajoute un caractère RISC
- 8 registres génériques 32 bits + 8 registres flottants ou MMX
- Processeur non orthogonal, registres spécialisés
- Instructions 2 adresses (`a += b` et pas `a = b + c`)
- 2 pipes U & V d'exécution superscalaire (+ 3 pour les accès à la mémoire du Pentium II)
- Conditions d'exécution sur les 2 pipes très compliqués, encore pire avec les micro-instructions du Pentium II
- Difficile de prévoir le comportement et d'écrire du code optimal
- IA-64 VLIW : 128 registres entiers de 64 bits, 128 registres flottants... MMX ?

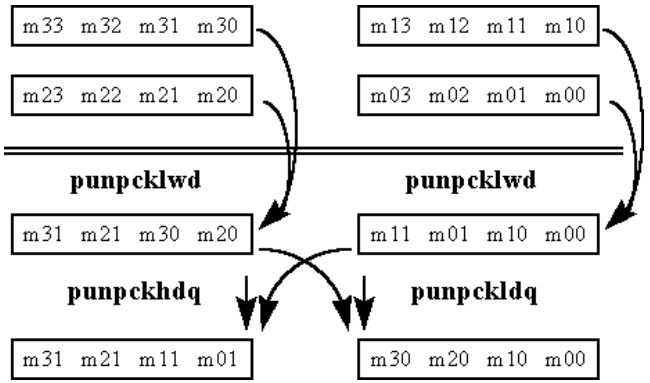
Mais processeur très répandu...

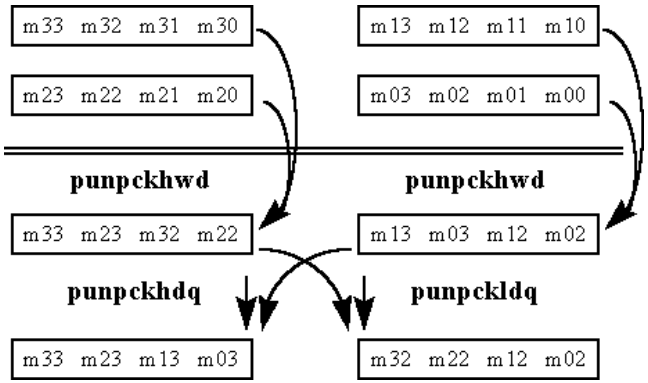

- Partir d'un algorithme et d'un code déjà bien optimisé
- Alignement des données sur 8 octets
- 1 seul décaleur et multiplieur disponible
- 1 seul accès à la mémoire ou MMX-registres entiers
- Localité du cache & précharge
- Utiliser les instructions « RISC »
- Ne pas mélanger code flottant et MMX
- Éviter le code auto-modificateur
- ...
- Utiliser les compteurs de mesure

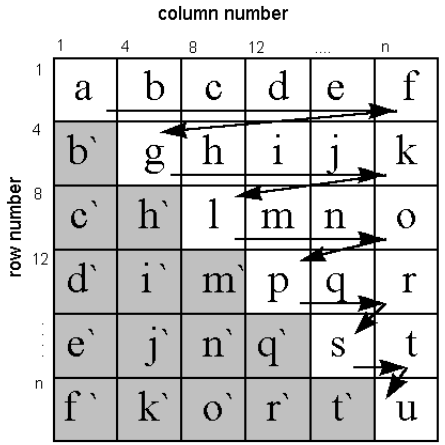

- Extensions registres MM en XMM 128 bits
- Permet aussi d'accueillir des flottants 32 ou 64 bits
- Rajout d'instructions flottantes
- ...Et de difficultés : exceptions flottantes
- Instructions de min/max
- `http://developer.intel.com/software/products/itc/sse2/simd2.htm`
- `http://developer.intel.com/software/products/itc/sse2/sse2_appnotes.htm`


$$A_{i,j} = B_{j,i}$$

- Déroulage de boucle & pipeline logiciel
- Approche naïve : 1 lecture & 1 écriture/élément $\equiv$ 2 cycles/élément
- Utilisation des instructions de (dé)compactage en traitant des sous-matrices 4×4 de 16 bits



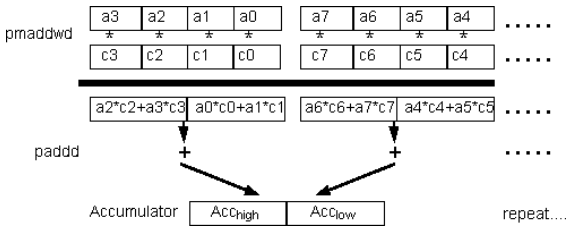




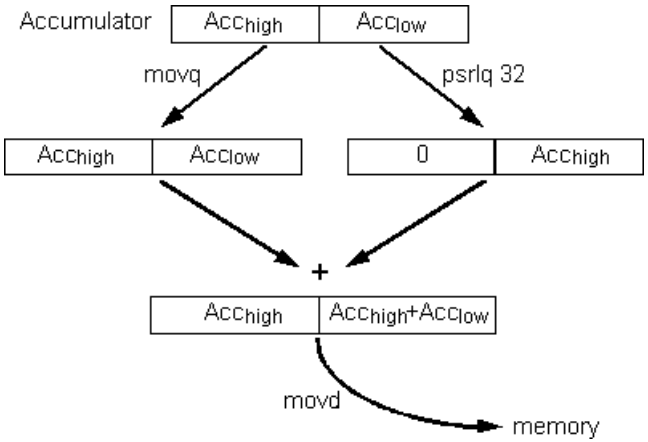
4 lectures, 4 écritures, 8 instructions $\equiv$ 1 cycle/élément

$$s = \sum_{i=1}^{n} x_i \cdot y_i$$

- Déroulage de boucle & pipeline logiciel $\rightsquigarrow$ sommes partielles
- Approche naïve : $2n$ lectures, n multiplications, $n-1$ additions, 1 écriture
- Utilisation de `pmadd` en utilisant 2 réductions séparées



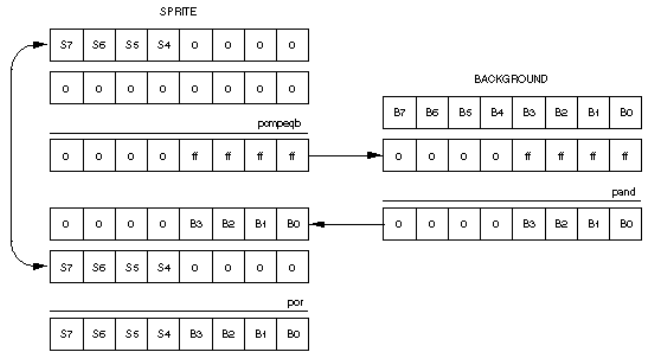
2 `pmaddwd`, 2 `paddd`, 2 lectures/8 éléments 16 bits



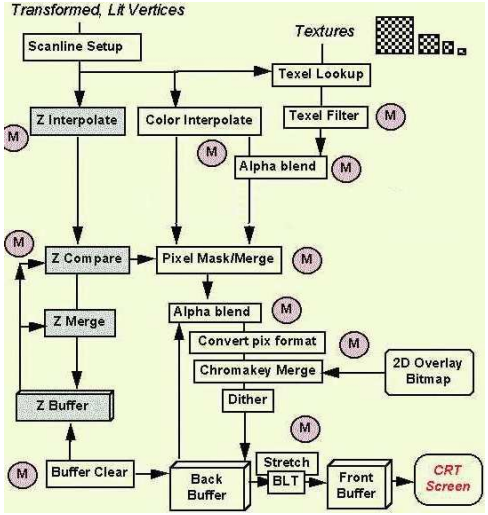
\+ 3 opérations de réduction finale

$$\text{écran}_{i+X,j+Y} = \text{sprite}_{i,j} \quad \texttt{si} \quad \text{sprite}_{i,j} \neq 0 \qquad (1)$$

$$\text{écran}_{i+X,j+Y} \quad \texttt{sinon} \qquad (2)$$



2 lectures, 1 écriture, 3 instructions pour 8 éléments 8 bits



Quand utiliser MMX ?

- Si pas d'accélérateur matériel
- Si pas les bonnes fonctions disponibles
- Si triangles petits (temps de démarrage du matériel)

Instructions inhabituelles...

- $\max(x, y) = x + (y - x)^+$ en non signé
- clipping non signé dans $[l, h]$
  $y = ((x + (-1 - h))^+ - (l - h - 1))^+ + l$
- Multiplication complexe `%mm0`=(d_r, d_i), `%mm1`=$(c_r, -c_i, c_i, c_r)$

```
punpckldq %mm0, %mm0   ;   %mm0=(dr, di, dr, di),
pmaddwd %mm1, %mm0     ;   %mm0=(dr cr − di ci, dr ci + di cr)
```

⇝ Pattern matching puissant

- Extraction du parallélisme (graphe de dépendance)
- Mise sous forme normale des espaces d'itération
- *If-conversion* ⇝ instructions de masquages
- Pattern matching
- Strip-mining
- Pipeline logiciel & précharge de cache
- Fusion de boucles parallèles conformantes (limitation débit mémoire) & privatisation de tableaux
- Ordonnancement des opérations & allocation des registres
- Restructuration des données autour des zones de calcul, alignement données/itérations (loop-peeling) ⇝ grandes zones
- Nettoyage : élimination de code mort,...

- Rajout de directives ? HPF ? Extension à plusieurs processeurs

En fait, interdépendances entre phases, haut et bas niveau...

- Passage du nid de boucle dans un formalisme d'algèbre linéaire

  Nid de boucle $\equiv$ Polyèdre

- Transformations matricielles. Strip-mining :

```
for (c = 0; c < C; c++) { f(c); }
```

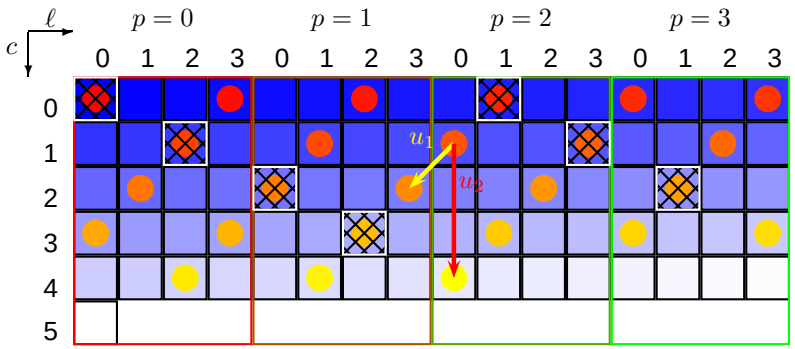
$$\{c | 0 \leq c < C\} \quad \rightsquigarrow \quad \{(c', c'') | c = \Gamma c' + c'', 0 \leq c'' < \Gamma, 0 \leq c < C\}$$

```
for (cp = 0; c < C; c += Gamma)
  for (cpp = 0; cpp < Gamma && cp + cpp < C; cpp++) { f(cp + cpp); }
```

- Forme normale de Hermite
- Regénérer du code à partir des équations

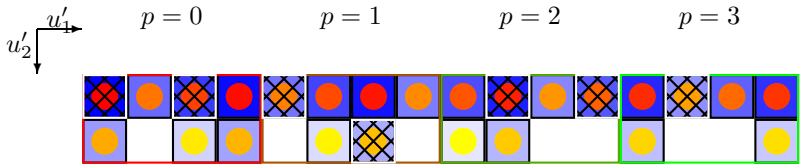

```
      real A(0:24), B(0:24)   ! 0 ≤ a_A ≤ 24, 0 ≤ a_B ≤ 24
!HPF$ template T(0:80)        ! 0 ≤ t ≤ 80
!HPF$ processors P(0:3)       ! 0 ≤ p ≤ 3
!HPF$ align A(i) with T(3*i)  ! a_A = 3 t
!HPF$ align B(i) with A(i)    ! a_A = a_B
!HPF$ distribute T(cyclic(4)) onto P ! t = 16c + 4p + ℓ
                                     ! 0 ≤ ℓ < 4
      A(0:U:3) = A(0:U:3) + B(1:U+1:3) ! i = 3i', 0 ≤ i ≤ U
                                       ! a = i
```

Par exemple $\mathtt{U} = 20$:



: template, : tableau, : calcul

- HPF : autorise des trous $\rightsquigarrow$ Compromis temps-espace
- Méthode :
  - Base (c, ℓ)
  - Base du cristal (u_1, u_2) via HERMITE
  - Rectangularisation du tableau en (u'_1, u'_2) par division entière



: template, : tableau, : calcul

Équipe SUIF de Stanford (Monica Lam)

- Optimiseur pour VIS ciblant décodeur vidéo MPEG
- Atomise le code, distribue les boucles
- Vectorisation automatique (nombre infini de registres, longueur infinie)
- Transformation du code vectoriel en VIS
- Réductions à rajouter
- Améliorer la localité (fusion de boucle, plus de vecteurs infinis)
- Encore un prototype

- Gain sur des tâches répétitives sur de petits types de données
- Expérimentation sur un code de reconnaissance d'empreintes digitales avec la SAGEM et étude de codes de sonar avec Thomson
- Optimisation style MMX à partir de l'expérience en parallélisation automatique
- Tâche encore complexe à automatiser par rapport aux machines parallèles classiques : parallélisme inter-processeur et intra-processeur
- Démarrage d'une thèse utilisant l'infrastructure de PIPS (ressemblances avec HPF)
- Intéressé par des collaborations industrielles

Liste des transparents