XFOR : une structure itérative pour le contrôle explicite de la distance de réutilisation des données

Imen Fassi [a,b], *Philippe Clauss* [b], *Alain Ketterlin* [b], *Alexandra Jimborean* [c], *Yosr Slama* [a]

[a] Dpt Informatique, Faculté des sciences, Université El Manar, Tunisie
[b] CAMUS, INRIA, Université de Strasbourg, France
[c] Dpt of Information Technology, Uppsala University, Suède

Exemple

- Code Seidel (Polybenchs) :

```
for (i=1 ; i<=n-2 ; i++)
 for (j=1 ; j<=n-2 ; j++)
  A[i][j] = (A[i-1][j-1]+A[i-1][j]+A[i-1][j+1]
           +A[i][j-1]+A[i][j]+A[i][j+1]
           +A[i+1][j-1]+A[i+1][j]
           +A[i+1][j+1])/9.0;
```

- chaque élément accédé 9 fois
- 4 éléments lus déjà mis à jour (ordre lexico)

Exemple

- Minimisation des distances de réutilisation :
  - découpage en 5 instructions : additions de 4 éléments non encore mis à jour + addition des autres éléments

```
xfor (i0=1,i1=1,i2=1,i3=1,i4=1 ;
      i0<=n-2,i1<=n-2,i2<=n-2,i3<=n-2,i4<=n-2 ;
      i0++,i1++,i2++,i3++,i4++ ;
      1,1,1,1,1 ; /* grains */
      0,1,1,1,1 ) /* offsets */ {
xfor (j0=1,j1=1,j2=1,j3=1,j4=1 ;
      j0<=n-2,j1<=n-2,j2<=n-2,j3<=n-2,j4<=n-2 ;
      j0++,j1++,j2++,j3++,j4++ ;
      1,1,1,1,1 ; /* grains */
      2,0,1,2,2 ) /* offsets */ {

      0: A[i0][j0] += A[i0][j0+1] ;
      1: A[i1][j1] += A[i1+1][j1-1] ;
      2: A[i2][j2] += A[i2+1][j2] ;
      3: A[i3][j3] += A[i3+1][j3+1] ;
      4: A[i4][j4] = (A[i4][j4]+A[i4-1][j4-1]
                      +A[i4-1][j4]+A[i4-1][j4+1]
                      +A[i4][j4-1])/9.0 ; } }
```

Exemple

- Minimisation des distances de réutilisation :
  - découpage en 5 instructions : additions de 4 éléments non encore mis à jour + addition des autres éléments

```
xfor (i0=1,i1=1,i2=1,i3=1,i4=1 ;
      i0<=n-2,i1<=n-2,i2<=n-2,i3<=n-2,i4<=n-2 ;
      i0++,i1++,i2++,i3++,i4++ ;
      1,1,1,1,1 ; /* grains */
      0,1,1,1,1 ) /* offsets */ {
xfor (j0=1,j1=1,j2=1,j3=1,j4=1 ;
      j0<=n-2,j1<=n-2,j2<=n-2,j3<=n-2,j4<=n-2 ;
      j0++,j1++,j2++,j3++,j4++ ;
      1,1,1,1,1 ; /* grains */
      2,0,1,2,2 ) /* offsets */ {
  0: A[i][j-2]   += A[i][j-1] ; /* translation de 0 sur axe i, 2 sur axe j */
  1: A[i-1][j]   += A[i][j-1] ; /* translation de 1 sur axe i, 0 sur axe j */
  2: A[i-1][j-1] += A[i][j-1] ; /* translation de 1 sur axe i, 1 sur axe j */
  3: A[i-1][j-2] += A[i][j-1] ; /* translation de 1 sur axe i, 2 sur axe j */
  4: A[i-1][j-2] = (A[i-1][j-2]+A[i-2][j-3] /* dependance => idem precedent */
                   +A[i-2][j-2]+A[i-2][j-1]  /* ordre lexico des offsets */
                   +A[i-1][j-3])/9.0 ; } }
```

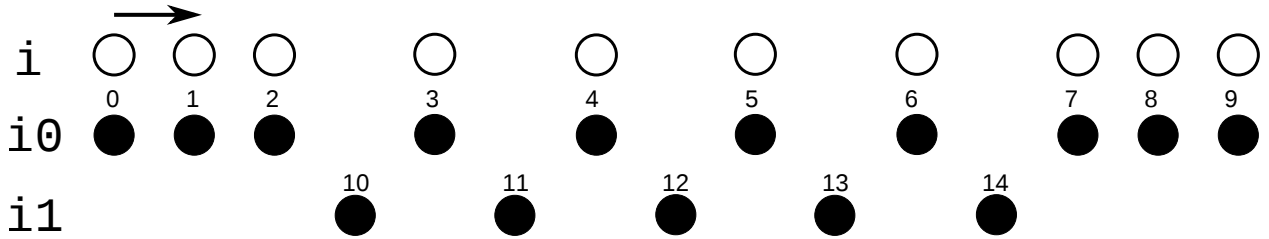
Syntaxe et sémantique

xfor (*index* = *expr*, [*index* = *expr*, ...] ;
index relop expr, [*index relop expr*, ...] ;
index+ = *incr*, [*index*+ = *incr*, ...] ;
grain, [*grain*, ...] ;
offset, [*offset*, ...]) {
label : {*statements*}
[*label* : {*statements*}...] }

Exemples : une boucle xfor

offset

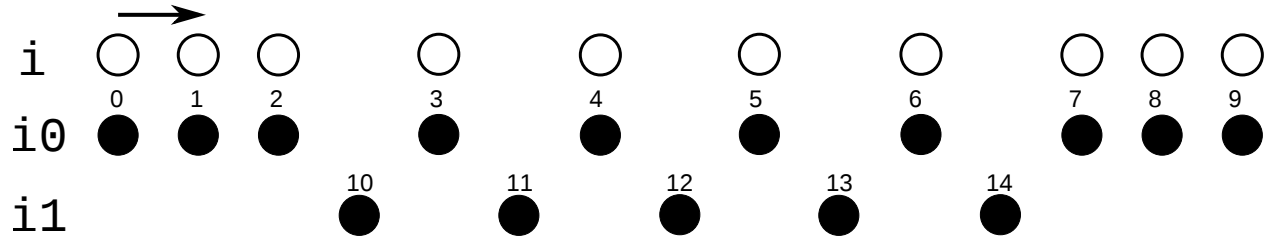
$$\textbf{xfor}\ (i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 + +, i_2 + +; 1, 1; 0, 2)$$



Exemples : une boucle xfor

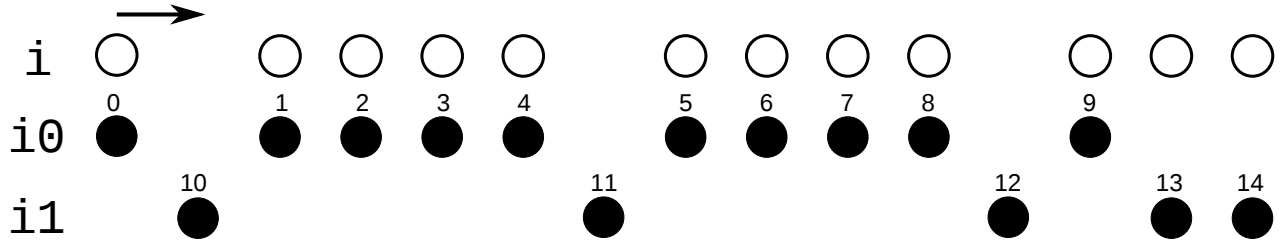
offset

xfor $(i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 ++, i_2 ++; 1, 1; 0, 2)$



grain + compression

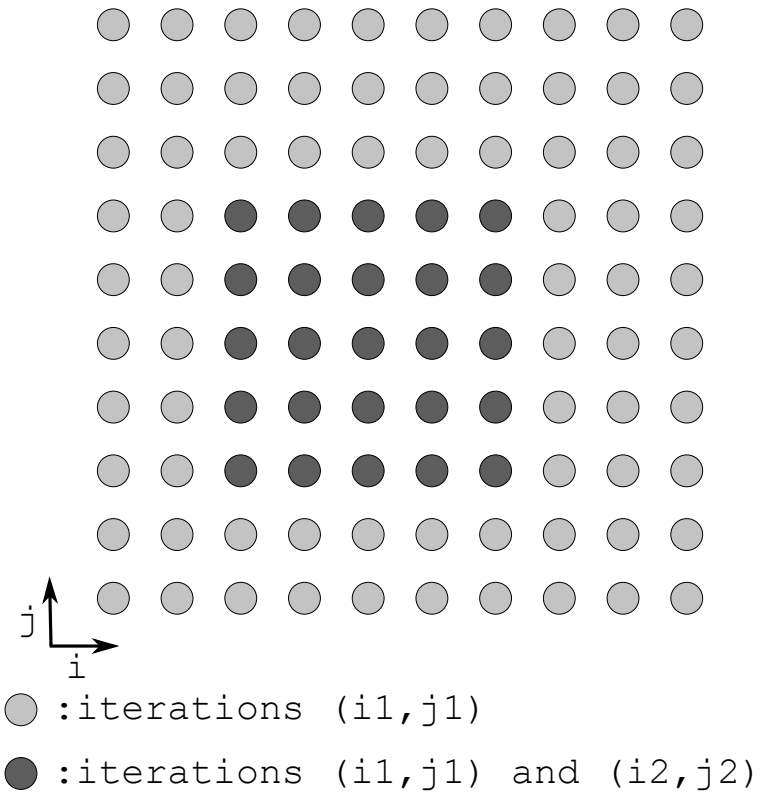
xfor $(i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 ++, i_2 ++; 1, 4; 0, 0)$



Exemples: nids de boucles xfor

offset

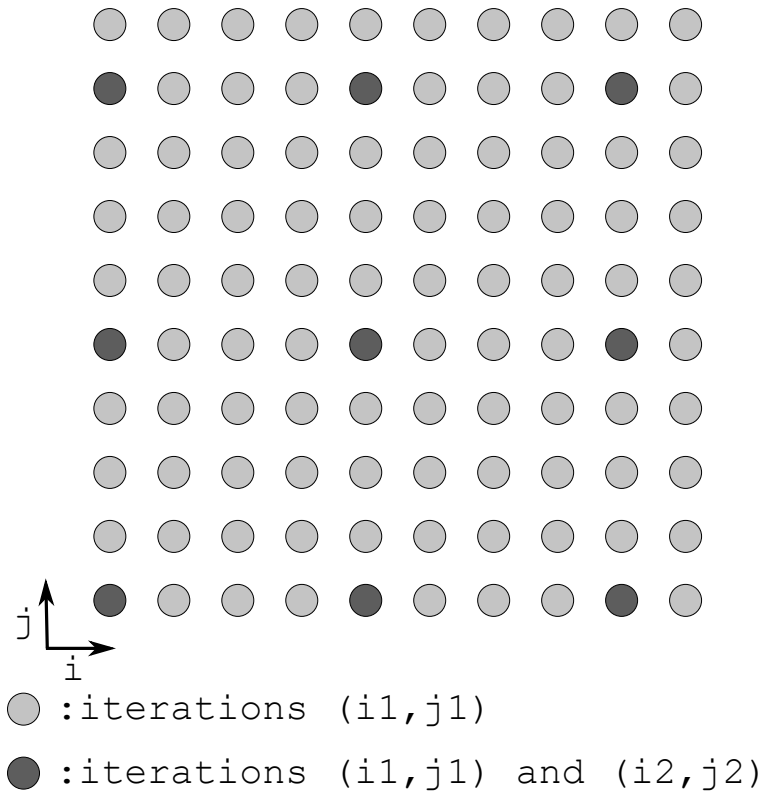
$$\textbf{xfor}\ (i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 5; i_1 + +, i_2 + +; 1, 1; 0, 2)$$
$$\textbf{xfor}\ (j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 5; j_1 + +, j_2 + +; 1, 1; 0, 2)$$



Exemples: nids de boucles xfor

grain

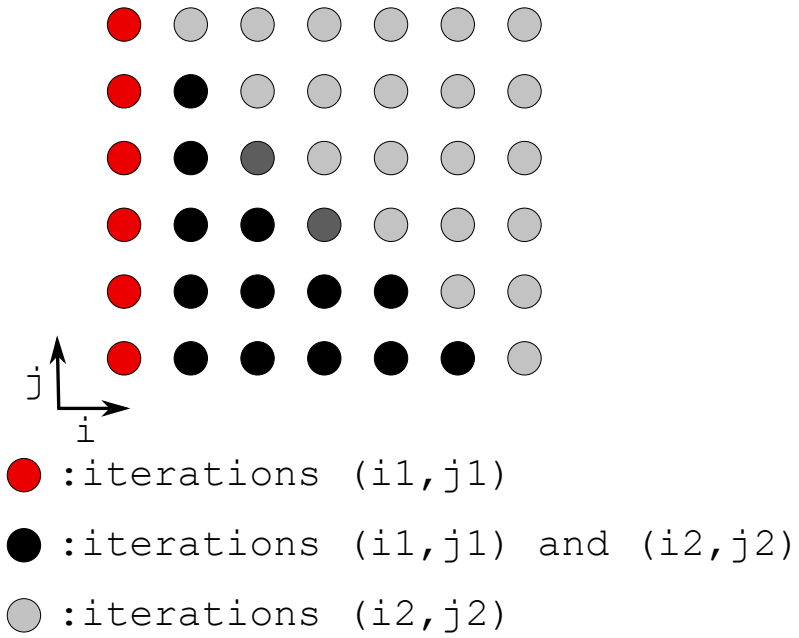
xfor $(i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 3; i_1 + +, i_2 + +; 1, 4; 0, 0)$
xfor $(j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 3; j_1 + +, j_2 + +; 1, 4; 0, 0)$



Exemples: nids de boucles xfor

affine bound + offset

xfor $(i_1 = 0, i_2 = 0; i_1 < 6, i_2 < 6; i_1 ++, i_2 ++; 1, 1; 0, 1)$
xfor $(j_1 = 0, j_2 = 0; j_1 < 6 - i_1, j_2 < 6; j_1 ++, j_2 ++; 1, 1; 0, 0)$



Compilateur XFOR : IBB (Iterate-But-Better)

- Génération d'une traduction en boucles «for» sémantiquement équivalente
  - domaines d'itération ramenés à un domaine d'indices commun
  - translations et dilatations selon les offsets et les grains
  - génération du code de parcours de l'union des domaines (CLooG)
  - code «inhumain» mais efficace
  - possibilité de directives OpenMP sur boucles xfor
- Démo sur l'exemple

Localité des données : exemple

- Code Seidel-xfor *unrolled-and-jammed* :

```
xfor (i0=1,i1=1,i2=1,i3=1,i4=1 ;
      i0<=n-2,i1<=n-2,i2<=n-2,i3<=n-2,i4<=n-2 ;
      i0+=2,i1+=2,i2+=2,i3+=2,i4+=2 ;
      1,1,1,1,1 ; /* grains */
      ?,?,?,?,? ) /* offsets */ {
xfor (j0=1,j1=1,j2=1,j3=1,j4=1 ;
      j0<=n-2,j1<=n-2,j2<=n-2,j3<=n-2,j4<=n-2 ;
      j0++,j1++,j2++,j3++,j4++ ;
      1,1,1,1,1 ; /* grains */
      ?,?,?,?,? ) /* offsets */ {

      0: { A[i0][j0] += A[i0][j0+1] ;
           A[i0+1][j0] += A[i0+1][j0+1] ; }
      1: { A[i1][j1] += A[i1+1][j1-1] ;
           A[i1+1][j1] += A[i1+2][j1-1] ; }
      2: { A[i2][j2] += A[i2+1][j2] ;
           A[i2+1][j2] += A[i2+2][j2] ; }
      3: { A[i3][j3] += A[i3+1][j3+1] ;
           A[i3+1][j3] += A[i3+2][j3+1] ; }
      4: { A[i4][j4] = (A[i4][j4]+A[i4-1][j4-1]
                       +A[i4-1][j4]+A[i4-1][j4+1]
                       +A[i4][j4-1])/9.0 ;
           A[i4+1][j4] = (A[i4+1][j4]+A[i4][j4-1]
                       +A[i4][j4]+A[i4][j4+1]
                       +A[i4+1][j4-1])/9.0 ; } }}
```

Localité des données : exemple

- 2 versions à différents offsets :
  - V_1 : (0,0,0,0,1) sur i & (0,0,0,0,0) sur j
  - V_2 : (0,0,0,0,0) sur i & (1,0,2,0,1) sur j

```
/**** V1 ****/
0: { A[i][j] += A[i][j+1] ;
     A[i+1][j] += A[i+1][j+1] ; }
1: { A[i][j] += A[i+1][j-1] ;
     A[i+1][j] += A[i+2][j-1] ; }
2: { A[i][j] += A[i+1][j] ;
     A[i+1][j] += A[i+2][j] ; }
3: { A[i][j] += A[i+1][j+1] ;
     A[i+1][j] += A[i+2][j+1] ; }
4: { A[i-1][j] = (A[i-1][j]+A[i-2][j-1]
                  +A[i-2][j]+A[i-2][j+1]
                  +A[i-1][j-1])/9.0 ;
     A[i][j]   = (A[i][j]+A[i-1][j-1]
                  +A[i-1][j]+A[i-1][j+1]
                  +A[i][j-1])/9.0 ; }
```

```
/**** V2 ****/
0: { A[i][j-1] += A[i][j] ;
     A[i+1][j-1] += A[i+1][j] ; }
1: { A[i][j] += A[i+1][j-1] ;
     A[i+1][j] += A[i+2][j-1] ; }
2: { A[i][j-2] += A[i+1][j-2] ;
     A[i+1][j-2] += A[i+2][j-2] ; }
3: { A[i][j] += A[i+1][j+1] ;
     A[i+1][j] += A[i+2][j+1] ; }
4: { A[i][j-1] = (A[i][j-1]+A[i-1][j-2]
                  +A[i-1][j-1]+A[i-1][j]
                  +A[i][j-2])/9.0 ;
     A[i+1][j-1] = (A[i+1][j-1]+A[i][j-2]
                  +A[i][j-1]+A[i][j]
                  +A[i+1][j-2])/9.0 ; }
```

- Quelle version est la plus rapide ?

Localité des données : exemple

- Exécutions sur un processeur Ivybridge Intel Core i5-3470 (3.20GHz) :

	V_1	V_2
exec. time (sec.)	1.29	3.20
cons. energy (joules)	21.17	47.41
L1 misses	120,190,581	80,150,636
L2 misses	40,090,106	40,086,200
stalled cycles	3,732,635,854	10,285,801,229

- Nombres de défauts de cache semblables
- Rapport de 2.5 en temps !
- différence significative en cycles inactifs... pourquoi ?

Localité des données : exemple

- Outil de profiling Intel VTune sur V_2 :

```
vaddsd %xmm2, %xmm3, %xmm1
lea (%r8,%rdx,1), %r11
add $0x1, %ecx                          47
vaddsd %xmm12, %xmm3, %xmm3
add $0x8, %rdx
vaddsd %xmm4, %xmm5, %xmm5
vmovsdq  %xmm1, -0x10(%rax)             41
vmovsdq  %xmm3, -0x18(%rdx)
vmovsdq  (%rax), %xmm0
vaddsd %xmm10, %xmm0, %xmm2
vmovsdq  %xmm2, -0x8(%rax)              57
vaddsd %xmm13, %xmm2, %xmm2
vmovsdq  %xmm5, -0x10(%rdx)             43
vmovsdq  (%r11,%rsi,1), %xmm8
lea (%rdi,%rax,1), %r11
add $0x8, %rax
vaddsd %xmm7, %xmm2, %xmm2              35
vaddsd %xmm8, %xmm2, %xmm2              91
vaddsd %xmm2, %xmm1, %xmm2             142
vaddsd %xmm5, %xmm1, %xmm1             126
vdivsd %xmm9, %xmm2, %xmm2
vaddsd %xmm1, %xmm2, %xmm1             889
vmovsdq  %xmm2, -0x10(%rax)            120
```

```
vaddsd %xmm1, %xmm0, %xmm1
vaddsd %xmm1, %xmm3, %xmm3             121
vaddsd %xmm6, %xmm4, %xmm1             148
vdivsd %xmm9, %xmm3, %xmm3
vaddsd %xmm3, %xmm0, %xmm0             913
vmovsdq  %xmm3, -0x10(%rdx)            155
vmovsdq  %xmm0, -0x8(%rax)
vmovsdq  (%rdx), %xmm4                  49
vmovsdq  %xmm1, -0x8(%rdx)               5
vaddsd %xmm4, %xmm0, %xmm0
vmovsdq  %xmm0, -0x8(%rax)              76
vmovsdq  0x8(%r11,%r9,1), %xmm10        43
vaddsd %xmm10, %xmm1, %xmm5              4
vmovsdq  %xmm5, -0x8(%rdx)
```

- Latences élevées
- Aléas de pipeline
- Contention sur les unités de calcul en v.f.

Localité des données : exemple

- Analyse du code source :

	accessed rows	succ. writes & reads
V_1	`i-1`	5 reads + 1 write
	`i`	3 reads + 5 writes
	`i+1`	4 reads + 4 writes
V_2	`i`	6 reads + 5 writes
	`i+1`	6 reads + 5 writes

Localité des données : conclusions

- La localité des données « n'est pas tout »
- Une « trop bonne » localité des données peut-être très pénalisante
- Les techniques d'optimisation de code, les compilateurs académiques (Pluto) et industriels (GCC) ne considèrent pas les aléas de pipeline !
- La structure XFOR permet d'ajuster la localité pour minimiser (encore plus) les latences des instructions (hors défauts de cache)
  - petites variations des valeurs des offsets, compilation itérative sur petite taille des données

Localité des données : conclusions

- Autre conséquence « étonnante » sur la consommation en puissance :
  - V_1 : $\frac{21.17 \text{ joules}}{1.29 \text{ sec.}} = 16,35$ Watts ; V_2 : $\frac{47.41 \text{ joules}}{3.20 \text{ sec.}} = 14,81$ Watts
  - $\sim$10% de consommation en puissance de moins pour V_2 !

Quelques benchmarks

Code	Pb size	Offset	Orig. time + Energy cons.	XFOR time + Energy cons.	Speed -up + Energy saving
2mm	1024	0,0 0,nj 0,0	14.592 237.204	3.498 57.379	**4.17** **75.81%**
3mm	1024	0,ni,2*ni 0,0,0 0,0,0	21.875 354.900	4.692 76.382	**4.66** **78.48%**
jacobi-2d	16K	0,1 0,1	72.990 503.272	22.937 167.256	**3.18** **66.77%**
fdtd-2d	10K	0,0,0,0 0,0,0,0	0.921 15.535	0.503 8.679	**1.83** **44.13%**
fdtd-apml	256	0,0,0,0 0,0,1,1 0,Cxm,0,Cxm	0.439 6.614	0.242 3.877	**1.81** **41.38%**
correlation	700	0,1,m+1,n+m+2 0,0,0,0 0,0,0,0	1.442 22.846	0.195 3.294	**7.39** **85.58%**
covariance	700	0,m,n+m 0,0,0 0,0,0	1.440 22.892	0.198 3.395	**7.27** **85.17%**
mvt	10K	0,0 0,0	1.570 24.743	0.222 3.625	**7.07** **85.35%**
gemver	10K	0,N,N,2N 0,0,N,0	1.857 28.849	0.490 8.070	**3.79** **72.03%**
seidel	4K	0,0,0,0,1 0,0,0,0,0	3.425 50.164	1.297 21.320	**2.64** **57.50%**

Vectorisation

- La minimisation des distances de réutilisation est un frein à la vectorisation automatique des compilateurs
  ⇒ Accroissement « suffisant » des offsets selon la taille des vecteurs et l'alignement

```
/* Jacobi 2D when no vector unit */
xfor (i0 = 1, i1 = 1 ; i0 < N-1, i1 < N-1 ;
      i0++,i1++ ; 1,1 ; 0,1)
xfor (j0 = 1, j1 = 1; j0 < N-1, j1 < N-1 ;
      j0++,j1++ ; 1,1 ; 0,1) {
 0: B[i0][j0] = 0.2 * (A[i0][j0]+A[i0][j0-1]
    +A[i0][1+j0]+A[1+i0][j0]+A[i0-1][j0]);
 1: A[i1][j1] = B[i1][j1]; }
```

```
/* Jacobi 2D when vector unit */
xfor (i0 = 1, i1 = 1 ; i0 < N-1, i1 < N-1 ;
      i0++,i1++ ; 1,1 ; 0,1)
xfor (j0 = 1, j1 = 1; j0 < N-1, j1 < N-1 ;
      j0++,j1++ ; 1,1 ; 0,6) {
 0: B[i0][j0] = 0.2 * (A[i0][j0]+A[i0][j0-1]
    +A[i0][1+j0]+A[1+i0][j0]+A[i0-1][j0]);
 1: A[i1][j1] = B[i1][j1]; }
```

Parallélisation de boucles

- La minimisation des distances de réutilisation est un frein à la parallélisation OpenMP des boucles les plus externes
  - Accroissement des offsets permettant de définir des tranches d'itérations parallèles + boucle englobante sur les tranches
  - Gain double : localité + parallélisme

```
#define k NUMBER_OF_THREADS /* Red-Black Gauss-Seidel */
for(i=1 ; i < (N-1)/2*k ; i+=2*k)
#pragma omp parallel for private(i0,i1,i2,i3,j0,j1,j2,j3) firstprivate(i) shared(u)
xfor (i0=i, i1=i+1, i2=i, i3=i+1 ;
      i0 < min(i+2*k,N-1), i1 < min(i+1+2*k,N-1),
      i2 < min(i+2*k,N-1), i3 < min(i+1+2*k,N-1) ;
      i0+=2, i1+=2, i2+=2, i3+=2 ;
      2,2,2,2 ; i-1,i,1+i+2*k,2+i+2*k)
xfor (j0=1, j1=2, j2=1, j3=2 ;
      j0 < N-1, j1 < N-1, j2 < N-1, j3 < N-1 ;
      j0+=2, j1+=2, j2+=2, j3+=2 ;
      2,2,2,2 ; 0,1,0,1) {
          0: u[i0][j0] = f(u[i0][j0+1], u[i0][j0-1],
                           u[i0-1][j0],u[i0+1][j0]);
          1: u[i1][j1] = f(u[i1][j1+1], u[i1][j1-1],
                           u[i1-1][j1],u[i1+1][j1]);
          2: u[i2][j2] = f(u[i2][j2+1], u[i2][j2-1],
                           u[i2-1][j2],u[i2+1][j2]);
          3: u[i3][j3] = f(u[i3][j3+1], u[i3][j3-1],
                           u[i3-1][j3],u[i3+1][j3]); }
```

Quelques benchmarks OpenMP

Code	Pb size	Offset	Orig. time	XFOR time	Speed -up
Red-Black	30K	0,1,1,2 0,1,0,1	1.44	0.874	**1.65**
2mm	1024	0,128,0,128 0,0,0,0 0,0,0,0	1.595	0.243	**6.62**
3mm	1024	0,ni,ni+128,0,ni,ni+128 0,0,0,0,0,0 0,0,0,0,0,0	2.41	0.45	**5.35**
jacobi-2d	16K	0,1 0,6	0.621	0.409	**1.51**
jacobi-1d	200M	0,8	0.49	0.315	**1.55**
fdtd-2d	10K	0,0,0,0 0,0,0,6	0.41	0.244	**1.68**
fdtd-apml	256	0,0,0,0 0,0,1,1 0,Cxm,0,Cxm	0.11	0.072	**1.53**
reg-detect	256 × 700	0,0,0,0,0,0 0,0,0,0,0,0 0..0,lgth-1	0.072	0.034	**2.12**
correlation	700	0,0,n,n,n,2*n,2*n, ... 2*n+1,2*n+1,3*n+1 0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0	0.31	0.07	**4.43**
covariance	700	0,0,n,n,n+1,n+1,2*n+1 0,0,0,0,0,0,0,0,0,0 0,0,0,0,0,0,0,0,0,0	0.307	0.067	**4.58**
gemver	10K	0,0,0,N-1 0,0,0,0	0.298	0.193	**1.54**

Conclusion

- XFOR : introduction de paramètres d'ordonnancement des instructions
  - Contrôle des distances de réutilisation et de la localité des données
- Relève du concept de « computer-assisted programming » (programmation du 21e siècle)
- Transformations moins générales que dans le modèle polyédrique, mais souvent suffisantes et plus efficaces
  - Comment rendre d'autres transformations accessibles (skewing) ?
- Exhibe des phénomènes précédemment peu visibles : cycles inactifs, latences des instructions, consommation en puissance
- Optimisation dans l'ère du « post-cache miss »
- Ordonnanceur XFOR : génération automatique de XFORs

MERCI

Université de Strasbourg
INRIA Nancy Grand-Est
http://team.inria.fr/camus