A study of some pitfalls preventing peak performance in polyhedral compilation using a polyhedral antidote

Philippe Clauss

Team CAMUS, INRIA, ICube Lab., CNRS, University of Strasbourg, France

IMPACT - January 19, 2015

Inria
INVENTORS FOR THE DIGITAL WORLD

caMus

The Polyhedral Model

- Advanced analysis and optimizing transformation techniques for Static Control Parts (SCoP)
  - software libraries and compilers: Pluto, ISL, PolyLib, CLooG, Candl, ...
- Speculative and dynamic adaptation of the polyhedral model for codes exhibiting a polyhedral behavior at runtime
  - VMAD, APOLLO

- Actual runtime performance of the generated codes = Uncontrolled issue!
  - heuristics used in static compilers
  - iterative and machine learning compilation frameworks: LetSee, Milepost GCC, ...
  - hardware architecture issues not handled explicitly

The XFOR loop structure

- Programming control structure assisted by an automatic code generator (IBB)
- Allows users to explicitly schedule statements of a loop nest by shifting and stretching each statement's iteration domain
- With XFOR,
  the schedule of statements is not defined by the iterator values,
  but by the *offset* (shift factor) and the *grain* (frequency factor)
- XFOR programs may often reach better performance than programs optimized by fully automatic polyhedral compilers
  - How?

5 identified performance gaps in automatic optimizers

1. **Insufficient data locality optimization**
2. **Excess of conditional branches in the generated code**
3. **Too verbose code with too many machine instructions**
4. **Data locality optimization resulting in processor stalls**
5. **Missed vectorization opportunities**

XFOR Syntax

```
xfor ( index=expr, [index=expr, ...];
       index<expr, [index<expr, ...];
       index+=cst, [index+=cst, ...];
       grain,  [grain, ...];
       offset, [offset, ...] ) {
   prefix : {statements}
}
```

where:

expr, *offset* : affine arithmetic expression.

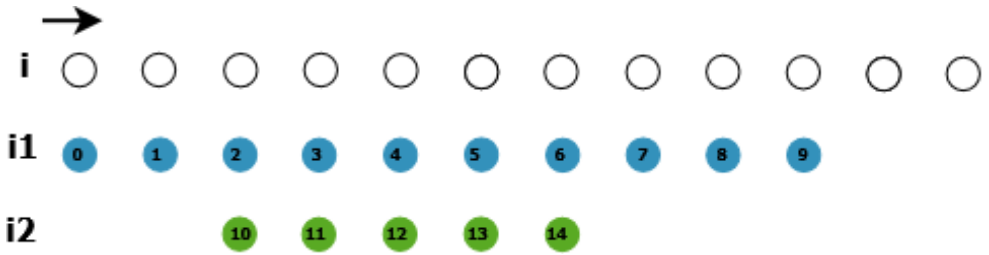
cst, *grain* : integer constant (grain ≥ 1).

prefix : positive integer associating statements to their corresponding for-loop

Examples : single XFOR loops

Offset

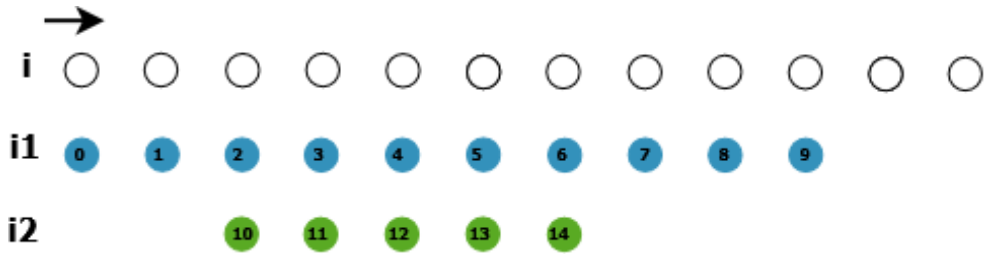
$$\textbf{xfor}\ (i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 + +, i_2 + +; 1, 1; 0, 2)$$



Examples : single XFOR loops

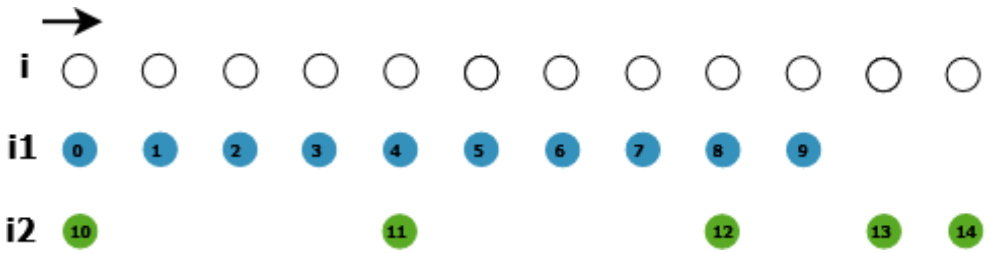
Offset

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



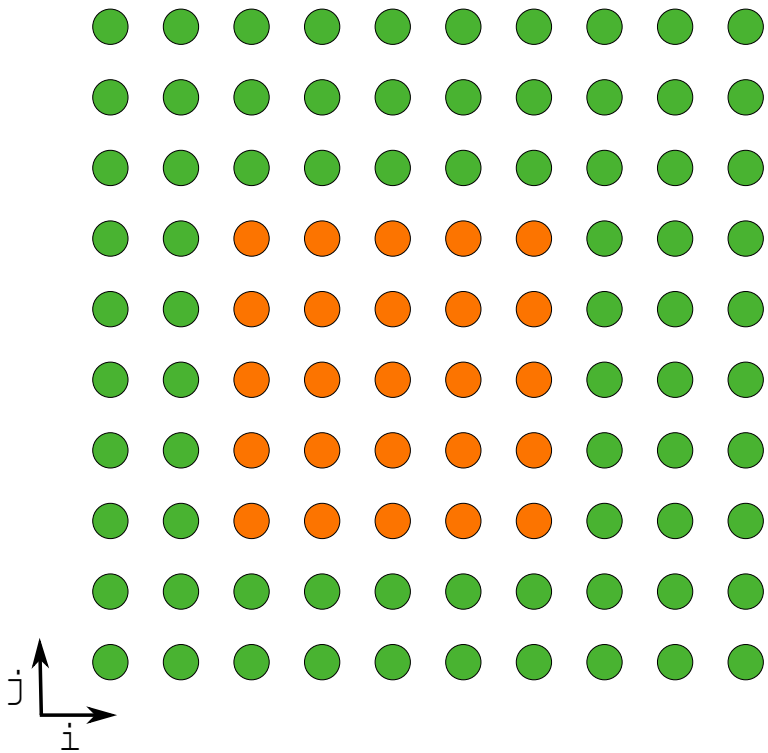
Grain + Compression

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



Examples : XFOR loop nest

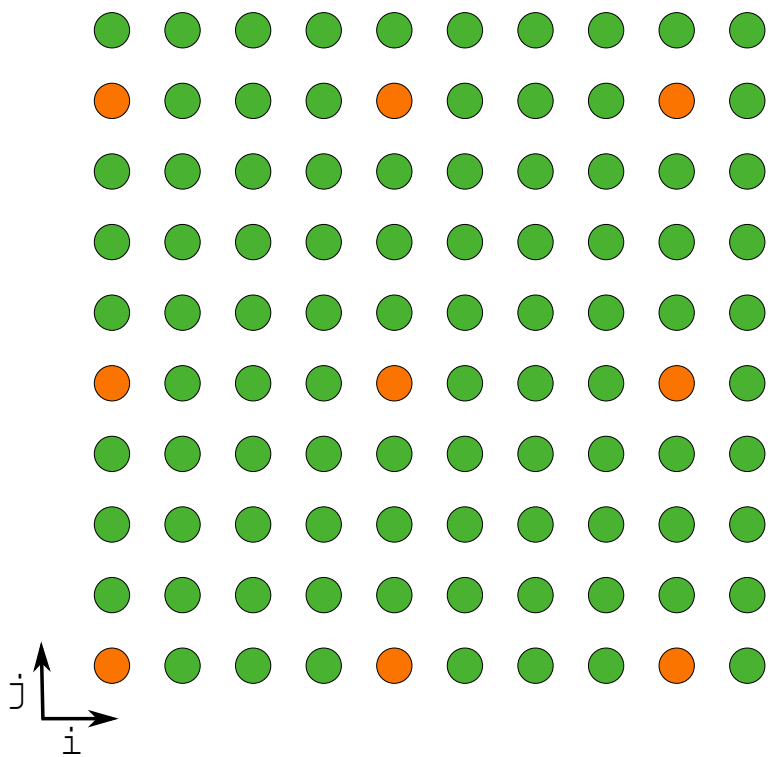
```
xfor (i1=0, i2=0 ; i1<10, i2<5 ;
      i1++, i2++ ; 1, 1 ; 0, 2)
xfor (j1=0, j2=0 ; j1<10, j2<5 ;
      j1++, j2++ ; 1, 1 ; 0, 2)
```



:itérations (i1,j1)

:itérations (i1,j1) and (i2,j2)

```
xfor (i1=0, i2=0 ; i1<10, i2<3 ;
      i1++, i2++ ; 1, 4 ; 0, 0)
xfor (j1=0, j2=0 ; j1<10, j2<3 ;
      j1++, j2++ ; 1, 4 ; 0, 0)
```



:itérations (i1,j1)

:itérations (i1,j1) and (i2,j2)

XFOR compiler: IBB (Iterate-But-Better), *Imen Fassi*

- Translation in a program of for-loops that are semantically equivalent
- Iteration domains reduced into one common iteration domain
- Shifts and dilatations applied according to offsets and grains
- Generation of the xfor-equivalent for-code scanning the union of domains by using CLooG
- Inhuman for-code but efficient
- OpenMP directives allowed with xfor loops (`omp [parallel] for`)

Highlighting the gaps

- Comparisons between xfor codes and Pluto-generated codes
  - Pluto's best performing codes among the use of options -tile (default size 32), -l2tile, -smartfuse, -maxfuse, -rar
- Comparisons between different versions of xfor codes
- Codes compiled using GCC 4.8.1 with options O3 and march=native
- CPU events collected using `perf` and `libpfm`

Collected CPU events

#CPU cycles:	number of CPU cycles, halted and unhalted.
#L1 data loads:	number of data references to the L1 cache.
#Li misses:	number of loads that miss the Li cache.
#TLB misses:	number of load misses in the TLB that cause a page walk.
#branches:	number of retired branch instructions.
#branch misses:	number of branch mispredictions.
#Stalled cycles:	number of cycles in which no micro-operations are executed on any port.
#Resource related stalls:	number of allocator resource related stalls.
#Reservation Station stalls:	number of cycles when the number of instructions in the pipeline waiting for execution reaches the limit. Exhibits the effect of long chains of dependences between close instructions.
#Re-Order Buffer stalls:	number of cycles when the number of instructions in the pipeline waiting for retirement reaches the limit. Exhibits the effect of long latency memory operations and TLB or cache misses.
#instructions:	number of retired instructions.

Gap 1: Insufficient data locality optimization

	Pluto	XFOR	Ratios
mvt			
#CPU cycles	3,824M	2,425M	-36.58%
#L1 data loads	748M	451M	-39.71%
#L1 misses	45M	50M	+10.71%
#L2 misses	**29M**	**5.8M**	**-80.09%**
#L3 misses	**38M**	**14M**	**-63.77%**
#TLB misses	**3.8M**	**0.7M**	**-82.62%**
#branches	224M	212M	-4.89%
#branch misses	470K	439K	-6.58%
#instructions	2,469M	2,010M	-18.58%
syr2k			
#CPU cycles	7,005M	5,671M	-19.05%
#L1 data loads	4,322M	2,158M	-50.06%
#L1 misses	**299M**	**137M**	**-54.18%**
#L2 misses	**8.4M**	**3.6M**	**-55.94%**
#L3 misses	**10M**	**5.1M**	**-48.57%**
#TLB misses	**4.3M**	**3.2M**	**-25.78%**
#branches	1,072M	1,078M	+0.58%
#branch misses	1,072K	1,084K	+1.03%
#instructions	11,890M	13,946M	+17.29%

	Pluto	XFOR	Ratios
3mm			
#CPU cycles	17,557M	4,358M	-75.18%
#L1 data loads	4,226M	2,440M	-24.36%
#L1 misses	**815M**	**206M**	**-74.67%**
#L2 misses	**554M**	**5.4M**	**-99.02%**
#L3 misses	**174M**	**3M**	**-98.25%**
#TLB misses	**541M**	**3.2M**	**-99.41%**
#branches	1,625M	813M	-49.96%
#branch misses	2,704K	1,630K	-39.73%
#instructions	11,331M	8,941M	-21.09%
gauss-filter			
#CPU cycles	3,457M	2,963M	-14.28%
#L1 data loads	873M	843M	-3.45%
#L1 misses	**75M**	**46M**	**-38.97%**
#L2 misses	**4.2M**	**2.4M**	**-42.33%**
#L3 misses	**29.5M**	**24.8M**	**-15.91%**
#TLB misses	**1.5M**	**0.7M**	**-49.78%**
#branches	724M	572M	-20.92%
#branch misses	622K	689K	+10.78%
#instructions	5,026M	4,652M	-7.44%

Gap 1: Insufficient data locality optimization

	Pluto	XFOR	Ratios
mvt - #stalled cycles	**2,742M**	**1,582M**	**-42.29%**
#Resource related stalls	**2,544M**	**1,347M**	**-47.05%**
#Reservation Station stalls	431M	447M	+3.63%
#Re-Order Buffer stalls	**2,008M**	**771M**	**-61.62%**
syr2k - #stalled cycles	**1,570M**	**1,346M**	**-14.27%**
#Resource related stalls	**1,495M**	**1,332M**	**-10.91%**
#Reservation Station stalls	327M	1,199M	+266.50%
#Re-Order Buffer stalls	**1,182M**	**132M**	**-88.80%**
3mm - #stalled cycles	**12,695M**	**524M**	**-95.87%**
#Resource related stalls	**12,392M**	**387M**	**-96.87%**
#Reservation Station stalls	**10,667M**	**379M**	**-96.44%**
#Re-Order Buffer stalls	**2,606M**	**38M**	**-98.52%**
gauss-filter - #stalled cycles	**1,351M**	**1,196M**	**-11.45%**
#Resource related stalls	**924M**	**824M**	**-10.82%**
#Reservation Station stalls	174M	150M	-13.88%
#Re-Order Buffer stalls	**171M**	**134M**	**-21.25%**

Gap 1: Insufficient data locality optimization - **mvt**

req: intra-statement + inter-statement data locality

```
/* Original code */
for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
  x1[i] = x1[i] + A[i][j] * y_1[j];
for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
  x2[i] = x2[i] + A[j][i] * y_2[j];

/* Pluto code */
for (t1=0;t1<=floord(n-1,32);t1++) {
 for (t2=0;t2<=floord(n-1,32);t2++) {
  for (t3=32*t1;t3<=min(n-1,32*t1+31);t3++) {
   for (t4=32*t2;t4<=min(n-1,32*t2+31);t4++) {
    x1[t3] = x1[t3] + A[t3][t4] * y_1[t4];
    x2[t3] = x2[t3] + A[t4][t3] * y_2[t4];}}}}

/* XFOR code: interchange + fusion */
xfor (i0=0, j1=0 ; i0<n, j1<n ; i0++, j1++ ; 1, 1 ; 0, 0) {
xfor (j0=0, i1=0 ; j0<n, i1<n ; j0++, i1++ ; 1, 1 ; 0, 0) {
 0 : x1[i0] = x1[i0] + A[i0][j0] * y_1[j0];
 1 : x2[i1] = x2[i1] + A[j1][i1] * y_2[j1];}}
```

Gap 1: Insufficient data locality optimization - **syr2k**

req: intra-statement + inter-statement data locality

```
/* Original & Pluto code */
for (i = 0; i < n; i++)
 for (j = 0; j < n; j++)
  for (k = 0; k < m; k++)
  {
   C[i][j] += alpha * A[i][k] * B[j][k];
   C[i][j] += alpha * B[i][k] * A[j][k];
  }

/* XFOR code: splitting + interchange + fusion */
xfor (i0=0, j1=0 ; i0<n, j1<n ; i0++, j1++ ; 1,1 ; 0,0 ) {
 xfor (j0=0, i1=0 ; j0<n, i1<n ; j0++, i1++ ; 1,1 ; 0,0 ) {
  0: temp0 = 0.0 ;
  1: temp1 = 0.0 ;
  xfor (k0=0,k1=0 ; k0<m,k1<m ; k0++,k1++ ; 1,1 ; 0,0 ) {
   0: temp0 += alpha * A[i0][k0] * B[j0][k0];
   1: temp1 += alpha * B[i1][k1] * A[j1][k1]; }
  0: C[i0][j0] += temp0 ;
  1: C[i1][j1] += temp1 ;
  }
}
```

Gap 2: Excess of conditional branches

- Tiling may be more penalizing than advantageous!
  - many additional loop levels and complex loop bounds made with combinations of *min*, *max*, *floor* and *ceiling* functions invocations
  - many more branches in the final generated code
  - more machine instructions

```
/* Pluto-Seidel tiled loop nest */
for (t1=0;t1<=floord(tsteps-1,32);t1++)
 for (t2=t1;t2<=min(floord(32*t1+n+29,32),
                    floord(tsteps+n-3,32));t2++)
  for (t3=max(ceild(64*t2-n-28,32),t1+t2);
       t3<=min(min(min(min(floord(32*t1+n+29,16),
                           floord(tsteps+n-3,16)),
                       floord(64*t2+n+59,32)),
                   floord(32*t1+32*t2+n+60,32)),
               floord(32*t2+tsteps+n+28,32));t3++)
   for (t4=max(max(max(32*t1,32*t2-n+2),16*t3-n+2),-32*t2+32*t3-n-29);
        t4<=min(min(min(min(32*t1+31,32*t2+30),16*t3+14),tsteps-1),
                -32*t2+32*t3+30);t4++)
    for (t5=max(max(32*t2,t4+1),32*t3-t4-n+2);
         t5<=min(min(32*t2+31,32*t3-t4+30),t4+n-2);t5++)
     for (t6=max(32*t3,t4+t5+1);t6<=min(32*t3+31,t4+t5+n-2);t6++) {
      A[-t4+t5][-t4-t5+t6] = ...;
```

Gap 2: Excess of conditional branches

	Pluto	XFOR	Ratios
seidel			
#CPU cycles	15,721M	7,476M	-52.45%
#L1 data loads	3,099M	672M	-78.31%
#L1 misses	12M	83M	+569.40%
#L2 misses	3.7M	1.2M	-65.64%
#L3 misses	3.9M	3.4M	-12.69%
#TLB misses	78K	688K	+783.18%
#branches	**387M**	**179M**	**-53.88%**
#branch misses	**456K**	**132K**	**-70.97%**
#stalled cycles	**11,297M**	**4,499M**	**-60.18%**
#RR stalls	**11,030M**	**4,4281M**	**-59.85%**
#RS stalls	**3,017M**	**440M**	**-85.39%**
#ROB stalls	**9,466M**	**3,982M**	**-57.93%**
#instructions	**10,015M**	**7,857M**	**-21.55%**
correlation			
#CPU cycles	425M	426M	+0.22%
#L1 data loads	224M	186M	-17.10%
#L1 misses	3.7M	12M	+223.95%
#L2 misses	2.2M	1M	-50.77%
#L3 misses	635K	395K	-37.83%
#TLB misses	294K	306K	+4.27%
#branches	**120M**	**78M**	**-34.39%**
#branch misses	**549K**	**231K**	**-58.01%**
#stalled cycles	**115M**	**47M**	**-58.79%**
#RR stalls	**81M**	**24M**	**-69.49%**
#RS stalls	**47M**	**3.7M**	**-92.10%**
#ROB stalls	**16M**	**14M**	**-13.31%**
#instructions	**906M**	**934M**	**+3.03%**

	Pluto	XFOR	Ratios
covariance			
#CPU cycles	419M	320M	-23.71%
#L1 data loads	217M	117M	-46.19%
#L1 misses	3.5M	22M	+539%
#L2 misses	1.9M	9M	+366.65%
#L3 misses	744K	496K	-33.42%
#TLB misses	247K	501K	+102.87%
#branches	**119M**	**35M**	**-70.40%**
#branch misses	**721K**	**199K**	**-72.37%**
#stalled cycles	**61M**	**123M**	**+100.75%**
#RR stalls	**59M**	**117M**	**+98.54%**
#RS stalls	**44M**	**43M**	**-1.40%**
#ROB stalls	**17M**	**75M**	**+344.54%**
#instructions	**1,050M**	**506M**	**-51.86%**

More L1 & TLB misses,
but faster!

Gap 3: Number of instructions

jacobi-2d	Pluto	XFOR1	XFOR2
#CPU cycles	12,136M	13,700M	12,641M
#L1 data loads	1,400M	1,530M	1,529M
#L1 misses	236M	206M	205M
#L2 misses	44M	6M	11M
#L3 misses	76M	68M	68M
#TLB misses	2.7M	2.8M	3M
#branches	657M	564M	650M
#branch misses	1,560K	1,448K	1,329K
#stalled cycles	9,265M	9,463M	8,673M
#Resource related stalls	8,317M	8,433M	7,606M
#Reservation Station stalls	1,123M	1,088	930M
#Re-Order Buffer stalls	5,435M	4,775M	4,740M
#instructions	**6,950M**	**9,370M**	**10,469M**

More cache misses,
but less instructions and faster!

Gap 4: Unaware data locality optimization

- 3 xfor code versions of the polybench `seidel` code which just differ by their offset values

```
for (t = 0 ; t <= tsteps-1 ; t++)
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 ;
}}}
```

Gap 4: Unaware data locality optimization

seidel	XFOR1	XFOR2	XFOR3
offsets-i	0,0,0,0,1	0,1,0,0,1	0,1,1,1,1
offsets-j	0,0,0,0,0	0,0,0,0,0	0,0,0,0,0
#CPU cycles	7,392M	11,393M	12,283M
#L1 data loads	986M	997M	837M
#L1 misses	123M	123M	103M
#L2 misses	1.9M	1.9M	1.6M
#L3 misses	3.5M	3.5M	3.5M
#TLB misses	725K	694K	693K
#branches	97M	94M	96M
#branch misses	74K	78K	78K
#stalled cycles	**5,100M**	**8,002M**	**9,367M**
#Resource related stalls	**5,076M**	**7,969M**	**9,334M**
#Reservation Station stalls	**1,543M**	**7,765M**	**9,130M**
#Re-Order Buffer stalls	**3,537M**	**170M**	**157M**
#instructions	6,131M	7,146M	6,503M

Why more stalled cycles?

Gap 4: Unaware data locality optimization - Intel VTune

XFOR1	ms
addsd %xmm7, %xmm0	
addsd %xmm1, %xmm0	44
divsd %xmm3, %xmm0	
movsdq %xmm0, -0x8(%r8)	8
movsdq -0x8(%rcx), %xmm2	
movsdq (%r9), %xmm13	72
addsd %xmm1, %xmm2	
movapd %xmm13, %xmm1	
addsd %xmm9, %xmm1	
addsd %xmm0, %xmm2	12
addsd %xmm7, %xmm1	
addsd %xmm13, %xmm2	
addsd %xmm5, %xmm2	
divsd %xmm3, %xmm2	20
movsdq %xmm2, -0x8(%rcx)	796
movsdq (%rax), %xmm11	28

XFOR2	ms
addsd %xmm11, %xmm2	
addsd %xmm0, %xmm2	70
addsd %xmm4, %xmm0	108
divsd %xmm3, %xmm2	
movsdq %xmm2, (%rdi)	542
addsd %xmm2, %xmm0	48
movsdq 0x8(%r9), %xmm9	64
addsd %xmm9, %xmm0	
addsd %xmm1, %xmm0	40
movapd %xmm10, %xmm1	78
divsd %xmm3, %xmm0	
movsdq %xmm0, (%rax)	526
movsdq 0x8(%rcx), %xmm4	40

XFOR3	ms
addsd %xmm9, %xmm0	28
addsd %xmm7, %xmm0	
addsd %xmm8, %xmm0	60
divsd %xmm3, %xmm0	48
movsdq %xmm0, -0x8(%rcx)	602
addsd %xmm0, %xmm1	20
movsdq (%r9), %xmm2	124
addsd %xmm2, %xmm1	
addsd %xmm13, %xmm1	96
divsd %xmm3, %xmm1	42
addsd %xmm1, %xmm2	824
movsdq %xmm1, -0x8(%rdx)	74

Gap 4: Unaware data locality optimization

```
/**** XFOR1 ****/
/* PREVIOUS ITERATION */
01  A[i][j-1] += A[i][j];
02  A[i+1][j-1] += A[i+1][j];
03  A[i][j-1] += A[i+1][j-2];
04  A[i+1][j-1] += A[i+2][j-2];
05  A[i][j-1] += A[i+1][j-1];
06  A[i+1][j-1] += A[i+2][j-1];
07  A[i][j-1] += A[i+1][j];
08  A[i+1][j-1] += A[i+2][j];
09  A[i-1][j-1] = (A[i-1][j-1] + A[i-2][j-2] + A[i-2][j-1] + A[i-2][j] + A[i-1][j-2])/9.0;
10  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;

/* CURRENT ITERATION */
11  A[i][j] += A[i][j+1];
12  A[i+1][j] += A[i+1][j+1];
13  A[i][j] += A[i+1][j-1];
14  A[i+1][j] += A[i+2][j-1];
15  A[i][j] += A[i+1][j];
16  A[i+1][j] += A[i+2][j];
17  A[i][j] += A[i+1][j+1];
18  A[i+1][j] += A[i+2][j+1];
19  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;
20  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;
```

```
/**** XFOR2 ****/
/* PREVIOUS ITERATION */
01  A[i][j-1] += A[i][j];
02  A[i+1][j-1] += A[i+1][j];
03  A[i-1][j-1] += A[i][j-2];
04  A[i][j-1] += A[i+1][j-2];
05  A[i][j-1] += A[i+1][j-1];
06  A[i+1][j-1] += A[i+2][j-1];
07   A[i][j-1] += A[i+1][j];
08  A[i+1][j-1] += A[i+2][j];
09  A[i-1][j-1] = (A[i-1][j-1] + A[i-2][j-2] + A[i-2][j-1] + A[i-2][j] + A[i-1][j-2])/9.0;
10  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; */

/* CURRENT ITERATION */
11  A[i][j] += A[i][j+1];
12  A[i+1][j] += A[i+1][j+1]; }
13  A[i-1][j] += A[i][j-1];
14  A[i][j] += A[i+1][j-1];
15  A[i][j] += A[i+1][j];
16  A[i+1][j] += A[i+2][j]; }
17  A[i][j] += A[i+1][j+1];
18  A[i+1][j] += A[i+2][j+1]; }
19  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;
20  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;
```

```
/**** XFOR3 ****/
/* PREVIOUS ITERATION */
01  A[i][j-2] += A[i][j-1];
02  A[i+1][j-2] += A[i+1][j-1];
03  A[i-1][j-2] += A[i][j-3];
04  A[i][j-2] += A[i+1][j-3];
05  A[i-1][j-2] += A[i][j-2];
06  A[i][j-2] += A[i+1][j-2];
07  A[i-1][j-3] += A[i][j-2];
08  A[i][j-3] += A[i+1][j-2];
09  A[i-1][j-3] = (A[i-1][j-3] + A[i-2][j-4] + A[i-2][j-3] + A[i-2][j-2] + A[i-1][j-4])/9.0;
10  A[i][j-3] = (A[i][j-3] + A[i-1][j-4] + A[i-1][j-3] + A[i-1][j-2] + A[i][j-4])/9.0; */

/* CURRENT ITERATION */
11  A[i][j-1] += A[i][j];
12  A[i+1][j-1] += A[i+1][j];
13  A[i-1][j-1] += A[i][j-2];
14  A[i][j-1] += A[i+1][j-2];
15  A[i-1][j-1] += A[i][j-1];
16  A[i][j-1] += A[i+1][j-1];
17  A[i-1][j-2] += A[i][j-1];
18  A[i][j-2] += A[i+1][j-1]; }
19  A[i-1][j-2] = (A[i-1][j-2] + A[i-2][j-3] + A[i-2][j-2] + A[i-2][j-1] + A[i-1][j-3])/9.0;
20  A[i][j-2] = (A[i][j-2] + A[i-1][j-3] + A[i-1][j-2] + A[i-1][j-1] + A[i][j-3])/9.0;
```

Gap 5: Insufficient handling of vectorization opportunities

	Pluto	XFOR	Ratios
jacobi-1d			
#CPU cycles	9,711M	9,063M	-6.67%
#L1 data loads	895M	885M	-0.03%
#L1 misses	110M	110M	-0.53%
#L2 misses	4M	4.7M	+16.78%
#L3 misses	54M	57M	+5.34%
#TLB misses	2.3M	2M	-15.51%
#branches	508M	505M	-0.48%
#branch misses	1,031K	1,174K	+13.91%
#stalled cycles	7,465M	6,844M	-8.32%
#instructions	4,891M	4,924M	+0.69%
fdtd-2d			
#CPU cycles	7,631M	5,679M	-25.58%
#L1 data loads	950M	962M	1.25%
#L1 misses	130M	114M	-12.29%
#L2 misses	5.6M	11.3M	+103.02%
#L3 misses	39M	32M	-18.81%
#TLB misses	1.8M	1.4M	-25.64%
#branches	345M	249M	-27.85%
#branch misses	755K	636K	-15.79%
#stalled cycles	5,844M	3,871M	-33.77%
#instructions	3,936M	4,427M	+12.46%

	Pluto	XFOR	Ratios
fdtd-apml			
#CPU cycles	2,969M	1,871M	-36.96%
#L1 data loads	360M	333M	-7.56%
#L1 misses	27M	30M	+10.85%
#L2 misses	971K	1,127K	+16.11%
#L3 misses	9.6M	9.2M	-3.55%
#TLB misses	710K	925K	+30.31%
#branches	97M	81M	-17%
#branch misses	476K	572K	+20.31%
#stalled cycles	2,196M	1,190M	-45.81%
#instructions	1,581M	1,448M	-8.46%

Gap 5: Insufficient handling of vectorization opportunities - **jacobi-1d**

```
/* Pluto code: reuse distance = 1 */
B[2] = 0.33333 * (A[1] + A[2] + A[3]);
for (t1=3;t1<=n-2;t1++) {
 B[t1] = 0.33333 * (A[t1-1] + A[t1] + A[t1 + 1]);
 A[t1-1] = B[t1-1]; }
A[n-2] = B[n-2];
```

```
/* XFOR code: reuse distance = 9  */
xfor(j0=2,j1=2;j0<n-1,j1<n-1;j0++,j1++;1,1;0,9) {
 0 : B[j0] = 0.33333 * (A[j0-1] + A[j0] + A[j0+1]);
 1 : A[j1] = B[j1]; }
```

Bridging the gaps?

- Source codes should be written by programmers in a form that is the simplest for the compiler

+ Compilers should generate codes that are the simplest for the **microprocessor**

- **Data locality is not an isolated issue** and must be careful of the other four issues: *excessive number of branches, instruction counts, long chains of short RAW dependences, vectorization*
- **Inter-statement data locality** is as important as intra-statement data locality, but must be careful of vectorization
- The grain of reasoning should be **the memory access**
- **Tiling is not always the best answer** to improve data locality *excessive number of branches, instruction count*

Conclusion

- The xfor structure is a polyhedral antidote to help addressing these gaps, until the perfect compiler and microprocessor have been developed, if they ever will be in the future

- The post-data-locality era of polyhedral optimization has started!