Multifor for Multicore

Imèn Fassi [a], Philippe Clauss [b], Matthieu Kuhn [c], Yosr Slama [a]

[a] Dpt of Computer Science, Faculty of Sciences, University El Manar, Tunisia
[b] Team CAMUS, INRIA, University of Strasbourg, France
[c] Team ICPS, ICube lab., University of Strasbourg, France

Why a Multifor construct?

Why a Multifor construct?

- Parallelism must naturally take part of the programming process
  - programming languages:
    - many new languages are or have been proposed, many have disappeared or are going to disappear
    - current successful languages can be extended

Why a Multifor construct?

- Parallelism must naturally take part of the programming process
  - programming languages:
    - many new languages are or have been proposed, many have disappeared or are going to disappear
    - current successful languages can be extended
  - code optimization and parallelization:
    - standard developers have to be raised to *20 years ago experienced programmers*
    - as they learned the use of functions, recursion, object programming, ... they should learn data layout optimization, simple loop transformations, mapping of iteration/data domains, ...
    - but without being forced to (optional constructs vs specific languages)

Why a Multifor construct?

- Parallelism must naturally take part of the programming process
  - programming languages:
    - many new languages are or have been proposed, many have disappeared or are going to disappear
    - current successful languages can be extended
  - code optimization and parallelization:
    - standard developers have to be raised to *20 years ago experienced programmers*
    - as they learned the use of functions, recursion, object programming, ...
      they should learn data layout optimization, simple loop transformations, mapping of iteration/data domains, ...
    - but without being forced to
      (optional constructs vs specific languages)
  - hardware/software support mechanisms for parallel programming cannot solve all parallel programming issues, when they do not even add some more problems (TM, VM, etc.)

Why a Multifor construct?

- The Polytope Model
  - most of its features are hidden to developers (automatic parallelization)
  - polyhedral transformations result often in (efficient but) unreadable code
  - the model's scope is not limited to a sequence of loop nests, and can be applied incrementally
  - *polyhedral programming* can promote the model and improve its efficiency

Why a Multifor construct?

- The Polytope Model
  - most of its features are hidden to developers (automatic parallelization)
  - polyhedral transformations result often in (efficient but) unreadable code
  - the model's scope is not limited to a sequence of loop nests, and can be applied incrementally
  - *polyhedral programming* can promote the model and improve its efficiency
- Multifor
  - a polyhedral programming control structure, providing a polyhedral view of the computation
  - facilitates the expression of some task parallelism, dataflow and MapReduce schemes
  - allows developers to express some loop fusion, mapping of domains, data reuse, ...

Syntax and semantics

$$
\begin{array}{ll}
\textbf{multifor} \ (& index_1 = expr, [index_2 = expr, ...] \ ; \\
& index_1 < expr, [index_2 < expr, ...] \ ; \\
& index_1 += cst, [index_2 += cst, ...] \ ; \\
& grain_1, [grain_2, ...] \ ; \\
& offset_1, [offset_2, ...] \ ) \ \ \{ \\
& prefix : \{statements\} \\
\} &
\end{array}
$$

where

- *expr*: affine arithmetic expressions on enclosing loop indices
- *cst*, *grain* and *offset*: integer constants
- $grain \geq 1, offset \geq 0$
- *prefix*: positive integer associating statements to their corresponding for-loop

Syntax and semantics

- Each for-loop composing the multifor-loop behaves as a traditional for-loop

Syntax and semantics

- Each for-loop composing the multifor-loop behaves as a traditional for-loop
- Every iteration domain is mapped on a same referential iteration domain, according its grain and offset
  - referential domain: union of the for-loop domains, dilated and shifted following their respective grain and offset
  - grain: frequency in which the loop is run, gcd of the grains of the overlapping for-loops per sub-domain (compression factor)
  - offset: gap between the first iteration of the referential domain and the first iteration of the loop

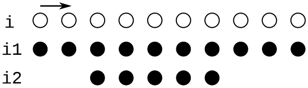
Syntax and semantics

- Each for-loop composing the multifor-loop behaves as a traditional for-loop
- Every iteration domain is mapped on a same referential iteration domain, according its grain and offset
  - referential domain: union of the for-loop domains, dilated and shifted following their respective grain and offset
  - grain: frequency in which the loop is run, gcd of the grains of the overlapping for-loops per sub-domain (compression factor)
  - offset: gap between the first iteration of the referential domain and the first iteration of the loop
- On overlapping for-loops iteration domains, respective iterations are run in any interleaved fashion or in parallel.

Examples: one multifor-loop

offset

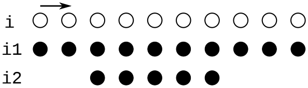
$$\textbf{multifor}\ (i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 + +, i_2 + +; 1, 1; 0, 2)$$



Examples: one multifor-loop

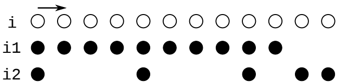
offset

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



grain + compression

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



Nested multifor-loops

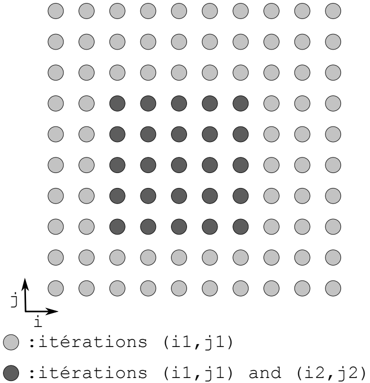
multifor ($index_1 = expr, index_2 = expr;$
$index_1 < expr, index_2 < expr;$
$index_1 += cst, index_2 += cst;$
$grain_1, grain_2; offset_1, offset_2$) {
$prefix : \{statements\}$
multifor ($index_3 = expr, index_4 = expr;$
$index_3 < expr, index_4 < expr;$
$index_3 += cst, index_4 += cst;$
$grain_3, grain_4; offset_3, offset_4$) {
$prefix : \{statements\}$
}
$prefix : \{statements\}$
}

- behaves as 2 for-loop nests $(index_1, index_3)$ and $(index_2, index_4)$
- the bounds are affine functions of the enclosing loop indices of the same for-loop

Examples: nested multifor-loops

offset

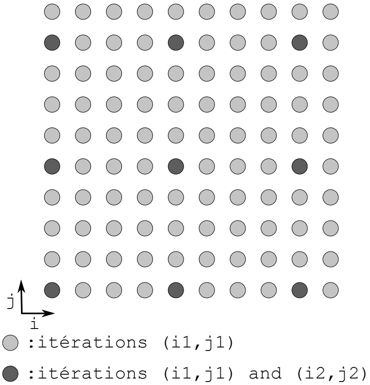
$$\textbf{multifor}\ (i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 5; i_1++, i_2++; 1, 1; 0, 2)$$
$$\textbf{multifor}\ (j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 5; j_1++, j_2++; 1, 1; 0, 2)$$



Examples: nested multifor-loops

grain

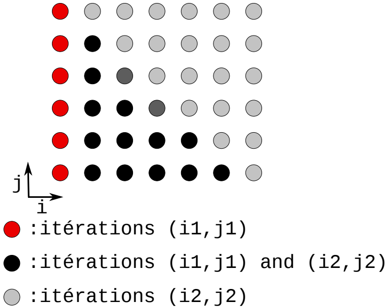
$$\textbf{multifor}\ (i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 3; i_1{+}{+}, i_2{+}{+}; 1, 4; 0, 0)$$
$$\textbf{multifor}\ (j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 3; j_1{+}{+}, j_2{+}{+}; 1, 4; 0, 0)$$



Examples: nested multifor-loops

affine bound + offset

$$\textbf{multifor}\ (i_1 = 0, i_2 = 0; i_1 < 6, i_2 < 6; i_1{+}{+}, i_2{+}{+}; 1, 1; 0, 1)$$
$$\textbf{multifor}\ (j_1 = 0, j_2 = 0; j_1 < 6 - i_1, j_2 < 6; j_1{+}{+}, j_2{+}{+}; 1, 1; 0, 0)$$



Multifor-loop parallelization and transformation

- Parallelization opportunities:
  - Running each for-loop as a separated thread
  - Parallelizing each for-loop in an OpenMP fashion
  - Parallelizing simultaneously in both ways
- Polyhedral transformations:
  - Of each for-loop
  - With a global view regarding their interactions (referential domain)

Another way of writing loop nests

- Imperfect nests:

$$\textbf{for } (i = 0; i < 10; i++)$$
$$\quad inst\_block_1$$
$$\quad \textbf{for } (j = 0; j < 10; j++)$$
$$\qquad inst\_block_2$$

or

$$\textbf{multifor } (i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 10; i_1++, i_2++; 1, 1; 0, 0)$$
$$\quad \textbf{multifor } (j_1 = 0, j_2 = 0; j_1 < 1, j_2 < 10; j_1++, j_2++; 1, 1; 0, 1)$$
$$\qquad 0 : inst\_block_1$$
$$\qquad 1 : inst\_block_2$$

Another way of writing loop nests

- Re-scheduling some statements, e.g. for data locality:

$$
\begin{array}{l}
\textbf{for}\ (i = 0; i < 100; i++) \\
\quad \textbf{for}\ (j = 0; j < 100; j++) \\
\qquad b += a[i][j] + 1; \\
\qquad c += a[i+1][j+1] + 2;
\end{array}
$$

transformed to:

$$
\begin{array}{l}
\textbf{multifor}\ (i_1 = 0, i_2 = 0; i_1 < 100, i_2 < 100; i_1++, i_2++; 1, 1; 1, 0) \\
\quad \textbf{multifor}\ (j_1 = 0, j_2 = 0; j_1 < 100, j_2 < 100; j_1++, j_2++; 1, 1; 1, 0) \\
\qquad 0 : b += a[i][j] + 1; \\
\qquad 1 : c += a[i+1][j+1] + 2;
\end{array}
$$

Another way of writing loop nests

- Tiling:

$$
\begin{array}{l}
\textbf{for } (it = 0; it < N; it+ = tsize1) \\
\quad \textbf{for } (jt = 0; jt < N; jt+ = tsize2) \\
\quad\quad \textbf{for } (i = it; i < it + tsize1; i++) \\
\quad\quad\quad \textbf{for } (j = jt; j < jt + tsize2; j++) \\
\quad\quad\quad\quad inst\_block
\end{array}
$$

or:

$$
\begin{array}{l}
\textbf{multifor } ([N/tsize1]\ i = [0, tsize1]; i < i + tsize1; i++; \\
\qquad\qquad [N/tsize1]\ 1; [0, tsize1]) \\
\quad \textbf{multifor } ([N/tsize2]\ j = [0, tsize2]; j < j + tsize2; j++; \\
\qquad\qquad\quad [N/tsize2]\ 1; [N/tsize2]\ 0) \\
\quad\quad * : inst\_block
\end{array}
$$

- Requires some extensions:
  - $[n]\ i$: n indices $i_1, i_2, ..., i_p$
  - $[a, b]$: n values $a, a + b, a + 2b, a + 3b, ...$
  - $[n]\ m$: $m, m, m, ...$ (n times) ; $*$: every nest executes

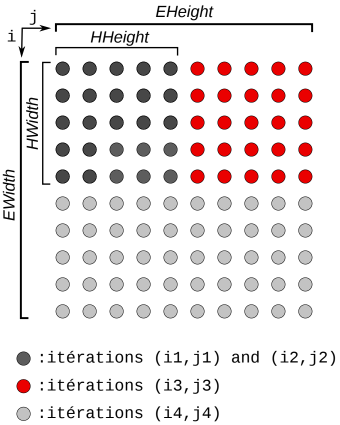
Some *real* examples

- Steganography: decoding phase where a $(HWidth \times HHeight)$ image is hidden in a $(EWidth \times EHeight)$ image

$$
\begin{array}{l}
\textbf{multifor } (i_1 = 0, i_2 = 0; i_3 = 0, i_4 = HWidth; i_1 < HWidth, \\
\qquad\qquad i_2 < HWidth, i_3 < HWidth, i_4 < EWidth; \\
\qquad\qquad i_1++, i_2++, i_3++, i_4++; 1,1,1,1; 0,0,0,0) \\
\textbf{multifor } (j_1 = 0, j_2 = 0, j_3 = HHeight, j_4 = 0; j_1 < HHeight, \\
\qquad\qquad j_2 < HHeight, j_3 < EHeight, j_4 < EHeight; \\
\qquad\qquad j_1++, j_2++, j_3++, j_4++; 1,1,1,1; 0,0,0,0) \\
\{ \\
\ 0: \ \text{// Retrieve the hidden image} \\
\qquad * HImage(i_1, j_1) = decode\_hidden(i_1, j_1); \\
\ 1: \ \text{// Retrieve the enclosing image} \\
\qquad * MImage(i_2, j_2) = decode\_main(i_2, j_2); \\
\ [2,3]: \ \text{// Retrieve the enclosing image} \\
\qquad * MImage([i_3, i_4], [j_3, j_4]) = *EImage([i_3, i_4], [j_3, j_4]); \\
\}
\end{array}
$$

Some *real* examples

- Steganography: multifor-loop nest scan of the images



Some *real* examples

- Red-Black Gauss-Seidel: traditional code

// Red phase

$$
\begin{array}{l}
\textbf{for } (i = 1; i < N-1; i++) \\
\ \textbf{for } (j = 1; j < N-1; j++) \\
\ \ \textbf{if } ((i+j)\ \%\ 2 == 0) \\
\ \ \ \ u[i][j] = f(u[i][j+1], u[i][j-1], u[i-1][j], u[i+1][j]);
\end{array}
$$

// Black phase

$$
\begin{array}{l}
\textbf{for } (i = 1; i < N-1; i++) \\
\ \textbf{for } (j = 1; j < N-1; j++) \\
\ \ \textbf{if } ((i+j)\ \%\ 2 == 1) \\
\ \ \ \ u[i][j] = f(u[i][j+1], u[i][j-1], u[i-1][j], u[i+1][j]);
\end{array}
$$

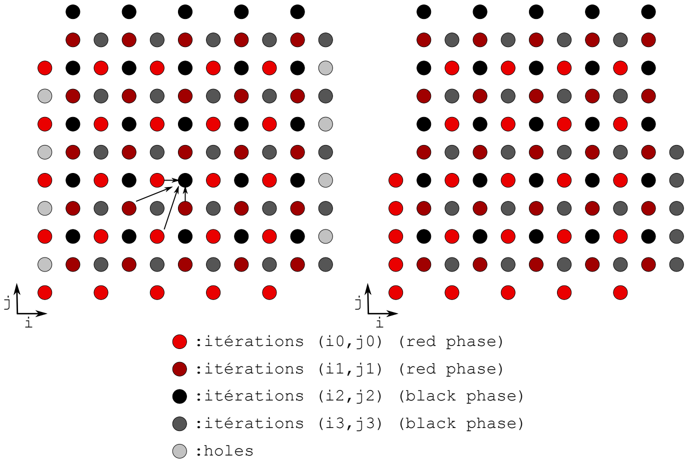
Some *real* examples

- Red-Black Gauss-Seidel: multifor code

$$
\begin{array}{l}
\textbf{multifor}\ (i_0 = 1, i_1 = 2, i_2 = 1, i_3 = 2; i_0 < N-1, i_1 < N-1, \\
\quad i_2 < N-1, i_3 < N-1; i_0+ = 2, i_1+ = 2, i_2+ = 2, \\
\quad i_3+ = 2; 2, 2, 2, 2; 0, 1, 1, 2) \\
\textbf{multifor}\ (j_0 = 1, j_1 = 2, j_2 = 2, j_3 = 1; j_0 < N-1, j_1 < N-1, \\
\quad j_2 < N-1, j_3 < N-1; j_0+ = 2, j_1+ = 2, j_2+ = 2, \\
\quad j_3+ = 2; 2, 2, 2, 2; 0, 1, 2, 1)\ \{ \\
0 : u[i_0][j_0] = \\
\quad f(u[i_0][j_0+1], u[i_0][j_0-1], u[i_0-1][j_0], u[i_0+1][j_0]); \\
1 : u[i_1][j_1] = \\
\quad f(u[i_1][j_1+1], u[i_1][j_1-1], u[i_1-1][j_1], u[i_1+1][j_1]); \\
2 : u[i_2][j_2] = \\
\quad f(u[i_2][j_2+1], u[i_2][j_2-1], u[i_2-1][j_2], u[i_2+1][j_2]); \\
3 : u[i_3][j_3] = \\
\quad f(u[i_3][j_3+1], u[i_3][j_3-1], u[i_3-1][j_3], u[i_3+1][j_3]); \\
\}
\end{array}
$$

Some *real* examples

- Red-Black Gauss-Seidel: multifor referential domain



Some *real* examples

- Red-Black Gauss-Seidel: possible generated for-loop code

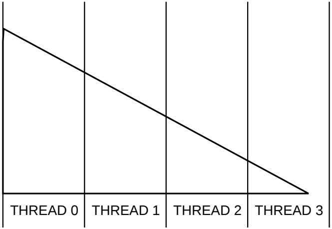
$$
\begin{array}{l}
\textbf{for}\ (j = 1; j < N-1; j+=2) \\
\ u[i][j] = f(u[i][j+1], u[i][j-1], u[i-1][j], u[i+1][j]); \\
\textbf{for}\ (i = 2; i < N-2; i+=2)\ \{ \\
\ \ \textbf{for}\ (j = 2; j < N-1; j+=2)\ \{ \\
\ \ \ u[i][j] = f(u[i][j+1], u[i][j-1], u[i-1][j], u[i+1][j]); \\
\ \ \ u[i][j+1] = f(u[i][j+2], u[i][j], \\
\qquad\qquad u[i-1][j+1], u[i+1][j+1]);\ \} \\
\ \ \textbf{for}\ (j = 1; j < N-1; j+=2)\ \{ \\
\ \ \ u[i+1][j] = f(u[i+1][j+1], u[i+1][j-1], \\
\qquad\qquad u[i][j], u[i+2][j]); \\
\ \ \ u[i+1][j+1] = f(u[i+1][j+2], u[i+1][j], \\
\qquad\qquad u[i][j+1], u[i+2][j+1]);\ \}\ \} \\
\textbf{for}\ (j = 2; j < N-1; j+=2)\ \{ \\
\ u[N-2][j] = f(u[N-2][j+1], u[N-2][j-1], \\
\qquad\qquad u[N-3][j], u[N-1][j]);\ \}
\end{array}
$$

A promising perspective: non-linear mapping

A promising perspective: non-linear mapping
Example:

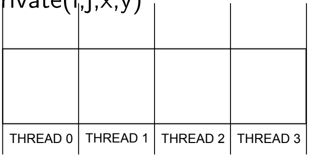
- Load unbalance

  \# pragma omp parallel for shared(a,b) private(i,j)
  for $(i = 0; i < N; i++)$
  for $(j = 0; j < N - i; j++)$
  $a[j][i] = b[j][i] + 12;$



- $N(N+1)/2$ iterations
- Such a loop nest would be better:

  \# pragma omp parallel for shared(a,b) private(i,j,x,y)
  for $(i = 0; i < N; i++)$
  for $(j = 0; j < (N+1)/2; j++)$
  $x = ?; y = ?;$
  $a[y][x] = b[y][x] + 12;$



A promising perspective: non-linear mapping
Example:

- Ranking polynomial of the first nest:

$$\forall(i,j) \in D_1, R_1(i,j) = Ni - \frac{i(i-1)}{2} + j + 1 = \left\{1, 2, ..., \frac{N(N+1)}{2}\right\}$$

- Ranking polynomial of the second nest:

$$\forall(i,j) \in D_2, R_2(i,j) = \frac{(N+1)}{2} i + j + 1 = \left\{1, 2, ..., \frac{N(N+1)}{2}\right\}$$

- Equation to be solved:

$$\forall(i,j) \in D_2, \exists(x,y) \in D_1 \text{ s.t. } R_1(x,y) = K = R_2(i,j)$$

A promising perspective: non-linear mapping

Example:

- Solving $R_1(x,0) - K = Ni - \frac{i(i-1)}{2} + 1 - K = 0$
- Two roots:

$$r_1 = \frac{2N + 1 - \sqrt{(2N+1)^2 + 8(1-K)}}{2}$$

$$r_2 = \frac{2N + 1 + \sqrt{(2N+1)^2 + 8(1-K)}}{2}$$

- $\lfloor r_1 \rfloor$ is the solution x of $R_1(x,y) = K$
- $\implies y = K - R_1(\lfloor r_1 \rfloor, 0) = K - N\lfloor r_1 \rfloor + \frac{\lfloor r_1 \rfloor(\lfloor r_1 \rfloor - 1)}{2} - 1$

A promising perspective: non-linear mapping
Example:

- Second loop nest:

  # pragma omp parallel for shared(a,b) private(i,j,x,y,K)
  for $(i = 0; i < N; i++)$
  for $(j = 0; j < (N+1)/2; j++)$
  $K = (N+1) * i/2 + j + 1;$
  $x = ((2 * N + 1) - \mathbf{sqrt}((2 * N + 1) * (2 * N + 1) + 8 * (1 - K)))/2;$
  $y = K - (N * x - x * (x - 1)/2 + 1);;$
  $a[y][x] = b[y][x] + 12;$

- sqrt is very time consuming: important slow-down

A promising perspective: non-linear mapping Example:

- New version: pre-computing a sufficient range of square roots

  $\#$ pragma omp parallel for shared(a,b) private(i,j,x,y,K)
  for $(i = 0; i < N; i++)$
  for $(j = 0; j < (N+1)/2; j++)$
  $K = (N+1) * i/2 + j + 1;$
  $x = ((2*N+1) - \mathbf{tab}[(2*N+1)*(2*N+1) + 8*(1-K)])/2;$
  $y = K - (N*x - x*(x-1)/2 + 1);;$
  $a[y][x] = b[y][x] + 12;$

- 1.3 speed-up with 12 threads with the second nest vs the first ($N = 4000$, AMD Opteron 6172, 12 cores, 2.1 Ghz)

Perspectives & conclusion

Perspectives & conclusion

Multifor

- Many possible extensions
  - loop indices used in other loops
  - variable grain and offset
  - parallelism in several dimensions (loops, grains, offsets)
  - non-linear control
    - multiwhile?
- Inter-nests code analysis and transformations
- Implementation in CLANG-LLVM

Non-linear mapping

- Other application opportunities
  - data locality, scheduling, ...
- Non-linear analysis

THANK YOU

University of Strasbourg
INRIA Nancy Grand-Est
http://team.inria.fr/camus