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?

Parallelism must naturally take part of the programming process: 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?

- 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

```
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} 
    }
```

where

- `expr`: affine arithmetic expressions on enclosing loop indices
- `cst, grain` and `offset`: integer constants
- `grain` ≥ 1, `offset` ≥ 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

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**

\[
\text{multifor } (i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 ++, i_2 ++; 1, 1; 0, 2)
\]

\[
\text{i} \quad \circ \circ \circ \circ \circ \circ \circ \circ \\
\text{i}_1 \quad \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \\
\text{i}_2 \quad \bullet \bullet \bullet \bullet \bullet \\
\]

**grain + compression**

\[
\text{multifor } (i_1 = 0, i_2 = 10; i_1 < 10, i_2 < 15; i_1 ++, i_2 ++; 1, 4; 0, 0)
\]

\[
\text{i} \quad \circ \circ \circ \circ \circ \circ \circ \circ \\
\text{i}_1 \quad \bullet \bullet \bullet \bullet \bullet \bullet \bullet \bullet \\
\text{i}_2 \quad \bullet \bullet \bullet \bullet \bullet \\
\]
Nested multifor-loops

\begin{verbatim}
multifor ( \( \text{index}_1 = \text{expr}, \text{index}_2 = \text{expr}; \)
\( \text{index}_1 < \text{expr}, \text{index}_2 < \text{expr}; \)
\( \text{index}_1 += \text{cst}, \text{index}_2 += \text{cst}; \)
\( \text{grain}_1, \text{grain}_2; \text{offset}_1, \text{offset}_2 ) \) { 
  prefix : \{ \text{statements} \}

multifor ( \( \text{index}_3 = \text{expr}, \text{index}_4 = \text{expr}; \)
\( \text{index}_3 < \text{expr}, \text{index}_4 < \text{expr}; \)
\( \text{index}_3 += \text{cst}, \text{index}_4 += \text{cst}; \)
\( \text{grain}_3, \text{grain}_4; \text{offset}_3, \text{offset}_4 ) \) { 
  prefix : \{ \text{statements} \}
}

prefix : \{ \text{statements} \}
\end{verbatim}

- behaves as 2 for-loop nests \((\text{index}_1, \text{index}_3)\) and \((\text{index}_2, \text{index}_4)\)
- the bounds are affine functions of the enclosing loop indices of the same for-loop
Examples: nested multifor-loops

offset

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

\(i\) \(j\): itéréations \((i_1, j_1)\)

\(i\) \(j\): itéréations \((i_1, j_1)\) et \((i_2, j_2)\)
Examples: nested multifor-loops

\begin{align*}
\text{multifor} & \quad (i_1 = 0, i_2 = 0; i_1 < 10, i_2 < 3; i_1 + +, i_2 + +; 1, 4; 0, 0) \\
\text{multifor} & \quad (j_1 = 0, j_2 = 0; j_1 < 10, j_2 < 3; j_1 + +, j_2 + +; 1, 4; 0, 0)
\end{align*}
Examples: nested multifor-loops

_affine bound + offset_

**multifor** \((i_1 = 0, i_2 = 0; i_1 < 6, i_2 < 6; i_1 +=, i_2 +=; 1, 1; 0, 1)\)

**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:

  ```
  for (i = 0; i < 10; i += 1)
    inst_block1
  for (j = 0; j < 10; j += 1)
    inst_block2
  ```

  or

  ```
  multifor (i1 = 0, i2 = 0; i1 < 10, i2 < 10; i1 ++, i2 ++; 1, 1; 0, 0)
  multifor (j1 = 0, j2 = 0; j1 < 1, j2 < 10; j1 ++, j2 ++; 1, 1; 0, 1)
  0 : inst_block1
  1 : inst_block2
  ```
Another way of writing loop nests

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

```plaintext
for (i = 0; i < 100; i++)
    for (j = 0; j < 100; j++)
        b += a[i][j] + 1;
        c += a[i + 1][j + 1] + 2;
```

transformed to:

```plaintext
multifor (i₁ = 0, i₂ = 0; i₁ < 100, i₂ < 100; i₁ ++, i₂ ++; 1, 1; 1, 0)
  multifor (j₁ = 0, j₂ = 0; j₁ < 100, j₂ < 100; j₁ ++, j₂ ++; 1, 1; 1, 0)
    0 : b += a[i][j] + 1;
    1 : c += a[i + 1][j + 1] + 2;
```
Another way of writing loop nests

- **Tiling:**

  ```
  for (it = 0; it < N; it+ = tsize1)
    for (jt = 0; jt < N; jt+ = tsize2)
      for (i = it; i < it + tsize1; i++)
        for (j = jt; j < jt + tsize2; j++)
          inst_block
  ```

  or:

  ```
  multifor ([N/tsize1] i = [0, tsize1]; i < i + tsize1; i++;
  [N/tsize1] 1; [0, tsize1])
  multifor ([N/tsize2] j = [0, tsize2]; j < j + tsize2; j++;
  [N/tsize2] 1; [N/tsize2] 0)
  * : inst_block
  ```

- **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

\[
\text{multifor} \ (i_1 = 0, i_2 = 0; i_3 = 0, i_4 = HWidth; i_1 < HWidth, \\
i_2 < HWidth, i_3 < HWidth, i_4 < EWidth; \\
i_1 ++, i_2 ++, i_3 ++, i_4 ++; 1, 1, 1, 1; 0, 0, 0, 0) \]

\[
\text{multifor} \ (j_1 = 0, j_2 = 0, j_3 = HHeight, j_4 = 0; j_1 < HHeight, \\
j_2 < HHeight, j_3 < EHeight, j_4 < EHeight; \\
j_1 ++, j_2 ++, j_3 ++, j_4 ++; 1, 1, 1, 1; 0, 0, 0, 0) \]

\[
\begin{align*}
0: & \quad \text{// Retrieve the hidden image} \\
& \quad HImage(i_1, j_1) = decode\_hidden(i_1, j_1); \\
1: & \quad \text{// Retrieve the enclosing image} \\
& \quad MImage(i_2, j_2) = decode\_main(i_2, j_2); \\
[2, 3]: & \quad \text{// Retrieve the enclosing image} \\
& \quad MImage([i_3, i_4], [j_3, j_4]) = EImage([i_3, i_4], [j_3, j_4]);
\end{align*}
\]
Some real examples

- Steganography: multifor-loop nest scan of the images
Some real examples

- Red-Black Gauss-Seidel: traditional code

```c
// Red phase
for (i = 1; i < N - 1; i++)
    for (j = 1; j < N - 1; j++)
        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]);

// Black phase
for (i = 1; i < N - 1; i++)
    for (j = 1; j < N - 1; j++)
        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]);
```
Some real examples

- **Red-Black Gauss-Seidel**: multifor code

```plaintext
multifor (i_0 = 1, i_1 = 2, i_2 = 1, i_3 = 2; i_0 < N - 1, i_1 < N - 1,
         i_2 < N - 1, i_3 < N - 1; i_0+ = 2, i_1+ = 2, i_2+ = 2,
         i_3+ = 2; 2, 2, 2, 2; 0, 1, 1, 2) 
multifor (j_0 = 1, j_1 = 2, j_2 = 2, j_3 = 1; j_0 < N - 1, j_1 < N - 1,
         j_2 < N - 1, j_3 < N - 1; j_0+ = 2, j_1+ = 2, j_2+ = 2,
         j_3+ = 2; 2, 2, 2, 2; 0, 1, 2, 1)
{
  0 : u[i_0][j_0] = 
      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] = 
      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] = 
      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] = 
      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]);
}
```

Some real examples

- **Red-Black Gauss-Seidel**: multifor referential domain

- Itérations \((i0,j0)\) (red phase)
- Itérations \((i1,j1)\) (red phase)
- Itérations \((i2,j2)\) (black phase)
- Itérations \((i3,j3)\) (black phase)
- Holes

---

Ph. Clauss - Multifor for Multicore - IMPACT 2013

January 21 - 19/29
Some real examples

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

```plaintext
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]);
for (i = 2; i < N - 2; i+ = 2) {
    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],
                       u[i - 1][j + 1], u[i + 1][j + 1]);
    }
    for (j = 1; j < N - 1; j+ = 2) {
        u[i + 1][j] = f(u[i + 1][j + 1], u[i + 1][j - 1],
                       u[i][j], u[i + 2][j]);
        u[i + 1][j + 1] = f(u[i + 1][j + 2], u[i + 1][j],
                            u[i][j + 1], u[i + 2][j + 1]);
    }
    for (j = 2; j < N - 1; j+ = 2) {
        u[N - 2][j] = f(u[N - 2][j + 1], u[N - 2][j - 1],
                        u[N - 3][j], u[N - 1][j]);
    }
```
A promising perspective: non-linear mapping
A promising perspective: non-linear mapping

Example:

- **Load unbalance**

  ```c
  # 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:**

  ```c
  # 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:
  \[
  \forall (i,j) \in D_1, R_1(i,j) = N i - \frac{i(i-1)}{2} + j + 1 \in \left\{1, 2, \ldots, \frac{N(N+1)}{2}\right\}
  \]

- Ranking polynomial of the second nest:
  \[
  \forall (i,j) \in D_2, R_2(i,j) = \frac{(N+1)}{2} i + j + 1 \in \left\{1, 2, \ldots, \frac{N(N+1)}{2}\right\}
  \]

- 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 \( R_1(x, y) = K \)

- \( \implies y = K - R_1(\lfloor r_1 \rfloor, 0) = K - N \lfloor r_1 \rfloor + \frac{r_1(\lfloor r_1 \rfloor - 1)}{2} - 1 \)
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) - 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

```c
#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) - 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
  \(N = 4000, \text{AMD Opteron 6172, 12 cores, 2.1 Ghz}\)
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