Pipelined Multithreading Generation in a Polyhedral Compiler

January 22nd 2020, IMPACT’20, HiPEAC, Bologna, Italy

Harenome Ranaivoarivony-Razanajato, Cédric Bastoul, Vincent Loechner

University of Strasbourg and Inria Nancy Grand Est
Motivating Example

```
1  for (int i = 1; i < N; ++i)
2      A[i] = f1(A[i], A[i - 1]);  // S1

3  for (int i = 1; i < N; ++i)
4      B[i] = f2(A[i], B[i - 1]);  // S2

5  /* ... */

6  for (int i = 1; i < N; ++i)
7      F[i] = f6(E[i], F[i - 1]);  // S6
```

(a) Sequential Program
Motivating Example

1. for (int i = 1; i < N; ++i)  
   A[i] = f1(A[i], A[i - 1]); // S1

2. for (int i = 1; i < N; ++i)  
   B[i] = f2(A[i], B[i - 1]); // S2

3. /* ... */

4. for (int i = 1; i < N; ++i)  
   F[i] = f6(E[i], F[i - 1]); // S6

(a) Sequential Program

(b) Dependency Graph

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanjato et al.
Motivating Example

1. for (int i = 1; i < N; ++i)
2.   A[i] = f1(A[i], A[i - 1]); // S1

3. for (int i = 1; i < N; ++i)
4.   B[i] = f2(A[i], B[i - 1]); // S2

5. /* ... */

6. for (int i = 1; i < N; ++i)
7.   F[i] = f6(E[i], F[i - 1]); // S6

(a) Sequential Program                (b) Pipelined Execution
Motivating Example

1 for (int i = 1; i < N; ++i)
2   A[i] = f1(A[i], A[i - 1]); // S1

3 for (int i = 1; i < N; ++i)
4   B[i] = f2(A[i], B[i - 1]); // S2

5 /* ... */

6 for (int i = 1; i < N; ++i)
7   F[i] = f6(F[i], F[i - 1]); // S6

(a) Sequential Program

(b) Pipelined Execution
Motivating Example

(a) Sequential Program

1  for (int i = 1; i < N; ++i)
2      A[i] = f1(A[i], A[i - 1]); // S1

3  for (int i = 1; i < N; ++i)
4      B[i] = f2(A[i], B[i - 1]); // S2

5      /* ... */

6  for (int i = 1; i < N; ++i)
7      F[i] = f6(E[i], F[i - 1]); // S6

(b) Pipelined Execution

S1(1), thread 1
S2(1), thread 1
S1(2), thread 2
Motivating Example

(a) Sequential Program

(b) Pipelined Execution

1 for (int i = 1; i < N; ++i)
2 A[i] = f1(A[i], A[i - 1]); // S1

3 for (int i = 1; i < N; ++i)
4 B[i] = f2(A[i], B[i - 1]); // S2

5 /* ... */

6 for (int i = 1; i < N; ++i)
7 F[i] = f6(E[i], F[i - 1]); // S6
Motivating Example

(a) Sequential Program

1. for (int i = 1; i < N; ++i)
2.   A[i] = f1(A[i], A[i - 1]); // S1

3. for (int i = 1; i < N; ++i)
4.   B[i] = f2(A[i], B[i - 1]); // S2

5. /* ... */

6. for (int i = 1; i < N; ++i)
7.   F[i] = f6(E[i], F[i - 1]); // S6

(b) Pipelined Execution

S1(1), thread 1
S2(1), thread 1
S3(1), thread 1
S1(2), thread 2
S2(2), thread 2
S1(3), thread 3
S2(3), thread 3
S3(3), thread 3
Motivating Example

(a) Sequential Program

1. for (int i = 1; i < N; ++i)
2. \( A[i] = f1(A[i], A[i - 1]); // S1\)

(b) Pipelined Execution

3. for (int i = 1; i < N; ++i)
4. \( B[i] = f2(A[i], B[i - 1]); // S2\)

5. /* ... */

6. for (int i = 1; i < N; ++i)
7. \( F[i] = f6(E[i], F[i - 1]); // S6\)

---

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanjato et al.

1
Motivating Example

(a) Sequential Program

```c
for (int i = 1; i < N; ++i)
    A[i] = f1(A[i], A[i - 1]); // S1

for (int i = 1; i < N; ++i)
    B[i] = f2(A[i], B[i - 1]); // S2

/* ... */

for (int i = 1; i < N; ++i)
    F[i] = f6(E[i], F[i - 1]); // S6
```

(b) Pipelined OpenMP target program

```c
#pragma omp parallel
{
#pragma omp for schedule(static) ordered nowait
    for (int i = 1; i < N; ++i)
        #pragma omp ordered
        A[i] = f1(A[i], A[i - 1]); // S1

#pragma omp for schedule(static) ordered nowait
    for (int i = 1; i < N; ++i)
        #pragma omp ordered
        B[i] = f2(A[i], B[i - 1]); // S2

/* ... */

#pragma omp for schedule(static) ordered nowait
    for (int i = 1; i < N; ++i)
        #pragma omp ordered
        F[i] = f6(E[i], F[i - 1]); // S6
}
```

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanajato et al.
Motivating Example

```c
1 #pragma omp parallel
2 {
3   #pragma omp for schedule(static) ordered nowait
4     for (int i = 1; i < N; ++i)
5       A[i] = f1(A[i], A[i - 1]); // S1
6 #pragma omp for schedule(static) ordered nowait
7     for (int i = 1; i < N; ++i)
8       B[i] = f2(A[i], B[i - 1]); // S2
9   /* ... */
10 #pragma omp for schedule(static) ordered nowait
11     for (int i = 1; i < N; ++i)
12       F[i] = f6(E[i], F[i - 1]); // S6
13 }
```

(a) Sequential Program

(b) Pipelined OpenMP target program

Speedup: 2.89

6 stages on an Intel Xeon E5-2620v3 @ 2.40 GHz, with $N = 100,000$

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanajato et al.
Goals

- Identifying software pipelines in a polyhedral compiler
- Generate pipelined multithreading using OpenMP
Polyhedral Model

Introduction

Background

Pipelined Multithreading Generation

Experimental Results

Conclusion
• #pragma based API for shared memory parallelism
• Worksharing constructs
  • #pragma omp for
  • #pragma omp task

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanajato et al.
• `#pragma` based API for shared memory parallelism
• Worksharing constructs
  • `#pragma omp for`
  • `#pragma omp task`
• Synchronization
  • `#pragma omp barrier`: explicit synchronization barrier
  • `omp_set_lock()` and `omp_unset_lock()`: explicit lock mechanism
OpenMP

- #pragma based API for shared memory parallelism
- Worksharing constructs
  - #pragma omp for
  - #pragma omp task
- Synchronization
  - #pragma omp barrier: explicit synchronization barrier
  - omp_set_lock() and omp_unset_lock(): explicit lock mechanism
- Clauses
  - nowait clause on worksharing constructs: omit the implicit barrier at the end of a worksharing construct
  - ordered clause on worksharing constructs: sequentialize a region
Polyhedral Model

Introduction

Background

**Pipelined Multithreading Generation**

Sequential Loop Fission

Relaxed nowait prerequisites

Alternative: Explicit synchronization

Experimental Results

Conclusion
Sequential Loop Fission

- Goal: maximize the number of pipeline stages
- Dependence analysis: identify Strongly Connected Components
Sequential Loop Fission

(a) Original loop body

(b) Fission of Strongly Connected Components

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanajato et al.
The safe use of the `nowait` clause between two parallel loops requires that there are no dependencies between the loops or that:

- the sizes of the iteration domains are equal
- the chunk size is either the same or not specified
- both loops are bound to the same parallel region
- none of the loops is associated with a SIMD construct
- the second loop depends only on the same logical iteration of the first loop
The safe use of the `nowait` clause between two *ordered* loops requires that there are no dependencies between the loops or that:

- the sizes of the iteration domains are equal
- the chunk size is either the same or not specified
- both loops are bound to the same parallel region
- none of the loops is associated with a SIMD construct
- the second loop depends only on the same logical iteration of the first loop
The safe use of the `nowait` clause between two ordered loops requires that there are no dependencies between the loops or that:

- the sizes of the iteration domains are equal
- the chunk size is either the same or not specified
- both loops are bound to the same parallel region
- none of the loops is associated with a SIMD construct
- the second loop depends only on the same logical iteration of the first loop
- the second loop depends on the same logical iteration or previous logical iterations of the first loop
Relaxed conditions on the nowait clause for ordered loops

(a) Parallel for and nowait

(b) Ordered for and nowait

1 #pragma omp parallel
2 {
3 #pragma omp for nowait
4 for (int i = 0; i < N; ++i)
5 A[i] = f1(A[i]);
6 #pragma omp for
7 for (int i = 0; i < N; ++i)
8 B[i] = f2(B[i], A[i]);
9 }

1 #pragma omp parallel
2 {
3 #pragma omp for ordered nowait
4 for (int i = 0; i < N; ++i)
5 #pragma omp ordered
6 A[i] = f1(A[i]);
7 #pragma omp for ordered
8 for (int i = 0; i < N; ++i)
9 #pragma omp ordered
10 B[i] = f2(B[i], A[i-1]);
11 }
Annotate sequential loops with `#pragma omp for ordered`

Enclose sequential loop bodies in `#pragma omp ordered regions`

Annotate loops with `nowait` where possible

Optimize by reverting ordered loops without `nowait` clauses to `#pragma omp single regions`
Introduction

Background

Pipelined Multithreading Generation

Sequential Loop Fission

Relaxed nowait prerequisites

Alternative: Explicit synchronization

Experimental Results

Conclusion
Explicit synchronization

- Loop blocking and loop fusion
- `#pragma omp for schedule(static, 1)` on the blocking loop
- `omp_set_lock()` and `omp_unset_lock()` before and after each loop of the pipeline
- up to $n \times m$ locks required for $m$ pipeline stages over $n$ threads
Explicit synchronization

(a) Original program

(b) Pipelined OpenMP target program

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanajato et al.
Future Work

Introduction

Background

Pipelined Multithreading Generation

Experimental Results

Conclusion
Experimental Setup

- Tested on an Intel Xeon E5-2620v3 @ 2.40 GHz, linux 5.3.7
- Code compiled using gcc 9.2.1 and clang 9.0.0 with options -O3 -march=native -fopenmp
- FIFO scheduling enabled and process priority set to 75
### Benchmarks

<table>
<thead>
<tr>
<th>benchmark</th>
<th>parallel loops</th>
<th>stages</th>
</tr>
</thead>
<tbody>
<tr>
<td>teaser</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>van_dongen(^1)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>wdf(^2)</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>mix</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

---

\(^1\) (Vincent H Van Dongen, Guang R Gao, and Qi Ning. “A polynomial time method for optimal software pipelining”. In: *Parallel Processing: CONPAR 92—VAPP V*. Springer, 1992, pp. 613–624)

Figure 7: Speedups or slowdowns over sequential version
Results – clang/libomp

Figure 8: Speedups or slowdowns over sequential version

Pipelined Multithreading Generation in a Polyhedral Compiler, Harenome Razanajato et al.
Future Work

Introduction

Background

Pipelined Multithreading Generation

Experimental Results

Conclusion
Contributions and Future Work

• Contributions:
  • Identifying software pipelines in a polyhedral compiler
  • Generating pipelined multithreading

• Future work:
  • Integration in an automatic parallelizer
  • Investigate methods to determine optimal block sizes
Contributions and Future Work

• Contributions:
  • Identifying software pipelines in a polyhedral compiler
  • Generating pipelined multithreading

• Future work:
  • Integration in an automatic parallelizer
  • Investigate methods to determine optimal block sizes
Appendix
