Exploiting the Correlation between Dependence Distance and Latency in Loop Pipelining for HLS

Jianyi Cheng, John Wickerson and George Constantinides

Imperial College London

{jianyi.cheng17, j.wickerson, g.constantindes}@imperial.ac.uk
Loop Pipelining in HLS

```c
for (int i = 0; i < N; i++) {
    A[i] = A[i] + 0.6;
}
```
Loop Pipelining in HLS

for (int i = 0; i < N; i++) {
    A[i] = A[i] + 0.6;
}

| Cycles | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | ...
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0</td>
<td>ld A[0]</td>
<td>A[0] = A[0] + 0.6</td>
<td>st A[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Loop Pipelining in HLS

```c
1 for (int i = 0; i < N; i++) {
2   A[i] = A[i] + 0.6;
3 }
```

<table>
<thead>
<tr>
<th>Cycles</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>i = 0</td>
<td>ld A[0]</td>
<td>A[0] = A[0] + 0.6</td>
<td>st A[0]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Initiation Interval (II):** The number of clock cycles between the start times of the same operation in two consecutive loop iterations.
**Motivating example**

Can Vitis HLS always find the optimal II?

```c
for (int i = 0; i < N; i++) {
    double e = vec[i];
    if (e > 0)
        vec[i+63] = (((((e+0.64)*e +0.7)*e+0.21)*e+0.33)*e+0.25)*e+0.125;
    else
        vec[i*i+9] = e * e;
}
```
Motivating example

Can Vitis HLS always find the optimal II?

```c
for (int i = 0; i < N; i++) {
    double e = vec[i];
    if (e > 0)
        vec[i+63] = ((((e+0.64)*e +0.7)*e+0.21)*e+0.33)
                     *e+0.25)*e+0.125;
    else
        vec[i*i+9] = e * e;
}
```

- Vitis HLS automatically finds an “optimal” II = 41
- By simulation, the actual optimal II = 2
Automatically Determining II

```c
float s = s0;
for (int i = 0; i < N; i++) {
    s = s + 0.6;
}
```
Automatically Determining II

```c
float s = s0;
for (int i = 0; i < N; i++) {
    s = s + 0.6;
}
```
Automatically Determining II

```c
float s = s0;
for (int i = 0; i < N; i++) {
    s = s + 0.6;
}
```

Dependence distance = 1

Latency of line 3 = 4 cycles
Automatically Determining II

```java
float s = s0;
for (int i = 0; i < N; i++) {
    s = s + 0.6;
}
```

Dependence distance = 1

Latency of line 3 = 4 cycles

\[
\left\lfloor \frac{4}{1} \right\rfloor = 4
\]
Automatically Determining II

```c
for (int i = 0; i < N; i++) {
    A[i+3] = A[i] + 0.6;
}
```
Automatically Determining II

```
for (int i = 0; i < N; i++) {
    A[i+3] = A[i] + 0.6;
}
```
Automatically Determining II

```c
for (int i = 0; i < N; i++) {
    A[i+3] = A[i] + 0.6;
}
```

Dependence distance = 3

Latency of line 2 = 7 cycles
Automatically Determining II

```c
for (int i = 0; i < N; i++) {
    A[i+3] = A[i] + 0.6;
}
```

Dependence distance = 3

Latency of line 2 = 7 cycles

\[
II = \left\lfloor \frac{7}{3} \right\rfloor = 3
\]
Automatically Determining II

**Iteration latency**: The number of clock cycles between the start times of an operation and its dependants

**Dependence distance**: The number of iterations that separate an operation from its dependants

\[ II = \left\lfloor \frac{\text{Iteration latency}}{\text{Dependence distance}} \right\rfloor \]
Motivating example

```c
for (int i = 0; i < N; i++) {
    double e = vec[i];
    if (e > 0)
        vec[i+63] = (((((e+0.64)*e
                     +0.7)*e+0.21)*e+0.33)
                     *e+0.25)*e+0.125;
    else
        vec[i*i+9] = e * e;
}
```
Motivating example

```java
for (int i = 0; i < N; i++) {
    double e = vec[i];
    if (e > 0)
        vec[i+63] = (((((e+0.64)*e +0.7)*e+0.21)*e+0.33)*e+0.25)*e+0.125;
    else
        vec[i*i+9] = e * e;
}
```
Motivating example

```c
for (int i = 0; i < N; i++) {
    double e = vec[i];
    if (e > 0)
        vec[i+63] = (((((e+0.64)*e +0.7)*e+0.21)*e+0.33) *
                     e+0.25)*e+0.125;
    else
        vec[i*i+9] = e * e;
}
```
Automatically Determining II

**Iteration latency**: The number of clock cycles between the start times of an operation and its dependants

**Dependence distance**: The number of iterations that separate an operation from its dependants

\[ II = \left[ \frac{\text{Iteration latency}}{\text{Dependence distance}} \right] \]
Automatically Determining II

**Iteration latency**: The number of clock cycles between the start times of an operation and its dependants

**Dependence distance**: The number of iterations that separate an operation from its dependants

\[ II = \left[ \frac{\text{Iteration latency}}{\text{Dependence distance}} \right] \]

Existing works

\[ II = \left[ \frac{\max_D \text{Iteration latency}}{\min_D \text{Dependence distance}} \right] \]
Automatically Determining II

**Iteration latency**: The number of clock cycles between the start times of an operation and its dependants

**Dependence distance**: The number of iterations that separate an operation from its dependants

$$II = \left\lfloor \frac{\text{Iteration latency}}{\text{Dependence distance}} \right\rfloor$$

Existing works

$$II = \left\lceil \frac{\max_D \text{Iteration latency}}{\min_D \text{Dependence distance}} \right\rceil$$

Our work

$$II = \max_D \left\lfloor \frac{\text{Iteration latency}}{\text{Dependence distance}} \right\rfloor$$
Our Contributions

• A new loop pipelining formulation that includes the correlations between the iteration latency and the dependence distance

• A SMT-based approach to support non-linear memory access analysis

• A fully automated HLS pass that finds the optimal II for a loop
Motivating example

Latency of line 4 = 82 cycles
Dependence distance = 63

Latency of line 8 = 8 cycles
Dependence distance = 9

```c
for (int i = 0; i < N; i++) {
    double e = vec[i];
    if (e > 0)
        vec[i+63] = (((((e+0.64)*e +0.7)*e+0.21)*e+0.33)
                      *e+0.25)*e+0.125;
    else
        vec[i*i+9] = e * e;
}
```
Motivating example

Latency of line 4 = 82 cycles
Dependence distance = 63

Latency of line 8 = 8 cycles
Dependence distance = 9

Original formulation:
\[ II = \frac{\max\{82, 8\}}{\min\{63, 9\}} = \frac{82}{9} = 10 \]

Our formulation:
\[ II = \max \left\{ \frac{82}{63}, \frac{8}{9} \right\} = \max\{2, 1\} = 2 \]
Experiments

- Normalised Performance
- Normalised Area

- 11.1x speedup
- 1.95x area
Exploiting the Correlation between Dependence Distance and Latency in Loop Pipelining for HLS

COME and TALK to us!

Tool url: https://github.com/JianyiCheng/iiProver