Imperial College London

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

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

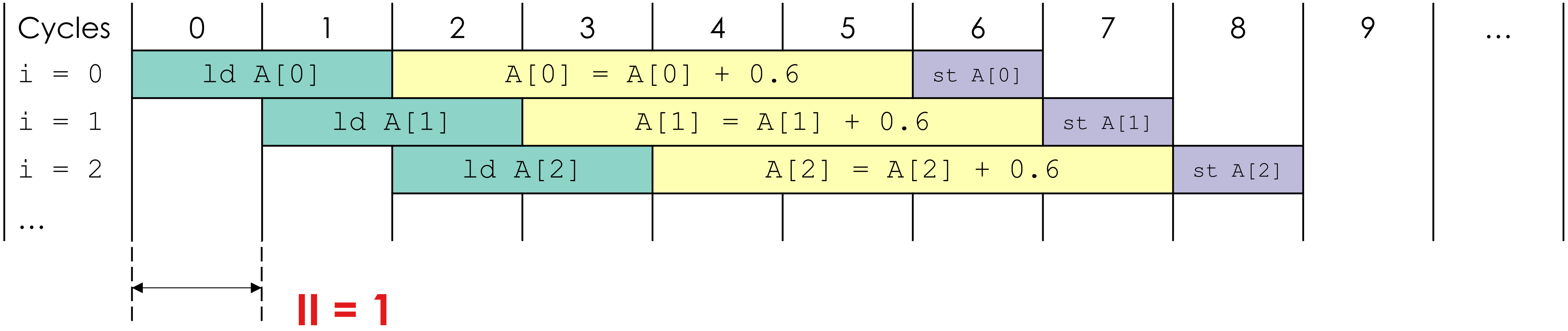
Loop Pipelining in HLS

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

Cycles	0	1	2	3	4	5	6	7	8	9	...
i = 0	ld A[0]		A[0] = A[0] + 0.6				st A[0]				
i = 1		ld A[1]		A[1] = A[1] + 0.6				st A[1]			
i = 2			ld A[2]		A[2] = A[2] + 0.6				st A[2]		
...											

Loop Pipelining in HLS

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



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?

```
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?

```
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

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

Automatically Determining II

```
1 float s = s0;
2 for (int i = 0; i < N; i++) {
3   s = s + 0.6;
4 }
```

Dependence distance = 1

Latency of line 3 = 4 cycles

Automatically Determining II

```
1  float s = s0;
2  for (int i = 0; i < N; i++) {
3    s = s + 0.6;
4  }
```

Dependence distance = 1

Latency of line 3 = 4 cycles

<table>
<tr><td>Cycles</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>...</td></tr>
<tr><td>i = 0</td><td colspan="4">s = s + 0.6</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>i = 1</td><td></td><td></td><td></td><td></td><td colspan="4">s = s + 0.6</td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>i = 2</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td colspan="4">s = s + 0.6</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><td></td><td></td></tr>
</table>

$$II = \left\lceil \frac{4}{1} \right\rceil = 4$$

Automatically Determining II

```
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

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

Dependence distance = 3

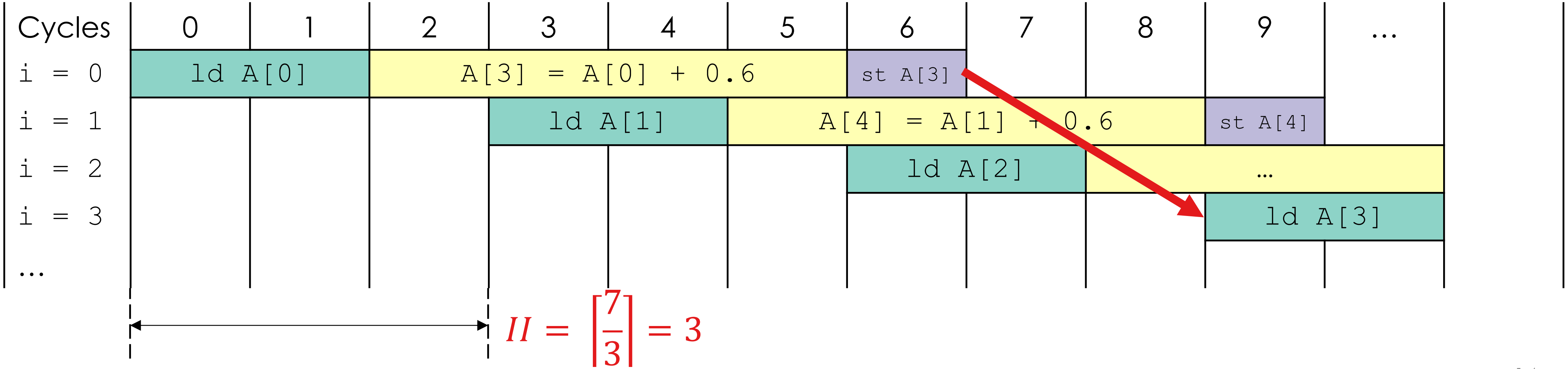
Latency of line 2 = 7 cycles

Automatically Determining II

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

Dependence distance = 3

Latency of line 2 = 7 cycles



$$II = \left\lceil \frac{7}{3} \right\rceil = 3$$

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

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

Motivating example

```
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

```
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

```
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

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

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

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

Existing works ➡

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

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

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

Existing works →

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

Our work ↘

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

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

```
1  for (int i = 0; i < N; i++) {
2    double e = vec[i];
3    if (e > 0)
4      vec[i+63] = (((((e+0.64)*e
5        +0.7)*e+0.21)*e+0.33)
6        *e+0.25)*e+0.125;
7    else
8      vec[i*i+9] = e * e;
9  }
```

Latency of line 4 = 82 cycles
Dependence distance = 63

Latency of line 8 = 8 cycles
Dependence distance = 9

Motivating example

```
1 for (int i = 0; i < N; i++) {
2   double e = vec[i];
3   if (e > 0)
4     vec[i+63] = (((((e+0.64)*e
5       +0.7)*e+0.21)*e+0.33)
6       *e+0.25)*e+0.125;
7   else
8     vec[i*i+9] = e * e;
9 }
```

Latency of line 4 = 82 cycles
Dependence distance = 63

Latency of line 8 = 8 cycles
Dependence distance = 9

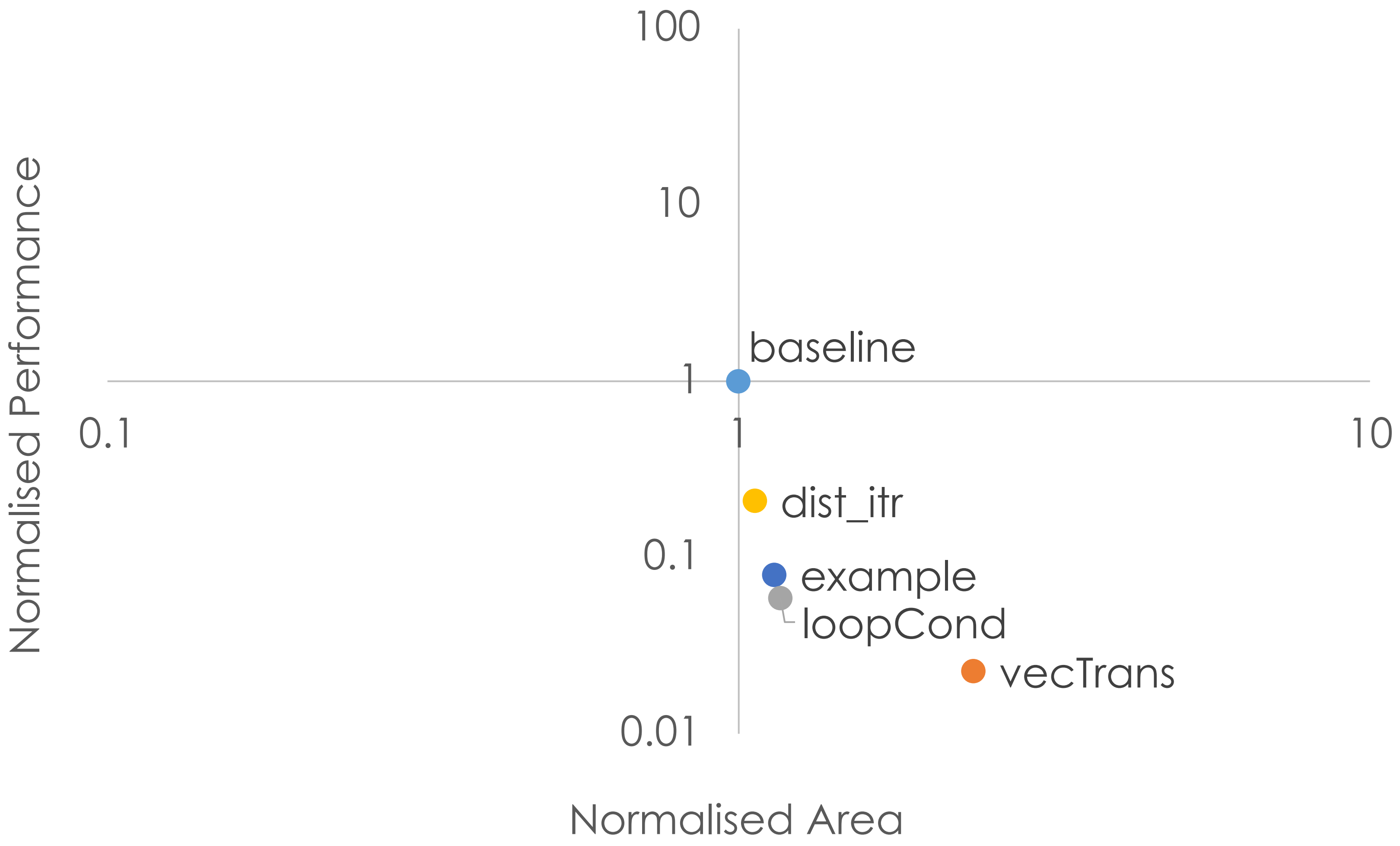
Original formulation:

$$II = \left\lceil \frac{\max\{82, 8\}}{\min\{63, 9\}} \right\rceil = \left\lceil \frac{82}{9} \right\rceil = 10$$

Our formulation:

$$II = \max\left\{ \left\lceil \frac{82}{63} \right\rceil, \left\lceil \frac{8}{9} \right\rceil \right\} = \max\{2, 1\} = 2$$

Experiments



11.1x speedup

1.95x area

Exploiting the Correlation between Dependence Distance and Latency

in Loop Pipelining for HLS

COME and TALK to us!

Paper

Tool

Paper url: https://jianyicheng.github.io/papers/ChengFPL21.pdf
Tool url: https://github.com/JianyiCheng/iiProver