International Conference
on Supercomputing
June 14 - 18, 2021. Worldwide online event

POLITECNICO DI MILANO

Bambu: High-Level Synthesis for Parallel Programming

Exploiting Vectorization in High-Level Synthesis of Nested Irregular Loops

Serena Curzel

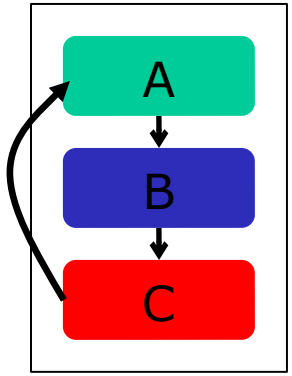
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
serena.curzel@polimi.it

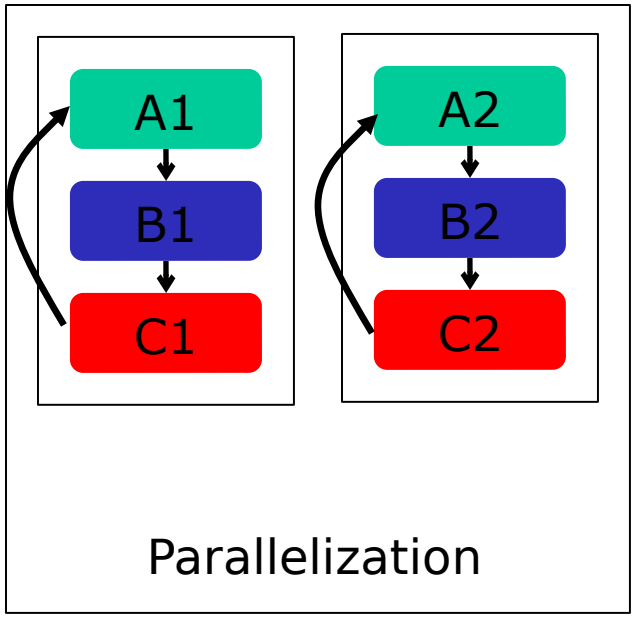
Outline

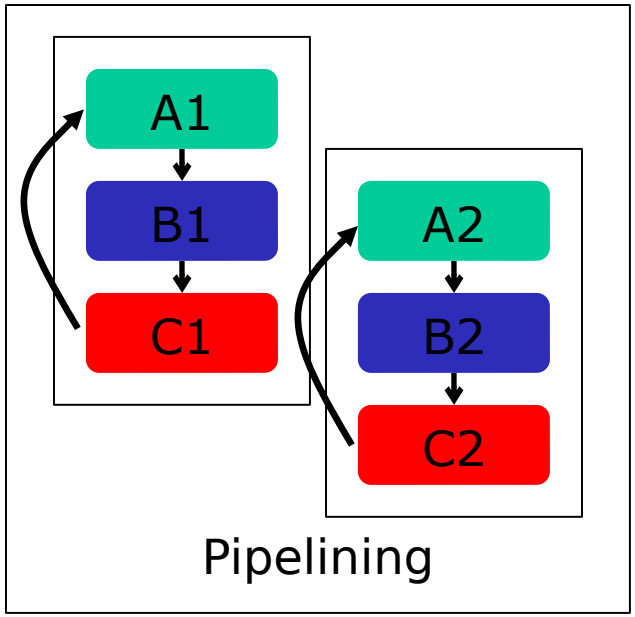
- Outer Loops Vectorization
- Proposed Design Flow
- Experimental Results

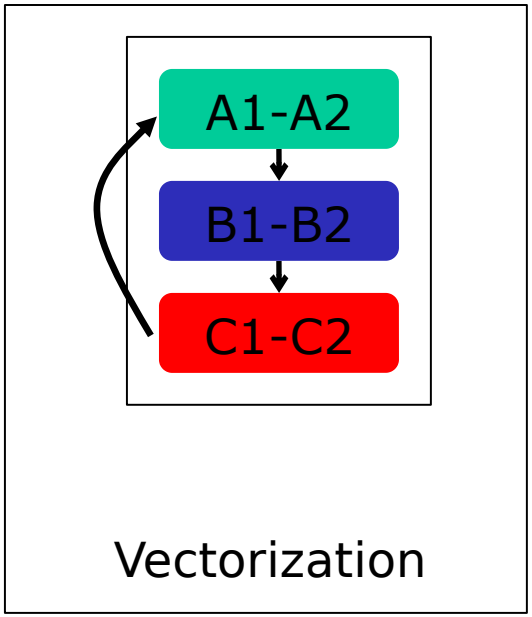
DoAll Loops and their Optimizations

- DoAll loops
  - All the iterations can be executed in parallel
  - Number of iterations may be unknown at compile time
- Optimizations:

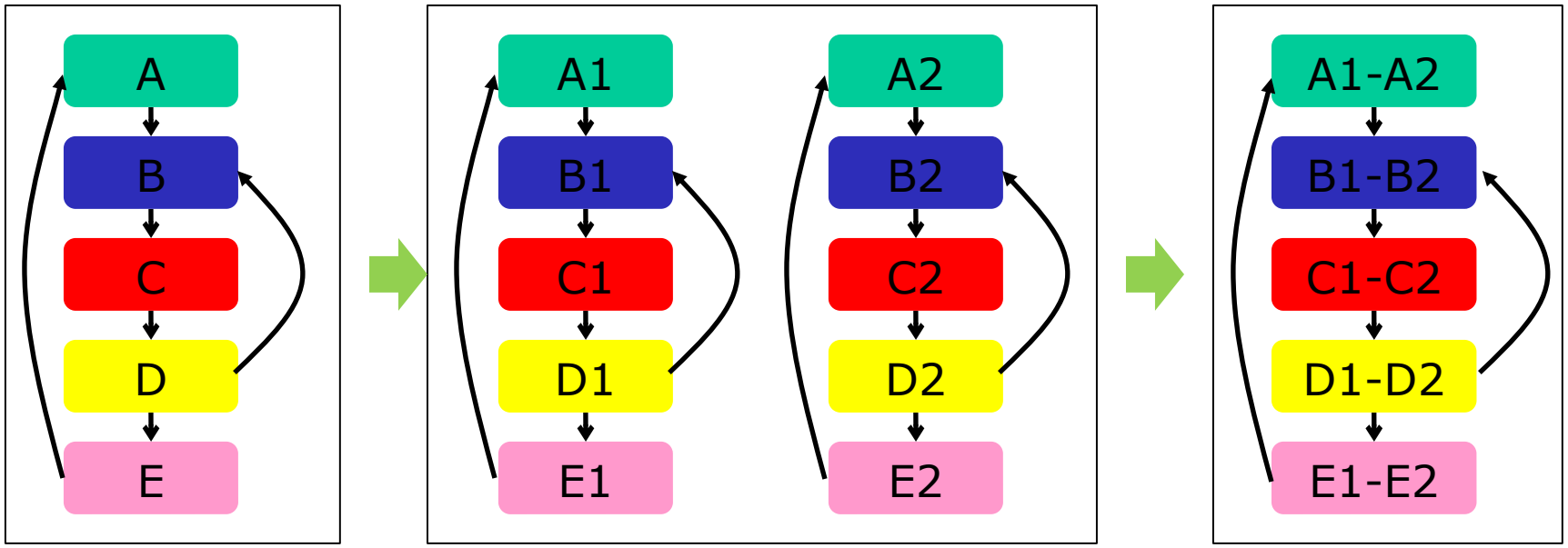








Outer Loops Vectorization

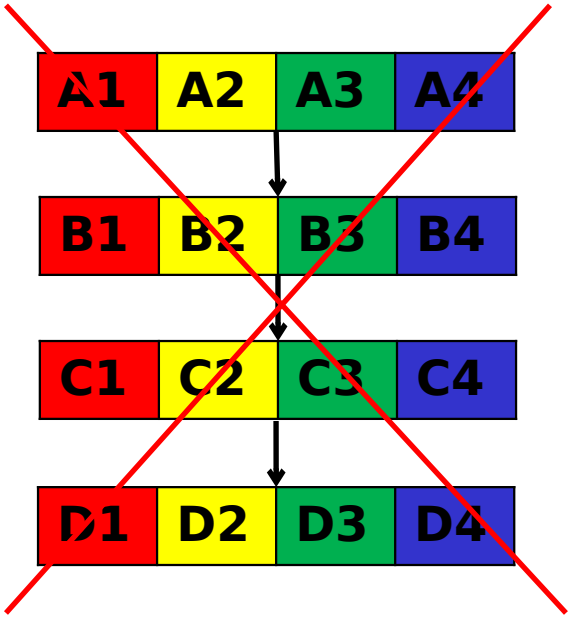


- It can be applied even if inner loops cannot be parallelized
- Operations of outer loops are executed simultaneously
- Operations of inner loops are executed in parallel only if they belong to different instances of outer loop

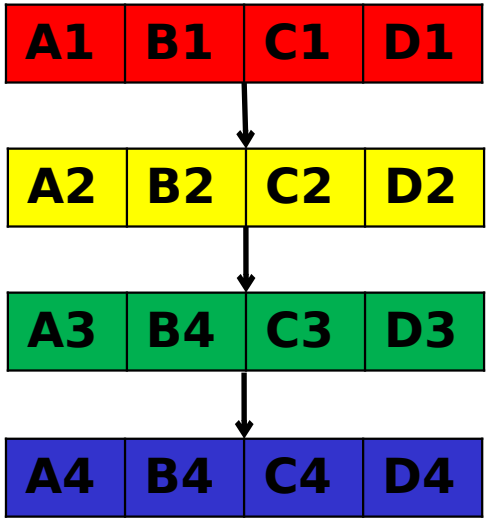
Outer Loops Vectorization


```
for(r=0; r<N; r++)
   for(c=0; c<N; c++)
      if(c > 0)
         out [r][c]=out[r][c-1]+in[r][c];
      else
         out[r][c]=in[r][c];
```

A1	A2	A3	A4
B1	B2	B3	B4
C1	C2	C3	C4
D1	D2	D4	D4



Inner Loop Vectorization



Outer Loop Vectorization

Outer Loops Vectorization Constraints

- Outer Loop must be DoAll loop
- Number of iterations must be multiple of the degree of parallelism
  - Ad-hoc management of last iterations would be required
- Nested loops cannot contain code in mutual exclusions

  `if(condition) {…} else {…}`

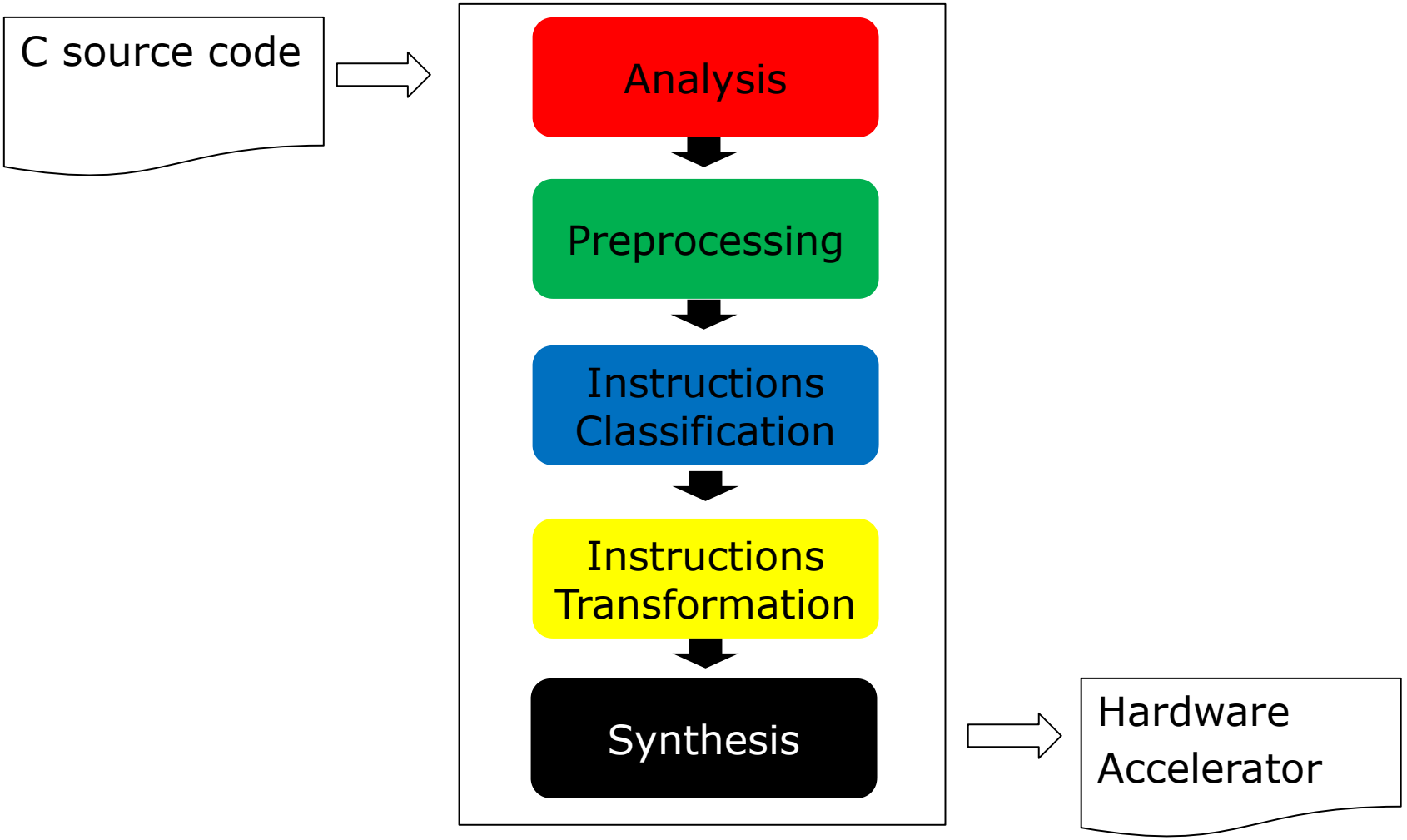
  is transformed into

  `if(condition) {…} if(!condition) {…}`
- Number of iterations of nested loops can depend on a value computed in outer loop
  - Nested loops can have an arbitrary number of iterations

Outer Loops Vectorization in High-Level Synthesis

- Vector functional units can be synthesized for all computation operations
- Advantages with respect to complete loop parallelization
  - Simpler Finite State Machine
    - Limits the area overhead
    - Limits the frequency reduction
  - Aligned memory accesses
    - Ad hoc memory allocation
  - Vector functional units can be obtained by
    - sharing scalar functional units
    - replicating scalar functional units
- Disadvantages
  - Larger memory infrastructure
  - More complex functional units

Proposed Design Flow



Proposed Design Flow: Analysis

- Identification of DoAll loops
  - Annotations (e.g., openMP pragmas)
  - Compiler analyses
  - Polyhedral analysis
  - Other analyses

```
#pragma omp simd
for(i=0; i<16; i++){
    sum=0;
    for(j=0; j<i; j++) {
        if(sum<10) {
            sum=sum+in[i][j];
        } else {
            sum=sum-in2[i][j];
    }
    res[i] = sum/k;
}
```

Proposed Design Flow: PreProcessing Transformation

- Removing of mutual exclusion code – potentially worsening performances
- Decomposition of complex operations
  - Vectorization can be applied selectively to the different parts of complex operations

```
#pragma omp simd
for(i=0; i<16; i++){
    sum=0;
    for(j=0; j<k; j++) {
        c = sum<10;
        if(c) {
            temp=in[i][j];
            sumS=sum+temp;
        }
        if(!c) {
            temp=in2[i][j];
            sumS=sum-temp;
        }
    }
    res[i] = sum/k;
}
```

Proposed Design Flow: Instructions Classification

- Vector instructions
- Multiscalar instructions
  - Cannot implemented as vector or
  - Too large to be synthesized as vector instruction
- DoAll loop instructions
- Nested loop instructions

```
#pragma omp simd
for(i=0; i<16; i++){
    sum=0;
    for(j=0; j<k; j++) {
        c = sum<10;
        if(c) {
            temp=in[i][j];
            sumS=sum+temp;
        }
        if(!c) {
            temp=in2[i][j];
            sumS=sum-temp;
        }
    }
    res[i] = sum/k;
}
```

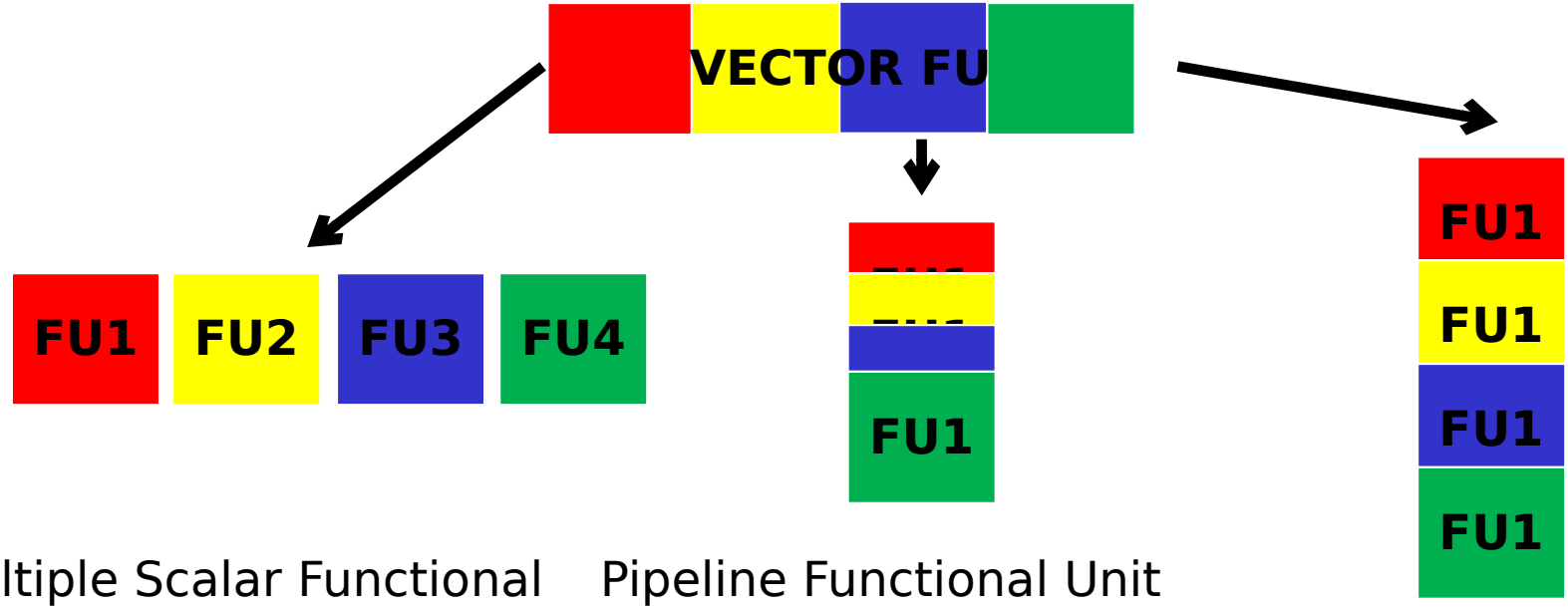
Proposed Flow: Instructions Transformation

- Vector instructions
  - Transformed in a single vector instruction
- Multiscalar instructions
  - Transformed in N scalar instructions
- DoAll loop instructions
  - Increment is fixed
- Nested loop instructions
  - Operands are fixed

```
#pragma omp simd
for(i={0,1}; i<16; i+={2,2}){
   sum={0,0};
   for(j=0; j<k; j++) {
      c = sum<{10,10};
      if(c[0] or c[1]) {
         temp[0]=in[i[0]][j];   (c[0])
         temp[1]=in[i[1]][j];   (c[1])
         sumS=sum+temp;   (c[0], c[1])
      }
      if(!(c[0] or c[1])) {
         temp[0]=in2[i[0]][j];(!c[0])
         temp[1]=in2[i[0]][j];(!c[1])
         sumS=sum-temp; (!c[0],!c[1])
      }
   }
   res[i[0]] = sum[0]/k;
   res[i[1]] = sum[1]/k;
}
```

Proposed Design Flow: Synthesis

- High Level Synthesis starting from transformed IR



Multiple Scalar Functional Units
+ Very good performances
+ Shared functional units
- Larger area

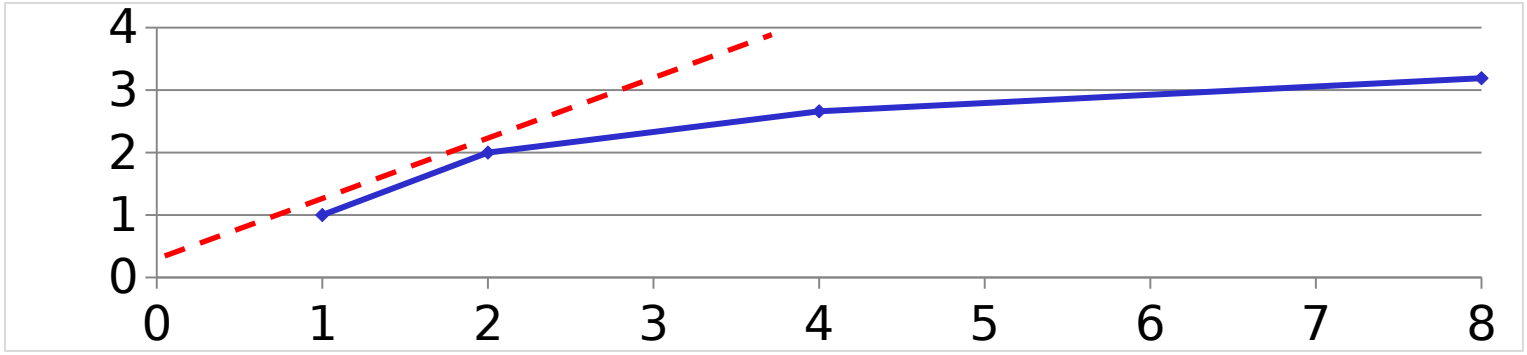
Pipeline Functional Unit
+ Good performances
- More Complex design

Scalar Functional Unit
+ Good area
- Worst performances

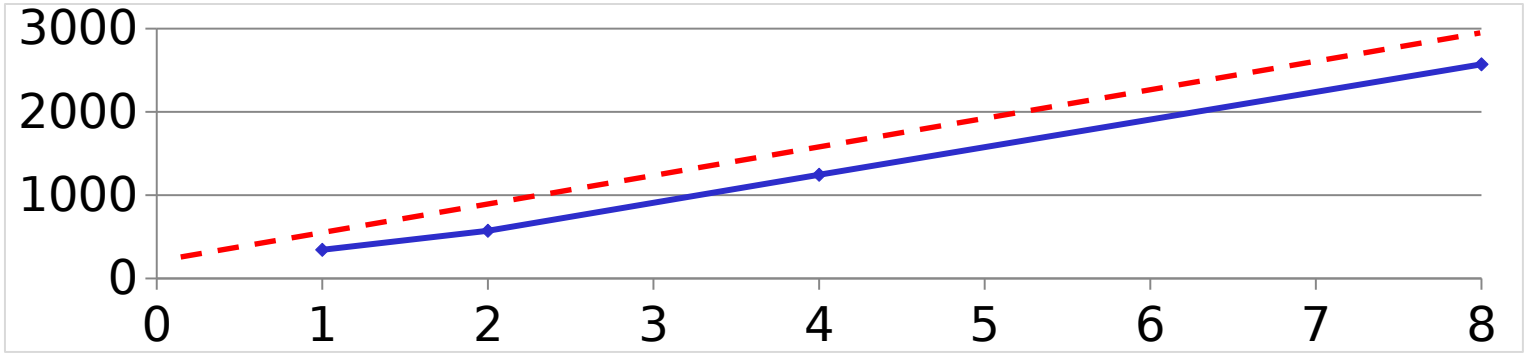
Experimental Evaluation

- Parallel benchmarks for High Level Synthesis annotated with #pragma omp simd
- Parallel degrees considered for vectorization: 1 2 4 8
- Target devices:
  - Xilinx Zynq-7000 xc7z020
  - Altera Cyclone II EP2C70F896C6
- Results:
  - Max speedup: 7.35x
  - Max reduction of area-delay product: 40%

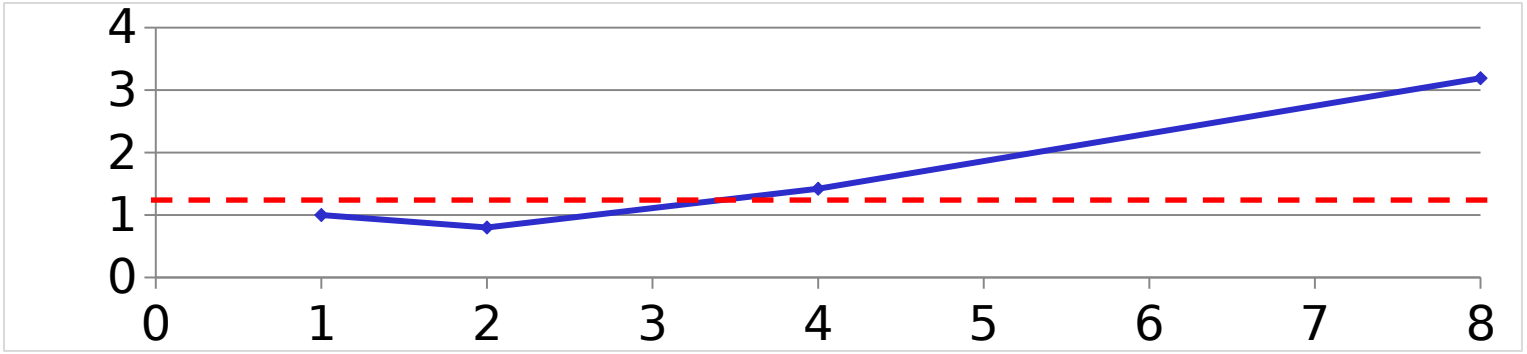
Experimental Evaluation: Case Study : Add



Speed-Up



Area (LUT-FF)



Area-Delay Product

Conclusions

- Vectorization of Outer Loops can be integrated in High Level Synthesis Flow
  - Synthesis of vector functional unit is required
- Memory accesses are the bottleneck which prevents obtaining maximum speedup
  - Different memory allocation is required