POLITECNICO DI MILANO

The 28th International Conference on Parallel Architectures and Compilation Techniques

Seattle, WA, USA, September 21st, 2019

PACT19 Tutorial

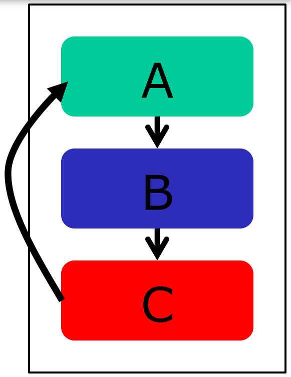
Exploiting Vectorization in High Level Synthesis of Nested Irregular Loops

Marco Lattuada

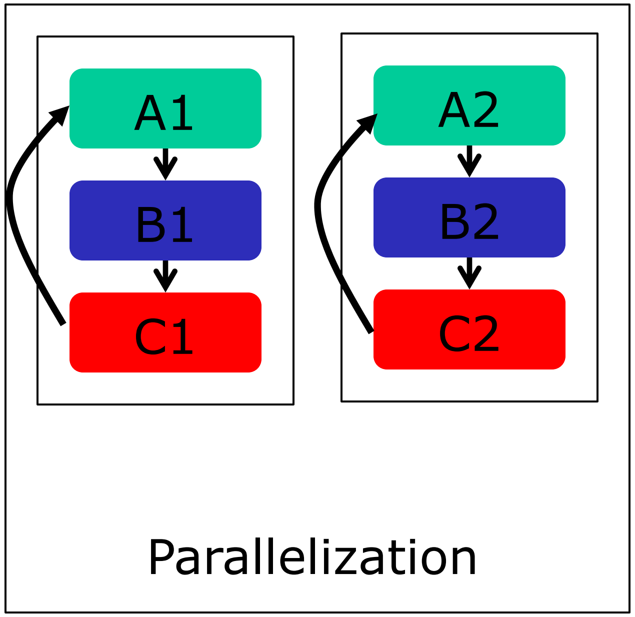
Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
marco.lattuada@polimi.it

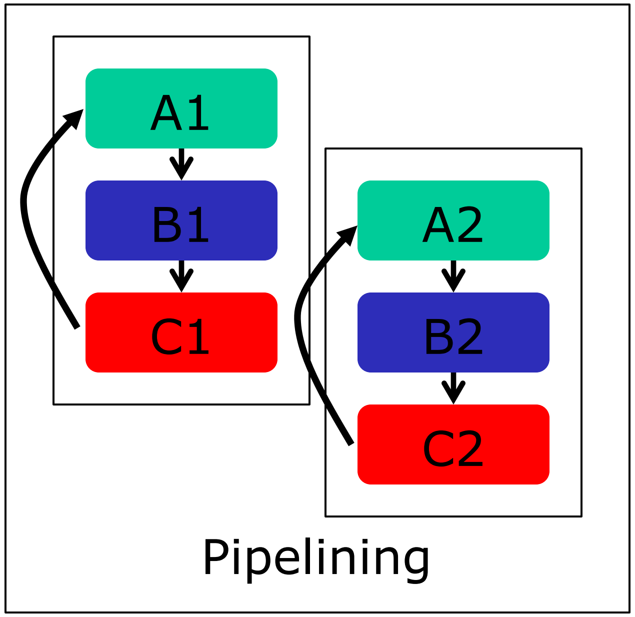
Outline

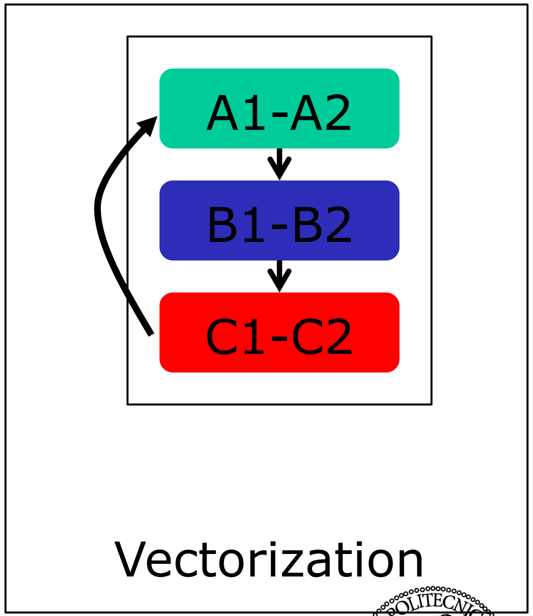
DoAll Loops and their Optimizations



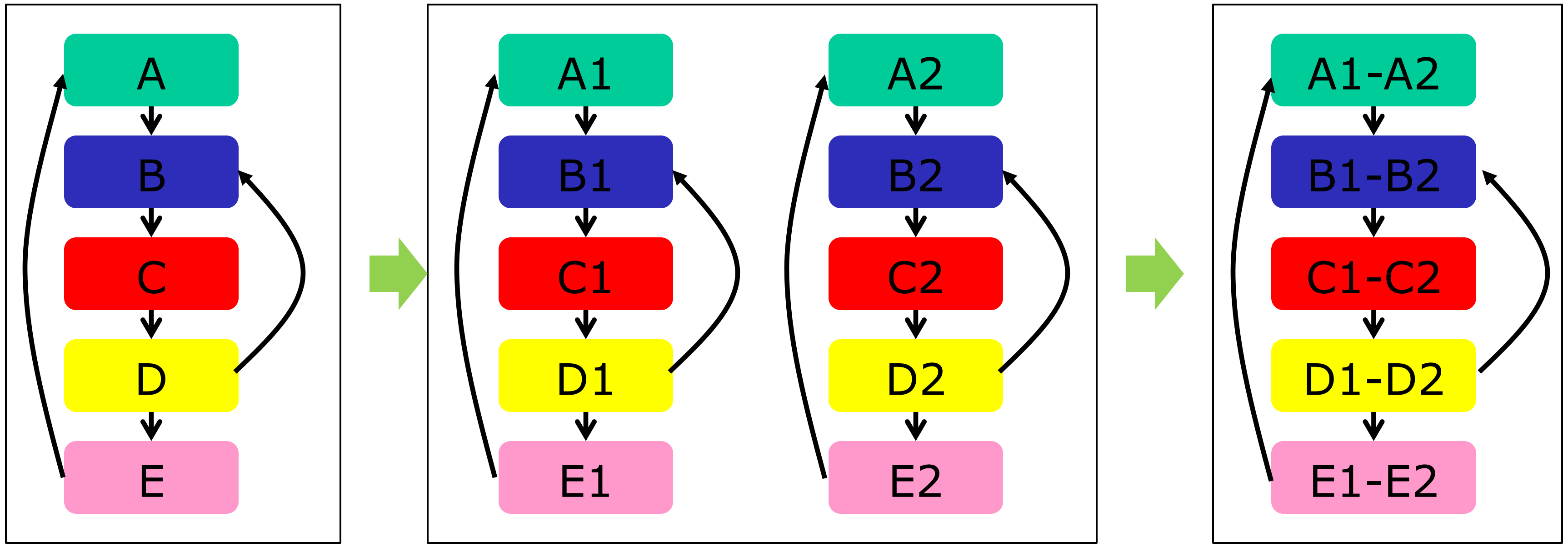
- DoAll loops
  - All the iterations can be executed in parallel
  - Number of iterations can be unknown at compile time
- Optimizations







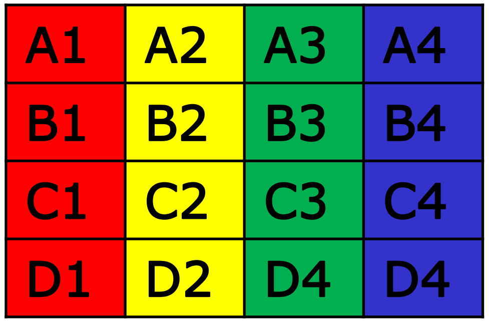
Outer Loops Vectorization

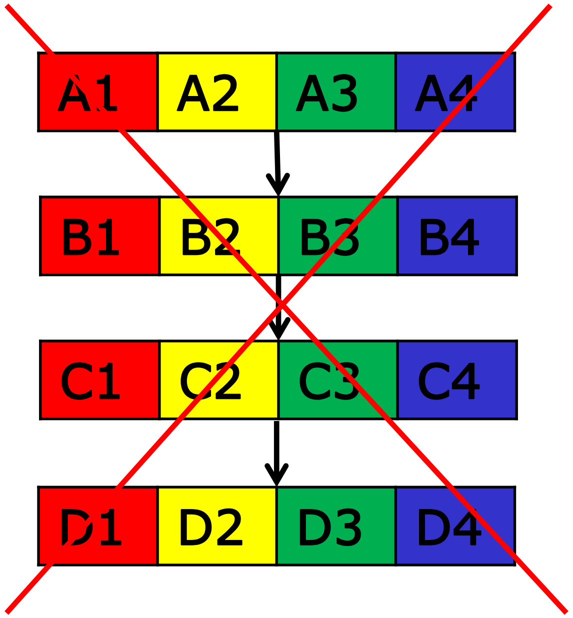


- It can be applied even if inner loops cannot be parallelized
- Operations of outer loops are executed simultaneously
- Operations of inner loops are parallelized only if they belong to different instances of outer loop

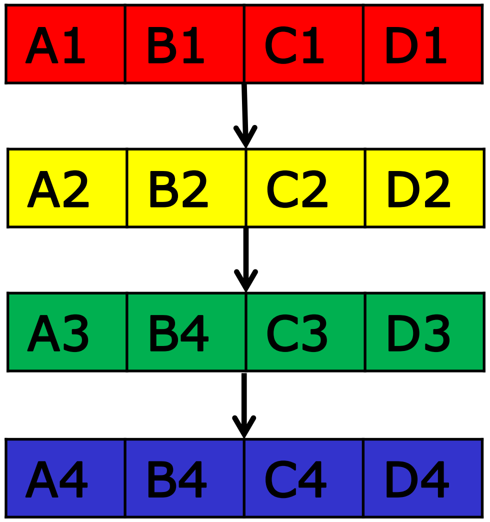
Outer Loops Vectorization

```
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];
```





Inner Loop
Vectorization



Outer Loop
Vectorization

Scenario

- 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

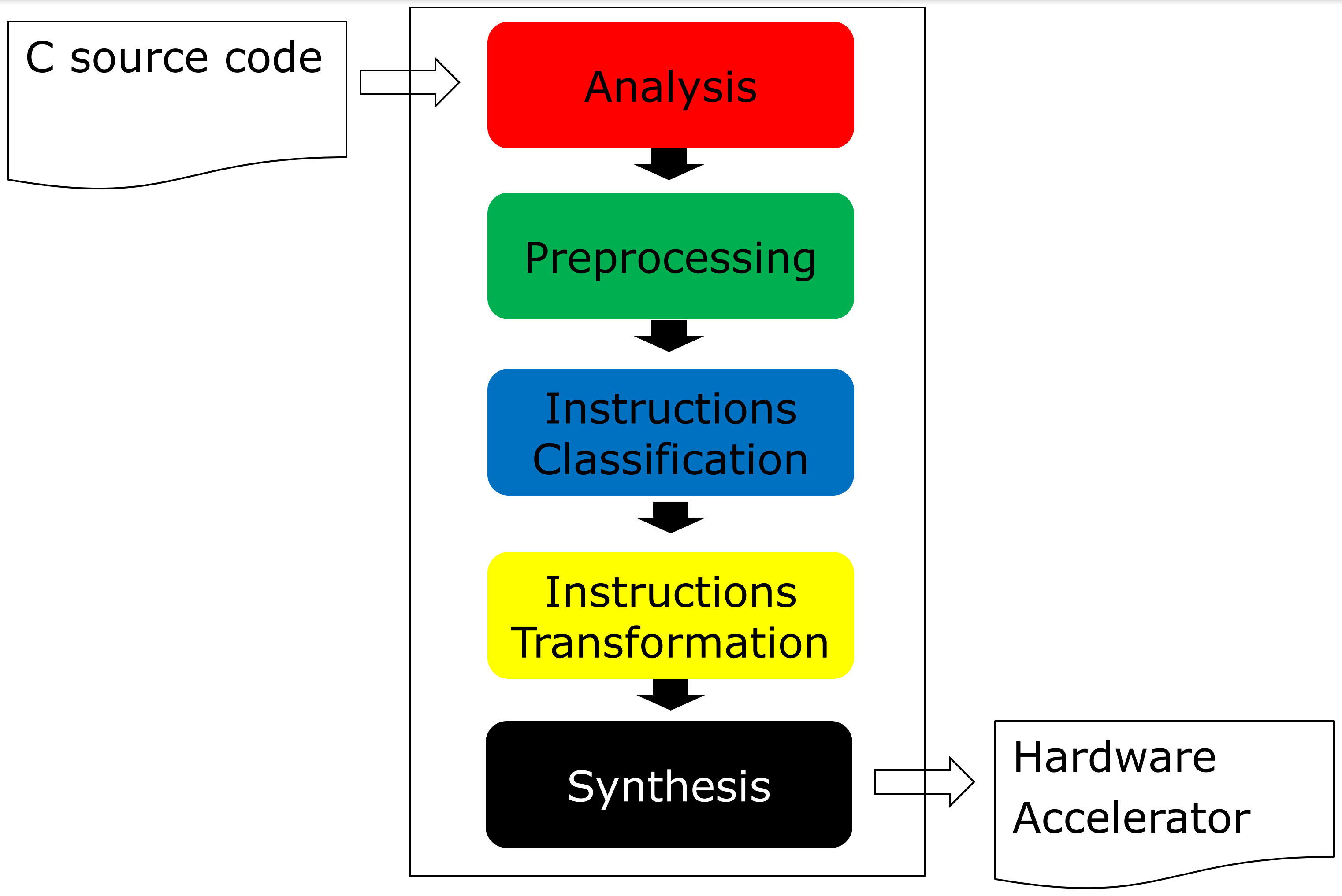
  ```
  if(condition) {…} if(!condition) {…}
  ```

- Number of iterations of nested loops can depend on a value computed in outer loop
  - Nested loops can be executed arbitrary number of times

Outer Loops Vectorization in High Level Synthesis

- Vector functional units can be synthesized for all the 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 flow



Proposed Flow: Analysis

- Identification of DoAll loops
  - Annotations (e.g., openMP pragmas)
  - Compiler analyses
  - Polyhedral analysis
  - Other analyses
- Code transformation
  - Removal of mutual exclusion code

```
#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 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 Flow: Instruction 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;
}
```

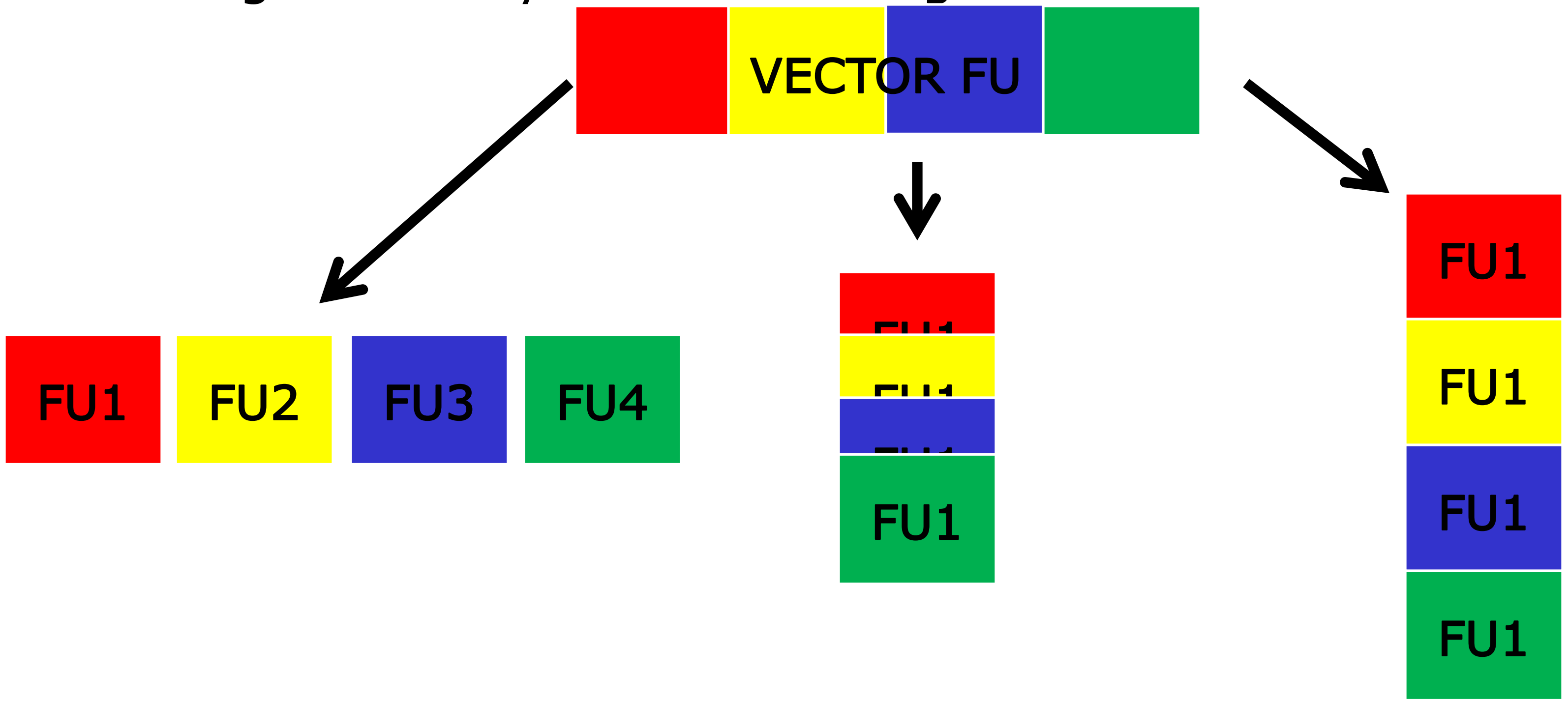
Proposed Flow: Instructions Transformation

- 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 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

First example – Code Transformation

- Generate an accelerator with vector size equal to one to see the effect of code transformation

Solution

```
bambu --compiler=I386_GCC49 --device-
name=5SGXEA7N2F45C1 --simulate -fwhole-program -fno-
delete-null-pointer-checks --clock-period=10 --experimental-
setup=BAMBU-BALANCED-MP -fdisable-tree-cunroll -fdisable-
tree-ivopts --param max-inline-insns-auto=1000  histogram.c
-fopenmp-simd=1 --pretty-print=output.c
```

Second example – Vector Size=4

- Generate an accelerator with vector size equal to 4 to evaluate the speed-up

Solution

```
bambu --compiler=I386_GCC49 --device-name=5SGXEA7N2F45C1 --simulate -fwhole-program -fno-delete-null-pointer-checks --clock-period=10 --experimental-setup=BAMBU-BALANCED-MP -fdisable-tree-cunroll -fdisable-tree-ivopts --param max-inline-insns-auto=1000  histogram.c -fopenmp-simd=4
```

Third Example – Different Vector Size

- Generate accelerators with vector size equal to 2,3,4,8 to evaluate the speedup

Solution

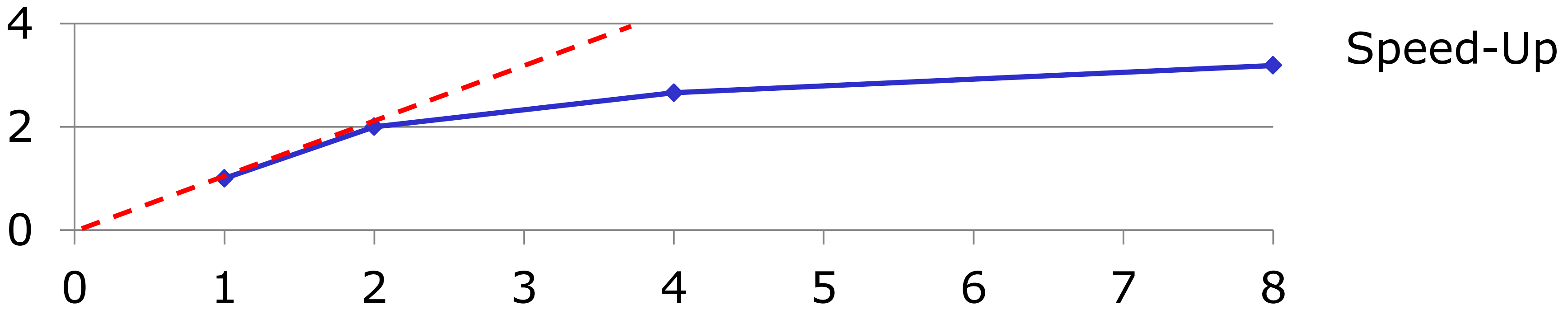
```
bambu --compiler=I386_GCC49 --device-name=5SGXEA7N2F45C1 --simulate -fwhole-program -fno-delete-null-pointer-checks --clock-period=10 --experimental-setup=BAMBU-BALANCED-MP -fdisable-tree-cunroll -fdisable-tree-ivopts --param max-inline-insns-auto=1000  histogram.c
-fopenmp-simd=2 --pretty-print=output.c
```

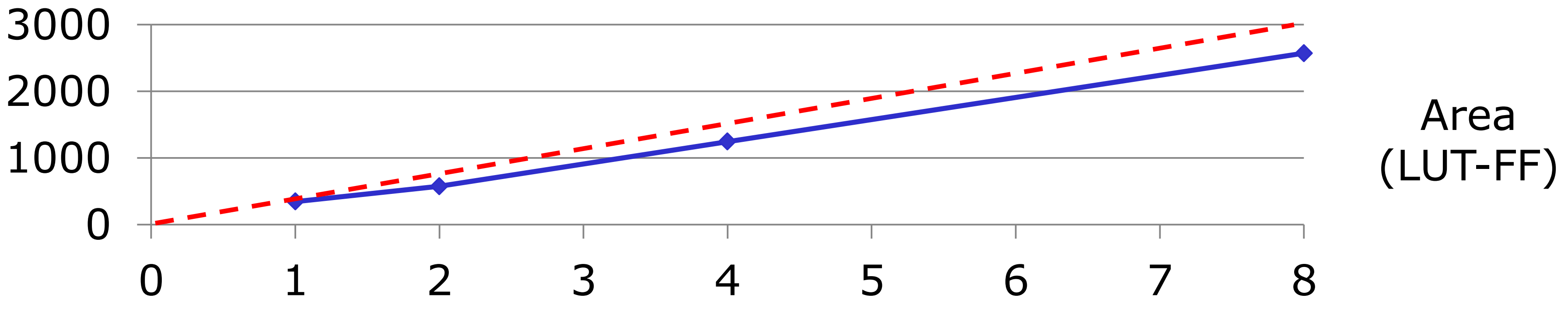
Etc.

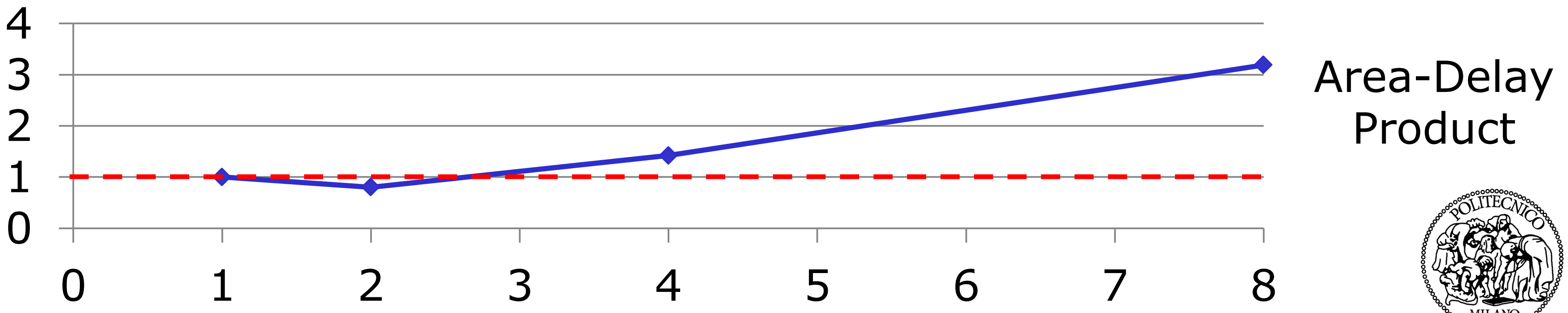
Experimental Evaluation

- 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







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