DATE 19

MARCH 25—29, 2019

FLORENCE, ITALY

FIRENZE FIERA

DESIGN, AUTOMATION AND TEST IN EUROPE THE EUROPEAN EVENT FOR ELECTRONIC SYSTEM DESIGN & TEST

AUTOMATIC CODE RESTRUCTURING FOR FPGAS: CURRENT STATUS, TRENDS AND OPEN ISSUES

Special Day on "Embedded Meets Hyperscale and HPC"

João MP Cardoso

jmpc@acm.org

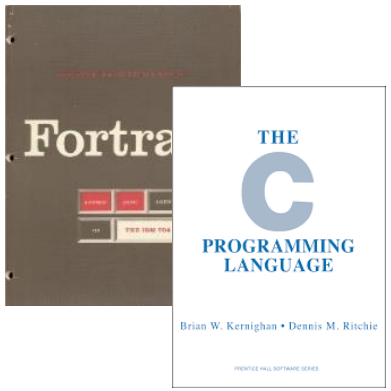
U.PORTO

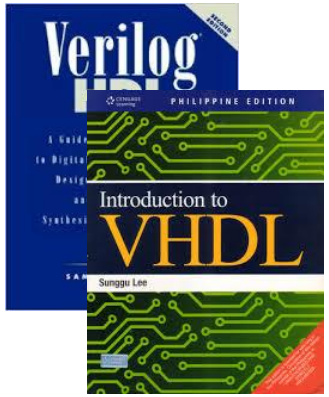
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

INESCTEC TECNOLOGIA E CIÊNCIA | LABORATÓRIO ASSOCIADO

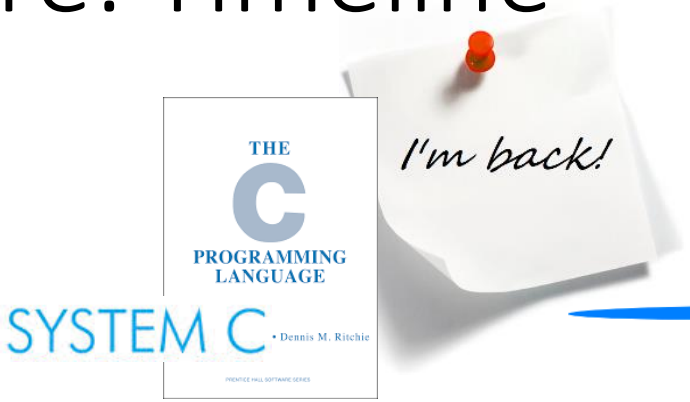
SPeCS Special-Purpose Computing Systems, languages and tools

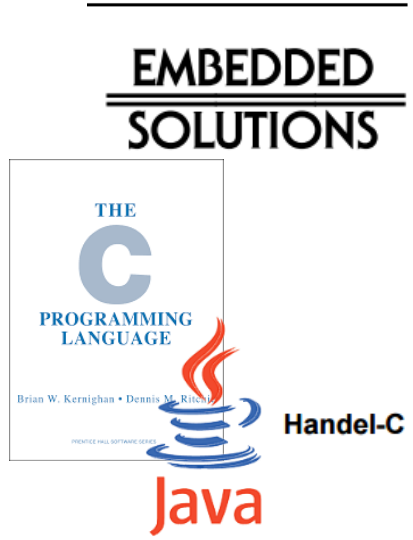
Compiling to hardware: Timeline

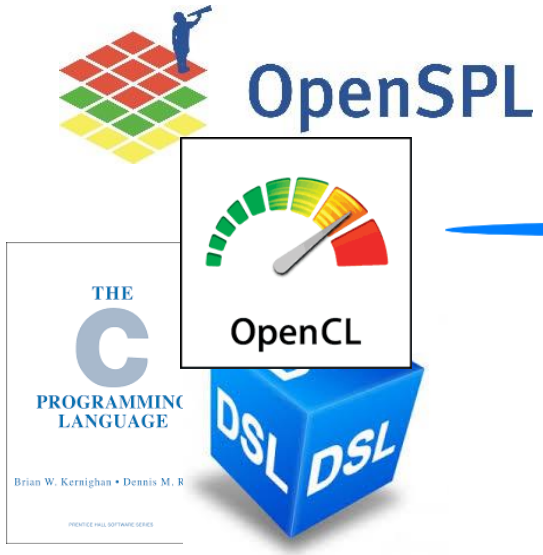




…



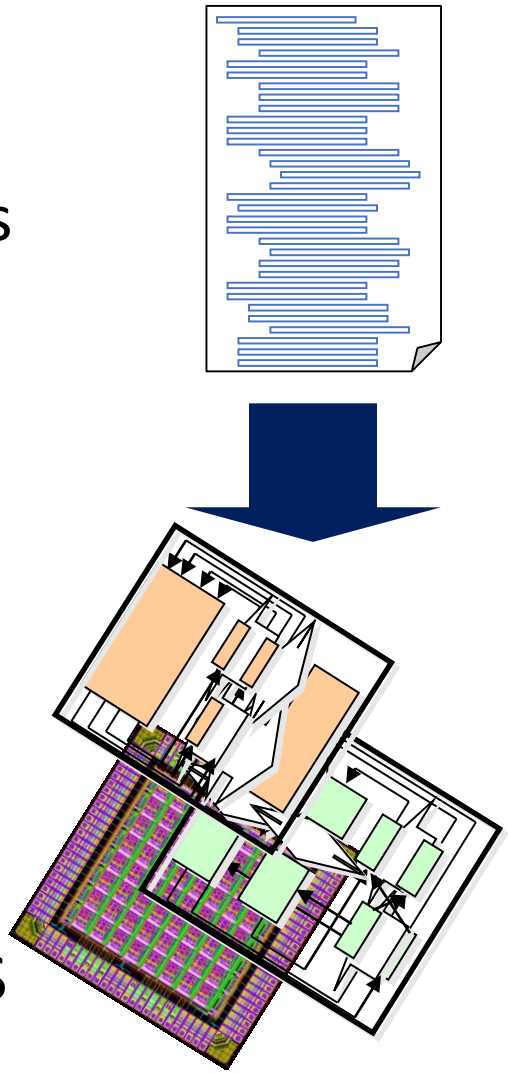




80’s | 90’s | 00’s | 10’s | 20’

Compiling to FPGAs (hardware)

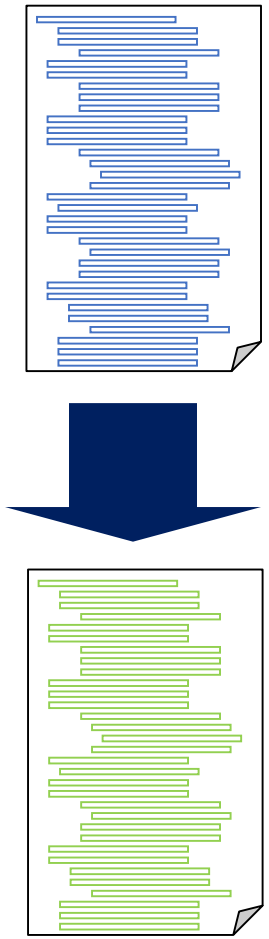
- Of paramount importance for allowing software developers to map computations to FPGA-based accelerators
  - Efficient compilation will improve designer productivity and will make the use of FPGA technology viable for **software programmers**
- Challenge:
  - Added complexity of the extensive set of execution models supported by FPGAs makes efficient compilation (and programming) very hard
- Years of research on High-Level Synthesis (mostly on hardware generation from C) and adoption of mature compiler frameworks are resulting in the effective use of HLS



Outline

- Intro
- Why source to source compilers?
- Code restructuring
- Some approaches for code restructuring
- Our ongoing work
- Conclusion
- Future work

Why source to source compilers?

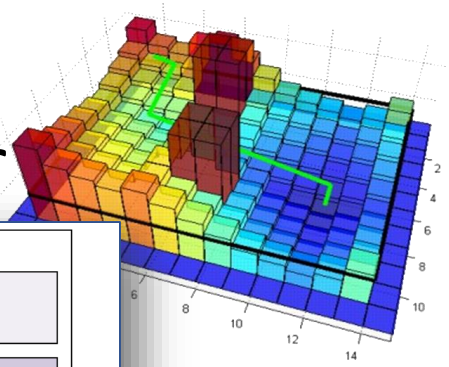


- There are many optimizations and code transformations that can be explored at the source code level
- Target code is still legible
- Not tied to a specific target compiler (tool flow) or target Architecture!

But:

- Not all optimizations can be done at source code level!
- Some code transformations are too specific and without enough application potential to justify inclusion in a compiler (unless the code is too important and must be regularly used/modified/extended)

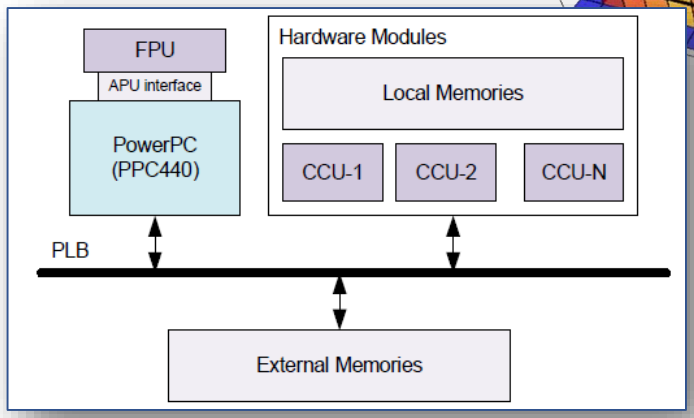
Source level code transf.: 3D Path Planner



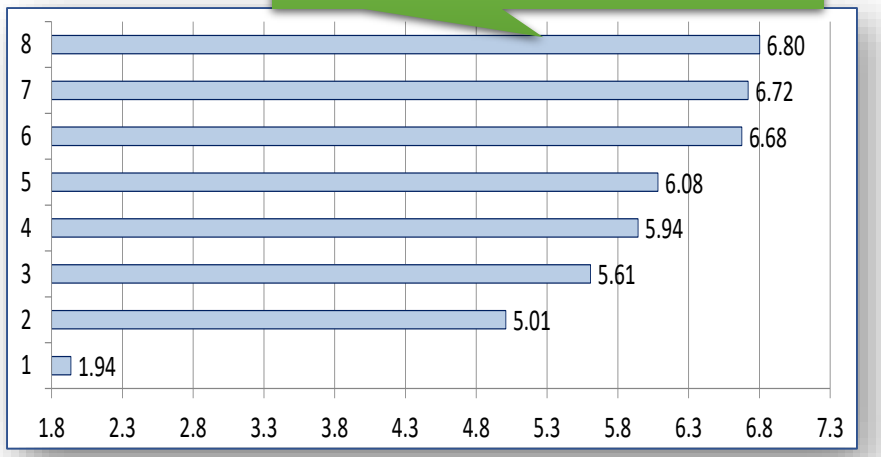
- Target: ML507 Xilinx Virtex-5 board, PowerPC@400 MHz, CCUs@100 MHz

Optimization	Strategy							
	1	2	3	4	5	6	7	8
Loop fission and move		✓	✓	✓	✓	✓	✓	✓
Replicate array 3×					✓	✓	✓	✓
Map gridit to HW core	✓	✓	✓	✓	✓	✓	✓	✓
Pointer-based accesses and strength reduction			✓	✓	✓	✓	✓	✓
Unroll 2×	✓	✓	✓	✓	✓	✓	✓	✓
Eliminating array accesses	✓	✓	✓	✓	✓	✓	✓	✓
Move data access								✓
Specialization → 3 HW cores							✓	✓
Transfer pot data according to gridit call				✓		✓	✓	✓
Transfer obstacles data according to gridit call			✓	✓	✓	✓	✓	✓
On-demand c								

FPGA resources	Implementation			
	1	2,3,4	5,6	7,8
# Slice Registers as FF	901	939	956	2,470
# Slice LUTs	1,182	1,284	1,308	2,148
# occupied Slices	531	663	642	1,004
# BlockRAM/# DSP48Es	34/6	34/6	98/6	98/12



Strategy 8: 6.8 × faster than pure software solution



Source: EU-Funded FP7 REFLECT project

See: Cardoso et al., **Specifying Compiler Strategies for FPGA-based Systems.** FCCM 2012

Simple code restructuring example

An FIR

Code restructuring: FIR example

```
// x is an input array
// y is an output array
#define c0 2, c1 4, c2 4, c3 2
#define M 256 // no. of samples
#define N 4 // no. of coeff.
int c[N] = {c0, c1, c2, c3};
...
// Loop 1:
for(int j=N-1; j<M; j++) {
   output=0;
   // Loop 2:
   for(int i=0; i<N; i++) {
      output+=c[i]*x[j-i];
   }
   y[j] = output;
}
```

Code restructuring: FIR example

```
// x is an input array
// y is an output array
#define c0 2, c1 4, c2 4, c3 2
#define M 256 // no. of samples
#define N 4 // no. of coeff.
int c[N] = {c0, c1, c2, c3};
...
// Loop 1:
for(int j=N-1; j<M; j++) {
   output=0;
   // Loop 2:
   for(int i=0; i<N; i++) {
      output+=c[i]*x[j-i];
   }
   y[j] = output;
}
```

```
// Loop 1
for(int j=3; j<M; j++) {
   x_3=x[j];
   x_2=x[j-1];
   x_1=x[j-2];
   x_0=x[j-3];
   output=c0*x_3;
   output+=c1*x_2;
   output+=c2*x_1;
   output+=c3*x_0;
   y[j] = output;
}
```

II=2

1 sample per 2 clock cycles

Code restructuring: FIR example

```
// x is an input array
// y is an output array
#define c0 2, c1 4, c2 4, c3 2
#define M 256 // no. of samples
#define N 4 // no. of coeff.
int c[N] = {c0, c1, c2, c3};
...
// Loop 1:
for(int j=N-1; j<M; j++) {
   output=0;
   // Loop 2:
   for(int i=0; i<N; i++) {
      output+=c[i]*x[j-i];
   }
   y[j] = output;
}
```

1

```
// Loop 1
for(int j=3; j<M; j++) {
   x_3=x[j];
   x_2=x[j-1];
   x_1=x[j-2];
   x_0=x[j-3];
   output=c0*x_3;
   output+=c1*x_2;
   output+=c2*x_1;
   output+=c3*x_0;
   y[j] = output;
}
```

II=2

1 sample per 2 clock cycles

2

```
x_0=x[0];
x_1=x[1];
x_2=x[2];
// Loop 1
for(int j=3; j<M; j++) {
   x_3=x[j];
   output=c0*x_3;
   output+=c1*x_2;
   output+=c2*x_1;
   output+=c3*x_0;
   x_0=x_1;
   x_1=x_2;
   x_2=x_3;
   y[j] = output;
}
```

II=1

1 sample per clock cycle

Code restructuring: FIR example

1

II=2

```
// Loop 1
for(int j=3; j<M; j++) {
    x_3=x[j];
    x_2=x[j-1];
    x_1=x[j-2];
    x_0=x[j-3];
    output=c0*x_3;
    output+=c1*x_2;
    output+=c2*x_1;
    output+=c3*x_0;
    y[j] = output;
}
```

1 sample per 2 clock cycles

2

```
x_0=x[0];
x_1=x[1];
x_2=x[2];
// Loop 1
for(int j=3; j<M; j++) {
    x_3=x[j];
    output=c0*x_3;
    output+=c1*x_2;
    output+=c2*x_1;
    output+=c3*x_0;
    x_0=x_1;
    x_1=x_2;
    x_2=x_3;
    y[j] = output;
}
```

II=1

1 sample per clock cycle

3

II=1

```
// Loop 1
for(int j=3; j<M; j+=2) {
    x_3=x[j];
    output=c0*x_3;
    output+=c1*x_2;
    output+=c2*x_1;
    output+=c3*x_0;
    x_0=x_1;
    x_1=x_2;
    x_2=x_3;
    y[j] = output;
    x_3=x[j+1];
    output=c0*x_3;
    output+=c1*x_2;
    output+=c2*x_1;
    output+=c3*x_0;
    x_0=x_1;
    x_1=x_2;
    x_2=x_3;
    y[j+1] = output;
}
```

2 samples per clock cycle

See: João M. P. Cardoso, Markus Weinhardt, **High-Level Synthesis.** FPGAs for Software Programmers 2016.

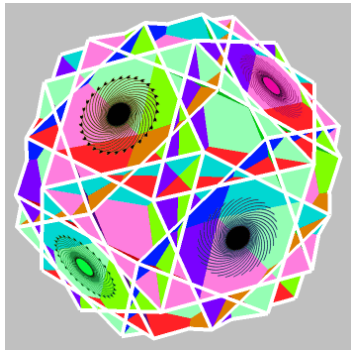
Code restructuring

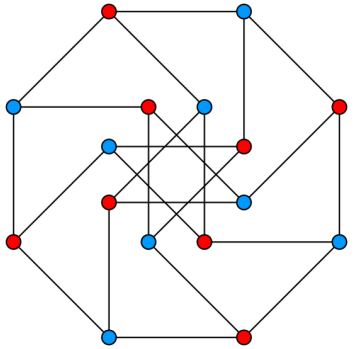
- Manual
  - Programmers need to know the impact of code styles and structures on the generated architecture – with similarities to the HDL developers, although in a different level
- Fully automatic with a source-to-source compiler (refactoring tool)
  - Need to devise the code transformations to apply and their ordering
  - Need source to source compilers integrating a vast portfolio of code transformations
- Semi-automatic with a source-to-source compiler (refactoring tool)
  - Code transformations automatically applied but guided by users
  - Users can define their own code transformations

Some approaches for code restructuring/opt.

- Flag selection
- Phase ordering
- Polyhedral models
- Graph-based transformations

- LegUp [Canis et al., ACM TECS'13]: flag selection and phase ordering (via LLVM + opt) [Huang et al., ACM TRETS'15]
- The Merlin Compiler and source to source optimizations by Cong et.al., FSP'16
- Polyhedral transformations by Zuo et al., FPGA'13
- Polyhedral in nested loop pipelining by Morvan et al., IEEE TCAD'13
- Graph-based code restructuring by Ferreira and Cardoso, FSP'18, ARC'19





Flag selection

- Generation controlled by enabling/disabling compiler flags – sequence of optimizations are the ones built-in and pre-fixed for each flag
- Suitable to most common approaches, but without taking full-advantage of customization/specialization

Helping but without solving the code restructuring problem!

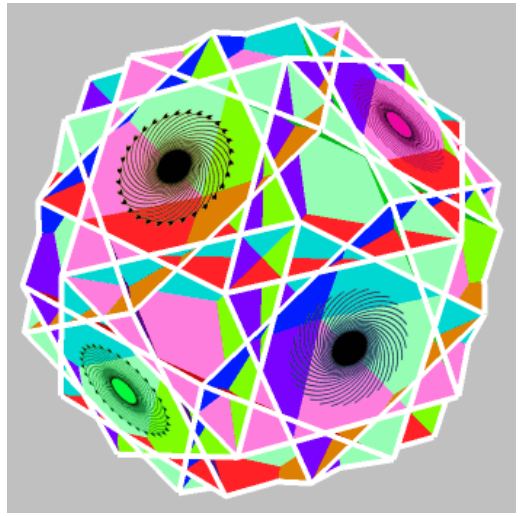
Phase ordering

- Providing specific sequences of compiler optimizations
- Problem is very complex as besides selecting the phases one needs to provide sequences – usually repeating phases
- Difficult to find the sequence!
- Fully dependent on the portfolio of phases a compiler may include – phases need to justify their inclusion (i.e., if they pay-off)

Limitations for solving the code restructuring problem!

Polyhedral models

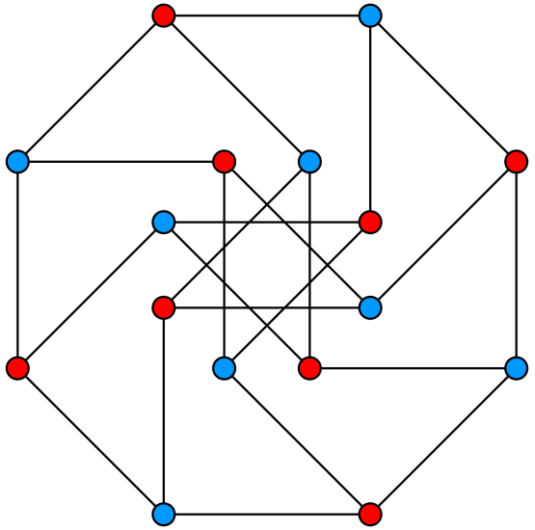
- Applied to *Static Control Parts* – require specific loop structures, statically known iteration spaces, limited to affine domains
- Pure polyhedral models transform iteration spaces – more advanced approaches combine the polyhedral model with AST transformations
- Able to provide useful code transformations and justify their inclusion in the portfolio of compiler optimizations



Helping on solving the code restructuring problem!

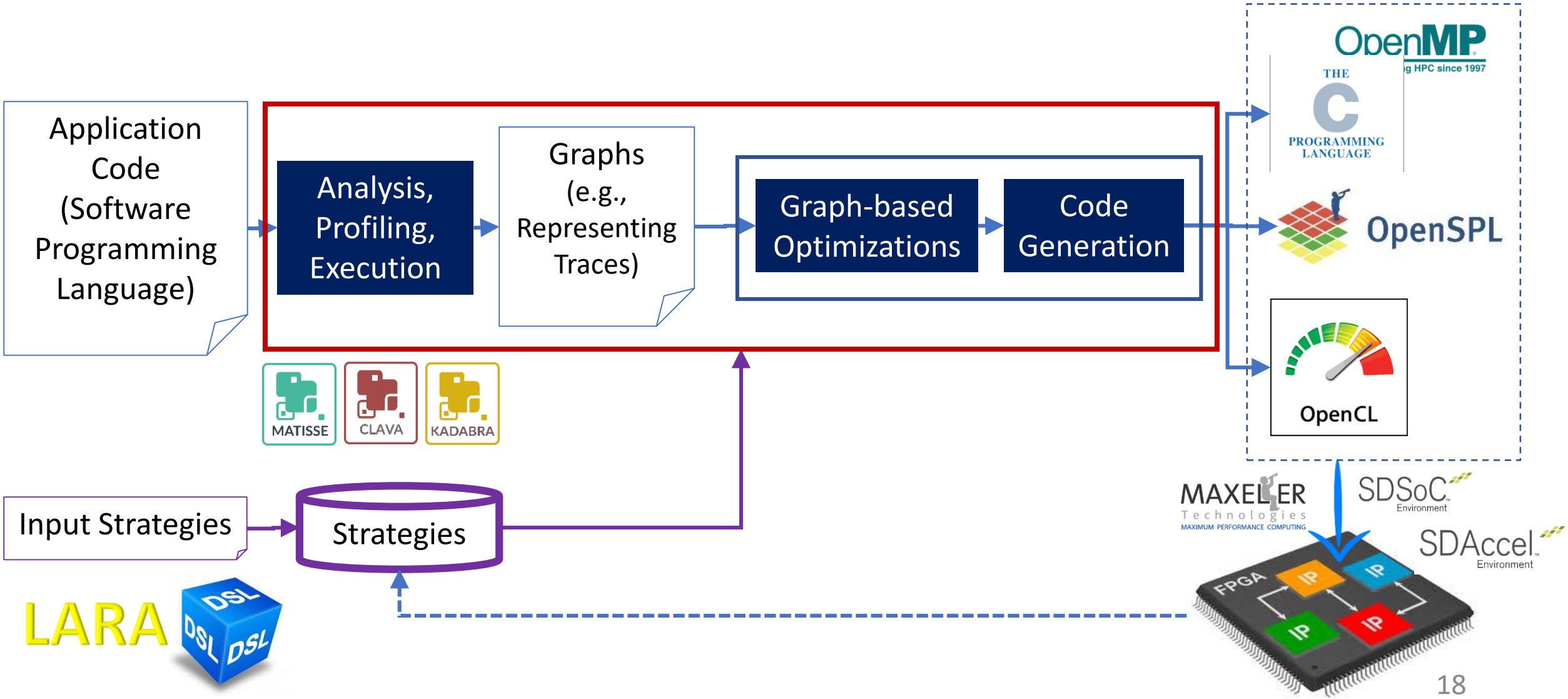
Graph-based transformations (our ongoing work)

- Traces of computations are represented in Dataflow Graphs (DFGs)
- Code restructuring problem is solved by graph transformations
- Able to achieve high-levels of code restructuring and suitable HLS directives

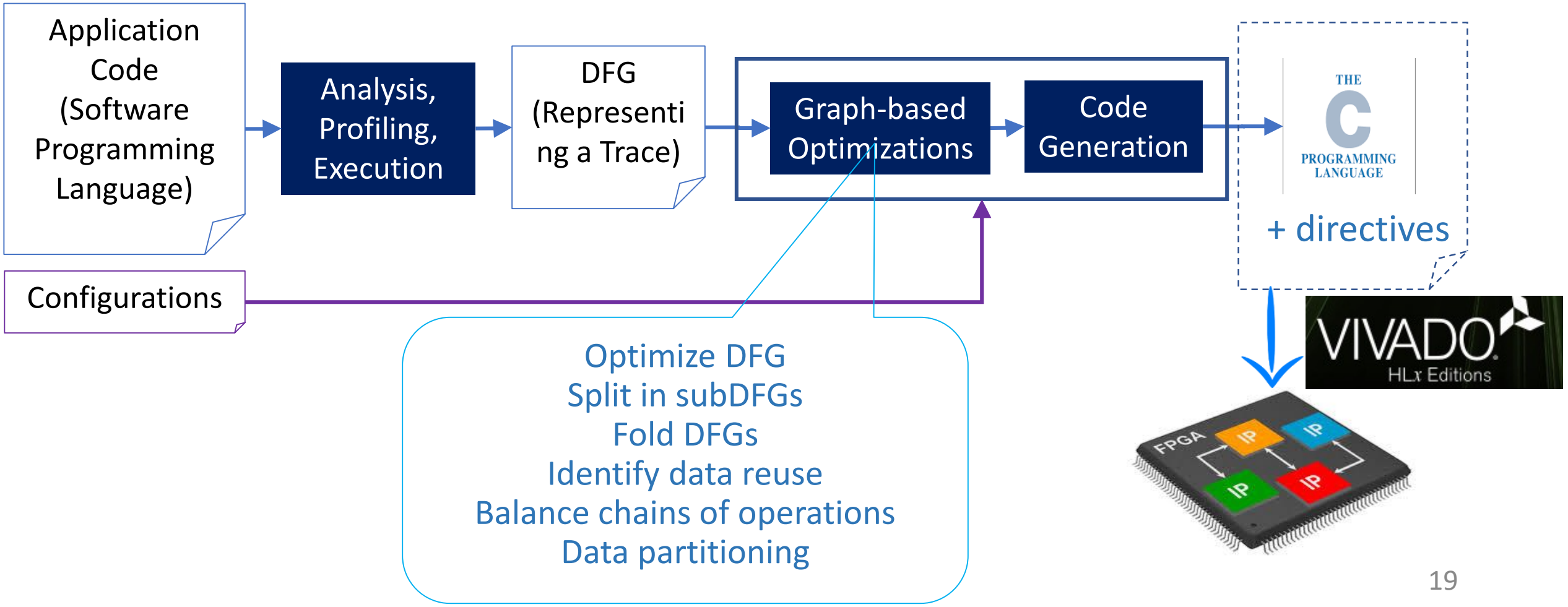


A proof of concept… scalability still needs to be solved!

Code restructuring: ongoing



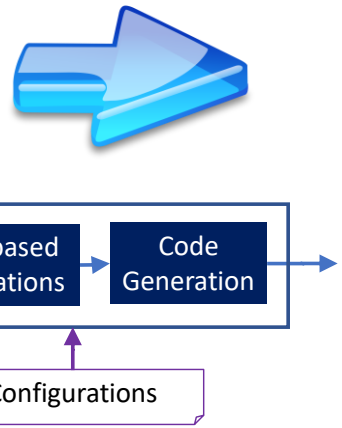
Code restructuring: graph-based approach



Example – filter subband

```
void filter_subband (double z[Nz], double
s[Ns], double m[Nm]){
   double y[Ny];
   int i,j;
   for (i=0;i<Ny;i++)
   {
     y[i] = 0.0;
     for (j=0; j<(int)Nz/Ny;j++)
          y[i] += z[i+Ny*j];
   }

   for (i=0;i<Ns;i++)
   {
     s[i]=0.0;
     for (j=0; j<Ny;j++)
        s[i] += m[Ns*i+j] * y[j];
   }
}
```




```
void result( double s[32], double z[512], double m[1024]){
#pragma HLS array_partition variable=s cyclic factor=16
#pragma HLS array_partition variable=z cyclic factor=16
#pragma HLS array_partition variable=m cyclic factor=64
s[0]=0;
…
s[31]=0;
  for( int i =0; i < 64; i=i+4){
  #pragma HLS pipeline
  partial_1_2 = z[i+320] + z[i+256];
  …
  y0 = final_partial_1;
  y0_a10 = final_partial_2;
  for( int j =0; j < 32; j=j+1){
     temp_1=m[(32)*j+i] * y0;
     temp_2=m[(32)*j+i+1] * y0_a10;
     …
     partial_in_1 = temp_1 + temp_2;
     partial_in_2 = temp_3 + temp_4;
     final_part_in = partial_in_1+ partial_in_2;
     s[j]=s[j] + final_part_in;
     }
  }
}
```

Source: Ferreira and Cardoso, ARC'2019

Experimental results

- Vivado HLS 2017.4
- Xilinx FPGA Artix-7 (xc7z020clg484-1)

Input	Description
C	Original code without modifications
C-inter	Input code optimized with basic directives such as pipelining
C-high	Improve C-inter with array partitioning and loop unrolling directives

Name	Speedup C	Speedup C-inter	Speedup C-high	Latency (#ccs)	Clock Period (ns)	#LUT	#FF	#DSP	#BRAM
Filter subband	81	5.8	5.8	293 (0.18)	17.1 (0.9)	47537 (7.1)	42589 (3.6)	118 (4.1)	0
Dotprod	16	5.6	1.0	255 (1)	8.9 (1.0)	294 (1.0)	581 (1.0)	8 (1.0)	0
Autocorrelation	297	98.6	47.5	16 (0.018)	8.6 (1.1)	8025 (4.0)	7114 (7.9)	160 (16.0)	0
1D FIR	237	30.0	16.2	120 (0.06)	8.7 (1)	4297 (0.9)	5641 (1.9)	192 (1.6)	0
2D Convolution	76	5.0	3.0	3886 (0.33)	8.7 (1)	6376 (1.2)	3408 (0.6)	57 (1.5)	0
SVM	123	3.5	3.5	3208 (0.28)	8.4 (1)	14203 (1.6)	12506 (1.6)	91 (1.6)	76 (1.11)

Source: Ferreira and Cardoso, ARC'2019

Ongoing and future work

- Comparisons to the approaches using the polyhedral model to restructure software code
- Scalability issues
  - How to avoid the need of explicit large graphs when dealing with large traces / loops with many iterations?
- Focus on optimizations regarding conditional paths
  - Use of different execution paths to create specialized accelerators and schemes to manage their execution at runtime
  - Merge of execution paths in order to avoid one specialized accelerator per execution path

Source: Ferreira and Cardoso, ARC'2019

Conclusion

- Source-to-source compilers as front-ends and HLS tools as the new backends for advanced compilation to FPGAs
- Compiling to FPGAs needs more efficient and aggressive code restructuring – a research challenge!
- Our recent efforts focus on an approach to optimize code for HLS based on unfolded graph representations and graph transformations – experimental results highlight the benefits of the approach
- A deeper study about code restructuring approaches needs to be done!

Thank you! Questions?

João MP Cardoso
jmpc@acm.org

U.PORTO
FEUP FACULDADE DE ENGENHARIA UNIVERSIDADE DO PORTO

INESCTEC
TECNOLOGIA E CIÊNCIA
| LABORATÓRIO ASSOCIADO

SPeCS
Special-Purpose Computing Systems, languages and tools

Acknowledgments

Afonso Ferreia
João Bispo
Pedro Pinto
Tiago Carvalho
Luís Reis

SMILES

CONTEXTWA

PhD schoolarships from FCT