Coupling Loop Transformations and High-Level Synthesis

Alexandru Pleşco
LIP, CNRS — ENS Lyon – UCB Lyon — Inria, France
alexandru.plesco@ens-lyon.fr

CNRS
Laboratoire de l'Informatique du Parallélisme
École normale supérieure de Lyon

18 octobre 2007

Presentation

Recent Trends in Embedded Systems

- Platform-based design
- Higher level of abstraction formalisms
- IP-reuse

Solution

- High-level synthesis (HLS) is used to accelerate dedicated hardware design and to provide a sufficient design abstraction level.

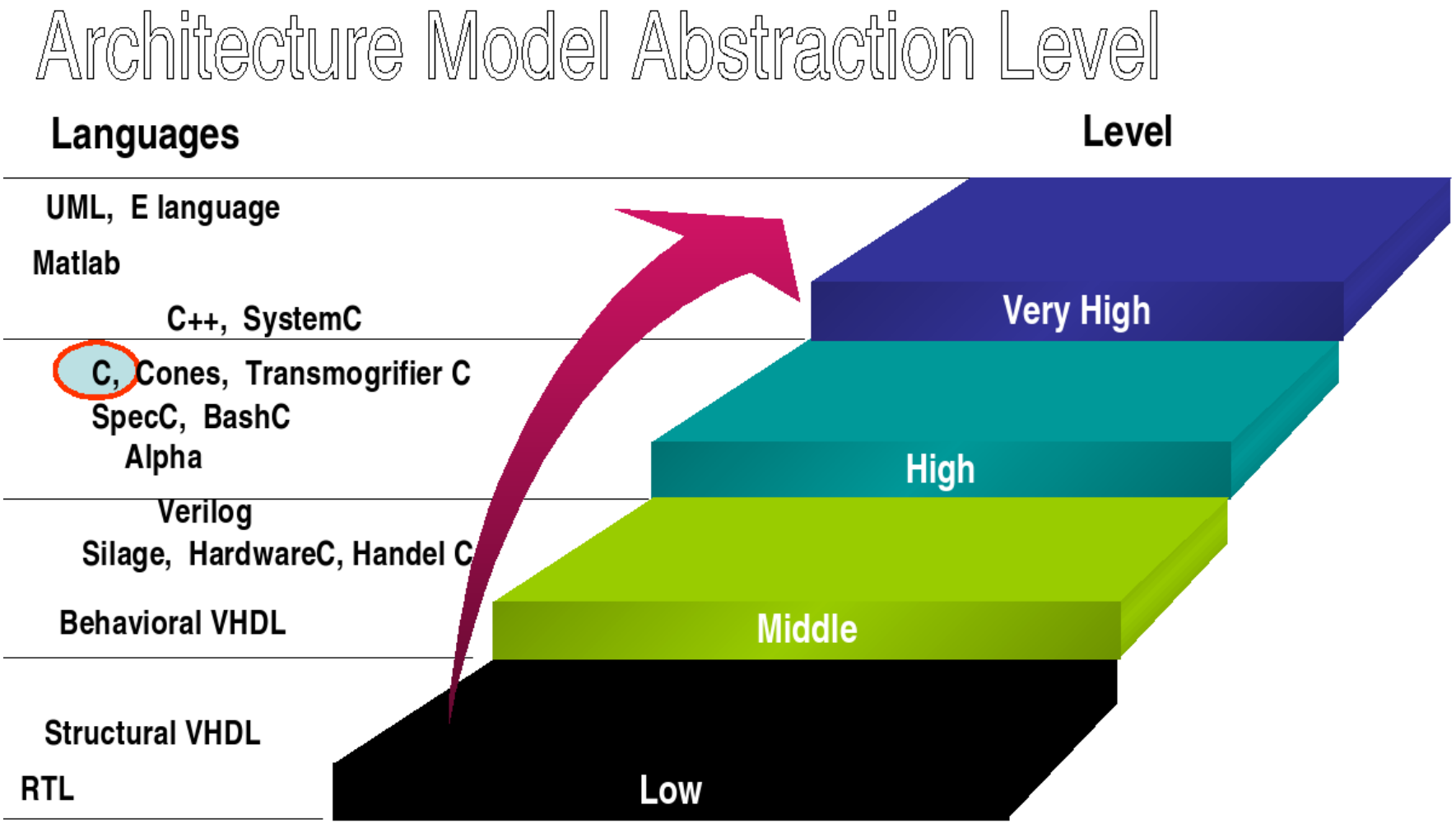
HLS technical problems

- Huge design exploration space (vairous parallelism, target architecture technology)
- Memory bottleneck (memory size and trafic optimizations)
- Efficient synthesis of loop nests (mostly found in power-hungry multimedia algorithms)

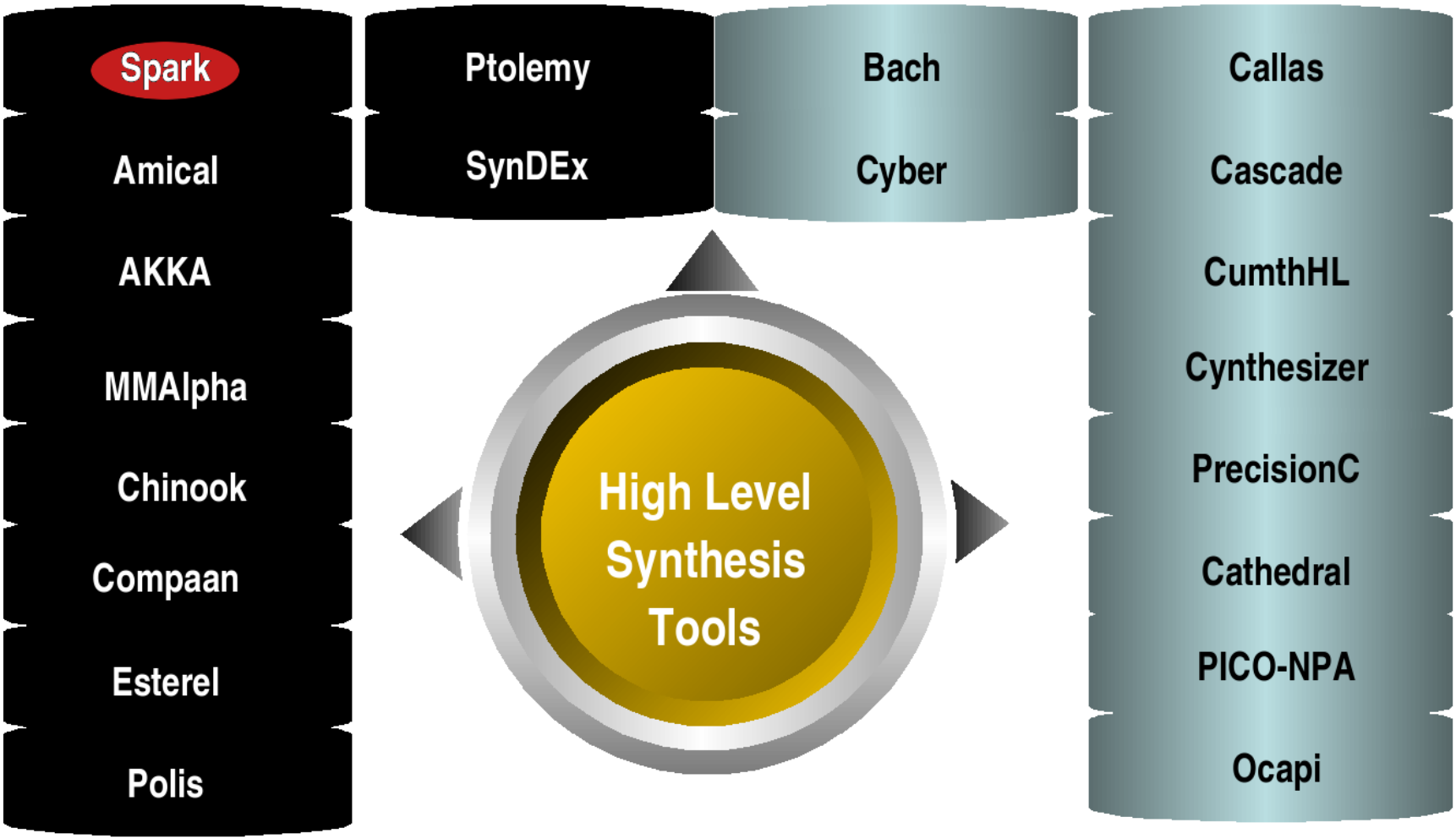
Preprocessing optimization step for HLS

Transform from the designer specification of the application to a specification suited for a specific HLS tool.

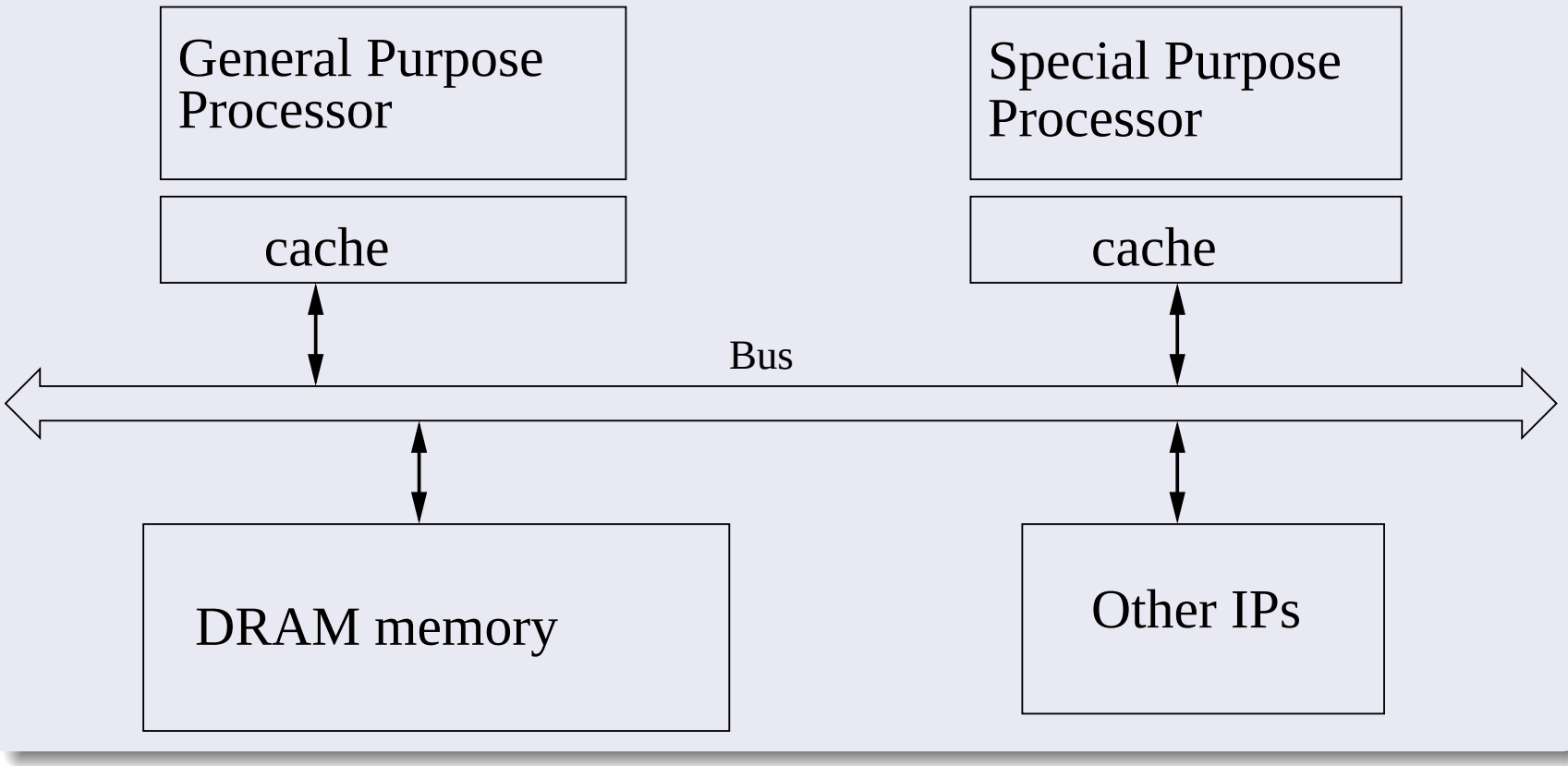
Abstraction Levels



Academic / Comercial HLS Frameworks and Compilers



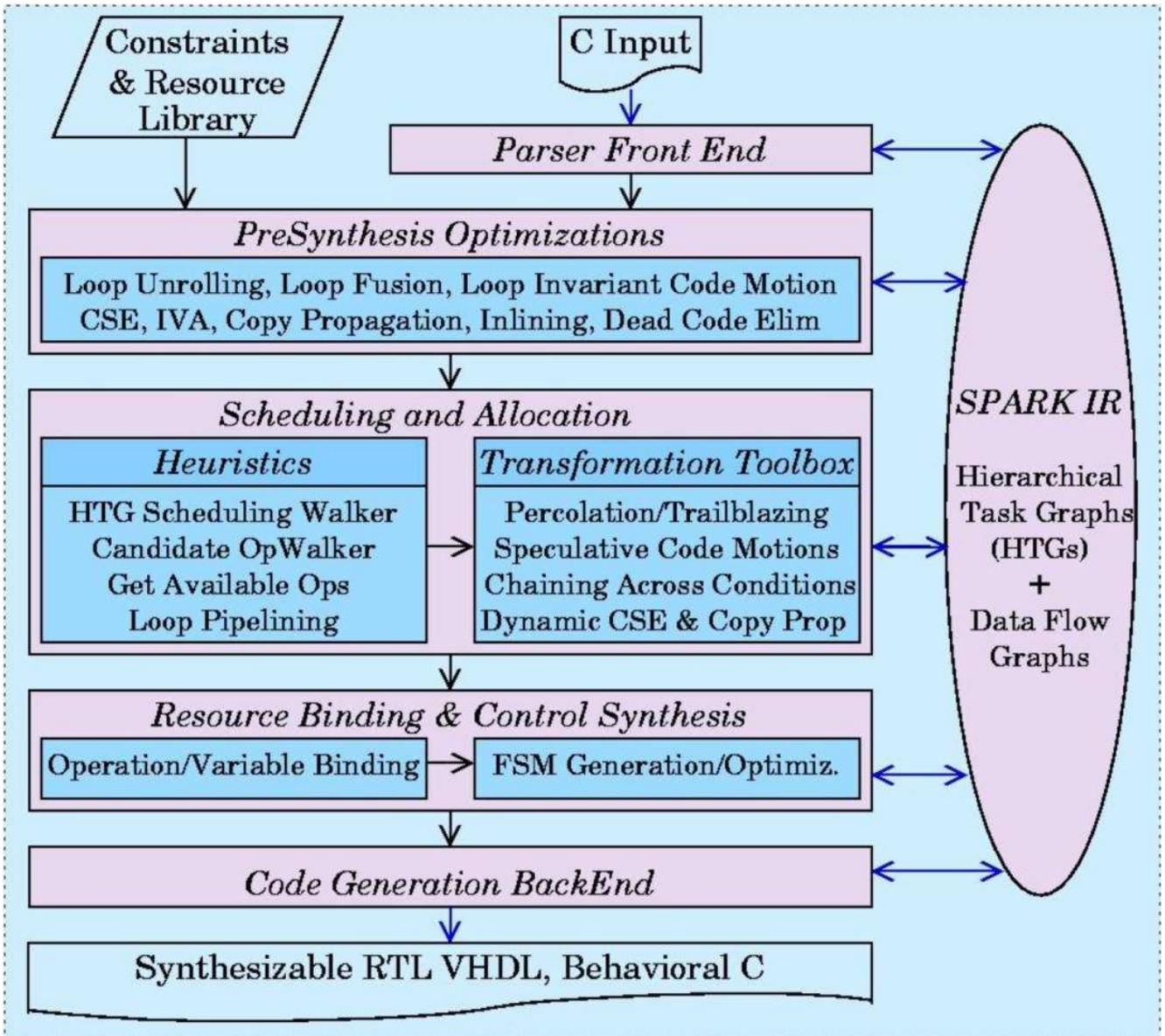
Target architecture, Shared memory model



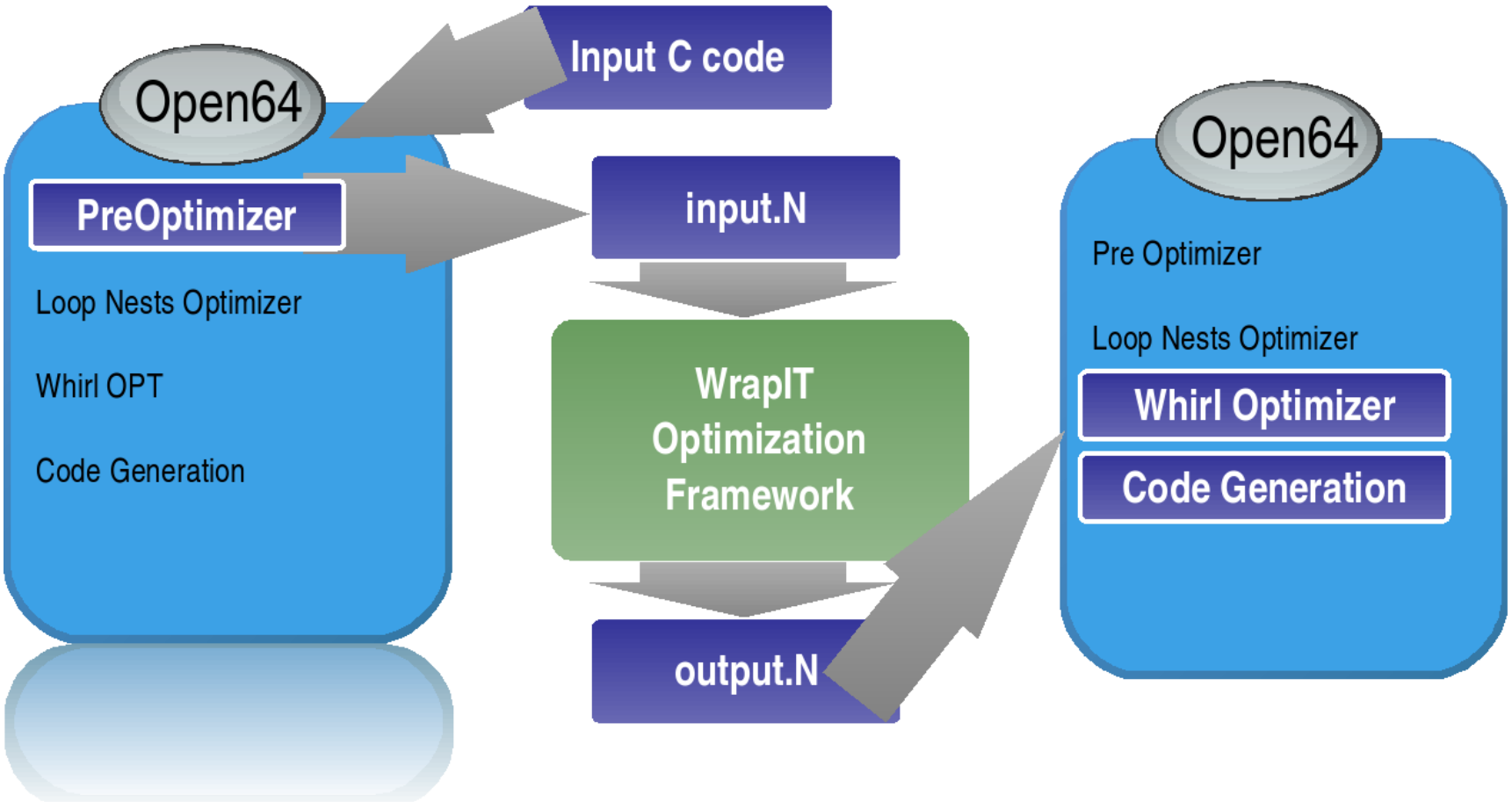
Analisys

- Dram memory is shared by the two masters (GPP & SPP) as found most of VLIW DSP processor
- The cache can be shared (HW/SW co-synthezis decision)
- SPP is synthesized using Spark HLS framework with timing and resource constraints
- Cache accesses are simulated using Dinero IV cache simulator

Spark HLS framework

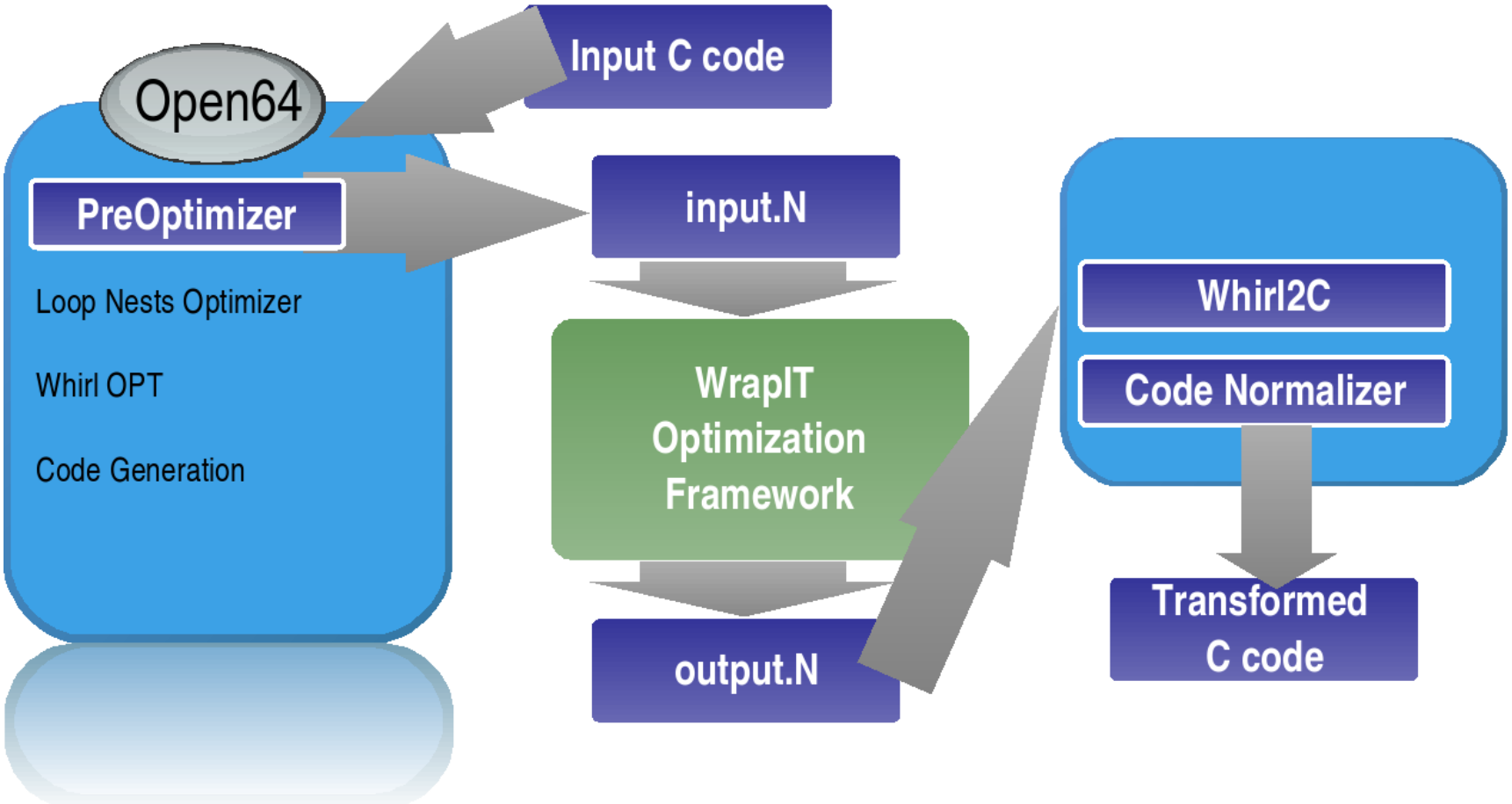


Wrapit design flow



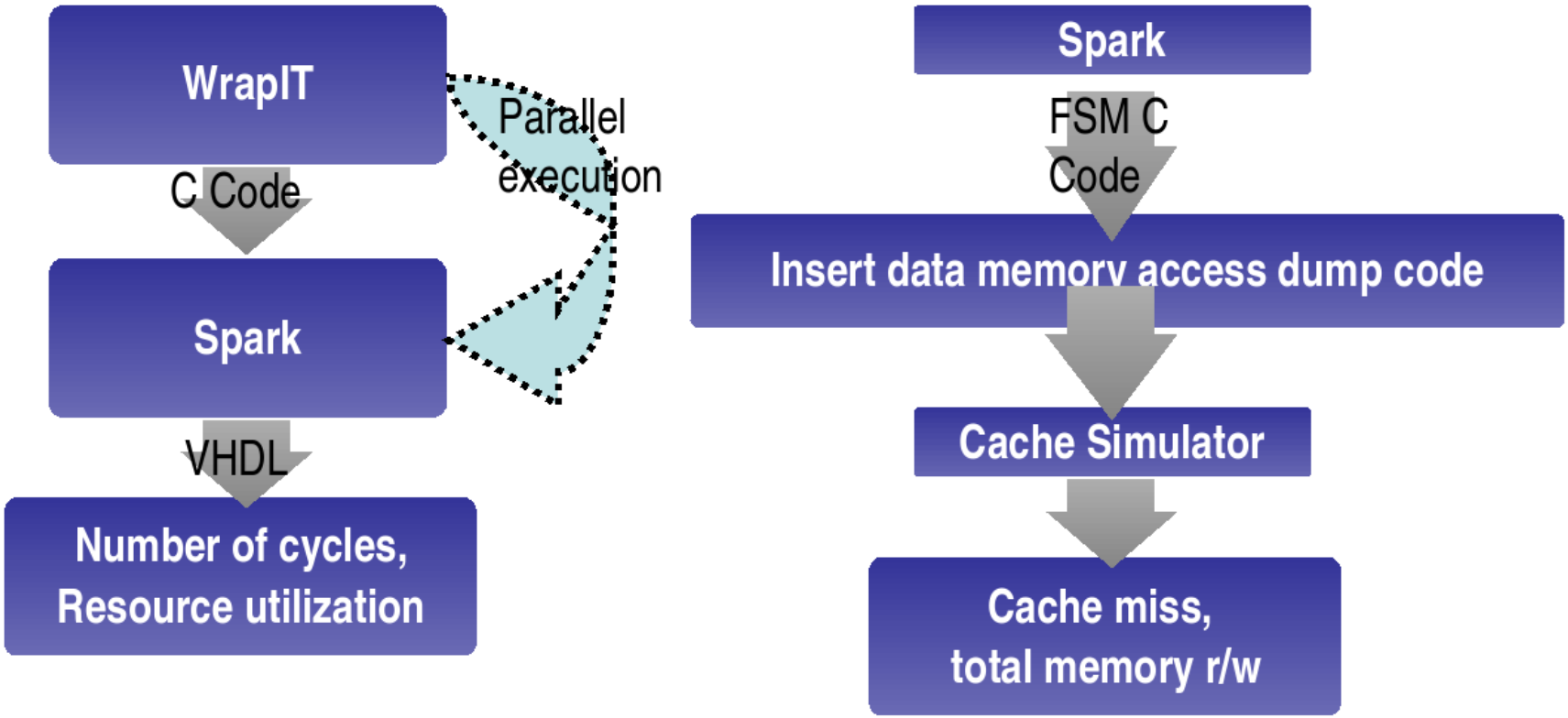
Design flow

Wrapit as preprocessing step to Spark HLS



Design flow

Wrapit + Spark simplified experimental setup



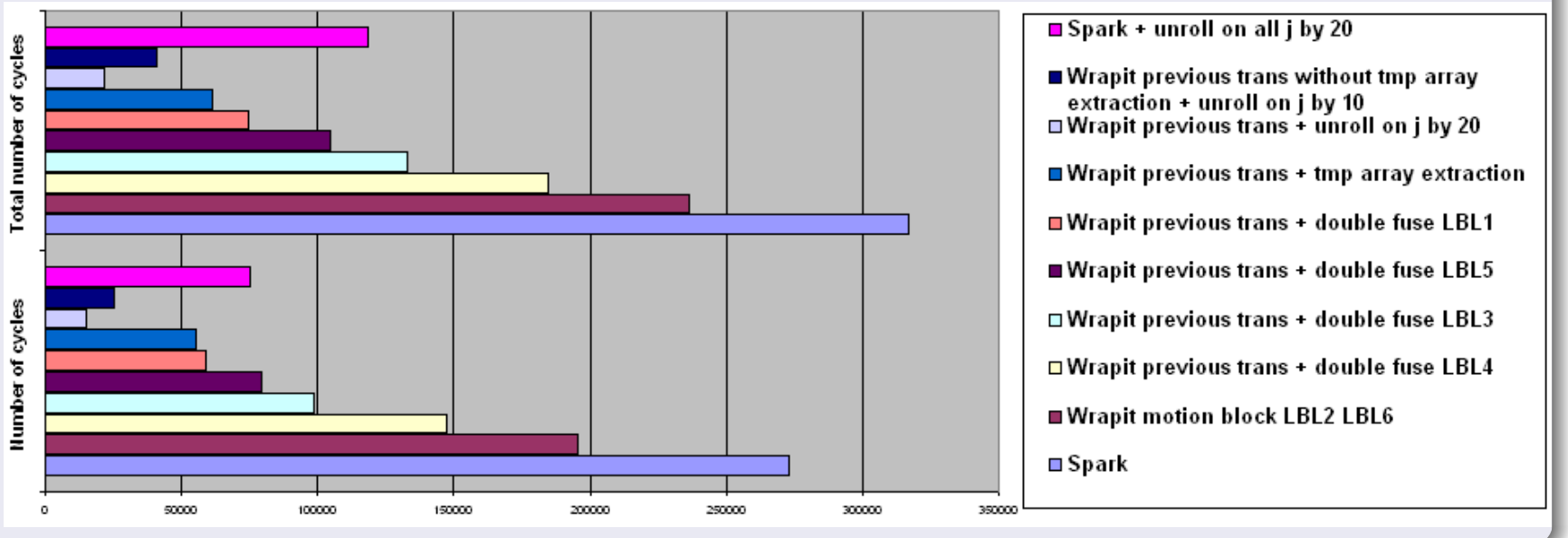
Synthesis of two examples

- Edge detection algorithm (Combined results from a Laplacian filter with horizontal and vertical Sobel filter
- YUV->RGB colorspace conversion module from the h263 / h264 video decoder from Mediabench II benchmark

Edge detection

```
#define N 100
int A[N*N], A1[N*N], B[N*N], B1[N*N], B2[N*N], C[N*N] ;
int main(void)
{
(doubly nested loops)
__URUK_LBL1 : - initialize temporar arrays with 0
__URUK_LBL2 : - initialize output array
__URUK_LBL3 : - apply Laplacian filter on A and store data on A1
__URUK_LBL4 : - apply horizontal Sobel filter on B, store on B1
__URUK_LBL5 : - apply vertical Sobel filter on B1, store on B2
__URUK_LBL6 : - merge the two results A1 and B2 and store data in C
}
```

Result chart for cache miss, memory R/W and number of execution cycles



Profiling

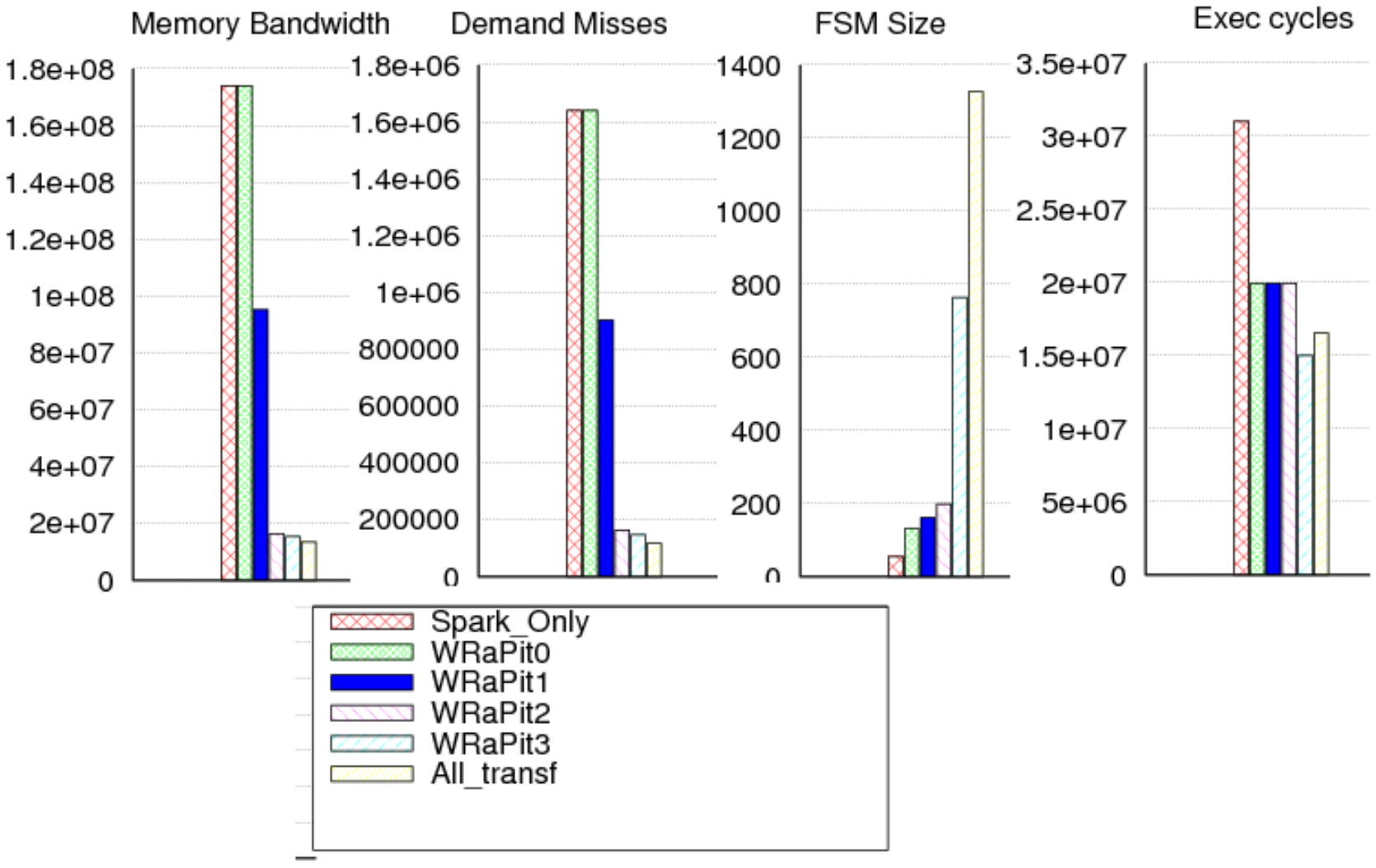
- Profiling was performed using OProfile (fixed image size 1000x1000)
- 62.33 % of execution time on YUV->RGB conversion
  - function call for vertical interpolations on U (conv420to422)
  - function call for vertical interpolations on V (conv420to422)
  - function call for horizontal interpolations on U (conv422to444)
  - function call for horizontal interpolations on V (conv422to444)
  - some code ... without any dependency to U or V arrays
  - transformation from yuv444 to RGB

Transformations

- **Spark_only** : Synthesis with spark without passing through WRaPIT
- **WRaPIT0** : Passing through WRaPIT without any loop transformations
- **WRaPIT1** : Loop reversal on vertical interpolation of U
- **WRaPIT2** : Added loop reversal on vertical interpolation of V
- **WRaPIT3** : Added fusion, strip-mining and shifts on horizontal and and vertical interpolations of U and V
- **All_transf** : Added code motion, strip-mining and loop interchange on U, V and RGB loops

h263 / h264 from Mediabench II Benchmark

Results



Conclusions

- Demonstrate potential gains of the approach using Open64 Wrapit compiler framework
- Performances can be increased dramaticaly
- The approach can also be used to transform the C code to be syntacticaly by the HLS tool

Next step in the research

Build an automatic toolbox to include all the useful transformations for a HLS tool