RWTH AACHEN UNIVERSITY

# CoEx: Novel Profiling-Based Algorithm/Architecture Co-Exploration for ASIP Design

**Juan Eusse, Christopher Williams, Rainer Leupers**

Chair for Software for Systems on Silicon (SSS)

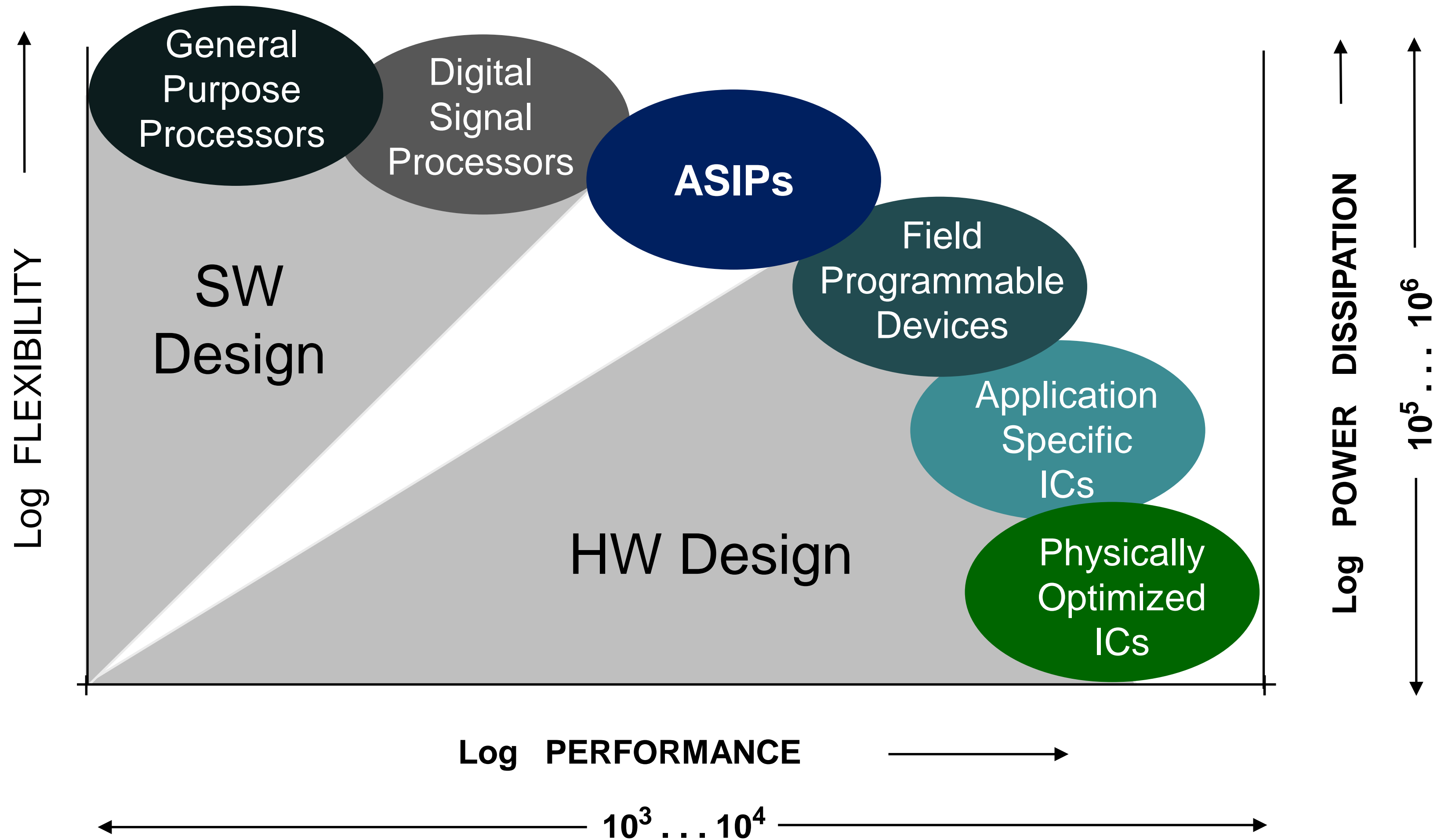
ReCoSoC 2013, Darmstadt July 10th, 2013

# OUTLINE

1 Why do we need ASIP oriented profiling?
2 Multi-Grained application profiling
3 CoEx implementation
4 Evaluation: Execution Overhead
5 Case Study: Planar-Marker detection for AR
6 Conclusions and future work

# 1. Why do we need ASIP oriented profiling? (I)

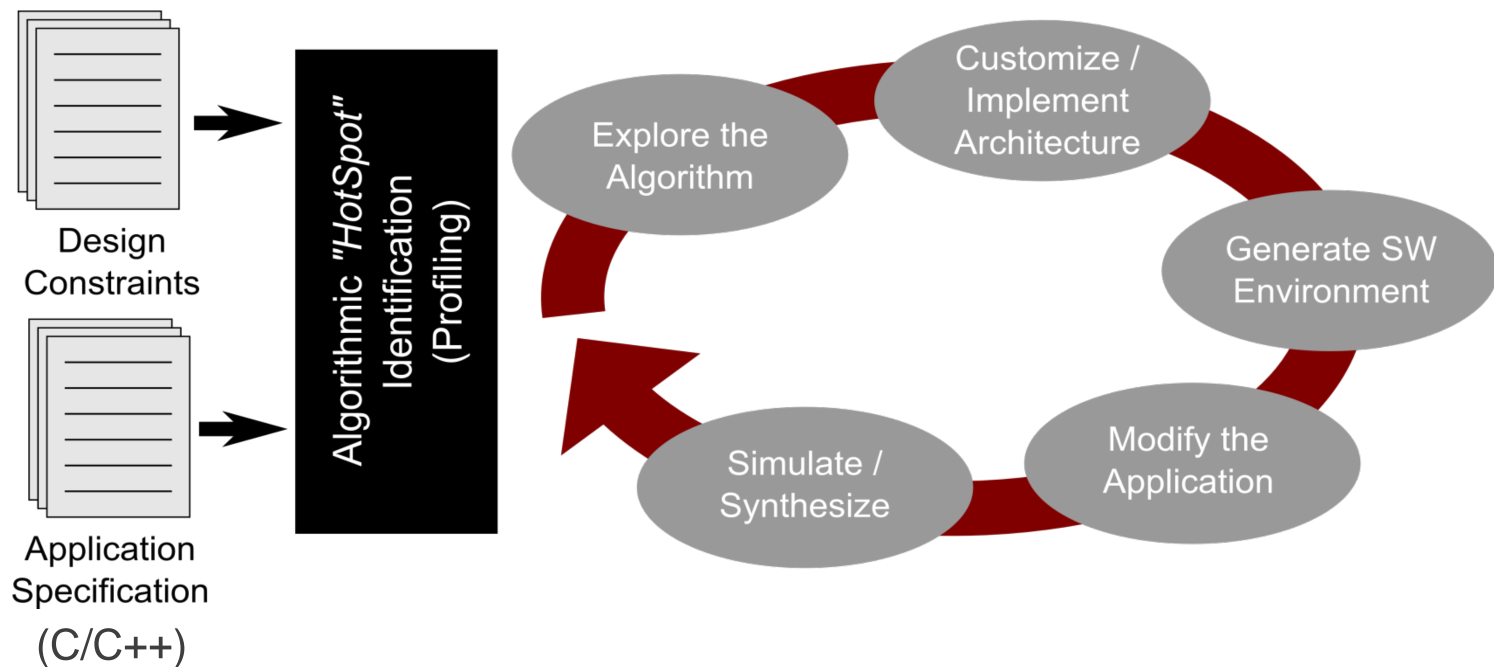
- **In a world of changing standards, how to keep the right amount of flexibility while being efficient?**




Source: T.Noll, RWTH Aachen

# 1. Why do we need ASIP oriented profiling? (II)

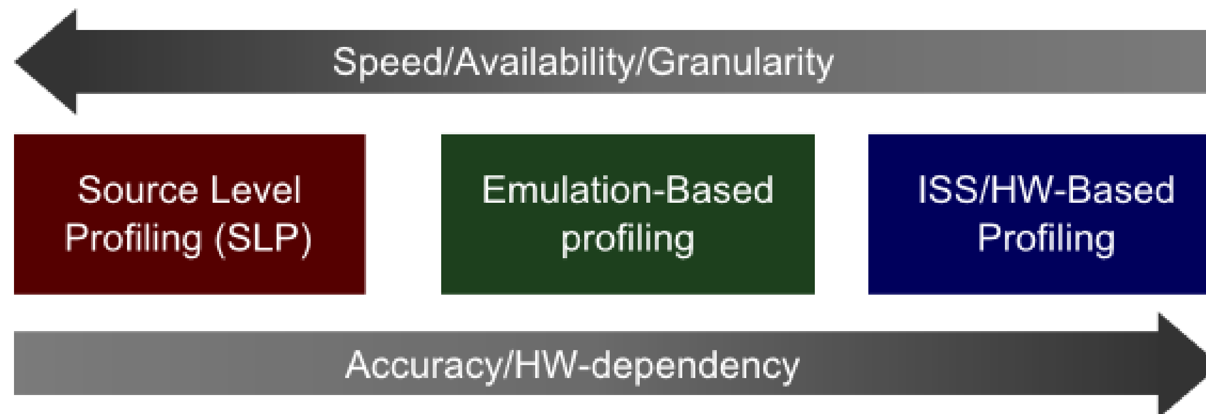
- **Architecture Description Languages (e.g. LISA) -based tools can:**
  - Generate the SW environment (assembler, linker, simulator, compiler)
  - Generate HDL descriptions
- **Profiling has remained the entry point to all ADL-based methodologies**



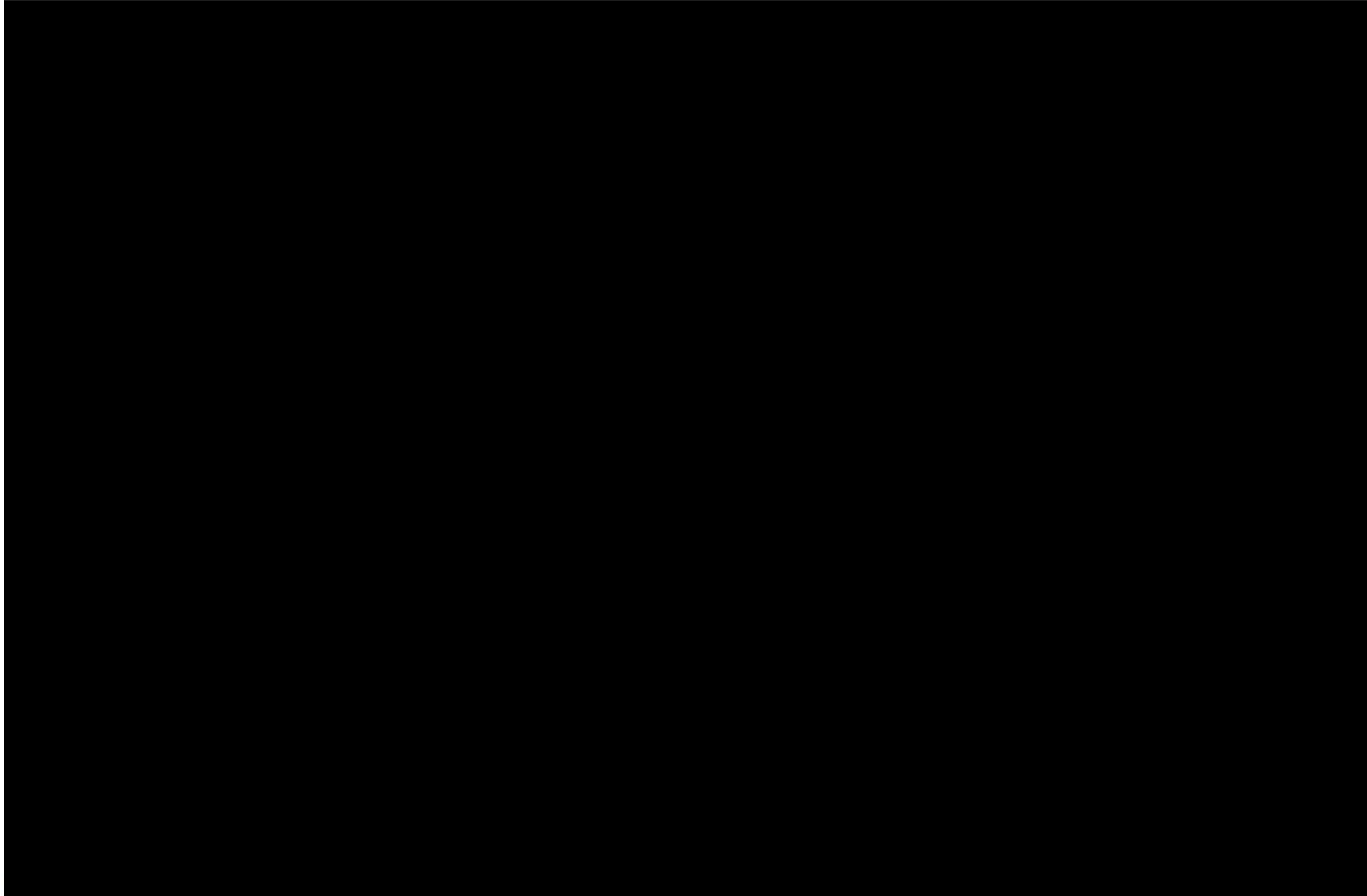
# 1. Why do we need ASIP oriented profiling? (III)

- **Input specification comes as “high-level” C/C++ code**
  - Usually directly from the algorithm designer

- **Profiling used only to detect application *“hotspots”***
  - SLP tools are intended for GP program analysis
  - Emulation-Based is more accurate but cannot be reused
  - ISS/HW based requires the existence of a target processor architecture

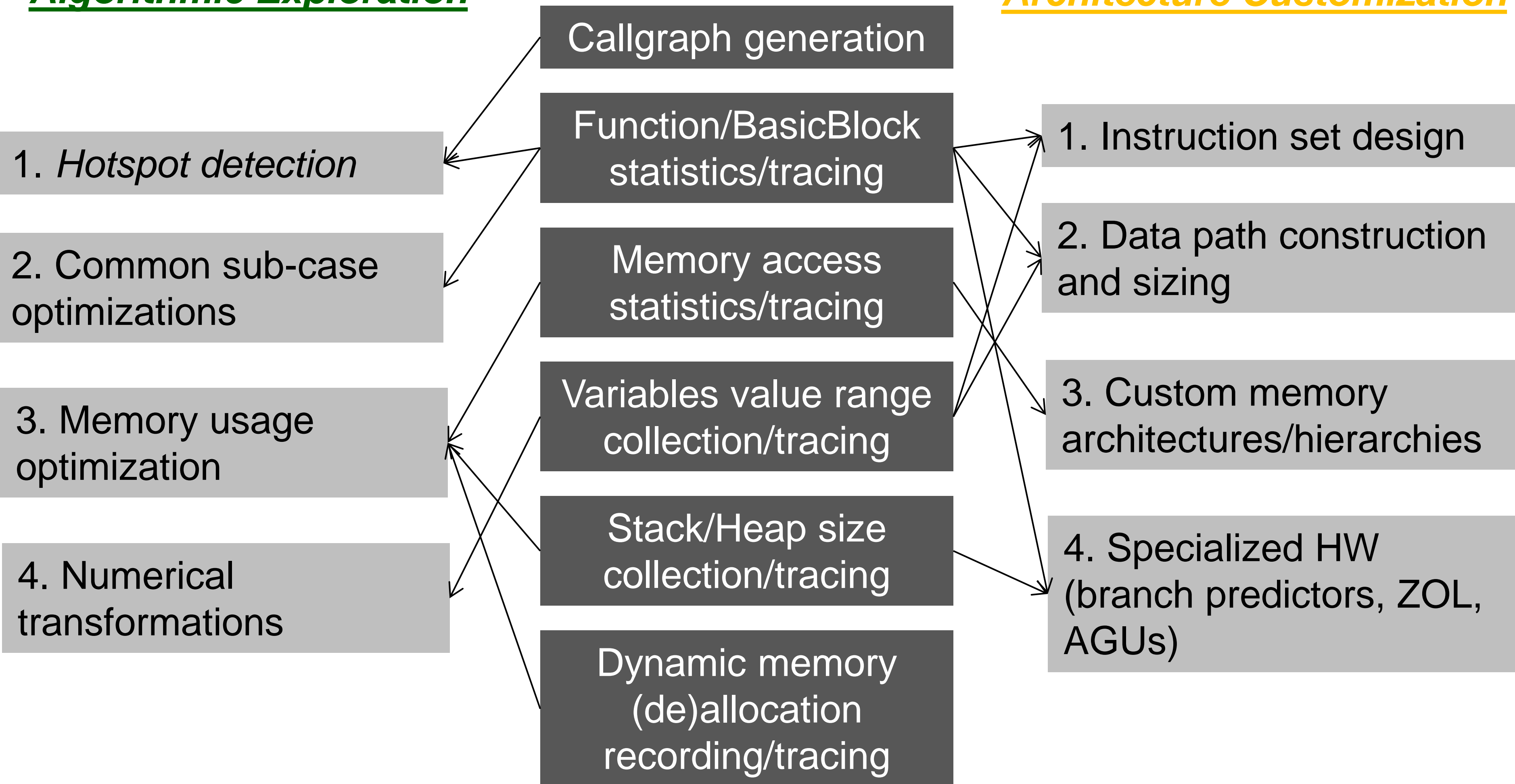


# 3. Multi-Grained application profiling (I)

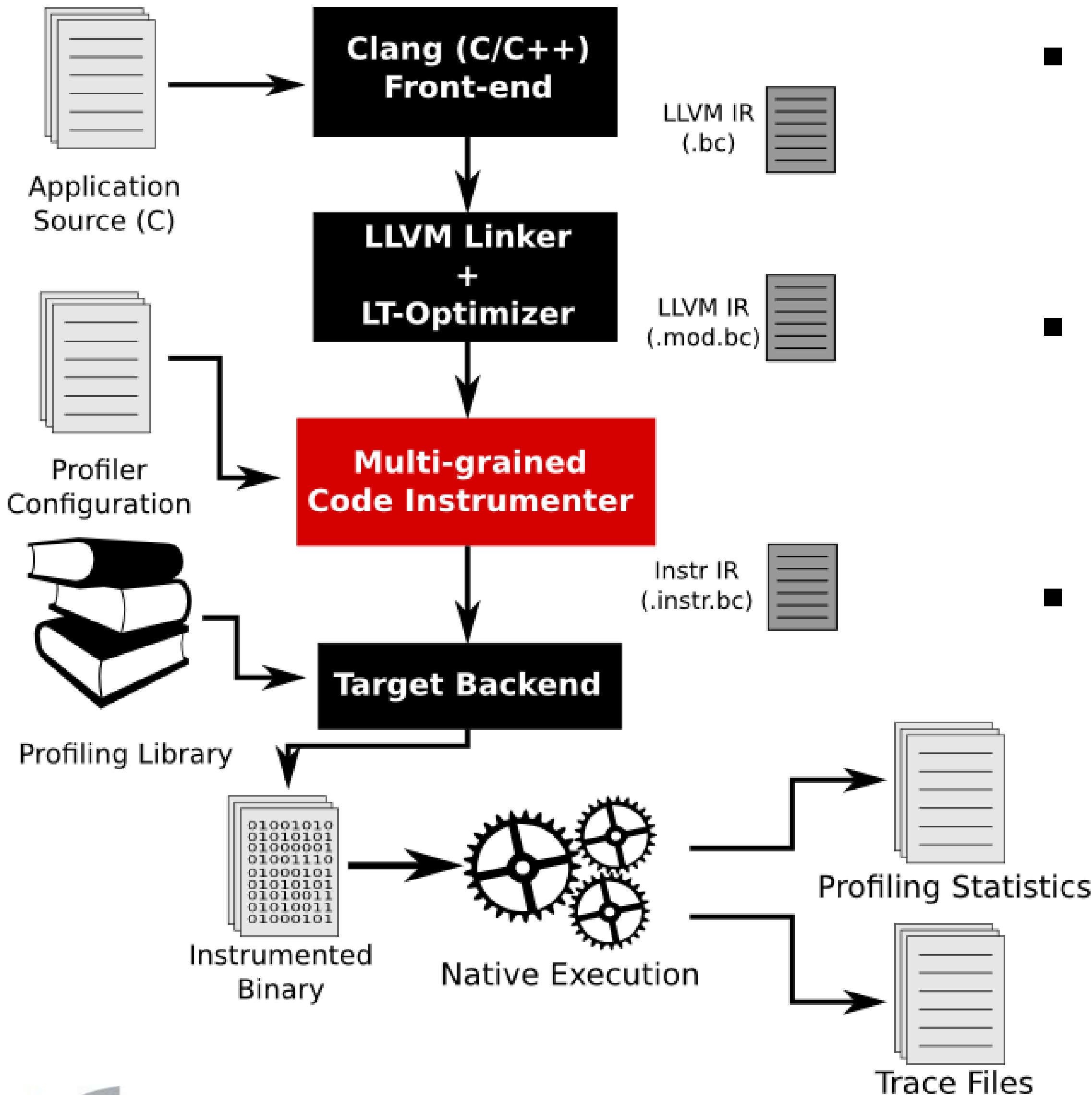


# 3. Multi-Grained application profiling (II)

- **Available profiling configurations related to the ASIP design stage**



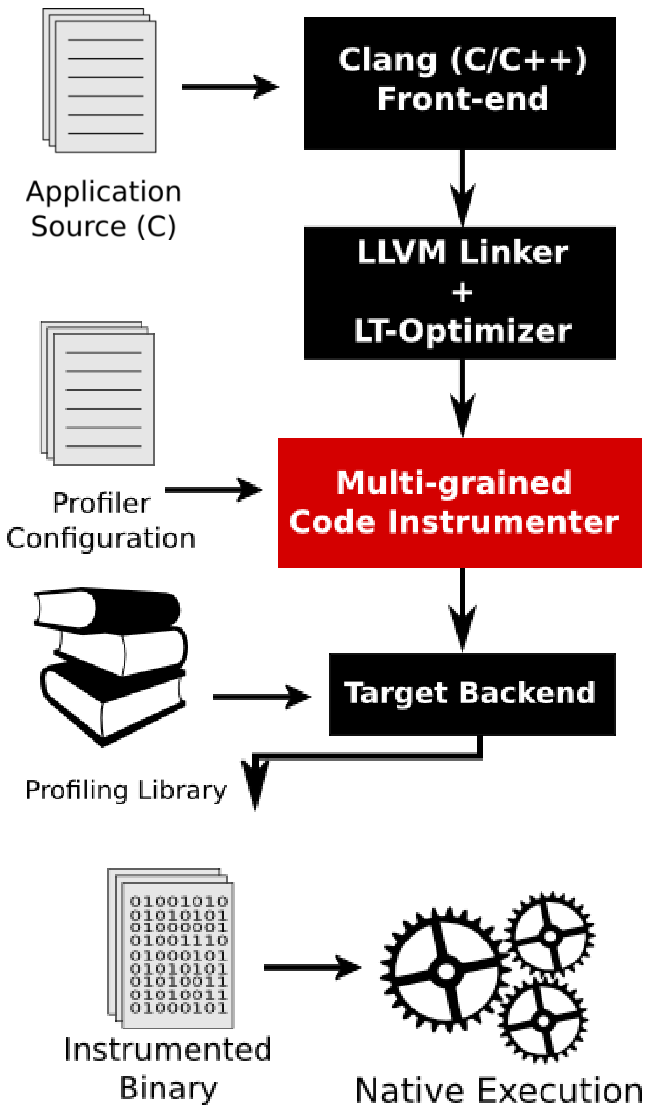
# 4. CoEx implementation (I)




- **Standalone Multi-Grained SLP based on LLVM code instrumentation**
- **Granularity of the *profiling scenario* is configured by the designer**
- **Generated profiling information is independent of the target architecture**



RWTH AACHEN UNIVERSITY




```c
void merge_lines(char cond) {
  float *sPtr = (float*)malloc(2*sizeof(float));
  if(cond) sPtr[index] = val; else sPtr[index+1] = val;
  free(sPtr);
}
```

+

```xml
1  <?xml ve                                      '?>
2  <profCon
3    <globa
4      <trac                                     oc="true"
5                                                ="true" />
6      <fnBbSt enabled="true"/>
7      <traceOutput filename="trace.txt"/>
                                                 alse"/>
14       ...
15     </function>
16   </functionConfig>
17 </profConfig>
```

C

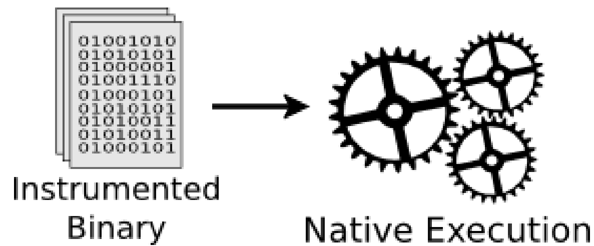
```
if.end:
 call void @free(i8* %8)
 ret %12
```

Pure LLVM IR

```
entrv:
entry:
 call void (i8*,...)* @__PROF_LSInitLocalInfo(...)
 %call = call i8* @__PROF_Malloc(i64 8)
 %tobool = icmp ne i8 %1, 0
 br i1 %tobool, label %if.then, label %if.else

if.then:
 call void @__PROF_FNProcessFunctionEntry(...)
 %call1 = call float @fn1()
 call void @__PROF_FNProcessFunctionExit(...)
 store float %call1, float* %arrayidx
 br label %if.end

if.else:
 call void @__PROF_Mem(...)
 store float %5, float* %arrayidx2
 br label %if.end

if.end:
 call void @__PROF_Free(i8* %9)
 ret %12
```

Instrumented LLVM IR

# 4. CoEx implementation (III)




- **Static File:**
  - Language dependent information
- **Dynamic File:**
  - Application execution extracted information
- **General Trace**
  - Functions, basic blocks, memory
- **Value Trace:**
  - Individual value traces

```
1  <profilerResults>
2  ...
3  <fu
4    <
5    <                                                     nt>
6                                                          >
7
8  ...
9  1:s
10 2:s                                                     :4
11 3:0
12 4:s
13 5:h
14 6:h
15 7:e
16 8:s
17   <
18   .
19   <
20 </D
23  <MaxStackSize> 26 </MaxStackSize>
24 </profilerResults>
```

```
|merge_lines|6|cond|8
|merge_lines|11|cond|8
|merge_lines|16|cond|9
|merge_lines|21|cond|2
|merge_lines|26|cond|7
|merge_lines|31|cond|5
|merge_lines|36|cond|7
|merge_lines|41|cond|2
|merge_lines|46|cond|8
|merge_lines|51|cond|6
```

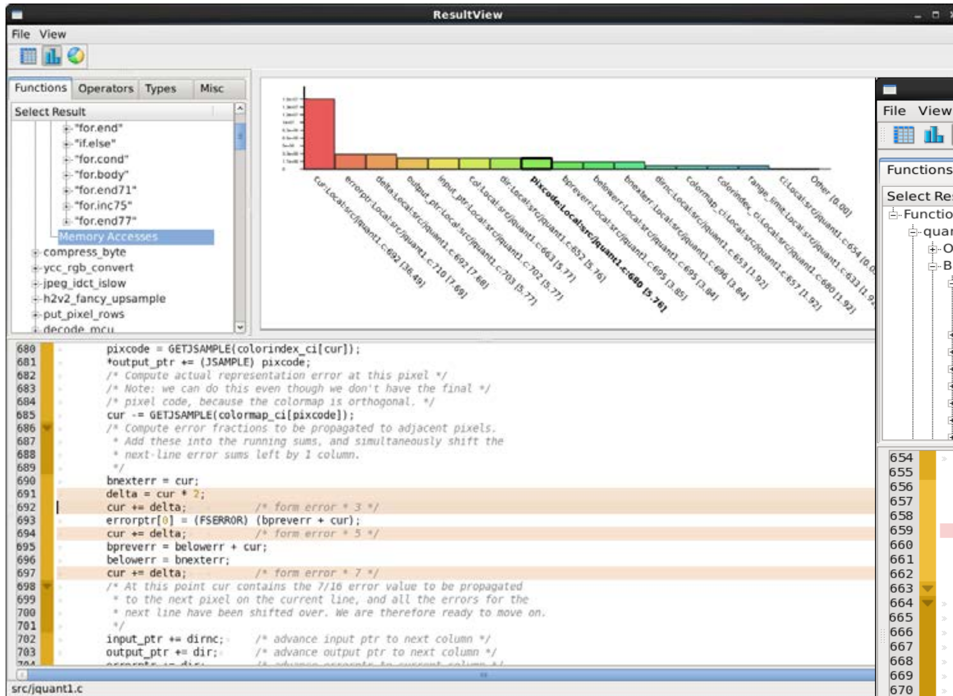
Value trace file

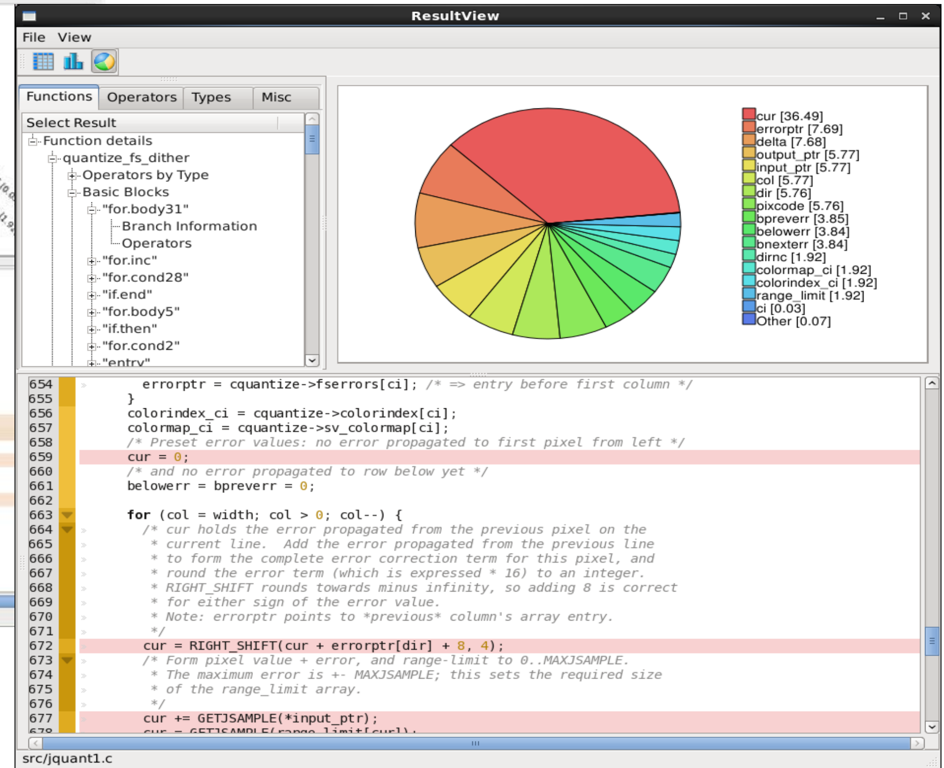
Dynamic output file

Size and type of output depends on the ***profiling scenario*** configuration

# 4. CoEx implementation (IV)

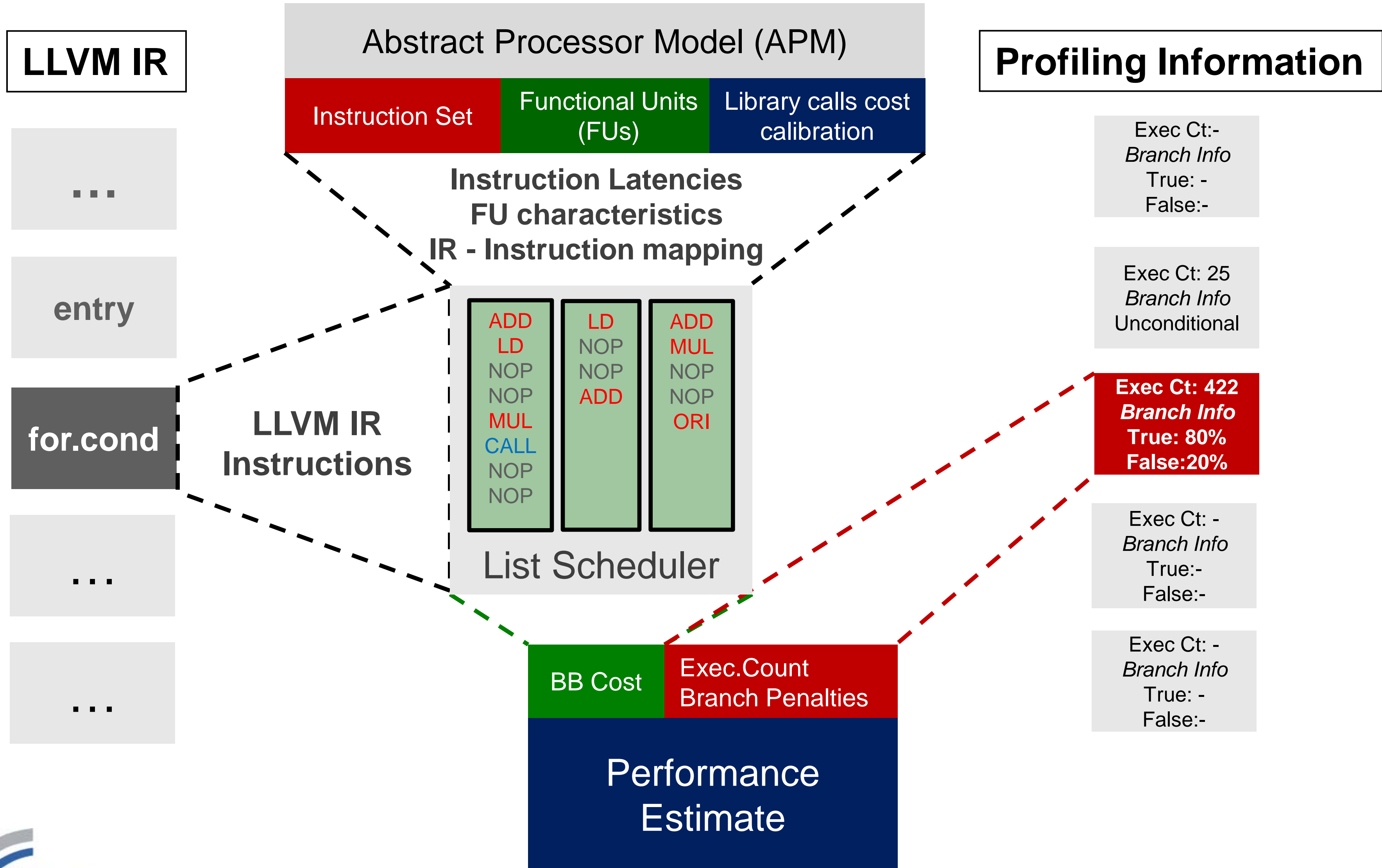
- **Profiling results visualization:**
  - Intuitive navigation through the profiling results
  - Linking/highlighting of the application source code



# 4. CoEx implementation (IV)

- **Pre-architectural performance estimation**

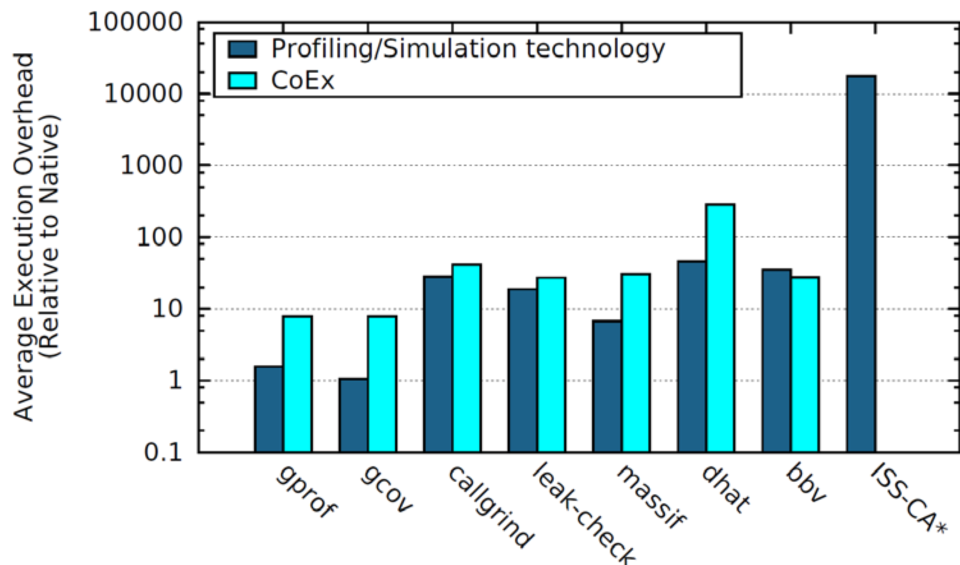





RWTH AACHEN UNIVERSITY

# 5. Evaluation: Execution Overhead (I)

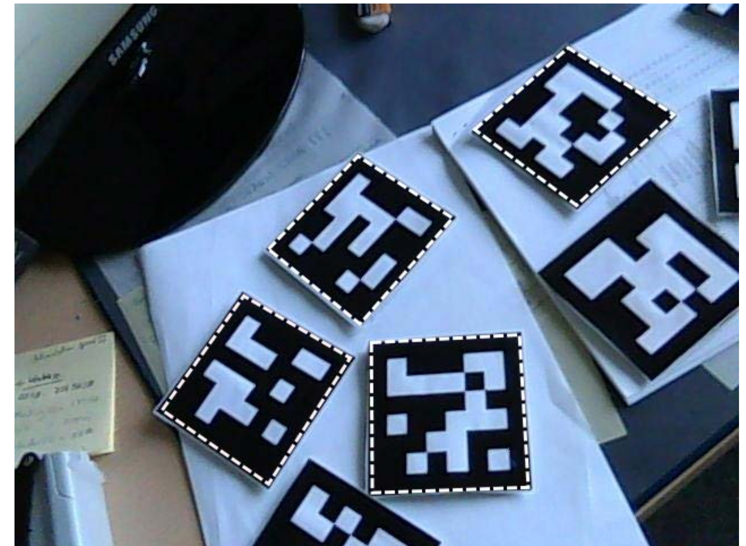
- **Instrumentation Overhead:**
  - Generated ***profiling scenarios*** for AES, JPEG, ADPCM, FFT(iFFT), Blowfish, Susan from DSPStone and EEMBC.
  - Two non-optimized applications considered:
    - Audio filter application
    - Planar marker detection for augmented reality – case study
  - ***Profiling scenarios*** tuned to match existing SLP analyses
  - Native execution time is the baseline for overhead calculations




Overhead compared with gprof, gcov, callgrind (function call), leak-check (dynamic memory), massif (stack/heap), dhat (memory accesses), bbv (basic block tracing)

# 6. Case Study: Planar-Marker detection for AR (I)

- **Customization of the PD_RISC processor for an AR application**
- **Detect black-and-white 2-Dimensional markers in an image**
  - Input specification consists on ~2900 lines of C code
  - Function pointers, recursion, SP floating-point, dynamic memory management heavily used
- **Algorithm steps:**
  1. Divide the image into 40x40px regions
  2. Detect pixels with strong magnitude changes
  3. Detect which belong to straight lines
  4. Merge compatible lines (super-lines)
  5. Extend super lines until corners
  6. Keep lines that have corners
  7. Build line chains
  8. Detect markers



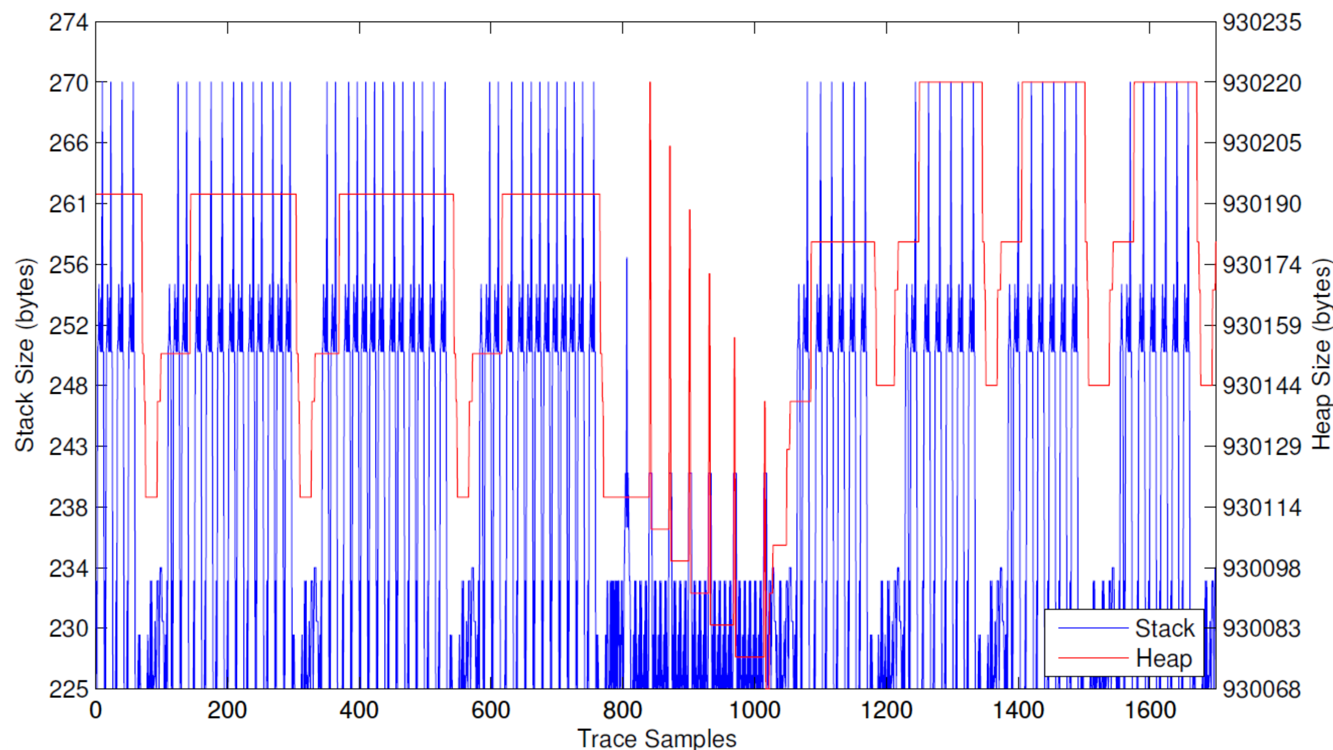
- ***Profiling Scenario 1:*** *Function/Basic Block/Timing analysis (no trace)*
  - Light-weight profiling (low execution overhead)
  - Steps (3) and (4) of the algorithm consume 29% and 40% total execution time

| | Memory Address | Load-Store | Integer Ops. | Floating Point Ops. | Function Execution Count |
|---|---|---|---|---|---|
| Line check | 11 | 70 | 3 | 25 | 693600 |
| 2x1 Vector Normalization | 12 | 50 | 2 | 10 | 734044 |
| 2x1 Dot Product | 4 | 10 | 0 | 3 | 2264043 |
| Square Root | 0 | 17 | 2 | 5 | 1073440 |
| 2x1 Vector Length | 4 | 9 | 0 | 3 | 1099245 |

  - 10% in calls to malloc/free

# 6. Case Study: Planar-Marker detection for AR (III)

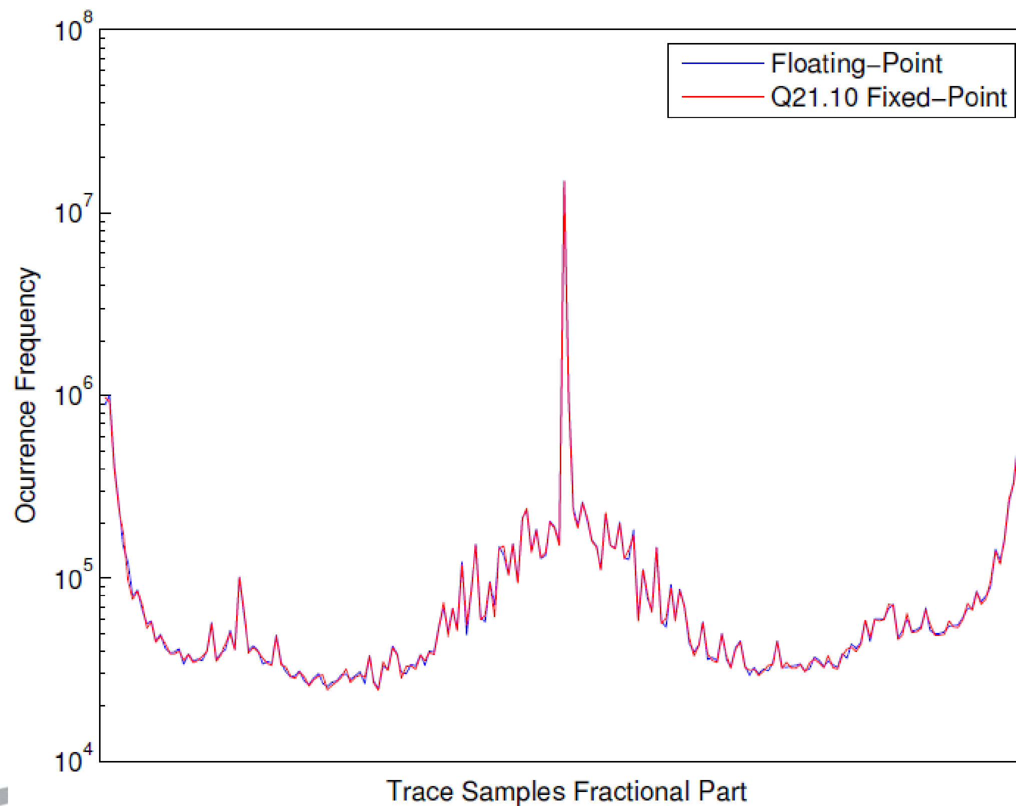
- ***Profiling Scenario 2:*** *Function/Basic Block profiling (stack/heap trace)*
  - Observed initial/final frame memory (de)allocation
  - Closer look revealed repetitive (de)allocation
  - Trace examination enabled:
    - Static memory and memory pool sizing




- **13% of overall execution speedup**
- **No architectural customization**

# 6. Case Study: Planar-Marker detection for AR (IV)

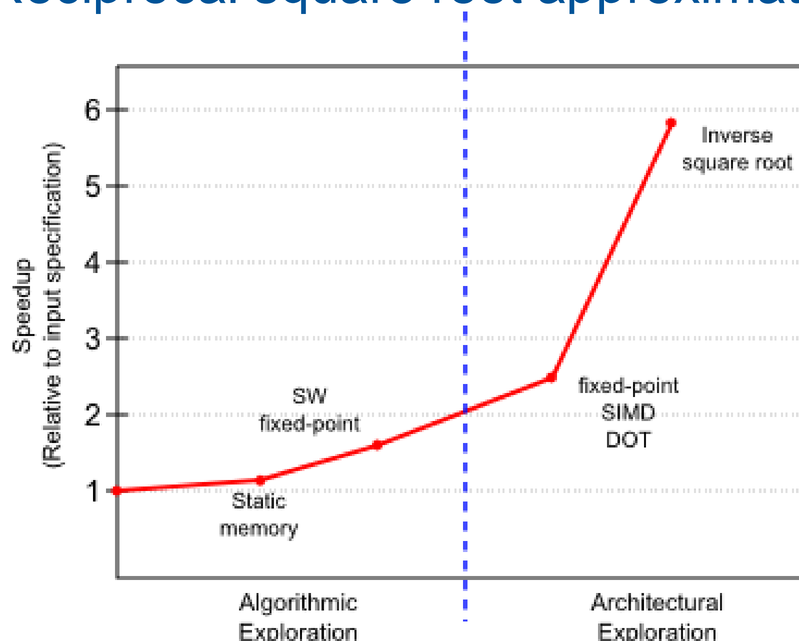
- ***Profiling Scenario 3:*** *Hotspot input/output value trace*
  - Traced hotspots from *profiling scenario 1*
  - Assumed a 32bit fixed point word
  - Explored MSE for different quantization schemes (using Matlab)




- ***Replaced floats by Q21.10 fixed point***
- ***27% further speedup***
- ***Still no architectural customization***

# 6. Case Study: Planar-Marker detection for AR (IV)

- ***Profiling Scenario 4:*** *Function/Basic Block/Memory Access profiling (Fn/BB traces enabled)*
  - *Exploration of the generated information through the GUI*
  - *Architecture customization only done using fusion-type instructions:*
    - Fixed point support for the ALU
    - SIMD addition, substraction and multiplication
    - Dot product for 2x1 vectors
    - Reciprocal square root approximation




**6x combined speedup achieved in only two days of design time**

- **Pre-architectural performance estimation of case study results**
  - Estimation performed after each successive algorithm/architecture iteration
  - Accuracy metric based on CA simulation results from ISS

| Application/Architecture Revision | ISS-CA Cycles | Estimated Cycles | Error (%) | ISS Time (sec) | Estimation Time (sec) | Estimation/ Simulation Ratio |
|---|---|---|---|---|---|---|
| Input specification + PD RISC (Base) | 3705186373 | 2970991784 | -19.82 | 4147 | 1.23 | 3371 |
| Static Memory + PD RISC (Base) | 3403357531 | 2688236170 | -21.01 | 3762 | 1.21 | 3109 |
| Fixed Point + PD RISC (Base) | 2658942738 | 2238013034 | -15.83 | 2991 | 1.22 | 2471 |
| Fixed Point + PD RISC (Fixed +Vector) | 1670310514 | 1365812907 | -18.23 | 2948 | 1.25 | 2358 |
| Fixed Point + PD RISC (Square Reciprocal approx.) | 622717072 | 514052942 | -17.45 | 2991 | 1.24 | 2412 |

# 7. Conclusions and future work (I)

- We propose Multi-Grained Profiling, which combines granularity levels according to the ASIP design stage to ease *algorithmic exploration, application optimization* and *architecture exploration.*

- We have implemented an MGP-enabled profiling tool (CoEx) to test the validity of the approach.

- Although the execution overhead regarding native execution is considerable, the amount of generated information and the possibility of re-using it for other analyses (i.e. performance estimation) compensates such overhead.

- A GUI has been developed to help the designer in the analysis of the generated profiling information.

# 7. Conclusions and future work (II)

- Pre-architectural performance estimation of early architectural decisions has been also explored, obtaining fairly accurate results without the need for application simulation on an ISS.
- In the case study we have shown that by using CoEx, a designer can grasp the inner workings of an application specification ina time efficient manner.
- Furthermore, we were able to customize the PD_RISC processor in just two days design time to detect planar markers in 2D images, obtaining 6x performance gains.
- Future work will explore more in depth performance estimation based on abstract processor models, in order to get more accurate results.

# Questions?

# Thank you!