MPSoC 2006

Profiling Based Architecture Optimization for Heterogeneous MPSoC

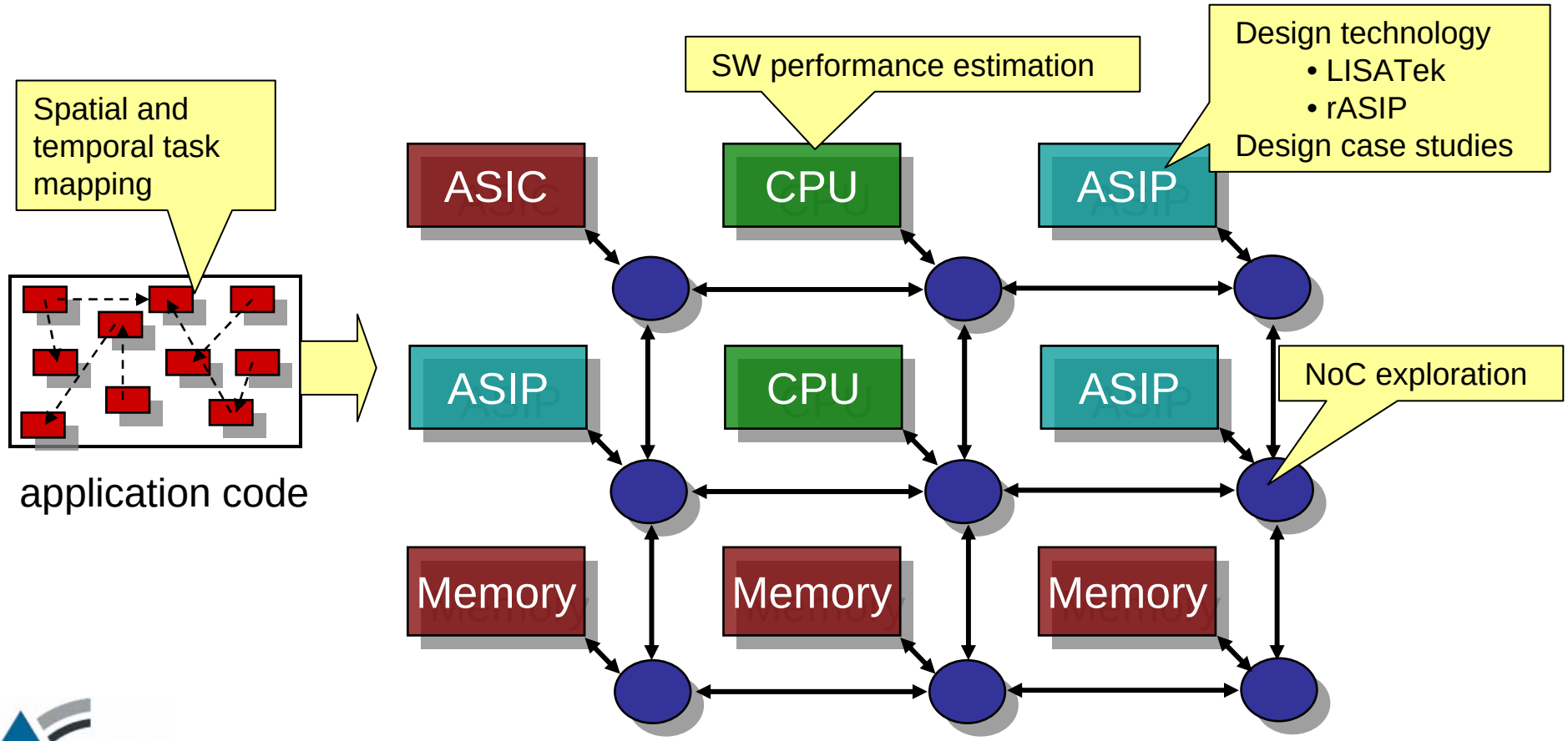
Rainer Leupers, Heinrich Meyr

RWTH Aachen University

Software for Systems on Silicon (SSS)

ISS MPSoC research focus areas

- Wireless+multimedia applications: heterogeneous MPSoC
  - Various processing elements, including ASIPs
  - (Application specific) NoC
- Architecture exploration/optimization via virtual prototypes

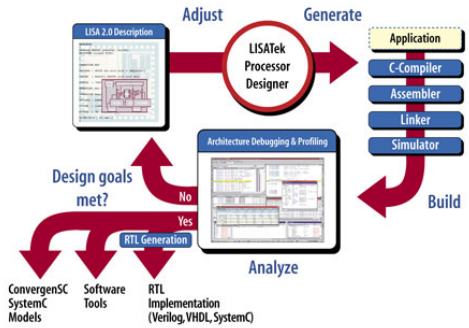


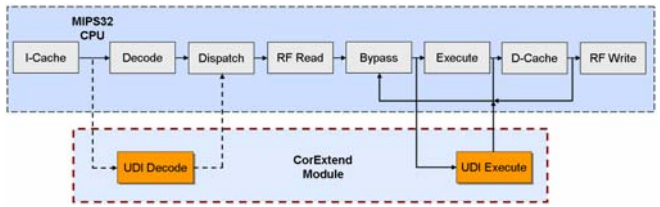
Overview

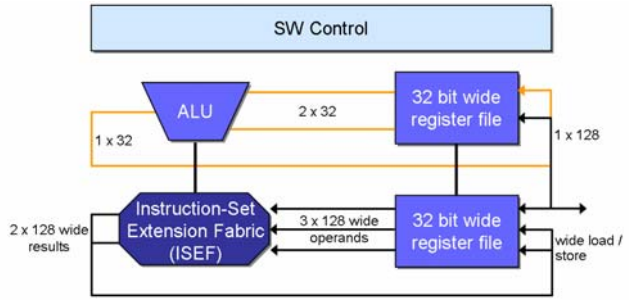
1. Customizable embedded processor design flow
2. MPSoC SW performance estimation framework
3. Future work

Today´s ASIP design technologies

- ADL based (ASIP-from-scratch)
  - E.g. LISATek, Target, Expression
  - Max. flexibility + efficiency, but significant design effort
- Configurable processor cores
  - E.g. Tensilica Xtensa, MIPS CorExtend, ARC Tangent
  - Pre-designed + pre-verified core
  - Efficiency via custom instruction set extensions (ISE)
- Special case: reconfigurable processors
  - E.g. Stretch

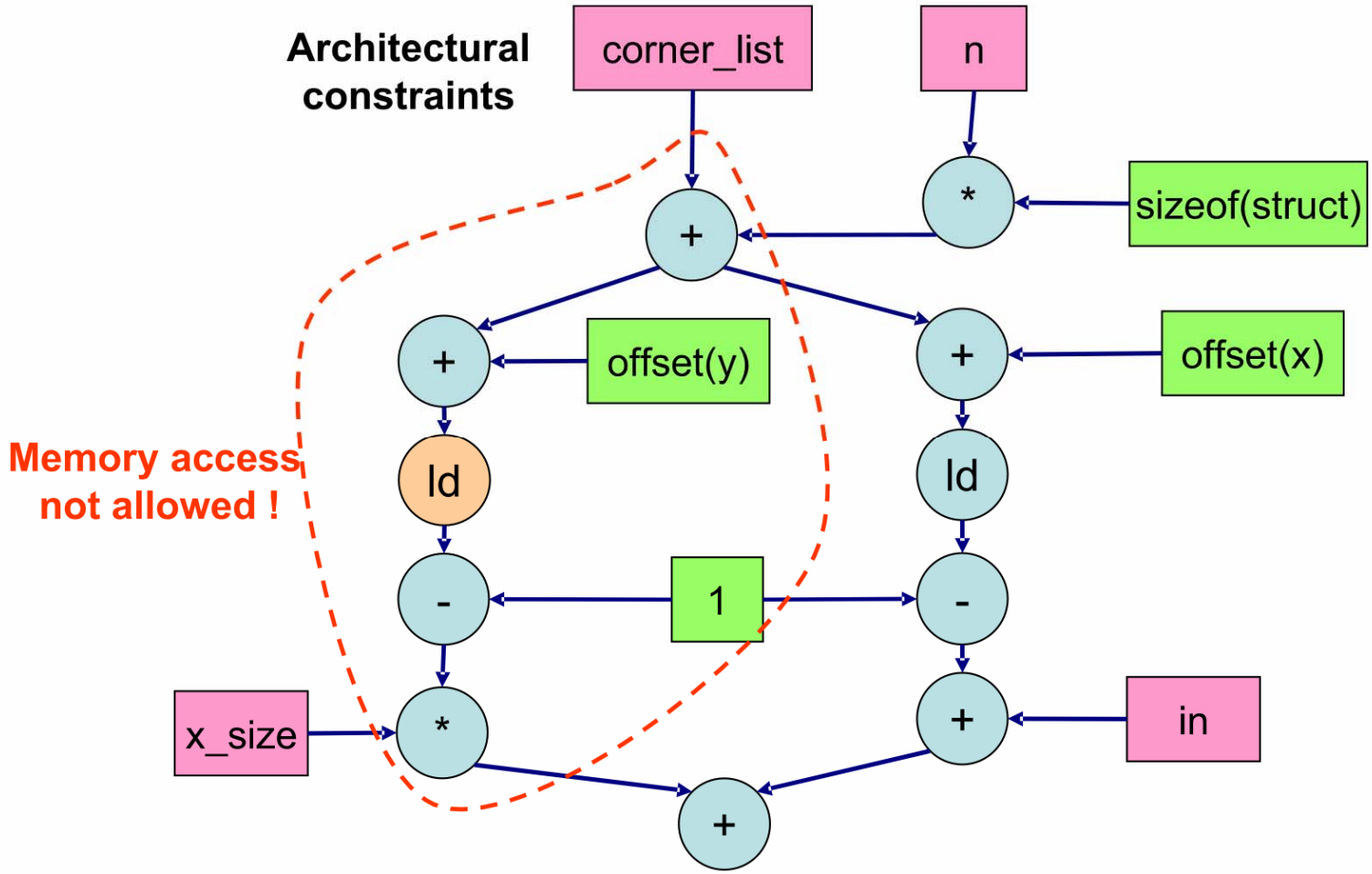






ISE: goal and constraints

- Primary goal: maximum application speedup under set of constraints



ISE: approach

- Complex optimization problem, huge design space, back annotation required
- Optimal solution cannot be found within reasonable computation times
- Our approach:
  - **Interactive ISE identification**
  - **Tools generate AT curve**
  - **For each point: use mix of ILP and heuristics to synthesize close-to-optimal ISEs**
  - **User reviews and selects design points of interest for further fine-grained exploration**
  - **RTL generation and logic synthesis**
- Short turnaround times

Processor customization flow

(1) Application code analysis

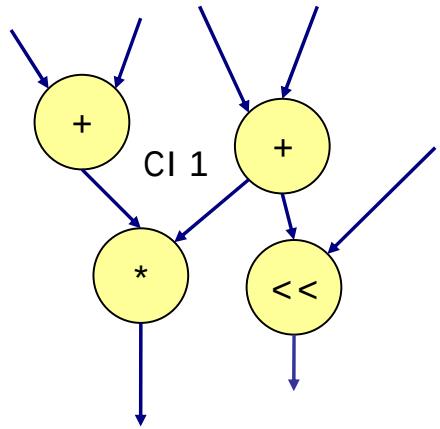
- At C source level
- Execution characteristics + hot spots

(2) Custom instruction (CI) identification

- Optimal set of CIs to speed up hot spot under area and machine constraints

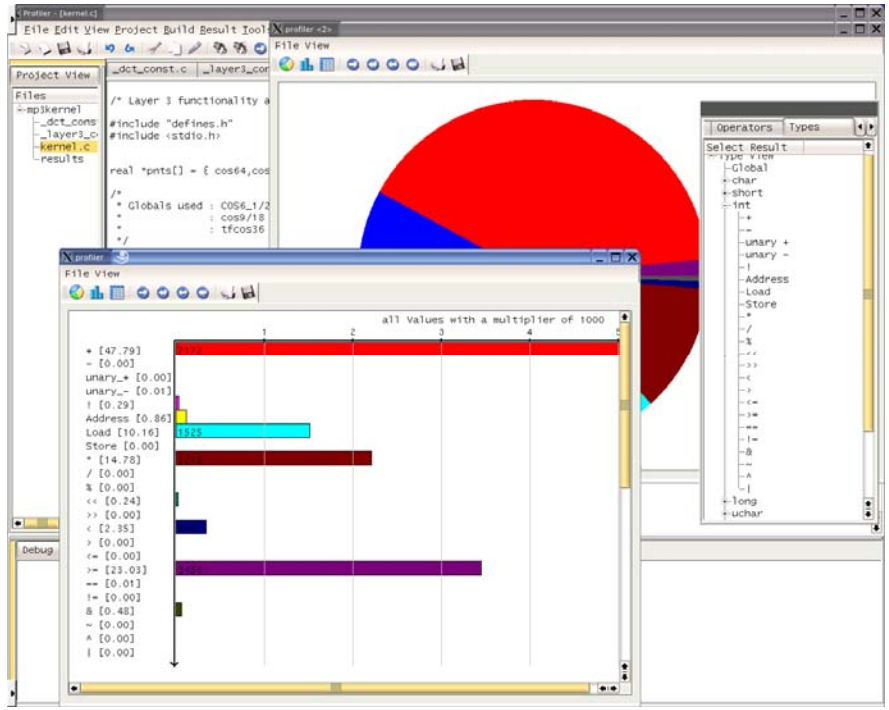
(3) CI implementation

- Interfacing with the configurable processor core
- Meet area and latency constraints



Code analysis by Micro-Profiler

- Fine grained C code profiling tool
- Instrumentation of 3-address C code intermediate format
- Makes all C operations visible
- Predicts compiler optimization effects
- Precise profiling of
  - Operator execution frequencies
  - Operation bit widths
  - Memory accesses/cache hits
  - ...
- Statistics export to ISE synthesis tool



Processor customization flow (cont.)

(4) SW adaptation and tools generation

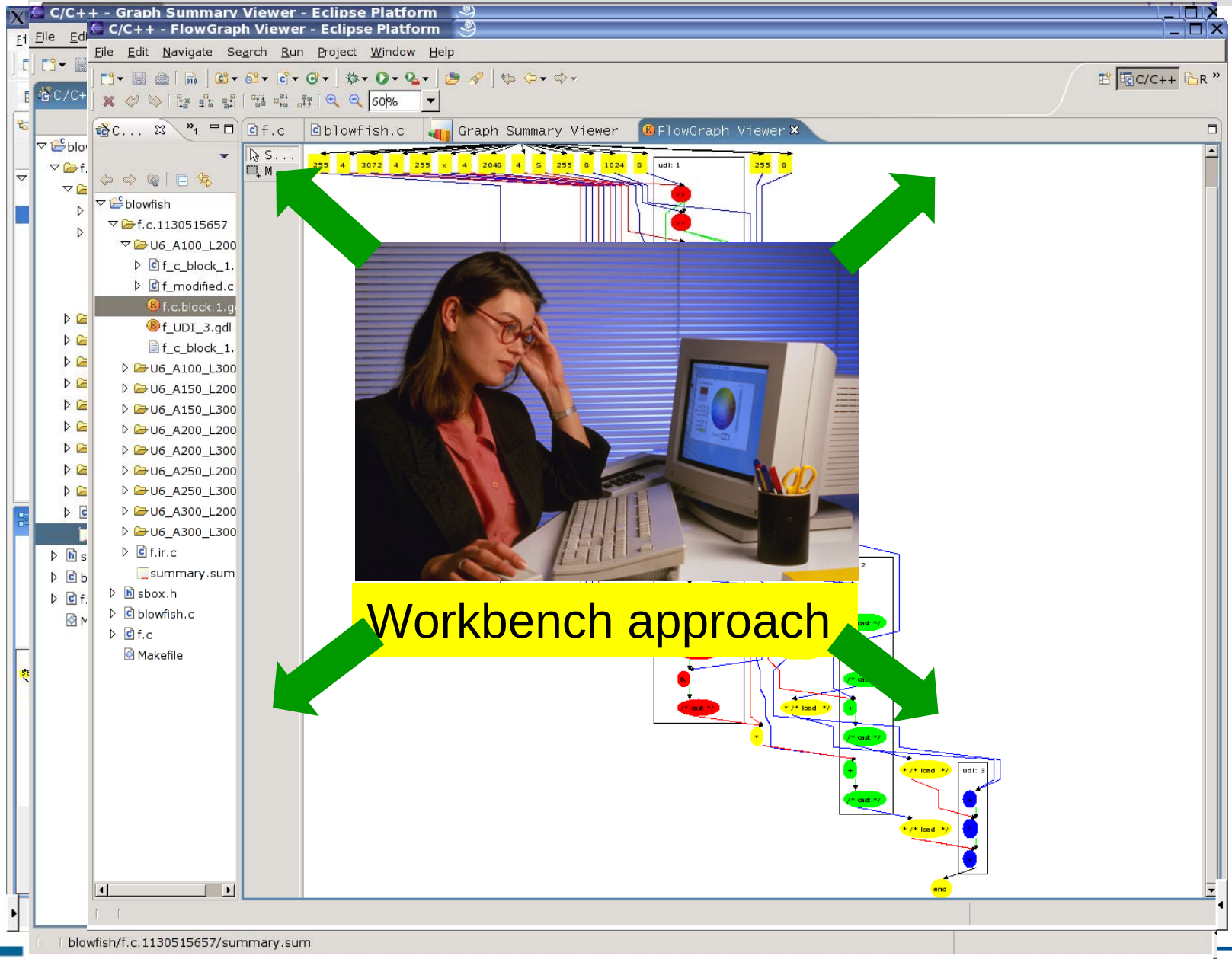
- Rewrite application C code to utilize CIs
- Adapt C compiler, ISS to support CIs

(5) HW architecture implementation

- Generate RTL HDL code for CIs
- Synthesize coprocessor for CIs

```
/*************************************
* Original Code of Function F
*************************************/
extern unsigned long S[4][256];
unsigned long F(unsigned long x)
{
        unsigned short a;
        unsigned short b;
        unsigned short c;
        unsigned short d;
        unsigned long  y;

        d = x & 0x00FF;
        x >>= 8;
        c = x & 0x00FF;
        x >>= 8;
        b = x & 0x00FF;
        x >>= 8;
        a = x & 0x00FF;
        y = S[0][a] + S[1][b];
        y = y ^ S[2][c];
        y = y + S[3][d];

        return y;
}

/*************************************
* Modified Code with CIs inserted
*************************************/
extern unsigned long S[4][256];
unsigned long F (
    unsigned long x)
{
   int _dummy_RS = 1, _dummy_RT = 1;
   unsigned long *t58, *t66, *t84, *t75;
   unsigned long t85,  ldB29C1, ldB28C0;
   unsigned long ldB41C2, ldB53C3;
   //
   // C Code for Block 1
   //
   t58 = CI_1 ( x, S);
   t66=CI_Unload_Internal (_dummy_RS, _dummy_RT, 3);

   ldB29C1 = * /* load  */ t66;
   ldB28C0 = * /* load  */ t58;

   t84 = CI_2 ( ldB28C0, ldB29C1);
   t75=CI_Unload_Internal (_dummy_RS, _dummy_RT, 5);

   ldB41C2 = * /* load  */ t75;
   ldB53C3 = * /* load  */ t84;
   t85 = CI_3 ( ldB53C3, ldB41C2);
   return t85;
```

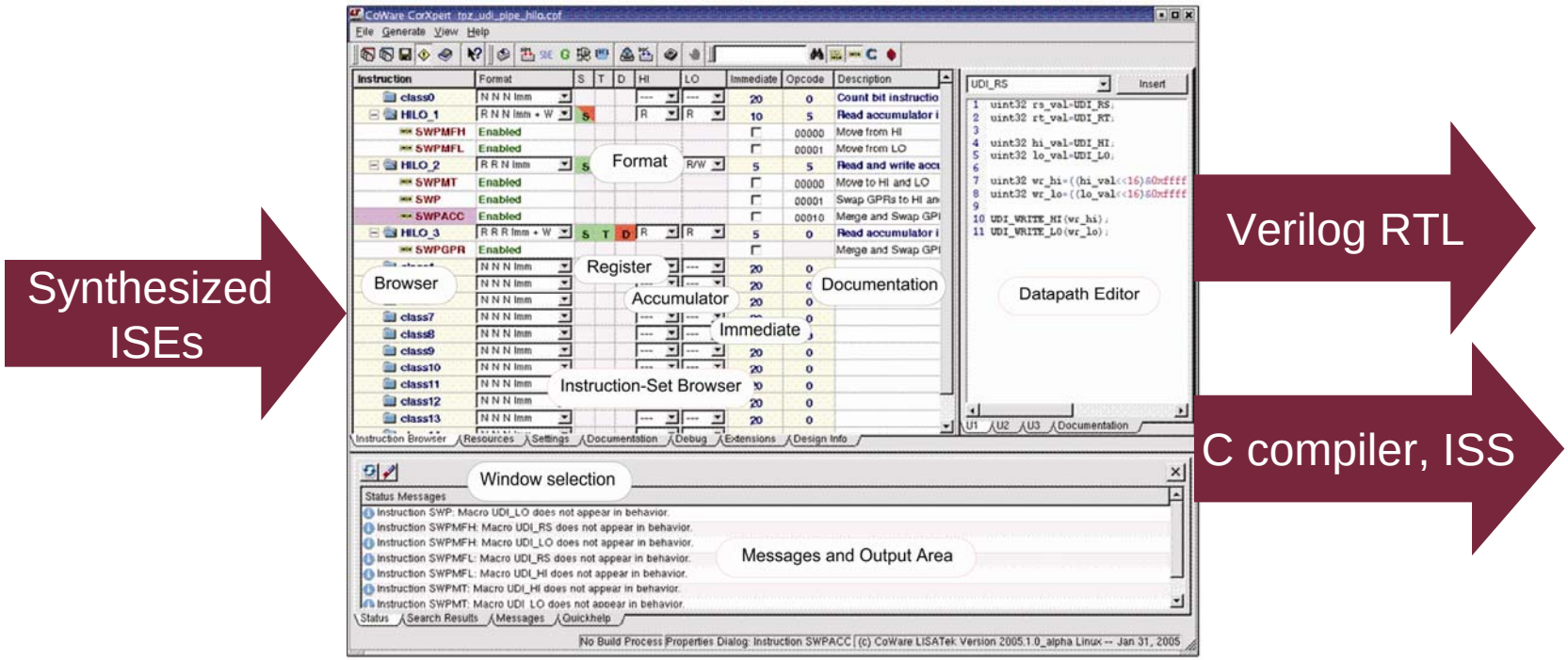
ISE design tools user interface



(3)+(4) CI implementation + SW tools adaptation

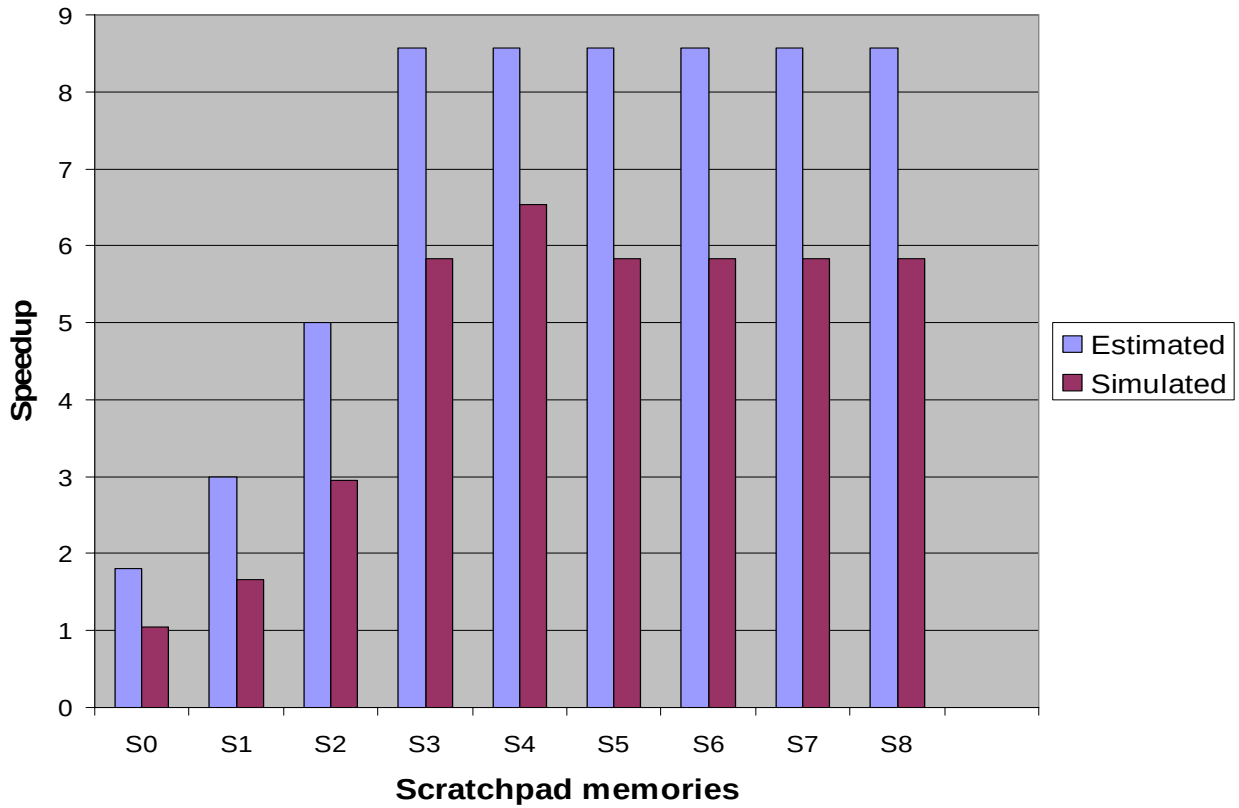
CoWare CorXpert tool

- C compiler, ISS retargeting, HDL generation for ISE
- Currently support for MIPS CorExtend
- Custom extensions to native MIPS tools
- Used as „backend“ here



Preliminary ISE results

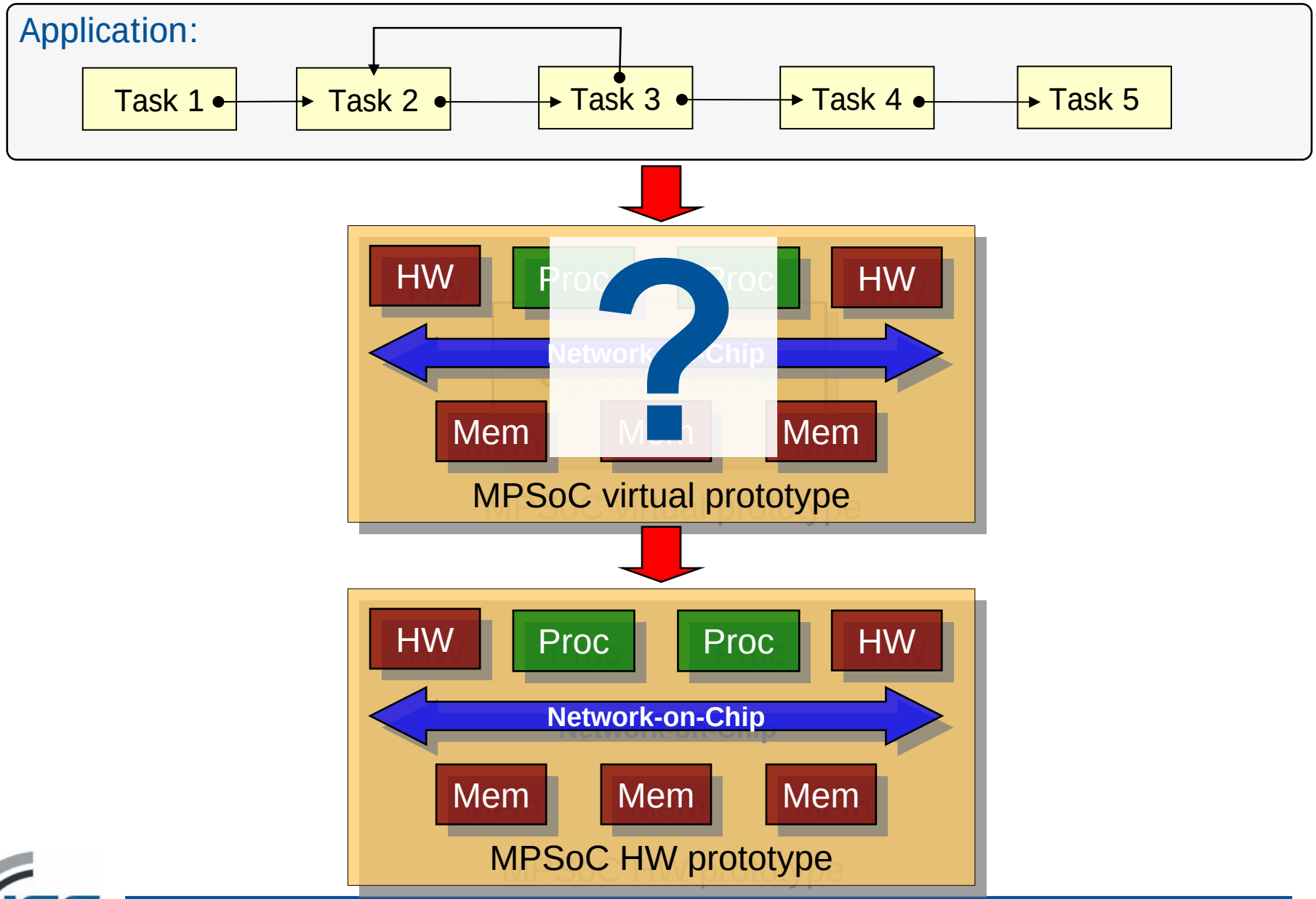
- Good speedup with few ISEs for simple kernels (e.g. DES)
- „Incremental“ improvements required: runtime, estimation, ...
- In progress: H.264 reconfigurable MPSoC case study



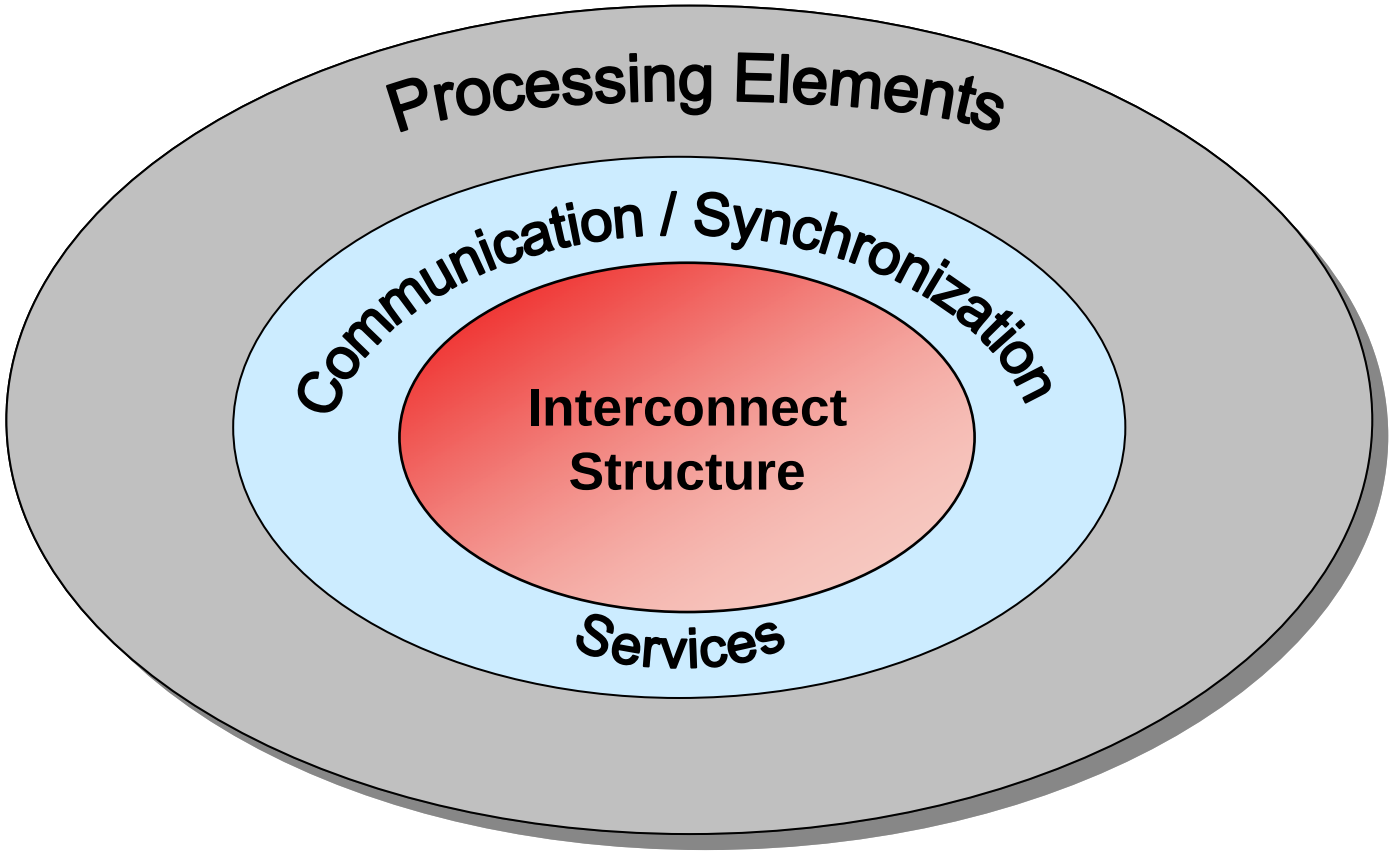
Overview

1. Customizable embedded processor design flow
2. MPSoC SW performance estimation framework
3. Future work

MPSoC design flow

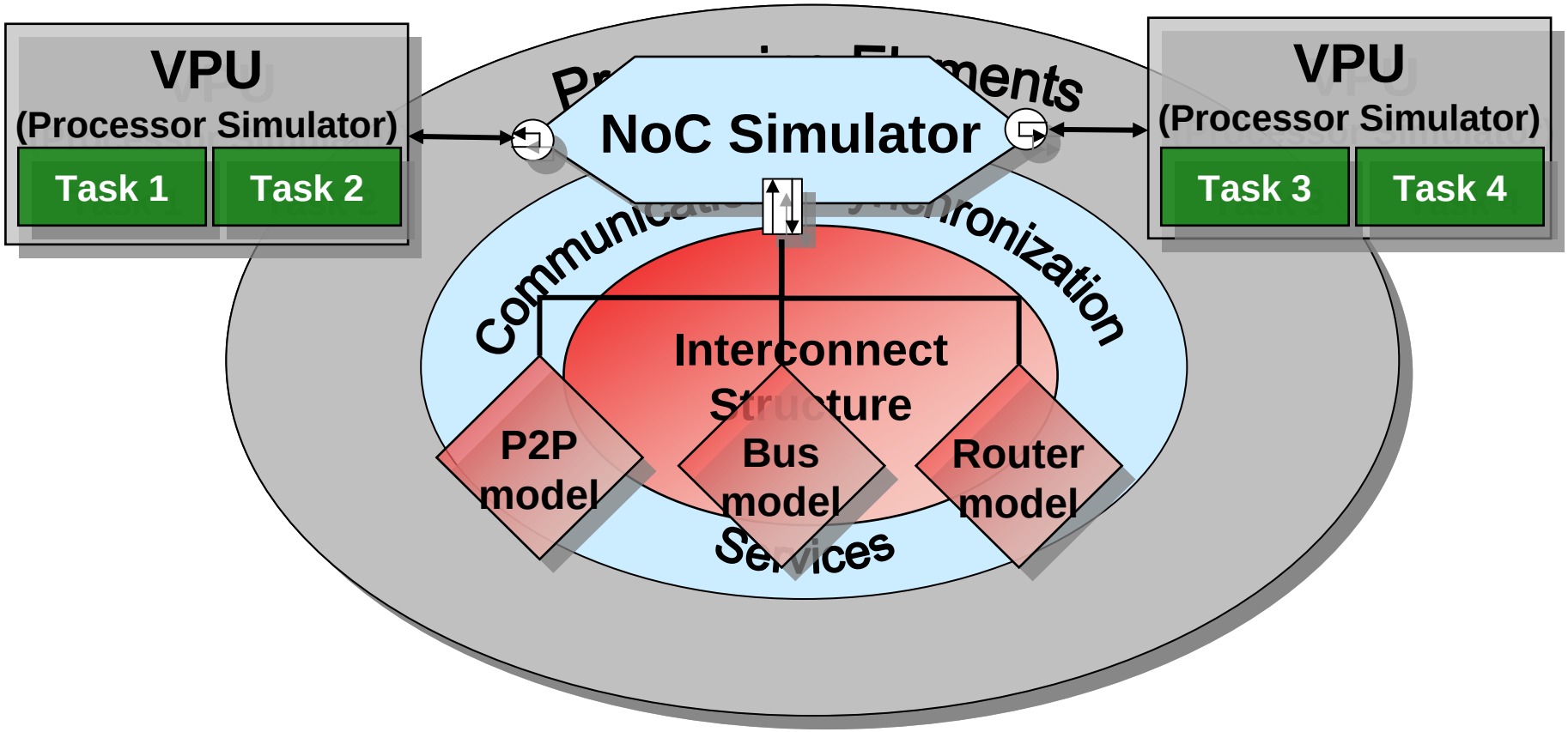


MPSoC exploration principles



- Divide and conquer
- Separate processing elements from communication
- Early SW performance estimation

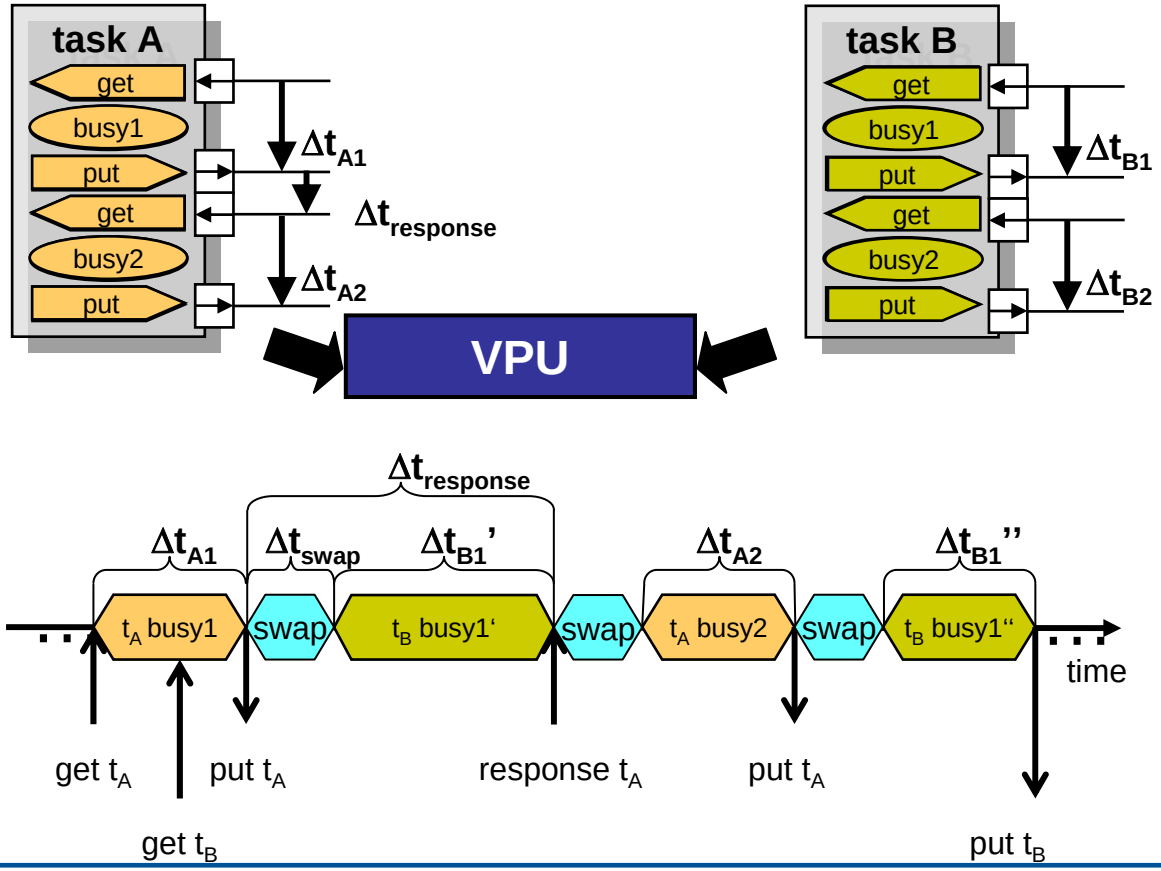
MPSoC virtual prototyping platform



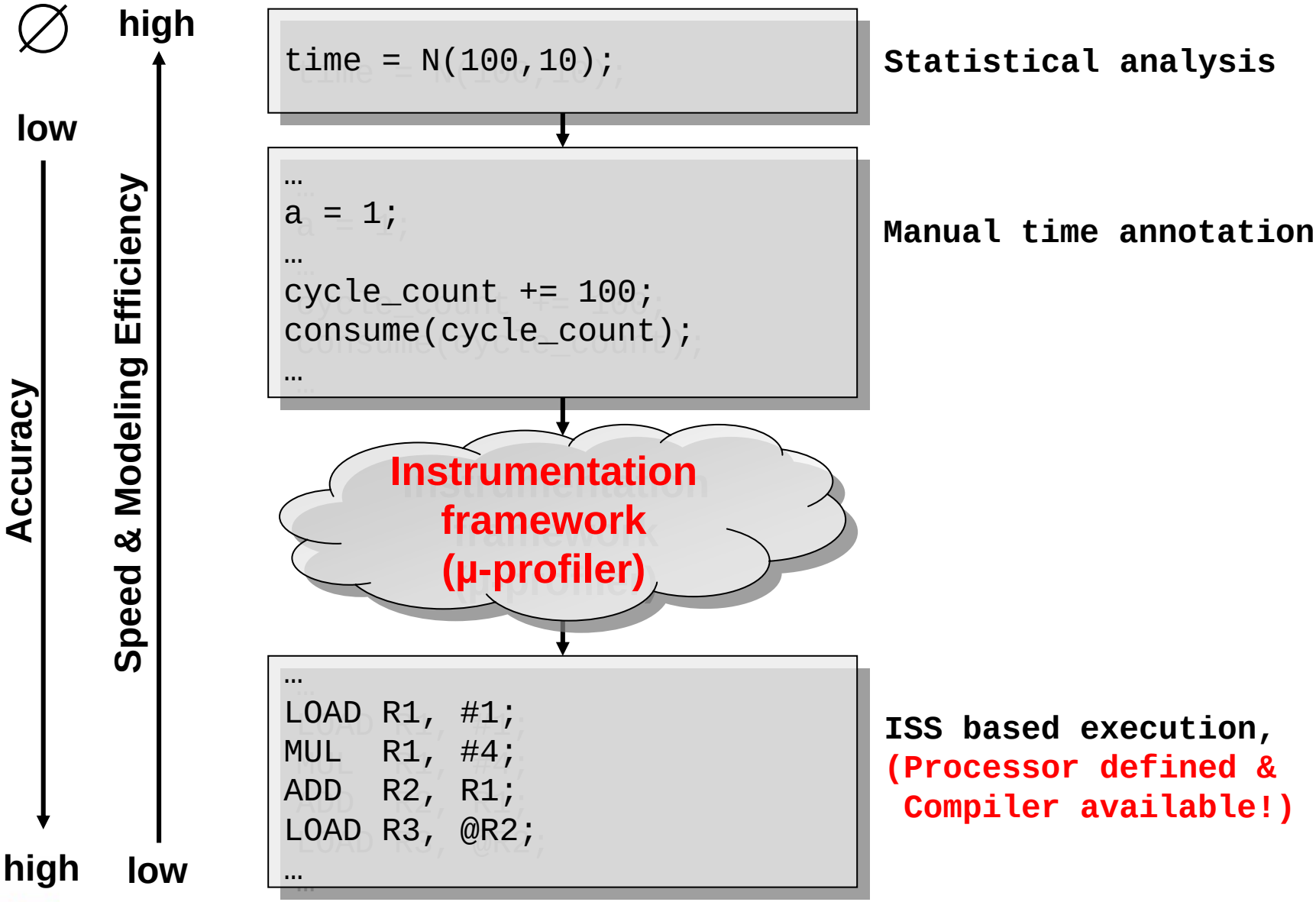
- Communication: CoWare Architect´s View Framework (AVF)
- VPU: virtual processing unit
- Enables modeling spatial and temporal task-to-PE mapping

VPU concept

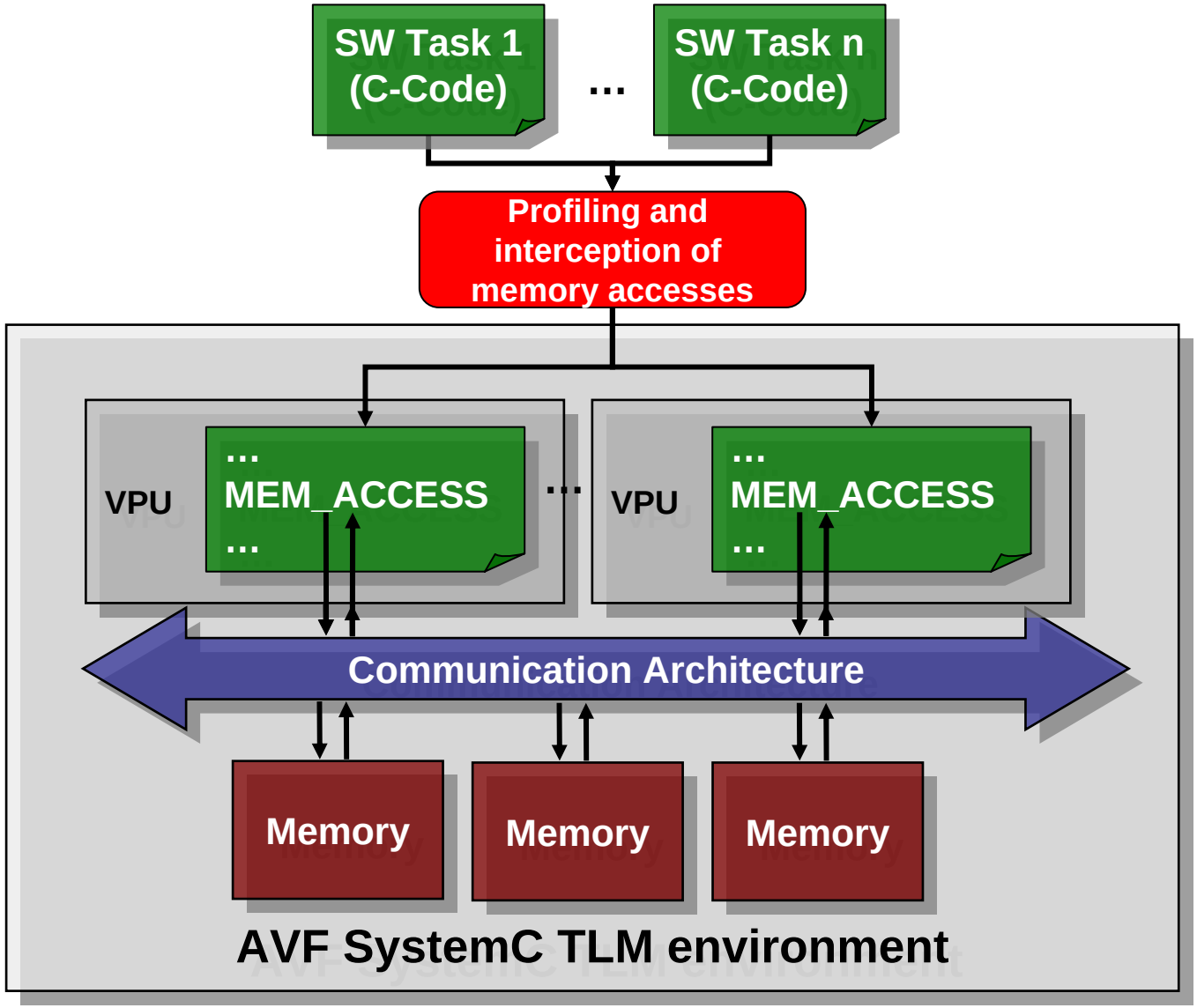
- „Pre-architecture“ exploration: abstract ISA modeling w/o ISS (native task C code execution)
- Abstract OS modeling (task context switch times)
- Problem: task timing and memory access (over NoC) modeling



Fast and accurate SW performance estimation

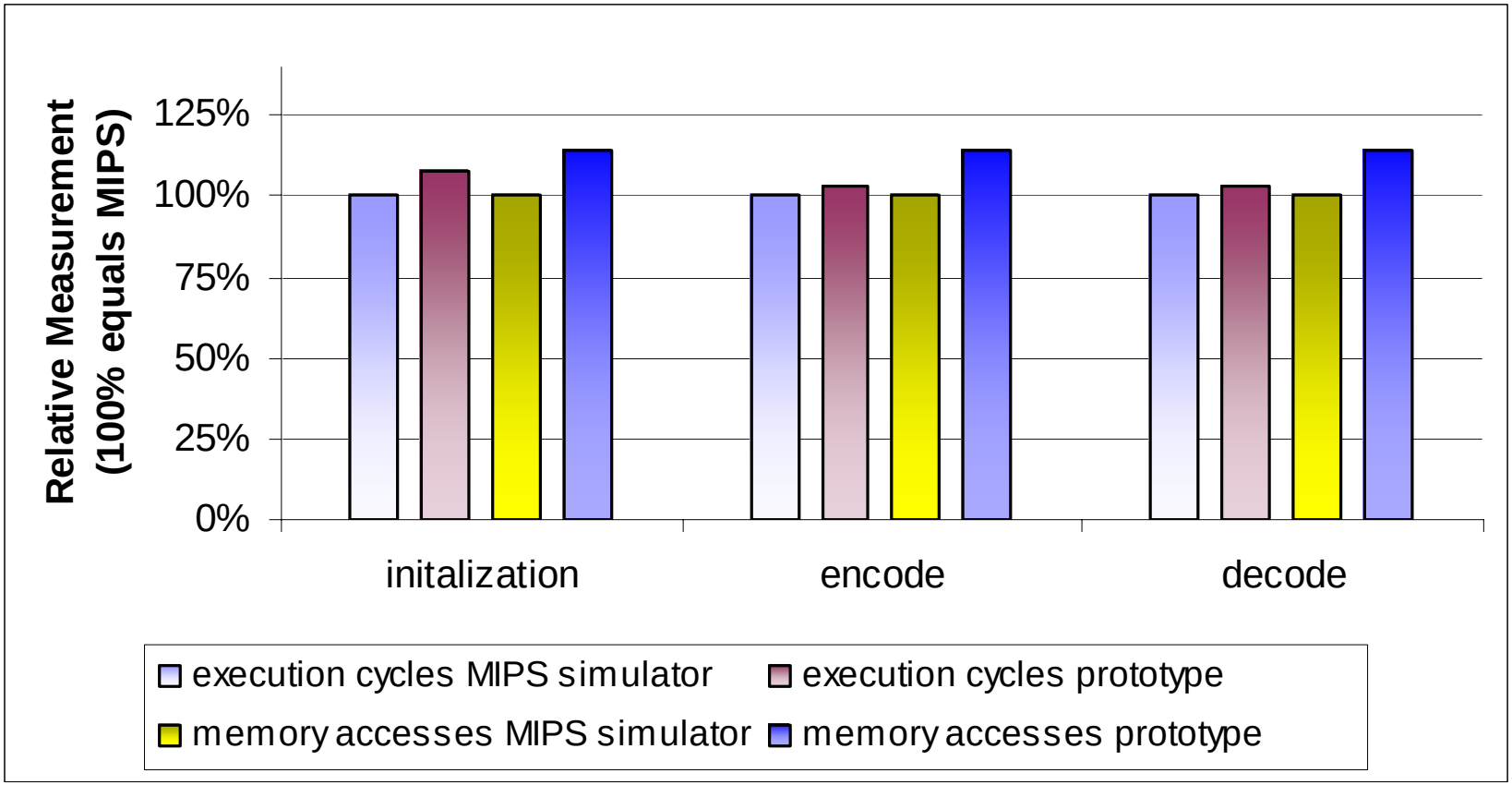


VPU/NoC co-exploration



Preliminary results for MIPS/Blowfish

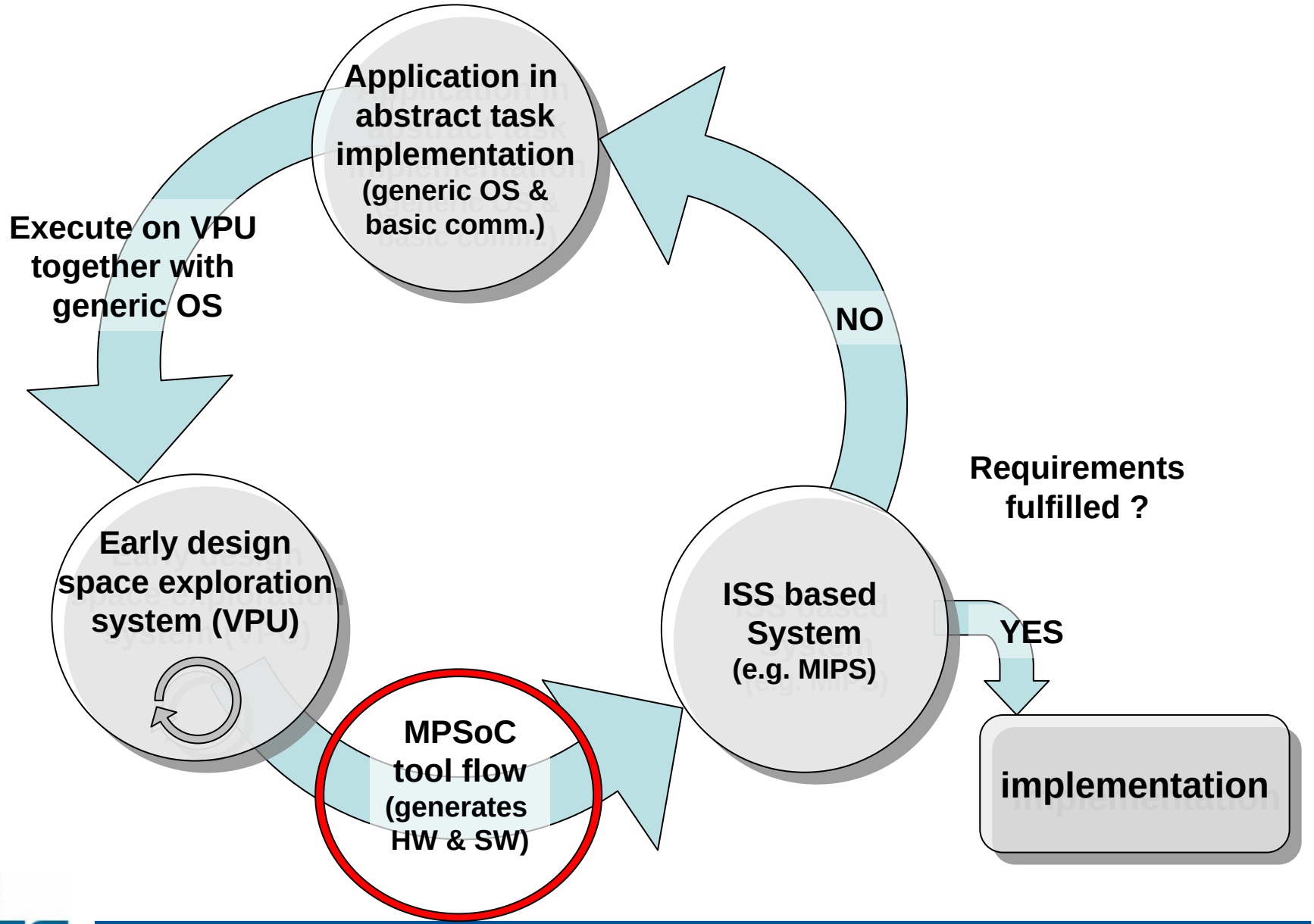
- High accuracy of VPU vs. MIPS instruction accurate ISS
- Need to analyze accuracy vs. cycle accurate ISS



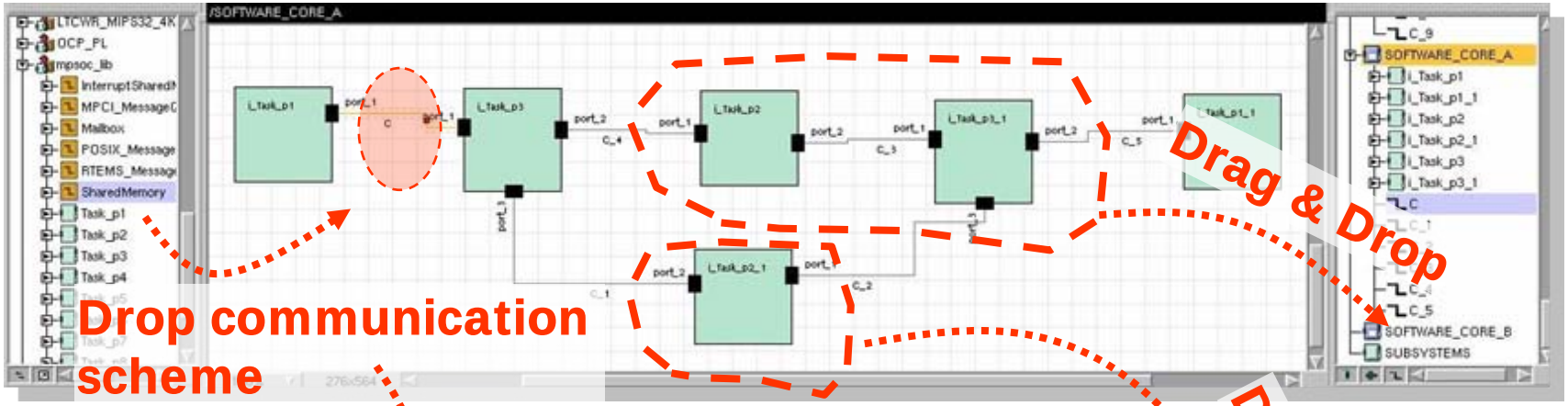
Overview

1. Customizable embedded processor design flow
2. MPSoC SW performance estimation framework
3. Future work

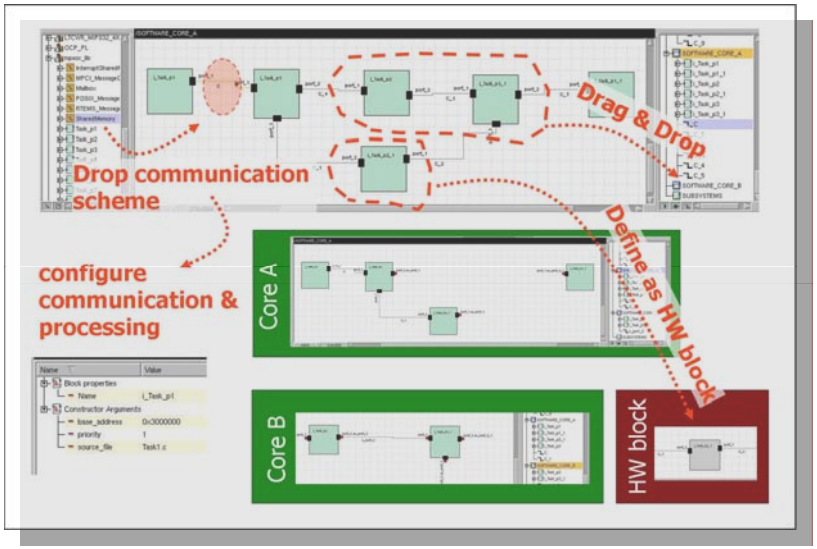
Automated MPSoC VP refinement

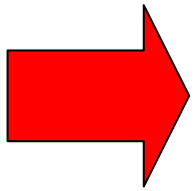


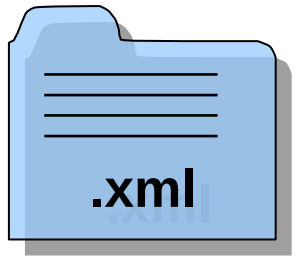
Task mapping tools



Automated MPSoC VP refinement

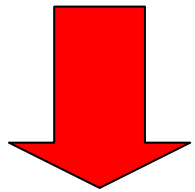






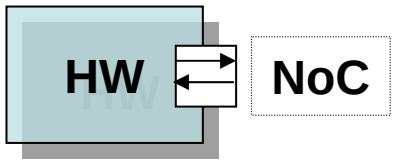
System description

MPSoC tools

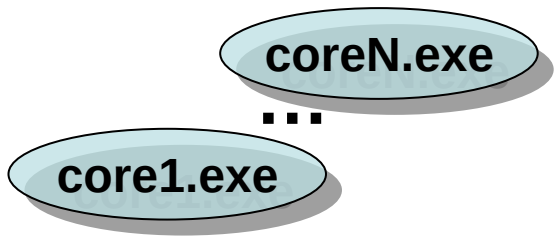


Replace generic OS & communication scheme

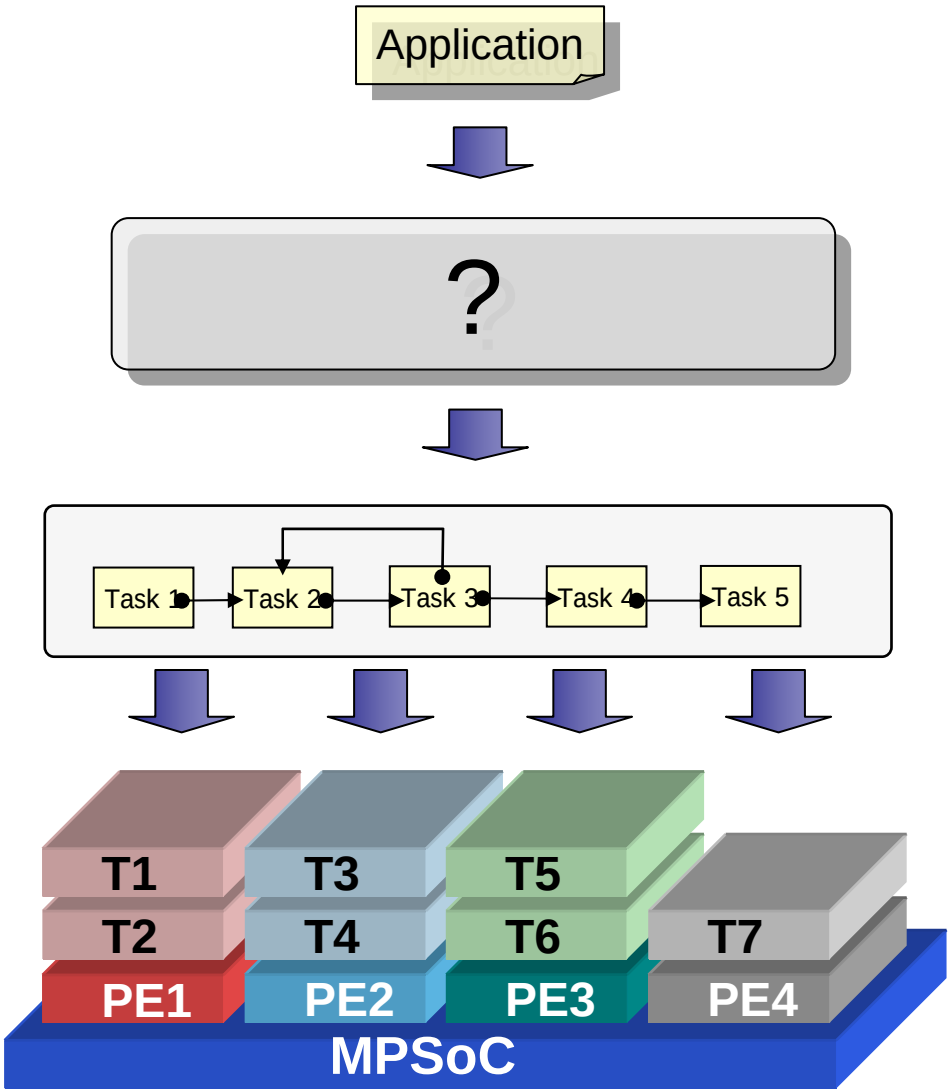
HW parts:



SW parts:
e.g. MIPS & RTEMS

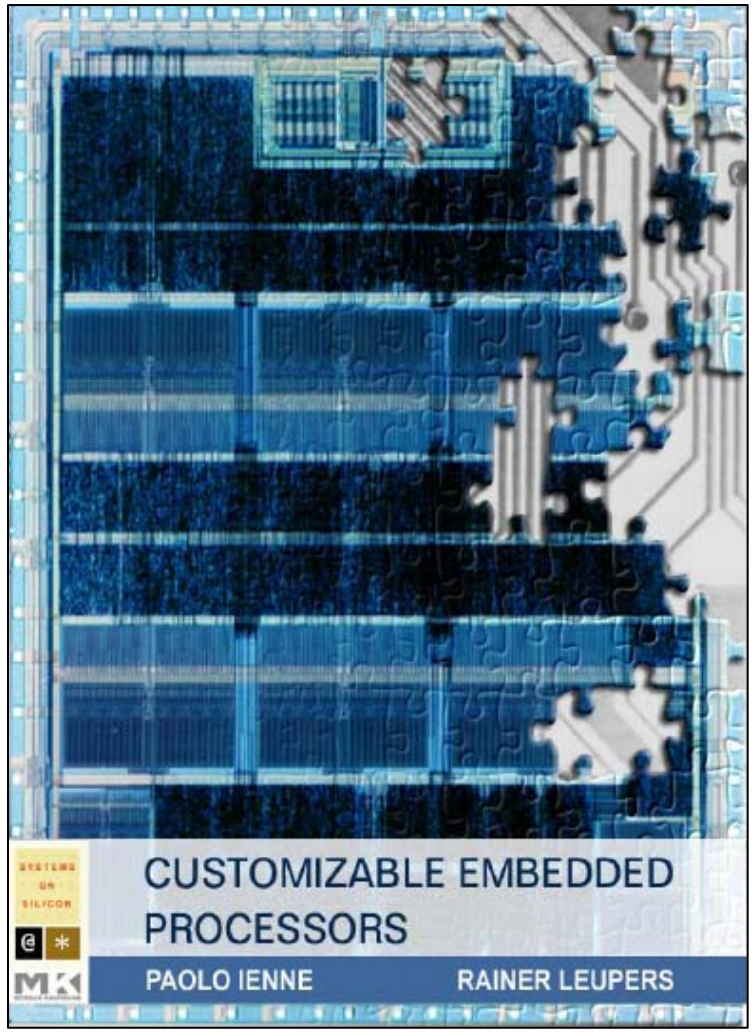


MPSoC programming: task graph generation/extraction



- How to obtain task graphs?
- Specification is not always given in a parallelized form, but e.g. as sequential C code
- TGFF is not the final solution
- Tool requirements
  - Semi-automatic parallelization
  - Help MPSoC programmers to extract the implicit parallelism inside an application
  - Compiler and profiling technology will be key for task graph generation from C/C++

Thank you !



Morgan Kaufmann @ DAC 2006