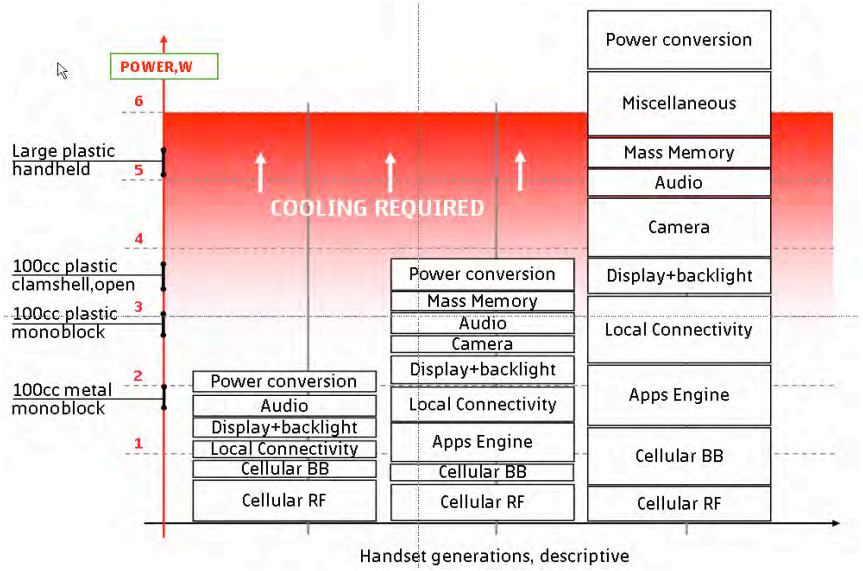
ISS/SSS Research Overview (System-level design tools)

Rainer Leupers
Software for Systems on Silicon (SSS)
RWTH Aachen University

Research hypotheses

- **Future embedded architecture styles**
  - Domain specific MPSoC platforms
  - Standard CPUs and customizable processors as building blocks
  - Network-on-chip based interconnect
- **Consequence**
  - Need for novel ESL tools
  - Increasing emphasis on eSW design tools



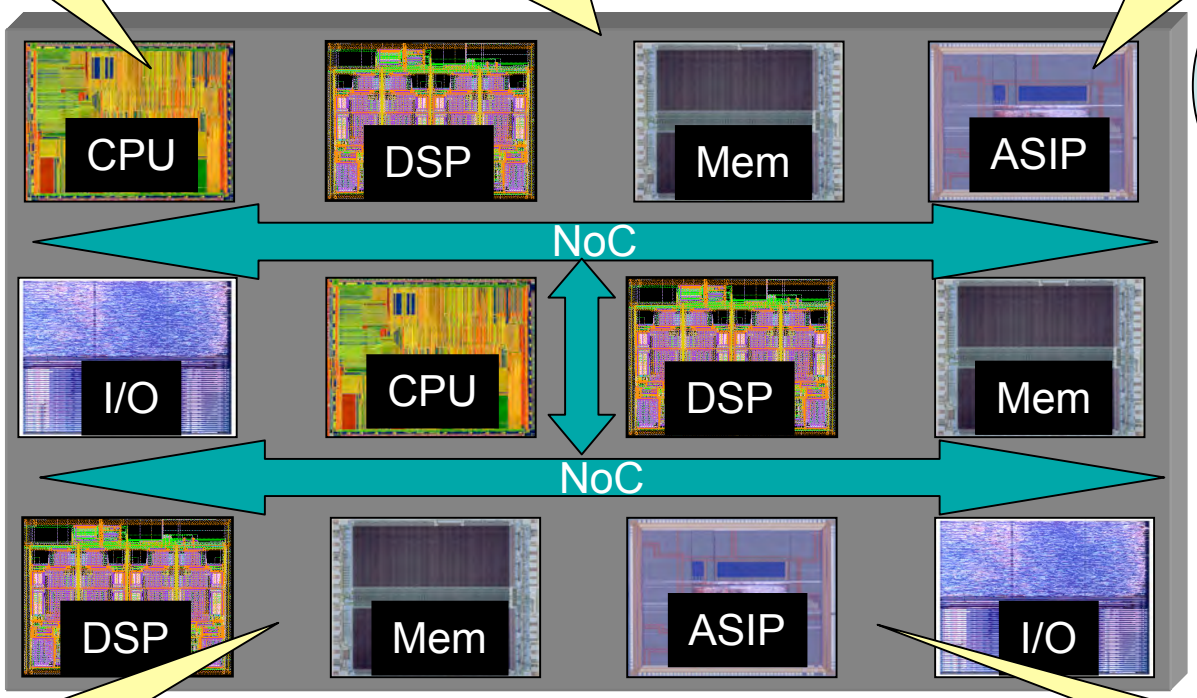
[Nokia]

	2003	2009	2013
Frequency (MHz)	**300**	**600**	**1500**
Giga Operations per Second	**0.3**	**14**	**2458**
Operations per Cycle	**1**	**23**	**1639**

ITRS roadmap for wireless terminals

© 2007 R. Leupers

Addressing MPSoC ESL tool challenges



C compilers
- retargeting
- code optimization

MPSoC platform exploration
- NoC explor
- PE explorat
- refinement

Torsten Kempf

Design tools
- µ-profiling
- ISA extensions
- reconf. ASIP

Kingshuk Karuri, Anupam Chattopadhyay

Virtual platforms
- MPSoC modeling
- Fast hybrid simulation

Stefan Kraemer, Lei Gao

MPSoC programming
- C code partitioning
- Task-to-PE mapping

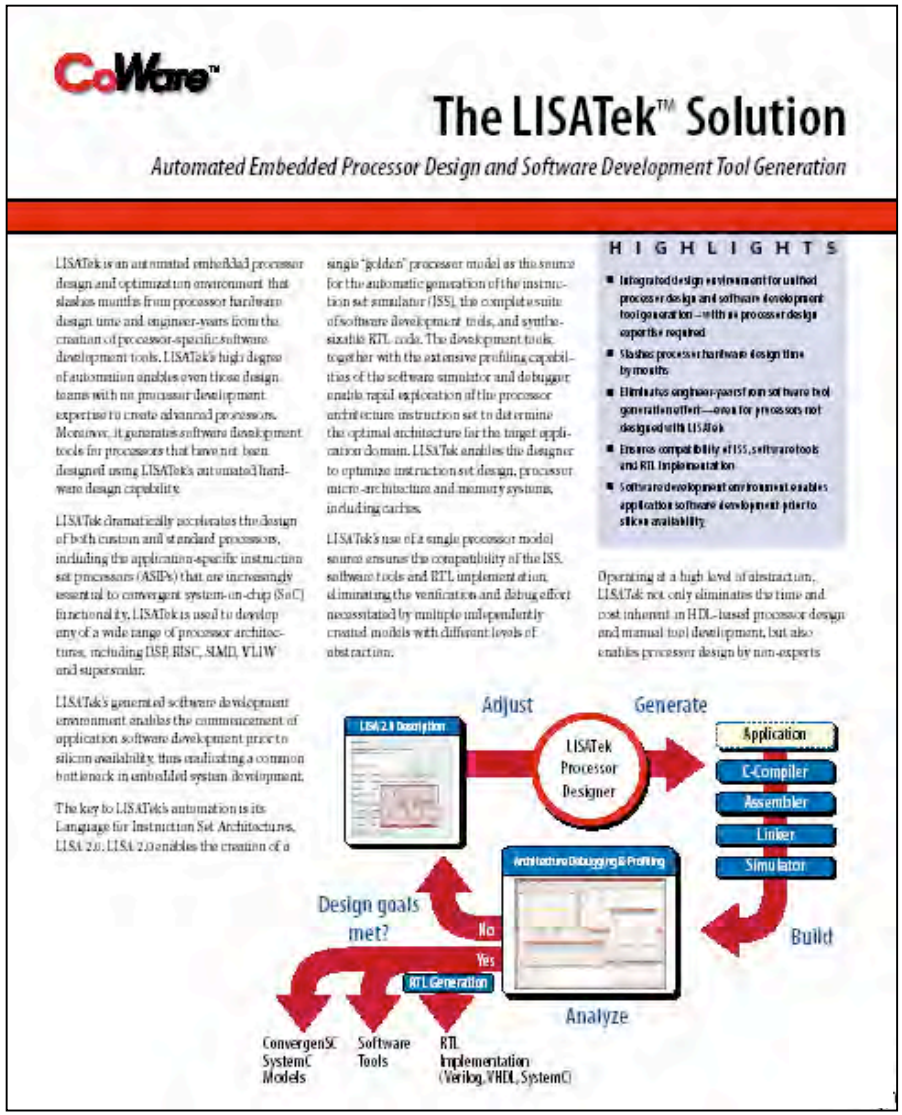
Overview: MPSoC ESL tool challenges

- C compilers
- ASIP design tools
- Fast MPSoC simulation
- MPSoC programming and compilation

© 2007 R. Leupers

LISATek ASIP architecture exploration

- Unified processor model in LISA 2.0 architecture description language (ADL)
- Integrated processor development environment
- Automatic generation of:
  - SW tools
  - HW models
- Now available as CoWare Processor Designer



CoWare™

The LISATek™ Solution

Automated Embedded Processor Design and Software Development Tool Generation

HIGHLIGHTS

- Integrated design environment for unified processor design and software development tool generation – with no processor design expertise required
- Slashes processor/hardware design time by months
- Eliminates engineer-years from software tool generation effort—even for processors not designed with LISATek
- Ensures compatibility of ISS, software tools and RTL implementation
- Software development environment enables application software development prior to silicon availability

LISATek is an automated embedded processor design and optimization environment that slashes months from processor hardware design time and engineer-years from the creation of processor-specific software development tools. LISATek's high degree of automation enables even those design teams with no processor development expertise to create advanced processors. Moreover, it generates software development tools for processors that have not been designed using LISATek's automated hardware design capability.

LISATek dramatically accelerates the design of both custom and standard processors, including the application-specific instruction set processors (ASIPs) that are increasingly essential to convergent system-on-chip (SoC) functionality. LISATek is used to develop any of a wide range of processor architectures, including DSP, RISC, SIMD, VLIW and superscalar.

LISATek's generated software development environment enables the commencement of application software development prior to silicon availability, thus eradicating a common bottleneck in embedded system development.

The key to LISATek's automation is its Language for Instruction Set Architectures, LISA 2.0. LISA 2.0 enables the creation of a single "golden" processor model as the source for the automatic generation of the instruction set simulator (ISS), the complete suite of software development tools, and synthesizable RTL code. The development tools, together with the extensive profiling capabilities of the software simulator and debugger, enable rapid exploration of the processor architecture: instruction set to determine the optimal architecture for the target application domain. LISATek enables the designer to optimize instruction set design, processor micro-architecture and memory systems, including caches.

LISATek's use of a single processor model source ensures the compatibility of the ISS, software tools and RTL implementation, eliminating the verification and debug effort necessitated by multiple independently created models with different levels of abstraction.

Operating at a high level of abstraction, LISATek not only eliminates the time and cost inherent in HDL-based processor design and manual tool development, but also enables processor design by non-experts.

© 2007 R. Leupers

LISATek C compiler generation

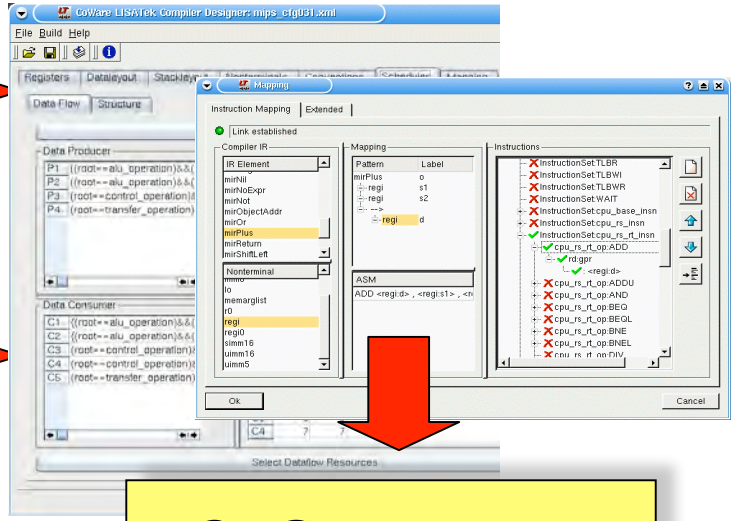
LISA processor model

```
SYNTAX      {
      "ADD" dst, src1, src2
}
CODING      {
      0b0010 dst src1 src2
}
BEHAVIOR    {
      ALU_read (src1, src2);
      ALU_add ();
      Update_flags ();
      writeback (dst);
}
SEMANTICS   {
      src1 + src2 → dst;
}
...
```

Autom. analyses

Manual refinement

GUI



CoSy system

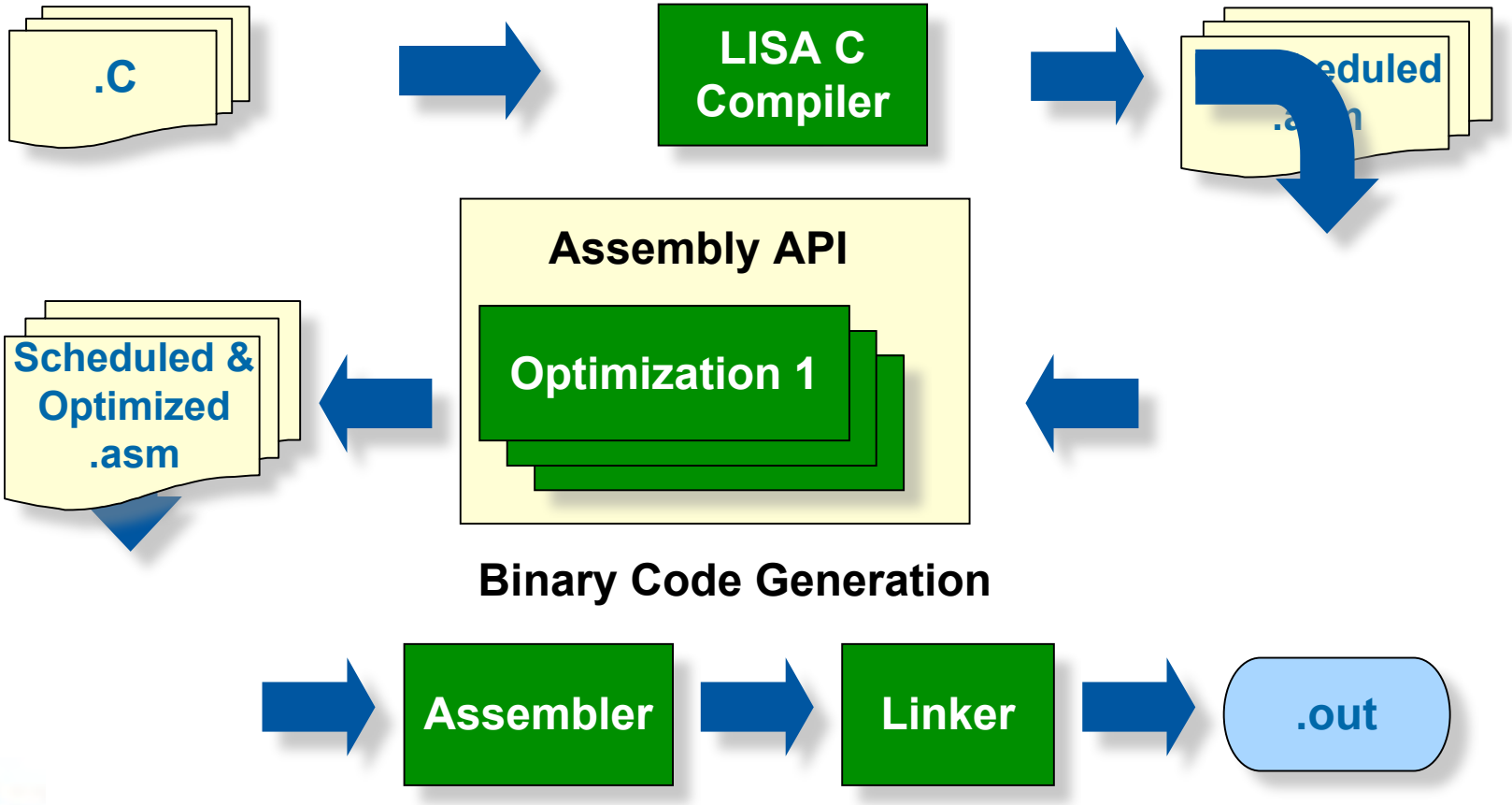
C Compiler

© 2007 R. Leupers

Adding retargetable code optimizations

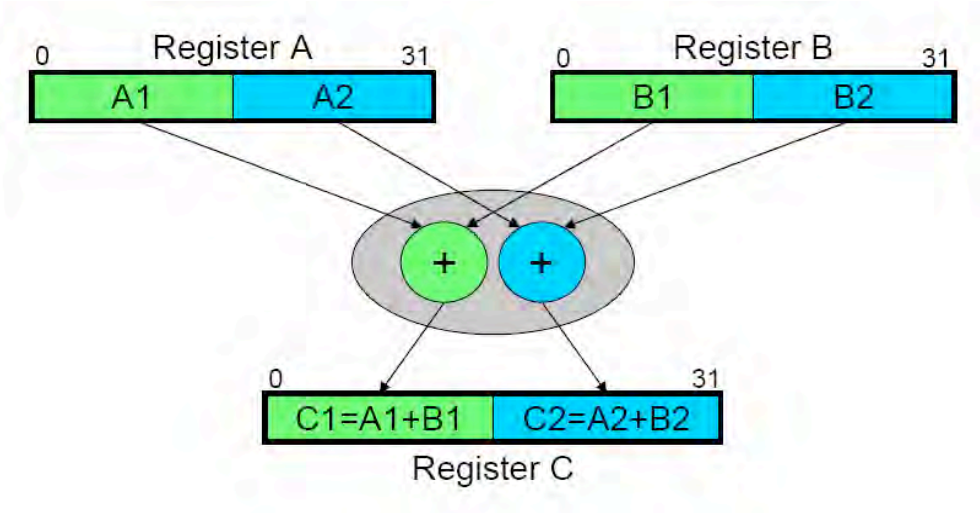
- High-level (compiler IR)
  - Enabled by CoSy´s engine concept

- Low-level (ASM):



© 2007 R. Leupers

Sample retargetable code optimization engines



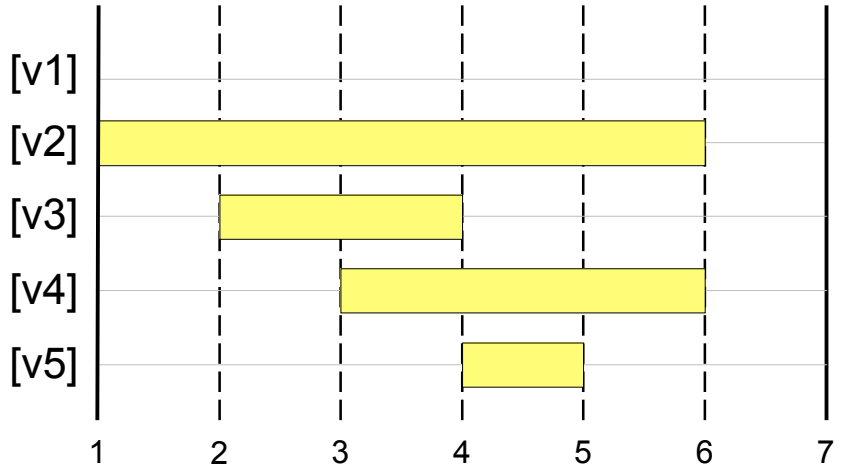
SIMD instructions

```
If ( a > b )
{
  // Then Block
}
Else
{
  //Else Block
}
```

```
      CMP    a,b
      JMPGT Then
      //Else Block
      JMP     EndThen
Then:
      //ThenBlock
EndThen:
```

```
CMPGT a,b,flag
[ flag ]   //ThenBlock
[! flag ] //ElseBlock
```

conditional instructions,
predicated execution



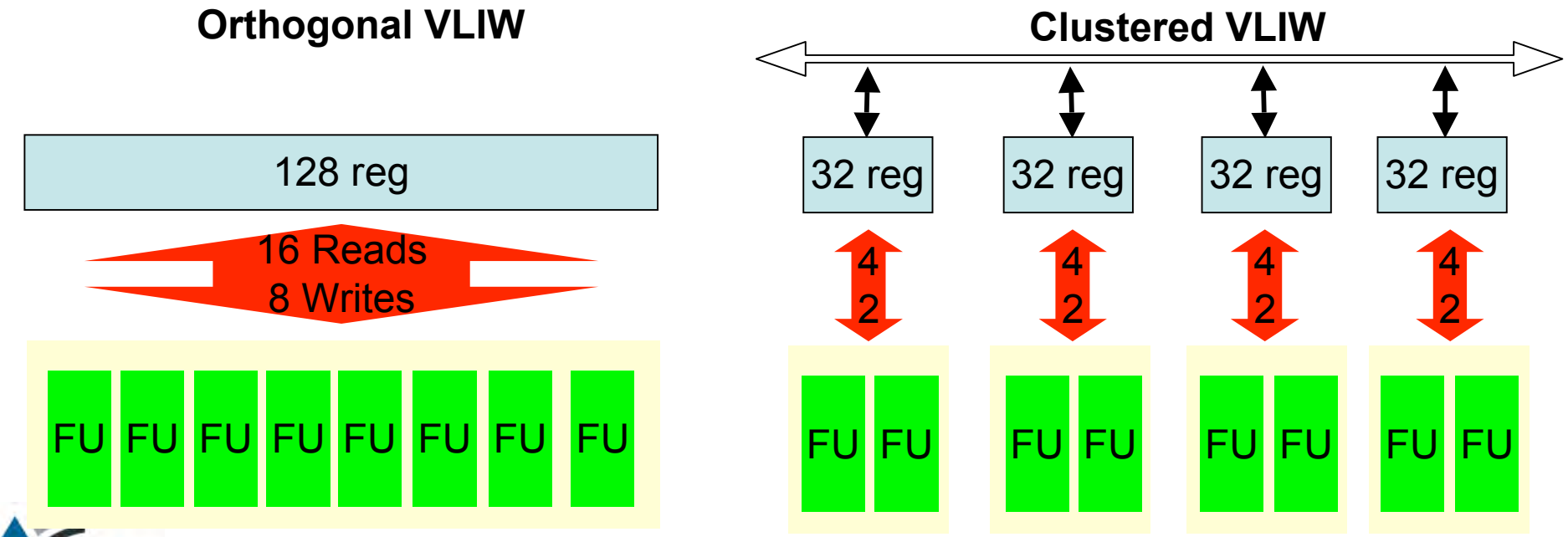
linear scan
register allocation

- [DATE2005, ISSS2006]
- Currently in productization

© 2007 R. Leupers

Compiler support for clustered VLIW

- Important intermediate step between scalar processors and MPSoC
- Reduced # of reg file ports, at the expense of inter-cluster communication code
- Good code quality requires novel compiler techniques (phase coupling, instruction partitioning, SW pipelining)



© 2007 R. Leupers

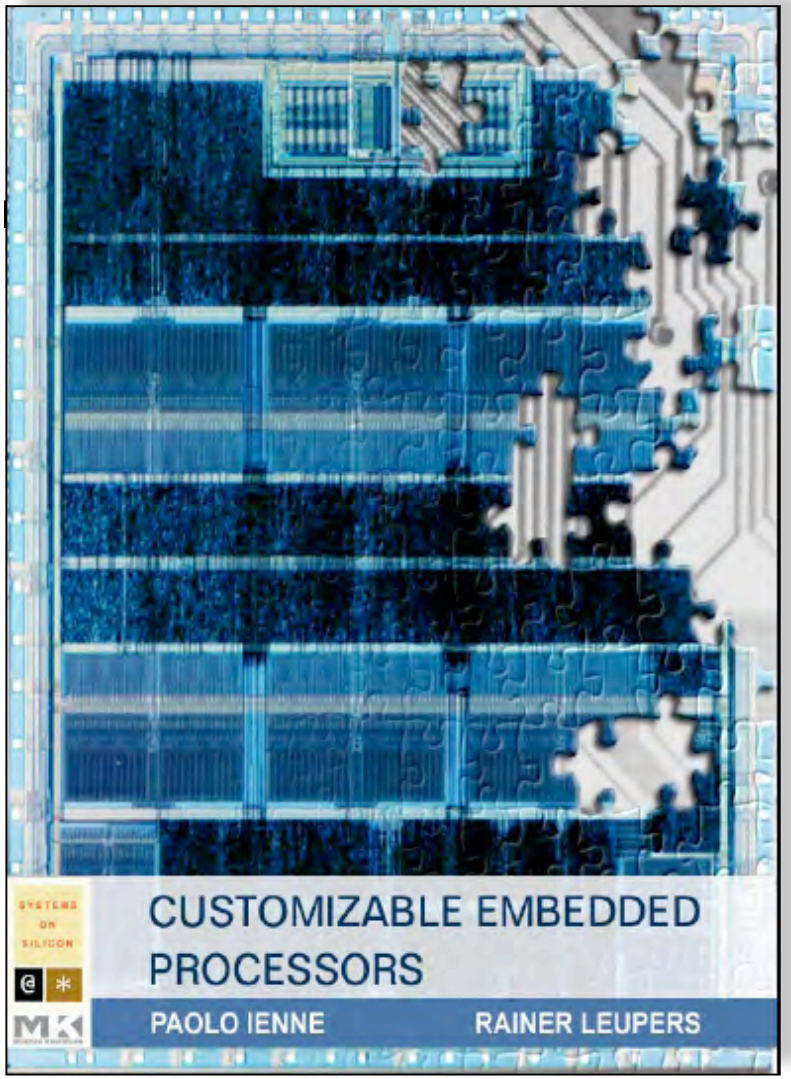
Overview

- C compilers
- ASIP design tools
- Fast MPSoC simulation
- MPSoC programming and compilation

© 2007 R. Leupers

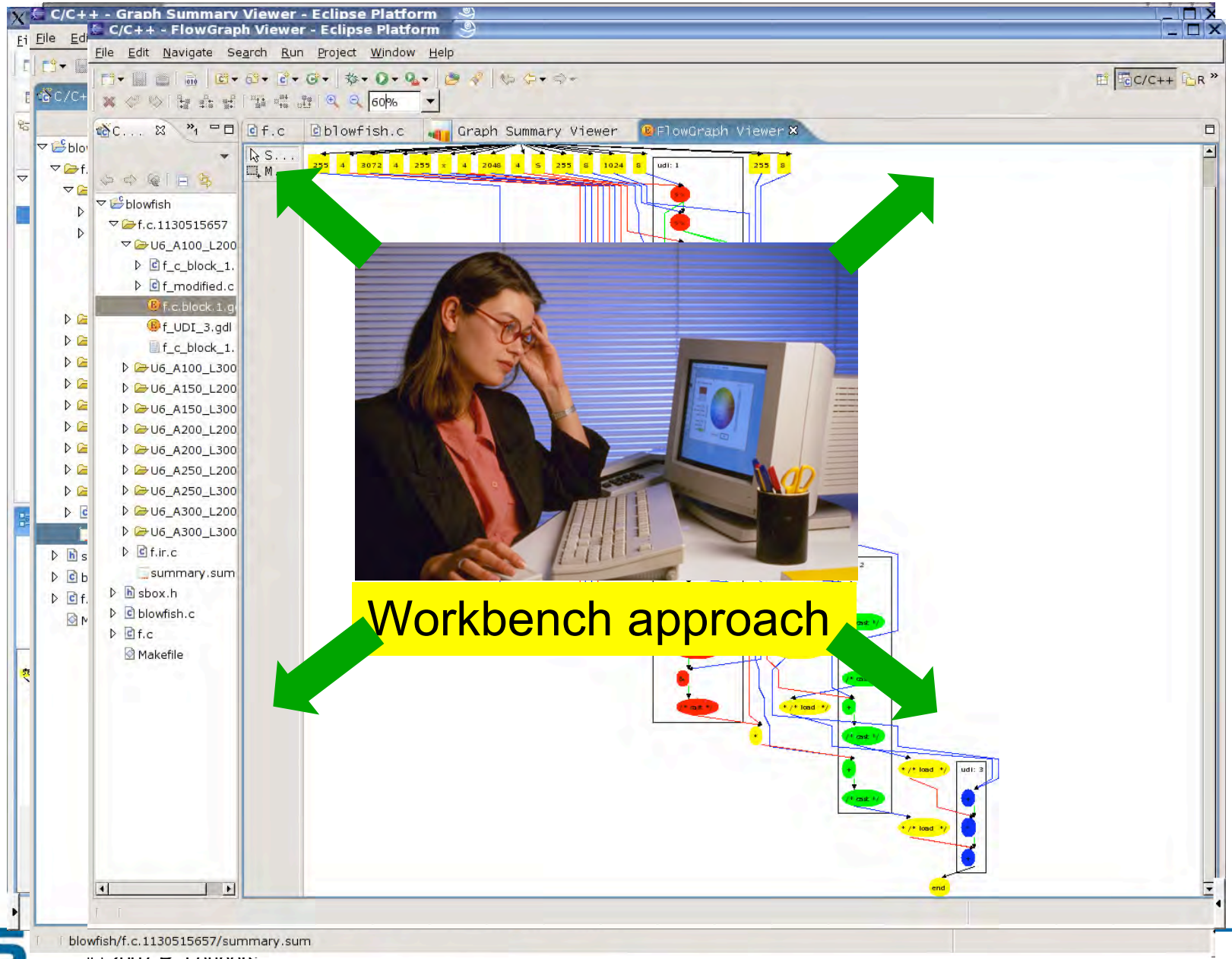
ASIP design technologies

- ADL based (ASIP-from-scratch)
  - E.g. LISATek, Target, Expression
  - Max. flexibility + efficiency, but significa 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



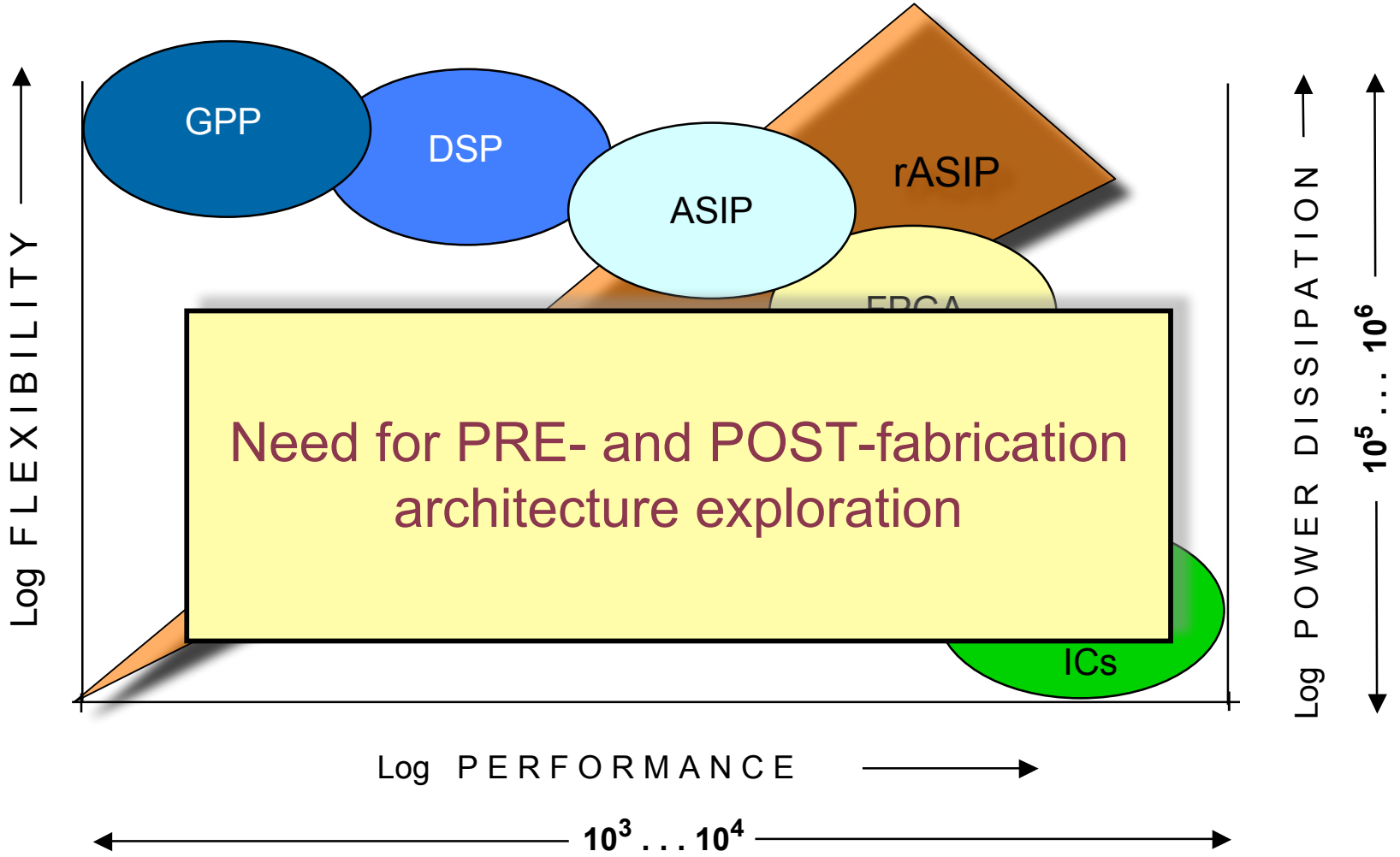
© 2007 R. Leupers

ISE design tools



© 2007 R. Leupers

A New Architecture Class: reconfigurable ASIPs



Source: T.Noll, RWTH Aachen

© 2007 R. Leupers

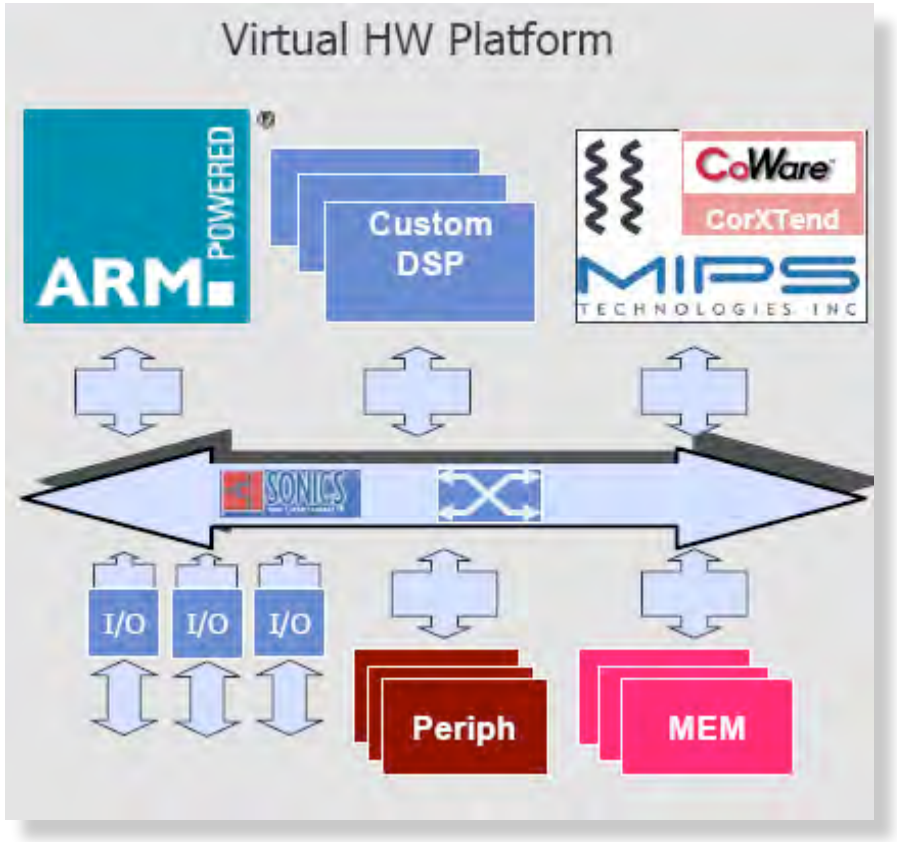
Overview

- C compilers
- ASIP design tools
- Fast MPSoC simulation
- MPSoC programming and compilation

© 2007 R. Leupers

What is a virtual platform?

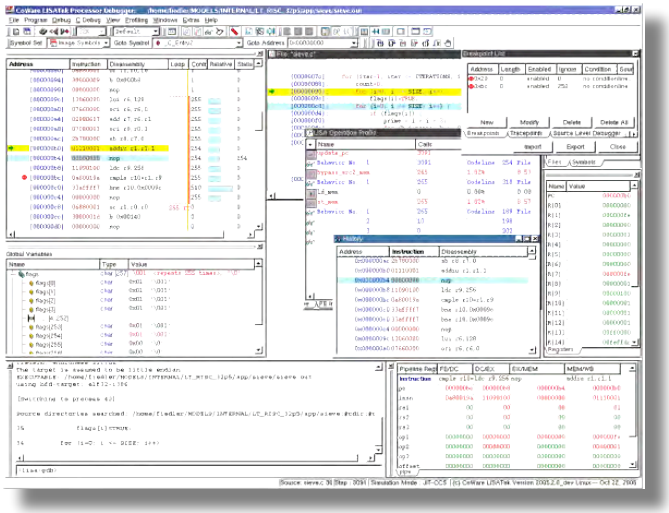
- A SW model of a HW SoC platform
- Enables...
  - HW platform architecture exploration and optimization
  - SW development, debugging, and optimization
  - Concurrent HW/SW design („HW/SW codesign“)
- Requirements
  - High simulation speed
  - Speed/accuracy trade-off
  - Flexibility
  - Usability for non-HW-experts



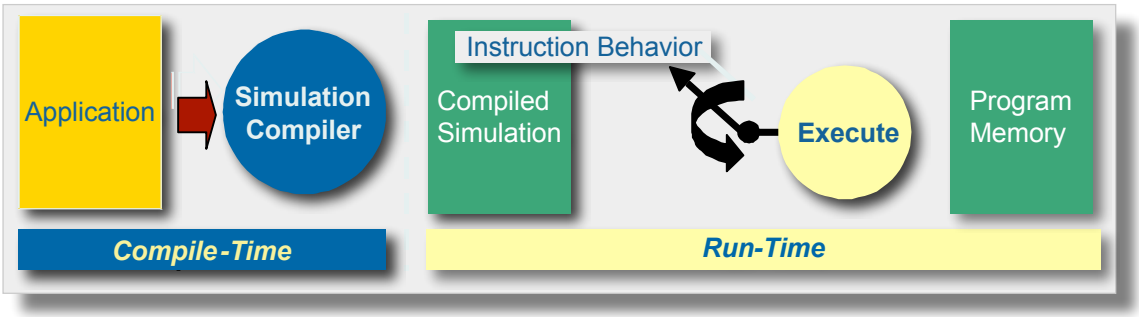
© 2007 R. Leupers

Need for speed

- Instruction set simulator (ISS) is at the heart of virtual platforms
- ISS speed evolution
  - Early interpretive: few KIPS
  - Fast interpretive: ~100 KIPS
  - Compiled: ~10 MIPS
  - SW Sim. Cache: ~10 MIPS
  - Binary translation: 50-100 MIPS

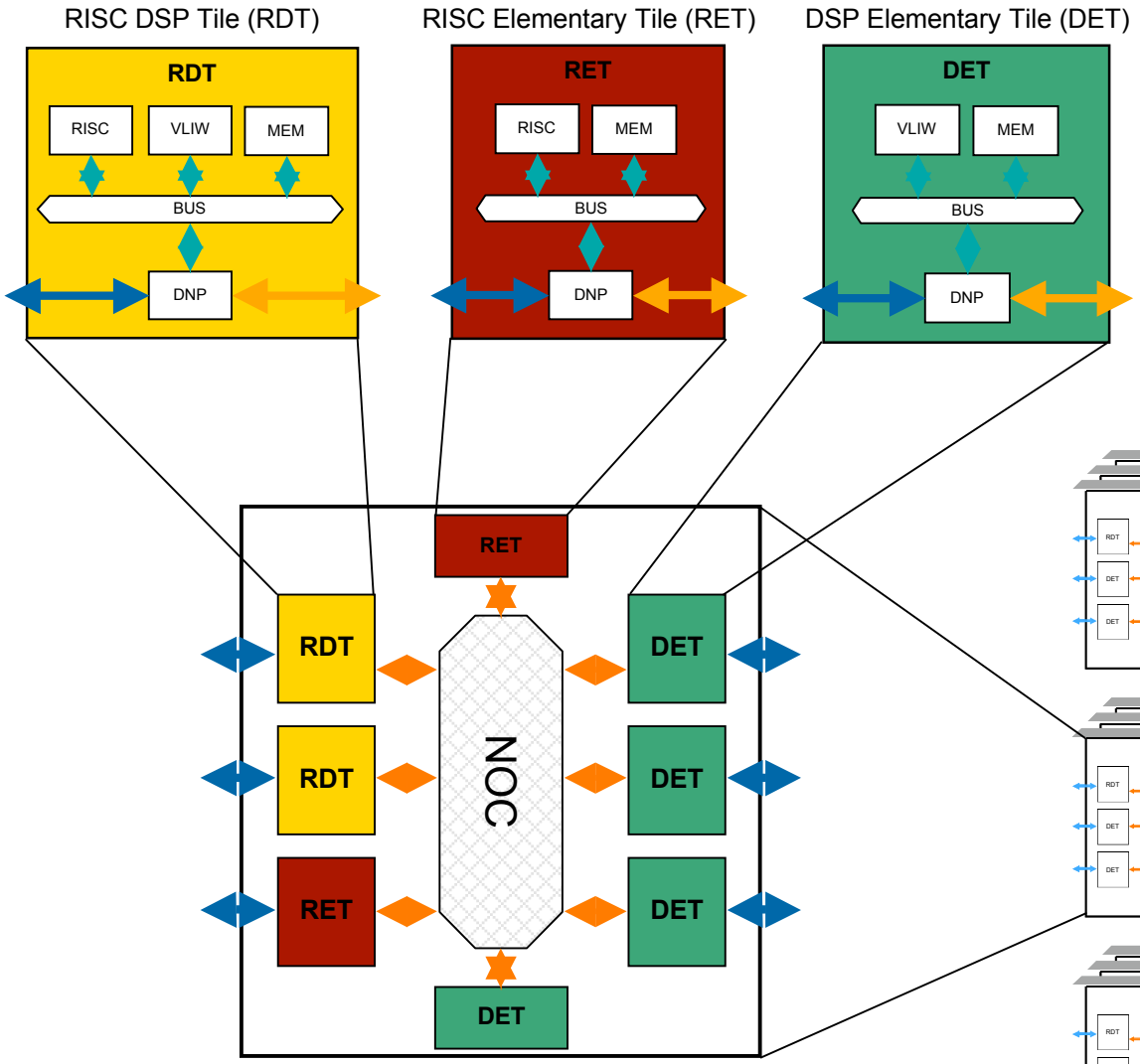


- Challenge:
  - Instruction-accurate ISS technology has been pushed to its limits
  - How to handle future many-core MPSoC?



© 2007 R. Leupers

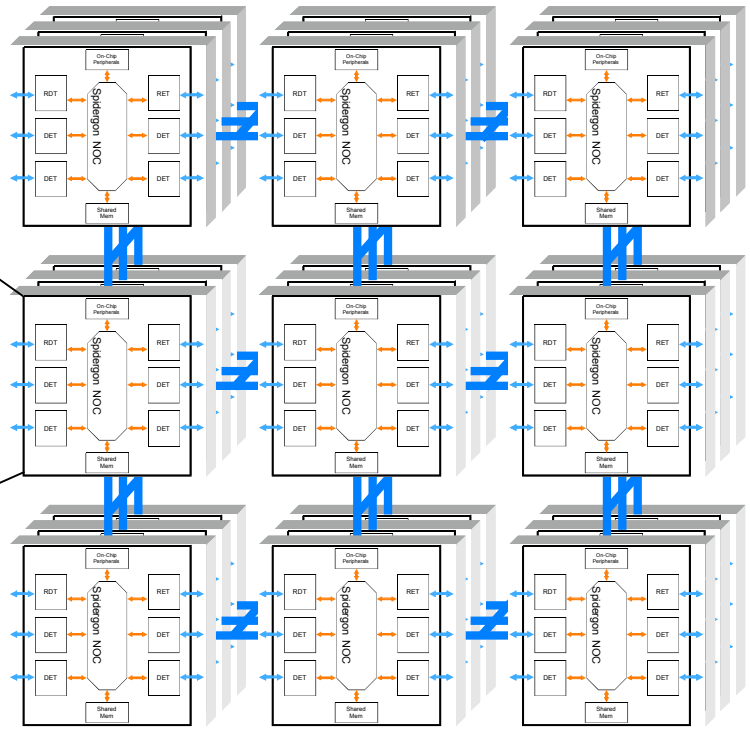
Example: SHAPES platform



Multi-Tile Chip

Scalable HW architecture for high-performance applications

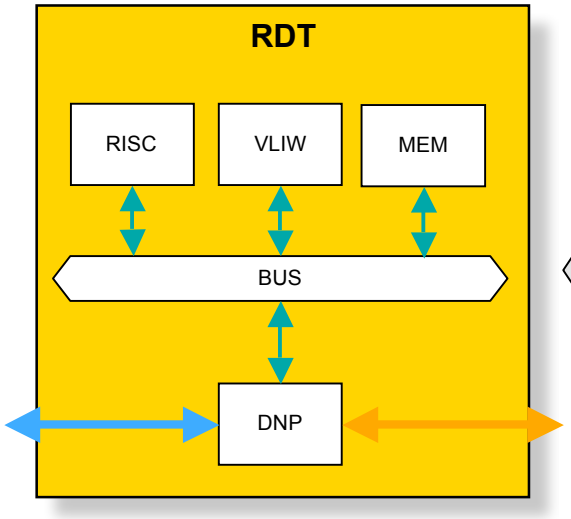
Multi-Chip System

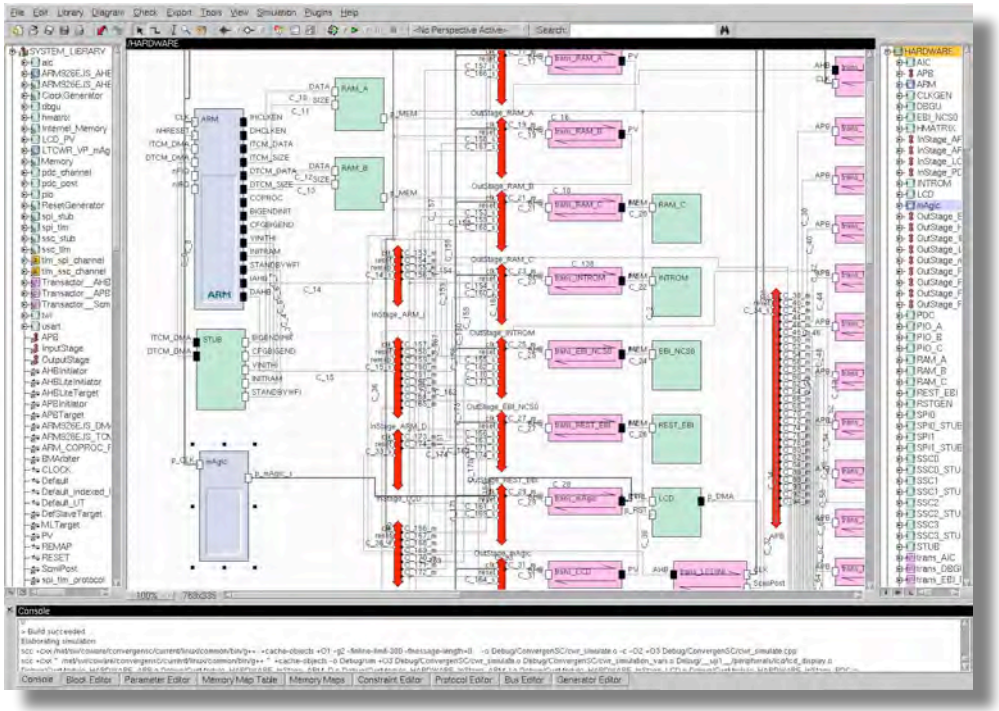


© 2007 R. Leupers

Virtual SHAPES platform

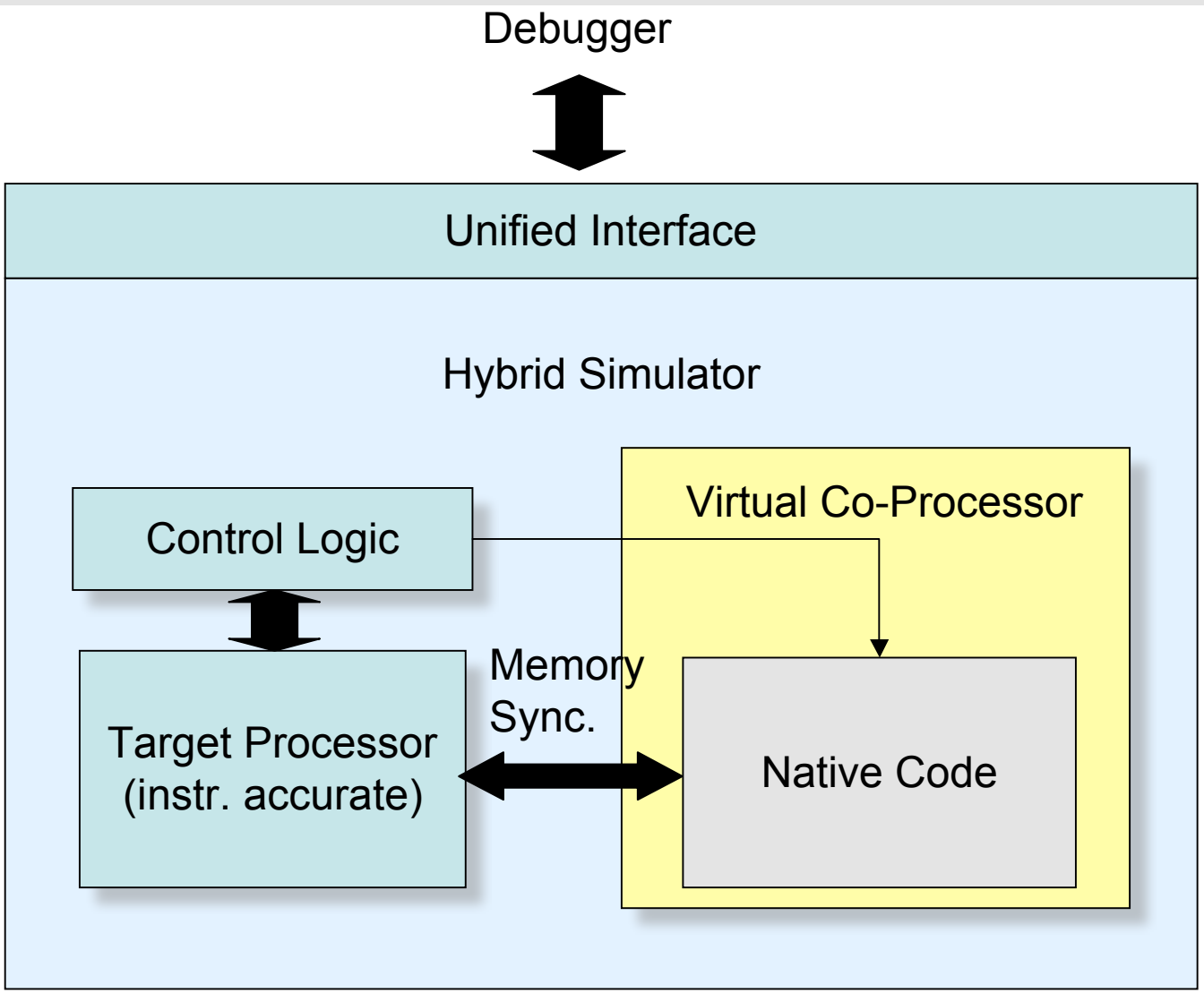
- Single-tile model (ARM9, Magic VLIW DSP, memories, peripherals) developed with CoWare Virtual Platform Designer
- Reasonable speed for one tile
- But: difficult to scale to multi-tile, multi-chip





© 2007 R. Leupers

Hybrid Simulator (HySim) overview



[MPSoC2007, ISSS/CODES2007, EMSOFT2007]

© 2007 R. Leupers

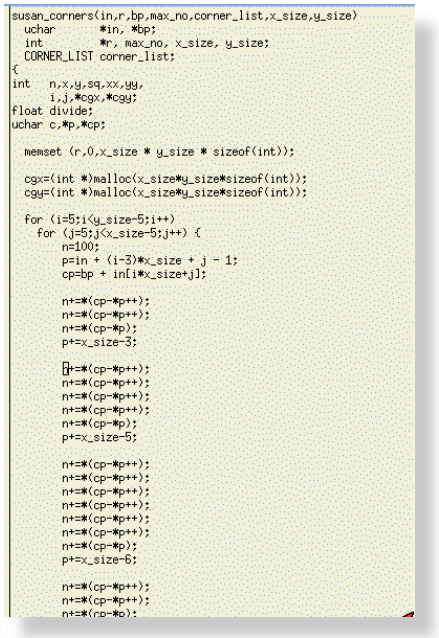
Overview

- C compilers
- ASIP design tools
- Fast MPSoC simulation
- MPSoC programming and compilation

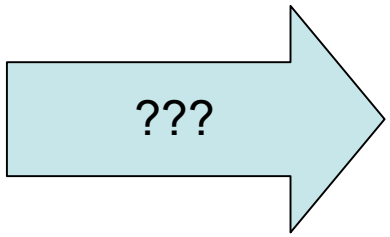
© 2007 R. Leupers

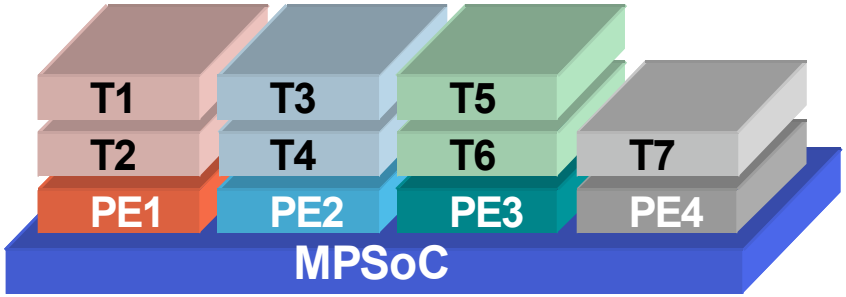
How to map a complex application to MPSoC?

- Current programming languages are sequential
- Parallel programming very tedious
- Need for spatial and temporal mapping
- Need for static and dynamic task creation/scheduling
  - OS or HW IP support
- Ideal case:
  - Compiler performs automated task-to-processor mapping and scheduling



```
susan_corners(in,r,bp,max_no,corner_list,x_size,y_size)
  uchar        *in, *bp;
  int          *r, max_no, x_size, y_size;
  CORNER_LIST corner_list;
{
int   n,x,y,sq,xx,yy,
      i,j,*cgx,*cgy;
float divide;
uchar c,*p,*cp;

  memset (r,0,x_size * y_size * sizeof(int));

  cgx=(int *)malloc(x_size*y_size*sizeof(int));
  cgy=(int *)malloc(x_size*y_size*sizeof(int));

  for (i=5;i<y_size-5;i++)
    for (j=5;j<x_size-5;j++) {
        n=100;
        p=in + (i-3)*x_size + j - 1;
        cp=bp + in[i*x_size+j];

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
        p+=x_size-3;

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
        p+=x_size-5;

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
        p+=x_size-6;

        n+=*(cp-*p++);
        n+=*(cp-*p++);
        n+=*(cp-*p);
```

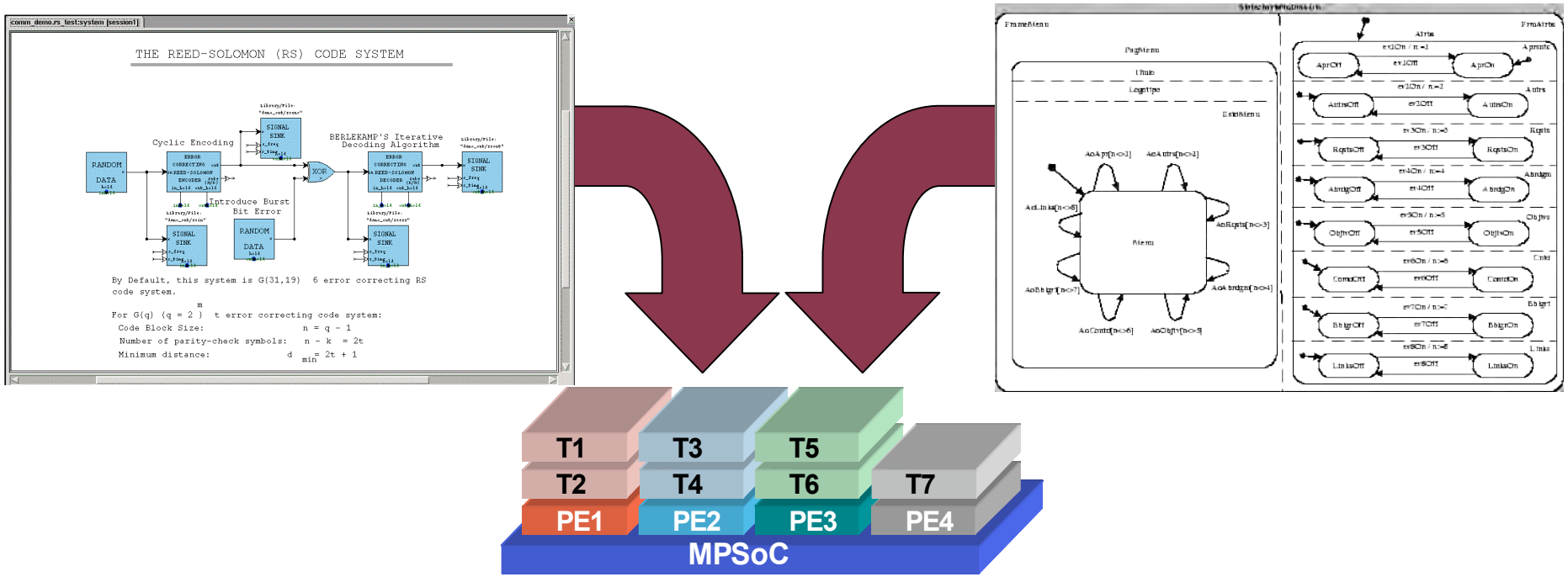




© 2007 R. Leupers

Long term solution

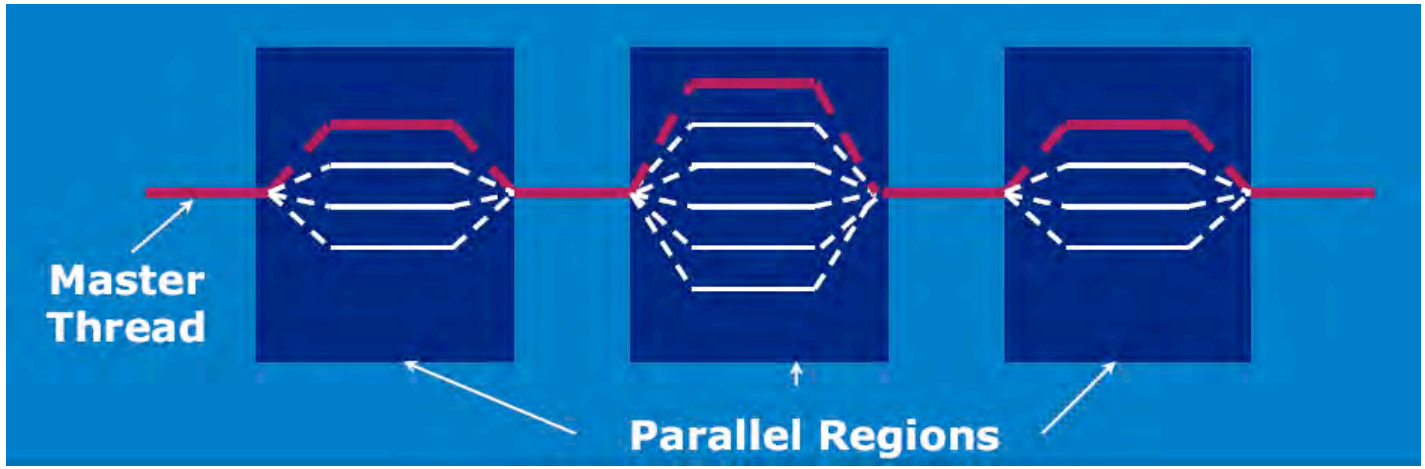
- New parallel programming languages
- Exploit parallelism inherent in high-level specs
- E.g. SPD or StateCharts



New programming models

- E.g. OpenMP for Intel multi-core programming
- Enhancing standard languages (C/C++) with parallel programming pragmas
- Problems:
  - Need OpenMP enabled compiler
  - Tedious manual identification of potential parallelism
  - Not safe (e.g. thread race conditions, critical regions), need thread checker etc.

```
#pragma omp parallel
#pragma omp for
      for (i=0; i<N; i++){
            Do_Work(i);
      }
```



© 2007 R. Leupers

MPSoC compiler concept (MAPS)

```
void JPEGtop(FILE *fp){
  int i, ii, j;
  short DCy=0, DCcb=0, DCcr=0; // DC values
  buf_state state; // for bitstream buffer state
  state.put_bits = 0; state.put_buffer = 0;
  for(i = 0; i < imageSizeYPadding;){
    for(ii = 0; ii < 8; ii ++){
      ReadOneLine(fp, i ++); // row 0: RGB => Y0/Y1,Cb0,Cr0
      ReadOneLine(fp, i ++); // row 1: RGB => Y0/Y1,Cb0,Cr0
      { DownsampleCbCr(i);} // Cb0,Cr0 => Cb,Cr
    }
    { // call the core functions
      int nR = (i - 8 >= imageSizeY); // 2nd row is dummy
      for(j = 0; j < imageSizeX; j += 16){
        int nC = (j + 8 >= imageSizeX); // 2nd col is dummy
        { // process Y components
          BLK8x8(&Y0[j],0,&DCy,&state,0);
          BLK8x8(&Y0[j+8],0,&DCy,&state,nC);
        }
        { // process Y components
          BLK8x8(&Y1[j],0,&DCy,&state,nR);
          BLK8x8(&Y1[j+8],0,&DCy,&state,nC+nR);
        }
        { // process Cb/Cr components
      }
    }
  }
}}
```

Sequential code

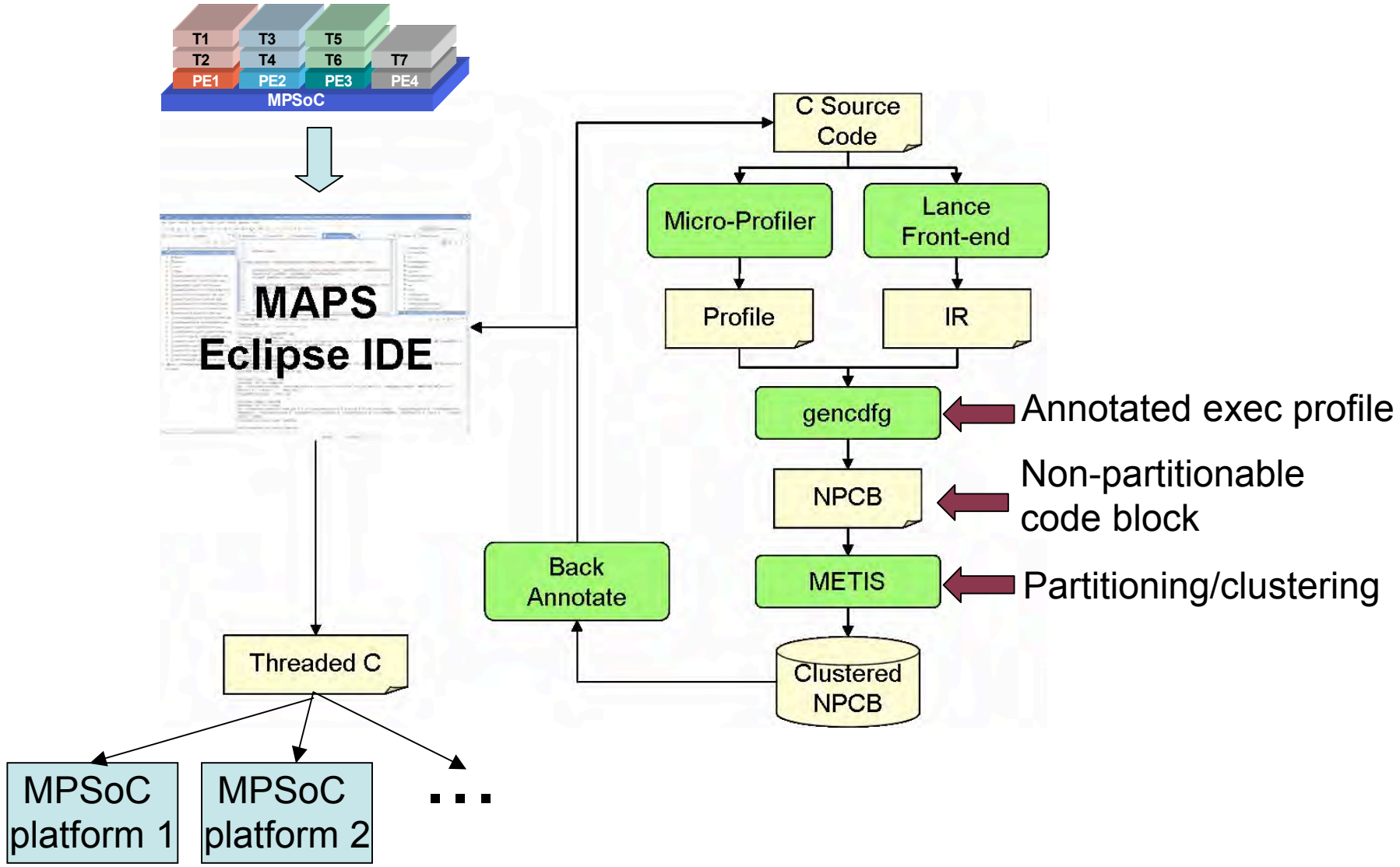

```
void JPEGtop(FILE *fp){
  , ii, j;
   DCy=0, DCcb=0, DCcr=0; // DC values
  tate state; // for bitstream buffer state
  .put_bits = 0; state.put_buffer = 0;
   = 0; i < imageSizeYPadding;){
  (ii = 0; ii < 8; ii ++){
  eadOneLine(fp, i ++); // row 0: RGB => Y0/Y1,Cb0,Cr0
  eadOneLine(fp, i ++); // row 1: RGB => Y0/Y1,Cb0,Cr0
  HREAD(Dsamp) { DownsampleCbCr(i);} // Cb0,Cr0 => Cb,Cr

  EAD(BLKcore) { // call the core functions
  nt nR = (i - 8 >= imageSizeY); // 2nd row is dummy
  or(j = 0; j < imageSizeX; j += 16){
   int nC = (j + 8 >= imageSizeX); // 2nd col is dummy
   THREAD(Y0) { // process Y components
     BLK8x8(&Y0[j],0,&DCy,&state,0);
     BLK8x8(&Y0[j+8],0,&DCy,&state,nC);
   }
   THREAD(Y1) { // process Y components
     BLK8x8(&Y1[j],0,&DCy,&state,nR);
     BLK8x8(&Y1[j+8],0,&DCy,&state,nC+nR);
   }
   THREA
     BLK8
     BLK8
   }
  }
 }
}}
```

Threaded code

- Application code profiling (static/dynamic)
- Semi-automatic code parallelization
- Task to processor mapping based on abstract MPSoC platform model

© 2007 R. Leupers

MAPS compiler flow



TCT backend

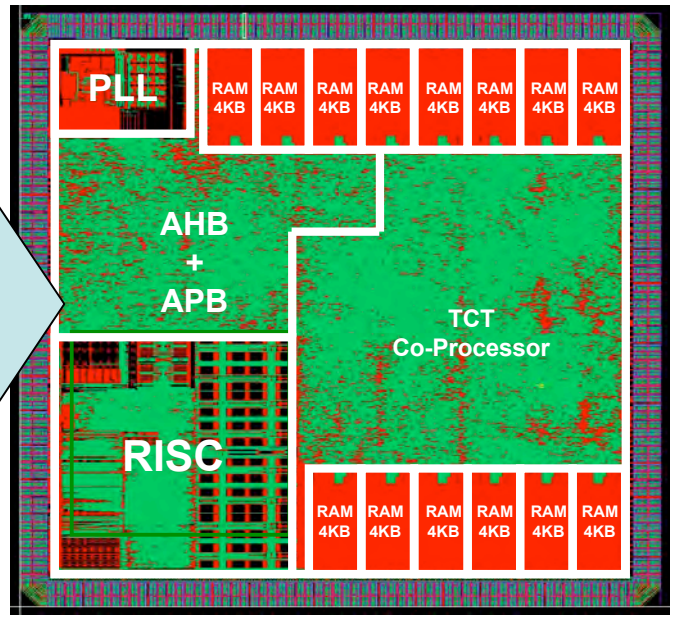
- ISS cooperation with TokyoTech
- First instance of MAPS backend being built for TCT
- TCT has tool support for threaded code mapping + simulators and chip prototype

From MAPS

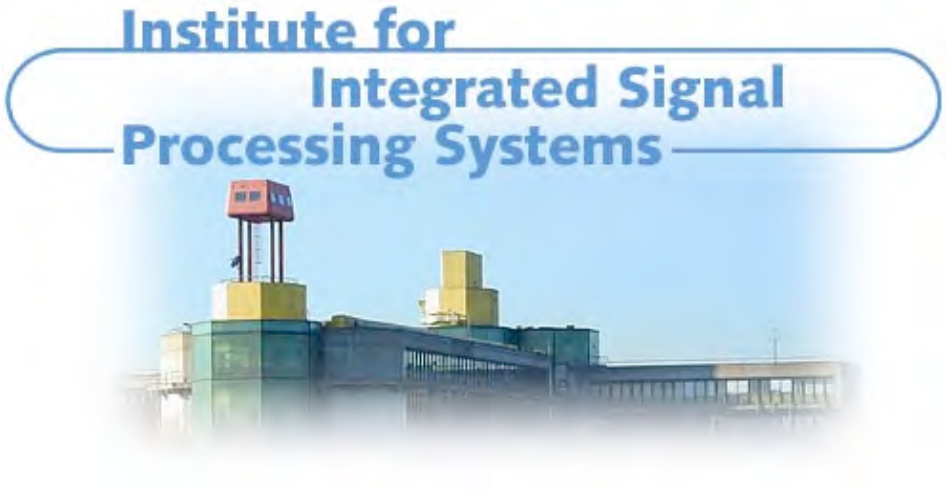
```
void JPEGtop(FILE *fp){
  int i, ii, j;
  short DCy=0, DCcb=0, DCcr=0; // DC values
  buf_state state; // for bitstream buffer state
  state.put_bits = 0; state.put_buffer = 0;
  for(i = 0; i < imageSizeYPadding;){
    for(ii = 0; ii < 8; ii ++){
      ReadOneLine(fp, i ++); // row 0: RGB => Y0/Y1,Cb0,Cr0
      ReadOneLine(fp, i ++); // row 1: RGB => Y0/Y1,Cb0,Cr0
      THREAD(Dsamp) { DownsampleCbCr(i);} // Cb0,Cr0 => Cb,Cr
    }
    THREAD(BLKcore) { // call the core functions
      int nR = (i - 8 >= imageSizeY); // 2nd row is dummy
      for(j = 0; j < imageSizeX; j += 16){
        int nC = (j + 8 >= imageSizeX); // 2nd col is dummy
        THREAD(Y0) { // process Y components
          BLK8x8(&Y0[j],0,&DCy,&state,0);
          BLK8x8(&Y0[j+8],0,&DCy,&state,nC);
        }
        THREAD(Y1) { // process Y components
          BLK8x8(&Y1[j],0,&DCy,&state,nR);
          BLK8x8(&Y1[j+8],0,&DCy,&state,nC+nR);
        }
        THREAD(C) { // process Cb/Cr components
          BLK8x8(&Cb[j>>1
          BLK8x8(&Cr[j>>1
        }
      }
    }
}}
```

Threaded code

mapping



Thank you !



www.iss.rwth-aachen.de

ISS