RWTH AACHEN UNIVERSITY

MPSoC Programming using the MAPS Compiler

Rainer Leupers, Jeronimo Castrillon,

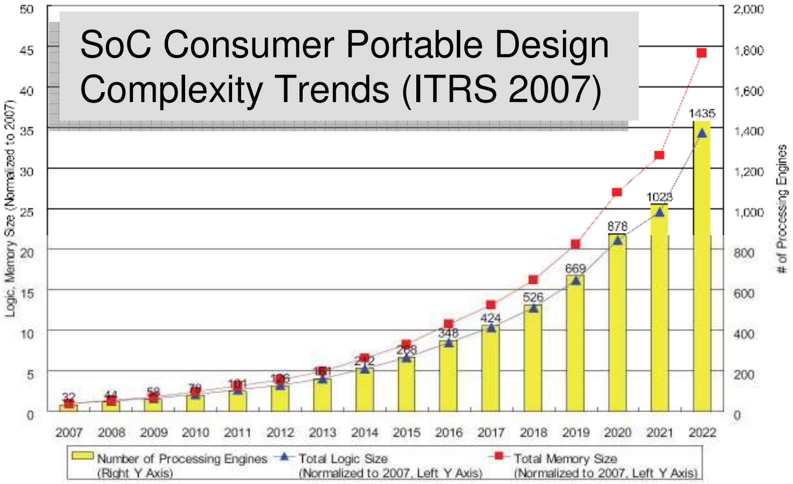
Institute for Integrated Signal Processing Systems
RWTH Aachen University, Germany

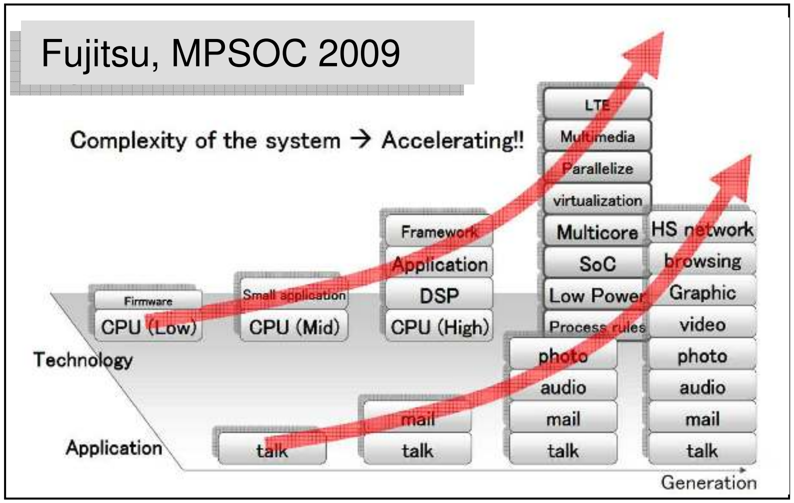
ASP-DAC

Taipei, Jan. 2010

Motivation: MPSoCs and the Productivity Gap

- **Multi-Processor Systems on Chip are a reality**
  - Increased HW and SW complexity


SoC Consumer Portable Design Complexity Trends (ITRS 2007)


Fujitsu, MPSOC 2009

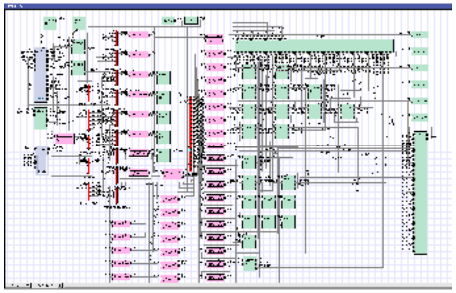
Complexity of the system → Accelerating!!

- **The productivity Gap: Requirements double every 10 months, HW/SW productivity every 2 years (**Ecker, Mueller, Doemer, 2008**)**

➔ Need better support for SW development in the MPSoC era

MAPS: Bridging the Productivity Gap

- **MAPS: MPSoC Application Programming Studio:**
  - Flexible input specification: 85% of embedded programmers use C/C++ (www.eetimes.com)
    - Legacy C-code and partitioning
    - Explicitly parallel C-like programming model (KPN)
  - Abstraction & retargetability:
    - Abstract APIs for early SW design
    - Code generation hides HW dependent SW
  - Functional validation:
    - Abstract simulator (HVP), no processor-specific tool chains involved
  - Mapping & Scheduling frameworks:
    - Manage the huge design space
  - Multiple application of different classes (real-time, best effort)



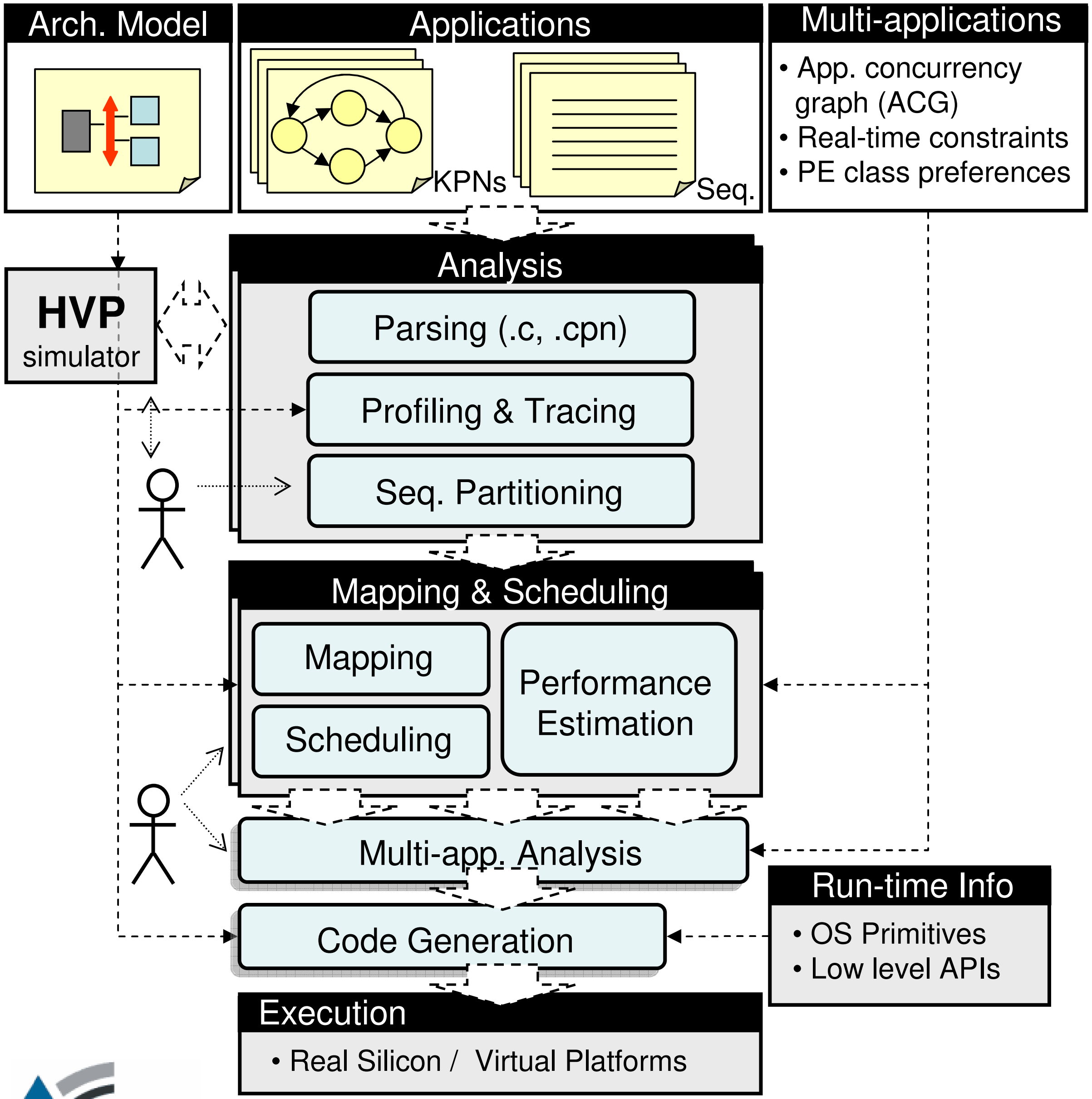
Source: Virtual Platform of Shapes RDT, SSS RWTH Aachen

Source: Chen, NTU, MPSoC 2008

Outline

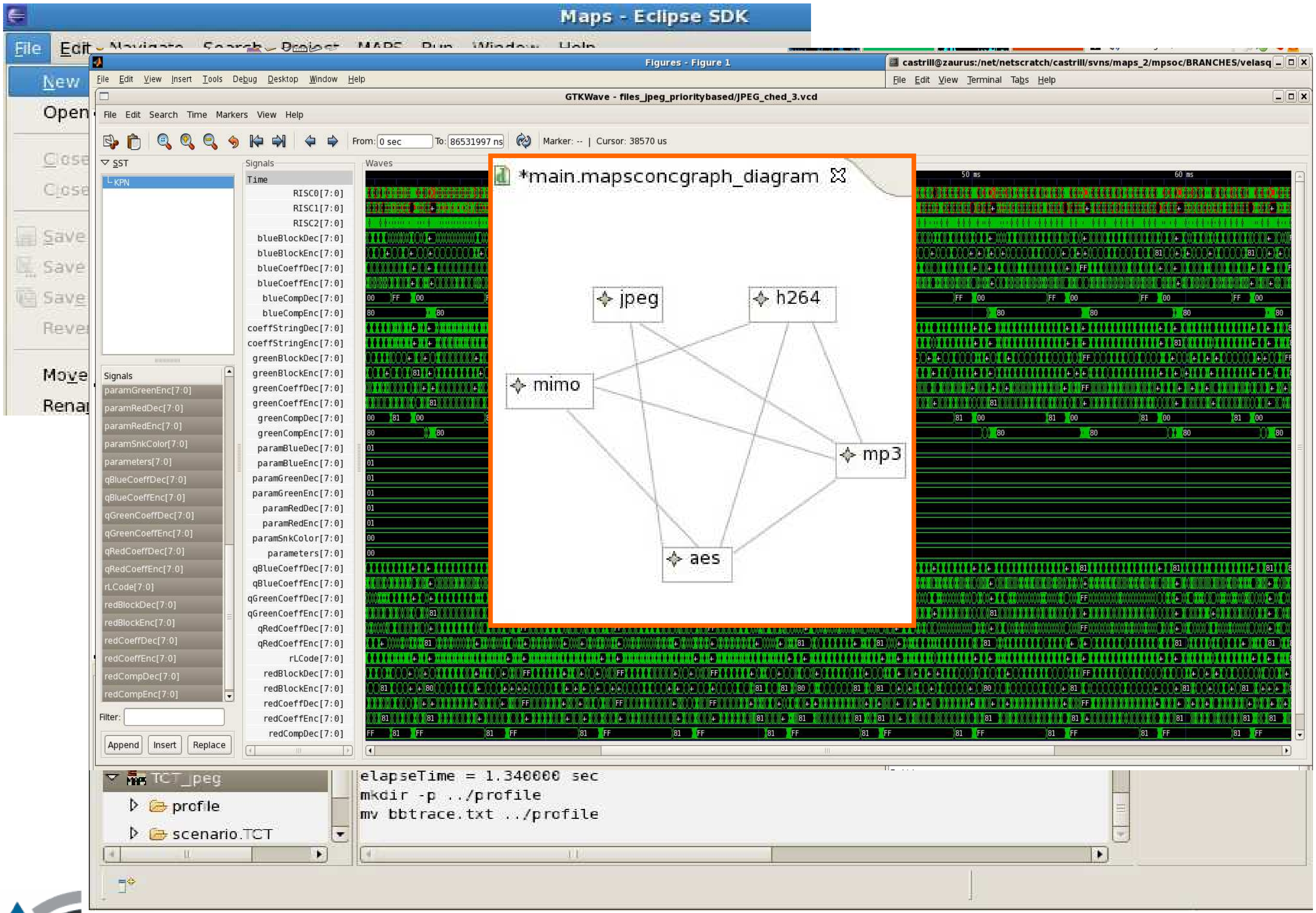
- Motivation
- MAPS Overview
- Sequential and Parallel Flows
- Results
- Conclusions and Outlook

MAPS Flow Overview



- **Architecture model for retargetability**
- **Applications:**
  - C code for legacy
  - Parallel code to leverage a-priori knowledge
- **Analysis Phase:**
  - Profile-driven
  - Interactive
- **Mapping/Scheduling:**
  - Extensible
  - Cost-table driven performance estimation
- **Multiple Application**
  - Interaction through ACG
  - Composition approach
  - Different app. classes

MAPS: Graphical User Interface

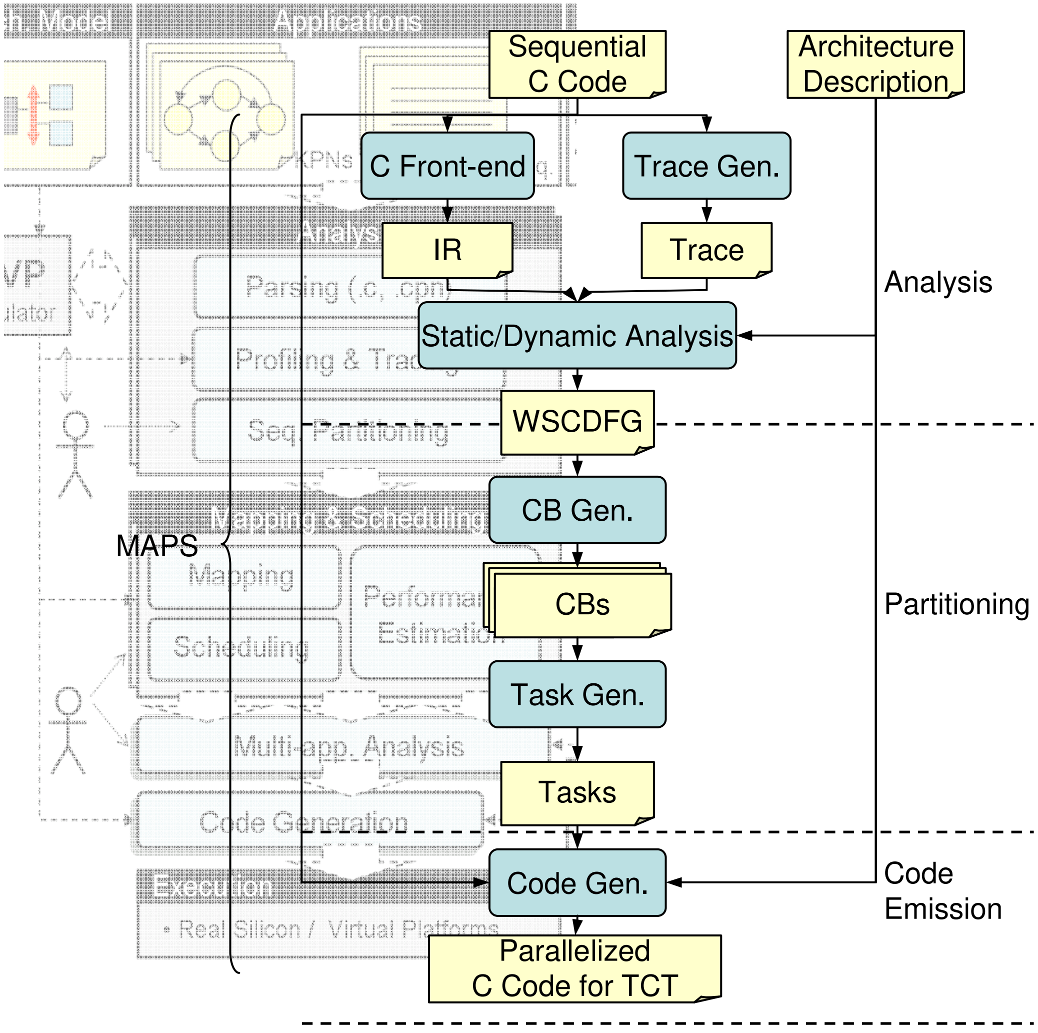


```
elapseTime = 1.340000 sec
mkdir -p ../profile
mv bbtrace.txt ../profile
```

Outline

- Motivation
- MAPS Overview
- ➔ **Sequential and Parallel Flows**
- Results
- Conclusions and Outlook

Sequential Flow: How it started…

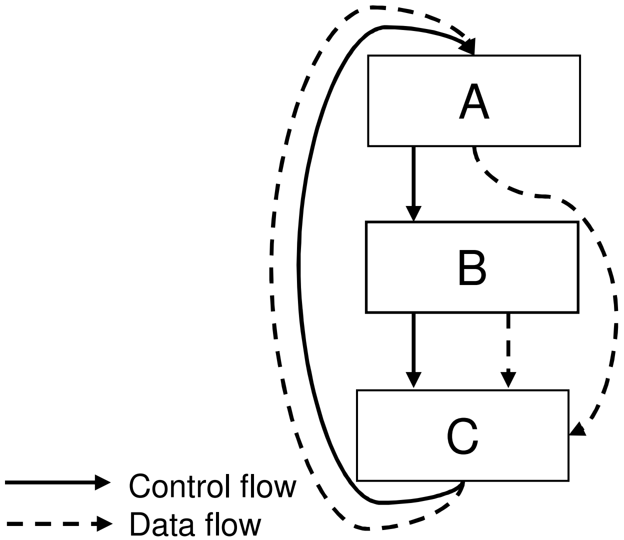


MAPS @ DAC'08

- **Sequential flow as presented in DAC 2008**
- **Key points:**
  1. Analysis phase: Traces for Dynamic Data Flow Analysis
  2. New analysis granularity: "Coupled" blocks as opposed to basic-blocks, functions,…
  3. Performance estimation: annotated 3-address-code IR via cost table
  4. Heuristic for hierarchical code partitioning
- **Simple code generation for TCT platform (TiTech, Tokyo)**
- **Execution on TCT virtual/real platform**

Sequential Flow: Improvements

- **Analyze Strongly Connected Components (SCC): improves parallel efficiency, i.e. less PEs – similar execution time**



time

PE: 1	2	3
A	B	
		C
A	B	
		C
A	B	

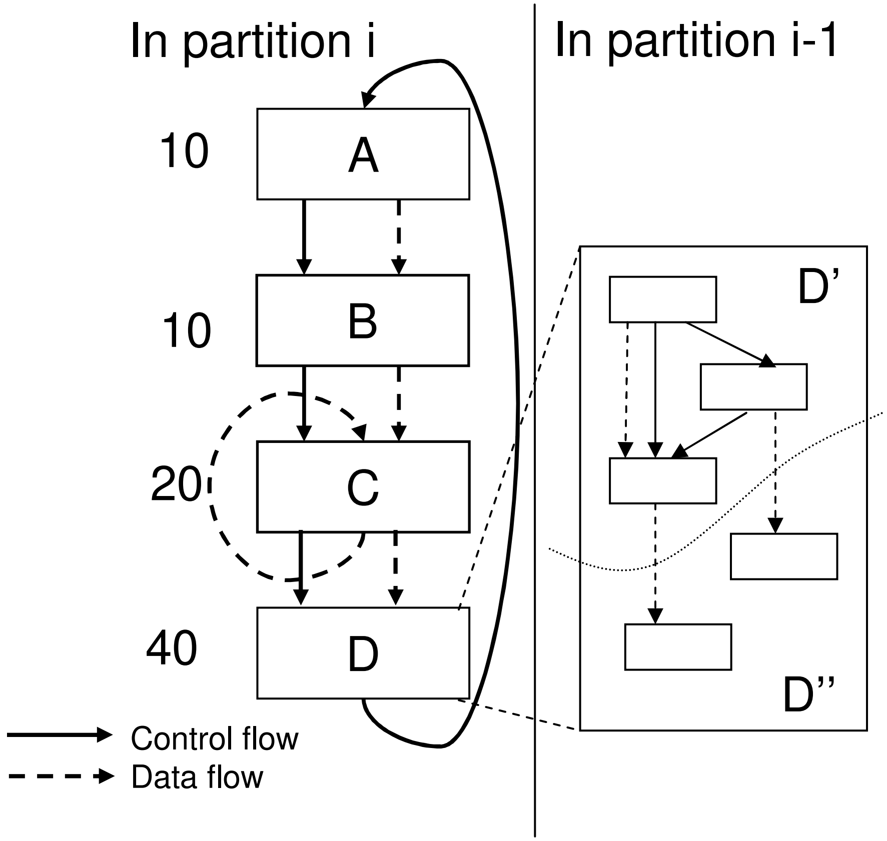
time

PE: 1	2	3
A	B	Spare PE
	C	
A	B	
	C	
A	B	

- SCCs are recognized and a heuristic is used to merge blocks in order to improve the parallel efficiency
- Especial care of nested SCCs

Sequential Flow: Improvements (2)

- **Balance partitions of functions in different locations of the Call Graph**



time

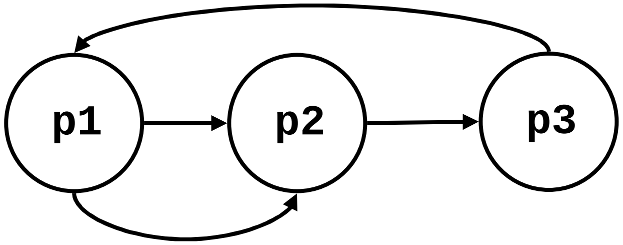
PE: 1	2	3	4
A			
A	B		
…	B …	C	
		C C	D
			D

time

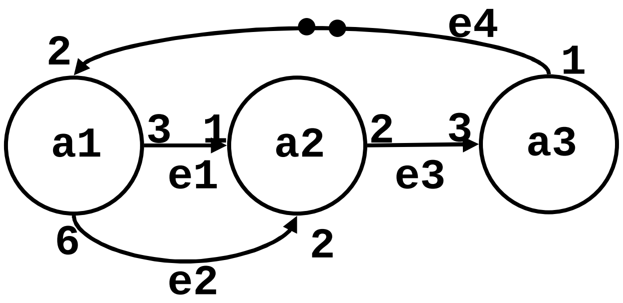
PE: 1	2	3	4
A B			
A B	C		
…	C	D’	
		D’	D’’
			D’’

Parallel Flow: Input Specification

- **Dataflow programming models gain everyday more acceptance… Which to use?**
  - HSDFs, SDFs, MRDFs, CFDF, KPN…



vs.



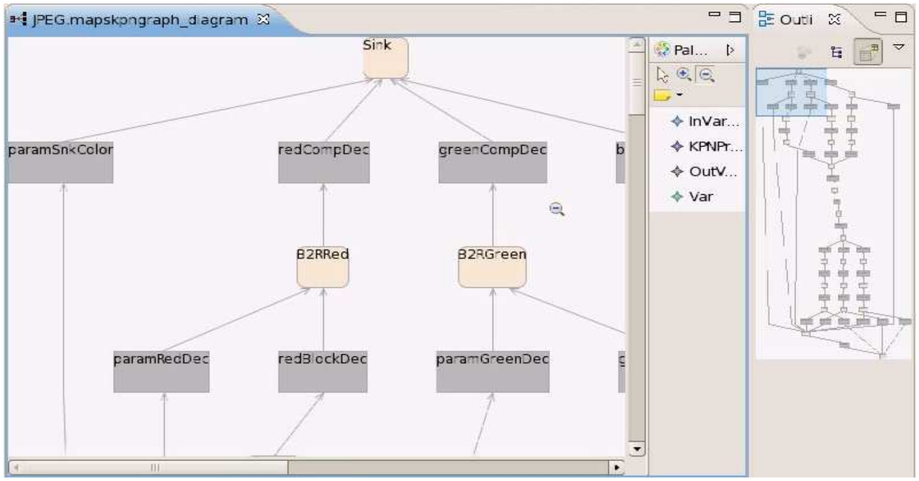
- **MAPS programming model: Based on the Kahn Process Networks (KPN) Model of Computation (MoC)**
  - Better expressiveness compared to other models
  - Simple semantics
  - More difficult to analyze and derive plausible schedules
    - Although comparable when handling multiple applications

Parallel Flow: Input Specification (2)

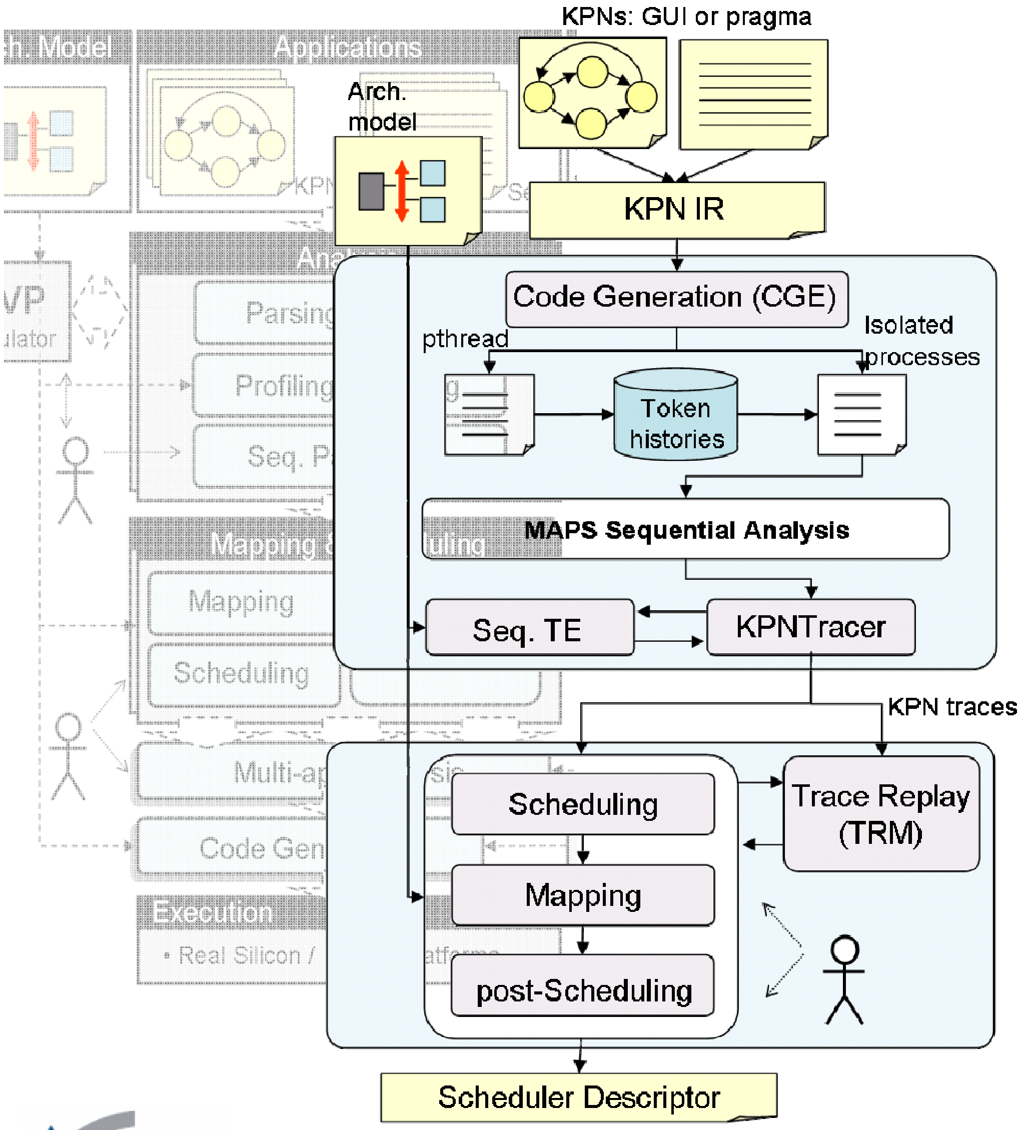
- ***Pragma* extensions to represent KPN applications. Ex. RLE Decoding:**

```
__fifo int A, B;
#pragma maps process rle_dec, in(A), out(B), prefer(risc)
{int cnt, val, i;}      // Local variables to the process
{  // Process body: Repeated for ever
  cnt = A;                   // Reads first token: count
  val = A;                   // Reads second token: value
  for (i = 0; i < cnt; ++i) {
   B = val;                  // Outputs count times val
}}
```

- **GUI equivalent editor/viewer:**



Parallel Flow



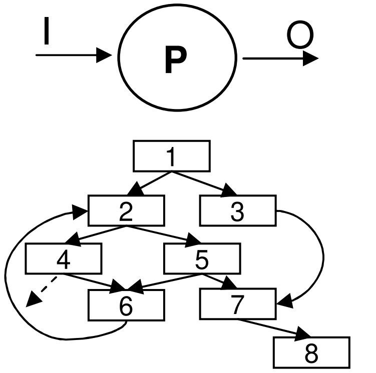
- **Parallel flow, details to appear in DATE Mar. 2010**

- **Key points:**
  1. Intermediate *pthread* code generation for tracing
  2. "*Sequentialized*" processes analyzed by traditional MAPS
  3. KPN tracer generates **KPN traces**
  4. Modular framework for scheduling and mapping: RR, RRWS, priority-based, FIFO,…
  5. TRM allows to compare different schedules

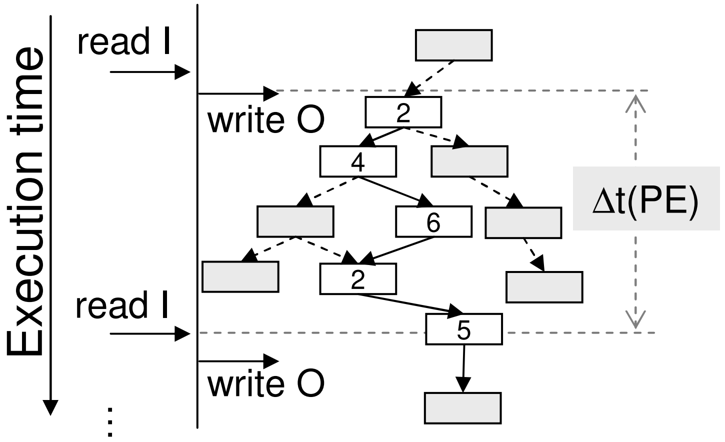
- **The scheduler descriptor can be used to generate code directly**

Parallel Flow: What is a KPN Trace?

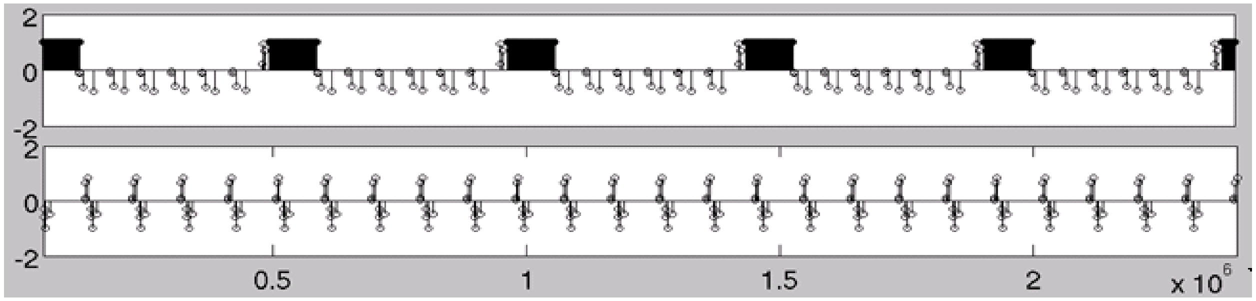
- **A sequential trace is a series of basic blocks**
- **The KPN tracer identifies in which BBs channels were accessed**



Compile time CDFG of P



- **A trace is a sequence of segments, where a segment is a sequence of BBs with a channel access in its last BB**

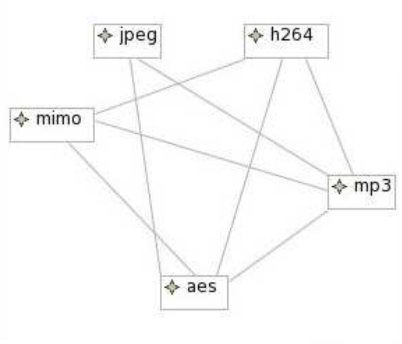


Handling Multiple Applications

- **Applications organized into classes:**
  - Hard/soft real time
  - Best effort

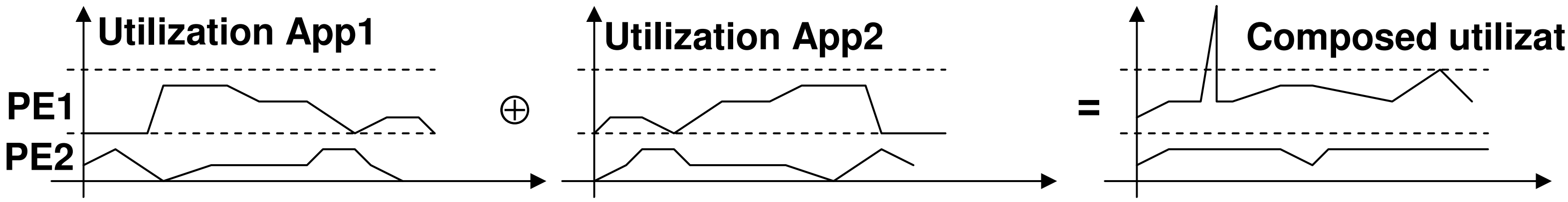
Source: Chen, NTU, MPSoC 2008

- **The Application Concurrency Graph (ACG) serves to describe the interaction among applications**



- **A sub-graph of the ACG represent a *use-case* or *multi-application scenario***
- **Schedules for different applications are computed separately**

- **Use-case analysis via *composition:***



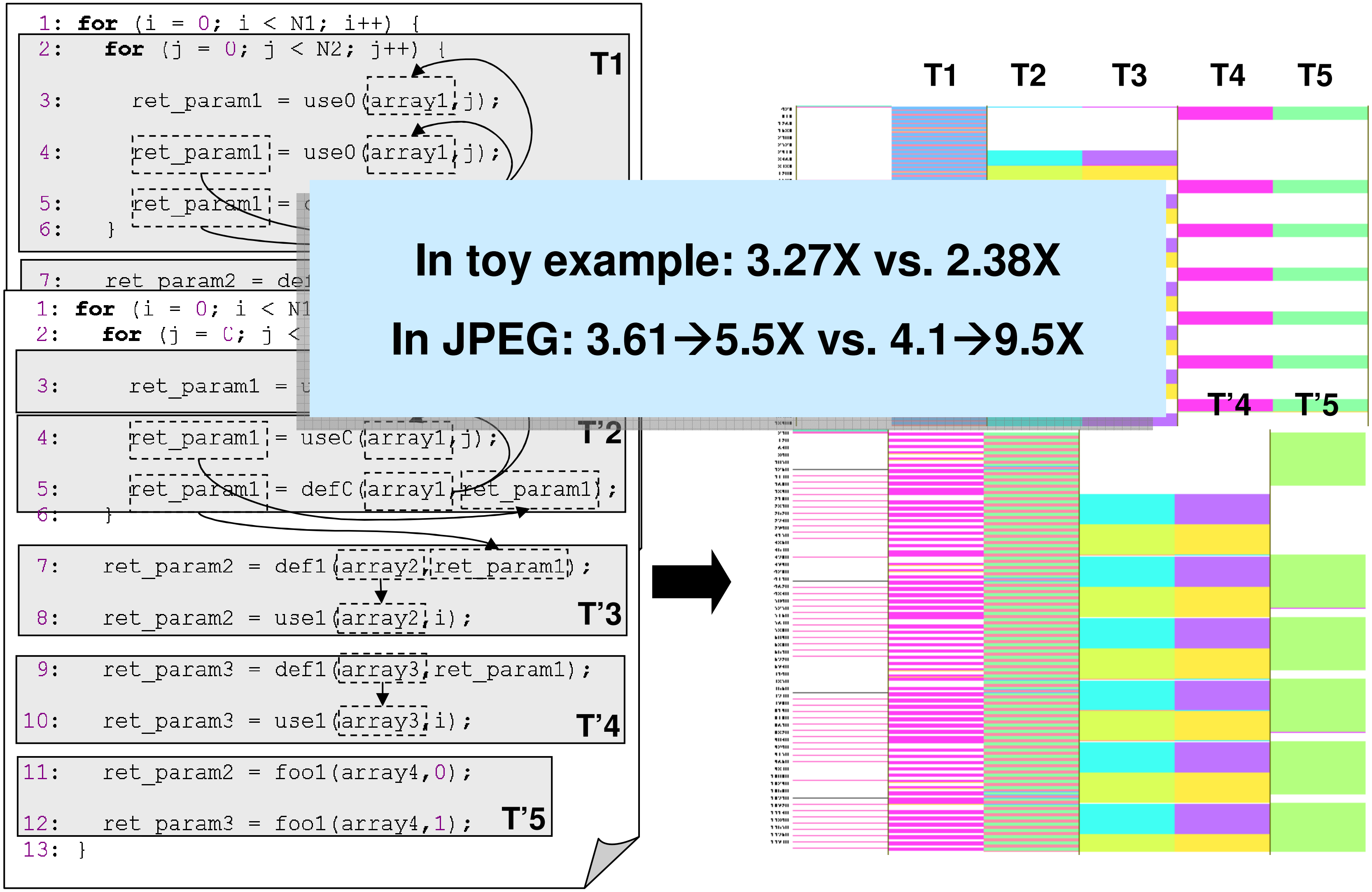
$f(t)$ $g(t)$ $h(t) = f(t) \oplus g(t)$

Outline

- Motivation
- MAPS Overview
- Sequential and Parallel Flows
- ➔ **Results**
- Conclusions and Outlook

Results: Sequential Flow

- **New partitioning passes: a toy example**



```
for (i = 0; i < N1; i++) {
  for (j = 0; j < N2; j++) {
    ret_param1 = use0(array1,j);
    ret_param1 = use0(array1,j);
    ret_param1 = c
  }
  ret_param2 = def
```

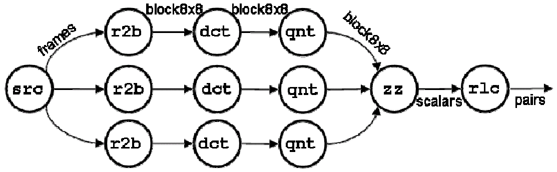
```
for (i = 0; i < N1
  for (j = C; j <
    ret_param1 = u
    ret_param1 = useC(array1,j);
    ret_param1 = defC(array1,ret_param1);
  }
  ret_param2 = def1(array2,ret_param1);
  ret_param2 = use1(array2,i);
  ret_param3 = def1(array3,ret_param1);
  ret_param3 = use1(array3,i);
  ret_param2 = foo1(array4,0);
  ret_param3 = foo1(array4,1);
}
```

In toy example: 3.27X vs. 2.38X

In JPEG: 3.61→5.5X vs. 4.1→9.5X

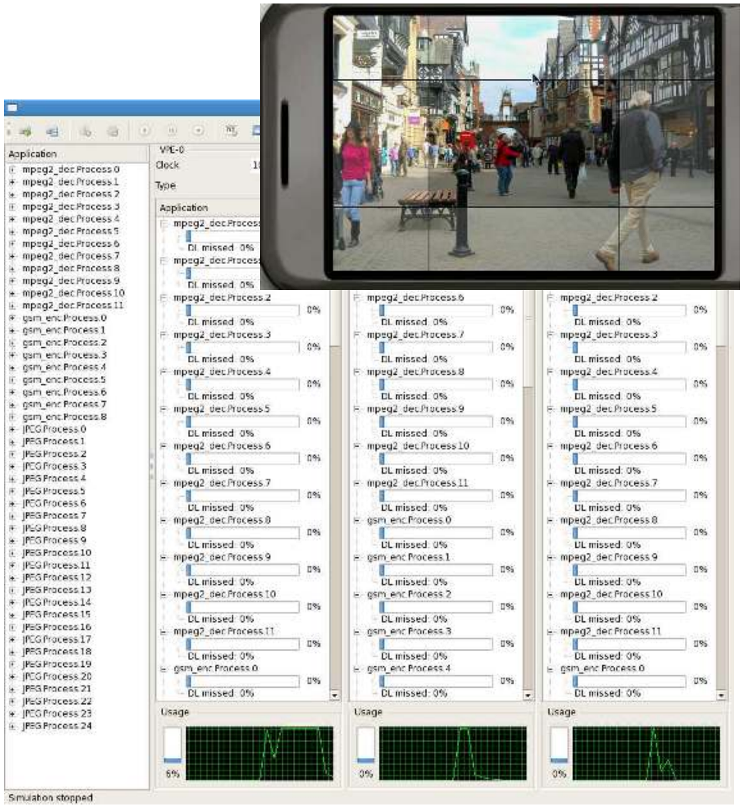
Results: Parallel & Overall Flow

- **The parallel flow has been tested on several real life applications:**
  - MPEG2, JPEG, GSM, MIMO,…



- **MAPS usability fully tested:**
  - Parsing/tracing/profiling
  - Functional validation

- **Later verification on different back-ends**
  - TI-OMAP, TCT, OSIP



Application

mpeg2_dec.Process.0
mpeg2_dec.Process.1
mpeg2_dec.Process.2
mpeg2_dec.Process.3
mpeg2_dec.Process.4
mpeg2_dec.Process.5
mpeg2_dec.Process.6
mpeg2_dec.Process.7
mpeg2_dec.Process.8
mpeg2_dec.Process.9
mpeg2_dec.Process.10
mpeg2_dec.Process.11
gsm_enc.Process.0
gsm_enc.Process.1
gsm_enc.Process.2
gsm_enc.Process.3
gsm_enc.Process.4
gsm_enc.Process.5
gsm_enc.Process.6
gsm_enc.Process.7
gsm_enc.Process.8
JPEG.Process.0
JPEG.Process.1
JPEG.Process.2
JPEG.Process.3
JPEG.Process.4
JPEG.Process.5
JPEG.Process.6
JPEG.Process.7
JPEG.Process.8
JPEG.Process.9
JPEG.Process.10
JPEG.Process.11
JPEG.Process.12
JPEG.Process.13
JPEG.Process.14
JPEG.Process.15
JPEG.Process.16
JPEG.Process.17
JPEG.Process.18
JPEG.Process.19
JPEG.Process.20
JPEG.Process.21
JPEG.Process.22
JPEG.Process.23
JPEG.Process.24

mpeg2_dec.Proces
DL missed 0%
mpeg2_dec.Proces
DL missed 0%
mpeg2_dec.Process.2 0%
DL missed 0%
mpeg2_dec.Process.3 0%
DL missed 0%
mpeg2_dec.Process.4 0%
DL missed 0%
mpeg2_dec.Process.5 0%
DL missed 0%
mpeg2_dec.Process.6 0%
DL missed 0%
mpeg2_dec.Process.7 0%
DL missed 0%
mpeg2_dec.Process.8 0%
DL missed 0%
mpeg2_dec.Process.9 0%
DL missed 0%
mpeg2_dec.Process.10 0%
DL missed 0%
mpeg2_dec.Process.11 0%
DL missed 0%
gsm_enc.Process.0 0%
DL missed 0%

Usage 6%

mpeg2_dec.Process.6 0%
DL missed 0%
mpeg2_dec.Process.7 0%
DL missed 0%
mpeg2_dec.Process.8 0%
DL missed 0%
mpeg2_dec.Process.9 0%
DL missed 0%
mpeg2_dec.Process.10 0%
DL missed 0%
mpeg2_dec.Process.11 0%
DL missed 0%
gsm_enc.Process.0 0%
DL missed 0%
gsm_enc.Process.1 0%
DL missed 0%
gsm_enc.Process.2 0%
DL missed 0%
gsm_enc.Process.3 0%
DL missed 0%
gsm_enc.Process.4 0%
DL missed 0%

Usage 0%

mpeg2_dec.Process.2 0%
DL missed 0%
mpeg2_dec.Process.3 0%
DL missed 0%
mpeg2_dec.Process.4 0%
DL missed 0%
mpeg2_dec.Process.5 0%
DL missed 0%
mpeg2_dec.Process.6 0%
DL missed 0%
mpeg2_dec.Process.7 0%
DL missed 0%
mpeg2_dec.Process.8 0%
DL missed 0%
mpeg2_dec.Process.9 0%
DL missed 0%
mpeg2_dec.Process.10 0%
DL missed 0%
mpeg2_dec.Process.11 0%
DL missed 0%
gsm_enc.Process.0 0%
DL missed 0%

Usage 0%

Simulation stopped

Source:
www.ti.com

Outline

- Motivation
- MAPS Overview
- Sequential and Parallel Flows
- Results

➔ **Conclusions and Outlook**

Conclusions and Outlook

- **MAPS – A fairly complete tool set for MPSoC programming was presented:**
  - Sequential (C) & parallel (KPN) input specification
  - Abstraction: functional simulation, APIs
  - Mapping & scheduling of single and multiple applications to heterogeneous MPSoCs

  … in a user friendly Eclipsed-based GUI

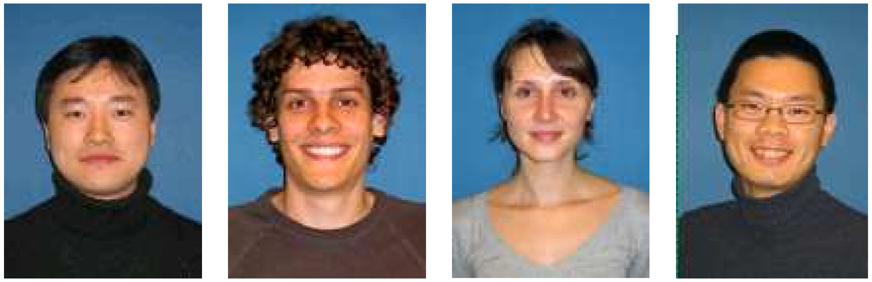
- **Current & future work in MAPS**
  - C extensions instead of *pragmas*, aka: **CPN**
  - Compiler development: CLANG, LLVM
  - Better performance estimation techniques: **TotalProf**
  - Improving mapping and scheduling heuristics
  - Research on *composability* for KPNs

Thank You!
Questions??

Acknowledgments:

This work has been supported by the UMIC (Ultra High-Speed Mobile Information and Communication) research centre. www.umic.rwth-aachen.de

The team:



maps@iss.rwth-aachen.de