ARTIST2 Summer School 2008 in Europe

Autrans (near Grenoble), France

September 8-12, 2008

Mapping C code on MPSoC for Nomadic Embedded Systems

Lecturer: Diederik Verkest

Group Science Director

IMEC, Leuven, Belgium

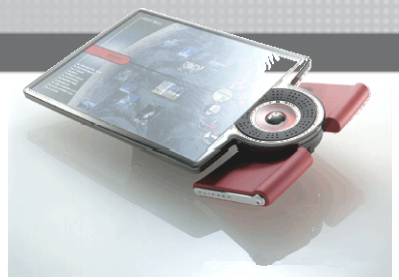
Professor at KU-Leuven

Professor at Vrije-Universiteit Brussel

Outline

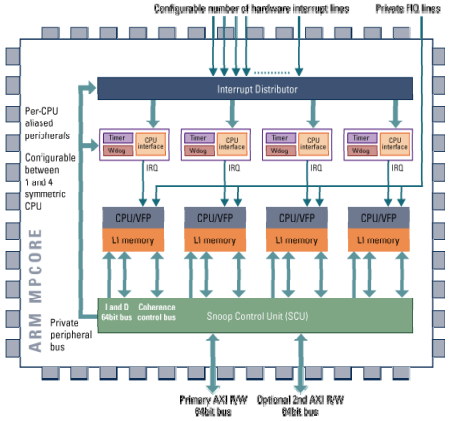
- Context
  - Nomadic embedded system
  - MPSoC mapping challenges
- IMEC research
  - Memory management for MPSoC
  - Parallelizing code for MPSoC
- Conclusions and glimpse of the future

What are “Nomadic Embedded Systems”?

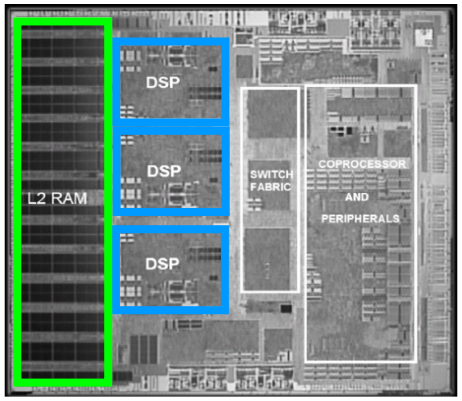




- Characteristics of nomadic systems
  - Real-time constraints
  - Power and energy constraints (battery)
  - Design time constraints (time to market)
  - Cost
  - Flexibility and performance
- Embedded core for flexibility and multi-core for performance

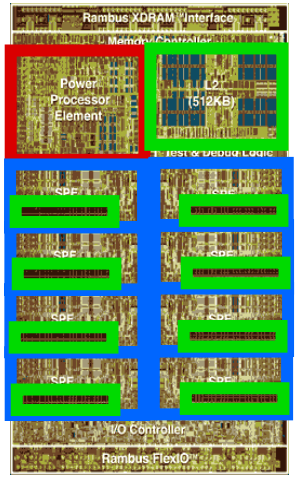
ARM



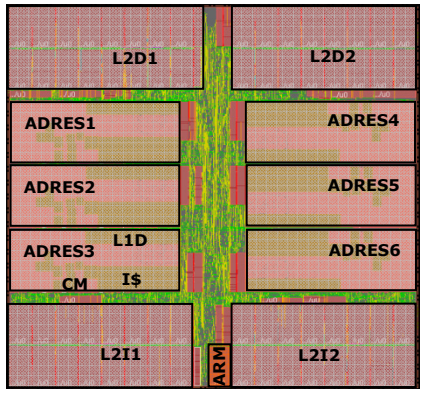
Texas Instruments

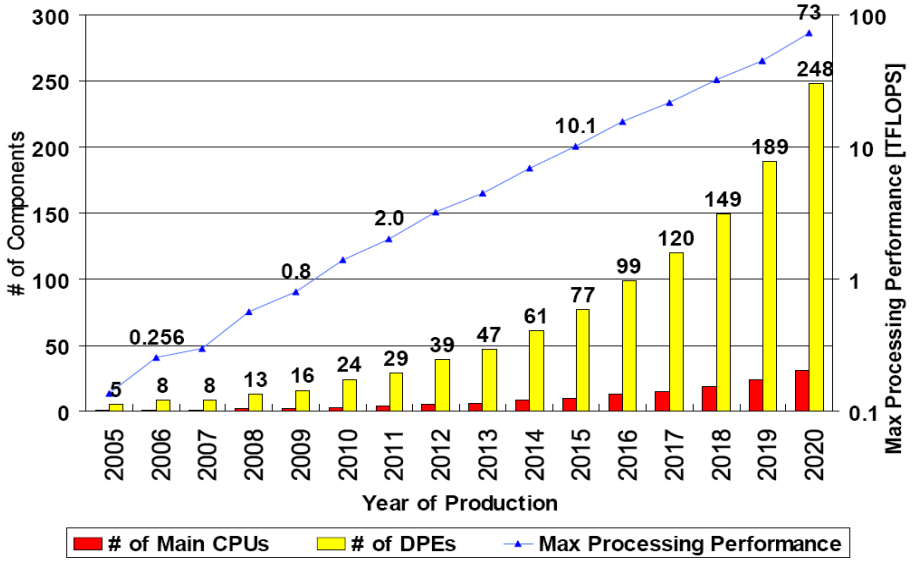


IBM Cell



IMEC





The MPSoC hardware is there

The key question:

"How to efficiently program it?"

Platform Evolution – Embedded System

Single core, sequential programs

- Designer uses sequential C code and assumes a single (shared) memory space
- Platform consist of a single processor with a cache and a main memory, interconnected by a bus.
- Good match between programming model and computing platform
- Designer focuses on algorithmic development and code optimizations

Multi core, parallel programs

- Memory access will not scale up
- Communication and synchronization becomes problematic
- Debugging is a nightmare
- Existing programming model breaks
- Parallelization, componentization and composability
- Keep scalability across platform generations (#processors/amount of memory/available bandwidth)
- Multi-application predictability

MPSoC hardware observations

- Caches are a main source of inefficiency and unpredictability
- Cache-coherency doesn't scale in multi-core platforms
- Central bus architectures or multi-bus architectures don't scale
- Alternatives exist...
  - Scratchpad instead of cache
  - Network-on-chip with service guarantees
- But add to the burden of the software developer

IMEC research goals

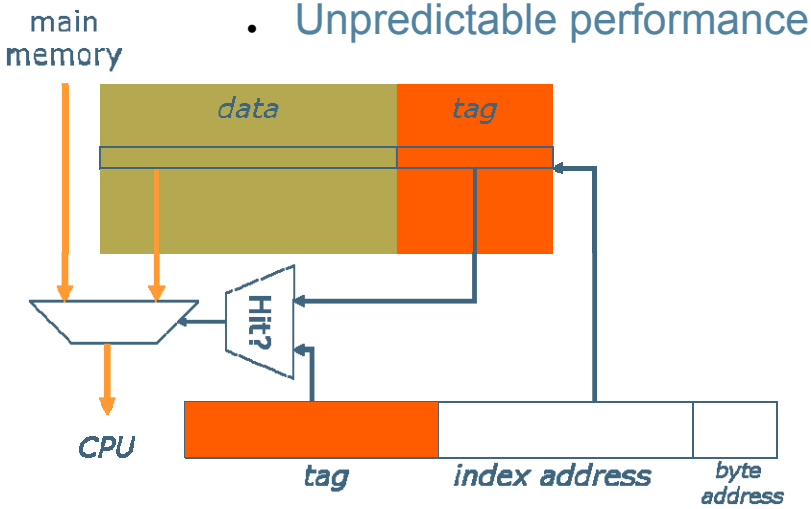
- Compiler-like tools to assist software developers
  - Sequential C code in, correct by construction (parallel) C code out
    - Taking care of scratchpad management
    - Assisting with parallelization of code
  - Embedded software developer in charge
- Run-time support to manage multiple applications in predictable fashion

IMEC research goals

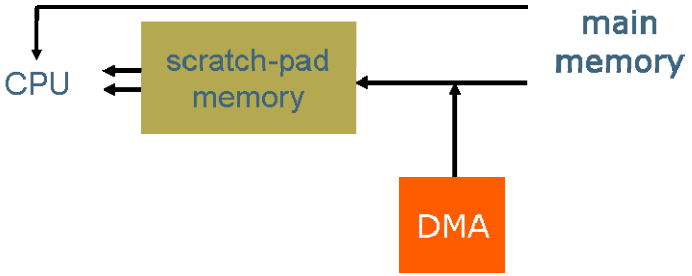
- Compiler-like tools to assist software developers
  - Sequential C code in, correct by construction (parallel) C code out
    - ***Taking care of scratchpad management***
    - Assisting with parallelization of code
  - Embedded software developer in charge
- Run-time support to manage multiple applications in predictable fashion

Cache vs. scratchpad

- Cache
  - Hardware
    - SRAM memory for data
    - Additional hardware
  - Use
    - No programmer effort required
    - Unpredictable performance

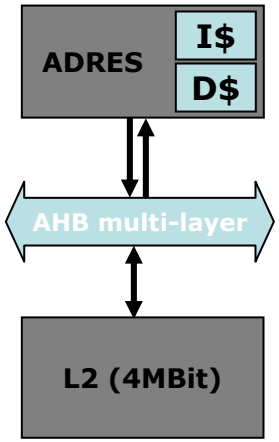


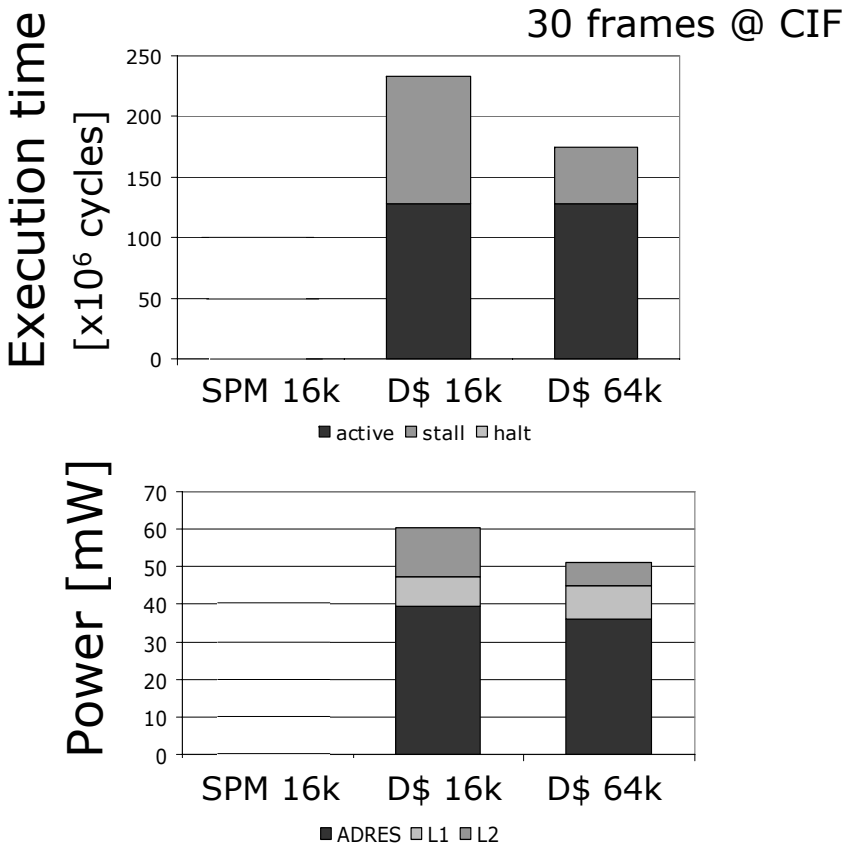
- Scratchpad
  - Hardware
    - SRAM memory for data
    - DMA
  - Use
    - Programmer in control: tedious
    - Predictable
    - More efficient



Cache vs. scratchpad - results

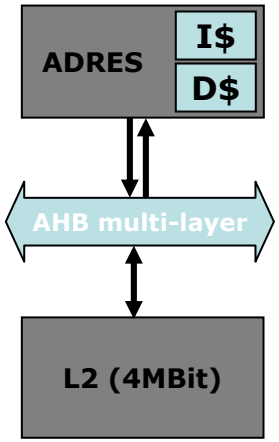
- MPEG-4 p2 SP encoder (±8950 lines of C code)

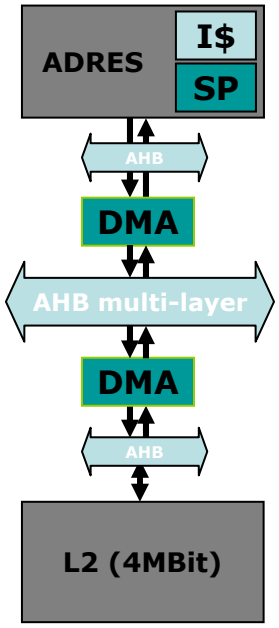


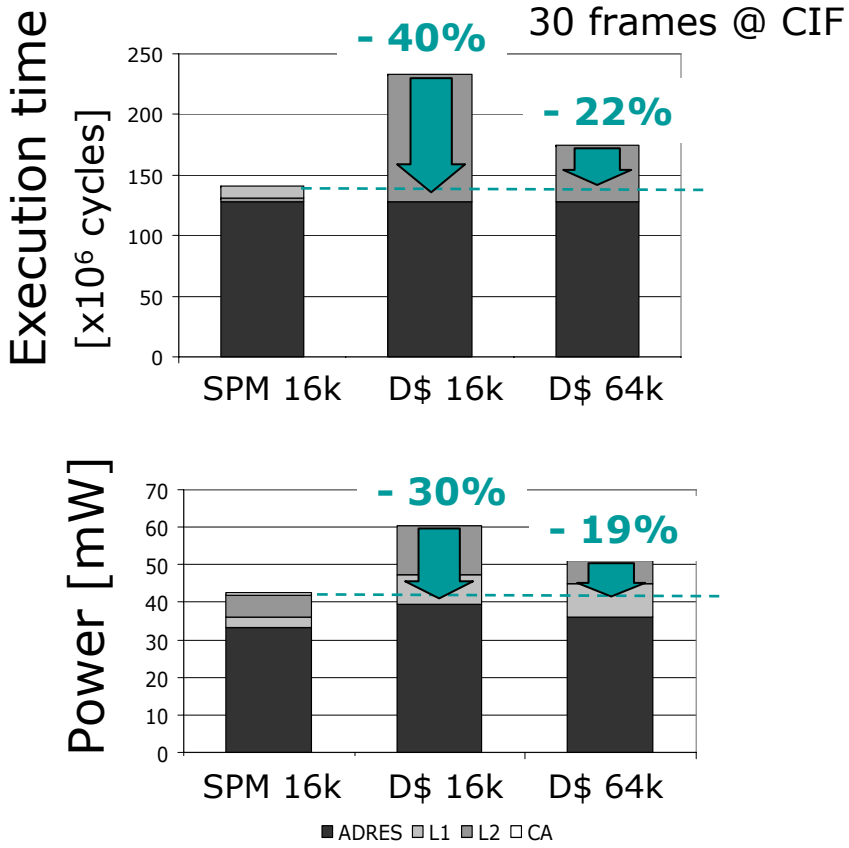


Cache vs. scratchpad - results

- MPEG-4 p2 SP encoder (±8950 lines of C code)

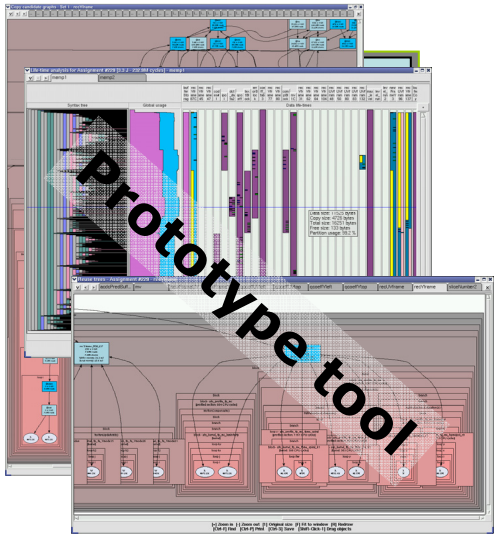




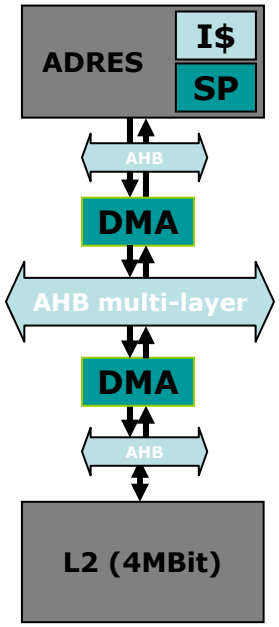


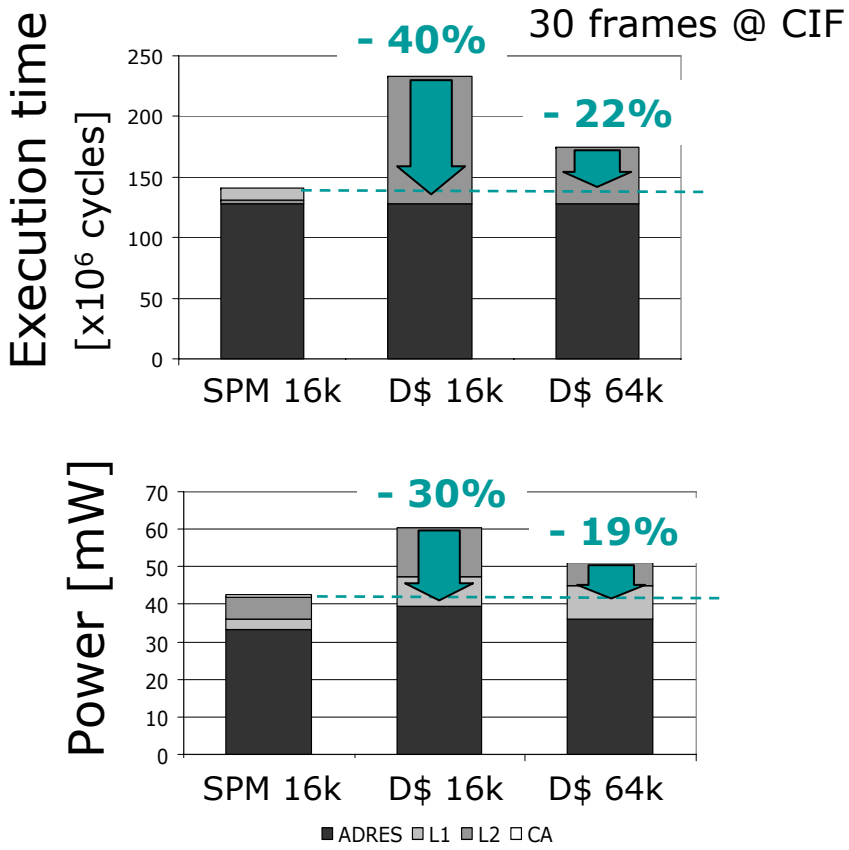
Cache vs. scratchpad - results

- MPEG-4 p2 SP encoder (±8950 lines of C code)



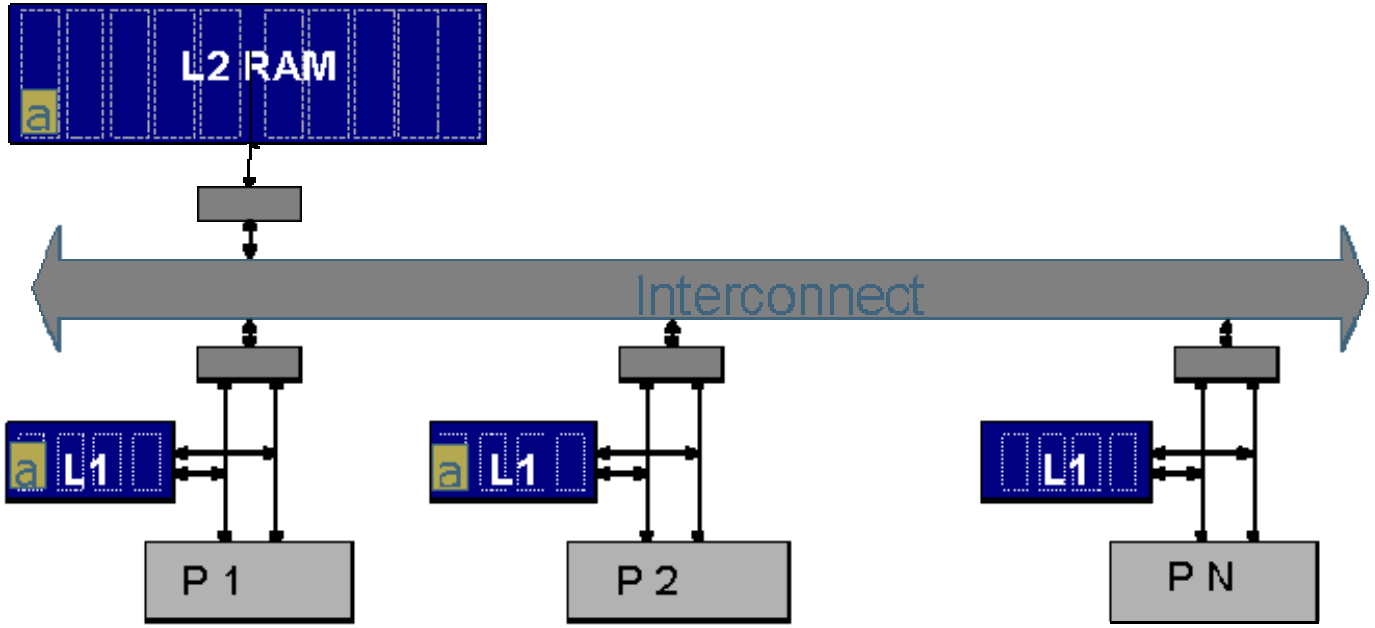
±9250 lines of C code
correct by construction





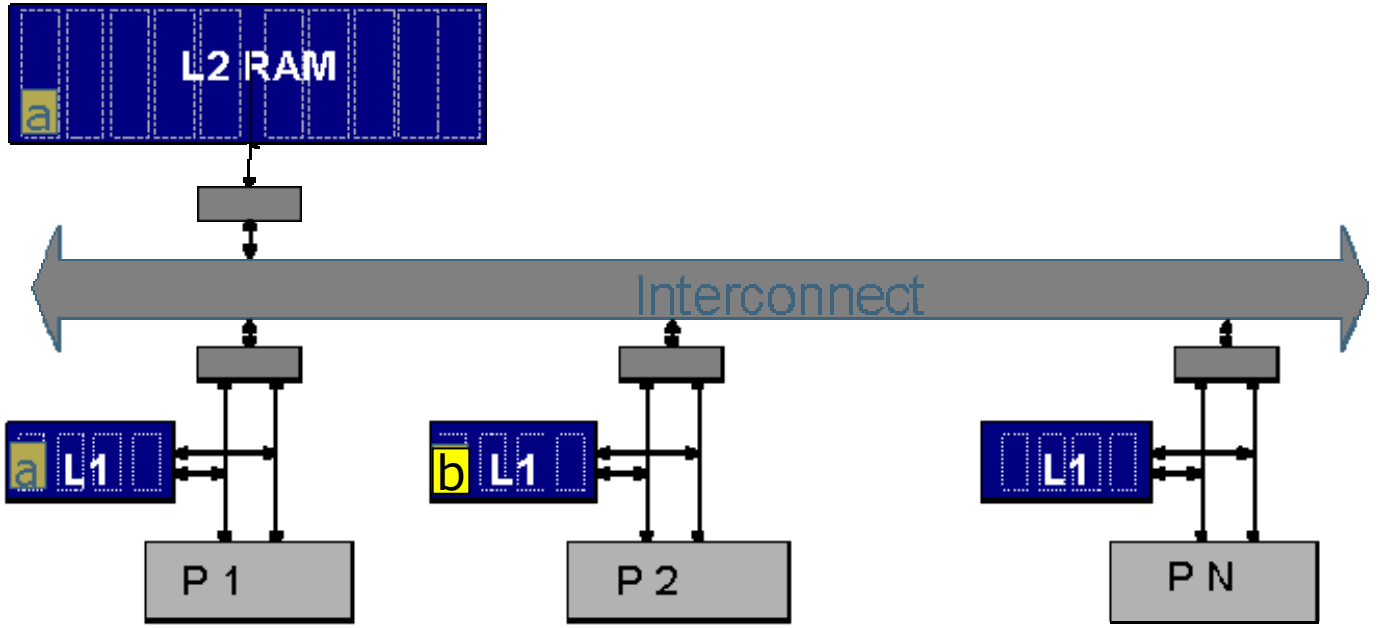
Caches and scratchpads in MPSoC

- Cache coherency



Caches and scratchpads in MPSoC

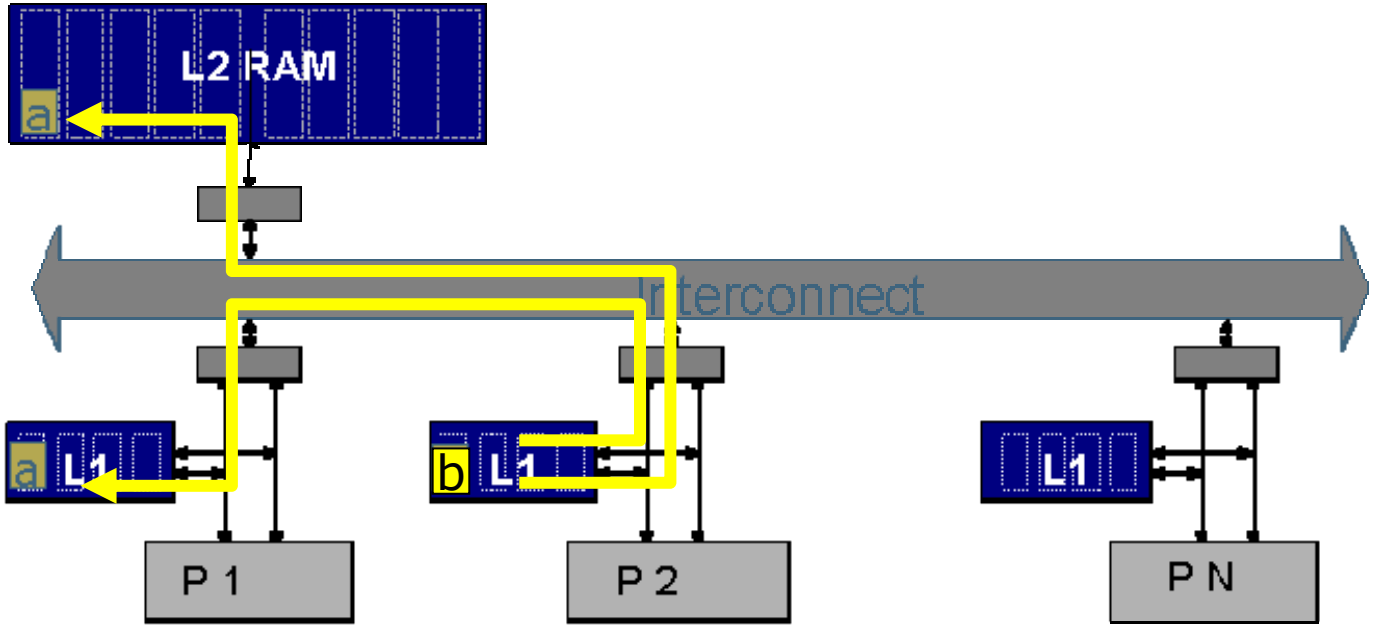
- Cache coherency



"b" overwrites "a" but only in local cache

Caches and scratchpads in MPSoC

- Cache coherency



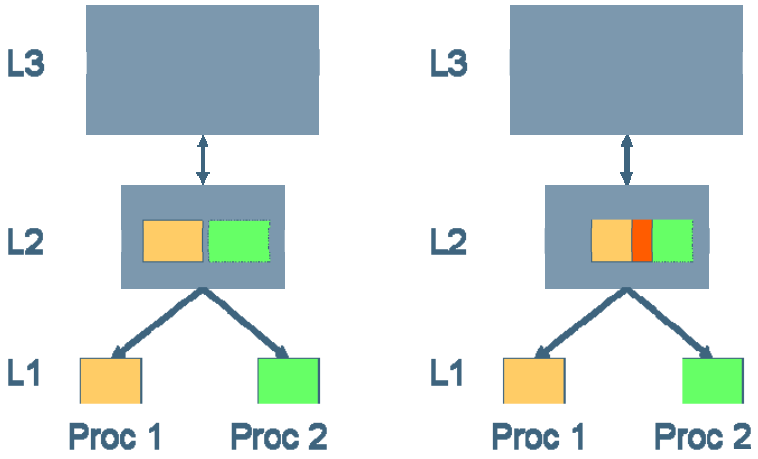
"b" has to be broadcasted to or invalidated in all other memory locations

Caches and scratchpads in MPSoC

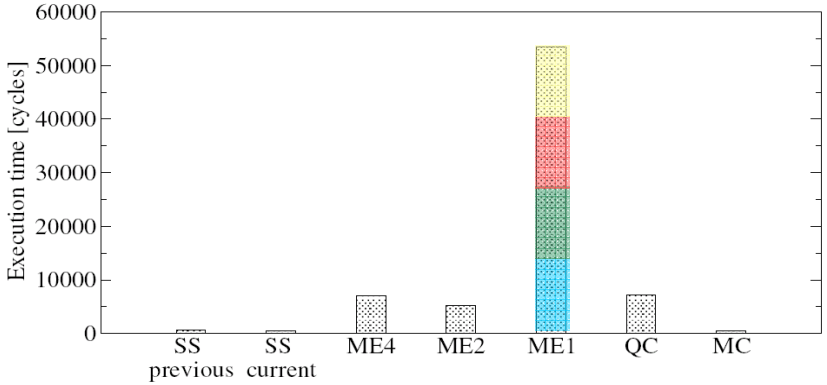
- Caches
  - Cache coherency issue
  - Cache coherency protocols
    - Bus Snooping
      - Energy waste
      - Not efficient when number of processor increases
    - Directory based
      - Better scaling
      - Hardware overhead
- Scratchpads

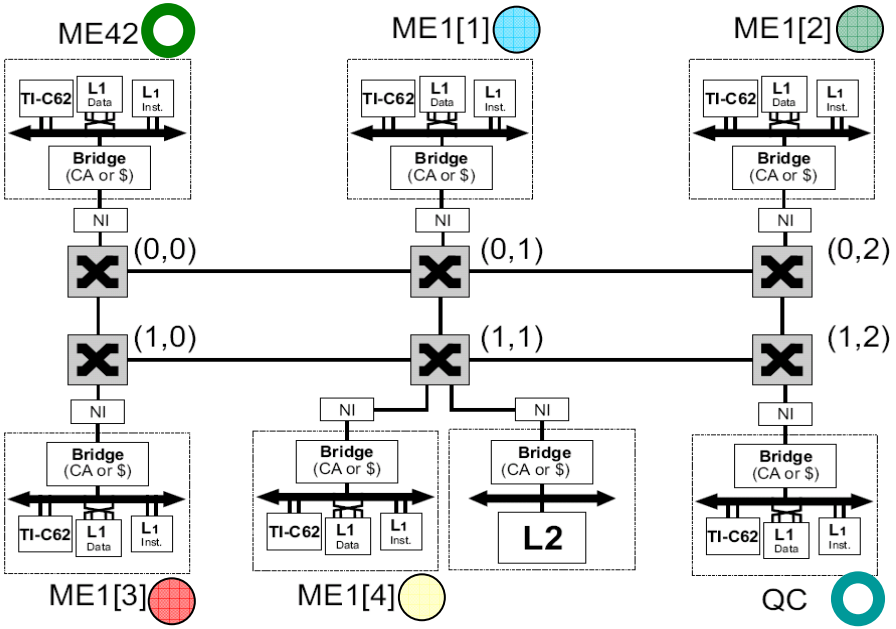
Caches and scratchpads in MPSoC

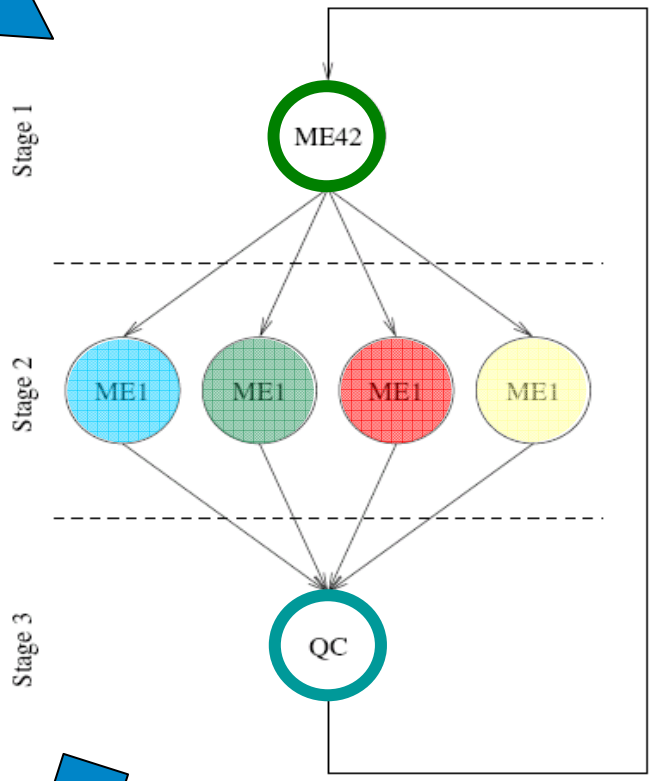
- Caches
  - Cache coherency issue
  - Cache coherency protocols
    - Bus Snooping
      - Energy waste
      - Not efficient when number of processor increases
    - Directory based
      - Better scaling
      - Hardware overhead
- Scratchpads
  - Efficient
  - Programmer in control
    - Synchronising accesses to shared data
    - Scheduling and synchronizing block transfers



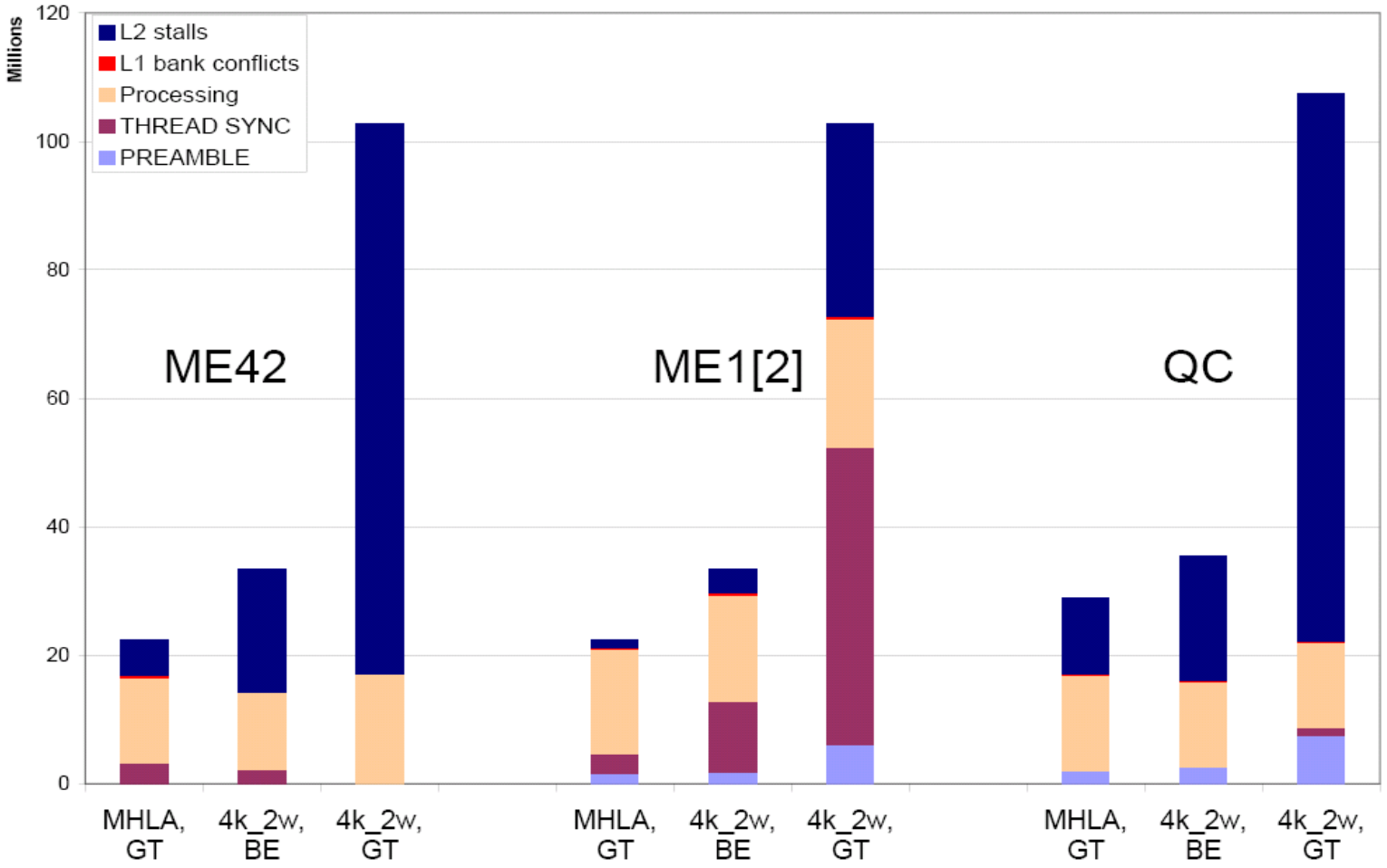
Cache vs. scratchpad in MPSoC - results







Cache vs. scratchpad in MPSoC - results



IMEC research goals

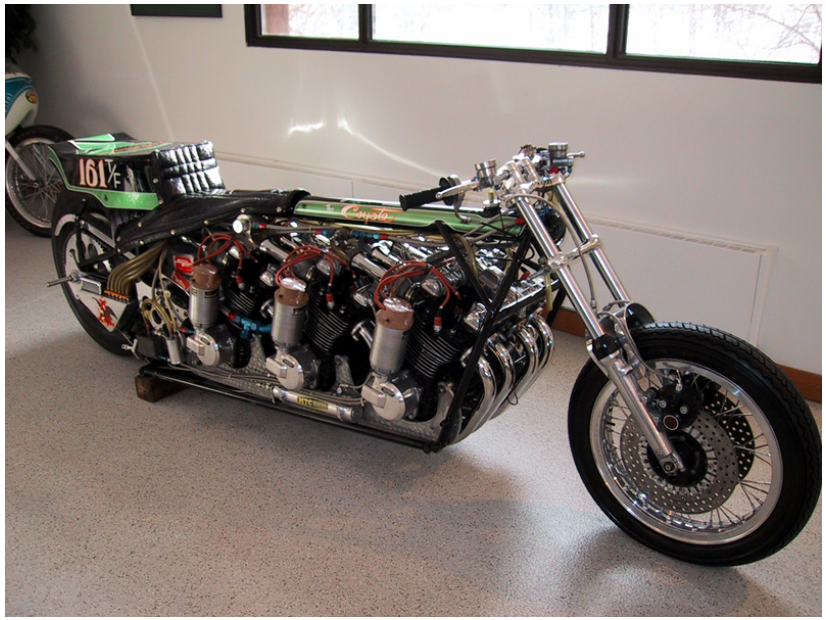
- Compiler-like tools to assist software developers
  - Sequential C code in, correct by construction (parallel) C code out
    - Taking care of scratchpad management
    - Assisting with parallelization of code
  - Embedded software developer in charge
- Run-time support to manage multiple applications in predictable fashion

IMEC research goals

- Compiler-like tools to assist software developers
  - Sequential C code in, correct by construction (parallel) C code out
    - Taking care of scratchpad management
    - ***Assisting with parallelization of code***
  - Embedded software developer in charge
- Run-time support to manage multiple applications in predictable fashion

Parallelization of code

- The challenge
  - N processors, N x speed-up?



Parallelization of code

- The challenge
  - N processors, N x speed-up?
- Current practice
  - DIY
  - Typical result
    - Long nights of debugging
    - Not the expected speed-up

Parallelization of code

- Challenges
  - Matching parallelism to available hardware
  - Parallelism limited by dependencies
    - Data, control
    - Resources
  - Dependencies lead to sequential execution
    - Some dependencies can be removed …
  - Dependencies implies communication and synchronization
    - Communication and synchronization is tricky

Order of memory accesses?

CPU 1 Executes:

```
A = 41; L = 1;
…
A = 42;
L = 0;
```

CPU 2 Executes:

```
…
while (L != 0)
   wait;
B = A;
```

Intended behavior: *B should equal 42*
→ variable L should protect access to A

Is this behaviour guaranteed?

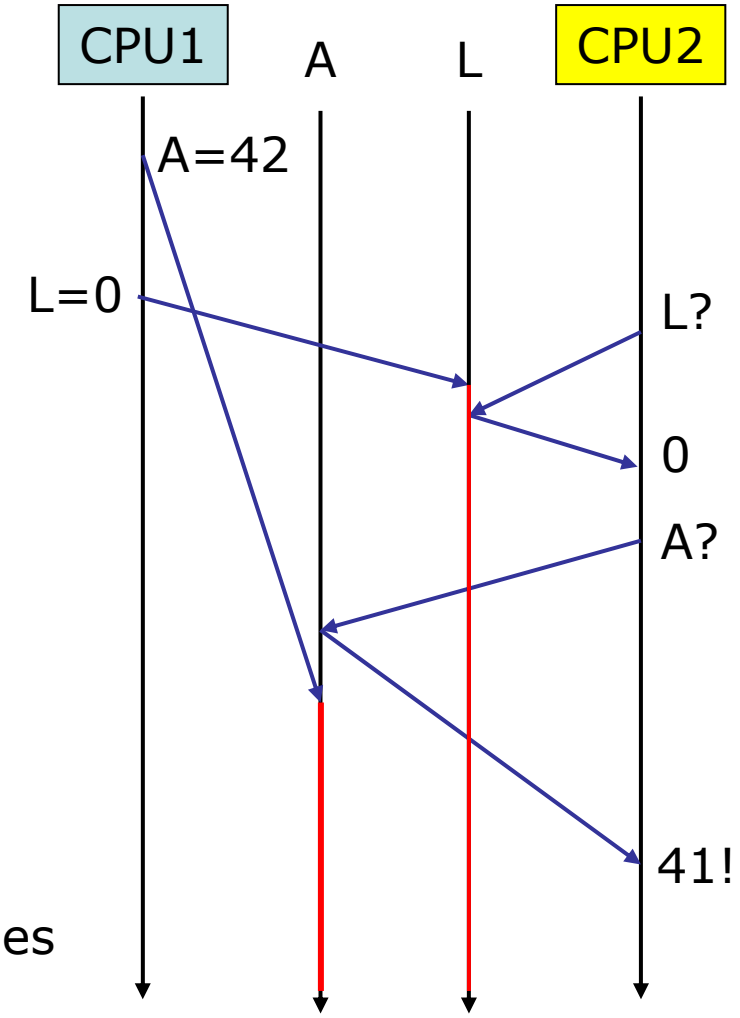
Order of memory accesses?

CPU 1 Executes:

```
A = 41; L = 1;
…
A = 42;
L = 0;
```

CPU 2 Executes:

```
…
while (L != 0)
   wait;
B = A;
```



Intended behavior: *B should equal 42*
→ variable L should protect access to A

Is this behaviour guaranteed?
NO: write of L may overtake write to A

Different processors have an
inconsistent view of the order of accesses

Parallelization of code

- Current and emerging practice
  - Using a multiprocessor OS/RTOS
  - Pre-parallelized libraries
  - Standards such as MPI and OpenMP
  - OpenMP (http://www.compunity.org/)
    - API that supports shared memory multi-processing
    - User specifies how sequential code should be parallelized, compiler generates parallel code using the API
    - User is responsible for correctness of requested parallelization.
    - Only simple parallelization schemes can be applied correctly on the code
    - Complex parallelization requires user to rewrite the code to obtain correct results

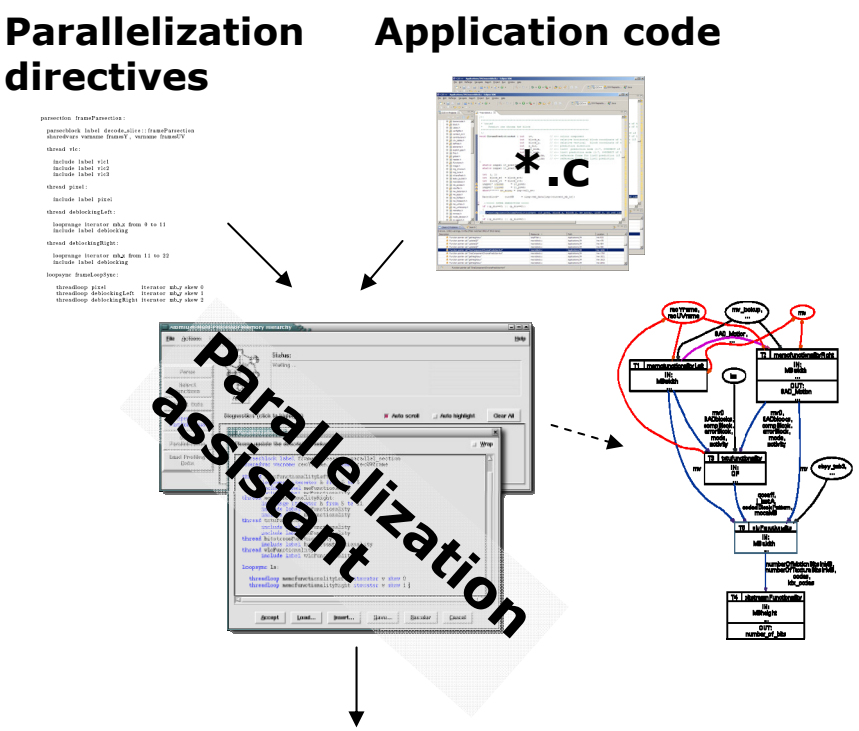
Parallelization of code

- Current and emerging practice
  - Using a multiprocessor OS/RTOS
  - Pre-parallelized libraries
  - Standards such as MPI and OpenMP
  - OpenMP (http://www.compunity.org/)

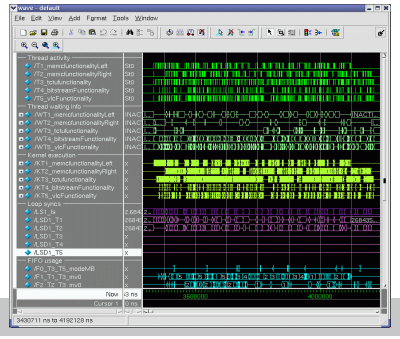
Parallelization of code

- Current and emerging practice
  - Using a multiprocessor OS/RTOS
  - Pre-parallelized libraries
  - Standards such as MPI and OpenMP
  - OpenMP (http://www.compunity.org/)
- Longer term solutions
  - New and better parallel programming models, languages
  - Parallelizing compiler
- Compiler-assisted parallelization

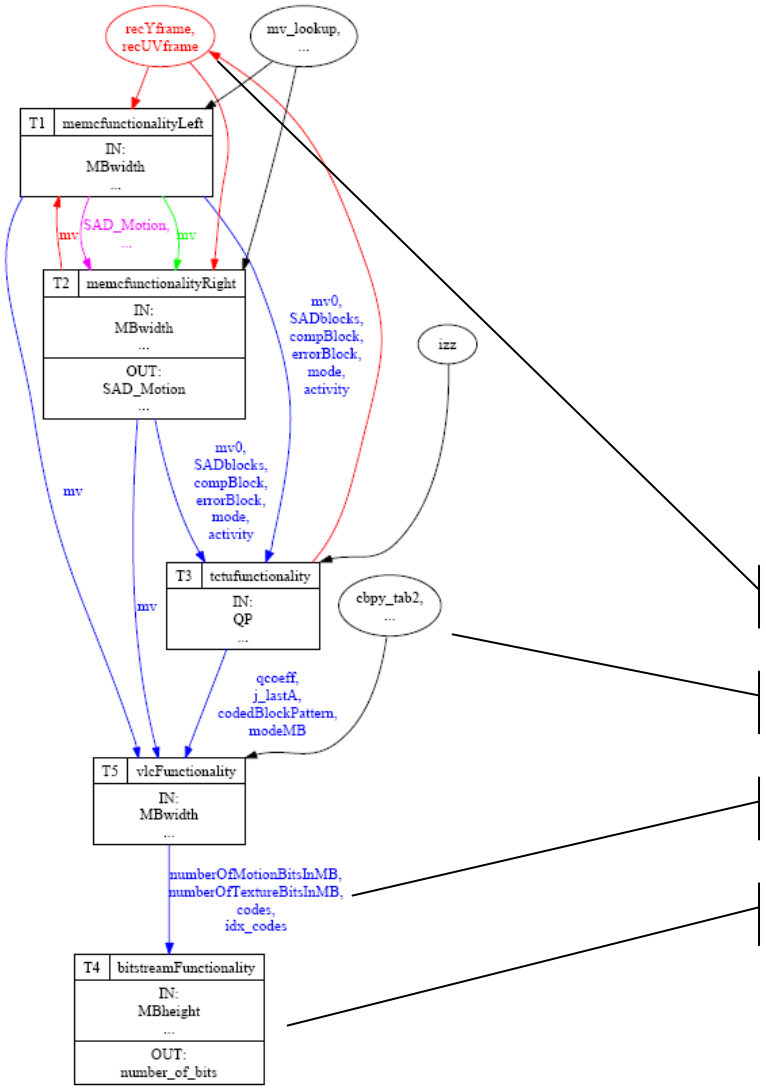
Compiler assisted parallelization: MPA



- ✓ Parallelizes sequential C source code
  - ✓ *Correct-by-construction multi-threaded code*
  - ✓ Designer in charge
  - ✓ More expressive directives than OpenMP
- ✓ Supports types of parallelism
  - ✓ Functional split
  - ✓ (Coarse) Data-level split
  - ✓ Combinations
- ✓ Dumps parallel code
- ✓ Sets up communication
  - ✓ Communication by means of FIFO's
  - ✓ DMA transfers
  - ✓ FIFO sizes determined by tool (initial version)



MPA static analysis



- Provides insight in parallel software architecture
- Helps in identification of parallelization bottlenecks

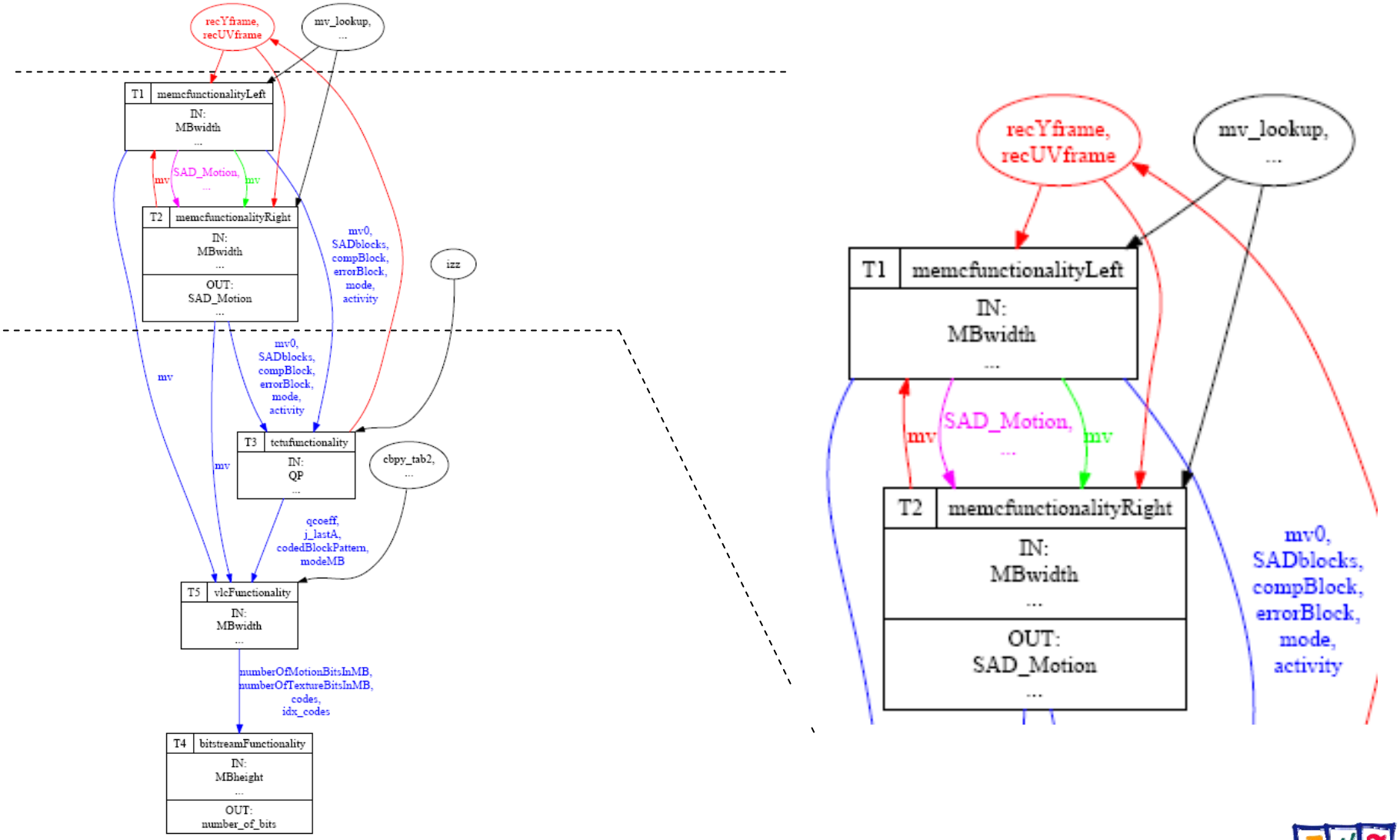
Shared data

Global constant data

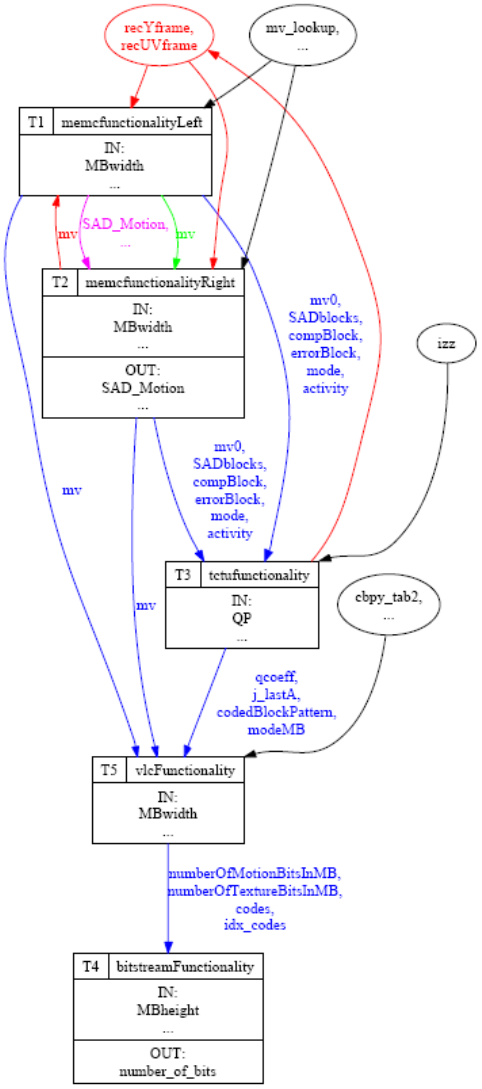
FIFO communicated data

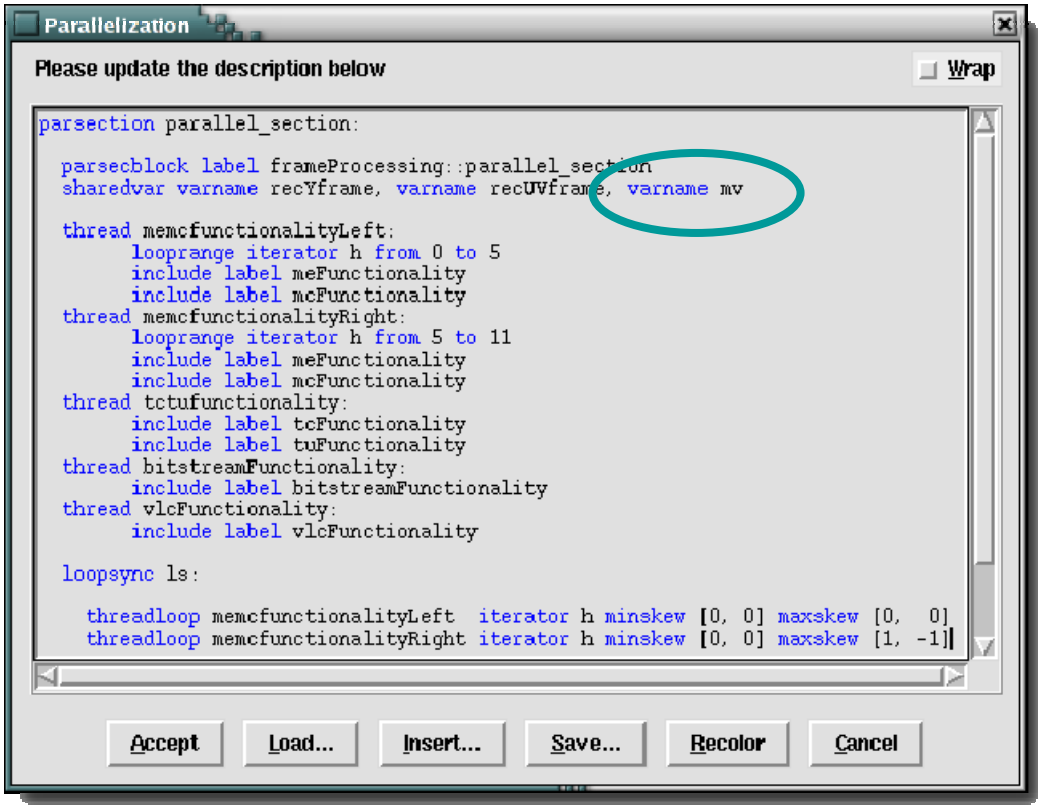
Thread

MPA static analysis

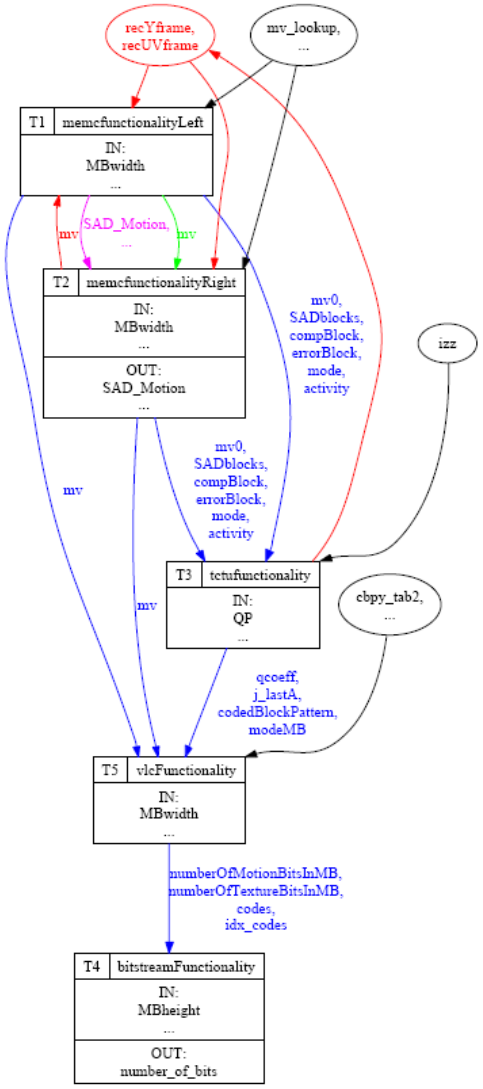


MPA user interaction

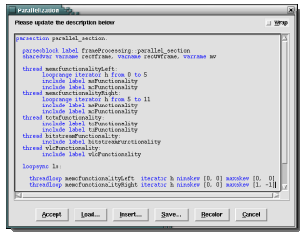


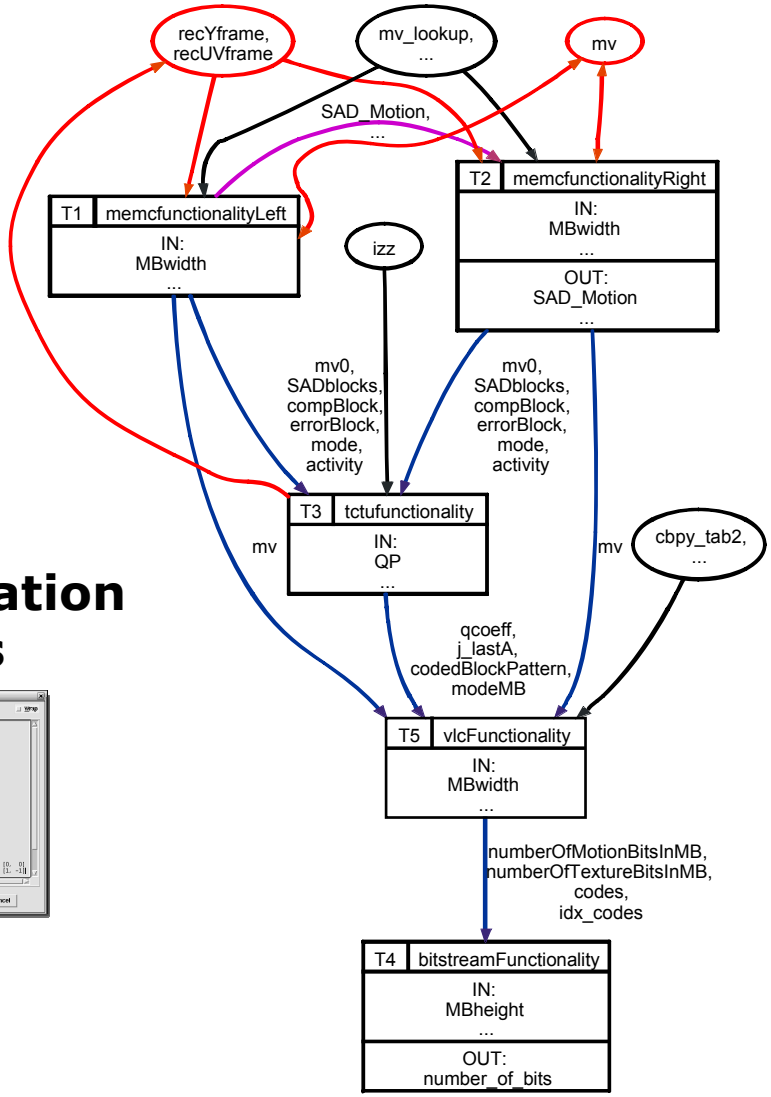


MPA user interaction

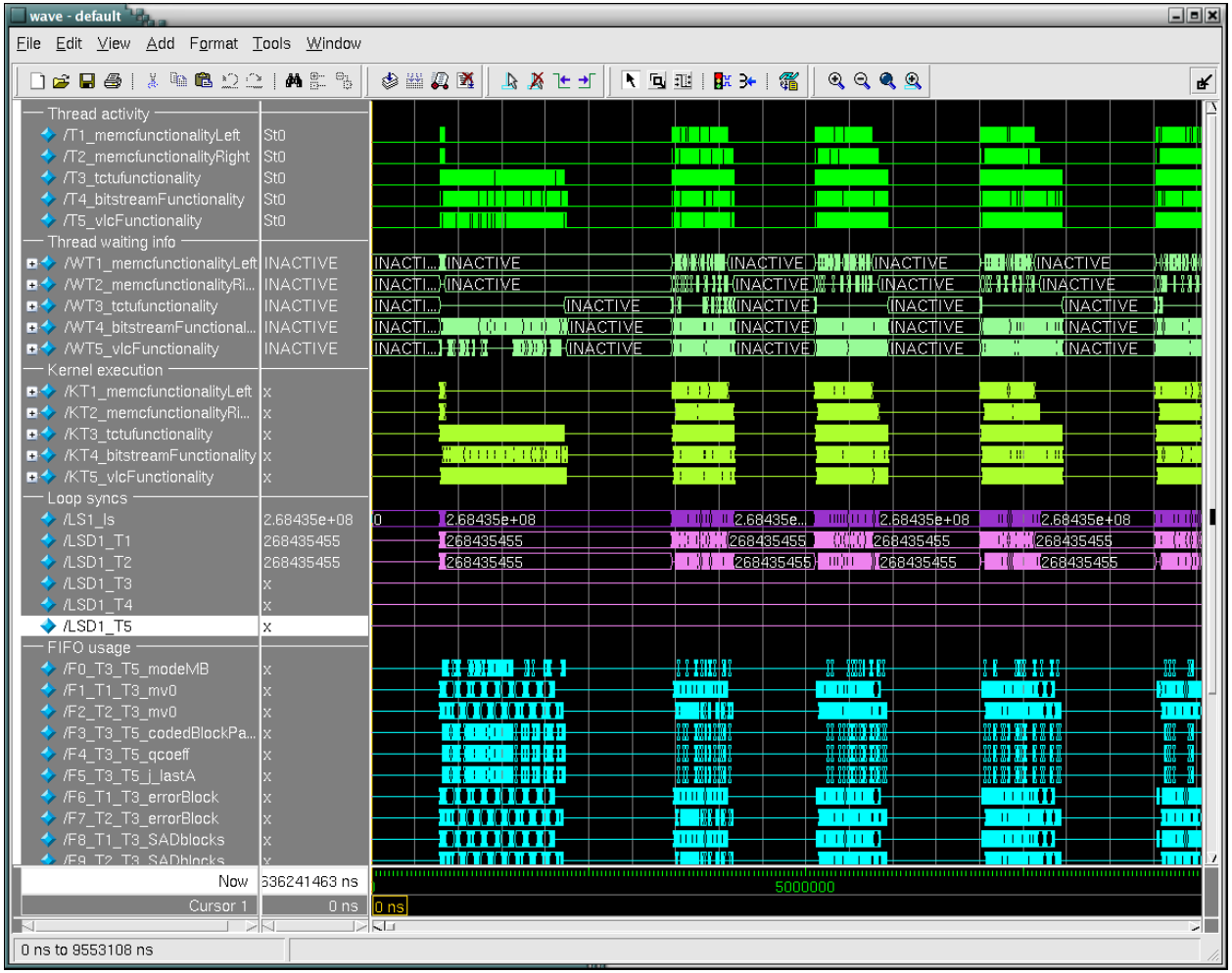


Rewrite parallelization directives





Evaluate quality of resulting parallel code



Thread activity

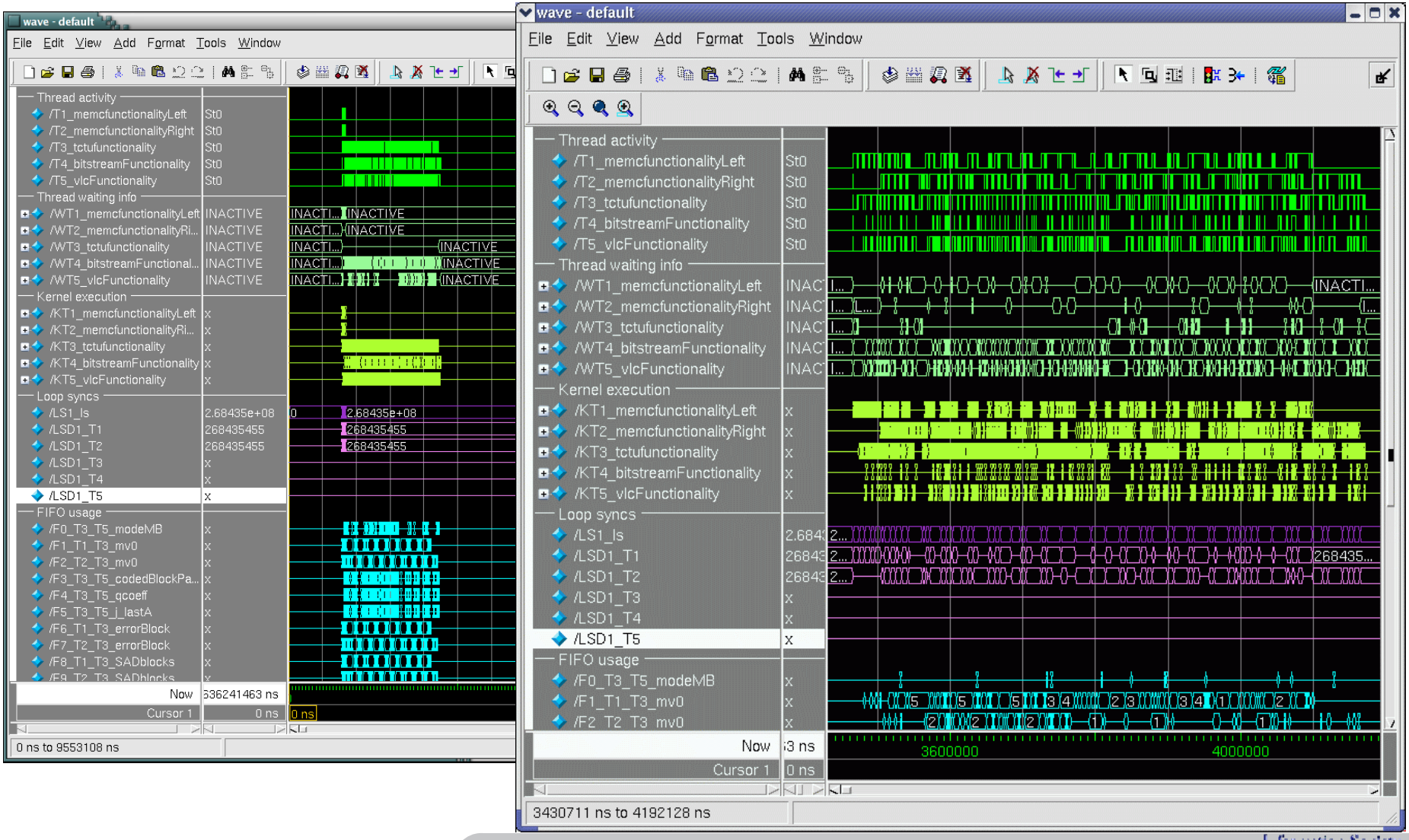
Thread idle

Kernel execution

Thread synchronization

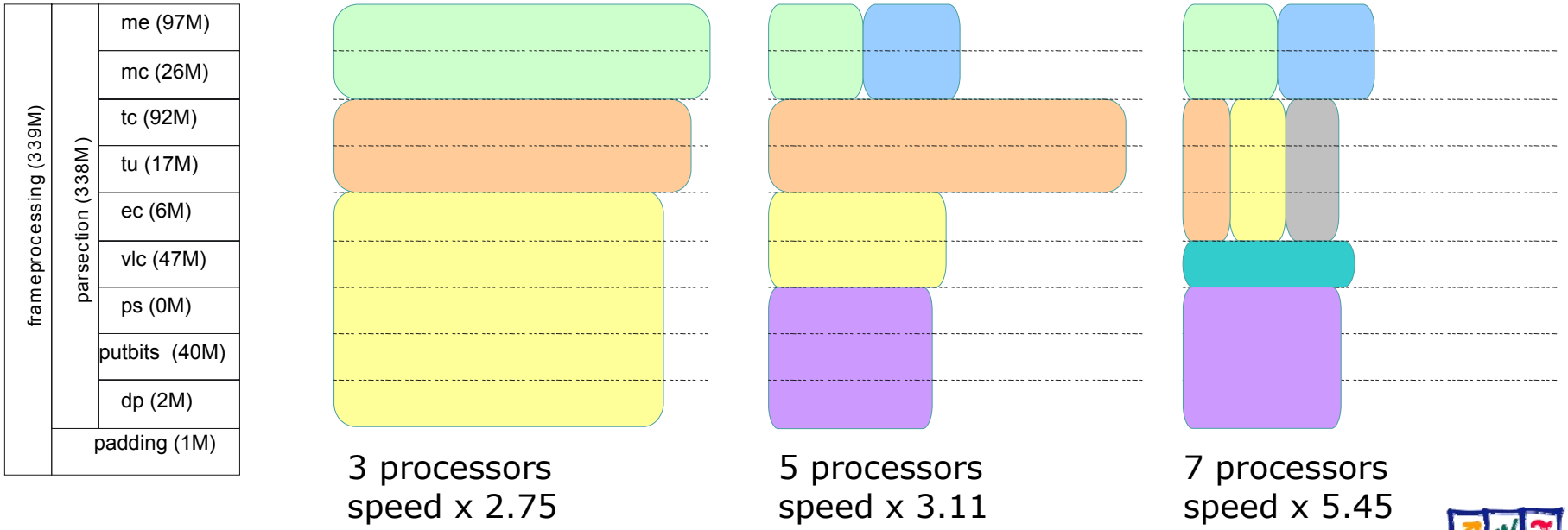
Communication buffers

Evaluate quality of resulting parallel code



Parallelization of code - results

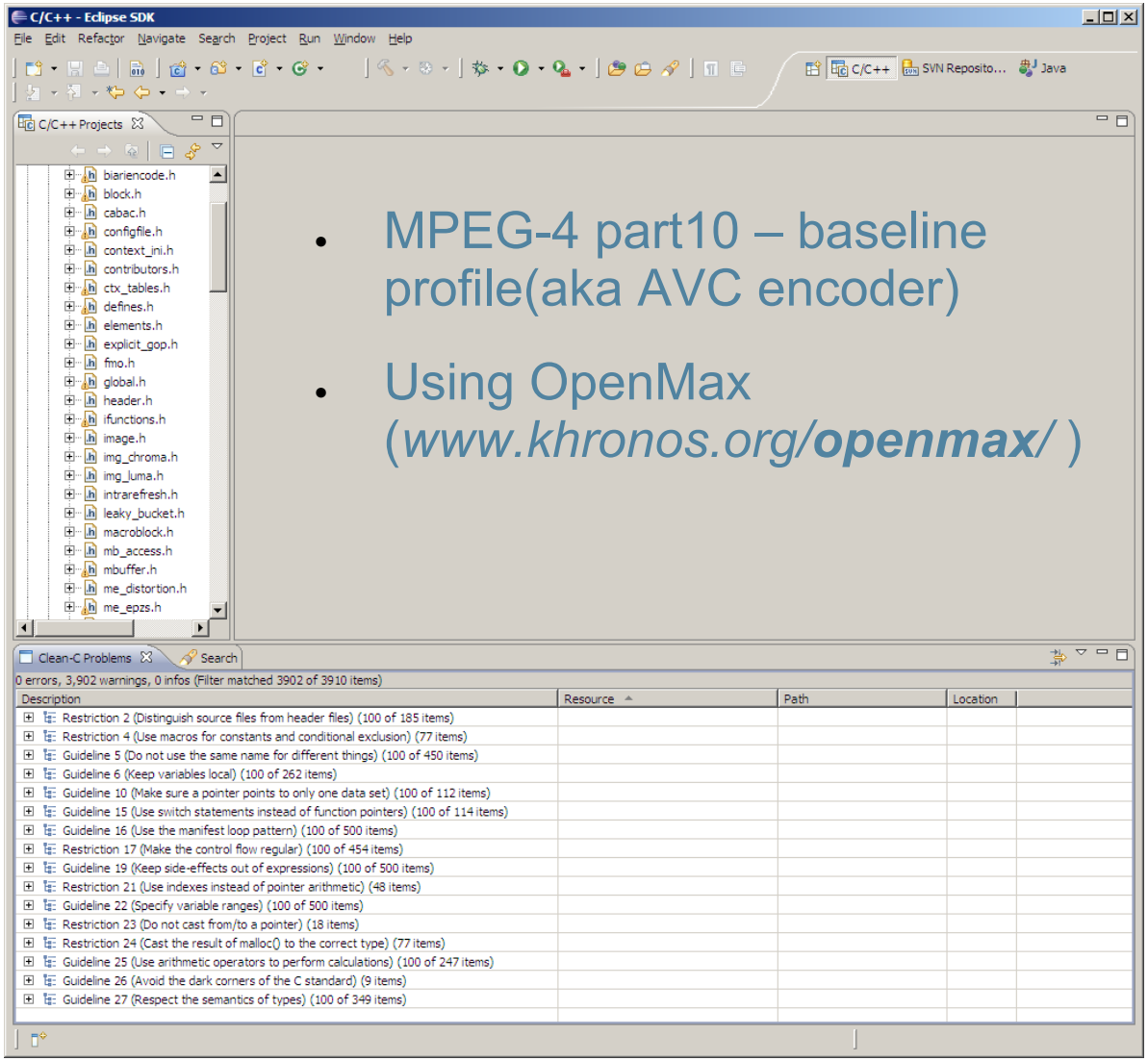
- Prototype tool used to explore different parallel software architecture for MPEG-4 part2 SP encoder
  - 10 parallelization alternatives explored in half a day
  - 20 to 30 lines of parallelization directives



IMEC research goals

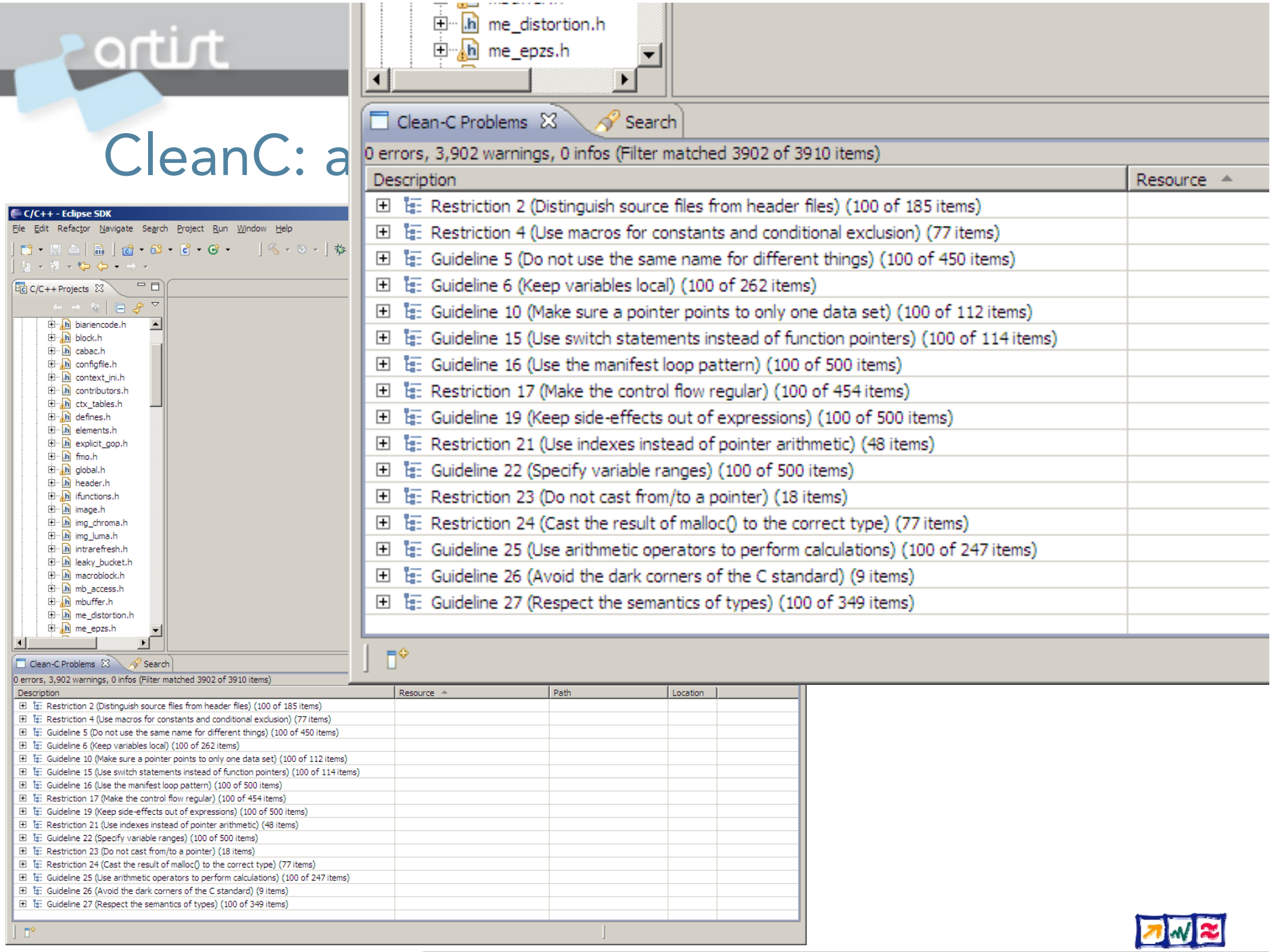
- Compiler-like tools
  - Assist application software developer to find efficient mapping on multi-processor
  - Generate correct by construction code
  - Based on static analysis of C-code
  - User interaction (read intelligence) required to obtain optimal results
- Does it work on any C code?
  - Static analysis has its limitations
  - IMEC promotes CleanC, a coding style that is MPSoC friendly
    - 28 guidelines and rules
    - CleanC code is easier to analyse
    - Plug-in for Eclipse 3.3 / CDT 4.0 IDE checks compliance
      → See http://www.imec.be/cleanC/

CleanC: an MPSoC-friendly coding style

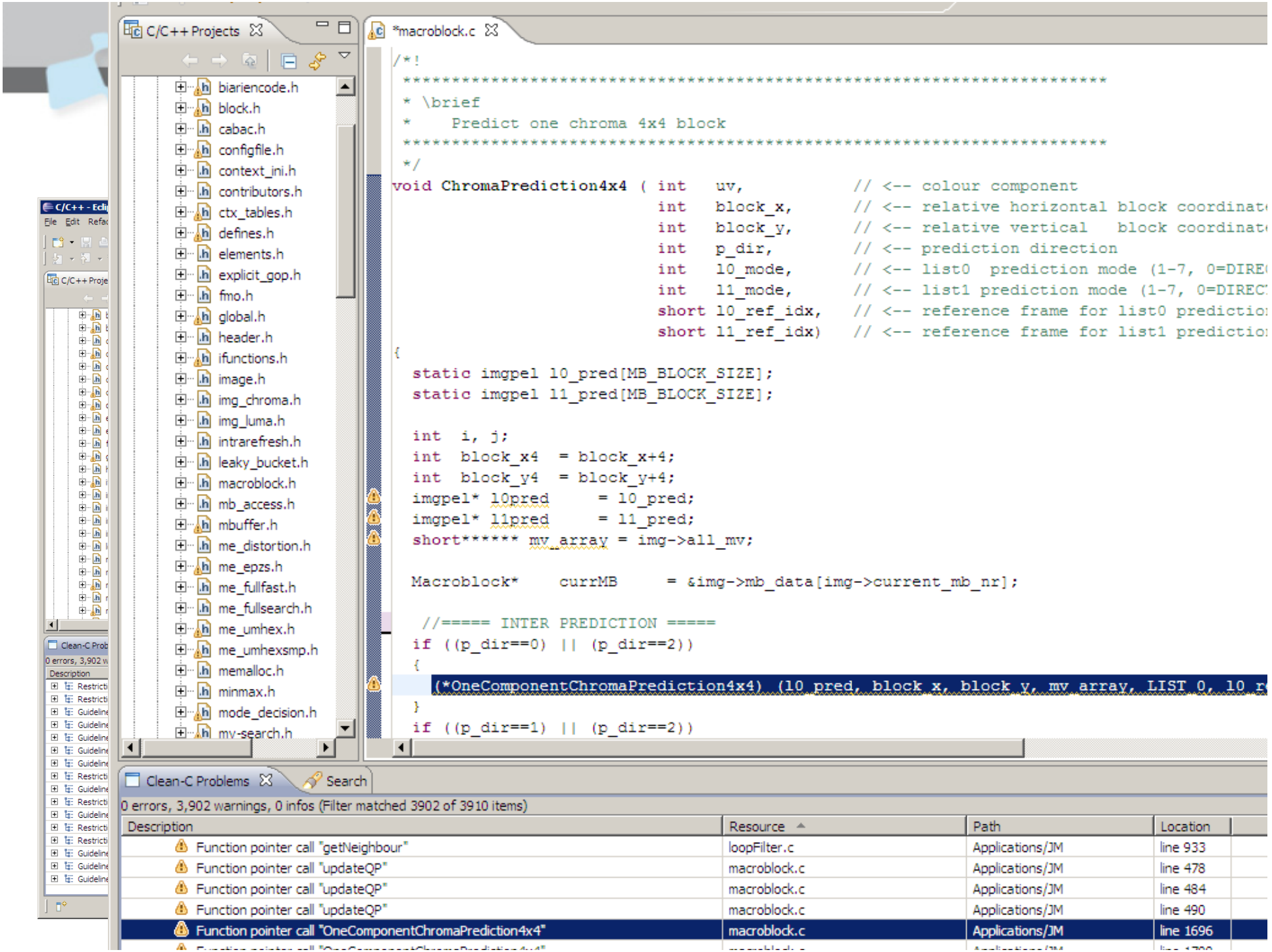


- MPEG-4 part10 – baseline profile(aka AVC encoder)
- Using OpenMax (*www.khronos.org/**openmax**/*)

CleanC: a



Information Society


C/C++ Projects
biariencode.h
block.h
cabac.h
configfile.h
context_ini.h
contributors.h
ctx_tables.h
defines.h
elements.h
explicit_gop.h
fmo.h
global.h
header.h
ifunctions.h
image.h
img_chroma.h
img_luma.h
intrarefresh.h
leaky_bucket.h
macroblock.h
mb_access.h
mbuffer.h
me_distortion.h
me_epzs.h
me_fullfast.h
me_fullsearch.h
me_umhex.h
me_umhexsmp.h
memalloc.h
minmax.h
mode_decision.h
*macroblock.c
/*!
* \brief
* Predict one chroma 4x4 block
*/
void ChromaPrediction4x4 (int uv, // <-- colour component
int block_x, // <-- relative horizontal block coordinat
int block_y, // <-- relative vertical block coordinat
int p_dir, // <-- prediction direction
int l0_mode, // <-- list0 prediction mode (1-7, 0=DIRE
int l1_mode, // <-- list1 prediction mode (1-7, 0=DIREC
short l0_ref_idx, // <-- reference frame for list0 predictio
short l1_ref_idx) // <-- reference frame for list1 predictio
{
static imgpel l0_pred[MB_BLOCK_SIZE];
static imgpel l1_pred[MB_BLOCK_SIZE];
int i, j;
int block_x4 = block_x+4;
int block_y4 = block_y+4;
imgpel* l0pred = l0_pred;
imgpel* l1pred = l1_pred;
short****** mv_array = img->all_mv;
Macroblock* currMB = &img->mb_data[img->current_mb_nr];
//===== INTER PREDICTION =====
if ((p_dir==0) || (p_dir==2))
{
(*OneComponentChromaPrediction4x4) (l0_pred, block_x, block_y, mv_array, LIST_0, l0_r
}
if ((p_dir==1) || (p_dir==2))
Clean-C Problems
Search
0 errors, 3,902 warnings, 0 infos (Filter matched 3902 of 3910 items)
Description
Resource
Path
Location
Function pointer call "getNeighbour"
loopFilter.c
Applications/JM
line 933
Function pointer call "updateQP"
macroblock.c
Applications/JM
line 478
Function pointer call "updateQP"
macroblock.c
Applications/JM
line 484
Function pointer call "updateQP"
macroblock.c
Applications/JM
line 490
Function pointer call "OneComponentChromaPrediction4x4"
macroblock.c
Applications/JM
line 1696

IMEC research goals

- Compiler-like tools to assist software developers
  - Sequential C code in, correct by construction (parallel) C code out
    - Taking care of scratchpad management
    - Assisting with parallelization of code
  - Embedded software developer in charge
- Run-time support to manage multiple applications in predictable fashion

IMEC research goals

- Compiler-like tools to assist software developers
  - Sequential C code in, correct by construction (parallel) C code out
    - Taking care of scratchpad management
    - Assisting with parallelization of code
  - Embedded software developer in charge
- ***Run-time support to manage multiple applications in predictable fashion***

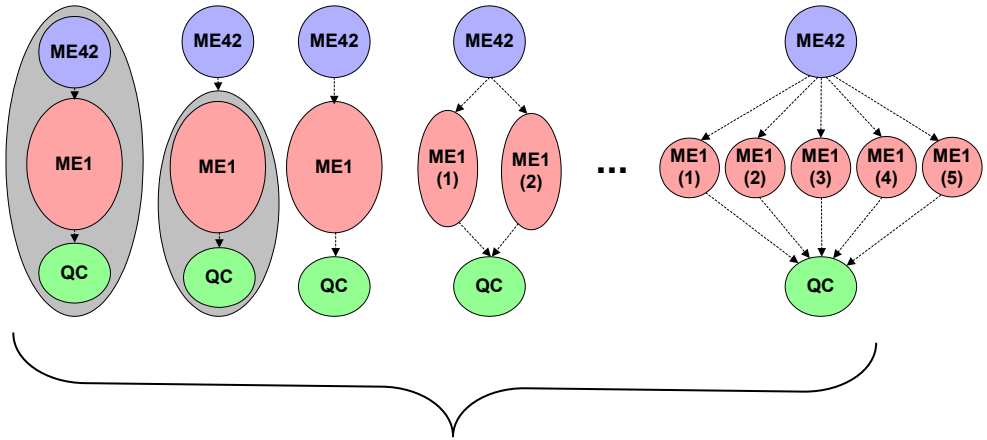
Multi-application mapping

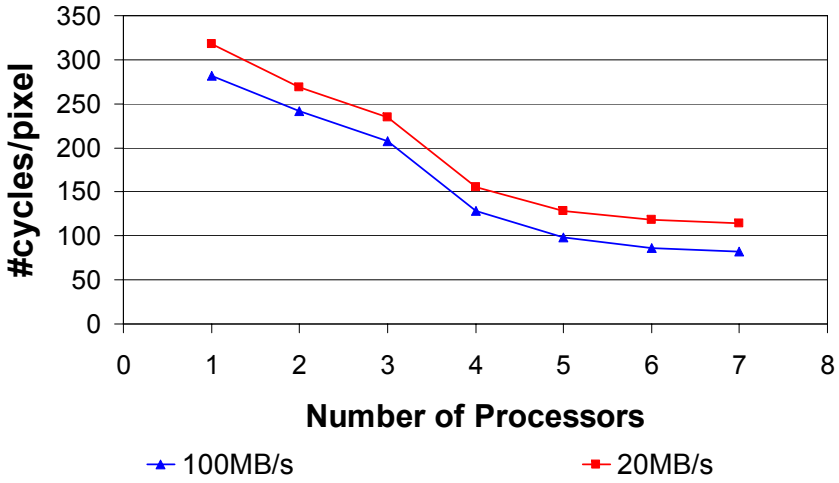
- Multiple applications on multi-core platforms
  - Processing power is available for executing multiple applications concurrently
    - MPEG-4 part 2 SP encoder with MPEG-4 part 10 decoder
    - Audio and video, graphics, ...
  - Communication and memory resources of platform are shared between applications
    - One application's behavior (mapping) affects other application's performance
    - This interaction is very dynamic
      - Application behavior is inherently dynamic
      - User or environment decide dynamically which applications need to run in parallel and what performance is required
    - Cannot analyze all possible interactions at design time

Multi-application mapping

- Run-time management system
  - Deals with interactions between applications
  - Information about application's behavior, expected platform resource usage, actual platform resource usage, etc is provided to run-time system manager as meta-data.
  - Run-time system manager uses meta-data to decide which resources to assign to which application
    - Provides predictable application behavior
    - Availability of multiple operating points for an application creates additional flexibility for run-time manager to accommodate several applications

Multi-application mapping

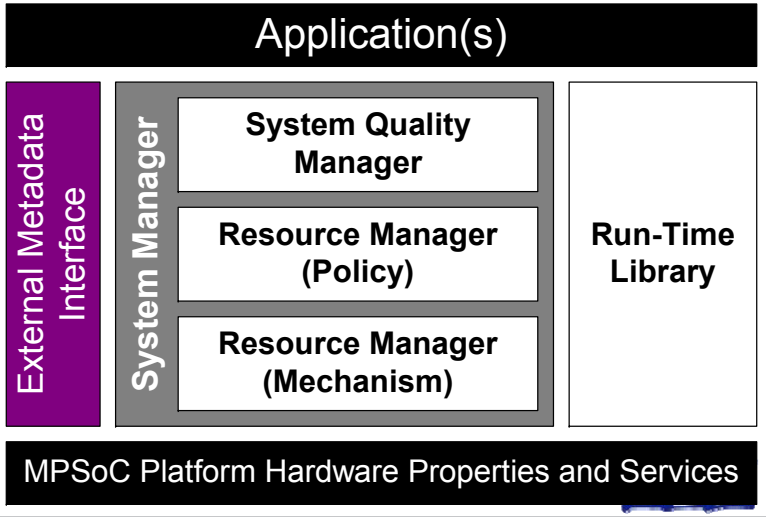




Video CODEC

Metadata

BW, #proc.



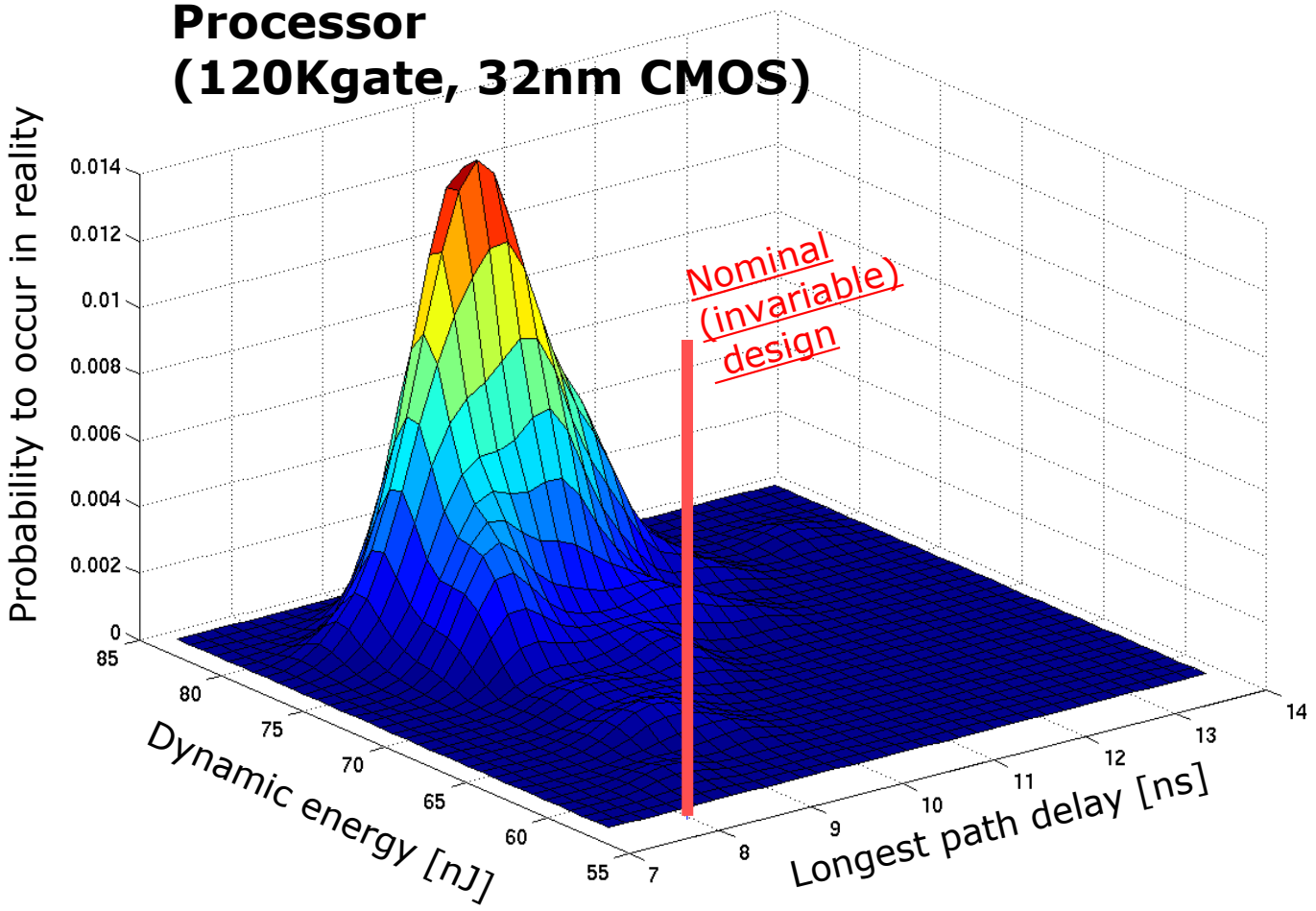
Conclusions

- IMEC research focuses on compiler-like tools (C in, C out) to program embedded multi-core systems
  - Exploiting scratch pad memory
  - Assisting in parallelization of code
  - Generate *correct-by-construction* code
  - CleanC: coding style to allow efficient static analysis
  - Predictability of multiple applications sharing multi-core platform through run-time control.
- More information
  - http://www.imec.be/
  - http://www.imec.be/cleanC

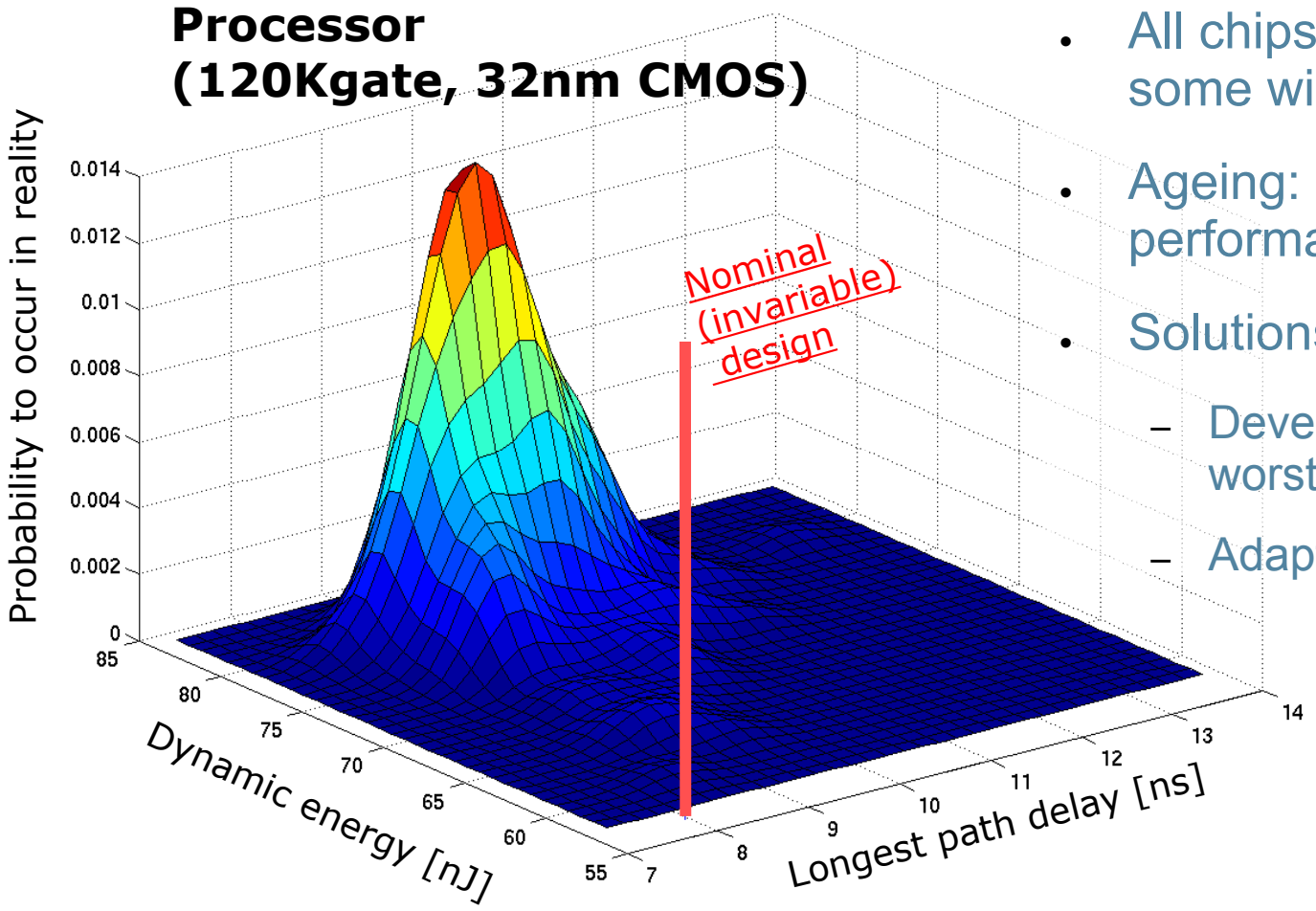
A glimpse of the future

- What the future brings
  - More and more run-time solutions required
    - Multi-core evolves to many-core
    - Applications and usage patterns grow increasingly dynamic
    - Advanced CMOS processing technology becomes a source of un-predictability: run-time mitigation of variability/reliability issues

Variability in process technology



Variability in process technology



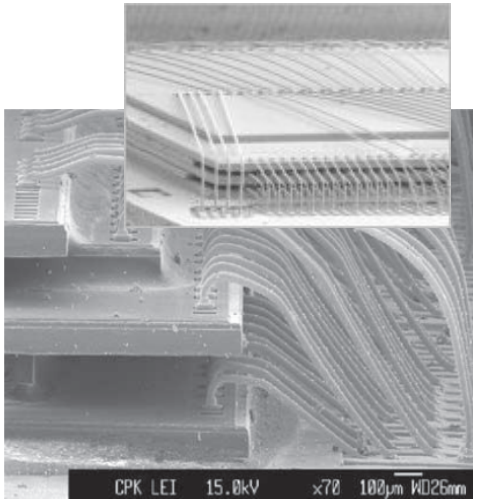
- All chips are equal – but some will be more equal
- Ageing: degradation of performance over time
- Solutions:
  - Develop software for the worst-case?
  - Adaptive system?

A glimpse of the future

- What the future brings
  - More and more run-time solutions required
    - Multi-core evolves to many-core
    - Applications and usage patterns grow increasingly dynamic
    - Advanced CMOS processing technology becomes a source of un-predictability: run-time mitigation of variability/reliability issues
  - Advances in integration technologies
    - 3D-stacking of devices with Through-Silicon-Via
    - Will radically change the way we build memory hierarchies and hence multi-core architectures

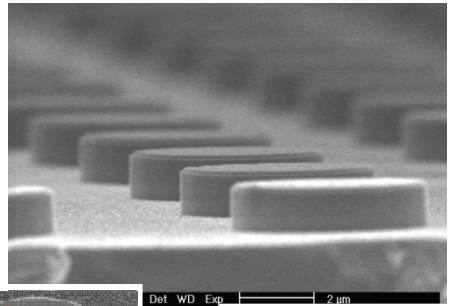
3D stacking – for MPSoC

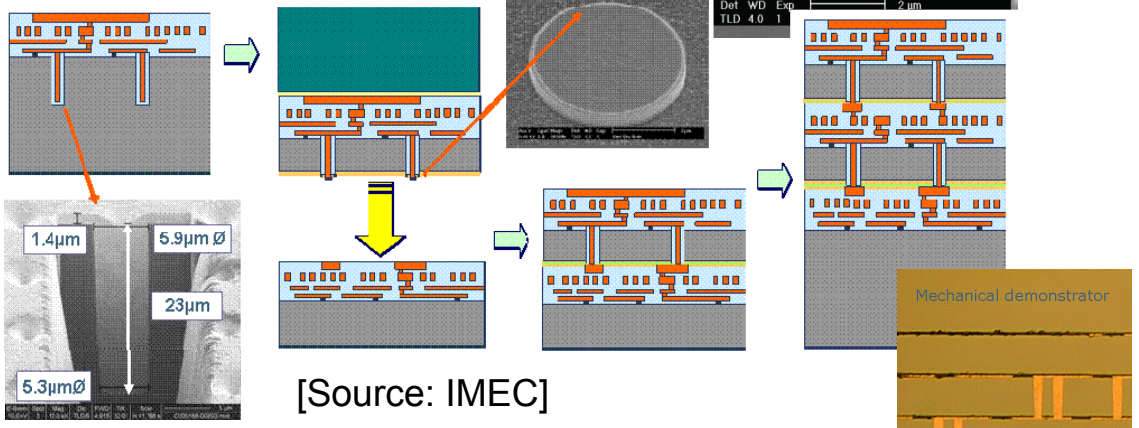
- Stacking chips – the traditional way



[Source: STATS ChipPAC]

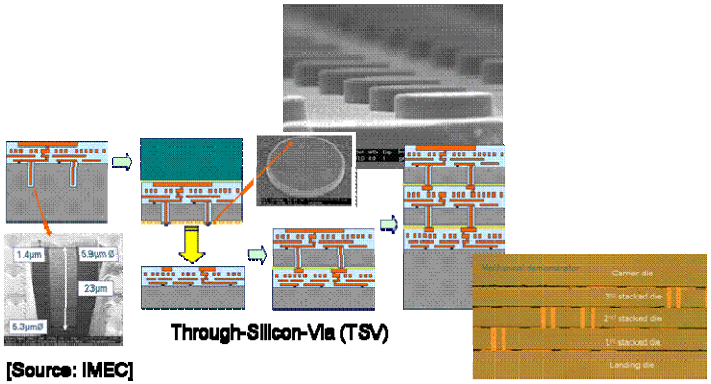
- and Through-Silicon-Via (TSV)



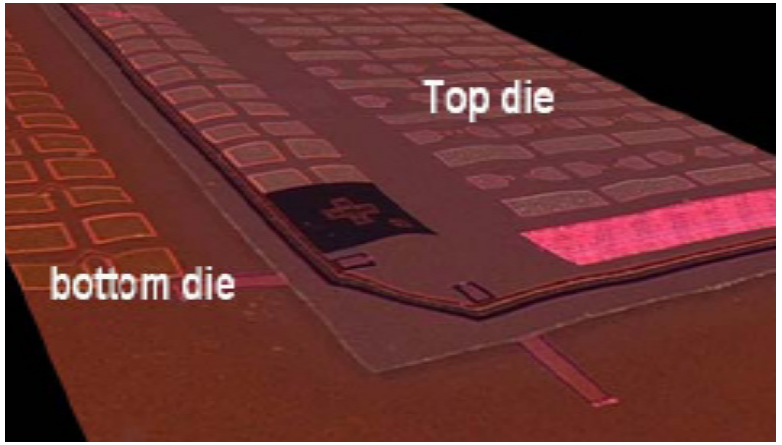


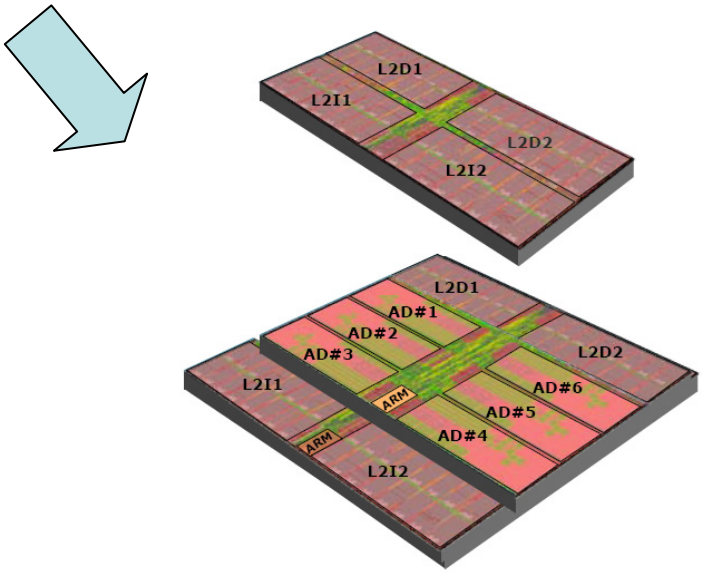
[Source: IMEC]

3D stacking – for MPSoC



[Source: IMEC]





Replace SRAM by stacked DRAM

40% of die area are SRAM
- Scale badly below 45nm
- High leakage
- Read/Write energy
- Low area efficiency

Thank you

And thanks to the IMEC research teams working on MPSoC, Technology-Aware Design, and 3D integration