technische universität
dortmund

fakultät für informatik
informatik 12

Efficient Computing in Cyber-Physical Systems

Peter Marwedel
TU Dortmund (Germany)
Informatik 12

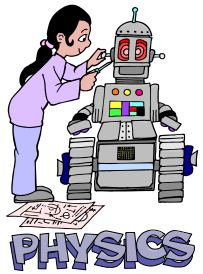
2013年 07 月 09 日

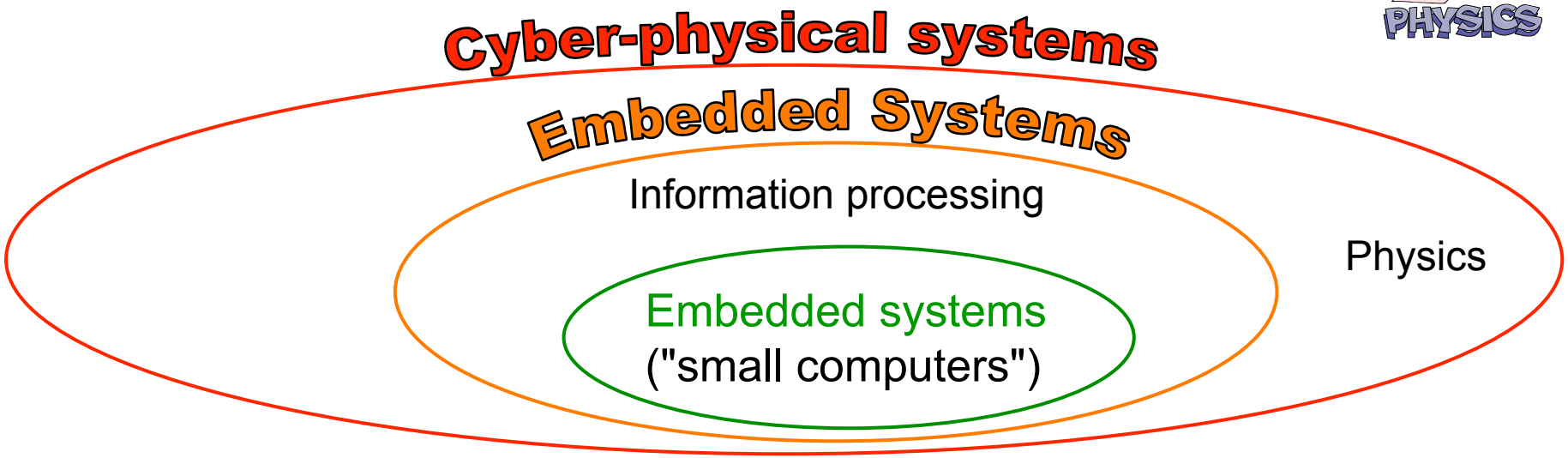
Photos: P. Marwedel

Cyber-physical systems and embedded systems

- Embedded systems (ES): information processing embedded into a larger product [Peter Marwedel, 2003]
- Cyber-physical systems (CPS): integrations of computation with physical processes [Edward Lee, 2006].

CPS = ES + physical environment



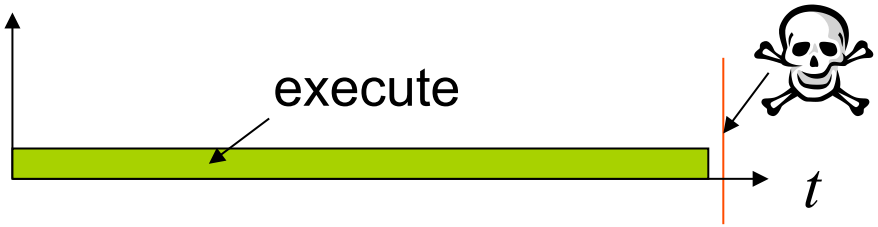


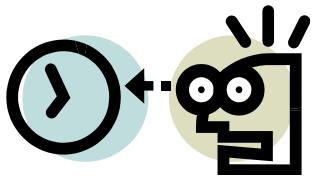
Requirements for CPS/ES design

1. Timing

The system must react within a time interval dictated by the environment

[Adopted from Bergé]





- Consequences frequently underestimated
- **Always!** Not subject to statistical arguments
  (except possibly extremely small probabilities)

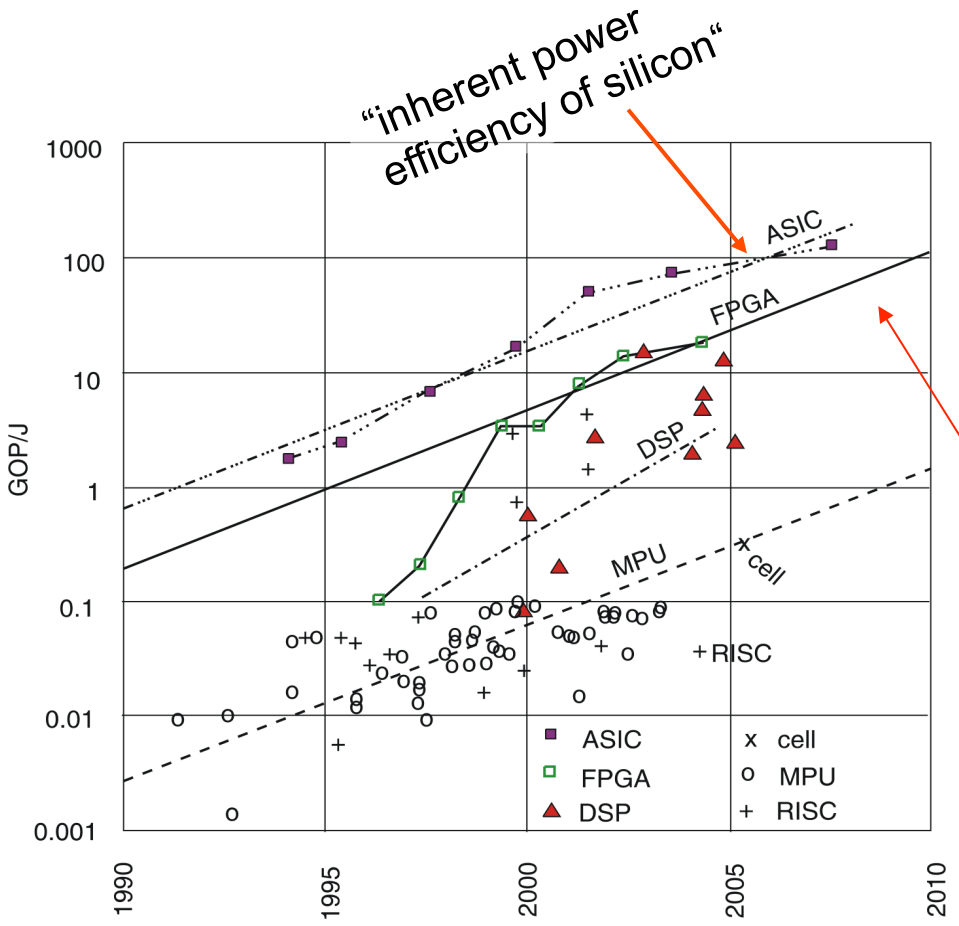
Requirements for CPS/ES design

2. Energy efficiency

The system must be energy efficient

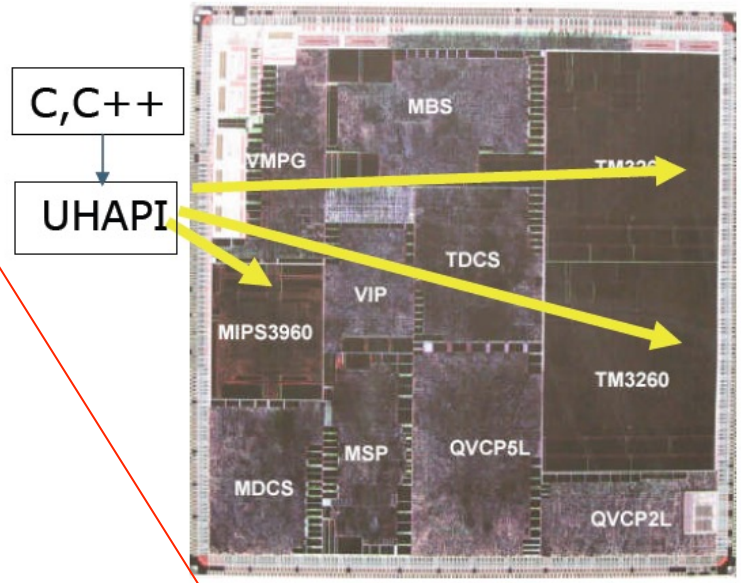
- Consequences frequently underestimated
  e.g. necessary use of heterogeneous processors

Consequences: heterogeneous processors



© Hugo De Man: From the Heaven of Software to the Hell of Nanoscale Physics: An Industry in Transition, *Keynote Slides*, ACACES, 2007

Nexperia Digital Video Platform NXP

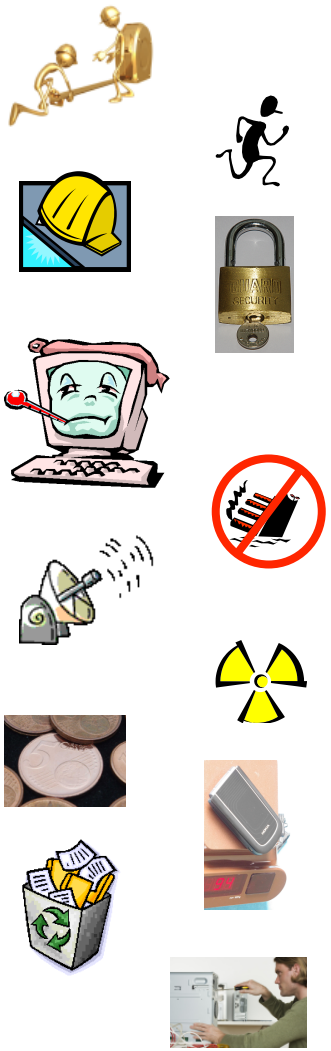


1 MIPS, 2 Trimedia
60 coproc, 250 RAM's
266MHz, 1.5 watt 100 Gops

Close to power efficiency of silicon

Requirements (3)

- exactness of results, QoS
- average case delay
- safety
- security
- thermal behavior
- reliability
- EMC characteristics
- radiation hardness
- cost, size
- weight
- environmental friendliness
- extendability …



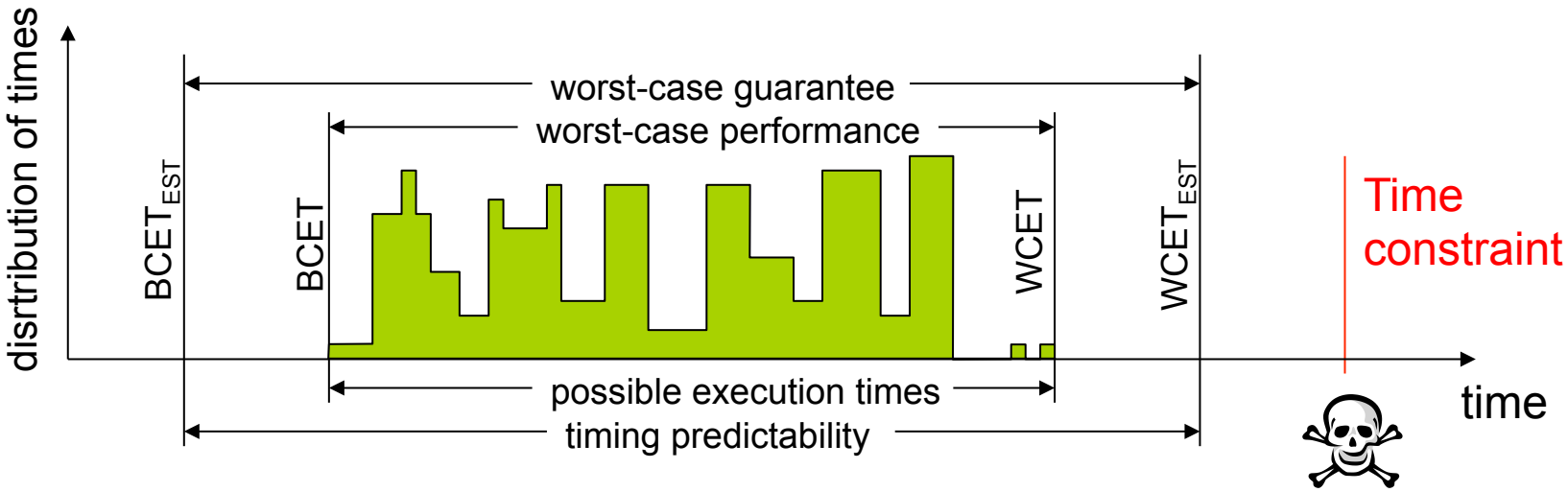
Threatened by increased openness

Outline

- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

Let's look at timing!

Definition of worst case execution time:



$WCET_{EST}$ must be

1. safe (i.e. ≥ WCET) and
2. tight ($WCET_{EST}$-WCET≪$WCET_{EST}$)

Current Trial-and-Error Based Development

1. Specification of CPS/ES system
2. Generation of Code (ANSI-C or similar)
3. Compilation of Code
4. Execution and/or simulation of code,
   using a (e.g. random) set of input data
5. Measurement-based computation of "estimated worst case execution time" ($WCET_{meas}$)
6. Adding safety margin m on top of $WCET_{meas}$:
   $WCET_{hypo} := (1 + m) * WCET_{meas}$
7. If "$WCET_{hypo}$" > deadline: change some detail, go back to 1 or 2.

Problems with this Approach

Dependability

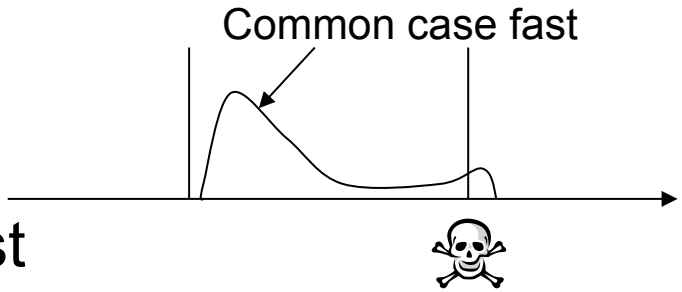
- Computed "$WCET_{hypo}$" not a safe approximation
- ☞ Time constraint may be violated

Design time

- How to find necessary changes?
- How many iterations until successful?

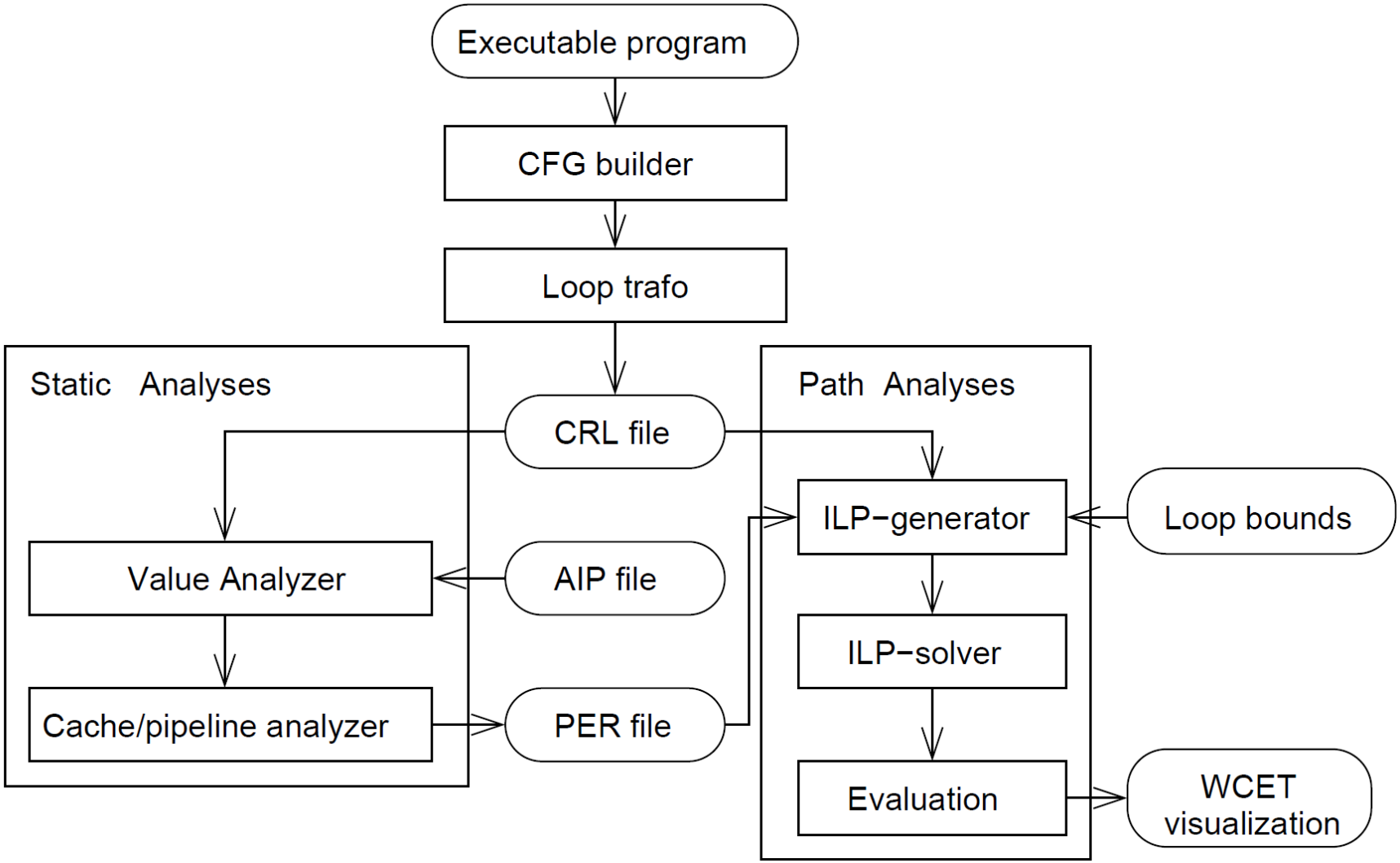
"Make the common case fast" a wrong approach for RT-systems

- Computer architecture and compiler techniques focus on average speed
- Circuit designers know it's wrong
- Compiler designers (typically) don't



"Optimizing" compilers unaware of cost functions other than code size

WCET estimation: aiT (AbsInt)



How to compute $WCET_{EST}$?
1. Generation of control flow graph (CFG)

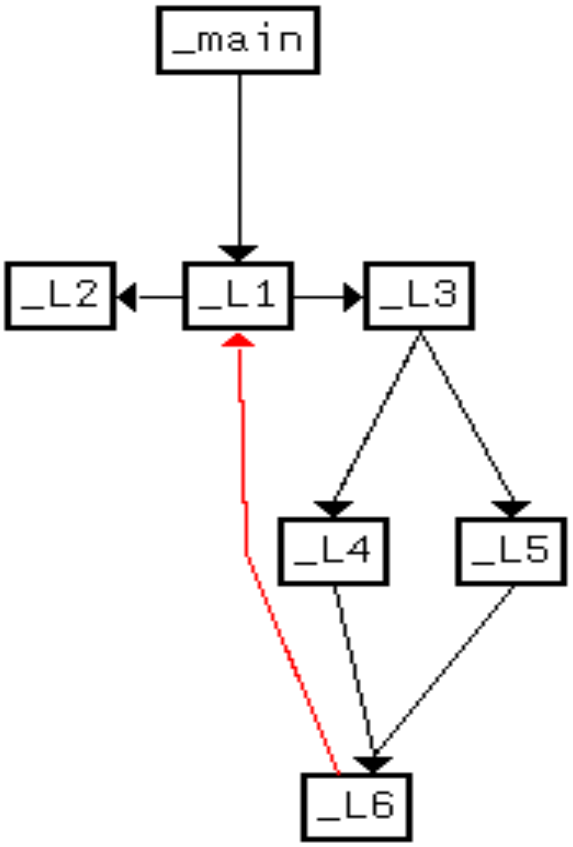
Program

```
int main()
{
  int i, j = 0;

  _Pragma( "loopbound min
                  100 max 100" );
  for ( i = 0; i < 100; i++ ) {
    if ( i < 50 )
      j += i;
    else
      j += ( i * 13 ) % 42;
  }

  return j;
}
```

CFG



2. Computation of $WCET_{EST}$ for each block

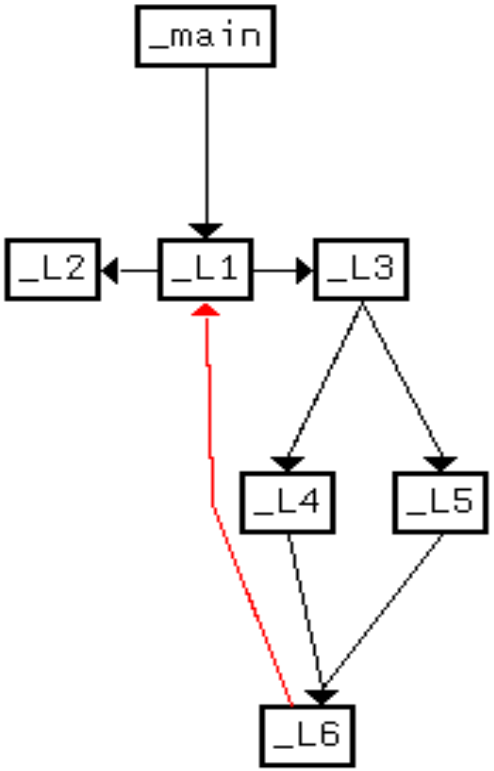
Program

```
int main()
{
  int i, j = 0;

  _Pragma( "loopbound min
                  100 max 100" );
  for ( i = 0; i < 100; i++ ) {
    if ( i < 50 )
      j += i;
    else
      j += ( i * 13 ) % 42;
  }

  return j;
}
```

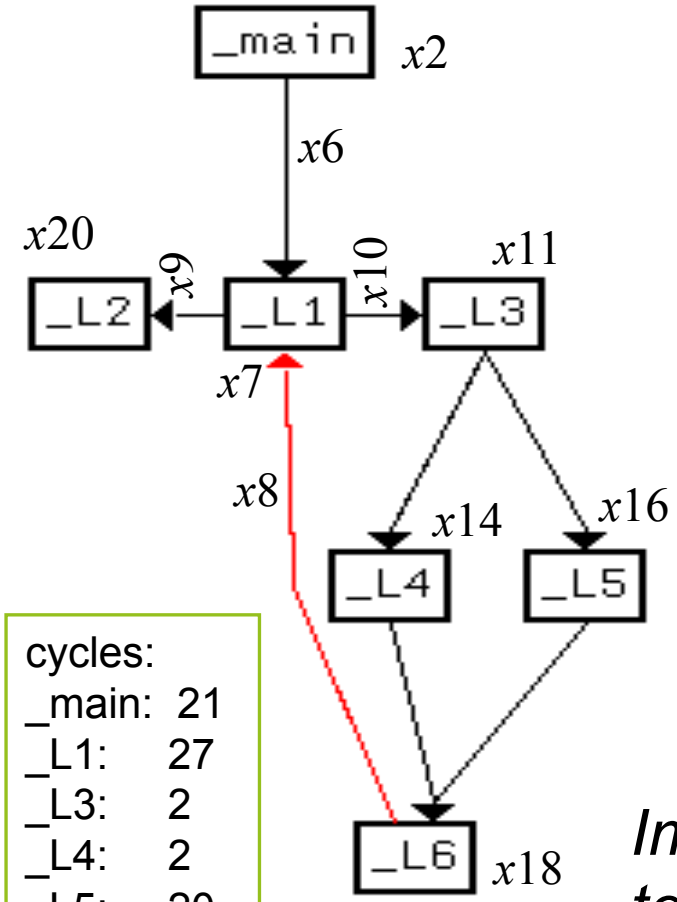
CFG



$WCET_{EST}$ of basic blocks using e.g. abstract interpretation (☞ AbsInt aiT tool for TriCore)

_main: 21 Cycles
_L1: 27
_L3: 2
_L4: 2
_L5: 20
_L6: 13
_L2: 20

3. Mapping to ILP program



cycles:
_main: 21
_L1: 27
_L3: 2
_L4: 2
_L5: 20
_L6: 13
_L2: 20

ILP problem

/* execution time, to be maximized*/

$21\ x2+27\ x7+2\ x11+2\ x14+20\ x16+13\ x18+20\ x19;$

/* Constraints model relations in the CFG */

/* Constraints for flow into node _L1 */

$x7 - x8 - x6 = 0;$

/* Constraints for flow out of node _L1 */

$x7 - x9 - x10 = 0;$

…

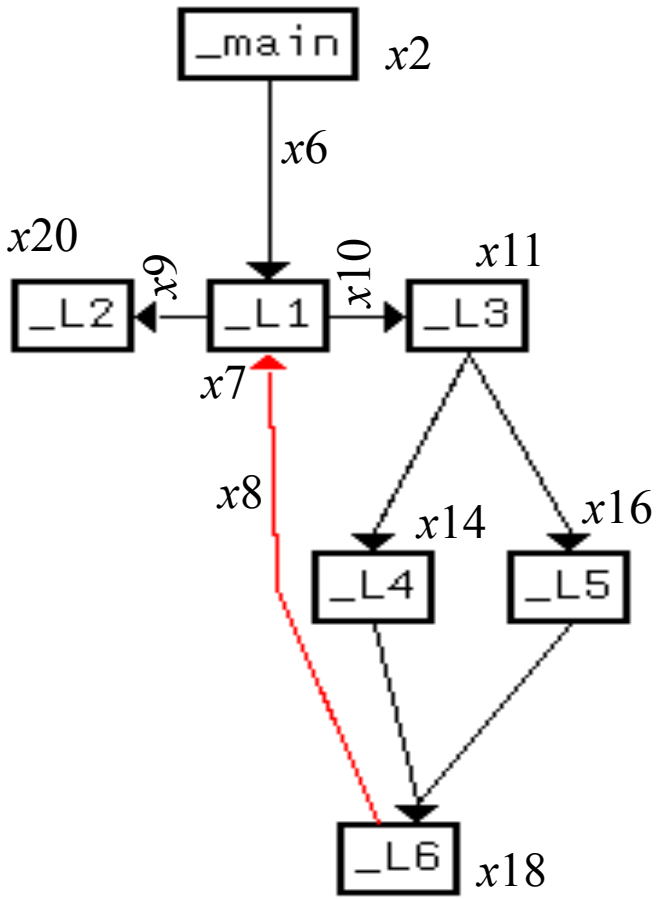
Implicit path enumeration technique: Explicit enumeration avoided!

4. Using ILP package

Objective function (# of cycles): 6268

Values maximizing execution time

x2	1
x7	101
x11	100
x14	0
x16	100
x18	100
x19	1
x0	1
x4	1
x1	1
x5	1
x3	1
x20	1
x6	1
x8	100
x9	1
x10	100
x12	100
x13	0
x15	0
x17	100



Outline

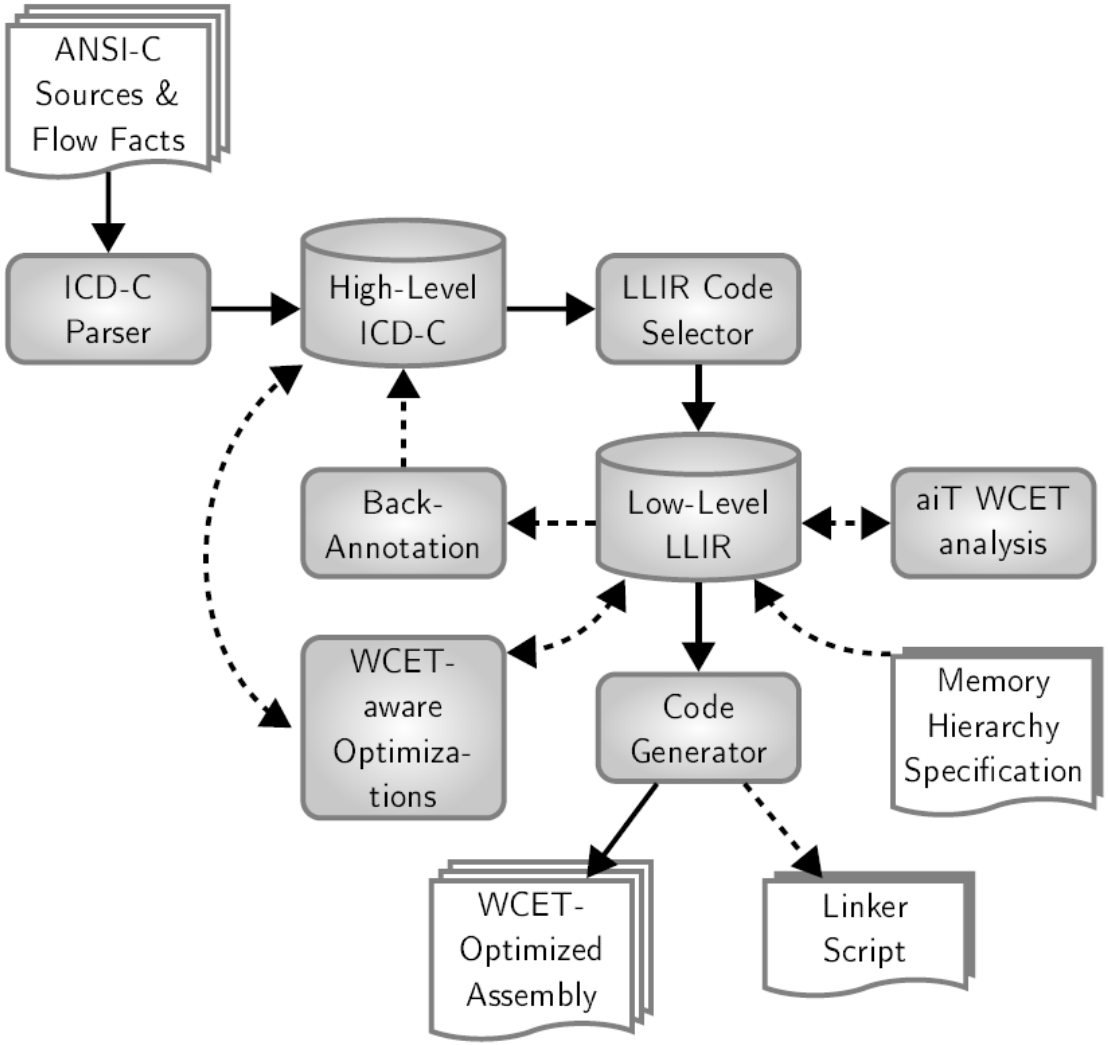
- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

Integration of WCET estimation and compilation

Computing $WCET_{EST}$ **after** code generation is too late.

Why not consider $WCET_{EST}$ as an objective function already in the compiler?

☞ Integration of aiT and compiler



WCET-oriented optimizations

- Extended loop analysis (CGO 09)
- Instruction cache locking (CODES/ISSS 07, CGO 12)
- Cache partitioning (WCET-Workshop 09)
- Procedure cloning (WCET-WS 07, CODES 07, SCOPES 08)
- Procedure/code positioning (ECRTS 08, CASES 11 (2x))
- Function inlining (SMART 09, SMART 10)
- Loop unswitching/invariant paths (SCOPES 09)
- Loop unrolling (ECRTS 09)
- Register allocation (DAC 09, ECRTS 11))
- Scratchpad optimization (DAC 09)
- Extension towards multi-objective optimization (RTSS 08)
- Superblock-based optimizations (ICESS 10)
- Surveys (Springer 10, Springer 12)

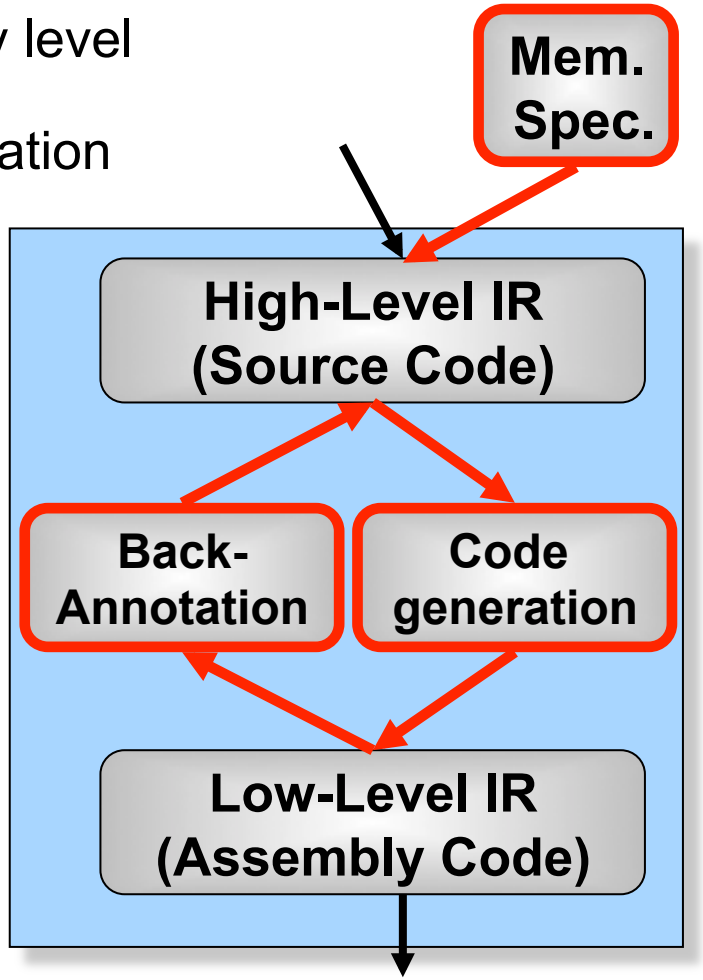
Loop Unrolling as an Example

- Unrolling replaces the original loop with several instances of the loop body
- Positive Effects
  - Reduced overhead for loop control
  - Enables instruction level parallelism (ILP)
  - Offers potential for following optimizations
- Unroll *early* in optimization chain
- Negative Effects
  - Aggressive unrolling leads to I-cache overflows
  - Additional spill code instructions
  - Control code may cancel positive effects

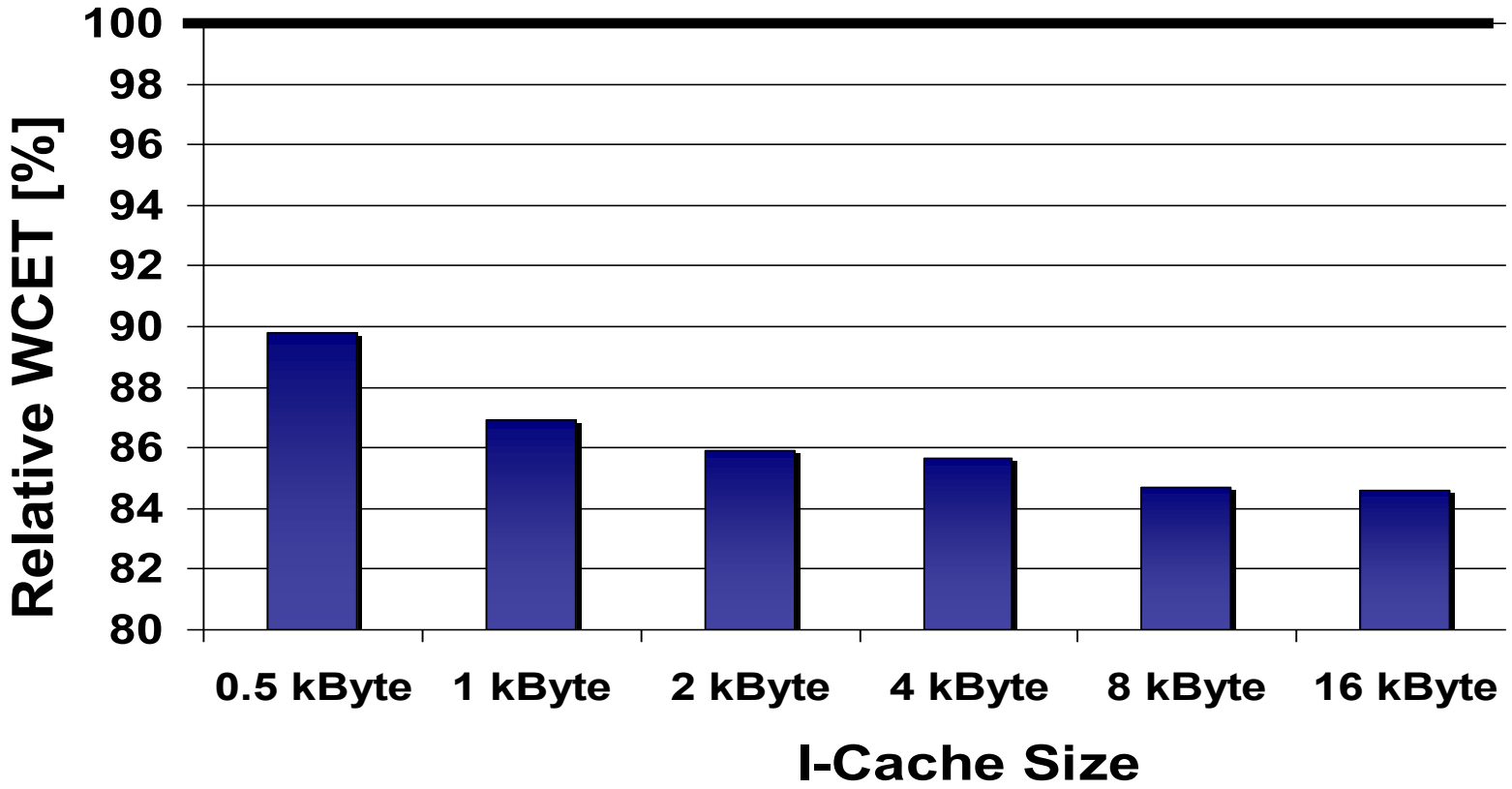
Consequences of transformation hardly known

$WCET_{EST}$-aware Loop Unrolling via Back-annotation

- $WCET_{EST}$-information available at assembly level
- Unrolling to be applied at internal representation of source code
- Solution: Back-annotation:
  Experimental $WCET_{EST}$-aware compiler WCC allows copying information:
  assembly code ☞ source code
  - $WCET_{EST}$ data
  - Assembly code size
  - Amount of spill code
- Memory architecture info available



Results for unrolling: $WCET_{EST}$

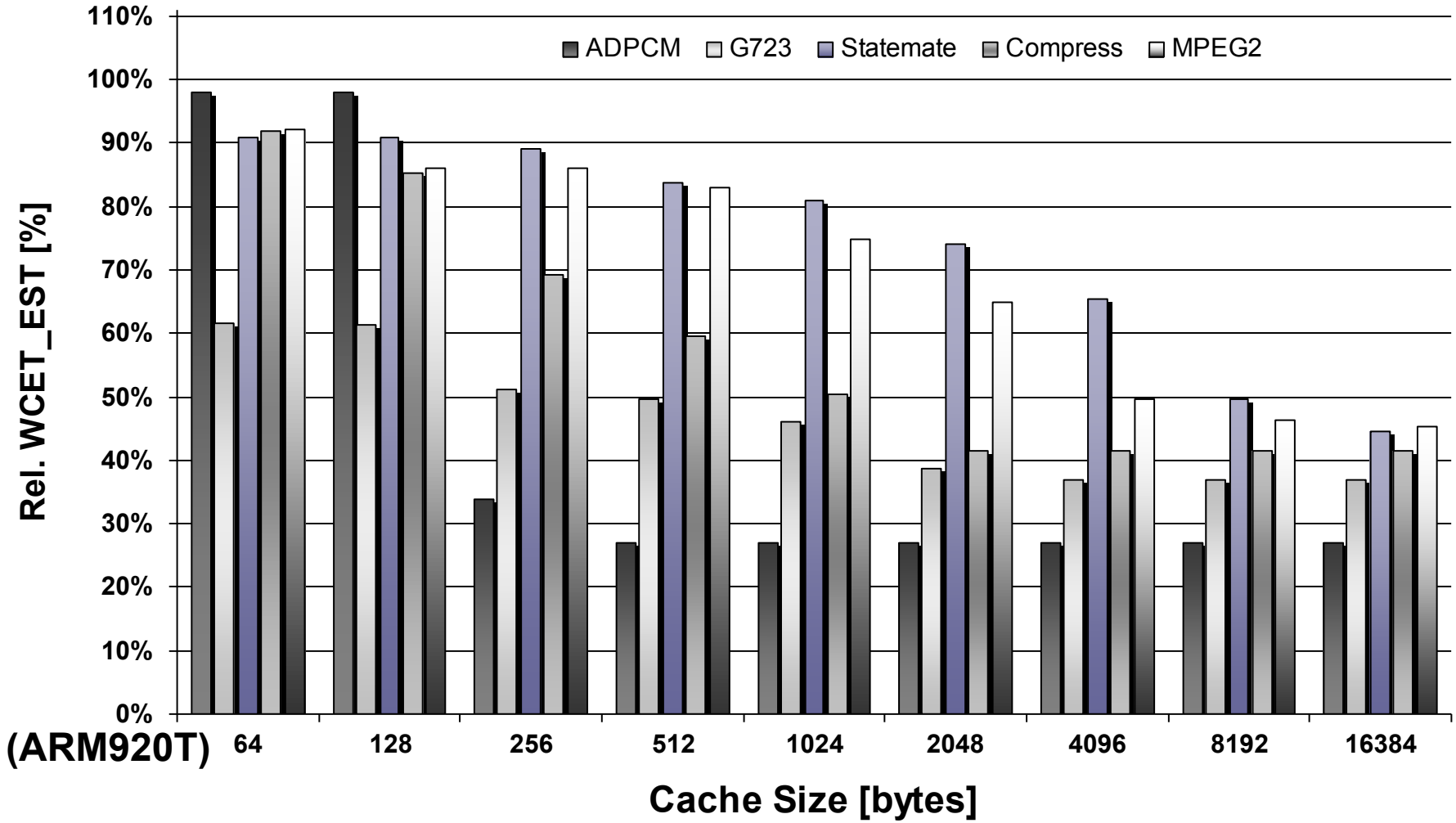


100%: Avg. $WCET_{EST}$ for all benchmarks with –O3 & no unrolling

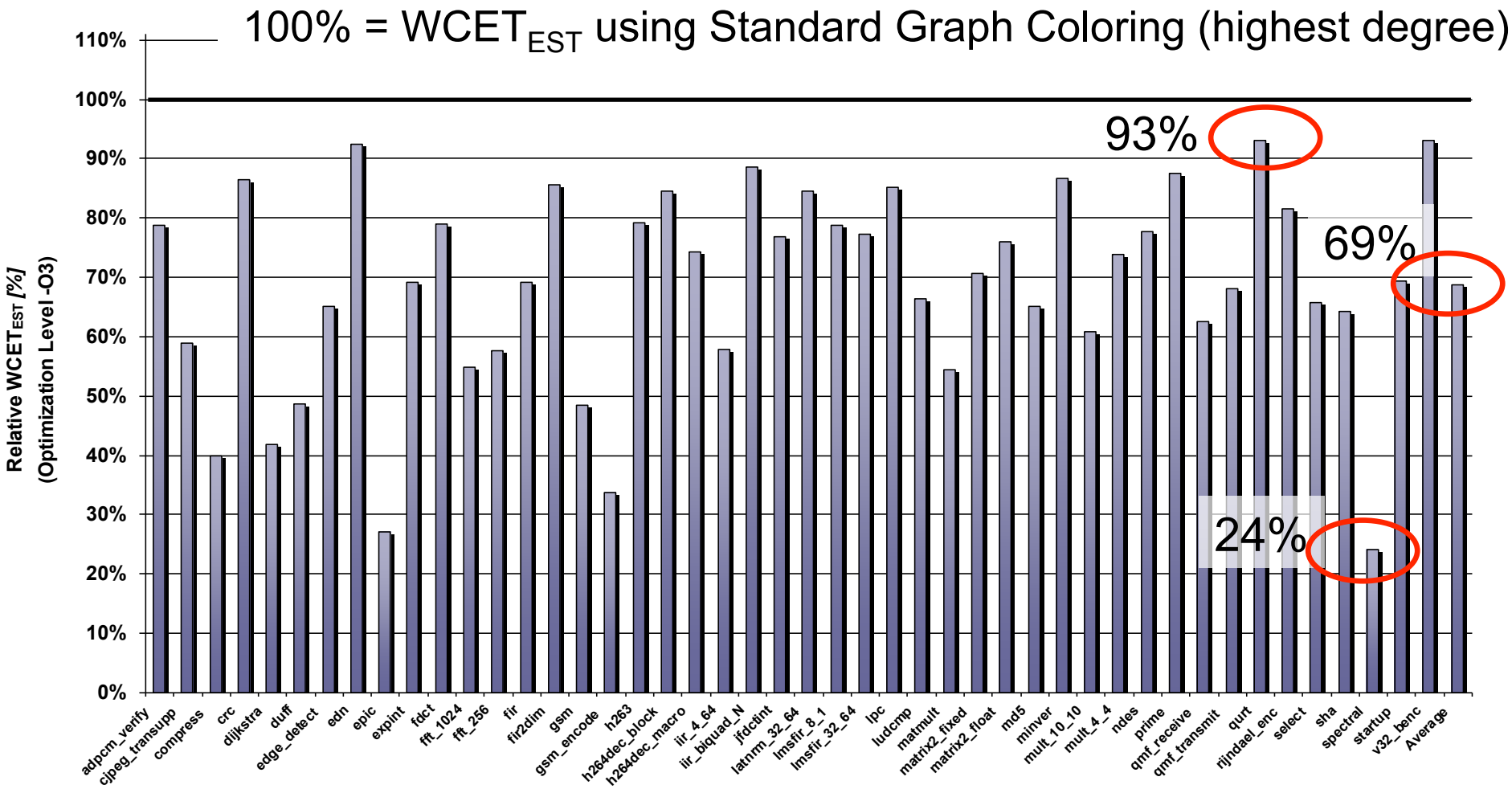
$WCET_{EST}$ reduction between 10.2% and 15.4%

$WCET_{EST}$-driven Unrolling outperforms standard unrolling by 13.7%

Relative $WCET_{EST}$ with I-*Cache Locking 5 Benchmarks/ARM920T/Postpass-Opt*

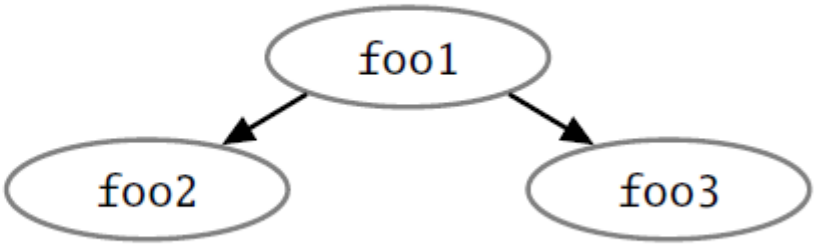


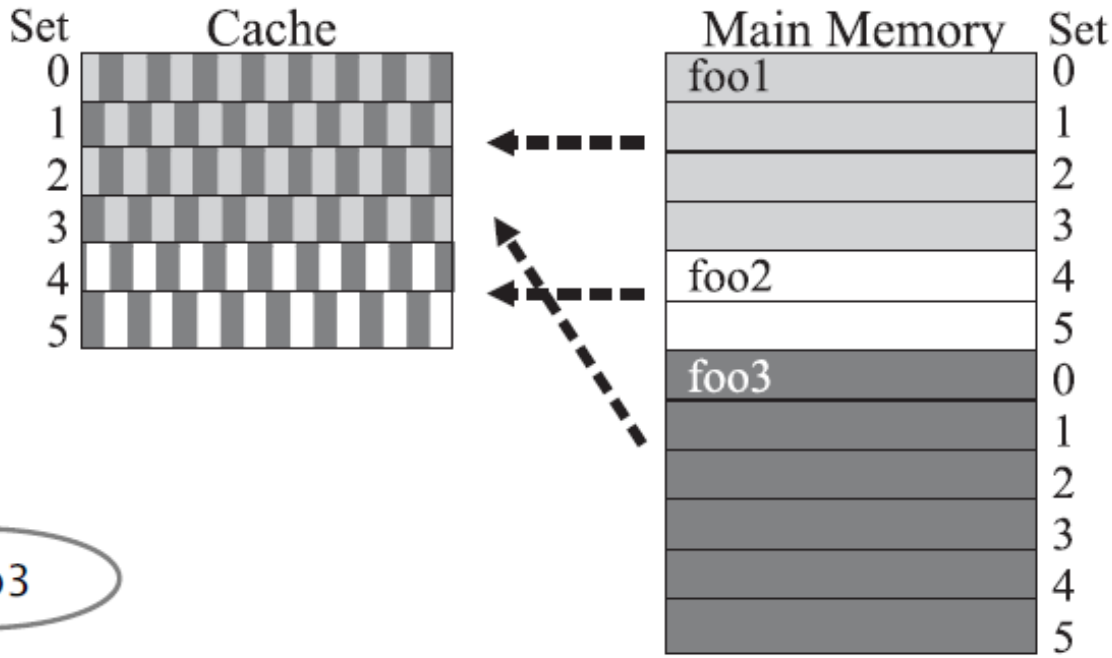
Register Allocation



Potential cache thrashing

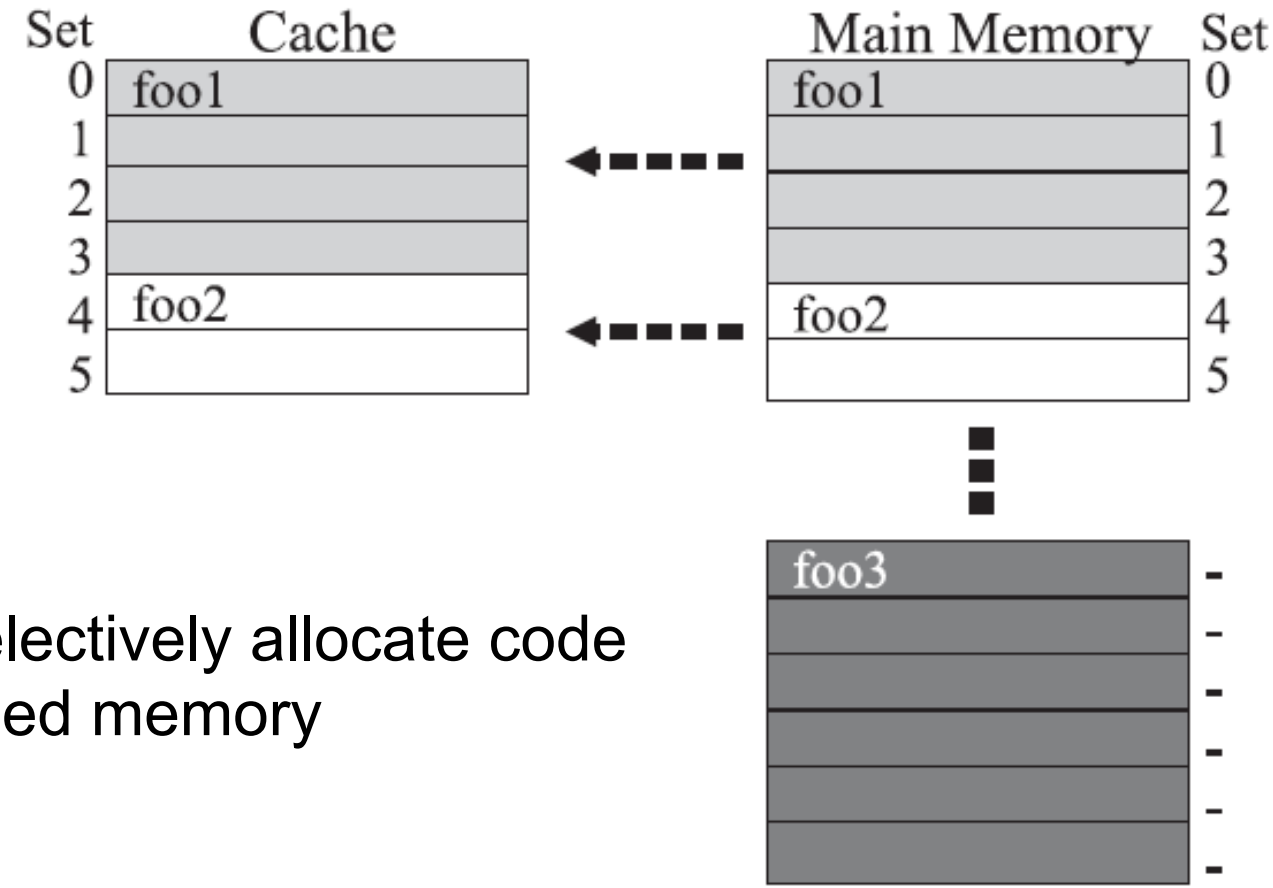
```
void foo1() {
 for(i=0; i<10; i++) {
  foo2();
  foo3();
 }
 ...
}
```





[S. Plazar: Memory-based Optimization Techniques for Real-Time Systems, PhD thesis, TU Dortmund, June 2012]

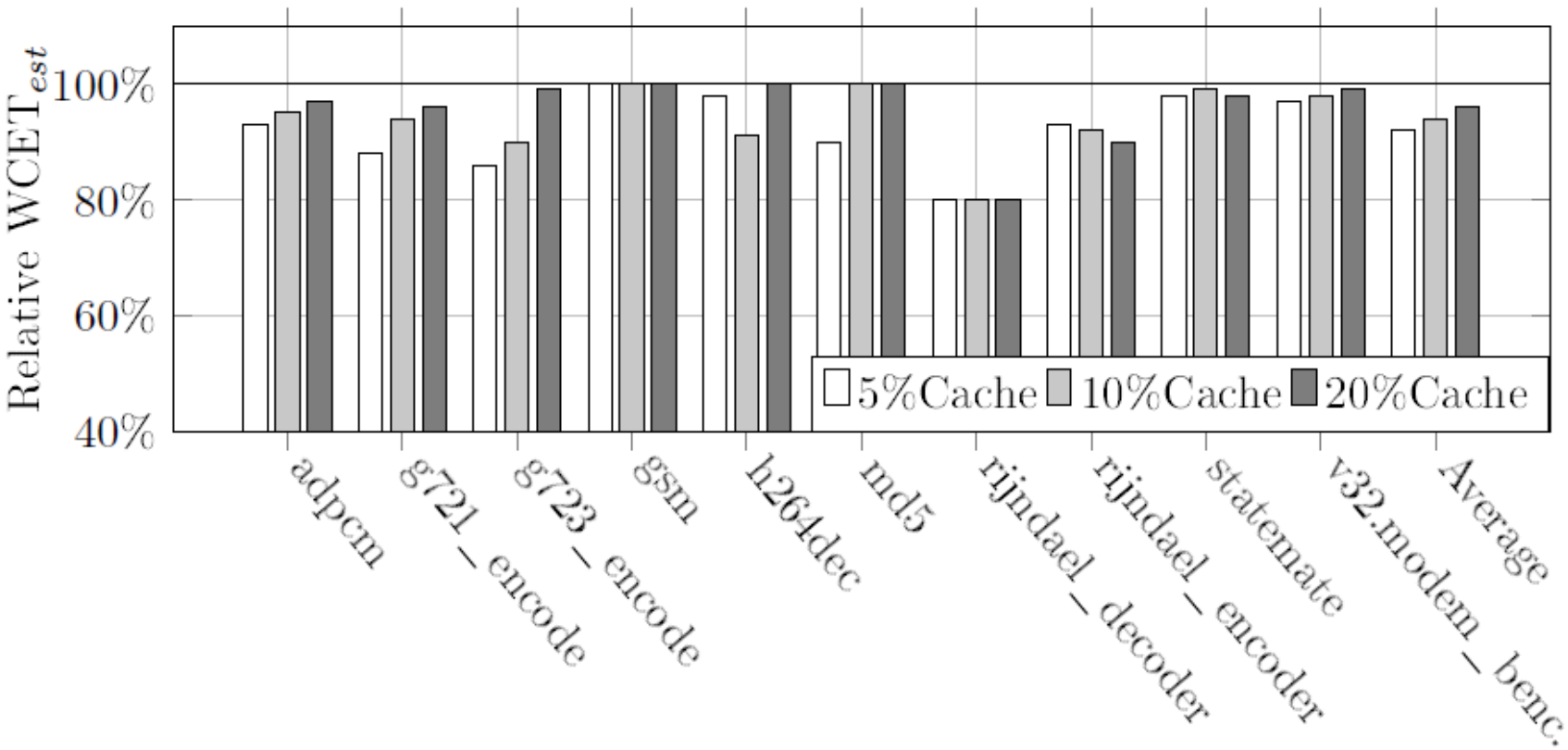
Avoiding cache thrashing



Key idea: selectively allocate code to uncached memory

S. Plazar: Memory-based Optimization Techniques for Real-Time Systems, PhD thesis, TU Dortmund, June 2012

Results

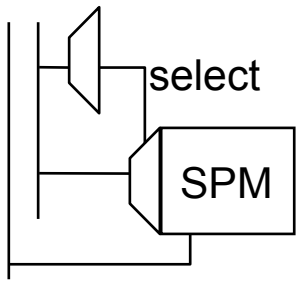


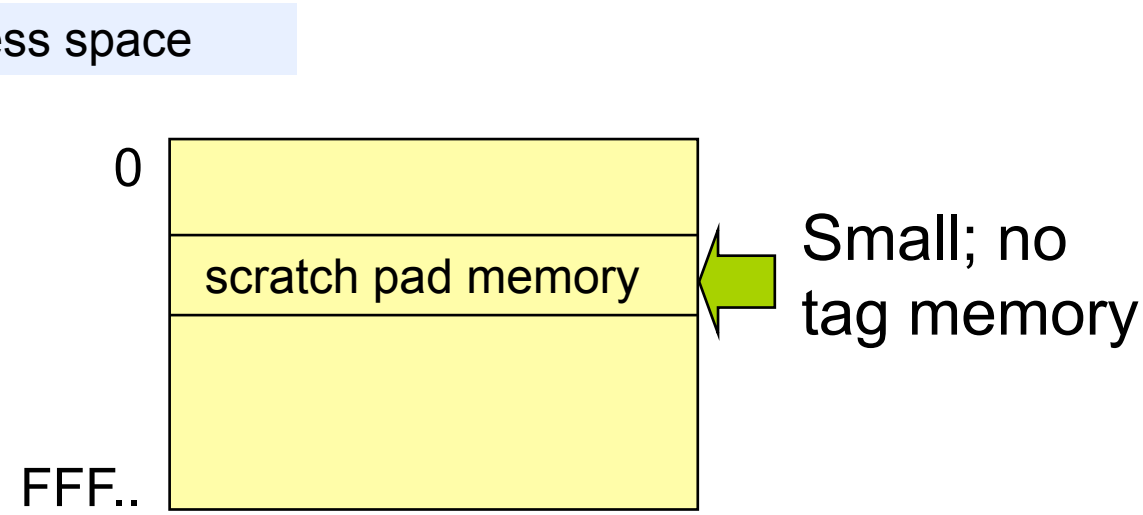
S. Plazar: Memory-based Optimization Techniques for Real-Time Systems, PhD thesis, TU Dortmund, June 2012

Scratch pad memories (SPM): Fast, energy-efficient, timing-predictable

SPMs are small, physically separate memories mapped into the address space;

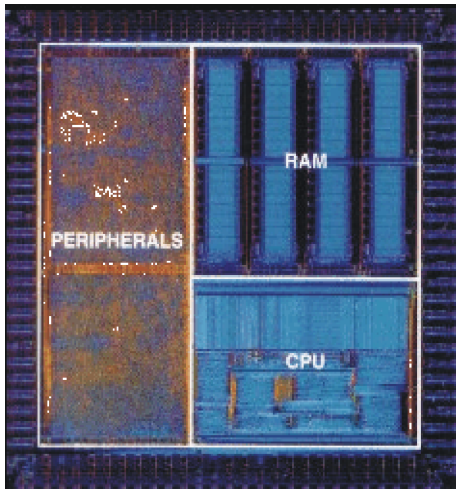
Selection is by an appropriate address decoder (simple!)





Example

ARM7TDMI cores, well-known for low power consumption



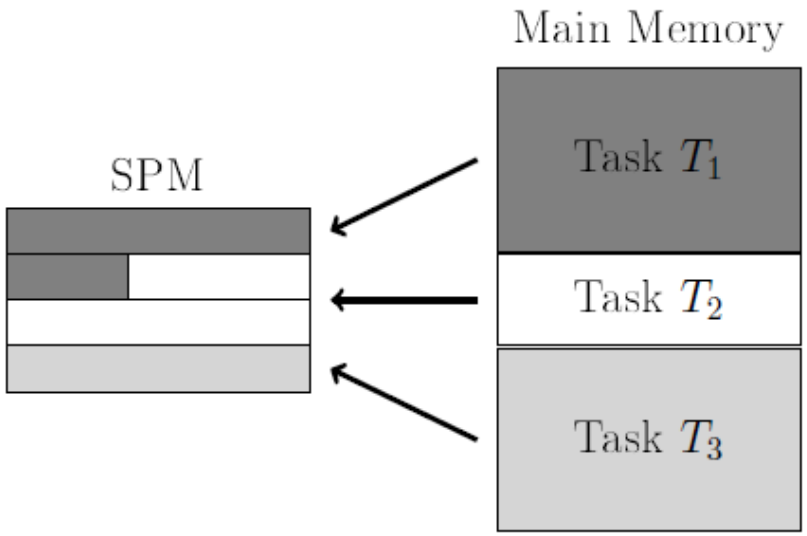
$WCET_{EST}$-aware SPM allocation

Setup

- Bosch Democar: Runnable: IgnitionSWCSync
  Part of task actuator
  activated every 90° of crankshaft angle
  ☞ time-critical
- $WCET_{EST}$-aware SPM allocation of program code by WCC, including fully automated $WCET_{EST}$ analyses using aiT and solution of the ILP for SPM allocation
- $WCET_{EST}$ reduced to about 50%, compared to gcc.

☞ PREDATOR project partners Bosch & Airbus interested in WCC

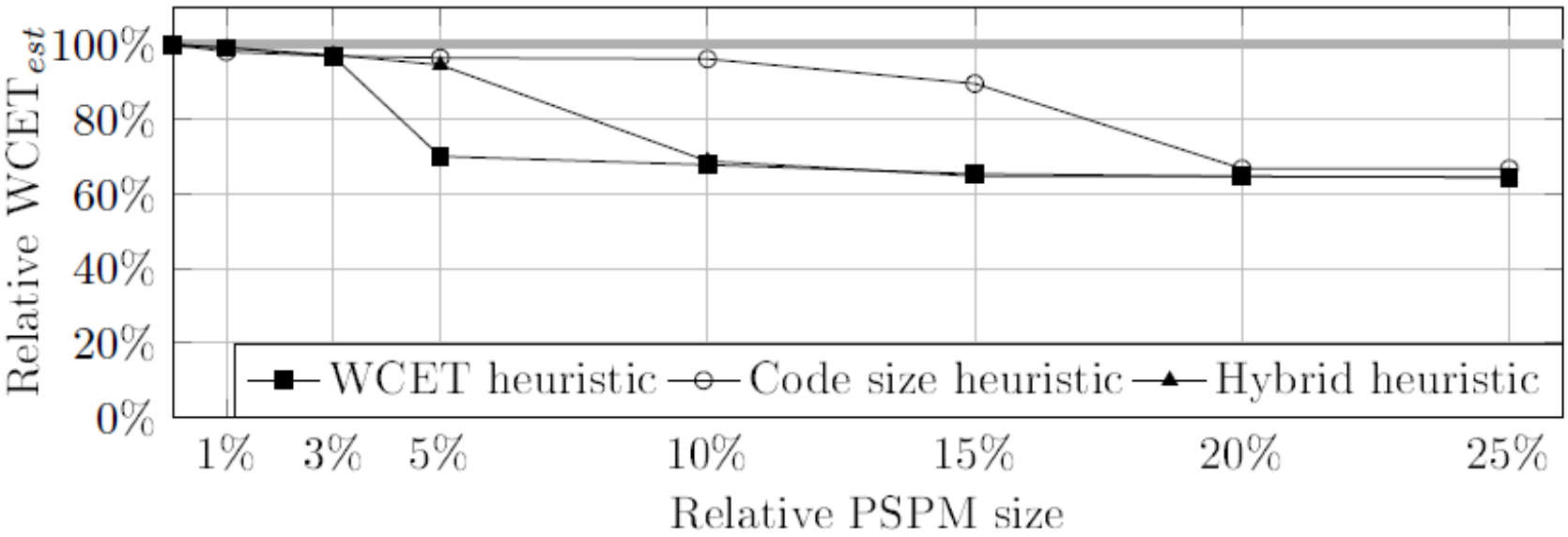
$WCET_{EST}$-oriented static SPM allocation for multi-task systems



Objective: $WCET_{EST}$

S. Plazar: Memory-based Optimization Techniques for Real-Time Systems, PhD thesis, TU Dortmund, June 2012

Results



S. Plazar: Memory-based Optimization Techniques for Real-Time Systems, PhD thesis, TU Dortmund, June 2012

Results (2)

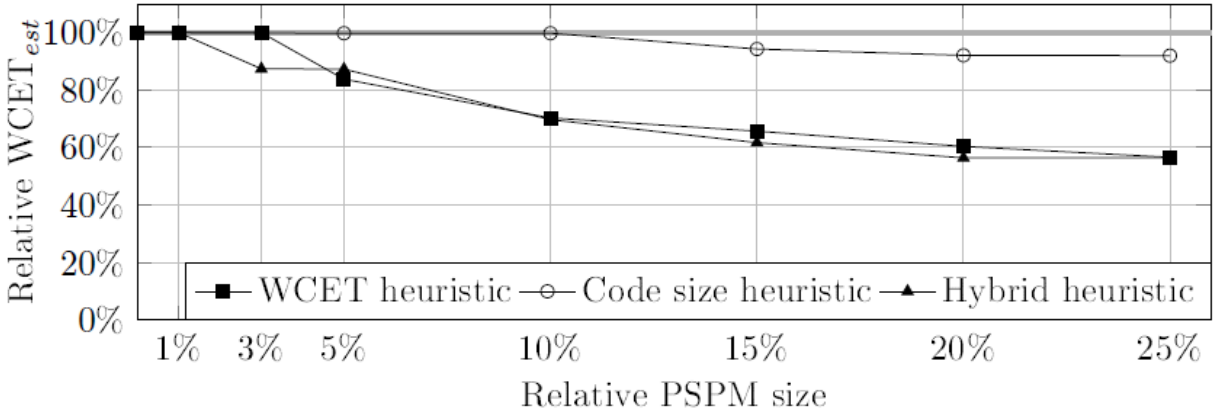


Figure 4.11: Optimized WCET$_{est}$ for multi-task PSPM allocation applied to $Set2$

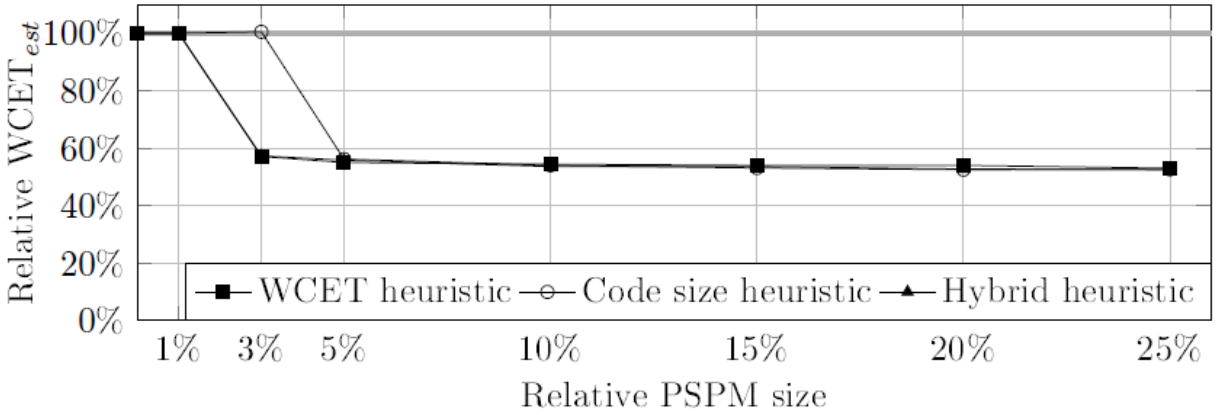


Figure 4.12: Optimized WCET$_{est}$ for multi-task PSPM allocation applied to $Set3$

S. Plazar: Memory-based Optimization Techniques for Real-Time Systems, PhD thesis, TU Dortmund, June 2012

Outline

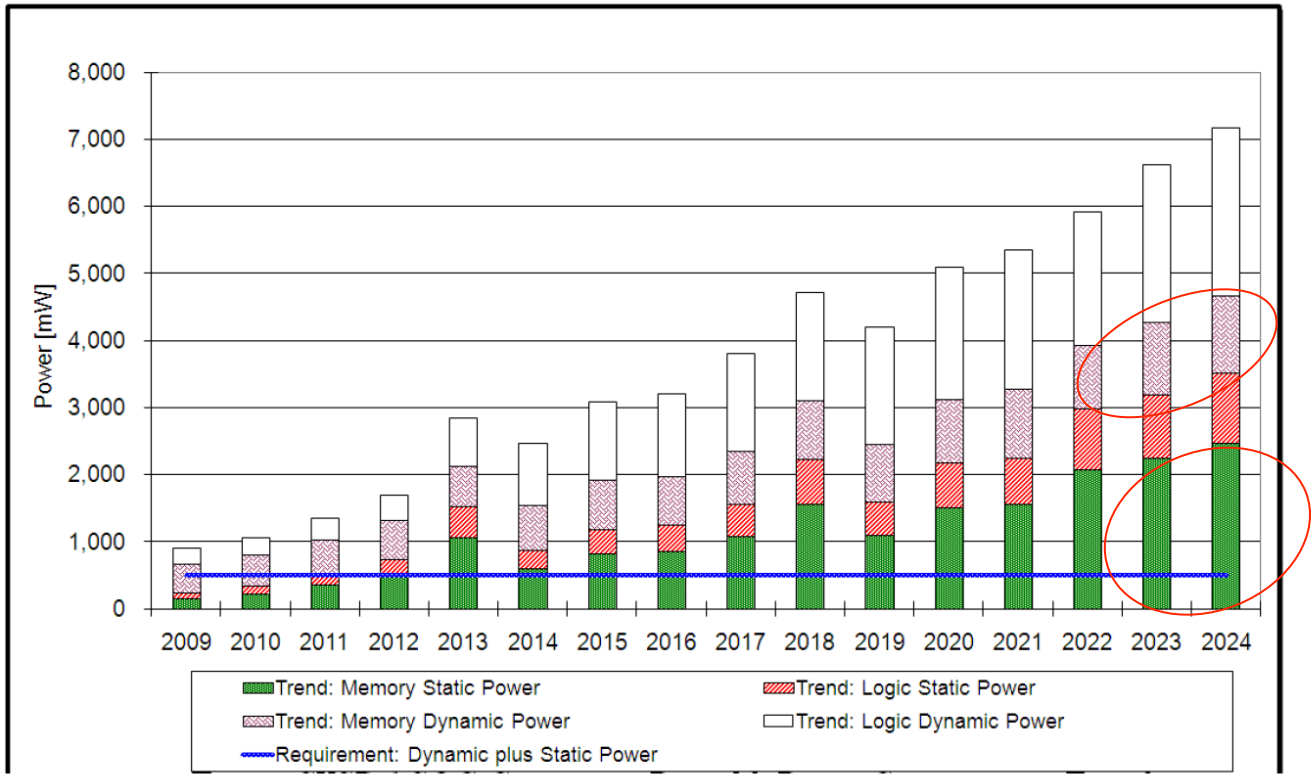
- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

Outline

- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

Where is the energy consumed?
- Consumer portable systems -

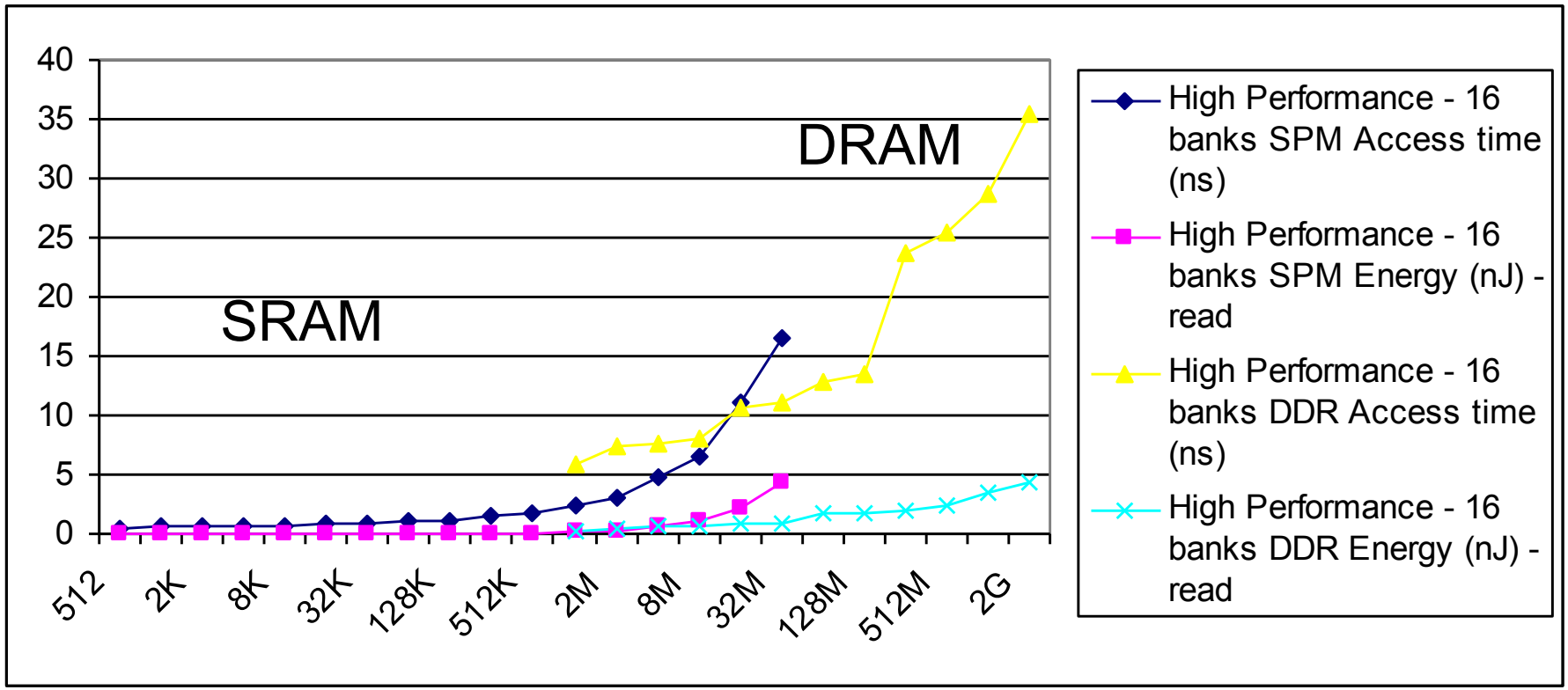
- According to *International Technology Roadmap for Semiconductors* (ITRS), 2010 update, [www.itrs.net]
- Current trends ☞ > constraint of 0.5-1 W for small mobiles.



- **It not just the logic, don't ignore memory!**

Energy consumption of memories

CACTI: Scratchpad (SRAM) vs. DRAM (DDR2):

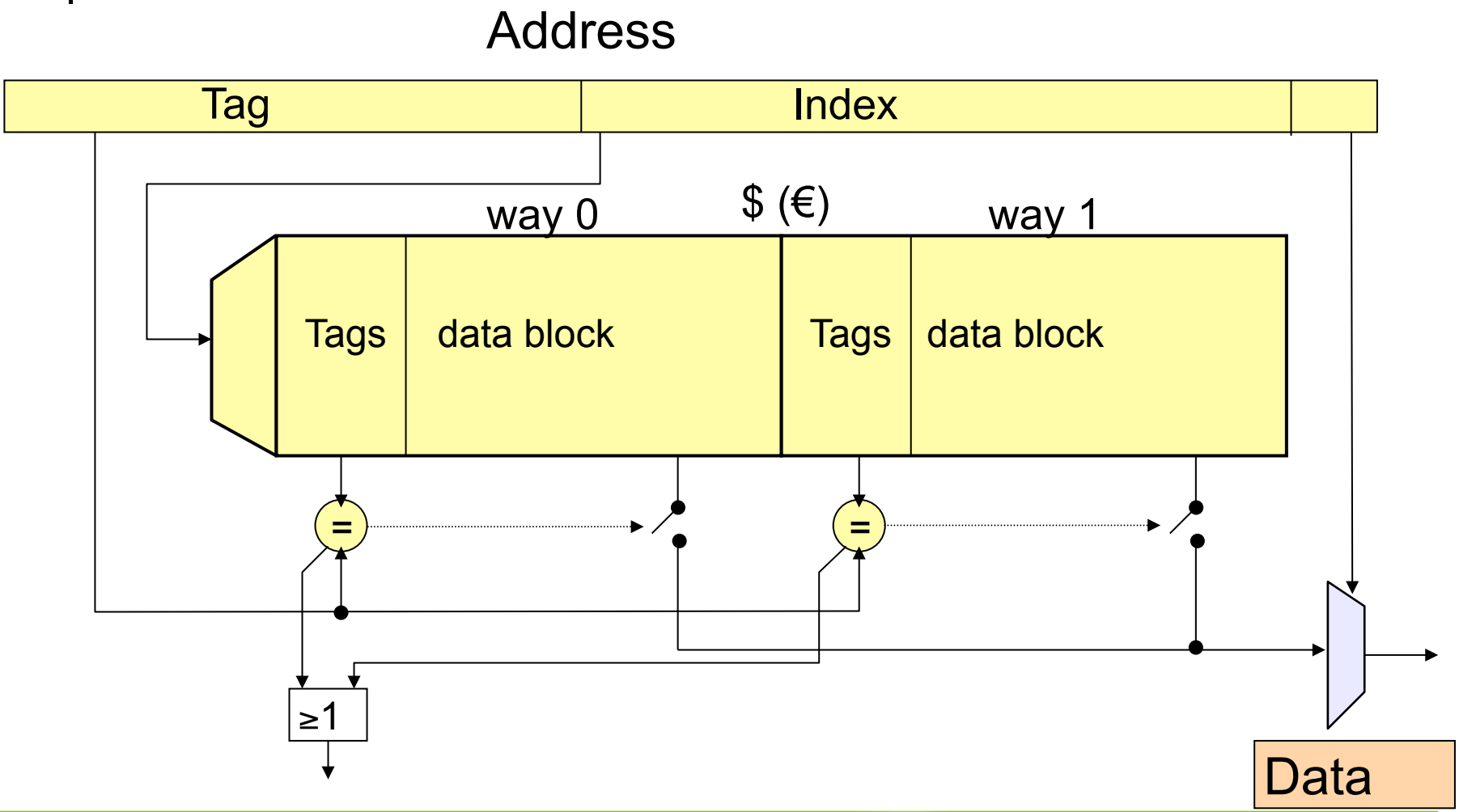


16 bit read; size in bytes;
65 nm for SRAM, 80 nm for DRAM

Source: Olivera Jovanovic, TU Dortmund, 2011

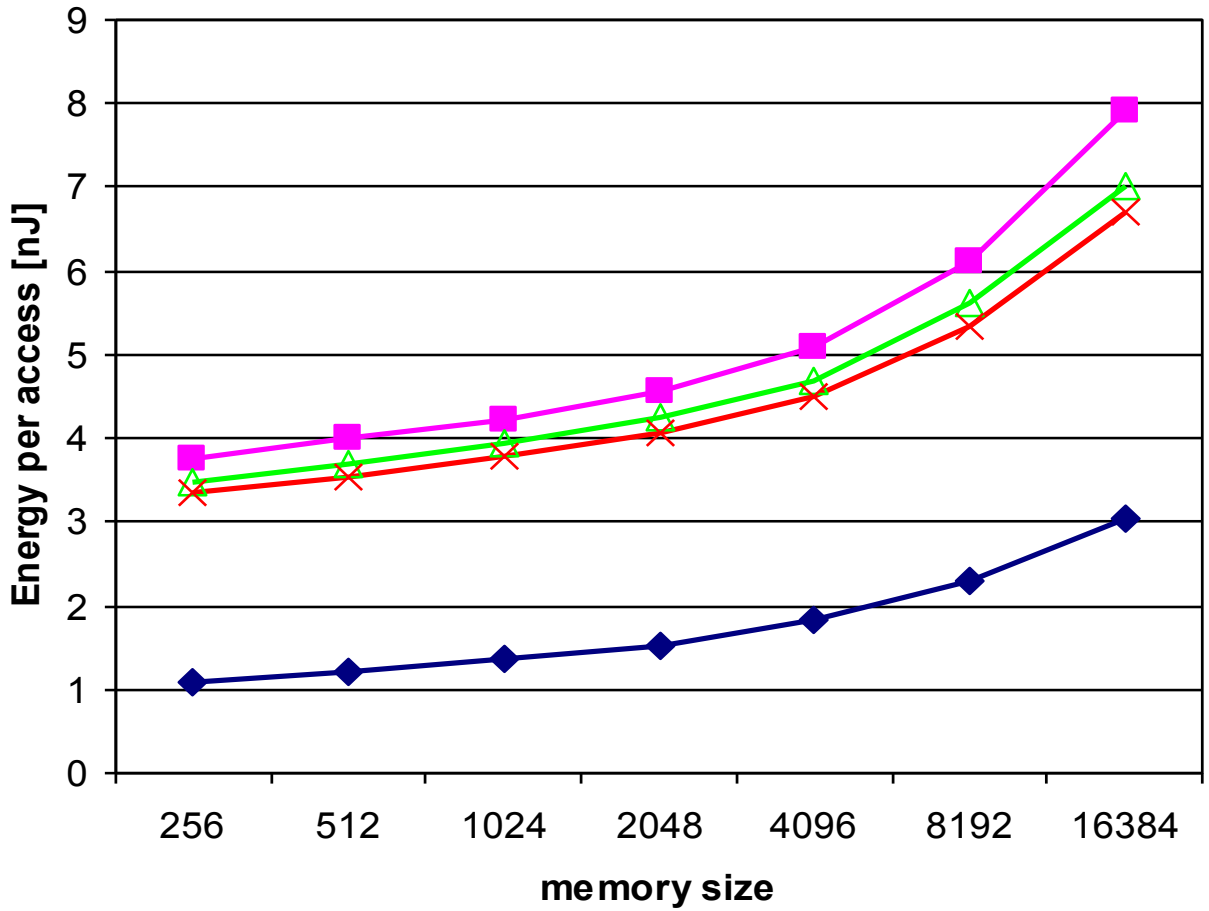
Pre-requisite: **Set-associative cache**
***n*-way cache**

|Set| = 2



Why not use (a) cache ?

Energy consumption in tags, comparators and muxes is significant.



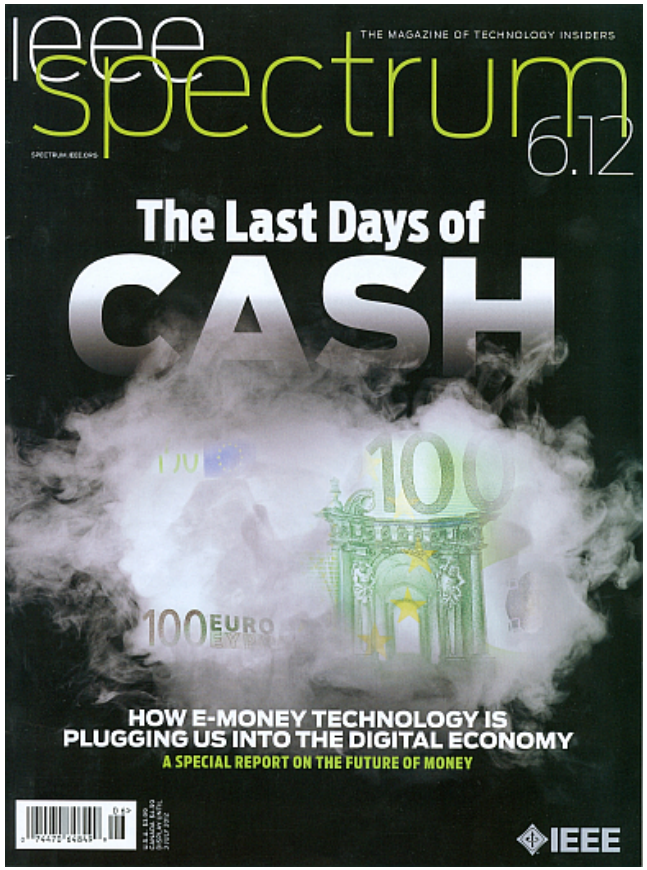
[R. Banakar, S. Steinke, B.-S. Lee, M. Balakrishnan, P. Marwedel. Scratchpad Memory: Design Alternative for Cache On-chip memory in Embedded Systems, *Intern. Workshop on Hardware/ Software Codesign* (CODES), 2002]

- Scratch pad
- Cache, 2way, 4GB space
- Cache, 2way, 16 MB space
- Cache, 2way, 1 MB space

☞ Why not get rid of cache?

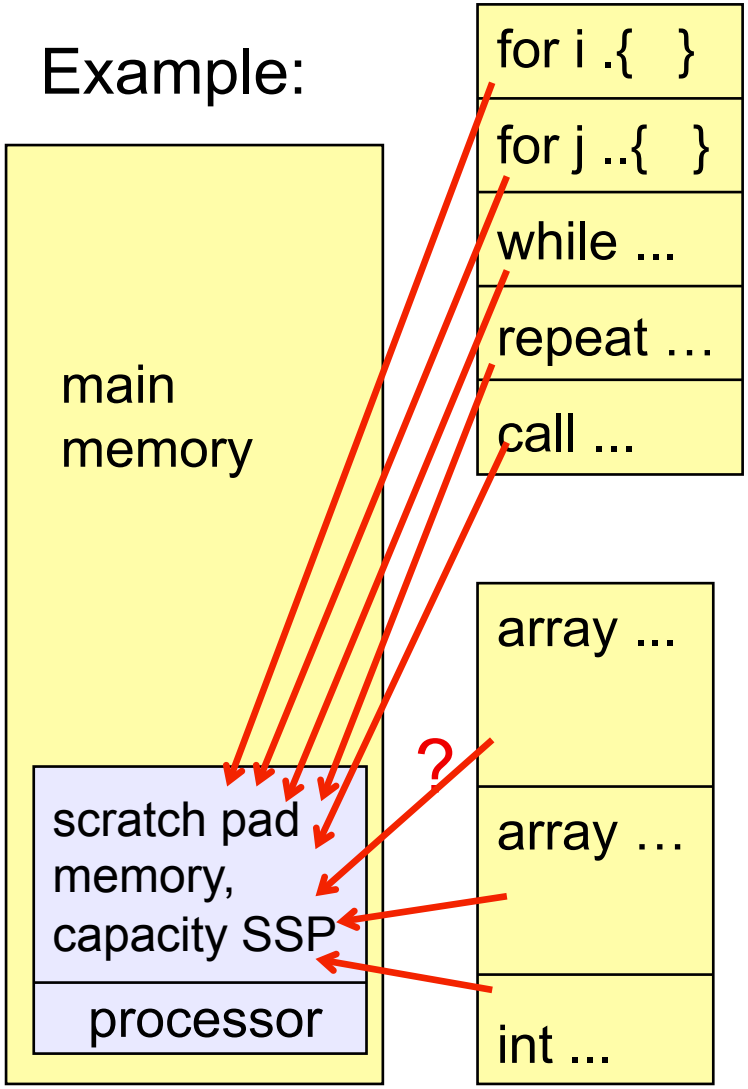
Less seriously …

Some people got rid of cache already!



Migration of data & instructions, global optimization model

Example:



Which memory object (array, loop, etc.) to be stored in SPM?

Non-overlaying ("static") allocation:

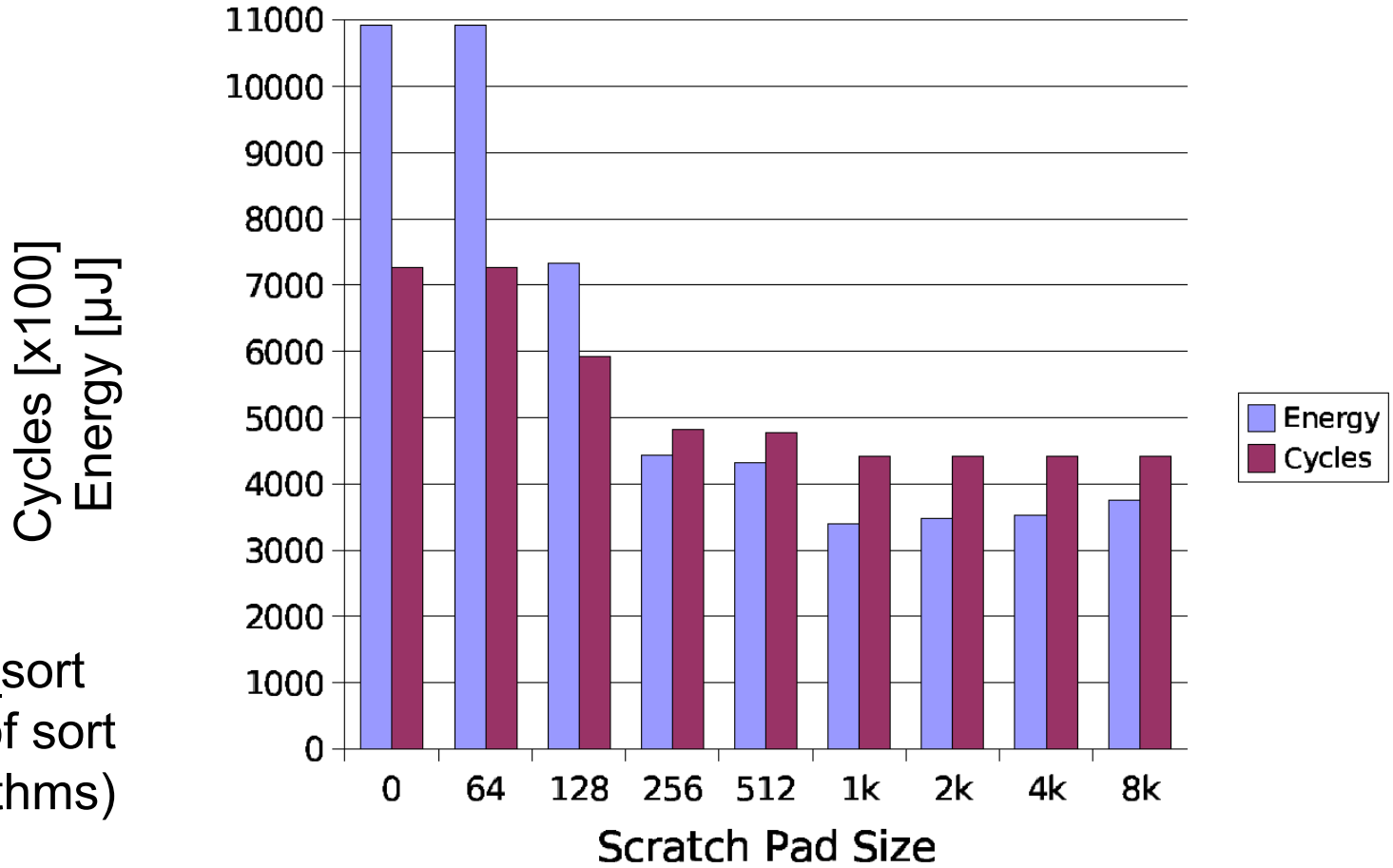
Gain g_k and size s_k for each object k. Maximise gain $G = \Sigma g_k$, respecting size of SPM $SSP \geq \Sigma\, s_k$.

Solution: knapsack algorithm.

Overlaying ("dynamic") allocation:

Moving objects back and forth

Reduction in energy and average run-time for migrating functions and variables

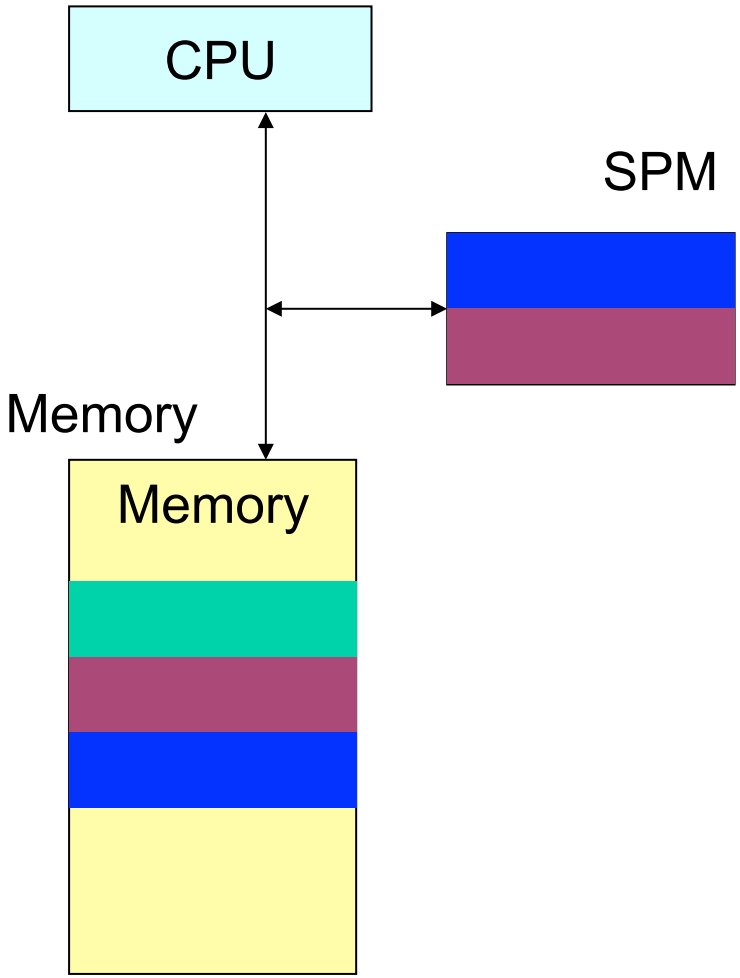


Multi_sort (mix of sort algorithms)

Measured processor / external memory energy + CACTI values for SPM (combined model)

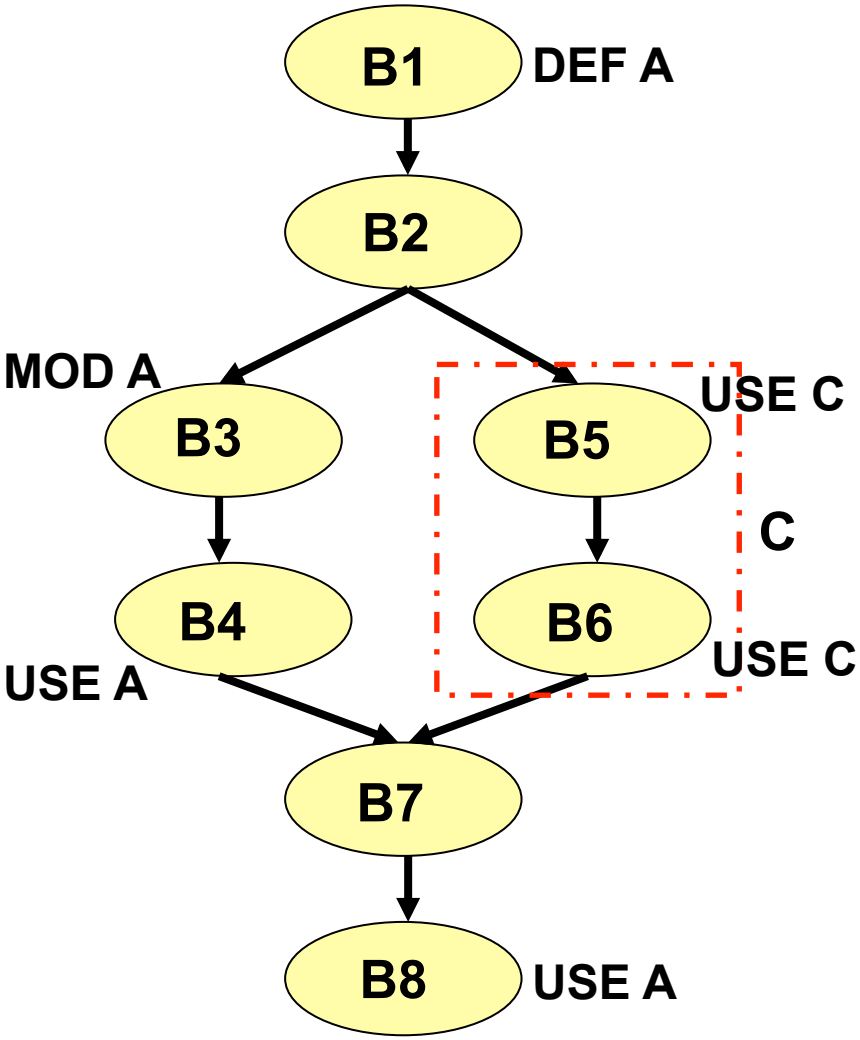
Numbers will change with technology, algorithms remain unchanged.

Non-overlaying allocation problematic for multiple hot spots ☞Overlaying allocation



- Effectively results in a kind of **compiler-controlled overlays** for SPM
- Address assignment within SPM required

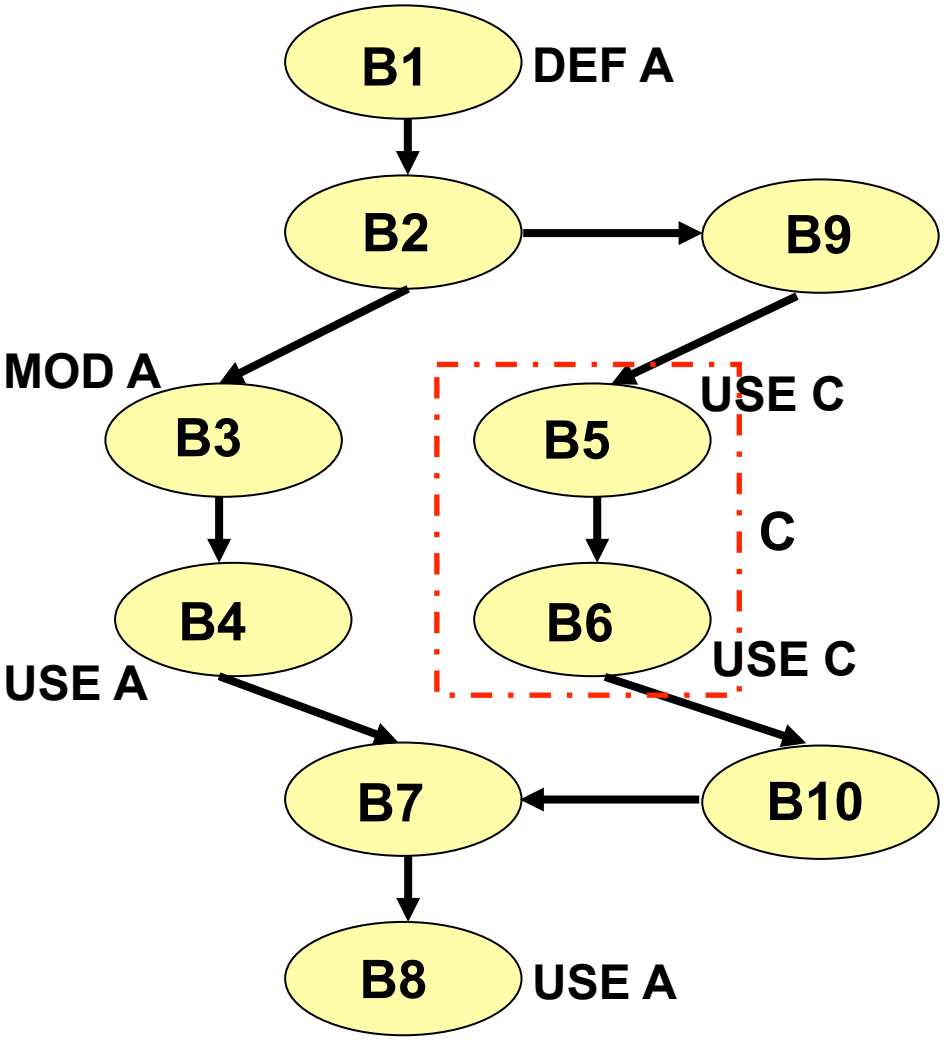
Overlaying allocation by Verma et al. (1)



Based on control flow graph.

[M.Verma, P.Marwedel: Dynamic Overlay of Scratchpad Memory for Energy Minimization, *ISSS*, 2004]

Overlaying allocation by Verma et al. (2)



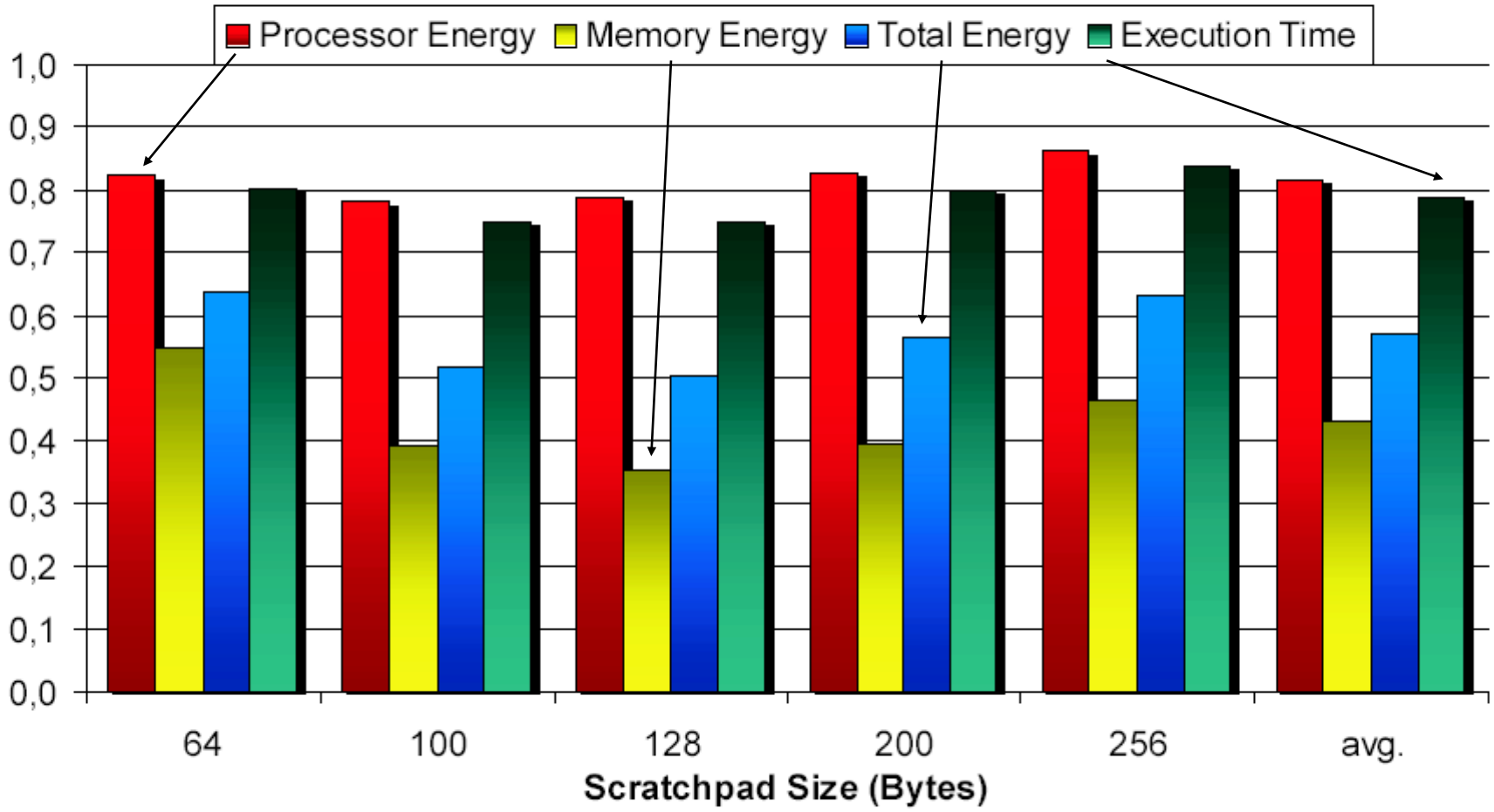

```
SPILL_STORE(A);
SPILL_LOAD(C);
```

Global set of ILP equations reflects cost/benefit relations of potential copy points

```
SPILL_LOAD(A);
```

Code handled like data

Runtime/energy reduction with respect to non-overlaying ("static") allocation

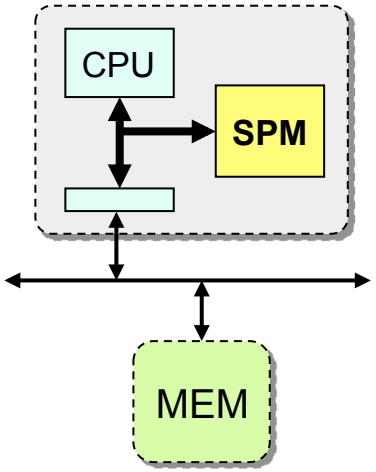


Dynamic set of multiple applications

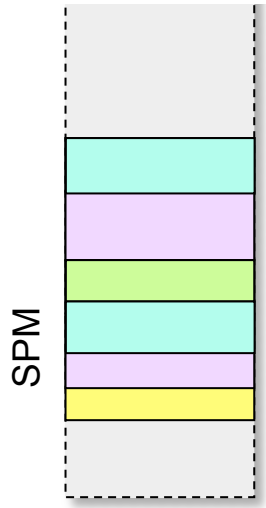
Compile-time partitioning of SPM no longer feasible

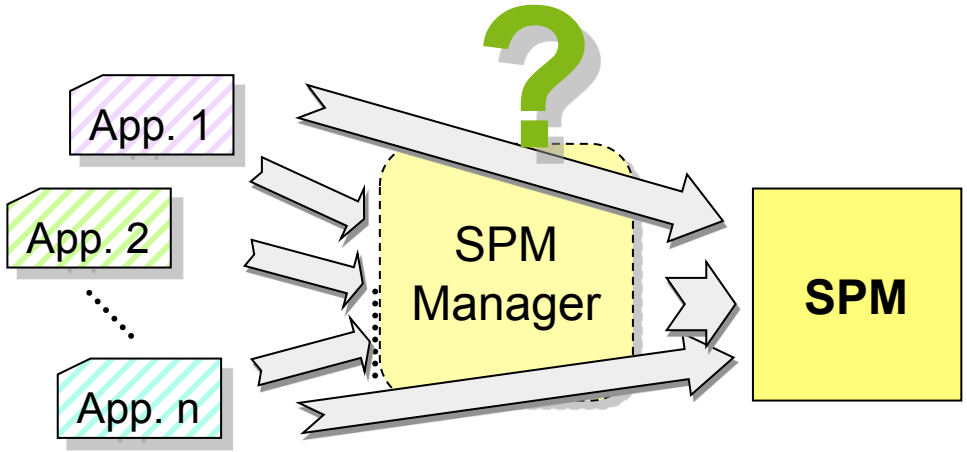
☞ Introduction of SPM-manager

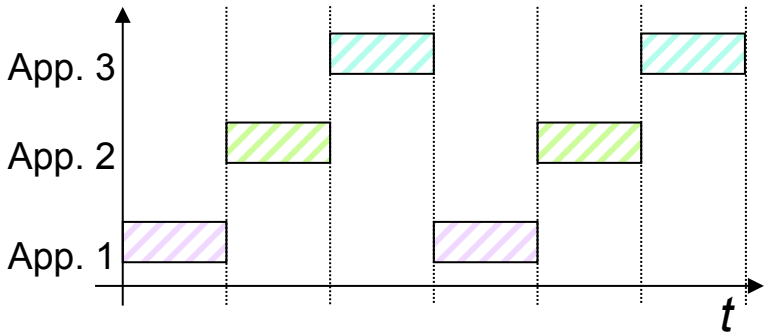
- Runtime decisions, but compile-time supported



Address space:







[R. Pyka, Ch. Faßbach, M. Verma, H. Falk, P. Marwedel: Operating system integrated energy aware scratchpad allocation strategies for multi-process applications, *SCOPES*, 2007]

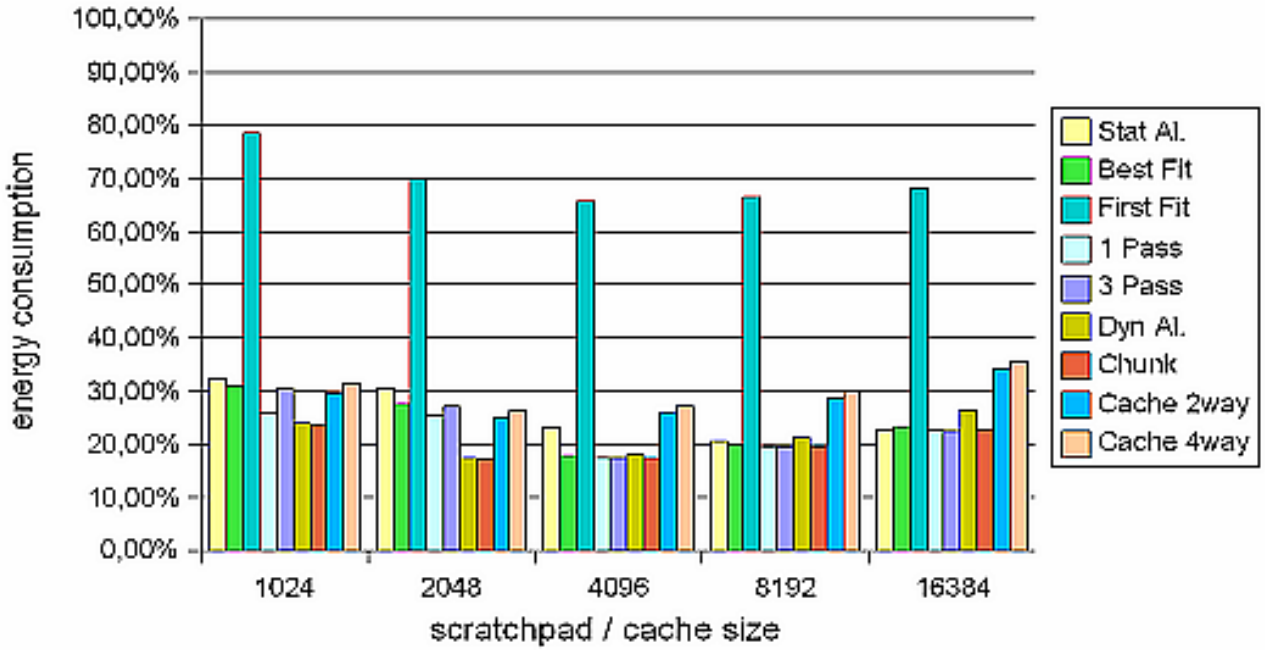
Comparison of SPMM to Caches for SORT

- Baseline: Main memory only
- SPMM peak energy reduction by 83% at 4k Bytes scratchpad
- Cache peak: 75% at 2k 2-way cache

- SPMM capable of outperforming caches
- OS and libraries are not considered yet

Chunk allocation results:

SPM Size	Δ 4-way
1024	74,81%
2048	65,35%
4096	64,39%
8192	65,64%
16384	63,73%

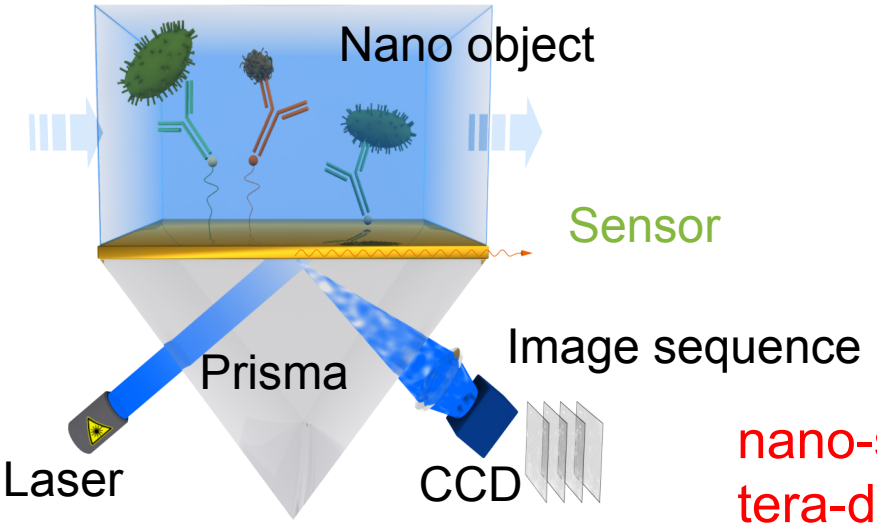


Outline

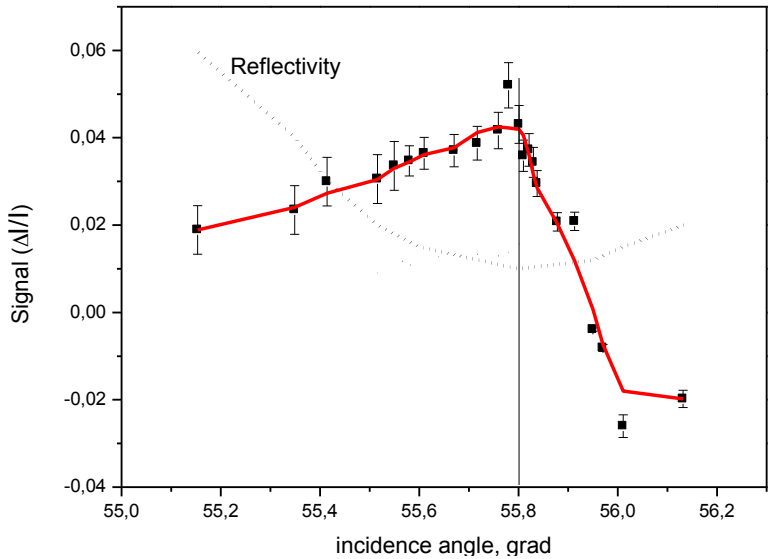
- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

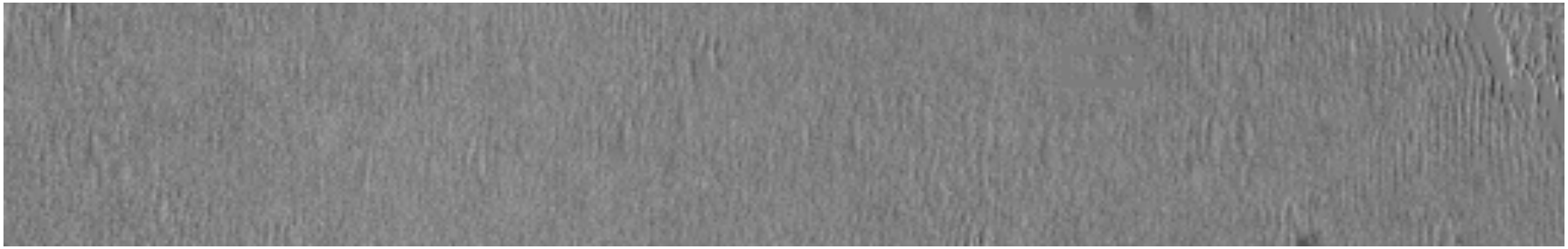
Energy efficient (bio-) virus detection

Detection of nano objects using plasmon effect
(Optical effect of objects ≪ wavelength of light)



nano-size/
tera-data





Detected200nm.avi

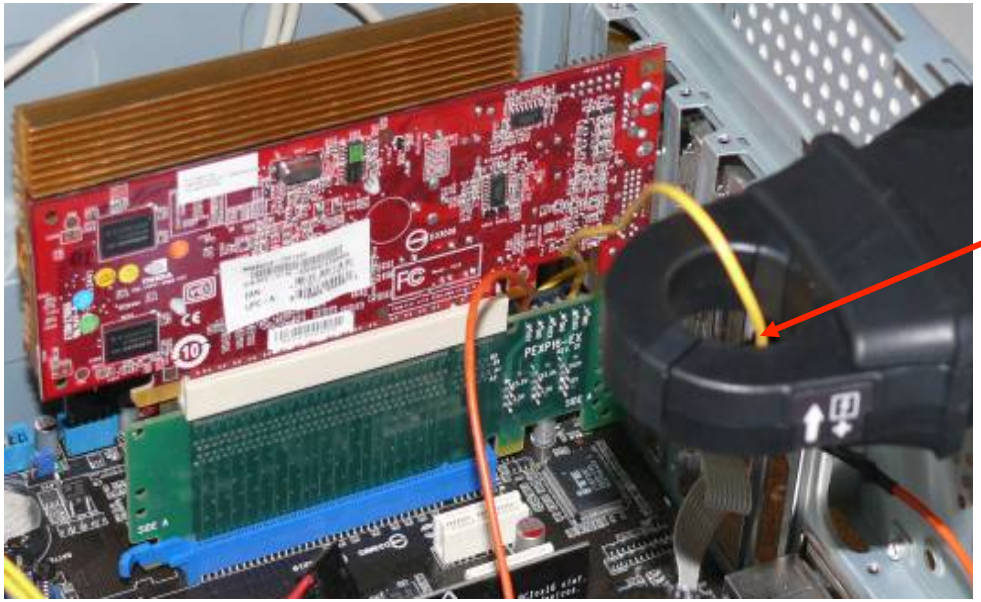
Timeliness in (bio-) virus detection (2)

- Low-end Mobile GPU meets realtime constraint up to 1024x256 pixels

Target: 30 frames per second at 1024x256 pixels

Frame Rate (fps)	Image Size			
	1024x128	1024x256	1024x512	1024x1024
CPU: i7-2600	23	20.1	16.1	11.5
CPU: i7-620M (mobile)	16	8.5	4.6	2.4
GPU: 3100M (mobile)	60	32.6	16	7.8
GPU: GTX 480	60	60	60	26.8
GPU: GTX 560Ti	60	60	60	40.3

Timeliness in (bio-) virus detection (3)



current clamp

Energy per frame CPU	3.26 J	5.84 J	10.52 J	Reduced to
Energy per frame GPU	0.93 J	1.56 J	2.76 J	avg 27%

C. Timm, A. Gelenberg, P. Marwedel, F. Weichert: Energy Considerations within the Integration of General Purpose GPUs in Embedded Systems. Intern. Conf. on Advances in Distributed and Parallel Computing, 2010

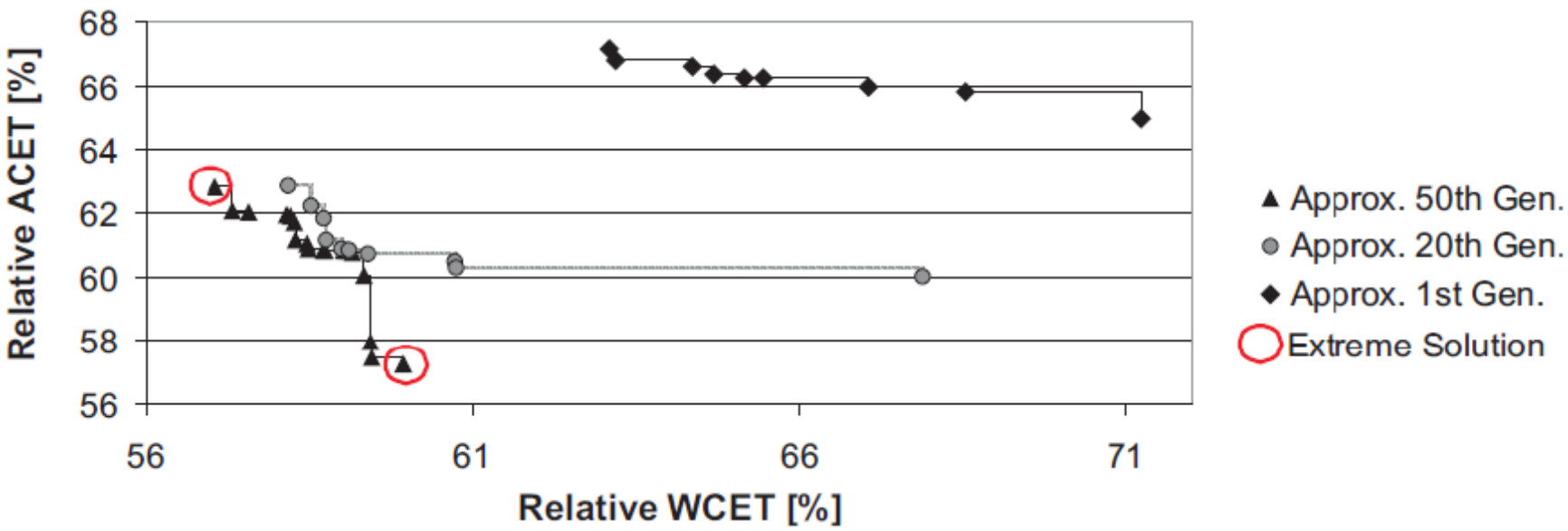
C. Timm, F. Weichert, P. Marwedel, H. Müller: Design Space Exploration Towards a Realtime and Energy-Aware GPGPU-based Analysis of Biosensor Data. Computer Science - Research and Development, ENA-HPC, 2011

Outline

- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

Tradeoff between average- and worst-case execution time for WCC

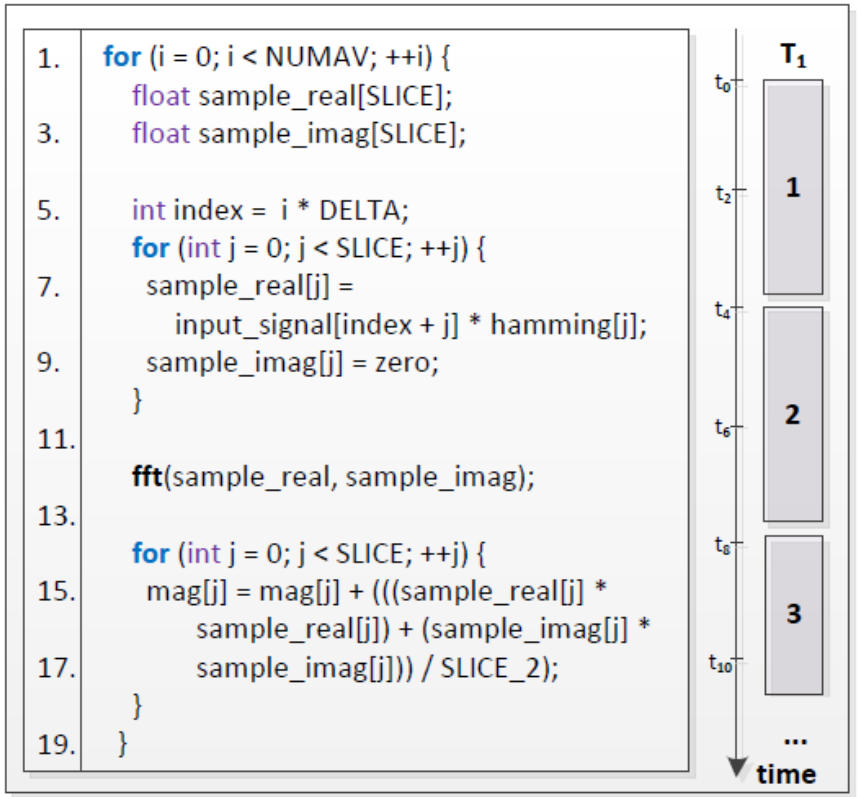
Using genetic algorithm to find good combination of standard optimizations



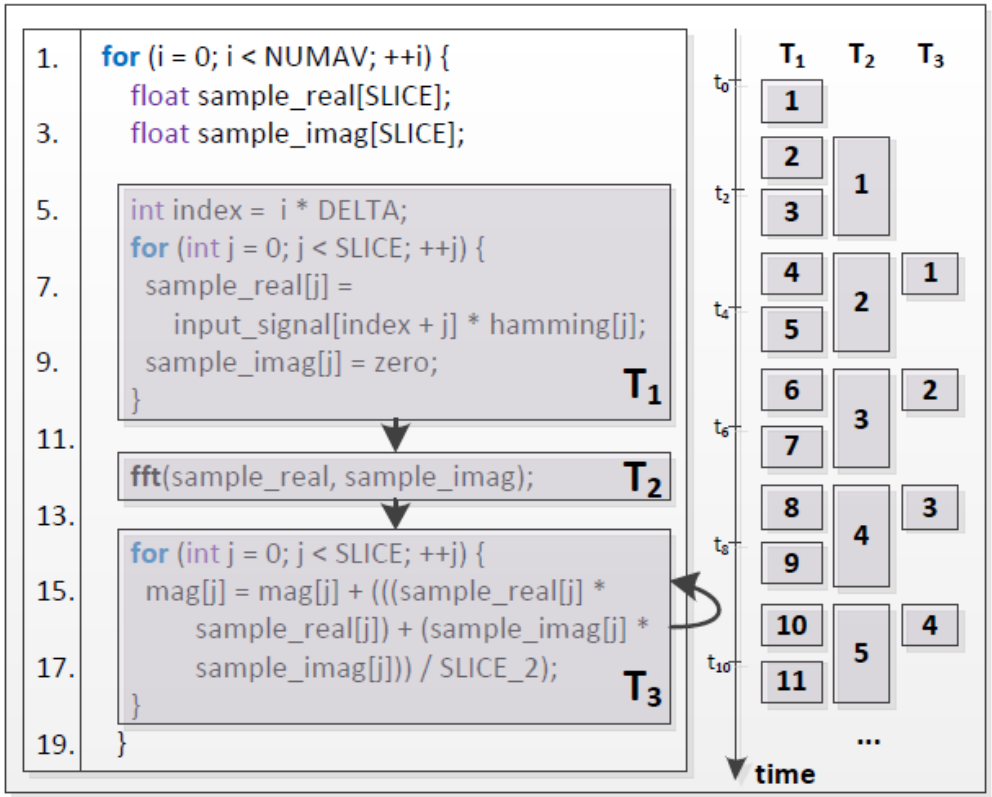
☞Achieved WCET and ACET reductions are very similar

Automatic parallelization for limited number of known cores (1)

Addressing the multi-core challenge …



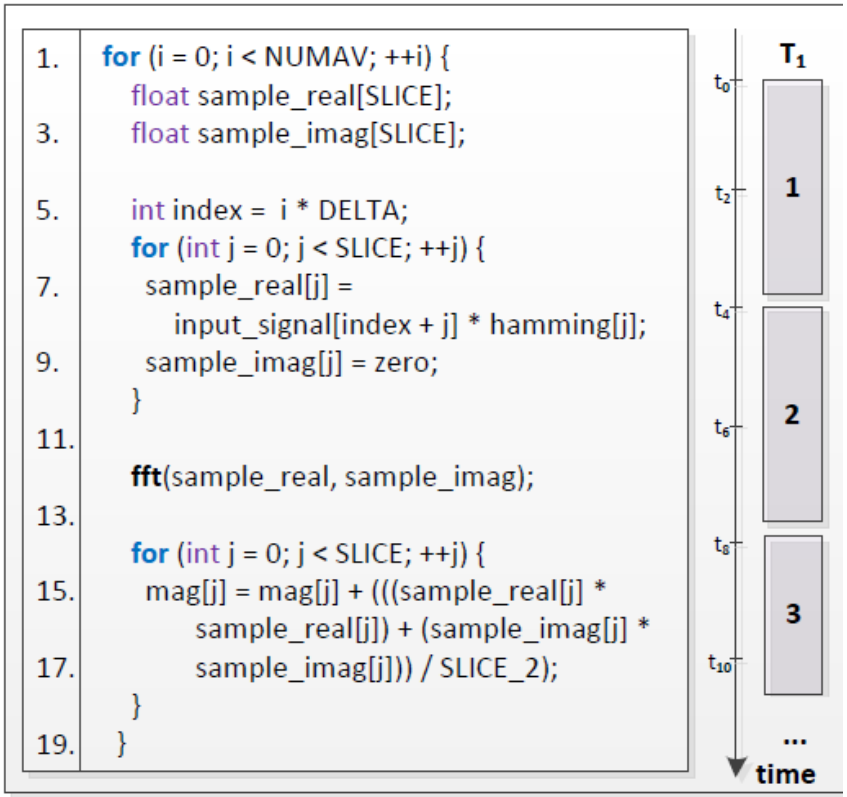
(a) Sequential code



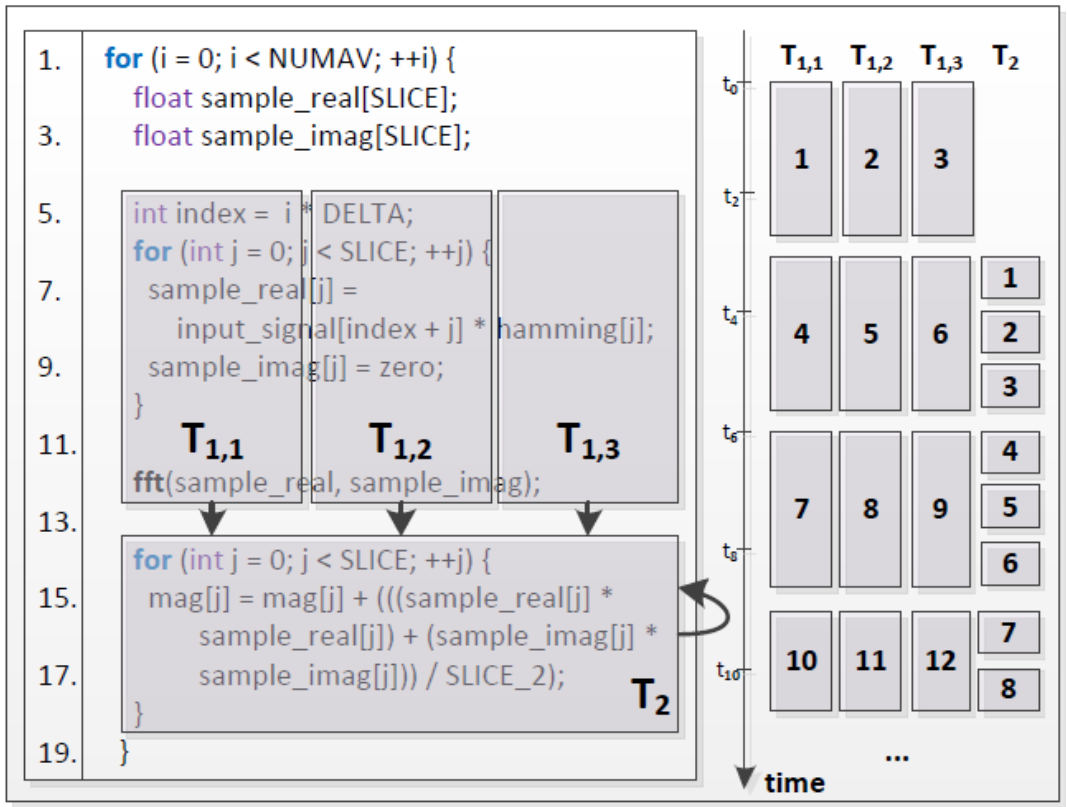
(b) Horizontal loop split

Automatic parallelization for limited number of known cores (2)

Exploiting properties of known platform

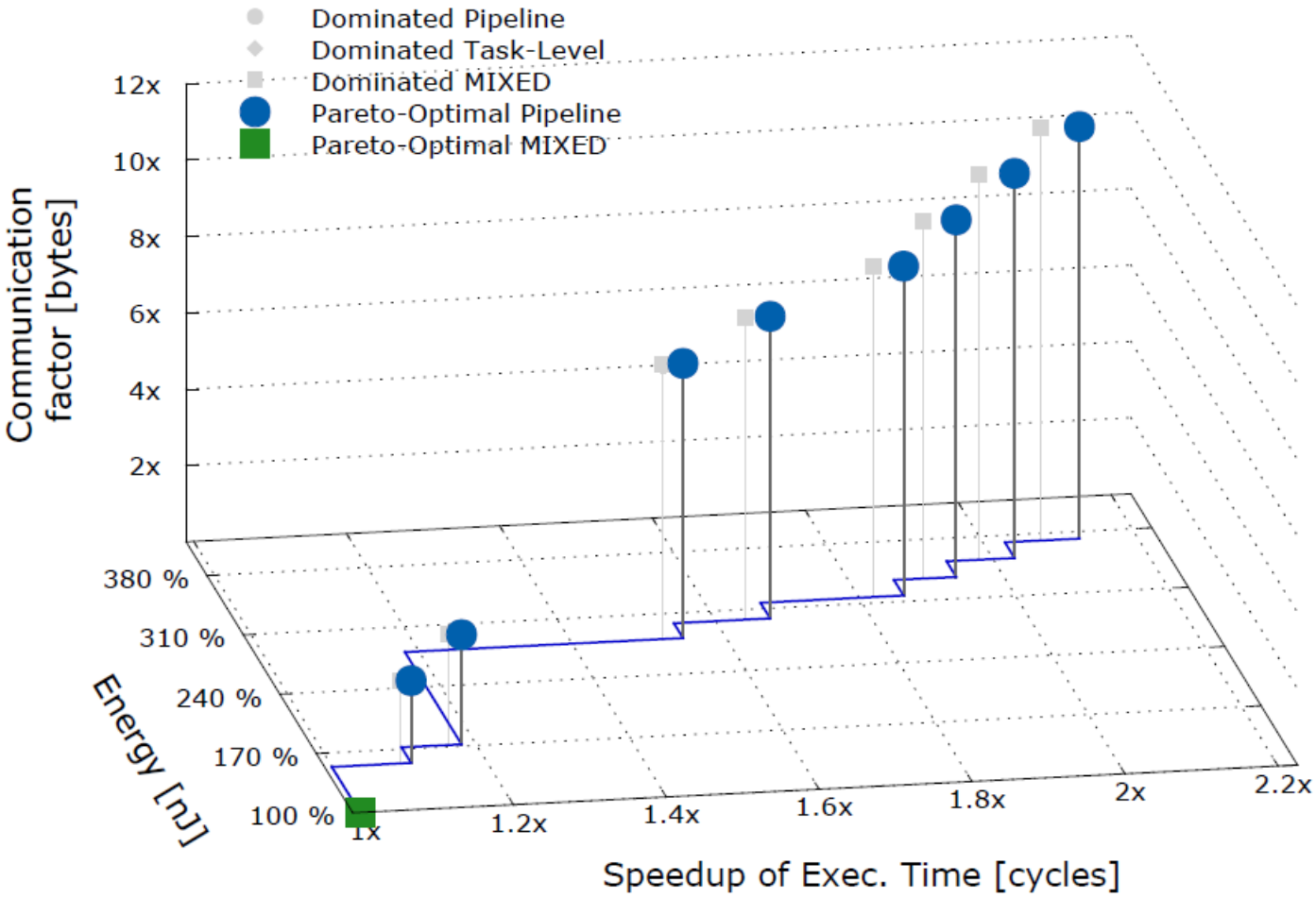


(a) Sequential code



(c) Horizontal and vertical loop split

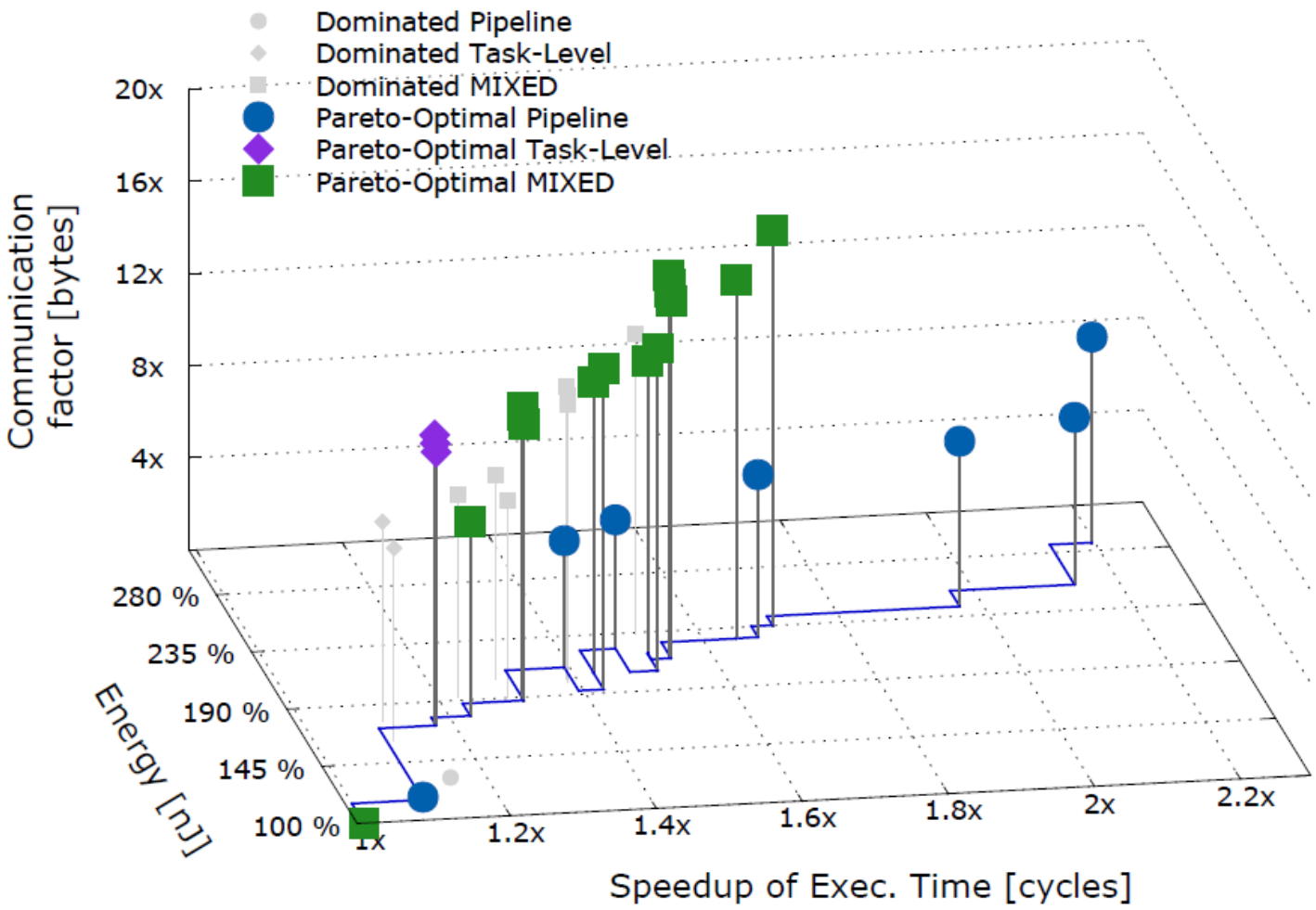
Multi-objective based parallelization of sequential programs (1)



(b) edge detect

D. Cordes, M. Engel, P. Marwedel, O. Neugebauer: Automatic Extraction of Multi-Objective Aware Pipeline Parallelism Using Genetic Algorithms, CODES/ISSS, 2012

Multi-objective based parallelization of sequential programs (2)



(a) spectral

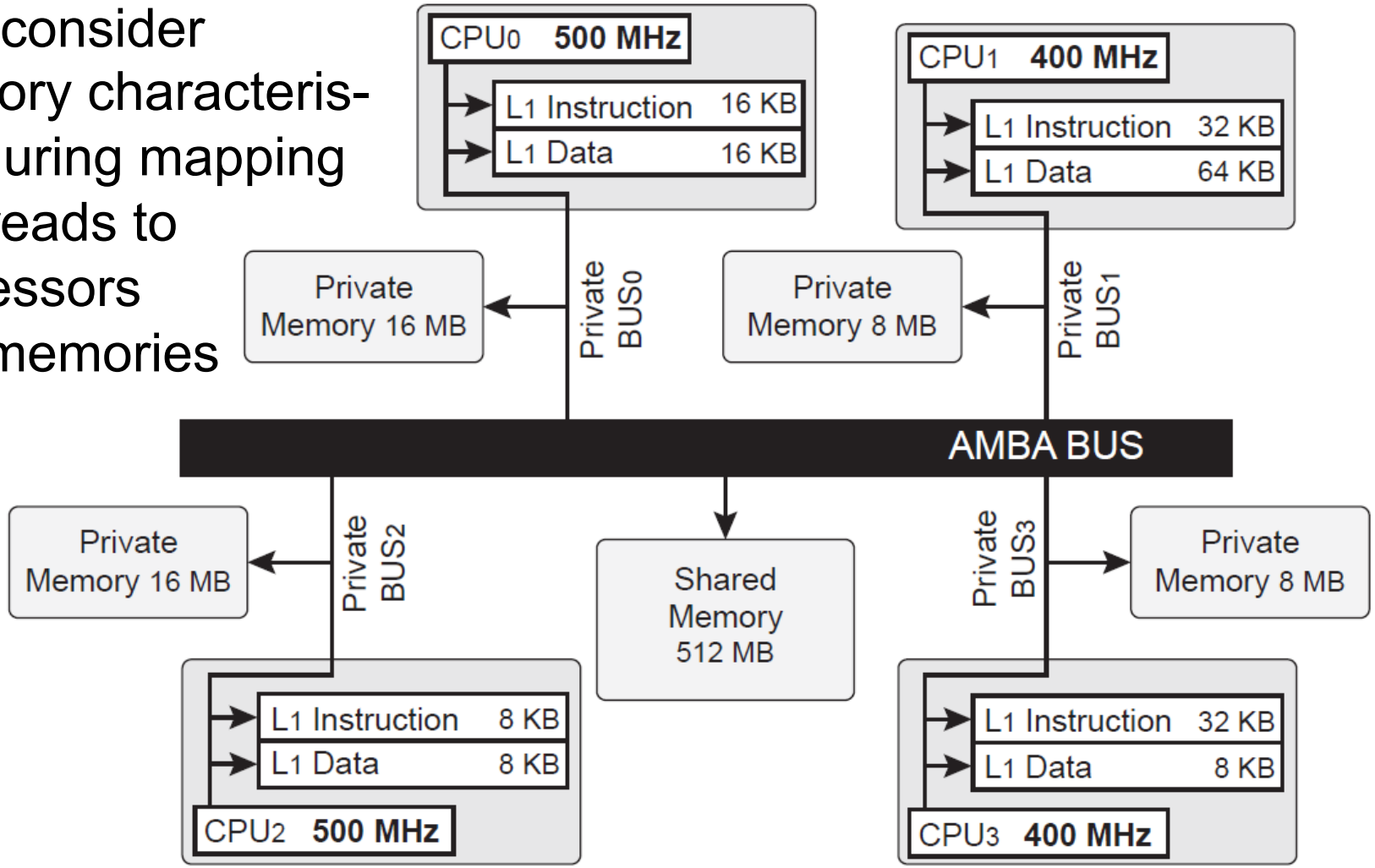
D. Cordes, M. Engel, P. Marwedel, O. Neugebauer: Automatic Extraction of Multi-Objective Aware Pipeline Parallelism Using Genetic Algorithms, CODES/ISSS, 2012

Outline

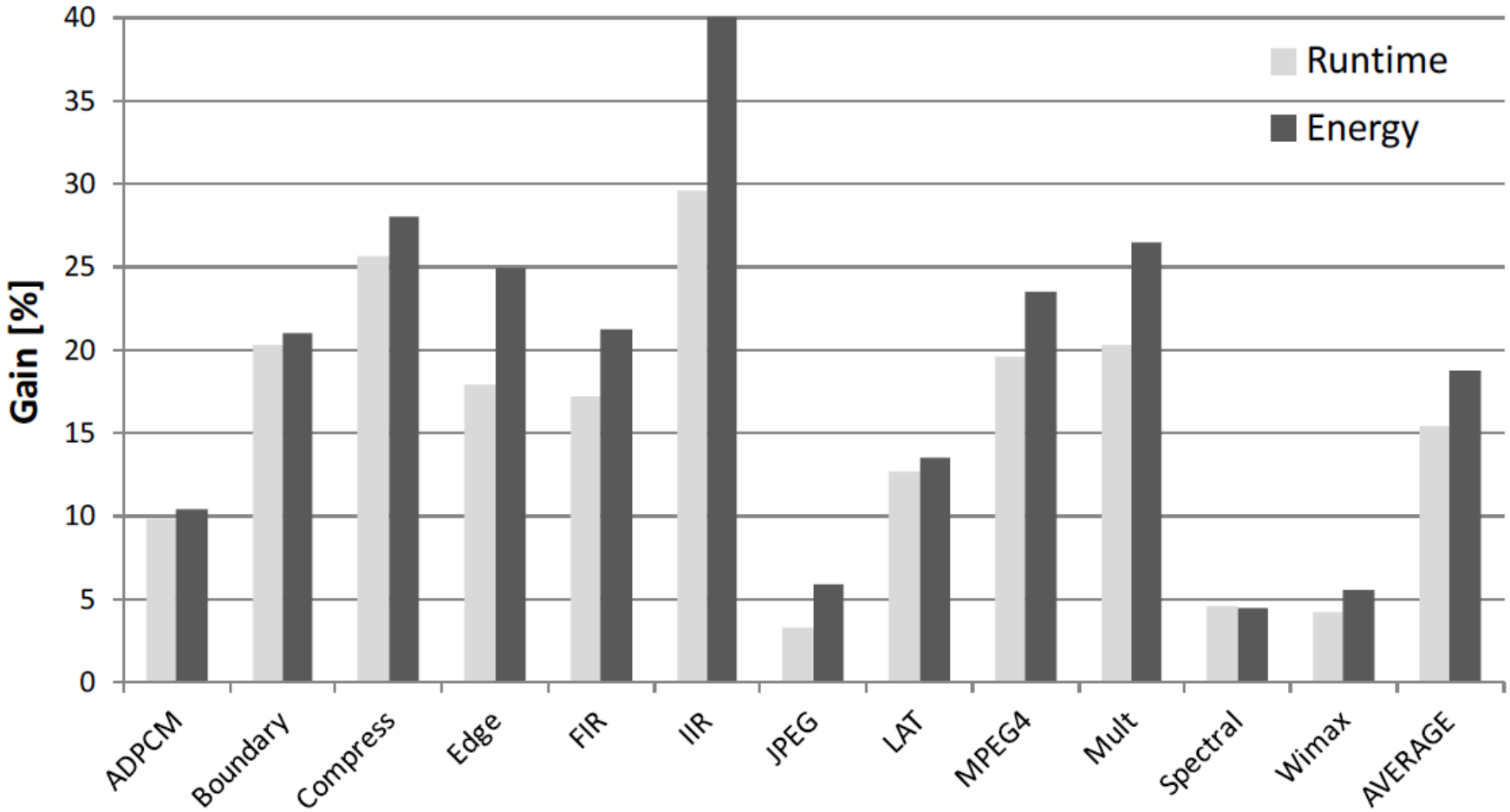
- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

MAMOT: A tool for memory mapping

Aim: consider memory characteristics during mapping of threads to processors and memories



Gains resulting from the use of memory information



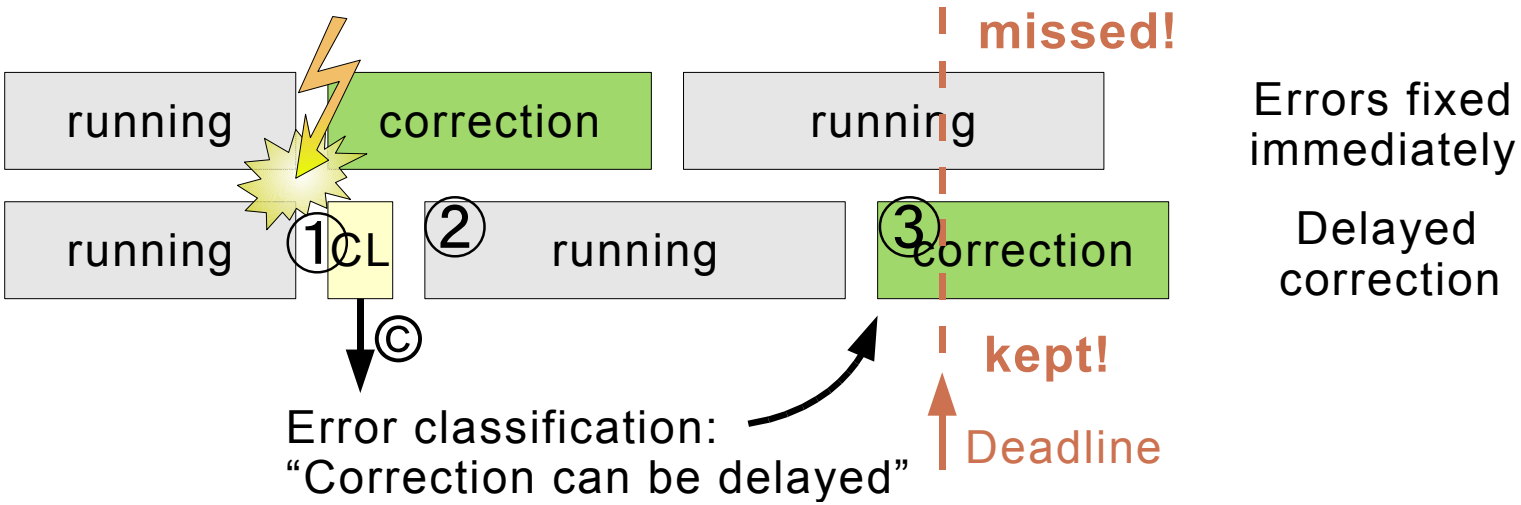
Olivera Jovanovic, Peter Marwedel, Iuliana Bacivarov, Lothar Thiele: MAMOT: Memory-Aware Mapping Tool for MPSoC, 15th Euromicro Conference on Digital System Design (DSD 2012) 2012

Outline

- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

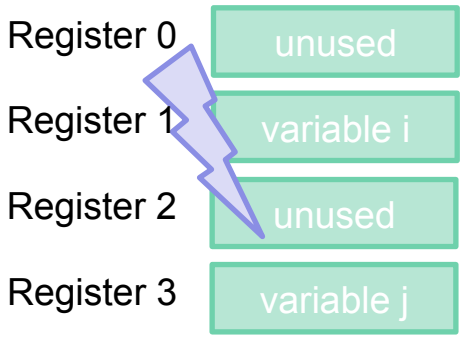
Tradeoff between reliability and timeliness

- Traditional error correction requires resources, including time
- Conflicting with timing requirements of real-time systems



Flexible Error Correction

- In some cases, errors have no effect
- In some cases, timing requirements are dominating & error correction may be compromised
- In other cases, error correction requirements are dominating & timing requirements may be compromised



No effect

Imprecise output

System crash

- Classification requires application knowledge

Analysis of H.264 video decoder
(~ 3500 lines of C code)

- Fraction of memory space

Resolution	Memory size of reliable data	Memory size of unreliable data
176 x 144	90 kB (55%)	74 kB (45%)
352 x 288	223 kB (43%)	297 kB (57%)
1280 x 720	1 585 kB (37%)	2 700 kB (63%)

- Impact of uncorrected errors:

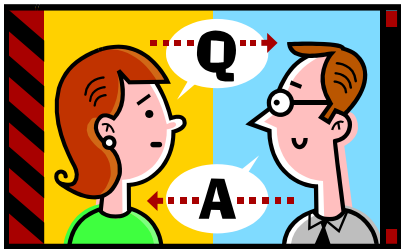
Injection into unreliable memory	240 faults / sec	80 faults / sec
Average PSNR of frames with errors	40.89 dB	50.57 dB

Deteriorated exactness, timeliness maintained!

Summary

- Motivation, introduction
- Timing
  - $WCET_{EST}$ computation
  - Worst-case execution time aware compilation
- Energy
  - Scratch pad memories
- Multi-objective optimization
  - High-speed processing for biosensor
  - Sequence of compiler optimizations, parallelization
  - Memory-optimized task mapping
  - Tradeoff reliability/timeliness
- Summary

Q&A?



Compile-time actions

Application knowledge provided by annotations

Impact of errors
Urgency of error correction
Feasible correction methods

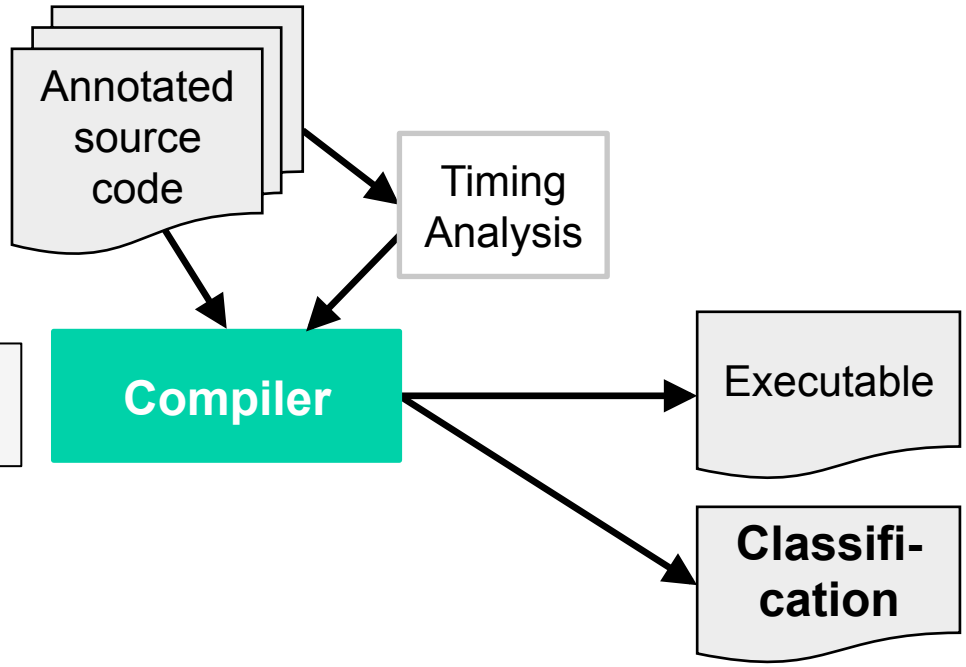
```
unreliable int j;
reliable   int control;
```

Propagation of annotations and inference of error classes

```
reliable int y = control;
```

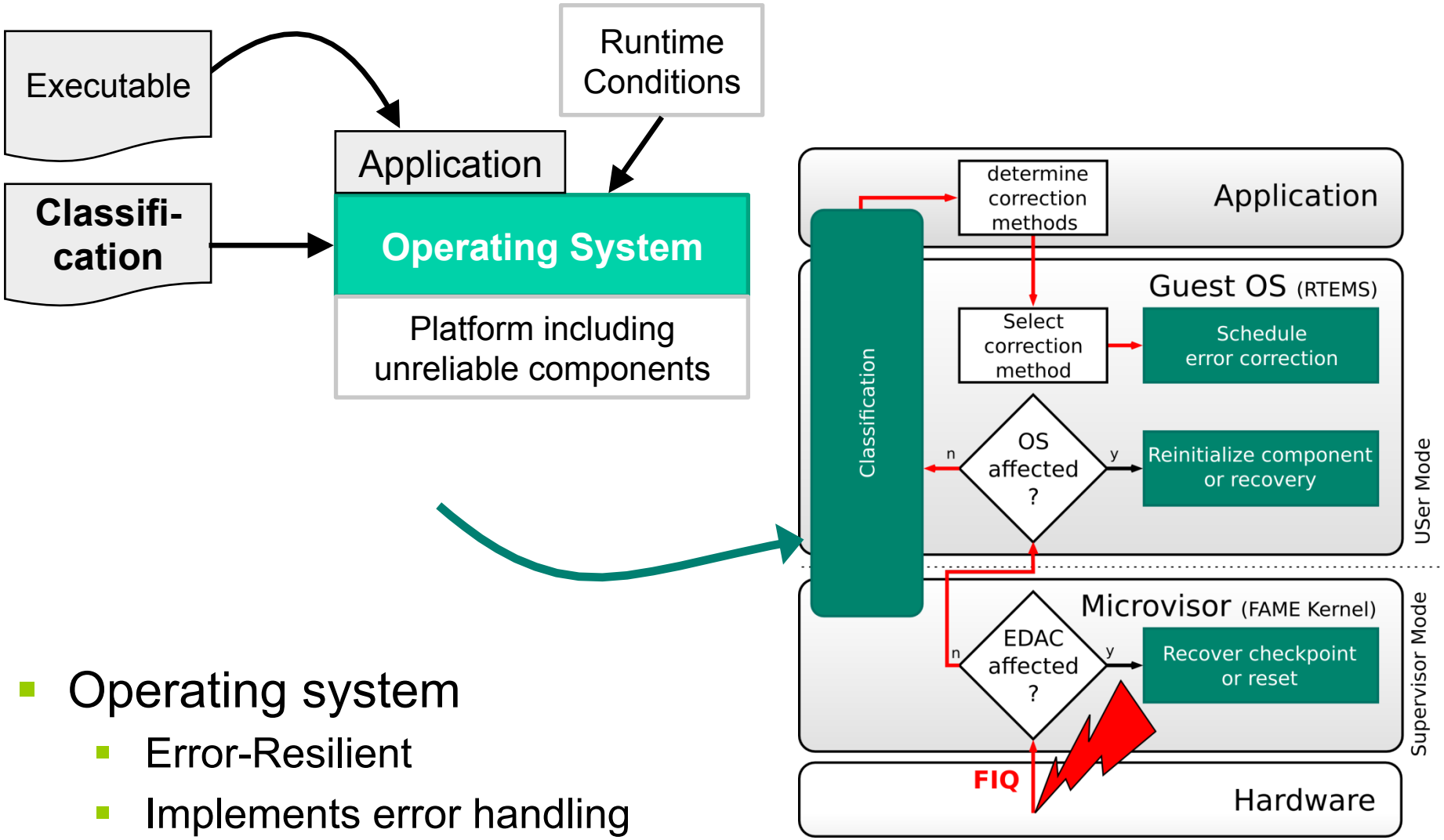
Encoding of classification

0110110101110111100010101110110011
1101110101101100011010101001001101



- Compiler
  - Source code analysis
  - Creates classification

Run-time actions

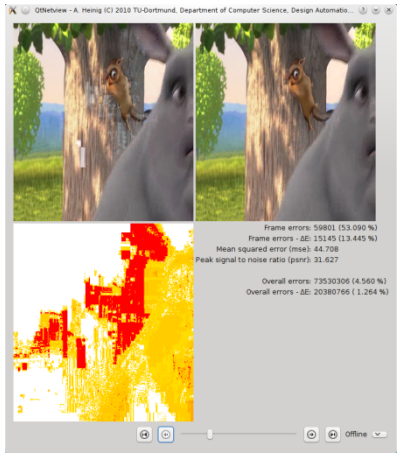


- Operating system
  - Error-Resilient
  - Implements error handling

Results

Analysis of H.264 video decoder (~ 3500 lines of C code)

- Fault-injection and error impact evaluation tools.



- **No crashes** due to fault-injection into unreliable data.
- Fraction of memory space required to store reliable data:

Resolution	Memory size of reliable data	Memory size of unreliable data
176 x 144	90 kB (55%)	74 kB (45%)
352 x 288	223 kB (43%)	297 kB (57%)
1280 x 720	1 585 kB (37%)	2 700 kB (63%)

Results (cont' d)

Impact of uncorrected errors on the quality of service:

Injection into unreliable memory	240 faults / sec	80 faults / sec
Average PSNR of frames with errors	40.89 dB	50.57 dB

Deteriorated reliability/QoS, timeliness maintained!

- Approach also applicable to probabilistic error models.
- Fraction of operations, whose results needn' t be exact:

Instruction Type	add	sub	rsub	mul	overall
Unreliable operations	18.59%	18.60%	43.01%	76.27%	13.36%

Overall covers all kinds of operations including comparisons and branches

Many different objectives important for cyber-physical/ embedded systems

- Worst case delay
- average case delay
- power consumption
- energy consumption
- reliability
- thermal behavior
- safety
- security
- cost, size
- weight
- EMC characteristics
- radiation hardness
- environmental friendliness
- extendability …

☞ Pareto optimality, multi-objective