

طراحی سیستم‌های تعبیه شده Embedded System Design

فصل پنجم - قسمت چهارم

پیاده‌سازی سیستم‌های تعبیه شده

Implementing Embedded Systems: Hardware / Software Codesign

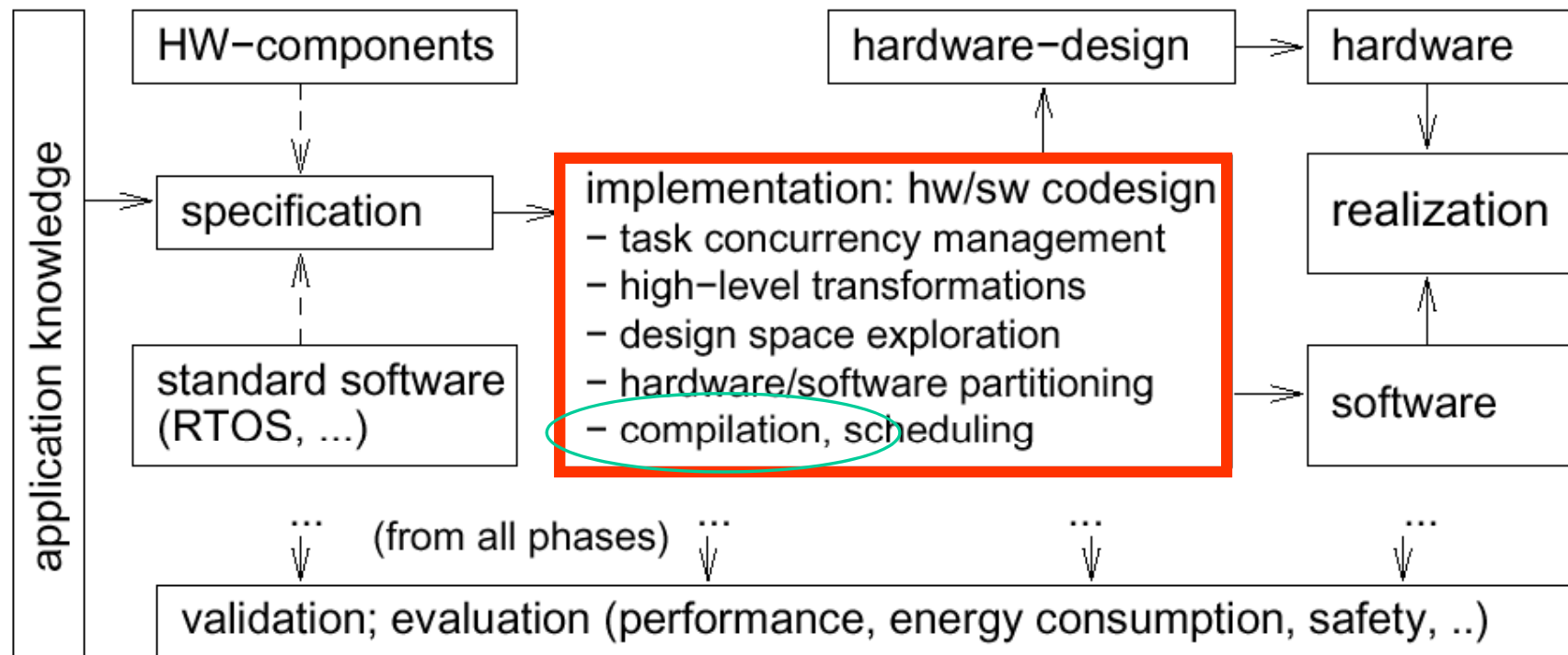
کاظم فولادی

دانشکده‌ی مهندسی برق و کامپیوتر
دانشگاه تهران

kazim@fouladi.ir



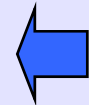
Hardware/Software Codesign



کامپایلرها برای سیستم‌های تعبیه شده

Compilers for embedded systems: Why are compilers an issue?

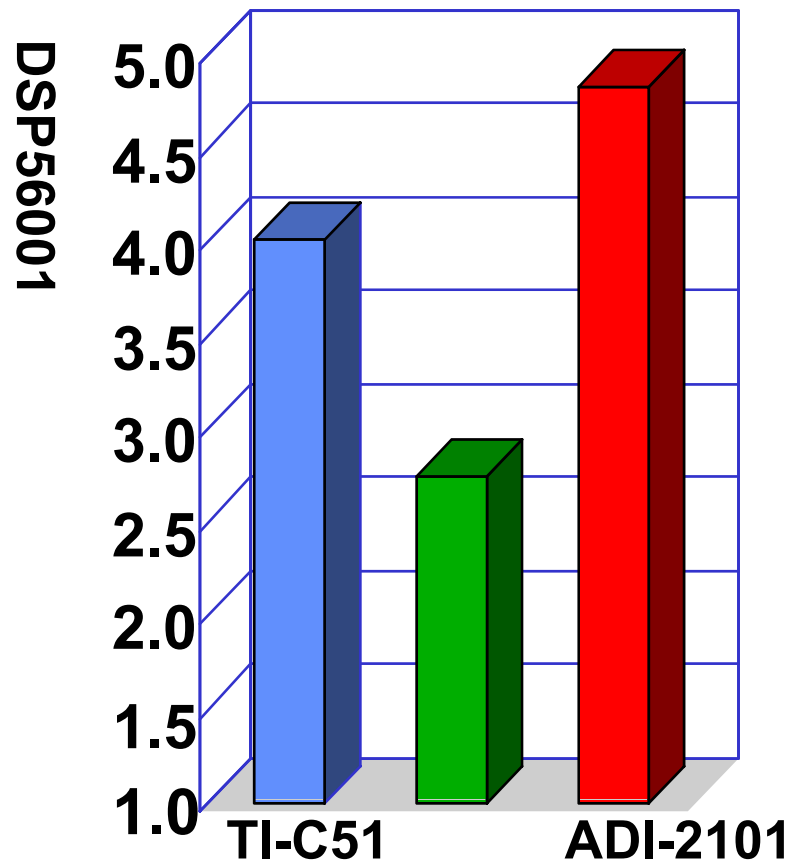
- **Many reports about low efficiency of standard compilers**
 - Special features of embedded processors have to be exploited.
 - High levels of optimization more important than compilation speed.
 - Compilers can help to reduce the energy consumption.
 - Compilers could help to meet real-time constraints.
- **Less legacy problems than for PCs.**
 - There is a large variety of instruction sets.
 - Design space exploration for optimized processors makes sense



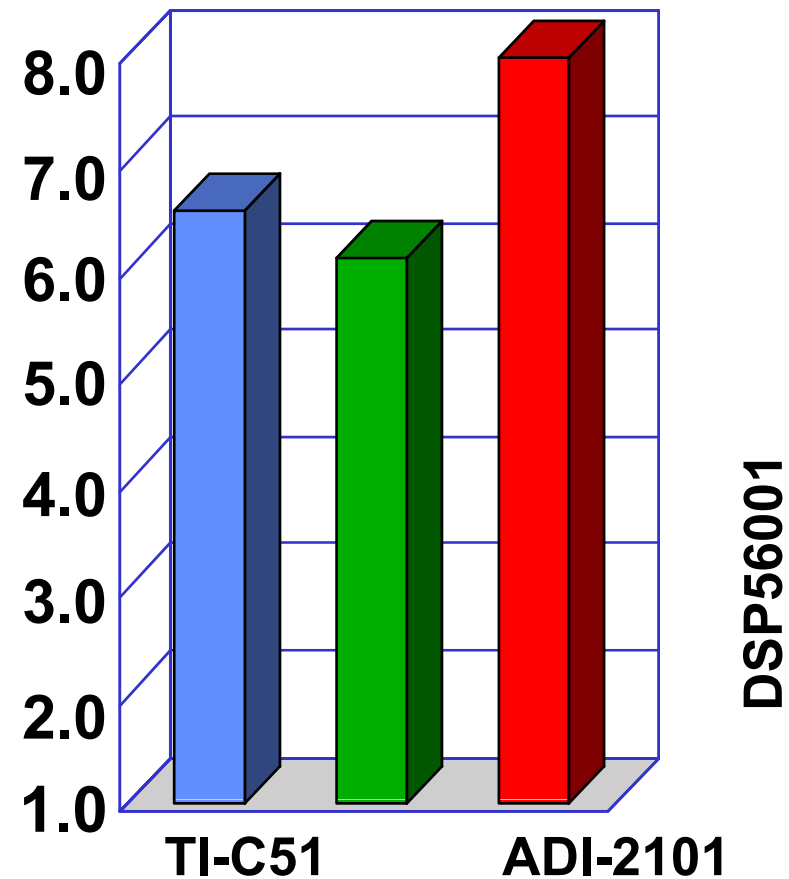
فقدان کارایی کامپایلرهای C برای DSPها

(Lack of) performance of C-compilers for DSPs

DSPStone (Zivojnovic et al.).
Data memory overhead [$\times N$]



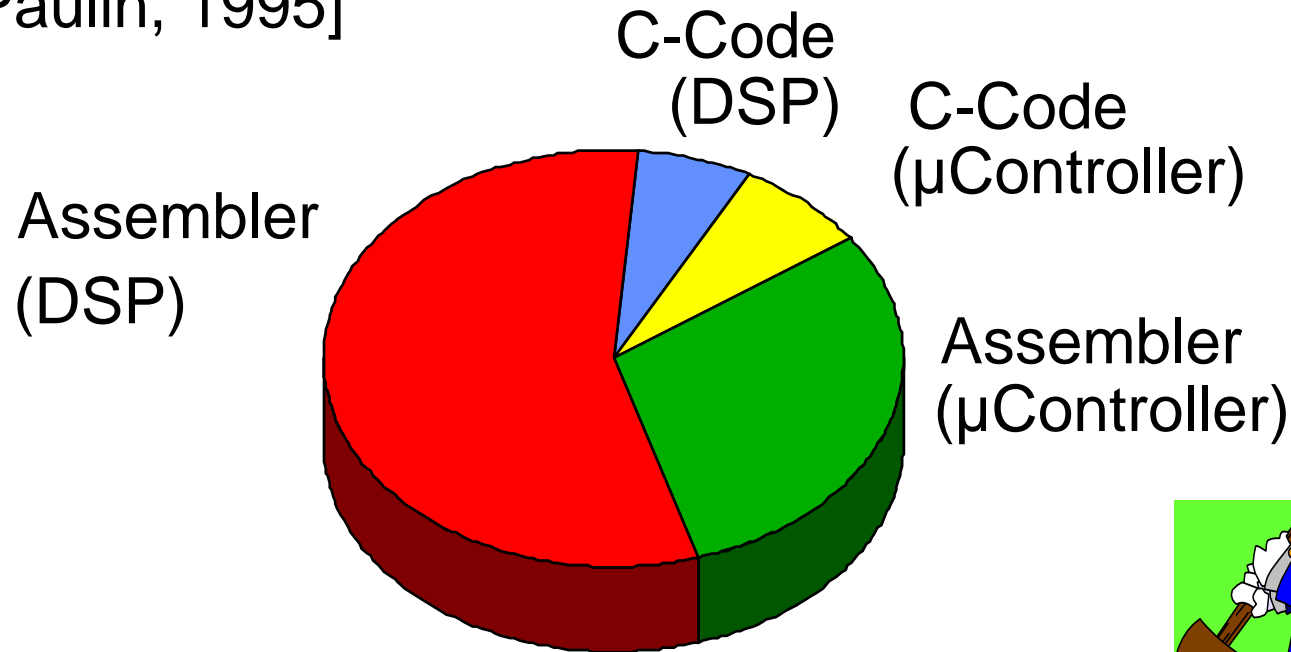
Example: ADPCM
Cycle overhead [$\times n$]



استفاده از زبان‌های اسمبلی در سیستم‌های تعبیه‌شده

Use of assembly languages in embedded systems

[Paulin, 1995]



Similar situation more recently



بهینه‌سازی‌های مورد نظر در این درس Optimizations considered in this course

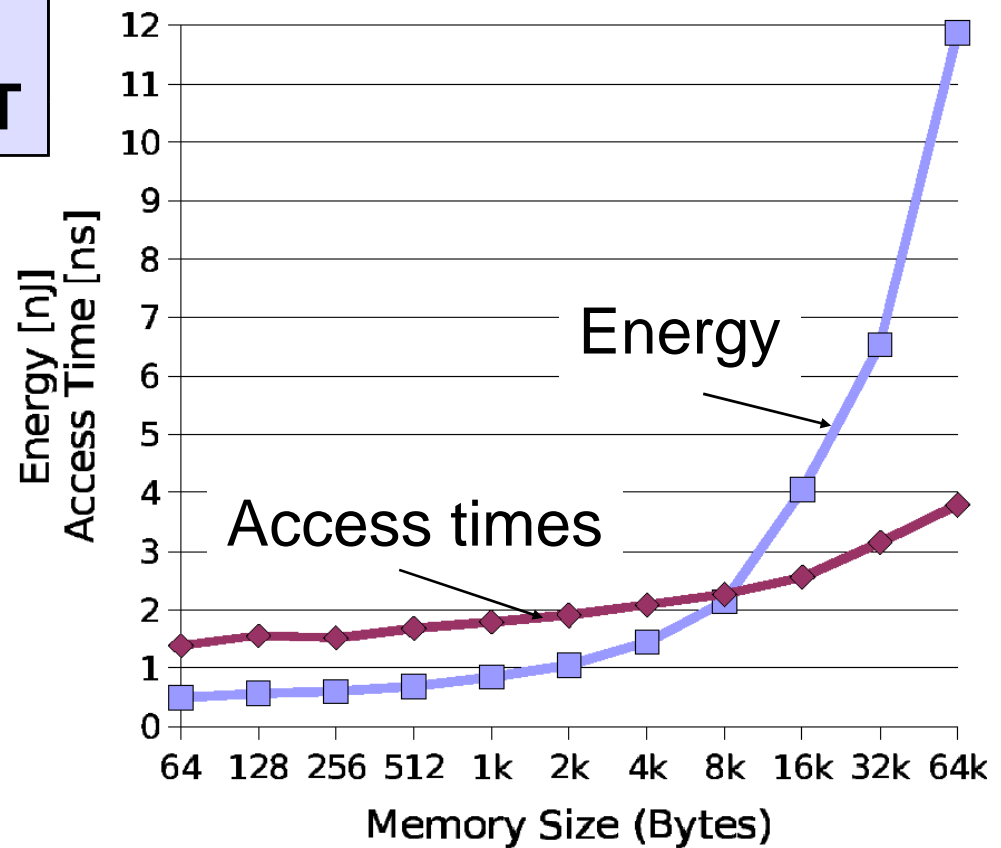
- Energy-aware & memory-aware compilation
- Compilation for digital signal processors
- Compilation for multimedia processors
- Compilation for VLIW processors
- Compilation for network processors
- Compiler generation, retargetable compilers and design space exploration



سه مسأله‌ی کلیدی برای سیستم‌های حافظه‌ی آینده


3 key problems for future memory systems

1. (Average) Speed
2. Energy/Power
3. Predictability / WCET



تلاش‌هایی برای کاهش انرژی

Efforts for Reducing Energy

- **Device Level**
 - Development of Low Power Devices
 - Reducing Power Supply Voltage
 - Reducing Threshold Voltage
- **Circuit Level**
 - Gated Clock
 - Path Transistor Logic
 - Asynchronous Circuits
- **System Level** 



بهینه‌سازی برای انرژی پایین همان بهینه‌سازی برای کارایی بالاست؟ Optimization for low-energy the same as optimization for high performance?

No !

- High-performance if available memory bandwidth fully used; low-energy consumption if memories are at stand-by mode
- **Reduced energy if more values are kept in registers**

```
LDR r3, [r2, #0]
ADD r3,r0,r3
MOV r0,#28
LDR r0, [r2, r0]
ADD r0,r3,r0
ADD r2,r2,#4
ADD r1,r1,#1
CMP r1,#100
BLT LL3
```

2096 cycles
19.92 μJ

```
int a[1000];
c = a;
for (i = 1; i < 100; i++) {
    b += *c;
    b += *(c+7);
    c += 1;
}
```

2231 cycles
16.47 μJ

```
ADD r3,r0,r2
MOV r0,#28
MOV r2,r12
MOV r12,r11
MOV r11,r10
MOV r0,r9
MOV r9,r8
MOV r8,r1
LDR r1, [r4, r0]
ADD r0,r3,r1
ADD r4,r4,#4
ADD r5,r5,#1
CMP r5,#100
BLT LL3
```



بهینه‌سازی‌های کامپایلر برای بهبود کارآمدی انرژی

Compiler optimizations for improving energy efficiency

- Energy-aware **scheduling**
- Energy-aware **instruction selection**
- **Operator strength reduction**: e.g. replace * by + and <<
- Minimize the **bitwidth of loads and stores**
- Standard compiler optimizations with **energy as a cost function**

E.g.: Register pipelining:

```
for i:= 0 to 10 do
  C:= 2 * a[i] + a[i-1];
```



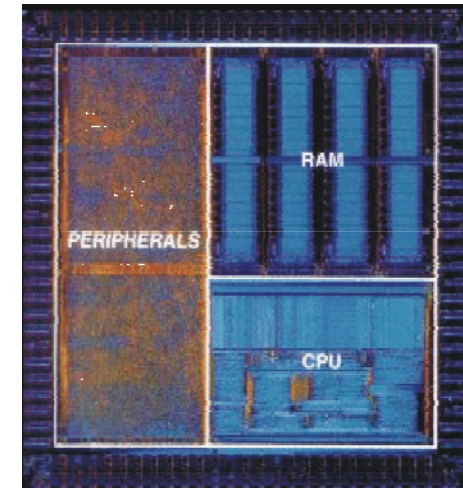
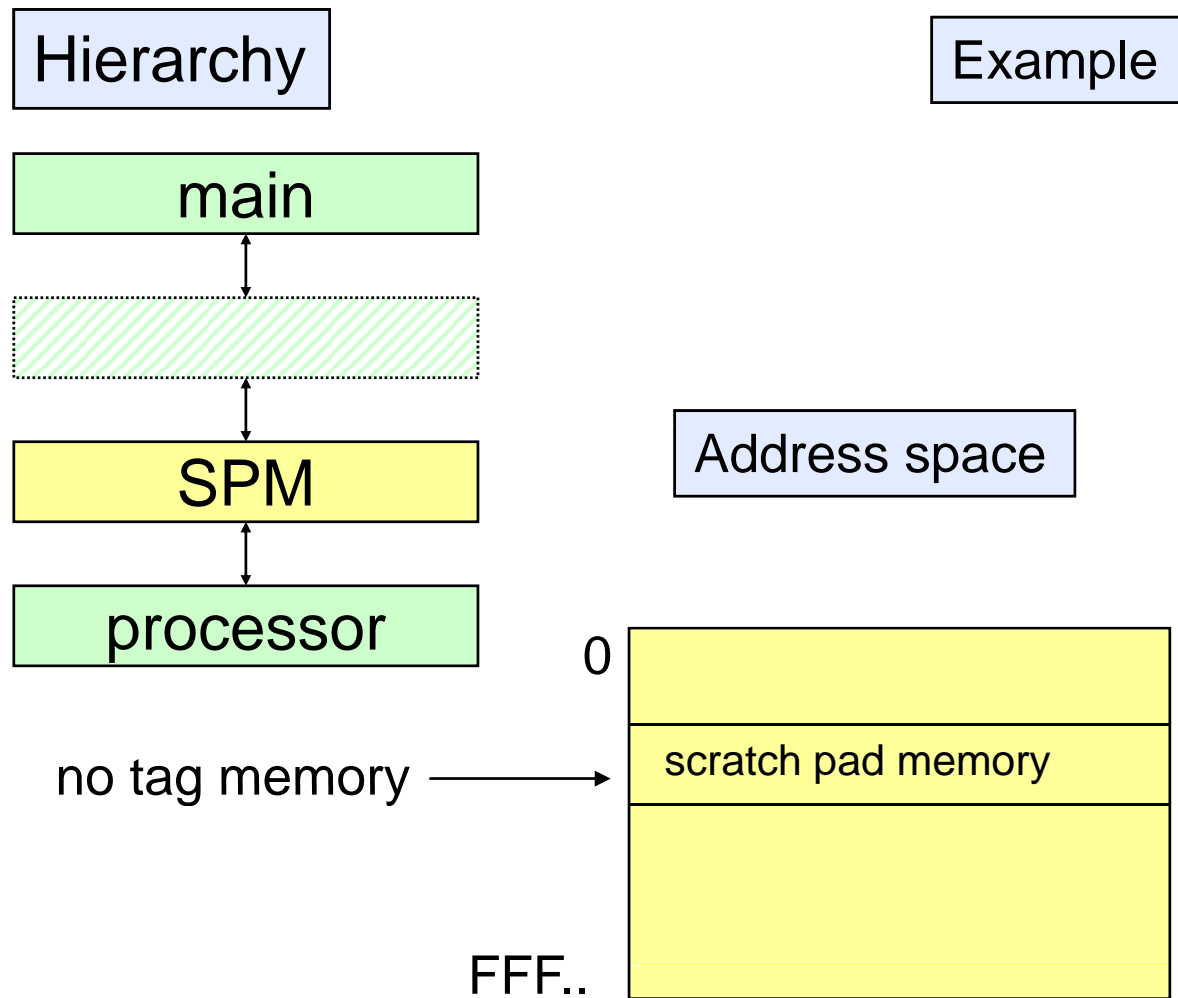
```
R2:=a[0];
for i:= 1 to 10 do
begin
  R1:= a[i];
  C:= 2 * R1 + R2;
  R2 := R1;
end;
```

Exploitation of the memory hierarchy



حافظه‌های سلسله‌مراتبی با استفاده از حافظه‌های چرکنویس

Hierarchical memories using scratch pad memories (SPM)



ARM7TDMI cores, well-known for low power consumption

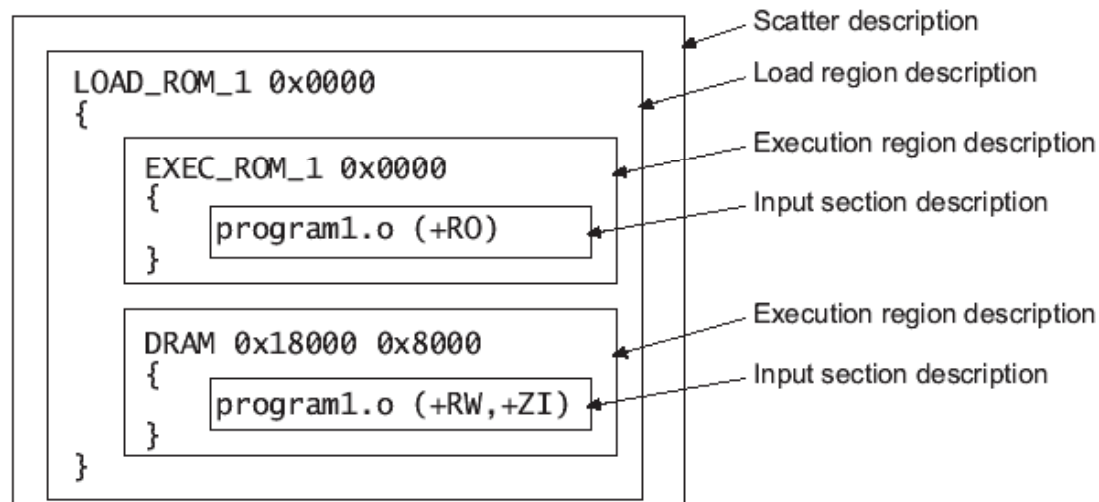
Very limited support in ARMcc-based tool flows

1. Use pragma in C-source to allocate to specific section:

For example:

```
#pragma arm section rwdata = "foo", rodata = "bar"
int x2 = 5; // in foo (data part of region)
int const z2[3] = {1,2,3}; // in bar
```

2. Input scatter loading file to linker for allocating section to specific address range



http://www.arm.com/documentation/Software_Development_Tools/index.html



Very limited usage for actual applications

Information received from designers:

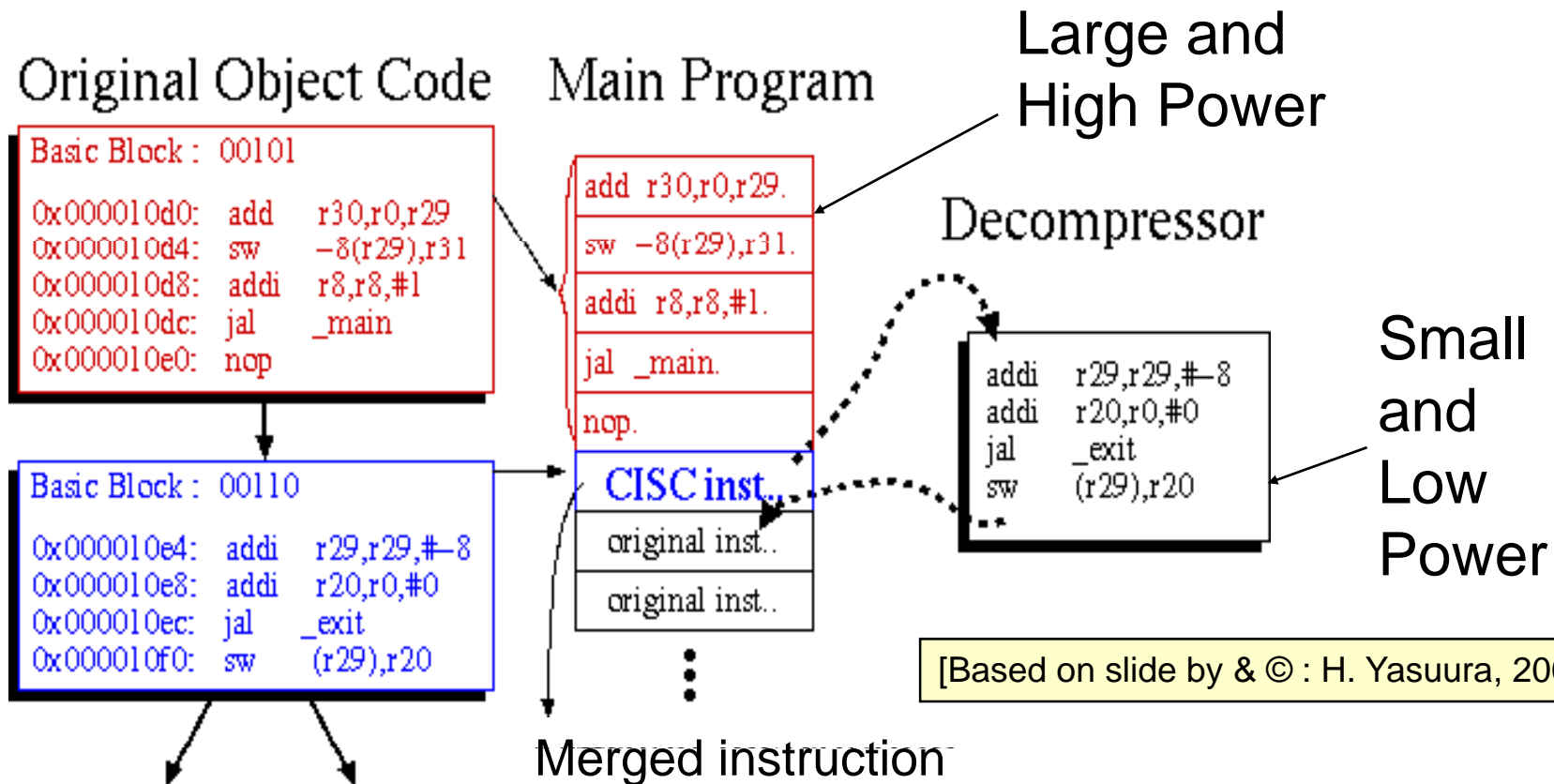
- Used for buffering speech in mobile phones (mobile phone company)
- Essentially no idea on how to exploit it (WLAN/Bluetooth specialists & major vendor)

Why not change this situation?

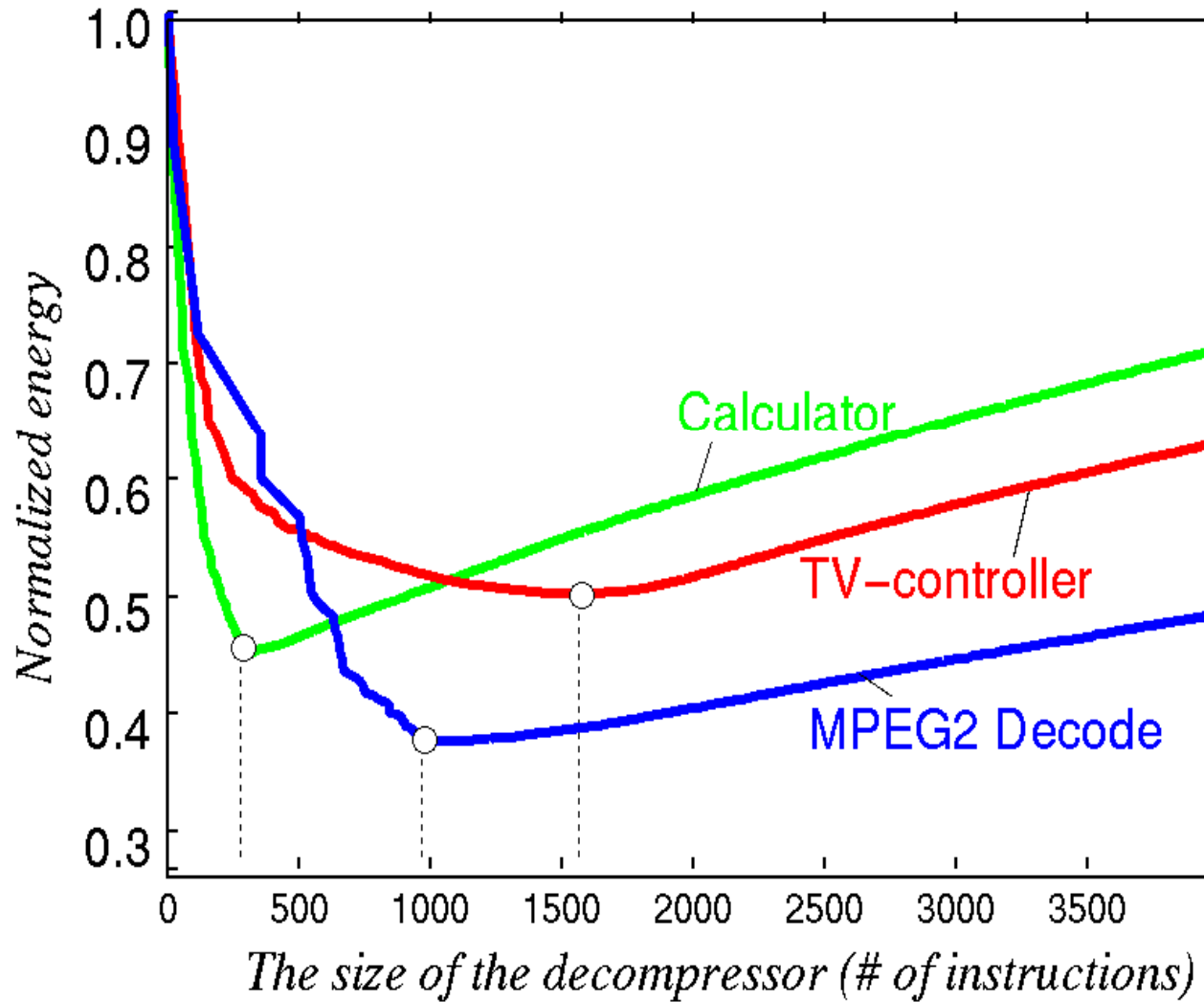


Migration of instructions to small memory Yasuura's architecture (Kyushu U.)

2 instruction memories: main memory + "decompressor" memory;
Compiler optimizing code allocation & size of the 2 instruction memories



Energy savings



[Based on slide by & © : H. Yasuura, 2000]



مهاجرت داده‌ها

Migration of data

Static Allocation: assignment of data structures to SPM based two params $IF(v)$ and $LCF(v)$

$VAC(v)$: *Variable Access Count*: num of accesses to v during its lifetime.

$IAC(v)$: *Interference Access Count*: num of accesses to other arrays during the lifetime of v

$IF(v)$: *Interference Factor*: $IF(v) = VAC(v) * IAC(v)$

$LCF(v)$: *Loop Conflict Factor*: num of access to other arrays in a loop nest

A variable v is migrated to SPM if it has

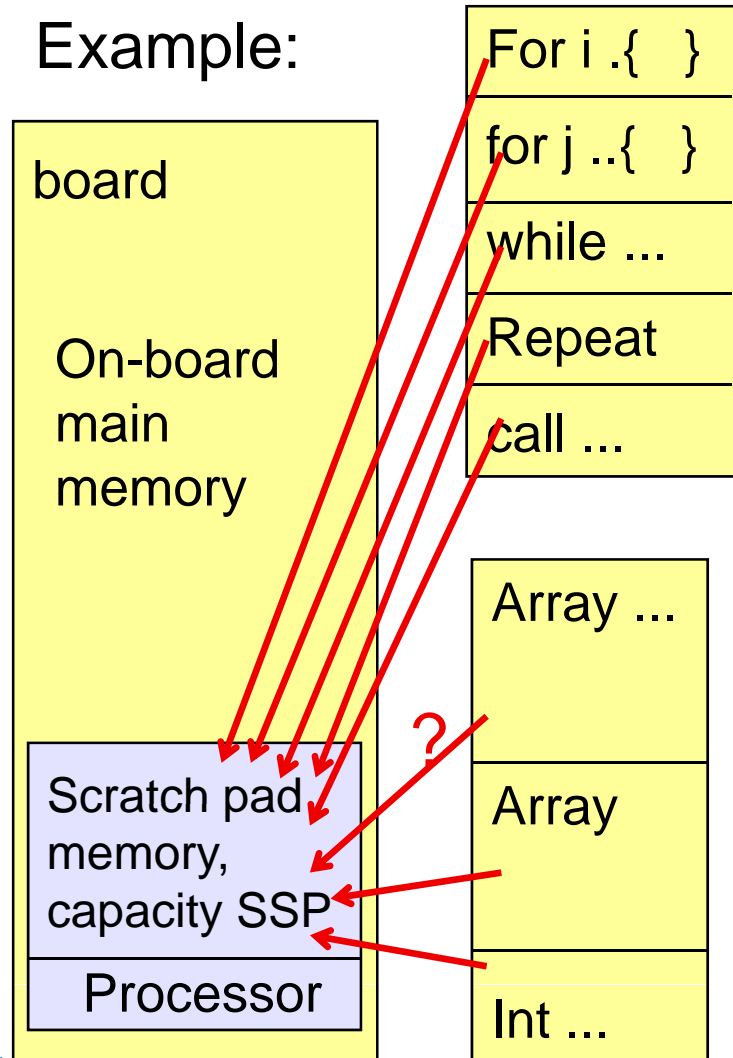
- high *Interference Factor* $IF(v)$
- high *Loop Conflict Factor* $LCF(v)$

[N.Dutt/P.Panda (U. Irvine), 1998]



مهاجرت داده‌ها و دستورالعمل‌ها، مدل‌های بهینه‌سازی سراسری

Migration of data and instructions, global optimization model (U. Dortmund)



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

Gain g_k and size s_k for each segment k .

Maximise gain $G = \sum g_k$, respecting size of SSP $SSP \geq \sum s_k$.

Static memory allocation:

Solution: knapsack algorithm.

Dynamic reloading:

Finding optimal reloading points.



بازنمایی برنامه‌ریزی صحیح – مهاجرت توابع و متغیرها – IP representation - migrating functions and variables-

Symbols:

$S(var_k)$ = size of variable k

n_k = number of accesses to variable k

$e(var_k)$ = energy **saved** per variable access, if var_k is migrated

$E(var_k)$ = energy **saved** if variable var_k is migrated (= $e(var_k) n(var_k)$)

$x(var_k)$ = decision variable, = 1 if variable k is migrated to SPM,
= 0 otherwise

K = set of variables

Similar for functions /

Integer programming formulation:

Maximize $\sum_{k \in K} x(var_k) E(var_k) + \sum_{i \in I} x(F_i) E(F_i)$

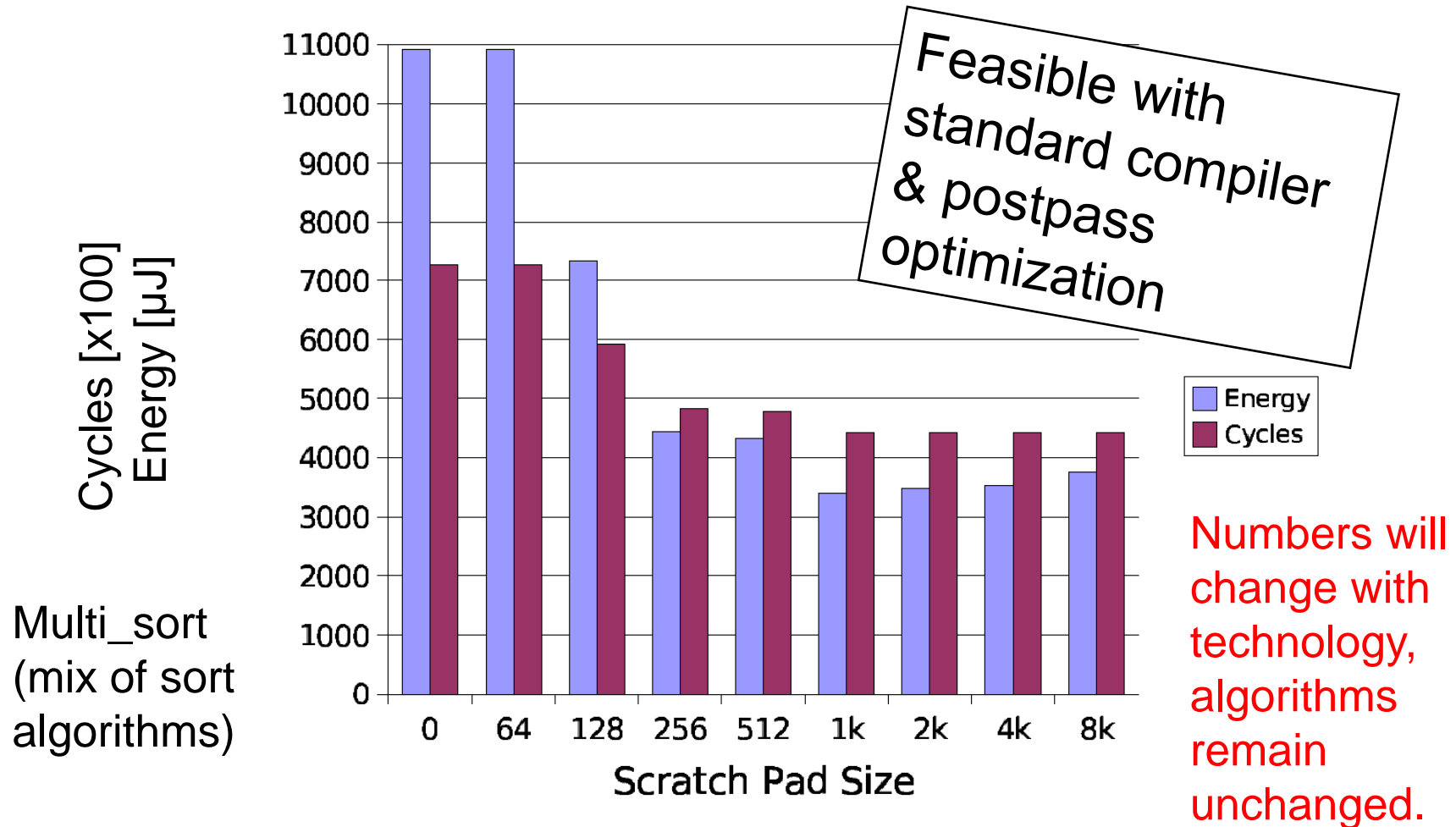
Subject to the constraint

$\sum_{k \in K} S(var_k) x(var_k) + \sum_{i \in I} S(F_i) x(F_i) \leq SSP$



کاهش در انرژی و زمان اجرای متوسط

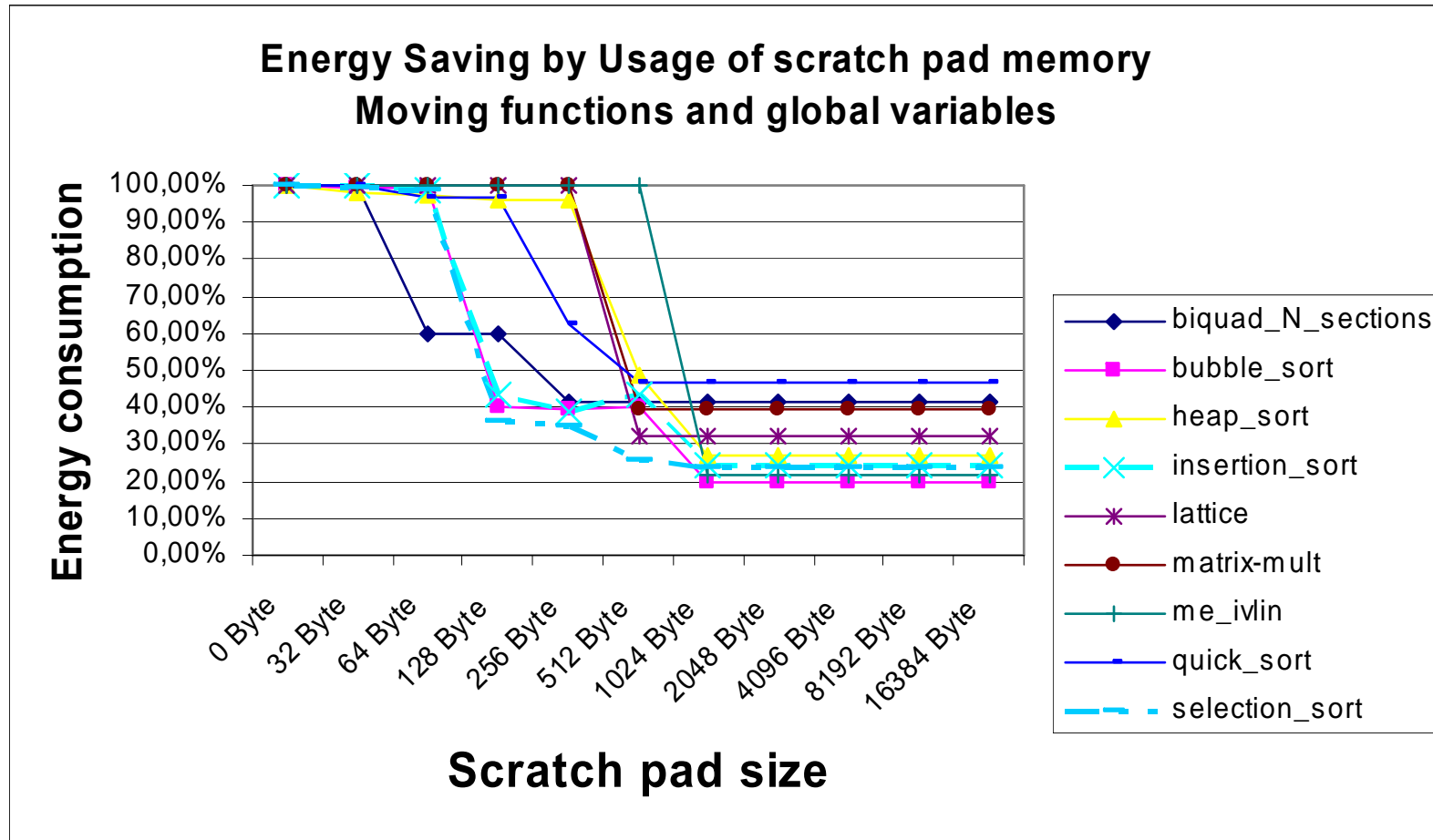
Reduction in energy and average run-time



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



Energy results for more benchmarks



Combined energy model [Steinke et al. (U. Dortmund), 2002]

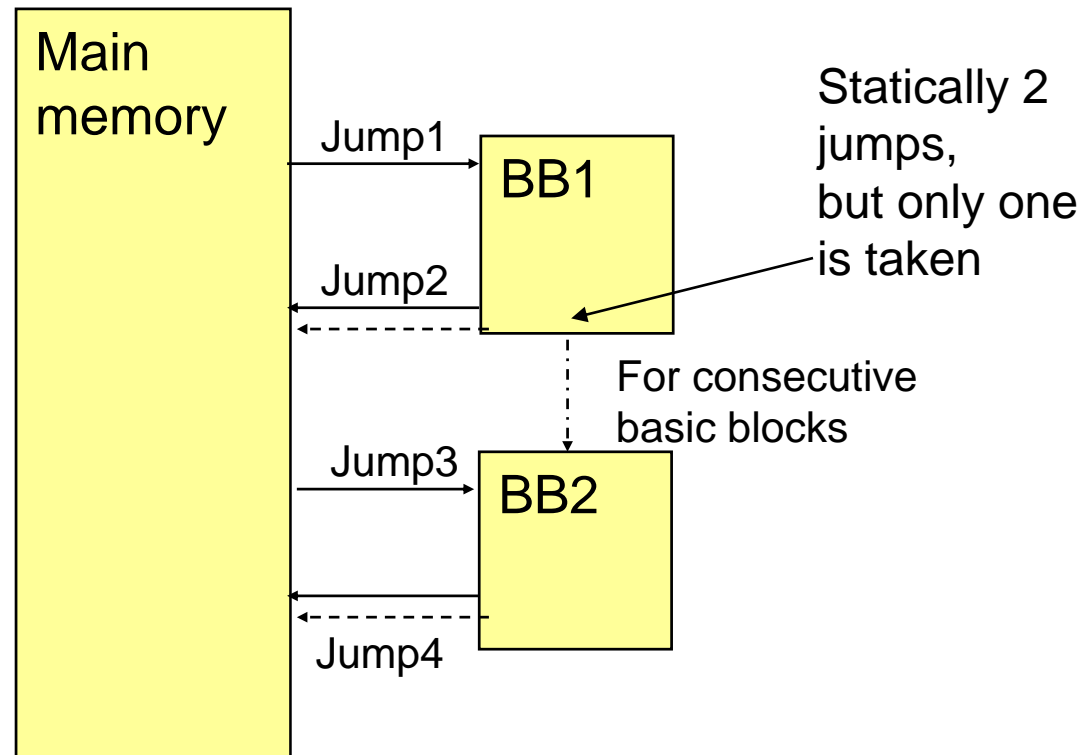


تخصیص بلاک‌های پایه

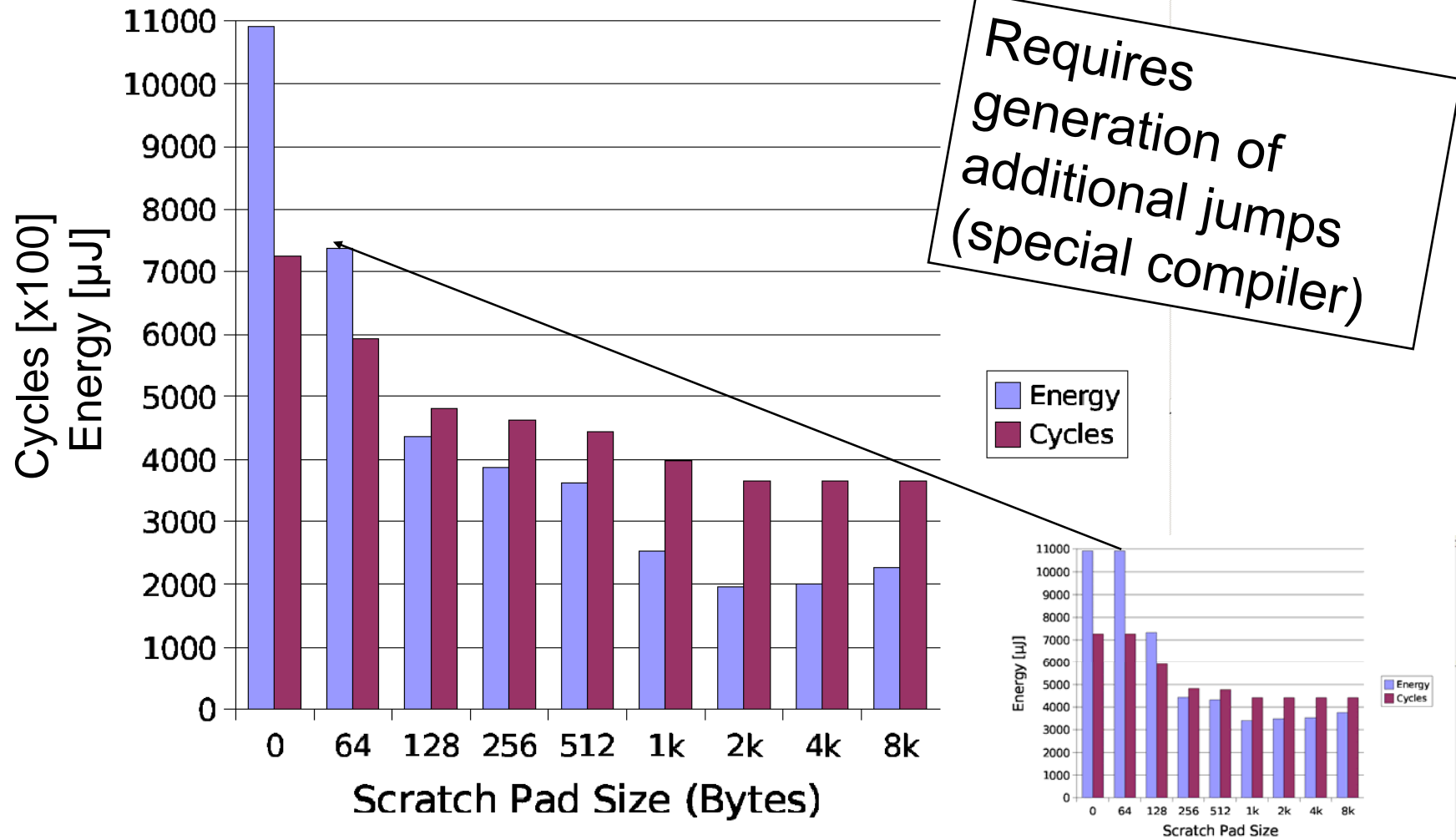
Allocation of basic blocks

Fine-grained granularity smoothens dependency on the size of the scratch pad.

Requires additional jump instructions to return to "main" memory.

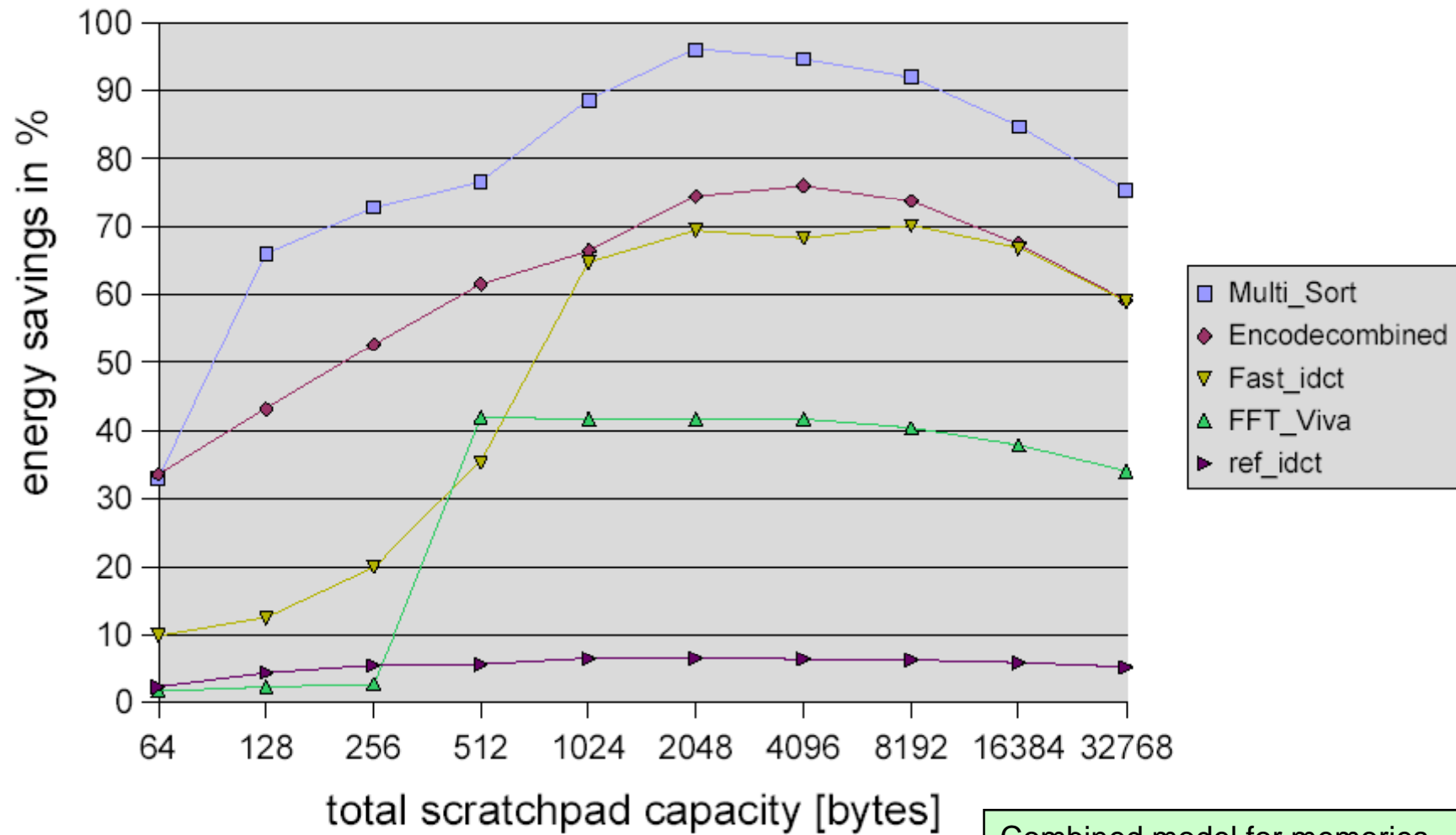


Allocation of basic blocks, sets of adjacent basic blocks and the stack



صرفه جویی برای انرژی سیستم حافظه بتنهایی

Savings for memory system energy alone

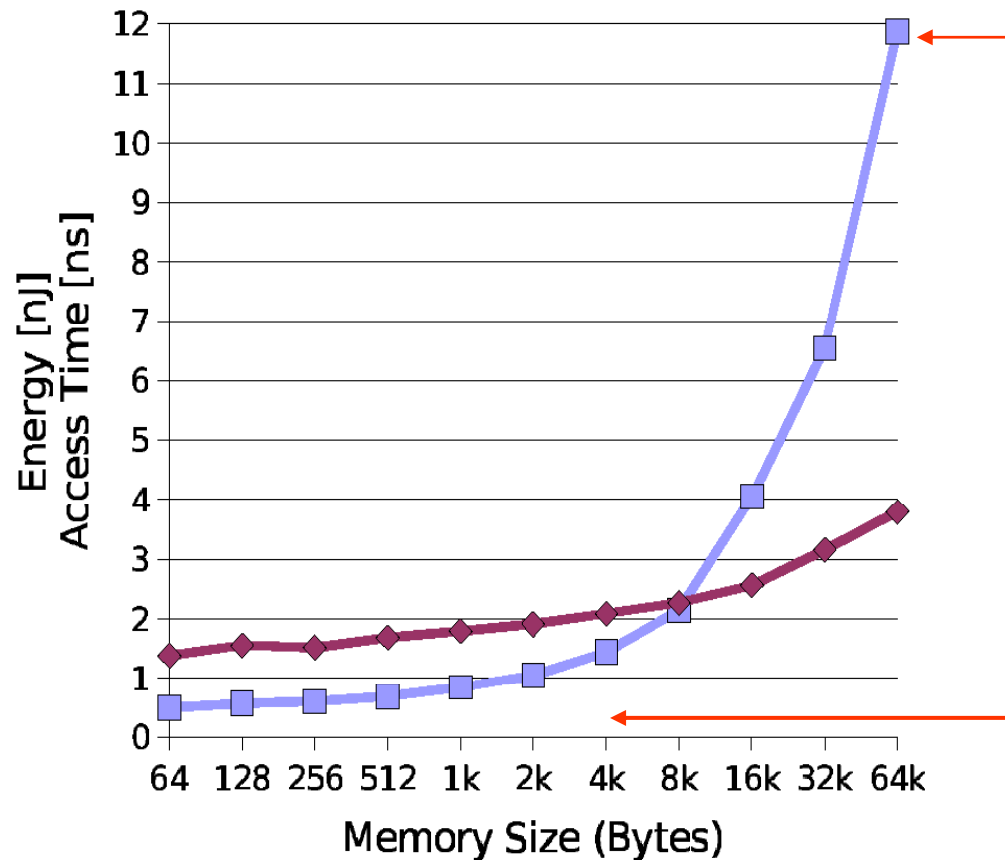


Combined model for memories



دلایل متضمن صرفه جویی زیاد در مصرف انرژی

Underlying reasons for large energy savings



Potential energy saving is determined by the difference between the energy consumption for large and small memories.

In this case:

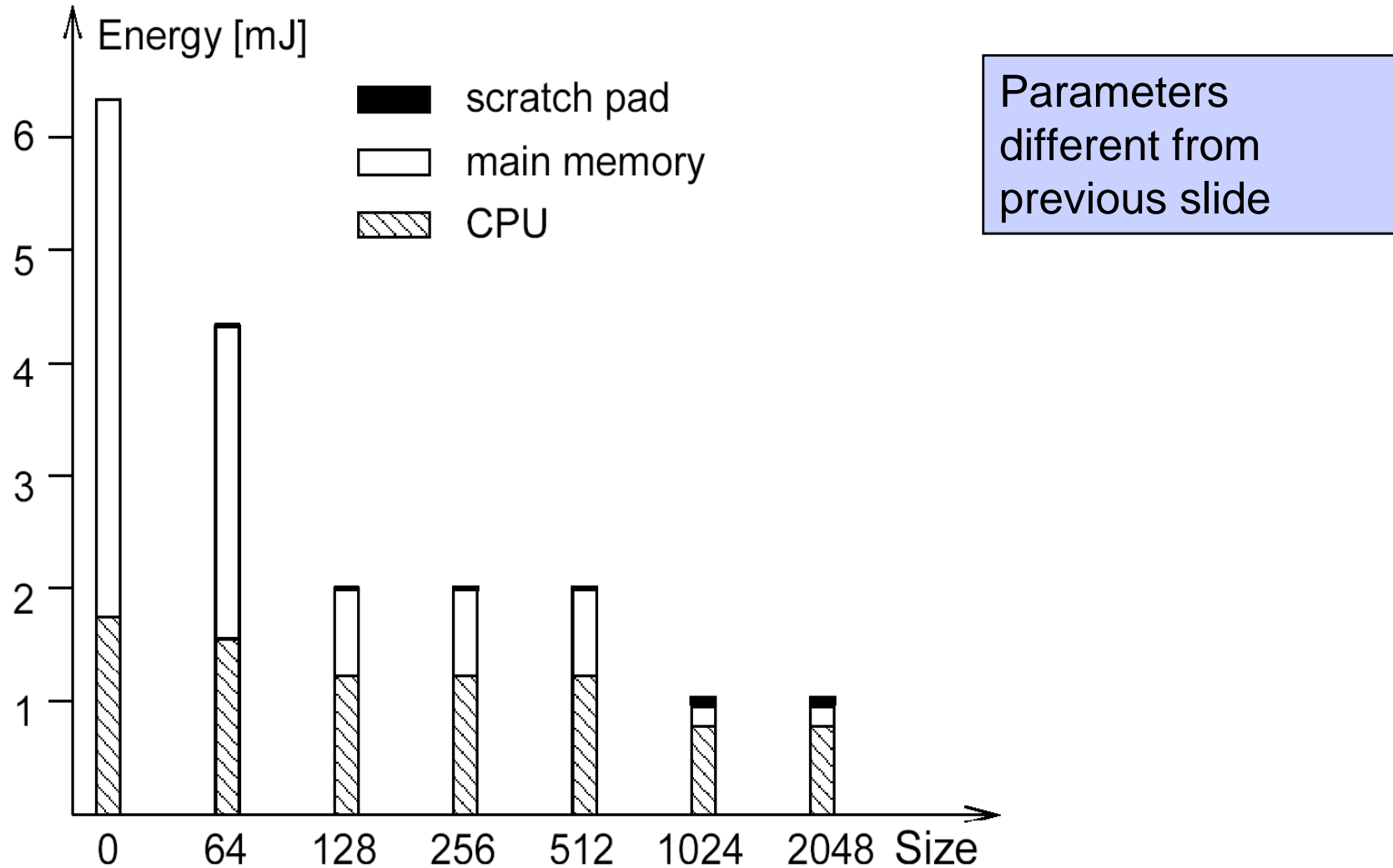
factor of ≈ 20

☞ savings of up to 95%

Can even be larger for larger memories



مصرف انرژی هر واحد کارکردی به عنوان تابعی از اندازه‌ی حافظه‌ی چرکنویس Energy consumption per functional unit, as a function of the SPM size



حساسیت به داده‌های ورودی

Sensitivity with respect to input data: results by Yasuura

$$\text{Reduction rate} = \frac{\text{Energy consumption in optimized memory}}{\text{Energy in memory without optimization}} \times 100$$

MPEG2 decoder

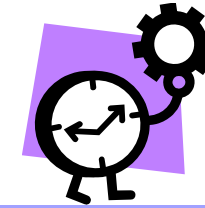
Executed data \ Optimized for	Image 1	Image 2	Image 3
Image 1	45.82 %	46.11%	46.48%
Image 2	46.81%	46.46%	48.80%
Image 3	46.42%	46.23%	45.76%
<u>Decompressor</u> Main Program	<u>1,134</u> 28,407	<u>1,212</u> 28,338	<u>988</u> 28,542

[Based on slide by & © : H. Yasuura, 2000]



قابلیت پیش‌بینی زمانی

Timing predictability



For embedded (hard or soft) real-time systems solutions have to provide **timing predictability**.

For satisfying timing constraints in hard real-time systems, predictability is the most important concern [Xu, Parnas].

Many solutions for PCs (caches) not designed for realtime:

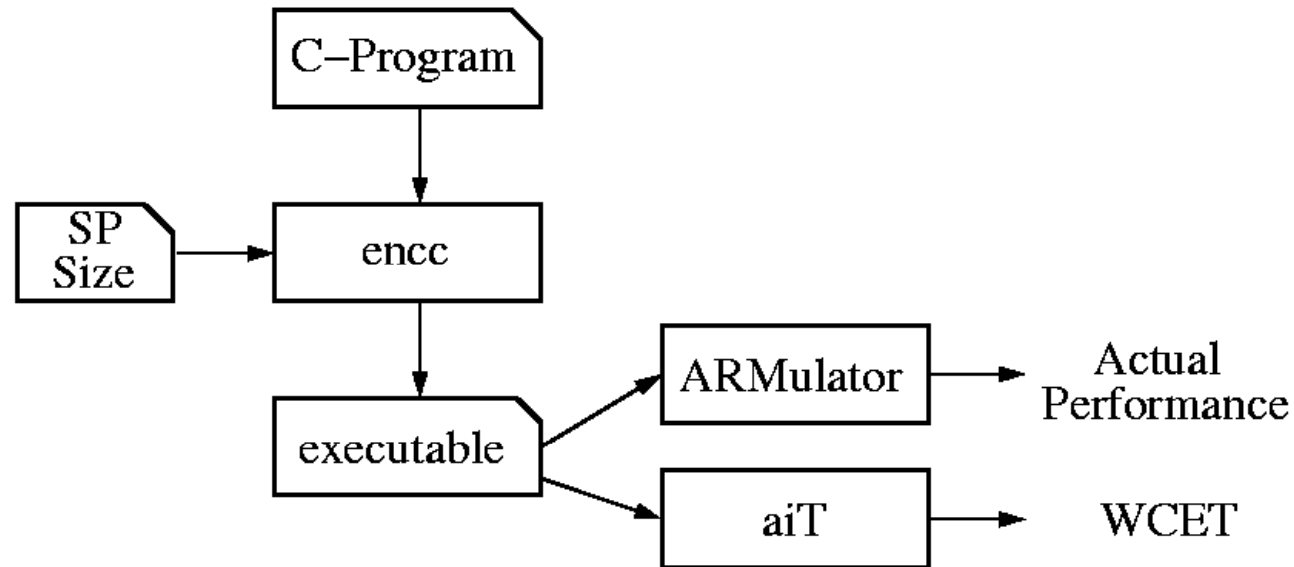
- Designed to improve the average case behavior

Pre run-time scheduling is often the only practical means of providing predictability in a complex system. [Xu, Parnas]

- ☞ Let's do this for the memory system
 - ☞ Scratch-pad/tightly coupled memory based predictability



Workflow



aiT:

- WCET analysis tool
- support for scratchpad memories by specifying different memory access times
- also features experimental cache analysis for ARM7



معماری‌های بررسی شده

Architectures considered

ARM7TDMI with 3 different memory architectures:

1. Main memory

LDR-cycles: (CPU,IF,DF)=(3,2,2)

STR-cycles: (2,2,2)

* = (1,2,0)

2. Main memory + unified cache

LDR-cycles: (CPU,IF,DF)=(3,12,6)

STR-cycles: (2,12,3)

* = (1,12,0)

3. Main memory + scratch pad

LDR-cycles: (CPU,IF,DF)=(3,0,2)

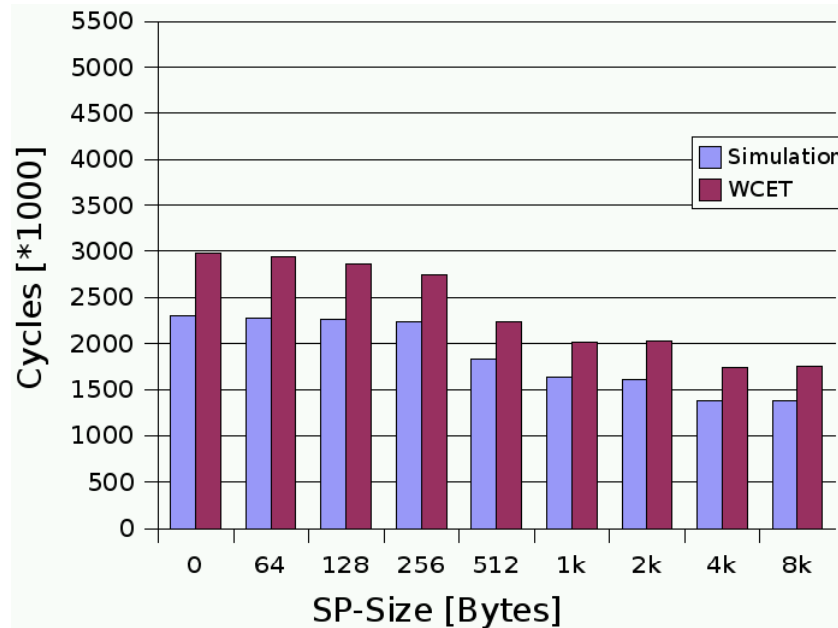
STR-cycles: (2,0,0)

* = (1,0,0)

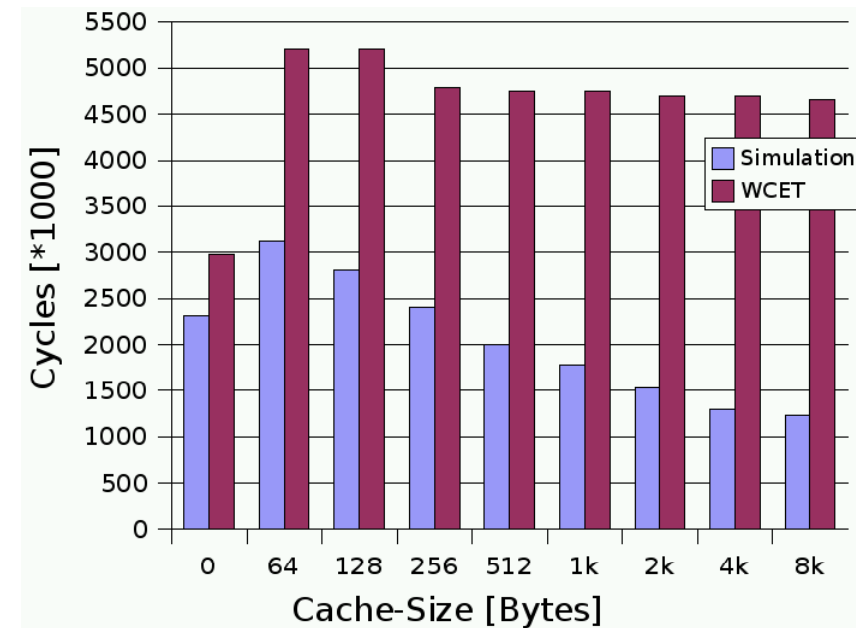


Results for G.721

Using Scratchpad:



Using Unified Cache:



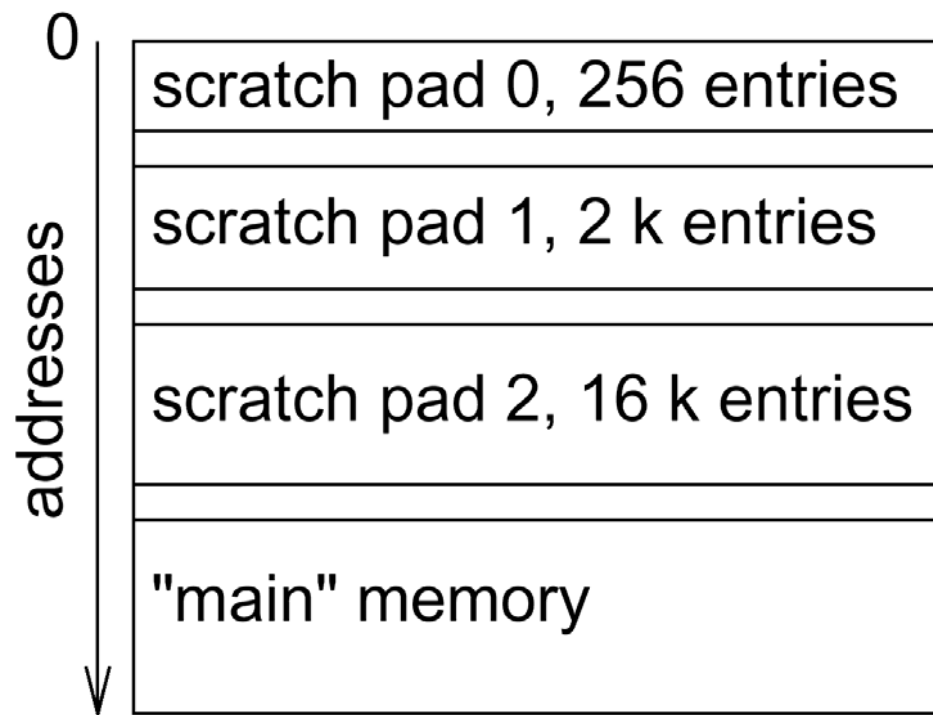
References:

- Wehmeyer, Marwedel: Influence of Onchip Scratchpad Memories on WCET: 4th Intl Workshop on worst-case execution time (WCET) analysis, Catania, Sicily, Italy, June 29, 2004
- Second paper on SP/Cache and WCET at DATE, March 2005



حافظه‌های چرکنویس چندگانه

Multiple scratch pads



بهینه‌سازی برای حافظه‌های چرکنویس چندگانه

Optimization for multiple scratch pads

$$\text{Minimize } C = \sum_j e_j \cdot \sum_i x_{j,i} \cdot n_i$$

With e_j : energy per access to memory j ,
 and $x_{j,i} = 1$ if object i is mapped to memory j , $=0$ otherwise,
 and n_i : number of accesses to memory object i ,
 subject to the constraints:

$$\forall j: \sum_i x_{j,i} \cdot S_i \leq SSP_j$$

$$\forall i: \sum_j x_{j,i} = 1$$

With S_i : size of memory object i ,
 SSP_j : size of memory j .



افزایهای در نظر گرفته شده

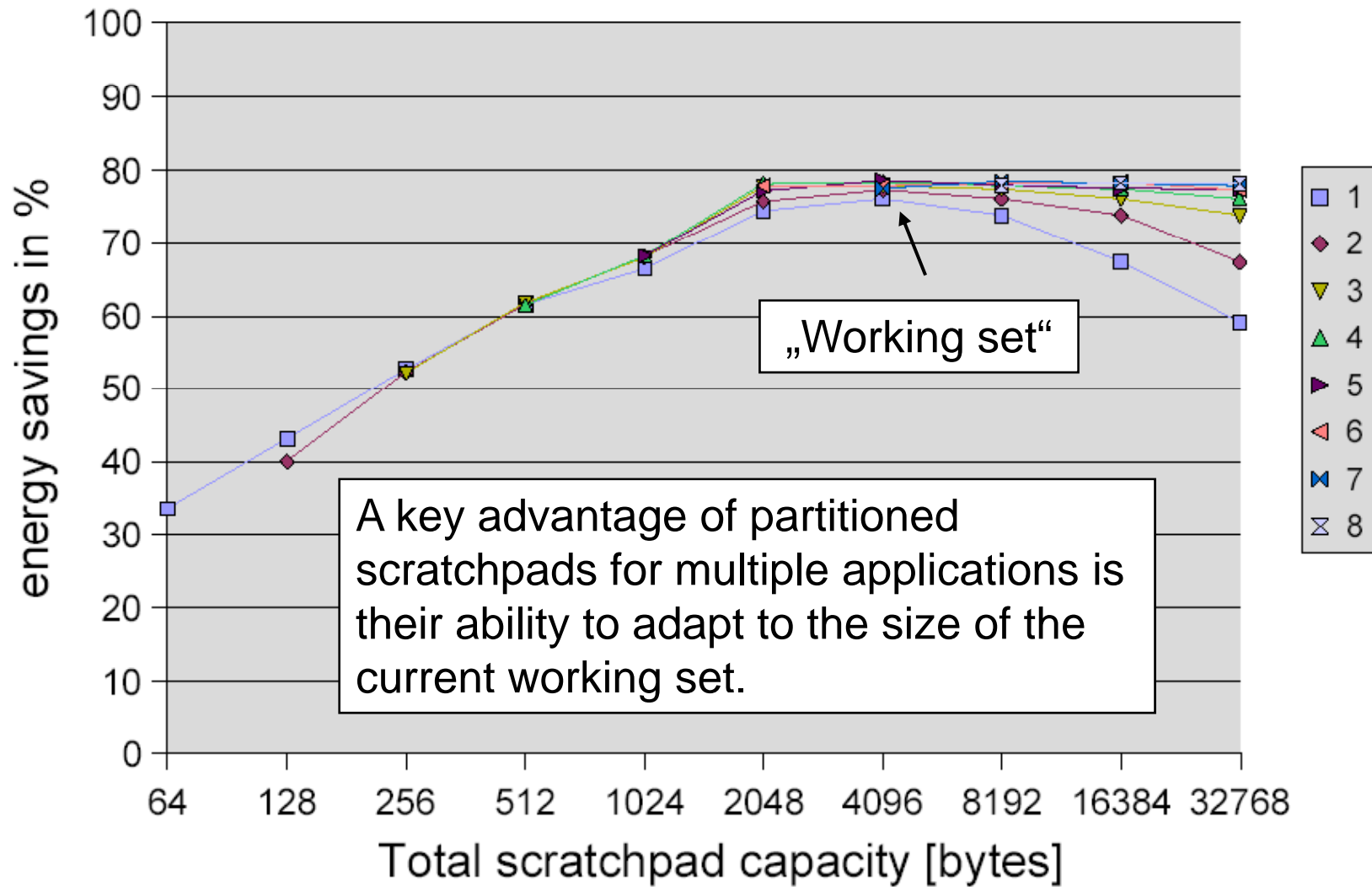
Considered partitions

# of partitions	number of partitions of size:						
	4K	2K	1K	512	256	128	64
7	0	1	1	1	1	1	2
6	0	1	1	1	1	2	0
5	0	1	1	1	2	0	0
4	0	1	1	2	0	0	0
3	0	1	2	0	0	0	0
2	0	2	0	0	0	0	0
1	1	0	0	0	0	0	0

Table 1: Example of all considered memory partitions for a total capacity of 4096 bytes

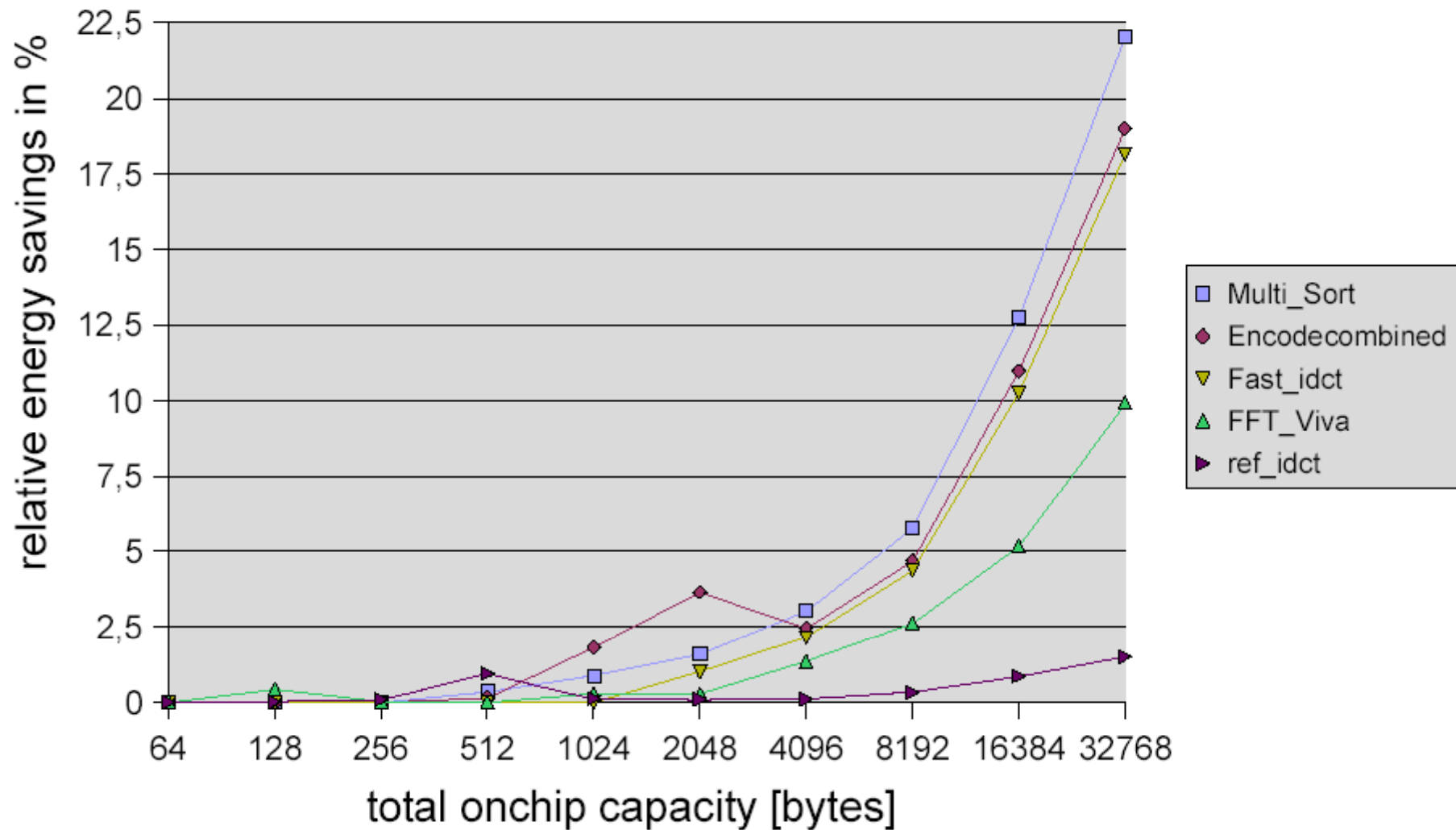


Results for parts of GSM coder/decoder



صرفه‌جویی در برابر حافظه‌های چرکنویس افزاشده

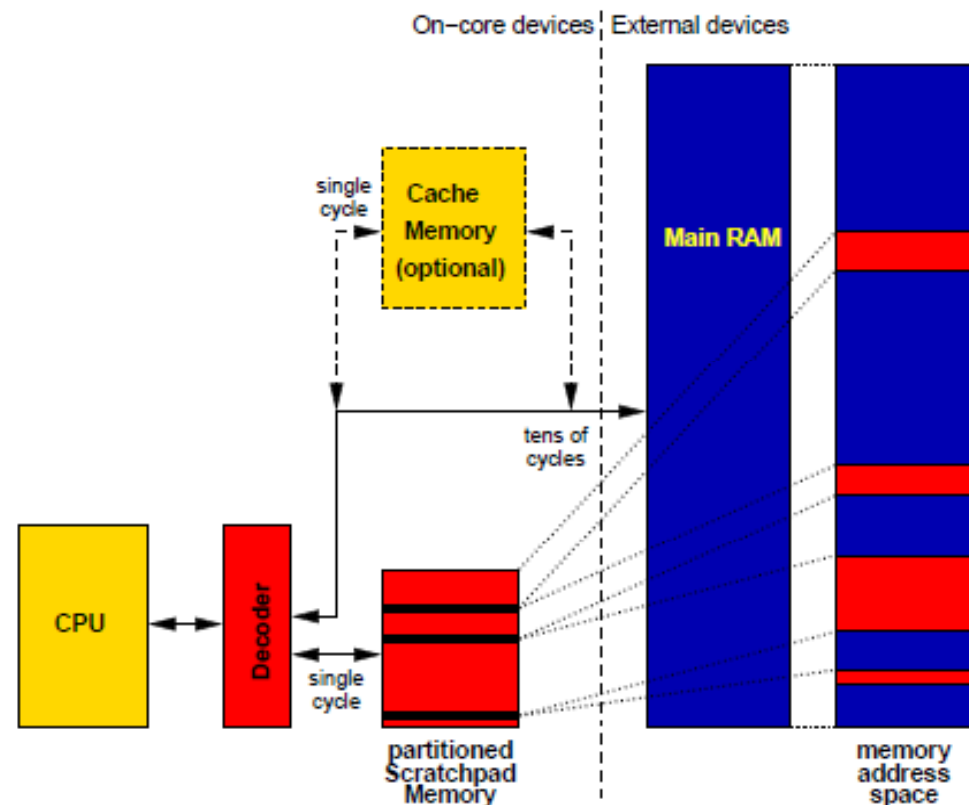
Savings over un-partitioned scratchpad



الگوریتم زمان چند جمله‌ای برای افراز حافظه‌ی چرکنویس روی تراشه

Polynomial-Time Algorithm for On-Chip Scratchpad Memory Partitioning

- Re-maps address ranges to SPMs;
- Modifies hardware, not application software;
- Based executable code, unchanged compiler;
- Pseudo polynomial, uses dynamic programming

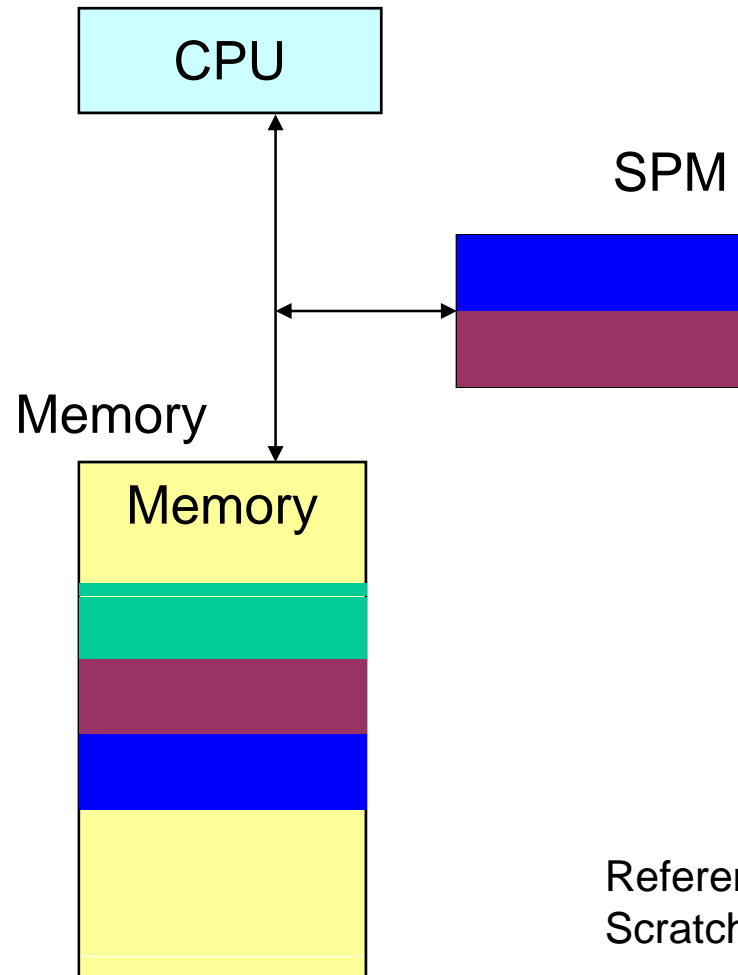


F. Angiolini, L. Benini, A. Caprara: **Polynomial-Time Algorithm for On-Chip Scratchpad Memory Partitioning**, *CASES*, 2003



جایگزینی پویا در میان حافظه‌ی چرکنویس

Dynamic replacement within scratch pad



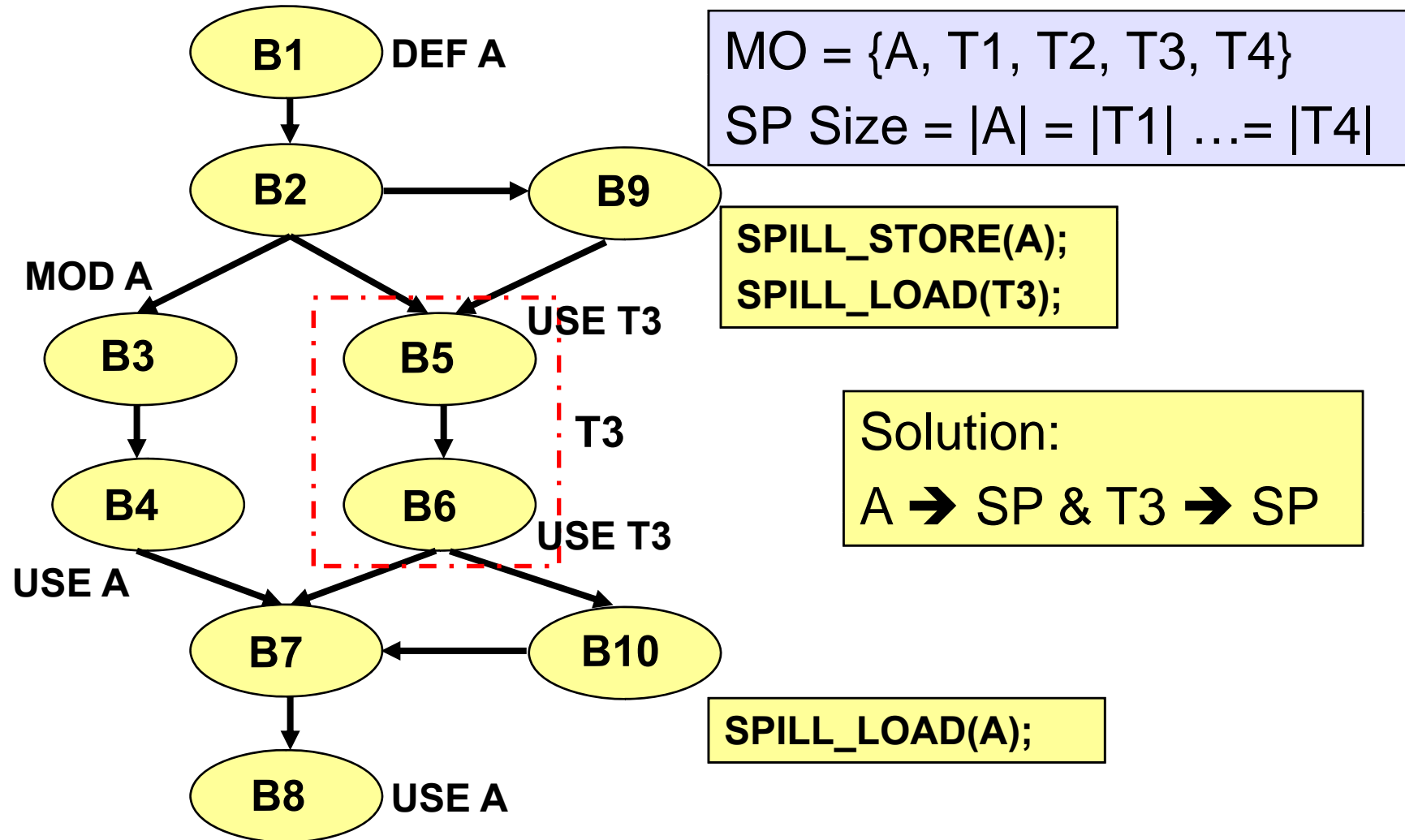
- Effectively results in a kind of **compiler-controlled segmentation/ paging** for SPM
- Address assignment within SPM required (paging or segmentation-like)

Reference: Verma, Marwedel: Dynamic Overlay of Scratchpad Memory for Energy Minimization, ISSS 2004



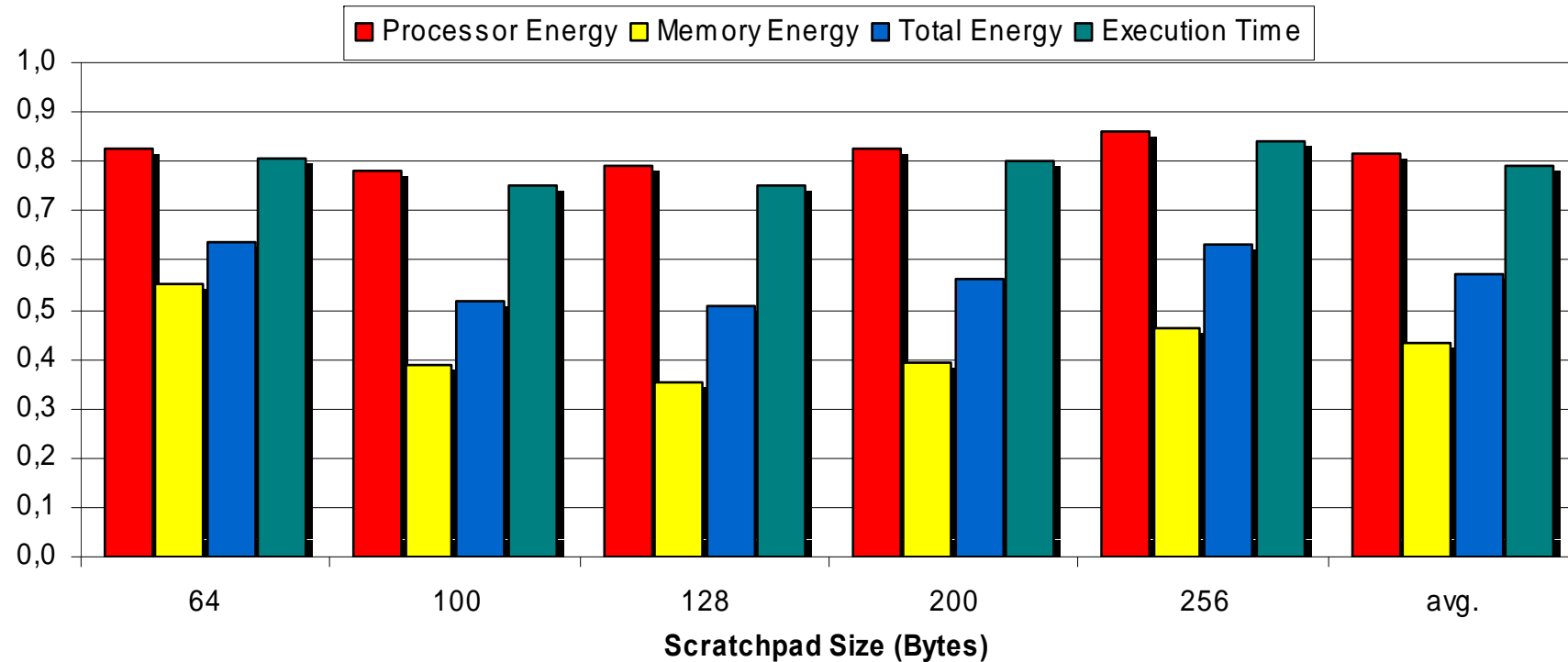
جایگزینی پویای داده‌ها در میان حافظه‌ی چر کنویس: بر اساس تحلیل زنده بودن

Dynamic replacement of *data* within scratch pad: based on liveness analysis



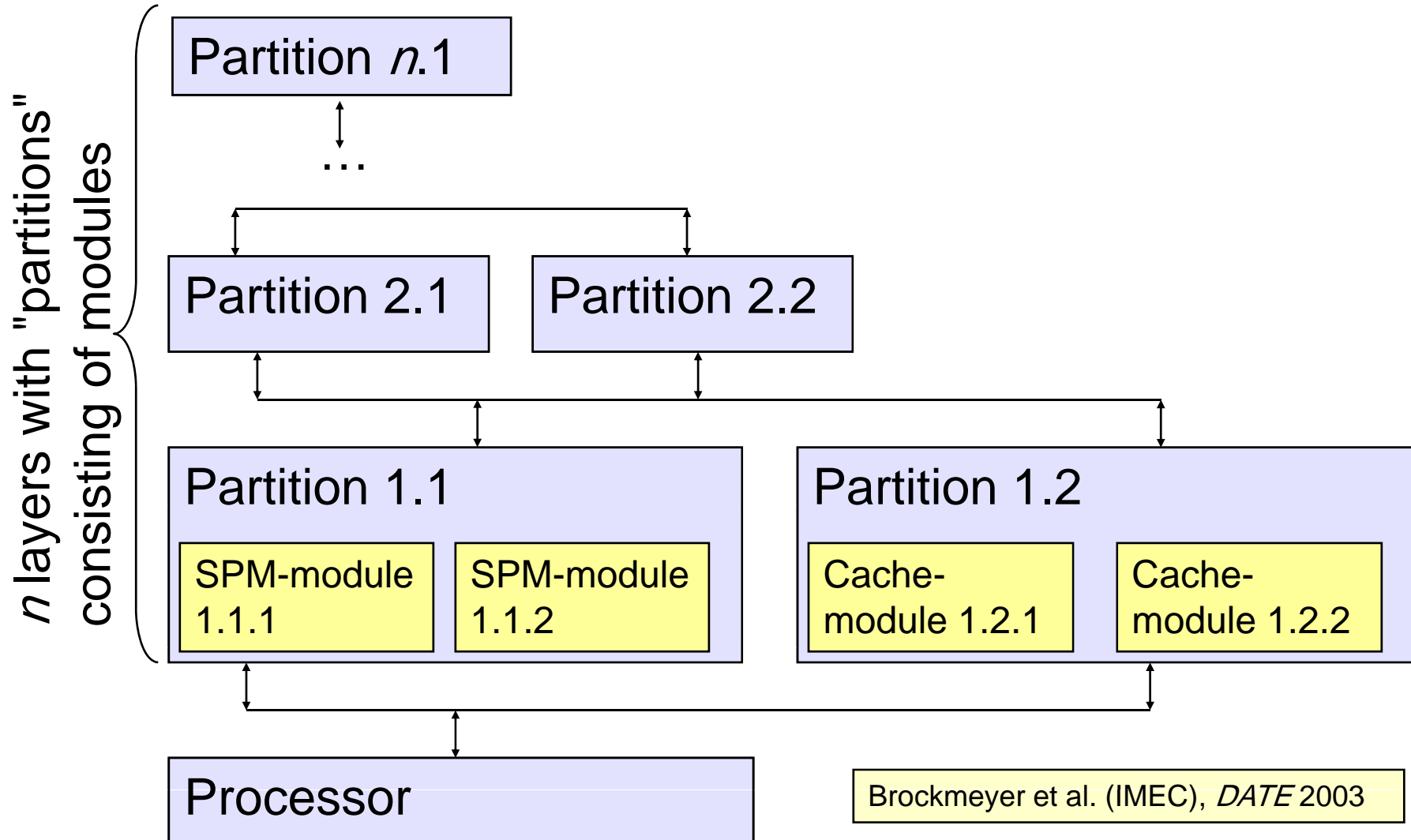
Dynamic replacement within scratch pad

- Results for edge detection relative to static allocation -



حافظه‌های سلسله‌مراتبی: انتساب لایه‌های سلسله‌مراتب حافظه

Hierarchical memories: Memory hierarchy layer assignment (MHLA) (IMEC)



انتساب لایه‌های سلسله‌مراتب حافظه - نامزدهای کپی - Memory hierarchy layer assignment (MHLA) - Copy candidates -

```
int A[250]
for (i=0; i<10; i++)
  for (j=0; j<10; j++)
    for (k=0; k<10; k++)
      for (l=0; l<10; l++)
        f(A[j*10+l])
size=0; reads(A)=10000
```

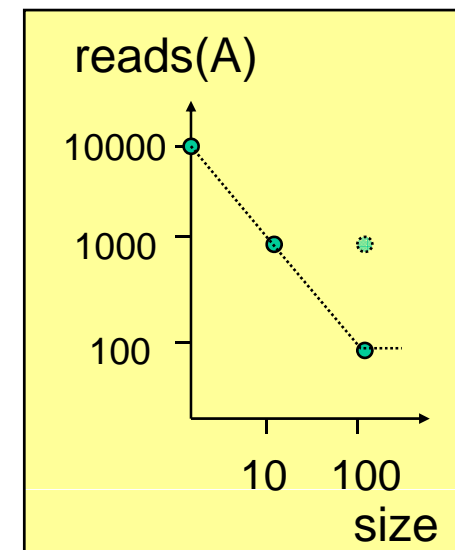
```
int A[250]
for (i=0; i<10; i++)
  for (j=0; j<10; j++)
    {A"[0..9]=A[j*10..j*10+9];
     for (k=0; k<10; k++)
       for (l=0; l<10; l++)
         f(A"[l])}
size=10; reads(A)=1000
```

Copy candidate

A', A" in small memory

```
int A[250]
for (i=0; i<10; i++)
  {A'[0..99]=A[0..99];
   for (j=0; j<10; j++)
     for (k=0; k<10; k++)
       for (l=0; l<10; l++)
         f(A'[j*10+l])}
size=100; reads(A)=1000
```

```
int A[250]
A'[0..99]=A[0..99];
for (i=0; i<10; i++)
  for (j=0; j<10; j++)
    for (k=0; k<10; k++)
      for (l=0; l<10; l++)
        f(A'[j*10+l])
size=100; reads(A)=100
```

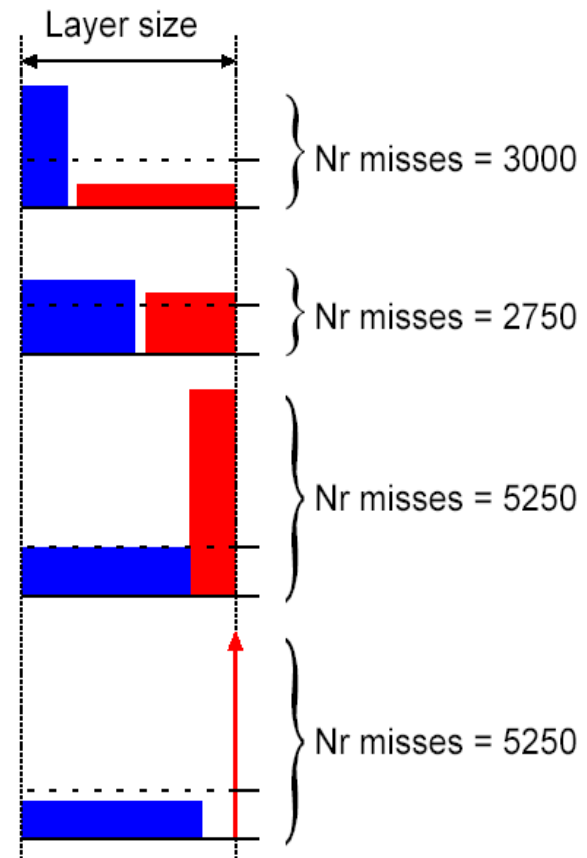
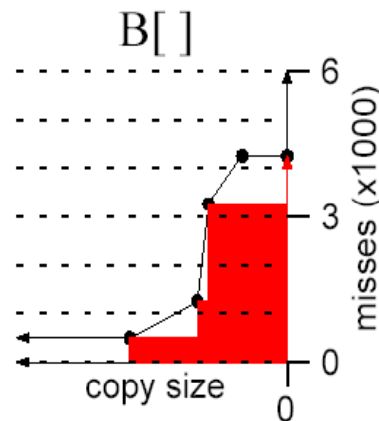
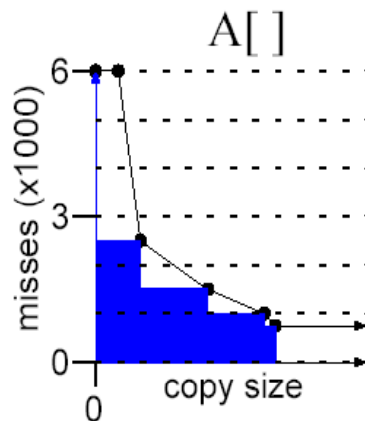


انتساب لایه‌های سلسله‌مراتب حافظه - هدف -

Memory hierarchy layer assignment (MHLA) - Goal -

Goal: For each variable: find permanent layer, partition and module & select copy candidates such that energy is minimized.

Conflicts between variables



Brockmeyer et al., DATE 2003



انتساب لایه‌های سلسله‌مراتب حافظه - رهیافت - Memory hierarchy layer assignment (MHLA) - Approach -

Approach:

- start with initial variable allocation
- incrementally improve initial solution such that total energy is minimized.

More general hardware architecture than the Dortmund approach, but no global optimization.



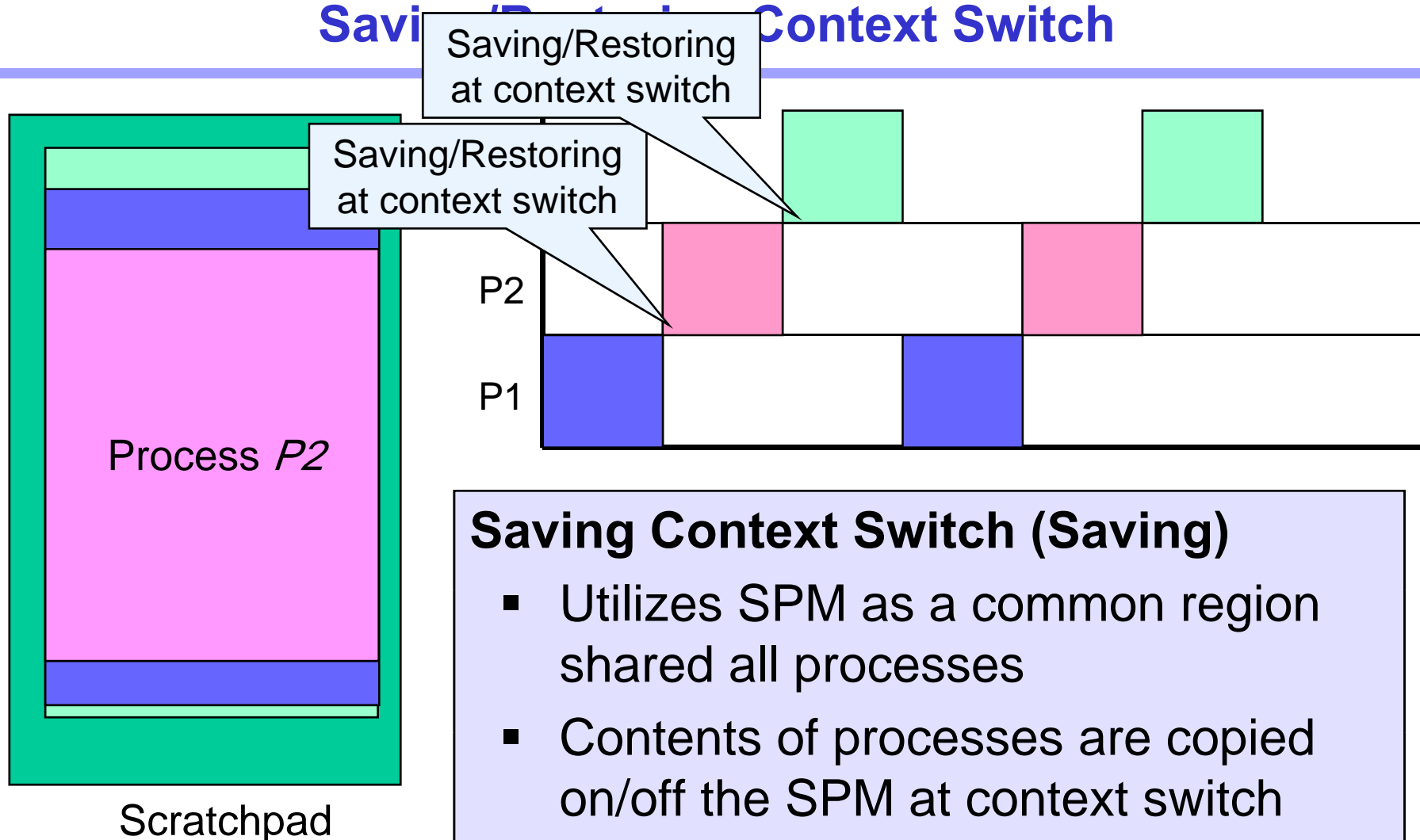
Work by Kandemir going in a similar direction

- M. Kandemir, I. Kadayif, A. Choudhary, J. Ramanujam, and I. Kolcu: Compiler-directed scratch pad memory optimization for embedded multiprocessors, *IEEE Transactions on VLSI (TVLSI)*, pp. 281-287, 2004
- M. Kandemir, J. Ramanujam, M. J. Irwin, N. Vijaykrishnan, I. Kadayif, and A. Parikh: A compiler based approach for dynamically managing scratch-pad memories in embedded systems, *IEEE Transactions on CAD (TCAD)*, pp. 243-260, 2004



تعویض متن با ذخیره / بازیابی

Saving/Restoring Context Switch

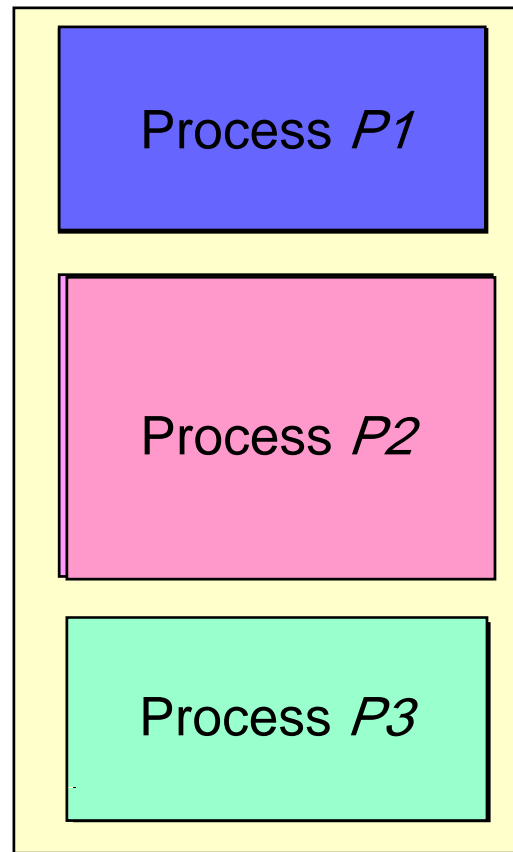


Saving Context Switch (Saving)

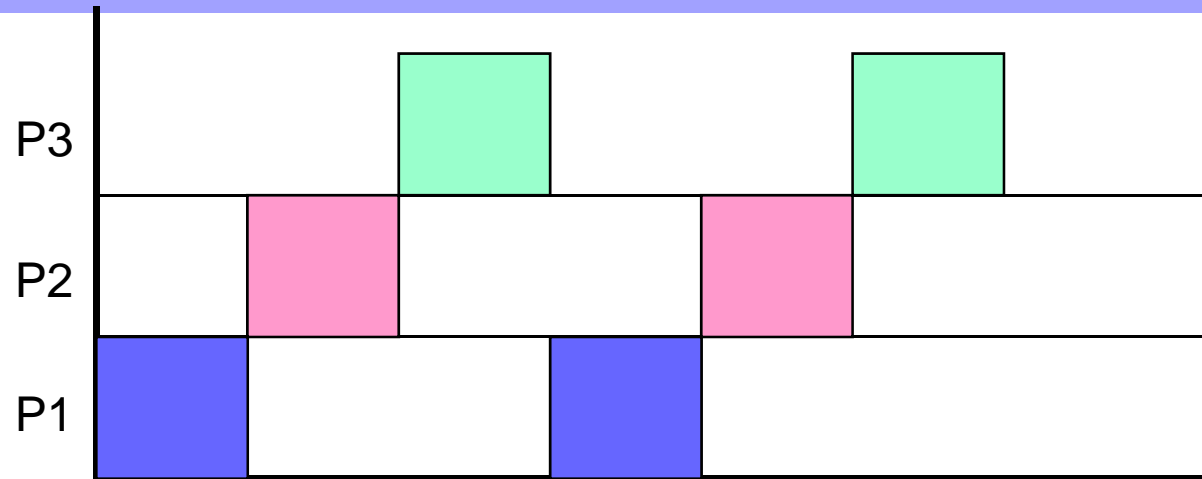
- Utilizes SPM as a common region shared all processes
- Contents of processes are copied on/off the SPM at context switch
- Good for small scratchpads

تعويض متن بدون ذخيره

Non-Saving Context Switch



Scratchpad



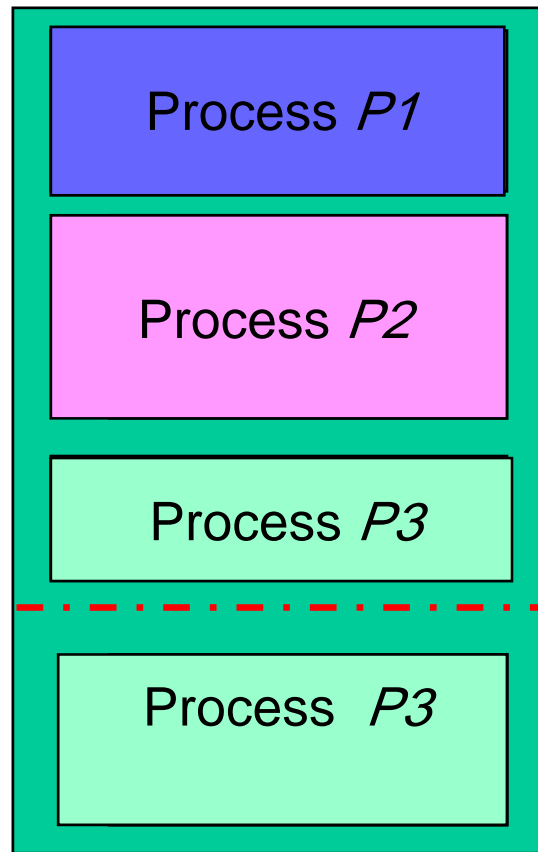
Non-Saving Context Switch (Non-Saving)

- Partitions SPM into disjoint regions
- Each process is assigned a SPM region
- Copies contents during initialization
- Good for large scratchpads

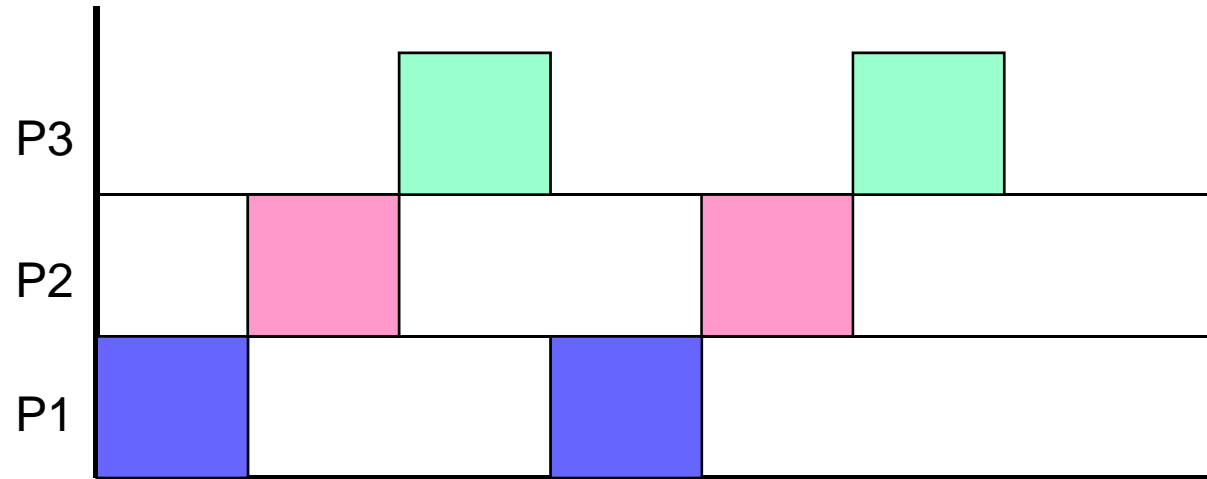


تعویض متن آمیخته

Hybrid Context Switch



Scratchpad



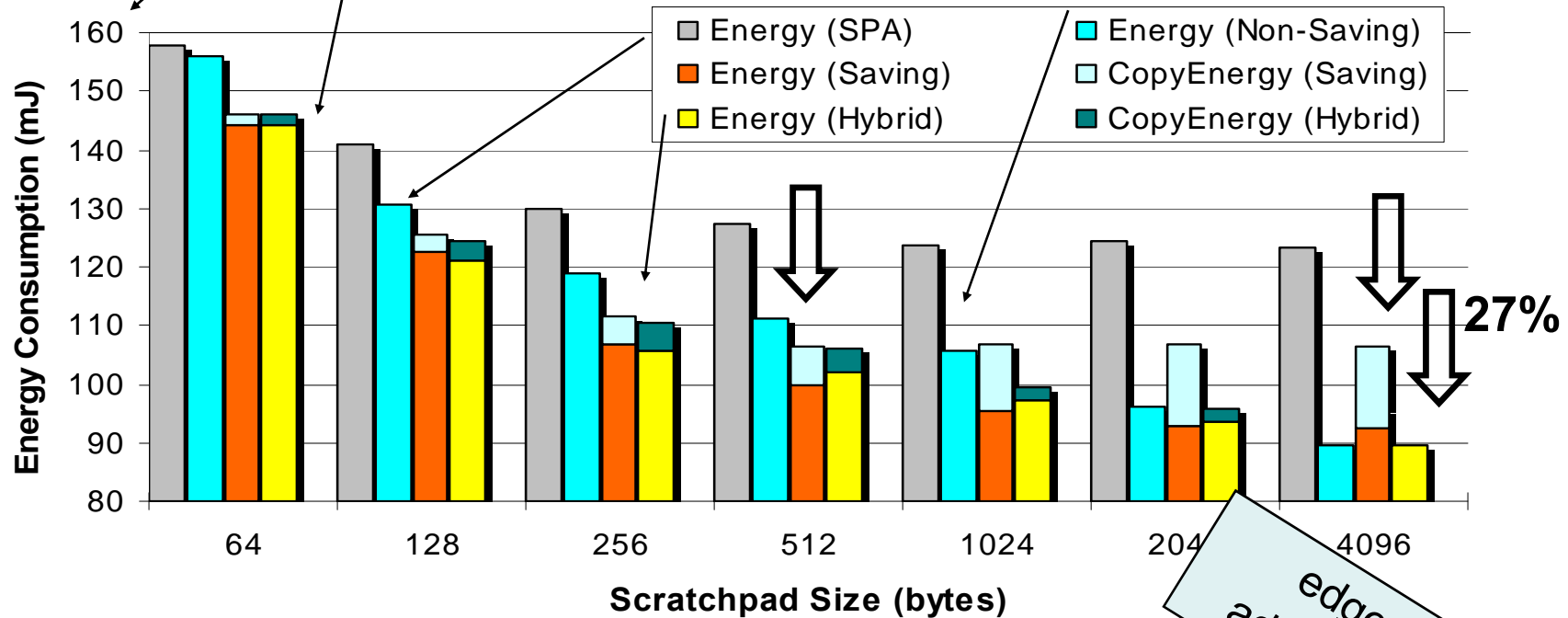
Hybrid Context Switch (Hybrid)

- Disjoint + Shared SPM regions
- Good for all scratchpads
- Analysis is similar to Non-Saving Approach
- Runtime: $O(nM^\beta)$



Multi-process Scratchpad Allocation: Results

SPA: Single Process Approach

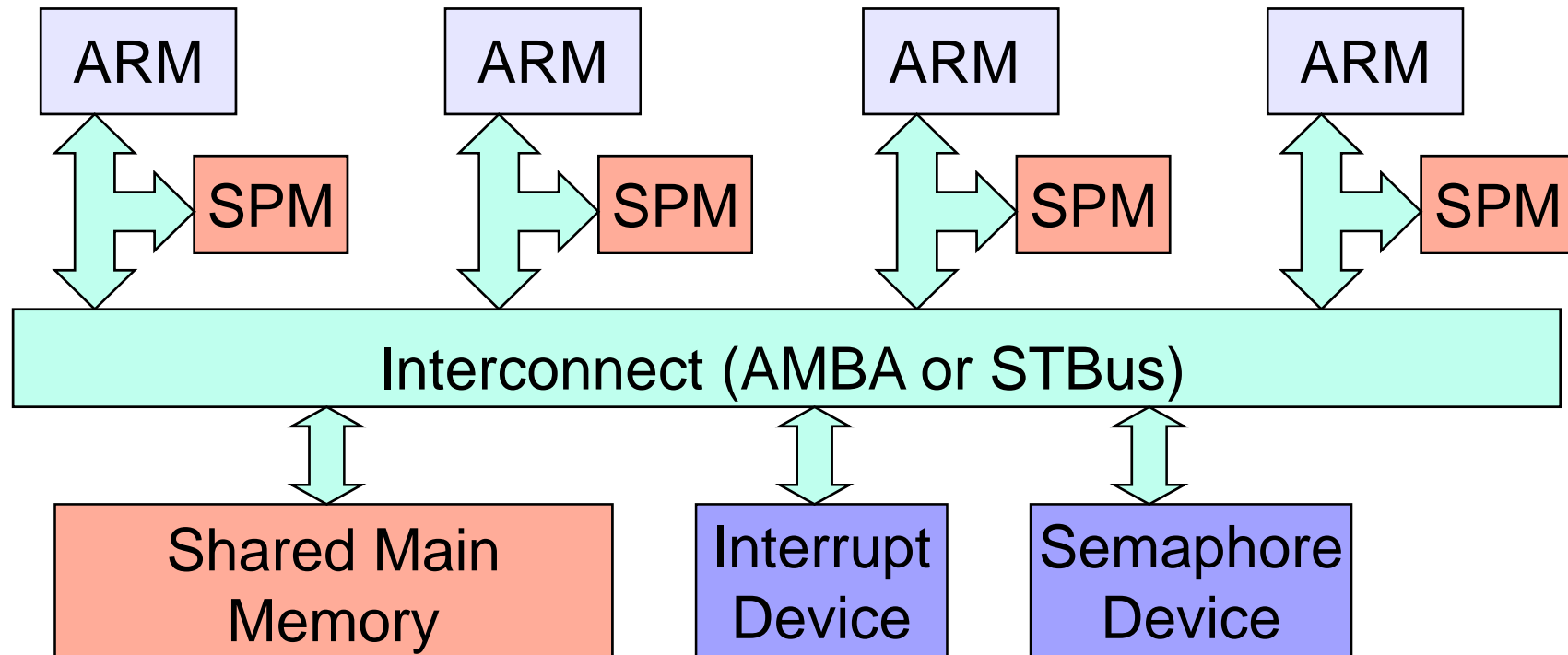


- For small SPMs (64B-512B) Saving is better
- For large SPMs (1kB- 4kB) Non-Saving is better
- Hybrid is the best for all SPM sizes.
- Energy reduction @ 4kB SPM is 27% for Hybrid approach

edge detection,
adpcm, g721, mpeg



Multi-processor ARM (MPARM) Framework



- Homogenous SMP ~ CELL processor
- Processing Unit : ARM7T processor
- Shared Coherent Main Memory
- Private Memory: Scratchpad Memory



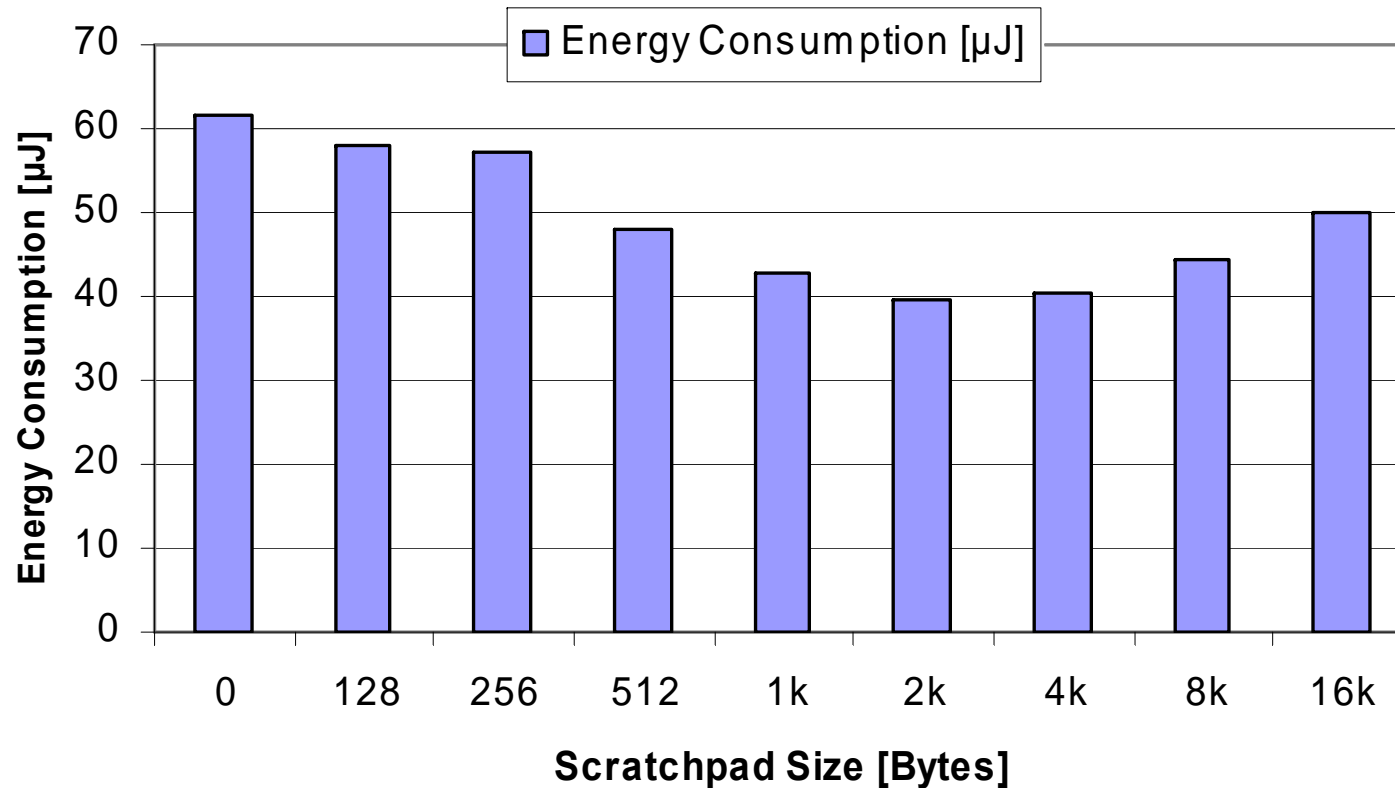
Slide 49

v2

Change the colors of the graphic and move private memories close to the processor

verma; 09/12/2005

Results (MOMPARM)



DES-Encryption: 4 processors: 2 Controllers+2 Compute Engines

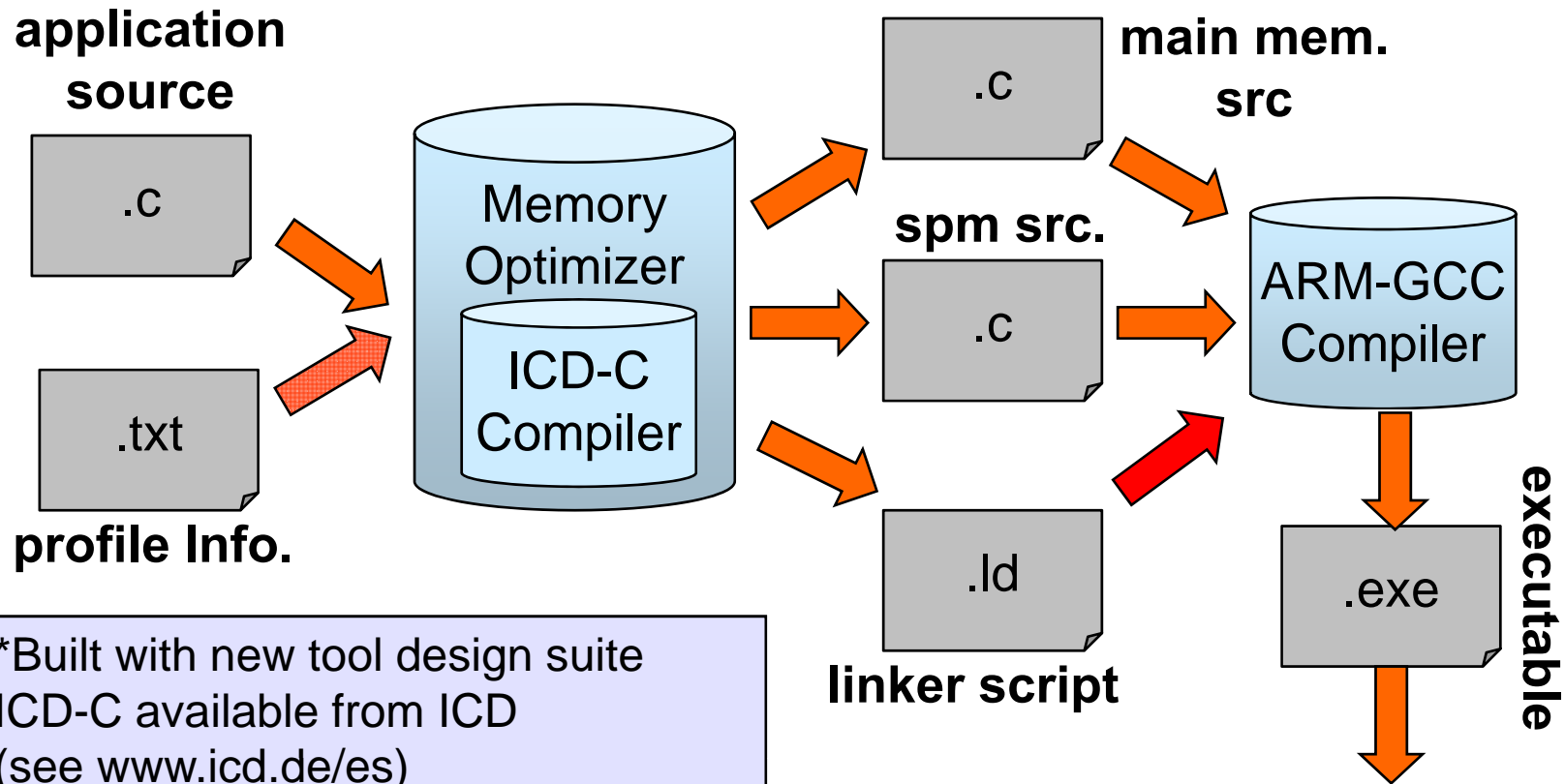
Energy values from
ST Microelectronics

Result of ongoing cooperation between U. Bologna and U.
Dortmund supported by ARTIST2 network of excellence.



Using these ideas with an gcc-based tool flow

Source is split into 2 different file by specially developed memory optimizer tool *.



*Built with new tool design suite ICD-C available from ICD (see www.icd.de/es)



Extensions

- Using DMA for copying blocks
 - Using DRAM
 - Applications to Flash memory
(copy code or execute in place):
according to own experiments: very much parameter
dependent
 - Multiprocessor systems
 - Freezing caches
 - Windows into large arrays
- } PhD thesis of
Lars
Wehmeyer

Acknowledgement: Credit goes to all students involved in the design of SPM tools at Dortmund, in particular to Stefan Steinke, Manish Verma and Lars Wehmeyer.



نتیجه گیری

Conclusion

Impact of memory architecture on execution times & energy.

The SPM provides

- Runtime efficiency
- Energy efficiency
- Timing predictability

Achieved savings are sometimes dramatic, for example:

- savings of ~ 95% of the memory system energy

Savings possibly even larger for larger applications.

