technische universität
dortmund

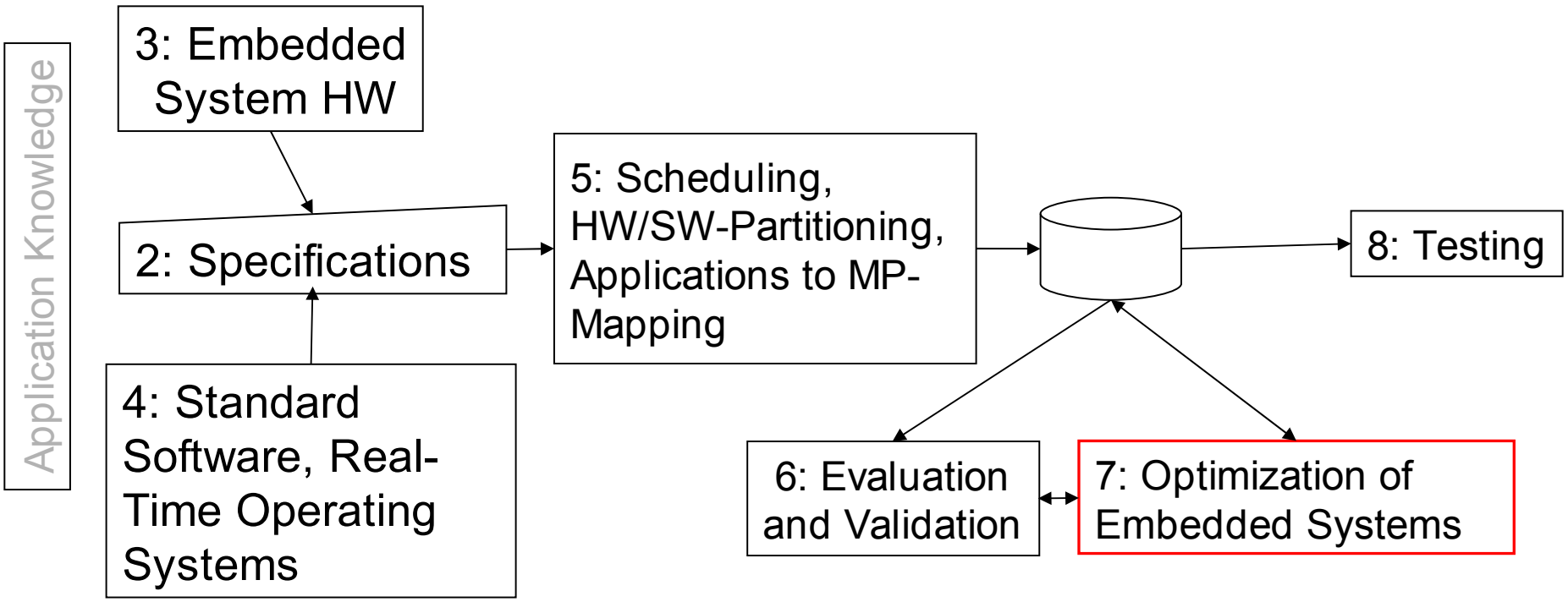
fakultät für informatik
informatik 12

Optimizations

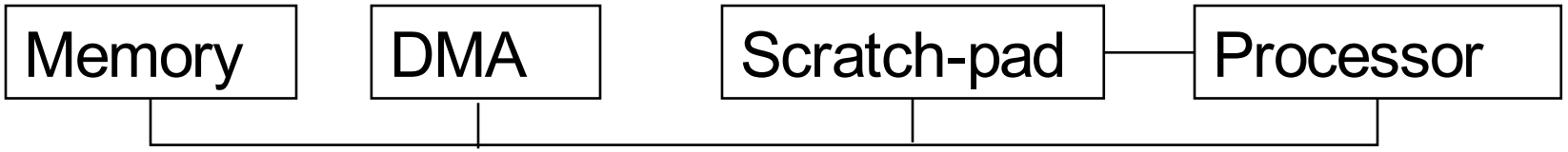
Peter Marwedel
TU Dortmund
Informatik 12
Germany

2009/01/17

Structure of this course



Hardware-support for block-copying



The DMA unit was modeled in VHDL, simulated, synthesized. Unit only makes up 4% of the processor chip.

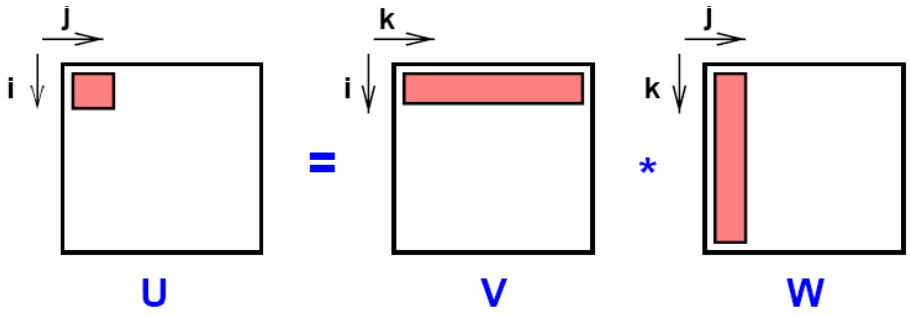
The unit can be put to sleep when it is unused.

Code size reductions of up to 23% for a 256 byte SPM were determined using the DMA unit instead of the overlaying allocation that uses processor instructions for copying.

[Lars Wehmeyer, Peter Marwedel: Fast, Efficient and Predictable Memory Accesses, *Springer*, 2006]

References to large arrays (1)
- Regular accesses -

```
for (i=0; i<n; i++)
  for (j=0; j<n; j++)
    for (k=0; k<n; k++)
      U[i][j]=U[i][j] + V[i][k] * W[k][j]
```



Tiling ☞

```
for (it=0; it<n; it=it+Sb)
  {read_tile V[it:it+Sb-1, 1:n]
  for (jt=0; jt<n; jt=jt+Sb)
    {read_tile U[it:it+Sb-1, jt:jt+Sb-1]
     read_tile W[1:n,jt:jt+Sb-1]
     U[it:it+Sb-1,jt:jt+Sb-1]=U[it:it+Sb-1,jt:jt+Sb-1]
                              + V[it:it+Sb-1,1:n]
                              * W [1:n, jt:jt+Sb-1]
        write_tile U[it:it+Sb-1,jt:jt+Sb-1]
   }}
```

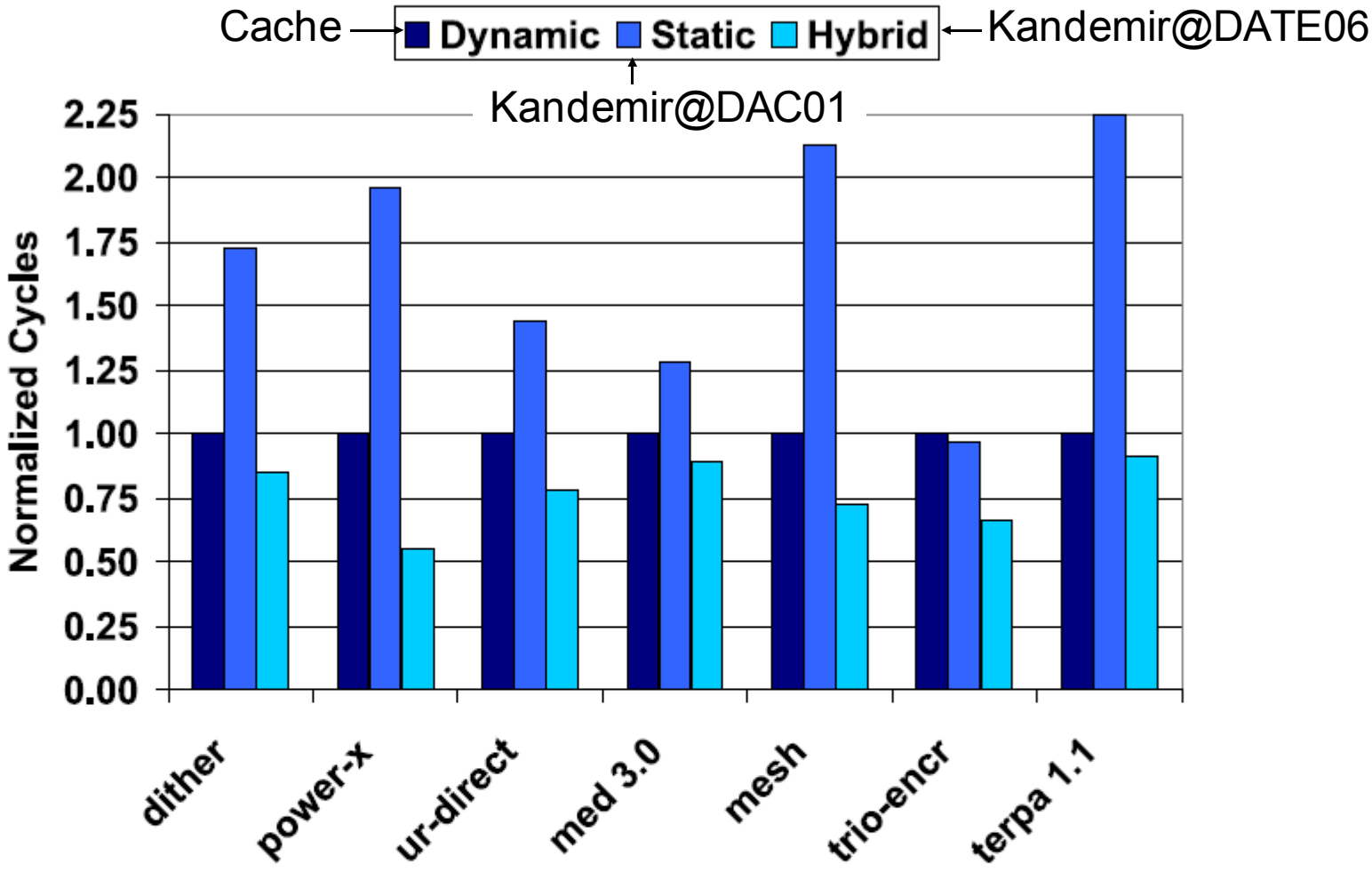
[M. Kandemir, J. Ramanujam, M. J. Irwin, N. Vijaykrishnan, I. Kadayif, A. Parikh: Dynamic Management of Scratch-Pad Memory Space, *DAC*, 2001, pp. 690-695]

References to large arrays
- Irregular accesses -

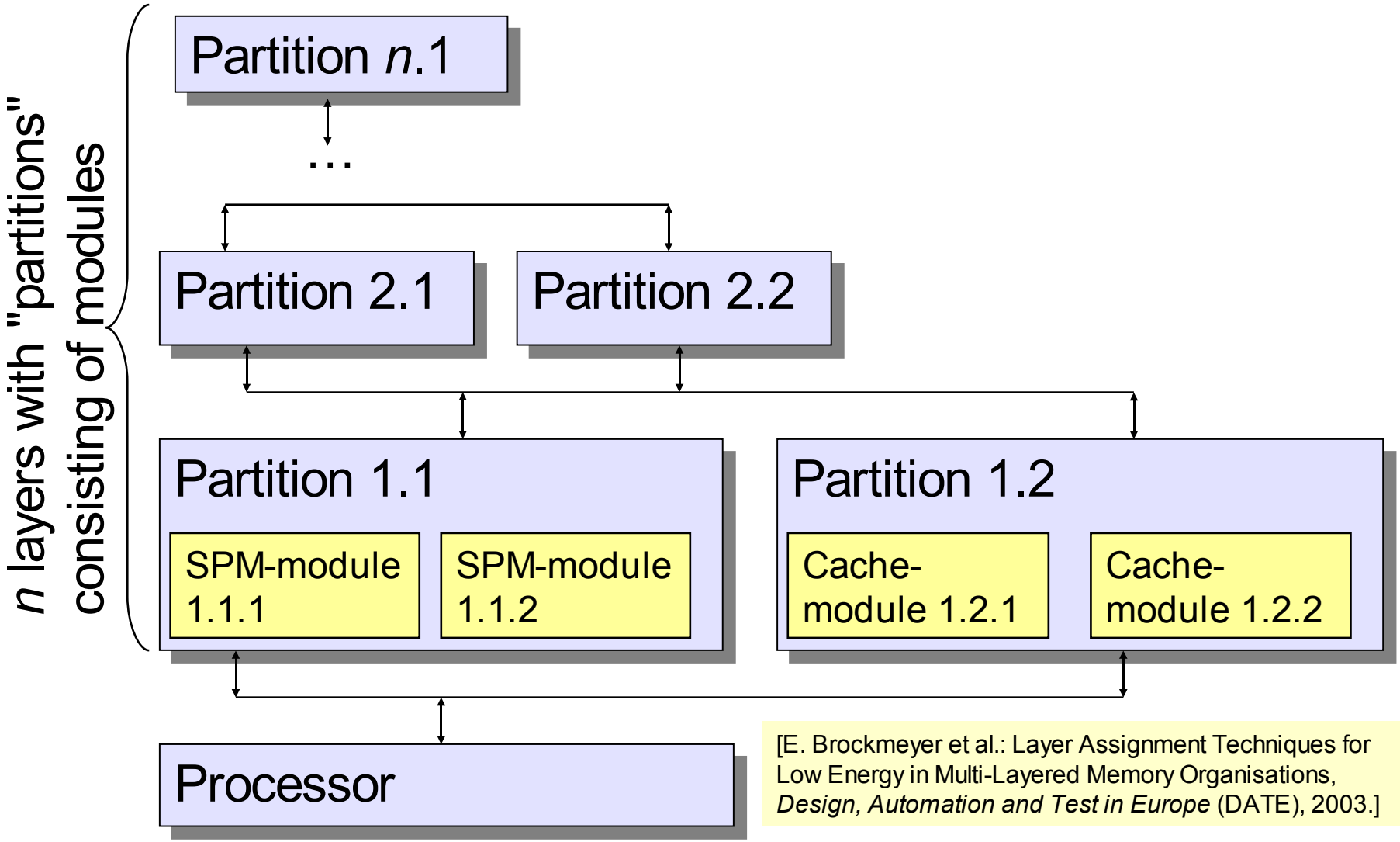
```
for each loop nest L in program P {
   apply loop tiling to L based on the access patterns of
      regular array references;
   for each assignment to index array X
     update the block minimum and maximum values of X;
   compute the set of array elements that are irregularly
     referenced in the current inter-tile iteration;
   compare the memory access costs for using
    and not using SPM;
   if (using SPM is beneficial)
     execute the intra-tile loop iterations by using the SPM
   else
     execute the intra-tile loop iterations by not
     using the SPM
}
```

[G. Chen, O. Ozturk, M. Kandemir, M. Karakoy: Dynamic Scratch-Pad Memory Management for Irregular Array Access Patterns, *DATE*, 2006]

Results for irregular approach



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



[E. Brockmeyer et al.: Layer Assignment Techniques for Low Energy in Multi-Layered Memory Organisations, *Design, Automation and Test in Europe* (DATE), 2003.]

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
```

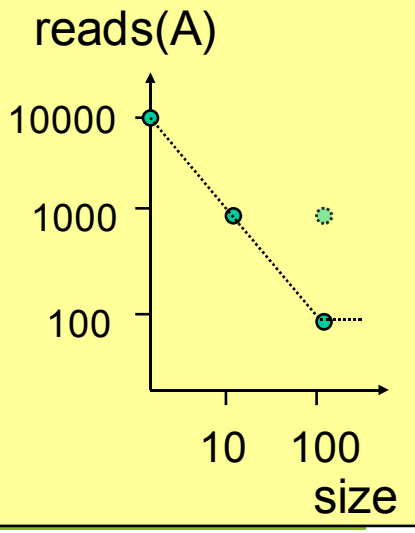
```
int A[250]
for (i=0; i<10; i++)
 for (j=0; j<10; j++)
 {A"[0..9]=A[j*10..j*10+9];
  for (k=0; k<10; k++)
   for (l=0; l<10; l++)
    f(A"[l])}
size=10; reads(A)=1000
```

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
```

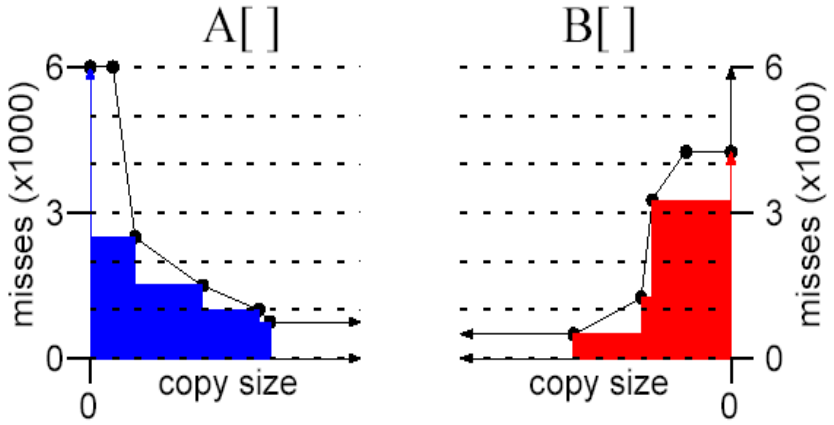
```
int A[250]
A'[0..99]=A[0..99];
for (i=0; i<10; i++)
 for (j=0; j<10; j++)
  for (k=0; k<10; k++)
   for (l=0; l<10; l++)
    f(A'[j*10+l])
size=100; reads(A)=100
```

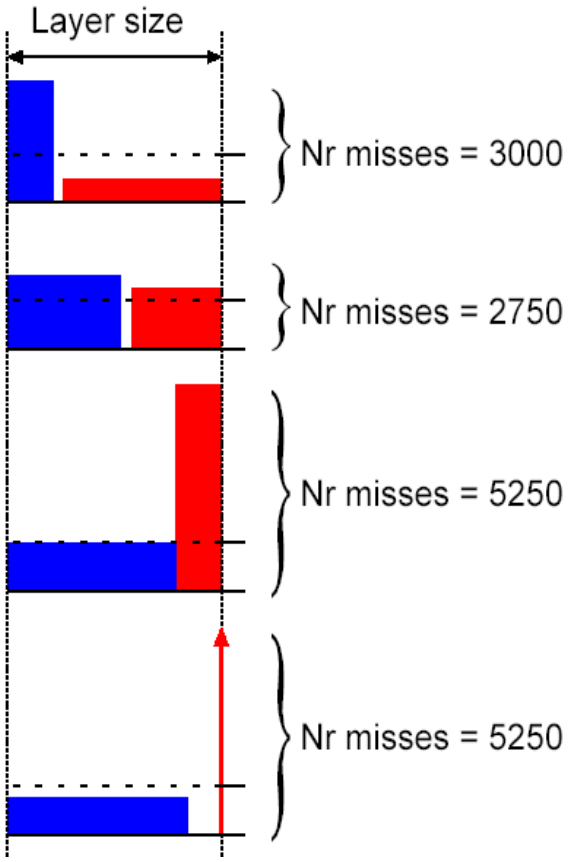


Memory hierarchy layer assignment (MHLA) - Goal -

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

Conflicts between variables





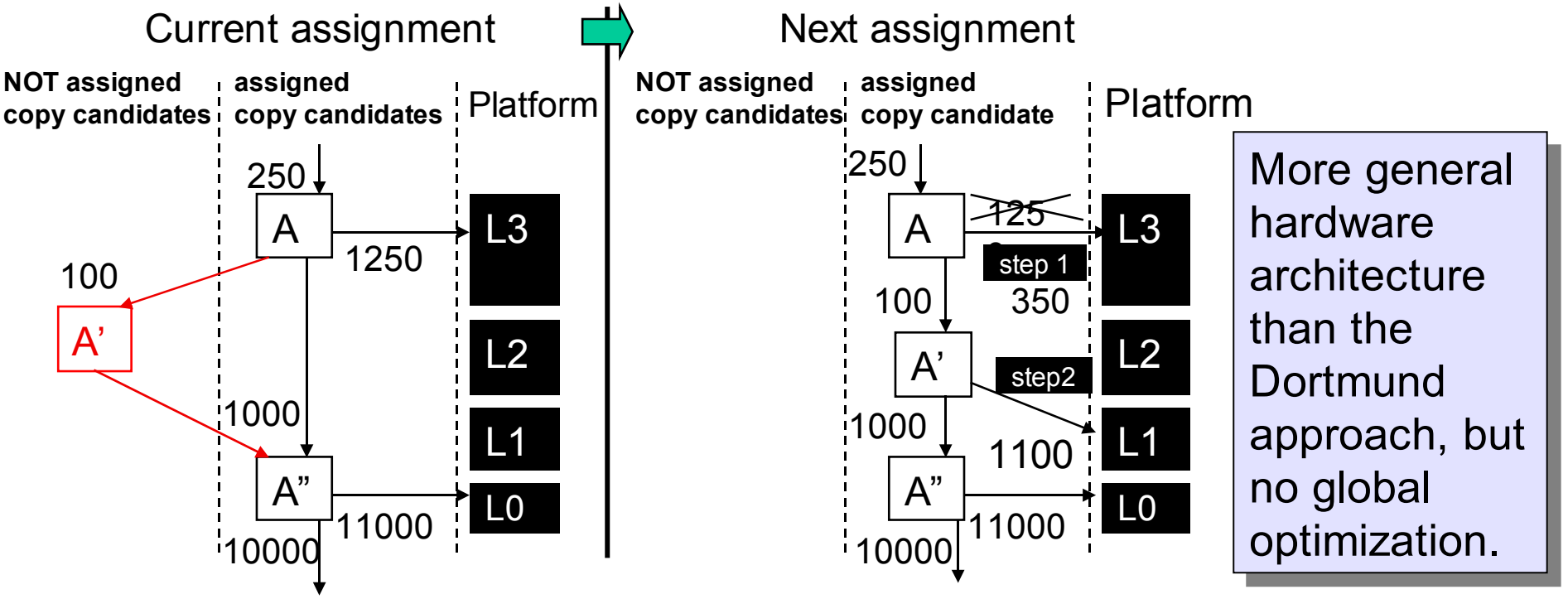
[E. Brockmeyer et al.: Layer Assignment Techniques for Low Energy in Multi-Layered Memory Organisations, *Design, Automation and Test in Europe* (DATE), 2003.]

Memory hierarchy layer assignment (MHLA) - Approach -

Approach:

- start with initial variable allocation
- incrementally improve initial solution

such that total energy is minimized.



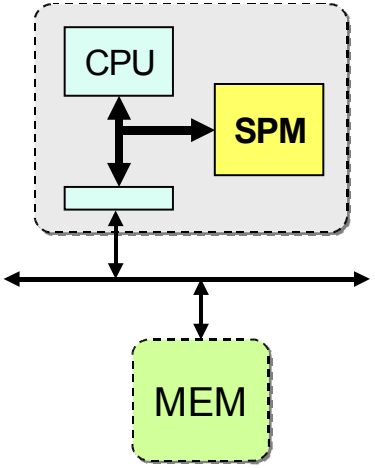
More general hardware architecture than the Dortmund approach, but no global optimization.

Dynamic set of multiple applications

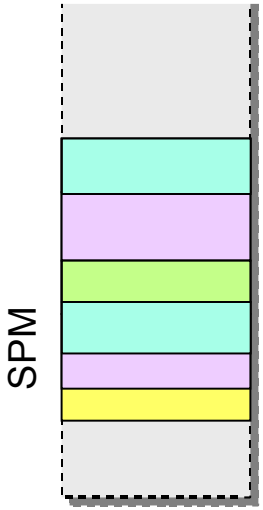
Compile-time partitioning of SPM no longer feasible

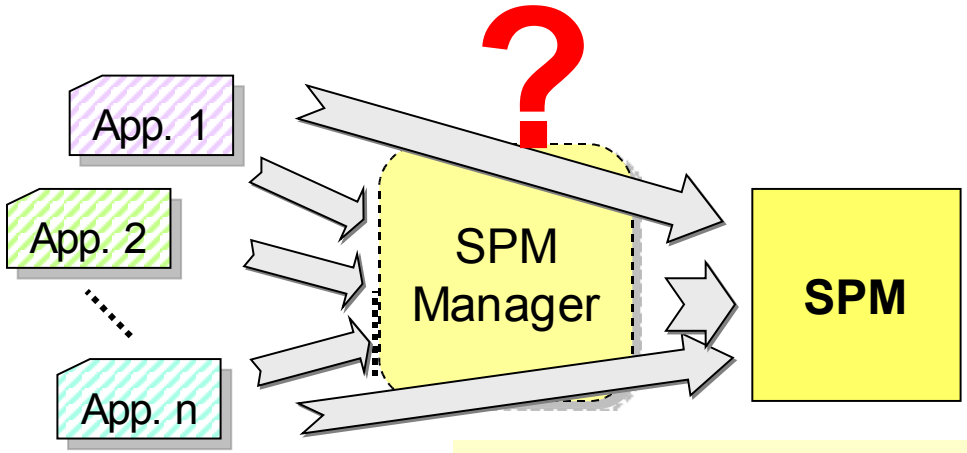
☞ Introduction of SPM-manager

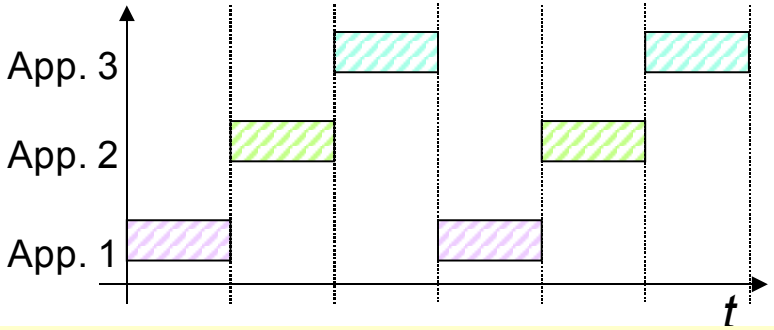
- Runtime decisions, but compile-time supported



Address space:



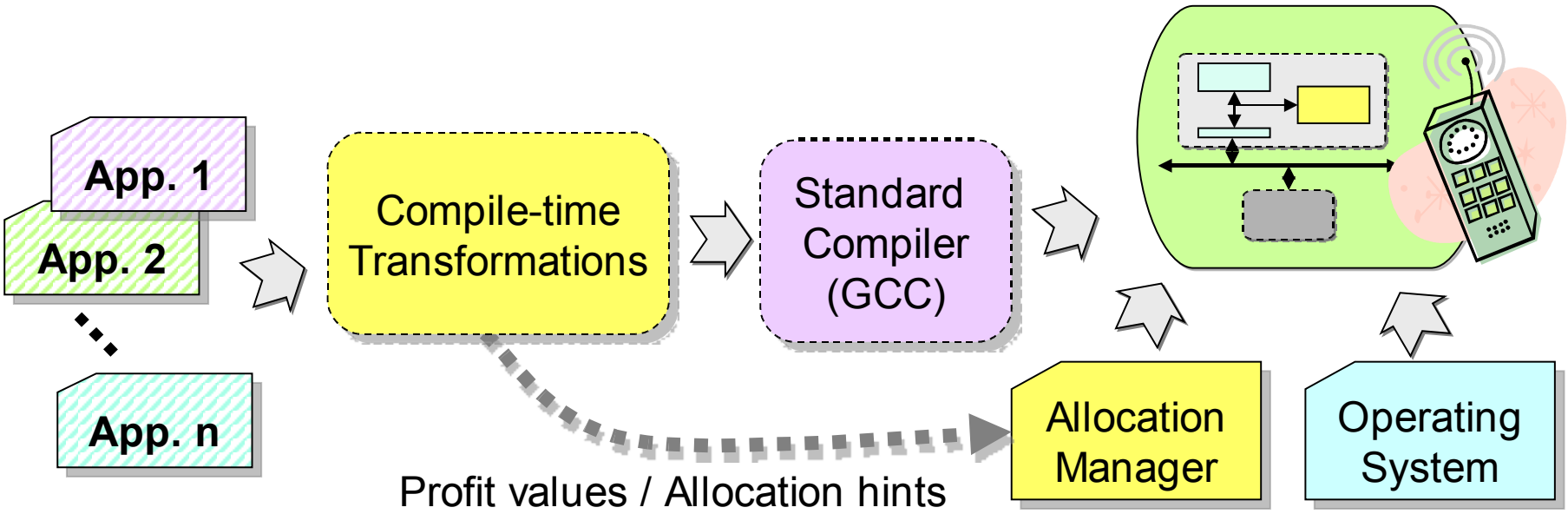




[R. Pyka, Ch. Faßbach, M. Verma, H. Falk, P. Marwedel: Operating system integrated energy aware scratchpad allocation strategies for multi-process applications, *SCOPES*, 2007]

Approach overview

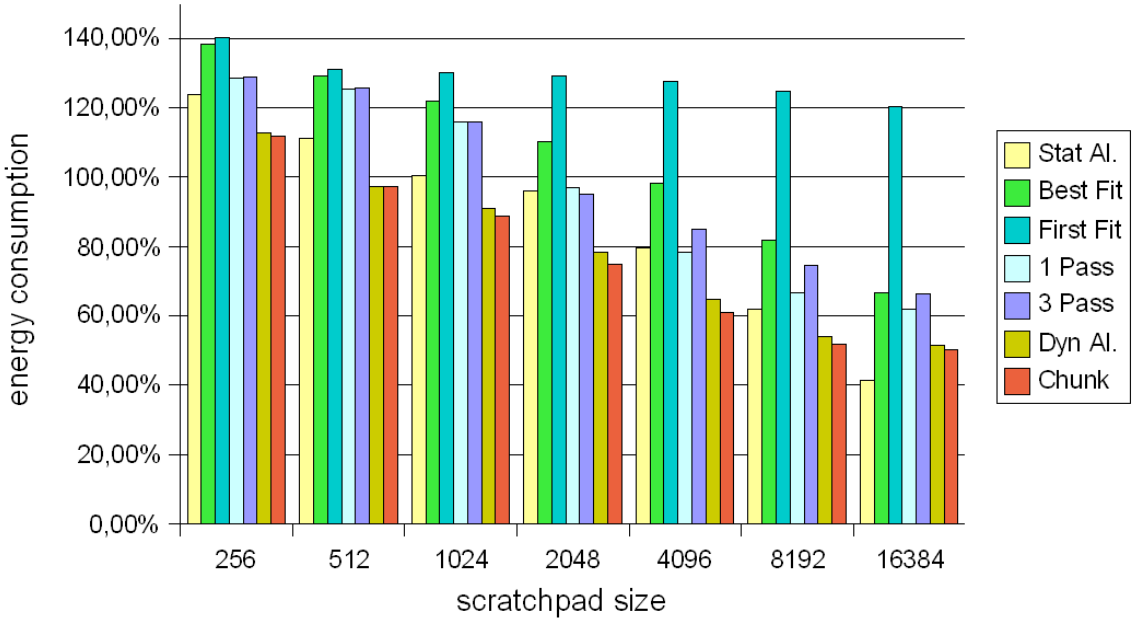
- 2 steps: compile-time analysis & runtime decisions
- No need to know all applications at compile-time
- Capable of managing runtime allocated memory objects
- Integrates into an embedded operating system

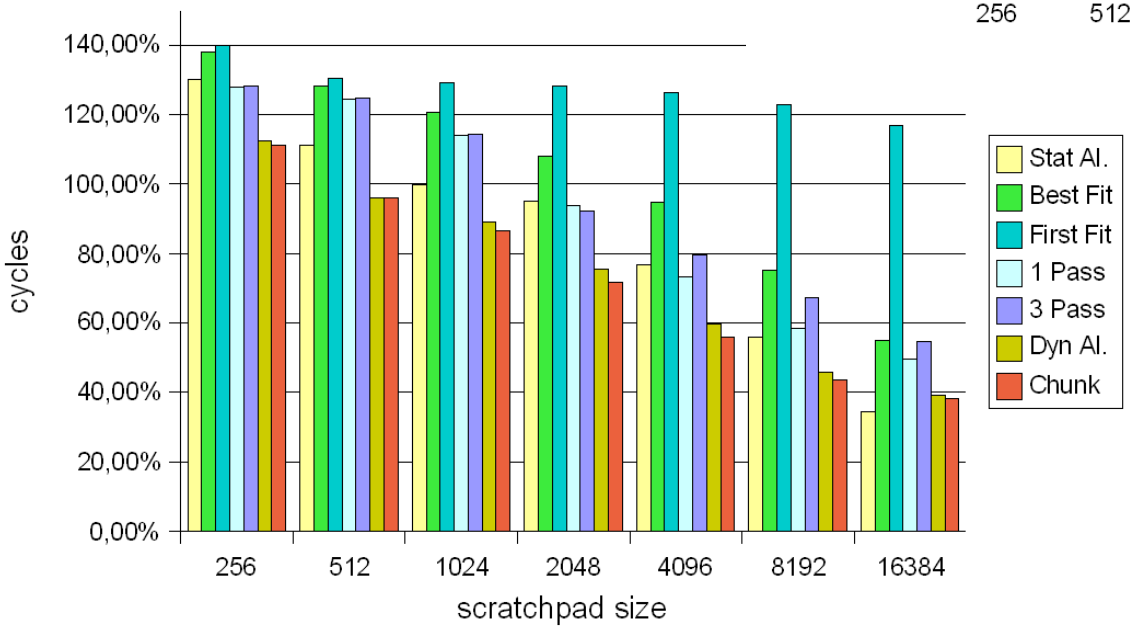


Using MPArm simulator from U. Bologna

Results

- **MEDIA+ Energy**
  - **Baseline: Main memory only**
  - **Best: Static for 16k → 58%**
  - **Overall best: Chunk → 49%**





- **MEDIA+ Cycles**
  - **Baseline: Main memory only**
  - **Best: Static for 16k → 65%**
  - **Overall best: Chunk → 61%**

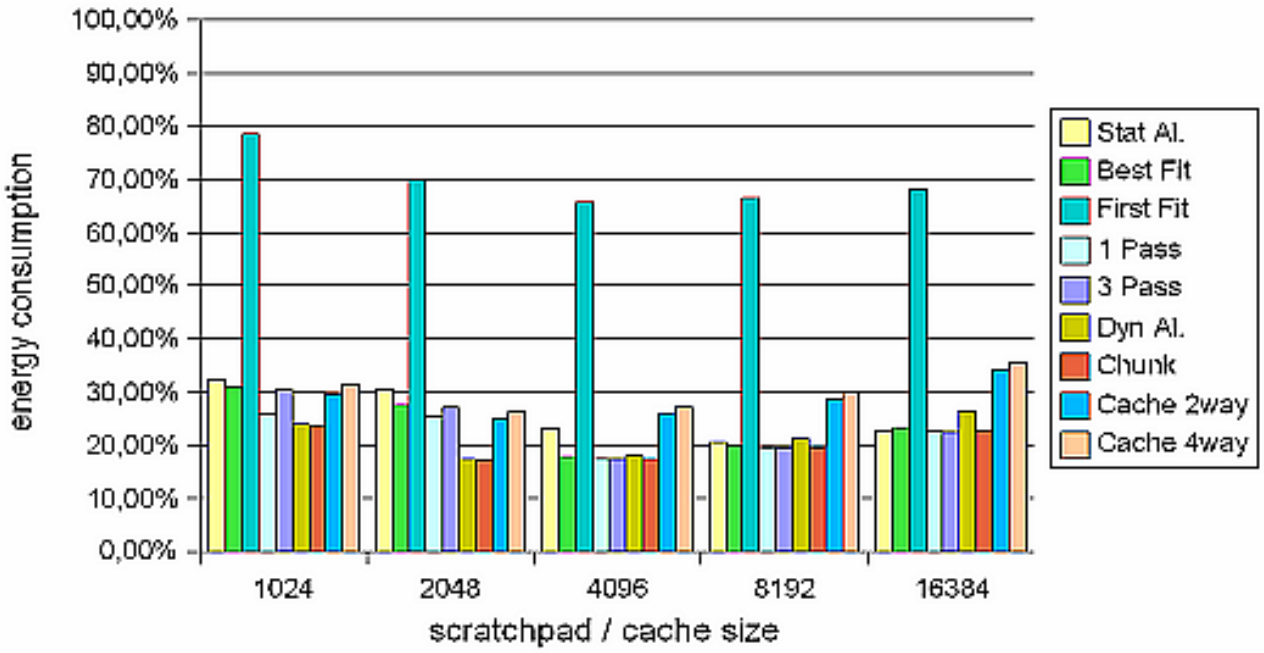
Comparison of SPMM to Caches for SORT

- 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:

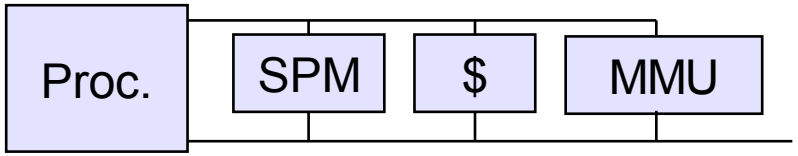
SPM Size	Δ 4-way
1024	74,81%
2048	65,35%
4096	64,39%
8192	65,64%
16384	63,73%



SPM+MMU (1)

How to use SPM in a system with virtual addressing?

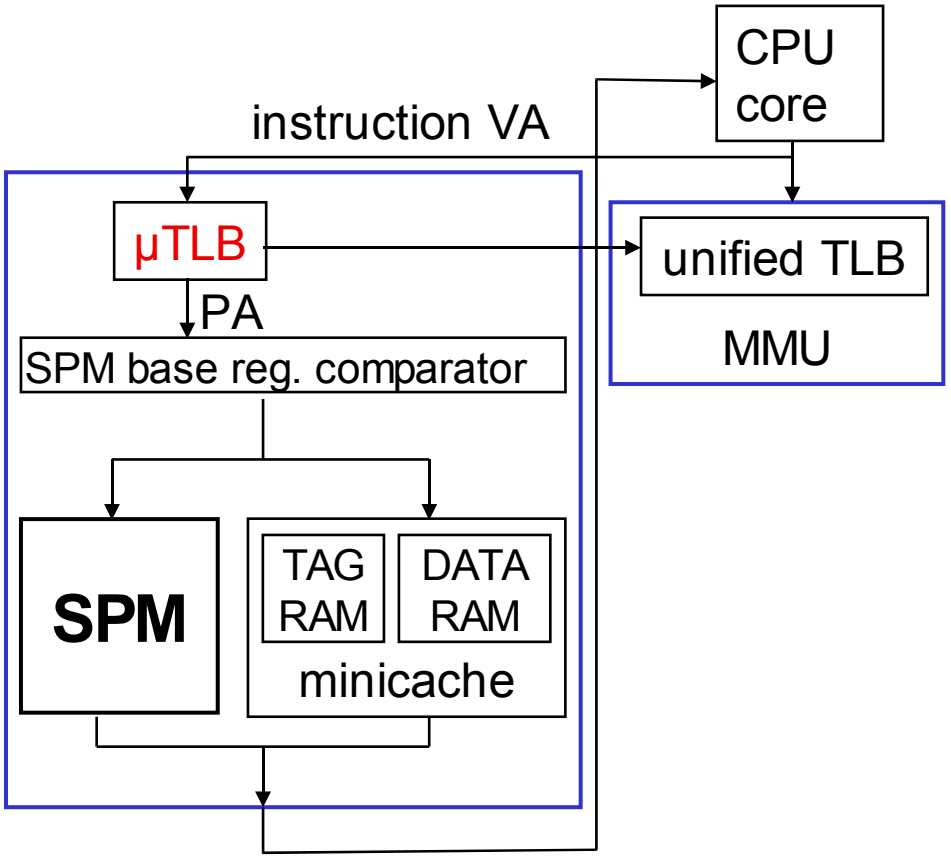
- **Virtual SPM**
  Typically accesses MMU
  + SPM in parallel
  ☞ not energy efficient
- **Real SPM**
  ☞ suffers from potentially long VA translation
- Egger, Lee, Shin (Seoul Nat. U.): Introduction of small **µTLB** translating recent addresses fast.



[B. Egger, J. Lee, H. Shin: Scratchpad memory management for portable systems with a memory management unit, *CASES*, 2006, p. 321-330 (best paper)]

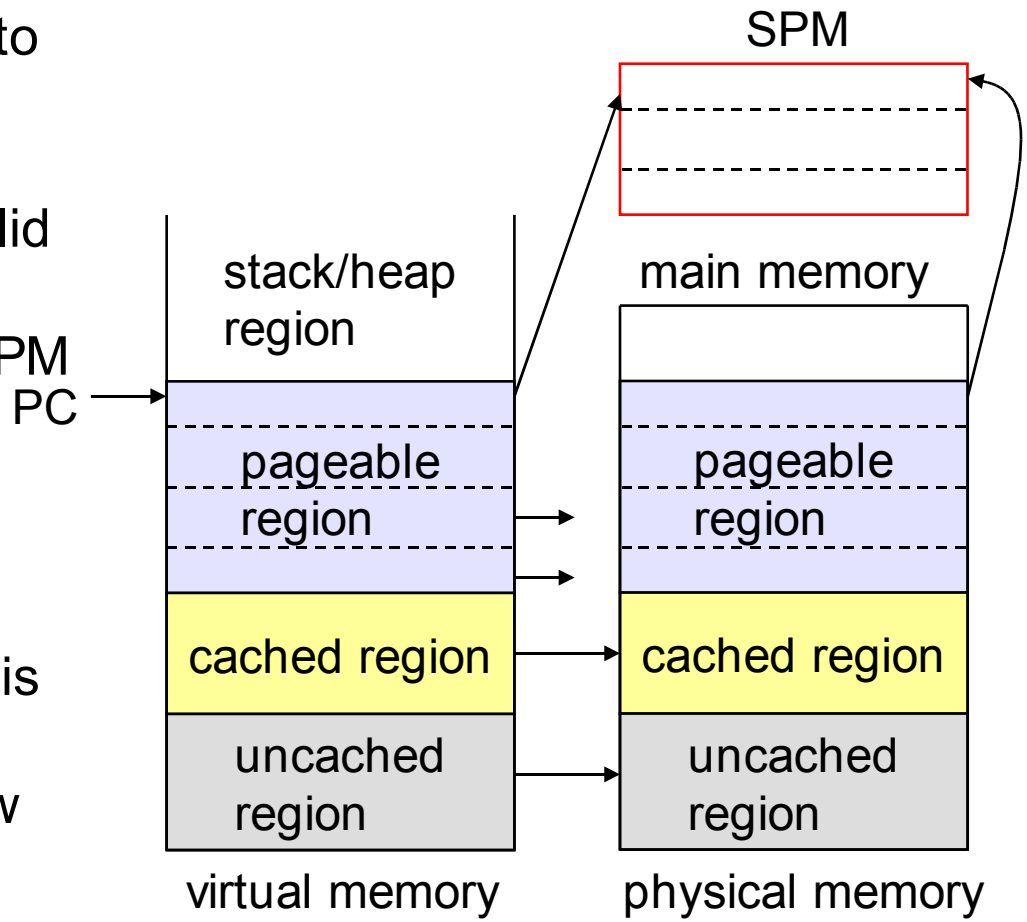
SPM+MMU (2)

- µTLB generates physical address in 1 cycle
- if address corresponds to SPM, it is used
- otherwise, mini-cache is accessed
- Mini-cache provides reasonable performance for non-optimized code
- µTLB miss triggers main TLB/MMU
- SPM is used only for instructions
- instructions are stored in pages
- pages are classified as cacheable, non-cacheable, and "pageable" (= suitable for SPM)

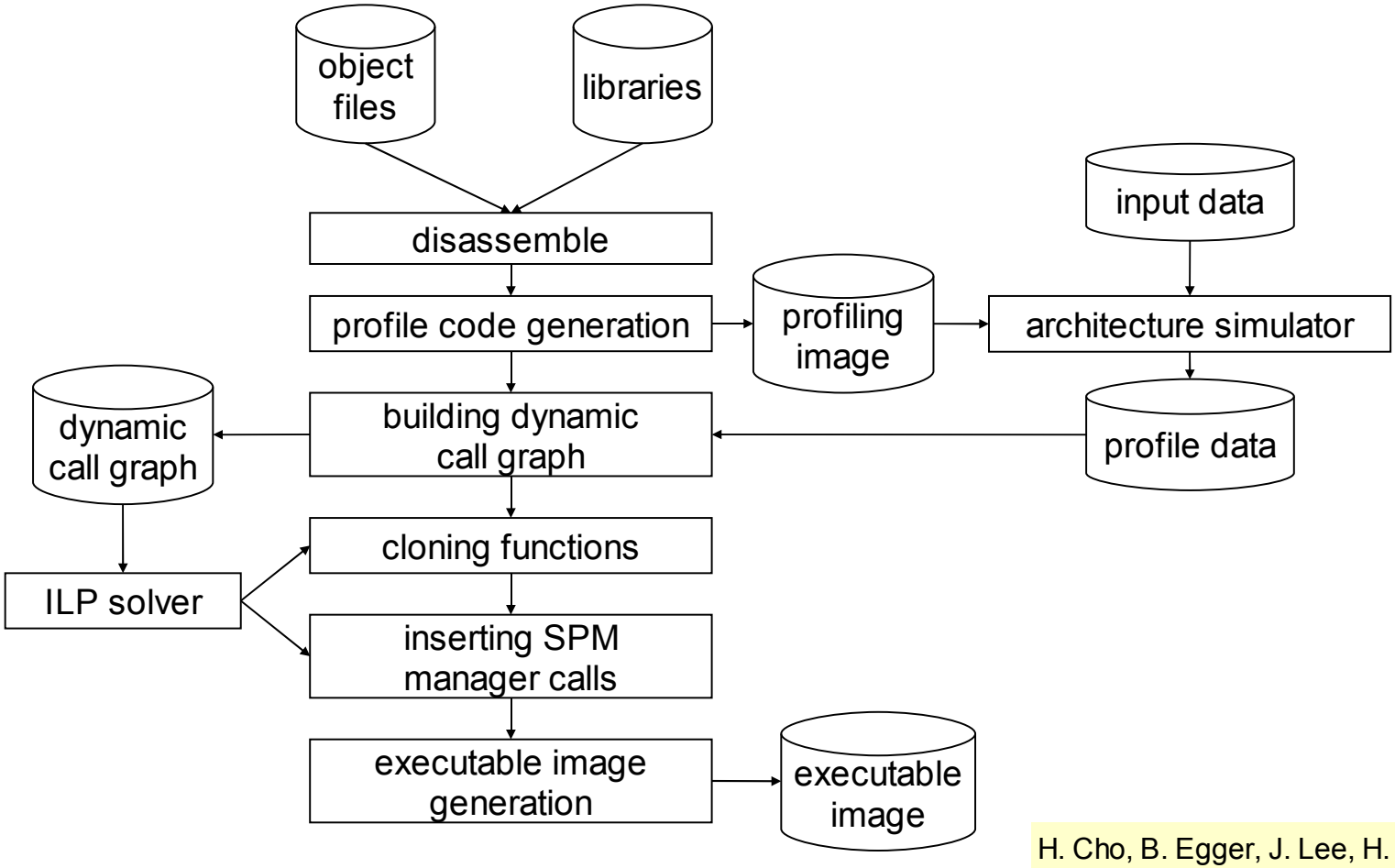


SPM+MMU (3)

- Application binaries are modified: frequently executed code put into pageable pages.
- Initially, page-table entries for pageable code are marked invalid
- If invalid page is accessed, a *page table exception* invokes SPM manager (SPMM).
- SPMM allocates space in SPM and sets page table entry
- If SPMM detects more requests than fit into SPM, SPM eviction is started
- Compiler does not need to know SPM size

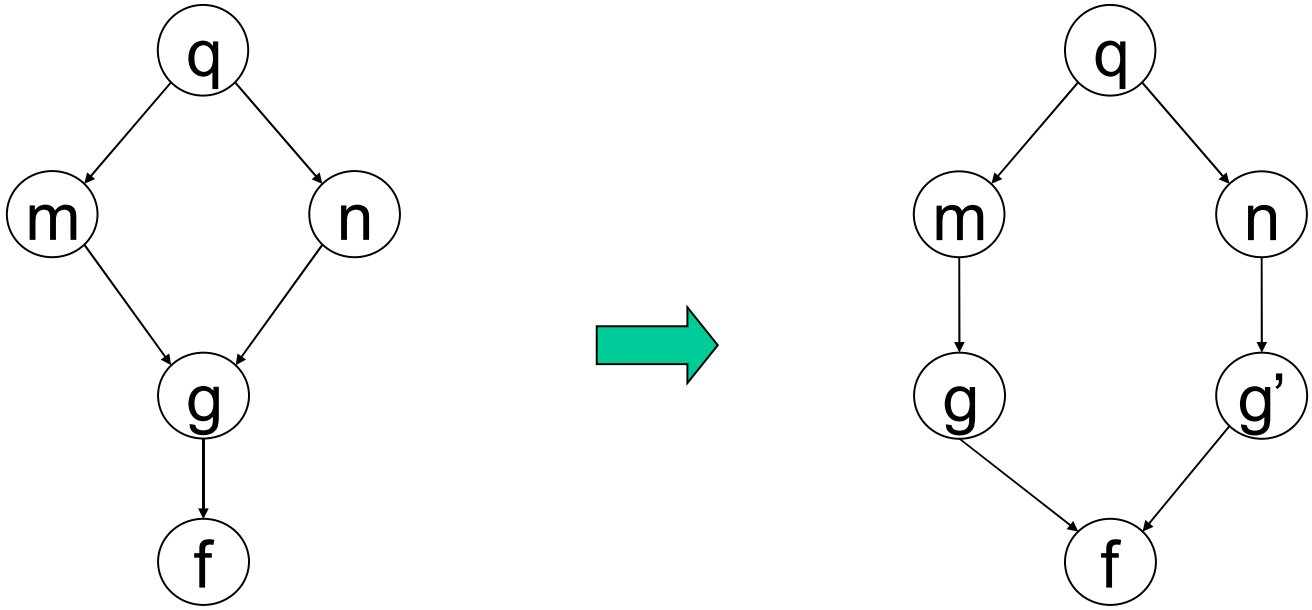


Extension to SNACK-pop (post-pass optimization)



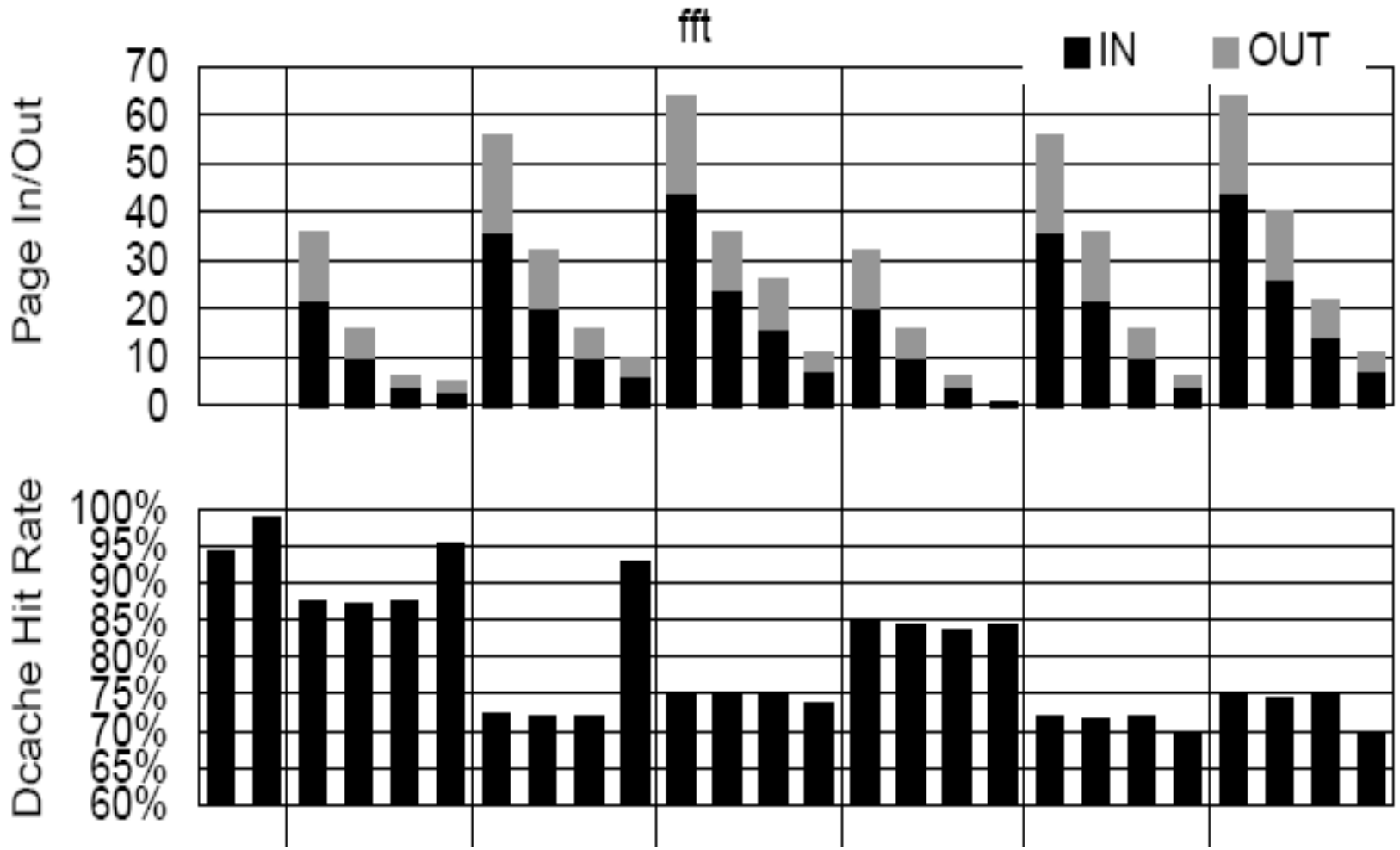
H. Cho, B. Egger, J. Lee, H. Shin: Dynamic Data Scratchpad Memory Management for a Memory Subsystem with an MMU, LCTES, 2007

Cloning of functions

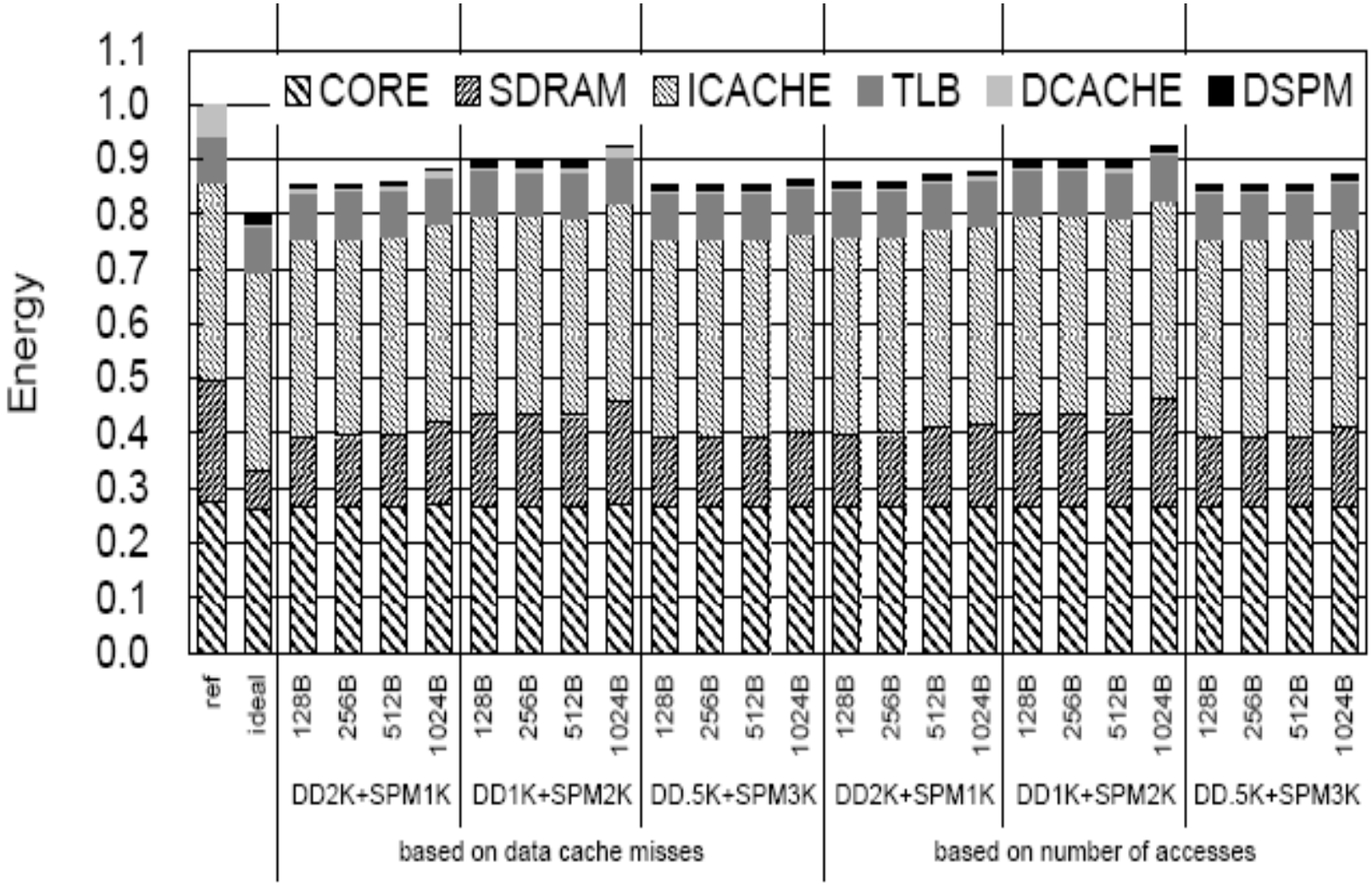


- Computation of which block should be moved in and out for a certain edge
- Generation of an ILP
- Decision about copy operations at compile time.

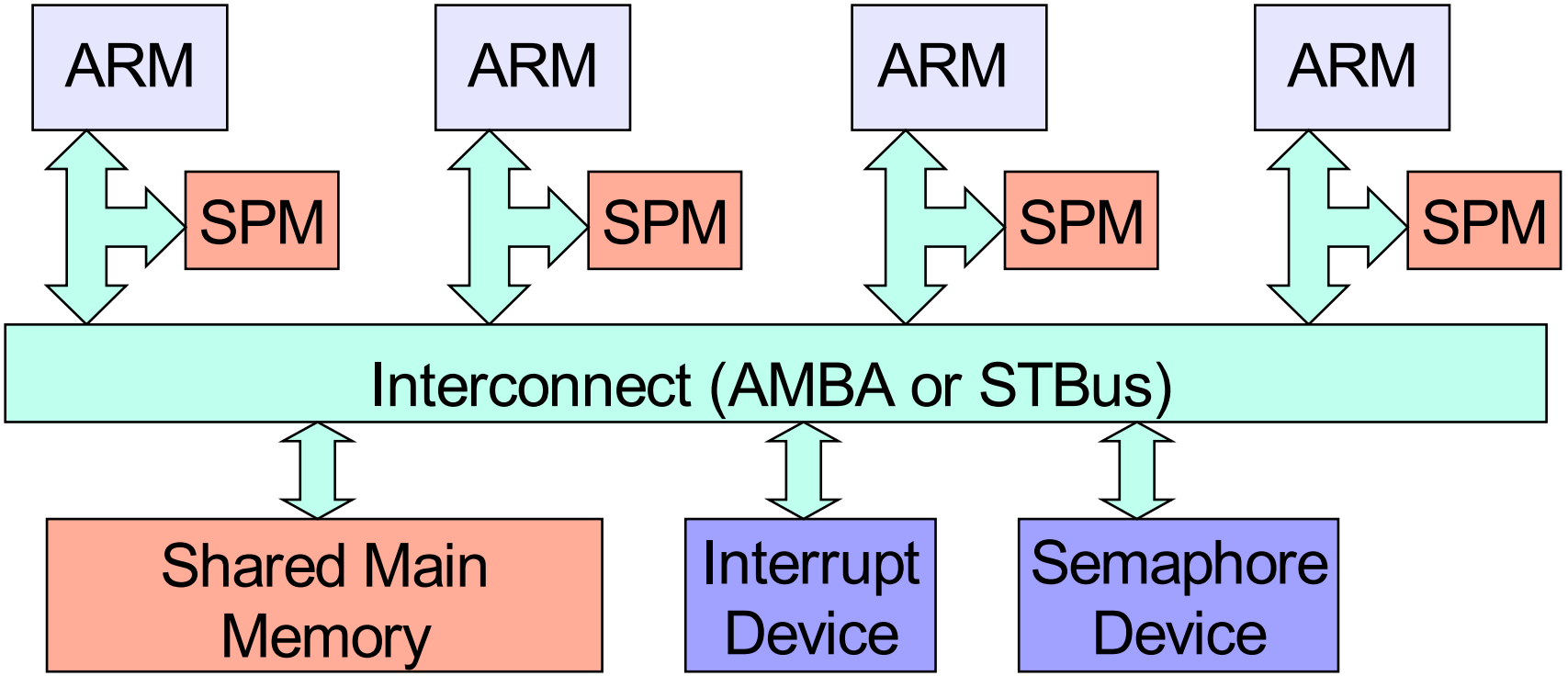
Results for SNACK-pop (1)



Results for SNACK-pop (2)

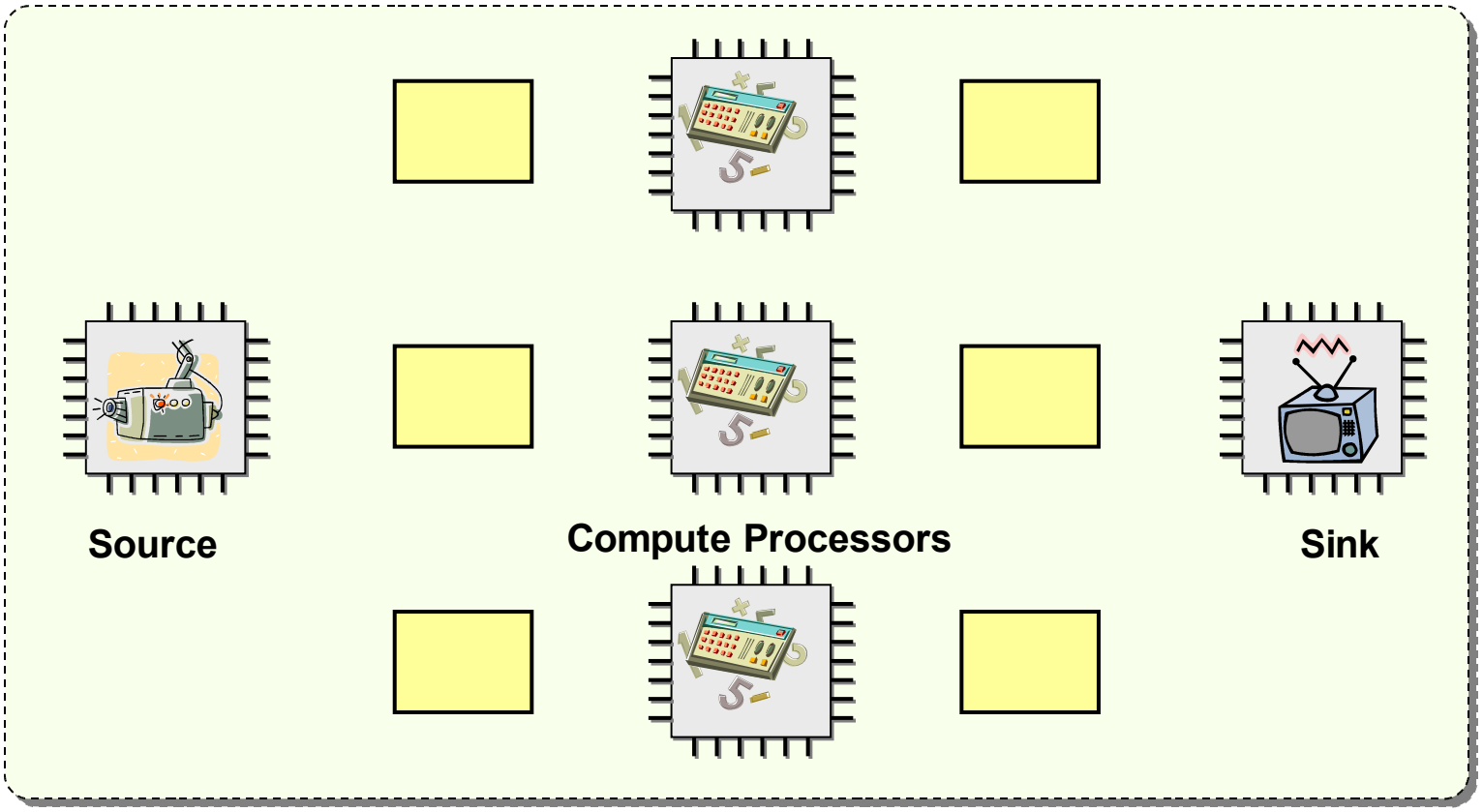


Multi-processor ARM (MPARM) Framework



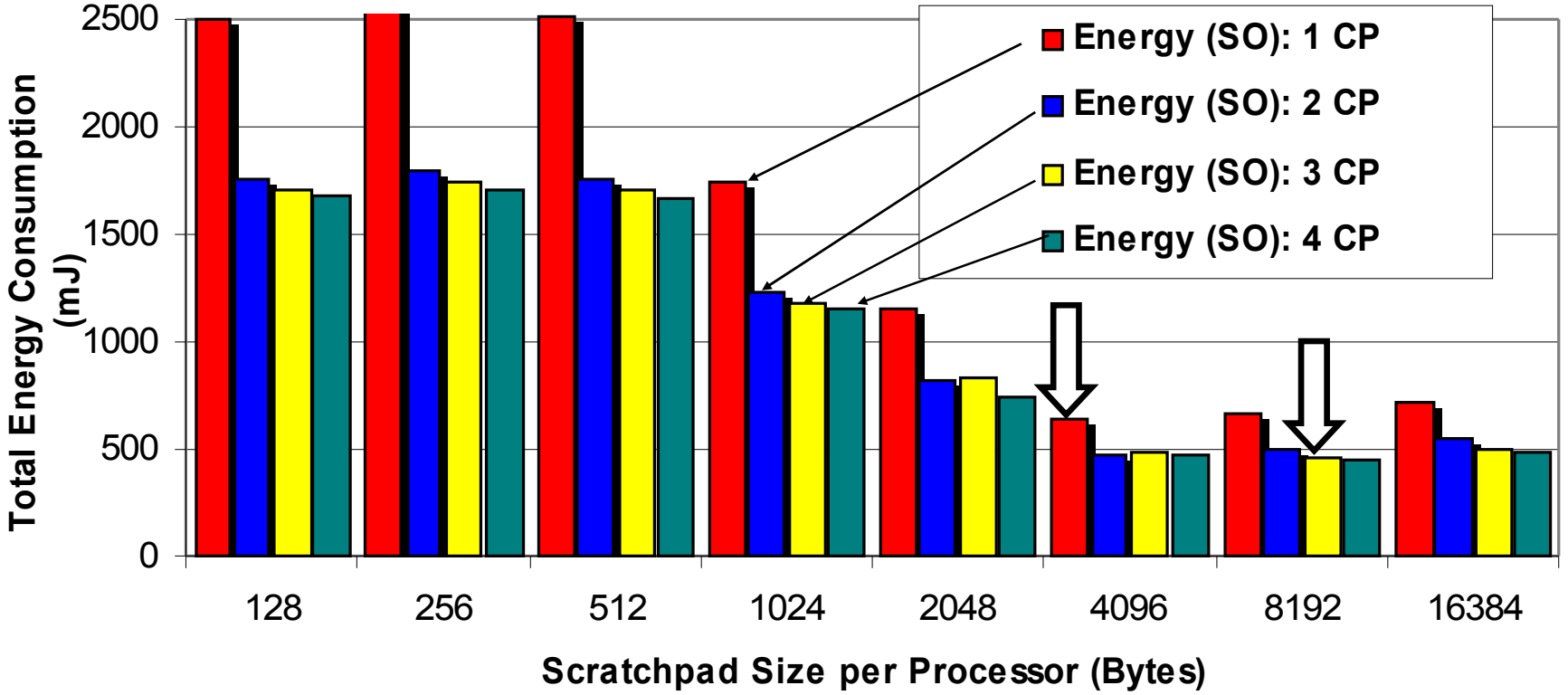
- Homogenous SMP ~ CELL processor
- Processing Unit : ARM7T processor
- Shared Coherent Main Memory
- Private Memory: Scratchpad Memory

Application Example:
Multi-Processor Edge Detection



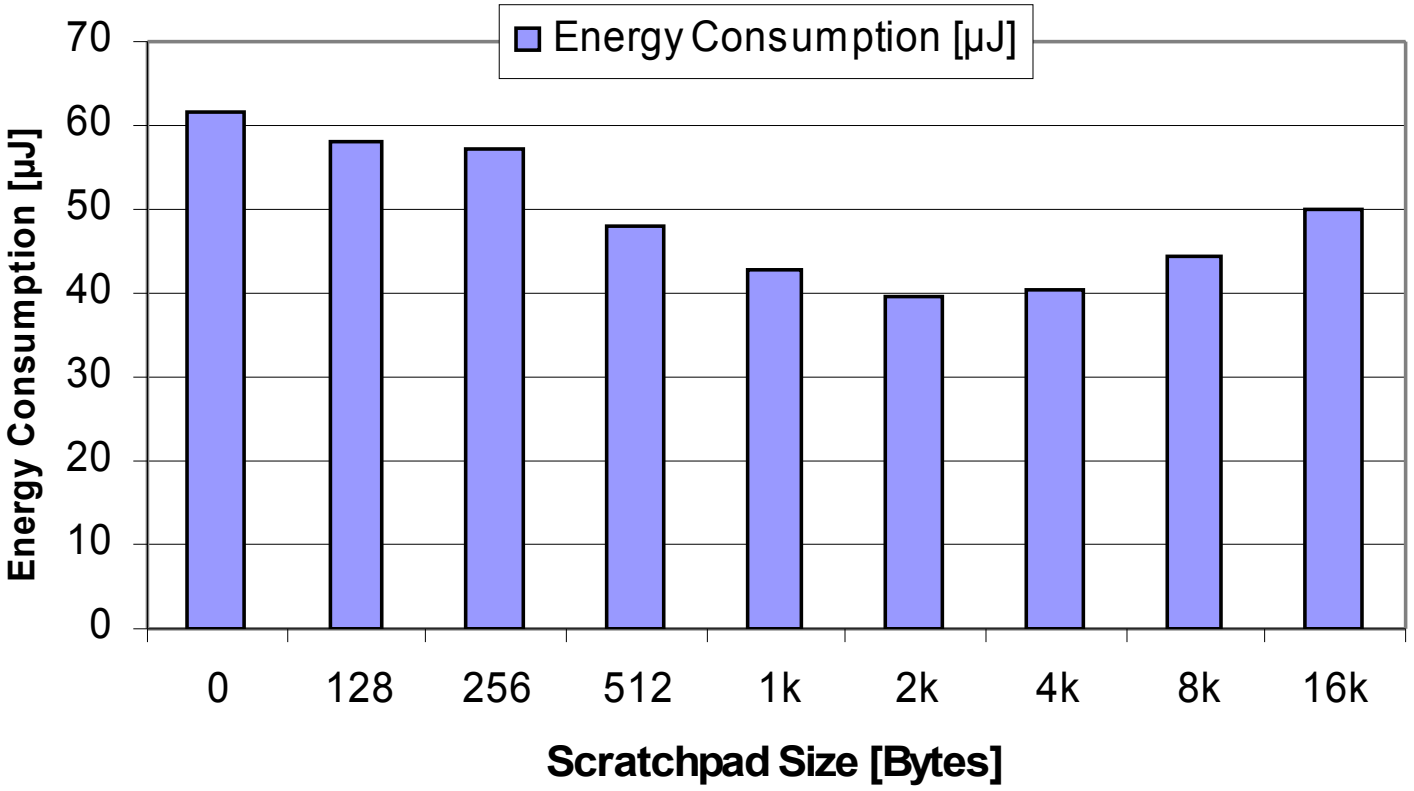
- Source, sink and *n* compute processors
- Each image is processed by an independent compute processor
  - Communication overhead is minimized.

Results:
Scratchpad Overlay for Edge Detection



- 2 CPs are better than 1 CP, then energy consumption stabilizes
- Best scratchpad size: 4kB (1CP& 2CP) 8kB (3CP & 4CP)

Results
DES-Encryption

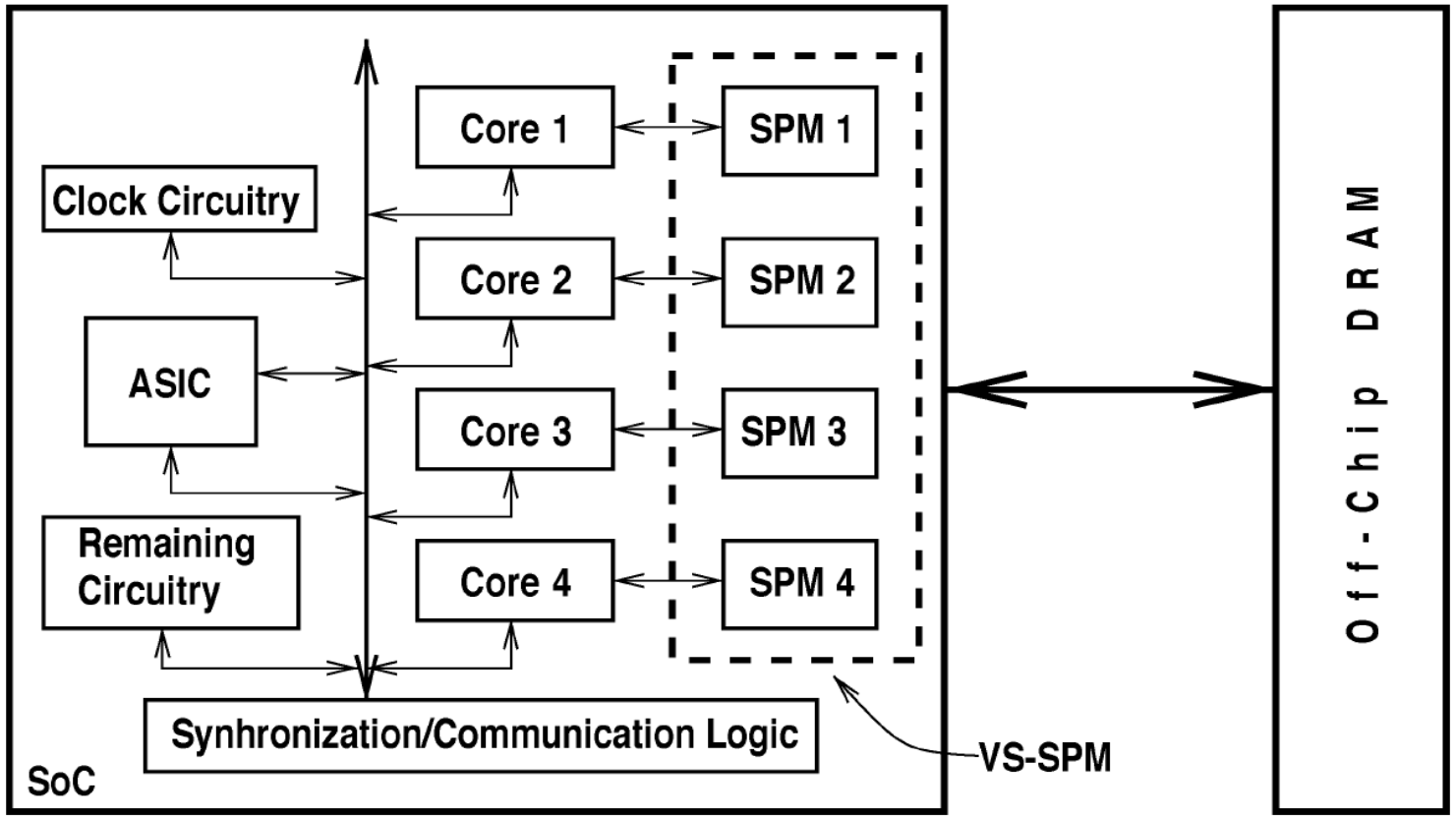


DES-Encryption: 4 processors: 2 Controllers+2 Compute Engines

Energy values from ST Microelectronics

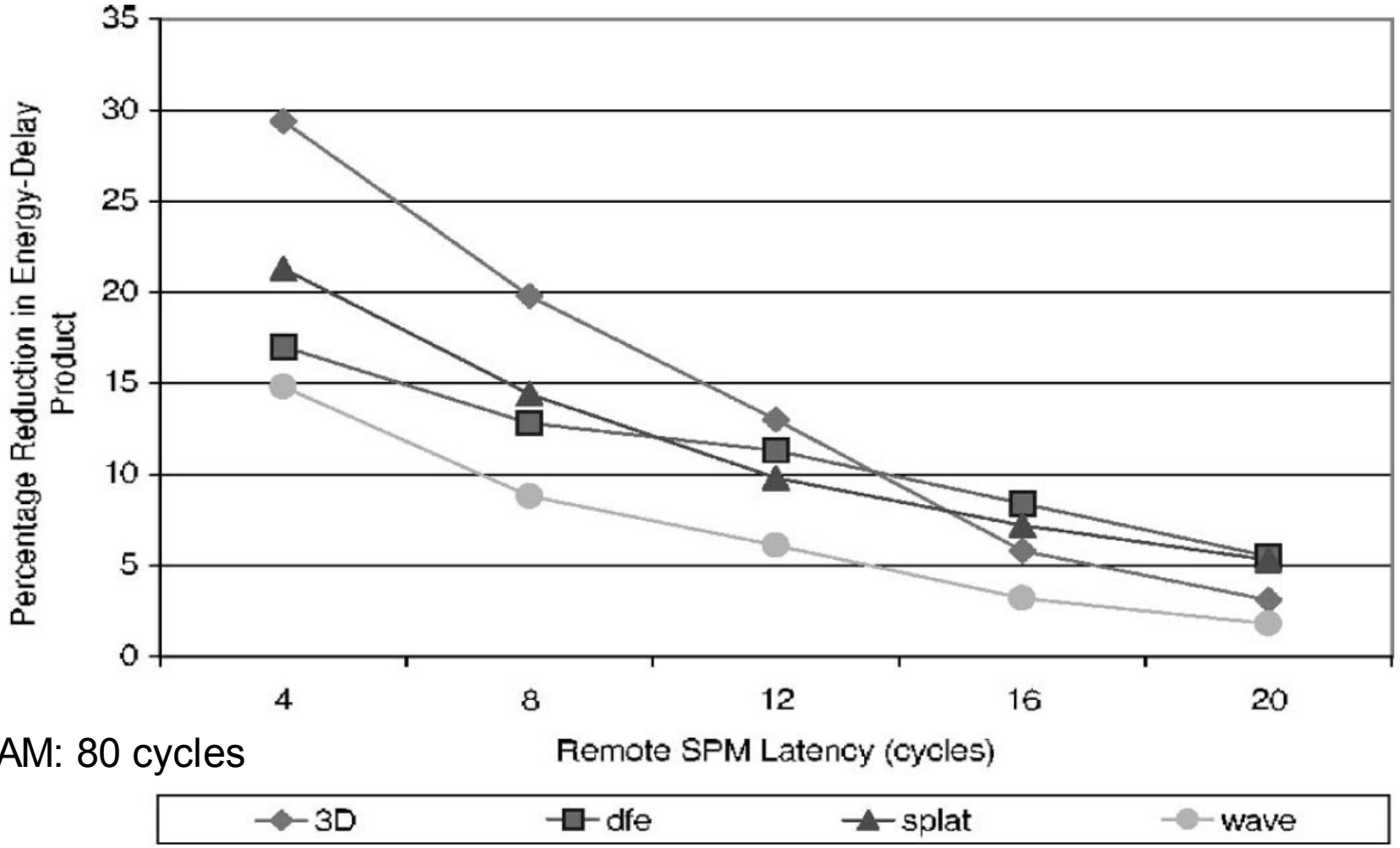
Result of ongoing cooperation between U. Bologna and U. Dortmund supported by ARTIST2 network of excellence.

MPSoC with shared SPMs



[M. Kandemir, I. Kadayif, A. Choudhary, J. Ramanujam, I. Kolcu: Compiler-Directed Scratch Pad Memory Optimization for Embedded Multiprocessors, *IEEE Trans. on VLSI*, Vol. 12, 2004, pp. 281-286]

Energy benefits despite large latencies for remote SPMs



DRAM: 80 cycles

Extensions

- Using DRAM
- Applications to Flash memory (copy code or execute in place): according to own experiments: very much parameter dependent

PhD thesis of Lars Wehmeyer

- Trying to imitate advantages of SPM with caches: partitioned caches, etc.

technische universität
dortmund

fakultät für informatik
informatik 12

Optimizations for Caches

Peter Marwedel
TU Dortmund
Informatik 12
Germany

2009/01/17

Improving predictability for caches

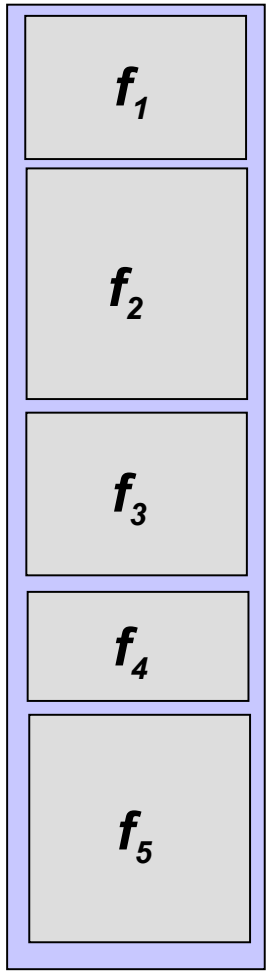
- Loop caches
- Mapping code to less used part(s) of the index space
- Cache locking/freezing
- Changing the memory allocation for code or data
- Mapping pieces of software to specific ways
  Methods:
  - Generating appropriate way in software
  - Allocation of certain parts of the address space to a specific way
  - Including way-identifiers in virtual to real-address translation

☞ "Caches behave almost like a scratch pad"

Code Layout Transformations (1)

Execution counts based approach:

- Sort the functions according to execution counts
  $f_4 > f_1 > f_2 > f_5 > f_3$
- Place functions in decreasing order of execution counts



[S. McFarling: Program optimization for instruction caches, 3*rd International Conference on Architecture Support for Programming Languages and Operating Systems (ASPLOS),* 1989]

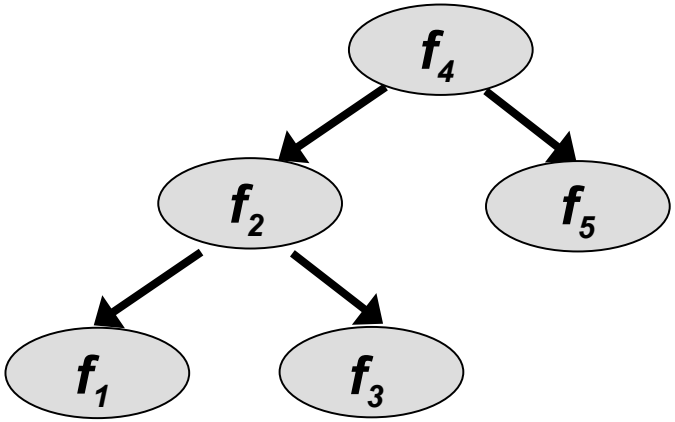
Code Layout Transformations (2)

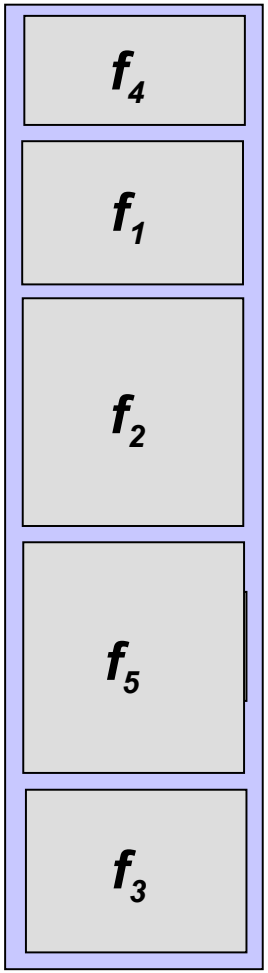
Execution counts based approach:

- Sort the functions according to execution counts
  $f_4 > f_1 > f_2 > f_5 > f_3$
- Place functions in decreasing order of execution counts

Transformation increases spatial locality.

Does not take in account calling order





Code Layout Transformations (3)

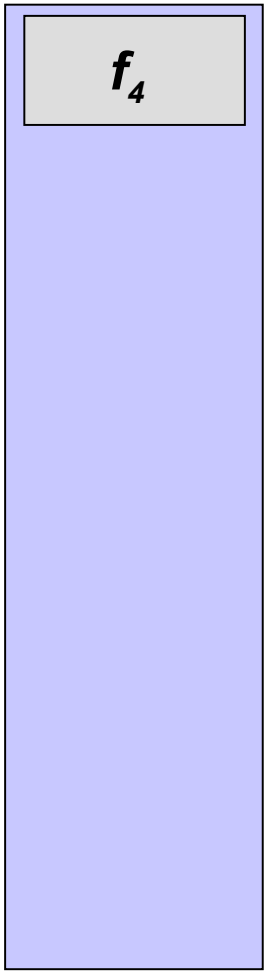
Call-Graph Based Algorithm:

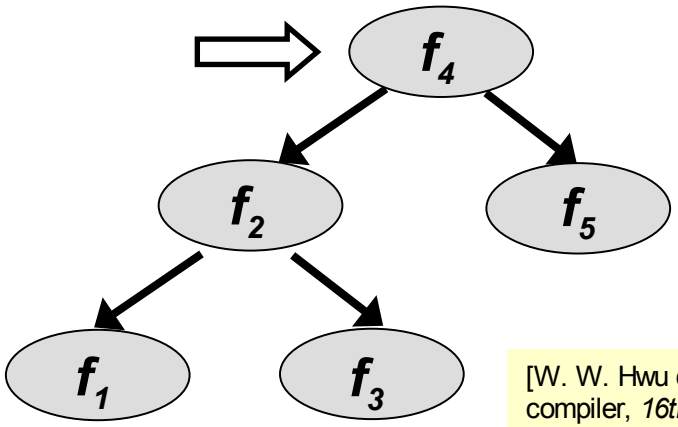
- Create weighted call-graph.
- Place functions according to weighted depth-first traversal.

  $f_4 > f_2 > f_1 > f_3 > f_5$

Increases spatial locality.

(2000)





[W. W. Hwu et al.: Achieving high instruction cache performance with an optimizing compiler, *16th Annual International Symposium on Computer Architecture,* 1989]

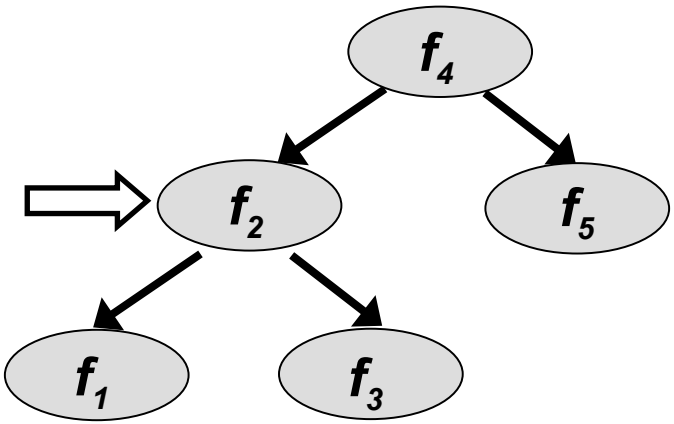
Code Layout Transformations (3)

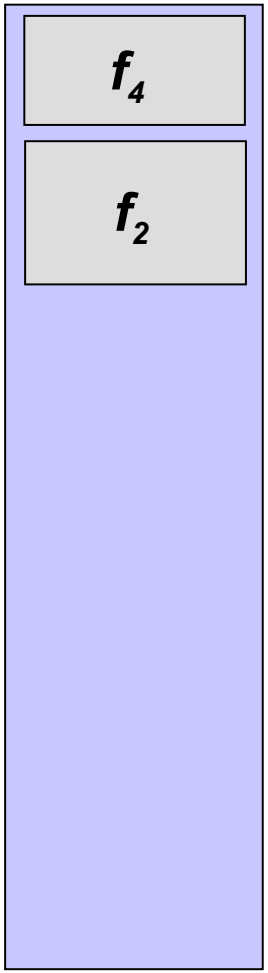
Call-Graph Based Algorithm:

- Create weighted call-graph.
- Place functions according to weighted depth-first traversal.

  $f_4 > f_2 > f_1 > f_3 > f_5$

Increases spatial locality.





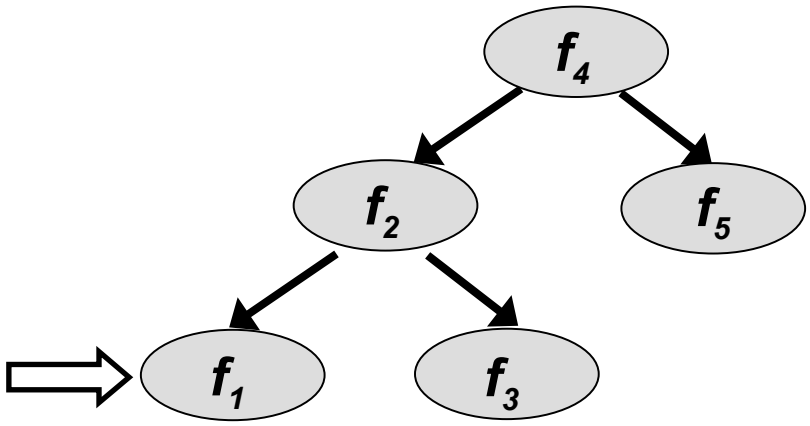
Code Layout Transformations (4)

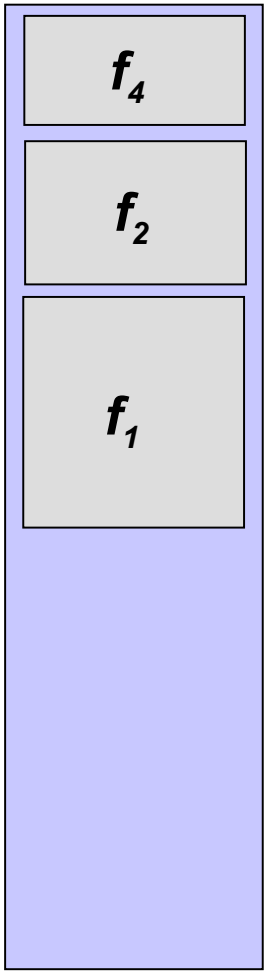
Call-Graph Based Algorithm:

- Create weighted call-graph.
- Place functions according to weighted depth-first traversal.

  $f_4 > f_2 > f_1 > f_3 > f_5$

Increases spatial locality.





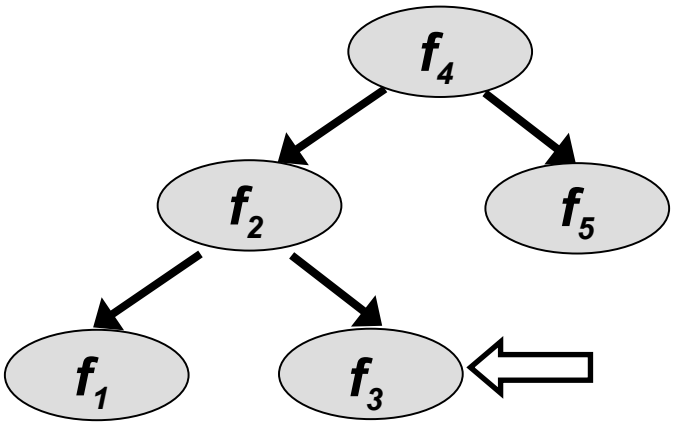
Code Layout Transformations (5)

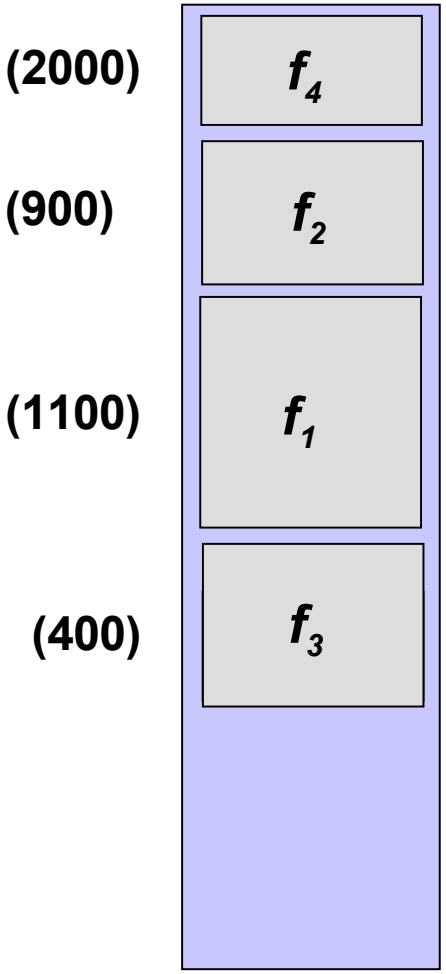
Call-Graph Based Algorithm:

- Create weighted call-graph.
- Place functions according to weighted depth-first traversal.

  $f_4 > f_2 > f_1 > f_3 > f_5$

Increases spatial locality.





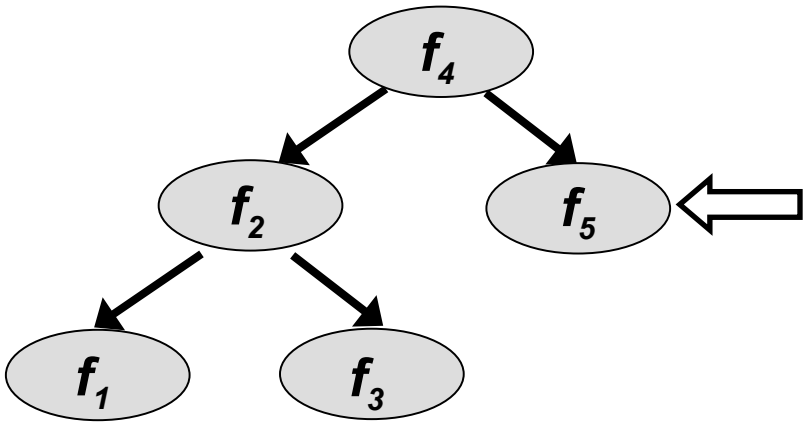
Code Layout Transformations (6)

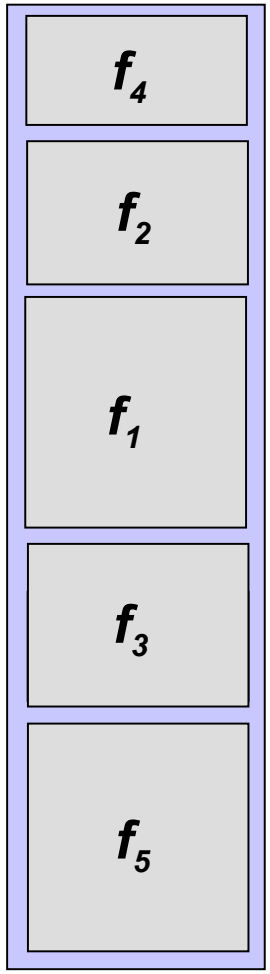
Call-Graph Based Algorithm:

- Create weighted call-graph.
- Place functions according to weighted depth-first traversal.

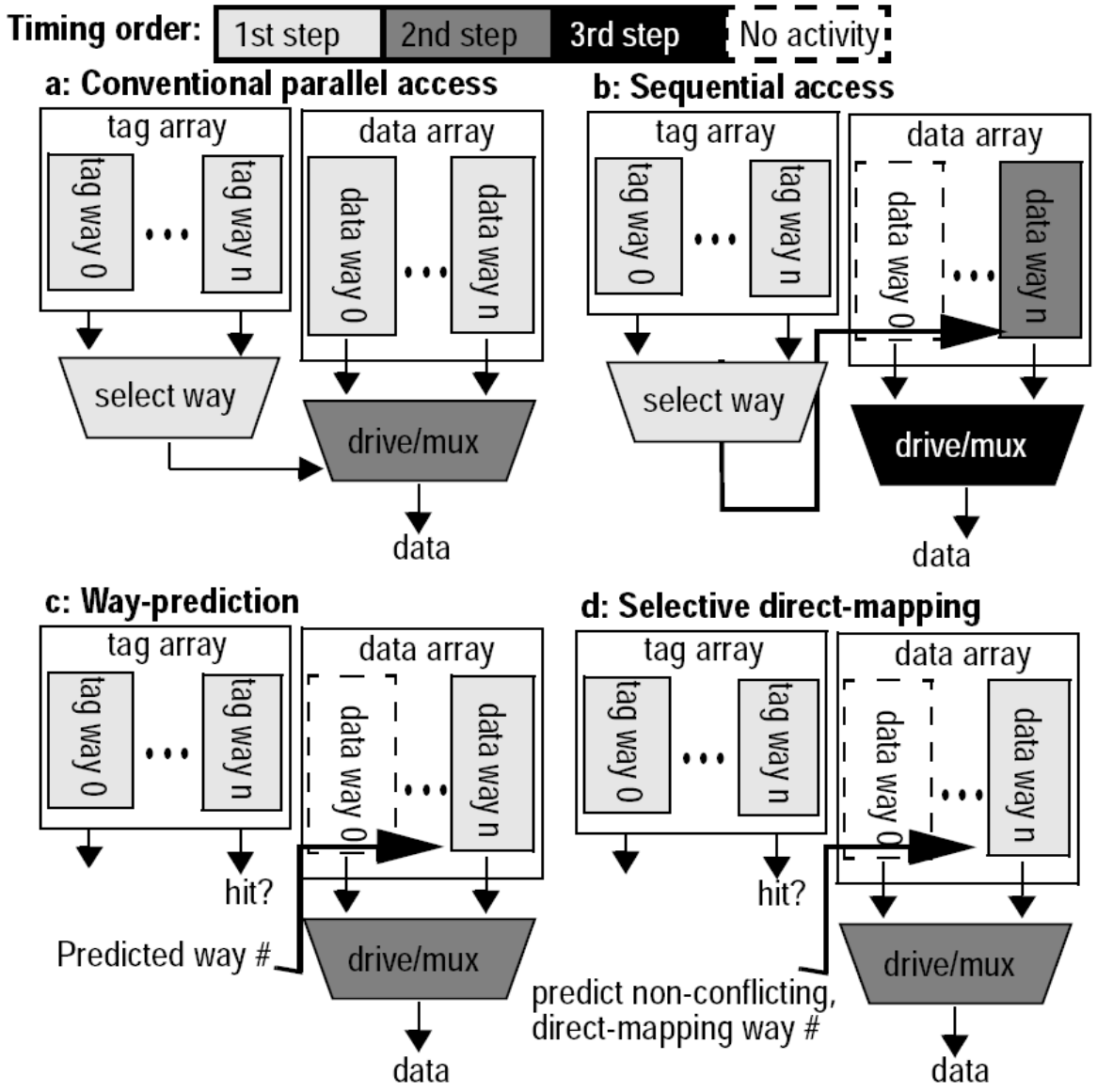
  $f_4 > f_2 > f_1 > f_3 > f_5$
- Combined with placing frequently executed traces at the top of the code space for functions.

Increases spatial locality.



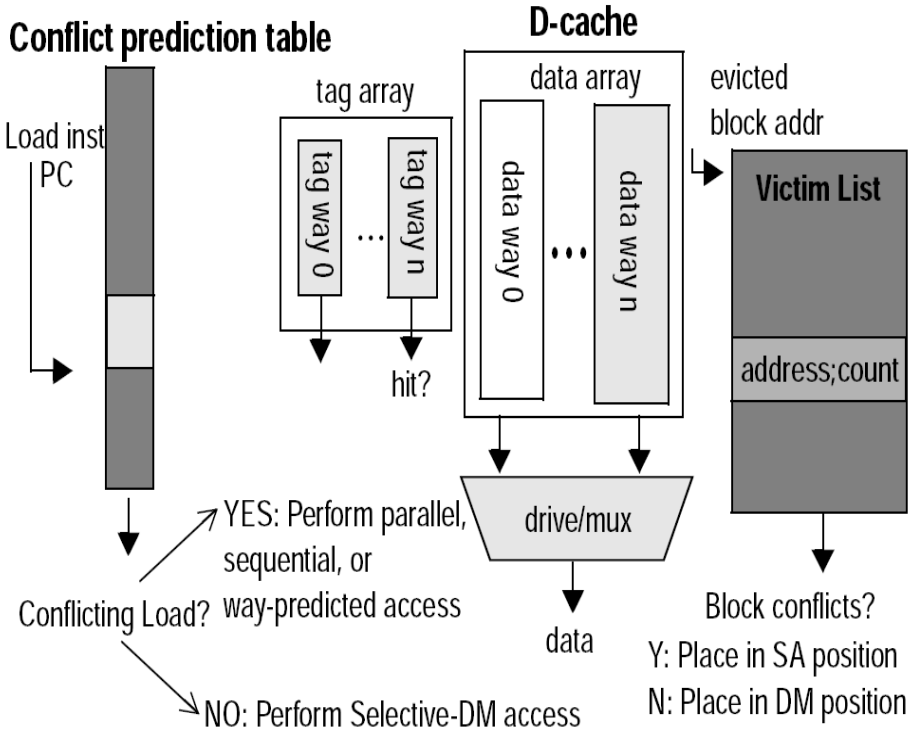


Way prediction/selective direct mapping



[M. D. Powell, A. Agarwal, T. N. Vijaykumar, B. Falsafi, K. Roy: Reducing Set-Associative Cache Energy via Way-Prediction and Selective Direct-Mapping, *MICRO-34*, 2001]

Hardware organization for way prediction



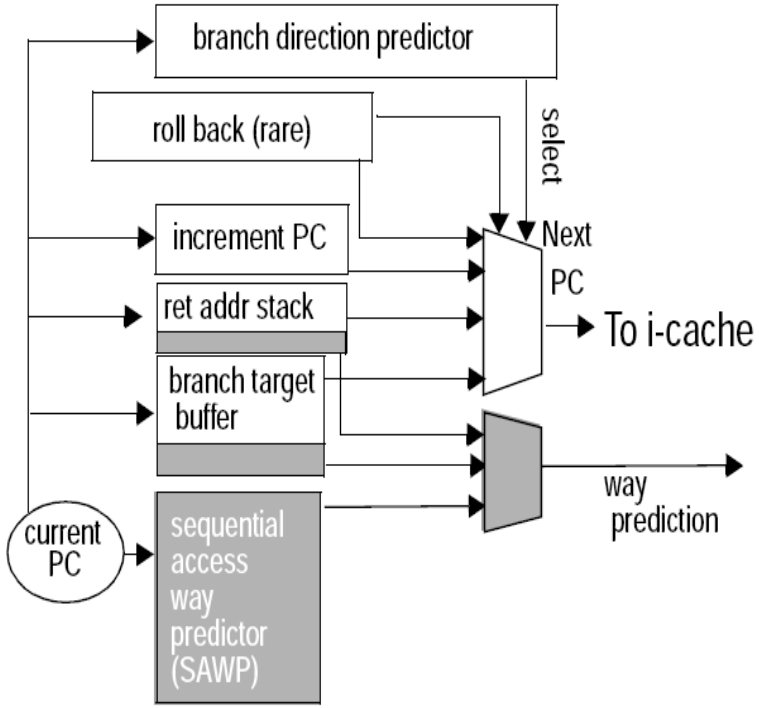


FIGURE 3: Fetch and i-cache access mechanism.

Results for the paper on way prediction (1)

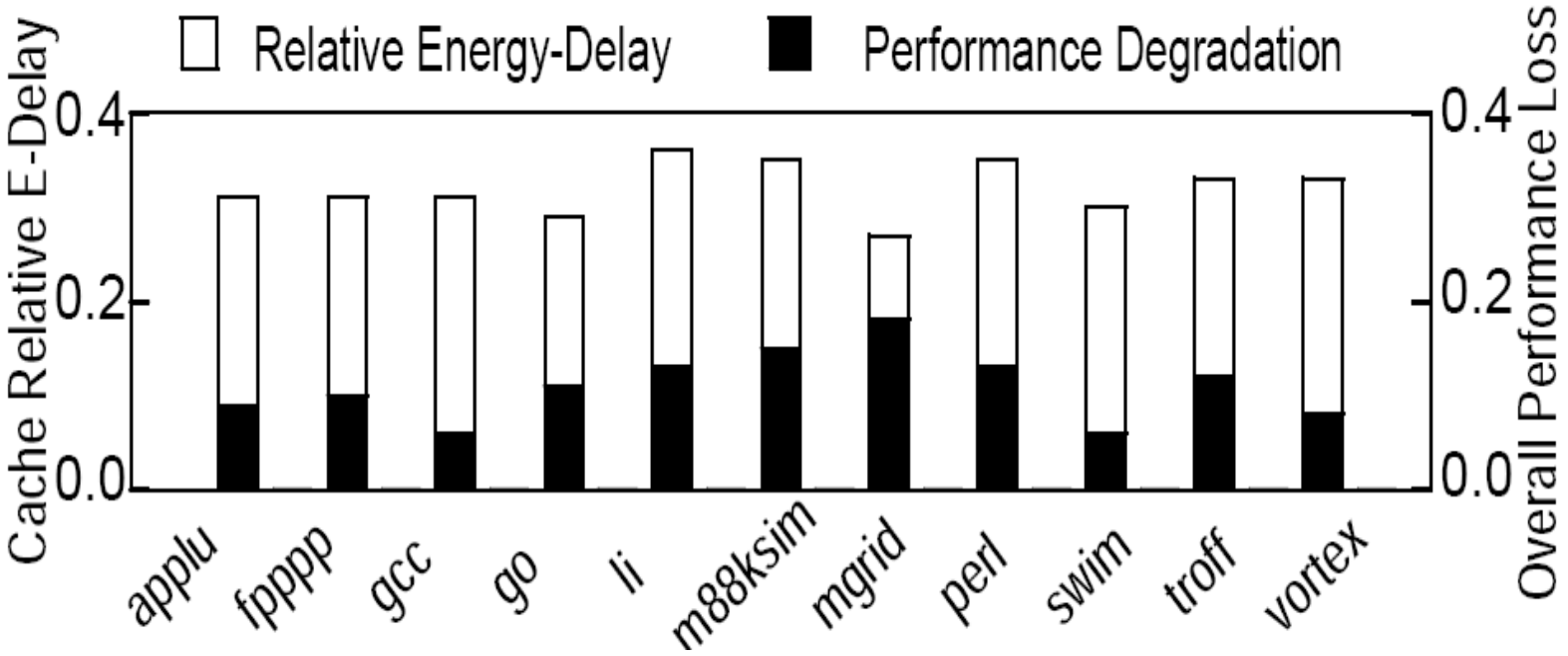
System configuration parameters

Instruction issue & decode bandwidth	8 issues per cycle
L1 I-Cache	16K, 4-way, 1 cycle
Base L1 D-Cache	16K, 4-way, 1 or 2 cycles, 2ports
L2 cache	1M, 8-way, 12 cycle latency
Memory access latency	80 cycles+4 cycles per 8 bytes
Reorder buffer size	64
LSQ size	32
Branch predictor	2-level hybrid

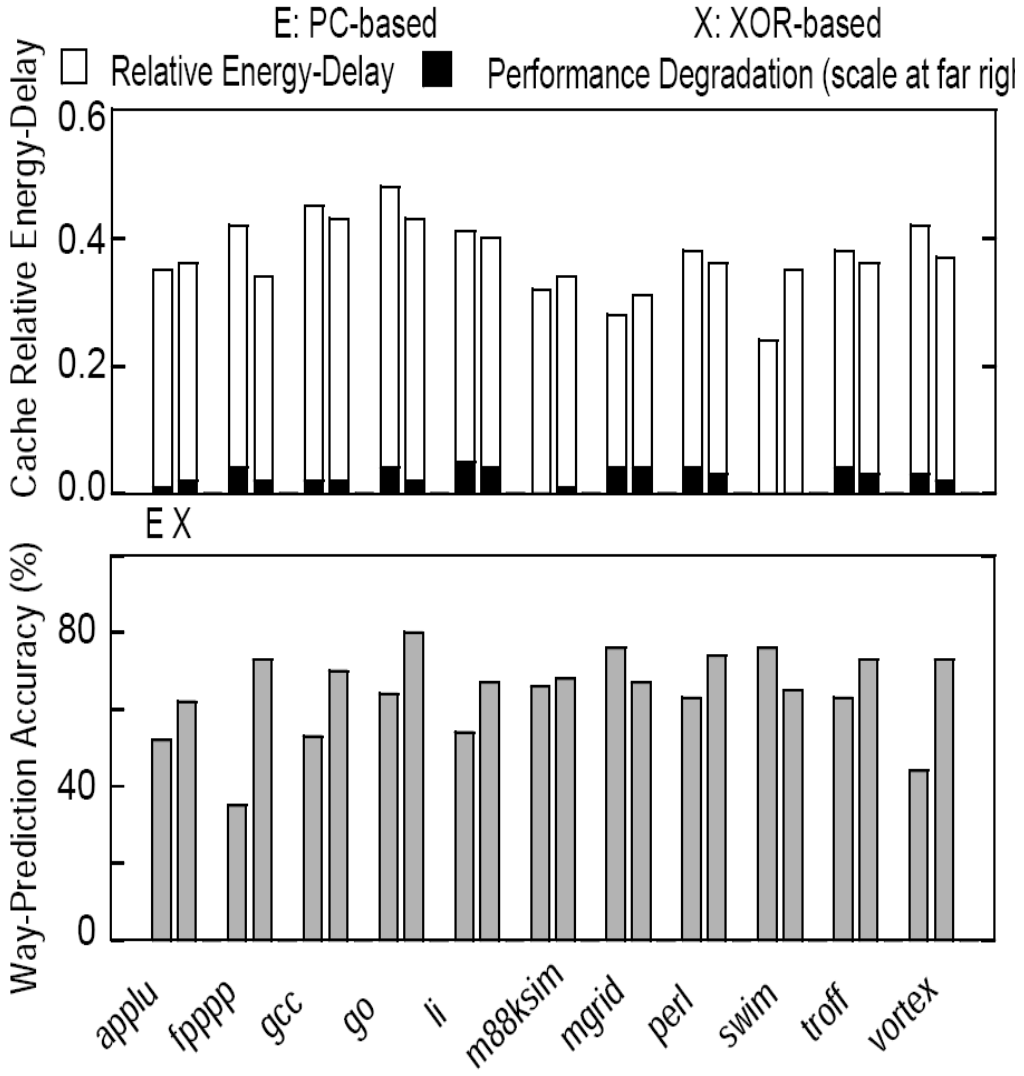
Cache energy and prediction overhead

Energy component	**Relative energy**
Parallel access cache read (4 ways read)	1.00
1 way read	**0.21**
Cache write	0.24
Tag array energy (incl. in the above numbers)	0.06
1024x4bit prediction table read/write	0.007

Results for the paper on way prediction (2)



Results for the paper on way prediction (2)



Summary

- Allocation strategies for SPM
  - Dynamic sets of processes
  - Multiprocessors
  - MMUs
  - Sharing between SPMs in a multi-processor
- Optimizations for Caches
  - Code Layout transformations
  - Way prediction