# Compiler-Directed Memory Hierarchy Design for Low-Energy Embedded Systems

Florin Balasa
**American University in Cairo**

Ilie I. Luican
**Microsoft Inc., USA**

Noha Abuaesh
**American University in Cairo**

Cristian V. Gingu
**Fermilab**

## Outline

- Memory management for embedded signal processing applications
- The problem: Memory hierarchy design for low-energy
- The model: Polyhedral framework for memory management
- The algorithms: Energy-aware assignment to memory layers

  Energy-aware memory banking
- Experimental results and conclusions

# Low-Power Memory Management

Real-time multidimensional signal processing systems

(video and image processing, telecommunications,
audio and speech coding, medical imaging, etc.)

The **performance+energy+area** of the whole system
are significantly influenced by ***the memory subsystem***

Gap between processor and memory speed ➡ memory unable
to provide data and instructions at the pace required by the processor

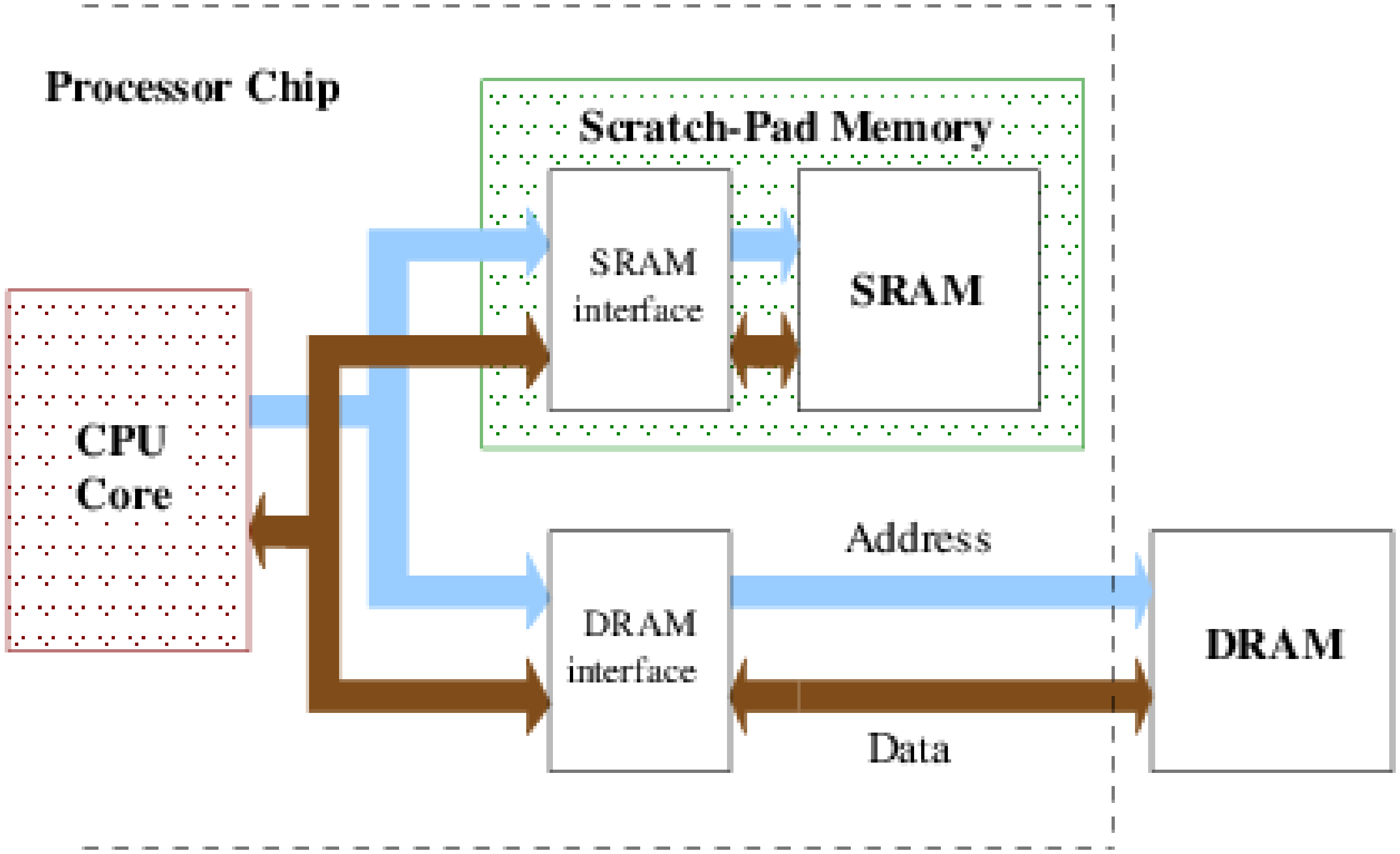
***"Memory Wall"*** [ Wulf & McKee 1995 ]
or ***"Memory Bottleneck"***

## Low-Power Memory Management

**Advanced memory architectures**
based on the concept of memory hierarchy

- lower levels in the hierarchy are made of small memories, close and tightly coupled to the computation units
- higher levels are made of increasingly large memories, far from the computation units
- the terms close and far refer to the effort needed to fetch/store a given amount of data from/to the memory
- the effort can be expressed in units of time or energy depending on the cost function of interest

# Simple Hierarchical Data Storage Organization



# Low-Power Memory Management

**SPMs vs. Caches**

**[ Banakar, Marwedel & *al.* 2002 ]**

**in embedded systems**

- 34% smaller area than caches of same capacity
- 40% lower power consumption than caches of same capacity
- 18% less clock cycles for a knapsack allocation algorithm
  (the access time of an SPM is typically within 1 clock cycle)

## Low-Power Memory Management

Scratchpad memories (SPMs) are on-chip SRAMs

The mapping of data to the SPM can be done statically

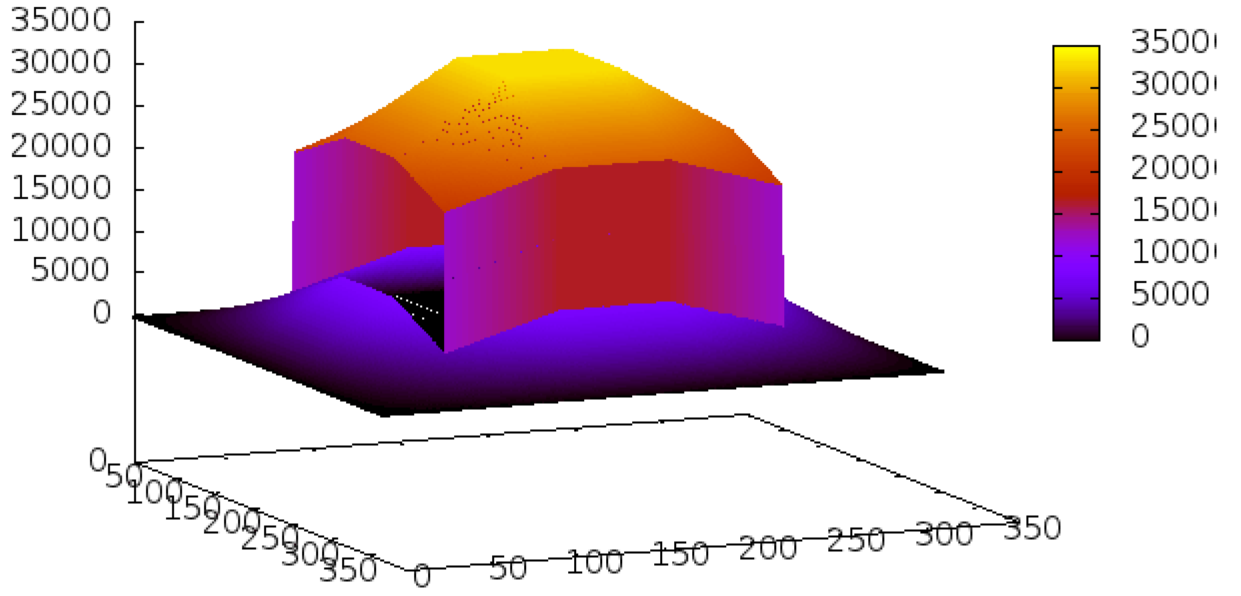
No need to check at run time the availability of data in the SPM

No need of comparators, of the miss/hit acknowledging logic

# Example of Behavioral Specification

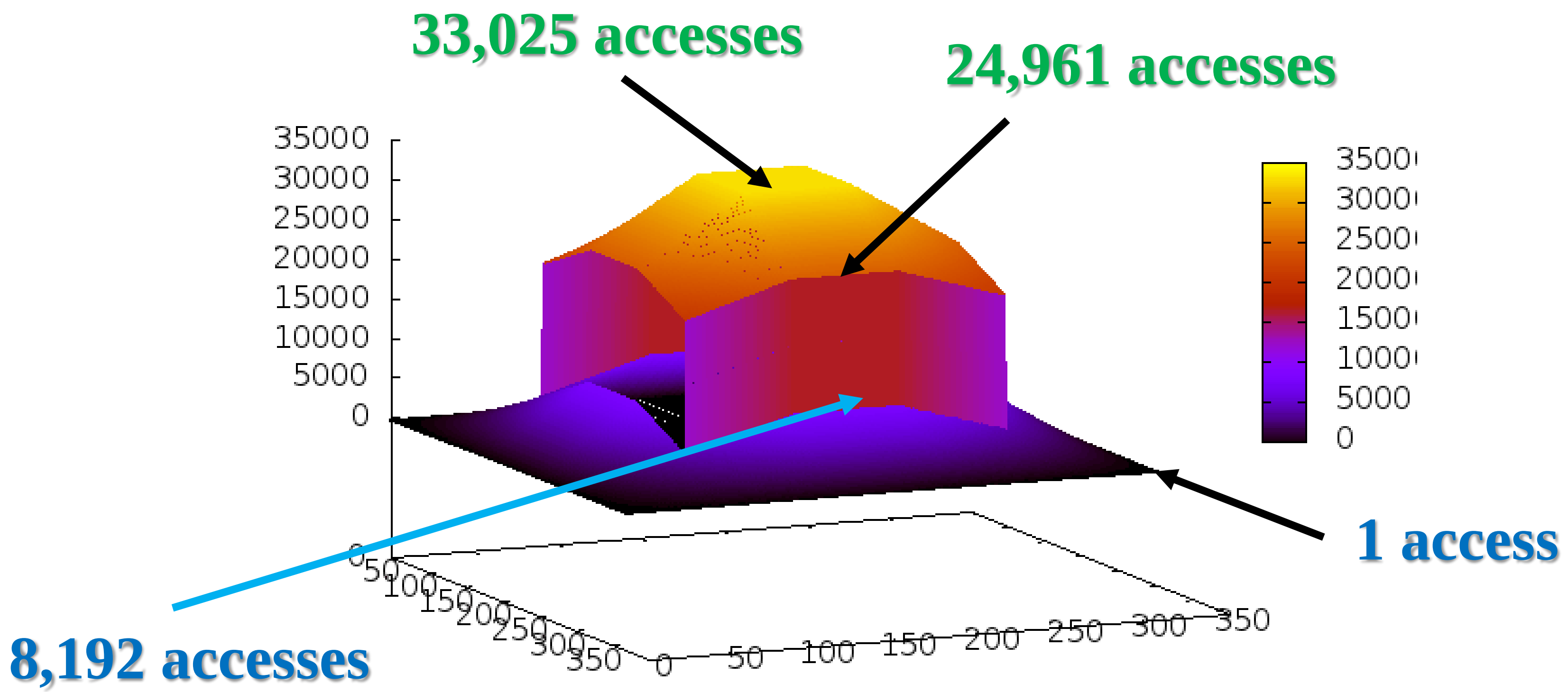
```
C[0] = 0;                                      //  input:  int A[256][256]
for (int i=64; i<192; ++i) {
   for (int j=64; j<192; ++j) {
      B[i][j][0] = 0;
      for (int k=i–64; k<=i+64; ++k) {
         for (int l=j–64; k<=j+64; ++l) {
            B[i][j][129*k – 129*i + l – j + 8321] = A[i][j]
                  – A[k][l] + B[i][j][129*k – 129*i + l – j + 8320] ;
         }
      }
      C[128*i + j – 8255] = B[i][j][16641] + C[128*i + j – 8256] ;
   }
}
out = C[16384] ;                               //  output:  out
```

## Signal Assignment to Memory Layers



Map of memory accesses to signal’s A index space

# Signal Assignment to Memory Layers



Map of memory accesses to signal’s A index space

## Signal Assignment to Memory Layers

The problem

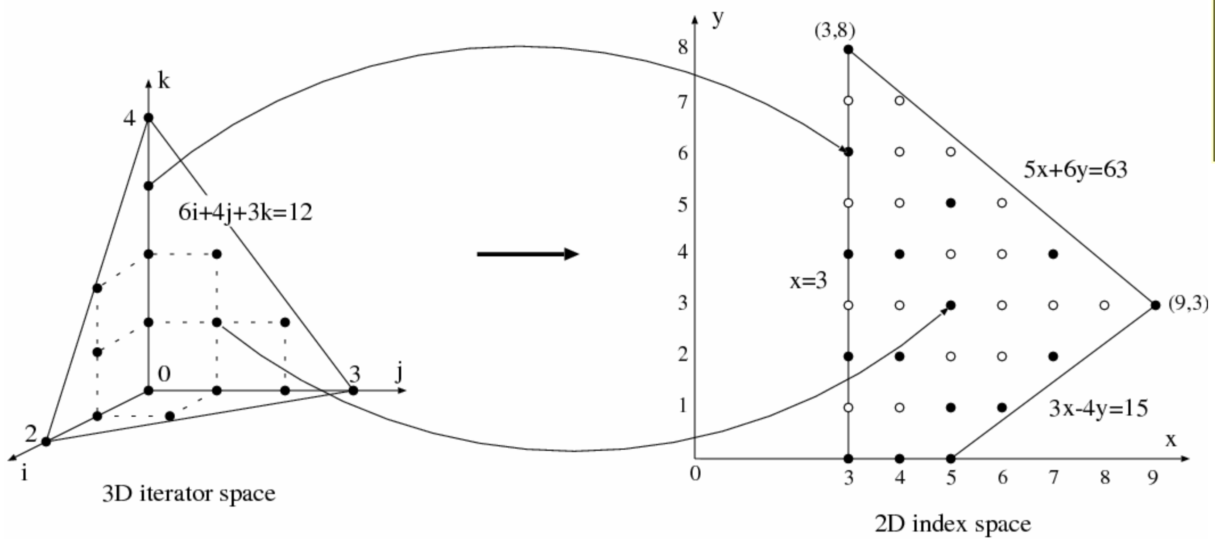
- the dynamic energy consumption per memory access is much smaller for the SPM than for external DRAM
- the arrays may have very non-uniform access patterns (the elements A[i][j] have 1 – 33,025 accesses)
- an array may need a lot of storage: it may not be possible to store it *all* within the SPM (array A needs 64 Kbytes)

Which array elements should be stored on-chip such that the (static+dynamic) energy consumption in the multi-layer memory subsystem be minimized ?

# Signal Assignment: the Model

3D iterator space

2D index space



```
for (i=0; i<=2; i++)
   for (j=0; j<= 3; j++)
      for (k=0; k<= 4; k++)
         if (6*i+4*j+3*k <= 12)
            … A [ i+2*j+3] [ j+2*k]  …
```

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 2 & 0 \\ 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} i \\ j \\ k \end{bmatrix} + \begin{bmatrix} 3 \\ 0 \end{bmatrix}$$

A[x(i, j, k)][y(i, j, k)]

## Signal Assignment: the Model

The array references represented as (linearly-bounded) lattices

**LBL**

$$\boxed{\left\{ x = T \cdot i + u \;\middle|\; A \cdot i \geq b \right\}}$$

**Intuitive observation**

The most intensely-accessed parts of an array are typically covered by several array references

Intersect the array references !

## Signal Assignment: the Model

$\rightarrow$ **Intersect lattices !**

$$\mathrm{LBL}_1 \cap \mathrm{LBL}_2 \rightarrow \mathrm{LBL}$$

$$\begin{cases} \mathrm{LBL}_1 = \{\, x = T_1 \cdot i_1 + u_1 \mid A_1 \cdot i_1 >= b_1 \,\} \\ \mathrm{LBL}_2 = \{\, x = T_2 \cdot i_2 + u_2 \mid A_2 \cdot i_2 >= b_2 \,\} \end{cases}$$

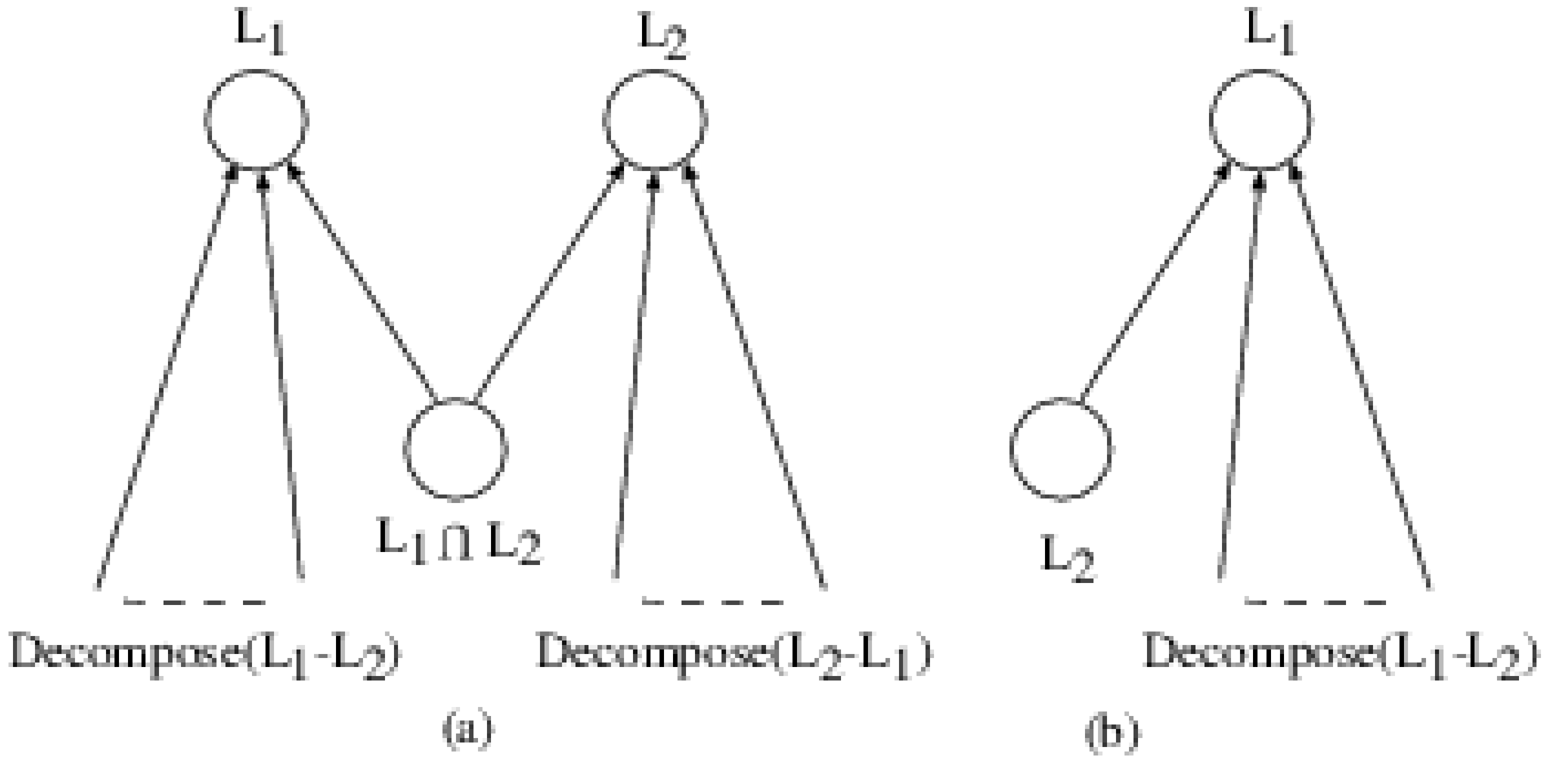
$T_1 \cdot i_1 + u_1 = T_2 \cdot i_2 + u_2$

**Diophantine system of eqs.**

$\{\, A_1 \cdot i_1 >= b_1 ,\ A_2 \cdot i_2 >= b_2 \,\}$

**New polytope**

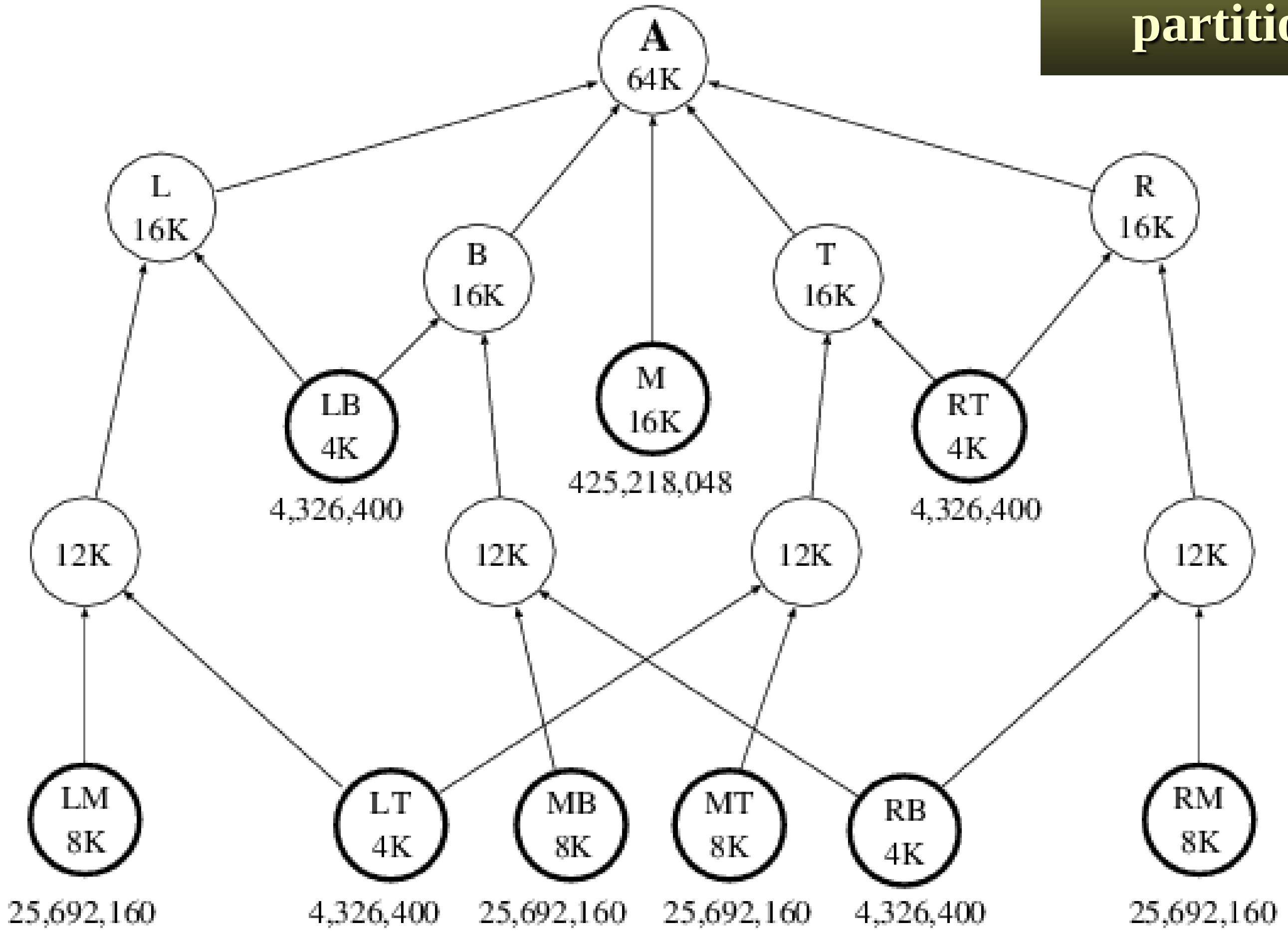
## Signal Assignment: the Model



Part of a DAG representing two lattices whose intersection is not empty
(the difference may comprise several lattices)

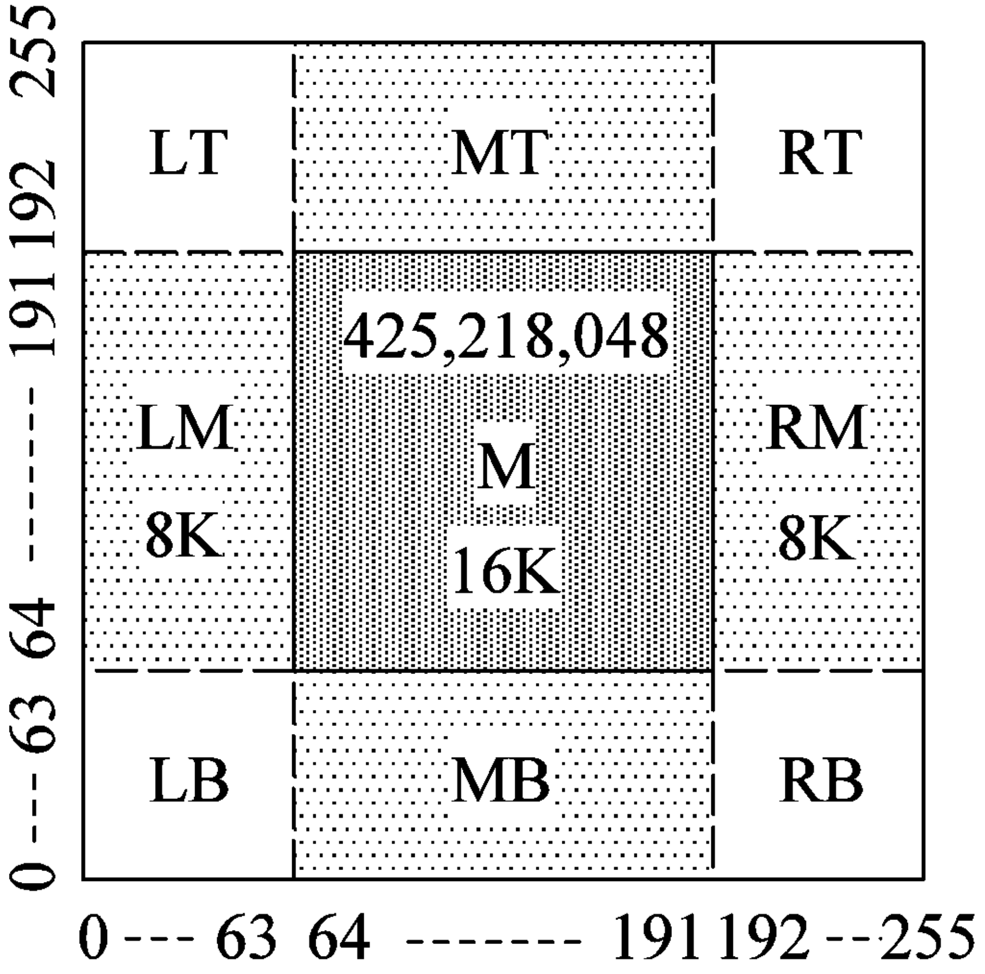
## DAG of LBLs

- The nodes without predecessors represent the partitions of the array space



## Partitioning the Array Space

The central part M of the array space of signal A receives 78% of the memory accesses



## Signal Assignment to Memory Layers

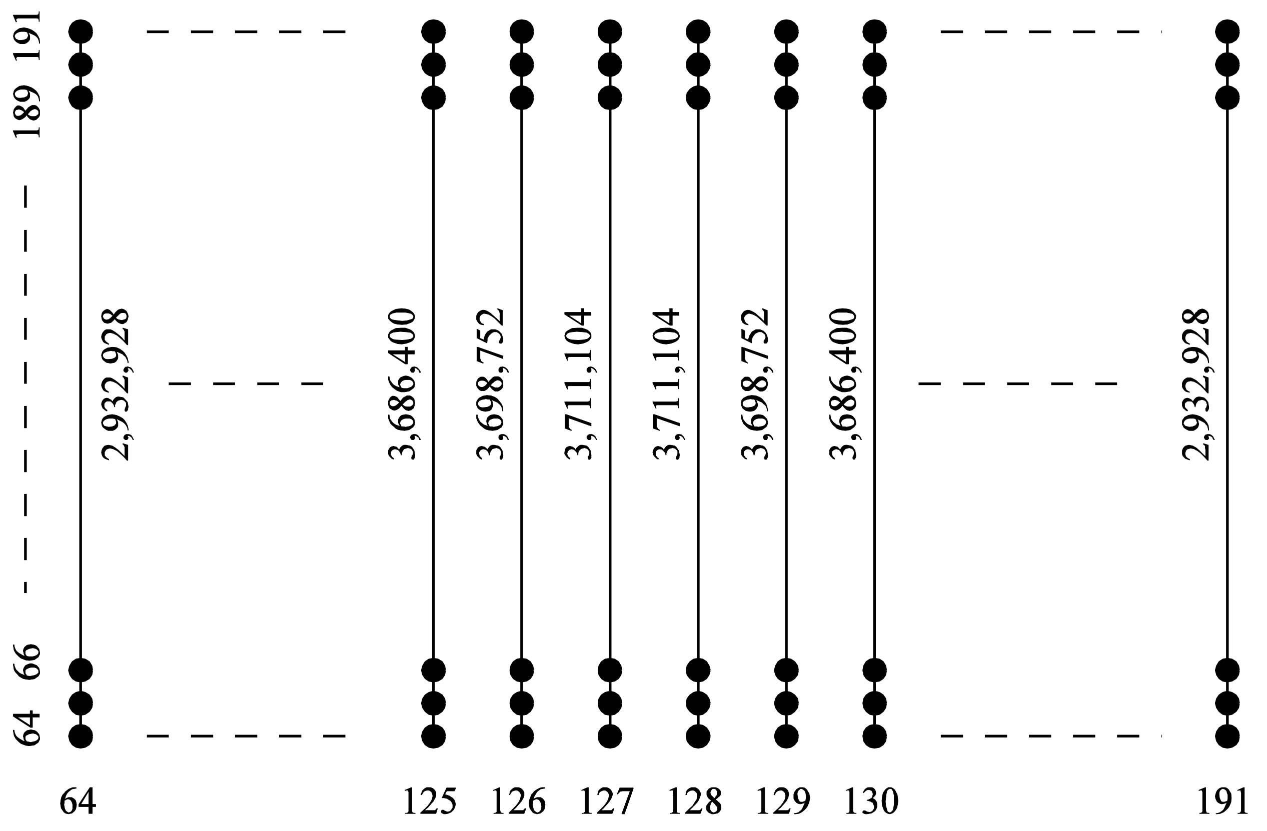
### The Assignment Algorithm

- focus on the LBLs whose assignment to the SPM layer would yield the largest energy reduction
- an LBL may need a lot of storage: it may not be possible to store it *all* within the SPM (lattice M needs 16 Kbytes)

What is to be done if the size of the SPM is too small relative to the storage requirement of a lattice ?

## Slicing the Large Lattices

The lattice M of 16Kbytes is split



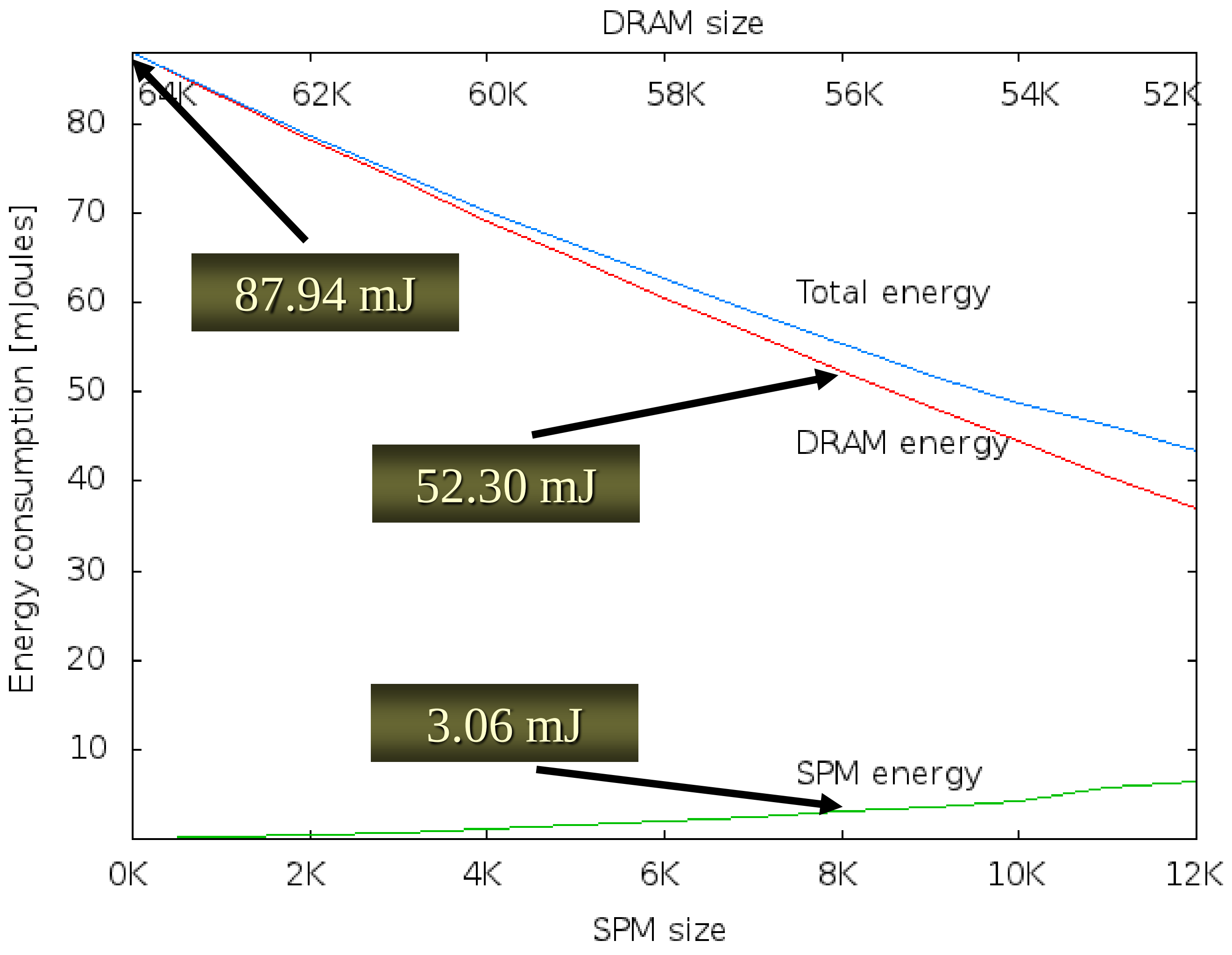
**Fine-grain lattices of 128 bytes**

## Signal Assignment to Memory Layers

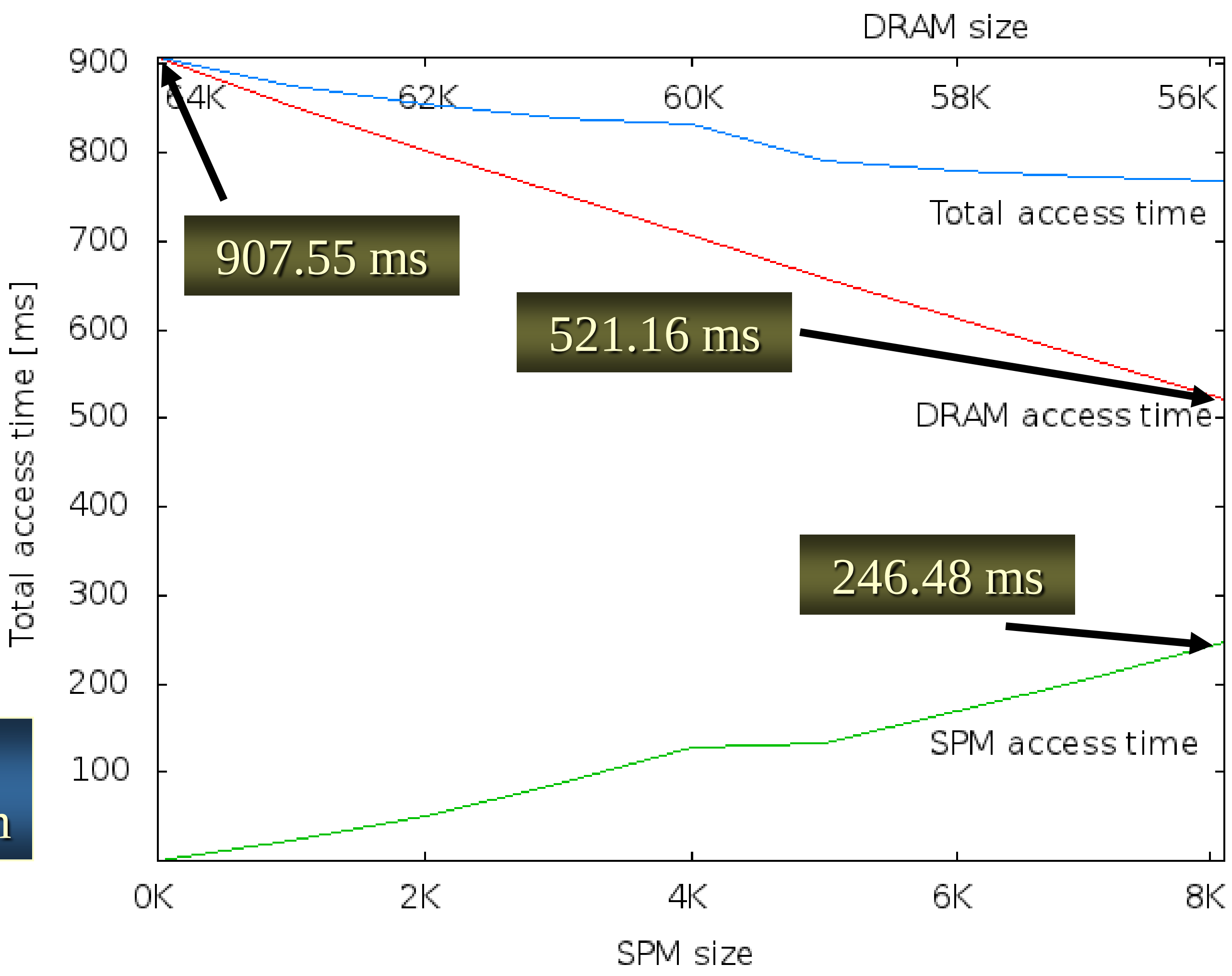
### The Assignment Algorithm

```
for ( each signal in the application code )
    partition its array space into disjoint LBLs ;
initially, assign all the disjoint lattices L to the DRAM ;
    size(DRAM) = Σ size(L) ;      size(SPM) = 0 ;
for ( each disjoint lattice L )  compute the energy reduction if  L is assigned to SPM ;
do {  select the lattice L yielding the largest reduction of energy ;
        if  ( size(SPM) + size(L) <= MAX_SPM_SIZE )
            assign L to the SPM:  size(SPM) += size(L);   size(DRAM) -= size(L) ;
            update the energy benefits of all the lattices still assigned to the DRAM ;
        else
            "slice" the lattice L relative to its next iterator and replace L by these LBLs ;
            compute the energy benefits of these new 'finer-granularity' lattices ;
}  until ( MAX_SPM_SIZE is reached  ||  the max level of "slicing" is reached ) ;
```

CACTI 6.5
Technology 32 nm
Frequency 400 MHz



The graphs of energy consumption by the SPM, DRAM, and the 2-layer memory system storing the array A of 64 Kbytes



CACTI 6.5
Technology 32 nm

The graphs of total access time to the SPM, to the DRAM, and to the 2-layer memory system storing the array A of 64 Kbytes

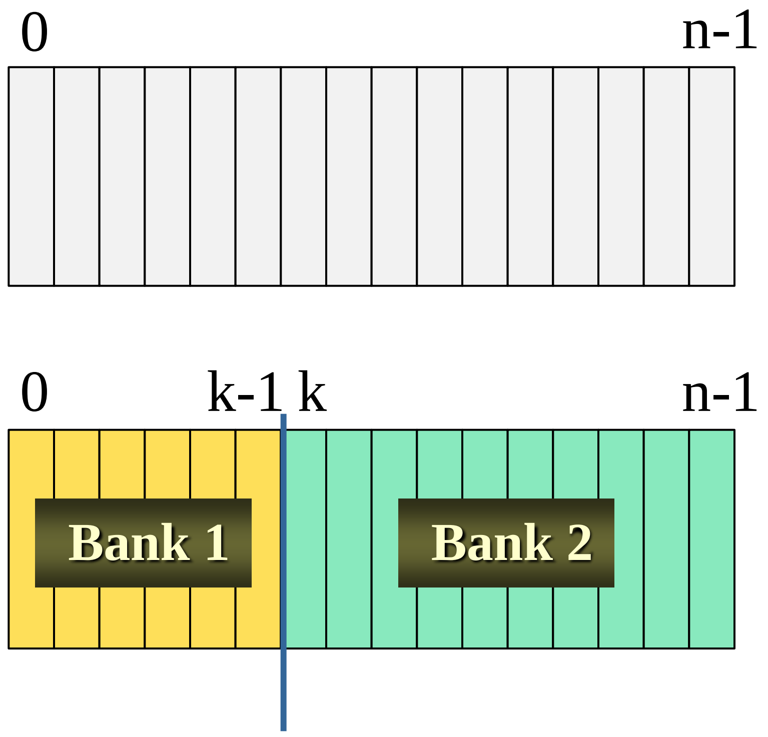
# Energy-aware SPM Banking

## Motivation

- Reduction of static & dynamic energy consumption with the size of the memory banks
- Possibility of turning on/off the banks independently

- Hardware cost overhead
- Energy overhead
  (but also time and area overhead)

# Energy-aware SPM Banking

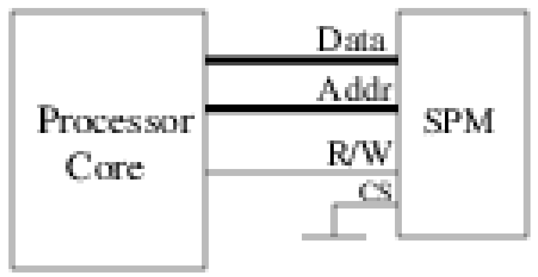


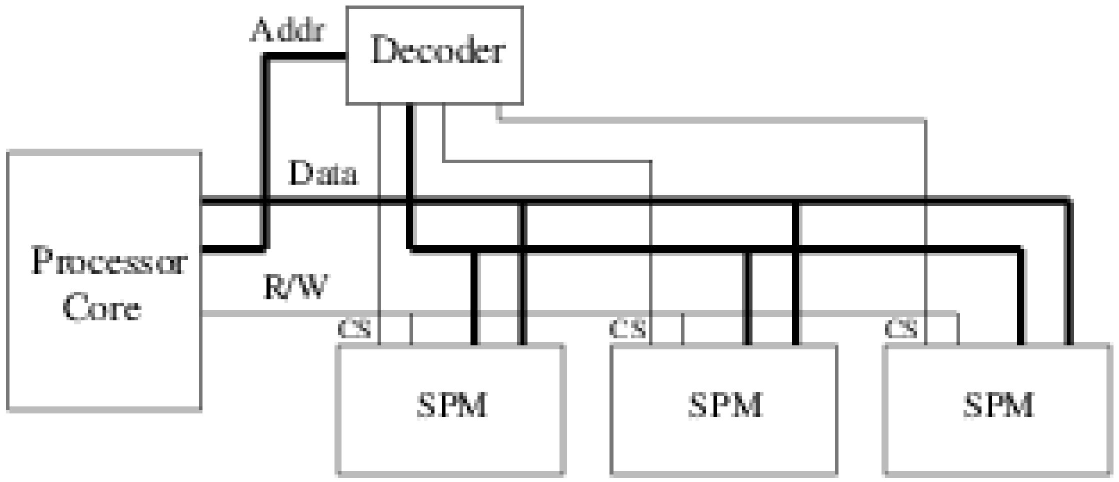
Monolithic SPM

Beneficial partitioning

$$E_{Bank1} + E_{Bank2} + \Delta E_{12} < E_{Monolithic\_SPM}$$

# On-chip Memory Banking





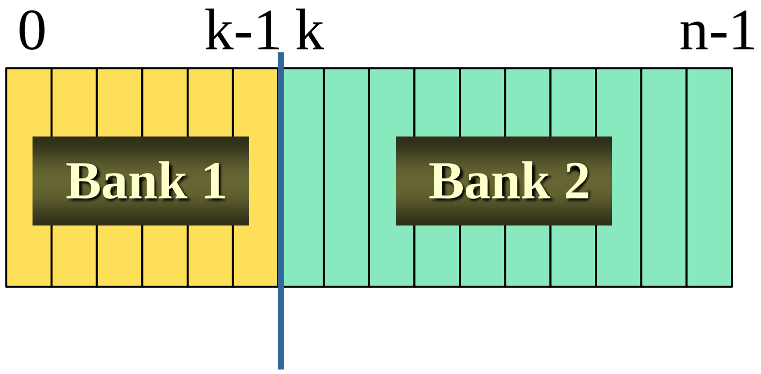
Architecture with monolithic SPM

Architecture with SPM partitioned in 3 banks

$\Delta E_{13}$

# Energy-aware SPM Banking

Exhaustive exploration with backtracking
Benini *et al.*, *IEEE Trans. CAD 2002*



Optimal 2-way partitioning

$$\text{Min}_k\{E_{\text{Bank1}} + E_{\text{Bank2}}\} + \Delta E_{12}$$

Using recursion → Optimal M-way partitioning

Exhaustive exploration with backtracking
Benini *et al.*, *IEEE Trans. CAD 2002*

- for M = 2, 3 : optimal solutions
- for M = 4 : optimal solution
- for M > 4

- **CPU: usually over 1 hour**
- **Partitions: many billions**

Exploration using dynamic programming
Angiolini *et al.*, *IEEE Trans. CAD 2005*

- Advantage: no need of presetting M
- Disadvantage: a huge data structure (matrix: | SPM | x | TraceAccesses | )

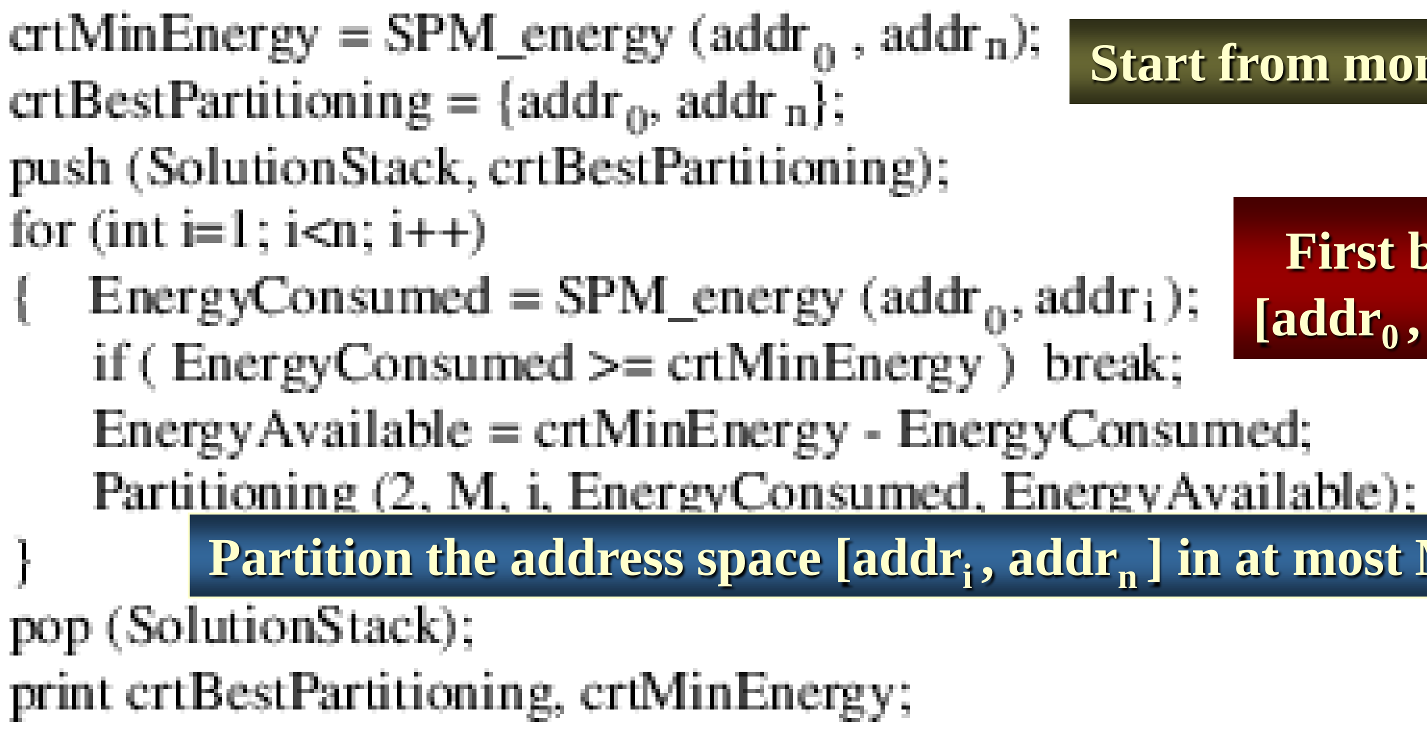
# Energy-aware SPM Banking

## The Banking Algorithm

| | |
|---|---|
| **Input 1** | maximum number of banks M |
| **Input 2** | array $[\Delta E_{12}, \Delta E_{23}, \ldots, \Delta E_{M-1,M}]$ whose elements $\Delta E_{k,k+1}$ are energy overheads implied by moving from k to k+1 banks |
| **Input 3** | array of SPM addresses where the lattices of signals are mapped into the on-chip memory |
| **Input 4** | array of R/W accesses for the on-chip lattices |
| **Output** | array of energetically-optimal bank boundaries |

# Energy-aware SPM Banking

## The Banking Algorithm



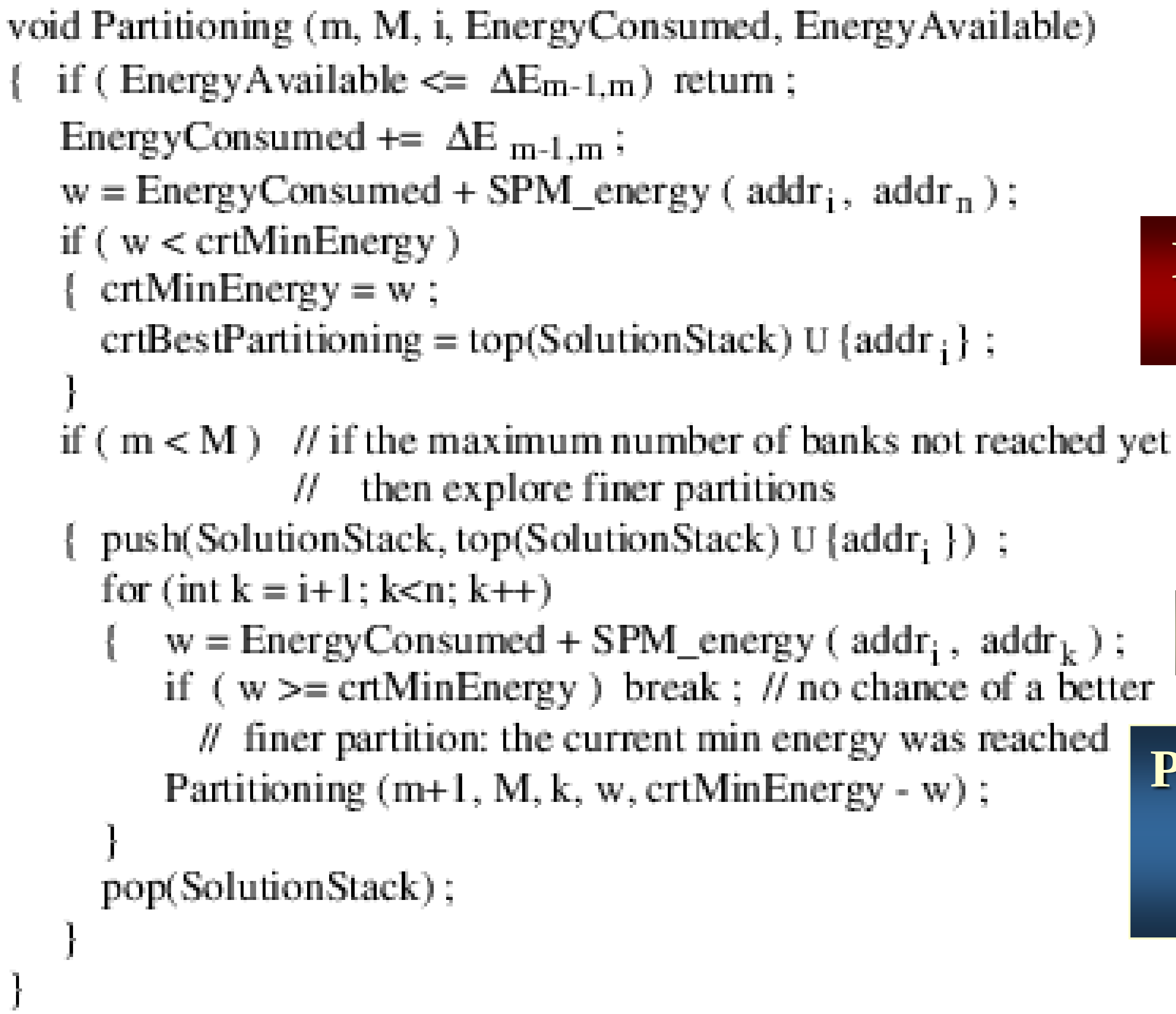
```
crtMinEnergy = SPM_energy ($addr_0$ , $addr_n$);
crtBestPartitioning = {$addr_0$, $addr_n$};
push (SolutionStack, crtBestPartitioning);
for (int i=1; i<n; i++)
{    EnergyConsumed = SPM_energy ($addr_0$, $addr_i$);
     if ( EnergyConsumed >= crtMinEnergy )  break;
     EnergyAvailable = crtMinEnergy - EnergyConsumed;
     Partitioning (2, M, i, EnergyConsumed, EnergyAvailable);
}
pop (SolutionStack);
print crtBestPartitioning, crtMinEnergy;
```

**Start from monolithic SPM**

**First bank: [$addr_0$ , $addr_i$ ]**

**Partition the address space [$addr_i$ , $addr_n$ ] in at most M-1 banks**

# Energy-aware SPM Banking

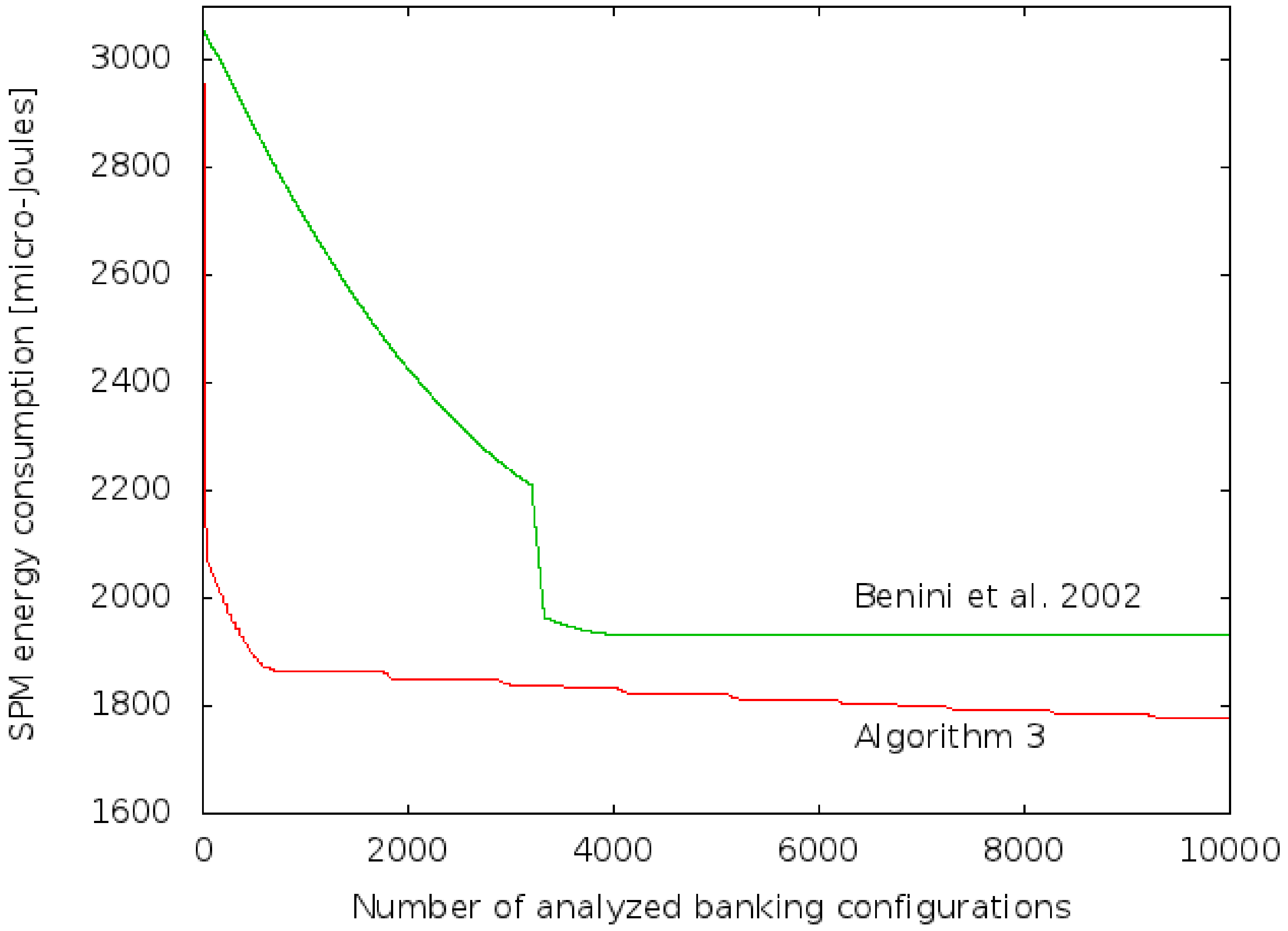


```
void Partitioning (m, M, i, EnergyConsumed, EnergyAvailable)
{   if ( EnergyAvailable <= $\Delta E_{m-1,m}$ ) return ;
    EnergyConsumed += $\Delta E_{m-1,m}$ ;
    w = EnergyConsumed + SPM_energy ( $addr_i$ , $addr_n$ ) ;
    if ( w < crtMinEnergy )
    {  crtMinEnergy = w ;
       crtBestPartitioning = top(SolutionStack) U {$addr_i$} ;
    }
    if ( m < M )   // if the maximum number of banks not reached yet
                   //   then explore finer partitions
    {  push(SolutionStack, top(SolutionStack) U {$addr_i$ }) ;
       for (int k = i+1; k<n; k++)
       {    w = EnergyConsumed + SPM_energy ( $addr_i$ , $addr_k$ ) ;
            if ( w >= crtMinEnergy ) break ; // no chance of a better
              // finer partition: the current min energy was reached
            Partitioning (m+1, M, k, w, crtMinEnergy - w) ;
       }
       pop(SolutionStack) ;
    }
}
```

**Last bank: [$addr_i$ , $addr_n$ ]**
**A better solution found !!**

**Bank m: [$addr_i$ , $addr_k$ ]**

**Partition the address space**
**[$addr_k$ , $addr_n$ ]**
**in at most M-m banks**



**Energy-aware SPM banking: the decrease of the SPM energy consumption as different SPM partitions are analyzed**

| Application | Scalars | Memory accesses | Energy DRAM [µJ] | SPM size | Energy savings DRAM+SPM | Energy savings [Brockmeyer] | CPU [sec] |
|---|---|---|---|---|---|---|---|
| Dynamic programming | 21,082,751 | 83,834,000 | 2,515,020 | 32K | 56.1 % | 35.1 % | 36.2 |
| Motion estimation | 265,633 | 864,900 | 25,947 | 1K | 50.7 % | 28.6 % | 12.5 |
| Durbin's algorithm | 252,499 | 1,004,993 | 30,150 | 512 | 62.2 % | 37.4 % | 8.9 |
| SVD updating | 3,045,447 | 29,500,000 | 885,000 | 16K | 46.5 % | 28.2 % | 24.5 |

## Experimental Results on Signal Assignment to the On-Chip and Off-Chip Memory Layers (dynamic + static energy evaluations: CACTI 6.5)

# Experimental Results

| Application | Address space | CPU Full_Expl. (M=4) [sec] | CPU [sec] M=8 | Energy savings vs. Full_Expl. (M=4) | Energy savings vs. monolithic SPM (M=8) |
|---|---|---|---|---|---|
| | | | | | |
| Motion detection | 9,600 | 5,289 | 32.2 | 9.69 % | 55.78 % |
| Motion estimation | 1,024 | 736 | 5.3 | 7.51 % | 47.21 % |
| Durbin's algorithm | 512 | 247 | 2.3 | 7.28% | 46.52 % |
| SVD updating | 16,384 | 7,524 | 65.2 | 10.94 % | 59.88 % |

Experimental Results on
Energy-Aware SPM Banking

## Conclusions

- Integrated CAD methodology for system-level exploration, focusing on memory management tasks
- Energy-aware data assignment technique based on a library of operations with integral polyhedra and lattices
- The assignment algorithm could be extended for an arbitrary number of layers of memory hierarchy if the functions of energy spent per access and static power versus memory size were available for each layer
- Energy-aware banking algorithm based on the computation of the intensity of memory accesses in the index space of arrays

The End

Thank you!