# Hierarchical Memory Size Estimation for Loop Fusion and Loop Shifting

**Qubo Hu**[1], Arnout Vandecappelle[2] ,
Martin Palkovic[2], Per Gunnar Kjelsberg[1]
Erik Brockmeyer[2], Francky Catthoor[2,3]

[1] Norwegian University of Science and Technology, Trondheim, Norway
[2] IMEC, Leuven, Belgium
[3] KU Leuven, Belgium

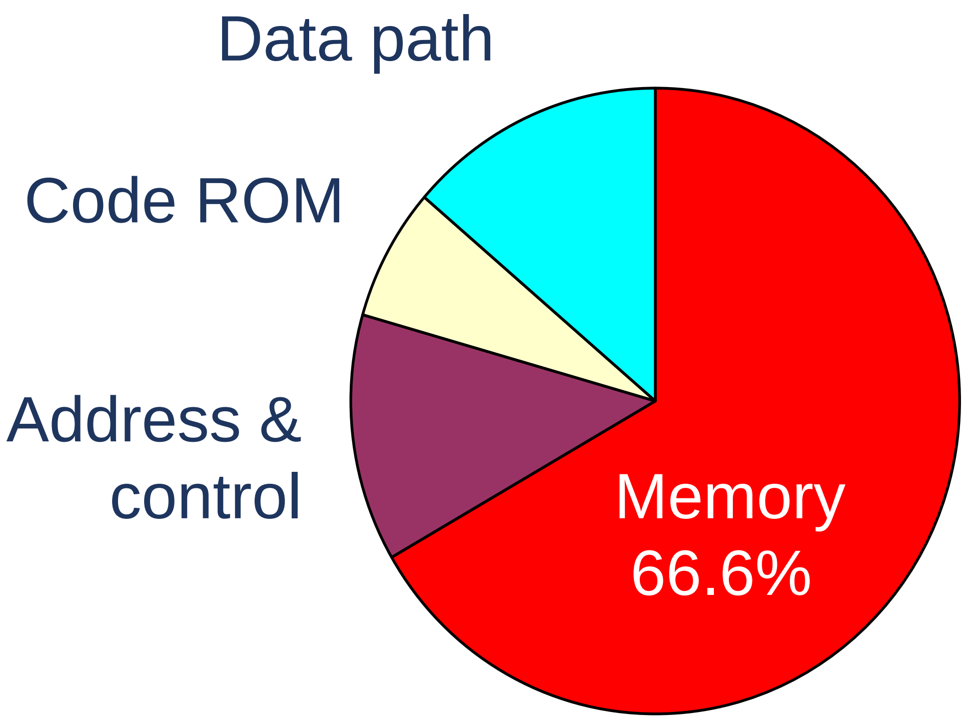
## Outline

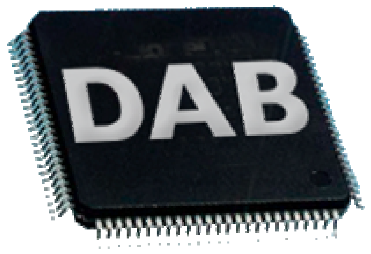
- Motivation
- Previous work
- Hierarchical Memory Size Estimation (HMSE) for loop transformations -- loop fusion & loop shifting
    - 1st round HMSE from source code
    - Incremental HMSE
- Experiments
- Conclusion

## Data transfer and storage is bottleneck

Data transfer and storage dominate

- Power consumption
- Size
- Performance

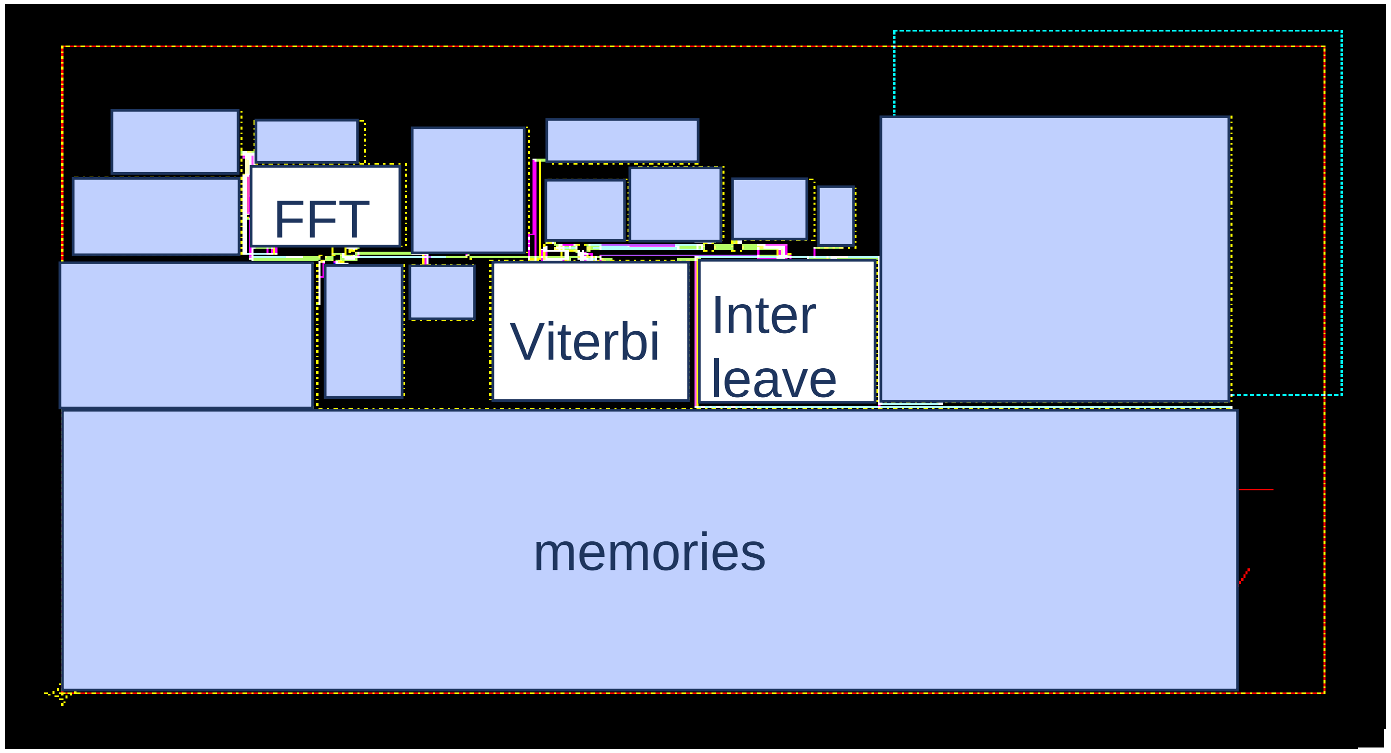




## Data transfer and storage is bottleneck

Data transfer and storage dominate

- Power consumption
- Size
- Performance

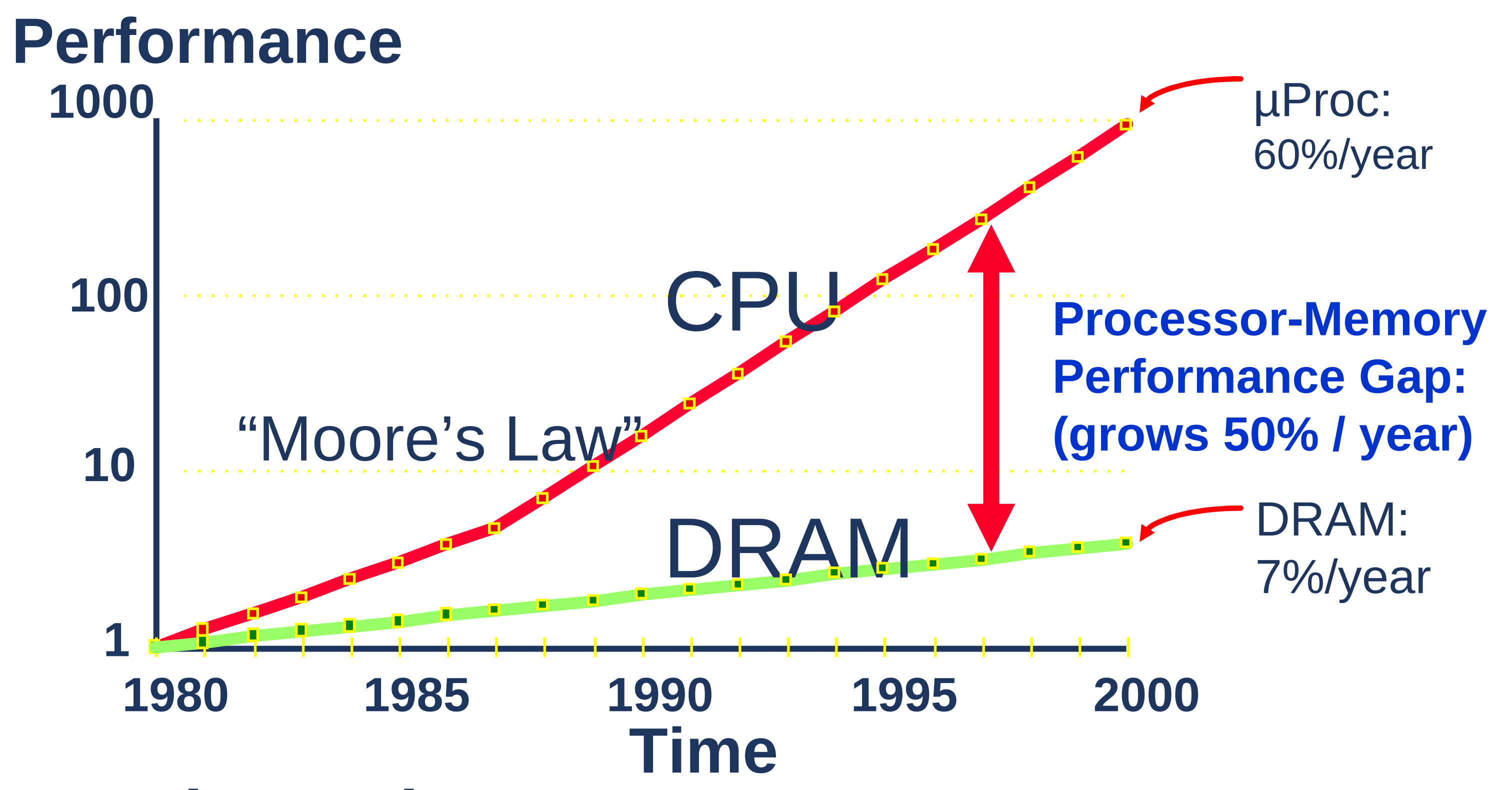


Digital Audio Broadcast (DAB) receiver chip

## Data transfer and storage is bottleneck

Data transfer and storage dominate

- Power consumption
- Size
- Performance



[Patterson]

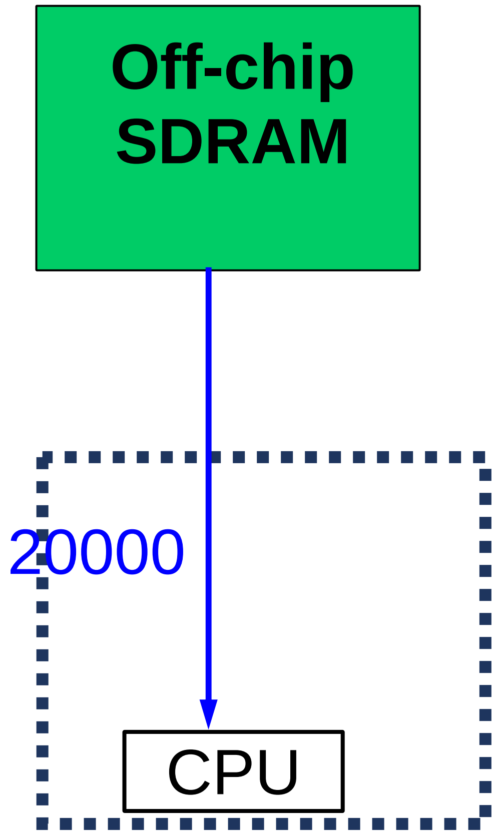
## Loop Transformations (LTs) are one of the most important high-level optimizations

- By improving data access locality and regularity
  - Shorten data access lifetime
  - Potentially give better data reuse and in-place mapping, and reduce the accesses to off-chip memory

- This paper focus on loop fusion and loop shifting

Before loop fusion

```
for(x=0;x<99;x++)
    for(y=0;y<99;y++)
        = f1( A[x][y] );
for(x=0;x<99;x++)
    for(y=0;y<99;y++)
        = f2( A[x][y] );
```

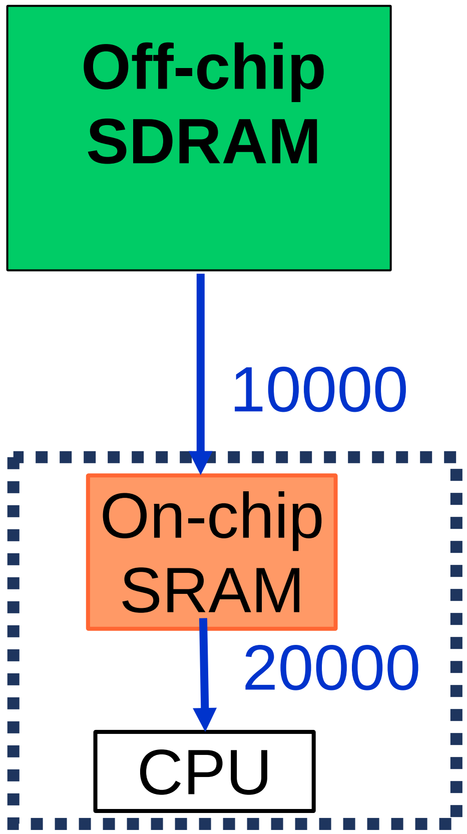


## Loop Transformations (LTs) are one of the most important high-level optimizations

- By improving data access locality and regularity
  - Shorten data access lifetime
  - Potentially give better data reuse and in-place mapping, and reduce the accesses to off-chip memory
- This paper focus on loop fusion and loop shifting

After loop fusion

```
for(x=0;x<99;x++)
    for(y=0;y<99;y++)
        = f1( A[x][y] );
        = f2( A[x][y] );
```



## Problem definition

- Improving data locality & regularity is very abstract cost function
  - DOES NOT represent the actual memory platform utilization
  - Different LTs can be optimal for different memory platforms
- LTs are usually performed globally at the high level
  - Memory platform is not defined yet
  - Huge number of LT possibilities exist
    - i.e. to find the optimal loop fusion for array contraction is a NP-problem
  - Ad-hoc decison might lead to final sub-optimal solution
    - Existing heuristics  for LTs usually result in single LT instance
- Estimation is hence necessary to find all the good LTs
  - Can Help find the right LTs when a memory platform is given later

## Previous works

- Previous works on memory size requirement estimation
  - [Balasa95], [Grun98], [Zhao99], [Kjeldsberg02], [Rydlandl03], [Hu04]
  - All focus on one layer memory
  - Not represent how the memory platform is exploited
    - Multiple layer memory hierarchy architecture widely used

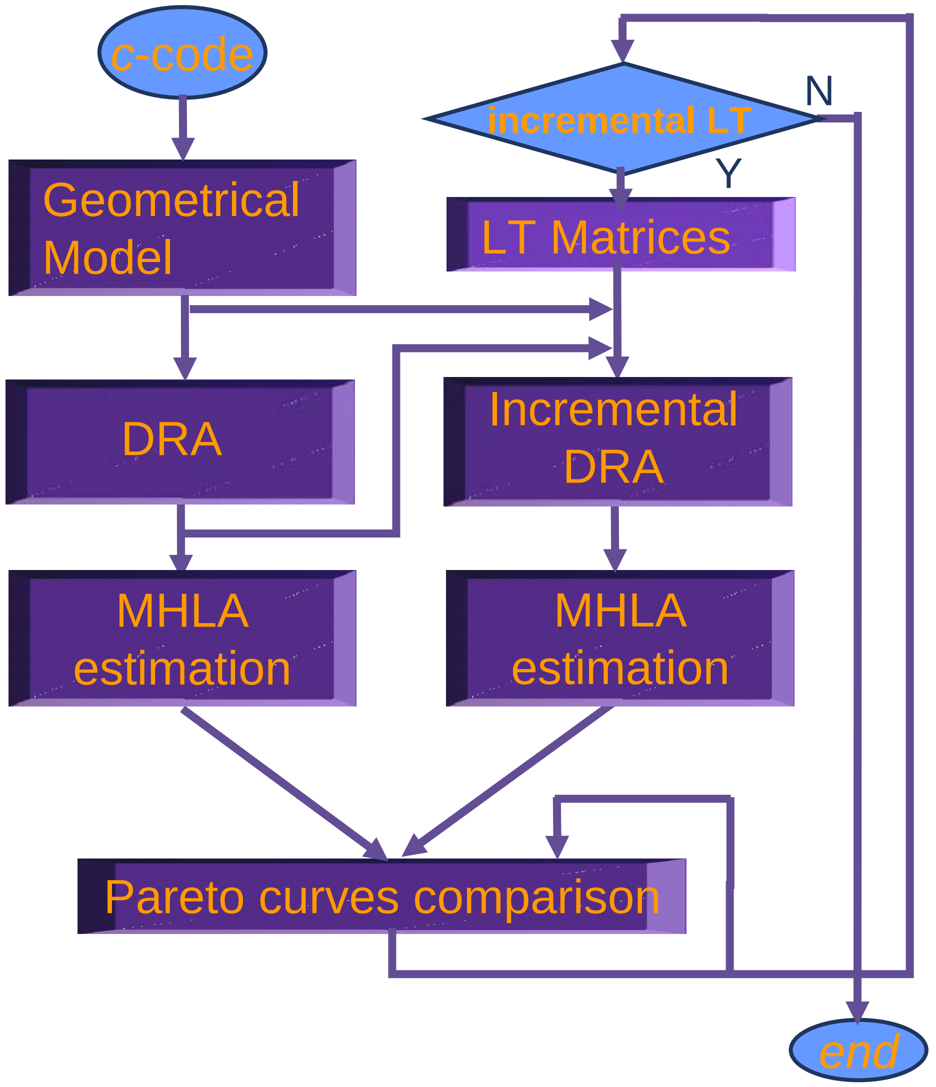
- We propose to do hierarchical memory size estimation to guide the LTs exploration
  - Target on Scratch-Pad Memory (SPM) based memory hierarchy platform

- Previous works on memory hierarchy platform exploitation
  - [Brockmeyer02] does the actual exploitation and too time consuming for the estimation purpose
  - [Kandemir02] only permits limited LTs

## HMSE flow graph

HMSE is classified as

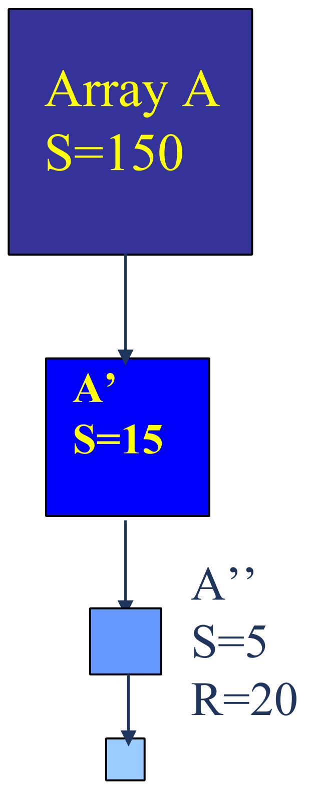
- 1st round HMSE
  - •Start from *c*-code
- Incremental HMSE
  - •Proceeded when LTs are performed incrementally
  - •To improve the estimation computation time

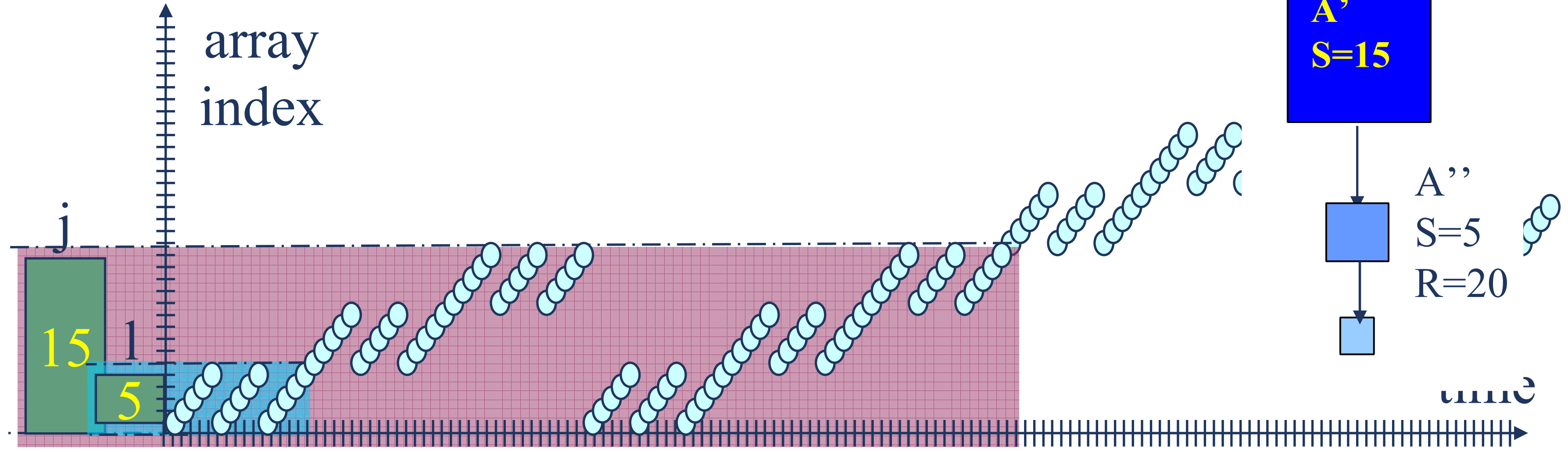


# Data Reuse Analysis (DRA)

- Identify the frequently accessed data
  - usually just a part of array
- Constructed at each loop nest dimension for every array

```
for (i=0; i<10; i++)
 for (j=0; j<2; j++)
  for (k=0; k<3; k++)
   for (l=0; l<3; l++)
    for (m=0; m<5; m++)
     … = A[i*15+k*5+m];
```

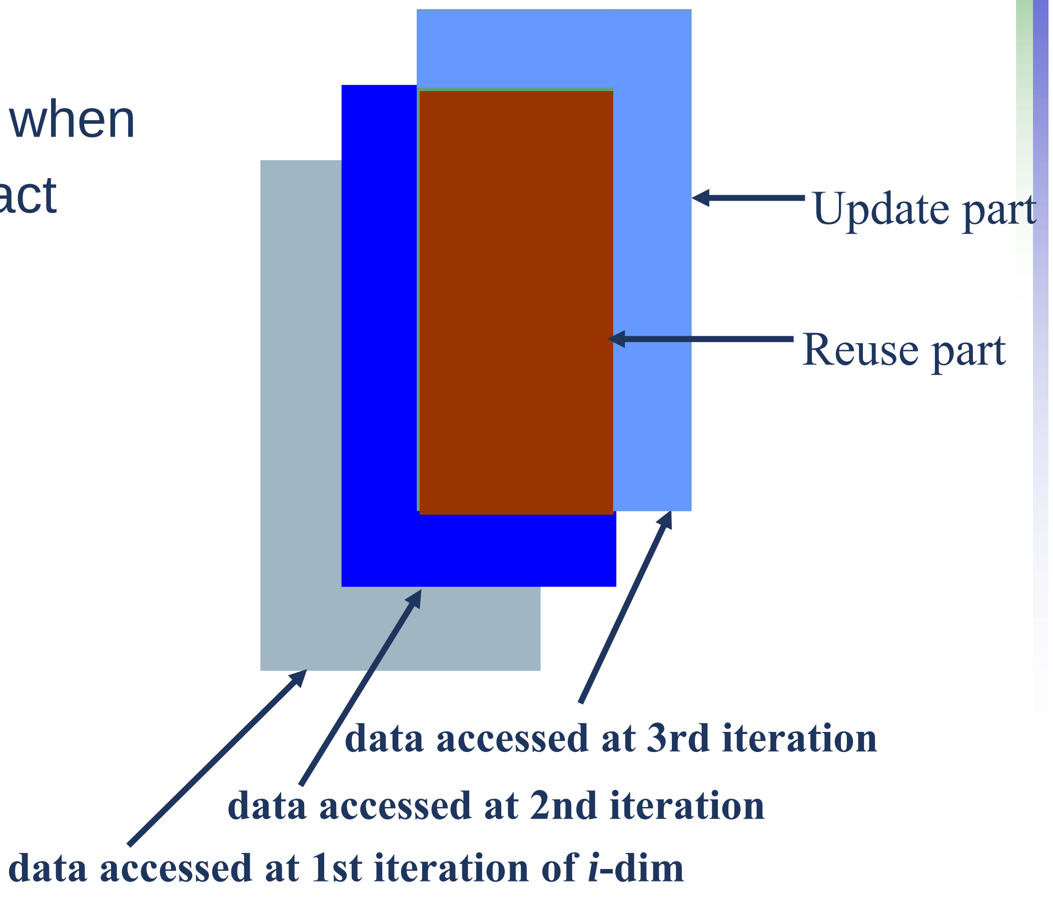




## Bounding-box based DRA

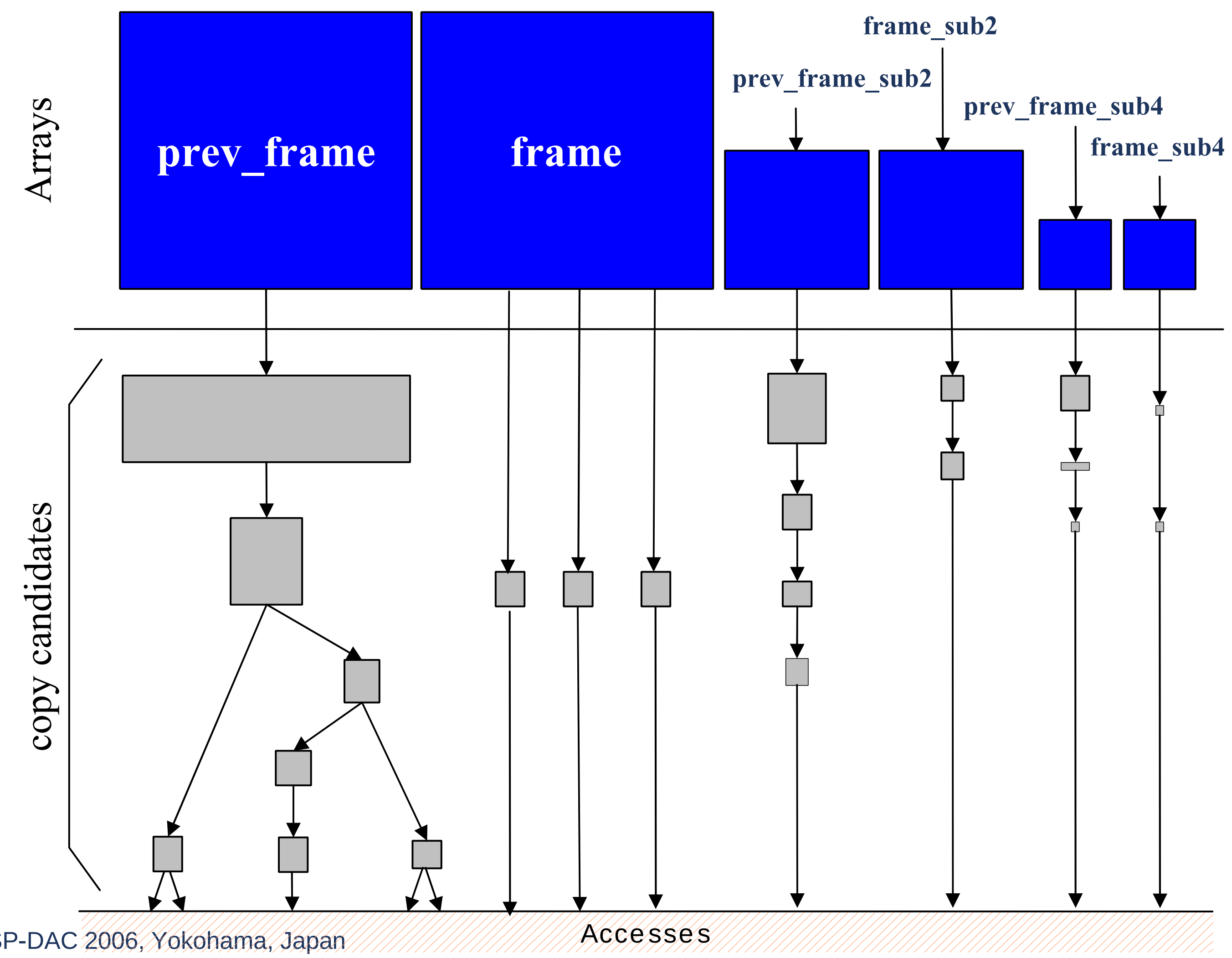
- The data accessed at each iteration at the analyzing dimension are represented as bounding box
  - Simple for analysis and also for the final code generation
  - Fast
  - Can loose some accuracy when the data accessed is not exact bounding box

```
for(i=0;i<10;i++)
  for(j=0;j<10;j++)
    for(k=0;k<5;k++)
      = fun( array[i+2k][i+k])
```


Update part
Reuse part
data accessed at 3rd iteration
data accessed at 2nd iteration
data accessed at 1st iteration of *i*-dim

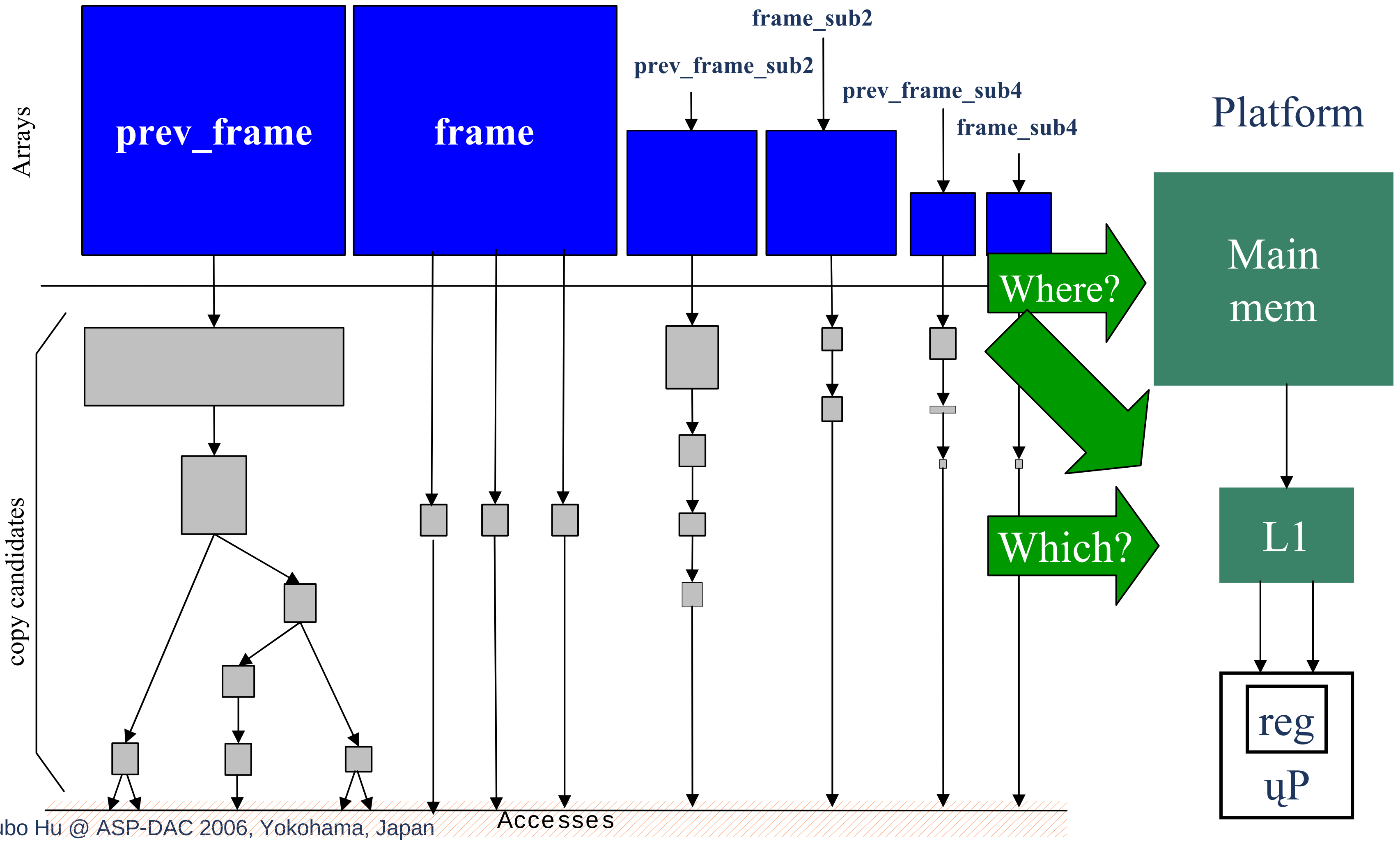
# Data reuse analysis output

Data Reuse Trees (DRTrees) output example



## Memory hierarchy layer assignment (MHLA)

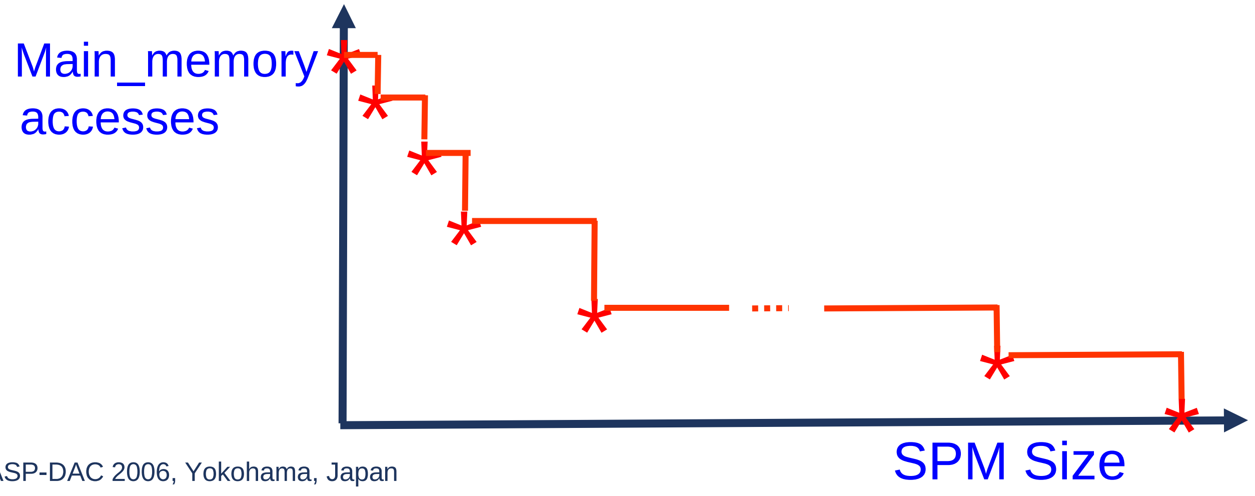
To map the heavily accessed data to the given memory hierarchy in order to optimize the power consumption

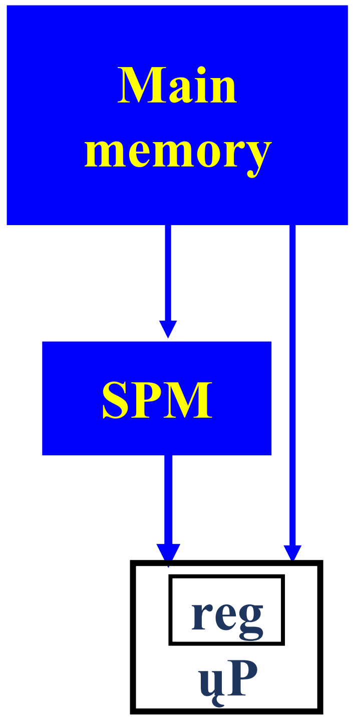


# Platform independent MHLA estimation

- Our MHLA estimation heuristic
  - Based on a two-layer memory hierarchy template
  - The most beneficial copy candidate (CC) or array is assigned at each time
    - •SPM layer size increasese
    - •Result in Pareto points

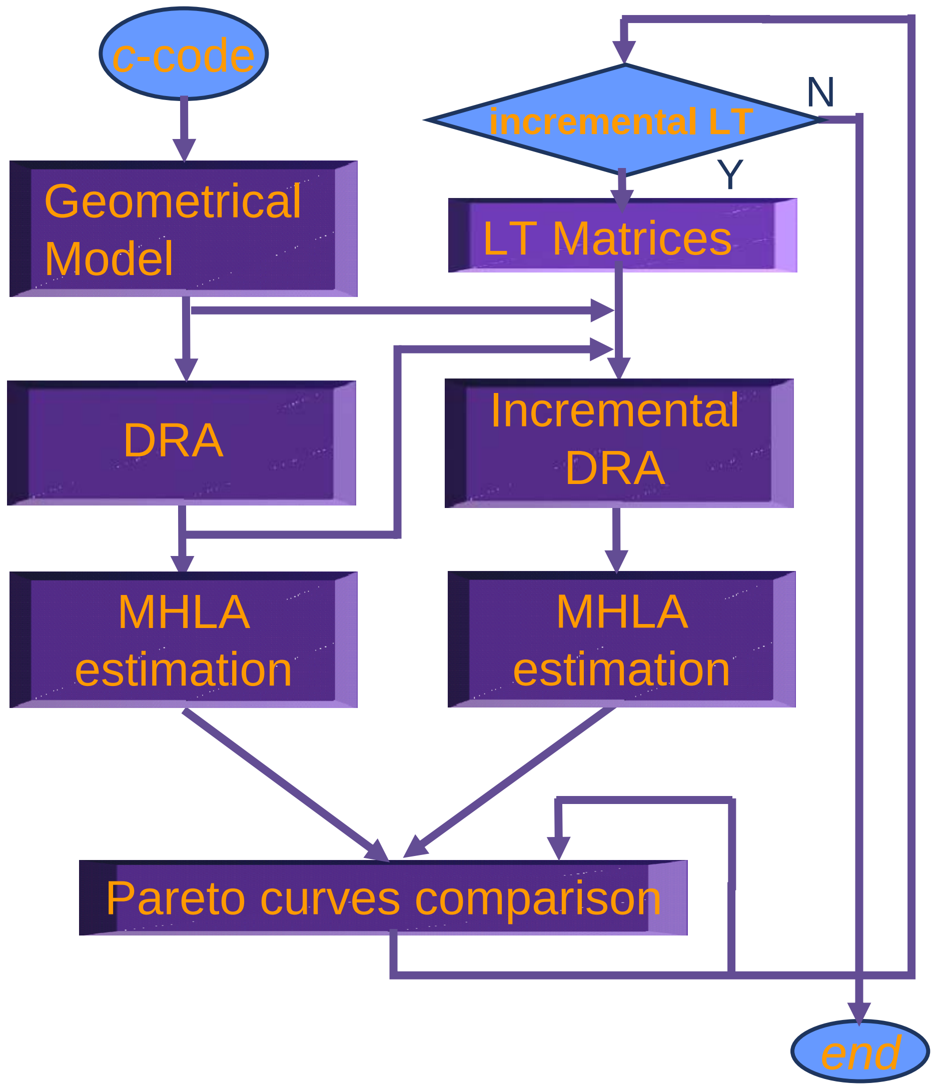
      $\#Accesses_{\mathbf{main\_memory}} \Leftrightarrow Size_{\mathbf{SPM}}$
    - •Output a Pareto curve
  - Fast as each CC and array is considered only once





## HMSE flow graph

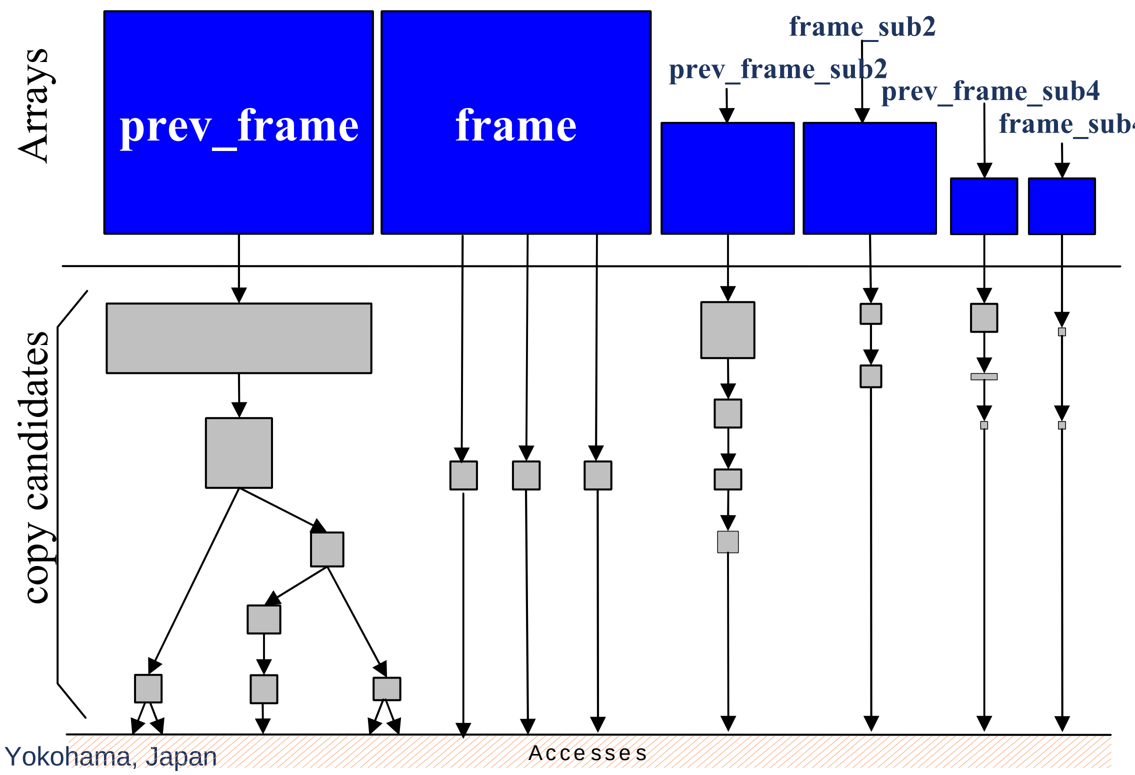
- Incremental HMSE
  - Proceeded when LTs are performed incrementally
  - Incremental LTs usually have local effects
    - •No need to redo the whole HMSE
  - Consists of
    - •Incremental DRA
    - •MHLA estimate
  - LTs performed are represented as matrix operations



## Incremental DRA: how-to-do

Three alternatives based on the evaluation of LT matrices

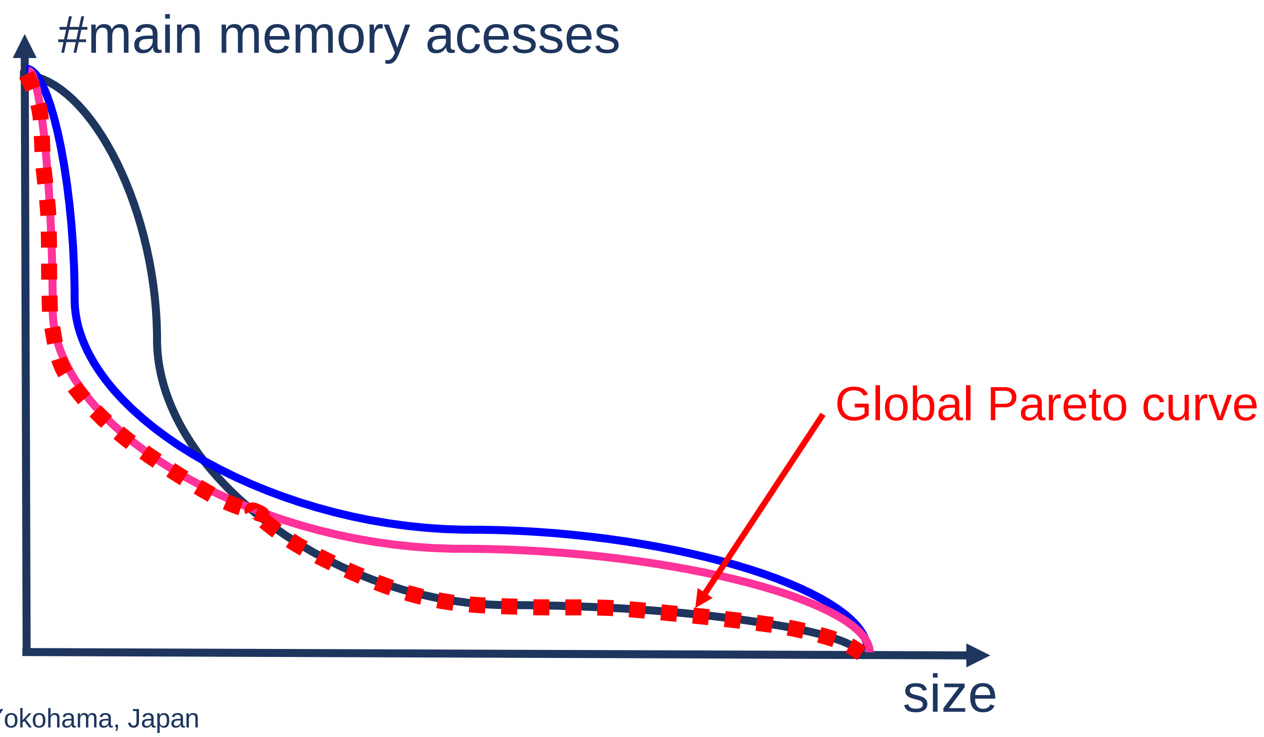
- Rebuild DRTrees for all arrays
- Only rebuild DRTrees for transformed arrays
- Locally update DRTrees
  - •Only update DRTrees at where transformations take effects



## Global Pareto curve generation

A global Pareto curve is generated among all Pareto curves

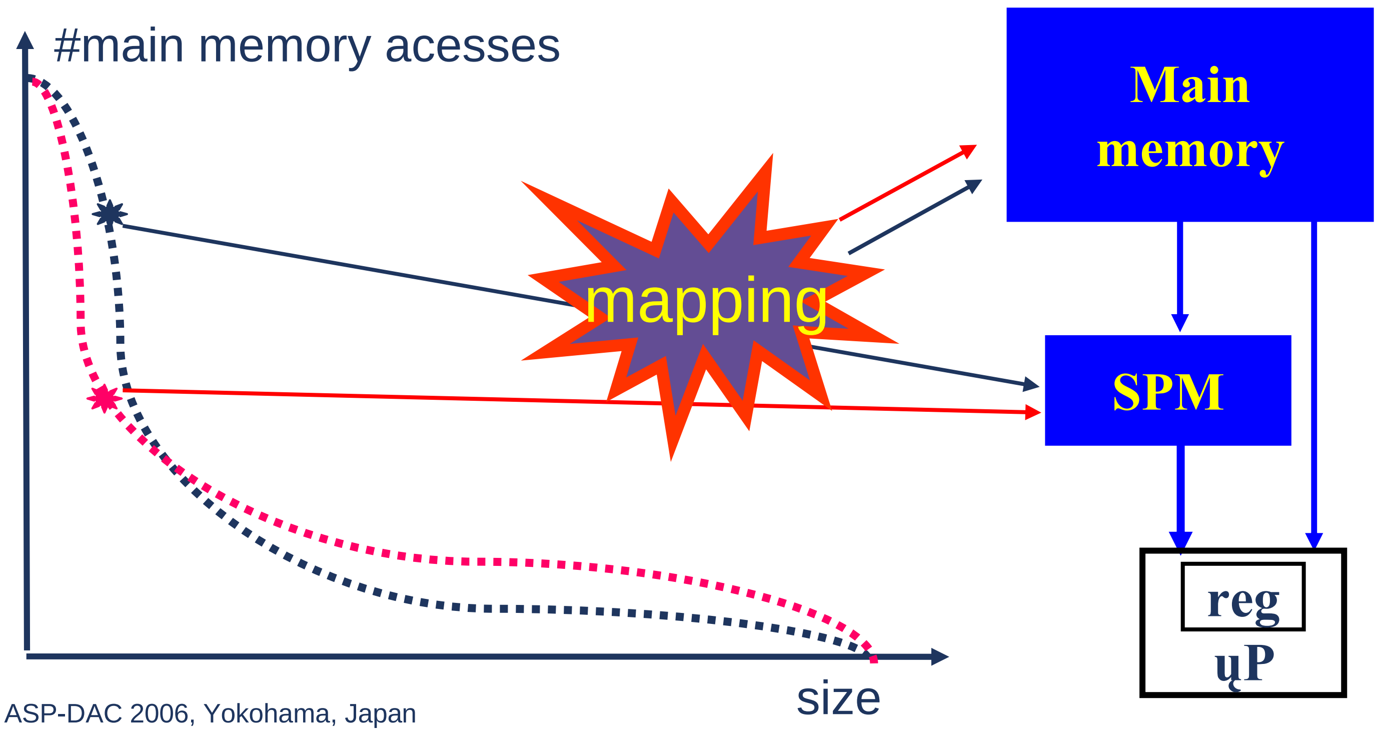
- Consists of all the optimal Pareto points
  - The one having lowest main memory accesses at the same SPM size among all points
- All LTs contributed to the optimal Pareto point are kept for that point



## Power estimate

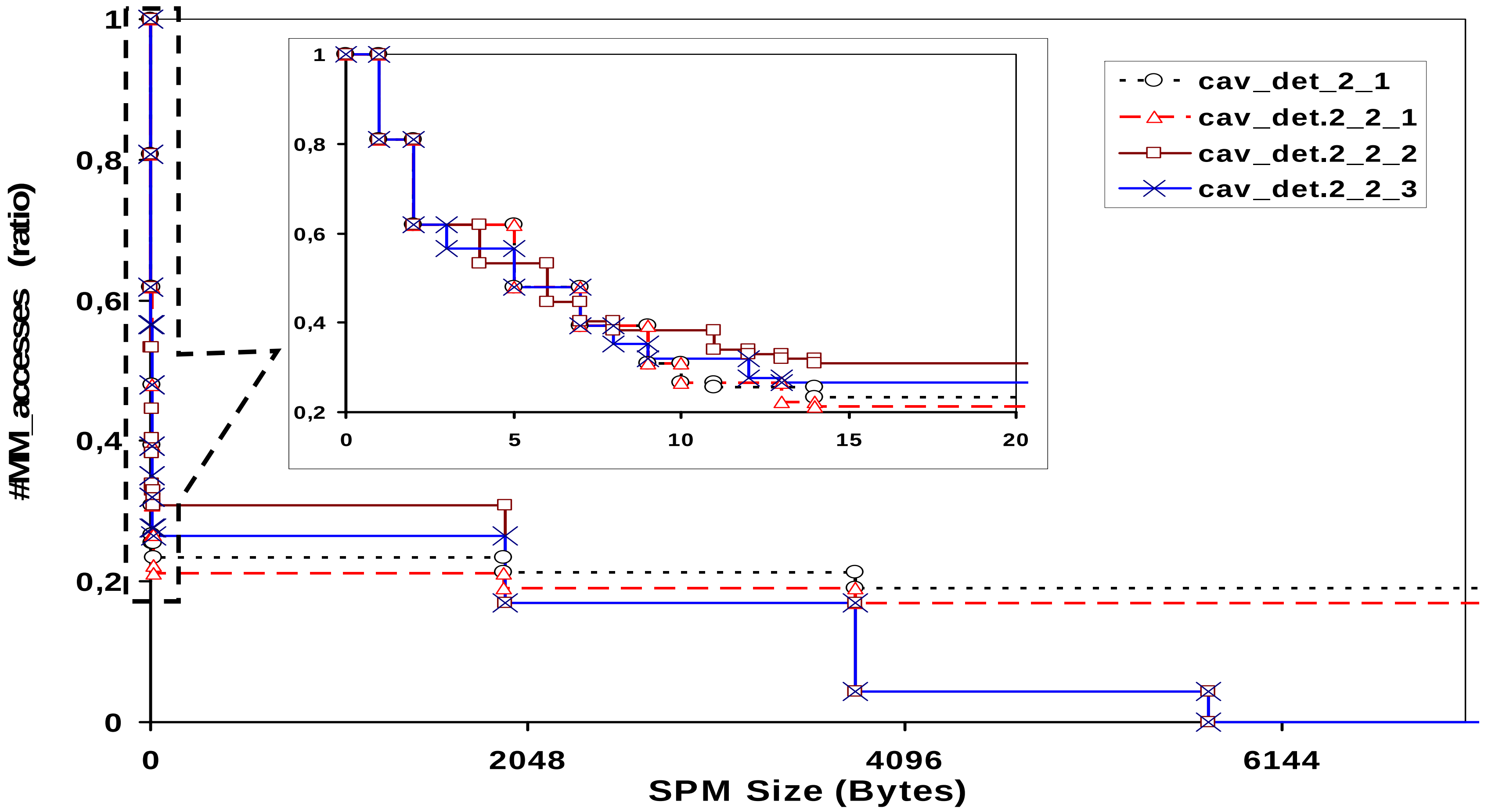
Pareto curve enables a power estimation for any two layer memory platform configurations

- Principle: the point having the closest size to the SPM layer for the given memory platform represents the optimal data mapping
- Total power = $\#accesses_{SPM} * power/access_{SPM} + \#accesses_{main\ memory} * power/access_{main\ memory}$

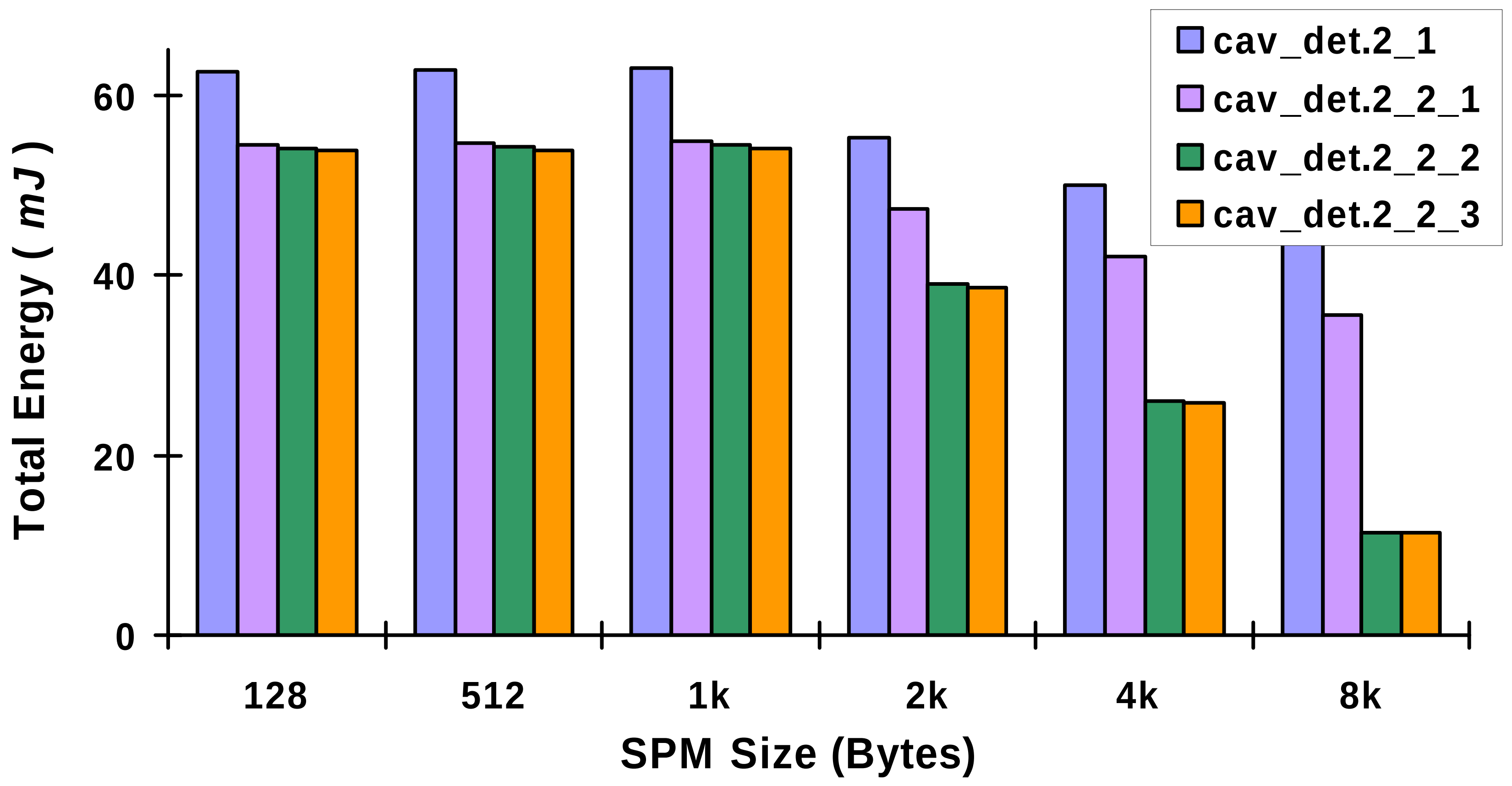


# Experiments on Cavity Detection

Pareto curves output for 4 selected versions of loop fusion and loop shifting performed

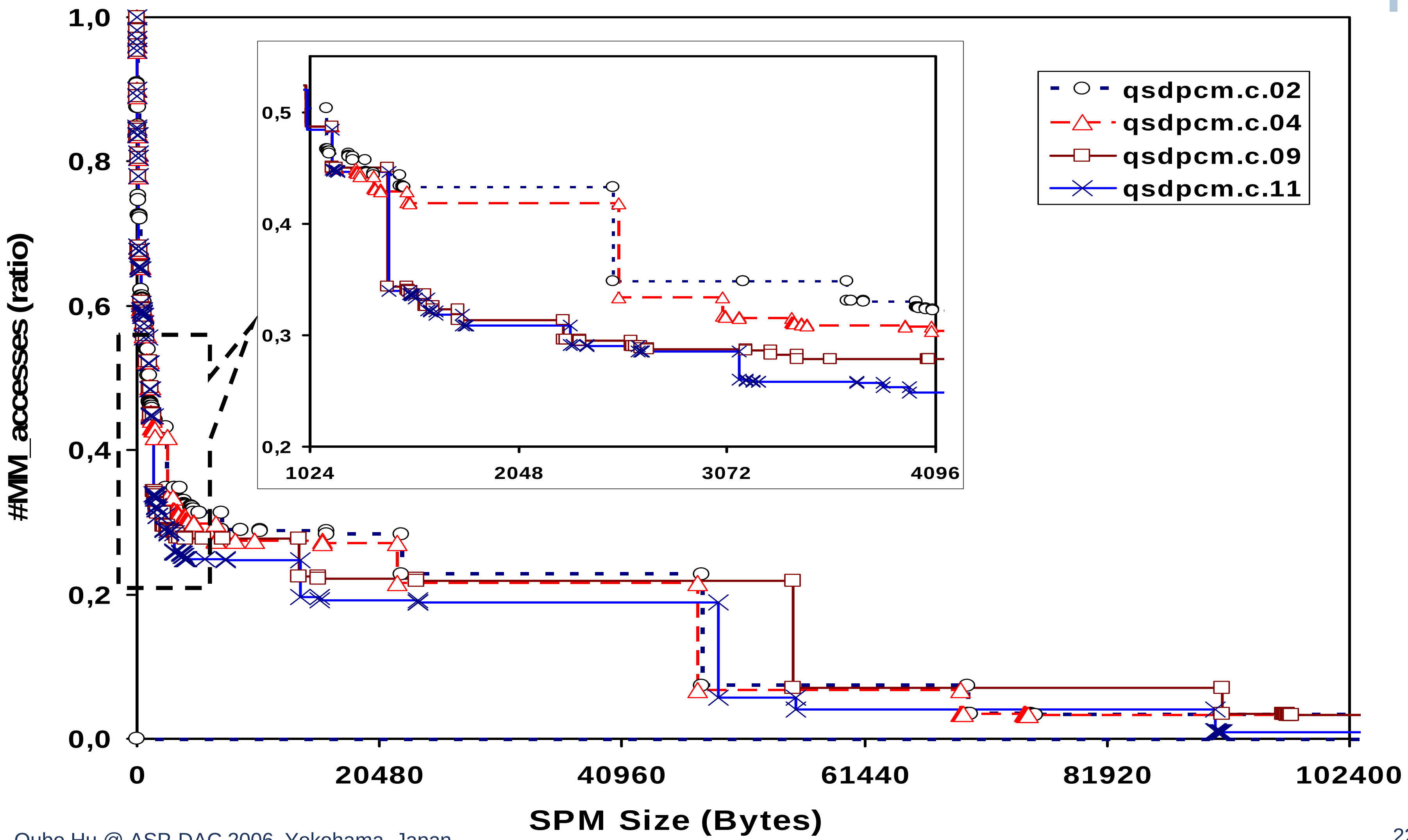


## Power estimate for Cavity Detection

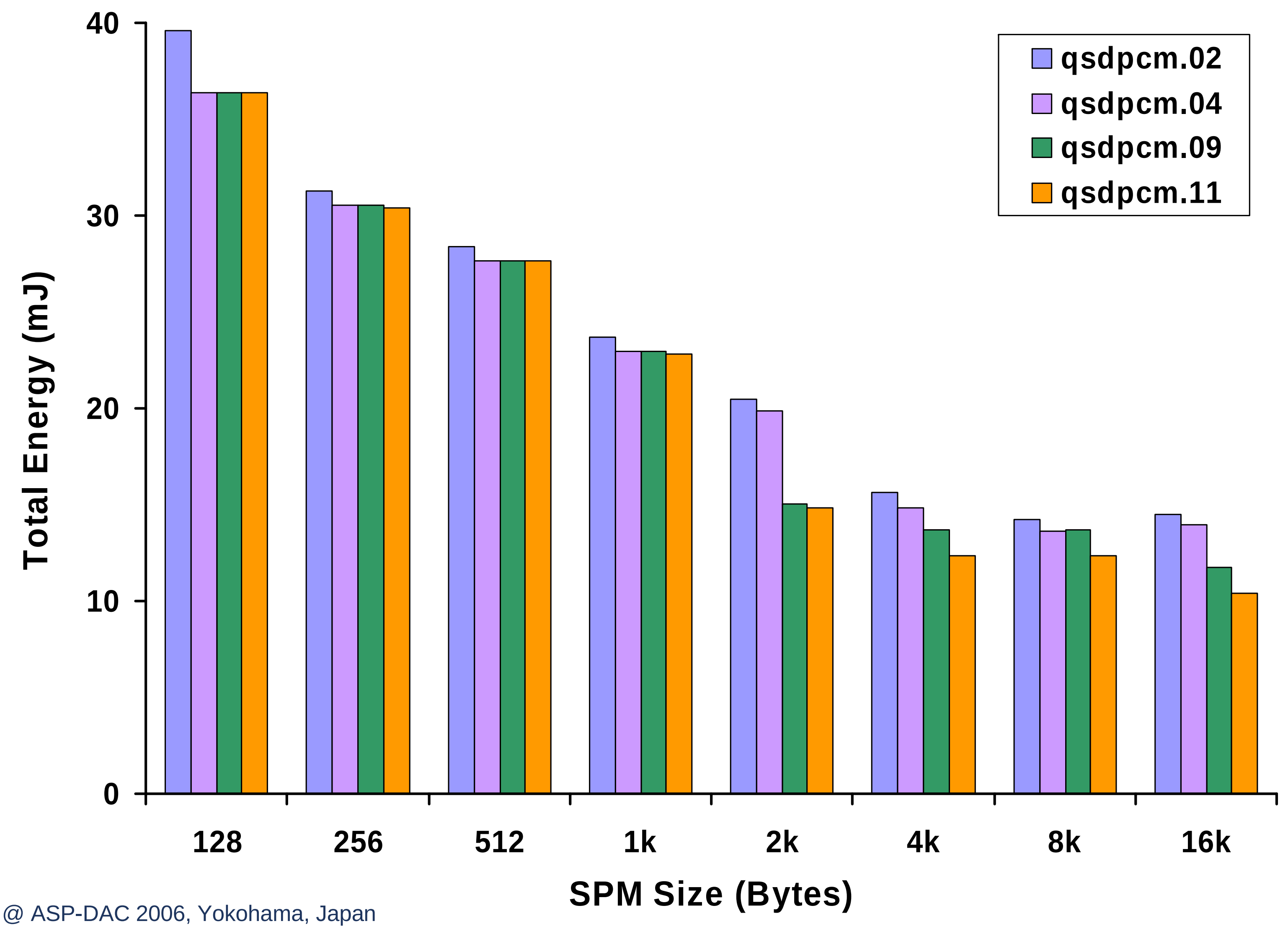


Less than 5% over-estimate compared to what is achieved with Atomium/MHLA tool

## Experiments on QSDPCM



# Power estimate for QSDPCM



## Conclusion and Future work

- A fast platform-independent HMSE methodology presented
  - Several techniques for improving the computation speed have been introduced
    - •Bounding-box based DRA
    - •A platform independent MHLA estimation
    - •Incremental DRA
- It reports all the good LTs performed
  - Experiments show our estimate is fast and also pretty accurate
  - Can help designer trade-off  the memory platform selection at later design stage
- Future work
  - To extend HMSE to support other affine loop transformations
  - To further take into account inplace mapping

Thanks you!

qubo.hu@iet.ntnu.no