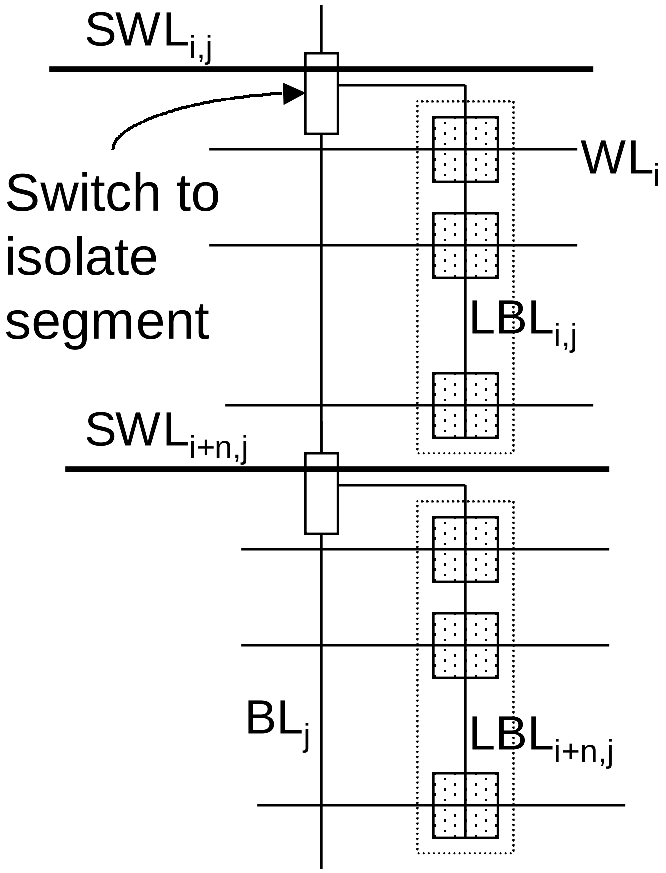
Reducing Memory Energy

- Improve the locality of memory accesses
  - Minimize the number of memory accesses
- Architectural and circuit techniques
- Optimizing interactions of compiler and cache architecture
- Technology changes

Cache: Bit Line Segmentation

- RAM cells in each column are organized into blocks selected by word lines
- Only the memory cells in the activated block present a load on the bit line
  - lowers power dissipation (by decreasing bit line capacitance)
  - can use smaller sense amps

Bit Line Segmentation



- Address decoder identifies the segment targeted by the row address and isolates all but the targeted segment from the common bit line
- Reduces bit line capacitance
- Has minimal effect on performance

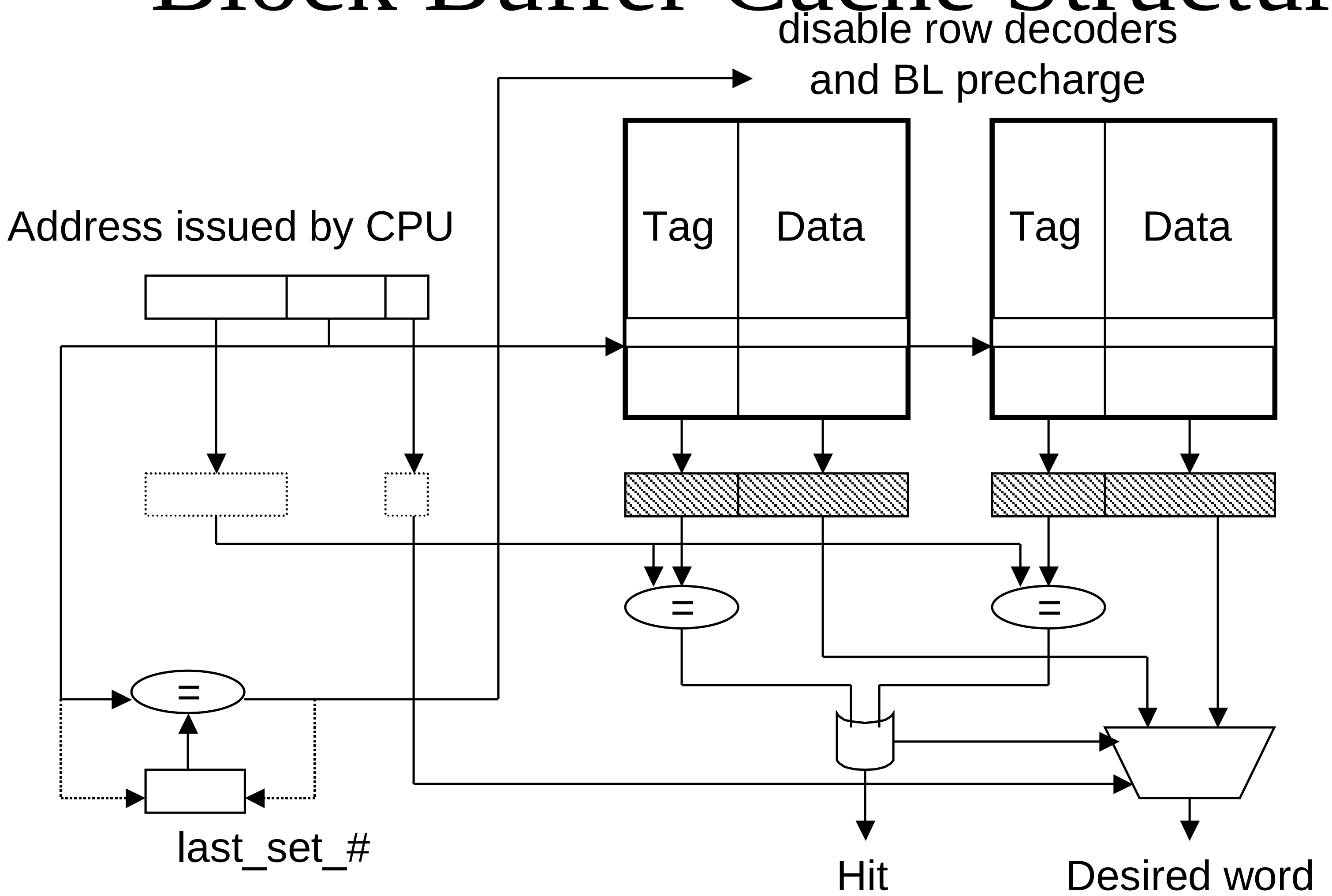
Other Circuit Optimizations

- Pulsed Word Line (PWL) and Isolated Bit Line (IBL)
  - Limit bit line swings
- HSPICE simulation of these designs for different organizations of 0.5Kbit SRAM
- 52% reduction when using DBL, PWL and IBL techniques
- Can capture influence in analytical model

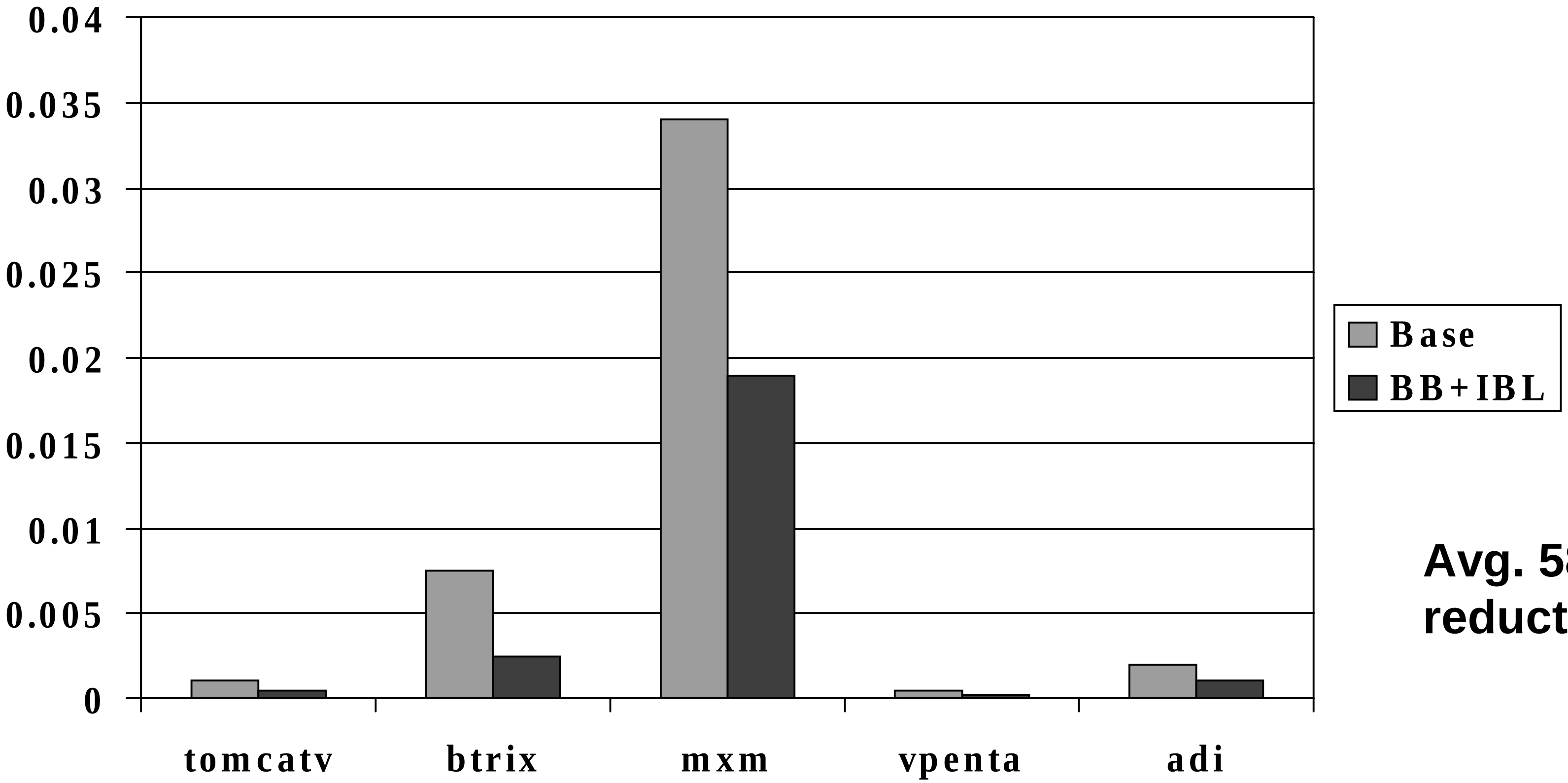
Cache Block Buffering

- Check to see if data desired is in the data output latch from the last cache access (i.e., in the same cache block)
- Saves energy since not accessing tag and data arrays
  - minimal overhead hardware
- Can maintain performance of normal set associative cache
- Analytical model can vary block buffer parameters

Block Buffer Cache Structure



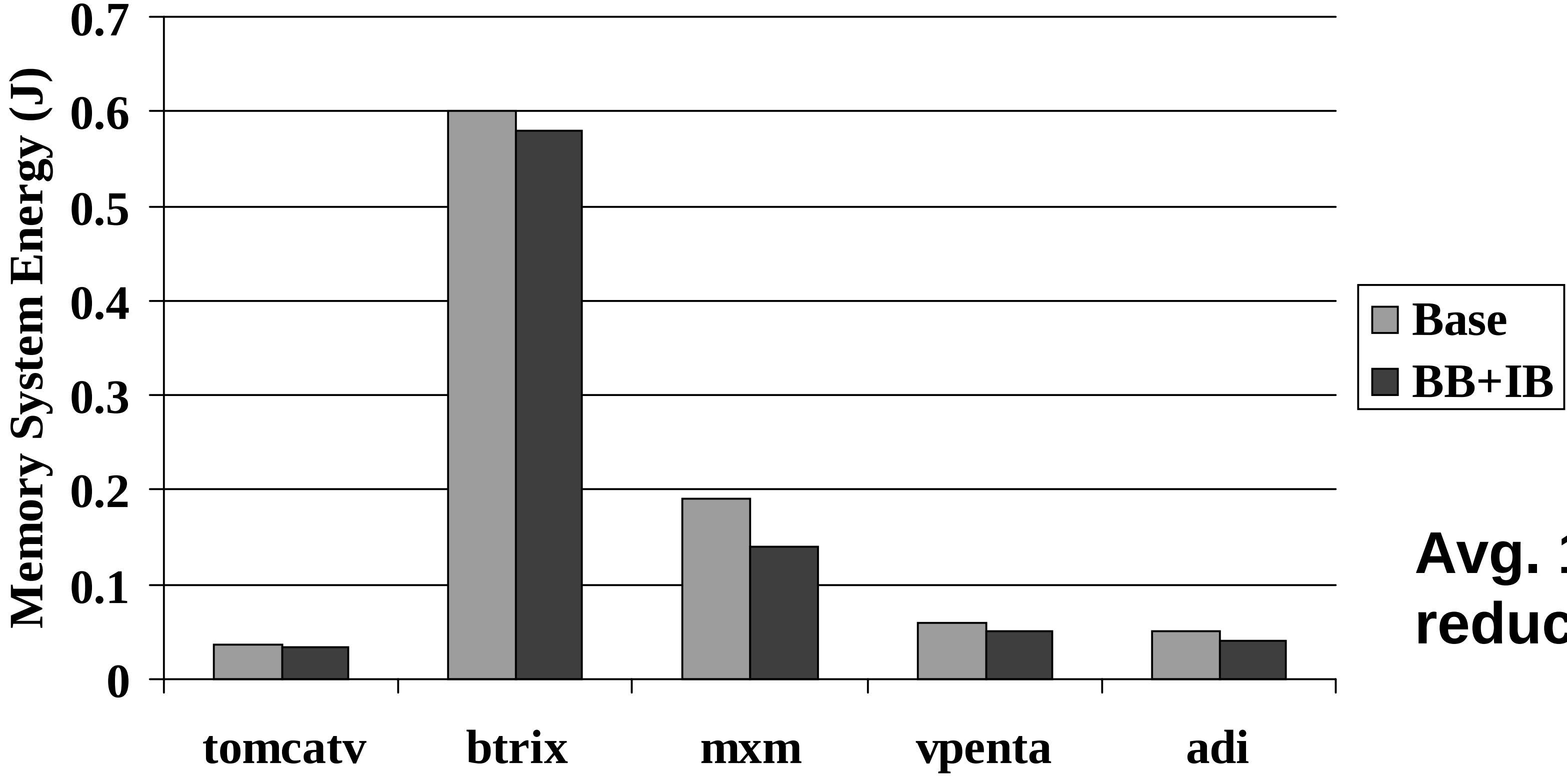
Dcache Energy



Avg. 58.7% reduction

8K 4way Dcache

Influence of Hardware Optimizations: Entire Memory View



Avg. 11% reduction

8K 4way Dcache

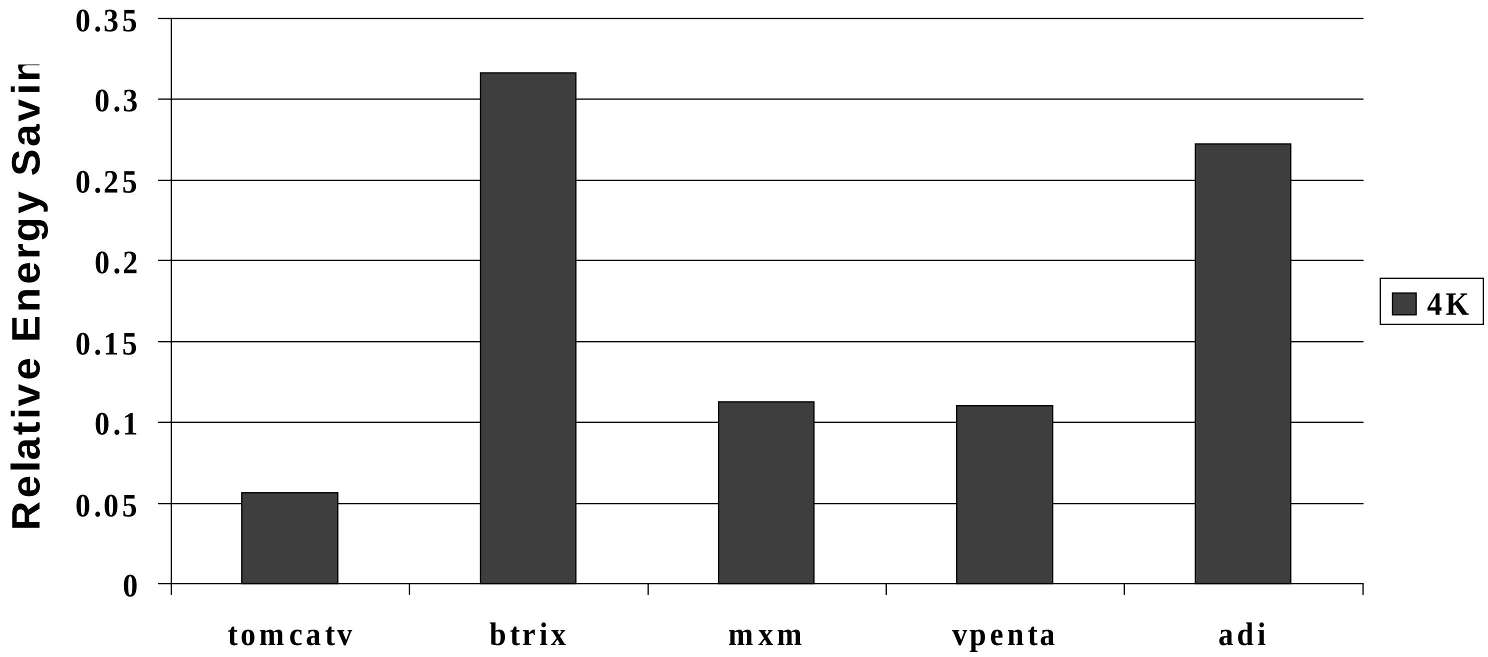
Reducing Memory Energy

- Improve the locality of memory accesses
  - Minimize the number of memory accesses
- Architectural and circuit techniques
- Optimizing interactions of compiler and cache architecture
- Technology changes

Relative Energy Savings

- (Energy_Reduction[w/Optimized Code]-Energy_Reduction[w/Original Code]) / Energy_Reduction[w/Original Code])
- Energy_Reduction due to hardware schemes

Compiler Optimizations and Block Buffering



Optimizing for Block Buffers

```
for(I=0; I<n; I++)
    for(J=0; J<n; J++)
        .. V[J][I];
```

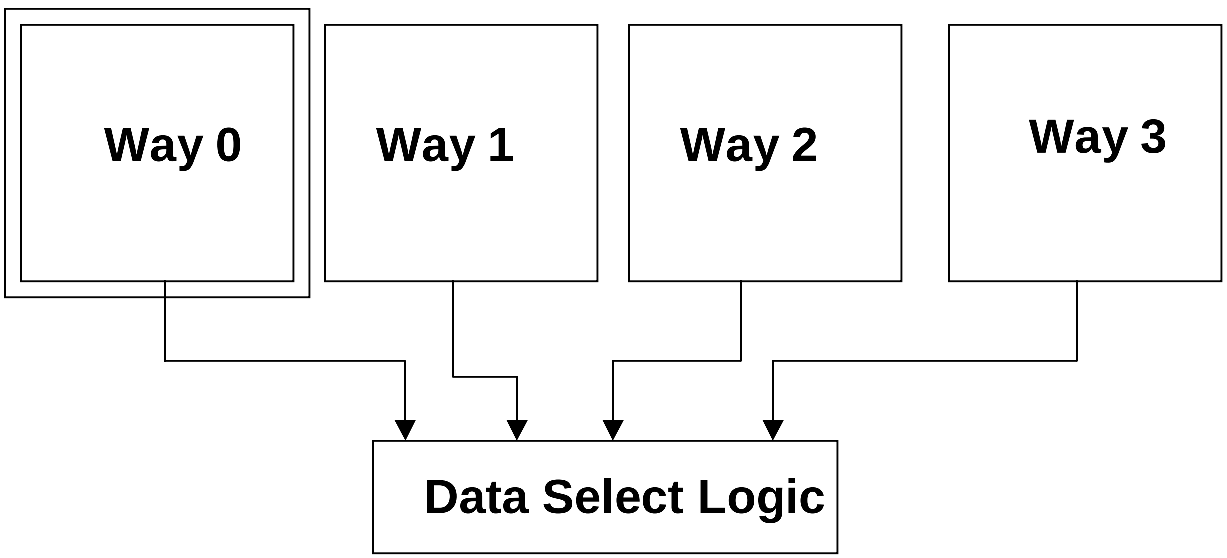
```
for(J=0; J<n; J++)
    for(I=0; I<n; I++)
        .. V[J][I];
```

Improves locality within block buffer

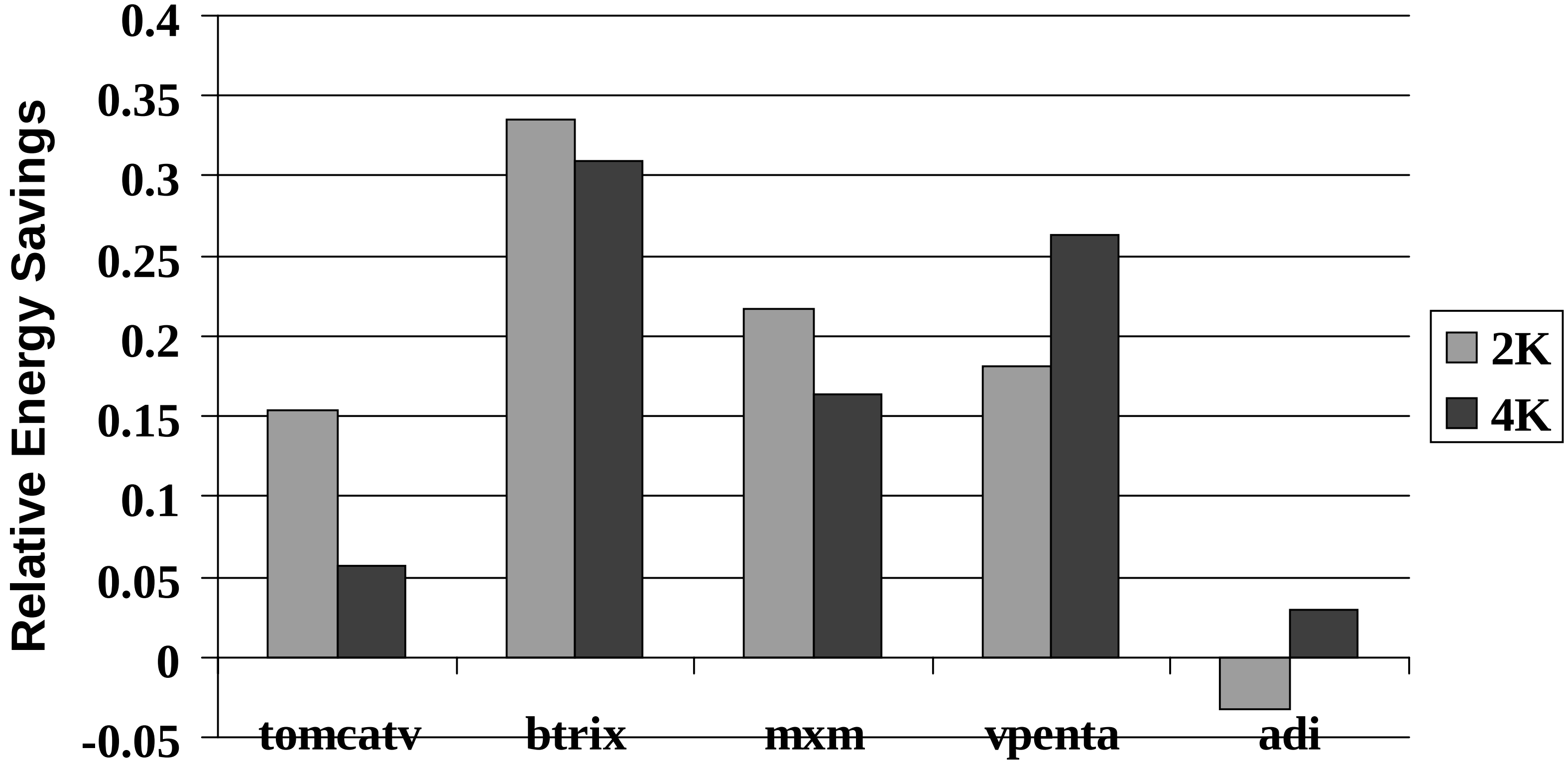
MRU Cache

- Access only the Most Recently Used way in a multi-way cache
- If it misses, access all the other ways
- If prediction is successful, can decrease the energy cost per access for other ways
- But has performance penalty and could increase system energy due to this penalty

Most Recently Used Cache



Compiler Optimizations and MRU Cache



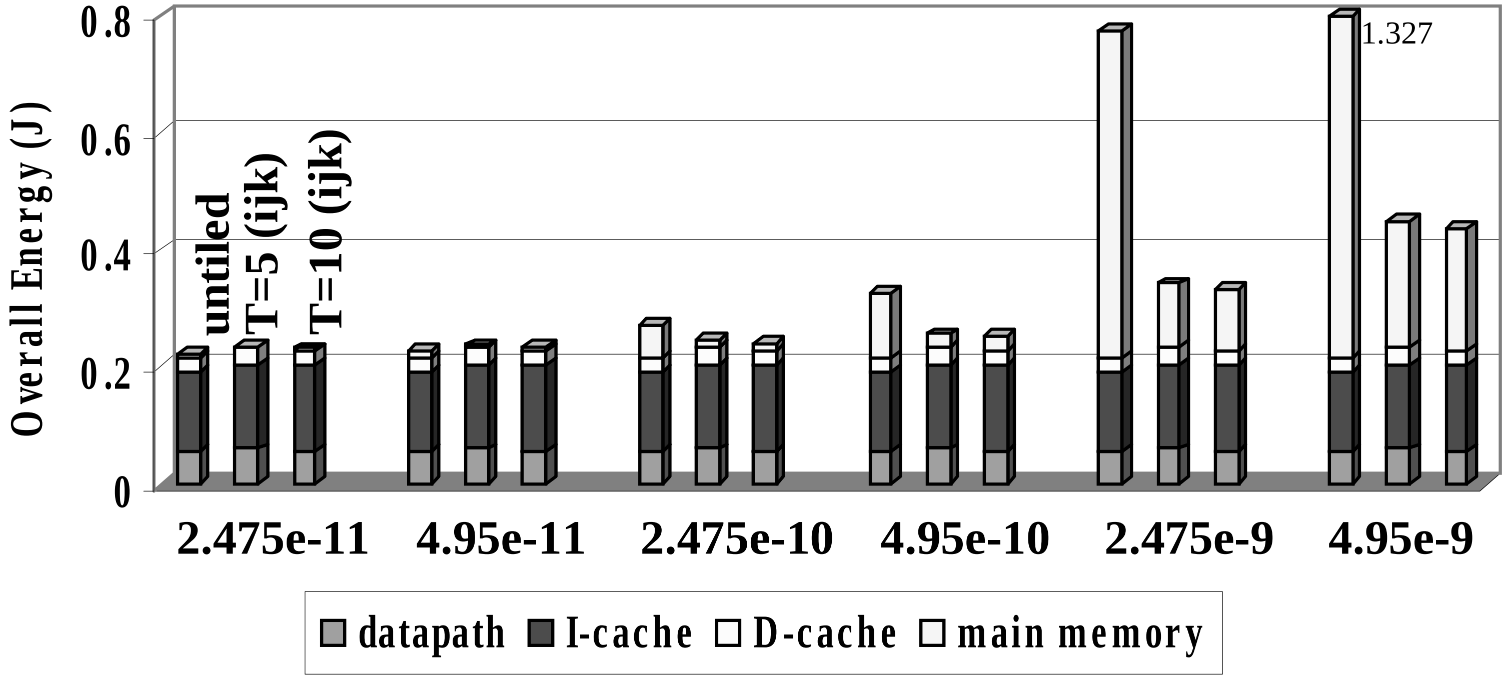
Interaction of Compiler and Hardware Optimizations

- Block buffering saved 19% more energy when using compiler optimizations
  - Block buffer hit rates improve
  - Loop permutations had maximum impact
- MRU caches saved 21% more energy when using compiler optimizations
  - Way prediction was more successful

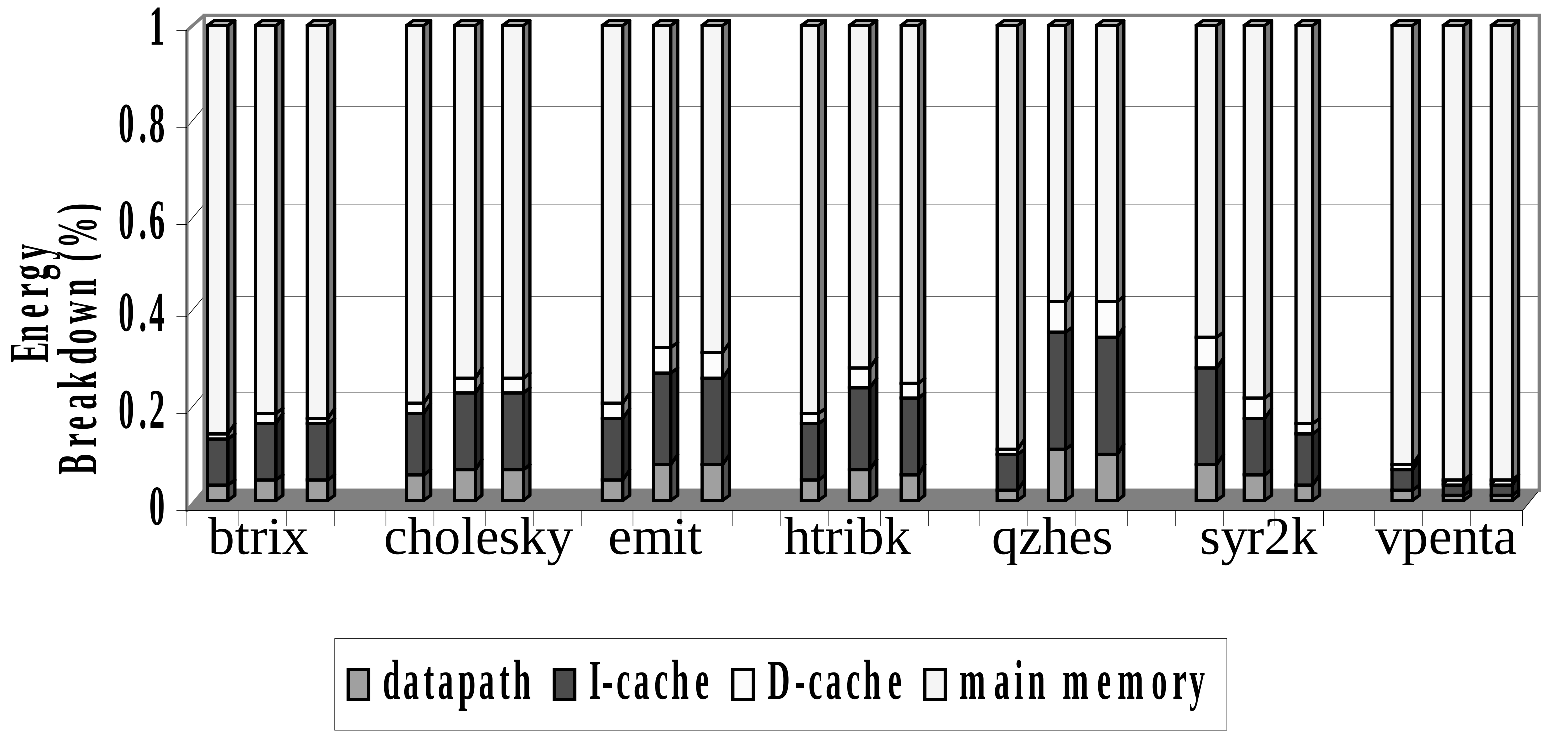
Reducing Memory Energy

- Improve the locality of memory accesses
  - Minimize the number of memory accesses
- Architectural and circuit techniques
- Optimizing interactions of compiler and cache architecture
- Technology changes and low-power operating modes

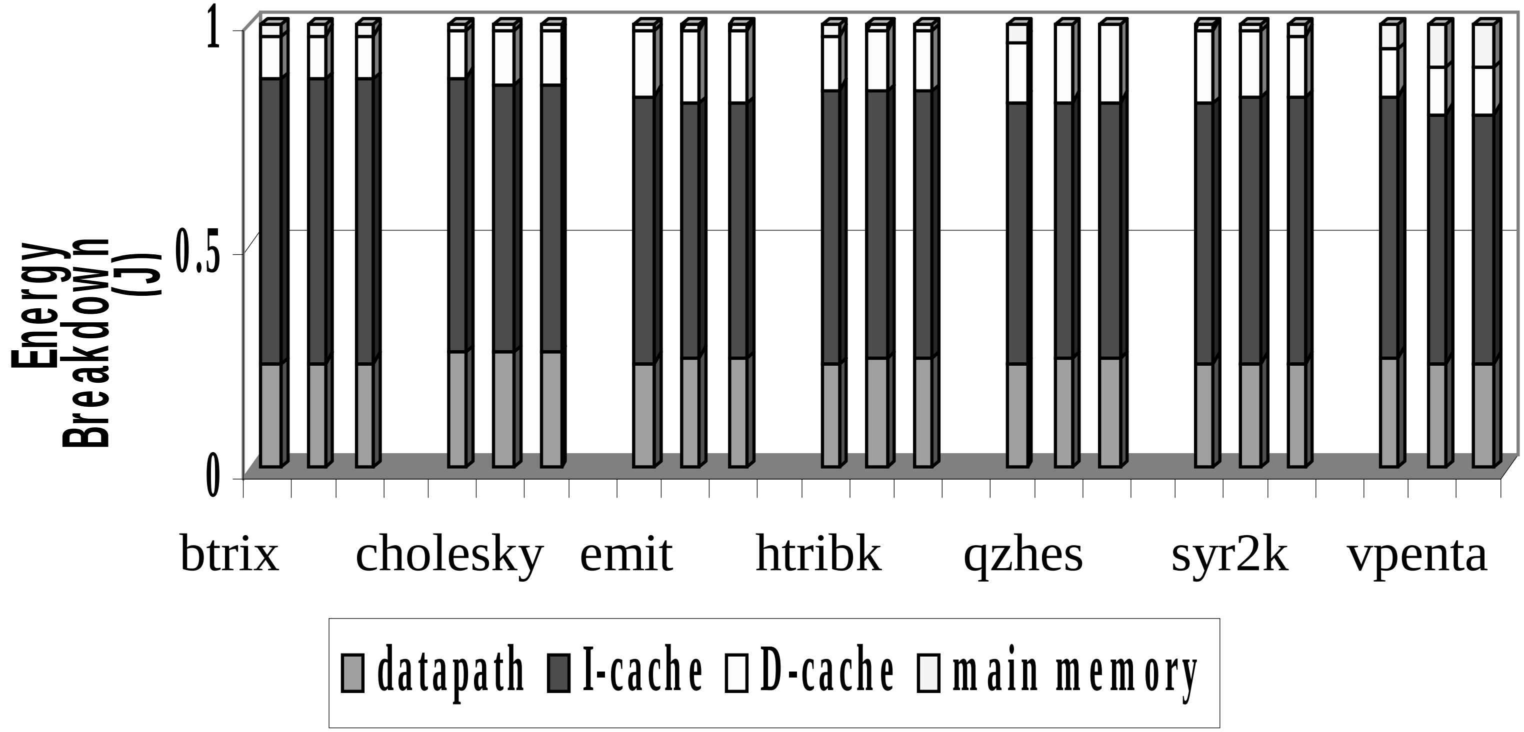
Sensitivity to Technology Changes



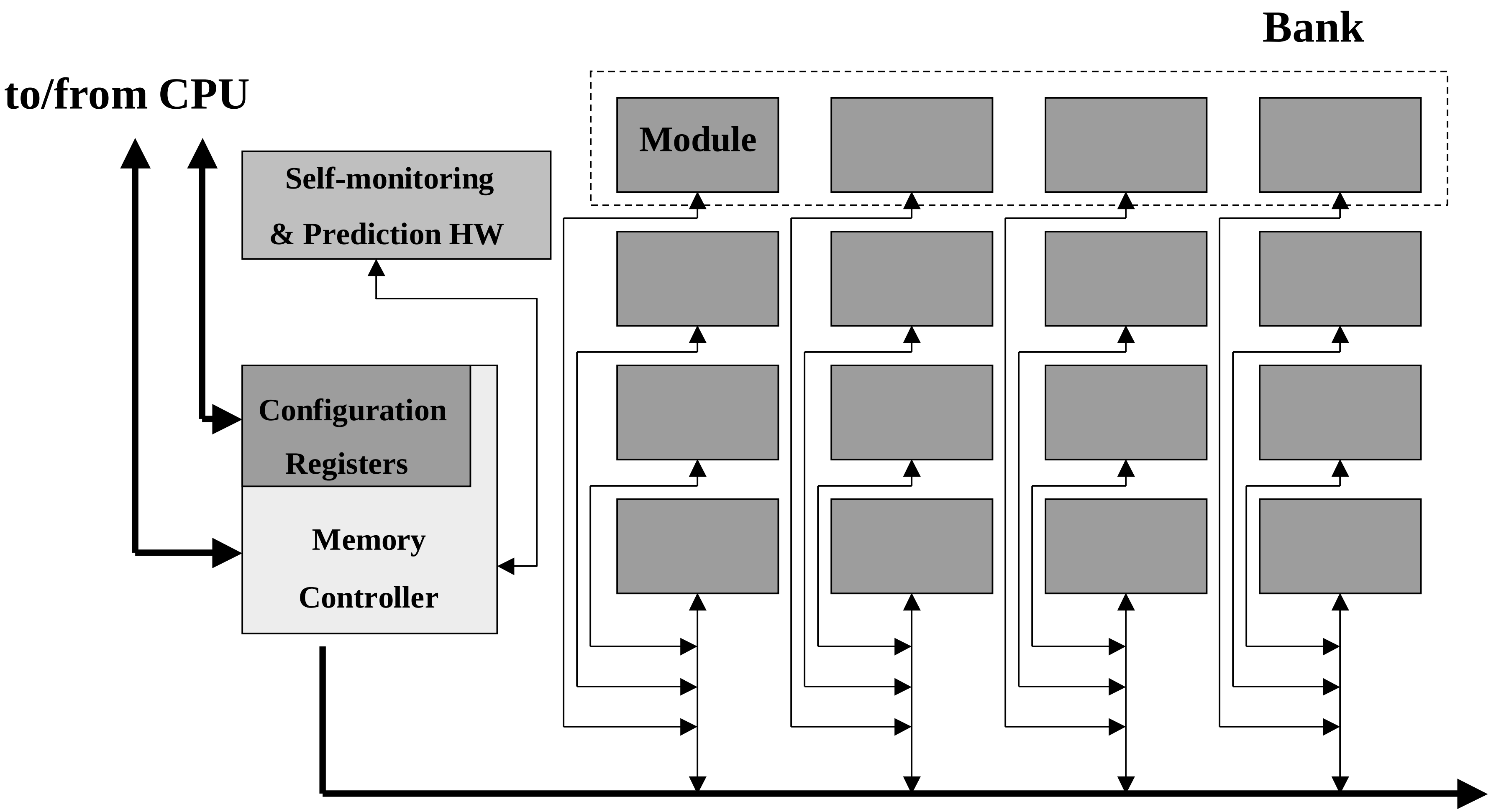
Technology Factor (Em=4.95e-9)



Technology Factor (Em=2.475e-11)



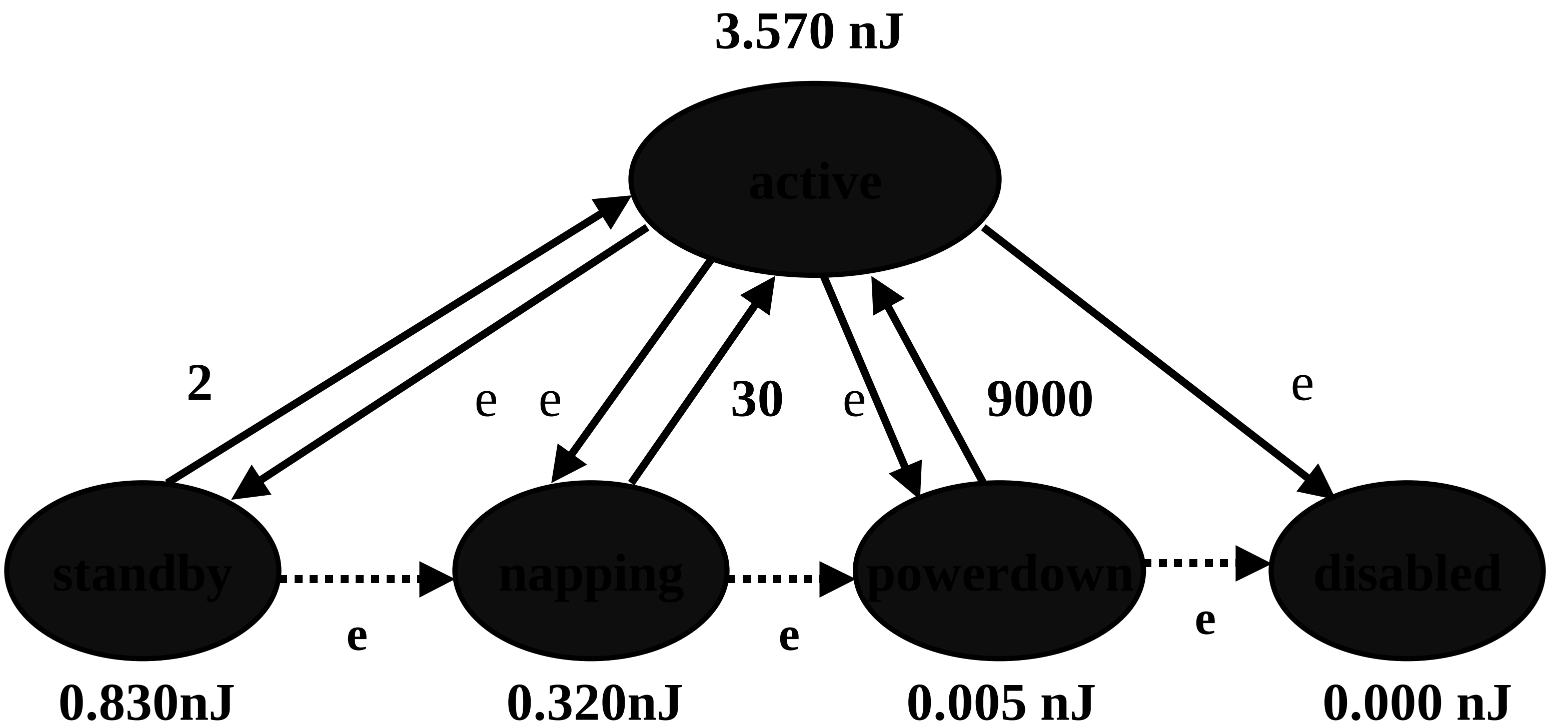
Partitioned Memory Architecture



Alternatives

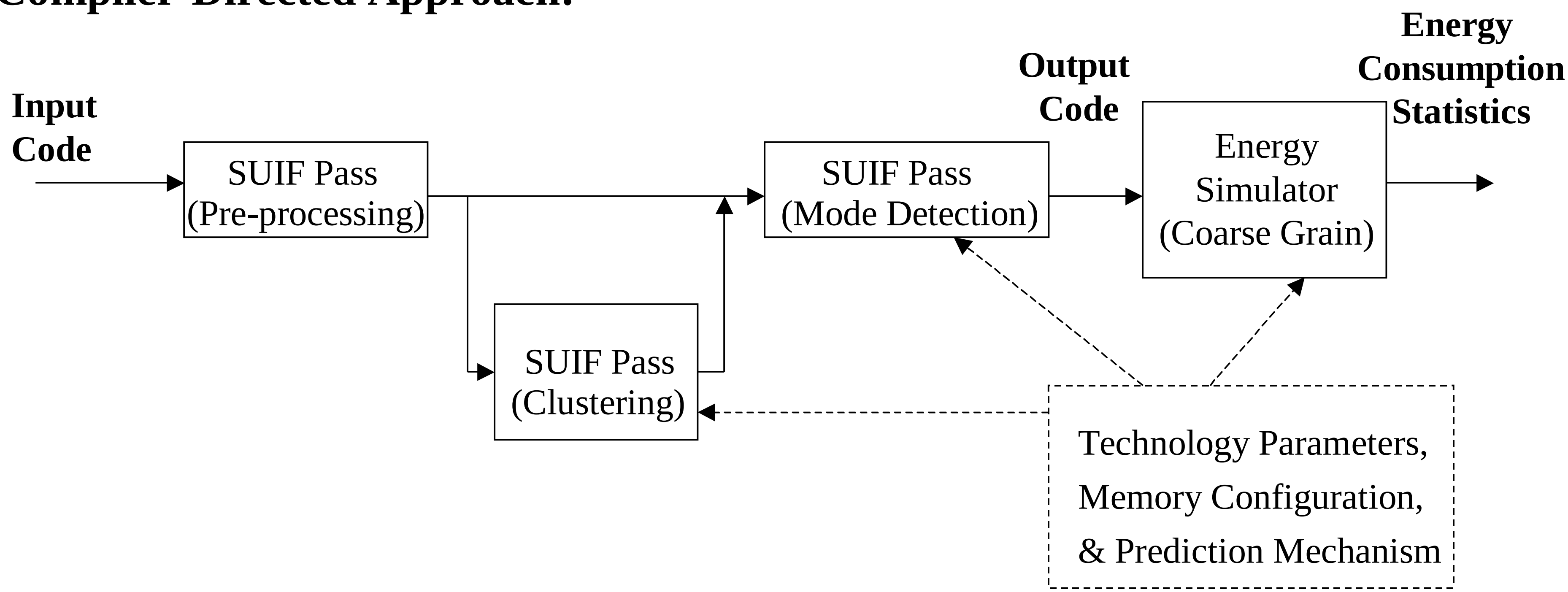
- All memory modules are ON all the time
- Mode Control Only
  - If not used, reduce power
  - No data/access pattern modifications
- Mode Control + Optimizations
  - Data Transformations (e.g., Clustering)
  - Loop Optimizations (e.g., Loop Splitting)

Power Modes


3.570 nJ
active
2
e
e
30
e
9000
e
standby
napping
powerdown
disabled
e
e
e
0.830nJ
0.320nJ
0.005 nJ
0.000 nJ

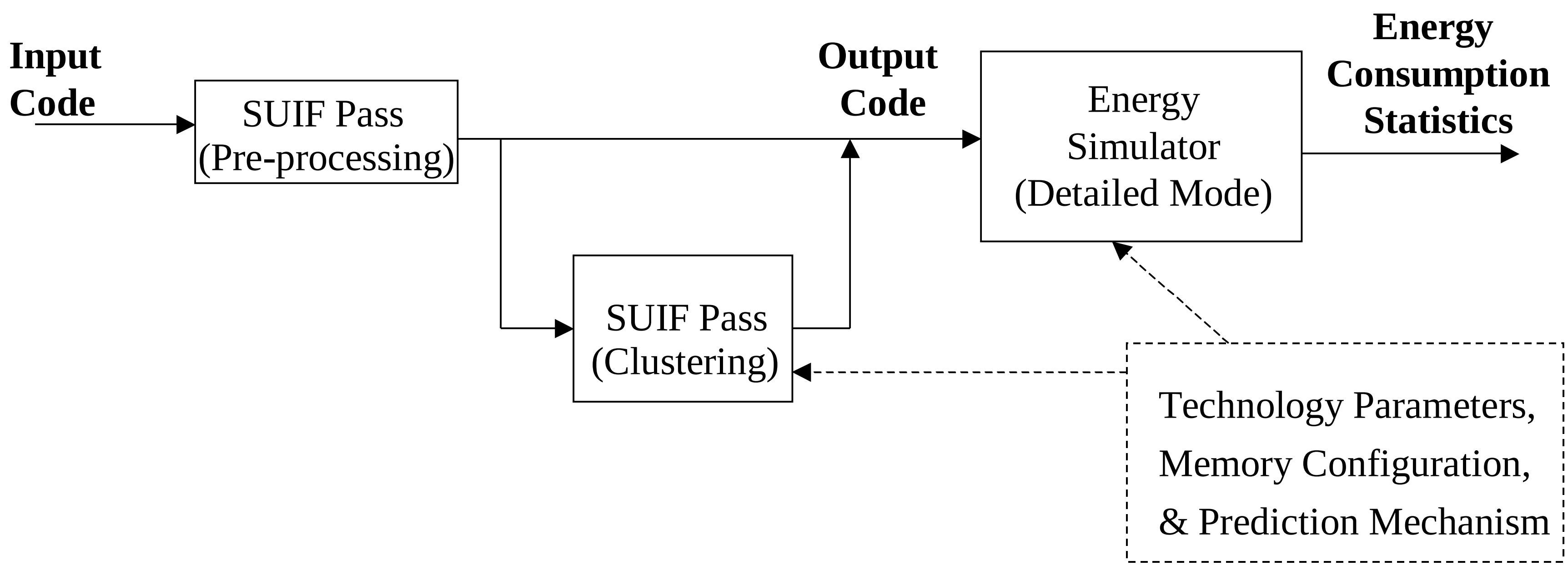
Experimental Setup

Compiler-Directed Approach:



Experimental Setup

Self-Monitored Approach:



Array Clustering Heuristics

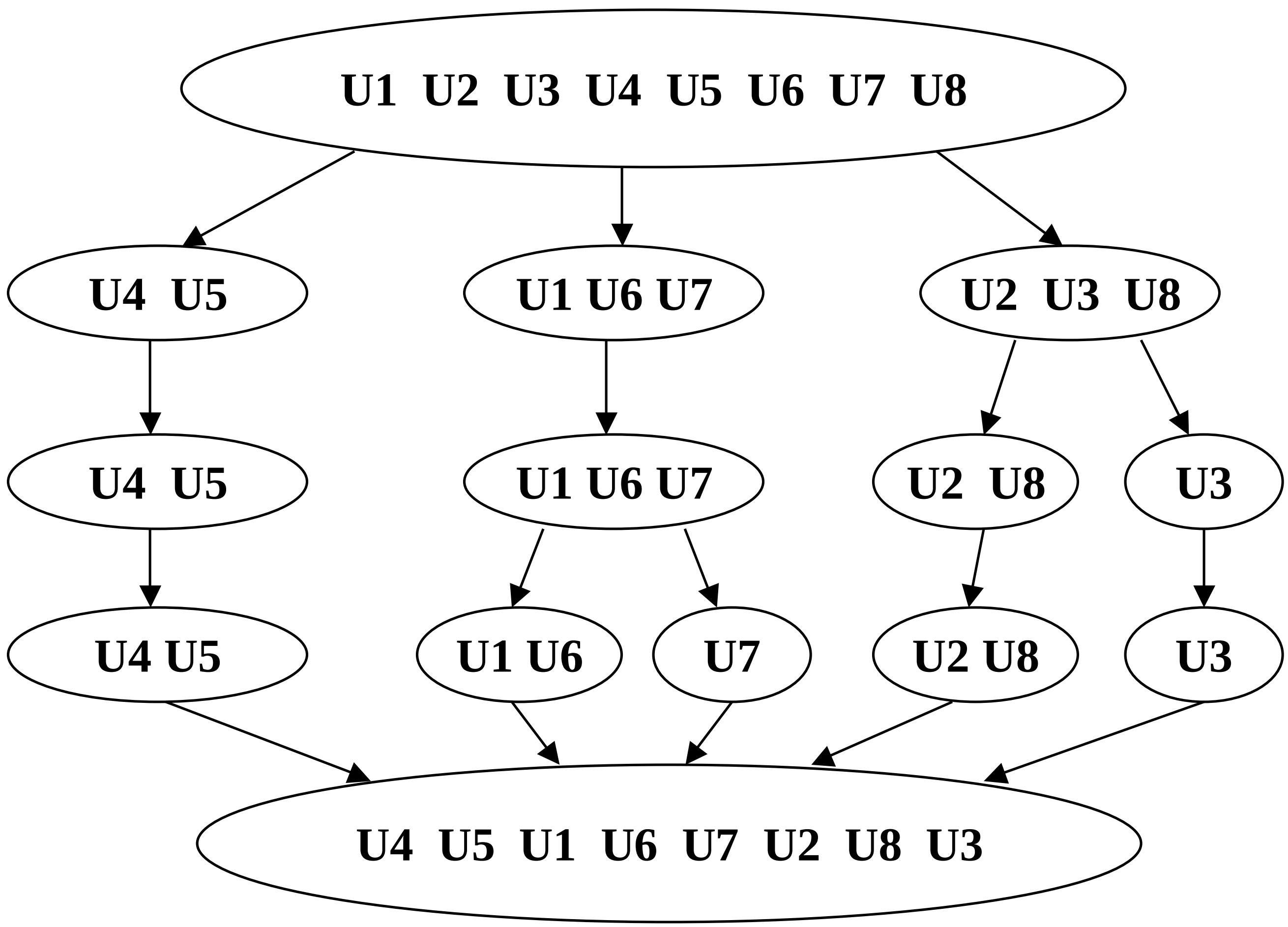
- Profile-Based
- Static Analysis-Based
  - Constructive Algorithm (Graph-Based)
  - Iterative Algorithm

Module/Bank Configuration is Important!

Array Access Profile (*vpenta*)

Phase Number	Array Variables							
	U1	**U2**	**U3**	**U4**	**U5**	**U6**	**U7**	**U8**
1	**X**			**X**	**X**	**X**	**X**	
2		**X**		**X**	**X**		**X**	**X**
3		**X**		**X**	**X**		**X**	**X**
4	**X**	**X**	**X**	**X**	**X**	**X**	**X**	**X**
5	**X**	**X**	**X**			**X**	**X**	**X**
6	**X**	**X**	**X**				**X**	**X**
7		**X**	**X**					**X**
8		**X**	**X**					**X**

Iterative Algorithm (*vpenta*)



Bank Access Profile (*vpenta*) (unoptimized)

Phase Number	**Memory Banks**							
	B0	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**
1	**X**			**X**	**X**			
2	**X**			**X**	**X**	**X**		
3	**X**			**X**	**X**	**X**		
4	**X**	**X**	**X**	**X**	**X**	**X**		
5	**X**	**X**	**X**	**X**	**X**	**X**		
6	**X**	**X**	**X**	**X**	**X**	**X**		
7	**X**	**X**	**X**	**X**		**X**		
8	**X**	**X**	**X**			**X**		

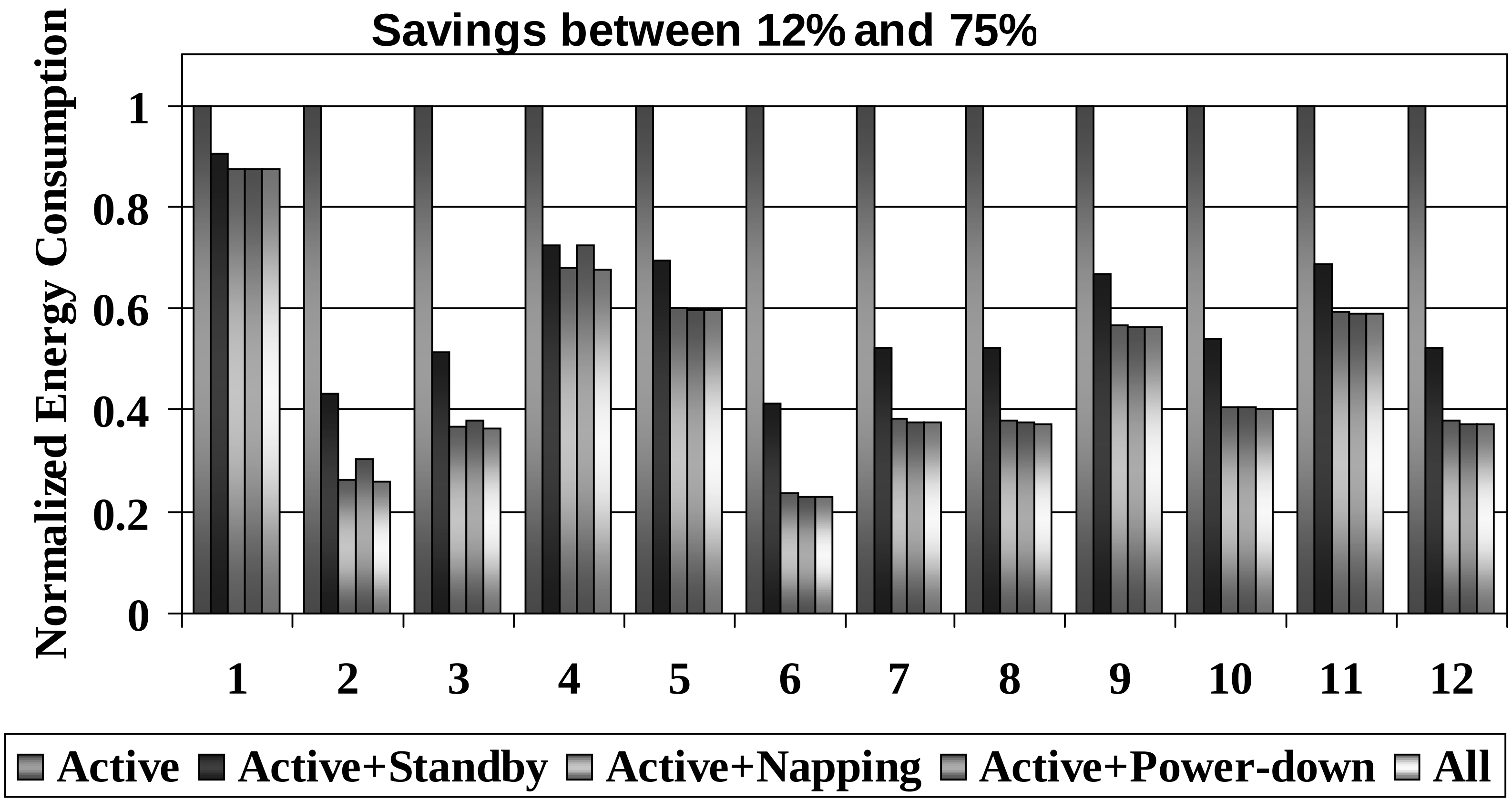
Bank Access Profile (*vpenta*) (optimized)

Phase Number	**Memory Banks**							
	B0	**B1**	**B2**	**B3**	**B4**	**B5**	**B6**	**B7**
1	**X**	**X**						
2	**X**		**X**	**X**				
3	**X**		**X**	**X**				
4	**X**	**X**	**X**	**X**				
5		**X**	**X**					
6		**X**	**X**					
7			**X**	**X**	**X**	**X**		
8			**X**	**X**	**X**	**X**		

Benchmark Codes

Benchmark Number	Benchmark Name	Data Size (MB)	Base Energy (mJ)
1	adi	48.0	3.38
2	dtdtz	61.8	2.55
3	bmcm	39.9	3.93
4	btrix	47.7	2.49
5	eflux	33.6	413.23
6	full_search	33.0	337.75
7	matvec	16.0	675.75
8	mxm	48.0	10.70
9	phods	33.0	1586.25
10	tomcatv	56.0	119.80
11	vpenta	44.0	506.68
12	amhmtm	48.1	7.40

Energy Savings
(Mode Control only)



Energy Savings
(Mode Control + Clustering)