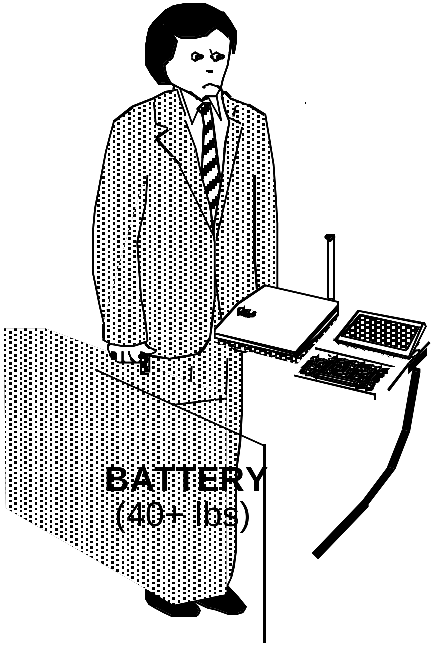
# Outline

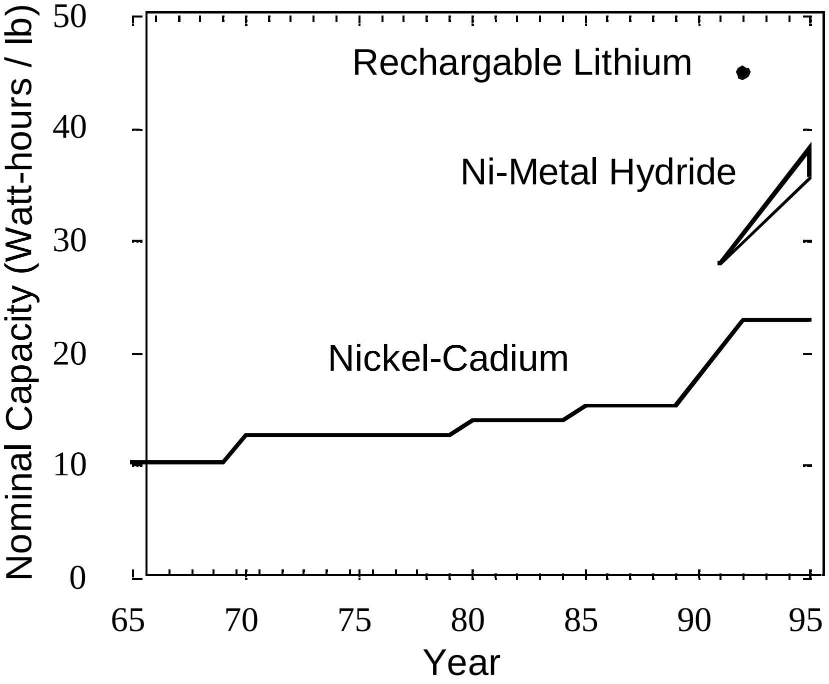
- Why Low-Power?
- Experimental Framework
- Estimating Energy Consumption
- Improving Energy Consumption
  - Hardware Optimizations
  - Software Optimizations
  - Combined Techniques
- Energy Behavior of Java Codes
- Conclusions
- Ongoing Work at Penn State
- References

# Why Low Power?

- Power consumption/dissipation in Watts
  - sets packaging limits
  - determines power ground wiring designs
  - impacts signal noise margin and reliability analysis
- Energy consumption in Joules
  - energy = power * delay (joules = watts * seconds)
  - lower energy number means less power to perform a computation at the same frequency
  - determines battery life in hours

# Why worry about power ?
# -- Battery Size/Weight





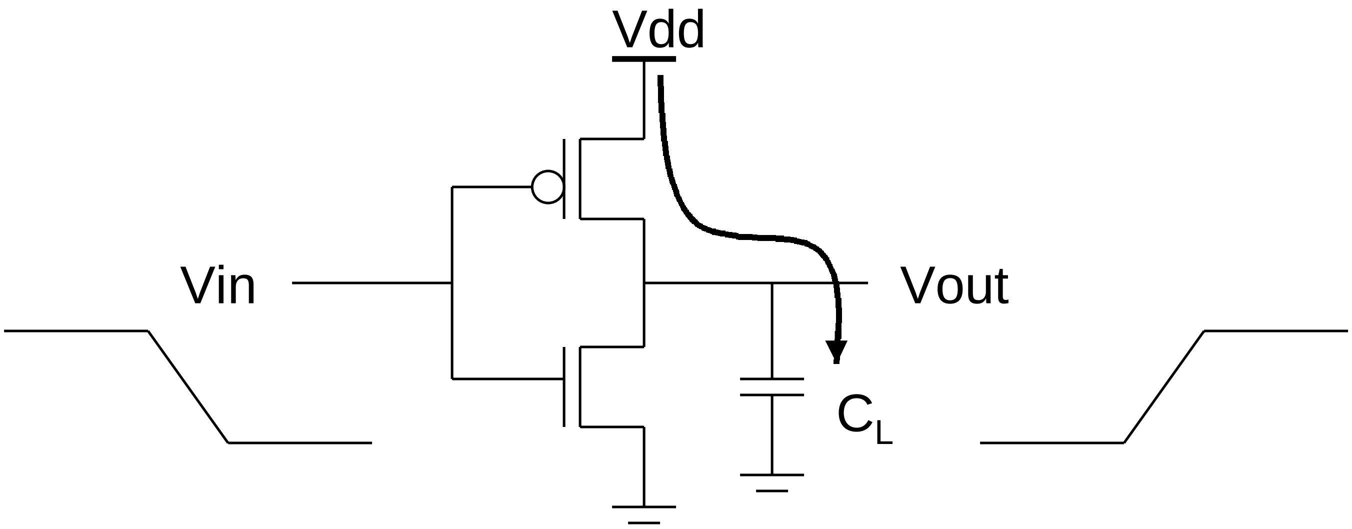
Expected battery lifetime increase
over the next 5 years: 30 to 40%

**From Rabaey, 1995**

# Where Does Power Go in CMOS?

- Dynamic Power Consumption
  - charging and discharging capacitors
- Short Circuit Currents
  - short circuit path between supply rails during switching
- Leakage Current (currently ~2%)
  - leaking diodes and transistors

# Dynamic Power Consumption

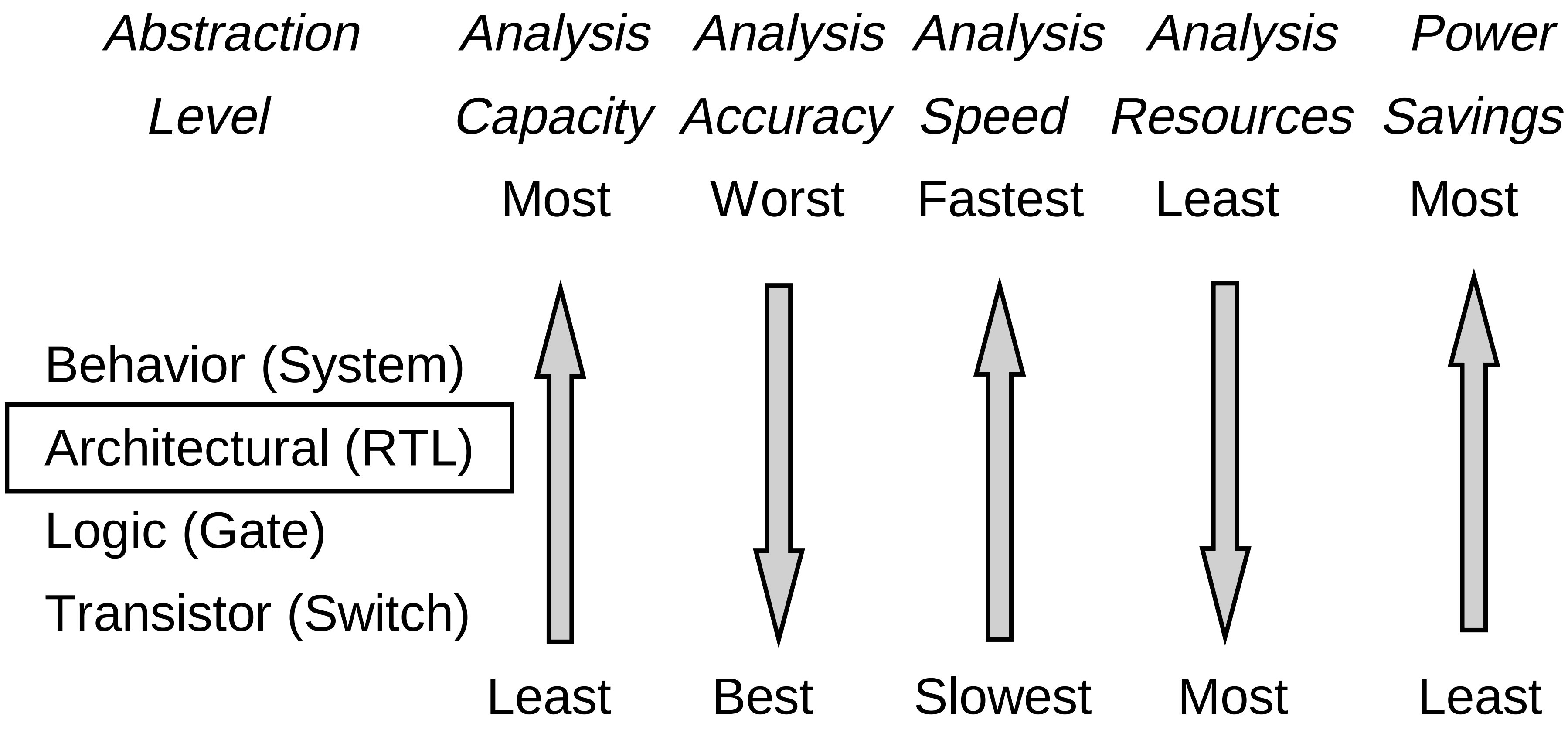


$$\text{Energy/transition} = C_L * V_{DD}^2 * P_{0\to1}$$

$$\text{Power} = \text{Energy/transition} * f = C_L * V_{DD}^2 * \overbrace{P_{0\to1} * f}^{f_{0\to1}}$$

Data dependent - a function of switching activity!

# Motivation

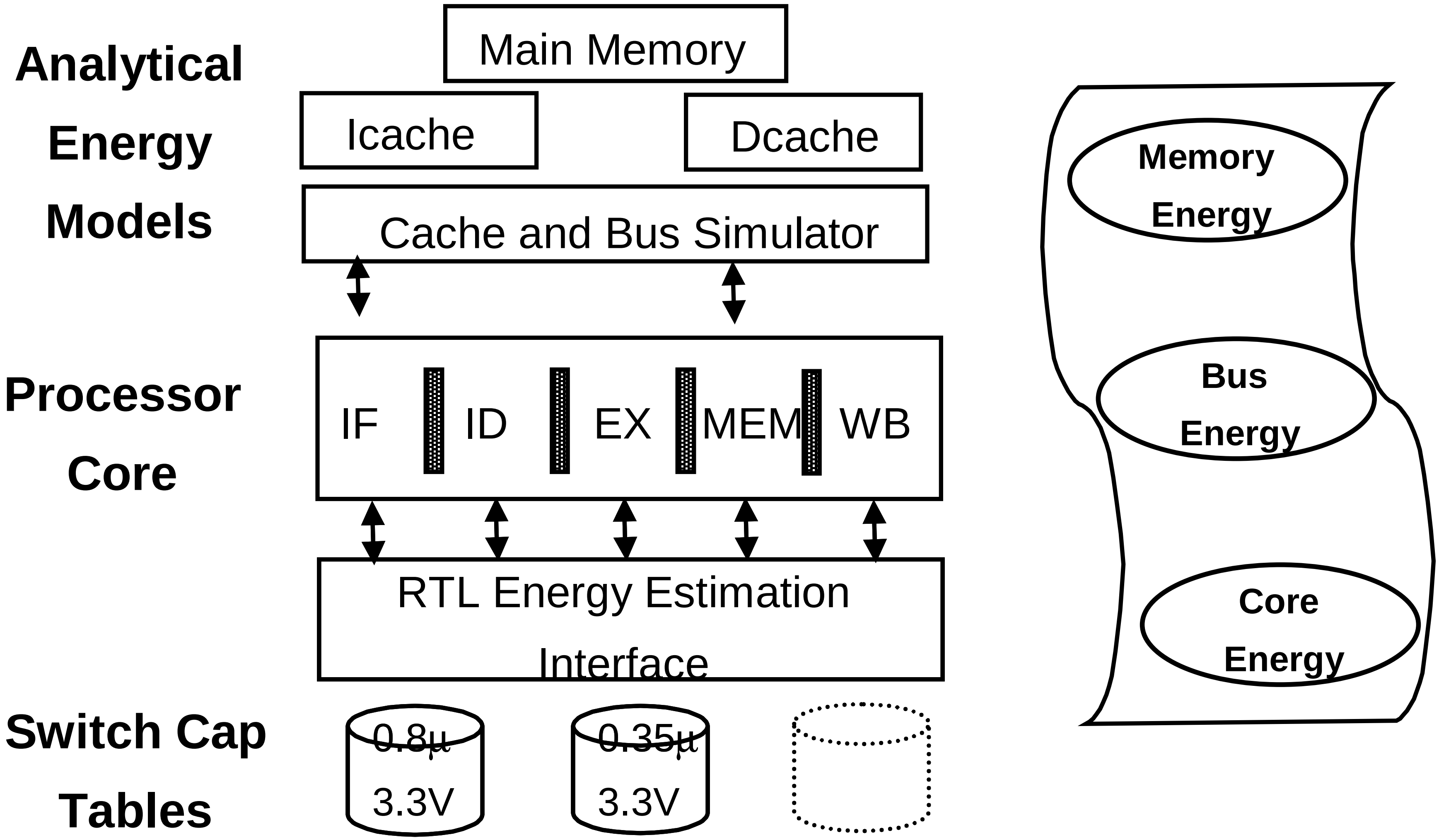


| *Abstraction Level* | *Analysis Capacity* | *Analysis Accuracy* | *Analysis Speed* | *Analysis Resources* | *Power Savings* |
|---|---|---|---|---|---|
| | Most | Worst | Fastest | Least | Most |
| Behavior (System)<br>Architectural (RTL)<br>Logic (Gate)<br>Transistor (Switch) | ↑ | ↓ | ↑ | ↓ | ↑ |
| | Least | Best | Slowest | Most | Least |

# Architectural Level Analysis Considerations

- Computationally efficient
  - requires predefined analytical and transition-sensitive energy characterization models
  - design only to RTL (with some idea as to the kind of functional units planned)
- Simulation based so can be used to support architectural, compiler, operating system, and application level experimentation
- WattWatcher (Sequence), ICPower (Solution Ware), PowerCompiler (Synopsys), academic tools (Wattch - Princeton, Avalanche - Princeton/NEC, Polis/Ptolemy - Italy/NEC, SimplePower - PSU)

# SimplePower Framework



Analytical Energy Models

Main Memory

Icache

Dcache

Cache and Bus Simulator

Processor Core

IF

ID

EX

MEM

WB

RTL Energy Estimation Interface

Switch Cap Tables

0.8µ 3.3V

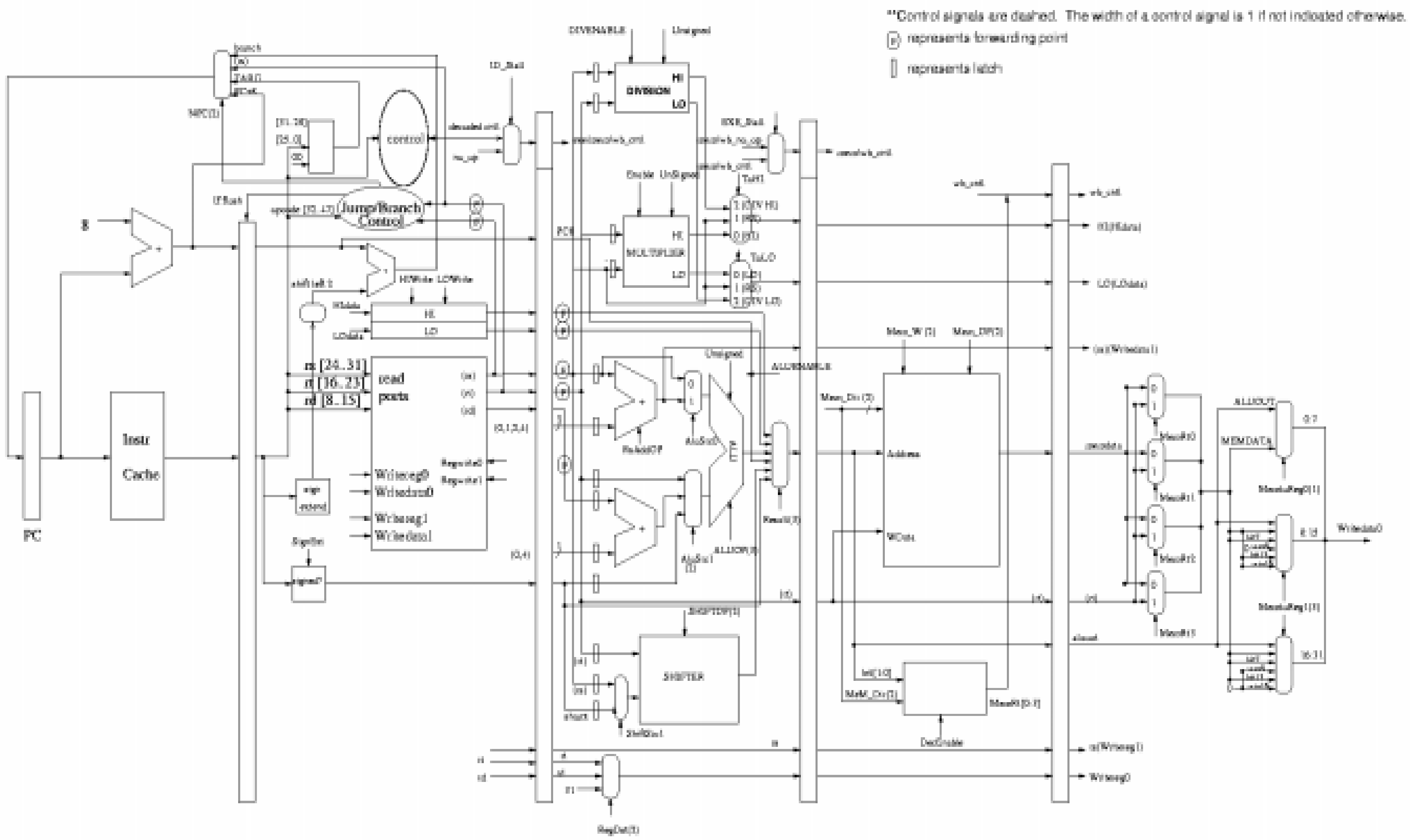
0.35µ 3.3V

Memory Energy

Bus Energy

Core Energy

# Architecture



**Control signals are dashed. The width of a control signal is 1 if not indicated otherwise.

(F) represents forwarding point

[] represents latch

# Energy Characterization

- Transition-sensitive energy models
  - – single energy tables: e.g., Adders
  - – multiple energy tables: e.g., Register files
  - – system level interconnect

- Analytical energy models
  - – cache and main memory

# Transition-Sensitive Energy Model

- Must first design and layout a functional unit and then simulate it to capture switch capacitances
  - Bit-independent – bus lines, pipeline registers
    - one bit switching does not affect other bit slices’ operations
  - Bit-dependent – ALU, MAC, decoders
    - one bit switching does affect other bit slices’ operations

- Once constructed, the models can be reused in simulations of other architectures built with the same technology

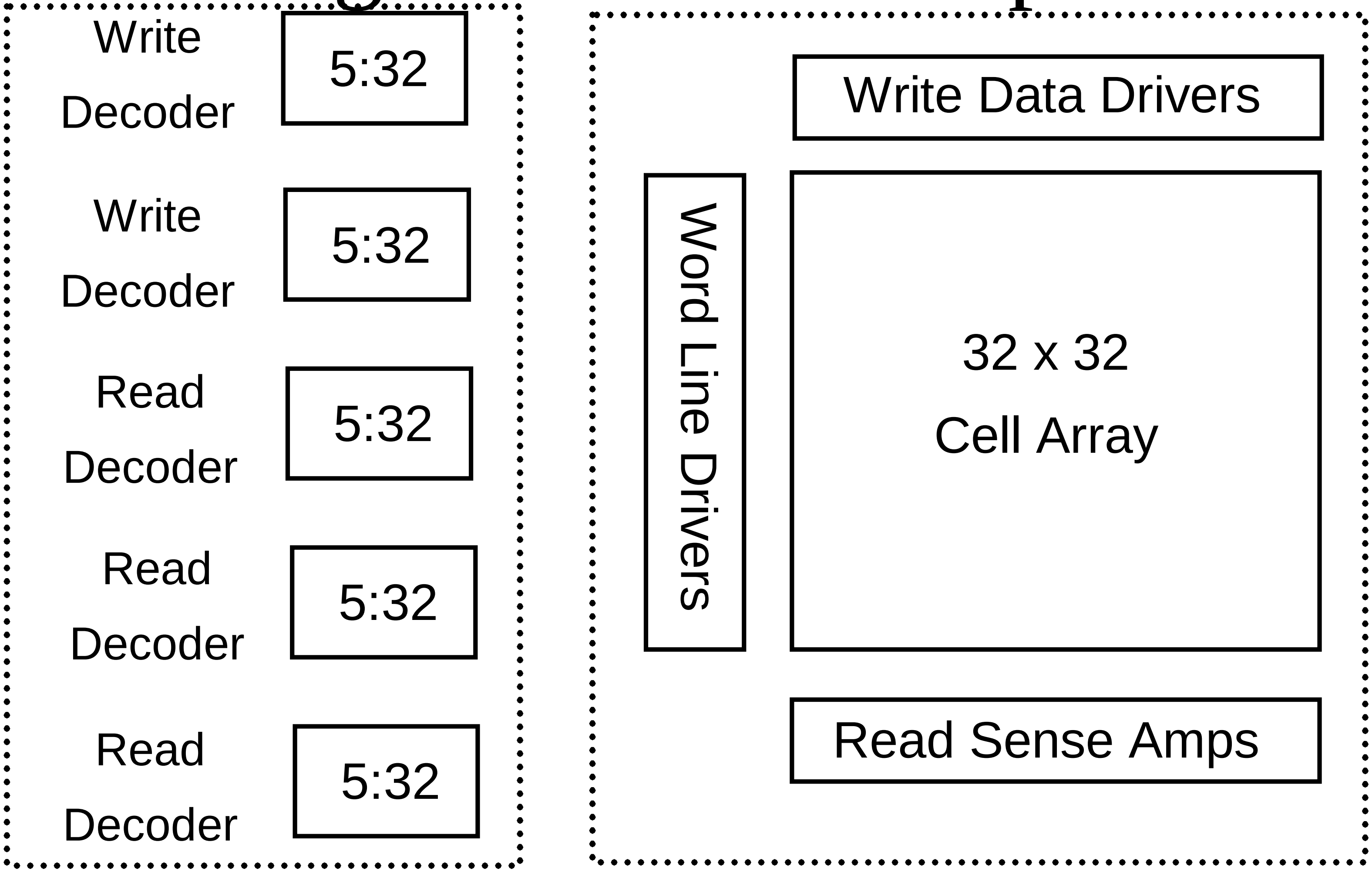
# Switch Capacitance Table

| **Previous Input Vector** | **Current Input Vector** | **Switch Capacitance** |
|---|---|---|
| 0 . . . 0 0 | 0 . . . 0 0 | $\text{cap}_{0 \rightarrow 0}$ |
| 0 . . . 0 0 | 0 . . . 0 1 | $\text{cap}_{0 \rightarrow 1}$ |
| | | |
| 1 . . . 1 1 | 1 . . . 1 1 | $\text{cap}_{2^n-1 \rightarrow 2^n-1}$ |

# Table Construction Issues

- Table can get huge to construct and store - $2^{2n}$ entries
- Existing clustering solutions reduce number of entries but increase error margin
- Hierarchical partitioning techniques
  - – Much smaller tables
  - – Reuse of same table for different instances of the component

# Register File Example



Write Decoder 5:32

Write Decoder 5:32

Read Decoder 5:32

Read Decoder 5:32

Read Decoder 5:32

Write Data Drivers

Word Line Drivers

32 x 32

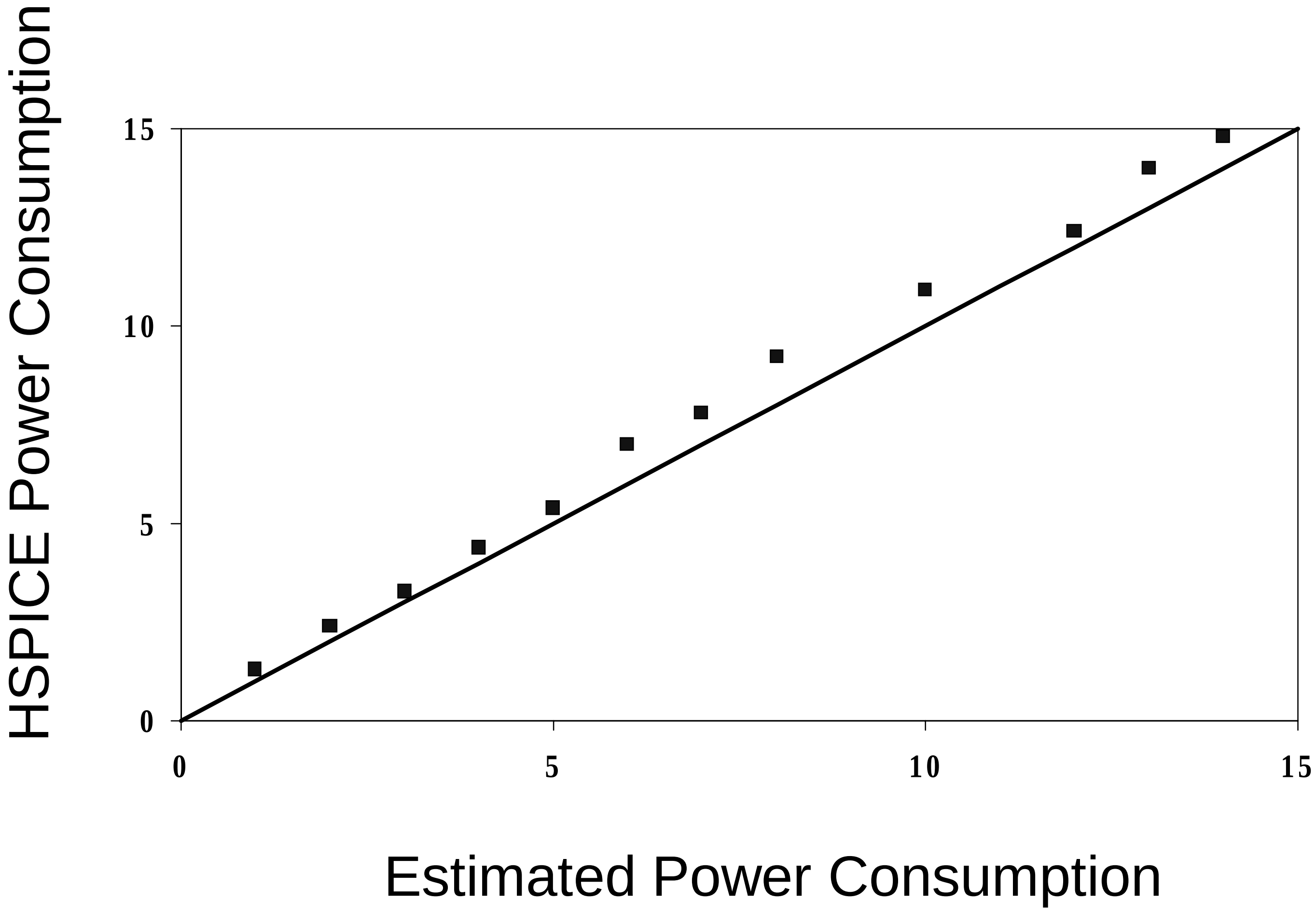
Cell Array

Read Sense Amps

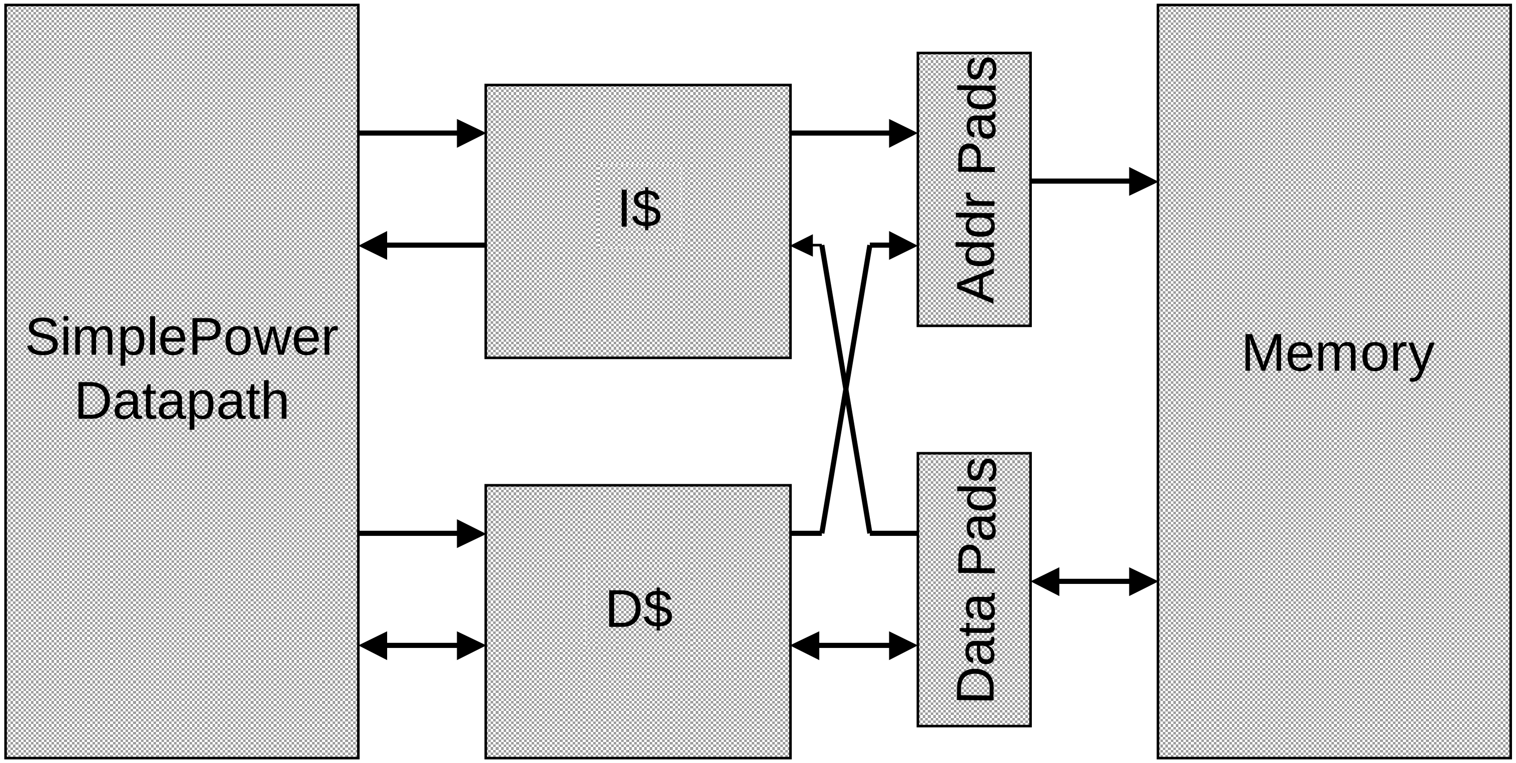
Bit Dependent Cells

Bit Independent Cells

# Validation of Register File Energy Model



# System Model



# Memory System Energy Model

- Energy dissipated by bitlines: precharge, read and write cycles
- Energy dissipated by wordlines: when a particular row is being read or written
- Energy dissipated by address decoders
- Energy dissipated by peripheral circuits - comparators, cache control logic etc.
- Off-chip main memory energy is based on per-access cost

# Analytical Energy Model Example

- On-chip cache

  Energy = Ebus + Ecell + Epad

  …

  Ecell = $\beta$*(wl_length)*(bl_length+4.8)*(Nhit + 2*Nmiss)

  wl_length = m*(T + 8L + St)

  bl_length = C/(m*L)

  Nhit = number of hits; Nmiss = number of misses;

  C=cache size; L = cache line size in bytes;

  m = set associativity; T = tag size in bits;

  St = # of status bits per line;

  $\beta$ = 1.44e-14 (technology parameter)

# Energy Optimization: Tools and Techniques

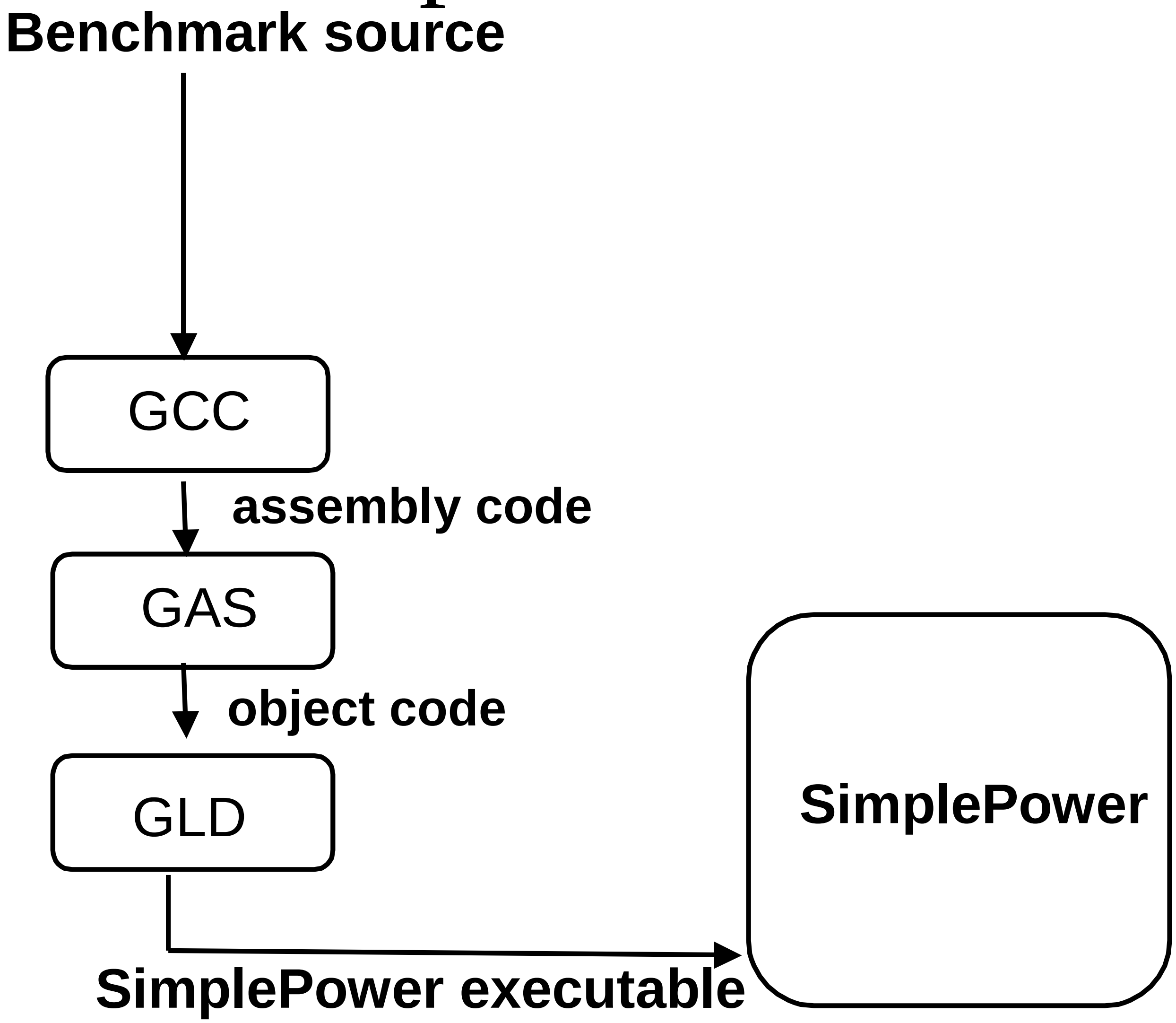
Mahmut Taylan Kandemir
Department of CSE, Microsystems Design Lab
Penn State University (www.cse.psu.edu/~mdl)

# Model Validation and Performance

- HSPICE validation for each of the modules
- 0.1ns for each input transition compared to ~9 minutes for analysis using HSPICE - register file

- Validated overall approach against current measurements for a commercial Merged DSP Processor - within 8.9% average error

# Compiler Framework



**Benchmark source**

GCC

**assembly code**

GAS

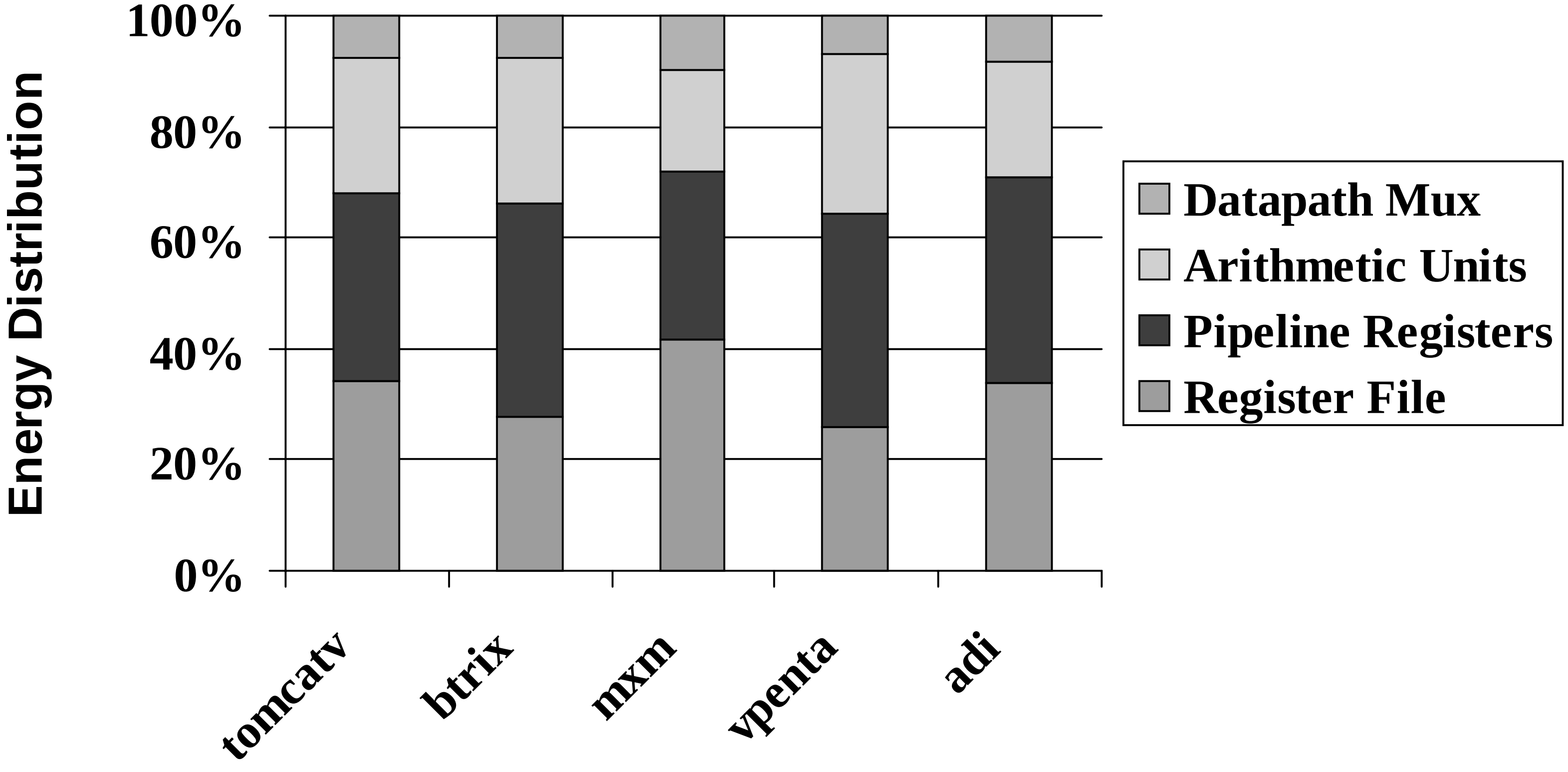
**object code**

GLD

**SimplePower executable**

**SimplePower**

# Datapath Energy Distribution



# Pipeline Registers

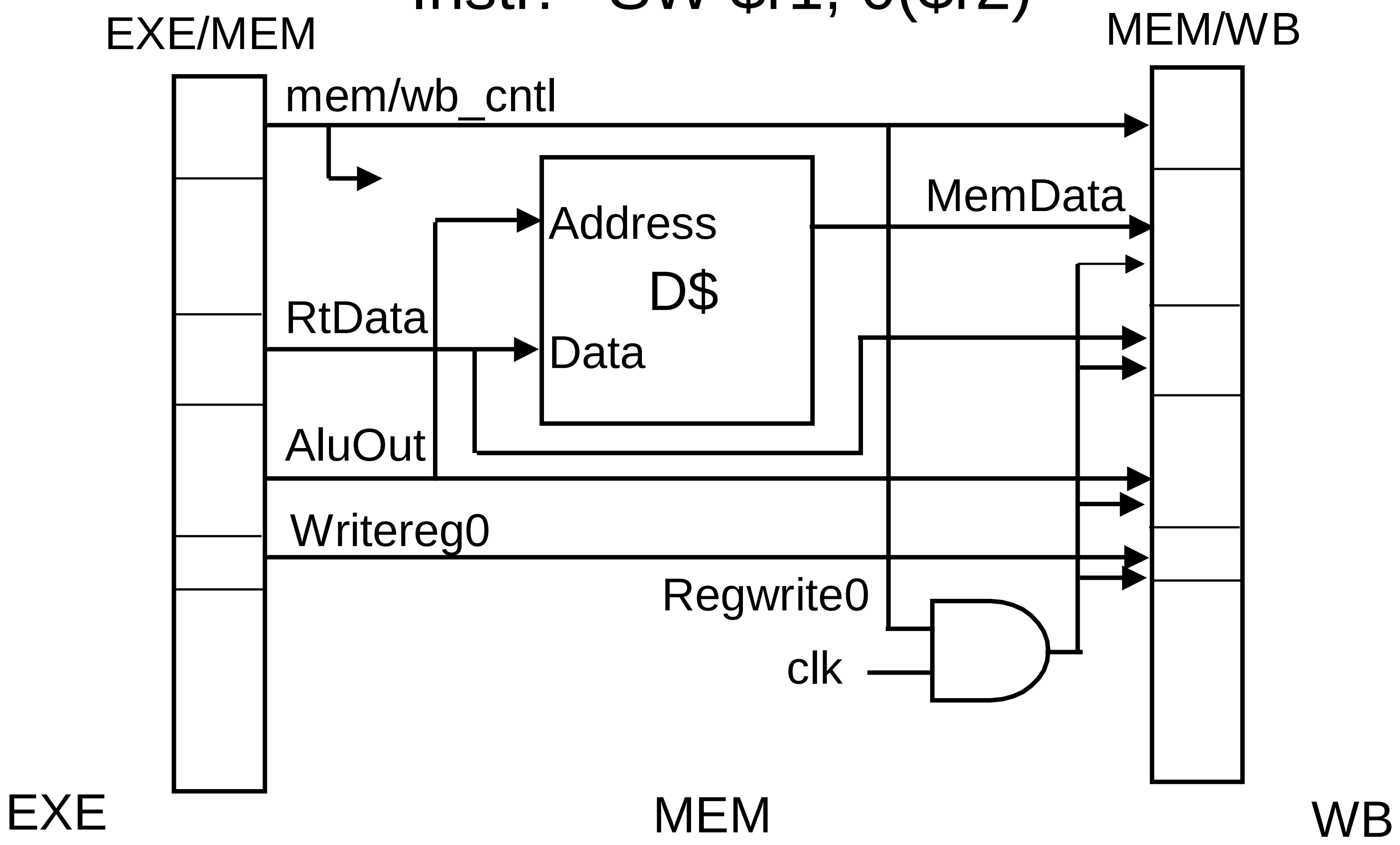
- Consume a lot of energy because they are clocked every cycle
  - Clock energy ($E_c$)
    - energy dissipated when the register is clocked with stable data
  - Data energy ($E_d$)
    - energy dissipated when the register is clocked and the data has changed so that the register changes state
  - Typically the data rate ($f_d$) is much lower than the clock rate ($f_c$)
- Also impacts clock energy since a large portion of clock energy is used to drive the sequential elements

# Pipeline Registers

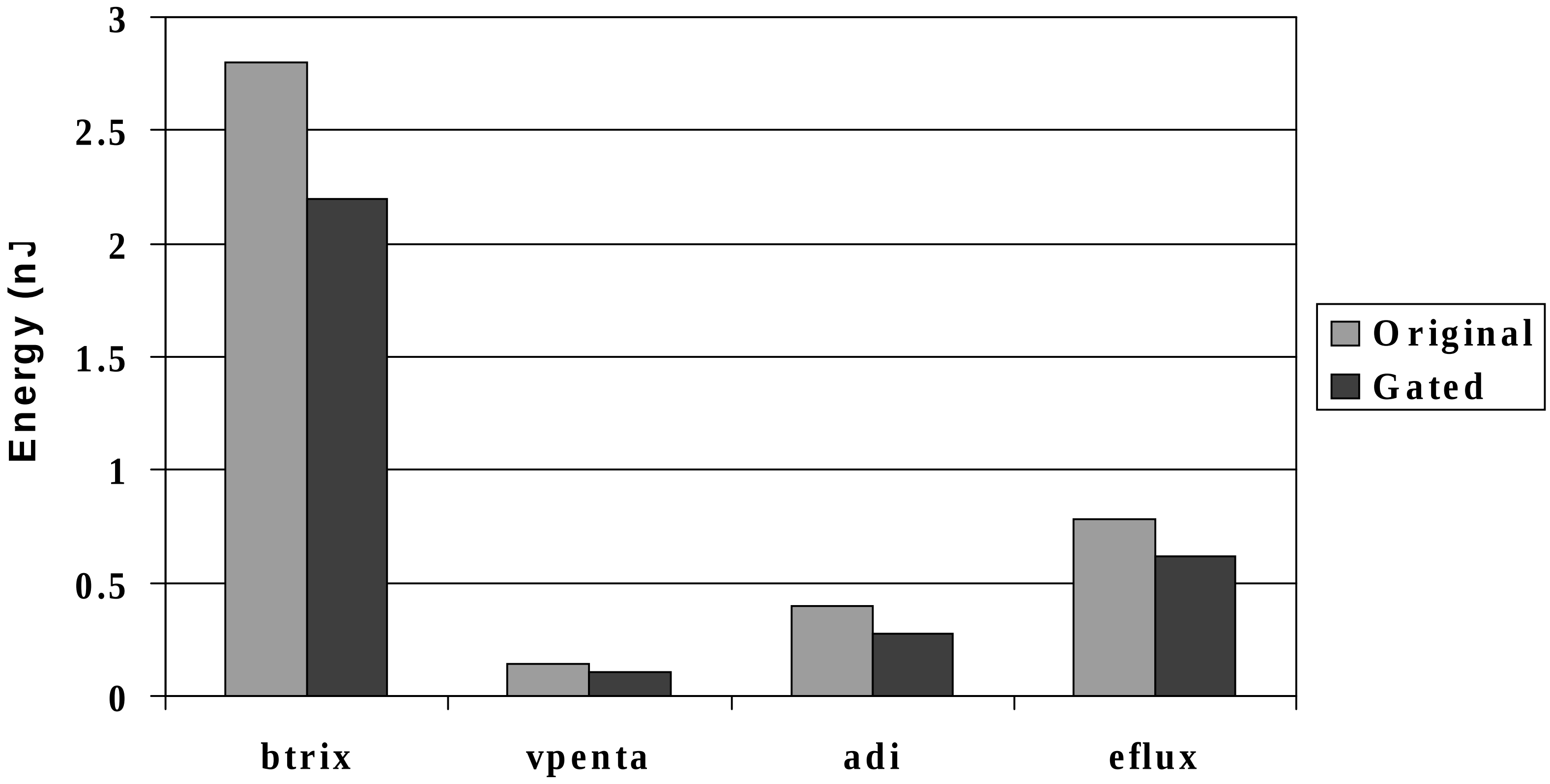
- Pipeline registers consume a large percentage of datapath power
  - 40% for 0.35μ
- Pipeline registers have large width
- Pipeline registers are clocked every cycle
- Not all clockings are necessary
  - use the decoded control signals to selectively gate the clock of pipeline register fields
  - only simple extra logic necessary
  - can be built into the clock buffer circuit

# Gated Pipeline Registers

Instr: SW $r1, 0($r2)



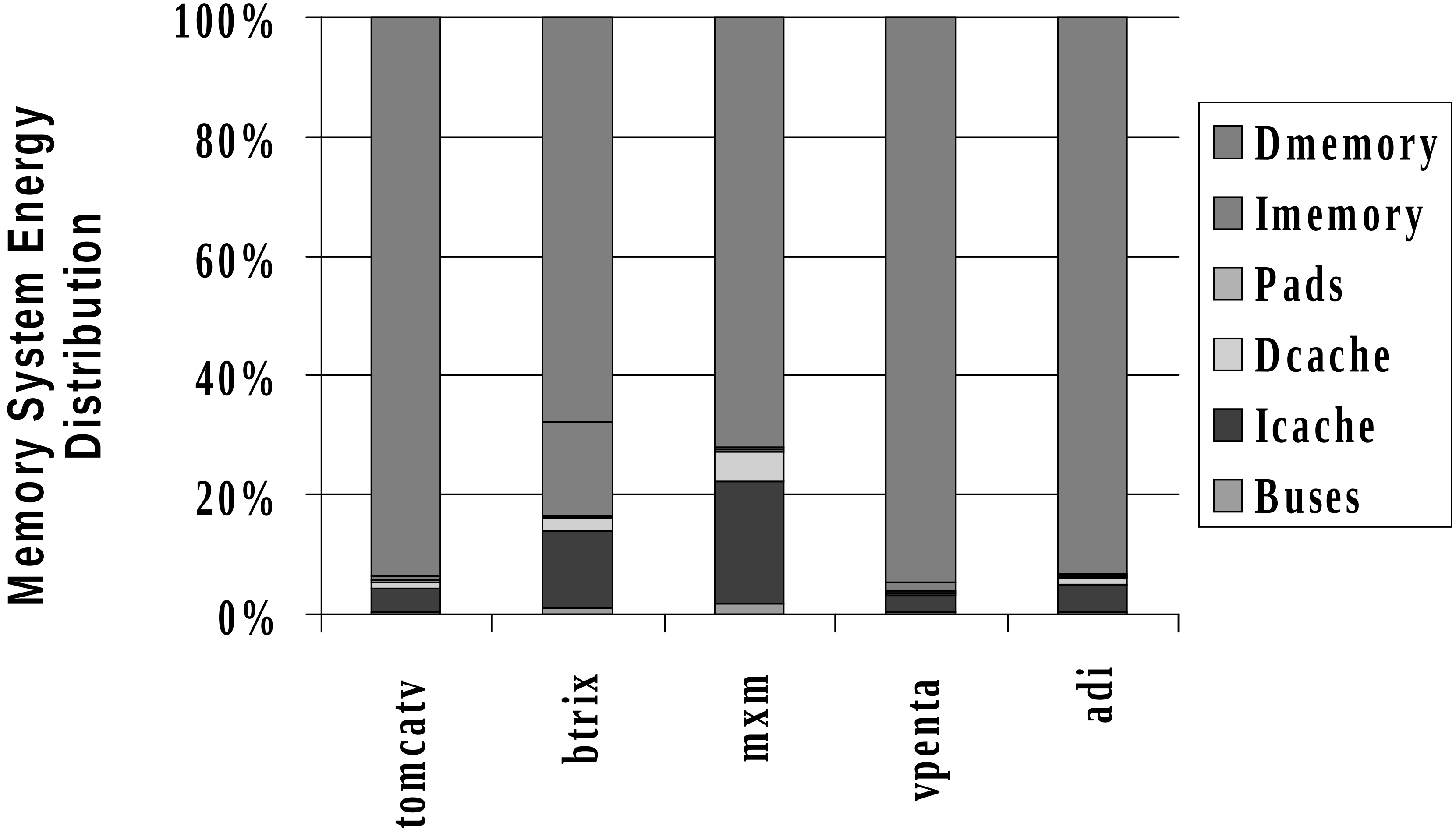
# Influence of Selective Gating (Datapath Energy)



# Other Datapath Optimizations

- Choice of functional unit
  - Trade speed for energy
  - Number of pipeline stages
- Influence of instruction scheduling
  - Energy vs. performance trade-offs [WVLSI’00]
  - Register re-labeling optimization [DAC’00]

# Memory Energy Distribution

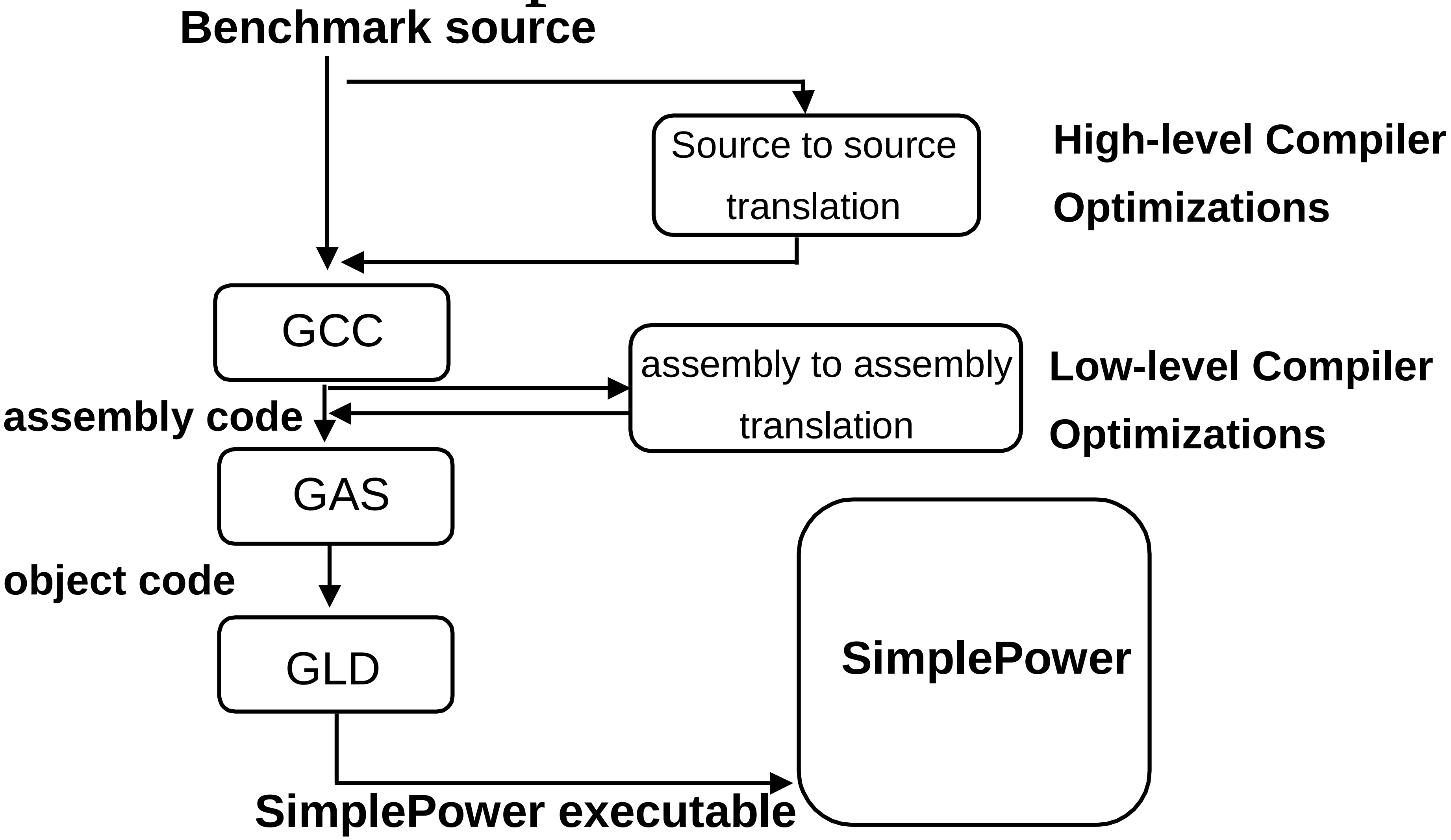


8K 4way Dcache, 8K 1way Icache

# 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

# Compiler Framework



Benchmark source
Source to source translation
High-level Compiler Optimizations
GCC
assembly code
assembly to assembly translation
Low-level Compiler Optimizations
GAS
object code
GLD
SimplePower
SimplePower executable

# Improving Locality: Loop Transformations

- Linear loop transformations
  - Loop permutation
  - Loop skewing
  - Loop reversal
  - Loop scaling
- Iteration space (loop) tiling
- Multi-loop transformations
  - Loop fusion/fission

# Example: Loop Permutation

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

↓

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

- Changes data access from column to row order; thus improving locality (array elements in one cache line are now accessed sequentially)
- Improves both cache performance and energy
- Might lead to complex loop bounds/subscript functions; thus increasing datapath energy

# Example: Loop Tiling

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

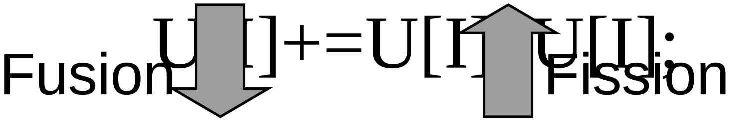
⇩

```
for(II=0; II<n; II+=T)
  for(JJ=0; JJ<n; JJ+=T)
    for(I=II; I<II+T-1; I++)
      for(J=JJ; J<JJ+T-1; J++)
        U[I][J] = V[I][J]
```

- Breaks up arrays too big to fit into cache
- Improves both cache performance and energy (better data reuse)
- Increases datapath energy:  more loops, more  comparisons, more branch ops

# Example:  Loop Fusion/Fission

```
for(I=0; I<n; I++)
   U[I]+=X*V[I];
for(I=0; I<n; I++)
   U[I]+=U[I]*U[I];
```



```
for(I=0; I<n; I++)
{  U[I]+=X*V[I];
   U[I]+=U[I]*U[I];
}
```

- Reduces the number of memory accesses
- Normally used along with scalar replacement
- Instruction cache energy might increase

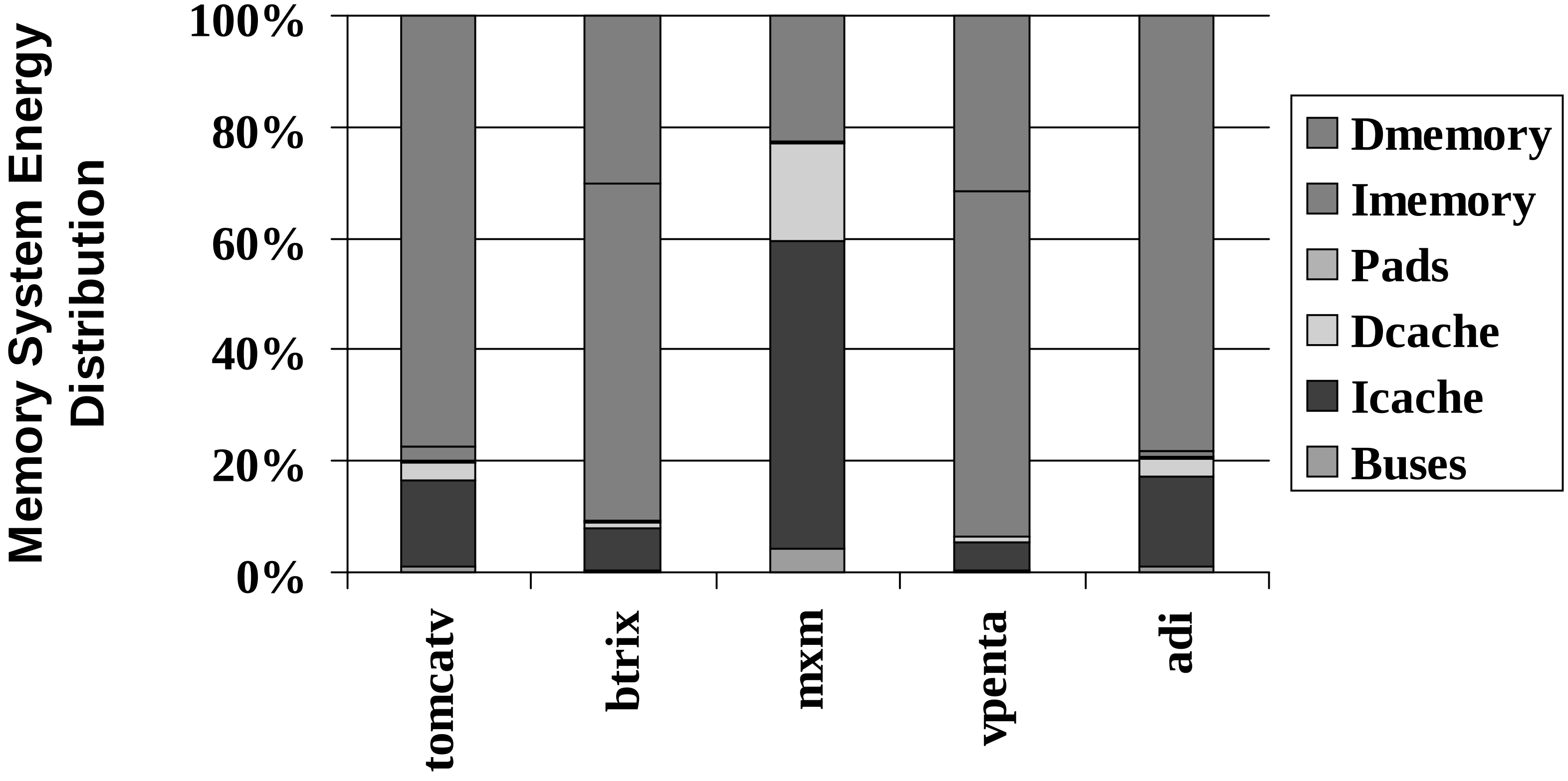
# Example: Loop Unrolling

```
for(I=0; I<n; I++)
   for(J=0; J<n; J++)
      for(K=0; K<n; K++)
         U[I][J]+=V[I][K]*W[K][J];
```

```
for(I=0; I<n; I+=b)
   for(J=0; J<n; J++)
      for(K=0; K<n; K++)
      {U[I][J]+=V[I][K]*W[K][J];
       U[I+1][J]+=V[I+1][K]*W[K][J];
          . . .
       U[I+b-1][J]+=V[I+b-1][K]*W[K][J];
      }
```

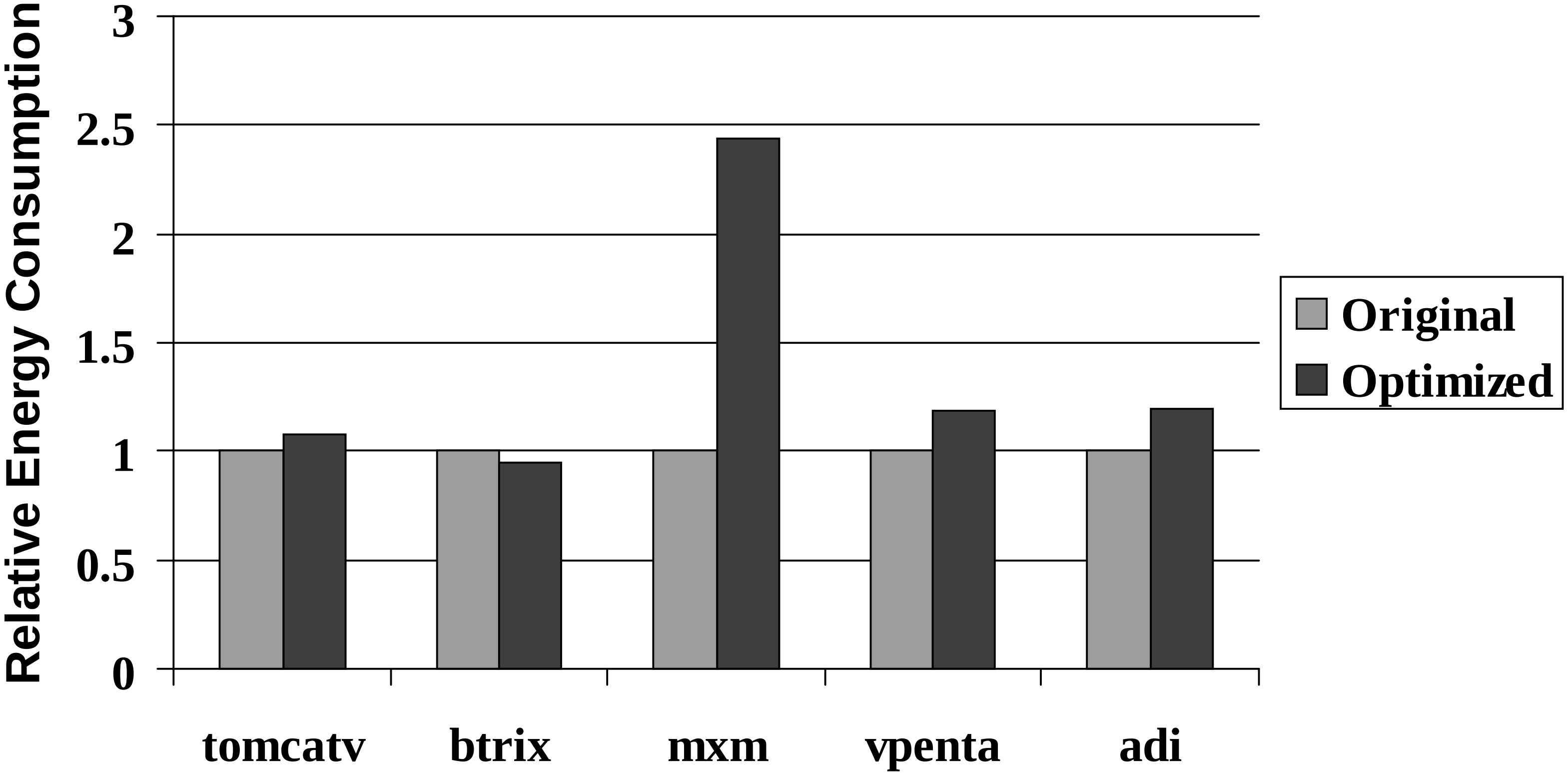
- Unroll outer loop b times so that W[K][J] can be accessed from the RegFile
- Improves both cache performance and energy (fewer accesses)
- Can improve datapath energy
- Can increase Icache energy

# Influence of Compiler Optimizations

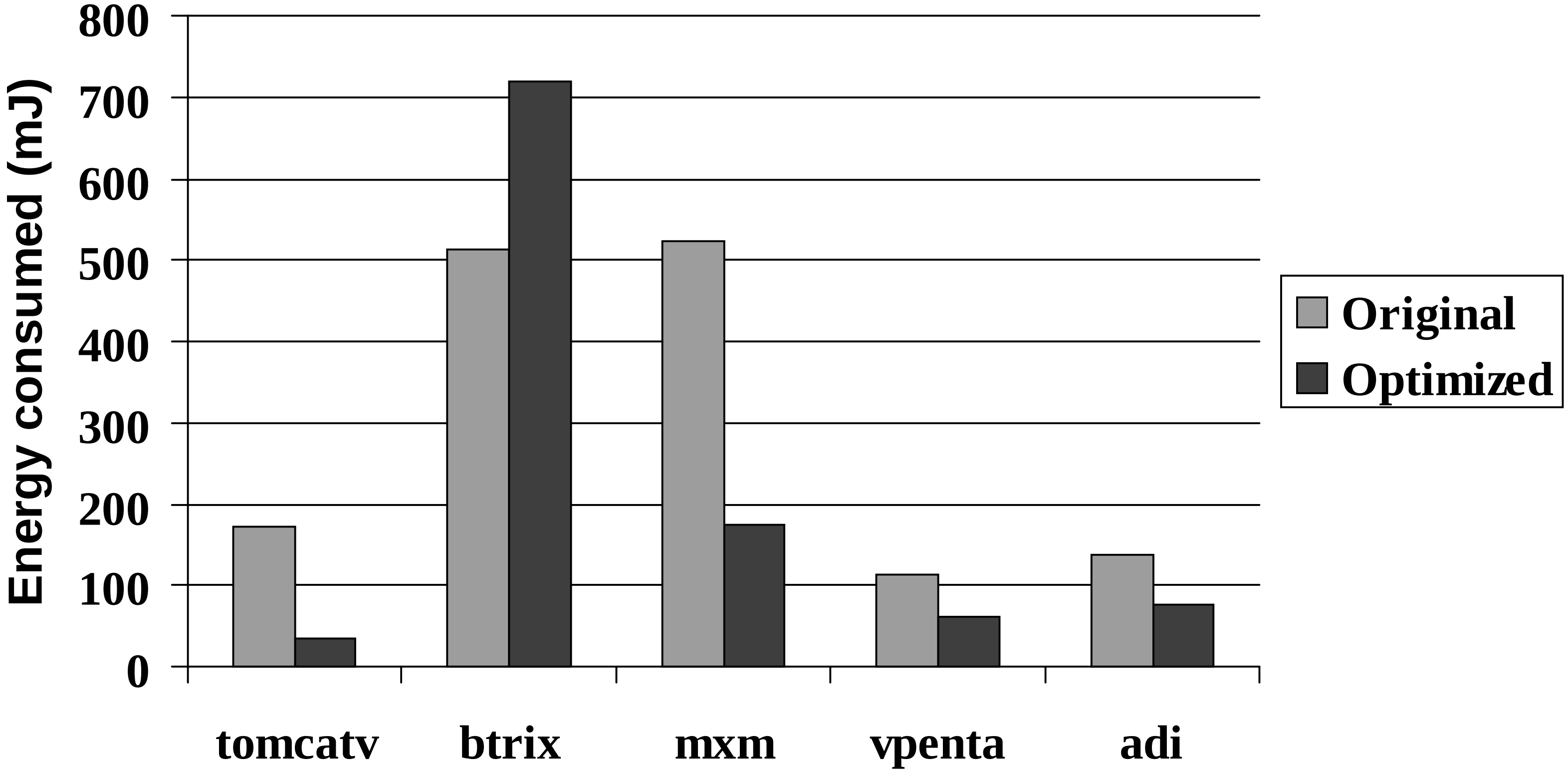


8K 4way Dcache, 8K 1way Icache

# Datapath Energy Consumption

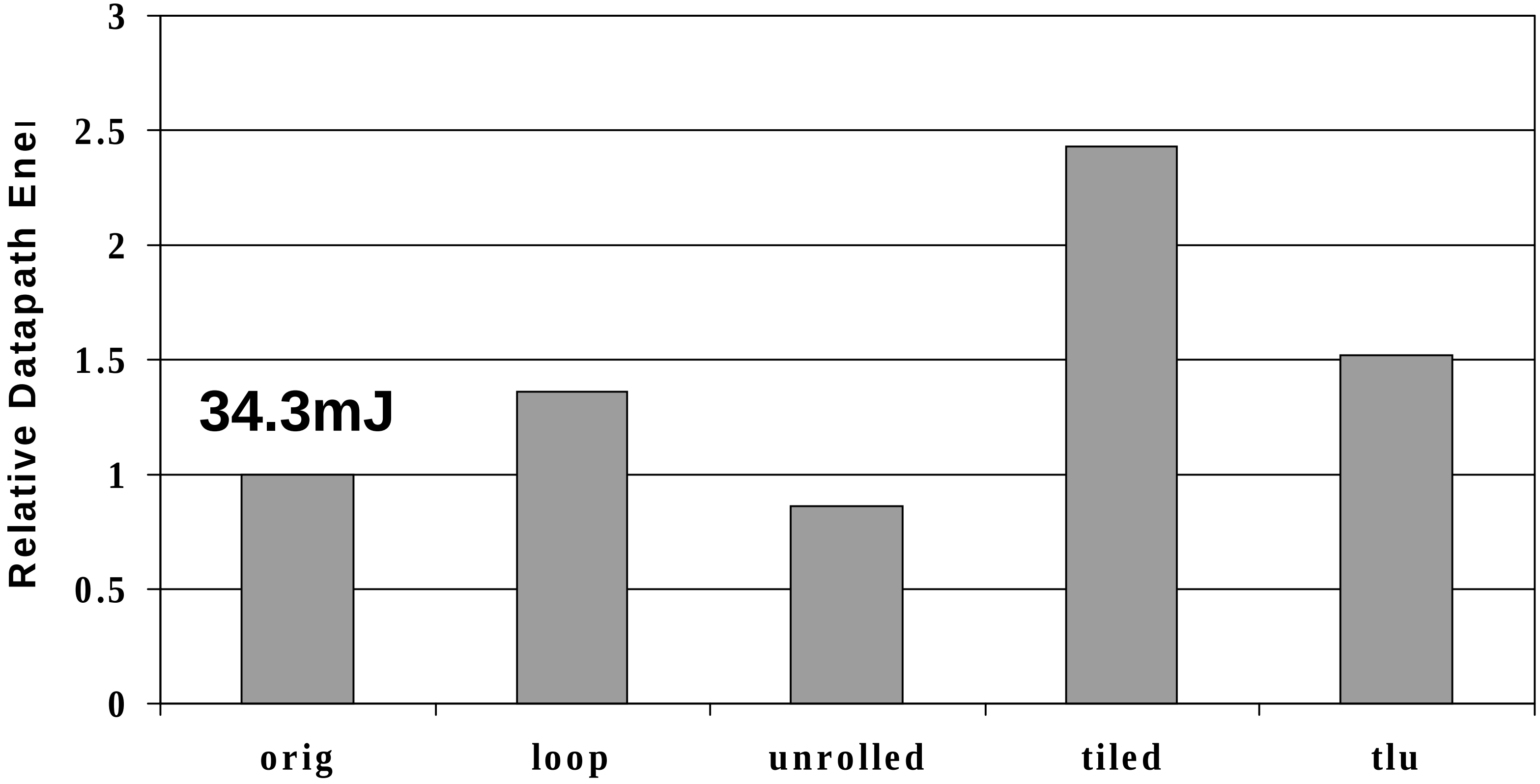


# Memory System Energy Consumption

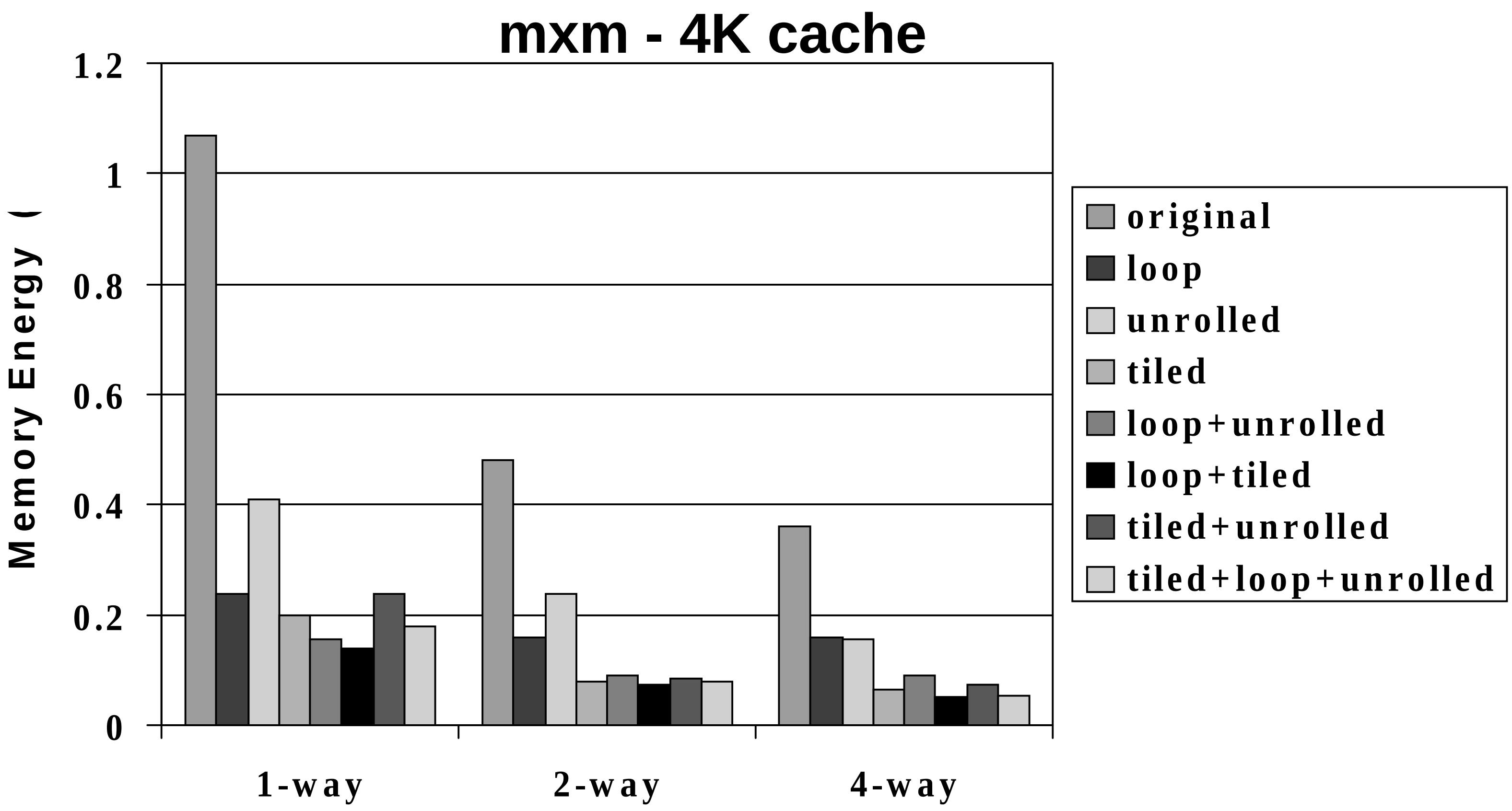


**4K 2way Dcache, 8K 1way Icache**

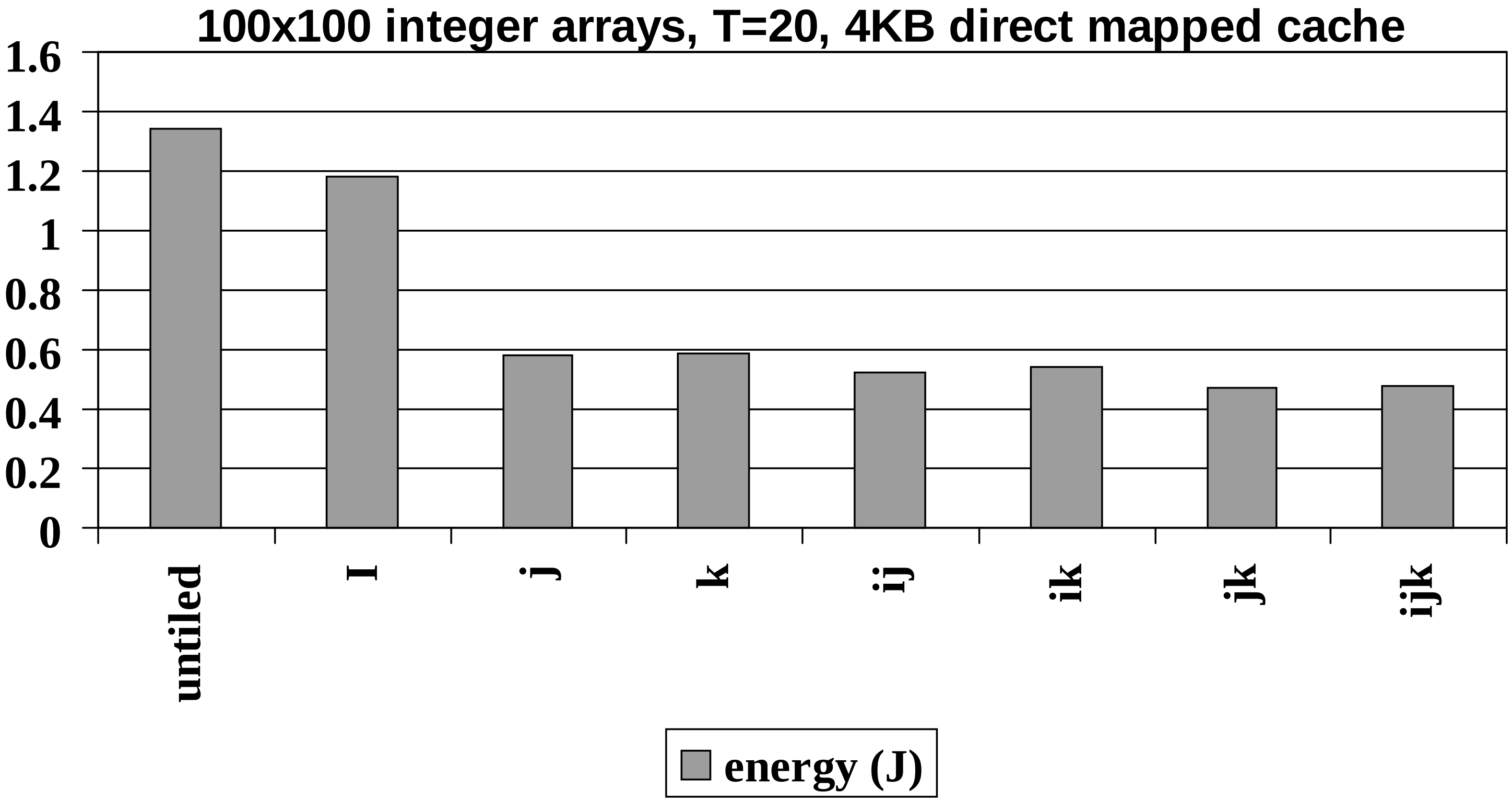
# Datapath Energy Consumption (mxm)



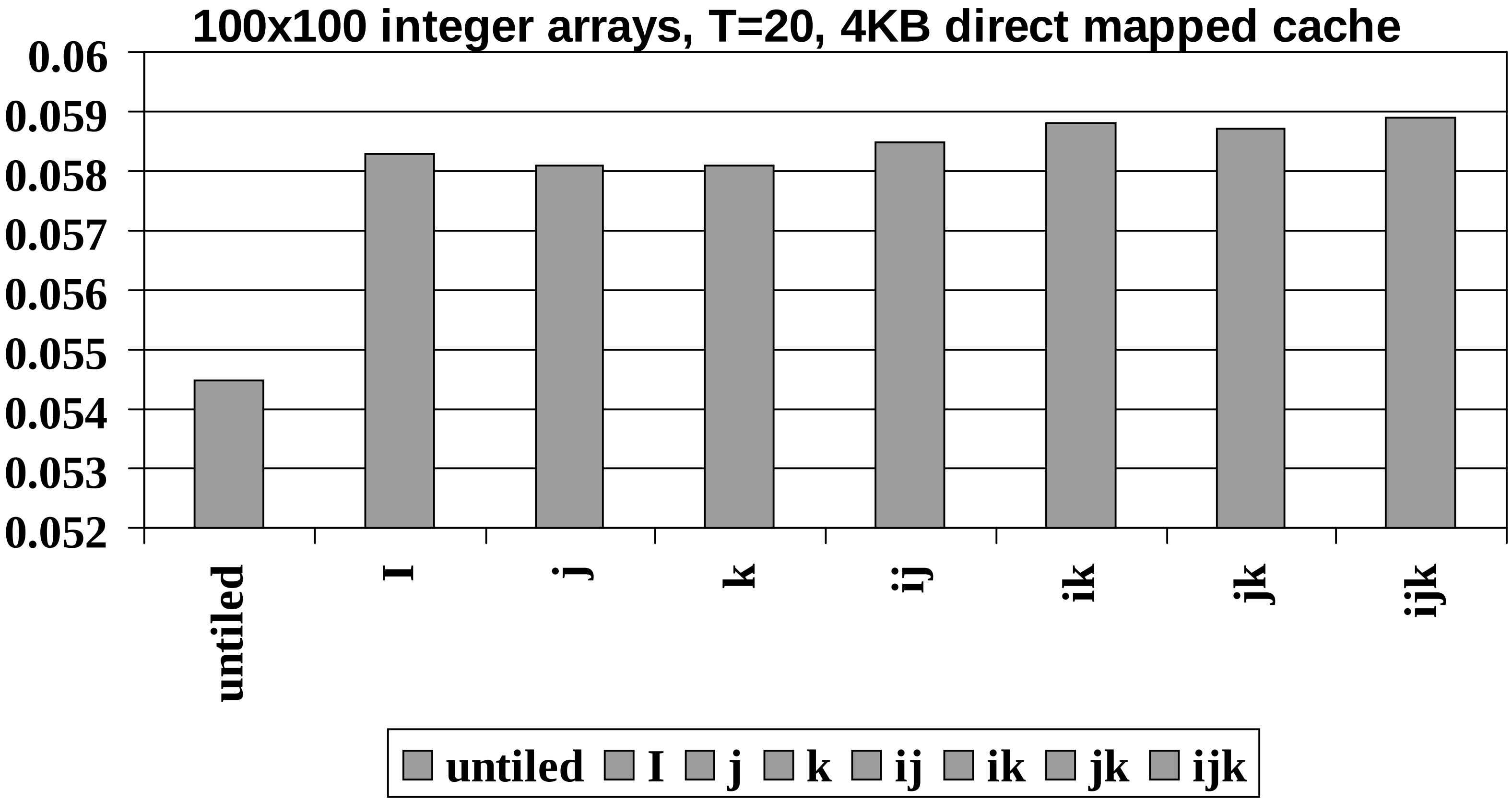
# Memory System Energy (mxm)



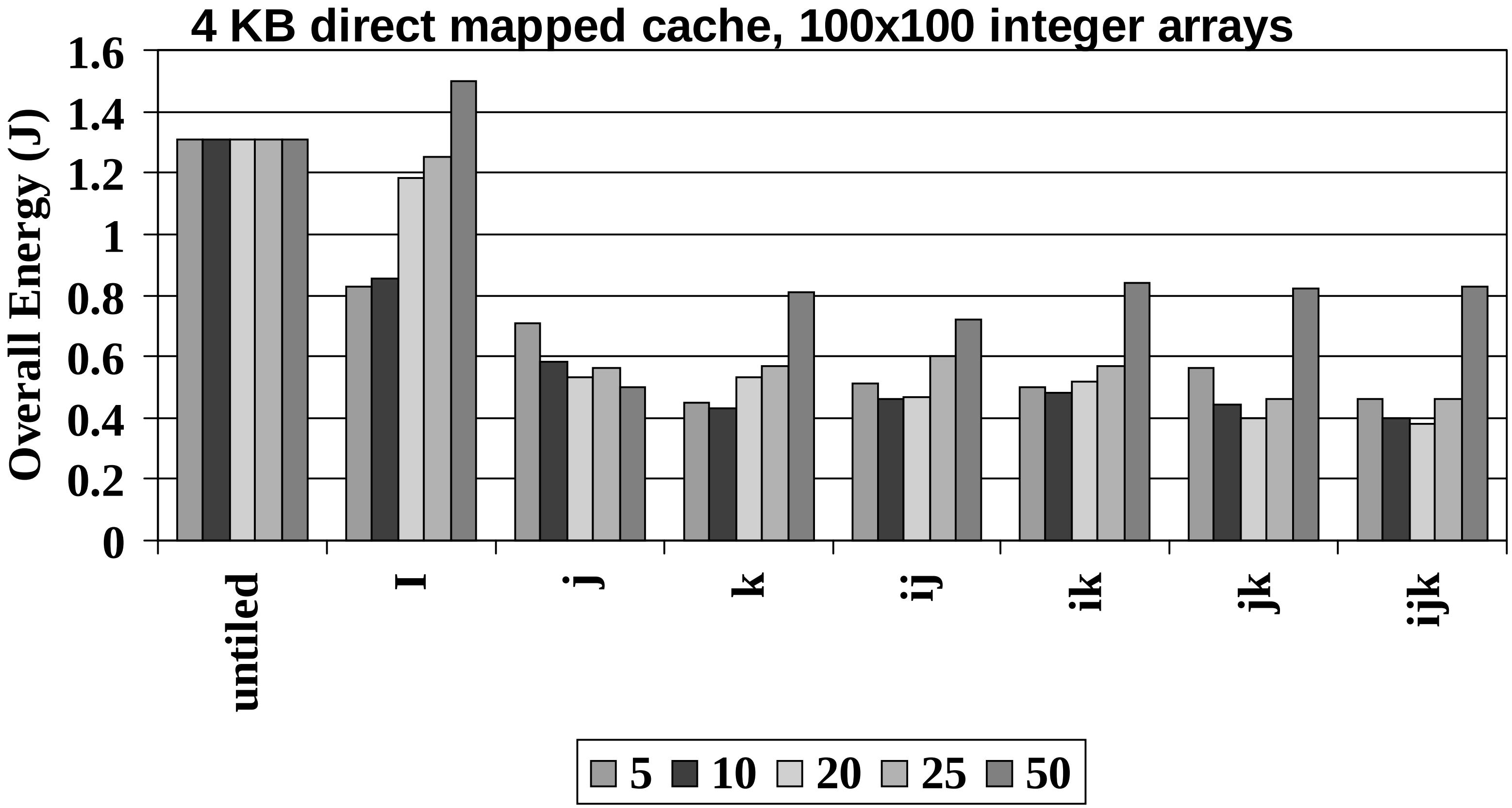
# Overall Energy (mxm)



# Datapath Energy (J) - mxm



# Sensitivity to the Tile Size (mxm)


4 KB direct mapped cache, 100x100 integer arrays
Overall Energy (J)
untiled
I
j
k
ij
ik
jk
ijk