Energy-Aware Memory Access Scheduling

Yongkui Han and Israel Koren
Electrical and Computer Engineering
University of Massachusetts, Amherst
January 30, 2004

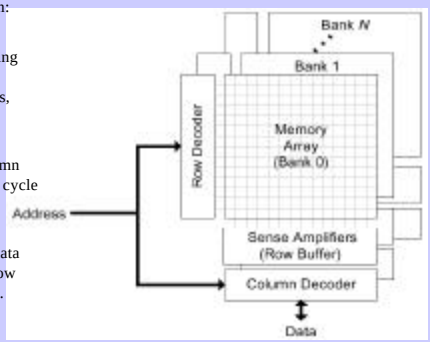
3-D structure of contemporary DRAM chips

Memory data location: (bank,row,column)

Three steps in accessing the memory data: precharge, row access, column access.

Once a row has been accessed, a new column access can issue each cycle until the bank is precharged.

It is faster to access data located in the same row than in different rows.



Motivation

- DRAM devices are accessed in units of L2 cache block size, e.g., 64 bytes.
- Energy values for Micron MT48LC16M16A2 SDRAM device:
  - Row activation and precharge energy: 20nJ
  - Column access for 64 bytes: 26nJ
- If we reorder the memory accesses to put together accesses to the same row, we can remove some unnecessary row activations and save DRAM energy.
- Next we compare a performance-aware scheduling policy with an energy-aware scheduling policy.

Example 1: putting together column accesses

(i,j,k) denotes an access to (bank, row, column)=(i,j,k)

1. original accesses:

   order = (0,0,0) (0,1,0) (0,0,4) (0,1,8) (0,0,8) (0,2,3)(0,1,12)

   energy = (20+26)*7 = 322nJ

2. performance-aware scheduling:

   order = (0,0,0) (0,0,4) (0,0,8) (0,1,0) (0,1,8) (0,1,12) (0,2,3)

   energy = 20+26*3+20+26*3+20+26 = 242nJ

3. energy-aware scheduling:

   identical to the above, saving 80nJ.

Example 2: critical word first

1. Original accesses:

order = (0,0,0) (0,0,1) (0,0,2) (0,0,3) (0,1,0) (0,1,1) (0,1,2) (0,1,3)

energy = 20+26*4+20+26*4 = 196nJ

(0,0,2) and (0,1,3) contain critical words.

2. Performance-aware scheduling:

order = (0,0,2) (0,1,3) (0,0,1) (0,0,2) (0,0,3) (0,1,0) (0,1,1) (0,1,2)

energy = 20+26+20+26+20+26*3+20+26*3 = 236nJ

3. Energy-aware scheduling:

order = (0,0,2) (0,0,0) (0,0,1) (0,0,3) (0,1,3) (0,1,0) (0,1,1) (0,1,2)

energy = 20+26*4+20+26*4 = 196nJ

We can save 40nJ compared to performance-aware scheduling.

Example 3: read-bypass-write

1. Original accesses:

order = (0,0,0)r (0,1,0)w (0,0,4)w (0,1,4)r (0,2,4)r ,

energy = 20+26+20+26+20+26+20+26+20+26 = 230nJ

2. Performance-aware scheduling:

order = (0,0,0)r (0,1,0)r (0,2,4)r (0,0,4)w (0,1,0)w

energy = 20+26+20+26+20+26+20+26+20+26 = 230nJ

3. Energy-aware scheduling:

order = (0,0,0)r (0,0,4)w (0,1,4)r (0,1,0)w (0,2,4)r

energy = 20+26+26+20+26+26+20+26 = 190nJ

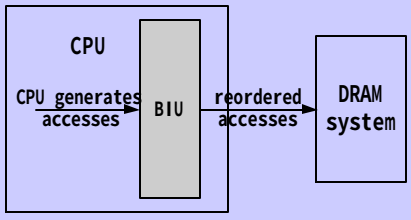
We can save 40nJ compared to performance-aware scheduling.

Three Scheduling Policies

- FCFS: First Come First Serve
  - this is the simplest one, just follow the original memory access order.
- RIFF: Read or Instruction Fetch First
  - give higher priority to memory read over memory write. this is a performance-aware scheduling policy, suggested by S. Rixner, et al.
- SRAF: Same Row Access First
  - give higher priority to memory accesses to the same row as the current one. this is an energy-aware scheduling policy we suggest.

Policy Implementation

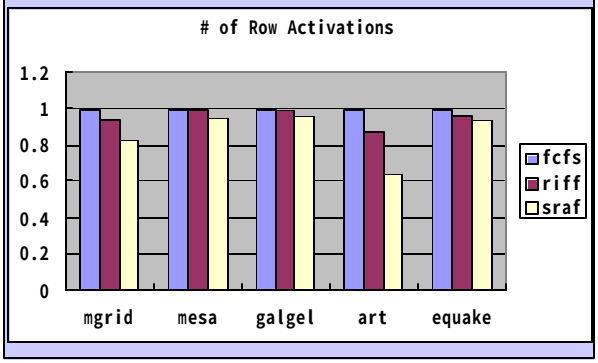
• All three policies are implemented at the BIU (Bus Interface Unit), which is located in the processor, before accesses go to the DRAM system from the processor.



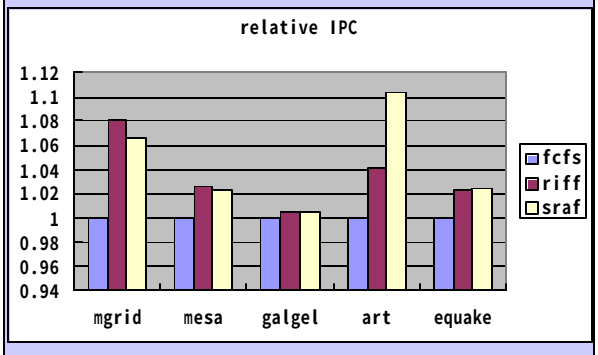
Experimental Setup

- Simulator: sim-mase (included in the simplescalar v4.0 test release) developed by UMich, which includes a DRAM simulator developed by UMD.
- 5 floating point SPEC2000 benchmarks.
- Simulation: fastforward 1 billion instructions, then simulate the next 100 million instructions in detail.
- Policy implementation overhead: Since the memory access scheduling operations can be overlapped with memory access operations, there is no performance overhead, and the energy overhead is negligible.

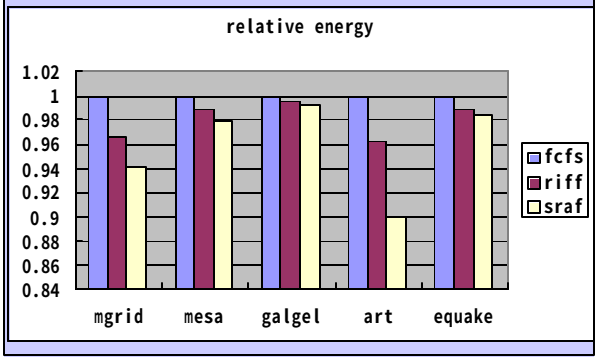
Simulation results: Number of row activations



Simulation results: Performance



Simulation results: Energy consumption



Conclusion

- Memory access order greatly affects the energy consumption of DRAM memory devices.
- By putting together accesses to the same row, we can remove many unnecessary row activations.
- We can save considerable energy through memory access scheduling, especially for memory access intensive applications.

Questions?