ESE532: System-on-a-Chip Architecture

Day 3: September 11, 2017
Memory

Today

Memory
- Memory Bottleneck
- Memory Scaling
- Latency Engineering:
  - Scheduling
  - Data Reuse: Scratchpad and cache
- Bandwidth Engineering
  - Wide word
  - Banking (form of parallelism for data)
- DRAM

Message

- Memory bandwidth and latency can be bottlenecks
- Minimize data movement
- Exploit small, local memories
- Exploit data reuse

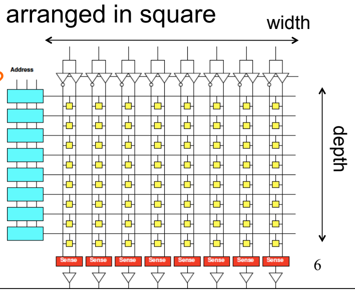
Memory Scaling

On-Chip Delay

- Delay is proportional to distance travelled
- Make a wire twice the length
  - Takes twice the latency to traverse
  - (can pipeline)
- Modern chips
  - Run at 100s of MHz to GHz
  - Take 10s of ns to cross the chip
  - Takes 100s of ns to reference off-chip data
- What does this say about placement of computations and memories?

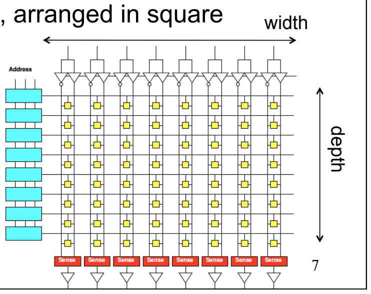
Memory Block

- Linear wire delay with distance
- Assume constant area/memory bit
- N-bit memory, arranged in square
- 4x capacity delay change?



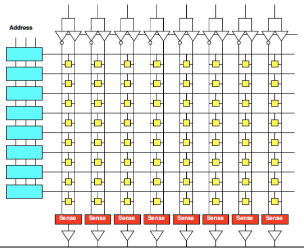
Memory Block

- Linear wire delay with distance
- Assume constant area/memory bit
- N-bit memory, arranged in square
- Width?
- Depth?
- Delay
  scale N?



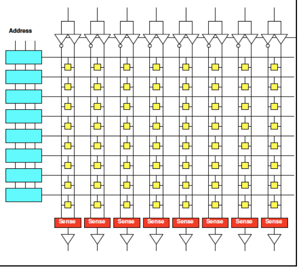
Memory Block Bandwidth

- N-bit memory, arranged in square
- Bits/cycle
  scale with N?



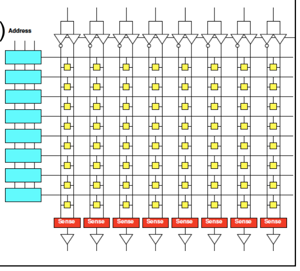
Memory Block Energy

- Assume read full width,
  - Energy scales with N
    - Activate sqrt(N) wires
    - Each wire sqrt(N) long
  - Or, energy/bit scales as sqrt(N)
- Larger memories cost more energy

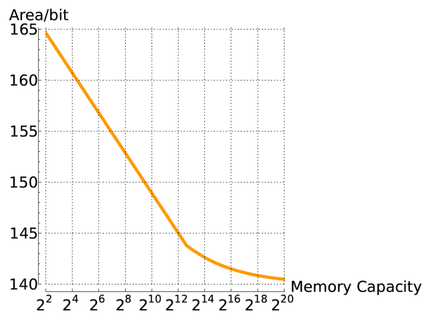


Memory Block Density

- Address, sense amplifies are large
- Scale slower than bit area
  - Bits scale N
  - Address sqrt(N)*log(N)
  - Sense amps sqrt(N)
- Large memories pay less per bit
  - denser

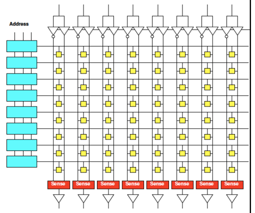


Compactness vs. Memory Size



Memory Scaling

- Small memories are fast
  - Large memories are slow
- Small memories low energy
  - Large memories high energy
- Large memories dense
  - Small memories cost more area per bit
- Combining:
  - Dense memories are slow



Latency Engineering Scheduling

Preclass 2

- 10 cycle latency to memory
- Throughput in each case?

```
Case 1:

  for(i=0;i<MAX;i++) {

    in=a[i]; // memory read
    out=f(in); // 10 cycle compute
    b[i]=out;
   }
```

```
Case 2:
next_in=a[0];
for(i=1;i<MAX;i++) {
    in=next_in;
    next_in=a[i];
    out=f(in);
    b[i-1]=out;
 }
b[MAX-1]=f(next_in);
```

Lesson

- Long memory latency can impact throughput
  - When must wait on it
  - When part of a cyclic dependency
- Overlap memory access with computations when possible
  - exploit parallelism between compute and memory access

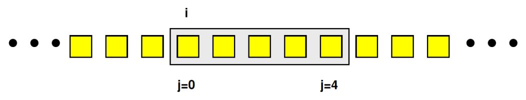
Latency Engineering Data Reuse

Preclass 3abc

- MAX=10^6 WSIZE=5
- How many reads to x[] and w[]?
- How many times execute t+=x[]*w[] ?
- Runtime
  read x,w 20
  t+=… 5

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
```

Preclass 3d

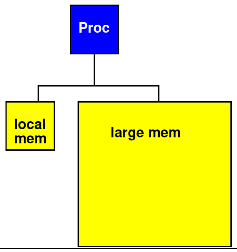


- How many distinct x[], w[] read?

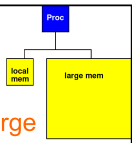
```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
```

Strategy

- Add small, local memory bank
  - Small memories are faster



Preclass 3e



- How use local memory to reduce large memory reads to 3d?
- How much local memory need?
- How much faster might small memory be?

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
```

Lesson

- Data can often be reused
  - Keep data needed for computation in
    - Closer, smaller (faster, less energy) memories
  - Reduces latency costs
  - Reduces bandwidth required from large memories
- Reuse hint: value used multiple times
  - Or value produced/consumed

Enhancing Reuse

- Computations can often be (re-)organized around data reuse

Reorganization Example

- What reused?
- Why problematic as written?
- How revise code to promote local reuse?

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
s=0;
for (i=0;i<MAX;i++) {
  s+=y[i];
  }
avg=s/MAX;
```

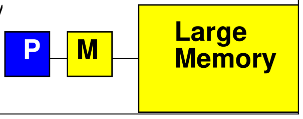
Reorganize

```
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  }
s=0;
for (i=0;i<MAX;i++) {
  s+=y[i];
  }
avg=s/MAX;
```

```
s=0;
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE;j++)
    t+=x[i+j]*w[j];
  y[i]=t;
  s+=t; // y[i]
  }
avg=s/MAX;
```

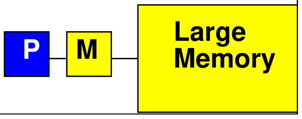
Processor Data Caches

- Traditional Processor Data Caches are a heuristic instance of this
  - Add a small memory local to the processor
    - It is fast, low latency
  - Store anything fetched from large/remote memory in local memory
    - Hoping for reuse in near future
  - On every fetch, check local memory before go to large memory



Cache

- **Goal:** performance of small memory with density of large memory



Processor Data Caches

- Demands more than a small memory
  - Need to sparsely store address/data mappings from large memory
  - Makes more area/delay/energy expensive than just a simple memory of capacity
- Don't need explicit data movement
- Cannot control when data moved/saved
  - Bad for determinism
- Limited ability to control what stays in small memory simultaneously

Terminology

- Cache
  - Hardware-managed small memory in front of larger memory
- Scratchpad
  - Small memory
  - Software (or logic) managed
  - Explicit reference to scratchpad vs. large (other) memories
  - Explicit movement of data

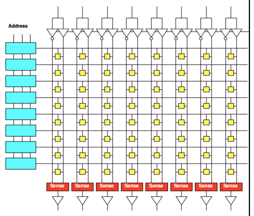
Bandwidth Engineering

Bandwidth Engineering

- High bandwidth is easier to engineer than low latency
  - Wide-word
  - Banking
    - Decompose memory into independent banks
    - Route requests to appropriate bank

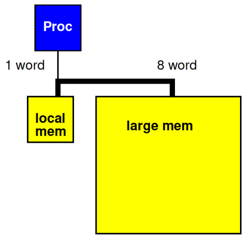
Wide Memory

- Relatively easy to have a wide memory
- As long as we share the address
  - One address to select wide word or bits
- Efficient if all read together



Revisit Preclass 3

- Use wide memory access to move 8 words (x[]) between large and small memory in a single cycle



Preclass 3 + wide

- Impact of 8 word datapath between large and local memory?
- WSIZE*MAX*(T_{comp}+3*T_{local})
- +WSIZE*(T_w+T_x)
- +MAX*T_x
- $5*10^6*(5+3)$
  $+5*(20+20)$
  $+10^6*20$
- $6*10^7$

```
for (j=0;j<WSIZE;j++)
  local_w[j]=w[j];
for (j=0;j<WSIZE-1;j++)
  local_x[j+1]=x[j];
for (i=0;i<MAX;i++) {
  t=0;
  for (j=0;j<WSIZE-1;j++)
    local_x[j]=local_x[j+1];
  local_x[WSIZE-1]=x[i+WSIZE-1];
  for (j=0;j<WSIZE;j++)
    t+=local_x[j]*local_w[j];
  y[i]=t;
  }
```

Preclass 4

```
for (i=0;i<MAX;i++) {
  y[i]=a[i]*b[i];
  }
```

```
for (i=0;i<MAX;i++) {
  y[i]=a[b[i]];
  }
```

Cycles reading a, b from large memory.
20 cycle read latency; MAX=10^6

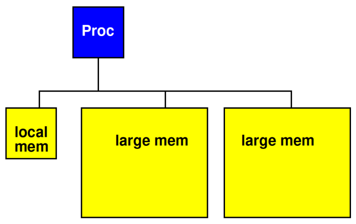
	a[i]*b[i]	a[b[i]]
1-word wide		
8-word wide		

Lesson

- Cheaper to access wide/contiguous blocks memory
  - In hardware
  - From the architectures typically build
- Can achieve higher bandwidth on large block data transfer
  - Than random access of small data items

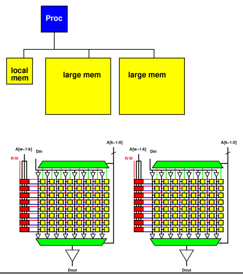
Bank Memory

- Break memory into independent banks
  - Allow banks to operate concurrently



Bank Memory

- Break memory into independent banks
  - Allow banks to operate concurrently



Preclass 4 Banked

```
for (i=0;i<MAX;i++) {
  y[i]=a[i]*b[i];
  }
```

```
for (i=0;i<MAX;i+=2) {
  local_a_even=a[i];
  local_a_odd=a[i+1];
  local_b_even=b[i];
  local_b_odd=b[i+1];
  y[i]=local_a_even*local_b_even
  y[i+1]=local_a_odd*local_b_odd
  }
```

20 cycle read latency; MAX=10^6
Put odd a, b in one bank, even in other.
Cycles reading a, b from large memory?

Preclass 4 Banked

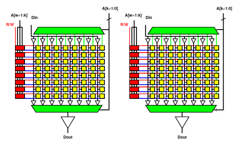
```
for (i=0;i<MAX;i++) {
  y[i]=a[b[i]];
  }
```

```
for (i=0;i<MAX;i+=2) {
  local_b_even=b[i];
  local_b_odd=b[i+1];
  local_a_even=a[local_b_even];
  local_a_odd=a[local_b_odd];
  y[i]=local_a_even;
  y[i+1]=local_a_odd;
  }
```

20 cycle read latency; MAX=10^6
Put odd a, b in one bank, even in other.
Cycles reading a, b from large memory?

Banking Costs

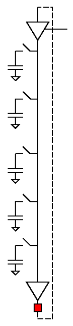
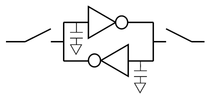
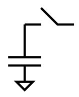
- Area compare
  - One memory bank
    - Depth 1024, width 64
  - Two memory banks
    - Depth 1024, width 32



DRAM

Dynamic RAM

- Goes a step further
- Share refresh/restoration logic as well
- Minimal storage is a capacitor
- “Feature” DRAM process is ability to make capacitors efficiently
- Denser, but slower

DRAM

- 1GB DDR3 SDRAM from Micron
  - http://www.micron.com/products/dram/ddr3/
  - 96 pin pakage
  - 16b datapath IO
  - Operate at 500+MHz
  - 37.5ns random access latency

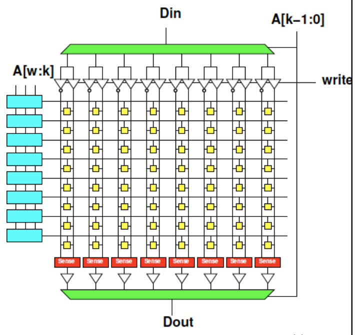
Table 1: Key Timing Parameters

Speed Grade	Data Rate (Mb/s)	Target RCD/RP/CL	^tRCD (ns)	^tRP (ns)	CL (ns)
-25E	800	5-5-5	12.5	12.5	12.5
-25	800	6-6-6	15	15	15
-187E	1,066	7-7-7	13.1	13.1	13.1
-187	1,066	8-8-8	15	15	15
-15E	1,333	9-9-9	13.5	13.5	13.5
-15	1,333	10-10-10	15	15	15

Options	Marking
• Configuration	
– 64 Meg x 16 (8 Meg x 16 x 8 banks)	64M16
– 128 Meg x 8 (16 Meg x 8 x 8 banks)	128M8
– 256 Meg x 4 (32 Meg x 4 x 8 banks)	256M4
• FBGA package (lead-free)	
– x4, x8; 86-ball FBGA (9mm x 15.5mm)	BY
– x16; 96-ball FBGA (9mm x 15.5mm)	LA
• Timing – cycle time	
– 2.5ns @ CL = 6 (DDR3-800)	-25
– 2.5ns @ CL = 5 (DDR3-800)	-25E
– 1.87ns @ CL = 8 (DDR3-1066)	-187
– 1.87ns @ CL = 7 (DDR3-1066)	-187E
– 1.5ns @ CL = 10 (DDR3-1333)	-15
– 1.5ns @ CL = 9 (DDR3-1333)	-15E

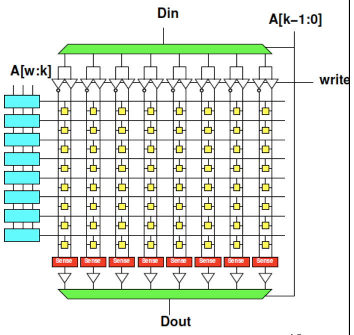
Memory Access Timing

- Access:
  1. Address
  2. Row fetch
  3. Column select
  4. writeback/ refresh
- Optimization for access within a row



Memory Access Timing

- Access
  1. Address
  2. Row fetch
  3. Column select
     - Can repeat
  4. writeback/ refresh
- Row 1024b—8192b
- Faster to access within row



DRAM

- 1GB DDR3 SDRAM from Micron
  - http://www.micron.com/products/dram/ddr3/
  - 96 pin pakage
  - 16b datapath IO
  - Operate at 500+MHz
  - **37.5ns** random access latency

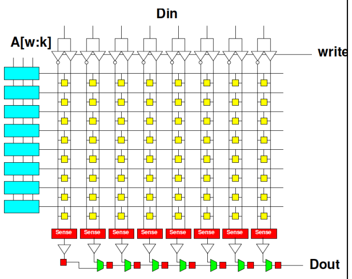
Table 1: Key Timing Parameters

Speed Grade	Data Rate (Mb/s)	Target RCD/RP/CL	^tRCD (ns)	^tRP (ns)	CL (ns)
-25E	800	5-5-5	12.5	12.5	12.5
-25	800	6-6-6	15	15	15
-187E	1,066	7-7-7	13.1	13.1	13.1
-187	1,066	8-8-8	15	15	15
-15E	1,333	9-9-9	13.5	13.5	13.5
-15	1,333	10-10-10	15	15	15

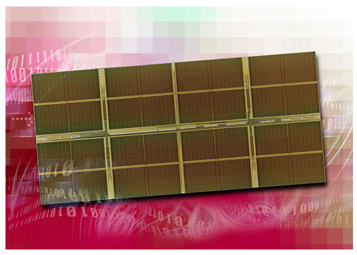
Options	Marking
• Configuration	
– 64 Meg x 16 (8 Meg x 16 x 8 banks)	64M16
– 128 Meg x 8 (16 Meg x 8 x 8 banks)	128M8
– 256 Meg x 4 (32 Meg x 4 x 8 banks)	256M4
• FBGA package (lead-free)	
– x4, x8; 86-ball FBGA (9mm x 15.5mm)	BY
– x16; 96-ball FBGA (9mm x 15.5mm)	LA
• Timing – cycle time	
– 2.5ns @ CL = 6 (DDR3-800)	-25
– 2.5ns @ CL = 5 (DDR3-800)	-25E
– 1.87ns @ CL = 8 (DDR3-1066)	-187
– 1.87ns @ CL = 7 (DDR3-1066)	-187E
– 1.5ns @ CL = 10 (DDR3-1333)	-15
– 1.5ns @ CL = 9 (DDR3-1333)	-15E

DRAM Streaming

- Reading row is 15ns
- 16b @ 500MHz
- 1024b row
- 1024/16
  - 64 words per row
- How supply 16b/2ns



1 Gigabit DDR2 SDRAM



[Source: http://www.elpida.com/en/news/2004/11-18.html]

DRAM

- Latency is large (10s of ns)
- Throughput can be high (GB/s)
  - If accessed sequentially
  - If exploit wide word block transfers
- Throughput low on random accesses
  - As we saw for random access on wide-word memory

Memory Organization

- Architecture contains
  - Large memories
    - For density, necessary sharing
  - Small memories local to compute
    - For high bandwidth, low latency, low energy
- Need to move data
  - Among memories
    - Large to small and back
    - Among small

Big Ideas

- Memory bandwidth and latency can be bottlenecks
- Exploit small, local memories
  - Easy bandwidth, low latency, energy
- Exploit data reuse
  - Keep in small memories
- Minimize data movement
  - Small, local memories keep distance short
  - Minimally move into small memories

Admin

- Reading for Wednesday on canvas
- HW2 due Friday