18-643 Lecture 12: FPGA Memory Architecture

James C. Hoe
Department of ECE
Carnegie Mellon University
Housekeeping

• Your goal today: think about memory from FPGA computing perspective
  (I assume you have taken a comp arch course)

• Notices
  – 10/15: midterm in class
  – 10/21 (due noon): Lab 3 AND Proposal Slides
  – 10/{22, 23, 24}: proposal presentations

• Readings: memory chapter in any good comp arch textbook
Required Proposal Outline

1. Introduction - what is the problem and metric
2. Motivation - why is it important/interesting
3. Overview of approach - what do you do exactly
4. Overview of expected results
5. Weekly “testable” milestones
6. Risk assessment (what are the known unknowns? do you have everything you need to start? do you know how to do every step? do you know already what will happen?)
Ground Rule #1:
Fast means Small and Expensive

• Bigger is slower
  – SRAM 512 Bytes @ sub-nsec
  – SRAM KByte~MByte @ nsec
  – DRAM GByte @ ~50 nsec
  – SSD TByte @ msec
  – Hard Disk TByte @ ~10 msec

• Faster is more expensive (dollars and chip area)
  – SRAM ~$10K per GByte
  – DRAM ~$10  per GByte
  – “Drives” ~$0.1  per GByte

Treat the values as ×/÷ 3x
Ground Rule #2: Locality is Good

- Temporal: after accessing A, how many other distinct addresses before accessing A again
- Spatial: after accessing A, how many other distinct addresses before accessing a near-by B
- Good locality implies
  - easier to cover more of working-set with a small (fast) memory
  - lower BW and/or more efficient use of BW to large (slow) memory

MMM::good; sparse MMM::bad; streaming::1 of 2
Ground Rule #3: Data Movement not Free

- Latency: transit time between source and dest
- Overhead: dead time spent in the act of sending or receiving not overlapped with concurrent compute
- Gap: wait time in between successive send’s or receive’s due to limited transfer BW

see LogP [Culler, et al., PPoPP93]
Ground Rule #4: Memory Perf Matters

- An algorithm has a cost in terms of operation count
  - \( \text{runtime}_{\text{compute-bound}} = \frac{\# \text{ operations}}{\text{FLOPS}} \)
- An algorithm also has a cost in terms of number of bytes communicated (ld/st or send/receive)
  - \( \text{runtime}_{\text{BW-bound}} = \frac{\# \text{ bytes}}{\text{BW}} \)
- Which one dominates depends on
  - ratio of FLOPS and BW of platform
  - ratio of ops and bytes of algorithm
- Average Arithmetic Intensity (AI)
  - how many ops performed per byte accessed
  - \( \frac{\# \text{ operations}}{\# \text{ bytes}} \)
DRAM, SRAM and all that
On-Chip Fast/Small Memory

- **LUT-RAM**: 64x1b or 32x2b
  - dual-port: [sync write/async read] + async read
  - also good as one 32b or two 16b shift-registers
- **BRAM**: 36Kb, variable aspect ratio 1~72b wide
  - true dual-port: 2x[sync read/write], separate clocks
  - fast-enough to be double-pump’ed
- Become FF array if infeasible as hard macro
FPGA Memory Peculiarities

- Large memory abnormally fast
- Large memory are “free” until your run-out
- Quantized memory options
  - jumps between FF-based vs. LUT-RAM vs. BRAMs
  - choose from fixed menu of sizes and aspect ratios
- Must manage RAM usage
  - don’t waste BRAM on small buffers
  - tune buffer sizes to natural granularities, e.g., zero incremental cost to go from 2Kb to 4Kb
  - pack buffers to share same physical array
Off-Chip Memory: DRAM

- Simple asynchronous request/reply queues
  - in-order or out-of-order (need tags)
  - multiple queues, separate read/write queues
- Long and variable but predictable latency
- Need very wide data word for bandwidth

\[ \text{e.g.}, \ 8\text{B}@200\text{MHz} \text{ is only } 1.6\text{GB/sec} \]
Variable Latency

- **DRAM organization**
  - (multiple ranks per DIMM)
  - (multiple chips per rank)
  - multiple banks per chip
- **Per bank**
  - long delay to new row
  - very fast to same row
- **Rows refreshed every 64ms**
  - bank unavailable for 30~40ns at a time
  - avg. ~1% unavailability
- **chip/package/board**
  add to total latency

16K~256K rows
×
1K columns

- per chip: 3.2G transfers/sec of 4b~16b
  - (per rank: 8B-wide ⇒ peak 25.6GB/sec)
Expect much more SRAM capacity and DRAM Bandwidth

- Interest in FPGA computing sets new “balance”
- 10 years ago
  - single-digit GB/sec to DRAM
  - single-digit Mbit SRAM
- New FPGAs (more than Moore increase)
  - GPU-level memory bandwidth (HBM, HMC, ...)
  - 10s MByte SRAM
  - also add GPU-level FP throughput

No free lunch in Watt/Perf though

- Will we see hardened memory hierarchy next?
Memory Organization
Memory Banking

- Partition storage onto multiple structures
- More BW for parallel, non-conflicting accesses

![Memory Banking Diagram]
Interleaving

- An array of \(N\) words; index is \(\log_2 N\) bits
- \(N\)-word total storage
  - divided into \(B\) banks; bank number is \(\log_2 B\) bits
  - each bank is \(W\)-word wide; word-offset is \(\log_2 W\) bits
  - index within bank is \(\log_2 (N/B/W)\) bits
- Objectives
  - maximize spatial locality in **green**
  - maximize entropy in **red**

In general interleaved & reordered
Example: Image Frame

- N pixels in $\sqrt{N}$-by-$\sqrt{N}$ frame
- Spatial locality in $\sqrt{W}$-by-$\sqrt{W}$ tiles
- Parallelism in same-column tiles

In SW, this is done with data layout and copying; in FPGA, this is just permuting address wires.
An Extreme Example: Convey RC

- Sustains 80GB/sec peak BW on irregular accesses by 4 user-application FPGAs
- Not cheap
  - 8 FPGAs memory controllers for 16 DDR2 channels
  - regular or prime \((2^5-1)\) interleaving
  - proprietary *gather-scatter* DIMM with independent addressing for each 8-byte word in a 64-byte DDR transfer
- Optimize for BW over latency
- \(~$50K, mostly in FPGAs and DRAMs\)
Classic Memory Hierarchy 101
(pre-multicore)

• Memory hierarchy level $i$ has access time of $t_i$

• Perceived access time $T_i$ is longer than $t_i$
  – a chance (hit-rate $h_i$) you find what you want $\Rightarrow t_i$
  – a chance (miss-rate $m_i$) you don’t find it $\Rightarrow t_i + T_{i+1}$
  – $h_i + m_i = 1.0$

• In general

$$T_i = h_i \cdot t_i + m_i \cdot (t_i + T_{i+1})$$

$$T_i = t_i + m_i \cdot T_{i+1}$$

Think this of as “miss penalty”

Note: $h_i$ and $m_i$ are of the references missing at level $i-1$

$h_{\text{bottom-most}} = 1.0$
FPGA hierarchy used differently

- 200MHz soft-logic cache
  - a miss to DRAM is not too many cycles away
  - if 4-byte access, 100% hit-rate only 0.8 GB/sec

- Remember to think spatial
  - distributed concurrent bandwidth! for spatial kernels
  - reduce off-chip memory bandwidth and power!

10s GB/sec per core
1 cycle latency @ GHZ+

10s GB/sec per channel
100s core clock latency

GB/sec per kernel
1 cycle latency @200MHz

10s GB/sec per channel
~10 fabric clock latency
Cache vs Scratchpad

• Manual scratchpad is easy for regular/structured locality
  – per-kernel scratchpad more opportunity and benefit in specialization
  – HW management does not lengthen critical path
  – prefetching can hide memory latency completely
  – 95% of the time: streaming or double-buffering

  These easy cases actually against cache heuristics

• Cache is useful when locality is not predictable ahead of time—remember to customized the cache if you have to use one!!!
Loop Nests to Streams


\[
\begin{align*}
\text{for (i=...}
\text{for (j=...}
\text{for (k=...}
\text{C[i][j] += A[i][k] * B[k][j]}
\end{align*}
\]

turn indexed memory references to streams
Custom Memory Support per Stream

| for(i=...  
  | for(j=...  
  | for(k=...  
  | GET A[i][k] |
|---|---|---|---|
| for(i=...  
  | for(j=...  
  | for(k=...  
  | GET B[k][j] |
|---|---|---|---|
| for(i=...  
  | for(j=...  
  | for(k=...  
  | GET C[i][j]  
  | READ SYNC |
|---|---|---|---|
| for(i=...  
  | for(j=...  
  | for(k=...  
  | PUT C[i][j]  
  | WRITE SYNC |

Synchronization on C[i][j] due to RAW dependence
Cache Repeated Data

for (k=... for (i=... for (j=... GET A[i][k]

repeat

DRAM
Coalesce Sequential Transfers

\[
\text{for}(k= \ldots \\
\text{for}(i=\ldots \\
\quad \text{for}(j=\ldots \\
\quad \quad \text{GET}\ B[k][j] \\
\text{for}(k=\ldots \\
\text{for}(i=\ldots \\
\quad \text{GET-ROW}\ B[k]}
\]
Strided Accesses

\[
\text{for}(k=\ldots \quad \text{for}(i=\ldots \quad \text{for}(j=\ldots \\
\quad \quad \text{GET } A[i][k]}
\]

strided accesses bad for caches and DRAM row-buffer
Fetch Blocks of Rows

for (k = ...
    for (i = ...
        [for (j = ... ]
        GET A[i][k]

Work on blocked regions
• access DRAM in rows
• buffer multiple rows
• permute data on-the-fly
to form strided column stream
Recognize Accumulators

for(k=... for(k=...
  for(i=... for(i=...
    for(j=... for(j=...
      GET C[i][j] PUT C[i][j]
      READ SYNC WRITE SYNC

Accum Buffer
Parting Thoughts

• Memory architecture and memory performance important to computing

• Spatial FPGA computing needs adjusted intuition
  – wider, more concurrent access
  – slow clock ticks
  – amenable to extreme specialization

• Look forward to
  – more SRAM, faster DRAM
  – more data movement BW within fabric
  – cache-coherence, better integration
  – hardwired, native memory architecture?