18-643 Lecture 13:
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
  – 10/12: Joe Melber Lecture: CoRAM
  – 10/17: Midterm (this is last lecture covered)
  – 10/19: Marie Nguyen Lecture: Smart Headlights
  – 10/20 (due noon): Lab 3 AND Proposal slides
  – 10/{24, 26}: proposal presentations

• Readings: memory chapter in any good comp arch textbook
Ground Rule #1: Fast means Small and Expensive

- Bigger is slower
  - SRAM 512 Bytes sub-nanosec
  - SRAM KByte~MByte nanosec
  - DRAM GByte ~50 nanosec
  - Hard Disk TByte ~10 millisec

- Faster is more expensive (dollars and chip area)
  - SRAM ~$10K per GByte
  - DRAM ~$10 per GByte
  - Hard Disk ~$0.1 per GByte

Treat the values as $\times/\div 3x$

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]

![Diagram showing data movement and latency, overhead, and gap](image)

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-reg
- 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

Emulating 1 Write + 2 Read

1W1R

• data_W
• addr_W
• we_W

1W1R

• data_R0
• addr_R0

• data_R1
• addr_R1
Emulating 2 Write + 1 Read

[LaForest and Steffan, ISFPGA’10]

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

e.g., 8B@200MHz is only 1.6GB/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

Aside: Why is DRAM slow?

- DRAM fabrication chosen to scale with Moore’s law in capacity and cost, not speed
- Between 1980 ~ 2012
  - 64K bit → 1024M bit (exponential ~41% annual)
  - $1M/Gb → $1/Gb
  - 250ns → 35ns (linear)
- This is a deliberately engineered path
  - Amdahl’s Other Law: capacity needs to grow with CPU performance
  - DRAM/processor speed difference reconciled by adding caches (L1 in 80s, L2 in 90s, L3 in 2000, ....)

Faster, less dense, more expensive DRAMs do exist
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?

Cache Coherent FPGA Accelerator (Intel QPI, IBM CAPI, Xilinx Zynq)

FPGA sees same addr space with CC
⇒ fine-grain sharing of data and work
⇒ irregular applications enabled
Memory Organization

Banking

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

```
data_W_0  addr_0  we_0  RAM  data_R_0
          |          |      |      |      |
          |          |      |      |      |
data_W_1  addr_1  we_1  RAM  data_R_1
          |          |      |      |      |
          |          |      |      |      |
      ...      |      |      |      |
data_W_*  addr_*  we_*  RAM  data_R_*
          |          |      |      |      |
          |          |      |      |      |
```

---

18-443-F17-1L13-515, James C. Hoe, CMU/ECE/CALCM, ©2017
Interleaving

- An array of $N$ words; index is $\lg_2 N$ bits
- $N$-word total storage
  - divided into $B$ banks; bank number is $\lg_2 B$ bits
  - each bank is $W$-word wide; word-offset is $\lg_2 W$ bits
  - index within bank is $\lg_2 (N/B/W)$ bits
- Objectives
  - maximize spatial locality in green
  - maximize entropy in red

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 \textit{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!**

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!!!
Designing Memory for Performance

AI and Roofline
[Williams&Patterson, 2006]

Attained Performance of a system (op/sec)

Al of application

runtime > max ( # op/FLOPS, # byte/BW)
> #op·max(1/FLOPS, 1/(AI·BW))

perf = min(FLOPS, AI·BW)
The Balancing Act

1. Kernels’ \textbf{op/sec} requires some \textbf{byte/sec} — a function of \textbf{kernel size}
2. On-chip SRAM “filters” kernel \textbf{byte/sec} down to DRAM \textbf{byte/sec} — a function of SRAM \textbf{capacity}
3. DRAM system offers some aggregate \textbf{byte/sec} — a function of \textbf{access pattern}

Basic AI Example: MMM

\begin{verbatim}
for(i0=0; i0<N; i0+=Nb)
  for(j0=0; j0<N; j0+=Nb)
    for(k0=0; k0<N; k0+=Nb) {
      for(i=i0;i<i0+Nb;i++)
        for(j=j0;j<j0+Nb;j++)
          for(k=k0;k<k0+Nb;k++)
            C[i][j]+=A[i][k]*B[k][j];
    }
\end{verbatim}

- Imagine a ‘\(N/N_b\)‘x’‘\(N/N_b\)’ \textbf{MATRIX} of \(N_b\times N_b\) matrices
  - inner-triple is straightforward \textbf{matrix-matrix} mult
  - outer-triple is \textbf{MATRIX-MATRIX} mult
- To improve AI, hold \(N_b\times N_b\) sub-matrices on-chip for data-reuse

need to copy block (not shown)
2D-FFT

• Row-column algorithm:

\[ 2D-DFT_{n \times n} = (DFT_{n} \otimes I_{n}) (I_{n} \otimes DFT_{n}) \]

Column Stage  Row Stage

Dataset:
(Logical abstraction of the 2D dataset)

Inefficient DRAM Access Patterns

• Row-wise traversal -> Sequential accesses
• Column-wise traversal -> Large strided accesses
**Tiled Layout and Access Patterns**

- **Row-major "blocked"**
  - Linear mem space
  - In row-buffer sized chunks

- **Design Generator w/ Tensor Formalism**
  - **2D-DFT**
    - \[ 2D-DFT_{n \times n} = \left( DFT_n \otimes I_n \right) \left( I_n \otimes DFT_n \right) \]
    - \[ = \prod_{i=0}^{1} \left( I_{n_i^2} \left( I_n \otimes DFT_n \right) I_{n_i^2} \right) \]
    - \[ = \prod_{i=0}^{1} \left( \left( L_{n_i^2} / k \otimes I_k \right) \left( I_{n_i / k} \otimes L_{n_i} / k \right) \left( I_{n_i / k} \otimes \left( I_k \otimes DFT_n \right) \right) \right) \]
  - **Row-column algorithm**
  - **Symmetric algorithm**
  - **Symmetric algorithm with tiling**
  - Write tiles column-wise
  - Transpose and re-tile on-chip
  - FFT processing
  - Linearize on-chip
  - Read tiles row-wise

- [Akin, et al., FCCM 2012]
Breadth First Search

Large graph has more than millions of nodes with may be handful edges per node

```java
foreach (node n in graph) n.dist=∞;

worklist = {root}; root.dist=0;

foreach (node n in worklist) {
    foreach (neighbor of n) {
        if (n.dist + 1 < neighbor.dist) {
            neighbor.dist = n.dist + 1;
            add neighbor to worklist;
        }
    }
}
```

(see http://iss.ices.utexas.edu/?p=projects/galois/benchmarks/bread_first_search)
Compressed Sparse Row (CSR) Adjacency Matrix

<table>
<thead>
<tr>
<th>SRC</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- **dest→** column
- **0** entry per-edge
- **1** entry per-node

Array indexed by row/src idx (holds offset into element array)

```
0 2 3 5
```

Array of all non-0 elements in row-order (holds col/dest index)

```
1 3 0 1 3 2
```

Large graph has more than millions of nodes with may be handful edges per node

Reference:
real code showing memory accesses

```cpp
while(wl.mHowmany) { // worklist not empty
    // repeat for each node on frontier
    int curr=wl.mList[wl.mDeq]; // S0
    int myDist=graph->mPerNode[curr].dist; // S1
    int numEdges=graph->mPerNode[curr].fanout; // S1
    int scan=graph->mPerNode[curr].edges; // S1
    { ... dequeue from worklist ...}
    while (numEdges--) {
        // repeat for each neighbor
        int dest=graph->mPerEdge[scan].dest; // S2
        int destDist=graph->mPerNode[dest].dist; // S3
        if ((myDist+1)<destDist) { // S4
            graph->mPerNode[dest].dist=myDist+1; // S4
            { ...enqueue dest to worklist...} // S5
        }
        scan++;
    }
}
```
Elastic HW Processing Pipeline

BFS Irregular Access Pattern

- Irregular and graph dependent
  - **S0 read worklist**: spatial locality, non-temporal
  - **S1 read node array (self)**: no locality
  - **S2 read edge array**: some spatial locality, non-temporal
  - **S3 read node array (neighbor)**: no locality
  - **S4 write node array (neighbor)**: temporal with S3
  - **S5 write worklist**: spatial locality, non-temporal

- S3 most problematic of all
  - S1 and S3 lack locality but S3 repeated per neighbor
  - same number of S2 and S3 but S2 has spatial locality
  - hard to fix short of caching all nodes on-chip
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?

18-643 Review Template: Summary

- What to look for in a paper/presentation
  - what is the question/problem?
  - why is this question/problem important?
  - why is this question/problem hard?
  - what is the answer/solution offered by the paper?
  - what is new, novel about the answer/solution?
  - how does the paper argue/support the answer/solution is correct/good?
- If you don’t know the answers, you didn’t “read” the paper, OR, the paper/presentation is bad
18-643 Review Template: Critic

• What to think about while reading
  – soundness: is the paper's answer/solution correct/good?
  – impact: is the paper's answer/solution important?
  – novelty: does the paper teach something new and not obvious?
  – strengths: what makes the paper stand out?
  – weaknesses: what could be improved?

• If you don’t know the answers, you didn’t “read” the paper
  Don’t accept what a paper says unless you agree