(Review)
Improving Memory Performance in a DRAM

- Increasingly important because fewer chips/system

Evolutionary
- Fast page mode
  - Allow multiple CAS accesses without need for intervening RAS
    - Optimizes sequential access, exploiting the row buffer (1024-2048 bits)
  - Extended Data Out (EDO): 30% faster in page mode
- Synchronous DRAM (SDRAM)
  - Avoid need for handshaking between chip and memory controller
  - Chip also has a register with number of requested bytes; these are transmitted without explicit requests from controller
- Double Data Rate (DDR) DRAM
  - Transmit data from chip on both the falling and rising edge of clock signal

A New DRAM Interface: RAMBUS

Outline

- Announcements
  - Assignment #4 due back November 27th
  - Final exam on December 11th: Room 402, WWH

- This lecture
  - Main memory organizations (cont’d)
    - RAMBUS
  - Multithreading and Multiprocessors
    - Convergence architecture

[Hennessy/Patterson CA:AQA (3rd Edition): Chapters 5, 6]
[Culler/Singh/Gupta, Parallel Computer Architecture: A Hardware/Software Approach: Chapter 1]
**RAMBUS Details**

- **First-generation interface**
  - Dropped RAS/CAS, replacing it with a packet-switched bus
  - Can multiplex address/data on this bus
  - Each chip acts as a memory bank (internally, 4 banks of memory cells)
  - Can transfer variable amount of data from a single request
  - Can perform its own refresh
  - Transfers data on both edges of its clock

- **Second-generation interface**
  - Separate row- and column-command buses, plus 18-bit data bus
  - Can perform 3 transactions simultaneously
  - 16/32 internal banks/chip

- RAMBUS improves bandwidth (at cost of latency, price premium)
  - Nintendo game machines (where number of chips is a restriction)

**Internal Organization of a RAMBUS DRAM Chip**

- Large number of internal banks
  - Shared row-buffers, so don’t get full parallelism
- Multiplexed data buses
- Packet-switched
- 400 MHz clock signal
  - Data on both rising and falling edge
  - 2 bytes/cycle x 2
    - Data A and B
  - 1.6 GB/s (1 chip)
- Compared to 800-1200 MB/s (DIMM)

**SDRAM vs. RAMBUS: Latency vs. Bandwidth**

**Multithreading and Multiprocessors**
Parallel Processing

- So far: focused on performance of a single instruction stream
  - ILP exploits parallelism among the instructions of this stream
  - Needs to resolve control, data, and memory dependences

- How do we get further improvements in performance?
  - Exploit parallelism among multiple instruction streams
    - Multithreading: Streams run on one CPU
      - Typically, share resources such as functional units, caches, etc.
      - Per-thread register set
    - Multiprocessing: Streams run on multiple CPUs
      - Each CPU can itself be multithreaded
      - Common issues:
        - synchronization between threads
        - consistency of data in caches (more generally, communication)

- We shall start off by talking about multiprocessing

A Taxonomy of Parallel Architectures

As per Flynn (1966)

- Single instruction stream, single data stream (SISD)
  - Uniprocessor
- Single instruction stream, multiple data stream (SIMD)
  - MMX, ISSE extensions discussed earlier are a simple form of this
  - Vector processors, data parallel machines
- Multiple instruction streams, single data stream (MISD)
  - ??
- Multiple instruction streams, multiple data streams (MIMD)
  - Each processor fetches its own instructions and operates on own data
  - Currently the most common
    - Flexibility
    - Cost-performance advantage because uses off-the-shelf microprocessors as building blocks

Parallel Architectures: Then and Now

- Historically, parallel architectures tied to programming models
  - divergent architectures, with no predictable pattern of growth

- Today: Consensus on a layered communication architecture
  - node architecture + support for communication and cooperation

Modern Layered Framework

<table>
<thead>
<tr>
<th>CAD</th>
<th>Database</th>
<th>Scientific modeling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiprogramming</td>
<td>Shared address</td>
<td>Message passing</td>
</tr>
<tr>
<td>Compilation or library</td>
<td>Operating systems support</td>
<td></td>
</tr>
</tbody>
</table>

Communication abstraction
User/system boundary
Hardware/Software boundary

Physical communication medium
Modern Layered Framework

- **Parallel applications**
  - CAD
  - Database
  - Scientific modeling

- **Programming models**
  - Multiprogramming
  - Shared address
  - Message passing
  - Data parallel

- **Communication abstraction**
  - Compilation or library
  - Operating systems support

- **User/system boundary**
- **Hardware/Software boundary**

**Physical communication medium**

- **Parallel applications**
- **Programming models**
- **Communication abstraction**
- **User/system boundary**
- **Hardware/Software boundary**

- **Modern Layered Framework**

  - **Communication abstraction: primitives for implementing the model**
    - supported directly by HW, by the OS, or by user-level software
    - today: convergence in organizations
      - HW/SW interface is relatively flat
      - compiler/software plays important bridging role

Evolution of Architectural Models

- **Rationale**
  - historically: machines tailored to programming models
    - architecture = prog. model + comm. abstraction + machine organization
  - now: separation of programming models and architectures
    - tracing the evolution helps identify core concepts, understand convergence

- **Five architectures of interest**
  - **dominant**: shared address space, message passing, data parallel
  - **others**: dataflow, systolic arrays

- **For each, let us examine**
  - programming model
  - motivation
  - intended applications
  - contributions to convergence

(1) Shared Address Space Architectures

- **Programming model**
  - process: virtual address space plus one or more threads of control
  - portions of address spaces of processes are shared
  - writes to shared address visible to all threads (in other processes as well)
Shared Address Space Architectures (cont’d)

- Motivation: Programming convenience
  - location transparency
    - communication is implicitly initiated by loads and stores
  - similar programming model to time-sharing on uniprocessors

- Communication hardware also natural extension of uniprocessor
  - addition of processors similar to memory modules, I/O controllers

Evolution: Four Organizations

- Mainframes
  - motivated by multiprogramming
  - extends crossbar for memory modules and I/O
    - initially, limited by processor cost
    - later, by cost of crossbar
    - high incremental cost
    - e.g., IBM S/390

- Minicomputers (SMPs)
  - motivated by multiprogramming, transaction processing
    - all components on a shared bus
      - latency larger than for uniprocessor
      - bus is bandwidth bottleneck
      - caching is key: coherence problem
    - low incremental cost

Example of an SMP: Intel Pentium Pro Quad

- All coherence and multiprocessing glue in processor module
- Highly integrated, targeted at high volume
- Low latency and bandwidth

Evolution: Four Organizations (contd.)

- Dance Hall
  - problem: interconnect cost (crossbar), or bandwidth (bus)
  - solution: scalable interconnection network
    - bandwidth scalable
    - however, larger access latencies
    - caching is key: coherence problem
  - e.g., NYU Ultracomputer

- Distributed Memory (NUMA)
  - message transactions across a general-purpose network
    - e.g. read-request, read-response
    - caching of non-local data is key
    - coherence costs
  - e.g., Cray T3E, Origin 2000
Example of a NUMA: Cray T3E

- Scales up to 1024 processors, 480MB/s links
- Nonlocal references accessed using communication requests
  - generated automatically by the memory controller
  - no hardware coherence mechanism (unlike SGI Origin)

(2) Message Passing Architectures

- Programming model
  - directly access only private address space (local memory), communicate via explicit messages (send/receive)
  - in simplest form, achieves pair-wise synchronization
  - model is decoupled from basic hardware operations
    - library or OS intervention for copying, buffer management, protection

Message Passing Architectures (cont’d)

- Complete computer as building block, including I/O
  - communication via explicit I/O operations

- High-level block diagram similar to distributed-memory shared address space machines

  - but communication integrated at IO level, needn’t be into memory system
  - like networks of workstations (clusters), but tighter integration
  - easier to build than scalable shared address space machines

Example of a Message Passing Machine: IBM SP-2

- Made out of essentially complete RS6000 workstations
- Network interface integrated in I/O bus (bandwidth limited by I/O bus)
Evolution of Message-Passing Machines

- Early machines: FIFO on each link
  - HW close to programming model
    • synchronous operations
  - replaced by DMA
    • enables non-blocking operations
    • buffered by system at destination

- Today: diminishing role of topology
  - topology important for store-and-forward routing
  - introduction of pipelined (cut-through) routing made it less so
    • Virtual cut through, wormhole routing
  - cost is in node-network interface

Towards Architectural Convergence

- Evolution and role of software have blurred boundary between programming models
  - send/recv on shared address space (SAS) architectures: buffers
  - global address space on message passing (MP) architectures
    • hashing and page-based (or finer-grained) shared virtual memory
- Convergence in hardware organizations as well
  - tighter network-interface integration even for message passing
  - hardware shared address-space passes messages at lower level
  - even clusters of workstations/SMPs are parallel systems
    • emergence of fast system area networks (SAN)

- Programming models are distinct, but organizations have converged
  - nodes connected by general network and communication assists
  - implementations also converging, at least in high-end machines

(3) Data Parallel Systems

- Programming model
  - operations performed in parallel on each element of data structure
  - logically single thread of control, performs sequential or parallel steps
    • conceptually, a processor associated with each data element

- Architectural model
  - array of many simple cheap processors, each with little memory
  - a control processor issues instructions
  - specialized and general communication, cheap global synchronization

- Original motivations
  - matches simple differential equation solvers
  - centralize high cost of instruction fetch/sequencing

Applications of Data Parallelism

- Each PE contains an employee record with his/her salary
  if salary > 100K then
    salary = salary * 1.05
  else
    salary = salary * 1.10
  - logically, the whole operation is a single step
  - some processors enabled for arithmetic operation, others disabled

- Other examples
  - finite differences, linear algebra
  - document searching, graphics, image processing

- Popular architecture in the late 1980s and early 1990s:
  - Thinking Machines CM-1, CM-2 (and CM-5)
  - Maspar MP-1 and MP-2
Evolution and Convergence

- Rigid control structure popular in the 1960s
  - cost savings of centralized sequencer high
  - Flynn taxonomy: SIMD
- Replaced by vectors in mid-70s
  - more flexible w.r.t. memory layout and easier to manage
- Revived in mid-80s
  - when only 32-bit datapath slices would fit on chip
- Other reasons for demise
  - simple, regular applications have good locality, can do well anyway
  - hardwiring data parallelism limits applications
- Lasting contributions
  - programming model converges with SPMD (single program multiple data)
  - need for fast global synchronization, structured global address space

(4) Dataflow Architectures

- Represent computation as a graph of essential dependences
  - logical processor at each node, activated by availability of operands
    - message (token) carrying tag of next instruction sent to next processor
    - tag compared with others in matching store; match fires execution

(5) Systolic Architectures

- Replace single processor with array of regular processing elements
  - orchestrate data flow for high throughput with less memory access

- Differences from other organizations
  - pipelining: nonlinear array structure, multi-directional data flow, each PE may have (small) local instruction and data memory
  - SIMD: each PE may do something different

- Original motivations
  - VLSI enables inexpensive special-purpose chips
  - represent algorithms directly by chips connected in regular pattern
Systolic Architectures (contd.)

1. Example realization: iWARP
   - Uses quite general processors: variety of algorithms on same hardware
   - But dedicated interconnect channels: register-to-register data transfer

2. Specialized, and ran into same problems as SIMD
   - General purpose systems work well for same algorithms (locality etc.)
   - Current day manifestation: Embedded digital signal processors (DSPs)

Convergence: Generic Parallel Architecture

1. A generic modern multiprocessor
   - Node: processor(s), memory system, plus communication assist
   - Network interface and communication controller
   - Scalable network

   Convergence allows lots of innovation, now within framework
   - Integration of assist with node, what operations, how efficiently...

Natural Extensions of Memory System

Small-scale Symmetric Multiprocessors
**Shared Memory Multiprocessors**

- Symmetric multiprocessors (SMPs)
  - uniform access to all of main memory from any processor
- Dominates the server market
  - building blocks for larger systems
  - arriving to desktop
- Attractive for both parallel programs and throughput servers
  - fine-grain resource sharing
  - automatic data movement and coherent replication in caches

- Uniform access via loads and stores
  - private caches reduce access latency, bandwidth demands on bus
  - however, introduce the cache coherency problem
    - values in different caches need to be kept consistent

**The Cache Coherence Problem**

- Processors see stale values
  - with write-back caches, value written back to memory depends on which cache flushes or writes back value (and when)
  - clearly not a desirable situation!

---

**So What Should Happen?**

- Intuition for a coherent memory system
  *reading a location should return latest value written (by any process)*
- What does latest mean?
  - several alternatives (even on uniprocessors)
    - source program order, program issue order, order of completion, etc.
  - how to make sense of order among multiple processes?
    - must define a meaningful semantics
- Is cache coherence a problem on uniprocessors?
  - Yes!
    - interaction between caches and I/O devices
      - infrequent software solutions work well
        - uncacheable memory, flush pages, route I/O through caches
      - however, the problem is performance-critical in multiprocessors
        - needs to be treated as a basic hardware design issue