18-547 Lecture 23
MMX & I/O
Instructor: Prof. Falsafi

Electrical & Computer Engineering
Carnegie Mellon University

Slides developed by Profs. Hill, Wood, Sohi, and Smith of
University of Wisconsin — Madison

Announcements

Project due tomorrow

Final exam: Fri. December 14 5:30pm-8:30pm, PH 125C

Exam:
- Mostly from after mid-term
- Categorized like midterm
  1. Concepts
  2. Reader
  3. Quantitative
  4. Design
**MMX**

Peleg & Weiser, IEEE Micro, August 1996

Goal: 2x performance in multimedia (audio, video, etc.)

Key technique: Single Instruction Multiple Data (SIMD)

- used in the past to build large-scale supercomputers

One instruction computes multiple data simultaneously

---

**MMX: Intro**

Most multimedia apps work on short integers

Pack data 8-bit or 16-bit data into 64-bit words

Operate on packed data like short vectors

- SIMD (around since Livermore S-1, 20 years ago!)

Integrate into x86 FP registers

Can improve performance by 8x (in theory)

- in reality performance is typically better than 2

- gives people another reason to think their PC is obsolete!
**MMX: Datatypes**

Native datatypes smaller than usual
- e.g., 8-bit pixels, 16-bit audio samples
Datatypes packed or compressed into 64 bits
- 1x64-bit quad-words
- 2x32-bit double-words
- 4x16-bit words
- 8x8-bit bytes

**MMX: Enhanced Instructions**

Saturating arithmetic \((100 + 200) \mod 256 = 44\)
Also MOV’s
- move MMX datatypes to and from memory
- loads followed by stores
Pack/Unpack
- go back and forth between MMX and normal datatypes
Needed by multimedia computations
**MMX: Performance Example**

16 element dot product

\[ [a_1 \ a_2 \ a_3 \ldots \ a_{16}] \times [b_1 \ b_2 \ b_3 \ldots \ b_{16}] = a_{1x}b_1 + \ldots + a_{16}b_{16} \]

Intel IA + few optimizations (compiler + hardware)

- 32 loads
- 16 multiplies
- 15 adds
- 12 loop control
- 72 instructions

MMX using 16-bit data: 16 instructions

---

**MMX Constraints: Adding to Pentium**

Share FP registers

- ISA extensions but perfect backward compatibility
- 100% OS compatible (no extra regs, flags, exceptions)

Bit in CPUID instruction

- so applications test for MMX and include code

Use 64-bit datapaths

Pipeline capable of 2 MMX IPC

Cascade memory and execution stages to avoid stalls
MMX and Vectors

No Vector Length
Memory load/store: stride of one
Arithmetic integer only
Conditionals: builds byte masks like vector masks
Data movement: pack/unpack like vector scatter/gather

I/O

Who cares about I/O?
What to consider about I/O?
Latency & throughput
Device characteristics and types
Some important devices: disks, graphics displays, networks
I/O system architecture: buses, I/O processors
High performance disk architectures
Why Care about I/O?

Amdahl's law
- if you only speed up CPU then I/O becomes the bottleneck
- e.g.,
  - suppose I/O takes 10% of the time
  - speed up CPU 10 times
  - system only speeds up by 5 times

Latency vs. Throughput

“There is an old saying: bandwidth problems can be cured with money. Latency problems are harder because the speed of light is fixed. You can’t bribe God.” - David Clark

Throughput:
- bandwidth
- I/Os per second

Latency:
- response time
Latency vs. Throughput

Who cares about latency?
- Care about throughput => context switch
- Fallacy
  - requires more memory
  - requires more processes
  - human productivity increases super-linearly with response time

I/O Overlap

I/O overlaps with computation in complicated ways

Time

User1  User2  User1

OS  OS  OS

I/O  I/O  I/O
**I/O Performance**

\[ \text{Time}_{\text{job}} = \text{time}_{\text{cpu}} + \text{time}_{\text{I/O}} - \text{time}_{\text{overlap}} \]

e.g.,

\[ 10 = 10 + 4 - 4 \]

Speed up CPU by 2x

What is \( \text{time}_{\text{job}} \)?

\[ \begin{align*}
\text{time}_{\text{job}} &= 5 + 4 - 4 = 5 \text{ (best)} \\
\text{time}_{\text{job}} &= 5 + 4 - 0 = 9 \text{ (worst)} \\
\text{time}_{\text{job}} &= 5 + 4 - 2 = 7 \text{ (average?)}
\end{align*} \]

**I/O Characteristics**

Supercomputers
- data transfer rate important
- many MBs per second for large files

Transaction processing
- I/O rate important
- “random” accesses
- disk I/Os per second

Time sharing filesystems
- small files
- sequential accesses
  - many creates/deletes
Device Characteristics

Behavior
- input - read once
- output - write once
- storage - read many times, usually write

Partner
- human
- machine

Data rate
- peak transfer rate

<table>
<thead>
<tr>
<th>Device</th>
<th>I or O?</th>
<th>Partner</th>
<th>Data Rate KB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse</td>
<td>I</td>
<td>Human</td>
<td>0.01</td>
</tr>
<tr>
<td>Graphics display</td>
<td>O</td>
<td>Human</td>
<td>60,000</td>
</tr>
<tr>
<td>Modem</td>
<td>I/O</td>
<td>Machine</td>
<td>2-8</td>
</tr>
<tr>
<td>LAN</td>
<td>I/O</td>
<td>Machine</td>
<td>500-6000</td>
</tr>
<tr>
<td>Tape</td>
<td>Storage</td>
<td>Machine</td>
<td>2000</td>
</tr>
<tr>
<td>Disk</td>
<td>Storage</td>
<td>Machine</td>
<td>2000-10000</td>
</tr>
</tbody>
</table>
Disk Operations

Seek: move head to track

- avg. seek time = \( \frac{\sum_{1}^{n} \text{seek}(l)}{n} \)
- \( n \) is # tracks, seek\((l)\) is time to seek lth track

Rotational latency: wait for sector

- avg. rotational latency \( 0.5/3600 = 8.3\text{ms} \)

Transfer rate

- typically 1-4 MB per second

Overhead

- controller delay
- queueing delay

Disk Performance

Avg disk access =

- avg seek time + avg rot. Delay + transfer + ovhd

\[ e.g., \ \\ 3600 \text{ rpm}, 2 \text{ MB/s} \ \\ \text{avg. seek time: 9ms} \ \\ \text{controller overhead: 1ms} \ \\ \text{read 512-byte sector} \ \\ 9\text{ms} + 0.5/3600 + 0.5\text{KB}/2 + 1 = 18.6\text{ms} \]
Alternatives to Disks

DRAMS

- SSD (solid-state disk)
  - standard disk interface
  - DRAM and battery backup
- ES (expanded storage)
  - software controlled cache
  - large (4K) blocks
  ✓ no seek time
  ✓ fast transfer rate
  ☹ expensive

Alternatives to Disks (Cont.)

FLASH memory

✓ no seek time
✓ fast transfer
✓ non-volatile
☹ bulk erase before write
☹ slow writes
☹ “wears” out over time
**Optical Disks**

- **Read-only**
  - CD-ROM
  - cheap & reliable
  - slow
- **Write once**
  - not so cheap
  - slow
- **Write many**
  - expensive
  - slow

Screen has many scan lines each of which has many pixels
Phosphorous acts as capacitor: refresh 30-60 times/second
**Graphics Display: Frame Buffer**

![Diagram of CPU, Memory, Frame Buffer, CRT with data rates](image)

**Frame Buffer (Cont.)**

Frame buffer stores bitmap

- One entry per pixel
  - black: 1 bit per pixel
  - gray-scale: 4-8 bits per pixel
  - color (RGB): 8 bits per color (24 bits)

Typical size 1560x1280 pixels

- Black and white: 250KB
- Color (RGB): 5.7 MB
Reducing Frame Buffer Cost

Key idea: only a small number of colors are used in one image

Color map:
- a table of pixel color values
- frame buffer stores table idx instead

Networks

Terminal networks
- machine-terminal
- star: point-to-point
- same room
- 0.3-19 Kbits
- RS232 protocol

LANs
- machine-machine
- bus, ring, star
- < 10 Km
- 0.1-100 Mbits/s
- ethernet
Networks (Cont.)

WAN
- machine-machine
- irregular structure: point-to-point
- > 10 Km
- 50-2000 Kbits/s
- Internet

LAN

E.g., ethernet
- one-write bus with collisions
- exponential backoff
- within building
- 10 Mb

Now ethernet is
- point-to-point to clients (switched network)
- with hubs
- clients s/w unchanged
- 100 Mb
WAN

E.g., ARPANET, Internet
- arranged as a DAG
- backbones now 1Gb/s
- 100 Gb/s in the future

TCP/IP protocol stack
- transmission control protocol
- Internet protocol

Key issues:
- Top-to-bottom system issues
- getting net into homes
- cable modem, ISDN, ADSL, etc.

I/O System Architecture

Hierarchical data paths
- divides bandwidth going down hierarchy
- often buses at each level

I/O processing
- program controlled
- DMA
- dedicated I/O processors
I/O System Architecture

Buses

<table>
<thead>
<tr>
<th>Option</th>
<th>High Performance</th>
<th>Low cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address/data lines separate?</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data lines</td>
<td>Wider</td>
<td>Narrower</td>
</tr>
<tr>
<td>Transfer size</td>
<td>Multiple words</td>
<td>Single word</td>
</tr>
<tr>
<td>Bus masters</td>
<td>Multiple</td>
<td>One</td>
</tr>
<tr>
<td>Split transaction</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Clocking</td>
<td>Synchronous</td>
<td>Asynchronous</td>
</tr>
</tbody>
</table>
I/O to CPU Interface

Bus Arbitration

Centralized star connection
- high cost
- high performance

Daisy chain
- cheap
- low performance

Distributed arbitration
- medium price/performance

Arbitration for next bus mastership overlap with current transfer
Bus Switching Methods

Circuit-switched buses
- held until request is complete
- simple protocol
- device latency affects bus utilization

Split-transaction or pipelined
- bus is released after request is initiated
- other of the bus until reply comes back
- Pipelined (packet-switched): reply comes back at a fixed time
- Split-transaction: reply must arbitrate
- complex bus protocol
- better utilization of bus

Standard I/O Buses

Param

<table>
<thead>
<tr>
<th>Param</th>
<th>S-bus</th>
<th>Micro Channel</th>
<th>PCI</th>
<th>SCSI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data width (bits)</td>
<td>32</td>
<td>32</td>
<td>32-64</td>
<td>8-16</td>
</tr>
<tr>
<td>Clock (Mhz)</td>
<td>16-25</td>
<td>Asynch</td>
<td>33</td>
<td>10/async</td>
</tr>
<tr>
<td>#masters</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>B/w, 32-bit read (MB/s)</td>
<td>33</td>
<td>20</td>
<td>33</td>
<td>20 or 6</td>
</tr>
<tr>
<td>B/w, peak (MB/s)</td>
<td>89</td>
<td>75</td>
<td>111</td>
<td>20 or 6</td>
</tr>
</tbody>
</table>
Memory Buses

<table>
<thead>
<tr>
<th>Param</th>
<th>HP Summit</th>
<th>SGI Challenge</th>
<th>Sun XDbus</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data width (bits)</td>
<td>128</td>
<td>256</td>
<td>144</td>
</tr>
<tr>
<td>Clock (Mhz)</td>
<td>60</td>
<td>48</td>
<td>66</td>
</tr>
<tr>
<td>#masters</td>
<td>Multiple</td>
<td>Multiple</td>
<td>Multiple</td>
</tr>
<tr>
<td>B/w, peak (MB/s)</td>
<td>960</td>
<td>1200</td>
<td>1056</td>
</tr>
</tbody>
</table>

Disk Arrays

Collection of individual disks
- each disk has its own arm/head

Data distributions

- Independent
- Fine-grain
- Coarse-grain
Disk Arrays (Cont.)

Independent addressing
- s/w user distribute data
- load balancing an issue

Fine-grain striping
- one bit, one byte, one sector
- #disks x stripe unit evenly divides smallest accessible data
- perfect load balance; only one request served at a time
- effective transfer rate approx. N times better than single disk
- access time can go up, unless synchronized disks

Coarse-grain striping
- data transfer parallelism for large requests
- concurrency for small requests
- load balanced by statistical randomization

Must consider workload to determine strip size

Redundancy Mechanisms

Disk failures are a significant fraction of hardware failures
- striping increases #corrupted files per failure

Data replication
- disk mirroring
- allow multiple reads
- writes must be synchronized

Parity protection
- use a parity disk
Redundant Array of Inexpensive Disks

RAID (Berkeley)
- arrays of small cheap disks provide high performance/reliability
  
  $D = \# \text{data disks}, \; C = \# \text{check disks}$

Level1: mirrored disks ($D = 1, \; C = 1$)
- overhead too high

Level2: bit interleaved array (e.g., $D=10, \; C=4$)
  - like ECC for DRAMs
  - read all bits across groups
  - merge update bits with bits not updated
  - recompute parity
  - rewrite full group including checks

RAID (Cont.)

Level3: hard failure detection and parity (e.g., $D = 4, \; C = 1$)
  - key: failed disk is easily identified by controller
  - no need for special code to identify failed disk

Level4: intra-group parallelism
  - coarse-grain striping
  - like level-3 + ability to do more than one small I/O at a time
  - write must update disk with data and parity disk
RAID (Cont.)

Level 5: rotated parity to parallelize writes
- parity spread out across disks in a group
- different updates of parities go to different disks

Level 6: two-dimensional array
- array of data is a two-dimensional
- with row and column parities

I/O Performance

Think of I/O as a queueing system
Requests enter the queue at a certain rate
Wait for service
Service takes certain time
Requests leave the system at a certain rate
**Little's Law**

Assume steady state => arrival rate == departure rate

Little's law:
- rate = avg # in system/avg response time
- applies to any queue in equilibrium

---

**I/O Performance (Cont.)**

Total time in system = time in queue + time in service
Total time is response time: that is what matters
Service rate = 1/time to serve
Population = queue length + avg. # of jobs in service
Utilization = arrival rate/service rate
Note that Little's law can be applied to individual components
- server: # in server = arrival rate x time in service
- queue: queue length = arrival rate x time in queue
**FIFO Queues**

Time in system = q length x service + residual service

Residual service time
- depends on probability distribution of service time

Avg residual service time = 1/2 x mean x (1 + CV^2)
- CV is the coefficient of variance = (variance/mean^2)^{1/2}
- Variance = E(X^2) - E(X)^2

CV^2 = 1 exponential distribution
CV^2 > 1 hyperexponential distribution
CV^2 < 1 hypoexponential distribution
constant => memoryless

---

**M/M/1 Queues: Exponential Distribution**

Time in q = q length x service + util x avg. residual service

Time in q = (service x (1 + CV^2) x util)/(2 x (1 - util))

if CV^2 = 1
- time in q = service x (util/(1 - util))
- which is why util should not get too high
Unix File System Performance

Cache files in memory
- memory much faster than disks

File cache is key
- OS parameters, cache size, write policy
- asynchronous writes => processor continues
- coherence in client/server systems