CSC266 Introduction to Parallel Computing using GPUs
Understanding Memory Performance

Sreepathi Pai
September 27, 2017
URCS
Outline

Introduction

Caches

Performance of Caches
Introduction

Caches

Performance of Caches
Matrix Multiply – IJK

- Multiplying two matrices:
  - A \( (m \times n) \)
  - B \( (n \times k) \)
  - C \( (m \times k) \) [result]

- Here: \( m = n = k \)

```cpp
for(ii = 0; ii < m; ii++)
    for(jj = 0; jj < n; jj++)
        for(kk = 0; kk < k; kk++)
            C[ii * k + kk] += A[ii * n + jj] * B[jj * k + kk];
```
for(ii = 0; ii < m; ii++)
    for(kk = 0; kk < k; kk++)
        for(jj = 0; jj < n; jj++)
            C[ii * k + kk] += A[ii * n + jj] * B[jj * k + kk];
Performance of the two versions?

- on 1024x1024 matrices of ints
- which is faster?
- by how much?
Performance of the two versions

• on 1024x1024 matrices
• Time for IJK: 0.554 s ± 0.003s (95% CI)
• Time for IKJ: 6.618 s ± 0.032s (95% CI)
What caused the nearly 12X slowdown?

- Matrix Multiply has a large number of arithmetic operations
  - But the number of operations did not change
- Matrix Multiply also refers to a large number of array elements
  - Order in which they access elements changed
  - But why should this matter?
Die shot of a processor (IBM Power 8)
Die shot of a processor (IBM Power 8)
Introduction

Caches

Performance of Caches
What are caches?
Caches are a kind of fast(er) memory.
Why don’t we build entire memory systems out of “cache memory”?

*see also:* joke about black boxes in aeroplanes.
Physical Issues

- Not all memory types are equal
  - Consider: SRAM, DRAM and magnetic storage
- Speed to access data
  - Depends on size and type of memory
  - SRAM > DRAM > Magnetic storage
- Density of storing data
  - Bits per square millimeter
  - SRAM < DRAM < Magnetic storage
The Memory Hierarchy – Part I

- Registers
  - managed by compiler
  - “logic”

- L1 cache
  - small (10s KB), usually 1-cycle access
  - SRAM (also “logic”)

- L2 cache
  - largish (100s KB), 10s of cycles
  - SRAM

- ...

...
• L3 cache
  • usually on multicores
  • much larger (MB), 100s of cycles
  • SRAM or (recently) embedded DRAM
• DRAM
  • off-chip, large (GB)
• HDD
  • Magnetic/Rotating Storage (TBs)
  • Flash memory (GBs)
Performance of the hierarchy?

Why structure memory in a hierarchy?

- Each level of hierarchy adds a delay
- Time to access memory increases!
  - Or does it?
Performance of the hierarchy

- Structures in memory hierarchy duplicate data stored further away
  - original meaning of the word *cache*
- If data is found at closer to processor (i.e. *hit*), read it from there
- Otherwise (i.e. *miss*), pass request one level up the hierarchy
Why the hierarchy works in practice

- Data Reuse (or “locality”)
  - Temporal (same data will be referred again)
  - Spatial (data close to each other in space will be referred close to each other in time)

- Speed differences
  - Time to access L1: 1ns
  - Branch mispredict: 3ns
  - Time to access L2: 4ns
  - Main memory access time: 100ns
  - SSD access time: 16μs
  - Rotating media access time: < 5 ms
  - From Latency Numbers Every Programmer Should Know
The cache equation (informal)

Assume a one-level cache (i.e. cache + RAM):

\[ \text{latency} = \text{latency}_{\text{hit}} \]

or

\[ \text{latency} = \text{latency}_{\text{miss}} \]
The cache equation for one level of caches

\[ \text{latency}_{\text{avg}} = (\text{fraction}_{\text{hit}}) \times \text{latency}_{\text{hit}} + (1 - \text{fraction}_{\text{hit}}) \times \text{latency}_{\text{miss}} \]
Outline

Introduction

Caches

Performance of Caches
Cache Organization

- RAM is directly addressable
- Caches duplicate RAM contents
  - accept same addresses
  - translate that address internally
- Translation is a many-to-one function
  - obviously, since caches are much smaller than RAM
- Therefore caches store:
  - data
  - “tag” (original address or part of original address)
  - tag is used to verify data address
Cache Lookup

- Building blocks of translation functions:
  - Direct Mapped
  - Associative Lookup
A Direct Mapped Cache

- Converts data address to cache location

\[ \text{index} = \text{fn}(\text{address}) \]

<table>
<thead>
<tr>
<th></th>
<th>tag</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>n</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Fully Associative Cache

- Searches for address in cache
- also known as content-addressable memory
Set Associative Caches

- 4-way set-associative cache
- Lookup a set based on address (direct-mapped)
- Lookup address only within the set (associative)
The 3C model of cache misses

- Compulsory (or Cold) misses
  - first reference to data
  - always occur (?)
- Capacity misses
  - data in cache is “evicted” once cache is full
  - miss to data being evicted from cache
  - these are absent in an infinite cache
- Conflict misses
  - miss due to many-to-one conflict
  - two different addresses map to the same cache address
  - these do not occur in a fully associative cache

Due to Mark Hill
Reducing misses using programming techniques
Summary

- Memory accesses are key to program performance
  - nearly always the bottleneck in most programs
- The memory hierarchy lowers memory access latency
  - Exploits “locality” of references
  - Size/speed tradeoffs
- Caches are organized differently than RAM
  - smaller
  - implications for performance
  - transparent to programmer
  - not transparent to performance!