Spinning your Wheels: CPU Time vs Instructions

John Baker MVS Solutions jbaker@mvssol.com
Claire Cates SAS Institute claire.cates@sas.com
Agenda

- Memory vs CPU?
  - Why is my application/system so slow?
- Fundamentals of a memory cache
- What are the different layouts across host platforms
- What can we DO about my system or application performance?
  - Practical actions to improve performance
- Q/A
Memory Evolution

Why do I care?

- Memory is cheap and plentiful (up to 10 TB on the z13)
- CPUs are fast (5 billion cycles per second)
- CPU time is my main concern
CPU/Memory Performance

The Memory Parking Lot

Main Mem

L4
L3
L2
L1

Your car

You
All CPUs wait at the same speed
The Meter is running but You’re not Moving
Raw CPU Speed no longer the defining performance factor
Memory Cache

- Used by the CPU to reduce the memory latency
- A section of Memory closer to the CPU
- Stores frequently used memory

- Design assumptions for the cache.
  - Data that is accessed once will more than likely be accessed again
  - When memory is accessed, memory near that location will be accessed.
Memory Cache

- Instruction Cache - used for executable instructions
- Data cache - used to speed up data fetch and store
  - L1 (level 1) - closest cache to the CPU - fastest - smaller
  - L2 (level 2) - if data is not in the L1 cache - slower than L1 but faster than main memory, larger than L1.
  - L1 - L2 ... caches may be shared on multi-core systems
  - Many systems now have L3 and L4 caches
Memory Cache Terms
Memory Terms

- **Latency**
  - The delay to access the memory.
  - Usually measured in clock cycles to return the requested data.
  - The slower the latency, the slower your program runs.

- **Bandwidth**
  - The pipeline carrying the memory from main memory to the processor.
  - If you saturate the pipeline, performance will be impacted.
Memory Cache Terms

- **Cache Line**
  
  Data is copied from main memory in a fixed size area. Typically 64 bytes long. Cache lines will be copied into main memory to satisfy the data request. Multiple cache lines may be copied.

- **Cache Hit**
  
  Data is found in the cache

- **Cache Miss**
  
  The data is not found in the cache. The CPU will need to load it from a higher level cache or main memory. You want to avoid Cache Misses.
Memory Cache Terms

- **Dirty Cache Line**
  
  When data is written to memory it needs to eventually be written back to main memory. It is dirty, if the contents have not been written back.

- **Write-Back Policy**
  
  The policy the CPU uses to determine when to write the dirty cache lines back to main memory.

- **Cache Coherence**
  
  Multiple CPU caches have a private copy of the same piece of memory.

  The process of making sure each of these copies have the updated “correct” content.
• Dirty Cache line and Cache Coherency
Memory Cache Terms

- **Evicted**
  As the cache becomes full and more cache lines are loaded, an existing cache line will need to be evicted.

- **Replacement Policy**
  The policy the CPU uses to determine which cache line to evict. LRU is the most commonly used policy.
Memory Cache Terms
Memory is allocated in pages

- **Pages**
  A fixed size *(page size)* block of memory that is mapped to areas of physical memory. Page Size is often 4K.

- **Page Table**
  The page table contains the translation from the virtual address to the physical address for the pages.
Memory Cache Terms
Applications access memory virtually

- Translation Lookaside Buffer (TLB)
  Used to speed up Virtual to Physical address translation.
  TLB contains the recent mapping from the page table.

- Prefetching
  The CPU guesses at what memory will be needed next and loads it.
  - Guess right can save latency
  - Guess wrong, can cost bandwidth and cache line evictions.
**z13 Cache Capacity**

- **L1**: Dedicated one per Core
  - 96 KB Instruction, 128KB Data

- **L2**: Dedicated one per Core
  - 2 MB Instruction, 2MB Data

- **L3**: One shared per 8 Cores
  - 3 per node, 6 per drawer
  - 64 MB

- **L4**: One shared per 24 Cores
  - 1 per node, 2 per drawer
  - 480 MB

- **Main Memory**: One per drawer
  - 2.5 TB (multiple DIMMs)
In with the New, Out with the Old

Least Recently Used

Demand Fetch

256 byte cache lines
Latency

- The faster your CPU, the more cycles may be wasted during a cache miss
Memory cache
Performance Issues

<table>
<thead>
<tr>
<th></th>
<th>L1 Cache</th>
<th>L2 Cache</th>
<th>L3 Cache</th>
<th>Main memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>4 clk</td>
<td>11 clk</td>
<td>14 clk</td>
<td>6ns</td>
</tr>
<tr>
<td>In-Page Random</td>
<td>4 clk</td>
<td>11 clk</td>
<td>18 clk</td>
<td>22ns</td>
</tr>
<tr>
<td>Full Random</td>
<td>4 clk</td>
<td>11 clk</td>
<td>38 clk</td>
<td>65.8 ns</td>
</tr>
</tbody>
</table>

Sandy Bridge Latencies for accessing memory. Clk stands for clock cycles and ns stands for nanoseconds.
How slow can it go?

From the SandyBridge numbers.

- Assume 3GHz processor executes 3 instructions per cycle
- Going to the L1 cache the processor stalls for 4 clk or the CPU could have executed 12 instructions.
- If the memory is in the L2 cache the CPU could have executed 44 instructions.
- Sequentially accessing main memory would result in stalling the CPU for $6 \times 9 \ (54)$ instructions.
- Randomly accessing main memory could result in stalling the CPU for almost 600 instructions.

![Graph](image)
UMA vs NUMA
Types of memory architectures

- **UMA - Uniform Memory Access**
  - All memory is equidistance from all processors therefore memory accesses are uniform
  - Timings are consistent between multiple runs.
  - As more processor are being added the bandwidth becomes saturated impacting performance. Impedes scalability.
Types of memory architectures

- **NUMA - Non-Uniform Memory Access**
  - Memory access times very depending on where the memory is located. For best performance you need to co-locate the memory on the same chip as the processor.
  - Hardware designers started using NUMA when more and more CPUs were being added to a chip. The CPUs were running into severe bandwidth issues - “starved” - therefore NUMA improves scalability if used correctly.
  - Accessing memory on your own chip “socket” is much faster than on the other “sockets”. Furthermore, the access time between different sockets can vary.
NUMA Specifics

- Local memory doesn’t have to go thru the interconnect (bandwidth)

- Local memory has less latency than remote

- The closer the memory is to the core the better the performance

- Numa is available on many systems for instance
  - Linux, Windows, Solaris (RedHat pre 6.3 had a bug)

- It may not optimal for the application to let the OS just place the memory. Varying run times.

- Io cache’s can affect NUMA memory
NUMA Specifics

- When a page is allocated it is not placed on a NUMA node until it is first touched. A hardware fault will be generated when a process touches or writes to an address (*page fault*) that has not been used yet. The physical page is allocated during page-fault handling.

- It is at the page allocation time that the allocation policy occurs.

- The default allocation policy is for the OS to place the page on the node where the CPU is running. Remember threads may migrate to another node.

- Varying number of Sockets and varying number of CPUs per socket.
System Performance
IBM says:

- Workload capacity performance is sensitive to three major factors:
  - instruction path length
  - instruction complexity
  - memory hierarchy ("The Nest")
Instruction Path Length

- Accumulated instructions required by the application as well as through z/OS and its subsystems (JES, Catalog, IOS…)

- May be viewed in two parts:
  1. Application path length
     - is what it is unless you change the application
  2. Machine path length
     - increases with number of concurrent tasks
     - Increases with number of logical processors
Instruction Complexity

- Type and sequence of instructions
  - If then / else
  - Loops
  - Goto statements
  - Sequential

- Dependent upon processor speed (GHz) and design (pipeline, branch prediction, etc.)
Memory Hierarchy

- Caches, data buses, and memory that stages instructions and data to the CPU
- Some caches are dedicated/local to a CPU
- Other caches are shared
- In IBM z architecture, the shared caches/memory and their data buses are referred to as “The Nest”
Relative Nest Intensity (RNI)

• LPAR scope
• Published formulas for each machine
• $z13 \text{ RNI} = \frac{2.6 \times (0.4 \times L3P + 1.6 \times L4LP + 3.5 \times L4RP + 7.5 \times \text{MEMP})}{100}$

Factors that affect RNI

• What can we control here?
Over configure = Underperform

<table>
<thead>
<tr>
<th>LPS</th>
<th>CPI</th>
<th>L1MP</th>
<th>L2P</th>
<th>L3P</th>
<th>L4LP</th>
<th>L4RP</th>
<th>MEMP</th>
<th>RNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>7.86</td>
<td>2.84</td>
<td>61.33</td>
<td>20.6</td>
<td>9.35</td>
<td>7.74</td>
<td>0.97</td>
<td>1.06</td>
</tr>
<tr>
<td>16</td>
<td>7.56</td>
<td>3.35</td>
<td>55.19</td>
<td>24.82</td>
<td>6.89</td>
<td>0.29</td>
<td>12.81</td>
<td>2.73</td>
</tr>
<tr>
<td>16</td>
<td>7.51</td>
<td>5.98</td>
<td>65.86</td>
<td>24.21</td>
<td>5.47</td>
<td>2.36</td>
<td>2.09</td>
<td>0.88</td>
</tr>
<tr>
<td>16</td>
<td>7.35</td>
<td>3.1</td>
<td>72.07</td>
<td>15.77</td>
<td>10.48</td>
<td>0.26</td>
<td>1.41</td>
<td>0.69</td>
</tr>
<tr>
<td>16</td>
<td>7.31</td>
<td>2.22</td>
<td>69.77</td>
<td>14.01</td>
<td>8.96</td>
<td>6.09</td>
<td>1.17</td>
<td>0.93</td>
</tr>
<tr>
<td>16</td>
<td>7.21</td>
<td>3.15</td>
<td>57.34</td>
<td>23</td>
<td>7.45</td>
<td>0.46</td>
<td>11.75</td>
<td>2.55</td>
</tr>
<tr>
<td>16</td>
<td>7.2</td>
<td>3.73</td>
<td>61.41</td>
<td>21.37</td>
<td>10.25</td>
<td>0.42</td>
<td>6.55</td>
<td>1.67</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LPS</th>
<th>CPI</th>
<th>L1MP</th>
<th>L2P</th>
<th>L3P</th>
<th>L4LP</th>
<th>L4RP</th>
<th>MEMP</th>
<th>RNI</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.97</td>
<td>0.45</td>
<td>55.56</td>
<td>41.04</td>
<td>2.68</td>
<td>0.03</td>
<td>0.7</td>
<td>0.56</td>
</tr>
<tr>
<td>2</td>
<td>0.97</td>
<td>0.48</td>
<td>40.36</td>
<td>55.79</td>
<td>3.32</td>
<td>0.04</td>
<td>0.5</td>
<td>0.67</td>
</tr>
<tr>
<td>2</td>
<td>0.97</td>
<td>0.4</td>
<td>76.25</td>
<td>19.18</td>
<td>3.31</td>
<td>0.05</td>
<td>1.22</td>
<td>0.48</td>
</tr>
<tr>
<td>2</td>
<td>0.98</td>
<td>0.5</td>
<td>39.16</td>
<td>58.1</td>
<td>2.21</td>
<td>0.04</td>
<td>0.49</td>
<td>0.66</td>
</tr>
<tr>
<td>2</td>
<td>0.98</td>
<td>0.48</td>
<td>37.17</td>
<td>57.57</td>
<td>4.65</td>
<td>0.03</td>
<td>0.59</td>
<td>0.74</td>
</tr>
<tr>
<td>2</td>
<td>0.99</td>
<td>1.21</td>
<td>9.7</td>
<td>88.55</td>
<td>1.51</td>
<td>0.01</td>
<td>0.23</td>
<td>0.86</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.53</td>
<td>40.35</td>
<td>56.6</td>
<td>2.55</td>
<td>0.05</td>
<td>0.45</td>
<td>0.65</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.41</td>
<td>49.89</td>
<td>47.25</td>
<td>2.35</td>
<td>0.05</td>
<td>0.46</td>
<td>0.56</td>
</tr>
</tbody>
</table>
Something we can control!

Relative Nest Intensity

- Batch
- Low
- Single
- Intensive
- Low
- High locality
- Simple
- Extensive

Application Type
- IO Rate
- Application Mix
- CPU Usage
- Dispatch Rate
- Data Reference Pattern
- LPAR Configuration
- Software Configuration Tuning

- Transactional
- High
- Many
- Light
- High
- Diverse
- Complex
- Limited
IBM: Software configuration tuning

- Note that there is little one can do to affect most of these factors...
- However, one factor that can be affected, software configuration tuning, is often overlooked but can have a direct impact on RNI. Here we refer to the number of address spaces (such as CICS AORs or batch initiators) that are needed to support a workload.
- This factor has always existed but its sensitivity is higher with today’s high frequency microprocessors.
- Spreading the same workload over a larger number of address spaces than necessary can raise a workload’s RNI as the working set of instructions and data from each address space increases the competition for the processor caches.
- Tuning to reduce the number of simultaneously active address spaces to the proper number needed to support a workload can reduce RNI and improve performance.

Batch Automation: Less is More

- Lower concurrency = lower RNI = higher throughput
Software Problems dealing with Memory Caches
Software Concerns

- Design assumptions for the cache.
  - Data that is accessed once will more than likely be accessed again.
  - When memory is accessed, memory near that location will be accessed.

- Fetch Utilization
  - Good Utilization all memory in the cache is used.
  - Poor Utilization very little of the cache line is used.
Software Concerns

Data Access Problems

Once the data is in the cache, use the cache line as much as possible before it is evicted!
Software Concerns

Cache Coherency and Communication Utilization

- When 2 or more threads share a common memory area and any data is written, cache problems can occur.
- When one thread writes to the area the cache for the other thread(s) will be invalidated.
- Care should be taken to reduce the number of writes into shared memory.

False Sharing

- 2 or more threads are using data in the same cache line.
- 1 thread writes to the cache line and it invalidates the data in the other thread(s) cache line.
- Often seen when allocating arrays of data based on the number of threads and shared by the threads.
Summary

- Memory has not kept up with increases in CPU performance
- Multiple architectures across different host platforms. Each architecture has different characteristics
- Ignoring the architecture can cause your system and/or application to not perform optimally.
- Q/A
About Us

- Claire Cates
  - SAS Institute
  - claire.cates@sas.com

- John Baker
  - MVS Solutions
  - jbaker@mvssol.com
42nd International Performance & Capacity Conference

SAVE the DATE
November 7-10, 2016
Hyatt Regency | La Jolla, CA