Memory Management in the TB Age

Scott Chapman
Enterprise Performance Strategies

scott.chapman@epstrategies.com
Questions?
Send email to Scott at scott.chapman@EPStrategies.com, or visit our website at http://www.epstrategies.com or http://www.pivotor.com.

Copyright Notice:
© Enterprise Performance Strategies, Inc. All rights reserved. No part of this material may be reproduced, distributed, stored in a retrieval system, transmitted, displayed, published or broadcast in any form or by any means, electronic, mechanical, photocopy, recording, or otherwise, without the prior written permission of Enterprise Performance Strategies. To obtain written permission please contact Enterprise Performance Strategies, Inc. Contact information can be obtained by visiting http://www.epstrategies.com.

Trademarks:
Enterprise Performance Strategies, Inc. presentation materials contain trademarks and registered trademarks of several companies.

The following are trademarks of Enterprise Performance Strategies, Inc.: Health Check®, Reductions®, Pivotor®

The following are trademarks of the International Business Machines Corporation in the United States and/or other countries: IBM®, z/OS®, zSeries® WebSphere®, CICS®, DB2®, S390®, WebSphere Application Server®, and many others.

Other trademarks and registered trademarks may exist in this presentation.
Performance Workshops Available

During these workshops you will be analyzing your own data!

- **Parallel Sysplex and z/OS Performance Tuning (Web / Internet Based!)**
  - Instructors: Peter Enrico and Scott Chapman
  - August 2016

- **WLM Performance and Re-evaluating of Goals**
  - Instructors: Peter Enrico and Scott Chapman
  - Chicago area
  - October 2016

- **Essential z/OS Performance Tuning Workshop**
  - Instructors: Peter Enrico, Tom Beretvas, and Scott Chapman

- **z/OS Capacity Planning and Performance Analysis**
  - Instructor: Ray Wicks
Agenda

• Review of processor speed
• Hierarchy of data accesses
• Review of processor caches and DAT
• Considerations: uses for more memory and things to watch out for
• Measurements you might want to track
Clock cycles and effective capacity

• Ideally, you’d like to get real work done each clock cycle
• z Processor speeds are really fast
  – z10 – 4.4 Ghz
  – z196 – 5.2Ghz
  – zEC12 – 5.5Ghz
  – z13 – 5.0 Ghz

Billions of cycles per second
1 Clock cycle = fraction of a nano second

• So 1ms to wait for an I/O = millions of clock cycles
How long is a cycle again?

- Just over 2 inches
  - Light, in a vacuum
  - Electrical signal in a circuit is much slower (40-70% of c)
  - 1 meter in fiber ~ 5 ns

- Need to make a round trip

- Signal paths aren’t as the mosquito flies
  - 7.7 Miles of wire in a zEC12 chip, >13 in z13

- Physical distance matters!
Data access hierarchy

- Register
- Memory
  - L1 Cache
  - L2 Cache
  - L3 Cache
  - L4 Cache
    - Local
    - Remote
  - Real
- Storage Class Memory
- Disk
  - Cache
  - SSD
  - Spinning
- Network

The farther the data is away from the processor, the more clock cycles will be spent accessing it.

Optimal performance & capacity utilization = keeping data as close to processor as possible!
The only good I/O is no I/O

• To keep the processor fed, data needs to be close and ready
  – Disk controller cache can help
  – SSDs can help read response times for cache misses
  – But the disk subsystem is still a long ways away

• Best way to improve I/O performance is to not do the I/O
  – Keep the data in memory
  – Finite limits on memory of course
Why not 4 TB of memory for every LPAR?
Memory costs

• Memory on z has historically been very expensive compared to other platforms

• Partly due to the robust nature of z memory
  – RAIM greatly enhances error detection and avoidance
  – Can sustain multiple component failures concurrently
    • Bit, lane, DRAM, DIMM, socket, even complete channels
  – Scott’s Mainframe Motto: Quick answers are nice, correct answers are required
  – Not an esoteric point: 2009 Google study: annual incidence of uncorrectable memory errors: 1.3%/machine, 0.22%/DIMM

• But what’s the biggest part of your mainframe budget?
  – Almost always: software, usually by a wide margin
  – (Not including staff costs, which can vary by geography)
## Memory prices coming down, sizes going up

<table>
<thead>
<tr>
<th>Year</th>
<th>Machine</th>
<th>Approx Max Memory / CEC</th>
<th>Max Memory / Book</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>z9EC</td>
<td>0.5 TB</td>
<td>128 GB</td>
</tr>
<tr>
<td>2008</td>
<td>z10EC</td>
<td>1.5 TB</td>
<td>384 GB</td>
</tr>
<tr>
<td>2010</td>
<td>z196</td>
<td>3 TB</td>
<td>768 GB</td>
</tr>
<tr>
<td>2012</td>
<td>zEC12</td>
<td>3 TB</td>
<td>768 GB</td>
</tr>
<tr>
<td>2015</td>
<td>z13</td>
<td>10 TB</td>
<td>2560 GB</td>
</tr>
</tbody>
</table>
What can we do with more memory?

• Process more data
  – More data generated today, richer data types
  – Support more dev/test environments

• Performance – I/O avoidance
  – Meet business goals or exploit new business opportunities
  – Offset other constraints

• CPU reduction – generally, it can be fewer cycles spent to get data from memory than disk
  – Also, avoids things being pushed out of cache while waiting on I/O
  – Less CPU generally means less software cost

• Make ourselves more efficient – stop micro-managing storage
  – Staff costs the other big piece of the mainframe budget
Hypothetical Improvements

- Improved response time may give WLM more flexibility in managing the work
- Or maybe we can constrain the R4H
- Velocity goals may need adjustment
Storage Class Memory (SCM)

- Nowhere near as fast as main memory, but much faster than going to disk
  - Even SSD disk
  - Physically closer (inside the CEC)
  - Avoids going through FICON
- Cheaper than memory, more expensive than disk
- Large relative to current common memory sizes, small relative to disk
  - 1.4TB increments (up to 4 increments/CEC)
- Initial uses:
  - Paging, including pageable 1 MB pages (not DB2 v11 BPs)
    - Ideal for large memory configurations
  - CF List structure storage (MQ shared queues)
Exercising SCM (zEC12 4xx)

- Note that “pages” really means 4K equivalent pages
- My calculations:
  - Average paging rate of 14311/sec
  - Average I/O rate of about 1056/sec
  - Page transfer time was about 0.000063 (63 microseconds)
- Much slower than memory, much faster than disk
Remember…

• Paging is not free—even with SCM!

• Avoid paging for production address spaces
  – Even though DB2 supports pageable large pages, you’re probably better off page-fixing them

• But some SCM paging for dev/test regions might be acceptable
  – Depending on usage patterns
  – Allow idle environments to page out / in as they are needed
  – Maybe allow more dev/test regions without more real memory
Going deeper in the hierarchy

- Register
- Memory
  - L1 Cache
  - L2 Cache
  - L3 Cache
  - L4 Cache
    - Local
    - Remote
  - Real
- Storage Class Memory
- Disk
  - Cache
  - SSD
  - Spinning
- Network

The farther the data is away from the processor, the more clock cycles will be spent accessing it.

Optimal performance & capacity utilization = keeping data as close to processor as possible!
• Approximation based on IBM docs
• Cache sizes scaled relative to each other
• Physical location of L1 cache unclear
z13 Storage Control Chip Schematic

- Approximation based on IBM docs
- NIC directory embedded in the 4 L4 areas
- L4 Controller schematic simplified
Data locality – L4 / Memory

Not to Scale
Memory address translation

Dynamic Address Translation & Prefixing: Virtual -> Real -> Absolute
z/OS 64-bit Address Translation

<table>
<thead>
<tr>
<th>RFX</th>
<th>RSX</th>
<th>RTX</th>
<th>SX</th>
<th>PX</th>
<th>BX</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 bits</td>
<td>11 bits</td>
<td>11 bits</td>
<td>11 bits</td>
<td>8 bits</td>
<td>12 bits</td>
</tr>
</tbody>
</table>

Page address

Page Table

Segment Table

Region Third Table

Region Second Table

Region First Table

Address Space Control Element

Real address
Large Page Address Translation

- **RFX**: 11 bits
- **RSX**: 11 bits
- **RTX**: 11 bits
- **SX**: 11 bits
- **BX**: 20 bits

**Address Space Control Element**

**Region Origin**
- **First Table**
- **Second Table**
- **Third Table**

**Real address**

**Page address**

**Segment Table**
### Giant Page Address Translation

<table>
<thead>
<tr>
<th>RFX</th>
<th>RSX</th>
<th>RTX</th>
<th>BX</th>
</tr>
</thead>
<tbody>
<tr>
<td>11 bits</td>
<td>11 bits</td>
<td>11 bits</td>
<td>31 bits</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>11</td>
<td>21</td>
</tr>
</tbody>
</table>

- **RFX**: Region First Table Origin
- **RSX**: Region Second Table Origin
- **RTX**: Region Third Table Origin
- **BX**: Real address

**Address Space Control Element**

**Region**

1. First Table
2. Second Table
3. Third Table

**Page address**

- Page address
- Page address
- Page address
- Page address
- Page address
- Page address
- Page address
- Page address
- Page address
Dynamic Address Translation

• DAT performed using multiple tables that point to different ranges of storage
• DAT is not free!
• Result of DAT cached in Translation Look-aside Buffers (TLB)
• TLBs are in L1 cache and managed by the hardware
• Relatively small
• Flushed when DAT table changes
• 1MB & 2GB pages make TLBs more effective
Estimated impact of TLB Misses

TLB CPU Miss Pct of CPU Average for Each System Over Time on CEC

Date, Time

Yaxis-1
SYSSL - Good Marker
Cache utilization & performance

• Memory is far away from the processor core and relatively slow
• Effective use of processor cache is important to keeping the processor “fed”
• Cache effectiveness measurements are in the Hardware Instrumentation Services SMF 113 records
  – Requires z/OS 1.8 +PTFs & z10 GA2
• Enable HIS and record the 113 records
  – Required for effective capacity planning on upgrade
SMF 113 Cache Measurements

• L1MP – Level 1 Misses per 100 Instructions

• CPI – Cycles Per Instruction
  – Estimated Finite CPI – effectively: penalty cycles per instruction due to the fact that caches are finite
  – Estimated Instruction Complexity CPI – CPI as if there were no penalty cycles (completely effective L1 cache)

• RNI – Relative Nest Intensity
  – Combined calculation for effect of overheads to get to cache and memory
  – Used for capacity planning on upgrade

• TLB CPU Miss Percent of CPU – How much CPU time goes to resolving DAT during TLB misses
SMF 113 - CPI and Estimated Finite CPI - For System Over Time

Date, Time

Yaxis-1
- CPI
- Est_F_CPI
- Severe Marker
SMF 113 - Verbose - L1MP for Each CPU for System Over Time

Average of L1MP
Zoomed in on just CPs

SMF 113 - Verbose - L1MP for Each CPU for System Over Time

Average of L1MP
Zoomed in on zIIPs
RNI – Evaluate over time

SMF 113 - Verbose - RNI for Each CPU for System Over Time

Average of RNI
Combine L1MP and RNI for workload "hint"

<table>
<thead>
<tr>
<th>L1MP</th>
<th>RNI</th>
<th>Workload Hint</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;3%</td>
<td>&gt;= 0.75</td>
<td>AVERAGE</td>
</tr>
<tr>
<td></td>
<td>&lt; 0.75</td>
<td>LOW</td>
</tr>
<tr>
<td>3% to 6%</td>
<td>&gt;1.0</td>
<td>HIGH</td>
</tr>
<tr>
<td></td>
<td>0.6 to 1.0</td>
<td>AVERAGE</td>
</tr>
<tr>
<td></td>
<td>&lt; 0.6</td>
<td>LOW</td>
</tr>
<tr>
<td>&gt;6%</td>
<td>&gt;=0.75</td>
<td>HIGH</td>
</tr>
<tr>
<td></td>
<td>&lt; 0.75</td>
<td>AVERAGE</td>
</tr>
</tbody>
</table>
How can you improve cache effectiveness?

• Enable HiperDispatch
• Make good use of large pages
• Upgrade to newer machine

<table>
<thead>
<tr>
<th>Year</th>
<th>Machine</th>
<th>L1 Data</th>
<th>L1 Inst.</th>
<th>L2 Data</th>
<th>L2 Inst.</th>
<th>L3</th>
<th>L4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2005</td>
<td>z9 EC</td>
<td>256KB</td>
<td>256KB</td>
<td>40MB</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
</tr>
<tr>
<td>2008</td>
<td>z10 EC</td>
<td>128KB</td>
<td>64KB</td>
<td>3MB</td>
<td>n/a</td>
<td>48MB</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>z196</td>
<td>128KB</td>
<td>64KB</td>
<td>1.5MB</td>
<td>24MB</td>
<td>192MB</td>
<td></td>
</tr>
<tr>
<td>2012</td>
<td>zEC12</td>
<td>96K</td>
<td>64K</td>
<td>1MB</td>
<td>1MB</td>
<td>48MB</td>
<td>384MB</td>
</tr>
<tr>
<td>2015</td>
<td>z13</td>
<td>128K</td>
<td>96K</td>
<td>2MB</td>
<td>2MB</td>
<td>64MB</td>
<td>224MB</td>
</tr>
</tbody>
</table>

• Consider more/slower CPUs instead of fewer/faster
  – More CPUs = More L1/L2/TLB
Considerations
Are you using all your memory?
Better... minimum available over a week
Do you have reserved memory?

- “Reserved” = “unused” = not allocated to any LPAR
- Some sites seem to have given all the memory to LPARs
  - This makes responding to new requirements difficult
- Some sites hold back some memory
  - Define some amount of reserved memory to each LPAR
  - Total reserved across all LPARs can be > actual reserved
    - Just means you can’t bring it all LPAR’s reserve elements online
    - Make sure reserved by LPAR < actual reserved, or won’t be able to bring that LPAR’s reserve element online
  - Consider specifying RSU=OFFLINE to make it reconfigurable
    - Then you can take all or part of it offline dynamically
    - May limit use of the area, e.g. not preferred for long-term pages
- **Scott’s recommendation:** don’t immediately give away all your memory
How active will that new memory be?

• How is your total cache size going to increase relative to your memory size?
• This may be a non-issue depending on how you plan on using that new memory
• But it may be something to consider: consider more/slower instead of fewer/faster processors
  – More CPUs = more L1/L2 cache
  – Be sure to investigate single-CP workloads running in unconstrained times before doing this
    • If they’re already suffering CPU delay, more/slower CPUs may be fine
**TLB miss overhead**

- If you add a whole bunch of memory as 4K pages you may see overhead due to TLB misses go up
  - Again, workload-dependent of course
- **Consider using 1M pages wherever possible**
  - Probably should be the default wherever you can
  - Non-authorized allocation non pageable large pages must be permitted to IARRSM.LRGPAGES
- Plan your LFA size & check that you got what you thought you were getting
  - DISPLAY VIRTSTOR,LFAREA
- If you have giant amounts of storage, consider giant pages
Estimated impact of TLB Misses

![Graph showing Estimated impact of TLB Misses](image_url)
DB2 Buffer Pools

• Going “Biggie Size” on DB2 buffer pools probably a good use of memory
• Start with BPs and/or objects that do the most I/O
  – BPs that don’t do a lot of I/O won’t give you a lot of benefit
• Don’t forget about the group buffer pools
  – Catterall’s ROT is probably still good: sum(local BPs) * 0.4
  – May get benefit from super-sizing the group BPs without increasing local BPs, but I’d prefer to make locals bigger
    • Keep the data as close to the CPU as possible
• Consider PGSTEAL(NONE) in DB2 10+ for pinning objects in a BP
  – DB2 will load the BP on first access and then assume no I/O needed for subsequent access
• Use Large (v10+) or Giant (v11+) pages
Other DB2 areas

• Prepared statement cache

• RID Pool
  – DB2 10 default is 400 MB, will use workfile database if not enough space in RID Pool

• Sort pool
  – Careful: this is per concurrent sort
  – V10 changed default to 10MB, max 128MB

• Utilities: consider giving more memory for improved performance
• Some shops restrict sort’s use of memory, either all the time, or perhaps at certain times of the day
• For example: restrict sort’s use of memory during the online day
  – This thinking made a lot of sense when memory was extremely precious
  – Today, CPU cycles may be more precious than memory because increased CPU usage = increased software cost
• It may be time to rethink your sort memory limits
  – “rethink” does not necessarily mean eliminate
Eliminate IEFUSI limits??

• IEFUSI (SMF Step Initiation Exit) is often used to limit the amount of memory a job (or STC) step can acquire
  – The idea being to prevent something from running away with lots of storage and causing severe performance issues

• My opinion: IEFUSI is still a good idea, but like sort, rethink limits
  – It used to be that most application jobs wouldn’t need more than 10s of MBs
  – Today Java batch jobs could easily use a GB or more

• Don’t forget to limit 64-bit storage
• Consider using the new parmlib member to replace IEFUSI
What about more Java?

• Java is definitely viable on the mainframe
• Java programmers may be more generally available than COBOL
  – JZOS API allows easy access to z/OS constructs
• Running on the specialty engines can save money
• One of the concerns in the past has been that Java will almost certainly require more memory to run the same function
  – If we have more memory, this is less of a concern, but…
Avoid unnecessarily large JVMs

- If you give a long-running JVM an arbitrarily large heap, it will use it
  - Garbage collection may cause paging spikes if heap has been paged out
- Avoid micromanagement, but every application doesn’t need GBs of heap space by default
- Avoid min heap = max heap until you’ve determined actual heap requirements
  - Smaller area of activity = better cache locality of reference
- Unfortunately, sizing requires testing
  - But probably not required for many batch programs: give them a max of 128-256MB and many will likely be fine
  - Choice of JDBC driver and settings may influence heap requirements
  - Having IEFUSI enforce some limit is probably a good idea
Examples of measurements to monitor
Page-in rate

Address Space Page-In Rate - All Systems

PRODPLEX

Yaxis-1
- SYSK
- SYSL
- Warning Marker
Paging by workload
Page-out rate

Address Space Page-Out Rate - All Systems

PRODPLEX

Y-axis: Average of Page-Out Rate

X-axis: Time (2016-05-18 00:00:00 to 2016-05-18 22:30:00)

- SYSK
- SYSL
- Warning Marker

EPS
Minimum available storage
TLB overhead

TLB CPU Miss Pct of CPU Average for Each System Over Time on CEC

Date, Time

Yaxis-1

SYSB - Good Marker
Summary

- Memory is getting cheaper, and is cheaper than CPU, so leverage memory to save CPU
- SCM can reduce cost of paging
  - Some paging may be acceptable for certain workloads
- Consider adjusting old memory limits
- Many memory statistics available, keep an eye on a few key ones as you start using more memory
  - And hopefully IBM will provide us with some more (2GB pages?!?)