Memory for MIPS: Leveraging Big Memory to Boost DB2 for z/OS CPU Efficiency

Michigan DB2 Users Group

March 25, 2015
Agenda

- The current landscape
- Getting your buffer pool house in order
- Being bold – but not reckless – in asking for more real storage for a DB2 subsystem
- New opportunities for exploiting RELEASE(DEALLOCATE)
- Other ways to trade memory for MIPS
The current landscape
z Systems server memory: getting BIG

- Seeing more production z/OS LPARs with 100+ GB of real storage

- What’s driving the trend towards larger z/OS LPAR memory sizes?
  - Desire for balanced configurations: more memory to go with more MIPS
    - Up to 101 engines on a zEC12 server, up to 141 engines on a z13 – each engine provides about 1000 MIPS of processing capacity
    - Not unusual to see 20 GB or more of real storage per engine
  - Price of z Systems memory now much lower than it used to be
    - And it got even less expensive with the z13
    - Depending on how much additional memory you buy when you upgrade a z196 or zEC12 server to a z13, the cost of the added memory can be up to 87% less than the cost of memory on a z196 or zEC12 mainframe
    - An IBM z Systems sales specialist can give you the details
New ways to leverage Big Memory on z

- Recent DB2 for z/OS developments provide more opportunities to use z Systems memory advantageously

- A few examples (covered in more detail later in presentation):
  - Page-fixed buffer pools
  - Use of larger real storage page frames for buffer pools (1 MB with DB2 10, 2 GB with DB2 11)
  - DB2-aware “pinning” of objects in buffer pools (DB2 10)
  - Thread-related virtual storage almost entirely above 2 GB “bar” with DB2 10 (when packages bound or rebound in DB2 10 environment)
    - More concurrent threads per subsystem
    - More virtual storage “head room” for use of RELEASE(DEALLOCATE) bind option – and DB2 10 delivered a new way to leverage RELEASE(DEALLOCATE) for CPU efficiency: high-performance DBATs
There are a lot of “sleeping gigabytes” out there

- At many DB2 for z/OS sites, LOTS of spare memory capacity
- Do you know your z/OS system’s demand paging rate?
  - Available via a z/OS monitor, this is the rate at which pages that have been sent by z/OS to auxiliary storage are paged back into system memory on-demand
  - In my experience, it’s often less than 1 per second, even during busy periods
    - If the demand paging rate is in the low single digits (or less) per second, z/OS LPAR memory is not stressed – *look for opportunities to use more of it as a means of boosting the CPU efficiency of your DB2 workload*
I call this memory for MIPS, and that’s what this presentation is all about
Getting your buffer pool house in order
What I mean…

- Before enlarging a DB2 subsystem’s buffer pool configuration, make sure that you’re getting the most out of the configuration as currently sized
First, customize settings for work file buffer pools

- Referring here to the buffer pools dedicated to the 4K-page and 32K-page work file table spaces
  - You DO assign work file table spaces to their own buffer pools, don’t you?

- Work file table spaces are different from others in a couple of ways that have implications for recommended buffer pool parameter settings
  - For one thing, almost all reads are of the prefetch variety
    - Why that matters: the default value of the VPSEQT buffer pool parameter (the percentage of a pool’s pages that can be occupied by pages read into memory via prefetch) is 80
    - Stay with that, and you’ll be wasting 20% of the buffers in a pool dedicated to work file table spaces
    - Increasing VPSEQT to 95-99% for a work file-dedicated pool should result in decreased read I/O activity, and fewer I/Os means less CPU consumption
Work file-dedicated pools – the other difference

- Different motivation for externalizing changed pages to disk
  - For other buffer pools, getting changed pages externalized to disk in a timely manner is important for DB2 restart performance
    - Forward log recovery phase of restart: data on disk updated to reflect changes committed, but not yet externalized, at the time of DB2 subsystem failure
    - The more changed-but-not-yet-externalized pages there are in buffer pools (other than work file pools) at time of DB2 failure, the longer restart will take
    - That being the case, you want fairly low values for deferred write thresholds (DWQT, VDWQT) for these pools (defaults of 30 and 5 are usually good)
  - Work files are like scratch pads for query processing – they are not recovered when DB2 is restarted (queries would be resubmitted)
    - Therefore, only need a level of changed-page externalization activity that is sufficient to prevent thrashing (i.e., avoid shortage of stealable buffers)
    - Fewer page writes = less CPU consumption
    - I’ve seen 70/40 used for DWQT/VDWQT for work file pools, and also 80/50

*But don’t take this too far…*
A warning about too-high DWQT/VDWQT values

- At one site, I saw values of 90 for both DWQT and VDWQT for a work file-dedicated buffer pool.

- The data manager threshold (aka DMTH) is reached when 95% of a pool’s buffers are non-stealable (either currently in use or changed-but-not-yet-externalized).
  - When DMTH is hit for a pool, there will be a GETPAGE for each row retrieved from a page cached in that pool.
    - So, if 20 rows are retrieved from a page, there will be 20 GETPAGEs vs. 1.
    - Result: significant increase in DB2 CPU consumption.
  - With DWQT and VDWQT both at 90, at this site DMTH was hit many times per hour for the buffer pool in question, unbeknownst to DB2 staff.
  - Check DB2 monitor display or statistics report, or output of DB2 command -DISPLAY BUFFERPOOL(BPn) DETAIL to see if DMTH hit.
    - You always want that value to be zero.
More good buffer stewardship: steal smart

- By default, DB2 utilizes a least-recently-used (LRU) algorithm for buffer stealing (i.e., to select occupied buffers that will be overwritten to accommodate new pages brought in from disk)
  - This is the right algorithm to use in most cases

- Before DB2 10, for a buffer pool used to “pin” database objects in memory (i.e., to cache them in memory in their entirety), the FIFO buffer steal algorithm was recommended
  - Why? Because FIFO (first in, first out) is simpler than LRU and so uses less CPU
  - If objects are cached in their entirety (i.e., if total number of pages for objects assigned to pool is less than number of buffers in pool), there won’t be any buffer stealing, so why spend CPU tracking buffer usage?

  *DB2 10 provided a better way to do this*…
Smarter pinning with DB2 10

- New buffer steal algorithm for pools: PGSTEAL(NONE)

- When PGSTEAL(NONE) is specified for a pool:
  - When an object (table space or index) assigned to the pool is first accessed, requesting application process will get what it needs and DB2 will asynchronously read all the rest of the object’s pages into the pool
  - When SQL statements targeting the pinned object are subsequently optimized, DB2 will assume that no I/Os will be required for access to the object’s pages, and this will factor into access path selection
  - If objects assigned to a PGSTEAL(NONE) pool have more pages than the pool has buffers, DB2 will automatically switch to FIFO buffer steal algorithm to accommodate new page reads from disk when pool is full
CPU savings via page-fixed buffer pools

- **PGFIX(YES)** specification for a buffer pool saves MIPS by making database read and write I/Os less costly
  - **PGFIX(YES)** buffers stay fixed in memory, so no need for DB2 to ask z/OS to fix (and subsequently release) a real storage page frame holding a buffer when data is read into, or written from, that buffer
    - For 32-page prefetch read, that’s 32 page-fix/page-release operations avoided
    - If DB2 data sharing, CPU savings for coupling facility writes and reads, too
DB2 10 and PGFIX(YES): not just cheaper I/Os

- With z10, z196, zEC12, and z13 servers (and z/OS 1.10 or later), part of memory resource can be managed in 1 MB page frames
  - LFAREA parameter in IEASYSxx member of PARMLIB

- DB2 10 will use 1 MB page frames to back a PGFIX(YES) buffer pool (if not enough 1 MB page frames to fully back pool, DB2 10 will use 4 KB frames in addition to 1 MB frames)
  - Larger page frame means more CPU-efficient translation of virtual storage addresses to real storage addresses
  - This CPU benefit is on top of the previously mentioned MIPS savings resulting from less-costly I/Os into and out of page-fixed buffer pools
DB2 11: 2 GB page frames for buffer pools

- **Prerequisites:**
  - z/OS 2.1 or 1.13 with fix for APAR OA40967
  - zEC12 or z13 server
  - LFAREA specification (in IEASYSxx member of PARMLIB) with non-zero value for 2G option

- **When it will be used:**
  - Buffer pool must be defined with PGFIX(YES) and FRAMESIZE(2G)
  - Size of buffer pool must be at least 2 GB
    - Amount backed by 2 GB frames will be integer portion of (pool size) / 2 GB
    - Remainder (if any) will likely be backed by a mix of 1 MB, 4 KB page frames
  - 2 GB page frames not likely to improve performance much versus 1 MB frames, unless pool is really big
    - One DB2 11-using organization found that the size of a buffer pool had to be at least 20 GB for 2 GB page frames to deliver significant CPU savings

New keyword for -ALTER BUFFERPOOL command
Reduce I/Os by redistributing buffers

- Referring here to reducing size of low-I/O pools and increasing size of high-I/O pools by a like amount
  - Goal: about same read I/O rate for reduced-size pools (don’t take too many buffers from these pools), and less read I/O activity for enlarged pools – all with no net increase in overall size of buffer pool configuration
    - Again, fewer I/Os = reduced CPU consumption

- Step 1: determine read I/O rate for each buffer pool, using information obtained from:
  - DB2 monitor (statistics report, or online display of buffer pool activity)
  - or-
  - Output of DB2 command -DISPLAY BUFFERPOOL(ACTIVE) DETAIL
    - Issue command once, then wait an hour and issue it again
    - Output of second issuance of command will capture 1 hour of activity – divide activity counters by 3600 to get per-second figures
Calculating a pool’s read I/O rate

- What you want is per-second data

- Total read I/O rate is sum of:
  - all synchronous reads (random and sequential) per second
  - and
  - all prefetch reads (sequential prefetch + list prefetch + dynamic prefetch) per second

- Example: BP1 and BP2 both have 40,000 buffers, and total read I/O rate is 20/second for BP1 and 2000/second for BP2
  - In that case, I’d consider decreasing size of BP1 by 20,000 buffers and increasing size of BP2 by 20,000 buffers
Being bold – but not reckless – in asking for more real storage for a DB2 subsystem
Would bigger be better?

- If your buffer pool house is in order, should you make the house bigger (i.e., increase the size of the overall buffer pool configuration)?
  - Depends – what’s the current total read I/O rate for each of your buffer pools (see slide 19)?
    - If read I/O rate for each buffer pool is less than 100 per second, making the pools larger is not likely to have much of a CPU-saving impact
    - If read I/O rate for one or more pools is in the 100s or 1000s per second, making that pool (or pools) larger could yield significant CPU efficiency benefits (because fewer I/Os = less CPU consumption)
If you’re going to make a buffer pool bigger...

- ...add enough buffers to make a difference
  - Adding another 1000 buffers to a pool that already has 80,000 buffers is not likely to move the needle very much
    - Increase the size of that 80,000-buffer pool by 20,000 or 40,000 buffers, and I’d say, “Now you’re talking”
    - If a pool is quite small – say, 10,000 4K buffers – and has a high read I/O rate, I might want to increase its size by a factor of two (or more)

- Some organizations are starting to “get it,” in terms of taking advantage of Big Memory to reduce DB2 CPU consumption via larger buffer pool configurations
  - The largest buffer pool configuration I’ve seen on one DB2 for z/OS subsystem is 90 GB (associated z/OS LPAR has about 212 GB of real storage)
Get on the same page as your z/OS sysprog

- Again, the measure of “memory stress” that I like to use is the demand paging rate
  - As you implement memory-for-MIPS changes, keep an eye on the z/OS LPAR’s demand paging rate, and don’t let this get out of hand
  - If the demand paging rate is in the low single digits (or less) per second, it’s not out of hand

- Whatever your LPAR’s demand paging rate, I’d be careful about using more than 50% of an LPAR’s memory for DB2 buffer pools
  - I’ve seen WLM size a buffer pool configuration at 30-40% of LPAR memory (ALTER BUFFERPOOL(BPn) AUTOSIZE(YES))
Keep monitoring read I/O rate for buffer pools

- I generally take a triage approach: focus efforts on the pools with the highest read I/O rates
  - Highest rate I’ve seen is 9000 read I/Os per second for one buffer pool

- My aim: if possible, get read I/O rate to less than 1000 per second for each buffer pool
  - If that objective is accomplished (or if I’m seeing diminishing returns with respect to enlarging a high-I/O buffer pool), I turn my focus to pools with read I/O rates between 100 and 1000 per second
    - Nice to get these below 100 per second, if possible

- Keep in mind: it’s not just about making existing pools larger
  - At some point, you may want to create a new BPy, and reassign objects to that pool from BPx
    - Can be particularly effective for separating “history” vs. “current” tables, access patterns for which tend to be different
If data sharing, don’t forget group buffer pools

- Sometimes people will make BPx larger across members of a data sharing group, and will forget to enlarge GBPx accordingly
  - If aggregate size of local BPs gets too large relative to the size of the corresponding GBP, you could end up with a lot of directory entry reclaims and that’s not good for performance
    - Can check on directory entry reclaim activity using output of DB2 command - DISPLAY GROUPBUFFERPOOL(GBPn) GDETAIL

- For a 4K GBP with the default 5:1 ratio of directory entries to data pages, directory entry reclaims likely to be 0 if size of GBP is 40% of combined size of associated local BPs
  - Example: 3-way group, BP1 at 40K buffers (160 MB) on each member
    - Good GBP1 size is 40% X (3 X 160 MB) = 192 MB
  - In a blog entry I provided a more general approach to GBP sizing
    http://robertsdb2blog.blogspot.com/2013/07/db2-for-zos-data-sharing-evolution-of.html
New opportunities for exploiting RELEASE(DEALLOCATE)
What’s good about RELEASE(DEALLOCATE)?

- Package bind option – DB2 will retain certain resources allocated to a thread (table space-level locks, package sections) until thread deallocation vs. releasing at commit.

- RELEASE(DEALLOCATE) can save MIPS versus RELEASE(COMMIT) when:
  - Thread in question persists through commits (examples are batch threads and CICS protected entry threads)
  - Thread is used repeatedly for the execution of the same package
    - In that case, RELEASE(DEALLOCATE) avoids CPU cost of repeated release and re-acquisition of same table space locks and package sections at each commit.
What’s the memory-for-MIPS angle?

- RELEASE(DEALLOCATE) increases memory utilization because it increases amount of virtual storage used by threads

- Before DB2 10, a significant portion of thread-related virtual storage was acquired below the 2 GB “bar” in DBM1
  - Talking about the part of the EDM pool used for PT (package table)
    - Copies of package sections allocated to threads
    - Use of RELEASE(DEALLOCATE) made more of this space non-stealable, which could potentially lead to program failures due to lack of space in EDM pool
DB2 10 thread storage

- 75-90% less usage of below-the-bar DBM1 virtual storage with DB2 10 vs. DB2 9
  - Primarily due to movement of almost all thread-related storage above 2 GB bar (requires package rebind!)

- Result: much more virtual storage “head room” for use of RELEASE(DEALLOCATE)

- Also, ZPARM no longer limits space available for package table (it’s no longer in EDM pool) – just need to have enough memory
More on DB2 10 and RELEASE(DEALLOCATE)

- Prior releases: RELEASE(DEALLOCATE) not honored when package executed via DBAT (i.e., a DDF thread)
  - Instead, treated as though bound with RELEASE(COMMIT)
  - Why: DBATs can stick around a LONG time, and there was concern that combination of RELEASE(DEALLOCATE) and DBATs would block DDL, binds, etc.

- DB2 10: when package bound with RELEASE(DEALLOCATE) is executed via a “regular” DBAT, that DBAT becomes a high-performance DBAT
  - RELEASE(DEALLOCATE) honored
  - High-performance DBAT dedicated to connection through which it was instantiated, vs. going back into DDF thread pool when transaction ends
  - If thread reused for same package, you get CPU benefit of RELEASE(DEALLOCATE)
More on high-performance DBATs

- High-performance DBAT will be terminated after being used to process 200 units of work (to free up resources)

- Can suspend honoring of RELEASE(DEALLOCATE) for packages executed via DBATs by issuing command -MODIFY DDF PKGREL(COMMIT)
  - Issue -MODIFY DDF PKGREL(BNDOPT) to switch back

- What if most of your SQL executed through DDF is dynamic?
  - Consider binding IBM Data Server Driver Package (or DB2 Connect) packages into the default NULLID collection with RELEASE(COMMIT), and into another collection with RELEASE(DEALLOCATE)
    - Have your higher-volume client-server transactions use that second collection to gain high-performance DBAT performance benefits (collection name can be specified as a data source property on the client side)
RELEASE(DEALLOCATE) recommendations

- **Best candidates:**
  - Packages frequently executed, and executed via persistent threads such as CICS-DB2 protected entry threads and high-performance DBATs
  - Packages associated with batch jobs that issue lots of commits
    - Batch bonus: greater benefit from dynamic prefetch, index lookaside

- **Operational considerations:**
  - Ensure that RELEASE(DEALLOCATE) packages do not acquire exclusive table space locks (check for lock escalation, LOCK TABLE)
  - May need to restrict execution of RELEASE(DEALLOCATE) packages during times of significant DDL, bind/rebind, utility operations
    - DDF: use -MODIFY DDF PKGREL command (see previous slide)
    - DB2 11: DDL, bind/rebind, utility operations can “break in” on a local persistent thread used to execute RELEASE(DEALLOCATE) packages (for high-performance DBATs, continue to use -MODIFY DDF command)
Other ways to trade memory for MIPS
Dynamic statement caching

- Global statement cache has been above the 2 GB bar in DBM1 address space since DB2 V8
  - Size determined by ZPARM parameter EDMSTMTC

- Larger statement cache = more cache “hits”
  - CPU savings achieved through resulting avoidance of full PREPAREs

- I regularly see dynamic statement cache hit ratios of 90% or higher, so you might want to aim for that on your system
RID list processing

- RIDs, or row IDs, are row location indicators found in index entries

- DB2 processes RID lists for things such as:
  - List prefetch
  - Index ANDing and index ORing

- CPU savings can be achieved if RID pool (sized by way of ZPARM parameter MAXRBLK) is large enough to enable RID processing for a query to complete in memory
  - DB2 10: default size of RID pool went to 400 MB, from 8 MB before
    - RID pool has been above the 2 GB bar in DBM1 since DB2 V8
In-memory RID list processing: a DB2 10 change

What happens when there is not enough space in the RID pool to allow a RID list processing operation to complete in memory:

- Before DB2 10: DB2 will abandon RID list processing and go with a table space scan for data access
- DB2 10: DB2 continues working on the RID list processing operation, using space in the work file database (but not for hybrid join)
  - That’s better than giving up and falling back to a table space scan, but not quite as CPU-efficient as getting entire RID list processing operation done in memory
  - CPU cost of completing RID list processing using work file space is reduced if the buffer pool dedicated to 32KB-page work file table spaces is large enough to keep read I/O rate down
  - New ZPARM parameter MAXTEMPS_RID can be used to limit amount of work file space that one RID list can use
The sort pool

- This is space in DBM1 (above the 2 GB bar since DB2 V8) that is used for SQL-related sorts (as opposed to utility sorts)

- The larger the pool, the more CPU-efficient SQL sorts will be
  - Size determined by ZPARM parameter SRTPOOL
  - Note that this is the maximum size of the sort work area that DB2 will allocate for each concurrent sort user

  - So, don’t go overboard here if you have a lot of concurrent SQL sort activity on your system
  - Default size of sort pool went to 10 MB with DB2 10, from 2 MB before
  - Maximum SRTPOOL value is 128 MB – largest I’ve seen on a DB2 for z/OS system is 48 MB (at that site, I/O rate for work file-dedicated buffer pools was very low – perhaps due to large sort pool)

  - As with any other memory-for-MIPS trade, if you make SRTPOOL larger, keep an eye on the z/OS LPAR’s demand paging rate – you want that to be in the single digits (or less) per second
Thanks

Robert Catterall
rfcatter@us.ibm.com