Introduction to HiperDispatch Management Mode with z10[1]

Donald R. Deese
Computer Management Sciences, Inc.
Hartfield, Virginia 23071-3113

HiperDispatch was introduced with IBM's z10 server, and is available (via PTFs) on z/OS V1R7, z/OS V1R8, and z/OS V1R9. HiperDispatch was designed to (1) minimize the z10 hardware performance degradation caused by processor cache misses, and (2) maximize the amount of CPU processing power associated with any single logical processor. To achieve these design objectives, HiperDispatch implemented new designs within z/OS and PR/SM, and implemented a regular exchange of information between PR/SM and z/OS. This paper provides an overview of the z10 hardware implications that require HiperDispatch, explains the algorithms that existed before HiperDispatch, presents an introduction to HiperDispatch concepts, and describes performance considerations when implementing HiperDispatch.

1.0 Hardware performance concepts of z10

To appreciate the importance of HiperDispatch, several general hardware concepts with z10 must be explained:

- The *book* concept of hardware design was introduced with z990. A book is a physical hardware design that encompasses processor units, memory, and connectors to I/O cages and other books.

  The z10 uses up to four books. The z10 Model E12 has one book, the Model E26 has two books, the Model E40 has three books, the Model E56 has four books, and Model E64 has four books.

- Each book includes a Multi-Chip Module (MCM).

  - The MCM contains the actual processor chips. The z10 MCM contains five processor chips, and each processor chip has up to four actual processors, for a possible total of 20 processors per MCM. However, not all z10 processor chips contain 20 actual processors. Except the E64, the MCM for all z10 models contains only 17 processors[2]. Of these 17 processors, there are 12 processors that can be configured[3] depending on how many processors are purchased, 3 System Assist Processors, and two spare processors.

    The Model E64 MCM contains one book that has an MCM with 17 processors as described. Each of the remaining three books has an MCM with 20 processors for a total of 60 processors on these three books. Of these 60 processors, there are 52 processors that can be configured and 8 System Assist Processors. Thus, the Model E64 has a total of 64 processors (12 in Book-1 plus 52 in Book-2 through Book-4) that can be configured, 11 SAP processors, and two spare processors.

    Each processor has 64KB cache memory for instructions and 128KB cache memory for data. This cache memory is called the *L1* cache.

    Each processor also has an *L1.5* cache. The L1.5 cache is 3MB in size and is used as an intermediate level cache to reduce the traffic between the L1 cache and the L2 cache.

---

[1]

[2] Even though each MCM contains either 17 or 20 processors, not all processors are necessarily enabled. The number of processors that are enabled depends on the amount of capacity purchased with the z10.

[3] Depending on how many processors have been purchased, a processor can be configured as a central processor (CP), a System Assist Processor (SAP), an Internal Coupling Facility (ICF) processor, an Integrated Facility for Linux (IFL) processor, a zSeries Application Assist Processor (zAAP), or a zSeries Integrated Information Processor (zIIP).

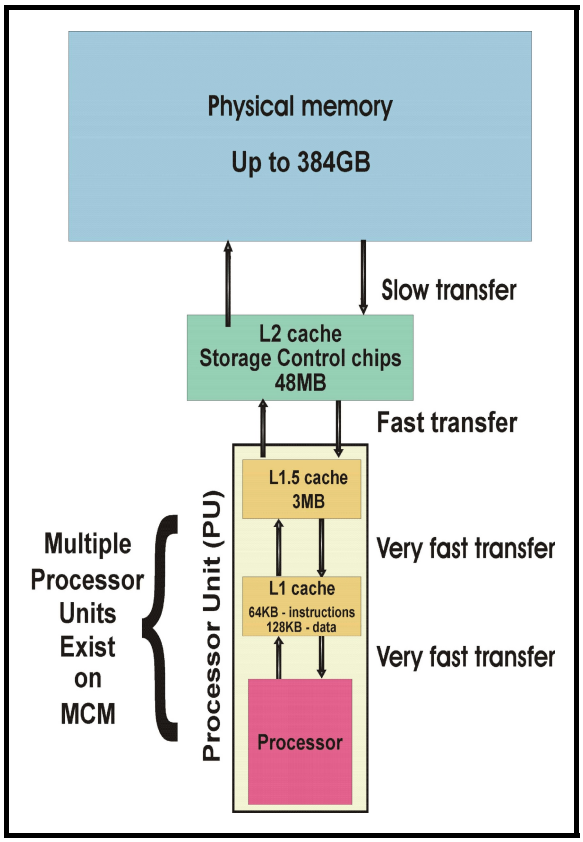
- The MCM also contains two Storage Control (SC) chips. The SC chips provide a total of 48MB of cache, which is shared by all 17 or 20 processors in the book. This cache memory is called the *L2* cache.

- Each book includes up to 384GB of physical memory. The total amount of memory varies, depending on the number of books and the amount of memory purchased on each book. Approximately 1.5TB of memory can be purchased with the z10 (384GB per book * 4 books = 1.5TB).

Processors access instructions or data that reside in the L1 cache. Instructions or data are loaded as needed from L2 cache into the L1.5 cache and then into the L1 cache (as mentioned above, there is separate L1 cache for instructions and for data). Instructions or data are loaded from physical memory into the L2 cache, as needed. The process is reversed if data is changed and must be written to physical memory.

The fastest processor performance can be obtained if instructions and data are contained in the L1 cache associated with each processor. Performance degrades if the hardware must load instructions or data from L2 cache into L1 cache. Performance degrades even more if the hardware must load instructions or data from physical memory into L2 cache. Exhibit 1 illustrates these concepts, showing a single Processor Unit.

Exhibit 1 shows a single processor unit configured in the MCM. However, Exhibit 1 does not make clear that multiple processor units can exist within the book, and that these multiple processor units can individually access L2 cache.



z10 HARDWARE WITHIN A BOOK

EXHIBIT 1

Exhibit 2 illustrates the situation from an overall book view, showing an example five processor units enabled in the book.

The z10 server operates under control of the Processor Resource/Systems Manager (PR/SM) hipervisor. With PR/SM, applications run on logical processors and these logical processors are assigned to physical processors.

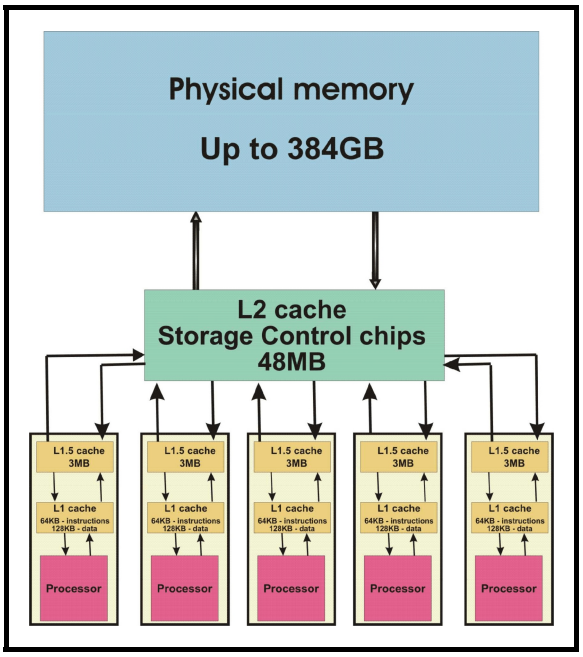
Each application running on any of the processor units has the potential to require that physical memory be loaded into L2 cache, from L2 cache to L1.5 cache on the physical processor in which the logical processor is executing, and from L1.5 cache to L1 cache on the physical processor in which the logical processor is executing.

Multiple books can exist with larger z10 environments. For these environments, an application could execute on one logical processor (with its corresponding physical processor). At the next dispatch of the application, the logical processor might be assigned to a different physical processor, and this physical processor might be in a different z10 book from the one the application was executing on the previous dispatch.

When this happens, required memory might be located in a different book (the book containing the physical processor and associated cache to which the application was previously dispatched). The memory contents must be fetched from the different book (perhaps including a fetch from physical memory to L2 memory in that book), and loaded into L2, L1.5, and L1 memory in the book and physical processor where the application now is executing.

Exhibit 3 illustrates the memory fetch and store interaction in this case.

In the situation shown in Exhibit 3, substantial hardware interaction must occur between the two books and their associated cache. This interaction not only performs the required transfer of data or instructions, but must also ensure memory coherency between the two books and the respective cache. Depending on the nature of the memory access (fetch only or also store contents of memory), substantial interaction with physical memory might be required.



MULTIPLE PROCESSOR UNITS WITHIN A BOOK

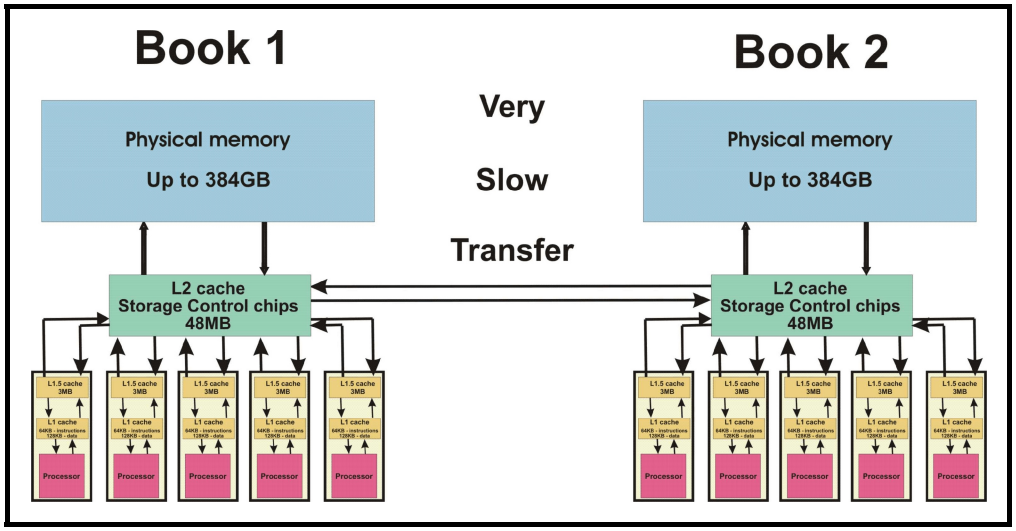
EXHIBIT 2

The various hardware interaction described above causes delays to the fast processors on the z10. These delays occur as a result of program execution, occur as a function of the physical processor on which programs execute from one dispatch to the next dispatch, and occur as a function of the book in which the physical memory resides relative to the physical processor on which the program executes.

- In the best case, the program is dispatched to the same processor as previously dispatched, instructions and data are grouped within small localities of reference (commonly referred to as the "working set" of the program), and the instructions and data reside in L1 cache. In this case, the delay would occur only when the L1 cache was loaded initially, or when new instructions or data would be loaded into the L1 cache.

- In the next best case, instructions and data are grouped within relatively small localities of reference, and the instructions and data reside in L2 cache in the same book as previously used. In this case, the delay would occur only when the L2 cache was loaded initially, or when new instructions or data would be loaded into the L2 cache.

- The problem becomes more acute when there is a poor locality of reference with either instructions or data, and the program is dispatched to a processor in a different book from the one to which previously dispatched. In this case, L1 and L2 cache must be frequently loaded. The result is significant decrease in performance for the fast z10 processors.



z10 MULTIPLE BOOKS

EXHIBIT 3

The hardware performance degradation increases in a *multiprogramming environment* when the z/OS Dispatcher must dispatch different application program code in the normal operation of multiple applications running in a system. The newly dispatched program code might require new instructions or data to be loaded into L1, loaded into L1.5 cache, and perhaps loaded into L2 cache.

With a *multiprocessor environment*, performance degradation could occur even if new program code did not require reloading of the L2 cache. This is because each physical processor has its own L1 and L1.5 cache. Consequently, dispatching a program to a different physical processor could require reloading of the L1 or L1.5 cache of the new processor as instructions or data were needed.

As a result of this hardware interaction, there can be significant delay in executing the instructions with corresponding performance degradation. Putting this degradation into perspective, the basic cycle time of a z10 processor is approximately 230 picoseconds. Up to two instructions may be decoded per cycle and up to three instructions may be executed per cycle, resulting in a very fast processor. According to a presentation at SHARE[4], *only one machine cycle* is required to fetch from L1 cache, but *more than 600 cycles are required to fetch from physical memory*. Depending on the instruction sequence and data being referenced, these substantial delays would be directly translated into performance degradation to the very fast z10 processors.

These performance issues are not new with z10. Several designs have been implemented in MVS, OS/390, and z/OS to minimize the hardware performance degradation[5] caused by dispatch of different program code. The HiperDispatch design is intended to help solve the performance issues that can seriously degrade the inherently very fast performance of z10.

2.0 PR/SM and z/OS before HiperDispatch

There is another potential performance problem before HiperDispatch. This problem occurs because of LPAR definitions that assign an excessive number of logical processors to an LPAR, and occurs because of the algorithm that PR/SM uses to dispatch logical processors to physical processors.

PR/SM allows multiple systems to be defined as logical partitions(LPARs). These LPARs can be defined to use dedicated or shared system Central Processor Complex (CPC) resources (e.g., processors, channels, I/O devices, and memory).

Processors that are assigned to LPARs can be *dedicated* to the LPAR or the processors can be *shared*[6] from a pool of processors. If the processors are from a pool of shared processors, the LPAR's processors are called *logical processors*. PR/SM manages the dispatching of logical processors to physical processors, based on definitions provided for all LPARs using the shared physical processors.

2.1 PR/SM allocation of LPAR capacity to logical processors

If LPARs are assigned shared processors, there is a mechanism that is used to indicate what percent of physical processors in the shared pool should be assigned to each specific LPAR. In the LPAR definition process, *each* LPAR is assigned a *weight* which is simply a numerical value, ranging from 1 to 999. For shared central processors, these weights are used to indicate the percent of processing capacity of the shared central processors that each LPAR is to be given.

The weights for all LPARs that are activated[7] are summed, and used as the divisor in a calculation of the percent of processor capacity that any specific LPAR is to be "guaranteed" by PR/SM.

Equation (1) shows the general concept:

$$\textit{LPAR percent share} = \frac{\textit{LPAR weight}}{\sum(\textit{all LPAR weights})} \qquad (1)$$

To illustrate this concept, suppose *twelve* physical central processors were shared, and there were four LPARs defined with the following weights and logical processors assigned. Exhibit 4 shows the PR/SM computations related to this situation.

---

[4] What's New with z/OS Release 10, John Eells, IBM Poughkeepsie, SHARE 110, Session 2838 February 26, 2008.

[5] For example, the Reduced Preemption algorithms were implemented in MVS/ESA SP3.1 to minimize the refresh of high speed processor cache.

[6] An LPAR may have dedicated physical processors or shared physical processors; except for a coupling facility LPAR with dynamic ICF expansion or enhanced dynamic ICF expansion, an LPAR cannot be assigned both shared and dedicated physical processors with PR/SM.

[7] LPARs can be deactivated or can be activated, based on operator command. The "share" calculation depends on the LPARs that are activated.

LPAR	WEIGHT	SHARE	#CP	#LCP	CP/LCP percent
LPAR1	400	40%	4.8	5	96%
LPAR2	400	40%	4.8	12	40%
LPAR3	150	15%	1.8	3	60%
LPAR4	50	5%	0.6	2	30%
TOTAL	1000	100%	12.0	22	

BASIC PR/SM COMPUTATIONS

EXHIBIT 4

- **WEIGHT and SHARE**. This example illustrates that a specific weight is meaningful only with respect to the total weights assigned to all LPARs. The example shows that LPAR1 and LPAR2 each have a weight of 400, with a resulting share of 40% (400/1000 = 0.40). LPAR3 has a weight of 150, with a resulting share of 15% (150/1000 = 0.15). LPAR4 has a weight of 50, with a resulting share of 5% (50/1000 = 0.05).

  PR/SM "guarantees" the processor share resulting from the weights assigned. Additionally, any LPAR normally can use more than the share derived from the weight, if other LPARs are not using their share of the processing capacity of the central processing complex (CPC). Note that PR/SM normally "enforces" the processor share resulting from the weights *only* when the processors are 100% utilized. Any LPAR normally *can use more than its share* when the shared processors are less than 100% utilized.

- **#CP**. The **#CP** column shows the equivalent number of shared central processors that each LPAR is "guaranteed" based on its specified WEIGHT and computed SHARE. Since there were twelve shared central processors in this example, the processor capacity *SHARE* is multiplied by *twelve* to yield the equivalent number of shared central processors "guaranteed" for each LPAR.

  For example, LPAR1 and LPAR2 each have a 40% "guaranteed" SHARE of shared central processor capacity, which represents "guaranteed" access to 4.8 (0.40* 12 = 4.8) shared central processors. LPAR3 has a 15% "guaranteed" SHARE of shared central processor capacity, which represents "guaranteed" access to 1.8 (0.15* 12 = 1.8) shared central processors. LPAR4 has a 5% "guaranteed" SHARE of shared central processor capacity, which represents "guaranteed" access to 0.6 (0.5* 12 = 0.6) shared central processors.

- **#LCP**. The **#LCP** column shows the number of logical central processors that were defined to PR/SM, for each LPAR. This is an arbitrary number assigned to each LPAR when the LPAR is defined, with the limit that the number cannot be larger than the total number of shared physical central processors.

  For this example, LPAR1 was assigned five logical processors and LPAR2 was assigned 12 logical processors. Perhaps LPAR2 was the most important LPAR, and the site wished it to have access to all shared physical central processors.

- **CP/LCP percent.** The **CP/LCP percent** column shows the percent of one central processor that each logical central processor is entitled to use. For each LPAR, the **#CP** column showed the number of shared physical central processors that the LPAR was entitled to use. However, these shared physical processors must be assigned to logical processors. Consequently, the number of shared physical processors must be divided between the number of logical processors.

  For example, LPAR1 was entitled access to *4.8 physical central processors* as described above. Since LPAR1 was assigned *five logical processors*, the 4.8 physical central processors must be divided among the five logical processors. Consequently, each of the five logical processors assigned to LPAR1 would have "guaranteed" access to 96% of a physical central processor (4.8/5 = 0.96).

Examining LPAR2 in the above example, LPAR2 was entitled access to *4.8 physical central processors* as described above. Since LPAR2 was assigned *12 logical processors*, the 4.8 physical central processors must be divided among the twelve logical processors. Consequently, each of the 12 logical processors assigned to LPAR2 would have "guaranteed" access to only 40% of a physical central processor (4.8/12 = 0.4).

This is a key point from a PR/SM logic view before HiperDispatch: PR/SM algorithms will "guarantee" an LPAR the share of central processor resources that is implied by its weight, but the share is apportioned **evenly to each logical central processor assigned to the LPAR**.

If HiperDispatch is *not active* for an LPAR, PR/SM continues to perform this dispatching algorithm.

It is worth discussing the seemingly illogical situation in which each of the 5 logical processors in LPAR1 is entitled to 96% of a physical processor, while each of the 12 logical processors in LPAR2 is entitled to 40% of a physical processor. If LPAR2 is entitled to 40% of 12 physical processors and only 12 physical processors exist in the CPC, how can LPAR1 achieve its share of 96% of 5 physical processors?

The answer is that PR/SM dispatches a logical processor to a physical processor at a dispatch interval[8]. Once the dispatch interval expires, PR/SM will *intercept* the logical processor and allow a different logical processor to be dispatched to the physical processor. Before HiperDispatch, the 96% usage is spread among more than 5 physical processors. Consequently, even though LPAR2 can use 40% of each of the 12 physical processors, LPAR1 can use 96% of 5 physical processors but the 96% is spread among more than 5 physical processors.

2.2 z/OS dispatching of work before HiperDispatch

As described above, without HiperDispatch PR/SM attempts to apportion each LPAR's share of processor capacity evenly among the logical central processors assigned to the LPAR. z/OS attempts to dispatch a work unit to the last logical processor used by the work unit. If that processor is not available (or if the work had not been previously dispatched), z/OS rotates dispatching of work on its dispatch queue to each logical processor assigned to the LPAR. This rotation is done to ensure that each logical processor in the LPAR gets its equal share of overall processor capacity.

One result of this "equal share" design among PR/SM and z/OS is that some applications can experience elongation of their execution time when the overall LPAR share of processor capacity is divided equally among many logical processors. From a capacity view, any particular application that is serialized in its use of processor capacity can use only the share of capacity that a particular logical processor can use. This is the "short engine" effect that has been discussed by IBM Washington System Center in presentations[9] at various conferences.

If HiperDispatch is not active for an LPAR, z/OS continues to perform this dispatching algorithm.

3.0 HiperDispatch concepts

HiperDispatch Management Mode is activated for an LPAR either by specifying HIPERDISPATCH=Y in the IEAOPTxx member of SYS1.PARMLIB, or under operator control. HiperDispatch can be activated on an individual LPAR basis. However, HiperDispatch Management Mode cannot be effective in an LPAR unless the LPAR has a share of CPC capacity that results in at least 0.5 (the "#CP" value) equivalent physical processors.

There are two operational components of HiperDispatch: the z/OS component and the PR/SM component. These components work together to exchange information about the topology of the logical processors used by the LPARs, and the physical processors used by the logical processors. Each of these components has been enhanced with algorithms that work together to minimize performance degradation to the very high speed

---

[8] The default dispatch interval ranges from 12.5 to 25 milliseconds, depending on how many logical processors have been defined for all LPARs in the CPC.

[9] For example, "Optimal Performance When Running CICS in a Shared LPAR Environment" presented at SHARE Session 1066, August 2004, Kathy Walsh, IBM Washington System Center.

processors operating on the z10, and to optimize the number of logical processors that are used by any LPAR operating in HiperDispatch Management Mode.

3.1 z/OS with HiperDispatch

For its part in the HiperDispatch design, z/OS dispatching design has been changed so (1) the number of logical processors that are active in an LPAR is the *minimum number* of logical processors that are necessary to handle the workload, and (2) any particular work unit is processed repeatedly by the same logical processor or by a small number of logical processors that represent physical processors *in the same book*.

With HiperDispatch, z/OS dynamically manages the weight given to *each logical processor* assigned to an LPAR. This weight is communicated to PR/SM, with the result that PR/SM can treat each logical processor as having a potentially unique share of a physical processor. This dynamic adjusting of *logical processor weights* in an LPAR is done instead of the algorithm prior to HiperDispatch, in which PR/SM has a
fixed weight associated with each logical processor and each logical processor has an equal share of CPC capacity.

To accomplish this, z/OS Dispatcher manages work in multiple *affinity dispatch queues*. Work units in an affinity dispatch queue are assigned an *affinity identifier*, and work units with the same affinity identifier normally will be assigned to the same affinity dispatch queue. z/OS monitors workflow performance, and can switch work units among affinity dispatch queues if necessary to achieve better performance.

One or more logical processors are associated with an affinity dispatch queue. These logical processors can be considered a *logical processor affinity pool*. z/OS interacts with PR/SM to ensure that each logical processor assigned to a logical processor affinity pool corresponds to a physical processor in the *same book* as all logical processors assigned to the logical processor affinity pool.

z/OS assigns a PR/SM weight to each logical processor assigned to an affinity dispatch queue. Each logical processor in a logical processor affinity pool has the same weight.

The weight that z/OS assigns to each logical processor allocated to an LPAR in HiperDispatch Management Mode falls into one of three categories:

- A weight that causes PR/SM to assign 100% of an equivalent physical processor to the logical processor. These logical processors are called **high polarity** logical processors. There can be multiple logical processor affinity pools with high polarity logical processors.

  z/OS can create a high polarity logical processor affinity pool (with an associated affinity dispatch queue) only if:

  - The LPAR has a share of CPC capacity that results in at least 1.5 equivalent physical processors.
  - Two or more logical processors were assigned to the LPAR.
  - The workload requires 1.5 or more equivalent physical processors (that is, the CPU processing requirements of the workload result in using the capacity of at least 1.5 physical processors).

  A high polarity logical processor affinity pool (with an associated affinity dispatch queue) cannot exist unless the above conditions are met.

  If sufficient capacity (weight) was specified for the LPAR, if sufficient processor capacity exists in the CPC, and if sufficient work exists for the LPAR, z/OS Dispatcher will maintain an average of four logical processors in each high polarity logical processor pool. As discussed below, this number of logical processors corresponds to the *physical processor affinity nodes* that PR/SM maintains.

- A weight that causes PR/SM to assign more than 0% but less than 100% of an equivalent physical processor to a logical processor. These logical processors are called **medium polarity** logical processors, and are assigned to the medium polarity logical processor affinity pool.

z/OS *will always maintain* a medium polarity logical processor affinity pool with at least 0.5 equivalent physical processor *share*, but less than 1.5 equivalent physical processor *share* (a high polarity logical processor would be created if the workload demand in the medium polarity affinity dispatch queue was at least 1.5 equivalent physical processors and if the LPAR was entitled to 1.5 or more equivalent physical processors).

The capacity share in the medium polarity affinity dispatch queue essentially is the processor share that remains after any high polarity logical processors have been assigned their 100% share, with the restriction that the medium polarity affinity dispatch queue will always have at least 0.5 equivalent physical processor share. z/OS will assign processor share *equally* among logical processors in the medium polarity logical processor pool.

- A weight that causes PR/SM to assign 0% of an equivalent physical processor to a logical processor. These logical processors are called **low polarity** processors. Low polarity logical processors are not associated with an affinity dispatch queue, but are considered *discretionary* logical processors. As will be discussed later, PR/SM *parks* low polarity logical processors.

  A logical processor will be assigned a weight of zero only if the capacity demands of the workload can be met by high polarity or medium polarity logical processors. If the workload's capacity demand increases and the capacity demand cannot be met by existing high polarity or medium polarity logical processors, z/OS can assign a positive weight to a parked logical processor (effectively, causing PR/SM to *unpark* the processor). The unparked logical processor would be assigned to the medium polarity logical processor pool.

  Recall that logical processors in the medium polarity logical processor pool have an equal share of processor resources. If a logical processor is parked or unparked, the share of the logical processors in the medium polarity logical processor pool would be adjusted so each logical processor[10] had an equal share.

  With HiperDispatch Management Mode active in an LPAR, the VARYCPU function of the Intelligent Resource Director (IRD) is disabled. This is because HiperDispatch will assign weights to logical processors that will cause PR/SM to park or *unpark* logical processors based on workload demand and on the overall weight assigned to the LPAR. The CPU Weight Management algorithms of IRD can still be used to manage weights among LPARs in an LPAR cluster.

The above process normally operates to maintain high polarity logical processors that have 100% share of physical processors, maintain one or two medium polarity logical processors to provide capacity that is less than 150% of equivalent physical processors, and park any remaining logical processors. The HiperDispatch algorithms dynamically adjust the number of active logical processors to the minimum necessary[11] to meet capacity requirements of the workload.

z/OS will attempt to assign work units in an affinity dispatch queue to the same logical processor as previously assigned. If z/OS cannot assign the work to that logical processor, after a suitable delay z/OS will select a different logical processor from the logical processor pool *that is associated with* the affinity dispatch queue. This design ensures that logical processors will consistently be assigned the same work.

z/OS monitors the amount of CPU time used by work in an affinity dispatch queue.

- If the capacity of the affinity dispatch queue is insufficient to handle the workload, z/OS will assign another logical processor to the logical processor pool associated with the affinity dispatch queue (if sufficient logical processors are available to the LPAR), to add additional capacity to the affinity dispatch queue.

- If the workload does not require the capacity of the logical processors assigned to an affinity dispatch queue, z/OS can remove a logical processor from the logical processor affinity pool.

- z/OS will always keep at least one logical processor assigned to the medium polarity logical processor pool.

---

[10] RMF reporting computes the average logical processor share over an entire RMF interval. Consequently, less than equal shares can be shown for logical processors when they are parked/unparked within an RMF interval.

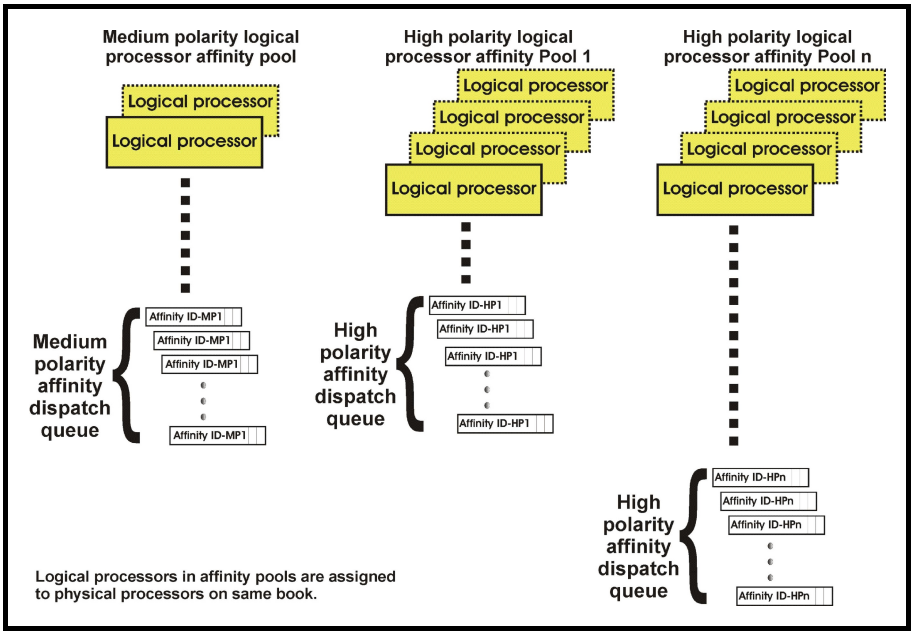
[11] In IBM WP101229, Steve Grabarits refers to this algorithm as maintaining a "working set" of logical processors.

- z/OS interacts with PR/SM when managing the logical processor pools. Each logical processor in a logical processor affinity pool corresponds to a physical processor in a *PR/SM affinity node* of physical processors. As will be discussed later, the PR/SM affinity nodes include only processors in the same book.

This dynamic adjustment of the logical processors assigned to the logical processor affinity pools, and maintaining affinity dispatch queues that are associated with the logical processor affinity pools, ensures that the number of active (unparked) logical processors is the minimum necessary to accomplish the work.

From one perspective, a logical processor affinity pool represents the processor capacity that is available to process all work units that have the same affinity identifier (that is, the work is in the same affinity dispatch queue). Work in an affinity dispatch queue can use only those logical processors in the logical processor affinity pool that is associated with the affinity dispatch queue[12].

Exhibit 5 illustrates this z/OS design.



z/OS AFFINITY DISPATCH QUEUES

EXHIBIT 5

If the available capacity in the existing logical processor affinity pool is exhausted, z/OS can increase the capacity (add another logical processor to the pool from the pool of parked logical processors). If the logical processor pool exceeds its target maximum of four logical processors, and if sufficient capacity and processor resource demand exists, z/OS can establish another logical processor affinity pool, create another affinity dispatch queue, and reassign work units to the new affinity dispatch queue.

As discussed below, PR/SM creates an *affinity node* of physical processors that correspond to the logical processors associated with a logical processor affinity queue. This design ensures that the physical processors will repeatedly process the same work from one dispatch of a logical processor to the next dispatch of the logical processor.

3.2 PR/SM Component with HiperDispatch

PR/SM with z10 establishes *affinity nodes* within a book. Exhibit 6 illustrates affinity nodes for high polarity logical processors.

- These affinity nodes consist of a target average of four physical processors and correspond to the logical processors (for high polarity affinity dispatch queues) that z/OS has created.

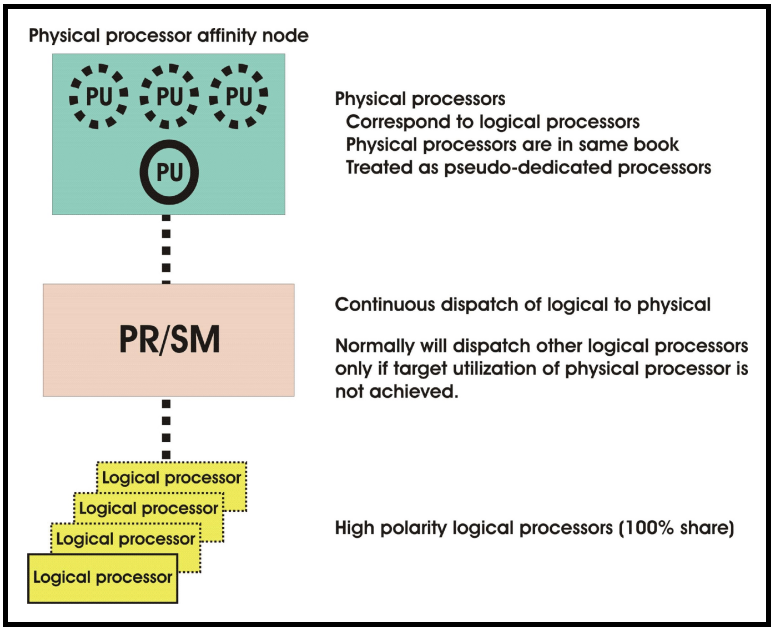
[12] SRB activity in the SYSSTC service class essentially bypasses the z/OS HiperDispatch management algorithms. This work can execute on any available logical processor. As IBM explains in WP101229 "z/OS: Planning Considerations for HiperDispatch Mode", SYSSTC typically supports short-running local SRBs required for transaction workflow in highly interactive work. This work would include VTAM, TCP/IP, and IRLM.

- Physical processors in an affinity node are on the same book.

- PR/SM can dynamically reconstruct affinity nodes among books. PR/SM periodically executes its optimization algorithms that establish primary book assignments for physical processors associated with the affinity nodes.

- PR/SM attempts to dispatch logical processors to the same physical processors previously used. This can help preserve L1 cache.

- If PR/SM cannot dispatch to the same physical processors, PR/SM will dispatch the logical processor to a physical processor in the same affinity node. This can help preserve L2 cache.



PR/SM AFFINITY NODE, HIGH POLARITY PROCESSORS

EXHIBIT 6

- For logical processors having a capacity share that results in 100% equivalent physical processor, PR/SM essentially will dedicate the physical processor to the logical processor.

  The dedicated processor status will exist so long as the workload uses the logical processor (and corresponding physical processor) at full capacity. If the workload demand does not require 100% of the physical processor, excess capacity can be used by other logical processors (thus the term "pseudo-dedicated" physical processor, since the affinity dedication applies only so long as 100% of the processor is used).

- At least one logical processor will have a capacity share that is greater than zero, but is not 100% of an equivalent physical processor. These are the medium polarity processors. Medium polarity processors can be dispatched on any available physical processor. PR/SM will attempt to dispatch the logical processor to a physical processor that was last used. If the last used physical processor is not available, PR/SM will attempt to dispatch the logical processor to a physical processor in the same book as last used.

- PR/SM will *park* logical processors that have a 0 percent share of an equivalent physical processor. PR/SM places these logical processors in a long-term wait state and they are not considered for dispatch by PR/SM algorithms.

  It should be noted that by parking logical processors, PR/SM is simply responding to information from z/OS. Since PR/SM (using LPAR weight information provided by z/OS) handles parking and unparking of logical processors, the VARYCPU management function in IRD is not active when the LPAR is operating in HiperDispatch Management Mode.

4.0 zIIP and zAAP processors with HiperDispatch

If an LPAR is operating in HiperDispatch Management Mode, zIIP and zAAP processors assigned to the LPAR also operate in HiperDispatch Management Mode. z/OS manages affinity dispatch queues for work directed to zIIP or zAAP logical processors, following the algorithms that apply to central logical processors. PR/SM manages affinity nodes for zIIP or zAAP physical processors, following the algorithms that apply to central physical processors. Special care must be taken to ensure that the PR/SM definitions are suitable for zIIP and zAAP logical processors when these logical processors are assigned to an LPAR operating in HiperDispatch Management Mode.

5.0 Performance considerations with HiperDispatch

In most situations, performance will improve for an LPAR operating in HiperDispatch Management Mode. The HiperDispatch algorithms can significantly increase the number of times that programs (1) are re-dispatched to the same physical processor (which would cause L1 and L1.5 cache to be reused more frequently) and (2) are re-dispatched to a physical processor in the same book (which would cause L2 cache to be reused more frequently). Since cross-book communication would decrease, the potentially serious performance degradation of the z10 processors would be minimized. The new z/OS design with HiperDispatch will optimize the number of logical processors active in an LPAR, and thus reduce the performance degradation inherent in a multiprocessor effect.

5.1. Potential performance improvement with HiperDispatch

IBM estimates[13] that performance improvement can range up to 10% improvement in Internal Throughput Rate (ITR). IBM does mention that performance can degrade for some LPARs and can degrade for some applications.

IBM suggests that the amount of performance improvement for an LPAR depends on such factors as:

- **The number of physical processors in the CPC**. Environments with a small number of physical processors will experience little or no improvement (for example, z10 E12 models would experience little improvement with HiperDispatch). This is because there are no cross-book memory communication problems that can exist in a single book, and a big performance advantage of HiperDispatch is to minimize reloading of L2 cache from cross-book communication.

  Environments with a large number of physical processors can experience more significant performance improvement. This is because HiperDispatch can minimize L2 cache reloading from cross-book communication. Additionally, HiperDispatch has more opportunity to create affinity nodes, to maintain logical processor affinity pools, and to more efficiently use affinity dispatching queues to continue dispatching of work to the same logical processors and thus to the same physical processors. This will tend to minimize reloading of L1.5 and L1 cache.

- **The number of logical processors assigned to LPAR**. LPARs with a small number of logical processors assigned will experience little improvement. This is because HiperDispatch cannot make large reductions in the number of active logical processors, and thus cannot significantly reduce the multiprocessor degradation.

  LPARs with a large number of logical processors assigned will experience more improvement. This is because HiperDispatch can normally reduce the number of active logical processors to the minimum necessary to process the work. This will have the effect of maximizing the amount of physical processor capacity used by each logical processor, and will minimize the multiprocessor degradation since there are fewer active logical processors.

- **The logical/physical processor ratio**. Environments with a small logical processor to physical processor ratio will experience less improvement. This is because the amount of PR/SM overhead and PR/SM dispatching delays to logical processors would already be small.

  Environments with a large logical processor to physical processor ratio will experience more improvement. This is because HiperDispatch can normally reduce the number of active logical processors to the minimum necessary to process the work, and park logical processors that are not needed. Using only the number of logical processors actually required to process the work will minimize the amount of PR/SM overhead and will minimize PR/SM dispatching delays to logical processors

- **The size of the locality of memory reference (working set)**. Environments with applications that *do not have* a small locality of reference will experience less improvement. This is because L1, L1.5, and L2 cache must be reloaded frequently regardless of HiperDispatch algorithms.

  Environments with applications that *do have* a relatively small locality of reference will experience more significant improvement. This is because PR/SM will repeatedly dispatch the applications to the same physical

[13] The referenced IBM estimates and suggestions are contained in WP101229.

processor. This action will improve L1, L1.5, and L2 cache hits. Since the physical processors are in the same book, HiperDispatch will significantly reduce the amount of cross-book memory communication.

5.2 Potential performance degradation with HiperDispatch

Depending on the environment, HiperDispatch Management Mode can cause performance degradation to applications. In some cases, the very actions that can improve performance and increase the Internal Throughput Rate can be harmful to individual applications and to LPARs not operating in HiperDispatch Management Mode.

- **Serialization of resource use**. HiperDispatch minimizes the number of logical processors that are active in an LPAR. In doing this, dispatchable work units are serialized on their use of processor capacity, with respect to the logical processors that are assigned to a particular affinity dispatch queue. From a queuing model view, this can mean that the number of "servers" has been decreased, and the queue time waiting for a server can increase.

  In a previous discussion showing the PR/SM weight and share computations, LPAR2 was assigned 12 logical processors (see Exhibit 4). If LPAR2 was not running in HiperDispatch Management Mode, LPAR2 would have 12 "servers" to which work units could be assigned. If LPAR2 was running in HiperDispatch Management Mode, work units in LPAR2 would be assigned to an affinity dispatch queue, to which at most 4 logical processors (or "servers") would be assigned. The result can be that any specific application might wait longer for access to processor capacity when the LPAR is operating under HiperDispatch Management Mode.

  One implication of this increased queue time for a processor is that Workload Manager execution velocity goals or response goals might need to be adjusted. If the wait for CPU increases significantly, the execution velocity goal might not be achievable. Similarly, Workload Manager response goals might not be met if an application waits for access to a processor.

  Additionally, the Workload Manager Goal Importance must be reviewed carefully to ensure that the Goal Importance assigned to a particular Workload Manager service class period correctly matches the relative importance of the Performance Goal as compared with other service class periods. IBM describes an example environment in which the Goal Importance of WebSphere front-end was connected to CICS/DB2 with highly interactive transactions. Initially, the Goal Importance for the WebSphere transactions service class was specified as equal to the Goal Importance of the CICS/DB2 service class. With HiperDispatch, it was necessary for the WebSphere highly interactive transaction service class to have a more important Goal Importance than the CICS/DB2 service class.

  As mentioned earlier, SRBs in the SYSSTC service class essentially bypass the z/OS HiperDispatch management algorithms. This SRB work can execute on any available logical processor, so the serialization of resource use (with potential delays for CPU access) does not apply to these SRBs. As IBM explains in WP101229 "z/OS: Planning Considerations for HiperDispatch Mode", SYSSTC typically supports short-running local SRBs required for transaction workflow in highly interactive work.

- **Repeatability of execution time**. Applications running in HiperDispatch Mode might experience varying execution or elapse time. This is because a particular application might be assigned to a medium polarity dispatch queue in one execution, but be assigned to a high polarity dispatch queue in another execution.

- **Impact of HiperDispatch on LPARs not in HiperDispatch Management Mode**. Without proper planning, running HiperDispatch Management Mode in one LPAR can have serious performance implications for LPARs that are not running in HiperDispatch Management Mode. This is because HiperDispatch can create high polarity logical processors (with 100% share of an equivalent physical processor) and PR/SM will essentially dedicate a physical processor to be used by the logical processor (this is the pseudo-dedicated processor described earlier). This assignment of a pseudo-dedicated physical processor to a logical processor can have serious implications, particularly in small systems.

  For example, consider a z10 Model 703 with two LPARs, each LPAR having three logical processors assigned. Suppose that HiperDispatch is turned on for LPAR1 and that LPAR1 has one high polarity logical processor, one medium polarity logical processor, and one parked logical processor. The high polarity processor is pseudo-dedicated. That leaves two physical processors that can be used by LPAR2.

LPAR2 is not participating in HiperDispatch Management Mode, so its share would be split equally among three logical processors. However, these three logical processors would no longer have access to three physical processors since one of the three physical processors would be dedicated to the high polarity processor on LPAR1.

LPAR2 would have access to only two physical processors (assuming that the workload on LPAR1 is sufficient to drive the high polarity processor to 100% for LPAR1). The three logical processors assigned to LPAR2 would be dispatched to two physical processors in a circular fashion, but no more than two logical processors could be active at any time. LPAR2 could effectively see its available capacity reduced by 1/3. Any work running on LPAR2 could experience significant delays because z/OS would still dispatch work to the three logical processors, but each logical processor would queue waiting its turn for a physical processor.

There are other similar scenarios in which HiperDispatch Management Mode running in one LPAR can have serious performance implications for LPARs that are not running in HiperDispatch Management Mode. These scenarios include an unequal availability of physical processor capacity assigned to logical processors not running in HiperDispatch Management Mode.

6.0 RMF Variables Related to HiperDispatch

From a performance analysis view, SMF Type 70 records provide variables that relate to HiperDispatch Management Mode within an LPAR. Sadly, IBM has chosen to classify "IBM Internal Use Only" the SMF Type 99 (Subtype 12) records that relate to HiperDispatch Management Mode internal algorithms.

Exhibit 7 shows the variables in SMF Type 70 that relate to HiperDispatch Management Mode.

SMF VARIABLE	DESCRIPTION
SMF70PFL	BIT MEANING WHEN SET 0 Content of SMF70UPI valid 1 Group flag 2 Polarization flag. This partition is vertically polarized. That is, HiperDispatch mode is active. The SMF70POW fields in the logical processor data section are valid for CPUs of this partition.
SMF70HHF	BIT MEANING WHEN SET 0 HiperDispatch is supported 1 HiperDispatch is active 2 HiperDispatch status changed during interval 3-7 Reserved
SMF70PAT	Logical processor parked time
SMF70POW	Weight for the logical processor when HiperDispatch mode is active. The value may be the same or different for all shared logical processors of the type described by this PR/SM record.

SMF VARIABLES USED TO EVALUATE HIPERDISPATCH

EXHIBIT 7

7.0 Conclusions

HiperDispatch Management Mode is an elegant design that solves several hardware and software problems that could cause serious performance degradation with z10.

7.1 Large z10 environments. HiperDispatch Management Mode will work best with large environments. There are implementation and performance considerations, but the result of the algorithms should yield substantial performance improvement in large environments.

7.2 Small z10 environments. HiperDispatch Management Mode might not be advisable with small z10 environments (those environments having a single book). This is because the L2 cache will not be affected if all physical processors are within the same book, as would be the case with small z10 environments. It is possible that HiperDispatch could actually cause decreased performance with small environments at moderate utilization levels.

- If HiperDispatch was not active, different applications would run on multiple logical processors (with the matching multiple physical processors). At moderate utilization levels, this could mean that the L1 and L1.5 cache of the physical processors might be more likely to be reused.

- If HiperDispatch minimized the number of logical processors (with the matching reduction in physical processors that were used), different applications would run on the same physical processor. These different applications would use the L1 and L1.5 cache, with the result that the cache of the smaller number of physical processors would have to be reloaded by the different applications.

7.3 Effect on LPARs not running HiperDispatch. LPARs not running in HiperDispatch Management Mode could experience serious performance issues that must be understood and resolved by installations that run some LPARs in HiperDispatch Management Mode and other LPARs not in HiperDispatch Management Mode.

8.0 Acknowledgments

The author would like to recognize the efforts of Al Sherkow (I/S Management Strategies, Ltd.), Nancy C. Roth (Utopia Management Services, Inc.) and Kevin Martin (McKesson) for their help in reviewing this document.

9.0 References

System z9 109 PR/SM Planning Guide (SB10-7041-01)
Chapter 3. Determining the Characteristics of Logical Partitions

z/OS Intelligent Resource Director (SG24-5922)
Chapter 3. How WLM LPAR CPU Management works (Section 3.5: LPAR Weights)

System z10 PR/SM Planning Guide (SB10-7153-00)
Chapter 3. Determining the Characteristics of Logical Partitions

IBM System z10 Enterprise Class Technical Introduction (SG24-7515-00)

IBM System z10 Enterprise Class Technical Guide (SG24-7516-00)

“WLM–Update for z/OS Release 9 and IBM System z10, The Latest and Greatest” (Horst Sinram,WLM Development, IBM Corporation, Boeblingen, Germany), Session 2540, SHARE Orlando 2008

“What’s New with z/OS Release 10" (John Eells, IBM Poughkeepsie), Session 2838, SHARE Orlando 2008.

WP101229 "z/OS: Planning Considerations for HiperDispatch Mode" (Steve Grabarits, IBM System z Performance).

"z/OS Workload Manager, WLM Update for z/OS Release 9 and 10" (Horst Sinram, IBM Germany Research & Development, z/OS WLM Development), Session 2544, SHARE San Jose 2008