

WHITE PAPER: zIIP EXPLOITATION

How Can You zIIP? Getting More From Your zIIP Engines

JANUARY 2008

Peter Morrison

CA NETMASTER DEVELOPMENT

Table of Contents

Executive Summary

SECTION 1 **Why zIIP?** 2

SECTION 2 **Exploiting the zIIP** 2

SECTION 3 **TCB and SRB Mode** 3

SECTION 4 **Issues to Consider for zIIP Exploitation** 4

SECTION 5 **Next Steps** 6

SECTION 6 **About the Author** 6

ABOUT CA **Back Cover**

Executive Summary

Challenge

Total cost of ownership (TCO) is on everyone's mind, so the introduction of specialty engines by IBM looked particularly attractive to enterprise customers with large mainframe farms. The cost of specialty engine MIPS was so attractive, as were the software licensing rules which stated that measured usage would only include general processor capacity. But how do you offload more work to the specialty engines?

Opportunity

IBM made it relatively easy to benefit from zAAPs (aimed at JAVA workloads), but the zIIPs only seemed to offload some DB2 work. There had to be more.

And there is. By understanding the nature of work being processed, non-DB2 workloads can be examined to determine their fitness for zIIP exploitation. In fact, software vendors are beginning to look at this challenge, responding to customer demand for lower processing costs and better return on investment (ROI).

Benefits

The effort of making code zIIP-exploitative can really pay off for some kinds of work, particularly monitoring and tracing products that add great value in managing and monitoring business processes, but do not directly contribute to the bottom line. And when software vendors exploit the zIIP, the potential benefits to the customer are both increased general processor capacity, as well as reduced software license costs.

SECTION 1

Why zIIP?

The new IBM System z Integrated Information Processor, also known as a zIIP, offers a high-speed specialty engine that can significantly reduce the costs associated with mainframe computing — if you know how to exploit its capacity.

For the past 20 years, the perceived cost of running business workloads on a mainframe drove companies to consider alternative hardware platforms employing operating systems such as UNIX, Windows and Linux. New workloads such as IBM WebSphere, which encouraged the pervasive use of CPU-intensive Java-based applications, required increases in mainframe capacity but businesses were reluctant to invest so heavily in new mainframe hardware.

Today, judicious exploitation of zIIP processors can increase effective capacity by offloading eligible work, thereby reducing the need for costlier upgrades and improving total cost of ownership (TCO).

Like the zAAPs (Application Assist Processors for Java code execution, XML parsing, etc.) and IFLs (Integrated Facility for Linux execution) that preceded them, zIIPs can be purchased for a one-time charge per engine, which includes no-charge replacement by faster zIIP engines when upgrading to a new machine. The cost of zIIP hardware capacity is approximately one quarter of an equivalent CP capacity. This means any infrastructure or application functions that can be diverted to a zIIP run at a much lower cost. Moreover, for customers with z9 BC series machines, the zIIP engines always deliver their full hardware capacity even if the CP engines are configured to run at less than full capacity, sometimes referred to as having been “crippled” or “knee-capped.”

Monitoring and systems management software are perfect candidates for zIIP exploitation. Executing these functions on a zIIP frees CP capacity for use by business applications and other non zIIP eligible software — and it costs less. Given that many monitoring products run at a higher priority than the subsystems they monitor, offloading the processing to zIIP can also mean less interference with business-critical work.

The zIIP engines provide software savings as well. Most software is priced based on the capacity represented by or the utilization of the configured general purpose (CP) engines, independent of any speciality engines. Consequently, any work moved off the CPs is free, from a software perspective. Such offloads may well defer processor upgrades — and their attendant software costs — for months, if not years. This strategy significantly improves mainframe price-performance, further reducing the platform’s TCO.

SECTION 2

Exploiting the zIIP

Clearly the goal is to redirect as much work as possible to specialty engines, ideally deferring CP capacity upgrades as long as possible.

In early 2007, CA announced zIIP exploitation features for eight of its products. Since then, IBM has extended its zIIP exploitation to include products such as the z/OS TCP/IP stack. Other software vendors may be considering zIIP exploitation as well.

To date, the software products that offer zIIP-specific functionality perform one of these three functions:

1. Assist in determining the value of purchasing one or more zIIP engines prior to their physical installation and/or activation.
2. Monitor zIIP utilization and effectiveness.
3. Exploit specialty engine capacity by enabling selected functions to execute on zIIP engines.

This article focuses on products that perform the third function.

IBM designed the z/OS zIIP enablement functions so that only certain types of software could exploit it. The specific API needed to make work eligible for zIIP execution is disclosed only to software vendors who have signed a special license agreement with IBM to obtain the information. Even so, IBM provides much information about the functional nature of zIIP support on its web site and in customer consumable documents. For example, IBM offers the following:

“The zIIP is designed so that a program can work with z/OS to have a portion of its enclave Service Request Block (SRB) work directed to the zIIP.” The DB2 work listed above executes in enclave SRBs, a portion of which is eligible for zIIP. DB2 stored procedures and user-defined functions do not use SRBs and so are not eligible. The z/OS Communication Server IPsec work listed above is planned to interact with z/OS Workload Manager to have all enclave Service Request Block (SRB) work associated with IPsec processing directed to zIIP.”¹

The important part of this description is highlighted. It tells us **software executing in SRB mode can be made to execute on a zIIP**. To a great extent, that’s all there is to it. Of course, execution in SRB mode is not for the faint-hearted, as will be made clear in this article.

SECTION 3

TCB and SRB Mode

To understand what workloads can run on a zIIP, it helps to understand the conditions under which this is possible. In the z/OS environment, a program can execute in one of two modes:

1. TCB mode, usually referred to as task mode
2. SRB mode

Most programs — and all non-privileged programs — execute under the control of a task. Each thread of execution is represented by a Task Control Block (TCB). A program can be broken into multiple tasks, allowing exploitation of multiple processors. Task mode programs can perform I/O, wait for events and use all manner of system services.

SRBs are very lightweight and efficient threads of execution that are available only to supervisor state software. SRB mode is used by operating system facilities and vendor programs to perform certain performance-critical functions. For example, when an I/O interrupt occurs, z/OS dispatches an SRB to the relevant address space to post an ECB and possibly perform other processing.

¹ IBM System z9 Integrated Information Processor (zIIP) <http://www-03.ibm.com/systems/z/zIIP/about.html>

Unlike TCB mode, there is no data structure representing the SRB while it is executing. SRBs also run with high priority, either at the (local) address space level or at a global system level. Both global and local SRBs can be interrupted and suspended for page faults and certain locks. If the SRB is suspended, the operating system stores the SRB's state temporarily, but otherwise this cost is never incurred by an SRB.

Global SRBs can be interrupted, but once dispatched, they are usually not preempted. Once the interrupt is serviced, the SRB resumes processing. A loop in a global SRB can be fatal.

As a historical side-note: because SRBs can be executed in any address space, they were the only way to access data in other address spaces prior to the introduction of cross-memory services in 1983.

In the mid-1990s, MVS/ESA 5.2 added new capabilities that made SRB mode more useful. That release introduced preemptible SRBs that compete for resources at a similar priority to TCBs. Client SRBs were also introduced, where the CPU time expended could be charged to a different address space than the one where the SRB executed. Also, with the advent of the Workload Manager “enclave” capability, SRBs could be grouped together with related task mode work and treated as a single entity for scheduling and workload management purposes. With increased use of stacking PC interfaces to many services, it became possible to write more sophisticated SRBs since, unlike the legacy SVC-based services, most z/OS supplied PC-based service functions are usable by both SRB and TCB mode programs.

Even with these changes, SRB mode remained largely unused outside of the operating system and key middleware products such as DB2. One reason is quite simple — execution in SRB mode requires running in supervisor state, and many programmers felt this raised the bar too high and so resorted to SRB mode only when absolutely necessary. SRB mode functions were also difficult to test and debug. Consequently, almost all product code written for the z/OS environment today runs in task mode, with SRB mode being employed only where absolutely essential.

SECTION 4

Issues to Consider for zIIP Exploitation

While it is theoretically possible to run many systems, middleware and infrastructure-related functions on a zIIP, there are practical matters that limit one's ability to do so. For example, Independent Software Vendors (ISVs) must consider issues such as:

- Does the product code currently run in privileged mode (APF, supervisor state, or a system key)? If not, adapting it to support privileged mode execution (a pre-requisite for SRB enclave use) may be too difficult a task to consider this code for zIIP exploitation.
- Does the product function consume significant CPU cycles? If not, the effort required to make it zIIP-eligible may not be worth it.
- Does the product code frequently invoke functions (such as data set I/O) that are difficult to accomplish in SRB mode? If so, it may not be worth a redesign to relocate just the eligible portions of processing to a separate process (independently dispatchable unit of work).

To be zIIP-enabled, software code (ISV or developed in-house) must be:

1. Reliable and stable. SRB-mode code can be dangerous and challenging to debug, so it's wise to start with reliable code.
2. Able to execute entirely in supervisor state. The APIs needed to schedule execution in SRB mode require this state and SRBs can execute only in supervisor state. Note that this does not necessarily mean key 0. In fact, in some implementations most execution is in another system PSW key, even while in SRB mode.
3. Loaded from an APF-authorized library and, depending on its overall addressability requirements, it may also need to be loaded into common storage and potentially page-fixed to avoid page faults. Note: code that already runs APF-authorized is likely to have dealt with these issues.

In addition, there are limitations that require switching back to TCB mode to perform certain functions that are available only to tasks. Among these are:

- ECB wait
- STIMER and STIMERM services

Some workarounds can circumvent these restrictions. For example, "ECB wait" can be emulated from within an SRB through the creative use of z/OS extended ECBs, which allow for the definition of an exit that's taken when a POST is directed to the ECB. This facilitates the replacement of standard WAIT processing with an approach that will work in SRB mode.

Basically, the approach is:

- Allocate a Pause Token for use by WAIT
- Register a POST Exit
- Define an ECB Extension that points to the POST Exit
- When a WAIT request is received, store the address of the ECB extension and set the high-order WAIT bit into the ECB. If the WAIT specifies a list of ECBs, repeat once per ECB. Use Compare-and-Swap (CS) to ensure proper serialization across tasks. If an already-posted ECB is detected, skip the following two steps (PAUSE and the POST exit)
- Issue PAUSE against the token allocated above
- When a POST occurs, the POST exit will be driven. At this point, simply issue PAUSE RELEASE and the waiting task will be resumed
- After waking from PAUSE, reset the WAIT bit to zero in the ECB (or each ECB, for WAIT with ECBLIST), again using Compare-and-Swap for proper serialization. (This is what occurs if a post-ahead was detected)

That's it. If the above steps are followed, the standard POST logic will work and the application can call this wait service in SRB mode. Note that the above approach handles the case where a single event must be posted. If you need to wait for multiple ECBs to be posted, the logic is a bit more complicated (and is left as an exercise for the reader).

For STIMER and STIMERM services, one can use the z/OS-provided SETDIE service, which sets and manages real-time intervals. This service manipulates the low-level timer element queue used by the timer interrupt handler. When a timer pop occurs, the WAIT/POST logic and ECBs described above can be used to signal timer expiration. This provides an alternative to STIMER(M) for SRB mode callers.

The requirements for SETDIE are onerous. One must fully understand the IBM documentation on this service prior to using it, failure to do so will likely result in dire and unintended consequences such as a complete z/OS system failure. However, with proper research and care a service that reliably manages one or more timers for SRB mode callers can be implemented.

SECTION 4

Next Steps

Many people may conclude that this kind of highly specialized coding effort is not feasible because they lack either the required time or expertise — or both. Even in cases where time and expertise are not the limiting factors, the cost of designing and implementing the required code changes may not be justified. However, there is clearly value in encouraging your ISVs to offload as much of the functionality of their products to zIIP engines as is reasonable. By doing so you can further improve mainframe TCO by exploiting the zIIP specialty engines while concurrently optimizing software license fees.

Remember, the zIIP specialty engines were not designed to be used exclusively by DB2, or by IBM products. Talk with your software vendors as well as IBM to determine how this new mainframe hardware capability can be leveraged to improve the TCO of your IT operation.

Specialty engines like the zIIP are an effective tool for maintaining the mainframe's competitive position relative to alternative and arguably less capable enterprise-class computing platforms. Such innovations, coupled with emerging technologies such as cell processors, continue to enhance the value of the mainframe, which remains the most reliable, secure and cost-effective platform for the concurrent hosting of diverse, bet-your-business applications and data.

SECTION 5

About the Author

Peter Morrison has been in the IT industry for 30 years. In that time he has worked in various roles, including Applications and System Programming, Database and Data administration. He has been involved with CA's NetMaster product family for the last 21 years, and is presently a senior architect in the CA NetMaster development lab.

CA, one of the world's largest information technology (IT) management software companies, unifies and simplifies complex IT management across the enterprise for greater business results. With our Enterprise IT Management vision, solutions and expertise, we help customers effectively govern, manage and secure IT.

WP05MFMNZE01E MP324680108