From A to zIIP

Leveraging the IBM speciality processor for mainframe cost savings

by Craig Mullins

cloudframe

From A to zIIP Leveraging the IBM Specialty Processor for Mainframe Cost Savings

By Craig Mullins

Executive Summary

Specialty processors are an IBM enhancement introduced to mainframe users in the early 2000s. These processors augment a mainframe's general processor, allowing organizations to shift certain workloads off the GPU. Why do this? While there are many reasons, one rises above all others— cost savings.

Mainframe billing can be complex and nuanced, but a general rule is that a significant portion of the cost is associated with peak usage rates on the general processor. Specialty processors aren't just cheaper to use; they are cheaper to acquire. The catch, however, is that the workloads that can use a specialty processor are highly defined and limited.

One of the specialty processors available is the Integrated Information Processor or zIIP. Like all specialty processors, there are restrictions on the workloads that can be redirected to the zIIP, and not all eligible applications will run on this processor. However, taking the time to understand which can shift workloads to the zIIP could potentially save your organization hundreds of thousands of dollars.

Allowable zIIP applications include IBM Watson AI services and workloads ranging from Db2 to Java. While moving eligible workloads makes sense, it can also be extraordinarily valuable to refactor legacy COBOL code to Java to take advantage of the zIIP processor.

However, anyone considering such refactoring is aware that it can be a challenge in and of itself. Finding the right tools and process for refactoring is critical. Many automated refactoring applications can't guarantee exact data output matches or result in a Frankenstein-like mishmash of COBOL and Java. Transpiling and cross-compiling applications with CloudFrame can eliminate these challenges, reducing time and testing and resulting in highly maintainable and data-identical code that can be shifted to the zIIP.

Of course, without identifying the right workloads to shift, proper planning, and a thorough understanding of how your organization's mainframe billing works, mistakes can still occur. However, by following some practical tips and best practices and utilizing the right tooling, your organization can realize significant savings without incurring massive technical debt or being mired in complex refactoring initiatives.

Contents

Understanding Mainframe Specialty Processors: zIIPS and More

If you are an IT professional who works on IBM z Series mainframes, then you've probably heard about zIIPs and other "specialty processors." But you may not know what they are, what they do, and why they exist. With that in mind, let's take a brief journey into the world of specialty processors.

Starting in the early 2000s, IBM began introducing several different types of specialty processors. The basic idea of a specialty processor is that it augments the main general-purpose CPUs. Instead of running all workloads on the general-purpose CPUs, specific workloads are shuttled to the specialty processors for execution.

Why is this useful or interesting to mainframe customers? Mainframe pricing and licensing are complex and can be pretty confusing. The specialty processor workload is not subject to IBM or (for the most part) independent software vendor (ISV) licensing charges. As anybody who has ever looked into mainframe software pricing knows, software cost can be many multiples more expensive than the hardware cost. Still, at a high level, your organization's monthly mainframe software bill is based on the peak average usage during the month.

Most mainframe software contracts are tied to the processor size of the machine on which the software is to be run. The cost of the software rises as the capacity on the mainframe rises. But if capacity can be redirected to a specialty processor, that workload is not factored into the software license charges. If enough workload can be redirected to specialty processors, meaningful cost savings can be realized.

If enough workload can be redirected to specialty processors, meaningful cost savings can be realized.

Another benefit of the specialty processors is that they are significantly cheaper than general-purpose processors. A standard mainframe CP can cost more than half a million dollars, whereas the list price of a specialty processor is about a quarter of the cost… and the street price of a specialty processor can be much less.

Specialty processors can be purchased for a one-time charge per engine, including no-charge replacement by faster zIIP engines when upgrading to a new machine. So many organizations today are augmenting their mainframes with specialty processors to delay costly upgrades.

But, of course, there is a catch! The specialty processors can only run certain types of workloads. There currently are three different types of mainframe specialty processors:

- ICF: Internal Coupling Facility — used for processing coupling facility cycles in a data-sharing environment.
- IFL: Integrated Facility for Linux — used for processing Linux on System Z workload on an IBM mainframe.
- zIIP: Integrated Information Processor — used for processing specific types of distributed workloads.

There used to be a fourth type of mainframe specialty processor, the zAAP, or Application Assist Processor. Its usage was designed specifically for Java workloads and XML parsing. However, late in 2009, IBM provided zAAP workloads to run on the zIIP, enabling organizations to run zIIP- and zAAP-eligible workloads on a single type of specialty processor, the zIIP.

The ICF and IFL are designed for specific types of workloads, dedicating coupling facility workload in the case of the ICF and processing Linux workload in the case of the IFL. By running these types of workloads on a specialty processor, the work will not apply to your monthly IBM software charges. The cost benefit is quite straightforward for these specialty processors. Although the zIIP offers a similar benefit, many nuances must be understood and considered.

Let's Talk About the zIIP

The zIIP is a dedicated processor designed to operate asynchronously with mainframe general processors (GPs). When you activate zIIP processors, some percentage of the relevant workload can be redirected off the general processors onto the zIIP specialty processor. The primary benefit of redirecting work to the zIIP is that IBM will not impose software charges on workloads that run on the zIIP.

Careful readers will note the phrase "relevant workload" in the previous paragraph. Not everything can run on the zIIP, only workloads that IBM deems as "new" are permitted. Originally, the zIIP was designed to support redirecting newer Db2 functionality, but over time the list of what is considered "new" by IBM has grown. At a high level, the current zIIP-supported workloads include Java application programs, IBM z/OS Container Extensions (zCX), IBM Watson Machine Learning for z/OS, IBM z15 System Recovery Boost, and some types of Db2 processing (e.g., XML, distributed queries, and some utilities). Other ISVs also have zIIP-enabled products, which enables portions of those workloads to run on zIIP processors.

There are limits to your usage of zIIPs that must be understood. First, there are limits on the number of zIIPs that can be installed. Originally, there could be no more than one zIIP per GP in a central processor complex (CPC). Today, some models allow two zIIPs per GP. Second, IBM's license agreement restricts the kind of code that is eligible to run on a zIIP; the code must run in a z/OS enclave under the control of an SRB (service request block).

Additionally, not all zIIP-eligible workloads will run on the zIIP. It can be troublesome to understand exactly what, when, and how much of the workload is being redirected. Nevertheless, the primary intent of the zIIP is to reduce your IBM software charge, and the more workload that can be redirected to the zIIP, the more your monthly cost savings can be.

The primary intent of the zIIP is to reduce your IBM software charge

Synopsis

Specialty processors are here to stay, and they can be used to help reduce your monthly IBM software license charges and thereby reduce the cost of mainframe computing. Although specialty processors introduce some complexity into management and capacity planning, organizations can benefit from exploiting them. IBM and ISVs continue introducing new offerings and functionality that can run on zIIPs, enabling organizations to utilize specialty processors for more varied workloads.

Knowing you can use the zIIP and identifying the right workloads requires understanding what zIIP eligible and workload redirection mean.

Digging Into the zIIP: What Does zIIP Eligible Mean?

As noted, the zIIP is an "information" processor, where the IIP in its name stands for Integrated Information Processor. When the zIIP was introduced by IBM in 2006, Db2 Version 8 was the first subsystem to take advantage of the zIIP. Over the ensuing years, IBM (and other vendors) have enabled additional workload eligible to be redirected to the zIIP. It stands to reason that the zIIP will be attractive to Db2 users.

That last sentence brings up two terms that need to be defined: eligible and redirect. Workload is **zIIP eligible** when it runs in an enclave SRB. While we'll delve into this more in later sections, it is essential to understand that only certain types of workloads are eligible for running on the zIIP. And there is the term "**redirect**." Simply because a workload is zIIP eligible does not mean it will run on the zIIP. The workload must be redirected from the general-purpose CPU to the zIIP, meaning that the system tries to run the workload on the zIIP, which may or may not happen.

Only specific types of workloads are eligible to be redirected to zIIPs. Let's consider those.

The first set of zIIP eligible workloads to consider is from a **Db2 for z/OS** perspective. The following types of Db2 workload can benefit from zIIPs:

- **Remote SQL requests using DRDA** to access Db2, including JDBC and ODBC access to Db2, and **native REST calls** made over HTTP. This includes native SQL stored procedures that run over a DDF distributed connection. Up to 60% of the workload for these Db2 SQL requests is eligible for redirection to the zIIP.
- **Parallel query operations** (as identified by the Db2 Optimizer) are typically used for complex business intelligence query processing like star-schema parallel query. Up to 100% of parallel query processing is eligible for redirection to the zIIP but only after reaching a CPU usage threshold. IBM defines the CPU usage threshold for each type of IBM Z system.
- **XML processing** performed by Db2, including up to 100% of XML schema validation and non-validation parsing; and up to 100% of the deletion of unneeded versions of XML documents.
- **Certain IBM Db2 utility processing** that maintains index structures. As much as 100% of the portion of the IBM LOAD, IBM REORG, and IBM REBUILD INDEX

utility function is used to maintain index structures and some portions of the IBM RUNSTATS utility processing, including column group distributed statistics processing.

- Up to 100% of processing for **Db2 system agents processing** under enclave SRBs (service request blocks) that execute in the Db2 master (MSTR) address space, the Db2 database services address space (DBM1), and the Db2 DDF address space (DIST). However, P-Lock negotiation processing is not eligible for redirection to zIIPs. To be clear, this includes many Db2 buffer pool processes such as prefetch, deferred write, page set castout, log read, and log write processing. Additional eligible processes include index pseudo-delete and XML multi-version document cleanup processing.

Even though the zIIP was initially designed for Db2 workload, IBM has made several other types of workloads eligible to be redirected to zIIPs. Perhaps the most significant of these additional workloads is **Java application programs**. Applications written in Java using IBM MQ as a Java client, or using IBM WebSphere Application Server and z/OS MF can redirect workload to zIIPs.

Java is one of the world's most popular programming languages, especially for developing enterprise applications in large organizations. It consistently ranks near the top of the TIOBE index, ranking programming language popularity. Furthermore, many organizations are modernizing their legacy applications to use Java as a part of digital transformation efforts. The ability to redirect Java workload from general-purpose CPUs to zIIPs can provide significant cost savings to organizations with heavily used Java applications.

Additional types of workloads that are zIIP eligible include:

IBM z/OS Container Extensions (zCX), which enable the deployment of Linux applications as Docker containers on z/OS as part of a z/OS workload, are eligible to be redirected to zIIPs. IBM zCX is another key contributor to the legacy modernization efforts of many organizations.

Organizations embracing AI can take advantage of the zIIP eligibility of **IBM Watson Machine Learning for z/OS and Apache Spark** for their AI and related workloads, which are growing in most large organizations.

Finally, zIIPs provide a significant boost to IBM Z system resiliency. The IBM **System Recovery Boost** function on the IBM z15 (and later) will utilize zIIPs as part of its processing to reduce the time needed to shut down and restart a system for outages. Furthermore, IBM **z/OS Communications Server** exploits the zIIP for portions of internet protocol security (IPSec) network encryption and decryption, as well as for select HiperSockets large message traffic. And IBM **z/OS Global Mirror** delivers DFSMS System Data Mover processing for zIIPs.

Some third-party independent software vendors (ISVs) have zIIP enabled some of their products, too. Check with your ISV software providers on what zIIP capabilities and plans are available for the products you use.

The Bottom Line

Multiple types of processing can run on the zIIP, helping to reduce the cost of your monthly IBM software bill. Remember that even with zIIPs installed, all potential workloads will not be redirected to the zIIP – only a percentage of it. Some people refer to the amount of workload that can be redirected as the IBM "generosity factor."

Now that you understand the zIIP better, it's time to look closely at what makes a workload eligible. You'll then better grasp what zIIPs could mean for your organization.

Types of Processing That Can Utilize zIIPs & Why You Want to use zIIPs

Let’s dig a little deeper into what makes a workload eligible for running on zIIPs, and look at why you would want to exploit zIIP processors for your workloads.

TCBs and SRBs

To fully comprehend what can and cannot run on a zIIP, we need to discuss TCBs and SRBs. Many Db2 DBAs and performance analysts first heard about TCBs and SRBs in an IBM performance class, but not everyone has taken one of those classes. And even for those who have, a refresh is probably in order.

For mainframe z/OS programs, code can execute in two modes: TCB mode, also known as task mode, or SRB mode. Most programs execute under the control of a task. Each thread is represented by a TCB or Task Control Block. A program can exploit multiple processors if composed of multiple tasks, as most programs are.

An SRB, or Service Request Block, is a control block that represents a routine that performs a particular function or service in a specified address space. SRBs are lightweight and efficient but have some limitations. Although an SRB is similar to a TCB in that it identifies a unit of work to the system, an SRB cannot “own” storage areas. SRB routines can obtain, reference, use, and free storage areas, but a TCB must own the areas. Operating system facilities and vendor programs typically use SRB mode to perform certain performance-critical functions.

In general, z/OS will dispatch Db2 work in TCB mode if the request is local or in SRB mode if the request is distributed. These parallel tasks are assigned the same importance as the originating address space.

Preemptable enclaves are used to do the work on behalf of the originating TCB or SRB address space. An enclave is a construct that represents a transaction or unit of work. Enclaves are a method of managing mainframe transactions for non-traditional workloads. You can think of an enclave as an anchor point for resource accumulation regardless of where the transaction is executing.

It is relatively easy to map the resources consumed to the actual transaction doing the consumption with traditional workloads. But with non-traditional workloads – web transactions, distributed processing, etc. – it is more difficult because the transaction can span platforms. Enclaves are used to overcome this difficulty by correlating more closely to the end user’s view of the transaction.

So even though a non-traditional transaction can comprise multiple "pieces" spanning many server address spaces and can share those address spaces with other transactions, the enclave gives you more effective control over the non-traditional workload.

Enclaves are grouped by common characteristics and service requests, and since they are preemptable, the z/OS dispatcher – and Workload Manager – can interrupt these tasks for more important ones. There are two types of preemptable SRBs: client SRBs and enclave SRBs.

From a Db2-perspective, if the request is distributed DRDA workload, then it will be executed in enclave SRBs. If the request is coming over a local connection, then it will be dispatched between TCBs, client SRBs, and in some cases, enclave SRBs (such as for parallel queries and index maintenance).

An SVC, or a supervisor call instruction, is a processor instruction that directs the processor to pass control of the computer to the operating system's supervisor program. Because zIIPS must run under an SRB, many commonly used z/OS services (used by TCBs) are not available; specifically, SVC calls other than ABEND. But we are getting deep into the weeds here, and these detailed nuances are more important for software engineers writing code for zIIPs than those using them.

Why use zIIPs?

Let's take a moment to circle back and answer the fundamental question: why should I use zIIPs? And the simple answer is cost reduction.

When work is redirected to, and then runs on, a zIIP instead of a general-purpose CP, that workload is not included in the MSU metrics for MLC software charges. Now that was a mouthful, so let's ensure we understand what we're talking about. First, the term MSU is an acronym for million service units. MSU has replaced MIPS (Millions of Instructions Per Second) as the standard measurement for mainframe capacity and consumption (see MSU versus MIPS). An MSU is a measurement of the amount of processing work that can be performed in an hour. One "service unit" originally related to an actual hardware performance measurement, but that is no longer the case; a service unit is an imprecise measurement. Nevertheless, IBM publishes MSU ratings for every mainframe model, so MSUs are used for modern mainframe capacity and workload measures.

Why use zIIPs? — the simple answer is cost reduction

The next term mentioned above without definition was MLC, which stands for Monthly License Charges. This refers to a specific category of mainframe software that is billed and paid for every month. Your organization will produce a report of each month's consumption and submit it to IBM. This report dictates your IBM software bill, based on usage, for your IBM MLC products. Some of the most common MLC products include z/OS, Db2, CICS, IMS, MQSeries, and COBOL. Specific pricing and terms and conditions for IBM MLC products are based on the pricing metric in your IBM contract(s). There are more details behind the scenes, but this is a sufficient overview for now.

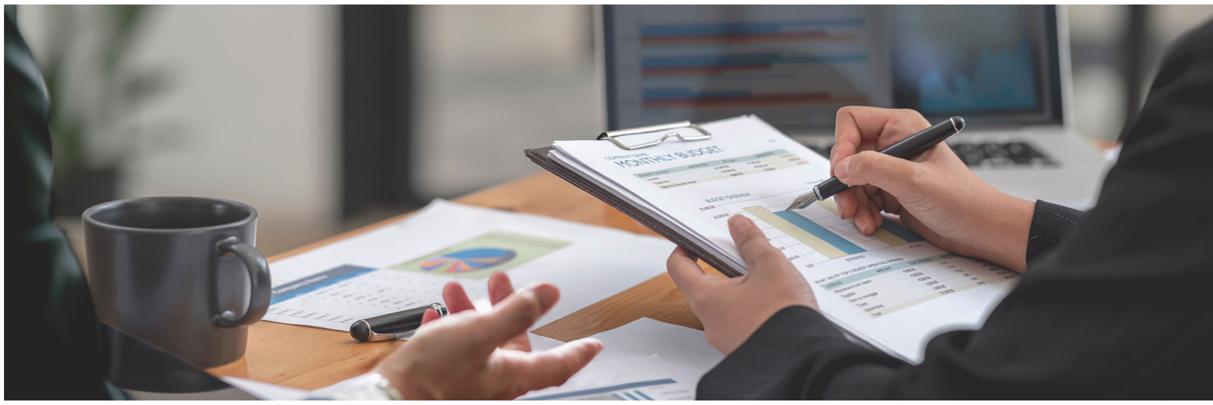
Monthly Licence Charges

So, what does all of this mean? Well, the workload that gets run on zIIPs does not get counted on your monthly processing capacity. Therefore, your software bill can be reduced, perhaps significantly, by using zIIPs. It would be remiss not to mention that there is also a hardware cost reduction because a zIIP costs considerably less than a general-purpose CP. So, when you use zIIPs to run workloads, they are being run at a reduced hardware cost.

An additional consideration on why you should use zIIPs is that, at times, they can provide a performance gain. Depending on the type of mainframe you use, the general-purpose CP may be kneecapped, meaning that processing power is artificially constrained. But the zIIP(s) you add to your system are never kneecapped. Consequently, the workload redirected to the zIIP may outperform the same workload if it were to run on the general-purpose (kneecapped) CP.

There are many factors to consider when understanding zIIPs and determining why and how best to use them. But, once you have a better understanding, you recognize they can play a valuable role in reducing or avoiding mainframe cost. Of course, as we'll see, cost savings is just the tip of the iceberg regarding the benefits zIIPs offer, especially when used with Java applications.

Common MLC products include z/OS, DB2, CICS, IMS, MQSeries, and COBOL



Java and the zIIP: The 5 Major Benefits

Java applications are one of the most important workload types that are zIIP eligible. These applications are inherently new compared to existing legacy mainframe code and therefore qualify for zIIP usage and offer a host of other benefits.

Before we delve into the benefits of Java, let's face it, the predominant language used by most mainframe applications is COBOL. According to Reuters, there are over 220 billion lines of COBOL code running production workload. That's a lot! But there are issues with COBOL as it is not taught in university computer science curricula, is a procedural language that is no longer in vogue, and is quite verbose. That said, lots of COBOL code exists and continues to work well.

Nevertheless, not much new work is being done using COBOL. Many new mainframe applications are written in Java, perhaps due to the following benefits.

Benefits

1. Reduced Cost
2. Easier Support
3. Code Maintenance
4. Speed?
5. Portability

1 — Reduced Cost

The first, and most important benefit, is that using Java can help you reduce your monthly IBM software bill. Java workload is zIIP eligible, and any Java workload that gets redirected to run on the zIIP will not be chargeable against your monthly bill.

As discussed in earlier sections, work that runs on a zIIP instead of a general-purpose CP is not included in the MSU metrics used to calculate your monthly MLC software charges. These metrics are calculated as a rolling four-hour average (R4HA) of LPAR MSU usage. The monthly LPAR peak of the R4HA, by product, determines your software bill. That means you are paying for capacity on a rolling four-hour average, not on the maximum capacity of your system or the maximum capacity utilized at any given time.

So easy enough, if you are adding Java workload (or converting existing workload to run on a JVM) and that work is redirected to and runs on a zIIP, it is not contributing to your R4HA or your monthly software bill. And the cost of the zIIP hardware itself is much less than the CPU the workload would run on if it were not zIIP eligible, meaning the cost savings are accumulated for multiple reasons.

Therefore, running Java instead of other types of work can significantly contribute to cost savings.

2 – Easier to Support

The next benefit of Java is that it can be easier to support than COBOL for various reasons. As we alluded to earlier, COBOL is aging, as are the programmers capable of coding and maintaining it properly. Of course, COBOL's age is not the problem; plenty of older things remain viable and thrive. And COBOL has not stayed static, stuck in the 1950s when it was developed. Nevertheless, skilled COBOL developers are not easy to find.

On the other hand, Java is a newer, thriving language. First released in 1995, Java can't be called shiny and new. Still, it is modern because it is object-oriented, taught in most college computer science programs, and is one of the world's most popular current programming languages. Java regularly ranks in the top three languages of the Tiobe Index, which tracks the popularity of computer programming languages.

3 – Code Maintenance

An easy-to-understand and -maintain code base is important to ensure effective application development and support. From the perspective of converting COBOL code to Java, though, this may be easier said than done. You agree that there is merit in converting some of your COBOL programs to Java, but how? Nobody has the time (or budget) to sit down a recode their applications line by line!

Converting from any programming language to another is a complex task that takes a long time and results in less-than-satisfying results. Without care and expertise, the converted code will not be efficient and is unlikely to take advantage of all the features of the target programming language. Even a derisive term, JOBOL, has been created to describe COBOL code that did not effectively convert to Java. In other words, it may be Java, but it still looks and feels like COBOL.

The key is to use conversion services built to understand how to convert from a procedural language like COBOL to an object-oriented language like Java. This is where an automated tool comes in handy. The CloudFrame Renovate and Relocate products provide code conversion tools, automation, and DevOps integration to deliver very maintainable, object-oriented Java that can integrate with modern technology available within your open architecture. It can be used to refactor COBOL source code to Java without changing data, schedulers, and other infrastructure components. It is fully automated and seamlessly integrates with the change management systems you use on the mainframe.

CloudFrame improves the business value of modernization by driving down the risk and effort required. Using CloudFrame services, you can convert COBOL to refactored Java that a Java programmer can work with effectively. In other words, it's not JOBOL. The Java code generated by CloudFrame will operate the same as your COBOL and produce the same output.

You can even use CloudFrame to refactor your COBOL to Java but keep maintaining the code in COBOL. Such an approach can allow you to keep using your COBOL programmers for maintenance and gain the zIIP eligibility of Java when you run the code.

4 – Speed?

Everybody knows that Java is slow, right? That is one of those common knowledge items passed around from mainframer to mainframer for so long that few question it. This may have been true a decade ago, but today, sometimes Java can run as fast as or even faster than COBOL.

Java in the zIIP may run faster that the equivalent COBOL in GPP

Some CloudFrame customers have found that the refactored Java ran faster than the COBOL from which it was converted. Of course, this is not to say that Java will consistently outperform COBOL, just like it is not fair to say that COBOL will consistently outperform Java.

Another consideration: if the Java code runs on a zIIP, and your main CPU is a kneecapped model, then the higher speed of the zIIP, which is never kneecapped, may cause your Java code to run faster than the equivalent COBOL.

5 – Portability

The final benefit of Java is its portability. A Java program can be easily transported from one environment to another because the Java source is compiled into bytecodes. The Bytecodes generated can be run on any machine with a JVM.

Summary

There are numerous benefits to modernizing your mainframe workload to run on Java. Among these benefits are reduced cost, easier support and maintenance, similar or better performance, and expanded portability. Java can open up your strategic options for mainframe usage and budget management. The next question is - what are your options for making better use of Java within your ecosystem?

Options for Converting from COBOL to Java

The combination of Java and zIIPs offer significant benefits, including cost-reducing, simplifying support and maintenance, delivering the performance you need, and improving portability. But most enterprise mainframe applications are written in COBOL.

Now, most organizations are not likely to embark on a full-fledged campaign to convert all of their COBOL to Java. But converting some programs to Java can make sense if you use the proper approach.

COBOL is Still Prevalent in Large Enterprises

Unless you work in a mainframe environment, it might surprise you that COBOL is still being used. But it is. And its usage is significant!

COBOL was designed for business data processing, and it is exceptionally well-suited for that purpose. It provides features for manipulating data and printing reports that are standard requirements for business. COBOL was purposely designed for applications that perform transaction processing like payroll, banking, airline booking, etc. These are programs where you put data in, process that data, and send a result.

COBOL was invented in 1959, so its history stretches back over 60 years; that's a lot of time for organizations to build complex applications to support their business. Over the years, IBM has delivered new capabilities and features that enable organizations to stay updated while maintaining their application portfolio. The current situation is that COBOL is in wide use across many industries.

Most global financial transactions are processed using COBOL, including 85 percent of the world's ATM swipes. According to Reuters, almost 3 trillion dollars in DAILY commerce flows through COBOL systems! The reality is that more than 30 billion COBOL transactions run every day. And there are more than 220 billion lines of COBOL in use today.

But COBOL applications face many challenges as experts in the field retire, and new developers are not trained in procedural languages like COBOL. Instead, colleges teach object-oriented languages, like Java. So new applications are commonly written in Java, even as legions of older applications remain in COBOL.

30 billion COBOL transactions run every day.

Even if converting everything at once from COBOL to Java would be too monumental of a task for most organizations, converting some COBOL to Java can make a lot of sense. It all boils down to your mindset and needs. The question to ask yourself is, "What type of fool are you?"

An old adage states: There are two different types of fools. One naively embraces and extolls everything old; the other credulously praises everything new. Do you embrace COBOL or praise Java? You can and probably should avoid being either type of fool by doing both! Yes, COBOL and Java can co-exist, perhaps for a long time, as you migrate to Java.

The Challenges of Maintaining COBOL

As you put your plan together, you might consider converting some of your COBOL applications to Java. An upcoming event (such as the end of support for a COBOL compiler or a wave of retirees hitting your development staff) may offer a ripe opportunity for converting.

co-exist

COBOL & Java Can...

Or you may want to take advantage of the benefits discussed in the last section, such as lower cost of running your applications, better portability, and improved application support. Nevertheless, converting to Java is a viable option, and many organizations are considering doing so, at least for some of their programs.

Keeping in mind the concerns about "all-or-nothing" conversions, most organizations will be working toward a mix of COBOL migrations and Java conversions, resulting in a mixture of COBOL and Java for their application portfolio.

As you plan for this, analyze and select appropriate candidate programs and applications for conversion to Java. Some tools can analyze program functionality to assist you in choosing the best candidates. For example, you will probably want to avoid converting programs that call other COBOL programs and programs that use pre-relational DBMS technologies, at least initially.

Converting COBOL to Java: Transform and Cross-Compile

At this point, you may be thinking, "Sure, I can see the merit in converting some of my programs to Java, but how can I do that? I don't have the time for my developers to re-create COBOL programs in Java going line-by-line!" But manual conversion is only one option, and it is by far the least desirable.

Using CloudFrame technology, you can perform two types of automated coded conversion: transform and cross-compile.

Using an expert toolset to automate code conversion makes a lot more sense. Using CloudFrame technology, you can perform two types of automated coded conversion: transform and cross-compile.

With transform, COBOL source code is automatically refactored into Java without changing data, schedulers, or other infrastructure components. It is fully automated and seamlessly integrates with the change management systems you use on the mainframe.

The Java code generated by CloudFrame will operate the same as your COBOL and produce the same output. You can even use options to maintain the COBOL 4.2 treatment of data, thereby avoiding the invalid data issues that can occur when you migrate to COBOL 6. This can help to reduce project testing and remediation time.

Perhaps even more importantly, the Java source code generated is Java, not a Frankenstein monster Java/COBOL combination that some folks refer to as JOBOL. The goal is to create Java code that Java developers will recognize as Java and be able to support without knowing any other legacy programming languages. Java code generated by CloudFrame regularly earns an "A" rating when processed by SonarCube code quality scoring (such as reported in this customer case study).

Using CloudFrame to transform and refactor your COBOL code to Java is a viable automated route for migrating to code that can be supported and maintained by Java programmers. Another option that may be more palatable to long-time COBOL shops is the CloudFrame cross-compile option.

You can use CloudFrame to refactor the COBOL to Java but keep maintaining the code in COBOL.

With cross-compile, you can use CloudFrame to refactor the COBOL to Java but keep maintaining the code in COBOL. The source code is COBOL, but it is cross-compiled to run in a JVM, making the workload zIIP-eligible. This approach is more fully described in this blog post (Consider Cross-Compiling COBOL to Java to Reduce Costs). It is also an excellent capability for shops with a lot of COBOL who are not comfortable refactoring everything to Java. You keep your COBOL until you are ready to shift to Java. You can quickly fall back to your COBOL load module with no data changes. The Java data is identical to the COBOL data except for date and timestamp.

Simply stated, CloudFrame offers automated software for refactoring COBOL code to Java and running using only Java. Or, if you are comfortable with your ability to support and code your applications in COBOL but are looking for the cost-savings that zIIPs can provide, then CloudFrame's cross-compile capabilities may be just what you are looking for.

Still, planning to move code to the zIIP doesn't remove all of the possible mistakes that can be made, especially when it comes to assumptions about what will work on the zIIP, planning for shifting workloads, and understanding exactly where your cost savings will come from.

Common zIIP Usage Mistakes and How to Identify Them

Many organizations using the IBM System z have begun to use zIIP processors to help reduce the overall cost of their mainframe environment. But there are some pitfalls that should be avoided.

Assuming Everything Will Run on the zIIP

One of the biggest mistakes you can make is assuming that everything that is eligible to run on the zIIP will actually run on the zIIP. Although this may seem like a reasonable assumption, it neglects to take into account what is sometimes referred to as the Generosity Factor.

When you activate your zIIP processor, some percentage of the relevant workload will be redirected off the main CP onto the zIIP – but not 100% of the workload. When an enclave is created by a product you are using, a parameter can be set to impact the CPU percentage that z/OS can make eligible to run on the zIIP. Think of this percentage as the Generosity Factor because it tells the system how generous to enable the workload for the enclave to be eligible for zIIPs.

Mistakes

Lack of Planning

Lack of Understanding

For example, if you look at the IBM Db2 12 for z/OS documentation in the section titled "Authorized zIIP uses for Db2 processing," you will see the following:

"SQL request workloads that use DRDA to access Db2 for z/OS® over TCP/IP connections and native REST calls over HTTP. Up to 60% of the Db2 for z/OS instructions execute such SQL requests when running in Enclave SRB Mode and accessing Db2 for z/OS."

This means that the generosity factor for distributed SQL requests in Db2 is 60%. There is a bit of nuance to how this is implemented, but the net result is that you will only get up to 60% of this workload to run on zIIPs.

Digging deeper into the documentation, you will see that other types of processing will have different generosity factors. For example, up to 100% of parallel query child processing can be run on the zIIP (after reaching a CPU usage threshold).

So far, we have discussed the generosity factor for Db2, which is an IBM product.

But ISVs also can make workloads run by their products zIIP-eligible. ISVs that zIIP enable workload using enclave SRBs are not typically going to throttle the redirection of their workload like IBM does. Although the API provides an option to set a "generosity factor," it is uncommon to be anything other than 100 percent. That said, be sure to understand how each of your ISV products works regarding zIIPs, including if they set a generosity factor other than 100%.

Not everything zIIP eligible can or will run on the zIIP.

It is essential to understand that not everything zIIP eligible can or will run on the zIIP. Another common mistake is not understanding that it is **not** possible to specifically direct the workload to run a zIIP. The workload can be specified as zIIP-eligible, but the decision whether to run on the zIIP or not is made at execution time by the Workload Manager (WLM).

Why might zIIP-eligible work not be run on the zIIP? Perhaps there are no zIIPs installed and available to be used. Or maybe the zIIPs that are available are all busy. Or, as we just discussed, perhaps the generosity factor comes into play. Although the workload is eligible, it falls outside the permitted percentage for this specific type of work.

Lack of Planning

As with most things, proper planning and preparation will go a long way toward successfully implementing zIIPs in your organization. Perhaps the most significant thing to consider is how much workload you will have that is zIIP-eligible and how many zIIPs you will need to deploy to support that workload.

Of course, Db2 is most likely to be the primary consumer of zIIP capacity, but don't forget other workloads such as Java, zCX container extensions, and XML processing. IBM publishes a nice list of zIIP-eligible software you should consult and compare to your environment's needs.

Once you know your zIIP potential, you must ensure sufficient zIIP capacity to process it. The more zIIP-eligible workload you have, the more zIIPs you will need to process it effectively.

Another consideration that is sometimes not handled appropriately is setting the IIHONORPRIORITY parameter correctly. IIHONORPRIORITY is a z/OS system setting that is maintained in the IEAOPTxx parmlib. There are two options: YES and NO.

Setting IIPHONORPRIORITY to YES indicates that standard CPs may execute zIIP-eligible and non-zIIP-eligible work in priority order - if zIIP processors are unable to execute all zIIP-eligible work. YES is the default.

On the other hand, if you specify IIPHONORPRIORITY=NO, work will not receive help from standard processors when there is insufficient zIIP capacity. This means that most work will wait for zIIP capacity to become available. There is a caveat: standard CPs can help when necessary to resolve contention for resources with non-zIIP processor eligible work.

You deploy zIIP processors to save money

The fundamental tradeoff is performance versus cost. You deploy zIIP processors to save money, so directing workload to the processors and keeping it there might seem to make sense. However, the potential performance implication is considerable, and therefore not only is YES the default, but it is the recommendation.

The best practice approach is to ensure sufficient capacity (for both zIIP and general-purpose CPs) to process your workload, set Honor Priority to YES, and monitor the situation, so there is minimal need to redirect workload from zIIPs back to your general-purpose CPs.

Lack of Understanding

The last common mistake we'll discuss here is a lack of understanding. To fully appreciate the cost-savings potential of zIIPs, you need to understand how IBM MLC software pricing and billing works. Of course, this is a complex topic that requires more that is outside of the scope of this eBook. So, instead, below, you'll find some advice.

The first thing you need to know is which IBM pricing metric is in place at your organization. If your shop uses a full-capacity metric or tailored-fit pricing, then every MSU you can redirect from the general-purpose engine to a zIIP can save you money. On the other hand, things become more difficult if you are using a sub-capacity metric.

There are numerous sub-capacity pricing metrics offered by IBM, including WLC, AWLC, EWLC, AEWLC, MWLC, and zNALC. At this point, don't worry too much about the acronyms and what they mean. The general idea for sub-capacity pricing is that you pay based on your MLC software's peak, monthly rolling-four average (R4HA) usage (by LPAR). IBM MLC software includes most system software, compilers, and selected system management tools.

The R4HA is calculated based on system utilization in MSUs. Each IBM machine has an MSU rating. This capacity is consumed by the LPARs and applications on the machine on an on-demand basis. As application consumption is based on, often unpredictable, demand, CPU/MSU may be very high or low at any moment in time. To allow for these brief spikes in consumption, IBM calculates the R4HA every five minutes. Each hour, the system takes the average of these 12 five-minute values. After four hours, the system will have an actual "Four-Hour Average". As the system runs, the calculation continues to "roll," so you have a continuously updated Rolling Four-Hour Average (R4HA). Each month, via the SCRT report, your organization reports the peak R4HA utilization to IBM, and a bill is generated based on that report.

So, with sub-capacity pricing, redirecting workload to zIIPs may or may not impact your monthly IBM software bill, depending on whether the workload is shifted during a peak. This means that sometimes using zIIPs will save you money, and sometimes they won't!

Earlier, we mentioned tailored-fit pricing with full-capacity pricing. Tailored-fit pricing is not exactly a complete capacity metric. It is not based on an R4HA peak. Therefore, any zIIP redirection from general-purpose CPs to zIIPs will save you money, eventually, when you use tailored-fit pricing.

The Bottom Line

To benefit from zIIPs, you need to understand what they offer and how they work and plan appropriately for your organization's workload and pricing agreement with IBM. In our penultimate section, we'll examine some best practices and practical tips for utilizing your zIIP processor to realize the greatest benefit from it.

Best Practices for zIIP Usage for COBOL and Db2

Having looked at what the zIIP is, how it can be used, how a modern language like Java can be leveraged on it, and what the most common mistakes when utilizing the zIIP are, it's time to consider best practices to make your zIIP usage a reality.

Why do this?

As we have discussed in prior sections, the primary reason to work on redirecting workload to zIIPs is to reduce cost. Workloads on the zIIP are not chargeable against your monthly IBM software bill. So, exploiting your installed zIIPs can result in significant cost savings.

How to do this?

Knowing how to improve your zIIP usage is actually pretty simple. It involves understanding what is zIIP-eligible and coding your applications to promote that type of usage. Your actual COBOL processing is not zIIP-eligible, but many types of activities enacted by your COBOL program can be zIIP-eligible.

Let's dig into the opportunities for zIIP usage in Db2 (with a COBOL mindset).

The first important thing is to review and stay up to date on the types of Db2 workloads that are zIIP-eligible. IBM maintains a list of what is zIIP-eligible. As of Db2 12 for z/OS, the list of authorized Db2 zIIP usage can be found here.

Workloads on the zIIP are not chargeable against your monthly IBM software bill.

Design your new COBOL programs and applications for **distributed processing**. SQL requests that use DRDA to access Db2 for z/OS over TCP/IP connections are zIIP eligible, with up to 60% of their instructions running on the zIIP. Similarly, programs issuing native REST calls over HTTP fall into the same category, with up to 60% being zIIP-eligible. Although it is not extremely common, it is possible to call a REST API from COBOL. Or you might choose to forgo COBOL for REST calls instead of relying on z/OS Connect or another approach.

So, the first thing to consider is how the application is being designed and how it will run. Using distributed SQL calls is probably the first, best thing that you can do to get COBOL programs to take advantage of your installed zIIP capacity.

But not everything can, or indeed should, be run as a distributed SQL request (or REST API call). Fortunately, other types of processing can be run on the zIIP. Db2 **parallel query child processes** for long-running parallel queries can also be run on zIIPs. There is a threshold after which up to 100% of these processes can be run on the zIIP.

So how can you encourage parallelism? There is no way to explicitly say, “This query must use parallelism.” Instead, when the query is bound (either statically or dynamically), the Db2 Optimizer decides if parallelism will be beneficial and, therefore, potentially used to satisfy the query request. At bind time, you can tell Db2 to consider using parallelism. For static SQL, you must code the DEGREE(ANY) parameter when you BIND or REBIND. You must set the CURRENT DEGREE special register for dynamic SQL to ‘ANY’. It is also possible to change the CURRENT DEGREE special register default from 1 to ANY for an entire Db2 subsystem by setting the value of the CDSSRDEF DSNZPARM parameter. This DSNZPARM sets the default for dynamic SQL only.

Keep in mind that parallelism is used for read-only queries only. Therefore, it makes sense to identify cursors that are read-only. If you bind your program using CURRENTDATA(YES) and Db2 cannot tell if the cursor is read-only, Db2 will not consider parallelism. Therefore, to optimize parallelism, it is a good practice to specify FOR READ ONLY or FOR FETCH ONLY for every CURSOR that will be used for reading data only.

Furthermore, queries run against partitioned table spaces will cause the Db2 Optimizer to more strongly consider parallelism. However, partitioning is not explicitly required for Db2 to invoke parallelism.

Many **IBM Db2 utility processes** are also zIIP-eligible. For example, up to 100% of the index maintenance tasks for LOAD, REORG, and REBUILD INDEX are zIIP-eligible. Portions of the RUNSTATS utility is also zIIP-eligible, but this typically has little to do with COBOL. So how can COBOL developers take advantage of the zIIP eligibility of IBM Db2 utilities?

The LOAD utility is very efficient, and the index maintenance needed is all zIIP-eligible

One tactic to consider is to avoid creating COBOL programs that perform many Db2 SQL INSERTs and instead use the LOAD utility. Of course, if the program only inserts a few rows, then the LOAD utility is probably not the best solution. Additionally, if a lot of pre-processing or other activity is required before the data can be inserted, then the LOAD utility may not be suitable.

Using the LOAD utility is worth considering if you have a large data set of records that need to be read and inserted into a table. The LOAD utility is very efficient, and the index maintenance needed is all zIIP-eligible. This is not the case for COBOL programs performing INSERTs.

Note that other ISVs offer Db2 utilities with varying degrees of zIIP eligibility. So, if you have Db2 utilities from BMC, Broadcom, or InfoTel be sure to consult their documentation for details on their zIIP exploitation.

If your program is processing XML data, this too can be zIIP-eligible. Up to 100% of XML schema validation and non-validation parsing and up to 100% of the deletion of unneeded versions of XML documents can be run on the zIIP.

Another consideration to keep in mind is that as your COBOL programs run and access Db2 data, many other incidental Db2 system tasks and work may be zIIP-eligible.

Up to 100% of the work done by Db2 system agents processing under enclave SRBs that execute in the Db2 MSTR, DBM1, and DDF address spaces is zIIP-eligible with a few exceptions. So, things that Db2 does during normal operations, like buffer pool processing and log operations, can be run on the zIIP as your COBOL programs run.

It pays to know what software you have and whether it is zIIP-enabled!

When it comes to sorting, IBM DFSORT workload is not zIIP-eligible. IBM does market a product called Db2SORT that works with Db2 utilities to zIIP enable some of the sort processing used by utilities. But none of that helps sorting for use by COBOL programs. However, if you have a lot of data to sort before processing it by your COBOL program, you might consider Syncsort (from Precisely) and the add-on ZPSaver component of Syncsort. It can be used to enable sort, copy, and SMS compression to be zIIP-eligible. It pays to know what software you have and whether it is zIIP-enabled!

As a final consideration, if you are not getting sufficient zIIP utilization for your COBOL programs, you can consider converting to Java, all of which is zIIP-eligible. Of course, this requires significant planning and investment but can be made easier by using tools like those marketed by CloudFrame to do the conversion for you.

Predictions for the Future of zIIP and Specialty Processors

Throughout this eBook, you've read about the current state of specialty processors for IBM Z mainframes with a particular concentration on the zIIP. We've looked at what they are, how they work, and why you might want to consider exploiting them. But in this final section, let's ruminate on the potential of specialty processors in the future.

It's impossible to know specifically what the future holds. But it is possible to make some observations based on existing practices and usage of specialty processors. So, of course, the "future" discussed here will be a guess... but it will be an educated guess!

A Level Set

The first thing to do is to confirm that the need for specialty processors is as strong or stronger than it has ever been. According to a recent BMC Mainframe Survey, 86% of the largest mainframe shops expect MIPS to grow in the coming year. So, with the potential of specialty processors to mitigate cost growth as capacity increases, it stands to reason that organizations should continue to utilize them.

Of course, you could make the case that reducing prices could be a more efficient way to mitigate the cost of mainframe software. When you think about it, the zIIP is really nothing more than a re-purposed general-purpose CP, with controls that enable only certain types of workload to run on it. The only real difference is that a zIIP always runs at full capacity, even if the CPC is kneecapped (meaning that it is a sub-capacity model designed to run at a lower capacity).

A zIIP always runs at full capacity

But this is a simplistic way to look at the situation. Specialty processors are designed to encourage specific workloads – typically newer ones – to run cheaply on the mainframe. This enables IBM to reduce the cost of mainframe computing and encourages growing the footprint of what runs on the platform. Simply lowering prices would not necessarily accomplish the same thing.

The impression given of IBM's perspective, and this is simply an opinion, is that decreasing prices overall does nothing to protect the large revenue stream IBM earns from mainframe hardware and software. By redirecting modern workloads (where there is much competition) to specialty processors, IBM can decrease the price of "modern" workloads while protecting the revenue it garners from "legacy" workloads, such as CICS and IMS transactions and batch programs (for which there is little to no competition).

An additional consideration is the cost of the specialty processors, which are much lower than the cost of a standard CP. And that means that specialty processors can reduce both software and hardware costs!

Types of Specialty Processors

Keep in mind that there are three different types of mainframe specialty processors: ICF (Internal Coupling Facility), IFL (Integrated Facility for Linux), and the one we've talked about the most, zIIP (Integrated Information Processor). We chatted about the purpose of each of these in the first section.

For the purposes of reflecting on their future, those are the specialty processors we will consider. Nevertheless, on IBM Z servers, there are some processor units (PUs) that are part of the system base but are designed for specific purposes. They include the System Assist Processor (SAP) used by the channel subsystem, the Integrated Firmware Processor (IFP) used in support of select features, and two spare PUs that can transparently assume any characterization during a permanent failure of another processor unit.

The cost of the specialty processors is much lower than the cost of a standard CP

Then there is the IBM Integrated Accelerator for Z Sort. Not technically a specialty processor, this feature was introduced with the IBM z15 and helps reduce CPU costs and improve the elapsed time for sorting. This is accomplished using a new instruction and optimally using virtual, real, and auxiliary storage.

Although you might want to think of these PUs and features as specialty processors, they are not exactly like the IFL, ICF, and zIIP, which are separately priced options. Nevertheless, if you extend the notion of what a specialty processor is, then it becomes obvious that we will continue to see more specialty hardware like this being introduced into future IBM Z hardware configurations.

But what about the zIIP and its cohorts?

Looking Into the Future

As we peer into the future to see if, how, and why specialty processors will be used, we first recognize that many organizations have implemented them and rely on them. This helps to secure their future, at least somewhat. If IBM were to consider eliminating specialty processors, it would negate the purpose for which they were developed, reducing the cost of newer workloads and extending the viability of the mainframe platform.

So, taking specialty processors away would be difficult unless IBM also created another way to achieve the same purpose. As long as specialty processors continue to work and are being used by customers, it is safe to assume that they will be viable for the long term.

On the other hand, introducing additional specialty processors might be more practical. For example, as use cases and computing techniques expand for Artificial Intelligence and Machine Learning, it would not be surprising if an AI specialty processor is introduced. Security and cryptography are other areas that might benefit from specialty processors. But, of course, this is pure speculation on my part.

You should also keep your eyes on IBM's pricing policies and announcements. As pricing metrics change, the cost patterns and expectations of users change. Therefore, a future pricing option might obviate the need for specialty processors or, indeed, change the use cases for which they are designed.

For example, you might consider the impact if your organization changes from a sub-capacity pricing metric like AWLC to the newer tailored-fit pricing metric. Although tailored-fit pricing is not exactly a full capacity metric, it is not based on an R4HA peak. Unlike sub-capacity pricing metrics, where only reduced workload during peak periods saves money, any zIIP redirection from general-purpose CPs to zIIPs will likely save you money, eventually, when you use tailored-fit pricing.

The Bottom Line

It appears that the future is bright for specialty processors. There should be no worry about implementing them today to reduce costs. Of course, it makes sense to keep an eye on IBM's announcements and pricing options to ensure that you are using specialty processors optimally at your site.

Conclusion

Mainframes remain a crucial piece of the ecosystem for many enterprises, government agencies, and other organizations. The expense of licensing and utilizing your mainframe is only a portion of the value that it brings to the business. At the same time, that doesn't mean there aren't ways to create savings opportunities - without the need to embark on a full modernization initiative.

The zIIP specialty processor is one of those opportunities. By running eligible "new" code on the zIIP and refactoring some of your legacy code to Java, it's possible to experience thousands of dollars in cost savings.

Refactoring COBOL code doesn't need to be intimidating. It is possible to take advantage of refactoring and cross-compiling COBOL to create maintainable Java applications that are data equivalent to their legacy code counterparts - Java code that can be shifted off the general processor and onto the zIIP.

How? By using CloudFrame Renovate and Relocate products. These tools offer clean code conversion, automation, and even DevOps integration, leaving you with object-oriented Java applications that can integrate with your modern applications and run on the zIIP. The savings realized could be used for headcount, modernization initiatives, and more. The ROI for CloudFrame is both fast and substantial.

Want more information about Renovate and Relocate? Download our product sheets and read case studies from our satisfied customers. Want to learn more about how CloudFrame can add to your budget and offer a seamless path to greater specialty processor usage? Schedule a call with us.

About the Author

Craig Mullins

Craig Mullins is a mainframe authority specializing in database administration (DBA), database management, and analytics. His career spans multiple decades, and he has worked both as a hands-on mainframe developer and database administrator, and as a trusted advisor for organizations optimizing their mainframe infrastructure and developing strategies for their mainframe systems.

Craig is the author of multiple Db2 books and is a sought-after presenter and speaker at mainframe-oriented events. He is recognized as an IBM Gold Consultant, IBM Champion for Data and AI, and listed by Analytics Week as one of the Top 200 Thought Leaders in Big Data & Analytics.

He is the President & Principal Consultant of Mullins Consulting, INC., where he leverages his years of experience and expertise in database systems development to create and conduct database classes and deliver systems analysis and design, data analysis, database administration, performance management, and data modeling projects.

About CloudFrame

CloudFrame frees mainframe applications from COBOL, giving customers automated, incremental, low-risk, low-cost ways to transform those applications into fully supportable cloud-native Java. Our products give customers control and the ability to transform mainframe applications fast, freeing them from COBOL and vendor dependency completely and fully enabling digital transformation.

CloudFrame, Inc.
100 Overlook Center, 2nd Floor
Princeton, NJ 08540

cloudframe

cloudframe.com