IBM Enterprise COBOL Performance with IMS

Performance Results and Best Practices

Jasdeep Singh

IMS On Demand Performance

IBM Systems, z Systems

Jasdeep K Singh/Silicon Valley/IBM

Abstract

This technical report provides an overview of performance results of IBM Enterprise COBOL for z/OS, V4.2, V5.1, V5.2 and V6.1, with the IBM Information Management System (IMS).

The technical report includes best practices for both tuning options when using COBOL V5 and optimizing the performance with IMS. It also explains the IMS tuning features and the z/OS APARs needed to get the best performance with COBOL V5 with Library Lookaside (LLA).

The technical report also shows the performance comparison of COBOL V4.2, V5, and the latest, V6.1, which enables the 64-bit support.

Software Group
San Jose

Note: Performance is based on measurements and projections using IMS benchmarks in a controlled environment. The results that any user will experience will vary depending upon many factors, including considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, the amount of zIIP capacity available during processing, and the workload processed. Therefore, results may vary significantly and no assurance can be given that an individual user will achieve results similar to those stated here. Results should be used for reference purposes only.

The test scenarios (hardware configuration and workloads) used in this document to generate performance data are not considered 'best performance case' scenarios. Performance may be better or worse depending on the hardware configuration, data set types and sizes, and the overall workload on the system.

The information contained in this document has not been submitted to any formal IBM test and is distributed on an "AS IS" basis without any warranty either expressed or implied. The use of this information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into their operational environment. While each item may have been reviewed by IBM for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The information provided in this paper was obtained at the IBM Silicon Valley Laboratory and is intended for migration and capacity planning purposes.

Introduction

IBM Enterprise COBOL for z/OS is a leading-edge, z/OS-based compiler that helps you create and maintain mission-critical, line-of-business COBOL applications to execute on your z/OS systems. It continues to give access to middleware software like CICS, DB2, IMS, and other transactional and data systems.

IBM Enterprise COBOL for z/OS, V5.1 and V5.2 allows you to generate your applications for higher levels of the z/Architecture and the performance optimization. Two new significant compiler options were introduced: ARCH and enhanced OPTIMIZE compiler options allow you to maximize the delivery of z/Architecture exploitation and performance optimization within your applications.

The new compiler option ARCH gives you the flexibility to have the Enterprise COBOL compiler generate code for higher levels of the z/Architecture. Higher ARCH levels instruct the compiler to exploit newer processor instructions to optimize and tune your application code to the latest levels of z/Architecture. The enhanced OPTIMIZE compiler option provides you with the flexibility to obtain multiple levels of increasing optimization that run from comprehensive low-level optimizations to more extensive optimizations that can improve the performance of your COBOL applications.

IBM Enterprise COBOL for z/OS, V6.1 is the first release of COBOL that enables 64 bit support in optimizer code and converts it into a 64 bit executable. This feature will allow customers to compile large application code.

Performance Testing with IMS

An IMS Full Function workload with HALDB driving a mix of transactions with an average of 25.5 DB calls and 6 DC/system service calls per transaction was used to test all COBOL versions. The new compiler options, ARCH(6), ARCH(7) and OPTIMIZE(2) were used to compile the IMS application with the link options as shown below:

Compiler Options:

```
//COMCOB5 JOB CLASS=A,MSGCLASS=H
//DSWCOB PROC
//COB    EXEC  PGM=IGYCRCTL,
//       PARM='NOMD,ARCH(6),OPTIMIZE(2),
```

Link Options:

```
//LKED   EXEC  PGM=IEWL,PARM='XCAL,LET,LIST,XREF,REUS,
//             FETCHOPT=(PACK,PRIME)',REGION=1M
```

In Enterprise COBOL V5.1, ARCH(6) is the default. Support for ARCH(6) option was officially removed in COBOL V5.2 and default of ARCH(7) is in effect for COBOL V5.2. Leaving the ARCH(6) option in COBOL V5.2 JCL will not affect the compilation, this option will be just ignored.

Lookaside Library (LLA) was used to manage caching of COBOL load modules (PDS load libraries) and program objects (PDSE load libraries). It was noticed that COBOL V5 modules were not being cached by LLA in the base versions of z/OS because of the structure of their resulting program objects. IBM® has corrected this in the service stream via LLA/VLF APAR OA45127 to enable LLA management of program objects with deferred segments as long as the number of segments does not exceed two. Hence, if you use LLA to manage caching of COBOL V5 libraries, this fix should be on your z/OS system. This APAR (OA45127) enables these objects to be eligible for caching in LLA/VLF and reduces the high CPU consumption and long I/O times which may happen otherwise.

One approach for loading large program objects is called page-fault driven loading. For libraries that are managed by LLA, in some cases performance can be improved by avoiding page-fault driven loading. If you use the binder option FETCHOPT=(PACK,PRIME) for a program object, the system does not use page-fault driven loading. The default binder option of FETCHOPT=(NOPACK,NOPRIME) allows use of page-fault driven loading.

COBOL V4 testing was also run using PDSE's to have a fair comparison with COBOL V5 using other similar software and hardware environment.

COBOL V5 performs similar to COBOL V4 with the LLA/VLF APAR applied. All tests were done using the same applications on the same hardware with same number of schedules per transaction.

The two middle bars in Figure 1 shows the decreased throughput (increased CPU) when running without the above mentioned APAR. This same effect would be seen even with the APAR if the object library was not being managed by LLA/VLF. This series of tests comparing COBOL V4.2 and V5.1 with and without APAR were run using IMS 13. Note that PDSE hyperspace was not enabled for those measurements.

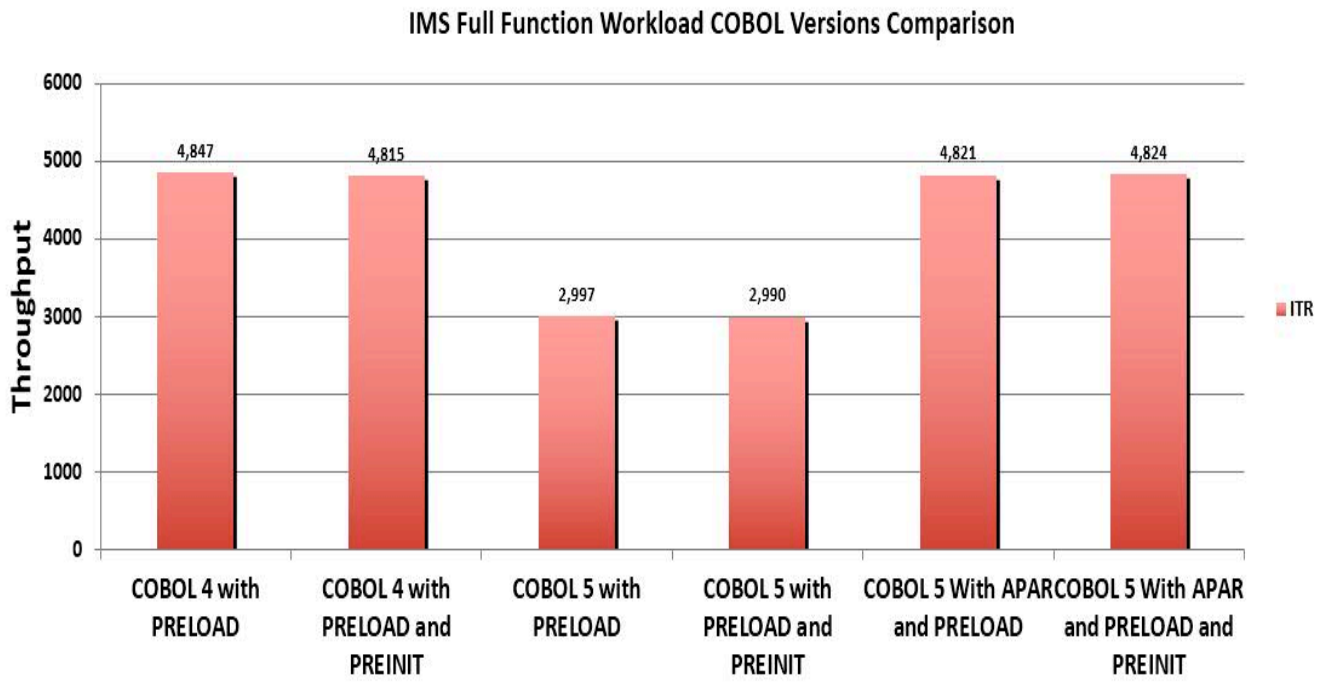

Figure 1: IMS Full Function Workload COBOL V4 and V5.1 Versions ITR Comparison with APAR

Using the same workload with LLA APAR enabled on z/OS, a series of comparison tests were run comparing COBOL V4.2, V5.1, V5.2 and V6.1. The results show similar performance throughout these versions with a variation of +/- 2%. Higher compile times were observed in COBOL V6.1 as compared to COBOL 5 due to 64 bit support, however the runtime performance was not affected as shown in Figures 2 and 3 below. This series of tests comparing COBOL V4.2, V5.1, V5.2 and V6.1 were run using IMS 14.

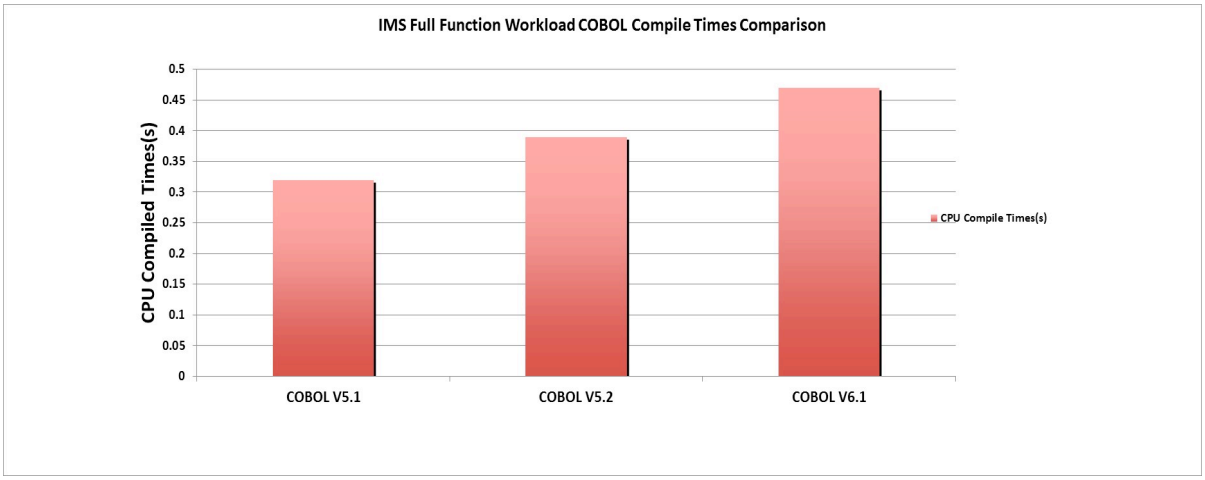

Figure 2: IMS Full Function Workload COBOL V5 and V6 CPU Time Comparisons

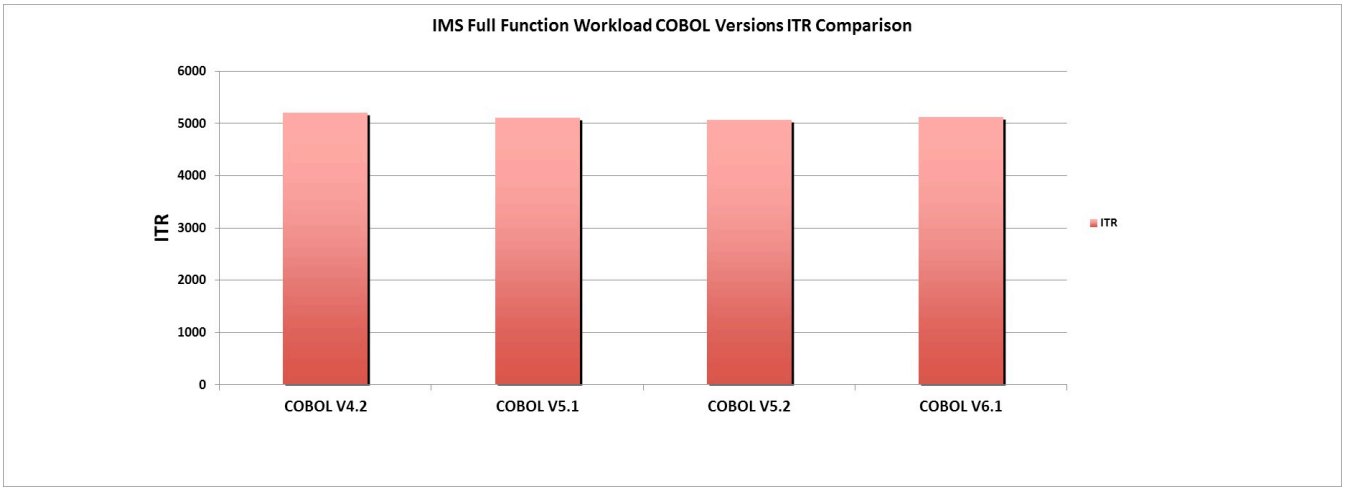

Figure 3: IMS Full Function Workload COBOL Versions ITR Comparison

Best Practices using COBOL

PRELOAD

Use the DFSMPLxx member of the IMS™ PROCLIB data set to make highly used z/OS®IMS, or user-written program modules permanently resident in the dependent region during region initialization. These modules will reside in what is known as the JPA or Job Pack Area which is the first area checked to find modules when they are linked, loaded, or fetched.

Making some modules resident can significantly improve throughput and response time for transactions which frequently refer to them. Of course you must have sufficient virtual and real storage available in these regions to obtain the most benefit.

COBOL and LE modules are recommended to be added to the PRELOAD list. It reduces response times, CPU service times and increases throughput significantly. Figure 4 shows the Transaction CPU time comparison for COBOL 5 with and without PRELOAD enabled using IMS 13.

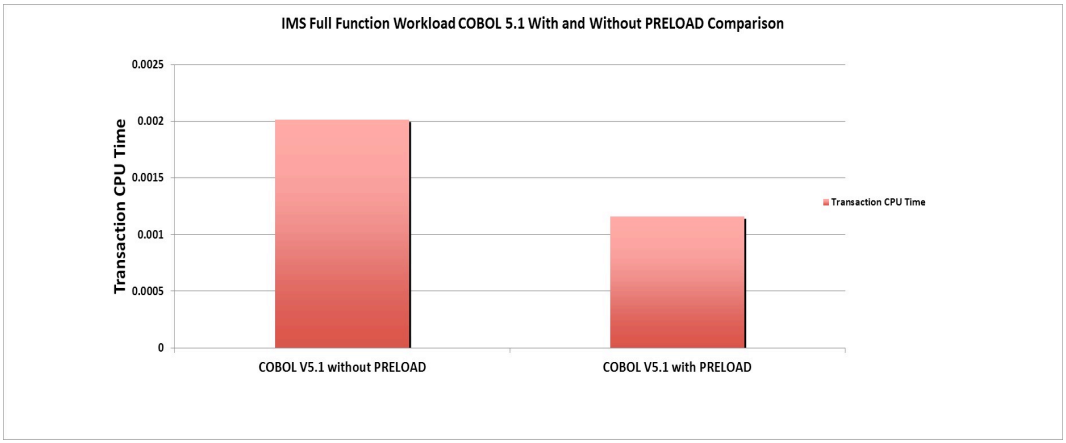

Figure 4: IMS Full Function Workload COBOL 5 Transaction CPU Time Comparison with PRELOAD

The following COBOL V5.1 members were preloaded in the DFSMPLxx member for the performance evaluations:

CEEBINIT,CEEPLPKA,CEEEV004,IGZXDMR,IGZXD24,IGZXLPKA,IGZCEV5,IGZXLPKD and IGZXLPKF

NOTE: The syntax of the PRELOAD member requires the comma as a delimiter. A space anywhere in the line will signal the end of the list and may not provide desired effect.

PREINIT

The PREINIT parameter is specified in the DFSMPR procedure, which is an online execution procedure that initiates an IMS™ message processing address space. Here is an example of the IMS Message Processing procedure with the PREINIT option specified. As you will notice the procedure specifies a suffix of M1 for the PREINIT parameter. This suffix is appended to DFSINT to determine the PREINIT member to process for this region. The module shown (CEELRRIN) is a sample which is supplied by IBM.

```
//PROC SOUT=A,RGN=56K,SYS2=,
//CL1=001,CL2=000,CL3=000,CL4=000,
//OPT=N,OVLA=0,SPIE=0,VALCK=0,TLIM=00,
//PCB=000,PRLD=,STIMER=,SOD=,DBLDL=,
//SSM=,PREINIT=M1,ALTID=,PWFI=N,
//APARM=,LOCKMAX=,APPLFE=,ENVIRON=,
//JVMOPMAS=,PARDLI=

  BROWSE    IMSPERF.IMS.PROCLIB(DFSINTM1) - 0
  Command ===>
 ******************************** Top of Data
 CEELRRIN
```

The reason to use PREINIT is to reduce the overhead once IMS scheduling is complete and before the application program actually receives control. This time can be viewed using the IMS monitor and looking at the Schedule to 1st DL/I call times. This time includes loading any necessary modules not in the preload list as well as initializing the

language environment. By using the PREINIT option LE keeps a subset of its resources in memory thus minimizing this time.

Keep in mind that this overhead occurs only when a program is scheduled into the region meaning that applications defined as WFI or using the PWFI option with good reuse of the same transaction in the same region would not benefit as much from this option. Of course the best performing option is minimize scheduling but when scheduling is required then the use of PREINIT can be of significant benefit.

Figure 5 shows the impact on Schedule to 1st DL/I time with the various options for our specific workload using IMS 13.

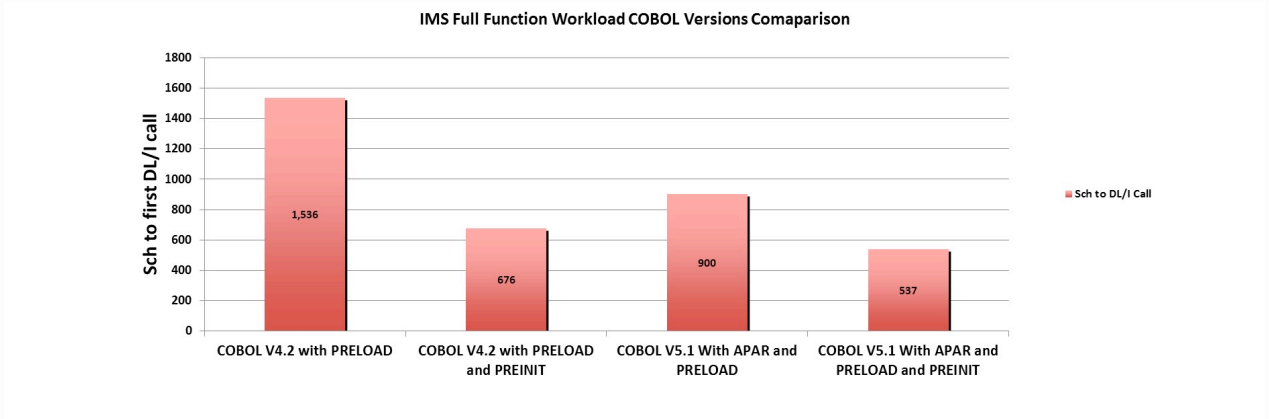

Figure 5: Schedule to first DL/I call Time for COBOLV5

PDSE Hiperspace Caching

PDSE Hiperspace Caching provides an opportunity for PDSEs to use central storage as a substitute for I/O operations at the expense of CPU utilization. It's an alternative to using LLA/VLF function for PDSE members. This section provides an overview of the setup needed for Hiperspace Caching and the performance efficiency as compared to LLA/VLF. COBOL 5 program objects without the loader APAR (OA45127) are not eligible for LLA caching, however they are eligible for Hiperspace caching to limit I/O operations without the APAR.

To enable Hiperspace caching for PDSE's, there are two possible scenarios:

1. For SMS managed PDSE to be eligible for Hiperspace caching, MSR (Milli Second Response) value associated with the storage class should be equal to 1.

2. For non SMS managed PDSE to be eligible for Hiperspace caching, they need to be included in the LNKLST for LLA/VLF. If LLA determines that a member cannot be cached in VLF, but otherwise would be eligible for caching in Hiperspace.

In our lab environment, LLA performed much better when compared to Hiperspace caching while using the same hardware and environment setup as shown in Figure 6 below. Significant performance degradation was noted while comparing PDSE Hiperspace Caching test with LLA/VLF using PRELOAD and PREINIT enabled in both scenarios using IMS 14. It is recommended to run with LLA/VLF as compared to PDSE Hiperspace caching due to more performance efficiency using LLA/VLF.

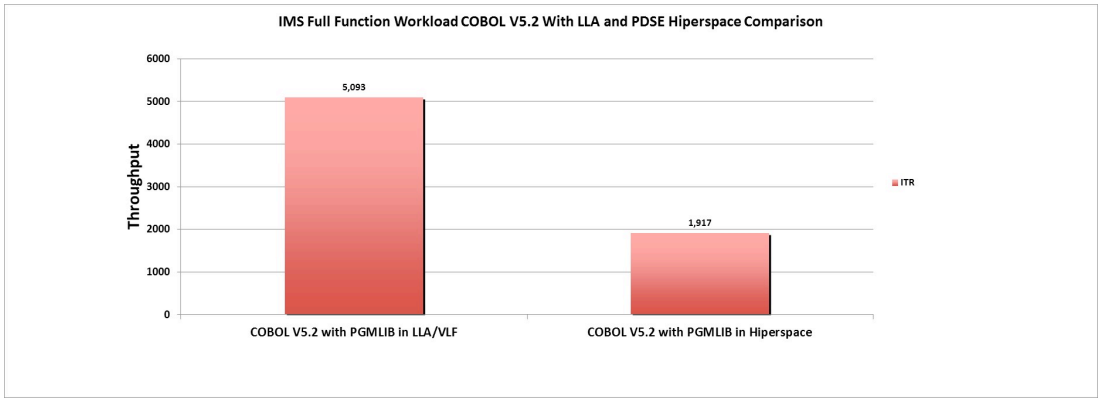

Figure 6: ITR comparison for LLA/VLF and Hiperspace Caching

Hiperspace caching parameters are specified in SYS.PARMLIB IDGSMSxx member.

First parameter is the PDSE_HSP_SIZE (PDSE1_HSP_SIZE).

PDSE_HSP_SIZE = x
PDSE1_HSP_SIZE = x

This parameter can be used to request up to 2047 MB for the PDSE Hiperspace. In order to activate Hiperspace Caching the PDSE(1)_HSP_SIZE must be set to a value greater than 0MB. The default HSP_SIZE value is 0MB making Hiperspace caching disabled by default. Recommended HSP_SIZE of 256MB was used for this performance test.

PDSE(1)_BUFFER_BEYOND_CLOSE was also added to retain the cached member pages of the dataset. This is useful for PDSEs which are frequently opened and closed.

Display SMS command (D SMS,OPTIONS) can be used to show the options set for Hiperspace caching.

The following is an example of the SMS options display. The display has been truncated to show only the options relevant to our discussion. As you can see the PDSE_HSP_SIZE is greater than zero to enable caching. For our experiments we set the BMFTIME to two minutes in order to get frequent statistics written to SMF but of course that is probably too frequent for most installations. Note that at the BMFTIME interval a 42.1 SMF record will be written for cache statistics. At this time a 42.6 record will also be written for those data sets being cached unless the data set has done no physical I/O in which case this record is not written.

```
D SMS,OPTIONS

IGD002I 09:25:55 DISPLAY SMS 270
ACDS     = SYS1SMS.IMSPER97.ACDS
COMMDS   = SYS1SMS.IMSPER97.COMMDS
ACDS LEVEL = z/OS V2.2
```

```
SMS PARMLIB MEMBER NAME = IGDSMS10
INTERVAL = 10            DINTERVAL = 150
SMF_TIME = YES           CACHETIME = 3600
CF_TIME = 3600           PDSE_RESTARTABLE_AS = YES
PDSE_BMFTIME = 120     PDSE1_BMFTIME = 120
PDSE_LRUTIME = 60        PDSE1_LRUTIME = 60
PDSE_LRUCYCLES = 15    PDSE1_LRUCYCLES = 15
LOCAL_DEADLOCK = 15      GLOBAL_DEADLOCK = 4
REVERIFY = NO            DSNTYPE = PDS
ACSDEFAULTS = NO         PDSESHARING = EXTENDED
OVRD_EXPDT = NO          SYSTEMS = 32
PDSE_HSP_SIZE = 256MB    PDSE1_HSP_SIZE   = 256MB
```

Verification of Hiperspace caching can be done by issuing D SMS,PDSE,HSPSTATS command and by looking in the DFSMS Record Type 42 Subtype 1 storage-class summary section that gives statistics on the BMF/Hiperspace Cache function. The following is an example of the above display command where you can see the IMSPERF library is set to cache in hyperspace.

```
D SMS,PDSE1,HSPSTATS
IGW048I PDSE HSPSTATS Start of Report(SMSPDSE1) 985
HiperSpace Size: 256 MB
LRUTime : 60 Seconds   LRUCycles: 15 Cycles
BMF Time interval 120 Seconds
---------data set name----------------------Cache--Always-DoNot
                                            Elig---Cache--Cache
J93SYS0.SIEALNKE                              Y      N      N
J93SYS0.SIEAMIGE                              Y      N      N
IMSPERF.JZOS.LOADLIB                          Y      N      N
CEE.SCEERUN2                                  Y      N      N
SYS1.SIEALNKE                                 Y      N      N
SYS1.SIEAMIGE                                 Y      N      N
SYS1.SHASLINK                                 Y      N      N
TCPIP.SEZALOAD                                Y      N      N
IMSPERF.COBOL.PGMLIB                          Y      Y      N
PDSE HSPSTATS  End of Report(SMSPDSE1)
```

More information on Hiperspace Caching can be found at:

http://www-01.ibm.com/support/docview.wss?uid=isg3T1022058

Considerations for Enterprise COBOL V5 and later programs

Programs in the list below contain CSECTs with the RMODE 24 attribute.
– Enterprise COBOL program that is compiled with the RMODE(24) or NORENT compiler options.
– VS COBOL II program that is compiled with the NORENT compiler option.
– Assembler program that contains CSECT with RMODE 24.

By default, the RMODE attribute of an Enterprise COBOL V5 program is RMODE ANY. When such program is linked with any of the above, the Binder will place RMODE 24 CSECTS in one segment, and the Enterprise COBOL V5 code in a second segment. There is also a third segment for the C-WSA class (new starting in COBOL V5). Program objects with more than 2 segments cannot be used with the Library Lookaside (LLA) facility. This issue can be avoided by specifying RMODE(24) compiler option explicitly for the Enterprise COBOL V5 program.

This would make the RMODE attributes consistent within the program object. Alternatively, you can also change the compilation of the COBOL programs in the above list by not using the RMODE(24) and NORENT options, and not having RMODE 24 CSECTs in assemble programs.