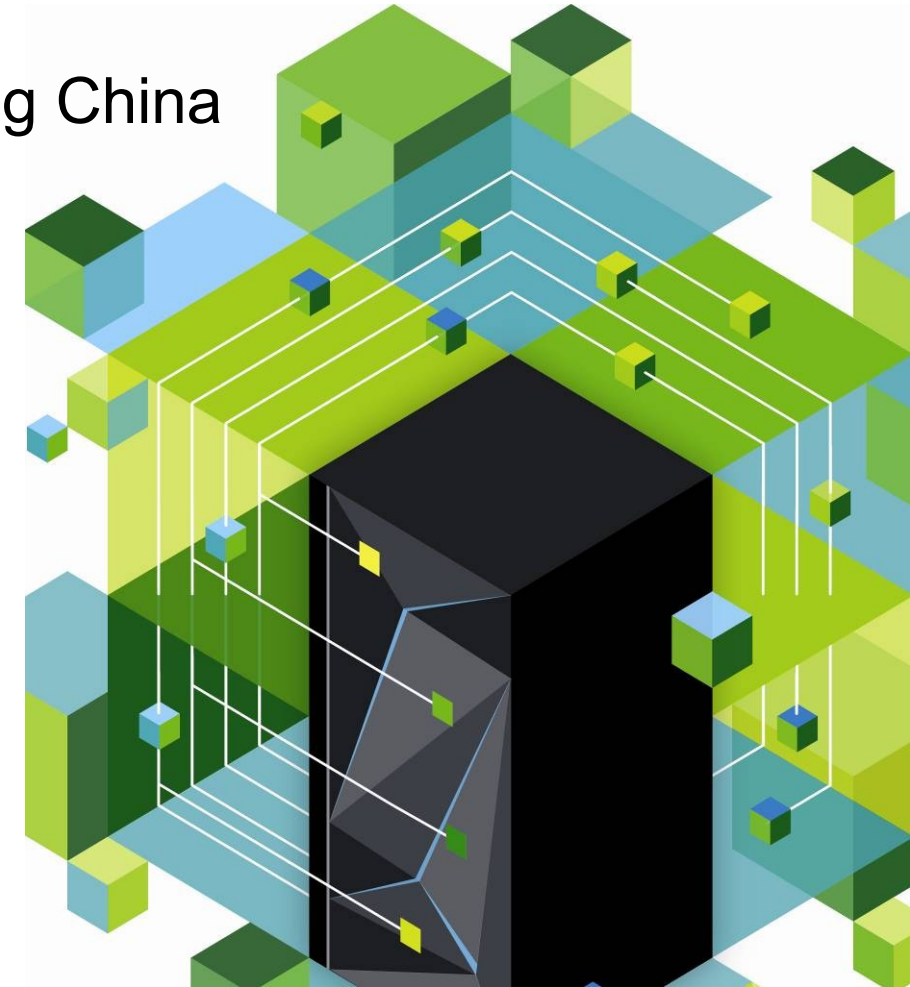
IBM i Application Program Optimization

IBM i Customer Study Tour, Beijing China

March 8th 2013



Gottfried Schimunek

Senior Architect
Application Design
IBM STG Software Development
Lab Services

IBM ISV Enablement

3605 Highway 52 North
Rochester, MN 55901

Tel 507-253-2367
Fax 845-491-2347

Gottfried@us.ibm.com

Agenda

- Introduction to IBM i Program Optimization techniques
- Program Optimization for CPU use
  - Blocked I/Os
  - Reducing/Avoiding Full Opens
- Program compiler background
  - TIMI – technology independent machine interface
  - Processor architecture adoption and hardware optimization through translator
- Program translator optimizations
  - Compiler options to optimize generated code
  - Program profiling to identify interdependent code optimization
  - Advanced Argument optimization
  - Adaptive code generation
  - Interprocedural Analysis
- Identifying Programs and Procedures
  - Hot spot analysis to identify programs, procedures or individual statements

Overview

- RPG and COBOL programs are optimized to run on IBM i with every release
  - will adopt the latest processor enhancements when
    - compiled or
    - when re-translated, e.g. when going from 5.4 to 6.1 or 7.1

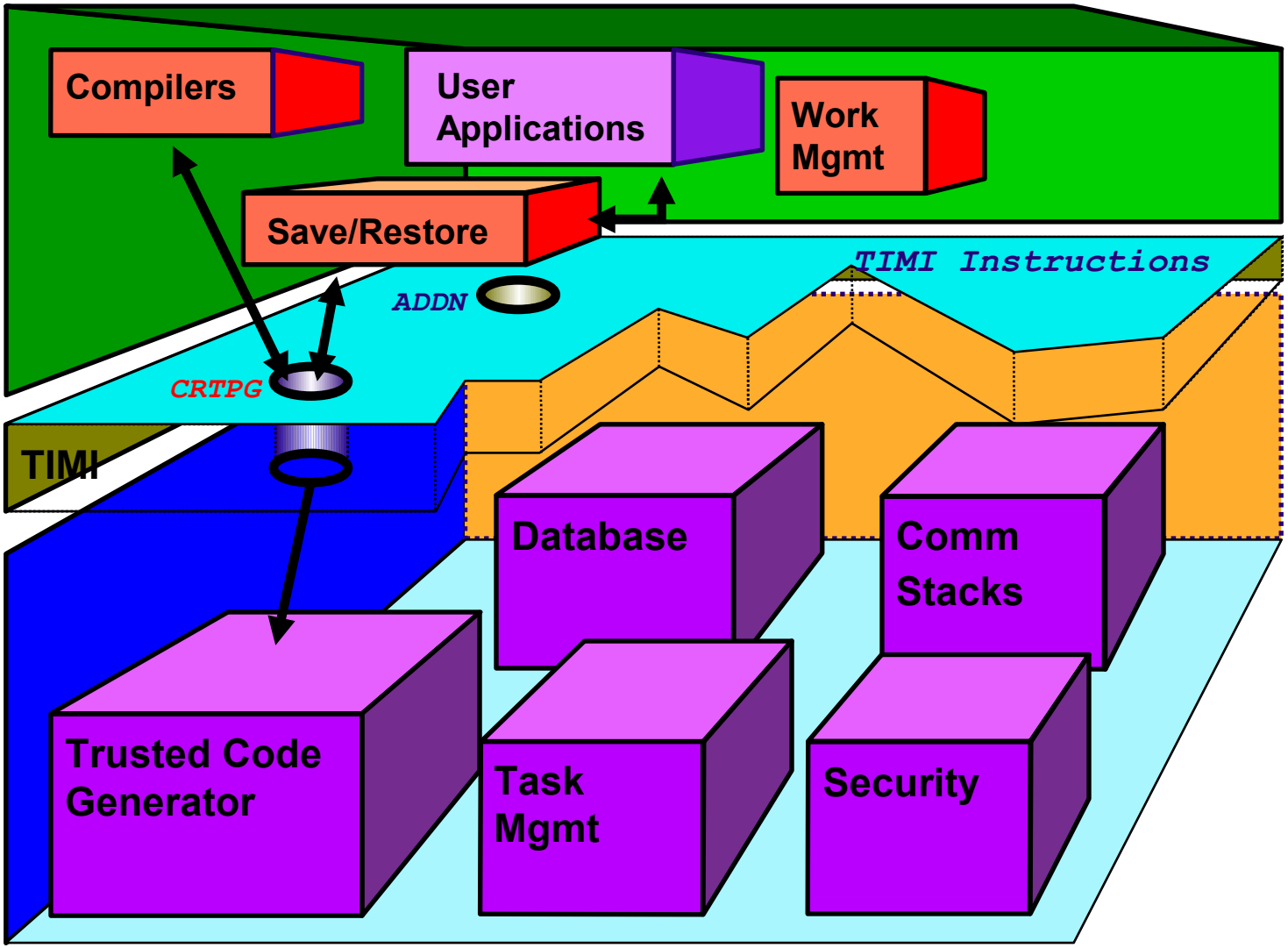
- Additional optimization can be done over and above what the compilers and the translator provide in order to improve performance in terms of runtime or CPU use and path length.

- Some of the following optimization techniques will trade off better optimization with the capability to debug the code, so be aware when deciding to use program optimization.

- It is generally good practice to select a few highly used programs with intense use of CPU cycles and perform optimization and when debugging is not required any longer.

- Some of these performance optimization techniques can be very significant and worth the effort for highly used programs in the interactive or batch environment

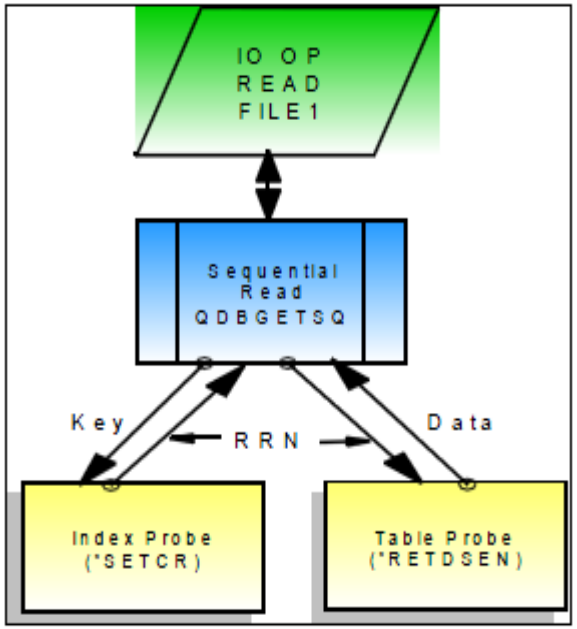
IBM i System Architecture



Blocked IO

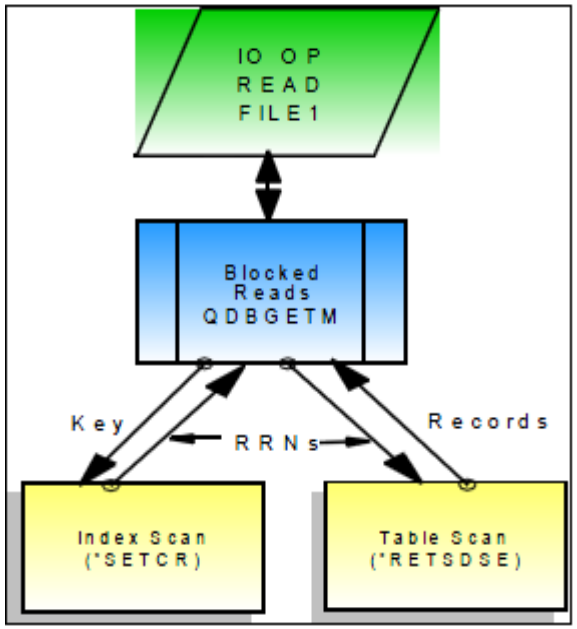
Using Blocked IO to Reduce CPU… and IO

SEQUENTIAL IO



1. Program issues a read operation
2. **QDBGETSQ** is called to
3. **SETCR** MI Instruction is called to return the RRN.
4. **RETDSEN** MI Instruction is called to do a table probe using the RRN
5. If successful **record data is returned to the program**.
6. This is done a record at a time.

BLOCKED IO



1. Program issues a read operation
2. **QDBGETM** is called to
3. **SETCR** MI Instruction is called to **return a set of RRNs based on block size specified.**
4. **Set of RRNs passed to RETSDSE** MI Instruction which returns a block of records
5. **Records stored in a file buffer** and **returned to HLL program a record at a time for each Read operation.**
6. Another call is made to QDBGETM when buffer is exhausted

Blocked IO –

- RPG compiler generates code to do blocking by default when one of the following is true:
  - File is program described, or if externally described, has only one format.
  - The file is an output file
  - The file is an input file and no random IO operations are done (ie READE, READPE, SETGT, SETLL, and CHAIN)
- Default blocking is 4K, so number or records that will fit into 4K are blocked.
- Explicitly request record blocking by:
  - Specifying the keyword BLOCK(*YES) on the file-description specification for the file
  - Using the CL command OVRDBF (Override with Database File) with SEQONLY((*YES #of records)) specified

Using Blocked IO to Reduce CPU… and IO

```
5761WDS V7R1M0  100416                SEU SOURCE LISTING                    11/02/12 12:18:50    LISAW         PAGE     1
SOURCE FILE . . . . . .  EDGE3/SOURCE
MEMBER  . . . . . . . .  READERNB
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
  100       *                                                                                          05/16/12
  200      H ALWNULL(*INPUTONLY)                                                                       05/16/12
  300       *                                                                                          05/16/12
  400      Fcustomer  IF   E             DISK    RENAME(customer:custr)                               10/24/12
  500      F*                                                                                          05/16/12
  600      C     1             SETLL     custr                                                        05/16/12
  700      C                                                                                           05/16/12
  800      C                                                                                           05/16/12
  900      C                   EVAL      *IN99 = *OFF                                                 05/16/12
 1000      C*                                                                                          05/16/12
 1100      C                   DOW       *IN99 = *OFF                                                 05/16/12
 1200      C                   READ      custr                                  99                    05/16/12
 1300      C                   ENDDO                                                                   05/16/12
 1400      C*                  MOVE      *ON           *INLR                                          10/02/12
 1500      C                   RETURN                                                                  05/16/12
 1600      C*                                                                                          05/16/12
                                * * * *  E N D  O F  S O U R C E  * * * *
```

```
5761WDS V7R1M0  100416                SEU SOURCE LISTING                    11/02/12 12:17:54    LISAW         PAGE     1
SOURCE FILE . . . . . .  EDGE3/SOURCE
MEMBER  . . . . . . . .  READER
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
  100       *                                                                                          05/16/12
  200      H ALWNULL(*INPUTONLY)                                                                       05/16/12
  300       *                                                                                          05/16/12
  400      Fcustomer  IF   E             DISK    RENAME(customer:custr) BLOCK(*YES)                   10/02/12
  500      F*                                                                                          05/16/12
  600      C     1             SETLL     custr                                                        05/16/12
  700      C                                                                                           05/16/12
  800      C                                                                                           05/16/12
  900      C                   EVAL      *IN99 = *OFF                                                 05/16/12
 1000      C*                                                                                          05/16/12
 1100      C                   DOW       *IN99 = *OFF                                                 05/16/12
 1200      C                   READ      custr                                  99                    05/16/12
 1300      C                   ENDDO                                                                   05/16/12
 1400      C*                  MOVE      *ON           *INLR                                          10/02/12
 1500      C                   RETURN                                                                  05/16/12
 1600      C*                                                                                          05/16/12
                                * * * *  E N D  O F  S O U R C E  * * * *
```

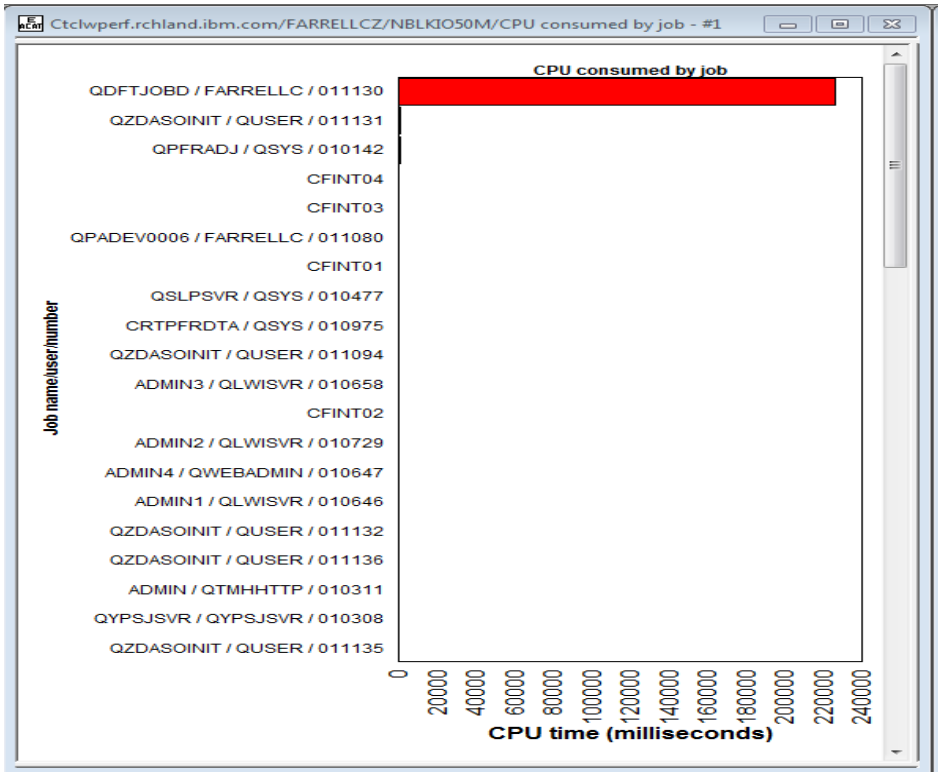
Using Blocked IO to Reduce CPU… and IO

```
5761WDS V7R1M0  100416                SEU SOURCE LISTING                    11/02/12 12:19:03    LISAW        PAGE    1
SOURCE FILE . . . . . .  EDGE3/SOURCE
MEMBER  . . . . . .  READERCLNS
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
  100 PGM                                                                                                  05/16/12
  200             DCL        VAR(&CNT) TYPE(*DEC) LEN(6 0) VALUE(1)                                        05/16/12
  300                                                                                                      05/16/12
  400 LOOP:                                                                                                05/16/12
  500             CALL       PGM(EDGE3/READERNB)                                                           10/26/12
  600             CHGVAR     VAR(&CNT) VALUE(&CNT + 1)                                                     05/16/12
  700             IF         COND(&CNT < 11) THEN(GOTO CMDLBL(LOOP))                                       10/26/12
  800 ENDPGM                                                                                               05/16/12
                          * * * *  E N D  O F  S O U R C E  * * * *
```

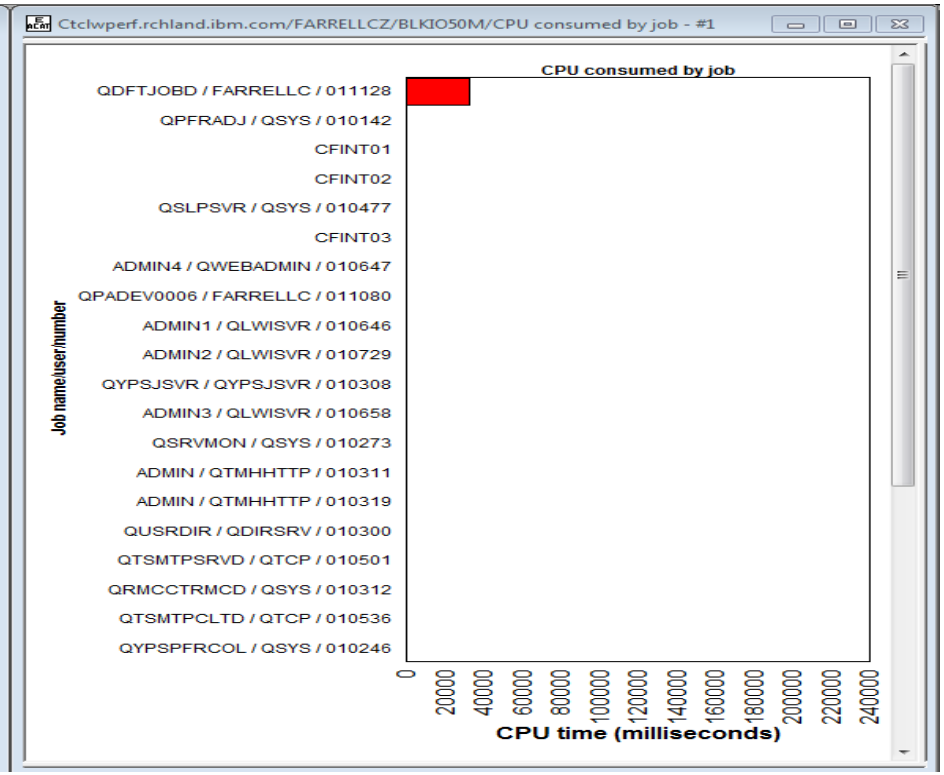
```
5761WDS V7R1M0  100416                SEU SOURCE LISTING                    11/02/12 12:18:04    LISAW        PAGE    1
SOURCE FILE . . . . . .  EDGE3/SOURCE
MEMBER  . . . . . .  READERCL
SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
  100 PGM                                                                                                  05/16/12
  200             DCL        VAR(&CNT) TYPE(*DEC) LEN(6 0) VALUE(1)                                        05/16/12
  300             OVRDBF     FILE(CUSTOMER) TOFILE(CLASS5M/CUSTOMER) +                                     10/02/12
  400                          MBR(CUSTOMER) SEQONLY(*YES *BUF32KB)                                        10/02/12
  500                                                                                                      05/16/12
  600 LOOP:                                                                                                05/16/12
  700             CALL       PGM(edge3/reader)                                                             10/02/12
  800             CHGVAR     VAR(&CNT) VALUE(&CNT + 1)                                                     05/16/12
  900             IF         COND(&CNT < 50000001) THEN(GOTO CMDLBL(LOOP))                                 06/06/12
 1000 ENDPGM                                                                                               05/16/12
                          * * * *  E N D  O F  S O U R C E  * * * *
```

Job Level CPU Consumed – Blocked IO

CPU Consumed – Non-Blocked IO



CPU Consumed – Blocked IO



STATS Collection at Program Level – Blocked IO

CPU Consumed – Non-Blocked IO

Ctclwperf.rchland.ibm.com/FARRELLCZ/NBLKIO50M/Summarized CPU and I/O by pgm/MI instruction - #1

MI complex instruction	Library name	Program name	Module name	Procedure short name	Inline CPU percent of total	Inline elapsed time percent of total	Times called	Calls made	MI complex instruction count	Inline CPU usecs	Cumulative CPU usecs
	QSYS	QCMD	QCMD	QCMD	.0001	.0000	0	11	28	196.4330	129,785,448.3280
	EDGE3	TESTRUN	TESTRUN	TESTRUN	.0000	.0000	0	6	0	5.8840	129,607,256.7320
	EDGE3	READERCLNS	READERCLNS	READERCLNS	.0000	.0000	1	23	0	57.9640	129,603,730.8410
	EDGE3	READERNB	READERNB	_QRNP_PEP_READERNB	19.0636	1.0869	10	50,000,021	11	24,983,710.4800	129,603,178.2690
	QSYS	QDBGETSQ	QDBGETSQ	QDBGETSQ	37.5210	1.6663	50,000,026	0	100,000,042	49,172,894.8890	104,619,443.7370
*SETCR					21.5069	.7475	50,000,182	0	0	28,185,719.0380	28,185,719.0380
*RETDSEN					20.8018	.7225	50,000,123	0	0	27,261,611.0940	27,261,611.0940

CPU Consumed – Blocked IO

Ctclwperf.rchland.ibm.com/FARRELLCZ/BLKIO50M/Summarized CPU and I/O by pgm/MI instruction - #1

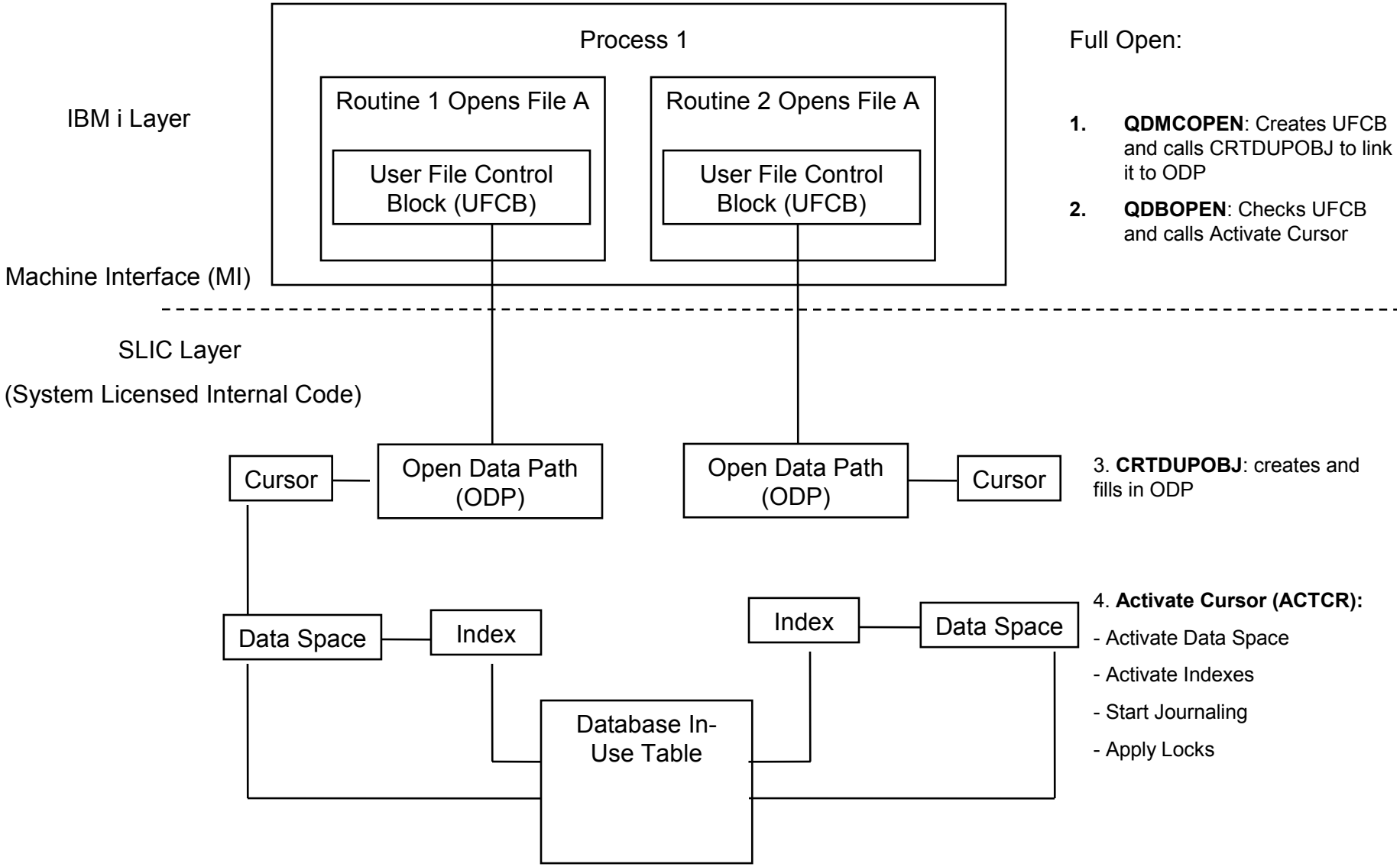
MI complex instruction	Library name	Program name	Module name	Procedure short name	Inline CPU percent of total	Inline elapsed time percent of total	Times called	Calls made	MI complex instruction count	Inline CPU usecs	Cumulative CPU usecs
	QSYS	QCMD	QCMD	QCMD	.0000	.0000	0	6	4	6.7940	32,043,943.5830
	EDGE3	TESTRUN	TESTRUN	TESTRUN	.0000	.0000	0	7	0	9.4170	32,042,285.2750
	EDGE3	READERCLS	READERCLS	READERCLS	.0002	.0000	1	26	0	63.1890	32,039,038.4270
	EDGE3	READER	READER	_QRNP_PEP_READER	4.4842	.4953	10	490,231	11	1,438,261.5930	32,038,319.3510
	QSYS	QDBGETM	QDBGETM	QDBGETM	1.6321	.2313	490,220	0	490,250	523,491.6990	30,599,883.4060
*RETSDSE					93.7715	8.5150	490,210	0	0	30,076,361.4100	30,076,361.4100
	QSYS	QWCPMNRR	QWCPMNRR	QWCPMNRR	.0001	.0000	0	0	7	44.4860	26,257.5130

STATS HIERARCHAL Program FLOW – NON-Blocked IO

Call level	Partial count status	Library name	Program name	MI complex instruction	Module name	Procedure name	Times called	Calls made	Calls to MI complex instructions	Inline CPU us	Inline percent CPU	Inline CPU us per call
0	Y	QSYS	QCMD		QCMD	QCMD	0	1	0	0	0	0
1	Y	EDGE3	TESTRUNHR		TESTRUNHR	TESTRUNHR	0	3	0	2.4410	.0000	0
2	Y	QSYS	QYPESTRP		QYPESTRP	_CXX_PEP__Fv	0	1	0	683.4040	.0028	0
3	N	QSYS	QMHSNDPM		QMHSNDPM	QMHSNDPM	1	0	10	10.0010	.0000	10.0010
4	N			*RSLVSP			2	0	0	8.9100	.0000	4.4550
4	N			*TESTAU			2	0	0	.5970	.0000	.2985
4	N			*FNDINXEN			2	0	0	5.2530	.0000	2.6265
4	N			*MATINVIF			1	0	0	.7050	.0000	.7050
4	N			*MATINVAT			2	0	0	2.8780	.0000	1.4390
4	N			*SNDPRMSG			1	0	0	3.4550	.0000	3.4550
2	N	QSYS	QCLCLCPR		QCLCLCPR	QCLCLCPR	1	0	3	9.8470	.0000	9.8470
3	N			*MATPTR			2	0	0	1.7530	.0000	.8765
3	N			*RSLVSP			1	0	0	12.2360	.0001	12.2360
2	Y	EDGE3	READERCLNS		READERCLNS	READERCLNS	1	5	0	43.7280	.0002	43.7280
3	N	QSYS	QCLRSLV		QCLRSLV	QCLRSLV	1	1	3	6.5050	.0000	6.5050
4	N			*MATINVAT			1	0	0	1.3610	.0000	1.3610
4	N			*MATPTRIF			1	0	0	1.2360	.0000	1.2360
4	N	QSYS	QLIADOPT		QLIADOPT	QLIADOPT	1	0	1	.2810	.0000	.2810
5	N			*RSLVSP			1	0	0	.7360	.0000	.7360
4	N			*DEQ			1	0	0	1.7670	.0000	1.7670
3	N	QSYS	QCLCLCPR		QCLCLCPR	QCLCLCPR	2	0	5	293.7240	.0012	146.8620
4	N			*MATPTR			1	0	0	.6730	.0000	.6730
4	N			*RSLVSP			2	0	0	3.9100	.0000	1.9550
4	N			*LOCKSL			1	0	0	1.6730	.0000	1.6730
4	N			*UNLOCKSL			1	0	0	.8920	.0000	.8920
3	Y	EDGE3	READERNB		READERNB	_QRNP_PEP_READERNB	2	9,565,604	2	4,921,443.6540	20.1132	2,460,721.8270
4	N			*INVP			1	0	0	.1730	.0000	.1730
4	N			*MATINVIF			1	0	0	.1110	.0000	.1110
4	N	QSYS	QDMCOPEN		QDMCOPEN	QDMCOPEN	1	1	9	13.7050	.0001	13.7050
5	N			*MATINVIF			1	0	0	.1110	.0000	.1110
5	N			*RSLVSP			2	0	0	10.0030	.0000	5.0015
5	N			*MATPTR			1	0	0	.3300	.0000	.3300
5	N			*MATCRAT			1	0	0	.7670	.0000	.7670
5	N			*TESTEXCP			1	0	0	.8610	.0000	.8610
5	N			*CRTDOBJ			1	0	0	20.0170	.0001	20.0170
5	N			*SETACST			1	0	0	2.1110	.0000	2.1110
5	N			*LOCK			1	0	0	2.4860	.0000	2.4860
5	N	QSYS	QDBOPEN		QDBOPEN	QDBOPEN	1	0	6	11.9000	.0000	11.9000
6	N			*MATPTR			2	0	0	1.0350	.0000	.5175
6	N			*MATDSAT			2	0	0	1.3780	.0000	.6890
6	N			*MATTHIF			1	0	0	.7980	.0000	.7980
6	N			*ACTCR			1	0	0	9.9230	.0000	9.9230
4	N	QSYS	QDBGETDR		QDBGETDR	QDBGETDR	2	0	1	10.3260	.0000	5.1630
5	N			*SETCR			1	0	0	11.1110	.0000	11.1110
4	Y	QSYS	QDBGETSQ		QDBGETSQ	QDBGETSQ	9,565,601	0	9,318,038	14,417,017.6050	78.9202	1.5072
5	N			*SETCR			4,659,019	0	0	2,569,731.6930	10.5021	.5516
5	N			*RETDSEN			4,659,019	0	0	2,559,324.8180	10.4596	.5493

Full Opens

Underlying Control Blocks of Database File – Full Open



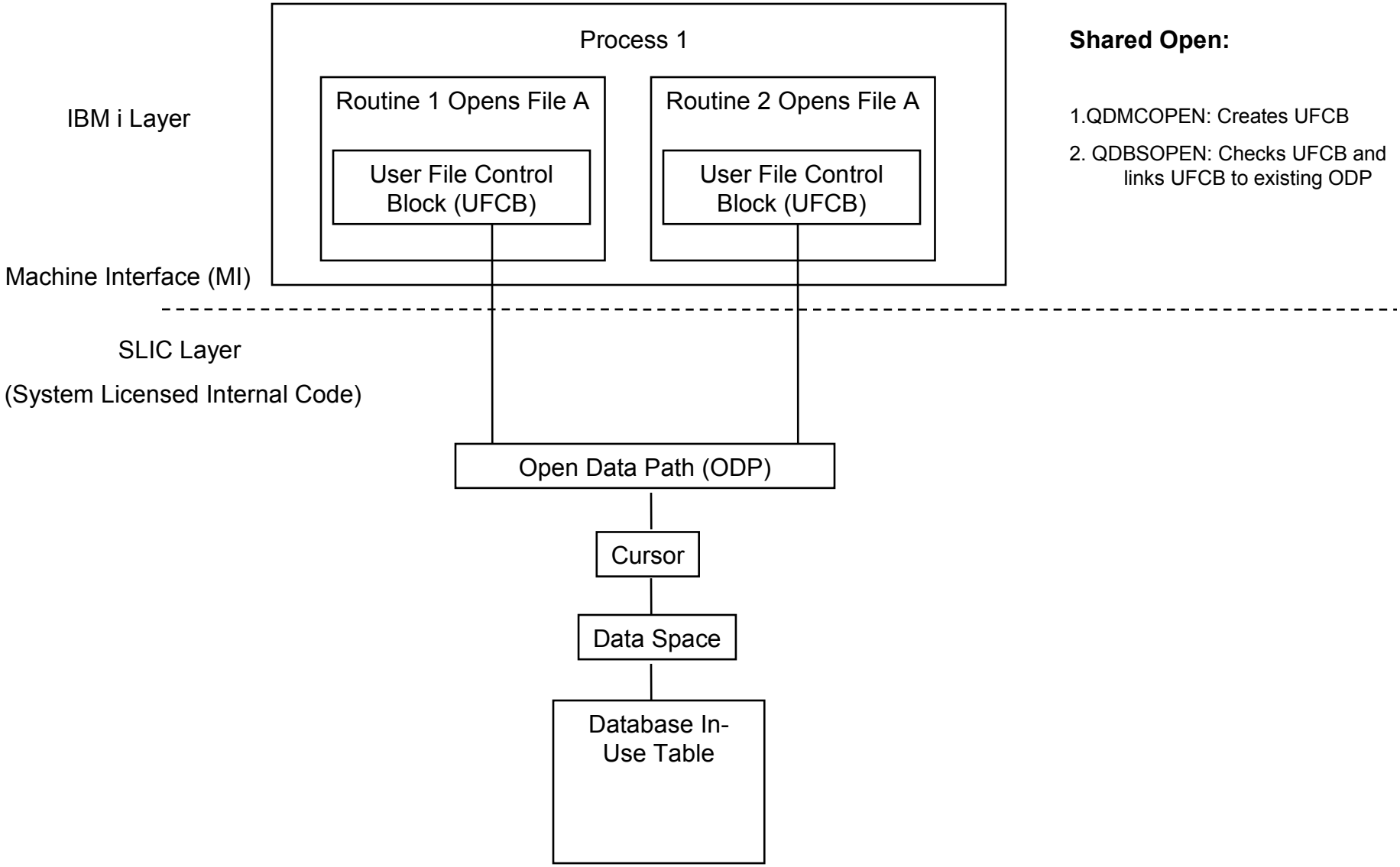
Full Open:

1. **QDMCOPEN**: Creates UFCB and calls CRTDUPOBJ to link it to ODP
2. **QDBOPEN**: Checks UFCB and calls Activate Cursor

3. **CRTDUPOBJ**: creates and fills in ODP

4. **Activate Cursor (ACTCR):**
- Activate Data Space
- Activate Indexes
- Start Journaling
- Apply Locks

Underlying Control Blocks of Database File – Shared Open



Shared Open:

1.QDMCOPEN: Creates UFCB

2. QDBSOPEN: Checks UFCB and links UFCB to existing ODP

Full (Hard) Close

- Unlock records
- Deactivate data spaces – Force changes to disk
- Deactivate index – Decrement index in use count
- Unlock data spaces
- Deletes the cursor and ODP
- Deletes the UFCB

- Shared Close
  - Objects left in use, no deactivate cursor, no forcing

Shared Open – Considerations

- Problem that arises from shared open is: Who is positioning the cursor
- If different programs in same job must have different positions at the same time, can't use shared opens
  - Can't share the same ODP
- When is it appropriate to do a Shared Open?
  - When a file is used by many different programs running in the same job
  - A file is used by a program that may get called many times within a job

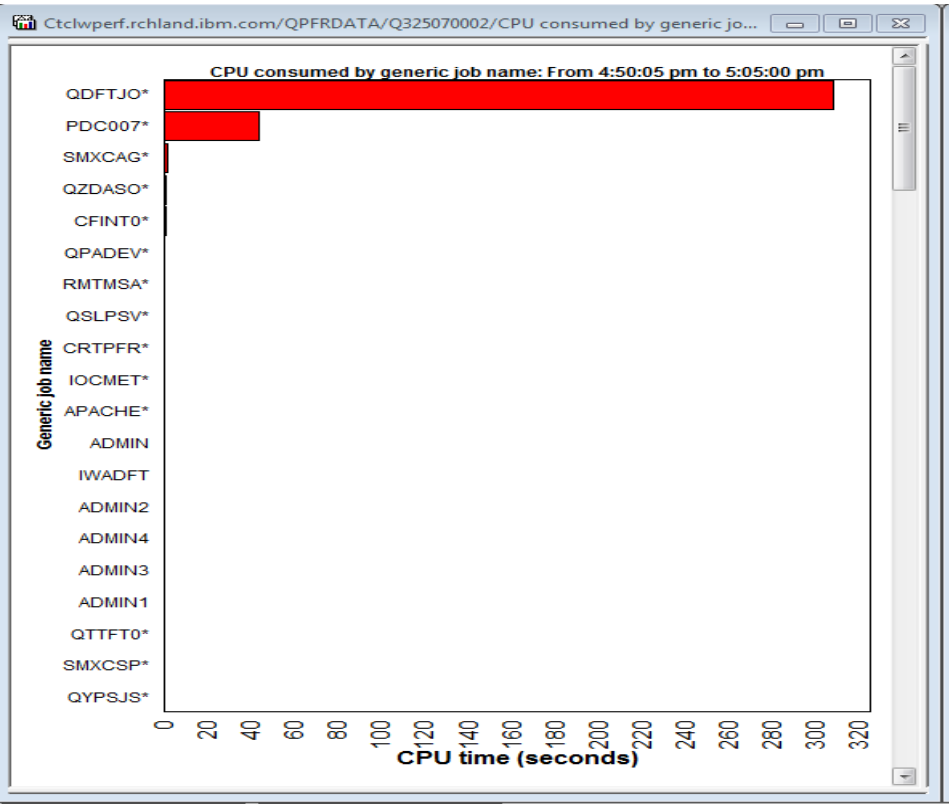
Using Shared Opens to Reduce CPU

```
5761WDS V7R1M0  100416                SEU SOURCE LISTING                    11/20/12 17:03:24    LISAW        PAGE    1
  SOURCE FILE . . . . . . .  COPREXLIB/CLASS
  MEMBER  . . . . . . . . .  CPUEXLOOPN
  SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
    100 PGM                                                                                                  10/22/12
    200 DCL &IBMDBLIB *CHAR 10 VALUE('CLASS5M   ')                                                           10/22/12
    300 DCL &LOOP *DEC (7 0)                                                                                 10/22/12
    400              ADDLIBLE   LIB(&IBMDBLIB)                                                               11/02/12
    500              DOUNTIL    COND(&LOOP = 3000000)                                                        11/06/12
    600              CHGVAR &LOOP (&LOOP + 1)                                                                10/22/12
    700              CALL       PGM(COPREXLIB/CPUEX1R1SO)                                                    11/02/12
    800              ENDDO /* &LOOP       */                                                                 10/25/12
    900 ENDPGM                                                                                               10/22/12
                              * * * *  E N D  O F  S O U R C E  * * * *
```

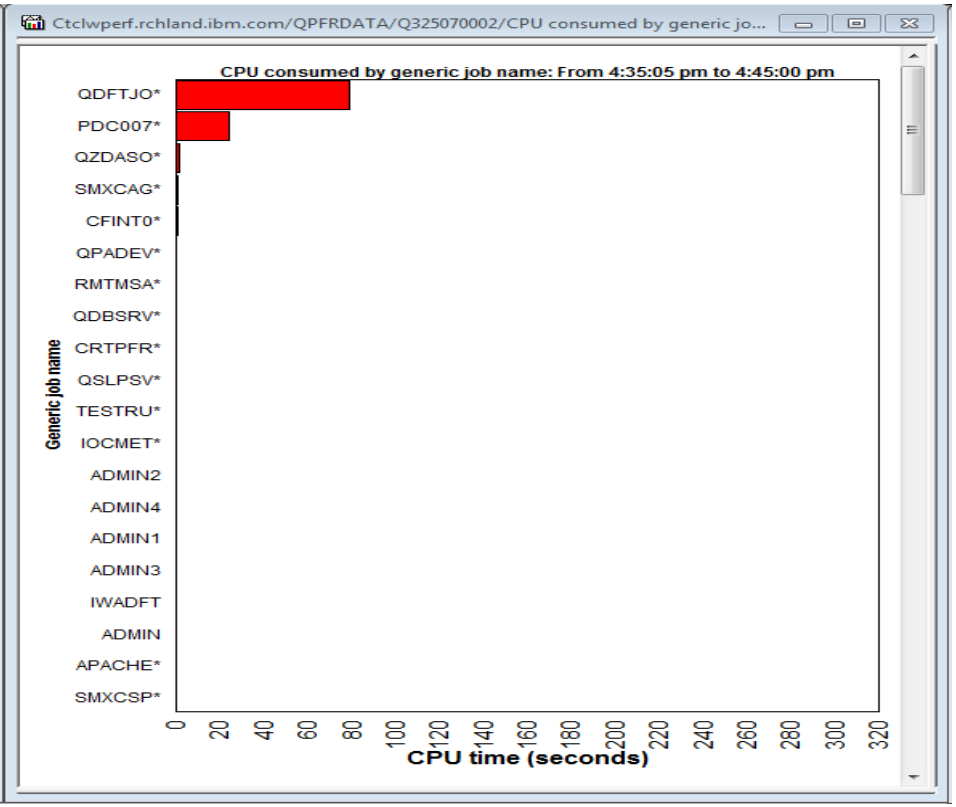
```
  5761WDS V7R1M0  100416                SEU SOURCE LISTING                    11/20/12 17:03:24    LISAW        PAGE    1
  SOURCE FILE . . . . . . .  COPREXLIB/CLASS
  MEMBER  . . . . . . . . .  CPUEXLOOPS
  SEQNBR*...+... 1 ...+... 2 ...+... 3 ...+... 4 ...+... 5 ...+... 6 ...+... 7 ...+... 8 ...+... 9 ...+... 0
    100 PGM                                                                                                  10/22/12
    200 DCL &IBMDBLIB *CHAR 10 VALUE('CLASS5M   ')                                                           10/22/12
    300 DCL &LOOP *DEC (7 0)                                                                                 10/22/12
    400              ADDLIBLE   LIB(&IBMDBLIB)                                                               11/02/12
    500              OVRDBF     FILE(CUSTOMER) TOFILE(CLASS5M/CUSTOMER) +                                    11/06/12
    600                           OVRSCOPE(*JOB) SHARE(*YES)                                                 11/06/12
    700              OPNDBF     FILE(CLASS5M/CUSTOMER) OPTION(*INP)                                          11/06/12
    800              DOUNTIL    COND(&LOOP = 3000000)                                                        11/06/12
    900              CHGVAR &LOOP (&LOOP + 1)                                                                10/22/12
   1000              CALL       PGM(COPREXLIB/CPUEX1R1SO)                                                    11/02/12
   1100              ENDDO /* &LOOP       */                                                                 10/25/12
   1200 ENDPGM                                                                                               10/22/12
                              * * * *  E N D  O F  S O U R C E  * * * *
```

Job Level CPU Consumed – Full Open

CPU Consumed – Full Open



CPU Consumed – Shared Open



STATS Collection at Program Level – Blocked IO full open vs shared open/close

Ctclwperf.rchland.ibm.com/FARRELLCZ/FULLOPEN/Summarized CPU and I/O by pgm/MI instruction - #5

MI complex instruction	Library name	Program name	Inline CPU percent of total	Inline elapsed time percent of total	Times called	Calls made	MI complex instruction count	Inline CPU usecs	Cumulative CPU usecs
	QSYS	QCMD	0	.0000	0	16	38	0	241,970,860.1510
	EDGE3	TESTRUN	.0000	.0000	0	6	0	5.7590	241,805,322.7200
	COPREXLIB	CPUEXLOOPN	1.2889	.0413	1	6,000,008	3	3,119,952.3780	241,800,161.1190
	COPREXLIB	CPUEX1R1SO	3.0846	.1356	3,000,000	6,000,000	3,000,001	7,466,578.5800	223,605,623.7220
	QSYS	QDMCOPEN	5.4889	.1790	3,000,022	3,000,044	27,000,222	13,286,288.28>	162,731,813.2160
	QSYS	QDBOPEN	5.1091	.2102	3,000,022	41	21,000,275	12,366,969.86>	76,073,504.2740
	QSYS	QDMCLOSE	2.9057	.1223	3,000,015	3,000,015	24,000,127	7,033,569.4650	53,084,223.7960
*CRTDOBJ			20.3029	.4171	3,000,020	0	0	49,629,151.03>	49,629,151.0380
*DESCR			17.0194	.3478	3,000,022	0	0	41,197,122.16>	41,197,122.1650
*MODS			13.0868	.2694	3,000,037	0	0	31,677,767.83>	31,677,767.8350
*ACTCR			11.7214	.2434	3,000,022	0	0	28,372,691.02>	28,372,691.0260
*RSLVSP			11.0369	.2439	12,001,5>	0	0	26,715,730.35>	26,715,730.3560
	QSYS	QCLCLCPR	1.8887	.0686	3,000,008	0	12,000,027	4,571,806.3020	15,073,851.0730
*MATPTR			1.2076	.0407	9,002,552	0	0	2,923,074.1300	2,923,074.1300
*LOCK			1.1113	.0280	3,000,260	0	0	2,690,079.2510	2,690,079.2510
*LOCKSL			.8006	.0310	6,000,361	0	0	1,938,009.2970	1,938,009.2970
*SETACST			.6451	.0190	3,000,374	0	0	1,561,627.2360	1,561,627.2360
*MATDSAT			.5350	.0223	6,000,116	0	0	1,295,080.2700	1,295,080.2700
*UNLOCK			.5257	.0162	3,000,260	0	0	1,272,419.6400	1,272,419.6400
*UNLOCKSL			.4135	.0193	6,000,361	0	0	1,001,019.8750	1,001,019.8750
*TESTEXCP			.3859	.0188	6,015,712	0	0	934,058.4750	934,058.4750
	QSYS	QDBCLOSE	.3051	.0137	3,000,022	2	6	738,514.9860	738,876.8650

Ctclwperf.rchland.ibm.com/FARRELLCZ/SHROPEN/Summarized CPU and I/O by pgm/MI instruction - #3

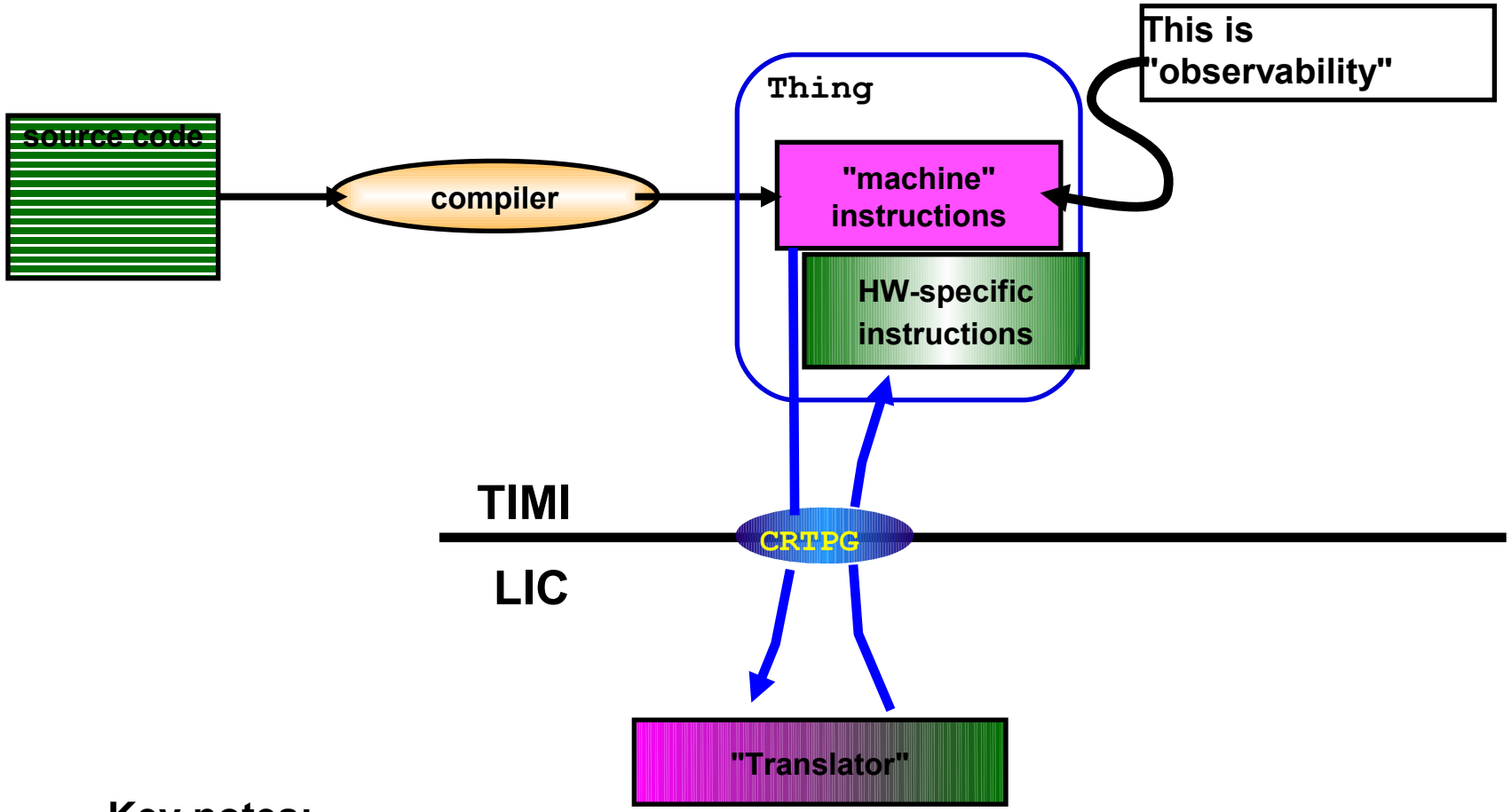
MI complex instruction	Library name	Program name	Inline CPU percent of total	Inline elapsed time percent of total	Times called	Calls made	MI complex instruction count	Inline CPU usecs	Cumulative CPU usecs
	QSYS	QCMD	0	.0000	0	15	10	0	41,989,192.5750
	EDGE3	TESTRUN	.0000	.0000	0	6	0	5.8840	41,915,867.1190
	COPREXLIB	CPUEXLOOPS	6.2287	.1741	1	6,000,014	3	2,616,434.0310	41,910,669.6730
	COPREXLIB	CPUEX1R1SO	14.5063	.5758	3,000,000	6,000,000	3,000,001	6,093,503.2830	25,600,082.4250
	QSYS	QCLCLCPR	9.5662	.3015	3,000,001	0	12,000,005	4,018,390.0390	13,693,290.9390
	QSYS	QDMCOPEN	17.2827	.5000	3,000,002	6,000,004	12,000,014	7,259,778.3780	13,264,619.6360
*RSLVSP			20.0309	.3826	6,000,310	0	0	8,414,174.2070	8,414,174.2070
	QSYS	QDMCLOSE	9.8989	.3061	3,000,001	1	12,000,010	4,158,112.7700	5,951,034.1400
	QSYS	QDMGETOV	6.6199	.1715	3,000,002	0	3,000,002	2,780,734.4100	2,939,475.9720
*LOCKSL			5.5782	.1714	9,000,119	0	0	2,343,174.8580	2,343,174.8580
*UNLOCKSL			3.5573	.1375	9,000,119	0	0	1,494,275.3920	1,494,275.3920
	QSYS	QDBSOPEN	2.5928	.0780	3,000,000	0	0	1,089,122.8590	1,089,122.8590
*TESTEXCP			1.6873	.0798	6,004,513	0	0	708,759.0350	708,759.0350
*MATINVIF			.8638	.0662	6,000,253	0	0	362,835.2070	362,835.2070
*INVP			.6960	.0388	3,004,440	0	0	292,341.6550	292,341.6550
*MATPRATR			.6750	.0375	3,004,803	0	0	283,534.6280	283,534.6280
	QSYS	QUIMGFLW	.0003	.0000	18	83	83	133.1130	43,965.9410
	QPDA	QUOCPP	0	.0000	0	3	0	0	43,965.9410
	QSYS	QUIMNDRV	0	.0000	0	2	0	0	43,965.9410
	QSYS	QUOCMD	0	0	0	2	0	0	43,965.9410
	QSYS	QUICMD	.0000	.0000	1	5	0	6.0800	43,965.9410
	QSYS	QUICMENU	0	0	0	2	0	0	43,965.9410

STATS HIERARCHAL Program FLOW – NON-Blocked IO

Call level	Partial count status	Library name	Program name	MI complex instruction	Module name	Procedure name	Times called	Calls made	Calls to MI complex instructions	Inline CPU us	Inline percent CPU	Inline CPU us per call
0	Y	QSYS	QCMD		QCMD	QCMD	0	1	0	0	0	0
1	Y	EDGE3	TESTRUNHR		TESTRUNHR	TESTRUNHR	0	3	0	2.4410	.0000	0
2	Y	QSYS	QYPESTRP		QYPESTRP	_CXX_PEP__Fv	0	1	0	683.4040	.0028	0
3	N	QSYS	QMHSNDPM		QMHSNDPM	QMHSNDPM	1	0	10	10.0010	.0000	10.0010
4	N			*RSLVSP			2	0	0	8.9100	.0000	4.4550
4	N			*TESTAU			2	0	0	.5970	.0000	.2985
4	N			*FNDINXEN			2	0	0	5.2530	.0000	2.6265
4	N			*MATINVIF			1	0	0	.7050	.0000	.7050
4	N			*MATINVAT			2	0	0	2.8780	.0000	1.4390
4	N			*SNDPRMSG			1	0	0	3.4550	.0000	3.4550
2	N	QSYS	QCLCLCPR		QCLCLCPR	QCLCLCPR	1	0	3	9.8470	.0000	9.8470
3	N			*MATPTR			2	0	0	1.7530	.0000	.8765
3	N			*RSLVSP			1	0	0	12.2360	.0001	12.2360
2	Y	EDGE3	READERCLNS		READERCLNS	READERCLNS	1	5	0	43.7280	.0002	43.7280
3	N	QSYS	QCLRSLV		QCLRSLV	QCLRSLV	1	1	3	6.5050	.0000	6.5050
4	N			*MATINVAT			1	0	0	1.3610	.0000	1.3610
4	N			*MATPTRIF			1	0	0	1.2360	.0000	1.2360
4	N	QSYS	QLIADOPT		QLIADOPT	QLIADOPT	1	0	1	.2810	.0000	.2810
5	N			*RSLVSP			1	0	0	.7360	.0000	.7360
4	N			*DEQ			1	0	0	1.7670	.0000	1.7670
3	N	QSYS	QCLCLCPR		QCLCLCPR	QCLCLCPR	2	0	5	293.7240	.0012	146.8620
4	N			*MATPTR			1	0	0	.6730	.0000	.6730
4	N			*RSLVSP			2	0	0	3.9100	.0000	1.9550
4	N			*LOCKSL			1	0	0	1.6730	.0000	1.6730
4	N			*UNLOCKSL			1	0	0	.8920	.0000	.8920
3	Y	EDGE3	READERNB		READERNB	_QRNP_PEP_READERNB	2	9,565,604	2	4,921,443.6540	20.1132	2,460,721.8270
4	N			*INVP			1	0	0	.1730	.0000	.1730
4	N			*MATINVIF			1	0	0	.1110	.0000	.1110
4	N	QSYS	QDMCOPEN		QDMCOPEN	QDMCOPEN	1	1	9	13.7050	.0001	13.7050
5	N			*MATINVIF			1	0	0	.1110	.0000	.1110
5	N			*RSLVSP			2	0	0	10.0030	.0000	5.0015
5	N			*MATPTR			1	0	0	.3300	.0000	.3300
5	N			*MATCRAT			1	0	0	.7670	.0000	.7670
5	N			*TESTEXCP			1	0	0	.8610	.0000	.8610
5	N			*CRTDOBJ			1	0	0	20.0170	.0001	20.0170
5	N			*SETACST			1	0	0	2.1110	.0000	2.1110
5	N			*LOCK			1	0	0	2.4860	.0000	2.4860
5	N	QSYS	QDBOPEN		QDBOPEN	QDBOPEN	1	0	6	11.9000	.0000	11.9000
6	N			*MATPTR			2	0	0	1.0350	.0000	.5175
6	N			*MATDSAT			2	0	0	1.3780	.0000	.6890
6	N			*MATTHIF			1	0	0	.7980	.0000	.7980
6	N			*ACTCR			1	0	0	9.9230	.0000	9.9230
4	N	QSYS	QDBGETDR		QDBGETDR	QDBGETDR	2	0	1	10.3260	.0000	5.1630
5	N			*SETCR			1	0	0	11.1110	.0000	11.1110
4	Y	QSYS	QDBGETSQ		QDBGETSQ	QDBGETSQ	9,565,601	0	9,318,038	14,417,017.6050	58.9202	1.5072
5	N			*SETCR			4,659,019	0	0	2,569,731.6930	10.5021	.5516
5	N			*RETDSEN			4,659,019	0	0	2,559,324.8180	10.4596	.5493

Program Translator Optimization for CPU use

IBM i Program Model Architecture



Key notes:

* Programs get compiled to an intermediate level of code - MI
* The IBM i architecture incorporates "translation" under the TIMI
* The Translator is necessarily (and *notoriously*) hardware-specific

Compiler options OPTIMIZE(*basic, *full)

Optimize the use of processors by specifying the OPTIMIZE parameter on the CRTRPGMOD and CRTBNDRPG command, e.g. OPTIMIZE(*BASIC or *FULL).

- Optimization may rearrange or eliminate some statements altogether. Consequently, field values presented by the debugger, as well as breakpoints and step locations, may be inaccurate.
- CRTRPGMOD and CRTBNDRPG commands):
  - **OPTIMIZE(*NONE)** is the default value. The compiler performs no optimization on the code. This option allows you to debug the program accurately, with the debugger presenting correct current field values.
  - **OPTIMIZE(*BASIC)** performs some optimization on the compiled code. The debugger will let you display variables, but the displayed value may not be the current value. The debugger will also allow variable changes, but you may encounter unexpected results if you try to use the changed variables.
  - **OPTIMIZE(*FULL)** generates the most efficient code -- and the compile process also takes longer. The program should run faster than with either of the other two options. The debugger will let you see (but not change) field values; but the debugger may not show you the correct current field values.

Compiler option OPTIMIZE(*basic, *full) – cont.

You can also use the commands **CHGMOD, CHGPGM or CHGSRVPGM** and the parameter OPTIMIZE(*YES) to optimize instructions being executed within your program. The DSPMOD, DSPPGM, and DSPSRVPGM commands will show you the current level of optimization for an object.

You can change optimization levels with the CHGMOD, CHGPGM, or CHGSRVPGM commands; these commands support the levels already discussed, but also add other options:

- **OPTIMIZE(*NONE|*NO|10)** -- These options are equivalent, and are discussed above. OPTIMIZE(*NO) is used only by the CHGPGM command.
- **OPTIMIZE(*BASIC|20)** -- These options are equivalent, and are discussed above.
- **OPTIMIZE(*FULL|30)** -- These options are equivalent, and are discussed above.
- **OPTIMIZE(*YES|40)** -- This is the highest possible optimization level. In addition to all level 30 optimization, level 40 optimization prevents call and instruction tracing. OPTIMIZE(*YES) is used only by the CHGPGM command.

Program profiling

Optimize your ILE RPG or COBOL programs by profiling and collecting procedure execution information and applying them to optimize the program structures and instruction flows

- Advanced optimization technique to reorder procedures, or code within procedures, in ILE programs and service programs based on statistical data gathered while running the program
- This reordering can improve instruction cache utilization and reduce paging required by the program, thereby improving performance.
- The semantic behavior of the program is not affected by program profiling.

Program profiling

- The performance improvement realized by program profiling depends on the type of application
- Can expect more improvement from programs that spend the majority of time in the application code itself, rather than spending time in the runtime or doing input/output processing.
- The performance of program code produced when applying profile data depends upon the optimizing translator correctly identifying the most important portions of the program during typical use
- Therefore, it is important to gather profile data while performing the tasks that will be performed by end users, and using input data that is similar to that expected in the environment in which the program will be run.
- Program profiling is available only for ILE programs and service programs and were compiled using an optimization level of *FULL (30) or above.
- Note: Because of the optimization requirements, you should fully debug your programs before using program profiling.

How to enable and start Program profiling

- Enable the program to collect profiling data
- Start the program profiling collection on the system with the Start Program Profiling (STRPGMPRF) command
- Collect profiling data by running the program through its high-use code paths. Because program profiling uses statistical data gathered while running the program to perform these optimizations, it is critical that this data be collected over typical uses of your application.
- End the program profiling collection on the system with the End Program Profiling (ENDPGMPRF) command.
- Apply the collected profiling data to the program by requesting that code be reordered for optimal performance based on the collected profiling data.

Advanced Argument Optimization

- Advanced argument optimization is a cross-module optimization that is used to improve the performance of programs that contain frequently run procedure calls, including C++ applications that make mostly nonvirtual method calls.
- Improved runtime performance is achieved by enabling the translator and binder to use the most efficient mechanisms to pass parameters and return results between procedures that are called within a program or service program.
- The Argument optimization (ARGOPT) parameter, with *YES and *NO as possible values, is available on the CRTPGM and CRTSRVPGM commands to support advanced argument optimization. Specifying ARGOPT(*YES) causes the program or service program to be created with advanced argument optimization. The default is *NO
- Can only be used at 6.1 and later
- For programs that consist of hundreds or thousands of modules, creation time can be significantly longer

Adaptive Code Generation

- Generally, you do not need to understand the details of the underlying hardware architecture so that the architecture can change smoothly over time
- In releases before 6.1, the optimizing translator for a given release was designed to generate only instructions that would run on all system models supported by that release
- As of 6.1, you can take advantage of all processor features on your systems, regardless of whether those features are present on other system models supported by the same release.
- Furthermore, programs can be moved from one system model to another and continue to run correctly, even if the new machine does not have all the processor features available on the original one. The technology used to achieve this is called adaptive code generation
- A *hardware feature* is a feature that has been added to the family of processors supported by IBM® i. For instance, one new feature available in some processors supported by 6.1, is a new hardware decimal floating-point unit. A processor with this unit is considered by ACG to have the decimal floating-point feature, while a processor without it does not have the decimal floating-point feature. The aggregation of all added features present in a processor is called that processor's feature set

Adaptive Code Generation

- When you create a program object using a command such as Create Program (CRTPGM) or Create Service Program (CRTSRVPGM), the binder determines the feature set for the program object and stores it as part of the program object.

- The first time the program object is activated on your system, the system checks to be sure that the program object is compatible with the target model associated with your system; that is, it ensures that your program does not use any features that are not available on your system.
  - Because you compiled the program object on this system, the program object always passes this compatibility check, and the program runs correctly

- Suppose you want to migrate this program object to another system that uses the same release, but has a different target model. The first time the program is activated on the system to which it is moved, the system performs the compatibility check against this system's target model
  - **If compatible, it is ok to run**
  - if the program requires any processor feature that is not supported by the system to which it was moved, then the system automatically **calls the optimizing translator to convert the program to be compatible**

Adaptive Code Generation
- the CodeGenTarget LICOPT

- The CodeGenTarget=Power6 option might be useful if you have an older machine that you use for software builds, but you want to deploy your software to machines using POWER6® processors. This option causes the optimizing translator to use POWER6 features, even though they might not be available on the current machine.
- The programs can then be deployed to your POWER6 systems without any conversion during the restore processing or the first activation.
  - However, the programs cannot be activated on the build machine without being converted.

- You need to ensure that programs are not activated on the build machine; otherwise, the POWER6 features are removed from the programs

- Features associated with POWER6 hardware include:
  - A hardware decimal floating-point unit
  - Efficient hardware support for ILE pointer handling

- Features associated with POWER7 hardware include:
  - A number of new instructions that may speed up certain computations, such as conversions between integer and floating-point values

Interprocedural analysis (IPA)

- At compile time, the optimizing translator performs both intraprocedural and interprocedural analysis.
  - Intraprocedural analysis is a mechanism for performing optimization for each function within a compilation unit, using only the information available for that function and compilation unit.
  - Interprocedural analysis is a mechanism for performing optimization across function boundaries. The optimizing translator performs interprocedural analysis, but only within a compilation unit.
- Interprocedural analysis that is performed by the IPA compiler option improves on the limited interprocedural analysis described above. When you run interprocedural analysis through the IPA option, IPA performs optimizations across the entire program. It also performs optimizations not otherwise available at compile time with the optimizing translator

Interprocedural analysis (IPA)

- The optimizing translator or the IPA option performs the following types of optimizations:
  - Inlining across compilation units
  - Program partitioning, reordering functions
  - Coalescing of global variables
  - Code straightening
  - Unreachable code elimination
  - Conversion of reference to value arguments
- Applications that will most likely show performance gains are those that have the following characteristics:
  - Contain a large number of functions
  - Contain a large number of compilation units
  - Contain a large number of functions that are not in the same compilation units as their callers
  - Do not perform a large number of input and output operations
- You should fully debug your programs before you use interprocedural analysis

Identifying programs and procedures –

program hot spot analysis

STATS Flat Analysis Provides …

Information for **all or selected** jobs in the collection that shows:

1. How many times it was called
2. How many times it called another program
3. It's total elapsed time
4. The number of MI Complex Instructions the program called

for each called program/module/procedure that is enabled for collection and for each MI Complex Instruction, a single output line per job or collection wide.

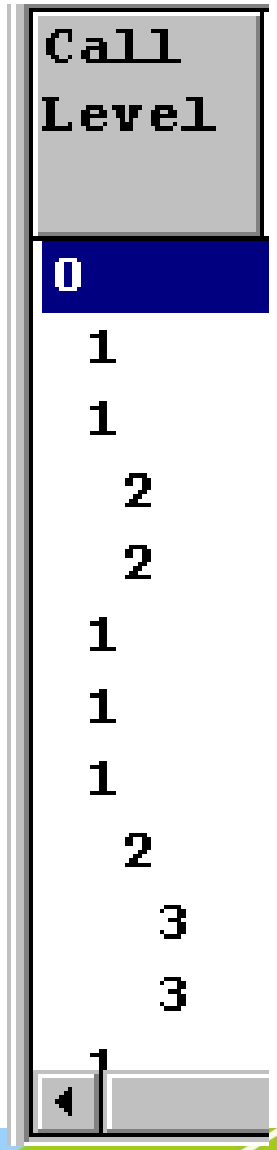
STATSHIER Provides

- Program/MI Complex Instruction/Module/Procedure within a selected job
- Times each program/MI complex instruction was called
- Calls programs made (NOTE: MI Complex Instructions are only called)
- Elapsed time
- Inline CPU
- Cumulative

STATSHIER Provides (Cont.)

- Disk IO counts
- Call levels
- The level of control a program/MI instruction is running at
- What program called it (all the way back via call stacks)
- What program or MI complex instructions it called (all the way down to MI level via call stacks)

Data Analysis



A Level 0 program
called this Level 1
and this Level 1
The second Level 1 called this Level 2
and this Level 2
Then the Level 0 called this Level 1
and this Level 1
and this Level 1
Which called this Level 2
Which called this Level 3
and this Level 3

The Call Level represents who called and whose calling

Summary

- IBM i Application Programs can be optimized for the use of CPU and I/Os or both
- Most of the optimization occurs outside the program as part of the IBM i translator
- Optimization can significantly improve performance and runtimes by
  - Reducing CPU and wait times
  - Streamlining object code
- Optimization techniques can lead to larger programs and can cause inability to debug