e-business

Performance Tips and Techniques for VSE COBOL and Language Environment

Disclaimer

The performance numbers given in this presentation were originally created with the MVS environment and have not been validated for VSE. However, the numbers can be used to show the relative savings that can be achieved under VSE.

Topics

- Compiler options for speed and profit
- Run-time options for speed and efficiency
- COBOL and LE features affecting run-time performance
- COBOL coding tips
- CICS performance considerations
- Year 2000 performance considerations

Compiler options for speed and profit

- **NUMPROC**
  - PFD is fastest, but MIG is faster than NOPFD
    - 11% faster to equal
- **OPTIMIZE**
  - OPTIMIZE(STD) is 4% faster than NOOPT, with a range of 17% faster to equal
  - OPTIMIZE(FULL) can be much faster for programs with VALUE clauses that are called in initial state (CALL and CANCEL, PROGRAM IS INITIAL)
    - One program that had 500 unreferenced data items was 80% faster with OPT(FULL) over STD
- **RENT**
  - and NORENT are equivalent

Compiler options for speed and profit ...

- **SSRANGE**
  - Compiled w/SSRANGE, run with CHECK(ON) is 4.3% slower than NOSSRANGE
  - Compiled w/SSRANGE, run with CHECK(OFF) is 2.1% slower than NOSSRANGE
  - Compiled w/SSRANGE, run with CHECK(ON) is 1.4% slower than with CHECK(OFF)
- **TEST**
  - TEST(ALL,SYM) is 14% slower than NOTEST, range equal to 59% slower
  - TEST(NONE,SYM) is equal to NOTEST
- **TRUNC**
  - TRUNC(OPT) is 27% faster than TRUNC(BIN) range 92% faster to equal
  - TRUNC(STD) is 26% faster than TRUNC(BIN) range 88% faster to equal
  - TRUNC(OPT) is 5% faster than TRUNC(STD) range 49% faster to equal

Compiler options for speed and profit ...

- **Recommendations for speed and useability**
  - NOCMPR2
  - FASTSRT
  - NUMPROC(MIG)
  - OPTIMIZE(STD)
  - RENT
  - NOSSRANGE
  - NOTEST or TEST(NONE,SYM)
  - TRUNC(OPT)

Run-time options for speed and efficiency

- **AIXBLD**
  - For VSAM files with alternate indexes
  - One VSAM program using AIXBLD is 8% slower than with NOAIXBLD
- **ALL31**
  - ALL31(ON) is 1% faster than ALL31(OFF), range 4% faster to equal
  - One program with many library routine calls was 10% faster wuth ALL31(ON)
- **CBLPSHPOP**
  - Changes behavior of CICS condition handling
  - CBLPSHPOP(OFF) much faster for 2nd thru nth EXEC CICS LINK
- **CHECK**
  - CHECK(ON) with SSRANGE is 2% slower than with CHECK(OFF)
    - range equal to 20% slower

Run-time options for speed and efficiency ...

- **RPTOPTS**
  - RPTOPTS(ON) is equivalent to RPTOPTS(OFF), range equal to 2% slower
- **RPTSTG**
  - RPTSTG(ON) is 5% slower than RPTSTG(OFF)
    - range equal to 37% slower
  - RPTSTG(ON) can degrade CALL intensive applications 160% or more!
- **RTEREUS**
  - For special cases only! Bad side-effects
  - Overhead of assembler MAIN calling COBOL sub is 99% less
- **STORAGE**
  - STORAGE(00,00,00) is 7% slower than STORAGE(NONE,NONE,NONE)
    - Range equal to 57% slower
    - CALL intensive applications can be 160% slower

Run-time options for speed and efficiency ...

- **STORAGE tuning**
  - Use the RPTSTG(ON) option to get storage reports
  - Use the values returned by the RPTSTG(ON) option as the size of the initial blocks of storage for the HEAP, ANYHEAP, BELOWHEAP, STACK, and LIBSTACK run-time options
- **IBM defaults for storage options(non-CICS)**
  - ANYHEAP(16K,8K,ANYWHERE,FREE)
  - BELOWHEAP(8K,4K,FREE)
  - HEAP(32K,32K,ANYWHERE,KEEP,8K,4K)
  - LIBSTACK(8K,4K,FREE)
  - STACK(128K,128K,BELOW,KEEP)
- **For COBOL-only applications:**
  - STACK(64K,64K,BELOW,KEEP)
- **For AMODE(31) applications:**
  - STACK(,,ANYWHERE,)

Run-time options for speed and efficiency ...

- **Recommendations for speed and useability**
  - NOAIXBLD
  - ALL31(ON)
    - if CICS or if all programs are AMODE=31
  - CHECK(OFF)
  - NODEBUG
  - RPTOPTS(OFF)
  - RPTSTG(OFF)
  - NORTEREUS
  - STORAGE(00,,,)
    - if coming from WSCLEAR environment
  - STORAGE(NONE,,,)
    - if coming from NOWSCLEAR environment
  - NOTEST or TEST(NONE,SYM)
  - TRAP(ON)

COBOL/LE features affecting performance

Assembler driver calling COBOL programs repeatedly

- Use a REUSABLE run-time environment

- **ILBDSET0, IGZERRE**
  - ILBDSET0 will set up both DOS/VS COBOL and COBOL portion of Language Environment
  - IGZERRE sets up only COBOL portion
  - ILBDSET0 should be converted to IGZERRE
  - Assembler main calling IGZERRE before calling COBOL is 99% faster than not calling IGZERRE first

- **CEEENTRY and CEETERM**
  - LE-conforming assembler MAIN calling COBOL is 99% faster than non LE-conforming

- **CEEPIPI**
  - Similar to IGZERRE, but works for all languages.
  - Can invoke MAIN or SUB programs

- **Add a COBOL stub in front of assembler driver**

COBOL/LE features affecting performance ...

Mixing older COBOL programs with newer ones

- **DOS/VS COBOL mixed with COBOL for VSE/ESA**
  - Both the DOS/VS COBOL and COBOL for VSE/ESA parts of Language Environment must be used
  - Costs extra at INIT and TERM
  - Converting to COBOL for VS/ESA will avoid DOS/VS COBOL run-time init and term

COBOL coding tips - table element references

- **Subscripting**
  - External decimal (USAGE DISPLAY)
  - Binary
- **Indexing**
  - INDEXED BY phrase of OCCURS
  - USAGE IS INDEX
- **Which is fastest, and which slowest?**
  - External Decimal = 4 instructions per reference
  - Binary with TRUNC(OPT or STD) = 2 instructions
  - Binary with TRUNC(BIN) = 4 instructions including CVD
  - Indexes = 1 instruction
- **Slowest is binary with TRUNC(BIN)**
- **For programs with lots of table processing, use INDEXES to really speed things up!**

COBOL coding tips - table element references ...

- **Given these data descriptions:**

```
77 SUB1  PIC 9(4) USAGE BINARY.
01  GRP1.
    05  TAB1 OCCURS 1000 INDEXED BY TABINDX.
        10 SALES       PIC 9(7) PACKED-DECIMAL.
        10 EXPENSES  PIC 9(7) PACKED-DECIMAL.
        10 INVENTORY PIC 9(7) PACKED-DECIMAL.
```

- **Slow code:**

```
PERFORM   VARYING SUB1
                  FROM 1 BY 1
                  UNTIL SUB1 > 1000
   COMPUTE SALES-TOTAL = SALES-TOTAL + SALES(SUB1)
   COMPUTE EXPENSE-TOTAL = EXPENSE-TOTAL + EXPENSES(SUB1)
   COMPUTE INVENTORY-TOTAL = INVENTORY-TOTAL + INVENTORY(SUB1)
END-PERFORM
```

- **Fast code:**

```
PERFORM   VARYING TABINDX
                  FROM 1 BY 1
                  UNTIL TABINDX > 1000
   COMPUTE SALES-TOTAL = SALES-TOTAL + SALES(TABINDX)
   COMPUTE EXPENSE-TOTAL = EXPENSE-TOTAL + EXPENSES(TABINDX)
   COMPUTE INVENTORY-TOTAL = INVENTORY-TOTAL + INVENTORY(TABINDX)
END-PERFORM
```

COBOL coding tips - I/O performance

VSAM files/datasets

- **Increase number of buffers**
  - Data buffers (BUFND) for sequential access
  - Index buffers (BUFNI) for random access
- **Control interval size (CISZ)**
  - A smaller CISZ results in faster retrieval for random processing at the expense of inserts
  - A larger CISZ is more efficient for sequential processing
  - In general: large CISZ and buffer space may improve performance
- **Fastest access mode? In general:**
  - SEQUENTIAL is fastest
  - DYNAMIC is next
  - RANDOM access is the least efficient
- **VSAM buffers above the 16 MB line**
  - Programs compiled with VS COBOL II Release 3.2 or later
  - Running under Language Environment/VSE 1.4

CICS performance considerations - bad news

- **EXEC CICS LINK slower under Language Environment than under VS COBOL II**
  - Each LINK or XCTL creates a new run-unit (enclave)
  - VS COBOL II transactions with lots of CICS LINKs can be up to 50% slower
- **Language Environment uses more storage under CICS than VS COBOL II**
  - May need to increase CICS extended user DSA
  - Will need to increase CICS user DSA if using ALL31(OFF)
  - Why more storage?
    - Language Environment phases are bigger than VS COBOL II
    - Language Environment has bigger control blocks than VS COBOL II
    - Language Environment has more pools of storage than VS COBOL II

CICS performance considerations - CALL

- **EXEC CICS LINK much slower than COBOL CALL**
  - TestcaseA:
    COB1--LINK-->COB2--LINK-->COB3
    COB1 EXEC CICS LINK to COB2 1000 times
    return via EXEC CICS RETURN
  - TestcaseB:
    COB1--DYNCALL-->COB2--DYNCALL->-COB3
    COB1 CALL identifier to COB2 1000 times
    return via GOBACK
  - TestcaseA CPU time: 0.85 SEC
  - TestcaseB CPU time: 0.17 SEC
  - EXEC CICS LINK has about 5 TIMES the overhead of COBOL dynamic CALL
- **Convert CICS LINK/XCTL to COBOL CALL**
  - Get even more benefit by going to CBLPSHPOP(OFF)!

CICS performance considerations - CALL ...

- **CBLPSHPOP run-time option**
  - Performance testing shows that overhead of CBLPSHPOP(ON) is significant!
    - No affect on EXEC CICS LINK/XCTL
  - For 2nd thru nth CALL to a program, overhead is 1500%!
  - Each CALL takes 15 TIMES the overhead of CBLPSHPOP(OFF)
- **When can I use CBLPSHPOP(OFF)?**
  - If no CICS HANDLE ABEND, HANDLE AID, HANDLE CONDITION, or IGNORE CONDITION statements
  - Any program that does their own EXEC PUSH/POP HANDLE
  - Any program that does not use CALL statements will not be affected

CICS performance considerations - good news!

- **New LE/VSE APAR PQ23382 and LE/VSE COBOL APAR PQ23385**
  - Reduces the overhead of a CICS LINK (available March 1999)
    - for an ALL31(ON) application by approximately 10%
    - for an ALL31(OFF) application achieved nearly a 30% CPU savings per CICS LINK
  - Includes VSCR which reduces amount of LE/VSE below the line storage by moving modules above 16mb line
- **Support for the new RUWAPOOL option is in the base code for CICS TS for VSE/ESA**

CICS performance considerations - storage

- **What to do?**
  - Increase ERDSASZE/EDSALIM for CICS TS VSE
  - Put Language Environment modules in SVA
  - Use storage tuning to reduce GETMAIN/FREEMAIN
    - Remember the 16 byte storage buffer when setting values: specified-value = desired-value - 16
    - Example: If you want 4K, specify 4080
  - Do not change RESERVE stack size to anything other than 0K
    - This almost implies that the RESERVE stack should be 8K in order for WSCLEAR to work.
    - The RESERVE stack is not needed under CICS. If an amount is coded, we end up spending an extra getmain BELOW for that size for every rununit.

CICS performance considerations - storage ...

- **What to do?**
  - use ALL31(ON)
  - Here are tables of storage usage for COBOL with Language Environment under CICS:

ALL31(OFF)	TRAN(*)	TRAN below	Enclave above	Enclave below
LE 1.5	13524	1528	20232	20400
LE 1.7	13564	1568	20992	21160

ALL31(ON)	TRAN(*)	TRAN below	Enclave above	Enclave below
above	Enclave	0	28424	0
below	13564	0	29184	0

Note: These numbers are based on MVS but can apply to VSE/ESA.