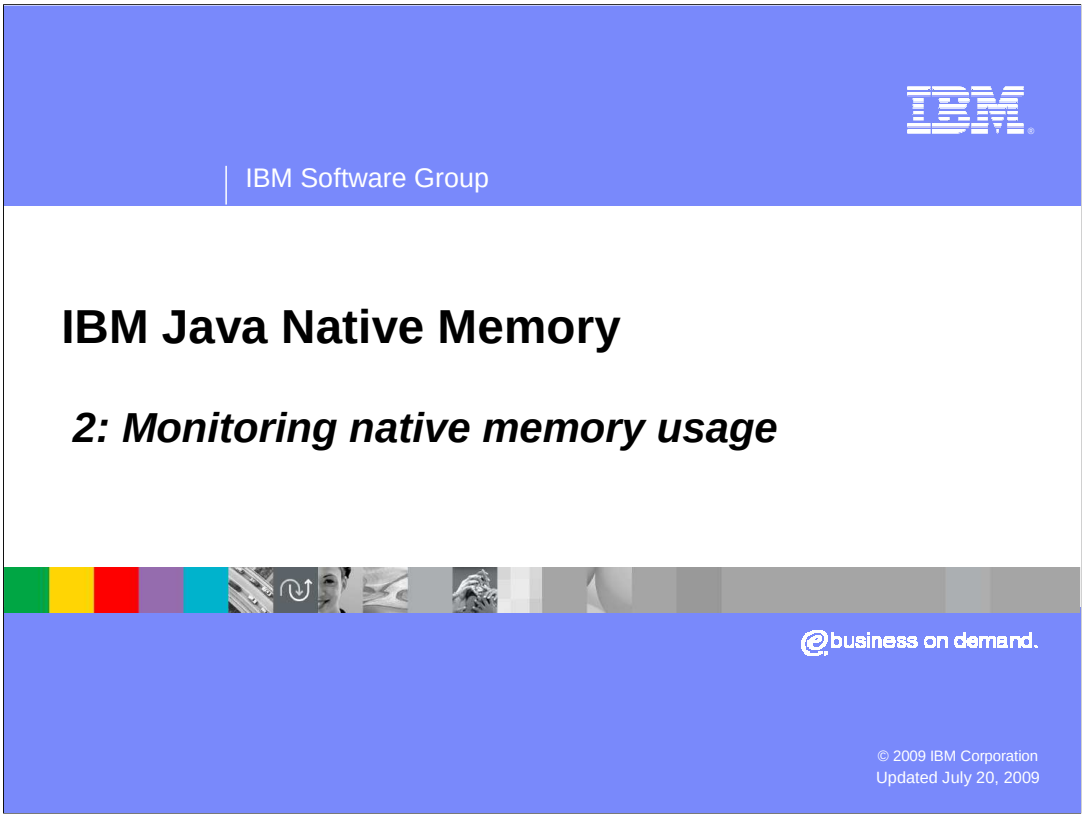
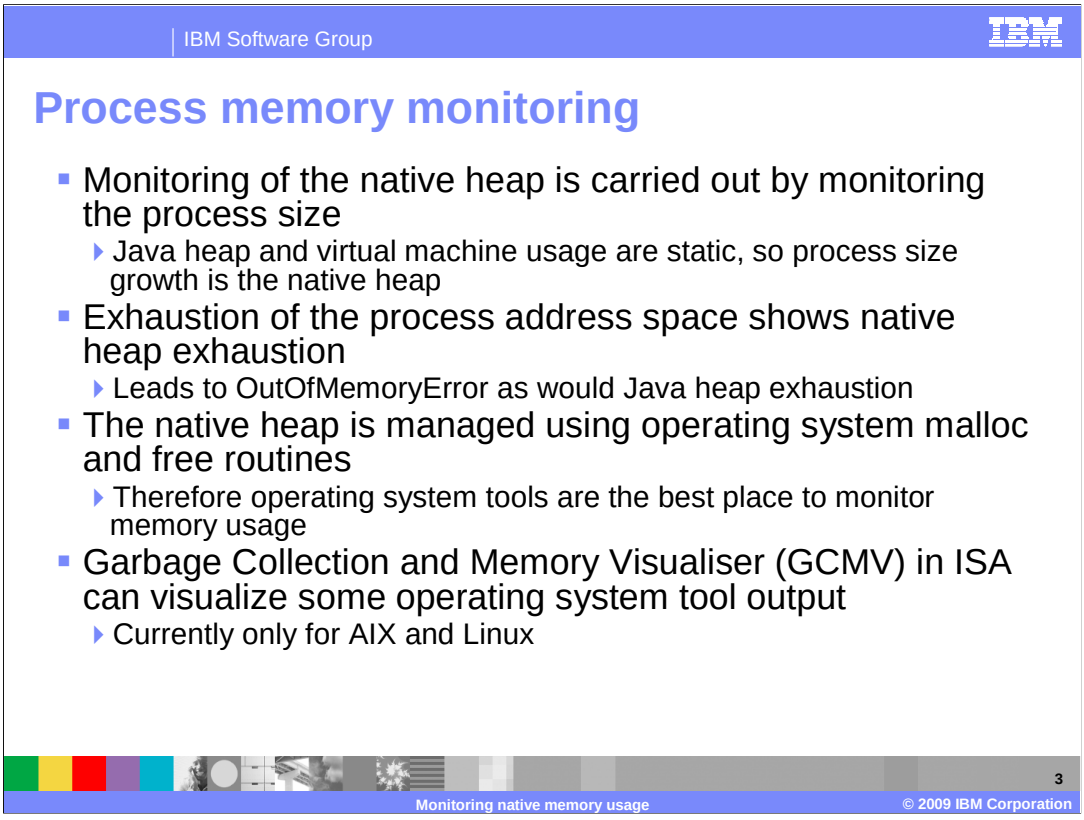


This is the second of three presentations on troubleshooting Java™ memory problems.

Process memory monitoring

- Examples of how to monitor memory usage on Windows®, AIX®, and Linux®
- Detecting heap exhaustion

The last presentation covered the introduction to debugging OutOfMemory errors. This presentation will explain how to monitor the memory usage to look for an Out Of Memory (OOM) exception in the native heap, with some examples on Windows, AIX, and Linux and detecting heap exhaustion.



Verbosegc will tell you if you have exceeded the Java heap, not the native heap, and you will most likely see a line of text saying “excessive gc encountered” to tell you why it’s happened.

Another option is to monitor the process memory. In the same way that you want to do verbosegc monitoring over time to help you to identify where memory leaks might occur, you can do process memory monitoring to spot leaks in the native heap before a failure occurs.

This is achieved by monitoring the entire process space. The Java heap is allocated at its maximum heap size when the process starts and this size is fixed. The VM is static in size, so, if the process address space grows, that shows growth in native heap.

The total memory used is the Java heap size plus the native heap size, so you can deduct the maximum Java heap to find the size of the native heap.

If the address space is exhausted, the native heap has been exhausted and you will see an OOM error.

The memory in the native heap is allocated using malloc and released with free, so operating system-based tools are best to monitor the memory usage. Tools vary between platform, unlike verbosegc.

GCMV is one of the tools available that can now visualize some of the OS tool outputs, which will be discussed later.

Process memory monitoring: Windows

- Recommended tool is “Perfmon”:
  - In Control Panel -> Admin Tools -> Performance
  - Can also be started using perfmon on the command line
- Displays several counters for a given process:
  - Relevant counter is “Virtual Bytes”
    - Memory that has been allocated; for instance, a malloc() request has been made
  - “Working Set” might also be on interest
    - Memory that is committed to; for instance, has been written to and is actively in use
  - Note: Maximum Java heap size is allocated at startup
    - But only the minimum heap size is written to (committed)

On Windows, the tool to use is called Perfmon. The other option is task manager but task manager is less appropriate for this.

You can start Perfmon using the command line or through the control panel.

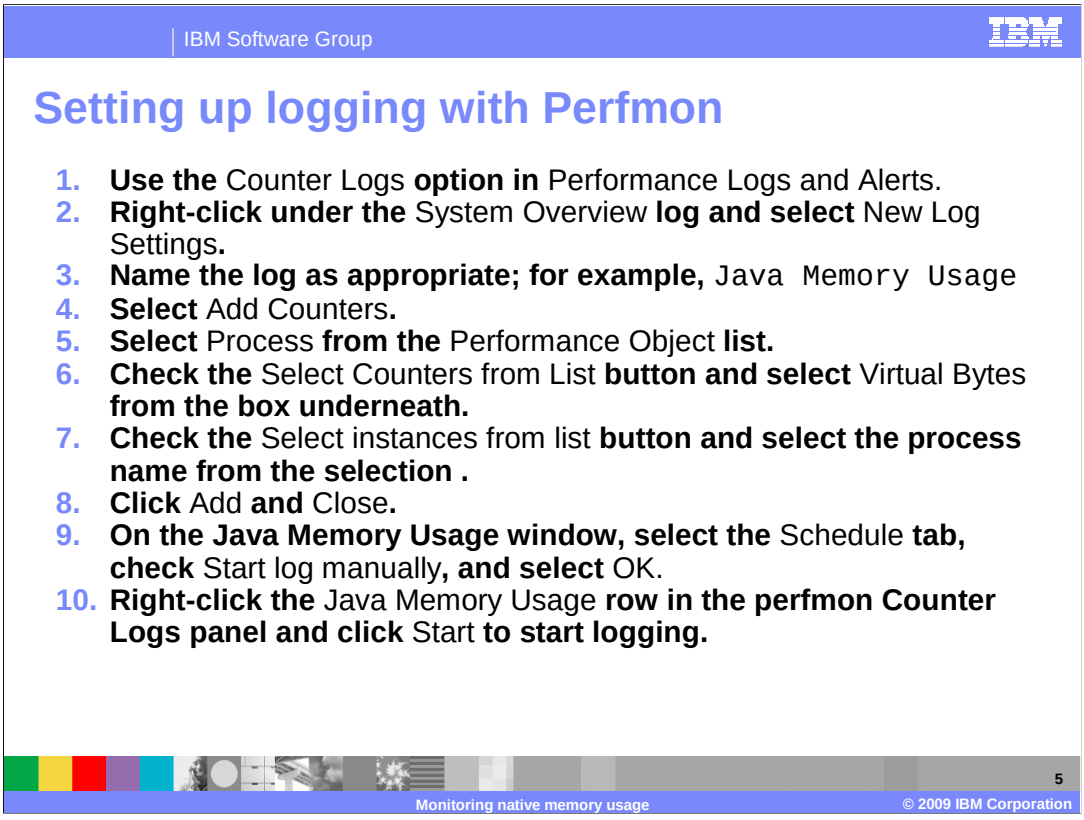
Perfmon gives you several counters, including the one called “virtual bytes”, which tells you about memory that has been allocated using malloc.

The “Working Set” counter is what you see in task manager, and this isn’t memory that’s been allocated but memory that’s been committed. The difference is whether you’ve written to it or not. Committed memory is memory that you’ve written data to.

For example, you can malloc 256 MB, and, if you write only to the bottom 5 MB at that time, then the Working Set will show 5 MB of memory usage, but you are actually using 256 MB of the memory.

Another example is if you use a test case to set the Java heap to 512 MB, but let the Java heap size fluctuate because the minimum and maximum are different. If you allocate to the heap and then delete, the Working Set value goes up and down. The Working Set shows this because a memory leak; which it isn’t, as memory was already allocated and you were using different amounts of that allocation at any time.

Virtual Bytes is the value to monitor because it is more accurate.



These instructions show how to set up logging with Perfmon.

If you use Perfmon you see a graph that moves across the screen. The problem is that you see only about 3 minutes of history, which is not good for long-term monitoring, but you can set up Perfmon to log to files, and the instructions here tell you how to do so for virtual byte counter.

Stopping logging and displaying the output

1. **Right-click the** Java Memory Usage **entry in the Counter Logs window and click** Stop**.**
2. **The name of the binary log file can be found in the log properties. (Double-click the** Java Memory Usage **line to view these).**
3. **Locate the physical log file from the file system.**
4. **Select the icon for** View Log Data **(persistent storage symbol).**
5. **Select** Log files **and** Add **the log file in the System Monitor Properties dialog box.**
6. **Select the** Data **tab, and click** Add **to select the data points in the log file.**
7. **In the** System Monitor Properties **dialog on the** Data **tab, click the** Scale **pull-down menu to select the correct scale (0.000001 for Virtual Bytes).**

The first three points explain how to stop the logging when you want to, and then how to load the log file back into Perfmon to use the graphing capabilities to look at history over time. The graph will show the full history.

Perfmon log

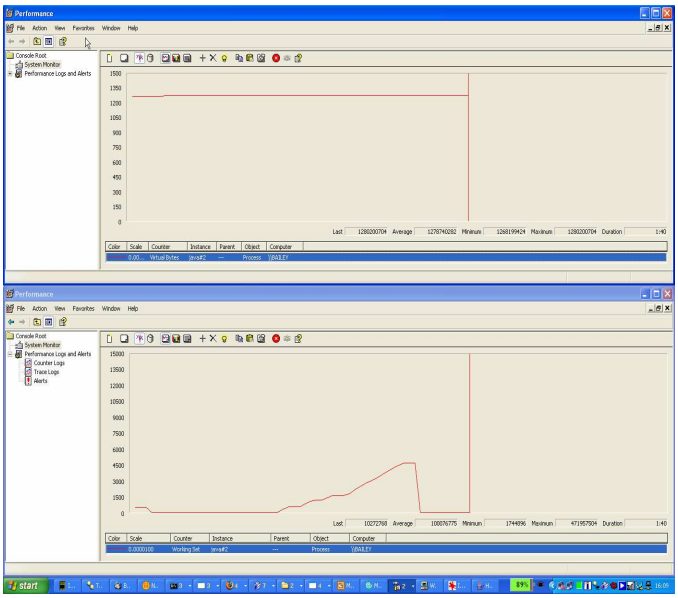
- Perfmon can log to text (.csv) file or binary
- CSV file format is as follows:

```
"(PDH-CSV 4.0) (GMT Daylight Time)(-60)","\\MY_COMP\Process(java)\Virtual Bytes"
"05/08/2008 16:33:56.859","1198592000"
"05/08/2008 16:34:11.859","1198592000"
"05/08/2008 16:34:26.859","1198592000"
"05/08/2008 16:34:41.859","1198592000"
"05/08/2008 16:34:56.859","1198592000"
```

- Can be imported into other tools:
  - Spreadsheet
  - Database
  - And so on…

Perfmon can log to a text file. This output is the result if you do so for Virtual Bytes. The first column is the time stamp, the second, shown in red, is the value in bytes.

Process memory: Perfmon view



Here is the sample output for Perfmon. It shows the difference in the virtual bytes, on the top line, and the working set, on the bottom line, for the test case mentioned earlier about adding objects to the Java heap and then removing in a batch.

The working set, on the bottom, shows the line increasing as Java heap usage went up and then dropped as it went down. The virtual bytes, on the top line, remains constant and shows no change in process heap at this point, but the working set implied there was a memory leak.

Process memory monitoring: AIX

- **Recommended tool is “Svmon”:**
  - Available on the AIX install image
  - Started using `svmon -P {pid} -m -r -i {interval}`
- **Displays a per segment breakdown of memory:**
  - Relevant value is “Addr Range” for heap segments
    - Memory that has been allocated; for instance, a malloc() request has been made
  - Heap segments are 0x3 -> 0xC inclusive

The recommended tool for AIX, which is an equivalent to Perfmon, is Svmon. In the same way as the task manager is not recommended for Windows, ps is best avoided on AIX .

You can start svmon using “svmon –P {pid} –m –r –i {interval}”, and the output can be piped to a file.

Recall that for Windows, Virtual Bytes should be used instead of Working Set. On AIX, “Address Range” should be looked at and not “In Use”. “In Use” goes down as memory is paged out if you are using paging. Only look at the “Address Range” data.

The last line on this slide explains that svmon breaks memory usage down into 256 MB segments. Svmon tells you about usage in each segment. For more information, you should read into how AIX uses user space and memory management. For example, 0x0 to 0x2 is used by the kernel, program data, and so on. User space for the Java process starts at 0x3 and typically goes up to 0xC. With Svmon, you can look at individual bits of memory used by Java and check program usage, thus native heap usage.

Process memory: Svmon output

```
Pid Command          Inuse      Pin     Pgsp  Virtual   64-bit    Mthrd
  25084 AppS           78907     1570      182    67840        N        Y
   Vsid      Esid Type Description           Inuse   Pin Pgsp Virtual Addr Range
  2c7ea         3 work shmat/mmap            36678     0    0 36656   0..65513
  3c80e         4 work shmat/mmap             7956     0    0  7956   0..65515
  5cd36         5 work shmat/mmap             7946     0    0  7946   0..65517
  14e04         6 work shmat/mmap             7151     0    0  7151   0..65519
  7001c         d work shared library text    6781     0    0   736   0..65535
      0         0 work kernel seg             4218  1552  182  3602   0..22017 :
                                                                      65474..65535
  6cb5a         7 work shmat/mmap             2157     0    0  2157   0..65461
  48733         c work shmat/mmap             1244     0    0  1244   0..1243
   cac3         - pers /dev/hd2:176297        1159     0    -     -   0..1158
  54bb5         - pers /dev/hd2:176307         473     0    -     -   0..472
  78b9e         - pers /dev/hd2:176301         454     0    -     -   0..453
  58bb6         - pers /dev/hd2:176308         254     0    -     -   0..253
   cee2         - work                         246    17    0   246   0..49746
  4cbb3         - pers /dev/hd2:176305         226     0    -     -   0..225
  7881e         - pers /dev/e2axa702-1:2048    186     0    -     -   0..1856
  68f5b         - pers /dev/e2axa702-1:2048    185     0    -     -   0..1847
  28b8a         - pers /dev/hd2:176299         119     0    -     -   0..118
```


Here is an example. Address range is listed on the right, and Esid is the segment number in process address space. That's where you'll get values between 0 and F.

Look at the Address Range on the right for segment 3 on the top line in red. It contains 0 to 65513, so those pages are 4 KB pages, totaling around 255-256 MB per section. You can see that segments 3, 4, 5, and 6 are filled and segment 7 is nearly filled. If you total those, you get your native heap usage. If that grows over time, your native heap usage is growing over time.

Process memory monitoring: Linux

- Recommended data is from /proc file system:
  - cat /proc/{pid}/status > {logfile}
    - Produces a snapshot of memory usage
  - A script can be run to generate several snapshots over time
- Useful value is VmSize
  - Memory that has been allocated; for instance, a malloc() request has been made

Finally, the same for Linux. The recommended way of looking at process size is in /proc/ file system, where you must look at status file for pid. Within that, look at the VmSize value. Perfmon does automatic sampling of a file, as does svmon, but there is no way of doing this in Linux without writing a script. The VM size change over time shows growth in memory usage.

Process memory: proc/pid/status output

```
Name:   java
State:  T (stopped)
SleepAVG:       102%
Tgid:   15266
Pid:    15266
PPid:   15218
TracerPid:      0
Uid:    604     604     604     604
Gid:    123     123     123     123
FDSize: 64
Groups: 122 123 124 30009
VmSize:   599888 kB
VmLck:         0 kB
VmRSS:     26340 kB
VmData:   560636 kB
VmStk:       204 kB
VmExe:        56 kB
VmLib:      7504 kB
Threads:        11
SigPnd: 0000000000000000
ShdPnd: 0000000000000000
SigBlk: 0000000000000000
SigIgn: 0000000000001000
SigCgt: 20000001800044ff
CapInh: 0000000000000000
CapPrm: 0000000000000000
CapEff: 0000000000000000
```


Here is some sample output; note the VmSize highlighted here in red on the 12th line. In this, VmSize is just under 600 MB. If that's growing, you have a potential problem.

The sampling rate for the status file itself is generated when you read it, and is only accurate at the point at which you read it. You could use a script to regularly read it.

The recommended tool for z/OS® is a tool by Rocket Software with a utility called MXI, which tells you about LE subpool usage.

Analyzing native memory with GCMV

- **Garbage collection and memory visualizer (GCMV):**
  - GCMV is capable of graphing some operating system memory logs:
    - Currently Linux and AIX only
      - Windows version is under development, but you are able to graph the logs using Perfmon
    - GCMV provides scripts to capture the data in the help file
- **Visualization makes it easier to see trends over time:**
  - Look for memory leak
  - Look for native heap footprint issues

GCMV is capable of graphing some of OS logs (AIX and Linux only at the moment)

Windows is under development, but can graph Perfmon output using Perfmon so you still have a way to visualize it.

For AIX and Linux, GCMV provides a script to generate data and ensures that it is a format that the GCMV can parse.

Visualization is useful for seeing trends over time

The GCMV can also be run stand-alone

Is the native heap exhausted?

- Native heap exhaustion is likely if the memory usage is approaching the user space limit:

Operating system	Additional options?	User space
AIX	-Xmx < 2.3GB	2.75 GB
	2.3G <= -Xmx < 3GB	3 GB
	-Xmx >= 3GB	3.25 GB
Linux		3 GB
	Hugemem Kernel	4 GB
Windows		2 GB
	/3GB	3 GB

The reason for looking at this is to determine if the problem is a native or Java heap OutOfMemory error. While it is useful to monitor trace over time to see if leaks will happen, if you want to know if your error came from the native heap or not you have to guess by looking at whether usage of the process address space is close to the limits available to you.

This table shows these limits, depending on the Java heap size affects usable space on Linux.

Finally, if you know that the OutOfMemory error is in the native heap, you need to profile it to find the cause using profiling.

That concludes the second part of this presentation on monitoring the memory usage and determining heap exhaustion. The final part is about using profiling to find the cause of the leak.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_2-MonitoringNativeMemoryUsage.ppt

This module is also available in PDF format at: ../2-MonitoringNativeMemoryUsage.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

IBM, the IBM logo, ibm.com, and the following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

AIX z/OS

If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of other IBM trademarks is available on the Web at "Copyright and trademark information" at http://www.ibm.com/legal/copytrade.shtml

Windows, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java, and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.