

Improving Compute Farm Throughput in Electronic Design Automation (EDA) Solutions



Abstract

Functional verification of Silicon on Chip (SoC) designs can contribute to as much as 70% of design cycle times. To develop 45nm and 32nm architectures, the computational resources needed and verification times required increase exponentially over earlier levels, with verification errors leading to production “respins”, costly both in dollar terms and time to market considerations. Increasing the degree of utilization in multi-core compute farms can significantly improve application throughput and speed new product introductions. IBM and eXludus recently demonstrated that MultiCore Optimizer (MCOpt) middleware can significantly improve the performance and usability of compute farms running Cadence’s Incisive Enterprise Simulator by increasing CPU utilization rates, reducing I/O wait times, and eliminating memory paging. These effects lead to shorter run times for small-memory jobs, as well as the concurrent execution and completion of large-memory jobs that otherwise would not complete. MCOpt middleware also removes the need for manual system tuning and/or user prediction of resource requirements.

System Throughput in the EDA Design Flow

One of the biggest challenges in Integrated Circuit (IC) and System on a Chip (SoC) design is around verification that designs work correctly. Coupled with this, transistor densities are increasing significantly of late. Designs are becoming much more complicated, and there is an increasing trend toward moving embedded software interactions into the hardware. This has led to an increase in the use of “coverage verification”. Coverage verification employs software simulations of hardware along with large quantities and random sets of tests to verify a given chip design. Due to the huge amount of processing time required to run such software simulations, it becomes impossible to verify an entire design completely during the cycle times typically allocated. As a result, during a logic design phase, critical functions of the design are chosen to be tested or “covered”.

Time to market is a critical component to business success, and even short delays in the product release can dramatically reduce product profitability. Increasingly, commercial designs must be brought to market as products according to shorter and shorter time frames.

Modern SoC implementations have hundreds of millions of transistor logic gates in their designs. Functional verification for those can easily contribute to 70% of the entire design cycle time. As SoC designs head for the transistor density required for 45- and 32-nanometer (nm) granularities, the computational resources and time required for the associated coverage verification activities increase exponentially. Increasingly, SoC designs are utilizing embedded software, and the development costs associated with that more often than not exceed the design costs of the hardware it runs on.



Functional verification is critical to the SoC design cycle in order to reduce and eliminate problems found in first production runs of Silicon. Problems not caught during verification often necessitate costly “respins” of production processes.

At current 90 nm device densities, a new mask set can cost in excess of \$1.5 million, and can introduce months of delay to a design cycle due to the need for reverification.

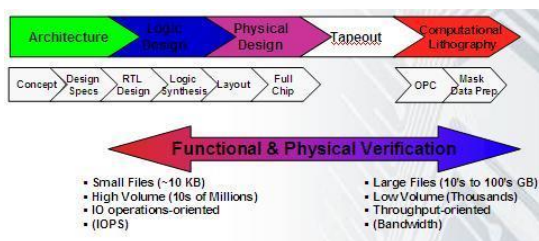


Figure 1: Verification in the EDA Design Cycle

Design verification is computationally intensive; a complete SoC verification can take many CPU Years to complete. Logic design verification is also characterized by the production of a very high volume, often in the tens of millions, of small files. The coupling of such computational requirements along with the business pressures of reduced time to market drives adoption of efficient verification processes to become a distinct competitive imperative. Historically, IC and SoC design teams have executed verifications as batch jobs on large “compute farms” using traditional job scheduler technology. As the sizes of logic designs have grown exponentially, so accordingly have the sizes of compute farms. It is not uncommon to find compute farms on the order of 10,000 processors spread across thousands of networked nodes.

Increased processor speeds and the introduction of multi-core processors have mitigated the bottleneck around verification to some extent, but the computational workhorse for design verification has been and continues to be the “strength in numbers”

approach of the compute farm. To contribute to reductions in design cycle times, a compute farm must provide a very high system throughput of jobs. However, high system throughput does not necessarily translate into reduced individual user response times.

Compute Farms and Job Scheduling

A compute farm is a pool of clustered servers providing high performance execution for CPU-, memory- and I/O-intensive jobs. Job scheduling mechanisms with memory considerations assume job memory demands are known in advance or are predictable based on user hints.

However, memory and other resource requirements may differ on a user or project level, and may change during the design cycle. So, analyzing resource requirements in advance of job execution can be a manually intensive activity subject to high degrees of error. For these reasons, while most widely used job scheduling mechanisms can accept resource hints, the functionality is often not used.

EDA workloads are typically data-intensive applications, a condition which causes memory resources in compute farms to be much more expensive relative to CPU resources. Jobs in a farm that have uneven memory allocations tend to cause page faults.

In addition, when a job triggers a page fault, the cache servicing the page I/O competes with the main memory footprint of the job itself, and indeed those of other jobs that are running on the farm. The overall performance of the farm and the response time of individual jobs are dependent on the effective use of the entire memory architecture of the system.

A small number of running jobs with abnormally high memory requirements can significantly increase the queuing delay times encountered by other jobs in the farm with nominal and normal memory requirements. As a result, there will be a slowdown in the execution of jobs as a whole, along with a corresponding decrease in the overall system throughput. Such a condition represents a classic “job blocking” issue, which



© Copyright IBM Corporation, 2008

occurs when a small number of large jobs block the execution of the majority of other jobs in a compute farm.

Taking single-threaded applications with large memory requirements and re-architecting them for multithreaded environments will not necessarily reduce their memory footprints. Due to problems associated with demand paging, applications are typically allocated enough memory so that their entire address spaces are resident.

Each thread in a parallel application typically communicates with its counterparts. To enhance throughput then, all threads should run simultaneously, but if the data associated with those threads is not in memory, then those suffer page faults. During page fault processing, affected threads become unavailable for communication, and therefore other threads in the application halt, awaiting synchronization events.

How eXludus MultiCore Optimization Assists

The eXludus MultiCore Optimizer technology (MCOpt) is middleware that dynamically optimizes system throughput to reduce the time needed to complete a given workload. MCOpt's mathematical algorithms provide real-time analysis of available system resources and processing tasks, and continuously creates work schedules that move the most jobs through the system in the least time possible. MCOpt increases core utilization rates, and eliminates the issues associated with excessive paging and blocking that can be introduced by large-memory jobs. MCOpt also prevents most job interference that may otherwise occur within a system and negatively impact results.

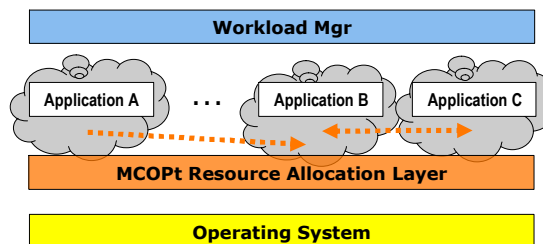


Figure 2: MCOpt Software Implementation

MCOpt requires no changes to existing applications, and works with job schedulers such as Platform Load Sharing Facility (LSF), Tivoli Workload Scheduler LoadLeveler, and the Portable Batch System (PBS). With MCOpt installed, users do not need to manually analyze job resource requirements and feed these requirements ('hints') to a job scheduler, as this work is automated within MCOpt.

Traditional job schedulers distribute jobs to individual nodes, controlling *where* and *when* jobs execute within a compute farm, but have no control over how jobs execute within those individual nodes. MCOpt controls *how* jobs execute within a node by controlling node-level resource allocations. Through MCOpt functionality, job scheduler dispatch latency – which can become lengthy on large compute farms – can be virtually eliminated. This translates into much lower idle CPU time.

Test Configuration

IBM and eXludus undertook a Proof of Concept of the MCOpt in an EDA logic design verification environment running Red Hat Enterprise Linux ES Version 4 with Update 7. The environment was composed of an IBM eServer xSeries grid running Cadence Design Systems' Incisive Enterprise Simulator, along with compute farm management handled by Tivoli Workload Scheduler LoadLeveler augmented by the eXludus MCOpt product.



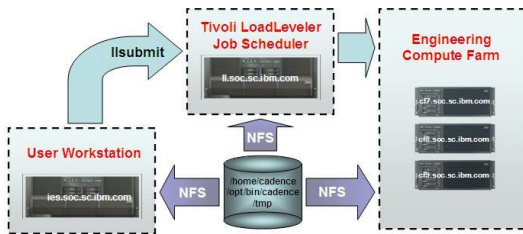


Figure 3: The eXtreme Design Automation Test Environment

Referring to Figure 3, the Engineering compute farm was comprised of a number of multi-core IBM eServer xSeries machines that executed a logic design batch workload submitted by users through the LoadLeveler scheduling machine. MCOPt was installed on each of the nodes in the compute farm. All file systems for each node in the compute farm plus the user workstation and the job scheduler machine were NFS cross-mounted. Job submission scripts run on the user workstation were altered slightly to invoke the MCOPt product once LoadLeveler dispatched a job to its target node in the compute farm.

The test workload was comprised of;

- 12 large-memory jobs (14GB resident memory), each having expected run times of approximately 780 seconds.
- 1350 and 1600 Cadence “ncsim” jobs, each of approximately 38MB in size with expected run times of between 15-25 seconds each, with I/O to the NFS mounted file system containing the design logic.

Jobs were queued to the compute farm with their execution on “hold” until all jobs had been submitted. The execution queues were then released, and the throughput and performance of the compute farm was subsequently monitored.

To determine the baseline the workload was first executed against the compute farm without the MCOPt technology enabled. The Tivoli Workload Scheduler was configured with a backfill scheduling algorithm.

Then, the test workload was executed with MCOPt enabled, and the compute farm performance was compared to and contrasted against the baseline result.

Baseline Test Results

Figure 4 shows the “top” output from one of the compute nodes in the farm during the baseline run. The node consisted of four AMD 8360SE 2.5GHz processors with 64GB of RAM overall.

```
xterm
top - 11:37:45 up 2 days, 21:23, 5 users, load average: 11.74, 3.24, 3.23
Tasks: 262 total, 8 running, 243 sleeping, 1 stopped, 0 zombie
Cpu(s): 4.1% us, 26.1% sy, 0.0% ni, 5.3% id, 63.0% wa, 0.0% hi, 1.5% si
Mem: 65909804k total, 65962760k used, 47044k free, 600k buffers
Swap: 67960824k total, 3114576k used, 64846248k free, 30876k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  CPU% MEM%   TIME+  COMMAND
10057 cadence  22   0 14.7g 7.7g 212 D 57.2 12.3  0:32.89 200895mem10_150
10060 cadence  20   0 14.7g 7.9g 212 D 50.3 12.6  0:32.05 220751mem4_1500
215  root      15   0   0     0   0 R 35.9  0.0  147:11.20 kswapd3
10056 cadence  19   0 14.7g 8.0g 212 D 33.9 12.7  0:33.53 133274mem3_1500
10059 cadence  18   0 14.7g 7.7g 212 R 21.3 12.3  0:34.65 150837mem9_1500
10053 cadence  19   0 14.7g 7.1g 212 D 21.0 11.4  0:32.60 153475mem12_150
10054 cadence  18   0 14.7g 8.1g 212 R 21.0 12.9  0:31.00 261272mem2_1500_
216  root      15   0   0     0   0 R 20.0  0.0  146:15.22 kswapd2
10055 cadence  19   0 14.7g 8.0g 212 D 19.7 12.8  0:32.30 53423mem1_1500_
218  root      15   0   0     0   0 R 15.1  0.0  147:55.00 kswapd0
217  root      15   0   0     0   0 R 14.9  0.0  147:47.57 kswapd1
10058 cadence  19   0 14.7g 7.8g 212 D 12.6 12.4  0:32.44 154245mem11_150
10290 cadence  18   0 7048 3592 1380 R 8.5  0.0  0:00.33 perl
25928 root      15   0 47240 1816 1264 S 6.9  0.0  72:00.33 X
10302 cadence  17   0 5772 2228 1248 D 4.9  0.0  0:00.19 perl
10214 root      18   0 9784 2536 2132 D 4.1  0.0  0:00.21 Load_starter
4634  root      16   0 6436 1084 624 S 2.8  0.0  0:56.53 top
9295  cadence  17   0 6436 1004 624 R 2.6  0.0  0:02.35 top
24950 root      15   0 71336 364 152 D 2.1  0.0  3:50.38 gccofd-2
```

Figure 4: Baseline test run at 11:37 showing swapping

It is immediately evident from the above that there were eight large-memory jobs running with only half the job resident in memory. There was intense paging activity, and the system exhibited a very high I/O wait time of 63%.

Additionally, there were no “ncsim” jobs running on the CPU queue, even though those jobs were available to run via the Tivoli Workload Scheduler queue.

```
xterm
top - 11:40:48 up 2 days, 21:26, 5 users, load average: 22.46, 12.17, 6.71
Tasks: 283 total, 23 running, 259 sleeping, 1 stopped, 0 zombie
Cpu(s): 8.0% us, 89.2% sy, 0.0% ni, 0.3% id, 1.6% wa, 0.0% hi, 0.3% si
Mem: 65909804k total, 6586664k used, 43164k free, 270k buffers
Swap: 67960824k total, 20626244k used, 47344580k free, 165748k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  CPU% MEM%   TIME+  COMMAND
10056 cadence  25   0 14.7g 8.1g 212 R 88.1 12.8  2:45.36 133274mem3_1500
216  root      25   0   0     0   0 R 97.5  0.0  148:28.27 kswapd2
215  root      25   0   0     0   0 R 95.2  0.0  149:40.34 kswapd5
10053 cadence  24   0 14.7g 7.2g 212 R 89.5 11.4  2:34.15 153475mem12_150
217  root      25   0   0     0   0 R 89.2  0.0  150:08.35 kswapd1
218  root      15   0   0     0   0 R 86.2  0.0  149:44.73 kswapd0
10060 cadence  24   0 14.7g 7.9g 212 R 85.9 12.0  2:32.46 220751mem4_1500
10054 cadence  24   0 14.7g 7.3g 212 R 84.2 11.5  2:35.17 150837mem9_1500
14143 cadence  25   0 59540 38m 7192 R 81.9 0.1  0:13.54 ncsim
10054 cadence  24   0 14.7g 7.9g 212 R 78.3 12.3  2:30.77 261272mem2_1500_
10057 cadence  24   0 14.7g 7.1g 212 R 72.3 11.2  2:30.68 200895mem10_150
10058 cadence  25   0 14.7g 7.6g 212 R 71.0 12.1  2:39.31 154245mem11_150
10055 cadence  24   0 14.7g 7.9g 212 R 69.3 12.6  2:32.77 53423mem1_1500_
14091 cadence  25   0 59416 38m 7192 R 61.9 0.1  0:16.51 ncsim
4634  root      15   0 6436 1084 624 R 55.0  0.0  1:15.31 top
9273  load1    16   0 24400 6128 4360 S 48.1  0.0  0:34.75 Load_starter
14477 cadence  25   0 5088 1680 988 R 31.2  0.0  0:00.34 lrun
25928 root      15   0 47240 1824 1276 R 24.2  0.0  72:57.84 X
14461 cadence  23   0 52812 1076 860 S 18.9  0.0  0:00.57 run.sh
```

Figure 5: Baseline test run at 11:40 showing ncsim



Figure 5 shows the same run a few minutes later. The large-memory jobs were still exhibiting large amounts of paging. However, logic verification jobs, as depicted by the “ncsim” processes in the rightmost column, were beginning to stream through the compute node and execute.

The test was scheduled to execute and complete over a 30 minute period. During that time, none of the 12 large-memory jobs completed – even though the expected execution time per job was less than 13 minutes as the paging activity mostly affected these jobs - and the average time for an “ncsim” job to complete was 12.1 seconds.

MCOPt Test Results

Upon completion of the baseline workload tests, the job submission scripts were altered to utilize the MCOPt functionality, and the workload was rerun. No other changes were made to the environment.

```
xterm
top - 18:59:48 up 2 days, 4:45, 4 users, load average: 10.58, 5.86, 2.44
Tasks: 308 total, 12 running, 295 sleeping, 1 stopped, 0 zombie
Cpu(s): 48.5% us, 3.9% sy, 0.0% ni, 47.2% id, 0.2% wa, 0.0% hi, 0.2% si
Mem: 65903804k total, 65742408k used, 157395k free, 23868k buffers
Swap: 6796024k total, 86300k used, 67874524k free, 3166620k cached

  PID USER      PR  NI  VIRT  RES  SHR  S  %CPU  %MEM    TIME+  COMMAND
 1809 cadence  25   0 14.7g 14g  616  R 100.2 23.3  3:55.82 67647mem4_1500_
18311 cadence  25   0 58672 38m 7356  R 100.2  0.1   0:05.40 ncsim
 1852 cadence  25   0 14.7g 14g  616  R 99.9 23.3  3:55.76 77064mem2_1500_
 1872 cadence  25   0 14.7g 14g  616  R 99.9 23.3  3:55.74 8303mem2_1500_1
 1884 cadence  25   0 14.7g 14g  616  R 99.9 23.3  3:55.53 203774mem1_1500
17160 cadence  25   0 58604 38m 7356  R 52.1  0.1   0:01.57 ncsim
17097 cadence  23   0 58296 37m 7356  R 31.5  0.1   0:00.95 ncsim
17109 cadence  24   0 58596 38m 7350  R 31.2  0.1   0:00.94 ncsim
17308 cadence  17   0 58352 37m 7348  R 23.2  0.1   0:00.70 ncsim
17314 cadence  18   0 58472 38m 7344  R 20.9  0.1   0:00.63 ncsim
 1376 loadl   16   0 23036 6788 4544  S  6.3  0.0   0:10.37 Load_startd
17379 cadence  17   0 7052 3660 1380  R  5.6  0.0   0:00.17 perl
17373 cadence  16   0 20216 3348 2144  S  2.0  0.0   0:00.06 ncsim
  352 cadence  16   0  6436 1328  852  R  0.7  0.0   0:14.12 top
 1283 root     16   0  6436 1322  856  S  0.7  0.0   0:15.33 top
17183 cadence  16   0 10200 4716 4012  S  0.7  0.0   0:00.02 Load_starter
26928 root     15   0 47240 1322 1288  S  0.7  0.0 59:02.45 X
  843 root     15   0  0 0 0  S  0.3  0.0 4:15.22 kjournald
16362 cadence  16   0 10200 4648 3952  S  0.3  0.0 0:00.02 Load_starter
17076 cadence  16   0 5148 1676 1268  S  0.3  0.0 0:00.01 perl
17150 cadence  16   0 5148 1676 1268  S  0.3  0.0 0:00.01 perl
17185 cadence  16   0 10200 4648 3952  S  0.3  0.0 0:00.01 Load_starter
```

Figure 6: MCOPt Run example

Figure 6 shows the workload running on one of the compute nodes several minutes after submission. Jobs were scheduled to run on the node using Tivoli Workload Scheduler LoadLeveler in exactly the same manner as in the baseline run. However, MCOPt provided intranode job scheduling based on each job’s memory requirements. To prevent excessive paging activity that would degrade system performance, MCOPt has started only

four of the large-memory jobs, and their entire data resident in memory. Therefore, the system exhibited no paging, and the multiple “ncsim” jobs were running without interference or disruption in the remaining memory and job slots. In addition, the system now exhibited little to no I/O wait time, and the balance of user to system time was much more favorable.

In sharp contrast to the baseline test during which no large memory job completed, the four large memory jobs on the node completed in about the expected 780 seconds, and then a *second round* of large memory jobs were executed. In this fashion, using the MCOPt functionality all of the jobs, both “ncsim” and large-memory jobs completed through the nodes of the farm in a 30 minute test. The average time for a single “ncsim” job to complete was 10.1 seconds.

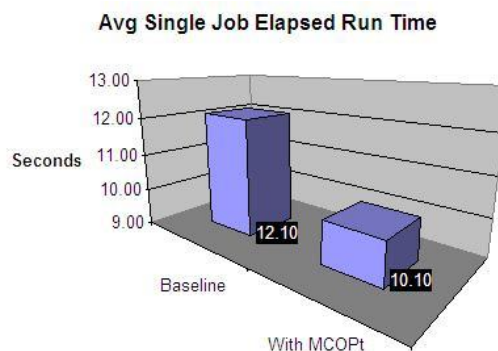


Figure 7: Comparison of “ncsim” job run time

As Figure 7 illustrates, the average job run time was reduced by 16.5%.

As previously indicated, the concurrently running large memory jobs were more heavily impacted by the detrimental performance effects of paging, and the impacts were dramatic. Due to testing time constraints, all job runs were terminated after 30 minutes. During the baseline runs, none of the large memory jobs completed, so definitive deltas – Baseline v. MCOPt – were not determined. However, it can be stated that the average large memory job run times “exceeded 30



minutes” during baseline testing v. an average of 13 minutes during the MCOPT enabled runs.

Avg Large Memory Elapse Run Time

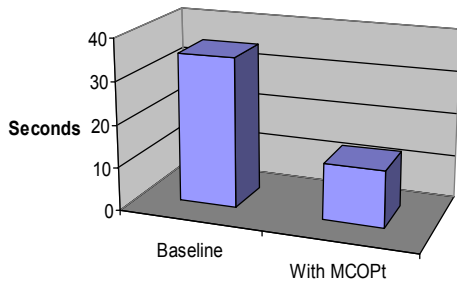


Figure 8: Comparison of large memory run time

As Figure 8 illustrates, the baseline average large memory job run time was greater than 100% longer than the MCOPT result.

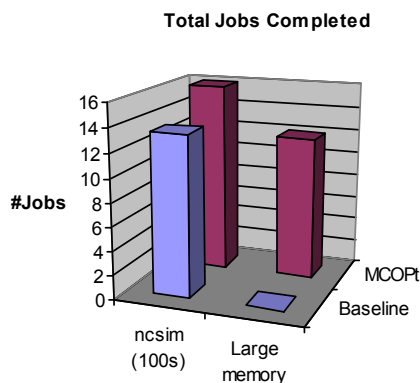


Figure 9: Completed Jobs Comparison

Figure 9 depicts the total number of jobs completed during the 30 minute test Baseline and MCOPT runs. MCOPT was significantly better at completing the mix of small and large memory jobs: the Cadence “ncsim” jobs ran faster with MCOPT and all large-memory jobs completed.

Conclusions

The paging impacts on the large-memory jobs (heavily impacted) and “ncsim” jobs (lightly impacted) can be explained through the Least Recently Used (LRU) paging algorithm used by Linux (and other UNIX flavors), which causes the page consumption rate (the rate at which an application references or modifies memory pages) to be constant for both small and large applications. However, large-memory applications have many more pages to process, and tend to have their pages less frequently accessed than those of smaller memory applications.

This results in the LRU paging algorithm selecting pages for removal from the large-memory jobs, where it is most likely to find the least recently used pages, to be paged out to disk. Afterwards, whenever the large-memory application refers to a removed page, it must block on an I/O request for a significant period until the required page is brought back into memory. This delay results in a further slowing-down of the rate at which the large memory application can access its pages, thereby enhancing their candidacy for the next cycle of page removal activity. The consequence of this is that large-memory applications incur a significant performance penalty when they share a processing node with smaller-memory applications in situations where paging is occurring.

This scenario primarily appears in due to a mixed workload of jobs that, when dispatched concurrently on the same node, contributes to interference between its members.

In large EDA compute farms, there are typically multiple projects and different design cycle workloads being run at any given time. These workloads often have very different resource requirements in terms of memory and I/O characteristics. Such types of job mixes are prone to the risks of resource contention and paging interference.

There are three possible solutions to this problem:

- Creation of a large job queue and the running of a single job at a time per node using that queue. This, however, potentially wastes node resources not



© Copyright IBM Corporation, 2008

consumed by a single large job. Moreover, if that large job exceeds the resource capacity of its node, paging will still occur.

- Deployment of a process on each node to monitor excessive paging, and to accordingly kill offending jobs. This does not solve the initial problem however, as a killed job would potentially exhibit its same behavior when restarted at a later date.
- Deployment of a node-level capacity management tool (such as MCOPT) that eliminates job interference and minimizes or prevents the occurrence of paging.

While speeding verification processing has important product time to market effects, optimizing job throughput across a compute farm brings difficulties:

- EDA applications themselves do not inherently utilize multi-core architectures efficiently.
- Dispatching mixed job workloads having very different resource requirements on compute farms often leads to performance-limiting job interference.
- Job scheduler dispatch latencies can lead to significant CPU idle time.
- Manually tuning jobs for specific server nodes via resource 'hints' is time-consuming and difficult.

As demonstrated by the test results documented herein, eXludus MCOPT middleware effectively solves the above problems by dynamically adapting to job requirements and devising real-time node resource allocations that optimize system throughput for any given workload.

Product data is subject to change without notice. This information could include technical inaccuracies or typographical errors. The performance data contained herein was obtained in a controlled, isolated environment. Actual results that may be obtained in other operating environments may vary significantly. While IBM has reviewed each item for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. 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. It is the user's responsibility to evaluate and verify the operation of any non-IBM product, program or service. All statements regarding IBM future direction and intent are subject to change or withdrawal without notice and represent goals and objectives only. 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 non-INFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. IBM is not responsible for the performance or interoperability of any non-IBM products discussed herein. 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. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both: AIX, AIX 5L, DS8000, eServer, iSeries, i5, Micro-Partitioning, OpenPower, POWER4 POWER5, pSeries, p5, Tivoli, TotalStorage

