Application-performance tuning and optimization on POWER6

Balaji V. Atyam
Calvin Sze
ISV Business Strategy and Enablement
February 2008

Table of contents

Abstract

This white paper provides information of new hardware features on IBM POWER6 processors and how operating systems and applications can take advantage of them. All existing IBM AIX or Linux application binaries are expected to work and perform on POWER6 processor-based systems without modifications. However, there are certain new features and significant architectural differences in POWER6 from its predecessor IBM POWER5 processor family. Many applications can perform significantly better with new binaries from the latest compilers and effective exploitation of new features, such as AltiVec, decimal floating point (DFP) and simultaneous multithreading (SMT).

Introduction

The IBM® POWER6™ processor is the next version of the family of processors that are based on the IBM PowerPC® architecture. POWER6, as with its predecessor IBM POWER5™ processor, includes the dual-core design (two processors on a single slice of silicon). The POWER6 processor maintains the same instruction pipe length. Each core has the ability to simultaneously run two tasks called *threads* — thereby, increasing the performance and throughput of the system. The ability to run two tasks at the same time is called *simultaneous multithreading (SMT)*. Based upon proven SMT and dual-core technology from POWER5, POWER6 extends the IBM leadership position by coupling high-frequency core design with a cache hierarchy and memory subsystem that are specifically tuned for ultra high-frequency multithreaded cores.

As with its predecessors, POWER6 is supported on both the IBM AIX® and Linux® operating systems. Through application programming interfaces (APIs) provided by IBM, independent software vendors (ISVs) and software developers can exploit the new hardware features in POWER6 to enhance the performance, reliability and usability of their applications.

The POWER6 processor can run as high as 4.7 GHz; it has 4 MB of L2 cache per core, an integrated memory controller and 32 MB of L3 cache. Scaling was a key design feature for this processor. The first set of POWER6 processor-based systems includes support for the next-generation PCI-Express I/O. The enhanced SMT implementation on POWER6 allows dispatching from both threads instead of one, and it has higher cache associativity, thus improving on the SMT performance delivered with the POWER5 processor.

POWER6 core

The POWER6 processor is manufactured by 65-nanometer silicon on insulator (SOI) technology with 10 levels of metal that provide a 30 percent performance improvement over the 90-nanometer SOI technology delivered with the POWER5 processor (when each is running at constant power). In addition, to achieve an optimum lower-power design, the POWER6 processor packs with tailored transistors for ultra-low power and tailored array cells for both high-performance and high-density applications. All array cells inside the processor use separate voltage supplies to enable low-voltage applications. The broad voltage range of the POWER6 processor further enhances its low-power and high-performance characteristics (HPC).

The POWER6 processor runs at approximately twice the frequency of the POWER5 (4-5 GHz) processor, and its instruction pipeline is similar to POWER5 in order to maximize its performance scalability. Consequently, as shown in Figure 1, many instruction sequences in POWER6 take half the time of POWER5. But, many other factors affect performance, for example, memory-in-cache latency is not reduced by the same ratio as cycle time ratio. Also, POWER5 is out-of-order execution, whereas POWER6 is primarily in-order execution.

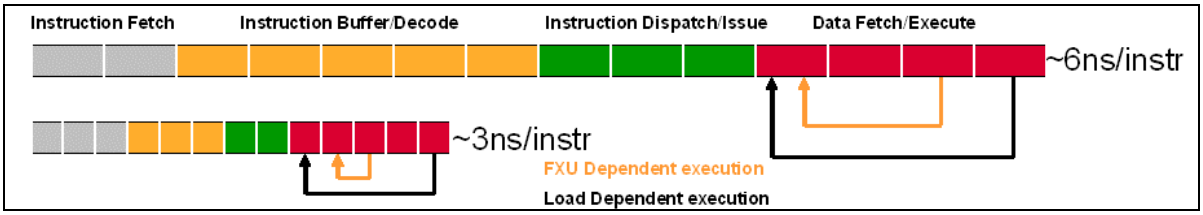

Figure 1. POWER6 and POWER5 instruction pipeline

In addition to the lower power consumption and higher frequency, POWER6 extends the functions of the POWER5 core. It includes a decimal-floating unit (DFU), a Motorola AltiVec (also called vector multimedia extensions or VMX) unit and a recovery unit. The enhanced two-way SMT is capable of dispatching seven instructions simultaneously from both threads; in contrast, POWER5 can dispatch a maximum of five instructions at one time by one thread.

As shown in Figure 2, the POWER6 processor has 4 MB private L2 cache per core and 32 MB nonsectored L3 cache per processor. There is support for L3 cache in three configurations: the module high-bandwidth configuration, the optional off-module configuration and the no-L3 option configuration.

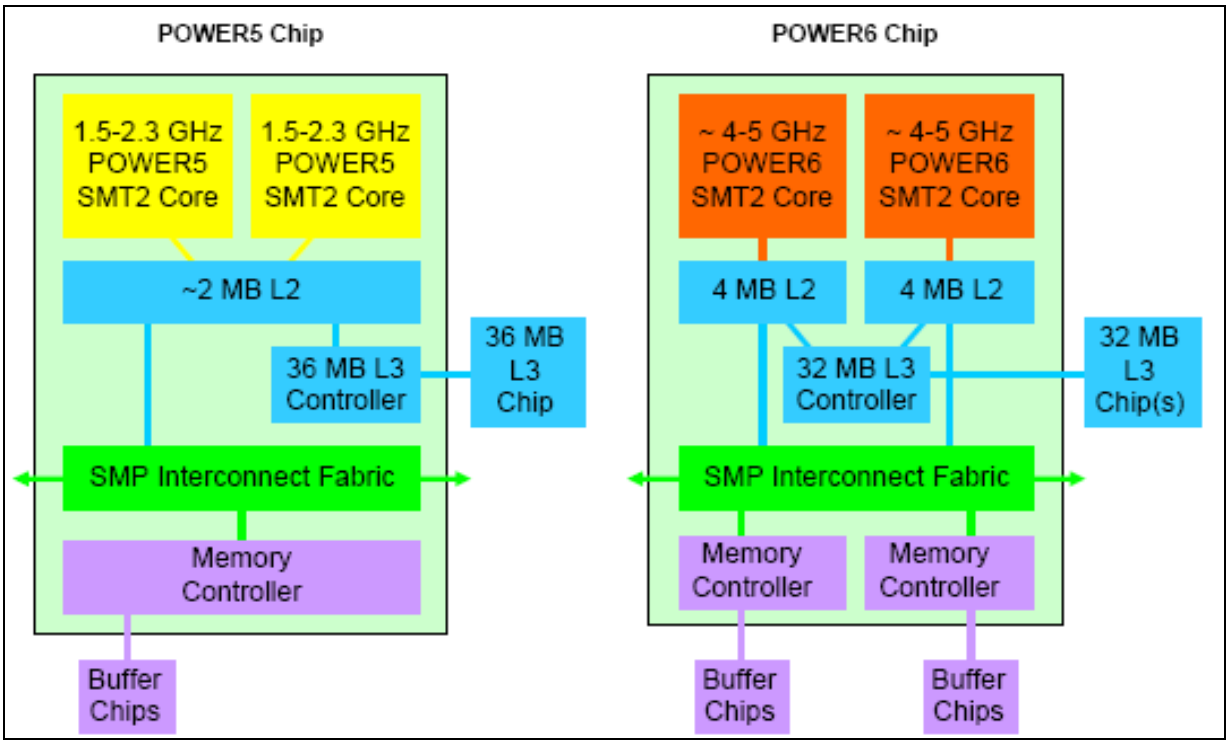

Figure 2. Evolution of the POWER6 processor structure

Getting the best performance on POWER6?

Applications that are built with the common PowerPC architecture and a specific POWER processor (such as POWER4 or POWER5) can run on the POWER6 platform without any modifications. However, the POWER6 architecture has several new features that differ from previous versions of the POWER chip. By using the latest IBM compilers and effectively exploiting SMT, it is often possible to obtain significant performance improvements.

FORTRAN and C/C++ compilers

IBM XL Fortran Enterprise Edition for AIX, V11.1 and IBM XL C/C++ Enterprise Edition for AIX, V9.0 compilers support the POWER6 specific tuning options, such as –qarch=pwr6 and –qtune=pwr6 or balanced. Some versions of gcc 4.1 and gcc 4.2 support POWER6 specific optimization using mcpu=power6.

General tuning options

The following options pertain to application tuning:

qtune=pwr6

- Instructs the compiler to schedule instructions for POWER6 optimization. You can use this with different –qarch options, but it is most commonly used with –qarch=pwr6.

qtune=balanced

- When used with -qarch=pwr5 (or pwr5x), this option generates a binary that runs on both POWER5 and POWER6 systems, but with scheduling improvements that should improve POWER6 performance.

qarch=pwr6

- Use POWER6-specific instructions. Compiling with –qarch=pwr6 –qtune=pwr6 will yield optimal performance on POWER6. **Note:** compiling with –qarch=pwr6 generates an executable that only runs on POWER6 or later processors.

qarch=pwr6e

- This generates optimizations than can take advantage of POWER6 *raw* mode instructions. The POWER6 raw mode includes instructions to move data directly between floating-point and general-purpose registers. Binaries built with this option cannot run under the default POWER6 mode. Furthermore, the binaries containing raw-mode instructions might not run on future POWER architectures.

qfloat=norngchk

- This new option produces faster software-divide and square-root sequences. It also eliminates control flow in the software div/sqrt sequence by not checking for some boundary cases in input values. By default, the optimization is used with –O3, unless –qstrict is also specified.

AltiVec

IBM initially offered the Motorola AltiVec execution unit on its PowerPC 970 processor and incorporated AltiVec into its POWER6 processor family. AltiVec is a single-instruction, multiple-data (SIMD) API.

SIMD packs *n*-like operations into a single instruction, resulting in a higher performing application. The applications that benefit most from AltiVec are those that operate on large arrays, doing calculations on the data. For example, the calculation C(n)=A(n)+B(n) benefits from AltiVec. Historically, AltiVec has been very appealing for high-performance, computational applications, as well as for multimedia and graphics-based applications. However, you can easily extend AltiVec to other high-computational application areas, such as cryptography, compression algorithms and signal processing.

AltiVec and autovectorization options

The following options pertain to using AltiVec and autovectorization:

qenablevmx (Fortran and C/C++)

- Use this option to compile C and FORTRAN source code that includes VMX/AltiVec intrinsics.

qaltivec (C/C++ only)

- Use this option to enable support of vector data types in C source code.

qvecnvol

- Use volatile vector registers.

qhot=simd

- Enable automatic recognition of loops that can be autovectorized.

qsclk=micro

- An additional option that you can try for autovectorizing FORTRAN source code.

qipa=malloc16

- This option asserts that arrays that are allocated off the heap with the malloc() method are aligned on 16-byte boundaries (suitable for compiler autovectorization).

qalias=noaryovrlp,nopteovrlp

- These options assert that the code meets specific conditions to allow autovectorization (that is, arrays are aligned correctly in memory, and arrays and pointers do not overlap in memory). Correct results are generated only if the assertions are valid.

Note: Additional compiler directives might be needed in the source code to support autovectorization.

Hardware decimal-floating point

Most computers today contain *binary floating-point* hardware. Though this satisfies some calculation requirements, binary floating-point logic cannot be used for financial, commercial and user-centric applications; nor is it useful for Web services. It is not possible to exactly represent the decimal data in these applications by using binary floating-point, which, many times, cannot precisely represent decimal fractions. For example, the value **0.1** needs an infinitely recurring binary fraction.

In contrast, a decimal-numbering system can represent **0.1** exactly. For these reasons, most real-world commercial data are held in a decimal form. Calculations on decimal data are carried out using decimal arithmetic, almost invariably operating on numbers held as an integer and a scaling power of **10**.

Software libraries exist today that process *decimal-floating point (DFP)* data to yield accurate results. However, because no standard for processing and storing data exists, many of these, proprietary software solutions that were developed in-house cause data inconsistencies when transferred between disparate systems. Performance is also relatively poor. The POWER6 processor contains a hardware-based DFP arithmetic unit to provide the best possible performance for decimal math computations.

The POWER6 processor provides approximately 50 new DFP instructions. It uses existing 32- and 64-bit floating-point registers. There are three IEEE-754r DFP data types, _Decimal32, _Decimal64 and _Decimal128. Each represents a decimal-float value. This white paper will discuss how to develop a DFP application to take advantage of thePOWER6 DFU.

Multiple page sizes

POWER6 provides 4 KB, 64 KB, 16 MB and 16 GB page sizes. Using a larger virtual-memory page size for an application's memory can significantly improve performance and throughput that results from hardware efficiencies that are associated with larger page sizes.

AIX supports all four page sizes, although 16 GB pages are only intended for very-high-performance environments — 64 KB pages are for general-purpose uses. Most workloads are likely to see a benefit by using the 64 KB pages rather than 4 KB pages. Table *1* shows the differences between each page size.

Memory region	**AIX application page size**			
	4 KB	**64 KB**	**16 MB**	**16 GB**
Text	Y	Y	Y	N
Stack	Y	Y	N	N
Data	Y	Y	Y	N
Shared	Y	Y	Y	Y

Table 1. The differences between each page size in AIX

You enable 64 KB pages by setting an environment variable and running the application. You can do this either in an application-startup script or as part of the AIX /etc/environment file. You can also enable 64 KB pages when linking an application. If an application needs to use 64 KB pages for its shared-memory regions, in addition to the environment-variable requirements, you need to make a few minor code changes. Most applications do not require any application changes to benefit from 64 KB pages.

For more information about multiple page sizes on AIX, visit the following Web site:
ibm.com/servers/aix/collaborationcenter

Linux supports only two page sizes, the base-page size and a large-page size. For all releases, the large-page size is 16 MB. Base-page sizes are either 4 KB or 64 KB. Again, 16 MB pages are for use in high-performance applications, and base-page sizes are for general-purpose uses. Linux supports large pages through the hugetlbfs mechanism. You can link the libhugetlbfs.a library with an application to use large pages independently for text and data. Table *2* shows the differences between each page size.

Memory region	Linux application page size		
	4 KB	64 KB	16 MB
Text	Y	Y	Y
Stack	Y	Y	N
Data	Y	Y	Y
Shared	Y	Y	Y

Table 2. The differences between each page size in Linux

Simultaneous multithreading

A very-high-frequency processor can spend more than half of its execution time waiting for cache and TLB misses. Given the trend for advances in processor cycle time and performance to increase faster-than-DRAM performance, it is expected that memory access delays will make up an increasing proportion of processor cycles per instruction. This is often referred to as the *memory wall*. One technique for tolerating memory latency that has been known for several years is multithreading. There are several forms of multithreading. A traditional form called fine-grained multithreading keeps *n* threads, or states, in the processor and interleaves the threads on a cycle-by-cycle basis. This eliminates all pipeline dependencies if *n* is large enough that instructions from the same thread are separated by a sufficient number of cycles in the execution pipelines. The form of multithreading that is implemented in the POWER5 architecture is called *simultaneous multithreading* (SMT) and is a hardware-design enhancement that enables two separate instruction streams (threads) to run simultaneously on the processor. It combines the multiple instruction-issue capabilities of superscalar processors with the latency-addressing capabilities of multithreading.

SMT for POWER5 has provided significant benefit in many commercial workloads, whereas the benefit of using SMT on POWER6 is much higher, even on HPC workloads. In fact, POWER6 SMT shows significant throughput improvements for most HPC loads or parallel applications. This is because the enhanced two-way SMT on POWER6 is capable of dispatching seven instructions simultaneously from both threads; in contrast, POWER5 can dispatch a maximum of five instructions by one thread in any given cycle. Thus if a thread cannot use a particular pipe, even though the other thread has an instruction for it, no instruction is dispatched to that pipe until the next cycle when the other thread can dispatch.

Tuning executables on POWER6 and POWER5

As discussed in the previous section, you can obtain significant performance improvements by creating executables with new compilers from POWER6 options. This white paper uses one of the life-sciences applications, NAMD, to show the application performance of POWER5 and POWER6 binaries on POWER5 and POWER6 systems. Two sets of compiler options are used to perform this study:

- `-O3 –qarch=pwr5 –qtune=pwr5`
- `-O3 –qarch=pwr6 –qtune=pwr6`

The first binary is used to benchmark on a 2.3 GHz IBM System p5™ 595 and a 4.7 GHz System p570. The second binary is used only on the System p570. Table 3 presents the benchmark. All runs are bound to four physical cores.

NAMD is a parallel molecular-dynamics set of code that is designed for the high-performance simulation of large, biomolecular systems. NAMD is designed to run efficiently on parallel systems for simulating large molecules. Based on Charm++ parallel objects, NAMD scales to hundreds of processors on high-end parallel platforms and tens of processors on commodity clusters by using a gigabit Ethernet. NAMD is a simulation engine that implements the molecular-dynamics algorithm, as applied to biomolecular systems with higher optimization and efficiency. NAMD V2.6 is used for the benchmark in this study, and this version is specifically optimized for IBM Power Architecture® technology.

An input case apoA1 is used for benchmarking. The description of this input – apoA1 (apolipoprotein A-I) is the primary protein constituent of HDL, defining its size and shape, solubilizing its lipid components, removing cholesterol from peripheral cells, activating the LCAT enzyme and delivering the resulting cholesterol esters to the liver. It consists of 92 224 atoms, 12A cutoff + PME every four steps, periodically. All benchmark numbers are measured in elapsed seconds (see Table 3). A lower number indicates better performance.

	Physical cores / computation threads			
Platforms	1 / 1	2 / 2	4 / 4	4 / 8
IBM p5 595 2.3 GHz POWER5+™ (with POWER5 binaries)	982.48	490.00	249.67	190.60
IBM System p570 4.7 GHz POWER6 (with POWER5 binaries)	935.25	461.23	232.33	144.11
IBM System p570 4.7 GHz POWER6 (with POWER6 binaries)	655.64	318.48	161.88	111.89

Table 3. NAMD V2.6 apoA1 performance benchmark numbers

From these benchmark numbers, it is apparent that POWER6 tuned binaries outperform POWER5 tuned binaries on a POWER6 system for this application. You can notice that about 40 percent performance is achieved with recompiled binaries for POWER6. From experience with many other applications, it is recommended that you use two binaries, one for POWER5 and another for POWER6, if feasible.

Exploiting AltiVec performance on POWER6

As mentioned, AltiVec is a single-instruction, multiple-data (SIMD) API. AltiVec implementation can provide significant performance improvement for some applications with the correct features that are computationally intensive and that have data precision that is lower than double. It is fairly challenging to incorporate SIMD methods in an application. Fortunately, the IBM XL C/C++ Enterprise Edition, V9.0 and IBM XL FORTRAN Enterprise Edition, V11.0 compilers provide automated SIMD capabilities to assist in this effort. The XLC/C++ compiler supports the AltiVec programming model and API on AltiVec capable systems. Both the IBM XL C/C++ and FORTRAN compilers can take advantage of AltiVec instructions and automatic SIMD vectorization to improve program performance in high-bandwidth data-processing and algorithmic-intensive applications. In addition to the AltiVec API support, the compilers also provide automated SIMD capabilities for application code by using the -qenable-vmx –qaltivec compiler options during compilation.

For more information on AltiVec programming, visit the following Web sites:

- en.wikipedia.org/wiki/Altivec,
- **ibm.com**/redbooks/abstracts/redp3890.html?Open
- freescale.com/files/32bit/doc/fact_sheet/ALTIVECFACT.pdf
- www-304.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/VSHA-6WPMEB

As noted in the previous sections, several options are needed to build a hand-coded VMX/AltiVec application. See the following recommended command lines:

- For C:
  ```
  xlc –qarch=pwr6 –qtune=pwr6 –qenablevmx –qaltivec
  ```
- For Fortran:
  ```
  xlf –qarch=pwr6 –qtune=pwr6 –qnablevmx
  ```

To automatically vectorize code, there are additional recommended options (as long as the appropriate alignment and pointer conditions are met):

- For C:
  ```
  xlc –qarch=pwr6 –qtune=pwr6 –qhot=simd –qnablevmx –qaltivec –
  qipa=malloc16
      -qalias=noaryovrlp,nopteovrlp
  ```
- For Fortran:
  ```
  xlf –qarch=pwr6 –qtune=pwr6 qhot=simd –qnablevmx  -
  qalias=noaryovrlp,nopteovrlp
  ```

To show the performance improvements that result from AltiVec, this study considers a life-sciences application, HMMER, which is an implementation of profile-hidden Markov models (profile HMMs). It compiles the results of multiple alignments and generates a statistical description of a sequence family's consensus. Among the programs in the HMMER package, hmmsearch contains the most computationally intensive functions. hmmsearch uses a profile HMM as the query to search against a sequence database for additional homologues of a modeled family. The standard distribution of HMMER has both AltiVec and standard (non-AltiVec) implementations and can be downloaded from http://hmmer.janelia.org/.

HMMER 2.3.2-altivec-gen2-mod is a high-performance version of HMMER in which the AltiVec implementation is much more efficient than the one from the standard distribution. Erik Lindahl at the Computational Structural Biology Stockholm Bioinformatics Center has implemented the AltiVec code in both the standard distribution and in the high-performance version of HMMER. HMMER 2.3.2-altivec-gen2-mod also contains additional optimization code from IBM. This high-performance version of HMMER can be downloaded from http://powerdev.osuosl.org/project/hmmerAltivecGen2mod.

Table 4 shows benchmark results from altivec-g2-mod and standard (non-ALTIVEC).

	Physical cores / computation threads			
Platforms	1 / 1	2 / 2	4 / 4	4 / 8
IBM p5-550 2.1 GHz POWER5+ standard	10305.32	5237.2	2592.34	2095.81
IBM BladeCenter JS22 Express 4.0 GHz POWER6 standard	5643.45	2849.99	1472.8	873.58
IBM BladeCenter JS22 Express 4.0 GHz POWER6 AltiVec-g2-mod	2388.05	1217.96	606.98	424.99

Table 4.HMMER V2.3.2 standard benchmark – HMMSEARCH – large test case

All benchmark numbers are measured in elapsed seconds. A lower number indicates better performance. IBM BladeCenter® JS22 Express outperformed the System p5 550 model for both the standard and AltiVec version of HMMER. The benchmark numbers shown in Table 4 clearly indicate that the performance of the HMMER application on POWER6 systems benefits from AltiVec exploitation by 100 to 150.

SMT performance on POWER5 and POWER6

Based on proven SMT and dual-core technology from POWER5, POWER6 extends the IBM performance-leadership position by coupling a high-frequency core design with a cache hierarchy and memory subsystem that are specifically tuned for ultra high-frequency multithreaded cores. The enhanced SMT implementation on POWER6 allows dispatching from both threads instead of one, and it has higher cache associativity, thus improving on the SMT performance delivered with the POWER5 processor. SMT for POWER5 provides significant benefit in many commercial workloads, whereas the benefit of using SMT on POWER6 is much higher — even on HPC workloads. In fact, POWER6 SMT shows significant throughput improvements for most HPC loads or parallel applications. As a rule, the SMT performance increase on POWER6 is twice that of using SMT on POWER5. Most HPC applications, notice a 20 percent or better performance improvement.

To show the SMT performance benefits on POWER6, this white paper considers three HPC applications in this study: Gaussian, NAMD and HMMER.

Gaussian

Researchers at Carnegie Mellon University initially developed the Gaussian program in the late 1960s. Today, the Gaussian program is the technology and market leader in electronic-structure modeling applications for chemical research. Gaussian can predict the structure of molecules, as well as the

chemical properties of complex reactions, under a variety of conditions — information that is often difficult or impossible to observe experimentally. Gaussian is used worldwide to perform chemical modeling for theoretical and practical research in a variety of industries, including pharmaceuticals, chemicals, material sciences, automotive, universities and government-research laboratories.

This white paper considers two hardware platforms: a 2.2 GHz System p5 570 POWER5 processor-based system and a 4.7 GHz System p570 POWER6 processor-based system, each with four cores for benchmarking. One of the standard Gaussian input files, test397 (RB3LYP/3-21g FORCE, C54H90N6O18), is used for benchmarking. Table 5 shows the performance numbers. All benchmark numbers are measured in elapsed seconds. A lower number indicates better performance. Both the number of physical cores and the compute threads that run in a given job are shown in the table.

Platforms	**Physical cores / computational threads**			
	1 / 1	**2 / 2**	**4 / 4**	**4 / 8**
IBM System p5 570 2.2 GHz POWER5+	4943	2514	1327	1148
IBM System p570 4.7 GHz POWER6	4098	2116	1095	711

Table 5. Gaussian benchmark numbers – test397

Only four physical cores of the p570 are used to achieve these results. four computational threads on four physical cores means that the benchmark is performed in single-thread mode. Eight computational threads on four physical cores means that the benchmark is performed in SMT mode.

SMT offers good performance gains for this benchmark input. The run time decreases from 1095 seconds (four threads on four physical cores) to 711 seconds (eight threads on four physical cores in SMT mode) on POWER6, whereas the run time on POWER5 decreases from 1327 seconds to 1148 seconds.

You can notice that SMT performance is much better on POWER6 compared to POWER5. The run time decreases by 54 percent on POWER6 and by 16 percent on POWER5 when using SMT to run eight threaded task on four physical cores.

NAMD

The application description for this HPC application is given in the Tuning executables on POWER6 and POWER5 section, and benchmark numbers are given in Table 3. From these performance numbers, it is apparent that SMT performance is better on POWER6 than on POWER5. The run time decreases by 31 percent on the POWER5 processor-based system, whereas there is a 61 percent decrease in the run time on POWER6 when using SMT to run eight threaded task on four physical cores.

HMMER

The application description for this application is given in the Exploiting AltiVec performance on POWER6 section, and the performance numbers are presented in Table 4. While using SMT to run eight threaded tasks on four physical cores, the run time decreases by 24, 69 and 43 percent, respectively, on POWER5+, POWER6 and POWER6 with AltiVec exploitation.

Decimal floating-point unit on POWER6

The performance of existing software libraries for decimal arithmetic is very poor compared to hardware speed. In some applications, the cost of decimal calculations can exceed the costs of both input and output processing tasks, and can form as much as 90 percent of the workload.

Concerns about poor performance of software libraries for decimal-arithmetic computation are very common. Software implementation is between one hundred and a thousand times slower than a hardware implementation. For example, a just-in-time (JIT)-compiled, 9-digit, BigDecimal division in Java™ 1.4 takes more than 13 000 clock cycles on an Intel® Pentium® processor-based system. Even a simple 9-digit decimal addition takes at least 1100 clock cycles. In contrast, a native-hardware decimal type can reduce the clock cycles so much that it attains a speed that is comparable to binary arithmetic.

The benefits of the hardware implementation are even more significant for multiplications and divisions at the higher precisions that are often used in decimal arithmetic; a 31-digit division in software can take up to 110 times as long as a 9-digit addition. Hence, with POWER6 hardware support, the performance of DFP operations can improve dramatically. Test applications that run atomic-decimal math operations show a speed-up that is 20 to 30 times faster with hardware DFP technology. However, the actual performance speedup for the user depends on the percentage of processing time that a given application spends in DFP computations.

For the C/C++ application developer, there are two options to take advantage of the POWER6 DFP unit.

- The IBM XL C/C++ Enterprise Edition for AIX, V9.0 compiler supports the DFP data types.
- A DFP library called *DFPAL* (DFlP abstraction layer) supports both hardware DFP instruction sets (on the DFP unit in POWER6) and software DFP operations through the DFP (decNumber) package.

The DFPAL API wraps around either the hardware instruction or the software (decNumber) module. There is additional function-call overhead with respect to compiler-native DFP. Early performance indicators show that it is slower than the native method, but despite a function-call overhead, DFPAL is still significantly faster than a software-based implementation of DFP operations.

If you want to use the decimal data type natively through the compiler, and upgrading the build environment is not a concern, then, for performance reasons, it is recommended that you use IBM XLC/C++ Enterprise Edition, Version 9.0 or migrate to Version 9.0 if you have an older environment. However, if switching the compiler, or application portability issues created by using a new data type that is not yet supported by other compilers are significant concerns, then DFPAL is a better option. Figure 3 shows you how to determine which compiler approach or DFPAL method to use when developing DFP applications.

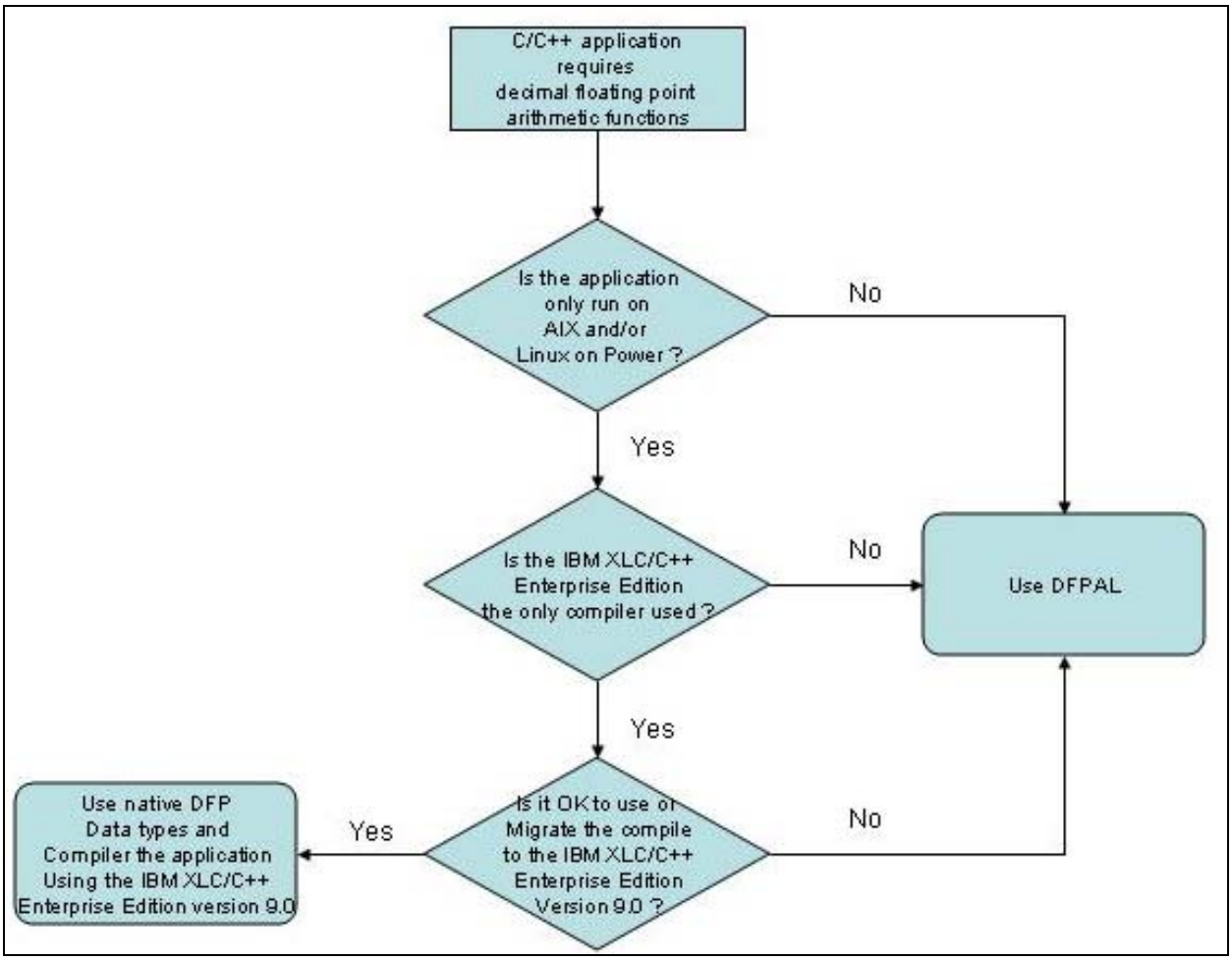

Figure 3. Decision flowchart how to include DFP in application

Using IBM C/C++ Enterprise Edition compiler

The IBM XL C/C++ Enterprise Edition, V9.0 compiler supports the new DFP data types. A POWER6 system is required to achieve hardware acceleration, but in the absence of POWER6 hardware, the compiler uses software libraries to deliver DFP arithmetic computations. The compiler flag –qdfp enables compiler support for DFP data types and literals.

The –qdfp or -qarch values do not support DFP instructions (for example, applications written for the IBM PowerPC, POWER3™, POWER4™ or POWE5 processor). Thus, the compiler automatically enables -qfloat=dfpemulate, and software emulation performs the DFP operations — even though the code runs on POWER6. List 1 shows an example of compiling a DFP application, dfpxlc_operations.c, on POWER6.

```
$ make
        xlc -qdfp -o dfpxlc_operations dfpxlc_operations.c
1506-1420 (W) The "-qfloat=dfpemulate" option is required for decimal floating-point
support on the target architecture. It has been set.
Target "all" is up to date.
$
```

List 1. Example of compiling a DFP application on POWER6

For DFP hardware acceleration, -qarch=pwr6 or –qarch=pwr6e, along with –qdfp, are required to generate the compiled code with hardware DFP instructions. However, the code and instructions only runs on POWER6 — it results in an illegal instruction with a core dump when run on POWER5 or earlier models.

The IBM XL C/C++ Enterprise Edition, V9.0 compiler supports three DFP types (_Decimal32, _Decimal64 and _Decimal128). List 2 shows a sample set of code that uses the new DFP data types

```
#include <stdio.h>
int main (int argc, char *argv[]) {
  _Decimal32 a32, b32, c32;
  _Decimal64 a64, b64, c64;
  _Decimal128 a128, b128, c128;
  a32=1;
  b32=-3;
  a64=1;
  b64=-3;
  a128=1;
  b128=-3;

  c64=(_Decimal64)b32;
  c32=(_Decimal32)__d64_abs(c64);
  printf("Decimal32  operation: Absolute value of %2.0Hf is %2.0Hf\n", b32, c32);
  printf("Decimal64  operation: Absolute value of %2.0Df is %2.0Df\n", b64,
__d64_abs(b64));
  printf("Decimal128 operation: Absolute value of %2.0DDf is %2.0DDf\n", b128,
__d128_abs(b128));
  printf("\n\n");
  c32=a32/b32;
  printf("Decimal32  operation: (%2.0Hf)/(%2.0Hf)=%16.35Hf\n", a32, b32, c32);
  c64=a64/b64;
  printf("Decimal64  operation: (%2.0Df)/(%2.0Df)=%16.35Df\n", a64, b64, c64);
  c128=a128/b128;
  printf("Decimal128 operation: (%2.0DDf)/(%2.0DDf)=%16.35DDf\n", a128, b128, c128);
  printf("\n\n");
}
```

List 2. DFP sample code using IBM C/C++ Enterprise Edition compiler

In List 2, the _Decimal32 variable b32 is first casted to the _Decimal64 variable c64.

Then, __d64_abs() is called to perform the absolute function on c64, because there are no DFP built-in functions for _Decimal32.

All _Decimal32 variables need to be casted to _Decimal64 or _Decimal128 data types before using corresponding __d64_ or __d128_ functions (for example, __d64_compare_signalling, __d128_compare_singaling).

Refer to the compiler reference manual that ships with IBM XL C/C++ Enterprise Edition for AIX, V9.0 for a complete list of DFP built-in functions.

Using DFP abstraction layer

DFPAL is a module that is written in the C programming language. DFPAL detects the availability of operating-system and hardware support for the DFP at run time. If such support is available, it automatically branches into the hardware path and uses hardware for the DFP arithmetic. On the other hand, if the operating system or hardware does not support DFP (as is the case on POWER5 or other processor architectures, for example), DFPAL branches into the software path and uses the decNumber module. The decNumber arithmetic module is a software implementation of the DFP arithmetic.

DFPAL and the required decNumber are available as a single package, for free, under the General Public License (GPL)-compatible, non-copyleft International Components for Unicode (ICU) license. The ICU license does not require applications that use DFPAL to be open source, as well. Therefore, you can retain your product's intellectual-property rights. For more information about ICU licenses, refer to the IBM ICU Web site at **ibm.com**/software/globalization/icu/license.jsp.

You compile DFPAL with a variety of compilers on multiple platforms (Linux on Intel x86, Linux on POWER, UNIX® and Microsoft® Windows® operating systems). You enable DFPAL portability through a combination of coding style and compile-time switches. Three sample build instructions (makefile) are included in the package. Follow the instructions in the makefile carefully, especially on the IBM AIX operating system. For example, you must use the -qlongdouble flag with the IBM XLC/C++ compiler on AIX. You can compile DFPAL as a standalone library or integrate it into the application itself.

Note: To download the DFPAL package, go to http://download.icu-project.org/files/DFPAL/. The compile instructions are available at www2.hursley.ibm.com/decimal/dfpal/compile.html.

After downloading and compiling DFPAL so that the DFAPAL library (libDFPAL.a) is available, you are ready to develop applications that use the DFP function, as provided by DFPAL. The remainder of this section shows how an application uses DFP operations on an AIX system through examples.

Similar to the IBM XLC/C++ Enterprise Edition compiler, DFPAL supports three DFP types: decimal32, decimal64 and decimal128. List 3 shows a sample set of code that uses these DFP data types.

```
#include <stdio.h>
#include <stdlib.h>

#include "dfpalct.h"
#include "dfpal.h"
#define BUFFER_SIZE 128

int main (int argc, char *argv[]) {
  decimal32 a32, b32, c32;
  decimal64 a64, b64, c64;
  decimal128 a128, b128, c128;
  char res0[BUFFER_SIZE];
  char res1[BUFFER_SIZE];
  char res2[BUFFER_SIZE];

  Int init_err, init_os_err;
  char *err_str=NULL;

  if (dfpalInit((void *)malloc(dfpalMemSize())) != DFPAL_ERR_NO_ERROR) {
    dfpalGetError(&init_err, &init_os_err, &err_str);
    fprintf(stderr, "DFPAL Init error :%d, error: %s\n",
      init_err, err_str);
    exit (1);
  }
  a64=dfpal_decimal64FromString("1");
  b64=dfpal_decimal64FromString("-3");

  a128=dfpal_decimal128FromString("1");
  b128=dfpal_decimal128FromString("-3");

  c64=decimal64Abs(b64);
  c128=decimal128Abs(b128);

  b32=decimal64ToDecimal32(b64);
  c32=decimal64ToDecimal32(c64);

  printf("Decimal32  operation: Absolute value of %s is %s\n",
                     dec64ToString(decimal64FromDecimal32(b32), res0),
                     dec64ToString(decimal64FromDecimal32(c32), res1));
  printf("Decimal64  operation: Absolute value of %s is %s\n",
                                dec64ToString(b64, res0),
                                dec64ToString(c64, res1));
  printf("Decimal128 operation: Absolute value of %s is %s\n",
                                dec128ToString(b128, res0),
                                dec128ToString(c128, res1));

  printf("\n\n");
  c64=decimal64Divide(a64,b64);
  a32=decimal64ToDecimal32(a64);
  b32=decimal64ToDecimal32(b64);
  c32=decimal64ToDecimal32(c64);

  printf("Decimal32  operation: ( %s)/(%s)=%s\n",
                     dec64ToString(decimal64FromDecimal32(a32), res0),
                     dec64ToString(decimal64FromDecimal32(b32), res1),
                     dec64ToString(decimal64FromDecimal32(c32), res2));
  printf("Decimal64  operation: ( %s)/(%s)=%s\n",
                                dec64ToString(a64, res0),
```

```
                                dec64ToString(b64, res1),
                                dec64ToString(c64, res2));
  c128=decimal128Divide(a128, b128);
  printf("Decimal128 operation: ( %s)/(%s)=%s\n",
                                dec128ToString(a128, res0),
                                dec128ToString(b128, res1),
                                dec128ToString(c128, res2));
  printf("\n\n");
}
```

List 3. DFP sample code using DFPAL

For each thread that uses the DFP data types and performs DFP operations, there is a requirement that the dfpal_Init function be performed immediately after the thread is created.

In List 3, no operations are specifically designed for decimal32 data types.

The Absolute operation is performed on a decimal64 variable b64.

Then, the result, c64, is converted to a decimal32 variable, c32, through the decimal64ToDecimal32() function.

Java applications on POWER6

This section describes suggested practices for, and the performance results of, using an IBM Software Development Kit (SDK) for Java on systems that are powered by the POWER6 processor. It discusses how to use large-page sizes. It also provides general Java coding and exploitation guidelines for the unique POWER6 hardware features, including DFP hardware.

Large pages

The operating system manages pages of the physical memory in a computer. The operating system allocates the pages to the Java programs that are running on the system. When the physical-memory pages are in short supply, the operating system writes some memory pages to a special area of the hard drive, called the *paging space*. Through a combination of the physical memory in the system and the paging space on the disk, the operating system supports a virtual memory that is larger than the physical memory and provides each program with a large virtual-memory space.

The original Power Architecture supports only a 4 KB page size. The POWER4 processor began to support a new 16 MB page size. The 16 MB pages require special setup in the operating system and have a number of restrictions —.they resided in a special reserved area of the physical memory, and that area was only available to tasks specifically requesting 16 MB pages and authorized to use those pages. The 16 MB pages were pinned in memory and could not be paged out to the disk-paging space.

The POWER5+ processor introduced support for additional page sizes, such as 64 KB pages. The new 64 KB pages function identically to the original 4 KB pages — they do not require any special setup, they are allocated from the same unreserved physical memory that is used for 4 KB pages, they are available to all tasks, and they can be paged out to the disk-paging space.

As mentioned, the POWER6 processor supports all of the page sizes supported by the POWER5+ processor. Java supports all of these page sizes through the -Xlp option. By default, Java uses only 4 KB pages. When the -Xlp or -Xlp16m option is specified, Java switches to 16 MB pages for the Java heap,

assuming the necessary operating-system setup and authorization is done. With the -Xlp64k option, Java uses 64 KB pages for the Java heap.

An application can significantly improve its performance by using large pages, because of the hardware efficiencies that are associated with larger page sizes. It is especially effective when applications allocate and use very large contiguous areas of virtual memory. Many Java applications fall into that category, notably if they allocate a large Java heap. The 16 MB and other very large pages are somewhat restrictive in their setup and usage, and are intended for very high-performance environments. In contrast, the medium-sized 64 KB pages are general-purpose and many workloads see a benefit by using them rather than the default 4 KB pages. The recommendation is that you always use the -Xlp64k Java option under normal circumstances, and performance will often improve by 6 to 10 percent or more. Where it is important to achieve the highest possible performance, you can perform the necessary operating-system setup and use the -Xlp16m Java option; this often improves performance by 8 to 16 percent, or more.

It is also possible to use the large pages for memory areas other than the Java heap. These other memory areas include the program text (machine instructions), data area (all other data outside of the Java heap) and stack.

Under the AIX operating system, it is possible to take advantage of large pages for the other data areas through the use of environment variables, such as in the following example:

```
export LDR_CNTRL="STACKPSIZE=64K"
```

Running Java after setting the stack size, as in this example, causes 64 KB pages to be used for the stack area. More details on changing the page sizes that programs use are available in the AIX and Linux on POWER documentation.

Java DFP application

The Java 6 BigDecimal class (based on, and identical to, the Java 5 BigDecimal class), is significantly different from its Java 1.4.x predecessor. The Java 6 class provides more constructors and a number of new methods (including: division, integral division, remainder and power). In addition, the MathContext object is added in Java 5. A MathContext instance specifies both the precision of a BigDecimal arithmetic result and a RoundingMode. Java 6 automatically exploits the POWER6 hardware DFP unit and runs BigDecimal functions in DFP hardware.

In this simple test that captures the essence of a telephone company's billing application, throughput is close to 16 percent faster when running on a POWER6 core with the DFU enabled, relative to an identical run on a POWER6 processor with DFU disabled. It also runs approximately 45 percent faster than on a comparable POWER5 processor.

Java applications with BigDecimal arithmetic operations (addition, subtraction, multiplication and division) can run 20 times faster on POWER6 with DFU enabled than with DFU disabled. For rounding operations, it can run four times faster on a DFU-enabled POWER6 system, according to IBM Microbenchmark testing.

In particular, Java does not have classes that correspond to _Decimal32, _Decimal64 and _Decimal128 data types, as there are in C/C++ (mentioned previously). However, you can accomplish arithmetic operations that are similar to these three encodings by specifying MathContext.DECIMAL32,

MathContext.DECIMAL64 and MathContext.DECIMAL128, respectively. The example in List 4 shows how to use BigDecimal to take advantage of the POWER6 DFU.

```
import java.math.*;
import java.text.*;
public class dfpOperations {
public static void main(String args[]) {
String sa, sb, sc;
BigDecimal a=new BigDecimal("1") ;
BigDecimal b=new BigDecimal("-3") ;
BigDecimal c=new BigDecimal("0") ;
DecimalFormat DF=new DecimalFormat("0.00000000000000000000000000000000000");
c=b.abs(MathContext.DECIMAL32);
sa=a.toString();
sb=b.toString();
sc=c.toString();
System.out.println("Decimal32 operation: Absolute value of "+sb+" is "+" "+sc);
c=b.abs(MathContext.DECIMAL64);
sc=c.toString();
System.out.println("Decimal64 operation: Absolute value of "+sb+" is "+" "+sc);
c=b.abs(MathContext.DECIMAL128);
sc=c.toString();
System.out.println("Decimal128 operation: Absolute value of "+sb+" is "+" "+sc);
System.out.println();
System.out.println();
c=a.divide(b, MathContext.DECIMAL32);
sc=DF.format(c);
System.out.println("Decimal32 operation: ( "+sa+")/("+sb+")="+sc);
c=a.divide(b, MathContext.DECIMAL64);
sc=DF.format(c);
System.out.println("Decimal64 operation: ( "+sa+")/("+sb+")="+sc);
c=a.divide(b, MathContext.DECIMAL128);
sc=DF.format(c);
System.out.println("Decimal128 operation: ( "+sa+")/("+sb+")="+sc);
}
}
```

List 4. Java sample code to use POWER6 DFP

Summary

This white paper has provided ways to get the best application performance on POWER6 by exploiting some of its new features. You have seen a demonstration of recompiling a specific application with appropriate POWER6 options that can improve performance significantly. Also, this white paper presents specific application-performance numbers with the exploitation of AltiVec feature in POWER6. The series of benchmarks discussed here clearly show that, for three different types of applications in life sciences, the performance benefits with SMT are much larger on POWER6, as compared to POWER5. For the applications with DFP operations, their performance can enjoy a significant boost on DFU-enabled POWER6 processor-based systems.

Resources

These Web sites provide useful references to supplement the information contained in this document:

- IBM System p™ Information Center
  http://publib.boulder.ibm.com/infocenter/pseries/index.jsp
- System p on IBM PartnerWorld®
  ibm.com/partnerworld/systems/p
- AIX on IBM PartnerWorld®
  ibm.com/partnerworld/aix
- IBM Publications Center
  www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US
- IBM Redbooks
  ibm.com/redbooks
  - *IBM eServer BladeCenter JS20 Programming Environment*
    www.redbooks.ibm.com/abstracts/redp3890.html?Open
- AIX Collaboration Center
  ibm.com/servers/aix/collaborationcenter
- IBM developerWorks®
  ibm.com/developerworks
- *Java Tuning for Performance on IBM POWER6* by Venkata Ravi Kumar Dadi and Levon Stepanian
  www-03.ibm.com/systems/p/software/whitepapers/java_tuning.html
- *Exploiting DFP arithmetic on POWER6 processor-based systems* by Calvin Sze and Punit Shah
  www-304.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/VPAA-78T95K
- *Understanding POWER6 - An overview for software developers and independent software vendors* by Calvin Sze and Balaji Atyam
  www-304.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/RVAL-73E6WL
- Optimizing ClustalW1.83 using AltiVec implementation by Yinhe Cheng
  www-304.ibm.com/jct09002c/partnerworld/wps/servlet/ContentHandler/VSHA-6WPMEB
- *AltiVec Technology*
  www.freescale.com/files/32bit/doc/fact_sheet/ALTIVECFACT.pdf
- AltiVec
  http://en.wikipedia.org/wiki/Altivec

About the authors

Balaji V Atyam is a senior technical consultant in the ISV Business Strategy and Enablement Organization at IBM Systems and Technology Group. His responsibilities include: porting, benchmarking, performance tuning, parallel programming and technical consulting services to key solution providers in the area of high-performance computing (HPC) on the IBM System p and System x™ platforms. You can contact Balaji at balaji@us.ibm.com.

Calvin Sze is an AIX and Linux consultant for the ISV Business Strategy and Enablement organization at IBM. He is based in Austin, Texas. Calvin's main role is to help solution developers bring their applications to the AIX and Linux operating systems, running on POWER processor-based technologies. Calvin has been involved in software development and system integration on both Linux and AIX platforms for more than 12 years. You can contact Calvin at calvins@us.ibm.com.

Acknowledgements

The authors want to thank Tom Edwards and Bruce Hurley for their encouragement and support. Also, we want to thank Ruzhu Chen, Yinhe Cheng and Carlos P. Sosa for application-benchmark numbers.

Trademarks and special notices

References in this document to IBM products or services do not imply that IBM intends to make them available in every country.

AIX, BladeCenter, developerWorks, IBM, the IBM logo, PartnerWorld, POWER, Power Architecture, POWER5, POWER5+, POWER6, POWER6+, PowerPC, Redbooks, System p, System p5 and System x are trademarks of International Business Machines Corporation in the United States, other countries, or both.

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

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

Intel, Intel Inside (logos), MMX, and Pentium are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a 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.

Information is provided "AS IS" without warranty of any kind.

Information concerning non-IBM products was obtained from a supplier of these products, published announcement material, or other publicly available sources and does not constitute an endorsement of such products by IBM. Sources for non-IBM list prices and performance numbers are taken from publicly available information, including vendor announcements and vendor worldwide homepages. IBM has not tested these products and cannot confirm the accuracy of performance, capability, or any other claims related to non-IBM products. Questions on the capability of non-IBM products should be addressed to the supplier of those products.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. 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.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.