Power Architecture® ISA 2.06 Stride N prefetch Engines to boost Application's performance

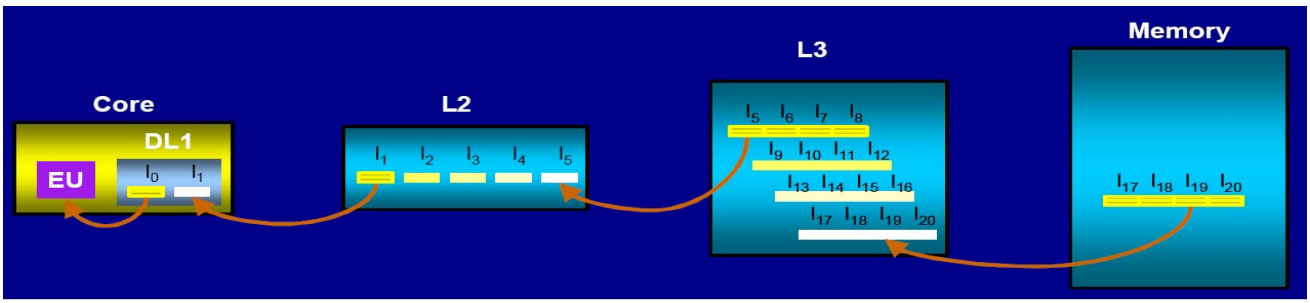
History of IBM POWER architecture:

POWER stands for Performance Optimization with Enhanced RISC. Power architecture is synonymous with performance. Introduced by IBM in 1991, POWER1 was a superscalar design that implemented register renaming andout-of-order execution. In Power2, additional FP unit and caches were added to boost performance. In 1996 IBM released successor of the POWER2 called P2SC (POWER2 Super chip), which is a single chip implementation of POWER2. P2SC is used to power the 30-node IBM Deep Blue supercomputer that beat world Chess Champion Garry Kasparov at chess in 1997. Power3, first 64 bit SMP, featured a data prefetch engine, non-blocking interleaved data cache, dual floating point execution units, and many other goodies. Power3 also unified the PowerPC and POWER Instruction set and was used in IBM's RS/6000 servers. The POWER3-II reimplemented POWER3 using copper interconnects, delivering double the performance at about the same price. Power4 was the first Gigahertz dual core processor launched in 2001 which was awarded the MicroProcessor Technology Award in recognition of its innovations and technology exploitation. Power5 came in with symmetric multi threading (SMT) feature to further increase application's performance. In 2004, IBM with 15 other companies founded Power.org.

Power.org released the Power ISA v2.03 in September 2006, Power ISA v.2.04 in June 2007 and Power ISA v.2.05 with many advanced features such as VMX, virtualization, variable length encoding, hyper visor functionality, logical partitioning, virtual page handling, Decimal Floating point and so on which further boosted the architecture leadership in the market place and POWER5+, Cell, POWER6, PA6T, Titan are various compliant cores. POWER6 is a high frequency design optimized for performance for the server market and for power. Power ISA has constantly improved the architecture performance and leadership in the market and has been front runner in providing higher and higher performance oriented processor design. With the release of Power ISA v.2.06 many new features are added to the architecture to further improve the performance of the architecture. One such feature which is added in Power ISA v.2.06 to improve application performance is Stride N prefetch.

Introduction to Prefetch:

Increasing memory latencies poses serious performance impact for workloads that are memory intensive. Data prefetch mechanism is to reduce the performance impact due to memory latencies. Typical technical workloads often access memory in regular, sequential patterns. Their working sets are so large that they often do not fit into the limited size of processor caches. Following figure 1 depicts the scenario of current prefetching mechanism:



POWER4 Hardware data prefetch schematic

Figure: 1

The data prefetch engines of processor can recognize the sequential data access patterns and initiate prefetching of adjacent data blocks from memory into processor cache to aid in performance for future accesses. Data prefetch engines continue to prefetch the data blocks as long as previously prefetched lines are consumed in sequential order. This will avoid stalls in the processor due to memory latencies as the required data is present in processor caches "ahead" of time.

Power architecture provides cache instructions to hint prefetch engines for data prefetching. Cache instructions such as dcbt/dcbtst allow applications to specify direction, depth, no of units and so on. These instructions if used will avoid the startup cost of automatic stream detection mechanism used by hardware. Hardware detection of stride stream is controlled by DSCR special purpose register. Hints provided to hardware engines using these data cache instructions may or may not be honored by processor.

Motivation for stride N prefetch:

Existing Power ISA Prefetch variants pose a limitation to have the data stream to be in adjacent 128 byte blocks. For example, if we have a data access pattern from different cache lines which are not adjacent but have fixed distance, existing data stream prefetch capability may not be effective.

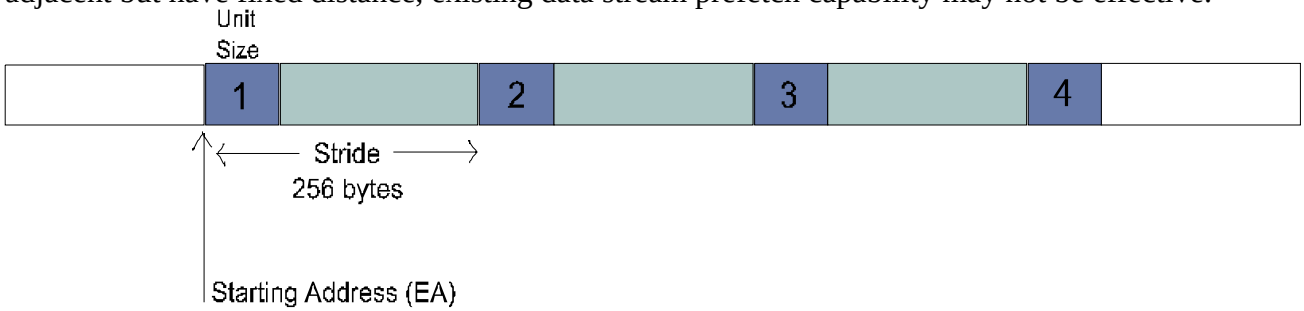


Figure:2

Here (Figure 2) data accessed are not in adjacent cache lines and initiating data stream prefetch in this case may bring many unwanted lines to the cache.

The new stride N feature in Power ISA 2.06 is more effective enhancement designed specifically for data accesses which are not adjacent in memory addresses but are separated by a fixed distance, that is when application accesses data in stride pattern. With this option, the processor prefetches lines based on the specified stride length. The data stream prefetch variant instructions dcbt/dcbtst are extended to include the stride N prefetch feature. To initiate Stride N data stream prefetch, application should use the new TH field encodings included in the instruction.

Data Cache instructions:

Data cache instructions control various aspects of the data cache. Currently Power ISA supports dcbt and dcbtst instructions with TH encodings 0b00000, 0b01000 and 0b01010. Instructions forms of dcbt and dcbtst presented here (Figure 3 and 4):

Data Cache Block Touch X-form

dcbt RA,RB,TH [Category: Server]
dcbt TH,RA,RB [Category: Embedded]

31	TH	RA	RB	278	/
0	6	11	16	21	31

Let the effective address (EA) be the sum (RA|0)+(RB).

Figure: 3

Data Cache Block Touch for Store X-form

dcbtst RA,RB,TH [Category: Server]
dcbtst TH,RA,RB [Category: Embedded]

31	TH	RA	RB	246	/
0	6	11	16	21	31

Let the effective address (EA) be the sum (RA|0)+(RB).

Figure: 4

The TH field in these instructions specifies different hints to the processor. Each TH field provides specific hint to prefetch engines to initiate either a data block or data stream.

Block data hint:

dcbt/dcbtst instructions with TH=0b00000-0b00111 provides a hint to the processor that program will probably soon access the data cache block containing the byte(s) addressed by EA. Prefetch engine will fetch exactly one cache line containing the byte addressed by EA.

Here is a sample code for block data hint accessing a EA x40000204:

```
lis     R1, 0x4000
ori     R1, 0x204
dcbt    R1, 0
```

Sample Code: 1 (Default TH field value is 0b00000).

Data Stream Hint:

dcbt/dcbtst instructions with TH=0b01000 and 0b01010 provide hints regarding a sequence of accesses to data elements called as “data stream”. There are two types of data streams. Load data stream and Store data streams. Data stream to which a program may perform “load” accesses is said to be “Load data stream” and instruction used to provide load data stream hint is dcbt. Data stream to which a program may perform “store” accesses is said to be “store data stream” and instruction used to

provide store data stream hint is dcbtst. When dcbt/dcbtst is used with these TH fields are called as “data stream variants”.

Applications can hint prefetch engines by executing a sequence of dcbt/dcbtst instructions to prefetch data streams. TH field encoding differs based on flavor of the fully specified stream and TH field encodings 0b01000 and 0b01010 are the currently supported fully specified streams. Minimum information needed to start a prefetch is the effective address(EA) of the data stream, direction and the stream ID.

TH	Description
01000	The ***dcbt/dcbtst*** instruction provides a hint that describes certain attributes of a data stream, and may indicate that the program will probably soon access the stream. The EA is interpreted as follows.

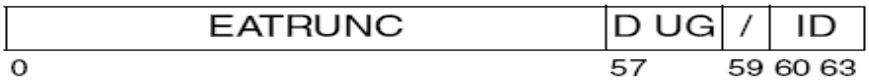


Figure: 5

TH 0b01000 is used to specify the truncated Effective address of the data stream to be fetched, stream direction and stream ID.

EATRUNC is the high order 57 bits (bit 0 to 56) of the effective address of the first element of the data stream (reason for only 57bits being used is that, prefetch will fetch a cache block from memory and each cache block is of 128 bytes).

‘D’ field specifies a direction of subsequent accesses from application.

ID specifies stream ID used for this data stream.

G (or GO) bit, when set, hardware prefetch engine will start the data stream prefetch with all other parameters to be default values such as number of units to be unlimited, non- transient, stride to be 128 bytes, offset to be zero, depth to be specified from DSCR[DPFD].

Follow the steps mentioned below to program the prefetch engines with minimum parameters to start the prefetching:

1)To initialize the Effective address, direction and streamID (TH 0b01000, G=1)

Here is a sample macro to program prefetch engines for data stream with minimal information:

```
#define LDSV_TH0b01000(EA, D, UG, STID)                {\
                                    unsigned long long localEA=EA, tmp=0;\
                                    localEA = ((localEA >> 7) << 7);\
                                    localEA |= ( ((D&0x1)<<6) | ((UG&0x1)<<5)|((STID&0xF)) );\
                                    __asm__ volatile("dcbt %0, 0, 0b01000"::"r"(localEA), "r"(tmp));\
                                  }
```

Sample Code:2 (LDSV – Load Data Stream Variant)

```
#define SDSV_TH0b01000(EA, D, UG, STID)              {\
                                    unsigned long long localEA=EA, tmp=0;\
                                    localEA = ((localEA >> 7) << 7);\
                                    localEA |= ( ((D&0x1)<<6) | ((UG&0x1)<<5)|((STID&0xF)) );\
                                    __asm__ volatile("dcbtst %0, 0, 0b01000"::"r"(localEA), "r"(tmp));\
                                    }
```

Sample Code: 3

TH 0b01010:

This encoding provides much more control over the stream prefetch. Application can specify the depth, number of units to fetch, whether the stream to be transient or not.

TH
0b01010

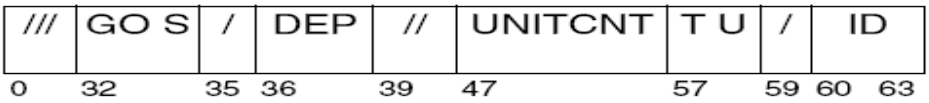


Figure: 6

Follow the steps mentioned below to program the prefetch engines with all parameters to start/stop the prefetching:

1) To initialize the Effective address, direction and streamID (TH 0b01000, G=0)
2) To initialize depth, number of units, transient/non-transient (TH 0b01010, G=0)
3) To start prefetch with GO bit on (TH 0b01010, G=1)

4)To stop stream prefetch with STOP bit on (TH 0b01010, G=0).

Values mentioned in the brackets of each procedure line is the corresponding TH field encoding to be used with GO bit value.

Here is a sample macro to provide a hint to processor to prefetch data stream:

```
#define LDSV_TH0b01010(EA, D, STID, GO, S, DEP, UNITCNT, T, U)        {\
                                    LDSV_TH0b01000(EA, D, 0, STID)\
                                    unsigned int local = 0, tmp=0;\
                                    local = (GO << 31) | (S << 29) | (DEP << 25);\
                                    local |= (UNITCNT << 7) | (T << 6) | (U << 5) | (STID&0xF);\
                                    __asm__ volatile("dcbt %0, 0, 0b01010"::"r"(local), "r"(tmp));\
                                    }
```

Sample Code: 4

```
#define SDSV_TH0b01010(EA, D, STID, GO, S, DEP, UNITCNT, T, U)        {\
                                    SDSV_TH0b01000(EA, D, 0, STID)\
                                    unsigned int local = 0, tmp=0;\
                                    local = (GO << 31) | (S << 29) | (DEP << 25);\
                                    local |= (UNITCNT << 7) | (T << 6) | (U << 5) | (STID&0xF);\
                                    __asm__ volatile("dcbtst %0, 0, 0b01010"::"r"(local), "r"(tmp));\
                                    }
```

Sample Code: 5

Sample macro to stop the stream:

```
#define LDSV_STREAM_STOP(STID)                  {\
unsigned int local = 0, tmp=0;\
local = (S << 29) | (STID&0xF);\
__asm__ volatile("dcbt %0, 0, 0b01010"::"r"(local), "r"(tmp));\
}
```

Sample Code: 6

```
#define SDSV_STREAM_STOP(STID)                  {\
unsigned int local = 0, tmp=0;\
local = (S << 29) | (STID&0xF);\
__asm__ volatile("dcbtst %0, 0, 0b01010"::"r"(local), "r"(tmp));\
}
```

Sample Code: 7

Details of Stride N Prefetch introduced in Power ISA 2.06:

Stride data stream:

Power ISA 2.06 now supports a new TH field encoding (TH 0b01011) for stride data prefetch. For data access patterns such as shown in fig, 7, existing fully specified stream prefetch may not be efficient, for the reason that unwanted data blocks may get fetched into cache.

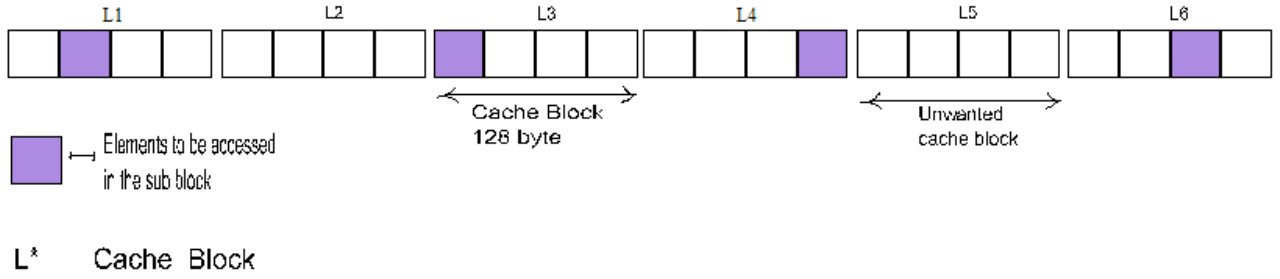


Figure: 7

In this example, each cache block is divided into four sub blocks of 32 bytes and stride distance between each data element accessed (blue boxes in the fig.7) is 7 sub blocks. If the existing fully specified stream method is used here, prefetch engine will bring the consecutive cache blocks from memory. In this case application does not need cache blocks L2 and L5, which not only wastes the bandwidth of bus but also increases the cache line congestion. Here the data accessed in memory are at a fixed distance. For such stride data access pattern, the new stride data stream capability is effective where prefetch engines bring the cache lines in a stride manner.

01011 The ***dcbt/dcbtst*** instruction provides a hint that describes certain attributes of a data stream.

The EA is interpreted as follows.

Figure: 8

Classic example of stride memory access is matrix multiplication considering the elements of the matrix are stored in linear array.

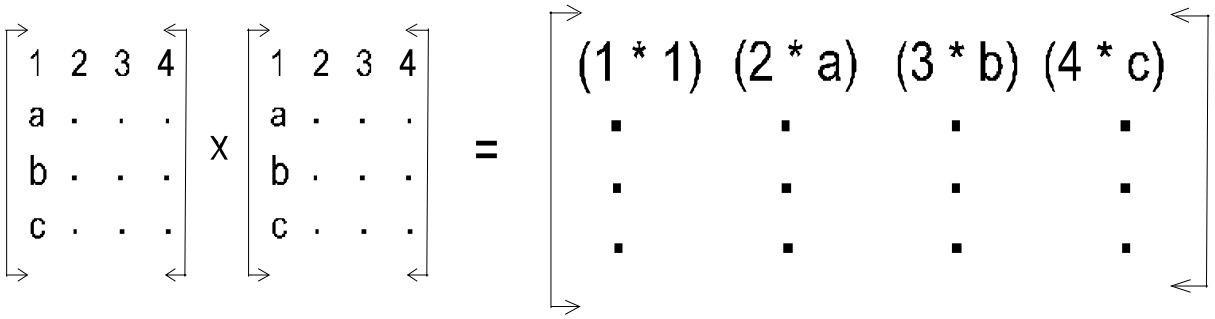


Figure: 9

Consider a matrix multiplication of A(*n*,*n*) x A(*n*,*n*) with A matrix having *n* row elements and *n* column elements. In this case, we need to fetch each column element to multiply with the row element which are placed at fixed distance of *n* elements (based on the data size). This fixed distance accesses naturally become a stride pattern access.

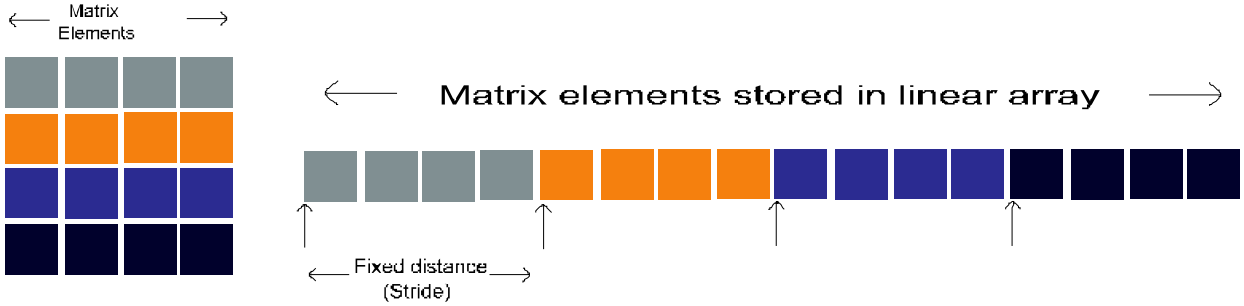


Figure: 10

Applications can program the engines to use stride feature with minimum parameters or with full control. In case of programming with minimum information, application has to provide the high order 57 bits of the effective address of first unit element, direction, stream ID, stride and offset. This will hint the engine with all other parameters with default values.

Follow the steps mentioned below to program the prefetch engines for stride prefetch with minimal parameters:

1) Initialize the Effective address, direction and streamID (TH 0b01000, G=0)
2) Initialize stride, offset and streamID (TH 0b01011, -)
3) Start prefetch with GO bit on (TH 0b01010, G=1)
4) Stop stream prefetch with STOP bit on (TH 0b01010, G=0).

Here is a sample macro to use to start a stride prefetch of a load/store data stream variant with minimum parameters:

```
#define LDSV_STRIDE_N_BASIC(EA, D, STID, GO, STRIDE, OFFSET)        {\
                                        LDSV_TH0b01000(EA, D, 0, STID)\
                                        unsigned int local = 0, tmp=0;\
                                        local = (STRIDE << 13 ) | (OFFSET << 8) | (STID&0xF);\
                                        __asm__ volatile("dcbt %0, 0, 0b01011"::"r"(local), "r"(tmp));\
                                        local= (1 << 31);\
                                        __asm__ volatile("dcbt %0, 0, 0b01010"::"r"(local), "r"(tmp));\
                                        }
```

Sample Code: 8

```
#define SDSV_STRIDE_N_BASIC (EA, D, STID, GO, STRIDE, OFFSET)        {\
                                        SDSV_TH0b01000(EA, D, 0, STID)\
                                        unsigned int local = 0, tmp=0;\
                                        local = (STRIDE << 13 ) | (OFFSET << 8) | (STID&0xF);\
                                        __asm__ volatile("dcbtst %0, 0, 0b01011"::"r"(local), "r"(tmp));\
                                        local= (1 << 31);\
                                        __asm__ volatile("dcbtst %0, 0, 0b01010"::"r"(local), "r"(tmp));\
                                        }
```

Sample Code: 9

To program prefetch engines with all parameters, an additional dcbt/dcbtst with TH 0b01010 is required in the sequence.

Follow the steps mentioned below to program the prefetch engines for stride prefetch with all parameters:

1) Initialize the Effective address, direction and streamID (TH 0b01000, G=0)
2) To initialize depth, number of units, transient/non-transient (TH 0b01010, G=0)
3) Initialize stride, offset and streamID (TH 0b01011, -)
4) Start prefetch with GO bit on (TH 0b01010, G=1)
5) Stop stream prefetch with STOP bit on (TH 0b01010, G=0).

Here is the sample macro to program stride prefetch with all the parameter values.

```
#define LDSV_STRIDE_N_ADVANCED (EA, D, STID, GO, S, DEP, UNITCNT, T, U, STRIDE, OFFSET)        {\
                                                LDSV_TH0b01010(EA, D, STID, 0, S, DEP, UNITCNT, T, U)\
                                                unsigned int local = 0, tmp=0;\
                                                local = (STRIDE << 13 ) | (OFFSET << 8) | (STID&0xF);\
                                                __asm__ volatile("dcbt %0, 0, 0b01011"::"r"(local), "r"(tmp));\
                                                }
```

Sample Code: 10

```
#define SDSV_STRIDE_N_ADVANCED (EA, D, STID, GO, S, DEP, UNITCNT, T, U, STRIDE, OFFSET)        {\
                                                SDSV_TH0b01010(EA, D, STID, 0, S, DEP, UNITCNT, T, U)\
```

```
unsigned int local = 0, tmp=0;\
local = (STRIDE << 13 ) | (OFFSET << 8) | (STID&0xF);\
__asm__ volatile("dcbtst %0, 0, 0b01011"::"r"(local), "r"(tmp));\
 }
```

Sample Code: 11

DCSR:

Data Stream Control Register is a privileged register. Content of this register affects the processor's handling of hardware-detected and software defined data streams.

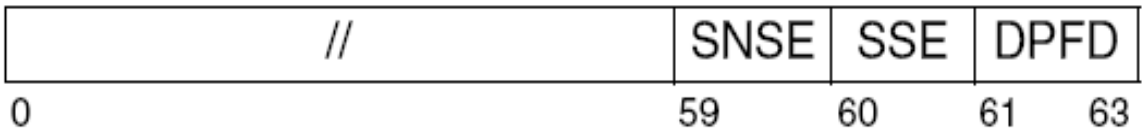


Figure 11: Data Stream Control Register

DPFD field value is considered as default depth field value if not specified in case software initiated data streams. SNSE field enable hardware detection of the stride N detection.

Market scope for stride N prefetch:

HPC market was the driving force behind stride N prefetch getting into Power ISA2.06. Various HPC applications like weather modeling and explosion simulations, auto manufacturers who do crash and other motion simulations, oil industry that does seismic analysis, biomedical engineering that does gene modeling and medical imaging, and more recently business analytics and data mining stand to benefit from this feature. Applications like generic physics modeling, visualization and multimedia-oriented compression and decompression belonging to the gaming industry can also get benefited by stride N feature

Summary:

Software prefetching is a promising technique for addressing the processor-memory performance gap. Applications can benefit from data prefetch capabilities of modern day super scalar processors.

Apart from applications, this feature can also be exploited by mathematical, engineering and scientific libraries. These are heavily used by high-performance computing applications which automatically benefits from this feature. By having this feature in common widely used libraries, applications using mathematical, engineering and scientific libraries will take advantage of this feature.

Stride N prefetch feature available in PowerISA 2.06 would help improve performance in scenarios where memory accesses are done in a stride manner. Using data prefetch instructions strategically in the application code will help boost application performance effectively.

Reference:

- Power ISA v2.06 Document - www.Power.org
- B. Sinharoy, R.N. Kalla, J.M. Tendler, R.J. Eickemeyer, and J.B. Joyner, “POWER5 System

Microarchitecture," *IBMJ. Res.& Dev.* **49**, No.4/5, 505–521 (2005).
- J.M. Tendler, J.S. Dodson, J.S. Fields,Jr., H. Le, and B. Sinharoy, "POWER4 System Microarchitecture," *IBMJ. Res.& Dev.* **46**, No.1, 5–25 (2002).
- L. Eisen, J.W. WardIII, H.-W. Tast, N. Mäding, J. Leenstra, S.M. Mueller, C. Jacobi, J. Preiss, E.M. Schwarz, and S.R. Carlough, "IBM POWER6 Accelerators: VMX and DFU," *IBMJ. Res.& Dev.* **51**, No.6, 663–683 (2007, this issue).
- POWER4 System Microarchitecture document – http://www-03.ibm.com/systems/p/hardware/whitepapers/power4.html

Authors:

Madhavan Srinivasan *IBM India System and Technology Labs, Bangalore India* (*masriniv@in.ibm.com*). Mr. Madhavan is a Staff Software Engineer and he joined IBM in 2003 after graduating from University of Madras with a B.E. degree in electrical and electronics engineering. As a member of the Power Architecture verification Tools Development Team at IBM, he concentrates on designing and developing architecture verification tools for POWER server processors. He was part of POWER5, POWER6 and latest POWER7 processor core bringup. His areas of interest are POWER architecture/micro architecture verification methodologies and OS internals.

Mehul Patel, *IBM India systems and technology lab, Bangalore, India* (*mehul.patel@in.ibm.com*). For past six years at IBM, Mehul has been developing hardware validation tools for IBM System p servers. These are system level test tools that are used to validate System p server's processor, memory and IO hardware. He has developed hardware validation tools for Power5, Power6 and upcoming Power7.

Deepak C Shetty *IBM India System and Technology Labs, Bangalore India* (deepakcs@in.ibm.com). Mr. Deepak holds a Bachelor of Engineering degree in Electronics from Pune University, India and Diploma in Advanced Computing from C-DAC, Pune. As a member of the IBM Hardware tools development team, he develops hardware validation tools for IBM systems (servers, blades, clusters) based on IBM PowerPC processors running on AIX and Linux. These are system level test tools that are employed to validate Processor, memory and I/O hardware. He has been working in this area for 6+ years developing tools for systems based on Power4, Power5, Power6 and the new Power7 processor. He holds 1 U.S. issued patent in the area of I/O verification. Other areas of interest include High Performance Computing and Software design.