EUROPEAN SYNCHROTRON RADIATION FACILITY
INSTALLATION EUROPEENNE DE RAYONNEMENT SYNCHROTRON
COMPUTING SERVICES

PICMG1.3 SHB Raid Performance

Evaluating next generation high-performance PC 4U computers

Petri Mäkijärvi

July 4, 2007

Abstract:
This paper reports a series of performance tests that were conducted on prototype industrial computer systems at the ESRF. PCI Express bus standard is making its way to the industrial systems. PICMG1.3 is a new industrial computer standard that comprises PCI Express bus. At the ESRF, camera detector systems are producing high bandwidth data. A data acquisition system could be designed around a PICMG1.3 compliant system. The report studies the eventual performances of such a data acquisition system by measuring prototype test rigs using simulator test programs. The study emphasizes Gigabit Ethernet data input and intermediate data storage on a RAID disk system. PCI Express bus characteristics are explained shortly and their influence to the system architecture discussed.

This page intentionally left blank.

Table of Contents

List of Figures

List of Tables

This page intentionally left blank.

1. Introduction

ESRF is using PICMG 1.1 based, industrial grade computers – often referred as “PC 4U” – in many control applications in the Machine control system and on beamlines. The system’s backplane is based on PCI-bus with some available ISA-bus slots. The ISA-bus is more or less electronics antique, although some ISA cards exist at the ESRF. The PCI-bus is very much alive and millions of installed PCI cards will make sure that it will not disappear in the near future.

But the aging PCI-bus has several drawbacks that makes it nearly impossible to use it in today’s high performance, high data volume applications, such as with Gigabit Ethernet adapters and with CCD camera detectors. The major penalizing factors in the PCI-bus design are its shared bus topology and its address/data multiplexing on the same 32-bit wide parallel bus. The below picture illustrates the system architecture of the actual, second generation PC 4U PCI-bus computer.

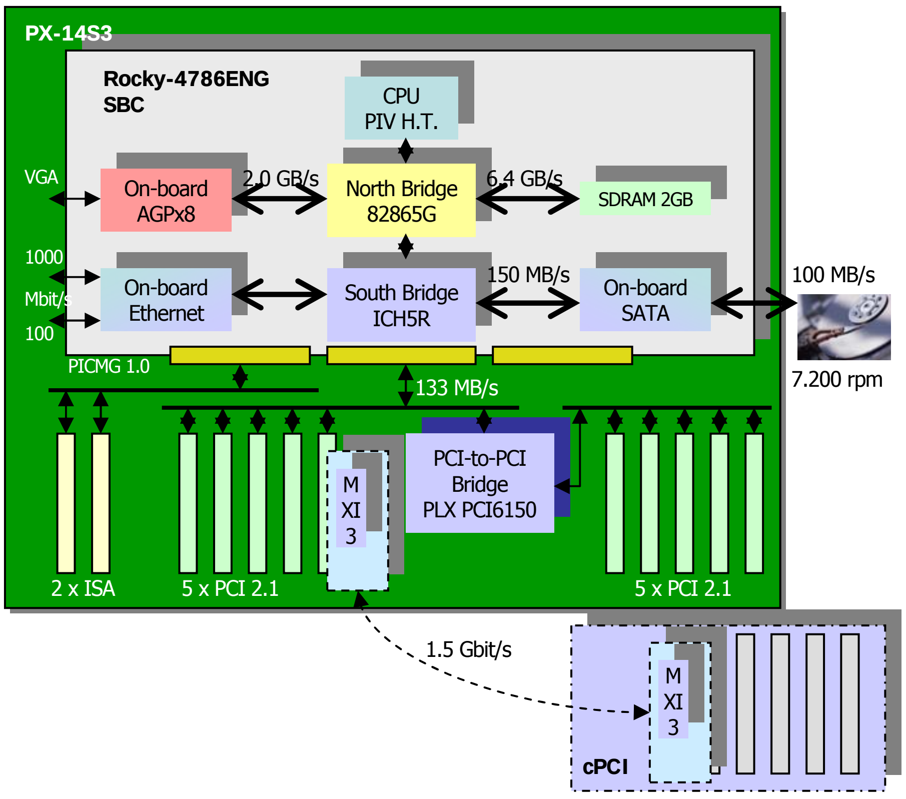


Figure 1: Actual PICMG 1.1 configuration

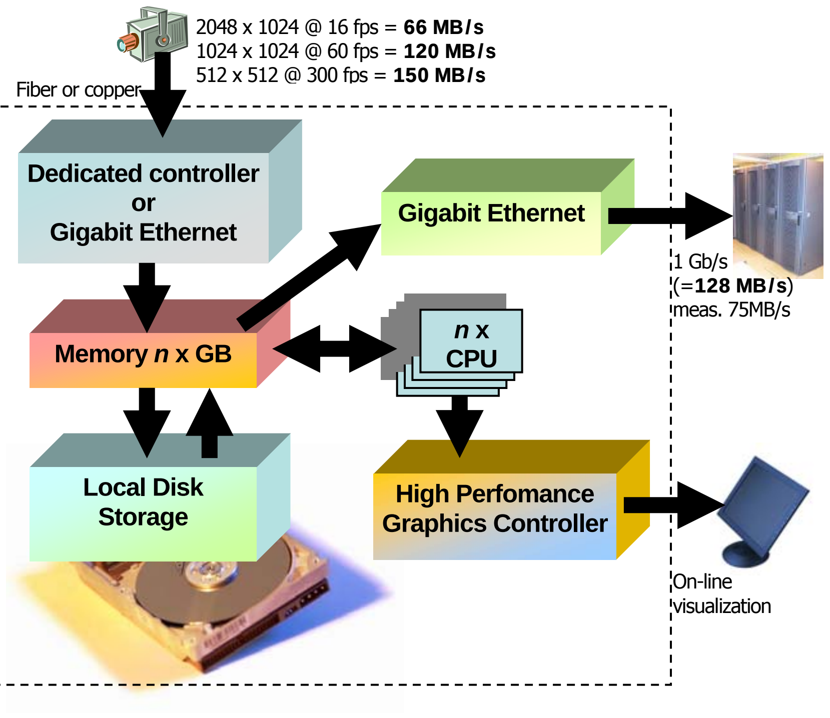
We can make following observations from the above diagram:

- The Single Board Computer (SBC) is well performing, modern computer equipment with sufficient I/O capabilities. Moore’s law makes that the North Bridge and South Bridge chips have evolved while the CPU, hard disk and Ethernet capacity increases.
- The graphics capabilities of the system are limited.
- The PICMG 1.1 design with its PCI 2.1 specification is more than ten years old and is showing its age. It is a shared bus where the 133 MB/s theoretical maximum transfer rate is shared with

all the devices connected on the PCI-bus. The active motherboard is also heavily bridged, creating delayed signal paths that are often not well understood by the end user.

1.1. High performance challenge

Although the current PCI-bus computer remains a reasonable solution for many low end I/O problems, it immediately shows its weaknesses should we need to cope with high end data acquisition systems. Let's take the problem of interfacing high performance cameras, such as *FReLoN*, *Dalsa* or *Sarnoff*, spiced with some future architectural expectations:



2048 x 1024 @ 16 fps = **66 MB/s**
1024 x 1024 @ 60 fps = **120 MB/s**
512 x 512 @ 300 fps = **150 MB/s**
Fiber or copper
Dedicated controller or Gigabit Ethernet
Gigabit Ethernet
1 Gb/s (=**128 MB/s**) meas. 75MB/s
Memory *n* x GB
n x CPU
Local Disk Storage
High Perfomance Graphics Controller
On-line visualization

Figure 2: High performance camera challenge

From the above data flow diagram we can immediately draw the following three system architectural conclusions:

1. The maximum of data transfer operations should occur on the computer board, through the North and South Bridges
2. In addition to powerful data transfer features we need local CPU and GPU power
3. We have to find a motherboard bus alternative for the PCI 2.1 bus

The first and the second item are likely to be dealt by Moore's law, which gives us hope that we can find a suitable Single Board Computer.

For the PCI 2.1 replacement the new PICMG 1.3 standard defines following bus systems:

- **PCI-X**
  64-bit wide version of PCI-bus gives an immediate factor of two performance boost. While the PCI-X 533 version of PCI-X 2.0 specification displays a whopping 8GB/s peak performance, this is rarely the case in the real world because the major PCI-bus flaw, shared topology remains in this design.

- **PCI Express**
  PCI Express **moves the highly successful PCI specification as it is** on a **serial**, **switched bus system**. Comparable to switched Ethernet in certain aspects, PCI Express allows non-shared, point-to-point connections between devices with data rates of **2.5 Gbit/s** and **5 Gbit/s** in both directions, simultaneously. Each pair of wire is called a *lane*. You can have 16 lanes for a graphics card, 8 lanes for a storage controller, 1 lane for a Gigabit Ethernet controller. More lanes multiply the theoretical maximum data transfer rate (*ex.* 16 lanes: 40 Gbit/s = 5 GB/s).

1.2. PCI Express switched fabric

PCI Express allows a star like system architecture with strong resemblance to the modern switched Ethernet fabric. Point to point connection is established between the PCI Express connected devices and the CPU. Also the direct memory access (DMA) and other resources are available to PCI Express devices without having to share the data bus with other devices in the same system. The PCI bus protocol is still present, contained in a transmission **protocol**, encapsulated in data transmission **packets**.

The packets can be prioritized and routed. From the PCI Express device's point of view it "owns" the data connection. The performance is limited to the quality of service that the communication switch is able to deliver to different, simultaneously connected devices. We can observe again the similitude with the telecommunications terminology. The existing PCI-bus based devices can be connected to the PCI Express based over a PCI Express to PCI-bus. The below picture illustrates an example of a PCI Express switched fabric.

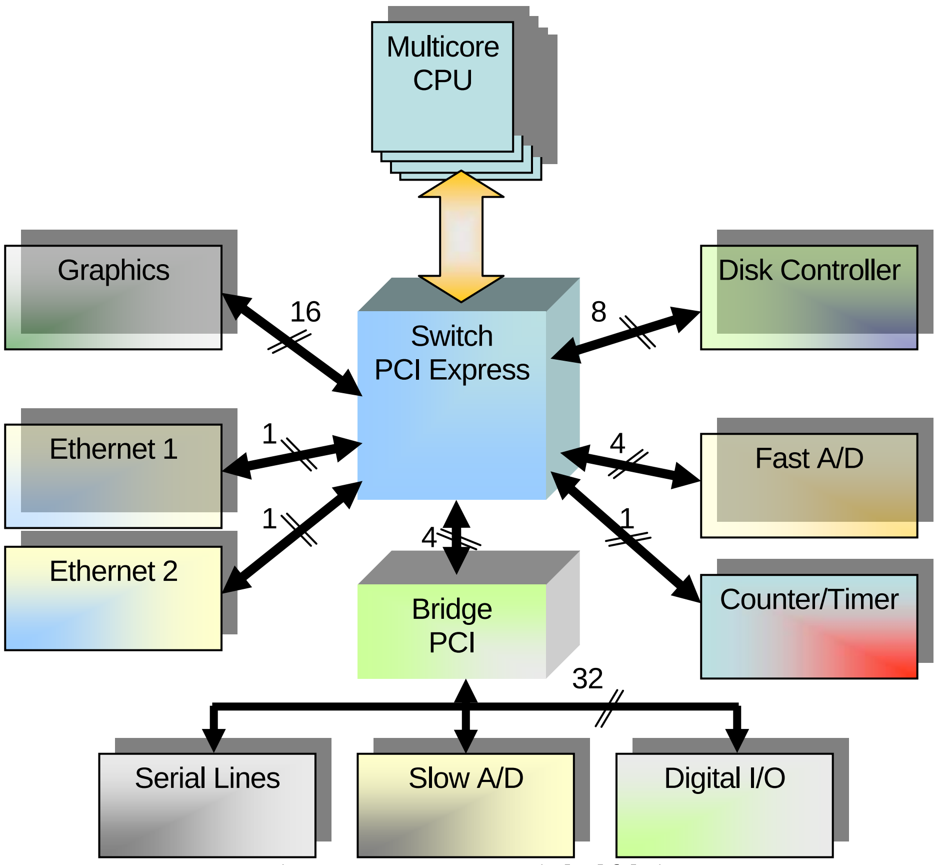


Figure 3: PCI Express switched fabric

2. Evaluation system’s architecture

There is an important base of PICMG 1.1 standard based control computers at the ESRF. The new PICMG 1.3 specification would provide us a smooth migration path while supporting both PCI-X and PCI-Express on active back plane boards. The current market offer for PICMG 1.3 based computers is still quite reduced but we have composed two test bench systems with the kind help of two of our suppliers. The below picture illustrates the architectural arrangement of the test bench.

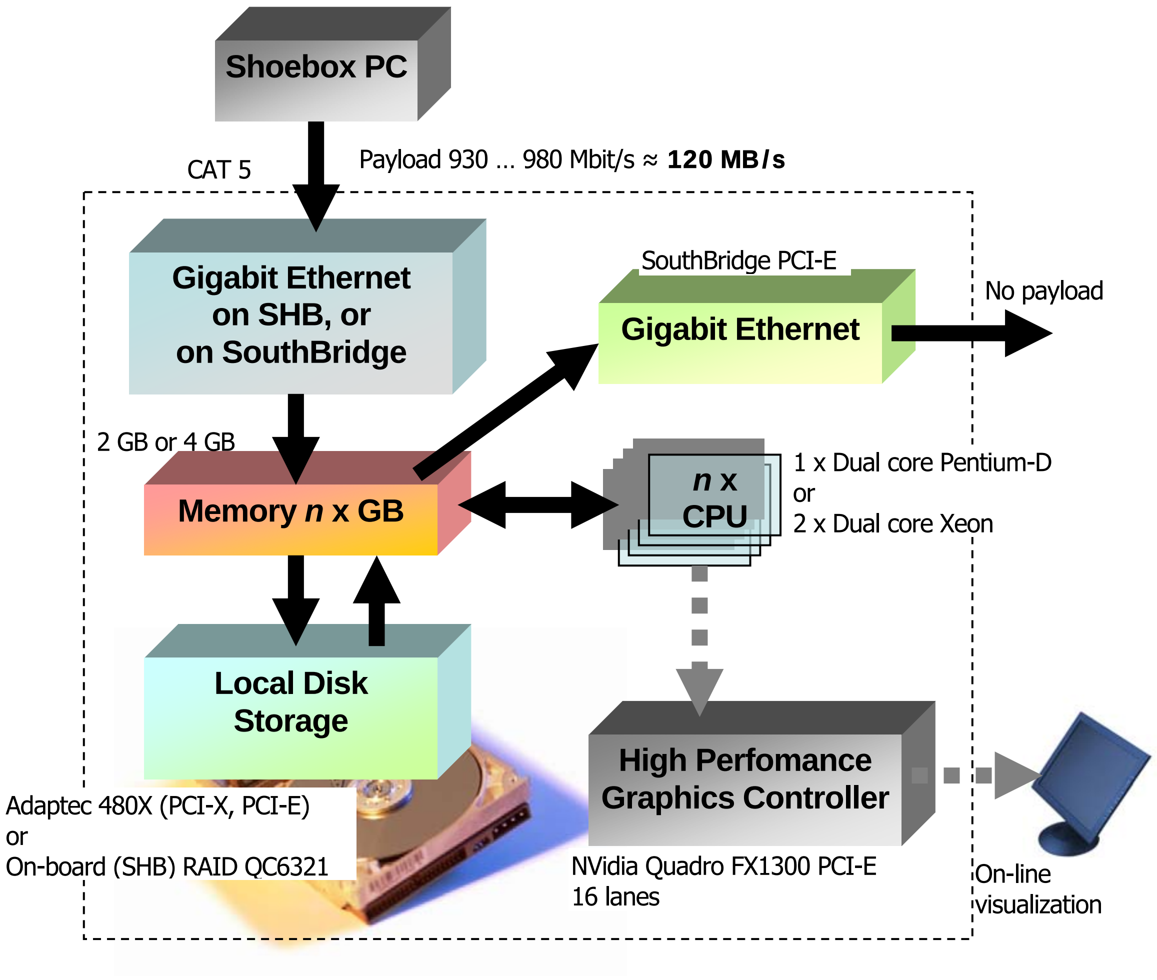

Shoebox PC
CAT 5
Payload 930 … 980 Mbit/s ≈ 120 MB/s
Gigabit Ethernet on SHB, or on SouthBridge
SouthBridge PCI-E
Gigabit Ethernet
No payload
2 GB or 4 GB
Memory n x GB
n x CPU
1 x Dual core Pentium-D
or
2 x Dual core Xeon
Local Disk Storage
Adaptec 480X (PCI-X, PCI-E)
or
On-board (SHB) RAID QC6321
High Perfomance Graphics Controller
NVidia Quadro FX1300 PCI-E
16 lanes
On-line visualization

Figure 4: Evaluation system architecture

- **Shoebox PC**
  is used generate a payload traffic on Gigabit Ethernet. With Maximum Transmission Unit (MTU) set to 8192, the entire bandwidth can be used easily with iperf 1.7 even when using TCP protocol.

- **Gigabit Ethernet controller**
  receiving the payload traffic from the shoebox PC was used with three different configurations: PCI Express x 1 on active motherboard, PCI Express x 1 on the Single Host Board (SHB, the CPU board) and PCI-X.

- **Memory**
  2 GB or 4 GB DDR2 memory depending of the tested system.

- **Local Disk Storage**
  was arranged as RAID-0 with two disks for most of the tests. A third, SATA disk outside the

RAID-0 arrangement was running for the operating system. Adaptec's 4805 (8 lane PCI Express) and 4800 (PCI-X) Serial Attached SCSI (SAS) RAID controllers were used to drive Seagate ST3146854SS 15000 rpm SAS disks.

A secondary RAID arrangement was set using SHB board's on-board RAID controller, provided by Intel's Southbridge QC6321. In third arrangement the RAID-0 functionality was provided by the Red Hat Linux operating system's kernel. Both of these arrangements were using Western Digital WD740ADFD-60 10.000 rpm SATA disks connected directly on the SHB board.

- **CPU**
  Tests were carried out with a single Dual Core Pentium –D (3.4 GHz) and with double Dual Core Xeon (2 GHz) CPUs.
- **Network storage** and **High Performance Graphics**
  These components were not tested, mainly because of lack of a suitable test program simulating the real life problem adequately. A graphics controller has been evaluated outside this test bench, namely PNY (NVidia) Quadro PX1300 (PCI Express, 16 lanes). It is currently tested on ID20.

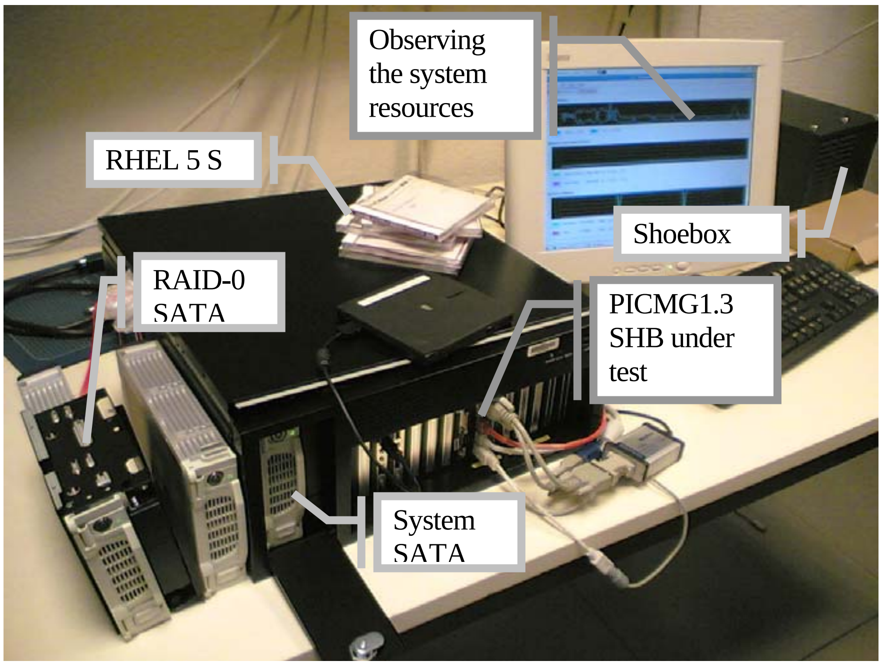


Figure 5: Nice photo of one of the test rigs

2.1. *Disk I/O test programs*

It goes beyond saying that benchmarking works only if the test software is adapted to the problem. This is particularly the case in our case, where the test program (See Appendix *BLISS "testdisk" program listing*) was developed by BLISS to test the actual hardware used with the *FReLoN* camera. Since the major behavior of a camera data acquisition driver (and its supporting processes and threads) is to **block write** to the hard disk. The block write capacity tests are therefore emphasized in this report.

In order to be more generic, another disk test program was used, called bonnie++. In addition to the large I/O testing, it tests creation and deleting time for small files, character based read and write operations and some other useful aggressions on the file system.

2.2. Operating systems

All tests were done with the following two operating systems:

1. ESRF/Linux 1.2.2: Red Hat Enterprise Linux 4 WS, v.4 32-bit for x86 (RHEL4WS)
2. Red Hat Enterprise Linux 5 Server, initial version, 32-bit for x86 (RHEL5S)

No significant performance variation can be reported between the two operating system versions. But because RHEL4WS was not able to manage SHB boards' on-board RAID controller (QC6321) **all results are reported as measured with RHEL5S**.

3. Test result summary

The actual testing was done over several weeks of time depending of the availability of the hardware. Tons of interesting data was mined over this period, especially with *bonnie++* test program. In this paper we are mainly interested in data storage of camera data. Therefore I will present below a summary of block write operations.

The first group of results is quite theoretical because the system does not have incoming camera data traffic.

In the second part, a payload is arranged to simulate the incoming, simultaneous camera data.

The third section of this chapter makes comments on the results, referring to the numbered bubbles next to the results.

3.1. Result summary with disk operations only

	Test name	BLISS single 4GB file write [MB/s]		bonnie++ 6GB block write [MB/s]		BLISS 8x512MB file write [MB/s]		bonnie++ 6GB block read [MB/s]	
Configuration	***Tested references***	PentiumD	2 x Xeon	PentiumD	2 x Xeon	PentiumD	2 x Xeon	PentiumD	2 x Xeon
SAS testbench									
2 x Seagate ST3146854SS **15000 rpm** aacraid-driverdisks 1.1.5-2429	**Adaptec 4805 (PCI Express)** 3	n/a	102	n/a	105	n/a	96	n/a	171
	Adaptec 4800 (PCI-X) 2	n/a	**128**	n/a	141	n/a	**134**	n/a	74
SATA testbench									
2 x Western Digital WD740ADFD-60 **10000 rpm**	**Intel Matrix Storage Manager Option ROM v5.6.2.1002 ESB2** 4	100	109	116	120	96	101	93	136
#1	**RHEL5S Linux kernel RAID** 1	**142**	138	**142**	141	**124**	119	**178**	168
	RHEL4WS Linux kernel RAID 5	121	n/a	116	n/a	100	n/a	155	n/a
SCSI (actual) reference system									
3 x Seagate ST373453LW **15000 rpm** Ultra320 SCSI	**Adaptec 29160N Ultra160 SCSI PCI** 6	n/a	97	n/a	101	n/a	101	n/a	126

Table 1: Result summary without network load

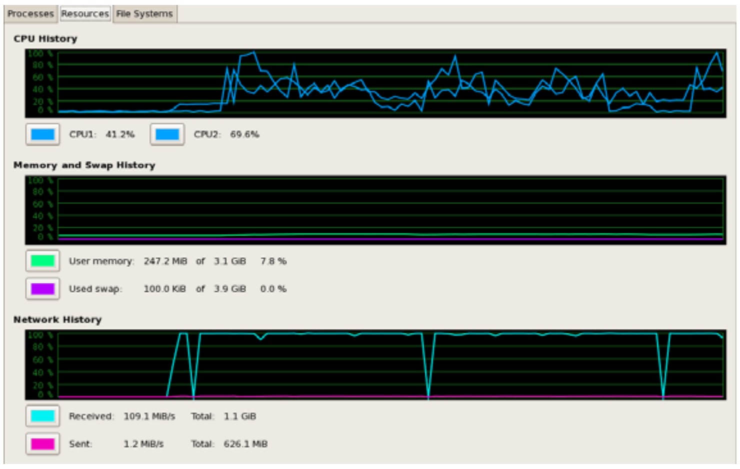
3.2. Disk operations with simultaneous incoming Ethernet payload

	Test name	BLISS single 4GB file write [MB/s]		bonnie++ 6GB block write [MB/s]		BLISS 8x512MB file write [MB/s]		bonnie++ 6GB block read [MB/s]	
Configuration	***Tested references***	PentiumD	2 x Xeon	PentiumD	2 x Xeon	PentiumD	2 x Xeon	PentiumD	2 x Xeon
SAS testbench									
2 x Seagate ST3146854SS **15000 rpm** aacraid-driverdisks 1.1.5-2429	**Adaptec 4805 (PCI Express)** 3	n/a	99	n/a	109	n/a	108	n/a	173
	Adaptec 4800 (PCI-X) 2	n/a	**89**	n/a	95	n/a	**94**	n/a	71
SATA testbench									
2 x Western Digital WD740ADFD-60 10000 rpm	**Intel Matrix Storage Manager Option ROM v5.6.2.1002 ESB2** 4	93	119	113	120	98	101	80	130
#1	**RHEL5S Linux kernel RAID** 1	138	**146**	138	**142**	**124**	120	**175**	166
	RHEL4WS Linux kernel RAID 5	111	n/a	122	n/a	96	n/a	146	n/a

Table 2: Result summary with Ethernet payload

3.3. Comments on results

1. The absolute winner for block write operations is the Red Hat Enterprise Linux 5 (Server) kernel (2.6.18) based RAID-0 configuration with SATA disks.
   Its supremacy is less clear with smaller Read/Write operations (not presented) but it is always performing much better than the average without any specific RAID hardware using only the standard SATA drives present on the SHB board.
   A small surprise was that the Dual Core Pentium-D was performing better in this test than the Double Dual-Core Xeon test system. But there is price to pay:
   In the test with simultaneous incoming Ethernet payload the CPU load was observed both on both test systems. The below graphs illustrates how there are situations where there is very little CPU capacity left on the Pentium-D test system. Although the Xeon system was observed to encounter the same type of CPU load on two of its virtual CPUs, it could happily perform other tasks with plenty of CPU capacity left on at least two other virtual CPUs.



CPU1: 41.2% CPU2: 69.6%

User memory: 247.2 MiB of 3.1 GiB 7.8 %
Used swap: 100.0 KiB of 3.9 GiB 0.0 %

Received: 109.1 MiB/s Total: 1.1 GiB
Sent: 1.2 MiB/s Total: 626.1 MiB

Figure 6: Pentium-D CPU/Network charge during RHEL5S Linux RAID-0 test

2. Although the Adaptec 4800 RAID-controller on PCI-X can be declared on honorable second place on this benchmark it can be criticized on its bad overall read performance. The test with the network payload was arranged so that Intel's server level Ethernet controller was used on the PCI-X bus. The parallel, shared bus structure shows immediately a sign of weakness and the disk performance will drop sharply.
   We can suggest that if PCI-X is to be used in high performance data acquisition I/O, it would be better not to have more than one device on the shared bus.

3. PCI Express version of Adaptec's RAID-controller, the 4805 was humming quite happily with constant performance no matter was there a network payload charge present or not.
   As a questionable feature we can remark that while we were observing the PCI Express bus with a PCI Express analyzer there was but four of the eight data lanes used for data transfer. One can ask is this a "feature" in the board's firmware or is it the Linux device driver that judges that no more lanes is needed because there is but two hard disk present in the configuration?

4. Both tested system came with the Intel's "R"-labeled Southbridge, which means that it contains a BIOS extension that allows the connected SATA disks to be arranged for RAID configuration. There is device driver support for this in Windows XP/Vista but unfortunately for us, Red Hat Enterprise Linux 4 is not supporting this chipset before the release 5 is out. Therefore all the tests were done with Red Hat Enterprise Linux 5 Server, where the chipset is recognized and RAID-0 can be configured accordingly.
   When looking at the good performance of the Linux kernel based RAID-0 configuration it becomes obvious that the BIOS based motherboard RAID is interesting only on Windows based system. But it can be a valuable asset for a SHB board should we purchase a commercial detector hardware which comes with software that runs only on Windows.

5. Red Hat Enterprise Linux 4 WS system's 2.6.9 kernel based Linux RAID is working a little bit slower in block write operations than its RHEL5S counterpart. But it is still fast enough to beat all the hardware/BIOS based alternatives that were tested. It is therefore a valid solution to make data acquisition on ESRF/Linux 1.x based systems (RHEL4WS).

6. The current data acquisition arrangement for camera data is a Dual Xeon, three SCSI-disks RAID system. It was evaluated with the same test bench software to give some reference values. You can observe that it is possible to achieve the same speed or better with two SATA disks on a ESRF/Linux 1.x based kernel RAID system.

4. Conclusion

Rapidly advancing detector camera technology and the advancement in switched, serialized data communications will require in the near future that the internal architecture of the front-end computers is modernized to use switched fabric, based on PCI Express. The imminent arrival of 10 Gbit/s copper Ethernet would require four 2.5 Gbit/s (GEN1) PCI Express lanes or two 5 Gbit/s (GEN2) lanes.

The actual design of camera interface boards would require a presence of at least one PCI-X slot. It would advisable to consider move this design on a four lane GEN1 PCI Express bus.

The intermediate data storage on the front end computers can be arranged using three or four Serial ATA (SATA) hard disk, directly connected to the PICMG 1.3 Single Host Board's Southbridge. The necessary RAID-0 configuration can be done using Linux kernel based RAID services.

It is necessary to have at least two Dual Core Xeon CPU to manage the switched communication fabric with high transfer rates while keeping up a CPU resource reserve for auxiliary tasks, such as detector image on-line display and data transfer to the remote storage systems.

Special thanks to Alejandro Homs (BLISS) for the camera performance oriented test program and to Pierre Pinel, Bernard Regad and Yves Nicolas (CCTF) for their assistance in test bench preparation.

ν

A) BLISS “testdisk” program listing

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <asm/fcntl.h>
#include <sys/stat.h>
#include <sys/time.h>

#define NAME_LEN     256
#define DATA_LEN     (16 * 1024 * 1024)

char fnamebase[NAME_LEN];
char *data;
int datalen;
int filenr;

void cleanupdata()
{
	free(data);
}

int initdata(int len)
{
	int *ptr, i;

	data = (char *) malloc(len);
	if (data == NULL) {
		fprintf(stderr, "Error allocating %d bytes\n", len);
		return -1;
	}
	datalen = len;

	if (atexit(cleanupdata) != 0) {
		fprintf(stderr, "Error registering atexit cleanupdata\n");
		cleanupdata();
		return -1;
	}

	ptr = (int *) data;
	for (i = 0; i < len / sizeof(int); i++)
		ptr[i] = random();

	return 0;
}

double elapsedtime(struct timeval *t0, struct timeval *t1)
{
	double elapsed;

	elapsed  = t1->tv_sec - t0->tv_sec;
	elapsed += (t1->tv_usec - t0->tv_usec) * 1e-6;
	return elapsed;
}

char *getfname(int fileidx)
{
	static char fname[NAME_LEN];

	sprintf(fname, "%s_%04d", fnamebase, fileidx);
	return fname;
```

```
}

int writefile(int fileidx, long long nbytes)
{
        int fd, writeb, left;
        long long all, aux;
        char *fname, *ptr;

        fname = getfname(fileidx);
        fd = open(fname, O_CREAT | O_TRUNC | O_WRONLY | O_LARGEFILE, 0666);
        if (fd == -1) {
                fprintf(stderr, "Error creating %s: %s\n", fname,
                                strerror(errno));
                return -1;
        }

        all = 0;
        while (all < nbytes) {
                aux = nbytes - all;
                if (aux > datalen)
                        aux = datalen;
                left = (int) aux;
                ptr = data;
                while (left > 0) {
                        writeb = write(fd, ptr, left);
                        if (writeb <= 0) {
                                fprintf(stderr, "Error writing on %s: %s\n",
                                        fname, strerror(errno));
                                return -1;
                        }
                        left -= writeb;
                        ptr += writeb;
                        all += writeb;
                }
        }

        if (close(fd) == -1) {
                fprintf(stderr, "Error closing %s: %s\n", fname,
                                strerror(errno));
                return -1;
        }

        return 0;
}

int writemultfiles(int nfiles, long long nbytes, int erase)
{
        int i;
        struct timeval t0, t1;
        double elapsed;
        long long all;

        if (erase)
                for (i = 0; i < nfiles; i++)
                        unlink(getfname(filenr + i));
        sync();

        gettimeofday(&t0, NULL);

        for (i = 0; i < nfiles; i++)
                if (writefile(filenr + i, nbytes) == -1)
                        return -1;
        filenr += nfiles;
        sync();
        gettimeofday(&t1, NULL);

        elapsed = elapsedtime(&t0, &t1);
        i = (int) (nbytes * nfiles / (1024 * 1024));
        printf("Transferred %4d MBytes (%4d files) in %7.3f s: %.2f MB/s\n",
                i, nfiles, elapsed, i / elapsed);

        return 0;
}

int makesequence(int erase)
{
        long long nbytes;
        int nfiles;

        filenr = 0;

        nbytes = 1024UL * 1024 * 1024;
        nbytes *= 4;
        nfiles = 1;
        if (writemultfiles(nfiles, nbytes, erase) == -1)
                return -1;

        nbytes = 8 * 1024 * 1024;
        nfiles = 512;
        if (writemultfiles(nfiles, nbytes, erase) == -1)
                return -1;

        nbytes = 8 * 1024;
        nfiles = 2048;
        if (writemultfiles(nfiles, nbytes, erase) == -1)
                return -1;

        return 0;
}


int main(int argc, char **argv)
{
        if (argc == 1) {
                fprintf(stderr, "Usage: %s outfilebase\n", argv[0]);
                exit(1);
        }
        strcpy(fnamebase, argv[1]);

        if (initdata(DATA_LEN) == -1)
                exit(1);

        if (makesequence(1) == -1)
                exit(1);

        if (makesequence(0) == -1)
                exit(1);

        return 0;
}
```