DL-TR-2006-004

CCLRC

HPC Integer Benchmarks: An Indepth Analysis of the Performance Sensitivity of Legacy codes on Current Hardware Platforms

Christine A. Kitchen, Martyn F. Guest, Michael Ehrig, Miles J. Deegan, Igor N. Kozin *and* **Richard Wain**

March 2006

Council for the Central Laboratory of the Research Councils

© 2006 Council for the Central Laboratory of the Research Councils

Enquiries about copyright, reproduction and requests for additional copies of this report should be addressed to:

Library and Information Services
CCLRC Daresbury Laboratory
Daresbury Warrington
Cheshire WA4 4AD
UK
Tel: +44 (0)1925 603397
Fax: +44 (0)1925 603779
Email: library@dl.ac.uk

ISSN 1362-0207

Neither the Council nor the Laboratory accept any responsibility for loss or damage arising from the use of information contained in any of their reports or in any communication about their tests or investigations.

HPC Integer Benchmarks: An Indepth Analysis of the Performance Sensitivity of Legacy codes on Current Hardware Platforms

Christine A. Kitchen, Martyn F. Guest, Michael Ehrig[†], Miles J. Deegan, Igor N. Kozin and Richard Wain
Computational Science and Engineering Department, CLRC Daresbury Laboratory, Daresbury, Warrington, Cheshire, WA4 4AD, UK
[†]Hewlett Packard, Schickardstrasse 32, D-71034 Boeblingen, Germany.
Date: 10th March 2006 (revised 5th November 2006).

Abstract

This paper presents a performance analysis of an HPC Integer Benchmark comprising seven legacy codes (and 13 associated data sets) designed to test both integer and boolean performance. In presenting an overview of single processor performance, data is analysed from a variety of Itanium2, Opteron and EM64T Xeon processors (16 machines in total), plus the IBM power5-based p5-575, with a particular focus on the emerging dual-core systems from AMD and Intel. The subsequent performance analysis considers for each benchmark the impact of memory latency and memory bandwidth and dependency on both clock speed and cache.

Defining a set of associated "RATE" benchmarks, we demonstrate clear pointers to the memory bandwidth issues on dual-core systems, and define a "workload" benchmark designed to fully reveal these effects when considering throughput workloads. Finally we present initial results from the MPI implementation of a number of the benchmark codes on a variety of parallel systems.

1. Introduction

Much of the performance evaluation work carried out by the Distributed Computing Group at CCLRC Daresbury Laboratory has naturally focused on 64-bit floating point arithmetic. To date this work has considered the performance attributes of current and emerging systems in scientific and technical computing, derived though a variety of synthetic and application-based floating point metrics. Numerous systems have been rigorously evaluated using important applications. Recent evaluations have included the Cray XD1 and Infinipath Clusters [1]. The primary goals of these evaluations are to a) determine the most effective approaches for using each system, b) evaluate benchmark and application performance, both in absolute terms and in comparison with other systems, and c) predict scalability, both in terms of problem size and in number of processors.

However, there are a number of established and emerging disciplines where a greater emphasis is given to the integer and boolean performance of processors e.g. the bioinformatics community, intelligence agencies etc. There are of course a number of well established benchmarks that focus on integer performance, notably SPEC [2] and HPCC [3]. The former dedicates the entire SPECint and SPECint_rate [4] benchmarks to track integer related performance, while HPCC includes the RandomAccess benchmark [5] to measure the rate of integer random updates of memory. In line with our approach to evaluating the floating point attributes of key application codes, we felt it timely to develop our own integer-based benchmarks to reflect the performance interests of the above communities.

The aim of the present work is to examine both the single and parallel processor performance of a set of integer-based benchmark codes *(comprising seven legacy codes and 13 associated datasets)* on a variety of different processors and hardware platforms. The overall approach is to assess the performance of each code in light of the different processor attributes and system architectures. Specifically we look to understand each code's dependency on clock speed, cache, both memory latency and bandwidth and the impact that each factor has on serial, rate and parallel work loads.

This report is broken down into a number of sections, with Section 2 providing a description of each of the benchmark codes and a summary of the variety of systems used in the benchmarks. Subsequent sections each focus on a specific aspect of the observed performance. Section 3 reports the overall serial benchmark times for each of the codes as a function of data set, and provides a performance comparison contrasting the Intel Itanium (IA64) performance with both the Intel Xeon EM64T (code name "Nocona" and "Irwindale") and AMD Opteron (x86-64). A more detailed analysis of these comparisons is given in Sections 4 and 5. Section 4 looks to rationalise the performance in terms of specific attributes of the systems under consideration e.g. memory latency and memory bandwidth, processor speed etc. Section 5 turns to software specific effects that impact on the observed performance – the choice of compiler, compiler optimisation level and the impact of coding language through a consideration of performance delivered when using C and FORTRAN.

The remaining sections look to extend the preceding focus on just single processor performance to a consideration of RATE (throughput), workload and parallel performance. Section 6 considers the impact of throughput processing with the development of RATE benchmarks, very much in line with the SPEC provision of both SPECint and SPECint_rate benchmarks [4]. Section 7 extends this analysis still further with the development of a simulated "workload throughput" benchmark, looking to reflect "real world" usage by simulating a job mix of all of the benchmark codes. By defining this set of associated RATE benchmarks, we demonstrate clear pointers to the memory bandwidth issues on dual-core systems, while the "workload" benchmark is designed to fully reveal these effects when considering throughput workloads. Finally we turn to the parallel performance of the benchmarks, and consider the parallel implementation and performance of a number of the integer codes using MPI on a variety of parallel systems. Note that a presentation version of this report is available [6].

2. The HPC Integer Benchmark and Evaluation Systems

The HPC Integer benchmark is comprised of 7 legacy codes, with 13 different data sets, designed to assess both integer and boolean performance. Originally developed for Cray vector supercomputers, the functionality and purpose of each the named codes are as follows:

- *Linemap*: performs a Gray-code search for a linear mapping (requires extensive use of popcnt).
- *Permutit*: permutes the order of bits of each entry in an array of 64-bit words.
- *Hadamard*: performs a transform similar to the Hadamard transform.
- *RuWarray*: calculates a read-update-write on random elements of a large array.
- *Treesearch*: undertakes a tree search to solve a 33-peg solitaire game.
- *Linequ*: solves a system of linear binary equations – 700 equations in 700 unknowns.
- *IObench*: writes a number of 64-bit words of bitstream to a file and reads the file.

In presenting an overview of processor performance for each of the above codes, performance data is analysed from a variety of Itanium2, Opteron and EM64T Xeon processors (16 machines in total, plus the IBM power5-based p5-575, see Table 1), with a particular focus on the emerging dual-core systems from AMD and Intel. The systems evaluated reside at a number of sites – some at Daresbury itself, plus those at both academic- and vendor-sites e.g. Xeon systems at Dell and a number of Itanium2-based RX systems at Hewlett Packard. The subsequent performance analysis considers for each benchmark the impact of memory latency, memory bandwidth and the dependency on both clock speed and cache.

Processor	CPU speed	Interconnect	Location
Opteron270	2.0GHz	Myrinet 2K	Leeds
Opteron248	2.2GHz	Myrinet 2K	RAL
Opteron875	2.2GHz	–	HP
Opteron150 (Cray XT3)	2.4GHz	Customised	Pittsburgh
Opteron250	2.4GHz	Rapid Array	DL
Opteron280	2.4GHz	–	HP
Opteron852	2.6GHz	Infinipath	Streamline
Xeon (Bensley)	3.46GHz	–	Intel
PowerEdge1850	3.2GHz	Infiniband	TACC
PowerEdge1850	3.2GHz	Infiniband	TACC
IBM p5-575	1.5GHz	Federation (HPS)	HPCx
Itanium2 (SGI Altix)	1.3GHz	NUMAlink	CSAR
Itanium2, (SGI Altix)	1.5GHz	NUMAlink	CSAR
Itanium2 (HP rx5670)	1.5GHz	ZX1	HP
Itanium2 (HP, rx8620)	1.6GHz	SX1000	HP
Itanium2 (HP rx1620)	1.6GHz	ZX1	HP
Itanium2 (HP SD64000B)	1.6GHz	SX2000	HP

Table 1: Summary of the Systems Evaluated.

Given the dependency of this analysis on the architectural characteristics of each of the accessed systems, we describe these in some detail below when outlining each of the systems characteristics:

1. The Cray XT3, **Bigben**, at Pittsburgh Supercomputing Centre: The XT3 is Cray's third-generation massively parallel processing system. The XT3 builds upon a single processor node, or processing element (PE), using the AMD *Opteron* model 150 processors. These processors are connected with a customized interconnect managed by a Cray-designed

Application-Specific Integrated Circuit (ASIC) called SeaStar. The compute PEs run a lightweight operating system kernel called Catamount. The Opteron core has three integer units and one floating point unit capable of two floating-point operations per cycle. Because the processor core is clocked at 2.4 GHz, the peak floating point rate of each compute node is 4.8 GFlops. The memory structure of the Opteron consists of a 64KB 2-way associative Level 1 (L1) data cache, a 64KB 2-way associative L1 instruction cache, and a 1 MByte (MB) 16-way associative, unified Level 2 (L2) cache. Each PE has 2 GByte (GB) of memory but only 1 GB is usable with the kernel used for our evaluation. The memory DIMMs are 1 GB PC3200 - the peak memory bandwidth per processor is 6.4 GB/s. Also, the Opteron 150 has an on-chip memory controller. As a result, memory access latencies with the Opteron 150 are in the 50-60 ns range.

2. An *Opteron* cluster, **Everest**, at Leeds University: The cluster comprises Sun Microsystems' Sun Fire V40z and V20z servers with dual-core AMD Opteron processors integrated by Streamline Computing. Seven of these (V40z) comprise four 2.2 GHz dual-core processors configured with 192 GB memory. Eighty seven V20z servers are interconnected with a Myrinet network; each of these comprises two 2.0 GHz dual-core processors sharing in total 0.7 TByte (TB) of distributed memory across 348 processor cores. The system runs the Linux (64-bit SuSE) operating system. The present benchmarks were run on the myrinet-connected V20z servers i.e. two 2.0 GHz dual-core processors per node.

3. An *Opteron* cluster, **SCARF**, at the Rutherford Laboratory integrated by Streamline Computing: 256 AMD Opteron 248 (2.2 GHz) processors with 2GB (224 processors) and 4 GB (32 processors) of memory per processor. The system is configured as 128, 2-way SMPs with Myrinet 2K (M3F-PCIXD-2) interconnect. The system runs the RedHat ES 3.0 operating system and PGI compilers.

4. An *Opteron* cluster at Streamline Computing: 32 AMD Opteron 252 (2.6 GHz) processors with 2GB of memory per processor. The system is configured as 16, 2-way SMPs with Pathscale interconnect. The system runs the RedHat ES 3.0 operating system and Pathscale (EKO v2.2) compilers.

5. The Cray XD1 at Daresbury: 70 AMD 2.4 GHz *Opteron* 250 processors with 2GB of memory per processor. System is configured as 35 x 2-way SMPs with Cray's proprietary RapidArray interconnect fabric. PGI compilers were used (6.0.8).

6. The SGI Altix 3700 system, **NEWTON**, at Manchester Computing Centre comprising 512 *Itanium2* processors and SGI's NUMAlink interconnect. Each processor has 256 KB L2 cache, 16 KB L1 data cache, 16 KB L1 instruction cache. The 1.3 GHz processors have 3 MB L3 cache, while the 1.5 GHz processors have 6 MB. The machine has an aggregate of 1 TB of shared memory. SGI NUMAlink provides sub-microsecond hardware latency, 3-5 microseconds MPI latency, sub-microsecond latency for one-sided communications, and 12.8GB/s aggregate bandwidth per brick (4 CPUs). The Operating System comprises a Linux kernel with SGI Propack extensions (based on Redhat 7.2).

7. HP *Itanium2* cluster at HP: 144 Itanium-2 1.5 GHz processors. The system is configured as 72 x 2-way SMP RX1620 nodes with an Infiniband and Gigbit interconnect. Each compute node has 18GB of memory. The system runs the HPUX (HPUX 11.23) operating system.

8. The HP Integrity Superdome SD64000B. The system features 64 socket (128 core) 1.6 GHz *Itanium2 Montecito* processors with 12 MB L3 cache per core. The SD64000B architecture is based on HP's new cell infrastructure (SX2000), with 533 MHz frontside bus. The system runs the HPUX (HPUX 11.23) operating system.

9. The **HPCx** phase2a system at Daresbury, comprising 96 IBM power5 eServer nodes (1.5 GHz) i.e. 1536 processors. The system is equipped with 3.2 TB of memory and 36 TB of disk. In the power5 architecture, a chip contains two processors, together with L1 and L2 cache. Each processor has its own L1 instruction cache of 32 KB and L1 data cache of 64 KB integrated onto one chip. Also on board the chip is the L2 cache (instructions and data) of 1.9 MB, which is shared between the two processors. Four chips (8 processors) are integrated into a multi-chip module (MCM). Two MCMs (16 processors) comprise one frame. Each MCM is configured with 128 MB of L3 cache and 16 GB of main memory. The total main memory of 32 GB per frame is shared between the 16 processors of the frame. The frames in the HPCx system are connected via IBM's High Performance Switch (HPS). Each frame is one 16-way LPAR – the names LPAR and frame are synonyms for computer node on HPCx phase2a.

In addition to the systems above systems, benchmarks have been run on a variety of single node server systems, including:

- A variety of Dell Poweredge 1850 nodes, comprising both "Nocona" and "Irwindale" EM64T dual processors. The former nodes, with 1MB L2 cache, were clocked at 2.8 GHz, 3.2 GHz, 3.4 GHz and 3.6 GHz. The Irwindale nodes, with 2MB L2 cache, were clocked at 3.0 GHz, 3.2 GHz, 3.4 GHz and 3.6 GHz. These nodes

were located at the TACC centre as part of the Wrangler cluster system, and at Dell.

- A prototype of the Intel Xeon 5080 processor - the EM64T “Bensley/Dempsey” platform - clocked at 3.46 GHz. The prototype PowerEdge 1950 node featured 4 cores, 2 chips, and 2 cores / chip. Each core has a primary Cache of 12KB (I) + 16KB (D) on chip, and a secondary L2 cache of 2MB (I+D).
- Opteron-based DL145 and DL585 servers from Hewlett Packard. The DL145 server comprised 2 x 2.4 GHz dual-core AMD Opteron 280 processors sharing a total of 2 GB memory. The DL585 server comprised 4 x 2.2 GHz dual-core AMD Opteron 875 processors sharing a total of 4 GB memory. The systems run the Linux (64-bit SuSE) operating system.
- An HP RX5670 server, with 4 x 1.5 GHz Itanium2 Madison processors, each with 6 MB L3 cache, 400 MHz frontside bus and ZX1 cell interconnect. The system runs the HPUX (HPUX 11.23) operating system.
- An HP RX8620 server, with 8 x 1.6 GHz Itanium2 Madison processors, each with 6 MB L3 cache, 400 MHz frontside bus and SX1000 cell interconnect. The system runs the HPUX (HPUX 11.23) operating system.
- An HP RX1620 server with 2 x 1.6 GHz Itanium2 Madison processors, each with 3 MB L3 cache, 533 MHz frontside bus and ZX1 cell interconnect. The system runs the HPUX (HPUX 11.23) operating system.

3. SERIAL PERFORMANCE: IA64 and x86-64

The analysis below concentrates on the performance differential between Itanium2 (IA64) with respect to Xeon EM64T and AMD Opteron (x86-64). A more thorough break down of the various code dependencies is examined in Section 4. It should be noted that throughout this report the focus is on performance and at no point is the price of the systems factored into the analysis. At this moment, it is probably fair to say that the Intel Xeon EM64T and AMD Opteron are comparable in price, while the Intel Itanium2 is, at a conservative estimate, a factor of 3 times the price of the x86-64 systems.

Before presenting the results of the current benchmarking, we provide in Table 2 tabulated SPECfp2000 [7] and SPECint2000 [4] results for many of the processors featuring in the present exercise. These should be viewed at best as illustrative of the expected performance differential between IA64 and x86-64 based systems, for it must be remembered that many of the results are now several years old, and that more aggressive figures might be forthcoming were they to be re-run with today’s compilers. Nevertheless, it is clear that the SPECfp ordering is quite different from that shown by SPECint. While the Itanium2 and Power5 CPUs dominate the former, the integer-based benchmarks reveal a clear advantage of the x86-64 CPUs. It is this ordering that might be expected in the current integer benchmarks.

SYSTEM, PROCESSOR & CPU SPEED	**SPECFP-2000**	**SPECINT-2000**
IBM eServer 326m Opteron270 / DC 2.0GHz	1785	1452
Pathscale ASUS SK8N Opteron248 / 2.2GHz	1691	1452
HP Proliant DL145 Opteron275 DC 2.2GHz	1878	1518
AMD TYAN 2865 Opteron150 2.4GHz	1955	1681
HP Proliant DL145 G2 Opteron280 DC 2.4GHz	1914	1672
HP Proliant DL145 Opteron252 2.6GHz	2084	1708
Dell Prec. Wkstn. 690 Xeon 5080 (Bensley) DC 3.73GHz	1932	1813
Dell PowerEdge 1850 P4 Xeon 3.2GHz / 2MB L2	1716	1555
Dell PowerEdge 1850 P4 Xeon 3.2GHz / 1MB L2 (est.)	1396	1383
SGI Altix 3000 Itanium2 1.5GHz / 6MB L3	2148	1243
SGI Altix 3000 Itanium2 1.3GHz / 3MB L3	1854	1019
HP Integrity RX1620 Itanium2 1.6GHz 3MB L3	2692	1452
HP Integrity RX5670 Itanium2 1.5GHz 6MB L3	2108	1312
IBM eServer p5-575 1.5GHz	2185	1143

Table 2: SPECfp and SPECint Performance

3.1 Linemap

A number of the integer benchmark codes – including *Linemap* – relied historically on efficient intrinsic bit manipulation functions. This of course led to much of the success of Cray given the hardware implementation of intrinsics such as population count (popcnt) and leading zero (leadz) count.

Linemap performs a Gray-code[1] search for a linear mapping which invokes extensive use of popcnt, the population count instruction. Popcnt is a function that counts the number of set bits in a data object. The three data sets invoked each specify two arguments – the number of dimensions in vector space (one in each case) and the number of vectors to be searched over, increasing from 20 to 21 and 22 in the three data sets.

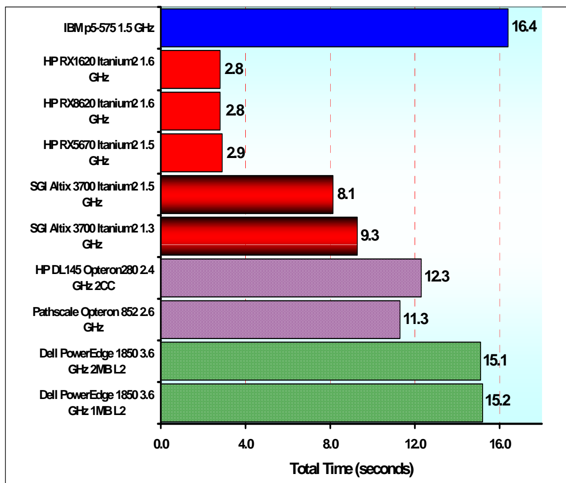


Figure 1: *Linemap* Performance using Dataset (1 20) on a variety of different processors.

The HP-UX Itanium systems ("HP RX" labelled systems in Figure 1) are seen to outperform all the other platforms by a significant margin including other Itanium-based solutions (e.g. the SGI Altix). Thus the HP-UX systems are faster than the SGI systems by a factor of three, and faster than the Opteron-based systems by a factor of over 4. Both EM64T and power5 systems are comparable in performance, slower than the Opteron CPUs by factors of 1.3 and 1.4 respectively.

The reason for this dominance of the HP RX-systems is that the HP-UX compiler provides pragmas in the C/C++ routines for a high performance popcnt / leadz implementation. Pragmas are a method specified by the C standard for providing additional information (machine dependent) to the compiler that means the compiler doesn't call these functions but inlines optimal assembly.

Another contributing factor to the Itanium's performance (applicable to all Itanium solutions) is that it has popcnt implemented in hardware, making it in principle much faster than other comparable CPUs.

There is also some slight dependence on the actual CPU speed, although the cache size has negligible impact on performance. These trends are examined in more detail in Section 4.1. Identical trends are observed when the problem size is increased – see Figures 2 & 3.

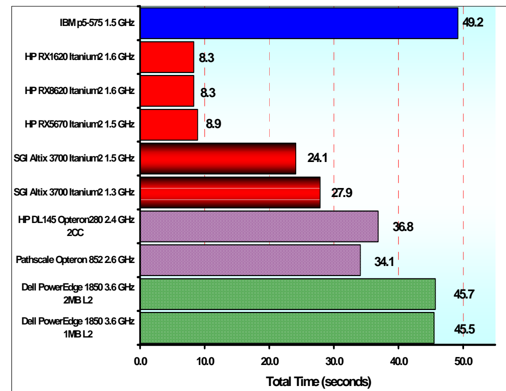


Figure 2: *Linemap* Performance using Dataset (1 21) on a variety of different processors.

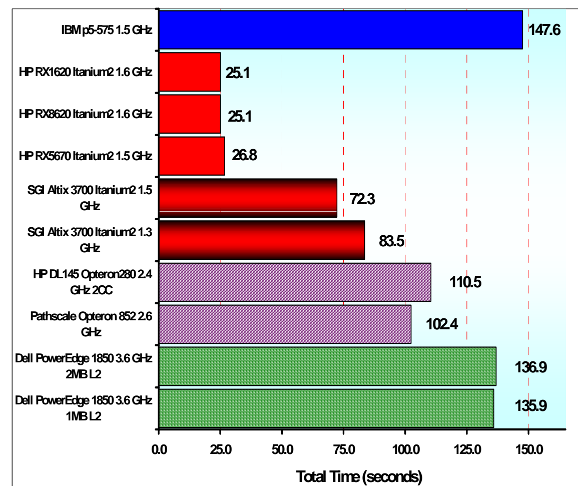


Figure 3: *Linemap* Performance using Dataset (1 22) on a variety of different processors.

3.2 Permutit

Permutit permutes the order of bits of each entry in an array comprising 24×10^6 words. In contrast to *Linemap*, there is little to choose between the performance of the four classes of system, with the Opteron and EM64T systems marginally faster than those featuring Itanium2 and power5 CPUs (see Figure 4). Thus the HP Itanium2 RX-systems are now comparable in performance to the corresponding SGI systems, although surprisingly the RX8620 system is seen to be much slower than both the

[1] A Gray-code, developed by Frank Gray, is a binary numeral system where two successive values differ in only one digit. Gray codes were originally designed to prevent spurious outputs from electromechanical switches. Today they are widely used to facilitate error correction in digital communications such as in digital terrestrial television.

RX1620 and RX5670 – by a factor of two. Initially understanding this performance differential gave cause for concern; we shall return to this point in Section 4.

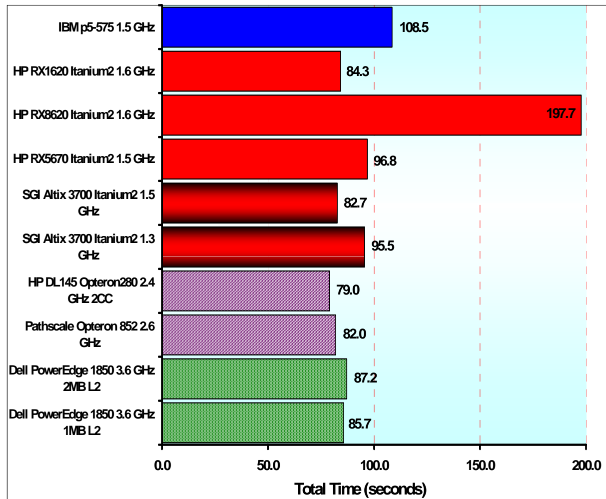


Figure 4: *Permutit* Performance comparison between IA64 and x86-64 architectures.

3.3 Hadamard

This code performs a generalised class of Fourier Transforms, similar to the *Hadamard* transform (for a description of the transform see text box at the end of this section). The two data sets invoked each specify two arguments – the number of bits involved in each transform – 24 in the first, 25 in the second – and the number of transforms to be carried out, 40 and 50 in the two data sets respectively.

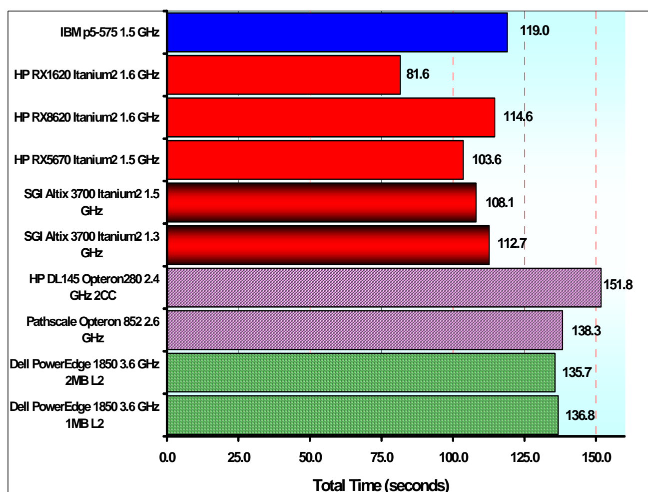


Figure 5: *Hadamard* transformation using the (24, 40) dataset.

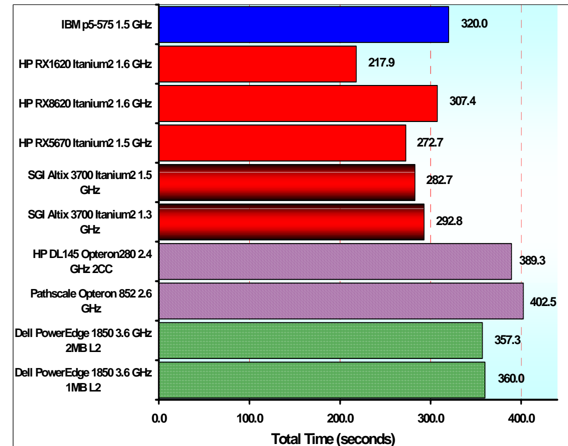


Figure 6: *Hadamard* Transformation using the (25, 50) dataset.

Figures 5 and 6 suggest that the Itanium2, at least in the HP RX1620, is again showing a slight performance advantage over the x86-64 systems when computing the Transformation. The p5-575 performs on a par with RX8620, while the EM64T and Opteron CPUs are comparable, albeit somewhat slower than both power5 and Itanium2 CPUs.

The impact of increasing the problem size is negligible

Hadamard Transform.
The Hadamard Transform (Hadamard transformation also known as the Walsh-Hadamard transformation) is an example of a generalised class of Fourier Transforms. In quantum information processing the Hadamard transformation, more often called the Hadamard gate in this context (cf. quantum gate), is a one-qubit rotation, mapping the qubit-basis states |0› and |1› to two superposition states with equal weight of the computational basis states |0› and |1›. Usually the phases are chosen so that we have

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}}\langle 0| + \frac{|0\rangle - |1\rangle}{\sqrt{2}}\langle 1|$$

in Dirac notation. This corresponds to the transformation matrix

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

in the |0›,|1› basis.
Many quantum algorithms use the Hadamard transform as an initial step, since it maps *n* qubits initialised with |0› to a superposition of all 2^n orthogonal states in the |0›,|1› basis with equal weight.
The Hadamard matrix can also be regarded as the Fourier transform on the two-element *additive* group of **Z**(2).
The Hadamard transform is used in many signal processing, and data compression algorithms.

on the performance differential between the architectures; the performance appears to scale linearly with problem size. A more detailed analysis of *Hadamard*'s dependencies on the system characteristics is presented in Section 4.3.

3.4 *RuWarray*

RuWarray calculates a read-update-write on random elements in an array.

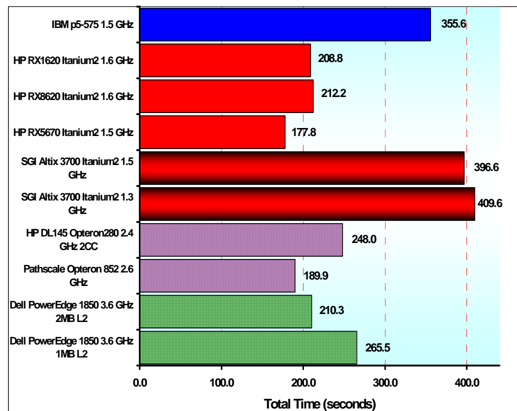


Figure 7: *RuWarray* performance using 3 × 10^9 random elements.

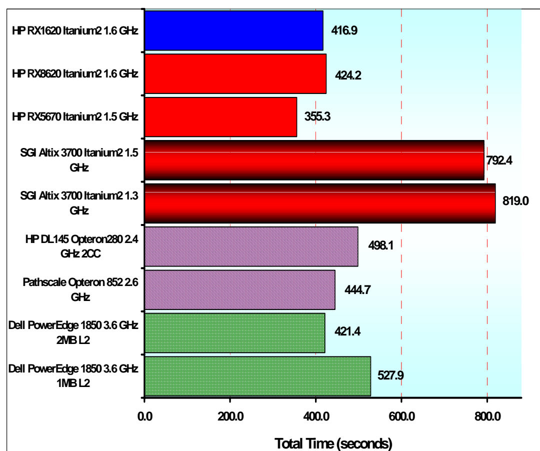


Figure 8: *RuWarray* performance using 6 × 10^9 random elements.

The performance of *RuWarray* for an increasing number of elements, shown in Figures 7-9, is comparable on HP's IA64 and the x86-64 platforms, with the HP RX5670 the fastest system. However the SGI Altix's Itanium implementation is surprisingly much slower. The cause, we suspect, is the use of the default pre-processing invoking the slow implementation branch. As with the *Hadamard* benchmark, performance trends are consistent across the various problem sizes. Increasing or decreasing the number of random elements in the problem set does not provide a specific chipset with a performance edge over the other systems.

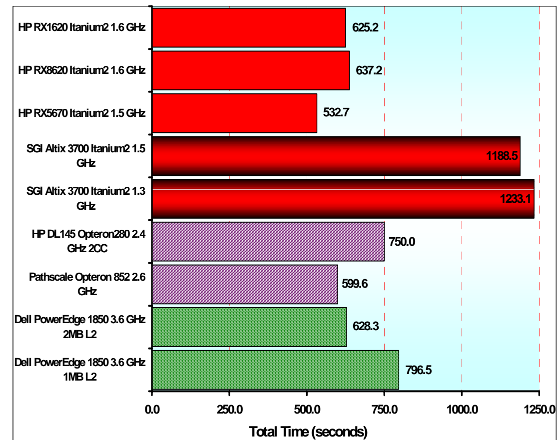


Figure 9: *RuWarray* performance using 9 × 10^9 random elements.

3.5 *Treesearch*

Treesearch, as the name implies, involves the use of a *Treesearch* algorithm to solve a 33-peg solitaire game.

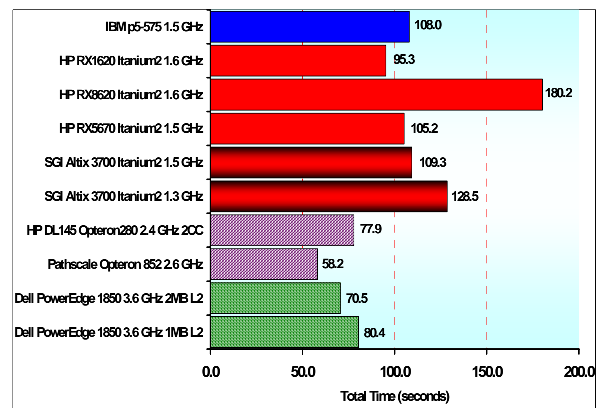


Figure 10: *Treesearch* serial performance on both IA64 and x86-64 architectures.

For *Treesearch* all the x86_64 architectures outperform the IA64 systems, with the HP RX8620 significantly slower than the other systems. The Opteron and EM64T systems exhibit comparable performance, with the 2.6GHz Opteron252 the fastest CPU. The performance dependencies of *Treesearch* are examined in more depth in Section 3.5, where both the CPU speed and memory latency prove to be important factors in governing the performance of the code.

3.6 Linequ

Linequ solves a system of linear binary equations using 700 equations with 700 unknowns. The equations are solved using Gaussian elimination with block reductions and partial pivoting. Extensive use is made of both the popcnt and leadz intrinsics.

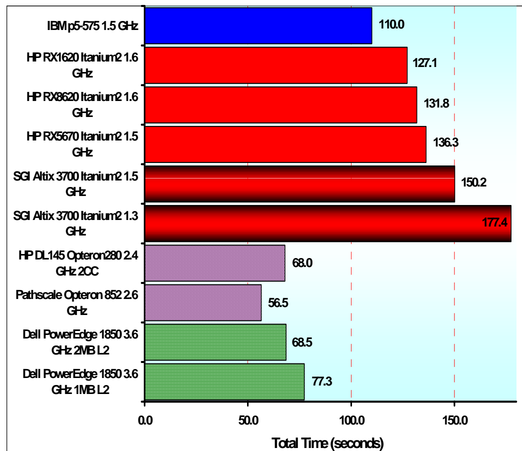


Figure 11: *Linequ* serial performance on both IA64 and x86-64 architectures.

As with *Treesearch*, the IA64 systems are seen to be significantly slower than the x86-64 systems. The problem would appear to lie in the integer multiplication routine, where the Itanium systems spend excessive time calculating loop addresses. A typical loop in *Linequ* is of the form:

```
SUBROUTINE REDUCE (SETNUM, MSIZE, NW,
MAXM, B, I, R, RR, EQN, KARY, IS)

PARAMETER (ONE=1)
INTEGER*8 R(0:NW-1,0:2**B-1)
INTEGER*8 EQN(0:NW-1,O:MSIZE-1)
INTEGER*8 KARY(O:MSIZE-1)

KK=ISHFT(I+IS,-6)

DO J = 0,I-1
     ENT = KARY(J)
     DO  MM = KK,NW-1
       EQN(MM,J)=IEOR(EQN(MM,J),R(MM,ENT))
     END DO
END DO
```

Matrices with two dimensions are used, where the array boundaries are passed by the caller, and are unknown at compile time. If the inner MM-loop is very short, most of the time will be spent calculating the addresses of R(MM,ENT). The x86-64 has the advantage over IA64 in that it can perform the integer multiplication in hardware, while the IA64 systems must convert the integer to floating point, perform the multiplication and then re-convert to integer format (analogous to the process deployed on PA-RISC architectures). Profiling studies suggest that these instructions are indeed using a significant proportion of the time in *Linequ*, thus explaining the performance lead of both the Intel Xeon EM64T and AMD Opteron (x86-64) systems. The 2.6GHz Opteron252 is again the fastest CPU.

3.7 IObench

IObench writes a number of 64-bit words of bitstreams to a file and reads the file. Two data sets are used, one with 10^8 words (Figure 12), the second with 10^9 (Figure 13).

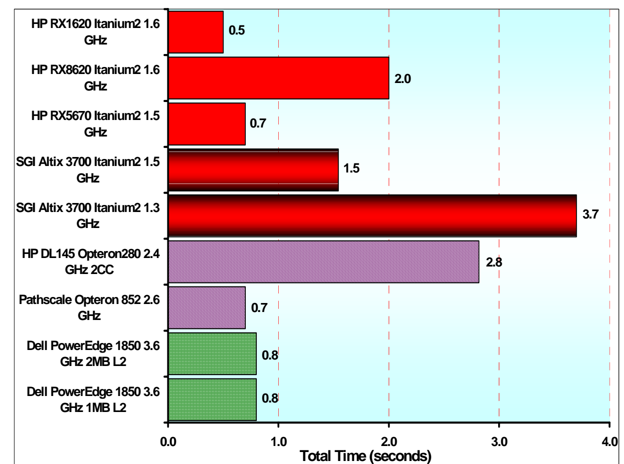


Figure 12: *IObench* with 10^8 64-bit words of bitstreams.

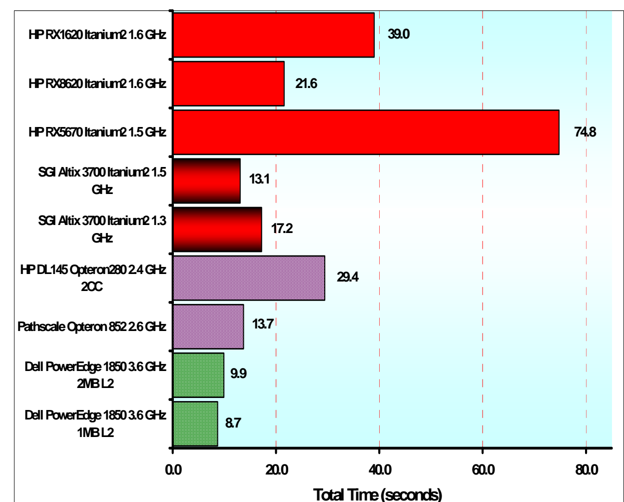


Figure 13: *IObench* with 10^9 64-bit words of bitstreams.

This benchmark is, as the name suggests, Input/Output (I/O) intensive, and is critically dependent on the disk subsystem in use. In terms of understanding the HP IA64 performance of Figures 12 and 13, only a single disk was available on the RX1620 and RX5670 systems, whilst the RX8620 had 3 striped disks. I/O is actually buffer cached in this benchmark, and should be explicitly opened with osync or a flush call added. Note that this benchmark is really dated, and should be replaced by e.g. IOzone, now the I/O standard benchmark of choice [8].

4. Serial Performance: Code Dependencies

This section looks to understand in more detail the performance of each of the benchmarking codes presented in Section 3, though an examination of performance dependency on the various aspects of the system architecture, such as CPU Speed, L2 and L3 cache effects and memory bandwidth/latency. This analysis was conducted by gaining access to a set of Dell PowerEdge 1850 servers with varying clock speeds and L2 cache sizes.

The Dell PowerEdge systems use Intel Xeon EM64T processors with the codenames “Nocona” containing 1MB L2 cache and “Irwindale” that has 2MB L2 cache. Table 3 summaries the CPU speeds of the systems used in this experiment.

Nocona (1MB/L2)	2.8	X	3.2	3.4	3.6
Irwindale (2MB/L2)	X	3.0	3.2	3.4	3.6

Table 3: Clock Speed Summary (GHz) for the Dell PowerEdge 1850 systems.

Using related systems in this fashion reduces the number of variables that might be influencing the observed performance, enabling an additional degree of confidence in determining the underlying cause of a particular trend.

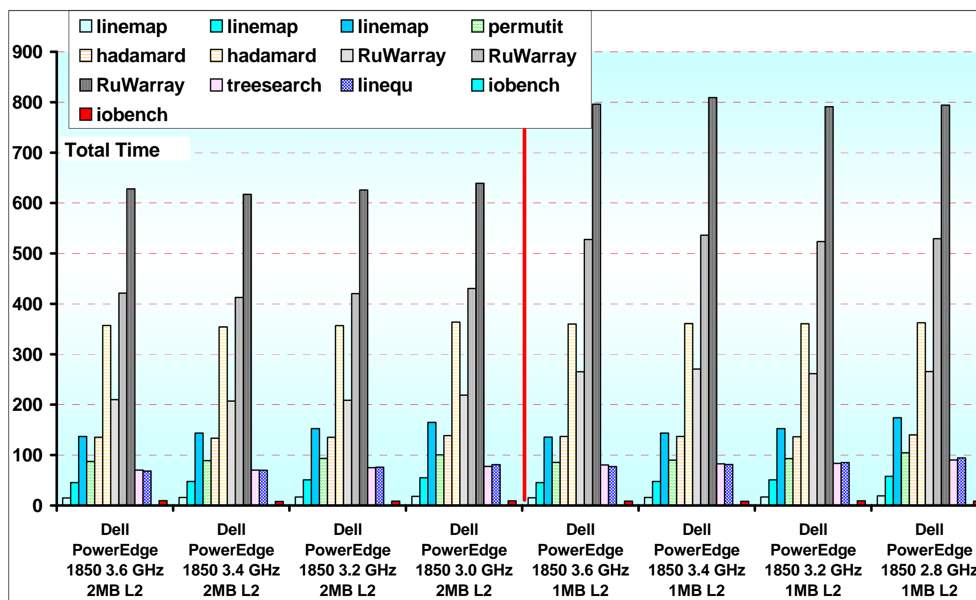


Figure 14: Summary of all HPC integer benchmarks on Dell PowerEdge 1850 systems

Figure 14 provides an overall summary of all 7 codes on Intel Xeon EM64T systems. To the left of the red central vertical line are all PowerEdge systems with 2MB L2 cache, in increasing clock speed, starting at the red line and working outwards to the left margin. To the right of the red line are all the Dell PowerEdge systems with 1MB L2 cache in decreasing clock speed working from the red line to the right hand side of the page.

Each of the codes is examined individually in the sections below. Figure 14 attempts to summarise the overall effects on all the codes. The following layout of the systems above is consistent in all subsequent graphs in this section:

← INCREASING CPU SPEED | DECREASING CPU SPEED →
2MB L2 CACHE | 1MB L2 CACHE

4.1 *Linemap* – Impact of Clock Speed & Cache

Figure 15 suggests that *Linemap* is dependent primarily on the CPU speed and that L2 cache has a negligible impact on performance. Concentrating on one half of the graph shows that performance improves with increasing clock speed. Comparing corresponding systems on either side of the line, we find almost identical performance, indicating that increasing L2 cache has no impact on the performance of the code i.e. the benchmark is

- CPU dependent
- Insensitive to L2 cache

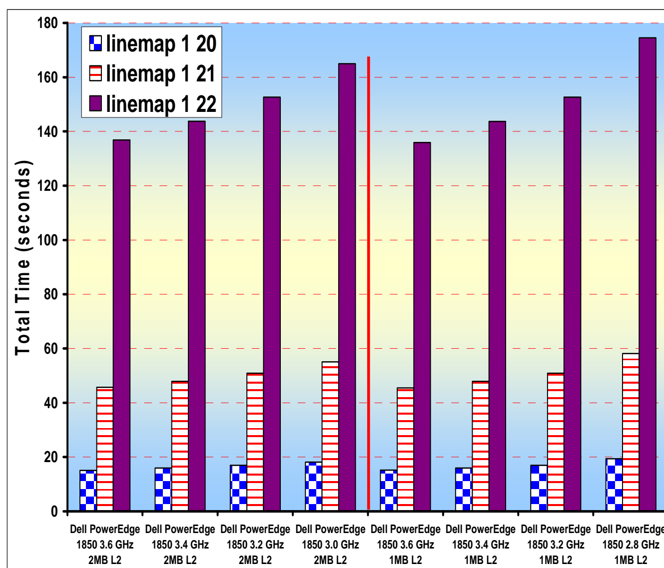


Figure 15: *Linemap* Performance on a range of Intel EM64T CPUs.

These trends are consistent across all 3 *Linemap* data sets (although these effects are not as obvious for the two smaller data sets as the performance is dominated by the largest data set total time, skewing somewhat the results for the 1,20 and 1,21 datasets).

4.2 *Permutit*: Impact of Clock Speed & Cache

Figure 16 suggests that *Permutit* demonstrates similar dependencies as *Linemap*. Time to solution decreases as CPU speed increases, but is insensitive to increasing the amount of L2 cache (comparing corresponding systems on either side of the line) i.e. the benchmark is

- CPU dependent
- Insensitive to L2 cache

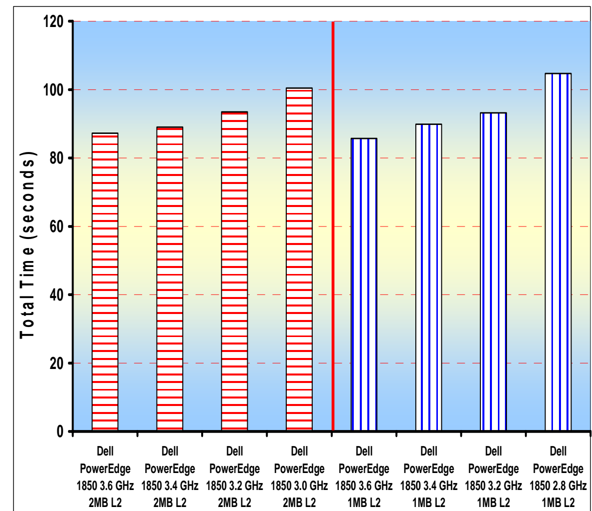


Figure 16: *Permutit* Performance on a range of Intel EM64T CPUs.

4.3 *Hadamard* – Impact of Clock Speed & Cache

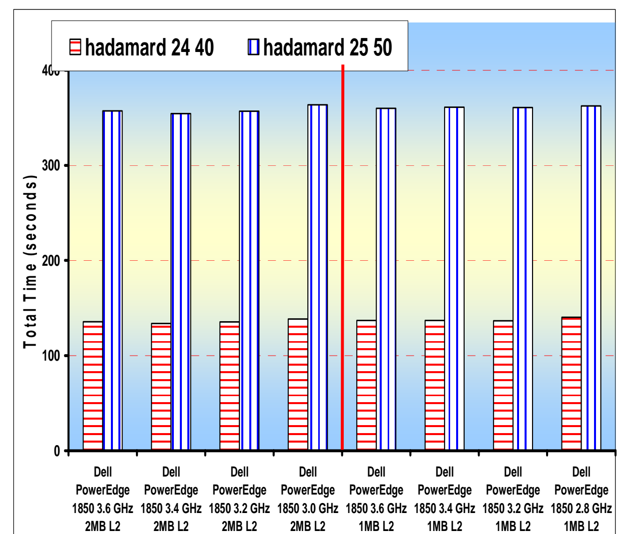


Figure 17: *Hadamard* Performance on a range of Intel EM64T CPUs.

Figure 17 strongly suggests that neither CPU speed nor L2 cache size has any influence on calculating a *Hadamard* transformation, as indicated by the uniform behaviour over all the systems i.e. the benchmark is

- Insensitive to CPU speed
- Insensitive to L2 cache

The insensitivity of a code to both CPU frequency and cache is a strong indicator that either I/O or memory bandwidth/latency is the performance bottleneck. Given that there is virtually no I/O from *Hadamard*, we can be

certain that the code is 100% memory bandwidth/latency bound.

4.4 *RuWarray* – Impact of Clock Speed & Cache

Increasing CPU speed has minimal impact on the performance of *RuWarray* (speed of performing a read-update-write on random elements of a large array). In contrast, increasing the amount of L2 cache available improves the codes performance, demonstrated by the shorter time requirements for systems on the left hand side of Figure 18 i.e. the benchmark is

- Insensitive to CPU speed
- L2 cache sensitive

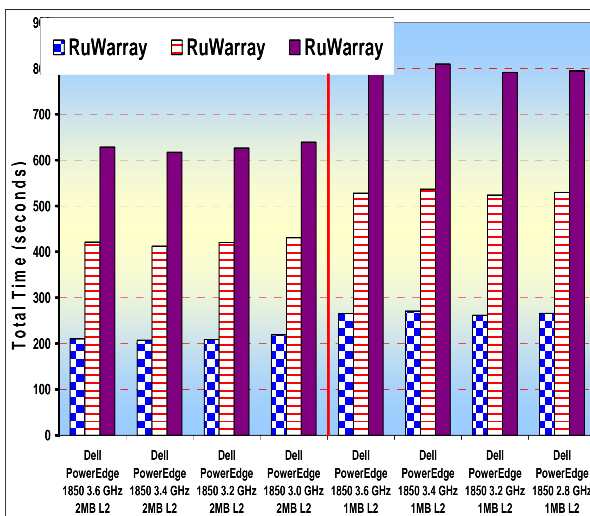


Figure 18: *RuWarray* Performance on a range of Intel EM64T CPUs.

4.5 *Treesearch* – Impact of Clock Speed & Cache

Treesearch has joint dependencies on both increasing CPU speed and level of L2 cache, as demonstrated not only by the increased performance of systems on either side of the "dividing" line (L2 cache), but also the improved performance when increasing clock speed for systems with the same L2 cache (comparing either the horizontal or vertical striped systems) in Figure 19 i.e. the benchmark is.

- Sensitive to CPU speed
- L2 Cache sensitive.

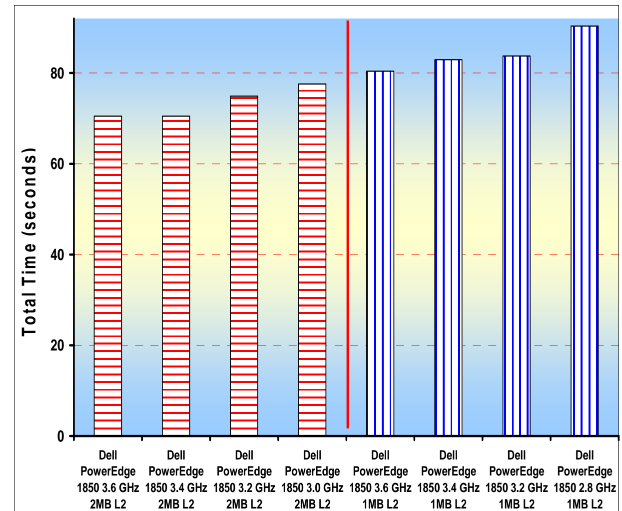


Figure 19: *Treesearch* performance on range of Intel EM64T CPUs.

4.6 *Linequ* – Impact of Clock Speed & Cache

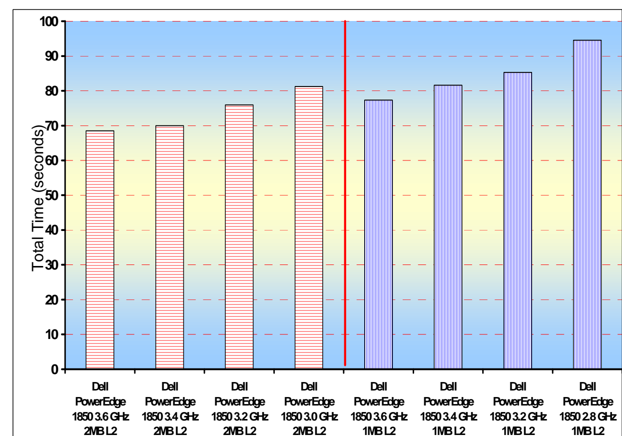


Figure 20: *Linequ* performance on a range of Intel EM64T CPUs.

As with *Treesearch*, Figure 20 shows that *Linequ* is sensitive to both CPU speed and the level of L2 cache i.e. the benchmark is

- Sensitive to CPU speed
- L2 cache sensitive

4.7 Impact of the Memory Subsystem.

In order to keep abreast of the advances in technology, the Distributed Computing group relies heavily on gaining access to a variety of systems through fostering existing relationships with various Tier1 and Cluster Integrator companies. As part of the integer benchmarking exercise, Hewlett Packard made available a variety of Itanium systems (using the HP-UX operating

system). We here take a closer look at some of the performance attributes of these systems, and compare these with the latest HP Itanium2 product on the market (the Montecito SuperDome SD64000B), which at the time of writing was in pre-production release, but is now commercially available (from February 2006).

Table 4 provides a reminder of the HP-UX Integrity systems used in the benchmarking report.

System	CPU Speed	Itanium Platform	L3 Cache (MB)	FSB (MHz)
rx5670	4×1.5GHz	Madison	6	400
rx8620	8×1.6GHz	Madison	6	400
rx1620	2×1.6GHz	Madison	3	533
SD64000B	1.6GHz	Montecito	12	533

Table 4: Characteristics of the HP Itanium2 systems (HP-UX).

These four systems have different clock speeds, cache sizes and front side bus (FSB) speeds. Whilst the cache has been demonstrated to be important in the above analysis, this is not always the major differentiator between these systems, as even the lowest L3 cache in Table 4 is larger than the L2 cache supplied with most of the x86-64 systems. One of the most important differences between these systems is the memory latency (primarily representative of a cache coherency effect). Table 5 gives an overview of the memory latencies of these four systems as well as the standard latency for both the AMD Opteron and SGI Altix. Also included are detailed memory latencies to main memory measured by S. R. Alam and co-workers [9].

System	Memory Latency (ns)
HP Integrity rx1620	110-120
HP Integrity rx5670	140
HP Integrity rx8620	280 [1], 400 [2]
HP Integrity SD64000B	180 [1], 350 [2]
Opteron (averaged)	75[3], 140[4]
Cray XT3 / Opteron 150 / 2.4 GHz [9]	51.4
Cray XD1 / Opteron 248 2.2 GHz [9]	86.5
Intel Xeon 3.0 GHz [9]	140.6
IBM p690 POWER4 1.3 GHz [9]	90.6
SGI Altix 3700	200-250

Table 5: Summary of the memory latency (in nanoseconds). [1]local to cell access; [2]access to memory on remote cell; [3]local access; [4]access to memory on remote CPU.

Table 5 shows there is a significant variation in the memory latencies between the HP Itanium solutions,; with the exception of the SD64000B, none of these latencies are particularly impressive compared with some of the current systems available on the market today. This stems from the fact that the HP cell-based solutions, in for example the rx8620, are reliant on the same cell infrastructure technology that was developed 5 years ago around the PA-RISC superdome. Extensive investment in redeveloping this infrastructure for the latest cell-based products means the latest systems e.g. the SD64000B, exhibit reduced latency, from 280ns to around 180ns (using the SX2000 chipset).

In many scientific benchmarks [10], where a significant proportion of the time is spent in performing floating point arithmetic, the rx8620 is between 20 – 40% slower than the rx1620. In many of these 64-bit scientific codes, the effect of the high memory latency is circumvented by accessing the memory in a predictable manner (for example unit stride, serial access). This involves the compiler employing memory pre-fetching and speculative loads to counteract the memory latency effects. However the codes used in the present Integer benchmarking suite are quite different from the 'usual' floating point scientific applications.

Given the cell developments above, the runtimes on the new Integrity "Montecito" SuperDome (SD64000B) should be much closer to the rx1620 times, thus highlighting the impact of the high memory latency in the current rx8620 system on the Integer benchmarks.

System	CPU & CPU Speed	Memory Latency (ns)	L3 Cache (MB)	FSB (MHz)
SD64000B	Itanium2 Montecito 1.6GHz	180	12 (per core)	533

Table 6: System specification of the new HP Integrity Superdome using the SX2000 chipset.

The above expectations were confirmed with access to HP's flagship system, a prototype Montecito Integrity Superdome SD64000B (64 socket / 128 core system) employing the latest cell infrastructure (the SX2000). The specifications of this system are provided in Table 6.

Case	Program	Time (seconds)	
		rx8620	**SD 64000B**
1	*Linemap*	2.8	2.8
2	*Linemap*	8.3	8.3
3	*Linemap*	25.1	25.0
4	*Permutit*	197.7	**80.1**

5	*Hadamard*	114.6	**75.5**
6	*Hadamard*	307.4	**195.6**
7	*RuWarray*	212.2	**131.7**
8	*RuWarray*	424.2	**262.8**
9	*RuWarray*	637.2	**394.3**
10	*Treesearch*	180.2	**100.1**
11	*Linequ*	131.8	126.6
12	*IObench*	2.0	0.7
13	*IObench*	74.8	**6.8**

Table 7: Summary of the Integer Benchmarking times (in seconds) for the rx8620 and the HP Integrity Superdome SD64000B.

This new machine (SD64000B) provides a real test of the integer code performance in terms of clock speed, memory latency and bandwidth. Table 7 gives the total run times for each of the integer benchmarks, on both the older rx8620 system and on the Montecito Superdome.

A comparison of these timings allows the following conclusions to be drawn:

- *Linemap* performs almost identically on all Itanium platforms. Neither latency nor cache has any measurable impact on performance.
- On the rx8620, *Permutit* gave poor performance. With the latest SX2000 chipset the code demonstrates a two fold improvement in performance, even out-performing the rx1620 (the lowest latency rx system).
- *Hadamard*. This code is heavily latency-bound, as demonstrated by the results in Table 7. There is a 30% improvement in the run times when moving to the lower latency Superdome.
- *RuWarray*: This code gave unexpected runtime figures (see section 4), where the rx5670 out performed the other Itanium systems, even though it had a slower clock speed and slower bus. This was attributed to the larger cache on the rx5670. The timings from SD64000B confirm this, as the Montecito SuperDome is much faster than the older Madison platforms. This was also demonstrated by the Dell PowerEdge analysis in Section 4.4.
- *Treesearch* would appear to be heavily latency bound. Section 4.5 demonstrated the dependence on both CPU speed and cache. The 44% improvement in run time using the SX2000 chipset confirms the cache / latency discussion of Section 4.5.
- *Linequ*. Whilst Section 4.6 shows the dependence of the code with respect to both CPU speed and L2 cache (1MB /2MB), it would appear that there is a ‘saturation’ point beyond which increasing the amount of cache has little impact on performance. This is shown in Table 7; although the cache is doubled compared to the rx8620 (12MB), the performance gain is minimal.
- *IObench* is much faster on the SD64000B. This is because of a large buffer cache (15GB) and a fast striped filesystem on the HP Superdome.

4.8 Performance Sensitivity Summary

Table 8 highlights the conclusions from Sections 3 and 4, summarising the performance attributes displayed by each of the integer benchmarking codes and which processor family displays optimum performance.

Code	Optimum CPU	Performance Sensitivity Analysis			
		CPU (GHz)	L2 cache	Mem. (MB/s)	Mem. (µs)
Linemap	Itanium	✓	X	X	X
Permutit	Opteron	✓	X	✓	✓
Hadamard	Itanium	X	X	✓	X
RuWarray	Itanium	X	✓	✓	X
Treesearch	Opteron	✓	✓	X	✓
Linequ	Opteron	✓	✓	X	X

Table 8: Summary of the Performance Trends from Sections 3 and 4 for the HPC Integer Benchmarking Suite.

It should be noted that the same performance attributes noted in the sections above were observed when performing the corresponding calculations on AMD Opteron systems.

5. Enhancing Performance

This section focuses on two important considerations when looking to enhance performance of the Integer benchmarking suite. First, we investigate the impact of different compiler optimisation levels on runtime performance for each of the codes. Secondly, we consider the impact of coding language; for *Hadamard*, there are two versions of the code currently available, one in FORTRAN, the second in C. By default the former version is typically used, for the C implementation was found historically to be significantly slower than the FORTRAN code, particularly on vector

machines, and has been largely ignored. Both versions of this code have now been built and the timings and output compared on a variety of systems.

5.1 Compiler Optimisation Levels

In order to determine the sensitivity of the Integer codes to the level of compiler optimisation, a single platform was taken (the 2.6 GHz AMD Opteron 852) and, using one of the latest compiler suites – Pathscale's EKO version 2.2.1, executables generated using different build options. Figure 21 captures the results from this exercise. Three different optimisation levels were invoked:

- -O2 (generates an optimised executable that is numerically safe – this is used by default);
- -O3 (generates a highly optimised executable, generally numerically safe);
- -O3 –ipa –OPT:OFAST (optimisations selected to maximise performance). Although these optimisations are generally safe, they may affect the achieved accuracy given the inevitable rearrangement of computations.

Figure 21 depicts the benchmark results normalised with respect to the performance of the codes built using the –O2 optimisation level.

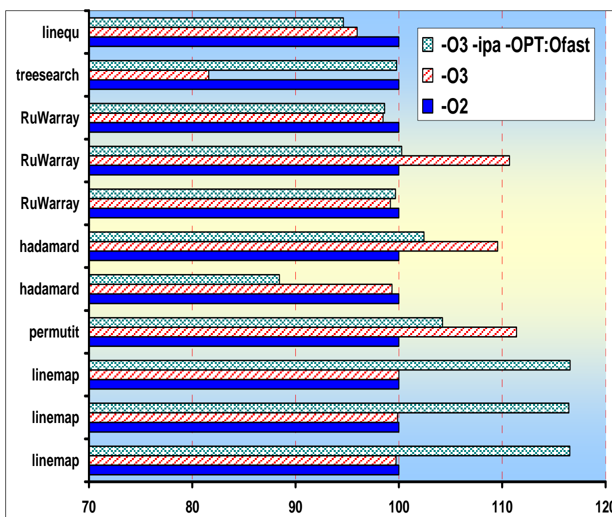


Figure 21: Effect of Different Compiler Optimisation Levels on the Performance of the Integer benchmarking Codes.

The impact of optimisation is seen to be highly dependent on the code in question, with some (e.g. *Treesearch*) showing much greater sensitivity than others. In the case of *Hadamard*, and to a lesser extent *RuWarray*, the impact on performance is also heavily dependent on the size of the input dataset. Interestingly, the higher optimisation level only benefits the smaller test case – indeed for the larger case it actually has a detrimental effect, causing longer run times.

In contrast, the size of dataset has no bearing on the relative performance of *Linemap* as a function of optimisation level. In fact *Linemap* appears to be the only code to show a marked improvement in performance at the highest optimisation level using the PathScale compiler. Of the six codes tested, four – *RuWarray, Hadamard*, *Permutit* and *Linemap* – show enhanced performance when progressing beyond the default –O2.

This exercise suggests, perhaps not surprisingly, that the impact of compiler optimisation on performance is dependent on the nature of the code and the subsequent datasets. It certainly shows that certain codes are more susceptible to optimisation than others and it is clearly important to understand the performance implications this might have. In general –O2 provides a robust optimisation that gives reasonably good performance for the majority of application codes.

5.2 *Hadamard* – C and FORTRAN codes

In the HPC Integer benchmarking suite there are two serial versions of the *Hadamard* code. One is written in FORTRAN and is the default build; the second, a little used C version. Prior to generating these two *Hadamard* executables, it was widely believed that the FORTRAN compiled version of the transform would out-perform the corresponding C version. This understanding was largely based on historical data relating to results on legacy vector systems.

Whilst the FORTRAN version of the code compiled without any modifications, it was necessary to comment out the "include <malloc.h>" line in the C code. Without this change, the code core-dumps in 64-bit mode because the malloc routine is unknown and therefore is assumed to generate a 32-bit return value. This is a common programmer's bug for 64-bit codes. Once the change had been made, the code compiled and executed with no further errors. The subsequent output files for both the C and FORTRAN executables verified correctly. Figures 22 and 23 provide the timings for both versions of the *Hadamard* executable.

These two figures clearly show that on all the systems used, the C version of the code outperforms the FORTRAN build. This is contrary to the historical understanding of the performance of the C code. The difference between the two executables is most pronounced on the x86-64 architecture, indicating a three to 4 fold increase in performance over the corresponding FORTRAN version.

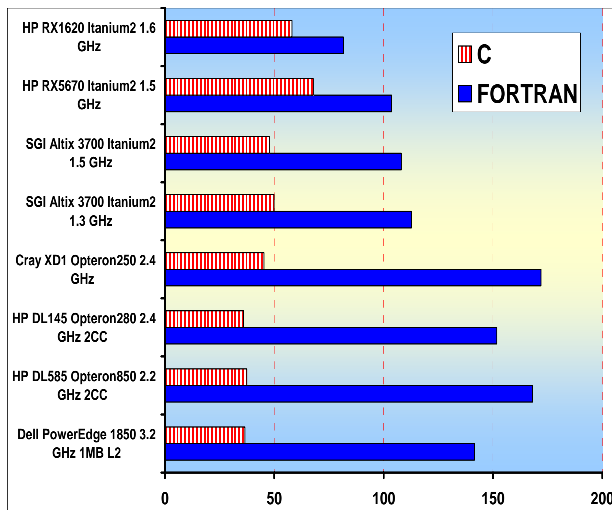
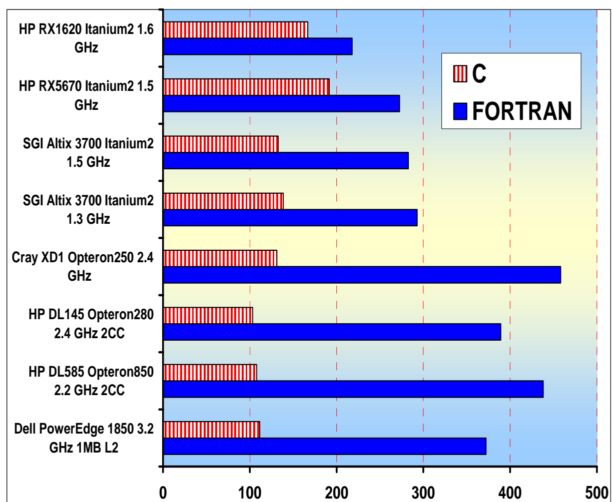


Figure 22: Execution time (seconds) using the C and FORTRAN builds of the *Hadamard* code. The (24,40) data set.



The Figure 23: Execution time (seconds) using the C and FORTRAN builds of the *Hadamard* code. The (25,50) data set.

HP-rx systems reveal the closest performance, but even here there is significant performance gain using the C compiled executable. This effect clearly reflects the advances in optimisation that have accompanied generations of development to C compilers, plus the major differential that FORTRAN enjoyed on the vector-based architectures of the 1980's and 90's.

6. Integer RATE Benchmarks.

Whilst understanding the serial, single processor performance of code remains important, the significance of benchmarks such as SPECint2000 has become more debatable given the advent of dual-core (and multi-core) systems. In order to understand the impact of utilising all the processing elements on a multi-core system, the throughput RATE-style benchmarks seem more attractive e.g. SPECint_rate2000 [4]. In this spirit, the HPC Integer benchmark has been extended to include an associated "RATE" Benchmark. This benchmark incorporates six of the present integer codes and eleven data sets; note that *IObench* was not included in this benchmark. There are two rate procedures depending on whether the aim is to interpret the performance of a single machine with 'n' CPUs, or if it is to compare different multi processor or multi-core systems.

1) Single Server with n CPUs

When comparing the RATE runs on n CPUs or n PEs, the rate for a given benchmark code is:

$$R_n = n \times (T_1 / T_n)$$

where T_1 is the elapsed time taken for a single processor run (normalised to a time of 100 units) and T_n the elapsed time taken for n copies of the code, one on each CPU, to complete (where elapsed time is given by the time of the last job to finish – time of first to start). The normalization above makes it easier to picture the performance impact when adding additional processors and to highlight any degradation in performance. If there is no performance degradation when using multiple cores, then for a dual core, dual processor system (4 processing elements, PEs), using 1 PE the rate, R_1, will be 100, using 2 PEs (one on each core) will yield a rate, R_2, of 200 and running on all 4 PEs would provide a rate, R_4,of 400.

In practice this is not usually found given the inevitable bottlenecks occurring when multiple processors are used, e.g., memory bandwidth issues due to, say, two processors requiring the same front side bus (FSB) to access main memory. The extent to which this affects the performance of the codes is heavily dependant on (i) the architectural design of the system in question, and (ii) the code itself, and whether it has extensive memory bandwidth requirements. We examine these effects in more detail in this section.

2) RATE-based Machine Comparisons

With a multi component benchmark, then for a given benchmark, 'i', and assuming a system with n PEs, we need to run 'n' instances of the benchmark code simultaneously and calculate the elapsed time.

$$elapsed\ time = time\ of\ last\ to\ finish - first\ to\ start\ time$$

The rate for benchmark i, *Ri*, is then calculated using:

$$Ri = n \times (Tref / Ti)$$

where T_{ref} is the elapsed time on a predefined arbitrary reference system scaled to a single processor (n=1); note that Tref is now normalised to an elapsed time of 100 units. Again, the latter normalization makes it easier to

picture the performance impact when adding additional processors and to highlight any degradation in performance.

An important note when running the rate benchmark is to use the taskset[2] comment. This is typically used to prevent process migration (job moving to another processor) during the benchmark and binds the process to a specific processing element. Taskset takes the form:

taskset –pc [cpu] X

where *[cpu]* specifies the PE in question, and *X* the running process or task.

Three systems are examined in this section:

1. A Dell Poweredge 1850 node, comprising "Nocona" EM64T dual processors with 1MB L2 cache.

2. An Opteron-based Supermicro node comprising 2 x 2.2 GHz dual-core AMD Opteron 275 processors.

3. A prototype of the Intel Xeon 5080 processor - the EM64T "Bensley/Dempsey" platform - clocked at 3.46 GHz. The prototype PowerEdge 1950 node featured 4 cores, 2 chips, and 2 cores / chip.

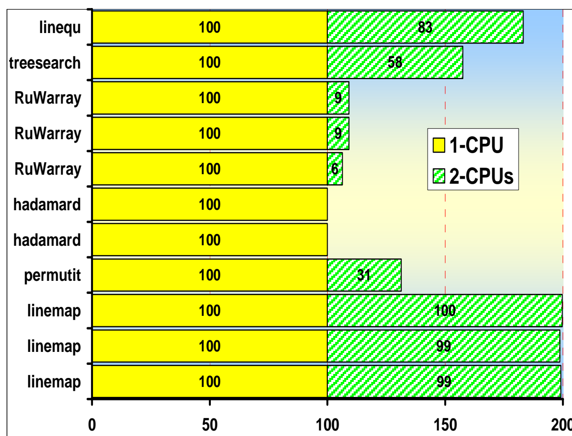


Figure 24: HPC Integer Rate Benchmark on a Dell Poweredge 1850 system with dual EM64T 3.2GHz processors, normalised with respect to the performance of a single processor (100%).

Figure 24 shows the rate benchmark results on the dual processor Dell Poweredge node. Thus for *Linemap* we find a linear increase in the benchmark when using both processors (i.e. a RATE figure of 100 + 100). The performance of the *Linequ* code also appears reasonable on this system. However the performance collapses dramatically for *Hadamard*, *RuWarray* and *Permutit* when both CPUs are in use. This confirms the strong dependency on memory bandwidth (MB/s) of these codes, as noted previously (see Table 4 in Section 4.7). Given that both processors share the same FSB to access memory, thus severely restricting available memory bandwidth, the observed impact on the performance of these codes, and the subsequent dramatic collapse in performance, is not surprising. Where there is very little dependence on memory bandwidth, the codes demonstrate a linear performance increase.

Figure 25 demonstrates the rate benchmark on the dual core Opteron275 system. In order to reflect the dual core architecture, we now shift nomenclature from CPU to PE (Processing Elements). In this system, we see that there is no major performance degradation when going from one to two PEs, for each RATE job will be allocated to a core on a distinct processor with dedicated FSB access to memory. There is, however a major performance degradation when all 4 PEs are utilised and, as with the Intel Xeon EM64T above, the codes that are dominated by memory bandwidth requirements show no additional performance when run on all 4 PEs. Codes such as *Linemap* with no real memory bandwidth requirements still demonstrate a linear increase in performance

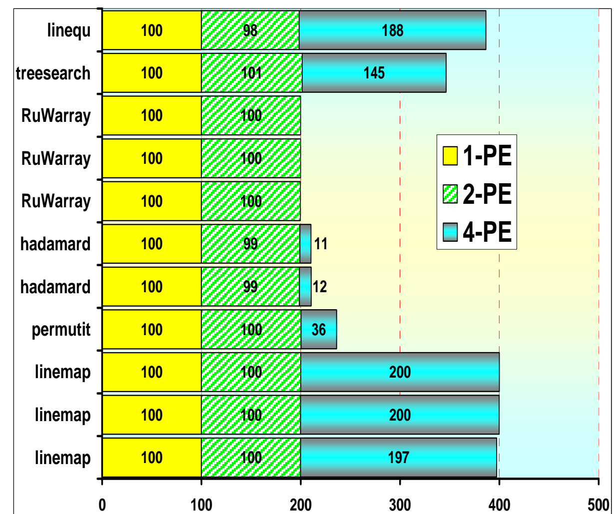


Figure 25: HPC Integer Rate Benchmark on an AMD Opteron 275/ 2.2GHz system, normalised with respect to the performance of a single processor (100%).

Finally, Figure 26 shows the dual core Xeon 5080 processor performance. Whilst this represents a major step forward by Intel compared to their original dual core systems (codenamed Paxville / Irwindale), where all four cores of the dual processor system shared the same FSB, the Bensley platform has each processor with its own unique path to memory, plus other performance enhancements e.g., fully buffered DIMMS.

[2] Taskset binds a process to a given set of CPUs on the system, so that the process will not run on any other CPU.

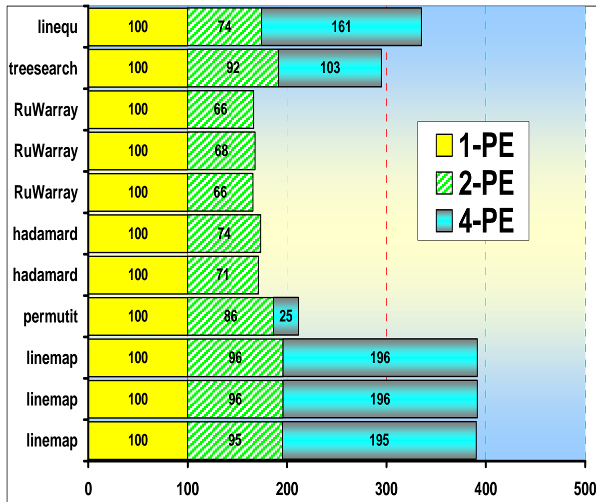


Figure 26: HPC Integer Rate Benchmark on an Intel Xeon 5080 / 3.46GHz system, normalised with respect to single processor performance (100%).

The smaller improvement found when moving from 1 to 2 PEs compared to the Opteron results of Figure 25 does suggest that the 5080 still has a few additional performance issues compared to the dual-core Opteron system. In the case of *RuWarray* this is as much as a 30% decrease in expected performance As with the Opteron systems, a major performance degradation occurs for the *Hadamard*, *RuWarray* and *Permutit* codes when all 4 PEs are invoked, when both cores of each processor will now be sharing the same FSB.

7. Throughput Workload Benchmarks.

All the examples in this report have concentrated on individual code performance – either on a single CPU or single compute node or server. Realistically codes used on a mid-range commodity compute cluster are unlikely to run in such a fashion, with dedicated single code usage and no other users running competing jobs on the system. We try now to simulate these effects using a throughput workload benchmark designed to give an indication of the performance of a system under heavy load. The throughput workload benchmark comprises the same seven programs used in the HPC Integer benchmark. Allowing for the various input job decks (as defined in Section 3) associated with these codes results in a total of 13 different test jobs. To simulate "real world" usage, this benchmark consists of submitting batch jobs in a predefined random sequence using a perl script to the system under evaluation. Each of the batch jobs is an invocation of one of the following 13 test-cases:

- case1 – *Linemap* (1 20)
- case2 – *Linemap* (1 21)
- case3 – *Linemap* (1 22)
- case4 – *Permutit*
- case5 – *Hadamard* (24 40)
- case6 – *Hadamard* (25 50)
- case7 – *RuWarray* ()
- case8 – *RuWarray* ()
- case9 – *RuWarray* ()
- case10 –*Treesearch*
- case11 – *Linequ*
- case12 – *IOBench* ()
- case13 – *IOBench* ()

All the jobs – 765 in total – are submitted to the scheduler with equal priority. Such a job mix should be sufficient to provide a reasonably thorough test of clusters with processor counts in the range of 16 to 64. The performance of the benchmark is calculated by measuring the wall-clock time starting from the submission of job 1 until the final job is completed. Note the final job to complete is not necessarily the same as the last job to start (job 765) – final in this context means the last job to finish.

Two systems have been used to test this throughput workload:

- A 72-node (144 processor) HP rx1620 Itanium2 system, 16GB RAM per node and using an Infiniband Interconnect.
- A 35-node (70 processor) Cray XD1 with AMD Opteron 250/2.4GHz processors, 4GB RAM per node, connected using Cray's Rapid Array fabric.

Case	Programme	Number of instances	Summed Time (seconds)
1	*Linemap*	60	168
2	*Linemap*	15	125
3	*Linemap*	5	125
4	*Permutit*	5	422
5	*Hadamard*	400	32640
6	*Hadamard*	100	21790
7	*RuWarray*	20	4176
8	*RuWarray*	10	4169
9	*RuWarray*	5	3126
10	*Treesearch*	10	953
11	*Linequ*	75	9533
12	*IObench*	40	20
13	*IObench*	5	195
	TOTAL	765	77350 (1290 minutes)

Table 9: Overview of the 765 jobmix for the HP rx1620 Cluster.

Several different node counts have been used to run the throughput benchmark in order to gauge the potential

performance gain ("speed-up") with increasing number of nodes. The elapsed time (minutes) is calculated by taking the time difference between the first and last output files. Table 9 gives a summary of each of the 13 test cases and the number of times the code is invoked in the 765 jobmix on the rx1620 cluster.

It should be noted that the weightings applied to the number of times a code appears in the script should be designed to simulate the expected usage of the cluster. Most commonly used codes should be given a higher weighting than codes that are only used by a minority of users. In this job mix *Hadamard* dominates the usage, with 500 occurrences in total.

The speedup is determined by combining all the times for the 765 jobs to obtain a "theoretical peak". In the case of the HP rx1620 this is 1290 minutes, as`shown in Table 9 i.e. it would require 1290 minutes to run all 765 jobs sequentially on one rx1620 processor.

Number of Nodes	Number of CPUs	Elapsed Time (minutes)	Speedup
1	1	1290 (est.)	1.0
16	32	81	15.9
32	64	49	26.3
48	96	35	36.9
64	128	29	44.5
72	144	25	51.6
∞	∞	11	117

Table 10: Workload Speedup using Multiple CPUs on the HP rx1620 (dual CPU, 1.6GHz Itanium2) Infiniband Cluster.

For an infinite number of processors, the time required to run the longest individual job is used as the elapsed time. On this system it is case9 (*RuWarray*) which requires 11 minutes on average to complete.

Table 10 demonstrates the speed up of the throughput mix as the processor count increases. Perfect speedup is not realised for a couple of reasons. First, the parallelisation is extremely course. Certain test cases require significantly longer than others to run, thus dominating the total runtime. In this scenario it is case9, which is invoked 5 times in the jobmix. Because this case has a longer execution time than the other codes, towards the end of the benchmark, all but 5 of the nodes are idle waiting for this case to complete. Also, as demonstrated earlier in this report, for memory bandwidth sensitive codes, when more than one processor per node is occupied, the performance of these codes decrease causing additional slow downs, thus affecting the overall speedup.

It takes a significant time for any scheduler to accept and schedule 765 jobs. This could be improved by assigning more than 4 jobs per 2-CPU systems, but this would then harm the 'shortest time to solution' approach for each job. The scheduler used on the rx1620 system was LSF. Performance of LSF could be fine-tuned by adjusting various parameters, reducing the time required to schedule the number of jobs. However for benchmarks to be reproducible on numerous systems, normally the default implementations of most applications are employed wherever possible to avoid any potential bias in the conclusions.

Similar trends are observed when the job mix is run on the Cray XD1 which uses the Rapid Array switching fabric and the Active Manager Job Management System (AM JMS) to schedule jobs. This resource manager is based on similar technology to sun grid engine (SGE). In order not to over saturate the scheduler during job submission, a pause between each of the 765 jobs was applied. This was controlled using the "sleep" command. The overall run times were extremely sensitive to this controlled submission setting, with it eventually being reduced to "sleep 1".

Number of Nodes	Number of CPUs	Elapsed Time (minutes)	Speedup
1	1	2331.7 (est.)*	1.0
8	16	167	14.0
16	32	85	27.4
24	48	58	40.2
32	64	48	48.6
35	70	43	54.2
∞	∞	12.7	184

Table11: Workload speedup using Multiple CPUs on a Cray XD1 (AMD Opteron 250 / 2.4GHz). *The estimated time was calculated using (Number of Instances of the code) × (single job elapsed time).

Whilst the Cray XD1 is comparable in performance to that observed on the rx1620 cluster, it doesn't quite show the same speedup. It would be interesting to repeat the exercise on an rx1620 using SGE to determine the impact of using an open source scheduler on the overall speedup in comparison to the commercial offerings such as LSF or PBS Pro.

Both these job mix cases used the default builds for all the HPC Integer codes. This meant the FORTRAN version of *Hadamard* was implemented in each case, despite evidence that the C version of the executable provides much improved performance. Table 12 shows the difference in performance of these job mixes on the XD1 when the C build of the *Hadamard* executable is used rather than the FORTRAN. Since *Hadamard* dominates the jobmix (500/765) it does indeed have quite a dramatic effect on performance.

Nodes	CPU	Workload with FORTRAN-based *Hadamard*		Workload with C-based *Hadamard*	
		Elapsed Time (mins)	Speed-up	Elapsed Time (mins)	Speed-up
1	1	2331.7	1.0	942.0	1.0
8	16	167	14.0	69	13.7
16	32	85	27.4	37	25.5
24	48	58	40.2	29	32.5
32	64	48	48.6	27	34.9
35	70	43	54.2	25	37.7
∞	∞	12.7	184	12.7	74.2

Table 12: Workload Speedups using both FORTRAN and C Versions of the *Hadamard* code on a Cray XD1 (AMD Opteron 250 / 2.4GHz).

8. MPI Parallel Integer Benchmarks

All of the preceding benchmark results have focused on the serial performance of the Integer benchmark codes. We finally turn to a consideration of the parallel performance of the suite, focusing on the MPI implementation of two of the codes, namely *Linemap* and *RuWarray*. Both codes have two associated input job decks.

Running the MPI parallel versions of the code will provide greater insight into the codes dependency on the cluster configuration, rather than just the processor, especially the requirements for interconnect bandwidth and latency. It will also highlight how well the code scales on the current generations of interconnect. The results presented in this section have all been performed using the HP-MPI routines on an rx1620 dual CPU cluster with Infiniband and Gigabit Ethernet interconnects. Analogous results were obtained on the Cray XD1, but are not presented in this report for reasons of space. Details can be made available upon request to the authors.

8.1 Parallel implementation of *Linemap*

The single processor performance of the MPI-version of *Linemap* gives very good agreement with the serial version of the code. Figure 27 demonstrates reasonably good scaling for the gigabit Ethernet interconnect but, surprisingly, Infiniband stops scaling at 8 processors. This is of course contrary to expectations, although there is a reasonably simple explanation. According to the Infiniband switch manufacturers, the switch tends to have a significant start-up time of up to several seconds for each job. This is attributed to the need to first build a routing table which requires a finite amount of time before the job commences. For standard applications, job times tend to be several hours in duration, in which case the few seconds required to build the table have a negligible impact on performance. In the case of *Linemap* which is an extremely short job, the time allocated to build the routing table becomes a dominant contributing factor in the recorded benchmarking time. There are no such “warm-up” requirements however for the Gigabit Ethernet switch.

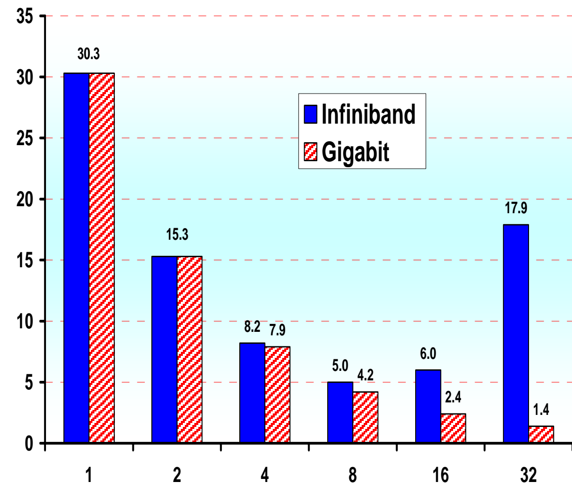


Figure 27: HP-MPI using *Linemap* on an HP rx1620 1.6GHz cluster with Infiniband and Gigabit Ethernet interconnects.

8.2 Parallel implementation of *RuWarray*

The two *RuWarray* cases are equivalent to test cases 7 and 9 in the HPC Integer benchmarking suite. Unlike *Linemap*, both *RuWarray* examples do not match the single CPU timing obtained during the jobmix benchmarks of Section 6. This can be traced to a completely different path in the source code.

Figures 28 and 29 show the elapsed times (in seconds) as a function of processor count for both test decks, featuring 3×10^9 and 9×10^9 random elements respectively. As the smaller job deck for *RuWarray* takes longer to complete than the *Linemap* benchmark, some of the highlighted performance issues around the infiniband switch do not have as great an impact and thus the code demonstrates scalability up to 8 CPUs. The larger benchmark scales up to 16 CPUs (Figure 29). This code, using either problem set, does not scale beyond these processor counts – in fact the performance starts to deteriorate at higher processor counts.

It would seem fair comment that the current MPI implementations of the two benchmark codes are far from optimal.

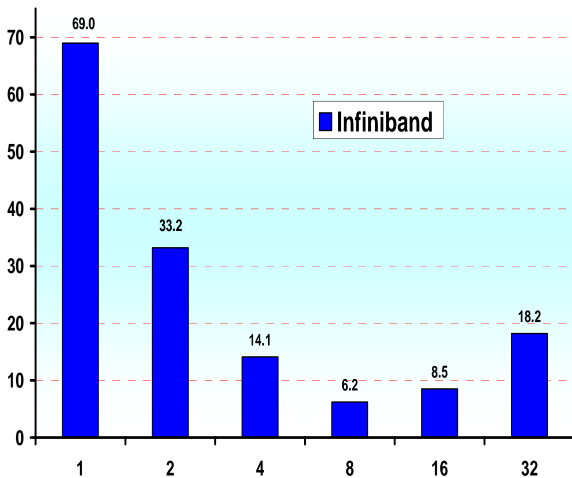


Figure 28: Parallel MPI performance of *RuWarray* using 3 × 10^9 random elements. Total Elapsed times (seconds) as a function of processor count.

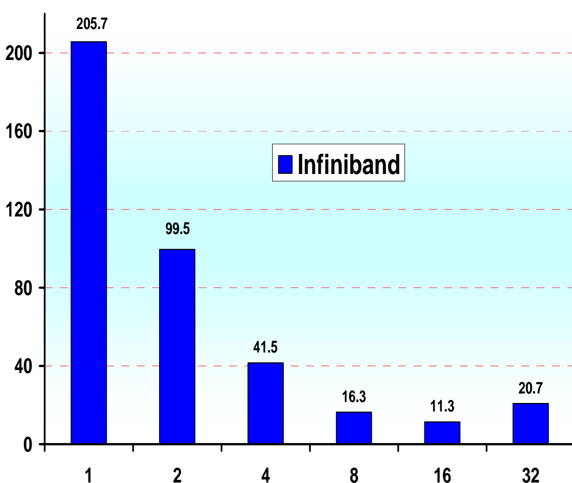


Figure 29: Parallel MPI performance of *RuWarray* using 9 × 10^9 random elements. Total Elapsed times (seconds) as a function of processor count.

9. Summary

This paper has introduced an integer-based benchmarking suite comprising seven legacy codes (with 13 associated datasets) designed to test both integer and Boolean performance. We have provided a detailed breakdown of the performance attributes of the codes comprising the suite, based on their execution on some 16 different systems. The focus has been primarily on measured performance on commodity-based processors including Itanium2, Opteron and EM64T Xeon systems together with IBM's proprietary Power5 (p5-575) processor. These evaluations have been carried out on both systems at Daresbury and those at external sites, the latter accessed via numerous collaborations involving both the academic and vendor communities.

The results have been systematically evaluated and presented across a number of key areas of code utilisation and associated performance in a number of stages:

1. Single Processor, serial performance. The overall serial benchmark times for each of the codes have been reported as a function of data set, and a performance comparison presented in each case which contrasts the Intel Itanium (IA64) performance in both HP and SGI systems with a variety of systems featuring both Intel Xeon EM64T and AMD Opteron (x86-64) CPUs. Contrary to the SPECint ratings, the Itanium2 processor is found to be the leading CPU in three of the benchmarks – *Linemap*, *Hadamard* and *RuWarray*. The Opteron processor is found to the fastest CPU in *Permutit*, *Treesearch* and Lineq.

We have sought to understand each code's dependency on clock speed, cache, and both memory latency and bandwidth, through a number of experiments in which both clock speed and cache levels have been systematically varied on a variety of Dell PowerEdge EM64T nodes.

2. We have extended the serial, single processor benchmarking approach to incorporate throughput, RATE-style benchmarks including six of the present integer codes and eleven data sets. The RATE benchmarks provide an additional performance probe crucial in understanding the impact of utilising all the processing elements on a multi-core system, and shed considerable light on the memory bandwidth requirements of each of the codes. The present approach is consistent with the SPECint2000 and SPECint_rate2000 benchmarks. In highlighting the memory bandwidth demands of the *Hadamard*, *RuWarray* and *Permutit* codes, this exercise has illustrated the FSB limitations in the EM64T series of Intel processors when compared to Opteron-based dual processor systems. In examining the emerging dual-core technologies from both AMD and Intel, we have considered the improvements underway within Intel's emerging generation of multi-core systems.

3. A consideration of software specific effects that impact on the observed performance has been undertaken through (i) the choice of compiler and compiler optimisation level, and (ii) the impact of coding language through a consideration of performance delivered when using C and FORTRAN. In the case of *Hadamard*, the C version of the code was surprisingly found to outperform the FORTRAN code on all platforms

4. Through the development of a Workload Benchmark, we have simulated "real-world" usage of a cluster and quantified the ensuing impact that fully populating the job scheduler has on the individual benchmark performance.

5. Parallel (MPI) Benchmarks – taking parallel MPI versions of two of the HPC Integer codes, we have examined the impact of Interconnect bandwidth and latencies on performance and the level of scalability achievable.

Finally, to summarise our findings across the serial, rate and parallel work loads described in this paper, we point to the performance sensitivity analysis of Section 4 in summarising the optimum processor family for each code, together with its dependency on clock speed, cache, and both memory latency.

Code	Optimum CPU	Performance Sensitivity Analysis			
		Clock speed (GHz)	L2 cache	Memory Bandwidth. (MB/s)	Memory Latency. (µs)
Linemap	Itanium	✓	X	X	X
Permutit	Opteron	✓	X	✓	✓
Hadamard	Itanium	X	X	✓	X
RuWarray	Itanium	X	✓	✓	X
Treesearch	Opteron	✓	✓	X	✓
Linequ	Opteron	✓	✓	X	X

10. References

[1] Comparative Study of Cray XD1 and PathScale InfiniPath clusters, I.N. Kozin, R. Wain, M. J. Deegan, M.F. Guest and C.A. Kitchen: http://www.cse.clrc.ac.uk/disco/publications/Cray_XD1_vs_InfiniPath_report.pdf

[2] The Standard Performance Evaluation Corporation (SPEC): http://www.spec.org/cpu2000

[3] Introduction to the HPCChallenge Benchmark Suite, J. Dongarra and P. Luszczek, ICL Technical Report, ICL-UT-05-01, (Also appears as CS Dept. Tech Report UT-CS-05-544), 2005. http://icl.cs.utk.edu/hpcc/

[4] Integer speed benchmarks, SPECint2000, http://www.spec.org/cpu2000/results/cint2000.html and Integer throughput benchmarks SPECint 2000 rates, http://www.spec.org/cpu2000/results/rint2000.html.

[5] The HPCS HPCChallenge RandomAccess benchmark, http://icl.cs.utk.edu/projectsfiles/hpcc/RandomAccess/

[6] http://www.cse.clrc.ac.uk/disco/DLAB_BENCH_WEB/HPCInteger/HPCinteger.pdf

[7] Floating point speed benchmarks, SPECfp2000, http://www.spec.org/cpu2000/results/cfp2000.html and Floating point throughput benchmarks SPECfp 2000 rates, http://www.spec.org/cpu2000/results/rfp2000.html

[8] The IOzone Filesystem Benchmark, http://www.iozone.org/

[9] Evaluation of the Cray XT3 at ORNL: a Status Report, S. R. Alam, R. F. Barrett, M. R. Fahey, O. E. Bronson Messer, R. T. Mills, P. C. Roth, J. S. Vetter and P. H. Worley, Oak Ridge National Laboratory, Published at the 2006 Cray User Group Meeting, Lugano, Switzerland.

[10] THE CCLRC / Intel Benchmark Project. 1.Serial Performance and Benchmarks. Igor Kozin, Miles Deegan, Martyn Guest, Christine Kitchen. http://www.cse.clrc.ac.uk/disco/Benchmarks/IntelProj.Serial.pdf

Acknowledgements

This work was performed under the auspices of the EPSRC's Distributed Computing Support Programme at CLRC Daresbury Laboratory. We thank HP for the technical support and access to numerous pre-production systems, Dell for access to the TACC cluster and EM64T systems, and to Cambridge On-line for making the various HP Dl45 and DL585 systems available.

Council for the Central Laboratory of the Research Councils
Chilton, Didcot, Oxfordshire OX11 0QX, UK
Tel: +44 (0)1235 445000 Fax: +44 (0)1235 445808

CCLRC Rutherford Appleton Laboratory
Chilton, Didcot,
Oxfordshire OX11 0QX
UK
Tel: +44 (0)1235 445000
Fax: +44 (0)1235 44580

CCLRC Daresbury Laboratory
Keckwick Lane
Daresbury, Warrington
Cheshire WA4 4AD
UK
Tel: +44 (0)1925 603000
Fax: +44 (0)1925 603100

CCLRC Chilbolton Observatory
Drove Road
Chilbolton, Stockbridge
Hampshire SO20 6BJ
UK
Tel: +44 (0)1264 860391
Fax: +44 (0)1264 860142

INVESTOR IN PEOPLE