

Optimising applications performance with compiler options for the Power5 on HPCx

C.M. Maynard

*EPCC, The University of Edinburgh, James Clerk Maxwell Building,
Mayfield Road, Edinburgh, EH9 3JZ, UK*

October 11, 2006

Abstract

An overview of the main architectural differences between the power4 and power5 chips is presented, with particular reference to simultaneous multi-threading. The use of different compiler flags is discussed. Finally an investigation of the effects of different compiler flags for two different versions of the fortran compiler, xlf9 and xlf10 is investigated by benchmarking two different application codes.

This is a Technical Report from the HPCx Consortium

© HPCx UoE Ltd 2005

Neither HPCx UoE Ltd nor its members separately accept any responsibility for loss or damage from the use of information contained in any of their reports or in any communication about their tests or investigations.

1 Introduction

The UK academic high performance computing service, HPCx, has been upgraded from the IBM Power4+ chip to the Power5. The Power5 has a number of enhancements compared to the Power4+, and yet the chip is designed to be binary compatible with the Power4. The aim of this report is to describe these enhancements, and how to optimise performance of applications using different compilers or compiler flags.

2 Power5 architecture

This section summarises the changes in micro-architecture from an applications perspective. The full details of the Power5 chip can be found in [1], and an overview of the HPCx phase 2a architecture can be found in [2]. Besides an increase in the specifications of some of the components such as an increase in some bus speeds, and the size of the level 2 (L2) cache, some modifications to the system structure of the Power4 were enacted. Specifically, the L3 cache has been moved from the memory side of the interconnection fabric to the processor side. The memory controller has now been moved on chip. This removes L3 hit traffic from the inter-chip buses. Latency to both L3 cache and memory is reduced and bandwidth to both increased. The effect of this improved memory architecture on some applications can be seen in [2].

A completely new feature of the Power5 chip is simultaneous multi-threading (SMT). Modern architectures make use of many advances to increase performance, such as caches, branch prediction, and out-of-order execution. Long instruction pipelines enable very high processor frequencies. This gives rise to very high peak performance. To achieve high performance is complicated by cache and memory latencies and branch misprediction. This coupled with long instruction pipelines can lead to low utilisation of execution units of the CPU, with the consequent impact on actual performance. To increase processor throughput a thread-level parallelism has been introduced.

There is more than one method for multi-threading. In coarse grain multi-threading only one thread executes at one time. When a long latency event, such as a cache miss is encountered, the hardware swaps execution to a different thread. In fine grain multi-threading again only one thread executes per cycle, but the hardware swaps thread each cycle. Finally, in simultaneous multi-threading (SMT) the hardware can schedule instructions from all threads concurrently. The Power5 system implements two threads per processor core. There are further differences between Power4 and Power5 chips, such as dynamic power management, but this not summarised here as it has no direct impact on applications.

The performance of SMT on HPCx is benchmarked for some applications in [3]. To enable SMT on HPCx, modifications to the loadleveler script are necessary, but not the application itself. Naively SMT can increase performance by a factor of two, although this is unlikely to be achieved. Actual performance will depend on individual applications. The reader is referred to [3] for details.

3 Compiler optimisations

The HPCx user guide [4] contains a detailed section on compiling for HPCx, some of which is necessarily repeated here. This section focuses on optimisations for the Power5, rather than, for instance, compiling an MPI code. The compilers are;

- FORTRAN 77 `xlf_r` V9.1
- FORTRAN 90 `xlf90_r` V9.1
- C `xlc_r` V7.0
- C++ `xlc_r` V7.0
- JAVA `javac`

where the `_r` suffix stands for *re-entrant* and denotes that the code is thread safe. This is required for Open MP codes and recommended for MPI codes.

By default the compilers use 32 bit addressing. To use 64 bit addressing the compiler flag is `-q64`. This increases the amount of memory available, as well as better memory management, and can be a good option for most codes. All objects and libraries must be compiled with the `-q64` flag. This can have a negative performance impact on C/C++ codes which use pointers heavily, as these pointers will be double in size.

There are many `-q` options, but perhaps the most relevant ones are `-qarch` and `-qtune`. The `arch` flag specifies the instruction set to the compiler, and the `tune` flag specifies which architecture the compiler should optimise for. For both flags the option can be set to be either `=pwr4` or `=pwr5`. Naively the `pwr5` option would be the best, but with the compilers currently installed on the service, for many applications, code compiled with this option may in fact run slower. Two examples are discussed below. It is worth experimenting for a particular application.

The `-On` flag is the main optimisation flag which can be specified with a range of values.

- `-O0` (FORTRAN only) fast compile and full debugging support
- `-O2` Low-level standard optimisation.
- `-O3` Memory and compile intensive optimisations. `-qhot=1` *higher order transformations* such as loop unrolling. Semantics of code can be altered, and hence result
- `-O4` `-qarch=native`, `-qtune=native` `-qipa=1` *inter-procedural analysis*. Inlining of functions
- `-O5` `-qipa=2`

Beyond `-O2`, the semantics of the code can be changed. Consequently, rounding errors may be different as the order of operations is modified. Thus the numerical value of the result may differ. When the optimisation flags `-O4` and higher are set, the `arch` and `tune` flags are set to native, so the code is compiled with `pwr5`.

Using fortran as an example compiler, the current recommendation is to compile with: `xlf_r -q64 -O3 -qarch=pwr4 -qtune=pwr4`.

Table 1: Benchmarks for different compile options for two application codes, in seconds

compiler options	dl_poly3 (s)	pchan (s)
xlf9, none	234	591
xlf9, pwr4	204	560
xlf9, pwr5	225	533
xlf10 none	230	566
xlf10, pwr4	196	543
xlf10,pwr5	202	526

With the caveats mentioned above, it is always wise to check that optimisations don't actually make the code run slower, and that the code gives the correct answer.

3.1 Benchmarking with the xlf compiler

Two codes are used to benchmark the xlf9.1 and xlf10.1 compilers, dl_poly3 [5] and pchan [6]. dl_poly3 is classical molecular dynamics code. The benchmark code was a system of Gramicidin molecules in water with 792,960 molecules. pchan is a finite difference code for simulating turbulent fluid flows. In [2], the performance these codes on the power4 chip was compared to that on the power5. For dl_poly3 with the Gramicidin benchmark, the code runs approximately 20% faster on a power5 system. For the pchan T3 benchmark (360³ grid) the same code runs approximately 100% faster on the power5.

In this investigation, the performance of application codes using different *compilers* and *compiler flags* on the power5 chip are compared. In particular, the effect of setting -qarch and -qtune set to pwr4 or pwr5 was examined. The Gramicidin benchmark was used for dl_poly and the T2 (240³ grid) benchmark for pchan. The remaining compiler flags were set as -q64 -03 and the code run on 32 processors. The results are shown in table 1.

There are several obvious features to note. The most obvious feature is that it depends on the application whether pwr4 or pwr5 flag should be set. With no architecture flag set, the codes run more slowly. For dl_poly3 pwr4 is quicker, for pchan pwr5 is quicker. The codes are have different memory footprint and data access patterns, so there is no reason why they should follow a consistent pattern. The xlf10 compiler generates code which executes more quickly, but the difference between the effect of the architecture compiler flags is smaller regardless of whether negative, or positive. Defining the variable Δ_{4-5} , from the time difference for each compiler version, of the application run time with pwr4 and pwr5 flags, as follows:

$$\Delta_{4-5} = \frac{T_{\text{pwr4}} - T_{\text{pwr5}}}{T_{\text{pwr4}}} \quad (1)$$

these are shown in table 2.

What is interesting is for both applications Δ_{4-5} is smaller for xlf10 than it is xlf9. This suggests that once the architecture flag is set to a power architecture, the xlf10 compiler is better at implementing efficient code, regardless of what the flag value is. Another way of looking at this is to define another time difference, Δ_{9-10} which compares

Table 2: Normalised time differences, Δ_{4-5} , to three significant figures, for each compiler version, for each application.

compiler version	dl_poly3	pchan
xlf9	-0.103	0.0482
xlf10	-0.0306	0.0313

Table 3: Normalised time differences, Δ_{9-10} , to three significant figures, for each compiler flag, for each application.

compiler flag	dl_poly3	pchan
none	0.0171	0.0423
pwr4	0.0392	0.0304
pwr5	0.102	0.0131

the different compilers for the same flags set.

$$\Delta_{9-10} = \frac{T_{\text{xlf9}} - T_{\text{xlf10}}}{T_{\text{xlf9}}} \quad (2)$$

This is shown in Table 3.

The newer compiler always produces faster executing code than the older one. However, where the use of compiler flags has enabled the compiler to implement efficient code, there is only a modest gain when moving to the new compiler. In contrast, where the compiler flags have produced relatively inefficient code, there is a larger speed up when moving to the new compiler, providing the architecture flags are set. If no architecture flags are set, then it depends on the application whether the newer compiler does significantly better. This supports the premise that the xlf10 compiler is better at implementing code efficiently once the architecture flags are set regardless of which power value they are given. For the application dl_poly3, the architecture flag should be set to pwr4 and the xlf10 compiler produces code which runs approximately 4% faster, for pchan it is pwr5, the code runs about 1% faster with the newer compiler.

The current version of the compiler on the HPCx service is xlf9. Whilst in the future the xlf10 compiler will be available, there are no immediate plans for an upgrade. However, for the two codes benchmarked at least, almost all the speed up achieved with the xlf10 compiler can be achieved with xlf9 by careful use of compiler flags. This is an empirical observation, and is clearly application dependent statement. When exploring compiler options it is always advisable to benchmark the code to firstly check for correctness, and secondly that the optimisations result in a performance gain.

4 Conclusions

This report describes the main differences between the Power4 and Power5 chips. Many applications will see a significant improvement in performance with the new architecture. SMT can benefit some applications. These enhancements do not require any changes to be made to application code. This was a key design goal of the Power5 chip. Finally,

compiler optimisation of codes was discussed, and two application codes were used to benchmark the FORTRAN xlf9 and xlf10 compilers. For these codes, whilst the new compiler produced a performance gain when compared to the old one, only modest gains were achieved when compiler flags are set appropriately.

References

- [1] B. Sinharoy *et al.* IBM J. Res & Dev. Vol. 49 NO 4/5 (2005) 505;
<http://researchweb.watson.ibm.com/journal/rd/494/sinharoy.html>
- [2] A. Gray *et al.*. HPCx TR602;
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0602.pdf
- [3] A. Gray *et al.* HPCx TR604;
http://www.hpcx.ac.uk/research/hpc/technical_reports/HPCxTR0604.pdf
- [4] <http://www.hpcx.ac.uk/support/introduction/index.html>
- [5] Smith, W. & Forester, T. 1996, DL_POLY: A General Purpose Parallel Molecular Dynamics Simulation Package. *J. Molec. Graphics* **14**, 136; Smith, W., Yong, C. & Rodger, M. 2002, DL_POLY: Applications to Molecular Simulation. *Molecular Simulation* **28**, 385; Todorov, I. & Smith, W. 2004, DL_POLY_3: The CCP5 National UK for Molecular Dynamics Simulations. *Phil. Trans. R. Soc. Lond. A* **362**, 1835;
<http://www.cse.clrc.ac.uk/arc/dlpoly.shtml>
- [6] PCHAN Webpage,
<http://www.cse.clrc.ac.uk/arc/pchan.shtml>