US QCD Computational Performance Studies with PERI

Y. Zhang[1], R. Fowler[1], K. Huck[2], A. Malony[2], A. Porterfield[1], D. Reed[1], S. Shende[2], V. Taylor[3], and X. Wu[3].

[1]Renaissance Computing Institute, Chapel Hill NC USA
[2]University of Oregon, Eugene OR USA
[3]Texas A&M University, College Station TX USA
Corresponding author e-mail: rjf@renci.org, yingz@renci.org

Abstract. We report on some of the interactions between two SciDAC projects: The National Computational Infrastructure for Lattice Gauge Theory (USQCD), and the Performance Engineering Research Institute (PERI). Many modern scientific programs consistently report the need for faster computational resources to maintain global competitiveness. However, as the size and complexity of emerging high end computing (HEC) systems continue to rise, achieving good performance on such systems is becoming ever more challenging. In order to take full advantage of the resources, it is crucial to understand the characteristics of relevant scientific applications and the systems these applications are running on. Using tools developed under PERI and by other performance measurement researchers, we studied the performance of two applications, MILC and Chroma, on several high performance computing systems at DOE laboratories. In the case of Chroma, we discuss how the use of C++ and modern software engineering and programming methods are driving the evolution of performance tools.

1. Introduction

Performance and productivity are issues of ever increasing performance in computational science. In order to generate scientific results efficiently, it is important that codes run well on our high performance systems. Even more important, it is vital that the scientists and programmers use their time effectively. Software, both application and systems, that delivers correct results, are easy to program, and that can be efficiently run across a wide variety systems, both current and future, is key to success in achieving these objectives.

The National Computational Infrastructure for Lattice Gauge Theory (USQCD, for short) is a SciDAC project that focuses on computational infrastructure for Lattice Quantum Chromodynamics (LQCD) computations. LQCD is computationally intensive, with some planned computations expected to consume hundreds of teraflop years. While part of the USQCD efforts have been the construction of special purpose systems, much of the current effort addresses the development of a stable software infrastructure to improve the long term productivity of computational physicists. For this software to be broadly accepted, it needs to achieve high performance across a wide spectrum of high end computer systems.

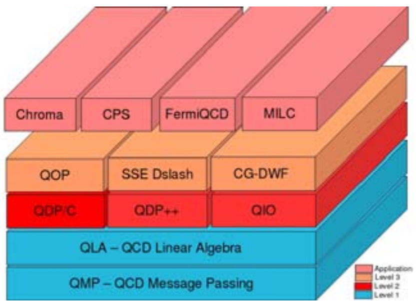
The Performance Engineering Research Institute (PERI) is a SciDAC project that focuses on improving the productivity of software performance engineering activities by improving performance analysis and modeling methods and especially by increasing the level of automation in measuring, analyzing, and tuning computational science applications. A key component of PERI is a strategy of active engagement with computational science groups, with other SciDAC projects, and especially with projects (SciDAC or not) that are the pioneering users of DOE's Leadership class systems. In addition to short term benefits to the application groups, these engagement activities are vital for ensuring that PERI activities stay on track towards meeting the long term needs of the high end computational science community.

In this paper we report some of the interactive engagement activities between USQCD and PERI. We begin by reporting on studies to analyze the performance characteristics of a recent version of the MILC (MIMD Lattice Computation) code. We then move on to discuss how Chroma, an LQCD infrastructure written using advanced techniques in C++, is serving as a driver for the development of new performance measurement and analysis tool methods.

2. Performance Evaluation of LQCD Codes and Libraries.

In this paper, we focus on performance studies for two US QCD applications on several HPC platforms using four performance analysis tools. Both applications use the software modules developed under SciDAC program.

The USQCD codes are based on a layered software architecture as illustrated in the figure to the right. The design and implementation of the libraries is a central part of the SciDAC project.



The MIMD Lattice Computation (MILC**)** infrastructure [1] is a package written in C. Developed by the MIMD Lattice Computation collaboration, it is used for simulations of four dimensional SU(3) lattice gauge theory on MIMD parallel machines. The latest version (7.4.0) of MILC, which is used in most of our experiments, utilizes five of the SciDAC software libraries [2]: QMP, QLA, QIO, QDP/C, and QOPQDP. The MILC version 7.1.11 for our computation efficiency studies uses QMP, QLA, QIO and QDP/C. QMP (QCD Message Passing) provides a standard communications layer for Lattice QCD based on MPI. QLA (QCD Linear Algebra) provides a standard interface for linear algebra routines. QIO (QCD Input/Output) provides as suite of input/output routines for lattice data. QDP/C is the C implementation of the QDP (QCD Data Parallel) interface. QOPQDP is an implementation of the QOP (QCD Operations) level three interface using QDP.

Table 1: Specification of four large-scale clusters

System Name	Jacquard	RENCI BlueGene/L	DataStar	Seaborg
Number of Nodes	356	1024	272	380
CPUs per Node	2	2	8	16
CPU type	2.2GHz AMD Opteron	700 MHz IBM PowerPC 440	1.5-1.7GHz POWER4	375MHz POWER3
Memory per Node	6GB	1GB	16-32GB	16-64GB
Network	InfiniBand	Torus	Federation	Colony
OS	Linux	Linux	AIX	AIX

We conducted our MILC performance studies on four HPC systems: the Jacquard [3] Linux cluster at NERSC (National Energy Research Scientific Computing Center), an IBM BlueGene/L (BGL) system [4] at RENCI, the Seaborg [5] IBM POWER3 SMP cluster at NERSC, and the DataStar [6] IBM POWER4 SMP cluster at SDSC (San Diego Supercomputing Center). Table 1 shows the configuration of the four systems. For these performance studies we used several performance tool sets that are well-suited for the kinds of scalability experiments we performed.

Prophesy[7] is a web-based infrastructure for the performance analysis and modeling of parallel and distributed applications. Prophesy includes a database for archiving performance data, system features and application details, to aid in online analysis and modeling of application performance. Prophesy allows for the development of linear as well as nonlinear models that can be used to predict the performance on different compute platforms or different numbers of processors.

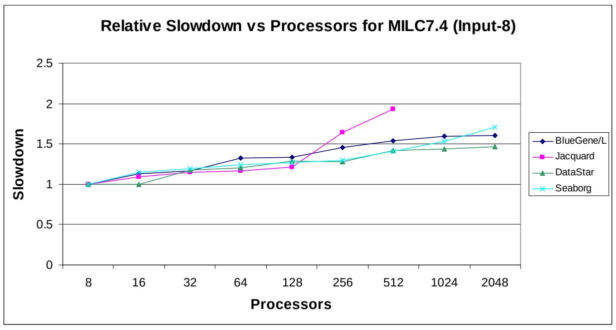
TAU[8] is a portable, scalable and integrated performance system supporting all DOE HPC platforms and all dominant programming languages, compilers, thread libraries, and communications libraries for HPC software development. TAU components include: multi-level performance instrumentation, multi-language automatic source instrumentation, flexible and configurable performance measurement, and a widely-ported parallel profiling and tracing system.

SvPablo[9] is graphical toolkit for instrumenting source code and browsing runtime performance data. SvPablo correlates application source code with dynamic performance data from both software and hardware measurement. It allows users to identify constructs that affects the performance at loop and function call level. It generates not only the performance statistical summary for the instrumented constructs, but also the process/thread based performance data via which, users can diagnose load imbalance problems.

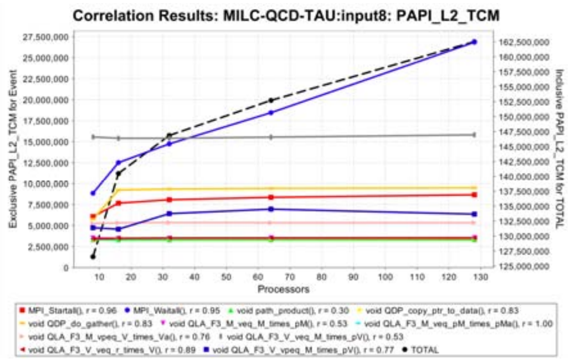
3. Performance analysis results

For MILC 7.4, we conducted both weak scaling and strong scaling experiments on the four platforms listed in Table 1 using Prophesy, and cache utilization studies on Jacquard using TAU. We also did computation efficiency studies for MILC 7.1.11 on IBM BGL using SvPablo.

Overall, MILC demonstrates good strong and weak scaling on all four systems. Figure 1(a) shows the weak scaling of MILC 7.4 with local lattice size of 8x8x8x8 (three space dimensions and one time dimension), presented with the relative slowdown of the code when the number of processors grows. We define that the relative slowdown is the execution time on *p* processors divided by the execution time on 8 processors. Given the fixed workload per processors, the relative slowdown quantifies how much the application performance degrades with increasing the number of processors. The best weak scaling is achieved on DataStar, the IBM Power4 system. On the Opteron cluster – Jacquard, MILC slows down significantly when the number of processors exceeds 128. For strong scaling, our result shows that the code has the best strong scaling on IBM BGL, taking advantage of its global tree network structure and nearest neighborhood communications.



(a)



(b)

Figure 1: Scaling and cache utilization analysis for MILC 7.4 with local lattice size of 8x8x8x8

The cache utilization analysis is conducted on the NERSC Opteron cluster. Figure 1(b) shows the correlation of level two (L2) cache misses of the whole program to the individual routines for local lattice size 8x8x8x8. The left Y-axis represents the L2 cache misses for individual routines. The right

Y-axis is the total number of L2 cache misses for the whole program – the dotted black line. The figure shows that the overall L2 cache misses increase as the number of processors grows. The L2 cache misses of MPI_Waitall() have very high correlation to the whole program, about 0.98. The cache misses for QLA routines remains steady as the number of processors increases.

4. Using USQCD code to drive new performance measurement and analysis.

The Chroma package is a new generation of LQCD code intended to improve simultaneously both performance and productivity in LQCD. It implements a new LQCD API developed under the SciDAC program to supports data-parallel programming constructs for lattice field theory and in particular lattice QCD. It does this by using advanced programming methods supported by C++. Chroma uses the SciDAC QDP++ data-parallel programming (in C++) to present a single high-level code image to the user. This approach can facilitate the generation of generate highly optimized code for many architectural systems including single node workstations, clusters of workstations via QMP, and classic vector computers. Chroma is also a leading candidate for use on emerging multi-core, multi-threaded processors.

Chroma embraces the use of generics in the form of templates, especially expression templates, to support modular programming practices while also facilitating compiler generation of very efficient executable code through the use of extensive inlining to enable aggressive post-inlining code optimization.

While this strategy has a lot of promise, the resulting code can be difficult to understand. Debugging codes with extensive templates has been recognized as a difficult problem because interactions of separately defined and developed templates can be very tricky. Apparently simple constructs in a source line can cause the instantiation, and inlining, of code from several “towers” of templates with source code coming from many different files. These same phenomena raise similar problems in performance measurement and analysis. Simple constructs can hide complexity because implicit and non-intuitive operations, such as hidden data copying, can be inserted in the generated object code.

These issues arise across a wide spectrum of “modern” codes, and are becoming an issue as compilers get more aggressive about inlining in older codes written in Fortran. Measurement approaches based on the insertion of instrumentation are impractical because many of the template methods that need to be measured are only a few machine instructions, *i.e.*, they cost less per operation than the instrumentation surrounding them. Hence, instrumentation at this granularity, even if done to optimized executable code, introduces unacceptable perturbations. Furthermore, inserting instrumentation prior to optimization, e.g., in source code, would perturb or inhibit subsequent optimizations. While tools based on event based sampling, such as HPCToolkit [9, 10, 11] or Oprofile[12], can collect measurements efficiently, the attribution of costs in such tools has been based either on the location of the code in the original source, or on calling context information. PERI researchers at Rice University and the University of North Carolina are addressing the problem analyzing the performance of such codes through extensions to HPCToolkit.

Using Chroma as the driving example, we present a preview of emerging extensions to HPCToolkit to collect, attribute, and display multiple performance metrics, including computed quantities, using three distinct strategies for cost attribution:

- The legacy method is a *source-oriented view*. It attributes the performance metrics within a flat representation of the source code file to the line (or statement) where the relevant program construct originally appeared in the input to the compiler.

- The second method is an *optimized object-oriented view*. Costs are attributed according to where they are incurred in the object code. The presentation of the costs is based on a hierarchical representation of the optimized object code that includes the trees of inclusions of template methods, macros, and other inlined code. The source code corresponding to the included code is presented at each location at which it is instantiated.
- The third method is a *calling context view*. Data is collected using a full call stack profiler and the data iis presented using the hierarchy of calling contexts. This view also captures inlining.

Each of these methods of cost attribution and display has its place in the analysis of performance. The emerging calling context and object code views have special value for the diagnosis of performance problems in highly-optimized codes using modern software engineering methods.

5. Conclusions

The work we present in this paper represents the symbiotic collaboration between two SciDAC projects. PERI, the computer science project, is providing performance measurements and analyses, including the tracking of performance and scalability across multiple systems and over time, to the LQCD community. In turn, the LQCD community is providing challenges that are driving the development of new performance measurement and analysis methods and tools.

References

[1] The MIMD Lattice Computation (MILC) Collaboration code, http://www.physics.utah.edu/~detar/milc
[2] US Lattice Quantum Chromodynamics, http://usqcd.jlab.org/usqcd-software/
[3] NERSC Jacquard, http://www.nersc.gov/nusers/resources/jacquard/
[4] UNC RENCI BlueGene/L, http://www.renci.org/about/computing.php
[5] NERSC Seaborg, http://www.nersc.gov/nusers/ resources/SP/
[6] SDSC DataStar, http://www.sdsc.edu/user_services/datastar/
[7] Xingfu Wu, Valerie Taylor, and Joseph Paris, A Web-based Prophesy Automated Performance Modeling System, the IASTED International Conference on Web Technologies, Applications and Services (WTAS2006), July 17-19, 2006, Calgary, Canada
[8] S. Shende and A. D. Malony, "The TAU Parallel Performance System," International Journal of High Performance Computing Applications, SAGE Publications, 20(2):287-331, Summer 2006
[9] Luiz DeRose and Daniel A. Reed, SvPablo: A Multi-Language Architecture-Independent Performance Analysis System, Proceedings of the International Conference on Parallel Processing (ICPP'99), Fukushima, Japan, September 1999
[10] J. Mellor-Crummey, R. Fowler, and G. Marin, HPCBiew: A tool for top-down analysis of node performance, Journal of Supercomputing, v. 23, 2002, pp. 81-104.
[11] N. Froyd, J.Mellor-Crummey, and R. Fowler, Efficient Call-stack Profiling of Unmodified, Optimized Code, Proc. Of the International Conference on Supercomputing (ICS2005), Cambridgs, MA, June, 2005, pp. 81-90.
[12] N. Froyd, N. Tallent, J. Mellor-Crummey, and R. Fowler, Call path profiling for unmodified, optimized binaries, Proceedings of the GCC and Gnu Toolchain Developers' Summit, Ottawa, June, 2006.
[13] J Levon *et al.*. OProfile. http://oprofile.sf.net/