Benchmarking Parallel Three Dimensional FFT Kernels with ZENTURIO*

Radu Prodan[1], Andreas Bonelli[2], Andreas Adelmann[3], Thomas Fahringer[4], and Christoph Überhuber[2]

[1] Institute for Software Science, University of Vienna, Liechtensteinstrasse 22, A-1090 Vienna, Austria
[2] Institute for Applied Mathematics and Numerical Analysis, Vienna University of Technology, Wiedner Hauptstrasse 8-10/1152, A-1040 Vienna, Austria
[3] Paul Scherrer Institut, CH-5232 Villigen, Switzerland
[4] Institute for Computer Science, University of Innsbruck, Technikerstrasse 25/7, A-6020 Innsbruck, Austria

Abstract. Performance of parallel scientific applications is often heavily influenced by various mathematical kernels like linear algebra software that needs to be highly optimised for each particular platform. Parallel multi-dimensional Fast Fourier Transforms (FFT) fall into this category too. In this paper we describe a systematic methodology for benchmarking parallel application kernels using the ZENTURIO experiment management tool. We report comparative results on benchmarking two three dimensional FFT kernels on a Beowulf cluster.

1 Introduction

The performance of parallel scientific applications is often influenced by various mathematical kernels like linear algebra software that needs to be optimised for high performance on each individual platform. Advanced parallel multi-dimensional Fast Fourier Transform (FFT) algorithms make use of such linar algebra software. While tools like the Automatically Tuned Linear Algebra Software (ATLAS) [1] are designed to automatically perform such optimisations, they usually have a strong narrow focus with limited hard-coded parametrisation and performance measurement options.

The ultimate goal of the work described by this paper is to evaluate several parallel FFT kernels for various configuration parameters like problem size, machine size, interconnection network, communication library, and target machine architecture. Our results will serve as input to a group of physicists at the Paul Scherrer Institut that have initiated and stimulated this work within the context of solving large scale partial differential equations. In this context, we formulated a generic methodology for benchmarking arbitrary software kernels for arbitrary configuration and run-time parameters using the ZENTURIO experiment management tool [2].

* This research is supported by the Austrian Science Fund as part of the Aurora project under contract SFBF1104.

M. Bubak et al. (Eds.): ICCS 2004, LNCS 3037, pp. 459–466, 2004.

This paper is organised as follows. Section 2 describes the ZENTURIO experiment management tool in brief. Section 3 is the main section of the paper devoted to our FFT benchmark experiments conducted with ZENTURIO. Section 4 concludes the paper.

2 ZENTURIO Experiment Management Tool

ZENTURIO is a semi-automatic tool for multi-experimental performance and parameter studies of parallel applications on cluster an Grid architectures. ZENTURIO has been designed as a distributed Grid service-based architecture, presented in detail in [2]. In addition to the Grid-enabled architecture, ZENTURIO can be run in light-weight cluster mode too, where all Grid services are replaced by simple Java objects that run on the cluster front-end. This mode was used for conducting the FFT benchmarks described in this paper and will be outlined in the remainder of this section.

Existing conventional parameter study tools [3] restrict parameterisation to input files only. In contrast, ZENTURIO uses a directive-based language called ZEN [4] to annotate any application files and specify arbitrary application parameters (e.g. program variables, file names, compiler options, target machines, machine sizes, scheduling strategies, data distributions, software libraries), as well as a wide range of performance metrics (e.g. cache misses, load imbalance, execution, communication, or synchronisation time). Additionally, constraint directives are introduced to filter erroneous experiments with no semantical meaning (see Example 1).

Through a graphical *User Portal*, the user inputs the application files, together with the compilation and execution commands. Based on the ZEN directives, an *Experiment Generator* module automatically instruments the application and generates the corresponding set of experiments. The SCALEA [5] instrumentation engine, based on the Vienna Fortran Compiler front-end that supports the MPI, OpenMP, and HPF programming paradigms, is used to instrument the application for performance metrics. After each experiment has been generated, an *Experiment Executor* module is responsible for compiling, executing, and managing its execution. The Experiment Executor interacts at the back-end with a batch job scheduler like fork, LSF, LoadLeveler, PBS, Sun Grid Engine, Condor, GRAM, or DUROC, as supported by our implementation. After each experiment has completed, the application output results and performance data are automatically stored into a PostgreSQL-based relational *Experiment Data Repository*. High level performance overheads related to the MPI, OpenMP, or HPF programming paradigms are computed using a post-mortem performance analysis component of SCALEA. An *Application Data Visualiser* module of the User Portal, based on the Askalon Visualisation Diagram package [6], provides a graphical interface to automatically query the repository and generate customisable visualisation diagrams that show the variation of any output parameters or performance metrics as a function of arbitrary input parameters (expressed through ZEN annotations).

3 Three Dimensional FFT Benchmarks

Our goal for this paper is to show a generic methodology for benchmarking and comparative analysis of parallel application kernels using ZENTURIO. We exemplify our techniques in the context of two three dimensional FFT algorithms which we briefly describe in the following.

FFTW [7] is a portable subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions of arbitrary input size, and of both real and complex data. Existing benchmarks [8] performed on a variety of platforms show that FFTW's performance is typically superiour to that of other publicly available FFT software, and is even competitive with non-portable, highly optimised vendor-tuned codes. The power of FFTW is the ability to optimise itself on the machine it executes through some pre-defined codelets run by a planner function before calling the real FFT.

wpp3DFFT developed by Wes Petersen at ETH Zurich uses a generic implementation of Temperton's in-place algorithm [9] for an $n = 2^m$ problem size, with the particular focus of making the transpose faster. The optimised algorithm pays a flexibility price, which restricts both the problem matrix size and the machine size to powers of two.

All experiments have been conducted on a single Intel Pentium III Beowulf cluster at ETH Zurich, which comprises 192 dual CPU Pentium III nodes running at 500 MHz with 1GB RAM, interconnected through 100 MBit per second Fast Ethernet switches. The nodes are organised into 24 node frames interconnected through 1 GBit per second optical links. Future work will repeat the benchmarks on various other parallel platforms, including IBM SP.

3.1 ZEN Parameter Annotations

There are three problem parameters which we vary in our benchmarks:

Problem size ranging from 2^3 to 2^8, expressed through source file annotations. Large problem sizes could not be run due to the limited amount of memory available on one node.

Communication library expressed by the `MPIHOME` ZEN variable in the application Makefile (see Example 1). The communication libraries under comparative study are LAM and MPICH-P4 MPI implementations. Shared memory has been used for communication within each SMP node. The constraint directive insures the correct association between the MPI library location and the `mpirun` command script defined in the PBS script used to submit the application on the cluster.

Machine size ranging from 2^1 to 2^6 dual nodes, each node running two MPI processes, expressed through PBS script annotations.

The total execution time, the transpose time, and MPI performance overheads have been measured using the performance behaviour directives based on the SCALEA performance library. Since small FFT problems have extremely

short execution times (order of milliseconds), they are prone to perturbations from the operating system or other background processes with low nice priority. To avoid such consequences, we repeat each experiment for a long enough amount of time (five minutes) and compute the mean of all measurements.

Example 1 (Sample Annotated Makefile).

```
MPIHOME = /usr/local/apli/lam
#ZEN$ ASSIGN MPIHOME ={/usr/local/apli/lam, /usr/local/apli/mpich}
#ZEN$ CONSTRAINT INDEX MPIHOME == run.pbs:MPIRUN
. . .
$(EXEC): $(OBJS)
  $(MPI_HOME)/bin/mpicc -o $(EXEC) $(OBJS) $(LIBS)
```

3.2 Benchmark Results

The annotations described in Section 3.1 specify a total of 72 experiments, which were automatically generated and conducted by ZENTURIO separately for each FFT algorithm. After each experiment has finished, the performance data collected is automatically stored into the Experiment Data Repository. The Application Data Visualiser module of the User Portal is used to formulate SQL query against the data repository and generate customisable diagrams that display the evolution of arbitrary performance metrics as a function of application parameters, mapped to arbitrary visualisation axes.

Figures 1(a) and 1(b) display the speedup curves of the two FFT algorithms, normalised against the lowest machine size executed (2 dual nodes), as a sequential experiment was not available. The speedup is bad for small problem sizes for which large parallelisation deteriorates performance. Large problem sizes offer some speedup until a certain critical machine size.

The explanation for the poor speedup curves is given by the large fraction used by the transpose (region 2) and MPI overheads (i.e. `MPI_Sendrecv_replace` routine to interchange elements in transpose) from the overall execution time, as displayed in Figure 1(c) (FFTW shows similar overhead curves). It is interesting to notice that both algorithms scale quite well until 16 dual nodes for a 2^8 problem size, after which the performance significantly degrades. The reason is the fact that larger machine sizes spawn across multiple cluster frames which communicate through 3 PCI switches, 2 Ethernet, and 2 Fast-Ethernet wires that significantly affect the transpose communication time. For small problem sizes, the execution time is basically determined by the transpose overhead that naturally increases proportional with the machine size (see Figures 1(d) and 1(e)). In contrast to wpp3dFFT, FFTW shows an interesting behaviour of keeping the transpose and the total execution time constant even for large machine sizes. The explanation is given by the load balancing analysis from Figure 2(a).

ZENTURIO offers a series of data aggregation functions, comprising maximum, minimum, average, or sum, for metrics measured across all parallel (MPI) processes or (OpenMP) threads of an application. Let $\mathcal{M}$ denote a performance

metric and $\mathcal{M}_i$ its measured instantiations across all n parallel processes or threads of a parallel application, $\forall\ i \in [1, n]$. We define the *load balance* aggregation function for the metric $\mathcal{M}$ as the ratio between the average and sum aggregation values: $\frac{\sum_{i=1}^{n} \mathcal{M}_i}{\sum_{i=1}^{n} \mathcal{M}_i}$.

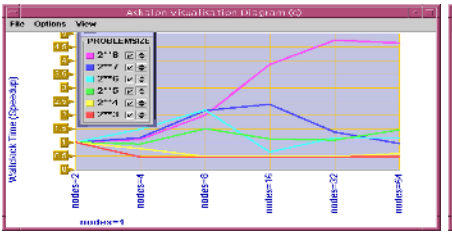
wpp3dFFT shows a good load balance close to 1 for all problem and machine sizes (see Figure 2(b)), while FFTW exhibits a severe load imbalance behaviour, the smaller problems are and the larger the machine sizes get (see Figure 2(a)). The explanation is the fact that FFTW in its planner function (that chooses optimised codelets for a certain platform) also detects that a machine size is too large for a rather small problem size to be solved. As a consequence, it decides to use only a subset of the processors for doing useful computation and transpose, while the remaining MPI processes simply exit with an `MPI_Finalize`. This explains the even execution time for small problem sizes shown in Figure 1(d).

Figure 1(f) shows a better performance of the LAM MPI implementation compared to MPICH for small problems and large machine sizes. Such experiments are bound to exchanging large number of small messages dominated by latencies, for which the LAM implementation seems to perform better. Large problem sizes shift the focus from message latency to network bandwidth, in which case both implementation perform equally well (see Figure 1(g)). Another suite of experiments currently under way on a local cluster at the University of Vienna shows that high speed interconnection networks (not available on the ETH cluster) like Myrinet give an approximate two fold improve in performance (see Figure 1(h)).

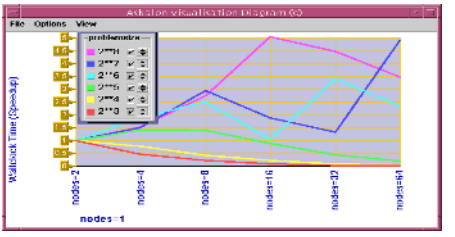
A comparative analysis of the two FFT parallel algorithms shows, as expected, a better performance of wpp3DFFT compared to FFTW for large problem sizes, which is due to the highly optimised wpp3DFFT transpose implementation (see Figure 2(c)). For small problem sizes, FFTW performs much better due to its intelligent run-time adjustment of machine size in the planning phase (see Figure 2(d)). The metric in which the ETH physicists are particularly interested is the ratio between the transpose and computation ratio, the latter being defined as the difference between the overall execution time and the transpose. This metric is comparatively displayed in Figures 2(e) and 2(f).

4 Conclusions

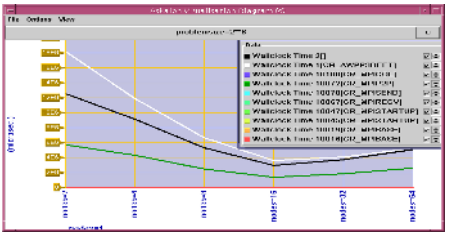
We have described a general methodology for benchmarking parallel application kernels using the ZENTURIO experiment management tool. We have applied the methodology for semi-automatic comparative benchmarking of two parallel FFT algorithms on a single Beowulf cluster. Parallel three dimensional FFT algorithms suffer from a severe communication bottleneck due to the highly expensive transpose (data communication) operation which increases proportional with the problem size. High performance networks like Myrinet are crucial for improving performance of such algorithms. LAM exhibits smaller latencies compared to MPICH in exchanging small messages, and similar bandwidth for large



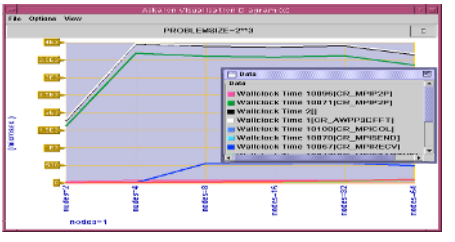
(a) FFTW Speedup.



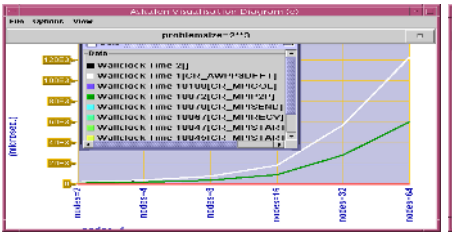
(b) wpp3DFFT Speedup.



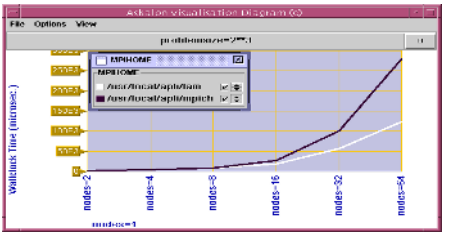
(c) wpp3DFFT Overheads (2^8 size).



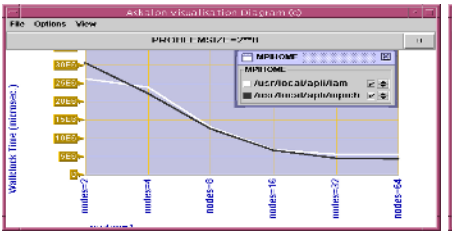
(d) FFTW Overheads (2^3 size).



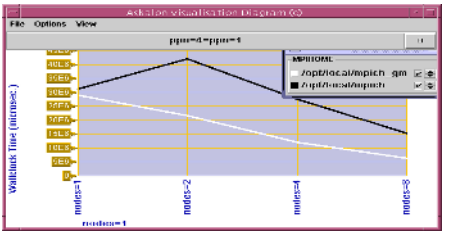
(e) wpp3DFFT Overheads (2^3 size).



(f) wpp3DFFT LAM versus MPICH.

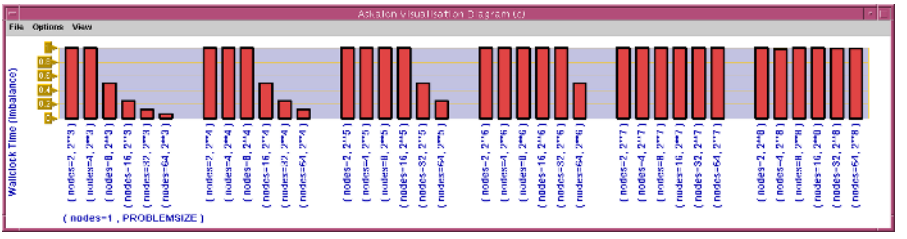


(g) FFTW LAM versus MPICH.

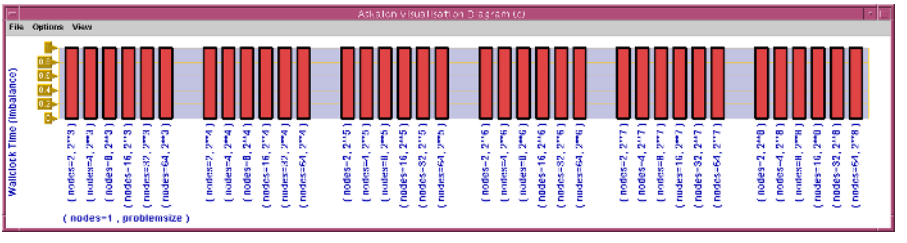


(h) Fast Ethernet versus Myrinet.

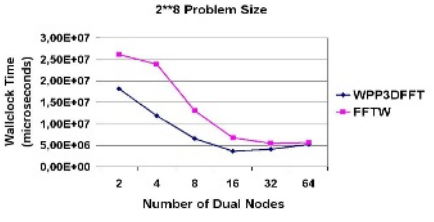
Fig. 1. Three Dimensional FFT Benchmark Results.



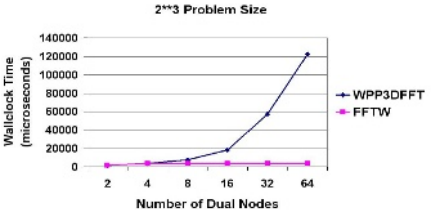
(a) FFTW Load Balance.



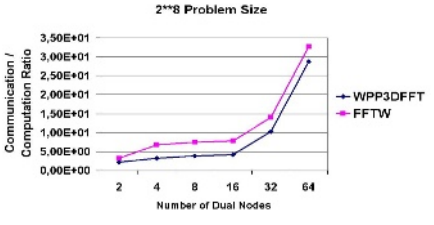
(b) wpp3DFFT Load Balance.



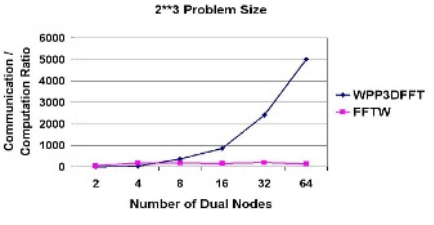
(c) FFTW versus wpp3DFFT.



(d) FFTW versus wpp3DFFT.



(e) FFTW versus wpp3DFFT.



(f) FFTW versus wpp3DFFT.

Fig. 2. Three Dimensional FFT Benchmark Results.

messages. wpp3DFFT algorithm performs better than FFTW in solving large FFTs due to an optimised transpose implementation. However, wpp3DFFT pays a flexibility price that restricts the problem and machine sizes to powers of 2, while FFTW can solve any problem size over arbitrary machine sizes. Smaller problem sizes are solved more efficiently by FFTW due to an intelligent run-time adjustment of machine size in the planning stage before calling the real FFT.

Future work will enhance ZENTURIO with a generic framework that employs standard heuristics like genetic algorithms and simulated annealing for solving a variety of NP complete optimisation problems, such as scheduling of single (workflow, MPI) Grid applications, as well as of large sets of applications for high performance throughput (complementary to [3]). The work will rely on existing Grid monitoring and benchmarking infrastructures like [10] and [11].

References

1. R. Clint Whaley and Jack J. Dongarra. Automatically Tuned Linear Algebra Software (ATLAS). In *Proceedings of the High Performance Networking and Computing Conference*, Orlando, Florida, 1998. ACM Press and IEEE Computer Society Press.
2. Radu Prodan and Thomas Fahringer. ZENTURIO: A Grid Middleware-based Tool for Experiment Management of Parallel and Distributed Applications. *Journal of Parallel and Distributed Computing*, 2003. To appear.
3. D. Abramson, R. Sosic, R. Giddy, and B. Hall. Nimrod: A tool for performing parameterised simulations using distributed workstations high performance parametric modeling with nimrod/G: Killer application for the global grid? In *Proceedings of the 4th IEEE Symposium on High Performance Distributed Computing (HPDC-95)*, pages 520–528, Virginia, August 1995. IEEE Computer Society Press.
4. Radu Prodan and Thomas Fahringer. ZEN: A Directive-based Language for Automatic Experiment Management of Parallel and Distributed Programs. In *Proceedings of the 31st International Conference on Parallel Proces sing (ICPP 2002)*. IEEE Computer Society Press, August 2002.
5. Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Parallel Programs. *Concurrency and Computation: Practice and Experience*, 15(11-12):1001–1025, 2003.
6. T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H.-L. Truong. ASKALON - A Programming Environment and Tool Set for Cluster and Grid Computing. www.par.univie.ac.at/project/askalon, Institute for Software Science, University of Vienna.
7. Matteo Frigo and Steven G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proc. 1998 IEEE Intl. Conf. Acoustics Speech and Signal Processing*, volume 3, pages 1381–1384. IEEE, 1998.
8. Matteo Frigo and Steven G. Johnson. benchFFT. http://www.fftw.org/benchfft/.
9. Clive Temperton. Self-sorting in-place fast Fourier transforms. *SIAM Journal on Scientific and Statistical Computing*, 12(4):808–823, July 1991.
10. The CrossGrid Workpackage 2. Grid Application Programming Environment. http://grid.fzk.de/CrossGrid-WP2/.
11. Hong-Linh Truong and Thomas Fahringer. SCALEA-G: a Unified Monitoring and Performance Analysis System for the Grid. In *2nd European Across Grid Conference (AxGrids 2004)*, Nicosia, Cyprus, Jan 28-30 2004.