On Using ZENTURIO for Performance and Parameter Studies on Cluster and Grid Architectures*

Radu Prodan and Thomas Fahringer
Institute for Software Science
University of Vienna
Liechtensteinstr. 22, 1090 Vienna, Austria
{radu,tf}@par.univie.ac.at

Franz Franchetti
Institute for Applied and Numerical Mathematics
Technical University of Vienna
Wiedner Hauptstr. 8-10/1152, 1040 Vienna, Austria
franz.franchetti@tuwien.ac.at

Michael Geissler
Photonics Institute
Technical University of Vienna
Gusshausstr. 27/387,
1040 Vienna, Austria
geissler@tuwien.ac.at

Georg Madsen
Institute for Physical and
Theoretical Chemistry
Technical University of Vienna
Getreidemarkt 9/156,
1060 Vienna, Austria
madsen@theochem.tuwien.ac.at

Hans Moritsch
Department of Business
Administration
University of Vienna
Brünner Str. 72,
1210 Vienna, Austria
moritsch@finance2.bwl.univie.ac.at

Abstract

Over the last decade, a dramatic increase has been observed in the need for generating and organising data in the course of large parameter studies, performance analysis, and software testings. We have developed the ZENTURIO experiment management tool for performance and parameter studies on cluster and Grid architectures. In this paper we describe our experience with ZENTURIO for performance and parameter studies of a material science kernel, a three-dimensional particle-in-cell simulation, a fast fourier transform, and a financial modeling application. Experiments have been conducted on an SMP cluster with Fast Ethernet and Myrinet communication networks, using PBS (Portable Batch System) and GRAM (Globus Resource Allocation Manager) as job managers.

1. Introduction

The development and execution management of scientific and engineering applications on complex, heterogeneous and non-dedicated distributed architectures, ranging from cluster architectures to widely distributed Grid structures, is a tedious, time-consuming, and error-prone process. Commonly these efforts require to conduct large number of executions for a given application. In previous work [14] we introduced the ZENTURIO experiment management tool for parameter studies, performance analysis, and software testings on cluster and Grid architectures. ZENTURIO provides the *ZEN directive-based language* [13] to specify arbitrary value ranges for any application parameter, including program variables, file names, compiler options, target machines, machine sizes, scheduling strategies, data distributions, etc. Application parameters are called *ZEN variables*. In addition, ZEN directives can be used to request a large variety of performance metrics (e.g. cache misses, load imballance, execution, communication, synchronisation time) for an application. ZENTURIO has been designed and implemented as a distributed service architecture based on the Jini technology (see Fig. 1). A graphical-based *User Portal* allows to input an application via its constituent files and displays an updated view of experiments, as they progress. From the ZEN-annotated application, ZENTURIO automatically generates the full set of experiments by using an *Experiment Generator* service. The SCALEA [17] performance analysis tool, which provides a complete Fortran90, OpenMP, and HPF front-end, is used to instrument the application for performance metrics. After being generated, experiments are automatically transferred to an *Experiment Executor* service, which launches and manipulates the experiments on the target machine. Upon their completion, the output and performance data are automatically stored into an Experiment Data Repository for post-mortem analysis. High level performance overheads are computed and stored by a post-mortem performance analysis component of SCALEA. An *Application Data Visualiser*, built on top of the Askalon visualisation diagrams [6], is used to automatically visualise the collected data across multiple experiments. A wide range of diagrams is supported, including barchart, linechart, piechart, and surface.

This paper illustrates the use of ZENTURIO for perfor-

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

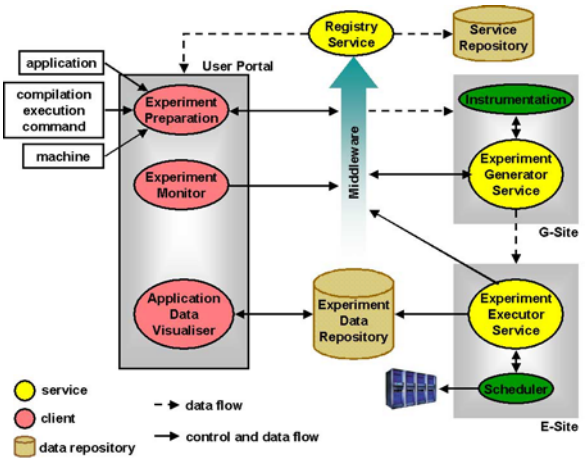


Figure 1. The architecture of ZENTURIO.

mance and parameter studies of four parallel applications. Our goal is to demonstrate that ZENTURIO is a generic practical tool that enables to easily set-up and automatically conduct and analyse large number of experiments for parameter and performance studies of parallel applications.

The next section summarises some related work. Section 3 illustrates the use of ZENTURIO on four high-performance parallel applications. Section 4 concludes the paper and outlines some future work.

2. Related Work

The ZOO project [11] has been initiated to support generic scientific experiment management by providing an experiment specification language and by supporting automatic experiment execution and data exploration. Parameterisation is limited to input and output files.

Nimrod [1] and ILAB [18] are tools that manage the execution of parameter studies. Unlike in ZENTURIO, parametrisation is limited to input files only.

Paradyn [12] supports performance experiment management through techniques for quantitatively comparing several experiments. In contrast to ZENTURIO, experiments have to be run manually. Performance analysis is done automatically for individual experiments only, based on historical data and dynamic instrumentation.

SKaMPI [15] provides a benchmarking environment for MPI codes. A runtime parameter file allows the description of a suite of measurements. A public performance database allows storage of benchmark data and interactive comparison of MPI operations across different implementations and platforms. Only a fixed number of machine and problem sizes can be controlled by the programmer. SKaMPI provides limited control for performance analysis, while ZENTURIO integrates a full performance analysis tool.

The US National Institute of Standards and Technology (NIST) developed an automated benchmarking toolset [3] which provides a central repository for all collected data and integrated analysis and visualisation mechanisms. Parametrisation is not generic like in ZENTURIO, but restricted to pre-defined application parameters.

The XPARE (eXPeriment Alerting and REporting) [4] tools specify automated benchmark regression testings for a given set of performance measurements of parallel applications. A historical panorama of the evolution of performance across software versions is provided. Apart from software versioning, no other parametrisation is addressed.

The Tracefile Testbed [7] is a new community repository for performance data of parallel applications. It allows users to search and retrieve tracefile metadata flexibly, based upon parameters such as computer platform used, types of events recorded, class of application, etc. Automatic execution of experiments and data collection are not addressed.

Unicore [16] facilitates the usage of supercomputers on the Internet. Unicore jobs are manually organised as directed acyclic graphs, consisting of job groups and tasks. There is no support for automatic invocation of multiple experiments, parameter studies, and performance analysis.

3. Experiments

In this section we describe several performance and parameter studies that have been conducted by ZENTURIO on four different parallel applications. All experiments have been run on a cluster of SMP nodes, each node consisting of 4 Intel Pentium III Xeon 700 MHz CPUs, with 1MB full-speed L2 cache, 2 GByte ECC RAM, running Linux 2.2.18-SMP. The cluster has two interconnection network installed: Fast Ethernet and Myrinet. Globus GRAM and PBS have been used as job schedulers.

3.1. LAPW0

LAPW0 is a material science kernel, part of the Wien97 package [2], that calculates the potential of the Kohn-Sham eigen-value problem. LAPW0 is implemented as a Fortran90 MPI program. Experiments have been scheduled on the SMP cluster using PBS. We have varied several application parameters by means of ZEN directives. The problem size is expressed by pairs of `.clmsum` and `.struct` input files, indicated in the `lapw0.def` input file (see Ex. 3.1). *ZEN substitute directives* are used to specify the file locations. The *ZEN index constraint directive* pairs the input files, according to the four problem sizes examined: 8, 16, 32, and 64 atoms. The machine size is controlled by the `nodes=1` and `no_procs` ZEN variables in the PBS script (see Ex. 3.2). `nodes=1` controls the number of SMP nodes, while `no_procs` the number of MPI processes to execute. The constraint directive ensures that the correct number of SMP nodes is allocated for each number of MPI processes. We examined the application performance for two interconnection networks: Fast Ethernet and Myrinet.

For Fast Ethernet, shared memory has been used to communicate inside SMP nodes. The PBS script also assigns the path of the `mpirun` command to the MPIRUN environment variable through a *ZEN assignment directive*. The application's Makefile (see Ex. 3.3) sets the MPILIB environment variable to the corresponding MPI library. The *global constraint file* (see Ex. 3.4) ensures the correct association between the network specific (Fast Ethernet and Myrinet) MPI libraries and the corresponding `mpirun` script. The execution time (mnemonic `WTIME`) and the communication time (mnemonic `ODATA` mnemonics) for the entire program (mnemonic `CR_P`) have been measured by using a *ZEN behaviour directive* (see Ex. 3.5).

Example 3.1 (Problem size – `lapw0.def`)
```
!ZEN$ SUBSTITUTE 125hour.clmsum={ .125hour.clmsum,
     .25hour.clmsum, .5hour.clmsum, 1hour.clmsum }
8,'ktp_.125hour.clmsum','old','formatted',0
...
!ZEN$ SUBSTITUTE 125hour.struct={ .125hour.struct,
     .25hour.struct, .5hour.struct, 1hour.struct }
20,'ktp_.125hour.struct','old','formatted',0
!ZEN$ CONSTRAINT INDEX
                125hour.clmsum == 125hour.struct
```

Example 3.2 (PBS Script – `run.pbs`)
```
#!ZEN$ SUBSTITUTE nodes\=1 = { nodes={1:40} }
#PBS -l walltime=0:29:00,nodes=1:fourproc:ppn=4
cd $PBS_O_WORKDIR
#!ZEN$ ASSIGN MPIRUN = {
           /opt/local/mpich/bin/mpirun,
           /opt/local/mpich_gm/bin/mpirun.ch_gm }
#!ZEN$ SUBSTITUTE no_procs = {1:40}
$(MPIRUN) -np no_procs ../SRC/lapw0 lapw0.def
#!ZEN$ CONSTRAINT INDEX 4*(nodes\=1 -1)<no_procs
      && no_procs <= 4*nodes\=1 && no_procs != 1
```

Example 3.3 (`Makefile`)
```
#ZEN$ ASSIGN MPILIB = { /opt/local/mpich/lib,
                        /opt/local/mpich_gm/lib }
LIBS = ... -lsismpiwrapper -L$(MPILIB) -lmpich
...
$(EXEC): $(OBJS)
       $(F90) -o lapw0 $(OBJS) $(LIBS)
```

Example 3.4 (Global constraint file)
```
!ZEN$ CONSTRAINT INDEX
               Makefile:MPILIB == run.pbs:MPIRUN
```

Example 3.5 (Source file – `lapw0.F`)
```
!ZEN$ CR PMETRIC WTIME, ODATA
```

Eight ZEN directives have been inserted into 6 ZEN files, based on which a total of 320 experiments were automatically generated and executed by ZENTURIO. The Application Data Visualiser has been used to automatically generate the diagrams from Fig. 2. Fig. 2(a) shows the scalability of the application for all four problems sizes. The scalability of the algorithm improves by increasing the LAPW0 problem size (number of atoms). For a problem size of 8 atoms (`125hour.struct`) LAPW0 does not scale at all. This is partially due to the extensive communication overhead with respect to the entire execution time. Fig. 2(c) shows the contribution of each computed overhead to the overall execution time for each experiment. We could not separate the unidentified overhead from the optimal execution time (sequential time divided by the number of processes) because a sequential implementation of LAPW0 could not be run due to physical memory limitations. For 64 atoms (`1hour.struct`), the application scales well up to 16 processes, after which the execution time becomes relatively constant.

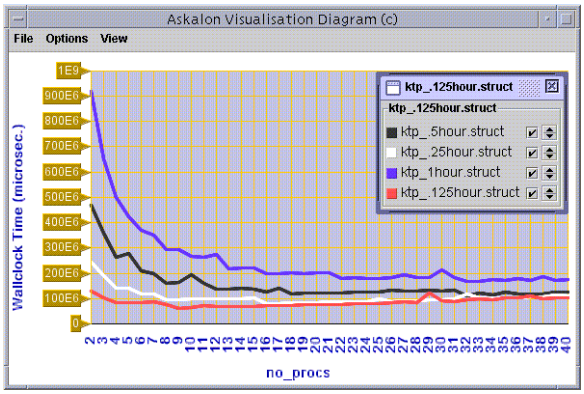
The interconnection network does not improve the communication behaviour (see Fig. 2(b)) because all receive operations are blocking, which dominates the effective transfer of the relative small amount of data among processes.

3.2. Three-Dimensional Particle-In-Cell

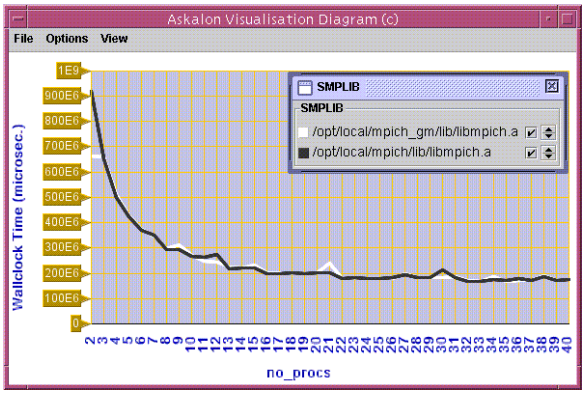
The three-dimensional Particle-In-Cell (3DPIC) application [10] simulates the interaction of high intensity ultrashot laser pulses with plasma in three dimensional geometry. 3DPIC has been implemented as a Fortran90/MPI code. We scheduled the experiments on the SMP cluster using GRAM over PBS as job manager. The application scalability has been tested on seven different machine sizes (1, 4, 9, 12, 16, 25, and 36 processes), expressed by the `count` argument of the RSL script (see Ex. 3.6). Based on the number of processes of one experiment, GRAM allocates the correct number of SMP nodes using PBS. Even though it is an MPI application, we have set the RSL job type to single, which allows us to vary the interconnection network locally within the cluster. The application is started using the shell script from Ex. 3.7, which assigns to the MPIRUN ZEN variable the path to the `mpirun` script. Similar to LAPW0, we study the impact of the communication network (Fast Ethernet and Myrinet) by annotating the application's Makefile (see Ex. 3.3). A global ZEN constraint file associates the `mpirun` command with the corresponding MPI library (see Ex. 3.8). Execution and communication time have been measured, as already shown in Ex. 3.5.

Example 3.6 (Globus RSL script – `run.rsl`)
```
+(&
 (resourceManagerContact="gescher/jobmanager-pbs")
(*ZEN$ SUBSTITUTE count\=4 =
                          {count={1,1,3,3,4,7,9}}*)
 (count=4)
 (jobtype=single)
 (directory="/home/radu/APPS/LAPW0/znse_6")
 (executable="script.sh") )
```

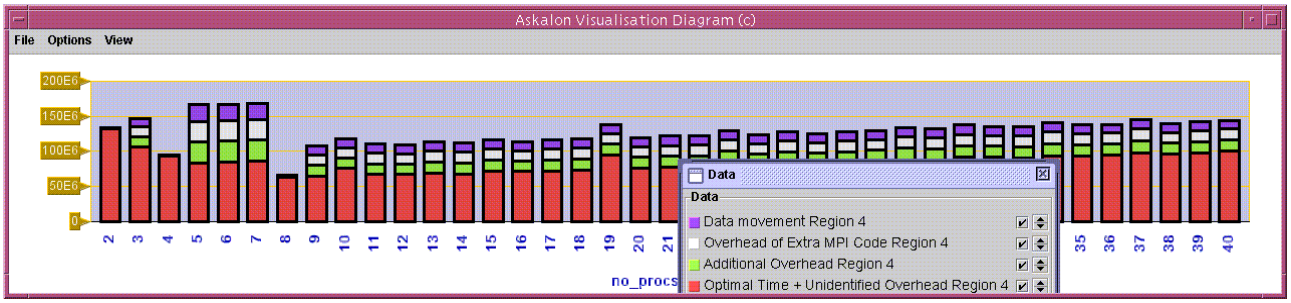
Example 3.7 (Shell script – `script.sh`)
```
#!/bin/sh
cd $PBS_O_WORKDIR
n = `wc -l < $PBS_NODEFILE`
#ZEN$ ASSIGN MPIRUN = {
           /opt/local/mpich/bin/mpirun,
           /opt/local/mpich_gm/bin/mpirun.ch_gm }
$(MPIRUN) -np $n -machinefile $PBS_NODEFILE lapw0
```



(a) Scalability study for 4 problem sizes using Fast Ethernet communication.

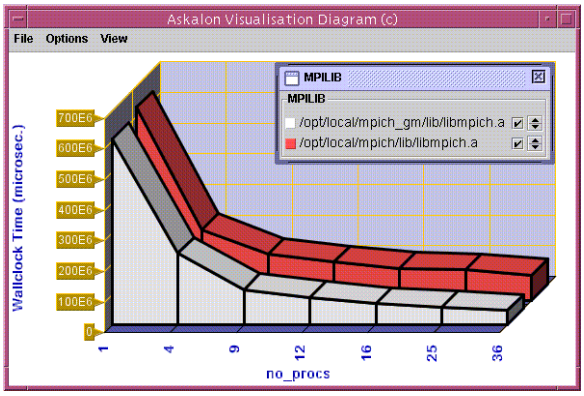


(b) Network comparison for 64 atoms problem size.

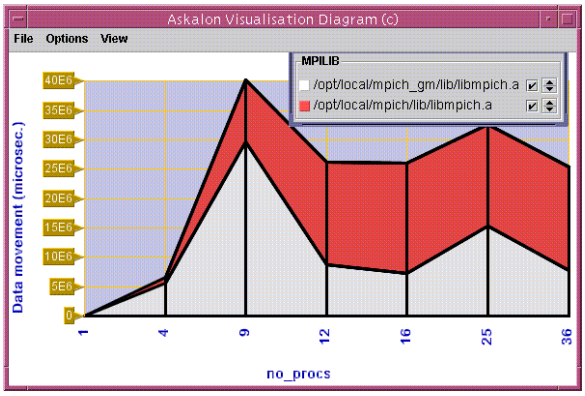


(c) Contribution of Myrinet communication metrics to the total execution time for 8 atoms problem size.

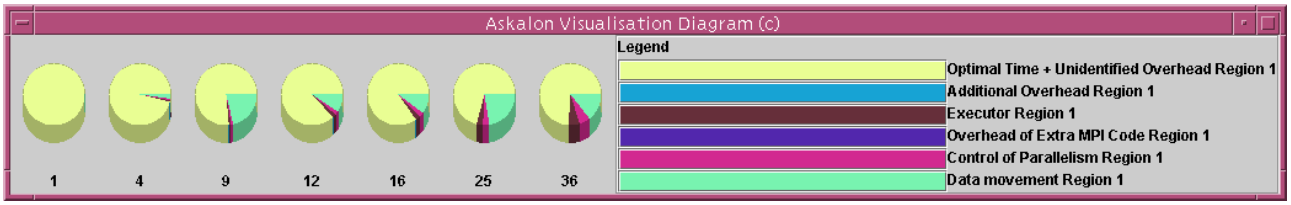
Figure 2. Visualisation Diagrams for LAPW0.



(a) Scalability study and network comparison (Fast Ethernet versus Myrinet).

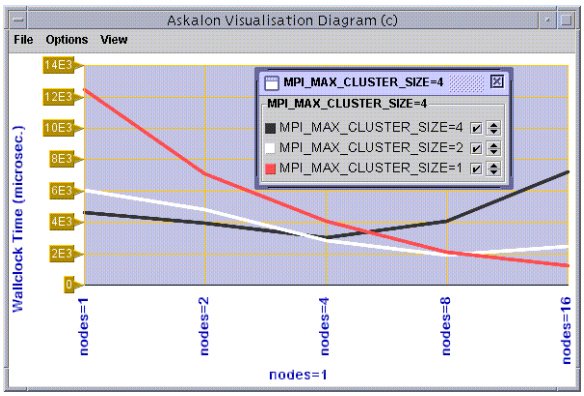


(b) Comparison of communication overhead for Fast Ethernet and Myrinet.

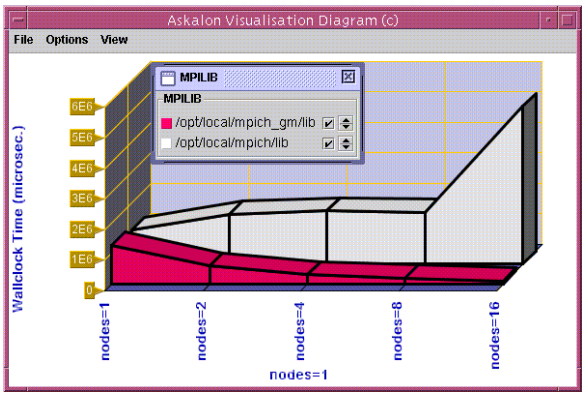


(c) Contribution of Myrinet communication metrics to the total execution time.

Figure 3. Visualisation Diagrams for 3DPIC.



(a) Scalability study for `length`=2^5 problem size, for different number of SMP nodes and MPI processes per node over Myrinet.



(b) Network comparison (Fast Ethernet versus Myrinet) for `length`=2^7 problem size using 4 MPI processes per SMP node.

Figure 4. Visualisation Diagrams for FFTW.

Example 3.8 (Global constraint file)

```
!ZEN$ CONSTRAINT INDEX
                Makefile:MPILIB==script.sh:MPIRUN
```

Five ZEN directives have been inserted into 4 files to generate fourteen experiments. Fig. 3(a) indicates a good scalability behaviour of the application. The use of Myrinet yields up to 50% improvement in performance compared to Fast Ethernet, which is explained by the reduced communication overhead (see Fig. 3(b)). Fig. 3(c) shows the relatively low ratio between the application execution time (one full pie) and the MPI overheads measured, which explains the good application scalability. As a sequential version of this application is not available, we did not separate the unidentified overhead from the optimal execution time.

3.3. FFTW

The Fastest Fourier Transform in the West (FFTW) [9] is an MPI C subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions, of both real and complex data and of arbitrary input size. We have created a Fortran90 wrapper program which calls a three-dimensional FFT, in order to be able to automatically instrument it with SCALEA.

We analysed seven problem sizes, represented by the three-dimensional square matrix size, ranging from 2^2 to 2^8 (see Ex. 3.9). We benchmark the subroutine call `fftwnd_f77_mpi` through a local ZEN behaviour directive (see Ex. 3.9). The experiments are submitted on the cluster using PBS. We run each problem size on 1, 2, 4, 8, and 16 SMP nodes by annotating the PBS script with the ZEN variable `nodes=1`. On each node we execute 1, 2 and 4 MPI processes, which we control through the `MPI_MAX_CLUSTER_SIZE` environment variable (see Ex. 3.10). The constraint directive ensures that the correct number MPI processes is started for each machine size and configuration. The application performance has been examined for two interconnection networks (Fast Ethernet and Myrinet) by annotating the application Makefile (see Ex. 3.3) and the PBS script (see Ex. 3.10), as already explained in Section 3.1. For Fast Ethernet, shared memory has been used for communication inside SMP nodes.

Example 3.9 (Source File – `benchF.f90`)

```
!ZEN$ SUBSTITUTE length\=256 = {length=2**{2:8}}
INTEGER, PARAMETER :: length=256
...
!ZEN$ CR FFTW_CALL PMETRIC WTIME BEGIN
      CALL fftwnd_f77_mpi_(plan, 1, loc, work, 1,
                           FFTW_TRANSPOSED_ORDER)
!ZEN$ END CR
```

Example 3.10 (PBS script – `run.pbs`)

```
#!ZEN$ SUBSTITUTE nodes\=1 = { nodes={1,2,4} }
#PBS -l walltime=0:29:59,nodes=1:fourproc:ppn=4
#!ZEN$ SUBSTITUTE MPI_MAX_CLUSTER_SIZE\=4 = {
                  MPI_MAX_CLUSTER_SIZE={1,2,4} }
#PBS -v MPI_MAX_CLUSTER_SIZE=4
cd $PBS_O_WORKDIR
#!ZEN$ ASSIGN MPIRUN = {
          /opt/local/mpich/bin/mpirun,
          /opt/local/mpich_gm/bin/mpirun.ch_gm }
#!ZEN$ SUBSTITUTE no_procs = {1,2,4,8,16}
mpirun -np no_procs benchF
#!ZEN$ CONSTRAINT INDEX 2^(nodes\=1 - 1) *
    2^(MPI_MAX_CLUSTER_SIZE\=4-1)==2^(no_procs-1)
```

Nine ZEN directives have been inserted into four files to generate a total of 210 experiments that have been automatically generated and conducted by ZENTURIO. Fig. 4 shows two samples from the variety of diagrams that have been automatically generated from the data stored in the data repository. It is well known that the performance of FFTW is dependent on enough work per processor and on fast networks. For a 2^5 problem size, FFTW scales well over Myrinet up to 16 MPI processes, beyond which the performance degrades (see Fig. 4(a)). For a large 2^7 matrix size, FFTW

performs by far better over Myrinet than over Fast Ethernet (see Fig. 4(b)). For this large problem size FFTW scales well over Myrinet, while Fast Ethernet produces a complete non-scalability.

3.4. Backward Pricing

The backward pricing kernel [5] is a parallel implementation of the backward induction algorithm which computes the price of an interest rate dependent financial product, such as a variable coupon bond. It is based on the Hull and White trinomial interest rate tree which models future developments of interest rates. We have performed a performance and a parameter study for this code.

3.4.1 Performance Study

Backward pricing has been encoded as an HPF+ application which uses HPF+ directives to distribute the data onto the SMP nodes. The application is compiled into a mixed OpenMP/MPI program by the SCALEA instrumentation engine built on top of the HPF+ Vienna Fortran Compiler (VFC). Intra-node parallelisation is achieved through OpenMP directives. Communication among SMP nodes is realised through MPI calls. We scheduled the experiments on the SMP cluster using GRAM. We annotated the RSL script to vary the machine size from 1 to 10 SMP nodes (see Ex. 3.11). The ZEN variable `count=4` is set with the number of SMP nodes. Based on the `count` RSL parameter, GRAM allocates the corresponding number of SMP nodes and uses an available local MPI implementation, which must be defined by the user default shell environment. In the current experiment, we have set our environment for MPICH using the p4 device over Fast Ethernet. The `MPI_MAX_CLUSTER_SIZE` environment variable ensures that the `mpirun` script starts only one MPI process per SMP node. The intra-node parallelisation is achieved by means of OpenMP. The application has been encoded such that it reads the number of OpenMP threads to be forked per SMP node from a file called `bench.in`. We annotated this input file with a ZEN substitute directive which varies the number of threads from 1 to 4 (see Ex. 3.12). The overall execution time, as well as the MPI communication and the HPF+ inspector/executor overheads have been measured.

Example 3.11 (Globus RSL script – `run.rsl`)

```
+(&
 (resourceManagerContact="gescher/jobmanager-pbs")
(*ZEN$ SUBSTITUTE count\=4 = {count={1:10}}*)
 (count=4)
 (jobtype=mpi)
 (environment=(MPI_MAX_CLUSTER_SIZE 1))
 (directory="/home/radu/APPS/HANS")
 (executable="bw_halo_sis") )
```

Example 3.12 (Input file – `bench.in`)

```
!ZEN$ SUBSTITUTE threads = { 1:4 }
threads
```

Two ZEN directives have been inserted into two files to produce 40 experiments automatically conducted by ZENTURIO. Fig. 5(a) shows a good scalability of this code. Backward pricing is a computational intensive application, which highly benefits from the inter-node MPI and intra-node OpenMP parallelisation. The overall wallclock time of the application significantly improves by encreasing the number of nodes and OpenMP threads per SMP node. Fig. 5(b) shows a very high ratio between the application total execution user time (one full bar) and the HPF and MPI overheads measured, which explains the good parallel behaviour. This ratio decreases for a high number of SMP nodes, for which the overheads significantly degrade the overall performance.

3.4.2 Parameter Study

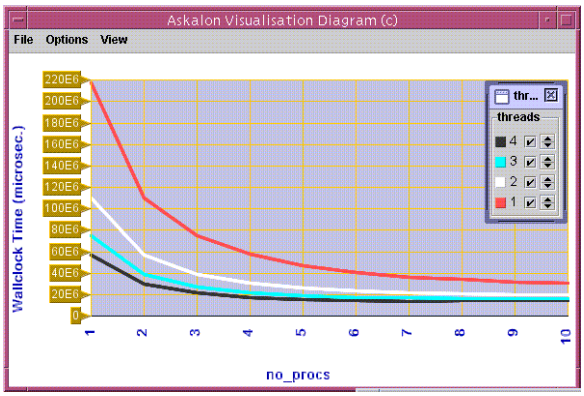
We performed two parameter studies for the Backward pricing code by varying four input parameters: (1) the coupon bond (ZEN variable `coupon` from 0.01 to 0.1 with increment 0.001); (2) the number of time steps, over which the price is computed (ZEN variable `nr_steps` from 5 to 60 with increment 5); (3) the coupon bond's end time (ZEN variable `bond%end`), which must be equal to the number of time steps; (4) the length of one time step (ZEN variable `delta_t` from 1/12 to 1 with increment 1/12). The application has been encoded such that it reads its input parameters from different input data files. ZEN assignment directives are inserted in the source code immediately after the corresponding `read` statements (see Ex. 3.13). Thus, the original `read` statement is made redundant. A constraint directive guarantees that the coupon bond's end time is identical with the number of time steps. We examined the effects of these input parameters on the total price output result.

Example 3.13 (Source file – `bw_halo.f90`)

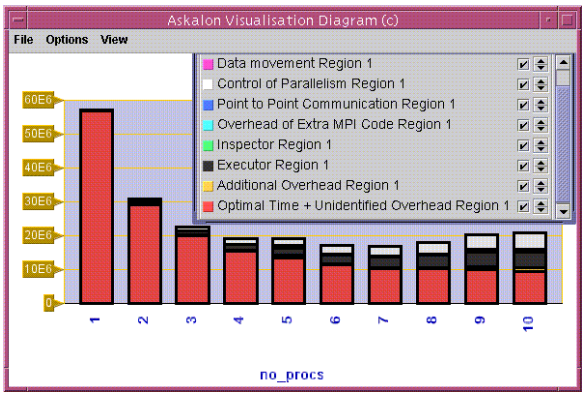
```
read(10,*) nr_steps
!ZEN$ ASSIGN nr_steps = { 5:60:5 }
...
read(10,*) delta_t
!ZEN$ ASSIGN delta_t = { 0.08, 0.17, 0.25, 0.33,
    0.42, 0.5, 0.58, 0.67, 0.75, 0.83, 0.92, 1 }
...
read(10,*) bond%end
!ZEN$ ASSIGN bond\%end = { 5:60:5 }
!ZEN$ CONSTRAINT VALUE nr_steps == bond\%end
...
read(10,*) bond%coupon
!ZEN$ ASSIGN bond\%coupon = { 0.01:0.1:0.001 }
```

A total number of 1481 experiments have been automatically generated. The output of each completed experiment containing the total price has been stored into the data repository. Two sample diagrams are depicted in Fig. 5.

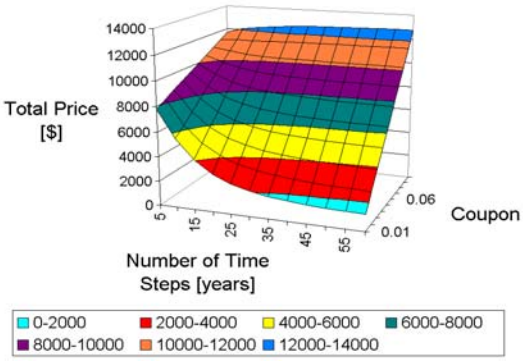
The three-dimensional surface in Fig. 5(c) shows the evolution of the total price as a function of the number of time steps and the coupon, with the following meaning: (a) the



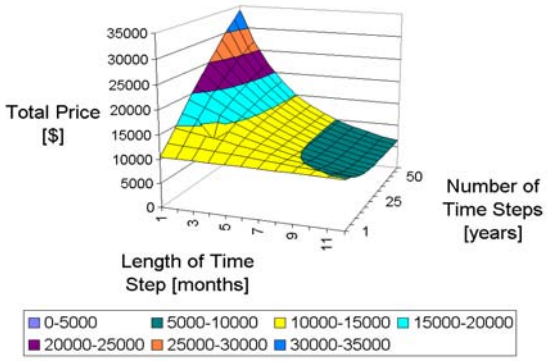
(a) Wallclock time for different number of nodes and OpenMP threads per SMP node.



(b) Contribution of MPI and HPF overheads to the overall execution time (4 threads per SMP node).



(c) Total price for length of time step (`delta_t`) = 1.0.



(d) Total price for coupon = 0.05.

Figure 5. Visualisation Diagrams for Backward Pricing.

price decreases with the maturity (number of time steps $\times$ length of time step), because the effect of discounting future payments increases (i.e. \$100 in 20 years are less then \$100 in 10 years), but only if the coupon is less than the interest rates (e.g. for 0.06, the coupon rate is greater than the interest rates); (b) the price increases with coupon, because the higher the coupon rate is, the higher the future payments are; (c) for very large maturities, the price linearly depends only on the coupon.

Fig. 5(d) shows the price evolution by varying the number of time steps and the length of a time step, with the following meaning: (a) the price decreases with the length of a time step, because a smaller payment number implies less money in the future; (b) depending on the number of time steps, the price may increase or decrease with maturity, depending on how much the smaller number of payments are compensated by smaller discount effects.

4. Conclusions

ZENTURIO has been purposely designed as a generic experiment management tool to support scientists and engineers in their effort to conduct large scale parameter studies, performance analysis, and software testings on cluster and Grid architectures. We have shown a few realistic scenarios of using ZENTURIO for performance and parameter studies of four parallel applications. A ZEN directive-based language is used to define values of interest for any application parameter in any source or input file of the application. Parametrisation is therefore no longer restricted to input files only, as in existing conventional parameter study tools [1, 18]. We have shown the use of ZEN directives for varying arbitrary application input files, input variables, machine sizes for an SMP cluster (both the number of nodes and the number of threads per SMP node), and communication networks (Fast Ethernet and Myrinet) within MPI. For each application we have used a small number of ZEN directives (regularly less than 10) to annotate arbitrary application source and input files and specify a few hundreds (up to even thousands) of experiments for each application. We have integrated a full performance analysis tool [17], based on a full-fledged Fortran 90, HPF, and OpenMP compiler front-end, providing powerful high-level overhead analysis. The experiments have been conducted (i.e. transferred to the target machine, compiled, and executed) by ZENTURIO fully

automatically. PBS for cluster and GRAM for Grid computing have been used to submit the jobs to an SMP cluster. For each completed experiment, the performance and output data are automatically stored into a data repository. An Application Data Visualiser, built on top of the Askalon visualisation diagrams [6], provides advanced techniques to display the performance and output data across multiple experiments, by mapping application parameters to visualisation axis. A wide variety of visualisation diagrams are supported, including linechart, barchart, piechart, and surface.

Future work involves the development of a Grid-oriented architecture based on the Web services technology and the Open Grid Service Architecture (OGSA) [8] specification.

Acknowledgement. We thank Andreas Bonelli from the Technical University of Vienna for providing the Fortran wrapper to the FFTW application.

References

[1] 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.

[2] Peter Blaha, Karlheinz Schwarz, and Joachim Luitz. *WIEN97: A Full Potential Linearized Augmented Plane Wave Package for Calculating Crystal Properties*. Institute of Physical and Theoretical Chemistry, April 2000.

[3] Michel Courson, Alan Mink, Guillaume Marcais, and Benjamin Traverse. An automated benchmarking toolset. In *HPCN Europe*, pages 497–506, 2000.

[4] J. Davison de St. Germain, Alan Morris, Steven G. Parker, Allen D. Malony, and Sameer Shende. Integrating performance analysis in the uintah software development cycle. In *Proceedings of the The fourth International Symposium on High Performance Computing (ISHPC-IV)*, pages 190–206, Kansai Science City, Japan, 2002.

[5] E. Dockner and H. Moritsch. Pricing Constant Maturity Floaters with Embeeded Options Using Monte Carlo Simulation. Technical Report AuR_99-04, AURORA Technical Reports, University of Vienna, January 1999.

[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] Ken Ferschweiler, Mariacarla Calzarossa, Cherri Pancake, Daniele Tessera, and Dylan Keon. A community databank for performance tracefiles. In Y. Cotronis and J. Dongarra, editors, *Euro PVM/MPI*, pages 233–240. Springer-Verlag, 2001. Lect. Notes Comput. Sci. vol. 2131.

[8] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. The Globus Project and The Global Grid Forum, January 2002. `http://www.globus.org/research/papers/OGSA.pdf`.

[9] M. Frigo and S. G. Johnson. FFTW: An adaptive software architecture for the FFT. In *Proceedings of the International Conference on Acoustics Speech and Signal Processing (ICASSP)*, pages 1381–1384, 1998.

[10] M. Geissler. *Interaction of High Intensity Ultrashort Laser Pulses with Plasmas*. PhD thesis, Vienna University of Technology, 2001.

[11] Yannis E. Ioannidis, Miron Livny, S. Gupta, and Nagavamsi Ponnekanti. ZOO: A desktop experiment management environment. In T. M. Vijayaraman, Alejandro P. Buchmann, C. Mohan, and Nandlal L. Sarda, editors, *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases*, pages 274–285, Mumbai (Bombay), India, 3–6 September 1996. Morgan Kaufmann.

[12] Karen L. Karavanic and Barton P. Miller. Experiment management support for performance tuning. In ACM, editor, *Proceedings of the SC'97 Conference*, San Jose, California, USA, November 1997. ACM Press and IEEE Computer Society Press.

[13] 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 Processing (ICPP-02)*, Vancouver, Canada, August 2002. IEEE Computer Society Press.

[14] Radu Prodan and Thomas Fahringer. ZENTURIO: An Experiment Management System for Cluster and Grid Computing. In *Proceedings of the 4th International Conference on Cluster Computing (CLUSTER 2002)*, Chicago, USA, September 2002. IEEE Computer Society Press. http://www.par.univie.ac.at/project/zenturio.

[15] R. Reussner, P. Sanders, L. Prechelt, and M. Mueller. SKaMPI: A detailed, accurate MPI benchmark. *Lecture Notes in Computer Science*, 1497:52, 1998.

[16] M. Romberg. The UNICORE architecture: Seamless access to distributed resources. *Proceedings of the 8th International Symposium on High Performance Distributed Computing HPDC-8*, pages 287–293, August 1999.

[17] Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Distributed and Parallel Program. In *8th International Europar Conference(EuroPar 2002)*, Lecture Notes in Computer Science, Paderborn, Germany, August 2002. Springer-Verlag.

[18] M. Yarrow, K. M. McCann, R. Biswas, and R. F. Van der Wijngaart. Ilab: An advanced user interface approach for complex parameter study process specification on the information power grid. In *Proceedings of Grid 2000: International Workshop on Grid Computing*, Bangalore, India, December 2000. ACM Press and IEEE Computer Society Press.