# On Using ZENTURIO for Performance and Parameter Studies on Clusters and Grids*

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

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

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

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

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

## Abstract

*Over the last decade, a dramatic increase has been observed in the ability to generate and store data in the course of parameter studies, performance analysis, and software testing. We have developed the ZENTURIO experiment management system for performance and parameter studies, as well as for software testing on cluster and Grid architectures. ZENTURIO uses a novel directive-based language named ZEN to specify arbitrary application parameters and a broad range performance metrics. A large number of experiments are automatically generated based on the ZEN directives which annotate arbitrary application files. A graphical user portal enables the user to create, launch, and monitor the experiments. Moreover, a limited fault tolerance mechanism has been included to restart experiments in case of casual system, job scheduler, or application faults. Graphical tools are used to automatically display output and performance data across multiple experiments. In this paper we describe our experience with ZENTURIO for performance and parameter studies of a material science kernel, a 3-dim. particle-in-cell simulation, a fast fourier transform, and a financial modeling application. Experiments have been conducted on a SMP cluster architecture with different communication networks, using PBS (Portable Batch System) and GRAM (Globus Resource Allocation Manager) as job managers.*

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

# 1 Introduction

The development and execution management of scientific and engineering applications on complex, heterogeneous and non-dedicated distributed architectures, ranging from cluster architectures [3] to widely distributed Grid structures [12], is a tedious, time-consuming, and error-prone process. Commonly these efforts require to conduct a large number of executions for a given application. Unfortunately, there has been very little work done so far that adequately supports experiment management through high-level specialised tools.

In previous work [17] we introduced the ZENTURIO experiment management system for cluster and Grid computing that alleviates parameter studies, performance analysis, and software testing. ZENTURIO defines a directive-based language called ZEN [16] to specify arbitrarily complex application and execution parameters. ZEN enables the programmer to invoke experiments for arbitrary value ranges of any problem parameter, including program variables, file names, compiler options, target machines, machine sizes, scheduling strategies, data distributions, etc. Moreover, ZEN directives can be used to request for a large variety of performance metrics (e.g. cache misses, load imballance, execution, communication, synchronisation time). Based on a ZEN-annotated application, ZENTURIO automatically generates and conducts the full set of experiments. A graphical-based user portal displays an updated view of experiments, as they progress. Upon their completion, the experiments output and performance data are automatically stored into an experiment data repository for post-mortem analysis. Visualisation tools are provided to automatically visualise the data across multiple experiments.

This paper evaluates the usefulness of ZENTURIO for performance and parameter studies of a variety of applications. Our goal is to demonstrate that ZENTURIO is a generic tool that provides a sophisticated infrastructure and user portal to automate and control experiments for sequential, parallel, and distributed applications.

The remainder of this paper is organised as follows. The next section outlines some related work. Section 3 gives an overview of the functionality of ZENTURIO. Experiments with using ZENTURIO on four AURORA applications are presented in Section 4. Section 5 concludes the paper.

# 2 Related Work

Our work on a system design for automatic experiment management is centered around three different research areas: scientific experiment management, performance analysis, and parameter studies.

The IST APART working group developed as part of Workpackage 3 [15] a generic design of an automatic performance analysis system and defined and categorised performance analysis experiments.

The Paradyn performance analysis tools [14] supports performance experiment management through techniques for quantitatively comparing several experiments and performance diagnosis based on dynamic instrumentation. Experiments have to be set up manually, whereas performance analysis is done automatically for every different experiment.

The ZOO project [13] has been initiated to support 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] is a tool that manages the execution of parameter studies. A parameterised experiment is specified by a declarative plan file. Nimrod generates one job for each unique combination of parameter values, by taking the cartesian product of all values. Unlike in ZENTURIO, the set of possible parameter value combinations cannot be restricted. Moreover, Nimrod can only manage already compiled applications and parameters outside of source files.

The ILAB [23] project controls parameter studies through a graphical user interface for input files. Several job execution models are provided for interacting with different schedulers, for instance, PBS and Globus schedulers.

The Unicore project [19] facilitates the usage of supercomputers on the Internet. Unicore jobs are organised manually as a directed acyclic graph, consisting of job groups and tasks. Such jobs comprise file transfers and binary/script file executions, including compilation and linking tasks. There is no support for automatic invocation of multiple experiments and performance analysis.

SKaMPI [18] provides a benchmarking environment for MPI codes, with the goal of analysing the runtime of MPI operations. A runtime parameter file allows the description of a suite of measurements based on specific parameter values. A public performance database allows storage of benchmark data and interactive comparison of various MPI performance aspects across different implementations and platforms.

# 3 ZENTURIO

ZENTURIO [17] is a tool purposely designed to automatically conduct a large number of experiments on cluster and Grid architectures. Its goal is to support performance analysis and tuning, parameter studies, and software testing.

Conventional parameter study tools [1, 23] commonly restrict parameterisation to input files only. In contrast, ZENTURIO uses a directive-based language called ZEN [16] to annotate arbitrary files. ZEN directives are used to assign value sets to so called ZEN variables (e.g. MPIRUN in Ex. 4.2, MPILIB in Ex. 4.3). A ZEN variable can represent any problem, system, or machine parameter, including program variables, file names, compiler options, target machines, machine sizes, scheduling strategies, data distributions, etc. The value set represents the list of interesting values for the corresponding parameter. There are four kinds of ZEN directives. The *ZEN substitute directive* uses a pre-processor-based string replacement mechanism to substitute a ZEN variable with one element from its value set (see Ex. 4.1 and 4.2). To complement some of the ZEN substitute directive's limitations, a *ZEN assignment directive* can be used to insert an assignment statement in the code (see Ex. 4.2

and 4.3). By default, the cartesian product of the value sets of all ZEN variables is computed. The number of possible value set combinations can be limited by means of *ZEN constraint directives* (see Ex. 4.1 and 4.4). ZEN also supports the specification of performance metrics for arbitrary code regions through the *ZEN performance behaviour directives* (see Ex. 4.5).

A file/application annotated with ZEN directives is called ZEN file/application. A *ZEN Transformation System* (ZTS) generates all ZEN file instances for a ZEN file, based on the ZEN directives inserted (see Fig. 1). The SCALEA [20] instrumentation system, which provides a complete Fortran90 OpenMP, MPI, and HPF front-end and unparser, is used to instrument the application for performance metrics. The ZEN performance behaviour directives are translated to SCALEA directives and command-line options. An Experiment Generator computes the cartesian product of all ZEN file instances to generate the corresponding ZEN application instances. A detailed description of ZEN is given in [16].

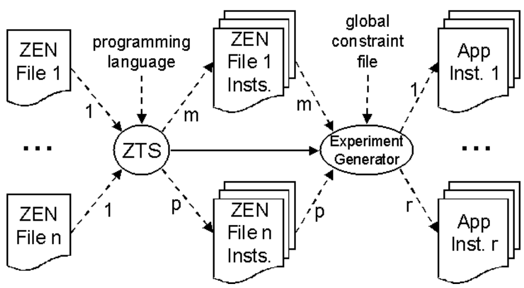



**Figure 1. ZEN application instance generation.**

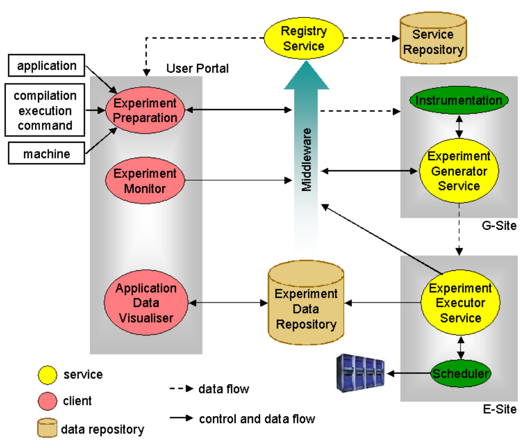



**Figure 2. The architecture of ZENTURIO.**

ZENTURIO is designed as a distributed service architecture, implemented based on the Jini distributed technology [8] (see Fig. 2). It consists of four services, distributed over several sites. A site is a host on the Internet where a service is installed and which can be remotely accessed.

The *User Portal* (UP) is a GUI-based client, which represents the entry point of the system. It commonly resides on the user's local machine. The UP consists of an Experiment Preparation (EP) client which supports the preparation of an application for execution, and an Experiment Monitor (EM) client which submits and monitors experiments. Through the EP, ZENTURIO receives as input a ZEN application (including the application source and the set of input and output files), one compilation and one execution command, and the machine name on which to execute the experiments of interest. A snapshot of a ZENTURIO UP monitoring an application is depicted in Fig 3.

These inputs are transferred to the Experiment Generator (EG) service at the G-Site. Based on the ZEN directives, EG automatically instruments the application and generates all application instances and the corresponding experiments (see Fig. 1). An *experiment* is an entity which is fully specified by an application instance, a compilation and an execution command, a compilation and an execution directory, and a set of output file names.

After the experiments have been generated, they are transferred to the E-site where an Experiment Executor (EE) service resides. The EE is a generic service responsible for compiling, executing, and managing the execution of applications on a target machine. EE interacts at the back-end with a scheduler. The current implementation of ZENTURIO supports PBS [21] for cluster and GRAM/DUROC [6] for Grid (Globus) applications. Transferring and compiling applications are optional tasks that can be omitted, for instance, when the client (site of user) and the target machines share a file system and the application has already been compiled.

ZENTURIO defines two types of output results for an experiment. The performance metrics (e.g. execution time, synchronisation, communication) are computed based on the ZEN behaviour directives by a post-mortem performance analysis module of SCALEA. The output files are indicated by the user in the experiment preparation phase. Upon experiment termination, both performance metrics and output files are stored into the Experiment Data Repository. The *Application Data Visualiser (ADV)* can be employed post-mortem by the user to automatically visualise performance and output data across a certain subset of experiments that have been executed for a ZEN application. The types of visualisation supported include linechart, barchart, piechart, and surface.

Fig. 4 shows the ADV dialog box for constructing a multi-experiment visualisation. The top left panel shows various performance metrics (e.g. barrier, collective communication, control of parallelism) which can be selected for visualisation. There are two views provided by ZENTURIO: (1) *Metric-to-Region* (shown in Fig. 4) displays the list of metrics computed. A set of regions is tabulated for which this metric was greater than zero; (2) *Region-to-Metric* (not shown in the paper) displays a hierarchy of code regions, for which a computed metric can be selected. Single or multiple metric selection is allowed. The top-right panel always displays the code region corresponding to the last selected metric. The bottom panel displays all ZEN variables of the ZEN application. Every ZEN variable has an associated list box that shows its value set. The visualisation is generated for all experiments whose ZEN variables match the selected values.

We have defined two special values for ZEN variable values:

**WildCard** matches any ZEN variable value. The corresponding ZEN variable is selected as a visualisation axis. For

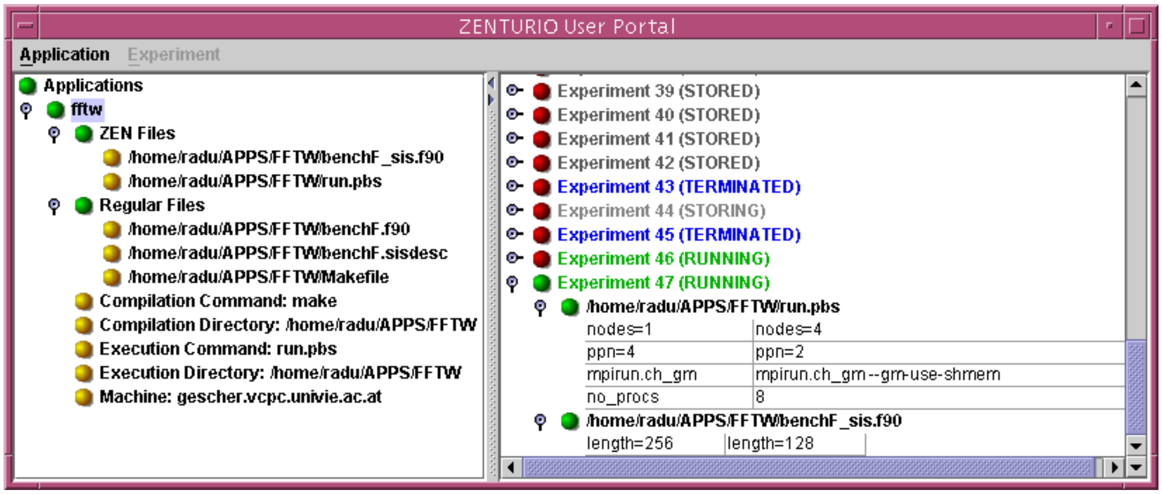



**Figure 3. ZENTURIO User Portal (UP) that shows the status of a running FFTW application (see Section 4.3).**

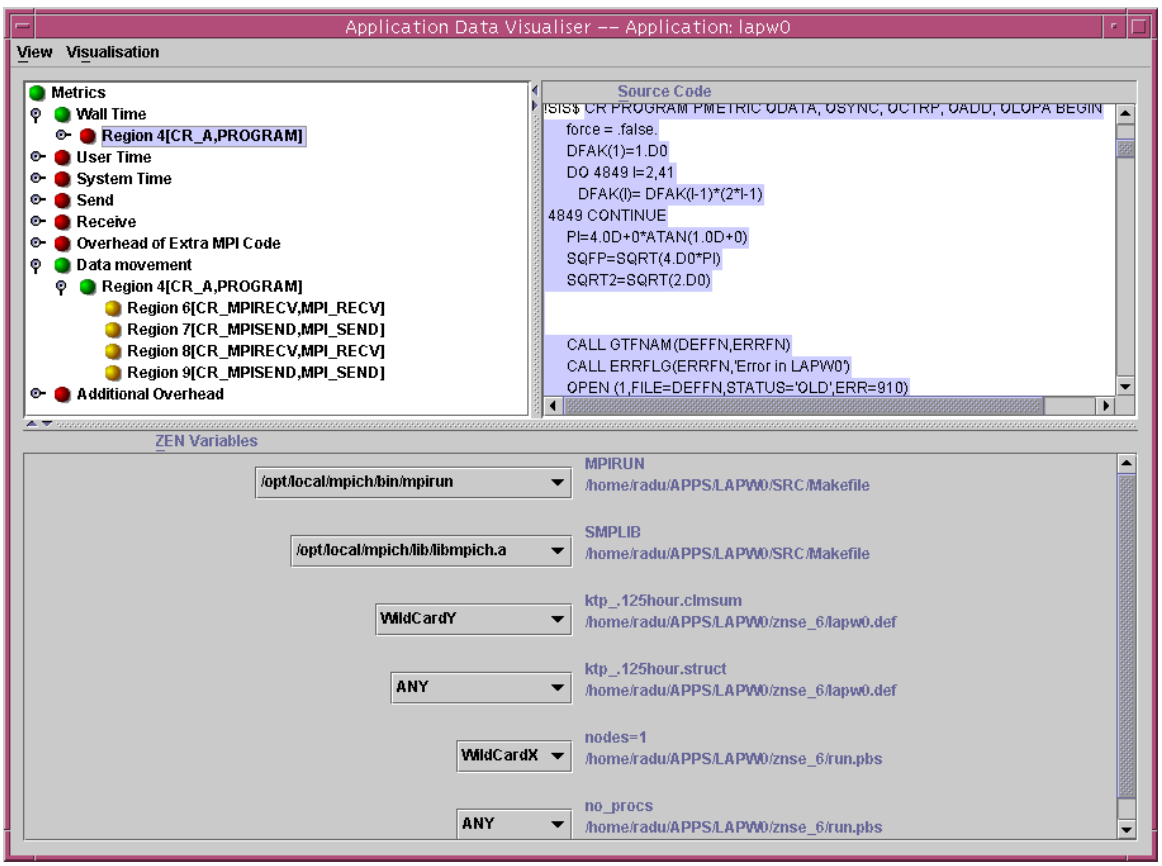



**Figure 4. The Application Data Visualiser (ADV) dialog-box which generates the 3-dim. diagram from Fig. 5(a) for the LAPW0 application (see Section 4.1). The two wild card selections indicate the two visualisation axis: number of MPI processes (ZEN variable** `no_procs`**) and the problem size (ZEN variable** `ktp_.125hour.clmsum`**). The ZEN variables** `nodes=1` **and** `ktp_.125hour.clmsum` **are given the** `ANY` **value because they are bound to** `no_procs` **and** `ktp_.125hour.struct` **via a ZEN constraint, respectively. The top-left panel contains a Metric-Region display of the program metrics, from which by the overall wallclock time was selected as the third visualisation axis. Linechart was chosen as visualisation diagram from the Visualisation menu.**

the moment a maximum of three wild cards is allowed. One, two, or three wild card selections, respectively imply a 2-dim., 3-dim. (see Fig. 5(a)), or 4-dim. (see Fig. 5(c)) visualisation. To control the axis onto which the values of a ZEN variable are mapped, we support three options for wild cards: *WildCardX*, *WildCardY*, and *WildCardZ* (i.e. ZEN variable is associated with the X-axis, Y-axis, respectively Z-axis). The metric value always constitutes one axis in the resulting visualisation diagram. In case of a multiple metric selection, a corresponding wild card must be selected from the menu. Otherwise the last selected metric is visualised only. Currently, we are in the process of integrating a 6D hierarchy-chart, which would allow the representation of two additional wild cards.

**ANY** matches any ZEN variable name. ANY is different from a wild card in the sense that the corresponding ZEN variable is not selected as a visualisation axis. This is useful when a ZEN variable is bound to another ZEN variable (which was selected as wild card) via a ZEN constraint.

A similar dialog box is used to visualise output parameters across multiple experiments. For this purpose a program must write the output data into a file or to standard output. The user selects a file name (or standard output) which corresponds to the output parameter to be visualised. By restricting the ZEN variable values as done for the performance metric visualisation, we choose the experiment subset for which the output parameter is visualised. Fig. 8(c) and 8(d) (see Section 4.4.2) contain two sample visualisation diagrams of a single output parameter.

A detailed description of ZENTURIO is given in [17].

# 4 Experiments

In this section we describe several performance and parameter studies that have been conducted under ZENTURIO for a variety of parallel and distributed applications. All experiments have been run on a cluster of SMP nodes, each node consisting of 4 Intel Pentium III Xeon 700 MHz CPU, with 1MB full-speed L2 cache, 2 GByte ECC RAM, running Linux 2.2.18-SMP. This clusters supports two interconnection networks, Fast Ethernet and Myrinet. Globus GRAM or PBS have been used as job schedulers.

## 4.1 LAPW0

LAPW0 is a material science kernel, part of the Wien97 package [5], that calculates the potential of the Kohn-Sham eigenvalue problem. LAPW0 is implemented as a Fortran90 MPI program. Experiments were scheduled on the SMP cluster using PBS [21]. 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. 4.1). 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. 4.2). `nodes=1` controls the number of SMP nodes, while `no_procs` the number of MPI processes to execute. The constraint directive ensures that the right number of SMP nodes is allocated for a each number of MPI processes. The application cannot be executed on one single process because of physical memory limitations. 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. The application's Makefile (see Ex. 4.3) sets the MPILIB environment variable to the corresponding MPI library. The global constraint file (see Ex. 4.4) 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`) were measured by using ZEN behaviour directives (see Ex. 4.5).

**Example 4.1 (Problem size – `lapw0.def`)**

```
4,'znse_6.inm','unknown','formatted',0
...
!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
...
58,'znse_6.vint','unknown','formatted',0
!ZEN$ CONSTRAINT INDEX 125hour.clmsum==125hour.struct
```

**Example 4.2 (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 4.3 (`Makefile`)**

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

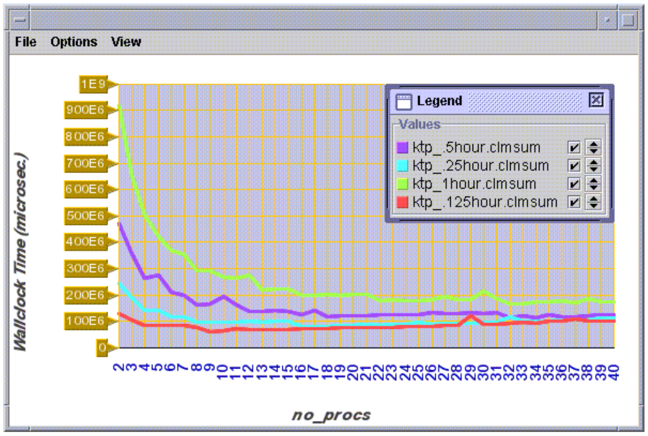
**Example 4.4 (Global constraint file)**

```
!ZEN$ CONSTRAINT INDEX Makefile:MPILIB==run.pbs:MPIRUN
```

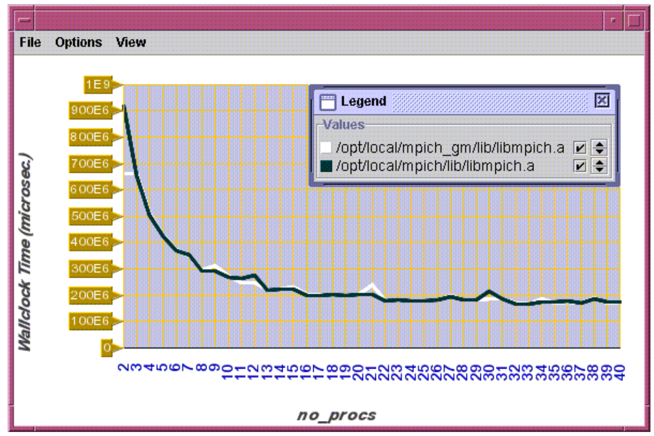
**Example 4.5 (Source file – `lapw0.F`)**

```
!ZEN$ CR PMETRIC WTIME, ODATA
```

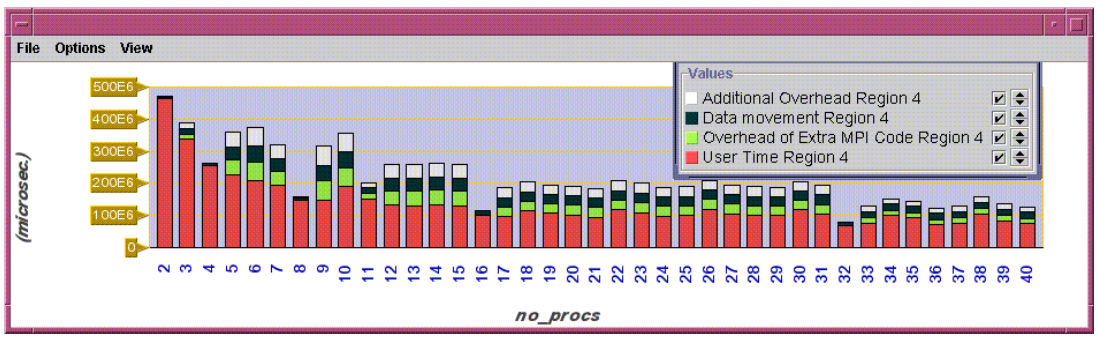
8 ZEN directives were inserted into 6 ZEN files, based on which a total of 320 experiments were automatically generated and executed by ZENTURIO. ADV was used to automatically generate the diagrams from Fig. 5. Fig. 5(a) shows the scalability of the application for all four problems sizes. The scalability of the algorithm improves by increasing the LAPW0 problem size (number of atoms). For a problem size of 8 atoms




(a) Wallclock time for 4 Problem Sizes using Fast Ethernet for inter-node communication. 3-dim. diagram, no_procs=WildCardX, ktp_.125hour.clmsum=WildCardY.




(b) Network Comparison for 64 atoms problem size (125hour.struct=1hour.struct). 3-dim. diagram, no_procs=WildCardX, MPILIB=WildCardY.




(c) Communication metrics for 8 atoms problem size (125hour.struct=125hour.struct) using Myrinet communication. 3-dim. Diagram, Metric=WildCardX, no_procs=WildCardY.

**Figure 5. ADV Diagrams for LAPW0.**

(125hour.struct=125hour.struct) LAPW0 does not scale at all. This is partially due to extensive communication overhead with respect to the entire execution time (see Fig. 5(c)). For 64 atoms (125hour.struct=1hour.struct) the application scales well up to 15 processors, after which the execution time becomes relatively constant.

The interconnection network does not improve the communication behaviour (see Fig. 5(b)) because all receive operations are blocking, which dominates the effective transfer of the relative small amount of data among processes.

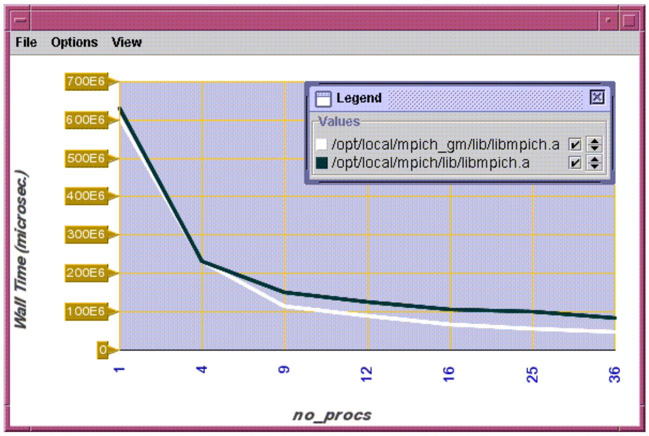
## 4.2 3-Dimensional Particle-In-Cell (3DPIC)

The 3-dim. Particle-In-Cell application [11] 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 has been coded such that in can run on 7 different machine sizes only, namely 1, 4, 9, 12, 16, 25, and 36 processes. This is expressed by the count argument of the RSL script (see Ex. 4.6).

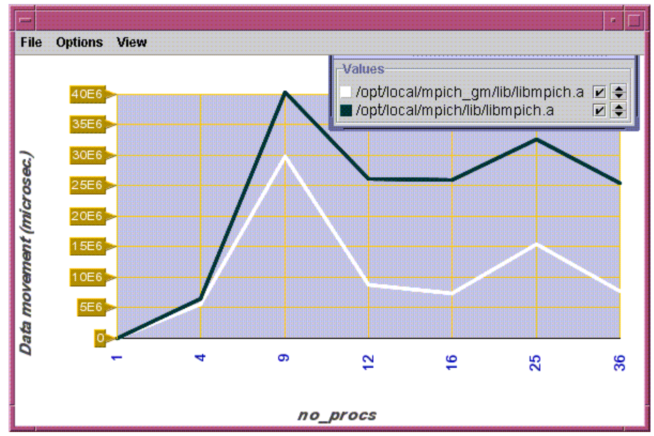
Based on the number of processes of one experiment, GRAM allocates the correct number of SMP nodes using PBS. The application is started using the shell script from Ex. 4.7, which sets the MPIRUN ZEN variable with the path to the mpirun script. Similar to LAPW0, we study the impact of the communication network (Fast Ethernet and Myrinet) by annotating the application's Makefile (see Ex. 4.3). A global ZEN constraint file associates the mpirun command with the corresponding MPI library (see Ex. 4.8). Execution and communication time have been measured, as already shown in Ex. 4.5.

**Example 4.6 (Globus RSL script – run.rsl)**

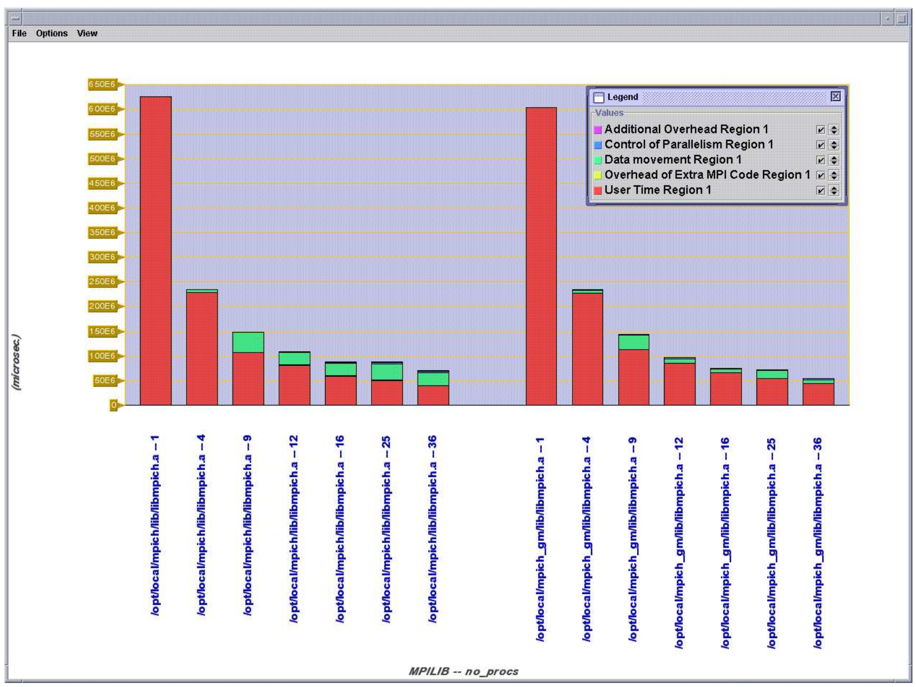
```
+
(&
 (resourceManagerContact="gescher/jobmanager-pbs")
(*ZEN$ SUBSTITUTE count\=4 ={count={1,1,3,3,4,7,9}}*)
 (count=4)
 (jobtype=single)
 (directory="/home/radu/APPS/LAPW0/znse_6")
 (executable="script.sh")
)
```




(a) Wallclock time network comparison, 3-dim. diagram, `no_procs=WildCardX`, `MPILIB=WildCardY`.




(b) Data movement network comparison, 3-dim. diagram, `no_procs=WildCardX`, `MPILIB=WildCardY`.




(c) Communication metrics, 4-dim. diagram, `Metric=WildCardX`, `no_procs=WildCardY`, `MPILIB=WildCardZ`.

**Figure 6. ADV Diagrams for 3DPIC.**

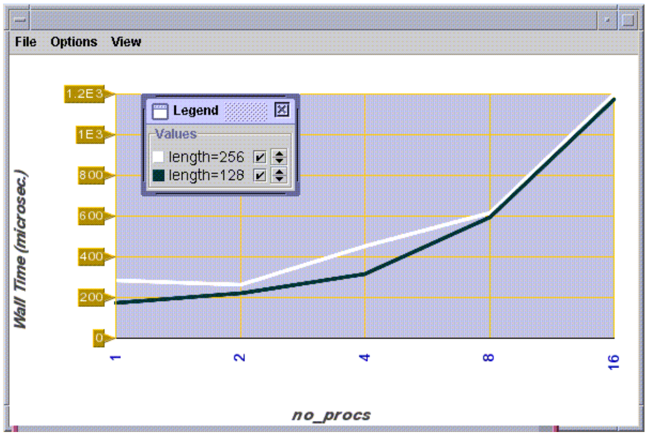
**Example 4.7 (Shell script – `script.sh`)**

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

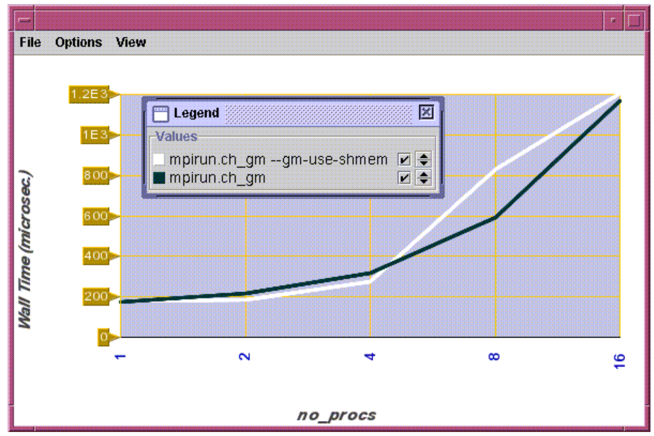
**Example 4.8 (Global constraint file)**

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

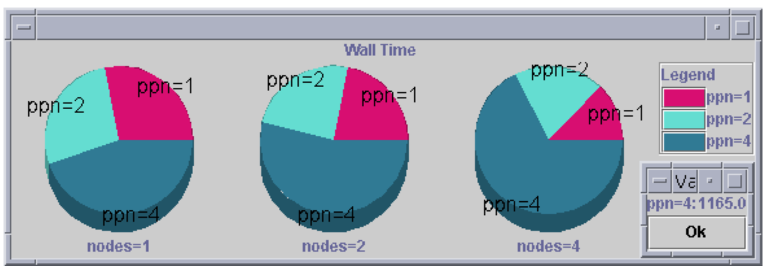
4 ZEN directives have been inserted into 4 files to generate 14 experiments. Fig. 6(a) indicates a good scalability behaviour of the application. The use of Myrinet yields up to 50% improvement in performance, compared to Fast Ethernet, which is explained by the reduced communication overhead (see Fig. 6(b)). Figure 6(c) shows the low ratio between the application user time and the MPI overheads measured, which explains the good application scalability. The user time is nearly equal between the Fast Ethernet and the Myrinet runs, while the communication overheads explains the execution time difference.




(a) Wallclock time of subroutine call `fftw_f77_mpi` for 2 problem sizes and Myrinet intra-node communication (`mpirun.ch_gm=mpirun.ch_gm`). 3-dim. diagram, `no_procs=WildCardX`, `length=256=WildCardY`.




(b) Intra-node communication comparison for `length=`$2^7$ problem size. 3-dim. diagram, `no_procs=WildCardX`, `mpirun.ch_gm=WildCardY`.




(c) Number of processes per SMP node comparison for `length=`$2^8$ problem size. 3-dim. diagram, `ppn=4=WildCardX`, `nodes=1=WildCardY`.

**Figure 7. ADV Diagrams for FFTW.**

## 4.3 FFTW

The Fastest Fourier Transform in the West (FFTW) [10] 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 3-dimensional FFT, in order to be able to perform automatic instrumentation with SCALEA.

In general, the performance of FFTW is dependent on enough work per processor and on fast network. We therefore analysed the two biggest problem sizes that we could run on our SMP cluster, namely for 3-dim. square matrixes of size $2^7$ and $2^8$ (see Ex. 4.9). We run each problem size on 1, 2, 4, 8, and 16 MPI processes (powers of 2) using 1, 2, and 4 SMP nodes (ZEN variable `nodes=1`, see Ex. 4.10). We also studied the impact on performance of the number of MPI processes per SMP node (ZEN variable `ppn=4`). The constraint directives ensures that the correct number of nodes and processors per node is allocated by PBS for each machine size (i.e. number of MPI processes). Myrinet has been chosen as communication network among SMP nodes. We examined two different communication options within one SMP node: Myrinet and shared memory. Shared memory communication within one SMP node is achieved by giving the `--gm-use-shmem` option to the `mpirun.ch_gm` command. All subroutine calls of the program (SCALEA `CR_S` mnemonic) have been measured (see Ex. 4.9).

**Example 4.9 (Source File – `benchF.f90`)**

```
!ZEN$ CR CR_S PMETRIC WTIME
...
!ZEN$ SUBSTITUTE length\=256 = { length=2**{7,8} }
INTEGER, PARAMETER :: length=256
```

**Example 4.10 (PBS script – `run.pbs`)**

```
#!ZEN$ SUBSTITUTE nodes\=1 = { nodes={1,2,4} }
#!ZEN$ SUBSTITUTE ppn\=4 = { nodes={1,2,4} }
#PBS -l walltime=0:29:59,nodes=1:fourproc:ppn=4
cd $PBS_O_WORKDIR
#!ZEN$ SUBSTITUTE mpirun.ch_gm = { mpirun.ch_gm,
                          mpirun.ch_gm\ --gm-use-shmem }
#!ZEN$ SUBSTITUTE no_procs = {1,2,4,8,16}
mpirun -np no_procs benchF
#!ZEN$ CONSTRAINT INDEX
  2^(ppn\=4 -1)*(2^(nodes\=1 -1)-1)<2^(no_procs -1)&&
  2^(no_procs -1) <= 2^(ppn\=4 -1) * 2^(nodes\=1 -1)
```

7 ZEN directives have been inserted into 2 files to generate a total of 48 experiments. Fig. 7(a) shows the wallclock time of the fftw_f77_mpi subroutine call for the two problem sizes analysed. FFTW does not scale which is mostly due to the relative reduced problem sizes analysed because of limited amount of physical memory available per SMP node. For a matrix size of $2^8$ the execution time slightly decreases for 2 MPI processes. Myrinet and shared memory produce similar performance up to 4 MPI processes. Above 4 processes, Myrinet tends to perform better due to higher bandwidth and better cash behaviour (see Fig. 7(b)). The non-scalability of the application is also shown in Fig. 7(c). A lower number of processes per SMP node produces a better performance.

## 4.4 Backward Pricing

The backward pricing kernel [7] is a parallel implementation of the backward induction algorithm to compute 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.

### 4.4.1 Performance Study

Backward pricing has been encoded as an HPF+ [2] application. HPF+ directives are used to distribute 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) [4]. 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 PBS. We annotated the PBS script to vary the machine size from 1 to 10 SMP nodes (see Ex. 4.11). The ZEN variable nodes=1 is set with the number of SMP nodes, which must be equal with the number of MPI processes, represented by the ZEN variable no_procs. This equality is achieved through a ZEN constraint directive. 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. 4.12). The overall execution time, as well as the MPI communication and the HPF+ inspector/executor overheads have been measured.

**Example 4.11 (PBS script – run.pbs)**

```
#!ZEN$ SUBSTITUTE nodes\=1 = { nodes={1:10} }
#PBS -l walltime=0:29:59,nodes=1:fourproc:ppn=4
#PBS -v MPI_MAX_CLUSTER_SIZE=1
#PBS -N benchF
cd $PBS_O_WORKDIR
#!ZEN$ SUBSTITUTE no_procs = {1:10}
mpirun -np no_procs bw_halo_sis
#!ZEN$ CONSTRAINT INDEX nodes\=1 == no_procs
```

**Example 4.12 (Input file – bench.in)**

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

Figure 8(a) shows a good scalability of this code. Backward pricing is a computational intensive application, which highly benefits from the intra-node shared memory OpenMP parallelisation. The overall wallclock time of the application significantly improves by encreasing number of OpenMP threads per SMP node. Fig. 8(b) shows a very high ratio between the application user time and the HPF and MPI application overheads, 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.

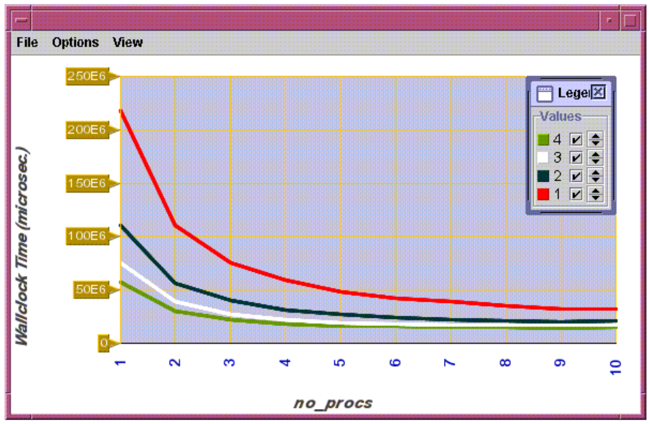
### 4.4.2 Parameter Study

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 was encoded such that these input parameters are read from different input data files. ZEN assignment directives are inserted in the backward pricing code immediately after the corresponding read statements (see Ex. 4.13). Thus, the original read statement is made redundant. A constraint directive is used to guarantee 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.

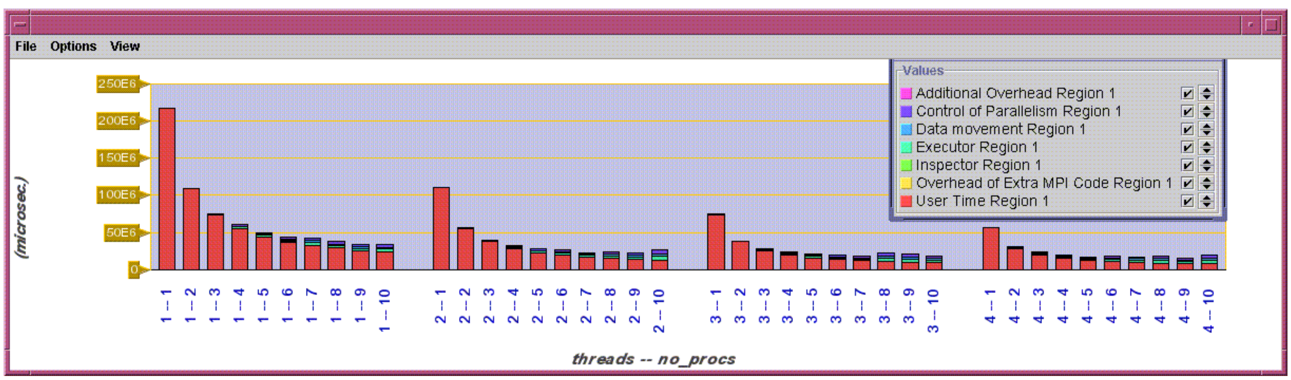
**Example 4.13 (Source file – bw_halo.f90)**

```
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 }
```

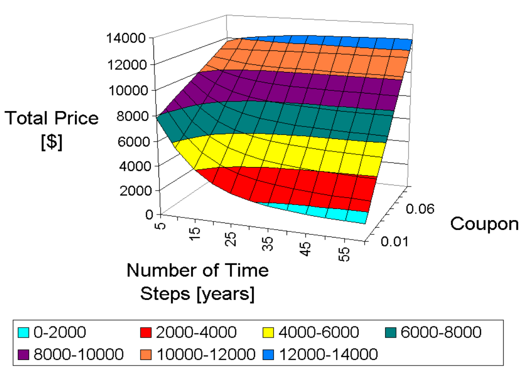
A total number of 1200 experiments were generated. Two sample diagrams are depicted in Figure 8. The 3-dim. surface in Fig. 8(c) shows the evolution of the total price (output parameter of the backward pricing code) as a function of the number of time steps and the coupon, with the following meaning: (a) the price decreases with the maturity (number of time steps × 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,



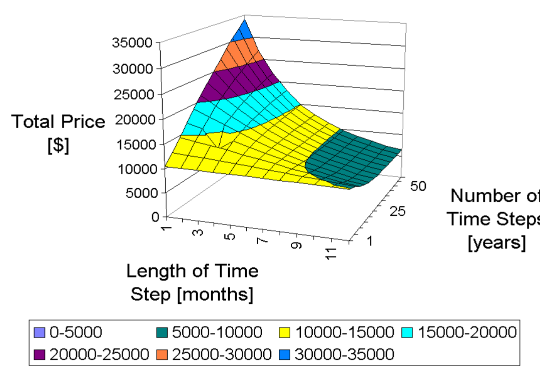

(a) Wallclock time for different number of OpenMP threads per SMP node. 3-dim. diagram, `no_procs=WildCardX`, `threads=WildCardY`.




(b) MPI and HPF Overhead Metrics. 4-dim. diagram, `Metric=WildCardX`, `no_procs=WildCardY`, `threads=WildCardZ`.




(c) Length of Time Step (`delta_t`) = 1.0. 3-dim. diagram, `nr_steps=WildCardX`, `coupon=WildCardY`.




(d) Coupon = 0.05. 3-dim. diagram, `delta_t=WildCardX`, `nr_steps=WildCardY`.

**Figure 8. ADV Diagrams for Backward Pricing.**

the higher the future payments are; (c) for very large maturities, the price linearly depends only on the coupon.

Figure 8(d) shows the price evolution by varying the number of time steps and the length of a time step, with the following explanation: (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.

## 5 Conclusions

Developing ZENTURIO has been driven by the need to support scientists and engineers in their effort to conduct parameter studies, performance analysis and tuning, and software testing for a wide variety of parallel and distributed systems, including cluster and Grid architectures. The main contributions of ZENTURIO are centered around the following aspects: (1) It employs a directive based language (ZEN) to specify arbitrary experiments by substituting values of any problem and machine parameter, to request for performance metrics for arbitrary code regions, and to control the number of experiments. ZEN directives are not changing the semantics of any file, as they are interpreted as comments and therefore ignored by systems that are not aware of them. (2) ZENTURIO does not assume pre-compiled and pre-installed application codes on the target machine. Compilation and file transfer is supported. (3) A graphical User Portal enables the user to control and monitor experiments. A flexible technique to automatically visualise performance and output data across large sets of experiments has been developed. (4) A flexible design of a distributed service architecture based on Java/Jini technology and an interface to a variety of job schedulers allows us to easily add new target architectures.

We have demonstrated the usefulness of ZENTURIO for performance and parameter studies of a variety of applications. We believe that ZENTURIO is a generic tool that provides a sophisticated infrastructure and user portal to automate and control experiments for sequential, parallel, and distributed applications. Future work involves the development of a Grid-oriented architecture based on the Web services technology [22] and the Open Grid Service Architecture (OGSA) [9] specification.

## 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] Vikram S. Adve, John Mellor-Crummey, Mark Anderson, Ken Kennedy, Jhy-Chun Wang, and Daniel A. Reed. Integrating Compilation and Performance Analysis for Data Parallel Programs. In *Proc. of the Workshop on Debugging and Performance Tuning for Parallel Computing Systems, IEEE Computer Society Press*, January 1996.

[3] Mark Baker and Rajkumar Buyya. Cluster computing at a glance. In Rajkumar Buyya, editor, *High Performance Cluster Computing*, volume 1, Architectures and Systems, pages 3–47. Prentice Hall PTR, Upper Saddle River, NJ, 1999. Chap. 1.

[4] S. Benkner. VFC: The Vienna Fortran Compiler. *Scientific Programming, IOS Press, The Netherlands*, 7(1):67–81, 1999.

[5] 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.

[6] Karl Czajkowski, Ian Foster, Nick Karonis, Stuart Martin, Warren Smith, and Steven Tuecke. A Resource Management Architecture for Metacomputing Systems. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, pages 62–82. Springer Verlag, 1998. Lect. Notes Comput. Sci. vol. 1459.

[7] 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.

[8] W. K. Edwards. Core Jini. *IEEE Micro*, 19(5):10–10, September/October 1999.

[9] 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`.

[10] 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.

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

[12] I. Foster and C. Kesselman and S. Tuecke. The Anatomy of the Grid. *The International Journal of High Performance Computing Applications*, 15(3):200–222, Fall 2001.

[13] 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.

[14] 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.

[15] B. Mohr. Design of Automatic Performance Analysis Systems, APART Workpackage 3: Implementation Issues. APART technical report, Forschungszentrum Jülich, Zentralinstitut für Angewandte Mathematik (ZMG), D-52425 Jülich, May 2000. www.kfa-juelich.de/apart.

[16] 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.

[17] 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.

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

[19] 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.

[20] 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. To appear.

[21] Veridian Systems. PBS: The Portable Batch System. `http://www.openpbs.org`.

[22] W3C. Web Services Activity. `http://www.w3.org/2002/ws/`.

[23] 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.