



CONTRACT NUMBER 031513

eDEISA
**EXTENDED DISTRIBUTED EUROPEAN
INFRASTRUCTURE FOR
SUPERCOMPUTING APPLICATIONS**

European Community Sixth Framework Programme
RESEARCH INFRASTRUCTURES
Integrated Infrastructure Initiative

Release of Version 1 of DEISA Benchmarks

Deliverable ID: eDEISA-D-eSA4-B2
Due date: May 31 2007
Actual delivery date: May 25 2007
Lead contractor for this deliverable: EPCC, UK

Project start date : June 1st, 2006
Duration: 2 years

Project co-funded by the European Commission within the Sixth Framework Programme (2002-2006)		
Dissemination Level		
PU	Public	X
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Table of Contents

Table of Contents	1
1.1 Executive Summary.....	2
1.2 References and Applicable Documents	2
1.3 Document Amendment Procedure	2
1.4 List of Acronyms and Abbreviations	2
2. Selection Process for the Initial Benchmark Suite	3
2.1 Status after PM6.....	3
2.2 Evaluation of Code Characteristics Sheets	3
2.3 Presentations at eSA4 Applications Festival	4
2.4 Proposal to DEISA Executive Committee.....	4
2.5 Initial Investigations of Codes.....	5
3. Contents of the Initial Benchmark Suite.....	5
3.1 Summary Table	5
3.2 Low-level Benchmarks	6
4. Future work.....	8
5. Conclusions	9
6. Appendix A: Summaries of Benchmarking Codes.....	10
6.1 CPMD	10
6.2 Quantum-ESPRESSO.....	12
6.3 DL_POLY	14
6.4 NAMD	17
6.5 IQCS.....	20
6.6 ECHAM5.....	21
6.7 NEMO.....	23
6.8 Gadget-2.....	26
6.9 RAMSES	28
6.10 SUCCESs.....	30
6.11 Fenfloss	32
6.12 GENE	33
6.13 PEPC.....	35
6.14 BQCD	37
6.15 SU3_AHIGGS.....	39

1. Introduction

1.1 *Executive Summary*

The objective of the eDEISA benchmarking project (task 3 in the eSA4 Applications Enabling Service Activity) is to design and support a benchmark suite that can be used easily to evaluate the performance of high-end supercomputers. It will include a number of real applications codes and associated datasets which are representative of the research undertaken by Europe's leading computational scientists. It will also include a set of low-level synthetic benchmarks to evaluate the performance of key components of the machine architecture.

This document is a report on the activities of the benchmarking group undertaken during months seven to twelve of the two-year eDEISA project.

Section 2 describes the process by which the contents of the initial benchmark suite have been decided.

Section 3 summarises the contents of the initial benchmark suite and gives base reference values for actual datasets measured on selected DEISA platforms.

Section 4 outlines the future workplan of the benchmarking team for project months thirteen to twenty four.

Section 5 contains some brief conclusions.

Appendix A contains summary descriptions of all the components of the benchmarking suite including brief instructions on how they are run and reference results for validation.

1.2 *References and Applicable Documents*

- [1] eDEISA Annex I – “Description of Work”, 27 March 2006.
- [2] eDEISA deliverable eDEISA-D-eSA4-B1: “Identification of an Initial Set of Application Codes and Low Level Benchmarks”, 18 December 2006

1.3 *Document Amendment Procedure*

Reviewed internally by EPCC and eSA4-T3 team, and by external eDEISA reviewer.

1.4 *List of Acronyms and Abbreviations*

CFD	Computational Fluid Dynamics
DECI	DEISA Extreme Computing Initiative
DEISA	Distributed European Infrastructure for Supercomputing Applications
HPCx	A 1.5GHz POWER5 p575 system operated by EPCC, UK
JUMP	Juelich MUlti-Processor, a.1.7GHz POWER4 p690+ IBM system
QCD	Quantum ChromoDynamics
ZAHIR	A 1.7GHz POWER4 p655 system at IDRIS, France

2. Selection Process for the Initial Benchmark Suite

As described in the PM6 deliverable “Identification of an Initial Set of Application Codes and Low Level Benchmarks”, the aim of the benchmarking activity is to provide a single, coherent collection of software that can easily be used to evaluate the performance of high-end supercomputers.

In order to be representative and accepted by the HPC community, the suite must cover a wide range of scientific applications. However, in order for a user to be able to port and run the suite in a straightforward manner within a reasonable amount of time it should probably contain less than ten codes. It is therefore not possible to cover all applications of HPC in Europe, so some selection criteria must be used.

2.1 Status after PM6

The outcome of the PM6 deliverable was a superset list of 42 applications codes proposed by all the DEISA partners. The planned process for selecting the initial benchmark release from this superset was:

1. Evaluation of benchmark superset based on high-level criteria
2. Selection of initial benchmark suite (v1) by DEISA Executive Committee
3. Initial investigation of benchmark v1 portability, performance and datasets
4. Initial benchmark release (v1)

The plan was that the benchmarking team would evaluate all candidates based on the high-level criteria contained in the characteristics sheets completed for every code. This would lead to a ranked list of codes for each of the scientific categories to be presented to the DEISA Executive Committee, which would decide on the contents of the initial benchmark (expected to be around 15 codes with at least two in each applications area) based on this information.

Although the work of the benchmarking team did lead to a list of 15 codes ratified by the Executive Committee, the actual process was not exactly the one proposed above. The actual process is described below.

2.2 Evaluation of Code Characteristics Sheets

The initial work of the benchmarking team had resulted in 53 proposals (42 separate codes) from across all the eDEISA partners. All partners were then required to complete a standard characteristics sheet (as presented in the PM6 deliverable) if the proposed code was to be considered for inclusion in the benchmark suite. A videoconference was held on 23 January 2007 to consider all these sheets with the aim of drawing up a ranked list for each scientific area.

The meeting was partially successful in that it reduced the list of codes from 42 to 28. In some of the scientific areas there was a clear leading code, eg GADGET-2 (Astrophysics), NAMM (Life Sciences) and CPMD (Materials). In other areas, only two candidates remained so both could be taken forward for further investigation (eg QCD and Plasma Physics). However, there were still 21 codes remaining which required to be reduced to a list of about 8 codes. The number and diversity of these remaining codes meant that it was not possible to come to a coherent decision at the meeting, so a further process had to be undertaken.

2.3 Presentations at eSA4 Applications Festival

A two-day meeting of all the eDEISA eSA4 applications staff had been arranged for 14-15 February 2007 at RZG in Munich. It was also going to be attended by most of the Executive Committee members. It was decided that this was a good venue at which to present the status of the benchmarking activity and to make progress on the contents of the initial benchmark. The presence of so many applications experts from all the eDEISA partners made this an ideal opportunity to come to an informed consensus on the remaining 21 codes.

A number of presentations were made by members of the benchmarking team:

- A single spreadsheet summarising the characteristics information for all 28 remaining codes was developed and circulated in advance.
- An overview talk was given by the benchmarking team leader covering the progress made so far. Short descriptions were presented for each of the seven codes already chosen for the initial benchmark.
- A summary talk was given for each of the following four applications areas: CFD and Combustion; Astrophysics; Life Sciences; Materials. A member of the benchmarking team was put in charge of each area, and it was their responsibility to develop the talk with input from all the appropriate experts.
- A short talk was given for each of the five remaining Earth Sciences and Climate Research codes. These tend to be very large applications which represent a substantial research and development investment from member states. As a result, it was felt that it was appropriate to treat them somewhat differently from the other applications and to allow partners to make their own individual cases for each code.

The presentation session was extremely successful and provided an excellent forum for discussing the benchmark suite. At the end of the meeting, the list of 28 remaining codes had been reduced to 19 based on the information presented by the benchmarking team.

2.4 Proposal to DEISA Executive Committee

The list of 19 codes was a number close enough to the ultimate target of 15 that it was thought appropriate to begin more detailed studies of all of the codes. In order to obtain comparable performance information on every code it was decided to attempt scaling runs of all codes on IBM POWER platforms on 64, 128, 256 and 512 processors. A range of IBM systems is available on the DEISA supercluster so this seemed the most appropriate initial target architecture. These IBM machines differ somewhat in their detailed make up (eg exact generation of POWER chip and clock speeds) so, to allow for absolute performance comparisons, it was also decided to run all codes on 64 processors of the Juelich JUMP system (an IBM 1.7GHz p690 machine).

Following these investigations, a brief proposal was made to the DEISA Executive Committee for the contents of the initial benchmark suite. This contained concrete recommendations for 13 codes to be drawn from six of the seven scientific applications areas. The choice of the codes from the five remaining candidates in Earth Sciences and Climate Research was left to the discretion of the Committee according to both scientific and strategic issues; the document only contained initial performance results for these codes.

At a meeting of the Executive Committee on 26 March 2007 the composition of the initial benchmark suite (15 applications codes) was decided.

2.5 Initial Investigations of Codes

After the executive committee meeting, work continued on the initial benchmark suite. In addition to finalising the baseline performance on POWER architectures, portability was now addressed. It was decided also to attempt to run every code on a different system, preferably the BSC MareNostrum machine, to investigate code portability and correctness.

3. Contents of the Initial Benchmark Suite

For the sake of completeness, full summaries of all the applications codes are included in Appendix A. These contain information about the following areas:

- **Description:** brief overview of the scientific relevance;
- **Compilation and Execution:** summary of how to compile and run;
- **Performance Metric:** how performance is measured and reported;
- **Validation:** which outputs can be used to check correctness;
- **Performance Results:** scaling results;
- **Performance Characteristics:** what architectural features are stressed;
- **Portability:** proof that code is correct on more than one platform.

In the summaries, performance is quoted as a relative efficiency when the number of processors is doubled. For example, an efficiency of 80% on 256 processors (referred to as E256) represents the parallel efficiency between 128 and 256 processors; in terms of the measured time, this is equal to $T_{128} / (2 * T_{256})$. This measure was used during the evaluation phase as it is simple to compute and easy to compare across different platforms. It is not possible to define a speedup with respect to a single processor as the datasets used here are too large.

3.1 Summary Table

Wherever possible, performance was measured on 64, 128, 256 and 512 processors of the Juelich JUMP system. With this consistent set of measurements now available it is possible, for almost all codes, to quantify performance in terms of a single speedup between 64 and 512 processors (i.e. T_{64} / T_{512}). The contents of the initial benchmark suite are summarised in Table 1. Where multiple datasets or runtime options are available, only the fastest result is presented (see Appendix A for full details). Total elapsed time (including overheads such as IO) on 64 CPUs of JUMP is also reported for reference purposes: the real time taken is an important factor when running any benchmark in practice.

It is worth emphasising that, in many cases, DEISA partners have ongoing strong relationships with the developers of the applications. As well as being important in the future when new datasets may have to be developed, these relationships have already proved extremely valuable during the initial selection process. We were generally able to solve any problems encountered in porting and running the codes very quickly as we had immediate access to expert knowledge.

There are a number of notes regarding Table 1:

- For Quantum-ESPRESSO, the current dataset is too small to exhibit scalability beyond 256 processors so the speedup presented is calculated on 256 processors; a larger dataset is planned for the future. The times here are for the first iteration and not for full convergence.

- IQCS performs computations relevant to Quantum Computing. We measure a basic IQCS kernel which is a relatively small piece of code; this may be considered as a low-level benchmark in the future if appropriate.
- ECHAM5 speedup is taken from runs on 16 and 128 processors of an NEC SX-8 vector system; see Appendix A for explanation.
- There are three codes listed in astrophysics because SUCCEs actually demonstrates many features of a CFD combustion code. We had no combustion codes under CFD, so SUCCEs was retained even though it means there are more than two codes in the astrophysics category.
- There is only one CFD code as the combustion part is covered by SUCCEs. The other CFD code, PDNS3D, was removed as its performance was predictable based solely on the low-level STREAMS benchmark.
- Fenfloss results are for weak scaling (all other codes are strong scaling) so the relevant speedup is 8 times the ratio of runtimes.
- GENE cannot be run on 64 CPUs due to memory constraints; the speedup assumes linear scaling from 64 to 128 CPUs.
- The runtime of BQCD is rather long. However, this can be reduced by a simple adaptation of the dataset which lowers the number of timesteps.

Applications Area	Code Name	Elapsed Time, mins on 64 CPUs	Speedup, 64 → 512 CPUs
Materials Science	CPMD	17	7.08
	Quantum-ESPRESSO	14	2.00
Life Sciences + Informatics	DL_POLY	1	4.85
	NAMD	1	5.30
	IQCS	7	6.16
Earth Sciences	ECHAM5	N/A	5.98
	NEMO	19	4.20
Astrophysics	Gadget-2	15	7.10
	RAMSES	27	7.78
	SUCCEs	45	6.75
CFD + Combustion	Fenfloss	1	3.95
Plasma Physics	GENE	1	8.20
	PEPC	44	6.22
QCD	BQCD	173	8.15
	SU3_AHIGGS	30	4.72

Table 1: Summary of Selected Applications Codes

3.2 Low-level Benchmarks

The work over the last 6 months has very much focused on the applications codes as this was by far the most time-critical work. The content of the low-level benchmark remains as proposed in the PM6 deliverable:

- The standard HPC kernel benchmarks covering CPU, memory and communications performance;
- The IOzone and B_eff_IO benchmarks to measure disk IO performance.

With the applications content of the initial benchmark now finalised we will start to investigate the performance of these low-level benchmarks in the context of the other benchmarking activities. There are a number of suggestions for additions to this list

contained in the PM6 deliverable which could be considered if none of the members of the applications suite stresses a particular feature of the parallel architectures.

4. Future work

The aims for the second and final year of the benchmarking activity are:

1. Reduce the current list of codes to as small a number as possible while still retaining a benchmark that tests all the important architectural features of parallel supercomputers;
2. Package up the benchmark into a coherent release that is straightforward to compile, run, validate and interpret.

As the second stage requires a substantial amount of work for every code, we do not plan to start packaging the benchmark suite until the final list of codes is decided. However, all codes are being collected and stored in a central location for internal distribution.

To reduce the current list of codes we will use the following procedure, building on the work done in the first year:

1. Develop larger datasets for those benchmarks where the current ones do not permit scalability to large numbers of processors (this will be done by experts in each application, often in collaboration with the developers);
2. Run the benchmarks on a wider range of DEISA platforms and collect performance results;
3. Analyse the results to determine whether different codes are displaying similar performance characteristics across different architectures, in which case one of them can be removed;
4. Propose a final benchmark suite with a target number of around eight codes.

We plan to finalise these stages by the end of PM18 which will result in a proposal to the DEISA Executive Committee.

After this, we will develop a coherent benchmark package with the aim of automating as many of the compilation, execution and validation stages as is feasible in the time available. We will also include any information regarding licensing issues. As is standard practice in this area we will require those running the benchmark to comply individually with the requirements of each code before running the benchmark as we are not in a position to resolve licensing issues for third-party applications.

As proposed in the technical annexe, the package will be accompanied by a set of rules describing the allowable code changes for each application. These will vary in detail from code to code, but in general we would expect that, in addition to choosing compiler flags, vendor-optimised libraries could be used in place of existing routines. Small changes to source would be allowed in specific circumstances (e.g. to facilitate the inclusion of an optimised library or the addition of compiler directives), but we would not permit complete rewriting of substantial sections of code.

5. Conclusions

The work of the benchmarking team in the first year has been successful in providing an initial set of benchmarks. These all require further investigation to produce a final release of a manageable size, but there is a clear plan to take them forward.

The initial release perhaps contains more codes than would have been expected at the start of the benchmarking activity. This was mainly due to the very large number of codes proposed during the initial survey, which meant that reducing the set required more work than anticipated. The most important decision taken during this initial selection phase was for the benchmarking team primarily to aim to provide information on the benchmark codes, with the actual decisions on which codes to retain being taken in consultation with all the DEISA partners. The presentation sessions devoted to benchmarking at the eSA4 Applications Festival in February was an extremely important component of the selection procedure. The risk of the number of codes remaining leading to too much work in the second year has been mitigated by some additional staff effort being transferred from other eSA4 activities.

6. Appendix A: Summaries of Benchmarking Codes

6.1 CPMD

Description

The CPMD code is a parallelized plane wave / pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics. CPMD is copyrighted jointly by IBM Corp and by Max Planck Institute, Stuttgart, and is distributed free of charge to non-profit organizations.

The basic algorithm involves the solution of the Hamiltonian problem through an annealing of electronic degrees of freedom, then the molecular dynamics is performed using the Car-Parrinello algorithm with which electronic and ionic degrees of freedom are evolved with equations of motion derived from the same Car-Parrinello Lagrangian. Of fundamental importance is the performance of the parallel 3D FFT used to compute forces on electronic degrees of freedom at each time step.

The datasets used for the benchmark are two water supercells with 128 and 256 water molecules at room density. The electron-ion interaction is represented using norm-conserving pseudopotentials in the form suggested by Martin and Troullier.

Compilation and Execution

To install and compile the code, the CPMD package includes a configuration script that produces the Makefile for the target architecture. In the case of a new architecture one has to choose the closest architecture among the ones included in the script, and then it is necessary to manually adapt the Makefile.

To run the code one simply has to specify the textual input file and provide the pseudopotentials file for the simulated atoms. In particular, for the benchmark the following command was issued: `./cpmd.x input.in $PP_LIBRARY > output.`

Performance Metric

The performance is measured by taking the average wall clock time per step, over a run of 10 steps of electronic annealing. The figures for the wall clock are taken from the code standard output, in lines that looks like:

NFI	GEMAX	CNORM	ETOT	DETOT	TCPU
1	6.791E-02	2.513E-03	-2160.008881	0.000E+00	6.40

The wall clock execution is also reported at the end of the code standard output, in a line that looks like: `ELAPSED TIME : 0 HOURS 1 MINUTES 37.69 SECONDS.`

Validation

As a validation criterion the electronic total energy at the end of the ten steps has been chosen, and is also reported in the code standard output. For the 128 and 256 water molecule data sets the values are respectively:

(K+E1+L+N+X)	TOTAL ENERGY =	-2186.42957358 A.U.
(K+E1+L+N+X)	TOTAL ENERGY =	-4372.85381378 A.U.

In general these values are accurate to the 10th digit.

NOTE: playing with different data distributions (e.g. using TASK GROUPS or mixed OpenMP/MPI execution) can give values that may differ from these. In this case the result to complete convergence may be used.

Performance Results

All performance results were achieved on JUMP.

128 WATER MOLECULES				
<i>Average time per timestep 1 thread per task</i>	64	128	256	512
	14.37s	6.132s	6.163s	7.111s
<i>efficiency</i>		128	256	512
		1.1660	0.497	0.4333
<i>wall-clock time (mm:ss)*</i>	64	128	256	512
	3:28	1:37	1:33	1:47
<i>Average time per timestep 2 threads per task****</i>	64	128	256	512
	20.653s	13.455s	5.411s	5.558s
<i>efficiency</i>		128	256	512
		0.7674	1.2433	0.4867
<i>Average time per timestep 4 task groups**</i>	64	128	256	512
	15.399s	8.207s	5.86s	3.242s
<i>efficiency</i>		128	256	512
		0.9381	0.7002	0.9037
<i>wall-clock time (mm:ss)*</i>	64	128	256	512
	4:20	2:25	1:47	1:10

256 WATER MOLECULES***				
<i>Average time per timestep 1 thread per task</i>	64	128	256	512
	71.703s	43.638s	18.588s	19.935s
<i>efficiency</i>		128	256	512
		0.82156	1.17382	0.63319
<i>wall-clock time (mm:ss)*</i>	64	128	256	512
	17:52	10:58	5:34	5:29
<i>Average time per timestep 2 threads per task****</i>	64	128	256	512
	103.91s	57.3s	32.911s	14.678s
<i>efficiency</i>		128	256	512
		0.90671	0.8705	1.121
<i>wall-clock time (mm:ss)*</i>	64	128	256	512
	16:25	9:45	6:01	3:18

* wall-clock time for the complete run (as read in the code standard output).

** run performed using 1 thread per processor and Tasks Group data distribution.

*** NOTE no run has been done with task groups data distribution for 256 molecules.

**** each thread takes a CPU, the geometry used is 2x32 (64, 128, 256)

Performance Characteristics

These benchmark mainly stress memory and network bandwidth.

Portability

The dataset for the 128 water molecules has been run on Marenostrum using 64 processors, and the validation criterion match. Here it is reported in the line of the output that serves for the validation:

(K+E1+L+N+X) TOTAL ENERGY = -2186.42957359 A.U.

6.2 Quantum-ESPRESSO

Description

Quantum ESPRESSO is an integrated suite of computer codes for electronic structure calculations and materials modelling at nanoscale. It is based on density functional theory, plane waves, and pseudopotentials (both norm-conserving and ultrasoft). Quantum ESPRESSO stands for opEn Source Package for Research in Electronic Structure, Simulation, and Optimization. It is freely available to researchers around the world under the terms of the GNU General Public License.

In particular, for the benchmark, the PWscf (pw.x) component has been used. PWscf is a total energy calculation code that computes the ground state electronic properties of materials. The basic algorithm solves the kohn-sham equations to find the Hamiltonian and the electronic states of the system. The kohn-sham equations are solved iteratively, adjusting the electronic density at each iteration. Diagonalisation of the Hamiltonian, which is needed at each iteration, is performed using the Davidson iterative algorithm.

The dataset used in the benchmark is related to a computation of the ground state energy of a gold surface with 96 atoms in the simulation supercell. Gold core electrons are represented with Vanderbilt ultrasoft pseudopotentials.

Compilation and Execution

The code for the installation and compilation takes advantage of the gnu autoconf configuration utility; on most architectures to compile the code it sufficient to do:

```
./configure
make pw
```

To run the code pw.x one needs to specify simply the textual input file, and to provide the pseudopotential files for each atomic species. In particular for the benchmark run the executable command was:

```
./pw.x -npool 2 -input ausurf.in > ausurf.out
```

Performance Metric

The performance is measured by taking the wall clock time for the first scf iteration and the wall clock time to complete convergence; these figures are reported in the code standard output in lines that look like:

```
total cpu time spent up to now is      8047.59 secs
```

(the time reported is wall clock even if in the output it is written as "cpu time")

The wall clock execution is also reported at the end of the code standard output, in a line that looks like:

```
PWSCF      :      2h 4m CPU time,          2h 7m wall time.
```

Validation

As a validation criterion the converged self consistent total energy is chosen, which is reported in the code standard output too. For the benchmark dataset this values is (as it appears in the standard output):

```
!      total energy          =-11427.08993380 Ry
      estimated scf accuracy <      0.00000043 Ryhere.
```

Performance Results

96 GOLD ATOM SURFACE				
	64	128	256	512
<i>time for first iteration</i>	576.87s	378.27s	287.12s	915.12s
<i>Efficiency</i>		128	256	512
		0.763	0.659	0.157
<i>time for convergence</i>	64	128	256	512
	8047.59s	6021.18s	4951.21s	5960.96s
<i>Efficiency</i>		128	256	512
		0.668	0.608	0.415
<i>wall-clock time (hh:mm)*</i>	64	128	256	512
	2:20	1:43	1:26	1:40

* wall-clock time: timing of run to complete convergence, including I/O and set up.

It is possible to stop the code at the first iteration - this can be a good starting point. However, running the code to complete convergence is required for a solid validation. Since the ratio between communications and computations change with the iterations, the ratio between the time to the first (heavier) iteration and the time to complete convergence gives more insight into the network performance.

Performance Characteristics

The benchmark stress mainly cache and main memory bandwidth, scalability is strongly influenced by network latency and less by network bandwidth

Portability

The benchmark has been run on MareNostrum using 64 processors, and the validation criterion matches. Here is reported the lines of the output that serve for the validation:

```
!   total energy           =-11427.08997399 Ry
   estimated scf accuracy  <    0.00000050 Ry
```

6.3 DL_POLY

Description

DL_POLY is a widely used, general purpose, molecular dynamics simulation package (over 2000 licences worldwide), developed at Daresbury Laboratories. The 55,000-line code is written in Fortran90 and parallelised with MPI. The version used here, DL_POLY_3, is parallelised using standard domain-decomposition techniques: some previous versions used replicated data and hence did not scale as well.

The performance and portability test have been limited to two data sets. The first, smaller data set is a 216,000 atom *Sodium Chloride* system, with unit electric charges on sodium and chlorine. The simulation runs at 500K with a NVT Berendsen ensemble; the SPME (Smooth Particle Mesh Ewald) method is used for the calculation of the Coloumbic interactions (DL_POLY_3 User Manual, version 3.06).

The second data set is a system of 16 *Gramicidin A* molecules in a water solution (256,768 water molecules), with a total of 792,960 atoms. The simulation runs at 300K using a NPT Berendsen ensemble with SPME and a SHAKE/RATTLE algorithm (DL_POLY_3 User Manual, version 3.06). The SHAKE/RATTLE algorithm is used to deal with constrained bonds, and is challenging to implement efficiently in parallel.

Compilation and Execution

The code provides three Makefiles in the subdirectory *build*: one for a parallel version of the code using MPI (*Makefile_MPI*), and two for serial compilation (*Makefile_SR1* and *Makefile_SR2*). The appropriate Makefile has to be copied into the subdirectory *source*.

The file *Makefile_MPI* contains the build information for several different target platforms. The executable name is set to *DLPOLY.Y* by default. The binary is placed into the *BINROOT* directory, *./execute* by default. To compile DL_POLY for a new target platform, the user needs to specify the **compiler** using following keywords:

- FC*: full path to the Fortran90 compiler
- FCFLAGS*: appropriate flags for FC (MPI include)
- LD*: path to the Fortran90 linker
- LDFLAGS*: appropriate flags for LD (MPI libraries)

Several example data sets are freely available. The code can be run by copying the example's input files (generally *CONTROL*, *CONFIG* and *FIELD*) into the *BINROOT* directory and executing *./DLPOLY.Y* – for job submissions on IBM systems, a sample LoadLeveler script is provided. A successful run of the code produces several files, including *OUTPUT*, which provides a summary of the job run.

Performance Metric

The code's performance is measured using both the total wall-clock time of an entire run, i.e. including start-up and closing-down times as well as IO, and the CPU time per timestep. The timings can all be found in the *OUTPUT* file, which is created in the *BINROOT* directory. The number of timesteps the code performs is defined in the file *CONTROL* – the time per timestep can be calculated by subtracting the start-up time from the wall-clock time after completion of all the steps, and dividing the difference by the total number of steps.

Validation

The validation criterion for each run is the total inertial energy of the system (the energy unit is defined by the *units* directive in *FIELDS* file). The total energy (*eng_tot*) can be found in the table of final averages listed in the *OUTPUT* file.

The expected total energy for the Sodium Chloride data set is $-7.7220\text{E}+09$; for the Gramicidin A data set it is and $1.8470\text{E}+05$.

Performance Results

The tables below list the scaling results for both data sets on the HPCx (1.5 GHz POWER5 p575) and JUMP systems. The wall-clock timings include start-up, close-down and IO. Efficiency is calculated based purely on the timesteps (i.e. the core computational loop).

SODIUM CHLORIDE WITH EWALD SUM (216,000 atoms)				
<i>time per timestep</i> (100 steps)	64	128	256	512
HPCx	1.215 s	0.665 s	0.381 s	0.256 s
JUMP	1.110 s	0.585 s	0.361 s	0.229 s
<i>efficiency (timesteps)</i>		128	256	512
HPCx		0.912	0.873	0.744
JUMP		0.948	0.809	0.788
<i>wall-clock time</i>	64	128	256	512
HPCx	48.6 s	31.6 s	24 s	34.5 s
JUMP	47 s	30 s	23 s	28 s

GRAMICIDIN A WITH WATER SOLVATING (792,960 atoms)				
<i>time per timestep</i> (30 steps)	64	128	256	512
HPCx	1.859 s	1.115 s	0.702 s	0.536 s
JUMP	1.974 s	1.181 s	0.684 s	0.514 s
<i>efficiency (timesteps)</i>		128	256	512
HPCx		0.834	0.794	0.654
JUMP		0.835	0.863	0.665
<i>wall-clock time</i>	64	128	256	512
HPCx	99 s	78 s	68.4 s	64.5 s
JUMP	107 s	81 s	67 s	66 s

* the datasets used for these performance runs are relatively small and were run over a small number of timesteps, therefore wall-clock times for the entire runs are short.

Performance Characteristics

The limiting factors for performance are MPI latency, both point-to-point and global communications, and FFT performance. Memory latency is also believed to be relevant (based on comparison of timings on POWER4 vs. POWER5). Which factors are most important is dependent to a large extent on the characteristics of the dataset.

Portability

In order to test the portability of this code, DL_POLY_3.06 was compiled (and run) on HPCx (power5), JUMP (power4+), EPCC's SUN Fire 15k server (running Solaris) as well as BSC's MareNostrum (PowerPC). The compilation proves unproblematic for

all these platforms. Reference values for the total energies were checked for JUMP, HPCx and MareNostrum and valid results are returned across all the platforms.

6.4 **NAMD**

Description

NAMD is a parallel molecular dynamics code designed for high-performance simulation of large biomolecular systems, and is based on Charm++ parallel objects. The package includes some interesting features such as message-driven execution and measurement-based load balancing. The 2.6 version involves around 65 k lines of C++ code (without taking into account the charm++ library).

NAMD uses a way of decomposition that easily generates the large amount of parallelism needed to occupy thousands of processors. Charm++ parallel objects and data-driven execution adaptively overlaps communication and computation, hiding communication latencies. In Charm++, objects may migrate between processors at runtime. This migration is controlled by the Charm++ load balancer.

The dataset used for the benchmark is the apoa1 bench; this benchmark involves 92,224 atoms, 12 Angstrom cutoff + Particle Mesh Ewald (PME) every four steps, with periodic boundary conditions. The version of NAMD used is 2.6. Input files and additional info about the dataset can be found at:

<http://www.ks.uiuc.edu/Research/namd/performance.html>

Compilation and Execution

Since Namd runs on top of Charm++, charm has to be built before building NAMD. Charm++ v 5.9 is provided in the NAMD source tarball. A build script that encapsulates the full process has to be used; the script runs with the following parameters:

```
build <target> <version> <options> [charmc-options ...]
```

Issuing `./build` without commands shows the different possible options for different architectures; as an example, we can build charm into MareNostrum (linux, ppc970, myrinet,gm) with the following command:

```
./build charm++ mpi-linux-ppc xlc64 gm --no-shared -O -DCMK_OPTIMIZE=1
```

After that, we can build NAMD. A script called `config` is provided in the root directory. This script is used to create the different makefiles for the configuration.

```
./config [debug] [tcl] [fftw] [<build_dir>/]<arch>[.comment] [suffix]
```

It is common to build NAMD with tcl and fftw features enabled, so we will need tcl and fftw external libraries. Paths to tcl and fftw are specified in the following files:

```
./arch/<arch>.fftw  
./arch/<arch>.tcl
```

To execute, one must issue the `namd2` binary with the configuration file as the first argument. It is important to point out that charm++ runs on top of MPI in most of the architectures, so, `namd2` is in fact an MPI application (i.e. `mpirun` or similar may be required to launch).

Performance Metric

There are two different performance metrics used, they both are computed by the namd binary and reported to the standard output file. Benchmark time provides averaged time per step after 20 and 25 simulation cycles. It is shown as follows:

>Info: Benchmark time: 512 CPUs 0.0108123 s/step 0.125143 days/ns 83821 kB memory

The wall clock time of the execution is reported as follows:

>WallClock: 17.906311 CPUTime: 17.906311 Memory: 88414 kB

Validation

Results can be validated by looking at the final energy values from the execution output: (step 500 for this benchmark). These results can be used as a reference. Minor changes may appear in the least significant digits.

```
>ENERGY:  500  20974.8936  19756.6577  5724.4523  179.8271  -
337741.4152  23251.1003  0.0000  0.0000  45359.0756  -
222495.4086  165.0039  -222135.7452  -222061.0904  165.0039  -
3197.5166  -2425.4140  921491.4634  -3197.5166  -2425.4140
```

Performance Results

These results were obtained at JUMP (AIX, IBM Power4 1.7 GHz, 32 procs, 128 Gb per node). Efficiency has been calculated taking into account the two different metrics used. The efficiency values are obtained as follows: $E_{128} = T_{64} / (2 * T_{128})$ etc. A precompiled version of NAMD (AIX_POWER_MPI, version 2.6) was used for the runs on JUMP.

APOA1 BENCHMARK (92,224 atoms)				
	64	128	256	512
<i>time per timestep</i>	0,0525s	0,0277s	0.0165s	0.0098s
<i>efficiency (timestep)</i>		128	256	512
		0,945	0,842	0,839
<i>wall-clock time</i>	64	128	256	512
	66,05s	31,76s	24,19s	23,18s
<i>efficiency (wall-clock time)</i>		128	256	512
		1,04	0,66	0,52

Performance Characteristics

From the communications point of view, point to point communication dominates, with a wide range of message sizes. Other important kernels are FFT calculations for the PME part.

Portability

Although the build process is not simple, NAMD relies only on standard C/C++ compilers, and can be built using different kinds of interconnects, so is highly

portable. It has been ported and validated on many architectures other than IBM AIX POWER4, for example MareNostrum (ppc970, myrinet, linux), without major issues.

6.5 IQCS

Description

This code is a part of a larger software package (IQCS) which has been developed to simulate ideal operations of a quantum computer on a classical computer in a very efficient way. The distributed part (iqcs) performs a successive Hadamard operation on a given number of qbits.

Compilation and Execution

Two parameters need to be compiled into the application.

1. Number of qbits to simulate (each additional qbit doubles memory demand);
2. \log_2 (number of MPI processes).

A shell script (createIQCSBench.sh) is provided for convenience to create separate directories and LoadLeveler job submission scripts for the runs. To run the benchmark:

1. Edit createIQCSBench.sh, and adjust parameters
2. Edit loadleveler.tmpl, and adjust parameters
3. ./createIQCSBench.sh
4. llsubmit iqcsbench_<qbits>_<cpus>_<procs>_<threads_per_tasks>/run.ll

Performance Metric

We use the time to perform one Hadamard operation for the given number of qbits.

Validation

The expectation values for s_x , s_y and s_z for each qbit are printed and should be zero for s_x , and 0.5 for s_y and s_z , for each qbit.

Performance Results

33 Qbits				
	64	128	256	512
<i>time for single Hadamard operation</i>				
MPI	240.05s	128.91s	70.79s	38.94s
MPI+OpenMP	359.67s	229.52s	123.85s	64.78s
<i>efficiency (timesteps)</i>		128	256	512
MPI		0.93	0.91	0.91
MPI+OpenMP		0.78	0.93	0.96
<i>wall-clock time</i>	64	128	256	512
MPI	454.68s	256.71s	135.19s	80.98s
MPI+OpenMP	668.41s	387.05s	213.63s	117.54s

Performance Characteristics

The benchmark stresses memory bandwidth and latency, and point-to-point communication.

Portability

IQCS has been ported to MareNostrum with no special adjustments except the compiler options. The number of qbits for MareNostrum had to be restricted to 32, as each process must not use more than 1.8 GB of memory.

6.6 ECHAM5

Description

ECHAM5 is the 5th generation of the ECHAM general circulation model. Depending on the configuration, the model resolves the atmosphere up to 10 hPa for tropospheric studies, or up to 0.01 hPa for middle atmosphere studies (often referred to as MAECHAM5).

ECHAM5 is relevant to many areas of science including studies done by the Intergovernmental Panel on Climate Change (IPCC), investigation of the paleo climate, data assimilation and simulations of extreme weather events.

Compilation and Execution

For compilation the code takes advantage of the autoconf gnu configuration utility. Nevertheless, to be able to compile the code on IBM Regatta systems some changes have to be applied to the generated makefile. These changes are documented in several README-files supplied with the code.

ECHAM5 also needs some 3rd party libraries: netCDF and LAPACK. To make them available to the code, additional changes (explicitly stating the libraries to link and/or location of the include-paths/library-paths) have to be made in the generated makefile or before calling the `./configure`-script in the configuration-files (to be found in the subdirectory `config`, starting with `mh-...`)

To run the code and setup the directory, where the experiment data resides, several batchscripts are available within the subdirectory `run`. These scripts start with `ham5run...`

Performance Metric

The performance metric is the number of forecast days for the model per day of real time. Performance measurements include IO.

Performance Results

The dataset being used for the benchmark has exhibited stability problems on the IBM JUMP system, and successful runs have not yet been performed on this machine. These issues are being addressed in collaboration with the code authors. Here we report results measured previously on the NEC SX-8 system at HLRS. The numbers of processors used (16 – 128) is less than would be used on the JUMP system; however, the vector processors on the SX-8 are significantly more powerful than the IBM's scalar processors. As a result, these tests correspond to running on resources of approximately the same power as 64 – 512 JUMP processors.

ECHAM5 with T255L60 data set (measured on NEC SX-8 vector machine)				
	16	32	64	128
<i>Forecast-days per day</i>	79.1	150.4	273.1	473.5
<i>efficiency</i>		0.95	0.91	0.87

Performance Characteristics

The limiting factor for performance at the moment is serial I/O and insufficient optimization of OpenMP on non-vector architecture for proper multi-core usage. The major sources of parallel overhead are I/O and all-to-all communication (transpositions).

Portability

The code currently runs on the following architectures:

- NEC SX-series
- IBM Power-4/5
- Cray XT-3/4
- commodity cluster with Infiniband interconnect (IA64, IA32, x64)

6.7 NEMO

Description

NEMO (Nucleus for European Modelling of the Ocean) is a state-of-the-art ocean modelling framework for oceanographic research and operational oceanography. It is a numerical platform for the ocean (dynamics and biochemistry) and the sea-ice. It has been developed for over 20 years in research laboratories. It is now widely used in many countries for research and operational projects covering the understanding of oceanic processes and climate modelling. It is a Fortran90 code, parallelized with MPI (standard domain decomposition). Four different data sets are available (which correspond to different values for the parameter *jp_cfg*, i.e. the number of horizontal points of the domain):

- *par_GYRE.1* (the smallest in size, *jp_cfg=1*, 1 degree ORCA);
- *par_GYRE.25* (*jp_cfg=25*, 1/2 degree ORCA);
- *par_GYRE.50* (*jp_cfg=50*, 1/4 degree ORCA);
- *par_GYRE.150* (the biggest one, *jp_cfg=150*, 1/12 degree ORCA).

Compilation and Execution

Compilation has to be done from the *~/Bin* directory. First, create the file *Make.inc* and adapt the different parameters (compiler name, libraries pathname, pre-processing keys and compilation flags) to the target architecture. Some example files (eg *Make.inc.IBM.-SP4*) can be found in this directory. By default, pre-processing keys are set for a mono processor run with: *P_P = key_zco key_gyre key_dynspgflt key_ldfslp key_zdfstke*. On vector architectures you can add *key_vectopt_memory* and *key_vectopt_loop* keys. For multi-processors runs (MPI), you must add the *key_mpp_mpi* key. To generate the executable, two parameter variables must be set:

- The number of processors (parameter NBPROC) has to be chosen from the following values: 64, 128, 256, 512 and 1024.
- The data set (parameter CONFIG) which has to be chosen from the following values: 1, 25, 50 and 150.

For example, if you choose the 1/2 degree GYRE configuration, you should use the *gmake* command above to build binaries for 64 and 128 processors:

```
gmake NBPROC=64 CONFIG=25
gmake clean
gmake NBPROC=128 CONFIG=25
```

The compilation creates a binary *nemo.\$NBPROC.\$CONFIG* in the *~/Bin* directory.

The NetCDF 3.6.1 library must be previously installed to complete the link step. The module *~/Src/NEMO/OPA_SRC/mpi_times.f90* which is used for timing must be compiled without integer promotion (for example, on IBM SP4, do not use the compiler flag *-qrealsize=8* but use the *-qautodbl=dbl4* flag).

The run must be submitted from the *~/Bin* directory. In this directory, there are some examples of script files adapted to IBM SP4 architecture. The executable needs only one input file (*~/Input/namelist*). The most important parameters in this file are:

- *nitend* (number of timesteps) allows you to adjust the elapsed time of the run.
- *nstock* is the output frequency of restart files. Generally, we set *nstock=nitend* (restart file is written one time at the end of run).

- *nwrite* is the output frequency of four GYRE* files. This parameter can be used to stress the IO subsystem of your architecture. With *nwrite=nitend*, files will be written only one time at the end of run. But with *nwrite=60* and *rdt=7200s*, the run will make outputs every 5 days of all variables.

Each MPI process writes its own five files (1 restart file and 4 GYRE* files). To minimize I/O, set "*nstock=nwrite=nitend*" (see *~/Input/namelist.minIO* file). On the contrary, if you want to stress the IO subsystem, you can use the file *~/Input/namelist.maxIO* with *nwrite=60*.

Performance Metric

The code prints at the end of the run, on the standard output, the elapsed time and CPU time per MPI process (in seconds). The timing includes everything (setup, IO, computation and communications).

Validation

Validation of a run is done by comparing the file *solver.stat* produced by your run to the same file (coming from a reference run on IBM SP4) in the directory *OutputRef*. For a given GYRE configuration and a fixed *nitend* value, if all runs are done on the same architecture, the files *solver.stat* should be strictly the same, bit to bit. If you change of architecture then some numerical differences could appear on the following outputs:

- *it=nitend* (number of timesteps) at the end of the file.
- *niter* (number of solver iterations per timestep) can be slightly different from one architecture to an other.
- *res* (solver residue for each timestep) can also be different, but variations should be less than 1.0E-08.

Performance Results

As well as reference results for the JUMP system, further investigation of datasets has been performed on ZAHIR at IDRIS. ZAHIR is also an IBM POWER machine, but is based on 1.7GHz POWER4 p655 nodes (as opposed to 1.7GHz POWER4 p690).

CONFIG=25, nitend=1500, noIO (3 GB)				
<i>efficiency (timesteps)</i>		128	256	512
JUMP		0.925	0.860	0.661
ZAHIR		0.885	0.780	
<i>wall-clock time</i>	64	128	256	512
JUMP	1139s	616s	358s	271s
ZAHIR	827s	468s	300s	

CONFIG=25, nitend=1500, IO (14 GB)				
<i>efficiency (timesteps)</i>		128	256	512
JUMP			0.840	
<i>wall-clock time</i>	64	128	256	512
JUMP		854s	507s	

CONFIG=50, nitend=1800, noIO (6.5 GB)				
		128	256	512
<i>efficiency (timesteps)</i>				
ZAHIR			0.918	
<i>wall-clock time</i>	64	128	256	512
ZAHIR		2063s	1123s	

CONFIG=50, nitend=1800, IO (58 GB)				
		128	256	512
<i>efficiency (timesteps)</i>				
ZAHIR			0.802	
<i>wall-clock time</i>	64	128	256	512
ZAHIR		2415s	1506s	

Performance Characteristics

Point-to-point communication and IO performance are the two features that are most stressed by the code. Considering the kernels, the solver and the parameterizations such as advection schemes, isopycnal diffusion and vertical mixing are the most demanding ones.

Portability

The code has run on many different types of architecture: vector architecture (NEC SX-[4-5-6-8], Earth Simulator, CRAY X1) and superscalar architecture (IBM SP[4-5], CRAY XT3, SGI Altix350, Cluster Linux, COMPAQ AlphaServer SC45, Fujitsu VPP5000). The only real difficulty is to find the appropriate compiler options. The sign function used in the code is not fully F95 compliant. The porting of the code on MareNostrum is in progress.

6.8 Gadget-2

Description

GADGET is a freely available code for cosmological N-body/SPH simulations on massively parallel computers with distributed memory. GADGET uses an explicit communication model that is implemented with the standardized MPI communication interface. The code can be run on essentially all supercomputer systems presently in use, including clusters of workstations or individual PCs. GADGET computes gravitational forces with a hierarchical tree algorithm (optionally in combination with a particle-mesh scheme for long-range gravitational forces) and represents fluids by means of smoothed particle hydrodynamics (SPH).

About 500 refereed publications have been written in the last five years using Gadget. The code is publicly available in a basic form, and is at the moment probably the most widely used simulation code in numerical cosmology worldwide.

Compilation and Execution

Gadget-2 relies on two 3rd party libraries: `FFTW v2` and `GSL`. A detailed `README` gives information on which specific configuration switches for `FFTW` are necessary. The application itself consists of two codes: a small application to generate the initial conditions and the Gadget-code itself. Both codes come with their own makefile. These makefiles have to be adjusted (e.g. include-paths/library-paths have to be set, to be able to link the needed additional libraries) to be able to compile the applications. Then the application to generate the initial conditions (called `N-GenIC`) must be run first. It takes the name of a parameter file as command line argument. More detailed instructions are found within the already mentioned `README`.

The simulation code itself (called `Gadget2`) is also run with the name of a parameter file as a command line argument – for more detailed information please read the supplied `README`.

Performance Metric

The benchmark executes 3 iterations. The performance results are based on the average wallclock time of the iterations 2 and 3.

The code writes data to standard output and to several files other files called: `balance.txt`, `cpu.txt`, `energy.txt`, `info.txt`, `parameters-usedvalues`, `timings.txt`. The location of these files is defined by an option within the parameter file for Gadget-2.

The file `cpu.txt` contains wallclock times of the first 3 iterations. The times are cumulative, so the times for the 3rd iteration include the times of the first and the second iteration. To exclude setup times, the wallclock time of the first iteration was subtracted from the time accumulated in the 3rd iteration time. This result was divided by 2 to get the average iteration time.

Validation

The standard output of the application contains information on the average particle impulse. Gadget-2 provides this data after every iteration. The lines with the validation criterion start with "type", but only the data for the last iteration is taken into account. The important values are stated after "`sqrt (<p^2>)`" in these lines and are expected to be 2.83647 and 2.83655

The important lines look like this:

```
type=0  dmean=500  asmth=244.141  minmass=0.138775  a=0.020308
sqrt (<p^2>)=2.83647  dlogmax=0.134366
type=1  dmean=500  asmth=244.141  minmass=0.902037  a=0.020308
sqrt (<p^2>)=2.83655  dlogmax=0.134363
```

Performance Results

All benchmark results were obtained on the FZJ JUMP system (IBM p690). The code does not use threads.

Gadget2				
<i>average wall-clock time (iteration 2 and 3)</i>	64	128	256	512
	211.975s	103.98s	53.265s	29.855s
<i>efficiency (average wall-clock time)</i>		128	256	512
		1.019	0.976	0.892
<i>wall-clock time (mm:ss)</i>	64	128	256	512
	14:24	08:26	04:55	03:50

Performance Characteristics

The most time-consuming step is a tree-walk through a large piece of memory that holds a hierarchical multipole expansion of the gravitational field, stored as an oct-tree. FFT performance is important, too, but usually subdominant in terms of the total wallclock time of the code.

The interconnect will be especially stressed by the code: a mixture of MPI_All-to-all (a transpose of a cube needed for a 3D FFT), and point-to-point communications that are executed in a hypercube pattern (using mostly MPI_Sendrecv), or asynchronously with MPI_Isend/MPI_Waitall. Scalability is however usually not limited by communication times per se, but rather by work-load imbalances which manifest themselves in wait times at synchronization points.

Portability

The code currently runs on the following architectures:

1. Linux Clusters of various flavors based in Intel Pentium/Xeon/Woodcrest, or AMD Athlon/Opteron, either with MPICH-1/2, or LAM-MPI.
2. IBM AIX machines based on Power 4/5, IBM Bluegene/L
3. IBM PowerPC Linux systems (e.g. MareNostrum)
4. Itanium-2 based systems like Altix 4700
5. Cray XT3

Vector systems have not yet been tested.

6.9 RAMSES

Description

RAMSES is a state-of-the-art Adaptive Mesh Refinement (AMR) code for astrophysical fluid dynamics. It is a tree-based AMR code which features an unsplit high-order Godunov scheme for hydrodynamics as well as state-of-the-art galaxy formation physics. Its parallelization strategy relies on an adaptive domain decomposition using a "space filling Peano-Hilbert curve". It is written in F90 and uses the MPI library.

The performance and portability tests have been limited to two data sets: the first, *sedov3d* (small data set), is the simulation of a 3D centred explosion (no AMR); the second, *horizon512* (big data set, with or without IO), is the simulation of formation of galaxies and large structures (with AMR).

Compilation and Execution

The *makefile* is in the *bin* directory and has to be adapted to the target architecture. The executable takes one argument: the name of the parameter file in the *input* directory (*sedov3d.nml*, *horizon512.nml*). For the big data set *horizon512*, four files are required to start a new computation (*ic_deltab*, *ic_velcx*, *ic_velcy* and *ic_velcz*). Initially, they are stored in the *input* directory. The *initfile* entry in the *INIT_PARAMS* namelist of the parameter file should contain the location of these files.

Output files are written to the directories *output_xxxxx* and *backup_xxxxx*. The *xxxxx* correspond to the process numbers (from 1) completed on the left by zeros to take five characters. These directories must exist before the execution. A small utility to create them is provided in the *utils* directory. If there are I/O specialized processes, they need their own directories *ionode_xxxxx* which contain the *output_xxxxx* and *backup_xxxxx* subdirectories for the computational processes they manage.

Performance Metric

The main performance metric is the total elapsed time in seconds printed by the application. This time include everything (setup, I/O, computation and communications). Timestep durations can also be used (also printed by the application)

Validation

There are two criteria that can be used to validate the results. The first is the physical time at the end of the run (given by *t=* on the lines beginning by "Fine step"). The second are the different final energies at the end of the run (given at the lines beginning by "Main step"). Depending on the data set, some of these values can be zero.

For example, for *sedov3d* (60 timesteps, 64CPUs on ZAHIR):

```
Main step= 60 mcons= 0.00E+00 econs=-6.18E-12 epot= 0.00E+00 ekin= 1.25E-01
Fine step= 60 t= 3.34776E-06 dt= 6.821E-08 a= 1.000E+00 mem=41.7%
```

horizon512 (30 timesteps, no I/O, 256CPUs on *zahir*):

```
Main step= 30 econs= 1.44E-03 epot=-3.75E-07 ekin= 2.51E-07 eint= 1.69E-12
Fine step= 30 t=-1.50773E+01 dt= 2.829E-01 a= 3.595E-02 mem=50.3% 90.7%
```

horizon512 (25 timesteps, high I/O (146GB), 256CPUs on *zahir*):

Main step= 25 econs= 1.95E-03 epot=-2.29E-07 ekin= 1.54E-07 eint= 1.64E-12
 Fine step= 25 t=-1.67151E+01 dt= 3.614E-01 a= 3.054E-02 mem=56.7% 88.9%

Performance Results

As well as reference results for the JUMP system, simulations have also been performed on ZAHIR at IDRIS. ZAHIR is also an IBM POWER machine, but is based on 1.7GHz POWER4 p655 nodes (as opposed to 1.7GHz POWER4 p690).

The performance difference between the two systems is substantial despite the equivalent clock speeds. This may be explained by the fact that the code stresses memory bandwidth, and the p655 has substantially better bandwidth per CPU than the p690.

Sedov3d, 512 ³ , no IO				
		128	256	512
<i>efficiency</i>				
JUMP		0.996	0.992	0.984
ZAHIR		0.972	0.997	
<i>wall-clock time</i>	64	128	256	512
JUMP	1634s	820s	413s	210s
ZAHIR	872s	449s	225s	

Performance Characteristics

The memory bandwidth and network communication are particularly stressed by the code. The most important kernel of the code in term of performance are: multigrid and conjugate gradient poisson solver; Godunov scheme's Riemann solver; the atomic physics module.

Portability

There are no known portability problems for this code. It already runs on many platforms: IBM Blade Center JS20/21 with Myrinet (MareNostrum, XLF Fortran), HP Alpha Custer with Quadrics (HP Fortran 90), HP Opteron Cluster with Infiniband (Portland Group Fortran and ifort), IBM/SP4 with Federation network (XLF Fortran), SGI ALTIX (Intel Fortran Compiler), Xeon Clusters with Intel Fortran Compiler and various interconnects (Gbits, Myrinet) and Mac OS X with gfortran.

6.10 SUCCEs

Description

The SUCCEs code employs state-of-the-art numerical techniques to model turbulent combustion in a large eddy simulation (LES) approach, including front-tracking techniques for following the flame propagation (level set method) and a localized turbulent subgrid-scale model. The code is the most efficient and best developed tool available to simulate Type Ia supernova explosions in 3 dimensions. Here, a white dwarf star explodes by being burned in a thermonuclear combustion.

With the help of the code it is possible to explore the physical nature of Type Ia supernova explosions. This is of outstanding relevance for astrophysics and cosmology, since on the basis of distance measurements to these objects, the new cosmological standard picture of an accelerated expanding universe has been established. This pointed to "Dark Energy" dominating the universe, and Type Ia supernovae are the best tool available to constrain the nature of this mysterious new energy form.

Compilation and Execution

Up to now the code does not use any configuration files; the configuration of a single run is compiled into the executable. There are different examples for compiler option files in the subdirectory "rules". To compile the code:

1. change to the "programs" subdirectory
2. set the environment variable PLATFORM to one of the files in the "rules" subdirectory (e.g. "setenv PLATFORM psiopt64" to compile with the options specified in rules/rules.psiopt64)
3. issue the following commands to compile the testcase bm_1.F:

```
> gmake clean  
> rm -f bm_1  
> gmake bm_1
```

To execute the code simply start the executable without any command line options:

```
> ./bm_1
```

Performance Metric

The performance metric was the summed up wallclock time of the iterations 2 to 10 (9 iterations). This excludes setup times.

Information on the current iteration is written to the standard output. The data for each iteration contains a line like this:

```
Wallclock time for this cycle: 179.33s
```

Validation

It is possible to validate SUCCEs with output which can be found in the protocol-file (the filename ends with a '.proto' and the file can be found in the output directory). For each timestep the following values can be found in this file:

- total energy without nuclear (to be found with: `grep 'total w' *.proto`)
- nuclear energy (to be found with: `grep Nuclear *.proto`)
- gravitational binding energy (to be found with: `grep Grav *.proto`)

The values depend a little bit on resolution and on the actual timestep so only the first 5 to 6 decimals have to be taken into account when validating SUCCEsSs on a new platform.

For example, on 64 CPUs the first three values of 'total energy without nuclear' are:

```
Total w/o nucl.: -5.202214535182E+50
Total w/o nucl.: -5.202213903600E+50
Total w/o nucl.: -5.202202065475E+50
```

On 128 CPUs the corresponding values are:

```
Total w/o nucl.: -5.202214535230E+50
Total w/o nucl.: -5.202213903648E+50
Total w/o nucl.: -5.202202065518E+50
```

Performance Results

All benchmark results were obtained on the FZJ JUMP system (IBM p690+). The code does not use threads.

SUCCEsSs				
	64	128	256	512
<i>summed up wall-clock time of iterations 2 to 10</i>				
JUMP	1193.57s	601.53s	315.65s	176.71s
<i>efficiency (summed up wall-clock time)</i>		128	256	512
JUMP		0.992	0.952	0.893
<i>wall-clock time (mm:ss)</i>	64	128	256	512
JUMP	45:25	23:15	12:33	07:26

Performance Characteristics

The most important kernel in terms of performance is a piecewise parabolic method. The code explicitly stresses I/O, and is also suitable for vector architectures.

Portability

The code currently runs on the following architectures:

- IBM Regatta Power 4/5 (RZG, FZJ, and HPCx Edinburgh)
- MareNostrum (Barcelona)
- Cray XT3 Jaguar
- NCCS Oak Ridge
- Linux cluster (AMD Opteron)

6.11 Fenfloss

Description

Fenfloss is a Finite Element package designed for the calculation of flows through water turbines. It can also be used for other purpose like blood flow. ALE techniques for fluid structure interaction are under development. The code can be used for both stationary and unstationary flow. For benchmarking purposes, a small grid generator program named "pagi" is provided which generates a rectangular grid replicated for any number of $n_x \times n_y \times n_z$ processors. In this way a simple scalable test grid is generated. The calculation is performed with suitable boundary conditions. Fenfloss is owned by the Institute for Hydraulic Machines of the University of Stuttgart.

Compilation and Execution

There are makefiles provided for pagi (in pagi) and fenfloss (source_fenfloss). The variables and arrays are defined as REALs (4 byte accuracy). Ideally, full production runs should be performed using 8-byte accuracy. Automatic compiler switches must be used to perform the conversion. In the main program the size of a large buffer in a common block has to be adjusted (e.g. parameter (lmax=10000000)).

Performance Metric

The version we tested does not contain special timing measurements. The overall time is to be measured with a fixed number of time steps. They will be set in the file flow.stf. IO is included as well as the setup phase for the calculations of the elements and stiffness matrices.

Validation

The output file "flow.info" contains residual errors and serves as proof of correctness.

Performance Results

Full performance results are currently available for the real*4 case on the Jülich Jump machine using three different data sets. Note that these results are obtained in a weak scaling mode with a fixed problem size (of 2000 nodes) per processor.

Fenfloss weak scaling, 2000 nodes per processor				
<i>time per timestep (50 steps)</i>	64	128	256	512
	0.425s	0.538s	0.609s	0.863s
<i>efficiency</i>		0.79	0.88	0.71
<i>wall-clock time</i>	21.27 s			

Performance Characteristics

The code is vectorized in all essential parts, in element generation, stiffness matrix assembly and linear solvers. The performance of these parts depends significantly on sustained memory bandwidth. The linear Krylov space solver involves a sparse matrix-vector multiplication which largely determines the time for the simulation, and a dot-product which may impose restrictions on machines with many processors. Bandwidth as well as latency of the inter-node network are important. IO bandwidth is not a bottleneck if many time steps are needed.

Portability

The code has been ported and is running on PC-Clusters and on NEC SX-8. No external libraries are needed.

6.12 GENE

Description

GENE is a gyrokinetics code for the simulation of plasma turbulence and is a so-called continuum (or Vlasov) code. Nonlinear gyrokinetic equations are solved on a fixed grid in five-dimensional phase space. All differential operators in phase space are discretised by fourth-order (compact) finite differences. GENE is of high relevance for the European plasma fusion community especially for the ITER project. The code can deal with arbitrary toroidal geometry and retains full ion/electron dynamics as well as magnetic field fluctuations. At present, GENE is the only plasma turbulence code in Europe with such capabilities.

Compilation and Execution

To install, go to the `gene11` directory and type: `'gmake install'`. The next step is to call the program `'newprob'` to make a problem directory in which the problem description files are copied and can be modified. The problem directories can be renamed freely.

Go to the problem directory you want to modify and type in the problem directory:

```
> gmake -f ../makefile
```

to compile the GENE code. In the subdirectory named after the operating system you are working on, you find an executable with the ending `_gene` and a link to the `par` file one directory up.

Now you can run the executable in a parallel environment and use as many OpenMP threads as required by setting the environment variable `OMP_NUM_THREADS`. The output directory is specified in the `par` file with the parameter `'diagdir'`. After running the program it contains the output files.

Performance Metric

The standard output of `gene` contains a line starting with `'time per time step'`. This time in seconds is the performance metric. It excludes IO and setup times when parameters are set properly.

Validation

The output directory contains an ascii file named `'nrg.dat'`. This file contains summary values for certain times. The 1st line contains one floating point number stating the time. The following lines are for each particle species. These lines contain eight floating point numbers each. These numbers should be validated. They must be identical except those being very small ($<1e-10$).

Performance Results

The benchmark results were obtained on the FZJ JUMP system (IBM p690+). Results on 64 CPUs are not available due to memory constraints.

<i>time per timestep</i>	64	128	256	512
JUMP		23.0 sec	11.1 sec	5.6 sec
<i>efficiency (timesteps)</i>		128	256	512
JUMP			1.0360	0.991
<i>wall-clock time (min:sec)</i>	64	128	256	512
JUMP		< 1:00	< 1:00	< 1:00

Performance Characteristics

The most important kernel of the code is an FFT. GENE especially stresses the interconnect (FFT, global reductions, all-to-all communication) and the memory bandwidth.

For matrix transposition in 2D FFT, interconnect latency becomes critical. For global sums of large data packets, interconnect bandwidth becomes critical. By selection of appropriate problem sizes, a GENE benchmark can help to discriminate between well and badly performing interconnects.

Portability

The code currently runs on the following architectures:

1. IBM Power4/5
2. IBM PPC (MareNostrum)
3. IBM BlueGene/L
4. SGI Altix 4700
5. Cray XT3

6.13 PEPC

Description

PEPC - Pretty Efficient Parallel Coulomb-solver - is a parallel tree-code for rapid computation of long-range Coulomb forces in N-body particle systems. Based on the original Barnes-Hut algorithm, the code uses successively larger multipole-groupings of distant particles to reduce the computational effort in the force calculation from the generally unaffordable $O(N^2)$ operations needed for brute-force summation, to a more amenable $O(N \log N)$ complexity.

The parallel version is a pure MPI implementation of the Warren-Salmon 'Hashed Oct Tree' scheme, including several different variations of the tree traversal routine - the most challenging component in terms of scalability. PEPC is divided into kernel routines and 'front-end' applications, which currently include a skeleton molecular dynamics program (PEPC-E), and a code for simulating laser- or beam-plasma interactions (PEPC-B). There is also a gravitational version (PEGS) for modeling astrophysical discs, developed in collaboration with the University of Cologne.

PEPC-B is unique in the field of plasma physics. It is the first code to employ a mesh-free algorithm for modeling kinetic (non-fluid) phenomena such as non-local electron transport and particle acceleration in dense plasmas. In contrast to traditional mesh-based Particle-in-Cell codes, PEPC can operate in a fully collisional regime, and can tackle open-boundary problems. Currently it is being used to investigate ion acceleration in Petawatt laser-plasma interactions - issues highly relevant to the Fast Ignitor Laser-Fusion concept (HIPER, ELI). Results from these studies are already having an impact on experimental campaigns at major laboratories (Rutherford-Appleton Lab, UK; GSI-Darmstadt; MPQ-Garching; U. Jena).

Compilation and Execution

PEPC provides a configure script for the configuration of the source code for a special machine. On Jump configure should be called as follows:

```
./configure OPT=-O4 -qipa=inline=key2addr -qipa=inline=make_hashentry
```

To create the executable of PEPC (bin/pepcb), execute make in the top directory:

To run PEPC, first enter or create a run directory. This is where all the main output will appear. A number of subdirectories pe0000, pe0001, ... pe<P-1> must exist or be created prior to the run, depending on the number of CPUs requested (P). The input files (run.h, ...) for testing and running the benchmark are provided in a directory ./bench_deisa.

Performance Metric

The performance is measured by the wall clock time spent in one iteration of the iteration loop. For preparing the benchmark run PEPC calculates 2 iterations for 20 million particles and creates a restart file. The benchmark run starts from this restart file and calculates 10 iterations. The average time for one iteration (in seconds) can be found at the end of the output file in the line starting with 'Loop total, average:'. The total wall clock time is reported in the output line starting with 'Total run time:'. Other timings and the MFlops/CPU rate have to be measured outside PEPC, e.g. using hpmcount on Jump.

Validation

The initial total energy (= sum of kinetic and potential energies) in the output file at the start of the benchmark run should be:

Total: 9.81574520 (normalised code units)

This may differ slightly from machine-to-machine depending on the random number generator. After 10 iterations, the final total energy should be:

Total: 9.81578869 (norm units)

Performance Results

Performance results on: Jump IBM Power4 p690

Compiler options: -O4 IPA

Configuration: neutral charge sphere comprising 20 million particles (fixed problem size) starting from restart file (2 timesteps for initialisation)

Timings for timestep loop (10 steps):

NEUTRAL CHARGE SHPERE (20 million particles)				
<i>time for 10 timesteps</i>	64	128	256	512
JUMP	162.17s	86.06s	45.72s	26.71s
<i>efficiency (timesteps)</i>		128	256	512
JUMP		0.94	0.94	0.88
<i>wall-clock time</i>	64	128	256	512
JUMP	2672s	1625s	1047s	787s

Performance Characteristics

The PEPC benchmark mainly stresses the communication system with global reduction, all-to-all and point-to-point in nearly equal measure. A major issue is the synchronisation.

Portability

Platforms currently supported by PEPC are IBM Power4, p690, IBM BlueGene/L, Linux x86 based clusters and Cray XD1. PEPC has also been successfully ported to Mare Nostrum and validation runs are underway.

6.14 BQCD

Description

BQCD (Berlin QCD Program) is a hybrid Monte-Carlo program that simulates Quantum Chromodynamics with dynamical standard Wilson fermions. The computations are performed on a four-dimensional regular grid with periodic boundary conditions. The updates are local (i.e., only communication with nearest neighbours is needed). The kernel of the program is a standard conjugate gradient solver with even/odd pre-conditioning. As a consequence all arrays are stored in an even/odd ordered fashion and the four indices are collapsed into a single one. The access to neighbours is handled by lists. The parallelization is performed by regular grid decomposition in the highest 3 dimensions. The values from the boundaries of the neighbouring processors are stored in the same array as the local values. The local values have indices 1, 2, ..., volume/2. The boundary values have indices > volume/2. The memory for the arrays is dynamically allocated during initialization. Apart from rounding errors the portable version of the benchmark program gives identical results for any grid decomposition. The code is on the order of 20,000 lines, mostly in Fortran 90.

Compilation and Execution

The platforms subdirectory contains a number of Makefile includes which describe the settings needed for the various platforms the code has been ported to. If suitable compiler switches are provided, OpenMP parallelism – for which there are directives in the code – is also activated. The architecture dependent parts of the program are encapsulated in a service module. After setting up a symbolic link to Makefile.var, the code should compile by simply calling `make fast`.

The binary `bqcd` is then called via the (possibly platform-specific) MPI startup mechanism (usually `mpirun` or `mpiexec`), specifying the number of MPI tasks. If a hybrid version of the code is run, it may be necessary to take special care of thread placement; and of course the environment variable `OMP_NUM_THREADS` must be appropriately set.

Finally, an input file is specified as an argument to `bqcd`, which provides some physical parameters, the lattice dimensions, and the distribution of the processors to the lattice setup. For the strong scaling runs described here, a lattice size of 48*48*48*96 was used.

Performance Metric

At the end of the run, performance statistics for various parts of the program are printed to standard output. The relevant metric for performance evaluation is given in the line starting with CG. The third column in this line gives the execution time in seconds, the last column the aggregate average performance in GFlop/s. The line starting with TOTAL gives the total execution time including I/O.

Validation

The output file `bqcd.000.1.info` contains an entry

```
PlaQEnergy    0.24963424201660E+00
```

which should be consistent but for the last three digits in all runs with the same input file apart from the lattice dimensions and the CPU distribution.

Performance Results

The results on JUMP for the above described lattice are given in the following table:

BQCD				
	64	128	256	512
<i>CG time</i>	7395s	3564s	1884s	938s
<i>efficiency (timesteps)</i>		128	256	512
		1.037	0.946	1.004
<i>wall-clock time</i>	64	128	256	512
	10369s	5095s	2661s	1186s

The number of OpenMP threads used on JUMP was 4, hence the number of MPI tasks used was equal to the number of cores divided by 4.

Performance Characteristics

The main stress area for this code is memory bandwidth (in a realistic parallel setup even if the local grid fits into the data cache, because even in that case data exchange with process neighbours generates significant memory traffic) and the communication network through exchange of ghost cells with nearest-neighbour processes. The dominating kernel for the hybrid Monte Carlo algorithm is a conjugate gradient (CG) solver.

Portability

The code was successfully run in MPP mode on an Altix 4700, and previously in MPP and hybrid mode on a Hitachi SR8000 (among other platforms). A Mare Nostrum compile and run are still to be performed.

6.15 SU3_AHIGGS

Description

SU3_AHIGGS is a lattice quantum chromodynamics code intended for computing the conditions of the Early Universe. The code has been developed by the research group headed by Professor Kari Rummukainen at the University of Helsinki, the University of Oulu, and CERN. SU3_AHIGGS is written mainly in C and it uses MPI communications. No special libraries are needed to run the program.

The input data for SU3_AHIGGS are 23 lines containing simulation parameters related to temperature, grid spacing, iterations, etc. The simulation starts from a random initial configuration. In this benchmark, the main parameters are set as follows:

```

nx = ny = nz = 32      (grid size in x, y, and z directions)
n_iteration = 40000   (number of iterations)
seed = 5563532       (seed for the random number generator)

```

In typical research simulations, the grid size is usually larger (64^3 – 256^3).

Compilation and Execution

The compilation and execution of SU3_AHIGGS is easy. First set compilers in two *Makefiles*. For example:

```

CC = xlc                (file libraries/Makefile)
MPICC = mpicc_r        (file su3h_n/Makefile)

```

Then run *gmake* in both directories:

```

cd libraries/
gmake all
cd ../su3h_n/
gmake su3_ahiggs

```

The SU3_AHIGGS executable is now ready.

To run the benchmark, copy the executable *su3_ahiggs* and the input files *beta*, *parameters*, and *status* to the working directory and launch the program. For example:

```

poe ./su3_ahiggs -procs 64

```

The input files are the same for all numbers of processors.

The program produces the files *correl*, *measure*, and *status*. Note that the file *status* is used for both input and output. Therefore, if you re-run a benchmark, make sure that you use the original *status* file.

Performance Metric

All performance information is written to the standard output during the execution. The program's performance is measured using the total wall-clock time, which includes all overheads (I/O, setup, etc.). The overheads are, however, very small (0.01–0.5% of the total time in this benchmark).

To get a result for the total wall-clock time, *grep* for *Wallclock* in the standard output.

Validation

There is no inbuilt validation criterion for SU3_AHIGGS, but you can use acceptance statistics for that purpose:

Acceptances (after last start: 40000 iterations):
Kennedy-Pendleton for gauge: 0.999698 (80000 sweeps)
Adjoint acceptance for gauge: 0.927798 (400000 sweeps)
Gaussian overrelaxation for Higgs: 0.995153 (320000 sweeps)
Heat bath for Higgs: 0.990244 (80000 sweeps)

(These results were obtained from a 64 processor run on the JUMP system at FZJ.)

Performance Results

The table below shows scaling results on JUMP (IBM p690 Power4+ 1.7GHz). The wall-clock times include all activities of the program (computation, communication, and I/O). The efficiency figures are calculated directly from the wall-clock times because I/O times are very small (0.01–0.5% of the total time in this benchmark).

EARLY UNIVERSE WITH 32*32*32 GRID POINTS AND 40000 ITERATIONS				
	64	128	256	512
<i>wall-clock time</i>				
JUMP	1821 s	984 s	577 s	386 s
<i>efficiency</i>		128	256	512
JUMP		0.9253	0.8527	0.7474

Performance Characteristics

The main computational task is the multiplication of 3x3 matrices. This task requires a good memory bandwidth. Assembly codes are available to this purpose for many platforms, though they were not used in this benchmark. The hand-written assembly routines may cut the total execution time by more than a factor of two.

SU3_AHIGGS uses a 3D domain decomposition method for parallelization. With many processors, a significant part of time is spent in MPI communications. A good interconnect is therefore of great importance for the scalability. In typical research simulations where the grid size is 64^3 – 256^3 , the code scales considerably better than here (up to several thousands of processors).

Portability

SU3_AHIGGS has been earlier tested on several platforms during the procurement process of CSC's new supercomputer. Within DEISA, the benchmark has been executed so far on two systems: FZJ's IBM p690 and SARA's SGI Altix. No problems have been encountered in any platforms. Below is the acceptance statistics from a 64 processor simulation executed at SARA:

Acceptances (after last start: 40000 iterations):
Kennedy-Pendleton for gauge: 0.999735 (80000 sweeps)
Adjoint acceptance for gauge: 0.927174 (400000 sweeps)
Gaussian overrelaxation for Higgs: 0.99506 (320000 sweeps)
Heat bath for Higgs: 0.989998 (80000 sweeps)