Technical Reports

CENTRO EURO-MEDITERRANEO
PER I CAMBIAMENTI CLIMATICI

SCO – Scientific Computing and Operations Division

ORCA025: Performance Analysis on Scalar Architecture

Italo Epicoco
University of Salento, Lecce

Silvia Mocavero
Scientific Computing and Operations, CMCC

Enrico Scoccimarro
Istituto Nazionale di Geofisica e Vulcanologia ((INGV) and Numerical Applications and Scenarios Division, CMCC

Giovanni Aloisio
Scientific Computing and Operations, CMCC
University of Salento, Lecce

Centro Euro-Mediterraneo
per i Cambiamenti Climatici
www.cmcc.it

November 2008 ■ TR29

ORCA025 Performance Analysis on Scalar Architecture

Summary

This technical report describes the porting and performance evaluation activities performed on ORCA025 code, implementing the global ocean general circulation model (OGCM) OPA. The code, currently available and optimized on vector architectures, has been ported on HP XC6000 Itanium2 scalar cluster, provided by the associate partner SPACI. The activity is mainly focused to evaluate how a scalar architecture based on Itanium2 processor behaves with oceanographic model that traditionally run on vector clusters. Performance analysis of the parallel code showed good results in terms of scalability.

Keywords: ORCA025, performance analysis, HP XC6000, Itanium2, HPC.

JEL Classification: C63

Version: 1.0
Release Date: 25.11.2008

Address for correspondence:
Italo Epicoco
Faculty of Engineering
Via per monteroni
73100 Lecce, Italy.
Ph: +39 0832 297235 E-mail: italo.epicoco@unile.it

Table of Contents

Introduction

This technical report describes the porting of ORCA025 code, developed for both vector and scalar architectures, on the HP XC6000 Itanium2 scalar cluster and the analysis of its performance.
ORCA025 code implements the global ocean general circulation model (OGCM) OPA (Ocean Parallelise) model [1] with an horizontal resolution of 0,25°. The code is written in Fortran 77 with some routines in Fortran 90 and others in C. It has been parallelized using the MPI library. The code, that we considered, is optimized to best suite on vector processors.
The main goal of performance evaluation in terms of scalability, on scalar architectures, is the possibility to use both scalar and vector clusters to run simulations within CMCC activity. Vector resources could not be always available: if the code shows good performance on scalar architectures, these resources (characterized by a high number of CPUs) can be used in addition to vector clusters, minimizing jobs queuing time.

1. Porting of ORCA025 code on Itanium2

The porting of ORCA025 parallel code (provided by INGV partner) on the scalar architecture needed of some changes during both the compilation and execution steps. Makefiles have been modified in order to optimize the use of Intel C and Fortran compilers. Some execution parameters have been changed to solve problems related to the memory limits on the single Itanium2 compute node.

1.1. Compilation

In order to compile ORCA025 parallel code, Intel *icc* and *ifort* compilers have been used instead of *sxmpicc* and *sxmpif90* compilers [2], used on vector SX clusters. Some optimization options have been introduced during the compilation to exploit Intel compilers features [3]. In particular we have used the following compilers options:

- **tpp2** : target optimization to the Itanium 2 processor.
- **Ip** : enables single-file IP optimizations (within files). With this option, the compiler performs inline function expansion for calls to functions defined within the current source file.
- **mp1** : improves floating-point precision. This option disables fewer optimizations and has less impact on performance than *–mp*. The *-mp* option restricts optimization to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI* and IEEE standards.
- **r8** : defines REAL declarations, constants, functions, and intrinsic as DOUBLE PRECISION (REAL*8), and defines COMPLEX declarations, constants, functions, and intrinsic as DOUBLE COMPLEX (COMPLEX*16).

ORCA025 code includes several features that can be enabled or disabled through several compilation keys. The *Makefile* has been modified in order to take into account the needed keys as reported in the following table:

key_orca_r025	key_flxqsr	key_resteuler
key_freesurf_cstvol	key_sst	key_mpi_isend
key_zdftke	key_tradmp	key_flx
key_hpgimplicit	key_temdta	key_flx_ecmwf
key_trahdfiso	key_saldta	key_tau
key_trahdfcoef2d	key_saldta_monthly	key_tau_ecmwf
key_dynhdfcoef2d	key_diahth	key_sst_ecmwf
key_dynhdfbilap	key_mpp	key_diamean
key_trahdfeiv	key_mpp_mpi	key_vorene_ens
key_convevd	key_r4	

1.2. Code changes

ORCA025 has been modified to solve some issues related to memory limits on the Itanium2 node. In particular, a call to the all-to-all *MPI_Allreduce* operation involves a very large dataset compared with memory capacity of each node. Thus, the 'reduce' operation applied on an array of *dim* elements has been split into *n* sequential calls with array of *dim/n* elements. This transformation implies a loose of performance but it is strictly required in order to complete the elaboration on XC6000 cluster. Follows code changes:

Original code

```
SUBROUTINE mpprsum(ptab,kdim)
……
#  elif defined key_mpp_mpi
C
C MPI VERSION
C
      INTEGER ierror
C
      CALL mpi_allreduce(ptab,pwork,kdim,mpi_double_precision
      $    ,mpi_sum,mpi_comm_world,ierror)
      ptab=pwork
```

```
C
#  else
……
RETURN
END
```

Modified code

```
SUBROUTINE mpprsum(ptab,kdim)
……
#  elif defined key_mpp_mpi
C
C MPI VERSION
C
INTEGER ierror
INTEGER i
INTEGER my_len
REAL ptab_tmp(kdim),pwork_tmp(kdim)
C
      DO i=1, kdim, 1021
         IF ((kdim - i + 1) .le. 1021) THEN
           my_len = kdim - i + 1
         ELSE
           my_len = 1021
         END IF
         ptab_tmp = ptab(i:i + my_len - 1 )
         CALL mpi_allreduce(ptab_tmp,pwork_tmp,my_len,
         $    mpi_double_precision,mpi_sum,mpi_comm_world,ierror)
         ptab(i:i + my_len - 1)=pwork_tmp(1:my_len)
      END DO
C
#  else
……
RETURN
END
```

---

2. ORCA025 code analysis on Itanium2

2.1. Configuration

ORCA025 parallel code performances have been evaluated running it on the HP XC6000 cluster, named *sigma.unile.it.* The cluster is located in the SPACI Consortium Lecce site. In this section we describe the hardware and software configuration used during the evaluation of the performance of the application

2.1.1. Hardware

The HP XC6000 cluster is a scalar machine equipped with 64 nodes with 2 processors Itanium2 (Madison - 1.4 Ghz), a local scratch disk SCSI Ultra-320 with 36 GB 15000 rpm and 4 GB of RAM for each node, for a total of 128 processors. The nodes are interconnected each other and with the storage system as depicted in Figure 1. In detail the nodes are classified into:

- n. 4 service node (n61-n64): devoted for the cluster management services like NFS, LSF, I/O interface, boot, shutdown, etc. They also represent the front-end nodes for the users.
- n. 60 compute node (n1-n60): devoted only for computing.

Two interconnection networks characterize the cluster:

1. a Gigabit Ethernet network
2. a Qsnet-II Elan4 network

The Gigabit Ethernet network is used both to manage the cluster and to share the filesystem among nodes through NFS. The Qsnet-II network is realized using a Quadrics Elan 4 interconnection switch at 800 MB/s characterized by a very slow latency and only used for MPI communications.

Nodes named n61 (sigma4) and n62 (sigma3) are I/O nodes; they are NFS servers exporting user's home to all of the other nodes. In particular, *home* and *home1* partitions are allocated on a heterogeneous external storage system, the HP SAN (Storage Area Network) EVA 3000 with 720GB, connected to the n61 and n62 nodes through a fiber channel connection at 2 Gb/s. n64 (sigma1) node is the head node used for management operations (boot, shutdown, etc).

All of the nodes mount both the volumes *home* and *home1* using the NFS server, through the two I/O nodes n61 and n62. Thus, when one or more applications, characterized by a high number of I/O operations, are executing, the NFS server could represent a bottleneck. To solve the problem, there is a local scratch on each node that can be used to store temporary computing data; in this way it is possible to profitably use the resources and optimize the job execution time.

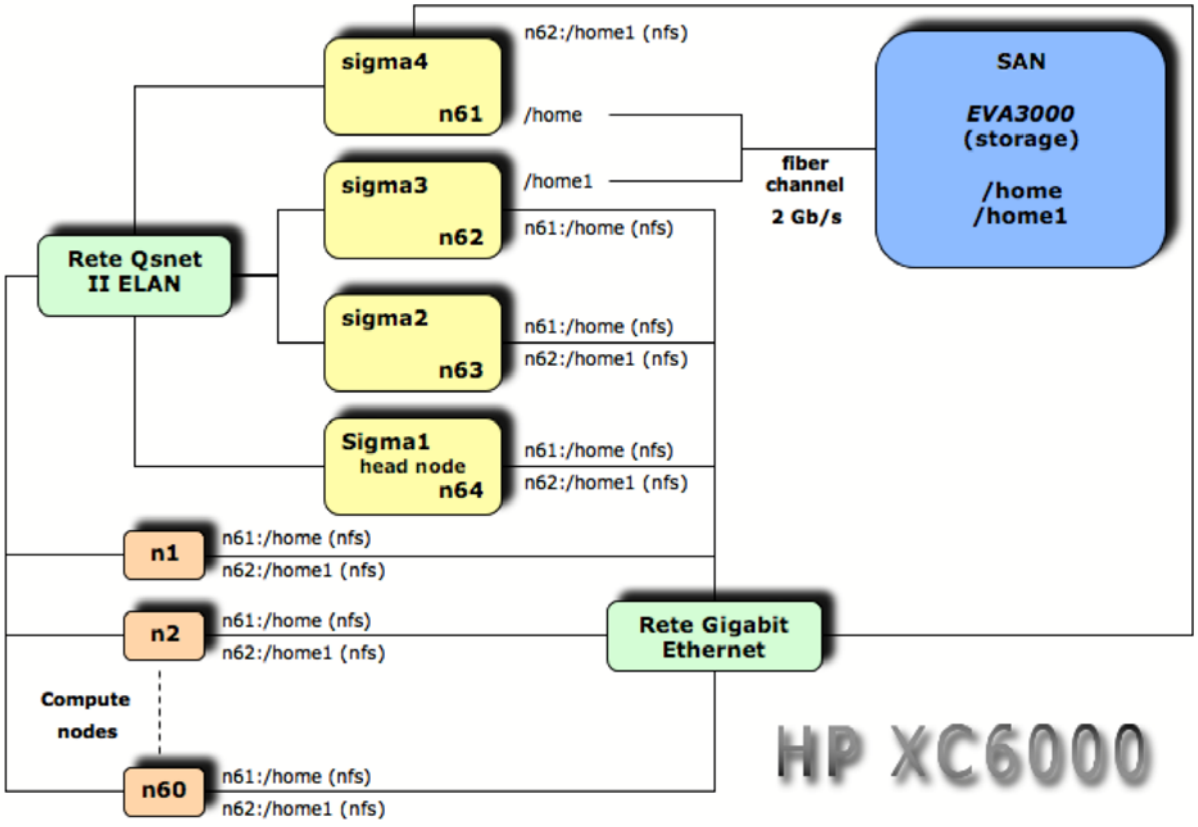

Figure 1 - HP XC6000 cluster architecture

2.1.2. Software

ORCA025 application can be launched using a shell script. Using this script the user can set execution parameters (the time range for the whole execution, the time steps for output saving, the execution directory, the executable path, etc.). The bottleneck introduced by NFS (see 2.1.1) has been overcome using the local disk of the compute nodes; the shell script has been extended in order to properly handle same preliminary operation; a local scratch directory (LOCAL_SCRATCH) has been introduced; input/output files are copied from the LOCAL_SCRATCH to the GLOBAL_SCRATCH (in the user home) and vice-versa. Follows modified sections of the shell script, in order to allow read/write access to the local disk:

```
BASEDIR=/home1/epico/ORCA/POG05B
LOCAL_SCRATCH=/tmp/scratch/orca
GLOBAL_SCRATCH=/home1/epico/ORCA/test
OUTDIR=${LOCAL_SCRATCH}
ARCHDIR=${GLOBAL_SCRATCH}/POG05B

b=`expr $RMS_RANK % 2`;
if [ $b -eq 0 ]; then
      mkdir -p ${OUTDIR}
      workdir=${OUTDIR}
      indir=$GLOBAL_SCRATCH/data/ORCA025
      bindir=${BASEDIR}/bin
      FORCING=${indir}
      SPIN=${indir}
      cd ${workdir}
      rm -f ${workdir}/*
      ……
elif [ $b -eq 1 ]; then
      workdir=${OUTDIR}
      indir=$GLOBAL_SCRATCH/data/ORCA025
      bindir=${BASEDIR}/bin
      FORCING=${indir}
      SPIN=${indir}
      cd ${workdir}
      while ! test -r tmp_`hostname`
      do
            a=1
      done
      rm tmp_`hostname`
fi
./opa025_${KEYMPP}cpus
……
```

```
b=`expr $RMS_RANK % 2`;
if [ $b -eq 0 ]; then
      touch `hostname`
elif [ $b -eq 1 ]; then
      while ! test -r `hostname`
      do
            a=1
      done
      rm opa025_${KEYMPP}cpus
      rm coordinates* ERA40* namelist ECMWF* runoff*
      rm maskglo.nc bathymetry Levitus98* EMPave_old.dat
      rm `hostname`
      if [ $RMS_RANK -gt 1 ]; then
            rm solver.stat time.step STmean.diagnostic MHT.diagnostic
            rm energy.diagnostic date.file EMPave.dat DCT.diagnostic OK
      fi
      /bin/mv * ${ARCHDIR}_${nyear}${month}${startday}/.
fi
touch $RMS_RANK
while ! test -r $RMS_RANK
do
      a=1
done
/bin/mv $RMS_RANK ${ARCHDIR}_${nyear}${month}${startday}/
cd ${ARCHDIR}_${nyear}${month}${startday}
k=0
while [ $k -lt ${KEYMPP} ]
do
      while ! test -r $k
      do
            a=1
      done
      k=$(expr $k + 1)
done
```

Tests have been performed simulating one day (*ndays=1*), fixing a time step of 720 seconds and saving measured variables every 120 time steps (*nmean=120*). With this configuration, the computed variables are saved into a file at the end of each simulated day; in our case this corresponds to the end of the entire simulation. OPA model implementation allows saving execution status into such a restart file. This option is generally used for the simulations characterized by a long life (over a week) time interval, in our case we disabled it.

2.2. Profiling

Tests have been executed starting from 32 processors and increasing the number of them until 64. Taking into account the memory available on each node, and considering that the application allocates a peak of 30GB, we can not use less then 16 nodes. On the other hand due, to the concurrent usage of the cluster by other users, we limited the test up to 64 processors.

2.3. Tracing

Early tests have shown that the execution times of the application, launched with the same configuration, were deeply different for different runs. In order to explain this behavior, the "Intel® Trace Analyzer and Collector 7.0 for Linux" suite has been used. It consists of the following two modules:

1. The Intel Trace Collector (ITC), a tool (for MPI applications) able to collect into a tracing file several parameters about the execution of the application. These information can be used to analyze the application performance;
2. The Intel Trace Analyzer, a graphic tool for the visualization of data collected into the tracing file by the ITC.

In Figure 2 execution time spent for each process is decomposed in communication time (red segment) and computing time (blue segment); in particular, a zoom in of the first 200 seconds of the execution has been taken into account. The chart highlights an unbalanced workload on P30 and P31 processes. Indeed while all of the processes reach the *MPI_Allreduce* function call after about 20 seconds, P30 and P31 are still computing and will be ready to participate to the collective operation only after 200 seconds. This obviously produces a general inefficiency of the application. Several run tests have been performed and results have shown that processes characterized by an unbalanced workload are not always the same. The experiments demonstrate that the slowing down is not due to the code implementation, but is caused by the use of two specific nodes (n33, n34).
Computation on these nodes was generally very slow, independently from the processes running on them (i.e. in P30 e P31 in Figure 2). When MPI collective functions are performed, the activity on these nodes slows down the activity of each others. Excluding these two nodes from the pool of

computing nodes, the workload gets balanced among processes. Without the faulty nodes, the execution time measured for many runs with the same configuration became similar reducing the variance of the measurements.

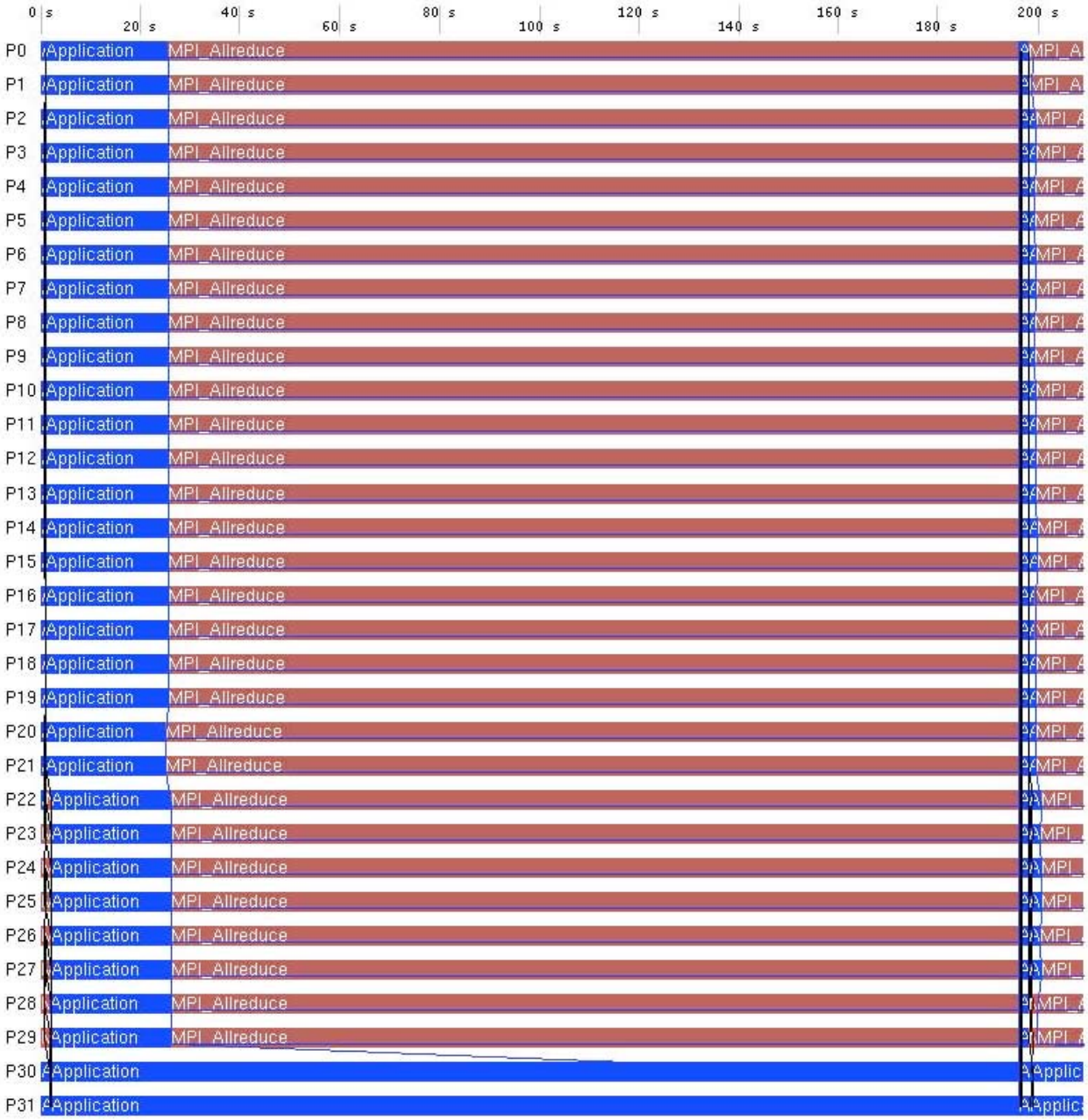

Figure 2 – Performance analysis using Intel Trace Collector & Analyzer tools. Computing time (blue) and communication time (red) during the former 200 seconds of execution.

2.4. Performance

In order to evaluate ORCA025 performance we got the average execution time over 5 runs for different configurations increasing the number of processes from 32 up to 64, with a step of 4. The charts in Figure 3 show efficiency, speed-up and execution time. Since sequential time cannot be evaluated due to the memory limit on a single node, efficiency and speed-up have been computed considering as referring time that one spent to execute the application on 32 processors. The following formulas have been used:

$$Speed - up_N = \frac{Parallel\ Execution\ Time\ on\ 32}{Parallel\ Execution\ Time\ on\ N} \qquad \text{where } N \geq 32 \text{ (number of processes)}$$

$$Efficiency_N = \frac{Parallel\ Execution\ Time\ on\ 32}{\frac{N}{32} \times Parallel\ Execution\ Time\ on\ N}$$

A speed-up of 2 is expected when processors number is equal to 64. In Table 1 execution time, efficiency and speed-up are listed.

Table 1 – Execution time, efficiency and speed-up when processors number increases

# Processors	Execution time (sec)	Speed-up	Efficiency
32	987,20	1,00	1,00
36	904,80	1,09	0,97
40	832,80	1,19	0,95
44	777,20	1,27	0,92
48	717,80	1,38	0,92
52	673,80	1,47	0,90
56	642,60	1,54	0,88
60	592,40	1,67	0,89
64	561,00	1,76	0,88

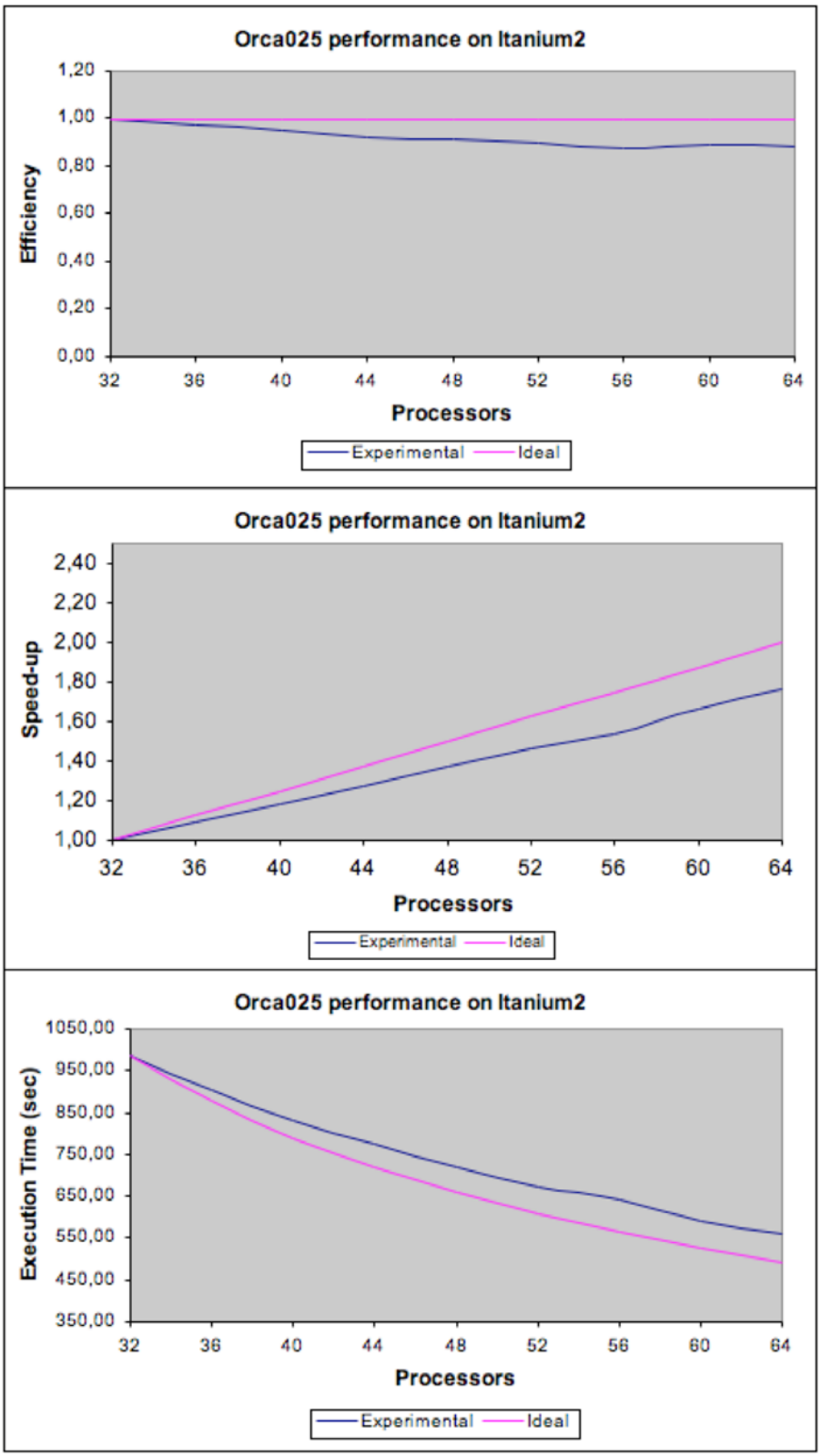

Figure 3 - Profiling of ORCA025 code: efficiency, speed-up, execution time when processors number increases

Table 2 reports the execution time, highlighting the processing time and the start-up time. In Figure 4, related execution time chart is shown. Start up operations include stage-in of the input files from global (GLOBAL_SCRATCH) to local (LOCAL_SCRATCH) working directory, the clean-up of temporary files at the end of the job and the copy of output file from local to global working directory.

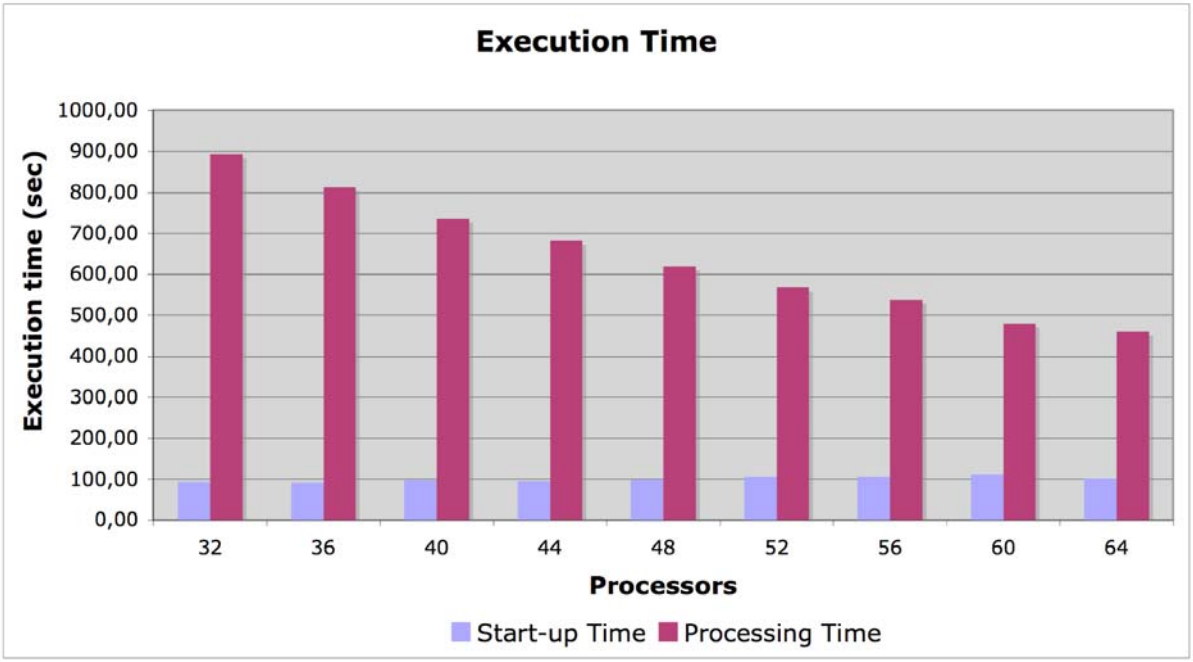

Figure 4 – Execution time components chart

Table 2 - Execution time components: start up time and processing time

	Start-up Time	Processing Time	Execution Time
32	94,20	893	**987,20**
36	92,80	812	**904,80**
40	96,80	736	**832,80**
44	95,20	682	**777,20**
48	98,80	619	**717,80**
52	105,80	568	**673,80**
56	105,60	537	**642,60**
60	111,40	481	**592,40**
64	101,00	460	**561,00**

In order to analyze the scalability and the efficiency of the parallel implementation we must consider only the 'processing time' and not the whole 'wall clock time' hence excluding the start-up time that has been introduced only for optimizing the I/O operations through the use of the local disk rather then the shared disk; moreover considering that 4GB of main memory are not sufficient for handling the computation of the model on one node and that 16 nodes are needed for computing the model without the use of disk swap space, we referred the scalability to the processing time measured in parallel on 32 processors. The charts in Figure 5 show scalability in terms of efficiency, speed-up and execution time, taking into account only the processing time, when the number of processors increases. Table 3 shows details on execution time, considering the average on 5 runs.

Table 3 - Processing time, efficiency and speed-up when processors number increases

# Processors	Processing time (sec)	Speed-up	Efficiency
32	893	1,00	1,00
36	812	1,10	0,98
40	736	1,21	0,97
44	682	1,31	0,95
48	619	1,44	0,96
52	568	1,57	0,97
56	537	1,66	0,95
60	481	1,86	0,99
64	460	1,94	0,97

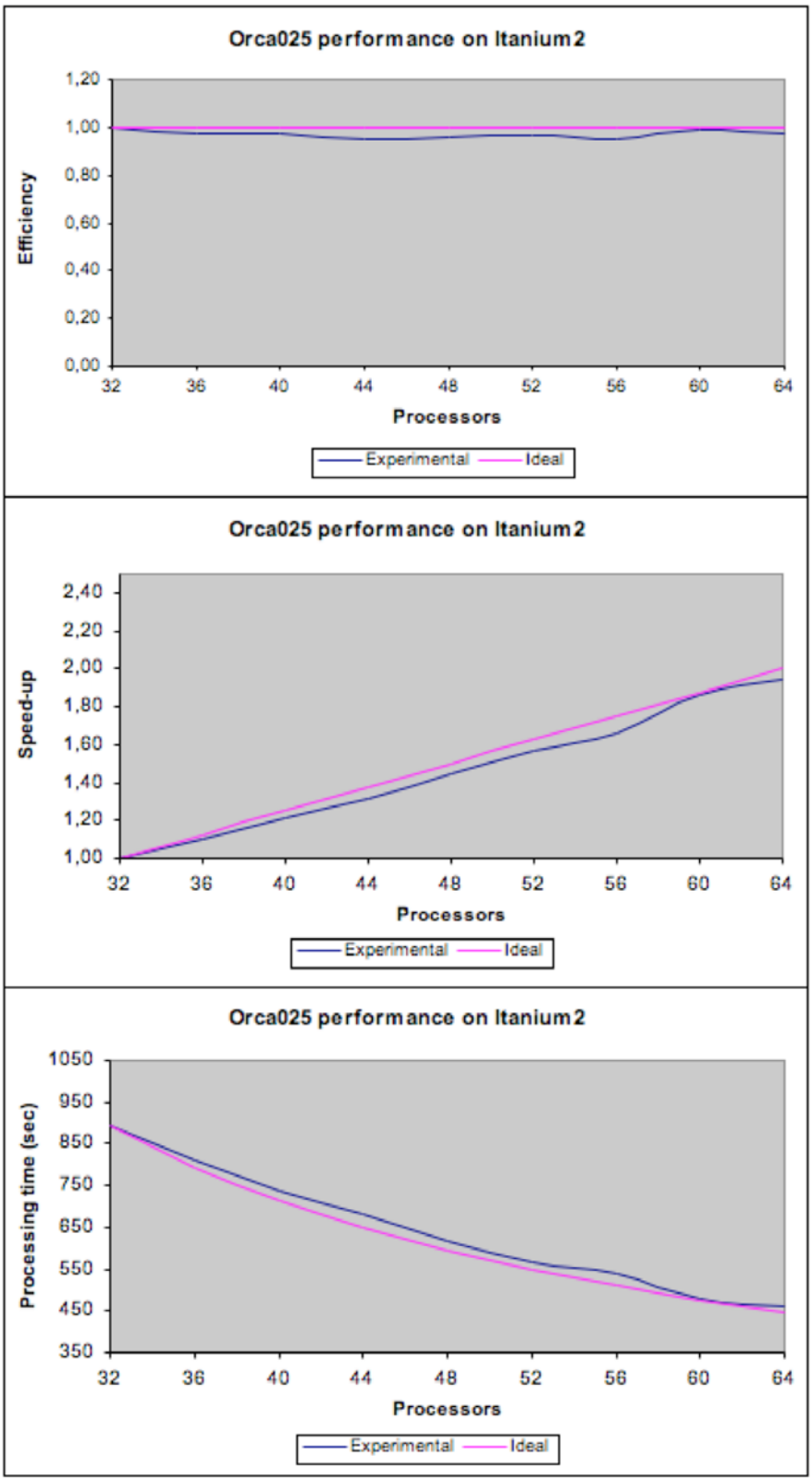

Figure 5 - Profiling of ORCA025 code: efficiency, speed-up, processing time

A more detailed analysis on the execution time highlights that the processing time is given by 3 main factors: I/O time, represents the time needed for I/O operations occurred during the simulation; MPI time, that represents the communication overhead introduced by the parallel implementation; computation time that is the time for the operations devoted only for computation. Table 4 and Figure 6 report the measured times.

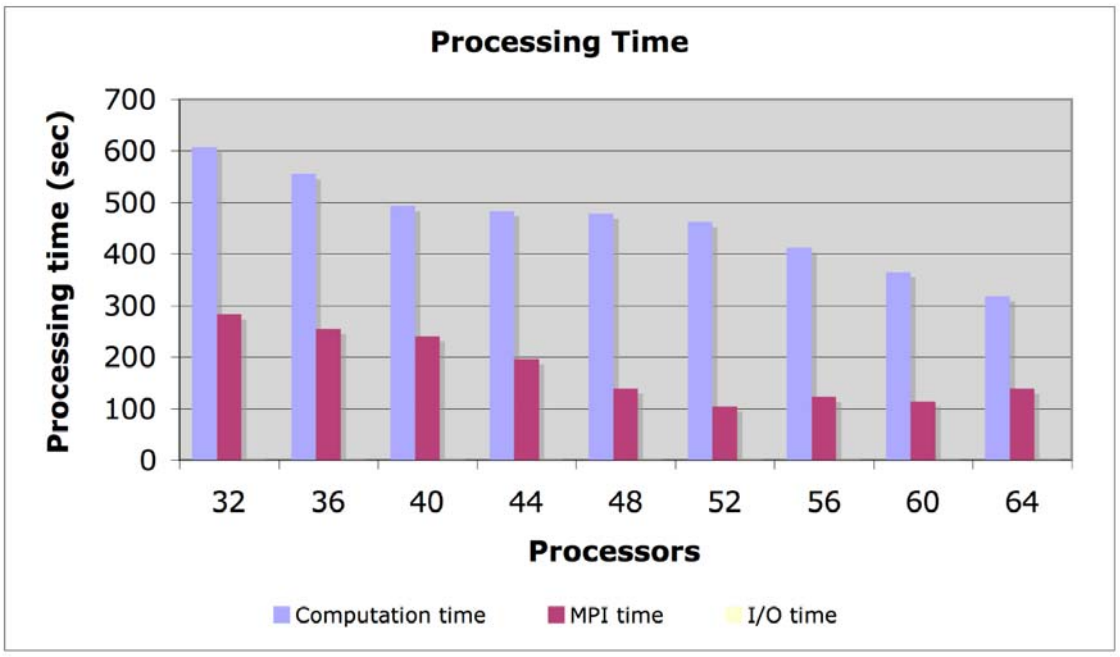

Figure 6 - Processing time components

Table 4 - Processing time components: I/O time, MPI time and computation time

	I/O Time	MPI Time		Computation Time	Processing Time
		Average	Standard deviation		
32	1,39	283,63	24,97	607,98	**893**
36	1,38	254,32	18,6	556,3	**812**
40	1,41	240,89	22,84	493,7	**736**
44	1,4	196,65	16,3	483,95	**682**
48	1,5	138,95	15,11	478,55	**619**
52	1,47	103,9	13,15	462,63	**568**

56	1,42	122,74	15,77	412,84	**537**
60	1,48	114,25	13,08	365,27	**481**
64	1,41	139,78	17,76	318,81	**460**

The analysis illustrated within the present report induces to the following conclusions:

- ORCA025 code shows optimal results in terms of scalability, referring to execution time on 32 processors. However, it is necessary to evaluate scalability using the sequential time as starting point, even if this kind of test must be performed on a scalar cluster which satisfies application memory requirements
- Avoiding physic limits due to the cluster architecture, execution time can be improved.

Bibliography

[1] Gurvan Madec, Pascale Delecluse, Maurice Imbard et Claire Lévy, *OPA 8.1 Ocean General Circulation Model Reference Manual*, **Institut Pierre Simon Laplace des Sciences de l'Environnement Global,** 1998.

[2] http://www.ksc.re.kr/user/data/nec_manual/nec/g1af09e/index_frame.html

[3] http://www.intel.com/cd/software/products/asmo-na/eng/346152.htm

[4] http://www.lle.rochester.edu/pub/support/lsf/pjs-contents.html