Performance Analysis for MPI Applications with SCALEA *

Hong-Linh Truong[1], Thomas Fahringer[1]
Michael Geissler[2], Georg Madsen[3]

[1] Institute for Software Science, University of Vienna
Liechtensteinstr. 22, A-1090 Vienna, Austria
{truong,tf}@par.univie.ac.at
[2] Photonics Institute, Technical University Vienna
Gusshausstrasse 27/387, A-1040 Vienna, Austria
geissler@tuwien.ac.at
[3] Institute of Physical and Theoretical Chemistry, Technical University Vienna
Getreidemarkt 9/156,A-1060 Vienna, Austria
gmadsen@theochem.tuwien.ac.at

Published in Proc. of the 9th European PVM/MPI User's Meeting (EuroPVMMPI 2002), Linz, Austria, September 29th - October 2nd, 2002

Abstract. The performance of message passing programs can be challenging to comprehend. In previous work we have introduced SCALEA, which is a performance instrumentation, measurement, analysis, and visualization tool for parallel and distributed programs. In this paper we report on experiences with SCALEA for performance analysis of two realistic MPI codes taken from laser physics and material science. SCALEA has been used to automatically instrument - based on user provided directives - the source codes, to compute performance overheads, to relate them to the source code, and to provide a range of performance diagrams in order to explain performance problems as part of a graphical user interface. Multiple-experiment performance analysis allows to compare and to evaluate the performance outcome of several experiments which have been conducted on a SMP cluster architecture.

1 Introduction

Message passing serves as an effective programming technique for parallel architectures and also enables the exploitation of coarse-grain parallelism on distributed computers as evidenced by the popularity of the Message Passing Interface (MPI).The performance of parallel and distributed message passing programs can be difficult to understand. In order to achieve reasonable performance the programmer must be intimately familiar with many aspects of the application design, software environment, and the target architecture. Performance tools play a crucial role to support performance tuning of distributed and parallel applications.

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

We have developed SCALEA [11, 10] which is a performance instrumentation, measurement, analysis, and visualization tool for parallel and distributed programs that focuses on post-mortem and on-line performance analysis for Fortran MPI, OpenMP, HPF, and mixed parallel programs. The approach supported by SCALEA is that it seeks to explain the performance behavior of each program by computing a variety of performance metrics based on a novel overhead classification [11, 10] for MPI, OpenMP, HPF and mixed parallel programs. In order to determine overheads, SCALEA divides the program sources into code regions and locates whether the performance problems occur in those regions or not. A highly flexible instrumentation and measurement system is provided which can be controlled by program directives and command line options.

A data repository is employed in order to store performance data (raw performance data and performance metrics) and information about performance experiments (e.g. source codes, machine configuration, etc.) which alleviates the association of performance information with experiments and the source code. SCALEA also supports multiple-experiment performance analysis that allows to compare and to evaluate the performance outcome of several experiments. A graphical user interface is provided to view the performance at the level of arbitrary code regions, processes and threads for single- and multi-experiments.

The rest of this paper is organized as follows: Related work is outlined in Section 2. Section 3 presents an overview of SCALEA. Experiments that demonstrate the usefulness of SCALEA are shown in Section 4, followed by conclusions and future work in Section 5.

2 Related Work

Paradyn [7] uses dynamic instrumentation and searches for performance bottlenecks based on a specification language. Performance analysis is done for functions and function calls. SCALEA has a more flexible mechanism to control the code regions for which performance metrics should be determined.

The Pablo toolkit[9] uses event tracing to develop performance tools for both message passing and data parallel programs. It provides an instrumentation interface that inserts user-specified instrumentation in the program. SCALEA is more flexible to control what performance metrics should be determined for which given code regions.

TAU [6] is a performance framework for instrumentation, measurement, and analysis of parallel Fortran and C/C++ programs. TAU supports multiple timing and hardware metrics. However, TAU does not provide a similar flexible mechanism as SCALEA to control instrumentation. SCALEA also provides more high-level performance metrics (e.g. data movement and control of parallelism overhead) than TAU.

Paraver [12] and VAMPIR [8] are performance analysis tools that also cover MPI programs. They process trace files to compute various performance results and provide a rich set of performance displays. SCALEA offers a larger set of performance metrics and enables a more flexible mechanism to control instrumentation and measurements.

EXPERT [3] detects a range of performance properties for message passing, OpenMP, and mixed programs. SCALEA provides a more flexible mechanism to control instrumentation and measurement.

None of the above mentioned tools employ a data repository to organize and store performance data based on which higher-level performance analysis can be conducted. Moreover, these tools lack the support of multiple performance experiment analysis and a flexible mechanism to control instrumentation (except Paradyn) as provided by SCALEA.

3 SCALEA Overview

In this section we give a brief overview of SCALEA. For more details the reader may refer to [11, 10]. Figure 1 shows the architecture of SCALEA which consists of several modules: SCALEA Instrumentation System, SCALEA Runtime System, SCALEA Performance Analysis & Visualization System, and SCALEA Performance Data Repository.

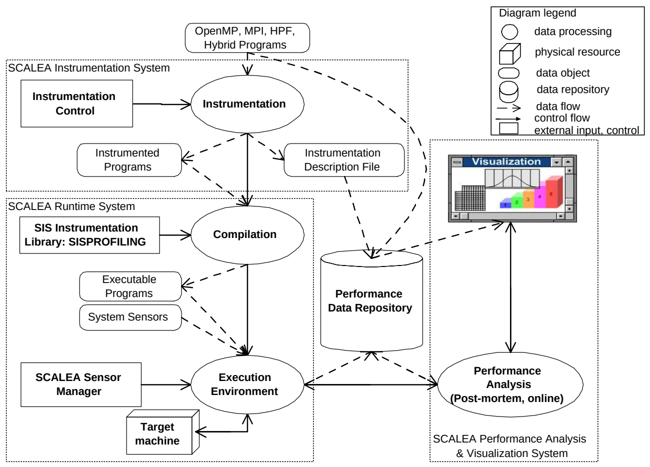


Fig. 1. Architecture of SCALEA

The SCALEA Instrumentation System(SIS) supports automatic instrumentation of Fortran MPI, OpenMP, HPF, and mixed OpenMP/MPI programs. SIS enables the user to select (by directives or command-line options) code regions and performance metrics of interest.

Moreover, SCALEA offers an interface for other tools to traverse and annotate the AST to specify code regions for which performance metrics should be obtained. Based on pre-selected code regions and/or performance metrics, SIS automatically analyzes source codes and inserts probes (instrumentation code) in the code which will collect all relevant performance information during execution of the program on a target architecture.

The SCALEA runtime system supports profiling and tracing for parallel and distributed applications, and sensors and sensor managers for capturing and managing performance data of individual computing nodes of parallel and distributed machines. The SIS profiling and tracing library collects timing, event, and counter information, as well as hardware parameters. Hardware parameters are determined through an interface with the PAPI library [2]. Various interfaces to other libraries such as TAU [6] are also supported.

The SCALEA Performance Analysis and Visualization module analyzes the raw performance data – collected during program executed and stored in the performance data repository – computes all user-requested performance metrics, and visualizes them together with the input program. Besides single-experiment analysis, SCALEA also supports multi-experiment performance analysis The visualization engine provides a rich set of displays for performance metrics in isolation or together with the source code.

SCALEA performance data repository holds relevant information about the experiments conducted which includes raw performance data and metrics, source code, machine information, etc.

4 Experiments

In order to evaluate the usefulness of SCALEA, we present two different experiments covering a laser physics and a material science application that have been conducted on a SMP cluster architecture with 16 SMP nodes (connected by Fast-Ethernet 100Mbps and Myrinet) each of which comprises 4 Intel Pentium III 700 MHz CPUs.

4.1 3D Particle-In-Cell (3DPIC)

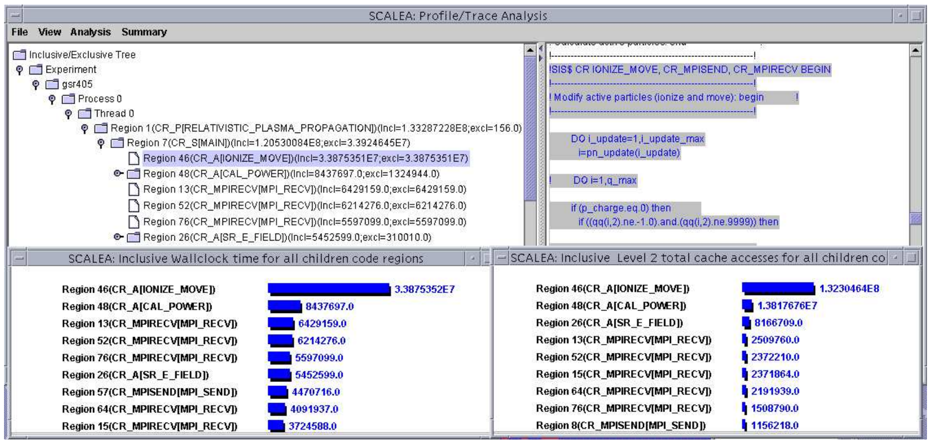
The 3D Particle-In-Cell application [5] simulates the interaction of high intensity ultrashort laser pulses with plasma in three dimensional geometry. This application (3DPIC) has been implemented as a Fortran90/MPI code.

We conducted several experiments by varying the machine size and by selecting the MPICH communication library for Fast-Ethernet 100Mbps. The problem size (3D geometry) has been fixed with 30 cells in x-direction (*nnx_glob*=30), 30 cells in y-direction (*nny_glob*=30), and 100 cells in z-direction (*nnz_glob*=100). The simulation has been done for 800 time steps (*itmax*=800).

Single Experiment Analysis Mode: SCALEA provides several analyses (e.g. *Load Imbalance Analysis, Inclusive/Exclusive Analysis, Metric Ratio Analysis, Overhead Analysis, Summary Analysis*) and diagrams to support performance evaluation based on a single execution of a program. In the following we just highlight some interesting results for a single experiment of the 3DPIC code with 3 SMP nodes and 4 CPUs per node.

The **Inclusive/Exclusive Analysis** is used to determine the execution time or overhead intensive code regions. The two lower-windows in Fig. 2 present the inclusive wallclock times and the number of L2 cache accesses for sub-regions of the subroutine *MAIN* executed by thread 0 in process 0 of SMP node gsr405.

The most time consuming region is *IONIZE_MOVE* because it is the most computation intensive region in 3DPIC which modifies the position of the particles by solving the equation of motion $\frac{d}{dt}(m\mathbf{v}) = q(\mathbf{E} + \frac{\mathbf{v}}{c} \times \mathbf{B})$ with a forth order Runge-Kutta routine. The related source code of region *IONIZE_MOVE* is shown in the upper-right window.



```
Inclusive/Exclusive Tree
  Experiment
    gsr405
      Process 0
        Thread 0
          Region 1(CR_P[RELATIVISTIC_PLASMA_PROPAGATION])(Incl=1.33287228E8;excl=156.0)
            Region 7(CR_S[MAIN])(Incl=1.20530084E8;excl=3.3924645E7)
              Region 46(CR_A[IONIZE_MOVE])(Incl=3.3875351E7;excl=3.3875351E7)
              Region 48(CR_A[CAL_POWER])(Incl=8437697.0;excl=1324944.0)
              Region 13(CR_MPIRECV[MPI_RECV])(Incl=6429159.0;excl=6429159.0)
              Region 52(CR_MPIRECV[MPI_RECV])(Incl=6214276.0;excl=6214276.0)
              Region 76(CR_MPIRECV[MPI_RECV])(Incl=5597099.0;excl=5597099.0)
              Region 26(CR_A[SR_E_FIELD])(Incl=5452599.0;excl=310010.0)
```

```
!----------------------------------------------------------!
!SIS$ CR IONIZE_MOVE, CR_MPISEND, CR_MPIRECV BEGIN
!----------------------------------------------------------!
! Modify active particles (ionize and move): begin        !
!----------------------------------------------------------!

      DO i_update=1,i_update_max
        i=pn_update(i_update)

!      DO i=1,q_max

      if (p_charge.eq.0) then
        if ((qq(i,2).ne.-1.0).and.(qq(i,2).ne.9999)) then
```

SCALEA: Inclusive Wallclock time for all children code regions

```
Region 46(CR_A[IONIZE_MOVE])      3.3875352E7
Region 48(CR_A[CAL_POWER])        8437697.0
Region 13(CR_MPIRECV[MPI_RECV])   6429159.0
Region 52(CR_MPIRECV[MPI_RECV])   6214276.0
Region 76(CR_MPIRECV[MPI_RECV])   5597099.0
Region 26(CR_A[SR_E_FIELD])       5452599.0
Region 57(CR_MPISEND[MPI_SEND])   4470716.0
Region 64(CR_MPIRECV[MPI_RECV])   4091937.0
Region 15(CR_MPIRECV[MPI_RECV])   3724588.0
```

SCALEA: Inclusive Level 2 total cache accesses for all children co

```
Region 46(CR_A[IONIZE_MOVE])      1.3230464E8
Region 48(CR_A[CAL_POWER])        1.3817676E7
Region 26(CR_A[SR_E_FIELD])       8166709.0
Region 13(CR_MPIRECV[MPI_RECV])   2509760.0
Region 52(CR_MPIRECV[MPI_RECV])   2372210.0
Region 15(CR_MPIRECV[MPI_RECV])   2371864.0
Region 64(CR_MPIRECV[MPI_RECV])   2191939.0
Region 76(CR_MPIRECV[MPI_RECV])   1508790.0
Region 8(CR_MPISEND[MPI_SEND])    1156218.0
```

Fig. 2. Inclusive/Exclusive analysis for sub-regions of the *MAIN* program

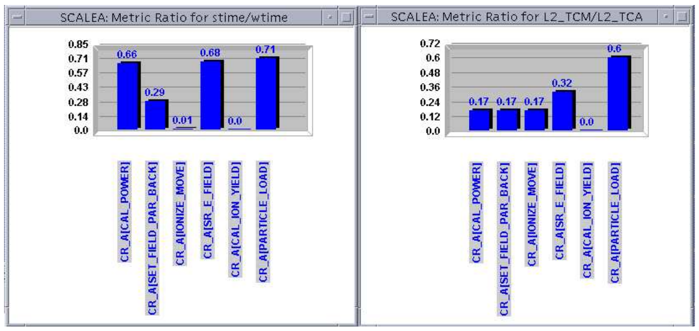
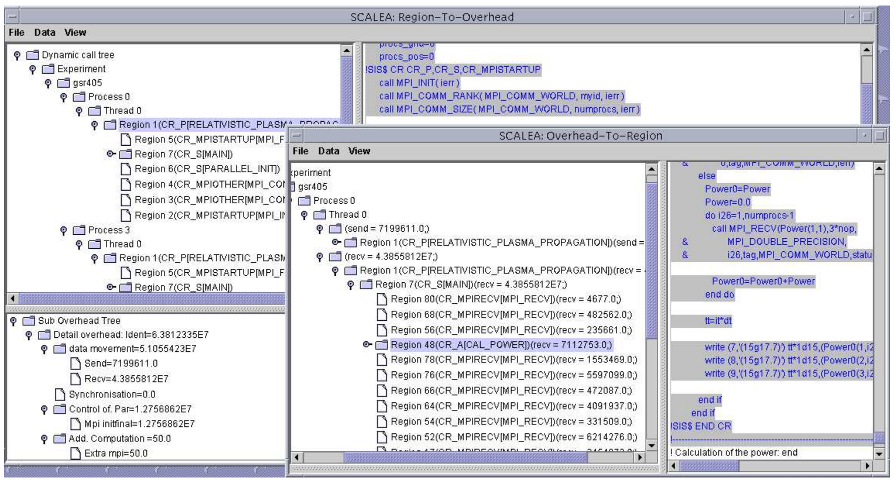


Fig. 3. Metric ratios for important code regions

The **Metric Ratio Analysis** of SCALEA is then used to examine various important metric ratios (e.g. cache miss ratio, system time/wall clock time, floating point instructions per second, etc.) of code region instance(s) in one experiment. Figure 3 shows the most critical system time/wall clock time and L2 cache misses/L2 cache accesses ratios together with the corresponding code regions. The code regions *CAL_POWER*,*SET_FIELD_PAR_BACK*, *SR_E_FIELD*, and *PARTICLE_LOAD* imply a high system time/wall clock time ratio due to expensive MPI constructs (included in system time). Both ratios are rather low for region *IONIZE_MOVE* because this region represents the computational part without any communication (mostly user time). The code region *PARTICLE_LOAD* shows a very high L2 cache misses/L2 cache accesses ratio because it initializes all particles in the 3D volume without accessing it again (little cache reuse).

The **Overhead Analysis** is used to investigate performance overheads for an experiment based on a overhead classification [11, 10]. In Fig. 4, SCALEA's *Region-To-Overhead* analysis (see the left-most window) examines the overheads of the code region instance with number 1 (thread 0, process 0, node gsr405) which corresponds to the main program. The main program is dominated by the data movement overhead (see the lower-left window) which can be further refined to the overhead associated with send and receive operations. We then use the *Overhead-To-Region analysis* to inspect regions that cause receive overhead for thread 0, process 0, and node gsr405 (see the Overhead-To-Region window in Fig. 4). The largest portion of the receive overhead in subroutine MAIN is found in region *CAL_POWER* (7.11 seconds out of 43.85 seconds execution time).



```
Dynamic call tree
  Experiment
    gsr405
      Process 0
        Thread 0
          Region 1(CR_P[RELATIVISTIC_PLASMA_PROPAG
            Region 5(CR_MPISTARTUP[MPI_F
            Region 7(CR_S[MAIN])
            Region 6(CR_S[PARALLEL_INIT])
            Region 4(CR_MPIOTHER[MPI_COI
            Region 3(CR_MPIOTHER[MPI_COI
            Region 2(CR_MPISTARTUP[MPI_IN
      Process 3
        Thread 0
          Region 1(CR_P[RELATIVISTIC_PLASM
            Region 5(CR_MPISTARTUP[MPI_F
            Region 7(CR_S[MAIN])
```

```
Sub Overhead Tree
  Detail overhead: Ident=6.3812335E7
    data movement=5.1055423E7
      Send=7199611.0
      Recv=4.3855812E7
    Synchronisation=0.0
    Control of. Par=1.2756862E7
      Mpi initfinal=1.2756862E7
    Add. Computation =50.0
      Extra mpi=50.0
```

```
      procs_pos=0
!SIS$ CR CR_P,CR_S,CR_MPISTARTUP
      call MPI_INIT( ierr )
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )
```

```
xperiment
  gsr405
    Process 0
      Thread 0
        (send = 7199611.0;)
          Region 1(CR_P[RELATIVISTIC_PLASMA_PROPAGATION])(send =
        (recv = 4.3855812E7;)
          Region 1(CR_P[RELATIVISTIC_PLASMA_PROPAGATION])(recv =
            Region 7(CR_S[MAIN])(recv = 4.3855812E7;)
              Region 80(CR_MPIRECV[MPI_RECV])(recv = 4677.0;)
              Region 68(CR_MPIRECV[MPI_RECV])(recv = 482562.0;)
              Region 56(CR_MPIRECV[MPI_RECV])(recv = 235661.0;)
              Region 48(CR_A[CAL_POWER])(recv = 7112753.0;)
              Region 78(CR_MPIRECV[MPI_RECV])(recv = 1553469.0;)
              Region 76(CR_MPIRECV[MPI_RECV])(recv = 5597099.0;)
              Region 66(CR_MPIRECV[MPI_RECV])(recv = 472087.0;)
              Region 64(CR_MPIRECV[MPI_RECV])(recv = 4091937.0;)
              Region 54(CR_MPIRECV[MPI_RECV])(recv = 331509.0;)
              Region 52(CR_MPIRECV[MPI_RECV])(recv = 6214276.0;)
```

```
        else
          Power0=Power
          Power=0.0
          do i26=1,numprocs-1
            call MPI_RECV(Power(1,1),3*nop,
     &            MPI_DOUBLE_PRECISION,
     &            i26,tag,MPI_COMM_WORLD,statu

            Power0=Power0+Power
          end do

          tt=it*dt

          write (7,'(15g17.7)') tt*1d15,(Power0(1,i2
          write (8,'(15g17.7)') tt*1d15,(Power0(2,i2
          write (9,'(15g17.7)') tt*1d15,(Power0(3,i2

        end if
      end if
!SIS$ END CR
!-----------------------------------------------------------
! Calculation of the power: end
```

Fig. 4. Overhead-To-Region and Region-To-Overhead analysis

An **Execution Summary Analysis** has been employed to examine the impact of communication on the execution time of the entire program. Figure 5 and 6 depict the *execution time summary* for the experiment executed with 3 SMP nodes and with 1 SMP node (4 processors per node), respectively. In the top window of Fig. 5 each pie represents one SMP node and each pie slice value corresponds to the average value across all processes of an SMP node (min/max/average values can be selected). By clicking onto an SMP pie, SCALEA displays a detailed summary for all processes in this node in the three lower windows of Fig. 5. Each pie is broken down into time spent in MPI routines and the remainder. Clearly, SCALEA indicates the dramatic increase of communication time when increasing number of SMP nodes. The MPI portion of the overall execution time corresponds approximately to 8.5% for the single SMP node version which raises to approximately 46% for 3 SMP-nodes. The experiments are based on a smaller problem size. Therefore, the communication to execution time ratio is rather high.

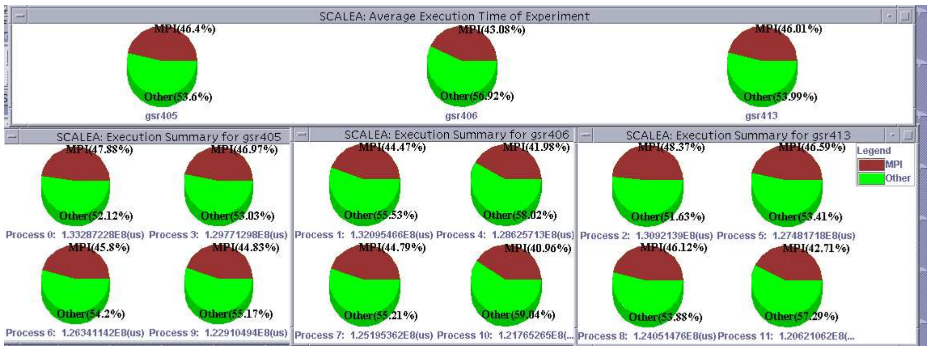

SCALEA: Average Execution Time of Experiment
MPI(46.4%)
Other(53.6%)
gsr405
MPI(43.08%)
Other(56.92%)
gsr406
MPI(46.01%)
Other(53.99%)
gsr413
SCALEA: Execution Summary for gsr405
MPI(47.88%)
Other(52.12%)
Process 0: 1.33287228E8(us)
MPI(46.97%)
Other(53.03%)
Process 3: 1.29771298E8(us)
MPI(45.8%)
Other(54.2%)
Process 6: 1.26341142E8(us)
MPI(44.83%)
Other(55.17%)
Process 9: 1.22910494E8(us)
SCALEA: Execution Summary for gsr406
MPI(44.47%)
Other(55.53%)
Process 1: 1.32095466E8(us)
MPI(41.98%)
Other(58.02%)
Process 4: 1.28625713E8(us)
MPI(44.79%)
Other(55.21%)
Process 7: 1.25195362E8(us)
MPI(40.96%)
Other(59.04%)
Process 10: 1.21765265E8(...
SCALEA: Execution Summary for gsr413
MPI(48.37%)
Other(51.63%)
Process 2: 1.3092139E8(us)
MPI(46.59%)
Other(53.41%)
Process 5: 1.27481718E8(us)
MPI(46.12%)
Other(53.88%)
Process 8: 1.24051476E8(us)
MPI(42.71%)
Other(57.29%)
Process 11: 1.20621062E8(...
Legend
MPI
Other

Fig. 5. Execution time summary for an experiment with 3 SMP-nodes

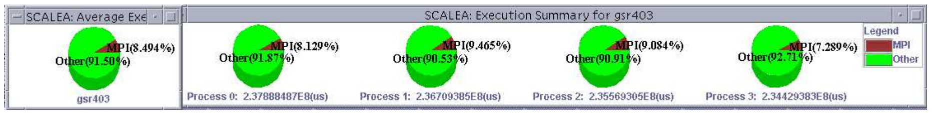


Fig. 6. Execution time summary for an experiment with 1 SMP-node

Multiple Experiment Analysis Mode: SCALEA provides various options to support multiple experiment performance analyses which can be applied to single or multiple region(s) ranging for single statements to the entire program.

Figure 7 visualizes the execution time and speedup/improvement of the entire 3DPIC application, respectively. With increasing machine sizes, also the system time raises (close to the user time for 25 CPUs) due to extensive communication.

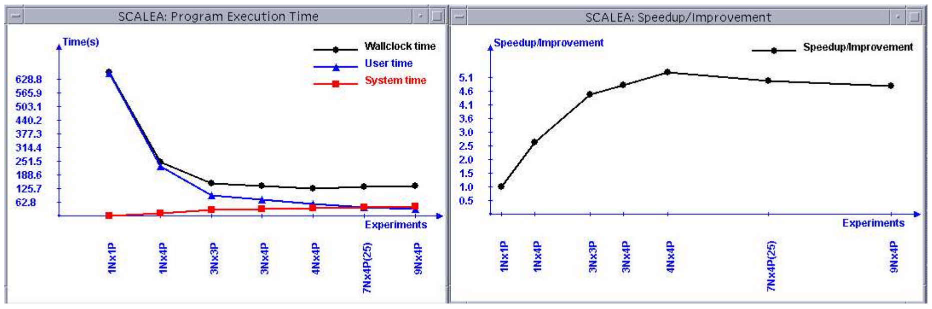
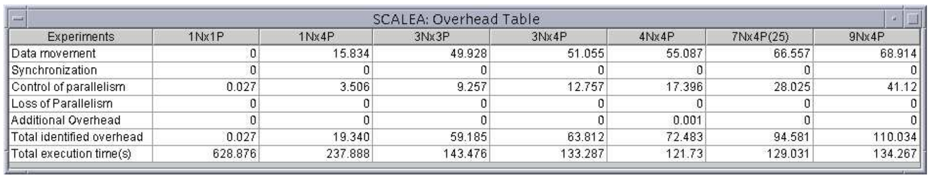


Fig. 7. Overall execution time and speedup/improvement for 3DPIC. *1Nx4P* means 1 SMP node with 4 processors (in case of 7Nx4P only 25 processors are used)

Overall, 3DPIC doesn't scale well for the problem size considered because of the poor communication behavior (see Fig. 8) and also due to control of parallelism (for instance, initialization and finalization MPI operations).



SCALEA: Overhead Table

Experiments	1Nx1P	1Nx4P	3Nx3P	3Nx4P	4Nx4P	7Nx4P(25)	9Nx4P
Data movement	0	15.834	49.928	51.055	55.087	66.557	68.914
Synchronization	0	0	0	0	0	0	0
Control of parallelism	0.027	3.506	9.257	12.757	17.396	28.025	41.12
Loss of Parallelism	0	0	0	0	0	0	0
Additional Overhead	0	0	0	0	0.001	0	0
Total identified overhead	0.027	19.340	59.185	63.812	72.483	94.581	110.034
Total execution time(s)	628.876	237.888	143.476	133.287	121.73	129.031	134.267

Fig. 8. Performance overheads of the 3DPIC application. Note that total and unidentified overhead are missing as no sequential code version is available for the 3DPIC.

4.2 LAPW0

LAPW0 [1] is a material science program that calculates the effective potential of the Kohn-Sham eigen-value problem. LAPW0 has been implemented as a Fortran90 MPI code.

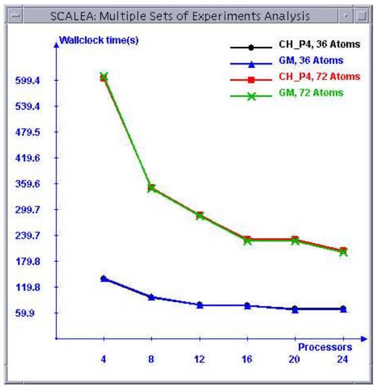


Fig. 9. Execution time of LAPW0 with 36 and 72 atoms. *CH_P4*, *GM* means that MPICH has been used for CH_P4 (for Fast-Ethernet 100Mbps) and Myrinet, respectively.

In [11] we described some results based on a single problem size and a fixed communication library and target machine network. In this section, we outline some additional performance analyses for different machine and problem sizes (36 and 72 atoms) with two different networks. The overall execution times provided by SCALEA's **Multi-Set Experiment Analysis** are shown in Fig. 9. The first observation is that changing the communication library and communication network from MPICH CH_P4 to MPICH GM does not lead to a performance improvement of the parallel LAPWO version. The second observation is that we cannot achieve better performance by varying the number of processors from 12 to 16 and 20 to 24 processors for the experiments with 32 atoms, or by varying the processor number from 20 to 24 processors for the 72 atom experiment. In order to explain this behavior we concentrate on the experiment with 36 atoms. Figure 10 visualizes the execution times for the most computational intensive code regions of LAPW0 supported by **Multiple Region Analysis**. The execution times of these code regions remain almost constant although the number of processors is increased from 12 to 24 and from 20 to 24. The LAPW0 code exposes a load balancing problem for 16, 20, and 24 processors. Moreover, code regions FFT_REAN0, FFT_REAN3, and FFT_REAN4 are executed sequentially as shown in Fig. 10. These code regions cause a large of loss of parallelism overhead (see Fig. 11).

In summary, LAPW0 scales poorly due to load imbalances and a large loss of parallelism overhead caused by the lack of parallelizing the FFT_REAN0, FFT_REAN3, and FFT_REAN4 code regions. In order to improve the performance of this application, these code regions must be parallelized as well.

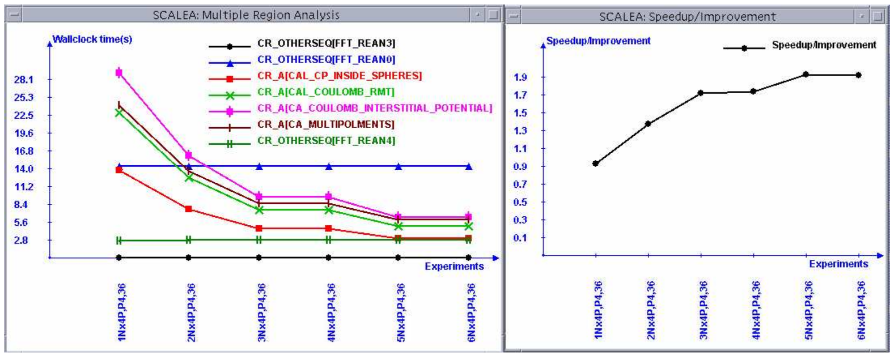
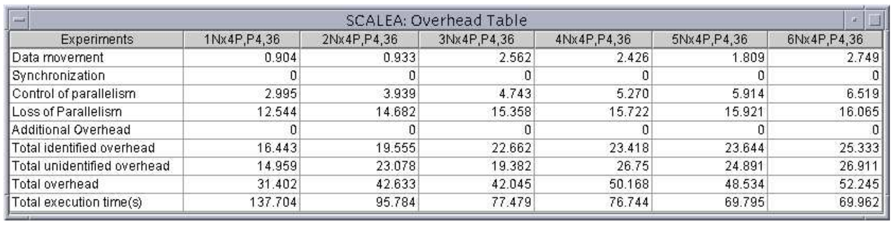


Fig. 10. Execution time of computationally intensive code regions. *1Nx4P,P4,36* means 1 SMP node with 4 processors using MPICH CH_P4 and the problem size is 36 atoms.


SCALEA: Overhead Table

Experiments	1Nx4P,P4,36	2Nx4P,P4,36	3Nx4P,P4,36	4Nx4P,P4,36	5Nx4P,P4,36	6Nx4P,P4,36
Data movement	0.904	0.933	2.562	2.426	1.809	2.749
Synchronization	0	0	0	0	0	0
Control of parallelism	2.995	3.939	4.743	5.270	5.914	6.519
Loss of Parallelism	12.544	14.682	15.358	15.722	15.921	16.065
Additional Overhead	0	0	0	0	0	0
Total identified overhead	16.443	19.555	22.662	23.418	23.644	25.333
Total unidentified overhead	14.959	23.078	19.382	26.75	24.891	26.911
Total overhead	31.402	42.633	42.045	50.168	48.534	52.245
Total execution time(s)	137.704	95.784	77.479	76.744	69.795	69.962

Fig. 11. Performance overheads for LAPW0.

5 Conclusions and Future Work

In this paper, we briefly gave an overview of SCALEA, and then described two different experiments with a laser physics and a material science code to examine the usefulness of our performance analysis approach.

Based on the flexible instrumentation mechanism and the code region performance analysis, SCALEA could determine the performance overheads that imply performance problems and relate them back to the code regions that cause them. SCALEA stores the collected performance data for all experiments in a data repository based on which a single- or multi-experiment analysis is conducted. This feature enables SCALEA to examine the scaling behavior of parallel and distributed programs. A rich set of performance diagrams is supported to filter performance data and to restrict performance analysis to relevant metrics and code regions.

SCALEA is part of the ASKALON performance tool set for cluster and Grid architectures [4] which comprises various other tools including an automatic bottleneck analysis, performance experiment and parameter study, and performance prediction tool. We are currently extending SCALEA for online monitoring of Grid applications and infrastructures.

References

1. P. Blaha, K. Schwarz, and J. Luitz. WIEN97, Full-potential, linearized augmented plane wave package for calculating crystal properties. Institute of Technical Electrochemistry, Vienna University of Technology, Vienna, Austria, ISBN 3-9501031-0-4, 1999.
2. S. Browne, J. Dongarra, N. Garner, K. London, and P. Mucci. A scalable cross-platform infrastructure for application performance tuning using hardware counters. In *Proceeding SC'2000*, November 2000.
3. F. Wolf and B. Mohr. Automatic Performance Analysis of SMP Cluster Applications. Technical Report Tech. Rep. IB 2001-05, Research Center Juelich, 2001.
4. T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Seragiotto, and H.-L. Truong. ASKALON - A Programming Environment and Tool Set for Cluster and Grid Computing. www.par.univie.ac.at/project/askalon, Institute for Software Science, University of Vienna.
5. M. Geissler. *Interaction of High Intensity Ultrashort Laser Pulses with Plasmas.* PhD thesis, Vienna University of Technology, 2001.
6. Allen Malony and Sameer Shende. Performance technology for complex parallel and distributed systems. In *In G. Kotsis and P. Kacsuk (Eds.), Third International Austrian/Hungarian Workshop on Distributed and Parallel Systems (DAPSYS 2000)*, pages 37–46. Kluwer Academic Publishers, Sept. 2000.
7. B. Miller, M. Callaghan, J. Cargille, J. Hollingsworth, R. Irvin, K. Karavanic, K. Kunchithapadam, and T. Newhall. The paradyn parallel performance measurement tool. *IEEE Computer*, 28(11):37–46, November 1995.
8. W. E. Nagel, A. Arnold, M. Weber, H.-C. Hoppe, and K. Solchenbach. VAMPIR: Visualization and analysis of MPI resources. *Supercomputer*, 12(1):69–80, January 1996.
9. D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. W. Schwartz, and L. F. Tavera. Scalable Performance Analysis: The Pablo Performance Analysis Environment. In *Proc. Scalable Parallel Libraries Conf.*, pages 104–113. IEEE Computer Society, 1993.
10. Hong-Linh Truong and Thomas Fahringer. SCALEA: A Performance Analysis Tool for Distributed and Parallel Program. In *8th International Europar Conference(EuroPar 2002)*, Lecture Notes in Computer Science, Paderborn, Germany, August 2002. Springer-Verlag.
11. Hong-Linh Truong, Thomas Fahringer, Georg Madsen, Allen D. Malony, Hans Moritsch, and Sameer Shende. On Using SCALEA for Performance Analysis of Distributed and Parallel Programs. In *Proceeding of the 9th IEEE/ACM High-Performance Networking and Computing Conference (SC'2001)*, Denver, USA, November 2001.
12. T. Cortes V. Pillet, J. Labarta and S. Girona. Paraver: A tool to visualize and analyze parallel code. In *WoTUG-18*, pages 17–31, Manchester, April 1995.