Performance evaluation of HPCN applications

Alessandro Merlo
Dipartimento di Informatica e Sistemistica – University of Pavia
Via Ferrata 1, I-27100 PAVIA (Italy)
E-mail: merlo@gilda.unipv.it

ABSTRACT

The performance attained by parallel programs executed on multiprocessor systems is largely influenced both by the characteristics of the code and by those of the system architecture. Indeed, parallel machines are typically message–passing systems consisting of hundreds of processing elements. Implementing high performance parallel applications is a difficult task and understanding sources of poor efficiency represents a key factor in every evaluation process. Performance analysis studies can be carried out by means of an integrated use of statistical and numerical techniques together with visualization methods. In this paper, the potentialities of this approach are presented by means of a case study dealing with a real kernel code used in weather forecasts.

1 Introduction

While designing a parallel application, programmers have to identify a parallelization strategy. In many cases, depending on the programming language adopted, on the characteristics of the numerical algorithms to be exploited, and on the topology of the target architecture, several possibilities arise for distributing data and tasks onto the processors.
Effective *a priori* predictions of the performance attained by means of different strategies are very difficult. Indeed, only accurate description of the code and of the communication and computational behavior of the multiprocessor system may help in identifying the real performance trade–offs. Usually, such an approach is based upon the combined used of a graphical representation of the application properties and of a description of the systems by means of Markov models or Petri nets (see, for example, [4] [14] [5]).
A posteriori evaluations are based on measurements collected at run–time on the real system. These studies result in accurate performance figures and can be used to diagnose efficiency losses. This approach relays on monitoring tools to dynamically gather timing data that, once stored into trace files, can then be analyzed to yield a characterization of programs behavior (see, for example, [12] [7] [9]).

This paper focuses on one of the major issue in evaluating, on a posteriori basis, the performance of parallel applications, that is trace file analysis. It reveals particularly useful in that it addresses the correspondences between the structure of the application and the characteristics of the target architecture. In the following, we present this approach and show how statistical

and numerical techniques can be applied to monitored data, together with visualization facilities, to obtain meaningful representations of parallel programs behavior.

In this paper, we analyze a kernel from the Integrated Forecasting Model (IFS) grid point transposition method [13]. This application, developed by the European Center for Medium–Range Weather Forecasts in Reading, implements a global spectral method and employs a space grid of more than 4 million points. The kernel represents the core transformation phase of the IFS code and has been implemented by means of the Message Passing Interface (MPI) communication library [17]. MPI is an attempt to standardize some message–passing routines that may be used in a large number of applications and that can be efficiently implemented on a wide range of parallel systems. Programs written with MPI can be easily ported to different architectures thus reducing the reimplementation efforts. Furthermore, MPI supports a detailed profiling interface which is able, at negligible overhead, to trace program executions and to collect information to be successively analyzed by means of performance tools. In the case study presented in this paper, traces obtained by means of this profiling interface on a Meiko CS–2 system have been analyzed by means of MEDEA (MEasurements Description, Evaluation and Analysis), a software tool which integrates some state–of–the–art techniques towards performance evaluation of parallel applications [8].

The paper is organized as follows. Section 2 briefly presents the monitoring–based approach towards performance evaluation. A description of the code used for our example study, and of the parallel system it has been run onto, is given in Section 3. An overview of how the statistical and numerical techniques implemented within MEDEA, coupled with its visualization facilities, can be applied to address different performance issues is given in Section 4. Finally, a few conclusions are drawn in Section 5.

2 The monitoring–based approach

Parallel machines are typically message–passing systems consisting of hundreds of processing elements. Implementing parallel application and attaining good performance is a difficult task, and understanding sources of poor efficiency represents a key factor in every evaluation study. As a result, techniques and tools that allow fast and accurate evaluations of different parallelization strategies and of the impact of different system architectures may significantly improve the resulting performance.

The real workload submitted to parallel systems can be viewed as the result of programmers' choices (that is, ways to parallelize existing software, use of particular numerical algorithms and programming paradigms, portability issues) and of the optimizations or modifications introduced when the source code is compiled on the target architecture (see Figure 1). The monitoring–based approach relays on facilities and tools to gather timing data at run–time. Interprocessor communications, synchronizations, and I/O operations are a few examples of "events" of interest for the measurement activity. They determine, along with the peak CPU speed, the resulting performance. In general, the amount of measurements is overhalming and needs to be refined in order to synthesize the information and extract easy–to–interpret tables

or diagrams (see, for example, [15] [9] [16]).

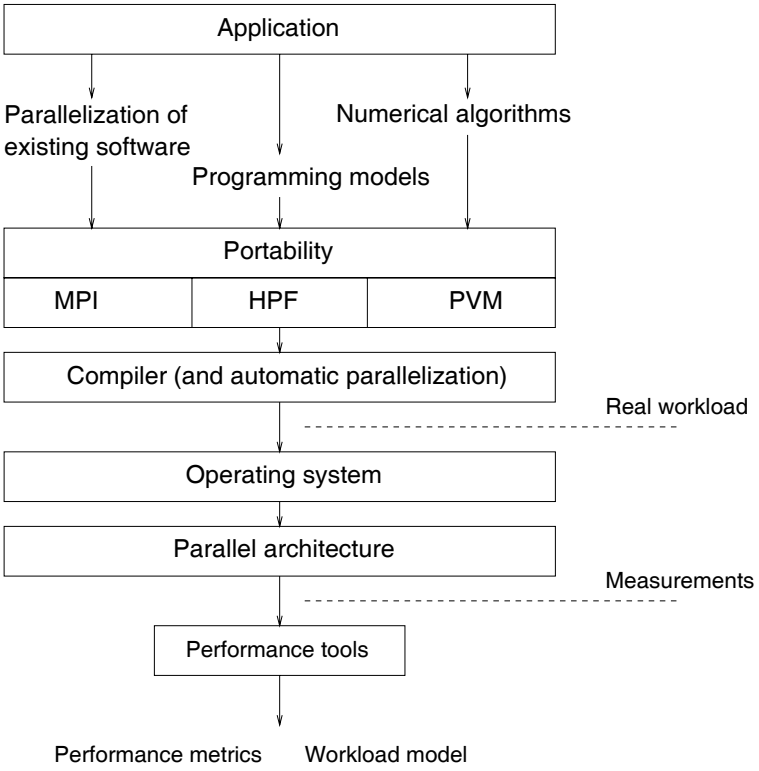

Figure 1: The measurement–based approach towards performance analysis of parallel applications.

This task is achieved by performance tools like MEDEA, which provides all the facilities to process the trace files and obtain high–level metrics and parameters to be used directly for performance study or for further analyses, respectively. The main characteristics of this tool are the possibility of interpreting various trace file formats (for example, those provided by PICL [10] and MPI), a Motif–based graphical interface and visualization aids, and a module, which implements a clustering algorithm [3], to drive the characterization of the application being studied. In particular, the analysis modules MEDEA is built upon allow the construction of synthetic workload models and ease the interpretation of the obtained results by means of visualization techniques.

3 Application and system descriptions

In what follows, brief descriptions of both the analyzed kernel and of the architecture of the system onto which measurements have been collected are given.

3.1 Description of the code

A kernel from the Integrated Forecasting Model (IFS) grid point transposition method has been analyzed. Within this type of methods, fields are transformed, at each timestep, between the physical domain, where forces are computed, and the spectral domain, where the horizontal terms of the involved differential equations are evaluated. The fields variables are represented as truncated expansions in terms of spherical harmonic functions. This results in Fourier transforms to be computed along latitudes and in Legendre transforms to be performed along longitudes.
The parallel message–passing version of the kernel is such that in any phase of the algorithm there is just only one direction in which the computations are non–locally coupled, even if this direction changes from phase to phase. These global dependencies, inherent in spectral models (see, for example, [1] [11]), create the need for global communications. As a results, each timestep can be summarized in the following stages. The inverse Legendre transforms of the spectral coefficients are locally computed by each processor. A transposition of the transformed fields takes place, involving global send/receive operations, in order to allow the Fourier transforms to be computed in parallel. Then, a new global transposition is needed before the direct Legendre transforms can be independently evaluated on each processor. The remaining computations in the spectral space can all be carried out locally. Additional communications are involved by the semi–Lagrangian advection in the timestepping scheme. This phase greatly improves the overall performance since longer timesteps can be used. However, this produces data dependencies that, although local in nature, change dinamically since the necessary data items come from different locations, which vary from grid point to grid point and from timestep to timestep.

3.2 Architecture of the system

The system used in our study is the Meiko CS–2 located at the European Center for Parallel Computing in Vienna. This system consists of 128 SuperSPARC scalar processors, without I/O capabilities. Each processor has 1 MByte cache and 64 MByte RAM. I/O operations are managed by 8 reserved SuperSPARC processing elements, each with 1 MByte cache, 128 MByte RAM and 2 GByte disk space.
The communication network does not use the processing elements as part of the network itself. This ensures that they are not affected by traffic generated by other applications running on different system partitions. Indeed, each processing element in the CS–2 system has a dedicated interface to the underlying communication network. This is a multi–stage packet network (fat tree), based on full 8x8 crosspoint switch, in which the bandwidth between two stages remains constant. The number of stages is, then, logarithmic in the number of processors. There is a wide range of interfaces to the CS-2 communications network, enabling codes which have previously been running on other machines to be ported easily. Each interface has a slightly different functionality and associated performance. Among the currently supported interfaces are PARMACS [6], PVM [2], and MPI.

4 Performance analysis: a case–study

Depending on the objectives of the study, each trace file, corresponding to the execution of the kernel with a given number of processors, can basically be analyzed from a *program–level* or a *processors–level* viewpoint. The former gives information such as the global execution, computation and communication times, and, when more trace files are given as input to MEDEA at a time, provides performance metrics such as speedup and signatures [8]. With the processors–level viewpoint more details on program behavior can be derived since a characterization on a per processor basis is applied by MEDEA. In what follows, both these types of analysis are presented. Furthermore, an example on how low–level instrumentation of the source code can sometimes reveal a valuable approach towards performance evaluation is also presented.

4.1 Program–level analysis

This level reveals very useful when scalability issues (with respect to the problem size or to the number of allocated processors) and comparison studies (between different architectures) are addressed.
In our case study, traces corresponding to executions of the kernel with a number of processors varying from 4 up to 64 have been analyzed by means of MEDEA. The 2–processor run has not been considered because, due to the characteristic of the code, no communications are performed when the grid point space is split into two partitions. Figure 2 summarizes the overall performance attained by the IFS kernel.

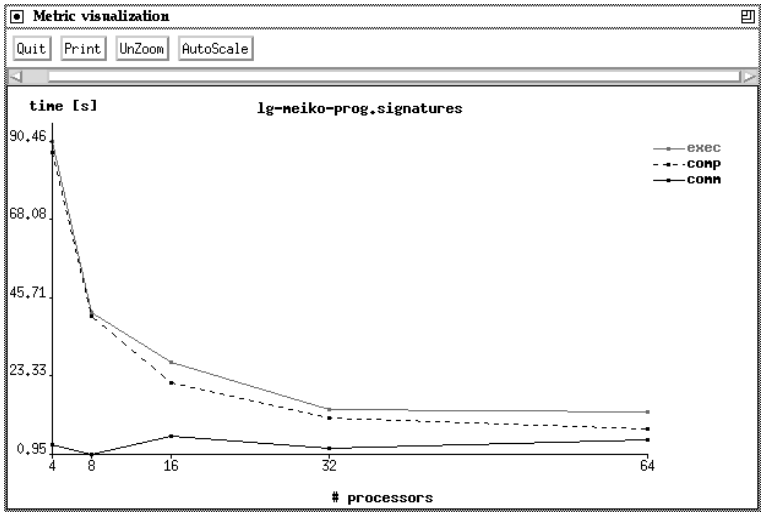

Figure 2: Execution, computation, and communication signatures for the analyzed code.

These curves express the total execution, computation and communication times of the application as a function of the allocated processors. Usually, the communication time is an increasing function of the number of processors, while the computation time has a monotonically decreasing behavior. The figure shows, for example, that the execution time does not decrease very much if we allocate more than 32 processing elements. Indeed, the total execution time

reduces from 13.7328 to 13.0220 seconds when we run the kernel with 32 and 64 processors, respectively. The performance losses, in this case, have been identified in the communication part of the code. Indeed, while with 32 processors the kernel spends 18% of the execution time performing send/receive operations, with 64 processors 4.8062 seconds are used for communications, accounting for 37% of the total execution time.
More insights on the behavior of these runs can be obtained by looking at the communication profiles derived by MEDEA (see Fig. 3). These curves represent the number of processors involved in communication operations as a function of the execution time of the kernel.

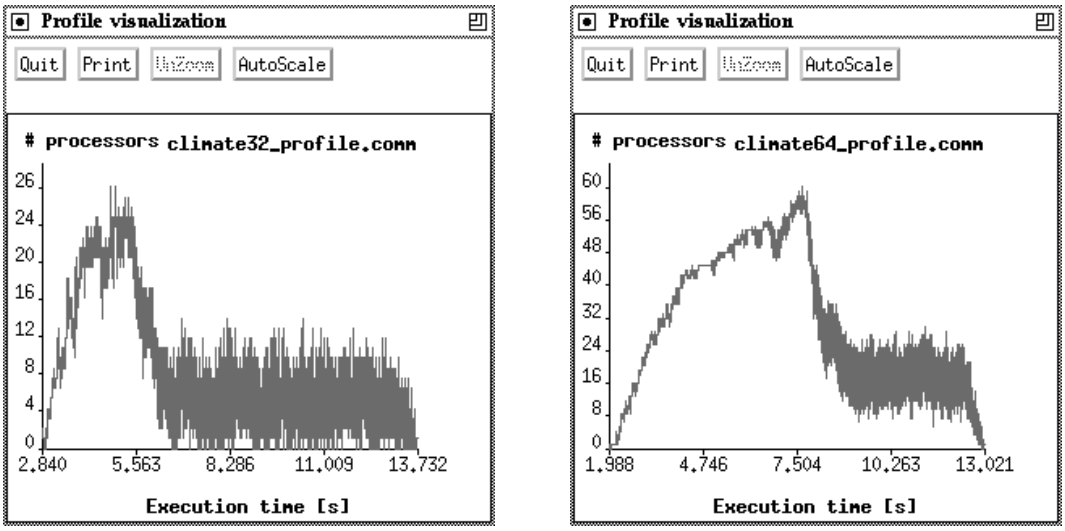

Figure 3: Communication profiles for the IFS kernel with 32 (left) and 64 processors (right).

As it can be seen, these profiles are characterized by a first phase, corresponding to the setup procedures of the kernel, where the processors distribute data among themselves before reaching a more regular communication activity, where the successive iterations in time of the weather model are performed. The setup phase lasts about 6.5 seconds in the 32–processors run (47% of the execution time) while 68% of the time is needed by the 64–processors run to complete it. As a matter of fact, when the number of processors increase the corresponding communication profile becomes more and more irregular. In order to better understand these results, more detailed analyses were required.

4.2 Processors–level analysis

When a trace file is analyzed at this level, attention is focused on the behavior of each individual processing element. To obtain a comparison between the 32–processor and the 64–processor runs of the kernel, we have used MEDEA to extract timings (e.g., execution, computation and communication times) and volume information (e.g., number of sent/received messages) on a per processor basis. Table 1 summarizes the analyses performed at this level.

As it can be seen, while in the 32–processor execution only two processing elements have a communication time that diverges from the mean value for more than 50%, the 64–processor run

	Communication time [sec]			Deviation	
	Mean	Max	Min	25%	50%
32 processors	2.4719	3.5201	0.7290	15	2
64 processors	4.8062	7.2694	0.2844	44	15

Table 1: Processor statistics for the IFS kernel.

is characterized by higher fluctuations, even if the number of communications per processor to be performed remains the same. Indeed, 15 processors out of 64 are characterized by a communication time less than 2.4031 seconds or greater than 7.2093 seconds. This analysis, together with the one performed at the program–level, reveals that the processors are not well synchronized. Indeed, this version of the kernel does not include an MPI_Barrier() call at the beginning and the end of the setup phase. This synchronization function should reduce the delays between processors, thus improving the overall performance. Currently, a modified version of the kernel is being run on the Meiko CS–2 system to address this issue.
At this level, also the times taken by each processors to execute each iteration of the weather forecast model have been compared. As a result, it has been noticed that, on the Meiko system, the transposition of local data reveals a time consuming task. In order to obtain more accurate performance figures low–level instrumentation of the kernel was necessary.

4.3 Low–level instrumentation

Figure 4 shows the piece of code where the transformation of interest takes place. This routine contains six nested loops. The MPI library allows the user to perform low–level instrumentation by means of calls to a specific extension function which identifies the start and end points of the event of interest. In our case, our objective was to collect information on the execution times of each of these loops.
However, such a detailed instrumentation resulted in huge amounts of data to be collected into trace files. Since the profiling mechanism implemented within MPI tends to be less intrusive as possible, all the run–time information is gathered into a memory buffer which is written down to physical memory only at the end of the execution. Due to main memory constraints, on the Meiko CS–2 system we were only able to gather execution times for the three loops highlighted in the figure, that is `Loop1`, `Loop2`, and `Loop3`.
In particular, our attention was focused on the first two loops, in order to investigate the overhead involved in loop management activity. Indeed, these loops are perfectly nested, in the sense that the outermost loop does nothing more than incrementing the loop index, while `Loop2` contains all the computation activity of the transposition routine. The times taken by these loops to complete for the 32–processor run of the kernel are shown in Figure 5.
Note that the execution times for `Loop1` are at least twice those measured for `Loop2`. This means that more than half of the total execution time of this routine is spent in control operations. This behavior has been observed also for executions with 16 and 32 processors.

```
c
c     Transpose data used on local processor
c     --------------------------------------

      z0=timef()
      IFLDL=0
      DO JVAR=1,KVAR(MYSETB)                                  Loop1
        DO JFLD=KFIELD(1,JVAR,MYSETB),KFIELD(2,JVAR,MYSETB)   Loop2

          ILPOS=IOFF
          IFLDL=IFLDL+1

          DO JGL=1,NDGLL

            DO JL=NSTA(NPTRFLOFF+JGL+IA,MYSETB)+NSTABUF(JGL),
     &           NSTA(NPTRFLOFF+JGL+IA,MYSETB)+NSTABUF(JGL)+
     &           NONL(NPTRFLOFF+JGL+IA,MYSETB)-1
              ILPOS=ILPOS+1
              ZBUF(ILPOS)=PGLAT(JL,IFLDL)
            ENDDO

          ENDDO
          DO JK=1,NGPTOT-JOFF,NPROMA                          Loop3

            IBLOCK=(JK-1)/NPROMA+1
            IKBEG = MAX(IOFF+1,JK)
            IKEND=MIN(NGPTOT-JOFF,JK+NPROMA-1)
            ICOUNT=0
            IF (IOFF+1 .GT. JK) ICOUNT = IOFF + 1 - JK

            DO JK1=IKBEG,IKEND
              ICOUNT=ICOUNT+1
              PGCOL(ICOUNT,JFLD,IBLOCK)=ZBUF(JK1)
            ENDDO
          ENDDO
        ENDDO
      ENDDO
```

Figure 4: Transposition routine on local data.

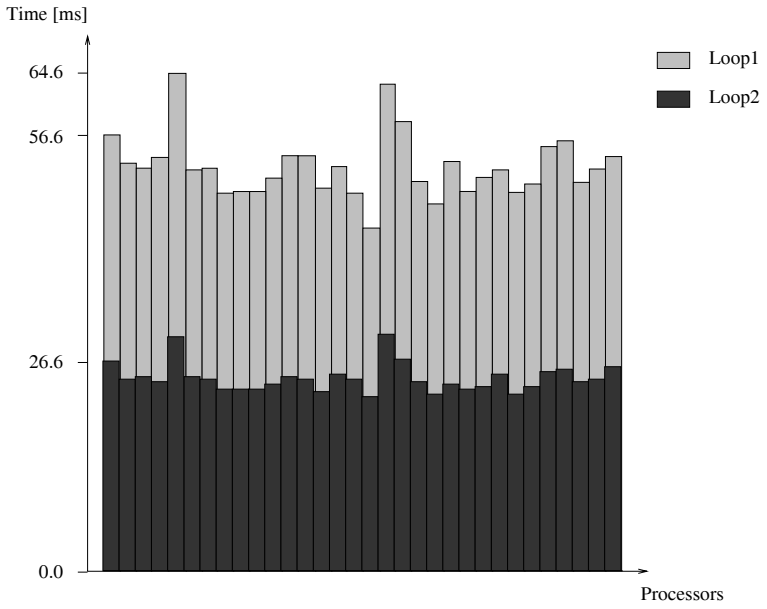

Figure 5: Times taken by `Loop1` and `Loop2` for a 32–processor execution.

5 Conclusions

In this paper the monitoring–based approach towards performance evaluation of parallel applications has been presented. In particular, this approach have been used to analyze a real kernel from the Integrated Forecasting Model, the European Center for Medium–Range Weather Forecasts production code. Instrumentation and measurements have been realized by means of the MPI library and the resulting trace files have been analyzed by MEDEA, a software tool which integrates a few state–of–the–art techniques for performance studies. In particular, it has been shown the effectiveness of this methodology and how different analysis levels are required in order to derive a complete description of the application behavior.

Acknowledgments

The research was supported in part by the ESPRIT IV Long Term Research Project No. 21033 HPF+ "Optimizing HPF for Advanced Applications". Many tanks to the European Center for Parallel Computing at Vienna for the use of the Meiko CS–2 system.

References

[1] B. Machenhauer. The Spectral Method. In *Numerical Methods Used in Atmospheric Models*, volume II of GARP Pub. Ser. No. 17, pages 121–275. JOC, World Metereological Organization, Geneva, 1979.

[2] G. A. Geist, A. L. Beguelin, J. J. Dongarra, W. Jiang, R. J. Manchek and V. S. Sunderam. *PVM: Parallel Virtual Machine – A Users Guide and Tutorial for Network Parallel Computing*. MIT Press, Boston, 1994.

[3] J. A. Hartigan. *Clustering Algorithms*. J. Wiley, 1975.

[4] J. Brehm, L. Dowdy, M. Madhukar and E. Smirni. PerPreT – A Performance Prediction Tool. In *Quantitative Evaluation of Computing and Communication Systems*, volume 977 of *Lecture Notes in Computer Science*. Springer–Verlag, 1995.

[5] J. Brehm, P. H. Worley and M. Madhukar. Performance Modeling for SPMD Message–passing Programs. Technical Report ORNL/TM–13254, Oak Ridge National Laboratory, Oak Ridge, TN, 1996.

[6] L. Bomans, R. Hempel and D. Roose. The Argonne/GMD Macros in Fortran for portable parallel programming and their implementation on the Intel iPSC/2. *Parallel Computing*, 15:119–132, 1990.

[7] M. Calzarossa and G. Serazzi. Workload Characterization: A Survey. *Proceedings of the IEEE*, 81:1136–1150, 1993.

[8] M. Calzarossa, L. Massari, A. Merlo, M. Pantano and D. Tessera. MEDEA – A Tool for Workload Characterization of Parallel Systems. *IEEE Parallel and Distributed Technology*, 3(4):72–80, 1995.

[9] M. T. Heath, A. D. Malony and D. T. Rover. Parallel Performance Visualization: From Practice to Theory. *IEEE Parallel and Distributed Technology*, 3(4):44–60, 1995.

[10] P. H. Worley. A New PICL Trace File Format. Technical Report TM-12125, Oak Ridge National Laboratory, Oak Ridge, TN, 1992.

[11] P. H. Worley and J. B. Drake. Parallelizing the Spectral Transform Method. *Concurrency: Practice and Experience*, 4:269–291, 1992.

[12] P. Heidelberger and S. S. Lavenberg. Computer Performance Evaluation Methodology. *IEEE Transactions on Computers*, 33(12):1195–1220, 1984.

[13] S. R. M. Barros and T. Kauranne. On the parallelization of global spectral weather models. *Parallel Computing*, 20:1335–1356, 1994.

[14] T. Fahringer. Estimating and Optimizing Performance for Parallel Programs. *IEEE Computer*, 28:47–56, 1995.

[15] T. LeBlanc, J. Mellor–Crummery and R. Fowler. Analyzing Parallel Program Executions Using Multiple Views. *IEEE Parallel and Distributed Computing*, 9(2):203–217, 1990.

[16] V. S. Adve, J. Mellor–Crummey, M. Anderson, K. Kennedy, J. Wang and D. A. Reed. *Integrating Compilation and Performance Analysis for Data-Parallel Programs.* IEEE Computer Society Press, 1996.

[17] W. Gropp, E. Lusk, N. Doss and T. Skjellum. A High–performance, Portable Implementation of the MPI Message–passing Interface Standard. Technical Report ANL/MCS–P567–0296, Argonne National Laboratory, Argonne, IL, 1996.