Integration of a compilation system and a performance tool: the HPF+ approach*

M. Calzarossa[1], L. Massari[1], A. Merlo[1], M. Pantano[2], D. Tessera[1]

[1] Dipartimento di Informatica e Sistemistica, Università di Pavia
Via Ferrata 1, I–27100 Pavia, Italia
{mcc,massari,merlo,tessera}@alice.unipv.it
[2] Institute for Software Technology and Parallel Systems, University of Vienna
Liechtensteinstr. 22, A–1090 Vienna, Austria
pantano@par.univie.ac.at

Abstract. The performance of HPF codes is influenced by the characteristics of the parallel system and by the efficiency of the compilation system. Performance analysis has to take into account all these aspects. We present the integration of a compilation system with a performance analysis tool aimed at the evaluation of HPF+ codes. The analysis is carried out at the source level. The "costs" of the parallelization strategies applied by the compiler are also captured such that a comprehensive view of the performance is provided.

1 Introduction

High Performance Fortran (HPF) is a high level language for parallel programming based on the SPMD paradigm. It provides a set of directives, which allow the programmer to express the potential parallelism of the code, without having to explicitly state the communications required to distribute the data among the allocated processors. It will be up to the compiler to take advantage of the HPF directives inserted into the source code and to generate a parallel code, to be executed on a parallel system. Hence, performance of HPF programs is a function not only of the the characteristics of the parallel system, but also of the HPF directives which influence the parallelization strategies adopted by the compiler. In this framework, the performance of HPF programs heavily relies on the efficiency of the compiler, since the programmer has only an indirect control over the generated parallel code. For example, it is important to estimate the "costs" of the parallelization strategies applied by the compiler, in that these costs represent a sort of overhead for the programs with the final effect to increase their overall execution time.

Performance tools have to address all these issues. Hence, a tight integration with the compilation systems is required to relate the achieved performance to the source code. Examples of such an integration are presented in [1], [6], [7],

* This work was supported in part by the European Commission under the Esprit IV Long Term Research Project n. 21033 "HPF+".

and [8]. In [1], the performance of Fortran D programs is analyzed at the source–level thanks to the integration of Fortran D compiler and the Pablo performance tool. The integration of TAU with the pC++ compiler and run–time system allows instrumentation, profiling, and tracing support [6]. The source–level evaluation of C, C++, Fortran and HPF codes is addressed by the MPP Apprentice tool [8] and by the Visualization Tool [7].

In this paper we present the integration of the VFC compilation system [3] and the Medea performance analysis tool [4]. This work is part of the Esprit LTR project "HPF+"[1], whose goal is to extend the current version of the HPF language and the related compiling technology to address the requirements of advanced application problems. HPF+ language includes all HPF–2 features, a few of HPF–2 approved extensions [5], and additional features for handling irregular codes.

The aim of our integration is to provide a comprehensive view of the performance of HPF+ programs whose outcomes can be used either by programmers to understand and to improve the performance of their codes and by compiler developers to optimize the efficiency of the compiler itself. Our integration relies on the instrumentation system (SIS) embedded into the VFC compilation system. Specific information about the various activities performed during the execution of the code is measured. Medea recognizes and interprets this information and provides a description of the achieved performance at the level of the HPF+ source code.

The paper is organized as follows. In Section 2, we present the main features of the VFC compilation system with particular emphasis to the SIS instrumentation system. Section 3 describes the possible levels of performance analysis as a consequence of the integration achieved between VFC and Medea. A few conclusions are drawn in Section 4.

2 The instrumentation system

VFC is a source–to–source compilation system from HPF+ codes to explicitly message passing codes. The parallelization strategies used in VFC are very general and applicable to programs with dynamically allocated or distributed arrays. VFC combines compile–time parallelization techniques with run–time analysis. For example, the parallelization of irregular independent loops is based on the inspector–executor paradigm [2] and can be described as a sequence of separate phases: work distribution, inspector, gather, executor and scatter.

In order to support performance analysis and tuning of HPF+ codes, VFC provides SIS, an embedded system for automatic program instrumentation.
SIS is able to instrument, via command–line compiler options, various code regions, such as, subroutines, independent loops, sequential loops, I/O operations and any arbitrary sequence of executable statements. Moreover, in order to instrument a statement (e.g., array assignment) or any arbitrary sequence of exe-

[1] More about the "HPF+" project at http://www.par.univie.ac.at/hpf+.

cutable statements, the programmer has to insert, into the HPF+ source code, SIS directives at the beginning and the end of the sequence to be measured.

The instrumentation system can be seen as composed of two main phases: the *code region selector* and the *parallel code instrumenter*. In the first phase, SIS analyzes the HPF+ code and collects compile–time information regarding each code region to be measured. The parallel code instrumenter instruments code regions while they are parallelized by VFC and also instruments the additional pieces of code inserted by the compiler into the generated code as a consequence of the applied parallelization strategies. In such a way, the overhead introduced by the compiler will also be measured at run–time. For example, in the case of independent loops, each individual phase (work distribution, inspector, gather, executor and scatter) is instrumented. In the case of parallelization of a subroutine call, which requires an implicit redistribution, SIS instruments the call itself and the pieces of code inserted by the compiler before and after the call.

The combined use of the code region selector and of the parallel code instrumenter provides the following information for each instrumented code region:

- type of instrumented region (subroutine, loop, independent loop, ...);
- name of the source file containing the code region;
- position in the HPF+ code in terms of line numbers;
- label identifying the instrumented region;
- measurement points inserted in the generated code and associated to the beginning and to the end of the instrumented code region.

Similar information is also provided for the pieces of code (e.g., work distributor, gather) inserted by the compiler into the generated code.
Note that SIS instruments all the communication activities associated to the individual regions and to the pieces of code and recognizes the different communication protocols.
The code instrumentation is performed by inserting in the generated code calls to run–time measurement libraries. At the end of execution of the generated parallel code, a tracefile containing timings for each measurement point and for the various communication activities is produced.
A one–to–one correspondence is then built between the positions in the source code and the measurement points inserted in the parallel code. This correspondence, together with the timings information contained into the tracefile, is used by Medea for a source–level performance analysis of HPF+ code, aimed at program tuning and compiler optimization. Each code region and its correspondent parallel version can be immediately identified and analyzed.

3 The performance analysis environment

As already pointed out, performance analysis environments have to be integrated with compilation systems in order to provide source–level analysis of HPF codes. In the HPF+ framework, Medea, integrated with VFC, is able to address these performance issues at various levels of detail. A comprehensive view of the

achieved performance is provided by applying a post–mortem analysis of the information collected at run–time. Such an analysis is based on statistical and visualization techniques.
Profiling of various HPF+ code regions, analysis of the overhead introduced by the compiler, analysis of the communication activities and of the behavior of individual processors are examples of the outcomes provided by Medea as a result of its integration with VFC. In order to be more effective, all these analyses correlate performance results with HPF+ source code.

Code profiling provides timings of the instrumented code regions, e.g., independent loops and subroutines. It also provides the number of occurrences, i.e., the number of times the code region has been executed. All these kinds of information are very useful for programmers in that they highlight the most time consuming code regions.
As an example, Figure 1 plots the execution times of various sequential loops, together with their number of occurrences. Labels associated to each bar refer to file name and line number of the HPF+ code containing the loop itself.

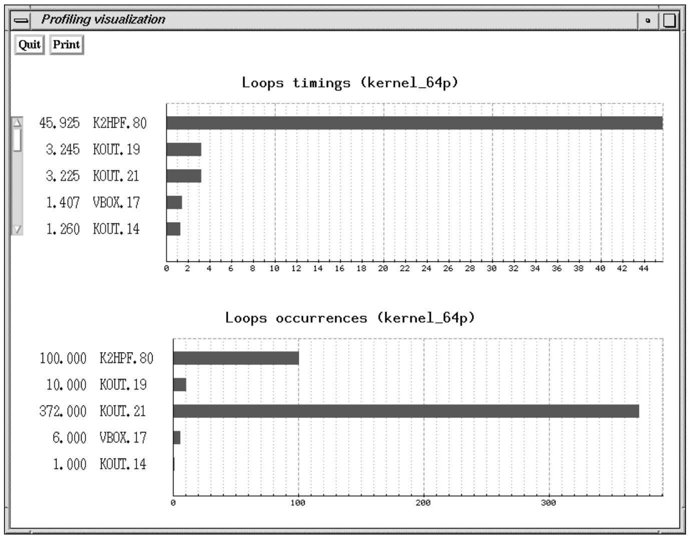

Fig. 1. Execution times, expressed in seconds, and number of occurrences of sequential loops.

The overall execution times shown in the figure account for the actual computation required by the code and for the various phases required by the parallelization strategies adopted by VFC. Hence, it is important to understand the contribution of these individual components in order to evaluate the overhead introduced by the compiler in terms of communication times and extra computation times. The evaluation of this overhead provides feedback both to programmers and to compiler developers on the efficiency of the code and of

the compiler. Large overheads may suggest to adopt different data distributions, or to optimize parallelization strategies. Note that the overhead analysis is well suited to quantify the costs of accessing distributed data within sequential loops and subroutines.
Figure 2 shows the breakdown of the execution time for a set of independent loops; the times spent in the five phases of the inspector–executor paradigm, namely, work distributor, inspector, gather, executor and scatter, are outlined. Let us remark that the actual computation of independent loops corresponds to the executor phase.
From all these diagrams, a link to the source code of the instrumented code regions can be obtained.

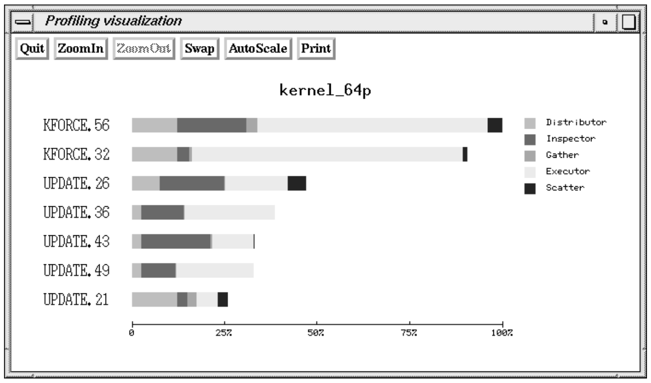

Fig. 2. Breakdown of the execution time for independent loops.

The analysis of the communication costs for each instrumented code region highlights the contributions of the various communication types, i.e., transmitting, receiving and collective exchanges. An example of communication time breakdown is shown in Fig. 3.

The communication analysis can be specialized on a per protocol basis, with the aim of evaluating the communication schedule. Inadequate communication patterns can lead to unbalanced timings among the allocated processors, as shown in Fig. 4. Such a situation may reflect unbalanced data/work distributions.

Dynamic behaviors of the communication patterns are investigated by means of communication profiles, which plot, as a function of the execution time, the number of processors involved in the various communication activities.
Metrics, such as, speedup and signatures, provide an overview of the degree of parallelism achieved by the program, by analyzing the overall execution time varying the number of allocated processors.

All the levels of analyses outlined above, e.g., profiling, overhead analysis, study of the communication patterns, provide deep insights into the program behavior from different perspectives. Hence, data analysis techniques are required

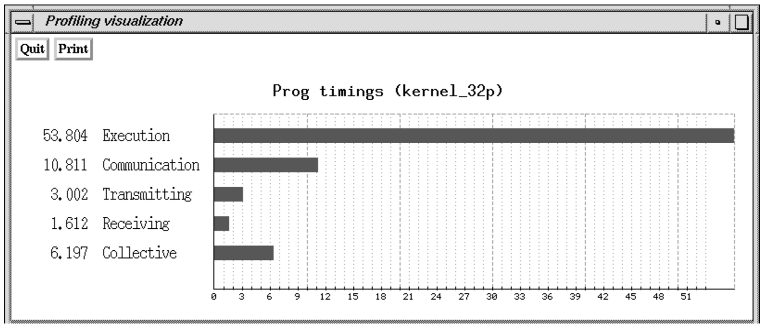

Fig. 3. Execution time and breakdown of the communication times, expressed in seconds.

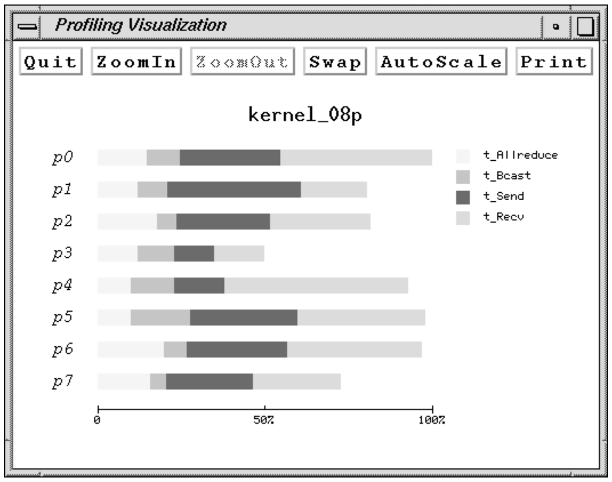

Fig. 4. Per protocol communication times for each allocated processor.

to summarize the large amount of collected information. Advanced statistical and numerical techniques, such as, clustering and fitting, are provided within Medea, and can be profitably used for the purpose.

4 Conclusions

In this paper we presented an environment for performance analysis of HPF+ codes developed in the framework of the Esprit HPF+ project. This work relies on a tight integration between the instrumentation system embedded in VFC and the post–mortem performance analysis tool Medea. This integration provides a source–level analysis of the performance of HPF+ codes which allows the evaluation of the efficiency of the compiler as well as of the each individual code region.

References

1. V.S. Adve, J. Mellor-Crummey, M. Anderson, K. Kennedy, J. Wang, and D. Reed. Integrating Compilation and Performance Analysis for Data–Parallel Programs. In M.L. Simmons, A.H. Hayes, J.S. Brown, and D.A. Reed, editors, *Workshop on Debugging and Performance Tuning for Parallel Computing Systems*, pages 25–51. IEEE Computer Society, January 1996.
2. S. Benkner, K. Sanjari, and V. Sipkova. Extension of VFCS for the compilation of project benchmarks Vers. 2. Document Esprit HPF+ Project D2.2b, September 1997.
3. S. Benkner, K. Sanjari, and V. Sipkova. Parallelizing Irregular Applications with the Vienna HPF/HPF+ Compiler. In *Proc. HPCN Europe 98*, Amsterdam, April 1998. Springer.
4. 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.
5. High Performance Fortran Forum. High Performance Fortran Language Specification - Version 2.0. Technical Report, Rice University, January 1997. http://www.crpc.rice.edu/HPFF/.
6. A.D. Malony, B.W. Mohr, P. Beckman, D. Gannon, S. Yang, and F. Bodin. Performance analysis of pC++: a portable data-parallel programming system for scalable parallel computers. In *Proc. 8th Int. Parallel Processing Symposium*, pages 75–84, Mexico, April 1994. IEEE Computer Society Press.
7. The Visualization Tool, http://ibm.tc.cornell.edu/ibm/pps/pe/.
8. W. Williams, T. Hoel, and D. Pase. The MPP Apprentice Performance Tool: Delivering the Performance of the Cray T3D. In K.M. Decker, editor, *Programming Environments for Massively Parallel Distributed Systems*, pages 333–345. Birkhauser Verlag, 1994.