Available online at www.sciencedirect.com
ScienceDirect
Future Generation Computer Systems 24 (2008) 121–132
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/fgcs

High-level application-specific performance analysis using the G-PM tool☆

Roland Wismüller[a,*], Marian Bubak[b], Włodzimierz Funika[c]

[a] *BSVS, University of Siegen, Germany*
[b] *Institute of Computer Science AGH-UST, Academic Computer Centre — CYFRONET, Kraków, Poland*
[c] *Institute of Computer Science AGH-UST, Kraków, Poland*

Received 26 September 2006; received in revised form 31 January 2007; accepted 26 March 2007
Available online 6 April 2007

Abstract

The paper presents an approach to overcome a traditional problem of parallel performance analysis tools: performance data often are too low level and cannot easily be mapped to the application's code structure, e.g. its execution phases. The G-PM tool offers the user an easy but flexible means to define her/his own high-level, application specific metrics based on existing metrics and application events. We discuss the basic concepts of G-PM from the user's point of view, its design, and some implementation issues, including the language PMSL which supports the specification of user-defined metrics. In the main part of the paper, we present a case study based on a real world medical application from the EU funded CrossGrid project, which demonstrates the concept of user-defined metrics as well as its usefulness in practice.

1. Introduction

Most of today's applications that require high computing performance are based on parallel programming using the message passing paradigm, as it is supported by MPI [17]. For this class of applications, tools that allow us to measure and improve their performance characteristics are vital for the applications' success. Generally, performance analysis tools can be based on three different techniques: tracing, profiling and online analysis. With tracing, performance analysis is done in two steps: while the application is executing, relevant *events* (such as the beginning and the end of a call to the `MPI_Send()` communication routine) and their time stamps are written to a file. In a subsequent offline step, different performance *metrics* (e.g. time spent in communication) can be computed from this trace file. Profiling avoids the necessity to store large trace files by computing a predefined set of metrics online, during the application's execution. These metrics typically are summaries over the whole execution. Online analysis can be viewed as a compromise between profiling and tracing, since – as with profiling – the tool computes performance metrics online, while on the other hand, – as with tracing – the information is still resolved in time. Different from both the other approaches, online analysis tools present the performance results *while* the application is executing and allow definition of new measurements based on these results.

Today, there is already a number of sophisticated performance tools supporting the analysis of parallel applications. In the report [14] the authors list 26 performance related tools just in the context of grid computing. However, even with these tools it is still difficult for programmers to optimize their applications based on the provided information. This has two major reasons:

- First, the information is often too low-level, since it is usually related to communication or even hardware events. For example, tools for MPI typically provide the time spent in `MPI_Barrier()` or `MPI_Recv()`, but they fail to provide information about load imbalance. This is because in general the way for measuring the metrics "load imbalance" is application specific. While in shared memory applications, load imbalance can usually be measured by comparing the waiting time at a barrier in the individual threads, message passing applications can also synchronize via messages. In

☆ Partially funded by the European Commission (project IST-2001-32243, CrossGrid) and KBN (grant 4 T11C 032 23).

* Corresponding address: Operating Systems and Distributed Systems BSVS, University of Siegen, Holderlinstr. 3, 57068 Siegen, Germany. Tel.: +49 271 740 4050; fax: +49 271 740 4049.
E-mail addresses: roland.wismueller@uni-siegen.de (R. Wismüller), bubak@agh.edu.pl (M. Bubak), funika@agh.edu.pl (W. Funika).

0167-739X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2007.03.008

such cases, the receive delay must be used as a basis for measuring the load imbalance (cf. Section 5.1).
- Second, linking the displayed performance data to the source code and the programmer's mental model of the application is rather difficult. The latter includes the structuring of the application's execution in well defined phases, e.g. the iterations of a numerical solver and the different phases within one iteration. As another example, consider a solver for linear equation systems based on LU-decomposition. Such a solver will consist of two phases: the decomposition phase and the solving phase. Performance information aggregated over all of these phases often is not sufficient to find specific bottlenecks in one phase. Tools based on tracing have some advantage here, since the information is resolved in time. However, it may be hard for the user to identify program phases from a time line diagram, which just shows program states and exchanged messages. In principle, the traced events could contain a link to the source code, but this would result in large amounts of data that need to be stored and analysed. In addition, even line number information such as `lu.f:231` does not easily allow identification of higher level program phases.

In our research, we address both difficulties by allowing the user to specify higher-level, application specific metrics at run-time. The online performance analysis tool G-PM, which originally has been designed for use in the grid, supports this via a Performance Metrics Specification Language, called PMSL. It allows, for example, definition of an application-specific metrics for load imbalance, as shown in Section 5.1. A distinguishing feature of PMSL is that the definition of metrics can also take into account events in the application. On the one hand, this allows to compute metrics from these events, e.g. in an iterative solver we can compute the convergence rate from an "iteration" event, which has the current residual error as its parameter. On the other hand, events can be used to mark and to identify specific program phases, which allows metrics related to these phases to be defined(cf. Section 5.3). These events are created by a small amount of user-defined source code instrumentation.

The main contribution of this paper is a case study presented in Section 5, which shows the usefulness of PMSL for the analysis of a real world application from the medical domain. In Section 2 we briefly discuss other approaches for higher-level performance analysis, while Section 3 outlines the main concepts of G-PM, especially its support for user-defined metrics and the PMSL language. An outline of G-PM's implementation is presented in Section 4.

2. Related work

The desire for tools supporting performance analysis at a higher level of abstraction is addressed by current research in several different ways. A rather ambitious approach is automatic performance analysis, which points the programmer to the exact cause and location of a bottleneck. Research in this field is done in, e.g. the Paradyn [22], KappaPi [8], EXPERT [32], APART [1], Aksum [12] and Periscope [15] projects. Automatic performance analysis is typically based on a (more or less) explicitly formalized description of performance properties and/or possible application bottlenecks. Based on this description, tools either perform an offline analysis of recorded trace files (KappaPi, EXPERT) or profiles (APART, Aksum), or they perform a hypothesis driven online search (Paradyn, Periscope).

A more pragmatic solution is the provision of higher-level metrics. E.g. the EXPERT tool [32] computes reasons for performance loss and represents them in a three-dimensional hierarchy, which the user can navigate. Ideally, these higher-level metrics should, however, not be hard-coded in the tool, but definable by the user. The G-PM tool presented in this paper exploits exactly this idea.

Besides G-PM, there are several other tools, which provide configurable metrics: Paradyn uses the language MDL [18] to define all of the online metrics it allows to be measured. In a similar spirit, but using a trace-based approach, EXPERT supports configurable metrics via the EARL language [31], which is based on the scripting language TCL. The goal in both cases is, however, to support the tool implementor, not the tool user. Thus, the metrics are defined at a rather low, implementation-orientated level, which is not suited for supporting user-defined metrics due to its complexity. ASL [10] and its Java descendant JavaPSL [11] enable a higher-level specification of performance *properties* used for automatic bottleneck detection. Although these languages are still intended to be used by tool developers, and performance properties are not quite the same as performance metrics, PMSL predominantly has its roots in the ASL. A really user-defined data analysis was first supported by the Pablo [23,27] and Paraver [9] tools. In contrast to G-PM, however, these tools are based on offline processing of trace data and offer weaker support for application specific metrics.

User-defined instrumentation, which is one of the building blocks of application-specific metrics in G-PM, is also supported by many other performance analysis tools. E.g. TAU [21,24] amongst other things allows instrumentation of the code with timers to measure the CPU time of program phases. The SCALEA tool [26] supports application-specific instrumentation via directives inserted into the application's source code. But in all these approaches, the instrumentation already implies a particular metrics, which is fixed at compile time, while in G-PM, many different metrics can be defined at run-time, using the same instrumentation.

3. A user's view of G-PM

G-PM supports online performance analysis for parallel MPI applications on the grid. Thus, it allows the user to define performance measurements during the application's run-time and also to examine their results while the application is still running. The results typically are numerical time series, i.e. the values of the measured metrics which evolve over time. G-PM can either display the current values, e.g. as bar graphs (cf. Fig. 6), pie charts, and matrix diagrams, or can show their temporal evolution, e.g. as curves over a time axis (cf. Fig. 8).