Parallel Computing 66 (2017) 1–21

Contents lists available at ScienceDirect

Parallel Computing

journal homepage: www.elsevier.com/locate/parco

Classification of thread profiles for scaling application behavior

Giovanni Mariani[a,*], Andreea Anghel[b], Rik Jongerius[a], Gero Dittmann[b]

[a] IBM Research, the Netherlands
[b] IBM Research, Zurich, Switzerland

ARTICLE INFO

Article history:
Received 28 July 2016
Revised 27 March 2017
Accepted 29 April 2017
Available online 1 May 2017

Keywords:
Parallel scaling
Profiling
Exascale
Clustering
Regression

ABSTRACT

Exascale applications will exploit a massive amount of parallelism. The analysis of computation and communication requirements at thread-level provides important insight into the application behavior useful to optimize the design of the exascale architecture. Performing such an analysis is challenging because the exascale system is not available yet. The target applications can be profiled only on existing machines, processing a significantly smaller amount of data and exploiting significantly less parallelism.

To tackle this problem we propose a methodology that couples *a)* unsupervised machine-learning techniques to consistently classify threads in different program runs, and *b)* extrapolation techniques to learn how thread classes behave at scale. The main contribution of this work is the classification methodology that assigns a class to each thread observed during a set of experimental runs carried out by varying the parallelism and the processed data size. Based on this classification we generate extrapolation models per thread class to predict the profile at a scale significantly larger than the initial experiments. The availability of per-thread-class extrapolation models simplifies the analysis of exascale systems because we manage a small number of thread classes rather than a huge number of individual threads.

We apply the methodology to different computing domains including: large-scale graph analytics, fluid dynamics, and radio astronomy. The proposed approach accurately classifies threads, whereas state-of-the-art techniques fail. The resulting extrapolation models have prediction errors of less than 10% for a real-life radio-astronomy case study.

1. Introduction

The exascale machine is on the horizon and will consist of a massive parallel-computing infrastructure. In some cases, an exascale system is designed specifically for a target application, for example in the field of radio-astronomy [11,25] and neuroscience [4]. To optimize the design of application-specific exascale systems, a characterization of computation and communication requirements of the target application at exascale is needed. Obtaining this characterization is challenging because profiling the application at exascale would require the availability of the exascale machine itself.

* Corresponding author.
E-mail address: giovanni.mariani@nl.ibm.com (G. Mariani).

http://dx.doi.org/10.1016/j.parco.2017.04.006
0167-8191/

This paper addresses the problem of analyzing thread-level application behavior with the goal of extrapolating profiles to exascale. In this work, the target applications are implemented either in OpenMP or MPI. We use the term "thread" to refer either to an OpenMP thread or to an MPI process depending on the programming paradigm in use. When scaling the input data or the parallelism of an application, the behavior of different threads might scale differently. Given the amount of parallelism in exascale systems, it is practically impossible to analyze the behavior of each thread individually. In this work we present a methodology to identify classes of threads whose behavior scales similarly, then we generate scaling models to extrapolate the behavior of each thread class.

An application workload can scale over a set of dimensions that represents both data size and application parallelism. For example, data-size dimensions for an ultra-high-definition video-rendering application [33] can be the frame resolution and the frame rate while the parallelism dimension for an OpenMP application is the number of OpenMP threads. We analyze the application behavior at a scale reasonable for today's machines by carefully designing experiments in the joint space of data and parallelism dimensions. Thread profiles collected in these experiments are classified using unsupervised machine-learning techniques. Classification is based on thread similarities without any a priori knowledge of the application. Once the threads are classified, analytic modeling techniques are applied to fit the observed scaling behavior for each thread class [9,12,28,29]. The proposed methodology is organized in three main phases:

- *Clustering.* Clusters similar threads within individual experimental runs.
- *Classification.* Associates thread classes to these clusters such as threads belonging to the same class scale similarly across runs.
- *Modeling.* Generates scaling models to extrapolate the behavior of each thread class.

The final goal of the proposed methodology is the extrapolation of the threads' behavior. The analysis at this granularity enables *a)* the identification of potential bottlenecks (i.e. threads whose communication or computation requirements do not scale well), and *b)* a better understanding of the heterogeneity of the threads in terms of their behavior. While the former analysis can be used by application developers to focus on the optimization of bottleneck threads, the latter analysis is valuable to system designers for tuning the computation and communication elements implementing the exascale system.

The contributions of the present work are the following:

- We present an unsupervised machine-learning technique to classify the threads executed in a parallel application.
- We propose a methodology to analyze the behavior of thread classes consistently over different experimental runs, each having a different number of threads. This analysis enables us to generate extrapolation models for each thread class by fitting the observed profiles.
- We validate the proposed methodology on a set of applications taken from different domains: *a)* a graph-analytics algorithm from the Graph 500 suite [40], *b)* a set of applications derived from computational fluid dynamics distributed in the NAS Parallel Benchmarks suite [3], and *c)* a real-life radio-astronomy case study [42].

The paper is organized as follows. Section 2 provides an analysis of the state of the art in the field of classification and extrapolation of computational processes. Section 3 defines the problem we are tackling and gives an example motivating the need for of an automated classification methodology when aiming at the extrapolation of thread profiles. Section 4 presents the details of the proposed approach. Section 5 provides empirical evidence of the accuracy of the proposed approach. Finally, Section 6 presents our conclusions.

2. Background

Unsupervised machine-learning techniques similar to the one used in this paper (i.e. clustering) are not new for the classification of executable processes. These techniques found four main applications: *a)* benchmarking [19,20,22,34], *b)* thread scheduling and mapping [8,27,31,39], *c)* phase detection [15,21,36,41], and *d)* scaling of thread profiles [15,26].

Benchmarking. Given a large set of applications, the main goal of benchmarking is the identification of a small application subset that well represents the initial set. This reduction is important to speedup the performance evaluation of computer architectures [18] and is achieved by identifying similarities between different benchmarks by clustering their profiles. In this work, we apply similar clustering techniques to identify similarities between different threads in an application. We organize a small set of thread classes whose scalability should be analyzed. The background work in the field of benchmarking [19,20,22,34] proposes a variety of clustering techniques and applies them in different application domains. For example, while some authors [22] apply factor analysis as pre-processing phase before clustering, others [19] compare genetic algorithms and principal component analysis as initial dimension-reduction technique. Then, Jia et al. [20] investigate big-data applications whereas Phansalkar et al. [34] focus on the SPEC CPU2006 benchmark suite [1].

In this work, the clustering process is carried out at thread level rather than at application level. The goal is to obtain a thread classification consistent over different runs of the target application. Techniques proposed for benchmarking cannot be directly applied to this end as explained in the motivating example (Section 3).

Thread scheduling. State-of-the-art clustering techniques at thread-level granularity are very popular [8,27,31] but their goal is not to identify thread classes along different application runs but rather the generation of thread clusters for a given application execution to optimize the mapping or the scheduling of the threads for a target hardware platform.