Parallel Computing 51 (2016) 37–45

Contents lists available at ScienceDirect

Parallel Computing

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

Pruning strategies in adaptive off-line tuning for optimized composition of components on heterogeneous systems

Lu Li[*], Usman Dastgeer, Christoph Kessler

IDA, Linköping University, S-581 83 Linköping, Sweden

ARTICLE INFO

Article history:
Available online 21 October 2015

Keywords:
Smart sampling
Heterogeneous systems
Component selection

ABSTRACT

Adaptive program optimizations, such as automatic selection of the expected fastest implementation variant for a computation component depending on hardware architecture and runtime context, are important especially for heterogeneous computing systems but require good performance models. Empirical performance models which require no or little human efforts show more practical feasibility if the sampling and training cost can be reduced to a reasonable level.

In previous work we proposed an early version of adaptive sampling for efficient exploration and selection of training samples, which yields a decision-tree based method for representing, predicting and selecting the fastest implementation variants for given run-time call context's property values. For adaptive pruning we use a heuristic convexity assumption. In this paper we consolidate and improve the method by new pruning techniques to better support the convexity assumption and control the trade-off between sampling time, prediction accuracy and runtime prediction overhead. Our results show that the training time can be reduced by up to 39 times without noticeable prediction accuracy decrease.

1. Introduction

In recent years, GPU-based heterogeneous systems have shown remarkable performance advantages over traditional multi-CPU systems. However, utilizing different types of processors within these systems efficiently remains a big challenge. Different types of processors require different programming models, thus, equipping a certain computation with multiple implementation variants on different types of processors, possibly using different algorithms and/or optimization settings, makes it possible to schedule a task, e.g. a call to such a component with multiple implementation variants, on an arbitrarily supported processing unit with higher efficiency on a problem instance, which leads to a better utilization of heterogeneity and load balancing.

Often, determining the best implementation variant over a problem instance is challenging, because the decision depends on both the invocation context's properties (such as problem size, data distribution etc.) and the hardware architecture (such as cache hierarchy, GPU architecture etc.). Hence, platform-dependent and context-dependent performance prediction plays a key role in such decisions.

Facing these challenges, two main trends coexist for prediction of the fastest implementation variant among several ones, either towards an analytical or empirical way. Compared with analytical models for performance prediction, which require extra efforts for model portability across fundamentally different platforms, e.g. a deep study of the major architectural features that

[*] Corresponding author. Tel.: +46704759692.
E-mail addresses: lu.li@liu.se (L. Li), usman.dastgeer@liu.se (U. Dastgeer), christoph.kessler@liu.se (C. Kessler).

http://dx.doi.org/10.1016/j.parco.2015.09.003
0167-8191/© 2015 Elsevier B.V. All rights reserved.

affect performance, empirical models require little or no explicit design efforts for portability at the expense of a usually higher cost for training data generation. Model portability is desirable since there already exists a wide variety of architectures, and more to come. Also, analytical modeling might be overkill in our case because implementation selection only requires predicting the expected winner, i.e. a relative (top-)ranking of implementation variants, not prediction of absolute performance metrics/values. The minimization of training time in empirical modeling for variant selection is addressed in this paper.

However, an empirical model is based on trial execution measurements, and the invocation context property value space, even if limited to e.g. problem sizes of interest, is often unpractically large such that performance modeling from exhaustive sampling might not be practical even if done off-line. Hence, efficiently choosing the best samples for training is vital for empirical prediction models to achieve both feasibility and reasonable prediction precision.

Two main alternatives exist for building an empirical model for performance prediction: online tuning and off-line tuning. The *online tuning* methods, as implemented e.g. in StarPU [1], build performance models incrementally at runtime when useful computations are executed and these measurements are obtained (almost) for free. However, online tuning can only perform well if enough well-distributed training examples are gathered, and it may experience a relatively long period of learning with still suboptimal selections, which we refer to as the “cold-start effect”.

Offline tuning allows to avoid the *cold-start effect* as long as the offline tuning overhead (including sampling) is within the user’s tolerance. In the long run, the offline tuning overhead will be amortized by reduced runtime overhead and the performance gained in many faster executions for a component during its life cycle. Furthermore, the training results by offline tuning can be fed into online tuning techniques so they start with a reasonably good basis and evolve over time.

In previous work [2] we suggested a base-line method for adaptively sampling over the whole invocation context property value space at deployment time and building a decision-tree based prediction model off-line that guides on-line selection. The technique was implemented in the PEPPHER composition tool [3] and recently also in the second-generation tuning framework of the SkePU skeleton programming library [4].

In this paper we design and evaluate an improved *smart-sampling* method for empirical off-line tuning which combines adaptive sampling with the techniques *light oversampling*, *thresholding* and *implementation pruning*, providing better prediction precision and increased pruning opportunities compared to previous work [2]. This results in lower sampling time and lower run-time and space overhead, with a better control of the trade-off between sampling time, accuracy and runtime overhead.

The rest of this paper is organized as follows: for self-containedness, we shortly review the PEPPHER component model and composition framework [3] and the base-line adaptive tuning mechanism [2] in Sections 2 and 3, respectively. The pruning strategies are presented in Section 4. Section 5 presents experimental results. Related work is discussed in Section 6, and Section 7 concludes and suggests future work. Due to space limitations, we omit some details of the evaluation that can be found in [5].

2. Composing PEPPHER components

To make component implementation selection easier, the PEPPHER framework [6] provides a tool chain to raise the programming level of abstraction and improve performance portability. Since the information needed for the selection can not be fully extracted or inferred from current programming languages, extra information needs to be provided in some form, such as XML annotations, pragmas etc. Our PEPPHER composition tool adopts the XML way for the reason that it is non-invasive to the existing components either in source code or binary form.

The *PEPPHER composition tool* [3] is a compiler-like static application build tool which generates, from the metadata, glue code containing implementation selection logic for PEPPHER component [6] and translates component calls into task form, as required by the PEPPHER run-time system, StarPU [1].

The selection by the composition tool can be performed statically based on programmer hints, or dynamically either by the prediction based on the off-line training data (as considered further in this work), or delegate the choice further to the PEPPHER run-time system.

3. Adaptive off-line tuning

This section summarizes our adaptive off-line tuning technique in [2], which is a full-phase implementation selection methodology consisting of an efficient sampling strategy, deployment-time off-line training, and generation of a compact dispatch data structure to be used in dynamic selection.

Our adaptive sampling strategy attempts to consider only significant training samples, which refer to points in the context property value space near the transition borders or decision boundaries where the fastest implementation variant or the classification decision shifts. If an implementation variant is a winner on two problem sizes of a one-dimensional problem, we assume that the variant wins on all problem sizes between the two, which means that the characteristic function for a variant to win in performance among all the other implementation variants is expected to be convex, and we leverage this *convexity assumption* for heuristic pruning of the sampling space.

We recursively split the context property value space into equal subspaces by binary partitioning up to a given cut-off depth (this maximum recursive refinement depth is referred to as the *training depth*) and invoke the component only with the input vectors corresponding to the vertices of each subspace. If we find a certain subspace of the context property value space where a specific implementation variant wins in performance on all corner points of the subspace, then the necessity to draw further training samples within the subspace becomes relatively low. We call such a subspace “closed” as we will not further explore