# Harnessing Parallelism in Multicore Systems to Expedite and Improve Function Approximation

Aurangzeb and Rudolf Eigenmann

Purdue University

**Abstract.** Approximating functions in applications that can tolerate some inaccuracy in their results can deliver substantial performance gains. This paper makes a case for harnessing available parallelism in multicore systems to improve performance as well as the quality of function approximation. To that end, we discuss a number of tasks that the function approximation schemes can offload to available parallel cores. We also discuss how leveraging parallelism can help provide guarantees about results and dynamically improve approximations. Finally, we present experimental results of a function approximation scheme.

## 1 Introduction

Many applications from different domains such as audio, video, machine learning, computer vision, gaming, data analytics, and simulations can tolerate a certain degree of inaccuracy in their results. Approximate computing aims to increase performance of these applications and/or reduce their power requirement in exchange for some tolerable loss in accuracy. Applications amenable to approximation can be concerned with performance, power, or both. In this paper, our focus is on performance only. The literature mentions a number of software, hardware, and hybrid techniques that work at different granularities. Application functions/procedures that have pure function behavior (i.e. they consistently produce the same output for a given input and have no side effects) lend themselves to approximation. Software function approximation schemes have been shown to offer significant performance benefits and we focus on black-box techniques [3, 1].

Software black-box function approximation schemes are oblivious to the internals of the original function and seek to approximate a candidate function based on its input-output behavior. This behavior is captured during *training*, which is a process of obtaining outputs from the original function. The schemes typically store the training inputs and corresponding outputs in some data-structure as training history. The schemes draw inferences from the raw history and prepare approximations by further processing the history and performing scheme-specific tasks. During *production*, the schemes choose and execute the approximations. Some schemes also have the capability to monitor the quality of approximation results at runtime. If needed, they can update the history and modify approximations dynamically. Figure 1 (a) depicts these tasks. Section 2

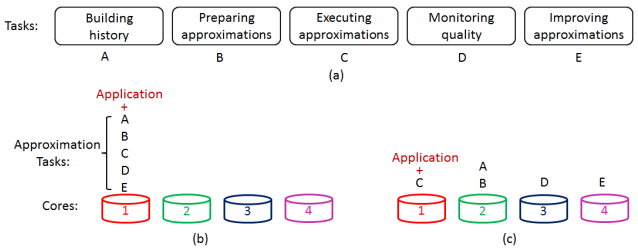


Fig. 1: (a) Tasks that black-box function approximation schemes perform. (b) Sequential execution of the application - all tasks assigned to one core. (c) Most of these tasks can be executed in parallel on multiple cores, as outlined in the subsequent sections.

describes how the schemes can harness available parallel cores in multicore systems by offloading some of these tasks to improve their performance and quality of results. By exploiting parallelism, they can also provide better monitoring of results and be equipped with the capability of dynamically improving the quality of results. Section 3 describes our experiments with a black-box function approximation scheme, called *history-based piecewise approximation* [1]. It divides the input range of a function into uniform and non-uniform regions and applies low-order polynomial approximation in each region.

## 2 Function Approximation and Available Parallelism

This section describes the tasks that software black-box function approximation schemes can offload to available parallel cores. In case of sequential applications, the schemes can freely use the parallel cores, whereas for parallel applications the cores are employed when idle, using low priority threads. Figure 1 (b) and (c) compare a scheme that does not exploit parallelism to one that does.

### 2.1 Building History

Function input-output history provides the basis for approximation to black-box function approximation schemes. Building a relevant history is important for accuracy of the approximation. Some schemes build the history offline. Where inputs during production may be significantly different than those seen during offline training, online training can improve the results. However, there may be overheads in such schemes, as the expensive original functions neeed to be called. Online training schemes can benefit from available parallel cores to build the history. In the simplest case, for every seen input, the scheme can invoke the original function in one of the available parallel cores and insert the results into the history. One drawback is that "cold start" may result in poor approximation until the history is rich enough. To overcome this problem, the schemes can speculatively build history. Below we describe some ways a scheme can do speculative training to build history online, harnessing available parallelism.

*Around Most Recent Input:* For speculative training, a scheme can use arbitrary inputs that are around the most recent actual input.

*In Most Frequent Region:* A scheme can divide the seen inputs in different regions and use arbitrary inputs in the most frequent region for speculative training.

*In Most Frequent Region of Higher Output Variation:* In addition to forming regions of seen inputs, a scheme can also track the output variation in those regions and can use arbitrary inputs in the most frequent region of the highest output variation.

### 2.2 Preparing Approximations

The schemes process the raw history, draw inferences, and perform scheme-specific tasks to prepare approximations for execution during production. For instance, the history-based piecewise approximation scheme [1] creates regions of input and computes polynomials for each region. It also considers the output variation in the regions and decides to use constants for some regions. During production, the scheme finds the region of the input and evaluates the corresponding polynomial. Offloading the inference and approximation preparation tasks to idle cores can improve the performance of a scheme.

### 2.3 Monitoring Quality

Monitoring the quality of approximation requires invoking the original function during production and comparing the exact result with the output obtained from executing approximation. Since it is an expensive process, a scheme can only monitor the output occasionally, which makes it difficult to provide guarantees for the quality of results. However, offloading the monitoring to available parallel cores can enable a scheme to potentially monitor the results of every input. It can also enable a scheme to provide guarantees for the approximations. For instance, a scheme may guarantee that a certain percentage of function invocation will result in an output that is within the specified tolerable error. At runtime, for each input, the scheme will decide whether to invoke the original function or the approximation, based on the monitoring information. Similarly, a scheme may offer statistical guarantees within a confidence interval.

### 2.4 Improving Approximations

The accuracy of approximation depends on many factors, including, the quality and quantity of training data, ability of drawing inferences, and sophistication of the approximation scheme. Harnessing idle cores can allow a scheme to dynamically improve its capabilities during runtime. It can help a scheme update its history by doing dynamic online training, draw new inferences, and improve its approximation strategies, without having any adverse effects on the performance. For example, it can allow the history-based piecewise scheme [1] to update history dynamically, adjust regions, compute new polynomials, and change approximation strategies for regions.

## 3 Experimental Results

This section describes results of our experiments with the history-based piecewise approximation scheme [1]. Currently, this scheme does not monitor results, offer guarantees, or dynamically improve results. However, it can be extended to reap the benefits of harnessing parallelism described in this paper. As for building history, it performs online training. We present results of testing three variants of the history-based non-uniform piecewise scheme on top++ application [2]. These variants are: BSA (binary search over sorted array), BST (binary search tree) and RBT (red-black tree). We chose the top++ application because the candidate function for approximation in this application is quite compute-intensive, which leads to higher overheads. The overheads of building history and preparing approximations by the variants of the non-uniform scheme for a training length of 125 are 60%, 54%, and 54%, respectively. We have extended the scheme to harness parallelism and used the *Around Most Recent Input (AMRI)* speculation described in Section 2.1. For each input, we use six speculative training inputs that are $\pm 0.04$ apart. Table 1 compares the application speedup and percentage error in results by the current versions of all variants of the non-uniform scheme that uses single core with ones by the new versions of the extended scheme that uses available parallel cores, for top++ application. The results show that employing three idle cores reduces the overhead of the scheme on a 4-core machine, substantially improving the average application speedup from 1.5x to 2.2x.

## 4 Conclusion

Software black-box function approximation schemes that aim to increase performance of applications amenable to approximation can harness available parallel cores in multicore systems to improve and expedite function approximation. They can leverage the idle cores in building history, preparing and improving approximations, and monitoring quality and offering result guarantees.

| | BSA | | BST | | RBT | |
|---|---|---|---|---|---|---|
| | Speedup | %Error | Speedup | %Error | Speedup | %Error |
| Current Version | 1.62x | 0.09% | 1.52x | 0.012% | 1.5x | 0.012% |
| AMRI Speculation | 2.3x | 0.06% | 2.14x | 0.006% | 2.08x | 0.006% |

Table 1: Effect of harnessing parallelism for building history using AMRI speculation on application speedup and percentage error of non-uniform piecewise schemes.

## References

[1] Aurangzeb and R. Eigenmann. History-based piecewise approximation scheme for procedures. 2nd Workshop on Approximate Computing (WAPCO), Jan 2016.

[2] M. Czakon and A. Mitov. Top++. http://www.alexandermitov.com/software/115-top-versions-and-downloads.

[3] M. Samadi, D. A. Jamshidi, J. Lee, and S. Mahlke. Paraprox: Pattern-based approximation for data parallel applications. In *ACM SIGARCH Computer Architecture News*, volume 42, pages 35–50. ACM, 2014.