Responses to reviewers

Valéria S. Girelli
Francis B. Moreira
Matheus S. Serpa
Danilo Carastan-Santos
Philippe O. A. Navaux

November 2020

We thank the reviewers for their comments, which helped to improve the quality of the paper. In this document, we included the comments of the reviewers and how we addressed each issue (formatted in **bold**).

Reviewer 1

The paper provides a better understanding of the memory prefetcher's role in the performance of High-Performance Computing (HPC) applications.

Authors performed an experimental campaign, executing the Numerical Aerodynamic Simulation Parallel Benchmark (NPB) using different levels of parallelism, in an Intel Skylake machine as well in a simulated environment based on both the ZSim and Sniper simulators, taking into account the prefetcher algorithms offered by both Skylake and the simulators.

Although the manuscript has a good structure, the reviewer feels that the lack of novelty and innovative research contents are major concerns. In particular, the paper presents neither an architecture nor an innovative prefetch algorithm to improve its role in the performance of HPC applications.

Comment 1

We understand and respect the reviewer's opinion. It was an important observation, which we improve in the paper and discuss in the following paragraphs.

We envision our paper as a practice investigation: there is a noticeable amount of concealment regarding the prefetcher algorithms for different architectures. These obscurities hinder the development of accurate architecture simulators and new computer architectures since companies utilize simulators to develop it.

In this scenario, we perform experimental research to understand what happens with parallel applications' performance when one processes them in modern architectures with prefetcher algorithms. To the best of our knowledge, there are very few works investigating the impact of the prefetcher over parallel architectures – with identification of performance bottlenecks, error sources, etc. – and no works at all that address the prefetcher performance on architecture simulators.

Our paper contributes to filling this hole in the literature. We conceived a paper that encapsulates all the practices and experiences we obtained when investigating memory prefetcher algorithms over parallel applications, accounting for real executions and simulations. We found many essential aspects of the prefetchers' behavior over parallel applications, notably its efficiency drop when parallelism increases. We also perceived many critical specificities of the ZSim and Sniper simulators when one uses these to simulate parallel applications' execution, taking into account the prefetcher. We pioneeringly highlight some sources (notably Figure 10) of the simulators' inaccuracy that can directly guide the simulators' developers to improve their simulators' accuracy.

In the revised version, we added a new section (Section 7) to strengthen the aspect mentioned above, emphasizing providing guidelines with technical details when pertinent. We are confident that the findings presented by the manuscript can give reliable guidance as to which practices one can follow to research and develop architectures, simulators, and prefetcher algorithms.

Reviewer 2

The paper describes a benchmark over a well known scientific application analyzing the memory cache prefetchers in parallel execution. The work compares two computational architecture simulators with a real CPU, with different memory prefetchers configurations. The conclusion shows the following outcomes:

- significant performance gain for memory prefetch for L3 and L2 when comparing to L1 prefetch.
- The parallel execution decreases the prefetch gain due to possible memory contention.
- The memory contention is not correctly performed by the simulators.

The extension compared to the WSCAD paper was perfectly achieved.

The minor revisions recommendation, doubts, and comments are:

I The performance described in Figure 2 and the next ones consider a fixed-size problem over the thread count increment (like in a strong-scaling evaluation)? If it is correct, a weak scaling performance analysis could also be added, which could help to find the reason for the performance decreasing. The paper suggests this effect occurs due to the possible memory contention in a multithread execution, but the difference in the problem granularity could also be affecting.

Comment 2

We thank the reviewer for the comment. All of the real and simulated experiments were executed using the input size class A of the NAS benchmark. We chose class A because it was the class that enabled us to perform simulations with Sniper in a reasonable time.

As cleverly pointed out, we agree that the input size can also be affecting the performance. With this in mind, in the revised version of the manuscript, we added experiments taking into account the input size class W, A, and B of the NAS benchmark. These new experiments can be found in Section 5.2 of the revised manuscript. We also added information regarding the NAS input size classes in the last paragraph of Section 4.2.

II What is the clock frequency used in all real executions? The processor Turbo frequency was disabled?

Comment 3

All executions were performed with the Intel Turbo Boost disabled, with a clock frequency of 2.1 GHz. However, we noticed some anomalies in the experiments, notably high variabilities (error bars) in some applications and metrics. We later found out that for core-level measurements, the Perf profiling tool collects data in a system-wide manner, meaning that system processes may interfere in the results. Therefore, we decided to rerun the experiments using the PAPI profiling tool since PAPI provides more control over the measurements and, most importantly, it collects performance data application-wide and not system-wide. We updated all figures that showed data about real executions with the updated results using PAPI. We added the information about Turbo Boost and clock frequency in Section 4.1 and the technical information about PAPI and Perf in Section 7.3.

III on page 11, it is stated that the prefetching performance decrease is due to irregular data access in memory. A detailed explanation about the CG-NPB could help to understand why this happened

Comment 4

We separated the CG-NPB discussion into a dedicated section (Section 5.1 of the revised manuscript) where we deepened the analysis of the CG-NPB application, with a better description of its communication pattern and its DRAM accesses, to shed light on the prefetcher's performance on GC-NPB.

IV The conclusion recommends special attention to highly parallel applications, which may benefit from turning off the prefetcher. The benchmark used comprises just parallel numerical models. Nowadays, other kinds of highly parallel applications are very common, like the ones used for graphs or Machine Learning. The recommendation could also be extended to any kind of highly parallel apps?

Comment 5

Analyzing the characteristics of each of the NAS applications and their performances during our investigation, we were able to generalize a set of application features (that we describe below) in a way that any highly parallel application that has these features can benefit from our recommendations. Our recommendations can be extended for (i) any non-embarrassingly parallel application and (ii) embarrassingly parallel applications with in-place memory operations. Many parallel applications are non-embarrassingly parallel since embarrassingly parallel preventing operations such as locking and synchronization are often required in parallel algorithms such as parallel Map Reduce or Stochastic Gradient Descent (the core of many Machine Learning Algorithms). Embarrassingly parallel applications with in-place memory operations are less common, such as the pseudo-random number generation performed by EP-NPB. Our recommendation's exceptions are embarrassingly parallel applications that operate over vectors/matrices, such as parallel vector sum or matrix multiplication. These applications do not suffer from contention caused by synchronization or locking and may benefit from prefetchers since the number of memory accesses of these applications is significant. We added the information mentioned above in Section 7.2 of the new version of the manuscript.