Highlighting PARCOACH Improvements on MBI

Philippe Virouleau, Emmanuelle Saillard, Marc Sergent, Pierre Lemarinier

Highlighting PARCOACH Improvements on MBI

Philippe Virouleau
Inria
Bordeaux, France
philippe.virouleau@inria.fr

Emmanuelle Saillard
Inria
Bordeaux, France
emmanuelle.saillard@inria.fr

Marc Sergent
Eviden
Echirolles, France
marc.sergent@eviden.com

Pierre Lemarinier
Eviden
Echirolles, France
pierre.lemarinier@eviden.com

ABSTRACT

PARCOACH is one of the few verification tools that mainly relies on a static analysis to detect errors in MPI programs. First focused on the detection of call ordering errors with collectives, it has recently been extended to detect local concurrency errors in MPI-RMA programs. Furthermore, the new version of the tool fixes multiple errors and is easier to use. This paper presents the improvements we made and the results we obtained on the MPI Bugs Initiative.

CCS CONCEPTS

• **Computing methodologies** → **Parallel programming languages**; • **Software and its engineering** → **Software testing and debugging**.

KEYWORDS

MPI, Verification, Static Analysis

ACM Reference Format:
Philippe Virouleau, Emmanuelle Saillard, Marc Sergent, and Pierre Lemarinier. 2023. Highlighting PARCOACH Improvements on MBI. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3624062.3624093

1 INTRODUCTION

PARCOACH [2, 3] is a MPI verification tool. It first detects potential call ordering errors during compile time and issues warnings. Then, it verifies the potential errors during execution. This combination of static and dynamic analyses enables an early detection of some errors, avoiding the execution of programs that can be time consuming. First focused on call ordering errors with collectives, PARCOACH has been recently extended to verify local and global concurrency errors in MPI-RMA programs [5, 6]. The static analysis now detects call ordering and local concurrency errors while the dynamic analysis verifies the errors detected at compile time and checks for global concurrency errors.

In this paper, we present new results of PARCOACH on the MPI Bugs Initiative (MBI) [4]. We compare two versions of the tool: PARCOACH v1.2 and the last version of PARCOACH, released recently (v2.4.0). Section 2 describes the differences between the two versions. Section 3 presents the results we obtained against MBI and Section 4 concludes this paper and details some future works.

2 PARCOACH VERSIONS COMPARISON

MBI aims at assessing current status of verification tools in order to help their developers improve these tools. For example, the results of Laurent *et al.* highlighted two limitations we decided to tackle in this paper: (1) a lack of RMA support in all existing verification tools, and (2) a lot of compilation errors in PARCOACH. Additionally, recent developments have allowed us to fix a lot of false positive detection in codes involving loops.

2.1 Support for RMA

The Remote Memory Access (RMA) allows processes to expose a part of their memory called *window* to perform one-sided communications (e.g., `MPI_Put`, `MPI_Get`). This feature of MPI is not yet widely used, mainly because it is challenging to ensure correctness of MPI-RMA programs. We developed a detection of local concurrency errors in MPI-RMA programs as part of a PhD thesis [1]. The method uses both a static and a dynamic analysis to detect errors [5, 6]. These analyses have been modernized and integrated into PARCOACH to complement the errors it can detect.

2.2 Improvements on Existing Results

Recent developments have allowed us to update PARCOACH from LLVM 9 to LLVM 15, and benefit from recent developments in the LLVM framework on various topics. First of all, writing the analyses and transformation passes have been made significantly easier thanks to numerous new helpers to inspect and manipulate the LLVM IR. Then from an architecture point of view, LLVM introduced new transformations and analyses managers. We migrated PARCOACH to properly use them to benefit from LLVM analyses' cache system and improve the tool performances. Finally, when it comes to the tool features, we focused the analysis improvements first on fixing compilations errors on the codes from MBI, and on

SC-W 2023, November 12–17, 2023, Denver, CO, USA

ACM ISBN 979-8-4007-0785-8/23/11…$15.00
https://doi.org/10.1145/3624062.3624093

```
Win_fence(0,win);

Get(buf,target1,win);
Get(buf,target2,win);

Win_fence(0,win);
```

(a) Example 1

```
Win_lock_all(0,win);

Put(buf, target, win);
buf = 8;

Win_unlock_all(0,win);
```

(b) Example 2

```
MPI_Datatype type = MPI_INT;

if(rank%2)
   type = MPI_FLOAT;

Iallreduce(&buf, &sum, 1, type, op, com, &req);
```

(c) Example 3

Figure 1: Examples of MPI erroneous situations.

reducing the number of false positive errors. We managed to do the latter by reworking the Breadth First Search algorithm used to find out if MPI collectives are called in the appropriate order by all MPI processes in a communicator: it had significant issues when dealing with loops, and we managed to rewrite it to be able to track the collectives called in (nested) loops.

Figure 1 shows three examples of MPI erroneous situations from MBI. The first two examples are local concurrency errors with MPI one-sided communications. In figure 1a, the two `Get` are writing in the same buffer `buf`. As there is no guarantee the first `Get` is finished before the second one, the result of `buf` is undefined. Figure 1b shows a similar situation where a `Put` reads `buf` while a store is writing in the same memory address (`buf=8`). These two local concurrency errors were not detected with PARCOACH v1.2 and are now detected with version 2.4.0. Figure 1c is an example of a parameter matching error. Even ranks call `Iallreduce` with `MPI_FLOAT` while the other ranks call the operation with `MPI_INT`. This code produced a compilation error with PARCOACH version 1.2 because some MPI nonblocking collectives were not supported in PARCOACH. The new version now supports all MPI functions.

2.3 User Experience Improvements

Over the past few months we also focused on improving the user experience, on the following aspects.

2.3.1 Packaging. PARCOACH used to require users to compile it from sources, and even sometimes to compile LLVM from sources. Our releases now address several setup and package managers: they include a shared library build (which assumes LLVM 15 is installed on the system), a static library build (which basically includes everything, for setup where LLVM 15 cannot easily be installed), and an RPM package. Since PARCOACH is also likely to be used in a HPC context, we made PARCOACH available through Guix [1] in the *guix-hpc* [2] channel.

2.3.2 Running PARCOACH. Users had to run PARCOACH by manually loading a shared library (plugin) into the LLVM optimizer *opt*, and manually running specific passes. We now provide a binary – `parcoachcc` – which can be prepended to an existing compilation command, and makes it easy to use either when manually compiling a file or when used in build systems such as autotools or CMake. For the latter, it allows running PARCOACH simply by changing the `CMAKE_<LANG>_COMPILER_LAUNCHER`.

2.3.3 Integrating PARCOACH in Existing Project. One key features of PARCOACH is its dynamic analysis. Like for most instrumentation tools, PARCOACH needs to first instrument the code, and then make sure its dynamic library is linked into the instrumented binary. We have improved the support for CMake users by providing a CMake package when installing PARCOACH, which makes it usable through `find_package`, and provide a single function to instrument a given CMake target.

These changes obviously made it easier to run the tool on the MBI test cases.

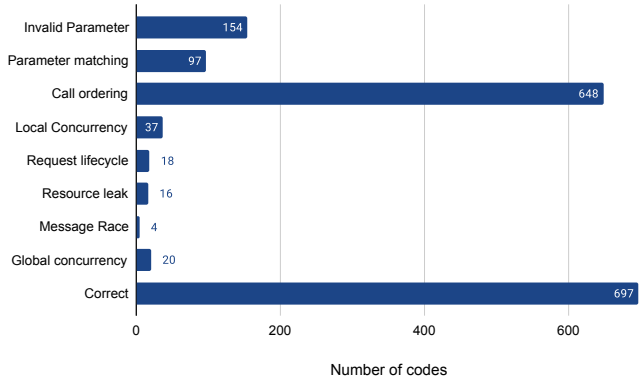

Figure 2: Number of correct and incorrect codes per error type in MBI

3 EXPERIMENTAL RESULTS

This section presents a comparison of PARCOACH v1.2 with PARCOACH v2.4.0 on MBI v1.0.0 (tag *paper*, commit 4ec1c8c4), available on Gitlab at https://gitlab.com/MpiBugsInitiative. MBI v1.0.0 contains 1691 codes including 697 correct codes and 994 incorrect codes. Figure 2 depicts the number of codes for each category of error as well as the number of correct codes.

The two versions of PARCOACH are compared regarding the metrics defined in MBI. Table 1 shows the results obtained for the two versions of PARCOACH. PARCOACH v2.4.0 is getting close to the results of an ideal tool with a coverage and conclusiveness of 1. All compilation errors have been fixed and the number of false positives has been significantly reduced. The overall accuracy, giving the proportion of correct diagnostics over all tests is equals to 0.79. This means that despite there are still errors not detected by the tool, PARCOACH is now able to correctly report several

[1] https://guix.gnu.org/
[2] https://gitlab.inria.fr/guix-hpc/guix-hpc

Table 1: PARCOACH Evaluation against the MPI Bugs Initiative benchmark (v1.0.0). CE=Compilation Error, TO= Time Out, RE=Runtime Error, TP=True Positive, TN=True Negative, FP=False Positive, FN=False Negative.

Tool	Errors			Results				Robustness		Usefulness				Overall accuracy
	CE	TO	RE	TP	TN	FP	FN	Coverage	Conclusiveness	Specificity	Recall	Precision	F1 Score	
PARCOACH v1.2	949	**0**	**0**	217	69	182	274	0.4388	0.4388	0.2749	0.442	0.5439	0.2486	0.1691
PARCOACH v2.4.0	**0**	**0**	**0**	670	679	18	324	**1**	**1**	0.9742	0.674	0.9738	0.801	0.7978
Ideal tool	*0*	*0*	*0*	*994*	*697*	*0*	*0*	*1*	*1*	*1*	*1*	*1*	*1*	*1*

errors. It is worth mentioning that we only compared the static analyses of PARCOACH. However it is possible to instrument codes in PARCOACH v2.4.0 to detect global concurrency errors during execution. This does not appear in the results.

To compare the two versions of the tool, we slightly change the docker image in MBI and the python script that builds and run PARCOACH on the codes. The detailed results for PARCOACH v1.2 (resp. v2.4.0) can be consulted on gitlab at https://parcoach.gitlabpages.inria.fr/versions-comparison-on-MBI/1.2/ (resp. https://parcoach.gitlabpages.inria.fr/versions-comparison-on-MBI/2.4.0/).

4 CONCLUSION

In this paper, we show significant improvement of PARCOACH error detection on MBI. While PARCOACH v1.2 was only focused on call ordering errors with collectives, PARCOACH v2.4.0 is able to detect local concurrency errors, has no compilation failure and is easier to use. The evaluation and assessment of a verification tool are directly linked to the correctness benchmark used. We noticed an unbalanced number of codes per error type which may benefit tools focused on call ordering errors. The results presented in this paper highlight further improvements that can be done in PARCOACH to detect more errors. We plan to add a new analysis to verify local concurrency errors with point-to-point communications in the next version of PARCOACH.

REFERENCES

[1] Tassadit Aitkaci. 2022. *Analyse et optimisations pour les applications HPC à mémoire distribuée et adressable globalement.* Ph. D. Dissertation. University of Bordeaux.

[2] Saillard Emmanuelle, Carribault Patrick, and Barthou Denis. 2014. PARCOACH: Combining static and dynamic validation of MPI collective communications. *IJHPCA* 28, 4 (2014), 425–434.

[3] Pierre Huchant, Emmanuelle Saillard, Denis Barthou, Hugo Brunie, and Patrick Carribault. 2018. PARCOACH Extension for a Full-Interprocedural Collectives Verification. In *Second International Workshop on Software Correctness for HPC Applications.*

[4] Mathieu Laurent, Emmanuelle Saillard, and Martin Quinson. 2021. The MPI Bugs Initiative: a Framework for MPI Verification Tools Evaluation. In *2021 IEEE/ACM 5th International Workshop on Software Correctness for HPC Applications (Correctness).* 1–9. https://doi.org/10.1109/Correctness54621.2021.00008

[5] Emmanuelle Saillard, Marc Sergent, Tassadit Célia Aitkaci, and Denis Barthou. 2022. Static Local Concurrency Errors Detection in MPI-RMA Programs. In *Correctness 2022 - Sixth International Workshop on Software Correctness for HPC Applications.* Dallas, United States.

[6] Aitkaci Tassadit, Sergent Marc, Saillard Emmanuelle, Barthou Denis, and Guillaume Papaure. 2021. Dynamic Data Race Detection for MPI-RMA Programs. In *EuroMPI 2021.*

A ARTIFACT DESCRIPTION

The following subsections give details on how to reproduce the results presented in the paper.

A.1 Software Availability and Dependencies

For the experiments, we relied on the MBI framework. The sources are available on gitlab at https://gitlab.inria.fr/parcoach/versions-comparison-on-MBI/. The repository contains updated versions of MBI v1.0.0 (commit ec1c8c4) to respectively use PARCOACH v1.2 and PARCOACH v2.4. PARCOACH is automatically installed in the Docker image provided by MBI.

A.2 Installation

PARCOACH is automatically installed in the Docker image provided in each MBI version folder from the aforementioned repository used for this paper. We changed the script `parcoach.py` in /script/tools/ to use the two versions of PARCOACH we wanted to compare. The Dockerfile was also updated to get the right version of LLVM.

A.3 Data Generation

To launch PARCOACH, we used the following command in the docker image:

```
python3 /MBI/MBI.py -c generate
python3 /MBI/MBI.py -x parcoach -c run
```

The first command generates the codes in a directory `gencodes/`. The second command creates a directory `logs/` containing the results of all tests. The number of FP, FN, TP and TN, as well as the metrics are computed with the following command:

```
python3 /MBI/MBI.py -x parcoach -c latex
```

A summary of the results in html format is available when launching the command:

```
python3 /MBI/MBI.py -x parcoach -c html
```

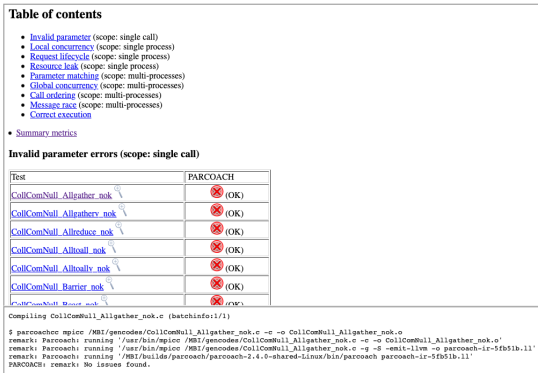

Figure 3: Screenshot of the dashboard

An online version of the results, as presented figure 3, can be consulted at https://parcoach.gitlabpages.inria.fr/versions-comparison-on-MBI.

A.3.1 PARCOACH v1.2. This version relies on LLVM 9 and builds PARCOACH from sources. The script in `scripts/tools/parcoach.py` clones PARCOACH repository and does a `git checkout 6990ff4` to go back to version 1.2.

A.3.2 PARCOACH v2.4.0. This version requires LLVM 15. The script in `scripts/tools/parcoach.py` retrieves the package of the tool and uses prebuilt binaries to compile all codes in MBI.