Systematic Development of Programs for Parallel and Cloud Computing using the Coq Proof Assistant

CS 485 – Undergraduate Research

Supervisor: Frédéric Loulergue

Fall 2019 or/and Spring 2020

1 Context

SyDPaCC [9, 8, 1] is a set of libraries for the Coq proof assistant. It allows to write naive functional programs (i.e. with high complexity) that are considered as specifications, and to transform them into more efficient versions. These more efficient versions can then be automatically parallelized before being extracted from Coq into source code for the functional language OCaml together with calls to the Bulk Synchronous Parallel ML (BSML) library.

For the optimization of sequential functions, SyDPaCC provides theorems such a the second homomorphism theorem that states that a homomorphic function is equivalent to a composition of map and reduce.

SyDPaCC also provides a set of algorithmic skeletons, programmed using BSML, and proved correct with respect to sequential functions. Algorithmic skeletons are higher-order function implemented in parallel, such as map and reduce but on distributed data structures. The way the correctness is stated is the basis of the automatic parallelization feature of SyDPaCC.

SyDPaCC is available at `https://sydpacc.github.io`.

2 The Projects

The following sub-sections describe several possible projects.

2.1 Applications of the List Homomorphism Theorems

SyDPaCC provides a formalization in Coq of the classical three theorems about list homomorphisms [9]. The current distribution of SyDPaCC contains only a few applications developed using these theorems. The goal of this project is to develop new applications using these theorems. The applications can be taken from the literature on list homomorphisms.

In developing an application using these theorems there are basically four steps, the last one being automatic:

- expressing the application as a sequential function performing the traversal of the input list from left to right,
- expressing the application as a sequential function performing the traversal of the input list from right to left,
- proving some simple properties about the operations used to implement the application,
- applying the theorems and the automatic parallelization mechanism of SyDPaCC to obtain a parallel program.

2.2 Application of the BSP Homomorphism Theory

In addition to the theory of lists (see previous sub-section), SyDPaCC provides an original theory of BSP homomorphisms [2]. It has already been used to develop verified parallel applications [8]. The goal of this project is to develop a new algorithm using this theory: sparse matrix-vector multiplication. A non-verified implementation of the algorithm exists for the C++ library OSL [6].

2.3 Applications using Trees

Recently several representations of sequential and distributed trees have been added to SyDPaCC as well as functions and skeletons manipulating these structures [10]. The goal of this project is to use these new features of SyDPaCC to implement verified algorithms on trees. The project will include experiments of the new applications on several parallel machines including Monsoon.

2.4 The Diffusion Theorem on Trees

The diffusion theorem and the associated accumulate skeleton of Hu, Iwasaki and Takeichi [3, 4] offer a way to calculate efficient programs when general accumulative computations are needed, and to efficiently parallelize them. There is an efficient C++ and MPI implementation of accumulate [5]. The diffusion theorem and accumulate extend the class of functions that can be efficiently derived into a parallel program. The diffusion theorem for lists is available in SyDPaCC [7].

The goal of the project is to add the diffusion theorem on trees to SyDPaCC. To do so, we need:

- a formalization of the diffusion theorem in Coq in such a way it can be easily used to derive efficient programs for general accumulative computations on trees,
- a verified implementation of the accumulate skeleton in Coq,
- an example of program developed using this theorem as well as experiments on a parallel machine.

3 Requirements

Minimum Requirements

- A taste for functional programming
- A taste for formal reasoning
- CS 396 Principles of Programming Languages
- CS 451 Mechanized Reasoning about Programs

Preferred Requirements

- CS 499 Parallel Programming

4 Laboratory

For the development and experiments, students will be given access to SSERL[1] (SICCS) and its machines including the Titan workstation (256 Gb of memory and 32 cores) as well as Monsoon[2].

References

[1] Kento Emoto, Frédéric Loulergue, and Julien Tesson. A Verified Generate-Test-Aggregate Coq Library for Parallel Programs Extraction. In *Interactive Theorem Proving (ITP)*, number 8558 in LNCS, pages 258–274, Wien, Austria, 2014. Springer. doi:10.1007/978-3-319-08970-6_17.

[2] Louis Gesbert, Zhenjiang Hu, Frédéric Loulergue, Kiminori Matsuzaki, and Julien Tesson. Systematic Development of Correct Bulk Synchronous Parallel Programs. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 334–340. IEEE, 2010. doi:10.1109/PDCAT.2010.86.

[3] Z. Hu, M. Takeichi, and H. Iwasaki. Diffusion: Calculating Efficient Parallel Programs. In *ACM SIGPLAN Workshop on Partial Evaluation and Semantics-Based Program Manipulation (PEPM'99)*, pages 85–94. ACM, January 22-23 1999.

[4] Z. Hu, H. Iwasaki, and M. Takeichi. An accumulative parallel skeleton for all. In *European Symposium on Programming (ESOP)*, number 2305 in LNCS, pages 83–97. Springer, 2002. doi:http://citeseer.nj.nec.com/article/hu02accumulative.html.

[5] Hideya Iwasaki and Zhenjiang Hu. A new parallel skeleton for general accumulative computations. *International Journal of Parallel Programming*, 32(5):389–414, 2004. doi:10.1023/B:IJPP.0000038069.80050.74.

[1] `https://sserl.github.io`
[2] `https://nau.edu/high-performance-computing`

[6] Joeffrey Légaux, Zhenjiang Hu, Frédéric Loulergue, Kiminori Matsuzaki, and Julien Tesson. Programming with BSP Homomorphisms. In *Euro-Par Parallel Processing*, number 8097 in LNCS, pages 446–457, Aachen, Germany, 2013. Springer. doi:10.1007/978-3-642-40047-6_46.

[7] Frédéric Loulergue. A verified accumulate algorithmic skeleton. In *Fifth International Symposium on Computing and Networking (CANDAR)*, pages 420–426, Aomori, Japan, November 19-22 2017. IEEE. doi:10.1109/CANDAR.2017.108.

[8] Frédéric Loulergue, Simon Robillard, Julien Tesson, Joeffrey Légaux, and Zhenjiang Hu. Formal Derivation and Extraction of a Parallel Program for the All Nearest Smaller Values Problem. In *ACM Symposium on Applied Computing (SAC)*, pages 1577–1584, Gyeongju, Korea, 2014. ACM. doi:10.1145/2554850.2554912.

[9] Frédéric Loulergue, Wadoud Bousdira, and Julien Tesson. Calculating Parallel Programs in Coq using List Homomorphisms. *Int J Parallel Prog*, 45:300–319, 2017. doi:10.1007/s10766-016-0415-8.

[10] Jolan Philippe. Systematic development of efficient programs on parallel data structures. Master's thesis, School of Informatics Computing and Cyber Systems, Northern Arizona University, May 2019.