Implementing Hammock Cost Techniques using the parallel programming paradigm of EFL (Embedded Flexible Language)

Oren Eliezer
Dept. of Mathematics
Azrieli Jerusalem College of Engineering
Shrayboim 26 Jerusalem, Israel
orenel@post.jce.ac.il

Moshe Goldstein
Dept. of Computer Science
Lev Academic Center - Jerusalem College of Technology
HaVaad HaLeumi 21, Jerusalem, Israel
goldmosh@g.jct.ac.il

Abstract— EFL is an embedded language which allows parallel programs to be written using the new Flexible Algorithms' approach to parallel programming, which has an imperative programming style, and ensures lock-free determinism. Hammock cost techniques have been proven useful in project management. They reduce much of projects' costs and use projects' resources more efficiently than other techniques. In this research we are proposing to implement the intrinsically parallel Hammock cost techniques by using EFL's parallel programming paradigm. We believe that this will allow to get better project overall performance.

Keywords—Flexible Algorithms; EFL; Parallel Programming; Project Management; Hammock Cost Techniques

I. INTRODUCTION

The concept of Hammock activities plays a central role in project management. Hammock activities are used to fill the time span between other "normal" activities, since their duration cannot be calculated or estimated at the initial stage of project planning. Typically, they have been used to denote the usage of equipment needed for a particular subset of activities without predetermining the estimated time the equipment must be present on site. The initial concept of Hammock activity and Hammock cost model was discussed in [11]. A few papers have been published, which deal with Hammock activities (or summary activities) and their use in practice. One of them [13] refers to a real-life huge project, in which the Hammock activities technique was used - the Westerscheldetunnel project. This tunnel provides a fixed link between Zeeuwsch-Vlaanderen and Zuid-Beveland in the Netherlands. The tunnel is very deep underground, so the planning and building process was very complex. The project was divided into four Hammock activities and approximately one billion Euros were saved.

EFL [1-3], an Embedded Flexible Language, was designed to allow an easier and implicit way of parallel programming, by embedding EFL code blocks into a sequential program written in any host programming language which provides parallel programming capabilities. Based upon the Flexible Algorithms approach to programming and computation [1], *deterministic* results are produced by the embedded EFL code blocks, whose execution is *lock-less* and *order-independent*.

In this paper we are proposing to use the EFL-based parallel programming and execution, to implement the intrinsically parallel Hammock cost techniques, establishing a new project management paradigm.

The paper is structured as follows: in section II, the main concepts of EFL and of Hammock activities, are described; in section III, the proposed EFL implementation of Hammock cost techniques is described; in section IV, conclusions, and a work plan, are presented.

II. THE MAIN CONCEPTS OF EFL AND HAMMOCK ACTIVITY

A. EFL

EFL ensures well-defined, deterministic, parallel and unordered sequential execution. The programming and execution models of EFL are described in [2]. EFL programming style is not too far from imperative programming style, allowing parallelism to be expressed implicitly rather than explicitly, significantly reducing parallel programming burden. EFL notation allows the implementation of parallel algorithms to be only slightly harder than the implementation of sequential algorithms, without giving up the full benefits of parallel execution. Also, EFL was designed with programming education in mind, trying to encourage programmers to develop their computational thinking capabilities towards Flexible Algorithms, which actually are a kind of abstraction of sequential and parallel programming, all together. EFL was designed to allow the embedding of parallel code blocks into a sequential host language program. A pre-compiler is needed to translate EFL embedded blocks of code into native parallel code, in the host language. Such a pre-compiler can be implemented for any host language, allowing programmers to learn EFL only once, using it for their parallel programming needs in different languages and parallel platforms. The approach used to implement EFL pre-compilers, and their use to produce parallel code in the host language/platform, is depicted in Fig. 1.

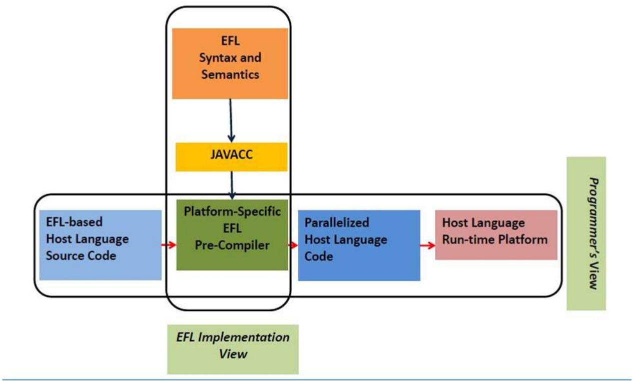
EFL Syntax and Semantics
JAVACC
EFL-based Host Language Source Code
Platform-Specific EFL Pre-Compiler
Parallelized Host Language Code
Host Language Run-time Platform
Programmer's View
EFL Implementation View

Fig. 1. EFL Implementation and use

Two EFL pre-compilers for Python as host language, have been implemented [3]. One of them has been implemented using the parallelization capabilities provided by (a modified version of) Python's Pools, available in the multiprocessing Python module [4]. A second version of the EFL pre-compiler, has been implemented upon the mpi4py-based module called DTM [5]. EFL usefulness had been validated by a case study that addresses an important kind of parallel programs, which are referred to as task trees [6] [7] [8] [9], and the work and span method [10] was used to estimate parallelism and speedup for general task trees, as well as for binary task trees [3]. That research, applied to EFL task trees, showed good speedup results. Both, the speedup of top-down and bottom-up parallel execution, over equivalent sequential Python code, increased almost linearly with the size of the task tree [3]. EFL had been designed and implemented during the last five years and continues being in active development. Results of its use in real-life applications will be reported elsewhere in the future.

B. Hammock activity in project management

A Hammock activity is an activity that we schedule between "regular" activities, since its duration cannot be estimated or calculated at the initial stage of project planning. A Hammock activity contains a collection of "regular" activities which have the same starting and ending points and/or need special equipment and/or special resource(s) [11]. A "regular" activity is different to a Hammock activity, by the fact that its duration time and starting/ending times can be determined at the beginning of the project. Since a Hammock activity has the same starting and ending points similar to a standard project, a Hammock activity should be considered itself as a project, except that there is no significance in the order between its inner activities. In order to understand the functional part of a Hammock activity in projects, we demonstrate it in an example. An entrepreneur intends to build a new building; he hires a professional construction contractor, who hires workers, determines and defines the responsibilities and functions between all the project's workers. When planning the project in its initial stage, the contractor realizes that there are some difficulties. For example, one of the major stages of the building process is planning and building the building's skeleton. The latter needs an extra expertise and/or special equipment, which the contractor lacks. Another difficulty is the fact that under the circumstances, the skeleton building should be too expensive.

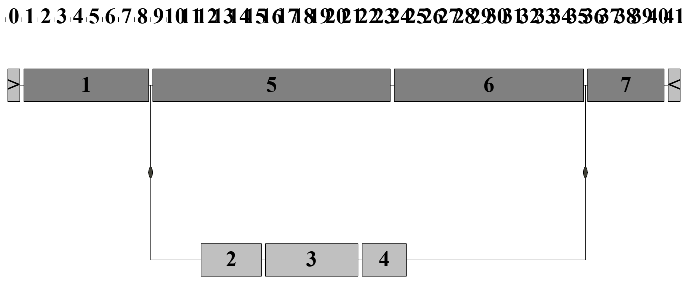

Fig. 2. An example of Hammock activity

As a result, the contractor negotiates with potential skeleton sub-contractors, who are specialized in building skeletons. The chosen sub-contractor gets the starting and ending dates of the skeleton stage out of the project contractor, and from that period he/she is responsible for all activities, tools, equipment, etc., which are needed in order to complete the skeleton stage on time. The project's contractor does not interfere in this stage, but he/she had to be sure that this stage should be finished on time. The latter is a good example of a Hammock activity in a project, and it is considered itself as a project. One important clue, regarding Hammock activities, is the term "Hammock cost".

In classic project management, the main objective is "total project makespan", which we intend to minimize. A Hammock activity is slightly different from a regular activity. It often needs a special equipment or special skill, and so, it has a different objective: to minimize the Hammock cost. The Hammock cost may be interpreted in many ways; here we will mention two different interpretations of the term: a minimization of Hammock duration [11], which is the same as makespan's minimization, or a minimization of the total cost of a Hammock activity when it is considered as a mini-project [12]. In Fig. 2, an example of Hammock activity is depicted. The example illustrates a Hammock activity, which contains three activities: 2,3 and 4, which are considered as Hammock members. The Hammock activity is $H = \{2,3,4\}$, which, as a whole, is carried out in parallel to activities no. 5 and no. 6, while the Hammock members are carried out in *any sequential order*, but *not in parallel* (*unordered sequential execution*, in Flexible Algorithms terms). The Hammock activity H begins at t=9 (its beginning time), as a successor of activity no.1, and ends at t=35 (its ending time), as a predecessor of activity no. 7. As a conclusion, H has a time range [9, 35], and the Hammock's hangers are activity no. 1 and activity no. 7, which are regular activities.

A Hammock activity cannot drive out (push out) any task activities. It behaves like a "rubber band" as it is stretched or compressed according to its successor and predecessor ─ not the other way around:

1. All Hammock members should have the same time-range – meaning that they should operate at the same time range, in parallel to other regular activities, which operate at the same time range.
2. All Hammock members should need the same sized resource(s) and constraint(s).

3. A Hammock member should not be a critical activity (the spare time between the end of an activity and the beginning of that which comes after it, called its "leg", must not be equal to zero). This helps a manager to put activities as part of the same unit: a Hammock.

III. Using EFL for implementing Hammock Activity-based Project Management

The Hammock problem under constraints (RCHCP) was solved for small-medium-large projects [12]. An interesting trade-off was shown between total project makespan and Hammock cost. The latter gives a motivation to use the Hammock activity concept in software projects. EFL programming, which is used in order to make parallel programming easier, motivated us to explore new research directions.

A large software project may be decomposed into parts. Following the same idea as Hammock activities in projects, we argue that each one of the parts of a software project may be treated as an "EFL part"; actually, an EFL code block which implements that part of the project. We propose a project to be decomposed into EFL parts. This newly proposed kind of project decomposition, called by us "Hammock-EFL" decomposition, will be done exactly the same as in the Hammock technique. That is to say that each EFL part is considered as a Hammock cost.

The main idea of the Hammock-EFL decomposition and project management procedure that we are proposing here, is to decompose a software project into major parts, which each one of them is an EFL part. The project's decomposition into EFL parts begins with the recognition of complex software code parts. When a complex software code part in a project is recognized, it is considered as a Hammock activity in the project. As we already said above, members of the same Hammock activity may be executed in *any sequential order*. That is to say that the EFL programming paradigm is a natural way for implementing a whole Hammock activity, among all the activities that compose a software project, which part of them may be executed in parallel to the Hammock activities, like we saw in Fig. 2.

We argue that the Hammock-EFL decomposition and project management procedure, proposed here, will allow to get a shorter project makespan and an efficient project's code. The stages of the proposed procedure are as follows:

1. Plan the whole project, including all tasks and sub tasks.
2. Recognize complex parts in the project (which should extend over the total makespan).
3. Decompose each complex part as a Hammock activity.
4. Implement each Hammock activity as an EFL part, using the EFL programming paradigm.

IV. Conclusions

In this paper, based on the idea of Hammock activities in projects, a new kind of project decomposition and management procedure is proposed: The Hammock-EFL procedure.

At this stage we do not have results yet. Our work plan for the realization and analysis of the proposed new project decomposition and management procedure, is as follows:

- A software package supporting it will be implemented.
- Its performance will be measured and will be compared with the performance of classic project management packages.

As soon as we will have all the results of the proposed study, we intend to publish them.

Acknowledgment

We want to thank our friend, and research partner, Dr. R.B. Yehezkael, for his insightful comments, which significantly improved the presentation and contents of this paper.

References

[1] R. B. Yehezkael, M. Goldstein, D. Dayan, and Sh. Mizrahi, "Flexible Algorithms: Enabling Well-defined Order-Independent Execution with an Imperative Programming Style", Proc. of ECBS-EERC 2015, IEEE Press, 2015, pp 75-82.

[2] M. Goldstein, D. Dayan, M. Rabin, D. Berlovitz, O. Berlovitz, R. B. Yehezkael, " Design principles of an embedded language (EFL) enabling well defined order-independent execution ", Proc. of ECBS 2017, ACM, 2017, 8 pages.

[3] D. Dayan, M. Goldstein, M. Popovic, Sh. Mizrahi, M. Rabin, D. Berlovitz, O. Berlovitz, E. Bussani Levy, M. Naaman, M. Nagar, D. Soudry, R. B. Yehezkael, "EFL: Implementing and Testing an Embedded Language Which Provides Safe and Efficient Parallel Execution", Proc. of ECBS-EERC 2015, IEEE Press, 2015, pp. 83-90.

[4] http://docs.python.org/py3k/library/multiprocessing.html..

[5] F-M De Rainville, F-A Fortin, M-A Gardner, Marc Parizeau and Christian Gagne, "DEAP: A Python Framework for Evolutionary Algorithms", Companion Proc. of GECCO'12, ACM, 2012.

[6] M. Popovic and I. Basicevic, "Test case generation for the task tree type of architecture", Information and Software Technology, vol. 52, no. 6, pp. 697-706, 2010.

[7] M. Popovic and I. Basicevic, "An Intel Cilk Plus Based Task Tree Executor Architecture", Proc. 11th WSEAS International Conference on Software Engineering, Parallel and Distributed Systems, 2012, pp 30-35.

[8] M. Popovic, M. Djukic, V. Marinkovic and N. Vranic, "On Task Tree Executor Architectures Based on Intel Parallel Building Blocks," Computer Science and Information Systems (ComSIS) , vol. 10, no. 1, pp. 369-392, 2013.

[9] K. Agrawal, C. E. Leiserson and J. Sukha, "Executing Task Graphs Using Work-Stealing," Proc. 2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS), 2010, pp 1-12.

[10] M. Popovic, M. Basicevic, M. Djukic and N. Cetic, "Estimating Parallelism of Transactional Memory Programs," Proc. 3rd IEEE International Conference on Information Science and Technology, 2013, pp. 437-443.

[11] G. Harhalakis, "Special features of precedence network charts", European Journal of Operational Research, vol. 49, no. 1, pp. 50–59, 1990.

[12] O. Eliezer and G. Csébfalvi, "A hybrid method for the resource-constrained project scheduling problem with hammock activities", Proc. of the First International Conference on Soft Computing Technology in Civil, Structural and Environmental Engineering, B. H. V. Topping and Y. Tsompanakis, Eds. Civil-Comp Press, Stirlingshire, UK, Paper 1, 2009.

[13] M. Vanhoucke, "Work Continuity Constraints in Project Scheduling", Journal of Construction Engineering and Management, vol. 132, no. 1, pp 14-25, 2006.