Bull. Nov. Comp. Center, Comp. Science, 48 (2024), 47–55

Experimental research of automatic active knowledge bases profiling and optimization based on machine learning methods on the example of array sorting problem*

Valeriy Sinyukov

Abstract. Paper discusses the problem of automatic determination of non-functional properties of operations and variables based on profiling in the active knowledge concept. Under the topic a wide range of issues outlining the direction of further research are identified. The solution architecture is proposed, the prototype of this solution is described. The viability of the main idea of this work is shown experimentally on the example of a simple computational model.

Keywords: automatic program construction, active knowledge, active knowledge bases, profiling, program optimization, non-functional program properties, AKA system, LuNA system.

Introduction

Creating an efficient application is a laborious process, which requires a specific qualification from the programmer. Gradually, as the amount of software increases, the main workload in programming shifts from writing code to finding external solutions (libraries, frameworks, modules, etc.) and adapting them to perform required tasks. This conditions the relevance for the automation of searching solutions and assembling programs from them. One of the solutions could be to use the active knowledge concept [1], which is based on the theory of parallel programs synthesis and systems on computational models [2].

As is with manual programming, for a given task there can be multiple programs that perform it, these programs may be written in different programming languages, they may implement different algorithms, use different external solutions, etc. Therefore, dependencies of non-functional properties of these programs on input data and the computer may vary. Ideally this factor should be taken into account when constructing the program. In order to accomplish this it is required that the subject constructing the program possesses information about these dependencies. This information can

*The study was carried out under state contract with ICMMG SB RAS FWNM-2022-0005.

sometimes be provided by a subject area specialist. Also this information can sometimes be obtained automatically based on statistics of constructed programs execution. The main idea behind this second approach is neither new nor unique for the active knowledge concept, it is that the efficiency of the program can be enhanced automatically as it is being used. Both of the described approaches are not universal and have their own scope. Thus, the main benefit of the second approach is its scalability: human could solve this task efficiently only for relatively small-sized systems. An automatic approach does not have such a flaw. It seems like machine learning is a good candidate for implementing such a task. The paper discusses an automatic determination of dependencies of non-functional properties and its particular case based on machine learning.

1. Terminology

Before formal problem statiting, we should introduce required terminology with simplifications, which are not essential in the context of this paper. Detailed description of the below terms can be found in [2].

The key concept in the active knowledge concept is a computational model. Computational model is an oriented bipartite finite graph, whose parts correspond to the set of operations and the set of variables. A variable represents some value, which is meaningful in the subject area, an operation is an ability to compute variable values based on other variables. Incoming and outgoing arcs of the operation determine input and output variables of this operation. For every operation there should be a software module without side effects that implements it. One module can correspond to multiple operations. A module can be represented for example as a consecutive procedure written in C++ programming language.

To construct a program based on the computational model the set of input variables, V, values of these variables and the set of output variables, W, which are to be computed, must be determined. If these sets are specified it is said that a VW-task is given.

If there is a subset of operations that can be executed to compute variable values until all the variables from W become computed, that subset is called VW-plan. It is worth noting that there can be multiple VW-plans, or there can be none.

Active knowledge base is an aggregate of computational model and description of non-functional properties of variables and operations.

Active knowledge system is a software, which constructs programs based on active knowledge bases.

In this paper, special attention is given to the concept of non-functional properties. Generally speaking, non-functional properties can vary a lot in different subject areas. An example of non-functional property of an oper-

ation could be its running time or memory consumption. Non-functional properties can be dependent on the input data and the computer, for example running time of a program implementing bubble sort quadratically depends on the size of the array being sorted. In more sophisticated cases dependencies between non-functional properties, input data and computer can be very non-trivial. For constructing better programs it can be beneficial to make predictions about non-functional properties as it will provide an opportunity to select operations that work better with given input data and computer. This process of predicting non-functional properties and constructing programs based on those predictions will be called active knowledge base optimization.

Active knowledge base optimization can be done based on statistics of active knowledge base usage. This statistics is represented with a profile. A profile will be understood as a subset of non-functional properties values collected during execution a program.

2. Problem statement

Now, when the required terminology has been introduced, we are going to state the problem, it is to automatically determine non-functional properties based on the profile to optimize active knowledge base. To reveal the topic of profiling and optimization in the active knowledge concept we shall consider its common and unique features compared to traditional software, like JIT-compilers [3] and MPI [4]. We shall also consider how exactly an active knowledge base optimization can be performed.

In MPI, processes communication is being profiled, for example, information about number of messages, their size and time for their processing is collected. In JIT-compilers, to perform speculative optimizations the number of jump instruction executions is counted. Systems like this most often operate on a low level, for example, level of intermediate representation of the program. Because of this, generally, they cannot perform algorithm-level optimizations like changing bubble sort to quick sort.

The situation in the active knowledge concept is the opposite, program and profiling information are defined on a high level, level of operations and variables. Such representation allows to perform algorithm-level optimization, like choosing a specific sorting algorithm depending on the input data. Generally such scenario can be described as follows: let there be two operations, which both use variables from set V* and both generate values for variables from set W*. It can be possible to approximate the dependency between some non-functional properties of those operations, for example, execution time, and some properties of the input variable v* from V*, for example, its size. In turn, during program construction the operation that fits the input data better will be chosen based on this approximation.

There can be multiple operations which are computed by the same software module. Thus, determined dependency of non-functional properties might be relevant to all operations, which are computed by the given module. This allows some non-functional properties of a newly added operation to be determined based on the module computing it.

As it was stated previously, non-functional properties can be arbitrary, specific to the subject area, they can depend on other non-functional properties and the computer non-trivially, but based on the profile these dependencies can be approximated. VW-task can contain an optimization criteria, for example, "It is more important to decrease memory consumption of the program, then to improve its speed". Depending on this criteria, the resulting program can vary. Since non-functional properties can be arbitrary, this allows constructed programs to be "sharpened" for a specific subject area and a specific use-case.

In an active knowledge system program can be constructed statically and dynamically. Such approach allows to predict non-functional properties statically, perform deduction of VW-plan and then adjust it dynamically as the program is being executed.

In this section, we have discussed some questions that arise in the wide-ranged topic of automatic profiling and optimization in the active knowledge concept. To perform a quality research of this topic these and other related questions should be investigated. In the next sections, the initial elaboration of the topic is described, the viability of the main idea of this work is shown on the example of a simple computational model. Also the initial research on a promising approach of approximating non-functional properties using machine learning methods is described.

3. Solution architecture

It is proposed to implement the above idea as a separate component of the active knowledge system. This component computes a number, which is an estimate for a non-functional property of a given operation or a given variable. This component does that based on the profile, VW-task identifier and the target operation or variable identifier. That estimate is used during VW-plan composition. For example, an estimate could correspond to an expected execution time of the operation. Further on, the described component will be called the optimizer. It is worth noting that even though the optimizer should not be tied to a particular active knowledge system, it is mainly targeted to be used with the LuNA system [1, 5].

The profiler should be implemented as a separate component also. In the context of this paper it is important that the profiler is a module that can somehow collect required execution statistics and save it in the profile storage accessible by other components of the active knowledge system.

During program construction and execution, modules perform the following actions:

- During construction, the optimizer is being queried to provide estimates, which affect how the program is being constructed.
- When queried, the optimizer extracts relevant profiles from profile storage and computes the estimate based on these profiles.
- The program is being executed.
- During execution, the profiler collects the profile and saves it in the profile storage.

The aforementioned workflow of an active knowledge system with the optimizer is shown in Figure 1.

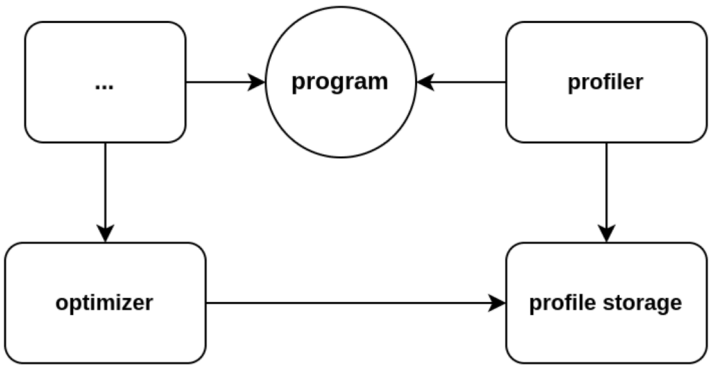

Figure 1. Active knowledge system with the optimizer and the profiler architecture. Ellipsis marks modules performing program construction and execution

4. Prototype

A monolithic prototype that is both the optimizer and the profile storage was developed using Python programming language. This prototype consists of the following parts:

- The core part contains the logic of processing requests for starting a new task, computing an estimate, saving profile and finishing a task.
- Estimating modules, which compute the estimate based on profile.
- Frontend. The prototype provides an ability to access it using command line interface and HTTP.
- Collected profiles, estimates, identifiers of tasks being executed. This information is stored in JSON files.

Sources of the prototype can be accessed as a repository on the Gitlab server[1] of the Laboratory of parallel programs synthesis of Institute of Computational Mathematics and Mathematical Geophysics of Siberian Branch of Russian Academy of Sciences.

5. Testing

Testing of the above prototype was conducted on the array sorting computation model shown in Figure 2. In this model there are two variables: initial array and sorted array, and two operations, which compute the value of the second variable based on the value of the first one. These operations correspond to two sorting algorithms: quick sort and bubble sort. Software modules that compute given operations are developed using "sorting" Python library [6].

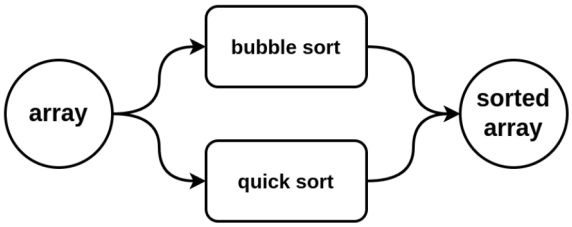

Figure 2. Array sorting computational model

Let us consider the task of optimizing sorting execution time. It is expected that bubble sort would work faster when sorting is performed on a small array, and quick sort would work faster when sorting is performed on a big array (the exact size depends on the computer and data type). In this experiment, array size and sorting execution time is used as a profile. Thus, the task for the optimizer is to predict operation execution time based on the array size and previous execution profiles.

Active knowledge system used for this experiment is the AKA system [7], into which the profiler and the module querying the optimizer were built in. Queries were performed using HTTP.

Testing was conducted for integer arrays of different sizes, elements of these arrays were generated randomly on every launch of the active knowledge base. Decision tree [8] was used as an estimating module. For each of the two operations there was one decision tree that was predicting its running time. Before a decision tree can be used for prediction, it must be trained. In this experiment, decision trees were trained on all the available profiles every time when a new request was made. Also it should be noted that these decision trees were not used until a corresponding operation was

[1] https://gitlab.ssd.sscc.ru/akso/akso

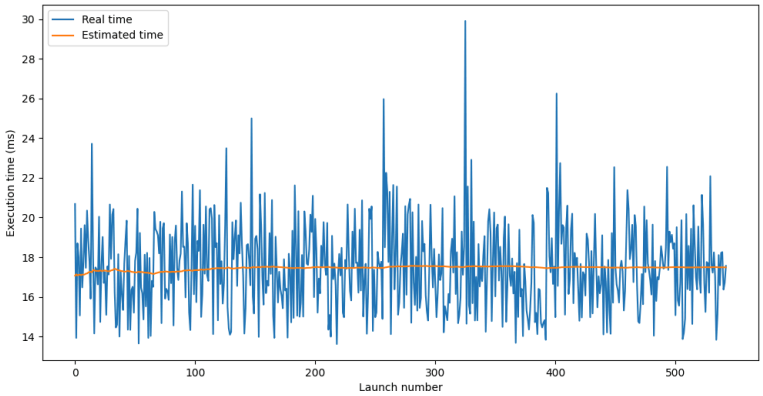

Figure 3. Dependency between the launch number and real and estimated execution time of sorting the array of size 50 using bubble sort

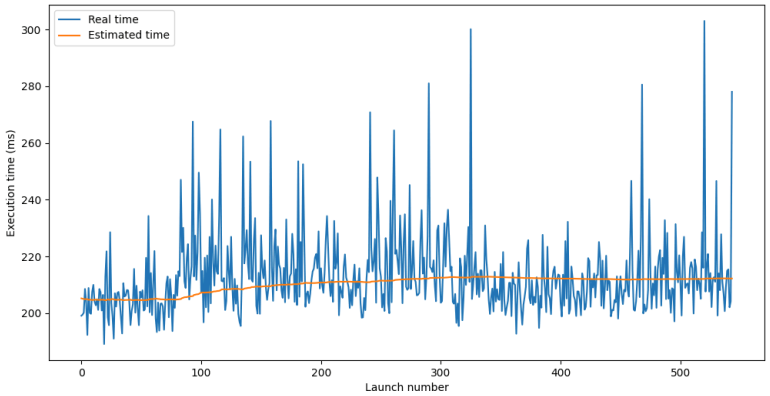

Figure 4. Dependency between the launch number and real and estimated execution time of sorting the array of size 100,000 using quick sort

performed 50 times, before that moment the optimizer was answering each request with an estimate equal to zero. The result of this experiment is shown in Figures 3 and 4.

Launches that were a part of the first 50 launches, when decision trees were not predicting, are not shown in the diagrams. As you can see, predicted values are close to the average execution time for the given operation and given array size. This result can be considered good because of two reasons. Firstly, when estimating execution time no array properties other than its size were taken into account, thus, it would be unexpected to get a better approximation. Secondly, such an approximation gives an estimate about which operation would work faster on a given array size on average, so it allows one to choose the operation which fits the given array better. In the table, the predicted execution time after 6,000 active knowledge base launches is shown for bubble sort and for quick sort.

Predicted execution time of sorting operations for arrays of different sizes

Array size	Bubble sort predicted time (ms)	Quick sort predicted time (ms)
3	17.14	18.18
5	17.16	17.43
10	18.07	17.08
20	18.76	17.02
50	17.48	18.91
100	17.89	17.22
500	32.75	17.79
1000	86.49	18.67
10,000	6,934.28	34.18
100,000	787,246.89	212.32

From the content of the table a conclusion can be drawn that on a given computer for sorting an array that has under 100 elements both operations can be used with almost equal efficiency, and for sorting an array that has 500 elements and more it is better to use the operation that uses quick sort.

Conclusion

This paper discusses the problem of automatic program construction from ready-made external solutions based on the active knowledge concept. Under this topic paper discusses the problem of automatic determination of non-functional properties of solutions based on profiling. This determination should assist in choosing appropriate external solutions based on the input data and the computer. A wide range of questions related to this topic is stated, architecture of solution is proposed, prototype of this solution and its testing results are described.

References

[1] Malyshkin V. Active knowledge, LuNA and literacy for oncoming centuries // Programming Languages with Applications to Biology and Security / C. Bodei, G. Ferrari, C. Priami (eds). — Springer, 2015. — P. 292–303. — (LNCS; 9465). — DOI: 10.1007/978-3-319-25527-9_19.

[2] Val'kovskij V.A., Malyshkin V.E. Synthesis of Parallel Programs and Systems on Computational Models. — Novosibirsk: Nauka, 1988 (In Russian).

[3] Ishizaki K., Kawahito M., Yasue T., et al. Design, implementation, and evaluation of optimizations in a just-in-time compiler // Proc. ACM 1999 Conf. on Java Grande (JAVA '99). — ACM: New York, NY, USA, 1999. — P. 119–128. — DOI: 10.1145/304065.304111.

[4] Gropp W., Lusk E., Skjellum A. Using MPI: Portable Parallel Programming with the Message Passing Interface. 3rd ed. — MIT Press, 2014.

[5] Malyshkin V.E., Perepelkin V.A. Construction of active knowledge bases for automatic design of solutions to applied problems based on the system LuNA // Parallel'nye vychislitel'nye tekhnologii — XVIII vserossijskaya nauchnaya konferenciya s mezhdunarodnym uchastiem, PaVT-2024, g. Chelyabinsk, 2–4 aprelya 2024 g. Korotkie stat'i i opisaniya plakatov. — Chelyabinsk: YuUrGU publ., 2024 — P. 57–68. — DOI: 10.14529/pct2024 (In Russian).

[6] Python "sorting" library web page. — https://pypi.org/project/sorting/ (Accessed 15.10.2024).

[7] Gorodnichev M., Lebedev D. Semantic tools for development of high-level interactive applications for supercomputers // J. Supercomput. — Vol. 77. — 2001. — P. 11866–11880. — DOI: 10.1007/s11227-021-03731-6.

[8] Quinlan J.R. Induction of decision trees // Mach. Learn. — Vol. 1. — 1986. — P. 81–106. — DOI: 10.1007/BF00116251.