Fast Static Performance Analysis of Parallel Program Schemes

Boris Sedov, Alexey Syschikov, Yuriy Sheynin, Vera Ivanova
Saint Petersburg State University of Aerospace Instrumentation
Saint Petersburg, Russia
{boris.sedov, alexey.syschikov, sheynin, vera.ivanova}@guap.ru

***Abstract*—During the embedded system development an estimation of software performance on different hardware is needed. Estimation of the expected acceleration of program execution on various numbers of processors, estimation of computation space etc. Such estimations, as other aspects of performance analysis complex, should help with different problems that arise at the intersection of software and hardware parts. For example, selection of the most successful hardware platform from available options for current program solution or modernization of the program for better resource use of given hardware platform and achieve best performance. In the paper tool of static analysis is considered. It is a part of the complex of program performance analysis in integrated development environment VIPE for perspective multicore embedded systems.**

I. INTRODUCTION

Software developing for multicore embedded systems is iterative process of design and programming. In each iteration, the structure of program could be changed. For example, detailing of various program components (lowering of the level of granularity), parallel structures, cycles optimization etc.

For parallel software, the opportunity of early estimation of potential parallelism and possible acceleration of the program depending on the number of platform processors is very important.

It gives to the developer information on how to correct the development process direction. He can check the required parallelism of the program, if there are unnecessary details in the program, does the algorithm and program correlate with hardware platform resources (memory, communications) etc. It also allows getting information about level of parallelism that will be required from the hardware platform. It is especially important in hardware-software co-design, which is typical for embedded systems development [1, 2].

The static performance analyzer of parallel program schemes could be such instrument. It allows approximately estimate parallelism level and potential acceleration of the program on various hardware configurations.

The static performance analyzer of parallel program schemes is developed as a part of the performance analysis complex. At the same time, this complex is a part of technology and design tools for portable software development for multicore embedded systems. The technology is presented in details in [3].

II. STATIC ANALYSIS OF PARALLEL PROGRAM SCHEMES

A. Representation of parallel programs

In the technology of coarse-grained visual programming, a program is represented as parallel scheme on the VPL visual programming language (Fig.1). More information about this visual programming approach could found in [4].

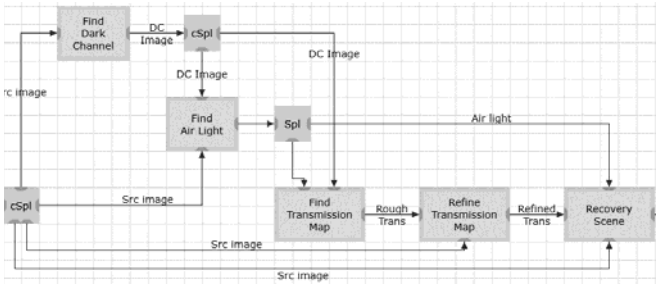

Fig.1. Example of the program scheme

During the scheme development, there is performed the iterative process of scheme elements refinement. Developer needs to estimate, either the required level of granularity and parallelism is achieved or the scheme needs further refinement or refactoring.

B. Estimation of the scheme parallelism

For each element of program scheme, the VIPE development environment allows to specify an execution time estimation in abstract units. For estimation of a developed but no fully implemented program such values should be sets manually. For fully implemented program VIPE environment provides the automated mode of program objects execution time measurement. Additionally generated code includes simplified profiling tools that collect performance statistics of the program execution on available

platform. Collected statistics could be loaded and for each program scheme element execution time data would be filled automatically.

The static analyzer uses a simple algorithm of scheme elements allocation on processors of abstract multicore hardware platform: ASAP [5]. Algorithm forms timeline of scheme execution for given number of processors and estimate the acceleration of a program (Fig.2). The estimation measured in percentage relatively to execution on a single processor.

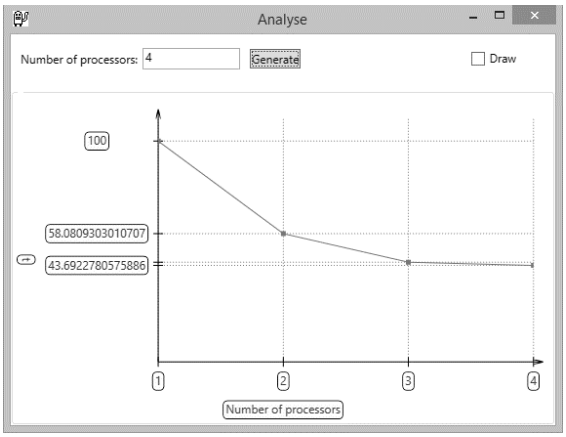

Fig.2. Program performance analysis on various number of processors

Of course, it is possible to use more complex and effective algorithms for the estimation system. But on the level of early estimation of developing parallel algorithm there is no information about target hardware platform, runtime environment and allocation algorithms, which are implemented there. Thus there is no reason to use more complex algorithms on this level.

C. Influence of factors on a program performance

Many factors influence on a program performance. For example, the level of parallelism may change dramatically depending from a structure of input data. For example, let's consider the program scheme of searching of one picture in another using the SURF [6] algorithm in the VIPE environment (Fig.3).

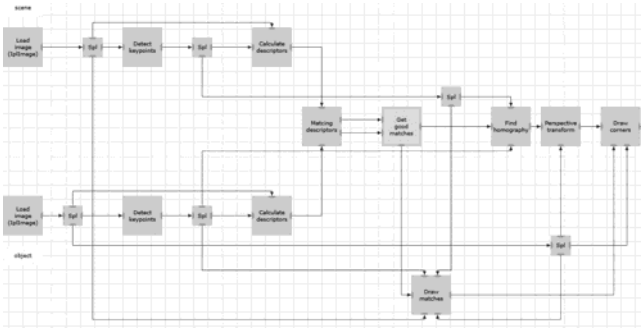

Fig.3. Scheme of image searching using SURF algorithm

At the first look, it seems that first half of a program is parallel and execution of such program on two processors will reduce the execution time of the program. Actually, the estimated performance growth on two-processor system for this program is less than 1.5% (Fig.4).

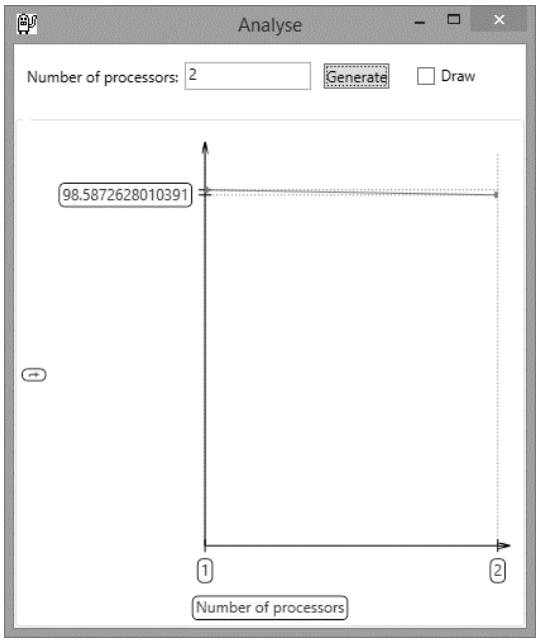

Fig.4. Diagram of estimated performance on two-processor system

The reason of such a small difference is the size of input data that are given for each parallel branch of the scheme. First branch get big picture for analysis (Fig.5), second one –small picture to look for (Fig.6).

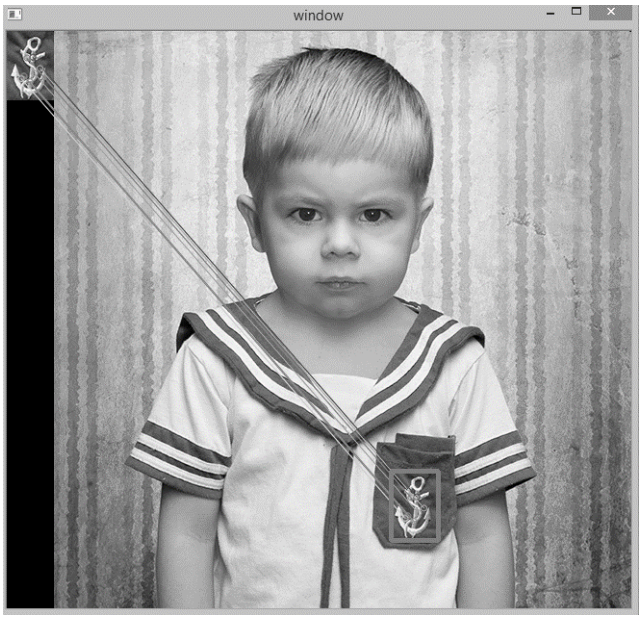

Fig.5. Analyzing image and result of the SURF program

Let's analyze factors that that affects the parallelism level of the program scheme and how presented methods of the static analysis processes them.

Fig.6. Searching image

D. Dependence of objects execution time from data size

Coarse-grained and middle-grained scheme operators represent data processing comparable to functions or procedures of traditional languages. Such functions have processing complexity dependence from processing data size. It gives significant influence on the program parallelism behavior. Here is an example of the simple program scheme of matrix processing (Fig.7).

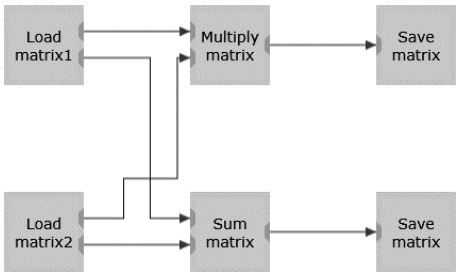

Fig.7. Program scheme of matrix processing

Matrix multiplication (the most simple algorithm) has the complexity $O(n^3)$ and matrix addition has the complexity $O(n^2)$. It is obvious that for small matrixes, such scheme will be executed really in parallel and for large matrixes execution will be almost sequential.

Static scheme analyzer allows assigning the complexity estimation of input data influence on the program objects execution time. The mode of operation with variable complexity of execution time allows estimating influence of various input data size on the parallelism level.

For the analysis purpose, the user assigns minimal data size N_{min}, base data size N_{base} (for this data the execution times of objects are assigned) and maximum data size N_{max}. Static analyzer estimate performance at border points and additionally at two intermediate points between these borders.

Result for the scheme of matrix processing analysis (Fig.7) for N_{min}=1, N_{max}=15 is shown on Fig.8.

The analysis results shows that a scheme parallelism decreases very fast on matrix sizes increase. It means that such program scheme is not suitable for parallel platform and it needs refactoring (for example, changing of the program structure with detailing/parallelization of the program component – matrix multiplication).

There are difficulties with more complex dependences from data size, such as shown in the scheme above (Fig.3). From the static analyzer point of view, there is no any problem. The real complexity is to provide to user friendly and informative interface for specification of such aspects. We are still working in this direction.

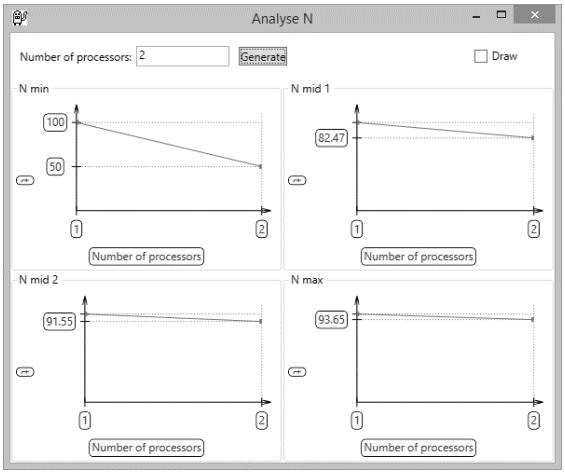

Fig.8. Diagram of dependence of program performance from the number of processors and input data size

E. Hierarchical structure of the program scheme

The program scheme in VPL could contain both terminal operators (nodes that are indivisible from the program scheme point of view) and complex operators (structured nodes). Structured nodes are designed for hierarchical structuring of the program. Structured nodes could contain in its body terminal operators and other complex operators (Fig.9).

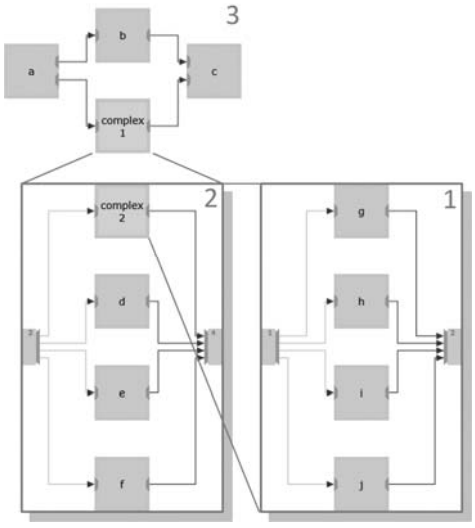

Fig.9. Hierarchical structure of the program scheme

The mechanism of a parallel execution of structured constructions hardly depends from hardware platform characteristics and used runtime. For example, OpenMP standard [7] implementation could execute complex structures in parallel mode and in sequentially mode, depending on set property (omp_set_nested / OMP_NESTED). Complex constructions could be allocated on all evaluable processors or on a part of them, etc.

There are two extreme models of the complex structure execution in the static analyzer: fully sequential and fully parallel.

In the sequential mode, all nodes of the complex operator body are allocated on the single processor (Fig.10).

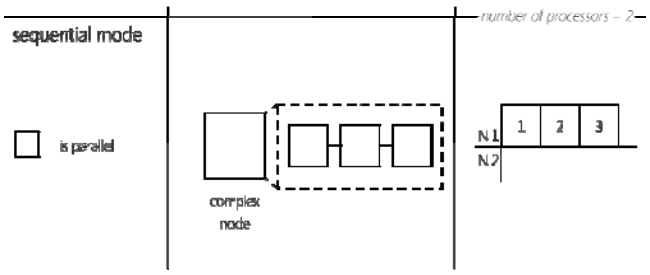

Fig.10. Sequential execution of the complex operator

In the parallel mode, all nodes of the complex operator body use all available processors according to common rules (Fig.11).

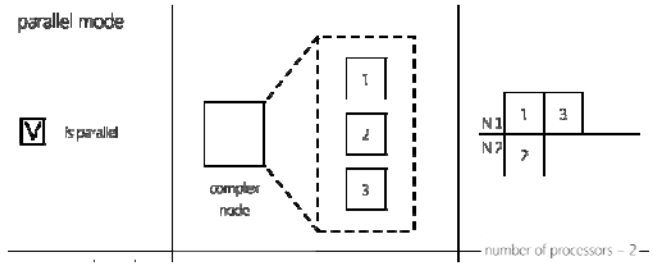

Fig.11. Parallel execution of the complex operator

Execution mode of each complex operator of the program should be specified by user with “Is parallel” property. Complex operators are allocated to processors according to their execution model during the scheme analysis. It will be illustrated on the scheme presented on the Fig.12, where C1 and C2 are complex operators.

In the sequential mode, complex operator is allocated on a single processor (Fig.13). Execution of the structured operator could not be interrupted until the end of its execution and any other operands could not be allocated to this processor. Execution time of this complex operator is equal to the execution time of all operators inside its body. Other processors could be occupied by other operands of the base scheme, including structure nodes.

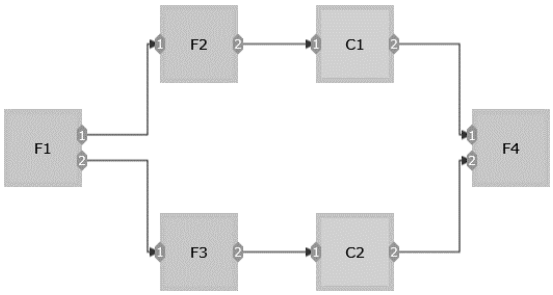

Fig.12. Execution mode illustration

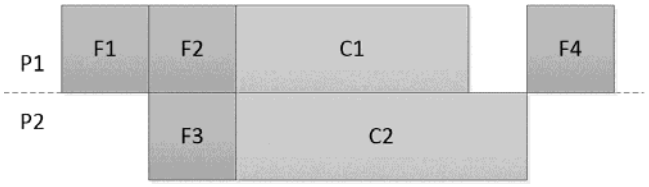

Fig.13. Allocation of complex operators in sequential mode

In the parallel mode, complex operator is allocated on all processors of the system (Fig.14). It does not been taken into account how many processors were actually occupied by operators of its body. Execution of the complex operator could not be interrupted until the end of its operation. Any other scheme operators at this time cannot be allocated.

Fig.14. Allocation of complex operators in parallel mode

The case of multiple complex operators, the distribution to processors is calculated starting from the lowest control operator and up to the top-level scheme. For the scheme from Fig.9, operators’ allocation is represented on the Fig.15, and its performance analysis on the Fig.16.

E. Conditional and iterative loops

In addition to terminal and complex operators, any programs could also contain conditional (while) and iterative (for) loops. Most computations of a program is

presented in such loops, and they have a significant influence on the program performance.

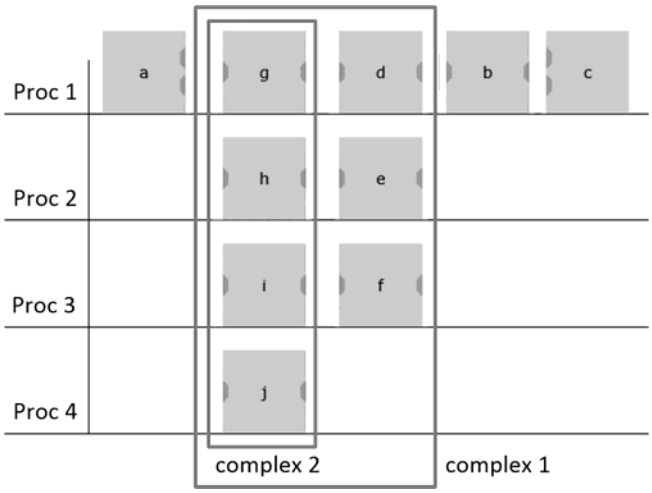

Fig.15. Allocation of the scheme with complex operators

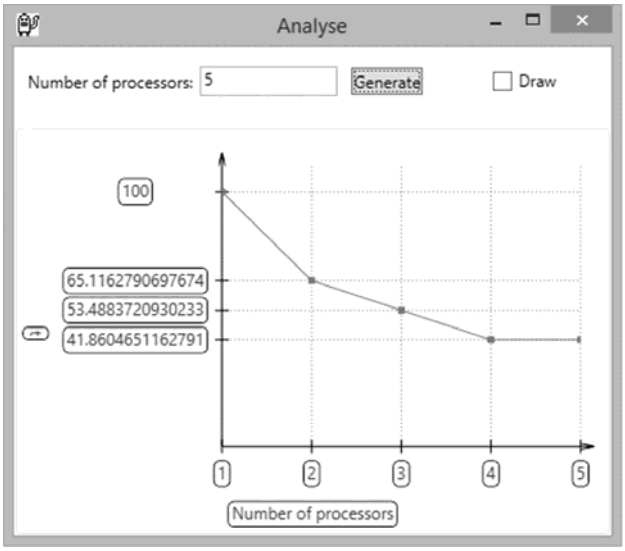

Fig.16. Performance analysis of the scheme with complex operators

For the program performance analysis, the most important loop parameters are the number of loop iterations (its body execution) and its execution models. The implemented method of a loop analysis combines both methods of processing terminal operators and complex operators.

The number of iteration can be defined by user manually or automated by using profiling tool that was mentioned above. The performance analysis system allows specifying the rule of dependence between input data size and number of iterations.

Calculation of the loop processing is made in the same way as for complex operators. For each loop user should also specify the execution model: sequential or parallel. Allocation of the cycle on processors is performed in a similar manner. The execution time of the loop is multiplied by the number of iterations according to the specified dependency rule.

Here is an example of the program with loops processing on Fig.17 and its allocation on Fig.18.

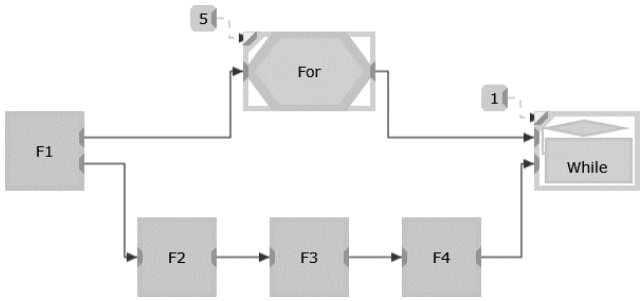

Fig.17. Example of the program with loops

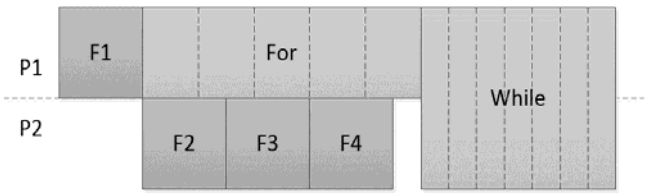

Fig.18. Allocation of the program with loops

III. RESULTS AND PERSPECTIVES

The static analyzer is a useful tool for quick early estimation of program characteristics. Among them there are performance, parallelism, maximal processors occupation, memory occupation etc. It allows estimating on early stages and with minimal effort the performance of program schemes on potential hardware platforms with various number of processors and for various input data size. At any stage of the development of a parallel algorithm for the task, a scheme designer has an opportunity to evaluate the program and identify a further way of its design.

Of course, the static analyzer is not designed for getting accurate time measurement of the program. There are too many factors, which depends on hardware platforms, runtime and parallel execution models and all of them affects the program performance. More detailed analysis can be done by using other tools of the performance analysis complex, for example, by using virtual simulator tool or simulator of the coarse-grained model of hardware platform.

ACKNOWLEDGMENT

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation under agreement n°14.575.21.0021, identifier RFMEFI57514X0021.

REFERENCES

[1] Bertels, K. L. M., *Hardware/Software Co-design for Heterogeneous Multi-core Platforms*. Springer, Drodrecht Heidelberg London New York, 2012.

[2] Teich, Jürgen, "Hardware/software codesign: The past, the present, and predicting the future", *Proceedings of the IEEE 100. Special Centennial IssueI,* 2012, pp. 1411-1430.

[3] Boris Sedov, Alexey Syschikov, Vera Ivanova, "Technology and Design Tools for Portable Software Development for Embedded Systems". *Proceedings of the 16th Conference of Open Innovations Association FRUCT printed by "University Telecommunications" Company*, 2014, pp. 86-93.

[4] Vera Ivanova, Boris Sedov, Yuriy Sheynin, Alexey Syschikov, "Domain-Specific Languages for Embedded Systems Portable Software Development", *Proceedings of the 16th Conference of Open Innovations Association FRUCT printed by "University Telecommunications" Company*, 2014, pp. 24-30.

[5] Baruch, Zoltan, "Scheduling algorithms for high-level synthesis", *ACAM Scientific Journal 5.1-2,* 1996, pp. 48-57.

[6] Juan, Luo, and Oubong Gwun. "A comparison of sift, pca-sift and surf." International Journal of Image Processing (IJIP) 3.4 (2009): 143-152.

[7] Broquedis, François, "Structuring the execution of OpenMP applications for multicore architectures", *Parallel & Distributed Processing (IPDPS)*, 2010.