Examination of the input dependence of data flow graph clustering

Gábor Wacha
Department of Measurement and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
Email: wacha@mit.bme.hu

Béla Fehér
Department of Measurement and Information Technology
Budapest University of Technology and Economics
Budapest, Hungary
Email: feher@mit.bme.hu

***Abstract*—Control- and data flow analysis of a given software can give insights on the internals of the algorithm. With the generation of a data flow graph based on runtime measurements, it is possible to separate data independent parts of a given program. However, the result of the runtime measurements is dependent on the input of the software.**

In this paper we introduce a method to estimate the impact of the input variables on the data flow graph, in particular on the different clusters of the graph. We test our approach with the JPEG image compression algorithm.

I. Introduction

With manycore architectures becoming more common, embedded application developers face a new problem – task distribution on the different processor cores.

Profile analysis can help to find an even load distribution on the different cores. Load balance aware task partitioning algorithms exist, for example in [1], however, the resulting system is likely to suffer performance drawbacks in cache usage: passing large amounts of data between the partitions assigned to different cores can result in more miss in the processor local cache.

The data flow analysis and spectral clustering method introduced in [2] can be used to find data independent partitions of the algorithm under test. From run-time measurements the amount of data passed between specific functions of the software is recorded into an Aggregated Dynamic Data Flow Graph (ADDFG) [3]. The ADDFG is a directed graph. The nodes of the ADDFG correspond to the software functions in the algorithm. Each edge has a weight which corresponds to the amount of data transferred between the functions during a given execution.

A spectral clustering method can show data-independent partitions, where communication between the functions residing in different partitions is kept as minimal as possible. With the explored partitions, a cache-aware task scheduling can be achieved.

However, the analysis relies on run-time measurements. One of the problems of run-time measurements is the input sensitivity of the result. The ADDFG – on which the clustering and the explored partitions are based – can be different on every execution of the algorithm.

We propose a method to examine the impact of the different input variables on the ADDFG, and particularly on the explored partitions. First, we introduce the assumptions we made on the software under measurement in Section II. Then we describe our analysis workflow in Sections IV, V and VI. Finally, we demonstrate our methods in Section VII.

II. Generalized software model

In this paper, we regard the software as a nonlinear, deterministic function, which takes input parameters, and returns a feature as output. This is different than the "classical" software, which takes input parameters, and returns a calculated output value, since a feature is a more abstract object: it can also be a call graph, a coverage result, or a given hardware state of an embedded system.

The software will be regarded as a gray box: every internal state and executed instruction is observable, but the activity (e.g. adding two values) of a specific processor instruction is unimportant.

A. Software parameters

Most algorithms have input parameters which influence the execution. We classify these input parameters into the following groups:

- *Main dataflow*: the input parameter is part of the data on which the software does its main calculation. For example, raw image data in the JPEG compression algorithm is part of the main dataflow.
- *Data length descriptor*: the input parameter describes the length of the main dataflow. This parameter is often not given explicitly, it is inferred from the size of the main dataflow. For example, the data length descriptor parameter is the image size in the JPEG compression algorithm.
- *Behavior selector*: the input parameter selects from different sub-algorithms. Different algorithms result in significantly different behavior. The different behavior typically manifests in different program states or different call graphs. For example, in a JPEG handler software, the input which selects between image compression or decompression is a behavior selector parameter.

We assume that the value of every input parameter is available at the start of the execution of the given software, however, in many embedded software the timing of an actual

input (e.g. the time when an interrupt arrives) can influence the output.

To model the time course, we can regard these inputs as value-time pairs. With this, timing information is included in the software parameters. The concrete analysis of timed input parameters is out of the scope of this paper.

B. Software output

As stated in Section II, we regard the output of the algorithm as a feature. In usual software profiling cases the feature can be an execution time, a call graph or a memory footprint. In our paper the output of the software is the ADDFG of the given program execution. We view the ADDFG as a graph adjacency matrix, with each element of the matrix is one output.

III. Outline of the analysis

Our method to analyze the impact of the input variables on the ADDFG can be summarized in the following steps:

1) *Input generation*. During this step we prepare the list of possible inputs for the algorithm under test. The main difficulty is to find a finite set of input combinations, which reflects the infinite set of possible inputs. We show how to handle the different kind of inputs introduced in Section II-A.
2) *Algorithm execution and ADDFG generation*. We execute and observe the algorithm with the inputs generated in Step 1. to create the output. The execution results in a set of ADDFGs. The main difficulty is to monitor the memory and register access of every executed instruction, which is needed for the ADDFG generation.
3) *Error estimation*. From the set of ADDFGs generated in Step 2., we give an estimation of the possible error of the clusterization method. The main difficulty is to find an estimation, which can be used with the ADDFG, and which also gives practical results.

IV. Input generation for the algorithm under test

Since we do not know a direct relationship between the input parameters and the ADDFG, to examine the impact of each input variable, we have to generate the ADDFGs for different inputs.

If the domain of all input parameters is a finite set, for every combination the ADDFG can be generated. However, most algorithms do not have this property. The solution can be to generate the ADDFG to a finite set of input combinations, while taking care that input data for the measurements reflects the input expected in the real world application.

In the following sections we assume that every input is already classified into the groups introduced in Section II-A. This assumption is valid, since a real world algorithm is a gray box: we have *a priori* knowledge what each input parameter does.

The main problem is that for each different kind of input the ADDFG can be different. This means that different kinds of input parameters should be regarded differently.

To simplify the input dependence analysis, we have the following assumptions:

- Behavior selector variables have finite number of combinations in real world applications. This means we can generate a set of ADDFGs for each behavior selector variable combination, thus partitioning the original problem into independent subproblems.
- The impact of the data length descriptor variables can be examined as follows. The control and data flows of a real world algorithms usually have a relationship with the data length describable in a closed form expression. This allows us to examine only a finite set of input lengths and deduce the input dependency over the data length from the results.

With these assumptions, only the impact of the main dataflow has to be examined, we can constrain the data length and the selected behavior to fixed values.

For the main dataflow input variables we need *a priori* knowledge on the use case of the software. We have to generate inputs with a fixed data length, which reflects real-usage input data.

To clarify, for example in a JPEG handling software we constrain the behavior as compression, and we generate different, but fixed size input images, where the image is from real world data.

V. Algorithm execution

To analyze the impact of the different input parameters on the ADDFG, the software should be run in an observeable, reproducible environment. To generate the ADDFG for each input generated according to Section IV, the memory or register access of each executed CPU instruction has to be traced.

If we trace only the instruction counter, the address of the accessed memory and the direction of the data transfer for each executed processor instruction, we can estimate the necessary bandwidth of the trace data.

In a standard 3-operand instruction set (like in the case of the ARM), one instruction can have up to 3 data accesses. Calculating with 32 bit memory addresses, a 32 bit instruction counter, and a processor running on a relatively low clock frequency of $\tilde{1}00$ MHz, the necessary bandwith is in the magnitude of at least 1 GByte/s.

This bandwidth is not available on trace hardware, in consequence, for tracing a processor simulator is used. The simulator executes each instruction, and stores the trace information. Because the effect of every executed instruction on the registers or on the memory is traced, modern techniques – such as binary translation – can not be used effectively.

The trace information for one execution – if we estimate the simulation time with 1 real world second – stores approximately a hundred million records. This data is inserted in a database, the trace log of each executed instruction is stored. Insertion of each instruction separately has strong impact on the performance of the simulation, since for every executed instruction, a database query should be generated, sent to the

database server and parsed. Prepared statements can achieve better performance, however the best performance is achieved with bulk loading the trace data into the database.

The following table shows the results of our measurements of the average simulation speed in MIPS using the PostgreSQL database server and OVP Microblaze simulator:

Insertion strategy	Simulated MIPS
Query	0.2
Prepared statement	0.3
Bulk insert	0.5
Without database insertion	0.6

According to the results, the simulation is two magnitudes slower than the real hardware with 100 MIPS. However, the simulator performance is enough to observe the execution of the embedded system and generate the ADDFG.

VI. Error estimation of the spectral clustering method

A. Finding the clusters of the ADDFG

The clustering algorithm is the following [4]. Let G be the weighted adjacency matrix of the directed ADDFG. The clustering algorithm is only usable on symmetric matrices, the undirected ADDFG should be defined. We assume that on each data transfer between functions the performance loss is the same regardless of the direction of the transfer. With this assumption, we can define the undirected ADDFG, where the direction of the edges can be ignored. The adjacency matrix A of the undirected ADDFG is calculated from G as stated in Eq. 1.

$$A = (G + G^T) - diag(G). \tag{1}$$

A is a symmetric matrix, on which the spectral clustering method [4] can be used. Let L be the unnormalized Laplacian matrix of A [5]:

$$l_{ij} = \begin{cases} deg(i) & \text{if } i = j \\ w(i,j) & \text{if } i \neq j, \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $deg(i)$ is the degree of the i-th vertex, and $w(i,j)$ is the weight of the edge between the i-th and j-th vertex. When there is no edge between i and j, $w(i,j)$ is 0.

The unnormalized Laplacian matrix can be calculated as $D - A$, where D is the diagonal degree matrix of the weighted symmetric adjacency matrix A:

$$D_{ij} = \begin{cases} deg(i) & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

The eigenvectors of L hold information about the clusters of A. All coordinates of the eigenvector corresponding to the smallest eigenvalue of L are the same, the eigenvalue is 0, resulting that the Laplacian matrix is singular. All the other eigenvalues are real [5].

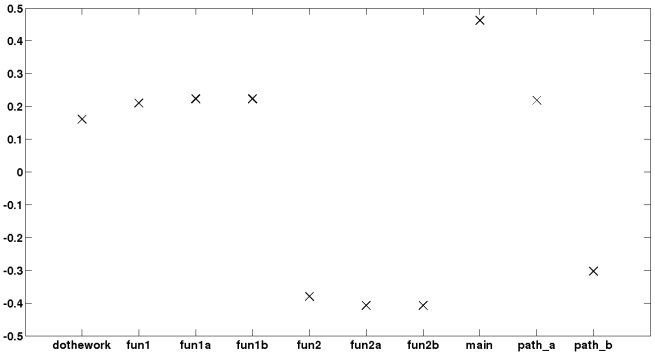

Fig. 1. Plot of the eigenvector coordinates corresponding the second smallest eigenvalue of L in a demonstration algorithm

The coordinates of the eigenvector of the second smallest eigenvalue can be used to find the clusters of A. Comparing the coordinates (which are real numbers, so a comparison can be done) of the eigenvector to a given threshold value can give two clusters of the graph. The i-th vertex of the ADDFG is in the first cluster if the i-th coordinate of the eigenvector is less than the threshold value, in the second cluster otherwise. 0 is often the most straightforward choice for two clusters [6].

Fig. 1. shows the plot of the coordinates of the eigenvector generated from the ADDFG of a demonstration algorithm. On the x axis the function name corresponding to the given coordinate is given. We can separate two main clusters if we select the threshold value as 0.

B. Error estimation

For every execution, the ADDFG – and consequently the Laplacian matrix L – can be different depending on the input. If we use the partitions explored resulting from a given execution for software parallelization, it is possible that for other inputs the difference in the ADDFG results in different clusters. Very different clusters will result in suboptimal parallel execution and cache usage.

The partitioning algorithm introduced in Section VI-A finds the clusters based on the eigenvectors of the graph Laplacian of the ADDFG. The problem of error estimation can be reworded as estimation of the sensitivity of the eigenvectors regarding perturbations on the Laplacian.

The perturbation theory of eigenvalues and eigenvectors is a remarkably active area of research. Many methods exist, see in [7], [8], [9] and [10]. However, because of the properties of the Laplacian matrix defined in Eq. 2, most approaches give unusable results: the Laplacian is singular, not positive definite, and its eigenvalues are not unique.

L. Huang et al. in [9] show a practically usable estimation on the clustering error. They define η, the *mis-clustering rate* as the proportion of samples that have different cluster memberships after data perturbation.

The first step of their method is to approximate the mis-clustering rate in case of eigenvector perturbation. Let L the Laplacian of the ADDFG, $\widetilde{L}$ a perturbed Laplacian, $dL =$

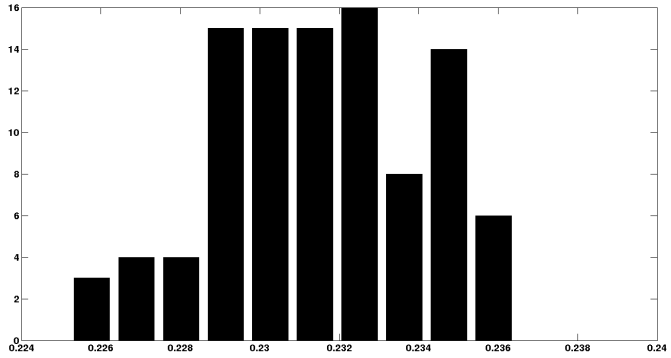

Fig. 2. Histogram of the edge weight between *encode_one_block* and *emit_bits_s* in the JPEG compression algorithm

$L - \widetilde{L}$. Let v_2 and $\widetilde{v}_2$ be the eigenvectors corresponding to the second smallest eigenvalue of L and $\widetilde{L}$ respectively.

An estimation of η is given as:

$$\eta \leq \|\widetilde{v}_2 - v_2\|^2 \tag{4}$$

To calculate η, an estimation to $\|\widetilde{v}_2 - v_2\|$ is necessary. According to [11], the norm of the difference of v_2 and $\widetilde{v}_2$ can be estimated as

$$\|\widetilde{v}_2 - v_2\| \leq \frac{\|4dL\|_F}{\nu - \sqrt{2}\|dL\|_F}, \tag{5}$$

where ν is the difference between the second and the third eigenvalues of L.

The Frobenius-norm $\|dL\|_F$ is calculated as the square root of the sum of the absolute squares of the elements of dL. With uncertainity evaluation, the worst case value of $\|dL\|_F$ can be calculated from the measured ADDFGs.

VII. Results

We tested the approach on the libJPEG compression. With different, 32x32 sized inputs, we generated the ADDFG of each execution.

The generated ADDFGs had the same edge weigths between the nodes, except *encode_one_block* and *emit_bits_s*. This is not unexpected, since the JPEG compression algorithm works on fixed size blocks. The only part of the algorithm which has different input sizes is the Huffman coding, since the Huffman coding is a variable length coding.

Fig. 2. shows the histogram of the edge weight of 100 executions. The edge weights of the ADDFG were normalized so that the trace of the ADDFG is 1.

As we stated in Section II,. the software is an unknown nonlinear function of the input parameters resulting in the ADDFG. This means, that the distribution of the edge weight is unknown. Normality tests show that the edge weight can not be modeled as a normal distribution. To give a confidence interval on the average edge weigth, Chebyshev's inequality can be used. We used a 90% confidence interval to examine the mis-clustering rate.

With the worst case perturbation (where the Frobenius norm of the error of the Laplacian matrix was pessimal), the mis-clustering rate was calculated as 0.01. This means only 1% of the 256 functions in the JPEG algorithm can be in the wrong cluster in case of an unfortunate input.

VIII. Conclusion

In this paper, we introduced a method to estimate the error of the clusterization of an ADDFG of a given algorithm. We described how the different kinds of input parameters can influence the generated ADDFG. A method was given to enable the fast simulation and observation of the algorithm under test. We estimated the mis-clustering rate with methods described in [9]. We tested our approach with the JPEG compression algorithm.

In future research, the impact of the timed input parameters can be examined. Also with input dependency analysis it could be possible to enable better profiling, directed code coverage measurement and algorithm optimization.

References

[1] J. Kang and D. Waddington, "Load balancing aware real-time task partitioning in multicore systems," in *Embedded and Real-Time Computing Systems and Applications (RTCSA), 2012 IEEE 18th International Conference on*, Aug 2012, pp. 404–407.

[2] G. Wacha and B. Fehér, "Examination of algorithms using dynamic data flow graphs," in *Proceedings of the 21st PhD Minisymposium*, 2014, pp. 44–47.

[3] I. Szabó, G. Wacha, and J. Lazányi, "Aggregated dynamic dataflow graph generation and visualization," *Carpathian Journal of Electronic and Computer Engineering*, vol. 6, no. 2, pp. 50–54, 2013.

[4] A. Y. Ng, M. I. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," in *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS*. MIT Press, 2001, pp. 849–856.

[5] B. Mohar, "The Laplacian spectrum of graphs," in *Graph Theory, Combinatorics, and Applications*. Wiley, 1991, pp. 871–898.

[6] U. Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, Dec. 2007. [Online]. Available: http://dx.doi.org/10.1007/s11222-007-9033-z

[7] W. L. Xiao Shan Chen and W. W. Xu, "Perturbation analysis of the eigenvector matrix and singular vector matrices," *Taiwanese Journal of Mathematics*, vol. 16, pp. 179–194, 2012. [Online]. Available: http://journal.taiwanmathsoc.org.tw/index.php/TJM/article/viewFile/1588/1263

[8] R. Mathias, "Spectral perturbation bounds for positive definite matrices," *SIAM J. Matrix Anal. Appl*, vol. 18, pp. 959–80, 1997.

[9] L. Huang, D. Yan, N. Taft, and M. I. Jordan, "Spectral clustering with perturbed data," in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds. Curran Associates, Inc., 2009, pp. 705–712. [Online]. Available: http://papers.nips.cc/paper/3480-spectral-clustering-with-perturbed-data.pdf

[10] F. M. Dopico, J. Moro, and J. M. Molera, "Weyl-type relative perturbation bounds for eigensystems of Hermitian matrices," 2000.

[11] G. W. Stewart and J. Guang Sun, *Matrix Perturbation Theory*. Academic Press, 1990.