# A Profiling Framework for Design Space Exploration in Heterogeneous System Context

Kamana Sigdel, Roel Meeuws, Koen Bertels

Computer Engineering, EEMCS

Delft University of Technology

{k.sigdel, r.j.meeuws, k.l.m.bertels}@tudelft.nl

*Abstract—*

**Design of embedded systems is subject to different types of design constraints such as execution cycles, power consumption, and memory consumption/bandwidth. At the same time, modern computing systems make increasing use of reconfigurable and heterogeneous architectures. The increasing heterogeneous nature of embedded system platform and the application makes the design of embedded system very complicated. In hardware/software co-design environment, to make early design decision such as mapping of application onto heterogeneous set of processors, it is necessary to perform exhaustive design space exploration in order to identify the various design parameters such as execution time, memory, bandwidth etc. Profiling is one of the such techniques that helps to measure these design parameter and helps to quickly find promising candidate for mapping onto heterogeneous set of processor. Profiling and tracing provide necessary information to analyze the programs statically and/or dynamically in order to determine relevant information for design space exploration, hardware/software partitioning and optimization. Towards this goal, in this paper, we present an outlook of the methodology for the dynamic analysis of the application for design space exploration, hardware/software partitioning and parallelization within Delft Workbench.**

*Keywords*: **Profiling, Reconfigurable Computing, Design Space Exploration, Hardware/Sofware Partitioning**

## I. INTRODUCTION AND BACKGROUND

The main goal of hardware/software co-design is to shorten time to market while reducing the design effort and costs of designed products. Design of embedded system is subject to many different types of constraints such as computation cycles, power consumption and memory consumption. Partitioning an embedded application among software running on a microprocessor and reconfigurable hardware such as FPGA( Field Programmable Gate Arrays) improves the performance of the embedded systems and optimize the design in terms of various attributes. In general, hardware implementation of a function has better performance than software implementation. Thus moving selected software region that requires significant computation cycles to configurable hardware can improve the performance of the whole system. However, there is no generally accepted methodology to separate applications onto hardware and software execution. Moreover, the increasing heterogeneous nature of embedded system platform and application makes the mapping of tasks onto different set of processors very complicated. One of the major requirement for such traditional partitioning process is to identify which application or part of the application can be implemented onto reconfigurable hardware. In the heterogeneous system context, this problem becomes more complicated as partition has to be further extended to support heterogeneous architecture which combines different architectures. Design Space Exploration of embedded system allows rapid performance evaluation of different design parameters, application to processor mapping and hardware/software partitioning. One of the way of selecting the critical software region for hardware/software partitioning is to use program analysis tools like profiling and tracing. Profiling and tracing provide necessary information to collect and analyze the program behavior which helps to pinpoint the performance or memory hotspot in the program and identify the candidate kernels for mapping into different set of processors. These functions can further be optimized in iterative manner to get the best candidate for hardware implementation. Towards this goal, the first step is to investigate the application based on various design criteria such as execution time, bandwidth, power consumption etc for optimization and design space exploration.

The organization of the paper is the following: Section II presents the system context and the problem definition. Section III discusses the related work. Section IV presents the dynamic profiling approach. Section V presents the methodology for measuring execution time and bandwidth with an example. And, finally section VI gives summary of the paper and future work.

## II. SYSTEM CONTEXT

Modern computing systems make increasing use of reconfigurable and heterogeneous architectures. The increasing heterogeneous nature of platform and the application makes the design of embedded system very complicated.

Due to its heterogeneous nature, the traditional design methodologies are not sufficiently suited for these emerging systems and developers need in-depth knowledge of both hardware and software in order to create an effective design of such system. In hardware/software co-design environment, to make early design decision, it is necessary to perform exhaustive design space exploration (DSE). DSE demands an exploration of a huge design space in a short period of time. For this it is essential to prune the design space using fast methods, which further can be refined precisely to a reduced design space. For making such early design decisions, it is very essential to identify the various design parameters such as execution time, memory, bandwidth etc. The measurement of these parameters can help reduce the design space exploration by determining whether kernels can be implemented efficiently on hardware. Profiling is one of the such techniques to measure these design parameter to quickly find promising candidate for mapping onto heterogeneous set of processors and thereby reduce the points in the design space that needs to be explored. The Delft Workbench [1] addresses optimal and rapid design of embedded systems from high-level descriptions, targeting a heterogeneous platform with combination of embedded processors, digital signal processing, reconfigurable hardware etc. The Delft Work Bench research project also supports developers during DSE phase.

The Delft Work Bench uses profiling and cost estimation techniques for design space exploration, hardware/software partitining and optimization. In Delft Work Bench context, we analyze the program statically and dynamically in order to determine relevant profiling information[2]. The dynamic analysis captures the dynamic behavior of the application and provides the measure of various design parameter such as execution time, bandwidth utilization, while with static analysis we make early estimates of hardware utilization, such as area, delay and bandwidth. This information provides necessary information on deciding which kernels to map onto hardware. In heterogeneous context, these information provides vital information for which kernels can be mapped onto particular processor set. Within this context, in this paper, we present a methodology for the dynamic analysis of the application for design space exploration, hardware/software partitioning and parallelization within Delft Workbench.

## III. RELATED WORK

The goal of profiling is to analyze the program statically and/or dynamically in order to determine relevant information for design exploration, hardware/software partitioning and parallelization. Multiple profiling techniques are de-veloped to analyze the input application behavior for different criteria such as performance, memory, power consumption etc.

The static code analysis performs the analysis of the source code without actually executing programs while dynamic analysis performs the analysis during program execution. Static analysis is less accurate than dynamic profiling as it is based on estimation of branch prediction execution, while dynamic profiling is slow and requires programmer intervention. For runtime profiling, the most common methods are code instrumentation [3][4], sampling [5][6],[6], or hardware profiling [7] [8], [9] and [10]. Instrumentation gives precise information than sampling while they have several overhead associated and results obtained may be skewed by inserted hooks. Sampling works with unmodified code but needs enough samples to be accurate. Researches have proposed several algorithms for program profiling and tracing such as basic block profiling, control flow - edge and path profiling [11][12][13], value profiling [14] etc at the same time there are various profiling tools those use these algorithm. Each of these tools has its own advantages and disadvantages in terms of speed, accuracy and overhead. Based on the requirement of the application to be profiled, it is necessary to modify these tools and make tradeoff between the offered criteria.

In Delft Work Bench, we combine both static and dynamic approaches to develop an efficient profiling tool in terms of accuracy and speed for hardware/software partitioning. During the profiling phase, relevant execution and data storage information is collected. In addition, preliminary estimations of the hardware and the software costs of the application's kernels are performed. The static estimation for Delft Work Bench is measured with a quantitative model and presented in the paper [15].

## IV. DYNAMIC PROFILING

The usual goal of dynamic code analysis is to determine which parts of a program should be optimized for speed or memory usage. Dynamic profiling is a performance analysis technique that measures the behavior of a program as it runs, particularly measuring the frequency and duration of function calls. The output is a stream of recorded events (a trace) or a statistical summary of the events observed (a profile). Dynamic profiling involves obtaining the execution time, memory usage etc of the program basically within the software code level (at function level, statement level or loop level) and analyzing them to get an idea of where the processor is spending most of its time or where program is using most of the bandwidth. Dynamic analysis

is performed with executing programs. This investigates the program's behavior using information gathered as the program runs.

The dynamic analysis captures the dynamic behavior of the application and provides the measure of various design parameter such as computation requirement and bandwidth requirement of the systems. In heterogeneous context, for making early design decision such as mapping of tasks or set of tasks onto multiple set of processors, it is also essential to know these parameter. Towards this goal, the first step is to measure computation requirement and the communication requirement of the application. The computation requirement is a measure of number of CPU cycles required to execute a particular task. And, the communication requirements is the measure of the amount of data that is transferred between different tasks.

## V. METHODOLOGY

The increasing heterogeneous nature of platform and the application makes the design of embedded system very complicated. For making early design decisions such as mapping of application into heterogeneous set of processors and hardware/software partitioning, it is very essential to identify the various design parameters such as execution time, memory, bandwidth etc. To measure such design parameters, it is necessary to capture the dynamic behavior of the system at run time in order to know the dependency between the tasks and the resources used by these tasks in the system. We want to use task graph for representing the dynamic behavior of the program in order to measure such parameters of the application. For this we want to represent the application as network of interconnected tasks where nodes in the graph represents the tasks and the interaction between these tasks are represented by the edges in the graph.

The application can be represented by a graph G = (V, E) where vertex V represents the tasks and the edge E represents the communication between tasks. Tasks can contain one or more functional blocks (such as blocks or loops). A task in the graph communicates with other task passing data through the communication channel between them. All the vertexes and edges are given some weighted value. This value is the measured value of the execution time of the task and the channel bandwidth between the tasks. For example in figure 1, parameters a, b, c, d, e represents the execution time of each tasks A, B, C, D and E respectively and parameters x, y, z, u, v, w represents the data transfer between the tasks. We call this graph a weighted task graph.
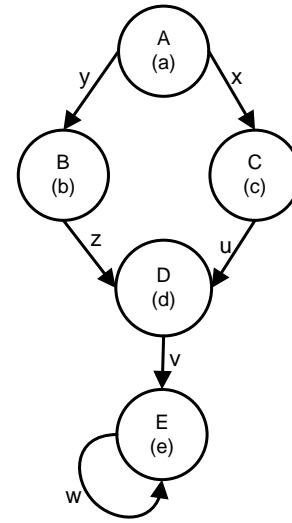


Fig. 1
WEIGHTED TASK GRAPH

There are two types of requirements of the application - communication and computation. The communication requirements is the measure of the amount of data that is transferred between two tasks. The computation requirement is a measure of number of CPU cycles required to execute a task. In very simple form, the problem of mapping of tasks into heterogeneous set of processors can be expressed as a function in terms of communication and computation requirement of the application. If F(x) is the mapping function then it can be represented as

```
F(x) = f (x) . g(x)
```

where f(x) represents the computation time of the tasks and g(x) represents the data transfer between two tasks. The obvious goal is to gain the maximum execution time and to minimize bandwidth between different tasks. If two tasks are mapped to different processors we want to minimize the communication between those tasks and similarly if there is a significant communication between two tasks we want to map them to same processor. We call the tasks tightly coupled if two tasks have considerable amount of data transfer between them. And, tasks are loosely coupled if there is less communication between them.

### A. Measuring Computation Time

As a first design parameter for system design, we want to measure the execution time of the tasks. The computation time is the time taken to execute a task. It can be represented as a time in second or percentage of execution time. There are various techniques available that can be used to measure the execution time of the tasks.

## B. Measuring Bandwidth

Another important parameter for system design in bandwidth. For measuring bandwidth, we want to measure the memory dependencies between different task and the amount of data transfer between them. Sharing of data involves communication, if there is sharing of data between two tasks then there is some some kind of sequential communication or sequential data transfer between these tasks. So as a first step of measuring bandwidth between two task, we want to know the dependency between two tasks. Knowing the dependency between the tasks, we can measure the data transfer between these tasks.

At the first step toward measuring data transfer between different tasks we want to distinguish separate tasks of the applications. The granularity of the task can be at function level or part of a function. At second step we want to know what data structures communicate data from one task to another and what structures are private to a task. We want to record the memory trace for every task of the application. The traces for a data structure can be grouped together to a logical address. We record the trace of all the task that accesses same logical address.

There are two different kinds of task events: memory read and memory write. For each instance of time for all the data structure, we record each event when there is a memory read and memory write in a table like structure. The table contains the task which writes to the address and how many bytes is written similarly which task reads the data and how many bytes are read. With this information, we create a tuple (Tw, Tr, Bytes) where Tw is the task which writes data, Tr is the task which reads data and Bytes is the number of bytes/bits read or written to certain logical address. This information can be used to construct weighted task graph. The weighted task graph in figure 3 is constructed from the tuples in table I.

In this case, we assume every node produces or consume tokens. We ignore the inter node communication and we just consider there is communication between two nodes only. In more formal way, we can measure this communication data in matrix like form. The cell (i,j) of the matrix represents the amount of data produced or consumed by node i with node j. If node i produces the data then it is represented by positive number while if the node consumes data from another node then it is represented by negative number. If there is no relation between two nodes then the value is zero. If a node has a connection to itself (self loop) then we assume the data produced by the node is consumed by the same node so we represent it just as a positive number.

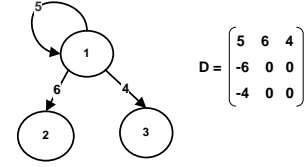ber. The matrix representation example for the given graph is in figure 2.



Fig. 2
MATRIX REPRESENTATION

| Write | Read | No. of Tokens | Tuple |
|-------|------|---------------|-------|
| T1 | T1 | 1 | |
| T1 | T1 | 2 | (T1, T1, 3) |
| T1 | T2 | 2 | |
| T1 | T2 | 1 | (T1, T2, 3) |
| T2 | T4 | 1 | (T2, T4, 1) |
| T2 | T5 | 1 | |
| T2 | T5 | 2 | (T2, T5, 3) |
| T1 | T3 | 2 | |
| T1 | T3 | 2 | (T1, T3, 4) |
| T3 | T4 | 2 | (T3, T4, 2) |
| T3 | T5 | 2 | (T3, T5, 2) |
| T4 | T5 | 1 | (T4, T5, 1) |

TABLE I
EXAMPLE TRACE OF THE MEMORY ACCESS

Using this weighted task graph we quantify the dynamic behavior of the system measuring various parameters such as,
• Task Run Graph - This gives the timeline at what instances of time which tasks is actively running. This will capture the dynamic behavior of the task in the system.
• Bandwidth Utilization - This gives the timeline at what instance of time which communication channel is occupied and how much bandwidth is used.

The bandwidth utilization of the system gives the minimum/maximum bandwidth gain of requirement of the system when particular set of tasks are mapped onto particular processor. Similarly, knowing the dependency between the tasks and the resource(in this example bandwidth) used by these tasks, we can pipeline/parallelize these tasks for faster execution. For instance at time Ts if there is sufficient bandwidth available, we can create multiple instances of the tasks and execute them simultaneously in multi-threaded way in order to make the execution faster.
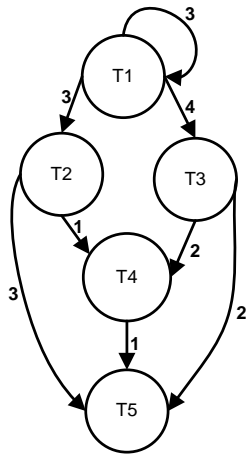
Fig. 3

WEIGHTED GRAPH CONSTRUCTED FROM THE TUPLES IN
TABLE I



Fig. 4

EXAMPLE WEIGHTED GRAPH OF MPEG2 WITH HOT-SPOT
HIGHLIGHTED

## C. Example

The figure 4 gives the example weighted graph for part of
MPEG2 application. In the figure the hot-spots are high-
lighted within a circle. The example shows both the mea-
sure of percentage of computation time of each tasks and
data transfer between them. Based on the measured value
of percentage of computation time a and the number of
byte transferred between the tasks, the application can be
mapped onto one or more processors. For instance, in or-
der to optimize the bandwidth and reduce inter task com-
munication `fullsearch` and `dist1` can be mapped
onto the same processor. There is a significant data trans-
fer between `fullsearch` and `dist1` and these tasks are
tightly coupled. In heterogeneous context, in order to re-
duce the inter process communication, the tightly coupled
tasks should be mapped onto same processors. Similarly,
`fdct`, `transform` and `sub_pred` can be mapped to the
same processors.

## VI. SUMMARY AND FUTURE RESEARCH

Modern computing systems make increasing use of re-
configurable and heterogeneous architectures. In hetero-
geneous environment, for making early design decisions
such as mapping of application into heterogeneous set of
processors and hardware/software partitioning, it is very
essential to identify the various design parameters such as
execution time, memory, bandwidth etc. Profiling is one
of the such techniques to measure these design parame-
ter to quickly find promising candidate for mapping onto
heterogeneous set of processors and provide relevant in-
formation for design space exploration. Towards this goal,
in this paper we presented the methodology for dynamic
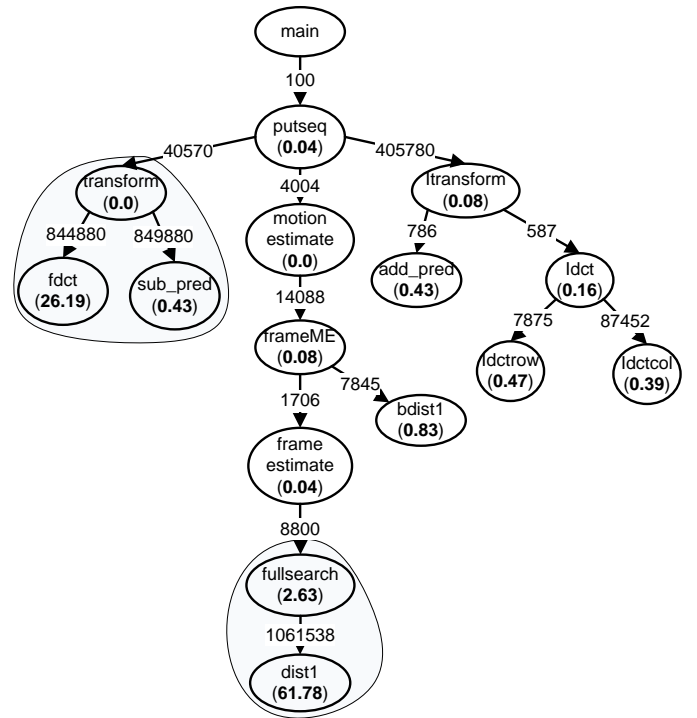analysis of the application for design space exploration,
hardware/software partitioning and parallelization within
Delft Workbench. The technique presented in the paper
expresses the dynamic behavior of the application in terms
of weighted task graphs and measures a application spe-
cific design parameter such as execution time and band-
width via a dynamic profiling mechanism. In this paper we
presented the outlook of the profiling method with exam-
ple data, in future this methodology will be implemented
within Delft Work Bench framework and real parameters
will be measured.

## REFERENCES

[1] "Delft workbench," Online: http://ce.et.tudelft.nl/DWB/.
[2] K. Bertels, G. Kuzmanov, E. M. Panainte, G. N. Gaydadjiev,
Y. D. Yankova, V. Sima, K. Sigdel, R. J. Meeuws, and S. Vassil-
iadis, "Profiling, compilation, and hdl generation within the hartes
project," in *FPGAs and Reconfigurable Systems: Adaptive Het-
erogeneous Systems-on-Chip and European Dimensions (DATE
07 Workshop)*, April 2007, pp. 53–62.
[3] A. Srivastava and A. Eustace, "Atom: a system for building cus-
tomized program analysis tools," *SIGPLAN Not.*, vol. 39, no. 4,
pp. 528–539, 2004.
[4] "http://valgrind.org/."
[5] S. L. Graham, P. B. Kessler, and M. K. McKusick, "gprof: a call
graph execution profiler," in *SIGPLAN Symposium on Compiler
Construction*, 1982, pp. 120–126.
[6] D. W. Wall, "Systems for late code modification," in *Code Gener-
ation - Concepts, Tools, Techniques*, R. Giegerich and S. L. Gra-
ham, Eds. Springer-Verlag, 1992, pp. 275–293.

[7] R. V. Peri, S. Jinturker, and L. Fajardo, "A novel technique for profiling programs in embedded systems," in *Second ACM Workshop on Feedback-Directed and Dynamic Optimization.*

[8] M. C. Merten, A. R. Trick, E. M. Nystrom, R. D. Barnes, and W. mei W. Hwu, "A hardware mechanism for dynamic extraction and relayout of program hot spots," in *ISCA*, 2000, pp. 59–70.

[9] M. C. Merten, A. R. Trick, C. N. George, J. C. Gyllenhaal, and W. mei W. Hwu, "A hardware-driven profiling scheme for identifying program hot spots to support runtime optimization," in *ISCA*, 1999, pp. 136–147.

[10] S. Narayanasamy, T. Sherwood, S. Sair, B. Calder, and G. Varghese, "Catching accurate profiles in hardware," in *9th International Symposium 184 on High-Performance Computer Architecture*, February 2003, pp. 269–280.

[11] T. Ball and J. R. Larus, "Optimally profiling and tracing programs," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 4, pp. 1319–1360, July 1994.

[12] ——, "Efficient path profiling," in *International Symposium on Microarchitecture*, 1996, pp. 46–57.

[13] T. Ball, P. Mataga, and M. Sagiv, "Edge profiling versus path profiling: the showdown," in *POPL '98: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages.* New York, NY, USA: ACM Press, 1998, pp. 134–148.

[14] B. Calder, P. Feller, and A. Eustace, "Value profiling," in *MICRO 30: Proceedings of the 30th annual ACM/IEEE international symposium on Microarchitecture.* Washington, DC, USA: IEEE Computer Society, 1997, pp. 259–269.

[15] R. J. Meeuws, Y. D. Yankova, K. Bertels, G. N. Gaydadjiev, and S. Vassiliadis, "A quantitative prediction model for hardware/software partitioning," in *Proceedings of 17th International Conference on Field Programmable Logic and Applications (FPL07)*, August 2007, p. 5.