Performance Analysis for Hardware/Software Co-synthesis

[1]Imed E. Bennour†, Michel Langevin†† and El M. Aboulhamid†

†Dép. d'I.R.O, Université de Montréal, CP 6128, Centre Ville, H3C-3J7 Montréal, PQ, CANADA
††GMD-SET, Schloβ Birlinghoven, D-53757 Sankt Augustin, GERMANY

***Abstract*— This paper presents a method for estimating the extreme case bounds (upper-bound and lower-bound) on the running time of a source program on a target hardware architecture. The source program may be any block of code containing data processing and control statements. The target architecture is specified by a set of functional units, a set of storage units and an interconnection network.**

I. INTRODUCTION

Hardware/Software (HW/SW) co-synthesis [7] refers to the design approach which mixes hardware and software implementations of complex numerical systems in order to reduce the design cost while satisfying the performance requirements. A hardware implementation has better performance, whereas a software implementation has lower cost and allows later modifications. Figure 1 shows the general process of HW/SW co-synthesis. One of the key task in this process is the partitioning of the original system description into hardware and software modules. Hardware modules will be implemented as a dedicated hardware, while software modules will be implemented as programs running on predesigned general-purpose processor. The decision to map functionalities into software or hardware parts is based on estimates of achievable performance and the implementation cost of the respective parts. An automatic tool to estimate the performances of hardware and software implementations (tasks A and B in Figure 1) enables the designer to quickly evaluate different design alternatives. This paper presents a method for estimating the upper-bound and the lower-bound on the performance of a control-data flow graph (CDFG) on a target hardware architecture. The target architecture is defined by a set of functional units (e.g., ALUs, multipliers), a set of storage units (RAMs, register-files), and an interconnection network. t_{LB} and t_{UB} are extreme case bounds on the performance of a CDFG, if for any input data the running time of the program belongs to $[t_{LB}, t_{UB}]$. Tight values of t_{LB} and t_{UB} not only give a good estimation of the achievable performance but they also permit to check if performance requirements can be satisfied by a target architecture. Let t_{max} denotes the maximal allowed performance of a CDFG. If $t_{max} \geq t_{UB}$, it means that for any input data the performance constraint will be satisfied. If $t_{max} < t_{LB}$, it means that the performance constraint cannot be met by the target architecture.

[1]This work has been partially sponsored by the "Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR)".

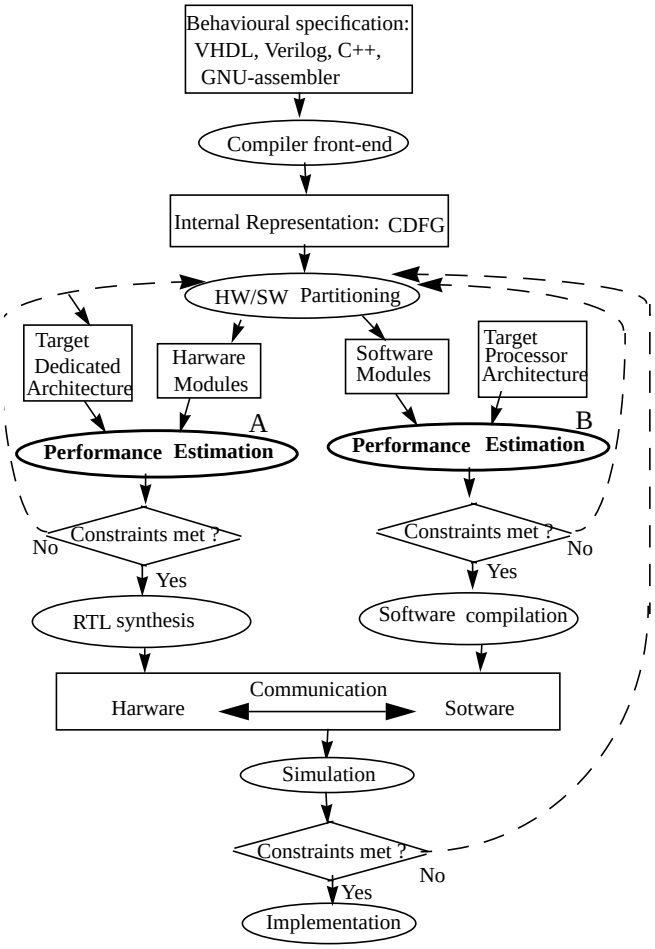

Figure 1. General framework of HW/SW co-synthesis.

The rest of this paper is organized as follows. Section 2 describes the estimation model used, and Section 3 presents the estimation method. Experimental results are given in Section 4.

II. MODEL OF ESTIMATION

There are two possible ways to estimate the extreme case bounds on the performance of a program on a target architecture: dynamic simulation and static estimation. Dynamic simulations consist of simulating the program execution with different input data, whereas static estimations are based on the analysis of the program structure. Static estimations are faster than dynamic simulations and insensitive to input data. The estimation

approach presented in this work is static.

A. The hardware architecture model

Development of an estimation technique requires the definition of the target architecture model. In this work, we use a parametrized bus-based architecture as target model. Figure 2 shows an instance of the parametrized bus architecture. The parameters defines:

– the number of functional units of each type, their speed and their pipeline-stage number;
- the number of memory ports;
- the number of register-file ports;
- the number of buses;
- and the connections between components.

The choice of the parametrized bus architecture was influenced by the following reasons:
1) It enables the specification of a wide variety of target architectures, ranging from completely sequential to massively parallel architecture;
2) The bus architecture is frequently used in microprocessor systems;
3) Most high level synthesis systems use bus-based architectures.

B. The CDFG model

A CDFG [3] is a graphical representation of source programs. A CDFG is a composed of two type of graphs: the control flow graph (CFG) and the data flow graph (DFG). A CFG is defined by a couple (BB, E), where BB is the set of nodes representing basic blocks, and E is the set of edges representing precedence execution order between basic blocks. To each basic block bb_i is associated a DFG (O_i, A_i), where O_i is the set of nodes representing atomic operations such additions and multiplications, and A_i is the set of arcs representing precedences between atomic operations. An execution of the CDFG consists of a sequential execution of basic blocks. A CDFG example is shown in Figure 3. Feed back edges in the CFG represent loop statements. We assume that each loop statement has a bounded number of iterations, otherwise the worst running time cannot be computed in general; it is well known that if a program contains unbounded loop statements then it is not possible to decide if its execution terminate. The maximal numbers of loop iterations are annotated in the source code by the user. The user can also specify the minimal numbers of loop iterations; default values are equal to zero. A *false-path* is a path in the CFG which is never executed due to incompatibility of two or more conditional branching. For example, if the conditions c_2 and c_3, in Figure 3, cannot be true at the same time, then the sequence of basic blocs $bb_d bb_f bb_g$ is never executed.

III. EXTREME CASE PERFORMANCE BOUNDS

A. Upper-bound estimation

The proposed technique for determining an upper bound on the performance of a CDFG is composed of two steps: (1) determining an upper bound on the performance of each basic block, and (2) deducing an upper-bound for the whole CDFG.

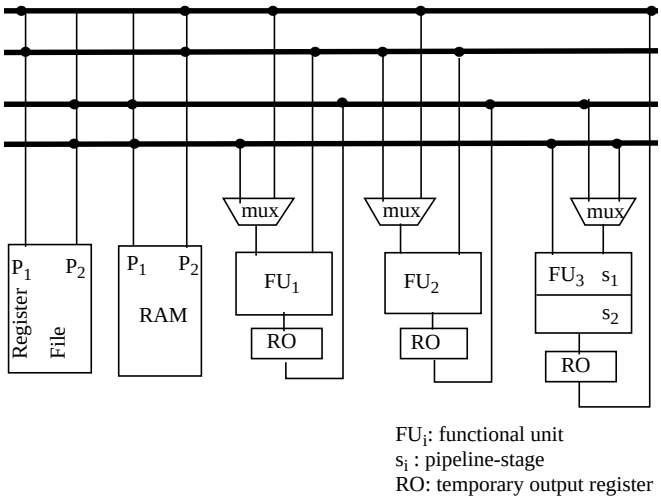

Figure 2. An instance of the parametrized bus-based architecture.

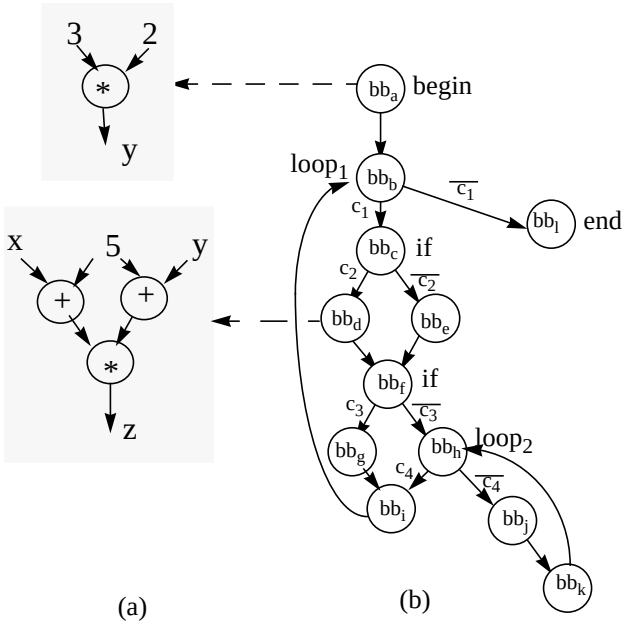

Figure 3. A CDFG example: (a) DFG, (b) CFG.

A.1 Upper-bound on the performance of a basic bloc

The running time of a basic block is equal to the time necessary to execute the operations in the corresponding sub-DFG. An Upper-bound on the performance of a basic block can be obtained by scheduling the corresponding DFG on the target hardware architecture. As we use a parametrized bus-based architecture as a target model, we developed a greedy scheduling algorithm similar to the one in [2].

A.2 Upper-bound on the performance of a CDFG

We define the length of a path in a CFG as the sum of the running times of basic blocks forming this path. An upper-bound on the performance of a CFG will be the length of the longest path when considering that (1) the running times of

basic blocks are equal to their upper-bound values, and (2) each loop is executed a maximal number of times.

One way of determining the length of the longest path in a CFG consists of first completely unfolding all loops and then computing the length of the longest one in the unfolded graph. The disadvantage of this method is that size of the unfolded graph will be very large if the loop-iteration numbers are high. The method that we use does not require loop unfolding. This method is described by the algorithm given in Figure 4.

Following is an illustration of this algorithm with the CFG of Figure 5(a). The number beside a node is the performance upper-bound of the corresponding basic block, while the number beside a feed-back edge is the maximal iteration number of the corresponding loop. *loop*$_2$ is the most inner loop, which is executed at most 20 times and each iteration takes (7+15+2) cycles. Thus, the running time of this loop will take at most 480 clock-cycles. In Figure 5(b) *loop*$_2$ is replaced by a single node with a weight equal to 480. After repeating the same process with *loop*$_1$ we obtain the graph of figure 5(c). We deduce that 5002 clock-cycles is an upper bound on the performance.

Notice that tighter upper-bound on the performance of a CDFG can be obtained if optimal DFG schedules are computed and/or if only CFG feasible-paths are considered. Unfortunately, each of these problems is NP-complete. An integer linear programming formulation was presented in [1], where some false-paths of the CFG could be avoided during the estimation.

Input: a CDFG, an upper-bound on the performance of each basic block, a maximal number of iterations of each loop;
Output: upper-bound on the performance of the CDFG;

Begin
-Associate to each basic block in the CFG a weight equal to its performance upper bound;
Repeat:
For all inner-most loops **Do**:
- Compute the length (*l*) of the longest path in the current loop body;
- Replace in the CFG the loop by a single node;
- Associate to this node a weight equal to (*l* * the maximal number of iterations of the loop);

End Do
Until no more loops in the CFG
- Compute the length of the longest path in the resultant graph. Return this length as an upper bound on the performance of the CDFG;

End

Figure 4. An algorithm for determining an upper-bound on the performance of a CDFG.

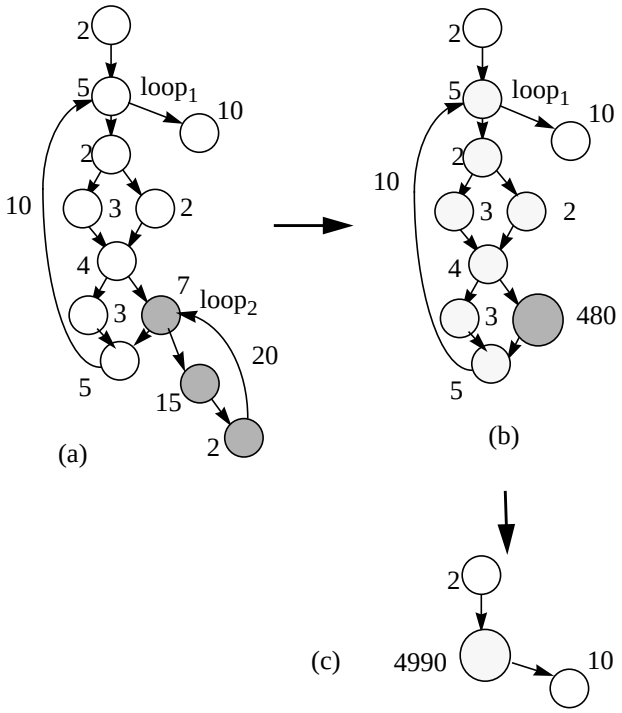

Figure 5. Illustration of the performance upper-bound algorithm.

B. Lower-bound estimation

A lower-bound on the running time of a CDFG can be derived from a lower-bound on the performance of each basic block and from a minimal number of iterations for each loop. The lower-bound will be the length of the shortest path in the CFG when considering (1) that the running times of basic blocks are equal to their lower-bound values, and (2) that each loop is executed a minimal number of times. The algorithm for computing the length of the shortest path is similar to the one used for computing the length of the longest path (Figure 4), except that for each loop we consider its shortest running time instead of longest running time.

A lower-bound on the performance of a basic block can be the length of the longest path in its DFG where the weights of nodes are set to operation-durations. Tighter lower-bounds can be obtained by taking into account resource constraints. Several algorithms have been proposed in the literature [4-6] for determining a lower-bound on the performance of a basic block under resource constraints. Currently we have implemented the algorithm presented in [5].

IV. EXPERIMENTAL RESULTS AND CONCLUSIONS

In Tables 1 and 2, we report part of the experimental results done with two benchmarks: square matrixes multiplication and matrix convolution. The first and the second benchmark contain three and two nested loops, respectively. In both benchmarks, nested loops have the same iteration number which is equal to matrix dimension. The source codes of these benchmarks were translated in GNU-assembler before analysis. The running time of the tool was

in order of seconds. The first column in Tables 1 and 2 indicates the iteration bounds of loops, and the last two column indicate the extreme case bounds on the performance under the resource constraints. From Table 1, we deduce that the performance of the first benchmark decreases by using more ALUs and/or register file ports. However, more than 3 ALU and/or 3 register file ports does not reduce the worst performance any more. We can deduce also that RAM ports are not critical resources. Results in Table 2 shows that all resource type are critical, and more than 4 ALU and/or 3 memory ports and/or 2 register file ports does not reduce the worst performance any more.

The estimation tool was developed in C++ using the co-synthesis SIR/CASTLE (Codesign and Architecture-driven Synthesis TooL Environment) database [7].

Table 1: Matrixes multiplication

# of Iterations of each loop	# of ALU	# of RAM ports	# of regis. file ports	Perform. lower bound (clock cycles)	Perform. upper bound (clock cycles)
50	1	1	1	$1.015\ 10^6$	$1.283\ 10^6$
	1	1	2	-	$1.150\ 10^6$
	1	1	3	-	$1.147\ 10^6$
	1	2	1	-	$1.283\ 10^6$
	1	2	1	-	$1.283\ 10^6$
	2	1	1	$0.510\ 10^6$	$1.280\ 10^6$
	2	1	2	-	$0.770\ 10^6$
	2	1	3	-	$0.767\ 10^6$
	2	2	1	-	$1.280\ 10^6$
	3	1	3	$0.385\ 10^6$	$0.642\ 10^6$
	3	1	4	-	-
	4	1	3	-	-
500	1	1	1	$1.001\ 10^9$	$1.250\ 10^9$
	1	1	2	-	$1.121\ 10^9$
	2	1	1	$5.011\ 10^8$	$1.253\ 10^9$
	2	1	2	-	$7.521\ 10^8$
	3	1	1	$3.760\ 10^8$	$1.253\ 10^9$
	3	1	2	-	$7.520\ 10^8$

Table 2: Matrix convolution

# of Iterations of each loop	# of ALUs	# of RAM Ports	# of RF Ports	Lower-bound (clock cycles)	Upper Bound (clock cycles)
50	1	1	1	$7.941\ 10^4$	$8.211\ 10^4$
	1	1	2	-	$8.206\ 10^4$
	2	1	1	$4.111\ 10^4$	$5.141\ 10^4$
	2	1	2	-	$4.381\ 10^4$
	2	2	1	$4.100\ 10^4$	$5.136\ 10^4$
	2	2	2	-	$4.371\ 10^4$
	3	2	1	$3.080\ 10^4$	$5.126\ 10^4$
	3	3	1	$2.820\ 10^4$	$5.126\ 10^4$
	3	3	2	-	$3.095\ 10^4$
	3	3	3	-	-
	4	3	2	-	$2.835\ 10^4$
	5	3	2	-	-
500	1	1	1	$7.769\ 10^6$	$8.021\ 10^6$
	2	1	1	$4.011\ 10^6$	$5.014\ 10^6$
	1	2	1	-	$8.211\ 10^4$
	2	1	1	$4.111\ 10^4$	$5.141\ 10^4$

References

[1] Y.-T. S. Li, S. Malik, "Performance Analysis of Embedded Software Using Implicit Path Enumeration", 32nd *Design Automation Conference*, 1995.

[2] J. Gong, D. D. Gajski, A. Nicolau, "A Performance Evaluator for Parameterized ASIC Architectures", *European Design Automation Conference*, 1994.

[3] D. D. Gajski, N. Dutt, A. Wu, S. Lin, "*HIGH-LEVEL-SYNTHESIS- Introduction to Chip and System Design*", Kluwer Academic Publishers, Boston, 1992.

[4] A. Sharma, R. Jain, "Estimation architectural resources and performance for high-level synthesis applications", IEEE Trans. on VLSI, Vol. 1, No. 2, pp. 175-190, 1993.

[5] M. Rim, R. Jain, "Lower-bound performance estimation for the high-level synthesis scheduling problem" *IEEE Trans. on CAD*, Vol. 13, pp. 81-88, 1994.

[6] M. Langevin, E. Cerny, "A recursive technique for computing lower-bound performance of schedules" *International Conference on Computer Design*, 1993.

[7] R. Camposano, J. Wilberg, "Embedded System Design", *Design Automation for Embedded Systems*, Vol. 1, pp. 5-50, 1996.