Fast Processor Core Selection for WLAN Modem using Mappability Estimation

Juha-Pekka Soininen, Jari Kreku, Yang Qu and Martti Forsell
VTT Electronics
PO. Box 1100 (Kaitoväylä 1), FIN-90571 Oulu, FINLAND
Tel: +358 8 551 2111
e-mail: Juha-Pekka.Soininen@vtt.fi

ABSTRACT

Mappability metric and a novel method for evaluating the goodness of processor core and algorithm combinations are introduced. The new mappability concept is an addition to performance and cost metrics used in existing codesign and system synthesis approaches. The mappability estimation is based on the analysis of the correlation or similarity of algorithm and core architecture characteristics. It allows fast design space exploration of core architectures and mappings with little modeling effort. The method is demonstrated by analyzing suitable processor core architectures for baseband algorithms of the WLAN modem. 140400 architecture-algorithm pairs were analyzed in total and the estimated results were similar to the results of more detailed evaluations. The method is not, however, limited to the WLAN modem, but is applicable for digital signal processing in general.

Keywords

mappability estimation, processor architecture evaluation, codesign, cost function

1. INTRODUCTION

Complete computer networks can be integrated into a single ASIC within the near future when the silicon capacity exceeds the billion-transistor mark [1]. It will also be possible to develop mobile terminals that combine the functionality of communication systems, multimedia appliances, audio and video systems, and computers. Personalized services, privacy, portability and mobility are features that require highly integrated power and performance optimized solutions. A set of applications can consist of tens or even hundreds of programs or functions. The processing can be bit-stream or data block oriented, database or data storage oriented, or real-time responses to sporadic external events. The overall load can consist of several simultaneous or parallel activities.

Both multimedia and wireless communication applications are difficult to implement energy efficiently. Dedicated solutions are required instead of general-purpose platforms. Heterogeneous multicomputer systems, where the architecture of each computer and processor and respective computation load are considered together, are viable alternatives when flexibility and variability is combined with performance and efficiency.

The design of an integrated multicomputer system will extend the problems faced with current system-on-chip (SoC) design with system level and computer architecture design problems [2, 3]. Performance design with analyses, simulations and measurements have been studied and used in computer architecture design [4]. In the codesign, the hardware complexity- and software performance estimation-based cost functions have been used [5].

The efficient reuse and high utilization of the platform's resources will be the success factors when we are designing and using an ASIC with 10-100 computers. Reuse at all levels is needed in order to manage the overall complexity. Efficient use of resources, resource sharing and programmability results in fewer overheads in terms of design effort, energy usage and cost. The key design problems will be the scaling of the computation, storage and communication capacities, mapping of functionality and resources, and the validation of quality characteristics such as performance, energy efficiency and resource utilization of both the subsystems and the complete network.

In this paper we present a mappability estimation method for evaluating the goodness of processor architecture and algorithm pair. By mappability of an architecture-algorithm pair we mean the degree of matching between resources provided by the processor architecture and requirements described by the algorithm. The perfect match would mean exactly the right number of resources and an optimal pipeline organization for the algorithm giving maximal utilization of functional units, instruction set and memory interface, so that increasing the number of any resources would only decrease the utilization respectively.

Mappability estimation method is targeted for the identification of good processor core and algorithm mappings. This problem must be solved in multicomputer SoC design flow before the final processors or algorithms are available, because in ASIC implementation it is also possible to optimize the system by changing the processor architecture or instruction set. Configurable processor cores are good examples of this new possibility. So, the mappability estimation can be used either for the evaluation or selection of the processor-algorithm pair by using models of existing processor cores like TMS320C6201 or ADSP-2181 [6], or for identification of "optimal" architecture to be implemented as a customizable RISC or VLIW core or as a custom processor. This paper discusses the latter approach.

The analysis is based on comparing the characteristics of algorithms and processor cores instead of mimicking the execution. This allows us to use simple models of both processors and algorithms and results to extremely rapidly analyze a large number of alternatives. Fast reduction of design space is the main benefit compared to traditional approaches.

The main limitation of our approach is that the effects of memory organization, cache and communication are not yet considered because they would require more detailed views of partitioning, compilation and application programs. That would increase the complexity of estimation significantly. The absolute performance of a processor is not yet taken into account in our approach. The designer has to check explicitly that the processor is capable of doing the computation within the specified time frame.

The paper is organized as follows: Section 2 gives an overview of related work; Section 3 introduces our approach; Section 4 describes the experimental results achieved with WLAN algorithms and 10800 processor core models; Section 5 concludes the paper.

2. RELATED WORK

The modern processor cores actively try to speed up system performance by using parallelism and overlapping. This makes the processor architecture design and evaluation a challenging task. In the quantitative approach [4] the information to be evaluated, e.g. execution time, Amdahl's law and locality of reference, is extracted from the execution or simulation of a set of benchmark programs. In the analytical approach the key metrics of the goodness of the processor's architecture, e.g. execution time of a parametric "average" program, power consumption or silicon area, are modeled by analytical equations [7]. The main principles of the physical approach are extracted from electronics and silicon engineering: locality of communication, asynchrony of operations, regularity of structure and overall simplicity [8]. Existing processors are typically compared using benchmark or implementations programs. Analytical performance modeling and simulation are applicable at earlier stages of design only because of the complexity problems [9].

In software estimation-based approaches the idea is to analyze the amount of required computation and to estimate the execution time with the used processor. The number of instructions can be derived from an intermediate representation of a program [10], specification language [11] or source code [12]. The execution time can be based on instruction or function level timing models. In the retargetable estimation approach [13] the parameterized architecture model allows study of the potential of the architecture. The results are more independent of the compiler's ability to optimize the code. In execution time estimation approaches the potential for executing the algorithm even more efficiently than the processor architecture allows is not analyzed as it is in our approach. In the S^3E^2S environment the processor selection is based on the comparison of algorithm properties, e.g. relative importance of memory accesses, data transformations and control instructions to the core's characteristics [14]. In our approach both algorithm and core architecture models are more detailed and, therefore, the mappability correlation function is more comprehensive.

3. ARCHITECTURE-ALGORITHM MAPPABILITY

The figure of merit proposed for architecture-algorithm pair is mappability, e.g. $M = (c, a)$, where c is core architecture and a is algorithm. The mappability is optimal when the hardware architecture does not constrain the execution and it does not have any unused capacity, as depicted in Figure 1. In an ideal case all the hardware in the core is participating in the execution of code all the time, but additional gates cannot be used because of the nature of the code.

Instruction capability, instruction execution speed and dynamic parallelism are core characteristics that affect performance. Instruction capability depends on instruction set, internal registers and memory interface. Execution speed depends on superpipelining degree, pipelining support and instruction types. Dynamic parallelism depends on execution architecture and the core's capability to utilize the potential in the algorithm's control flow.

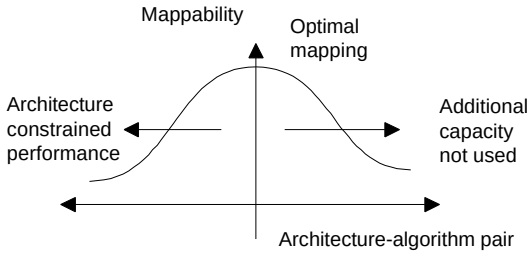

Figure 1. Mappability function.

The computation constraints from the algorithm are caused by its control flow structure and data dependencies. The control flow determines the executed program paths. It depends on input data or events. Data dependencies constrain the order and the amount of parallelism in which the operations can be executed.

The Core-Algorithm Mappability Analysis Approach (CAMALA) extracts essential characteristics of the core and algorithm and analyses how much correlation exists between the algorithm's computation requirements, e.g. data dependencies and control flow structure, and core properties, e.g. instruction set, execution architecture and memory interface. The algorithm characterization is based on compilation, profiling and bound checking. The result is our intermediate representation graph. The core model consists of the instruction models described using weighted data dependency graphs, and interface and architecture parameters.

3.1 MODELING AND CHARACTERIZATION

The algorithm a can be modeled as a control flow graph $A = (V, G)$, where nodes $v_j \in V$ are basic blocks and arcs $g_j \in G$ are branches. Each basic block v_j has an execution weight w_j and its operation can be modeled as a data dependency graph $W_j = (O_j, U_j)$. In W the nodes $o_k \in O_j$ are primitive operations of target independent representation and arcs $u_k \in U_j$ are data

dependencies. Each arc u_k has a label describing its data type dt_k. Each arc $g_j \in G$ has a branching probability P_j.

The core c consists of instructions $i_j \in I$ and execution architecture parameters. Each instruction i_j can be modeled as a weighted data dependency graph similar to W_j and instruction's execution cost C_j. If the core is not capable of implementing all the primitive operations in O, then virtual instructions are added to I to model the required subroutines. The architecture parameters are listed in Table 1.

Table 1. Core parameters.

Parameter	Values
Branch prediction technique, P_T	no/static/dynamic
Dynamic prediction efficiency P_e	0-100%
Superpipeling degree, D	1-N
Bypassing , D_B	yes/no
Number of execution paths, E	1-N
Number of registers R	0-N
Read buses B_R	0-N
Write buses B_W	0-N
Read/Write buses B_D	0-N
Program bus B_P	yes/no
Bus width B_w	1-N
Floating point cost factor C_f	1-N
Word length cost factor C_w	1-N

3.2 MAPPABILITY CORRELATION

In our approach the mappability estimate $m_i(c,a)$ is calculated for a single data dependency graph at the time using a simple ratio of estimates

$$m_i(c,a) = \begin{cases} \dfrac{e(c)}{e(a)}, & e(c) \le e(a) \\ \dfrac{e(a)}{e(c)}, & e(a) < e(c) \end{cases} \quad (1)$$

where $e(c)$ is the estimated core characteristic and $e(a)$ is the estimated algorithm characteristic. The result is then extended to cover the whole algorithm using weighted average of mappability values

$$M_c(c,a) = \frac{\sum_{i=1..|V|} (w_i \cdot m_i(c,a))}{\sum_{i=1..|V|} w_i} \quad (2)$$

where a is the control flow graph of the algorithm, w_i is the weight and m_i the mappability value for node i.

The good mappability of an algorithm and a processor core requires that the instruction-set is suitable for the required computation, the execution architecture supports the logical and effective ordering of operations and the data is available when needed. In order to manage the complexity of estimation, we have divided the correlation problem into six orthogonal parts, of which the average is the overall mappability.

Instruction set correlation depends on how effectively and extensively the core's instruction set can be used for the given algorithm. Effectiveness relates to the instructions' ability to execute the desired functionality. In the algorithm model the graphs $W_j = (O_j, U_j)$ will be replaced with the graphs $W'_j = (I_j, U'_j)$ by replacing the primitive operations $o \in O_j$ with implemented instructions $i \in I_j$. The procedure is basically similar to what happens in a compiler's back-end. The data types and available accuracy needs to be checked. If they do not match, the instructions costs are multiplied with floating point cost factor C_f or word length cost factor, C_w respectively. In equation 1 the algorithm estimate is the number of executed operations and the core estimate is the total cost of the executed instructions scaled with the ratio of used instructions to implemented instructions, k, which describes the extent of instruction usage.

Internal data availability correlation expresses how effectively registers can be used. In program execution the registers store intermediate results and often-used operands. Knowing how many registers can be used requires that we know the number of intermediate results $r(a)$ during algorithm execution. In the algorithm model the intermediate results are the arcs v in data dependency graphs W, so for each node we calculate the maximum number of intermediate results in one schedule step using ASAP and ALAP schedules, so $e(a) \sim |U_j|$. Because the schedules are not constrained by resources, they express the extent to which parallelism and registers can be exploited. The nodes belonging to a loop must be combined to one node for this correlation. The external dependencies inside the loop are considered internal dependencies and the new node weight is the smallest weight of the nodes belonging to the loop. The correlation for one node and number of registers can be calculated using equation 1, where $e(c) \sim R$ is the number of registers in the core model.

External data availability correlation deals with bus efficiency. The bus usage should correlate to the bus capacity. The number of program bus operations is assumed equal to the number of executed instructions, i_i. The number of data operations is estimated using external dependencies in W and operations without successors. It is assumed that external dependencies that are not part of a loop require bus operations, because the program flow is undeterministic, and that the results of the operations without successors must be written into memory, because, otherwise, the whole operations are useless. In the number of read and write operations, i_r and i_w, the bus widths are taken into account. The estimated algorithm characteristic is then the bus usage, e.g. $e(a) \sim i_i + i_r + i_w$.

The number of execution units limits the usability of bus capacity, because the core cannot use more data or instruction words than

there are units. We have modeled the effect by adjusting the bus capacity with the number of functional units. The estimated processor characteristic is the effective bus capacity, e.g. $e(c) \sim B \sim B_W + B_D$.

Control flow continuity correlation depends on the number of branch instructions and superpipelining degree. The branch instruction ratio is the estimated algorithm characteristic and is calculated by dividing the number of branch instructions $|i_b|$ by the number of all instructions $|I_j|$ in a program path, e.g. $e(a) \sim |i_b| / |I_j|$.

The branch prediction reduces the number of executed branch instructions and branch penalties. If static branch prediction techniques are used, it is assumed that the compiler can optimize all branches correctly and, in the case of dynamic prediction technique, the value must be given as a parameter P_e . The estimated core characteristic is the effective superpipelinining degree, $e(c) \sim (1 - P_e) \cdot D$.

The best mappability value is achieved when we have a long superpipeline and few branches, because the execution of instructions can be overlapped effectively. If we have a short superpipeline and a lot of branches, the branch penalties are small. If we have a short superpipeline and few branches, we do not exploit all the overlapping possibilities of the algorithm, and if we have a long superpipeline and lot of branches, the overlapping benefits are wasted because of branch penalties.

In **data flow continuity correlation** the idea is that if the execution order of instructions is fixed by data dependencies, it will cause data hazards and degrade the pipeline efficiency. The degree of data dependency can be estimated by analyzing the number of instructions in a schedule step in unconstrained ASAP or ALAP schedule of W_j . The instructions that can be scheduled on the same step can fill the pipeline without data hazards. The topology of W and bypassing support of the processor has an effect on the mobility of instructions, d , and a higher mobility makes it easier to exploit the pipeline more efficiently. The estimated algorithm characteristic is then $e(a) \sim d \cdot \bar{i}_j$, where $\bar{i}_j$ is average number of instruction in a schedule step. The core characteristic is the superpipelining degree, $e(c) \sim D$.

The **execution unit availability correlation** compares operation level parallelism in an algorithm to the number of parallel execution units. The parallelism of algorithm is constrained by the data dependencies and it can be studied by dividing the number of instructions in W by the number of steps in the shortest possible schedule, so $e(a) \sim \bar{i}_j$.

The parallelism of the architecture is constrained by the available execution units in each parallel execution path. The parallel execution units in the core are, typically, not alike and all the execution units cannot execute all the instructions. We estimate the number of parallel execution units by calculating the coverage of execution unit c by dividing the possible instruction/unit by all instructions/core and then adjusting the number of execution paths E with this value, e.g. $e(c) \sim c \cdot E$.

3.3 CAMALA PROTOTYPE TOOL

The presented analysis method is implemented using SUIF and SUIF2 compilers, IR converter and CAMALA correlator written with C++ in PC/Linux environment. The correlator is about 6000 lines of C++ and a typical analysis takes only a few seconds in a 333 MHz laptop PC.

In the analysis procedure the algorithm is given as ANSI-C-program and the core model as a text file. The C-code is compiled in SUIF intermediate format with SUIF2. For the target-independent optimizations we have used SUIF. The SUIF intermediate format is then converted into our representation using an IR converter. We have used the gcov tool of gcc in the execution count and branch probability analyses. Bounds of data values have been analyzed manually and the results added to the algorithm model. The mappability values can then be calculated with the CAMALA tool and the results analyzed. In addition to the final mappability value, the tool also gives optimal values to core parameters.

4. EXPERIMENTAL RESULTS

Mappability values are difficult to compare to the measurable system characteristics such as cost, performance, resource utilization or power consumption because they all show only a part of the problem and because they have dependencies. For example, the execution time might keep getting lower with more powerful processors but mappability values start to decrease because all the resources of the processor cannot be used effectively anymore. On the other hand, reaching 100% utilization does not guarantee that all the potential of the algorithms has been exploited. It is also impossible to combine these quality characteristics into one quality measure, because the relative weights of these factors depend on the type of system, production volumes and other non-technical issues.

We have tested our method by identifying suitable processor architectures for the Hiperlan/2 WLAN transceiver and by analyzing their feasibility. We first modeled the main algorithms of the WLAN modem in C and generated our intermediate representations using SUIF2, gcov and the CAMALA tool. The Viterbi algorithm that is a major part of Hiperlan/2 was excluded due to its very high processing power requirements that make a hardware implementation more feasible.

Secondly, we generated a database of 10800 processor architecture models. These models have different architectural parameters for superpipelining degree (D), number of execution paths (E), number of registers (R) and number of data buses (B). They represent the typical means to improve performance. In the database the $D = [1,20]$, $E = [1,6]$, $B = [1,6]$ and $R = [1,64]$. All the other parameters were fixed so that the size of the database would stay reasonable. We also excluded the instruction set correlation by assuming identical instruction sets, because in this case the objective was to identify the main architectural characteristics of the processors prior to entering into the detailed design.

Thirdly, we calculated 140400 mappability values for alternative mappings. Based on this information, we finally identified the most suitable architectures for each algorithm, the best architecture for all algorithms, and sets of architectures and

mappings where the computational load of architectures is feasible.

Table 2 presents the average parameter values based on the ten best mapping processors and the MOPS requirements for each algorithm. For the encoding, demodulation and symbol filtering, the suitable processor structure is notably different than for the other algorithms, and their worst-case MOPS requirements are quite high. This indicates that the hardware implementation might be preferable. For the rest of the algorithms, the parameter values are similar enough to justify executing them on the same processor. In fact, this, or complete hardware implementation, also seems to be the solution in commercial modems [15].

However, we wanted to analyze in more detail what kind of single or asymmetric multiprocessor systems could be feasible assuming that every algorithm in Table 2 would be implemented in software. The architectures and mappings were selected so that average mappability was optimized. Figure 2 summarizes the results. As expected, the mappability values decrease when the number of processors is reduced because the algorithms would be executed in non-suitable processors.

Table 2. Average architecture parameters of best ten architectures for each WLAN modem algorithm and computational complexities of those algorithms.

Algorithm	***D***	***E***	***R***	***B***	**MOPS**
Convolutional encoding	3.5	1.5	1.0	1.5	54 - 486
Guard interval insertion	7.5	2.0	4.0	2.0	80
Interleaving	7.5	2.0	3.0	2.0	12 - 72
Modulation	4.5	2.0	4.0	2.0	12
Symbol filtering	9.5	3.0	12.0	1.0	120
Deinterleaving	6.5	3.0	3.0	2.0	12 - 72
Synchronization	10.5	2.0	8.0	2.0	200
FFT	10.5	2.0	9.0	1.0	296
Frequency error correction	10.5	2.0	8.0	1.0	156
Guard interval deletion	7.0	2.0	4.0	1.0	-
BPSK demodulation	4.0	1.5	2.0	2.5	60 - 504
QPSK demodulation	4.5	1.0	3.0	2.0	
QAM16 demodulation	5.5	1.0	6.0	2.0	

As we can see from Table 2 and Figure 2, there is no single processor that suits all. With more processors the average mappability can be increased considerably, but after seven processors only minor improvements can be achieved.

The selected architectures and mappings, and MOPS requirements in the two- and three-processor cases are shown in Table 3. The computational capacity requirement of a single processor solution would be almost 2000 MOPS, even without the Viterbi algorithm. In the two-processor case the algorithms that can exploit longer pipeline and internal registers are mapped to a more complex architecture (D = 11, E = 2, R = 9, and B = 1), except synchronization that would need more resources to external communication. The rest of the algorithms were mapped to a simpler architecture (D = 5, E = 2, R = 4, and B = 2). In the three-processor case the demodulation functions were mapped to a more optimal and simpler architecture (D = 5, E = 1, R = 3, and B = 2). According to this analysis, the pure software modem is not feasible unless the processors are used for other purposes when communication is not needed. Mappability-based partitioning alone did not result in a very balanced system. However, if we were to implement the convolutional encoding and demodulation algorithms in the hardware, it would lower the requirements for both architectures in the two-processor case to a reasonable level of 500 MOPS.

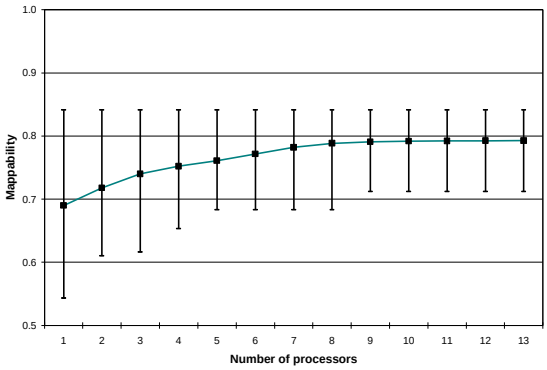

Figure 2. The mappability values and their variations (minimum and maximum mappability values) in the case of multiple processors.

Table 3. Mappings and estimated complexities of two and three processor core solutions.

Number of cores	**Core type**	**Algorithms mapped to core**	**MOPS**
2	11-2-9-1	Symbol filtering, FFT, frequency error correction	572
	5-2-4-2	Synchronization, encoding, demodulation, etc.	1426
3	11-2-9-1	Symbol filtering, FFT, frequency error correction	572
	5-1-3-2	All demodulation algorithms	504
	5-2-4-2	Synchronization, encoding, etc.	922

The mappability estimation is a fast and easy way of performing design space exploration in case of the WLAN modem. The results that are presented here are in line with both the commercial implementations of modems and common understanding of the implementations of these algorithms. However, the results are

only applicable to analyzed algorithm models. The optimization of algorithms has not been considered. The effect of instruction set has also been omitted. The instruction sets of the architectures used in this study would be very different from each other in real cases and it would affect the final results. The effects of memory organization have also been ignored. Again, the architectures would benefit differently from advanced caches, etc.

5. CONCLUSIONS

This paper presents a method for the estimation of algorithm-architecture mappability, which identifies how different characteristics of algorithms and architectures correlate. It can be used for the identification of required architecture characteristics for a set of algorithms or for the selection of processor cores in the design of a multiprocessor system on chips. Only simple algorithm and architecture models are needed, therefore the method is suitable for fast reduction of design space. The method is demonstrated with the CAMALA tool.

The ideas have been applied to the identification of suitable architectures for Hiperlan/2 transceiver algorithms. 140400 different architecture-algorithm pairs were analyzed and the results were equal to existing implementations and more extensive analyses of algorithm characteristics. Currently, only the internal execution architecture of the core is considered in the approach. The effect of memory organization and caches are excluded.

The extension of the method to general-purpose processing seems possible. Integration with the estimation of required computing capacity of the core, the addition of memory organization effects and integration with the validation of performance would improve the usability of the methods significantly. However, the mappability estimation as such is one of tools needed by system-on-chip architects today.

6. ACKNOWLEDGEMENTS

The work was partially funded in the Finnish - Swedish research program Explorative System-Integrated Technologies (EXSITE) by the TEKES, VINNOVA, Nokia, Ericsson, VTT and Spirea. We thankfully acknowledge the support from research professor Aarne Mämmelä and his team. Mr. Mika Kasslin from Nokia and Mr. Kari Tiensyrjä and Mr. Tapio Rautio from VTT Electronics have also given extremely valuable comments.

7. REFERENCES

[1] Allan, A., et al, “2001 Technology Roadmap for Semiconductors”, *Computer*, Vol. 35., No. 1, 2002, pp. 42-53

[2] Chang, H. et al., *Surviving the SOC Revolution – A Guide to Platform-Based Design*, Kluwer Academic Publishers, 1999, 235 pp.

[3] Kreutzer, K. et al., “System Level Design: Orthogonalization of Concerns and Platform Based Design”, *IEEE Transactions on Computer-Aided Design of Circuits and Systems*, Vol. 19, No. 12, December 2000, pp. 1523 –1543.

[4] Hennessy, J.L. & Patterson, D. A., *Computer Architecture – A Quantitative Approach, 2nd ed*. Morgan Kauffman Publishers Inc, 1996, 760 pp.

[5] Ernst, R., ”Codesign of Embedded Systems: Status and Trends”. *IEEE Design and Test of Computers*, Vol. 15 No. 2, 1998 pp. 45-53.

[6] Soininen, J.-P. et al., "Mappability Estimation of Architecture and Algorithm", *Proceedings of Design Automation & Test in Europe 2002 Conference*, 2002, p. 1132.

[7] Krishna, C. (ed.), "Performance Modeling for Computer Architects", IEEE CS Press, Los Alamitos, CA, 1996, 391 pp.

[8] Flynn, M. et al., “Deep-Submicron Microprocessor Design Issues”, *IEEE Micro*, Vol. 19, No. 4, July-August 1999, pp. 11-22.

[9] Heidelberger, P. & Lavenberg, S., Computer Performance Evaluation Methodology, *IEEE Transaction on Computers*, Vol. C-33, No. 12, 1984, pp. 1195-1220

[10] Gong, J. et al., “Performance evaluation for application-specific architectures*”, IEEE Transactions on VLSI*, Vol. 3, No. 4, December 1995, pp 483-490.

[11] Suzuki, K. & Sangiovanni-Vincentelli, A., “Efficient Software Performance Estimation Methods for Hardware/Software Codesign”, *Proceedings of 33rd Design Automation Conference*, 1996 pp. 605-610.

[12] Lazarescu, M. et al., “Compilation-based Software Performance Estimation for System Level Design”, *Proceedings of High Level Design Validation and Test Workshop*, 2000, pp. 167-172.

[13] Ghazal, N. et al., “Retargetable Estimation Scheme for DSP Architectures”, *Proceedings of the Asia and South Pacific Design Automation Conference*, 2000, pp. 485-489.

[14] Carro, L. et al., “System Synthesis for Multiprocessor Embedded Applications”, *Proceedings of Design Automation and Test in Europe*, 2000, pp. 697-702.

[15] Grass, E. et al., “On the Single-Chip Implementation of a Hiperlan/2 and IEEE 802.11a Capable Modem*”, IEEE Personal Communications*, December 2001, pp. 48-57.