ISSN No. 0976-5697

Volume 2, No. 4, July-August 2011

International Journal of Advanced Research in Computer Science

RESEARCH PAPER

Available Online at www.ijarcs.info

A Multi Layer Technique for Performance Estimation for Asip Design Space Exploration

Manoj Kumar Jain
Associate Professor in Computer Science
Mohanlal Sukhadia University,
Udaipur, Rajsthan, India
manoj@cse.iitd.ernet.in

Abstract: Application Specific Instruction Set Processor or ASIPs are designed for a given application or for a set of applications. Since application set is limited, a better analysis of applications is possible which helps in identifying their special characteristics. These characteristics are used in ASIP design space exploration. This exploration suggests the optimum design which meets the stringent design constraints. The exploration is supported by estimation tools. Performance estimation is one such tool. Various researchers had suggested two types of techniques for performance estimation, namely, simulator based, and scheduler based approaches. They seem to be contrary to each other. This paper proposes that they are not contrary; in fact they are complimentary to each other. Since the scheduler based approaches use a very coarse model of architecture so they might not be as accurate as simulator based approaches. But the scheduler based approaches are very fast in nature and can handle a larger design space as they are not dependent on retargetable compilers and retargetable simulators. So we propose a new technique for performance estimation. A scheduler based approach should be used for an early design space exploration as the other approach is not suitable at this stage. This layer will suggest a few possible architectures suitable for input application. These architectures can be further analysed by a simulator based technique.

Keywords: Application Specific Instruction Set Processor (ASIP), simulation, synthesis, time to market, instruction set simulator, scheduler based, and performance estimation.

I. INTRODUCTION

Now a day's embedded systems are used everywhere. A few examples include wireless handsets, networked sensors, smart cards, network routers, gateways, firewalls, and servers. The heart of embedded system is usually implemented either using a general purpose processor (GPP), or using an application specific integrated circuit (ASIC), or combination of both.

GPPs are flexible but do not meet out many design constraints like performance requirement, area and power constraints. On the other hand ASICs are very rigid in nature, and they are expansive. Application Specific Instruction Set Processor (ASIP) has emerged as a popular solution. ASIP provides a design which meets out the design constraints and has a limited flexibility.

A good survey of ASIP design methodologies is available in [1]. Five main steps identified in ASIP synthesis are application analysis, design space exploration, instruction set generation, code synthesis and hardware synthesis.

Typically ASIP design starts with analysis of the applications. These applications with their test data should be analyzed statically and dynamically using some suitable profiler before proceeding further in the design process. Inputs from the application analysis step are used along with the range of architecture design space to select a suitable architecture(s). The selection process typically can be viewed to consist of a search technique over the design space driven by a performance estimator. Instruction set is generated or synthesized for chosen architecture. Software tool set including operating system, editors, compilers, debuggers etc. are generated. A synthesizable hardware description of the selected architecture is provided to synthesize the processor. Design space exploration is driven by various estimators. Performance estimation is one such estimator. Performance estimator can be simulator based or scheduler based.

This paper is organized as follows. II Section Describes a typical ASIP Design methodology. Design space exploration technique is described in Section III. IV Section presents a simulator based approach for performance estimation. V Section presents a scheduler based approach for performance estimation. A brief comparison of both is presented in Section VI. Section VII presents proposed multi layer performance estimation technique. Paper concludes with conclusions in Section VIII.

II. ASIP DESIGN METHODOLOGY

Gloria et al [2] defined some main requirements of the design of application-specific architectures. Important among these are as follows:

- Design starts with the application behavior.
- Evaluate several architectural options.
- Identify hardware functionalities to speed up the application.
- Introduce hardware resources for frequently used operations only if it can be supported during compilation.

ASIP fits in between these two and provides flexibility at lower cost than general programmable processors. According to MK Jain et al [1] design of ASIP can be typically divided in five steps which is shown in Figure 1:

- Application Analysis
- Architecture design space Exploration.
- Instruction-set generation

- Code synthesis
- Hardware synthesis

A. Application Analysis

ASIP design starts with analysis of application, analysis of test-data and design constraints. An application written in any high level language is analyzed both statically and dynamically which is then stored in some suitable intermediate format, which is then used in the subsequent steps.

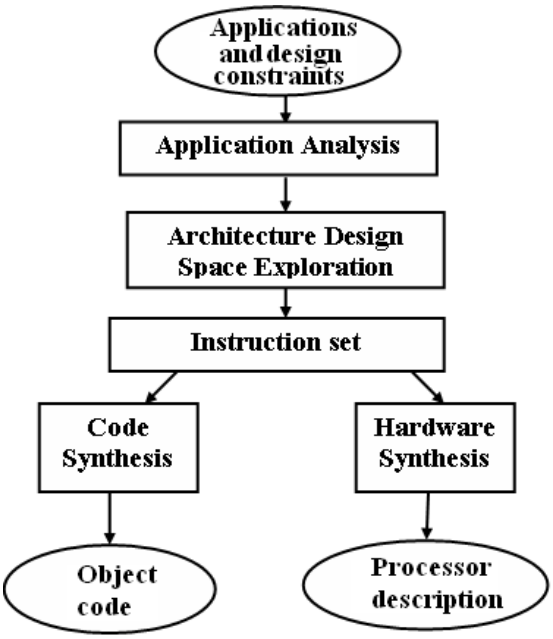

Figure 1. Flow Diagram of ASIP design Methodology

B. Architecture Design Space Exploration

It involves identifying the broad architectural features of the ASIP. First of all, the architectural space to be explored is defined, keeping in view the parameters extracted during application analysis and the input constraints. Architecture is defined using some standard Architecture Definition Language (ADL) as EXPRESSION [3] and LISA [4].

C. Instruction Set Generation

Instruction set is to be generated for that particular application and for the architecture selected. This instruction set is used during the code synthesis and hardware synthesis steps.

D. Code Synthesis

Compiler generator or retargetable code generator is used to synthesize code for the particular application or for a set of application.

E. Hardware Synthesis

In this step the hardware is synthesized using the ASIP architecture template and instruction set architecture starting from a description in VHDL/VERILOG using standard tools.

III. DESIGN SPACE EXPLORATION

Architecture exploration starts with the application analysis. We need to input the parameters of application analysis along with the identified architecture design space to the process block which is responsible for performance estimation. Then we need to do the performance estimation for the inputted architecture along with the search control and then the architecture will be selected. Figure 2 explains the procedure of architecture explorer.

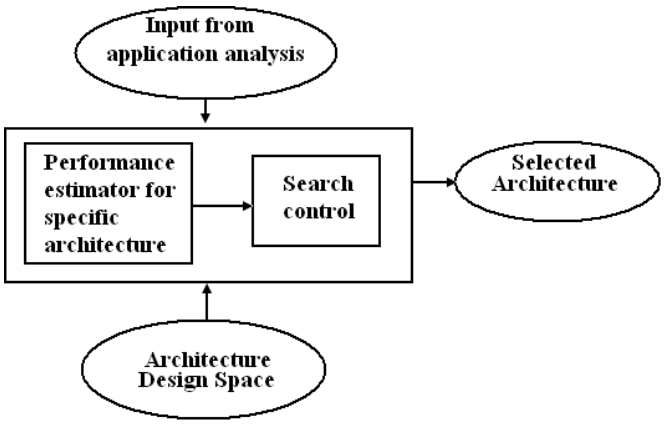

Figure 2. Block Diagram of an Architecture Explorer

Performance estimation which drives the design space exploration is done by simulator based approach or by scheduler based approach.

IV. SIMULATOR BASED PERFORMANCE ESTIMATION

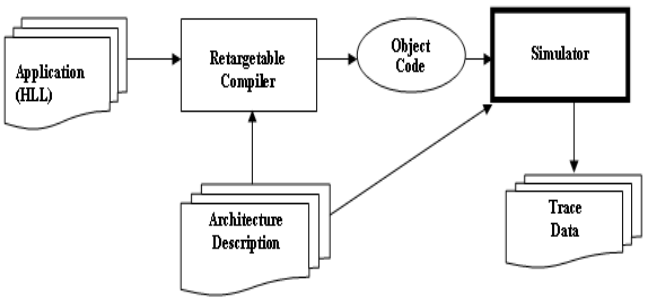

Figure 3. A typical simulator based performance estimation technique.

In such approaches [Figure 3], first code is generated for the target processor using retargetable compiler. Then this code and processor description is supplied to a retargetable simulator which simulates the code for target processor and gives performance estimation.

Kienhuis et al. [5] constructed a retargetable simulator for an architecture template. For each architecture instance, a specific simulator is derived in three steps. The architecture instance is constructed, an execution model is added and the executable architecture is instrumented with metric collectors to obtain performance numbers. Object oriented principles together with a high-level simulation mechanism are used to ensure retargetability and efficient simulation speed.

V. SCHEDULER BASED PERFORMANCE ESTIMATION

Such approaches take a very simple architecture model as input. A suitable internal representation of the input application is generated. Application is run on host machine to gather profile information. A suitable processor configuration is chosen by an explorer using a scheduler for performance

estimates. An illustrative approach of this class suggested by Gupta et al [6] and Ghazal et al [7] (Figure 4 and 5) is briefly described here.

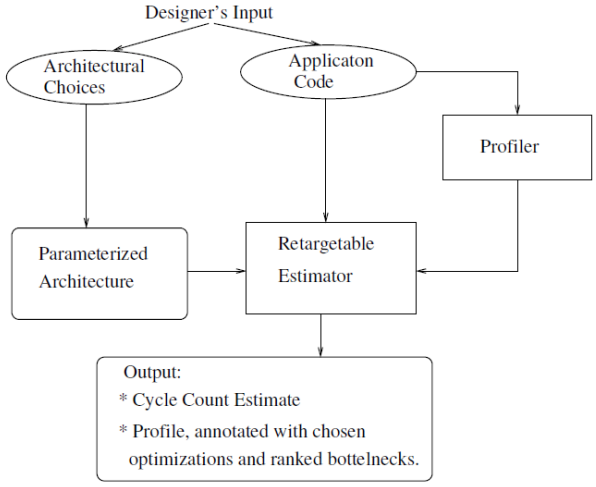

Figure 4. Overall flow of Retargetable Estimator

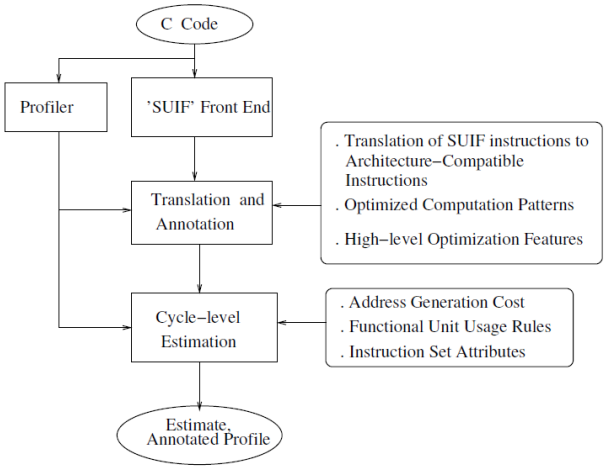

Figure 5. Flow of Estimation Scheme.

Each processor architecture considered for evaluation is described by the architecture model. Designer can chose from a database of previously captured architecture or can enter his own. Input application is described in high level language (C). Prediction of optimal run-time behaviour is achieved through profiling and a series of passes that search for generic optimizations, and more importantly, for the application-specific optimization features. The estimator provides the designer with a total cycle count estimate, and a profile of the code, annotated with the chosen optimizations and a ranking of the dominant code segments.

VI. SIMULATOR V/S SCHEDULER BASED PERFORMANCE ESTIMATION

When we look at both the approaches for performance estimation, we find that simulator based performance estimation technique is being used by various researchers right from the beginning of the research work on ASIP design starting from around 1990. This approach seems to be trivial and have been used by researchers for such a long time. Many architectural features have been explored so far using this approach. The problems associated with simulator based techniques are as follows.

Simulator based techniques need retargetable compiler and retargetable simulator for performance estimation. Generally it is not feasible to get such compilers and simulators which can retarget a large design space. Based on the retargetablility, retargetable compilers can be classified as automatic-retargetable or parameterizable, user retargetable and developer retargetable depending on the effort involved in retarget. Retargetable compilers can be divided mainly into three categories. So basically the design space which can be explored gets limited by capabilities of these compilers and simulators. The other problem associated with such approaches is large simulation times. Both these problems make such techniques unsuitable for design space exploration, especially for an early design space exploration where the design space is huge.

As time progressed, new architectural features added and the volume of the design space started expanding drastically. In 2000 a couple of researchers raised this issue and proposed scheduler based approaches as an alternative. Since its inception was almost a decade later that simulator based approach, a very few architectural features has been explored in such approaches. These approaches are very fast compared to simulator based approaches. But they may not be as accurate as their counterpart.

VII. MULTI LAYER PERFORMANCE ESTIMATION

Our proposed multi level performance estimation technique is shown in Figure 6. Initial design space is huge as architecture contains a number of parameterizable architectural features and each feature can have a number of possible values. It seems to be unrealistic or infeasible to explore such a huge design space using simulator based performance estimator. Some of the reasons are already discussed when we have compared these two techniques. Practically it would not be possible to have code generators for each configuration of such design space. There is a well known trade-off between retargetability and the quality of the generated code. So even if retargetable attempts to generate code for large designs, it can generate only sub optimized code. Long simulation times also do not allow us to use this approach at this level.

In contrast to this, scheduler based approaches use retargetable estimators which are very fast. Since scheduler based approach do not involve code generation and simulation and is faster than simulator based techniques so it would be better if such technique can be used at this level. These approaches do not take a detail model of architecture as they are not generating code for configuration under evaluation. Its architecture model is very simple so it is very easily retargetable which is very much significant at an early design space exploration considering the volume of design space at this level. It is correct that such technique may not generate results as accurate as that of simulator based technique. Considering this fact, rather than one particular processor and memory configuration a set of possible configurations should be suggested as outcome of this technique. This set of configurations will constitute design space for layer 2. In our opinion scheduler based approach is unavoidable at level. This way both techniques seems to be complimentary to each other rather that contrary to each other.

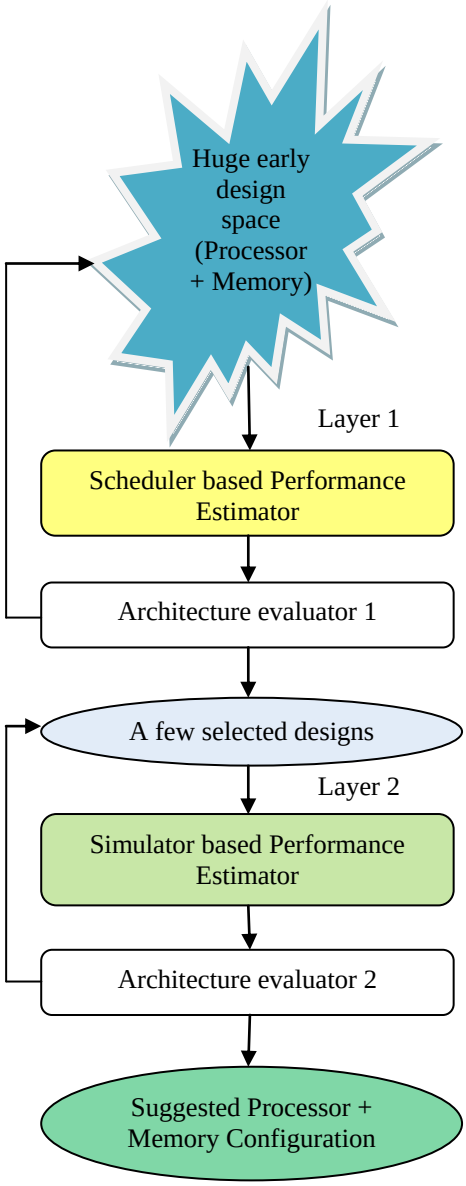

Figure 6. Proposed Multi Level Performance Estimator

Simulator based approach is used explore design space at layer 2. At this layer only a few designs are there in the design space. Simulator based approach uses a fine model of architecture and is supposed to be more accurate compare to scheduler based approach. Due to very small design space even comparatively longer simulation times will not be a problem and will meet out time to market constraint.

Suitable architecture evaluators are used at each level. They take generated estimates and helps in deciding about that configuration as well as it also helps in suggesting next configuration for evaluation. It also helps in pruning the design space. Retargetable simulator technique and scheduler based performance estimation technique are presented here.

A. Proposed Retargetable Simulator Technique

Proposed Retargetable Simulator technique is shown in Figure 7. Processor specification and Scheduled and optimized code are input to simulator. Processor Model, Interconnect model and Memory model interact among themselves. The entire model in conjunction simulates the instructions of the programs and provides the Execution statistics of the input program.

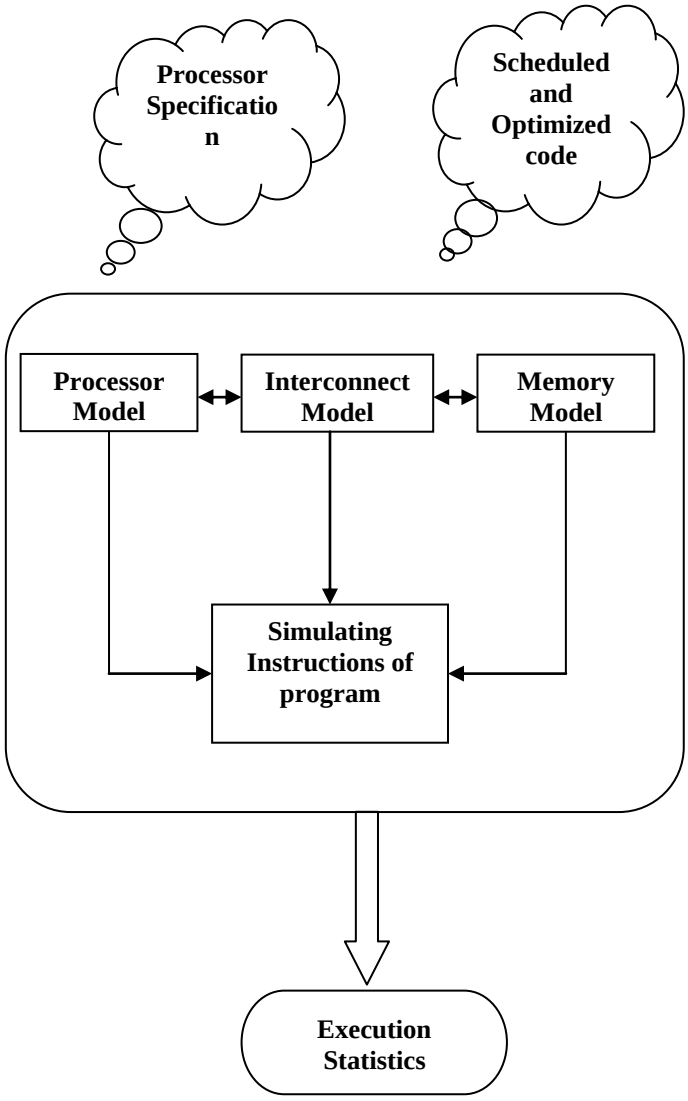

Figure 7. Methodology of our Retargetable Simulator

Our simulator supports the interconnection of bus. All processing elements and all memory modules are connected through a common bus. Uniform shared memory access is assumed, that is, access of any memory module from any processor takes the same amount of time (ignoring delays due to bus contention).

The simplest interconnection strategy is to use a single bus which is being shared by every other component for communication. Though this strategy is easy to implement, as the number of processor go up, the bus becomes the bottleneck. All the components connected to this bus should tune their interfaces to use the bus protocol. Apart from this, designers have to implement some arbitration mechanism to resolve the conflicts.

There are several other interconnection choices which enhance the communication bandwidth at higher cost. These are: multiple buses, multistage interconnection network (MINs) and crossbar switches. Out of these, crossbar establishes one to one correspondence, but this is most expensive.

The analyzed interconnection network is quite general and their performance is application dependent. However, in an

application specific multiprocessor architecture, one needs analyze the possible communication traffic of the application. This analysis will lead to the decision if one of the above interconnection is employed or a custom interconnection based on traffic is explored.

B. Proposed Retargetable Scheduler Based Performance Estimation Technique

We use the concept of Register Reuse Chains (RRCs), which makes it possible to do register allocation for unscheduled code blocks. While doing register allocation, an attempt is made to minimize the schedule overhead which will occur due to limited registers. A priority based resource constrained list scheduler is used to get local performance estimates of each block. Global analysis is performed at the function level to find additional schedule overhead required to handle global needs which are ignored when the local estimates are generated. This overhead is added to the local estimates to produce the total estimates.

Our performance estimation methodology is shown in figure 8. Input application (written in C) is profiled using *gprof* to find execution count of each basic block as well as functions. These execution counts are used to multiply with the estimated execution times. Intermediate representation is generated using *SUIF* [8]. Control and dependency analysis is done using this intermediate representation. Control flow graph is generated at the function level whereas the data flow graph is generated at the basic block level.

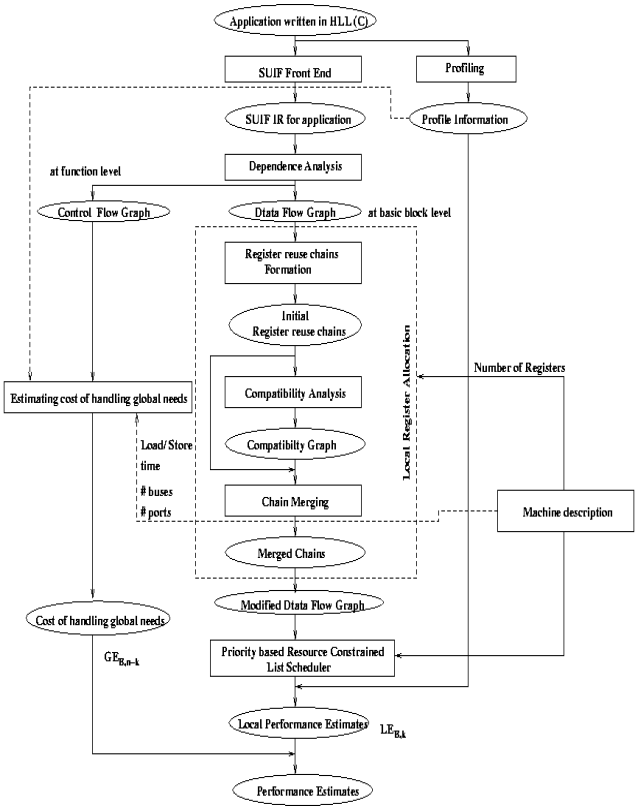

Figure 8. Scheduler based Performance Estimation Technique

For each basic block B, local register allocation is performed taking the data flow graph and number of registers to be used for local register allocation (say *k*) as input using a modified register reuse chains approach. Data flow graph may be modified because of additional dependencies as well as spills inserted during register allocation. This modified data flow graph is taken as input by a priority based resource constrained list scheduler, which produces schedule estimates. This estimate is multiplied by the execution frequency of block B to compute local estimate ($LE_{B,k}$) for this block.

Local estimates are produced for all the basic blocks contained in a function, for the complete range of register file sizes to be explored. Schedule overheads needed to handle global needs with limited number of registers are computed using life time analysis of variables. For each block, we need information on variables used, defined, consumed, live at entry and exit points of this block. This additional global needs overhead is also generated for the complete range of number of registers for each basic block. Then, we decide on the optimal distribution of the registers available (say *n*) into registers to handle local register allocation (*k*) and registers to handle global needs (*n-k*), such that overall schedule estimate for that block is minimized.

Overall estimate for a block B can be expressed as

$$OE_B = min_k(LE_{B,k} + GE_{B,n-k}) \quad (1)$$

where OE_B is the total schedule estimate for basic block B, and $GE_{B,n-k}$ is the overhead to handle global needs with *n-k* registers. OE_B values for all blocks are summed up to produce estimates at the function level. Estimates for all functions are added together to produce overall estimate for the application i.e. et_R. So et_R can be expressed as

$$etR = \sum_{for\ each\ function} \ \sum_{for\ each\ basic\ block\ B} (OE_B) \quad (2)$$

VIII. CONCLUSION

Embedded Systems are the need of the hour. They are being used everywhere. Combination of a processor (typically a GPP) and some ASICs are used for embedded system. This implementation assumes that GPP is fine for them. But this may not true. This problem is resolved with the invention of ASIP which is basically a customized processor. Important steps involved in ASIP synthesis include application analysis, design space exploration, instruction set generation, code synthesis and hardware synthesis.

Design space exploration is the most crucial step in ASIP synthesis. This exploration is supported by estimators. We have considered performance estimation. We found that two different approaches exist in literature for performance estimation which seems to be contrary in nature. We propose a novel approach and suggested a multi layer performance estimations. According to our approach, scheduler based approach should be used for an early design space exploration considering the volume of design space at this level and the design space which can be handled by scheduler based approach, its easy retargetability characteristics, its faster speed. Since scheduler based approach uses a very simple processor and memory configuration its estimates may not be very accurate. So a set of possible designs suggested by scheduler based approach at level 1 are further estimated in detail using simulator based approach at layer 2.

IX. REFERENCES

[1] M.K. Jain, M. Balakrishnan, and A. Kumar, "ASIP Design Methodologies: Survey and Issues", In Proceedings of the IEEE

/ ACM International Conference on VLSI Design. (VLSI 2001), pages 76–81, January 2001.

[2] Gloria A. D.; Faraboschi, P., "An evaluation system for application specific architectures.", In Proc. Micro-23, 27-29 Nov. 1990, pp. 80-89.

[3] A. Halambi, P. Grun, A. Khare, V. Ganesh, N. Dutt, A. Nicolau, EXPRESSION: A Language for Architecture Exploration through Compiler/Simulator Retargetability, In Proceedings of the Design Automation and Test in Europe (DATE), pages 485–490, March 1999.

[4] S. Pees, V. Zivojnovic, H. Mey, LISA- Machine Description Language for Cycle Accurate Models of Programmable DSP Architectures, In Proceedings of the Design Automation Conference (DAC), pages 933–938, June 1999.

[5] B. Kienhuis, E. Deprettere, K. Vissers, and P. van derWolf. The Construction of a Retargetable Simulator for an Architecture Template. In *Proceedings of the Sixth International Workshop on Hardware/Software Co-design 1998 (CODES/CASHE '98)*, pages 125–129, March 1998.

[6] T. V. K. Gupta, P. Sharma, M. Balakrishnan, S. Malik,, Processor evaluation in an embedded systems design environment, In Proc. VLSI Design 2000, pages 98-103, January 2000.

[7] N. Ghazal, R. Newton, and J. Rabaey. Retargetable Estimation Scheme for DSP Architecture Selection. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 485–489, January 2000.

[8] SUIF Homepage. http://suif.stanford.edu/

M.K. Jain received the M.Sc. degree from M.L. Sukhadia University, Udaipur, India, in 1989. He received M.Tech. Degree in Computer Applications and PhD in Computer Science & Engineering from IIT Delhi, India in 1993 and 2004 respectively. He is Assistant Professor in Computer Science at M.L. Sukhadia University Udaipur, India since 1993. His current research interests include application- specific-instruction- set processor design, wireless sensor networks, semantic web and embedded systems.