Processor Evaluation in an Embedded Systems Design Environment

T.V.K.Gupta, Purvesh Sharma, M Balakrishnan
Department of Computer Science & Engineering
Indian Institute of Technology, Delhi, INDIA

Sharad Malik
Department of Electrical Engineering
Princeton University, NJ, USA

Abstract

In this paper, we present a novel methodology for processor evaluation in an embedded systems design environment. This evaluation can help in either selecting a suitable processor core or in evaluating changes to an ASIP. The processor evaluation is carried out in two stages. First, an architecture independent stage in which processors are rejected based on key application parameters and secondly, an architecture dependent stage in which performance is estimated on selected processors. The contribution of our work includes identification of application parameters which can influence processor selection, a mechanism to capture widely varying processor architectures and an instruction constrained scheduler. Initial experimental results suggest the potential of this approach.

1 Introduction and Objective

Embedded Systems can be found in a large variety of applications today like transportation systems, networking and mobile communication. Typically, they comprise of a processor and some application specific hardware built around it. The software is used for quick, easy and cost effective implementation while the hardware is used to speedup critical parts of the algorithm. Till now, the design of embedded systems was largely carried out in an ad-hoc manner. With dramatic decrease in silicon costs, it is now possible to implement very complex systems on a single chip. With significant increase in performance requirements of applications, the expected complexity of such systems will require a rigorous design methodology with a need for extensive support design tools. The focus of the Embedded Systems Project at IIT Delhi is to develop a systematic methodology for embedded systems design.

The embedded systems project aims at the development of a design methodology for embedded systems for vision/image processing applications. Given a system specification, by following the methodology and with the help of the tools developed to support it, the user will be able to synthesize a system that meets the specification constraints. Figure 1 shows the overall design flow of the system with the following main steps - specification, partitioning, synthesis and system integration.

The processor evaluation is one of the components of the above project. The objective of the processor evaluation tool, given any application behavioral specification and a range of target processors, is to evaluate the processors for their suitability. This is intended to be achieved in two steps:

- Elimination of unsuitable processors on the basis of major mismatch between application requirements and architecture characteristics.
- Evaluation of remaining processors by a quick estimation of the application performance in terms of number of cycles.

The above steps require identification of a set of parameters that characterize the processor architecture to facilitate their evaluation. Similarly, on the application side, some parameters have to be extracted out to capture the application characteristics. The final objective is to automate the evaluation process with major thrust on retargetability of the application. The dotted box in figure 1 shows the processor evaluation activity as part of the overall project.

It is often the case that at the evaluation stage of the project, one doesn't have the compilers and simulators for all the range of processors that one needs to consider. Moreover, compilers for most of the DSP processors are not efficient [1, 2]. Further, the range of processors can have widely varying underlying architectures. Hence this project attempts to bring all the target processors on a common platform to evaluate them.

Another major application of our processor evaluation methodology will be in the design of ASIPs. We can quickly estimate the performance improvement achievable due to architectural modifications. These modifications may include the addition of new instructions, increase in the number of functional units, registers, memory accesses in parallel, depth of pipeline, etc. Thus, our tool would be able to predict whether such modifications would enable an ASIP to meet the application's real time constraints.

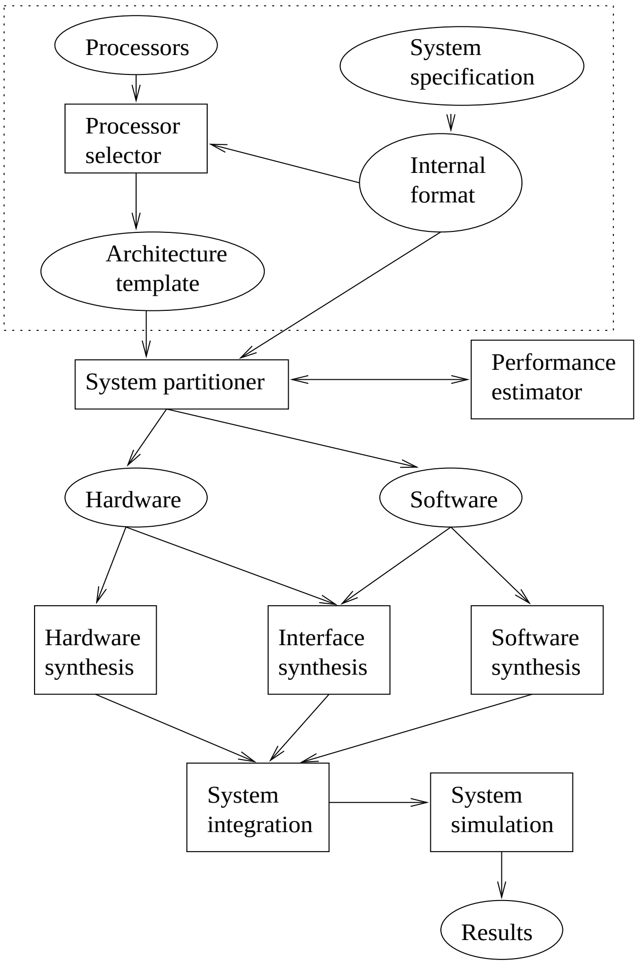

Figure 1. Abstract design flow

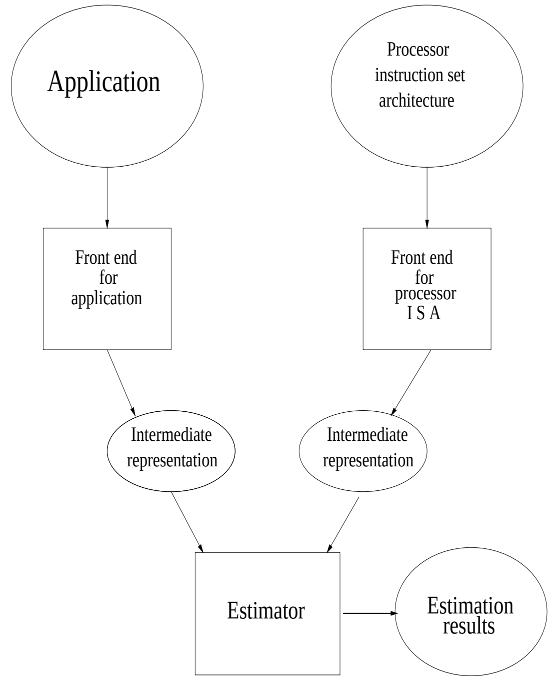

Figure 2. Proposed methodology for processor evaluation

2 Overall Approach

The overall approach for processor evaluation is shown in figure 2. The methodology takes as input the application as well as the processor architecture specification. The application is to be converted to some intermediate representation (IR) using a front end tool. Simultaneously, the processor architecture features have to be captured. The estimator takes the two inputs and generates estimates of code size and execution time. This information regarding code size and execution time is appended to the IR for a range of processsors. The partitioner uses this information for software performance estimation. A key feature we would like to build in our estimator is the possibility of trading off quality/accuracy of estimates against estimation time.

Currently, we are using **SUIF** (Stanford university intermediate format) [4] as the front end tool for the application. SUIF is basically a front end of a compiler that converts the application described in C into an intermediate format. The libraries provided by SUIF can be used to extract relevant information from the intermediate representation (IR). A number of parameters characterizing the application are extracted from the intermediate representation. For example, one such parameter is the average number of instructions that can be executed in parallel. This indicates the nature of concurrency in the application which when analysed along with the performance specifications, translate to the requirements for the processor. For details of other parameters and their relevance in the processor selection process, refer to section 4. In order to capture the architectural characteristics, we have developed a description format. The format describes the processor in terms of operations that are executed by the different functional units along with their execution time, concurrency of execution and constraints on their concurrency. Details of the format are described in section 3.

3 Architecture Representation

Architectural features of the processors play a key role in the matching of an application to the processor. For extracting key features of the processors under consideration, we used a simple processor description format.

This format is an adaptation from the machine descrip-

Table 1. Application Parameters

S.No.	Parameter	Relevant Processor Architecture Features
1	Average block size	Branch penalty
2	No. of multiply-accumulate operations	multiply-accumulate as a custom operation
3	Ratio of address computation instructions to data computation instructions	Separate address generation ALU
4	Ratio of I/O instructions to total instructions	Memory bandwidth requirement
5	Average arc length in the data flow graph	Total no. of registers
6	Unconstrained ASAP scheduler results	Operation concurrency and lower bound on performance

tion format which is used to describe the TriMedia processors from Philips [5]. This format incorporates the following :

- Different types of functional units, associated operations and delay properties of each functional unit such as latency, recovery & throughput.
- The number of functional units of each type.
- Number of registers.
- Number of operation slots in each instruction.
- Restrictions on mapping operations to slots.
- Concurrent Load/Store operations.

In architectures with multiple functional units, we have encountered both the following situations :

- Latency of an operation depends on the functional unit performing the operation.
- Latency depends on the operation being performed independent of the functional unit on which it is mapped.

To take care of these possibilities, we associate a latency with both the operation and the functional unit. The final delay associated with an operation on a specific functional unit is the product of the delay of the functional unit and the delay of that operation. For details of the grammar used for describing the processor architecture, refer to our report [7].

4 Application Characteristics

In order to perform the match between the application and the processor architecture, we have approached the problem in two stages. In the first stage, we extract certain parameters from the application, independent of the target processor architecture. These parameters give an indication of some of the characteristics of a suitable processor. Hence, in this stage, processor architectures which are uncorrelated with the application requirements are eliminated. This stage is particularly useful, when one has to decide among a large number of variable processors of very different capabilities. In the second stage of evaluation, we take up the selected processors. Here, the target processors are evaluated to get an estimate of the number of cycles. This is done by an architecture constrained scheduler that schedules the application on a processor architecture, as captured by the description file.

4.1 Application parameters

As mentioned earlier, we have extracted some parameters from the application. Table 1, second column shows the parameters that are being currently extracted. The third column lists the architectural features relevant to this parameter which influence the performance of the application on a processor. From an analysis of these parameter values vis-a-vis architectural features, it should be possible to 'grossly' evaluate suitability or otherwise of a processor.

The "average block size" parameter gives an indication of an acceptable branch penalty. If the block size is too small, then a deeply pipelined processor with a high branch penalty will have larger overheads. The number of "mac" operations parameter indicates the frequency of the multiply-accumulate operation in the application. Therefore, depending on the percentage of "mac" operation, the designer can quickly analyze the effect of the presence/absence of "mac" instruction in the processor. The "Ratio of address computation instructions to data computation instructions" parameter indicates the effect of a separate address generation unit in the target architecture. The "Ratio of I/O instructions to total instructions" parameter influences the memory bandwidth requirement. Higher ratio implies a need for a processor that supports more concurrent load/stores. The "Average arc length" parameter shows the average number of cycles between the generation of a scalar and its consumption in the data flow graph. Thus, it influences the size of the register file of a processor. A higher value means that the variables have to be kept in the register file for a longer time and therefore, more number of registers are required. This can lead to accepting/rejecting single accumulator processors.

The unconstrained ASAP scheduler schedules the application using the "as soon as possible" strategy. It outputs a histogram that indicates the level of concurrency present in the application, i.e. the operation lelevel parallelism that can be utilized by the application.

A number of simplifying assumptions have been made in the development of the above modules. For example, in the average block size module, the instructions associated with the condition code evaluation of the conditional structures and loops are ignored. It is assumed that each array instruction contributes to the total instructions by at least twice the number of dimensions it has with two instructions to compute the array element address in each dimension. Further, the array accesses are assumed to point to data that is always in the memory. Similarly, in the ASAP scheduler used in this stage, no resource constraint is taken into account. The instructions within a block are scheduled concurrently whereas blocks within a function are scheduled sequentially. This assumption is valid for architectures not supporting speculative execution. All scalars referenced in a block are assumed to be brought into the register file at the start cycle of the block. These assumptions are likely to lead to lack of accuracy in estimation. Extensive testing would be required to establish the collective effect of these on our estimates.

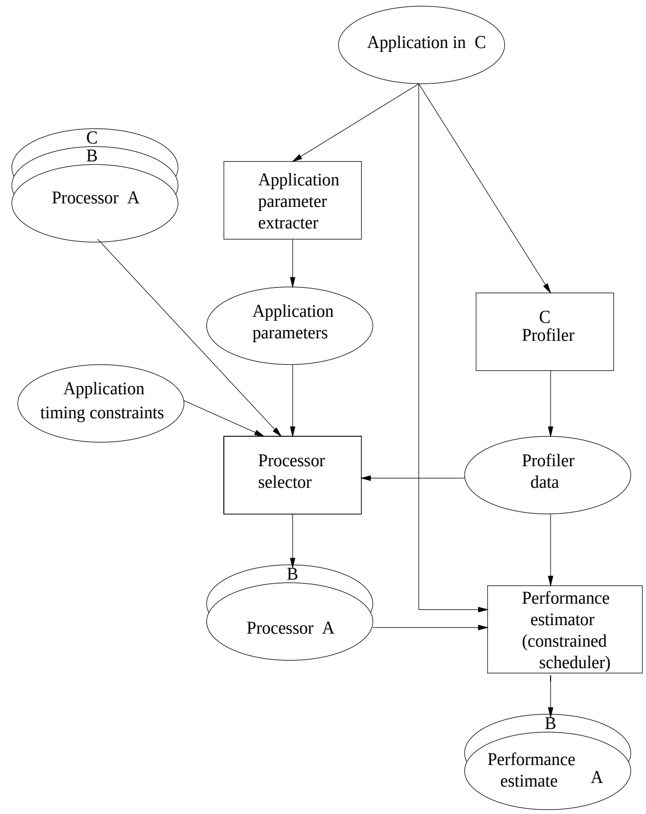

Figure 3. Processor evaluation: Flow diagram

4.2 Architecture Constrained Scheduler

This is the second stage of the evaluation process. Till now, the architectural aspects and the application characteristics were being considered independently. But in order to do performance estimation of the number of clock cycles, the two have to be considered together by one analyzer. We have developed a constrained scheduler that estimates the number of clock cycles, given the description file of the target processor architecture. A key feature of this scheduler is its capability to reflect the flexibility(or otherwise) of the instruction set to handle concurrency in its output. Instruction encoding of many processors don't permit arbitrary simultaneous invocation of all functional units.

The architecture constrained scheduler is based on the list scheduling algorithm [8]. The scheduler does a pass on the application to generate the data dependencies. After that, a priority is attached with each operation which is based on the height of the operation in the dependency graph. This priority function is used by the list scheduler to choose from available operations. The constrained scheduler reads the processor description file to assign operations to the functional units. The estimator also takes the profiling information of the application to get the number of iterations of each block of code. Depending on the iteration count, the estimator generates the cycle count.

A number of simplifying assumptions are taken in the design of this scheduler. Some of these are: all operations operate on operands in registers and all operations involved in the address computation of an array instruction are carried out by the insertion of explicit address computation instructions. A first fit heuristic is followed when assigning functions to functional units and slots.

5 Overall Environment and Estimation Results

Figure 3 shows the detailed flow diagram of the processor evaluator. First, the application is passed through a profiler "gprof" [10] to estimate the frequency of various blocks in the C program. This is followed by two stages of evaluation: the architecture independent stage and the architecture dependent stage. In the architecture independent stage, parameters are extracted from the application, as explained in section 4.1. These parameters along with the range of target processors, application timing con-

Table 2. Extracted Application Parameters

Function	Average block size	"mac" operations	address/data instructions	IO/Total instructions	Average arc length	Average instr./cycle
do_flip_h	5.4	1	0.9	0.3	26.8	1.8
do_flip_v	5.7	1	0.9	0.2	40.3	1.5
do_transpose	3.7	1	1.0	0.2	37.4	1.3
do_rot_90	5.0	1	1.1	0.2	50.1	1.4
do_rot_270	5.1	1	1.1	0.2	45.6	1.5
do_rot_180	6.4	1	1.2	0.2	78.7	1.6
do_transverse	7.0	1	1.4	0.1	68.8	1.7
transpose_crit ical_parameters	7.5	1	1.6	0.2	16.0	2.0
trim_right_edge	3.1	0	0.3	0.3	12.2	1.4
trim_bottom_edge	3.1	0	0.3	0.3	12.2	1.4
jtransform_request _workspace	17.6	2	0.6	0.1	21.1	1.6
jtransform_adjust _parameters	4.3	0	0.3	0.1	31.0	1.5
jtransform_execute _transformation	49.0	0	0	0.1	2.4	4.2
jcopy_markers_setup	1.6	0	0	0.2	7.7	1.1
jcopy_markers_exec	2.2	0	0.3	0.3	60.6	1.1

straints and the profiling data are input to a processor selector. Processor selector rejects processors that don't meet the constraints. This selector will scale the various parameters based on certain gross characteristics of the processors, such as branch delay, presence/absence of "mac" operation, depth of the pipeline, memory bandwidth etc. The output is a smaller subset of processors. The scheduler takes the profiler output as input along with the application and the processor description files of the selected processors. After scheduling, this gives the performance estimate of each of the processors in terms of number of clock cycles. We have designed and implemented the application parameter extractor, the constrained scheduler and automated the process for extracting the key information from the profiler output. The processor selector part still remains to be implemented.

For the testing of our tools, we have used the functions from the jpeg code [9]. We have run our parameter extractor on certain functions of the jpeg code. Certain test images were taken and some transformations were carried out during the testing. Table 2 gives a summary of the application characteristic parameters that were extracted. Each column corresponds to the characteristics as described in section 4.1. Figure 4 shows the histogram generated by the ASAP scheduler on function 'do_transpose' of jpeg code. The histogram indicates the concurrency in the code. It can be noticed from the results that there is a lot of variation in all the parameter values. For example, the average block size ranges from 1.6 to 49 operations. Similarly, average arc length ranges from 2.4 to 60.6 cycles and the ratio of address computation instructions to data computation instructions ranges from 0 to 1.6. These experiments show that the range of these parameters is substantial and can help in selecting/rejecting processors in performance critical applications.

For the testing of the scheduler, we have captured instruction sets of two processors; Philips' TriMedia (a VLIW processor) and TI's TMS320C6x (a DSP processor). Presently, specialised instructions like SIMD are not captured. We ran the scheduler on the above functions of the jpeg code. The results of the cycle count are given in table 3. It can be seen that the TI cycle count is higher by 17% to 170% vis-a-vis the TriMedia cycle count. We have also performed the validation of the results for the TriMedia processor by using the proprietary tools of Philips. For most cases, our estimates are within 30% of the results obtained from the proprietary tools. We are analyzing rather large discrepancies in some other cases to improve our estimates.

6 Conclusions and Further work

In our work, we have developed a methodology for processor evaluation. We have attempted to do so through a

Table 3. Comparison of Actual and Estimated Cycle Counts

Function	Estimate TI	Estimate TriMedia	Actual TriMedia
jtransform_request _workspace	41	35	31
jtransform_adjust _parameters	44	28	34
jtransform_execute _transformation	64	23	20
jround_up	49	37	54
access_virt_barray	4581	3161	3180
free_pool	583	390	417
jinit_memory_mgr	209	159	116

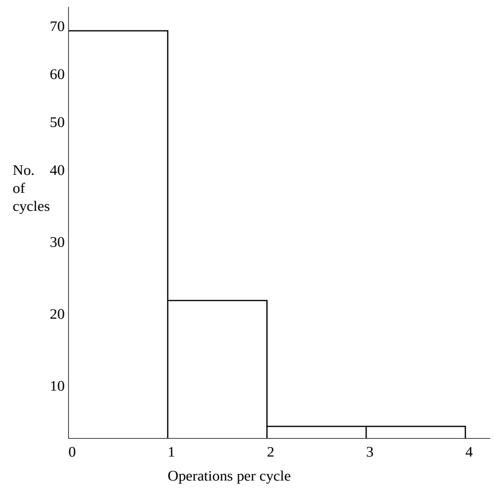

Figure 4. Histogram generated by ASAP scheduler on function do_transpose of jpeg code

two-stage process. The first stage is architecture independent while second stage does the architecture dependent performance estimation. We have developed and implemented the parameter extractor and the constrained scheduler. Preliminary results suggest that our approach has the potential to quickly eliminate unsuitable processors and evaluate modifications in ASIPs.

Our further plans include:

- Identify a larger set of application parameters and architectural features
- Strategies for choosing processors from parameter values and architectural features by estimating their influence and
- Increasing the range of supported processors

Finally, this methodology has to be applied on complete applications and results of performance estimation is to be compared with proprietary tools more extensively for validation. The methodology though applicable in general, is likely to be effective when the parameters and their relative influences are tuned to a specific application domain.

7 Acknowledgements

This research has been funded by a grant from the Naval Research Board, Government of India.

References

[1] S. Liao, et al. “Code generation and Optimization Techniques for Embedded Digital Signal Processors”, Proceedings of the NATO Advanced Study Institute on Hardware/Software Co-Design, In G. DeMicheli and M. Sami, editors, *Hardware/Software Co-Design*, Kluwer Academic Publishers, 1996, pp 165-186.

[2] G.Araujo and S.Malik, “Code generation for embedded DSP processors”, ACM Tr on Design Automation for Electronic Systems, 1999.

[3] Embedded Systems Methodology, Progress report for Aug98-Jan99, CSE Dept., IIT-Delhi, 1998-99.

[4] The SUIF Library, A set of core routines for manipulating SUIF data structures, SUIF Compiler System, Version 1.0, Stanford Compiler Group.

[5] Philips TriMedia SDE Reference I, Programming Languages and File Formats, 1998.

[6] TMS320C62xx CPU and Instruction Set, Reference guide, 1997.

[7] T.V.K.Gupta and Purvesh Sharma, “Processor Evaluation: Embedded Systems Methodology”, B.Tech. Project report, CSE Dept., IIT Delhi, May 1999.

[8] Giovanni De Micheli, Synthesis and Optimization of Digital Circuits, 1994, McGraw-Hill International Editions, Electrical Engineering Series.

[9] Independent JPEG’s Group’s software, ftp://ftp.uu.net/ graphics/jpeg/jpegsrc.v6a.tar.gz

[10] The GNU profiler, gprof, http://www.gnu.org/ manual/gprof-2.9.1/html_mono/gprof.html.