DESIGN SPACE EXPLORATION WITH AUTOMATIC GENERATION OF IP-BASED EMBEDDED SOFTWARE

Júlio C. B. de Mattos[1], Lisane Brisolara[1], Renato Hentschke[1], Luigi Carro[1,2], Flávio R. Wagner[1]

[1] *Federal University of Rio Grande do Sul, Computer Science Institute, Av. Bento Gonçalves, 9500 - Campus do Vale - Porto Alegre, Brasil;*

[2] *Federal University of Rio Grande do Sul, Electrical Engineering Av. Oswaldo Aranha 103 - Porto Alegre, Brasil*

Abstract: Automatic embedded software generation and IP-based design are good approaches to achieve a short design cycle due to stringent time-to-market requirements. But design automation must also consider application-specific requirements. This paper presents a mechanism for the automatic selection of software IP components for embedded applications, which is based on a software IP library and a design space exploration tool. The software IP library has different algorithmic implementations of several routines commonly found in different application domains. These routines have been characterized in terms of power, performance, and area, for a given architectural platform. The design exploration tool allows the automatic configuration of an optimized solution for a specific application, by selecting routines whose combination best match system requirements. Experimental results are presented and demonstrate that a very expressive design space can be explored with this approach.

Key words: Design Space Exploration; Embedded Software, IP Components.

1. INTRODUCTION

The fast technological development in the last decades exposed a new reality: the widespread use of embedded systems. Nowadays, one can find these systems everywhere, in consumer electronics, entertainment,

communication systems and so on. In embedded applications, requirements like performance, reduced power consumption and program size, among others, must be considered. Since different platforms and cores are available, precise technical metrics regarding these factors are essential for a correct comparison among alternative architectural solutions, when running a particular application.

To the above physically related metrics, one should add another dimension, which is software development cost. In platform-based design, design derivatives are mainly configured by software, and software development is where most of the design time is spent. But the quality of the software development also directly impacts the mentioned physical metrics.

Presently, the software designer writes the application code and relies on a compiler to optimize it. Compiler code optimizations for embedded systems have been traditionally oriented towards improving performance, reducing memory accesses or space [1,2,3], for instance targeting code to specialized architectures, reducing cache misses, or compressing code.

It is widely known that design decisions taken at higher abstraction levels can lead to substantially superior improvements. Software engineers involved with software configuration of embedded platforms, however, do not have enough experience to measure the impact of their algorithmic decisions on issues such as performance and power. Therefore, this paper proposes a more pragmatic approach, consisting in the use of a software library and a design space exploration tool to allow an automatic software IP selection. The software IP library contains alternative algorithmic implementations for routines commonly found in embedded applications, whose implementations are previously characterized regarding performance, power, and memory requirements on a given platform. A similar approach is followed in [6], but restricted to implementations of Simulink blocks.

By offering a range of algorithmic solutions for usual problems that may be critical in several applications, the designer may choose the solution that best matches particular application requirements. By exploring design alternatives at the algorithmic level, that offer a much wider range of power, performance, and memory size values, the designer is able to automatically find, through the exploration tool, corner cases that result in optimizations far better that those reported by later code optimizations. As a very important side effect, the choice of an algorithm that exactly fits the requirements of the application, without unnecessarily wasting resources, may allow a more efficient use of the underlying hardware, for instance reducing supply voltage, clock frequency, and area to a minimum.

This paper is organized as follows. Section 2 discusses related work in the field of embedded software optimization. Section 3 gives an overview of the target architecture. Section 4 presents our approach to design space

exploration, introducing the library with its characterization and the exploration tool. Section 5 presents experimental results, and, finally, section 6 draws conclusions and introduces future work.

2. RELATED WORK

Power-aware software optimization has gained attention in recent years. It has been shown [8] that each instruction of a processor has a different power cost. By taking these costs in consideration, a 40% power improvement obtained by code optimizations is reported [8]. Reordering of instructions in the source code has been also proposed [9], considering that power consumption depends on the switching activity and thus also on the particular sequence of instructions, and improvements of up to 30% are reported. In [10], an energy profiler is used to identify critical arithmetic functions and replace them by using polynomial approximations and floating-point to fixed-point conversions.

Recent efforts are oriented towards automatic exploration tools that identify several points in the design space that correspond to different trade-offs between performance and power. In [4], Pareto-optimal configurations are found for a parameterized architecture running a given application. Among the solutions, the performance range varies by a factor of 10, while the power range varies by a factor of 7.5. In [5], the best performance/power figures are selected among various application-to-architecture mappings.

In [6], a library of alternative hardware and software parameterized implementations for Simulink blocks that present different performance, power, and area figures is characterized. Our approach is similar, but instead of aiming at a partitioning between software and hardware functions, it concentrates on algorithmic variations of software routines that are commonly found in a wide range of embedded applications. This way, it provides design space exploration for given platforms.

3. TARGET ARCHITECTURE

The software library characterization has been performed for a platform based on a Java microcontroller, called femtoJava [7], although the methodology is general and can be applied for other processor architectures as well. The Java microcontroller implements a hardware execution engine through a stack machine that is compatible with the Java Virtual Machine (JVM) specification. A CAD environment that automatically synthesizes the microcontroller for a target application [7] is available, using only a subset

of instructions critical to the specific application. This way, the impact of algorithmic-level optimizations against compiler level optimizations can be measured.

4. THE PROPOSED APPROACH

Figure 1a shows a traditional design flow, where the designer receives the application specification and, after coding it in some language, compiles it for a chosen platform. In this approach, the designer must know the target platform, and all optimizations are trusted to the compiler.

In our approach, illustrated in Figure 1b, the design starts with an application specification at a high level of abstraction, and the application code is generated automatically by a tool. This tool allows design space exploration based on a software IP library, the application specification, and the designer knowledge. Note that, in this approach, the designer does not need to know the target platform, because this information is used only for the library characterization. This methodology allows the automatic selection of software IPs to better match a certain platform. Moreover, if the application constraints might change, for example with tighter energy demands or smaller memory footprint, a different set of SW IPs might be selected. The same reasoning applies when the underlying platform is changed.

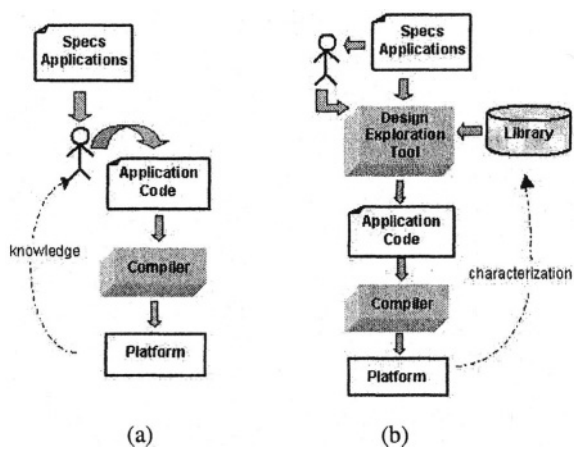

Figure 1. (a) Traditional approach (b) Our approach.

In this work, a configurable power estimation simulator, called CACO-PS [11], was used to collect all measures during library characterization. Specifically, considering a certain platform and for each algorithmic implementation of the library functions, it measures the performance (in

cycles), the memory usage (for data and instruction memories), and the power dissipation (in gate capacitances – GC – that switch during execution, considering CPU, RAM, and ROM). In the next sections, the software IP library and the design space exploration tool are presented in more detail.

4.1 Software IP library

As it has been already mentioned, the library contains different algorithmic versions of the same function, thus supporting design space exploration. Since embedded systems are found in many different application domains, this investigation has been started using classical functions: Sine – Two ways to compute the sine of an angle are provided. One is a simple table search, and the other one uses the CORDIC (Coordinate Rotation Digital Computer) algorithm [12]; IMDCT – The Inverse Modified Discrete Cosine Transform is a critical step in decompression algorithms like those found in MP3 players. Together with windowing, it takes roughly 70% of the processing time [13]. Others functions are implemented like Table Search, Square Root and Sort.

4.2 Library characterization

To illustrate the results of library characterization using different algorithmic versions of the same function, there are two routines selected: the sine and the Inverse Modified Discrete Cosine Transform. Table 1 shows the main results of the characterization of the four different implementations of the IMDCT function. The IMDCT4 implementation has the better results in terms of performance and power dissipation, but the size of program memory significantly increases. The opposite happens with the IMDCT1 implementation, which has far better results in terms of program memory, but consumes about 3 times more cycles and power.

Table 1. IMDCT characterization

Characteristic	IMDCT1	IMDCT2	IMDCT3	IMDCT4
Program size (bytes)	344	2,137	4,260	15,294
Data mem (bytes)	3546	3546	3546	3546
Performance (cycles)	140,300	97,354	92,882	51,345
Power (x 10^6 GCs)	1,149.43	940.83	1,021	905.89

Table 2 illustrates the characterization of the alternative implementations of the sine function. There are some entries in Table 2 that are pretty obvious. Since Cordic is a more complex algorithm, program memory size is larger than with Table Look-up, as well as the number of cycles required for computation. It is interesting to notice, however, that the data memory size

seems to be almost the same. This, however, is caused by the fact that data in Table 2 was obtained for a sine resolution of 1 degree. As the resolution increases, the amount of data memory increases exponentially for the Table Look-up algorithm, but only sublinearly for the Cordic algorithm. The increase in memory reflects not only in the required amount of memory, but also in the power dissipation of a larger memory.

Table 2. Sine characterization

Characteristic		Cordic	Table
Program size (bytes)		206	88
Data mem (bytes)	1°	184	220
	0.5°	184	400
	0.1°	184	1840
Performance (cycles)		2,447	136
Power (GCs)	1°	6,274,770	350,237
	0.5°	6,274,770	391,197
	0.1°	6,274,770	637,117

4.3 Evaluating a complete application

In all examples presented above, the design space concerning performance, power, and memory footprint was large. However, the availability of different alternatives of the same routine is just a first step in the design space exploration of the application software. One must notice that embedded applications are seldom implemented with a single routine. There is another level of optimization, which concerns finding the best mix of routines among all possible combinations that may exist in an embedded application.

In order to better illustrate the concept, let us take as an example the IMDCT function. Taking into account program and data memory sizes, performance, and power, there are 16 possible combinations of the four versions of the IMDCT and cosine functions. Some of them are very interesting, depending on particular application requirements:

- If memory space has the highest priority, one can combine the IMDCT1 core with a table look-up cosine calculation with a resolution of 1 degree. This requires only 3,730 bytes of data memory and 432 bytes of program memory, although it presents the worst figures for performance and power;
- If an application must respond in at most 200,000 cycles and the cosine calculation requires a high resolution (0.1 degree), then the best is to combine the IMDCT2 core and the CORDIC-based cosine calculation. This is the combination that fulfils the above restrictions and requires less

memory space (3,730 bytes of program memory and 2,343 bytes of data memory);

- If performance and power have the highest priority, combining the IMDCT4 core with a table look-up cosine is the best alternative. It requires only 56,242 cycles and 906.25×10^6 gate capacitances of power consumption.

4.4 Design space exploration tool

The Dragon Lemon tool maps the routines of an embedded program to an implementation using instances of the software IP library, so as to fulfil given system requirements. The user program is modeled as a graph, where the nodes represent the routines, while the arcs determine the program sequence. The weight of the arcs represents the number of times a certain routine is instantiated. It is also possible to model parallel routine calls, in case the underlying hardware has parallel processing capabilities.

To generate the application graph representing the dynamic behavior of the application, an instrumentation tool was developed. It is based on BIT *(Bytecodes Instrumentation Tool)* [14] that allows the dynamic analysis of Java Class files, generating a list of invoked methods with its corresponding number of calls, which can be mapped to the application graph.

In the exploration tool, before the search begins, the user may determine weights for power, delay and memory optimization. It is also possible to set maximum values for each of these variables. The tool automatically explores the design space and finds the optimal or near optimal mapping for that configuration. The cost function of the search is based on a trade-off between power, timing, and area. Each library option is characterized by these three factors. The exploration tool normalizes these parameters by the maximum power, timing and area found in the library. The user can then select weights for the three variables. This way, the search can be directed according to the application requirements. If area cost, for example, must be prioritized because of small memory space, the user may increase the area weight. Although one characteristic might be prioritized, the others are still considered in the search mechanisms. As output, Dragon Lemon also provides 2D and 3D Paretto curves. For both curves, the user may select which variables (power, delay, or memory) will be used in each axis (x, y and z).

5. RESULTS

Two sets of experiments have been executed. First, our methodology has been evaluated with small synthetic examples, but trying to address real applications, like an Address Book and a Game, running in parallel with a MP3 player. The second experiment shows that the design exploration tool is also able to explore much larger search spaces.

The application A_Book+MP3 is implemented as two parallel processes. The first one runs typical Address Book tasks, such as *table search, insert,* and *sort.* Calculator features, such as *square root* and *sine* calculation, have also been added. In parallel to this process, an MP3 player is executed. The design space exploration for the MP3 considers the different implementations of the IMDCT function, which is dominant in the MP3 decode routine. Two different architectural variations have been tried. In the first one, a single processor has been used. In the second option, one processor executes the MP3 algorithm, while the other processor deals with the other tasks of the Address Book. The results are shown in Table 4. For the option with equal weights for power, timing and area, the best solution is to use *hash* and *quick-sort* routines. However, this configuration is changed when the area weight increases. *Hash* is replaced with a *sequential search,* while the *quick-sort* is replaced with *insert-sort.* Table 4 also shows that increasing the number of processors does not significantly decreases the running time. This happens simply because the IMDCT calculation running time is much larger than the other tasks.

The Game application is implemented by three parallel processes. There is a rendering part, which allows the exploration of sine and square root routines. In parallel, there is the MP3 decoder part, with the IMDCT exploration. The third part comprises the game logic itself and AI computation, which will perform table searches, insertions, and sort functions. The difference from the previous application is that the parallel processes will have similar running times, while in the Address Book the IMDCT time determines the total running time. Results are also presented in Table 4. It is clear that this time we took advantage of additional processors in the architecture, because of the higher parallelism of tasks with equal complexity.

All results in Table 4 come from an exhaustive search, running in less than a second of execution time. Figure 3 shows the Paretto curve found for the first row of Table 4. The curve shows Running Time in the x-axis and Area in the y-axis. It is clear that by increasing the expected running time the designer is able to use smaller memory spaces.

Table 4. Results of design space exploration for Address Book + MP3 and Game*

Application	# proc	Weights P	T	A	Power (x 10^7)	Delay (x 10^7)	Memory Area
A_Book + MP3	1	1	1	1	36,23	20,54	20175
A_Book + MP3	2	1	1	1	36,23	20,52	20089
A_Book + MP3	1	1	1	10	36,23	20,54	20089
A_Book + MP3	2	1	1	10^4	37,63	38,93	6711
Game	1	1	1	1	40,10	43,69	20175
Game	3	1	1	1	40,10	20,48	20175

*Power in gate capacitances, delay in clock cycles, memory area in bytes.

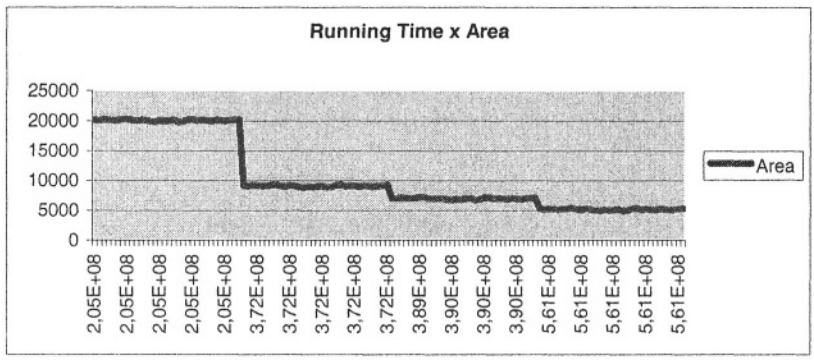

Figure 3. Paretto curve for first row of Table 4.

6. CONCLUSIONS AND FUTURE WORK

This paper proposed a new methodology for software IP selection in a design space exploration context, considering performance, power, and memory area requirements. It is based on software IP library that is previously characterized for a given architectural platform and uses a genetic tool for automatic design space exploration and IP selection.

Experimental results have confirmed the hypothesis that there is a large space to explore based on algorithmic decisions taken at higher levels of abstraction, much before compiler intervention. Selecting the right algorithm might give orders of magnitude of gain in terms of physical characteristics like memory usage, performance, and power dissipation. As a future work, we plan to enlarge the library and to investigate the impact of different memories with different power-delay products, so that one can better tune the algorithms to the underlying platform.

REFERENCES

[1] N.Dutt, A.Nicolau, H.Tomiyama, A.Halambi. "New Directions in Compiler Technology for Embedded Systems." *Asia-Pacific Design Automation Conference,* Jan. 2001. Proceedings, IEEE Computer Society Press, 2001.

[2] V.Dalal, C.P.Ravikumar. "Software Power Optimizations in an Embedded System". VLSI Design Conference, Jan. 2001. Proceedings, IEEE Computer Science Press, 2001.

[3] M.Kandemir, V.Vijaykrishnan, M.J.Irwin, W.Ye. "Influence of Compiler Optimizations on System Power". In: IEEE Transactions on VLSI Systems, vol. 9, n. 6, Dec. 2001.

[4] T.Givargis, F.Vahid, J.Henkel. "System-Level Exploration for Pareto-optimal Configurations in Parameterized Systems-on-a-chip". ICCAD'01 - International Conference on Computer-Aided Design, San Jose, Nov. 2001.

[5] A.Nandi. R.Marculescu. "System-Level Power/Performance Analysis for Embedded Systems Design". Design Automation Conference, Las Vegas, June 2001. Proceedings, ACM, 2001.

[6] L.M.Reyneri, F.Cucinotta, A.Serra, L.Lavagno. "A Hardware/Software Co-design Flow and IP Library Based on Simulink". DAC'01 - Design Automation Conference, Las Vegas, June 2001. Proceedings, ACM, 2001.

[7] S.Ito, L.Carro, R.Jacobi. "Making Java Work for Microcontroller Applications". In: IEEE Design & Test of Computers. vol. 18, n. 5, Sept-Oct 2001.

[8] V.Tiwari, S.Malik, A.Wolfe. "Power Analysis of Embedded Software: a First Step Towards Software Power Minimization". In: IEEE Transactions on Very Large Scale Integration *(VLSI) Systems,* vol. 2, n. 4, Dec. 1994.

[9] K.Choi, A.Chatterjee. "Efficient Instruction-Level Optimization Methodology for Low-Power Embedded Systems". International Symposium on System Synthesis, Montréal, Oct. 2001. Proceedings, ACM, 2001.

[10] A.Peymandoust, T.Simunic, G. de Micheli. "Complex Library Mapping for Embedded Software Using Symbolic Algebra". DAC'02 - Design Automation Conference, New Orleans, June 2002. Proceedings, ACM, 2002.

[11] A.C.Beck Filho, F.R.Wagner, L.Carro. "CACO-PS: A General Purpose Cycle-Accurate Configurable Power Simulator". SBCCI'03 – 16th Symposium on Integrated Circuits and Systems Design. São Paulo, Brazil, Sept. 2003. Proceedings, IEEE Computer Society Press, 2003.

[12] A.Omondi. Computer Arithmetic Systems: Algorithms, Architecture and Implementation. Prentice Hall, 1994.

[13] K.Salomonsen, S.Søgaard, E.P.Larsen. Design and Implementation of an MPEG/Audio Layer III Bitstream Processor, Master Thesis, Aalborg University, 1997.

[14] H.B.Lee, B.G.Zorn. "BIT: A Tool for Instrumenting Java Bytecodes". USITS'97 - USENIX Symposium on Internet Technologies and Systems, Dec. 1997.