Exploring The Algorithmic Design Space using High Level Synthesis

Miodrag Potkonjak, Jan Rabaey

HYPER is a high level synthesis system, targeted at numerically intensive applications. By shifting the emphasis from the traditional high level synthesis tasks (such as scheduling and assignment) to the domain of transformations, new venues for high level synthesis are opened. One of the most exciting among them, with potentially the largest impact on the quality of the design, is design and selection of the algorithms for a given application.

After a brief overview of the HYPER system, we concentrates on the exploration of the algorithmic design space. We show how HYPER can improve the performance or cost of real life applications with orders of magnitude by guiding and conducting a proper algorithmic design selection process.

1. HYPER

Real time applications in areas such as communications, speech, image and video processing, radar, sonar and computerized tomography often require high performance dedicated datapaths. The structure of the datapath is strongly correlated to the structure of the computations. The controller is small. The synthesis of this type of architecture is an involved and meticulous process, which implies a strong need for sophisticated CAD tool support. HYPER addresses exactly this class of numerically intensive algorithms. A detailed description of the HYPER system is given elsewhere [1].

The input to HYPER is a description of application in an applicative signal-flow language Silage [1]. The Silage description is translated into an intermediate CDFG (control data flow graph) format, which serves as the central repository, on which all synthesis tools are operating and all results are annotated. The tool library includes tasks, such as simulation, hardware module selection, transformations, allocation, assignment, scheduling and background memory management. Although several high level synthesis systems [2, 3, 4] use one or more transformations during synthesis, HYPER is the only one which treats them in systematic optimization intensive fashion. HYPER currently uses three types of transformations with respect to their scope: suboperational level (e.g. substitution of multiplication by add/shifts), basic block (all algebraic and redundancy manipulation transformations) and control structure (loop unfolding, loop expansion and software retiming and pipelining) transformations.

The single most important future of HYPER is the synthesis manager program, which supports the exploration of the design space using classical high level synthesis tools, transformations and a set of estimation and feedback information tools. The synthesis manager uses a single global design quality measure, called the resource utilization, to direct the design space exploration. It also guides an ergonomic user interface, which facilitates user interaction. Once a final result is obtained, HYPER maps the annotated CDFG into an architectural description, which is translated into layout using the LAGER silicon compiler [5].

2. ALGORITM DESIGN AND SELECTION

Even a superficial, brief survey of the algorithm design research in several important application areas reveals that a large variety of algorithms exist to address a single problem. For example. for fast Fourier transform (FFT) and discrete cosine transform (DCT), which are often used and many DSP areas, there exist dozens of fast algorithms [6].

In some cases, there is an obvious superiority of one algorithm over others in terms of performance. However, more often the optimality of an algorithm depends upon objective function and constraints. Although the selection of the right algorithm for a targeted task has a major impact on the final quality of the design, this area is currently more art than science or engineering and designers almost exclusively rely on intuition instead of accurate quantitative procedures. This intuitive approach is however bound to break down. Progresses in technology have increased the application complexity and have made parallel processing a more viable alternative. Selecting the right algorithm within this setting is becoming increasingly intractable for a human designer.

Algorithm Design

Transformations are a powerful tool for the exploration of parallelism. However, their effect is obviously constrained by the initial algorithm's computational structure. Therefore, the only tools with more potential than transformations are located in the area of algorithm design and selection.

An obvious goal in this area is the automatic design of algorithms (from a given specifications), such that the resulting CDFG can be easily transformed into a final implementation with maximal performance or minimal cost. It is very likely that this goal will remain elusive for years to come. However, it appears that there exist several ways to do efficient algorithm design for restricted application domains.

For instance, the algorithms used in wide classes of applications have a substantial freedom in their computational structure. This makes them prime candidates for a search to determine, which structures possess the most parallelism. Consider for instance the class of algorithms, which address the iterative solution of sets of equa-

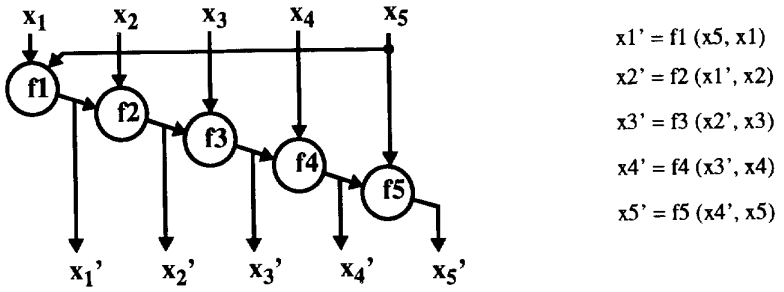

Figure 1: Parallelism of Gauss-Seidel iterations: The initial updating order.

tions. Examples of such algorithms are shown in Figure 1 and Figure 2. They all have the following structure:

$$x(t+1) = f(x(t)), \quad t = 0, 1, \quad ... \tag{1}$$

where each $x(t)$ is an n-dimensional vector and $f(.)$ is some function which has as domain and range some subset of the n-dimensional space. If the sequence $\{x(t)\}$ generated by the above iteration converges to a *point* x^*, and if the function $f(.)$ is continuous, then x^* is a fixed point of f, which satisfies the relationship $x^* = f(x^*)$. For example, iterative methods are often used for the solution of sparse system of equations, or for the maximization and minimization of a function by search for the zeroes of the derivatives.

When all the components of x are updated simultaneously, the method is often called the Gauss-Jacoby iteration. An alternative approach is to update one equation at a time. Equation (1) can then be expressed in the following form:

$$x_i(t+1) = f_i(x_1(t+1), ..., x_{i-1}(t+1), x_i(t), ..., x_n(t)), \quad i = 1, ..., n \tag{2}$$

This type of iteration is often called the Gauss-Seidel (GS) iteration and is illustrated in Figure 1 for one particular problem instance. Since the GS algorithm incorporates the most recent information at each step [1], it often converges faster than Gauss-Jacoby iterations.

In the GS iterative algorithm, the order of updating is not fixed. Instead of starting from x_1 and proceeding forward, we can permute the updating order (Figure 2). Of course, this results in a new algorithm with different convergence properties. Nevertheless, all GS algorithms will converge to the same fixed point after a number of iterations. By analyzing the speed of convergence and the available parallelism, we can, for a particular application instance, design the algorithm, which is the best

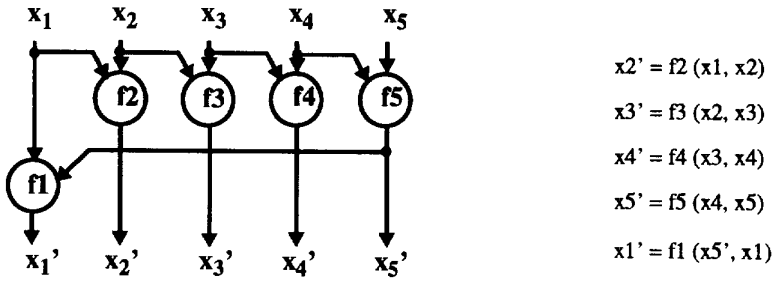

Figure 2: Parallelism of Gauss-Seidel iterations: The increase in parallelism and the reduction of the critical path due to the change in the updating order.

suited for implementation. For example, it is easy to see that the algorithm of Figure 2 (A2) has a significantly shorter critical path than the algorithm in Figure 1 (A1). Using HYPER we found that on the system of equations $\{x_1 = 2x_5 - x_1^2;\ x_2 = 4x_1^5 - x_2x_1;\ x_3 = x_3^2 - 3x_2;\ x_4 = x_4^3 - x_3 - 3;\ x_5 = 3x_5 - x_4.\}$ the algorithm A2 needs 14.2 times less hardware than the algorithm A1 for the same speed of convergence.

Similar algorithm design selection techniques can be used in a variety of other algorithms. They include algorithms for the solution of ordinary differential equations (such as the Runge-Kutta methods), partial differential equation solvers and various minimization and maximization algorithms. One especially interesting domain is the class of probabilistic optimization methods.

From the above, it becomes obvious that, in order to be effective and realistic, an algorithmic designer needs a set of tools, which can help him to predict or analyze the speed of convergence (and other performance properties, e.g. numerical stability) and the implementation properties of a proposed algorithm. While simulation (or closed form analysis for some application classes) is the preferred medium for the former task, tools like HYPER can help to predict and analyze the latter.

Algorithm Selection and Tuning: Proof of Concept Example

We propose the following procedure, which helps a designer to select of the most promising algorithm for a given problem instance over a set of goals and constraints.

Given a set of potentially useful algorithms $\wp$, the first step is to identify the so called *non-inferior* algorithms. An algorithm A is called inferior to an algorithm B, if the following criteria are fulfilled:

(1) Algorithm B uses fewer operations of all types than algorithm A;

(2) Algorithm B has a shorter critical path and shorter iteration bound than algorithm A;

(3) According to all other available lower bound and statistical estimations, algorithm B is always superior to the algorithm A. (For this task we use HYPER).

An algorithm A is then called non-inferior, if it is not inferior with respect to all algorithms in $\wp$.

For all non-inferior algorithms, we use branch and bound (B&B) techniques to select the best one for the given set of goals and constraints. During B&B, the implementation cost is estimated using the lower bound estimation techniques, available in HYPER. To reduce the impact of the initial algorithm specification, the algorithms are optimized and tuned over a set of transformations. Since HYPER employs fast estimation and synthesis techniques, usually one afternoon is sufficient for the process of algorithm selection and tuning

In this section, the potentials and the impact of the algorithm selection technique will be demonstrated using an eight order Avenhaus Bandpass filter. It will be shown that transformations are an integral part of the algorithm selection process and that ignoring this point can result in inferior solutions or wrong decisions. Also, it will be demonstrated that algorithm selection is a complex process and that the results most often deviate dramatically from intuitive insights.

The filter under study in this paper was first presented by Avenhaus [7] and has often been used in the digital filter research and practice. For example, Crochiere [8] presented in an depth discussion of several digital filter structures, which can implement the required frequency response. He compared the structures according to their statistical coefficient word length, the required number of operations and the amount of the parallelism and serialism. Although this analysis and presentation is prototype example of excellency in research, we will show that the presented measures are far from being sufficient to select a proper structure for ASIC implementation. Actually, our analysis of the example implicates that this manually conducted procedure often leads to misleading conclusions.

We consider the following five structures of the Avenhaus filter, designed by Crochiere [8]: direct-form II (DF), cascade form (CAS), parallel form (P), continued fraction form type 1B (CF) and ladder structure (L) [8]. We limited our analysis to the five mentioned structures, since our goal is not to find the "ultimate" structure for the implementation of the Avenhaus eight order bandpass filter over all structures ever proposed, but to present a procedure to classify arbitrary computational structures given a set of constraints and implementation goals.

Experimental Results

All five structures were described using the Silage and passed to the HYPER environment. Table 1 (assembled using data from [8]) shows the number of multiplications and additions and the statistical word length for all five forms. We simulated all five examples in fixed point precision in all phases of the design process, and indeed, all of them produced the required frequency response. A small correction was needed for the direct form II, as, due to a typographical error, two coefficients were interchanged in [8].

structure	number of multiplication	number of additions	statistical word length
direct form II	16	16	21
cascade	13	16	12
parallel	18	16	11
CF	18	16	23
ladder	17	32	14

Table 1: Number of operations and Statistical Word Length.

structure	total	additions	shifts	subtractions	transfers
direct form II	214	58	103	46	7
cascade	93	31	40	18	4
parallel	112	33	51	24	4
CF	192	54	95	43	-
ladder	115	35	49	31	-

Table 2: Number of operations after Multiplication Expansion. The transfer operations are register-register transfers, needed for the implementation of delays.

Table 2 shows the number of operations in the five structures after the multiplication strength reduction. While, of course, there is a correlation between the number of multiplications and the word length in the initial form and the number of shift-operations after the expansion. However, only a precise analysis can determine the final effect of the application of the multiplication strength reduction. For example, although the direct form has both fewer multiplications and a shorter word length

than the continuous fraction, the latter eventually requires fewer shifts and a smaller number of operations.

Table 3 shows the length of the critical path for all five forms in the initial format and after the application of retiming and pipelining. All examples are implemented using the word lengths as indicated in Table 1. We decided to use only those two transformations, because they never alter the bit width requirements. During the optimization for the critical path, we used the Leiserson-Saxe algorithm for both retiming and pipelining [9]. During the optimization for area, retiming and pipelining for resource utilization was used [1].

structure	initial	retimed	pipelined
direct form II	980	686	686
cascade	527	341	279
parallel	609	522	261
CF	2332	1908	1908
ladder	2835	2835	630

Table 3: Critical path for the Avenhaus Eight Order Bandpass Filter (in nsec - for a 2 μm technology).

In the initial structures, the ratio in critical paths between the fastest (cascade) and the slowest (ladder) structures equals 5.4. When no extra latency is allowed, this ratio becomes even higher (8.3) as the cascade structure is very amenable to the retiming, while this is clearly not the case for the ladder filter. If we allow the introduction of one pipeline stage, the parallel form becomes the fastest structure. It is important to notice that, when throughput is the major concern, other filter structures can be conceived, which result in even smaller critical paths. For example, the recently proposed maximally fast implementation of linear computations [10] reduces the critical path to only 174 nano-seconds without introducing any additional latency. This is the improvement by the factor of 16.3. over the original ladder structure. When additional latency is allowed, this factor can be improved to arbitrarily high levels [10].

Table 4 shows the area of the final implementation for the five structures for six different sampling periods. For all five forms, results are shown for both the original and the transformed structures. The ratio between the largest and smallest implementations is even higher than the performance ratios. For example, for a sampling period is 1 microsecond, only the direct, cascade and parallel forms are feasible alternatives. The ratio of the implementation areas between the direct form and the cascade form equals 12.9. This ratio is increased to 15.1 for the retimed cascade

form. Interestingly enough, there is not a single instance were pipelining improved the area requirements. This is the consequence of the fact that in all designs, except for the fastest implementations, the major part of the area cost is located in the registers and not the execution units or the interconnect.

structure	sampling period (in μsec)					
	5	4	3	2	1	0.5
df II I	20.32	20.32	22.88	26.36	92.49	-
df II PR	19.86	19.86	21.75	30.65	53.98	-
cascade I	6.63	6.63	6.63	6.63	8.97	-
cascade P	5.98	5.98	5.98	5.98	6.11	11.36
parallel I	6.71	6.71	6.71	6.71	8.67	-
parallel R	5.92	5.92	5.92	5.92	7.16	-
parallel P	5.92	5.92	5.92	5.92	7.16	13.42
CF I	14.65	22.67	-	-	-	-
CF RP	13.56	19.54	27.19	-	-	-
ladder IR	5.73	7.38	-	-	-	-
ladder P	5.73	7.38	10.77	16.87	-	-

Table 4: Implementation Area for six different throughput requirements.
I - initial structure; R - retimed structure; P - pipelined structure.

Looking again at Table 4, we see that, depending upon the required throughput, the minimal area is obtained by different structures. For the fastest designs (when throughput rate is higher than 1 MHz), the pipelined cascade (when additional latency is allowed) and the parallel forms are the smallest solutions. For medium speeds (sampling period between 1 and 4 μs) the parallel form achieves the smallest area. Finally and most surprisingly, when the throughput requirements are the least strict, the ladder form is the most economical implementation. Ladder form does not neither have the smallest bit width nor the fewest number of operations. However, its regular and balanced structure requires few registers and few interconnects, which results in a slightly smaller area than other implementations.

It is very important to observe that the previous analysis does not indicate that any form is a-priory superior in terms of speed or area. The results depend strongly upon the required frequency response, the applied transformations, the objective

function (e. g. power or testability) or the performance parameters (for instance signal to noise ratio or overflow behavior). Our point is that for a given set of goals and constraints, only a quantitative analysis such as proposed in this paper, can provide a more qualified view, which can help designers in making the proper decisions during the design process.

All results presented in this section, if not otherwise stated, were obtained using the HYPER system. All results were generated and analyzed in a time span of two hours, what demonstrates that efficiency of HYPER is high enough to address the algorithm selection and tuning problem.

3. SUMMARY

The concept of algorithmic design space exploration is introduced and its importance for the quality of the final implementation is established. It is demonstrated how conducting the design selection process in the high level synthesis system HYPER improves the performance or the cost of implementation for a given application by more than an order of magnitude.

bREFERENCES

[1] J. Rabaey, at al. Fast Prototyping of Data Path Intensive Architecture. *IEEE Design and Test*, 8(2): 40-51, June 1991.

[2] H. Trickey. Flamel: A high-Level Hardware Compiler. *IEEE Transaction on CAD*, 6(2):259-269, February 1987.

[3] R.A. Walker and D.E. Thomas. Behavioral Transformation for Algorithmic Level IC Design. *IEEE Trans. on CAD*, 8(10):1115-1127, October 1989.

[4] G. Goossens, J. Rabaey, J. Vandewalle and H. De Man. An Efficient Microcode Compiler for Application Specific DSP Processors. *IEEE Transaction on CAD*, 9(9):925-937, 1990.

[5] R.W. Brodersen, ed. *Anatomy of a Silicon Compiler*. Kluwer, 1992.

[6] R. E. Blahut. *Fast Algorithms for Digital Signal Processing*. Addison-Wesley, 1985.

[7] E. Avenhaus. On the design of digital filters with coefficients of limited word length, *IEEE Trans. on Audio and Electroacoustics*. 20(2):206-212, April 1972.

[8] R. Crochiere and A. V. Oppenheim. Analysis of Linear Networks. *Proc. of the IEEE*, 63(4)581-595, April 1975.

[9] C.E. Leiserson and J.B. Saxe. Retiming Synchronous Circuitry. *Algoritmica*, 6(1):5-35, 1991

[10] M. Potkonjak and J. Rabaey. Maximally Fast and Arbitrarily Fast Implementation of Linear Computations. In *Proc. ICCAD*, pages 304-308, 1992.

[11] R. Camposano and R.A. Walker. *A Survey of high-level synthesis system*. Kluwer, 1991.

[12] M.C. McFarland, A.C. Parker and R. Camposano. The High-Level Synthesis of Digital Systems. *Proceedings of the IEEE*, 78(2): 301-317, February 1990.

Miodrag Potkonjak
C&C Research Laboratories, NEC USA
4 Independence Way
Princeton, NJ 08540
USA
e-mail: miodrag@ccrl.nj.nec.com

Jan Rabaey
EECS Department
University of California
Berkeley, CA 94720
USA
e-mail: jan@zabriskie.berkeley.edu