Algorithm Selection: A Quantitative Optimization-Intensive Approach[1]

Miodrag Potkonjak
Dept. of Computer Science, University of California, Los Angeles, CA 90095

Jan Rabaey
Dept. of EECS, University of California, Berkeley, CA 94720

Abstract

Hardware-software codesign has been widely accepted as a backbone of future CAD technology. Implementation platform selection is an important component of hardware-software codesign process which selects, for a given computation, the most suitable implementation platform. In this paper, we study the complementary component of hardware-software codesign, algorithm selection. Given a set of specifications for the targeted application, algorithm selection refers to choosing the most suitable algorithm for a given set of design goals and constraints, among several functionally equivalent alternatives. While implementation platform selection has been recently widely and vigorously studied, the algorithm selection problem has not been studied in CAD domain until now.

We first introduce algorithm selection problem, and analyze and classify its degrees of freedom. Next, we demonstrate an extraordinary impact of algorithm selection for achieving high throughput, low cost, and low power implementations. We define the algorithm selection problem formally and prove that throughput and area optimization using algorithm selections are computationally intractable problems. We also introduce an efficient technique for low-bound evaluation of the throughput and cost during algorithm selection and propose a relaxation-based heuristic for throughput optimization. Finally, we present an algorithm for cost optimization using algorithm selection.

For the first time, hardware-software codesign is treated using explicit quantitative, optimization intensive methods. The effectiveness of methodology and proposed algorithms is illustrated using real-life examples.

1. Preliminary versions of this work appeared in the *Proc. IEEE/ACM ICCAD International Conference on Computer Aided Design*, November 1994 and in *Proc. 1996 IEEE ICASSP - International Conference on Acoustic, Speech, and Signal Processing*, June 1995.

1.0 Motivations

1.1 Hardware-Software Codesign: Scope, Relevance and Issues

There is a wide consensus among CAD researchers and system designers that rapidly evolving system-level synthesis research, and in particular hardware-software codesign, is a backbone of future CAD technology and design methodology. Numerous comprehensive efforts by researchers with widely differing backgrounds have been recently reported [Sri91, Cho92, Kal93, Sal93, Tho93, Wol93, Gup94, Ver94, Woo94, Wol94].

At a very coarse level of resolution, hardware-software codesign is usually associated with assignment (partitioning) of various blocks of the targeted computation to either a programmable architecture or to a custom hardwired ASIC, so that the implementation cost of the system is minimized while the given performance criteria are met. At a finer level of resolution, the current efforts in hardware-software codesign address a selection of a specific implementation platform (e.g. a particular model of a particular DSP processor or a specific type of ASIC design generated with a specific set of tools and specific design methodology). Figure 1 graphically illustrates a typical instance of the implementation-platform selection problem on a very

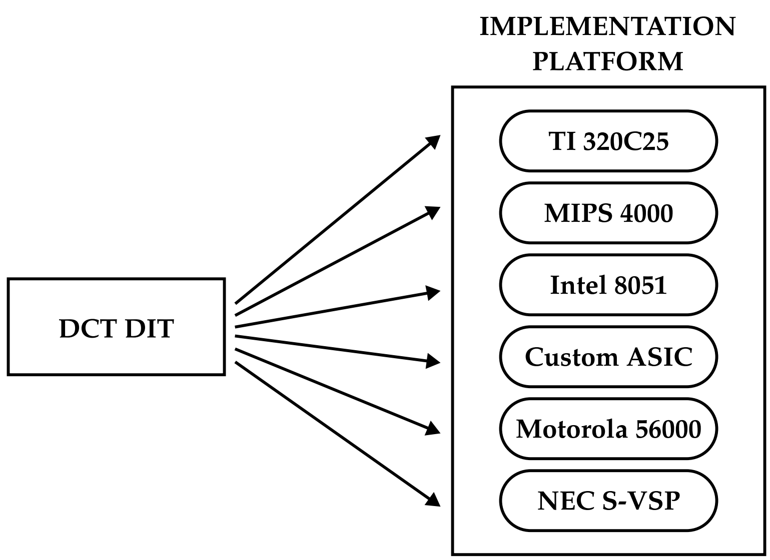

Figure 1: Implementation Platform Problem: For a given algorithm (Decimation in Time DCT) select the most suitable hardware or software platform.

small example of selecting a right platform for realization of Discrete Cosine Transform (DCT). In general, for a various targeted level of performances different platforms are most economic solution with respect to, for example, silicon area or power requirements (see Section 2.3).

Figure 2 recasts the hardware-software codesign task from a more global perspective. It is recognized that for a specified functionality of the application (DCT) the designer does not have only choice of implementation platform, but also may select among a variety of functionally equivalent algorithms. All algorithms are derived

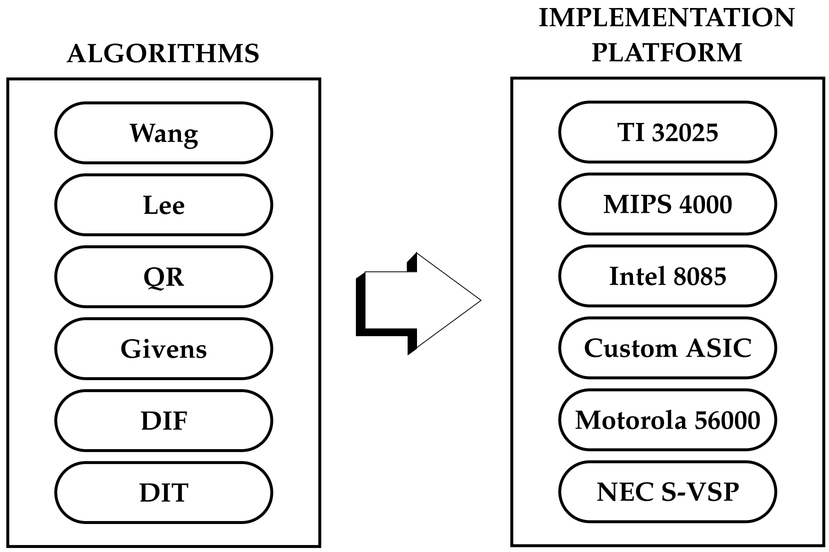

Figure 2: System Level Design: The task is to select the best match between several algorithm for a given application (DCT) and corresponding best platform among several implementation platforms

by noted researchers and are based on application of deep mathematical knowledge (see Section 2.3). The automatic generation of the algorithms is far beyond capabilities of current synthesis and compilation systems which use only basic algebraic and redundancy manipulation laws as transformation mechanisms and either simple scripts or generic. search algorithms for transformations ordering. As it is documented in the next

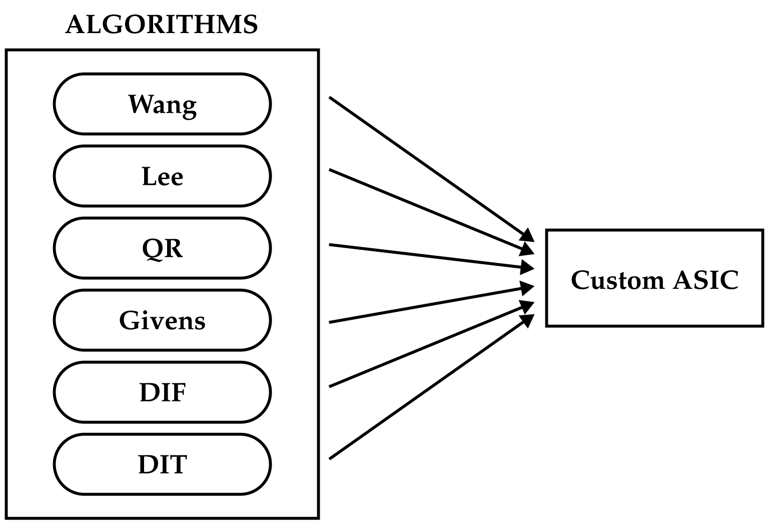

Figure 3: Algorithm Selection problem: For a given hardware (or software) platform and given application (DCT) select the most suitable algorithm for a given set of goals and constraints

section, even on this small example, a properly selected algorithm reduces a silicon area of implementation or power by more than an order of magnitude for a given level of performance. This spectrum of differences in the resulting quality of the design is often significantly higher than the difference produced by using different synthesis and compilation tools.

Considering simultaneously selection of both algorithm and architecture is, obviously, often formidable and cumbersome problem. As the first step, we study the more restricted, algorithm selection problem, shown in Figure 3. The goal is to select an algorithm which optimizes a given design goal for a given set of design constraints. The fixed implementation style (architecture) is assumed. The restricted problem is interesting in its own. As it is presented in Sections 3 and 4 many of results for the algorithm selection problem, can be easily generalized and applied on the overall algorithm-architecture selection problem.

1.2 Research Goals and Paper Organization

We have two primary goals. The first is to establish the importance of the algorithm selection task as a design and CAD optimization step, and give an impetus to development of this important aspect of system level design. The proper realization of this goal will result in a more complete picture of system-level design and its role in design and CAD-based optimization process. The second goal is to build foundations, design methodology, CAD environment, and specific algorithms which support effective algorithm selection. To realize this goal, we treat algorithm selection using quantitative, optimization intensive methods.

The paper is organized in the following way. We first describe importance, key issues and CAD requirements for the effective support of algorithm selection process. We start the study of algorithm selection by classifying sources of structurally different functionally equivalent algorithms. Next, we identify key components (estimations and transformations) of CAD system needed for proper support of algorithm selection.We establish a focused and tractable formulation of the algorithm selection problem at sufficiently high level of abstraction, so that the optimization task can be handled using efficient algorithms. At the same time the abstraction level is detailed enough that improvements achieved at the high level of abstraction indeed correspond to improvements in the final implementation.

In Section 3 we study optimization of throughput using algorithm selection. We analyze computational complexity of the optimization problem, derive sharp lower bound techniques, and present relaxation-based heuristic for solving the problem. In the similar fashion in Section 4 the algorithm selection is analyzed as a tool for cost optimization. We conclude the paper by comparing the presented work with related research and outlining directions for future research.

2.0 Algorithm Selection: Issues and Classification

2.1 Why black-box algorithms exist and why there are many different black box algorithms for a given application?

Transformations are changes in the structure of a computation, so that the functionality (the input/output relationship) is preserved. It is sometimes argued that all algorithms for a given set of functional specifications can be derived using the automated application of transformations. In this subsection we present four sets of conditions, which indicate that the existence of a variety of different, functionally equivalent algorithms are an unavoidable reality and that their importance must be recognized outside of the role of transformations. Actually, as it will be demonstrate through the paper, transformations and algorithm selection are distinct, but closely related system level synthesis tasks. For the highest quality implementations both tasks have to be considered, as well as the interaction between them.

We conclude this subsection by itemizing classes of sources which dictate a need and cause existence of different, functionally equivalent algorithms.

1. **[Exceptionally] clever use of algebraic and redundancy manipulation transformations.** For many commonly used computational procedures algorithm developers invested significant efforts to derive a specific set of transformations and their specific ordering which are exceptionally effective only on the targeted functionality. A deep, specialized mathematical and theoretical computer science knowledge is applied using a sophisticated and the problem instance tailored sequence of transformations steps.

   One of the most illustrative examples of this type of functionally equivalent algorithms are families of Winograd's fast Fourier algorithms (WFT) and recent Feig's and Duhamel's two dimensional DCT [Bla85, Win78, Fei92]. Their derivation is based on the use of matrix factorization and tensor algebra. Other typical examples include a great variety of fast sorting and convolution algorithms [Bla85].

   Derivation of all mentioned algorithms is far beyond the power of current compiler and behavioral synthesis system. Note that even in rare cases when the algorithms can be automatically synthesized, the algorithm selection task is still pending.

2. **Different approximations and use of semantic transformations.** Often the user specifies application in such a way that there is no exact corresponding control-data flow graph (CDFG), or the user requirements are such that allow a margin of tolerance. For example, many compression schemes are lossy, i.e. they do not preserve all information components, but only one which value is larger than the user specified threshold [Pen93]. Another interesting example in this category is IIR filters in digital signal processing literature [Mit93]. A great variety of transformations conducted in frequency domain are used for derivation of the various commonly used structures. Many of the transformations are based on the use of a variety of approximations [Mit93].

   Closely related to the class of algorithms which represent different approximations within a given set of margins of tolerance with respect to different parameters, are algorithms which are obtained using semantic transformations. Semantic transformations are transformations which are particularly effective when the algorithm is run on sets of data with a particular structures. Currently, semantic transformations are most often in algorithms for queries in databases [Ulm89], although several applications of semantic transformations were also reported in design of application specific hardware [Lop91].

3. **Different solution mechanism**. Often there exist algorithms which are based on different conceptual approaches for solving a given problem. For example, quick sort, which uses random selection of a pivoting element, can not be transformed in insertion or bubble sort, which are fully deterministic [Knu73]. All three algorithms, of course, produce the identically sorted data sets, but the last two are based on different solution mechanism than the quick sort.

   Other typical examples are compression algorithms and text search algorithms. For instance, vector quantization, transform based methods (e.g. DCT), fractal and wavelet based algorithms are based on different algorithms, and even when lossless compression is targeted, they result in different CDFGs.

4. **Use of higher levels of abstraction during the specification of the application.** In many situations the user is not specifically interested that the output has a specific exact form, as long as it satisfies some properties. For example, as long as all statistical tests are passed, the user is insensitive to a specific values of the sequence of numbers generated by any of several random-number generators. For example, although random-numbers generators of uniformly distributed random numbers between 0 and 1 generate different sequences of random numbers, all random numbers generators are equally acceptable, as long as the generated sequences are generating uniformly distributed numbers between 0 and 1.

   A similar situation is in many widely used applications. Most often the only relevant parameter in compression of stream of data, is the compression level. All compression schemes, which may result in a variety of compressed strings, are equally good at functional level as long they achieve requested level of compression. A similar situation is in cryptography applications where schemes such as various instance of RSA and DES [Sim91] are satisfying user requirements for security, although different outputs are produced.

2.2 How many different algorithms are there for a given application?

A natural and interesting question related to the algorithm selection problem is to ask how many different algorithms are (usually) available as the candidates for a given functionality. The number of options for a given functionality is interesting not only from a theoretical point of view, but also has a number of ramifications during the exploration of algorithm selection in synthesis process. For example, different optimization approaches are most appropriate when very few options are available in comparison to when a large number of functionally equivalent algorithms are available.

The analysis of a number of applications indicates that the number of instances is case dependent, and that each of the following three cases is not uncommon. The same survey shows, however, that the first scenario is, at least currently, dominating in practice. Therefore, we will mainly concentrate on this case.

1. **Few algorithms.** In a majority of cases there exist several algorithms, often with sharply different structural characteristics. Usually, each of the options is superior to all others in one (or sometimes few) criteria, which were explicitly targeted during the development of the algorithm. For example, the algorithms are often designed so that the smallest number of multiplications is used, or that the smallest number of all operations is used, or that have shortest critical path, or that the algorithm has exceptional numerical properties. Sometimes a particular architectural properties (e.g. multiplier-accumulator (MAC)) are targeted, or some structural property is the guiding principle (locality of data-flow or regularity).

2. **Finitely many, but vary large number of different algorithms.** In few cases a large number of different algorithms are available. This is most often the case when a specific design principle is discovered, and its application results in a large number of algorithms which can be generated. A typical example from this category is Johnson-Burrus Fast Fourier Transform [Joh83]. This algorithm is used to develop a large number of Fast Fourier Algorithms by combining small FFT algorithms. In particular to get Johnson-Burrus FFT one writes a Fourier Transform as the matrix product between the matrix of coefficients W and the vector of input data v.

$$F = W^* v$$

Johnson and Burrus showed that using matrix factorization W can be represented as the product of matrices W' and W", and they can be represented in the following forms:

$$W' = (G'H')\,B'\,(E'F')$$

$$W'' = (G''H'')\,B''\,(E''F'')$$

The reason for this factorization is that the resulting matrices have very few non-zero entries, and that many of entries are equal to +1 or -1. The interesting point now is that the product W' * W" can be rearranged by permuting the order in which matrix multiplication is conducted, as long as the relative order of matrices with the same superscript is maintained. For the mathematical justification of this step see [Bla85, Section 8.4]. For example W can be calculated as

$$W' = (G'G'')\ (H'H'')\ (B'B'')\ (E'E''F'F'')$$

or many other ways.

Each different order results in different algorithms which have, in the general case, different number of operations, different critical paths, and in general different structures. Combinatorial analysis shows that in this way we can produce several thousand different FFT algorithms [Bla85].

3. **Infinitely many.** The final scenario, when arbitrarily many options can be easily generated and analyzed, is from the methodology viewpoint of special interest. This situation arises when the initial definition of the algorithm specification has one or more parameters which can be set to any value from some continues set of values.

   The typical example of algorithms from this class are overrelaxation, and successive overrelaxation Jacobi and Gauss-Seidel iterative methods for solving linear systems of equations [Ber89]. For instance Jacobi overrelaxation (JOR) method and Gauss-Seidel successive overrelaxation (SOR) are given by the following two algorithms respectively:

$$x_i(t+1) = (1-\gamma)x_i(t) - \frac{\gamma}{a_{ii}}\left[\sum_{j \neq i}^{n} a_{ij}x_j(t) - b_i\right] \quad \textbf{(EQ 1)}$$

$$x_i(t+1) = (1-\gamma)x_i(t) - \frac{\gamma}{a_{ii}}\left[\sum_{j<i} a_{ij}x_j(t+1) + \sum_{j>i} a_{ij}x_j(t) - b_i\right] \quad \textbf{(EQ 2)}$$

   In the first case parameter γ can take any value between 0 and 1. Note that for different values of γ the algorithm has different speed of convergence. The fastest converging value of γ depends on the set of linear equations being solved.

   In some sense this situation can be treated as qualitative change of initially targeted algorithm selection to automatic algorithm design. Actually, for example, already Johnson and Burrus named their work as algorithm design [Joh83].

2.3 What is the difference in the "quality" of different algorithms; is it important to select the right algorithm?

These questions can be answered at two levels. However, regardless of which level is considered, the answer is identical: there is a very high difference. Usually there is an order of magnitude difference or more in the most important design metrics of the final implementation, depending on which algorithm selected.

The first level is one taken by theoretical computer science and numerical algebra and analysis communities. Asymptotic complexity theory [Gar79] clearly distinguishes between algorithms of different asymptotic complexity and states that algorithms of lower complexity will eventually, as the size of considered instance increases, be superior to the algorithms of higher complexity by an arbitrarily large difference. For example linear algorithms for sorting are getting more and more superior to sorting algorithm of quadratic complexity. Similar situation is numerical analysis, where the speed of convergence to correct solutions is treated in almost identical fashion.

CAD treatment of the algorithm selection process enables significantly finer resolution among algorithms. Even when only algorithms of the identical asymptotic computational complexity and the same asymptotic speed of convergence are considered they will require sharply different resources for a given set of constraints. To establish credibility of this claim we conducted comprehensive studies on two real life examples: 8th order Avenhaus bandpass filter and two dimensional 8X8 DCT.

The selection of the appropriate filter structure for a specified frequency response has long been recognized as one of the crucial factors determining almost all filter characteristics such as numerical stability and implementation complexity [Cro75]. As a result, many sophisticated filter structures have been proposed. While the numerical side of the filter behavior is well understood, the implementation complexity issues have been rarely or only marginally addressed.

The filter under study in this paper was first presented by Avenhaus [Ave72] and has often been used in the digital filter design, analysis and research. For example, Crochiere and Oppenheim [Cro75] presented in an depth discussion of several digital filter structures, which can implement the required frequency response. They compared the structures according to their statistical coefficient word length, the required number of multiplies and adds, the total number of operations and the amount of the parallelism and serialism. Although their 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 considered the following five structures of the Avenhaus filter, designed by Crochiere and Oppenheim [Cro75]:

1. direct-form II;
2. cascade form composed of direct form II sections;
3. parallel form;
4. CF - continued fraction expansion structure type 1B;
5. Gray and Markel's ladder structure.

The first four structures correspond to different representations of the same transfer function. The first three of them (direct-form II, cascade and parallel form) are well known and were considered very early in the filter design literature [Mit93]. The direct form corresponds to the representation of the transfer function as a ratio of polynomials, the cascade form represents the transfer function as a product of second order polynomial ratios, while the parallel and continued fraction forms correspond to partial and continued fraction expansion respectively. The continued fraction representation was first proposed by Mitra and Sherwood [Mit72]. The fifth structure is proposed by Gray and Markel and is based on an exploration of the relationship between two port networks in the analog circuit and digital filtering theory [Gra73].

All five Avenhaus structures were described using the Silage language and passed to the Hyper high level synthesis environment [Rab91]. They were simulated in both double floating point and fixed point precision in all phases of the design process to ensure that the obtained structures were in agreement with the required frequency response.

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

Table 1: Number of operations and Statistical Word Length.

Table 1 (assembled using data from [Cro75]) shows the number of multiplications and additions and the statistical word length for all five forms. The statistical word length is the minimum number of bits needed, such that the resulting structure still meets the frequency domain specifications (as determined by a statistical analysis). We simulated all five examples, and indeed, all of them produced the required frequency responses. A small correction was needed for the direct form II, as, due to a typographical error, two coefficients were interchanged in the Crochiere and Oppenheim paper.

structure	total	additions	shifts	subtractions	transfers
direct form II	215	58	103	46	7
cascade	94	31	40	18	4
parallel	113	33	51	24	4
continuous fraction	205	55	106	43	-
ladder	116	35	49	31	-

Table 2: Number of operations after Multiplication Expansion. Note that all structures have one input operation in addition to the listed Operations. 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, which substitutes all multiplications by a proper combinations of shifts and adds. Generally speaking, 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 on the application of the multiplication strength reduction.

For example, although the direct form has both fewer multiplications (16 vs. 18) and a shorter word length (21 vs. 23) than the continuous fraction, the latter eventually requires fewer shifts (103 vs. 92) and a smaller number of operations (215 vs. 178). From this table, it already becomes clear that, even when implementing the filter on a general purpose, single processor computer, the selection of the right algorithm is extremely important (for instance because of the wide span over the number of operations: 94 vs. 215).

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 retiming algorithm for both retiming and pipelining [Lei83,Pot92a]. During the optimization for area, retiming and pipelining for resource utilization was used [Pot91]. As will be demonstrated below, those few transformations had already profound effects on the final results.

structure	initial [nsec]	retimed [nsec]	pipelined [nsec]
direct form II	980	686	686
cascade	527	341	279
parallel	609	522	261
continuous fraction	2332	1908	1908
ladder	2835	2835	630

Table 3: Critical path for the Avenhaus Eight Order Bandpass Filter (in µsec - 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 transformation, 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 [Pot92b] reduces the length of the critical path to only 174 nanoseconds without introducing any latency. This is an improvement by a factor of 16.3 over the original ladder structure. When additional latency is allowed, this factor can be improved to arbitrarily high levels [Pot92b].

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. Note that the feasibility of achieving the required performance is a function of the selected structure and the applied transformations. It can be observed that the ratio between the largest and smallest implementation is even higher than the mentioned performance ratios.

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
CFI	14.65	22.67	-	-	-	-
CF RP	13.56	19.54	27.19	-	-	-
ladder RI	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.

For example, for a sampling period of 1 micro-second, 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 improved 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. Most often, pipelining increases register requirements even more.

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 (designs where 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. Notice that ladder form does 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.

algorithm	area [mm^2]	T area [mm^2]	T power [nJ/ sample]	T critical path [nsec]
direct	92.96	35.86	79.57	380
DIF	9.58	4.29	13.39	600
DIT	9.62	4.78	16.26	620
wang	10.07	9.27	20.77	600
lee	9.58	2.93	12.19	560
QR	9.43	7.61	16.68	560
Givens	16.85	9.91	27.17	600
mcm	92.96	8.78	21.64	340

Table 5: DCT: the area of implementations before application of substitution of constant multiplication by shifts and additions and area, critical path length and the estimates of power requirements after the application of constant multiplication substitution with shifts and additions (Columns with prefix T).

The Discrete Cosine Transform (DCT) was introduced by Ahmed, Natarjan, and Rao in 1974 [Ahm74]. It has found a wide spectrum of applications in many engineering areas including image processing, data compression, signal analysis [Rao90] mainly due to its remarkable similarity to the statistically optimal Karhunen-Loeve transform for highly correlated signals. Recently, it has been promoted to a position of ubiquitous computation, mainly due to its acceptance as a part of number image and video compression standards (H261, JPEG, MPEG,...) [Pen93]. In the last 20 years numerous fast algorithms have been proposed for optimizing the implementation cost. Interestingly, among current day ASIC products, the direct form of DCT is most often used [Rao90].

Due to its widespread use, design of fast algorithms started soon after the DCT was introduced. A book by Rao and Yip [Rao90] provides a good review of majority DCT algorithms analyzed in the paper.

We synthesized, under the identical throughput constraints, the following eight DCT algorithms: Lee - Lee's recursive sparse matrix factorization algorithm [Lee84], Wang- Suehiro-Hatori's version of the Wang planar rotation-based sparse matrix factorization DCT [Wan85, Sue86], DIT - recursive decimation in time algorithm [Yip84], DIF - recursive decimation in frequency algorithm [Yip85], QR - QR decomposition based hybrid planar rotation algorithm [Vet86] Givens - Givens rotation-based algorithm [Loe88], MCM - automatically synthesized algorithm, which applies only one multiple constant multiplication transformation on the generic DCT transform, and direct- the direct, generic definition of DCT algorithm. Table 5 shows the area of implementations before application of substitution of constant multiplication by shifts and additions and area, critical path length and the estimates of power requirements after the application of constant multiplication substitution with shifts and additions. We see that difference of optimized area due to algorithm selection is up to factor greater than 12(35.86 vs. 2.93), power by factor of 6.5 (79.57 nJ vs. 12.19 nJ) and critical path by 82% (620 vs. 340). If we compare the largest non-optimized design versus the smallest optimized design the area differential is by a factor larger than 31.7 times. The importance of algorithm selection and transformations is apparent.

Even though the two presented applications are rather small (both examples have only one basic block for which one of few algorithms has to be selected), the presented results clearly indicate that the algorithm selection step is a place where key trade-off in the quality of implementations are made during design process. Another interesting and important observation from this section is that for a different set of constraints and goals, different algorithms are best choice. Also note, that even a very limited set of transformations very influentially alters the quality of algorithms, and its suitability for a given set of goals and constraints.

It is worth mentioning that all results presented in this section, if not otherwise stated, were obtained using the Hyper system [Rab91]. All results were generated and analyzed in a time span of two hours, which demonstrates that the efficiency of the current generation of high level synthesis tools is high enough to address the algorithm selection and tuning problem.

2.4 Algorithm Selection-based Design Methodology

A popular and widely quoted analogy states that system level design is currently in a state which is very similar to the state of VLSI IC design and CAD IC level tools in the second half of the seventies [Mea80]. The analogy was recently cleanly crystallized and well summarized by W. Wolf [Wol93]. At that moment, he noted, it was clear that although industrial designers had been building chips widely and successfully, full potential of rapidly developing IC fabrication technology is not well utilized. Only with the introduction of systematic procedures to IC design, and quantitative posing of simulation, physical, logic synthesis, testing, and most recently high level synthesis tasks, IC designers of today can, in a fraction of time, design an order of magnitude more efficient design using CAD tools.

Similarly, system level design in general and algorithm selection in particular are mandatory part of design process for innumerable application specific systems. However, they are currently treated using ad-hoc methods and often result in unnecessarily low performance and expensive computing systems. The analogy states that the proper approach to hardware-software codesign is to develop a new design methodology.

We believe that the proper way to develop the new design methodology is to explicitly use quantitative optimization intensive methods. Only in this case, CAD tools developers will be able to provide effective help to both system and algorithm designers. In order to support the algorithm selection-based quantitative optimization intensive design methodology three major CAD/compiler components are needed: synthesis, estimation and transformation tools.

The need for synthesis tools is self-evident. Synthesis process is cumbersome and involves numerous involved details, which often take an overwhelming part of design efforts, if conducted manually. Synthesis tools release an application designer from this tedious task. A part of synthesis tools are already provided by compilers and high level synthesis tools. However, new level of abstraction requires also new tools. In this paper we present one of them, for algorithm selection.

Estimations are probably the single most important tool for both algorithm and implementation platform selection. Only when the effect of decisions done on high level of abstraction can be properly correlated with final implementation design metrics, it is meaningful to talk about the way in which decisions are selected. While the problem is well understood when both algorithm and some of implementation platforms are selected, new degrees of freedom impose new requirements on modeling tools for system level design. During modeling both size and structural properties of both algorithm and architecture have to be taken into account.

Transformations are algebraic-low, redundancy manipulation or control flow changes in algorithms so that functionality is not altered. They are natural complement to algorithm selection. From the design exploration point of view, algorithm selection enables global, high distance choices. Transformations, on the other hand, are enabling local and detailed scanning of design space. As it was illustrated on both Avenhaus filter and DCT examples, for best results both components of design space explorations are important.

While the mechanisms for transformations can be directly transferred from compiler and high level synthesis work, their coordinated application with algorithm and implementation platform selection is one of the important aspects of future system level design.

3.0 Optimizing Throughput Using Algorithm Selection

Until now all presented examples had only one basic block for whose implementation several algorithms were available. Majority of real-life applications are significantly more complex and have a number of blocks for which a number of algorithm choices exist. In the next two sections we will study algorithm selection problem in this more complex and realistic formulation.

In this section, we first briefly outline key underlying assumptions and targeted application domains. After that, we formulate the problem of throughput optimization using algorithm selection. Next, we establish the computational complexity of the problem, by showing that the problem is NP-complete. After that we present an effective sharp lower bound estimation technique for the problem and finish by developing a heuristic for solving throughput optimization using algorithm selection problem.

3.1 Preliminaries

We selected as a model of computation synchronous data flow [Lee87]. The computation targeted for implementation is represented using block-level control-data flow graph (CDFG) format. In this format blocks are (mainly semantically) encapsulated pieces of computations. They are connected by data and control edges. For each block a number of structurally different, but functionally equivalent algorithms represented by flat CDFGs is available. Data edges represent data transfers between the blocks and control edges denote additional timing and precedence constraints imposed by the nature of the application.

Since the synchronous dataflow model of computation implies semi-infinite stream of data, delays (states) are used to denote boundaries of the iterations and provide mechanisms for proper initialization of computations. We denote delays by rectangles with inscribed D.

Synchronous dataflow model of computation has two major implications to the algorithm selection problem: it implies static scheduling done during synthesis and its well behaved structure enables use of accurate estimations tools. Synchronous dataflow model of computation is widely used in both industry and research studies [Lee95] and is in particular well suited for specification of many DSP, communication, control, and other numerically intensive applications. Although, in the rest of the paper we will exclusively use the synchronous dataflow model of computation, it is important to note that all presented results and algorithm can be directly used in many other computational models, in particular those who belong the family of dataflow network process models [Lee95].

During the optimization for throughput we do not impose any constraints on the number of threads of controls. During optimization for area one thread of control is assumed. Note that this assumption is with no consequence, if as preprocessing step merging of individual blocks is allowed.

3.2 Problem Formulation

Figure 4 shows an instance of the algorithm selection problem for throughput optimization. The overall application is depicted by a number of basic blocks which are interconnected in a specific, application dictated, manner. For each block, a number of different CDFGs (corresponding to different algorithms) is shown in Figure 4b. The number of options for block B_i is denoted by n_i. Each block, B_i, has m_i inputs and k_i output terminals. Each block is characterized, using the standard critical path algorithm by a matrix of size m_i X k_i distances between any two pairs of terminals. The goal is to select for each block a CDFG, so that the overall critical path (which corresponds to the throughput of the system) is minimized.

We will further illustrate the problem, using a very small instance of the algorithm selection problem shown in Figure 5. Each block has two different algorithmic options shown in Figure 5b. If for each block the first options are selected, the resulting critical path is 20 cycles long. However, if for both choices the second CDFG options are chosen, the resulting critical path is only 14 control step long, although each component has the critical path of length 12.

More formally the throughput optimization using the algorithm selection problem, can be stated, in the standard Garey-Johnson [Gar79] format:

PROBLEM: CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION

INSTANCE: Given a 2-level hierarchical directed CDFG(B,E). At the higher level of the hierarchy each node

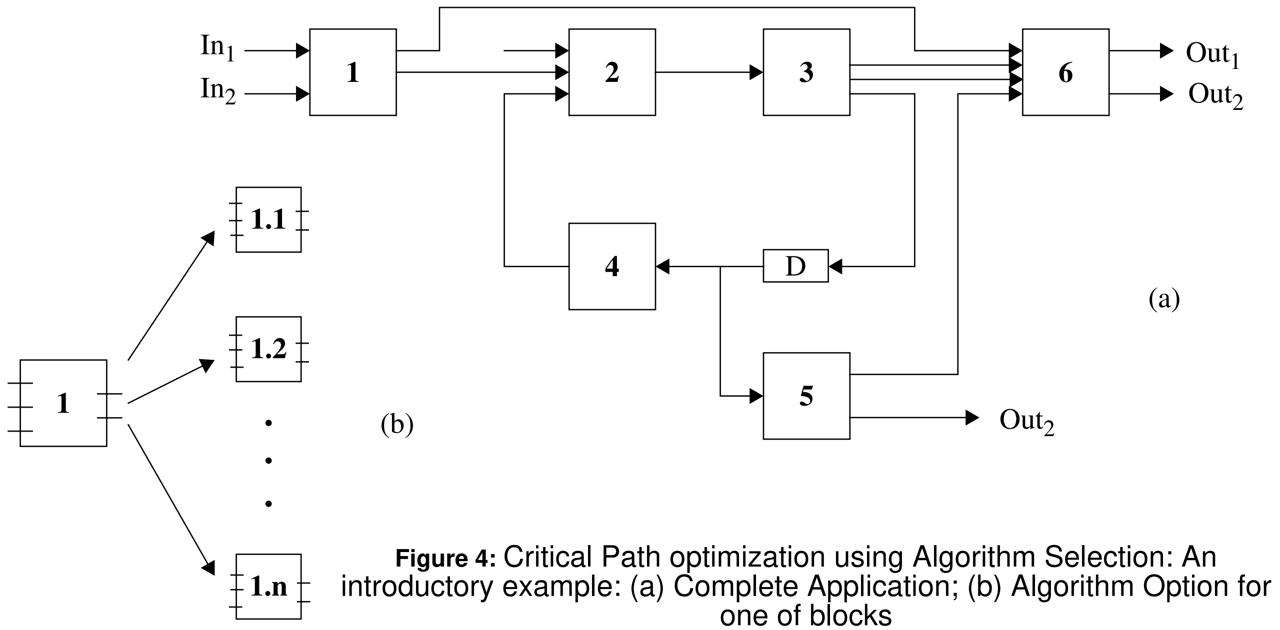

Figure 4: Critical Path optimization using Algorithm Selection: An introductory example: (a) Complete Application; (b) Algorithm Option for one of blocks

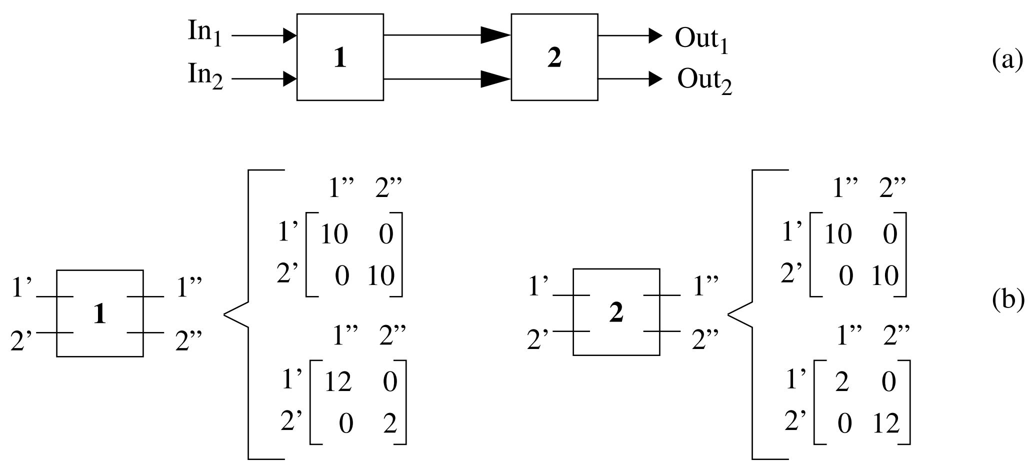

Figure 5: A simple instance of the critical path optimization problem using algorithm selection

is a block, denoted by B_i. Each block can be substituted at the lower level of hierarchy using any of n_i CDFGs which can be used for its realization. Each block, B_i, has m_i input and k_i output terminals. The distances between any pair of input and output terminals are given.

QUESTION: Is there a selection of CDFGs for each block so that the resulting critical path is at most L cycles long.

3.3 Optimizing Throughput Using Algorithm Selection is NP-complete problem

In this section we will prove, using the standard Karp's polynomial reduction technique that the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION is NP-complete problem. In particular, we will use the local replacement technique [Gar79, page 66].

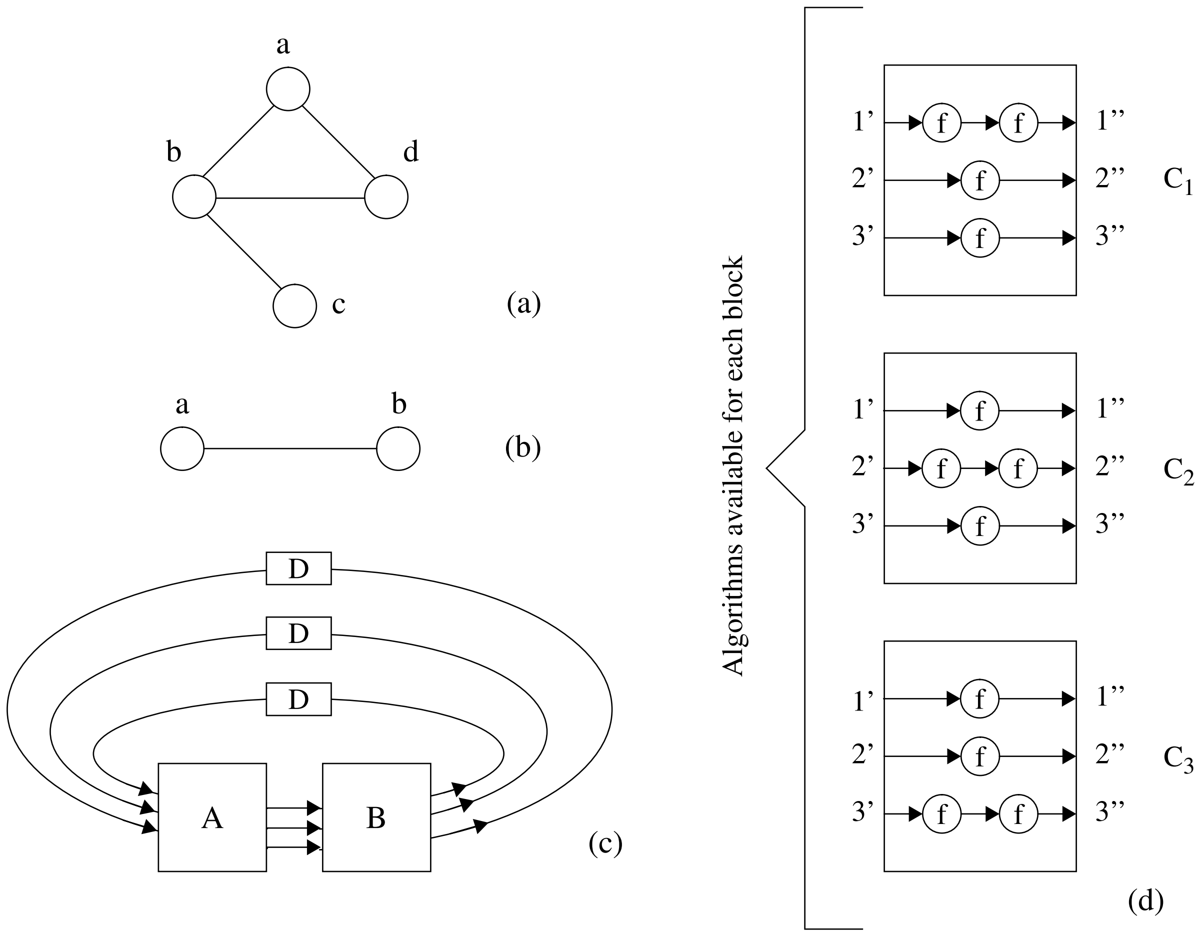

Figure 6: Critical Path optimization using Algorithm Selection is NP-complete problem: Transformations from the GRAPH 3-COLOROBALITY problem: (a) Instance of the graph 3-coloring problem: (b) the smallest instance of he graph 3-coloring problem; (c) the corresponding instance of the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION problem; (d) algorithms available for each block in the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION problem

The problem is in NP class because a selection combination which has the critical path L, if exists, can be checked quite easily. All what is necessary is to substitute each selected CDFG in the corresponding block and calculate the length of the critical path using the standard critical path algorithm [Cor90].

We will now polynomially transform the GRAPH K-COLORABILITY problem to CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION to complete NP-completeness proof. For the sake of completeness, we first state the GRAPH K-COLORABILITY problem. The problem is denoted by GT4 in [Gar79].

PROBLEM: GRAPH K-COLORABILITY PROBLEM (CHROMATIC NUMBER).

INSTANCE: Graph $G = (V, E)$, positive integer $K \leq |V|$.

QUESTION: Is G K-colorable, i.e., does there exist a function $f: V \rightarrow \{1, 2, ..., K\}$ such that $f(u) \neq f(v)$

whenever $\{u, v\} \in E$?

The GRAPH K-COLORABILITY problem is solvable in polynomial time for $K = 2$, but remains NP-complete for all $K \geq 3$ [Gar79]. In the rest of the proof, for the sake of clarity we will use GRAPH 3-COLORABILITY problem. The proof requires only very minor straightforward changes, if the GRAPH K-COLORABILITY problem is instead targeted.

Suppose we are given an arbitrary graph *G* as shown in Figure 6a. We will first polynomially transform *G* to a corresponding CDFG so that finding a solution in polynomial time to the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION implies a polynomial time solution to the corresponding instance of the GRAPH 3-COLORABILITY problem.

For each node *V* in *G* the CDFG contains a corresponding node (block) at the higher level of hierarchy. Each block has 3 input and 3 output terminals. Each block can be implemented with one of the 3 CDFGs which have the lengths of critical path between 3 inputs (2,1,1), (1,2,1) and (2,1,1). An illustration of internal structures of possible choices for each block is shown in Figure 6d.

If graph *G* has an edge between nodes $\{u,v\}$, then the CDFG at the higher level of hierarchy has the corresponding blocks connected as indicated in Figure 6c. An illustration of the mapping of an instance of the smallest GRAPH 3-COLORABILITY problem to an instance of the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION is presented in Figures 6b and 6c. It is assumed, without any influence on the mapping, that one block in the CDFG serves as both primary input and the primary output of the computation.

It is easy now to see that if we are able to solve the instance of the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION so that the critical path is long 3 (1+2) cycles, we can easily produce graph coloring with 3 colors in *G*. The conversion of the solution is achieved by coloring a node *v* in the graph *G* with a color 1 if the choice for the option (2,1,1) is made to the corresponding block in CDFG. Similarly colors 2 and 3 correspond to selections (1,2,1) and (1,1,2) respectively. Note that if for two nodes in the G have the same color, then the length of the critical path in the CDFG is at least 4 (2+2).

Therefore, the CRITICAL PATH OPTIMIZATION USING ALGORITHM SELECTION belongs to the class of NP-complete problems.

3.4 Lower-Bound Estimations

It is well known that no polynomial complexity solution exists for NP-complete problems [Gar79], so it is difficult to evaluate the quality of the proposed optimization algorithm. Recently, in both high level synthesis [Sha93, Rab94] and system level synthesis [Ver94], it has been demonstrated that sharp lower bounds are often close in a variety of design metrics to the solution produced by synthesis systems. It has been documented that sharp estimation bounds are directly useful in the number of synthesis and evaluation tasks, such as scheduling, assignment, allocation, transformations and partitioning [Rab94].

An effective and very efficient lower bound can be easily derived for throughput optimization using algorithm selection. The algorithm is given by the following simple three-step pseudo-code.

Algorithm for lower bound estimation for throughput optimization using algorithm selection

1. *Find for each block super-algorithm implementation;*
2. *Implement each block using its super-algorithm.*
3. *Using the standard critical path calculate the critical path.*

The superalgorithm is a new option for each block. It has as distances between its pairs of inputs and outputs the shortest distance which can be achieved by any of the algorithms that can be used for this block. The computational complexity of the estimation algorithm is dominated dynamic programming-based all pairs shortest path algorithm [Law76], and as explained in the next subsection is O (m * n^2), where m is the number of the algorithmic options for all blocks and n is the number of operations in the largest algorithmic option.

3.5 Relaxation-Based Heuristic

One of the most important applications of estimations is to provide a suitable starting point for development of algorithms for solving the corresponding problem. Using this idea, we developed the algorithm for throughput optimization using algorithm selection, given by the following pseudo-code.

Discrete Relaxation-based algorithm for throughput optimization using algorithm selection

1. *Calculate all IO distances for all available algorithmic options;*
2. *Eliminate all inferior options for each block;*
3. *Assign to each block the superalgorithm;*
4. *while there is a block with the superalgorithm solution{*
    5. *Find ε-critical network;*
    6. *For each block find all pairs on inputs/outputs on the critical network;*
    7. *For each block on ε-critical network calculate sum of differences between super-algorithm and three best choices among available options.*
    8. *Find the block which has the greatest sum of differences;*
    9. *Select the algorithm for this block which makes the current overall critical path minimal;*

}

The first preprocessing uses dynamic programming-based all pairs shortest path algorithm [Law76] to extract all relevant information from available algorithmic options for critical path minimization. The second preprocessing step eliminates all options which are uniformly inferior to some of the other candidates. Algorithm A is inferior to algorithm B, if all pairs on corresponding input/output terminals, the length of the longest path between the pair of terminals in A is at least as the longest path in B. This problem is known as the dominance problem, and can be solved in linear time using the plane sweeping algorithm [Cor90].

In the second step to each block superalgorithm is assigned. Superalgorithm is a new composite algorithm which has as the distances between a given pair of input i and output j the shortest distance as provided by any available algorithm for this block.

The general idea of the algorithm is to start the decision making process with the most critical, high impact, choices (selections which influence the final solution most) first, and make them early, so that their potential bad effects are minimized later with successive steps. Low-impact choices are left for the end when there is no more time to minimize their bad effects using successive selection steps. Distinction between high and low impact choices is made using ε-critical network and a set of differences in the lengths between available options.

ε-critical network is a network which contains all nodes on the critical path and paths which lengths is within ε% of the critical path. In this work we used $\varepsilon = 10$. ε-critical network is often using in several CAD domains, such as logic synthesis [Sin88] and high level synthesis [Cor93], as an efficient way to abstract the most important parts of overall design which are most relevant to the quality of the final solution. It can be easily and rapidly computed using dynamic programming approach [Cor93].

The second component which drives the decision process is that all superalgorithms have to be replaced with one of the available choices. Obviously, if this replacement will significantly increase the critical path it is better to make this decision early, so that later on the better approximation of the final solution is available.

We will illustrate the optimization algorithm using an example shown in Figure 7. The hierarchical CDFG

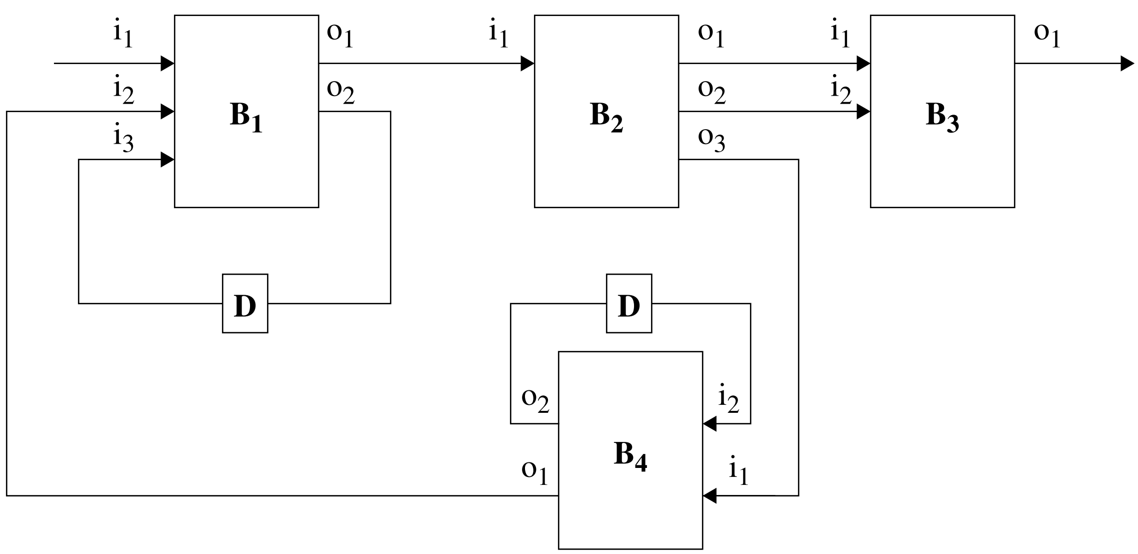

Figure 7: Algorithm for throughput optimization using algorithms selection: example, hierarchical CDFG. For detailed explanation see Section 3.4

has 4 blocks: B_1, B_2, B_3, and B_4. For each block four algorithmic options are available, as shown in Figure 8. For example, for block B1 the algorithms $A_{1,1}$, $A_{1,2}$, $A_{1,3}$, and $A_{1,4}$ are available. The

$$\underset{A_{1,1}}{\begin{array}{c c} & \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{array}{c} i_1 \\ i_2 \\ i_3 \end{array} & \begin{bmatrix} 3 & 5 \\ 4 & 4 \\ 5 & 6 \end{bmatrix} \end{array}} \quad \underset{A_{1,2}}{\begin{array}{c} \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{bmatrix} 3 & 5 \\ 3 & 4 \\ 5 & 5 \end{bmatrix} \end{array}} \quad \underset{A_{1,3}}{\begin{array}{c} \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{bmatrix} 4 & 4 \\ 5 & 4 \\ 6 & 2 \end{bmatrix} \end{array}} \quad \underset{A_{1,4}}{\begin{array}{c} \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{bmatrix} 7 & 3 \\ 3 & 5 \\ 6 & 6 \end{bmatrix} \end{array}}$$

$$\underset{A_{2,1}}{\begin{array}{c c} & \begin{array}{ccc} o_1 & o_2 & o_3 \end{array} \\ i_1 & \begin{bmatrix} 2 & 3 & 5 \end{bmatrix} \end{array}} \quad \underset{A_{2,2}}{\begin{array}{c} \begin{array}{ccc} o_1 & o_2 & o_3 \end{array} \\ \begin{bmatrix} 3 & 4 & 9 \end{bmatrix} \end{array}} \quad \underset{A_{2,3}}{\begin{array}{c} \begin{array}{ccc} o_1 & o_2 & o_3 \end{array} \\ \begin{bmatrix} 4 & 2 & 4 \end{bmatrix} \end{array}} \quad \underset{A_{2,4}}{\begin{array}{c} \begin{array}{ccc} o_1 & o_2 & o_3 \end{array} \\ \begin{bmatrix} 2 & 5 & 3 \end{bmatrix} \end{array}}$$

$$\underset{A_{3,1}}{\begin{array}{c c} & o_1 \\ \begin{array}{c} i_1 \\ i_2 \end{array} & \begin{bmatrix} 1 \\ 3 \end{bmatrix} \end{array}} \quad \underset{A_{3,2}}{\begin{array}{c} o_1 \\ \begin{bmatrix} 3 \\ 1 \end{bmatrix} \end{array}} \quad \underset{A_{3,3}}{\begin{array}{c} o_1 \\ \begin{bmatrix} 2 \\ 2 \end{bmatrix} \end{array}} \quad \underset{A_{3,4}}{\begin{array}{c} o_1 \\ \begin{bmatrix} 1 \\ 4 \end{bmatrix} \end{array}}$$

$$\underset{A_{4,1}}{\begin{array}{c c} & \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{array}{c} i_1 \\ i_2 \end{array} & \begin{bmatrix} 5 & 5 \\ 9 & 9 \end{bmatrix} \end{array}} \quad \underset{A_{4,2}}{\begin{array}{c} \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{bmatrix} 6 & 6 \\ 7 & 7 \end{bmatrix} \end{array}} \quad \underset{A_{4,3}}{\begin{array}{c} \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{bmatrix} 7 & 7 \\ 6 & 6 \end{bmatrix} \end{array}} \quad \underset{A_{4,4}}{\begin{array}{c} \begin{array}{cc} o_1 & o_2 \end{array} \\ \begin{bmatrix} 9 & 9 \\ 5 & 5 \end{bmatrix} \end{array}}$$

Figure 8: Algorithm for throughput optimization using algorithms selection: example, distances between all pairs of inputs and outputs for available flat CDFGs which correspond to different algorithmic options. For detailed explanation see Section 3.4

distances between all pairs of inputs/outputs are shown in matrix form. The same figure shows the distances matrices of the corresponding superalgorithms for each block denoted by S_i, i=1,...,4.

In the second step of the heuristic approach the algorithm option $A_{1,1}$ is eliminated since it is dominated by $A_{1,2}$ and option $A_{3,4}$ is eliminated since it is dominated by $A_{3,1}$. he critical path for the hierarchical CDFG after the step 3 of the optimization algorithm is path $i_3B_1o_1$ -> $i_1B_2o_3$ -> $i_1B_1o_1$ and has length 14 (5 + 4 + 5). In the first pass of the optimization algorithm for throughput optimization in step 6 blocks B_1, B_2, and B_4 are denoted as blocks which are on the ε-critical network. During the same pass in steps 7 and 8 as most critical block B_4 is identified. For this block in the step the algorithm $A_{4,1}$ is selected. After this selection the critical path remains 14. In the next 3 passes of the algorithm the following decision are successively made A1,2 -> B1, A2,2 -> B2, and finally A3,3 -> B3. The final critical path is 14 which is provably optimal as indicated by the estimation algorithm presented in the previous subsection.

Design	Worst Random Sampling Rate [nsec]	Median Random Rate [nsec]	Best Random Rate [nsec]	Optimized Sampling Rate [nsec]
LMS transform filter	780	700	620	420
NTSC signal formatter	1,060	920	780	480
DPCM video signal coder	2,160	1,560	1,280	1,000
4-stage AD converter	10,240	3,740	2,320	1,520
5-stage AD converter	16,812	4,212	1,926	828
Low Precision Data Acquisition System	11,520	4,200	2,420	1,600
High Precision Data Acquisition System	23,952	6,000	3,456	1,200
DTFM - dialtone phone system	4,032	960	320	256
ADPCM	9.088	3328	2,048	960
Subband Image Coder	2,352	1,704	912	108

Table 6: Sampling Rate (critical path) optimization using Algorithm Selection

The computational analysis of the proposed algorithm is simple, all individual steps take linear time, except the first one. The analysis shows that worst case run time is bounded by O (NB * NHE) as dictated by the while loop or by the first step which calculates the distances between all pairs of inputs and outputs for all algorithmic options. NB and NHE denote the number of blocks and the number of hierarchical edges respectively. Since the graphs in the first step are or can be easily reduce to acyclic equivalents, the run time of this step for one algorithmic option is O (n^2) [Law76, page 68]. In practice all run times for the overall optimization algorithms were less than one second SUN SparcStation 2.

3.6 Experimental Results

Table 6 illustrates the effectiveness of the throughput optimization using algorithm selection ten audio and video designs. First, algorithm selection is done randomly 11 times, and the worst, the median, and the best critical paths are recorded for each design. The average (median) reductions of critical path are by 79.4% (87.8%), 64.3% (66.5%), 44.5% (37%) respectively for the worst, the median, and the best random solution, resulting in average (median) improvements of throughput by factors 10.7 (8.2), 4.33 (3.04), 2.45 (1.62) respectively. In all cases lower bounds were tight, indicating that the relaxation based heuristic achieved the optimal solution.

4.0 Optimizing Cost Using Algorithm Selection

Once when the throughput requirements are ensured, often the dominant metrics of quality of design is its cost. For example, a common design scenario in DSP and communications applications is the area optimization under standardized throughput requirements. In this section we address the problem of optimizing cost under throughput constraints using algorithm selections. We discuss the problem in the simplified version where a single thread of control is assumed and a fixed implementation platform, full ASIC custom implementation, is targeted. The generalization to the fully general case is conceptually simple, but at the technical level more complex optimization algorithms and more complex set of estimation tools are needed. The most important principle behind the methods presented in this section is that all techniques are build by limited modifications of available high level synthesis algorithms. We believe that the most effective way for developing techniques for system level problem is to reuse high quality, already proven in practice, high level synthesis algorithms whenever it is possible. In particular, we used the techniques from the Hyper high level synthesis system [Rab91]. When other implementation platforms are targeted, the reusability principle indicates that it is advisable to take y compilation and evaluation tools which are already available as the starting point.

4.1 Problem Formulation, Computational Complexity and Lower-Bound Computation

The cost optimization using algorithm selection problem can be formulated in the same way as the critical path optimization problem. The only difference is, of course, that now the goal is the minimization of the implementation cost under the critical path constraint. We assume that the implementation platform is custom ASIC. In our realization of the proposed approach we used the Hyper system [Rab91] for producing final ASIC implementations.

It is easy to see that the problem is NP-hard, because even when the application has only one algorithm and only one choice for algorithm, finding the minimal cost is NP-complete. This is because the scheduling problem itself is an NP-complete problem [Gar79].

The lower bound estimation algorithm for the area optimization using algorithm selection problem is build on the top of Hyper estimation tools and the notion of superalgorithm. However, now superalgorithm instead of targeting critical path encapsulates the minimum required amount of each type of hardware (e.g. multiplier or register). We assume that reader is familiar with Hyper estimation techniques, for detail exposition see [Rab91, Rab94]. First each block is treated individually. For each algorithm available for realization of the block and each type of resource (e.g. each type of execution unit or interconnect) a hardware requirements graph is build for all values of available time starting from the critical path time, to time when only one instance of the resource is required.

Requirements for each block are built by selecting for each type of resource the best possible algorithm selection, taking this resource exclusively into account. Note that for different resources in general, different algorithms are selected when different resources are considered. We refer to requirements of a such implementation for a block as superalgorithm implementation. The implementation costs of the superalgorithm for each block are used as entries in the Hyper's hierarchical estimator, which optimally, in polynomial time, allocates time for each resource so that final implementation cost is minimized. This cost is lower bound for the final implementation. The lower bound estimation approach can be summarized using the following pseudo-code:

Algorithm for lower bounds derivation for the area optimization using algorithm selection:

```
1. Using superalgorithm selection for each block construct Resource Utilization Table for each Resource;
2. For each block assuming superalgorithm implementation while there exist block for which the final
   choice is not made{
     4. Recompute allocated times for each block;
     5. Recompute estimated Final Cost;
}
```

The computational complexity and run time are dominated by time required to create the lists which show the trade-off between the allocated time and required resources for all algorithms for all blocks. This time is just slightly supelinear (O(n log n)), where n is the number of nodes [Rab94]. Usual run time for one algorithm with several hundreds nodes is at most a few seconds on Sparctstation 2.

4.2 Area Optimization using Algorithm Selection

The key technical difficulty during algorithm selection for area optimization is interdependencies between the allocated time for a given block and most suitable algorithmic option. We resolve this problem by successively making decisions about the selected algorithms and refining the decisions about allocated times for individual blocks. The algorithm for cost optimization has two phases. In the first phase, we use min-bounds to select the most appropriate algorithm for each block and to allocate the available time for each block. In the second phase the solution is implemented using the Hyper hierarchical scheduler and the procedure for resource alocation.The first phase of the algorithm for the cost optimization using algorithm selection is given using the following pseudo-code.

Algorithm for area optimization using algorithm selection:

```
1. Using superalgorithm selection for each block construct Resource Utilization Table for each Resource;
   while there exist block for which the final choice is not made{
     2. Select the most critical block;
     3. Select the most suitable implementation for the block;
     4. Recompute allocated times for each block;
     5. Recompute estimated Final Cost;
}
7. Using the Hyper high level synthesis system implement the selected algorithms while satisfying
overall timing constraints;
```

Table6 shows a typical example of Resource Utilization Table. The table is constructed in the identical way as

Block	cycles	IO/ Cycles	*/Cycles	+/Cycles	Registers	*->+
block1	t1	1	0	1	37	0
block2	t2	0	.5	4	24	2.2
block3	t3	.3	2	.7	39	1.3
total	$t = \sum t_i$	1	2	4	39	3

Table 7: Resource Utilization Table used in the optimization procedure for cost optimization using algorithm selection. The shaded entries are the dominant entries. (see Section 4.1)

the initial allocation in Hyper. For a detailed exposure of Hyper scheduling tools see [Pot89]. Each row has as an entry the estimation requirements for the superalgorithm implementations (see Section 4.1). The bottom row has as entries with the maximum entry in each column for a particular resource.

The most critical block is the block that is evaluated as the block which influences the most the final requirements. The order in which blocks are selected is dictated by an objective function which combines four criteria:

1. Weighted sum of dominant entries that the row has. The dominant entry is the entry which has the maximal value in some column. Each entry is scaled by its explicit or estimated cost. If there are many hardware-expensive dominant entries in the row, this block dictates the overall hardware requirements, and it is more important to make the selection for this block early to get the correct picture of final requirements as soon as possible. This information is later used to guide other algorithm selection choices.

2. How many different algorithms are composing superalgorithm. If there are requirements of many different algorithms composing the superalgorithm requirements, after the particular selection, many of entries for this row will increase. This can lead to high increase in the overall cost.

3. How many different algorithms are contributing to the dominant values. The intuition is similar as in the previous case, but the importance of dominant entries is emphasized.

4. How sensitive the superalgorithm and choices are to time changes. The choices most sensitive to time should be made first, so that redistribution of time is correctly driven.

All four criteria are combined in the rank-based objective function. Once the block is selected, all available algorithms for this block are evaluated as possible choices. The most suitable algorithm is one which least increases the current overall cost (given in the last row). The algorithm is selected by substituting and evaluating all choices. Steps 4 and 5 are updates which are needed so the correct current picture of the partial solution is obtained after each decision

Table 7 shows an instance of resource utilization table at the beginning of the optimization procedure. The most critical row is the second row, since the cost of multipliers is the highest. Therefore, the optimization procedure will first make the decision of which algorithm is most suitable for implementation of this row, assuming the available time of t2 clock cycles.

The second phase is just the direct application of the Hyper synthesis tools for allocation, hierarchical scheduling and assignment [Pot92]. Note that at the beginning of the second phase the algorithm selection process is completed, and only remaining task is hierarchical scheduling and assignment.

Example	The worst random feasible area [mm^2]	Median random area [mm^2]	The best random area [mm^2]	Optimized area [mm^2]
LMS transform filter	92.96	Infeasible	92.96	9.62
NTSC signal formatter	170.77	122.08	100.53	5.86
DPCM video signal coder	124.54	Infeasible	85.80	4.43
4-stage AD converter	88.82	Infeasible	45.58	6.08
5-stage AD converter	96.03	96.03	62.79	8.23
Low Precision Data Acquisition System	67.76	60.89	44.09	4.02
High Precision Data Acquisition System	N/A	Infeasible	Infeasible	20.08
DTFM dialtone phone system	129.96	45.52	23.19	4.52
ADPCM	245.80	Infeasible	212.86	10.29
Subband Image Coder	188.92	Infeasible	167.55	10.86

Table 8: Optimizing Area using Algorithm Selection. Infeasible as an entry in the Table denotes that the critical path was longer than the available time.

Table 8 shows the area reduction due to algorithm selection. The experimental procedure was the same as the one presented in the previous section, only now only 5 random implementation where conducted for each design. In many cases the critical path non-optimized design for longer than the available time. Both the average and the median improvements over the best random selection are by a factor 12.6. In all cases the discrepancy between the lower bound and the final area was less than 25%. The computational complexity and the run time of the optimization algorithm is dominated by employed synthesis and estimation algorithms. The run time for presented examples are within several minutes [Rab91].

5.0 Related Work

To the best of our knowledge, this is the first work which addresses the algorithm selection process as a quantitative optimization intensive CAD effort. Implementation platform selection problem, mainly under the name of hardware-software codesign, has been recently extensively studied from both CAD and compiler viewpoints [Sri91, Cho92, Kal93, Sal93, Tho93, Gup94, Wol93, Woo94]. In a short amount of time, an impressive advance of research is achieved. However, majority of the hardware-software codesign studies and research efforts, until now, have been concentrating on building framework for synthesis, specification, and/or simulation and were treating the synthesis process mainly from qualitative point of view. Most often, only at the end, when the final implementation is produced, the realization is evaluated and sometimes compared to several other also completely done implementations.

Interestingly, the closely related but much more difficult task of algorithm design has been addressed much more often than the algorithm selection problem. Some of those efforts, actually, were very modest in their scope and goals and certainly can not justify use of term "algorithm design". For example, the first FORTRAN compiler (1954) was announced as algorithm synthesizer.

However, since the late sixties, artificial intelligence (AI) community devoted remarkable efforts to develop algorithm design tools. In the early stage of this research the following four research directions attracted the greatest amount of attention: resolution-based methods, rewrite systems, transformational approaches, and schema-based programming.

The theorem-proving approaches to algorithm design [Gre69] and [Wal69] are based on the Floyd's observation [Flo67] that an algorithm can be derived from a formal specification by using theorem-proving techniques [Rob65]. While initially the methods were based on resolution which provides a way for automatic production of proofs in first-order logic[Rob65], later developed nonclausal resolution [Man80] facilitates shorter run times. The major problems with this approach include exceptionally high execution times and a need that the user specifies a formal theory-based initial specification.

Another set of formal techniques to algorithm design, rewrite system, is based on Knuth-Bendix [Knu70] method. Most notable efforts in this domain include [Hof82, Der85].

Darlington [Dar81] pioneered the use of transformational rules which start with a compact and a non-efficient specification and evolves toward specification which corresponds to efficient implementation. Various transformational approaches employed a variety of techniques which improve efficiency of program derivation [Bar86, Kan79]. The key difficulties with this method include a need for encoding of thousands of transformations and a need for developing a good order of transformations. While the first problem is mainly software development related, the second one is a fundamentally difficult problem.

Several excellent surveys of AI and software engineering research in algorithm design are available [Bar82, Bal85, Low91]. Lowry [Low91] discusses in detail key fundamental obstacles related to majority AI-based algorithm design approaches. Finally, schema-based programming are interactive techniques [Wil83, Ric88] which are based on use of library of templates. Templates are cliches which facilitate algorithm design.

More recently, conceptually more complex AI methods have been incorporated in software engineering efforts for algorithm design. The most popular among them are knowledge-compilation methods [McC87], formal derivations systems [Bac88, Low89], and cognitive [Ste89] and planning [Lin89] methods. For example, the DRACO system [Fre87] employs artificial intelligence techniques to design programs using reusable program parts. The software reusability efforts are surveyed in [Big89].

In the CAD research, algorithm design efforts ware conducted by D. Setliff and R. Rutenbar who developed VLSI physical synthesis algorithms [Set91] using ELF synthesis architecture [Kan83]. The developed wire routers are technology adaptable. More recently this effort evolved toward efforts to develop an automatic synthesis system for real-time software [Smi91]. It was demonstrated how RT-SYN automatic synthesis program developed six different versions of Fast Fourier Transform (FFT). Several artificial intelligence methods for automatic synthesis of a few common DSP algorithm algorithms, such as the FFT, have been reported in [Opp92]. Another interesting AI-based approach to algorithm design has been developed at MIT [Abe89]. The approach supports design and analysis of complex numerical analysis algorithms.In coding theory and applications, El Gamal et al. [ElG87] used the simulated annealing-based search to design several error correction codes with superior performance to existing manually designed codes.

Recently, several notable efforts in the DSP research literature reported efforts for developing systems for algorithm design. The emphasis in this domain have been on development of algorithms with few operations (or few multiplication) for a limited domains, such as FFT [Joh93, Bla85].

Several researchers from the theoretical computer science community used computer support for design analysis, mainly through animation [Ben90, Bro85] and experimental comparison of heuristic algorithms [Joh90].There have also been several attempts to support algorithm selection and tuning in the theoretical computer science literature. These have involved using extensive computer simulation, experimentation, and statistical analysis [McG92].

Comparative and quantitative studies of suitability of a given algorithm for implementation has been, from time to time, conducted in several engineering areas, most often in digital signal processing, communications and control applications. While in majority of those studies as relevant design metrics were targeted the number of operations or the length of the critical path, several studies resulted in completely built systems. For example, recently six different groups built several HDTV systems which are currently being comparatively tested. Each group used different algorithms for the same set of specifications. Algorithms have been selected using manual ad-hoc techniques to analyze design trade-offs influenced by various selection decisions. For example, recently the HDTV consortia, after extensive comparison, selected the vestigial sideband (VSB) modulation algorithm over the quadrature amplitude modulation (QAM) algorithm as a part of future standard [Pet95].

At the technical combinatorial optimization level, once the semantic content is abstracted from the algorithm selection problem, there is a significant similarity between the topic discussed in this paper and technology mapping in logic synthesis research [Keu87, Det87], module selection in high level synthesis [Not91, Cor93], and code generation in compiler design [Aho89]. The key difference in the optimization task is that algorithm selection assumes significantly richer timing constraint modelling. Note that all the mentioned synthesis tasks assume that distances between any pair of input and output terminals are identical, while our formulation does not impose any constraints on the allowed set of the distances.

Even close relationship from combinatorial optimization point of view exist between the algorithm selection and the circuit implementation problem [Cha91, Li93] which in its most general formulation assumes general timing model. The main difference between the circuit implementation problem and the algorithm selection problem is that the latter has an additional degree of freedom as a consequence of hardware sharing, which additionally complicates both cost modelling and optimization.

6.0 Future Directions

Although, both system level design and algorithm selection are fields in a very early stage of development, several key directions for future efforts can be outlined by analyzing the presented results. The directions can be classified in several classes. Some of the directions are straightforward at the conceptual level. For example, there is an apparent need to target other design metrics in addition to throughput and area, such as power, fault tolerance and testability.

Key future directions in algorithm selection include development of:

1. Synthesis and evaluation infrastructure;
2. Integration of Algorithm and Implementation Platform Selection;
3. Algorithm Design;
4. Software Libraries.

The first direction, synthesis and evaluation infrastructure, includes already discussed (Section 2.4) estimations and transformations tools. As illustrated in Section 5 there is a close relationship between implementation platform selection and algorithm selection. Although both tasks are already individually highly impacting the quality of final design, their simultaneous addressing of both tasks will certainly additionally amplify the impact.

The long-standing goal of algorithm design will again move in a spot of research attention. While previously the algorithm design was driven mainly by intellectual curiosity, now the main driving force will be economic advantage which can provide a competitive superiority at system level design. On the technical level, instead of previously used artificial intelligence techniques, which where not effective in addressing the algorithm design problem, optimization intensive CAD and compiler techniques will provide a new infrastructure which will, hopefully, be more effective. The intrinsic difficulties, both conceptual and computational, of the algorithm design task will most likely channel initial efforts in algorithm design toward specific, economically important, domains. A typical example of the work along this line is that encapsulated by the Johnson-Burrus dynamic programming procedure for the design of FFT algorithms with few operations.

Reusability is one key technique for handling always increasing complexity of designs. Algorithm selection is naturally built on the idea of reusability of algorithms already available for given sub-tasks of the targeted application. Development of comprehensive libraries which have a variety of algorithms for commonly required task will additionally emphasize the importance of algorithm selection. Note that several system level design system are already using earlier versions of such libraries [Kal93].

The libraries will also ease the burden of programming. For example, Ptolemy [Kal93] synthesis system provides visual programming capabilities, where the user can select pre-programmed algorithms for tasks.

7.0 Conclusion

As a part of effort to establish CAD-based design methodology for system level design, we introduced the algorithm selection problem. After demonstrating a high impact of the task on the quality of the final implementation, we studied two specific problems in system level design: optimization of throughput and cost using algorithm selection in system level design. Both problems are studied in quantitative optimization-framework. It is proven that problems are NP-complete. Lower bounds calculation and heuristic solutions are proposed for the problems.

All claims are supported using experimental studies on real-life examples. Directions for further comprehensive efforts related to the algorithm selection are outlined.

8.0 References:

[Abe89] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E.Sacks, G.J. Sussman, J. Wisdom, K. Yip, “Intelligence in Scientific Computing”, Comm. of the ACM, Vol. 32, No. 5, pp. 546-562, 1989.

[Ahm74] N. Ahmed, T. Natarajan, K.R. Rao, “Discrete cosine transform”, IEEE Transactions on Communications, Vol. 23, No. 1, pp. 90-93, 1974.

[Aho89] A.V. Aho, M. Ganapathi, S.W.K. Tjiang, “Code Generation Using Tree Matching and Dynamic Programming”, ACM Trans. on Programming Languages and Systems, Vol. 11, No. 4, pp. 491-516, 1989.

[Ave72] E. Avenhaus, “On the design of digital filters with coefficients of limited word length”, IEEE Trans. on Audio and Electroacoustics, Vol. 20, pp. 206-212, 1972.

[Bac88] R.J.R. Back, A Calculus of Refinements for Program Derivation”, Acta Informatica, Vol. 25, pp. 593-624, 1988.

[Bal85] R. Balzer: “A 15-Year perspective on Automatic Programming”, IEEE Trans. on Software Engineering, Vol. 11, pp. 1257-1267.

[Bar82] A. Barr, E. Feignbaum, eds, “Handbook of Artificial Intelligence, Vol. 2, Addison-Wesley, Reading, MA, 1982.

[Bar86] D. Barstow, “A Perspective on Automatic Programming”, in “Readings in Artificial Intelligence and Software Engineering, eds. C. Rich and R.C. Waters, pp. 537-559, Morgan-Kaufman, 1986.

[Ben90] J.L. Bentley, B.W. Kernighan, “A System for Algorithm Animation: Tutorial and USer Manual”, Unix Research System Papers, Tenth Edition, Volume II, pp. 451-475, 1990.

[Ber89] D.P. Bertsekas, J.N. Tsitsiklis, “Parallel and Distributed Computation: Numerical Methods”, Prentice Hall, Englewood Cliffs, NJ, 1989.

[Bla85] R.E. Blahut, “Fast Algorithms for Digital Signal Processing”, Addison-Wesley, 1985.

[Big89] T.J. Biggerstaff, A.J. Perlis, editors, “Software Reusability”, Addison-Wesley, 1989.

[Bro85] M. Brown, R. Sedgewick, “Techniques for algorithm animation”, IEEE Software, pp. 28-39, January 1985.

[Cha90] P. Chan, “Algorithms for library-specific sizing of combinatorial logic”, Design Automation Conference, pp. 353-356, 1990.
[Cho92] P. Chou, R. Ortega, G. Borrielo, “Synthesis of Mixed Hardware-Software Interfaces in Microcontroller-Based Systems”, ICCAD-92, pp. 488-495, 1992.
[Cor90] T.H. Cormen, C.E. Leiserson, R.L. Rivest, “Introduction to Algorithms”, The MIT Press, Cambridge, MA, 1990.
[Cor93] M.R. Corazao, M. Khalaf, L. Guerra, M. Potkonjak, J. Rabaey, “Instruction Set Mapping for Performance Optimization”, pp. 518-521, ICCAD93, November 1993.
[Cro75] R.E. Crochiere, A. V. Oppenheim, “Analysis of Linear Networks”, Proceeding of the IEEE, Vol. 63, No. 4, pp. 581-595, 1975.
[Dar81] J. Darlington, “An experimental program transformation and synthesis system”, Artificial Intelligence, Vol. 16, No. 1, pp. 1-46, 1981.
[Der85] N. Dershowitz, “Synthesis by Completion”, The Ninth International Joint Conference on Artificial Intelligence, pp. 208-214, 1985.
[Det87] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, A Wang, “Technology mapping in MIS”, Int. Conf. Computer-Aided Design, pp. 116-119, 1987.
[ElG87] A.A. ElGamal, L.A. Hemachandra, I. Shperling, V.K. Wei, “Using Simulated Annealing to Design Good Codes”, IEEE Trans. on Information Theory, Vol. 33, No. 1, pp. 116-123, 1987.
[Ern93] R. Ernst, J. Henkel, T. Benner, “Hardware-Software Cosynthesis for Microcontrollers”, IEEE Design and Test, Vol. 10, No. 4, pp. 64-75, 1993.
[Fei92] E. Feig, S. Winograd, “Fast Algorithms for the Discrete Cosine Transform”, IEEE Transactions on Signal processing, Vol. 40, No. 9, pp. 2174-2193, 1992.
[Flo67] R.W. Floyd, “Assigning Meaning to Programs”, Proc. of the Symposia in Applied Mathematics, Vol. 19, pp. 19-32, 1967.
[Fre87] P. Freeman, “A conceptual analysis of the DRACO approach to constructing software systems”, IEEE Transactions on Software Engineering, Vol. 13, No. 7, pp. 830-844, 1987.
[Gar79] M.R. Garey, D.S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-completeness”, W.H. Freeman, New York, NY, 1979.
[Gra73] A.H. Gray, J.D. Markel, “Digital lattice and ladder filter synthesis”, IEEE Trans. on Audio Electroacoustic, Vol. 21, pp. 491-500, 1973.
[Gre69] C. Green, Application of Theorem Proving to Problem Solving”, Proc. of the First International Joint Conference an Artificial Intelligence, pp. 219-239, 1969.
[Gup94] R.K. Gupta, C.N. Coehlo, G. De Micheli, “Program Implementation Schemes for Hardware-Software System”, IEEE Computer, Vol. 27, No. 1, pp. 48-55, 1991
[Had91] R.A. Haddad, T.W. Parsons, “Digital Signal processing: Theory, Applications, and Hardware”, Computer Science Press, New York, Ny 1991.
[Hof82] C.M. Hoffman, M.J. O’Donnell, “Programming with equations”, ACM Trans. on Programming Languages and Systems, Vol. 4, No. 1, pp. 83-112, 1982.
[Kal93] A. Kalavade, E.A. Lee, “A Hardware-Software Codesign Methodology for DSP Applications”, IEEE Design & Test of Computers, Vol. 10, No. 3, pp. 16-28, 1993.
[Kan79] E. Kant, “A Knowledge-Based Approach to Using Efficiency Estimation in Program Synthesis”, Proc. of the Sixth International Joint Conference an Artificial Intelligence, pp. 457-462, 1979.
[Kan83] E. Kant, “On the Efficient Synthesis of Efficient Programs”, Artificial Intelligence, Vol. 20, No. 3, pp. 253-306, 1983.
[Keu87] K. Keutzer, “DAGON: technology binding and local optimization by DAG matching”, Design Automation Conference, pp. 341-347, 1987.
[Knu70] D.E. Knuth, P.B. Bendix, “Simple word problems in universal algebras”, in “Computational Problems in Abstract algebras”, ed. by J. Leech, Pergamon Press, pp. 263-297, 1970.
[Knu73] D.E. Knuth, “Sorting and Searching”, vol. 3 of The Art of Computer Programming, Addison-Wesley, Reading, MA, 1973.
[Joh83] H.W. Johnson, C.S. Burrus, “The Design of Optimal DFT Algorithms Using Dynamic Programming”, IEEE Trans. on Acoustic, Speech, and Signal Processing, Vol. 27, No. 2, pp. 169-181, 1983.
[Joh90] D.S. Johnson, “Local Optimization and the travelling salesman problem”, 17th Colloquium on Automata, Languages and Programming, pp. 446-461, 1990.
[Lee84] B.G. Lee, “A new algorithm to compute the discrete cosine transform”, IEEE Transactions on Acoustic, Signal, and Speech Processing, Vol. 32, No. 12, pp. 1243-1245, 1984.

[Lee87] E. A. Lee and D. G. Messerschmitt, ``Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing'', *IEEE Trans. on Computers,* Vol. 36, No. 1, pp. 24-35, 1987.

[Lee95] E.A. Lee, T.M. Parks, "Dataflow Process Networks", *Proc. of the IEEE*, Vol. 83, No. 5, pp. 773-799, 1995.

[Lei83] C.E. Leiserson, F.M. Rose, J.B. Saxe, "Optimizing synchronous circuits by retiming", Proceedings of Third Conference on VLSI, pp. 23-36, Computer Science Press, 1983.

[Li93] W-N. Li, A. Lim, P. Agarwal, S. Sahni, "On the circuit implementation problem", IEEE Trans. on CAD, Vol. 12, No. 8, pp. 1147-1156.

[Lin89] T.A. Linden, "Planning by Transformational Synthesis", IEEE Expert, Vol. 4, No. 2, pp. 46-55, 1989.

[Loe88] C. Loeffler, A. Ligtenberg, G.S. Moschyz, "Algorithm-architecture mapping for custom DSP chips", International Symposium on Circuits and Systems, pp. 1953-1956, 1988.

[Lop91] D. Lopresti, "Rapid Implementation of a Genetic Sequence Comparator Using Field-Programmable Logic Arrays", Advanced Research in VLSII, pp. 138-152, 1991.

[Low76] E.L. Lawler, "Combinatorial Optimization: Networks and Matroids", Holt, Rinehart and Winston, New York, NY 1976.

[Low89] M.R. Lowry, "Automating Synthesis through Problem Reformulation", Ph.D. Thesis, Dept. of Computer Science, Stanford university, 1989.

[Low91] M.R. Lowry, R.D. McCartney, eds, "Automating Software Design", AAAI Press, Menlo Park, CA, 1991.

[Man80] Z. Manna, R.J. Waldinger: "A Deductive Approach to Program Synthesis", ACM Trans. on Programming Languages and Systems, Vol. 2, No. 1, pp. 90-121, 1980.

[McC87] R. McCartney, "Synthesizing algorithms with Performance Constraints", Proc. of the Sixth National Conference on Artificial Intelligence, pp. 149-154, 1987.

[McG92] C. McGeoch, "Analyzing Algorithms by Simulation: Variance Reduction Techniques and Simulation Speedups", ACM Computing Surveys, Vol. 24, No. 2, pp. 195-212, 1995.

[Mea80] C.A. Mead, L.A. Conway, "Introduction to VLSI systems", Addison-Wesley, Reading, MA, 1980.

[Mit72] S.K. Mitra, R.J. Sherwood, "Canonic realization of digital filters using the continued fraction expansion", IEEE Trans. on Audio Electroacoustics", Vol. 21, pp. 185-194, 1972.

[Mit93] S.K. Mitra, J.F. Kaiser, "Handbook for Digital Signal Processing", John Wiley & Sons, Inc., New York, NY, 1993

[Not91] S. Note, W. Geurts, F. Catthoor, H. De Man, "Cathedral-III" Architecture-Driven High Level Synthesis for High Throughput DSP Applications", Design Automation Conference, pp. 597-602, 1991.

[Opp92] A.V. Oppenheim, S.H. Nawab, editors, "Symbolic and Knowledge-based Signal Processing", Englewood Cliffs, NJ, 1992.

[Pen93] W.B. Pennebaker, J.L. Mitchell, "JPEG still image data compression standard", Van Nostrand, New York, NY, 1993.

[Pet95] E. Petajan, "The HDTV Grand Alliance Systems", Proc. of IEEE, Vol. 83, No. 7, pp. 1094-1105, 1995.

[Pot89] M. Potkonjak, J.Rabaey, "A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graphs", 26th ACM/IEEE Design Automation Conference, Las Vegas, NV, pp. 7-12, June 1989.

[Pot91] M. Potkonjak, J. Rabaey, "Optimizing the Resource Utilization Using Transformations", Proc. IEEE ICCAD-91, Santa Clara, pp. 88-91, November 1991.

[Pot92a] M. Potkonjak, J. Rabaey, "Pipelining: Just Another Transformation", IEEE ASAP-92, pp. 163-175, 1992.

[Pot92b] M. Potkonjak, J. Rabaey, "Maximally Fast and Arbitrarily Fast Implementation of Linear Computations", IEEE ICCAD-92, Santa Clara, CA, pp. 304-308, November 1992.

[Rab91] J. Rabaey, C. Chu, P. Hoang, M. Potkonjak, "Fast Prototyping of Datapath-Intensive Architectures", IEEE Design and Test of Computers, Vol. 8, No. 2, pp. 40-51, June 1991.

[Rab94] J.M. Rabaey, M. Potkonjak, "Estimating Implementation Bounds for Real Time DSP Application Specific Circuits", IEEE Trans. on CAD, Vol. 13, No. 6, to be published, 1994.

[Rao90] K.R. Rao, P. Yip, "Discrete Cosine Transform", Academic Press, Inc., San Diego, CA 1990.

[Ric88] C. Rich, R.C. Waters, "The Programmer's Apprentice: A Research Overview", IEEE Computer, Vol. 21. No. 11, pp. 10-25, 1988.

[Rob65] J.A. Robinson, "A Machine-Oriented Logic Based on the Resolution Principle", Journal of the ACM, Vol. 12, No. 1, pp. 23-41, 1965.

[Sal93] M.N. Salinas, B.W. Johnson, J.H. Aylor, "Implementation-Independent Model of an Instruction Set Architecture in VHDL", IEEE Design & Test of Computers, Vol. 10, No. 3, pp. 42-54, 1993.

[Set91] D.E. Setliff, R.A. Rutenbar, "On the Feasibility of Synthesizing CAD Software from Specifications: Generating Maze Router Tools in ELF", IEEE Trans. on CAD, Vol. 10, No. 6, pp. 783-801, 1991.

[Sha93] A. Sharma, R. Jain, "Estimating Architectural Resources and Performance for High-Level Synthesis Applica-

tions", IEEE Trans. on VLSI Systems, Vol. 1, No. 2, pp. 175-190, 1993.
[Sim91] G. Simmons, "Contemporary Cryptology: The Science on Information Integrity", IEEE Press, New York, NY 1991.
[Sin88] K.J Singh, A. Wang, R. Brayton, A. Sangiovanni-Vincentelli, "*Timing Optimization of Combinational Logic*", IEEE International Conference on Computer-Aided Design, pp. 282-285, 1988.
[Sri92] M.B. Srivastava, R. Brodersen, "Rapid-Prototyping of Hardware and Software in a Unified Framework", ICCAD-92, pp. 152-155, 1992.
[Ste89] D.M. Steier, "Automating Algorithm Design with an Architecture for General Intelligence", Ph.D. Thesis, School of Computer Science, Carnegie Mellon University, 1989.
[Sue86] N. Suehiro, M. Hatori, "Fast Algorithms for the DFT and other sinusoidal transforms", IEEE Transactions on Acoustic, Signal, and Speech Processing, Vol. 34, No. 6, pp. 642-644, 1986.
[Tho93] D.E. Thomas, J.K. Adams, H. Schmitt, "A Model and Methodology for Hardware-Software Codesign", IEEE Design & Test of Computers, Vol. 10, No. 3, pp. 6-15, 1993.
[Ull89] J.D. Ullman, "Database and Knowledge-Base Systems, Volume II: The New Technologies", Computer Science Press, Rockville, MD, 1989.
[Ver94] I. Verbauwhede, J. Rabaey, "Synthesis of Real-Time Systems: Solutions and Challenges", to appear in Journal of VLSI Signal processing, 1994.
[Vet86] M. Vetterli, A. Ligtenberg, "A discrete Fourier-cosine transform chip", IEEE Journal on Selected Areas in Communications, Vol. 4, No. 1, pp. 49-61, 1986.
[Wal69] R.J. Waldinger, R.C. Lee, "Prow: A Step toward Automatic Program Writing", Proc. of the First International Joint Conference an Artificial Intelligence, pp. 241-252, 1969.
[Wan85] Z. Wang, "Fast Algorithms for discrete W transform and for the discrete Fourier Transform", IEEE Transactions on Acoustic, Speech, and Signal processing, Vol. 32, No. 8, pp.803-816, 1994.
[Wil83] D.S. Wile, "Program Developments: Formal Explanations of Implementations", Comm. of the ACM, Vol. 26, No. 11, pp. 902-911, 1983.
[Win78] S. Winograd, "On Computing the Discrete Fourier Transform", Vol. 32, No. 1, pp. 175-199, 1978.
[Wol93] W. Wolf, "Guest Editor's Introduction: Hardware-Software Codesign", IEEE Design & Test of Computers, Vol. 10, No. 3, page 5, 1993.
[Woo94] N-S. Woo, A.E. Dunlop, W. Wolf, "Codesign from Cospecification", IEEE Computer, Vol. 27, No. 1, pp. 42-48, 1994.
[Yip84] P. Yip, K.R. Rao, "Fast DIT algorithms for DCT and DST", Circuits, Systems, and Signal processing, Vol. 3, No. 4, pp.387-408, 1984.
[Yip85] P. Yip, K.R. Rao, "DIF algorithms for DCT and DST", International Conference on Acoustic, Signal, and Speech Processing, pp. 776-779, 1985.