Algorithm Selection: A Quantitative Computation-Intensive Optimization Approach

Miodrag Potkonjak
C&C Research Laboratories
NEC USA
Princeton, NJ 08540

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

Abstract

Given a set of specifications for a targeted application, algorithm selection refers to choosing the most suitable algorithm for a given goal, among several functionally equivalent algorithms. We demonstrate an extraordinary potential of algorithm selection for achieving high throughput, low cost, and low power implementations.

We 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. We also present an algorithm for cost optimization using algorithm selection. The effectiveness of methodology and algorithms is illustrated using examples.

1.0 Motivations

There is a wide consensus among CAD researchers and system designers that rapidly evolving hardware-software codesign research is a backbone of future CAD technology and design methodology [3, 5, 6, 12, 13].

At a very coarse level of resolution, hardware-software codesign is usually associated with the assignment of various blocks of the targeted computation to either a programmable architecture or to a custom ASIC, so that the cost of the system is minimized while the given performance criteria are met. At a finer level, 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) for a given task.

We recast the hardware-software codesign task from a more global perspective. It is recognized that for a specified functionality of the application, the user not only has a choice of implementation platform, but also may select among a variety of functionally equivalent algorithms. As it is documented in the next section, even on a 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 difference in the resulting quality of the design is often significantly higher than the difference produced by using different synthesis and compilation tools.

Considering simultaneously the selection of both algorithm and architecture is, obviously, often a formidable and cumbersome problem. As the first step, we study the more restricted algorithm selection problem. The fixed implementation is assumed. The restricted problem is interesting on its own. As it is presented in Sections 3 and 4, many of the results for the algorithm selection problem can be easily generalized and applied on the overall algorithm-architecture selection problem.

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 the development of this important aspect of system level design. 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.

2.0 Issues and Classification

2.1 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 role must be recognized outside of the role of transformations. Actually, as it will be demonstrated through the paper, transformations and algorithm selection are distinct, but closely related, 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. **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 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) [2].

2. **Different approximations and use of semantic transformations.** Often the user specifies an 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, but only that which is larger than the user specified threshold.

   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.

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 a random selection of a pivoting element, cannot be transformed in insertion or bubble sort, which are fully deterministic. All three algorithms, of course, produce the identically sorted data sets, but the last two are based on a different solution mechanism than the quick sort.

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 random-number generator. For example, although random-numbers generators of uniformly distributed random numbers between 0 and 1 generate different sequences of random numbers, all random number generators are equally acceptable, as long as the generated sequences are generating uniformly distributed numbers between 0 and 1.

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

A natural and interesting question related to algorithm selection 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 the synthesis process. For example, different optimization approaches are the most appropriate when very few options are available as compared 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 strongly 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 the 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 a few) criteria, which were explicitly targeted during the development of the algorithm. For example, the algorithms are often designed so that the fewest number of multiplications or the fewest number of operations is used, that have shortest critical path, or that the algorithm has exceptional numerical properties.

2. **Finitely many, but very large number.** Sometimes a large number of different algorithms is available. This is the most often 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 is Johnson-Burrus FFT. Combinatorial analysis shows that in this way we can produce several thousands of different FFT algorithms [2].

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 arrises when the initial definition of the algorithm specification has one or more parameters which can be set to any value from some continuous 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 [1].

2.3 What is the difference in the "quality" of different algorithms?

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 in an order of magnitude, in all the design metrics.

The first level is one taken by theoretical computer science and numerical algebra and analysis communities. Foe Asymptotic complexity theory 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. A similar situation is numerical analysis, where speed of convergence to correct solutions is treated in an 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 illustrate credibility of this claim, we analyzed a two dimensional 8X8 DCT.

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

Table 1: All numbers are for 1.2 micron technology.

Due to its acceptance as a part of number image and video compression standards (H261, JPEG, MPEG,...) and widespread use, a large number of fast algorithms started for the DCT was introduced. A recent book [11] provides a good review of the 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, Wang- Suehiro-Hatori's version of the Wang planar rotation-based sparse matrix factorization DCT, DIT - recursive decimation in time algorithm, DIF - recursive decimation in frequency algorithm, QR - QR decomposition based hybrid planar rotation algorithm, Givens - Givens rotation-based algorithm, MCM - automatically synthesized algorithm, which applies only one multiple constant multiplication transformation on the generic and direct- the direct, generic definition of DCT algorithm. Table 5 shows the area of implementations before the 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 a factor greater than 12(35.86 vs. 2.93), energy by a factor of 6.5 (79.57 nJ per sample vs. 12.19 nJ per sample) 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.

2.4 Design Methodology

We believe that the proper way to develop the new design methodology for algorithm selection 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. The synthesis process is cumbersome and involves numerous and 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, the new level of abstraction also requires 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 a 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. During modeling, both size and structural properties of both algorithm and architecture have to be taken into account.

Transformations are algebraic-laws, redundancy manipulation or control flow changes in algorithms so that functionality is not altered. They are the natural complement to algorithm selection. From a 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. 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 Throughput Optimization

3.1 Problem Formulation

Figure 1 illustrates 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, there are a number of different CDFGs (corresponding to different

algorithms) as shown in Figure 1b. The number of options for

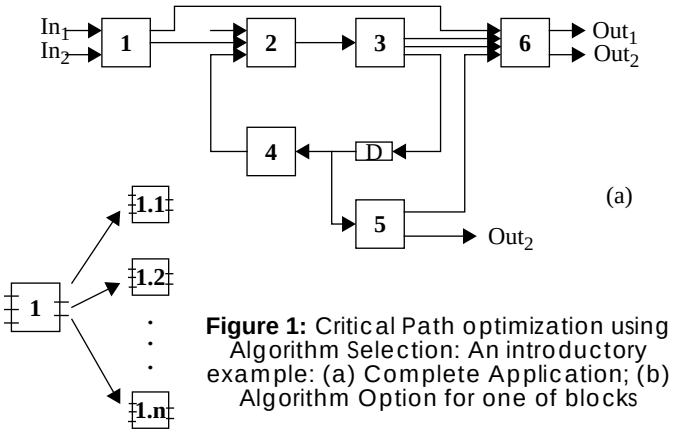

Figure 1: Critical Path optimization using Algorithm Selection: An introductory example: (a) Complete Application; (b) Algorithm Option for one of blocks

block B_i is denoted by n_i. Each block, B_i, has m_i inputs and k_i outputs. Each block is characterized, using the standard critical path algorithm, by a m_i X k_i matrix which contains distances between any two pairs of terminals. The goal is to select for each block a CDFG, so that the overall critical path is minimized.

3.2 Lower-Bound Estimations

We proved that the algorithm selection problem for throughput optimization is an NP-complete problem, by using a transformation from the graph coloring problem [8]. 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 [10].

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

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.

3.3 Relaxation Based Heuristic

One of the most important uses of estimations is that they provide a suitable starting point for the 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:

1. *Eliminate all inferior options for each block;*
2. *Assign to each block the list of all non-inferior algorithm;*

while there is a block with more than one option {

3. *Find ε-critical network;*
4. *For each block find all pairs on inputs/outputs on the critical network;*
5. *For each block on ε-critical network calculate sum of differences between super-algorithm and each of choices.*
6. *Find the block which has the greatest sum of differences;*
7. *Select the algorithm for this block which makes the current overall critical path minimal;*

}

The first preprocessing step eliminates all options which are uniformly inferior to some of the other candidates.

The general idea of the algorithm is to start with the most critical choices (selections which influence the solution most) first, and make them early, so that their 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. ε-critical network and set of differences are an efficient way to abstract the most important parts of overall design which are the most relevant to the final solution.

3.4 Experimental Results

Table 6 illustrates the effectiveness of the throughput optimization using algorithm selection on one audio (LMS DCT domain filter) and two video applications. The average reduction of critical path is by 48%, resulting in an average improvement of throughput by 108%. In all cases lower bounds were tight, indicating that the relaxation based heuristic achieved the optimal solution.

Example	Initial Rate [nsec]	Optimized Rate [nsec]
LMS DCT filter	780	420
NTSC formater	1,060	480
DPCM coder	2,160	1000

Table 2: Sampling Rate (critical path) optimization using Algorithm Selection

4.0 Cost Minimization

Once the throughput requirements are ensured, often the dominant metrics of quality of design is its cost. We now address the problem of optimizing cost under throughput constraints using algorithm selections. A single thread of control is assumed and full custom ASIC is targeted. The generalization to the fully general case is conceptually simple, but a 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 built by limited modification of already available high level synthesis algorithms. We believe that the most effective way for developing techniques for a 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 [9].

4.1 Problem Formulation, Computational Complexity and Lower-Bound Computation

The cost optimization using the algorithm selection problem can be formulized 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, as provided by the Hyper system.

The problem is NP-hard, because even when the application has only one algorithm and only one choice for an algorithm, finding the minimal cost is NP-complete. This is so, because scheduling is an NP-complete problem.

The lower bound estimation algorithm is built on top of Hyper estimation tools and the notion of super block. We assume that the reader is familiar with Hyper estimation techniques (for detail exposition see [9, 11]). First each block is treated individually. For each algorithm for the block and each type of resource (e.g. each type of execution unit or interconnect) a hardware requirements graph is built for all values of available time, starting from the critical path time to the time when only one instance of the resource is required. Then, requirements for each block are build 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. We refer to requirements of such an 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 the final implementation cost is minimized. This cost is the lower bound for the final implementation.

4.2 Algorithm for Area Optimization

The algorithm for cost optimization has two phases. In the first phase, we use min-bounds to select the most appropriate algorithm 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:

1. Using superalgorithm selection for each block construct Resource Utilization Table for each Resource; while there exists 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;

}

Table 2 shows a typical example of Resource Utilization

Block	cycles	IO/Cycle	*/Cycle	Reg	*->+
block1	t1	1	0	37	0
block2	t2	0	.5	24	2.2
block3	t3	.3	2	39	1.3
total	t = Âti	1	2	39	3

Table 3: 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)

Table. The table is constructed in the identical way as the initial allocation in Hyper. For detailed exposure of Hyper scheduling tools see [7]. Each row has as an entry the estimation requirements for the superalgorithm implementations (see Section 4.1). The bottom row has as the entries 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. How many dominant entries the row has. The dominant entry is the entry which has the maximal value in some column. If there are many dominant entries in the row, this block dictates the overall hardware requirements. It is important to make the selection for this block early to get the correct picture of final requirements as soon as possible.

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

3. How many different algorithms are contributing to dominant values. The intuition is similar to the previous case.

4. How sensitive the superalgorithm and choices are to time changes. The choices which are the 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 choices for this block are evaluated. The most suitable algorithm is one which least increases the current overall cost (given in the last row). Steps 4 and 5 are updates which provide the accurate picture of the partial solution after each decision

The second phase is the direct application of the Hyper tools for allocation, scheduling and assignment.

Example	Initial area [mm^2]	Optimized Area [mm^2]
LMS DCT filter	92.96	9.62
NTSC formater	170.77	5.86
DPCM coder	109.77	4.43

Table 4: Optimizing Area using Algorithm Selection

Table 3 shows the area reduction due to algorithm selection. The average improvement is by factor larger than 21 times. In all cases the discrepancy between the lower bound and the final area was less than 10%.

5.0 Related Work and Future Directions

To the best of our knowledge this is the first work which addresses the algorithm selection process as a quantitative optimization intensive CAD effort.

Interestingly, the related and more difficult task of algorithm design has been addressed much more often. For example, the AI community devoted remarkable efforts to develop transformation-based algorithm design tools [4]. However, except for a very limited domain, this effort did not produce a significant impact.

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 the majority of those studies the number of operations or the length of the critical path were targeted as the design metrics, 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.

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 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 synthesis and evaluation infrastructure, integration of algorithm and implementation platform selection, development of algorithm design techniques, and development of software libraries. All future directions are described in detail in [8].

6.0 Conclusion

As a part of an effort to establish CAD-based design methodology for system level design, we introduced the algorithm selection problem. After demonstrating a high impact of the synthesis task on the final implementation, we studied, using quantitative optimization approach, two specific problems in system level design: optimization of throughput and cost using algorithm selection in system level design.

7.0 References:

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

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

[3] P. Chou, R. Ortega, G. Borrielo: “Synthesis of Mixed Hardware-Software Interfaces in Microcontroller-Based Systems”, Proc. of IEEE ICCAD-92, pp. 488-495, 1992.

[4] J. Darlington: “An experimental program transformation and synthesis system”, Artificial Intelligence, Vol. 16, No. 1, pp. 1-46, 1981.

[5] 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, 1994

[6] 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.

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

[8] M. Potkonjak, J.Rabaey: “Algorithm Selection: A quantitative computation-intensive optimization approach”, Technical Report, 1994.

[9] 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.

[10] J.M. Rabaey, M. Potkonjak: “Estimating Implementation Bounds for Real Time DSP Application Specific Circuits”, IEEE Trans. on CAD of IC, Vol. 13, No. 6, pp. 669-683, 1994.

[11] K.R. Rao, P. Yip: “Discrete Cosine Transform”, Academic Press, Inc., San Diego, CA 1990.

[12] M.B. Srivastava, R. Brodersen: “Rapid-Prototyping of Hardware and Software in a Unified Framework”, Proc. of IEEE ICCAD-92, pp. 152-155, 1992.

[13] W. Wolf: “Hardware-Software Co-Design of Embedded Systems”, Proc. of the IEEE, Vol. 82, No. 7, page 967-989, 1994.

[14] W. Ye, R. Ernst, T. Benner, J. Henkel: “Fast Timing Analysis for Hardware-Software Co-Synthesis”, 1993 IEEE International Conference on Computer Design, pp. 452-457, 1994.