

AN INTEGRATED FRAMEWORK OF DESIGN OPTIMIZATION AND SPACE MINIMIZATION FOR DSP APPLICATIONS

Qingfeng Zhuge, Edwin H.-M. Sha

Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA

Chantana Chantrapornchai

Faculty of Science
Silpakorn University
Nakorn Pathom, Thailand 73000

ABSTRACT

This paper presents an Integrated Framework of Design Optimization and Space Minimization (IDOM) for generating the minimum number of functional units with schedule length and memory constraints. Our algorithm efficiently prunes the search space, and eliminates inferior design points by the following: 1) selecting a minimum set of candidate unfolding factors, 2) integrating optimization techniques, i.e. unfolding and extended retiming, to generate a compact schedule. We present the theorems and algorithms for design space exploration. The experimental results show that IDOM algorithm generates smaller configurations, and reduces the search cost to 3% of that consumed by the standard method.

1. INTRODUCTION

In a design process, it's desirable to find the minimum configuration of functional units or processors to execute a program with performance and memory constraints. The challenge is how to find a good design in a huge design space and how to generate high-quality design solution. For DSP applications, optimization techniques such as unfolding and software pipelining are commonly used to optimize the schedule length of a loop schedule [2, 5, 7]. It is important for a designer to be able to choose proper unfolding factor and software pipelining degree efficiently and wisely, so that the search space can be reduced. Therefore, a good design space exploration algorithm needs to exploit the optimization techniques and the properties of design metrics. Without this capability, the design space exploration either easily fails to find any feasible solution, or cannot generate a good design solution.

A number of research results are published on optimization problems [7, 8] and design exploration [1, 4]. However, very few work addresses the combination of various

optimization techniques and their effects on the design exploration results. In this paper, we propose an Integrated Framework for Design Optimization and Space Minimization (IDOM). It distinguishes itself from the traditional design exploration in several ways: (1) it combines unfolding [7] and extended retiming [6] to produce a superior solution satisfying schedule length and memory constraints, (2) it significantly reduces the design space by exploiting the properties of unfolded and retimed data flow graph. Thus, the minimum configuration for an application can be found quickly and effectively.

Our contributions in this paper are summarized as following. We present the theoretical foundation for an integrated framework for design optimization and space minimization. We propose an IDOM algorithm and compare it with the other three design exploration algorithms which are based on traditional approaches. Our experimental results on a set of well-known benchmarks show that IDOM algorithm reduces the search cost to only 3% of that in standard method averagely. Our experiments also show that IDOM algorithm constantly achieves the minimal search costs and the minimal configurations.

In the next section, we presents the basic concepts and theorems of the integrated framework for design optimization and space minimization. Section 3 provides the algorithms and computation cost for several design space exploration algorithms. In Section 4, we present the experimental results on a set of benchmarks. Finally, concluding remarks are provided in Section 5.

2. OPTIMIZATION TECHNIQUES FOR DESIGN SPACE MINIMIZATION

In this section, we present the theoretical foundation of the integrated framework for design optimization and space minimization. We provide an overview of the basic principles of retiming and unfolding in Section 2.1. An in-depth analysis for these optimization techniques based on data flow

This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, USA, and by NECTEC, NT-B-06-4D-16-509, Thailand.

graph model reveals underlying relations between the design parameters. Section 2.2 presents the theorems of this relationship. It shows that the search space and search cost of design space exploration can be greatly reduced based on the properties and relations between the design parameters considering unfolding factor, retiming function and iteration period.

2.1. Basic Principle

A **data flow graph (DFG)** $G = (V, E, d, t)$ is a node-weighted and edge-weighted directed graph, where V is a set of computation nodes, $E \subseteq V \times V$ is a set of edges, d is a function from E to \mathbb{N} representing the number of delays on each edge, and $t(v)$ represents the computation time of each node.

Programs with loops can be represented by cyclic DFGs. An *iteration* is the execution of each node in V exactly once. Inter-iteration dependencies are represented by weighted edges. For any iteration j , an edge e from u to v with delay $d(e)$ conveys that the computation of node v at iteration j depends on the execution of node u at iteration $j - d(e)$. An edge with no delay represents a data dependency within the same iteration. An iteration is associated with a static schedule. A static schedule must obey the precedence relations defined by the DFG. The **cycle period** $c(G)$ of a data-flow graph G is the computation time of the longest zero-delay path, which corresponds to the schedule length without resource constraint.

Given a DFG G which may contain cycles, retiming and unfolding can be used to minimize the execution time of all tasks in one iteration.

Retiming [5] is one of the most effective graph transformation techniques for optimization. It transforms a DFG to minimize its cycle period in polynomial time by redistributing delays in the DFG. The commonly used optimization technique for DSP applications, called *software pipelining*, can be correctly modeled as a retiming.

A *retiming* r is a function from V to \mathbb{Z} that redistributes the delays in the original DFG G , resulting a new DFG $G_r = (V, E, d_r, t)$ such that each iteration still has one execution of each node in G . The delay function changes accordingly to preserve dependencies, i.e., $r(v)$ represents delay units pushed into the edges $v \rightarrow w$, and subtracted from the edges $u \rightarrow v$, where $u, v, w \in G$. Therefore, we have $d_r(e) = d(e) + r(u) - r(v)$ for every edge $u \rightarrow v$ and $d_r(\ell) = d(\ell)$ for every cycle $\ell \in G$.

The extended retiming allows the delays to cut the execution time of a multiple-cycle node to achieve optimal schedule length [6]. Due to the space limitation, we will not illustrate the extended retiming in details.

Unfolding [3, 7] is another effective technique for improving the average cycle period of a static schedule. The original DFG G is unfolded f times, so the unfolded graph G_f consists of f copies of the original node set. Thus, a

schedule with unfolding factor f contains f iterations of the original DFG.

For any DFG G , the average computation time of an iteration is called the **iteration period** of the graph. The instruction-level parallelism between the iterations in an unfolded loop helps to improve the iteration period $P = c(G_f)/f$. For a DFG contains a loop, the iteration period is bounded from below by the iteration bound of the graph which is defined as follows:

Definition 2.1. *The **iteration bound** of a data-flow graph G , denoted by $B(G)$, is the maximum time to delay ratio of all cycles in G . This can be represented by the equation $B(G) = \max_{\forall l} T(l)/D(l)$, where $T(l)$ and $D(l)$ are the summation of all computation times and the summation of delays, respectively, in a cycle l .*

It is clear that unfolding will increase code size by a factor of f . We want to find the minimum f with retiming r so the iteration period of the resultant loop schedule is optimal. But it is very likely that such an optimal f is too large for the program to fit into a small-size on-chip memory. Therefore, we need to explore what will be the good f , r for satisfying both iteration period and memory constraints.

2.2. Theorems

The problem in combining unfolding and retiming is to choose an unfolding factor that can achieve the iteration period constraint via retiming. By exploiting the properties of unfolded graph, we can find the relationship between unfolding factor and corresponding minimum feasible cycle period as stated in Theorem 2.1 which shows the necessary and sufficient conditions to find a legal static schedule [2].

Theorem 2.1. *Let $G = (V, E, d, t)$ be a given data flow graph, $f \in \mathbb{Z}^+$ an unfolding factor and $c \in \mathbb{R}$ a cycle period, there exists a legal static schedule without resource constraints iff $c/f \geq B(G)$ and $c \geq \max_v t(v)$, $\forall v \in V$. Thus, given unfolding factor f , the minimum cycle period $c_{\min}(G_f) = \max(\max_v t(v), \lceil f \cdot B(G) \rceil)$.*

On the other hand, given an iteration period constraint, we can quickly know the feasible unfolding factors that are possible to produce a schedule that satisfies the constraint as stated in the following theorem.

Theorem 2.2. *Let $G = (V, E, d, t)$ be a data flow graph, f an unfolding factor, P a given iteration period constraint. The following statements are equivalent:*

1. *There exists a legal static schedule of unfolded graph G_f with iteration period less than equal to P .*
2. $B(G) \cdot f \leq c_{\min}(G, f) \leq P \cdot f$.

For example, given the cycle period of an unfolded graph $c_{\min}(G_f) = 5$, we would like to know if unfolding factor $f = 2$ can achieve the iteration period constraint $P = 7/3$. Since $c_{\min}(G_f) > P \cdot f = 14/3$, the iteration period constraint cannot be achieved with unfolding factor $f = 2$.

By exploiting the properties of the unfolded graph in the previous theorems, we are able to directly obtain the feasible cycle period of the unfolded and retimed data flow graph *before* really doing unfolding, retiming and scheduling, so that the search space and search cost can be greatly reduced.

3. ALGORITHMS

In this section, we describe three different design space exploration algorithms given a set of functional unit types. Each algorithm employs different techniques to perform the design space exploration and produce optimized design solutions. We will also show their computation costs.

The algorithms are: STDu, STDur and IDOM. Algorithm STDu is a standard method which uses unfolding to optimize the schedule length and search the design space. Given a data flow graph G , the iteration period constraint P , and the maximum unfolding factor f_{\max} decided by the memory constraint. Algorithm STDu generates the unfolded graph G_f for each unfolding factor $1 \leq f \leq f_{\max}$. For each unfolded graph, the algorithm computes the upper bound ub_i for each type of functional units by As Soon As Possible scheduling. Then, the algorithm schedule the DFG G with each possible configuration using list scheduling. The search space for one unfolding factor consists of $\prod_i ub_i$ points. Algorithm STDu exhaustively searches the space for f_{\max} unfolded graphs. Note that using unfolding only may not be able to find a solution satisfying iteration period constraint even with unlimited functional units, because the unfolding factor is upper bounded by memory constraint.

Algorithm STDur is another standard method that applies retiming to optimize schedule length of the unfolded graphs. By using retiming, the cycle period of an unfolded graph is optimized before scheduling. Therefore, algorithm STDur is able to find more feasible solution than algorithm STDu. However, retiming introduces more computation cost.

We develop an algorithm to implement the Integrated Framework of Design Optimization and Space Minimization (IDOM). The IDOM algorithm prunes the design space by exploiting the graph properties and integrating unfolding and extended retiming to produce high quality design solution. In step 1, the algorithm computes the minimum feasible cycle period c_{\min} using Theorem 2.1. In step 2, it immediately eliminates the infeasible unfolding factors from a set of unfolding factors $1 \leq f \leq f_{\max}$ by using Theorem 2.2, which selects a minimum set of candidate unfolding factors $F = \{f : c_{\min}/f \leq P\}$. It means that we do not need to generate and schedule all the unfolded

graphs. Therefore, the computation cost of design space exploration is significantly reduced. Step 3 generates unfolded graphs for each unfolding factor $f \in F$. In step 4, the algorithm performs extended retiming instead of the traditional one to find optimal cycle period for an unfolded graph. Step 5 computes the upper bounds of functional units. Step 6 computes the lower bound of each type of functional units with latency bound $\lfloor P \cdot f \rfloor$ to further reduce the search space. Finally, in step 7, the algorithm schedules the retimed unfolded graphs with list scheduling in the pruned search space.

The computation cost of each design space exploration algorithm can be derived using the complexity of list scheduling as a unit of the computation cost. The complexity of unfolding is $O(f|V| + f|E|)$, and the complexity of list scheduling is $O(|V| + |E|)$. Let F be a set of feasible unfolding factors, and S_T be the set of points in the design space that will be searched by a design space exploration algorithm. Here we compute the search cost of an algorithm in terms of the number of times list scheduling is applied to the original graph. For STDu, the cost of searching can be easily calculated as the addition of unfolding cost and scheduling cost, i.e., $C_u + C_s = \sum_{f_i \in F} f_i + \sum_{f_i \in F} f_i |S_T|$.

The search cost of the other algorithms can be calculated in the similar manner. Since we know that the complexity of retiming is $O(|V||E| \log |V|)$, we can compute cost of applying retiming to the unfolded graph as $C_{tr} = \sum_{f_i \in F} f_i^2 |V| + \sum_{f_i \in F} f_i^2 |V| \log(f_i |V|)$. Thus, the cost of searching for STDur is $C_u + C_{tr} + C_s$.

The IDOM algorithms significantly reduces $|F|$ and $|S_T|$. The computation cost of extended retiming is $C_{er} = f_i^2 |V|$. The search cost of IDOM is $C_u + C_{er} + C_s$.

4. EXPERIMENTAL RESULTS

To demonstrate the performance and quality of IDOM algorithm, we conduct a series of experiments on benchmarks including biquad filter (Biquad), differential equation (DEQ), allpole filter (All pole), fifth order elliptic filter (Elliptic) and 4-stage lattice filter (4-Stage) as shown in Table 1. Each experimental case is processed by two design exploration algorithms, one is STDur, the other is IDOM. To make the cases more complicated for design space exploration, we apply different slow down factors to the original circuits [5, 6], and arbitrarily choose the computation times for additions and multiplications. We also assume the maximum unfolding factor is 5, which is bounded by memory constraint.

In the experiment of ‘‘Biquad’’, assume that addition takes 1 time unit and multiplication 4 time units. We apply a slow-down factor of 4 to the original circuit. The resulting circuit has an iteration bound of 3/2. Assume that the iteration period constraint is 5/3. An configuration satisfies the iteration

period constraints can be found by STDur is shown in column “Solutions”, with unfolding factor $f=4$, 3 adders and 16 multipliers. It has an iteration period (field “P”) of 6/4. With the same case, IDOM find the minimum configuration of 3 adders and 10 multipliers with unfolding factor only 3. The resulting schedule satisfies both iteration period and memory constraints. Furthermore, the search cost of IDOM is only 4% of STDur, as shown in column “Ratio”.

The experiment settings for the other cases are described in the following. For “DEQ”, assume that an addition operation takes 1 time unit, and a multiplication operation takes 3 time units. We use a slowdown factor of 5, iteration bound of 8/5 and iteration period of 7/4. For “Allpole”, an addition operation takes 2 time units, and a multiplication operation takes 5 time units. We use a slowdown factor of 5, iteration bound of 18/5 and iteration period constraint of 15/4. For “Elliptic”, an addition takes 1 time unit, and a multiplication takes 5 time units. We use a slowdown factor of 5, iteration bound of 18/5 and iteration period constraint of 15/4. For “4-Stage” lattice, an addition takes 1 time unit, and a multiplication 6 time units. We use a slowdown factor of 2, iteration bound of 7/4 and iteration period constraint of 9/5.

Applying the algorithms on all these experimental cases clearly yields the conclusion that IDOM significantly reduces the search cost of design exploration process compared to the standard method. Its search cost is only 3% of STDur on average. Furthermore, IDOM always find the smaller configurations for all the cases.

| Benchmarks (Algo.) | Search Points | Search Cost | Ratio | Solutions | | | |
|-----------------------|------------------|----------------|--------------|-----------|-----------|-----------|-------------|
| | | | | f | Add | Mult | P |
| Biquad(STDur) | 228 | 2165 | | 4 | 4 | 16 | 6/4 |
| Biquad(IDOM) | 4 | 87 | 0.04 | 3 | 3 | 10 | 5/3 |
| DEQ(STDur) | 486 | 6017 | | 5 | 10 | 18 | 8/5 |
| DEQ(IDOM) | 6 | 120 | 0.02 | 3 | 5 | 15 | 5/3 |
| Allpole(STDur) | 510 | 7957 | | 5 | 10 | 10 | 18/5 |
| Allpole(IDOM) | 6 | 156 | 0.02 | 3 | 10 | 3 | 11/3 |
| Elliptic(STDur) | 694 | 19062 | | F | F | F | F |
| Elliptic(IDOM) | 2 | 142 | 0.007 | 2 | 6 | 9 | 5 |
| 4-Stage(STDur) | 909 | 15329 | | F | F | F | F |
| 4-Stage(IDOM) | 55 | 640 | 0.04 | 4 | 7 | 33 | 7/4 |

Table 1: Applying STDur and IDOM algorithms on various benchmarks.

5. CONCLUSION

In this paper, we have presented an Integrated Framework of Design Optimization and Space Minimization (IDOM) to find the minimum configuration under schedule length and memory constraints. The novelty of this framework is that it integrates several optimization techniques and exploits

the properties of unfolded retimed DFG to prune the design space and generate superior results. The optimization techniques combined in IDOM framework are unfolding and extended retiming. The study of the properties of unfolded retimed DFG efficiently reduces the number of feasible unfolding factors. The experimental results show that the search cost of IDOM is only 3% of the standard method on average. The development of this integrated framework shows the significant impacts of proper utilization of optimization techniques and exploitation of graph properties on the design space exploration cost and the quality of design solutions.

6. REFERENCES

- [1] C. Chantrapornchai, E. H.-M. Sha, and X. S. Hu. Efficient acceptable design exploration based on module utility selection. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 19(1):19–29, Jan. 2000.
- [2] L.-F. Chao and E. H.-M. Sha. Static scheduling for synthesis of DSP algorithms on various models. *J. of VLSI Signal Processing*, 10:207–223, 1995.
- [3] L.-F. Chao and E. H.-M. Sha. Scheduling data-flow graphs via retiming and unfolding. *IEEE Trans. on Parallel and Distributed Systems*, 8(12):1259–1267, Dec. 1997.
- [4] S. Chaudhuri, S. A. Blythe, and R. A. Walker. A solution methodology for exact design space exploration in a three dimensional design space. *IEEE Trans. on VLSI Systems*, 5(1):69–81, Mar. 1997.
- [5] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, Aug. 1991.
- [6] T. O’Neil, S. Tongsimma, and E. H.-M. Sha. Extended retiming: Optimal scheduling via a graph-theoretical approach. In *Proc. IEEE Intl. Conf. on Acoustics, Speech, and Signal Processing*, volume 4, pages 2001–2004, Mar. 1999.
- [7] K. K. Parhi. *VLSI Digital Signal Processing Systems: Design and Implementation*. John Wiley & Sons, 1999.
- [8] Q. Zhuge, Z. Shao, and E. H.-M. Sha. Optimal code size reduction for software-pipelined loops on dsp applications. In *Proc. IACC Intl. Conf. on Parallel Processing*, pages 613–620, Aug. 2002.