An Optimal Software-Pipelining Method
for Instruction-Level Parallel Processors based on Scaled Retiming

F. Fernández  A. Sánchez  A. Duarte
Dept. Fotónica, FI-UPM ESCET-URJC ESCET-URJC
Campus Montegancedo  Campus de Móstoles  Campus de Móstoles
28660 Madrid, Spain  28933 Madrid, Spain  28933 Madrid, Spain
felipe.fernandez@es.bosch.com an.sanchez@escet.urjc.es a.duarte@escet.urjc.es

Abstract

Software pipelining is an instruction-level loop scheduling method for achieving high performance fine-grain parallelism on VLIW processors. This paper presents a novel software pipelining method for non-pipelining parallel processors based on integer scaling and retiming transformations. This approach generalises and simplifies the analogous extended retiming model of Sha et al. [2][3][4]. Matrix techniques are used in order to simplify the corresponding graph transformations. Some general properties rescued from Algebraic Graph Theory are applied in order to obtain general scheduling techniques: Node and Cycle methods.

The two-phase scheduling method considered is first defined by means of two standard linear programming problems. We transform the corresponding problems into some variants of the maximum cost to time ratio problem and shortest path problem, in order to obtain efficient polynomial time algorithms. An example of software pipelining optimization of a digital correlator is also given.

1. Introduction

Real-time applications in areas like signal and image processing usually require high computer performance. Moreover, the demand for optimized solutions in multidimensional signal processing applications has recently increased with the evolutions of these techniques and the extensive use of new parallel processors.

Most of computation-intensive DSP applications require to be executed under some specific time constraints associated with high data input rates. The design of software for the execution of repetitive low-level operations is strongly dependent on the optimization techniques. On the other hand, to optimize the execution time of computation intensive applications, the designers need to analyze the embedded parallelism in repetitive patterns usually referred to a nested loop.

An important factor in high performance parallel computers today is the optimization of parallelism in nested loops of scientific programs. With the advent of VLIW superscalar and pipelined processors, the exploitation of fine-grain instruction-level parallelism has become a major challenge of parallel compilers. Software pipelining has been proposed as an effective fine-grain scheduling technique for this objective. The key idea of software pipelining [5] is to increase instruction-level parallelism, and then the throughput of a loop by properly overlapping several iterations of the loop in a single execution cycle [5]. Software pipelining allows starting the execution of one iteration of the loop before another finishes.

In general, iterative and recursive algorithms, found in DSP and image processing applications, can be represented by a data-flow graphs (DFG). The nodes of a DFG represent tasks and the edges represent data dependencies among the tasks. Graph techniques are applied to the DFG in order to increase the degree of parallelism of involved algorithms. The process of assigning a starting time to each node at computation time on a DFG is known as static scheduling. The execution of all tasks of a DFG is called iteration, and the average computation time per iteration is called iteration period. Sha et al. [5] presented an algorithm, called DFG scheduling based on a Gantt diagram, used to construct a valid schedule for the considered DFG.

In our approach, we consider an integer period and integer time model for the schedule, where each operation start in the beginning of a time unit and has an integer execution time. Non-pipelining static schedule, where the next copy of a node cannot start execution before the previous copy has finished execution, is also used. Our proposed model also considers absence of resource constraints.

A great deal of research has been done in order to optimize DSP applications by applying retiming techniques [6]. Traditional retiming techniques do not
always achieve optimal schedule. Extended retiming [2][3][4] allows transforming a DFG into a new one whose clock period is optimal. In this paper, we propose a new simpler model than extended retiming called scaled retiming model. The main advantage of our model is that the weights of DFG arcs obtained are integer numbers (in the scheduling graph consider by extended retiming, these weights are rational), and this way the resulting model is simpler and easier to transform. Moreover, our approach has a more general orientation since it is based on Algebraic Graph Theory [12][16][19] and it gives a general scheduling solution for a considered DFG.

In the considered model, the achievement of an optimal scheduling for a DFG is carried out in two phases: Phase I computes the minimum iteration period of the graph and Phase II obtains the optimal retiming for each node that minimizes the total number of algorithmic delays of the graph. The optimal schedule of a general iteration of a loop is directly derived from the obtained retiming.

We also show how the two phases for the calculation of the optimal scheduling can be efficiently implemented using derived versions of Bellman-Ford algorithm [6][11] and Howard algorithm [13][14].

The rest of the paper is organized as follows: Section 2 reviews the extended retiming model proposed by Sha et al.; Section 3 introduces our proposed scaled retiming model; in Section 4, a method for obtaining the optimal retiming of a DFG using linear programming is explained; Section 5 describes an optimal scheduling procedure based on the maximum cost to time ratio problem and shortest path problem; Section 6 illustrates our approach using a digital correlator example and Section 7 outlines the conclusions.

2. Extended retiming model

The Data Flow Graph (DFG) considered by Sha et al. [2][3][4] can be defined as an arc and node weighted digraph model:

$$G = \{ V, A, [D], [T] \}$$

where $V$ is the ordered set of nodes that represents the set of task $(|V|=n)$, $A$ is the ordered set of oriented edges or arcs of the corresponding graph and represents the set of precedence relations among the nodes $(|A|=m)$, $[D]$ is an ordered set of delays, associated with each arc of the graph (algorithmic delays) and $[T]$ is an ordered set of computational times associated with each node (called computational or architectural constraint delays).

Note that in this framework the computational delays are defined on the nodes of the corresponding DFG.

Sha et al. [2][3][4] has introduced a new transformation method named extended retiming. This extended model has been introduced since the traditional retiming does not always achieve optimal schedules [2][3][4]. Extended retiming model obtains a new graph $G_S = \{ V, A, [W] \}$ called scheduling graph from the original data flow graph $G$. The unique change is the reweighting of each arc $a$ according to the formula:

$$w(a) = d(a) - t(u)/\alpha$$

where $a$ is the arc between $u$ and $v$, $d(a)$ are the algorithmic delays, $t(u)$ is the computational time of the node $u$ and $\alpha$ is the iteration period.

In Sha’s approach, the new arc weights of scheduling graph obtained are rational. Our alternative point of view is to reweigh also the arcs but with integer values ($\alpha$-scaled), so we obtain a new-scaled scheduling graph in which all arcs have an integer weight given by:

$$w(a) = \alpha d(a) - t(u)$$

The new equivalent scaled slack graph $G_{sh}$ introduced in this paper and more deeply described in next sections, can be represented by:

$$G_{sh} = \{ V, A, \alpha[D]-[T] \}$$

The iteration period or bound $\alpha$ is classically defined as the maximum computational time to algorithmic delay ratio for all circuits in $G$ [22]. It can be shown that $\alpha$ is a feasible iteration period, if the slack graph contains no negative-weight circuits.

Leiserson and Saxe [10] have introduced a very simple algorithm to solve the minimum-retiming problem based on the shortest path method. From this schema, Sha et al. [2][3][4] have developed an extended retiming method for the corresponding rational scheduling graph.

To compute an optimal retiming, Sha introduces an additional reference node in the scheduling graph with zero-weight directed arcs from this node to all the original nodes. In this new graph, the shortest path from the new node to the rest is calculated. In order to obtain a rational normalised retiming from the rational scheduling graph, the following relative complex formula is given [4]:

$$r(v) = \lfloor sh(v)-X \rfloor + \min \{ 1, (\alpha t(v) \cdot (sh(v)-X)+sh(v)-X) \}$$

where $sh(v)$ is the shortest path from the reference node and X is the minimum shortest path.
Next section describes our equivalent graph model that avoids the previous rational to integer retiming conversions and simplifies the corresponding process.

3. Scaled retiming model

The Data Flow Graph (DFG) considered in this paper is analogously described throughout a double weighted arc digraph model:

\[ G = \{ V, A, [D], [T] \} \]

where \( V \) is the ordered set of nodes that represents the set of task \( (|V| = n) \), \( A \) is the ordered set of oriented edges or arcs of the corresponding graph and represents the set of precedence relations among the nodes \( (|A| = m) \), \([D]\) is an ordered set of delays, associated with each arc of the graph (algorithmic delays) and \([T]\) is an ordered set of computation times associated with each arc (called computational or architectural constraint delays).

Figure 1 shows a simple example of this DFG, where the numbers above the arcs represent computational delays and the black solid rectangles represent algorithmic delays.

Note that in our framework the computational delays are defined on the arcs and not on the nodes of the DFG.

Standard retiming R \([10][17]\) is a transformation used to change register delay locations in synchronous circuits or DFG without affecting the input/output characteristics of the circuit.

The scaled retiming considered in this paper initially multiply by a constant \( \alpha \) all the algorithmic delays of a DFG. The \( \alpha \)-slow graph defined in \([10]\) is an analogous operational transformation to obtain a slowed down version of the initial DFG. However, the \( \alpha \)-scaled graph considered in this paper has the same data rate and produces a result every clock period of the initial circuit. This implies an implicit transformation of each initial register into \( \alpha \) equivalent \( \alpha \)-overclocked registers (named phase registers). This consideration allows us to only deal with integral retiming transformations. This way it is possible to relocate a fraction of an original register, as in the extended retiming model, but using an integer number of phase registers.

![Fig.1: Data Flow Graph G1](image)

Node Method. A matrix form of the scaled retiming transformation can be defined throughout the following expression:

\[ [DF] = [C] [R] + \alpha [DI] \]

where \([DI]\) and \([DF]\) are respectively the column \( m \)-vectors of initial and final algorithmic delays; \( \alpha \) is the scaling factor (iteration period) applied; \([R]\) is a column \( n \)-vector that represents the corresponding retiming applied; and \([C]\) is an arc-node incidence matrix defined as

\[ c_{ij} = \begin{cases} 0 & (v_j \text{ is the tail of } e_i) \\ 1 & (v_j \text{ is the head of } e_i) \\ 0 & (\text{Else}) \end{cases} \]

From algebraic graph theory \([19][20]\) the following properties are directly deduced:

- Retiming is a potential vector, consequently the following expression holds

\[ [DF] = [C][R] + [\alpha DI] = [C][([R]+k)\alpha DI] \]

- Incremental delay graph is gradient vector, and for that reason the incremental delay vector \([DF]-[\alpha DI]\) is orthogonal to an every cycle vector. Therefore, the following dual cycle method can be used.

Cycle Method. A matrix form of the scaled retiming transformation can also be defined throughout the following dual scaled retiming method:

\[ [B] [DF] = [B] [\alpha DI] \]

or \([B][DF-\alpha DI]=[B][\Delta D]=[0]\)

where \([B]\) is a cycle-arc incidence matrix of each connected component of the corresponding DFG, defined as

\[ b_{ij} = \begin{cases} 1 & (e_j \text{ belongs to cycle } i \text{ and has the same sense}) \\ -1 & (e_j \text{ belongs to cycle } i \text{ and has the opposite sense}) \\ 0 & (\text{Else}) \end{cases} \]

The rows of a cycle matrix can be easily computed as a basis of the kernel of the corresponding transposed arc-node incidence matrix \([C]^T\).

The iteration period or scaling factor \( \alpha \) can also be specified by the following constraint

\[ [B][T] \geq [B][\alpha DI] \]

where \([B] \geq 0\) represents in this case the complete circuit-arc incidence matrix.
In cycle method, the retiming vector \([R]\) can be easily computed by means of a spanning tree for each connected component of the DFG. An arbitrary value can be assigned to retiming of each root nodes. The remainder nodes retiming can be easily computed considering the incremental delays of the corresponding tree branches.

Therefore, cycle and cocycle subspaces \([I]\) can be used to compute the corresponding optimal retiming solution.

4. Optimal scheduling via two-phase linear programming

This section considers the problem of finding an optimal scheduling for a DFG, which can be solved in two steps.
I Minimisation of integer iteration period
II Minimisation of total number of delays

Phase I. Scaling factor can be found in polynomial time by solving the following linear program \([15][21]\):

\[
[DF] = [C] [R] + \alpha [DI] \\
[DF] \geq [T] \\
[R] \geq 0 \\
\text{Minimise } \alpha_i \\
\alpha = [\alpha_i]
\]

Phase II. Retiming can be found by solving the following linear program \([15][21]\), using the scaling factor \(\alpha\) obtained in phase I.

\[
[DF] = [C] [R] + \alpha [DI] \\
[DF] \geq [T] \\
[R] \geq 0 \\
\text{Minimise } ([1,1,\ldots,1][DF])
\]

Node-arc incidence matrices \([C]\) are totally unimodular \([20]\). Therefore, solving the previous linear programming problem gives always an integer optimum solution that can be found in polynomial time.

Phases I and II can be optimised by extracting the strongly connected components of the graph and solving the problem separately for each strongly connected component.

The general retiming is given by:

\[
[RG] = [R] + k
\]

where \(k\) represents an arbitrary constant.

If it is supposed that all tasks of the DFG are referred or aligned to time zero, the initial scheduling can be easily obtained by relocating the tasks in the opposite direction to the registers or delays relocation.

\[
[S(0)] = -(RG) = -(R + k) = [R] + \max [R] + \phi_0
\]

where the scheduling vector \([S(0)]\geq 0\) and \(k=\max[R]-\phi_0\).

The initial phase \(\phi_0\) represents the initial schedule value:

\(0\leq \phi_0 \leq \alpha - 1\).

The general scheduling that can be graphically shown in a Gantt diagram, has the following expression:

\[S(i)] = \alpha i + [S(0)]\]

where \(i\) is the iteration number.

Example

For the DFG of Figure 1, the minimum iteration period \(\alpha=4\) is obtained in phase I, applying the corresponding linear programming node algorithm. This iteration bound is also given by the referred dual circuit expression:

\[\alpha = (3+2+3)/2=4\]

Figure 2 shows the corresponding scaled slack graph \(G_{1H}\) defined in Section 2:

\[G_{1H} = \{ V, A, 4[D]-[T] \} \]

Fig. 2: Slack Graph \(G_{1H}\)

In phase II, we obtain the following scaled retiming:

\[R(A) = 5 \quad S(A) = 0 + 4 i + \phi_0 \]
\[R(B) = 2 \quad S(B) = 3 + 4 i + \phi_0 \]
\[R(C) = 0 \quad S(C) = 5 + 4 i + \phi_0 \]

The scheduling diagram for the iterations \(i=0..2\) is given in Figure 3.

Fig. 3: Scheduling Graph with \(\phi_0 = 0\).

This particular solution matches the solution presented in \([2][3][4]\). Four different integer solutions \(\alpha=4\) can be
obtained considering an initial phase $\varphi_\nu = 0.3$. The difference among them is also observed in the corresponding prologue and epilogue.

5. Efficient optimal scheduling algorithms

In this section we are going to show more efficient ways to obtain an optimal scheduling based on the scaled retiming model. As in fourth section, the optimal scheduling is obtained in two analogous phases.

I. Scaling computation. In the first phase the Maximum Cost to Time Ratio Problem (MCTR) is solved to obtain the minimum integer iteration period $\alpha$.

II. Retiming computation. In the second phase, using the iteration period obtained in previous phase, the Shortest Path Problem (SPP) is computed to find an optimal retiming $[R]$ for the considered scaled DFG.

The classical algorithms for the maximum cost to time ratio problem (MCTR) can compute the iteration period of a DFG using the following specification

$$\alpha = \max \left( \frac{[B][T]}{[B][DI]} \right)$$

where $[B] \geq 0$ represents in this case the complete circuit-arc incidence matrix of all circuits of the corresponding DFG.

There are several polynomial algorithms in the literature that can compute that iteration period, such that Karp’s algorithm $\Theta (m \cdot n)$, Dasdan’s algorithm $O(m \cdot n)$ and Lawler’s algorithm $(\log(n)O(m \cdot n))$. In [8][9] there is a practical review of the most important algorithms to compute the MCTR.

We have implemented the Lawler’s algorithm [18] slightly modified. This algorithm is based on a binary search over the possible values of $\alpha$. The maximum value is the sum of all computational delays and the minimum, the lower computational delay. The algorithm checks for each iteration period if there is a negative cycle in the DFG, throughout Bellman-Ford algorithm. If one is found, then the chosen $\alpha$ is too large, so it is decreased; if it is not, then the chosen $\alpha$ is too small, so it is increased. The algorithm finishes when the interval for the possible value of $\alpha$ becomes too small. This binary search has a well known complexity of $O(\log(n))$, but in our case, the iteration period has the constraint that it is an integer value, so the convergence of the algorithm is very fast.

Another very interesting algorithm is the Howard’s Policy Improvement Scheme [13][14], which has a known bound of exponential complexity, but experimentally is almost linear.

In the second phase (Retiming Computation), the scaled slack graph $G_H$ is constructed, using the iteration period $\alpha$ calculated in phase I:

$$G_H = \{ V, A, \alpha[DI][T] \}$$

An optimal retiming is obtained solving the corresponding shortest path problem on the slack graph $G_H$, (with an additional reference node). The normalized-scaled retiming solution is directly obtained by means of the following expression:

$$[R] = [\text{sh}(V)] - \max \{ \text{sh}(V) \}$$

where $\text{sh}(V)$ represents the correspondent shortest path algorithm vector from the reference node.

We have used the Bellman-Ford algorithm to compute the shortest path $O(m \cdot n)$ [6][11]. Note that in this phase Howard’s Policy Improvement algorithm can also be used [13][14].

6. Software pipelining of a digital correlator

In this section, we apply the methodology developed in previous sections to a real example. In figure 4 we present the DFG of a digital correlator, used by Leiserson and Saxe[10].

The algorithms described in sections 4 and 5 are applied to this DFG. These algorithms (Linear Programming, Integer Lawler and Bellman-Ford) have been developed in MATLAB, since in this environment they are very easy to implement and compare their relative performance.

The relative running time and the results are shown in tables 1 and 2.

As we can see in these tables, with both methods, we obtain the same solution. The only difference is in the computation time (the solution obtained by linear programming is about ten times slower).

The general schedule vector is in this case:

$$[S(i)] = \alpha i + [S(0)] = \varphi_\nu + 10 i + 24 - [R]$$
Table 1: Solution with linear programming algorithm.

<table>
<thead>
<tr>
<th>Phase I</th>
<th>CPU Time (s)</th>
<th>Optimal solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lin. Prog.</td>
<td>0.35</td>
<td>( \alpha = 10 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase II</th>
<th>CPU Time (s)</th>
<th>Optimal solution</th>
</tr>
</thead>
</table>

Table 2: Solution with Lawler and Bellman-Ford algorithm.

<table>
<thead>
<tr>
<th>Phase I</th>
<th>CPU Time (s)</th>
<th>Optimal solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer Lawler</td>
<td>0.05</td>
<td>( \alpha = 10 )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Phase II</th>
<th>CPU Time (s)</th>
<th>Optimal solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bellman-Ford</td>
<td>0.01</td>
<td>( R[n_1] = 0 ), ( R[n_2] = 24 ) ( R[n_3] = 10 ), ( R[n_4] = 21 ) ( R[n_5] = 17 ), ( R[n_6] = 14 ) ( R[n_7] = 24 ), ( R[n_8] = 7 )</td>
</tr>
</tbody>
</table>

7. Conclusions

This paper has presented a novel software pipelining graph technique based on scaled retiming and Algebraic Graph Theory. A two-phase linear programming gives a very simple polynomial time solution for the optimal scheduling problem. Alternative methods based on Bellman-Ford and Howard algorithms have been considered in order to obtain more efficient solutions. An optimization of Lawler algorithm for the computation of the integer iteration period has also been described.

The experimental results, carry out on the optimized software pipelining algorithms presented, have shown a very good running time behavior.

8. References


