Harmonic Scheduling
of Linear Recurrences for Digital Filter Design *

Haigeng Wang Nikil Dutt Alexandru Nicolau
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92717

Abstract

Linear difference equations involving recurrences are fundamental equations that describe many important signal processing applications. For high sample rate digital filter applications, we need to effectively parallelize the linear difference equations used to describe digital filters—a difficult task due to the recurrences inherent in the data dependences. We present a novel approach, Harmonic Scheduling, that exploits parallelism in these recurrences beyond loop-carried dependencies, and which generates optimal schedules for parallel evaluation of linear difference equations with resource constraints. This approach also enables us to derive a parallel schedule with minimum control overhead, given an execution time with resource constraints. We also present a Harmonic Scheduling algorithm that generates optimal schedules for linear filters described by second-order difference equations with resource constraints.

1 Introduction

Digital filter designs are typically described using linear difference equations that express the filter’s current response $y(nT)$ as a function of past responses:

$$y(nT) = a_0x(nT) + a_1x((n - 1)T) + a_2x((n - 2)T) + \cdots + a_k x((n - k)T) + \cdots + b_0 y((n - 2)T) + \cdots + b_m y((n - m)T). \quad (1)$$

Here the filter response at time $nT$, where $T$ is the sample period and $n$ is a running index, is the sum of product terms of $m$ past responses, $y((n - 1)T), \ldots, y((n - m)T)$, where $m$ is the order of the difference equation. The pattern of data dependences exhibited by (1) involves recurrences, since $i$ is a linear combination of the outputs at time $t - 1, 2, \ldots, i - m$. This recurrent dependence pattern is often referred to as a loop-carried dependence (LCD). The difference equation (1) is a special case of band linear recurrences. In many high sample rate applications, it is crucial to parallelize the classes of signal processing described by the linear difference equation, in order to maximize throughput given a limited number of resources (e.g., functional units or chip area). In the high-level synthesis of high performance signal processors, we need effective algorithms for:

- (1) minimizing resources given a throughput constraint for a digital filter, or (2) maximizing throughput (i.e., generating optimal schedules), given resource constraints for the filter. Both kinds of algorithms require effective techniques for parallelizing difference equations.

However, difference equations are difficult to parallelize due to the LCD’s in the recurrences, particularly when resource constraints are imposed. Previous research in scheduling for high level synthesis has exploited parallelism for signal processing in the forms of pipelining (i.e., overlapping the processing of a sample input with subsequent inputs), and concurrency (i.e., simultaneously computing operations that may be executed in parallel in processing one sample sequence while preserving data dependences). Although these scheduling techniques produce good results, we can further exploit parallelism by taking advantage of the recurrent patterns described by LCDs. This is important for a difference equation (1) that exhibits LCDs, since there is not much parallelism within the LCD’s; i.e., the speedup of throughput with current parallelization techniques is bounded by a small constant regardless of how many functional units are used.

In this paper, we present an approach, Harmonic Scheduling, that devises optimal schedules for parallel evaluation of linear difference equations with resource constraints in the context of high level synthesis of digital filters. Harmonic Scheduling generates optimal parallel schedules for evaluating band linear recurrences. The paper is organized as follows. Section 2 briefly reviews the previous work in scheduling for high level synthesis and parallel linear recurrence evaluation. Section 3 lists the assumptions and definitions. Section 4 highlights the idea of using Harmonic Scheduling to exploit recurrences. Section 5 gives an algorithm for generating optimal schedules for 2nd-order difference equations, and illustrates the algorithm with an example. Section 6 summarizes the paper.

2 Previous Work

Related work has been done in two areas: scheduling algorithms for high level synthesis and parallel computing. Scheduling algorithms for high level synthesis have been exploiting three forms of parallelism: concurrency, functional pipelining, and loop pipelining. Concurrency[4, 2] explores instructions that may be executed simultaneously. Functional pipelining techniques overlap instructions along the data paths [4, 14]. Loop pipelining techniques transform a sequential loop into a loop with parallelism across multiple iterations extracted while preserving the program’s semantics. Since scheduling with resource constraints in general is NP-complete, heuristic-based loop pipelining techniques [5, 18, 7] have been developed to compact loops with given resource constraints. Percolation-based loop pipelining techniques [17] first compact a loop into its optimal parallel counterpart and then apply resource constraints on the program version. All of the techniques above preserve the data(true) dependences (LCD is a special case of data dependence), and hence do not try to reduce the critical path by exploiting parallelism beyond data dependences. Techniques that extract parallelism beyond data dependences have just started to receive attention in the high-level synthesis and

---

*This work was supported in part by NSF grants CCR8704367, MIP9009239 and ONR grant N0001486K0215.
digital signal processing communities. A few techniques based on
tree height reduction [6, 12, 10, 16] have been proposed to shorten,
using extra operations, the critical paths consisting of a
sequence of associative operations, e.g., a sequence of additions
or a sequence of multiplications. But these techniques do not
extract parallelism in recurrent loops involving additions and
multiplications beyond LCD's that have dependence distances
of one, and hence can at best achieve a speedup bounded by
constant as number of functional units increases.

In the field of parallel computing, parallel evaluation of linear
recurrences has been studied for quite some time. However, the
underlying model differs from high-level synthesis, since paral-
lel computing typically deals with a number of identical, multi-
function processors, while high-level synthesis deals with a va-

ty of functional units that differ in several attributes (e.g.,
functionality, cost, speed). We briefly review some results from
parallel computing. Kogge and Stone [8] described a technique
called recursive doubling for computing the first-order linear
recurrence system with unlimited resources. Chen and Kuck
developed an algorithm [1] for computing N m-th order band
linear recurrence with p processors that achieves a time bound
of \((Np)/(2^{m^2}+3m) + O(N^2 \log p/m)\). Hays and Kung [8]
established a pair of lower and upper bounds of \(3N/(p+1/2)\) and
\(3N/(p+1/2) + O(\log p)\) time steps respectively with p proces-
sors for parallel evaluation of the Horner expression, which is
equivalent to evaluating the last equation in a first-order band
linear recurrence, i.e., evaluating \(x_2\) only without having to
compute \(x_1, ..., x_{N-1}\). Gajski [3] lowered the time bound of
Chen further to \((2m^2 + 3m)/p + O(N^2 \log p/m)\). For \(p \geq m + 1\)
and \(N > p^2\), for computing the band linear recurrences with p
MIMD processors. Recently, Wang and Nicolau [20] proposed a
novel approach, Harmonic Scheduling, which generates optimal
schedules for linear recurrences that achieve an execution time of
\((2m^2 + 3m)/p + O(N^2 \log m)\), for \(m \geq 1, p \geq (m^2 + 3m^2 / 2m - 1/2m)/(2 + \log m)\).

When \(p < (m^2 + 3m^2 - 1/2m)/(2\log m)\), their schedules also give better
execution time than the previously published fastest schedules.
They proved that this is the strict time lower bound for
\(m = 1, 2, 3\). In addition to the time bounds, their method also
facilitates deriving schedules with the best program-space effi-
ciency (so that a schedule can better fit into caches), given an
execution time.

3 Assumptions and Definitions

In order to highlight the technique for exploiting recurrences,
we make some simplifying assumptions about the target ar-
chitecture and the functional units used. The technique we
describe is still applicable if these assumptions are relaxed to
model more realistic architectures with a range of functional
units. We assume that the target architecture for the DSP con-
sists of a set of functional units (FUs) that can perform a logic
operation, an addition or a multiplication in one time step (i.e.,
unit time). We assume that each functional unit has a dedi-
cated control unit (with a register-file or memory for microin-
structions) that controls the functional unit in each time step.
We also ignore the effect of the number of intermediate regis-
ters (storage units) required to store the results of redundant
operations and we ignore intercommunication delays. Further,
we assume that we are given an initial allocation of functional
units (resources). Our objective is to generate an optimal sched-
ule (i.e., minimal number of time steps) for this allocation that
exploits LCD recurrences, followed by a minimization of the
control overhead for each functional unit (i.e., the size of the
top level microprogram for each functional unit).

We now define the terms frequently used in this paper. A
computation A can be performed using a sequential schedule
\(A_{seq}\) on a design with a single functional unit, or using a par-
allel schedule \(A_{par}\) on a design with \(p\) functional units\(^1\). We
denote the time to run a schedule \(A_{par}\) on a design with \(p\) func-
tional units as \(T_p(A_{par})\) (or \(T_p\) or \(T\) when unambiguous). We
refer to \(T_p\) and \(T\) interchangeably as execution time, time steps
or steps. The time to compute \(A\) sequentially (on a single func-
tional unit) is denoted by \(T_1(A_{seq})\). The speedup of a design
with \(p\) functional units over a design with a single functional unit
for computation \(A\), is denoted \(S_p(A) = T_1(A_{seq})/T_p(A_{par})\), or
simply \(S_p = T_1/T_p\) when unambiguous. The efficiency of this
computation is \(E_p = S_p/p\), which can be interpreted as ac-

tual speedup divided by the maximum possible speedup using \(p\)
functional units. Let \(O_p\) be the number of operations executed
in some computation using \(p\) functional units. We define oper-
ation redundancy to be \(R_p = O_p/O_1(\geq 1)\), where \(O_1 = T_1\).
Finally, we define utilization as \(U_p = O_p/T_p \leq 1\), where \(T_p\)
is the maximum number of operations that \(p\) functional units
can perform in \(T_p\) steps.

The final values of a linear recurrence are the values com-
puted by the definition of the recurrence, i.e., the values as-
gigned to the left-hand side of the statements in the loop body
corresponding to Equation (refdifference). The final operations
of a linear recurrence are operations that compute the final
values in the sequential computation. The redundant operations
are operations that compute auxiliary values introduced by the
parallel schedules in an effort to speed up the computation of
the final values. The redundant values refer to the auxiliary
values computed by the redundant operations.

We introduce some terms for the matrix representation of
linear recurrences illustrated using first-order linear recurrences.
The sequential evaluation of \(N\) first-order linear recurrences can
be expressed as a matrix chain multiplications:

\[
[ x_{N+1} ] = [ x_1 ] \left[ \begin{array}{ccc} a_{21} & 0 & 0 \\ a_{22} & 0 & 0 \\ a_{23} & 1 & 0 \\ \end{array} \right] \left[ \begin{array}{ccc} c_1 & 1 & 0 \\ c_2 & 1 & 0 \\ c_3 & 1 & 0 \\ \end{array} \right] ... \]

for \(1 \leq i \leq N\) and \(1 \leq k \leq N\). The sequential matrix
chain multiplication above can be represented as a dependence tree
as shown in Figure 1. Note that although the matrix formula-
lation of the linear recurrences looks similar to the systolic array
approaches[11], we specifically address parallelization of linear
recurrences beyond LCD's in this work while the previous sys-
tolic approaches do not.

The leaves of the computation tree, \(X_1, X_2, ..., X_m\), are the
matrices, called the operand nodes. The nodes on the top fringe
of the tree, \(X_1, ..., X_m\), \(X_i = X_{i-1}M_i\) for \(2 \leq i \leq k\), are the
results of each recurrence in the system packaged in matrix form,
called the result nodes or final nodes. The matrix multiply only a
sequence of two arithmetic operations, a multiply fol-

dowed by an add, are necessarily done. The column of numbers
above a result node represents the time steps at which the single
FU is allocated to execute the operations in that node. The
number of operations in a sequential evaluation of \(N\) m-th or-
der banded linear recurrences is \(2mN\). The computation above
cannot be significantly parallelized without breaking the de-
pendencies and introducing redundant operations, because each
result node \(X_i, 2 \leq i \leq k\), depends on the previously computed
result node \(X_{i-1}\).

\(^1\)We distinguish between our algorithm and schedule - our
algorithm produces a schedule (for parallel evaluation of the
given linear recurrence) which, when run on a set of functional
units, actually evaluates the linear recurrence in parallel.
The basic objective of Harmonic Scheduling [13, 20] is to simultaneously minimize the number of redundant operations and maximize the utilization of the allocated FU's; we call these tasks the dual requirements of Harmonic Scheduling. We know that the number of final operations in any schedule equals at least the number of operations in the sequential schedule, which cannot be reduced due to the definition of the required outputs. The only mechanism for significantly speeding up the computation is to use multiple FU's to compute redundant values ahead of the final value computation; this shortens the critical path in the computation of some final values, and makes available multiple final values in the fewest possible parallel steps by using previously computed (final and redundant) values.

In order to compute as many final values as possible given a fixed number of FU's, a schedule should do as few redundant operations as possible. However, one cannot reduce the number of redundant operations arbitrarily, say, to zero, since this will sequentialize the computation of final results (thus making a parallel schedule degenerate into a sequential schedule in the limit), and will reduce the speedup to a minimum. Intuitively, the fastest parallel schedules for a fixed number of FU's should and would satisfy the dual requirements. Such a state, in which the dual requirements are satisfied, can be seen as having achieved "harmony" among the conflicting goals, hence the name "Harmonic Scheduling".

A parallel schedule can be divided into a number of periods that have the same organization. Such a period is constructed with schedule-components of different types, based on the number of arithmetic operations in the component. \((m+1)\) types are needed for an optimal schedule for \(m\)th-order linear recurrences. Figure 3 shows the types of schedule-components for first-order linear recurrences. The first type of component (type-1) consists of a final vector-by-matrix multiplication using two operations and a final vector-by-matrix multiplication that uses the redundant matrix-by-matrix product, for example, nodes \(R_3\) with \(X_3\) and \(R_2\) with \(X_6\) make two components of the second type.

Figure 1: A dependence tree for the sequential computation of a 1st-order linear recurrence

Figure 2: A dependence tree for parallel computation of a 1st-order linear recurrence

Figure 3: Two types of components in a parallel schedule for first-order banded linear recurrence

4 Harmonic Scheduling for Exploiting Band Linear Recurrences

The next section illustrates the procedure for constructing the computation tree at the operation level. A detailed example illustrating the procedure for constructing the computation tree at the operation level.
5 Harmonic Scheduling of Second-Order Recursive Filters

In this section, we derive parallel schedules with resource constraints for second-order recursive filter sections using Harmonic Scheduling. In filter design, the sensitivity analysis indicates that a less-sensitive filter structure may be obtained by breaking up the transfer function into lower-order sections and connecting these sections in parallel or in cascade. Although high-order blocks may be attractive in some applications, the second-order section is a good building block to use in parallel or cascade structures [19]. Needless to say, the Harmonic Scheduling method applies to difference equation breaking up the transfer function into lower-order sections and section is obtained from coefficients to zero except for $a_1$ and $b_2$, yielding $8$ recursive second-order all pole section with complex conjugate poles [19]. With little modification, the same schedule (for the simple second-order section) can be applied to general second-order sections.

$$y(n) = x(n) + a_1 y(n-1) + a_2 y(n-2). \quad (2)$$

This is a second-order linear recurrence with constant coefficients, which if parallelized with LCD-preserving techniques would give no further speedup beyond two functional units. Further, $y(n)$ can be expanded into the following form in terms of matrix multiplications

$$[y_n \ y_{n-1}] = [y_1 \ y_0] \begin{bmatrix} b_1 & 1 & 0 & 0 \\ b_2 & 0 & 0 & 1 \\ x_1 & 0 & 1 & 0 \\ x_2 & 0 & 0 & 1 \\ \vdots & \ddots & \ddots & \ddots \\ x_n & 0 & 0 & 1 \end{bmatrix}.$$  

where $y_1, x_1$ stand for $y(i), x(i)$ respectively. For any redundant matrix multiplication in the chain, the results in the first two rows of the product can be precomputed since they remain the same through all periods of a schedule. Based on the number of arithmetic operations in a matrix multiplication above, three types of matrix multiplication will be involved in our parallel schedules, the first type using 4 arithmetic operations, the second type using 6 and the third type using 8 respectively. Next, we go through the Harmonic Scheduling phases for difference equation (2).

In the first phase, we solve some inequalities for the numbers of all types of components to be used in our parallel schedule. Harmonic Scheduling establishes the following system of inequalities and asks for integer solutions:

$$\begin{align*}
4w_0 + 6w_1 + 8w_2 & \leq \frac{T_p}{N^p}, \\
2w_0 + w_1 & \leq \frac{4w_0 + 6w_1 + 8w_2}{p}, \\
w_0 & \geq w_1, \quad w_1 \geq 0, \quad w_2 \geq 0.
\end{align*} \quad (4)$$

where $w_0, w_1, w_2$ are numbers of the first, second, and third type components respectively, and $p$ is the number of functional units. The first equation relates the pattern of a period of a parallel schedule to its execution time. It essentially says that, in order to achieve an execution time $T_p$ for $p$ functional units, a parallel schedule must produce $w_0 + 2w_1 + w_2$ final results (i.e., sample outputs) using every $4w_0 + 10w_1 + 8w_2$ operations. The other inequalities serve as necessary conditions for the feasibility of solutions. The first inequality says that a solution would give an infeasible schedule (i.e., the schedule cannot achieve the execution time $T_p$), if the length of the critical path in a period (described by the left-hand side) were greater than the number of steps required by the number of operations. The other three simple inequalities constrain the solution elements to be non-negative. Specifically, a parallel schedule must have some second type components.

We use execution time $T_p = (8p - 4)/(p(p+1))$ in inequalities (4), which gives a speedup of $(p + 1)/(2 - 1/p) > p/2$ over sequential schedules - the best time so far. Although it is not the strict time lower bound, the strict time lower bound can be achieved by lowering $T_p$ in inequalities (4). A set of integer solutions to inequalities (4) parameterized for $p$ is given below:

$$w_0 = 2t_0, \quad w_1 = 2t_1, \quad w_2 = (p - 3)t_0 \quad (5)$$

Not all integer solutions given by those expressions can be made into feasible schedules. But for any $p > 2$, there exists $t_0$ such that a feasible schedule can be built out of the solutions. A detailed discussion is given in [20]. The following algorithm outlines the second to the fourth phases of Harmonic Scheduling in constructing a feasible schedule using the solutions above.

Input: $p$ functional units for constructing a schedule for parallel evaluation of the second-order difference equation (2), with $w_0$ components of the first type, $w_1$ of the second type and $w_2$ of the third type.

Output: a parallel schedule.

procedure construct.schedule.for.lr(p, w0, w1, w2)
1. call procedure build.computation.tree(components of all types);
2. call procedure FU.slot.allocation(tree, slot.sets);
3. transform the computation tree with allocated functional unit slots into a schedule;
end (construct.schedule.for.lr)

Figure 4: The computation tree in matrix multiplications for $p = 4$. To illustrate the second through the fourth phases, we assume $p = 4$ and choose $t_0 = 1$, thus having $w_0 = 2, w_1 = 2$, and $w_2 = 1$. In the second phase, we construct a computation (dependence) tree for a period of the parallel schedule with the given components obtained from the first phase, using the following procedure. The computation tree for $p = 4$ is shown in Figure 4.

procedure build.computation.tree(w0, w1, w2)
built $w_1$ redundant trees by associating a second type of component with each first type of component; while (there is a third type component)
for $i = 1$ to $w_1$ do
  give a third type component to the $i$th redundant end for
end while
end (build.computation.tree)

In the third phase, we allocate functional unit slots to operations in the computation tree. The functional unit slots are
allocated to the final operations of the leading period and to the redundant operations of the succeeding period. The functional unit slots are allocated on a "most-urgent-first" policy, meaning that a functional unit slot is always assigned to compute the operation which if not computed by that slot would ruin the full functional unit utilization. Note that there are multiple ways of allocating the functional unit slots. The slot allocation procedure is similar to the one in [20].

Figure 5 shows the computation tree for the first two periods expanded down into the arithmetic operation level with functional unit slot allocation. Each box represents an arithmetic operation. The number inside a box following the operator is the functional unit slot assignment into the final parallel schedule. The operations below outputs $y_2$ through $y_6$ are in the leading period and those below outputs $y_7$ through $y_{22}$ in the succeeding period (the length of the period $r = 5$). The symbols $t_1$ through $t_{23}$ correspond to memory holding intermediate results. $b_1, b_2$ are the coefficients given in difference equation (2). $m_1 = b_1^2 + b_2, m_2 = b_1 b_2, m_3 = m_2 b_1 + b_2, m_4 = m_3 b_1 + b_2$ are precomputed coefficients, since these same coefficients are used throughout all the periods.

In the fourth phase, we write the computation tree with functional unit slot assignments into the final parallel schedule. We compose the final computation of the leading period and the redundant computations of the succeeding period. The functional unit slots are allocated on a "most-urgent-first" policy, meaning that a functional unit slot is always assigned to compute the operation which if not computed by that slot would ruin the full functional unit utilization. Note that there are multiple ways of allocating the functional unit slots. The slot allocation procedure is similar to the one in [20].
Published fastest schedule for 2nd-order difference equation.

Filters described by second-order difference equations with resource constraints. The parallel schedules generated with Harmonic Scheduling achieve an execution time of

\[ T_p = \frac{8p - 4}{p + 1} N, p > 2, \]

yielding a speedup of \((p+1)/(2 - 1/p) \geq p/2\) over sequential schedules, which also have the minimum schedule size (thus the least schedule size) with respect to the execution time. This is the fastest parallel schedule up to date for the second-order difference equation (2). It also has the smallest schedule size in comparison with the previously published parallel schedules. Currently, we are working on relaxing some of the simplifying assumptions made in this paper to make the target architecture and the design model more realistic. In particular, we are working on adapting this technique to find optimal schedules for mth order difference equations with resource constraints under practical conditions such as using multi-cycle operations (e.g., multiplication), and functional units of different types, operations and delays. Future work needs to address the effect of additional storage introduced by temporaries needed for redundant operations, as well as the impact of connectivity and bindings on Harmonic Scheduling.

References


<table>
<thead>
<tr>
<th>number of FU's</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>previous fastest schedule's size</td>
<td>54</td>
<td>104</td>
<td>170</td>
<td>252</td>
<td>350</td>
<td>464</td>
<td>594</td>
<td>740</td>
</tr>
<tr>
<td>speedup improved by %</td>
<td>20%</td>
<td>16%</td>
<td>13%</td>
<td>11%</td>
<td>9.9%</td>
<td>8.7%</td>
<td>7.8%</td>
<td>7.1%</td>
</tr>
<tr>
<td>our schedule size</td>
<td>36</td>
<td>36</td>
<td>120</td>
<td>180</td>
<td>252</td>
<td>336</td>
<td>432</td>
<td>540</td>
</tr>
<tr>
<td>schedule size saved by %</td>
<td>33%</td>
<td>65%</td>
<td>29%</td>
<td>29%</td>
<td>28%</td>
<td>28%</td>
<td>27%</td>
<td>27%</td>
</tr>
</tbody>
</table>

Table 1: The improvement in speedup and the reduction in schedule size by our schedule on the previously published fastest schedule for 2nd-order difference equation.