

Abstract—Time-critical sections of multi-dimensional applications, such as image processing and computational fluid dynamics are, in general, iterative or recursive. Most of these applications require each iteration to be executed under a specific time constraint associated with the data input rate. The design of circuits dedicated to perform such repetitive tasks is highly dependent on optimization techniques to achieve the desired execution time. The existence of branch instructions within the recursive code (loop) may degrade the performance of the optimized code. Branch predication techniques utilize predicate registers to control the validity of speculatively computed results and prevent those branch hazards. These registers are significant obstacles in the performance gain achievable by the overlap of successive iterations of nested loops. This paper presents a process of designing and dimensioning those registers while optimizing the computational time of the loop.

I. INTRODUCTION

Multi-dimensional applications, such as image processing and seismic analysis, usually require the high computer performance obtained from the implementation of Application Specific Integrated Circuits (ASICs). The critical sections of such applications consist of nested loops with the possibility of embedded conditional branch instructions. Current commercial systems use branch predication techniques, which can also be applied in the design of ASIC systems. Those techniques utilize predicate registers to control the validity of computed results. The optimized design and allocation of such registers becomes then a significant factor in the performance of the system. This paper shows how the chained multi-dimensional retiming technique can be used to optimize the ASIC execution schedule while specifying the characteristics of the predicate registers, reducing the overall execution time of the nested loops found in the critical sections of those applications.

Most of the previous results on scheduling loops are solely based on one-dimensional uniform problems, which do not consider the effects of conditional branches [1,5,16,19]. This study focuses on the parallelism inherent to multi-dimensional applications involving branch decision, ignored by the one-dimensional methods [2,9]. Some recent research has been conducted in the scheduling of multi-dimensional applications, such as the affine-by-statement technique [3] and the index shift method [13]. However, these methods are targeted to uniform loops with no conditional instructions. Other methods focus on multi-processor scheduling and are not applicable to the problems covered in this paper [6,7,8,11,14,15].

In previous studies, the authors introduced the concept of multi-dimensional retiming, which allowed the restructure of the loop body represented by a general form of multi-dimensional data flow graph (MDFG), while preserving data dependencies [16,18]. In an MDFG, the nodes represent the loop body operations and the edges, the data dependencies between operations. By associating each execution instance of the loop with an integral index in a Cartesian space, it is possible to compute a multi-dimensional distance between the production and consumption of each data value. The application of a multi-dimensional retiming to an MDFG permit the iterations of the original loop body to be naturally overlapped, making the existent parallelism explicit.

Branch predication techniques use if-conversion methods to transform the conditional code in straight-line code [20]. The new code uses Boolean guards to implement the conditional execution of individual operations. Such a transformation reduces existing control dependencies to more common data dependencies. However, it requires the use of special registers, commonly named predicate registers to store the Boolean values. This paper develops a new method for scheduling cyclic MDFGs with an infinite number of non-pipelined processing resources and predicate registers available. The schedule length is associated with the number of control steps, i.e., the clock cycles of the circuit design, required to execute all operations in the loop body. Considering the assumption that each functional unit can execute in one time unit, such schedule can be reduced to one time unit.

Such a technique is implemented through a polynomial time scheduling algorithm, resulting from a modification to the chained retiming algorithm presented in [17]. In this algorithm, nodes are retimed in such a way that all operations can be executed in parallel. A significant problem in this mechanism is the handling of the required predicate register.

Next section presents an overview of multi-dimensional retiming and the branch predication technique, followed by a discussion on the basic fundamentals of the multi-dimensional scheduling technique. The final algorithm and register design are developed next. An example of application
of the new method demonstrates its efficiency. A summary finalizes the paper.

II. BACKGROUND

Multi-dimensional retiming

A multi-dimensional data flow graph $G$ consists of a tuple $(V,E,d,t)$, where $V$ is the set of computation nodes, $E$ represents the set of dependence edges, $d$ is a function representing the multi-dimensional delays between two nodes, and $t$ is a function representing the computation time of each node. An iteration is equivalent to the execution of each node in $V$ exactly once. An iteration is associated with a static schedule that is repeatedly executed for the loop. The earliest starting time for the execution of node $u$, $ES(u)$, is the first scheduling control step following the end of the execution of all nodes predecessors of $u$ by a zero-delay edge. This can be represented as: $ES(u) = \max \{1, ES(v_i) + t(v_i)\}$ for all $v_i$ preceding $u$ by an edge $e_i$ such that $d(e_i) = (0,0,0,...,0)$. A schedule vector $s$ determines the sequence of the execution of the iterations.

An MD retiming $r$ redistributes MD delays in an MDFG $G$, creating a new MDFG $G_r = (V,E,d,t)$. A retiming vector $r(u)$ applied to a node $u \in V$ represents the offset between the original iteration containing $u$, and the one after retiming. Such vector represents MD delays pushed into the edges $u \rightarrow v$, and subtracted from the edges $w \rightarrow u$, where $u, v, w \in V$. The chained MD retiming technique [13] is one of the possible methods able to compute a legal MD retiming for some MDFG. This technique is characterized by important properties:

1. A legal MD retiming $r$ of a node in an MDFG $G$, with all its incoming edges having non-zero delays, is any vector orthogonal to a schedule vector $s$ that realizes $G$.
2. If $r$ is a MD retiming function orthogonal to a schedule vector $s$ that realizes $G= (V,E,d,t)$, and $u \in V$, then $(k \cdot r)(u)$ is also a legal MD retiming.
3. The chained MD retiming algorithm transforms $G$ to $G_r$, such that $G_r$ is realizable and fully parallel.

The algorithm responsible to compute the chained retiming is able to produce a fully parallel MDFG.

Branch Predication

Predicated execution, also called guarded execution, provides the means to prevent the performance loss generated by the speculative execution of instructions in processors with multiple functional units, following a conditional branch command. Under this technique, an instruction is executed based upon a Boolean value called predicate. If the predicate is false, the instruction results are nullified, not changing the processor state. The predicated execution combined with if-conversion allows the removal of branch instructions from the instruction stream and consequent elimination of control dependencies, which are automatically transformed in data dependencies. As a simple example of this technique let us examine the code below:

```plaintext
if A > 5 then B = 2
else B = 3
endif
```

The use of the if conversion technique would need a Boolean guard $p$, which receives the result of the test of the condition $A > 5$. In the predicated execution, $p$ is associated to each of the two instructions assigning new values to $B$. After $p$ is calculated true or false, then one of the assignment instructions is discarded while the other is completed. The code below shows the predicated format for this example:

```plaintext
p = A > 5
B = 2 | p /* B=2 if p == true */
B = 3 | !p /* B=3 if p == false */
```

In such a situation, the two instructions assigning new values to $B$ could be fetched and partially executed in parallel with the decision process. However, when such a condition is found within a loop, the overlap of iterations may not be possible due to the need to reuse the predicate register $p$.

III. ALGORITHM

This section describes how to combine the chained multi-dimensional retiming and the concept of predicated execution in order to allow full fine-grain parallelism of the instructions comprising the body of a nested loop.

An example of a two-dimensional problem consists of an edge detection algorithm based in the computation of the Laplace edge enhancement [12]. This problem can be represented by an acyclic graph as shown in figure 1, where the node $F$ is equivalent to an if statement. In a processor with one arithmetic logic unit, one can easily conclude that the minimum schedule length, required to execute the edge detection process, would be equivalent to 7 control steps. Assuming that an infinite number of computational units are available and are able to execute one operation in one control step, the schedule length is reduced to 5 control steps by the application of a traditional list scheduling method [4], as shown in figure 2(a). Figure 2(b) shows a more compact schedule in which node $D$ has been rescheduled to the first control step. This new schedule can be associated with the retimed graph presented in figure 3(a), where the two-dimensional delay $(0,1)$ was pushed through nodes $A$ and $B$ of the original MDFG (Note that $(0,1)$ is arbitrarily chosen here). The vector $(1,0)$ is also possible, as are many others). Intuitively, the retimed node no longer precedes the operation $D$ within the same iteration.
The overlap of different iterations of the loop requires the reutilization of the predicate registers. In order to solve this problem, such registers must be implemented as shift-registers able to keep track of the changes in the Boolean guards. The chained multi-dimensional retiming algorithm modifies the loop in such a way that all instructions can be run in parallel. The use of overlapped iterations is the basis for this parallelism. In order to avoid the reutilization of the predicate registers, these registers must be dimensioned in such a way to store their Boolean values until they are required. The new version of the algorithm allows the designer to obtain the dimensions of the predicate registers in such a way that overlapped iterations do not interfere with each other. The algorithm combining the multi-dimensional retiming techniques and applicable to the if-converted code can be summarized in the pseudo-code shown next:

**Algorithm MDRPRED (G = (V,E,d,t) )**

Choose \( s = (s_1,s_2,...,s_n) \) such that \( s \cdot d(e) > 0 \) for any \( e \in E \).
Choose \( r \) such that \( r \perp s \)
\[
\forall \ u \in V , \ LV(u) \leftarrow 0
\]
\[
LV_{\text{max}} \leftarrow 0
\]
Queue\( V \leftarrow \varnothing \)

/* remove original edges with non-zero delays */
\[
E \leftarrow E - \{ e_i \mid d(e_i) \neq (0,\ldots,0) \}
\]

/* queue schedulable nodes */
Queue\( V \leftarrow \) Queue\( V \cup \{ u \in V, \text{s.t. indegree}(u) = 0 \} \)
while Queue\( V \neq \varnothing \)
  get\( u, \) Queue\( V \)
schedule\( (u) \)
  /* propagate the level values to successor nodes of \( u \) */
  \[
  \forall \ v \text{ such that } u \leftarrow v
  \]
  indegree\( (v) \leftarrow \) indegree\( (v) - 1
  \]
  \[
  LV(v) \leftarrow \max\{LV(v), LV(u)+1\}
  \]
  \[
  LV_{\text{max}} \leftarrow \max\{LV(v), LV_{\text{max}}\}
  \]
  /* check for new schedulable nodes */
  if indegree\( (v) = 0 \)
    Queue\( V \leftarrow \) Queue\( V \cup \{ v \} \)
  endif
endwhile

/* compute the multi-dimensional retiming */
\[
\forall \ u \in V , \ r(u) = (LV_{\text{max}} - LV(u))
\]
/* compute the size of the shift registers required for the predicate registers between decision \( u_p \) and nodes \( v_i \) */
\[
\forall \ p \in P , \ s(p) = \max(LV(v_i[p]) - LV(u_p)) + 1
\]

End MDRPRED

Let us examine how the algorithm works. In the first step, the nested loop is translated in a MDFG, which is the input to the algorithm. Since there is no restrictions on the schedule vector \( s \), it can easily selected under the condition that its product by any graph dependency be positive. After choosing \( s \), a basic retiming vector perpendicular to \( s \) is computed. Non-zero delay edges are removed making the graph an acyclic one.
IV. SUMMARY

The design of ASICs can provide the necessary performance to critical nested loops of multi-dimensional applications. The performance degradation experienced through the execution of conditional branches can be eliminated by using branch predication techniques. This paper showed an algorithm that efficiently combine multidimensional retiming and branch predication in the scheduling of the optimized loops and in the design of the predicate registers required by the branch predication techniques.

REFERENCES


