A New Parallel Reconfigurable Computing Architecture and Hidden Markov Model Application

Yung-Chuan Jiang, Anand Paul, Jhing-Fa Wang
Department of Electrical Engineering,
National Cheng Kung University
ycjiang@mail.chna.edu.tw

Abstract—**Parallel processing techniques are increasingly found in reconfigurable computing, especially in digital signal processing (DSP) applications. In this paper, we design a parallel reconfigurable computing (PRC) architecture which consists of multiple dynamically reconfigurable computing (DRC) units. The hidden Markov model (HMM) algorithm is mapped onto the PRC architecture. First, we construct a directed acyclic graph (DAG) to represent the HMM algorithms. A novel parallel partition approach is then proposed to map the HMM DAG onto the multiple DRC units in a PRC system. This partitioning algorithm is capable of design optimization of parallel processing reconfigurable systems for a given number of processing elements in different HHM states.**

Index Terms—**FPGA, parallel processors, reconfigurable processing, HMM, partitioning algorithm.**

I. INTRODUCTION

Reconfigurable computing (RC) is a promising alternative to application-specific integrated circuits (ASIC) and general-purpose processor systems, providing software processor flexibility, hardware coprocessor efficiency, high throughput and enhanced speed. Field programmable gate arrays (FPGAs) are the most common devices used for RC, but "on-the-fly" dynamic reconfiguration has emerged as an attractive technique for minimizing reconfiguration time. In particular, dynamically reconfigurable computing (DRC) [1] [2] is receiving growing interest because its utilization of computational logic units can be dramatically improved by logical time-sharing. On-chip resources can be reused, cutting hardware costs and improving performance. Therefore, this paper utilizes dynamically reconfigurable computing architecture to implement the HMM algorithm. Massive parallelism, a special focus of this present paper, is considered a strong contender improved DSP chip designs. Moreover, the inherent parallelism in the HMM algorithm motivates us to propose a parallel processing architecture for its implementation.

The parallel processing technique is generally used in a multiple instruction, multiple data (MIMD) architecture [5] to help optimize system performance. MIMD organization with multiple processors and I/O processors access one or more memory modules via a bus. Traditional non-dynamic RC architecture may be considered as multiple processor elements (PEs) sharing a single physical memory. A control processor executes the RC units, accessing memory through a shared memory bus. However, the MIMD architecture has the following drawback. Because all memory references pass through the common bus, there is a data bottleneck at this bus.

To reduce bus access numbers, equipping each processor with its own local memory as a dynamically reconfigurable computing machine with several configurations is desirable. The following discussion will assume the availability of such hardware. Prior studies have shown that architecture consisting of several parallel non-dynamic RC units implemented as parallel FPGA can improve system performance [7]-[8]. In consequence, this present paper extends the above ideas by presenting a parallel reconfigurable computing (PRC) machine which combines several parallel FPGA arranged as parallel dynamically reconfigurable computing machines. In the PRC architecture, the proposed DRC-based processors are directly and easily usable in a symmetric multi-processor organization [5]. Figure 1(a) shows the proposed PRC architec-

ture. A set of individual dynamically reconfigurable parallel processing units (DRPPU), each implemented on its own FPGA, are connected in parallel by a common bus. Each DRPPU has its own local memory. Each DRPPU is composed of an array of configurable logic blocks (CLB), as seen in Figure 1(b). Such architecture maximizes potential system performance for high computation and data intensive applications [9]-[11] such as MPEG-4, H.264 video encoding [7], and HMM recognition [12].

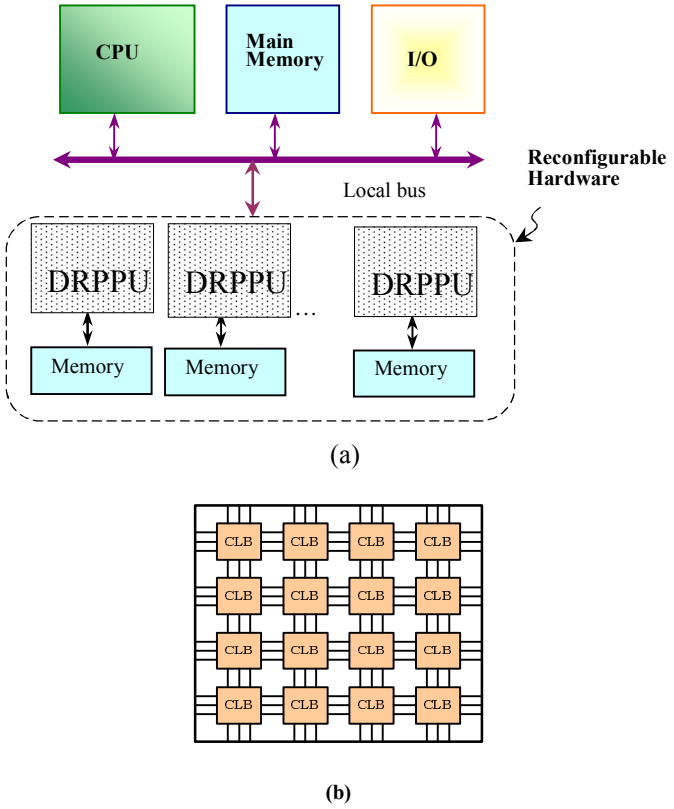


Figure 1. (a) Parallel reconfigurable computing architecture: each DRPPU has a local memory. (b) DRPPU architecture.

In a PRC system, the DRPPU is a dynamically reconfigurable computing processor similar to the fine grained architecture of DRFPGA [6]. DRC units have been used to realize large systems using multiple configurations [3]-[4]. Importantly for PRC implementation, the HMM algorithm can be partitioned into multiple stages and stored in configuration memory planes. In general, DRC units hold only one active configuration in any time frame. Each configuration is called a cycle. All combinational logic is evaluated and flip-flop values are updated in one cycle. [6]. Each CLB has microregisters that store intermediate values generated from combinational logic for later cycle use and also hold flip-flop values for next cycle use. A cycle begins by saving all previous cycle CLB results in microregisters and then a new configuration is loaded into the active configuration of the local memory.

The remainder of this paper is organized as follows. Section II introduces the HMM algorithm and uses data flow graphs (DFG) to represent it. Section III presents the problem formulation. Section IV proposes parallel partitioning for mapping HMM DAG onto a PRC system. Experimental results are presented in Section V. Finally, conclusions are given in Section VI.

II. Hidden Markov Model and Corresponding Data Flow Graph

A. Hidden Markov Model

The Hidden Markov model is a class of statistical models useful for analyzing a discrete time series of observations such as a stream of acoustic elements extracted from a speech signal. An HMM is characterized by the state transition probability distribution, observation probability distribution and initial state probability distribution. For an HMM consisting of N states $S_1, S_2, \ldots, S_N$, we denote the state transition probability distribution as $\boldsymbol{A} = \{ a_{ij} \}$, where a_{ij} is the state transition probability from state S_i to state S_j. The observation probability distribution is represented by $\boldsymbol{B} = \{b_j(\boldsymbol{o}_t)\}$, where $b_j(\boldsymbol{o}_t)$ is the probability of having an observation vector $\boldsymbol{o}(t)$ at time-step t being in the state j. The initial state probability distribution is represented by $\boldsymbol{\pi} = \{\pi_j\}$, where π_j is the initial probability in the j-th state. The above three distributions can be indicated compactly by $\lambda = (\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi})$. For an observation sequence $\boldsymbol{O} = [\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_T]$, the recognition result relies on HMM probability evaluation, i.e. calculating the observation sequence probability $P(\boldsymbol{O}|\lambda)$. In our VLSI design, we focus on the recognition capability of the HMM. The HMM parameter λ is assumed to be computed in advance and stored in the memory unit. The HMM algorithm in this paper performs the HMM probability evaluation process.

While the conventional HMM evaluation process computes $P(\boldsymbol{O}|\lambda)$ using the forward-backward algo-

rithm, but this paper does so by the alternative Viterbi algorithm method. The Viterbi algorithm has been widely researched and efficient implementations in speech recognition have been proposed in [13]. For left-to-right HMMs, probability evaluation using the Viterbi algorithm can be described as in equation 3. The state decoding problem is solved by equation 2. We define the *time-step probability* = $\delta_t(j)$, which computes the probability of being in state j in time-step t.

$$\delta_1(j) = \pi_j \cdot b_j(o_1) \qquad \text{for } t = 1,\ 1 \le j \le N. \tag{1}$$

$$\delta_t(j) = \max_{0 \le i \le N}[\delta_{t-1}(i) \cdot a_{ij}] \cdot b_j(o_t)$$

$$\text{for } 2 \le t \le T,\ 1 \le j \le N. \tag{2}$$

$$P(o|\lambda) = \max_{1 \le i \le N}[\delta_T(i)] \qquad \text{for } t = T \tag{3}$$

By the above equations, the Viterbi algorithm requires a large number of multiplications to extract the state sequence. From the standpoint of hardware design, minimizing the number of multipliers optimizes the architecture. Because a multiplier occupies a large logical block (a large number of CLB), the area cost of the hardware is increased by using many multipliers. For example, it can be seen in Table I that the CLB number of a multiplier is greater than that of an adder. Therefore, we perform the Viterbi algorithm in the logarithm domain so that multiplication operations can be replaced by addition operations.

The Log-Viterbi algorithm is described as follows:

$$\delta_1(j) = \pi_j + b_j(o_1) \qquad \text{for } t = 1,\ 1 \le j \le N. \tag{4}$$

$$\delta_t(j) = \max_{0 \le i \le N}[\delta_{t-1}(i) + a_{ij}] + b_j(o_t)$$

$$\text{for } 2 \le t \le T,\ 1 \le j \le N. \tag{5}$$

Denote D as the feature order and represent $\boldsymbol{o}_t = [\boldsymbol{o}_{t1}, \boldsymbol{o}_{t2}, \ldots, \boldsymbol{o}_{tD}]$ as the observation vector received in time-step t. In continuous mixture density HMM, an observation probability $b_j(\boldsymbol{o}_t)$ for the observation vector $\boldsymbol{o}_t$ in the state j can be represented as:

$$b_j(o_t) = \frac{1}{\sqrt{(2\pi)^D \cdot \prod_{k=1}^{D} \sigma_{jk}}} \cdot \exp\left(-\tfrac{1}{2} \sum_{k=1}^{D} \frac{(o_{tk} - \mu_{jk})^2}{\sigma_{jk}^{\ 2}} \right) \tag{6}$$

where μ_{jk} and σ_{jk} are the mean vectors and diagonal covariance matrices, respectively, for the state index j and the dimension index k. Equation 6 is transformed into the logarithm domain as follows:

$$log\ b_j(o_t) = \log\left(\frac{1}{(2\pi)^D \prod_{k=1}^{D} \sigma_{jk}} \right) + \sum_{k=1}^{D} \frac{(o_{tk} - \mu_{jk})^2}{-2\sigma_{jk}^{\ 2}}$$

$$= C_j + \sum_{k=1}^{D} \left(\tilde{\sigma}_{jk} (o_{tk} - \mu_{jk}) \right)^2 \tag{7}$$

where $C_j = \log\left(\frac{1}{(2\pi)^D \prod_{k=1}^{D} \sigma_{jk}} \right)$ and $\tilde{\sigma}_{jk} = -\frac{1}{2\sigma_{jk}^{\ 2}}$

B. Operation Weight in HMM DFG

In this subsection, the operations involved in HMM probability evaluation will be organized by a data flow graph (DFG). The DFG, $G = (V, E, W)$, consists of $|V|$ nodes and $|E|$ edges, where each node represents an operation and each edge $e \in E$ represents a dependence between nodes. For each node $v \in V$, there exists a weight $w \in W$. The DFG formulation will be discussed in greater detail in Section III.The HMM algorithm operation set which includes addition, subtraction, comparison, and multiplication. In a DRC system, these operations are implemented by configurable logic blocks (CLBs) [6]. An operation may need several CLBs. Take the observation probability generation for example. It has a set of operations that can be represented by DFG, where a node corresponds to an operation and an edge corresponds to the operation's relation. DRC performance of 4-bit multiplication, subtraction and addition are implemented respectively by 14 CLBs, 7 CLBs and 3 CLBs [4]. Since the weight w is the number of CLBs in a node, the respective weights are w(mult) = 14, w(sub) = 7 and w(add) = 3. Detailed discussion is given in Table I of Section V.

III. Problem Formulation

A. Motivation

Minimizing the hardware execution time to perform the HMM algorithm in a PRC system is a primary goal in this paper. To quantify the execution time, every mapped DRPPU process in the HMM DAG is assumed to require a constant and equal unit cycle time, which includes both the reconfiguration and execution times for that DRPPU process. Each DRPPU is implemented in a single FPGA. All the FPGA's have equal size and equal performance capabilities. If we treat the CLB utilization of each DRPPU as equal, then the total number of CLBs in a DRPPU is equal. The task of a DRPPU is designed to be completed in a single cycle, after which a new configuration is invoked. For DRPPU run in parallel (Section IV below), we define the total execution time as $T_{exe} = k \times$ (DRPPU *reconfiguration time*) + $k \times$ (DRPPU e*xecution time*), where k is the number of configurations, i.e. non-parallel DRPPU used in the graph. Recall that a configuration executes the DRPPU process in one cycle, which we will use in the following as a standard cycle. Hence, an application's total execution time is equal to k cycles.

The PRC hardware architecture will be considered in this study. Partitioning methods for the architecture will be considered. For PRC architecture, consideration of parallel processing in temporal partitioning improves execution time. In this context, the challenge is to exploit the parallelism inherent in HMM. Importantly, operation duplication in a given application is usually allowed in a parallel partitioning solution. Hence, a partitioning technique is developed with this parallel processing consideration in mind. Where a layer is defined as a counter for a parallel array of DRPPU at one cycle time, the *depth* of a solution is defined as the total number layers in the solution. Optimizing a PRC solution finding a minimum depth duplication-permitted solution. Thus, a HMM DAG is partitioned into sub-graphs (each representing a DRPPU under one configuration) with a minimum depth solution. Each sub-graph consists of a set of clustered sub-graphs. Each sub-graph is selected by the greedy method, one at a time. This method will be shown to find the minimum depth partitioning of the HMM algorithm.

B. Terminologies and Problem Formulation

The different states of an HHM can be represented by a directed acyclic graph (DAG), $G = (V, E, W)$, where V is a set of n nodes and E is a set of edges. A *primary input* (*PI*) node, which has no in-coming edge, represents an input signal. A *primary output* (*PO*) node, which has no out-going edge, represents an output signal. Except for PI and PO nodes, each node in V represents the implementation of a functional operation such as addition or subtraction. Figure 2(a) gives an example of a DAG representing a HMM application, in which nodes a, b, c, d, e, f, g and h are PI nodes and nodes o, w and t are PO nodes. A directed edge $e_{ij} = <v_i, v_j>$, $e_{ij} \in E$ exists if the function input represented by v_j depends on the function output represented by v_i. For each node v_i in a node set V, $v_i \in V$, there exists a weight $w_i \in W$ that represents DRPPU area of functional operation implementation, v_i. Notice that the weight for every PI and PO node is zero.

Although DAGs have been used in many prior studies of RC systems, most of them dealt with serial processing. This presented study uses DAGs to deal with parallel processing issues. To do so, this study introduces the concept of a *block,* which we use to designate subsets (subgraphs) of the DAG that can be performed independently and in parallel with each other. A block B is a set of nodes V_B which comprise that block. Nodes within V_B can be any of the nodes within a DAG except the PI and PO nodes. In the following diagrams, nodes are designated by single circles and blocks are designated by circles enclosed in dotted lines, as seen in Fig. 2(a). We define the *area* of any block B as the sum of the weights of the nodes of that block. Since the weight of a node equals the number of CLB in a node, then the area of a block also equals the number of CLB in the block, also called the DRPPU logic capacity, A_{DRPPU}, of any B. Any B is considered *feasible* if the area of B ($A_B = \sum_{v \in B} w_b$) is less than or equal to the DRPPU logic capacity, A_{DRPPU}, i.e. $A_B \le A_{DRPPU}$. In Figure 2(a), for example, the block ***F*** including nodes z, l, n, and s is feasible if $A_{DRPPU} = 23$.

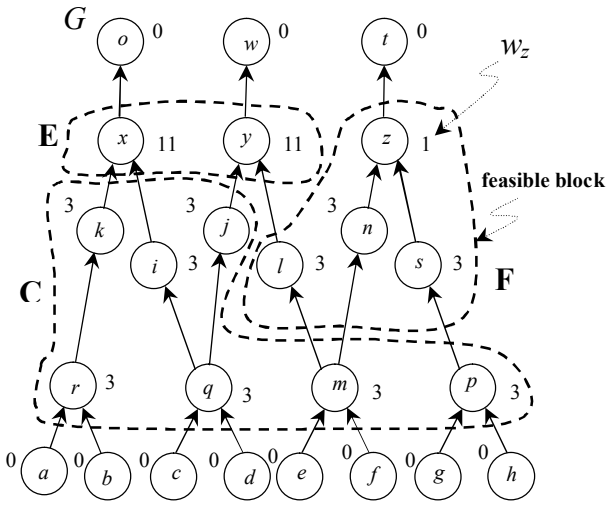
To ensure proper execution sequence, each node

must be scheduled in a block no later than all its output nodes. This is a temporal constraint. Constraints which determine the temporal ordering of the nodes in the DAG are called *precedence constraints* [4]. Even when a graph is acyclic, blocks may exist in a cyclic-relation and cannot satisfy precedence constraints after partitioning. A set of feasible blocks in which precedence constraints are satisfied is called a *feasible partitioning solution*.

In PRC parallel processing reconfigurable architecture, a given DAG is partitioned into feasible blocks such that each block can be implemented in a single DRPPU. The processing of blocks at the same time is allowed up to a maximum for the hardware; in our current example, each DRPPU is implemented by a single FPGA, so the maximum number of blocks that can be processing at the same time equals the number of FPGA. The FPGA (and therefore the blocks) that can be executed in one cycle we designate as a "layer."

Therefore, the PRC partitioning problem for depth optimization can be formulated as a graph-based problem as:

> Given an HMM DAG and the allowing parallel concurrent number of DRPPUs and with respect for the maximum number of parallel concurrent number of DRPPUs, find a feasible partitioning solution with a minimum depth number.



(a)

Blocks (layer)

C (1)

E, F (2)

(b)

Fig. 2. DAG of a program or application.

IV. Proposed Parallel Partitioning for Mapping HMM DAG onto PRC

This section discusses using a single root floor cone to partition a graph so as to have minimum depth. After this, the node duplication effect (an important factor in depth determination) is discussed. The greedy method will be used to find the partitioning solution for a DAG, $G = (V, E)$. Particularly, we consider iteration using the greedy method to select feasible floor cones leads to find the minimum-depth partitioning solution.

A. Floor Cone Properties and Node Duplication

A feasible block $B = (V, E)$ is called a *feasible cone* if there exists a node $v \in V$ such that for every node $u \in B$ there is a directed path from u to v in B. The node v is called the *root* of the floor cone. If all the cone's fan-in nodes are PI nodes, the feasible cone is called a *floor cone*. Let $C_u = (V_u, E_u)$ and $C_v = (V_v, E_v)$ be two cones. C_u and C_v are said to *overlap* if $V_u \cap V_v \neq \phi$.

Let C_v be a feasible floor cone tipped at v. If a feasible floor cone tipped at every successor of v is not feasible, C_v is called a *maximum floor cone* (MFC). For example, in Figure 4(a) if $A_{DRPPU} = 23$, the floor cone including the nodes $\{x, k, i, r, q\}$ is a MFC. On the other hand, the floor cone including the nodes $\{k, r\}$ is a floor cone but not a MFC because x is a successor of k and the floor cone tipped at x is a feasible floor cone. Clearly, the MFC_v of node v is a maximal floor cone. Moreover, an MFC has the following important properties.

Lemma 1: If $w \in MFC_v$, then floor cone $C_w \subseteq MFC_v$.

Proof: For any node $u \in C_w$, if a path does not exist from u to root w, it contradicts the assumption that

$u \in C_w$. Therefore, for every node $u \in C_v$, there is a directed path from u to v in C_v. This implies that for every node $w \in \mathrm{MFC}_v$ there is a directed path from w to v in MFC_v. Therefore, $C_w \subseteq \mathrm{MFC}_v$. ■

Lemma 2: If $V_o = \mathrm{MFC}_v \cap \mathrm{MFC}_w$ and $u \in V_o$, then floor cone $C_u \subseteq V_o$.

Proof: If MFC_v and MFC_w exist and overlap V_o, then $u \in V_o$. Therefore, two paths must exist from u to v and w. This implies that $u \in \mathrm{MFC}_v$, $u \in \mathrm{MFC}_w$, and because of Lemma 1, then $C_u \subseteq \mathrm{MFC}_v$ and $C_u \subseteq \mathrm{MFC}_w$. ■

Node duplication performance uses a DRPPU to cover a MFC that contains an overlapped region. Node duplication is generally very important to depth optimization because duplication usually increases DAG parallelism. Without node duplication, many multiple fan-out nodes may have to be explicitly implemented with DRPPUs, possibly causing large depth in the partitioning solution. Hence, these feasible cones are allowed to overlap, which means that nodes in the overlapped region must be duplicated when mapping DRPPUs. In fact, in order to achieve depth optimization, our algorithm is capable of duplicating nodes automatically when necessary.

Traditionally in DRPPU temporal partitioning, a graph is partitioned into several sub-graphs. The previously proposed algorithms [3] and [4] do not consider whether sub-graphs have parallelism or not. We consider that if the sub-graphs have inherent parallelism, after which we organize the properly-partitioned sub-graphs so that they can be executed during the same cycle, thereby improving performance.

B. MFC-Processing for DRPPU Capacity

In the preceding discussion we presented techniques for minimizing the depth in parallel partitioning. Assuming that the maximum parallelism of a given array is k (in this case, the number of FPGA in our simulated prototype's parallel array; in a more general case, this is the number of DRPPU in the array), then after finding all feasible floor cones to the mapped k DRPPUs, we still cannot be sure each MFC represents the maximal capacity of the DRPPU. Thus we define the *maximal mapping graph* MMG to be a set of MFCs such that MFC $\subseteq$ MMG. The objective of MFC-processing in our parallel partitioning algorithm is to find k MMG (k-MMGs) by selecting all MFCs that can be collocated under the DRPPU k capacity constraint (i.e. $\leq k$) and to confirm that the MFCs can collocate in a single layer.

On the other hand, it should be noted that MFC-processing affects the depth, i.e. the number of layers in the design. Since increasing the number of layers increases the execution time of the design, it is desired to minimize the depth. This is accomplished by use of a bin-packing technique which treats the problem as a matter of minimizing the number of MMGs per MFC set. Since there are m MFCs in a MFCs set, the size of each MFC is c_i which is a positive integer. We further give positive integers B and C as the number of bins and the bin capacity, respectively. The bin packing problem determines the minimum number of bins which can accommodate all m items. In general, the goal of bin packing is to find the minimum number of bins into which a set of boxes can be packed. In this case the bin packing problem is NP-complete, with the bins corresponding to the number of the DRPPUs and the boxes corresponding to the set of MFCs. The capacity of each bin is C, and the size of each box is the area size of the MFC. Herein, the bin packing algorithm used is First Fit Decreasing (FFD).

Figure 3(a) shows an example of the set of the MFCs. There are seven MFCs in G, with each floor cone having size weight $c_1 = 3$, $c_2 = 6$, $c_3 = 2$, $c_4 = 1$, $c_5 = 5$, $c_6 = 7$ and $c_7 = 2$ respectively. If the bin capacity of C is 9 then each floor cone corresponds to box capacity c_i, where $1 \leq i \leq 7$. First, we sort objects so that $c_i \geq c_{i+1}$, $1 \leq i \leq 7$ as shown in Figure 3(b) and then pack object i in bin j where j is the least index such that bin j can contain object i. In Figure 3(c) the final contents of the packed bins are 9, 9 and 8, such that $B = 3$.

Hence, the number of MFCs is decreased by applying bin packing as an MFC-processing step. When the number of bins is more than the k-MMG, we choose the k number of the largest capacity in bin B. There are other methods that can further decrease area cost after the MFCs have been produced.

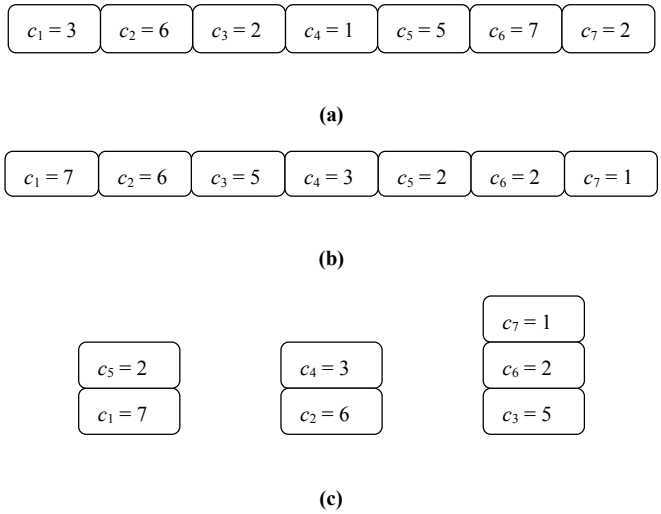


Figure 3. (a) A set of MFCs; (b) Sorting these MFCs; (c) Reducing the number of MFCs according to the bin packing FDD.

Alternative partitioning methods and their different results have been presented above. It has been shown that the objective is a minimum number of layers when a graph G is partitioned by applying $\leq k$ number of MMG's.

Theorem 1: If a DAG is partitioned by MFC with consideration of the $\leq k$-MMGs, then the result of partitioning is an optimal depth solution.

Proof: Given a graph G and in a PRC system with k DRPPU, G is partitioned into subgraphs to find the depth. Based on Lemmas 1 and 2, we can obtain every MFC with an optimal solution. Hence in graph G, k-MMG was found by the all MFCs. According to the MMG definition, each MMG mapped to a DRPPU has maximal area. Let S_g be a set of MMGs. Now we use divide and conquer to obtain the solution. For graph G, after we find k-MMG's S_g, the new graph G is produced by the procedure, $G = G - S_g$. Recursively, the step of finding k-MMG's S_g until all node have been processed in graph G. It means that the minimum-depth solution of G is the union of $\{S_g\}$ and the minimum-depth solution of $G - S_g$. Hence, account of the depth is an optimal solution since each step of finding k-MMG's is an optimal solution. ■

C. The Minimizing Depth Algorithm

A PRC parallel partitioning algorithm is presented in this subsection. Determining every MFC in a graph is first introduced. Then the minimum depth obtained recursively by obtaining k-MMGs is explained. Assume that the area constraint is 23, i.e. $A_{DRPPU} = 23$. For example, 3-MFCs including C_x, C_y and C_z in Figure 2 can be selected for the optimal solution, where 3-MFCs is equal to 3-MMGs.

The minimizing depth algorithm is given in Fig. 4. The algorithm needs to find all feasible floor cones to determine k-MMGs. The major work of parallel partitioning is to find every MFC in G. In every floor cone graph, there are three steps, namely calculating the area, sorting the area, and the labeling nodes. Application of the traversal technique applies a depth first search (DFS) for calculating the floor cone area. A floor cone is constructed with the node as a root if the floor cone is feasible. In the second step the sorting technique uses MergeSort. Moreover, we label the chosen root v and its predecessors, Pre (v) to get the S_c set, where S_c is a set of MFCs. Let TP (v) = $\bigcup_{u \in Pre(v)} C_u$. Based on the S_c set, our parallel algorithm can find k-MMGs such that the k-MMGs, S_g is obtained by using FFD method in the MFC-processing. When a new DAG $G = G - S_g$ is obtained, we return to the generate-MMG's step in the partitioning procedure to find the new depth of the feasible cone. New MMGs are generated to increase the number of depths so that the partitioning depth in graph G increases until all nodes is covered to the MFC.

```
Algorithm Determining Depth:
    FindMinimumDepth(G)
    Comment: G(V, E) is a directed acyclic graph
    depth = 0;
    L = ∅;
    SortList = ∅;
    S_c = ∅;
    S_g = ∅;
    for every node u in G
        L: = List L of all of nodes in topological order;
    end of the for loop
    while (L ≠ ∅) do
        for every node v in G do;
            CalculateArea(C_v) by DFS;
            if Area(C_v) ≤ A_DRPPU then do
                SortList ← C_v
        end of the for loop
        while (SortList ≠ ∅) do
            Sort area of SortList by MergeSort
            for every floor cones C_v in SortList do
                if C_v is the MFC then
                    S_c = S_c ∪ C_v;
                    L = L – {v ∪ Pre(v)};
                    SortList = SortList – {C_v ∪ TP(v)};
                end of the if loop
```

```
    end of the while loop
    for every MFC in S_c
        Finding S_g by FFD;
    end of the for loop
    G = G – S_g;
    S_g = ∅;
    S_c = ∅;
    depth = depth + 1;
end of the while loop
```

Figure 4. The minimum-depth algorithm for parallel partitioning

D. The Complexity of the Algorithm

The whole partitioning procedure has a low time complexity. Before finding the MFC, all the nodes are processed by topological sorting. Hence during portioning, the nodes are visited in topological order. The time complexity of visiting the nodes is $\mathbf{O}(|V| \bullet log|V|)$ where V is the number of nodes in the given graph. The task of finding each MFC includes calculating the area, sorting the area and labeling nodes. Each floor cone area is calculated by the depth first search (DFS) methodology. Hence, the time complexity of calculating the area is $\mathbf{O}(|V| \bullet log|V|)$. In sorting the area, the number of floor cones is no more than the number of nodes V. The time complexity is $\mathbf{O}(|V|)$. When labeling nodes, a node is visited once. The complexity of labeling nodes is $\mathbf{O}(|V|)$. In the MFC-processing, to apply FFD the number of MFC is no more than the number of nodes V. The time complexity is $\mathbf{O}(|V| \bullet log|V|)$. The minimum-depth partitioning solution selects k-MMG's in each greedy method iteration. Therefore, depth determining takes $\mathbf{O}(|V|^2 \bullet log|V|)$ time. In conclusion, the total time complexity is bounded by $\mathbf{O}(|V|^2 \bullet log|V|)$.

V. Experimental Results

The proposed PRC partitioning algorithm was implemented in C language on a Blade 1000 workstation. For performance evaluation the algorithm was assigned to partition a published DAG for an HMM algorithm. We derive 4 DAGs for this algorithm for four different levels of HMM complexity, i.e. for four different state numbers. Since higher state numbers imply much larger DAGs, this challenges the ability of our algorithm to minimize depth. Also, since algorithm depth is directly related to application execution speed, this is a good demonstration of the speed improvement that can be obtained by PRC.

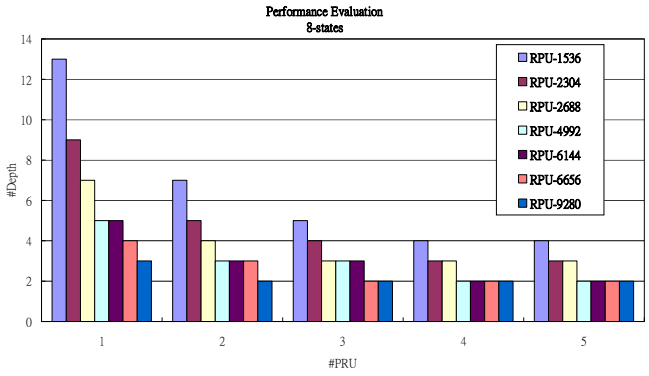
Our proposed algorithm can compute a theoretically unlimited number of parallel FPGA modules (DRPPU) but, for reasons of simple comparison, we perform simulations for parallel arrays from one to 5 DRPPU and for DRPPU areas (A_{DRPPU}) from 1536, 2304, 2688, 4992, 6144, 6656 to 9280 CLB. Thus we are demonstrating the ability of our algorithm to help design massively parallel architecture. In fact, FPGA is intrinsically capable of such function but application up to the present time has been largely linear, due to lack of design tools and the habitual persistence of traditional thinking. It will be seen that speed optimization is obtained by use of larger DRPPU numbers. Functional operations such as addition, subtraction, multiplication are implemented by CLBs of a DRPPU. Table I shows the number of CLBs in each of these basic operations, while columns 2 to 4 show the different bit widths. The following data present the results of using our algorithm to mapping HMM DAGs of varying state numbers onto PRC architecture. The HMM state numbers mapped are 4, 8, 12 and 24. Here we use PRU abbreviation to present the DRPPU in the below Table and Figure.

TABLE I. AREA OF THE OPERATIONS EXPRESSED IN XC4000 CLBS

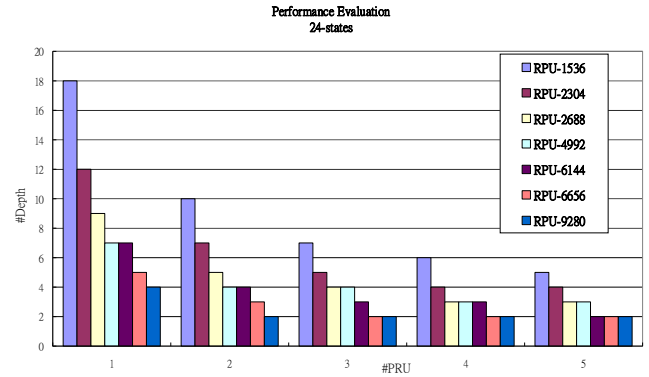
#CLBs / Operator	4 bits	8 bits	16 bits
Addition	3	5	9
Subtraction	7	13	25
Comparison	11	16	29
Multiplication	14	27	50

The results of partitioning the HMM (2 DAGs, 8 and 24 states) is presented in Fig. 5, with results given as minimum depth for each state conditions for a given number of parallel processing units (# of DRPPU), where each DRPPU group is subdivided into A_{DRPPU} (DRPPU area, i.e. number of CLB per DRPPU). Note that when the number of DRPPU equals one, this indicates a single FPGA which is equivalent to non-parallel processing. This value is given for comparison and to demonstrate

the capability of the presented algorithm. In Fig. 5, it is obvious that the depth decreases (i.e. the speed increases) as the number of DRPPU increases and also as the number of CLB per DRPPU increases.



(a)



(b)

Fig. 5. Evaluation results of architecture performance based on depth number: (a) 8-state HMM; (b) 24-state HMM.

Finally, we compare the speed performance of the same HMM implementation in different architectures. This is presented in Tables II and III. Columns 2 to 4 respectively show the T_{exe} for the same HMM as implemented by general purpose processing (GPP), parallel processing elements without the local memory of DRPPU (Fig. 1(a)) and the PRC method with fully equipped DRPPU. Performance comparison is based on time-step execution time T_{exe} analysis of the results of a 4-, 8-, 12- and 24-state speech recognition Viterbi HMM algorithm decoder in C running on a Core 2 1.86GHz CPU 2GB RAM machine with the each time-step of the execution time assigned to 10 ms. Table II shows the results for PE and DRPPU consisting of 1536 CLB each. For the 4-, 8-, 12- and 24-state HMM implementations, the proposed PRC design demonstrated average 12.26 and 3.53 execution time improvement relative to the GPP and the PE designs. Table III compares these systems when the computing block capacity is 2688 CLBs. Here PRC showed relative execution time improvements of approximately 17.12 and 3.36 with respect to the GPP and PE designs. The major improvement observed for the proposed PRC architecture (Fig. 1(a)) adopted in our DRPPU is attributable to the faster reconfiguration time resulting from the virtue improved bus cycle time.

TABLE II. COMPARISON OF DIFFERENT DESIGNS FOR EXECUTION TIME (NS)

Time / States	GPP	Parallel (1536clbs)	PRC (1536clbs)	Improvement	
				GPP	Parallel
4-states	10000	2910	660	14.15	3.41
8-states	10000	2970	660	14.15	3.50
12-states	10000	4000	880	10.36	3.55
24-states	10000	4120	880	10.36	3.68
Average				12.26	3.53

TABLE III. COMPARISON OF DIFFERENT DESIGNS FOR EXECUTION TIME (NS)

Time / States	GPP	Parallel (2688clbs)	PRC (2688clbs)	Improvement	
				GPP	Parallel
4-states	10000	1950	460	20.74	3.24
8-states	10000	1990	460	20.74	3.33
12-states	10000	3015	690	13.49	3.37
24-states	10000	3120	690	13.49	3.52
Average				17.12	3.36

VI. CONCLUSIONS

The addition of parallel processing techniques to reconfigurable computing has the potential to improve DSP applications. Therefore, multiple processing units arranged according to traditional parallel processing techniques are being applied for high computation and data intensive applications such as HMM. This paper has presented a minimum-depth partitioning algorithm for parallel reconfigurable computing. It is shown that application speed can be improved by increasing parallelism of the parallel DRC units. The resulting

high-parallelism design has somewhat higher total chip area because of redundancy between parallel units. The proposed algorithm can accept arbitrary chip area constraints or maximum parallelism constraint and then optimize for speed.

REFERENCE

[1] J. Noguera and R. M. Badia, "HW/SW Codesign Techniques for Dynamically Reconfigurable Architectures," *IEEE Transactions on VLSI*, vol. 10, pp. 399-415, Aug. 2002.

[2] T. Fujii *et al*, "A dynamically reconfigurable logic engine with a multiconfiguration/ multi-mode unified-cell architecture," in *Proc. IEEE Int. Solid-State Circuits Conf.*, 1999, pp. 364-365.

[3] G. M. Wu, J. M. Lin, and Y. W. Chang, "Generic ILP-based approaches for time-multiplexed FPGA partitioning," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 10, pp. 1266-1274, Oct. 2001.

[4] Y. C. Jiang and J. F. Wang, "Temporal Partitioning Data Flow Graphs for Dynamically Reconfigurable Computing," *IEEE Transactions on VLSI*, vol. 15, no. 12, pp. 1351-1361, Dec. 2007.

[5] W. Stallings, "Computer organization and architecture: designing for performance," Pearson Education, 2003.

[6] [Online]. Available: http://www.xilinx.com/

[7] L. F. Chen, Y. K. Lai, "VLSI architecture of the reconfigurable computing engine for digital signal processing applications," *IEEE Circuits and Systems Conference.*, ISCAS '04. pp. 937-40, May 2004.

[8] Vissers, K. A, "Parallel processing architectures for reconfigurable systems," Design, Automation and Test in Europe Conference and Exhibition, 2003 pp. 396 - 397

[9] H. Schmit *et al*, "PipeRench: A virtualized programmable datapath in 0.18 micron technology," *IEEE Custom Integrated Circuits Conference.*, pp. 63-66, May 2002.

[10] H. Singh, G. Lu, M. Lee, F. J. Kurdahi, N. Bagherzadeh, E. Filho, R. Maestre, "MorphoSys: Case Study of a Reconfigurable Computing System Targeting Multimedia Applications," *Proceedings Design Automation Conference* (DAC'00), pp. 573-578, Los Angeles, California, May 2000.

[11] R. Maestre, F. J. Kurdahi, M. Fernández, R. Hermida, N. Bagherzadeh, H. Singh, "Kernel Scheduling Techniques for Efficient Solution Space Exploration in Reconfigurable Computing," Special Issue on Modern Methods and Tools in Digital System Design, in the *Journal on System Architecture*, 47, pp. 277-292, 2001.

[12] S. A. Fahmy, Peter Y. K. Cheung and W. Luk, "Hardware acceleration of Hidden Markov Model decoding for person detecction," Design, Automation and Test in Europe Conference and Exhibition, 2005, pp. 8-13.

[13] Y. Zhu and M. Benaissa, "A novel acs scheme for area-efficient viterbi decoders," In Proc. IEEE International Symposium on Circuits and Systems. ISCAS '03., vol. 2, pp. 264–267, May 2003.