Design of discrete-coefficient FIR filters on loosely connected parallel machines

Sun, Y.; Lim, Yong Ching; Yu, Ya Jun

2002

Lim, Y. C., Sun, Y., & Yu, Y. J. (2002). Design of discrete-coefficient FIR filters on loosely connected parallel machines. IEEE Transactions on Signal Processing, 50(6), 1409-1416.

https://hdl.handle.net/10356/91423

https://doi.org/10.1109/TSP.2002.1003064

Design of Discrete-Coefficient FIR Filters on Loosely Connected Parallel Machines

Yong Ching Lim, *Fellow, IEEE*, Y. Sun, and Ya Jun Yu, *Student Member, IEEE*

***Abstract*—This paper presents a new branch-and-bound mixed-integer linear programming-based algorithm for designing discrete-coefficient finite-impulse response (FIR) filters using a cluster of workstations as the computation platform. The discrete coefficient space considered in this paper is the sum of signed power-of-two space, but the technique is also applicable to other discrete coefficient spaces. The key issue determining the success of the algorithm is the ability to partition the original problem into several independent parts that can be distributed to a cluster of machines for solution. The master–slave model is adopted for the control of the machines. Test run results showed that super linear speedup (i.e., the speedup factor is more than the number of machines running in parallel) may be achieved.**

I. Introduction

IF A finite-impulse response (FIR) filter is to be implemented in a digital signal processor (DSP) with adequate word length and computational power, it is not necessary to optimize the filter subject to finite word length and discrete coefficient constraints. If the filter is to be implemented on a platform whose cost is sensitive to circuit complexity such as in a full custom application specific integrated circuit (ASIC), in order to minimize implementation cost, it is necessary to optimize the filter subject to finite word length and discrete coefficient constraints.

The three major components of a digital filter are delay, adder, and multiplier. The cost of each of these components depends very much on the implementation methodology. For example, each delay element requires 16 transistors when implemented using a shift register in static CMOS technology. If implemented using a static CMOS RAM, it requires six transistors per bit of delay element. When implemented on DRAM, each delay element requires only one transistor. In multiple-valued logic technology, several binary bits of delay element share one transistor. Thus, the cost of a delay element differs widely from one implementation methodology to another.

A static CMOS full adder requires about two dozen transistors. Pseudo nMOS or nMOS implementation requires half the number of transistors. The number of full adders required to implement an adder is equal to the word length. The number of transistors required is approximately doubled if carry lookahead is used.

Manuscript received December 29, 2000; revised February 5, 2002. The associate editor coordinating the review of this paper and approving it for publication was Prof. Paulo S. R. Diniz.

Y. C. Lim and Y. J. Yu are with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore (e-mail: elelimyc@nus.edu.sg).

Y. Sun is with Silicomp Asia Pte Ltd., Singapore.

Publisher Item Identifier S 1053-587X(02)04391-X.

The cost of a multiplier depends very much on its speed. In a fully parallel implementation, the number of full adders required by a multiplier is equal to the square of the word length. Because of its very high complexity, its speed is also slower than an adder. The complexity of a multiplier can be reduced in serial implementation by trading off its speed.

From the above analysis, it is clear that the design of the cost-wise globally optimum filter requires *a priori* knowledge of a large number of factors, many of which are particular to specific implementation methodology. This paper focusses on optimizing the complexity of the filter's coefficient values since the multiplier is apparently the element that requires the largest number of transistors in an ASIC implementation. Many papers on finite wordlength or power-of-two design technique [1]–[7], [13]–[21] and sparse coefficient techniques [22]–[26] have been published in the literature. In this paper, our efforts focus on developing very efficient algorithms for the design of FIR filters whose coefficient values are the sum of a limited number of signed power-of-two (SPT)terms. Since, in binary arithmetic, multiplying a power-of-two with another number is a very simple process, filters whose coefficient values are the sum of a limited number of SPT terms are essentially without multipliers. Mixed-integer linear programming (MILP) is the only known method that can provide the global optimum solution to the design of FIR filters with SPT coefficient values [2], [4], [27], although other computationally very much less demanding technique exist [29]. As MILP requires an excessively long computation time when solved on a single machine, this paper reports on designing this class of filters on loosely connected parallel machines such as a cluster of workstations. It should be noted that the computer time required by running MILP is several orders of magnitude [29] of that required by other suboptimum techniques if only one processor is used; running MILP on parallel machines still requires more computer time than running other techniques on a single processor, unless a very large number of processors are available.

The advantage of using a cluster of workstations as the computational platform is that a cluster of a large number of workstations is easily available. A disadvantage is that there may be many users running unrelated tasks on the workstations so that the available computing resources for each task fluctuates in an unpredictable manner. Furthermore, communication between workstations is slow.

An introduction to the branch and bound (B&B) MILP for the purpose of defining notations is presented in Section II, and existing parallel algorithms are briefly described in Section III. Section IV presents an overview of our master–slave algorithm. Detailed descriptions of our slave algorithm and master algorithm are presented in Sections V and VI, respectively. The performance of our algorithms are analyzed in Section VII.

II. B&B MILP

In order to fix idea and to define notations, we briefly explain the B&B MILP technique for the design of discrete coefficient FIR filters. In particular, we will explain why the depth-first search technique is a good technique for designing FIR filters when only a single machine is available.

In the design of an FIR filter using MILP, the frequency response of the filter is evaluated on a dense grid of frequencies forming a set of linear inequalities. An objective function, which is the weighted peak ripple magnitude or a linear combination of the weighted peak ripple magnitude and the passband gain [2], is then minimized using a linear programming (LP) algorithm. Call this LP problem P_0. (See the Appendix.) The LP solution is the minimax optimum solution when the coefficient values of the filter are not subject to the discrete value constraint. Let this solution be S_0. From S_0, a filter coefficient $h(n_0)$ whose value is desired to be discrete is identified. Let the value of $h(n_0)$ in S_0 be $\tilde{h}(n_0)$. Let the permitted discrete value of $h(n_0)$ immediately larger than and smaller than $\tilde{h}(n_0)$ be $\lceil \tilde{h}(n_0) \rceil$ and $\lfloor \tilde{h}(n_0) \rfloor$, respectively. Since $h(n_0)$ must be discrete, it must satisfy either $h(n_0) \geq \lceil \tilde{h}(n_0) \rceil$ or $h(n_0) \leq \lfloor \tilde{h}(n_0) \rfloor$. P_0 is then partitioned into two subproblems P_{00} and P_{01}. P_{00} may be obtained by imposing the constraints $h(n_0) \leq \lfloor \tilde{h}(n_0) \rfloor$ [or $h(n_0) \geq \lceil \tilde{h}(n_0) \rceil$] on P_0. P_{01} may be obtained by imposing the constraints $h(n_0) \geq \lceil \tilde{h}(n_0) \rceil$ [or $h(n_0) \leq \lfloor \tilde{h}(n_0) \rfloor$] on P_0.

Let the solution of $P_{0,\ldots,j}$ be $S_{0,\ldots,j}$. In general, if $S_{0,\ldots,j}$ is not a discrete solution, then $P_{0,\ldots,j}$ is partitioned into two subproblems $P_{0,\ldots,j,0}$ and $P_{0,\ldots,j,1}$ by adding the constraints

$$h(n_{0,\ldots,j}) \leq \left\lfloor \tilde{h}(n_{0,\ldots,j}) \right\rfloor \tag{1}$$

and

$$h(n_{0,\ldots,j}) \geq \left\lceil \tilde{h}(n_{0,\ldots,j}) \right\rceil \qquad j = 0 \text{ or } 1 \tag{2}$$

to $P_{0,\ldots,j}$, provided that $\tilde{h}(n_{0,\ldots,j})$ is not already a discrete value. A B&B tree, as shown in Fig. 1, is thus obtained. Note that the union of the discrete solution spaces for $P_{0,\ldots,j,0}$ and $P_{0,\ldots,j,1}$ is the same as the discrete solution space for $P_{0,\ldots,j}$. Thus, the optimum discrete solution found in $P_{0,\ldots,j,0} \cup P_{0,\ldots,j,1}$ is the same as that in $P_{0,\ldots,j}$, where $\cup$ denotes the union operator. Hence, the B&B algorithm produces the global discrete optimum solution.

Each subproblem $P_{0,\ldots,j}$ represents a vertex in the B&B tree. $P_{0,\ldots,j,k}$ is called a successor of its predecessor $P_{0,\ldots,j}$, where $k = 0$ or 1. The line joining $P_{0,\ldots,j}$ and $P_{0,\ldots,j,k}$ is called a tree branch. When no further exploration will be performed on a vertex (for whatever reason), that vertex is said to be fathomed. A vertex is a live vertex if its successors have not been completely explored. The lexicographical order of a vertex $P_{0,\ldots,j}$ is $0, \ldots, j$. (In lexicographic term, 00 precedes 000, 000 precedes 001, and 001 precedes 01.) The branch length of a vertex $P_{0,\ldots,j}$ is the number of tree branches joining P_0 and $P_{0,\ldots,j}$.

Let $f_{0,\ldots,j}$ be the value of the objective function in $S_{0,\ldots,j}$. Since $P_{0,\ldots,j}$ is obtained by adding a constraint on $h(n_{0,\ldots,j})$, we have $f_{0,\ldots,j} \leq f_{0,\ldots,j,k}$. Thus, if there exists a known discrete solution with objective function value smaller than $f_{0,\ldots,j}$, the part of the B&B tree below $P_{0,\ldots,j}$ need not be searched any further since the solution is not going to be better than the known solution; $P_{0,\ldots,j}$ is fathomed.

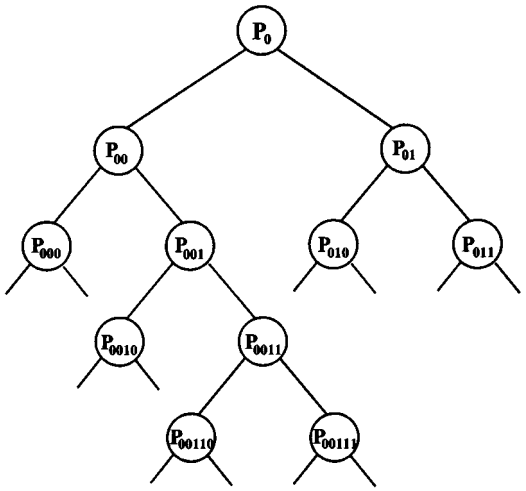

Fig. 1. Branch and bound tree. Each circle represents a vertex of the tree.

III. Parallel Algorithms

There are many reports on the development of algorithms for executing on parallel plateforms. Parallel algorithm techniques are specific to particular applications and are designed to optimize specific performance criteria. Karp and Zhang [8] suggested a parallel best-first search B&B algorithm with randomized load balancing strategy. The algorithm adopted a synchronous model, i.e., a processor could not proceed until all other processors had completed their respective computational and communicational tasks in the current iteration. Therefore, its speedup factor is poor because of the synchronization constraint. In [9], Yang and Das proposed and evaluated a parallel best-first search B&B algorithm implemented on a multistage interconnection network system called the butterfly system. This technique achieved a good speedup factor due to both the reasonably small number of node evaluations and a relatively low communication overhead, even though the number of processors was large. Unfortunately, it required a special hardware platform, i.e., the machines must be wired in a special way. Kudva and Pekny [10] developed a tool that was aimed at reducing the burden associated with designing and implementing B&B algorithms in a distributed computer network environment. There was no centralized task scheduling processor. Therefore, no single processor could be a bottleneck, and a failed processor could be replaced by a good one. However, the job of workload balancing is complicated; a processor requiring a task needed to poll its neighboring processors to check if any one of them could allow it to share its work. This caused a significant increase in communication overhead. In [11], Taudes and Netousek studied the feasibility of implementing parallel B&B algorithms on a cluster of workstations, which can be classified as a loosely coupled multicomputer. Based on the experimental results, they concluded that given proper tuning, a distributed B&B algorithm could yield satisfactory speedup on a cluster of workstations; the required tuning was specific to their particular vertex-cover-problem, which was quite different from the design of digital filters using MILP. In [12], Wiegers eliminated the detrimental anomaly for a depth-first parallel B&B algorithm of a master–slave model using several

measures, such as evaluating nodes synchronously and using local information to select branching variable. In the design of digital filters using MILP, we had never encountered the occurrence of detrimental anomaly. Detrimental anomaly is the anomaly where the speed of a multiprocessor system becomes slower than that of a single processor system. Our interest was on achieving a high acceleration anomaly.

IV. Overview of Our Algorithm

Our algorithm belongs to the master–slave type. The computational platform is a cluster of loosely connected workstations (the machines). Each slave algorithm will run on a machine. The master algorithm does not require many computing resources to execute and may share the computing resources of a machine already executing a slave algorithm. Direct communication is permitted only between master and slave algorithms. Direct communication between slave algorithms is not permitted.

Because of the slow communication speed between machines, communications between master and slaves is kept to a minimum. The master performs initialization, assigning subproblems to the slaves, and informing the slaves on any discrete solution already found. Each of the slaves performs B&B tree search on a subproblem adopting a depth-first search strategy. Upon discovering an improved discrete solution, the slave will immediately report this improved discrete solution to the master. The master will then broadcast the objective function value of the improved solution to all other slaves. When a slave has completed its assigned task, it will report to the master to request for a new job assignment; upon receiving the "job request" signal, the master will cut a scion from a live vertex of a slave and graft it to the slave requesting new job. A scion is a section of the B&B tree such as that containing P_{001} and its successors in Fig. 1.

The computing time required to find the optimum solution is influenced by the following factors.

a) After optimizing $P_{0,\ldots,j}$, it is necessary to identify $h(n_{0,\ldots,j})$ for partitioning $P_{0,\ldots,j}$ into $P_{0,\ldots,j,0}$ and $P_{0,\ldots,j,1}$. The choice of $h(n_{0,\ldots,j})$ significantly affects the size of the B&B tree and, hence, the computing time.

b) When a slave has completed its assigned task, it will request that the master assign it a new task. A scion will be cut from a live vertex in another slave and be grafted to the slave requesting the new task. Several strategies for selecting the scion for transfer are being investigated, and their effects are reported in Sections V and VI of this paper.

c) The efficiency of the slave solving the assigned task obviously also affects the computing time.

V. Slave Algorithm

The slave adopts a depth-first search strategy as follows. Suppose that the problem $P_{0,\ldots,j}$ is assigned to a given slave. $P_{0,\ldots,j}$ is called the root vertex in that slave. After $P_{0,\ldots,j}$ has been optimized, a coefficient, say $h(n_{0,\ldots,j})$, is selected, and $P_{0,\ldots,j}$ is partitioned into two successors by imposing the constraints $h(n_{0,\ldots,j}) \leq \lfloor \tilde{h}(n_{0,\ldots,j}) \rfloor$ and $h(n_{0,\ldots,j}) \geq \lceil \tilde{h}(n_{0,\ldots,j}) \rceil$, respectively, on $h(n_{0,\ldots,j})$. The constraint that will yield a smaller penalty value is assigned to $P_{0,\ldots,j,0}$. The one that will yield a larger penalty value is assigned to $P_{0,\ldots,j,1}$. The penalty value [28] of a constraint is the degradation on the objective function when the constraint is imposed. $P_{0,\ldots,j,1}$ is stored, and $P_{0,\ldots,j,0}$ is optimized. In general, at any vertex $P_{0,\ldots,k}$ of the B&B tree, two subproblems $P_{0,\ldots,k,0}$ and $P_{0,\ldots,k,1}$ are generated by imposing constraints on a coefficient $h(n_{0,\ldots,k})$. The constraint that yields a smaller penalty value is assigned to $P_{0,\ldots,k,0}$. $P_{0,\ldots,k,1}$ is stored for future exploration, and $P_{0,\ldots,k,0}$ is explored immediately. Each time a subproblem is partitioned, the branch length of the B&B tree is increased by one. The exploration of $P_{0,\ldots j,\ldots,0}$ continues until a discrete solution is found or until a point where it is known that further exploration will not yield a solution better than the one best currently known. If a discrete solution is found, the discrete solution is transmitted to the master. A B&B tree, as shown in Fig. 2, is generated. In this strategy, the search advances quickly, deep into the B&B tree and, hence, earns the name "depth-first search." In Fig. 2, it is assumed that either $S_{0,\ldots,m,0}$ is a discrete solution or $f_{0,\ldots,m,0}$ is larger than the objective function value of a known discrete solution. $P_{0,\ldots,m,0}$ is fathomed, and the search switches to explore $P_{0,\ldots,m,1}$ and its successors in a depth-first manner, as illustrated in Fig. 3. After completing the exploration of the successors of $P_{0,\ldots,m,1}$, the search is backtracked, as shown in Fig. 3.

After the optimization of $P_{0,\ldots,k}$ has been completed, $h(n_{0,\ldots,k})$ is identified for partitioning $P_{0,\ldots,k}$ into $P_{0,\ldots,k,0}$ and $P_{0,\ldots,k,1}$. It is hoped that the constraint imposed on $h(n_{0,\ldots,k})$ is so unfavorable for $P_{0,\ldots,k,1}$ that $P_{0,\ldots,k,1}$ need not be searched when the depth-first search algorithm backtracks. Thus, all $h(n)$ that are in the basis vector of the simplex algorithm are tested one-by-one to see which one produces the largest penalty value. The value of $n_{0,\ldots,k}$ is the value of n that will produce the largest penalty value when the constraint is imposed on $h(n)$. In order to save computing time, when computing the penalty value, $f_{0,\ldots,k,1}$ is not evaluated but is estimated by taking the value in the next simplex iteration. The one that yields the largest penalty value is selected as $h(n_{0,\ldots,k})$.

Let $P_{original\text{-}root}$ be the root vertex of a tree in a slave machine. After all the search-worthy successors of $P_{original\text{-}root,0}$ have been explored, the algorithm backtracks to $P_{original\text{-}root}$ and then switches to explore the successors of $P_{original\text{-}root,1}$. Once the algorithm switches to explore $P_{original\text{-}root,1}$, it will never be necessary to backtrack to $P_{original\text{-}root}$. Thus, $P_{original\text{-}root}$ can be discarded, and $P_{original\text{-}root,1}$ becomes the new root vertex. The master is informed of the change in root vertex.

A slave may be interrupted by the master under two circumstances, namely, 1) to update the best known objective function value of a discrete solution and 2) to request the slave to transfer a scion. It should be pointed out that a slave's local record of the best-known objective function value is not a record of the best objective function value obtained by the slave itself; it is a record of the globally best-known objective function value.

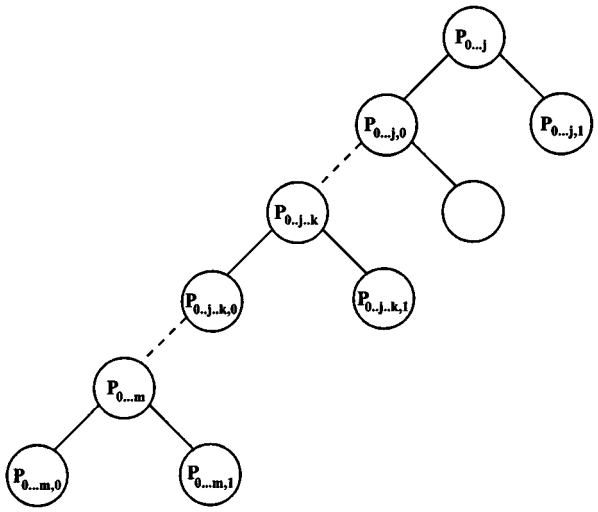

Fig. 2. Depth-first tree search.

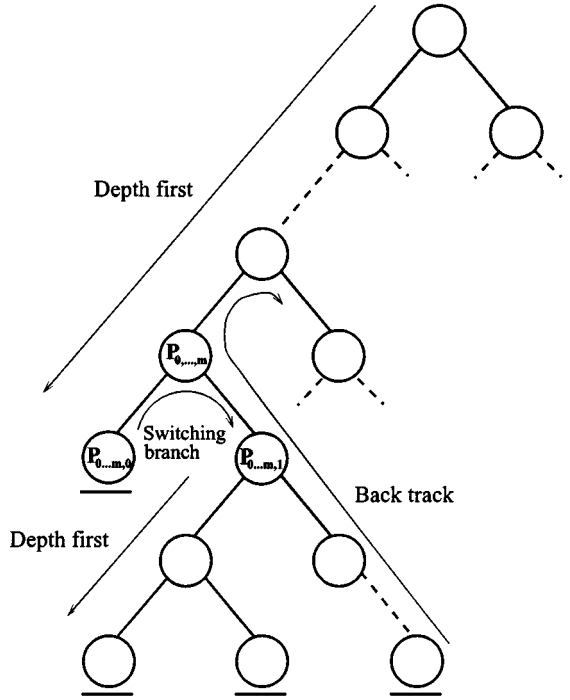

Fig. 3. Switching branch and back tracking processes in a depth-first search. A vertex with an underscore signifies that the vertex is fathomed.

If a slave is requested to transfer a scion, the slave will examine the objective function value of $P_{root,1}$, where P_{root} is the root vertex. If $P_{root,1}$ is search worthy, $P_{root,1}$ is transferred to the master. After $P_{root,1}$ is transferred, P_{root} is discarded, and $P_{root,0}$ becomes the new root vertex. The master will be informed regarding the change in the root vertex. If the objective function value of $P_{root,1}$ is larger than the best-known objective function value, $P_{root,1}$ is not search worthy and is discarded. $P_{root,0}$ becomes the new root vertex. New $P_{root,1}$ is examined for possible transfer to the master in the same way as old $P_{root,1}$ was examined. Information regarding the new root vertex is also transmitted to the master.

A slave interrupts the master under the following conditions.

1) After P_{root} has been partitioned into $P_{root,0}$ and $P_{root,1}$, the relevant P_{root} information is transmitted to the master. P_{root} may either be the root vertex of the scion grafted to the slave by the master or a vertex that is being promoted to the status of a root vertex.
2) When the slave has found a discrete solution whose objective function value is better than the best-known objective function value, the solution is transmitted to the master.
3) When a slave has searched all the search-worthy vertices of the B&B tree assigned to it, it will request a new job assignment from the master.

Adopting the depth-first search technique for the slave has several important advantages.

a) The storage space requirement for storing the unsolved subproblems of the B&B tree is not too large.
b) Suboptimum discrete solutions are produced during the optimization process. The suboptimum intermediate solution obtained by one slave can be used by other slaves to terminate any search along an unfruitful path. This results in a reduction in the total number of vertices being searched and causes an acceleration in finding the global optimum solution as the number of slave machines running in parallel is increased. Because of the reduction in the total number of vertices, the speed of solving the problem may be increased by a factor larger than the factor by which the number of machines running in parallel is increased. The phenomenon that the speedup factor is larger than the number of machines running in parallel is called super linear speedup.
c) In the event of having insufficient computing resources to obtain the optimum solution, the suboptimum solution produced by the depth-first search during the optimization process may be useful.

VI. Master Algorithm

The master performs initialization and keeps a record of the relevant root vertex information of the B&B tree of each of the slave algorithm. Based on this information, whenever a slave interrupts the master to request for a new job assignment, the master identifies a slave where a scion is to be cut for grafting to the idling slave. Whenever a slave informs the master on the discovery of a discrete solution whose objective function value is better than the previously best-known objective function value, the master broadcasts this currently best-known objective function value to all the slaves.

The master algorithm consists of two phases. In the first phase, the slaves are assigned with jobs as soon as possible in the following manner. After P_0 is solved, it is partitioned into P_{00} and P_{01}. P_{00} and P_{01} are solved in two separate slaves. All other slaves (if any) will be queued up for job assignment since it is not possible to assign a job to the idle slaves until at least one of the slave solving P_{00} or P_{01} has partitioned its P_{root} into $P_{root,0}$ and $P_{root,1}$. When any of the slaves that have been assigned task has partitioned its P_{root} into $P_{root,0}$ and $P_{root,1}$, the master will request that slave to cut a scion and graft the scion to an idling slave in the waiting queue. This process

continues until all the slaves have been assigned a respective task; this ends the first phase of the master algorithm.

The second phase of the master algorithm starts immediately after the conclusion of the first phase. During the second phase, when a slave requests a new job assignment from the master, the master will place the requesting slave in a queue if there is more than one slave requesting a job assignment. (It is immaterial which slave in the queue is served first.) The master then chooses a slave among those slaves currently busy searching the B&B tree and requests for it to contribute a scion based on a predefined strategy. We have investigated three possible strategies for selecting a slave to contribute the required scion. The selection is based on either

1) the relative lexicographical values;
2) the branch lengths;
3) the objective function values of the root vertices of all the B&B trees of all the slaves.

- Strategy 1) Select the slave whose root vertex is lexicographically most preceding.
- Strategy 2) Select the slave where the branch length of its root vertex is the shortest.
- Strategy 3) Select the slave whose root vertex has the smallest objective function value.

We will use the scenario of Fig. 4 to illustrate the above strategies. In Fig. 4, there are three busy slaves S_1, S_2, and S_3 searching the subtrees T_1, T_2, and T_3, respectively. Assume that the root vertices of the three subtrees are A, B, and C, and the corresponding objective function values are 0.3, 0.1, and 0.2, respectively. When an idling slave S_4 requests a job assignment, a live vertex must be transferred to it. For Strategy 1), vertex A is lexicographically most preceding. Thus, slave S_1 is selected to contribute a scion. For Strategy 2), the branch length of vertex C is the shortest, and slave S_3 is selected. Finally, for Strategy 3), vertex B has the smallest objective function value; hence, slave S_2 is selected.

Note that when there are only two slaves, the three strategies are indistinguishable. This is because when a slave is idle, the remaining one is the only one that can be selected.

VII. Performance Analysis

Arithmetic Mean Versus Geometric Mean: We make use of the results collected from a large number of examples to evaluate the performance of the various strategies. Since the results vary from example to example, comparisons are made based on the average values of the results. There are many ways to determine the mean value of a collection of numbers; arithmetic mean and geometric mean are two of the ways. The arithmetic mean has several obvious advantages, but it has the disadvantage that the mean value is determined mainly by those numbers with large values; the effect of a number with small magnitude is almost the same as that of a zero. The geometric mean has the advantage that equal fractional change in the numbers will yield equal fractional change in the geometric mean. Obviously, whether the arithmetic mean is a more suitable choice than the geometric mean or vice versa will depend on specific circumstances. We choose the geometric mean for the purpose of putting equal weight on all the data obtained.

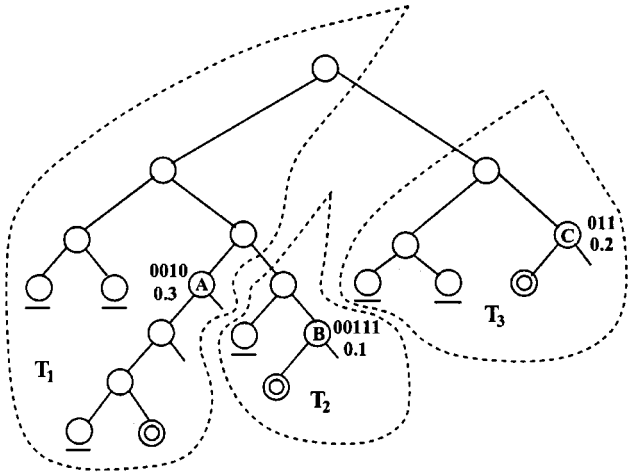

Fig. 4. Vertices with two concentric circles are the vertices being explored currently by the slave machines.

Cost Ratio: Suppose that P processors are used in parallel to solve a given problem. The *cost* of solving the problem, which is denoted by $T_C(P)$, is defined as the sum of the CPU execution time (the time the computer is actually actively involved in solving the problem) of the P processors. The *cost ratio* $C(P)$ is defined as the ratio of the cost of solving the problem using P processors to the cost of using one processor, i.e.,

$$C(P) = \frac{T_C(P)}{T_C(1)}.$$

In our master–slave model, the CPU time required by the master is ignored in all analysis because it is negligibly small when compared with that required by the slaves. Hence, P is actually the number of the slave processors. Acceleration anomaly is said to have occurred if $C(P) < 1$.

Size Ratio: The number of vertices on the B&B tree is a measure of the *size* of a problem. Let $N(P)$ be the total number of vertices when P slaves are used. The *size ratio* is given by

$$Z(P) = \frac{N(P)}{N(1)}.$$

Speedup Factor: A very commonly used, and one of the most important, criterion determining the usefulness of a parallel algorithm is the *speedup factor*. The *speedup factor* $S(P)$ is defined as the ratio of the wall clock execution time needed if only one processor is used to that needed when P processors are used in parallel, i.e.,

$$S(P) = \frac{T_K(1)}{T_K(P)}$$

where $T_K(P)$ is the wall clock execution time when P processors are used. The wall clock time is the terrestrial time, as indicated by a wall clock or wrist watch.

A high *speedup factor* indicates effective use of processors. For most parallel algorithms, the *speedup factor* is less than P when solving a problem using P processors. However, for our parallel algorithm, the speedup factor may be greater than P due to the acceleration anomaly as a result of the reduction in the total number of vertices being searched when the number of slave machines running in parallel is increased.

Efficiency: The *efficiency* is an important measure for the performance of a parallel algorithm. It indicates the effectiveness of the processors. Denoted by $E(P)$, the *efficiency* is given by

$$E(P) = \frac{S(P)}{P}.$$

Since $E(P)$ is the ratio of $S(P)$ to P, it is also the *normalized speedup factor*. When acceleration anomaly occurs, $E(P) \geq 1$.

Hardware Utilization: Because of changes in the total number of vertices being searched when P processors are used, the *efficiency* is not suitable to describe the extent to which the slave processors are utilized. The ratio of CPU time to wall clock time is a better indicator for the utilization of the slave processor. In our algorithm, all slaves commence computation and stop computation at almost the same time; therefore, they have almost the same wall clock time $T_K(P)$. Therefore, the ratio of the total CPU time of all the slaves, $T_C(P)$ to the wall clock time $T_K(P)$ gives the number of effective slaves that are being utilized. By computing the ratio of the number of effective slaves to the number of all slaves, a measure called *hardware utilization*, which is defined as

$$U(P) = \frac{T_C(P)}{T_K(P) \times P}$$

is obtained.

Since, for each slave, the wall clock time is greater than or equal to the CPU time, the product of $T_K(P)$ and P is greater than or equal to $T_C(P)$. Hence, the value of $U(P)$ ranges from 0 to 1 and is suited to describe the extent to which the slave processors are utilized.

In Figs. 5–9, $\overline{C}(P)$, $\overline{Z}(P)$, $\overline{S}(P)$, $\overline{E}(P)$, and $\overline{U}(P)$ are the geometric means of $C(P)$, $Z(P)$, $S(P)$, $E(P)$, and $U(P)$, respectively. The statistics shown in Figs. 5–9 were derived from a random choice of results obtained from about 160 examples with specifications spreading over a diverse range. They include even and odd length filters, symmetrical and antisymmetrical impulse response filters with various coefficient resolutions, and lowpass, highpass, bandpass, and bandstop filters with various band edges. The hardware platform was a cluster of HP9000/712 workstations used exclusively (i.e., no other jobs were running on the machines) for the experiments. The total number of workstations in the cluster was 24 and shared a 10 Mb/s communication link. The workstations in the same cluster that were not being used by this experiment were running unrelated tasks. The loading of the communication link due to the other workstations running unrelated tasks was unknown. Nevertheless, actual simulation results showed that the communication overhead was very much lower than the computational cost. Fig. 5 shows the $\overline{C}(P)$ versus P plots. As can be seen from Fig. 5, the values of $\overline{C}(P)$ are less than 1 due to acceleration anomaly. For a given P, $\overline{C}(P)$ corresponding to Strategy 3) is the smallest, and that corresponding to Strategy 1) is the largest. $\overline{C}(P)$ is a monotonically decreasing function of P for Strategy 3), indicating that acceleration anomaly increases with P. For Strategy 1) and Strategy 2), $\overline{C}(P)$ decreases with P for small P but increases with P for large P. Whenever a scion is grafted to a slave, an overhead is incurred in setting up the optimization problem in the slave. Communication overhead is also incurred in the data transfer. The number of times scion is grafted to slave increases with increasing P. Thus, overhead cost increases with increasing P. For $\overline{C}(P)$ to decrease with increasing P, the *size* of the problem must decrease at a rate faster than the increase in the overhead cost as P increases.

Fig. 6 shows the *average size ratio* $\overline{Z}(P)$ versus the number of slaves P plots. As can be seen from Fig. 6, the *average size ratios* are all less than unity. This shows that intermediate suboptimum solutions obtained by a slave machine are indeed useful for other slave machines to terminate the searches along unfruitful paths, causing a reduction in the total number of vertices being searched. This is the main contribution to acceleration anomaly. Notice also from Fig. 6 that the values of $\overline{Z}(P)$ for Strategy 1) and Strategy 2) are similar and approaches a constant asymptotically as P increases. Since overhead cost increases with increasing P, $\overline{C}(P)$ for Strategy 1) and Strategy 2) increases with increasing P for large values of P when $\overline{Z}(P)$ fails to decrease. The values of $\overline{Z}(P)$ for Strategy 3) are significantly smaller and decrease with increasing P; this explains why $\overline{C}(P)$ continues to decrease with increasing P for Strategy 3). Future research in this area will focus on finding strategies to decrease the *size ratio* significantly.

Fig. 7 shows the *average speedup factor* $\overline{S}(P)$ versus the number of slaves P plots. As can be seen from Fig. 7, $\overline{S}(P)$ is almost a linear function of P, at least for small P, and that Strategy 3) offers the highest *speedup factor*. It is expected that a strategy with a smaller $\overline{C}(P)$ will yield a higher $\overline{S}(P)$.

Fig. 8 shows the *average efficiency* $\overline{E}(P)$ versus the number of slaves P plots. The values of $\overline{E}(P)$ increase with P for small P but decrease with P for large P. The decrease in $\overline{E}(P)$ for increasing P for Strategy 3) when P is large is less significant, compared with the other two strategies. Therefore, the scalability (the suitability for use with a large number of processors) for Strategy 3) is the highest among the three strategies.

Fig. 9 shows the *average hardware utilization* $\overline{U}(P)$ versus the number of slaves P plots. Low $\overline{U}(P)$ indicates high communication overhead. Since communication overhead increases for increasing P, $\overline{U}(P)$ is a monotonic decreasing function of P. A strategy with a lower data transfer between master and slave will have a higher $\overline{U}(P)$. The results of Fig. 9 show that Strategy 2) yields the highest hardware utilization. This is expected since the scion is cut as close to the root as possible; this results in longer search path in the slave and results in less data transfer between master and slave. Hardware utilization will affect the *speedup factor* significantly if it is too low, such as in the case where the master and slaves are linked by a very slow communication link. Thus, in the situation where the communication link between the machines is very slow, such as in the case where the machines are physically separated by long distances, Strategy 2) may out perform Strategy 3) in terms of *speedup factor*.

Since, in most cases, the main objective of parallelization is to increase the *speedup factor*, Strategy 3) is the best strategy when the machines running in parallel are interconnected by a reasonably fast communication link. Strategy 2) may be the best strategy when the machines are interconnected by a slow communication link.

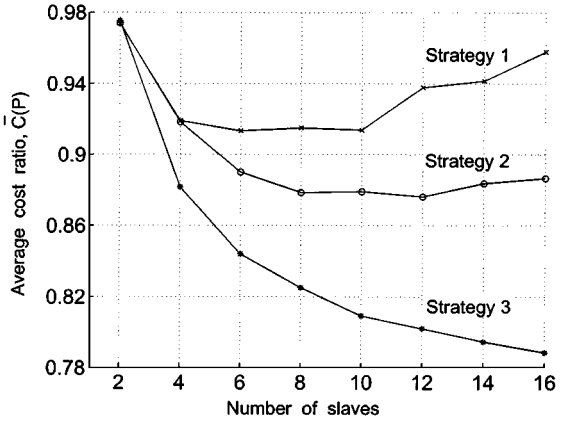

Fig. 5. Average *cost ratio* versus number of slaves plots.

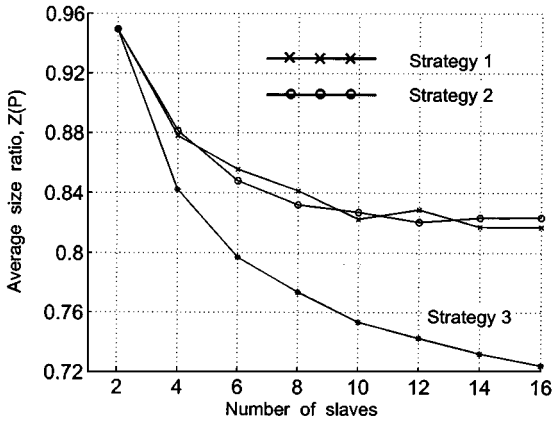

Fig. 6. Average *size ratio* versus number of slaves plots.

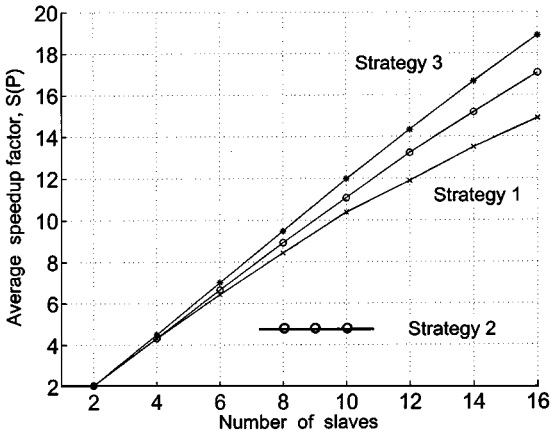

Fig. 7. Average *speedup factor* versus number of slaves plots.

VIII. Conclusions

In general, when a cluster of machines is running in parallel to solve a branch and bound problem, the branch and bound problem must be partitioned and distributed to the machines. In most cases, the aim is to achieve a high *speedup factor*. The best strategy for partitioning the problems differ from one type of problem to another. In this paper, we consider the design of discrete-coefficient FIR filters using the mixed integer linear programming technique. Three strategies for partitioning the problems are investigated. All three strategies offer significant reduction in the turnaround time for the design of filters with discrete coefficients. In addition, Strategy 3), which makes partitioning based on the smallest objective function value, produces the highest *speedup factor*. Our machines are interconnected through a reasonably high-speed communication link. We notice that Strategy 2), which makes partitioning based on the shortest branch length from the root, has a higher *hardware utilization*, implying a lower data transfer between machines. Thus, Strategy 2) may produce the highest speedup factor if the machines running in parallel are interconnected by a very slow communication link.

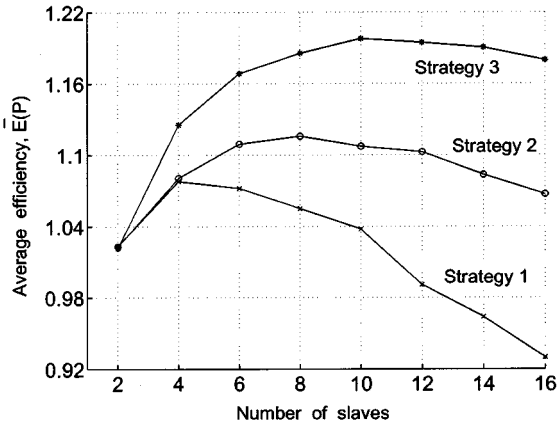

Fig. 8. Average *efficiency* versus number of slaves plots.

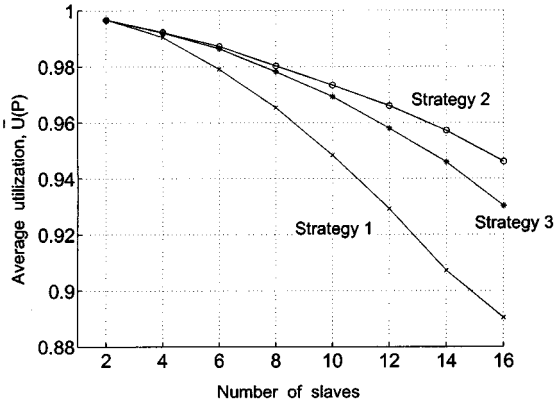

Fig. 9. Average *utilization* versus number of slaves plots.

Appendix

Let ω be the frequency variable, and let the frequency response of the filter be $H(\omega)$. $H(\omega)$ is a function of the filters coefficient values. Let $D(\omega)$ be the desired frequency response weighting function, and let $k(\omega)$ be the desired frequency response ripple magnitude weighting function. Let b and δ be the gain variable and ripple variable, respectively. A linear programming problem may be formulated as follows:

$$H(\omega) - bD(\omega) \leq \delta k(\omega)$$

$$H(\omega) - bD(\omega) \geq -\delta k(\omega)$$

$$b \geq b_l$$

$$b \leq b_u$$

$$\text{minimize } f = \delta - \alpha b.$$

The value of α in the objective function may be determined by using any one of the methods reported in [2].

REFERENCES

[1] D. C. Munson, "On finite wordlength FIR filter design," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-29, p. 329, Apr. 1981.

[2] Y. C. Lim, "Design of discrete-coefficient-value linear phase FIR filters with optimum normalized peak ripple magnitude," *IEEE Trans. Circuits Syst.*, vol. 37, pp. 1480–1486, Dec. 1990.

[3] D. M. Kodek, "Design of optimal finite wordlength FIR filters using integer programming techniques," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-28, pp. 304–308, June 1980.

[4] Y. C. Lim, S. R. Parker, and A. G. Constantinides, "Finite word length FIR filter design using integer programming over a discrete coefficient space," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-30, pp. 661–664, Aug. 1982.

[5] Y. C. Lim and S. R. Parker, "FIR filter design over a discrete powers-of-two coefficient space," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 583–591, June 1983.

[6] Y. C. Lim, "Efficient special purpose linear programming for FIR filter design," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. ASSP-31, pp. 963–968, Aug. 1983.

[7] D. M. Kodek and K. Steiglitz, "Comparison of optimal and local search methods for designing finite wordlength FIR digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-28, pp. 28–32, Jan. 1981.

[8] R. M. Karp and Y. Zhang, "Randomized parallel algorithms for backtrack search and branch-and-bound computation," *J. Assoc. Comput. Mach.*, vol. 40, pp. 765–789, July 1993.

[9] M. K. Yang and C. R. Das, "Evaluation of a parallel branch-and-bound algorithm on a class of multiprocessors," *IEEE Trans. Paral. Distrib. Syst.*, vol. 5, pp. 74–86, Jan. 1994.

[10] G. K. Kudva and J. F. Pekny, "DCABB: A distributed control architecture for branch and bound calculations," *Comput. Chem. Eng.*, vol. 19, pp. 847–865, June 1995.

[11] A. Taudes and T. Netousek, "Implementing branch-and-bound algorithms on a cluster of workstations—A survey, some new results and open problems," in *Proc. Workshop Paral. Algo. Transput. Optimiz.: Paral. Comput. Math. Optimiz.*, 1990, pp. 79–102.

[12] R. M. Wiegers, "Parallel branch-and-bound for mixed integer programming with monotone increasing speedup," in *Proc. EUROSIM Conf. Massively Paral. Process. Appl. Develop.* , 1994, pp. 513–520.

[13] H. J. Oh and Y. H. Lee, "Design of discrete coefficient FIR and IIR digital filters with prefilter-equalizer structure using linear programming," *IEEE Trans. Circuits Syst. II*, vol. 47, pp. 562–565, June 2000.

[14] S. Samadi, H. Iwakura, and A. Nishihara, "Multiplierless and hierarchical structures for maximally flat half-band FIR filters," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 1225–1230, Sept. 1999.

[15] B. W. Wah, Y. Shang, and Z. Wu, "Discrete Lagrangian methods for optimizing the design of multiplierless QMF banks," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 1179–1191, Sept. 1999.

[16] C. L. Chen and A. N. Willson, "A Trellis search algorithm for the design of FIR filters with signed-powers-of-two coefficients," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 29–39, Jan. 1999.

[17] J. S. Mao, S. C. Chan, W. Liu, and K. L. Ho, "Design and multiplier-less implementation of a class of two-channel PR FIR filterbanks and wavelets with low system delay," *IEEE Trans. Signal Processing*, vol. 48, pp. 3379–3394, Dec. 2000.

[18] S. Samadi, Y. Igarashi, and H. Iwakura, "Design and multiplierless realization of maximally flat FIR digital Hilbert transformers," *IEEE Trans. Signal Processing*, vol. 47, pp. 1946–1953, July 1999.

[19] J. H. Lee and D. C. Tang, "Optimal design of two-channel nonuniform-division FIR filter banks with -1, 0, and $+1$ coefficients," *IEEE Trans. Signal Processing*, vol. 47, pp. 422–432, Feb. 1999.

[20] L. D. Milic and M. D. Lutovac, "Design of multiplierless elliptic IIR filters with a small quantization error," *IEEE Trans. Signal Processing*, vol. 47, pp. 469–479, Feb. 1999.

[21] Y. C. Lim, R. Yang, D. Li, and J. Song, "Signed power-of-two term allocation scheme for the design of digital filters," *IEEE Trans. Circuits Syst. II*, vol. 46, pp. 577–584, May 1999.

[22] Y. C. Lim, "Frequency response masking approach for the synthesis of sharp linear phase digital filters," *IEEE Trans. Circuits Syst.*, vol. CAS-33, pp. 357–364, Apr. 1986.

[23] Y. C. Lim and B. Farhang-Boroujeni, "Fast filter bank FFB," *IEEE Trans. Circuits Syst. II*, vol. 39, pp. 316–318, May 1992.

[24] Y. C. Lim and Y. Lian, "The optimum design of one- and two-dimensional FIR filters using the frequency response masking technique," *IEEE Trans. Circuits Syst. II*, vol. 40, pp. 88–95, Feb. 1993.

[25] ——, "Frequency-response masking approach for digital filter design: Complexity reduction via masking filter factorization," *IEEE Trans. Circuits Syst. II*, vol. 41, pp. 518–525, Aug. 1994.

[26] Y. C. Lim and S. H. Low, "Frequency-response masking approach for the synthesis of sharp two-dimensional diamond-shaped filters," *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 1573–1584, Dec. 1998.

[27] Y. C. Lim and A. G. Constantinides, "New integer programming scheme for nonrecursive digital filter," *Electron. Lett.*, vol. 15, no. 25, pp. 812–813, Dec. 1979.

[28] R. S. Garfinkel and G. L. Nemhauser, *Integer Programming*. New York: Wiley, 1972.

[29] Y. C. Lim and S. R. Parker, "Discrete coefficient FIR digital filter design based upon an LMS criteria," *IEEE Trans. Circuits Syst.*, vol. CAS-30, pp. 723–739, Oct. 1983.

Yong Ching Lim (S'80–M'80–SM'92–F'00) received the A.C.G.I. and B.Sc. degrees in 1977 and the D.I.C. and Ph.D. degrees in 1980, all in electrical engineering, from Imperial College, University of London, U.K.

From 1980 to 1982, he was a National Research Council Research Associate with the Naval Postgraduate School, Monterey, CA. Since 1982, he has been with the Department of Electrical Engineering, National University of Singapore, where he is currently a Professor. His research interests include digital signal processing and VLSI circuits and systems design. He is currently serving as an Associate Editor for *Chinese Journal of Electronics*. He also served as an Associate Editor for *Circuits, Systems and Signal Processing* from 1993 to 2000.

Dr. Lim was selected to receive the 1996 IEEE Circuits and Systems Society's Guillemin-Cauer Award, the 1990 IREE (Australia) Norman Hayes Award, the 1977 IEE (U.K.) Prize, and the 1974–-1977 Siemens Memorial (Imperial College) Award. He is currently serving as a Distinguished Lecturer for the IEEE Circuits and Systems Society. He served as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS from 1991 to 1993 and from 1999 to 2001. He served as the Chairman of the DSP Technical Committee of the IEEE Circuits and Systems Society from 1998 to 2000 and with the Technical Program Committee's DSP Track as the Chairman for ISCAS' 1997 and ISCAS' 2000 and as a Co-chairman for ISCAS' 1999. He is a member of Eta Kappa Nu.

Y. Sun received the B.Eng. degree in computer engineering in 1992 from the University of Science and Technology, Beijing, China. He received the M.Eng. degree in electrical engineering in 1996 from the National University of Singapore (NUS).

From 1998 to 1999, he served as a Research Engineer with the Department of Electrical Engineering of NUS. His research interests include digital signal processing and digital image processing. He is currently serving as an engineer with Silicomp Asia Pte Ltd.

Ya Jun Yu (S'98) received both the B.Sc. and M.Eng. degrees in biomedical engineering from Zhejiang University, Zhejiang, China, in 1994 and 1997, respectively. She is pursuing the Ph.D. degree at the National University of Singapore (NUS).

From 1997 to 1998, she was a teaching assistant with Zhejiang University. She joined the Department of Electrical and Computer Engineering, NUS, as a Post Master Fellow in 1998. Her research interests include digital signal processing and VLSI circuits and systems design.