Ant Colony Optimal Algorithm: Fast Ants on the Optical Pipelined R-Mesh

Ken D. Nguyen and Anu G. Bourgeois

Department of Computer Science
Georgia State University
P.O. Box 3994
Atlanta, Georgia 30302-3994
Email: {knguyen, abourgeois}@cs.gsu.edu

Abstract

In this paper, we demonstrate how to implement and improve two Ant Colony Optimization (ACO) algorithms on the Optical Pipelined Reconfigurable Mesh (PR-Mesh): the generic ACO and the Fast Ant Colony Optimization (FACO) algorithm. The run-time complexity of our improved generic ACO algorithm, with x generations each generation having m ants, on an nxn PR-Mesh is $O((x \cdot m + n)logn)$, which outperforms the currently best known electrical model implemented in [3] with run-time complexity of $O(x \cdot (m + n)logn)$.

Our FACO algorithm on PR-Mesh yields $O(((\frac{z}{n \cdot log^2 n}) + \frac{n}{logn}) \cdot loglogn)$ run-time complexity for n^2 jobs while the existing FACO algorithm on the electrical model yields a run-time complexity of $O((z + n)log^* n)$ but can only handle $log^2 n$ jobs, where z is the total number of ants from all generations.

In addition, we propose a theoretical FACO algorithm on a three dimensional PR-Mesh solving n^2 jobs in $O(x \cdot (m + n) \cdot logn)$ time.

I. Introduction

The Ant Colony Optimization (ACO) algorithm is a greedy approach. It has been used to solve many combinatorial NP problems such as the Traveling Salesman Problems (TSP) [1], the Flow-Shop problems [2], the Single Machine Total Tardiness problem (SMTTP) [3], the Resource Constrained Project Scheduling problem [4], the Quadratic Assignment problem (QAP) [5] [6], graph coloring [7], and routing in communication networks [8] [9], etc.

The ACO algorithm was introduced by Marco Dorigo [10] in his dissertation in 1992 and is described by the behavior of ants finding food. In general, ants from a colony will dispatch searching for food in any random path. Each ant will leave some pheromone on its path to, and from, the food source. The amount of pheromone can be different depending on the goodness of the food found. A portion of pheromone on the path will be evaporated after a period of time. All following ants leaving the colony will follow the path(s) that has the strongest pheromone. A generation of ants is defined by the order in which they are leaving the colony. All ants in a generation are working independently. Since the ants keep following the strongest pheromone path(s), only the best found path(s) will remain. Because of the nature of the ACO algorithm, it is best to implement the algorithm on a parallel computer model, where each ant can be simulated by a processing element (PE) and the pheromone values on the path(s) are represented by a two dimensional matrix. There are many approaches to implement the ACO. For example, Delisle P. et al. [17] implemented the ACO on OpenMP machines with a run-time complexity of $O(n^2)$, where n is the number of jobs, Merkle and Middendorf [3] implemented the ACO algorithm on the electrical Reconfigurable Mesh (R-Mesh) with a run-time complexity of $O(x \cdot (m+n) \cdot log^* n)$, where x is the number of generations and each generation has m ants. Merkle and Middendorf's implementation is currently the fastest among all implementations due to the characteristics of the reconfigurable mesh. However, all ACO algorithm implementations on electrical models are constrained by the communication delays between processing elements.

In this paper, we present several techniques to implement the ACO algorithm and its fast version FACO on the Optical Pipelined Reconfigurable Mesh (PR-Mesh) model efficiently and effectively. With our new approach on the PR-Mesh model, the run time complexity of FACO algorithm is improved significantly. The FACO algorithm can be scaled-up, scaled-down or extending to a three dimensioned PR-Mesh depending on the number of jobs and available resources.

The R-Mesh model Merkle and Middendorf used and the PR-Mesh model utilized in this work belong to the same architecture class called Reconfigurable Architectures. Traditionally, a reconfigurable architecture is one that can dynamically alter the structure connecting its components at every step of a computation leading to a change in its functionalities and capabilities.

This paper is organized as follows: Section II describes the Reconfigurable Mesh models and their operations, Section III describes the existing ACO and FACO algorithms and their implementations on an R-Mesh, Sections IV and V describe our implementations of the ACO and FACO algorithms on the PR-Mesh model, Sections VI and VII describe new techniques to improve the performance and extend the capability of the FACO algorithm, and the last section summarizes our work.

II. Reconfigurable Mesh Models

The common characteristic of all reconfigurable models is the capability to reconfigure themselves at every step of computation. All components connected to a bus can receive data in constant time. The feature that leads to different reconfigurable models is the types of the buses connecting the components in each model. The model using electrical buses can configure cycles while the one using optical buses cannot. On the other hand, the model using optical buses can transmit multiple messages to multiple destinations on a single bus simultaneously, while the other cannot. The following sections describe more about these two models.

A. Optical Pipelined Reconfigurable Mesh

A optical PR-Mesh [12] is a two dimensional grid of processors, in which each processor has four ports connected by optical buses. Eight buses connect the four ports of a processor to its neighbors as in Figure 1. Each processor controls a set of local switches that allows the processor to segment or fuse the buses at each interconnection. The local fusing configures a linear bus connecting neighboring processors. Each such linear bus exactly resembles a *Linear Array with a Reconfigurable Bus Systems* (LARPBS) [14].

An LARPBS is an array of N processors $p_0, p_1, \ldots, p_{n-1}$ connected by an optical bus. The optical bus is constructed from three identical waveguides: the $message$, $select$ and $reference$ waveguides. The waveguides are divided into two segments: receiving segment and transmitting segment. The $message$

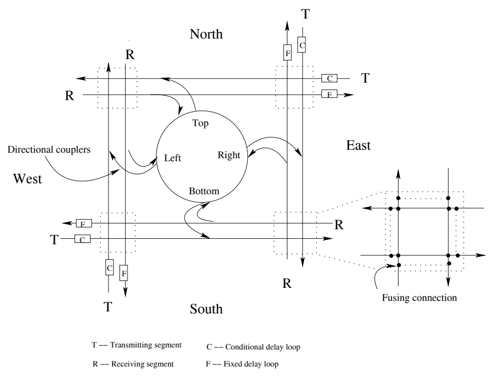

Fig. 1. PR-Mesh processor connections

waveguide is used for sending data, the $select$ and $reference$ waveguides are used for sending addressing information. Each processor p_i is connected to the optical bus with two directional couplers. One coupler is used for writing data on the transmitting segment (upper segment) of the bus and the other one is used for reading the data from the receiving segment (lower segment) of the bus. Thus, the LARPBS uses six couplers for each processor p_i to connect to the three waveguides. Two consecutive processors are connected by an optical fiber segment of one unit pulse-length,Δ, as in Figure 2,(for simplicity, the figure omits the data waveguide, which resembles the reference waveguide). The optical signals are propagated unidirectional from left to right on the transmitting segment and from right to left on the receiving segment.

The LARPBS has a set of conditional-delay switches on the transmitting segment of the *select* bus and a set of fixed-delay switches on the receiving segment of the *reference* bus. The LARPBS uses the *coincident pulse technique* [15] to route messages by manipulating the $select$ and $reference$ signals, through the switches, on separate bus so that they will coincide at the desired destination processor(s). Each processor has a *select frame* and a *reference frame* of N-bit each, which can be used to send a message by injecting, up to N slots, its pulse signals into the frames. When processor p_i detect a coincidence of the $select$ and $reference$ pulses on the receiving segment, it reads the data frame. Any subsequence coincidences at processor p_i in the same bus cycle will be ignored.

In addition, the LARPBS contains a set of segment switches on the transmitting and receiving segment of the bus. When all the segment switches are set to *straight*, the bus system operates exactly like a regular pipelined bus system. Setting the switches at p_i to *cross*, the bus is segmented into two separate bus systems, one consists of $p_0, p_1 \ldots, p_i$ and the other consists of $p_{i+1}, p_{i+2}, \ldots, p_{n-1}$.

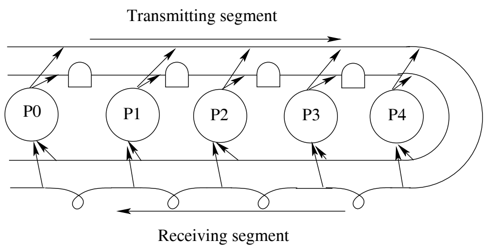

Fig. 2. A five-processor LARPBS model

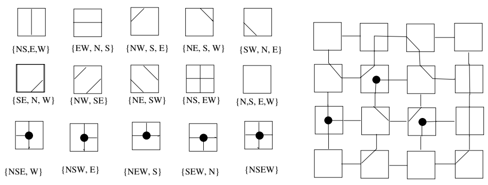

Fig. 3. local possible connections of ports within a PE(right), R-Mesh (left)

B. Electrical Reconfigurable Mesh Description

An electrical Reconfigurable Mesh (R-Mesh) [11] is a two dimensional grid of PEs connected by electrical buses. Each processor can locally configure any or all of its ports to the connecting buses, as in Figure 3, within one time step. Signals propagating on a bus are nondirectional. Thus, the R-Mesh can form cycles. Due to the nature of the bus, to avoid collision, the COMMON write rule is used for concurrent write operations.

III. ACO and FACO on R-Mesh

Generally, permutation problems require one to find the optimal permutation from a set of n given items. These items represent the distances between cities as in the Travelling Salesman problems (TSP) [1] or the jobs as in Single Machine Total (Weighted) Tardiness problems (SMTWTP) [3]. A generic ant colony optimization algorithm uses an nxn pheromone matrix to find the permutations of the items. In this section we describe Middendorf's implementation of a generic ant algorithm (ACO) and its fast version (FACO) on the R-Mesh model.

A. ACO algorithm on R-Mesh

In [3], Middendorf embedded the nxn pheromone matrix M into an nxn R-Mesh. Each processor p_{ij} contains a pheromone value τ_{ij} and a heuristic value $\eta_{ij}, (i, j \in [1, n])$. The ants are pipelined through the R-Mesh from the first row. Each ant selects an unselected item from row i as it goes through the R-Mesh. The information about which items have not been selected is kept in a selectable set S in the ant's memory. The next item is always chosen from S according to the Pseudo-Random-Proportional Action Choice Rule [3], using probability q_0, where $0 \leq q_0 \leq 1$ is a parameter for the algorithm. An ant chooses item j that has not been selected so far which maximizes

$$\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} \tag{1}$$

where α nd β are constants that determine the relative influence of the pheromone value on the ant's decision in selecting the next item. The probability of item to be selected is $1 - q_0$. This probability is computed as q_{ij}

$$q_{ij} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{h \in S} \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}} \tag{2}$$

(Most often, heuristic values η_{ij} are not used). An ant stops its journey when it reaches row n, at which point its solution is found. On a system of m ants, the last ant reaches row i after $n + (m - 1)$ steps. When all ants have found their solutions, their solutions will be compared to existing solutions in order to determine which ants can update the solutions. The selected ants update their solutions and the pheromone on their paths. This completes an ant generation. The algorithm will run for x generations, $x \geq 0$.

In general, an ant selects an item in row i as follows: when the ant is in row $i - 1$, p$_{i-1,j}$ knows whether item j has been selected in one of the rows $1 \ldots (i - 1)$ or not, p$_{i-1,j}$ sends this information to $p_{i,j}$ as the ant moves to row i. This step can be done in time $O(1)$ by configuring $p_{i-1,j}$ $\{\overline{NS}$, E, W$\}$. Next, the ant selects an item in row i by computing the prefix-sums of $\tau_{ij} \cdot \eta_{ij}$, where $\tau_{ij} \cdot \eta_{ij} = \tau_{ij_1} \cdot \eta_{ij_1} + \tau_{ij_2} \cdot \eta_{ij_2} + \cdots + \tau_{ij_k} \cdot \eta_{ij_k}, k \in [1, n - (i - 1)]$, which is the prefix-sums of $\tau_{ij} \cdot \eta_{ij}$ of all elements in $S, S = \{j_1, j_2, \ldots, j_{n-(i-1)}\}, j_1 < j_2 < \cdots < j_{n-(i-1)}$, across row i. The first PE in row i then randomly chooses a value $z, z \in [0, \sum_{j \in S} \tau_{i,j}^{\alpha} \cdot \eta_{i,j}^{\beta})$, and broadcasts z to all PEs in the row. $p_{i,j}$ is selected when $z \in [\sum_{l<j, l \in S} \tau_{ij}^{\alpha} \cdot \eta_{il}^{\beta}, \sum_{l \leq j, l \in S} \tau_{ij}^{\alpha} \cdot \eta_{il}^{\beta},)$. The prefix-sums can be done in time $O(logn)$ using binary tree reduction method [11](page 5), and the selected PE can be determined by comparing the prefix-sums of $p_{i-1,j}$ and $p_{i,j}$ in time $O(1)$. When all the ants in a generation have found their solutions, which ant is allowed to update the pheromone is determined. The selected ants store their solutions and the pheromone value is evaporated from every PE before the next generation of ants starts. The amount of pheromone evaporated is defined follow:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} \tag{3}$$

where ρ is any predefined factional parameter for pheromone evaporating function. To force ants from following generations to follow the near best found solutions, an elitist ant is designated to update the pheromone according to the best found solution in each generation.

Since each step of the algorithm can be carried out in constant time, except the prefix-sums computation, the algorithm run time is $O(x \cdot (m+n) \cdot logn)$, where x is the number of generations.

B. FACO algorithm on the R-Mesh

The fast version of ACO algorithm is the result of abandoning the generational pheromone update and simplifying the probability distribution. First, instead of updating the pheromone at the end of each generation, an ant is allowed to update the pheromone when some criteria is met [18]. In particular, an ant waits until the following $\lceil (m-1)/2 \rceil$ ants have found their solutions. The ant is allowed to update the pheromone only if its solution is better than the $m'-1, m' \leq m$, best solutions that have been found by the preceding $\lfloor (m-1)/2 \rfloor$ ants and the following $\lfloor (m-1)/2 \rfloor$ ants. If multiple ants with equal solutions, only the $(m/m')th$ ant is allowed to update. Evaporation is done when an ant updates the pheromone. Therefore, the running time of the updated ACO algorithm is $O((x \cdot m + n) \cdot logn)$ instead of $O(x \cdot (m+n) \cdot logn)$. Secondly, each item per row is assigned two bit-values h and l, where $h > l > 0$, for low and high probabilities of being selected next. These values are determined by a threshold function t. For each item $j \in S$, define

$$g(\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}) = \begin{cases} h & if\ \tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta} > t \\ l & otherwise \end{cases} \tag{4}$$

In p_{ij}, for each $j \in S$, a bit l_{ij} is set to 1 if $g(\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}) = l$, otherwise, a bit h_{ij} is set to 1. Next, the number of PEs with bit l set to one ,n_l, and the number of PEs with bit h set to one, n_h, in each row are determined. A random number $z, z \in [0, n_l \cdot l + n_h \cdot h)$, is selected as in the generic ACO algorithm. If $z \in [(p-1) \cdot l, p \cdot l)$, the p^{th} PE in the row with an l bit set to 1 is selected. If $z \in [n_l \cdot l + (p-1) \cdot h, n_l \cdot l + p.h)$ then the p^{th} PE in the row with an h bit set to 1 is selected. The t, h, and l values could be changed during the run of the algorithm.

Similarly, the Pseudo-Random-Proportional Action Choice Rule is modified as follows: let $t_\tau > 0$ be a pheromone threshold and $h_\tau > l_\tau > 0$. For each item $j \in S$ define

$$g(\tau_{ij}) = \begin{cases} h_\tau & if\ \tau_{ij}^{\alpha} > t_\tau \\ l_\tau & otherwise \end{cases} \tag{5}$$

an ant will select the item that maximizes the value of $g(\tau_{ij}) \cdot \eta_{ij}^{\beta}$.

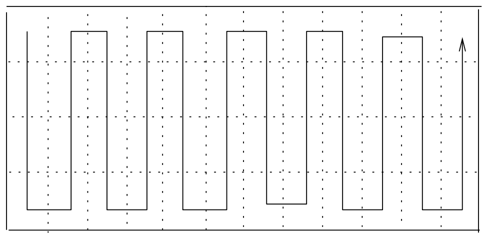

Fig. 4. Embedding a row of the pheromone matrix in a submesh of the R-mesh

C. Implementation of FACO algorithm on the R-Mesh and its Run Time

Middendorf proposed [3] to use an $nlog^2n$ x (n/log^2n) R-mesh. Then a row of pheromone matrix forms a log^2nx(n/log^2n) submesh as in Figure 4. The sums n_l and n_h can be computed in $O(log^*n)$ as stated in [16] and [3]. After selecting a random number z, the p^{th} PE with bit l set to one is selected. By [3] and [16], the sum of all l bits in the first $i \cdot log^2n$ columns, $i \in [1, (n/log^4n)]$, of all sub-meshes can be found in time $O(logn^*n)$. The p^{th} PE with l-bit is set is determined using the same technique. Therefore, the total time to select an item is $O(log^*n)$.

At every step, there will be at most $n + m/2 \leq \frac{3}{2}n$ solutions need to be stored. In order for an ant to update its solution, it must know the rank of its solution. Therefore, the stored solutions must be ranked by their goodness. The rank of the new solution can be computed in $O(log^*n)$ by comparing the solution to the stored solutions in parallel, and a bit is set when it is worse. Addition of the resulting bits gives the rank of the new solution. The pheromone can be updated in $O(1)$ as the ant updating the solution using the Formula 3.

Therefore, this FACO algorithm run-time complexity is $O((z+n) \cdot log^*n)$, where z is the number of ants on an $nlogn^2$ x n/log^2n R-mesh. This run time can only be achieved if the number of different weights is at most log^2n [3]. In the next section, we describe techniques to implement the ACO on PR-Mesh model.

IV. ACO algorithm on PR-Mesh (ACO-PR)

As stated in Section II the optical model cannot configure any cycles. To implement the ACO algorithm on a PR-Mesh, we have to ensure that none of the steps in the algorithm configures a cycle in the PR-Mesh. When an ant moves from row $i-1$ to row i, $p_{i-1,j}$ passes this information to $p_{i,j}$ using only column j and forms no cycles. $p_{i,j}$ broadcasts its new information to all PEs in row i, which does not form any cycle either. Next, the prefix-sums of row i is computed using binary tree reduction on row i, which also uses no cycle. The first PE in row i chooses the z value and broadcasts it to all PEs in row i, and $p_{i,j}$ determines whether it is selected or not using only row i for communication. The last step, when all ants have found their solutions, the new found solutions must be routed to the PEs that holding the previously found solutions to determine whether any ant has found a better solution. Let us assume the m best solutions found so far are positioned in the last row of the PR-Mesh. If we keep each ant in a column, i.e. ant $k, 1 \leq k \leq m$ moves from row 1 to row n using only column k, then ant k will store its solution to $p_{n,k}$ when it reaches row n. Obviously, the ACO algorithm can be implemented on a nxn PR-Mesh with run time $O(x \cdot (m+n) \cdot logn)$.

In the next section, we show how to determine the goodness of a solution and update the pheromone in constant time. Using that technique, each ant can update its solution and the pheromone immediately if its solution is better than previously found solutions. The run time will be reduced to $O((x \cdot m + n) \cdot logn)$.

V. FACO algorithm on PR-Mesh (FACO-PR)

The FACO algorithm can be simulated in the PR-Mesh as follows: instead of finding the sums of of l bits n_l and h bits h_l of each row, we compute the prefix-sums of these bits. The prefix-sums can be computed in time $O(1)$ [11](page 374). After computing the prefix-sums, the p^{th} PE with l bit or h bit set to 1 is the one has the prefix-sums equals to p. This information is obviously can be determined in time $O(1)$.

Updating the solutions in PR-Mesh is as follows: first, we use the last two rows of the PR-Mesh to store $\frac{3}{2}n$ possible solutions. These two rows are connected in a snake-like array, where the last PE of the mesh, $p_{n,n}$, is the head of the bus and the last PE of row (n-1), $p_{(n-1),n}$, is the tail of the bus, actually, we only use the lower half, $p_{n-1,0}, \ldots, p_{n-1,\lceil n/2 \rceil}$, of row (n-1). We call this bus the *solution bus*. Whenever an ant finds a solution, this solution is broadcasted to all the PEs in the *solution bus*. Each PE in the first $\frac{3}{2}n$ of the bus, starting from the bus head, holding a solution will set a bit to 1 if its solution is better than or equal to the new solution, otherwise set its bit to 0. The sum of these bits is the rank of the new solution. Each PE that set a bit to 0 (holding worse solution) in the previous step increases its rank by 1. Then all the solutions are routed to their locations based on the rank. The best solution will have the lowest rank. The sum of these bit can be added in $O(1)$ time [11](page 374). The routing can be done in $O(1)$ time [11](page 369).

Evaporating the pheromone is done similar to the original algorithm. The updating ant sends a signal to all PEs in its path to evaporate the pheromone. Since the goodness of each solution can be determined in constant time, we can allow the solution and the pheromone to be updated as soon as the solution emerges to make it available for $m-1$ ants in the pipeline. In addition, we only need store at most m solutions, which can be stored in the last row of the PR-Mesh.

Since each step of the FACO algorithm on PR-Mesh runs in $O(1)$ time, the algorithm yields $(z-1+n)$ steps or $O(z+n)$, for z is the total number of ants leaving the colony, which is better than the one running on the R-Mesh by $log^{*}n$ factor. $O(z+n)$ is the lower bound for ACO algorithm on the pipelined reconfigurable mesh, and this is not hard to verify. By the nature of the ACO algorithm, the decision of choosing the next move of an ant depends on the locations it has visited. It can not visit a location twice nor it can visit location i before visiting location $i-1$. Therefore, the ant takes exactly n steps to find its solution. In addition, an n states pipeline with z jobs take at least $n+(z-1)$ time slots to finish. Thus, the run time of the FACO algorithm on the PR-Mesh is optimal.

VI. FACO Algorithm on sub-PR-Meshes (FACO-PR-Sub)

Since the ACO algorithm practices the greedy approach, where each ant chooses the best possible path adjacent to its current location, the over all solution may not be optimal. In this section we present a divide and conquer approach, in which the solution will be the optimal of sub-problems and each sub-problem is solved by applying the FACO-PR algorithm.

A. Selection on PR sub-mesh

In the ACO algorithm, an ant chooses the next item by selecting an item in the next row that maximizes the pheromone and heuristic values $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$ (similarly for the ants in the FACO algorithm). If all the items are equally likely to be selected, the probability of this selection behavior can be represented as: $P = \prod_{j=0}^{n-1} \frac{1}{n-j} = \frac{1}{n!}$. To speedup the algorithm, we propose the following: First, we divide the nxn PR-Mesh into blocks (sub-mesh) of size $\frac{n}{logn} \times \frac{n}{logn}$, giving $log^2 n$ such blocks. The solution for each block is carried out the same as described in the FACO-PR algorithm. Next, the solution for all the blocks is determined using one of the following two methods. (i) sum the sub-solutions of all blocks in the same block-row, then find the maximum of all the block-rows (Max Row method). (ii) find the maximum sub-solutions of each block-column, then add all the maximum values to obtain a solution (Max Column method). The two methods share almost the same behavior. The selection probability for all

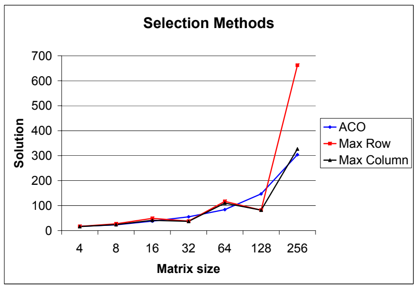

Fig. 5. Simulation of the original ACO selection, Max Row selection and Max Column selection Methods - The Max Column selection method resembles the original ACO selection method

the disjoint blocks is $P_b = \prod_{j=0}^{\frac{n}{logn}-1} \frac{1}{\frac{n}{logn}-j} = (\frac{logn}{n})!$. In addition, the probability of picking the best sub-solution block out of a block-column, or block-row, is $\frac{1}{logn}$, and $P_c = (\frac{1}{logn} \cdot \frac{1}{logn} \cdots \frac{1}{logn}) = (\frac{1}{logn})^{\frac{1}{logn}} = \frac{1}{(logn)^{logn}}$ for $logn$ block-columns. Therefore, the new selection probability is $P = P_b \cdot P_c = \frac{logn!}{(logn)^{logn} \cdot n!}$.

Since the solution of the problem closely depending on the distribution of the items, it is not clear which selection method, the original one or the proposing ones, yields better solution. Based on our simulation (see Figure 5) of the three selection methods, where the pheromone values are randomly generated and distributed, the Max Column method resembles the original selection method. Next, we will describe the Max Column method in details, (the Max Row Method is similar).

B. Applying Max-Column Selection Method on PR-Mesh

Ants used in the ACO models are control logics and often represented as iterations of running an algorithm; therefore, requiring more ants may not impose any physical restriction on the algorithm. Initially, we partition an nxn PR-Mesh into log^2n sub-meshes, or blocks, of size $\frac{n}{logn}$x$\frac{n}{logn}$. The nxn PR-Mesh M can be viewed as a $lognxlogn$ matrix, where each element is a $\frac{n}{logn}$x$\frac{n}{logn}$ sub-mesh as in Figure 6. Assuming we have at least z ants, $z \geq log^2n$. We designate the last row of the nxn PR-Mesh as the *solution bus* as in Section V. First we distribute z/log^2n ants to each block. The ants are pipelined through the sub-mesh in the same way as described in Section V. After an ant finds its solution in its block $b_{i,j}$, where $i, j \in [1, logn]$, the best of all the solutions found by these

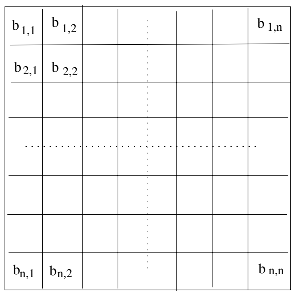

Fig. 6. An nxn PR-Mesh partitioned into $lognxlogn$ blocks

block-columns is determined. In order to find the best sub-solution of a column, all sub-solutions $s_1, s_2, \ldots, s_{logn}$ of block-columns are routed to the last row of block $b_{logn,j}$ of each block-column j, where $p_{n,i}$ is the destination of s_i.

Next, in each block $b_{logn,j}$ the solutions are broadcasted to all other PEs through the columns of the block. $p_{i,i}$ of the block broadcasts its solution along row i, any PE holding a worse solution sets its bit to 1. The bitwise-or over every column bit of the blocks is computed. A result zero of a column signifies the solution in that column is the best solution. Each one of these steps take $O(1)$ time. Next, the solution to the problem is determined by adding all the best sub-solutions. This solution is then routed to the *solution bus* for ranking and storing as described in the previous section. The run time of every step involved in this method is either constant or similar to the FACO-PR, except the summing of partial solutions. The sum of these $logn$ solutions can be computed in $O(loglogn)$ time using binary-tree reduction. It takes $O(\frac{n}{logn})$ time to fill the pipeline. Therefore, the run time of the FACO-PR-Sub is $O(((\frac{z}{log^2n}) + \frac{n}{logn}) \cdot loglogn)$.

VII. FACO Algorithm on 3D PR-Mesh

Combinatorial problems that have their weights changing dynamically during the execution of an algorithm and require iterating over the solutions many times for the solutions to converge to an optimal solution tend to use a large number of ants to solve. In this section, we describe a technique to extend the FACO-PR-Sub to multidimensional PR-Mesh to solve such problem effectively.

A. n Jobs FACO Algorithm on 3D PR-Mesh (FACO-PR 3D)

In general, if the number of ants used in the ACO algorithms is relatively large, the FACO can be extended to run on a k-dimensional PR-Mesh. In particular, we will

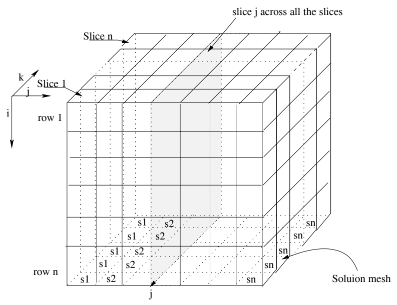

Fig. 7. An $nxnxn$ PR-Mesh

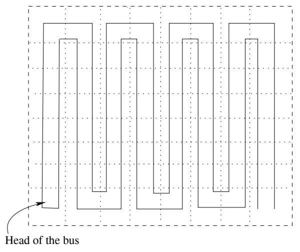

Fig. 8. An nxn PR-Mesh configured int a snake-like linear bus

describe a technique to extend the FACO to run on an $nxnxn$ PR-Mesh in logarithmic time.

A 3D PR-Mesh can be viewed as n of nxn PR-Meshes (n slices) connected by the optical buses in the third dimension k, where all the last rows of these slices form an nxn *solution mesh* (see Figure 7). Let row n^th of each slice holds the best n solutions found so far - we call these rows as *solution buses*. The FACO algorithm can be implemented as follows: First, let the ants pipelining through the slices as in the 2D PR-Mesh. After n steps, there will be n ants arriving to the last row with their solutions, one in every slice. For the first arriving n solutions, we must sort them by their ranks. This can be achieved by using Leighton's column sort algorithm on an nxn mesh, as described in [11] (page 377), in $O(1)$ time. After sorting, these solutions are broadcasted to all other slices. On every subsequence step, the ranks of n new solutions are determined and routed to their positions in the *solution buses*. To extract the best n solutions we perform the following stages.

First, the solutions in $p_{n,j,1}$ are swapped with the solutions in $p_{n,j,n}$ if j is even. Second, $p_{n,j,1}, p_{n,j,2}, \dots, p_{n,j,n}$ are connected to form a bus going across all slices. $p_{n,j,k}$ is the head of the bus if j is odd, otherwise, $p_{n,j,n}$ is the head of the bus. All PEs holding new solutions and head PEs are set to active. All active PEs have their solutions ranked and routed to their positions. At this moment, all the new solutions are packed to the head of the buses. Next, the *solution mesh* is configured into a snake-like bus,(see Figure 8), where $p_{n,1,1}$ is the head of the bus. The best n solutions are routed to the head of the bus by computing the prefix-sum of the inactive PEs and letting all active PEs send their solutions to the PEs of distances equal to their prefix-sums toward the head of the bus. The final step is to propagate the best n solutions to each *solution bus* in the *solution mesh*. This is done in two steps:$p_{n,1,k}$ sends its solution to $p_{n,k,k}$ and $p_{n,k,k}$ broadcasts its value across slices $p_{n,k,1}, p_{n,k,2}, \dots, p_{n,k,n}$. All of these steps run in $O(1)$ time.

The pheromone is updated in each slice similar to the earlier described algorithm. $p_{n,k,k}$ updates slice k if the its solution is new. Next, n^2 buses connecting PEs in the first slice to the last slice are formed to broadcast the new pheromone values to all the slices.

As a result, the run time for this algorithm is $O((\frac{z}{n}) + n)$, where z is the total number of ants. Using the same method described in Section VI, the run time of this algorithm can be lowered to $O(((\frac{z}{n \cdot log^2 n}) + \frac{n}{logn}) \cdot loglogn)$.

B. nxn Jobs Ant Colony Algorithm on 3D PR-Mesh (FACO-PR 3Dx)

To solve a problem with n^2 jobs using the PR-Mesh as described in Section V we will need an n^2xn^2 PR-Mesh to store the pheromone values. In this section, we will use the result of the previous section to solve this problem efficiently. Let the n^2 jobs be distributed along the j and k directions of the 3D-PR-Mesh such that each slice containing n jobs. The solutions are computed in both directions i and j as in Figure 7. This algorithm requires at least $n + 1$ ants. Each ant steps down from the first row to the last row of its slice and selects an item similar to the the general ACO described in Section III-A simultaneously. We call the ant from the first slice the *leader ant*. When the *leader ant* is on row i and selecting item j, the weight of that item w_j is sent to the crossed slice j (containing $p_{1,j,1}, p_{2,j,1}, \dots, p_{n,j,n}$) . This is possible since each PE has many separate buses, (see Figure 1). For clarity, we can visualize the first slice as an additional nxn mesh attaching to the $nxnxn$ cube. The solution of slice j is the sum of its solution and the weight w_j of the *leader ant*. After n steps, the *leader ant* tailors its solution by summing the solutions of all n crossed slices in $O(logn)$ time. The pheromone is updated on each crossed slice as described in previous sections.

At the end of each generation, the maximum values τ_{ij} and $\eta_{ij}, (i, j \in [1, n])$ must be routed to the first slice for the *leader ant* to work properly. This step is done as follows: First, each $p_{i,j}$ on slice 1 becomes the head of a bus connecting all other $p_{i,j}$ from all other slices. Next, the max values are computed and routed to the heads of

TABLE I. Algorithms' Run times

Algorithm	Run Time
ACO	$O(x \cdot (m+n) \cdot logn)$
ACO-PR	$O((z+n) \cdot logn)$
FACO	$O((z+n) \cdot log^{*}n)$
FACO-PR	$O(z+n)$
FACO-PR-Sub	$O((\frac{z}{log^2n} + \frac{n}{logn}) \cdot loglogn)$
FACO-PR 3D	$O((\frac{z}{n \cdot log^2n} + \frac{n}{logn}) \cdot loglogn)$
FACO-PR 3Dx	$O(x \cdot (m+n) \cdot logn)$

TABLE II. Algorithms' Characteristics

Algorithms	Number of ants	Jobs	Max. Wts.	Mesh Size
ACO	$x \cdot m$	n	n^2	nxn
ACO-PR	$x \cdot m$	n	n^2	nxn
FACO	$x \cdot m$	n	log^2n	$nlog^2nx\frac{n}{log^2n}$
FACO-PR	$x \cdot m$	n	n^2	nxn
FACO-PR-Sub	$\geq \frac{log^2n}{x}$	n	n^2	nxn
FACO-PR 3D	$\geq \frac{n \cdot log^2n}{x}$	n	n^2	$nxnxn$
FACO-PR 3Dx	$\geq n+1$	n^2	n^3	$nxnxn$

the buses. Each bus works independently. These steps can be done in $O(1)$ time [11] (page 376).

The total run time to compute the first solution is $O(n+logn)$. The run time of the 3D PR-Mesh is $O(x \cdot (m+n) \cdot logn)$, where x is the number of generations each having m ants. This run time can be further reduced by applying the partitioning technique in Section VI to the algorithm.

VIII. Conclusion

In this paper, we show that the Ant Colony Optimization (ACO) and its simplified version Fast Ant Colony Optimization (FACO) algorithms with z ants can be simulated on the Optical Pipelined Reconfigurable Mesh (PR-Mesh) with better time complexity than the electrical reconfigurable mesh. Unlike the electrical model, which restricts the number of different weights to be at most log^2n, the optical model imposes no restriction on the weights. In particular, we present techniques to implement the FACO algorithm on the PR-Mesh with run time $O((\frac{z}{log^2n} + \frac{n}{logn}) \cdot loglogn)$ and $O((\frac{z}{n \cdot log^2n} + \frac{n}{logn}) \cdot loglogn)$ on 2D PR-Mesh and 3D-PR-Mesh (FACO-PR-Sub and FACO-PR 3D), and a version of the FACO algorithm with n^2 jobs on a 3D-PR-Mesh (FACO-PR 3Dx) in $O(x \cdot (m+n) \cdot logn)$. The run-time complexity comparison of the two models is given in Table I, and the characteristics of the algorithms on these models are given in Table II, (note: x is the number of generations, m is the number of ants per generation, $z = x \cdot m$, and italicized algorithms are on the electrical reconfigurable mesh).

References

[1] M. Dorigo and L. M. Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem," IEEE trans. on Evolutionary Comp. Vol. 1: 53-66, 1997.

[2] T. Stützle, " An ant Approach for the Flow Shop Problem," Proc. 6th European Congress on Intellegent Techniques and Soft Computing, Verlag Mainz: Aachen, Vol. 3: 1560 - 1564, 1998.

[3] D. Merkle and M. Middendorf, " Fast Ant Colony Optimization on Runtime Reconfigurable Processor Arrays,". Journal of Genetic Programming and Evolvable Machines, Vol. 3: 345-361, 2002.

[4] D. Merkle, M. Middendorf, and H. Schmeck, " Ant Colony Optimization for Resource-Constrained Project Scheduling," IEEE trans. on Evolutionary Comp. Vol. 6: 333 - 346, 2002.

[5] L. M. Gambardella, E. Taillard, and M. Dorigo, "Ant Colonies for the Quadratic Assignment Problem," Journal of the Operational Research Society, Vol. 50:167 - 176, 1999.

[6] V. Maniezzo, A. Colorni, and M. Dorigo, "The Ant System Applied to the Quaratic Assignment Problem," IEEE Trans. Knowledge and Data Engineering, Vol. 11: 769 - 778, 1999.

[7] D. Costa and A. Hertz, " Ants can colour graphs," Journal of the Operational Research Society, Vol. 48:295-305, 1997.

[8] G. Di Caro and M. Dorigo, " Ant colonies for adaptive routing in packet-switchined communications networks," Proceeding of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature, Vol. 1498:673-682, 1998.

[9] R. Schoonderwoerd, O. Holland, J. Bruten, and L. Rothkrantz, " Ant-based load balancing in telecommunication networks," Adaptive behavior Vol. 5:169-207, 1996.

[10] M. Dorigo, Optimization, learning and natural algorithm, Ph.D Thesis, DEI, Politecnico di Milanio, Italy, 1992.

[11] R. Vaidyanathan and J. Trahan, "Dynamic Reconfiguration - Architectures and Algorithms," Kluwer Academic/Plenum Publishers, ISBN 0-309-48189-8, 2003.

[12] J L. Trahan, A. G. Bourgeois, and R. Vaidyanathan, "Tighter and Broader Complexity Results for Reconfigurable Models," Parallel Processing Letters, Vol. 8: 271-282, 1998.

[13] M. Middendorf, "Bit-summation on the Reconfigurable Mesh," Parallel and Distributed Computing, Proc. of the 11 IPPS/SPDP'99 Workshops, 6th Reconfigurable Architectures Workshop RAW-99, J. Rolim et al. (eds.), Springer: Berlin, LNCS, Vol. 1586:625 - 633, 1999.

[14] Y. Pan and K. Li, " Linear Array with a Reconfigurable Pipelined Bus Systems: Concepts and Applications," Information Science 106:237 - 258, (1998).

[15] C. Qiao and R. Melhem, " Time-Division Optical Communication in Multiprocessor Arrays," IEEE Trans. Comput. 42:577 - 590, 1993.

[16] K. Nakano."Prefix-sums Algorithms on Reconfigurable Mesh," Par. Process. Letter, 5:23 - 25,1995.

[17] Delisle P., Krajecki M., Gravel M., Gagn C., "Parallel implementation of an ant colony optimization metaheuristic with OpenMP," International Conference on Parallel Architectures and Compilation Techniques, Proceedings of the 3rd European Workshop on OpenMP (EWOMP01), September 2001, Barcelone, Espagne.

[18] V. Maniezzo, " Exact and approximate nonditerministic tree-search procedures for the quaratic assignment problem," INFORMS Journal on Computing, Vol. 11:358-368, 1999.

[19] T. Stützle, H.H. Hoos,"The MAX-MIN ant system and local search for the traveling salesman problem," Proceedings of the IEEE International Conference on Evolutionary Computation(ICEC'97): 309314, 1997.