Basic Operations on a Partitioned Optical Passive Stars Network with Large Group Size *

Amitava Datta and Subbiah Soundaralakshmi

Department of Computer Science & Software Engineering
The University of Western Australia
Perth, WA 6009
Australia
email : {datta,laxmi}@cs.uwa.edu.au

Abstract. In a *Partitioned Optical Passive Stars (POPS)* network, $n = dg$ processors are divided into g groups of d processors each and such a POPS network is denoted by $POPS(d, g)$. There is an optical passive star (OPS) coupler between every pair of groups. Each OPS coupler can receive an optical signal from any one of its source nodes and broadcast the signal to all the destination nodes. The time needed to perform this receive and broadcast is referred to as a time *slot* and the complexity of an algorithm using the POPS network is measured in terms of number of slots it uses. Since a $POPS(d, g)$ requires g^2 couplers, it is unlikely that in a practical system the number of couplers will be more than the number of processors. In other words, in most practical systems, the group size d will be greater than the number of groups g, i.e., $d > \sqrt{n} > g$. Hence, it is important to design fast algorithms for basic operations on such POPS networks with large group sizes. We present several fast algorithms for basic arithmetic operations on $POPS(d, g)$s such that $d > \sqrt{n} > g$. Our algorithms require significantly less number of slots for these operations compared to the best known algorithms for these problems designed by Sahni [8].

keywords: optical computing, partitioned optical passive stars network, arithmetic operations

1 Introduction

The *Partitioned Optical Passive Stars (POPS)* network was proposed in [3–5, 7] as a fast optical interconnection network for a multiprocessor system. The POPS network uses multiple optical passive star (OPS) couplers to construct a flexible interconnection topology. Each OPS coupler can receive an optical signal from any one of its source nodes and broadcast the received signal to all of its destination nodes. The time needed to perform this receive and broadcast is

* This research is partially supported by a University of Western Australia research grant.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2329, pp. 306–315, 2002.

referred to as a *slot*. A POPS network is divided into g groups of d processors each. Hence, the total number of processors in the network is $n = dg$. We denote such a POPS network as a $POPS(d, g)$ network.

Berthomé and Ferreira [1] have shown that POPS networks can be modeled by directed stack-complete graphs with loops. This is used to obtain optimal embedding of rings and de Bruijn graphs into POPS networks. Berthomé *et al.* [2] have also shown how to embed tori on POPS networks. Gravenstreter and Melhem [4] have shown the embedding of rings and tori into POPS networks.

Sahni [8] has shown simulations of hypercubes and mesh-connected computers using POPS networks. Sahni [8] has also presented algorithms for several fundamental operations like data sum, prefix sum, rank, adjacent sum, consecutive sum, concentrate, distribute and generalize. Though it is possible to solve these problems by using existing algorithms on hypercubes, the algorithms presented by Sahni [8] improve upon the complexities of the simulated hypercube algorithms. In another paper, Sahni [9] has presented fast algorithms for matrix multiplication, data permutations and BPC permutations on the POPS network. One of the main results in the paper by Sahni [8] is the simulation of an SIMD hypercube by a $POPS(d, g)$ network. Sahni [8] has shown that an n processor $POPS(d, g)$ can simulate every move of an n processor SIMD hypercube using one slot when $d = 1$ and $2\lceil d/g \rceil$ slots when $d > 1$. It only makes sense to design specific algorithms for a $POPS(d, g)$ network for the case when $d > 1$. For the case $d = 1$, the POPS network is a completely connected network and it is easier to simulate the corresponding hypercube algorithm. Moreover, any algorithm designed for a $POPS(d, g)$ network should perform better than the corresponding simulated algorithm on the hypercube.

Most of the algorithms designed by Sahni have different complexities for different group sizes. He usually expresses the complexity of these algorithms for three cases, **(i)** $d = g = \sqrt{n}$, **(ii)** $d < \sqrt{n} < g$ and **(iii)** $d > \sqrt{n} > g$. Gravenstreter and Melhem [4] mention that for most practical systems, the number of couplers will be less than the number of processors. Indeed, for a $POPS(d, g)$, with $dg = n$, if the number of groups $g > \sqrt{n}$, we need $g^2 > n$ couplers to connect such a network. It is unlikely that a practical system will be built with such a high number of couplers. One would expect that the number of couplers in most practical systems will be significantly smaller than the number of processors. In other words, the group size $d > \sqrt{n}$ in a practical system and hence, $g < \sqrt{n}$. We feel that cases **(i)** and **(ii)** are unrealistic in most practical systems since in these cases the number of OPS couplers g^2 is equal to or greater than n, the number of processors. Hence, our main motivation is to design fast algorithms for $POPS(d, g)$ networks such that $d > \sqrt{n} > g$.

In this paper, we present fast algorithms for two fundamental arithmetic operations, data sum and prefix sum, for POPS networks with large group size and hence, small number of groups. Our algorithms improve upon the complexities obtained by Sahni for these problems when $d > \sqrt{n} > g$. We present the comparison of the algorithms by Sahni [8] and our algorithms in Table 1.

Problem	*Sahni's [8] algorithm*	*Our algorithm*
Data sum	$\lceil \frac{d}{g} \rceil \log n$	$\frac{d}{g} + 2\log g - 1$
Prefix sum	$\frac{2d}{g}(1 + \log g) + \log d + 1$	$\frac{2d}{g} + 4\log g + 6$

Table 1. The comparison between our algorithms and Sahni's [8] algorithms. All complexities are in terms of number of slots. In this paper, all logarithms are to the base 2.

The rest of this paper is organized as follows. In Section 2, we discuss some details of the POPS network. We present our data sum algorithm in Section 3. Finally, we present the prefix sum algorithm in Section 4.

2 The POPS network

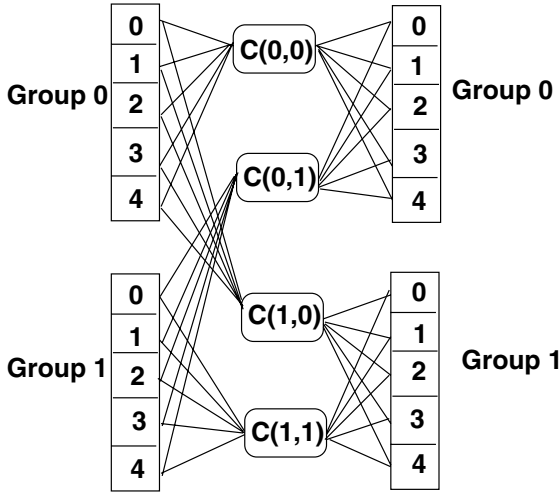

Fig. 1. A 10-processor computer connected via a $POPS(5, 2)$ network.

A POPS(d, g) network partitions the n processors into g groups of size d each and optical passive stars (OPS) couplers are used to connect such a network when every pair of groups is connected by a coupler. Hence, overall g^2 couplers are needed. Each processor must have g optical transmitters, one transmitter each for transmitting to the g OPSs for which it is a source node. Also, each processors should have g optical receivers, for receiving data from each of the g couplers. Each OPS in a $POPS(d, g)$ network has degree d. In one slot, each OPS can receive a message from any one of its source nodes and broadcast the

message to all of its destination nodes. However, in one slot, a processor can receive a message from only one of the OPSs for which it is a destination node. Melhem *et al.* [7] observe that faster all-to-all broadcasts can be implemented by allowing a processor to receive different messages from different OPSs in the same slot. However, in this paper, we will assume that only one message can be received by a processor in one slot. A ten-processor computer connected via a $POPS(5, 2)$ network is shown in Figure 1.

The g groups in a POPS network are numbered from 1 to g. A pair of groups is connected by an optical passive star (OPS) coupler. For coupler $c(i, j)$, the source nodes are the processors in group j and the destination nodes are the processors in group i, $1 \leq i, j \leq g$. Note that we have shown each group twice in Figure 1 to indicate that processors in each group are both receivers and senders.

The most important advantage of a POPS network is that its diameter is 1. A message can be sent from processor i to processor j ($i \neq j$) in one slot. We use the notation $group(i)$ to indicate the group that processor i is in. To send a message from processor i to processor j, processor i uses the coupler $c(group(j), group(i))$. Processor i first sends the message to $c(group(j), group(i))$ and then $c(group(j), group(i))$ sends the message to processor j. Similarly, one-to-all broadcasts also can be implemented in one slot.

We refer to the following result obtained by Sahni [8].

Theorem 1. *[8] An n processor $POPS(d, g)$ can simulate every move of an n processor SIMD hypercube using one slot when $d = 1$ and using $2\lceil d/g \rceil$ slots when $d > 1$.*

One consequence of Theorem 1 is that we can simulate an existing hypercube algorithm on a POPS network.

In our algorithms, we use the following results obtained by Sahni [8].

Lemma 1. *[8] If n data items are distributed one per processor in a $POPS(g, g)$ with $n = g^2$ processors, their sum can be computed in $\lceil \frac{g}{g} \rceil \log n = 2 \log g$ slots.*

Lemma 2. *[8] If n data items are distributed one per processor in a $POPS(d, g)$ with $n = gd$ processors, with $1 < d \leq g$, their prefix sum can be computed in $3 + \log n + \log d$ slots.*

3 Data sum

Initially n data items are distributed among n processors, one item per processor in a $POPS(d, g)$. The purpose of the *data sum* operation is to compute the sum of these n data items and bring the sum in the first processor of the first group. We consider the case when $d > \sqrt{n} > g$.

Our algorithm is based on the following strategy. Our main aim is to reduce the number of data items rapidly so that there are only g data items in each group. Since there are g groups, we can use Sahni's algorithm [8] i as stated in Lemma 1 at this point to compute the data sum of these g^2 remaining elements.

To start with, we divide the processors in each group into two categories.

• The first g processors in each group i are called *receivers*. The set of g receivers in the i-th group, $1 \leq i \leq g$, is denoted by R_i and the j-th receiver in the i-th group is denoted by $r_{i,j}$, $1 \leq j \leq g$.
• The $(d-g)$ processors other than the receivers in each group are called the *senders*. The set of senders in the m-th group, $1 \leq m \leq g$, is denoted by S_m.

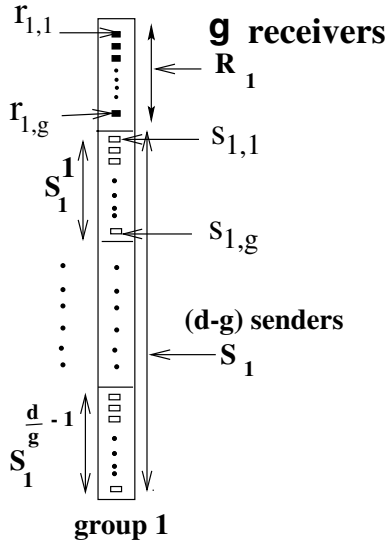

Fig. 2. The *Receiver* and *Sender* sets for group 1.

Consider the i-th group, $1 \leq i \leq g$. First, we divide the set $S_i (1 \leq i \leq g)$ of $d-g$ senders in the group $i(\ 1 \leq i \leq g)$ into $\frac{d}{g}-1$ subsets, each consisting of g processors and we denote them by $S_i^1, \ldots, S_i^{\frac{d}{g}-1}$. Note that there are g senders in each subset S_i^j, $1 \leq j \leq \frac{d}{g}-1$. We denote the senders in the subset S_i^j of S_i by $s_{i,(j-1)g+1}, s_{i,(j-1)g+2}, \ldots, s_{i,jg}(\ 1 \leq j \leq (\frac{d}{g}-1))$. The structure of the first group is shown in Figure 2.

In each stage, we perform the following computation in each group $i(1 \leq i \leq g)$ in parallel. At stage $k(\ 1 \leq k \leq \frac{d}{g}-1)$, the participating subsets are $S_1^k, S_2^k, \ldots, S_g^k$ from group 1, group 2, ..., group g respectively. For the subset S_j^k in group j, the senders $s_{j,(k-1)g+1}, \ldots, s_{j,kg}$ send their data to the receivers $r_{1,j}$ in group 1, $r_{2,j}$ in group 2 ..., $r_{g,j}$ in group g respectively. The couplers used for this data movement are $c(*, j)$ where '*' indicates all the values from 1 to g. This computation takes one slot. This data movement is performed in parallel by all the g subsets $S_i^k (1 \leq i \leq g)$. Hence, the computation at stage k takes one slot. Further, g^2 data items are moved in stage k and g^2 different couplers

are used since each subset S_i^k of S_i $(1 \leq i \leq g)$ in group i uses the g couplers connected to group i. Note that each receiver receives only one data item. The computation is the same for each stage except that the participating subsets are different. Since there are $\frac{d}{g} - 1$ stages, the total number of slots is $\frac{d}{g} - 1$. The data movement at Stage 1 is illustrated in Figure 3.

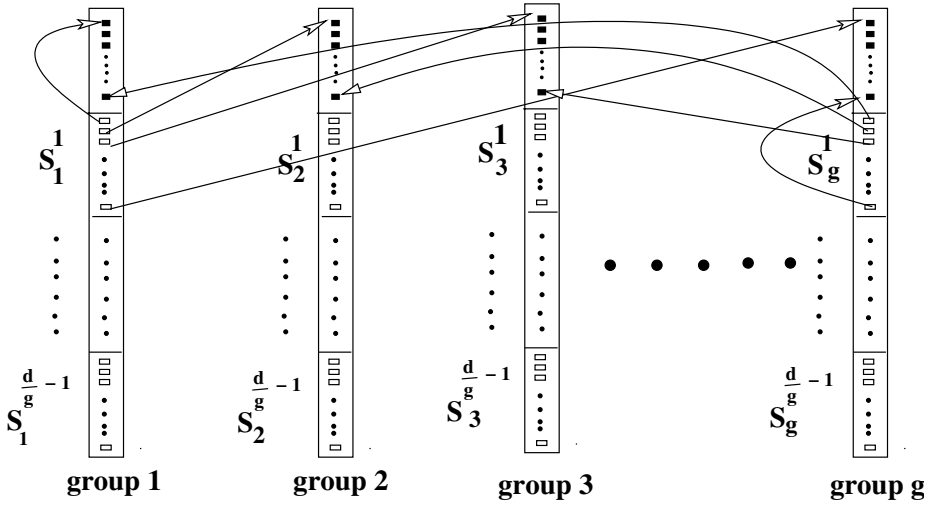

Fig. 3. At stage 1, the participating subsets are $S_1^1, S_2^1, \ldots, S_g^1$. The data movement is shown for subsets S_1^1 and S_g^1.

Once the receivers receive the data items, they add the received data with their existing data. In the k-th stage $(1 \leq k \leq \frac{d}{g} - 1)$, the g senders $s_{i,(k-1)g+1}, \ldots, s_{i,kg}$ in S_i^k send their data items$(1 \leq i \leq g)$ to i-th receivers $r_{1,i}, \ldots, r_{g,i}$ of the groups $1, 2, \ldots, g$ respectively. Each receiver add the data item that it receives at this stage to its accumulated sum. Note that after stage k, each group has $d - kg$ senders holding data items.

After $\frac{d}{g} - 1$ stages, all the senders have sent their data to the receivers. Hence, each group has now g receivers holding their accumulated sums. Our next task is to add the accumulated sums in these g^2 receivers and bring this sum to the first processor in the first group. We do this using Sahni's [8] algorithm in Lemma 1. This computation takes $2 \log g$ slots. Hence, the sum of all the data items in the $n = dg$ processors is computed after $\frac{d}{g} + 2 \log g - 1$ slots. We have shown the following lemma.

Lemma 3. *The sum of n data items initially given in a $POPS(d, g)$ with one data item per processor, $dg = n$ and $d > \sqrt{n} > g$, can be computed in $\frac{d}{g} + 2 \log g - 1$ slots.*

4 Prefix sum

In this section, we present an algorithm on a $POPS(d, g)$ for computing prefix sum in $\frac{2d}{g}+4\log g+6$ slots when $d > \sqrt{n} > g$. The strategy behind our algorithm is the following.

We first divide each group of d processors into g subgroups with $\frac{d}{g}$ processors in each subgroup. We denote the set of g subgroups in group i by $S_i, 1 \leq i \leq g$. The j-th subgroup in S_i is denoted by $S_{i,j}, 1 \leq j \leq g$. We compute the prefix sum in each such subgroup $S_{i,j} (1 \leq i, j \leq g)$ with respect to the elements in $S_{i,j}$ and we call these prefix sums as subgroup prefix sums. After this computation, we are left with g groups and g subgroup prefix sums in each group.

Consider the k-th element in $S_{i,j}$. Next we add two contributions to the subgroup prefix sum of $S^k_{i,j}$ to get its overall prefix sum. The first contribution is the sum of all the elements in the groups 1 to $i-1$. The second contribution is the sum of all the elements in the subgroups $S_{i,1}$ to $S_{i,j-1}$. Our algorithm is divided into three phases.

Phase 1.

The purpose of this phase is to distribute copies of the g subgroups of each group among the other groups. Consider the i-th group and its set $S_i = \{S_{i,j} | 1 \leq j \leq g\}$. Note that each subgroup $S_{i,j}$ contains $\frac{d}{g}$ elements. We use the same notation $S_{i,j}$ to denote the processors as well as the input elements in the subgroup $S_{i,j}$. The copies of the elements in $S_{i,j}$ are sent to the processors in the subgroup $S_{j,i}$ of S_j in group j, one element per processor, by using the coupler $c(j, i)$. This data movement is done for all $S_{i,j}, 1 \leq i \leq g, 1 \leq j \leq g$, in parallel. Note that there is no coupler conflict as for any two couplers, both of their indices do not match. In other words, we are transmitting data from g^2 subgroups and we have g^2 couplers and hence, there is no coupler conflict.

This data transmission is performed in $\frac{d}{g}$ slots since there are these many elements in each subgroup $S_{i,j}$ and a coupler can receive one data item in each slot.

Phase 2.

We compute the subgroup prefix sums in this phase. Consider the subgroup $S_{i,j}$ of S_i, the j-th subgroup in the i-th group. We denote the $\frac{d}{g}$ elements in this subgroup as $S^k_{i,j}, 1 \leq k \leq \frac{d}{g}$. Recall that copies of all the elements in $S_{i,j}$ has been sent to the subgroup $S_{j,i}$ in Phase 1.

Now, the processors in $S_{j,i}$ broadcast the elements of $S_{i,j}$ to the processors in $S_{i,j}$ in $\frac{d}{g}$ slots. This broadcast is done by the processors in $S_{j,i}$ one by one.

- In the first slot, $S^1_{j,i}$ broadcasts the element $S^1_{i,j}$ to the processors $S^k_{i,j}, 2 \leq k \leq \frac{d}{g}$. These processors add $S^1_{i,j}$ with the input they hold. In general, in the m-th slot, $1 \leq m \leq \frac{d}{g}$, the processor $S^m_{j,i}$ broadcasts the element $S^m_{i,j}$ to the processors $S^k_{i,j}, m+1 \leq k \leq \frac{d}{g}$.
- This computation is done by all the subgroups in all the groups in parallel,

i.e., in the subgroups $S_{i,j}, 1 \leq i \leq g, 1 \leq j \leq g$. Note that the coupler used to do the broadcasts from processors in $S_{j,i}$ to processors in $S_{i,j}$ is $c(i,j)$. Hence, all the subgroups can communicate with the help of the g^2 couplers and there is no coupler conflict as each coupler $c(i,j)$ will have its two indices i and j different from all other couplers. Further, each process broadcasts at most one element and receives at most one element in each slot. The total number of slots required in this phase is $\frac{d}{g}$.

Note that at the end of the $\frac{d}{g}$ slots, all the processors in $S_{i,j}$ have computed the prefix sum of its element with respect to the elements in $S_{i,j}$ We call these prefix sums as subgroup prefix sums and the subgroup prefix sum of the element $S^k_{i,j}, 1 \leq i,j \leq g$ is denoted by $SubPrefix(S^k_{i,j})$.

end of Phase 2

For the computation in the next phase, we are interested in the prefix sums $SubPrefix(S^{\frac{d}{g}}_{i,j}), 1 \leq i,j \leq g$. Note that $S^{\frac{d}{g}}_{i,j} = \sum_{m=1}^{\frac{d}{g}} S^m_{i,j}$, i.e., the last element $S^{\frac{d}{g}}_{i,j}$ of the subgroup $S_{i,j}$ holds the sum of all the elements in it. Hence, the sum $\sum_{j=1}^{g} SubPrefix(S^{\frac{d}{g}}_{i,j})$ gives the sum of all the elements in the i-th group and we call this the *group-sum* for group i and denote it by GS_i.

Consider the element $S^k_{i,j}(1 \leq k \leq \frac{d}{g})$, i.e., the k-th element of the j-th subgroup $S_{i,j}$ in group i. We denote the actual prefix sum of $S^k_{i,j}$ w.r.t all the elements in $POPS(d,g)$ by $prefix(S^k_{i,j})$. The computation of $prefix(S^k_{i,j})$ is performed as follows. After the computation in Phase 2, we know $SubPrefix(S^k_{i,j})$. We add the following two quantities to $SubPrefix(S^k_{i,j})$ to compute $prefix(S^k_{i,j})$.

(i). The first quantity is $\sum_{m=1}^{i-1} GS_m$, i.e., the contribution of all the groups before the i-th group.

(ii). The second quantity is $\sum_{p=1}^{j-1} SubPrefix(S^{\frac{d}{g}}_{i,p})$, i.e., the contribution of all the subgroups before the j-th subgroup in group i. We discuss the computation of these two contributions **(i)** and **(ii)** in Phase 3.

Phase 3.

After the computation in Phase 2, each of the g groups has g subgroup prefix sums. Hence, for each group i, we can compute the prefix sums of the subgroup prefix sums $SubPrefix(S^{\frac{d}{g}}_{i,1}), SubPrefix(S^{\frac{d}{g}}_{i,2}), \ldots, SubPrefix(S^{\frac{d}{g}}_{i,j})$. by considering our $POPS(d,g)$ as a $POPS(g,g)$ and using Sahni's algorithm [8] as described in Lemma 2. In this case, $d = g$ and $n = g^2$ and hence, the number of slots required is $3 + \log n + \log d = 3 + \log(g^2) + \log g = 3 + 3\log g$.

Note that the quantity $GS_i, 1 \leq i \leq g$, will be available in the i-th group after this computation. We now compute the prefix sums of the GS_i's by using our $POPS(d,g)$ as a $POPS(1,g)$ and by using Sahni's algorithm for simulating a hypercube algorithm on a $POPS(1,g)$. This computation takes $\log g$ slots.

Now, the last processor in each group $i, 1 \leq i \leq g$ has got the prefix sums $\sum_{j=1}^{i} GS_j$. This processor computes $\sum_{j=1}^{i-1} GS_j = \sum_{j=1}^{i} GS_j - GS_i$ and broadcasts this quantity to all the processors in group $i, 1 \leq i \leq g$ by using the

coupler $c(i, i)$. All processors add this quantity to their current prefix sum. This computation takes one slot.

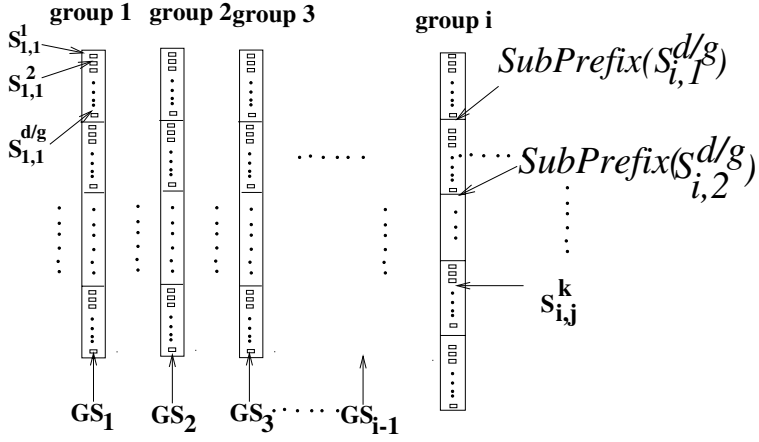

Fig. 4. Illustration for the computation in Phase 3. $prefix(S^k_{i,j})$ is computed by adding $\sum_{m=1}^{i-1} GS_m$ and $\sum_{p=1}^{j-1} SubPrefix(S^{\frac{d}{g}}_{i,p})$ to $SubPrefix(S^k_{i,j})$.

We have to compute the final contribution in the prefix sum of each element due to the contribution of the subgroup prefix sums within its own group. Recall that we already know the subgroup prefix sums due to the computation in the first part of this phase. Each group i has g subgroup prefix sums $SubPrefix(S^{\frac{d}{g}}_{i,1})$, $\sum_{p=1}^{2} SubPrefix(S^{\frac{d}{g}}_{i,p}), \ldots, \sum_{p=1}^{j} SubPrefix(S^{\frac{d}{g}}_{i,p})$, $\ldots, \sum_{p=1}^{g} SubPrefix(S^{\frac{d}{g}}_{i,p})$.

We need to broadcast the subgroup prefix sum $\sum_{p=1}^{j-1} SubPrefix(S^{\frac{d}{g}}_{i,p})$ to all the processors in $S_{i,j}$ and these processors should update their existing sum by adding this quantity. If we use the coupler $c(i, i)$ to do this broadcast, then we will need g slots. Instead, we do this broadcast for all the groups in two slots in the following way.

• We specify $g - 1$ processors from each of the g groups as receivers. We denote the $g - 1$ receivers for group j, $1 \leq j \leq g$ by $r_{j,k}, 1 \leq k \leq (g - 1)$. The processors $S^{\frac{d}{g}}_{i,j} (1 \leq j \leq (g - 1))$ in group i holding the $(g - 1)$ subprefix sums $\sum_{p=1}^{j} SubPrefix(S^{\frac{d}{g}}_{i,p})$ send these sub prefix sums to the i-th receivers in each group. For example, $SubPrefix(S^{\frac{d}{g}}_{i,1})$ is sent to the receiver $r_{1,i}$, $\sum_{p=1}^{2} SubPrefix(S^{\frac{d}{g}}_{i,p})$ is sent to the receiver $r_{2,i}$ and in general, the sum $\sum_{p=1}^{m} SubPrefix(S^{\frac{d}{g}}_{i,p})$, $1 \leq m \leq (g - 1)$ is sent to the receiver $r_{m,i}$, for all

$1 \leq i \leq g$. This data movement takes one slot. Processors in group i use the couplers $c(*, i)$, where '*' means all indices $1, \ldots, g$. Hence, no two couplers used in this data transfer, will have both indices same and in the same order. This shows that there is no coupler conflict.

• Next, the j-th receiver $r_{k,j}, 1 \leq k \leq (g-1)$ in group k broadcasts the sum $\sum_{p=1}^{k} SubPrefix(S_{i,p}^{\frac{d}{g}})$ to all the processors in the subgroup $S_{j,k+1}$ using the coupler $c(j,k)$. This broadcast is done by all the receivers $r_{k,j}, 1 \leq k \leq g, 1 \leq j \leq (g-1)$. There is no coupler conflict as the the coupler indices depend on the corresponding receiver index and no two receivers have matching indices.

• Finally, each processor in each of the subgroups updates its prefix sum by adding the received quantity with its existing prefix sum.

The total slots required for the computation in this phase is $4 \log g + 6$. This concludes the computation of prefix sum. Hence, we have the following lemma.

Lemma 4. *The prefix sum of n data items initially given in the $n = dg$ processors of a $POPS(d, g)$ with $d > \sqrt{n} > g$, can be computed in $\frac{2d}{g} + 4 \log g + 6$ slots.*

References

1. P. Berthomé and A. Ferreira, "Improved embeddings in POPS networks through stack-graph models", *Proc. Third International Workshop on Massively Parallel Processing Using Optical Interconnections*, pp. 130-135, 1996.
2. P. Berthomé, J. Cohen and A. Ferreira, "Embedding tori in partitioned optical passive stars networks", *Proc. Fourth International Colloquium on Structural Information and Communication Complexity (Sirocco '97)*, pp. 40-52, 1997.
3. D. Chiarulli, S. Levitan, R. Melhem, J. Teza and G. Gravenstreter, "Partitioned optical passive star (POPS) multiprocessor interconnection networks with distributed control", *Journal of Lightwave Technology*, **14** (7), pp. 1901-1612, 1996.
4. G. Gravenstreter and R. Melhem, "Realizing common communication patterns in partitioned optical passive stars (POPS) networks", *IEEE Trans. Computers*, **47** (9), pp. 998-1013, 1998.
5. G. Gravenstreter, R. Melhem, D. Chiarulli, S. Levitan and J. Teza, "The partitioned optical passive stars (POPS) topology", *Proc. Ninth International Parallel Processing Symposium*, IEEE Computer Society, pp. 4-10, 1995.
6. V. Prasanna Kumar and V. Krishnan, "Efficient template matching on SIMD arrays", *Proc. 1987 International Conference on Parallel Processing*, pp. 765-771, 1987.
7. R. Melhem, G. Gravenstreter, D. Chiarulli and S. Levitan, "The communication capabilities of partitioned optical passive star networks", *Parallel Computing Using Optical Interconnections*, K. Li, Y.Pan and S. Zheng (Eds), Kluwer Academic Publishers, pp. 77-98, 1998.
8. S. Sahni, "The partitioned optical passive stars network : Simulations and fundamental operations", *IEEE Trans. Parallel and Distributed Systems*, **11** (7), pp. 739-748, 2000.
9. S. Sahni, "Matrix multiplication and data routing using a partitioned optical passive stars network", *IEEE Trans. Parallel and Distributed Systems*, **11** (7), pp. 720-728, 2000.