Compiled Communication for All–optical TDM Networks

Xin Yuan   Rami Melhem   Rajiv Gupta
Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260
{xyuan,melhem,gupta}@cs.pitt.edu
http://www.cs.pitt.edu/{~xyuan, ~melhem, ~gupta}

Abstract

While all–optical networks offer large bandwidth for transferring data, the control mechanisms to dynamically establish all–optical paths incur large overhead. In this paper, we consider adapting all–optical multiplexed networks in multiprocessor or multicomputer environment by using *compiled communication* as an alternative to dynamic network control. Compiled communication eliminates the runtime overhead by managing network resources statically. Thus, it can employ complex off–line algorithms to improve resource utilization. We studied several off–line connection scheduling algorithms for minimizing the multiplexing degree required to satisfy communication requests. The performance of compiled communication is evaluated and compared with that of dynamically controlled communication for static communications in a number of application programs. Our results show that compiled communication out–performs dynamic communication to a large degree. Since most of the communication patterns in scientific applications are static, we conclude that compiled communication is an effective mechanism for all–optical networks in multiprocessor environments.

Keyword: Compiled Communication, Time-division Multiplexing, Interconnection Network, All-optical.

1 Introduction

With the increasing computation power of parallel computers, interprocessor communication has become an important factor that limits the performance of parallel computers. Due to their capability of offering large bandwidth and low latency, all–optical interconnection networks are considered promising networks for future massively parallel computers. In all–optical networks, communications are carried out in a pure circuit–switching fashion in order to avoid electronic/optical or optical/electronic conversions at intermediate nodes. Specifically, packet switching techniques, which are usually used in electronic multicomputer and multiprocessor interconnection networks, are at a disadvantage when optical transmission is used. The lack of suitable photonic logic devices makes it extremely inefficient to process packet routing information in the photonic domain. Moreover, conversion of this information into the electronic domain increases the latency at intermediate nodes relative to the internode propagation delay.

Although this optical/electronic conversion may be acceptable for large distributed networks [4], it represents a disadvantage for multiprocessor networks in which internode propagation delays are very small.

Multiplexing techniques are used in optical networks to fully utilize the large bandwidth of optics. Multiplexing enables multiple virtual channels to be formed on a single link. It is possible to concurrently establish multiple connections using a single fiber in a multiplexed network. Therefore for a given topology multiplexing increases the connectivity of the network. Many research efforts have focussed on two multiplexing techniques for optical interconnection networks, namely *time–division multiplexing* (TDM) [12, 13, 14] and *wavelength–division multiplexing* (WDM) [1, 4]. TDM multiplexes the links by establishing different virtual channels during different *time slots* while WDM multiplexes the links by having different virtual channels use different *wavelengths*. By using TDM or WDM, each link can support multiple channels with each channel operating at a speed close to the electronic processing speed.

While fiber optic networks have the potential for providing large bandwidth, the establishment of all–optical paths from sources to destinations places strict demands on the control of the interconnection network. Specifically, the network control, be it centralized or distributed, is usually performed in the electronic domain and thus is very slow in comparison to the large bandwidth supported by the optical data paths. One way to minimize the control overhead is to avoid the need for dynamic control by using *compiled communication* techniques whenever the communication patterns can be determined statically. This technique eliminates the overhead of establishing the all–optical paths at runtime. Although compiled communication is most effective for communication patterns that can be determined at compile time, the use of compiled communication is expected to improve the overall network performance significantly since over 95% of the communications in large scientific programs can be determined completely or parametrically at compile time [10]. Moreover, multiplexing improves the efficiency of compiled communication by reducing the frequency of network reconfiguration and the need for inserting additional synchronization operations at reconfiguration points.

In this paper, we investigate the effectiveness of applying compiled communication technique to time–multiplexed all–optical networks. Several off–line connection scheduling heuristics are proposed and evaluated. Performance of compiled communication is evaluated for a set of application programs. We compare the compiled communication with dynamic switching networks with and without multiplexing. We observe large performance gains when compiled communication is used for static patterns extracted from a set of programs.

This paper is organized as follows. In Section 2, we briefly describe the time–division multiplexing technique. The discussion of the compiled communication is presented in Section 3, where heuristics for scheduling connections are described and evaluated. Section 4 compares the performance of compiled communication with that of dynamic control assuming a torus topology, and Section 5 presents the conclusion of this work.

2 Time–Division Multiplexing (TDM)

In general, any $N \times N$ network, other than a completely connected network, has a limited connectivity in that only subset of the possible N^2 connections can be established simultaneously without conflict. For all-optical communication, the network must be capable of establishing any possible connection in one hop, that is, without intermediate relaying or routing. Hence, the network must be able to change the connections it supports at different times. We consider switching networks in which the set of connections that may be established simultaneously, that is, the state of the network, is selected by changing the contents of hardware registers. An example of such a network is the two-dimensional torus network shown in Fig. 1, where each processor is connected to a 5×5 electro-optical switch, which is, in turn, connected to four other switches in a torus topology [14]. By controlling the contents of an electronic control register, a switch can connect any of its five optical inputs to any of its five optical outputs. The state of the network is determined by the states of all its switches. The set of connections that are established in a given network state is called a *configuration.* A set of connections is a *configuration* if no two connections in the set share a common link. Fig. 1 shows the configuration $\{(4,1),(5,3),(6,10),(8,9),(11,2)\}$ realized in a 4×4 torus.

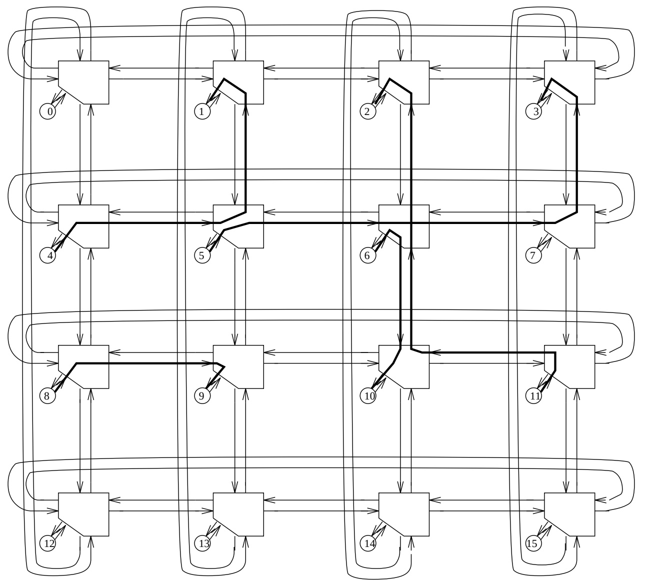

Figure 1: configuration $(4,1),(5,3),(6,10),(8,9),(11,2)$.

In order to time–multiplex a network the time domain is divided into a repeated sequence of K time slots of fixed duration, where K is called the *multiplexing degree.* If C_1, C_2, ..., C_K are K configurations for the network, then TDM realizes all the K configurations by periodically cycling through the K network states with each state establishing the connections in one configuration. This cycling of states can be accomplished efficiently by using circular shift registers to control each switch. Note that when $K = 1$, the network degenerates to the traditional circuit–switching network. The network control under TDM is synchronous, that is, a global clock is distributed to each processor in the system to align the time slots. Many current commercially massively parallel systems, such as CRAY-T3D and CM5, support a distributed clock that is synchronous in all processors.

Dynamic control of TDM networks can be either *centralized* or *distributed* [13]. The centralized control mechanism uses a single controller to process all the requests and thus it does not scale with the system size. Since our approach targets large systems, the dynamic control mechanism is assumed to be distributed. Consequently, a fixed multiplexing degree is assumed because varying the multiplexing degree with distributed control is too costly to implement [13]. Distributed dynamic control schemes for TDM networks are discussed in [15, 17].

Compiled communication can be supported in TDM networks by inserting, in to the code executed on the processors, instructions to appropriately set the registers controlling the switches. Writing onto these registers must be synchronized to avoid non-deterministic network states. Since the multiplexing degree for the network is controlled by the compiler, different multiplexing degrees can be used in different phases of the parallel program to establish the connections needed for interprocessor communication.

3 Compiled Communication

Compiled Communication has recently drawn the attention of several researchers [3, 8]. It has been used in combination with message passing in the iWarp system [6]. Compiled communication is used for a specific set of patterns while other communications are handled using message passing. The prototype system described in [3] avoids the cost of supporting multiple communication models. It relies exclusively upon compiled communication. However, the performance of this system is severely limited due to frequent dynamic reconfiguration of the network. In this paper we also assume that the system supports only the compiled communication model. However, we are able to avoid frequent reconfiguration through TDM.

The communication patterns in an application program can be broadly classified into two categories: *static patterns* that can be recognized by the compiler and *dynamic patterns* that are only known at run-time. Compiled communication handles the static patterns with high efficiency and at the same time it provides acceptable performance for dynamic patterns. Since recent studies [10] have shown that over 95% of the patterns in scientific computations are statically known, compiled communication provides an efficient means for handling communication requests. For the static patterns, compiled communication computes a minimal set of network configurations that satisfy the requirements of the patterns. For the dynamic patterns, compiled communication uses a predetermined set of configurations to allow communication between all processors in the system.

There are many advantages of using compiled communication for handling static patterns. Since the control algorithms are executed off-line by the compiler, complex strategies to manage the network resources can be employed. The control network, along with the limitations in its implementation, such as the head-of-line effect due to single queue [16], is altogether eliminated. Since no routing decisions are made at runtime, the packet header can also be eliminated causing the network bandwidth to be utilized more effectively. The combination of TDM and compiled communication enables the reduction of network synchronization and reconfiguration overhead. Finally, compiled communication can compute the minimal number of configurations and hence the multiplexing degree required for the static patterns. In contrast a dynamic technique must choose a multiplexing degree without any information about the communication patterns. Thus, dynamic configuration techniques utilize TDM less effectively than compiled communication.

In order to apply compiled communication to a large scale multiprocessor system, three main issues must be addressed:

Communication pattern recognition: This issue has been considered by many researchers since information on communication patterns has been previously used to perform communication optimizations [2, 11, 7, 9]. The stencil compiler for CM-2 recognizes *stencil* communication patterns [2]. Chen and Li [11] incorporated pattern extraction mechanism in a compiler to support the use of collective communication primitives. Techniques for recognizing a broad set of communication patterns were also proposed in [7]. In this paper we rely upon existing techniques for identifying communication patterns.

Compiling static patterns: Because the communication time in a multiplexed network is proportional to the multiplexing degree, compiled communication computes the minimal multiplexing degree required for satisfying the static communication patterns. We have developed a number of message scheduling algorithms for computing the minimal set of configurations that satisfy a given set of communication patterns. Note that each configuration set corresponds to a multiplexing degree. It has been shown that optimal message scheduling for arbitrary topologies is NP-complete [4]. Therefore our algorithms are heuristics that are demonstrated to provide good performance. The configurations found by these heuristics are established using TDM.

Handling dynamic patterns: Compiled communication cannot handle dynamic patterns efficiently. However, since only a small portion of the communications are dynamic, some techniques can be used to ensure a correct execution of programs in the present of dynamic communication patterns without degrading the overall network performance significantly. Some possible approaches for handling dynamic patterns are as follows. One approach is to setup all-to-all pattern among all the nodes in the system. This way each node has a time slot to communicate with every other node. However, establishing paths for all-to-all communication can be prohibitively expensive for a large system. Another approach is to use static TDM to embed a logical communication topology into the physical network and to emulate communications in multihop systems. Detailed comparison of these approaches is beyond the scope of this paper.

The rest of the section will focus on compiling static pattern through connection scheduling. Before describing the connection scheduling algorithms in detail, we first present some definitions to formally state the problem of connection scheduling. We denote a connection request from a source s to a destination d as (s, d).

Definition: A pair of connection requests (s_1, d_1) and (s_2, d_2) are said to *conflict*, if they cannot be simultaneously established.

Definition: A *configuration* is a set of connection requests $\{(s_1, d_1), (s_2, d_2), ..., (s_m, d_m)\}$ such that no two requests in the set conflict.

Definition: Given a set of communication requests $R = \{(s_1, d_1), (s_2, d_2), ..., (s_m, d_m)\}$, the set $MC = \{C_1, C_2, ..., C_t\}$ is a **minimal configuration set** for R iff:

- each $C_i \in MC$, $1 \leq i \leq t$, is a configuration and each request $(s_i, d_i) \in R$, $1 \leq i \leq m$, is contained in exactly one configuration in MC; and
- for each pair of configurations C_i and $C_j \in MC$, there exists a request $(s_i, d_i) \in C_i$ and a request $(s_j, d_j) \in C_j$ such that (s_i, d_i) conflicts with (s_j, d_j).

As discussed in section 2, a minimal configuration set of size K can be supported using a multiplexing degree of K. Thus, the goal of connection scheduling heuristics is to compute minimum configuration set for a given request set R. Next we describe three connection scheduling heuristics.

3.1 Greedy Algorithm

In the greedy algorithm, a configuration is created by repeatedly including additional connections into the configuration until no additional connection can be established in that configuration. If additional requests remain, another configuration is created. This process is repeated till all requests have been included in some configuration. The algorithm described is a modification of an algorithm proposed in [13]. The algorithm is shown in Fig. 2. The time complexity of the algorithm is $O(|R| \times max_i(|C_i|) \times K)$, where $|R|$ is the number of the requests, $|C_i|$ is the number of connections in configuration C_i and K is the number of configurations generated.

```
(1)   MC = $\phi$, k = 1
(2)   repeat
(3)     $C_k = \phi$
(4)     for each $(s_i, d_i) \in R$
(5)         if $(s_i, d_i)$ does not conflict
(6)           with any connection in $C_k$
(7)         then
(8)           $C_k = C_k \bigcup \{ (s_i, d_i) \}$
(9)           R = R $-$ { $(s_i, d_i)$ }
(10)        end if
(11)    end for
(12)    MC = MC $\bigcup$ { $C_k$ }
(13)    k = k + 1
(14)  until $R = \phi$
```

Figure 2: The greedy algorithm.

Consider the linearly connected nodes shown in Fig. 3. The result of applying the greedy algorithm to schedule connection requests set {(0, 2), (1, 3),(3, 4), (2, 4)} is shown in Fig. 3(a). In this case, (0, 2) will be in time slot 1, (1, 3) in time slot 2, (3, 4) in time slot 1 and (2, 4) in time slot 3. Therefore, a multiplexing degree of 3 is needed to establish the paths for the four connections. However, as shown in Fig. 3 (b), the optimal scheduling for the four connections, which can be obtained by considering the connection in different order, is to schedule (0, 2) in slot 1, (1, 3) in slot 2, (3, 4) in slot 2 and (2, 4) in slot 1. This assignment only uses 2 time slots to establish all the connections.

3.2 Coloring Algorithm

The greedy algorithm processes the requests in an arbitrary order. In this section, we will describe an algorithm that applies a heuristic to determine the order in which to process the connection requests. The heuristic assigns higher priorities to connection requests with fewer conflicts. By giving the requests with less conflicts higher priorities, each configuration is likely

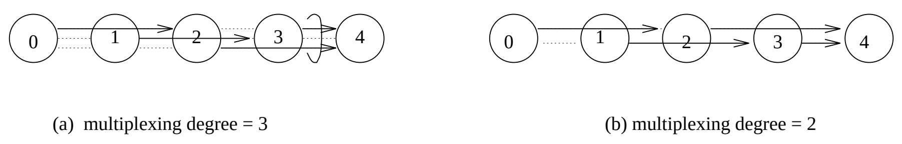

Figure 3: Scheduling requests (0, 2), (1, 3),(3, 4), (2, 4).

to accommodate more requests and thus the multiplexing degree needed for the patterns is likely to decrease.

We formulate the problem of computing the minimal configuration set as a graph coloring problem. A coloring of a graph is an assignment of a color to each node of the graph in such a manner that no two nodes connected by an edge are assigned the same color. The nodes in the graph correspond to connection requests. Edges are introduced between nodes that represent conflicting requests. This graph is referred to as a *conflict graph.* It can be easily shown that the number of colors used to color the graph is equal to the number of configurations needed to handle the connection requests. In order to minimize the multiplexing degree, our coloring algorithm attempts to minimize the number of colors used in coloring the graph.

```
(1)     Construct conflict graph G = (V, E)
(2)     Calculate the priority for each node
(3)     MC = $\phi$, k = 1
(4)     NCSET = V
(5)     repeat
(6)       Sort NCSET by priority
(7)       WORK = NCSET
(8)       $C_k = \phi$
(9)       while (WORK $\neq \phi$)
(10)         Let $n_f$ be the highest priority node in WORK
(11)         $C_k = C_k \bigcup \{(s_f, d_f)\}$
(12)         NCSET = NCSET $-\{n_f\}$
(13)         for each $n_i \in NCSET$ and $(f, i) \in E$ do
(14)           update the priority of $n_i$
(15)           WORK = WORK - $\{n_i\}$
(16)         end for
(17)      end while
(18)      MC = MC $\bigcup \{C_k\}$
(19)      k = k + 1
(20)    until NCSET = $\phi$
```

Figure 4: The graph coloring heuristic.

Since the coloring problem is known to be NP-complete, we use a heuristic for graph coloring. The heuristic determines the order in which to color the nodes by assigning and updating priorities to nodes. To enforce the *less conflict connection first* heuristic, we assign the priority for a connection request to be the ratio of the number of links in the connection to the degree of the corresponding node in the uncolored conflict subgraph. The algorithm is summarized in Fig 4. It should be noted that after a node is colored, our algorithm updates the priorities of uncolored nodes. This is because in computing the degree of an uncolored node, we only consider the edges that connect the node to other uncolored nodes. The algorithm finds a solution in linear time with respect to the size of the conflict graph. The time complexity of the algorithm is $O(|R|^2 \times max_i(|C_i|) \times K)$, where $|R|$ is the number of the requests, $|C_i|$ is the number of requests in configuration C_i, and K is the total number of configurations generated.

3.3 Ordered AAPC Algorithm

The graph coloring algorithm has better performance than the greedy heuristic. However, for dense communication patterns the heuristics cannot guarantee that the multiplexing degree found would be bounded by the minimum multiplexing degree needed to realize the all-to-all communication pattern. The algorithm described in this section targets dense communication patterns. By grouping the connection requests in a more organized manner, better performance can be achieved for such patterns.

Dense communication patterns may at most require *all-to-all personalized communication* (AAPC) where each node sends a message to every other node in the system. Any communication pattern can be embedded in AAPC. Many algorithms have been designed to perform AAPC efficiently for different topologies [8]. Among these algorithms, the ones that are of interest to us are the phased AAPC algorithms in which the AAPC connections are partitioned into contention–free phases. Since all the connections in each AAPC phase are contention–free, they form a configuration referred to as an *AAPC configuration.* A set of *AAPC configurations* for the AAPC communication pattern is called the *AAPC configuration set.* Some phased AAPC algorithms are optimal in that every link is used in each phase and every connection follows the shortest path.

If the connection requests are reordered according to an AAPC configuration set such that requests belonging to the same AAPC configuration are adjacent to each other, the use of the greedy scheduling algorithm to schedule the reordered requests will result in at most the number of AAPC configurations involved. For example, following the algorithms in [8] at most $N^3/8$ phases are needed for AAPC communication in a $N \times N$ torus. Therefore in a $N \times N$ torus, $N^3/8$ degree is enough to satisfy any communication pattern. Note that using greedy or coloring algorithms usually results in larger multiplexing degree for dense communication patterns.

To obtain better performance on dense communication patterns, it is better to keep the connections in their AAPC format as much as possible. It is therefore better to schedule the phases with higher link utilization first. This heuristic is used in the ordered AAPC algorithm depicted in Figure 5. In ordered AAPC algorithm, the rank of the AAPC phases is calculated so that a phase with higher utilization has a higher rank. The phases are then scheduled according to their ranks. The time complexity of this algorithm is $O(|R|(lg(|R|)+max_i(|C_i|)\times K))$, where $|R|$ is the number of the requests, $|C_i|$ is the number of requests in configuration C_i, and K is the number of configurations needed.

3.4 Performance of the Scheduling Algorithms

In this section, we study the performance of the connection scheduling algorithms on the 8×8 torus topology. The metric used to compare the algorithms is the multiplexing degree needed to establish the connections. The performance of the algorithms is evaluated using randomly

```
(1)  PhaseRank[*] = 0
(2)  for($s_i, d_i$) $\in R$ do
(3)      let $(s_i, d_i) \in A_k$
(4)      PhaseRank[k] = PhaseRank[k] + length(($s_i, d_i$))
(5)  end for
(6)  sort phase according to PhaseRank
(7)  Reorder R according the sorted phases.
(8)  call greedy algorithm
```

Figure 5: Ordered AAPC scheduling algorithm.

generated communication patterns, random data redistribution patterns, and some frequently used communication patterns. Next we describe how these patterns were obtained.

Random patterns: A random pattern consists of a certain number of random connection requests. A connection request is obtained by randomly generating the source and the destination. Uniform probability distribution is used to generate the sources and destinations.

Random data redistribution patterns: Many languages, such as CRAFT FORTRAN, allow an array to be redistributed within the program. Data redistributions usually result in interprocessor communication. In this study, we consider data redistributions of a 3-dimensional array of size $64 \times 64 \times 64$. The array is distributed using the block–cyclic distribution along each dimension. We denote a block–cyclic distribution as *p:block(s)*, where p is the number of processors in the distribution and s is the block size. The random data redistribution patterns are obtained by randomly generating the source and destination data distributions with regard to the number of processors allocated to each dimension and the block size of each dimension. Precautions are taken to ensure that the total number of processors is exactly 64 and the block size is chosen such that each processor contains a part of the array.

Frequently used patterns: The frequently used patterns include ring, nearest neighbor, hypercube, shuffle–exchange, and all–to–all patterns. These patterns are frequently used in real application programs since many algorithms have been designed using these patterns. A good connection scheduling algorithm should be able to handle these patterns effectively.

Table 1 shows the performance of algorithms for random communication patterns. The first column gives the number of connections in the random patterns. Columns 2, 3, and 4 give the multiplexing degrees obtained by using greedy, coloring, and ordered AAPC algorithms respectively. The multiplexing degrees shown are the average of 100 random patterns with specific number of connections. Column 5 shows the multiplexing degree obtained by a *combined algorithm*, which runs the coloring algorithm and the ordered AAPC algorithm and uses the better of the two results for scheduling. In compiled communication, more time can be spent to obtain better runtime network utilization. Hence, the combined algorithm can be used to obtain better results by the compiler. The percentage improvement achieved by the combined algorithm over the greedy algorithm is shown in the last column. We observe that the coloring algorithm is always better than the greedy algorithm and the AAPC algorithm is better than the other algorithms when the communication is dense. We can see that for sparse random patterns (100 - 2400 connections), the improvement varies from 3.8% to 7.2%. Larger improvements are obtained for dense communication patterns. For example, the combined algorithm improves multiplexing degree by 43.1% over the greedy algorithm for the all–to–all comunication pattern.

No. of Conn.	Greedy Alg.	Coloring Alg.	AAPC Alg.	Combined Alg.	Improv-ement
100	7.0	6.7	6.9	6.6	6.3%
400	16.5	16.1	16.5	15.9	3.8%
800	27.2	25.9	26.5	25.6	6.3%
1200	36.3	34.5	35.3	34.2	6.1%
1600	45.0	43.5	43.4	42.8	5.1%
2000	53.4	50.4	50.4	49.7	7.4%
2400	60.8	57.5	57.4	56.7	7.2%
2800	68.8	64.4	62.4	62.4	10.2%
3200	76.3	70.8	64	64	19.2%
3600	83.9	76.8	64	64	31.1%
4000	91.6	83.0	64	64	43.1%

Table 1: Performance for random patterns.

No. of Conn.	No. of Patterns	Greedy Alg.	Coloring Alg.	AAPC Alg.	Combined Alg.	Improv-ement
0 - 100	34	1.2	1.2	1.2	1.2	0.0%
101 - 200	50	5.9	4.9	4.8	4.6	28.3%
200 - 400	54	10.6	9.7	10.0	9.5	11.6%
401 - 800	105	17.7	15.9	16.0	15.5	14.2%
801 - 1200	122	31.7	28.7	28.6	27.6	14.9%
1201 - 1600	0	0	0	0	0	0%
1601 - 2000	15	46.3	42.8	35.1	35.1	31.9%
2001 - 2400	77	55.5	51.5	51.9	50.4	10.1%
2401 - 4031	0	0	0	0	0	0%
4032	43	92	83	64	64	43.8%

Table 2: Performance for data distribution patterns.

Table 2 shows the performance of the algorithms for random data redistribution patterns. The table lists the results for 500 random data redistributions. The first column lists the range of the number of connection requests in each pattern. The second column lists the number of data redistributions whose number of connection requests fell into the range. For example, the second column in the last row indicates that among the 500 random data redistributions, 43 of them results in 4032 connection requests. The results show that the multiplexing degree required to establish connections resulting from data redistribution is less than those required for random communication patterns. For data redistribution patterns, the improvement obtained by using the combined algorithm ranges from 10.1% to 31.9% for sparse communications, which is larger than the improvement obtained for random communication patterns. The only dense communication pattern that resulted from data redistribution is the all–to–all pattern.

Table 3 shows the performance for some frequently used communication patterns. In the ring and the nearest neighbor patterns, no conflicts arise in the links. However, conflicts arise in the communication switches. The performance gain is higher for these specific patterns when the combined algorithm is used.

Pattern	No. of Conn.	Greedy	Coloring	AAPC	Comb.	Improvement
ring	128	3	2	2	2	50%
nearest neighbor	256	6	4	4	4	50%
hypercube	384	9	7	8	7	28.6%
shuffle–exchange	126	6	4	5	4	50%
all–to–all	4032	92	83	64	64	43.8%

Table 3: Performance for frequently used patterns.

4 Compiled Communication vs Dynamic control

In this section, we compare the performance of the network under compiled communication with the performance of the network under a dynamic control mechanism is used. When using compiled communication, the control of the network is simple. The compiler schedules the communication using one of the algorithms described in the previous section. We use the combined algorithm in our simulation. At runtime, the network registers are loaded before the communication takes place, and thus all the paths for the communications are established before the processors start communicating. When using dynamically controlled communication, a path reservation protocol must be supported. Detailed discussion about the path reservation protocols can be found in [15, 17]. Next, we briefly describe the path reservation protocol used in our simulation study and then present the results of experimentation.

4.1 Dynamic Path Reservation Protocol

In addition to the optical data network, we assume that there is a *shadow network* which is used to exchange the control information needed to establish paths. We also assume that the same physical topology is used for the data and the shadow networks. The traffic on the shadow network consists of small control messages and thus is much lighter than the traffic on the data network. For this reason, electronic packet switching communication may be used for the shadow network. Alternatively, a virtual channel on the data network can be reserved exclusively for exchanging control messages.

The distributed reservation protocol can be descirbed as follows. When a processor generates a new request, it sends a reservation packet to the destination. The reservation packet goes through the network, reserving the available virtual channels along the path and carrying the information about the available virtual channels towards the destination. The reservation fails when there is no available virtual channel in some link along the path. When the reservation packet reaches the destination, the destination will select from the set of available virtual channels a channel to be used for the connection and send an acknowledge packet with this information to the source node. The acknowledgement packet follows the same path as the reservation packet in the reverse direction. Along the path, the acknowledgement packet releases the reserved virtual channels that are not selected and sets the switch according to the selected channel. When the acknowledgement packet reaches the source, the path for the communication request is established and the source node can start sending data. After the source node finishes sending, it releases the virtual channels by sending a release message to the destination.

4.2 Performance Evaluation

A cycle level network simulator for time-multiplexed 2-dimensional torus was developed to study the performance of compiled communication and dynamically controlled communication. The system parameters used in the performance evaluation are:

• *Topology*: 8×8 torus.
• *Routing algorithm*: XY routing is used and along each dimension messages follow the *Odd–Even shortest–path* routing, that is, a path from a node i to a node j follows the shortest path if the distance between i and j is not equal to $n/2$. If the distance is equal to $n/2$, then the path is established clockwise if i is odd and counter–clockwise if i is even.
• *Multiplexing degree*: For compiled communication, we assume that each link has enough virtual channels so that the set of arbitrary connections can be established simultaneously. For dynamically controlled communication, a fixed multiplexing degree must be used. The multiplexing degrees considered are 1, 2, 5 and 10.
• *Control packet transmission time*: 1 time slot.
• *Control packet processing time*: 1 time slot.
• *Control packet retransmit time*: a random number range from 1 to 29 time slots.
• *Processor mapping*: We assume that the physical identifiers of the nodes follow the row-major numbering and the mapping function between logical processor numbers and physical identifiers is the identity function.

Three application programs, namely GS, $TSCF$ and $P3M$, were used in this study. The GS benchmark uses Gauss–Siedel iterations to solve Laplace equation on a discretized unit square with Dirichlet boundary conditions. The $TSCF$ program simulates the evolution of a self–gravitating system using a self consistent field approach. $P3M$ performs particle–particle particle–mesh simulation. Table 4 describes the static communication patterns that arise in these programs. While GS and TSCF programs contain only one communication pattern each, P3M, which is a much larger program, contains five static communication patterns. All of the above patterns are in the main iterations of the programs.

Table 5 shows the communication time for these communication patterns. Listed in the tables are the size of the problem and the communication times (the unit of time used is a time slot) for compiled communication and dynamic communication. As mentioned earlier, we assume sufficient multiplexing degree to support all the patterns in compiled communication. Thus, no network reconfiguration is required to establish each pattern. For dynamic communication, we evaluated the performance of fixed multiplexing degree of 1, 2, 5 and 10. The size of the problem affects the message size except for the TSCF program. The following observations can be made from the results in Table 5. First, the compiled communication out–performs dynamic communication in all cases. The communication time taken using dynamic protocol was

Pattern	Type	Description
GS	shared array ref.	PEs are logically linear array, Each PE communicates with two PEs adjacent to it.
TSCF	explicit send/recv	hypercube pattern
P3M 1	data redistrib.	(:block, :block, :block) → (:, :, :block)
P3M 2	data redistrib.	(:, :, :block) → (:block, :block, :)
P3M 3	data redistrib.	(:block, :block, :) → (:, :, :block)
P3M 4	data redistrib.	(:, :, :block) → (:block, :block, :block)
P3M 5	shared array ref.	PEs are logically 3–D array, each PE communicates with 26 PEs surrounding it

Table 4: Communication pattern description.

2 to 20 times greater than the communication time associated with compiled communication. Larger performance gains are observed for communication with small message sizes (e.g., the TSCF pattern) and dense communication (e.g., the P3M 2 pattern). Second, the multiplexing does not always improve the communication performance for dynamic communication. For example, a multiplexing degree of 1 results in best performance for the pattern in GS. This is because the dynamic control must use a fixed multiplexing degree and is not able to adapt to the optimal multiplexing degree for a given communication pattern.

Pattern	Problem Size	Compiled Comm.	Dynamic Comm.			
			1	2	5	10
GS	64×64	35	105	118	171	251
	128×128	67	137	154	251	411
	256×256	131	265	304	411	731
TSCF	5120	19	344	268	270	300
P3M 1	$32 \times 32 \times 32$	831	3905	3625	2018	1861
	$64 \times 64 \times 64$	6207	12471	10754	10333	9619
P3M 2,3	$32 \times 32 \times 32$	382	9999	6094	4661	4510
	$64 \times 64 \times 64$	2174	17583	14223	10360	9320
P3M 4	$32 \times 32 \times 32$	457	3309	2356	1766	1722
	$64 \times 64 \times 64$	3369	9161	7674	7805	7122
P3M 5	$32 \times 32 \times 32$	40	583	374	371	480
	$64 \times 64 \times 64$	68	673	457	445	505

Table 5: Communication time for static patterns.

In this study we observe that for a well designed parallel program, the fine grain communications that result from shared arrays usually cause sparse communication with small message sizes. For a communication system to efficiently support such communication, the system should have small latency. Optical networks that use dynamic control have a large startup overhead. Thus they cannot support this type of communication efficiently. As shown in our simulation results, with compiled communication the startup overhead is eliminated and fine grain communication is performed efficiently. We also observe that the data redistribution usually results in dense communication with large message sizes. In this case, the control overhead does not significantly affect communication efficiency. However, dense communication results in large number of conflicts in the system, and the dynamic control system may not be able to resolve these conflicts efficiently. Our simulation confirms the conclusion in [8] that static management of the dense communication patterns results in large performance gains.

In summary, compiled communication achieves high performance for all types of static patterns. Four factors contribute to the performance gain. First, compiled communication eliminates dynamic control overhead. This is most significant for communication with small message sizes, where the overhead in dynamic communication is large compared to the communication time. Second, compiled communication takes the whole communication pattern into consideration, while dynamic communication, which considers the connection requests one by one, suffers from the head–of–line effect [16]. Third, the off–line message scheduling algorithm further optimizes the communication efficiency for compiled communication. Fourth, compiled communication allows the system to use various multiplexing degrees for different communication patterns. In dynamic communication, control mechanism with variable multiplexing degree is very difficult to implement and results in large overhead. Each communication pattern has an optimal multiplexing degree. If the system provides a multiplexing degree smaller than the optimal value, many messages will be blocked and the communication will not be efficient. If the system provides a multiplexing degree larger than the optimal value, bandwidth will be lost due to the unused time slots.

5 Conclusion

Connection oriented communication is needed in all–optical networks to avoid the inefficiency of converting between optical and electronic signals at intermediate nodes. However, the overhead of dynamically controlling connection oriented communication may be relatively large, especially for communication patterns that involve short messages. Compiled communication techniques may be used efficiently to reduce communication overhead for static patterns, which account for a large portion of the communication in large parallel scientific applications.

In this paper, we studied the performance of compiled communication for static patterns in time-multiplexed optical networks. Efficient algorithms for connection scheduling were proposed and evaluated. We conclude that compiled communication is an efficient communication model for all–optical networks in multiprocessor environments. We are currently investigating techniques to use statically determined multiplexed sequences for communication patterns that cannot be determined at compile time.

References

[1] C. A. Brackett, "Dense wavelength division multiplexing networks: Principles and applications," *IEEE Journal on Selected Areas of Communications*, Vol. 8, Aug. 1990.

[2] M. Bromley, S. Heller, T. McNerney and G. L. Steele, Jr. "Fortran at Ten Gigaflops: the Connection Machine Convolution Compiler," in *Proc. of SIGPLAN'91 Conf. on Programming Language Design and Implementation.* June, 1991.

[3] F. Cappelllo and C. Germain. "Toward high communication performance through compiled communications on a circuit switched interconnection network," in *Proceedings of the Int'l Symp. on High Performance Computer Architecture*, pages 44-53, Jan. 1995.

[4] I. Chlamtac, A. Ganz and G. Karmi. "Lightpath Communications: An Approach to High Bandwidth Optical WAN's," *IEEE Trans. on Communications*, Vol. 40, No. 7, July 1992.

[5] A. Feldmann, T. M. Stricker and T. E. Warfel. "Supporting sets of arbitrary connections on iWarp through communication context switches," in *5th ACM Symp. on Algorithms and Architectures*, pages 203-212, 1993

[6] T. Gross. "Communication in iWarp Systems," in *Proceedings Supercomputing*'89, pages 436–445, ACM/IEEE, Nov. 1989.

[7] M. Gupta and P. Banerjee. "A Methodology for High–Level Synthesis of Communication on Multicomputers," in *International Conference on Supercomputing*, 1992.

[8] S. Hinrichs, C. Kosak, D.R. O'Hallaron, T. Stricker and R. Take. "An Architecture for Optimal All–to–All Personalized Communication," in *6th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 310-319, June 1994.

[9] S. Hinrichs. "Compiler Directed Architecture–Dependent Communication Optimization," *Ph.D dissertation*, School of Computer Science, Carnegie Mellon University, 1995.

[10] D. Lahaut and C. Germain, "Static Communications in Parallel Scientific Programs," in *PARLE'94, Parallel Architecture & Languages*, Athen, Greece, July 1994.

[11] J. Li and M. Chen. "Compiling Communication –efficient Programs for Massive Parallel Machines," *IEEE Trans. on Parallel and Distributed Systems*, 2(3):361-376, July 1991.

[12] R. Melhem, "Time–Multiplexing Optical Interconnection Network; Why Does it Pay Off?" in *Proceedings of the 1995 ICPP workshop on Challenges for Parallel Processing*, pages 30–35, August 1995.

[13] C. Qiao and R. Melhem, "Reconfiguration with Time Division Multiplexed MIN's for Multiprocessor Communications." *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 4, April 1994.

[14] C. Qiao and R. Melhem. "Reducing Communication Latency with Path Multiplexing in Optically Interconnected Multiprocessor Systems," in *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 34-43, January 1995.

[15] C. Qiao and Y. Mei, "Wavelength Reservation Under Distributed Control," in *IEEE/LEOS summer topical meeting: Broadband Optical Networks*, August 1996.

[16] K.M. Sivalingam and P.W. Dowd, "Latency hiding strategies of pre–allocation based media access protocols for WDM photonic networks," in *Proc. 26th IEEE Simulation Symposium*, pages 68 – 77, Mar. 1993.

[17] X. Yuan, R. Gupta and R. Melhem, "Distributed Control in Optical WDM Networks," in *IEEE Conf. on Military Communications*, Oct. 21-24, 1996, McLean, VA.