Automatic Transformation for Overlapping Communication and Computation

Changjun Hu, Yewei Shao, Jue Wang, and Jianjiang Li

School of Information Engineering, University of Science and Technology Beijing
NO.30 Xueyuan Road, Haidian District, Beijing, P.R.China
huchangjun@ies.ustb.edu.cn, yeweishao@gmail.com,
ncepu5@gmail.com, jianjiangli@gmail.com

Abstract. Message-passing is a predominant programming paradigm for distributed memory systems. RDMA networks like infiniBand and Myrinet reduce communication overhead by overlapping communication with computation. For the overlap to be more effective, we propose a source-to-source transformation scheme by automatically restructuring message-passing codes. The extensions to control-flow graph can accurately analyze the message-passing program and help perform data-flow analysis effectively. This analysis identifies the minimal region between producer and consumer, which contains message-passing functional calls. Using inter-procedural data-flow analysis, the transformation scheme enables the overlap of communication with computation. Experiments on the well-known NAS Parallel Benchmarks show that for distributed memory systems, versions employing communication-computation overlap are faster than original programs.

Keywords: Parallel compiling; Communication optimization; Control-flow analysis; Source-to-source transformation.

1 Introduction

Message-passing is widely used in parallel programs and is a standard interface for message-passing parallel programs written in C, C++, or Fortran that supports point-to-point communications (send, receive, isend, ireceive) and collective operations (broadcast, gather, scatter, alltoall, alltoallv). The algorithm presented in this paper is applicable to message-passing codes. Our platform is a set of sixteen processor nodes, connected with an infiniBand switch. Current infiniBand switches have Remote Direct Memory Access (RDMA) capability and support that non-blocking communication can progress concurrently with computation.

The benefit of overlapping communication and computation in parallel computing has been extensively studied in the past decade. We can classify previous works into three kinds. Some of researches are achieved by compiled methods [1, 2, 3, 4, 5]; some of them have been performed in the field of Global Address Space languages [6, 7, 8] or achieved by particular hardware [9, 10, 11]. However, these techniques may be effective for overlapping communication and computation only in a single loop. In this paper, we present a transformation scheme to overlap communication with computation using inter-procedural data-flow analysis.

Compared with previous researches, our main contributions are as follows:

- Using inter-procedural data-flow analysis to find the minimal region from producer to consumer in context of message-passing programs.
- We propose a transformation scheme to overlap communication with computation.
- We evaluate some NAS benchmarks to validate our transformation.

The rest of this paper is organized as follows. Section 2 gives the algorithm to create the control-flow graph for message-passing programs. Section 3 describes a source-to-source transformation scheme to optimize the parallel programs. Section 4 evaluates the performance of NAS benchmarks using our transformation algorithm. Section 5 places this paper in the context of related work. Section 6 presents conclusions.

2 Control-Flow Graph for Message-Passing Program

The compiler must characterize the control-flow and the data-flow of programs, so that the programs can be optimized in next step. It is regrettable that the previous control-flow graph does not consider the message-passing call, which can result in less precise and even incorrect analysis results. To resolve these problems, Shires et al. [12] give an extension to the control-flow graph called the MPI-CFG. A motivation example will be given in figure1, which is a generic code segment in SPMD (Single Process, Multiple Data) parallel program. The array dum is communicated between statement S1 and S2. In figure 2 MPI-CFG contains control-flow edges represented with solid lines and a communication edges represented with dash lines. This is the start point of our work.

```
        if ( rank == 0 ) then
           do j=jst,jend
              dum(1,j)=g(1,nx,j,k)
                ……
            end do
S1:         MPI_SEND( dum(1,jst), 5*(jend-jst+1), dp_type,
                    south, from_n, MPI_COMM_WORLD,
                    IERROR )
        else
S2:          MPI_RECV( dum1(1,jst), 5*(jend-jst+1), dp_type,
                    north, from_n, MPI_COMM_WORLD,
                    status, IERROR )
              ……
         end if
        …=dum(1,jst)
```

Fig. 1. A code segment of SPMD program

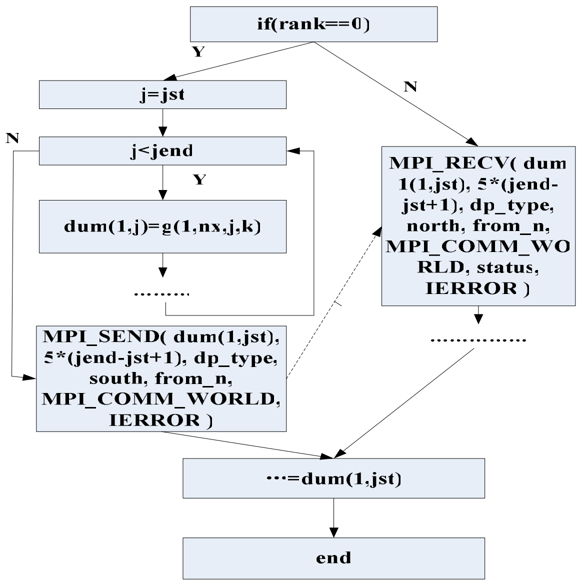

if(rank==0)
Y
N
j=jst
N
j<jend
Y
dum(1,j)=g(1,nx,j,k)
....
MPI_SEND(dum(1,jst), 5*(jend-jst+1), dp_type, south, from_n, MPI_COMM_WORLD, IERROR)
MPI_RECV(dum 1(1,jst), 5*(jend-jst+1), dp_type, north, from_n, MPI_COMM_WORLD, status, IERROR)
....
···=dum(1,jst)
end

Fig. 2. The MPI-CFG of the code segment presented in Figure1

3 The Overlap of Communication and Computation

There are two challenges involved in data-flow analysis. The first challenge is to identify message-passing variables and characterize corresponding data accesses. The second challenge is that producer-consumer data-flow analysis need to be performed to ensure communication-computation overlap.

3.1 Inter-procedural Data-Flow Analysis for Message-Passing Programs

To characterize and analyze data accesses in message-passing program, we perform the inter-procedural dataflow analysis for message-passing variables. Message-passing variable is defined as data that may be communicated or related to communication data, such as the parameters in message-passing call and the compiler identifies these message-passing variables using the algorithm described in Figure 3. In Figure 3, we first locate communication statements, and then obtain their parameters and functional calls. If the parameters are communicated through communication statement, we add these parameters into our Message-passing Variables list. If the communication statement has function calls, we should go into these functions and get parameters from these calls, and then add these parameters into our list. Finally, our list is a set of variables that may be communicated or related to communication data.

```
Algorithm list_message-passing_variables
Input :
        A - An MPI program.
Output:
        V – A List of Message-passing Variables in A
Begin
(1)  Set V= $\Phi$
(2)  do $\forall$Comm(L), which is an message passing call in A, L is the
        set of  variables in Comm(L)
(3)    do $\forall$Pr, Pr is a procedure defined within A
(4)        do $\forall$F, F is function call with Pr
(5)         do $\forall$Pa, Pa is a parameter of F
(6)           if ( Pa $\in$ L)
(7)             Let FP be the procedure that defines F
(8)             Let PA be the procedure parameter of FP
                   corresponding to function parameter Pa
(9)             set V = V $\cup$ PA
(10)           end if
(11)         end do
(12)        end do
(13)    end do
(14)  end do
```

Fig. 3. Algorithm to create list of message-passing variables

```
Algorithm : Find the minimal region from producer to consumer
Input:
              V – A List of Message-passing Variables in A
              MPI control flow graph signed RSD
              Data Dependence
Output:
              PP – the Set of the place that produce variables in V
              PC – the Set of the place that consume variables in V
              Region  –  the Set of region, while Region = {minPlace_P,
              minPlace_Q   minPlace_P $\in$ PP, minPlace_Q $\in$ PC}
Begin
       (1)    Initialize PP = $\Phi$, PC = $\Phi$,  Region = $\Phi$
       (2)      do $\forall$ variable L, L $\in$ V
       (3)           Set LastDistance = MAX (LastDistance is the Distance
                     between two places that we want to know, MAX is just a single)
       (4)           Set minPlace_P = 0( minPlace_P is the place of producer that
                     the minimal place from producer to communication statement)
       (5)           Set minPlace_Q = 0(minPlace_Q is the place of consumer that
                     the minimal place from communication statement to consumer)
       (6)           Set W(L) = set of produce L
       (7)           do $\forall$ P, P $\in$ W(L)
       (8)                  CurrentDistance = getDistance ( P, Comm(L))
                            (Comm(L) is the communication statement about
                                   variable L, getDistance(P,Comm(L)) is a function to
                                   compute the distance from p to Comm(L))
       (9)                  if ( CurrentDistance < LastDistance)
       (10)                        LastDistance = CurrentDistance
       (11)                        minPlace_P = P
       (12)                 PP = PP $\cup$  minPlace_P
       (13)          end do
       (14)          LastDistance = MAX
       (15)          Set FU(L) = set of future use of L
       (16)          do $\forall$ Q, Q $\in$ FU(L)
       (17)                 CurrentDistance = getDistance ( Comm(L), Q)
       (18)                 if ( CurrentDistance < LastDistance)
       (19)                        LastDistance = CurrentDistance
       (20)                        minPlace_Q = Q
       (21)                 PC = PC $\cup$  minPlace_Q
       (22)          end do
       (23)          Region = getRegion ( minPlace_P, minPlace_Q )
       (24)     end do
```

Fig. 4. Algorithm to construct the minimal region from producer to consumer

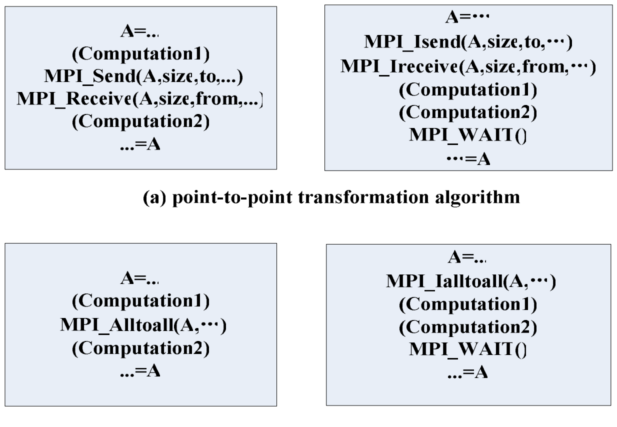
3.2 Constructing the Minimal Region from the Producer to Consumer

In this section we give producer-consumer relationship analysis which can be applied at any level. If the statement S1 precedes S2 in execution order, then S1 < S2. Dependence between two statements in program is relation that constrains their execution order and control dependence constrain that arises from control-flow graph. Data dependence arises from flow of data. Therefore, we will give the types of producer-consumer dependencies. If S1 < S2, and S1 sets value and later S2 uses it, then call it producer-consumer. If S1 < S2, and S1 uses some variable value and S2 sets it, then call it anti producer-consumer. We treat producer-consumer and anti producer-consumer differently.

To expose the maximum available opportunity for overlapping communication with computation, the algorithm shown in Figure 4 resolves the minimal region from producer to consumer, which contains the communication function calls. Each variable could be produced and be consumed in multi places in the program, and we only pay attention to the minimal region from the producer to the consumer. Getting variable from the Message-passing Variables list which is described in Figure 3, we locate the places that produce this variable and then we select the place called minPlace_P that is the minimal place from the producer to the communication statement which contains variable. Then we choose the place called minPlace_Q that is the nearest place consuming the variable after communication statement. Finally, our minimal region is from minPlace_P to minPlace_Q.

3.3 Transformation Algorithm

We classify the communication patterns into two cases, blocking communication and non-blocking communication. To overlap communication and computation, messages are initiated early using non-blocking sends/receives and completed just before the consumption point at the receiver with a wait.



```
A=...
(Computation1)
MPI_Send(A,size,to,...)
MPI_Receive(A,size,from,...]
(Computation2)
...=A
```

```
A=···
MPI_Isend(A,size,to,···)
MPI_Ireceive(A,size,from,···)
(Computation1)
(Computation2)
MPI_WAIT()
···=A
```

(a) point-to-point transformation algorithm

```
A=...
(Computation1)
MPI_Alltoall(A,···)
(Computation2)
...=A
```

```
A=...
MPI_Ialltoall(A,···)
(Computation1)
(Computation2)
MPI_WAIT()
...=A
```

(b) collective transformation algorithm

Fig. 5. Transformation algorithm for overlapping communication and computation

MPI provides a direct interface to non-blocking point-to-point operations, while non-blocking collective operations to overlap communication and computation are not directly supported by the MPI standard. For blocking communication, we change it into non-blocking communication using the techniques of Hoefler et al [13, 14]. In other words, we change MPI_Send into MPI_Isend, MPI_Receive into MPI_Ireceive, and get the non-blocking communications including MPI_Ibcast, MPI_Igather, MPI_Iscatter, MPI_Ialltoall and MPI_Ialltoallv. Since collective communication and point-to-point communication are used in a different way, they should be considered separately.

We give the transformation algorithm presented in Figure5. The algorithm based on the minimal region from producer to consumer described in Figure5 and guarantee the maximization of overlapping communication and computation.

4 Experimental Results

To evaluate the effect of our strategy, the performance comparisons between the original program and our optimized program using our transformation algorithm. The experimental environment is a set of sixteen processor nodes, connected with a high-performance infiniBand switch. Each node has an Intel Xeon 3.0G processor with 1024KB L2 Cache, and the switch has a Remote Direct Memory Access (RDMA) capability, whereby non-blocking message-passing communication can progress concurrently with computation. The Operating System is RedHat Linux version FC3, with Kernel 2.6.9 and we use MVAPICH2 1.0[15] for communication over InfiniBand. Time is measured by inserting MPI_Wtime() calls before and after the region we want to execute.

To validate the transformation scheme, we design several experiments implementing the algorithm of overlapping communication and computation using data-flow analysis to apply to NAS benchmarks. The NAS parallel benchmarks [16] are a set of programs designed originally to evaluate supercomputers. We use NPB 2.4 [17] implementation written in MPI and give some experiments based on the NAS parallel programs which confirm to our algorithm, such as LU, IS, BT, MG.

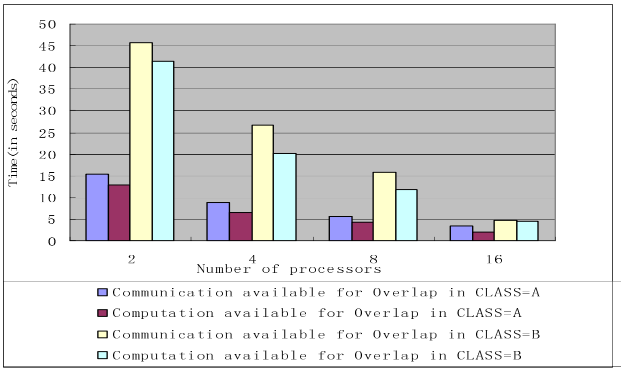


Fig. 6. Performance in LU benchmark

Figure6 shows the performance of LU benchmark before and after optimized program. The transformed program succeeds in tolerating the communication latency and reducing the execution time by almost from 10% to 17% going from two to sixteen nodes both in class A problem size, while reducing time from 5% to 17% in class B problem size.

Figure7 shows the time taken in communication and computation available for overlap both in class A problem size and class B problem size in LU benchmark. The time taken in communication available for overlap is close to the time taken in computation and it occupies a large proportion in the parallel program. Therefore, optimized program of LU benchmark succeeds in reducing the execution time taken in parallel program.

Figure8 shows the performance of MG benchmark while Figure9 shows the communication and computation available for our algorithm. The time taken in communication available for overlap is much larger than the time taken in computation. Although the time spent in communication occupies a large proportion in the parallel program, the actual time spent in overlapping is relatively low. In Figure 8 the transformation algorithm reduces the execution time only from 5% to 8%. Therefore,

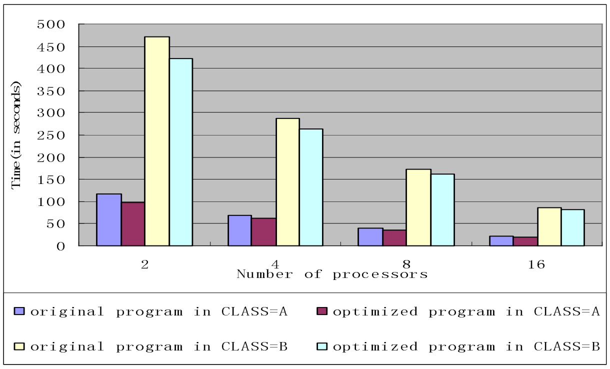


Fig. 7. Communication and computation available for overlapping in class A and class B problem size (LU benchmark)

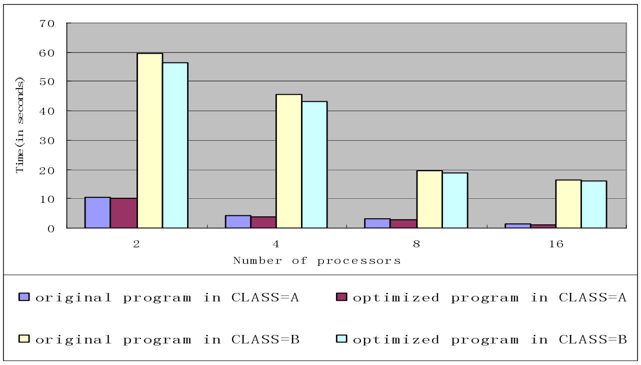


Fig. 8. Performance in MG benchmark

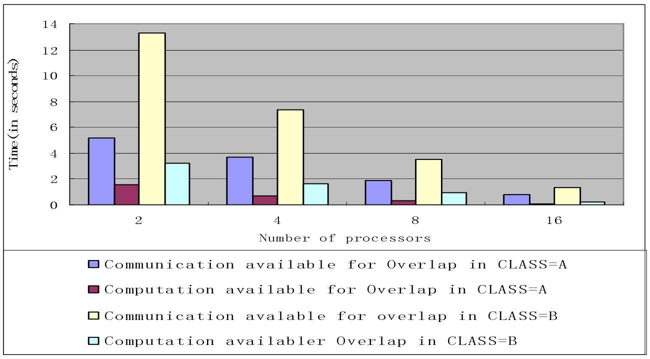


Fig. 9. Communication and computation available for overlapping in class A and class B problem size (MG benchmark)

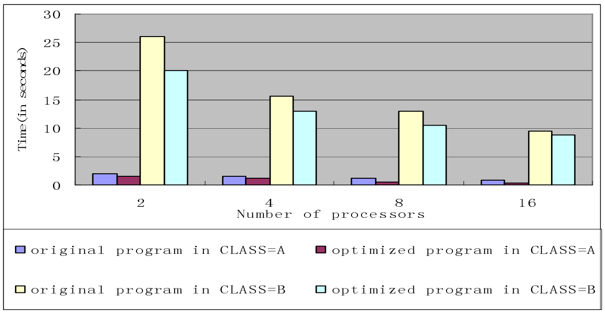


Fig. 10. Performance in IS benchmark

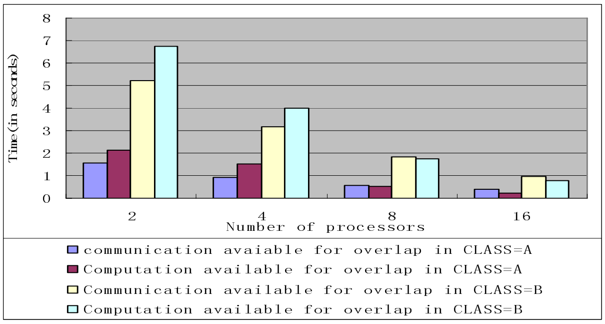


Fig. 11. Communication and computation available for overlapping in class A and class B problem size (IS benchmark)

transformation algorithm achieves good performance only if the time taken in communication available for overlap is close to the time taken in computation. The time taken in available to overlap occupies relatively low in the whole parallel program execution in BT benchmark, so it does not show an obvious result in this benchmark

Figure10 shows the performance of IS benchmark while Figure11 shows the communication and computation available for our algorithm. It is obviously seen from Figure 11 that the descent speed of computation is faster than the descent speed of communication in going from two to sixteen nodes. On two to four nodes, the time taken in communication available for overlap is lower than the time taken in computation, while on eight to sixteen nodes the time taken in communication available for overlap is larger than the computation. Since the time taken in communication available for overlap is close to the time taken in computation and the time occupies a large proportion in the parallel program execution, Figure 10 shows that the optimized program can reduce the time from 11% to 39% in class A problem size and from 17% to 22% in class B problem size.

From the above observations, the improvement of performance depends on the following aspects. The first aspect is the proportion of actual time for overlapping occupied in parallel programs. The second one is the time taken in communication should be close to the time taken in computation available for overlap.

Even after our overlap communication and computation, the communication volume is still very high. However, overlap allows us to tolerate the communication latency considerably.

5 Related Work

Control-flow frameworks have been extended by Shires et al. [12], which represents the semantics of message-passing by including communication edges between message-passing procedure calls. This control-flow graph can not describe non-blocking communication accurately. To resolve this problem, our analysis contains inter-procedural in the control-flow graph and inter-procedural in the data-flow graph.

Reducing communication latency using overlap communication and computation has been used in the past decade. HPF compilers [18] proposed a notion of posting of sends as early as possible and receiving as late as possible in order to overlap communication with computation. Some later approaches have suggested the overlapping of communication and computation [2, 19, 20, 21, 22], but they are limited to overlap them in a single loop.

Hoefler et al. [13, 14] gives non-blocking collective operations which are obvious extensions to message-passing. Kennedy et al. [23] presents a communication placement framework that reduces communication latency. The difference between us is that the communication placement can be determined by a sequence of simple unidirectional analyses while we add communication edges and use inter-procedural analysis in control-flow graph. This is the important starting points for our work. In our previous work [24], we pipelined an irregular loop by splitting inspector phase and using corresponding dependence analysis.

To the best of our knowledge, this paper presents the first approach to overlap communication and computation by the inter-procedural analysis of message-passing programs. Algorithm could be used in both point-to-point communications and collective operations.

6 Conclusions

In this paper, we present a transformation scheme to achieve overlapping communication and computation based on inter-procedural data-flow analysis. The data-flow analysis gives the RSD of each variable in message-passing calls and the minimal region from producer to consumer. Finally, we give transformation scheme to accomplish our optimization. To study the impact of our optimization, we give some experiment results to illustrate that our strategy is useful for improving the performance of the message-passing programs.

Acknowledgments

The research is partially supported by the Key Technologies Research and Development Program of China under Grant No.2006038027015, the Hi-Tech Research and Development Program (863) of China under Grant No. 2006AA01Z105 and No. 2008AA01Z109 Natural Science Foundation of China under Grant No.60373008, and by the Key Project of Chinese Ministry of Education under Grant No. 106019 and No.108008.

References

1 Basumallik, A., Eigenmann, R.: Optimizing Irregular Shared-Memory Applications for Distributed-Memory Systems, PPOPP, New York, USA, March 29-31 (2006)
2 Fishgold, L., Danalis, A., Pollock, L., Swany, M.: An automated approach to improve communication-computation overlap in cluster. NIC Series, vol. 33, pp. 481–488. John von Neumann Institute for computing, Julich (2006)
3 Danalis, A., Pollock, L., Swany, M.: Automatic MPI application transformation with ASPhALT. IEEE, Los Alamitos (2007)
4 Danalis, A., Kim, K.-Y., Pollock, L., Swany, M.: Transformations to Parallel Codes for Communication-computation Overlap. ACM, New York (2005)
5 Kreaseck, B., Carter, L., Casanova, H., Ferrante, J.: On the Interference of Communication on Computation in Java. IEEE, Los Alamitos (2004)
6 El-Ghazawi, T.A., Carlson, W.W., Draper, J.M.: UPC specification, v. 1.1 (2003), http://upc.gwu.edu/documentation
7 Hilfinger, P., Bonachea, D., Gay, D., Graham, S., Liblit, B., Pike, G., Yelick, K.: Titanium language reference manual. tech report ucb/csd-01-1163, u.c. berkeley (November 2001)
8 Numrich, R.W., Reid, J.K.: Co-Array Fortran for parallel programming. ACM FortranForum 17(2), 1–31 (1998)
9 Goumas, G., Sotiropoulos, A., Koziris, N.: Minimizing completion time for loop tiling with computation and communication overlapping. In: Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001), April 23–27, 2001, p. 39. IEEE Computer Society, Los Alamitos (2001)
10 Gupta, S.K.S., Huang, C.-H., Sadayappan, P., Johnson, R.W.: Atechnique for overlapping computation and communication for block recursive algorithms. Concurrency: Practiceand Experience 10(2), 73–90 (1998)

11 Sohn, A., Biswas, R.: Communication studies of dmp and smp machines. Technical Report NAS-97-005,NASA Ames ResearchCenter (March 1997)
12 Shires, D., Pollock, L., Sprenkle, S.: Program Flow Graph Construction for Static Analysis of MPI programs. In: International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 1999) (June 1999)
13 Hoefler, T., Lumsdaine, A., Rehm, W.: Implementation and Performance Analysis of Non-Blocking Collective operations for MPI. In: SC 2007, Reno, Nevada, USA, November 10-16 (2007)
14 Hoefler, T., Lumsdaine, A.: Optimizing non-blocking collective operations for infiband (April 2008); Accepted for publication at the CAC 2008 in conjunction with the IDPDS 2008
15 `http://mvapich.cse.ohio-state.edu`
16 `http://www.nas.nasa.gov/software/NPB`
17 Bailey, D., Barszcz, E., Barton, J., Browning, D., Carter, R., Dagum, L., Fatoohi, R., Fineberg, S., Frederickson, P., Lasinski, T., Schreiber, R., Simon, H., Venkatakrishnan, V., Weeratunga, S.: The NAS parallel benchmarks, Tech. Rep. RNR-94-007, NASA Ames
18 Gupta, M., Miskiff, S., Schonberg, E., Seshadri, V., Shields, D., Wang, K., Ching, W., Ngo, T.: An HPF compiler for the IBM SP2. In: Proceedings of Supercomputing 1995, San Diego, CA (1995)
19 Ishizaki, K., Komatsu, H., Nakatani, T.: A loop transformation algorithm for communication overlapping. International Journal of Parallel Programming 28(2), 135–154 (2000)
20 Tseng, E.H.Y., Gaudiot, J.L.: Communication generation for aligned and Cyclic(k) distributions using integer lattice. IEEE Transactions on Parallel Distributed Systems 10(2), 136–146 (1999)
21 Lancu, C., Husbands, P., Chen, W.: Message Strip Mining Heuristics for High Speed Networks. In: VECPAR (2004)
22 Bell, C., Bonachea, D., Nishtala, R., Yelich, K.: Optimizing Bandwidth Limited Problems Using One-Side communication and overlap. In: 20th International parallel & Distributed Processing Symposium (IPDPS) (2006)
23 Kennedy, K., Sethi, A.: A Communication Placement Framework with Unified Dependence and Data-flow Analysis. In: Proceeding 3rd International Conference on High Performance Computing, December 19-22, 1996, pp. 201–208 (1996)
24 Hu, C., Yao, G., Wang, J., Li, J.: OpenMP Extensions for Irregular Parallel Applications on Cluster. In: Chapman, B.M., Zheng, W., Gao, G.R., Sato, M., Ayguadé, E., Wang, D. (eds.) IWOMP 2007. LNCS, vol. 4935. Springer, Heidelberg (2008)