DOI 10.12694/scpe.v18i4.1335
Scalable Computing: Practice and Experience ISSN 1895-1767
Volume 18, Number 4, pp. 375–385. http://www.scpe.org © 2017 SCPE

DATA FLOW ANALYSIS OF MPI PROGRAM USING DYNAMIC ANALYSIS TECHNIQUE WITH PARTIAL EXECUTION

K. B. MANWADE[*] AND D. B. KULKARNI[†]

Abstract. Message Passing Interface (MPI) is a dominant parallel programming paradigm. MPI processes communicate with each other by sending or receiving messages through communication functions. The application's communication latency will be less if processes are scheduled on nearest cores or nodes and communication latency will be more if processes are scheduled on farthest cores or nodes.The communication latency can be reduced by using topology-aware process placement technique. In this technique, MPI processes are placed on the nearest cores if they have more communication between them. To find the communication pattern between processes, analysis of MPI program is required. Various techniques like static, symbolic and dynamic analysis are available for finding communication pattern of MPI program. These techniques are either taking more time for analysis or fail to find correct communication pattern. In this paper, we have proposed DAPE (Dynamic Analysis with Partial Execution) technique for analysis of MPI program, which finds correct communication pattern in less time as compared to existing techniques. The experimental results show that the proposed technique outperforms over the existing techniques.

Key words: Topology-aware process placement, Communication pattern of MPI program, Dataflow analysis of MPI program

AMS subject classifications. 15A15, 15A09, 15A23

1. Introduction.

MPI [1] is a dominant message passing paradigm used for parallel programming. Formerly it was used for computer systems where every node was having a single processor on which a single process could be executed. The development of multi and many core technologies made scheduling of MPI processes more difficult. Nowadays, scheduling of more than one process on multi and many core nodes is possible. For such nodes, data localities need to be considered. If the two processes have more communication between them then scheduling them on two nearest cores improves the performance of a program. This leads to the topology-aware process placement [2] concept. The topology aware process placement involves three steps 1) finding communication pattern or topology of a program 2) finding architectural details or topology of nodes 3) scheduling processes on nodes. To find a communication pattern of the program, its analysis is required. In literature, various techniques such as static analysis, symbolic analysis, and dynamic analysis [3] [4] [5] are proposed for the analysis of MPI program. The data structure DFG (Data Flow Graph) represents the flow of data in the program through different functions and processes. Construction of DFG for MPI program is challenging because of its dynamic and SPMD nature. Processes of MPI program can communicate through communication functions in which receiver, sender, tag, communicator, size of the message, an address of message buffer, MPI data type, root process for collective function etc are mentioned. If these functions contain variable or MPI "wildcard" variables (MPI_ANY_SOURCE, MPI_ANY_TAG etc) then identifying the correct communication pattern using static analysis is difficult. The static analysis technique takes less time for analysis but communication patterns may not be correct. Therefore to find correct communication pattern dynamic analysis of the program should be performed. In this technique first run of the program is required which takes more time for the large program. The symbolic analysis is used at compile time and symbolic presentation of each instruction is required. In this paper, we proposed a new technique called dynamic analysis with partial execution, which detects correct communication pattern in less time as compared to existing techniques.

This paper is organized as follows: Section 2 presents work related to MPI program analysis. Section 3 reviews the concept of MPI program analysis, techniques of program analysis and motivation of the proposed work. The concept of partial execution and its assumptions are presented in section 4. The details of proposed technique are given in section 5. The performance comparison of the proposed technique with existing techniques is presented in section 6. Finally, in section 7, the conclusions are noted.

[*]Ph.D.Research Center:-Department of Computer Science and Engineering, Walchand College of Engineering, Sangli, Maharashtra, India. University:- Shivaji University, Kolhapur, Maharashtra, India. (mkarveer@gmail.com).

[†]Department of Information Technology, Walchand College of Engineering, Sangli, Maharashtra, India. (d_b_kulkarni@yahoo.com).

2. Related work. In [6], authors have applied the concept of static analysis to MPI program. They have constructed CFG (Control Flow Graph) for MPI programs, then the graph is converted into MPI-CFG where MPI communication function calls are analyzed and corresponding lines are joined between the nodes of CFG. In an analysis of MPI communication function sender, receiver, tag, communicator parameters etc are considered. They have matched sender and receiver by using three categories; explicitly identifier of the process, a variable containing process identifier and MPI "wildcard" variables for process identifier. To match sender and receiver processes, the first case uses constant identifier, the second case uses substitution values and the third case uses annotation on MPI-CFG graph. In [7], authors have constructed CFG for MPI program and by using fan IN, fan OUT notation the communication pattern of processes have been identified. For example, if sender process X and receiver Y are mapping in the record of both IN and OUT then it is assumed that both X and Y have communication between them otherwise not. For matching the sender and receiver process approximation method is used in [8]. Two approaches are used for this purpose; constant propagation and static slicing of CFG for non-constant propagation in MPI functions.

In [9] authors used two phases of analysis like (i) static pass to check whether the number of completed calls is matching with non-blocking calls (ii) static analysis to check whether the sequence of all collective calls is matched with all possible execution paths. These two phases detect an incorrect collective pattern in MPI program which leads to deadlock. The instrumentation overhead of these phases is very low but its functionality is limited to detection of deadlock in the program.

In [10] authors have used parallel version of static analysis with a fixed number of processes and they have extended functionality of Fuse framework to parallel implementation. The compositional approach is used which extend sequential data flow analysis to work with MPI program.

Dynamic analysis tools like ISP [11], DAMPI [12] are available for analysis of MPI applications. In these tools, program communication pattern is constructed by matching sender and receiver of the messages. These tools explore the non-determinism in the program but are used for performance analysis only. These tools are not used for detecting communication pattern of the processes. Also, tools like MPIPP [13] use dynamic analysis. The communication pattern determined by profiling MPI program takes more time for analysis. The MPI program profiling time is reduced in FACT [14] tool by using time-slicing method but it is facing the problem of non-determinism.

In [15] authors, reduced original program to program slice through static analysis and execute the program slice to acquire communication trace. The program slices preserve all variables and statements in the original program. For sliced program compile time analysis is performed. They have implemented FACT tool and evaluated with seven NBP programs as well as the sweep3D program.

In [16] authors implemented MOPPER deadlock detection tool which takes MPI program as input and produced deadlock detection as output. MOPPER first compile MPI program and then execute it using ISP tool. The ISP tool outputs a canonical trace of the input program along with the matches-before partial order. MOPPER then computes the over-approximation as match-setapp. Then both match-before and match-setapp are passed to SAT solver to find the deadlock in the communication.

The symbolic analysis method is used for MPI program analysis in [17] [18]. All MPI instructions are represented as a sequence of symbols by using 'instruction type'. By using this concept message blocks are identified. Then using symbolic representation CFG is constructed for the program. The combination of both control and data flow graph (CDFG) is used in [19] for MPI program along with SUIF [20] compiler framework. They have identified three types of communication patterns such as static, persistent and dynamic. To identify communication pattern of application symbolic analysis technique is used in TASS [21].

In this paper, we have proposed a novel method based on partial execution of a program which overcomes the limitations of existing techniques.

3. Techniques For MPI program analysis. The taxonomy of MPI program analysis techniques is shown in figure 3.1.

3.1. Static Analysis of Programs. Static analysis technique predicts program behavior like correctness of the program, control flow in the program, logical errors in the program etc without executing the program. In this technique, control flow graph is generated with help of control flow instructions in the program. By matching send instruction with corresponding receive instruction in the control flow graph, the communication

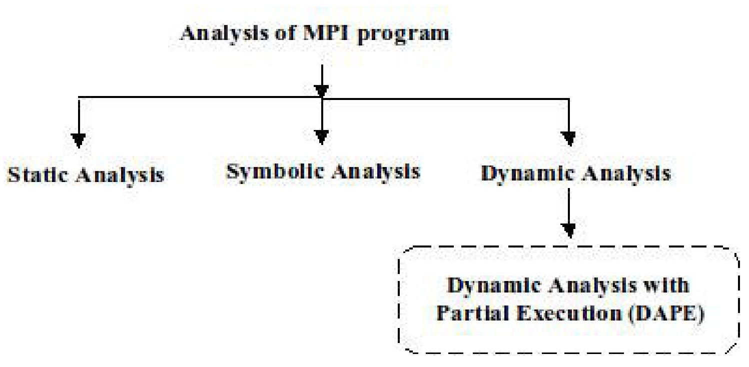


Fig. 3.1. *Techniques for MPI program analysis*

pattern of the program is found. The time efficiency of this technique is more as it finds a communication pattern without executing MPI program. This technique finds communication pattern in less time. It fails to find correct communication pattern in case of following conditions:

1. MPI_Send(...,...,...,dest,...,...)/ MPI_Recv(...,...,...,src,...,...):
   where 'dest' and 'src' are the variables and their values will be known at runtime.
2. MPI_Recv(...,...,...,MPI_ANY,...,...)/ MPI_Recv(...,...,..,..,MPI_ANY_TAG,..):
   where source of message and tag are indicated by "wildcard" variable 'MPI_ANY' and 'MPI_ANY_TAG' respectively whose value will be known at runtime.
3. Related messages:
   If the process sends a message to any process immediately after receiving a message from another process, then the two messages are related with happened before relation and its communication pattern will be decided at runtime.

3.2. Dynamic Analysis of Programs. In dynamic analysis technique, to find communication pattern of a program, instrumentation instructions are added to the program. The modified program is executed and execution traces are recorded. After analyzing execution traces, a communication pattern of a program is found. As communication patterns are found by executing a program, the discovered communication pattern is more accurate. The time efficiency of this technique is less as compared to static analysis technique.

3.3. Symbolic Analysis of Programs. In this technique instead of actual input, the symbolic constant inputs are given to the program. By using symbolic input all possible conditions in the control flow instructions are checked to verify the execution flow of the program. Because of non-determinism in the verification process, more time is required as compared to static analysis. It fails to discover correct communication pattern if variables are used in communication function.

3.4. Summary of existing techniques. To schedule processes using topology-aware process placement, a technique for finding correct communication pattern of a program in less time is required. Existing techniques are either taking more time for program analysis or fail to find correct communication pattern. To detect correct communication pattern in less time, we have proposed a DAPE technique for analysis of MPI program. The details of partial execution and its algorithm are given in section 4 and 5 respectively. Advantages and disadvantages of the existing technique are summarized in Table 3.1.

4. Analysis of MPI program.

4.1. Partial execution of MPI program. To understand the concept of partial execution, consider following MPI program (Listing 1) which contains four types of instructions.

a) Control flow [Line Number 9 and 15]
b) Computation [Line Number 8, 11, 13, 17, 19,21]
c) Communication [Line Number 12 and 18]
d) Declaration.

Table 3.1
Comparison of MPI program analysis techniques

Sr. No.	Technique	Advantages	Disadvantages
1	Static analysis	1.It is scalable as actual program execution is not required. 2.It takes less time as compared to other techniques.	1.Automation of this technique is difficult as program annotation by the user is required. 2.It cannot handle “wildcard” variables in MPI_Recv() function.
2	Dynamic analysis	1.A communication race conditions which are arising from “wildcard” variable in MPI_Recv() function are handled. 2.As it executes the entire program, it explores all possible behavior of a program.	1.It takes more time for analysis as the complete execution of a program is required. 2.It identifies communication patterns specific to a particular input.
3	Symbolic analysis	1.It verify properties of all possible behavior of MPI program 2.As input is symbolic value, it explores program behavior independent of input value	1.For its implementation sophisticated theorem proving technique and symbolic interpretation is required. 2.It cannot scale beyond a relatively a small number of processes.

Listing 1
The sample MPI program

```
int main(int argc, char * argv[])
{
 int rank, numprocs;
 char buffer[20];
 MPI_Init(&argc, &argv);
 MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  ...................computation
 if(rank%2==0)
 {
     ...................computation
    MPI_Send(&buffer,10,MPI_CHAR,rank+1,tag,MPI_COMM_WORLD);
     ...................computation
 }
 else
 {
     ...................computation
    MPI_Recv(&buffer,10,MPI_CHAR,rank-1,tag,MPI_COMM_WORLD);
     ...................computation
  }
   ...................computation
}
```

While finding communication pattern of a program, the focus should be given to sender, receiver, and size of the message, not on the actual result of program execution. It is sufficient to record the sender, the receiver and the size of the message without executing communication or computing instructions. This can be done by using instrumentation techniques. We have developed the DAPE technique to find communication pattern of

MPI programs and its details are given in section 5. The preprocessed MPI program is given as input to this algorithm to perform instrumentation. In preprocessing the instructions from the input program are organized in such a way that each line will contain single instruction. In the second step, necessary variables and instructions containing those variables are identified. The necessary variables are those variables from MPI communication function which indicates either the sender or the receiver or the size of a message. In the third step, instrumentation instructions are added to generate a trace for communication pattern of MPI programs. During the first run of a program, only instructions containing necessary variables and instrumentation instructions are executed. The instrumented program for the program in listing 1 is shown in listing 2.

Listing 2
The instrumented program for sample MPI program

```
int main(int argc, char * argv[])
{
   int rank, numprocs;
   char buffer[20];
   MPI_Init(&argc, &argv);
   MPI_Comm_size(MPI_COMM_WOLRD, &numprocs);
   MPI_Comm_rank(MPI_COMM_WOLRD, &rank);
   // ..................computation
   if(rank%2==0)
    {
      //..................computation
      MPI_Send(&buffer,10,MPI_CHAR,rank+1,tag,MPI_COMM_WORLD);
      fprintf(logfile,"Sender:%d␣Reciever:%d␣Size:%d","src,dest,size");
      //..................computation
    }
   else
    {
      //..................computation
      MPI_Recv(&buffer,10,MPI_CHAR,rank-1,tag,MPI_COMM_WORLD);
      fprintf(logfile,"Sender:%d␣Reciever:%d␣Size:%d","src,dest,size");
       //..................computation
     }
   //..................computation
}
```

The instrumented program is compiled by using MPI compiler and executed as usual with a required number of processes. Each process will write communication logs in 'logfile'. By merging these log files communication patterns for the entire program is accumulated and communication matrix for a program is generated. That matrix gives topology of MPI program which will be further used for topology-aware process placement.

4.2. Assumptions for partial execution.

Assumption 1: The time taken for execution of MPI program involves computation time and communication time:

$$\text{Execution time of program} = \text{computation time} + \text{communication time} \tag{4.1}$$

In dynamic analysis technique, during the first run of a program both communication and computing instructions are executed. Therefore for execution of large programs, more time will be taken. In DAPE technique, instructions containing necessary variables (variables from MPI communication functions) and instrumentation instructions are executed instead of executing all instruction in the program. This will minimize the time required for program execution. In this way, a program is partially executed instead of complete execution.

Assumption 2: The communication pattern of application depends on two conditions:

F1: Control flow of program
  If sending or receiving of a message depends on control instruction in the program, then 'F1' condition will be satisfied.

F2: Data flow in the program
  If sending or receiving of a message depends on receiving a message from any process then 'F2' condition will be satisfied.

If the communication depends on F1 then for each and every execution same communication pattern will be generated. But if it depends on F1 and F2 then communication pattern depends on input data and will be different for each execution.

5. Dynamic Analysis with Partial Execution. The 'INSTRUMENTOR' procedure from Algorithm 1 finds message blocks which are executed by different processes of the program. From message blocks, the necessary variables are identified and added to list of the array as shown in 'FINDNECESSARYVARIABLE' procedure. During instrumentation of an MPI program for each line following actions will be taken based on the type of instruction:

I. If the line contains necessary variable then do not put the comment on that line [Line 17].
II. If the line contains control instructions then do not put the comment on that line [Line 18].
III. If the line contains communication instructions then put the comment on that line and also write instrumentation instruction [Line 19].
IV. If the line does not contain necessary variable and it is declaration type then put a comment on that line [Line 20].
V. For any other type of instructions put a comment.

5.1. Parameters for comparing proposed technique. The DAPE technique has the following merits over the existing techniques:

1. Time Efficiency:
   In dynamic and symbolic analysis complete execution of a program is needed therefore the time required for analysis is more. In case of DAPE technique very less time is required as the program is partially executed.
2. The accuracy of communication pattern:
   Even though DAPE executes program partially, it executes all necessary instructions on which communication pattern depends. Therefore it finds correct communication pattern like dynamic and symbolic analysis does.
3. Scalability:
   In dynamic analysis as the number of processes in a program increases the time required for analysis also increases. The DAPE technique takes the same amount of time for any number of processes as it executed only required instructions. Therefore it is scalable with respect to a number of processes.

6. Experimental results. We have conducted experiments to check the accuracy, performance, and scalability of the proposed technique. All the experiments are conducted on WCE-Rock cluster [22]. This cluster contains three nodes connected using InfiniBand network and total 56 cores. Each core has the processing power of 2.25 GHz. The total main memory in the cluster is 29 GB and physical storage of 5 TB. For testing purpose, ten MPI programs having different communication patterns are used. As shown in Figures 6.1(a) and 6.1(b), the program execution traces of DAPE are simple as that of ISP tool. Therefore time required to find communication pattern is less in case of DAPE technique.

6.1. The accuracy of DAPE. The communication patterns for ten MPI programs are found by using both ISP tool and DAPE technique. From each pattern number of point to point and collective calls in the program are found. As shown in Table 6.1, the DAPE technique finds a same number of point to point and collective communication calls as that of ISP tool.

By analyzing the execution traces (Figures 6.1(a) and 6.1(b)) generated by ISP tool and DAPE technique, the communication pattern of MPI program is stored in matrix format as shown in Tables 6.2 and 6.3. To check the accuracy of DAPE technique, communication matrix generated by it is compared with communication

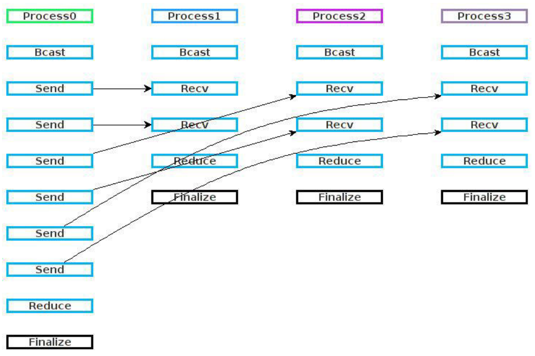
Algorithm 1 DAPE algorithm

```
 1: Input : P                                                    ▷ The MPI program
 2: Output : P'                                         ▷ Instrumented MPI program
 3: procedure PREPROCESSING(P)
 4:     while (!EOF(P)) do
 5:         L = Read line from P
 6:         if ('L' Contains single instruction ) then               ▷ Skip that line
 7:         else
 8:             Arrange instructions such that each line contains single instruction
 9:         end if
10:     end while
11: end procedure
12: ------------------------------------------------------------------------------
13: procedure INSTRUMENTOR(P)
14:     NV=FindNecessaryVariables(P)
15:     while (!EOF(P)) do
16:         L = Read line from P
17:         if ('L' Contains Necessary Variable from NV) then   ▷ Don't put comment on that line
18:         else if ('L' Contains Control Instruction) then     ▷ Don't put comment on that line
19:         else if ('L' Contains Communication Instruction) then ▷ Perform instrumentation and put a
    comment on that line
20:         else if ('L' Contains Declaration Instruction) then       ▷ Put comment on that line
21:         end if
22:     end while
23:     return P'
24: end procedure
25: ------------------------------------------------------------------------------
26: Input : P                                                    ▷ The MPI program
27: Output : NV                          ▷ List of necessary variables from program p
28: procedure FINDNECESSARYVARIABLES(P)
29:     Find message block
30:     Find communication function
31:     Get 4th parameter from the function. If it is variable add to the list of necessary variable(NV)
32:     Get 1st and 2nd parameter and compute the size of the message from both parameters
33:     Get rank variable from control flow instruction & determine the source of message
34:     return NV
35: end procedure
```

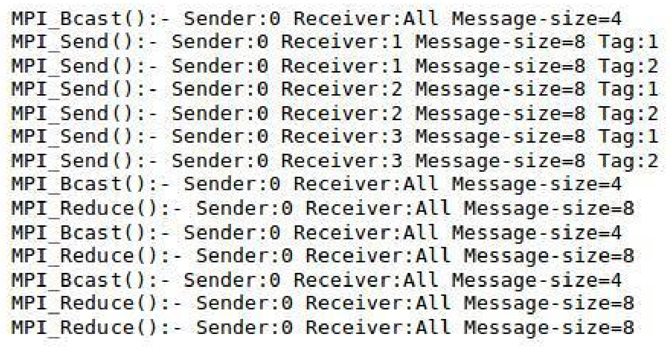
matrix generated by ISP tool. The DAPE technique has the same accuracy like ISP tool as it generates same communication matrix as ISP tool. Thus the accuracy of DAPE technique is same as that of ISP tool.

6.2. The efficiency of DAPE. The time required for MPI program analysis is used to measure the efficiency of the technique. As shown in Figure 6.2, DAPE takes less time for analysis as compared to ISP tool.

6.3. Scalability of DAPE. To find communication pattern of MPI program for the different number of processes, the program is executed with a different number of processes. In case of ISP tool, as the number of processes in the program increases the analysis time also increases. But in case of DAPE tool, there is less variation in analysis time even if a number of processes increases. As shown in figure 6.3, DAPE is scalable with respect to a number of processes in the program.



(a) Analysis of MPI program using ISP



```
MPI_Bcast():- Sender:0 Receiver:All Message-size=4
MPI_Send():- Sender:0 Receiver:1 Message-size=8 Tag:1
MPI_Send():- Sender:0 Receiver:1 Message-size=8 Tag:2
MPI_Send():- Sender:0 Receiver:2 Message-size=8 Tag:1
MPI_Send():- Sender:0 Receiver:2 Message-size=8 Tag:2
MPI_Send():- Sender:0 Receiver:3 Message-size=8 Tag:1
MPI_Send():- Sender:0 Receiver:3 Message-size=8 Tag:2
MPI_Bcast():- Sender:0 Receiver:All Message-size=4
MPI_Reduce():- Sender:0 Receiver:All Message-size=8
MPI_Bcast():- Sender:0 Receiver:All Message-size=4
MPI_Reduce():- Sender:0 Receiver:All Message-size=8
MPI_Bcast():- Sender:0 Receiver:All Message-size=4
MPI_Reduce():- Sender:0 Receiver:All Message-size=8
MPI_Reduce():- Sender:0 Receiver:All Message-size=8
```

(b) Analysis of MPI program using DAPE

FIG. 6.1. *Analysis of MPI program*

TABLE 6.1
Comparison of DAPE technique with ISP tool [No. of processes=80]

MPI Program	ISP Tool		DAPE Technique	
	#Calls (Point to Point) + (Collective)	Analysis Time	#Calls (Point to Point) + (Collective)	Analysis Time
Array.c	28+1	4.087654	28+1	2.029162
Blocking-Comm.c	1 (deadlock)	2.062398	8+0	0.0068
Hellosend.c	1+0	2.115488	1+0	0.00003
Matrix-mul.c	49+0	4.474906	49+0	0.001052
Quad.c	14+2	3.34783	14+2	0.000156
Ring.c	48+0	9.821415	48+0	0.000088
Gauss-Elimination.c	0+8	4.868085	0+8	3.901431
Integration.c	7+0	2.070263	7+0	0.198444
Safty.c	0+1	2.376012	0+1	0.246936
Prime.c	0+38	11.419859	0+38	9.366117

7. Conclusion.

The communication latency is the performance bottleneck for MPI programs on multi and many core systems. By using topology-aware process placement, this latency can be minimized. In literature, various MPI program analysis techniques are proposed. Each technique has its own pros and cons. In this paper, we have proposed a new technique for analysis of MPI program. The DAPE technique finds correct communication pattern in a program except for few programs where conditions F1 and F2 both are satisfied i.e. where program communication pattern depends on both data flow in the program and control flow of the program. The time efficiency of DAPE technique is more as compared to ISP tool. Also, it is scalable with a number of processes in the program as compared to ISP tool.

In future, we are planning to extend the functionality of DAPE which will correctly detect communication pattern of a program even if both F1 and F2 conditions are satisfied. Also, we are planning to integrate our technique with Open MPI so that it can be used while compiling the program.

REFERENCES

[1] MESSAGE PASSING INTERFACE FORUM, *MPI: A message-passing interface standard*, MPI forum., (2009).
[2] GUILLAUME MERCIER AND JEROME CLET-ORTEGA, *Towards an Efficient Process Placement Policy for MPI Applications in Multi-core Environments*, Recent Advances in Parallel Virtual Machine and Message Passing Interface, Volume 5759 (2009), pp. 104-115.

TABLE 6.2
Communication matrix generated by DAPE for 'Ring.c' program

	P0	P1	P2	P3	P4	P5
P0	0	50	0	0	0	0
P1	0	0	50	0	0	0
P2	0	0	0	50	0	0
P3	0	0	0	0	50	0
P4	0	0	0	0	0	50
P5	50	0	0	0	0	0

TABLE 6.3
Communication matrix generated by DAPE for 'Matrix_mul.c' program

	P0	P1	P2	P3	P4	P5
P0	0	4	4	4	4	4
P1	3	0	0	0	0	0
P2	3	0	0	0	0	0
P3	3	0	0	0	0	0
P4	3	0	0	0	0	0
P5	3	0	0	0	0	0

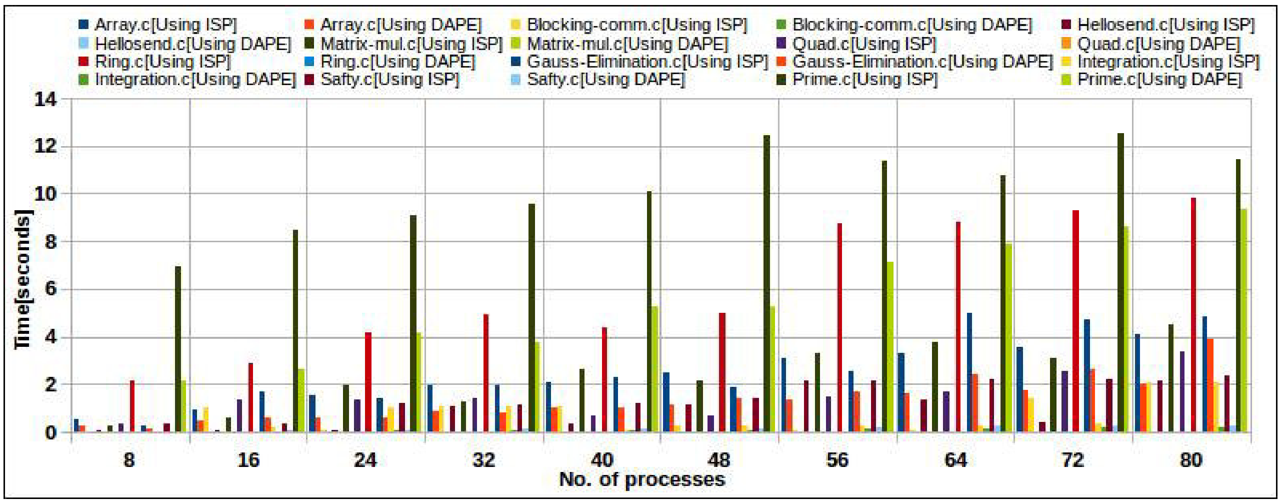


FIG. 6.2. *Time required for analysis: ISP vs DAPE*

[3] S. F. SIEGEL AND G. GOPALAKRISHNAN, *Formal Analysis of Message Passing*, Lecture Notes in Computer Science, volume 6538 (2011), pp.2-18.

[4] G. GOPALAKRISHNAN, ROBERT M. KIRBY, STEPHEN SIEGEL, RAJEEV THAKUR, WILLIAM GROPP, EWING LUSK, BRONIS R. DE SUPINSKI, MARTIN SCHULZ, GREG BRONEVETSKY, *Formal Analysis of MPI-Based Parallel Programs: Present and Future*, ACM transaction on Communication, (2011), http://www.computer.org/portal/web/csdl/doi/10.1109/SC.2010.

[5] SHUYI SHAO, YU ZHANG, ALEX K. JONES, RAMI MELHEM, *Symbolic Expression Analysis for Compiled Communication*, Proceedings on IEEE International Parallel and Distributed Processing Symposium, ISBN: 978-1-4244-1693-6 (2008), pp.1-8.

[6] DALE R. SHIRES AND LORI POLLOCK, *Program Flow Graph Construction for Static Analysis of Explicitly Parallel Message-Passing Programs*, University Of Wisconsin-Madison, (2000).

[7] MICHELLE MILLS STROUT, BARBARA KREASECK, PAUL D. HOVLAND, *Data-Flow Analysis for MPI Programs*, Proceedings on International Conference on Parallel Processing, ISBN: 0-7695-2636-5 (2006), pp.175-184.

[8] ANDREW J. MCPHERSON, VIJAY NAGARAJAN, AND MARCELO CINTRA, *Static Approximation of MPI Communication Graphs for Optimized Process Placement*, The 27th International Workshop on Languages and Compilers for Parallel Computing, (2014).

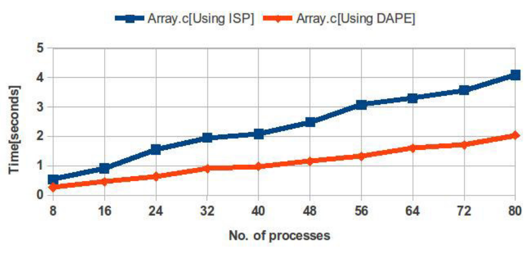
[9] JULIEN JAEGER, EMMANUELLE SAILLARD, PATRICK CARRIBAULT, DENIS BARTHOU, *Correctness analysis of MPI-3 non blocking communication in PARCOACH*, European MPI Users Group Meeting, Bordeaux, France, (2016).

[10] SRIRAM ANANTHAKRISHANAN, GREG BRONEVETSKY, MARK BARANOWSKI, *ParFuse: Parallel and compositional analysis of message passing programs*, Lecture notes in computer science, Springer (2017).

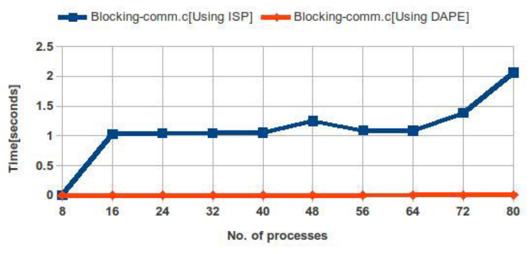
[11] SARVANI S. VAKKALANKA, SUBODH SHARMA, GANESH GOPALAKRISHNAN, ROBERT M. KIRBY, *ISP: a tool for model checking MPI programs*, Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming, ISBN: 978-1-59593-795-7 (1971),pp.285-286.

[12] ANH VO, SARVANI VAKKALANKA & RAJEEV THAKUR, *Formal Verification of Practical MPI Programs*, Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming, ISBN: 978-1-60558-397-6 (2009), pp.261-270.

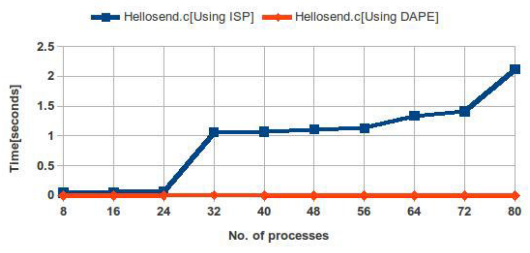
[13] HU CHEN, WENGUANG CHEN, JIAN HUANG, BOB ROBERT, H. KUHN, *MPIPP: an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters*, Proceedings of the 20th annual international conference on Super-



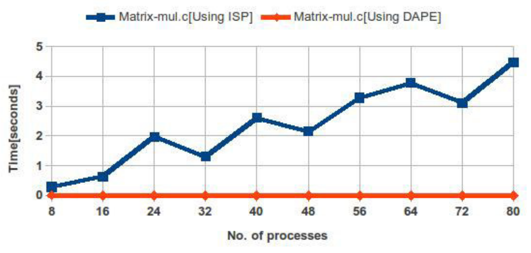
(a) Array.c



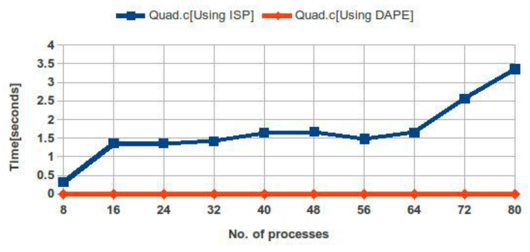
(b) Blocking-comm.c



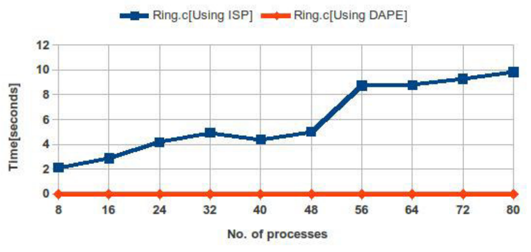
(c) Hellosend.c



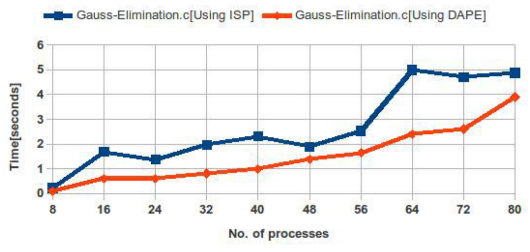
(d) Matrix-mul.c



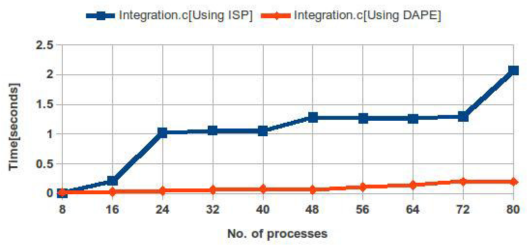
(e) Quad.c



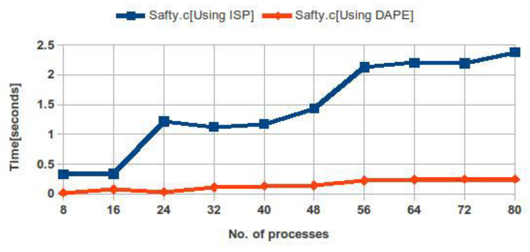
(f) Ring.c



(g) Gauss-Elimination.c



(h) Integration.c



(i) Safty.c

FIG. 6.3. *Scalability of ISP vs DAPE*

computing, ISBN:1-59593-282-8 (2006),pp.353-360.

[14] Jidong Zhai, Tianwei Sheng, Jiangzhou He, Wenguang Chen, Weimin Zheng, *FACT: fast communication trace collection for parallel applications through program slicing*, Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, ISBN: 978-1-60558-744-8 (2009), Article-27.

[15] Yan Li, Jidong Zhai, Keain Li, *Communication analysis and performance prediction of parallel application on large scale machines*, Advances in system analysis, software engineering and high performance computing book series (2016).

[16] Vojtach Iorejt, Saurabh Joshi, Daniel Kroening, Ganesh Narayanaswamy, Subhodh Sharma, *Precise predictive analysis for discovering communication deadlocks in MPI programs*, ACM transaction on programming, languages and systems, Volume 39, issue 4, (2017).

[17] Xianjin Fu, Zhenbang Chen, Yufeng Zhang, Chun Huang, Wei Dong and Ji Wang, *MPISE: Symbolic Execution of MPI Programs*, Proceedings of IEEE 16th International Symposium on High Assurance Systems Engineering, ISBN: 978-1-4799-8111-3 (2015),pp.181-188.

[18] Sultan Aljahdali, Mosaid Al Sadhan, Alaa Ismail Elnashar, *Two automated techniques for analyzing and debugging mpi-based programs*, 24th International Conference on Computers and Their Applications in Industry and Engineering, ISBN: 9781618393227 (2011),pp.344.

[19] Shirley Moore, David Cronk, Kevin London, and Jack Dongarra, *Review of Performance Analysis Tools for MPI Parallel Programs*, Proceedings of the 8th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, ISBN:3-540-42609-4 (2001), pp.241-248.

[20] Sriram Aananthakrishnan, Ganesh Gopalakrishnan, Bronis R. de Supinski, Martin Schulz, Greg Bronevetsky, *A Scalable and Distributed Dynamic Formal Verifier for MPI Programs*, Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking,Storage and Analysis, ISBN: 978-1-4244-7559-9 (2010), pp.1-10.

[21] Stephen F. Siegel, Yi Wei, Timothy K. Zirkel, *TASS: The Toolkit for Accurate Scientific Software*, Proc. Mathematics in Computer Science, (2011).

[22] http://wce.ac.in/it/landing-page.php?id=9.

Edited by: Dana Petcu
Received: May 19, 2017
Accepted: Nov 13, 2017