DETERMINING INTUITIONISTIC FUZZY DEGREE OF OVERLAPPING OF COMPUTATION AND COMMUNICATION IN PARALLEL APPLICATIONS USING GENERALIZED NETS

Pavel Tchesmedjiev, Peter Vassilev
Centre for Biomedical Engineering, Bulgarian Academy of Sciences
Acad. G. Bonchev Str., Bl. 105, Sofia-1113, Bulgaria
E-mails: *pavel@clbme.bas.bg* , *peter.vassilev@gmail.com*

Abstract
Overlapping of computation and communication can reduce overall application execution time and all parallel applications could benefit to a certain degree. In order to achieve this we present a new approach for determining the degree of overlapping.
Keywords: Communication, Computation, Generalized net, Intuitionistic fuzzy sets, Modelling, MPI, Overlapping, Parallel process

1 INTRODUCTION

Overlapping is a high performance software mechanism that enables concurrent execution of independent communication and computation activities. The major objective of overlapping is to reduce the overall application time by utilizing hardware and software architectures that offer concurrent progress of communication and computation. Overlapping is one of the fundamental algorithmic approaches for improving parallel performance. All parallel applications could benefit from overlapping to a certain degree.

This paper focuses on a new approach for determining degree of overlapping. This approach includes evaluating the intuitionistic fuzzy value of the degree, using a generalized net model (GN, see [1]).

2 COMMON GENERALIZED NET MODEL FOR MPI PARALLEL FUNCTIONS

In order to determine the value of overlapping we will introduce a common generalized net for modeling and simulating parallel applications.

The following MPI functions (definitions are given in the C programming language; see [3]) are commonly used in parallel programs:

- *int MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root, MPI_Comm comm)* – the value of the data, pointed by **buf*, of type *datatype* and containing *count* elements, is sent to all other processes(associated to the communicator comm). So afer calling the function every process ends with a copy of the data.
- *int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)* – the value of the data, pointed by **sendbuf* , of type *datatype* and containing *count* elements from each process are sent to the process with rank *root* and placed in the **recvbuf*, as operation *op* is committed before that.
- *int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)* – the value of the data, pointed by **buf* , of type *datatype* and containing *count* elements is sent to the destination process from the communicator *comm* - *dest*, using additional information *tag*.
- *int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)* – the master process requests for information from another process *source* from the communicator *comm* and blocks until the an answer is returned. The information is represented by the value, pointed by **buf* , of type *datatype* and containing *count* elements with additional information *tag*. Status is returned by the variable **status*.

The described parallel routines can be represented in the common GN model for n parallel processes and one master process that activates them – see Fig. 1.

For the master process, represented by transition Z_0 :

$$Z_0 = <\{l_0\}, \{l_1,\ l_3\ , \ldots\ l_{2i+1}, \ldots\ l_{2n+1}\}, r_0>$$

$$r_0 = \begin{array}{c|cccccc} & l_1 & l_3 & \cdots & l_{2i+1} & \cdots & l_{2n+1} \\ \hline l_0 & W_{0,1} & W_{0,3} & \cdots & W_{0,2i+1} & \cdots & W_{0,2n+1} \end{array}$$

For one of the parallel processes, represented by transition Z_{i+1}, $i = \{0,n\}$:

$$Z_{i+1} = <\{l_{2i+1}\}, \{l_{2i+2}\}, r_{i+1}>$$

$$r_{i+1} = \begin{array}{c|c} & l_{2i+2} \\ \hline l_{2i+1} & true \end{array}$$

The transitions represent parallel processes. When the master process sends a message to other parallel processes a token is created and the

respective transition is activated. The tokens have the following properties – data, count and datatype. The conditions $W_{0,1}$, $W_{0,3}$, $W_{0,2i+1}$, $W_{0,2n+1}$ for the case of *MPI_Bcast, MPI_Reduce* and *MPI_Send* functions are always TRUE.

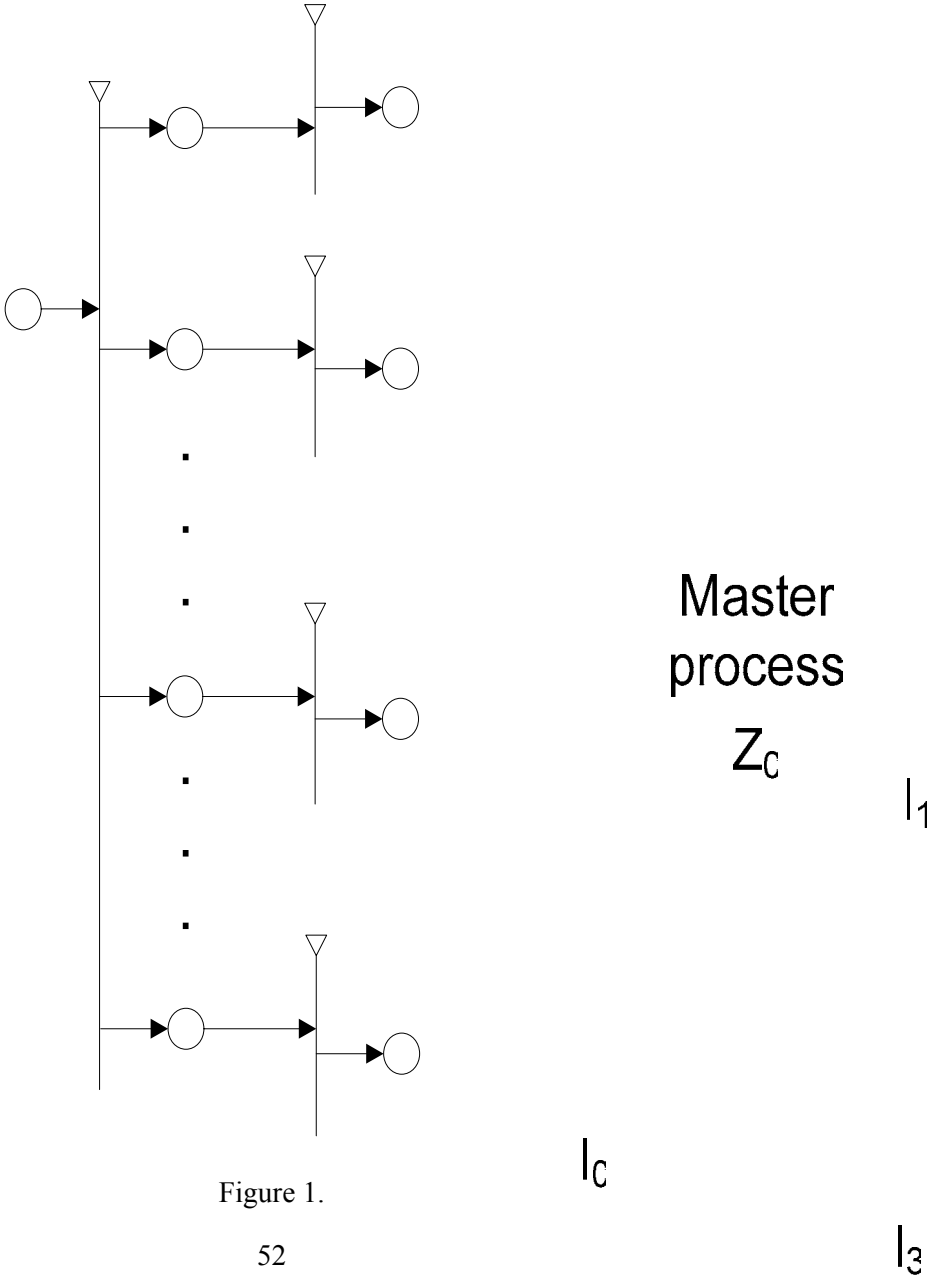

Figure 1.

When the used function is *MPI_Recv*, $W_{0,1}$, $W_{0,3}$, $W_{0,2i+1}$, $W_{0,2n+1}$ are initially FALSE and become TRUE when the master process is ready with the requested token (data).

3 DEFINITION OF OVERLAPPING

If the total execution time (see [2]) of a parallel application is T_p the computation time is T_{comp} and the communication time is $T_{c,}$ the objective of overlapping can be expressed as follows:

$$T_p = T_{comp} + T_c - T_{o,}$$

Where T_o is the time saving achieved through overlapping. Traditional approaches for parallel performance improvement emphasize minimization of T_{comp} and T_c and typically rely on increasing raw processor speed and network speeds. Clearly, overlapping presents alternative approach for improving performance. Overlapping depends primarily on efficient software and hardware architectures rather than on top speed components.

Overlapping utilizes software techniques such as asynchronous message-completion notification, low processor overhead, independent message progress. Major hardware features of the parallel system needed for effective overlapping are availability of excess bandwidth of the memory subsystem and intelligent network interface controllers capable of accessing host memory. The introduced common GN model provides the needed abstraction of all hardware and software layers of a real parallel system.

The relation between the communication and computation activities plays an important role in achieving effective overlapping. To illustrate this importance, a computation activity X and a communication activity Y are reviewed. The durations of these activities are, respectively, t_x and t_y. For simplicity we can assume that there are no preparation times and the entire communication and computation times can be represented as s sum of N individual activities, such as T_{comp} (*estimated*) = $\sum t_x$ and T_c (*estimated*) = $\sum t_y$ or if we want to be more precise

$$T_{comp}\,(actual) = (1 + \pi(T_{comp})).T_{comp}\,(estimated)$$

and

$$T_c\,(actual) = (1 + \pi(T_c)).T_c\,(estimated)_,$$

where $\pi(T_{comp})$, $\pi(T_c)$ are numbers between 0 and 1 that reflect the delays of the tasks (in the ideal case the value would be 0). We also assume that X and Y should be algorithmically independent – they can be executed concurrently without violating the correct semantics of the algorithm.

If the reviewed computation and communication activities are not overlapped, the serial time for their completion is $t_s = t_x + t_y$. If the two activities are ideally overlapped, the total time t_o will be the larger of t_x and t_y and $t_o = \max(t_x, t_y)$. The overall time reduction is

$$S = (t_x + t_y) / \max(t_x, t_y)$$

In order to quantify the speedup because of overlapping, the relationship between X and Y is represented as $t_y = q.t_x$, where $q > 0$ is a factor that represents how much Y is longer or shorter than X. Then speedup is as follows:

$$S = (t_x + q.t_x) / \max(t_x, q.t_x) = (1 + q)t_x / \max(t_x, q.t_x)$$

In the first case let $q > 1$, that is, communication is longer than computation, then

$$S = (1 + q).t_x / (q.t_x) = (1 + q) / q.$$

This expression shows that if q approaches 1, then S approaches 2, and when q approaches positive infinity, S approaches 1. In the second case when $0 < q \leq 1$, meaning that the communication time is shorter than the computation time, then $S = 1 + q$. So when q approaches 0, S approaches 1 while for q approaches 1, S approaches 2. So $1 \leq S \leq 2$, the maximum achieved when $t_x = t_y$.

4 DETERMINING INTUITIONISTIC FUZZY DEGREE OF OVERLAPPING

We will introduce a new metric, used for accurate modelling of overlapping – degree of overlapping, denoted by $\mu(d_0)$ and degree of indeterminacy (uncertainty) $\pi(d_0)$. As in all Intuitionistic Fuzzy Set (IFS, see [4]) we will impose the restrictions $\mu+\pi \leq 1$. The degree of overlapping is used as a quantitive description of the capabilities of a system in order to perform effective overlapping of communication and computation. This quantification is necessary for devising a realistic model of parallel execution time. Overlapping is now expressed in the form:

$$T_p = T_{comp} + T_{pc} - ((1 - \pi(T_o)).T_o(\mu(d_0))), \quad \text{for } 0 \leq \mu(d_0) \leq 1,$$

where measures $\pi(T_o)$ the uncertainty of the time of the executed algorithm.

Degree $\mu(d_0)$ is chosen so that the maximum impact of overlapping on the overall performance is achieved as the value of $\mu(d_0)$ approaches 1. As we assumed earlier, $\pi(d_0)$ will approach 0 and that's why $0 \leq \mu(d_0) \leq 1$. There are several reasons for introducing degree-of-overlapping metric. First it can be

used to distinguish between hardware platforms and software systems that have specific optimizations for achieving effective overlapping and those that do not provide such optimizations. Second, this new metric helps to provide more realistic model of overlapping that accurately quantifies the performance benefits to applications. And third it can be used for a guideline for improving parallel platform architectures and application algorithms implementation. Using the proposed common GN model and IF values will help even more for achieving this tasks.

Degree $\mu(d_0) = 1$ means that the system can achieve ideal overlapping and $\mu(d_0) = 0$ means that no overlapping is possible. Ideal overlapping is impossible because the communication activities require certain processor overhead and the message transfers compete to some degree with the central processor to access to the main memory. As a result of CPU analysis, the communication activity Y, introduced earlier, can be presented as a sum of two sub-activities Y_h and Y_c , where Y_h denotes all activity that cause processor overhead and resource contention, and Y_c is the component that can be overlapped with computation. During Y_h no overlapping is possible, because CPU is busy with communication-related work. The time of activity Y can be represented as sum of two components $t_y = t_{yh} + t_{yc}$ then the serial time of X and Y is:

$$t_s = t_x + t_{yh} + t_{yc}$$

and the overlapped time is:

$$t_o = t_{xh} + \max(t_{yc}, t_x)$$

Clearly the longer the overhead time t_{xh} is, the smaller the effect of overlapping will be. So we can formally define the degree of overlapping as:

$$\mu(d_0) = t_y / t_{xy} = t_{yc} / t_{yh} + t_{yc}$$

and finally $t_o = t_x + (1 - \mu(d_0)).t_y$, when $t_x > t_{xc}$ and $t_o = t_y$, when $t_x \le t_{yc}$

We will consider the case when when $t_x > t_y$ since this condition can be easily identified by a single experiment. In summary the values of the degree of overlapping are in the range $0 \le \mu(d_0) \le 1$.

Degree $\mu(d_0)$ can be determined using the GN model's execution times as follows:

$$\mu(d_0) = 1 + (t_x - t_o) / t_y = (t_s - t_o) / t_y$$

Common generalized net for executing parallel application and determining the degree of overlapping is shown on Fig. 2.

The transitions from Z_1 to Z_{n+1} represent the n parallel processes that are executing computation and communication activities. Each token keeps the needed time values. The degree of overlapping $\mu(d_0)$ is calculated by the token in place l_{3n+4}.

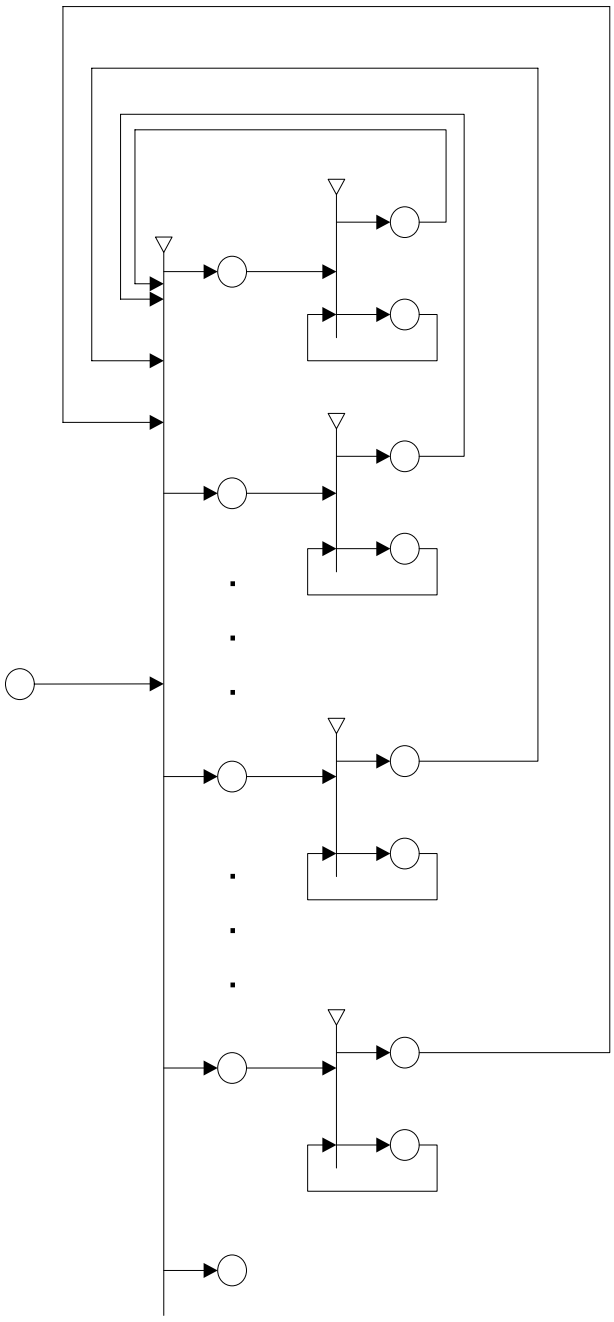

Figure 2.

5 CONCLUSION

By determining the IF value of the degree of overlapping capabilities of a system we can perform effective overlapping of communication and computation. By using common generalized net model we can evaluate this value and use it for further enhancements of the parallel system and software.

ACKNOWLEDGEMENTS

This work was supported by Grant BIn-2/09 “Design and development of intuitionistic fuzzy logic tools in information technologies” of the National Science Fund of Bulgaria.

REFERENCES

[1] Atanassov, K., Generalized Nets, World Scientific, Singapore, 1991

[2] Dimitrov, Rossen, Overlapping of communication and computation and early binding, University of Mississipi, 2001

[3] Gropp, William, Lusk, Ewing, Skjellum, Anthony, Using MPI – Portable Parallel Programming with the Message-Passing Interface, The Mit Press, 1997

[4] Atanassov, K, Intuitionsitic Fuzzy Sets. Springer Physica-Verlag, Heidelberg, 1999.