Ninth Int. Workshop on GNs, Sofia, 4 July 2008, 84-88

Modelling self-scheduling matrix-vector multiplication using Generalized Nets

Pavel Tcheshmedjiev

Centre for Biomedical Engineering „Prof. Ivan Daskalov“
Bulgarian Academy of Sciences, Academy of Sciences
Acad. G. Bonchev Str., Bl. 105, Sofia-1113, Bulgaria
pavel@clbme.bas.bg

Abstract. In the present paper Generalized Nets are proposed as a common ground for modelling parallel self-scheduling algorithm. We will demonstrate how a GN model can be applied in the context of matrix-vector multiplication and present the advantages of this approach

Keywords: Generalized Net, parallel processes, modelling, MPI, self-scheduling, matrix multiplication

1 Introduction

Fast computers have stimulated the rapid growth of new way of doing science. The two broad classical branches of theoretical science and experimental science have been joined by computational science. Computational scientists simulate on supercomputers phenomena too complex to be reliably predicted by theory and too dangerous or expensive to be reproduced in the laboratory. Success in computational science have caused demand for supercomputing resources to rise sharply.

This paper focuses on a new approach for designing parallel(MPI) programs by modelling parallel processes in the terms of generalized nets. Using generalized nets will help to imagine clearly and understand the parallelism from one side, and analyze and tune the efficiency of the distributed application from another. GN models are created faster, and can be easily maintained, changed or extended. When the GN model is ready the real computing can be executed in the form of the parallel program.

2 GN transitions for MPI parallel functions

The following mpi functions(definitions are given in the C programming language) 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 maser 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 one single 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 :

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

	l_1	l_3	l_{2i+1}	l_{2n+1}
l_0	$W_{0,1}$	$W_{0,3}$	$W_{0,2i+1}$	$W_{0,2n+1}$

For one of the parallel processes, represented by transition Z_{i+1} :

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

	l_1
l_0	TRUE

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.

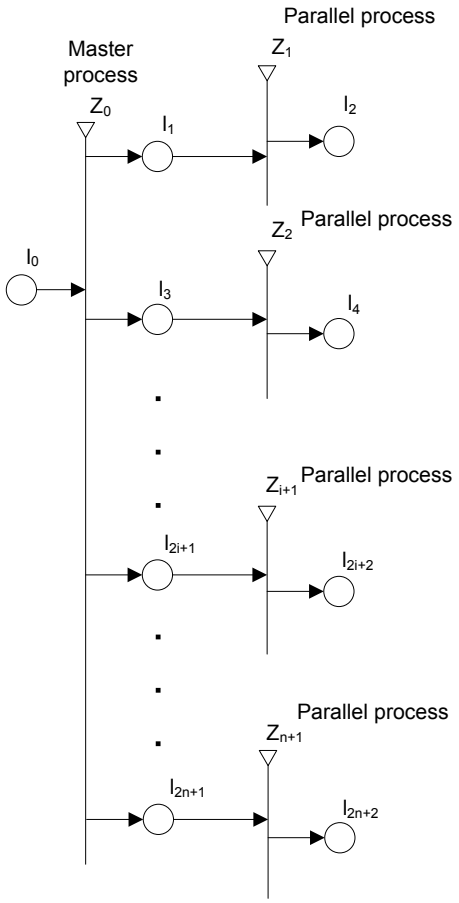

Fig. 1.

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.

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 GN model of self-scheduling matrix-vector multiplication

The idea of the self-scheduling parallel algorithm is that one process, which can be called the master process, is responsible for coordinating the work of the others. The mechanism is particularly appropriate when the other processes(the worker or slave processes) do not have to communicate with one another and when the amount of work that each slave must perform is difficult to predict. In the context of MPI programs it is implemented using the *send* and *receive* routines.

The unit of work to be given out will consist of the dot product of one row of the matrix **A** with the (column) vector **b**. The master begins by broadcasting **b** to each slave. It then sends one row of the matrix **A** to each slave. At this point the master begins a loop, terminated when it has received all of the entries in the product. The MPI program body of the loop consists of receiving one entry in the product vector from whichever slave sends one, then sending the next

task to that slave. In other words, completion of one task by a slave is considered to be a request for the next task. Once all tasks have been handed out, termination messages are sent instead.

Each slave, after receiving the broadcast value of **b**, also enters a loop, terminated by the receipt of the termination message from the master. The MPI program body of the loop consists of receiving a row of **A**, forming the dot product with b, and sending the answer back to the master.

The master uses the *send* operation to send a row of **A** to a slave. The responses from the slaves are received by the *receive* operation.

The described MPI parallel algorithm can be represented by a common GN model. The GN model can also be used for other parallel programs, as shown in the related paper – *"Modelling MPI parallel processes for cluster environment using Generalized Nets"*

The generalized net for parallel matrix-vector multiplication :

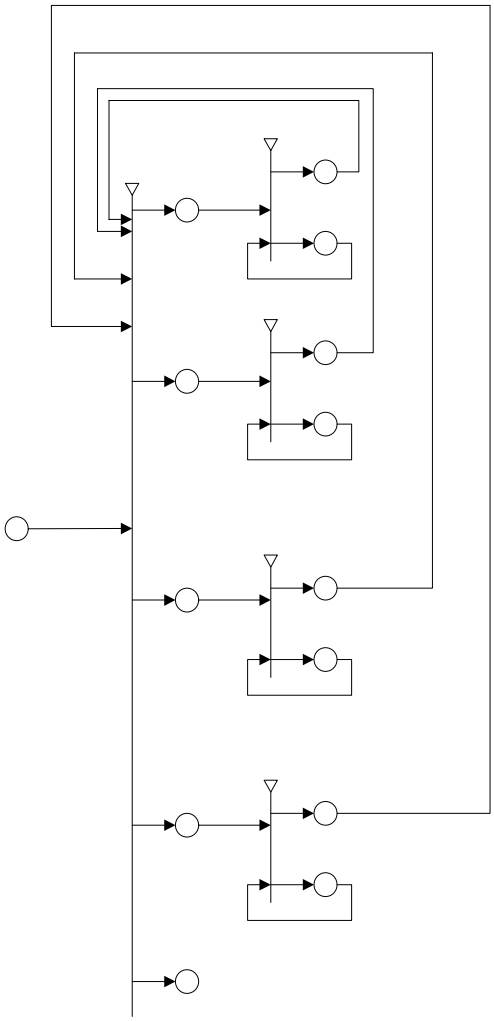

Fig. 2.

The transitions from Z_1 to Z_{n+1} represent the *n* slave processes that are calculating the vector product. The local vector products are stored as properties of the tokens, circulating through places l_{3i+3}. After calculating the respective vector product, it is sent to the master

process, represented by transition Z_0 through the tokens from places l_{3i+2}. The final matrix is accumulated by the token in place l_{3n+4}.

5 Conclusion

By modeling parallel matrix-vector multiplication we have seen one more time some of the main advantages of the GN model over the technical program approach - simplicity of representation, the flexibility of design, and most important – the usage of the parallel nature of the GN models. The GN-model not only provide appropriate representation of the relations between the individual processes but allows quick and easy modifications of the model if there is such requirement . It also describes any given parallel algorithm without the usual complexity of the real program, where the programmers should consider every computational and communicational detail of the custom implementation.

References

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

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