11

Optimization of Parallel FDTD Computations Based on Program Macro Data Flow Graph Transformations

Adam Smyk[1] and Marek Tudruj[2]

[1]*Polish-Japanese Institute of Information Technology*
[2]*Institute of Computer Science Polish Academy of Science*
Poland

1. Introduction

This chapter concerns numerical problems that are solved by parallel regular computations performed in rectangular meshes that span over irregular computational areas. Such parallel problems are more difficult to be optimized than problems concerning regular areas since the problem cannot be solved by a simple geometrical decomposition of the computational area. Usually, a kind of step-by-step algorithm has to be designed to balance parallel computations and communication in and between executive processors. The Finite Difference Time Domain (FDTD) simulation of electromagnetic wave propagation in irregular computational area, numerical linear algebra or VLSI layout design belong to this class of computational problems solved by unstructured computational algorithms (Lin, 1996) with irregular data patterns. Some heuristic methods are known that enable graphs partitioning necessary to solve such problems (NP-complete problem (Garey et al., 1976)), but generally two kinds of such methods are used: direct methods (Khan et al., 1995) and iterative methods (Khan et al., 1995; Kerighan & Lin, 1970; Kirkpatrick et al., 1983; Karypis & Kumar, 1995; Dutt & Deng, 1997). Direct methods are usually based on the min-cut optimization (Stone & Bokhari, 1978). The iterative methods are mainly based on extensions of the algorithms of Kernighan-Lin (Kerighan & Lin, 1970), next improved by Fidducia-Mattheyses methods (FM)(Fiduccia & Mattheyses, 1982). There are also many kinds of various program graph partitioning packages like JOSTLE (Walshaw et al., 1995), SCHOTCH (Scotch, 2010) and METIS (Metis, 2008) etc. All of them enable performing efficient graph partitioning but there are two unresolved problems that have been found out. In the case of very irregular graphs, partitioning algorithms used in these packages can produce a partition that can be divided into two or more graph parts placed in various disjointed locations of the computofational area. As it follows from observed practice, there are no prerequisites to create such disjoint partitions, because in almost all cases it increases a total communication volume during execution in distributed systems. The second disadvantage is that the partitioning methods mentioned above do not take into account any architectural requirements of a target computational system. It is very important especially in heterogeneous systems, where proper load balancing allows efficient exploiting all computational resources and simultaneously, it allows reducing the total time of computations.

In (Smyk & Tudruj, 2006) we have presented a comparison of two algorithms: redeployment algorithm and CDC (Connectivity-based Distributed Node Clustering) algorithm (Ramaswamy et al., 2005). The first one is an extension of the FM algorithm and it is divided into three main phases. In the first phase, a partitioning of the FDTD computational area is performed. It provides an initial macro data flow graph to be used in further optimizations. The number of created initial macro nodes is usually much larger than the number of processors in the parallel system. Therefore, usually a merging algorithm phase is next executed. Several merging criteria are used to balance processor computational loads and to minimize total inter-processor communication. The obtained macro data flow graphs are usually adjusted to current architectural requirements in the last algorithm phase. A simple architectural model can be used for this in a computational cells redeployment. The second algorithm is a modification of the CDC algorithm known in the literature (Ramaswamy et al., 2005). It is decentralized and is based on information exchange on the whole computational area executed between neighboring nodes. In this chapter we present a hierarchical approach for program macro data flow graph partitioning for the optimized parallel execution of the FDTD method. In the proposed algorithm, we try to exploit the advantages of two mentioned above algorithms. In general, the redeployment algorithm is used to reduce the execution time of the optimization process, while the main idea of the CDC algorithm enables obtaining an efficient partitioning.

The chapter is composed of five parts. In the first part, the main idea of the FDTD problem and its execution according to macro data flow paradigm is described. In the next three parts, the redeployment and the CDC algorithms are described. We present experimental results which compare both of these algorithms. We also present a special memory infrastructure (RB RDMA) used for efficient communication in distributed systems. In the last part of this chapter we present an implementation of the hierarchical algorithm of FDTD program graph partitioning.

2. FDTD implementation with the macro data flow paradigm

Finite Difference Time Domain (FDTD) method is used in simulation of high frequency electromagnetic wave propagation. In general, the simulated area (two or three-dimensional irregular shape) can contain different characteristic sub-areas like excitation points, dielectrics etc. (see Fig. 1). The whole simulation is divided into two phases. In the first phase, whole computational area must be transformed into a discrete mesh (a set of Yee cells).

Each discrete point, obtained in this process, contains alternately (for two dimensional problem) electric component Ez of electromagnetic field and one from two magnetic components Hx or Hy (Smyk & Tudruj, 2006). In the second phase of the FDTD method, we perform wave propagation simulation (see Fig. 2). In each step of simulation, the values of all electric vectors (Ez) or the magnetic components (Hx, Hy) are alternately computed. Electromagnetic wave propagation in an isotropic environment is described by time-dependent Maxwell equations (1):

$$\nabla \times H = \gamma E + \varepsilon \frac{\partial E}{\partial t}, \qquad \nabla \times E = -\mu \frac{\partial H}{\partial t} \tag{1}$$

and can be easily transformed into their differential forms (2)

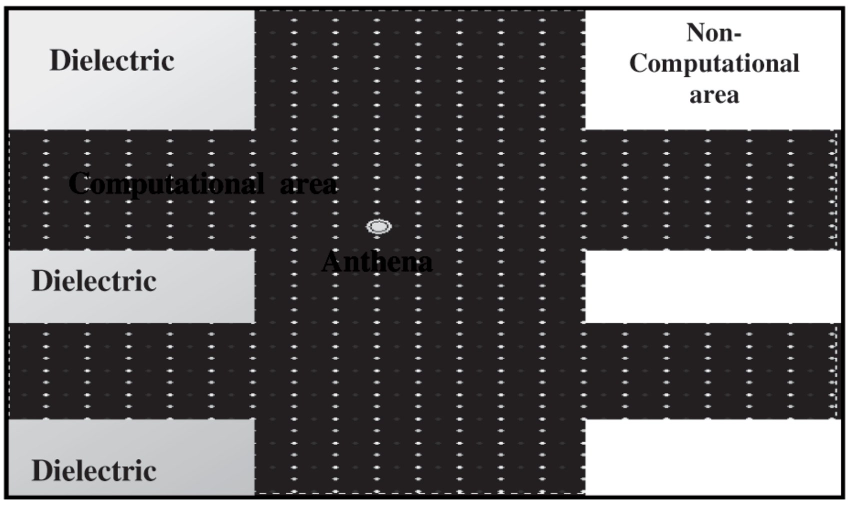


Fig. 1. Model of the 2 Dimensional FDTD Simulation.

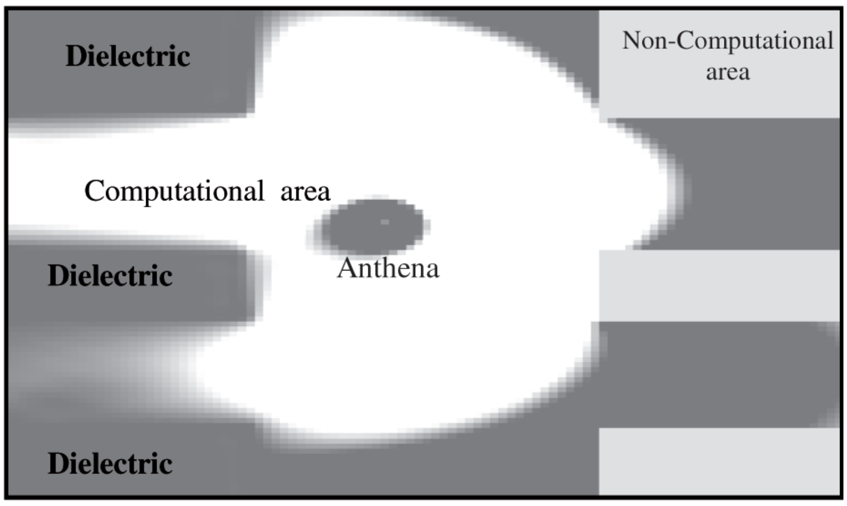


Fig. 2. FDTD Simulation in Action.

$$\begin{cases} \bar{E}_z^n(i,j) = CA_z(i,j)\bar{E}_z^{n-1}(i,j) + CB_z(i,j)\cdot \\ \qquad [\bar{H}_y^{n-0.5}(i+1,j) - \bar{H}_y^{n-0.5}(i+1,j) + \\ \qquad + \bar{H}_x^{n-0.5}(i,j-1) - \bar{H}_x^{n-0.5}(i,j+1)] \\ \bar{H}_x^n(i,j) = \bar{H}_x^{n-1}(i,j) + RC\cdot[\bar{E}_z^{n-0.5}(i-1,j) - \bar{E}_z^{n-0.5}(i+1,j)] \\ \bar{H}_y^n(i,j) = \bar{H}_y^{n-1}(i,j) + RC\cdot[\bar{E}_z^{n-0.5}(i,j-1) - \bar{E}_z^{n-0.5}(i,j+1)] \end{cases} \tag{2}$$

This computational process can be executed in parallel way. In this case, FDTD computations in the mesh are divided into fragments assigned to computational partitions. Each partition contains a number of computations that are mapped onto separate processing elements of a parallel machine. For regular shapes of the computational area (e.g. rectangular), it can be done by a stripe or block partitioning that allows obtaining almost ideal balance of computations on all available processors with minimal communication volume of data transmissions. For computational areas with irregular shapes, such an approach will not provide satisfactory partitioning. It needs a more advanced analysis of data dependencies. In this case, FDTD computation is represented by a data flow graph (Fig. 3) which is iteratively

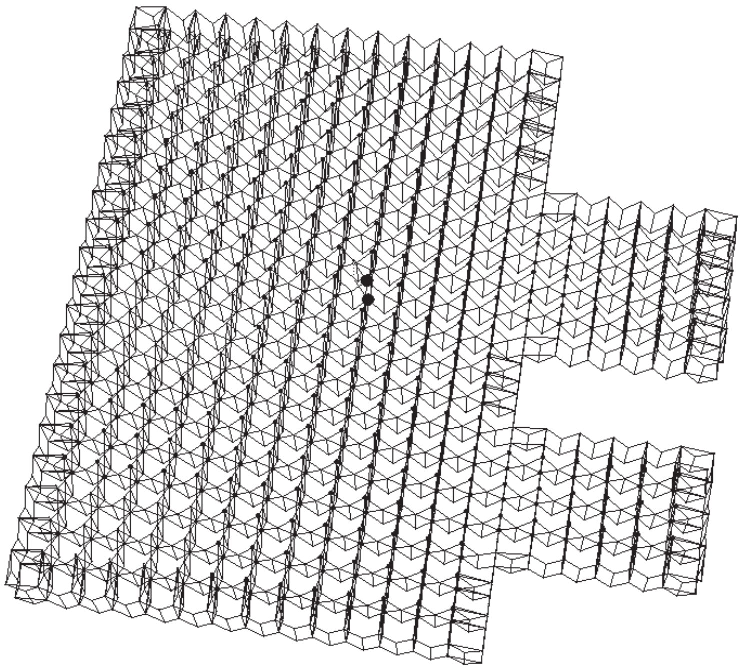


Fig. 3. Data Flow Graph for Two Iterations of the FDTD Problem for Irregular Computational Area.

transformed into a macro data flow graph. The number of macro nodes corresponds to given number of processing elements in an executive computer system. Such transformation from a data flow graph to a macro data flow graph takes into consideration both proper load of all processing nodes and the minimal number of data transmissions. To solve this problem, we have decided to design the FDTD method programs based on the macro data flow paradigm using an algorithm which is a combination of two algorithms described above: the redeployment (R) and the CDC algorithms. We call this algorithm RCDC (Redeployment with Connectivity-based Distributed Node Clustering). It is described in the last section.

3. Macro data flow graph optimization with cell redeployment algorithm

In this section, an outline of the redeployment algorithm will be presented. In the first step, we create a data flow graph of computations which represents basic data dependencies (see Fig. 3 and 4). This data flow graph will be used by a redeployment algorithm to define and optimize a macro data flow graph with n macro data nodes, where n is the number of processors. At the start, we create M macro nodes (where M is number of computational cells). Each computational cell is assigned to a separate macro node. Computations in each simulation sub-area are represented by one macro node. According to macro data flow paradigm, a macro node can be executed only if all external input data have arrived to the physical processor on which this macro node has been mapped. In the redeployment algorithm we do not perform any geometrical analysis of computational area. The whole optimization process is based on analysing data dependencies in the computational FDTD mesh.

The optimization algorithm is composed of three steps: simulation area partitioning, macro nodes merging and redeployment of cells. During the partitioning step we define macro data flow nodes for a given computational area. First, we determine computational "leader" nodes in the FDTD mesh. We have implemented two methods for choosing leaders. In the first method, we create a coarse regular mesh (LM) of cells spanned over the computational

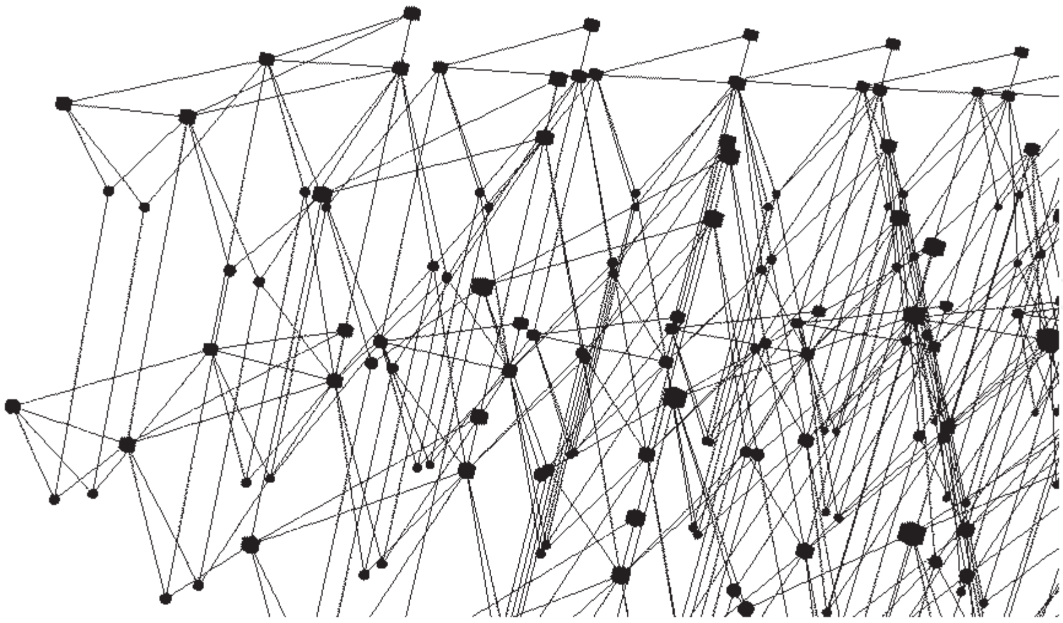


Fig. 4. Data Flow Graph for Two Iterations of the FDTD Problem (zoomed left bottom corner of Fig. 3).

mesh (CM). If cell A from LM is covered by cell B from CM, the cell B is qualified to be a leader. In an alternative method (P1), leaders are chosen only from circumference of the simulated area. Next, computational cells (that have not been earlier qualified as leaders) are gradually included into macro nodes assigned to the nearest leader. Additionally, all data-dependent macro nodes are connected by weighted edges. The weight represents the communication volume. The number of macro data flow nodes significantly exceeds the assumed number of processors. Additionally, already created macro data flow nodes can vary in their computational load. Such a macro data flow graph will be further optimized in the next phase, which is a merging phase. It enables reducing a total number of macro nodes to be equal to the number of processors with simultaneous coarse load balancing. Two macro nodes A and B are merged if: node A has the smallest execution time in the whole macro data flow graph and node B fulfills one chosen criterion from those shown in Table 1.

In the final phase, the program macro flow graph obtained after merging must be transformed so as to reduce program execution time by load balancing in executive processors for a given system configuration. Standard MDFG (macro data flow graph generated for all iterations of the FDTD problem) will be transformed into a Macro Node Communication Graph (MNCG - macro data flow graph generated for one iteration of the FDTD problem). The MDFG can be “compressed” into a MNCG because in each iteration, the pattern of data dependency is the same. So, in our algorithm, we can analyze only data dependencies in one iteration. After that, we identify the set of all cliques existing in the MNCG. A clique is a set of all macro nodes that are directly connected by edges with one, "central" macro node.

The main idea of this phase of the algorithm is based on equalizing execution time among all cliques. To achieve it, we will redeploy chosen subsets of computational cells between two selected cliques. The number of cells that can be redeployed is determined by the difference between average execution time of one clique and average execution time of one step in the MDFG. The whole redeployment phase is constantly monitored by execution time checking, so our optimization method in a step-by-step way tries to find the best data distribution among the executive processors, in which a given simulation problem will be performed.

Id	Rule priority	Description
MR0	Computational load balancing	Two least loaded adjacent nodes will be merged
MR1	Computational load balancing	The most loaded node will be merged with the least loaded adjacent node
MR2	Computational load balancing	The least loaded node will be merged with the most loaded adjacent node
MR3	Communication optimisation - edge cut reduction	Two most loaded adjacent nodes will be merged
MR4	Communication optimisation - edge cut reduction	The node with the biggest communication volume will be merged with its neighbour with the biggest communication volume
MR5	Communication optimisation - edge cut reduction	The node with the smallest communication volume will be merged with its neighbour with the biggest communication volume
MR6	Communication optimisation - edge cut reduction	The node with the smallest communication volume will be merged with its neighbour with the smallest communication volume
MR7	Communication optimisation - edge cut reduction	The node with the biggest communication volume will be merged with its neighbour with the biggest communication volume
MR8	Computational load balancing with edge cut reduction	The least loaded node will be merged with the adjacent node with the biggest communication volume
MR9	Computational load balancing with edge cut reduction	The least loaded node will be merged with the adjacent node with the lowest communication volume

Table 1. Chosen Merging Rules for Redeployment Algorithm.

The optimization algorithm requires precising three input parameters which describe problem configuration: a macro data flow graph, the speed of computational node and the throughput of available communication system. A simplified scheme of the redeployment algorithm is presented in Fig. 5. After all redeployment steps, we must re-compute an execution time for a modified MNCG graph. If new execution time is bigger than the old one, the redeployment is not profitable, and two cliques chosen for the redeployment operation are marked as examined. Because moving the computational cells between these two cliques is not profitable, they will not be chosen for next redeployment operation. If the execution time is better, the last redeployment operation can be validated. The optimization algorithm will be finished in the three following cases:

1. All cliques are marked as examined – it is not possible to perform any new redeployment operation.

2. The assumed number of consecutive redeployment steps, does not produce any increase of speedup – it is not possible to perform any redeployment with success (it is useful for large number of cliques).
3. The *CliqueB* parameter is close to 1, where *CliqueB* is a value, which determines clique balance in the MNCG graph, computed from the following equation:

$$CliqueB = \frac{absolute(L - S)}{total\ number\ of\ cliques}$$

where: L (S) is the number of cliques whose maximal execution time is larger (smaller) than the average execution time for the MNCG. If none of these conditions are met, then next two cliques are chosen and the redeployment step is repeated.

4. CDC - partitioning algorithm

The CDC (Connectivity-based Distributed Node Clustering) algorithm is a graph partitioning algorithm which is used to divide peer to peer networks into a given number of clusters. Unlike the redeployment method, the CDC is a decentralized algorithm. In this algorithm, only the nearest vicinity of nodes is needed to be analyzed to perform efficient partitioning operation whereas in the redeployment algorithm we must know the shape of whole computational area.

The CDC algorithm consists of two phases:

1. Phase 1 - choosing leader nodes (called originators (Ramaswamy et al., 2005)) – it can be done similarly as in the redeployment algorithm, but here the number of originators must be exactly equal to the number of computational nodes. In order to fulfill this condition, we have implemented another method of choosing originators. In our new method, we sort all computational cells by their coordinates (first by Y co-ordinate and after by X coordinate). After that we set every P cells to be originators, where

$$P = \frac{total\ number\ of\ computational\ cells}{number\ of\ processors}$$

2. Phase 2 - It is an iterative phase. It begins, when each node, which was previously chosen as an originator, sends a messages to its neighbor nodes. In all next iterations, each node that has received any messages in the previous iteration, re-sends the messages to their neighbors. Each message consists of following attributes:
   - source originator id (OID) – it indicates also unique identity of a cluster. This attribute is set by originator node, and it cannot be changed by any other nodes.
   - weight (W) – it describes a distance from originator node with a given OID number. The originator node sets W to 1.
   - time to live (TTL) – it describes, how many times this message will be resent.

The whole optimization process is parameterized using several factors. The most important are: MinWeight and MaxTTL. Both of them are used to reduce the total number of messages during the execution of the CDC algorithm. Parameter MaxTTL is an initial value for TTL attribute of each message. As value of MaxTTL increases, the number of generated messages increases as well. MinWeight parameter determines the minimal value for a message. If attribute W in a message M is smaller than MinWeight parameter, the message M will be

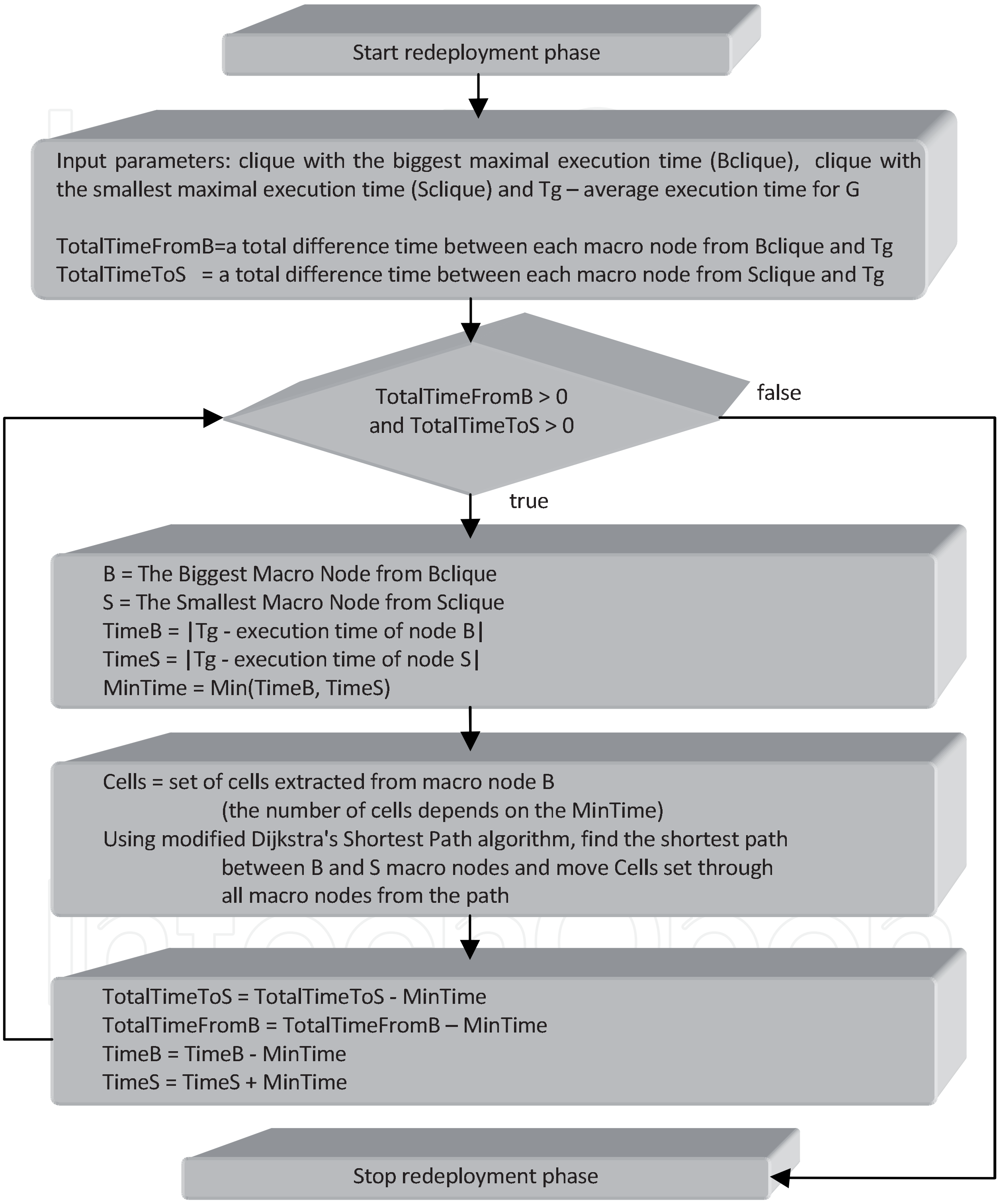

Start redeployment phase
Input parameters: clique with the biggest maximal execution time (Bclique), clique with the smallest maximal execution time (Sclique) and Tg – average execution time for G
TotalTimeFromB=a total difference time between each macro node from Bclique and Tg
TotalTimeToS = a total difference time between each macro node from Sclique and Tg
TotalTimeFromB > 0 and TotalTimeToS > 0
false
true
B = The Biggest Macro Node from Bclique
S = The Smallest Macro Node from Sclique
TimeB = |Tg - execution time of node B|
TimeS = |Tg - execution time of node S|
MinTime = Min(TimeB, TimeS)
Cells = set of cells extracted from macro node B (the number of cells depends on the MinTime)
Using modified Dijkstra's Shortest Path algorithm, find the shortest path between B and S macro nodes and move Cells set through all macro nodes from the path
TotalTimeToS = TotalTimeToS - MinTime
TotalTimeFromB = TotalTimeFromB – MinTime
TimeB = TimeB - MinTime
TimeS = TimeS + MinTime
Stop redeployment phase

Fig. 5. Scheme of the Redeployment Phase of the Algorithm.

```
Cell C receives a message M
Table T on cell C is updated T[M.OID]= T[M.OID]+ M.W
A new message (Mnew) on the cell C is created
Mnew.OID = M.OID
Mnew.TTL = M.TTL−1
Mnew.W = Mnew.W/(Degree of cell C)
if Mnew.TTL>0 and Mnew.W>= MinWeight then
    the message Mnew will be re−sent to all neighbors of C cell
if Mnew.TTL<0 or Mnew.W<MinWeight then
    the message Mnew will be destroyed
if any messages exist then
    goto 1
```

Fig. 6. Second Phase of the CDC Algorithm.

destroyed. Each node contains a table T with total sum of weights from all received messages sourced by originators. The CDC algorithm in phase 2 is presented in Fig. 6. Phase 2 is repeated until all messages are destroyed. After that, each cell is attached to a cluster C, where C is equal to the index of the element from T with maximal value.

5. Experiment results

Both of these methods have been implemented, executed and tested for several shapes of simulation area and for several system configurations. To perform it we have introduced a simple architectural model of the executive system (Bharadwaj et al., 1996). It includes homogenous multiprocessor systems and is described by two values: processor computational speed, communication performance, see Table 2. The RB RDMA (Rotating Buffers-based Remote Direct Memory Access) is a special kind of communication facility. It enables data transmissions with a very small engagement of processor time (Smyk & Tudruj, 2003; 2004; Hitachi, 1997), which is usually done in the background of computations. The transmissions proceed without any data buffering by the operating system, so the overheads of this kind of communication is very small.

5.1 Rotating buffers infrastructure RB RDMA

The logical structure of the memory used in the rotating buffers facility is presented in Fig. 7. The RB RDMA infrastructure was designed for Hitachi SR2201 supercomputer, but it can be easily adapted to various distributed parallel computational systems like Cell/BE PS3 for example or even to cluster systems with network cards supporting DMA communication (or other compatible).

Local memory of a computational node is logically divided into two parts *LAM* (*Locally Accessed Memory*) with data used only for local computations with direct access from application program level and *GAM* (*Globally Accessed Memory*) with data used for data exchange. Access to *GAM* is available (from local and remote sites) only through the rotating buffers memory infrastructure.

GAM area is divided into *N* separate sub-area pairs: *RCA* (*Remote Confirmation Area*) and *RDM* (*Remote Data Memory*), where *N* is the number of remote processors. Each pair is used to perform communication between two given remote processors. The numbers of rotating buffers in the send and receive memory parts are fixed and denoted by *NSB* and *NRB*,

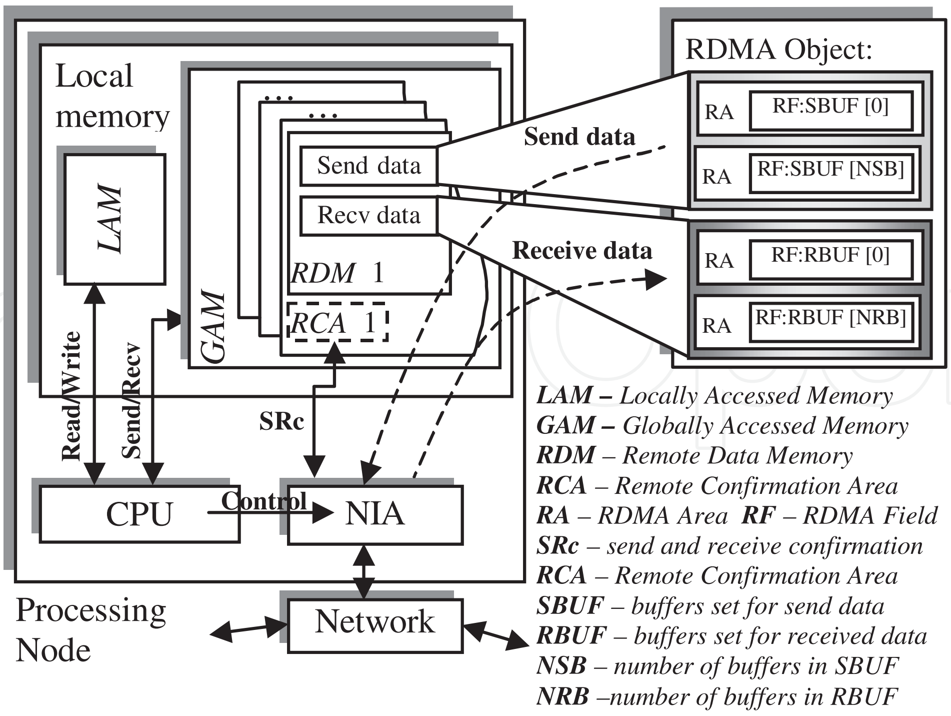

Local memory
LAM
GAM
...
...
Send data
Recv data
RDM 1
RCA 1
Send data
Receive data
RDMA Object:
RA
RF:SBUF [0]
RA
RF:SBUF [NSB]
RA
RF:RBUF [0]
RA
RF:RBUF [NRB]
Read/Write
Send/Recv
SRc
CPU
Control
NIA
Processing Node
Network
LAM – Locally Accessed Memory
GAM – Globally Accessed Memory
RDM – Remote Data Memory
RCA – Remote Confirmation Area
RA – RDMA Area RF – RDMA Field
SRc – send and receive confirmation
RCA – Remote Confirmation Area
SBUF – buffers set for send data
RBUF – buffers set for received data
NSB – number of buffers in SBUF
NRB –number of buffers in RBUF

Fig. 7. Memory Structure in Rotating-Buffers Method (For One processing Node on the HITACHI SR2201 supercomputer).

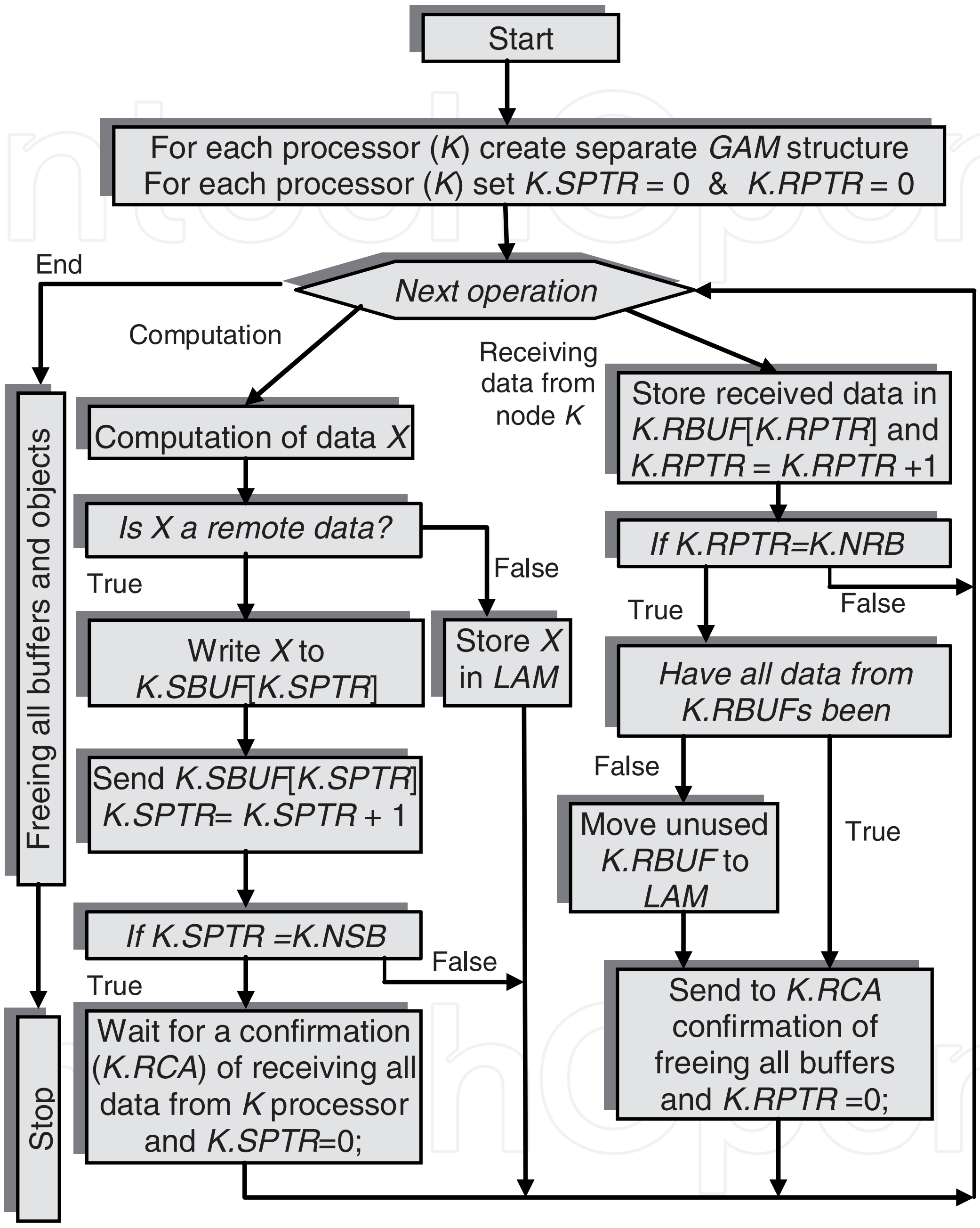
respectively. All buffers which are defined in a *RDM* area are used only for data transmission. To avoid possible data overwriting, an additional control has to be introduced. This control is based on the *RCA* areas which are assigned independently to each *RDM* area. Each *RCA* area is intended to send and receive additional control messages which determine if the buffers from a *RDM* are ready to receive new data. A *RCA* consists of only two sets of one buffer each (by analogy to a *RDM* where the number of buffers can change, here the *NSB* and *NRB* numbers are always equal 1). They are used to exchange synchronized messages between two processors.

The control flow in the rotating buffers method for one processing node is presented in Fig. 8. Data are exchanged between a local processor and a remote processor *K*. On both these nodes, the described above control and communication infrastructure was created. Additionally, for each processor two pointers *K.SPTR* and *K.RPTR* are created. They are used to indicate a next free buffer in which new data (to be sent to processor *K*) will be placed (*K.SPTR*) or new data just received from *K* will be written (*K.RPTR*). These two pointers determine control for a rotating access to available buffers and introduce periodical synchronization between two communicating nodes, which assures that no data which are transferred from one node to another, will be lost (overwritten).

5.2 Redeployment algorithm efficiency

First, we have tested the speedup of macro data flow graph execution after program profiling without and with the use of cell redeployment optimization algorithm, see Figure 9.

We can observe that the best speedup was obtained for computational systems with shared memory (FF, MF, SF) and in two cases with RB RDMA communication (MM, SM). It is independent of the number of processors, with or without redeployment of computational nodes, and even of the efficiency of a single computation node. We can see the significant


Start
For each processor (K) create separate GAM structure
For each processor (K) set K.SPTR = 0 & K.RPTR = 0
End
Next operation
Computation
Receiving data from node K
Freeing all buffers and objects
Computation of data X
Is X a remote data?
True
False
Write X to K.SBUF[K.SPTR]
Store X in LAM
Send K.SBUF[K.SPTR]
K.SPTR= K.SPTR + 1
If K.SPTR =K.NSB
True
False
Wait for a confirmation (K.RCA) of receiving all data from K processor and K.SPTR=0;
Stop
Store received data in K.RBUF[K.RPTR] and K.RPTR = K.RPTR +1
If K.RPTR=K.NRB
True
False
Have all data from K.RBUFs been
False
True
Move unused K.RBUF to LAM
Send to K.RCA confirmation of freeing all buffers and K.RPTR =0;

***K.SBUF / K.RBUF** – buffers set for send/received data to/from K*
***K.SPTR / K.RPTR** – pointer for free buffer from K.SBUF / K.RBUF*
***K.NSB** / **K.NRB** – number of buffers in K.SBUF / K.RBUF*

Fig. 8. Control Flow of the Rotating-Buffers Method for One Processing Node.

Architecture symbol	Computational speed of a single processor	Communication speed between two processors
FF	Fast – 1GFlops	Fast – Shared memory
FM	Fast – 1GFlops	Medium – RB RDMA
FS	Fast – 1GFlops	Slow – MPI
MF	Medium – 0.3 GFlops	Fast – Shared memory
MM	Medium – 0.3 GFlops	Medium – RB RDMA
MS	Medium – 0.3 GFlops	Slow – MPI
SF	Slow – 0.02 GFlops	Fast – Shared memory
SM	Slow – 0.02 GFlops	Medium – RB RDMA
SS	Slow – 0.02 GFlops	Slow – MPI

Table 2. Symbols and Architectural Model Parameters.

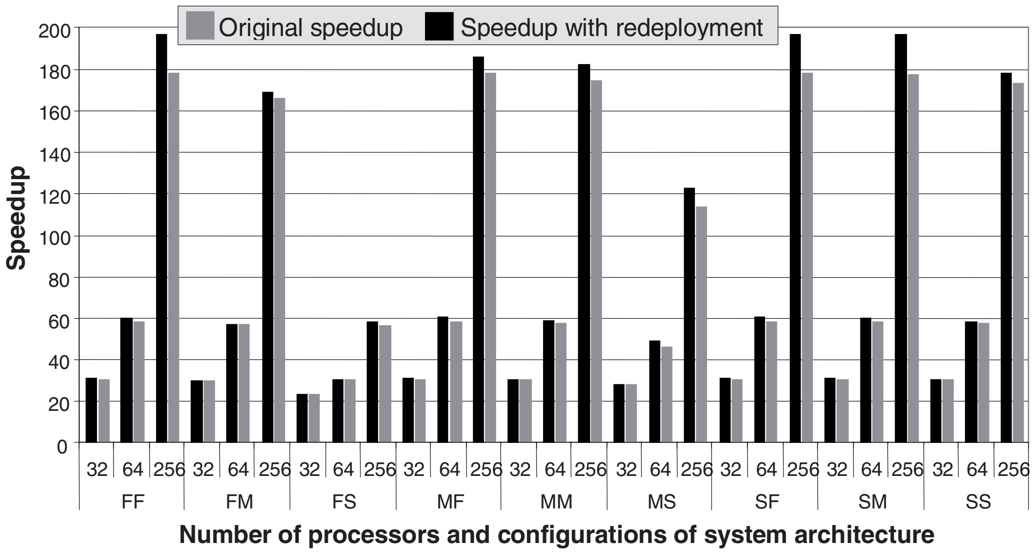


Fig. 9. MDFG Execution Speedup Versus Single Node Execution for Various Computational System Configurations.

increase of speedup for 256 processors. We can also notice that for systems with RB RDMA communication and with fast processors (FM) slightly worse speedup was obtained. It can be explained by bad relation between computational and communication parameters of the system, which could not match program requirements. It is especially noticeably in the case of systems with slow communication (FS, MS) where sometimes the speedup decreased dramatically. We can observe also that a noticeable increase of speedup (~10% for 256 processors) appears when a redeployment operation has been applied. It means, that the balancing of execution time is possible only for configurations with large number of processors and with relatively fast communication systems.

5.3 Redeployment algorithm versus CDC

In next experiments, we have compared the efficiency of the redeployment algorithm with our implementation of the CDC algorithm. The results are presented in Figure 10.
We can see that the speedup of CDC in almost all cases has been much better than that of the redeployment optimization what would indicate the superiority of the CDC algorithm.

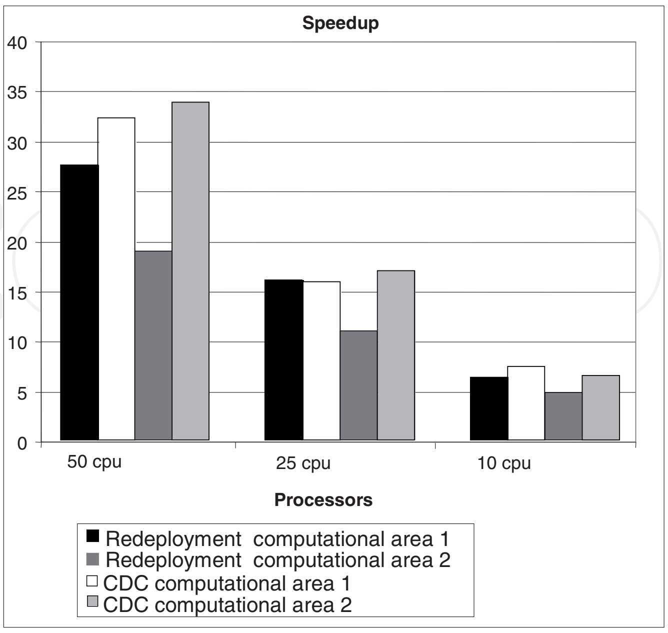


Fig. 10. Comparison of Speedup for the Redeployment Method with the CDC Algorithm.

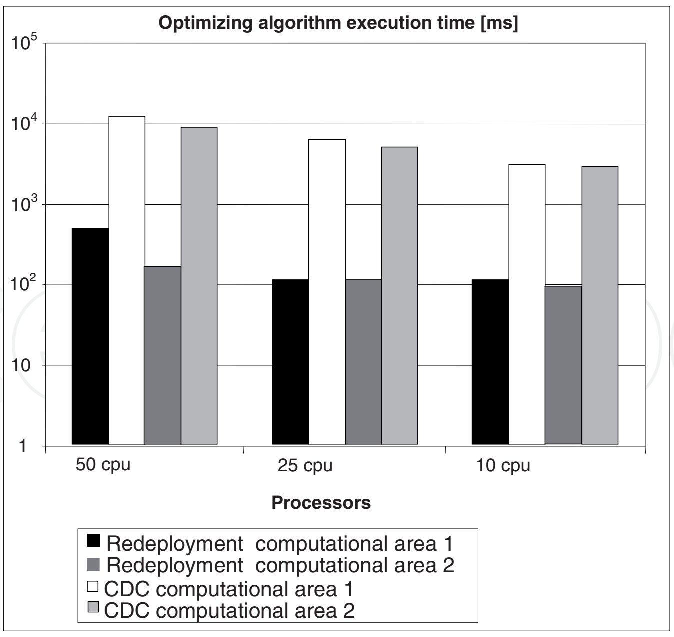


Fig. 11. Comparison of the Total Optimization Execution Time for Redeployment and CDC Algorithm.

Unfortunately, we have observed some unfavorable features of this algorithm. The first one is that the execution time of this algorithm (Figure 11) was from 25 to 50 times longer in comparison to redeployment algorithm. It can completely eliminate the use of CDC algorithm for computational areas with large number of cells. In our experiments, we could not simulate areas larger than 1000 cells. It is because large number of messages are generated. We observed also that CDC algorithm is very sensitive to two simulation parameters: MinWeight and initial value of TTL time. These parameters are decisive for messages lifespan and unfortunately, their values considerably depend on the shape of computational area.

6. Hierarchical algorithm

Based on the results presented in the previous section, we can observe that the redeployment algorithm is much faster than the CDC algorithm. The convergence time of the redeployment algorithm is almost two times shorter in comparison to CDC. However, for system configurations with a big number of processors, the parallel speedup of the FDTD programs obtained using the redeployment method is lower than that obtained with the use of the CDC algorithm. A difficult problem in CDC is setting the initial values of the parameters MinWeight and TTL time. It is especially true for a large number of executive processors. Taking into considerations all pros and cons of the two methods we propose a hierarchical method of FDTD program optimization (Redeployment with Connectivity-based Distributed Node Clustering - RCDC). This method consists of two main steps. In the first step, we apply the standard redeployment algorithm. In the second step, we switch to a modified CDC method. The standard redeployment algorithm step is used here to reduce the number of nodes in the input data flow graph (in fact to reduce the optimization time), while the CDC algorithm step will be exploited to obtain the best possible parallel simulation speedup.

Efficiency of the RCDC algorithm strongly depends on the choice of the time point in which switching between these two algorithms should take place. Because the parallel simulation speedup provided with the CDC algorithm is usually significantly better than that obtained by the redeployment, we decided to modify the standard redeployment algorithm so as to execute it in two phases: generation of an initial MDFG based on wave propagation area partitioning and the MDFG nodes merging with load balancing to obtain the given number of macro nodes. In fact, the number of macro nodes obtained in the second phase is from 5 to 20 times bigger than the assumed number of executive processors. The final reduction of the number of macro nodes together with communication optimization (to minimize and balance internodes data transmissions.) will be performed during the CDC step. In our implementation, the CDC algorithm does not take into account load balancing in the executive processors, so after this step, some load imbalance is possible. To avoid it, we have introduced the last phase with redeployment of computational cells.

The final scheme of the RCDC algorithm is as follows (see Fig. 12):

1. Generation of initial MDFG based on wave propagation area partitioning;
2. MDFG nodes merging (with initial load balancing) and optimization with re-deployment of computational cells;
3. CDC optimization (communication optimization).

A crucial problem in the RCDC algorithm is a decision when we should switch from phase 2 to 3. To answer this question, we can consider several independent methods. In the first method, we can compute the diameter of our MDFG and if it corresponds (it means that the most distant macro nodes can send and receive messages to/from each others in the

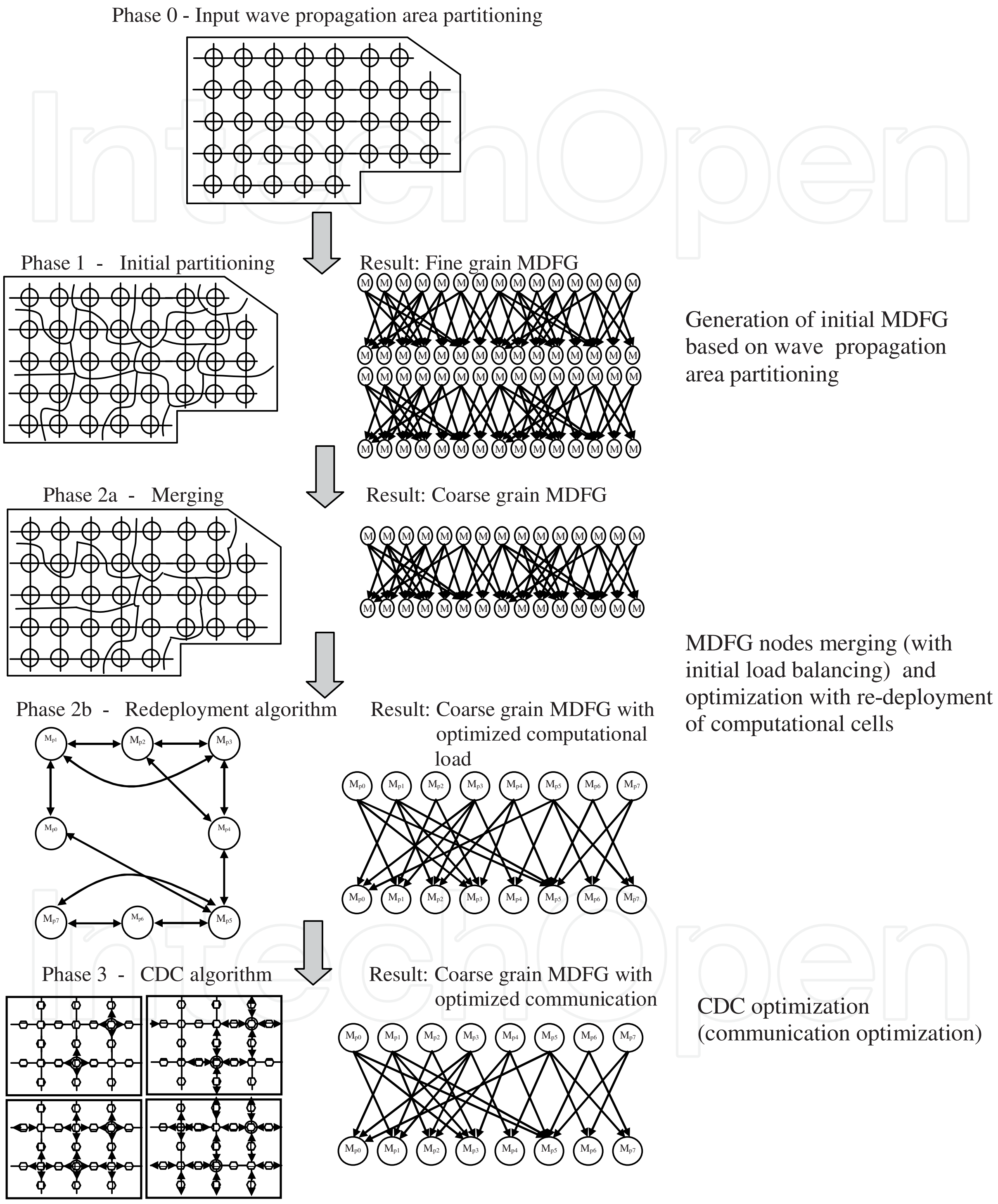

Phase 0 - Input wave propagation area partitioning
Phase 1 - Initial partitioning
Result: Fine grain MDFG
M
Generation of initial MDFG based on wave propagation area partitioning
Phase 2a - Merging
Result: Coarse grain MDFG
MDFG nodes merging (with initial load balancing) and optimization with re-deployment of computational cells
Phase 2b - Redeployment algorithm
Mp0
Mp1
Mp2
Mp3
Mp4
Mp5
Mp6
Mp7
Result: Coarse grain MDFG with optimized computational load
Phase 3 - CDC algorithm
Result: Coarse grain MDFG with optimized communication
CDC optimization (communication optimization)

Fig. 12. RCDC Hierarchical Algorithm Scheme.

Problem configuration	Computational area size	Computational area type	Number of processors
1	Small	Regular	50
2	Large	Irregular	50
3	Small	Regular	25
4	Large	Irregular	25
5	Small	Regular	10
6	Large	Irregular	10

Table 3. Problem Configurations Used in the Experiments.

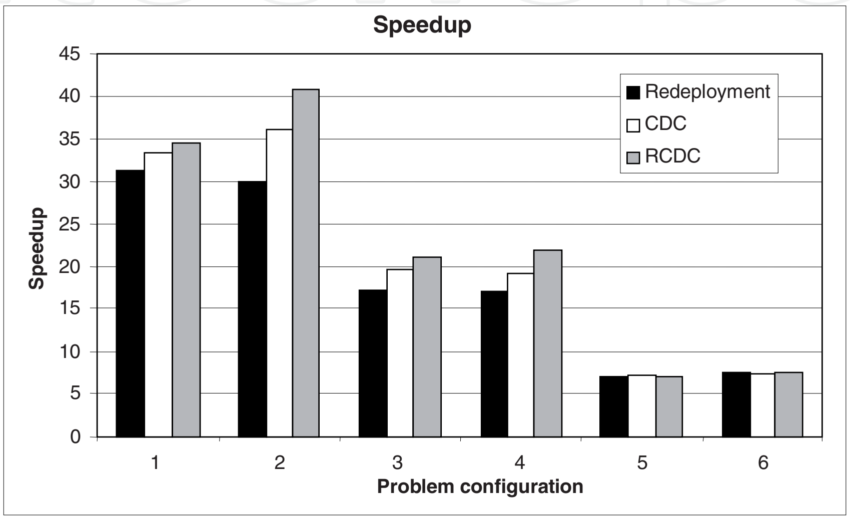


Fig. 13. Comparison of the Parallel Simulation Speedup for Redeployment, CDC and RCDC Algorithms.

sense of the CDC algorithm) with MinWeight and/or TTL time parameters, we can start phase 3. This method needs some additional computations but its estimation can be precise only for regular graphs. In case of very irregular graphs, it can distort information about the graph structure, which can cause that the optimization time for phase 3 will be much bigger. In the second method, we can assume a time of the optimization after which phase 3 is entered. In the third method, we can assume a static number of macro nodes created in our MDFG after which the phase 2 will be terminated and a passage to phase 3 will take place. In our implementation, we have assumed the use of this method. It does not introduce any additional costly computations in phase 2 and it allows us to find a coarse estimation of the parameters (MinWeight, TTL time) needed for phase 3. The number of macro nodes obtained in phase 2 is very important for the rest of the algorithm. If it is too large, the total optimization time will take too much time, and if it is too small, the obtained speedup can be unsatisfactory. We have assumed that dependently on the structure of the MDFG (regular or irregular), switching from phase 2 to 3 is done when the number of macro nodes is from 5 to 20 times bigger than the assumed number of computational macro nodes.

As it can be expected, the RCDC hierarchical method produced better partitioning in comparison to redeployment and the CDC methods. It is especially visible for configurations with large number of processors (see Figure 13, configurations 2 and 4).

For small computational areas (configurations 5 and 6), no significant differences between all discussed algorithms are observed. Additionally, we can see that when the number

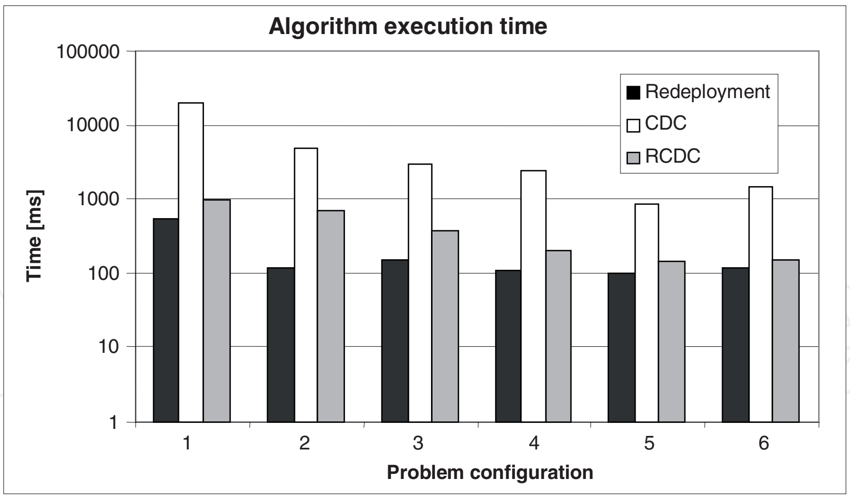


Fig. 14. Comparison of the Algorithm Execution Time for Redeployment, CDC and RCDC Algorithms.

of processors grows, the RCDC algorithm has much more degrees of freedom and it can produce better partitioning. Even though, in our implementation of the CDC method, we have not considered problems concerned with load balancing, the partitioning efficiency was satisfactory. It is because, the granularity of the input graph was very fine and computational load in each cell was equal to others. In case of the RCDC, we must carefully switch between the redeployment and the CDC algorithm phases. The redeployment algorithm will be stopped if the number of macro nodes in MDFG divided by k (where k is a positive integer) is equal to the number of processors. If it is fulfilled, we can switch to phase 3 of the RCDC algorithm. The execution time of the optimization by the RCDC algorithm is much shorter than CDC method and slightly bigger than the redeployment algorithm, see Figure 14. It can be dynamically adjusted to the k value. When the k value grows, the behavior of the RCDC algorithm is closer to the CDC method. Otherwise it works like the redeployment algorithm.

7. Conclusions

In this chapter, we presented a hierarchical approach to the optimized program macro data flow graph design for execution of FDTD simulations in parallel systems. The presented RCDC algorithm combines two independent methods for the FDTD data flow graph optimization: the cell redeployment and CDC algorithm. There are several differences between these two methods. The first method is fully centralized and the macro data flow graph is created in three phases: computational area partitioning, merging and redeployment. The CDC method is decentralized with only local current knowledge of the simulation area. In the RCDC algorithms we wanted to merge both of these methods in order to obtain more efficient parallel simulation speedup (comparable to the speedup obtained in the CDC) and to shorten the execution time of the optimization. It turned out that such a hierarchical combination of the two algorithms has improved partitioning of data flow graphs for the FDTD problem and additionally such a hierarchical optimization takes significantly less time than the CDC method.

8. References

Lin, H.X., van Gemund, A.J.C. & Meijdam, J. (1996). Scalability analysis and parallel execution of unstructured problems, *Eurosim'96 Conference.*

Garey, M., Johnson, D. & Stockmeyer L. (1976). Some simplified NP-complete graph problems, *Theoretical Computer Science*, 1, pp. 237-267.

Khan, M.S. & Li, K.F. (1995). Fast Graph Partitioning Algorithms, *Proceedings of IEEE Pacific Rim Conference on Communications, Computers, and Signal Processing*, Victoria, B.C., Canada, May 1995, pp. 337-342.

Kerighan, B.W. & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs, *AT&T Bell Labs. Tech. J.*, 49:291-307, February 1970.

Kirkpatrick, S., Gelatt, C.D. & Vecchi, M.P. (1983). Optimization by simulated annealing, *Science*, 220(4598):671-680, May 1983.

Karypis, G. & Kumar, V. (1995). Unstructured Graph Partitioning and Sparse Matrix Ordering, *Technical Report*, Department of Computer Science, University of Minesota, 1995 (http://www.cs.umn.edu/~kumar).

Dutt, S. & Deng W. (1997). VLSI Circuit Partitioning by Cluster-Removal using Iterative Improvement Techniques, *Proc. IEEE International Conference on Computer-Aided Design*, pp.350-355.

Stone, H.S. & Bokhari, S.H. (1978). Control of distributed process, *IEEE Computer*, 11(7):97-106, July 1978.

Fiduccia, C.M. & Mattheyses, R.M. (1982). A Linear Time Heuristic for Improving Network Partitions., DAC, pp. 175–181.

Bharadwaj, V., Ghose, D., Mani, V., Robertazi T.G. (1996). Scheduling Divisible Loads in Parallel and Distributed Systems, *IEEE Computer Society Press*, Los Alamitos, California.

Smyk, A. & Tudruj, M. (2003). RDMA Communication Based on Rotating Buffers for Efficient Parallel Fine-Grain Computations, *PPAM 2003*, Czestochowa, Poland.

Smyk, A. & Tudruj, M. (2004). RDMA Control Support for Fine-Grain Parallel Computations, *PDP 2004*, La Coruna, Spain.

Hitachi Ltd. (1997). HI-UX/MPP - Remote DMA -C- User's Guide Manual Number: 6A20-3-021-10(E), Second Edition: January 1997.

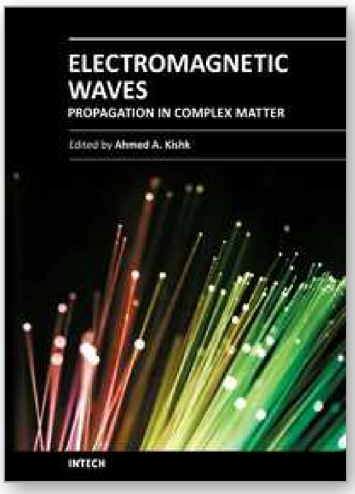
Ramaswamy, L.; Gedik, B.; Liu, L. (2005). A distributed Approach to Node Clustering in Decentralized Peer-to-Peer Networks, *IEEE Transactions on Parallel and Distributed Systems*, Vol.16, No.9, September 2005.

Smyk, A. & Tudruj, M. (2006). Parallel FDTD Computations Optimized by Program Macro Data Flow Graph Redeployment, *PARELEC 2006*, Bialystok, Poland, September 2006.

Walshaw, C., Cross, M., Everett, M.G., Johnson, S. (1995). JOSTLE: Partitioning of Unstructured Meshes for Massively Parallel Machines, *Parallel Computational Fluid Dynamics: New Algorithms and Applications 1995*, Elsevier.

http://www.labri.fr/perso/pelegrin/scotch.

http://glaros.dtc.umn.edu/gkhome/views/metis.



Electromagnetic Waves Propagation in Complex Matter

Edited by Prof. Ahmed Kishk

ISBN 978-953-307-445-0

Hard cover, 292 pages

Publisher InTech

Published online 24, June, 2011

Published in print edition June, 2011

This volume is based on the contributions of several authors in electromagnetic waves propagations. Several issues are considered. The contents of most of the chapters are highlighting non classic presentation of wave propagation and interaction with matters. This volume bridges the gap between physics and engineering in these issues. Each chapter keeps the author notation that the reader should be aware of as he reads from chapter to the other.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Adam Smyk and Marek Tudruj (2011). Optimization of Parallel FDTD Computations Based on Program Macro Data Flow Graph Transformations, Electromagnetic Waves Propagation in Complex Matter, Prof. Ahmed Kishk (Ed.), ISBN: 978-953-307-445-0, InTech, Available from: http://www.intechopen.com/books/electromagnetic-waves-propagation-in-complex-matter/optimization-of-parallel-fdtd-computations-based-on-program-macro-data-flow-graph-transformations

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821