Peer-reviewed & Open access journal **ATI - Applied Technologies & Innovations**
www.academicpublishingplatforms.com Volume 4 I Issue 1 I April 2011 Ipp. 12-21

Algorithm for micropipeline buffer control

Dimitar Tyanev, Violeta Bozhikova, Stefan Gerganov, Bozhidar Georgiev
Faculty of Computing and Automation,Technical University of Varna, Bulgaria
e-mail: vbojikova2000@yahoo.com

The paper focuses on the problem of hardware implementation of the computational process containing conditional transitions. Asynchronous organization of pipelines at microoperational level is provided for the hardware. Its characteristic feature is that it includes both one and multi-cycle micropipeline units. Because of these circumstances, the outgoing pipeline results do not correspond in the same order to the tasks running in the pipeline. The article presents the synthesized original logical structure of the micropipeline buffer and a specific to its service strategy through which their correct order is restored when reading results from the buffer. Another programming model of the structure of the buffer is described, by which its behavior was studied in different possible situations. In addition, a programming model of the structure of the buffer was created and its behavior in different possible situations is examined. The results of numerical experiments with the programming model are presented. Based on them recommendations are formulated about the parameters of the buffer and the structure of the pipeline.

Keywords: Micropipeline, conditional transition, restore order, micropipeline buffer.

Nature of the problem

Consider the presented structure in Figure 1 of an exemplary algorithm that is hardware implemented and whose performance is pipeline organized. It should be borne in mind that this algorithm is detailed and its executable blocks are implemented using hardware methods outlined in (Tyanev, Josiffov, Kolev, 2007), (Tyanev, Kolev, Josifov, 2007), (Tyanev, Kolev, Yanev, 2007), (Tyanev, Kolev et al., 2009), (Tyanev, Yanev et al., 2009) and (Tyanev, Kolev et al., 2010). Each executable block of the block diagram represents a separate one-stroke or multiple-stroke micropipeline unit, as set out in (Tyanev, 2009), (Tyanev and Popova, 2010) and (Kolev and Tyanev, 2010). Obviously, the presented algorithm can be defined as "branched". The condition for transition CJ (conditional jump) in the model algorithm forms the following possible ways of the computational process:

a. *Begin*; 1; 2; (CJ=true); 3; 6; *End.*

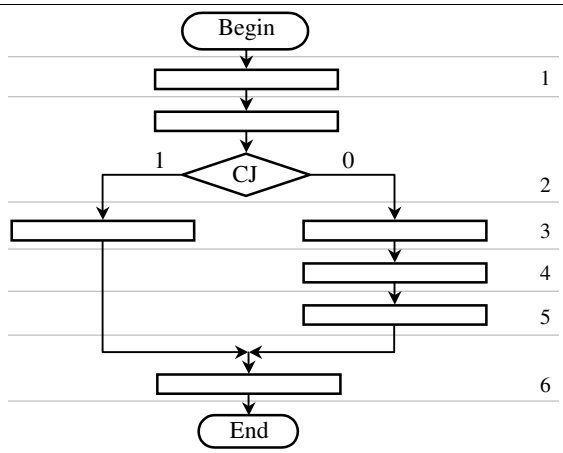
b. *Begin*; 1; 2; (CJ= false); 3; 4; 5; 6; *End.*

where with 1,2,3, ... are assigned level numbers associated with the implementation of the algorithm.

If we consider that each *Begin-End* algorithmic path is unique, the corresponding successive micropipeline units in the parallel branches are referred to the same row of the pipeline, for which there are a total of 6.

Therefore, at level 3 for example, two micropipeline units are assigned. At the time of the execution of each task, at reaching that level, according to the value of the condition of transition, only one of the units is used (the left one or the right one). At the common point (the entrance to the 6th level), which combines both branches, the queries, that accompany the intermediate results received in these branches, must refer to different tasks, launched earlier in the micropipeline. The order in which these results have reached the common point can not be expected to fully comply with the order in which tasks are running, to which they belong. In other words, it can hardly be expected that the results will be received in the correct order in the receiving unit.

FIGURE 1. STRUCTURE OF A MODEL ALGORITHM



The main reason for this conflict is the branching itself.

Considering the different number of micropipeline units in the branches as well as the delays introduced from them, it is quite possible to have a situation where the front of the calculation process of a later launched task is ahead of a previously started task at the common point.

As a result of this configuration is not correct to expect the order of the outgoing pipeline results to be fully consistent with the order of the running tasks.

In other words, the result that came out first, is hardly the result of the first launched pipeline task.

If the order of results is important for further calculations, it should be restored. So the reasoning above is the essence of the problem, to which this study is devoted. Also, this is one of the problems discussed in (Tyanev, 2009).

The problem of restoring the order of the calculation results that go down from pipeline is generally considered in (Patterson et al., 2005) and (Hennessy et al., 2003). The solution described there is about the pipeline of machine commands in digital processors. In this type of pipeline it arises because of the parallelism, artificially introduced in the "execution" level, leading to super-scalarity.

In this sense, the reasons discussed here for the problem arising on a micro-operational level are radically different.

While on the high (command) level the problem is usually solved by software, in the low (microoperational) level it should be solved by the hardware.

Pipeline's shadow

After the conclusion made above we could consider that the results that appear at the outcome of the pipeline, i.e. the results of the micro pipeline unit in the last sixth level of the example, will probably not be arranged correctly. This means that these results cannot be used immediately for further tasks and must be temporarily stored until their order is removed.

Since the output of the pipeline will be continuously giving out results, their storage could be realized by a buffer memory type FIFO. FIFO-buffer types are a natural choice for classic pipelines without branches.

In our case, the structure and the strategy for servicing the pipeline is considerably more complicated, but we will keep these conditional designations.

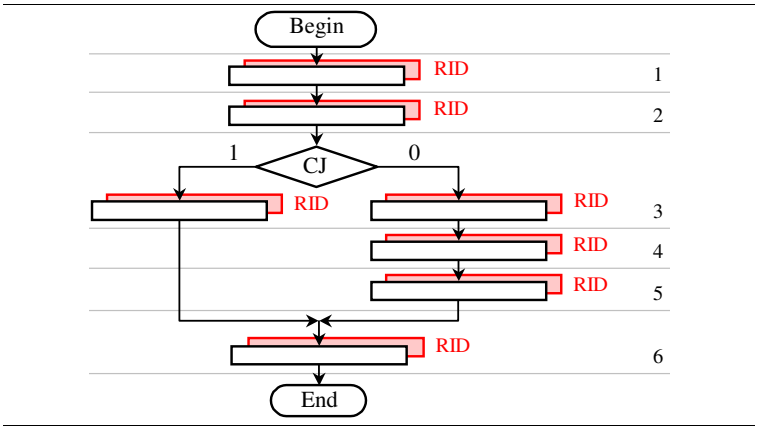
Let us start assuming that the number of cells in FIFO-buffer is equivalent to the levels in the pipelines, i.e. according to the example they will be 6. Let us assume further that the calculations are completed for 6 consecutive starts of the pipeline and the identity of the results is expressed with the sequence: 4, 2, 1, 5, 3 and 6. So we understand that the first result, descended from the pipeline, belongs to the task that is running as the fourth in order, and the result of the first started task came out as the third in order. The last, sixth result has come out without having been rearranged or outgrown. The numbers in the example-sequence do not represent results themselves, but only their starting number, i.e. on exit the initial numbers are not arranged in natural order, thereby illustrating the above assumption. If after the serial record the buffer is full, the recording should be blocked.

The task, to restore the correct order cannot be solved only through the results themselves. Additional information is needed. Each result (intermediate or final) must be accompanied by an identifier, which at any time will show to which task performed in the pipeline will belong to this result. This identifier must descend from the pipeline along with the final outcome of the task. The identifier should be used in the final stage when the result should be stored in pipeline's buffer in the right place. It follows that the pipeline buffer will be managed by a special algorithm other than FIFO-known algorithm.

Additional hardware is needed in the pipeline to materialize the above considerations. Each micro- pipeline unit together with the input data will have to adopt the identifier of this data. The unit will have to store this identifier at the end of carrying out the calculations, and will have to submit it to the next point with the result that it has received.

Therefore, the registers of the pipeline should be supplemented with registers supporting identifiers. The passive role of these registers allows us to call the whole totality "shadow" of the pipeline, in which Figure 2 gives us a clear picture.

FIGURE 2. POSITION OF THE REGISTERS OF IDENTIFIERS - "SHADOW" OF THE PIPELINE



The identification scheme proposed here for restore order, which should take the results, is not based on order numbers. The use of order numbers or addresses (Patterson et al., 2005) and (Hennessy et al., 2003) in each variant is potentially endangering to the algorithm that maintains the buffer pipeline by fixing the same tags. This is due to the extreme length of the logical units that can generate them. Therefore, the order numbers will be used here only for clarification of the proposed new strategy.

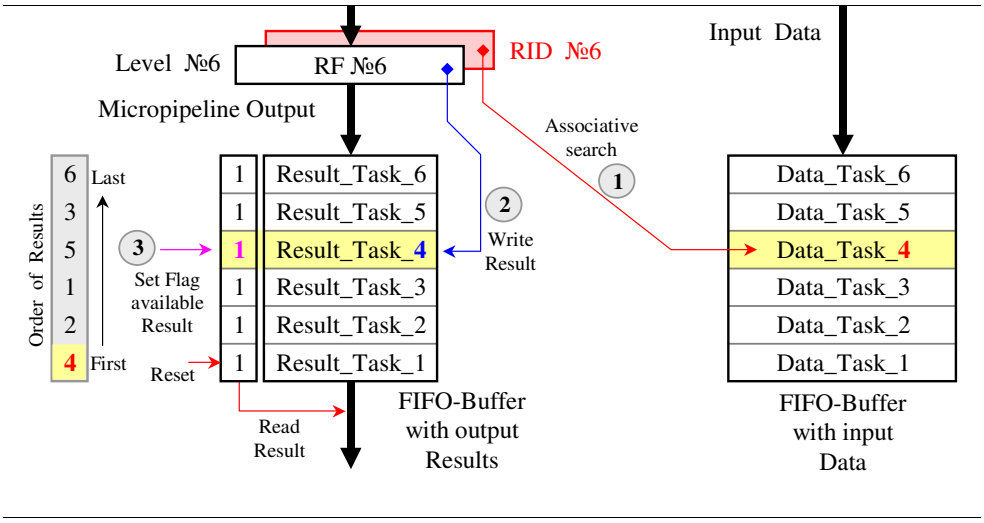
At any time and in any place in the pipeline, for the identification of both the intermediate and the final results, it is proposed here to use the input data of each task. Thus, input

data will accompany the obtained results and will follow their path from unit to unit along the front of the computing process. In the last unit, unchanged, they will come out together with the final result, for which they were used. The solution is illustrated in Figure 2, where behind the micro pipelines units, are visible RID registers, containing identification data.

Pipeline's buffer algorithm for servicing

Output data (results and identifiers for their association) are used to solve the task in the manner illustrated in Figure 3.

FIGURE 3. STRUCTURE OF MICRO-PIPELINED BUFFER



Only the register-latch RF of the last micropipeline unit from which the finished results "descend" is shown in the structure and behind it - the register with identifications data RID, accompanying the results. The micropipeline buffer's structure contains FIFO-buffer, in which the input data of each new task, started in the pipeline is consistently recorded. Please note that the input data in this buffer is arranged in the correct starting order. As an example, a sequence of 6 tasks for the pipeline operation is given and should complete these tasks in the order already shown - 4, 2, 1, 5, 3, and 6.

A FIFO-buffer for the output results is placed parallel to the buffer of input data. Each cell of this buffer has an additional bit representing the sign (flag) for the existence (presence) of a result which has not yet been read the i.e. still stored in the buffer.

The initial state of the micropipeline buffer should be determined by force, in which all its fields are reset.

The proposed buffer structure serves the micropipeline using the following original strategy:

Recording in micropipelines buffer

1. At the moment, when the output of the pipeline delivers a result, the accompanying identification data, used as associative sign, performs an associative search in the buffer with input data. The figure shows the case with the outcome of the first ready task No. 4. The coincidence (RID No.6) - (Address 4) provides access to the same address

(same line) in the output buffer. In this sense, the buffer of the micro pipeline works as an associative memory;

2. The final result is recorded in the available buffer cell for outcomes. According the example, the result of task 4 is the first outgoing result. Also;
3. Recording of "1" in the affiliate "bit for presence" of the open cell is performed.

The process of recording the results going out from the pipeline continues according this algorithm until you get a result whose order number requires recording in the output (the bottom) cell of the FIFO-buffer. According to the logic of the FIFO-buffer containing the input data and managing the buffer with the results, the result just recorded belongs to a task whose order number exactly matches the input. Of all the tasks involved in the pipeline, the task whose outcome should be recorded in the output cell buffer results, we will call the earliest task. In other words, the result entered in this cell is subject to immediate reading. This situation can be identified easily by the "bit for presence" of the cell being set to "1". After reading the results of the exit cell of the FIFO-buffer, it is considered free, and then a general movement of data to the output is realized.

Reading of micropipeline buffer

Before we make clear reading operation, we must stress once again that as a result of the "write" operation, the results of individual tasks have already been arranged in the output buffer in full compliance with the initial order. Thus, their reading can be totally appropriate to the discipline FIFO.

Reading the results obtained from micropipeline, is an external device task, which here is not an object of attention. However, it should be noted that the reading is subject only to the output FIFO-buffer cell. The contents of this box is available for reading, only when it's bit for presence is recorded as "1". This "1" acts as a signal to resolve the reading operation, as illustrated in Figure 3.

After reading the results of the outgoing cell of the buffer, the entire line of fields can be considered free. According to the functioning FIFO, the content of the buffer is moved toward to the exit, which releases the incoming line, where the next record can be realized

Special situations in the algorithm

The algorithm shown above should be refined further. This is due to the different situations that can be created as a result of the input data and the parameters of the calculation process in pipelines. We mean both the parameters of the structure of individual unit pipelines and the delays, which also further depends on the specific data.

Tasks with same data

To begin we shall consider the situation where the successive task at the entrance of the pipeline has input data that match the input data of the tasks running before, still contained in the buffer. Here we should note that if a task has input data, which coincides with those of a previously started task, then its algorithmic path "*Begin-End*" will be the same. It follows that the later launched task can not anticipate the performance of the previously launched task. Moreover, its outcome will be the same. In the above-described scenario, the fact that input data coincides with those of other tasks will be established immediately, when the first of the running the "same data" tasks is coming down from the pipeline.

At this point (point No.1 in the upper sequence) the associative searching in the buffer for input data will establish a match with yet existing records. This situation is illustrated in

Figure 4 where we assume for example that tasks 2, 3 and 6 in the presumed sequence 4, 2, 1, 5, 3, 6, have the same input data.

FIGURE 4. CASE OF 3-FOLD ASSOCIATIVE MATCH

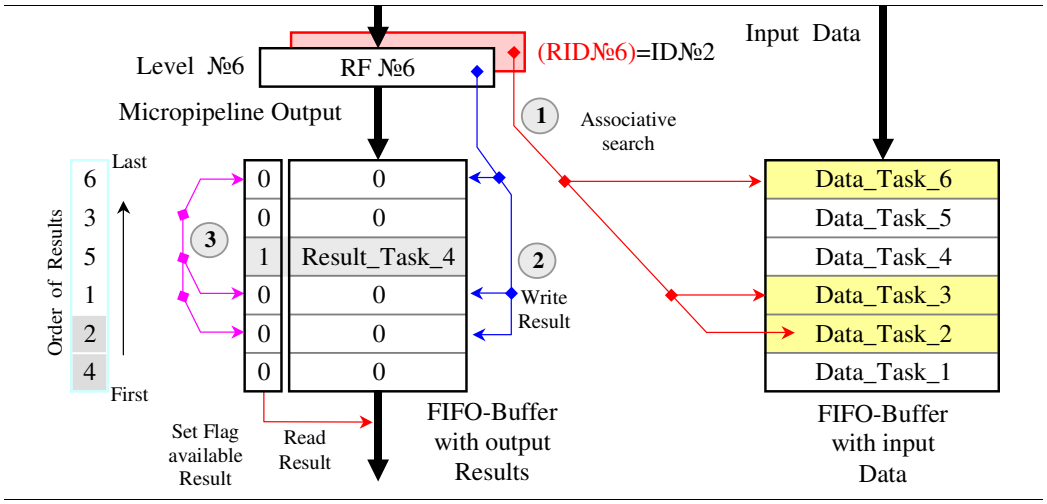


FIGURE 5. 3-FOLD DUPLICATION OF THE OUTCOME OF TASK NO. 2

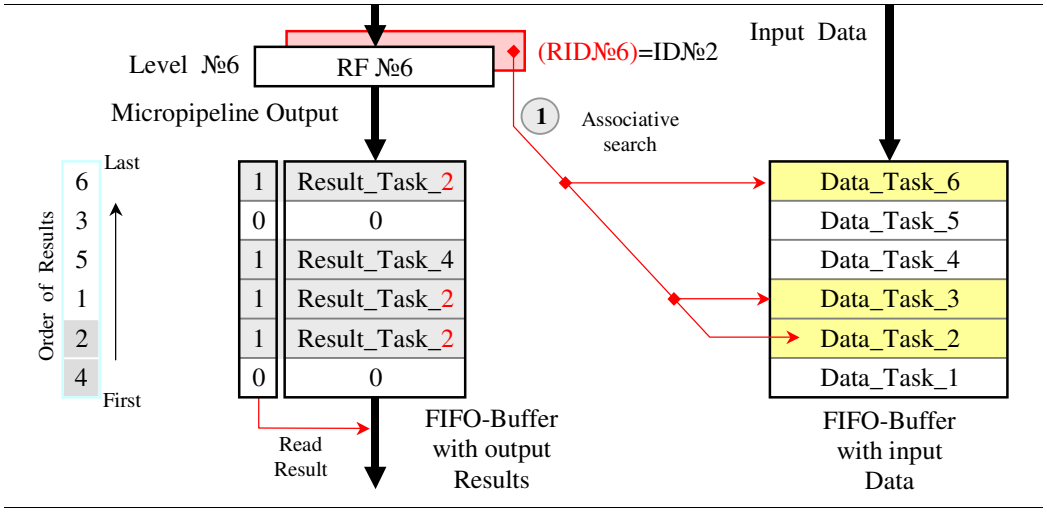


Figure 4 depicts the following: 6 tasks were started, but task number 4 is the first task that is completed and recorded its results. Upon completion of the next task 2 (as shown in the figure) an associative attribute match of the Task 3 and Task 6 (see yellow colored cells) is visible. So a question arises: in which cell to record the outcome of task 2? There are two possibilities: to write only in its own cell, as does task 4, or to enroll in all cells generating associative matching.

Since, under the same input data, all these tasks (2, 3 and 6) will get the same results, the second option for recording just gets executed ahead of the normal recording of the results from tasks 3 and 6. This means that calculations tasks 3 and 6 will take place (after already having completed tasks No.2) and will continue to run in pipelines, but its output

may not record their results since their respective cells will already have a "1" as a "sign of presence".

Therefore, the read out data will be in the correct order. In order not to complicate the management algorithm of the buffer, memory task 3 and task 6 may always record their results upon completion, without being influenced by the existing record. Assuming the first draft of behavior for recordings is performed with the result of task 2, the provisional content of micropipelined buffer will be as shown in Figure 5.

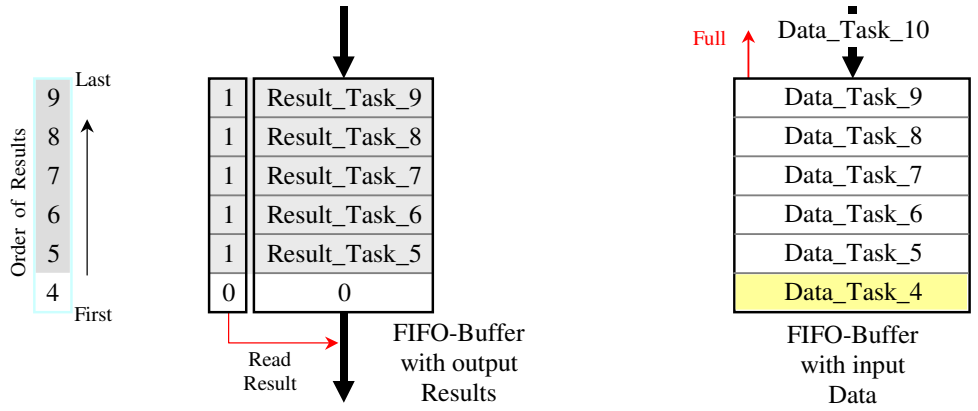
Occlusion of algorithmic paths

Secondly, we shall consider the situation where the next task can not be loaded in the pipeline because it cannot record in the buffer of its input data.

Recording could not be performed when the input buffer is full. We will examine a sample situation on the structure of Figure 1, provided that the volume of micropipelined buffer continues to be 6 cells.

Assume that initial 6 tasks are sequentially running in the pipeline. Thus, the input data of these tasks are filled the input buffer. Let assume further that the execution of tasks 1, 2 and 3 has undergone a branch "lie" of the algorithm (micropipeline units at levels 3, 4 and 5) and the execution of task 4 is deviated into branch "truth", where the calculations in micropipeline unit of this branch (level 3) have remained a long time. Under this condition, tasks 1, 2 and 3 will reach level 6, will successfully complete their work and will record their results. Let us assume that, while Task 4 continues to hold in the left branch of level 3, the following tasks 5 and 6 will take the path of the first three. If Task 4 continues to hold, tasks 5 and 6 will successfully complete and record their results. When this happens, the buffer of the results will have only one empty cell - this for task 4. As the results of Tasks 1, 2 and 3 will be marked with bits of presence, they will be freely read. As a result of these readings, in the output cell of the buffer for the results an empty cell will arrive - for the outcome of task 4.

FIGURE 6. IN EXPECTATION OF THE RESULT OF THE EARLIEST TASK



Thus, the readings will be terminated. Although the completed tasks are 5 (No. 1, 2, 3, 5 and 6), only the first three of them will be able to release their results. During the reading of the results of the first three tasks, the FIFO-buffer for input data will exempt input cells, which will be a prerequisite for loading in the pipeline three new tasks with serial numbers 7, 8 and 9. Assuming that the execution of these tasks also goes along the right

branch of the algorithm and their implementation ahead of task 4, they can successfully complete and record their results. The state of the buffer of the pipeline at this point can be illustrated with Figure 6.

The next task No. 10 could not be loaded into the pipeline, because its input buffer is full. At this point all micropipeline units are in a state of expectation, excepting the unit which continues the calculations associated with Task 4. In this situation, task 10 cannot be loaded because the input buffer is full. The situation in the pipeline is remarkably similar to a blockage of a blood vessel of the so-called thrombus.

When task 4 leaves level 3 and goes into level 6 it will finish and will record its result in the buffer of the pipeline. From that moment on, all the standard results for tasks 4 to 9 can be read. The conveyor will start task 10 and later, the next after it, when the buffer get free entry box.

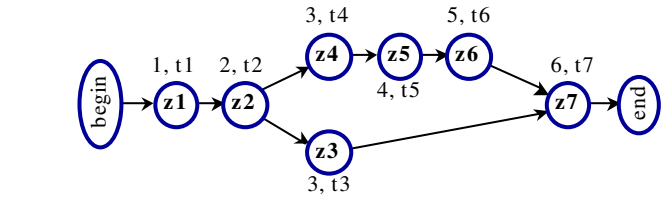
We must accept that such prolonged detention of calculations can occur in any multi-stroke unit of the micropipeline. Moreover, with such prolonged detention we can hit multiple units simultaneously, which means the presence of "thrombosis" in the corresponding algorithmic paths "*Begin-End*".

From this analysis it follows that in such situations the pipeline does not lose its working capacity, but significantly reduces its productivity. If a micropipeline unit often falls into a prolonged detention, it is clear that the place of this unit proves to be a "close". Certainly, in such cases the designer should make the necessary structural changes to the pipeline. It is clear that that the volume of the buffer of the pipeline is important for the productivity of the pipeline.

Buffer modeling and experimentation

Based on the commented pipeline and the buffer designed to restore the order of the results, a programming model was created through which the system was tested and experiments were produced. In the model, the pipeline was formally presented by a directed node-weighted graph G = (Z, U) - Figure 7.

FIGURE 7. THE GRAPH G=(Z,U), REPRESENTING THE PIPELINE



The interpretation of the graph, presented in Figure 7 is the following:

The set Z of the graph nodes, with power M=|Z|, presents the set of the micropipeline units (for this example M is 7);

The set U⊆Z×Z of the graph's edges models the set of the links between the micropipeline units;

Through the set N, a weight factor (integer number) that shows the number of the buffer's level, is associated with every node z_i, $z_i \in Z$, $i = 1, M$ of the graph. Since several micropipeline units may belong to the same level, the number of the nodes with the same weighting factors is ≥ 1;

Through the set T a set of weight factors is associated with every node of the graph G. Each weight factor t_j, $t_j \in T$, $j = 1, M$ shows the degree of delay that the corresponding node z_j micropipeline unit brings in the calculation process.

With this formalization, all possible routes for the movement of the tasks in the pipeline will be a decision of the task of finding all possible paths between the fictitious nodes "begin" and "end" in the graph G=(Z,U).

P1: *begin* - z1 - z2 - z3 - z7 - *end*;

P2: *begin* - z1 - z2 - z4 - z5 - z6 - z7 - *end*.

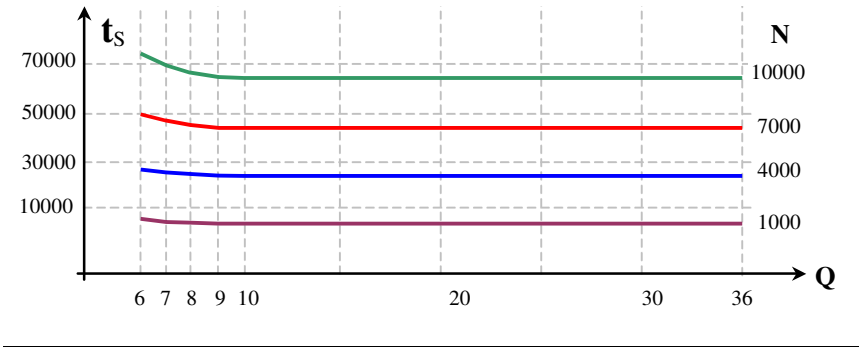
Taking into account the specifics of this case in which there is a variety of processes in pipelines and with the goal to increase the effectiveness of the program in terms of memory, the graph G is modeled by dynamic adjacency lists. Moreover, each node is described by a structure with fields: input (embedded structure), delay of the operation in the unit and links with adjacent nodes (pointers).

Both the input and output buffers are implemented as arrays of structures. In the input buffer the structure describes the input data and in output - the output data and their readiness for reading. The algorithms handling these structures implement the algorithm of work of the pipelines buffer described above.

Purpose of the programming model was experimentation of the joint operation of the pipeline and the buffer, experimentation of key services to the latter one, as well as experiments on the influence of its parameters on system performance as a whole. In the program, for each node a suitable operation was chosen. The input data for each running task were randomly generated. In each unit, the delay in calculating the results is recorded in "number of program bars". This allowed quantifying the productivity of the pipeline in a specified volume of buffer and a number of tasks loaded for execution.

The experiments had to answer the following main question: how the volume of pipeline buffer affects the productivity (performance) of the pipeline? For each experiment, the volume of buffer "Q" was examined in the interval [6, 36] cells and the number N of running tasks from 1000 to 10000. The graphs on Figure 8 illustrate the resulting execution time t_S for the specified number of tasks.

FIGURE 8. INFLUENCE OF THE VOLUME OF PIPELINE BUFFER ON THE PRODUCTIVITY



The conclusion, based on the analysis of the results is that when the volume of the pipeline buffer increases, the system productivity grows exponentially, reaching saturation after almost doubling the volume of buffer. This positive effect is amplified by the number of completed tasks in the pipelines.

Conclusion

The task to restore the order of the output results was formulated in (Tyanev, 2009). This article presents a solution to this problem. The solution complements the complex solution of the investigated problems with the design of micropipelines, realizing common algorithmic structures. Obtaining solutions to these problems will allow us to take algorithms in a microoperational level, which at present are software implemented. Their hardware implementation, combined with pipeline organization would significantly increase the productivity of the systems that implement them.

Based on the analysis of possible pipeline buffer conditions, we found the need to signal "Full", that must be functionally connected with pipeline automate of the initial (starter) unit of the pipeline. The pipeline automate of the starting point should be blocked if it reaches the alert Full=1. This means, the starting points automate of pipelines that will hold output buffer for their results must be designed differently. We find this result to significant because it is sufficiently common and consistent with the overall objective mentioned in the beginning.

References

Kolev, S., Tyanev, D., 2010. "Early set to zero micropipeline," International Conference on Computer Systems and Technologies - CompSysTech'10, Sofia, Bulgaria, pp.25-30. http://portal.acm.org/citation.cfm?id=1839379.1839385&coll=DL&dl=GUIDE&CFID=16649328&CFTOKEN=79928997

Tyanev, D., 2009. "Four-phase micro-pipeline with one-cycle and multi-cycle micro-pipeline sections," Computer Science and Technologies, Publication of Computing and Automation Faculty Technical University of Varna, Bulgaria, ISSN 1312-3335, VII 1/2009, pp.3-12.

Tyanev, D., Josiffov, V., Kolev, S., 2007. "Operational structures without controlling automata," International Workshop on Network and GRID Infrastructures, 27-28 Sept. 2007, Bulgarian Academy of Sciences, Sofia, Bulgaria.

Tyanev, D., Kolev, S., Josifov, V., 2007. "Method for realization of self-controlling loop apparatus structures," Proceedings of Technical University of Varna, 2007, ISSN 1311-896X, pp.130-35, http://www.tyanev.com/resources/docs/Document_V_42.pdf

Tyanev, D., Kolev, S., Yanev, D., 2007. "Method for realization of self-controlling loop apparatus structures - part 2," Computer Science and Technologies, Publication of Computing and Automation Faculty Technical University of Varna, Bulgaria, ISSN 1312-3335, V 2/2009, pp.23-30.

Tyanev, D., Kolev, S., Yanev, D., 2009. "Micro-pipeline section for condition-controlled loop," International Conference on Computer Systems and Technologies - CompSysTech'09, Ruse, Bulgaria, pp.I.4 (1-5), http://portal.acm.org/citation.cfm?id=1731740.1731752&coll=DL&dl=GUIDE&CFID=16649328&CFTOKEN=79928997.

Tyanev, D., Kolev, S., Yanev, D., 2010. "Race condition free asynchronous micro-pipeline units," International Conference on Computer Systems and Technologies - CompSysTech'10, Sofia, Bulgaria, pp.31-37. http://portal.acm.org/citation.cfm?id=1839379.1839386&coll=DL&dl=GUIDE&CFID=16649328&CFTOKEN=79928997

Tyanev, D., Popova, S., 2010. "Asynchronous micro-pipeline with multi-stage sections," ICEST'2010, Ohrid, Macedonia.

Tyanev, D., Yanev, D., Kolev, S., 2009. "Method for realization of self-controlling loop apparatus structures," Fifth International Scientific Conference Computer Science'2009, Sofia, Bulgaria, Proc.'2009, ISBN: 978-954-438-853-9, pp.154-158.

Hennessy, J., Patterson, D., 2003. Computer Architecture. A Quantitative Approach, 3rd Ed., Morgan Kaufman Publishers, ISBN 1-55860-596-7.

Patterson, D., Hennessy, J., 2005. Computer organization and design. The Hardware / Software Interface, 3rd Ed., Morgan Kaufman Publishers, ISBN 1-55860-604-1.