Critical processor download planning device in the multiprocessor systems

D B Borzov, R G Basov, V S Titov and Yu V Sokolova

Department of Computer Science, The Southwest State University, Kursk, Russia
E-mail: borzovdb@mail.ru

Abstract. The article deals with the critical multi-processor systems, which can include high-availability objects, and which performance is critical for any kind of activity. Such as a person, country, company, organization, etc. In this case, failure leads to a reduction in the system's response time and a further decrease in its performance, which means the availability factor. Thus, the usage of software solutions to this issue is unacceptable, and therefore it is necessary to use specialized hardware for load planning.

1. Problem definition
In modern critical multi-processor systems, there is a need for an operational response from the computer system. Critical systems are systems whose failures lead to losses (economic, physical, human, etc.). In case of failure, such systems are subject to high requirements for performance, reliability, security, safety, etc. At the same time, the expenses associated with making changes to the system or replacing it (direct, indirect, etc.) are more important than losses in the event of a direct or indirect failure of the multi-processor system itself. Obviously, the most important thing is to minimize the time and hardware costs required for a multi-processor system to respond to an emergency situation. Critical situations that occur in multi-processor systems include failures of internal processor modules, such as aircraft pilot cabins, surveillance, tracking, targeting systems, nuclear systems, etc. In the event of a failure of a multi-processor system, its performance decreases and its speed decreases, which cannot be allowed in critical systems. One of the solutions to this problem may be the processors load planning in multi-processor systems. In this case, it can avoid simultaneously loading multiple processors with a single task (program, sub-program, algorithm, file, etc.) and, with this, to plan the sequence of incoming tasks so that they are served simultaneously. This allows to reduce unplanned downtime and at the same time increase the availability factor along with improved performance. In this regard, the article proposes a method, algorithm, and device for processor load planning in critical multi-processor systems, which provides an increase in the performance of multi-processor systems and an increase in their availability factor.

2. Introduction
For a multiprocessor system [1, 2] consisting of P processors, calculations from \( X = \{x_1, x_2, \ldots, x_j, \ldots, x_n\} \) tasks specified by a certain algorithm are defined, reduced to a complexity vector \( W = \|w_i\| \), where \( N = n = |x_i|, i \in X \). In this case, tasks i come at a random time and their number at the entrance to the queue \( Q \leq Q \), where Q is the total amount of tasks (bits, bytes, kilobytes,
etc.) that are in the queue, and the tasks themselves can come at a random time. Tasks are sent to the processor at an instant of time $r_i \geq 0$ and are set in advance by values $w_i$.

For a multi-processor system, it is pre-determined that the processors are the same, i.e. they have the same clock speeds, performance and architecture, they are not interrupted for other tasks, that is, they do not have priorities and are equal [3]. A multi-processor system is defined by a set of $P_r$, which is represented as: $P_r = \{ P_1, P_2, \ldots, P_{\beta}, \ldots, P_{\lambda} \}$, where $P_{\beta}$ are the processors of a multi-processor system, and $(P_{\beta} = \prod X)$ [4].

Therefore, it needs to set a schedule for tasks that takes into account all the specified restrictions. That is, it is necessary for each task $x_i \in X$ to determine the start time $S_j$ such that $S_j \geq r_j$ so that $S_j + w_i \leq S_j$, where $S_i, S_j$ are the start times of other tasks, and $i, j \in X, i \neq j$. Analytically, the search for the processor load schedule can be described by mapping [5]:

$$\alpha_q = \{ x_{q_1}, x_{q_2}, \ldots, x_{q_i}, \ldots, x_{q_n} \} \rightarrow \{ p_{q_1}, p_{q_2}, \ldots, p_{q_i} \},$$  

(1)

where $q = 1, \ldots, n$, a symbol «$\rightarrow$» shows an operation of tasks planning of a set $X$ on the corresponding set of processors $P_r$.

Here $q$ is the number of the next assigning option corresponding to the $q$-assigning option. The cardinality of the set $\Psi = \alpha_q$ of possible mappings (1) is equal to the number of possible assignments of tasks $\{ x_{q_i} \}$ in the matrix $X:|\Psi| = N!$.

Based on the formalized statement of the problem, its corresponding mathematical statement looks as follows [1-2]:

$$\sum_{j=1}^{n} r_j \rightarrow \min$$  

(2)

Taking into account (1-2) and based on the theoretical assumptions presented above, a method for pipelining processor load in critical multi-processor systems is proposed in the study based on (1-2). It is proposed to assign program fragments independently to the processors of a multi-processor system without taking into account other operators. The proposed method consists of the following steps [6, 7]:

1. Getting a set of tasks $X = \{ x_1, x_2, \ldots, x_i, \ldots, x_n \}$ which require a processor load plan, a set of processors $P_r = \{ P_1, P_2, \ldots, P_{\beta}, \ldots, P_{\lambda} \}$ of a multi-processor system as well as writing as a matrix $PrX = [P_{\beta}, x_i]$.

2. Getting the minimum element in a row and column $Pr X_{\beta \alpha}$ and subtracting it from all other elements.

3. Search for a zero in a row $Pr X_{\beta \alpha}$ and delete the remaining zeros in the row.

4. Search for a zero in a column $Pr X_{\beta \alpha}$ and delete the remaining zeros in the column.

5. Repeat steps 3-4 for all elements $Pr X_{\beta \alpha}$.

6. Making the minimum number of horizontal and vertical intersections through the marked zeros.

7. Searching for the minimum among numbers that are not crossed out by straight lines and subtracting it from these numbers.

8. Adding the minimum number to the numbers at the intersection of straight lines.
9. Repeat steps 2-8 until zero is found in \( \text{Pr} X_{\beta \alpha} \).

10. If \( \text{Pr} X_{\beta \alpha} = 0 \), then a sub-program \( i \) is assigned on a processor \( j \).

11. Repeat steps 1-11 until the remaining subprogram assignments are found for the processors of the multi-processor system.

12. Getting the ready tag \( \text{Exec} \).

13. Analysis of availability bits of the field \( \text{Work} \) of the tag \( \text{Exec} \). If \( \text{Work} = 1 \), then read the number of the free processor and step 1, otherwise step 4.

The total duration of all operations is calculated using the formula:

\[
T = \sum_{j=1}^{n} r_j \rightarrow \min.
\]  

(3)

To solve the problem of processor load planning, an algorithm is proposed that reduces the planning time for parallel execution of subprograms on the processors of a multi-processor system, excluding control and data connections between them according to the criteria (1-3).

Based on the proposed method, an algorithm for processor load planning in multi-processor systems was developed, consisting of the following steps [6, 7].

1. Set the task matrix \( \text{Pr} X_{\beta \alpha} \), where \( \alpha = 1, P \) is the number of tasks of the multi-processor system, and \( \beta = 1, P \) is the number of processors.

2. Search in a row \( \min_j \text{Pr} X_{\beta \alpha} \) and subtract it from all its elements.

3. Determining in the column \( \min_j \text{Pr} X_{\beta \alpha} \) and calculating from all its elements.

4. Search for the minimum element in the row and fix it. If there are still zeros in the row, then eliminate them.

5. Perform step 4 for all rows \( \text{Pr} X_{\beta \alpha} \).

6. Search for a zero in the column and fix it. If there are still zeros in the column, then eliminate them.

7. Perform step 6 for all \( \text{Pr} X_{\beta \alpha} \).

8. If there are no zeros in the \( \text{Pr} X_{\beta \alpha} \), then select the minimum number of horizontal and vertical intersections through the marked zeros and perform step 9, otherwise steps 2-7.

9. Search for the minimum value among numbers that are not crossed out by straight lines and subtract it from these numbers. Adding the minimum number to the numbers at the intersection of straight lines.

10. Repeat steps 2-13 until one zero appears in \( \text{Pr} X_{\beta \alpha} \).

11. If \( \text{Pr} X_{\beta \alpha} = 0 \), then a sub-program \( i \) is assigned on a processor \( j \).

12. Repeat steps 1–11 until the remaining subprogram assignments are found for the processors of the multi-processor system.

In accordance with the proposed method, the algorithm sets the initial matrix \( \text{Pr} X_{\beta \alpha} \) at the first step, where \( \alpha \) are the rows that define the set of tasks for which it is necessary to find the processor load schedule, and the columns – the set of processors, respectively.

3. Results

In accordance with the presented mathematical problem definition, tasks for a multi-processor system consisting of \( P \) processors are reduced to a complexity vector \( \|w\|_n \). As follows from the
proposed method and algorithm for processor load planning, in fact, the processor uses two operations of addition and subtraction, performed according to the classical vector scheme. In turn, this leads to the possibility of using a vector processor, the block diagram of which is shown in Figure 1 [1, 8, 9].

![Figure 1. Block diagram of the vector processor.](image)

Processing of all n components of operand vectors is set by a single vector command (Figure 1). A common structure is when the ALU consists of separate addition and multiplication blocks. For example, sometimes a block for calculating the inverse of a division operation is implemented as $X \left( \frac{1}{Y} \right)$.

In [9], it is assumed that a typical computing device looks like the one shown in Figure 2.

![Figure 2. Block diagram of the computing device: RAM random access memory, Rg registers and logic circuits unit, ALU arithmetic logic unit (the ALU control device is not presented), CU control unit, MU bus capacity matching unit, AB address bus, DB data bus, SB status bus, CB control bus, ED external device.](image)
As follows from Figures 2, to implement the functions of a vector processor, it is necessary to determine the bit sizes of the main elements of the upper processor level, which include the address bus (AB), data bus (DB), RAM, control bus (CB) and elements of the lower processor level: the ALU operating automate and control devices (CD) of the ALU.

The basic data for vector processor design are used to determine a capacity of address bus, data bus and control bus, and the result is as follows:

– as the capacity of data to be transferred no more than eight, choose the data bus DB(7-0);
– as the queue awaiting execution is six and it is supposed to use the vector processor, so get the address bus AB(7-0).

As it is assumed to use four commands in a vector processor, their coding requires two binary bits, and hence the resulting is field capacity of operation code OC(1-0).

Obtain the command format shown in Figure 3.

| OC | CF | AT | R1 | R2 |
|----|----|----|----|----|
| 0  | 1  | 2  | 3  | 4  |
| 10 | 11 |
| 17 |
| 31 |

Figure 3. The command format code.

Based on the fields capacity of the command format, the capacity of control bus CB(31-0) is defined.

Based on the defined command format capacity the table of command format coding options is obtained (Table 1).

| Operation Code | Command Format | Addressing type code | Operation contain |
|----------------|----------------|----------------------|------------------|
| Addition 00    | RR             | 00                   | AH.AL := POH[R₁] + POH[R₂] |
| Subtraction 00 | RR             | 01                   | AH.AL := POH[R₁] + POH[R₂] |
| УП on > 0     | RR             | 10                   | if (SF = 0) and (ZF = 0) then PC := R₁ |
| УП on < 0     | RR             | 10                   | if SF = 1 then PC := R₂ |

The list of relevant micro-operations is shown in Table 2.

| Control device output signals | Micro-operation |
|------------------------------|-----------------|
| y₀                           | RgBH.RgBL := DOP(RgBH.RgBL) |
| y₁                           | AL := RgAL + RgBL |
| y₂                           | AH := RgAH + RgBH + c |
| y₃                           | AH.AL := DOP(AH.AL) |
| y₄                           | SF := AH(31) |
| y₅                           | ZF := (AH.AL = 0) |
| y₆                           | OF := AH(31) ⊕ AH(30) |
| y₇                           | PC(13:0) := CB(35:22) |
| x₁                           | OC = "TC>0" |
| x₂                           | OC = "TC<0" |
| x₃                           | OC = "TC=0" |
A functional diagram of the ALU operating automate is constructed. After developing a generalized ALU algorithm, coding micro-operations and logical conditions, the control signals \( \{y_i\} \) are arranged according to the assignment marking in the ALU algorithm (Table 2).

The functional diagram of a vector processor (Figure 3) is developed based on the designed operating automate and an arithmetic-logic device. The functional processor diagram is a combine of the operating automate of the lower processor level and ALU, its control device, bit bus capacity matching units.

4. Conclusion

In this paper, a device for processors load planning was proposed for critical multi-processor systems, focused on hardware implementation. The software simulation is performed that confirms the validity of the application of the developed vector device for processor load planning in critical multi-processor systems. The software simulation is conducted, it is found, there is a dependence of the tasks loading effect on the number of processors in multi-processor systems as well as the dependence of a choice of necessary and sufficient their quantity for processing certain data volume.

Acknowledgments

The authors are grateful to the organizers of the X International Scientific and Practical Conference "Information and Measuring Equipment and Technologies" for the opportunity to participate and publish research.

References

[1] Tsilker B I A 2007 Organization of computers and systems (Piter: SPb)
[2] Ore O 1968 Theory of graphs (Moscow: Nauka)
[3] Hua S, Yang Z 2013 Open Automation and Control Systems 5 38–44 doi: 10.2174/1874444301305010038
[4] Multicore architecture: problematic aspects 2007 Electronics NTB. # 1. Available at: https://www.electronics.ru/journal/article/514
[5] Mikushin A V, Sazhnev A M, Sedinin V I et al 2010 Digital devices and microprocessors (SPb: BKhv-Petersburg).
[6] Trakhtengerts Eh A 1091 Introduction to the theory of analysis and parallelization of computer programs in the process of translation (Moscow: Nauka) p 254
[7] Borzov D B, Chernetskaia I E 2020 Designing a computer processor (Kursk: Southwestern state university) p 199
[8] Basov R G, Borzov D B, Loktionova O G et al 2019 Program for loading simulation of a multi-processor system. Certificate of state registration of computer programs # 2019616927.
[9] Basov R G, Borzov D B, Titov V S et al 2020 Modern science: topical problems of theory and practice. Series: Natural and technical sciences 42–49 doi: 10.37882/2223-2966.2020.10.07