Critical pipeline of the acyclic wave processor

A A Khusainov

1 Komsomolsk-on-Amur State University, 27, Lenina prosp., Komsomolsk-on-Amur, 681013, Russia

E-mail: husainov51@yandex.ru

Abstract. This paper is devoted to studying the performance of an acyclic wave processor consisting of heterogeneous processor elements having various delays. Data channels have sufficiently large buffers. The main result is the proof of the conjecture that for an arbitrary amount of input data, an acyclic wave processor contains a pipeline whose operating time is equal to the data processing time using the entire wave processor. This pipeline is called critical. For partially testing the conjecture, a multi-threaded application simulating the work of a two-dimensional pipeline has been developed. Earlier, the author proposed an algorithm for calculating the input processing time, the complexity of which depends not only on the number of stages but also on the amount of data. The proven conjecture eliminates this drawback, it allows building algorithms for calculating data processing time, the complexity of which does not depend on the amount of data. In particular, for any class of acyclic wave processors that have some algorithm of polynomial complexity for listing maximum pipelines, it is easy to construct a polynomial algorithm for calculating the data processing time for an arbitrary volume.

1. Introduction

We adhere to the definitions and notation [1], except that we now call the acyclic wavefront array processor (AWAP) an acyclic wave processor (AWP). The array of processor elements under consideration is heterogeneous, and the data transfer channels have sufficiently large buffers. Let $W$ be AWP, and let $T_W(n)$ be the processing time of the data volume $n$. For calculating the time $T_W(n)$, a directed graph with weighted vertices was constructed in [1] whose maximum path weight is equal to the processing time of the given amount of data by $W$. Based on this, an algorithm was proposed for calculating $T_W(n)$. The complexity of this algorithm is $O(m^2n)$, where $m$ is the number of processor elements. The disadvantage of the algorithm is that its complexity depends on the amount of data.

Our work removes this flaw. In [1], a conjecture was put forward that every acyclic wave processor $W$ for an arbitrary amount of data $n$ contains a pipeline having a processing time of $n$ elements equal to $T_W(n)$. For $n = 1$, this is a known fact that the project execution time specified by the network is equal to the weight of the critical path. We prove this conjecture (Theorem 1). Based on this conjecture, it is possible to construct algorithms for calculating the data processing time for an acyclic wave processor, the complexity of which does not depend on the amount of data. In particular, for any class of acyclic wave processors that have some algorithm of polynomial complexity for enumerating the maximal pipelines, we obtain a polynomial algorithm for calculating the processing time for an arbitrary amount of data (Corollary 1).
An acyclic wave processor consists of a set $E$ of processor elements (stages) located at the vertices of some finite acyclic directed graphs. The edges of this graph correspond to data channels. We assume that the transmission time of the data element through the channel is 0, and the channel memory buffer is large enough so that the transmission speed does not slow down. For example, an asynchronous computing pipeline is an acyclic wave processor with a sufficient channel buffer size of 1. We also do not assume that data is transmitted using the handshake method [2], [3].

Acyclic wave processors have a wide variety of applications: for the development of signal processors [4] and coprocessors [5], in cloud computing [6], and production automation [7]. For study them, Petri nets [8], [9], [10] are used. The wave processor model used by us is a particular case of the model of a weighted marked graph [10].

2. Preliminaries
We adhere to the definitions and notation from [1]. Let $\mathbb{N} = \{0, 1, 2, 3, \ldots \}$ be the set of all non-negative integers. The Petri net is considered as a quadruple $(P, T, pre, post)$, where $P$ is a finite set of places, $T$ is a finite set of transitions, $pre: T \rightarrow \mathbb{N}^P$ and $post: T \rightarrow \mathbb{N}^P$ are functions such that for all $t \in T$ and $p \in P$, the number $pre(t)(p)$ is equal to the number of arcs $p \rightarrow t$, and $post(t)(p)$ is equal to the number of arcs $t \rightarrow p$. A marked partial graph is a Petri net $(P, T, pre, post)$ such that for all $p \in P$ the conditions $\sum_{t \in T} pre(t)(p) \leq 1$ and $\sum_{t \in T} post(t)(p) \leq 1$ are satisfied. A place that does not have incoming arcs is called initial and not having outgoing arcs is called final. A wave processor or wave partial graph $(W, \tau)$ is a marked partial graph $W = (P, T, pre, post)$ with delays $\tau: T \rightarrow \mathbb{N}$, each transition of which has at least one outgoing arc and at least one incoming arc. It is called acyclic if its Petri net does not have directed cycles. The input data volume for the wave processor is an integer $n \geq 0$, which determines the initial marking of its Petri net according to the formula $M_0(p) = n$ if $p$ is the initial place, and $M_0(p) = 0$ otherwise. Terminal marking is defined as $L(p) = n$ if $p$ is a final place, and $L(p) = 0$, otherwise. The directed path between the places of the acyclic wave processor is called the pipeline connecting these places. Figure 1 (a) shows the pipeline connecting $p$ and $q$. It is easy to see that any transition of the acyclic wave processor belongs to some pipeline connecting some initial place with some final one. Thus, an acyclic wave processor can be considered as a working process in the sense of [7]. If the transition is the first (resp. last) in any pipeline containing it connecting the places, then it is called initial (resp. final). Let $W = (C, E, pre, post)$ be an acyclic wave processor. Denote by $init(W)$ the set of all initial transition, $fin(W)$ the set of all final transitions, $Tran(W) = E$ the set of all transitions. Delays for $e \in Tran(W)$ are defined as the values $\tau(e)$ of some function $\tau: Tran(W) \rightarrow \mathbb{N}$. Note that if we remove the initial place having an outgoing arc connecting it to a non-initial transition, the processing time $T_p(n)$ is not change. Similarly, we can delete the final place having an incoming arc connecting it to a non-final place. Figure 1 (b) shows a wave processor with an input data volume n. Processing time is not change if you delete the place $p_5$ or the place $p_6$.

Figure 1. The pipeline connecting the places $p$ and $q$ (a). Petri net of acyclic wave processor. Transition $t_0$ is initial, and $t_1$ is final (b).

For an arbitrary $e \in E = Tran(W)$, consider the set of all transitions and places that allow paths to $e$ and contain $e$, as well as places $c \in C$ allowing arcs $e \rightarrow c$. Add to these transitions and places all
the arcs from $W$ connecting them. Denote by $\Lambda(e)$ the resulting acyclic wave processor. Define the relation $e' \prec e$ on $E$, which means the existence of a place $c \in C$ and arcs $e' \rightarrow c \rightarrow e$.

**Proposition 1.** Let $W = (E, C, pre, post)$ be an acyclic wave processor with delays $\tau: E \rightarrow \mathbb{N}$. Then $T_W(n) = \max_{e \in fin(W)} T_{\Lambda(e)}(n)$, and for all $e \in E$ and $n \geq 1$ we have the equalities $T_{\Lambda(e)}(n) = \max \left( T_{\Lambda(e)}(n-1), \max_{e' \prec e} T_{\Lambda(e')} \right) + \tau(e)$ where $T_{\Lambda(e)}(0) = 0$.

This proposition follows from [1, Proposition 1] since $T_{\Lambda(e)}(n) = T(e,n)$ where $T(e,n)$ denotes minimum time for which the transition $e$ fires $n$ times.

For an arbitrary pipeline $K = (C, E, pre, post)$ with delays $\tau(t)$, using Proposition 1 and induction along the length of the pipeline, we obtain a useful well-known formula:

$$T_K(n) = \sigma(K) + (n - 1)\mu(K),$$

where $\sigma(K) = \sum_{t \in E} \tau(t)$, $\mu(K) = \max_{t \in E} \tau(t)$.

3. Results

Below everywhere, the notation $K \subseteq W$ is mean that $K$ is the pipeline contained in the wave processor. The pipeline $K \subseteq W$ is called critical for a given amount of data $n$ if $T_K(n) = \max_{K' \subseteq W} T_{K'}(n)$. Denote by $W + e$ the wave processor with a new transition $e$. Its set of transitions is equal to $Tran(W) \cup \{ e \}$, and for each transition $e' \in fin(W)$, the place $c_{e'}$, and two arcs are added: $e' \rightarrow c_{e'}$, and $c_{e'} \rightarrow e$.

**Proposition 2.** Let $W$ be an acyclic wave processor. For any $e \in Tran(W)$ and $n \geq 1$, it is true

$$T_{\Lambda(e)}(n) = \max_{K \subseteq \Lambda(e)} T_K(n).$$

To prove this, we introduce the definition: the height of the acyclic wave processor $W$ is the maximum length of the pipelines contained in $W$. Let the set $fin(\Lambda(e) \setminus \{ e \})$ consist of $m$ elements $e_1, \ldots, e_m$. Denote by $U$ the wave acyclic processor $\Lambda(e_1) \cup \cdots \cup \Lambda(e_m)$, consisting of places, transitions and arcs of Petri nets $\Lambda(e_i)$, for all $1 \leq i \leq m$. If the height $\Lambda(e)$ is 1, then $\Lambda(e)$ has one transition, whence it follows that formula (2) is valid for all $n \geq 1$.

Let us prove formula (2) for $n = 1$. According to [1, Corollary 1], the time $T_W(1)$ is equal to the maximum path weight in the weighted graph $\Gamma(W)$ defined in [1]. By formula (1), for every pipeline $K$, the time $T_K(1)$ is equal to the weight of the maximum path in the directed graph $\Gamma(W)$. So $T_W(1) = \max_{K \subseteq W} T_K(n)$. Substituting $W = \Lambda(e)$, we obtain the proved formula for $n = 1$.

Now we prove formula (2) using induction on pairs $(n, h)$, where $n > 1$ is the amount of data and $h > 1$ is the height of the acyclic wave processor of the form $\Lambda(e)$. Let this formula be true for all $\Lambda(e')$ of height $k < h$. And let it be true for any amount of data $n' < n$ for the acyclic wave processor $\Lambda(e)$ of height $h$. We prove it for $\Lambda(e)$ of height $h$ and data volume $n$. According to Proposition 1, it is true $T_{\Lambda(e)}(n) = \max_{K \subseteq \Lambda(e)} T_K(n)$. Since, by the induction hypothesis, for every $i \in \{1, \ldots, m\}$ there exists a pipeline $K_i \subseteq \Lambda(e_i)$ such that $T_{\Lambda(e_i)}(n) = T_{K_i}(n)$, then $T_{\Lambda(e)}(n) = T_{K_i}(n)$ for some $i \in \{1, \ldots, m\}$. In addition, there exists a pipeline $K \subseteq U$ such that $T_{\Lambda(e)}(n-1) = T_{K+e}(n-1)$. Proposition 1 leads to the equality $T_{\Lambda(e)}(n) = \max_{K \subseteq \Lambda(e)} T_K(n), T_{K+e}(n-1)) + \tau(e)$. Two cases are possible.

In the first case, there exists $i \in \{1, \ldots, m\}$ such that $T_{K_i}(n) \geq T_{K+e}(n-1)$. Formula (1) leads us to the equality $T_{\Lambda(e)}(n) = T_{K_i}(n) + \tau(e) = \sigma(K_i) + (n - 1)\mu(K_i) + \tau(e)$. It is easy to see that this implies the inequality $\tau(e) \leq \mu(K_i)$ whence it follows that in the first case, the desired (critical) pipeline in $\Lambda(e)$ is equal to $K_i + e$.

In the second case, for all $i \in \{1, \ldots, m\}$, the inequality $T_{\Lambda(e)}(n) < T_{K+e}(n-1)$ holds. This leads to $T_{\Lambda(e)}(n) = T_{K+e}(n-1) + \tau(e)$. It follows from (1) the equality $T_{\Lambda(e)}(n) = \sigma(K + e) +$
$(n - 2)\mu(K + e) + \tau(e)$, with the help of which the inequality $\tau(e) \geq \mu(K)$ is proved. We obtain $\mu(K + e) = \tau(e)$, when $T_{A(e)}(n) = \sigma(K + e) + (n - 1)\mu(K + e)$. Therefore, $T_{A(e)}(n) = T_{K+e}(n)$. Thus, in any case, there is a pipeline contained in $\Lambda(e)$ such that $T_{A(e)}(n)$ is equal to the processing time of $n$ data elements using this pipeline.

**Theorem 1.** For any acyclic wave processor $W$ with delays and the amount of data $n \geq 0$ there is the equality $T_W(n) = \max_{K \subseteq W} T_K(n)$.

The proof follows from $T_W(n) = \max_{e \in f(m(W))} T_{A(e)}(n)$ (Proposition 1) and from Proposition 2.

This theorem allows constructing an algorithm for calculating $T_W(n)$, the complexity of which does not depend on $n$. For this purpose, it suffices to consider any algorithm for enumerating pipelines $K \subseteq W$. For any $n$, $T_W(n)$ is equal to the maximum of the numbers $T_K(n) = \sigma(K) + (n - 1)\mu(K)$. The complexity of calculating each of these numbers is no more than $O(m)$, where $m$ is the number of stages of the acyclic wave processor. It follows that if the complexity of the algorithm for enumerating the maximum pipelines of this processor is $O(f(m))$, then there is an algorithm for finding the number $T_W(n)$ of complexity $O(fm(m))$.

**Corollary 1.** For any class of acyclic wave processors having some algorithm of polynomial complexity for enumerating the maximum pipelines, there is a polynomial algorithm for calculating the processing time for an arbitrary amount of data.

Consider, for example, a two-dimensional pipeline. It consists of $m = pq$ stages $t_{ij}$, $1 \leq i \leq p$, $1 \leq j \leq q$. Figure 2 (a) shows the Petri net of a two-dimensional pipeline at $p = 2$, $q = 3$. It is easy to see that for a fixed $p$, the complexity of enumerating its maximum pipelines is $O(m^p)$, and therefore there is an algorithm for calculating the processing time, of complexity $O(m^{p+1})$.

4. **Applications**

We give an example of the application of Theorem 1 for calculating the network schedule of continuing education courses in programming for teachers. We set it using Table 1, shown below in the format from [11, Example 6.5-1]. For each discipline, no more than one pair per day is conducted. The method of constructing a Petri net of a network diagram is described in [1].

**Table 1.** Network schedule of continuing education courses.

| Discipline                | Previous disciplines | Delay (days) |
|---------------------------|----------------------|--------------|
| A: Computer Graphics      | -                    | 7            |
| B: Computational Mathematics | A                   | 7            |
| C: Algorithms and Data Structures | -                  | 8            |
| D: Object-oriented programming | A, C               | 3            |
| E: Parallel Programming   | A, D                 | 6            |

Figure 2 (b) shows the Petri net of the network diagram. There are three maximum pipelines in the constructed acyclic wave processor. Consider the words ABC, ADE, CDE, composed of the symbols of the stages that make up these pipelines. We get $T_{ABE}(n) = 20 + 7(n - 1)$, $T_{ADE}(n) = 16 + 7(n - 1)$, $T_{CDE}(n) = 17 + 8(n - 1)$. The maximum of these functions at $n$ takes the values $T_W(n) = 20 + 7(n - 1)$, for $n \leq 2$, and $T_W(n) = 17 + 8(n - 1)$, for $n > 2$. 


Figure 2. Petri net of the two-dimensional (2 × 3)-pipeline (a). Petri net for the planning of continuing education courses (b).

5. Computer model

To partially test the conjecture, the author developed a computer model of the wave processor using a multi-threaded application. Each stage of a multi-threaded application is implemented as a thread containing a cycle consisting of reading an element from each input channel, performing a stage operation, and writing the results to the output channels. The exchange of data between stages through channels is based on Dijkstra’s algorithm for solving the problem of producer and consumer. The operation is simulated by waiting for the time delay of the stage, set in milliseconds. This method allowed developing a program for calculating the performance of a two-dimensional pipeline.

The two-dimensional pipeline consists of \( p \times q \) stages \( t_{ij} \), \( 1 \leq i \leq p \), \( 1 \leq j \leq q \). Therefore, it can be set using the table of delays \( \tau(t_{ij}) \). The example shown in Table 2 defines a two-dimensional \((3 \times 7)\)-pipeline.

| \( k \) | \( \tau_{k1} \) | \( \tau_{k2} \) | \( \tau_{k3} \) | \( \tau_{k4} \) | \( \tau_{k5} \) | \( \tau_{k6} \) | \( \tau_{k7} \) |
|-------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 1     | 1              | 40             | 40             | 40             | 40             | 40             | 40             |
| 2     | 1              | 1              | 1              | 1              | 40             | 40             | 40             |
| 3     | 85             | 1              | 1              | 1              | 1              | 1              | 40             |

Figure 3 shows a graph of the processing time versus data volume for a two-dimensional pipeline obtained as a result of the experiment.

Figure 3. Graph of processing time versus data volume.

The graph is displayed in the form of black dots with coordinates \((X,Y)\), where \( X \) is the data volume and \( Y \) is equal to the processing time of the \( X \) volume data. For comparison with theoretical
results, we plotted the lines using the lines whose values are equal to the maximum values of the processing times of X elements for pipelines contained in a two-dimensional pipeline. Running the program for various parameters showed that the theoretical results (Theorem 1) are consistent with the experiment. The graph is displayed in the form of black dots with coordinates \((X, Y)\), where \(X\) is the data volume and \(Y\) is equal to the processing time of the \(X\) volume data. For comparison with theoretical results, we plotted the lines using the lines whose values are equal to the maximum values of the processing times of \(X\) elements for conveyors contained in a two-dimensional pipeline. Running the program for various parameters showed that the theoretical results (Theorem 1) are consistent with the experiment.

6. Conclusion
There are various problems associated with calculating the processing time of a given amount of data for a wave processor. Processing time depends on the data transfer time. It may depend on the size of the channel buffer. The results of works devoted to the performance of wave processors usually contain estimates for large \(n\) \([4], [8]\). An exception is the results of the article \([12]\) and subsequent works by its authors. They study homogeneous two-dimensional pipelines.

We studied the processing time for an acyclic wave processor for a given amount of data and various stage delays. We developed an application for calculating data processing time for a two-dimensional pipeline. The conjecture of a critical pipeline was tested. We proved the critical pipeline conjecture and established the existence of processing time calculation algorithms for an acyclic wave processor, the complexity of which does not depend on the amount of data. The results can be used both in the development of processors and multi-threaded applications and in the construction of network schedules for mass and serial production.

7. Acknowledgments
This work was performed as a part of the Strategic Development Program at the National Educational Institutions of the Higher Education, No. 2011-PR-054.

References
[1] Khusainov A A 2019 Calculation of data processing time for an acyclic wavefront array processor Journal of Physics: Conference Series 1333 022006
[2] Kung S-Y, Arun K S, Gal-Ezer R J, Rao D V B 1982 Wavefront array processor: language, architecture, and applications IEEE Trans Comput C-31 1054-1066
[3] Sano K, Nakahara H 2018 Hardware Algorithms Principles and Structures of FPGAs (Singapore: Springer) 137-177
[4] Kung S-Y 1988 VLSI array processors (Englewood Cliffs, NJ: Prentice Hall)
[5] Merchant F, Chattopadhyay A, Raha S, Nandy S K, Narayan R 2017 Accelerating BLAS and LAPACK via Efficient Floating Point Architecture Design Parallel Process. Lett. 27 03n04
[6] Tolosana-Calasanz R, Baares J A, Colom J M 2018 Future Generation Computer Systems 87 888-909
[7] Meyer PJ, Esparza J, Völzer H 2018 Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TAKAS 2018) (Cham: Springer) 3-19
[8] Ramamoorthy C V, Ho G S 1980 IEEE Transactions on Software Engineering 5 440-449
[9] Amparore EG, Donatelli S, Beccuti M, Garbi G, Miner A 2018 Transactions on Petri Nets and Other Models of Concurrency vol 13 (Berlin:Springer) 73-92
[10] Devillers R, Hujsa T 2018 International Conference on Applications and Theory of Petri Nets and Concurrency (Cham: Springer) 19-39
[11] Taha H A 2007 Operations Research: An Introduction (Upper Saddle River, NJ: Prentice Hall)
[12] Hoisie A, Lubeck O, Wasserman H 1999 Lecture Notes in Control and Information Sciences vol 249 (London: Springer) 171-187