Łukasz DRÓŻDŻ[1], Jerzy ROJ[1]

Politechnika Śląska, Katedra Metrologii, Elektroniki i Automatyki (1)

doi:10.15199/48.2018.11.06

Wpływ zastosowania obliczeń równoległych na czas wykonywania rozmytych algorytmów sterowania

Streszczenie. *W artykule poruszono zagadnienia związane z wykorzystaniem obliczeń równoległych przy implementacji algorytmów sterowania bazujących na zasadach logiki rozmytej. Przedstawiono realizację przykładowego regulatora, w którym wykorzystano standard specyfikacji przetwarzania współbieżnego OpenMP oraz QtConcurrent. Opisano przeprowadzone badania porównawcze standardowego algorytmu sekwencyjnego z algorytmem wykorzystującym obliczenia równoległe pod względem szybkości działania.*

Abstract*. In the paper issues related to the use of a parallel computation in the implementation of control algorithms based on the principles of fuzzy logic have been described. The implementation of an exemplary regulator, in which the OpenMP standard and QtConcurrent is used has been presented. Comparative studies of a standard sequential algorithm with an algorithm that uses a parallel computation in terms of speed of operation were described. (**Use of parallel computation influence on compute time of fuzzy control algorithms**).*

Słowa kluczowe: logika rozmyta, sterowanie rozmyte, obliczenia równoległe.
Keywords: fuzzy logic, fuzzy logic control, parallel computing.

Wstęp

Większość obecnie stosowanych mikroprocesorów stanowią konstrukcje umożliwiające przeprowadzanie wielu obliczeń w jednym czasie. Algorytmy sterowania bazujące na zasadach logiki rozmytej wymagają w wielu przypadkach przeprowadzenia skomplikowanych obliczeń w celu wyznaczenia wartości sygnału sterującego [1]. Obliczenia te związane są z wyznaczeniem stopni przynależności przesłanek wejściowych oraz przeprowadzeniem wnioskowania mającego na celu wyznaczanie stopni przynależności przesłanek wyjściowych [2]. Zarówno w przypadku wyznaczania stopni przynależności przesłanek wejściowych jak i wyjściowych, kolejne obliczenia nie są od siebie zależne, przez co mogą odbywać się równolegle. Implementacja algorytmu wykorzystującego wiele procesorów do przeprowadzania obliczeń jest często trudna [3]. Wykonywanie wielu operacji jednocześnie wymaga bowiem ich synchronizacji i wiąże się z potrzebą zastosowania odpowiednich mechanizmów, których użycie wymaga od programisty posiadania specjalistycznej wiedzy [4]. Obecnie istnieje jednak wiele bibliotek, takich jak np. „OpenMP", „TPL", „TBB", „Cilk" czy „QtConcurrent", pozwalających na zrównoleglanie algorytmów bez konieczności użycia tzw. niskopoziomowych prymitywów (ang. Low level thread primitives) do obsługi wątków.

Obliczenia równoległe znajdują szerokie zastosowanie przy przeprowadzaniu symulacji komputerowych, obróbce wideo, generowaniu obrazów, a nawet przy przetwarzaniu informacji znajdujących się w bazach danych [5]. Ich zastosowanie pozwala na skrócenie czasu potrzebnego do uzyskania wyników obliczeń bez konieczności stosowania większych częstotliwości sygnału zegarowego. Wymagany jest jednak system dysponujący więcej niż jednym procesorem fizycznym. Koncepcja ta znajduje także zastosowanie w procesie przetwarzania dużych ilości danych z wykorzystaniem logiki rozmytej [6].

Schemat działania regulatora rozmytego

Regulator wykorzystujący zasady logiki rozmytej (w skrócie - regulator rozmyty) można przedstawić jako układ posiadający *n* wejść (wielkości wejściowych) oraz *k* wyjść (wielkości wyjściowych). Z każdą wielkością wejściową powiązana jest dowolna liczba przesłanek wejściowych, których stopnie przynależności wyznaczane są na etapie rozmycia. Na podstawie wartości stopni przynależności przesłanek wejściowych, w procesie wnioskowania, wyznaczane są wartości stopni przynależności przesłanek wyjściowych. Obliczone wartości poddawane są wyostrzaniu w celu wyznaczenia wartości wielkości wyjściowych.

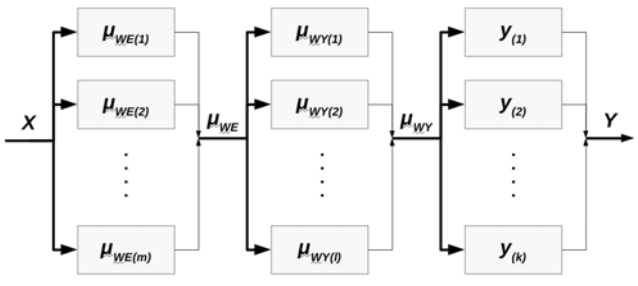
Liczba przesłanek wejściowych, zbiór reguł wnioskowania oraz liczba przesłanek wyjściowych zależą od decyzji projektanta algorytmu. Podobna sytuacja dotyczy doboru metod obliczania stopni przynależności przesłanek oraz sposobu wyostrzania wartości zmiennych rozmytych. W praktyce omawiany regulator posiada najczęściej jedno wyjście (sygnał sterujący) oraz kilka wejść (np. wartość uchybu regulacji, wartość wielkości wpływających na proces regulacji itd.) [7]. Uogólniony schemat regulatora przedstawiono na rysunku 1. Działanie regulatora rozmytego opisują równania:

$$\mu_{WE_i} = f(x_1, x_2, \ldots, x_n), \tag{1}$$

$$\mu_{WY_i} = f(\mu_{WE_1}, \mu_{WE_2}, \ldots, \mu_{WE_m}), \tag{2}$$

$$y_i = f(\mu_{WY_1}, \mu_{WY_2}, \ldots, \mu_{WY_l}), \tag{3}$$

w których: μ_{WE} jest przesłanką wejściowa, μ_{WY} – przesłanką wyjściową, x – wielkością wejściową, y – wielkością wyjściową, n – liczbą wielkości wejściowych, m – liczbą przesłanek wejściowych, l – liczbą przesłanek wyjściowych, k – liczbą wielkości wyjściowych, $\boldsymbol{X}$ – wektorem wielkości wejściowych, $\boldsymbol{Y}$ – wektorem wielkości wyjściowych.



Rys. 1. Uogólniony schemat rozmytego algorytmu sterowania

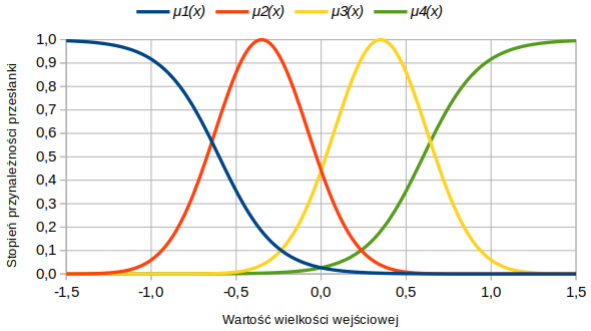
Na podstawie zależności 1-3 oraz rysunku 1 można zauważyć, że obliczenia związane z kolejnymi etapami działania algorytmu mogą być przeprowadzane niezależnie. Należy jednak zaznaczyć, że przed rozpoczęciem kolejnego etapu wszystkie wartości wielkości związanych z etapem poprzednim muszą zostać obliczone. Wynika to z faktu, że obliczane w każdym etapie wartości zależą jedynie

od wartości wielkości wyznaczonych w etapie poprzednim. Opisana właściwość umożliwia wykorzystanie obliczeń równoległych w celu zmniejszenia czasu potrzebnego do wypracowania wartości sygnału sterującego, a co za tym idzie – zastosowanie platformy wieloprocesorowej do efektywnego przeprowadzania obliczeń.

Opis przykładowego algorytmu regulacji

Obiekt badań stanowił przykładowy regulator rozmyty [8], którego ogólny schemat przedstawiono na rysunku 2. Parametrami wejściowymi regulatora są dwie wielkości – *A* oraz *B* – przyjmujące wartości z dziedziny liczb rzeczywistych z zakresu <-1,5; 1,5>. Wielkością wyjściową regulatora jest liczba rzeczywista *u* przyjmująca wartości z zakresu <-1; 1>. Z każdą wielkością wejściową związane są cztery przesłanki wejściowe, których stopnie przynależności opisują funkcje: sigmoidalna oraz Gaussa, przedstawione na rysunku 3.

Rys.2. Ogólny schemat badanego regulatora rozmytego



Rys.3. Funkcje stopni przynależności przesłanek wejściowych

Zestaw przesłanek wyjściowych stanowi sześć przesłanek [9]. Zależności stopni przynależności kolejnych przesłanek wyjściowych od stopni przynależności przesłanek wejściowych, nazywane zbiorem reguł wnioskowania, opisują równania:

(4) $$\mu_{U1} = (\mu_{A1} \cap \mu_{B1}) \cup (\mu_{A3} \cap \mu_{B4}) \cup (\mu_{A4} \cap \mu_{B3}),$$

(5) $$\mu_{U2} = (\mu_{A1} \cap \mu_{B4}) \cup (\mu_{A2} \cap \mu_{B2}) \cup (\mu_{A4} \cap \mu_{B2}),$$

(6) $$\mu_{U3} = (\mu_{A1} \cap \mu_{B2}) \cup (\mu_{A2} \cap \mu_{B3}) \cup (\mu_{A3} \cap \mu_{B1}),$$

(7) $$\mu_{U4} = (\mu_{A1} \cap \mu_{B3}) \cup (\mu_{A2} \cap \mu_{B1}) \cup (\mu_{A3} \cap \mu_{B2}),$$

(8) $$\mu_{U5} = (\mu_{A3} \cap \mu_{B3}) \cup (\mu_{A4} \cap \mu_{B4}),$$

(9) $$\mu_{U6} = (\mu_{A2} \cap \mu_{B4}) \cup (\mu_{A4} \cap \mu_{B1}).$$

Podczas wyostrzania zastosowana została metoda wysokości opisana zależnością

(10) $$u = \frac{\sum_{i=1}^{l} (\mu_i \cdot y_i)}{\sum_{i=1}^{l} (\mu_i)},$$

gdzie: μ_i – wyznaczony stopnień przynależności kolejnej przesłanki wyjściowej, y_i – wartość singletonu powiązanego z przesłanką wyjściową, l – liczba zdefiniowanych przesłanek wyjściowych.

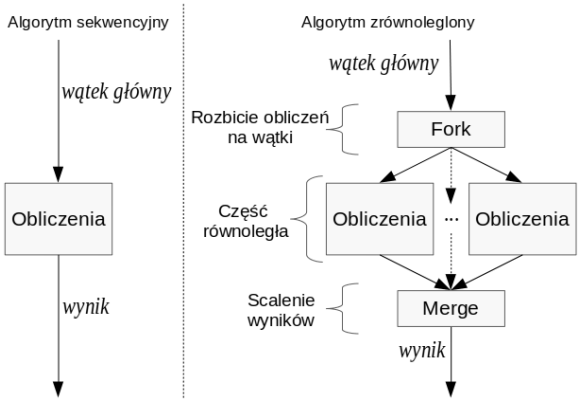
Wyznaczenie wartości wielkości wyjściowej wymaga zatem wyznaczenia wartości ośmiu przesłanek wejściowych, sześciu przesłanek wyjściowych oraz przeprowadzenia procesu wyostrzania. Kolejne wartości singletonów powiązanych z przesłankami wyjściowymi zestawiono w tabeli 1.

Tabela 1. Przyjęte wartości singletonów dla przesłanek wyjściowych

Parametr	Wartość
Sterowanie y_1	-1,00
Sterowanie y_2	-0,35
Sterowanie y_3	-0,10
Sterowanie y_4	0,20
Sterowanie y_5	0,45
Sterowanie y_6	1,00

Sekwencyjna oraz zrównoleglona implementacja algorytmu sterowania

Klasyczne, jednozadaniowe podejście, zakłada sekwencyjne obliczanie kolejnych wartości stopni przynależności przesłanek bez użycia współbieżnych części algorytmu [10]. Każda wartość jest obliczana w ściśle określonej kolejności, a po jej wyznaczeniu obliczane są kolejne. Takie podejście wyklucza wykorzystanie możliwości platformy wieloprocesorowej. Jest ono jednak najłatwiejsze w implementacji, ponieważ nie wymaga od projektanta potrzeby synchronizacji kolejnych części algorytmu. Porównanie działania przykładowego algorytmu zrównoleglonego z jego klasyczną, jednozadaniową wersją przedstawiono na rysunku 4.



Rys.4. Porównanie działania algorytmu sekwencyjnego ze zrównoleglonym

```
// Zmienne przechowujące obliczane wartości
double members = 0.0;
double value = 0.0;

// Wyznaczenie stopni przynależności przesłanek wejściowych
#pragma omp parallel for
for (int i = 0; i < sizeI; ++i)
{
     vIn[i] = inputMembers[i]->value(A, B);
}

// Wyznaczenie stopni przynależności przesłanek wyjściowych
// wraz z jednoczesnym wyostrzaniem otrzymanego wyniku
#pragma omp parallel for reduction(+: value, members)
for (int i = 0; i < sizeO; ++i)
{
     members += outputMembers[i](vIn);
     value += members * singletonValues[i];
}

// Zwrócenie wartości w postaci ostrej
return value / members;
```

Rys.5. Algorytm zrównoleglony przy użyciu standardu OpenMP

```
// Zmienne przechowujące obliczane wartości
double members = 0.0;
double value = 0.0;

// Zmienna pomocnicza
int i = 0;

// Utworzenie obiektów synchronizacji obliczeń
QFutureSynchronizer<FuzzyVariable> inSynchronizer;
QFutureSynchronizer<FuzzyVariable> outSynchronizer;

// Wyznaczenie stopni przynależności przesłanek wejściowych
for (const auto& f : inputMembers) inSynchronizer.addFuture(
    QtConcurrent::run(f, &MembershipFunction::value, A, B));

// Pobranie kolejnych wyników obliczeń
for (const auto& v : inSynchronizer.futures())
{
    vIn.append(v.result());
}

// Wyznaczenie stopni przynależności przesłanek wyjściowych
for (const auto& f : outputMembers) outSynchronizer.addFuture
    QtConcurrent::run(f, vIn));

// Wykonanie procesu wyostrzania wyniku obliczeń
for (const auto& res : outSynchronizer.futures())
{
    value += res.result() * singletonValues[i++];
    members += res.result();
}

// Zwrócenie wartości w postaci ostrej
return value / members;
```

Rys.6. Algorytm zrównoleglony przy użyciu biblioteki QtConcurrent

W celu zrównoleglenia obliczeń dokonano odpowiednich modyfikacji algorytmu sekwencyjnego. Implementacja oparta na wykorzystaniu biblioteki OpenMP [11], na poziomie kodu źródłowego, nie różni się znacząco od pierwotnej wersji algorytmu. Kod źródłowy został jedynie uzupełniony o dyrektywy preprocesora odpowiedzialne za rozbicie na wątki kolejnych kroków pętli w celu ich zrównoleglenia [12]. Implementacja wykorzystująca bibliotekę QtConcurrent wymagała znacznej modyfikacji sekwencyjnej wersji algorytmu w celu dopasowania kodu źródłowego do wymagań stawianych przez bibliotekę. Kod źródłowy omawianych algorytmów zamieszczono na rysunkach 5 oraz 6.

Analiza czasu realizacji obliczeń

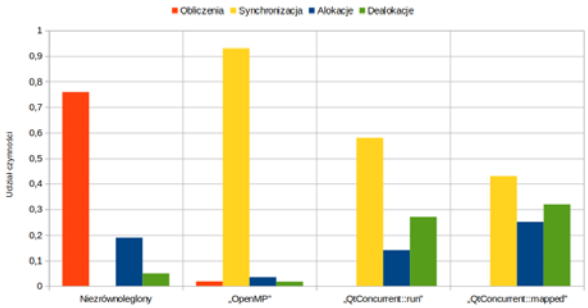
W celu przetestowania, w jakim stopniu zrównoleglenie oryginalnej wersji algorytmu sterowania wpłynęło na czas realizacji obliczeń wykonano iteracje, w których każda prezentowana wersja algorytmu obliczała kolejne 100000 wyników. Wartości zmiennych wejściowych zostały wylosowane przed rozpoczęciem iteracji i były jednakowe dla każdego z algorytmów. Podczas badania rejestrowano czas na początku i końcu iteracji, a następnie obliczano czas jej trwania. Użytą platformą testową był komputer klasy PC wyposażony w procesor o architekturze „x86-64" posiadający 6 rdzeni fizycznych taktowanych zegarem 4,4 GHz. Program testowy uruchomiono na systemie Debian 9 opartym o jądro Linux 4.13.0-1-amd64. Wyniki testów zestawiono w tabeli 2.

Tabela 2. Zestawienie wyników badań

Wersja algorytmu	Czas [ms]
Niezrównoleglony	54
OpenMP	911
QtConcurrent::run	9841
QtConcurrent::mapped	10943

Analizując parametry platformy testowej oraz właściwości przedstawionego algorytmu można zauważyć, że równolegle może być obliczanych sześć różnych wartości. W przypadku zastosowania 8 procesorów istniałaby możliwość przeprowadzania od 6 (etap wnioskowania) do 8 (etap fuzyfikacji) obliczeń równolegle. Zgodnie z prawem Amdahla [13] można zatem spodziewać się około sześciokrotnego skrócenia czasu potrzebnego na wypracowanie sygnału sterującego.

Na podstawie wyników badań można zauważyć, że początkowe założenia o wzroście prędkości działania algorytmu po jego zrównolegleniu okazały się zupełnie błędne. Zrównoleglony algorytm okazał się znacznie wolniejszy od jego synchronicznego odpowiednika. W celu określenia przyczyny tej sytuacji wykonano analizę działania algorytmów za pomocą analizatora wywołań „Callgrind" [14]. Na rysunku 7 przedstawiono, w skali względnej, udziały najważniejszych elementów algorytmu w czasie jego wykonywania.



Rys.7. Wyniki analizy działania badanych algorytmów

Wyniki analizy wskazują na to, że w przypadku algorytmu zrównoleglonego jego najistotniejsza część, jaką jest wykonywanie obliczeń, stanowi najmniejszy udział w jego czasie trwania. Oznacza to, że same obliczenia trwają zdecydowanie krócej niż pozostałe czynności. Wyjaśnia to tym samym dlaczego badany algorytm równoległy oblicza wynik zdecydowanie dłużej niż ich jego synchroniczny odpowiednik. Uzyskane wyniki prowadzą do konkluzji, że w rozpatrywanym przypadku zrównoleglenie algorytmu przyniosło same negatywne efekty – poza wydłużonym czasem pracy uruchomienie algorytmu wymaga również większej ilości pamięci. Zaistniały problem zilustrowano na rysunku 8.

Rys. .8. Przedstawienie przebiegu zrównoleglonej wersji algorytmu

Problem przedstawiono na przykładzie, w którym posłużono się platformą 3-procesorową. Wątek W_1 odpowiedzialny jest jedynie za procesy synchronizacji obliczeń, a wątki W_2 – W_4 za ich wykonywanie. Można zauważyć, że proces uruchamiania obliczeń (tworzenie zadań, alokacja pamięci) zajmuje więcej czasu, niż trwają uruchomione przez niego obliczenia (wyznaczanie wartości wielkości opisanych w równaniach 1-3). Po wykonaniu obliczeń każdy wątek przechodzi do fazy oczekiwania, przez co powiązany z nim procesor nie wykonuje w tym czasie żadnych zadań. Taka sytuacja jest bardzo niekorzystna z uwagi na niepełne wykorzystanie zasobów

obliczeniowych oczekujących na zadania procesorów oraz ze względu na potrzebę uruchamiania kolejnych zadań.

Prawidłowy przebieg zrównoleglonego algorytmu przedstawiono na rysunku 9. Można zauważyć, że proces uruchamiania obliczeń trwa znacznie krócej, niż same obliczenia. Po uruchomieniu wszystkich obliczeń każdy procesor realizuje powierzone mu zadanie, a gdy wszystkie wyniki zostaną obliczone następuje ich scalenie (zwolnienie pamięci, podsumowanie wyników cząstkowych i wyznaczenie ostatecznego wyniku obliczeń), które również przebiega znacznie szybciej, niż obliczenia. Taka sytuacja pozwala niemal w 100% wykorzystać zasoby obliczeniowe wszystkich procesorów z uwagi na fakt, że przez bardzo niewielki czas procesory oczekują na działania konieczne do uruchomienia obliczeń.

Rys.9. Prawidłowy przebieg algorytmu współbieżnego

Należy zauważyć, że sytuacja przedstawiona na rysunku 8, wraz ze wzrostem ilości zadań i procesorów, przynosi jeszcze bardziej negatywne efekty. Wraz ze wzrostem liczby zadań rośnie również czas potrzebny na ich uruchomienie, przez co rzeczywisty czas obliczeń pojedynczego procesora w odniesieniu do całkowitego czasu działania algorytmu drastycznie maleje.

Uwagi końcowe i wnioski

Problem zastosowania obliczeń równoległych przy implementacji rozmytych algorytmów sterowania okazuje się zagadnieniem skomplikowanym. Z uwagi na niewielką złożoność obliczeniową funkcji opisujących stopnie przynależności przesłanek, procesy związane ze zrównolegleniem obliczeń okazują się pochłaniać więcej zasobów, niż same obliczenia. Użycie gotowych rozwiązań, sprawdzających się w ogólnych zastosowaniach, jest zatem nieskuteczne i nie może zostać zastosowane podczas implementacji algorytmów sterowania bazujących na zasadach logiki rozmytej.

Bardziej szczegółowa analiza przedstawionego problemu wymagałaby na przykład implementacji algorytmu z wykorzystaniem biblioteki pthreads. Inne rozwiązanie stanowić może również wykorzystanie dedykowanego urządzenia złożonego z kilku mikrokontrolerów lub zastosowanie układu FPGA [15]. Możliwe jest także zaprojektowanie analogowego regulatora rozmytego [16]. Są to jednak rozwiązania kosztowne w realizacji i skomplikowane.

Wraz ze wzrostem skomplikowania narzędzi umożliwiających zrównoleglanie obliczeń wzrasta również złożoność algorytmów odpowiedzialnych za synchronizacje i uruchamianie zadań oraz zapotrzebowanie na pamięć operacyjną. Można zauważyć, że algorytm zrównoleglony przy użyciu biblioteki QtConcurrent wykonywał się około 10 razy dłużej od algorytmu zrównoleglonego za pomocą standardu OpenMP. Wynika to z faktu, że narzędzia udostępniane przez bibliotekę QtConcurrent umożliwiają wiele dodatkowych operacji, niedostępnych w standardzie OpenMP, takich jak możliwość śledzenia postępu obliczeń, dynamicznego wstrzymywania i wznawiana obliczeń, czy też zarządzanie kolejnością przeprowadzania obliczeń. Narzędzia te są bardzo pomocne podczas przetwarzania dużej ilości danych, w szczególności gdy prowadzone obliczenia są czasochłonne i istnieje konieczność np. informowania użytkownika o przewidywanym czasie zakończenia obliczeń.

Podczas projektowania algorytmu warto wziąć pod uwagę ewentualne potrzeby i możliwości opracowania jego wersji zrównoleglonej. Część dostępnych narzędzi stawia konkretne wymagania odnośnie sposobu implementacji kodu źródłowego, przez co ewentualne zrównoleglenie obliczeń może wymagać gruntownych zmian w sposobie zapisu algorytmu.

***Autorzy**: dr hab. inż. Jerzy Roj, Politechnika Śląska, Katedra Metrologii, Elektroniki i Automatyki, ul. Akademicka 10, 44-100 Gliwice, E-mail: jerzy.roj@polsl.pl; mgr inż. Łukasz Dróżdż, Politechnika Śląska, Katedra Metrologii, Elektroniki i Automatyki, ul. Akademicka 10, 44-100 Gliwice, E-mail: lukasz.drozdz@polsl.pl.*

LITERATURA

[1] Kuantama E., Vesselenyi T., Dzitac S., Tarca R., PID and Fuzzy-PID Control Model for Quadcopter Attitude with Disturbance Parameter, *International Journal Of Computers Communications & Control*, 4 (2017), nr.12, 519-523

[2] Yager R., Podstawy modelowania i sterowania rozmytego, *Wydawnictwo Naukowo-Techniczne*, Warszawa 1995

[3] Stefanowicz Ł., Wiśniewski R., Wiśniewska M.: Akceleracja obliczeń komputerowych za pomocą układów graficznych z wykorzystaniem technologii CUDA, *Pomiary Automatyka Kontrola*, 57 (2011), nr.8, 954-956

[4] Butenhof D., Programming with POSIX threads, *Addison-Wesley*, Boston 1997

[5] Byczkowska-Lipińska L., Cegielski M., Równoległe obliczenia w modelowaniu stanów nieustalonych w obwodach elektrycznych przy pomocy systemów klastrowych, *Przegląd Elektrotechniczny*, 2 (2007), 110-112

[6] Garduño E., Herman G., Parallel Fuzzy Segmentation of Multiple Objects, Int J Imaging Syst Technol, 18 (2008), 336-344.

[7] Pietrusiewicz K., Dworak P., Rozmyte dostrajanie regulatora prędkości serwonapędu DC, *Przegląd Elektrotechniczny*, 2 (2009), 112-114

[8] Piegat A., Modelowanie i sterowanie rozmyte. *Akademicka Oficyna Wydawnicza EXIT*, Warszawa 1999

[9] Wiktorowicz K., Zajdel R., O doborze reguł sterowania dla regulatora rozmytego, *Pomiary Automatyka Kontrola*, 51 (2005), nr.1, 44-46

[10] Korniak J., Rojek R., Sterowanie rozmyte modelem suwnicy przemysłowej, *Pomiary Automatyka Kontrola*, 57 (2011), nr.3, 235-237

[11] Chapman B., Jost G., Van Der Pas R., Using OpenMP: portable shared memory parallel programming, *The MIT Press*, London 2008

[12] Olas T., Wprowadzenie do zrównoleglania aplikacji z wykorzystaniem standardu OpenMP

[13] Tuura L., Innocente V., Eulisse G., Analysing CMS software performance using IgProf, OProfile and callgrind, *IOP Publishing Ltd*, 119 (2008), nr.4, 1-8

[14] Skinderowicz R., Programowanie współbieżne, wstęp do obliczeń równoległych, http://skinderowicz.pl/ (dostęp lipiec 2018)

[15] Asanovic K., Bodik R., Catanzaro B.C., Gebis J.J., The landscape of parallel computing research: A view from Berkeley, California 2006

[16] Guo S., Peters L., Surmann H., Design and Application of an Analog Fuzzy Logic Controller, *IEEE Transactions on Fuzzy Systems,* 4 (1996), nr.4, 429-438