

Struktura algorytmiczna układu do mnożenia długich operandów

Marek Gliszczyński, Alexandr Tariov

Streszczenie: W pracy została zaprezentowana zrationalizowana struktura algorytmiczna do obliczania iloczynu dwóch operandów N -elementowych ze zredukowaną liczbą układów mnożących i sumatorów dla dowolnej dużej wartości N . Pozwala to przy implementacji zmniejszyć nakłady obliczeniowe lub zapotrzebowanie na zasoby sprzętowe oraz stworzyć dogodne warunki do efektywnej realizacji operacji mnożenia dużych liczb w dowolnym sprzętowo-programowym środowisku implementacyjnym.

Słowa kluczowe: algorytm, mnożenie, długie liczby, operacje macierzowe

1. WPROWADZENIE

Operacja mnożenia długich operandów jest podstawą rozmaitych algorytmów kryptograficznych [1].

Niekiedy właśnie w takich zastosowaniach zachodzi konieczność przeprowadzenia operacji mnożenia na liczbach długości kilkaset i więcej bitów, które fizycznie nie mogą być obsługane przez współczesne układy mnożące bez wstępnego podziału. Ponadto bazowa operacja mnożenia dwóch liczb długości N bitów wymaga N^2 cząstkowych, jednobitowych operacji mnożenia, co dla dużych N powoduje kwadratowy wzrost operacji arytmetycznych. Stąd wynika potrzeba poszukiwania możliwości redukcji złożoności obliczeniowej operacji mnożenia długich liczb całkowitych. Dla ilustracji istnienia takiej możliwości rozważmy najpierw przykład mnożenia liczb dwucyfrowych.

Jeśli dwucyfrowe liczby x i y zdefiniujemy jak w równaniach (1):

$$x = x_1b + x_0, \quad y = y_1b + y_0, \quad (1)$$

gdzie b jest podstawą liczb x i y , to operacja mnożenia tych sprowadza się do następującego wzoru:

$$xy = x_1y_1b^2 + (x_1y_0 + x_0y_1)b + x_0y_0. \quad (2)$$

i, jak widać, wymaga wykonania 4 operacji mnożenia oraz 3 operacji dodawania odpowiednich cyfr.

Powstało wiele algorytmów spełniających te kryteria. Ze względu na prostotę i intuicyjność najpopularniejszym rozwiązaniem jest algorytm Karatsuby [6], według którego operacja mnożenia przedstawionych liczb x , y wygląda następująco:

$$xy = x_1y_1b^2 + ((x_0 + x_1)(y_0 + y_1) - x_0y_0 - x_1y_1)b + x_0y_0. \quad (3)$$

Sposób wykonania operacji mnożenia za pomocą wzoru (3) zapewnia zysk jednego mnożenia kosztem większej ilości dodawań i przekształceń w stosunku do „naiwnego” sposobu realizacji owej operacji za pomocą wzoru (2). Procedura realizacji mnożenia w algorytmie Karatsuby zakłada rekurencyjny podział liczby na dwie części aż do momentu operowania na pojedynczych cyfrach (bitach). Sposób ten zapewnia złożoność $O(n^{1,585})$ w stosunku do $O(n^2)$ metody naiwnej, ale ze względu na koszty wywołań rekurencyjnych nie stosuje się tego algorytmu dla liczb mniejszych niż kilkaset bitów.

Większy zysk można uzyskać wykorzystując jedną z modyfikacji algorytmu Karatsuby, np. metodę Toom-Cooka [2, 4, 10], która zakłada podział na trzy lub więcej części, jednak mniejszą liczbę mnożeń okupuje znacznie większą komplikacją, co z kolei uniemożliwia stosowanie tego algorytmu dla mniejszych liczb. Wersja algorytmu z podziałem na $k=3$ części (Toom-3) daje złożoność $O(n^{1,465})$.

W przeciągu ostatnich lat za najlepszy pod względem złożoności obliczeniowej ze wszystkich znanych dotąd algorytmów był uznawany algorytm Schönhage-Strassena, stosowany jedynie do mnożenia bardzo długich liczb składających się z dziesiątków tysięcy bitów. Ten algorytm wykorzystuje do realizacji mnożenia szybką transformatę Fouriera [9].

Jeszcze w początkach pierwszych pięciu lat 21 stulecia wydawało się, że w zakresie metod mnożenia długich liczb całkowitych, większych nowości jakościowych nie należy się spodziewać. Niespodziewanie taką zmianą jakościową stało się wylansowanie w 2007 roku nowego algorytmu pozwalającego na dalszą redukcję złożoności obliczeniowej tego typu operacji [6].

Prezentowana w artykule struktura układu do mnożenia długich operandów opiera się jednak na podstawowej metodzie Karatsuby z podziałem liczb na dwie części ze względu na prostotę tego rozwiązania. Głównym celem była taka modyfikacja, która pozwoliłaby na przeprowadzenie cząstkowych operacji mnożenia równolegle. Opisywana metoda reprezentowana jest za pomocą przekształceń macierzowo-wektorowych [12] i wprowadza podział działań na przekształcenia przed mnożeniami, operacje mnożenia oraz działania po

mnożeniach. Ponadto proponowane podejście likwiduje prawie zupełnie konieczność stosowania rekurencji, co umożliwia predefiniowanie wszystkich macierzy przekształceń.

Z uwagi na konieczność podziału liczb na dwie części rekurencyjnie zakładamy długość liczb $N = 2^m$, chociaż po uprzednim uzupełnieniu liczb innej długości po lewej stronie zerami można stosować algorytm dla dowolnej długości liczb.

2. SYNTEZA STRUKTURY ALGORYTMU MNOŻENIA DŁUGICH LICZB

Określimy najpierw znaczenie wykorzystanych w dalszej części artykułu symboli:

$\mathbf{I}_N^{(\leftarrow \alpha)}$ – macierz jednostkowa o wymiarze określonym za pomocą dolnego indeksu, natomiast indeks górny, (jeżeli jest) wskazuje liczbę pozycji cyklicznego przesunięcia wierszy macierzy w kierunku wskaźnika [5];

$\begin{bmatrix} \blacksquare \\ \blacksquare \end{bmatrix}$, $\blacksquare \blacksquare$ – symbole pionowej i poziomej konkatencji dwóch lub więcej macierzy [11];

$\otimes$ – symbol iloczynu Kroneckera [3];

W celu syntezy struktury algorytmicznej specjalizowanej jednostki procesorowej realizującej operację mnożenia długich operandów wprowadźmy kilka konstrukcji macierzowych:

- macierze definiujące przekształcenia na iteracjach poprzedzających mnożenie:

$$\mathbf{A}_{2^{m-k} \cdot 3^k \times 2^{m-k+1} \cdot 3^{k-1}}^{(k)} = \mathbf{I}_{3^{k-1}} \otimes \mathbf{T}_{3 \times 2} \otimes \mathbf{I}_{2^{m-k}}; \quad (4)$$

- macierze definiujące przekształcenia na iteracjach następujących po mnożeniu:

$$\mathbf{B}_{3^m}^{(k)} = \mathbf{I}_{3^{k-1}} \otimes \mathbf{T}_3 \otimes \mathbf{I}_{3^{m-k}}, \quad (5)$$

$$\text{gdzie } k = \overline{1, m}, \quad \mathbf{T}_3 = \begin{bmatrix} 1 & 0 & 0 \\ -1 & -1 & 1 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{T}_{3 \times 2} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 1 \end{bmatrix};$$

- macierz sumowania wyjść o tych samych współczynnikach (bazach liczb):

$$\begin{cases} \mathbf{P}_1^{(0)} = 1 \\ \mathbf{P}_{(2^{m+1}-1) \times 3^m}^{(m)} = \begin{bmatrix} \blacksquare \\ \blacksquare \end{bmatrix} (\mathbf{P}_{(2^m-1) \times 3^{m-1}}^{(m-1)} \begin{bmatrix} \blacksquare \\ \blacksquare \end{bmatrix} \mathbf{O}_{2^m \times 3^{m-1}}) (\downarrow (2i)^{m-1}) \end{cases} \quad (6)$$

- opcjonalną macierz współczynników wagowych:

$$\mathbf{W}_{2N-1} = \text{diag}(b^{2N-2}, b^{2N-1}, \dots, b^0), \quad (7)$$

gdzie b jest podstawą mnożonych liczb.

Jeśli mnożone liczby $\mathbf{X}$ i $\mathbf{Y}$ zdefiniujemy, jako wektory, których elementami są kolejne cyfry tych liczb, to wynik mnożenia $\mathbf{Z}$ można zdefiniować następująco:

$$\mathbf{Z}_{(2N-1) \times 1} = \mathbf{W}_{2N-1} \mathbf{P}_{(2^{m+1}-1) \times 3^m}^{(m)} \mathbf{B}_{3^m}^{(1)} \mathbf{B}_{3^m}^{(2)} \dots \mathbf{B}_{3^m}^{(m)} \times$$

$$\times \mathbf{S}_{3^m} \mathbf{A}_{3^m \times 2 \cdot 3^{m-1}}^{(m)} \mathbf{A}_{2 \cdot 3^{m-1} \times 2^2 \cdot 3^{m-2}}^{(m-1)} \dots \mathbf{A}_{2^{m-1} \cdot 3 \times 2^m}^{(1)} \mathbf{X}_{N \times 1} \quad (8)$$

Macierz $\mathbf{S}_{3^m}$ jest diagonalną macierzą mnożeń:

$$\mathbf{S}_{3^m} = \text{diag}(\mathbf{A}_{3^m \times 2 \cdot 3^{m-1}}^{(m)} \mathbf{A}_{2 \cdot 3^{m-1} \times 2^2 \cdot 3^{m-2}}^{(m-1)} \dots \mathbf{A}_{2^{m-1} \cdot 3 \times 2^m}^{(1)} \mathbf{Y}_{N \times 1})$$

Macierz tą uzyskuje się identycznie do przekształceń wektora $\mathbf{X}_{N \times 1}$, jak to jest pokazane we wzorze (8) – wykorzystane te same macierze (4). Operacje na obu wektorach mogą być wykonywane równocześnie,

a odpowiadające sobie wyniki częściowe mnożone symultanicznie, lub w kolejności pojawiania się (jeśli oba operandy są zmiennymi, nie trzeba konieczne tworzyć macierzy, co jest pokazane na rysunku 3).

Jedynie wyznaczenie macierzy $\mathbf{P}$ wymaga zastosowania rekurencji, jednak jest to szybka operacja, a postać macierzy zależy jedynie od długości liczb i można wyprowadzić potrzebne macierze jeszcze przed właściwym mnożeniem. Głównym zadaniem tej macierzy jest redukcja długości wektora wynikowego i uproszczenie konstrukcji macierzy $\mathbf{W}$. Przykład macierzy $\mathbf{P}$ dla przypadku $N = 4$ przedstawiono poniżej.

$$\mathbf{P}_{7 \times 9}^{(2)} = \begin{bmatrix} 1 & & & & & & & & \\ & 1 & & & & & & & \\ & & 1 & 1 & & & & & \\ & & & & 1 & & & & \\ & & & & & 1 & 1 & & \\ & & & & & & & 1 & \\ & & & & & & & & 1 \end{bmatrix}$$

Rozpatrzmy przykład struktury algorytmicznej układu

w przypadku, gdy $N = 4$. Wówczas procedura (4) przybiera postać (9):

$$\mathbf{Z}_{7 \times 1} = \mathbf{W}_7 \mathbf{P}_{7 \times 9}^{(2)} \mathbf{B}_9^{(1)} \mathbf{B}_9^{(2)} \mathbf{S}_9 \mathbf{A}_{9 \times 6}^{(2)} \mathbf{A}_{6 \times 4}^{(1)} \mathbf{X}_{4 \times 1}, \quad (9)$$

zaś poszczególne macierze:

$$\mathbf{A}_{6 \times 4}^{(1)} = \mathbf{T}_{3 \times 2} \otimes \mathbf{I}_2, \quad \mathbf{A}_{9 \times 6}^{(2)} = \mathbf{I}_3 \otimes \mathbf{T}_{3 \times 2},$$

$$\mathbf{B}_9^{(1)} = \mathbf{T}_3 \otimes \mathbf{I}_3, \quad \mathbf{B}_9^{(2)} = \mathbf{I}_3 \otimes \mathbf{T}_3,$$

$$\mathbf{W}_7 = \text{diag}(b^6, b^5, b^4, b^3, b^2, b^1, b^0),$$

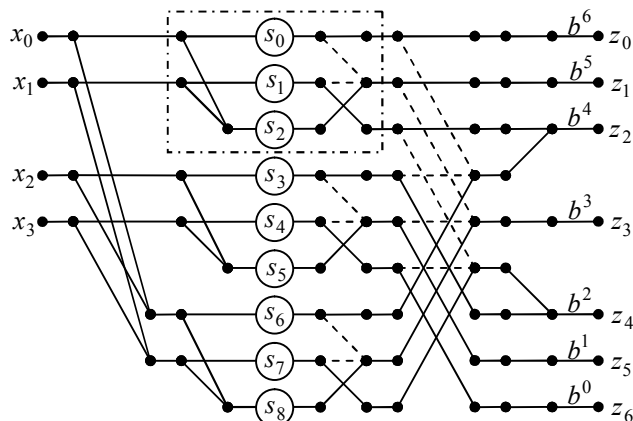
$$\mathbf{S}_9 = \text{diag}(y_0, y_1, y_0 + y_1, y_2, y_3, y_2 + y_3, y_0 + y_2, y_1 + y_3, y_0 + y_1 + y_2 + y_3).$$

Przedstawiony algorytm zakłada podział liczb aż do pojedynczych cyfr, co powoduje, że elementy wektora wyjściowego $\mathbf{Z}$ mogą być liczbami dwucyfrowymi. Aby uzyskać wynik w postaci wektora pojedynczych cyfr należy wykonać operację przeniesienia, której przykładowy kod znajduje się na rysunku 1 (środowisko Matlab). Można także wymnożyć wektor wyjściowy przez macierz współczynników $\mathbf{W}$, wówczas po zsumowaniu wszystkich wyjść otrzymamy wynik mnożenia w postaci pojedynczej liczby.

```
% przeniesienie
zz(1, 2:2*N)=z;
for i=2*N:-1:2
    zz(i-1)=zz(i-1)+floor(zz(i)/10);
    zz(i)=mod(zz(i), 10);
end
```

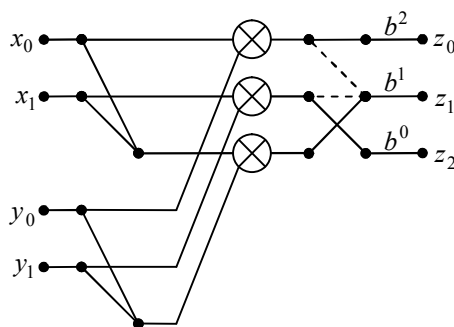
Rys. 1. Kod funkcji przeniesienia w środowisku Matlab

Rysunki 2 oraz 3 przedstawiają modele grafo-strukturalne ilustrujące strukturę algorytmiczną układu procesorowego do wyznaczania iloczynu liczb zgodnie z opracowaną procedurą odpowiednio dla $N = 4$ i $N = 2$. Liniami prostymi oznaczone są operacje transferu danych. W przypadku tych modeli skupienie linii prostych w odpowiednich punktach oznacza operację dodawania, przy czym przerywane linie oznaczają negację, natomiast linie rozchodzące się (rozgałęzienia) – zwykłe operacje dublowania danych. Kółkami na tych grafach są przedstawione operacje (bloki) mnożenia przez stałą w nich wpisaną (Rys. 2) lub mnożenie stałych znajdujących się na wejściach (Rys. 3).



Rys. 2. Model grafo-strukturalny organizacji procesu obliczeniowego wyznaczania iloczynu liczb według procedury (8) dla przykładu $N=4$

Zaznaczony ramką fragment na rysunku 2 odpowiada modelowi dla mnożenia liczb długości $N = 2$ (Rys 4).



Rys. 3. Model grafostrukturalny dla przykładu $N=2$, z jednoczesnym wyznaczaniem mnożników (macierzy S)

3. OSZACOWANIE LICZBY OPERACJI CZĄSTKOWYCH MNOŻENIA ORAZ OPERACJI DODAWANIA

Jak widać, zaproponowany w artykule algorytm pozwala zredukować łączną liczbę operacji mnożenia oraz dodawania względem metody „naiwnej”. Dla rozważonego przykładu mamy 9 mnożeń zamiast 16 oraz 22 dodawania zamiast 9, przy czym operacja dodawania jest mniej kosztowna niż operacja mnożenia.

Dla dowolnego N liczba mnożeń niezbędnych do realizacji algorytmu może być opisana za pomocą następującego wzoru:

$$\theta_x = 3^m, \quad (10)$$

natomiast liczba dodawań jest określona jako:

$$\theta_+ = 2m \cdot 3^{m-1} + \sum_{i=1}^m (2^i \cdot 3^{m-i}). \quad (11)$$

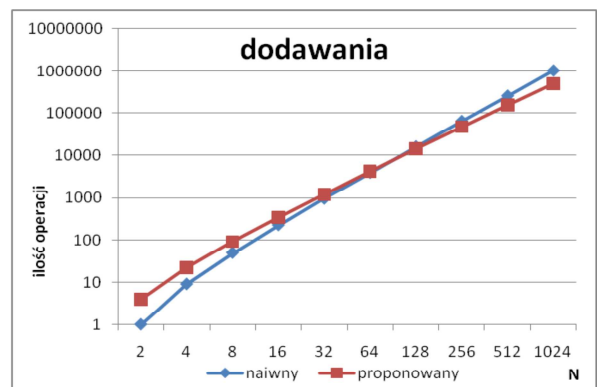
W tabeli 1 przedstawione są wyniki oszacowania liczby bloków arytmetycznych wymaganych przy implementacji opracowanego rozwiązania w porównaniu z implementacją metody „naiwnej”.

Okazuje się, że mimo 3 sumatorów więcej dla pierwszego kroku obliczeń nadwyżka ta zaczyna maleć wraz ze wzrostem długości liczby i dla $N=128$ jest już mniej sumatorów do wykonania dodawań cząstkowych niż przy implementacji metody naiwnej. Powodem tej tendencji spadkowej jest znaczna redukcja bloków mnożenia, więc dla dużych liczb rozpatrywanie wpływu stosunku kosztu sumatora względem bloku mnożenia dla różnych układów cyfrowych jest bezzasadne. Należy jednak zauważyć, że podana liczba sumatorów nie uwzględnia tych ostatnich, redukujących ilość wyjść (macierz P), ponieważ nie jest to krok obowiązkowy i zależy on od implementacji.

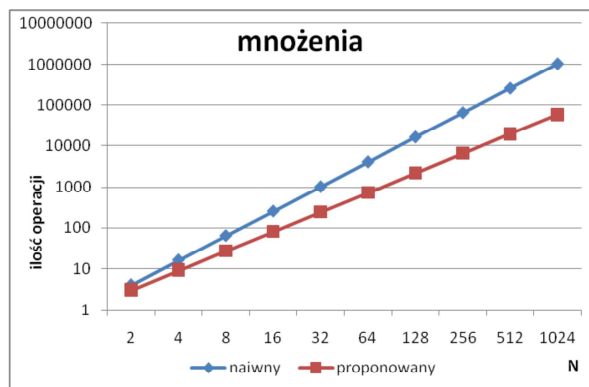
Tabela 1.
Oszacowanie liczby bloków arytmetycznych dla różnych długości liczb (N)

N	Bloki mnożenia		Bloki dodawania	
	naiwny	proponow.	naiwny	proponow.
2	4	3	1	4
4	16	9	9	22
8	64	27	49	92
16	256	81	225	346
32	1024	243	961	1232
64	4096	729	3969	4246
128	16384	2187	16129	14324
256	65536	6561	65025	47602
512	262140	19683	261120	156440
2^{10}	1048600	59049	1046500	509710

Na poniższych wykresach w skali logarytmicznej zaprezentowane są oszacowane wartości niezbędnej liczby dwuwęściowych sumatorów (Rys. 4) oraz bloków mnożenia (Rys. 5) wymaganych przy implementacji rozpatrywanego w artykule rozwiązania w stosunku do metody naiwnej.



Rys. 4. Ilość operacji dodawania (sumatorów) w zależności od długości mnożonych liczb



Rys. 5. Liczba operacji mnożenia (układów mnożących) w zależności od długości mnożonych operandów

4. PODSUMOWANIE

W artykule opisano strukturę algorytmiczną układu procesorowego do liczenia iloczynów długich, N -elementowych operandów. Na „racjonalność” prezentowanego rozwiązania wpływ ma przede wszystkim znaczna redukcja liczby bloków, realizujących cząstkowe operacje mnożenia, ale także możliwość przeprowadzenia tych mnożeń równoległe, jeśli tylko platforma implementacyjna na to pozwala.

Mimo, iż podobne rozwiązania były proponowane [8], przedstawione rozwiązanie posiada kilka dodatkowych atutów, takich jak wyeliminowanie rekurencji, prostota bądź intuicyjność implementacji. Przewagi te wynikają z zastosowania sukcesywnie wybranego sposobu zrównoleglenia obliczeń na etapie syntezy struktury algorytmicznej układu oraz racjonalnego doboru konstrukcji macierzowych, opisujących przekształcenia danych na każdym etapie realizacji wektorowo-macierzowej procedury obliczeniowej.

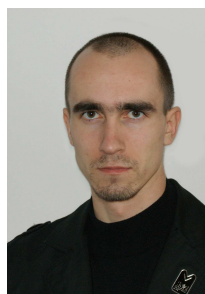
W artykule dla przejrzystości zostały pokazany warianty syntezy struktur układów (modele grafostrukturalne) dla przykładów, gdy $N = 2$ oraz $N = 4$. Oczywiście jest, że w podobny sposób mogą być skonstruowane efektywne (posiadające mniej bloków mnożących oraz sumatorów) i równoległe struktury układów do mnożenia dowolnie długich liczb.

Kolejną rzeczą, nad jaką należałoby zastanowić się jest opłacalność sprzętowej realizacji proponowanego podejścia w zależności od długości operandów, lub też celowość implementacji metody, na bazie systemu wielordzeniowego, który w pełni by wykorzystał potencjał tego rozwiązania.

LITERATURA

- [1] Biernat J., *Arytmetyka Komputerów*, Wydawnictwo Naukowe PWN, 1996.
- [2] Bodrato M., *Toward Optimal Toom-Cook Multiplication for Univariate and Multivariate Polynomials in Characteristic 2 and 0*, WAIFI'07, Springer, 2007.
- [3] Burrus C. S., Parks T. W., Potts J. F., *DFT/FFT and Convolution Algorithms and Implementation*, John Wiley & Sons, 1985.

- [4] Cook S. A., *On the minimum computation time of functions*, Harvard University - PhD thesis, 1966.
- [5] Dagman E.E., Kukharev. G.A., *Szybkie dyskretne transformaty ortogonalne*, Wydawnictwo Nauka, 1983.
- [6] Fürer M., Faster integer multiplication, STOC '07: Proceedings of the thirty-ninth annual, ACM symposium on Theory of computing (New York, NY, USA), ACM, 2007, pp. 57-66.
- [7] Karatsuba A., Ofman Y., *Multiplication of Many-Digital Numbers by Automatic Computers*. Doklady Akad. Nauk SSSR Vol. 145, 1962, pp. 293-294. Translation in *Physics-Doklady* 7, 1963, pp. 595-596.
- [8] Liu C.-B., Huang C.-H., *Design and Implementation of Long-Digit Karatsuba's Multiplication Algorithm Using Tensor Product Formulation*, In The Ninth Workshop on Compiler Techniques for High Performance Computing, 2003, pp. 23-30.
- [9] Schönhage A., Strassen V., *Schnelle Multiplikation großer Zahlen*, Computing 7, 1971, pp. 281-292.
- [10] Toom A.L., The complexity of a scheme of functional elements simulating the multiplication of integers, Dokl. Akad. Nauk SSSR 150 (1963), 496-498, (in Russian). English translation in Soviet Mathematics 3, 714-716, 1963.
- [11] Ćariov A., *Modele algorytmiczne i struktury wysokowydajnych procesorów cyfrowej obróbki sygnałów*, Szczecin, Informa, 2001.
- [12] Ćariov A., *Strategie racjonalizacji obliczeń przy wyznaczaniu iloczynów macierzowo-wektorowych*. Metody Informatyki stosowanej, Nr 1, 2008, str. 147-158.
- [13] Toom A., *The Complexity of a Scheme of Functional Elements Realizing the Multiplication of Integers.*, Soviet Mathematics - Doklady, 1963, v. 3, pp 714-716.



mgr. inż. Marek Gliszczyński
Zachodniopomorski Uniwersytet
Technologiczny w Szczecinie
Wydział Informatyki
Katedra Architektury Komputerów
i Telekomunikacji

ul. Żołnierska 49
71-210 Szczecin
tel.: 502 147 122

e-mail: mgliszczynski@wi.zut.edu.pl



dr hab. inż. Alexandr Ćariov
Zachodniopomorski Uniwersytet
Technologiczny w Szczecinie
Wydział Informatyki
Katedra Architektury Komputerów
i Telekomunikacji

ul. Żołnierska 49
71-210 Szczecin
tel.: 60 775 31 53

e-mail: atariov@wi.ps.pl