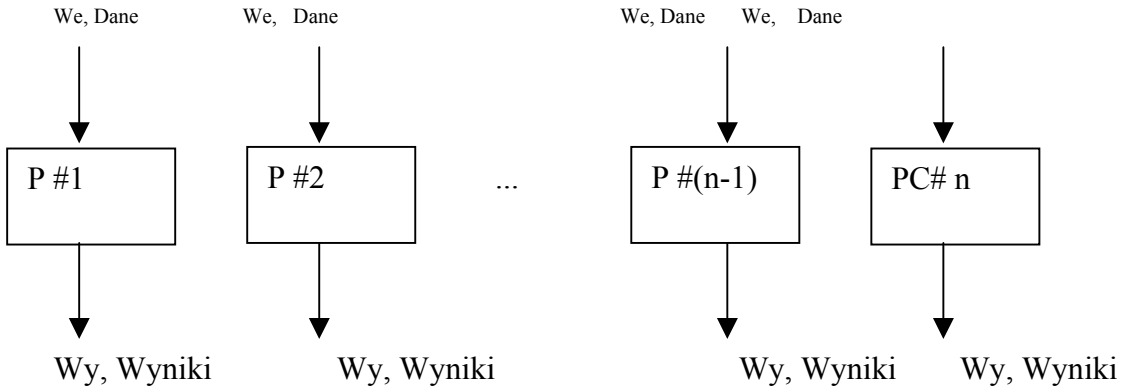
11.1 Przetwarzanie równoległe, potokowe i sieci systoliczne

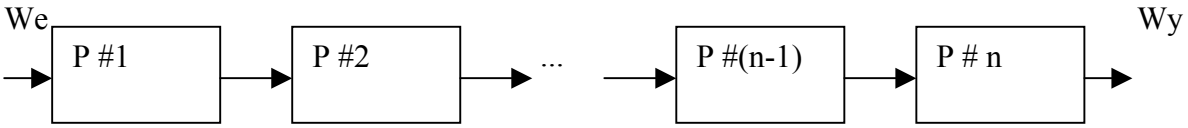
1. Przetwarzanie równoległe, potokowe i sieci systoliczne

Równoległy system przetwarzania danych (ang. *parallel processing*) pokazany jest na rys. 1. Pomysł, by w tym samym czasie realizować poszczególne fragmenty algorytmu, jest naturalną drogą pozwalającą przyśpieszyć system cyfrowy. Nie każdy jednak algorytm daje się rozbić na takie niezależne części, by mógł być realizowany równolegle. Z drugiej strony, jeśli mamy *n* procesorów, nie musimy na wszystkich realizować tego samego zadania.



Rys. 1. Przetwarzanie równoległe danych; P# k oznacza procesor o numerze *k*

Innym sposobem przyśpieszenia przetwarzania jest tzw. *przetwarzania potokowe* (ang. *pipelining*). W przetwarzaniu potokowym *n* procesorów realizuje fragmenty algorytmu, tak jak dzieje się to na taśmie produkcyjnej, przekazując „półprodukt" do następnego procesora po ustalonym czasie *T*. Schemat blokowy układu przetwarzania potokowego pokazany jest na rys.2.



Rys. 2. Przetwarzanie potokowe

Układy systoliczne lub - jak czasem mówimy - *sieci systoliczne* lub *tablice systoliczne* (ang. *systolic array*) łączą dwie wyżej opisane koncepcje przyśpieszania realizacji algorytmu: przetwarzanie równoległe i przetwarzanie potokowe i mają następujące cechy:

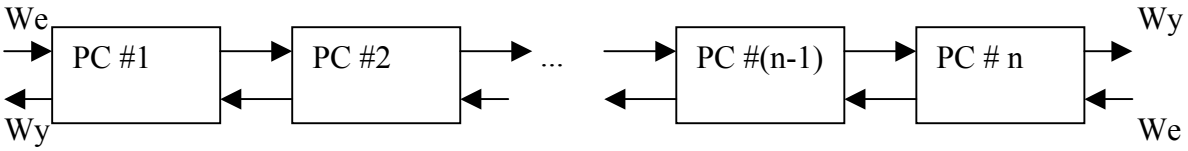
- regularność struktury
- modularność
- lokalność połączeń
- przetwarzanie równoległe i potokowe

Sieci systoliczne służą do realizacji różnych ważnych algorytmów np.:

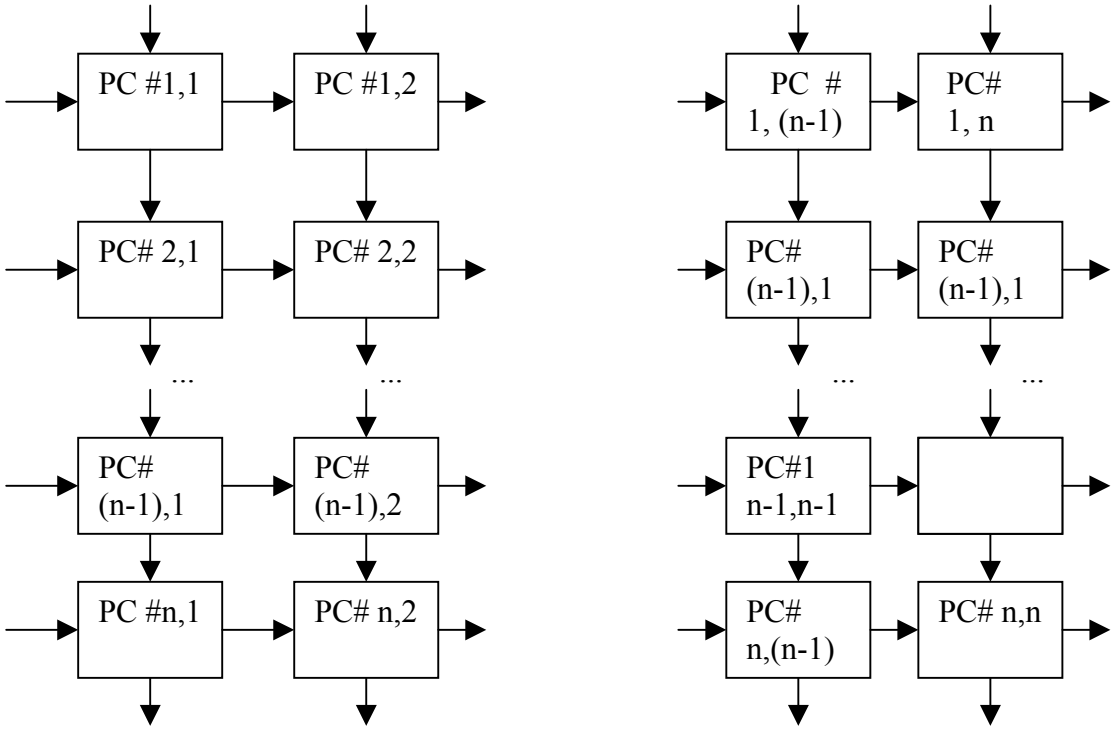
- sortowania
- obliczania iloczynu skalarnego wektorów,
- mnożenia wektora przez macierz

- mnożenia macierzy
- obliczania DFT (w gruncie rzeczy jest to też mnożenie wektora przez macierz)
- realizacji metody najmniejszych kwadratów

a)



b)



Rys. 3. Typowe struktury sieci systolicznych a) struktura jednowymiarowa b) struktura dwuwymiarowa

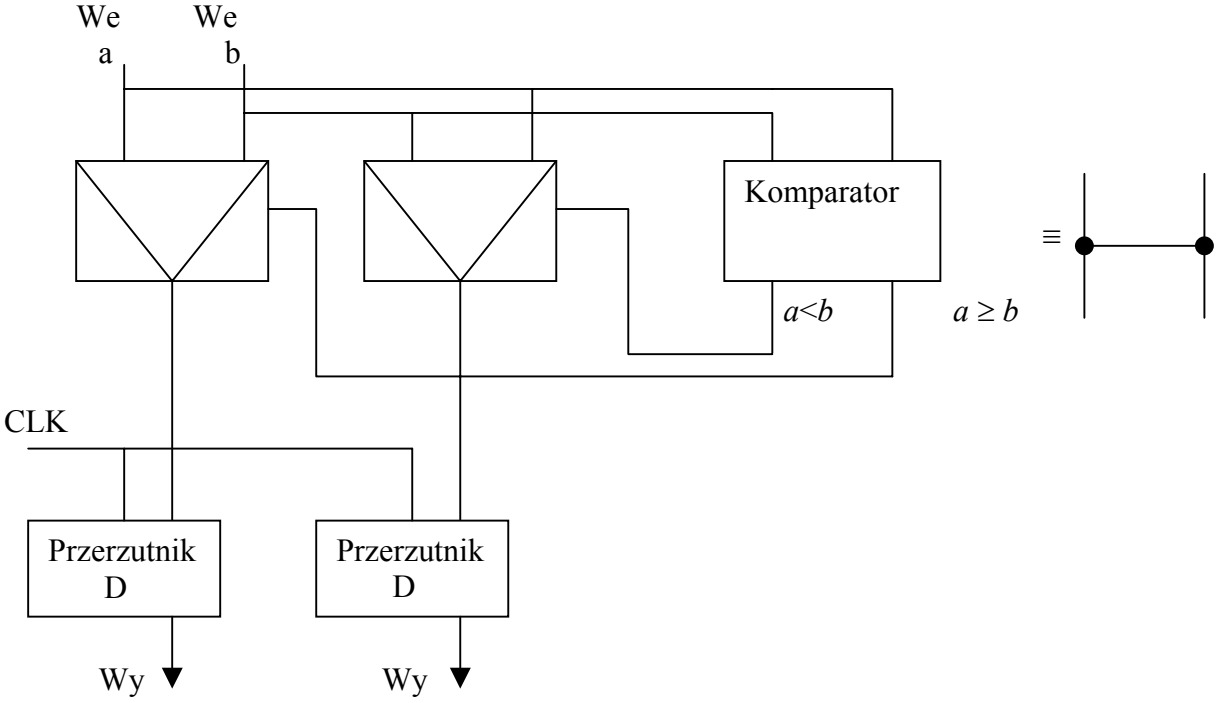
Omówimy dokładniej *sieci systoliczne sortujące* lub - jak mówimy krócej - *układy sortujące*. Układy sortujące są ważnym z praktycznego punktu widzenia układem służą np. do realizacji filtrów medianowych i kwantylowych w cyfrowym przetwarzaniu sygnałów.

Warto w tym miejscu przypomnieć, co to jest sortowanie. Sortowanie ciągu liczbowego $a_1, a_2, ..., a_n$ to jego porządkowanie. Formalnie rzecz biorąc sortowanie ciągu sprowadza się do znalezienia takiej permutacji $\pi :< 1, n > \rightarrow < 1, n >$, by spełnione były nierówności $a_{\pi(1)} \leq a_{\pi(2)} \leq ... \leq a_{\pi(n)}$.

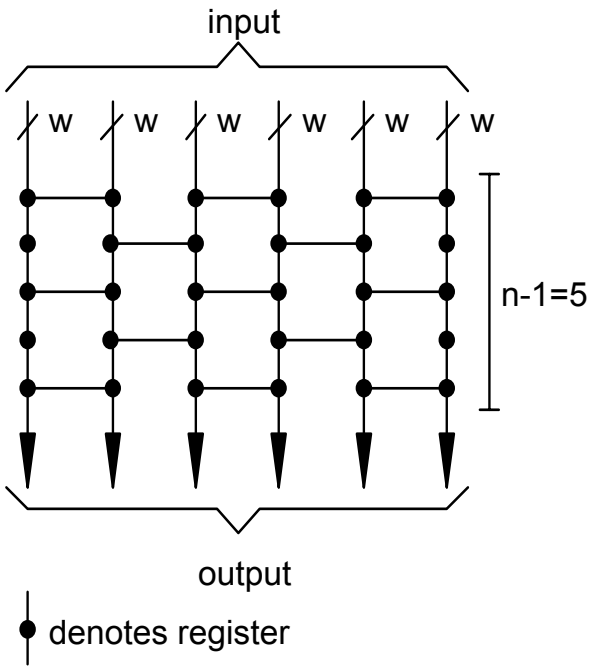
Algorytmów sortowania jest dużo, nie wszystkie jednak dobrze nadają się do - jak mówimy w żargonie - „systolizacji". W sieciach systolicznych znajdują zastosowanie trzy algorytmy sortowania:

- algorytm bubblesort - sortowanie bąbelkowe
- algorytm modified bubblesort – zmodyfikowane sortowanie bąbelkowe
- algorytm insertion sort – sortowanie przez wstawianie

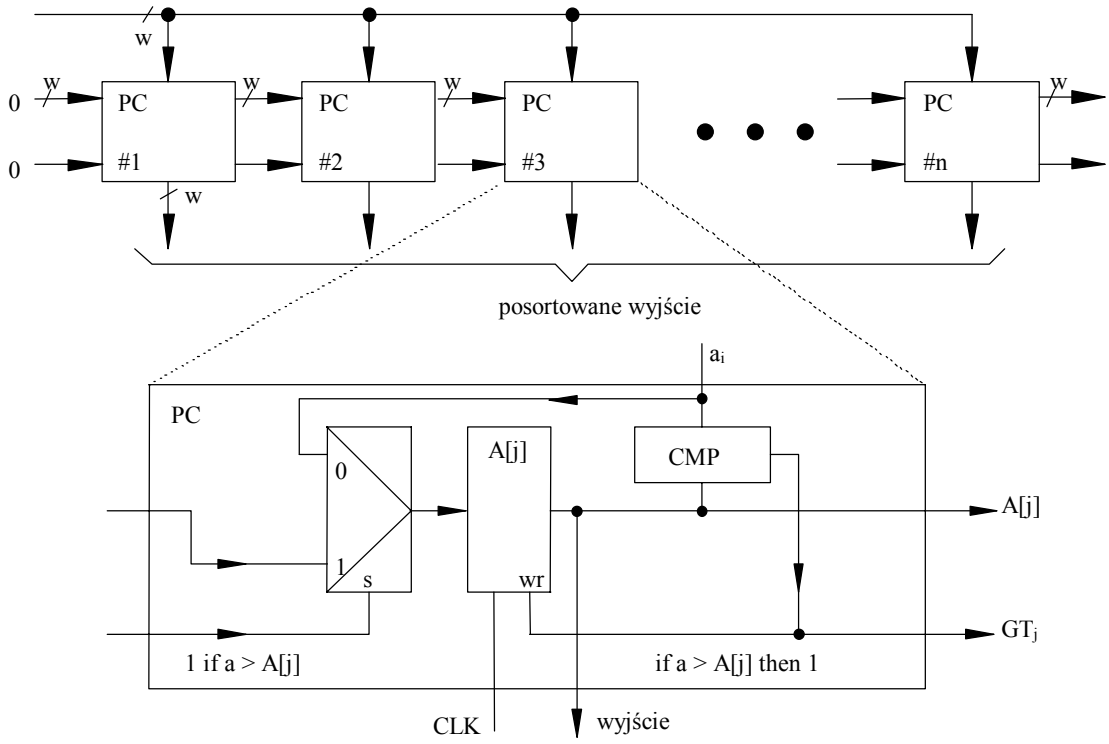
Na rys. 4. pokazana jest komórka podstawowa PC (ang. *processing cell*) sortującej sieci systolicznej, a na rys. 5. pokazana jest systoliczna sieć sortująca oparta na wykorzystaniu algorytmu modified bublesort.



Rys. 4. Układ komórki podstawowej PC oraz jej uproszczony symbol (po prawej)



Rys. 5. Sieć systoliczna sortująca (pięcioelementowy ciąg skończony liczb w kodzie NKB) oparta na algorytmie modified bubblesort; na wejście podawane są słowa o długości *w*



Rys. 6. Sieć systoliczna sortująca (ciąg skończony o długości *n*) oparta na algorytmie insertion sort; na wejście podawane są słowa o długości *w;* liczby zapisywane są w kodzie NKB

2. Arytmetyka rozłożona

Co to jest *arytmetyka rozłożona* i do czego służy? Jest to równoległo-szeregowa metoda - czy raczej koncepcja - układowej realizacji układów obliczających wartość kombinacji liniowej.

$$y = \sum_{i=1}^{k} a_i x_i \qquad (*)$$

gdzie $a_1, a_2, ..., a_k$ są stałymi (reprezentowanymi przez słowa binarne), a $x_1, x_2, ..., x_k$ danymi wejściowymi (reprezentowanymi również przez słowa binarne)

Na wyrażenie (*) można patrzeć jak na iloczyn skalarny dwóch wektorów z przestrzeni R^n : $(a_1, a_2, ..., a_k)$ i $(x_1, x_2, ..., x_k)$.

Metoda dobrze nadaje się do realizacji filtrów cyfrowych NOI, SOI, do obliczania DFT i do realizacji cyfrowych filtrów adaptacyjnych (wymiana zawartości pamięci daje nowy filtr cyfrowy).

Rozwiązania układowe arytmetyki rozłożonej dobrze nadają się do realizacji iloczynu skalarnego $(a_1, a_2, ..., a_k)$ z takim „przesuwającym" się wektorem:

$$x_1, x_2, x_3, ..., x_k$$
$$x_2, x_3, ..., x_k, x_{k+1}$$
$$x_3, x_4, ..., x_{k+1}, x_{k+2}$$

a to właśnie robią filtry FIR.

Istota rzeczy: zastępujemy mnożenie sumowaniem wyników częściowych zapisanych w pamięci. Metoda okazuje się tanią, prostą i oszczędną układowo i prosta w realizacji.

Wyjaśnimy koncepcję arytmetyki rozłożonej na przykładzie. Niech liczby $a_1, a_2, ..., a_k \in Q$, $x_1, x_2, ..., x_k \in Q$ będą reprezentowane w zapisie $U2$. Załóżmy, że dla każdego $n \in N$ chcemy obliczyć $y(n)$ zdefiniowane tak:

$$y(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) + b_1 y(n-1) + b_2 y(n-2) \qquad (**)$$

Takim właśnie równaniem różnicowym opisywana jest tzw. sekcja bikwadratowa filtru NOI (nieskończona odpowiedź impulsowa).

Przyjmijmy, że wartości bezwzględne wszystkich sygnałów nie przekraczają jedności i są przedstawione za pomocą B+1 bitów w zapisie $U2$, tzn.:

$$x(k) = -x_k^0 + \sum_{j=1}^{B} x_k^j 2^{-j} = \overbrace{(x_k^0, x_k^1, x_k^2, ..., x_k^B)}^{B+1 \ bitów}$$

słowo binarne reprezentuje $x(k)$ („rozpisujemy sygnały na bity”)

$$y(k) = -y_k^0 + \sum_{j=1}^{B} y_k^j 2^{-j} = (y_k^0, y_k^1, y_k^2, ..., y_k^B)$$

gdzie, $x_k^j, y_k^j \in \{0,1\}$. Nasze wyrażenie (**) na y(n) możemy teraz zapisać następująco:

$$y(n) = -\overbrace{(x_n^0 a_0 + x_{n-1}^0 a_1 + a_2 x_{n-2}^0 + b_1 y_{n-1}^0 + b_2 y_{n-2}^0)}^{bity \ znaków} + \qquad (***)$$
$$+ \sum_{j=1}^{B} 2^{-j} (a_0 x_n^j + a_1 x_{n-1}^j + a_2 x_{n-2}^j + b_1 y_{n-1}^j + b_2 y_{n-2}^j)$$

Niczego nadzwyczajnego nie zrobiliśmy, poza zmianą kolejności sumowania.

Zauważmy, że w wyrażeniach w nawiasach a_0, a_1, a_2, b_1, b_2 są stałe, zatem każdy „nawias” jest wartością pewnej funkcji pięciu zmiennych.

$$f(e_1, e_2, e_3, e_4, e_5) = a_0 e_1 + a_1 e_2 + a_2 e_3 + b_1 e_4 + b_2 e_5$$

gdzie $e_1, e_2, ..., e_5 \in \{0,1\}$. Nasza funkcja f może przyjmować tylko 32 wartości. Funkcję tę można zrealizować jako układ kombinacyjny lub zapamiętać w pamięci PROM.

Zatem przepisując (***) z użyciem nowego oznaczenia dostajemy:

$$y(n) = -f(x_n^0, x_{n-1}^0, x_{n-2}^0, y_{n-1}^0, y_{n-2}^0) + \sum_{j=1}^{B} 2^{-j} f(x_n^j, x_{n-1}^j, x_{n-2}^j, y_{n-1}^j, y_{n-2}^j)$$

Widać, że wyeliminowaliśmy w ten sposób mnożenie, a $y(n)$ obliczać możemy przez sumowanie i przesuwanie odpowiedniej wartości funkcji f.