Teoria

1 Historia rozwoju komputerów

1. Liczydło
2. Pascalina - maszyna licząca Pascala (dodawanie i odejmowanie)
3. Maszyna mnożąca Leibniza (dodawanie, odejmowanie, mnożenie, dzielenie, pierwiastek kwadratowy
4. Maszyna różnicowa - Charles Babbage, obliczanie wartości matematycznych do tablic
5. Maszyna analityczna - Charles Babvage, programowalna za pomocą kart perforowanych
6. Elektryczna maszyna sortująca i tabelaryzująca Holleritha 1890
7. Kalkulator elektromechaniczny Mark I, tablicowanie funkcji, całkowanie numeryczne, rozwiązywanie równań różniczkowych, rozwiązywanie układów równań liniowych, analiza harmoniczna, obliczenia statystyczne
8. Maszyny liczące Z1: pamięć mechaniczna, zmiennoprzecinkowa reprezentacja liczb, binarna jednostka zmiennoprzecinkowa
9. Z3: Pierwsza maszyna w pełni automatyczna, kompletna w sensie Turinga, pamięć przekaźnikowa
10. Colossus i Colossus 2
11. ENIAC
12. EDVAC - J. von Neumann (wtedy utworzył swoją architekturę)
13. UNIVAC I (pierwszy udany komputer komercyjny)
14. IBM 701, potem 709
15. po 1955 zaczyna się zastosowanie tranzystorów w komputerach (komputery II generacji)
16. po 1965 komputery III generacji z układami scalonymi
17. od 1971 komputery IV generacji - z układami scalonymi wielkiej skali inegracji VLSI

2 Architektura CISC

2.1 Znaczenie

Complex Instruction Set Computers

2.2 Przyczyny rozwoju architektury CISC

- Drogie, małe i wolne pamięci komputerów
- Rozwój wielu rodzin komputerów
- Duża popularność mikroprogramowalnych układów sterujących (prostych w rozbudowie)
- Dążenie do uproszczenia kompilatorów: im więcej będzie rozkazów maszynowych odpowiadających instrukcjom języków wyższego poziomu tym lepiej; model obliczeń pamięć – pamięć.

2.3 Cechy architektury CISC

- Duża liczba rozkazów (z czego te najbardziej zaawansowane i tak nie były używane)
- Duża ilość trybów adresowania (związane z modelem obliczeń)
- Duży rozrzut cech rozkazów w zakresie:
  - złożoności
  - długości (szczególnie to - nawet kilkanaście bajtów)
  - czasów wykonania
- Model obliczeń pamięć - pamięć
- Niewiele rejestrów - były droższe niż komórki pamięci i przy przełączaniu kontekstu obawiano się wzrostu czasu przełączania kontekstu (chowanie rejestrów na stos i odwrotnie)
- Przerost struktury sprzętowej przy mało efektywnym wykorzystaniu list rozkazów

CIEKAWOSTKA: Przeanalizowano jakieś tam programy i w procesorze VAX 20% najbardziej złożonych rozkazów odpowiadało za 60% kodu, stanowiąc przy tym ok 0.2% wywołań.
W procesorze MC68020 71% rozkazów nie zostało nawet użytych w badanych programach

3 Architektura RISC

3.1 Znaczenie

Reduced Instruction Set Computers.

3.2 Przyczyny rozwoju

- Poszukiwanie optymalnej listy rozkazów
- Chęć wykonania mikroprocesora o funkcjach pełnego ówczesnego procesora

3.3 Pierwszy procesor RISC

Procesor RISC I (1980), D. Patterson (Berkeley University)
Założenia projektowe:

- Wykonanie jednego rozkazu w jednym cyklu maszynowym
- Stały rozmiar rozkazów – uproszczenie metod adresacji
- Model obliczeń rejestr – rejestr: komunikacja z pamięcią operacyjną tylko za pomocą rozkazów LOAD i STORE.
- Wsparcie poprzez architekturę języków wysokiego poziomu.

Efekty realizacji fizycznej:

- 44 420 tranzystorów (ówczesne procesory CISC zawierały ok. 100 000 tranzystorów)
- lista rozkazów = 32 rozkazy
- dwustopniowy potok – strata tylko 6% cykli zegara, zamiast 20% (w związku z realizacją skoków)

3.4 Cechy architektury RISC

1. Stała długość i prosty rozkaz formatu
2. Nieduża liczba trybów adresowania
3. Niezbyt obszerna lista rozkazów
4. Model obliczeń rejestr-rejestr - dostęp do pamięci operacyjnej tylko w rozkazach LOAD i STORE
5. Duży zbiór rejestrów uniwersalnych
6. Układ sterowania – logika szyta
7. Intensywne wykorzystanie przetwarzania potokowego
8. Kompilatory o dużych możliwościach optymalizacji potoku rozkazów

3.5 Format rozkazu procesora RISC I

7	1	5	5	1	13
OPCODE	SCC	DEST	SRC1	IMM	SRC2

- OPCODE–kod rozkazu
- SCC – ustawianie (lub nie) kodów warunków
- DEST – nr rejestru wynikowego
- SRC1 – nr rejestru zawierającego pierwszy argument
- IMM – wskaźnik natychmiastowego trybu adresowania
- SRC2 – drugi argument lub nr rejestru (na 5 bitach)

3.6 Realizacja wybranych rozkazów

3.6.1 Rozkazy arytmetyczne

- **Tryb rejestrowy**: (IMM=0) R[DEST] $\leftarrow$ R[SRC1] op R[SRC2]
- **Tryb natychmiastowy:** (IMM=1) R[DEST] $\leftarrow$ R[SRC1] op SRC2

3.6.2 Rozkazy komunikujące się z pamięcią

- **LOAD** R[DEST] $\leftarrow$ M[AE]P
- **STORE** M[AE] $\leftarrow$ R[DEST]

3.6.3 Adres efektywny

- **Tryb z przesunięciem** AE = R[SRC1] + SRC2 = RX + S2
- **Inny zapis powyższego** AE = RX + S2
- **Tryb absolutny** AE = R0 + S2 = S2 (R0 $\equiv$ 0)
- **Tryb rejestrowy pośredni** AE = RX + 0 = RX

Tryb absolutny oraz tryb rejestrowy pośredni są przypadkami szczególnymi.

Tabela 1: Rejestry

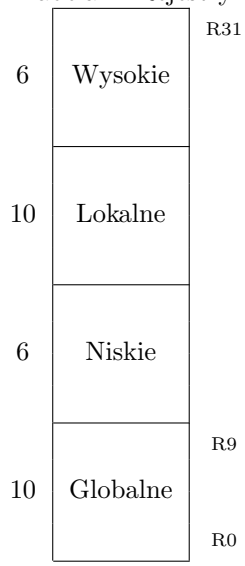
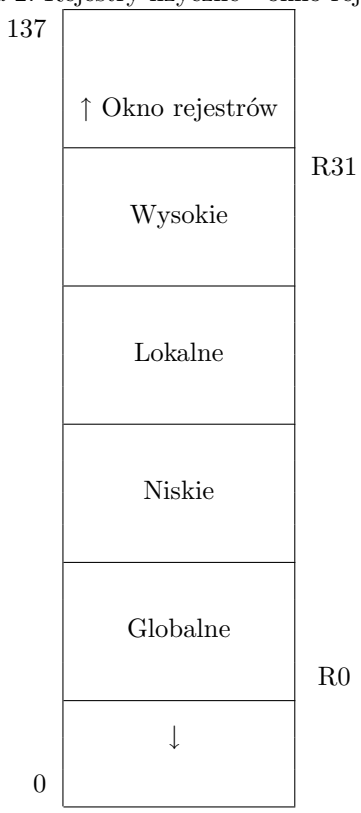


Tabela 2: Rejestry fizyczne - okno rejestrów



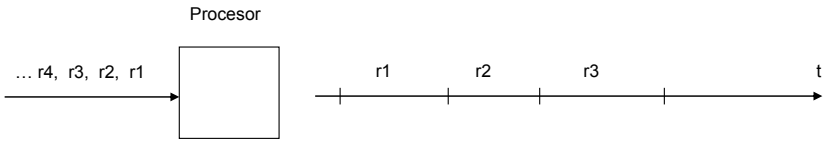
3.7 Logiczna organizacja rejestrów procesora RISC I

3.8 Okno rejestrów

4 Mechanizmy potokowe

4.1 Realizacja rozkazów w procesorze niepotokowym

Rozkazy wykonywane są liniowo w czasie - jeden po drugim, w takiej kolejności w jakiej przyjdą do procesora.

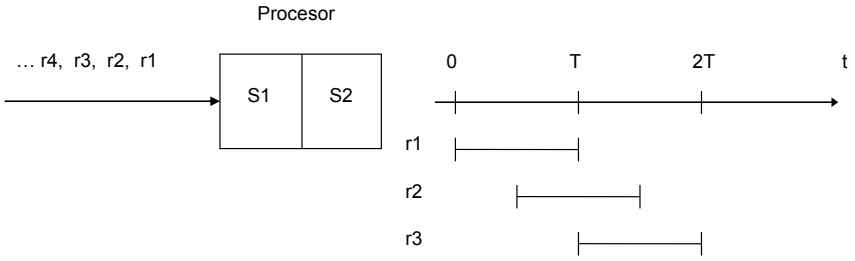


4.2 Potokowe wykonanie rozkazówdla prostej organizacji cyklu rozkazowego

Prosty podział procesora na moduły:

- S1 - pobranie rozkazu
- S2 - wykonanie rozkazu

Zakładając, że czas pracy obu modułów jest równy, wówczas 3 rozkazy mogą zostać wykonane w 2 okresach. 1 T - pobranie i wykonanie rozkazu. W momencie gdy pierwszy rozkaz zostanie pobrany, w chwili 0.5 T S1 może pobrać kolejny.



4.3 Podział cyklu rozkazowego na większą liczbę faz

Na przykładzie cyklu rozkazowego komputera Amdahl 470:

1. Pobranie rozkazu
2. Dekodowanie rozkazu
3. Obliczenie adresu efektywnego
4. Pobranie argumentów
5. Wykonanie operacji
6. Zapis wyniku

Zasada działania jest dokładnie taka sama jak w przypadku podziału na dwie fazy. Załóżmy, że jeden rozkaz wykonuje się w 7iu taktach zegarowych. 1 T = 7 F. Wówczas w momencie gdy rozkaz numer 1 znajduje się w 5tym takcie wykonania rozkaz numer 5 może zostać pobrany.

Rozkazy	Fazy zegarowe						
	1	2	3	4	5	6	7
S1	r1	r2	r3	r4	r5	r6	r7
S2		r1	r2	r3	r4	r5	r6
S3			r1	r2	r3	r4	r5
S4				r1	r2	r3	r4
S5					r1	r2	r3
S6						r1	r2

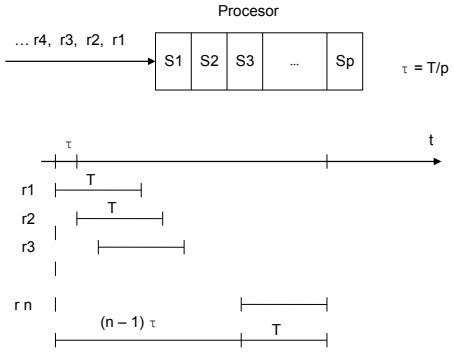
Tabela 3: Realizacja ciągu rozkazów w wielostopniowym procesorze potokowym.

4.4 Analiza czasowa potokowej realizacji ciągu rozkazów

Założenia:

- **P** - liczba faz
- **T** - okres
- $\frac{T}{P} = \tau$ - czas wykonania pojedynczej fazy

$(n-1) \times \tau$ - czas rozpoczęcia wykonywania n-tego rozkazu.



4.5 Przyspieszenie dla potokowego wykonania rozkazów

- Czas wykonywania rozkazu w procesorze niepotokowym (dla n rozkazów)

$$t = n \times T$$

- Czas wykonywania rozkazu w procesorze potokowym dla idealnego przypadku, gdy $\tau = \frac{T}{P}$

$$t = (n-1) \times \tau + T = (n - 1 + P) \times \frac{T}{P}$$

- Przyspieszenie jest stosunkiem czasu wykonywania rozkazów dla procesora niepotokowego do czasu dla procesora potokowego.

$$\lim_{n \to \infty} \frac{n \times T}{(n - 1 + P) \times \frac{T}{P}} = P$$

Maksymalne przyspieszenie (dla modelu idealnego) jest równe ilości faz.

4.6 Problemy z potokową realizacją rozkazów

Problemem związanym z realizacją potokową jest **zjawisko hazardu**.

- **Hazard sterowania** – problemy z potokową realizacją skoków i rozgałęzień.
- **Hazard danych** – zależności między argumentami kolejnych rozkazów
- **Hazard zasobów** – konflikt w dostępie do rejestrów lub do pamięci

4.7 Rozwiązanie problemu hazardu sterowania

- Skoki opóźnione
- Przewidywanie rozgałęzień

4.8 Skoki opóźnione

4.8.1 Założenia

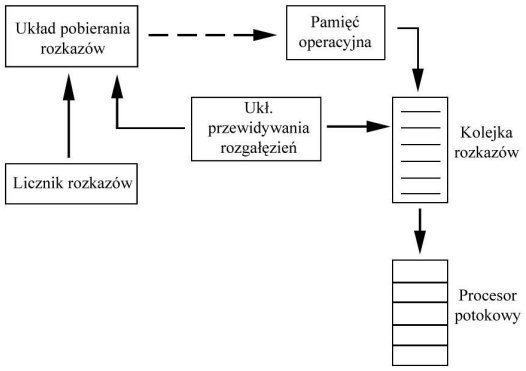
- Rozkaz następny po skoku jest zawsze całkowicie wykonywany
- To znaczy, że efekt skoku jest opóźniony o jeden rozkaz

4.8.2 Działanie

Zmienia kod programu w trakcie kompilacji, jeśli widzi taka potrzebę. Sprowadza się to do dwóch możliwości:

- Modyfikacja programu - dodanie rozkazu NOP po instrukcji skoku JMP
- Optymalizacja programu - zmiany kolejności wykonywania rozkazów

4.9 Przewidywanie rozgałęzień



4.9.1 Strategie

1. **Statyczne**
   - przewidywanie, że rozgałęzienie (skok warunkowy) zawsze nastąpi
   - przewidywanie, że rozgałęzienie nigdy nie nastąpi
   - podejmowanie decyzji na podstawie kodu rozkazu rozgałęzienia (specjalny bit ustawiany przez kompilator)
2. **Inne**
   - przewidywanie, że skok wstecz względem licznika rozkazów zawsze nastąpi
   - przewidywanie, że skok do przodu względem licznika rozkazów nigdy nie nastąpi
3. **Dynamiczne**
   - Tablica historii rozgałęzień.

4.9.2 Tablica historii rozgałęzień

Składa się z:

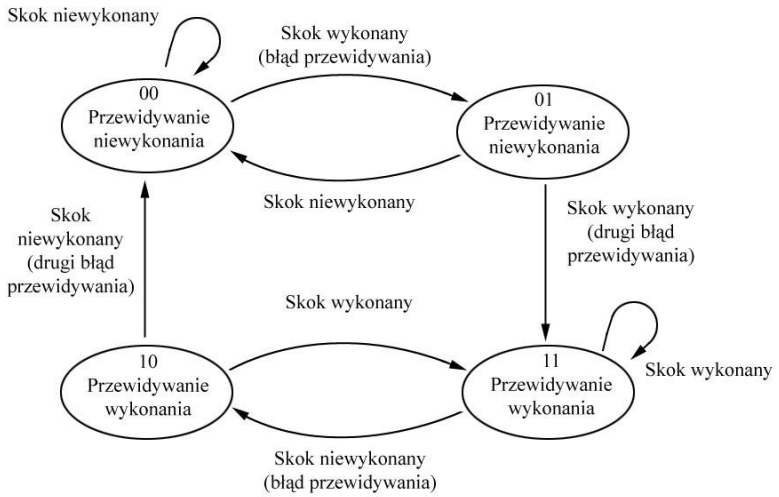
- Bit ważności
- Adres rozkazu rozgałęzienia
- Bity historii
- Adres docelowy rozgałęzienia (opcja)

Operacje wykonywane na tablicy historii rozgałęzień

- Sprawdzenie, czy adres rozkazu rozgałęzienia jest w tablicy
  - **Nie** – wtedy:
    * przewidywanie rozgałęzienia jest wykonywane według jednej ze strategii statycznych
    * do tablicy jest wpisywany adres rozkazu rozgałęzienia, informacja o wykonaniu/niewykonaniu rozgałęzienia (bit historii) i (opcjonalnie) adres docelowy rozgałęzienia

  - **Tak** - wtedy:
    * przewidywanie rozgałęzienia jest wykonywane według bitów historii
    * do tablicy jest wpisywana informacja o wykonaniu/niewykonaniu rozgałęzienia (uaktualnienie bitów historii)

- 1 bit historii - algorytm przewidywania rozgałęzień dla jednego bitu historii - kolejne wykonanie rozkazu rozgałęzienia będzie przebiegało tak samo jak poprzednie.
- 2 bity historii
  - algorytm przewidywania rozgałęzień dla dwóch bitów historii bazuje na 2-bitowym automacie skończonym.
  - Interpretacja dwóch bitów historii (x y):
    * y: historia ostatniego wykonania skoku (0 – nie, 1 – tak)
    * x: przewidywanie następnego wykonania skoku (0 – nie, 1 – tak)
    * Ogólna zasada przewidywania - zmiana strategii następuje dopiero po drugim błędzie przewidywania.



4.10 Metody rozwiązywania hazardu danych

4.10.1 Co to jest?

Hazard danych - zależności między argumentami kolejnych rozkazów wykonywanych potokowo.

4.10.2 Metody usuwania hazardu danych

Jest kilka sposobów:

- Sprzętowe wykrywanie zależności i wstrzymanie napełniania potoku
- Wykrywanie zależności na etapie kompilacji i modyfikacja programu (np. dodanie rozkazu NOP)
- Wykrywanie zależności na etapie kompilacji, modyfikacja i optymalizacja programu (np. zamiana kolejności wykonywania rozkazów)
- Wyprzedzające pobieranie argumentów (zastosowanie szyny zwrotnej)

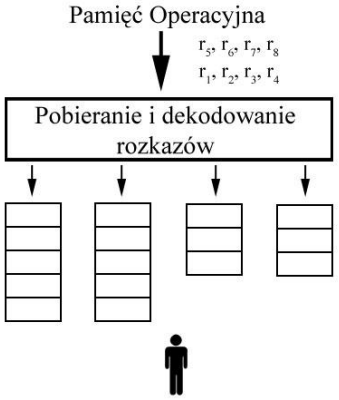
4.10.3 Problem

Jeśli faza wykonania rozkazu nie będzie mogła być wykonana w jednym takcie (np. dla rozkazów zmiennoprzecinkowych), to zachodzi konieczność wstrzymania napełniania potoku.

5 Architektura superskalarna

5.1 Co to jest?

Architektura umożliwiająca wykonanie w jednym takcie większej od 1 liczby instrukcji.



5.2 Cechy architektury superskalarnej

Możliwość wykonania kilku rozkazów w jednym takcie, co powoduje konieczność:

- Kilku jednostek potokowych
- Załadowania kilku rozkazów z pamięci operacyjnej w jednym takcie procesora

5.3 Zależności między rozkazami

5.3.1 Prawdziwa zależność danych

Read After Write (RAW)
Występuje w momencie kiedy jeden rozkaz wymaga argumentu obliczanego przez poprzedni rozkaz. Opóźnienie eliminowane za pomocą "wyprzedzającego pobierania argumentu" - dana nie jest zapisywana do rejestru, tylko pobierana bezpośrednio z poprzedniego rozkazu, który znajduje się w akumulatorze (jeżeli dobrze rozumiem rysunek ze slajdu 21, wykład 4).

5.3.2 Zależność wyjściowa

Write After Write (WAW)
Gdy rozkazy zapisująca dane do tego samego rejestru wykonują się równolegle to drugi z nich musi czekać aż pierwszy się zakończy. Układ sterujący musi kontrolować tego typu zależność.

5.3.3 Antyzależność

Write After Read (WAR)
W przypadku gdy pierwszy rozkaz czyta wartość rejestru, a drugi zapisuje coś do tego rejestru i oba wykonują się równolegle, to drugi musi czekać aż pierwszy odczyta swoje. Ten rodzaj zależności może zostać wyeliminowany przez sprzętowe wykrywanie zależności i wstrzymanie napełniania potoku na co najmniej takt.
Jeśli faza wykonania rozkazu nie będzie mogła być wykonana w jednym takcie (np. dla rozkazów zmiennoprzecinkowych), to zachodzi konieczność wstrzymania napełniania potoku.

5.3.4 Wnioski

- Dopuszczenie do zmiany kolejności rozpoczynania wykonania (wydawania) rozkazów i / lub zmiany kolejności kończenia rozkazów prowadzi do możliwości wystąpienia zależności wyjściowej lub antyzależności.

- Zawartości rejestrów nie odpowiadają wtedy sekwencji wartości, która winna wynikać z realizacji programu

5.4 Metody eliminacji zależności

5.4.1 Metoda przemianowania rejestrów

- Stosowana w przypadku zwielokrotnienia zestawu rejestrów.
- Rejestry są przypisywane dynamicznie przez procesor do rozkazów.
- Gdy wynik rozkazu ma być zapisany do rejestru Rn, procesor angażuje do tego nową kopię tego rejestru.
- Gdy kolejny rozkaz odwołuje się do takiego wyniku (jako argumentu źródłowego), rozkaz ten musi przejść przez proces przemianowania.
- Przemianowanie rejestrów eliminuje antyzależność i zależność wyjściową.

Przykład procesu przemianowania rejestrów:

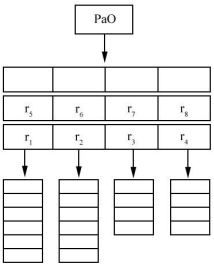
- I1: R3b ← R3a op R5a
- I2: R4b ← R3b + 1
- I3: R3c ← R5a + 1
- I4: R7b ← R3c op R4b

W powyższym przykładzie rozkaz I3 może być wykonany jako drugi (co zapobiegnie zależnościom RAW między I1 i I2 oraz I3 i I4), lub nawet jako pierwszy.

6 Architektura VLIW

6.1 Co to jest?

VLIW - *Very Long Instruction Word.*



6.2 Cechy

- Wspólna pamięć operacyjna
- Szeregowanie rozkazów

6.3 Szeregowanie rozkazów przez kompilator

- Podział rozkazów programu na grupy
- Sekwencyjne wykonywanie grup
- Możliwość równoległej realizacji rozkazów w ramach grupy
- Podział grupy na paczki
- Paczka = 3 rozkazy + szablon (3 x 41 + 5 = 128 bitów)
- Szablon - informacja o jednostkach funkcjonalnych, do których kierowane mają być rozkazy i ewentualna informacja o granicach grup w ramach paczki

6.4 Redukcja skoków warunkowych - predykacja rozkazów

Rozkazy uwarunkowane - uwzględnianie warunku w trakcie realizacji rozkazu.

6.5 Spekulatywne wykonanie rozkazów LOAD

- Problem: chybione odwołania do PaP (cache) i konieczność czekania na sprowadzenie do PaP linii danych
- Rozwiązanie: przesunięcie rozkazów LOAD jak najwyżej, aby zminimalizować czas ewentualnego oczekiwania.
- Rozkaz CHECK sprawdza wykonanie LOAD (załadowanie rejestru)

7 Wielowątkowość

7.1 Co to jest?

- Cecha systemu operacyjnego umożliwiająca wykonywanie kilku wątków w ramach jednego procesu
- Cecha procesora oznaczająca możliwość jednoczesnego wykonywanie kilku wątków w ramach jednego procesora (rdzenia)

7.2 Sprzętowa realizacja wielowątkowości

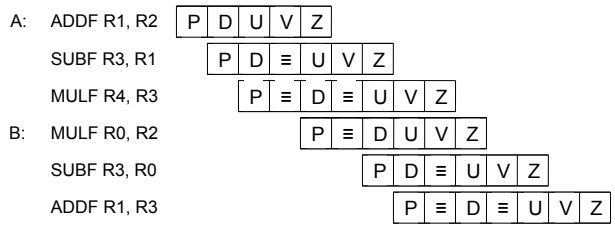
Celem współbieżnej realizacji dwóch (lub więcej) wątków w jednym procesorze (rdzeniu) jest minimalizacja strat cykli powstałych w trakcie realizacji pojedynczego wątku w wyniku:

- chybionych odwołań do pamięci podręcznej,
- błędów w przewidywaniu rozgałęzień,
- zależności między argumentami kolejnych rozkazów

7.3 Wielowątkowość gruboziarnista

Coarse-grained multithreading.

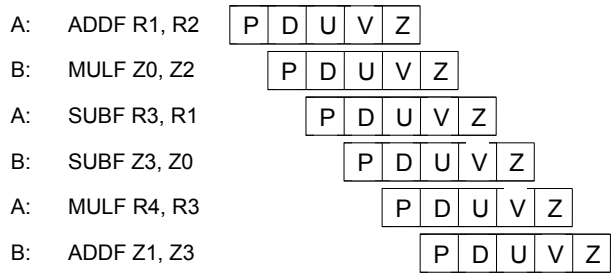
- Przełączanie wątków następuje przy dłuższym opóźnieniu wątku w potoku (np. chybione odwołanie do pamięci podręcznej (nawet L2))
- W niektórych rozwiązaniach rozpoczęcie nowego wątku następuje dopiero po opróżnieniu potoku
- Zaletą jest prostota procesu przełączania wątków
- Wadą takiego rozwiązania są straty czasu przy krótszych opóźnieniach potoku



7.4 Wielowątkowość drobnoziarnista

Fine-grained multithreading.

- Przełączanie wątków następuje po każdym rozkazie
- Wątek oczekujący (np. na dostęp do pamięci) jest pomijany
- Zaletą jest unikanie strat nawet przy krótkich opóźnieniach wątków
- Istotnym wymaganiem dla procesora jest szybkie (w każdym takcie) przełączanie wątków
- Pewną wadą jest opóźnienie realizacji wątków w pełni gotowych do wykonania



7.5 Warunki sprzętowej realizacji wielowątkowości

- powielenie zestawów rejestrów uniwersalnych (lub powielenie tabel mapowania rejestrów)
- powielenie liczników rozkazów
- powielenie układów dostępu do pamięci podręcznej (tabel stron)
- powielenie sterowników przerwań

7.6 Wielowątkowość w procesorze dwupotokowym

Reguły realizacji i przełączania wątków:

1. **Wielowątkowość gruboziarnista**
   - wątek realizowany w kolejnych taktach do momentu wstrzymania rozkazu
   - do obu potoków wprowadzane są rozkazy tylko jednego wątku (w jednym takcie!)
2. **Wielowątkowość drobnoziarnista**
   - w kolejnych taktach realizowane są naprzemiennie rozkazy kolejnych wątków (przełączanie wątków co takt)
   - do obu potoków wprowadzane są rozkazy tylko jednego wątku (w jednym takcie!)
3. **Wielowątkowość współbieżna** (SMT -Simultaneous multithreading)
   - wątek realizowany do momentu wstrzymania rozkazu
   - do obu potoków w jednym takcie mogą być wprowadzane rozkazy różnych wątków

7.7 Mankamenty współbieżnej wielowątkowości

- Rywalizacja wątków w dostępie do pamięci podręcznej - mniejsza wielkość PaP przypadająca na wątek
- Większe zużycie energii (w porównaniu z procesorami dwurdzeniowymi)
- Możliwość monitorowanie wykonania jednego wątku przez inny wątek (złośliwy), poprzez wpływ na współdzielone dane pamięci podręcznej - kradzież kluczy kryptograficznych

8 Klasyfikacja komputerów równoległych

8.1 Formy równoległości w architekturze komputerów

8.1.1 Równoległość na poziomie rozkazów

Wykonywanie w danej chwili wielu rozkazów w jednym procesorze.

- Mechanizmy potokowe - w procesorach CISC i RISC
- Architektura superskalarna i VLIW

8.1.2 Równoległość na poziomie procesorów

Wykonywanie w danej chwili wielu rozkazów w wielu procesorach.

- Komputery wektorowe
- Komputery macierzowe
- Systemy wieloprocesorowe
- Klastry (systemy wielokomputerowe)

8.2 Rodzaje równoległości w aplikacjach

8.2.1 Równoległość poziomu danych

DLP - Data Level Parallelism.
Pojawia się kiedy istnieje wiele danych, które mogą być przetwarzane w tym samym czasie.

8.2.2 Rówoległość poziomu zadań

TLP - Task Level Parallelism.
Pojawia się kiedy są tworzone zadania, które mogą być wykonywane niezależnie i w większości równolegle.

8.3 Drogi wykorzystania równoległości aplikacji w architekturze komputerów

- **Równoległość poziomu rozkazów** (ILP - Instruction Level Parallelism) - odnosi się do przetwarzania potokowego i superskalarnego, w których w pewnym (niewielkim) stopniu wykorzystuje się równoległość danych.
- **Architektury wektorowe i procesory graficzne** - wykorzystują równoległość danych poprzez równoległe wykonanie pojedynczego rozkazu na zestawie danych.
- **Równoległość poziomu wątków** (TLP - Thread Level Parallelism) - odnosi się do wykorzystania równoległości danych albo równoległości zadań w ściśle połączonych systemach (ze wspólną pamięcią), które dopuszczają interakcje między wątkami.
- **Równoległość poziomu zleceń** (RLP - Request Level Parallelism) - odnosi się do równoległości zadań określonych przez programistę lub system operacyjny. Ta forma równoległości jest wykorzystywana w systemach luźno połączonych (z pamięcią rozproszoną) i klastrach.

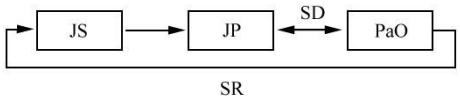
8.4 Klasyfikacja Flynna

M. Flynn, 1966

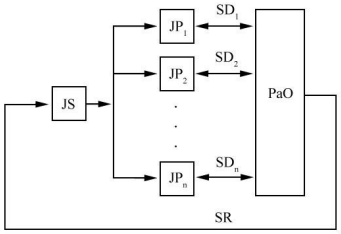
8.4.1 Kryterium klasyfikacji

Liczba strumieni rozkazów i liczba strumieni danych w systemie komputerowym.

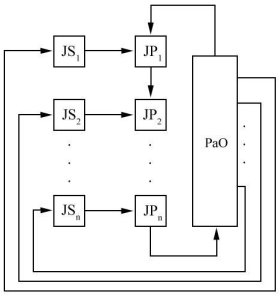
- **SISD**: Single Instruction, Single Data Stream



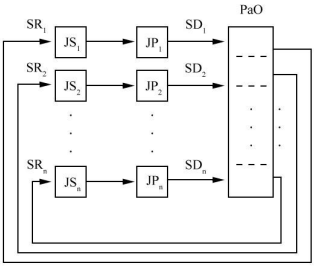
- **SIMD**: Single Instruction, Multiple Data Stream



- **MISD**: Multiple Instruction, Single Data Stream



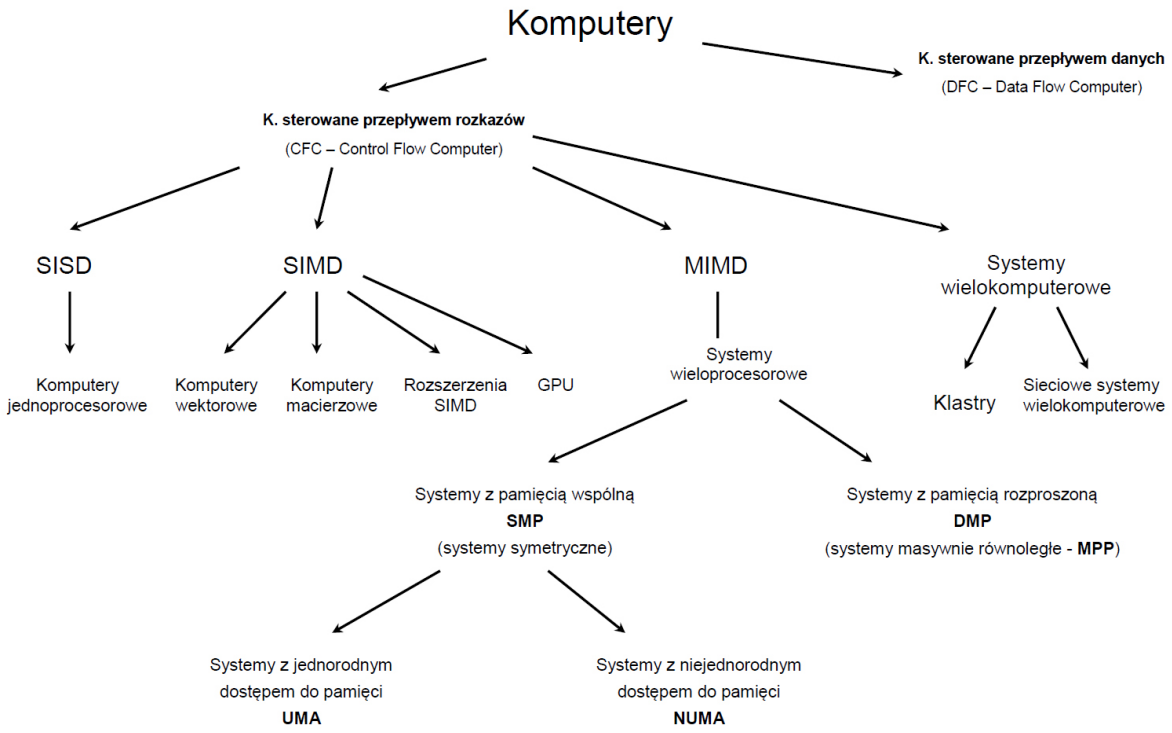
- **MIMD**: Multiple Instruction, Multiple Data Stream



Gdzie:

- **JS** – Jednostka sterująca
- **JP** – Jednostka przetwarzająca
- **PaO** – Pamięć operacyjna

8.4.2 Klasyfikacja opisowa



9 Architektura SIMD

9.1 Co to jest?

- Tłumaczenie *Single Instruction Multiple Device*

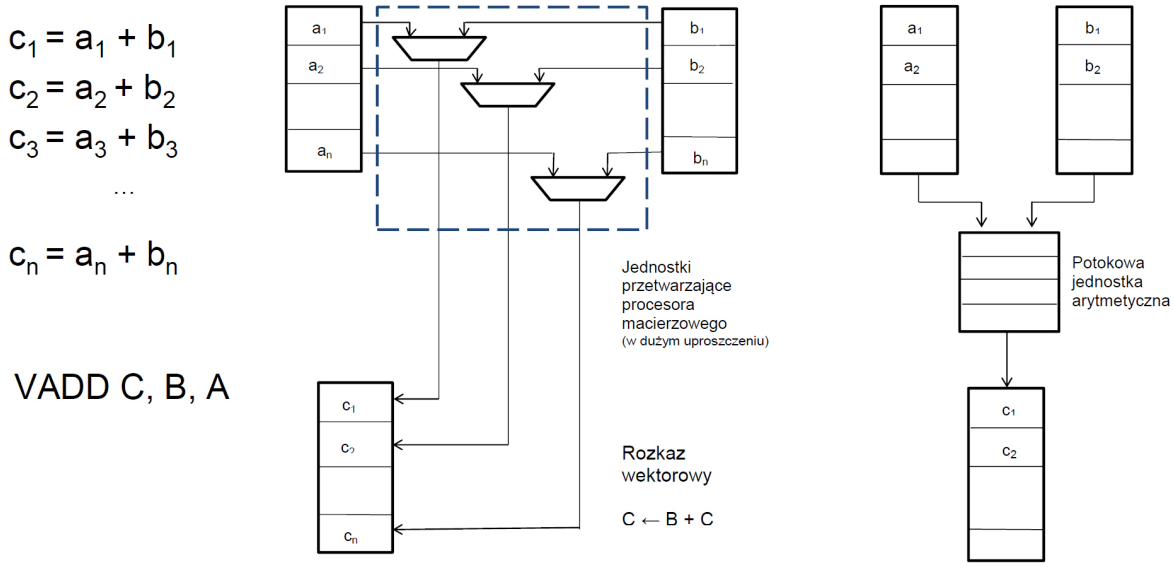
- Cecha wyróżniająca dla programisty - rozkazy wektorowe (rozkazy z argumentami wektorowymi).

- Dwa różne podejścia do sprzętowej realizacji rozkazów wektorowych:

  – Komputery (procesory) macierzowe

  – Komputery wektorowe

Idee realizacji obu (macierzowy i wektorowy):



9.2 Komputery wektorowe

9.2.1 Lokalizacja wektorów danych

- Pamięć operacyjna (STAR 100)
- Rejestry wektorowe (Cray -1)

9.2.2 Przykład rozkazu

Rozkaz dodawania wektorów: VADDF A,B,C,n
Czas wykonania:

$$t_w = t_{start} + (n-1) \times \tau$$

W komputerze macierzowym czas wykonywania tego rozkazu jest równy *const*.

9.2.3 Przyspieszenie

Przyspieszenie jest stosunkiem czasu wykonywania w komputerze klasycznym (szeregowo) do czasu wykonywania w komputerze wektorowym.

$$a = lim_{n\to\infty} \frac{15 \times \tau \times n}{t_{start} + (n-1) \times \tau} = 15$$

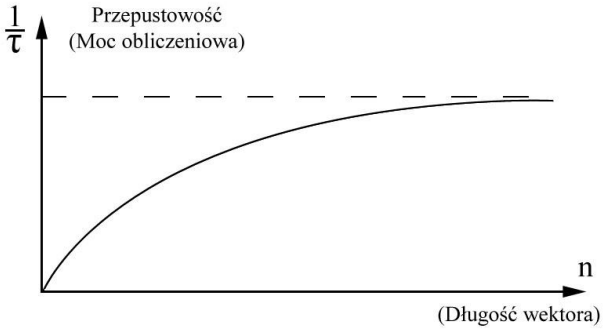
9.2.4 Przepustowość

Przepustowość (moc obliczeniowa) jest stosunkiem ilości operacji zmiennoprzecinkowych do czasu ich wykonania.

$$Przep = lim_{n\to\infty} \frac{n}{t_{start} + (n-1) \times \tau} = \frac{1}{\tau}$$

Wymiarem przepustowości jest FLOPS.

9.2.5 Zależność mocy obliczeniowej od długości wektora



9.2.6 Podsumowanie

1. Hardware
    - rozkazy wektorowe
    - duża liczba potokowych jednostek arytmetycznych (specjalizowanych)
    - duża liczba rejestrów (nawet kilkaset tysięcy)
2. Software
    - klasyczne języki: Fortran, C
    - klasyczne algorytmy
    - kompilatory wektoryzujące

9.2.7 Zastosowanie

- Numeryczna symulacja ośrodków ciągłych
- Równania różniczkowe, równania różnicowe, układy równań algebraicznych (rachunek macierzowy)
- Dziedziny zastosowań:
    - prognozowanie pogody
    - symulacja aerodynamiczna
    - sejsmiczne poszukiwania ropy naftowej i innych surowców
    - symulacja reakcji jądrowych
    - medycyna i farmacja
    - obliczenia inżynierskie dużej skali

9.3 Komputery macierzowe

9.3.1 Co to jest?

Architektura komputerów macierzowych - model SIMD w dwóch wariantach:

- SIMD - DM (z pamięcią rozproszoną)
- SIMD - SM (z pamięcią wspólną)

9.3.2 Elementy komputera macierzowego

1. **Jednostka sterująca** - procesor wykonujący rozkazy sterujące i skalarne oraz inicjujący wykonanie rozkazów wektorowych w sieci elementów przetwarzających.
2. **Elementy przetwarzające** (procesorowe) - jednostki arytmetyczno-logiczne wykonujące operacje elementarne rozkazów wektorowych.
3. **Sieć łącząca** - łączy elementy przetwarzające między sobą lub z modułami pamięci operacyjnej; warianty:
   - sieć statyczna: pierścień, gwiazda, krata, drzewo, hipersześcian
   - sieć dynamiczna: jednostopniowa; wielostopniowa (wyróżnia połączenia blokujące i nieblokujące)

9.3.3 Podsumowanie

- Architektura SIMD
- Jednostka sterująca + jednostka macierzowa
- Rozkazy wektorowe - wykonywane synchronicznie w sieci (macierzy) EP
- Skomplikowana wymiana danych między EP
- Trudne programowanie - konieczność tworzenia nowych wersji algorytmów

9.4 Model SIMD w procesorach superskalarnych

9.4.1 Technologia MMX

- 8 rejestrów 64-bitowych MMX
- Nowe typy danych
- Rozszerzony zestaw instrukcji (57 instrukcji)
- Realizacja operacji na krótkich wektorach wg modelu SIMD

9.5 Technologia SSE

- 8 rejestrów 128-bitowych
- Osiem 16-bitowych argumentów (elementów wektora) typu integer
- Cztery 32-bitowe argumenty integer/fplub dwa 64-bitowe
- Operacje zmp na 4-elementowych wektorach liczb 32-bit (pojed. prec.)

10 Karty graficzne i architektura CUDA

10.1 Charakterystyka

- GPU - Graphics Processing Unit
- Wcześniejsze GPU - specjalizowane języki (HLSL, GLSL czy NVIDIA Cg), tylko rendering
- CUDA (Compute Unified Device Architecture) -architektura wielordzeniowych procesorów graficznych (GPU)
- Uniwersalna architektura obliczeniowa połączona z równoległym modelem programistycznym
- wsparcie dla języków C/C++

- GPGPU = GPU + CUDA
- CUDA - obsługiwana przez karty graficzne GeForce i GeForce Mobile od serii 8 (GeForce 8800), nowsze układy z rodzin Tesla i Quadro, Fermi, obecnie Kepler

10.2 Architektura CUDA

- W miejsce oddzielnych potoków przetwarzających wierzchołki i piksele wprowadzenie uniwersalnego procesora przetwarzającego wierzchołki, piksele i ogólnie geometrię, a także uniwersalne programy obliczeniowe
- Wprowadzenie procesora wątków eliminującego „ręczne" zarządzanie rejestrami wektorowymi
- Wprowadzenie modelu SIMT (single-instruction multiple-thread), w którym wiele niezależnych wątków wykonuje równocześnie tę samą instrukcję
- Wprowadzenie współdzielonej pamięci oraz mechanizmów synchronizacji wątków (barrier synchronization) dla komunikacji między wątkami

10.3 Multiprocesor strumieniowy

Architektura GT 200.

- 8 rdzeni C1 -C8 (SP)
- podręczna pamięć instrukcji (ang. instruction cache),
- podręczna pamięć danych (ang. constant cache) - pamięć tylko do odczytu,
- pamięć współdzielona (ang. shared memory)
- 16 384 rejestry,
- jednostka arytmetyczna wykonująca obliczenia zmiennoprzecinkowe podwójnej precyzji (fp64),
- dwie jednostki arytmetyczne przeznaczone do obliczania funkcji specjalnych (ang. special function unit),
- pamięć globalna

10.4 Model programistyczny CUDA

- Specjalny kompilator NVCC
- Podział programu na kod wykonywany przez procesor (ang. *Host code*) i przez urządzenie (kartę graficzną) (ang. Device code) - kernel
- Realizacja operacji równoległych według modelu SIMT (*Single Instruction Multiple Threading*)

10.5 Wykonanie obliczeń z użyciem architektury CUDA (5 faz)

1. Przydzielenie w pamięci globalnej obszaru pamięci dla danych, na których będą wykonywane obliczenia przez kernel.
2. Przekopiowanie danych do przydzielonego obszaru pamięci.
3. Zainicjowanie przez CPU obliczeń wykonywanych przez GPU, tj. wywołanie kernela.
4. Wykonanie przez wątki (z użyciem GPU) obliczeń zdefiniowanych w kernelu.
5. Przekopiowanie danych z pamięci globalnej do pamięci operacyjnej.

10.6 CUDA procesor (rdzeń)

- Potokowa jednostka arytmetyczna zmp
- Potokowa jednostka arytmetyczna stp
- Ulepszona realizacja operacji zmp FMA (fused multiply-add) dla pojedynczej i podwójnej precyzji

11 Wątki

11.1 Co to jest?

- Wątek reprezentuje pojedynczą operację (a single work unit or operation)
- Wątki są automatycznie grupowane w bloki, maksymalny rozmiar bloku = 512 wątków (w architekturze Fermi i wyższych –1024 wątki).
- Bloki grupowane są w siatkę (grid -kratę)
- Grupowanie wątków –bloki o geometrii 1, 2 lub 3-wymiarowej
- Grupowanie bloków –siatka (grid) o geometrii 1, 2-wymiarowej
- Wymaga się, aby bloki wątków tworzących siatkę mogły się wykonywać niezależnie: musi być możliwe ich wykonanie w dowolnym porządku, równolegle lub szeregowo.

11.2 Grupowanie wątków w bloki i siatkę

- Siatka o geometrii jednowymiarowej (trzy bloki wątków)
- Każdy blok -geometria dwuwymiarowa (wymiary 2 x 3)

11.3 Sprzętowa organizacja wykonywania wątków

- Przy uruchomieniu *kernela* wszystkie bloki tworzące jego siatkę obliczeń są rozdzielane pomiędzy multiprocesory danego GPU
- Wszystkie wątki danego bloku są przetwarzane w tym samym multiprocesorze
- W danej chwili (cyklu) pojedynczy rdzeń multiprocesora wykonuje jeden wątek programu
- Multiprocesor tworzy, zarządza, szereguje i wykonuje wątki w grupach po 32, nazywanych wiązkami (*warp*).
- Wiązki są szeregowane do wykonania przez *warp scheduler*. Wiązka wątków jest wykonywana jako jeden wspólny rozkaz (analogia do rozkazu SIMD, tzn. rozkazu wektorowego)
- Sposób wykonania wiązki wątków (rozkazu SIMD) zależy od budowy multiprocesora:
  - Dla architektury Fermi (32 procesory w jednym multiprocesorze / 2 *warp-schedulery* = 16 procesorów na 1 wiązkę) wiązka jest wykonywana jako 2 rozkazy - wiązka jest dzielona na dwie połówki (*half–warp*) wykonywane jako 2 rozkazy (te same, ale na dwóch zestawach danych).
  - Dla architektury Tesla (8 procesorów w jednym multiprocesorze, 1 *warp-scheduler*) wiązka jest dzielona na cztery ćwiartki (*quarter-warp*) wykonywane jako 4 kolejne rozkazy (te same, ale na czterech zestawach danych).
- Konstrukcja *warp schedulera* umożliwia uruchomienie wielu wiązek wątków współbieżnie - *warp scheduler* pamięta wtedy adresy wiązek, przypisane im rozkazy SIMD oraz ich stan (gotowość do wykonania lub stan oczekiwania na pobranie danych z pamięci).
- Współbieżne uruchomienie wielu wiązek pozwala zmniejszyć straty związane z oczekiwaniem na dane (zwykle długi czas dostępu do pamięci).

12 Rodzaje pamięci multiprocesora

12.1 Pamięć globalna

Duża pamięć, o czasie życia aplikacji (dane umieszczone w tej pamięci są usuwane po zakończeniu aplikacji), dostępna dla każdego wątku w dowolnym bloku, ale o dość długim czasie dostępu wynoszącym ok. 400-600 taktów zegara.

12.2 Pamięć współdzielona

Niewielka pamięć o czasie życia bloku (zakończenie działania bloku powoduje usunięcie danych w niej przechowywanych), dostępna dla każdego wątku w bloku dla którego jest dedykowana, o bardzo krótkim czasie dostępu.

12.3 Pamięć stałych

Niewielki fragment pamięci globalnej, który jest cache-owany, przez co dostęp do niego jest bardzo szybki. Jest ona tylko do odczytu. Czas życia pamięci stałych oraz jej dostępność jest taka sama jak pamięci globalnej.

12.4 Rejestry

Niewielka, bardzo szybka pamięć o czasie życia wątku (po zakończeniu wątku dane z rejestrów są usuwane). Tylko jeden wątek może w danym momencie korzystać z danego rejestru.

12.5 Pamięć lokalna i pamięć tekstur

Podobnie jak w przypadku pamięci stałych, są to dedykowane fragmenty pamięci globalnej. Pamięć lokalna jest wykorzystywana do przechowywania danych lokalnych wątku, które nie mieszczą się w rejestrach, a pamięć tekstur posiada specyficzne metody adresowania i cache-owanie specyficzne dla zastosowań graficznych.

13 Systemy wieloprocesorowe (UMA)

13.1 Rodzaje

- Systemy z pamięcią wspólną
- Systemy z pamięcią rozproszoną

13.2 Systemy z pamięcią wspólną

- Systemy z jednorodnym dostępem do pamięci (UMA – *Uniform Memory Access*)
- Systemy z niejednorodnym dostępem do pamięci (NUMA – Non - *Uniform Memory Access*)

13.2.1 Klasyfikacja

- Systemy ze wspólną magistralą
- Systemy wielomagistralowe
- Systemy z przełącznicą krzyżową
- Systemy z wielostopniową siecią połączeń
- Systemy z pamięcią wieloportową
- Systemy z sieciami typu punkt - punkt

13.3 Skalowalność

System skalowalny - System, w którym dodanie pewnej liczby procesorów prowadzi do proporcjonalnego przyrostu mocy obliczeniowej.

13.4 Systemy ze wspólną magistralą

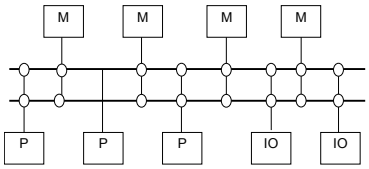
- Prostota konstrukcji – niska złożoność układowa całości
- Niski koszt
- Łatwość rozbudowy – dołączenia kolejnego procesora, ale tylko w ograniczonym zakresie
- Ograniczona złożoność magistrali (jej szybkość jest barierą)
- Niska skalowalność

13.4.1 Protokół MESI

Ten rodzaj systemu posiada problem zapewnienia spójności pamięci podręcznych (*snooping*). W celu rozwiązania go wykorzystuje w tym celu protokół MESI:

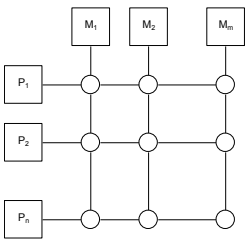
- I - invalid
- S - shared
- E - exclusive
- M - modified

13.5 Systemy wielomagistralowe



- Wielokrotnie zwiększona przepustowość
- Konieczność stosowania układu arbitra do sterowania dostępem do magistral
- Rozwiązania kosztowne

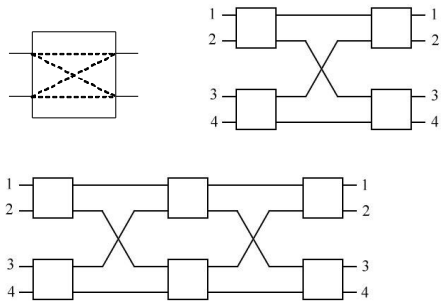
13.6 Systemy z przełącznicą krzyżową



- Zadania każdego przełącznika
  - Rozwiązywanie konfliktów dostępu do tego samego modułu pamięci
  - Zapewnienie obsługi równoległych transmisji, krzyżujących się w przełączniku
- Duża przepustowość
- Duża złożoność układowa $O(n^2)$

- Wysoki koszt
- Problem: realizacja techniczna przełącznicy krzyżowej dla dużych n – wykorzystanie wielostopniowych sieci połączeń

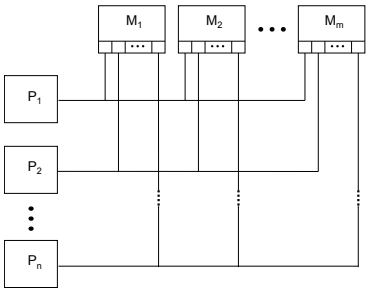
13.7 Systemy z wielostopniową siecią połączeń



Przykłady:

- Nieblokująca sieć Closa
- Sieć wielostopniowa Benesa
- Sieć wielostopniowa typu Omega

13.8 Systemy z pamięcią wieloportową



- Każdy procesor ma niezależny dostęp do modułów pamięci – poprawa wydajności
- Układ sterowania modułu pamięci rozstrzyga konflikty dostępu – większa złożoność
- Możliwość skonfigurowania części pamięci jako prywatnej dla jednego lub kilku procesorów
- Stosowana technika zapisu do pamięci cache – *write through* – poprzez własną magistralę i port w module
- Ograniczona liczba portów w module

13.9 Systemy z sieciami typu punkt-punkt

13.9.1 HyperTransport (HT)

- Technologia wprowadzona w 2001 przez HyperTransportConsortium(AMD, Apple, Cisco, Nvidia, Sun i in.)
- Przeznaczona do łączenia procesorów, pamięci i układów we/wy -technologia HT zastąpiła wspólną magistralę
- Dla łączenia wielu procesorów stosowana razem z techniką NUMA

- Topologia punkt -punkt
- Sieć dwukierunkowa, szeregowo/równoległa, o wysokiej przepustowości, małych opóźnieniach,
- Łącza o szerokości 2, 4, 8, 16, or 32 bity. Teoretyczna przepustowość: 25.6 GB/s (3.2GHz x 2 transfers per clock cycle x 32 bits per link) dla jednego kierunku lub 51.2 GB//s łącznie dla obu kierunków transmisji
- Transmisja pakietów składających się z 32-bitowych słów

13.9.2 Intel QuickPath Interconnect (QPI)

- Następca Front-SideBus (FSB) -magistrali łączącej procesor z kontrolerem pamięci
- Tworzy bardzo szybkie połączenia między procesorami i pamięcią oraz procesorami i hubami we/wy
- Tworzy mechanizm „scalablesharedmemory" (NUMA) –zamiast wspólnej pamięci dostępnej przez FSB, każdy Procesor ma własną dedykowaną pamięcdostępną przez IntegratedMemory Controleroraz możliwość dostępu do dedykowanej pamięci innych procesorów poprzez QPI
- Podstawową zaletą QPI jest realizacja połączeń punkt-punkt (zamiast dostępu przez wspólną magistralę)

13.10 Podsumowanie

- Symetryczna architektura – jednakowy dostęp procesorów do pamięci operacyjnej oraz we/wy
- Utrzymanie spójności pamięci podręcznych (cache):
  - *snooping* - metoda starsza i mało skalowalna (głównie w systemach ze wspólną magistralą)
  - katalog - metoda lepiej skalowalna, stosowana razem z sieciami typu punkt - punkt
- Łatwe programowanie (realizacja algorytmów równoległych)
- Niska skalowalność:
  - Mechanizm *snoopingu*, zapewniający spójność pamięci podręcznych węzłów systemów UMA nie jest skalowalny dla bardzo dużej liczby węzłów.
  - W systemach z dużą liczbą węzłów, posiadających lokalną pamięć, dla zapewnienia spójności pamięci podręcznych jest stosowane rozwiązanie oparte na katalogu.

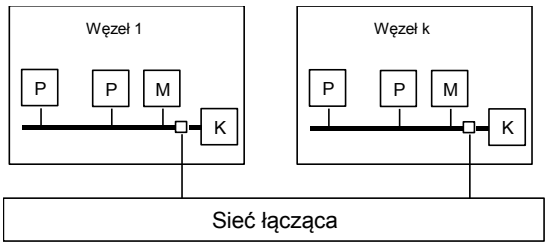
14 Systemy NUMA

Systemy wieloprocesorowe z niejednorodnym dostępem do pamięci.
NUMA (*Non-Uniform Memory Access*).

14.1 Rodzaje

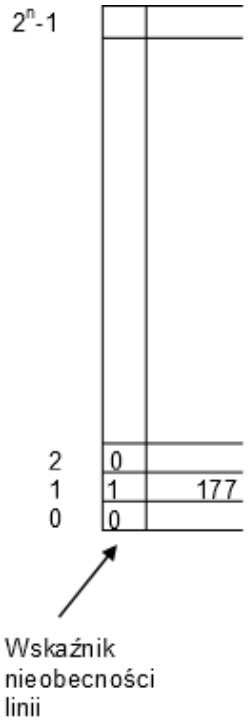
- NC-NUMA (*Non-cached NUMA*)
  - Odwołania do nielokalnej PaO przekierowywane do odległych węzłów
  - Odwołania do nielokalnej PaO wolniejsze ok. 10 razy
- CC-NUMA (*Cache Coherent NUMA*)
  - Wykorzystuje węzły i katalogi

14.2 Węzeł



14.3 Katalog

14.3.1 Najprostsza postać katalogu



14.3.2 Rozmiar katalogu

- dla PaO węzła = 1 GB i linii = 128 B katalog musiałby mieć 2^{23} pozycji
- dla PaO węzła = 8 GB i linii = 128 B katalog musiałby mieć 2^{26} pozycji
- dlatego katalog praktycznie jest zazwyczaj realizowany jako pamięć asocjacyjna, o znacznie mniejszych rozmiarach

14.3.3 Różne warianty organizacji katalogu

Zawartość pozycji katalogu:

1. numer węzła aktualnie posiadającego daną linię
2. k numerów węzłów aktualnie posiadających daną linię
3. n bitów (1 bit na węzeł) wskazujących posiadanie linii przez węzeł
4. element listy pól z numerami węzłów aktualnie posiadających daną linię

14.4 System DASH

14.4.1 Co to jest?

- Pierwszy system CC-NUMA wykorzystujący katalog
- 16 węzłów połączonych kratą
- Węzeł: 4 procesory MIPS R3000 + 16 MB RAM + katalog
- Linia PAP = 16 B
- Katalog = 1 M wierszy 18-bitowych
  - Wiersz katalogu= 16 bitów obecności linii w węzłach + stan linii
  - **Stan linii**: uncached, shared, modified

14.4.2 Interpretacja stanu linii

- **uncached** – linia pamięci jest tylko w pamięci lokalnej (domowej)
- **shared** – linia PaP została przesłana do odczytu do kilku węzłów (ich pamięci lokalnych)
- **modified** – linia PaP jest w pamięci domowej nieaktualna (została zmodyfikowana w innym węźle)

14.4.3 Operacja odczytu

- Stan żądanej linii = *uncached* lub *shared* → linia jest przesyłana do węzła żądającego; stan linii := *shared*.
- Stan żądanej linii = *modified* → sterownik katalogu domowego żądanej linii przekazuje żądanie do węzła x posiadającego linię. Sterownik katalogu węzła x przesyła linię do węzła żądającego oraz węzła domowego tej linii; stan linii := *shared*.

14.4.4 Operacja zapisu

- Przed zapisem węzeł żądający linii musi być jedynym jej posiadaczem.
- Węzeł żądający posiada linię
  - stan linii = *modified* → zapis jest wykonywany
  - stan linii = *shared* → węzeł przesyła do domowego katalogu tej linii żądanie unieważnienia innych kopii linii; stan linii := *modified*
- Węzeł żądający nie posiada linii → wysyła żądanie dostarczenia linii do zapisu
  - stan linii = *uncached* → linia jest przesyłana do węzła żądającego; stan linii := *modified*
  - stan linii = *shared* → wszystkie kopie linii są unieważniane, potem jak dla *uncached*
  - stan linii = *modified* → przekierowanie żądania do węzła x posiadającego linię. Węzeł x unieważnia ją u siebie i przesyła żądającemu.

14.4.5 Katalog czy snooping

Mechanizm spójności oparty o katalog jest bardziej skalowalny od mechanizmu „snooping", ponieważ wysyła się w nim bezpośrednią prośbę i komunikaty unieważniające do tych węzłów, które mają kopie linii, podczas gdy mechanizm „snooping" rozgłasza (*broadcast*) wszystkie prośby i unieważnienia do wszystkich węzłów.

14.5 Podsumowanie

- PaO fizycznie rozproszona, ale logicznie wspólna
- Niejednorodny dostęp do pamięci - PaO lokalna, PaO zdalna
- Utrzymanie spójności pamięci podręcznych (cache) - katalog
- Hierarchiczna organizacja: procesor – węzeł (system UMA) – system NUMA
- Zalety modelu wspólnej pamięci dla programowania
- Dobra efektywność dla aplikacji o dominujących odczytach z nielokalnej pamięci
- Gorsza efektywność dla aplikacji o dominujących zapisach do nielokalnej pamięci
- Skalowalność: 1024 – 2560 rdzeni

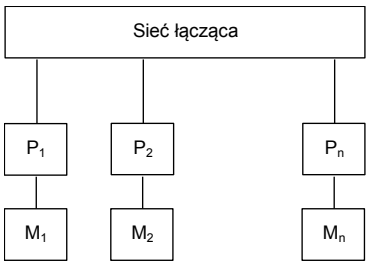
15 Systemy SMP - podsumowanie

- Symetryczna architektura – jednakowy dostęp procesorów do pamięci operacyjnej oraz we/wy (na poziomie fizycznych przesyłów – tylko w systemach wieloprocesorowe z pamięcią wspólną fizycznie - UMA)
- Utrzymanie spójności pamięci podręcznych (cache):
  - systemy UMA - snooping lub katalog
  - systemy NUMA - katalog
- Łatwe programowanie (realizacja algorytmów równoległych)
- Niska (UMA) i średnia (NUMA) skalowalność

16 Systemy MMP

Systemy wieloprocesorowe z pamięcią rozproszoną.
MPP – *Massively Parallel Processors*.

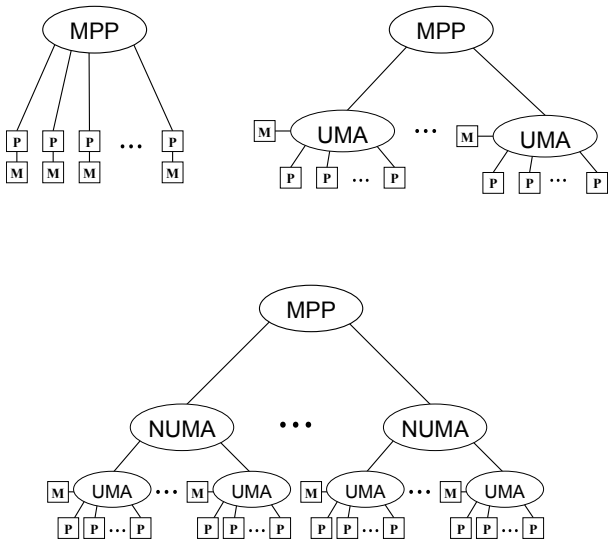
16.1 Uproszczona organizacja



16.2 Hierarchiczna organizacja

16.2.1 Rodzaje węzłów

- Procesor + PaO
- System wieloprocesorowy z pamięcią wspólną UMA
- System wieloprocesorowy NUMA (z niejednorodnym dostępem do pamięci)



16.3 Topologia

Sieci łączące węzły MPP.

- Hipersześcian
- Krata 2D, 3D
- Torus 2D, 3D
- Przełącznica krzyżowa (hierarchiczne przełącznice krzyżowe)
- Sieci wielostopniowe (Omega, Butterfly, grube drzewo, Dragonfly i inne)
- Sieci specjalizowane (*proprietary / custom network*)

16.4 Obsługa przesyłu komunikatów

Obsługa przesyłu komunikatów w węzłach systemu wieloprocesorowego.

- Programowa obsługa przesyłu przez procesory węzłów pośredniczących (systemy I generacji)
- Sprzętowa obsługa przesyłu przez routery węzłów pośredniczących, bez angażowania procesorów (systemy II generacji)

16.5 Narzędzia programowe

Narzędzia programowe wspierające budowę programów z przesyłem komunikatów:

- PVM (Parallel Virtual Machine)
- MPI (Message Passing Interface)
- Inne (Cray SHMEM, PGAS)

16.6 Podsumowanie

- Hierarchiczna architektura
- Węzeł: procesor, system UMA, (system NUMA)
- Bardzo duża skalowalność
- Wolniejsza (na ogół) komunikacja – przesył komunikatów
- Dedykowane, bardzo szybkie, sieci łączące

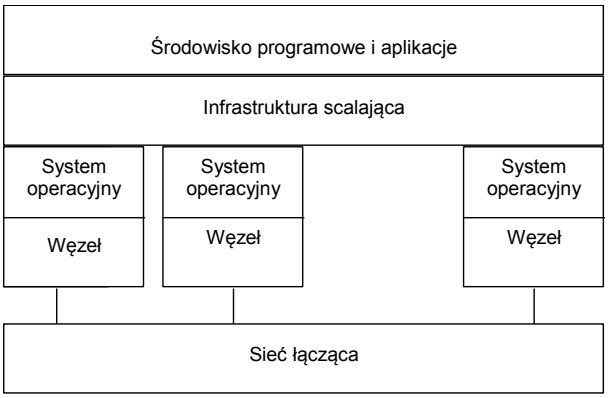
17 Klastry

Klaster (komputerowy) (ang. *cluster*).

17.1 Definicja

- Klaster to zestaw pełnych komputerów (węzłów) połączonych razem, pracujących jako jeden system.
- Wikipedia:
  Klaster komputerowy -grupa połączonych jednostek komputerowych, które współpracują ze sobą w celu udostępnienia zintegrowanego środowiska pracy
- A computer cluster consists of a set of loosely connected computers that work together so that in many respects they can be viewed as a single system.

17.2 Ogólna struktura systemów typu klaster



17.3 Ogólna charakterystyka klastrów

17.3.1 Węzły

- Serwery SMP
- Pełne komputery: PC, stacje robocze

17.3.2 System operacyjny

- Linux
- Unix
- Windows

17.3.3 Infrastruktura scalająca

- MPI (*Message Passing Interface*)
- PVM (*Parallel Virtual Machine*)
- SSI (*Single System Image*)

17.3.4 Komunikacja między węzłami

- Przesył komunikatów

17.3.5 Sieci łączące

- Sieci specjalizowane – starsze rozwiązania
- Sieci LAN (*Local Area Network*)
- Sieci specjalizowane -nowsze rozwiązania

17.3.6 Cele budowy klastrów

- Wysoka wydajność (klastry wydajnościowe, klastry obliczeniowe) (*High-performace clusters*)
- Wysoka niezawodność (klastry niezawodnościowe) (*High-availability clusters*)
- Równoważenie obciążenia (*Load balancing clusters*)

17.3.7 Inne

- Zależność moc obliczeniowa –niezawodność
- Korzystny wskaźnik wydajność -cena

17.4 Sieci łączące klastrów

17.4.1 Sieci specjalizowane - starsze rozwiązania

- HiPPI (*High Performance Parallel Interface*) – pierwszy standard sieci "near-gigabit" (0.8 Gbit/s) - kabel 50-par (tylko superkomputery)
- Memory Channel

17.4.2 Sieci LAN

- Ethernet
- Fast Ethernet
- Gigabit Ethernet

17.4.3 Sieci specjalizowane - nowsze rozwiązania

Systemowe SAN.

- Myrinet
- Quadrics (QsNet)
- SCI (Scalable Coherent Interface)
- InfiniBand

17.5 Fibre Channel

- Technologia sieciowa o dużej przepustowości (16 Gb/s) stosowana zwykle do łączenia komputera z pamięcią zewnętrzną
- Standard magistrali szeregowej definiujący wielowarstwową architekturę
- Powstał w 1988 jako uproszczona wersja HIPPI
- Łącza światłowodowe i miedziane
- Głównym stosowanym protokołem jest SCSI, ponadto ATM, TCP/IP

17.6 Sieci łączące -różnice

- Parametry – przepustowość, czas opóźnienia
- Topologia
  - Ethernet – magistrala, gwiazda, struktury hierarchiczne
  - Sieci specjalizowane – sieć przełączników (*switched fabric*) – popularna topologia „grubego drzewa"

17.7 Klastry o wysokiej niezawodności

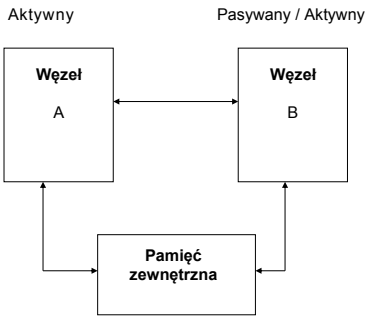
17.7.1 Czynniki tworzące wysoką niezawodność klastrów

1. Redundancja węzłów (mocy obliczeniowej)
2. Dostęp do wspólnych zasobów (pamięci zewnętrznych)
3. Mirroring dysków
4. Mechanizmy kontrolujące funkcjonowanie węzłów
5. Redundancja sieci łączących (dla 3 rodzajów sieci)
6. Redundancja zasilania

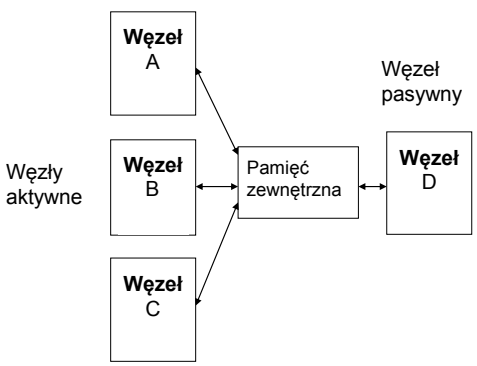
17.7.2 Redundancja węzłów / mocy obliczeniowej

Tryby pracy węzłów:

- Model klastra „aktywny - pasywny"
- Model klastra „aktywny - aktywny"



- Modele mieszane

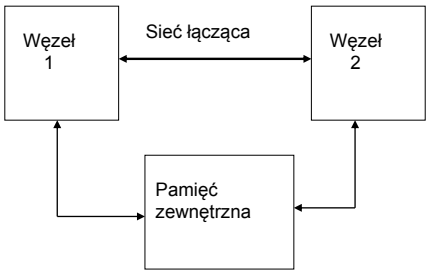


17.7.3 Warianty modelu mieszanego

- *N + 1* – jeden węzeł dodatkowy, o uniwersalnych możliwościach zastąpienia każdego z pozostałych.
- *N + M* – zwiększenie liczby dodatkowych węzłów do M w celu zwiększenia redundancji w przypadku dużej różnorodności usług świadczonych przez węzły.
- *N – to – N* – kombinacja modeli „aktywny -aktywny" oraz „N + M" –redystrybucja zadań węzła uszkodzonego na kilka innych węzłów aktywnych

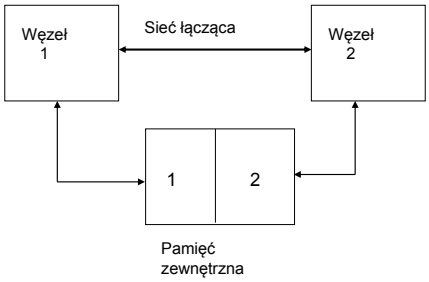
17.8 Warianty dostępu do wspólnych zasobów

17.8.1 Zasada „współdziel wszystko"



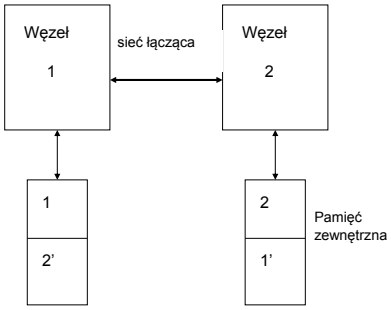
Wszystkie węzły mają dostęp do wspólnej pamięci zewnętrznej (kontrola dostępu przez mechanizmy blokad).

17.8.2 Zasada „nie współdziel nic"



Węzły nie współdzielą tego samego obszaru dysku –każdy ma dostęp do własnej części; po awarii czynny węzeł otrzymuje prawo dostępu do całego dysku.

17.8.3 Mirroring



Każdy węzeł zapisuje dane na własny dysk i automatycznie (pod kontrolą odpowiedniego oprogramowania) jest tworzona kopia tego zapisu na dyskach innych węzłów.

17.9 Podsumowanie

- Węzły – typowe serwery SMP „z półki” + pełna instancja Systemu Operacyjnego
- Sieci łączące – standardy: Gigabit Ethernet, Infiniband
- Komunikacja między węzłami (procesami) – przesył komunikatów
- Bardzo wysoka skalowalność
- Cele budowy: wysoka wydajność lub/i wysoka niezawodność, równoważenie obciążenia
- Korzystny wskaźnik: cena/wydajność