OWS1
(systemy z pamięcią współdzieloną)

Rafał Walkowiak

Wersja: wiosna 2017

Wewnętrzna współbieżność przetwarzania procesora

- Uwarunkowania:
  1. Dotychczas imponujący wzrost prędkości taktowania procesora 1000 razy szybciej w ciągu 10 lat – obecnie ograniczony wielkością wydzielanej mocy cieplnej i fizycznymi wymiarami elementów struktury
  2. Ograniczenia we wzroście prędkości pracy pamięci (memory wall)
  3. Technologiczne możliwy wzrost złożoności układów scalonych - podwojenie złozoności w ciągu 18 miesięcy – prawo Gordona Moora
- Jak wykorzystać możliwości dla wzrostu efektywności przetwarzania ?
  - Równoczesna realizacja wielu instrukcji w czasie jednego cyklu zegarowego - potokowość, wiele potoków - superskalarność
  - Równoległość pracy modułów procesora
  - Wiele rdzeni przetwarzających w pojedynczych procesorach

Potokowość i współbieżne wykonanie instrukcji
Współbieżność wewnętrzna

Potoki wykonywania instrukcji zapewniają ich wykonanie w wielu równoległych krokach (np. 20 kroków w Pentium4) – kroki o niewielkiej złożoności stąd możliwość szybkiego taktowania – efekt skrócenie czasu realizacji instrukcji przez potokowość (równoległość)

Potok realizujący wiele instrukcji równocześnie wymaga predykcji kierunku realizacji kodu w punktach rozgałęzień kodu dla właściwego zapełnienia potoku.

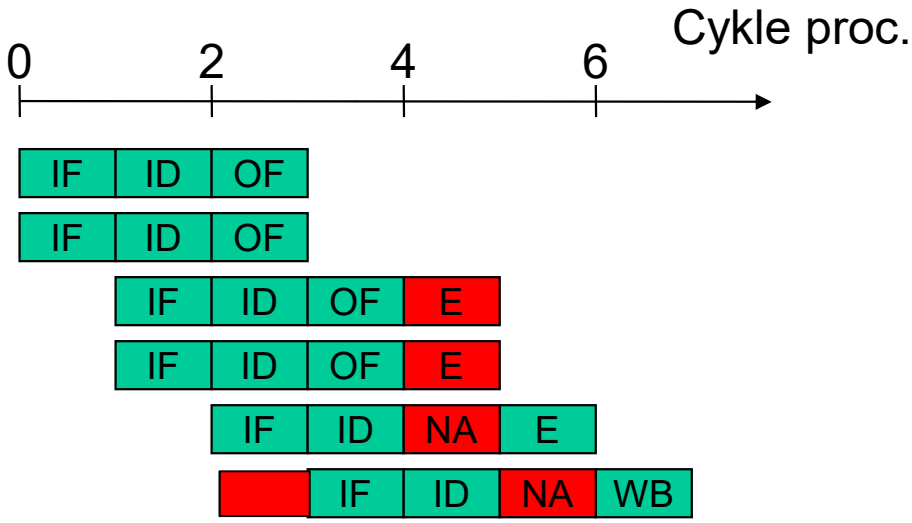
Wiele potoków pozwala na jednoczesne zakończenie realizacji kilku instrukcji – kilka instrukcji w cyklu dzięki równoległości potokowej i wielodrożności procesorów np. two-way superscalar execution (dwudrożne wykonanie kodu).

Superskalarne przetwarzanie - przykład

load R1, @1000
load R2, @1008
Add R1, @1004
Add R2, @100C
Add R1, R2
Store R1, @2000

load R1, @1000
Add R1, @1004
Add R1, @1008
Add R1, @100C
Store R1, @2000

load R1, @1000
Add R1, @1004
load R2, @1008
Add R2, @100C
Add R1, R2
Store R1, @2000



IF: POBRANIE INSTRUKCJI

ID: DEKODOWANIE

OF: POBRANIE OPERANDU

E: WYKONANIE

WB: ZAPIS

NA: BRAK DZIAŁANIA

Współbieżne przetwarzanie w 2 potokach

Przetwarzanie superskalarne i potokowe

Problemy – konflikty :

- Zależność między danymi – wynikiem jest oczekiwanie np. w takcie 5 i 6 (No operation)
- Zależności rozgałęzień (proceduralne) – przewidywanie kierunku przetwarzania po rozgałęzieniu jest krytyczne dla efektywności przetwarzania superskalarnego – zapełnienie potoku przetwarzania – które rozkazy kolejno po rozgałęzieniu
- Konflikty zasobowe –
  - przykład:
  - 2 potoki przetwarzania instrukcji a jedna jednostka wykonawcza zmiennoprzecinkowa lub
  - niskie wykorzystanie wielu jednostek wykonawczych

Przetwarzanie superskalarne i potokowe

Rozwiązania:

- Zmiana kolejności realizacji instrukcji programu –

  out-of-order (dynamic) instruction issue (przetwarzanie dynamiczne - cecha procesora)

  – sprzętowe badanie możliwości równoległej realizacji instrukcji
- Układy przewidywania rezultatu rozgałęzienia kodu
- Pobieranie rozkazów i danych z wyprzedzeniem
- Wiele jednostek wykonawczych, lecz niski poziom wykorzystania tych zasobów (dla typowych programów) prowadzi typowo do konstrukcji procesorów posiadających do 4 potoków przetwarzania

Problemem jest: ograniczenie wzrostu efektywności procesorów przez ograniczony poziom wewnętrznej współbieżności kodu sekwencyjnego.

- Brak możliwości wzrostu efektywności przetwarzania poprzez polepszenie poziomu zrównoleglenia realizacji programów sekwencyjnych

oraz

- duża dostępność układów logicznych w dostępnych technologiach produkcji obwodów cyfrowych

prowadzi do koncepcji przyspieszenia obliczeń poprzez:

- wiele rdzeni w procesorze i równoległości na poziomie programów i wątków.

Procesory z długim słowem rozkazowym

VLIW (ang. very long instruction word) processor - zastosowanie kompilatora do odkrywania sekwencji niezależnych instrukcji nadających się do współbieżnej realizacji;

Instrukcje, które mogą być realizowane współbieżnie są:

- określane za pomocą jednego słowa rozkazowego, które składa się z wielu instrukcji realizowanych współbieżnie przez wiele jednostek wykonawczych.

Problemem jest brak znajomości przez kompilator stanu procesu (analiza tylko off-line) i stąd utrudniona efektywna generacja paczek instrukcji, problemem np. przewidywanie kierunku przetwarzania po rozgałęzieniu w kodzie.

Do tej grupy procesorów należą zaproponowane przez Intel i HP procesory EPIC (explicit parallel instruction computing) np. Intel Itanium - paczka 3 instrukcji realizowanych w równoległych potokach procesora.

Ograniczenia efektywności systemu pamięci

- Parametry systemu pamięci :
  - opóźnienie (ang. latency) - czas odpowiedzi systemu pamięci na żądanie danych przez procesor,
  - przepustowość systemu pamięci (ang. bandwidth) - ilość danych dostarczana procesorowi przez pamięć w jednostce czasu,
  - wielkość pamięci podręcznej procesora – pp szybkka pamięć, z której korzysta bezpośrednio procesor,
  - wielkość linii pamięci podręcznej (ang. cache line) - **liczba sąsiednich słów pobierana jednorazowo do pamięci podręcznej** z pamięci operacyjnej w sytuacji, gdy w pamięci nie ma wartości (danych) z interesującego procesor zakresu (wystąpił brak trafienia),
  - wielkość bufora translacji adresów - TLB.

Opóźnienie dostępu do pamięci powoduje często spadek efektywności przetwarzania systemu komputerowego zależny od rodzaju kodu, a w szczególności od:

- wymaganej w kodzie liczby dostępów do pamięci przypadających na operację - wskaźnik dostępu – ang. memory access rate (głównie odczytów z pamięci – gdyż umożliwiają bezpośrednio wykonanie operacji procesora) i

- lokacji danych (z której pamięci muszą dane do procesora być dostarczone, zawsze pośrednio przez najbliższą procesorowi pp L1).

Ograniczenia efektywności przetwarzania - system pamięci

Pamięć podręczna (pp):

- Zmniejszenie wypadkowego opóźnienia dostępu do pamięci poprzez zastosowanie pp. Procesor realizuje dostęp do pamięci operacyjnej (odczyt i zapis) poprzez pamięć podręczną.
- Stosunek trafień do pp (ang. hit ratio) – określa iloraz;
    - liczby odwołań do pamięci w sytuacji gdy żądana linia (zawierająca adres odwołania) była w pamięci, do
    - liczby wszystkich odwołań do tej pamięci.

Średni czas dostępu do słowa w pamięci - T

- t_{pp} - czas dostępu do pamięci podręcznej
- t_m - czas dostępu do pamięci głównej
- h – stosunek trafień do pp
- $T = h\, t_{pp} + (1-h)\, t_m$
- h=0? h=1? Przykład kodu ?

Ograniczenia efektywności przetwarzania - system pamięci

- Pozytywny efekt zastosowania pp danych wynika z:
  - wielokrotnego wykorzystania danych z pp (szybki dostęp) sprowadzonych jednokrotnie - jako linia pp (wolny dostęp)

Cechami programów, które to zapewniają są:

- czasowa lokalność odwołań (ang. temporal locality of reference) – dane raz sprowadzone do pamięci zostaną użyte wielokrotnie zanim zostaną z pamięci usunięte lub unieważnione, brak clo powoduje niski stosunek trafień do pp.
- przestrzenna lokalnością odwołań (plo) - (ang. spatial locality of memory access) korzystanie w kodzie z danych zajmujących sąsiednie lokacje w pamięci – brak plo powoduje niski stosunek trafień do pp i/lub do bufora translacji adresów. Jeżeli tablica jest zapisywana wierszami w pamięci to kolejne dostępy do tablicy powinny też, jeśli to możliwe, być realizowane wierszami, gdyż:

1. umożliwia to wykorzystanie wartosci obecnych w pp dzięki pobraniu ich we wcześniej wykorzystywanej linii pp.
2. umożliwia to korzystanie maksymalnie długo z jednego wpisu odwzorowania stron w buforze translacji adresów.

Ograniczenia efektywności przetwarzania – bufor translacji adresów danych

Aby dokonać dostępu do danych odczyt/zapis konieczne jest wyznaczenie adresu gdzie dane się znajdują.

Procesor na podstawie kodu wyznacza adres wirtualny (adres danych dostępnych w systemie komputerowym).

Aby zrealizować dostęp konieczne jest wyznaczenie rzeczywistego adresu, pod którym dane się znajdują w pamięci operacyjnej komputera.

Pamięć procesów składa się ze stron wirtualnych, ciągłej przestrzeni adresowej, strona wirtualna zawiera typowo 4 KB danych lub kodu i wymaga 4KB pamięci operacyjnej.

Adres wirtualny jest zamieniany automatycznie (sprzętowo - szybko) na adres rzeczywisty pamięci operacyjnej („adres ramki”) za pomocą bufora translacji (TLB – część procesora, każdego rdzenia oddzielnie) jeśli w pamięci operacyjnej żądana strona się znajduje, a jej adres jest zapisany w buforze.

Brak strony lub brak adresu w buforze translacji jest w sposób kosztowny obsługiwany programowo przez system operacyjny. Należy minimalizować liczbę cyklicznie używanych stron do takiej, jaka się mieści w pamięci operacyjnej i takiej jaka mieści się w TLB.

Bufor translacji adresów – TLB
bufor translacji adresów danych - DTLB

Bufor (TLB) to pamięć podręczna „adresów ramek” dla stron wirtualnych. Wielkość TLB jest ograniczona do ok. 512- 1024 wpisów na rdzeń procesora (pamieć operacyjna rzedu GB mieści 10exp6 stron wirtualnych).

Zakładamy, że wykorzystywane strony wirtualne dostępne są w pamięci. Nasz problem – liczba „jednocześnie” wykorzystywanych stron wirtualnych.

Jeżeli rzadko zmieniamy stronę (rzadko program odwołuje się do nowej strony) to koszt obsługi braku adresu jest średnio niewielki. Jeśli liczba wykorzystywanych stron jest mniejsza od wielkości bufora translacji adresów to dostęp do adresów fizycznych i a następnie dostęp dodanych jest szybki.

Jeśli korzystamy w kolejnych odwołaniach cyklicznie do danych, które zawarte są na większej liczbie stron niż liczba adresów w TLB to koszt sprowadzania adresów stron do DTLB dominuje koszt przetwarzania.

Efektywny kod (1)

Charakteryzuje się lokalnością czasową -

korzysta z danych, które znajdują się w pamięci podręcznej danych, nie zostały z niej usunięte przez brak miejsca lub unieważnienie wynikające z powielenia danych i zapisu dokonanego w innych pamięciach podręcznych innych procesorów.

Charakteryzuje się lokalnością przestrzenną

Dzięki realizacji dostępu do danych umieszczonych kolejno w pamięci:

• rzadko korzysta ze stron nie mieszczących się aktualnie w pamięci operacyjnej,

•rzadko korzysta ze stron wirtualnych, których adresów nie ma w buforze translacji.

Efektywny kod (2)

Optymalne jest jednokrotne pobranie strony z dysku do pamięci operacyjnej oraz jednokrotne wpisanie odwzorowania adresów do bufora translacji (DTLB). Jeśli jednokrotne wystąpienie podczas realizacji kodu na procesorze zdarzenia tego typu (pobranie tych samych danych do pamięci operacyjnej/wpis potrzebnego określonego odwzorowania do bufora translacji) nie jest możliwe (dla danego kodu, wielkości instancji i użytego komputera) to należy dążyć do minimalizacji liczby tych zdarzeń przez odpowiednią konstrukcję kodu zapewniającą wprowadzenie minimalnej liczby etapów przetwarzania realizowanych na aktualnie dostępnych efektywnie w systemie informacjach (danych w pamięci operacyjnej/ odwzorowaniach adresów w DTLB).

Cykliczne odwołania do zbyt dużego zbioru stron wirtualnych (zbioru większego niż rozmiar DTLB) powodują konieczność wielokrotnego uzupełnienia odwzorowania w buforze translacji. Sytuacja staje się wtedy tym bardziej problematyczna (przetwarzania powolne) im częściej kolejny dostęp dotyczy nowej strony wirtualnej.

Ograniczenia efektywności przetwarzania – system pamięci (bez TLB)

Przykład: Wpływ przepustowości pamięci na wydajność przetwarzania
Dane:

- System superskalarny realizujący potencjalnie 4 instrukcje zmiennoprzecinkowe w cyklu,
- **procesor** pracujący z zegarem 1GHZ - szczytowa wydajność procesora to cztery miliardy operacji zmiennoprzecinkowych na sekundę - 4GFLOPS 4*10exp9 operacji w czasie 1s.
- **pamięć podręczna** z czasem dostępu 1 cyklu zegara i linią długości 8 słów,
- **pamięć systemu** z 100 ns opóźnieniem, transmisją blokową na magistrali o szerokości 1 słowa, taktowanej z prędkością 200 MHz,
- Założenie: algorytm wymaga 1 słowa dla realizacji jednej instrukcji zmiennoprzecinkowej,
- stosunek trafień do pamięci podręcznej wynosi 75% (przykład)

Ograniczenia efektywności pamięci – rozwiązanie dla przykładu

Analiza teoretyczna:

Z danych - stosunek trafień - wynika, że 3 na 4 słowa są pobierane bezpośrednio z pamięci podręcznej,czyli

Dostęp do 4 słów jest realizowany w czasie:

- dane bezpośrednio z pp - 3 * 1ns (1GHz)
- dane pośrednio z pp (obliczenie kosztu pobrania z pamięci systemu) –
  - do pp 100 ns +7*5 ns pierwsze (z 8) słowo z pamięci systemu po czasie opóźnienia pamięci, kolejne słowa (transmisja blokowa) po cyklu magistrali (200MHz)
  - z pp 1 ns

Razem – 3+ 135+1=139 ns dla czterech słów

Instrukcja wymagająca średnio 1 słowa będzie mogła zatem być zrealizowana średnio raz na ok. 35 ns.

Rzeczywista przepustowość systemu pamięci dla tego kodu;

– liczba słów na sekundę: 1 / (35 *10 exp 9) = 28 *10exp 6 < 30 M słów / sek.

- w kodzie 1 słowo to jedna instrukcja więc uzyskamy maksymalną wydajność przetwarzania kodu: < 30 * 10exp6 FLOPS (zamiast 4 *10exp9 FLOPS).

Ograniczenia efektywności pamięci – rozwiązanie przykładu

Szczytowa wydajność procesora to cztery miliardy operacji zmiennoprzecinkowych na sekundę - 4GFLOPS

Maksymalna wydajność przetwarzania kodu (ograniczona dostępem do pamięci): 30 MFLOPS

Zmniejszenie szczytowej wydajności procesora ponad 100 krotnie jest efektem:

- czasu dostępu do pamięci systemowej i niskiego stosunku trafień do pamięci podręcznej (kod),

Efektywność przetwarzania zależy również od:

- wielkości linii pamięci podręcznej i
- prędkości magistrali danych.

“Walka“ z ograniczeniami efektywności systemu pamięci – metody ukrywania opóźnienia pamięci

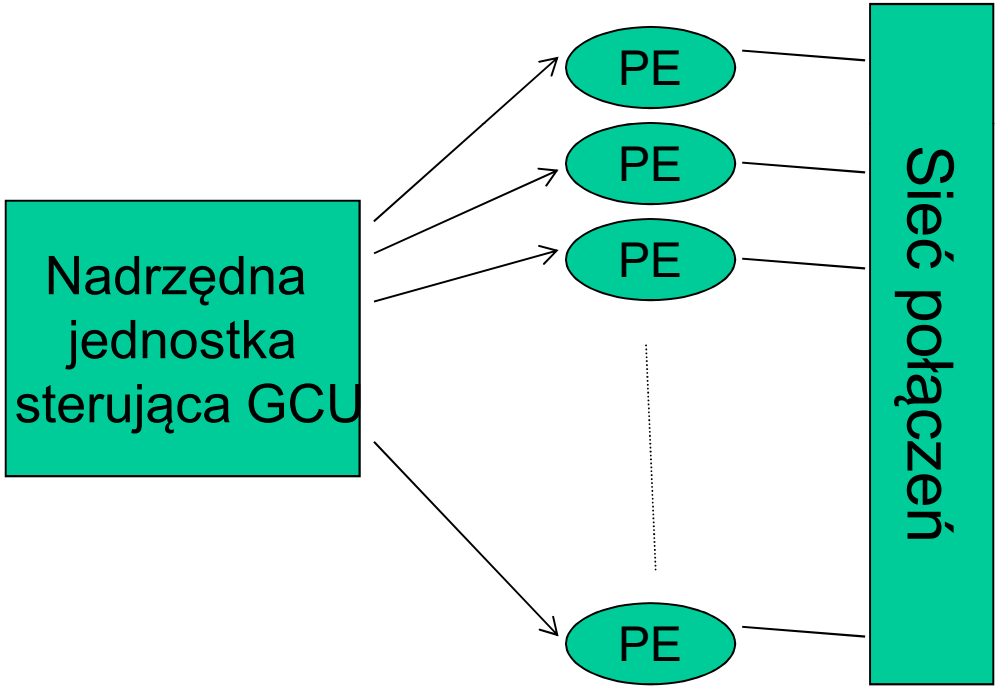
- Żądanie z wyprzedzeniem – określanie zapotrzebowania na dane z wyprzedzeniem, aby dotarły do pamięci podręcznej wtedy gdy będą potrzebne (realizowane przez kompilatory i procesory).
- Wielowątkowość – Zastosowanie wielu wątków pozwala na realizacje przetwarzania wątku gotowego w czasie, gdy pozostałe wątki oczekują na dane z pamięci
  - Szczególnie procesory „wielowątkowe” pozwalają na zarządzanie kontekstem wielu wątków i szybkie przełączanie w momencie realizacji żądań zasobowych poszczególnych procesów.
  - Wzrost liczby wątków może jednak powodować spadek stosunku trafień do pamięci podręcznej (mniejsza pamięć przypadająca na wątek), a w konsekwencji konieczność wzrostu przepustowości magistrali dla zapewnienia odpowiedniej obsługi żądań pamięciowych wątków.
  - Wielowątkowość systemów GPU

Struktury sterowania systemów równoległych (1)

Sterowane scentralizowane – jedna jednostka sterująca nadzorująca synchronicznie pracujące jednostki wykonawcze

- SIMD (ang. Single instruction stream, multiple data stream),
- ta sama instrukcja realizowana we wszystkich jednostkach wykonawczych (procesory określane jako procesory macierzowe) (maszyny Illiac IV, CM-2, MasPar MP-1, obecnie układy wykonawcze procesorów).

Ten typ przetwarzania obecny jest w rozszerzeniach procesorów Intela (AMD) - jednostki MMX, Pentium - SSE (ang. streaming SIMD Extensions). Maska aktywności jednostki określa biorące udział w kolejnym kroku przetwarzania jednostki wykonawcze PE (w zależności od potrzeb/kodu)

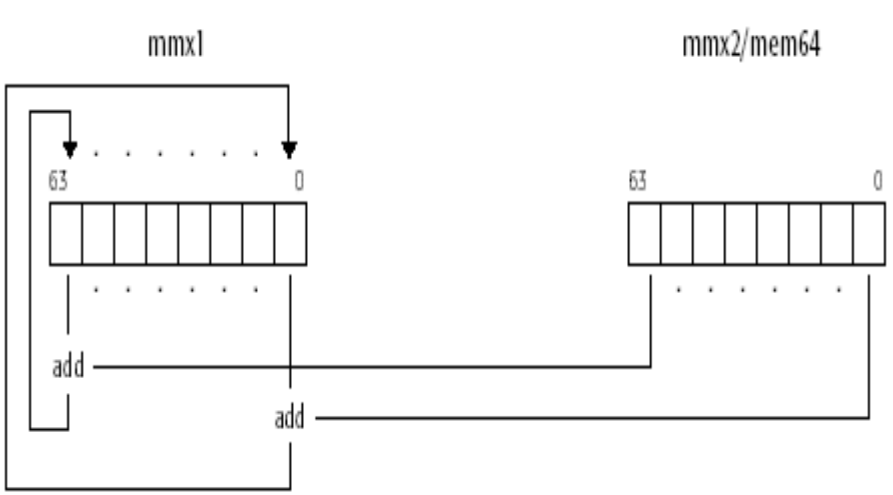


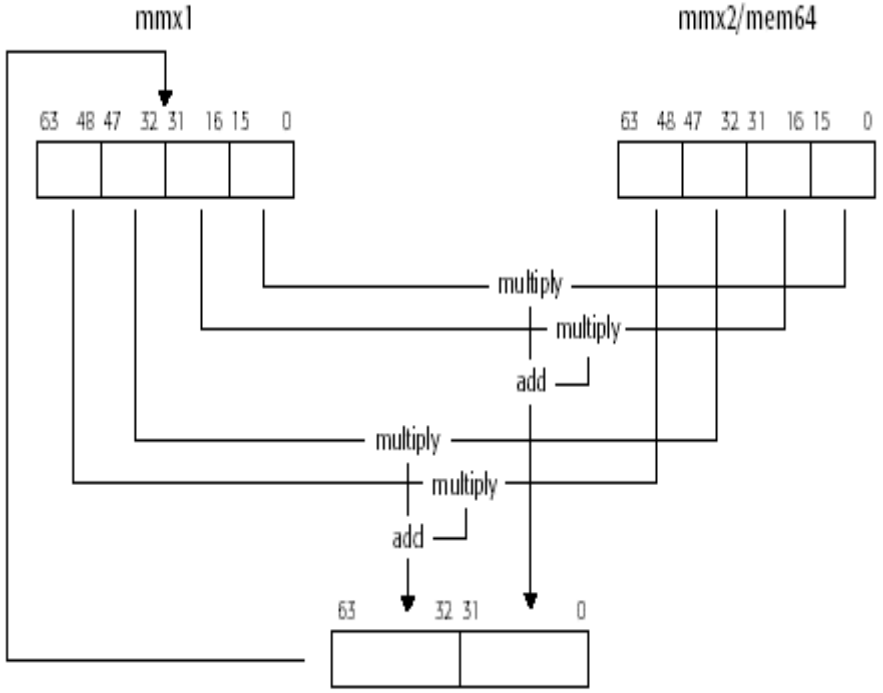
Operacje SIMD - technika MMX i SSE

MMX od pentium MMX rok 1996

- Wykorzystanie 64 bitowych rejestrów koprocesora arytmetycznego oznaczanych jako MMX0-7
- **Równoległe operacje na danych** - wektorach 64 bitowych – **poziom równoległości** zależy od rozmiaru słowa, słowa 8,16,32,64 bitowe, zmienne całkowite
- Instrukcje:
  - Operacje arytmetyczne, logiczne, porównania, arytmetyka modulo lub z saturacją
  - Przenoszenie danych między rejestrami MMX a innymi rejestrami i pamięcią
  - Funkcje zmiany formatu danych wektora
  - Instrukcje realizowane na 2 rejestrach (zapis do rej. operandu)

Przykłady rozkazów MMX





SSE –Streaming SIMD Extension

Współdzielone zasoby wykonawcze dla fpu i simd, lecz oddzielne 128 bitowe rejestry: XMM0- XMM7

Jeden typ danych do rejestrów:

- 4 liczby zmiennoprzecinkowe pojedynczej precyzji - 32-bit

SSE 2

Użycie rejestrów XMM dodatkowo do szerszego zakresu typów zmiennych (SSE) :

- 2 X 64-bit DP FP lub
- 2 X 64-bit int lub
- 4 X 32-bit int lub
- 8 X16-bit short int lub
- 16 X 8-bit - bajty lub znaki.

Przykład: Pobranie, dodawanie i zapis wektora 4 operandów

```
movaps xmm0, [v1] ;      xmm0 = v1.w | v1.z | v1.y | v1.x
addps xmm0, [v2] ;   xmm0 = v1.w+v2.w | v1.z+v2.z | v1.y+v2.y | v1.x+v2.x
movaps [vec_res], xmm0
```

SSE 3, SSE 4 to kolejne instrukcje, szybsza realizacja operacji, więcej rejestrów.

Struktury sterowania systemów równoległych

Sterowanie rozproszone – element przetwarzający może realizować dowolny program niezależnie od pozostałych system MIMD (ang. multiple instruction stream, multiple data stream),

Wersją modelu MIMD jest SPMD (ang. single program ...) realizacja tego samego kodu na różnych danych realizowalny w MIMD jako kod zawierający *if-else* bloki.

W MIMD program i system operacyjny jest zawarty w pamięci każdego procesora, w SIMD niekoniecznie.

SIMD architektury specjalizowane – dla specyficznych zastosowań - np. procesory graficzne

MIMD bardziej odpowiednie dla zastosowań o nieregularnej naturze.

Systemy ze współdzieloną przestrzenią adresową

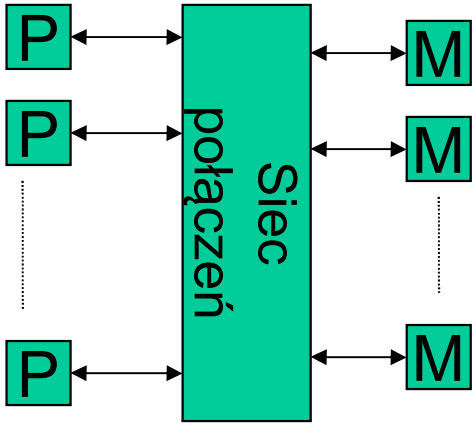
Systemy wieloprocesorowe (ang. multiprocessors) –przestrzeń adresowa współdzielona

SMP (ang. symmetric multiprocessing) – wszystkie procesory mają dostęp do wszystkich zasobów systemu (pamięci i systemu we/wy), jedna przestrzeń adresowa, możliwość przetwarzania dla wszystkich procesorów w trybie jądra (ang. kernel mode – system operacyjny przejmuje realizację wątku programu – np. wiele procesorów realizuje operacje i/o) Występuje: Przestrzeń adresowa lokalna (własna) i przestrzeń adresowa globalna (wspólna dla wszystkich procesorów).

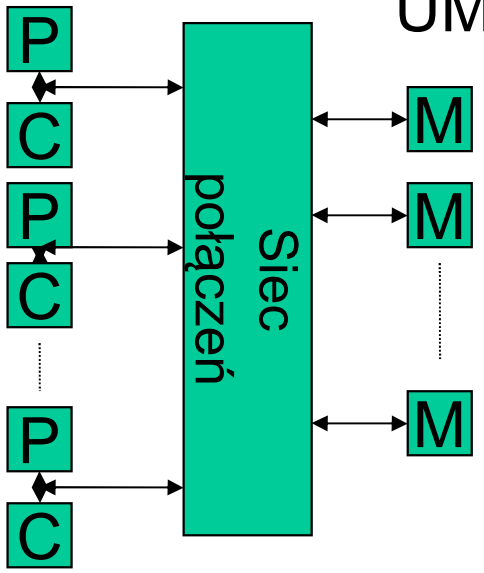
System wieloprocesorowy typu UMA – (ang. uniform memory access multicomputer) - jednakowy czas dostępu każdego z procesorów do dowolnego słowa pamięci.

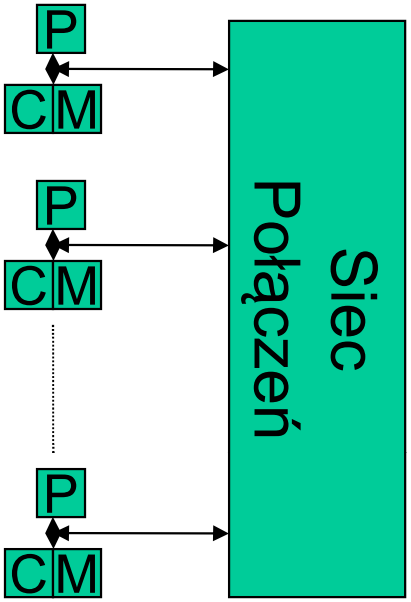
System wielokomputerowy typu NUMA – (ang. Non-uniform memory access multicomputer) – czas dostępu uzależniony od używtej pamięci: lokalnej czy globalnej.

Systemy wieloprocesorowe/wielokomputerowe



UMA





NUMA

C – pamięć podręczna

M – pamięć

Klastry – systemy równoległe zwarte z pamięcią rozproszoną

(logicznie i fizycznie)

- Komputer składający się z węzłów (komputerów – jednostek przetwarzających – procesory, pamięci, magistrale) i sieci połączeń
- Konstelacje – gdy więcej procesorów w węźle niż węzłów w systemie,
- Komputery masowo równoległe – obecnie klastry z dziesiątkami tysięcy i setkami tysięcy procesorów
- Superkomputery – w roku 2010 klastry o mocy rzędu biliarda (10^{15}) operacji zmiennoprzecinkowych na sekundę.

Model systemu równoległego - PRAM

PRAM – równoległy odpowiednik modelu RAM, wspólna przestrzeń adresowa

CECHY:

- jednoczesny dostęp każdego procesora do dowolnej komórki pamięci w jednostkowym czasie,
- Idealizacja maszyn równoległych z SM,
- prosta analiza algorytmów,
- brak efektywnego algorytmu dla PRAM świadczy o braku efektywnego algorytmu dla dowolnej maszyny równoległej.

Maszyny RAM, PRAM

Maszyna RAM

- Jednostaka przetwarzająca i program
- Tasma wejściowa odczytu i tasma wyjściowa zapisu
- Rejestry (każdy mieści wartość integer dowolnego rozmiaru)
- Jednostkowy czas instrukcji

Maszyna PRAM – SM SIMD

- Wiele procesorów z rejestrami prywatnymi (bez taśm)
- Wspólna pamięć
- Dostęp do pamięci w jednostce czasu
- Wejście i wyjście algorytmu w określonych komórkach pamięci
- **Synchroniczna** realizacja przez procesory instrukcji: odczyt operandu, obliczenia i zapis
- EREW, CREW, ERCW, CRCW równoległe lub wyłączne dostępy do pamięci
- Współbieżny zapis wymaga arbitrażu w przypadku konfliktu dostępu; Strategie zapisu: jednolita, dowolna i priorytetowa.

model PRAM a rzeczywistość

Model pełnego równoległego dostępu do pamięci - nierealizowalny w praktyce ze względu na nierealność (wysoki koszt) zapewnienia *p* procesorom jednoczesnego dostępu do dowolnego słowa w pamięci.

Przykład: Algorytm dla CRCW PRAM z jednolitą strategią zapisu

Problem: Wyznaczenie maksimum z tablicy TAB[m]

Jak to zrobić w 3 krokach ?

Przykład: Algorytm dla CRCW PRAM z jednolitą strategią zapisu

```
for i=1,m
    do parallel R[i]=TRUE; //początkowo każdy element i jest największy

 for i=1,m
     for j=1,m
                 do parallel  if (A[i]<A[j]) R[i]=FALSE;
 -- jeśli element jest mniejszy to otrzymuje FALSE
 -- jeśli procesor zapisuje to zapisuje FALSE – CW ok.
 -- R[i] dla wszystkich elementów największych (równych sobie) zachowuje TRUE
for i=1,m
     do parallel if (R[i]==TRUE) MAX= A[i];
```

Jeżeli zapisy dla różnych i np. i1 i i2 to istnieje tylko jedna wartość A[i1] = A[i2] (powielona na wielu pozycjach tablicy) dla której nie ma wartości większych.

Wykonanie programu przez m^2 procesorów zajmuje czas stały O(1). Jednoczesny zapis dotyczy jednakowych danych.

Współdzielenie danych w systemach równoległych

- W systemach równoległych dane współdzielone mogą być powielone w wielu pamięciach podręcznych.
- Zalety replikacji:
  - Obniżenie opóźnienia dostępu i wymagań przepustowości pamięci
  - Mniej rywalizacji o dane odczytywane przez wiele procesorów

STRATEGIE ZAPISU:

Write-through cache – zapis do pp jest synchronicznie odzwierciedlany w pamięci globalnej, każdy zapis uaktualnia pamięć operacyjną (rzadko spotykany)

Write-back cache – zapis w pamięci operacyjnej jest dokonywany w wyniku zewnętrznego żądania odczytu danej.

Spójność pamięci podręcznej zapewnia, że:

- dane zapisane przez jeden procesor będą udostępniane pozostałym do momentu ponownego zapisu,
- wszystkie procesory widzą taką samą kolejność realizacji przez inne procesory zapisów (np. pewnych zapisów nie musza widzieć) i mają zapewniony odczyt aktualnej wartości.

Spójność pp w systemach wieloprocesorowych

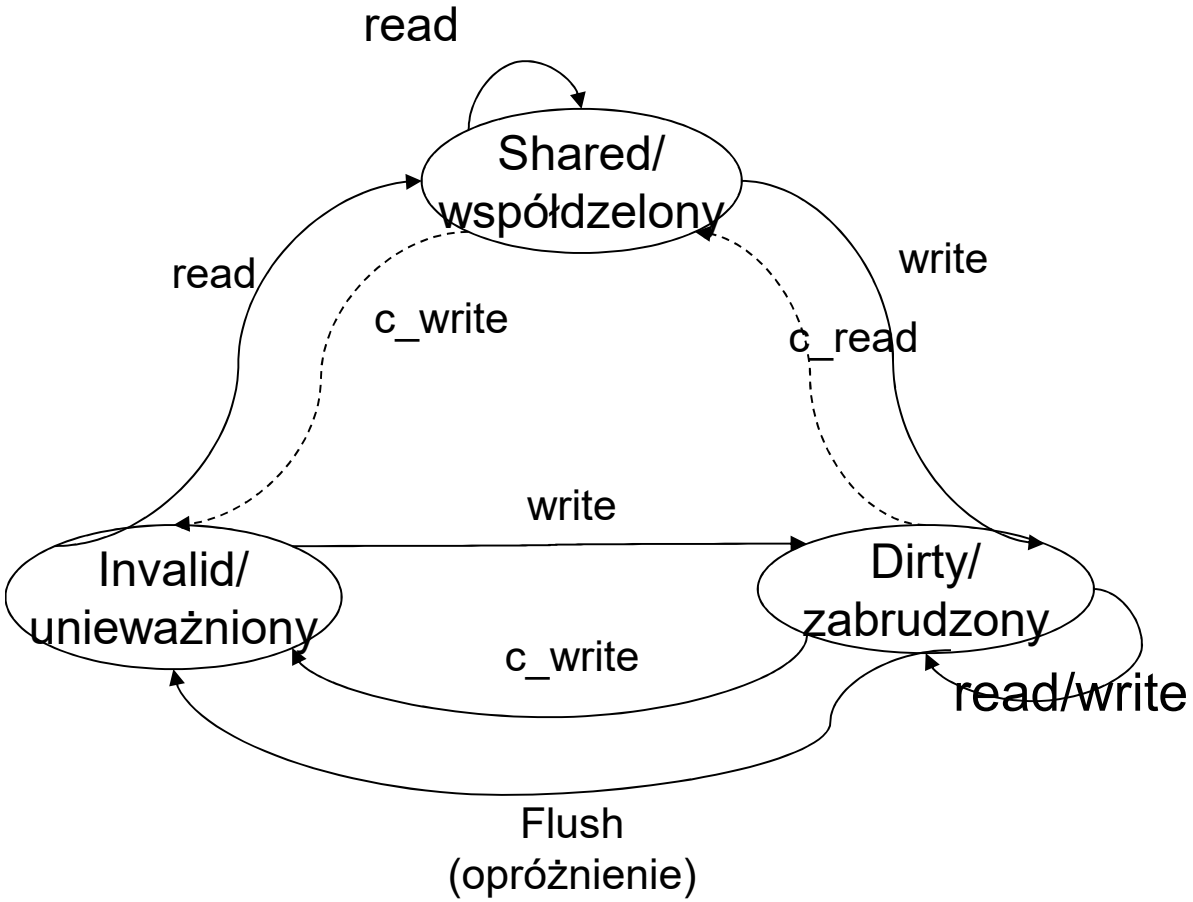
- Protokoły: unieważniania lub uaktualniania służą dla zapewnienia spójności danych - zapewniają istnienie szeregowego porządku wykonania instrukcji realizowanych współbieżnie.
- *Protokół unieważnienia*

powoduje, że w przypadku uaktualnienia lokalnej kopii danych pozostałe kopie danych zostają unieważnione – konsekwencja to wstrzymywanie przetwarzania ze względu na oczekiwanie na dane unieważnione. Częściej stosowany obecnie ze względu na znaczenie przepustowości pamięci i magistrali.

- *Protokół uaktualniania*

polega na tym, że w przypadku uaktualnienia lokalnej kopii danych pozostałe kopie danych zostają uaktualnione - konsekwencja to narzut komunikacyjny wynikający z przesyłania danych, które nie zawsze będą jeszcze wykorzystywane, przesłania następują przy każdej modyfikacji każdego słowa wielosłowowej linii pp

3 stanowy protokół zapewnienia spójności danych pamięci podręcznej



Linia pamięci podręcznej przechodzi między stanami na skutek instrukcji realizowanych przez lokalny procesor (read/write/flush) oraz akcji protokołu zapewnienia spójności w odpowiedzi na działania innych procesorów.

Po modyfikacji linii pp jest ona lokalnie oznaczona jako Dirty w celu zapewnienia, że procesor ten obsłuży kolejne żądania dostępu innych procesorów do danych w tej linii pp.

Zapis wartości do linii unieważnionej (unieważnionych) jest poprzedzony pobraniem aktualnej zawartości linii, w której zmienna się znajduje. Procesor posiadający wersję aktualną udostępnia ją żądającemu procesorowi i dokonuje jej zapisu do pamięci globalnej (w przypadku: write back cache).

Protokół zapewnienia spójności- przykład

Procesor1	Processor2	PP Proc1	PP Proc2	PAMIĘĆ
read x	read y			
write x=x+1	write y=y+1			
read y	read x			
write x=x+y				
	write y=x+y			

Zmienne należą do różnych linii pamięci podręcznej, zapisy wymagają uzyskania przez procesor dostępu w trybie wyłącznym,
Dirty – linia „zabrudzona” - obszar zmodyfikowany - wyłączny dostęp
Shared – linia współdzielona - obszar„współdzielony„
Invalid – linia nieważna – obszar „nieważny”

Protokół zapewnienia spójności- przykład

Procesor1	Processor2	PP Proc1	PP Proc2	PAMIĘĆ
				x= 5, D y=12,D
read x	read y	x=5,S	y=12,S	x= 5, S y=12,S
write x=x+1	write y=y+1	x=6,D	y=13,D	x= 5, I y=12,I
read y	read x	x=6,S y=13,S	x= 6,S y=13,S	x= 6, S y=13,S
write x=x+y		x= 19,D y=13,S	x= 6,I y=13,S	x= 6, I y=13,S
	write y=x+y	x= 19,D y=13,I	x= 6, I y=19,D	x= 6, I y=13,I

Read pobranie wartości z pp do rejestru procesora.
Write zapis wartości z rejestru do prywatnej pp.
Procesory mają prywatne pamięci podręczne. Dane mogą być powielone w pp. Zmienne należą do różnych obszarów - linii pamięci podręcznej, zapisy wymagają uzyskania przez procesor dostępu w trybie wyłącznym.
Dirty – linia „zabrudzona” - zmodyfkowany, wyłączny dostęp
Shared – linia „współdzielony„
Invalid – linia „nieważny”

Tablica przejść linii pp między stanami

stan	read	write	flush	c-read	c-write
shared	shared	dirty	invalid	shared	invalid
dirty	dirty	dirty	invalid	shared	invalid
invalid	shared	dirty	invalid	invalid	invalid

c-read i c-write ocznaczają efekt działań protokołu zapewnienia spójności, operacja flush powoduje zapisanie bloku danych do pamięci globalnej i w przypadku konieczności ponownego wykorzystania ponowne jego wczytanie.

Spójność pamięci podręcznej

protokół podglądania (ang. Snoopy cache coherence protocol) .

- Procesor monitoruje przesłania na magistrali dotyczące swoich linii pp.
- Procesor zapisuje lokalnie stan swoich danych.
- Wykrycie zewnętrznego żądania odczytu linii, którego stan jest „dirty” powoduje przesłanie przez procesor lokalnej kopii linii do żądającego przesłania procesora.
- Jeżeli natomiast zdalnie nastąpił zapis do linii pamięci, którego kopia jest lokalnie przechowywana to następuje unieważnienie jego zawartości.
- Operacje na linii „dirty” są realizowane lokalnie
- Mechanizm wymaga rozgłaszania do procesorów informacji o operacjach na pamięci (funkcje zapewnienia spójności). Rozgłaszanie to brak skalowalności (liczba operacji wzrasta z liczbą uczestników). Lepsza efektywność w przypadku zapamiętywania *istnienia/braku stanu współdzielenia danych* (czy linia jest współdzielona czy jest wyłączna? – pozwoli na obniżenie wymagań przepustowości magistrali – komunikacja tylko do zainteresowanych).

Spójność pamięci podręcznej

mechanizmy katalogowe

- Protokół zapewnienia spójności bazujący na katalogu scentralizowanym lub rozproszonym (lepsza skalowalność).
- Pamięć centralna jest rozszerzona o pamięć katalogową (PK), w której zapisywane są informacje na temat korzystających z poszczególnych stron pamięci procesorów. Te procesory będą uczestniczyły w dystrybucji informacji zapewniających spójność.
- W przypadku rozproszonych PK znika także wąskie gardło jakim jest obsługa protokołu spójności w oparciu o jedną PK – wtedy możliwa jest jednoczesna realizacja wielu operacji zapewnienia spójności.

- Maszyny ze sprzętowym rozwiązaniem problemu spójności pamięci podręcznej nazywane są ccNUMA, mogą wykorzystywać SCI (scalable coherent interconnect) protokół IEEE.

Zarządzanie pamięcią
Pamięć wirtualna

Pamięć wirtualna

- przydzielana w blokach o wielkości „strony pamięci wirtualnej”,
- udostępniana procesom po zapisaniu strony do obszaru pamięci operacyjnej - rzeczywistej – do tzw. „ramki pamięci”,
- umożliwia przydział procesom większej ilości pamięci niż jest dostępna fizycznie w systemie,
- aby proces mógł pobrać dane konieczne jest odwzorowanie adresu wirtualnego na aktualny adres fizyczny, pod którym dane aktualnie się znajdują,
- konieczna jest zatem translacja adresów wirtualnych na adresy fizyczne.

Zarządzanie pamięcią
Bufor translacji adresu (TLB)

- Każdy dostęp procesora do pamięci powoduje konieczność określenia fizycznego adresu pod którym znajduje się wartość spod określonego w kodzie adresu wirtualnego.
- Odwzorowanie (translacja) jest realizowane przez bufor translacji adresu TLB (ang. translation lookaside buffer), który zawiera adresy fizyczne ostatnio translowanych adresów wirtualnych.
- Pamięć TLB może posiadać strukturę wielopoziomową i może być oddzielna dla danych i kodu. W przypadku braku adresu wirtualnego w TLB system operacyjny korzystając z katalogu i tablic stron określa brakujący adres i wpisuje go do TBL wykonując procedurę obsługi przerwania (znaczy koszt czasowy).

Zarządzanie pamięcią II

Brak informacji w TLB jest nazywany brakiem trafienia do TLB. Niski stosunek trafień do TLB jest spowodowany niską przestrzenną lokalnością kodu. Np. w programie (język C) odczyt różnych elementów kolumny tablicy z długimi wierszami.

Gdy znany jest adres fizyczny operandu wtedy można określić:

- czy pobierana z pp L1 (współbieżność działań) wartość jest poprawna,
- skąd należy wartość pobrać, czy jest w strukturze pp czy trzeba pobrać z pamięci operacyjnej .

W przypadku braku linii z żądanymi danymi w pp poziomu1 (cache L1) brakująca linia (cache miss) jest pobierana z pamięci niższego poziomu pamięci podręcznej (L2, L3) lub pamięci operacyjnej.

W przypadku braku strony z żądanymi danymi w pamięci operacyjnej (page fault) żądana strona odczytywana jest z dysku.

Koszty dostępu do pamięci

dla SGI Onyx2 komputer z współdzieloną pamięcią rozproszoną (DSM)

Rodzaj dostępu	Czas obsługi [cykl]
rejestry procesora	0
pp L1 trafienie	2-3
pp L1 brak trafienia dane ładowane z pp L2	8-12
pp L2 brak trafienia dane pobrane z pamięci operacyjnej, adres wirtualny w TLB	75-250 150-500ns
brak adresu w TLB – konieczność załadowania adresu fizycznego do TLB, strona znajduje się w pamięci operacyjnej	2000
brak strony pamięci - konieczność załadowania strony wirtualnej z dysku - opóźnienie 10^8 cykli	10^8 100ms

Koszty komunikacji w maszynach z pamięcią współdzieloną - analiza jakościowa

Uwzględniane informacje o systemie:

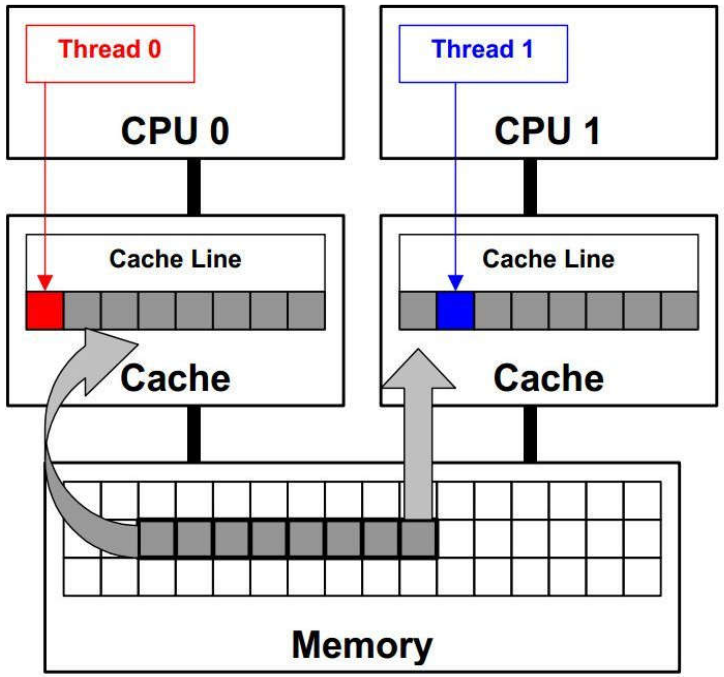
- Struktura pamięci – dostęp do zmiennych lokalnych znacznie szybszy niż zmiennych zdalnych (które dane lokalne, a które zdalne? - nie zawsze proste do określenia).
- Ograniczone wielkości pamięci podręcznej – przekroczenie rozmiaru, nadpisanie danych, konieczność wielokrotnego pobierania, narzut operacji zapewnienia spójności danych.
- Kolejność operacji na wielu procesorach wpływa na kolejność dostępu, unieważnienia i uaktualniania danych.
- Optymalizacja kodu przez zapewnienie lokalności przestrzennej i lokalności czasowej odwołań do pamięci.

Koszty komunikacji w maszynach z pamięcią współdzieloną - analiza jakościowa cd

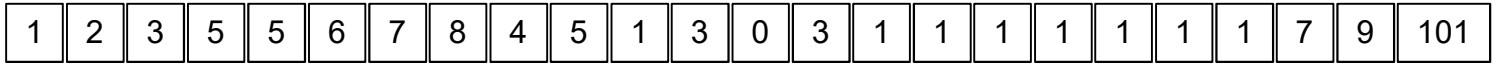
Wstępne pobieranie danych wprowadzane przez kompilatory i sprzęt pozwala ukryć opóźnienia dostępu do pamięci.

Nieprawdziwe współdzielenie (false sharing) – dodatkowe narzuty czasowe wynikajace z modyfikacji przez różne procesory różnych wartości ulokowanych w tej samej linii pamięci – efektem jest unieważnienie linii i wstrzymanie dostępu do momentu sprowadzenia do pp linii w postaci ostatnio zmodyfikowanej.

Rywalizacja w dostępie współdzielonym – opóźnienia dostępu wynikające z współubiegania się o te same dane.



Rozważmy prosty problem sumowanie - redukcja



Sumowanie elementów tablicy

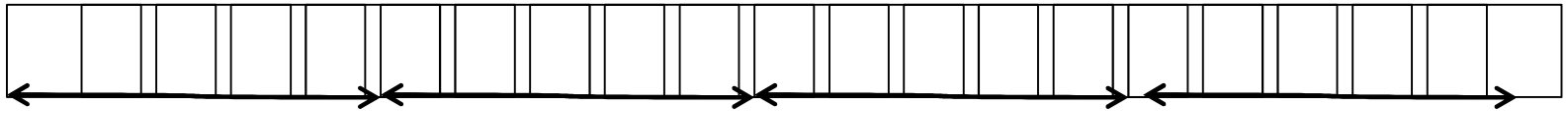
```
a1 = __rdtsc();

for(int i=0;i<MAX;i++)   suma+=table[i];

a2 = __rdtsc();
printf("Czas obliczeń =%llu us \n", (a2-a1)*3/10000);
```

Czas przetwarzania sekwencyjnego:

dla MAX=100000 i Intel 2,8 GHz i5 (4 rdzenie)

250 us

Zróbmy to równolegle



Sumowanie elementów tablicy za pomocą 4 procesorów

```
a1 = __rdtsc();

#pragma omp parallel
#pragma omp for shared(table,suma)
for(int i=0;i<MAX;i++)    suma +=  table[i];

a2 = __rdtsc();
printf("Czas obliczeń =%llu us \n", (a2-a1)*3/10000);
```

Równoległe sumowanie elementów – wyścig

Wyniki:

niepoprawne modyfikajce – wyścig - rezultat błędny wynik `suma`

(ciąg 2 dostępów jednego wątku do zmiennej: odczyt i zapis, rozdzielone dostępami do tej zmiennej innych wątków)

```
Poprawnie:
5, odczyt1, zapis1,6, odczyt2,zapis2,7
Niepoprawnie:
5, odczyt1, odczyt2,zapis1,6,zapis2,6
```

Rozwiązanie:

Synchronizacja dostępu do zmiennej

Równoległe sumowanie elementów – poprawny program liczy się powoli

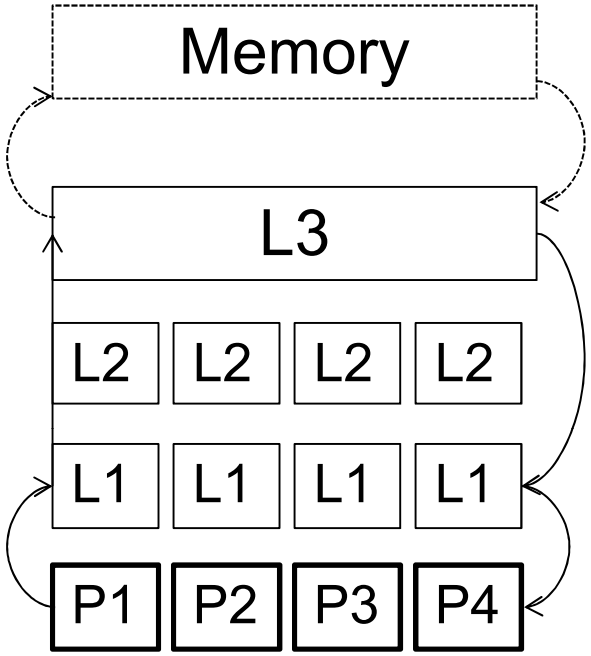
Rozwiązanie 2:

```
a1 = __rdtsc();

#pragma omp parallel
#pragma omp for shared(table,suma)
for(int i=0;i<MAX;i++)
#pragma omp atomic
   suma+=table[i];

a2 = __rdtsc();
printf("Czas obliczeń =%llu us \n", (a2-a1)*3/10000);
```

Wynik poprawny: czas pracy około 2800 us czyli zamiast skrócenia czasu obliczeń - 11 razy dłużej niż sekwencyjnie.

Dlaczego ?

Równoległe sumowanie elementów



Przepisywanie wartości współdzielonej między pamięciami

```
#pragma omp parallel for
for(int i=0;i<MAX;i++)
#pragma omp atomic
    suma+=table[i];
```

Koszty:

- koszt oczekiwania
- koszt dostępu do zamka (mutex_lock) dla synchronizacji dostępu do zmiennej
- koszt przepisywania wartości pomiędzy pamięci: procesorem, pamięcią podręczną L1 jednego rdzenia, współdzieloną pp L3, L1 innego rdzenia, a nawet do innego procesora (gdy wiele fizycznych procesorów)

suma jest:
współdzielona, modyfikowana ale
nie jest lokalna !! – nie jest w sposób ciągły dostępna w rejestrze lub w pamięci podręcznej

Równoległe sumowanie elementów

Rozwiązanie 3:

Większa lokalność przetwarzania – prywatna suma

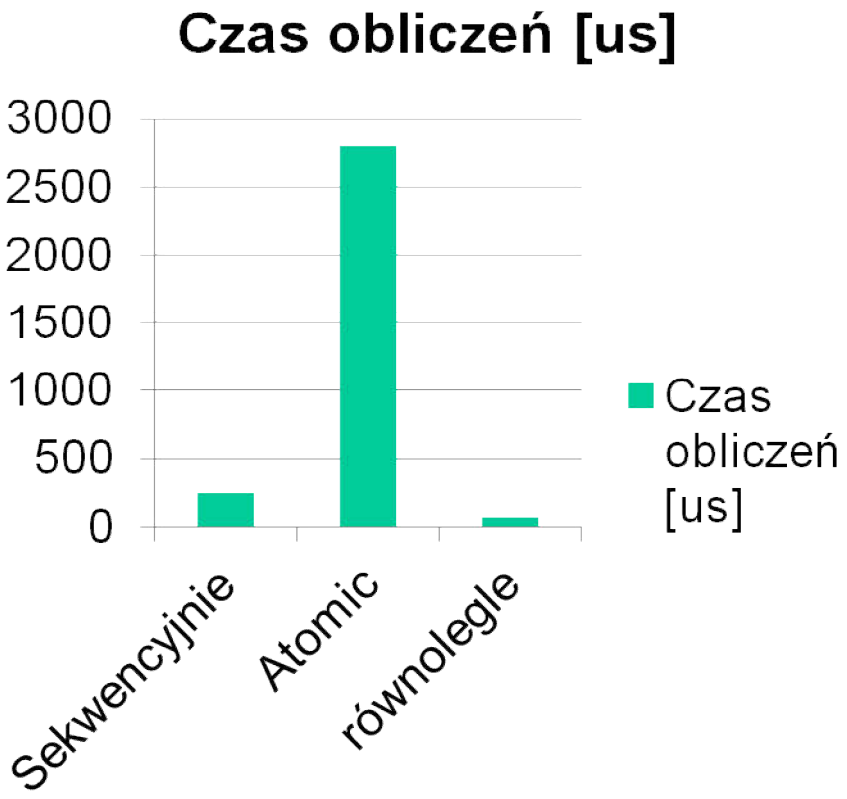
```
#pragma omp parallel
    {
    float sumal=0.0;
#pragma omp for
    for(int i=0;i<MAX;i++) sumal+=table[i];
#pragma omp atomic ;poza pętlą – każdy wątek jeden raz
        suma+=sumal;
    }
```

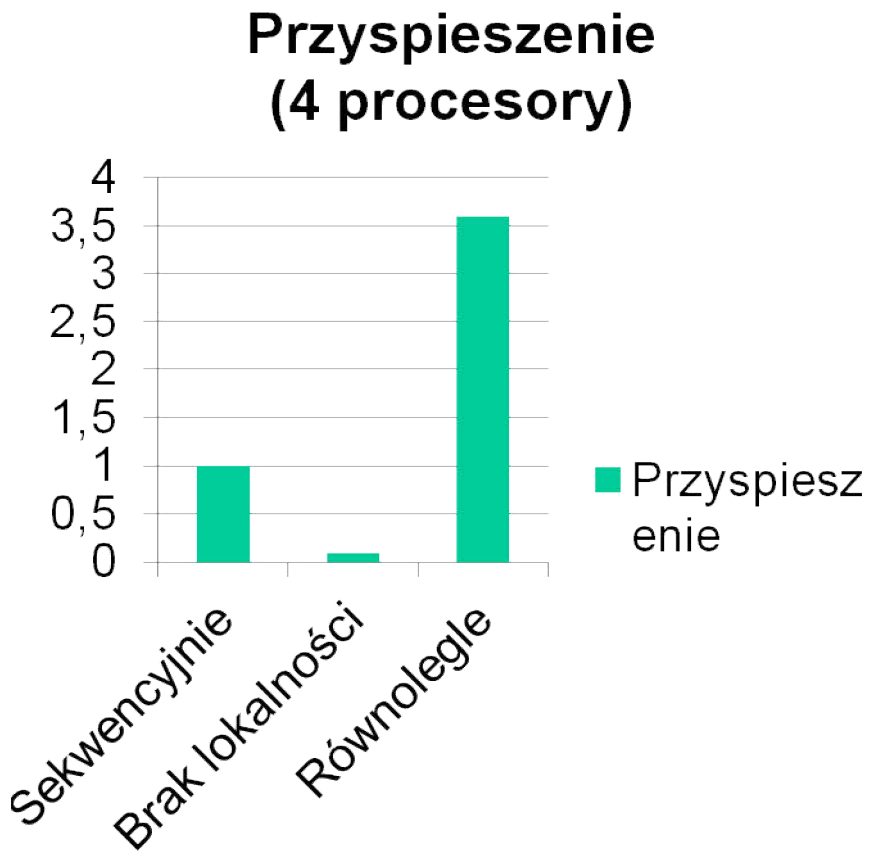
Czas przetwarzania na 4 procesorach: 70 us czyli 3,6 x szybciej niż sekwencyjnie.

Wnioski:

1. efektywna równoległość = maksymalizacja lokalności dostępów do danych
2. równoległość to często konieczność synchronizacji obliczeń (ale synchronizacji efektywnej – rzadko występującej).

Równoległe sumowanie





Koszty komunikacji w maszynach z pamięcią współdzieloną – analiza ilościowa

Określenie czasu dostępu do odczytu *m* słów w maszynach z pamięcią współdzieloną:

$t_a = t_s + m\,t_w$ czas dostępu do odczytu bez rywalizacji (w przypadku dostępu do tych samych danych przez różne procesory ze względu na konieczne uszeregowanie transmisji czasy t_s się zwielokrotniają).

t_s- współczynnik czasu dostępu wynikający z narzutów:

- mechanizmu spójności,
- sieci połączeń i pamięci,

uwzględniany raz we wzorze na czas dostępu do ciągu *m* słów współdzielonych (nawet większych od rozmiaru pp ze względu na mechanizmy ukrywania opóźnienia)

t_w - współczynnik czasu dostępu do jednego słowa w pamięci podręcznej

- W przypadku konieczności zapisu współdzielonych danych przez jeden z procesorów konieczne uwzględnienie zapisu (przez jeden) i (odczytu przez pozostałe) aktualnej wartości danej z pamięci.
- Wzór nie uwzględnia nieprawdziwego współdzielenia oraz nakładania się obliczeń i komunikacji

Procesory systemu równoległego wykonują w podanej kolejności operacje (każdą w sposób atomowy- niepodzielny) dostępu do zmiennych A,B,C (zmienne A i B leżą w tej samej linii pamięci podręcznej zmienna C natomiast w innej). Kolejność realizacji operacji przez procesory jest następująca:

Kolejność 1 2 3 4 5 6 7 8 9 10 11

Procesor 1 2 3 1 2 1 3 1 2 1 1

Operacje:

A+=B , C=1, B=2*C, A+=B, B=3*C, A+=B, B=4*C, A+=B, B=C, A+=B, A+=B

Przed przetwarzaniem pamięć podręczna nie zawiera danych. Dla poszczególnych operacji proszę określić w poszczególnych procesorach wystąpienie zdarzeń dotyczących ich lokalnej pamięci podręcznej:

- pobranie danych do pamięci podręcznej,
- unieważnienia linii danych w pamięci,
- trafienia do pamięci.

Dostęp (odczyt lub zapis) procesora do powyżej wymienionych danych jest możliwy za każdym razem pod warunkiem obecności ich w pamięci podręcznej. Obowiązuje protokół zapewnienia spójności bazujący na unieważnianiu nieaktualnych kopii linii pamięci podręcznej.

Zadanie sprawdzające