Literatura

1. Introduction to Parallel Computing; Grama, Gupta, Karypis, Kumar; Addison Wesley 2003
2. Wprowadzenie do obliczeń równoległych, Zbigniew Czech, Wydawnictwo Naukowe PWN, 2010.
3. Designing and Building Parallel Programs; I.Foster, http://www-unix.mcs.anl.gov/dbpp/
4. Podstawy programowania współbieżnego, M. Ben-Ari, WNT
5. Programowanie współbieżne i rozproszone, Z.Weiss, T.Gruźlewski, WNT
6. Software Optimization for High-Performance Computing; Wadleigh, Crawford, Prentice Hall 2000

Zakres przedmiotu

1. Motywacje do zagadnień przetwarzania ||
2. Architektury maszyn ||
3. Modele przetwarzania ||
4. Podstawy projektowania algorytmów ||
5. Ocena efektywności programów ||
6. Komunikacja w systemach ||
7. Algorytmy ||

Dlaczego warto studiować przetwarzanie równoległe ?

- Przetwarzanie sekwencyjne jest dziś całkiem szybkie
- Jednak:
  - Potrzebujemy ciągle rosnącej wydajności
  - Budujemy systemy równoległe np. procesory wielordzeniowe, wieloprocesorowe karty graficzne
  - Potrzebujemy umiejętności tworzenia programów równoległych
  - Równoległość to duże wyzwanie intelektualne dla informatyki – modele, języki, algorytmy, sprzęt

Zamiana sytuacji w technologii mikroprocesorów

Lata 1986- 2002 mikroprocesory przyspieszały średnio z roku na rok o około 50%

Od 2003:

- wzrost efektywności około 20% na rok,
- obserwacja granic wzrostu prędkości wynikających z:
  - właściwości fizycznych materiałów (odprowadzanie energii cieplnej, cieplno - niestabilne systemy, opóźnienia transmisji sygnału),
  - wykorzystania już możliwości zrównoleglenia przetwarzania na poziomie pojedynczego procesora (długość słowa, superskalarność)
  - wysokiej złożoności struktury mikroprocesora

Dziś równoległość przetwarzania nie tylko w superkomputerach i centrach naukowych. Zamiast projektować i realizować szybsze mikroprocesory umieszcza się wiele procesorów w ramach pojedynczego układu scalonego np. rdzenie.

Oczekiwania na wzrost efektywności - przykłady

- Modelowanie klimatu, prognozowanie pogody
- Nauki biologiczne (badania genomu, struktury białek itp)
- Badania nad lekarstwami, szczepionkami
- Poszukiwania nowych źródeł energii
- Analiza danych w gospodarce

Rola programistów – co jest problemem ?

- Dodatkowe rdzenie **nie powodują** wzrostu efektywności obliczeń - jeśli programiści ich nie wykorzystają (dla kodu sekwencyjnego).
- Efektywność przetwarzania programów sekwencyjnych **nie wzrasta** w systemach wielordzeniowych, wieloprocesorowych.
- Konieczna wiedza o **programowaniu równoległym** i umiejętności przeprowadzenia **równoległych obliczeń**.

Możliwe podejścia

- Ponowne napisanie programów sekwencyjnych aby stały się ||
- Zastosowanie programów zrównoleglających:
  - Ograniczony zakres powodzenia
  - Zrównoleglenie tylko pewnych konstrukcji kodowych – np. równoległe wykonanie pętli
  - Trudne zadnie
- Podejście „od nowa” - zaprojektowanie algorytmu ||, tak aby odzwierciedlał równoległość dostępną w problemie i metodzie jego rozwiązania.

Dalsze uzasadnienia dla przetwarzania równoległego

- Skrócenie czasu realizacji systemów wieloprocesorowych (lub wielordzeniowych)
  - wykorzystanie standardowych interfejsów sprzętowych pozwala na zastosowanie nowych architektur i technologii procesorów,
  - zaawansowane narzędzia projektowe i testujące.
- Postęp w standaryzacji środowisk przetwarzania równoległego zapewniający dłuższy cykl życia aplikacji równoległych (MPI, Open MP, OpenGL)

Budowa sprzętu - równoległość wewnętrzna

- **Wzrost złożoności** układów scalonych (IC) – podwojenie złożoności w ciągu 18 miesięcy (wg Gordona Moora) nowe technologie wytwarzania IC

- Dla wykorzystania **możliwego technologicznie** wzrostu złożoności IC i jego przekształcenia na wzrost wydajności przetwarzania konieczna praca nad równoległością wewnętrzną procesora (potokowość, superskalarność, dynamiczne przetwarzania rozkazów, predykcja rozgałęzień, wstępne pobranie kodu i danych), konieczne jej zrozumienie.

Prędkość dostępu do pamięci
(ang. memory wall)

- Przyrost prędkości zegarów procesorów (40 % rocznie) przewyższa wzrost prędkości dostępu do jednostki pamięci (10 % rocznie). Efektem jest konieczność korzystania z szybszych systemów pamięci – pamięci podręcznej. Efektywność systemów jest zależna od stopnia obsługi żądań dostępu do pamięci w ramach szybkiej pamięci podręcznej.
- Zasada lokalności obowiązująca przy projektowaniu algorytmu równoległego prowadzi do kodu efektywnie korzystającego z pamięci podręcznej - Korzyści z równoległości dla sekwencyjności

Lokalność przetwarzania, dostępu do danych

- W systemach równoległych ogólna przepustowość procesory-pamięć jest większa niż w systemach jednoprocesorowych z powodu:
  - większej pamięci podręcznej,
  - równoległego dostępu procesorów do pamięci.
- Wynik: podział na (równolegle realizowane) podproblemy umożliwia większą lokalność przetwarzania i wyższą efektywność dostępu do pamięci, a dodatkowo wzrost rozmiaru pp (w systemie równoległym) prowadzi do efektywnego przetwarzania (możliwe że skrócenia czasu w większym stopniu niż wynikający z użytej liczby procesorów).

Przetwarzanie danych

- W pewnych zastosowaniach komputerów wielkie ilości danych muszą być przeanalizowane, a ilość danych powoduje nierealność zbierania ich w jednej lokacji. Sytuacja ta prowadzi naturalnie do rozproszonego i równoległego przetwarzania.

Jak pisać programy równoległe?

1. **Równoległość na poziomie zadań** – podział wśród procesorów zadań realizowanych dla rozwiązania problemu.
2. **Równoległość na poziomie danych** – podział danych używanych w rozwiązaniu problemu, każdy procesor realizuje podobne operacje na swoich danych.

Rodzaje równoległości: przykład 1– poprawianie prac egzaminacyjnych

- Egzamin:
  - 8 pytań,
  - 150 studentów
- Poprawianie prac egzaminacyjnych:
  - 4 asystentów

Rodzaje równoległości przykład 1– poprawianie prac egzaminacyjnych

- Asystent A poprawia Grupę I1
- Asystent B poprawia Grupę I2
- Asystent C poprawia Grupę I3
- Asystent D poprawia Grupę I4

•Równoległość danych

LUB

- Asystent A poprawia pytania 1 i 2
- Asystent B poprawia pytania 3 i 4
- Asystent C poprawia pytania 5 i 6
- Asystent D poprawia pytania 7 i 8

•Równoległość zadań

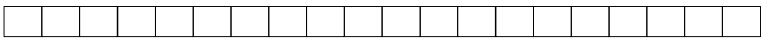
Cele zajęć

- Koncepcje – patrzeć na problemy poprzez „równoległe okulary”
- Algorytmy - różne zasoby, różne cele
- Języki – mniej sterowania przepływem obliczeń, więcej niezależności
- Sprzęt – optymalizacja komunikacji i ukrycie komunikacji,
- Programowanie – opis przetwarzania bez użycia pojęcia kolejności

Przetwarzanie równoległe, a rozproszone

Pojęcie	Przetwarzanie Równoległe	Przetwarzanie Rozproszone
Cel	Prędkość	Wygoda, elastyczność
Interakcje	Częste	Rzadkie
Ziarnistość	Małe ziarna zadań	Duże ziarna zadań
Niezawodność	Zakładana	Nie zakładana

Rozważmy prosty problem



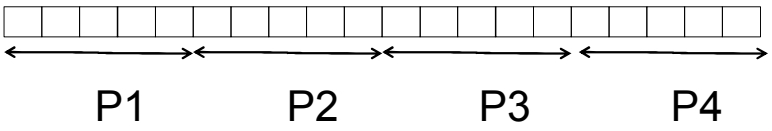
Sumowanie elementów tablicy

```
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 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 – rezultat nadpisania odczytanych i wykorzystywanych przez inny procesor wartości zmiennej `suma`
- Rozwiązanie:
  - Synchronizacja dostępu do zmiennej

Równoległe sumowanie elementów – poprawny program liczy się powoli

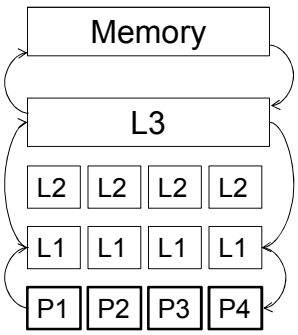
Rozwiązanie 2:

```
a1 = __rdtsc();
#pragma omp parallel 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

```
#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 danych pomiędzy poziomami pamięci: procesorem, pamięcią podręczną L1 jednego rdzenia, współdzieloną pp L3, L1 innego rdzenia i do innego procesora
- `suma` jest modyfikowana i nie jest lokalna !! – nie jest w sposób ciągły dostępna w rejestrze

Równoległe sumowanie elementów

Rozwiązanie 3:

Większa lokalność przetwarzania – prywatna suma

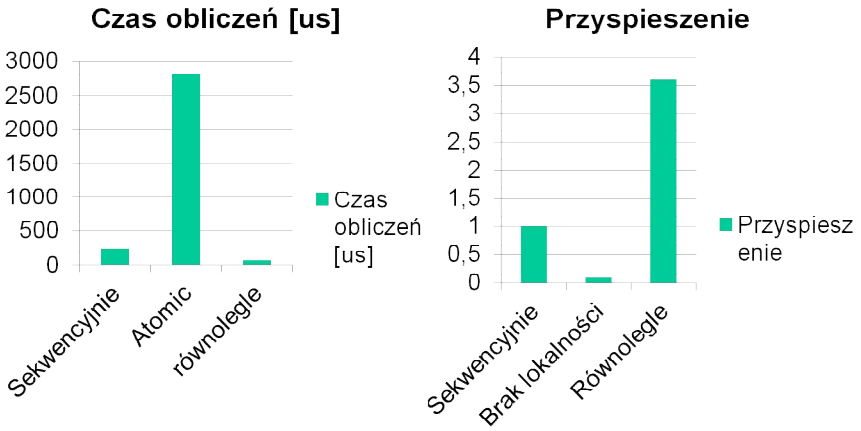
```
#pragma omp parallel
        {
        float sumal=0.0;
#pragma omp for
        for(int i=0;i<MAX;i++) sumal+=table[i];
#pragma omp atomic
                suma+=sumal;
        }
```

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 elementów



Cele w programowaniu równoległym

- Skalowalność: kolejne procesory mogą zostać dodane dla szybszego rozwiązania problemu
- Wydajność: efektywnie wykorzystać dostępne procesory - przyspieszenie
- Przenośność: oprogramowanie nadające się do przenoszenia między systemami