Podstawy OpenMP

Zbigniew Koza

Wydział Fizyki i Astronomii
Uniwersytet Wrocławski

Wrocław, 12 listopada 2012

Spis treści

Część 1

Do czego służy OpenMP?

- Środowisko porogramistyczne (API) umożliwiające programowanie współbieżne w systemach z pamięcią dzieloną
- Przenośne
- Podlegające standaryzacji
- Łatwe do nauki, łatwe do stosowania
  - Automatyzacja tworzenia i niszczenia wątków
  - Automatyzacja przydziału pamięci „prywatnej” dla wątków
- Umożliwia stopniowe zrównoleglanie istniejącego kodu szeregowego
- Umożliwia traktowanie tego samego kodu jako kodu równoległego bądź i szeregowgo

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

gdzie

- S – przyspieszenie „równoległe”
- P – liczba procesorów
- T_P – czas wykonania programu przez P procesorów
- P – liczba procesorów
- $f_{||}$ – ułamek tworzony przez zrównoleglony kod

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$
- Dla $f_{||} = 0.8$ i $P = 16$ mamy $S = 4$

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$
- Dla $f_{||} = 0.8$ i $P = 16$ mamy $S = 4$
- Dla $f_{||} = 0.8$ i $P = 32$ mamy $S = 4.4$

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$
- Dla $f_{||} = 0.8$ i $P = 16$ mamy $S = 4$
- Dla $f_{||} = 0.8$ i $P = 32$ mamy $S = 4.4$
- Dla $f_{||} = 0.8$ i $P \to \infty$ mamy $S \uparrow 5$

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$
- Dla $f_{||} = 0.8$ i $P = 16$ mamy $S = 4$
- Dla $f_{||} = 0.8$ i $P = 32$ mamy $S = 4.4$
- Dla $f_{||} = 0.8$ i $P \to \infty$ mamy $S \uparrow 5$
- Wniosek: dla „małych” $f_{||}$ zwiększanie P nie ma sensu

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$
- Dla $f_{||} = 0.8$ i $P = 16$ mamy $S = 4$
- Dla $f_{||} = 0.8$ i $P = 32$ mamy $S = 4.4$
- Dla $f_{||} = 0.8$ i $P \to \infty$ mamy $S \uparrow 5$
- Wniosek: dla „małych” $f_{||}$ zwiększanie P nie ma sensu
- Wniosek: należy zrównoleglać możliwie jak największą część kodu

Prawo Ahmdala

Prawo Ahmdala

$$S = \frac{T_1}{T_P} = \frac{1}{f_{||}/P + 1 - f_{||}}$$

- Dla $f_{||} = 0.8$ i $P = 4$ mamy $S = 2.4$
- Dla $f_{||} = 0.8$ i $P = 16$ mamy $S = 4$
- Dla $f_{||} = 0.8$ i $P = 32$ mamy $S = 4.4$
- Dla $f_{||} = 0.8$ i $P \to \infty$ mamy $S \uparrow 5$
- Wniosek: dla „małych” $f_{||}$ zwiększanie P nie ma sensu
- Wniosek: należy zrównoleglać możliwie jak największą część kodu
- Uwaga: $f_{||}$ może zależeć od P

Co to jest proces?

Proces

Proces to program załadowany do pamięci operacyjnej

Co to jest proces?

Proces

Proces to program załadowany do pamięci operacyjnej

- System operacyjny przydziela procesom zasoby
- System operacyjny separuje procesy (wirtualizacja pamięci operacyjnej)

Co to jest wątek?

Wątek

Wątek (ang. thread) jest to fragment programu wykonywany (współbieżnie) w ramach jednego procesu

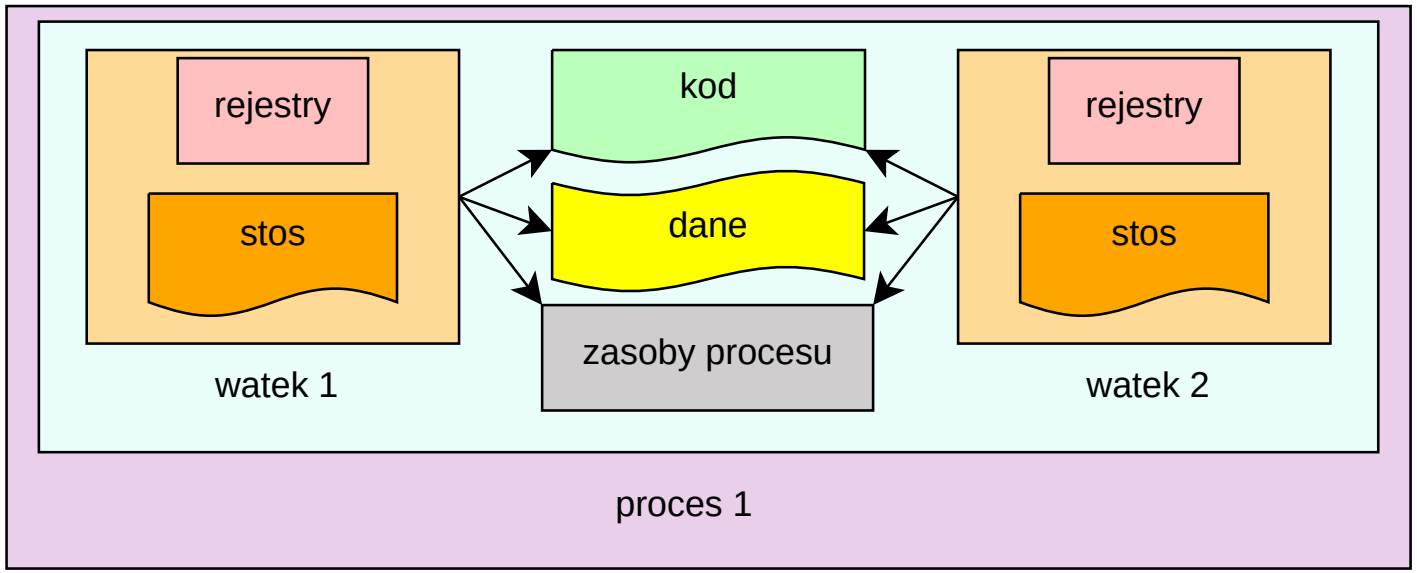
Co to jest wątek?

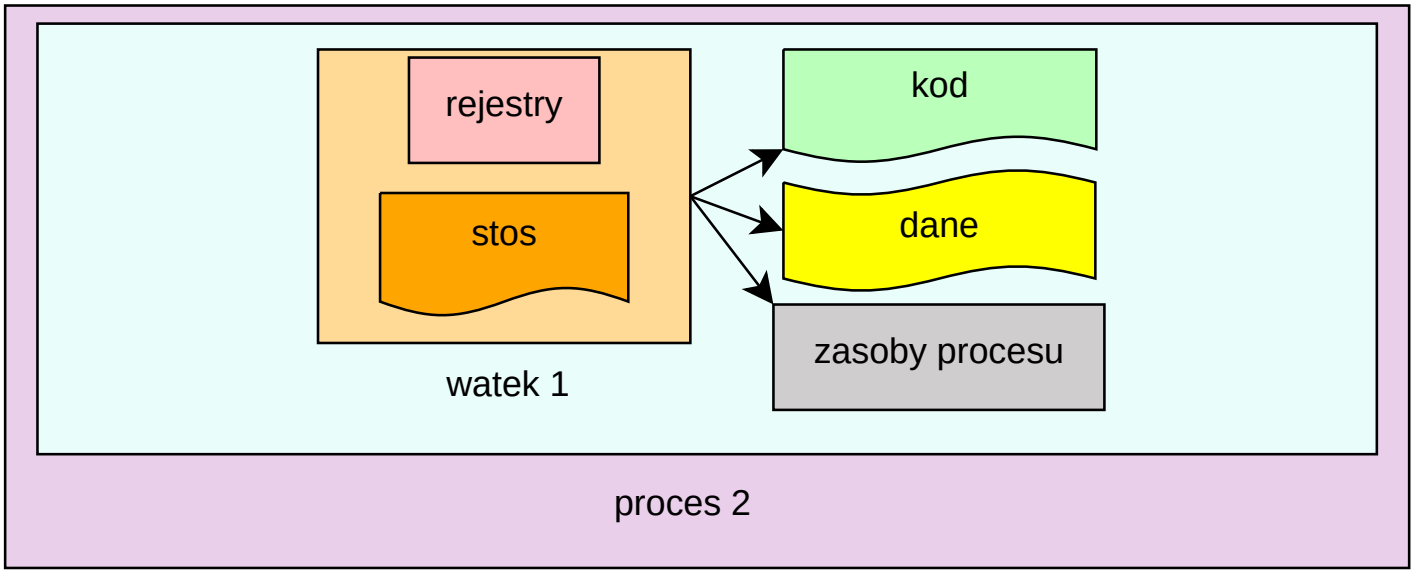
Wątek

Wątek (ang. thread) jest to fragment programu wykonywany (współbieżnie) w ramach jednego procesu

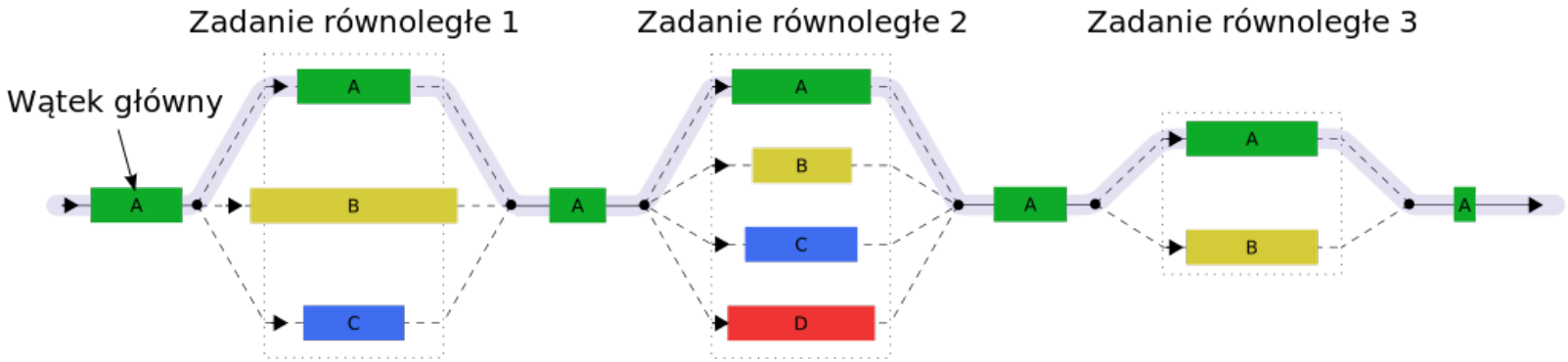
- W jednym procesie może istnieć wiele wątków
- Wątki procesu współdzielą wszystkie systemowe zasoby procesu, m.in. przestrzeń adresową, listy otwartych plików, gniazd, itp.
- Niewiele zasobów „prywatnych”: licznik programu, rejestry, stos,...
- Wątki można dość szybko tworzyć i niszczyć
- Wątki mogą się komunikować ze sobą bez pośrednictwa systemu operacyjnego (wspólny blok zmiennych globalnych)

Wątki i procesy





Fork and Join



- Zawsze istnieje wątek główny o identyfikatorze 0
- Grupa współbieżnych wątków to zespół (ang. team)

Na czym polega OpenMP?

- OpenMP wymaga od programisty:
  - identyfikacji fragmentów kodu, które mają być zrównoleglone
  - podania dodatkowych informacji, o jaki rodzaj zrównoleglenia chodzi i jak wątki mają ze sobą współpracować

Na czym polega OpenMP?

- OpenMP wymaga od programisty:
  - identyfikacji fragmentów kodu, które mają być zrównoleglone
  - podania dodatkowych informacji, o jaki rodzaj zrównoleglenia chodzi i jak wątki mają ze sobą współpracować
- w zamian za to OpenMP:
  - ukrywa przed programistą szczegóły zarządzania wątkami
  - realizuje podaną przez programistę strategię zrównoleglania kodu

Główne składniki OpenMP

- Dyrektywy kompliatora (`#pragma omp ...`)
- Biblioteka specjalnych funkcji
- Zmienne środowiskowe

Języki programowania

- FORTRAN
- C
- C++

Możliwości OpenMP

OpenMP posiada mechanizmy umożliwiające

- Tworzenie i niszczenie zespołów wątków wykonywanych współbieżnie
- Określanie sposobów dzielenia się pracą przez wątki
- Deklarowanie zmiennych współdzielonych i prywatnych
- Synchronizowanie pracy wątków (na wiele różnych sposobów)

Tworzenie zespołów wątków (fork/join)

```
#pragma omp parallel
```

- Obejmuje zakresem jedną instrukcję (zwykle blokową), po której następuje „join”
- Domyślnie uruchamia zespół wątków wykonujący identyczne instrukcje

Tworzenie zespołów wątków (fork/join)

```
#pragma omp parallel
```

- Obejmuje zakresem jedną instrukcję (zwykle blokową), po której następuje „join”
- Domyślnie uruchamia zespół wątków wykonujący identyczne instrukcje
- Dlatego zwykle potrzebne są dyrektywy specyfikujące podział pracy

Dyrektywy specyfikujące podział pracy

- W zasięgu dyrektywy `#pragma omp parallel`:

```
#pragma omp for
{
...
}
```

```
#pragma omp sections
{
...
}
```

- Poza dyrektywą `#pragma omp parallel` (notacja skrócona):

```
#pragma omp parallel for
...

#pragma omp parallel sections
...
```

Nie każdą pętlę można zrównoleglić

- ????

```
int tab[100];
tab[0] = 0;
#pragma omp parallel
for (int i = 1; i < 100; ++i)
   tab[i] = tab[i-1] + 1;
```

- OK

```
int tab[100];
tab[0] = 0;
#pragma omp parallel for
for (int i = 1; i < 100; ++i)
   tab[i] = i*(i+1)/2;
```

Model pamięci

Zmienne mogą być współdzielone przez wątki lub prywatne w wątku

- ```
  int i;
  #pragma omp parallel shared(tab) private(i)
  for (i = 1; i < 100; ++i)
     tab[i] = i*(i+1)/2;
  ...
  ```
- Użycie zmiennych prywatnych przyspiesza program
- Zmiany wartości zmiennych współdzielone stają się widoczne dla innych wątków w punktach synchronizacji
- Wątki mogą posiadać tymczasowo lokalne kopie zmiennych współdzielonych (pamięć cache)
  ```

Synchronizacja wątków

- Najtrudniejszy problem w programowaniu równoległym
- OpenMP oferuje m.in.:
  - bariery (w tym bariery domyślne),
  - sekcje krytyczne,
  - operacje atomowe,
  - operacje wykonywane przez pierwszy wątek

Przykład: mnożenie macierzy przez wektor

```
void mul(int size, double *a, const double *b, const double *c)
{
   int i, j;
#pragma omp parallel for \
          default(none) shared(size, a, b, c) private(i, j)
   for (i = 0; i < size; ++i)
   {
      a[i] = 0.0;
      for (j = 0; j < size; ++j)
         a[i] += b[i*size + j] * c[j];
   } /*--- End of omp parallel ---*/
}
```