Politechnika Poznańska

Wydział Informatyki

Przetwarzanie Równoległe

Równoległe sumowanie komórek pamięci za pomocą wielu wątków przetwarzania

Autorzy:
Adam Szczepański
Mateusz Czajka

Prowadzący:
dr Rafał Walkowiak

11 lutego 2014

Spis treści

1 Informacje o projekcie

1.1 Dane autorów

Mateusz Czajka	106596
Adam Szczepański	106593

1.2 Historia projektu

1. Jest to pierwsza wersja projektu. Dokumentacja elektroniczna została przesłana w dniu 20 stycznia 2013.

2 Wstęp

2.1 Opis problemu

Głównym założeniem projektu było zapoznanie się biblioteką OpenMP na podstawie równoległego sumowania komórek tablicy. W ramach projektu zrealizowaliśmy 4 algorytmy sekwencyjne oraz 2 algorytmy zrównleglone. Celem zastosowania czterech różnych algorytmów sekwencyjnych było zbadanie wpływu sekcyjności pamięci podręcznej, wyprzdzającego pobrania danych do pamięcy podręcznej oraz kolejności uszeregowania pętli na czas realizacji zadania. W przypadku algorytmów zrównoleglonych badaliśmy wpływ kolejności uszeregowania pętli na końcowy rezultat.

Nasze badania podzieliliśmy na 3 spójne części. Kolejno badaliśmy

- rozmiar danych
- sekcyjność pamięci
- wyprzedzające pobranie

na czas realizacji problemu.

Sam problem sprowadzał się do zsumowania wartości komórek w tabeli. Dla zachowania czytelności w kolejności pętli zastosowaliśmy tablicę dwuwymiarową. Ponieważ sam problem jest prosty obliczeniowo zmuszeni byliśmy do stosowania maksymalnego rozmiaru tablicy tj $[2^{28} - wierszy]$ na $[2^4 - kolumn]$.

2.2 Punkt odniesienia (algorytm sekwencyjny - kolejność ij)

Punktem odniesienia dla wszystkich algorytmów był podstawowy algorytm sekwencyjny w którym sumowaliśmy elementy tablicy wierszami.

```
__declspec(noinline) int sum_ij() {
  int sum = 0;
  for (int i=0; i<ROWS; i++) {
    for (int j=0; j<COLS; j++) {
      sum += tab[i][j];
    }
  }
  return sum;
}
```

2.3 Badane algorytmy

2.3.1 Algorytm sekwencyjny - kolejność ji

```
__declspec(noinline) int sum_ji() {
  int sum = 0;
  for (int j=0; j<COLS; j++) {
    for (int i=0; i<ROWS; i++) {
      sum += tab[i][j];
    }
  }
  return sum;
}
```

2.3.2 Algorytm zrównoleglony - kolejność ij

```
__declspec(noinline) int sum_par_ij() {
  int sum = 0;
  int i;
#pragma omp parallel for default(none) shared(tab) private(i) reduction(+:sum)
  for (i=0; i<ROWS; i++) {
    for (int j=0; j<COLS; j++) {
      sum += tab[i][j];
    }
  }
  return sum;
}
```

2.3.3 Algorytm zrównoleglony - kolejność jj

```
__declspec(noinline) int sum_par_ji() {
  int sum = 0;
  int j;
#pragma omp parallel for default(none) shared(tab) private(j) reduction(+:sum)
  for (j=0; j<COLS; j++) {
    for (int i=0; i<ROWS; i++) {
      sum += tab[i][j];
    }
  }
  return sum;
}
```

2.3.4 Algorytm na sekcyjność pamięci

```
__declspec(noinline) int sum_sec() {
  int sum = 0;
  for (int j=0; j<COLS; j++) {
    for (int k=0; k<CACHE_LINES_ON_PAGE; k++) {
      for (int i=k; i<ROWS; i+=CACHE_LINES_ON_PAGE) {
        sum += tab[i][j];
      }
```

```
    }
  }
  return sum;
}
```

2.3.5 Algorytm na pobranie z wyprzedzeniem

```
int tmp;

__declspec(noinline) int sum_pf() {
  int sum = 0;
  int i;
  for (i=0; i<ROWS-1; i++) {
    for (int j=0; j<COLS; j++) {
      sum += tab[i][j];
      tmp = tab[i+1][j];
    }
  }

  for (int j=0; j<COLS; j++) {
    sum += tab[i][j];
  }
  return sum;
}
```

Spis rysunków

Spis tablic