Równoległe algorytmy sortowania

Algorytmy sortowania

- Algorytmy sortowania dzielą się na
  - wewnętrzne (bez użycia pamięci dyskowej)
  - zewnętrzne (dla danych nie mieszczących się w pamięci operacyjnej)
- Innym podziałem algorytmów sortowania jest wyróżnienie
  - algorytmów opartych wyłącznie na porównywaniu elementów
    - optymalna złożoność obliczeniowa $\Theta(n \log n)$
  - algorytmów wykorzystujących dodatkową wiedzę o elementach (np. zakres wielkości)
    - optymalna złożoność obliczeniowa $\Theta(n)$

Sortowanie szybkie

- Sortowanie szybkie jest jednym z najpopularniejszych algorytmów sortowania ze względu na:
  - optymalną złożoność oczekiwaną $\Theta(n \log n)$
  - prostotę
  - małe wymagania pamięci (sortowanie w miejscu)
- Sortowanie szybkie wykorzystuje strategię dziel i rządź
- Zrównoleglenie sortowania szybkiego jest niezwykle łatwe

Sortowanie szybkie – wersja równoległa OpenMP

```
void qs(int *x, int l, int h) {

    int newl[2], newh[2], i, m;

    m = podziel(x,l,h);
    newl[0] = l; newh[0] = m-1;
    newl[1] = m ; newh[1] = h;

    #pragma omp parallel
    {
       #pragma omp for nowait
       for (i = 0; i < 2; i++)
          qs(x, newl[i], newh[i]);
    }
}
```

Równoległe sortowanie szybkie

- Analiza złożoności obliczeniowej pokazuje, że wykonanie równoległe niewiele poprawia czas realizacji algorytmu:
  - $T_{sekw}(n) = 2T_{sekw}(n/2) + \Theta(n) = \Theta(n\log n)$
  - $T_p(n,n) = T_p(n/2,1) + \Theta(n) = \Theta(n)$
- Kluczem do zwiększenia wydajności równoległej jest zrównoleglenie procedury *podziel*
- Istnieją algorytmy, które dla p istotnie mniejszych od n potrafią uzyskać oczekiwaną złożoność wykonania równoległego równą
  - $T_p(n,p) = \Theta(n/p \log(n/p))$
- Pozostaje problem złożoności pesymistycznej, której prawdopodobieństwo rośnie przy realizacji równoległej

Równoległe sortowanie bąbelkowe

- Sortowanie bąbelkowe - złożoność obliczeniowa $\Theta(n^2)$
- Sekwencja operacji zamiany dwóch sąsiadujących elementów (operacja porównaj-zamień, *compare-exchange*)
  - sekwencja jest nieusuwalnie sekwencyjna
- Warianty algorytmu, o wyższej złożoności obliczeniowej (także $\Theta(n^2)$) dające się łatwo zrównoleglić
- Sortowanie nieparzyste-parzyste (*odd-even transposition*)
  - zrównoleglenie fazy zamiany
  - operacja porównaj-zamień zamienia się w operację porównaj-rozdziel (*compare-split*)

Sortowanie nieparzyste-parzyste

```
nlocal = n/p;
qsort(data, nlocal, sizeof(int), IncOrder); // lokalne sortowanie
if (id % 2 == 0) {    oddrank = id-1;   evenrank = id+1;
} else {              oddrank = id+1;   evenrank = id-1;  }
if((oddrank==-1)||(oddrank==p))   oddrank  = MPI_PROC_NULL;
if((evenrank==-1)||(evenrank==p)) evenrank = MPI_PROC_NULL;
for(i=0; i<p-1; i++) {
    if (i%2 ==1) {   /* faza nieparzysta */
        MPI_Sendrecv( data, nlocal, MPI_INT, oddrank, 1,
     rdata, nlocal, MPI_INT, oddrank, MPI_COMM_WORLD, &status);
    } else {             /* faza parzysta */
        MPI_Sendrecv( data, nlocal, MPI_INT, evenrank, 1,
     rdata, nlocal, MPI_INT, evenrank, MPI_COMM_WORLD, &status);
    }
    CompareSplit(nlocal, data, rdata, wspace);
}
```

Sortowanie nieparzyste-parzyste

- Złożoność równoległa sortowania nieparzyste-parzyste
  - $T_p(n,p) = \Theta(n/p \log(n/p)) + \Theta(n)$
  - *p* faz, ponieważ w jednej fazie element może przesunąć się tylko o jedną pozycję
  - warianty algorytmu, które starają się zmniejszyć liczbę faz, redukując oczekiwaną złożoność algorytmu równoległego do standardowej wartości (dla $p < n$)
    - $T_p(n,p) = \Theta(n/p \log(n/p))$

Sortowanie kubełkowe

- Sortowanie kubełkowe
  - elementy rozłożone równomiernie w znanym przedziale
  - rozdzielenie wszystkich elementów do kubełków i posortowanie wewnątrz kubełków
  - czas działania sekwencyjnego dla *m* kubełków
    - $T_{sekw}(n,m) = \Theta(n \log(n/m))$
  - dla osiągnięcia optymalnej złożoności liczba kubełków powinna być rzędu liczby elementów

Sortowanie kubełkowe

- Zrównoleglenie algorytmu
  - przydzielenie każdego kubełka innemu procesorowi
- W pierwszej fazie procesory sortują przydzielone sobie dane, w drugiej przesyłają do odpowiednich kubełków (procesorów), w trzeciej sortują ostatecznie w kubełkach
- Czas realizacji równoległej w przypadku równomiernego rozłożenia elementów w kubełkach
  - $T_p(n,p) = \Theta(n/p \log(n/p))$
- Założenie równomiernego rozkładu może nie być realistyczne, można wtedy zmodyfikować algorytm, tak aby podział na kubełki pozostał optymalny (tzw. algorytm sortowania próbkowego, *sample sort*)