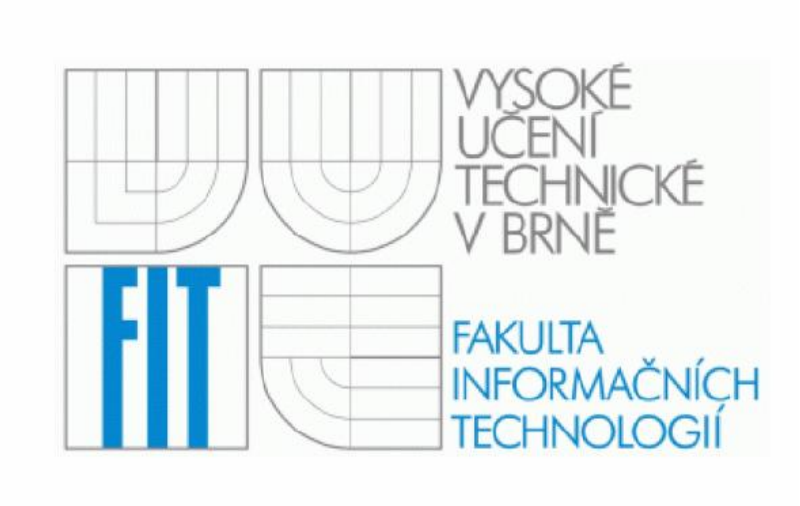

Paralelní a distribuované algoritmy
1MIT – 2007/2008

Implementace algoritmu odd-even transposition sort

Autor
Jiří Hrazdil, xhrazd06

Zadání

Implementujte algoritmus Odd-even transposition sort v jazyce PM2.

Komentovaný výpis algoritmu

Procedura *init* načte ze vstupu počet procesorů a pak v cyklu řazená čísla, která ukládá do sdíleného pole *values*.

```
procedure init;
var p, i : word;
begin
  read(p);  // nacteni poctu vstupnich hodnot
  P := p;   // inicializace poctu procesoru
  for i := 1 to p do
    read(values[i]);  // nacitani razenych cisel a ukladani do pole
  end;
end init;
```

Procedura *sortfirst* testuje paralelně pro všechny liché procesory, které mají souseda vpravo, zda-li je hodnota souseda menší, než hodnota testovaného procesoru, a v případě, že ano, tak hodnoty prohodí.

```
procedure sortfirst; // provede serazeni pro liche procesory
var i : word;
    temp : word;
begin
  par i := 1 to P sync do   // iteruj synchronne pres vsechny procesory
    if (i mod 2 = 1) then   // pro kazdy lichy procesor
      if (i + 1 <= P) then  // ktery ma vpravo souseda
        if (values[i+1] < values[i]) then // pokud je nase hodnota vetsi
nez hodnota souseda
          temp := values[i];  // prohod hodnoty
          values[i] := values[i+1];
          values[i+1] := temp;
        end;
      end;
    end;
  end;
end sortfirst;
```

Procedura *sortsecond* testuje paralelně pro všechny sudé procesory, které mají souseda vpravo, zda-li je hodnota souseda menší, než hodnota testovaného procesoru, a v případě, že ano, tak hodnoty prohodí.

```
procedure sortsecond;
// provede serazeni pro sude procesory
var i : word;
    temp : word;
begin
  par i := 2 to P sync do   // iteruj synchronne pres vsechny procesory
    if (i mod 2 = 0) then // pro kazdy sudy procesor
      if (i + 1 <= P) then  // ktery ma vpravo souseda
        if (values[i + 1] < values[i]) then // pokud je nase hodnota vetsi
nez hodnota souseda
          temp := values[i];    // prohod hodnoty
          values[i] := values[i + 1];
          values[i + 1] := temp;
        end;
```

```
      end;
    end;
  end;
end sortsecond;
```

Procedura *finish* vypíše na výstup jak počet řazených hodnot, tak seřazené hodnoty.

```
procedure finish;
var i : word;
begin
  write(P); // vypis pocet procesoru/hodnot
  chwrite(32);
  for i := 1 to P do
    write(values[i]);  // vypis serazene hodnoty
    chwrite(32);
  end;
end finish;
```

Hlavní smyčka programu vykoná n/2-krát oba kroky řazení. (V případě lichého počtu prvků na vstupu je ale poslední volání procedury *sortsecond* zbytečné.)

```
begin
for i := 1 to (P-1) / 2 + 1 do // provadej n/2-krat (konstrukce zaruci
spravny pocet iteraci i v pripade licheho poctu prvku)
  sortfirst();
  sortsecond();
end;
@CLOCK
end prl1.
```

Popis algoritmu

Odd-even transposition sort je řadicí algoritmus podobný bubble sortu. Díky své konstrukci je vhodný k paralelnímu běhu. Vstupem algoritmu je vektor hodnot, výstupem seřazený vektor hodnot. Nejdříve všem procesorům přiřadíme postupně n řazených hodnot v_i ($i = 1, \ldots, n$). Algoritmus probíhá cyklicky ve dvou krocích. V prvním kroku se pracuje s lichými procesory, ve druhém kroku se sudými procesory. V každém kroku procesor porovná svoji hodnotu v_i s hodnotou procesoru vpravo od sebe v_{i+1} (např. svého následovníka při použití jednosměrně vázaného seznamu) a v případě, že je $v_i > v_{i+1}$, dojde k prohození obou hodnot. Stejným způsobem se postupuje i ve druhém kroku se sudými procesory. Tento postup nám zaručí, že nikdy k jednomu procesoru nepřistupuje více jiných současně. Maximálně po n krocích je posloupnost seřazena.

Teoretická časová složitost

Při každém ze dvou kroků se paralelně provádí jedno porovnání a dva přenosy. Tyto operace jsou nezávislé na počtu prvků, jejich provedení má konstantní časovou složitost. Celkem se provede maximálně n kroků, takže složitost algoritmu je lineární:

$$t(n) = O(n)$$

Experimenty

Provedl jsem celkem 7 měření. Počty prvků, počet hodinových taktů a jejich poměry jsou uvedeny v následující tabulce.

počet prvků / procesorů	počet hodinových taktů	počet taktů na 1 prvek
5	2830	560
10	4644	461
15	7365	489
20	9179	457
30	13714	456
40	18249	455
50	22784	455

Z tabulky vidíme, že při vzrůstajícím počtu prvků se doba provádění zvětšuje lineárně.
U dvou posloupností, ve kterých byl lichý počet prvků, vidíme nelinearitu – ta je způsobena prováděním jednoho zbytečného kroku řazení při lichém počtu prvků (např. u patnáctiprvkové posloupnosti stačí 15 kroků, ale program jich provádí šestnáct).

Hodnoty naměřené při experimentech odpovídají teoretické časové složitosti, implementace algoritmu je správná.

Závěr

Projekt poskytl možnost praktického seznámení s programovacím jazykem PM2 a s programováním pro architekturu PRAM a osvěžil znalost funkce algoritmu Odd-even transposition sort.