Paralelní a distribuované algoritmy – dokumentace
Projekt 1 : Pipeline merge sort

Bc. Marek Kidon

7. dubna 2015

Dokumentace k 1.projektu do předmětu Paralelní a distribuované algoritmy (PRL). V dokumentu je rozebráno zadaní, popis řešení, analýza algoritmu PMS, časová složitost, naměřené hodnoty času potřebného pro seřazení v závislosti na počtu prvků a komunikace mezi *sortery*. Časová složitost je zobrazena formou grafu a komunikace je zobrazena za použití *Message Sequence Chart* diagramu.

1 Zadání projektu

Pomocí knihovny Open MPI implementujte v jazyce C/C++ algoritmus **Pipeline merge sort**.

Vstup

Soubor *numbers* obsahující čísla velikosti 1 byte, která jdou bez mezery za sebou.

Výstup

Výstup na stdout se skládá ze dvou částí:

1. Jednotlivé načtené hodnoty v jednom řádku oddělené mezerou (vypsat po načtení prvním procesorem).
2. Jednotlivé seřazené hodnoty oddělené novým řádkem (od nejmenšího po největší).

Postup řešení

Vytvořte testovací skript test, který bude řídit testování. Tento skript bude mít tyto vlastnosti:

1. Bude pojmenován *test* nebo *test.sh*.
2. Bude přijímat 1 parametr a to *pocet_hodnot*.

Skript vytvoří podle velikosti parametru *pocet_hodnot* soubor *numbers* s náhodnými čísly a následně spustí program s počtem procesorů *log_2(pocet_hodnot)+1*. Skript nakonec smaže vytvořenou binárku a soubor *numbers*.

2 Analýza algoritmu

Algoritmus řadí za pomocí *lineárního pole procesorů*. Data lineárně prochází polem procesorů, každý procesor P_i sloučí 2 seřazené posloupnosti délky 2^{i-2} do jedné posloupnosti délky 2^{i-1}. Všechny procesory beží paralelně. Jednotlivé procesory jsou spojeny frontami, 2 vstupní a 2 výstupní. Výjimku tvoří první a poslední procesor, kde první má jednu vstupní frontu a poslední jen jednu výstupní.

- Každý procesor začíná, když má v první frontě seřezenou posloupnost alespoň délky 2^{i-2} a v druhé alespoň 1 prvek.
- P_i začne svou činnost v cyklu

$$1 + \sum_{j=0}^{i-2} 2^j + 1 = 2^{i-1} + i - 1$$

.

- Algoritmus skončí za $(n-1)+2^{i-1}+i-1$ cyklů.

Pro samotné sloučení 2 posloupností stačí operace porovnání. Není třeba žádného seřazování, neboť vstupní posloupnosti již seřazené jsou. Stačí tedy porovnávat vždy jen ty prvky, které se nachází ve frontě na prvním místě. A do výstupní fronty zařadit ten, jež je větší (menší) v případě, že řadíme sestupně (vzestupně). Volba výstupní fronty probíha dle počtu již odeslaných čisel, a to za pomocí následující operace zbytkových tříd : $(counter/base)\%2$, kde *counter* je počet již odeslaných prvků a *base* je základ dle vzorce 2^{i-2}.

3 Složitost a Optimálnost

Časová složitost algoritmu:$t(n)=O(n)$
Algoritmus je **optimální**:$c(n)=t(n).p(n)=O(n.\log n)$

4 Naměřené hodnoty

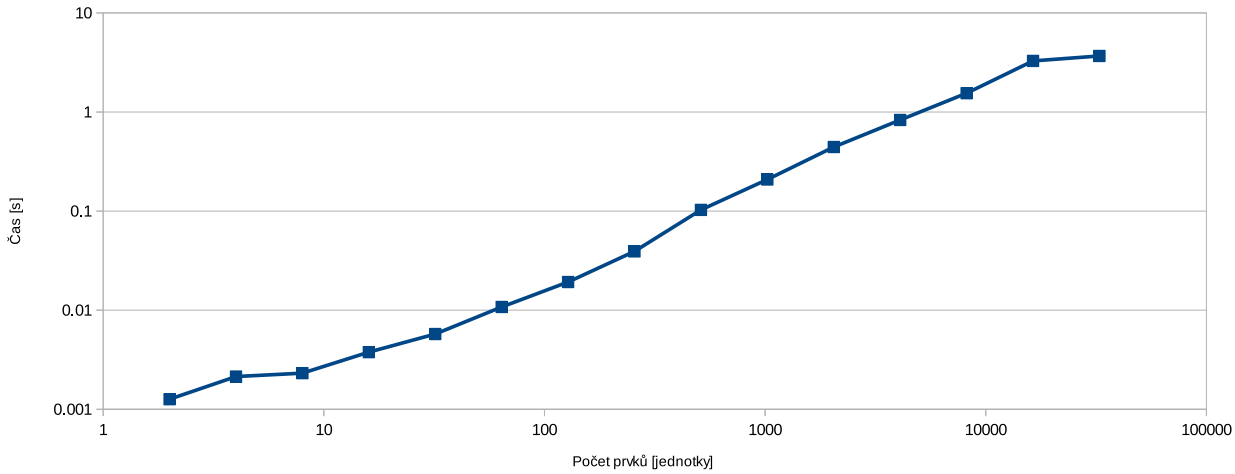
Měření bylo provedeno za použití knihovny *chrono*, a měřil se čas potřebný k seřazení dané posloupnosti měřeno od začátku práce prvního *sorteru* po dokončení práce posledního. Výsledky jsou dané v časových jednotkách, mohou se tedy při ruznách bězích lišit, kvůli přepínání kontextu operačního systému. Celý test byl proveden při stabilním zatížení stroje a pokusy nad jednolivými vstupními sekvencemi o stejné délce byly provedeny několikrát a aritmeticky zprůměrovány. O přes výše uvedenou nepřesnost, závislost mezi jednotlivými hodnotami v tabulce odpovídá a na výslednou křivku nemá vliv.
Pro měření mělo 2 specifika odchylující se od zadání. Pro generovnání byl namísto zdroje */dev/random* použit */dev/urandom*. Veškerý výstup na stdout byl přesměrován do */dev/null*.

počet prvků	2	4	8	16	32	64	128	256	512
čas[s]	0.00126	0.00213	0.00231	0.00377	0.00574	0.01076	0.01923	0.03917	0.10286

počet prvků	1024	2048	4096	8192	16384	32768
čas[s]	0.10987	0.44353	0.83116	1.54915	3.27791	3.67929

Tabulka 1: Přehled naměřených časů v závislosti na počtu prvků.

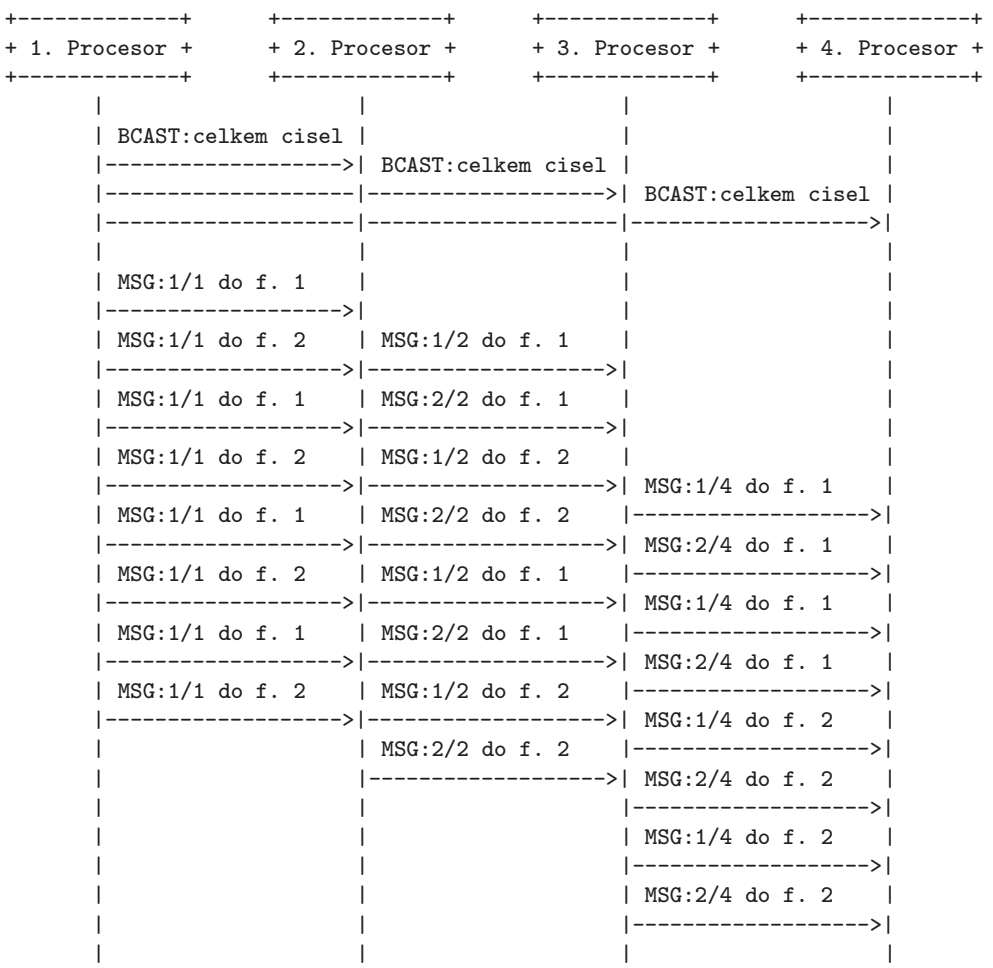
Na obrázku 1 je graficky znázorněn vztah mezi počtem prvků a časem potřebným k jejich seřazení.



Obrázek 1: Závislost počtu prvků na časem nutném k jejich seřazení.

5 Komunikace mezi procesory

Procesory mezi sebou používají velmi jednoduchý komunikační protokol. Aby se předešlo komplikovaným scénářům, Každý procesor komunikuje pouse se svým následníkem, nikoli se svým předchůdcem. Je všal nutné, aby při zasílání zprávy, procesor vždy vědel, do které fronty následníka má číslo umístit. Na počátku první procesor po zpracování vstupu, rozešle všem procesorům informaci o celkovém počtu prvků. Ten je použit jako ukončovací podmínka. Poté začína samotné řazení. Prvky jsou vybírány z front pomocí operace <= v případě, že jsou obě fronty neprázdné nebo je prvek vybrán bez porovnání, v případě, že opačná fronta je prázdná.



Obrázek 2: Ukázka zasílání zpráv mezi prvními čtyřmi procesory. Diagram by mohl pokračovat pro další procesory analogicky.