

Paralelní a distribuované algoritmy – dokumentace

Paralelní celulární automat

Bc. Jaroslav Sendler, xsendl00
xsendl00@stud.fit.vutbr.cz

11. dubna 2012

Dokumentace k 2.projektu do předmětu Paralelní a distribuované algoritmy (PRL). Obsahuje popis zadání, rozbor a analýzu Paralelního celulárního automatu. V závěru dokumentu se nachází komunikační protokol mezi „procesory“ (způsob zasílání zpráv). Pro vizualizaci je využit sekvenční diagram.

1 Zadání

Pomocí knihovny Open MPI implementujte celulární automat, který bude využívat paralelního prostředí pro urychlení výpočtu. Celulární automat bude implementovat pravidla hry *Game of life*.

Vstup: Soubor „lattice“ reprezentující mřížku automatu a obsahující binární číslice 0,1, kde 0 znamená mrtvou a 1 znamená živou buňku. Číslice 0 a 1 budou uspořádány do obdélníkové matice, kde každý řádek bude zpracováván právě jedním procesorem (navíc můžete použít jeden řídicí/synchronizační procesor). Zároveň můžete počítat s tím, že všechny řádky jsou stejně dlouhé. Následuje příklad, jak vypadá soubor *lattice*. Příkladem budiž:

```
00000000
00111000
01110000
00000000
```

Výstup: Na standardní výstup vypište stav matice po požadovaném počtu kroků a to tak, že každému řádku bude předcházet id procesoru a dvojtečka. Tento formát je zvolen z toho důvodu, že procesory budou hodnoty vypisovat v náhodném pořadí (pro seřazení výstupu použijte utilitu *sort* ve spouštěcím skriptu, nezapomeňte vyřešit dvoumístná id procesorů) Přesný formát výstupu je opět nutno dodržet kvůli strojové kontrole výstupu. Za nedodržení budou strhávány body. Příklad výstupu po 3 krocích:

```
0:00010000
1:01001000
2:01001000
3:00100000
```

Postup: Vytvořte testovací skript se jménem *test* nebo *test.sh*. Skript přijímá právě jeden parametr a to počet kroků. Skript spočte počet řádků (aby bylo jasné, kolik je třeba procesorů), přeloží a spustí program s parametrem *pocet_kroku*. Je vhodné spočítat i počet sloupců a předat ho programu kvůli načítání souboru (každý procesor si pak může načíst vlastní část souboru hodnot - vlastní řádek). Po načtení (ideálně paralelně - každý procesor svůj řádek) hodnot je proveden zadaný počet iterací podle předaného parametru a nakonec jsou na standardní výstup vypsány řádky jednotlivých procesorů. Vzhledem k tomu, že použijete utilitu *sort*, řádky budou seřazeny správně, jak mají být v matici.

2 Rozbor a analýza algoritmu

Algoritmus Pipeline merge sort pracuje s lineárním polem procesorů $p(n) = \log n + 1$, kde n je počet prvků určených k seřazení a $+1$ značí první procesor, který načítá vstupní data.

Základní body algoritmu: •

- Data nejsou uložena v procesorech, ale postupně do nich vstupují.
- Každý procesor spojuje dvě seřazené posloupnosti délky 2^{i-2} .
- Procesor P_i se spustí, když má na jednom vstupu posloupnost délky 2^{i-2} a na druhém 1, tedy začne $2^{i-2} + 1$ cyklů po procesoru P_{i-1} .
- P_i začne v cyklu

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

- P_i skončí v cyklu $(n - 1) + 2^{i-1} + i - 1$.
- Celý algoritmus skončí po $2n + \log n - 1$ cyklech.

Rozbor algoritmu

Algoritmus Pipeline merge sort pracuje s $\log_2(n) + 1$. Každý pracovní (vyjma prvního) z procesorů obsahuje dvě fronty s maximální délkou i , kde i je číslo procesoru ($i \geq 1$). Tedy pro druhý procesor $P_2 = 2$, třetí $P_3 = 3$, čtvrtý $P_4 = 4$ a tak dále. Algoritmus začíná první procesor, který načítá vstupní čísla a po jednom (bez porovnání) je zasílá druhému procesoru. Ten je ihned přijímá a střídavě je ukládá do první nebo do druhé fronty.

Při naplnění front správným počtem prvků (u P_2 ≥ 1 a P_2 ≥ 1) začínají pracovní procesory třídit data. Porovnávají se dvě čísla, jedno z první fronty, druhé z druhé fronty, a dle zadání se buď menší z nich nebo větší zašle vedlejšímu procesoru. Toto porovnání 2. procesor dělá pouze jednou, 3. procesor 2x, 4. 4x a tak dále. Ten jej přijme a také ukládá do front. Porovnávat začne při splnění dříve uvedených podmínek. Tímto způsobem pracují všechny procesory.

Výjimkou je poslední procesor, který při naplnění jedné fronty a druhé o obsahu alespoň jednoho čísla začíná porovnávat a výsledek (seřazená čísla) tiskne na standardní výstup. Při situaci, kdy má jednu frontu prázdnou, tak druhá obsahuje již seřazenou posloupnost prvků, a proto jsou postupně její prvky bez dalších operací posílány na výstup.

Fronty jednotlivých pracovních procesorů obsahují po seřazení n -tice. Každý další procesor spojuje/seřazuje posloupnosti (obsah obou front) předešlého procesoru.

3 Teoretická složitost algoritmu

časová složitost: $2n + \log n - 1$ cyklů, kde n je počet prvků k seřazení
tedy $t(n) = O(n)$

cena: $t(n) \cdot p(n) = O(n) \cdot (\log n + 1)$, kde $p(n)$ je počet „procesorů“
tedy $c(n) = O(n * \log n)$, což je optimální

4 Naměřené hodnoty

V tabulce 1 je zobrazena závislost mezi počtem vstupních prvků a časem potřebným k jejich seřazení. Výsledky byly zjištěny při experimentování s posloupnostmi různých délek. Pro každou hodnotu bylo provedeno 10 měření a následně udělán průměr. Měření probíhalo na školním serveru Merlin pomocí příkazu *time* při vypnutých výpisech. Do výsledných hodnot není započítáno generování vstupních prvků příkazem *dd*.

počet prvků	2	4	8	16	32	64	128	256	512
čas[s]	0,066	0,105	0,145	0,171	0,176	0,186	0,205	0,324	0,386

počet prvků	1024	2048	4096	8192	16384	32768
čas[s]	0,441	0,772	1,176	2,379	4,715	4,921

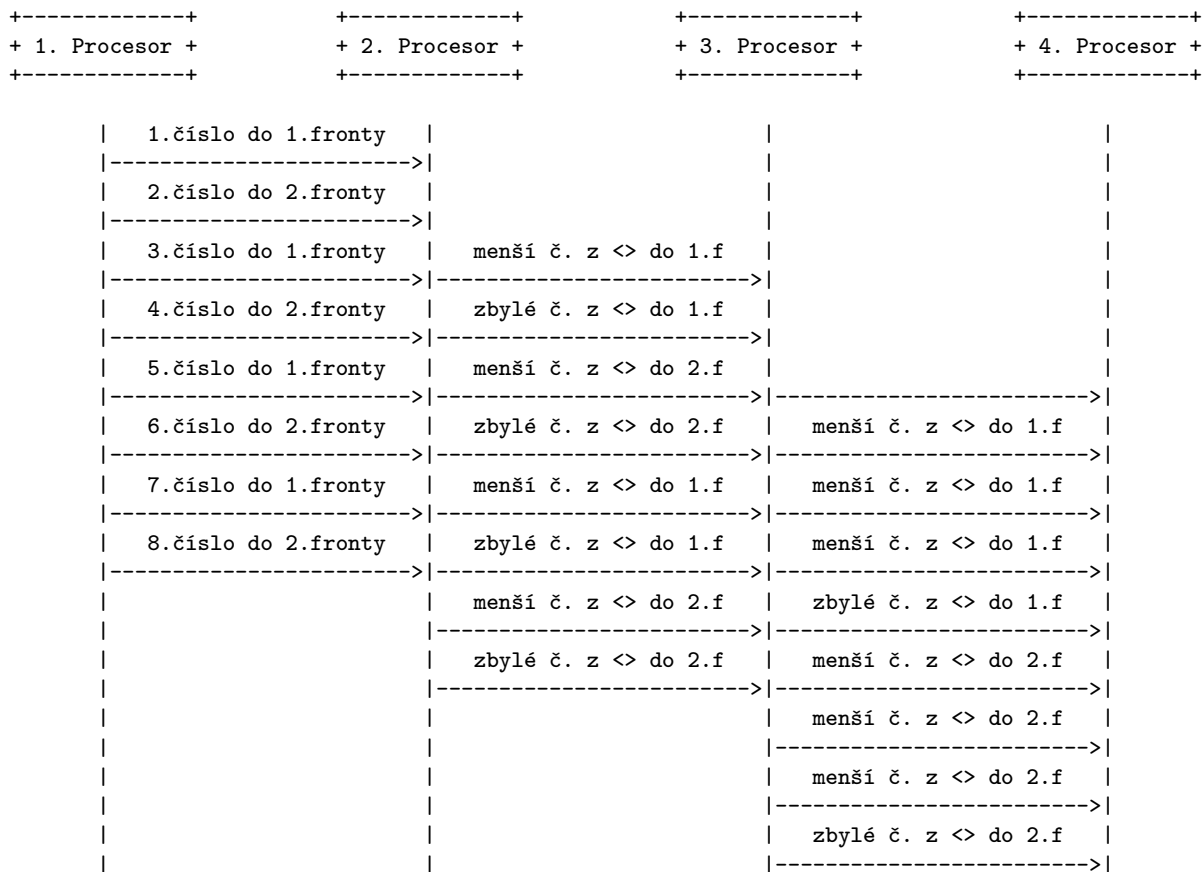
Tabulka 1: Přehled naměřených časů v závislosti na počtu prvků.

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

5 Komunikační protokol

Na obrázku 1 zobrazeném níže je pomocí sekvenčního diagramu znázorněna komunikace mezi jednotlivými procesory. Pro zjednodušení a lepší přehlednost je jako vstup pro Pipeline merge sort algoritmu použito 8 neseřazených čísel. Tedy počet procesorů je $\log_2(8) + 1$ což je 4.

V samotném diagramu jsou použity následující zkratky: č – > číslo, f – > fronta, <> – > porovnání dvou čísel.



Obrázek 1: Příklad komunikace 4 procesorů v algoritmu Pipeline merge sort.