Dokumentácia projektu 2 do predmetu Paralelní a distribuované algoritmy

Pokorný Fridolín
xpokor32@stud.fit.vutbr.cz

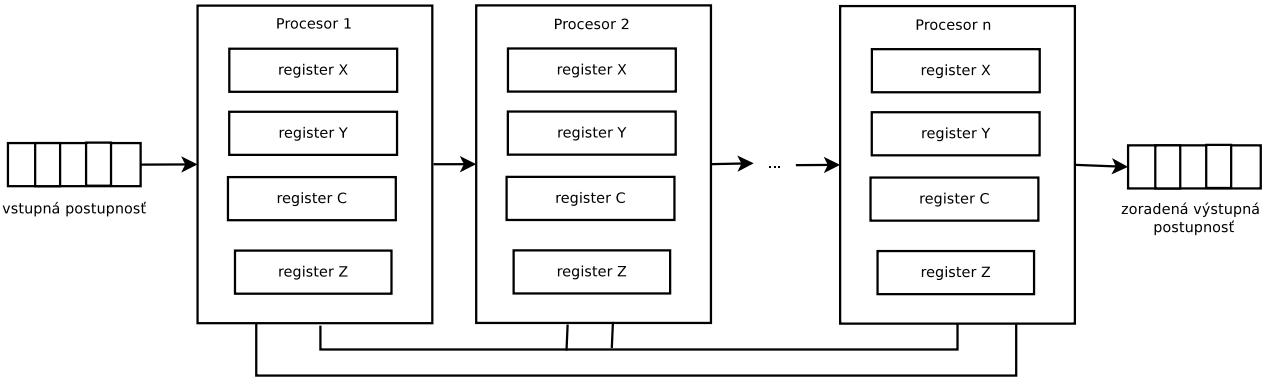
31. marca 2014

1 Zadanie

Cieľom projektu bolo vytvoriť pomocou knižnice *Open MPI* implementáciu algoritmu *Enumeration Sort* s lineárnou topológiou. Program je riadený testovacím skriptom `test.sh`, ktorý pomocou programu `dd` vytvorí binárny súbor, kde každy bajt reprezentuje radené číslo. Následne je tento binárny súbor vstupom pre program `es`, ktorý postupnosť zoradí využitím knižnice *Open MPI* použitím algormitmu *Enumeration Sort*.

2 Rozbor a analýza algoritmu

Enumeration Sort s lineárnou topológiou je paralelný radiaci algoritmus pracujúci na linárnej architektúre, na ktorej sú jednotlivé procesory spájané zbernicou. Naviac sú procesory lineárne spájané pomocou lineárneho spojenia, ktoré umožňuje prenesenie hodnoty registru do registra susedného procesoru. Každý procesor disponuje tromi registrami pre uchovanie hodnoty – register X, register Y a register Z. Naviac procesor obsahuje register C, ktorý uchováva relatívne poradie hodnoty v radenej postupnosti na základe uskutočnených porovnaní s ostatnými hodnotami radenej postupnosti. Celá architektúra je pre prehľadnosť znázornená na obrázku 1.



Obr. 1: Ilustrácia použitej architektúri pre n radených čisiel.

Princíp algoritmu:

1. Všetky registre C inicializuj na hodnotu 1.
2. Opakuj $2 \cdot n$ krát, pre $1 \leq k \leq 2 \cdot n$:
   (a) Pokiaľ vstup nie je vyčerpaný, vstupný prvok x_i vlož do registra X_i pomocou zbernice a do registra Y_1, pričom obsah všetkých registrov Y je lineárnym spojením posunutý doprava.

(b) Každý procesor s neprázdnymi registrami X a Y porovná obsahy týchto registrov a na základe pravdivosti relácie $X > Y$ inkrementuje register C.

(c) Ak je vstup vyčerpaný ($k > n$), procesor P_{k-n} pošle po zbernici obsah svojho registru X procesoru P_{Ck-n}, ktorý ho vloží do svojho registra.

3. V nasledujúcich n cykloch procesory posúvajú obsah svojich registrov doprava a procesor P_n produkuje zoradenú postupnosť.

Knižnica *Open MPI* poskytuje rozhranie, pomocou ktorého je možné simulovať komunikáciu prostredníctvom zbernice či lineárneho prepojenia niekoľkých procesorov. Pri zasielaní správ po zbernici je nutné uviesť ako príjemcu, tak aj odosielateľa správy. Každý procesor tak musí vedieť komu správu odosiela, ako aj každý procesor musí vedieť, od ktorého procesoru správu prijíma. Výnimkou je možnosť využitia *broadcast*.

V programe bolo upravené indexovanie vzhľadom pre použité indexovanie knižnice *Open MPI* ako aj na indexovanie v použitom programovacom jazyku `C++`. Registre sú preto indexované od 0, ako aj procesory, či prvky v radenej postupnosti. Vzhľadom na nutnosť udania adresáta pri zasielaní správ, bol využitý mechanizmus *broadcast* pre zaistenie očakávania správy na strane príjemncu hodnoty registra Z. Nakoľko procesor 0 pracuje hneď po načítaní hodnoty a vzhľadom na použitie mechanizmu *broadcast* (musia sa podieľať všetky procesory), pôvodný cyklus algoritmu pre $1 \leq k \leq 2 \cdot n$ bol rozdelený na dva cykly. Na podstate algoritmu to však nič nemení (dôvod je hlavne sprehľadnenie zdrojového kódu).

Algoritmus zo svojej podstaty nedokáže spracovať postupnosť s duplicitnými hodnotami. Každý procesor bol preto obohatený o register *z_count*, ktorý počíta počet zápisov do registra Z a tým detekuje duplicitné hodnoty. Pri výpise je hlavnému procesoru zaslaný obzah registra Z ako aj obsah registra *z_count* pre zaistenie požadovaného počtu duplicitných hodnôt na výstupe.

Pre inicializáciu registov bola použitá symbolická konštanta `REG_EMPTY`, ktorá nadobúda hodnotu, ktorá sa vo vstupnej postupnosti nemôže vyskytnúť z dôvodu rozsahu 1 bajtu (0 – 255). Ďalej sú využívané symbolické konštanty `REG_TAG_X`, `REG_TAG_ZC` a `REG_TAG_Y` pre zasielanie správ pomocou zbernice a lineárneho spojenia. Pomocou týchto symbolických hodnôt sú sémanticky označované hodnoty podľa priradenia jednotlivým registrom vo volaniach `MPI_Recv()` a `MPI_Send()` (položka *TAG*).

3 Zložitosť algotitmu a experimentalne výsledky

Asymptotická zložitosť algoritmu:

$$\mathcal{O}(1) + \mathcal{O}(2 \cdot n) + \mathcal{O}(n) = \mathcal{O}(n)$$

kde n je počet radených hodnôt. Jednotlivé zložky vyjadrujú:

$\mathcal{O}(1)$ – inicializácia registra C
$\mathcal{O}(2 \cdot n)$ – distribúcia hodnôt a porovnania
$\mathcal{O}(n)$ – výpis hodnôt

Pre výpočet pomocou tohoto algoritmu je nutné použiť n procesorov, asymptotická časová zložitosť je lineárna $\mathcal{O}(n)$ a celková cena kvadraticky rastie v závislosti od počtu radených hodnôt $\mathcal{O}(n^2)$.

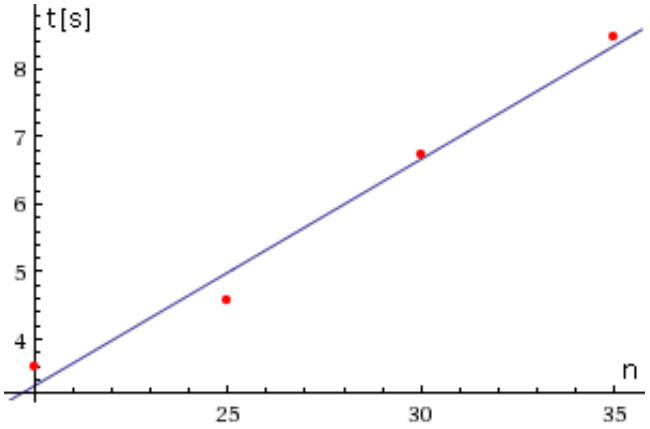
Pre vizualizáciu bolo uskutočnených niekoľko testov, v ktorých bolo predmetom zistiť dobu výpočtu programu. Pre testovanie bola využitý program `GNU time`. Réžia spojená so spustením programu (otvorenie súboru, inicializácia knižnice *Open MPI*, ...) neovplyvňuje veľkosť vstupu. Samotné načítavanie hodnôt zo súboru je súčasťou algoritmu. Vzhľadom na invariatnosť réžie voči veľkosti vstupu je preto možné program `GNU time` využiť pre meranie relatívnych časov výpočtu. Výsledky týchto meraní sú uvedené v tabuľke 1, interpolácia týchto hodnôt je predvedená na obrázku 2.

4 Komunikácia procesorov

Komunikácia procesorov simulujú funkcie `MPI_Recv()` a `MPI_Send()` pre zasielanie správ po lineárnom prepojení alebo po zbernici. Tieto funkcie sú súčasťou knižnice *Open MPI*. Celý proces komunikácie je znázornený

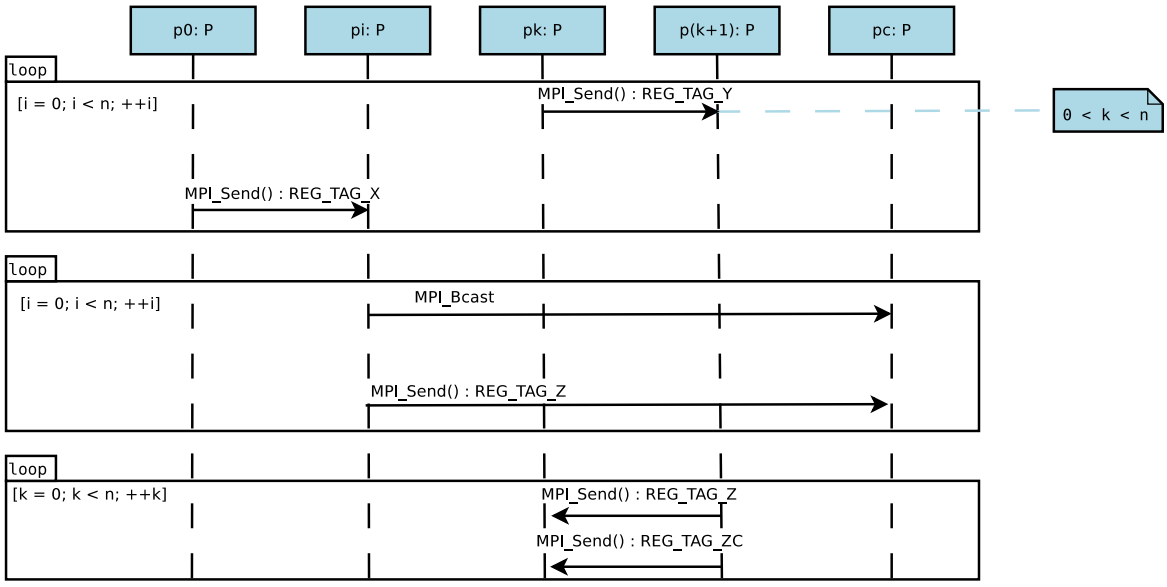
Počet prvkov	Priemerný čas výpočtu $[s]$
20	3,577
25	4,561
30	6,712
35	8,456

Tabuľka 1: Namerané hodnoty pri časových testoch.



Obr. 2: Lineárna interpolácia priemeru nameraných hodnôt pri časových testoch.

na obrázku 3.



Obr. 3: Sekvenčný diagram komunikácie procesorov.

5 Zhrnutie

Experimentálne výsledky uvedené v sekcii 3 potvrdzujú lineárnu triedu asymptotickej zložitosti. Komunikáciu medzi procesormi bolo nutné prispôsobiť vzhľadom na dostupné rutiny použitej knižnice *Open MPI*, ktoré sú bližšie priblížené v sekcii 2. Komunikáciu znázorňuje sekvenčný diagram v sekcii 4. Algoritmus bol úspešne implementovaný.