Ladislav Šulák [xsulak04@stud.fit.vutbr.cz]
Projekt do predmetu Paralelní a distribuované algoritmy
Carry Look Ahead Parallel Binary Adder
4/2016, FIT VUT v Brně

Rozbor a analýza algoritmu

Algoritmus vykonáva výpočty nad binárnym stromom, pričom uzly reprezentujú jednotlivé procesory. Koreňový uzol na začiatku distribuuje číslice zo vstupných binárnych čísel listovým uzlom, teda vždy 2. Počet listov, teda aj číslic musí byť mocninou čísla 2 a počet procesorov potom $p(n) = (2 * n) - 1$.

Algoritmus sa delí na viac fáz, postupne:

- výpočet hodnôt `initial D`, čo je vykonávané paralelne v každom listovom procesore, teda časová zložitosť je v tomto kroku konštantná. Priestorová zložitosť je určená počtom bitov vo vstupných číslach, čo je zároveň aj počet listových procesorov, teda tiež konštantná,
- suma prefixov poľa `initial D`, čo spočítame v logaritmickom čase, pretože prechádzame binárny strom algoritmom `scan`. Priestorová zložitosť je počet všetkých procesorov,
- pole `Carry` hodnôt získané z hodnôt v predchádzajúcom kroku. Je posunuté o 1 rád vyššie a jeho výpočet je vykonávaný v každom uzle paralelne. Časová zložitost tohto kroku je teda konštantná a priestorová je počet všetkých listových procesorov. Tento krok je vykonávaný tak, že sa hodnoty posunú doprava a následne doľava o listový procesor, avšak celková zložitosť ostáva rovnaká,
- výsledná hodnota v každom liste je získaná operáciou `xor` nad číslicami zo vstupných čísel, ktoré má daný procesor už od začiatku. V prípade, že je z poľa `Carry` v danom uzle prítomná hodnota `generate`, pripočítame 1 a vykonáme modulo 2. Časová zložitosť je vďaka tomuto prístupu opäť konštantná, ako aj priestorová.

Na záver každý listový procesor posiela svoju hodnotu koreňovému procesoru, ktorý vypisuje výsledok postupne od procesoru s najmenším id, ktorý obsahuje číslo `MSB` a pokračuje smerom k `LSB`, uloženým v procesore s najväčším id. Táto činnosť má konštantnú časovú žložitosť, čo neovplivní celkovú časovú a priestorovú zložitosť a ani cenu.

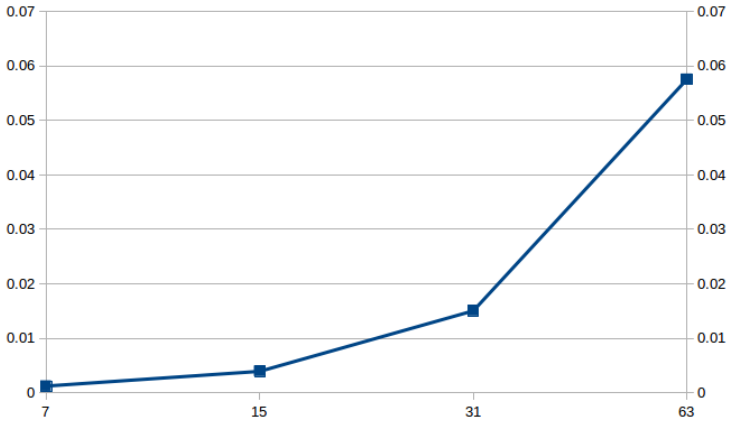
Čo sa týka celkovej teoretickej časovej zložitosti, tak je kvôli týmto krokom logaritmická vzhľadom k počtu bitov vstupných čísel. Priestorová je $2 * n - 1$, teda lineárna. Celková cena je daná súčinom týchto zložitostí, teda linearitmická, čo nie je optimálne.

Implementácia

Algoritmus bol implementovaný v jazyku C++ s využitím knižnice Open MPI v prostredí Ubuntu 15.10, kde bol aj testovaný. Vstupné čísla boli dodané v textovom súbore, pričom neboli generované automaticky, ale zadávané autorom tejto práce. Program výslednú zoradenú postupnosť vypíše na štandardný výstup.

Experimenty

Experimenty neboli realizované automatizovane, ale autorom práce postupne. Pri meraní časovej zložitosti bola využívaná funkcia `MPI_Wtime()`, ktorá merala čas vykonávania procesu. Počet procesorov bol volený 7, 15, 31 a 63 a každý experiment bol vykonávaný 8 krát, dokopy teda 40 meraní. Časová zložitosť je zachytená na grafe 1. Pri experimentoch bolo nutné pracovat s počtom číslic, ktoré sú mocninami 2, kvôli povahe algoritmu. Namerané hodnoty idú kvôli tomu skokovo a nie v užších intervaloch. To je jeden z dôvodov, prečo je graf v istej miere skreslený a nie je úplne linearitmický. Ďalším dôvodom je, že s narastajúcim počtom procesorov rástla réžia s tým spojená.



Obr. 1: Časová zložitosť algoritmu. Na X-ovej ose je znázornený počet procesorov a na Y-ovej ose aritmetický priemer časov behov každého procesu v danom experimente.

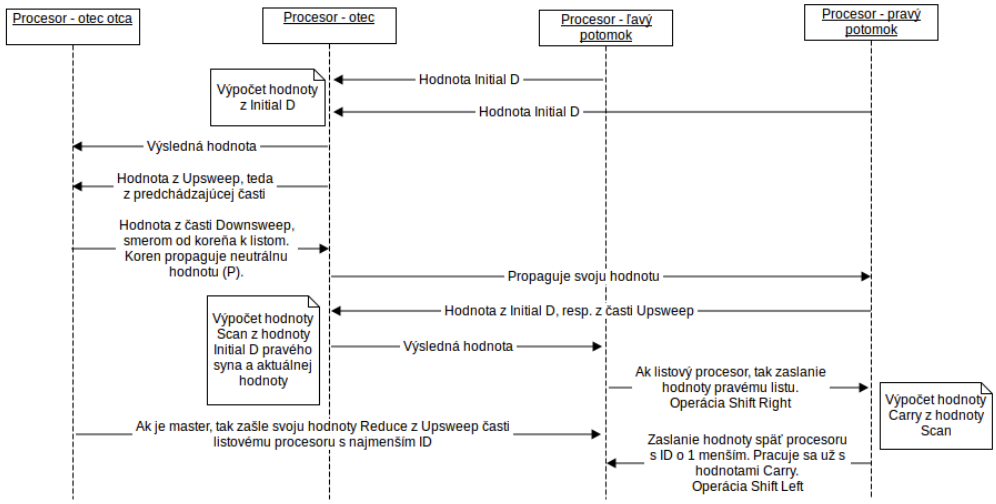
Komunikačný protokol

Komunikačný protokol, ktorý bol navrhnutý a pomocou ktorého je realizovaný zadaný algoritmus je znázornený na obrázku 2. Protokol sa skladá z 2 väčších celkov, z časti `Upsweep` a `Downsweep`. `Upsweep` funguje približne nasledovne:

- dva uzly, napríklad listové, zasielajú svoje hodnoty vyššie nadradenému procesoru v binárnom strome. Typicky sa jedná o výpočet hodnôt poľa `initial D`,
- ten na základe nich vypočíta novú hodnotu a zašle ju svojmu otcovi. Ak je to už koreňový procesor, tak sa časť `Upsweep` končí a koreň získal hodnotu `Reduce`.

`Downsweep` je realizovaný takto:

- vyššie nadradený procesor pošle svojmu pravému potomkovi hodnotu, ktoré prijíma od svojho otca. Ak je to koreň, tak distribuuje pravému synovi neutrálnu hodnotu, čo je v tomto algoritme hodnota `Propagate`,
- hodnota zasielaná ľavému synovi sa musí vypočítať. Získa sa operáciou `scan` nad hodnotami z pravého syna (ktorú mal uloženú z časti `Upsweep`, musí mu ju opätovne poslať) a aktuálnej hodnoty obdržanej od otca. Operácia nie je komutatívna, takže je nutné zachovat toto poradie, vzhľadom na to, že listový procesor s najmenším ID obsahuje MSB z daného vstupného čísla a najpravejší obsahuje LSB.



Obr. 2: Komunikačný protokol implementovaný v tomto projekte pre paralelný výpočet znázorňuje komunikáciu medzi procesorom, jeho otcom a 2 potomkami v binárnom strome.

Posledné kroky algoritmu:

- operácia `Shift Right`, teda každý listový procesor zasiela svoju hodnotu procesoru s číslom ID o 1 väčším, teda tomu napravo. Listový procesor najviac vľavo prijíma hodnotu od koreňa. Táto hodnota reprezentuje hodnotu `Reduce`, ktorá slúži na stanovenie toho, či došlo k pretečeniu,
- potom sa všetky hodnoty pošlú opäť doľava, čím sa myslí operácia `Shift Left`,
- výsledné `Carry` hodnoty sa sčítajú s hodnotami vstupných číslic, kde sa použije operácia `xor` a výsledok pošle každý listový procesor koreňu. Ten ich vypíše podľa poradia čísel procesorov, teda aj podľa korektného poradia čisiic po výslednom sčítaní, prípadne nasledované informáciou, či došlo k pretečeniu.

Záver

Táto dokumentácia popisuje algoritmus Carry Look Ahead Parallel Binary Adder, ktorý bol úspešne implementovaný v paralelnom prostredí knižnice Open MPI s využitím jazyka C++. Po implementácií bola vykonávaná rada experimentov, ktoré mali porovnať teoretickú zložitosť so skutočnou. Pri väčšom počte zadaných procesorov algoritmus výsledky v istej miere skresľuje, kvôli tomu, že bol projekt testovaný na nízkom počte fyzických jadier, konkrétne na dvoch[1]. S menšími hodnotami nameraná zložitosť odpovedá očakávanej.

[1] http://ark.intel.com/products/67355/Intel-Core-i5-3210M-Processor-3M-Cache-up-to-3_10-GHz-rPGA