1. Úvod

Cílem projektu je pomocí knihovny Open MPI implementovat v jazyce C/C++ algoritmus Carry Look Ahead Binary Parallel Adder.

1.1 Vstup programu

Vstupem programu je dvojí posloupnost bitů oddělená znakem nového řádku. Délka těchto bitů je vždy mocnina 2. Program tuto posloupnost bitů načte a pomocí algoritmu Carry Look Ahead Binary Parallel Adder tyto dvě hodnoty sečte a vypíše na standardní výstup. V případě přetečení toto taktéž program vypíše na standardní výstup.

1.2 Výstup programu

Výstup programu je buď na standardní výstup nebo na standardní chybový výstup. V případě korektního chování programu je na standardní výstup vypsán identifikátor procesoru a hodnota jemu vypočteného bitu. Totéž je zopakováno pro všechny listové procesory. Jednotlivé procesory jsou odděleny novým řádkem. V případě přetečení je toto taktéž vypsáno na standardní chybové výstup.

2. Analýza algoritmu

2.1 Popis algoritmu

Samotný algoritmus pracuje nad binárním stromem. Algoritmus funguje jako binární sčítačka, která dvě binární čísla je schopná sečíst v log(n) krocích. Toho je docíleno tím, že před samotným výpočtem, se vypočítají všechny carry bity.

2.2 Analýza algoritmu

Algoritmus probíhá v několika fázích. Nejprve je s konstantím časem spočtena hodnota d, kterou využijeme pči výpočtu bitů přenosu. Díky tomu je pak za použití algoritmů UpSweep a DownSweep vypočtena suma prefixů. Následně samotný výpočet již proběhne v čase n. Pro vykonání algoritmu je nicméně potřeba 2 * n – 1 procesorů.

Tedy výsledné složitosti jsou následujicí:

$$t(n) = O(\log n)$$
$$p(n) = n$$

Pokud známe tyto dvě složitosti jsme dále schopni si odvodit celkovou cenu algoritmu.

$$c(n) = t(n)*p(n) = O(n * \log n)$$

Z čehož vyplývá, že algoritmus není optimální.

3. Implementace

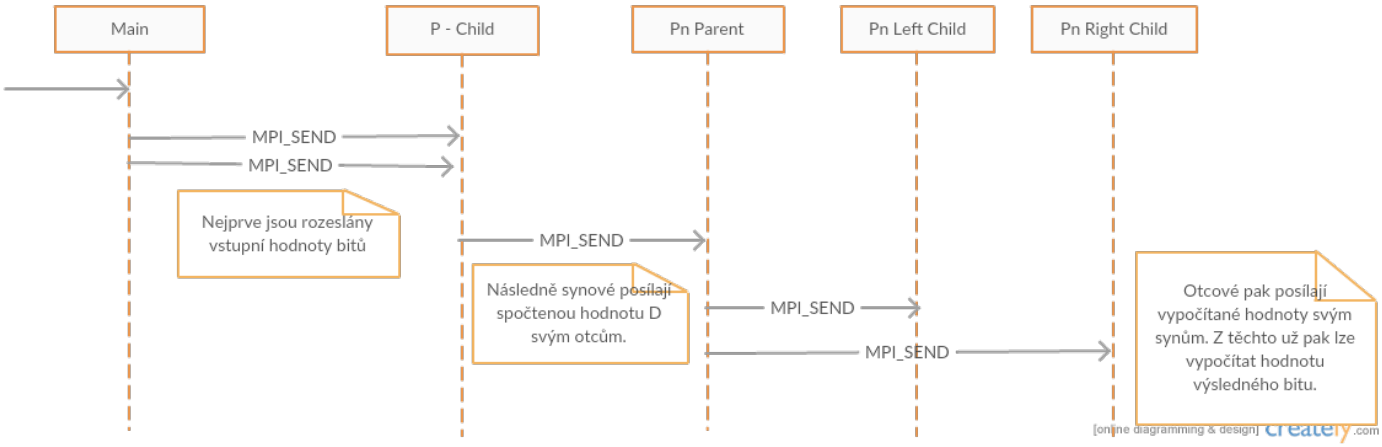
K implementaci algoritmu bylo použito jazyka C++ a knihovny Open MPI. Celá implementace algoritmu je rozdělena do dvou větších celků.

3.1 Příprava vstupních dat

V první části algoritmu bylo nejprve třeba připravit vstupní data pro následné sčítání. Obě čísla jsou tedy načtena ze vstupního souboru a pomocí knihovny MPI jsou jednotlivé bity rozeslány procesorům.

3.2 Implementace algoritmu

O průběh algoritmu se stará funkce parallelBinaryAdder, která nejprve rozešle listům hodnoty jednotlivých bitů. Následně synové spočtou a vrátí hodnotu d, podle které se určí, jestli bude docházet k přenosu. Náasledně otcové svým synům zašlou již vypočtené hodnoty synům a následně lze pak již spočítat výsledné sečtené bity. Více lze vyčíst z obrázku 3.2.1.



Obrázek 3.2.1 Sekvenční diagram

4. Experimenty a testování

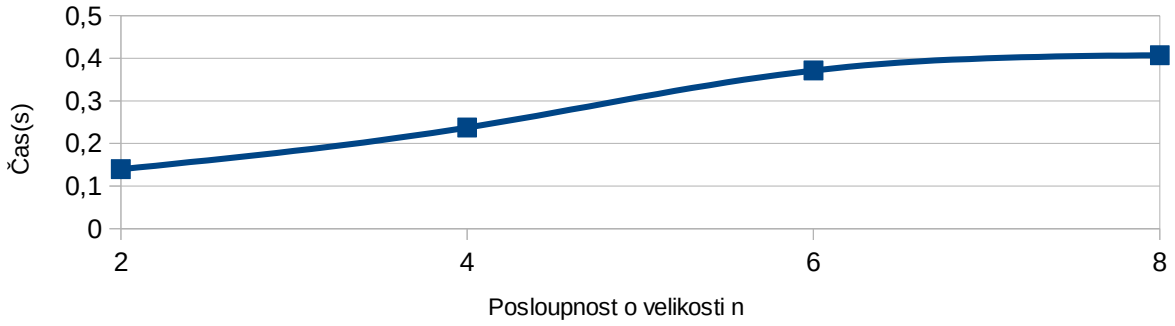
Veškeré experimenty byly prováděny na školním serveru *Merlin* s unixovým operačním systémem. Testy byly prováděny pro vstupní posloupnost o velikosti 2, 4, 6 hodnot. Více hodnot bohužel server *Merlin* neumožňuje z důvodu limitu na počet přidělených procesorů. Veškeré testy byly opakovány celkem desetkrát se stejnými parametry, přičemž pro zlepšení výsledků testování byly ve finále vyloučeny nejrychlejší a nejpomalejší výsledky řazení. Čas byl měřen pomocí unixové utility *time*.

	Posloupnost $n = 2$	Posloupnost $n = 4$	Posloupnost $n = 6$	Posloupnost n = 8
Pokus č. 1	0.132s	0.243s	0.453s	0.477s
Pokus č. 2	0.138s	0.217s	0.347s	0.444s
Pokus č. 3	0.150s	0.227s	0.351s	0.355s
Pokus č. 4	0.150s	0.220s	0.324s	0.350s
Pokus č. 5	0.148s	0.257s	0.390s	0.412s
Pokus č. 6	0.121s	0.284s	0.376s	0.386s
Pokus č. 7	0.166s	0.223s	0.414s	0.410s
Pokus č. 8	0.111s	0.227s	0.315s	0.423s
Průměr	0.1395s	0.23725s	0.37125s	0.407125s

Tabulka 4.1 Naměřené hodnoty implementovaného algoritmu

Naměřené hodnoty jsou zanesené do grafu na obrázku 4.1, ze kterého lze vidět téměř logaritmický průběh měřeného algoritmu. Nicméně data nezobrazují úplně přesný logaritmický průběh, tuto anomálii by šlo pravděpodobně eliminovat testováním algoritmu na více procesorech a s delší vstupní posloupností, případně tato odchylka může být způsobena využitím utility *time*, která nezměří pouze průběh algoritmu, ale taktéž načítání vstupní posloupnosti dat a práci se standardním výstupem. V poslední řadě může být mírná odchylka

způsobená chybou/neoptimalizací algoritmu.



Obrázek 4.1 Graf znázorňující vztah velikosti řazené posloupnosti a doby řazení.

5. Závěr

V projektu jsem implementoval algoritmus Carry Look Ahead Parallel Binary Adder a následně provedl ověření časové složitosti. Z výsledků získaných testováním a experimenty lze odvodit, že výsledná časová složitost implementovaného algoritmu odpovídá teoreticky spočtené.