

## Rozbor a analýza algoritmu

### Topologie

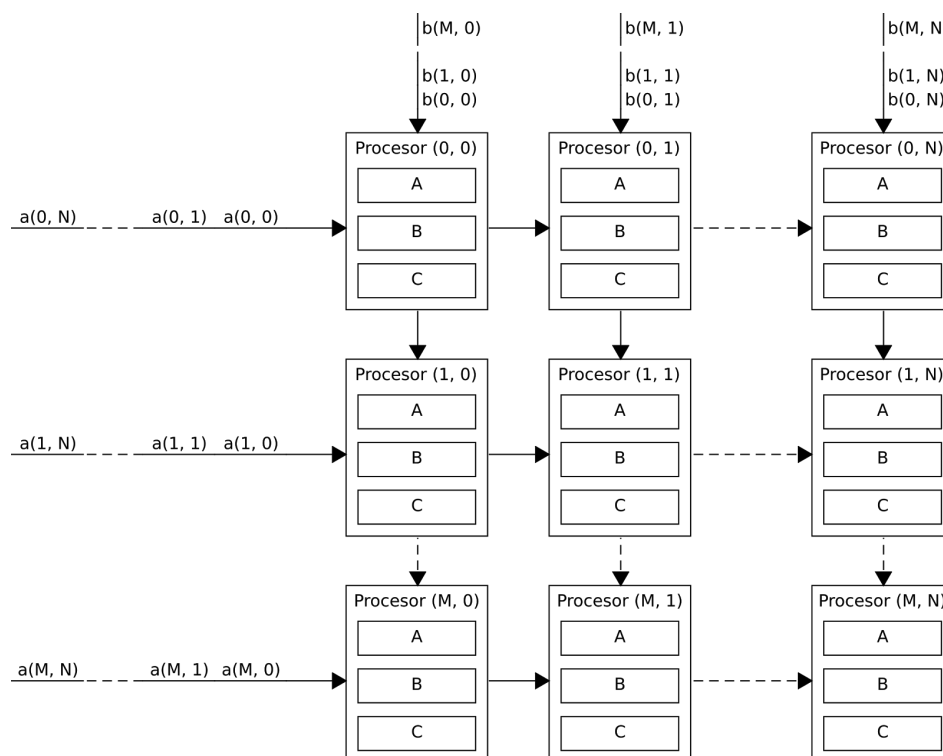
Mesh Multiplication je paralelní algoritmus pro násobení matic, kdy pro každý prvek výsledné matice existuje právě jeden procesor. Ten má za úkol počítat výsledek násobení pouze v daném prvku matice. Jako vstupní hodnoty mu slouží data z příslušného sloupce a řádku násobených matic. Tyto hodnoty jsou lineárně přesouvány mezi procesory a to takovým způsobem, že každý prvek každého řádku (sloupce) je postupně propagován do každého procesoru reprezentující prvek na daném řádku (sloupci) výsledné matice. Postup přesouvání dat ve vodorovném směru je synchronizován s krokem ve směru svislém, takže v každém kroku výpočtu procesoru jsou na jeho vstupu obě hodnoty nově získané. Každá z procesorů tedy disponuje registry:

- **Registr A**, pro uchování hodnoty ze vstupní matice A. Tato hodnota je posouvána vodorovně
- **Registr B**, pro hodnotu prvku z matice B. Tato hodnota je přesouvána svisle
- **Registr C**, akumulátor, který uchovává dočasný (i konečný) výsledek násobení v daném prvku matice

Speciální úlohu zastávají procesory prvního řádku a prvního sloupce. Ty vstupní hodnoty nezískávají od svých předchůdců v řadě či sloupci, ale naopak je čtou přímo ze souboru. Ty navíc disponují vstupními frontami:

- **Fronta A**, pro hodnoty odpovídajícího řádku vstupní matice A nebo
- **Fronta B**, pro hodnoty prvků ve sloupci matice B

V konečném důsledku pak topologie vypadá následovně:



### Princip algoritmu

Mnou zvolená algoritmická reprezentace samotného násobení matic je prakticky shodná s uvedeným postupem v přednášce<sup>1</sup>:

1. Pro každý procesor nastav hodnotu **Registru C** jako  $C = 0$

<sup>1</sup> <https://www.fit.vutbr.cz/study/courses/PDA/private/www/h005.pdf> , slide 18

2. Opakuj  $N + M$  krát, pro  $0 \leq i < N + M$ , kde  $N$  je počet sloupců výsledné matice a  $M$  je počet řádků:
  - a. Pokud je procesor v prvním sloupci, získá hodnotu **Registru A** z vlastní **Fronty A**, v opačném případě ji získá od procesoru vlevo od sebe
  - b. Pokud je procesor v první řadě, získá hodnotu **Registru B** z vlastní **Fronty B**, v opačném případě ji získá od procesoru nad sebou
  - c. Spočítá a zapamatuje si hodnotu **Registru C** jako  $C = C + (A * B)$
  - d. Pokud procesor neleží v posledním sloupci, pošle hodnotu svého **Registru A** svému sousedovi vpravo
  - e. Pokud procesor neleží v posledním řádku, pošle hodnotu svého **Registru B** svému sousedovi dolů

## Implementace

Pro implementaci algoritmu bylo především třeba nejdříve vyřešit správné načtení a přípravu dat. Toho jsem docítil následovně:

1. Nejprve si první procesor zjistí velikost výsledné matice, provede kontrolu platnosti dat v souborech a následně údaje o počtu řádků a sloupců pošle ostatním procesorům pomocí `MPI_Bcast`
2. Každý procesor zjistí, zda-li leží v prvním řádku či prvním sloupci
3. Pokud procesor náleží k prvku v prvním řádku v souboru, načte si patřičný řádek a uloží si jeho hodnoty do své vstupní fronty
4. Pokud je procesor v prvním sloupci, pak prochází vstupní soubor druhé matice a ukládá si do své fronty pouze ty prvky, které leží na patřičném místě v řádku matice

Takto načtená data se pak jen vyvolávají z fronty svých procesorů a nijak samotný algoritmus nezpomalují čtením dat ze souboru.

Za zmínku dále stojí implementační detail zobrazování výstupu. Pro tuto činnost bylo z důvodů synchronizace a správného pořadí dat nejlepší si zvolit jeden procesor jako řídící. Tento procesor pak přijímá hodnotu výsledného registru od všech ostatních procesorů a postupně je vypisuje na standardní výstup. Pro přehlednost a jednoduchost jsem zvolil první procesor. Ten totiž může bez obav nejdříve vypsat svá data a následně přijímat od všech ostatních ve zvoleném pořadí, aniž by hrozila ztráta jeho vlastních dat.

## Složitost algoritmu

Asymptotická složitost<sup>2</sup> algoritmu

$$O(1) + O(n) = O(n)$$

Časová složitost

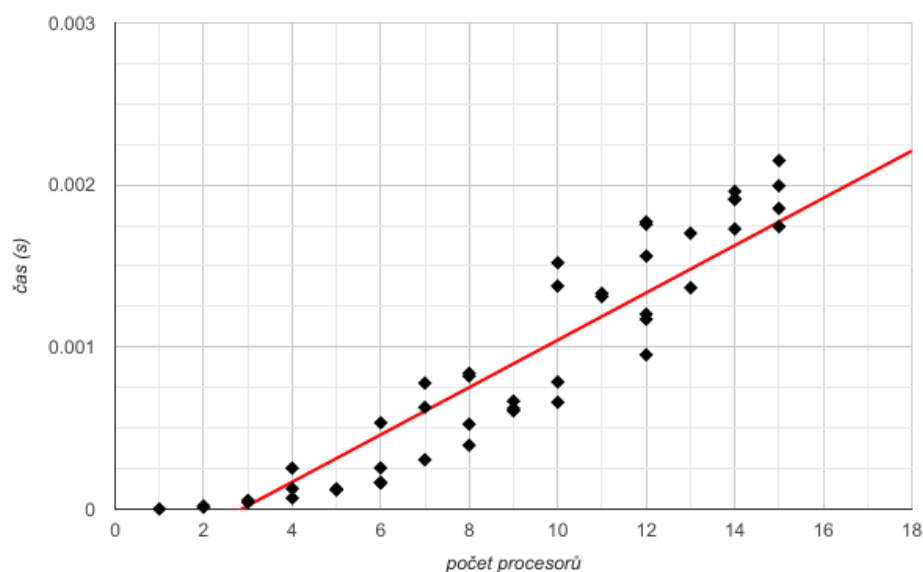
$$O(n)$$

Prostorová složitost

$$O(n^2)$$

Celková cena

$$O(n) * O(n^2) = O(n^3)$$

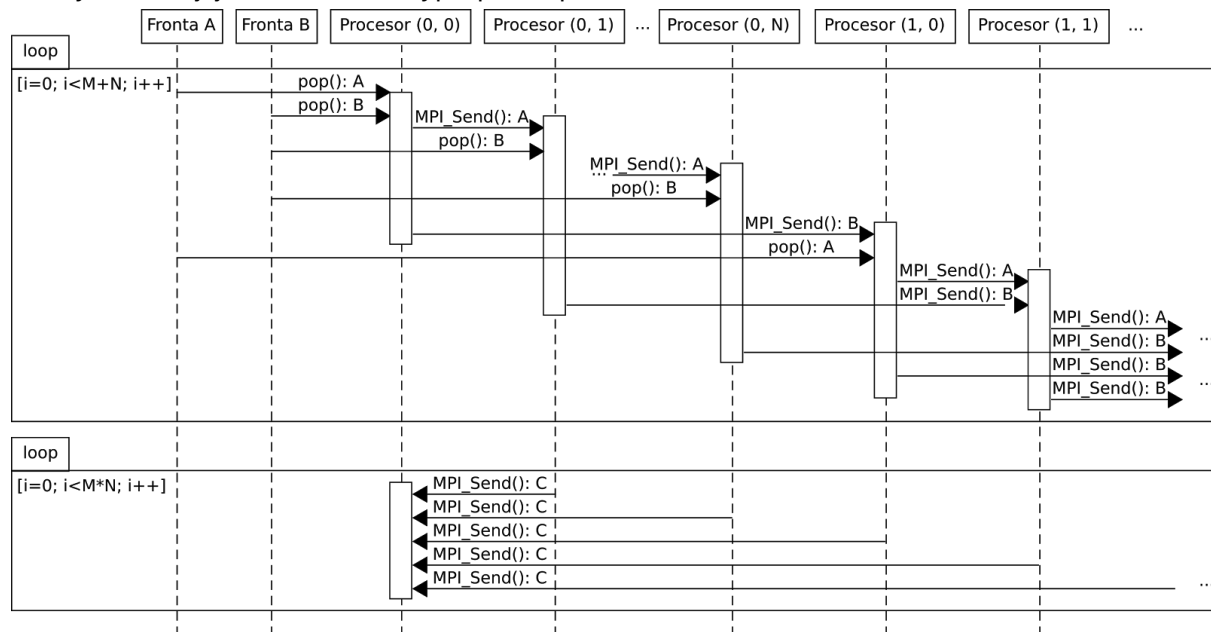


<sup>2</sup> Každá složka součtu odpovídá jednomu kroku podle uvedeného algoritmu

Tyto teoretické hodnoty byly podrobeny praktickému testu, kdy pro různé velikosti vstupních matic, a tudíž i různé počty použitých procesorů, bylo sledováno trvání výpočtu. Pro každou velikost bylo vygenerováno 10 náhodných dvojic vstupních matic s hodnotami v rozsahu \$RANDOM (Bash) a druhým rozměrem o náhodné velikosti od 2 do 10. Takto vytvořené matice byly vynásobeny, přičemž byl měřen čas. Pro každý počet procesorů bylo měřeno trvání výpočtu pro všechny topologie, tedy například pro 12 procesorů byly vyzkoušeny všechny tyto možnosti:  $1 * 12$ ,  $12 * 1$ ,  $2 * 6$ ,  $6 * 2$ ,  $3 * 4$ ,  $4 * 3$ . Měření probíhalo bez zohlednění trvání načítání vstupů do paměti a také shromažďování výsledků – měřeno bylo tedy pouze trvání samotného algoritmu Mesh Multiplication. Měřeny byly časy pro počet procesorů 1 až 15, celkově tedy bylo naměřeno 46 sad měření průměrných časů z 10 běhů programu.

## Komunikační protokol

Pro komunikaci mezi procesory disponuje knihovna OpenMPI funkcemi `MPI_Send` a `MPI_Recv`. Následující diagram zpráv znázorňuje veškeré operace při kterých spolu procesory musí kooperovat a koordinovat svou činnost. V prvním cyklu tedy probíhá Mesh Multiplication, druhý cyklus symbolizuje sběr výsledků a jejich akumulaci a výpis prvním procesorem.



## Závěr

Experimentální výsledky měření časové složitosti potvrzují předpokládané. Avšak je zde zřejmá vysoká fluktuace hodnot. A to i přes to, že byla snaha eliminovat jakékoliv další vnější vlivy. Možná příčina je v nedostatečné náhodě velikosti vstupních matic (druhého rozměru, než toho, který definuje počet procesorů). To v souvislosti, kdy v sadě proběhlo pouze 10 měření může ovlivnit konečnou průměrnou hodnotu.

Implementace má také své úskalí, které mohlo být vyřešeno lépe. Tím je načítání vstupních dat, kdy každý proces s frontou přistupuje ke vstupním souborům. Ideální přístup by byl, kdy pouze jeden procesor otevře soubor, postupně načte hodnoty (pro **Frontu A** či **Frontu B**). Mé řešení produkuje zbytečnou zátěž pro I/O operace a naráží na systémová limity ve chvíli, kdy je procesorů v prvním sloupci a řadě tolik jako je limit pro počet filedescriptorů.