Dokumentace k projektu č. 2 předmětu PRL

Mesh Multiplication

Jakub Stejskal

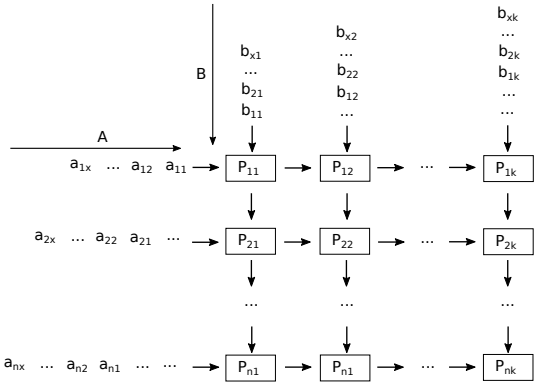
25. dubna 2017

Rozbor algoritmu

Algoritmus *Mesh Multiplication* násobí dvě vstupní matice na mřížce procesorů o velikosti $n \times k$ kde:

- n je počet řádků první matice
- k je počet sloupců druhé matice

Aby mohli být matice vynásobeny musí pro matice platit omezení velikosti takové, že $n \times x$ pro velikost první matice a $x \times k$ pro velikost druhé matice. Jinými slovy počet sloupců první matice je shodný s počtem řádků druhé matice.



Obrázek 1: Nákres sítě pro násobení matic.

Algoritmus

Prvky matic A a B jsou přiváděny do jednotlivých procesorů 1. řádku a 1. sloupce stejně, jak je vidět na obrázku . Jakmile procesor obdrží obě potřebné hodnoty provede jejich vynásobení a hodnotu přičte do svého registru C. Každý procesor $P_{i,j}$ obsahuje registr C, ve kterém uchovává výslednou hodnotu matice z pozice [i,j].

Po provedení výpočtu jsou původní vstupní hodnoty odeslány sousedním procesorům. Hodnoty matice A vlevo, hodnoty matice B dolů.

Analýza

Prvky a_{n1} a b_1k potřebují celkem $n+k+x-2$ kroků, aby se dostaly do posledního procesoru $P_{n,k}$.

- časová složitost: $t(n) = O(n^2)$
- prostorová složitost: $p(n) = O(n)$
- cena: $c(n) = t(n) \cdot p(n) = O(n^2) \cdot O(n) = O(n^3)$

Sekvenční algoritmus má složitost $O(n^3)$. Složitost optimálního algoritmu není známa, ale platí následující: $O(n^x), 2 < x < 3$. Paralelní verze násobení matic tedy není optimální.

Implementace

Implementace byla provedena v jazyce C++ s využitím knihovny *Message Passing Interface (MPI)* pro paralelní komunikaci jednotlivých procesorů.

Na rozdíl od algoritmu uváděném ve slajdech byla výsledná implementace obohacena o jeden řídící procesor. Celkový počet procesorů je tedy $p = (n \times k) + 1$. Přidaný procesor načítá vstupní soubory a posílá jednotlivé hodnoty procesorům v 1. řádku a 1. sloupci. Tato modifikace byla povolena na fóru předmětu PRL.

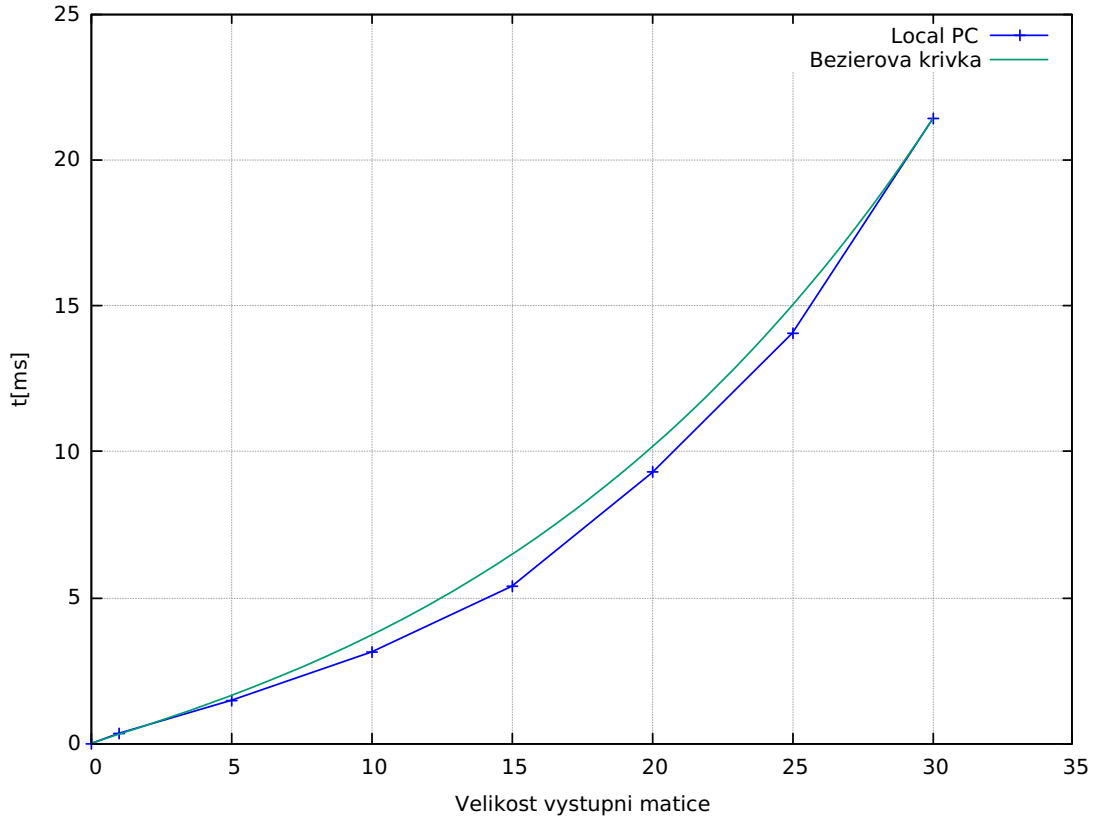
Pro rozeznání jednotlivých odeslaných hodnot byla vytvořena označení (TAG). Každý procesor pak podle tagu rozpozná, zda přijal hodnotu matice A nebo hodnotu matice B a to hlavně kvůli dalšímu přeposílání získaných prvků. Vlastní tag má také rozeslaná hodnota s velikostí matic.

Řídící procesor odešle vždy $x-krt$ správnou hodnotu z matice B procesorům v prvním řádku a z matice A procesorům v prvním sloupci pomocí `MPI_Send()`. Každý procesor tedy obdrží hodnoty z matice A a B celkem $x-krt$ pomocí `MPI_Recv()`. Odesílání hodnot ostatních procesorů je ovlivněno jejich *id*, které slouží k určení, zda procesor není hraniční a nemá již nikomu nic posílat jako je vidět na obrázku .

Při obdržení obou hodnot každý procesor přičte jejich vynásobenou hodnotu do svého registru C (původne inicializován na nulu) a až poté odešle hodnoty dál. Výslednou hodnotu C odešle po obdržení x prvků z A a B řídícímu procesoru.

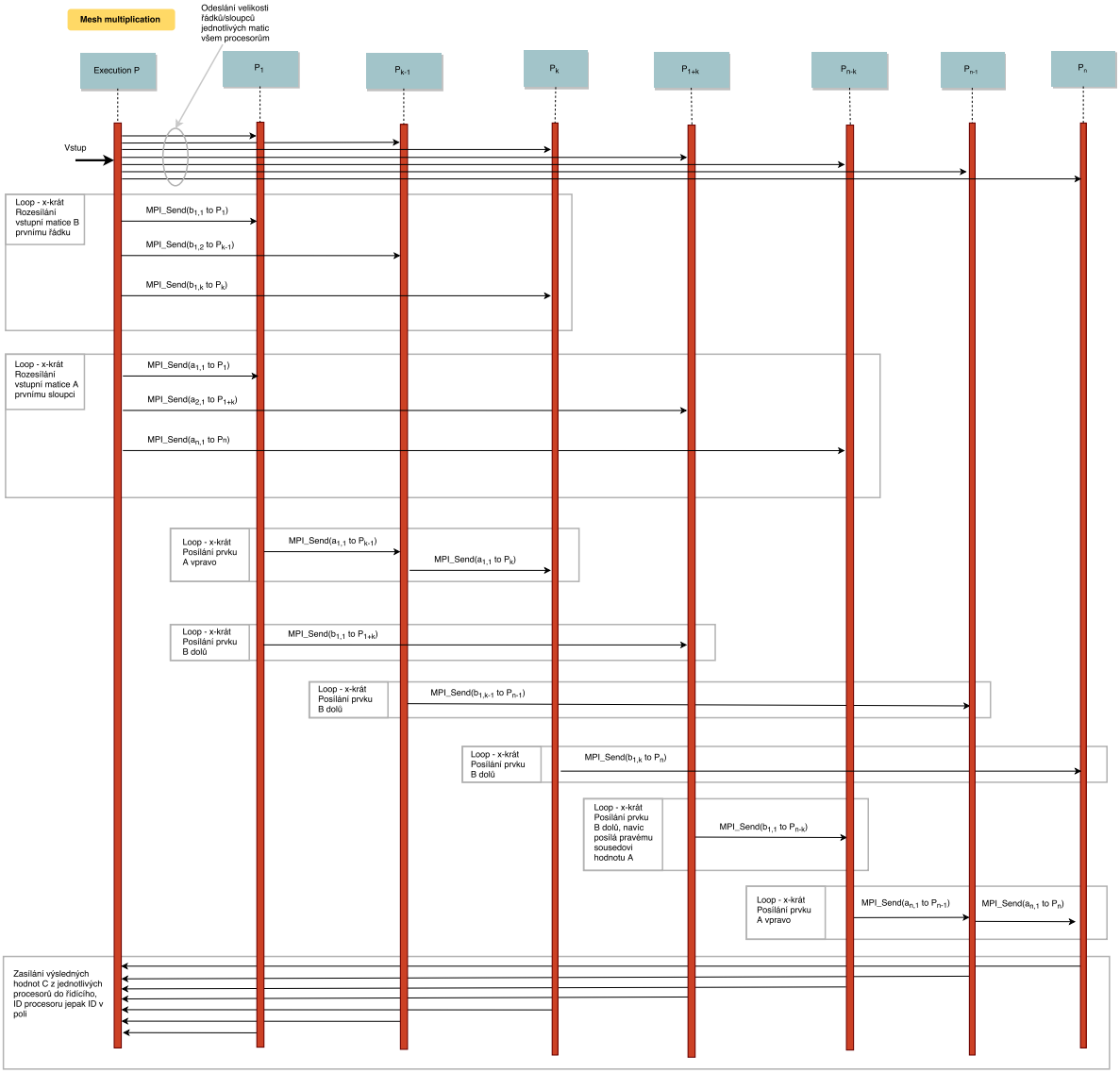
Experimenty s vytvořenou realizací

Pro měření času řazení vstupních posloupností byla využita funkce `MPI_Wtime()` z knihovny *MPI*. Měření bylo provedené na vlastním počítači při běžné zátěži. Každé měření bylo provedeno celkem 300-krát a naměřené hodnoty byly následně zprůměrovány. Z obrázku 2 je vidět,že časová náročnost je kvadratická.



Obrázek 2: Graf provedeného měření.

Komunikační protokol



Obrázek 3: Sekvenční diagram pro komunikaci procesorů.

Závěr

Implementace dosahuje stejných vlastností, jako algoritmus probíraný během přednášek. Časovou složitost závažně neovlivnil ani přidaný řídící procesor pro rozesílání prvků jak je vidět na obrázku 2. Drobné odchylky od přesné kvadratické složitosti jsou způsobeny výkyvy výkonu testovacího stroje.

- časová složitost: $t(n) = O(n^2)$
- prostorová složitost: $p(n) = O(n)$
- cena: $c(n) = t(n) \cdot p(n) = O(n^2) \cdot O(n) = O(n^3)$

Implementace tedy není optimální, protože cena nedosahuje potřebné hodnoty.