Mesh Multiplication

Paralelní a distribuované algoritmy

Autor: Jakub Sadílek
Login: xsadil07
Datum: 3.5.2021

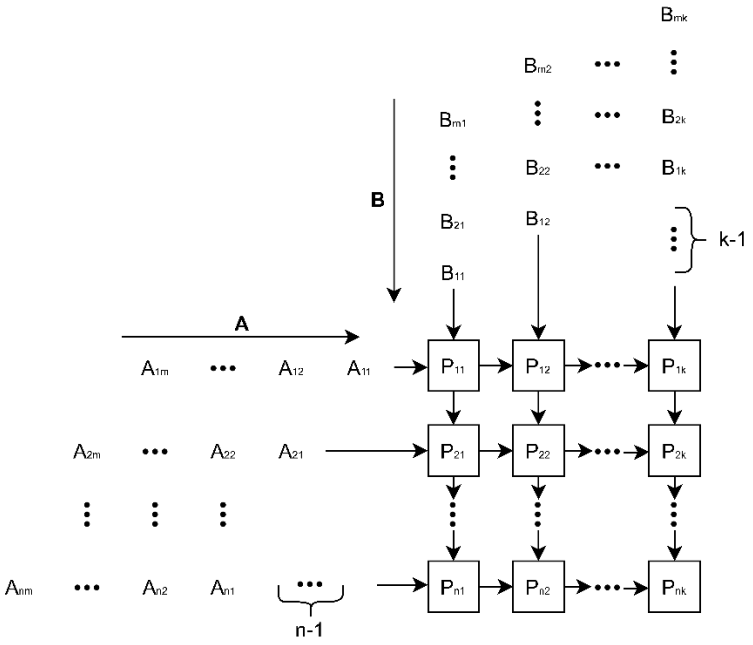
1. Úvod

Cílem tohoto projektu je implementovat paralelní algoritmus Mesh Multiplication pro výpočet výsledku po násobení dvou matic. Projekt je napsán v jazyce C++ za pomoci knihovny Open MPI. Níže v této dokumentaci je podrobněji popsán princip a analýza algoritmu, implementace včetně komunikace mezi procesory, testování a na závěr je uvedena ukázka spuštění programu.

2. Popis algoritmu

Mesh Multiplication je paralelní algoritmus pro násobení dvou matic, který využívá ke svému běhu mřížku procesorů, kde každý procesor je zodpovědný právě za jednu hodnotu buňky ve finální matici. Aby bylo násobení těchto matic vůbec možné, je nutné, aby šířka první matice byla stejně velká jako výška druhé matice. Potom $\boldsymbol{n}$ je výška první matice a $\boldsymbol{k}$ je šířka druhé matice. Výsledná matice bude mít tvar $\boldsymbol{n}$: $\boldsymbol{k}$ a k výpočtu bude potřeba přesně $\boldsymbol{n} * \boldsymbol{k}$ procesorů. Zároveň je důležité zmínit, že násobení matic není komutativní operace.

Výpočet probíhá tak, že první matice se nasouvá do mřížky procesorů postupně zleva a jednotlivé řádky matice jsou zpožděny o 1 krok. Druhá matice se nasouvá do mřížky shora a jednotlivé sloupce jsou opět posunuty o 1 krok výpočtu. Na začátku je hodnota všech procesorů vynulována a jakmile procesor obdrží obě hodnoty (hodnotu zleva a hodnotu shora) provede vynásobení těchto hodnot a výsledek přičte ke své aktuální hodnotě. Následně pošle hodnotu zleva svému sousedovi napravo a hodnotu shora svému spodnímu sousedovi (viz obrázek 1). Poté co jsou zpracovány všechny hodnoty všemi procesory, bude každý procesor obsahovat hodnotu buňky výsledné matice na dané souřadnici.



Obrázek 1: Mřížková architektura

3. Teoretická složitost algoritmu

Pro výpočet výsledné matice se o každou buňku stará jeden procesor. Je-li výška první matice $\boldsymbol{n}$ a šířka druhé matice $\boldsymbol{k}$, potom výsledná matice bude mít rozměr $\boldsymbol{n}$: $\boldsymbol{k}$ (viz teorie násobení matic) a tedy bude zapotřebí $\boldsymbol{n} * \boldsymbol{k}$ procesorů. Jelikož všechny procesory mají stejnou časovou složitost s rozdílem, že každý začíná nebo končí vlastní výpočet v jiný okamžik (jinak čeká), uveďme pro jednoduchost výpočet složitosti na posledním procesoru v matici ($\boldsymbol{P_{nk}}$).

První prvek z poslední řady z první matice se k výpočtu dostane za $\boldsymbol{n - 1}$ kroků (viz obrázek 1), poslední prvek z této řady se dostane na řadu $\boldsymbol{m - 1}$ kroků po prvním prvku. Zároveň musí tento prvek projít celou šířkou maticí procesorů, než so dostane k poslednímu procesoru, tedy dalších $\boldsymbol{k}$ kroků. Celkový počet kroků pro prvek $\boldsymbol{A_{nm}}$ (viz obrázek 1) bude $\boldsymbol{n + m + k - 2}$.

Stejný princip platí i pro poslední prvek z druhé matice ($\boldsymbol{B_{mk}}$). Ten se do výpočtu dostane $\boldsymbol{m - 1}$ kroků po prvním prvku z posledního sloupce, který dojde na řadu za $\boldsymbol{k - 1}$ kroků od začátku algoritmu. Opět, než prvek projde výškou maticí procesorů k poslednímu procesoru, zabere mu to $\boldsymbol{n}$ kroků. Tedy celkový počet kroků posledního prvku z druhé matice, než dojde k poslednímu procesoru je $\boldsymbol{k + m + n - 2}$, tedy stejný počet kroků jako pro poslední prvek z první matice.

Pro jednoduchost můžeme uvážit násobení čtvercových matic. Potom poslední prvky budou spočteny posledním procesorem za $\boldsymbol{n + n + n - 2}$ kroků, po zjednodušení dostaneme lineární časovou složitost $\boldsymbol{O(n)}$.

Je-li časová složitost $\boldsymbol{t(n) = O(n)}$ a je-li zapotřebí $\boldsymbol{n * k}$ procesorů, potom $\boldsymbol{p(n) = O(n^2)}$ dostaneme z rovnice $\boldsymbol{c(n) = t(n) * p(n)}$ cenu algoritmu $\boldsymbol{c(n) = O(n) * O(n^2) = O(n^3)}$.

4. Implementace

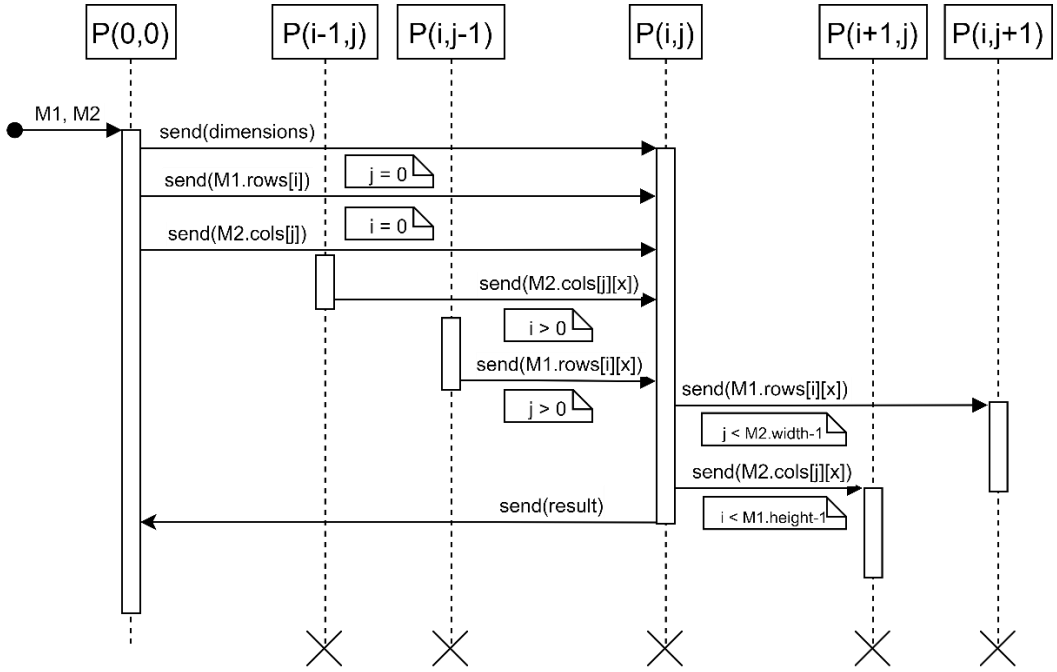
Program je podle zadání implementován v jazyce C++ za použití knihovny Open MPI. Na začátku programu je inicializováno MPI prostředí a první procesor ve funkci `firstProcPreparation()` kromě ošetření chybných vstupů, načte oba soubory s maticemi, které uloží do tříd `Matrix`. Následně rozešle pomocí funkce `MPI_Send()` informace o rozměrech matic všem procesorům, kteří je přijmou pomocí funkce `MPI_Recv()` a uloží si je do třídy `Proc`, která obsahuje podstatné informace k výpočtu pro každý procesor. Kromě rozměrů matic třída `Proc` obsahuje i ID procesoru, jeho souřadnice v matici a hodnotu buňky, pro kterou počítá hodnotu. Poté, co každý procesor zná rozměry matic, může si spočítat svoji pozici v matici pomocí funkce `setCoordinates()`. Na konci této přípravy už jen první procesor odešle řádky první matice procesorům v prvním sloupci a sloupce druhé matice procesorům v první řadě. Průběh přípravy pro ostatní procesory je proveden obdobně ve funkci `procPreparation()`.

Následuje výpočet výsledku ve funkci `meshMult()`, kde každý procesor provede m iterací (viz kapitola 3). Na začátku každé iterace procesor přijme obě hodnoty (zleva a ze shora) potřebné k výpočtu buď z počáteční fronty, jedná-li se o počáteční procesor, kde hodnoty obdržel na začátku od prvního procesoru nebo přijme hodnotu od svého souseda nalevo či nahoře pomocí funkce `MPI_Recv()`. Pak provede výpočet v daném kroku podle vztahu $\boldsymbol{hodnota = hodnota + h * v}$, kde $\boldsymbol{h}$ je prvek přijatý zleva a $\boldsymbol{v}$ je prvek přijatý se shora. Po spočtení nové hodnoty, pošle prvek přijatý zleva procesoru vpravo není-li procesor v posledním sloupci a prvek přijatý shora pošle spodnímu procesoru není-li procesor v poslední řadě.

Po skončení výpočtu pošlou všechny procesory ve funkci `sendResults()` své hodnoty prvnímu procesoru, který je ve funkci `receiveResults()` přijme a uloží do třídy `Matrix`. Pak první procesor vypíše výslednou matici ve funkci `printMatrix()`, která ji očekává jako parametr a program končí.

5. Komunikační protokol

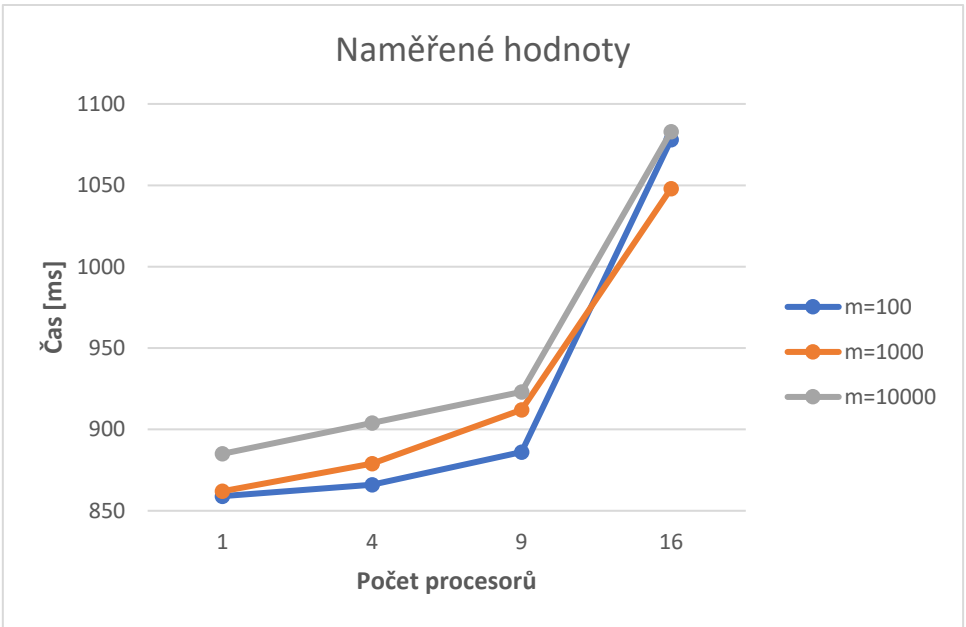
Níže na obrázku 2 je zobrazena sekvenční komunikace mezi procesory v prostředí MPI. Na začátku je vyobrazeno, jak 1. procesor distribuuje rozměry matic a následně pošle počátečním procesorům odpovídající data. Poté si procesory mezi sebou předávají hodnoty jak je definováno v algoritmu a nakonec jejich výsledky jsou poslány zpět k 1. procesoru.



Obrázek 2: Sekvenční diagram komunikace procesorů

6. Testování

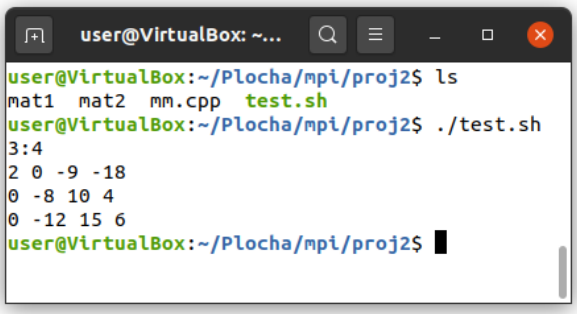
V následujícím grafu jsou uvedeny výsledné časy při měření rychlosti programu utilitou `time` na školním serveru Merlin při násobení matic různých velikostí, kde m značí šířku první matice a výšku druhé matice. Výsledné matice měli rozměry $n : n$.



Graf 1: Výsledky naměřených časů programu při výpočtu matic různých velikostí

7. Spuštění

Program očekává ve stejném adresáři soubor `mat1` a `mat2` se vstupními maticemi, kde v souboru `mat1` je na prvním řádku uvedená výška 1. matice a v souboru `mat2` je na prvním řádku uvedena šířka 2. matice. Poté lze spustit skript `test.sh`, který provede překlad a spustí program (viz obrázek 3), který vypíše výsledek. Po doběhnutí programu skript smaže vygenerovaný binární soubor.



Obrázek 3: Výpis programu

8. Závěr

V této dokumentaci jsme popsali princip fungování paralelního algoritmu Mesh Multiplication pro násobení matic a odvodili jeho teoretickou složitost. V kapitole o implementaci jsou zmíněny důležité úseky programu pro jeho snadnější pochopení. Dále je znázorněna komunikace mezi procesory v tomto algoritmu pomocí sekvenčního diagramu a v kapitole o testování jsou v grafu vyneseny výsledné časy při měření rychlosti výpočtu různě velkých matic. Výsledná křivka v grafu naměřených hodnot je opravdu s menšími odchylkami lineární, jak bylo teoreticky odvozeno, až na hodnotu 16, kde můžeme pozorovat znatelný nárust času. Je to dáno tím, že server Merlin neumožňuje alokovat 16 fyzických procesorů a bylo je tedy nutné simulovat pomocí přepínače `oversubscribe` na menším počtu procesorů. Zároveň je důležité poznamenat, že naměřený čas je nad celým během programu, tedy včetně IO operací, nastavení prostředí aj., samotné řazení zabere mnohem méně času.