Dokumentace k druhému projektu do PRL – Mesh Multiplication

Kateřina Fořtová (xforto00)

duben 2020

1 Rozbor a analýza algoritmu

Úkolem projektu byla implementace a rozbor jednoho z paralelních maticových algoritmů – Mesh Multiplication. Tento algoritmus vykonává násobení dvou matic na mřížce $n \times k$ procesorů. Matice A má rozměr $m \times n$, matice B má rozměr $n \times k$. Musí tedy platit obecná podmínka násobení matic, kdy počet sloupců první matice je roven počtu řádků druhé matice. Řádky mřížky jsou indexovány indexy v intervalu $1 \ldots m$, sloupce mřížky procesorů pak indexy v rozmezí $1 \ldots k$. Fázově posunuté prvky matic A a B se postupně přivádějí do procesorů prvního řádku a prvního sloupce. Každý z procesorů v mřížce obsahuje prvek c_{ij}, který je počítán na základě přivedených vstupů a a b dle následujícího vzorce: $c_{ij} = c_{ij} + (a \cdot b)$.

Prvky a_{m1} a b_{1k} potřebují k přijetí posledním procesorem $P(m, k)$ počet kroků $m + k + n - 2$. Proměnná $p(n)$ značí počet procesorů.

Časová složitost se dá vyjádřit následovně:

$$t(n) = O(n) \cdot p(n) = O(n^2)$$

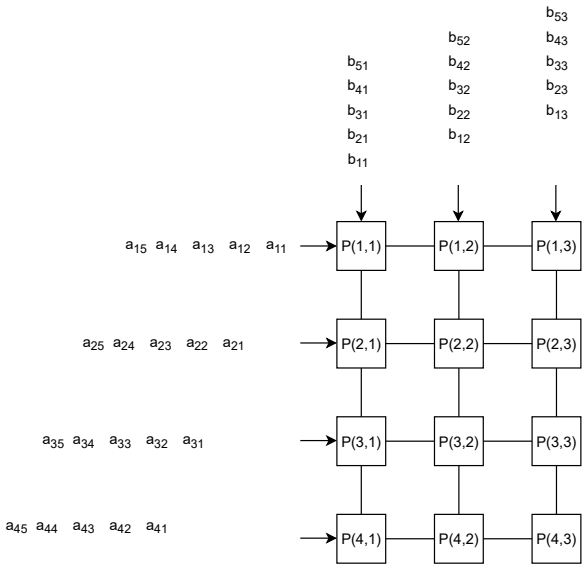
Prostorová složitost lze vyjádřit následovně:

$$s(n) = p(n)$$

Cenu algoritmu lze spočítat jako:

$$c(n) = t(n) \cdot p(n) = O(n^3)$$

Cena algoritmu není optimální. Algoritmus Mesh Multiplication má stejnou cenu jako sekvenční průběh násobení matic. Optimální sekvenční algoritmus pro násobení matic není známý. Při algoritmu Mesh Multiplication jsou vždy v části běhu algoritmu některé z procesorů nečinné. První z procesorů, které byly na počátku zpracovávání matic, jsou nečinné ke konci běhu algoritmu. Procesory, které zpracovávají násobení jako jedny z posledních, jsou zase nečinné na začátku algoritmu. Proto není celková cena algoritmu optimální. [1].



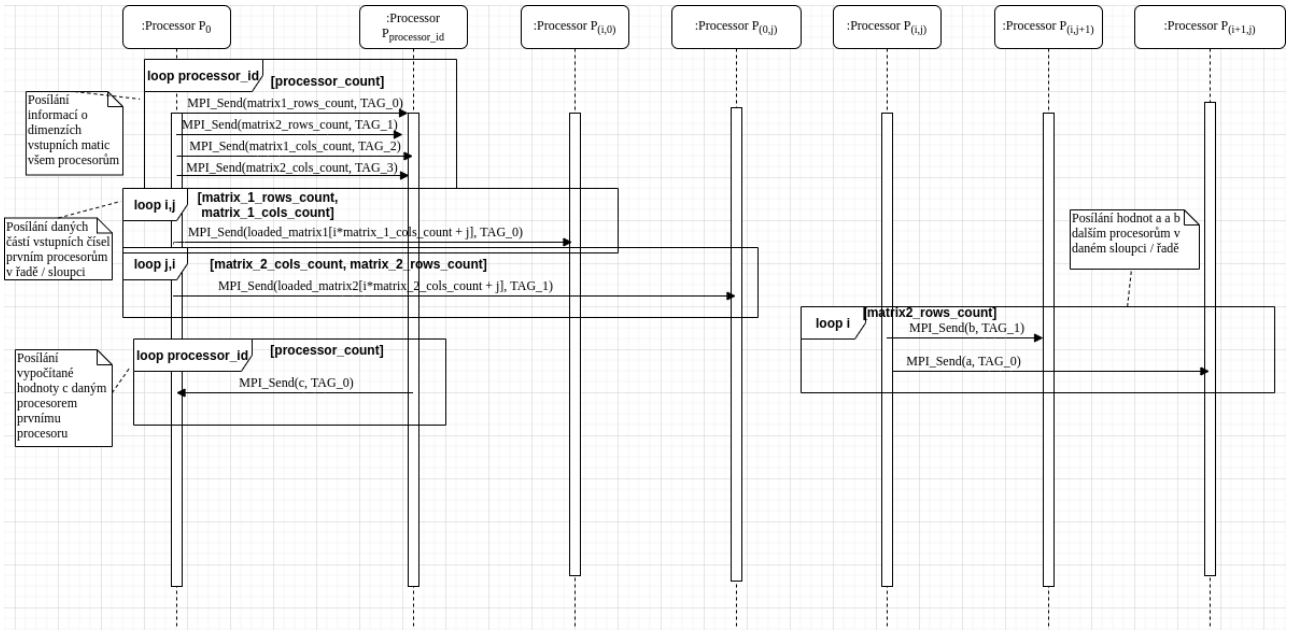
Obrázek 1: Příklad architektury mřížky pro algoritmus Mesh Multiplication

2 Implementace

Algoritmus Mesh Multiplication byl implementován v jazyce C++ s využitím knihovny OpenMPI. Pro veškerou meziprocesorovou komunikaci byly využity pouze funkce `MPI_Send` a `MPI_Recv`. Na začátku algoritmu si první procesor načte vstupní soubory dvou matic, které má algoritmus vynásobit. Postupně do dvou vektorů načte prvky obou matic a zjistí jejich rozměry – počet řádků a sloupců. Je provedena kontrola, zda počet sloupců první matice odpovídá počtu řádků druhé matice, což je nezbytná podmínka pro násobení matic. Informace o dimenzích matic následně první procesor pošle všem ostatním procesorům. Následně si každý procesor spočítá svoje vlastní souřadnice v mřížce procesorů – každý procesor tedy ví, na jaké pozici v mřížce se nachází. První procesor pošle všechny hodnoty načtené matice prvním procesorům v prvním sloupci a prvním řádku. Tyto procesory přijmou daný řádek nebo sloupec matice a uloží si ji do vektoru, ze kterého budou posílat prvky dalším procesorům. Při výpočtu samotného vynásobení si pak na základě pozice v mřížce procesory buď zjistí hodnoty a a b ze svých vektorů, nebo je přijímají od předchozího procesoru. Všechny procesory poté provádějí výpočet své hodnoty $c_{ij} = c_{ij} + (a \cdot b)$. Hodnoty a a b se dále posílají dalším příslušným procesorům v mřížce. První procesor po skončení výpočtu násobení matic prvně zjistí svoji vlastní hodnotu c a uloží ji do vektoru, který bude obsahovat finální získanou matici. Poté přijme vypočítané hodnoty c od všech ostatních procesorů a postupně je za sebe uloží do daného vektoru. Výsledné hodnoty z vektoru jsou následně zobrazeny uživateli ve správné formě výsledné matice spolu s počtem řádků a sloupců. Při meziprocesorové komunikaci bylo využito více tagů, které slouží k správnému oddělení odesílaných a přijímaných hodnot.

3 Komunikační protokol

Komunikační protokol je znázorněn s využitím sekvenčního diagramu. Komunikace mezi procesory je zobrazena pomocí funkcí `MPI_Send` a `MPI_Recv`, kdy v argumentu je pro zjednodušení uvedena pouze posílaná nebo přijímaná hodnota a rozlišovací tag. Sekvenční diagram zobrazuje zasílání informací o dimenzí obou vstupních matic všem procesorům, zasílání části matic pro první procesory na řádku / sloupci mřížky, vlastní zasílání hodnot proměnných a a b dalším procesorům v mřížce na daném řádku / sloupci a zaslání finální vypočtené hodnoty c každým z procesorů prvnímu procesoru, který provádí finální výpis vypočtené matice.



Obrázek 2: Obecný sekvenční diagram pro algoritmus Mesh Multiplication

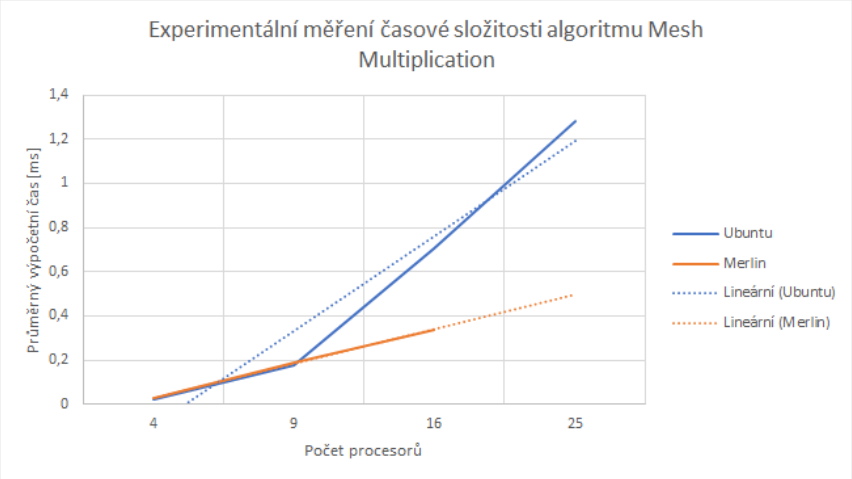
4 Experimenty

Při měření experimentů bylo potřeba provádět měření výpočetního času bez načítání matic ze souboru, přípravných úkonů (zaslání dimenzí matic všem procesorům, uložení matic pro první procesory do vektorů) a nebo finálního výpisu výsledku. Je tedy měřen pouze výpočetní čas algoritmu Mesh Multiplication, tak jak je vysvětlený ve studijních materiálech, tedy do měření není ani zahrnuté finální získání výsledků hodnoty c ze všech procesorů mřížky prvním procesorem. K měření bylo využito funkce z knihovny OpenMPI `MPI_Wtime`, která vrací výpočetní čas v sekundách, pro přehlednost byl čas převeden na milisekundy. Při měření bylo nutné před začátkem a koncem měření zavolat funkci `MPI_Barrier(MPI_COMM_WORLD)`, aby bylo zajištěno, že všechny procesory dosáhly dané části programu. Pouze

první procesor provádí výpis finálního výpočetního času na obrazovku. Projekt byl testován na lokálním systému Ubuntu 20.04.2 LTS (Intel Core i5-8250U CPU @ 1.60GHz × 8) a školním serveru `merlin.fit.vutbr.cz`. Pro každý případ bylo provedeno 30 měření. Bylo experimentováno s různými rozměry matic. Pro školní server byly vynásobeny pouze matice do dimenze 4×4.

	Dimenze obou vstupních matic	**Počet procesorů**	**Min [ms]**	**Max [ms]**	**Průměr [ms]**
Ubuntu	2×2	4	0,009	0,092	0,025
Merlin	2×2	4	0,020	0,035	0,026
Ubuntu	3×3	9	0,121	0,271	0,177
Merlin	3×3	9	0,121	0,288	0,189
Ubuntu	4×4	16	0,288	4,121	0,703
Merlin	4×4	16	0,196	0,477	0,337
Ubuntu	5×5	25	0,675	2,480	1,284

Tabulka 1: Výpočetní čas algoritmu na základě počtu procesorů



Obrázek 3: Graf pro experimentální měření časové složitosti

Graf výše ukazuje vztah mezi počtem procesorů a průměrným výpočetním časem algoritmu na obou zařízeních. Body z obou měření jsou proloženy lineárními spojnicemi trendu, aby bylo vidět možné odchýlení od teoretické časové složitosti. Z teoretického rozboru by měl výpočetní čas růst lineárně s počtem procesorů. Můžeme vidět, že u školního serveru se výpočetní čas navyšuje zhruba se stejnou konstantou, u lokálního stroje toto platí pouze do počtu devíti procesorů. Poté se výpočetní čas navyšuje s vyšší konstantou.

5 Závěr

Úkolem projektu byla implementace paralelního algoritmu Mesh Multiplication. Pro experimentování bylo využito prostředí školního serveru a lokálního stroje. Naměřená časová složitost obecně odpovídala teoretické, avšak při měření na lokálním stroji od určitého počtu procesorů rostl výpočetní čas s trochu vyšším přírůstkem, jak při menším počtu procesorů. Na lokálním stroji bylo experimentováno s rozměry vstupních matic 5×5, na školním serveru byly provedeny experimenty do rozměrů vstupních matic 4×4. Algoritmus byl řádně otestovaný s využitím příkladových matic v zadání a dále s čtvercovými maticemi různých dimenzí. Při experimentech bylo potřeba správně měřit čas pouze samotného algoritmu, nikoliv přípravných úloh nebo výpisů.

Reference

[1] HANÁČEK, P. *Paralelní a distribuované algoritmy – Vyhledávání* [online]. Poslední změna 15. května 2020 [cit. 29. dubna 2021]. Dostupné na: <https://bit.ly/3e0KYas>.