VYSOKÉ UČENÍ TECHNICKÉ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Praktické paralelní programování

Projekt – MPI paralel heat solver

Málek Jakub (xmalek17) 28.4.2024

1. Implementace

Řešení projektu mi zabralo více času, než jsem předpokládal. Největší problém byl asi zorientovat se ve složitější struktuře tříd projektu, zjistit na co která třída a funkce je a jak se používají. Později, jak jsem si udělal základní přehled šla implementace už jednodušeji.

Odevzdávaná část projektu se nachází v souborech *parallelHeatSolver.cpp/hpp*, které obsahují celou implementaci paralelní verze daného solveru. Hlavičkový soubor obsahuje mimo definice metod, také moje atributy, které mi udržují následující hodnoty:

- Pole pro nové a staré teploty, pro parametry a mapu domény
- Počet řádků a sloupců na kolik je daná doména rozřezána
- Parametry pro výpočet nové teploty
- Indexy do 1D pole pro určení počátečních hodnot různý typů halo zón
- Velikost lokální matice
- Velikost celé matice
- Datové typy pro matice, řádky a sloupce
- Komunikátory
- Okno pro nepárovou komunikaci

Ve zdrojovém souboru se pak nachází samotná implementace, kde se při inicializaci volají metody, které inicializují topologii, komunikátory, MPI typy, inicializuji MPI okno pro nepárovou komunikaci, alokuji paměť pro data a připravuji výměnu halo zón. Následují metody pro scater a gether, které pouze rozprostřou data na ranky a následně je sesbírají, pak výpočet dat v halo zónách připravující na výměnu, a nakonec samotná párová a nepárová komunikace. Metoda *run* pak všechny tyto metody volá a dohromady dává funkční výpočet teploty. Navíc je v metodě volán výpočet prostředích hodnot matice k dosažení úplného výsledku. Na závěr každé iterace může být volána funkce, která vypočítá průměr hodnot z prostředního sloupce, pro kontrolu mezivýsledků. V konečné fázi je implementována metoda pro paralelní ukládání do souboru pro uložení výsledů v čase.

2. Profiling

Pro testování projektu jsem použil přiložené scripty, které obsahují spuštění projektu na výpočetních uzlech Barbory, pro získání dat.

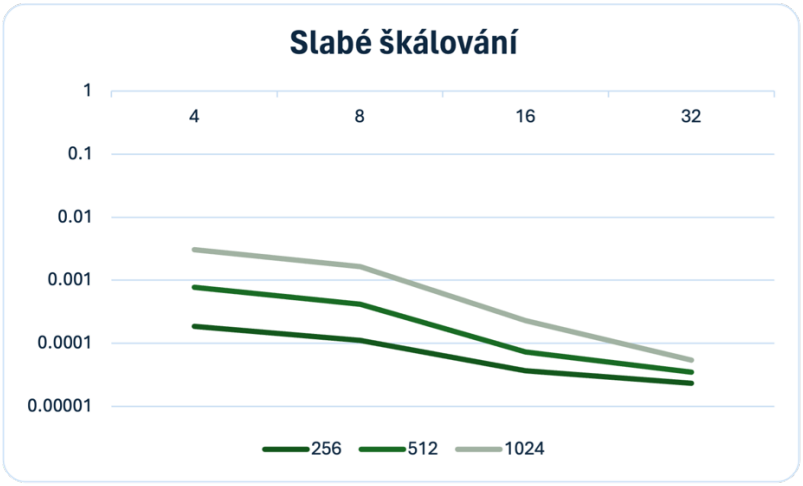
2.1. Slabé a silné škálování

Pro silné a slabé škálování jsem zvolil výpočty nad 4, 8, 16 a 32 procesory s velikostí dat od 256 do 4096. Naměřené hodnoty jsem pak přepsal do tabulky a nechal si vykreslit následující grafy.

Silné škálování by v ideálním případě mělo opisovat přímku y=-x. V mém případě se tak úplně neděje. Nejlépe je na tom datová sada s velikostí dlaždice 4096, kde nejspíše se nejvíce zanedbává počáteční inicializace MPI a komunikace mezi procesy. Oproti tomu datová sada s dlaždicí 256 mírně stoupá, což naznačuje zhoršení výkonu, kvůli distribuci malých částí matice a nadbytečné komunikaci.

Slabé škálování nám ukazuje závislost počtu procesu na úměrně se zvyšující práci, kde by křivka měla být v ideální případě konstantou. V mém případě jde lehce dolů, což naznačuje zhoršení efektivity paralelismu. Opět je důvodem příliš komunikace na malé datové sadě.

Obr. 1 – Silné škálování



Obr.2 – Slabé škálování

2.2. Input / output

V případě, že bych stihl implementovat výstup do souboru, tak bych pro testování použil stejná data jako pro slabé a silné škálování. Testoval bych rychlost sekvenčního a paralelního zápisu na různě velké domény dat a pro různé počty procesorů. Výsledek by měl být takový, že v případě sekvenční varianty rychlost klesá lineárně, zatímco paralelní verze má pro menší data horší výsledky a lepších výsledků dochází až u větších datových sad, kde se potlačuje čas strávený komunikací.

2.3. Profilování pomocí nástroje Vampir

Nástroj Vampir vám je schopný ukázat v čase co který proces prováděl za operaci a kdy čekal na jaké bariéře a z jakého důvodu. Také porovná množství času stráveného v komunikace a času stráveného výpočtem dat. Bohužel jsem toto měření již nestihl v rámci projektu udělat.

3. Závěr

Za pomocí profilování jsme schopni odhalit chyby a nedostatky naší paralelní implementace. Hlavní příčiny nedostatků mohou být zbytečné čekání na jiný proces, špatné zarovnání dat do cache pamětí, špatně zvolený počet procesů, velikost úlohy. Navíc se může aplikace chovat na každém stroji trochu jinak a je tedy těžké odladit nějaký program pro více typu procesorů, které mají různé cache paměti, různé propustnosti sběrnic či síťových karet, nebo jinou topologii.

V ideálním případě pro lepší vyváženost zatížení by bylo dobré načítat data paralelně pro každý rank a nepoužívat funkce scater a gether, které nadměrně zatěžují ROOT_RANK.

Měření je také skresleno zvolením malých datových sad, které nezvládnou dostatečně zanedbat inicializaci MPI a zároveň komunikují s velmi malými bloky dat, kde komunikace stojí více času než samotný výpočet.