Praktické paralelní programování
Projekt č. 1 - MPI a paralelní I/O

David Bayer, ibayer@fit.vutbr.cz
Jiří Jaroš, jarosjir@fit.vutbr.cz

Termín odevzdání: 26. dubna 2024 23:59:59
Hodnocení: až 25 bodů

1 Úvod

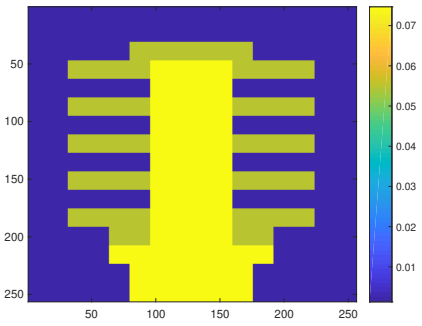
Cílem projektu je osvojit si základní principy programování se zasíláním zpráv s využitím knihovny MPI, použití knihovny HDF5 pro paralelní I/O a vyhodnocení chování implementace pomocí nástrojů založených na Score-P. Součástí projektu je také ověření škálování na několika uzlech superpočítače Barbora.

Jako modelový problém byla zvolena simulace šíření tepla 2D řezem procesorového chladiče věžovité konstrukce. Tepelným zdrojem je tedy procesor (zde aproximováno konstantní teplotou), ze kterého se teplo „difuzí“ šíří do konstrukce chladiče odkud je odváděno proudem vzduchu. Tepelná energie tedy opouští zkoumanou oblast spolu s proudem vzduchu kolmým k ose řezu chladiče.

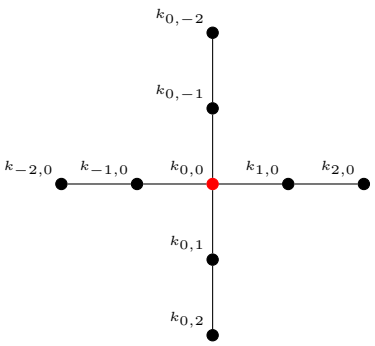
Průběh simulace je ukládán v celé doméně pomocí knihovny HDF5 do souboru pro pozdější analýzu.

1.1 Numerická metoda řešení

Pro řešení problému byla zvolena varianta metody konečných diferencí (FDTD[1]). Tato metoda je jedním z nejjednodušších, ale bohužel také nejméně efektivních přístupů k řešení této třídy problémů. Metoda konečných diferencí v čase (Finite Difference Time Domain) je založena na diskretizaci simulační domény do uniformní mřížky v prostoru a čase, tedy $N_x \times N_y \times N_t$. Zde uvažujeme pouze čtvercovou mřížku, kde $N_x = N_y$. V každém bodě této mřížky je definován parametr tepelné vodivosti (α) materiálu v daném bodě (měď, hliník nebo vzduch). Pro oblasti, které jsou tvořeny vzduchem dále definujeme parametr β, který proporčně reprezentuje proudění vzduchu.



(a) Tepelná vodivost v doméně



(b) 9-bodový (4+4-okolí) stencil

Obrázek 1: Materiály v médiu (a) a výpočetní stencil[2](b) používaný v simulaci.

[1] http://www.eecs.wsu.edu/~schneidj/ufdtd/ufdtd.pdf
[2] https://en.wikipedia.org/wiki/Stencil_code

Samotný výpočet každého z N_t časových kroků spočívá v aktualizaci každého bodu domény na základě jeho 4+4-okolí (viz obrázek **??**). V případě simulace s homogenní tepelnou vodivostí a bez odvodu tepla je každý bod vypočten jako průměr jeho vlastní teploty a teplot v jeho 4+4-okolí. Rozšíříme-li simulaci o heterogenní materiál a odvod tepla lze aktualizaci v každém kroce zapsat rovnicí jako:

$$u_{(i,j)}^{t+1} = \beta_{(i,j)} u_{(i,j)}^{0} + \left(1 - \beta_{(i,j)}\right) k_{(i,j)} \sum_{(m,n)\in S} \alpha_{(i+m,j+n)} u_{(i+m,j+n)}^{t}, \tag{1}$$

Kde $u_{(i,j)}^{t}$ je teplota v bodě (i, j) v čase t, $\beta_{(i,j)}$ je parametr odvodu tepla, který proporčně odpovídá rychlosti proudění vzduchu v bodech domény, kde je materiálem vzduch (jinde má hodnotu 0). S je množina bodů určující tvar okolí (viz obrázek **??**). Parametr $k_{(i,j)} = 1/\sum_{(m,n)\in S} \alpha_{(i+m,j+n)}$ normalizuje tepelnou vodivost v okolí vyhodnocovaného bodu.

Problém je doplněn o počáteční a hraniční podmínky, kde počáteční teplota v doméně je stejně jako teplota okolí nastavena na 20 °C. Hraniční podmínka je tvořena konstantní teplotou (tedy Dirichletova podmínka) a je modelována jako pásek 2 bodů kolem okrajů domény. Hraniční podmínka také modeluje zdroj tepelné energie, kde v místě kontaktu procesoru s chladičem je teplota 100 °C.

2 Popis projektu a implementace

Obsah archivu zadání je rozdělen do složek `sources` a `scripts`.

- Složka `sources` obsahuje zdrojové kódy implementací simulátoru a generátoru vstupních souborů simulace. Součástí složky jsou zdrojové soubory `ParallelHeatSolver.*`, ve kterých třeba implementovat paralelní verzi simulátoru.
- Složka `scripts` obsahuje skript pro vygenerování testovacích dat různé velikosti a sadu `SLURM` skriptů, které lze použít pro snadné ověření škálování vašeho řešení.

2.1 Struktura zdrojového kódu

Zdrojový kód projektu je rozdělen do několika tříd, které jsou implementovány v následujících souborech:

- Soubor `data_generator.cpp` obsahuje implementaci generátoru vstupních dat a na ostatních zdrojových souborech je nezávislý.
- Soubor `main.cpp` obsahuje vstupní bod programu a zajišťuje inicializaci `MPI`, parsování parametrů aplikace, načtení vstupního souboru a následné spuštění příslušné implementace simulátoru. Umožňuje také provést verifikaci výsledků (porovnáním se sekvenční implementací), nebo zpřístupnit ladící informace v podobě uložení výsledku simulace do obrázku.
- Soubor `AlignedAlloc.hpp` obsahuje implementaci konceptu *Allocator*[3] pro alokaci zarovnané paměti. Ten je pak možné použít např. pro `std::vector`.
- Soubor `Hdf5Handle.hpp` implementuje šablonovanou třídu `Hdf5Handle`, která umožňuje bezpečnější práci s `HDF5` objekty a jejich automatickou destrukci pomocí konceptu `RAII`[4].
- Soubory `utils.*` obsahují pomocné funkce pro ověření výsledků simulace a ukládání obrázků.
- Soubory `MaterialProperties.*` obsahují implementaci třídy reprezentující informace o materiálu načteného ze vstupního souboru, které následně zpřístupní zbytku aplikace.
- Soubory `SimulationProperties.*` implementují třídu `SimulationProperties`, která slouží pro parsování argumentů předaných aplikaci na příkazové řádce.
- Soubory `HeatSolverBase.*` obsahují implementaci základní třídy solveru (`HeatSolverBase`), která zajišťuje jeho základní funkci a slouží jako bázová třída pro jeho různé implementace. Nejdůležitější částí třídy jsou metody `computePoint(...)`, která implementuje rovnici **??** ve specifikovaném bodě a `run(...)`, jenž je ryze abstraktní metodou umožňující provést výpočet simulace. Dostupná je také metoda `updateTile(...)`, která aplikuje `computePoint(...)` na dlaždici paralelně pomocí prostředků `OpenMP`[5] (*multithreading* i vektorizace).

[3] https://en.cppreference.com/w/cpp/named_req/Allocator
[4] https://en.cppreference.com/w/cpp/language/raii
[5] https://www.openmp.org

- Soubory `SequentialHeatSolver.*` obsahují referenční (sekvenční) implementaci solveru. Tyto soubory také ukazují, jak pracovat se zarovnaným alokátorem a `HDF5` objekty s pomocí `Hdf5Handle`.
- Soubory `ParallelHeatSolver.*` obsahují kostru implementace paralelního solveru, kterou je vaším úkolem doplnit.

2.2 Sekvenční solver

Sekvenční verze algoritmu je implementovaná ve třídě `SequentialHeatSolver`, kterou naleznete v souborech `SequentialHeatSolver.*` (některé znovupoužitelné části lze také nalézt v její bázové třídě `HeatSolverBase`). Tato verze slouží jako referenční řešení problému bez použití paralelizace a lze ji použít pro ověření výsledků paralelní verze. Kód této implementace je okomentován komentáři s číslováním, které odpovídá následujícím odrážkám.

1. Pokud byl specifikován název výstupního `HDF5` souboru je tento soubor vytvořen, jinak se průběh simulace neukládá. Konstruktor třídy také alokuje pomocné pole.
2. Prvním krokem je inicializace výstupního i pomocného pole počátečními hodnotami ze vstupního pole `mMaterialProps.getInitialTemperature()`.
3. Hlavní smyčka simulace, kde nová teplota v každém bodě domény je vypočtena pouze na základě jejích předchozích hodnot. Aby nedocházelo k výpočtu nad nesprávnými daty (např. použití hodnoty již aktualizovaného souseda) je k výpočtu použita dvojice polí (`workTempArrays[]`), kde `workTempArrays[0]` obsahuje vždy právě aktualizované hodnoty. Na konci hlavní smyčky (před prohozením polí) tedy `workTempArrays[0]` odpovídá času t a `workTempArrays[1]` času $t - 1$.
4. Průchodem všech bodů domény (krom bodů hraničních podmínek) vypočteme pomocí metody `computePoint` nové hodnoty teploty. V případě paralelní implementace lze použít `updateTile` na místo explicitních smyček. Implementaci těchto metod je možné nalézt v `HeatSolverBase.hpp` a `HeatSolverBase.cpp`.
   (a) Spočteme index aktuálního bodu a jeho osmi sousedů (indexujeme 2D pole uložené po řádcích jako 1D pole).
   (b) Normalizujeme `domainParams` (tepelnou vodivost materiálu) na hodnoty v intervalu $\langle 0; 1\rangle$ (nula značí dokonalý izolant, jednička supravodič). Tato úprava urychlí šíření tepla doménou, tudíž zmenší počet iterací celé simulace nutných pro dosažení stejného výsledku.
   (c) Spočítáme novou teplotu bodu na základě teploty a tepelné vodivosti sousedů i jeho samotného. Zde je důležité uvědomit si potřebu dvou polí `oldTemp` a `newTemp`. Kdybychom používali pouze jedno pole, někteří sousedé aktuálně počítaného bodu už by měli aktualizované hodnoty z této iterace a dostávali bychom chybné výsledky. V paralelní verzi by to navíc vedlo na nedeterministické chování (pokud by souseda aktuálního bodu mělo na starosti jiné vlákno, mohlo by v prvním běhu programu aktualizovat hodnotu tohoto souseda před aktuálním bodem a podruhé třeba až po aktuálním bodu).
   (d) Pokud aktuální bod reprezentuje vzduch, snížíme jeho teplotu. Tím simulujeme ochlazování chladiče aktivním větráním.
5. Spočítáme průměrnou teplotu v prostředním sloupci domény. Tato část má větší význam až v paralelní verzi, kde bude možné porovnávat průběh této hodnoty oproti té sekvenční (musí být totožné - s odchylnou na úrovni numerické chyby *float*).
6. Pokud se má v aktuální iteraci zapisovat na disk, zapíšeme nově spočtená data do souboru.
7. Prohodíme ukazatele polí `workTempArrays[0]` a `workTempArrays[1]`. Tím budeme v následující iteraci považovat právě vypočtený výsledek jako výchozí stav.
8. Pomocí metody `printProgressReport(...)` vypíšeme postup simulace. Tato metoda vypisuje postup pouze každou N-tou iteraci (toto je zajištěno voláním metody `shouldPrintProgress(...)`, kterou můžete použít pro detekci iterací simulace, kdy je třeba vypočíst průměrnou teplotu).
9. Pomocí metody `printFinalReport(...)` vypíšeme report o průběhu simulace (jako potřebný čas a průměrná teplota po poslední iteraci).
10. Pokud je celkový počet iterací lichý, jsou poslední výsledky uloženy v pomocném poli `mTempArray` a je tedy nutné je zkopírovat do výstupního pole.

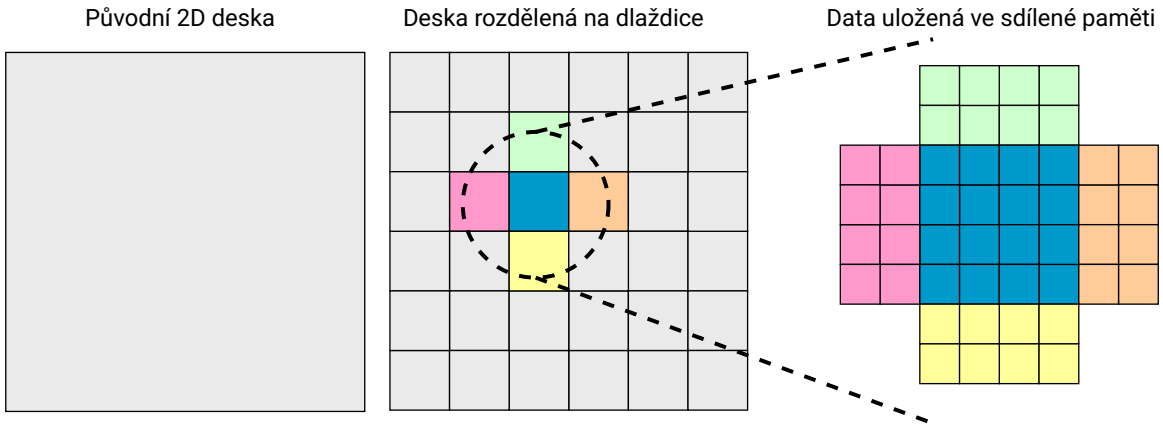
3 Co je úkolem studenta?

Jak bylo zmíněno v úvodu, cílem projektu je především osvojit si práci s `MPI` a `HDF5` a také prokázat porozumění chování jednoduchého (ale přesto realistického) distribuovaného algoritmu. Prvním úkolem je tedy vytvořit paralelní/distribuovanou implementaci simulace šíření tepla ve 2D. To znamená implementovat třídu `ParallelHeatSolver` v souborech `ParellelHeatSolver.hpp` a `ParellelHeatSolver.cpp`. Druhým úkolem je pak vytvořit dokument `xloginNN.pdf` (dle vašeho loginu), který bude obsahovat vyhodnocení chování vašeho řešení (tedy především grafy škálování a poznatky zjištěné profilovacími nástroji). Výstupem je tedy trojice souborů:

- `ParellelHeatSolver.hpp` - kód implementace
- `ParellelHeatSolver.cpp` - kód implementace
- `xloginNN.pdf` (dle vašeho loginu) - vyhodnocení řešení

3.1 Paralelní MPI implementace [až 19 bodů]

Typickým přístupem je dekompozice simulační domény, kde každý proces zpracovává pouze její část. Optimální volbou je dekompozice do čtvercových, nebo obdélníkových dlaždic (v závislosti na počtu procesů). Tento přístup minimalizuje počet bodů sdílených mezi procesy a tedy potřebnou komunikaci. V našem případě tvoří sdílené body (tzv. *halo zóny*) okolí hloubky 2 na každé hraně dlaždice (obrázek ??).



Obrázek 2: Příklad 2D dekompozice domémy. Dekompozice v jedné dimenzi redukuje počet sousedů na polovinu.

Důležitým prvkem distribuovaných simulací tohoto typu je také překrytí výpočtu a komunikace. Za tímto účelem je třeba nejdříve vypočítat hodnoty v halo zónách, tyto body odeslat sousedním procesům a až poté počítat vnitřní body dlaždice. Po vyhodnocení vnitřních bodů dojde k přijetí halo zón od okolních dlaždic (ty budou použity v následující iteraci). K tomuto účelu je vhodné použít neblokující volání MPI.

1. Konstruktor třídy je rozdělen do několika částí, z nichž každá je zodpovědná za inicializaci různých součástí simulátoru a je třeba je volat ve správném pořadí v závislosti na vaší implementaci.

   - Metoda `initGridTopology()` inicializuje topologii procesů. Jejím úkolem je rozdělit doménu o straně $N = 2^k$ mezi $P = 2^p$ procesů, kde $k, p \in \{0, 1, \ldots, n\}$ a $k > p$. Požadovanou dekompozici lze zjistit pomocí `mSimulationProps.getDecompGrid(nx, ny)`. Implementace musí podporovat následující 1D a 2D dekompozice:

     (a) 1D dekompozice ve směru x, tedy $n_x = P$ a $n_y = 1$.

     (b) Rovnoměrná 2D dekompozice, kde pro sudé p lze doménu rozdělit do čtvercových dlaždic jako $n_x = n_y = \sqrt{P}$ (např. pro 16 procesů a doménu 256×256 budou procesy uspořádány v mřížce 4×4 a každý bude mít svoji lokální 64×64 dlaždici). V případě lichého p je

nutné doménu rozdělit do obdélníků jako: $n_x = \sqrt{P/2}$, $n_y = 2n_x$ (např. pro 32 procesů a 256×256 vznikne mřížka 4×8 a dlaždice 64×32).

Metoda by měla vytvořit komunikátor kartézské mřížky, určit pozici procesu v této mřížce a zjistit sousedy, se kterými si bude vyměňovat *halo zóny*. Rovněž by měla vytvořit komunikátor pro paralelní výpočet průměrné teploty v prostředním sloupci domény.

- Metoda `initDataDistribution()` inicializuje distribuci dat mezi procesy. Jejím úkolem je inicializovat proměnné, jako jsou globální a lokální rozměry dlaždic, a `MPI` datové typy pro kolektivní operace *scatter* a *gather*. **Protože počítáme s (4+4)-okolím musí mít okraje dlaždice odpovídající šířku (tedy 2 body, proměnná `haloZoneSize`).**
- Metoda `initHaloExchange()` má za úkol inicializovat proměnné a `MPI` datové typy pro neblokující výměnu okrajů mezi procesy. Ty jsou prováděny buď pomocí *peer-to-peer* (P2P) komunikací nebo vzdáleného přístupu do paměti (RMA) v závislosti na zvoleném módu simulace (lze zjistit pomocí `mSimulationProps.isRunParallelP2P()` a `mSimulationProps.isRunParallelRMA()`).
- Metoda `allocLocalTiles()` alokuje paměť pro lokální dlaždice a jejich okolí. Je třeba alokovat:
  - 1 pole pro specifikaci materiálu (datový typ *int*).
  - 1 pole pro teplotní vlastnosti materiálu (datový typ *float*).
  - 2 pole pro teplotu (jedno pro aktuální a jedno pro předchozí hodnoty, datový typ *float*).

Pokud je specifikován název výstupního souboru vytvoříme nový výstupní HDF5 soubor v proměnné `mFileHandle` metodami `openOutputFileSequential()` nebo `openOutputFileParallel()` v závislosti na tom, zda je vyžadováno použití paralelního I/O (zjistíme pomocí `mSimulationProps.useParallelIO()`). V opačném případě se průběh simulace neukládá. **V případě sekvenčního I/O bude soubor vytvářet pouze jeden (root) proces, naopak při použití paralelního I/O musí soubor vytvářet všechny procesy!** V metodě `openOutputFileParallel()` dále nutno implementovat:

- vytvoření *file access property list* (FAPL) v proměnné `fapl` a
- nastavení využití MPI-IO spolu s vhodným zarovnáním souboru.

2. Po implementaci inicializace objektu je potřeba implementovat další pomocné metody simulátoru.
   - Metody `scatterTiles(...)` a `gatherTiles(...)` implementují kolektivní distribuci globální dlaždice nebo sebírání lokálních dlaždic. Jedná se o šablonované metody omezené pro datové typy *int* a *float*. Pomocí *type trait* `std::is_same_v<T, type>` lze zjistit, zda je datový typ `T` shodný s `type` a tím zvolit správný `MPI` datový typ pro kolektivní komunikace.
   - Metoda `computeHaloZones(...)` implementuje výpočet okrajů lokální dlaždice. Výpočet lze provést pomocí metody `updateTile(...)` s náležitými parametry. Věnujte pozornost tomu, abyste nepočítali některé části okrajů dvakrát.
   - Metoda `startHaloExchangeP2P(...)` implementuje neblokující výměnu okrajů pomocí neblokujících P2P komunikací. Vytvářené MPI requesty ukládejte do výstupního pole `requests`. V případě, že proces nemá v daném směru žádného souseda, je třeba nastavit odpovídající prvek pole `requests` na `MPI_REQUEST_NULL`.
   - Metoda `startHaloExchangeRMA(...)` implementuje neblokující výměnu okrajů pomocí vzdáleného přístupu do paměti (RMA).
   - Metody `awaitHaloExchangeP2P(...)` a `awaitHaloExchangeRMA(...)` implementují čekání na dokončení výměny okrajů.
   - Pomocná metoda `shouldComputeMiddleColumnAverageTemperature()` vrací true, pokud se proces účastní výpočtu průměrné teploty v prostředním sloupci domény.
   - Metoda `computeMiddleColumnAverageTemperatureSequential(...)` implementuje sekvenční výpočet průměrné teploty v prostředním sloupci domény. Tato metoda pracuje nad globální dlaždicí a měla by být volána pouze *root* rankem. Inspirujte se sekvenční verzí a využijte redukce v OpenMP.
   - Metoda `computeMiddleColumnAverageTemperatureParallel(...)` implementuje paralelní výpočet průměrné teploty v prostředním sloupci domény. Inspirujte se sekvenční verzí, využijte redukce v OpenMP a MPI.

- Metoda `storeDataIntoFileParallel(...)` implementuje paralelní zápis výsledků simulace do souboru. Zde je třeba doplnit spočítání globálních offsetů a velikostí pro každý proces a následně zapsat lokální dlaždice do souboru.

3. Metoda `run()` spustí simulaci. Před samotným začátkem simulace je ještě třeba provést několik kroků:

   (a) distribuovat počáteční teploty a vlastnosti materiálu mezi procesy pomocí metody `scatterTiles(...)` (počáteční hodnoty má k dispozici pouze *root* proces, který je získá pomocí metod `getInitialTemperature()`, `getDomainParameters()` a `getDomainMap()` objektu `mMaterialProps`),

   (b) provést výměnu okrajů počáteční teploty a parametrů materiálu mezi procesy pomocí metody `exchangeHaloZonesP2P(...)` a počkat na její dokončení,

   (c) zkopírovat obsah prvního pole s počáteční teplotou do do druhého.

   V simulační smyčce jsou definovány proměnné `oldIdx` a `newIdx` s hodnotami indexů se starými a novými templotami domény. Ve smyčce je pak potřeba:

   (a) vypočítat nové hodnoty teploty v okrajích lokální dlaždice voláním metody `computeHaloZones(...)` a vyměnit je s okolními procesy pomocí metod `startHaloExchangeP2P(...)` nebo `startHaloExchangeRMA(..` v závislosti na zvoleném módu simulace,

   (b) dopočítat nové hodnoty teploty vnitřních bodů lokální dlaždice metodou `updateTile(...)`,

   (c) počkat na dokončení výměny okrajů metodami `awaitHaloExchangeP2P(...)` nebo `awaitHaloExchangeRMA(..`

   (d) uložit výsledky simulace do souboru (pokud je to požadováno) pomocí sekvenčního nebo paralelního I/O (zjistíme pomocí `mSimulationProps.useParallelIO()`) a

   (e) vypočítat průměrnou teplotu v prostředním sloupci domény a vypsat informace o průběhu simulace.

   Nakonec je potřeba sesbírat výsledky simulace z jednotlivých procesů metodou `gatherTiles(...)` na *root* proces do parametru `out` a provést finální report metodou `printFinalReport(...)`.

4. V destruktoru objektu volejte metody `deinit*()` a `dealloc*()` v opačném pořadí, než byly volány jejich inicializační ekvivalenty v konstruktoru. V deinicializačních metodách uvolněte všechny alokované zdroje.

3.2 Vyhodnocení chování řešení [až 6 bodů]

Součástí řešení projektu je dokument obsahující vyhodnocení chování vašeho řešení. Účelem tohoto krátkého dokumentu v **rozsahu 1 až 2 strany** je prokázat, že jste porozuměli chování paralelního algoritmu a jste schopni rozhodnout, zda je toto řešení efektivní.

Dokument musí obsahovat grafy silného a slabého škálování vytvořené na základě dat získaných pomocí testovacích skriptů (tedy pro 1 až 8 uzlů) a měl by zodpovědět následující otázky:

- Jaký je rozdíl mezi škálováním/efektivitou 1D a 2D dekompozice a čím je tento rozdíl způsoben?
- Jaký je vliv paralelního I/O v porovnání se sekvenčním?
- Jakým způsobem lze zefektivnit paralelní I/O?
- Jak se liší množství komunikace mezi jednotlivými procesy v 1D a 2D dekompozici? Je zátěž vyrovnaná?
- Jaký přínos má překrytí komunikace a výpočtu?

Tyto otázky by mělo být možné zodpovědět na základě vašich grafů škálování a měření pomocí nástrojů Score-P, Cube a Vampir. Očekáváme, že v dokumentu uvedete screenshoty z těchto nástrojů s komentáři, případně uvedete jiný způsob, jak byly vaše poznatky získány (např. "Do kódu jsem si přidal další časovač pro měření nepřekryté části komunikace").

3.3 Poznámky k postupu při řešení

- Nejdříve je vhodné implementovat rozdělení domény mezi jednotlivé procesy a následné sesbírání a ukládání průběhu simulace.
- Druhým nejjednodušším krokem je implementovat paralelní verzi bez překrytí komunikace výpočtu vnitřní části dlaždice (tj. nejdříve počítáme celou dlaždici a pak vyměňujeme okraje).
- Pokud se rozhodnete nevyužít metodu `updateTile(...)` a její funkcionalitu budete implementovat sami, **nezapomeňte na paralelizaci a vektorizaci pomocí OpenMP**. Jinak se hybridní implementace nebude chovat očekávaně.
- Rozdělení domény mezi jednotlivé procesy a následné sesbírání průběhu simulace nemusí být realizováno pomocí vzdáleného přístupu do paměti (v módu 2). Bude postačovat když se tímto způsobem budou vyměňovat halo zóny. Předpokládá se překrytí vypočtu s komunikací, a proto volte vhodnou metodu která umožňuje souběžný vzdálený přístup a lokální modifikaci hodnot.
- Pomocí `MPI_Type_vector` nebo `MPI_Type_create_subarray` si můžete vytvořit datové typy pro adresování např. dlaždice uvnitř domény nebo sloupce uvnitř dlaždice. Pozor, zřejmě bude nutné ošetřit tyto typy pomocí `MPI_Type_create_resized`.
- Data okolních bodů můžete držet buď v samostatných polích (pak ale musíte implementovat vlastní verzi `computePoint(...)`) nebo můžete dlaždici vytvořit o něco větší.
- Pro výpočet průměrné hodnoty v prostředním sloupci **je nutné** využít vlastní komunikátor.
- Při ladění paralelního I/O se soustřeďte na eliminaci souběžného přístupu a odstranění nutnosti přeskládání dat před zápisem. Pro obdržení plného bodového hodnocení není nutno prokázat nárůst výkonu, optimalizace ale musí být logicky odůvodněna ve odevzdaném dokumentu.
- Jelikož je v rámci zjednodušení povolen pouze počet MPI procesů roven mocnině dvou, budeme spouštět na jednom uzlu Barbory maximálně 32 procesů. Ukázka takového spuštění je v `*.sh` souborech ve složce `scripts`. Zpravidla stačí předat správný počet procesů SLURM systému

  ```
  salloc ... -N 1 --ntasks-per-node 32 ...
  ```

  a program spustit jako

  ```
  mpiexec -np <P> ./ppp_proj01 <parametry>
  ```

4 Překlad a spuštění projektu

Implementaci je možné testovat na libovolném stroji za předpokladu, že jsou dostupné následující knihovny a nástroje:

- C++ překladač s podporou C++17
- CMake verze 3.22 nebo novější - https://cmake.org/cmake/help/latest/
- MPI (Intel MPI, Open MPI nebo MSMPI, ale měly by fungovat i ostatní) - https://www.open-mpi.org/doc/current/
- Parallel HDF5 (přeložit lze i se sekvenční, ale nebude fungovat paralelní IO) - https://portal.hdfgroup.org/display/HDF5/HDF5
- (Score-P) pro analýzu výsledné implementace - http://scorepci.pages.jsc.fz-juelich.de/scorep-pipelines/docs/scorep-6.0/html/

Za předpokladu, že jsou tyto závislosti dotupné lze rozbalený projekt přeložit pomocí CMake. Pokud váš překladač nepodporuje OpenMP, jej zakázat pomocí přepínače `ALLOW_OPENMP=OFF`. Výsledkem by měla být dvojice spustitelných souborů: `ppp_proj01` (samotný projekt) a `data_generator` (generátor vstupních dat).

V případě, že budete projekt překládat pro analýzu (profilování nebo trasování) pomocí nástroje Score-P je situace mírně komplikovanější. Jelikož Score-P pracuje na úrovni překladače je nutné pro

překlad použít Score-P wrapper. V případě překladu pomocí CMake však chceme wrapper použít poze pro překlad, ale ne při konfiguraci projektu pomocí CMake. Toho lze dosáhnout následujícím postupem: Výsledkem jsou opět spustitelné soubory, v tomto případě však obsahují značky a volání nástroje Score-P a při jejich spuštění obvyklým způsobem dojde k profilování a uložení výsledků do složky `scorep_YYYYMMDD...` v aktuálním adresáři.
Při změně prostředí, jako změna modulů, nebo nastavení proměnných CC/CXX doporučujeme vymazat obsah adresáře "build", nebo vytvořit nový!

4.1 Lokální PC

Na lokálním PC je možné závislosti nainstalovat z repositářů distribuce, nebo přeložit ze zdrojových kódů:

- OpenMPI - https://download.open-mpi.org/release/open-mpi/v4.0/openmpi-4.0.3.tar.bz2
- HDF5 - https://www.hdfgroup.org/downloads/hdf5/source-code

Spuštění projektu na lokálním PC je pak snadné pomocí: `mpiexec -np <P> ./ppp_proj01 ...`

4.2 Barbora

K výpočetnímu clusteru Barbora se můžete připojit obvyklým způsobem, avšak není vhodné spouštět paralelní úlohy přímo na login uzlu. Login uzel by měl být využíván pouze k překladu a vyhodnocení výsledků. Pro překlad použijte moduly `Intel 2021b`, které je možné načíst pomocí: nebo Je možné požádat o přidělení uzlu v interaktivním módu pro testování projektu příkazem:

Skripty pro měření škálování

Pro měření škálování je připraveno několik skriptů v adresáři `scripts`. Skripty předpokládají, že jsou spuštěny z adresáře, kde se nacházejí, tedy `scripts` a očekávají projekt přeložený pomocí `intel/2021b` v adresáři `build` (tedy `build/ppp_proj01` vzhledem k samotnému skriptu). Skripty také předpokládají vstupní data vygenerovaná pomocí skriptu `generate_data.sh`.

Prvním skriptem je `run_full_mpi_2d.sh`, který testuje škálování 2D dekompozicí domén o velikosti hrany 256 až 4096 mezi 1 až 128 procesů na 8 uzlech. Pro každý počet procesů a dekompozici je spuštěny testy s vypnutým, sekvenčním a paralelním I/O. Výsledky testů jsou uloženy do CSV souboru.

Skripty `run_full_hybrid_1d.sh` a `run_full_hybrid_2d.sh` pracují obdobně, avšak používají pouze 1 až 32 procesů, kde každý proces použije k výpočtu 6 vláken. Tím dochází ke změně poměru mezi časem potřebným k simulaci a komunikaci (uvažujeme-li shodnou dekopozici).

Parallel I/O

Pro testování paralelní verze I/O je třeba správně nastavit souborový systém `Lustre Filesystem` (LFS). Aby bylo možné zapisovat z několika uzlů paralelně, je nutné nastavit patřičné rozdělení mezi několik "object storage targets" (OSTs), to lze provést příkazem: Tímto příkazem dojde k nastavení LFS, tak aby rozložilo zátěž mezi 16 OSTs a každý OST bude dostávat blok o velikosti 1 MB. Toto nastavení by se mělo projevit nárůstem výkonu paralelního I/O při práci s velkými doménami na větším počtu uzlů.
Změny nastavení LFS se projevují pouze na nově vytvořených souborech, po změně nastavení je tedy vhodné existující výstupní soubory odstranit!

Skripty pro Score-P

Za předpokladu, že máte projekt přeložený pomocí `Score-P` (`Score-P/8.0-iimpi-2021b`) a `intel/2021b` ve složce `build_prof` můžete použít skript `run_scorep_profile.sh` pro vygenerování profilovacích a trasovacích informací. Tento skript spustí projekt celkem 4×, 1D a 2D dekompozici s povoleným profilováním pomocí `Score-P` a obě dekompozice znovu s povoleným trasováním. Ve všech případech je použito celkem 16 procesů na 4 uzlech a 6 vláken na proces s doménou o velikosti 1024 a vypnutým zápisem na disk. Jelikož velikost trasovacích souborů roste relativně rychle, jsou běhy omezeny na 100 iterací a navíc je pro trasování použit filtrovací soubor (`ppp_scorep_filter.flt`) s následujícím obsahem:

```
SCOREP_REGION_NAMES_BEGIN
    EXCLUDE
        HeatSolverBase::computePoint
        *std::*
SCOREP_REGION_NAMES_END
```

Tím je z trasování vyloučena metoda `HeatSolverBase::computePoint` a jakékoliv volání ze jmeného prostoru `std`. Důrazně doporučujeme do filtrovacího souboru přidat funkce/metody, které jsou volané pro každý bod domény zvlášť.

Výsledkem profilování a trasování je čtveřice adresářů s názvy ve tvaru "`scorep_YYYYMMDD...`", jejichž obsah můžete načíst nástroji `Cube` (profilovací data) nebo `Vampir` (trasovací data).

4.3 Ovládání generátoru a simulátoru

Generátor má následující parametry spuštění:

```
  -o <string>  - název výstupního hdf5 souboru
  -n <integer> - velikost domény (pouze mocnina dvou až do 1024,
                 více by se počítalo příliš dlouho)
  -H <float>   - teplota procesoru
  -C <float>   - teplota chladiče a okolního vzduchu v počátečním stavu
  -h           - zobrazí nápovědu
```

Simulátor šíření tepla má tyto parametry:

```
Povinné argumenty:
  -m <0-2>     - mód simulace (mód 0 - sekvenční verze, mód 1 - paralelní verze P2P,
                 mód 2 - paralelní verze RMA)
  -n <integer> - počet iterací (časových kroků) simulace, více == delší běh
  -i <string>  - název souboru s vlastnostmi materiálu v doméně (generuje data_generator)

Volitelné argumenty:
  -t <integer> - počet vláken na jeden MPI proces hybridní verze (defaultně 1)
  -o <string>  - název výstupního souboru; pokud není zadán, nic se nezapisuje
  -w <integer> - hustota zápisu na disk (např. hodnota 10 znamená, že se do souboru
                 zapíše výsledek každé desáté iterace)
  -a <float>   - air flow rate (rychlost proudění vzduchu žebry chladiče), rozumné
                 hodnoty jsou v intervalu <0.5, 0.0001> (0.5 značí odebrání 50% tepla
                 po každém časovém kroku)
  -d           - debug mode - viz dále
  -v           - verification mode - porovná výsledky sekvenční a paralelní verze
                 a vypíše OK nebo FAILED (pouze při parametru -m = 1 a -m = 2)
  -b           - batch mode - informace o běhu jsou na stdout vypisovány v CSV formátu
  -B           - batch mode - navíc vypíše hlavičku CSV, implikuje -b
  -p           - paralelní I/O - příznak povolení paralelního zápisu do souboru
  -r <string>  - vytvoří (v případě současného použití -d nebo -v) obrázek
                 s vizualizací stavu po poslední iteraci (hodnoty jsou normalizované!)
  -g           - nastaví dekompozici na 2D (uniformní, nebo v poměru 1:2)
```

5 Debugging

Výsledná aplikace má v sobě zabudovaný způsob pro připojení debuggeru. Podmínkou je, že aplikace běží na jednom uzlu (nebo lokálně na PC), je spuštěna v interaktivním módu a byla přeložena s debug symboly (`Debug` konfigurace).

1. Nejprve je program třeba spustit s parametrem `-d` (zbytek parametrů je stejný jako pro běžný běh). Doporučujeme spustit program pouze s dvěma procesy.

   ```
   mpiexec -np <P> ./ppp_proj01 -d -m 1 -n 100 -i input.h5 -o output.h5
   ```

2. Po spuštění všechny procesy čekají na připojení debuggeru ve smyčce `waitForDebugStart()`. Nyní je nutné připojit stejné množství instancí debuggeru k procesům (jejich PID lze zjistit např. pomocí `ps`).

3. Nastavte breakpointy ve všech debuggerech na řádek se smyčkou `while`. Vykonávání programu by se mělo zastavit a čekat na vaši interakci.

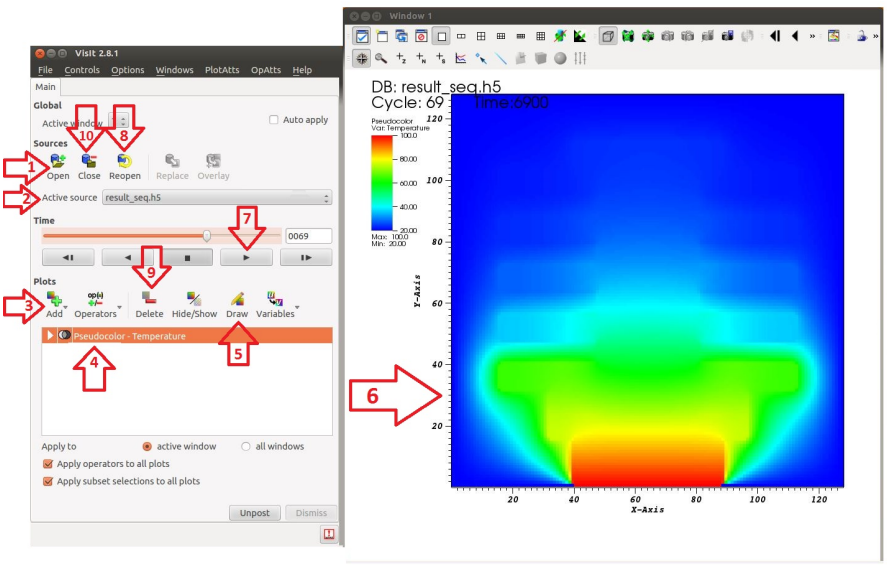
4. Abyste mohli pokračovat ve výpočtu, je třeba ve všech debuggerech manuálně přepsat hodnotu proměnné `debugStarted` na `true`. Nyní již program může pokračovat ve výpočtu a vy máte možnost sledovat jeho chování pro každý proces zvlášť.

6 Testování a profilování

Pro základní testování funkčnosti vaší paralelní implementace lze využít vestavěný verifikační mód. Pro tento účel lze použít parametr `-v`, který porovná výsledné rozložení tepla proti sekvenční simulaci (v kombinaci s parametrem `-r <PREFIX>` uloží normalizovaný rozdíl do obrázku jako `<PREFIX>_abs_diff.png`). Parametr `-d` umožňuje vypsat hodnoty teplot v celé doméně v textové formě, nebo v kombinaci s `-r` opět uložit do obrázku.
Rozdíl mezi výsledky paralelní a sekvenční verzí by měl pohybovat v řádu 10^{-3} (úplnou shodu není možné očekávat jelikož výpočty ve `float` nejsou asociativní)

Situace je mírně komplikovanější při ověřování funkce paralelního zápisu do souboru. V tomto případě doporučujeme nástroj `VisIt`[6], který umožňuje vizualizaci uložených dat včetně časového průběhu. Za předpokladu, že výsledek simulace je uložen v souboru `result_par.h5` lze jeho obsah zobrazit nástrojem `VisIt` následujícím způsobem.

Z domovské stránky[7] si stáhněte nástroj `VisIt`. Po rozbalení spusťte spustitelný soubor s názvem `visit`. Otevře se hlavní okno aplikace (obrázek **??**, čísla v závorkách v následujícím popisu odpovídají číslům v šipkách na obrázku). Klikněte na tlačítko `Open` (1) v sekci `sources` a vyberte `result_par.h5`. Název souboru by se měl objevit v `Active source` (2). Nyní můžeme začít vizualizovat. Klikněte na tlačítko `Add` (3), vyberte `Pseudocolor` a následně `Temperature`. V bílém okénku se objeví vybraný filtr (4). Nyní klikněte na tlačítko `Draw` (5) a v novém okně (6) se vykreslí první časový krok vizualizace, kdy ve spodní části je krátký proužek s teplotou $100°C$ a ve zbytku je teplota $20°C$ (na obrázku je zobrazený 69. časový krok). Celou simulaci lze spustit v sekci `Time` kliknutím na tlačítko `Play` (7), popř. ručně přecházet mezi jednotlivými časovými kroky pomocí táhla. Pro znovuotevření souboru (např. po novém běhu simulace) slouží tlačítko `Reopen` (8). To ale naneštěstí ne vždy funguje, takže je většinou nutné smazat všechny filtry tlačítkem `Delete` (9), soubor zavřít tlačítkem `Close` (10) a znovu otevřít.



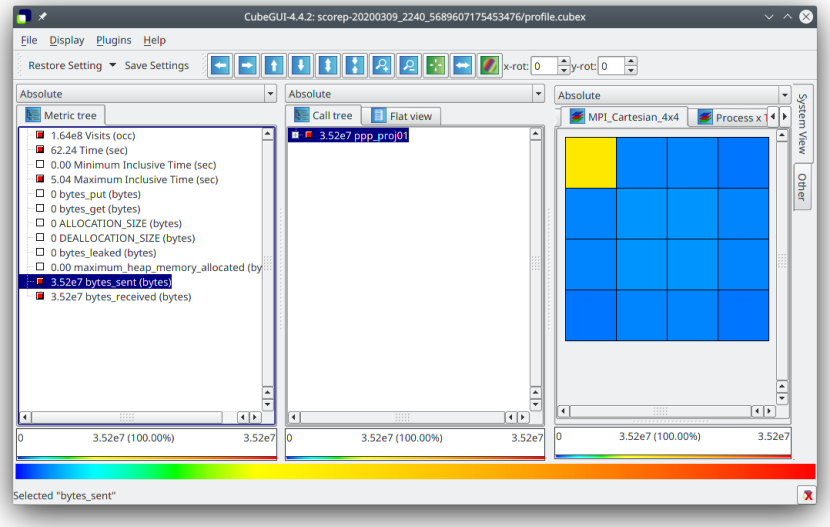
Obrázek 3: VisIt - Popis grafického uživatelského rozhraní.

[6] https://wci.llnl.gov/simulation/computer-codes/visit
[7] https://wci.llnl.gov/simulation/computer-codes/visit/executables

6.1 Zobrazení Score-P dat

Profilovací a trasovací data sesbíraná např. pomocí skriptů popsaných v sekci **??** je možné pomocí nástrojů `Cube`[8] a `Vampir`[9], které jsou dostupné na Barboře pomocí modulů `Score-P/8.0-iimpi-2021b` a `Vampir`. Nejsnadnější možností pro použití těchto nástrojů je vzdálená plocha VNC, nebo X forwarding (`ssh -XC ...`) z login uzlů Salomonu.

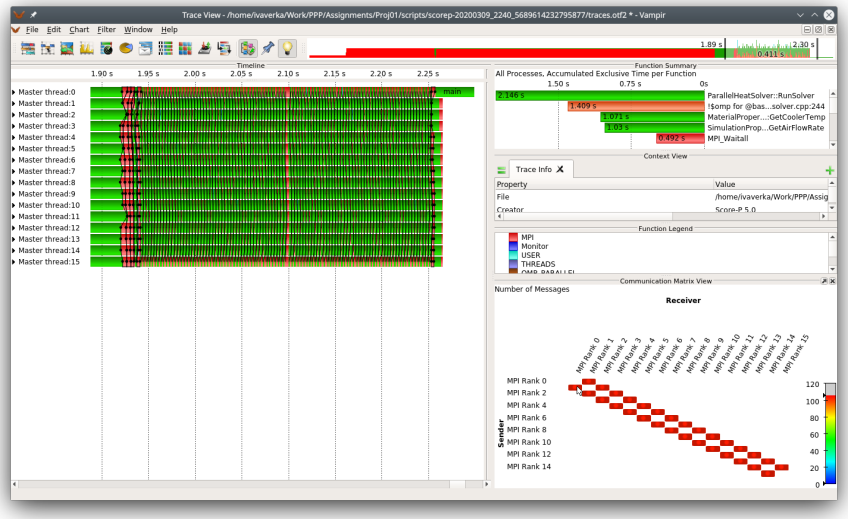
Nástroj `Cube` umožňuje zobrazit profilovací data uložená `Score-P` v souborech `.../profile.cubex`. Obrázek **??** ukazuje profil 2D dekompozice simulačního kódu. V levém sloupci jsou měřené veličiny (jako velikost odeslaných/přijatých dat pomocí MPI), v prostředním sloupci je možné procházet strom volání (`call-tree`) programu a konečně v levém sloupci je vizualizace použitého kartézkého komunikátoru a velikost odeslaných dat jako "heat map".



Obrázek 4: Cube - Profil 2D dekompozice na 16 procesech.

`Vampir` je určený především k vizualizaci trasovacích dat, tedy událostí v čase. Pro získání těchto dat pomocí `Score-P` je nutné povolit jejich záznam proměnnou prostředí `SCOREP_ENABLE_PROFILING=true`. Důležitým krokem je také omezení množství zaznamenaných dat pomocí filtrování (viz sekce **??** a také skript `run_scorep_profile.sh`).

[8] https://www.scalasca.org/software/cube-4.x/documentation.html
[9] https://vampir.eu/



Obrázek 5: Vampir - Trace 1D dekompozice na 16 procesech.

Obrázek **??** ukazuje hlavní okno nástroje `Vampir` s otevřeným trace souborem `.../traces.otf2`, který je záznamem běhu 1D dekopozice. Hlavní, levá část okna ukazuje časový průběh událostí všech MPI procesů a jejich vláken (vlákna jsou zde skryta) a komunikaci mezi nimi. Záložka "Function Summary" zobrazuje čas strávený v jednotlivých voláních uvnitř právě zobrazované oblasti časové osy a "Communication Matrix View" ukazuje komunikační matici.

7 Odevzdání a bodování

Odevzdat je třeba soubory **ParallelHeatSolver.*** obsahující vaši implementaci a soubor **xloginNN.pdf** obsahující vyhodnocení vaší implementace a grafy škálování. Tyto soubory zabalte pomocí zip (nebo pomocí targetu `pack`), pojmenujte vaším loginem a odevzdejte do informačního systému. Za úplné vypracování projektu je možné získat až 25 bodů:

- **2 body** - Rozdělení domény mezi procesy a sesbírání před každým zápisem na disk.
  Zde je bezpodmínečně nutné použít kolektivní komunikace MPI!
- **3 body** - Základní verze bez překrytí komunikace, správné předávání Halo zón, správné výsledky a výpočet teploty ve středním sloupci.
- **3 body** - Překrytí komunikace a výpočtu.
- **3 body** - Korektní implementace módu vzdáleného přístupu do paměti.
- **2 body** - Efektivita řešení.
- **4 bodů** - Paralelní ukládání průběhu pomocí Parallel HDF5.
- **2 bodů** - Vhodné nastavení paralelního I/O.
- **6 bodů** - Dokumentace obsahující grafy silného a slabého škálování, porovnání 1D a 2D dekompozic a vysvětlení naměřených výsledků. Vhodné je také popsat zjištění získané pomocí nástroje Score-P, Cube a případně Vampire.

Součástí hodnocení je použití vhodných technik MPI pro řešení daného problému a vysvětlení získaných výsledků v dokumentaci!