

# Praktické paralelní programování

## Projekt č. 1 - MPI a paralelní I/O

Kristian Kadlubiak, [ikadlubiak@fit.vutbr.cz](mailto:ikadlubiak@fit.vutbr.cz)  
Jiří Jaroš, [jarosjr@fit.vutbr.cz](mailto:jarosjr@fit.vutbr.cz)

**Termín odevzdání:** 1. května 2022 23:59:59  
**Hodnocení:** až 25 bodů

## 1 Úvod

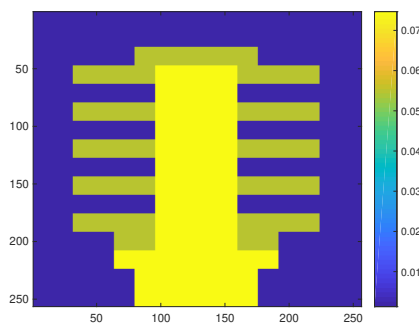
Cílem projektu je osvojit si základní principy programování se zasíláním zpráv s využitím knihovny MPI, použití knihovny HDF5 pro paralelní I/O a vyhodnocení chování implementace pomocí nástrojů založených na Score-P. Součástí projektu je také ověření škálování na několika uzlech superpočítače Barbora.

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.

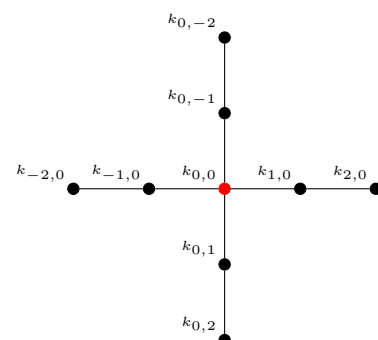
Průběh simulace je ukládán v celé doméně pomocí knihovny HDF5 do souboru pro pozdější analýzu. Zaznamenány jsou však výsledky pouze omezené množiny časových kroků simulace, tak aby byla velikost zaznamenaných dat omezená.

### 1.1 Numerická metoda řešení

Pro řešení problému byla zvolena varianta metody konečných diferencí (FDTD<sup>1</sup>). 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 ( $\alpha$ ) materiálu v daném bodě (měď, hliník nebo vzduch). Pro oblasti, které jsou tvořeny vzduchem dále definujeme parametr  $\beta$ , 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<sup>2</sup>(b) používaný v simulaci.

<sup>1</sup><http://www.eecs.wsu.edu/~schneidj/ufdtd/ufdtd.pdf>

Samotný výpočet každého z  $N_t$  časových kroků spočívá v aktualizaci každého bodu domény na základě jeho 4+4-okolí (viz obrázek 1). 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-okolích. Rozšíříme-li simulaci o heterogenní materiál a odvod tepla lze aktualizaci v každém kroce zapsat rovnicí jako:

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

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 1). 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 jak základ pro implementaci samotného simulátoru, tak kód generátoru vstupních souborů pro simulaci. Výstupem generátoru je soubor v HDF5 formátu, který obsahuje specifikaci simulační domény a je použit jako vstup simulátoru.

Tato složka také obsahuje soubory **parallel\_heat\_solver.\***, které jsou připraveny pro implementaci třídy **ParallelHeatSolver**, kterou je třeba vytvořit a odevzdat.

- Složka **scripts** obsahuje skript pro vygenerování testovacích dat různé velikosti a sadu PBS 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 **base.h** obsahuje deklarace pomocných tříd **AutoHandle** a **AlignedAllocator**. První jmenovaná umožňuje automatické uvolnění objektů knihoven jako HDF5 pomocí **RAII**<sup>3</sup>. Druhá je pak implementací C++ konceptu<sup>4</sup>, který alokuje paměť zarovnanou na 64 bytů a lze ji použít například pro zarovnání paměti objektů jako **std::vector**.
- Soubory **material\_properties.\*** obsahují implementaci třídy reprezentující informace o materiálu načteného ze vstupního souboru, které následně zpřístupní zbytku aplikace.
- Soubory **simulation\_properties.\*** implementují třídu **SimulationProperties**, která slouží pro parsování argumentů předaných aplikaci na příkazové řádce.
- Soubory **base\_heat\_solver.\*** obsahují implementaci základní třídy solveru (**BaseHeatSolver**), která zajišťuje jeho základní funkci a slouží jako bazová třída pro jeho různé implementace. Nejdůležitější částí třídy je metoda **ComputePoint(...)**, která implementuje rovnici 1 ve specifikovaném bodě a metoda **RunSolver(...)**, kterou musí implementovat každý solver. Dostupná

<sup>2</sup>[https://en.wikipedia.org/wiki/Stencil\\_code](https://en.wikipedia.org/wiki/Stencil_code)

<sup>3</sup>[https://en.wikipedia.org/wiki/Resource\\_acquisition\\_is\\_initialization](https://en.wikipedia.org/wiki/Resource_acquisition_is_initialization)

<sup>4</sup>[https://en.cppreference.com/w/cpp/named\\_req/Allocator](https://en.cppreference.com/w/cpp/named_req/Allocator)

je také metoda `UpdateTile(...)`, která aplikuje `ComputePoint(...)` na dlaždici paralelně pomocí prostředků `OpenMP`<sup>5</sup> (vlákna i vektorizace).

- Soubory `sequential_heat_solver.*` obsahují referenční (sekvenční) implementaci solveru. Tyto soubory také ukazují, jak pracovat se zarovnaným alokátozem a automaticky uvolňovaným “handle” pro HDF5 objekty (`AutoHandle`).

## 2.2 Sekvenční solver

Sekvenční verze algoritmu je implementovaná ve třídě `SequentialHeatSolver`, kterou naleznete v souborech `sequential_heat_solver.*` (některé znovupoužitelné části lze také nalézt v její bázevé třídě `BaseHeatSolver`). 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 `m.materialProperties.GetInitTemp()`.
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 `base_heat_solver.h` a `base_heat_solver.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 odchylkou na úrovni numerické chyby *float*).
6. Aktuální iteraci zapíšeme do souboru. Abychom omezili množství zapisovaných dat, ukládáme pouze ty iterace, jejichž index je dělitelný parametrem `GetDiskWriteIntensity()`, tedy pokud má `GetDiskWriteIntensity()` hodnotu 10, uloží se každá desátá iterace.
7. Prohodíme ukazatele polí `workTempArrays[0]` a `workTempArrays[1]`. Tím budeme v následující iteraci považova právě vypočtený výsledek jako výchozí stav.

---

<sup>5</sup><https://www.openmp.org>

8. Pomocí metody `PrintProgressReport(...)` vypíšeme progress simulace. Tato metoda vypisuje progress 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 `m_tempArray` a je tedy nutné je zkopírovat do výstupního pole.

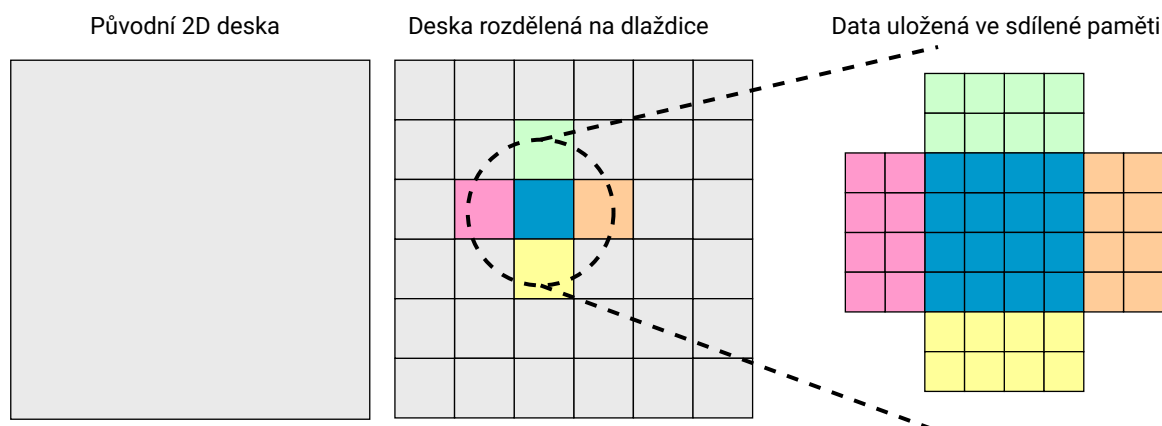
### 3 Co je úkolem studenta?

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 `parallel_heat_solver.h` a `parallel_heat_solver.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:

- `parallel_heat_solver.h` - kód implementace
- `parallel_heat_solver.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 2).



Obrázek 2: Příklad 2D dekompozice domény. 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. Pokud je specifikován název výstupního souboru (`m.simulationProperties.GetOutputFileName()` není prázdné) vytvoříme nový výstupní soubor s názvem doplněným o koncovku `par` (tento název lze získat voláním stejné metody s argumentem `"par"`). V opačném případě se průběh simulace neukládá. **V případě sekvenčního I/O bude soubor vytvářet pouze jeden (nultý) proces, naopak při použití paralelního I/O musí soubor vytvářet všechny procesy!**
2. Doménu o straně  $N = 2^k$  je nutné rozdělit mezi  $P = 2^p$  procesů, kde  $k, p \in \{0, 1, \dots, n\}$  a  $k > p$ . Požadovanou dekompozici lze zjistit pomocí `m.simulationProperties.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  $256 \times 256$  budou procesy uspořádány v mřížce  $4 \times 4$  a každý bude mít svoji lokální  $64 \times 64$  dlaždici). V případě liché  $p$  je nutné doménu rozdělit do obdélníků jako:  $n_x = \sqrt{P/2}$ ,  $n_y = 2n_x$  (např. pro 32 procesů a  $256 \times 256$  vznikne mřížka  $4 \times 8$  a dlaždice  $64 \times 32$ ).
3. Každý proces si vytvoří pomocná pole pro svou dlaždici a její okolí:
  - 2 pole pro data (opět je nutné zajistit, aby výpočet aktuální iterace vycházel pouze z hodnot té předchozí)
  - 1 pole pro materiál
  - 1 pole pro teplotní vlastnosti materiálu
4. Počáteční hodnoty pro tato pole má k dispozici pouze nultý proces v polích `GetInitTemp()`, `GetDomainParams()` a `GetDomainMap()` objektu `m.materialProperties`. Je tedy třeba tyto data (včetně překrývajících se oblastí) rozdistribuovat příslušným procesům.
5. Uložíme si aktuální čas pro pozdější výpočet celkové doby běhu simulace (stačí, aby toto provedl nultý proces).
6. Spustíme samotnou simulaci s `m.simulationProperties.GetNumIterations()` kroky.
7. V každé iteraci se projdou všechny body dlaždice (kromě okrajů domény, ty zůstávají pevné) a jejich teploty se aktualizují.
  - (a) Výpočet probíhá analogicky k sekvenční verzi.
  - (b) Pro dosažení co nejvyššího zrychlení je nutné nejprve spočítat okrajové body dlaždice, tyto odeslat okolním procesům, následně spočítat vnitřní body. Tím dojde k překrytí výpočtu vnitřních bodů a komunikace. Implementace musí podporovat oboustrannou komunikaci i vzdálený přístup do paměti (módy 1 a 2). **Protože počítáme s (4+4)-okolím musí mít okraje dlaždice odpovídající šířku (tedy 2 body).**
  - (c) Zápis do souboru provádějte pomocí nultého uzlu (v případě sekvenčního I/O), nebo pomocí všech uzlů (v případě paralelního I/O). Požadovaný typ zápisu lze zjistit voláním `m.simulationProperties.IsUseParallelIO()`.
  - (d) Výpočet průměrné teploty ve středím pásu domény a tisk informací o průběhu simulace pomocí nultého procesu a voláním metody `PrintProgressReport(...)`. Iterace ve kterých je třeba provést výpočet průměrné teploty lze detekovat voláním `ShouldPrintProgress(...)` s číslem aktuální iterace. Je také vhodné si uvědomit, že tento výpočet musí provádět pouze dlaždice v pásu procházejícím středem domény.
8. Tisk výsledného času simulace, průměrné teploty (nezapomeňte přepočítat po poslední iteraci) atd. je opět možné vypsát voláním `PrintFinalReport(...)` pomocí jednoho z procesů.
9. Výsledné teploty z jednotlivých dlaždic je nutné shromáždit v nultém procesu a vrátit pomocí parametru `outResult` (pozor, tento parametr obsahuje prázdný vektor pro všechny krom nultého procesu).

### 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

- Nejprve 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. nejprve 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 výpočtu 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.
- Pro uložení dat do souboru sekvenční metodou můžete použít metodu `StoreDataIntoFile`, které je třeba předat HDF5 soubor (handle), číslo iterace a ukazatel na samotná data (tuto metodu musí volat pouze jediný proces, který má soubor otevřený). Implementovat paralelní metodu ukládání dat pomocí HDF5 je vaším úkolem.
- Celou implementaci je vhodné rozdělit do logických celků jako: otevření výstupního souboru, výpočet rozdělení domény do dlaždic, vytvoření MPI datových typů a výpočet průměrné teploty ve středním sloupci a ty následně implementovat jako různé metody třídy, kterou tvoříte.

Pozn.: 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 `*.pbs` souborech ve složce `scripts`. Zpravidla stačí předat správný počet procesů PBS systému

```
qsub ... -l select=1:ncpus=36:mpiprocs=32 ...
```

a program spustit jako

```
mpirun ./ppp-proj01 <parametry>
```

## 4 Překlad a spuštění projektu

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

- C++ překladač s podporou C++11
- CMake - <https://cmake.org/cmake/help/latest/>
- MPI (Intel MPI nebo Open MPI) - <https://www.open-mpi.org/doc/current/>
- Parallel HDF5 - <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 dostupné lze rozbalený projekt přeložit pomocí CMake a Make.

```
mkdir build && cd build
cmake ../sources
make
```

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 pouze pro překlad, ale ne při konfiguraci projektu pomocí CMake. Toho lze dosáhnout následujícím postupem:

```
mkdir build_prof && cd build_prof
I_MPI_CC=icc I_MPI_CXX=icpc \
SCOREP_WRAPPER=off CC=scorep-mpicc CXX=scorep-mpicxx cmake ../sources
I_MPI_CC=icc I_MPI_CXX=icpc \
SCOREP_WRAPPER_INSTRUMENTER_FLAGS=---thread=omp make
```

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>  

```
./configure --prefix=CESTA
make install
```
- HDF5 - <https://www.hdfgroup.org/downloads/hdf5/source-code>  

```
./configure --enable-parallel --prefix=CESTA
make install
```

Spuštění projektu na lokálním PC je pak snadné pomocí: `mpirun -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 překlada a vyhodnocení výsledků. Pro překlad doporučujeme moduly Intel 2020b, které je možné načíst pomocí:

```
ml CMake intel/2020b HDF5/1.10.6-intel-2020b-parallel
```

Je možné požádat o přidělení uzlu v interaktivním módu pro testování projektu příkazem:

```
qsub -q qprod -A DD-21-22 -l select=1:ncpus=36:mpiprocs=32:omphreads=1 -I
```

### 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/2020b` 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.pbs`, 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.pbs` a `run_full_hybrid_2d.pbs` 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 dekompozici).

### 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 patričné rozdělení mezi několik "object storage targets" (OSTs), to lze provést příkazem:

```
lfs setstripe -S 1M -c 16 /scratch/project/dd-21-22/$USER
```

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` a `intel/2020b` ve složce `build_prof` můžete použít skript `run_scorep_profile.pbs` 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
    BaseHeatSolver::ComputePoint
    *std::*
SCOREP_REGION_NAMES_END
```

Tím je z trasování vyloučena metoda `BaseHeatSolver::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 - vypsání výsledných teplot v doméně na stdout
- v - verification mode - porovná výsledky sekvenční a paralelní verze a vypíše OK nebo FAILED
- b - batch mode - informace o běhu jsou na stdout vypisovány v CSV formátu
- h - batch mode - navíc vypíše hlavičku CSV
- 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 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)**

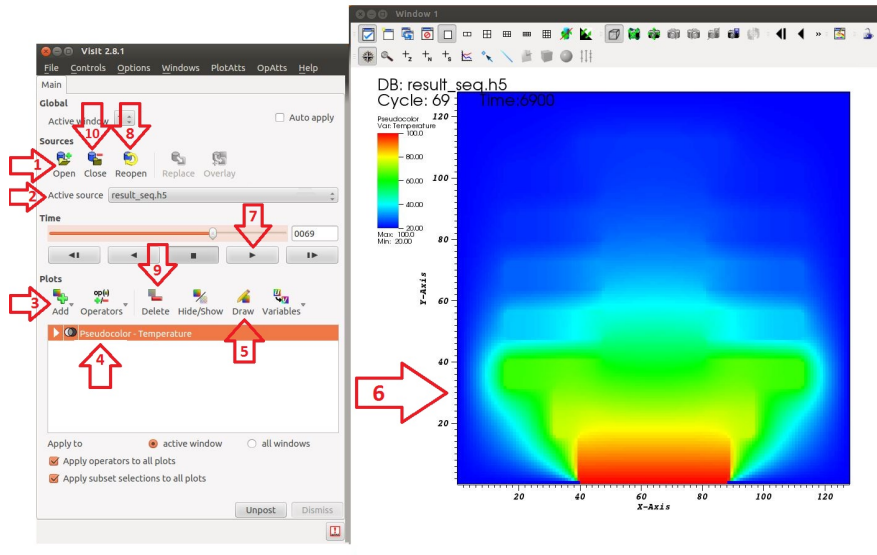
Situace je mírně komplikovanější při ověřování funkce paralelního zápisu do souboru. V tomto případě doporučujeme nástroj VisIt<sup>6</sup>, který umožňuje vizualizaci uložených dat včetně časového průběhu. Za předpokladu, že výsledek simulace je uložen v souboru `result_par.h5` lze jeho obsah zobrazit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

Z domovské stránky<sup>7</sup> si stáhněte nástroj VisIt. Po rozbalení spusťte spustitelný soubor s názvem `visit`. Otevře se hlavní okno aplikace (obrázek 3, čísla v závorkách v následujícím popisu odpovídají číslům v šípkách 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

<sup>6</sup><https://wci.llnl.gov/simulation/computer-codes/visit>

<sup>7</sup><https://wci.llnl.gov/simulation/computer-codes/visit/executables>

ve spodní části je krátký proužek s teplotou  $100^{\circ}\text{C}$  a ve zbytku je teplota  $20^{\circ}\text{C}$  (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.

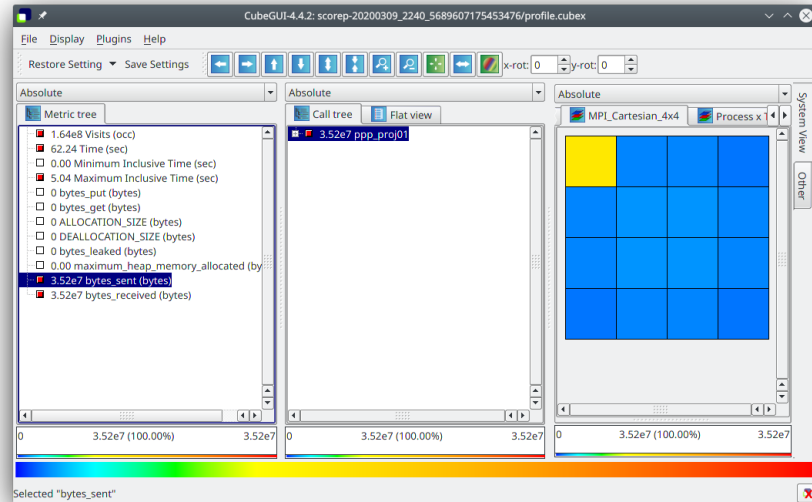
## 5.1 Zobrazení Score-P dat

Profilovací a trasovací data sesbírání např. pomocí skriptů popsanych v sekci 4.2 je možné pomocí nástrojů **Cube**<sup>8</sup> a **Vampir**<sup>9</sup>, které jsou dostupné na Salomonu pomocí modulů **Score-P/6.0-intel-2020a** 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 4 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 kartézského komunikátoru a velikost odeslaných dat jako “heat map”.

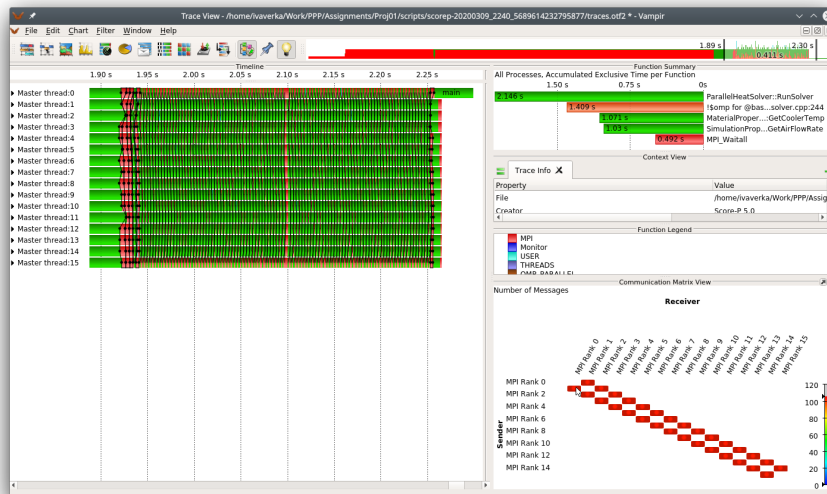
<sup>8</sup><https://www.scalasca.org/software/cube-4.x/documentation.html>

<sup>9</sup><https://vampir.eu/>



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 4.2 a také skript `run_scorep_profile.pbs`).



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

Obrázek 5 ukazuje hlavní okno nástroje Vampir s otevřeným trace souborem `.../traces.otf2`, který je záznamem běhu 1D dekompozice. 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.

## 6 Odevzdání a bodování

Odevzdat je třeba soubory `parallel_heat_solver.*` obsahující vaši implementaci a soubor `xloginNN.pdf` obsahující vyhodnocení vaší implementace a grafy škálování. Tyto soubory zabalte pomocí zip, pojme-

nujte vaším loginem a odevzdejte do WISu. Za úplné vypracování projektu je možné získat až 20 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!**