PPP – Projekt: Simulace ohřevu chladiče procesoru

Byly implementovány všechny části projektu dle zadání. Tato dokumentace obsahuje stručný popis implementace a testování. Následně jsou popsány výsledky profilování.

Implementace

Třída `ParallelHeatSolver` obsahuje v konstruktoru veškerou inicializaci potřebnou pro další výpočet. Samotná inicializace je rozdělená do několika dalších metod, které se zabývají inicializací jednotlivých logických částí.

Je získána požadované dekompozice, vypočteny rozměry dlaždic a rozměry rozšířené dlaždice (je používána dlaždice rozšířená o okraje velikosti 2 na všech stranách), pro které je alokována paměť. Jsou vypočítány indexy, od kterých se mají počítat nové hodnoty v dlaždici (kvůli přidaným okrajům a okrajům domény, které zůstávají fixní). Dále jsou vypočteny indexy sousedů, je vytvořen komunikátor a vypočten index pro výpočet teploty prostředního sloupce.

Jsou vytvořeny všechny potřebné datové typy. Pomocí *subarray* je vytvořen datový typ pro dlaždice (bez okrajů, pouze část, kterou přímo počítá daný proces). Typ dlaždice si vytvoří každý proces, proces *root* si navíc vytvoří datový typ dlaždice v rámci celé matice, zde je navíc provedena úprava pomocí *resize*, aby bylo možné odesílat jednotlivé dlaždice. Pro výměnu okrajů jsou vytvořeny pomocí *vector* dva datové typy reprezentující levý/pravý a horní/dolní okraj matice. Všechny datové typy jsou vytvořeny dvakrát, pro datové typy `float` a `int`.

Pokud je to zadán, tak je otevřen soubor pro zápis pomocí knihovny `HDF5`. Pro paralelní zápis je provedená celá inicializace včetně zvolení korektních pozic dat pro zápis pomocí *hyperslab*. Pokud je zvolena výměna okrajů pomocí vzdáleného přístupu do paměti tak jsou vytvořena dvě `RMA` okna.

Na konci inicializace dojde ve dvou krocích k rozeslání dat z procesu *root*. Jsou rozeslány jednotlivé dlaždice pomocí funkce `MPI_Scatterv`. Poté si již jednotlivé procesy vymění okraje se svými sousedy pomocí neblokujících oboustranných komunikací. Při této inicializační výměně se posílají hodnoty teplot v obou polích a také hodnoty vlastností materiálu.

V metodě `RunSolver` se na začátku každé iterace simulace vypočtou hodnoty na okrajích, kde má proces sousedy. Poté se zahájí výměna, buďto pomocí neblokujících zaslání a přijetí, nebo pomocí vzdáleného přístupu do paměti sousedů pomocí `MPI_Put`, podle zvoleného módu. Poté se dopočítá střed dlaždice a počká se na skončení komunikace pomocí `MPI_Waitall` nebo `MPI_Win_fence`. Vzhledem k použití dvou polí pro výpočet hodnot teploty, tak je nutné pro vzdálený přístup do paměti použít dvě okna, která se prohazují stejně jako použitá pole hodnot, a uzavření okna se vždy provádí jen pro právě využívané okno.

Testování

Správnost výpočtu byla kontrolována porovnáním se sekvenční verzí pomocí parametrů `-v` a `-d`. Podobně výpočet hodnot v prostředním sloupci byl testován porovnáním hodnot se sekvenční verzi. Pro ověření správnosti ukládání hodnot do souboru byla vypnuta vektorizace v metodě `UpdateTile` a vytvořené soubory byly porovnány se souborem vytvořeným sekvenční verzí pomocí nástroje `h5diff`.

Profilování

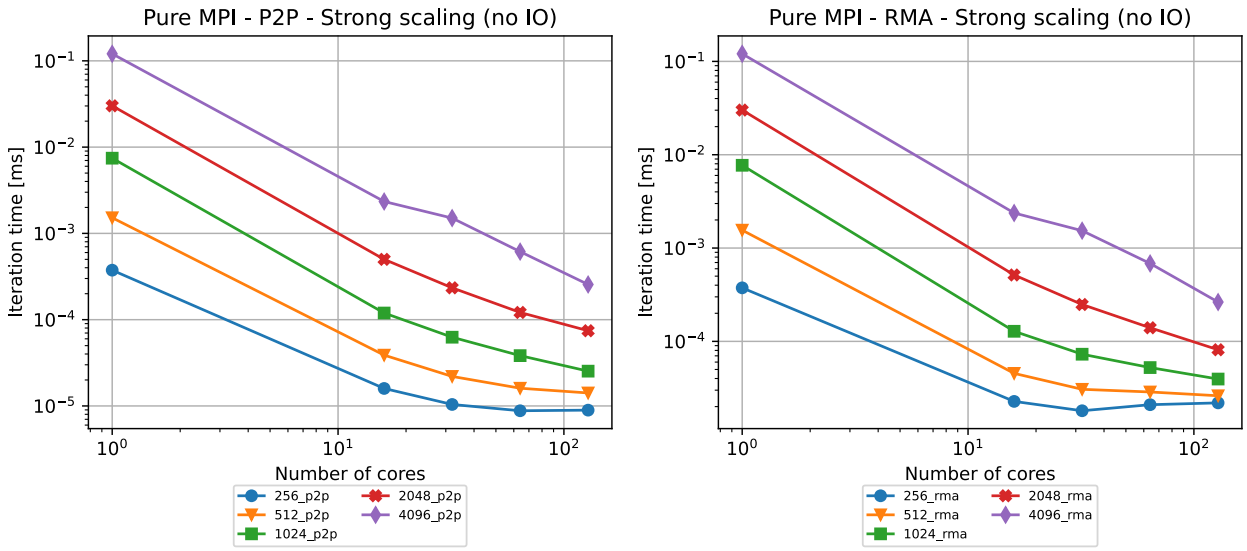
Pro profilování byly použity skripty ve složce `scripts`, grafy byly vytvořeny upravením skriptu `generate_plots.py`.

Efektivita výpočtu

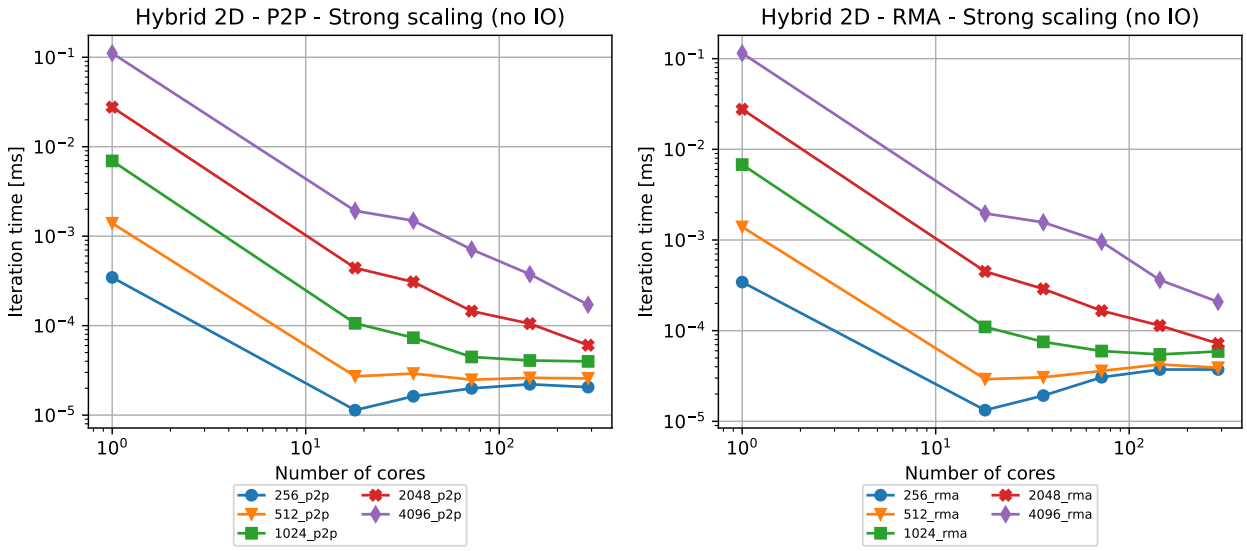
Z grafů silného škálování 1, 2 je vidět, že řešení škáluje dobře a zvyšuje se rychlost výpočtu se zvyšujícím se počtem jader. Pro menší velikosti matice už více jader nepřináší výrazné zrychlení, což je dáno nedostatečnou velikostí dlaždic, aby přidaný výkon převážil nad přidanou režií a komunikací.

V grafech je dále vidět, že výpočet pomocí čistého MPI 1 i výpočet s použitím vláken při použití 2D dekompozice 2 vykazuje podobné výsledky. Podobné jsou i výsledky pro 1D dekompozici 3 při použití výměny okrajů pomocí oboustranné komunikace. Pro největší velikosti matice je 2D dekompozice viditelně efektivnější a škáluje lépe, což je zřejmě dáno tím, že pro tyto rozměry překrytí menších komunikací se 4 sousedy funguje lépe, než výměna větších bloků dat jen se dvěma sousedy. Pro menší rozměry matice by teoreticky měla naopak být efektivnější 1D dekompozice, jelikož zprávy pro 2D dekompozici by byly příliš malé a přidávaly by zbytečnou režii, podle grafů je ale rozdíl zanedbatelný.

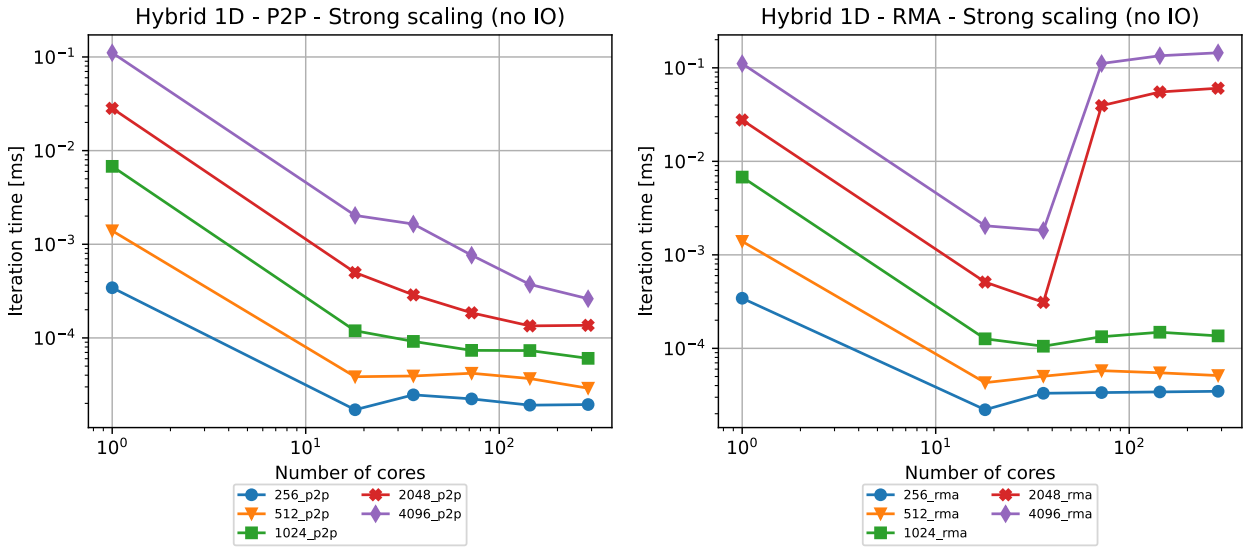
Výměna okrajů pomocí vzdáleného přístupu do paměti vykazuje také podobnou efektivitu jako ostatní způsoby, i když zrychlení je mírně nižší. Výjimkou je 1D dekompozice 3 pro největší velikosti matice při použití 72 a více jader. V tomto případě dojde k výraznému zpomalení výpočtu a výpočet je neefektivní. Je možné, že tento problém je způsoben příliš dlouhým čekáním na zápis velkého množství dat do paměti sousedů, ale je zvláštní, že se tento problém takto výrazně neprojevuje u 2D dekompozice a ani u standardního zasílání zpráv.



Obrázek 1: Grafy silného škálování pro čisté MPI s 2D dekompozicí. Pro výměnu okrajů je použita oboustranná komunikace nebo vzdálený přístup do paměti.



Obrázek 2: Grafy silného škálování pro hybridní MPI využívající vlákna s 2D dekompozicí. Pro výměnu okrajů je použita oboustranná komunikace nebo vzdálený přístup do paměti.



Obrázek 3: Grafy silného škálování pro hybridní MPI využívající vlákna s 1D dekompozicí. Pro výměnu okrajů je použita oboustranná komunikace nebo vzdálený přístup do paměti.

Ukládání do souboru

Vyhodnocení efektivity ukládání do souboru je možné z grafů silného škálování pro výpočet s ukládáním do souboru pomocí sekvenčního nebo paralelního `IO` 4. Škálování na všech grafech je podobné bez ohledu na použití vláken a použitou dekompozici. Pro menší velikosti matice je sekvenční `IO` výrazně efektivnější. Pro ostatní velikosti matice je většinou sekvenční `IO` stále efektivnější než paralelní. Pouze pro největší velikost matice začíná být paralelní `IO` stejně nebo mírně rychlejší.

Je zřejmé, že pro menší velikosti matice se nevyplatí použití paralelního `IO` kvůli vyšší režii. Poslat menší množství dat procesu *root* a sekvenční uložení je méně časově náročné. Se zvětšující se velikosti matice a tím pádem i dlaždice a posílaných zpráv začíná být efektivnější ukládání paralelně na každém procesu. Paralelní `IO` by také možná mohlo být zefektivněno další konfigurací, např. nastavením *chunk size*.

Cube

Analýza množství komunikace jednotlivých procesů byla provedena pomocí programu Cube. Na prvním obrázku 5 je vidět počet odeslaných zpráv v rámci hlavní smyčky samotného výpočtu simulace pro 1D a 2D dekompozici matice. Pro 1D dekompozici je vidět, že první a poslední proces (tj. procesy na obou okrajích matice) odesílá jen polovinu zpráv oproti vnitřním procesům. U 2D dekompozice odesílají nejméně zpráv 4 procesy počítající rohy matice, navíc přibyl rozdíl mezi okraji matice a středem matice.

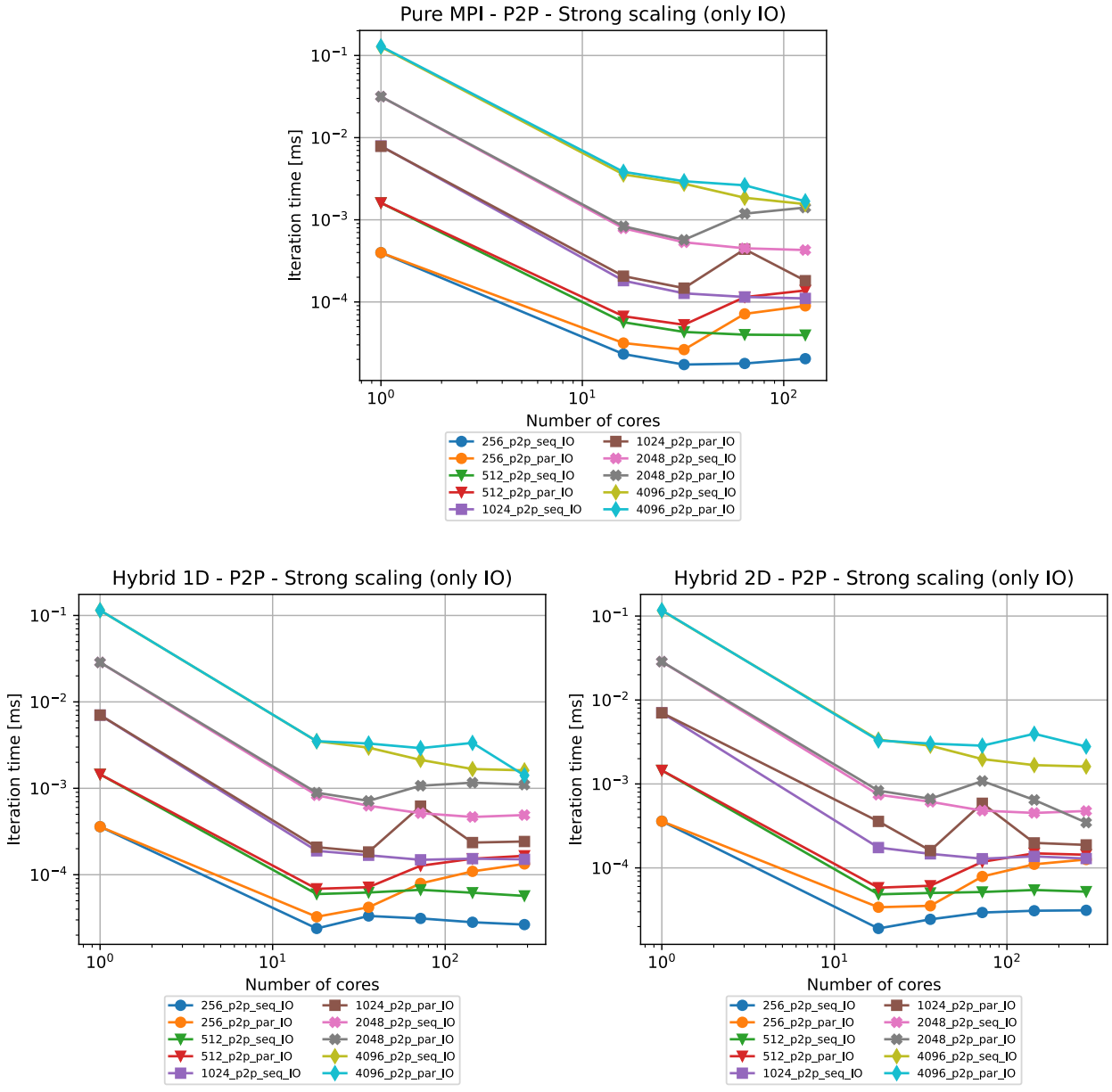
Druhý obrázek 6 ukazuje počet odeslaných bytů v rámci celého výpočtu včetně inicializace pro 1D dekompozici. Zde je opět vidět, že poslední krajní proces odeslal méně dat, ale také, že první proces (*root*) odeslal výrazně více dat než ostatní procesy.

Menší počet odeslaných dat u procesů počítajících rohy a okraje matice je daný způsobem rozdělení matice mezi procesy, jelikož je rozdělení rovnoměrné, tak procesy, které mají méně sousedů, provádějí méně komunikace. Velký počet dat odeslaný prvním procesem je daný počátečním rozesíláním dlaždic. Komunikační zátěž mezi jednotlivými procesy tedy není vyrovnaná.

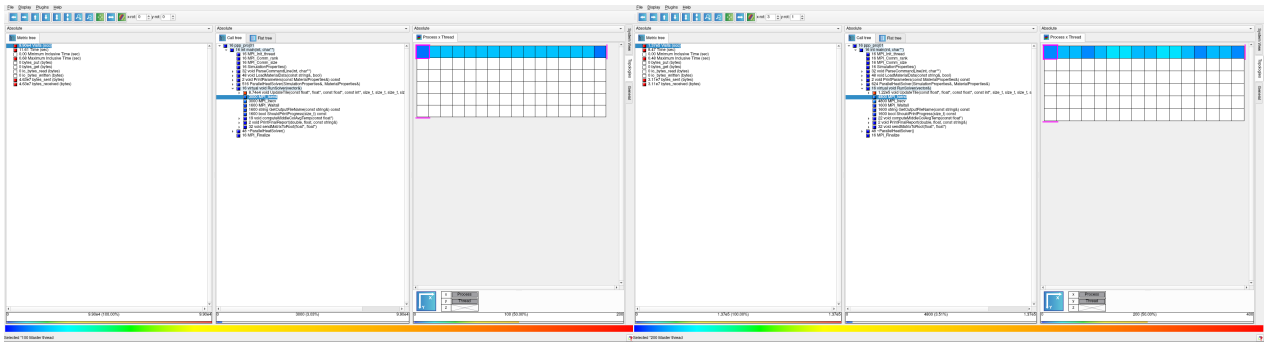
Vampir

Pro vyhodnocení efektivity výpočtu a překrytí komunikace výpočtem byl použit program Vampir. Výsledky jsou ukázány na běhu programu s 16 procesy a 2 vlákny se vstupní matici o déle strany 2048 za použití 1D dekompozice.

Přehled je vidět na obrázku 7. Celkový běh programu trval necelé 4 sekundy a relativně významná část je zabraná inicializací vláken. Ve zbylé části je naprostá většina zabraná užitečným výpočtem (`OMP_LOOP`). Na obrázku 8 je vidět překrytí komunikace výpočtem.

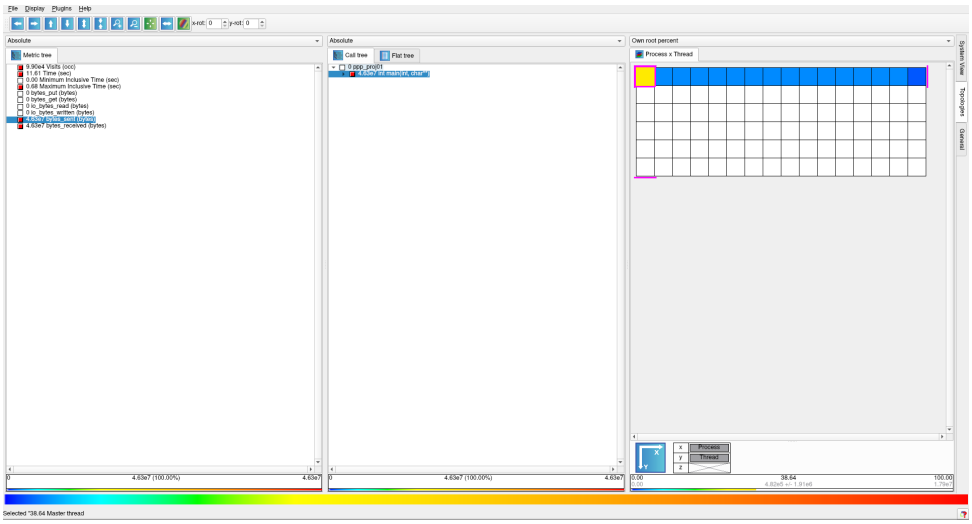


Obrázek 4: Grafy silného škálování pro výpočet se zápisem do souboru sekvenčním nebo paralelním způsobem při použití čistého MPI s 2D dekompozicí nebo hybridního MPI s 2D nebo 1D dekompozicí (s výměnou okrajů pouze pomocí oboustranných komunikací).

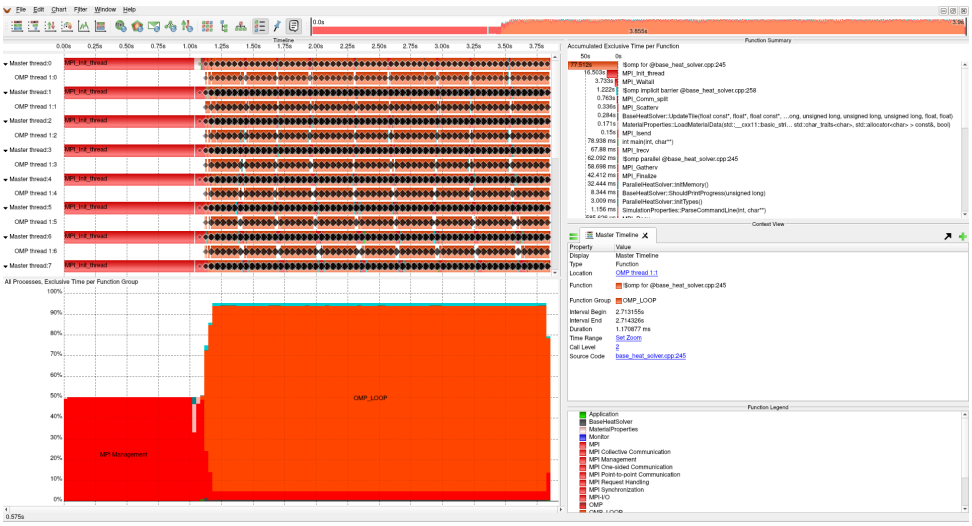


(a) 1D dekompozice (b) 2D dekompozice

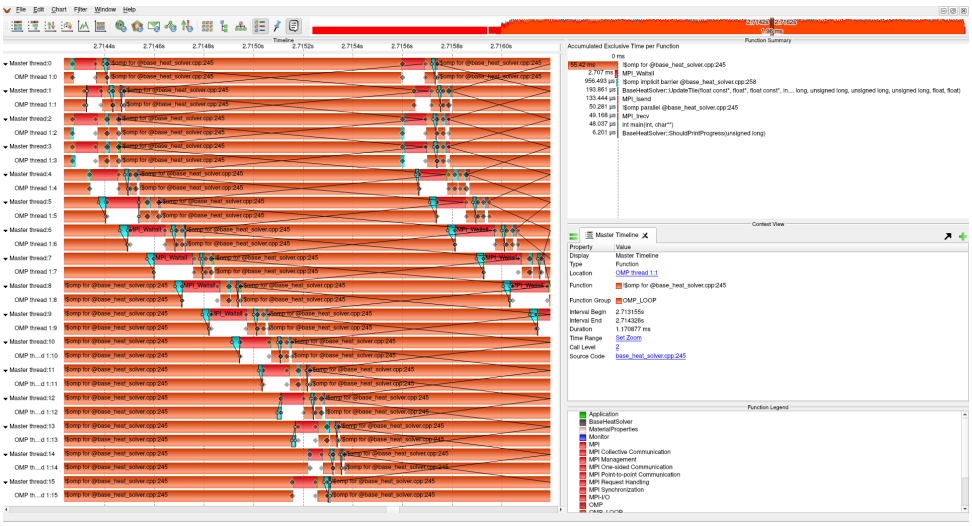
Obrázek 5: Profilování pro 16 procesů v programu Cube. Zobrazení počtu volání funkce `MPI_ISend` v rámci výpočtu simulace.



Obrázek 6: Profilování pro 16 procesů v programu Cube. Zobrazení počtu odeslaných bytů v rámci celého výpočtu (1D dekompozice).



Obrázek 7: Profil celého programu pro 16 procesů a 1D dekompozici v programu Vampir.



Obrázek 8: Ukázka překrytí komunikace výpočtem pro 16 procesů a 1D dekompozici v programu Vampir.

Závěr

Podařilo se implementovat všechny části požadované v zadání a byla vyhodnocena funkčnost a efektivita vytvořeného řešení. Řešení relativně dobře škáluje se zvyšujícím se počtem jader a je zajištěno překrytí komunikace výpočtem.

Dekompozice v 1D se ukázala jako méně efektivní pro větší velikosti matice. Použití vzdáleného přístupu do paměti pro výměnu okrajů je méně efektivní než použití oboustranných komunikací. Použití 1D dekompozice a vzdáleného přístupu do paměti pro větší velikosti matice je extrémně neefektivní.

Paralelní ukládání do souboru je pro většinu testovaných velikostí matice méně efektivní než sesbírání dat na procesu *root* a sekvenční uložení. Pro větší rozměry už se ale rozdíl snižuje a při dalším zvětšení by zřejmě začal být efektivnější paralelní přístup. Také by mohla pomoci další optimalizace paralelního přístupu do souboru.

Problémem algoritmu je inicializační fáze, kdy dělá větší část práce proces *root*, který poté rozesílá ostatním procesům data, na které musí počkat před začátkem výpočtu. Tímto je snížena efektivita celého výpočtu.