Praktické paralelní programování
Dokumentace k projektu č. 1
MPI a paralelní I/O

Jakub Vlk (xvlkja07)

April 26, 2025

1 Úvod

Tento projekt se zaměřuje na implementaci 2D simulace šíření tepla v procesorovém chladiči s využitím metody konečných diferencí (FDTD). Hlavním úkolem bylo navrhnout paralelní řešení pomocí standardu MPI.

Projekt kombinuje implementaci numerického modelu s pokročilými aspekty paralelního programování: kartézskou topologií procesů, optimalizací komunikace neblokujícími operacemi a efektivním ukládáním dat pomocí knihovny HDF5.

2 Popis implementace

Implementace vychází z dekompozice 2D domény na pravidelné obdélníkové dlaždice, kde každý MPI proces zpracovává jednu oblast. Pro správu paměti lokálních dlaždic (včetně hraničních zón) a další pomocné úlohy byly využity standardní kontejnery jazyka C++.

2.1 Komunikátory

V projektu byly použity dva hlavní MPI komunikátory:

- `gridComm` - reprezentuje 2D kartézskou topologii procesů a slouží ke zjištění sousedů a výměně hraničních hodnot
- `avgTempComm` - sdružuje procesy obsahující část globálního prostředního sloupce domény a slouží pro paralelní výpočet průměrné teploty

2.2 Datové typy

Pro efektivní přenos dat byly využity odvozené MPI datové typy:

1. **Pro distribuci/sběr dat:** Typy `tileTypeFloat`, `tileTypeInt`, `m_localActiveAreaTypeFloat` a `m_localActiveAreaTypeInt` definují rozložení dlaždic a jejich aktivních oblastí.
2. **Pro výměnu hraničních zón:** Typy `horizontal_strip_type` a `vertical_strip_type` popisují okrajové "proužky" dat, které je třeba vyměnit mezi sousedními procesy.

2.3 KaMPIng

V rámci experimentu byla implementována varianta výpočtu průměrné teploty pomocí moderní C++ knihovny KaMPIng[1]. Cílem bylo nahradit standardní postup s `MPI_Comm_split` a `MPI_Reduce` voláním kolektivní operace `kamping::reduce`.

Integrace knihovny byla úspěšná, ale vyžadovala lokální úpravy a specifickou verzi CMake, což způsobilo nekompatibilitu na cílovém clusteru. S ohledem na dlouhé čekací doby na výpočetní uzly (i více než 10 hodin) jsem prioritizoval povinné části projektu. Upravený `CMakeLists.txt` pro kompilaci s KaMPIng je přiložen k projektu.

Z experimentu vyplynuly následující postřehy:

Výhody KaMPIng:

- *Redukce počtu parametrů a typová bezpečnost:* Efektivní využití C++ šablon snižuje riziko chyb.
- *Moderní API:* Pojmenované parametry (např. `kamping::send_buf`) zvyšují čitelnost.
- *Snadná definice vlastních operací:* Definování redukčních operací je přímočaré.
- *Správa zdrojů (RAII):* Automatická správa zdrojů prostřednictvím objektů.

Nevýhody:

- *Složité chybové hlášky:* Šablony generují obtížně srozumitelná chybová hlášení.
- *Dokumentace:* Dostupné materiály by mohly být komplexnější.
- *Závislost na znalostech C++:* Vyžaduje dobrou znalost moderního C++.
- *Kompilace a závislosti:* Kompatibilita s HPC prostředím může být problematická.
- *Stabilita API:* Jako aktivně vyvíjená knihovna podléhá API změnám.

3 Otázky k projektu

- **Jaký je rozdíl mezi škálováním/efektivitou 1D a 2D dekompozice a čím je tento rozdíl způsoben?**

  Z grafů zrychlení (Obrázek 4) a efektivity (Obrázek 2) vyplývá, že 2D dekompozice konzistentně předčí 1D dekompozici, zvláště při větším počtu procesů. Hlavní příčinou je efektivnější poměr komunikace k výpočtům - zatímco v 1D procesy komunikují podél dlouhé hrany, v 2D komunikují podél kratších hran, což vede k menšímu objemu přenášených dat a nižší komunikační režii.

- **Jaký je vliv paralelního I/O v porovnání se sekvenčním?**

  Překvapivě, ve všech měřených případech bylo sekvenční zapisování rychlejší než paralelní. Pravděpodobnou příčinou byla vysoká vytíženost superpočítače Barbora - nebyl jsem jediný, kdo se v danou chvíli pokoušel o paralelní zápis.

- **Jakým způsobem lze zefektivnit paralelní I/O?**

  Paralelní I/O lze zefektivnit několika způsoby:

  – Optimalizací nastavení souborového systému Lustre (striping).

[1] `https://github.com/KaMPIng/KaMPIng`

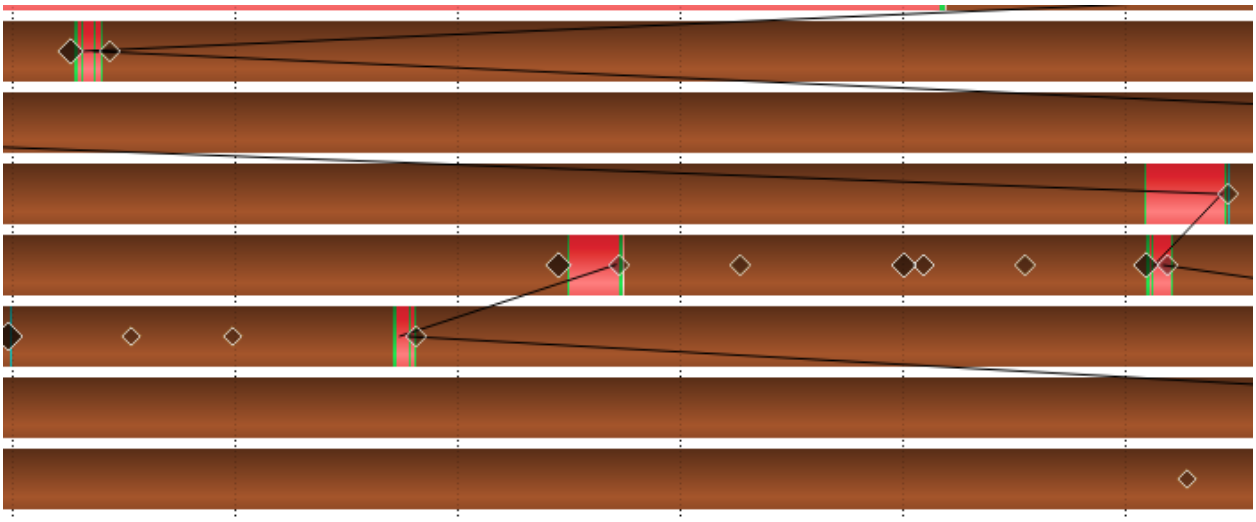


Figure 1: Znázornění překrytí komunikace a výpočtu: hnědě jsou vyznačeny výpočty v rámci funkce updateTile, červeně explicitní čekání.

  - Důsledným používáním kolektivních MPI-IO operací.
  - Vhodným nastavením HDF5 chunkingu a zarovnání dat.
  - Měřením v době s minimálním vytížením systému.
  - Agregací menších zápisů nebo využitím asynchronního I/O.

- **Jak se liší množství komunikace mezi jednotlivými procesy v 1D a 2D dekompozici? Je zátěž vyrovnaná?**

  Objem komunikace pro výměnu hraničních zón je na proces obecně menší u 2D dekompozice než u 1D (pro více než 4 procesy) díky lepšímu poměru povrchu k objemu. Výpočetní zátěž je při rovnoměrném rozdělení domény dobře vyrovnaná v obou případech. Komunikační zátěž však není dokonale vyrovnaná - okrajové procesy komunikují s menším počtem sousedů než vnitřní procesy.

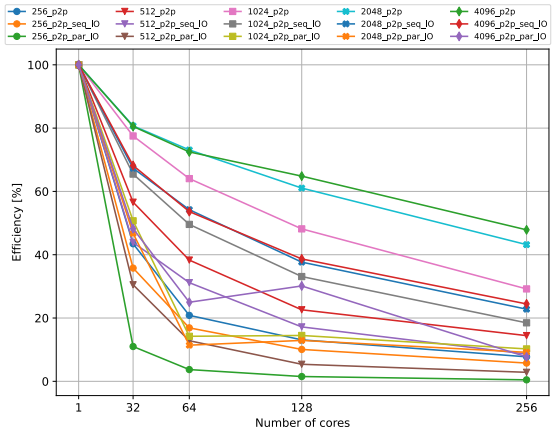
- **Jaký přínos má překrytí komunikace a výpočtu?**

  Překrytí komunikace a výpočtu umožňuje provádět výpočty nezávislé na přenášených datech souběžně s neblokující komunikací. Tím se skrývá latence přenosu dat, procesor nezůstává nečinný a zkracuje se celkový čas simulace. Tento efekt je vidět na obrázku 1, kde procesy pokračují ve výpočtu i přes vzájemné závislosti.

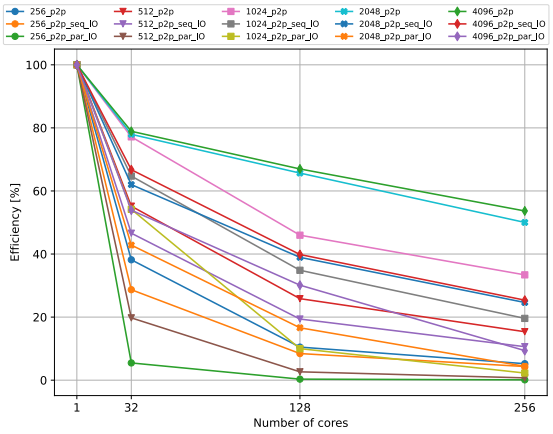
4 Grafy získané pomocí skriptů

4.1 Grafy efektivity

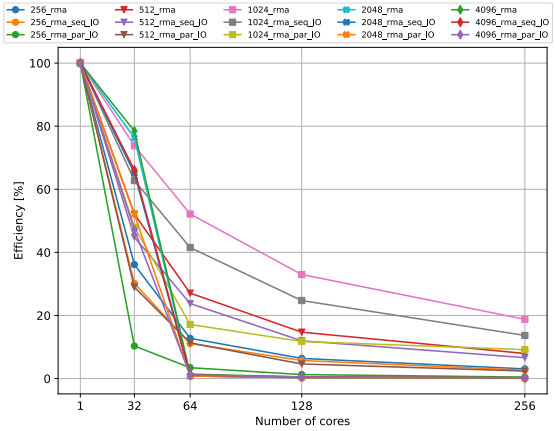
Grafy efektivity (Obrázek 2) ukazují očekávaný pokles s rostoucím počtem jader, přičemž 2D dekompozice si udržuje vyšší efektivitu déle než 1D. Rozdíly mezi P2P a RMA nejsou příliš výrazné, ale P2P se jeví jako obecně efektivnější.



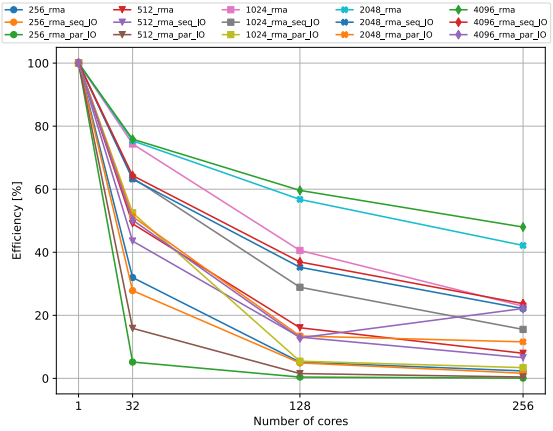
(a) Efektivita: Hybrid 1D P2P.



(b) Efektivita: Hybrid 2D P2P.



(c) Efektivita: Hybrid 1D RMA.

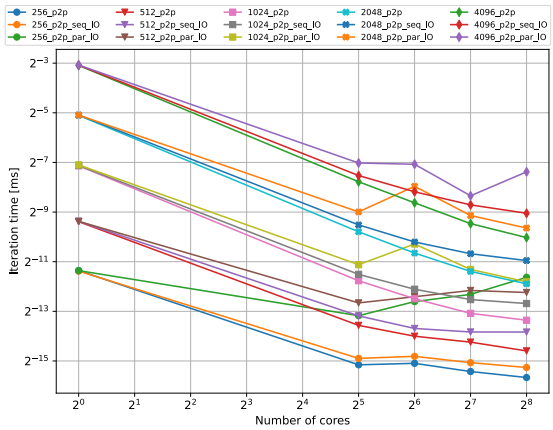


(d) Efektivita: Hybrid 2D RMA.

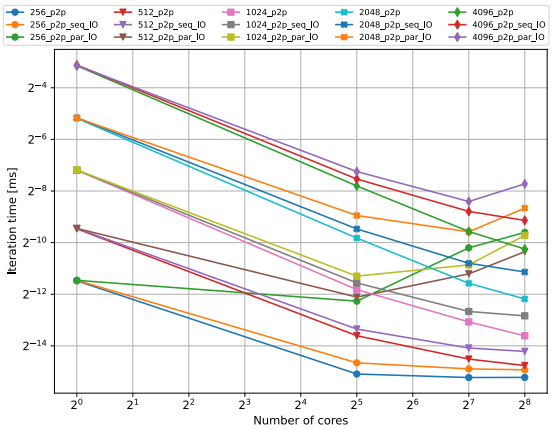
Figure 2: Efektivita paralelní implementace pro hybridní MPI+OpenMP běhy.

4.2 Grafy silného škálování

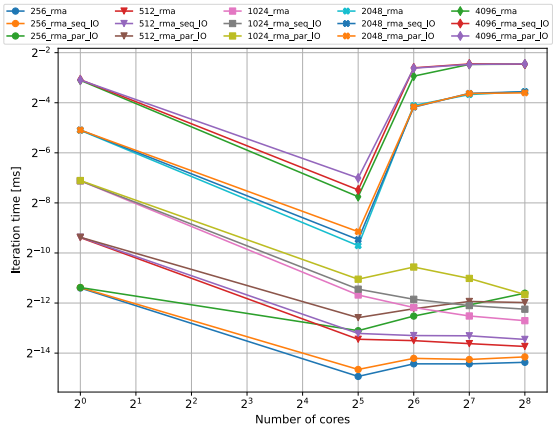
Grafy silného škálování (Obrázek 3) ukazují pokles vykonávacího času s přidáváním jader. 2D dekompozice opět vykazuje nižší časy než 1D. U konfigurace 1D RMA je při přechodu z 16 na 32 jader patrná stagnace pro některé velikosti problému, což naznačuje dosažení limitu škálovatelnosti nebo zvýšenou režii RMA.



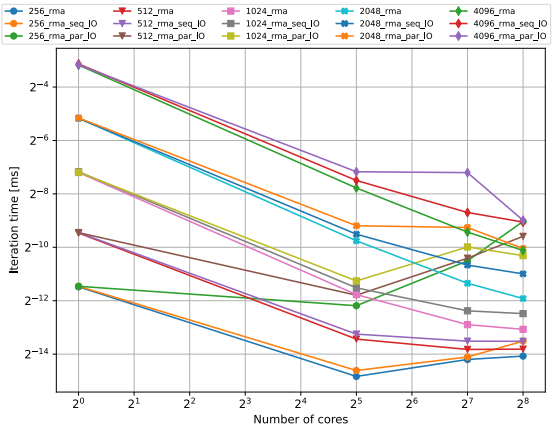
(a) Škálování: Hybrid 1D P2P.



(b) Škálování: Hybrid 2D P2P.



(c) Škálování: Hybrid 1D RMA.

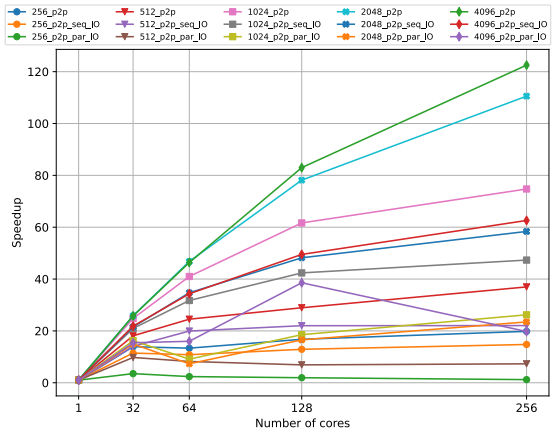


(d) Škálování: Hybrid 2D RMA.

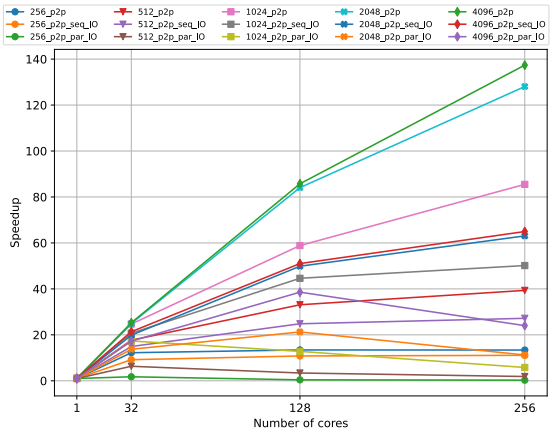
Figure 3: Silné škálování (vykonávací čas) pro hybridní MPI+OpenMP běhy.

4.3 Zrychlení (Speedup)

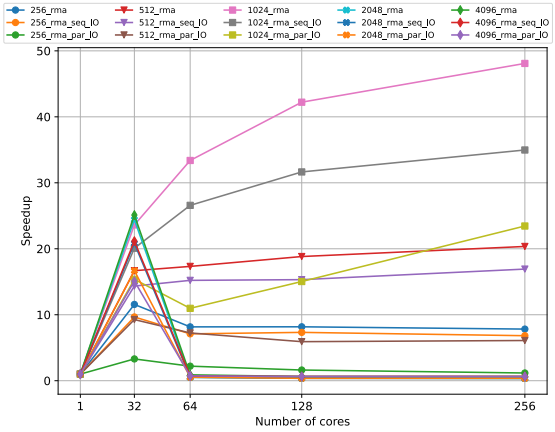
Grafy zrychlení (Obrázek 4) potvrzují předchozí závěry. 2D dekompozice dosahuje výrazně lepšího zrychlení než 1D. Rozdíl mezi P2P a RMA není markantní, P2P se jeví jako mírně lepší nebo srovnatelná. Větší problémy škálují obecně lépe.



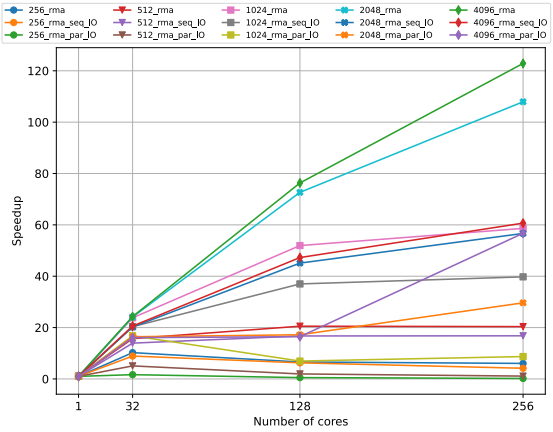
(a) Zrychlení: Hybrid 1D P2P.



(b) Zrychlení: Hybrid 2D P2P.



(c) Zrychlení: Hybrid 1D RMA.

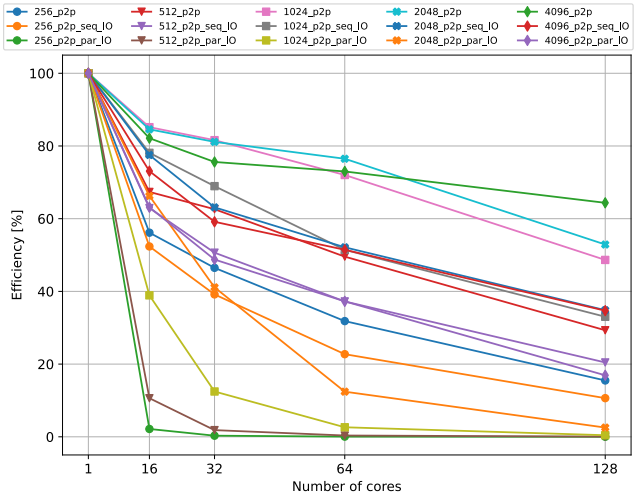


(d) Zrychlení: Hybrid 2D RMA.

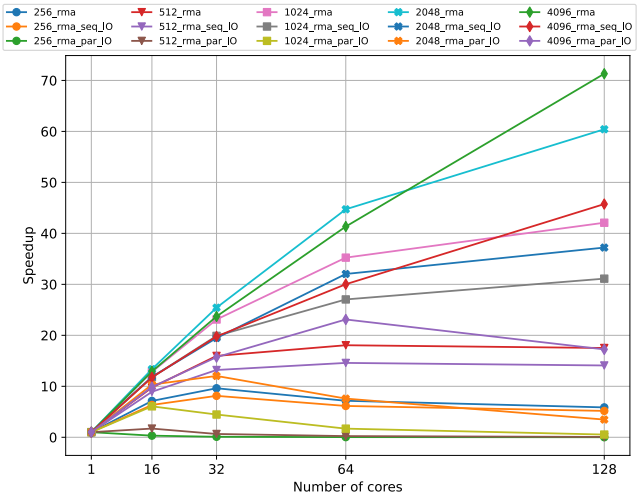
Figure 4: Zrychlení (Speedup) pro hybridní MPI+OpenMP běhy.

4.4 Čisté MPI

Pro srovnání byly provedeny i testy s čistě MPI variantou (jeden proces na jádro, bez OpenMP vláken) s 2D dekompozicí. Tyto výsledky jsou uvedeny odděleně, protože byly výrazně pomalejší než hybridní přístup.



(a) Efektivita: MPI s 2D dekompozicí, výměna okrajů pomocí P2P.



(b) Zrychlení: MPI s 2D dekompozicí, výměna okrajů pomocí RMA.

5 Profilování

Bohužel jsem se s profilováním na Barboře nedostal na řadu ani po 24 hodinách čekání. Profilování jsem tedy provedl na svém počítači a screenshoty níže reprezentují data naměřená v tomto prostředí.[2]

[2]Pro vizualizaci ve Vampire jsem lokálně nainstaloval ScoreP a data jsem přenesl na Barboru. To samé platí pro Cube a Visit. Na svém domácím "superpočítači" disponuji skromnými 24 vlákny. :)

5.1 Vampir

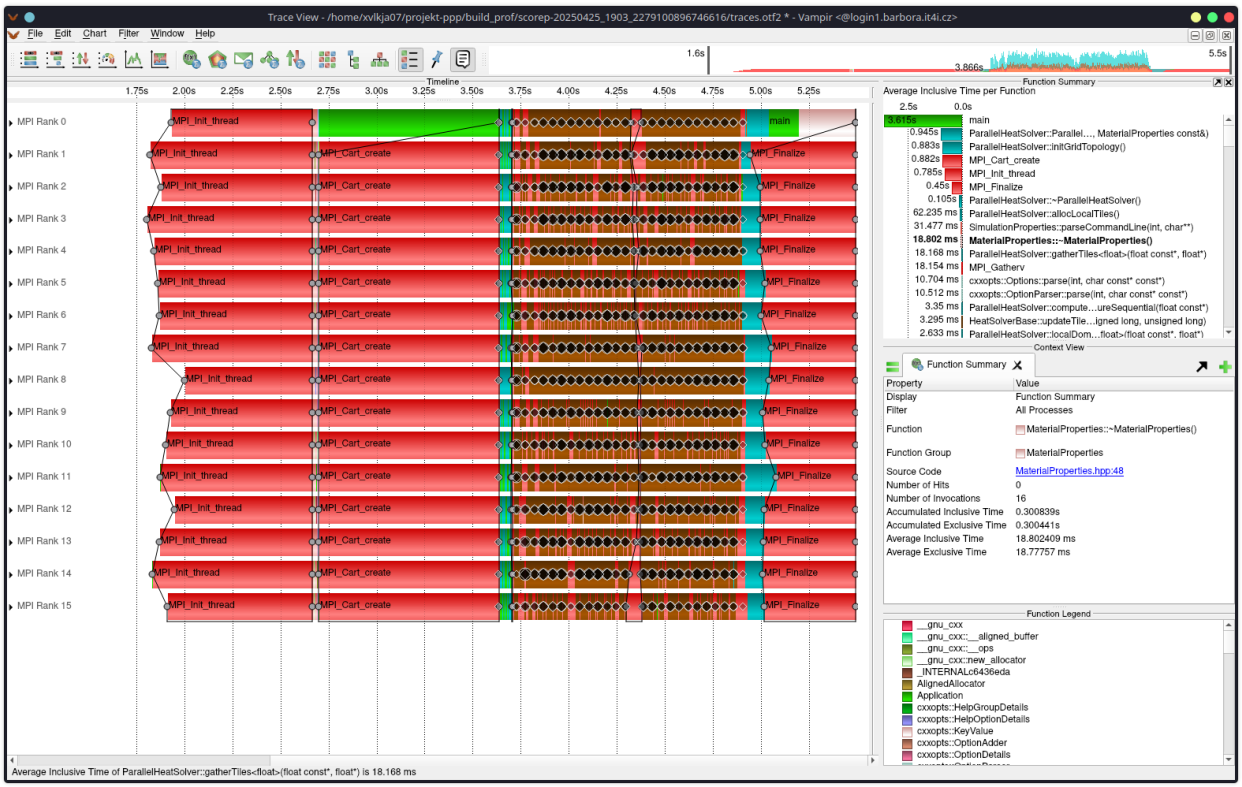


Figure 6: Vampir: Časová osa a souhrnné statistiky pro hybridní běh s 2D dekompozicí (16 procesů, P2P). Je zřetelný čas strávený v jednotlivých funkcích, MPI operacích a OpenMP bariérách. Relativně dlouhá inicializace může zahrnovat alokace, scatter a počáteční výměnu hraničních zón.

5.2 Cube

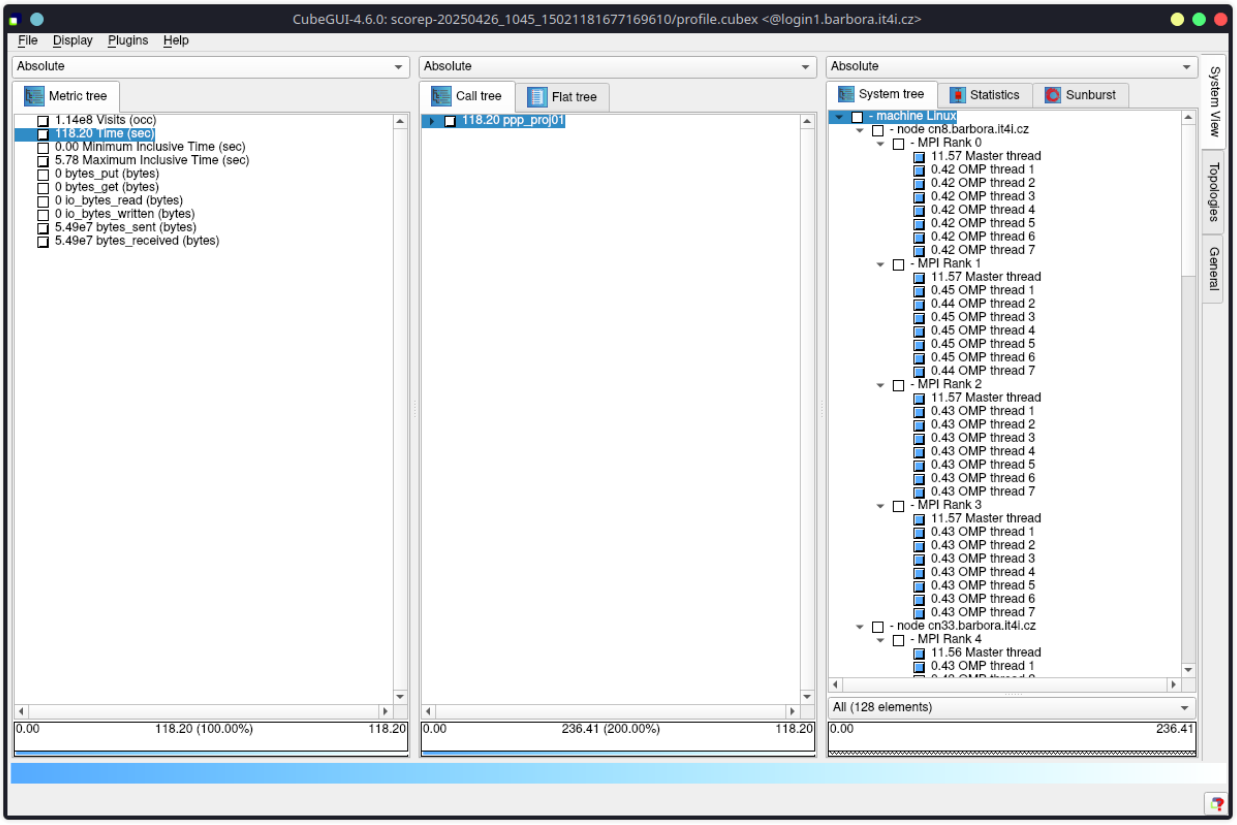


Figure 7: Cube: Zobrazení exkluzivního času stráveného v jednotlivých funkcích a jak efektivitu rozložení mezi ranky. Metriky potvrzují, že nejvíce času zabírá výpočetní část (`computePoint`, `updateTile`) a čekání na MPI komunikaci (`MPI_Waitall`)(Nakonec se mi povedlo na posledních chvíli spusti profilování abych získal subory pro cube).

5.3 VisIt

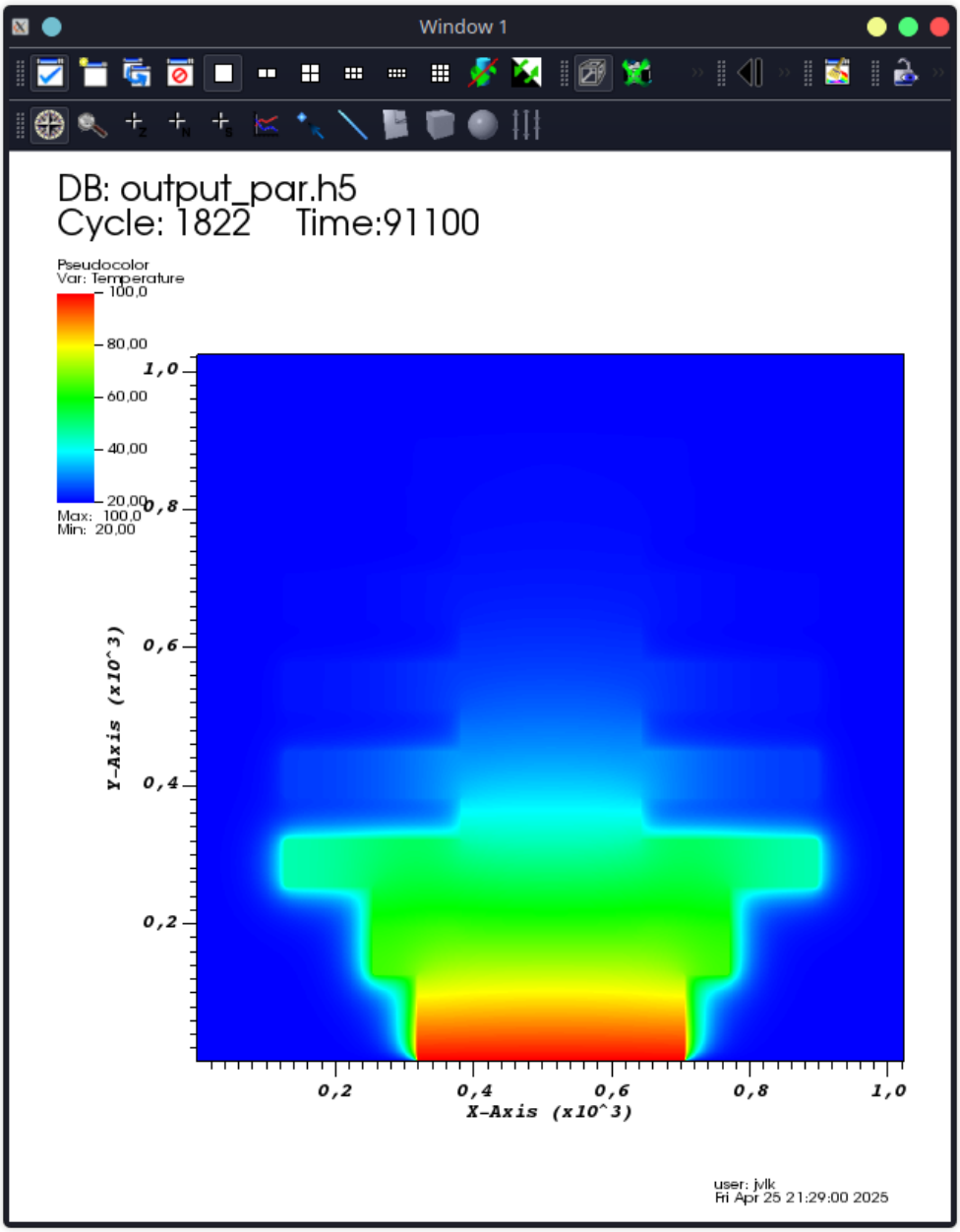


Figure 8: VisIt: Vizualizace teplotního pole v pozdějším kroku simulace. Je vidět šíření tepla z centrálního zdroje (procesoru) do struktury chladiče a jeho žeber.

6 Závěr

Projekt úspěšně demonstroval implementaci paralelní simulace šíření tepla pomocí MPI a hybridního přístupu MPI+OpenMP. Zkoumáním různých strategií dekompozice domény (1D vs. 2D) a komunikačních mechanismů (P2P vs. RMA) byla potvrzena výhoda 2D dekompozice z hlediska škálovatelnosti díky redukci komunikační režie.

Během projektu jsem získal praktické zkušenosti s tvorbou odvozených MPI datových typů, neblokující komunikací, paralelním I/O pomocí HDF5 a základy profilování pomocí nástrojů Score-P, Vampir a Cube. Experiment s knihovnou KaMPing ukázal potenciál moderních C++ wrapperů pro MPI, ale i výzvy spojené s jejich integrací v HPC prostředí.