Vysoké učení technické v Brně
Fakulta informačních technologií

Praktické paralelní programování

Projekt č. 1 – MPI a paralelní I/O

Bc. Vojtěch Mimochodek (xmimoc01) **28. dubna 2024**

1 Úvod

Cílem projektu bylo naimplementovat simulaci šíření tepla 2D řezem procesorového chladiče, přičemž byl kladen důraz na zasílání zpráv s využitím knihovny MPI a HDF5 pro paralelní I/O. Součástí je také vyhodnocení na superpočítači Barbora a ověření škálování.

2 Implementace

Vypracované zdrojové kódy projektu se nachází v souborech *ParallelHeatSolver.cpp* a *ParallelHeatSolver.h*. Implementace zahrnuje všechny požadované části projektu definované zadáním.

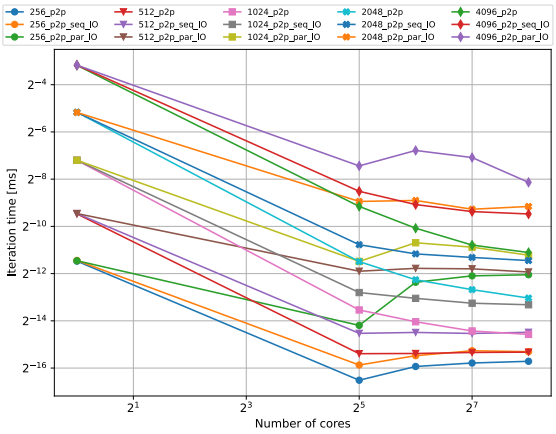
Na začátku běhu programu jsou v konstruktoru inicializovány potřebné proměnné. Jsou to rozměry celkové domény, rozměry jednotlivých submatic, inicializace mřížkové topologie a komunikátoru pro výpočet středního sloupce, vytvoření datových typů pro vnitřní halo zóny i pro vnější prostor k načtení halo zón od sousedů či vytvoření oken do sdílené paměti pro případ, kdy je program spuštěn s komunikací skrze RMA. Dále je celková doména rozdělena na submatice a ty jsou pomocí funkce *scatterv()* rozeslány jednotlivým procesům. Takto jsou rozeslány počáteční teploty v doméně, mapa a parametry. Po úspěšném rozeslání hodnot submatic jsou rozeslány pomocí P2P hodnoty teplot a parametrů vnitřních halo zón sousedům potřebných pro výpočet v první iteraci. Následně tok programu vstupuje do iterativní smyčky, ve které jsou nejprve spočítány hodnoty v halo zónách, které jsou následně rozeslány pomocí P2P nebo RMA. Během komunikace jsou počítány prostřední matice, čímž je zajištěno překrytí komunikace a výpočtu. Výpočet hodnot je zajištěn pomocí funkce *updateTile()*. Na konci iterace jsou prohozeny lokální pole pro uchovávání hodnot teplot. Tak aby nově vypočtené hodnoty byly v další iteraci jako výchozí. Pokud je povolen zápis, jsou s každou iterací zapsány hodnoty do souboru pomocí knihovny HDF5. A to buď sekvenčně nebo paralelně. Rovněž může být vypočítána a vypsána průměrná teplota ve středním sloupci. Po dokončení iterativní smyčky jsou pomocí funkce *gatherv()* sesbírány hodnoty matic ze všech procesů a seskládány zpět do matice o velikosti původní domény. Nakonec je z této finální matice na root ranku sekvenčně vypočítána průměrná teplota ve středním sloupci.

3 Profiling

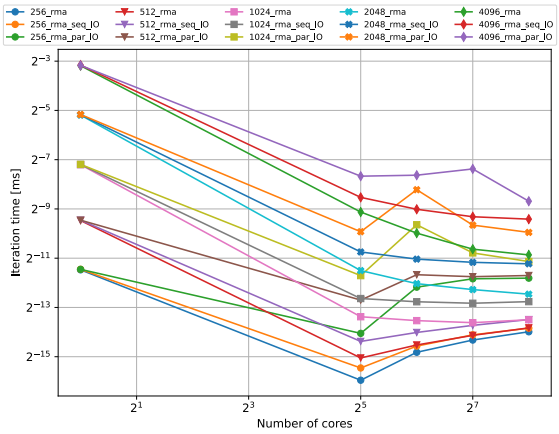
Řešení projektu bylo otestováno všemi přiloženými scripty ze složky */scripts*. Veškeré scripty proběhly v pořádku a bez chyb.

3.1 Silné škálování

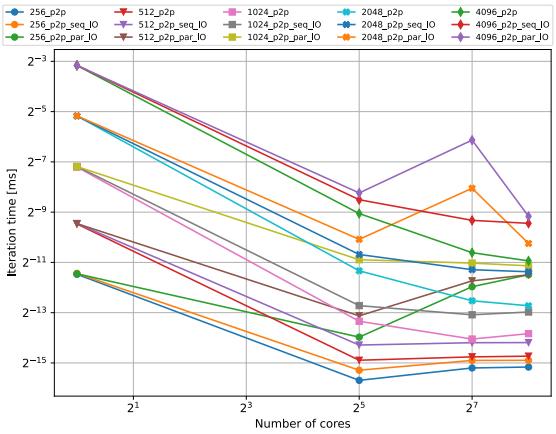
První dva obrázky ukazují výsledky scriptů *run_full_hybrid_1d.sh*. Druhé dva pak *run_full_hybrid_2d.sh* a třetí dva *run_full_mpi_2d.sh*



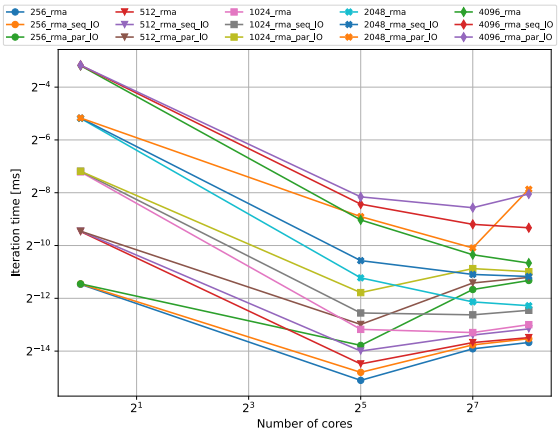
Obrázek 1: Silné škálování 1D s P2P komunikací, script hybrid



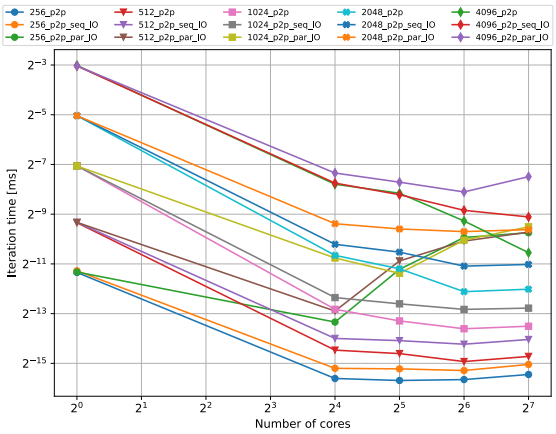
Obrázek 2: Silné škálování 1D s RMA komunikací, script hybrid



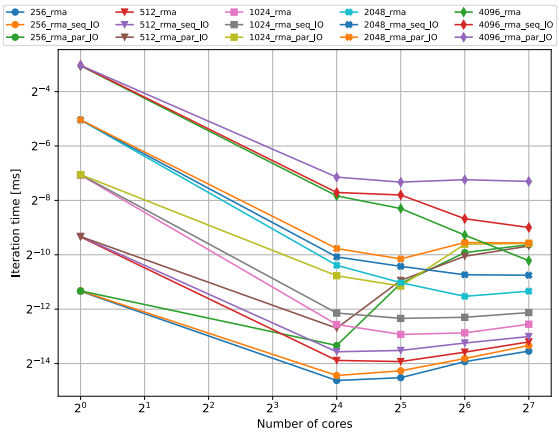
Obrázek 3: Silné škálování 2D s P2P komunikací, script hybrid



Obrázek 4: Silné škálování 2D s RMA komunikací, script hybrid



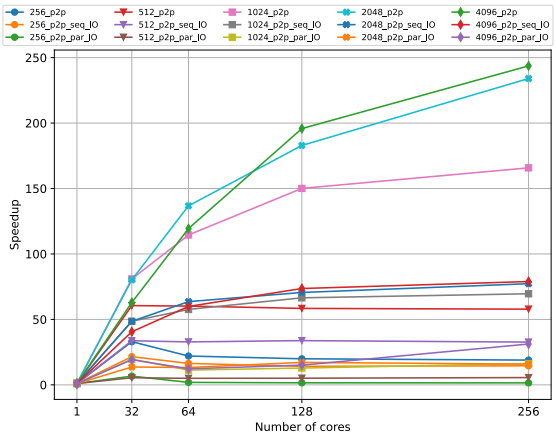
Obrázek 5: Silné škálování 2D s P2P komunikací, script mpi



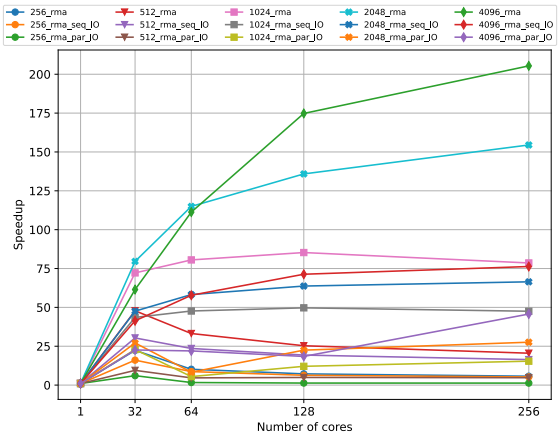
Obrázek 6: Silné škálování 2D s RMA komunikací, script mpi

3.2 Slabé škálování

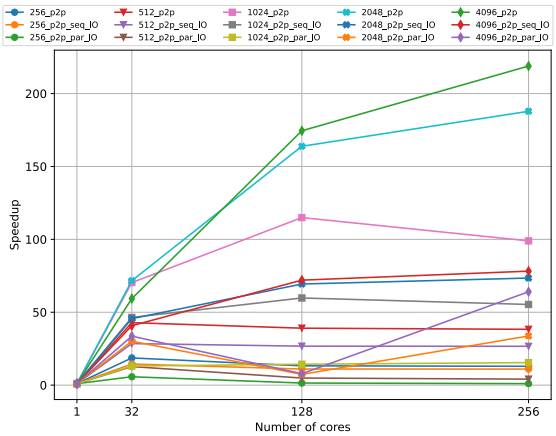
Grafy s výsledky pro slabé škálování jsou rozvrženy stejně jako v sekci u silného škálování ve stejném pořadí dle scriptů.



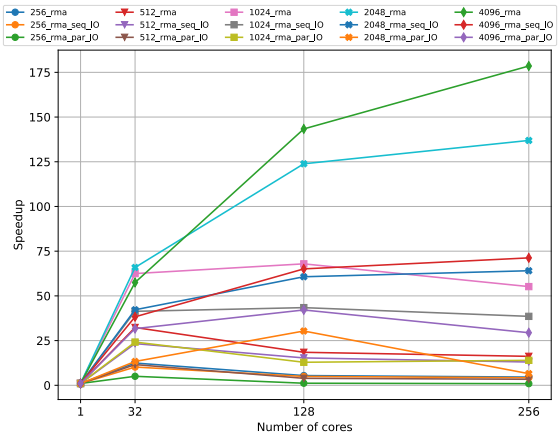
Obrázek 7: Slabé škálování 1D s P2P komunikací, script hybrid



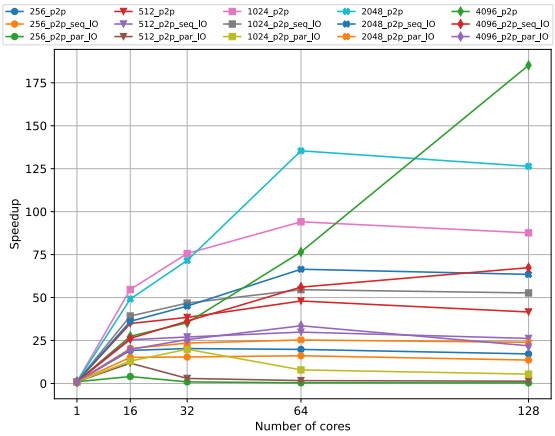
Obrázek 8: Slabé škálování 1D s RMA komunikací, script hybrid



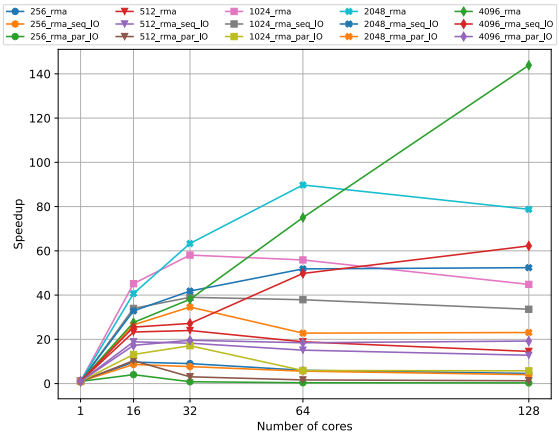
Obrázek 9: Slabé škálování 2D s P2P komunikací, script hybrid



Obrázek 10: Slabé škálování 2D s RMA komunikací, script hybrid



Obrázek 11: Slabé škálování 2D s P2P komunikací, script mpi



Obrázek 12: Slabé škálování 2D s RMA komunikací, script mpi

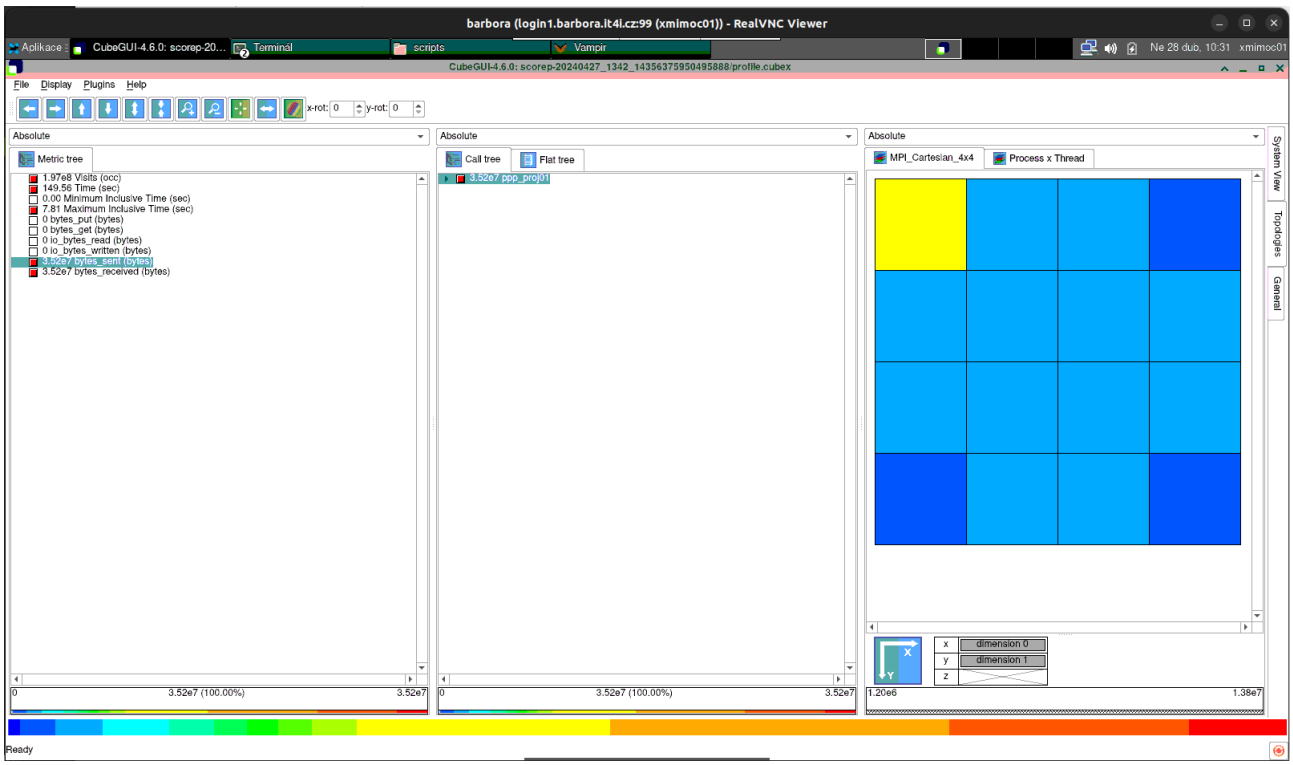
U silného škálování se zdá malé zlepšení u 2D, u slabého škálování se zrychlení zdá trochu lepší u 1D dekompozice. U silného i slabého škálování však jsou výsledky mezi 1D a 2D dekompozicí velmi podobné a výkon se tedy příliš drasticky neliší. Je možné, že hlavním důvodem těchto výsledků je výpočet nových hodnot pomocí funkce *updateTile()*. Jisté rozdíly lze také sledovat při použití jiných přístupů v komunikaci - P2P a RMA.

Zápis do souboru dává smysl až při větších velikostí domén. Pro malé velikosti je efektivnější sekvenční zápis do souboru.

Největší efektivitu mají ty běhy bez I/O operací.

3.3 Vyhodnocení pomocí Cube

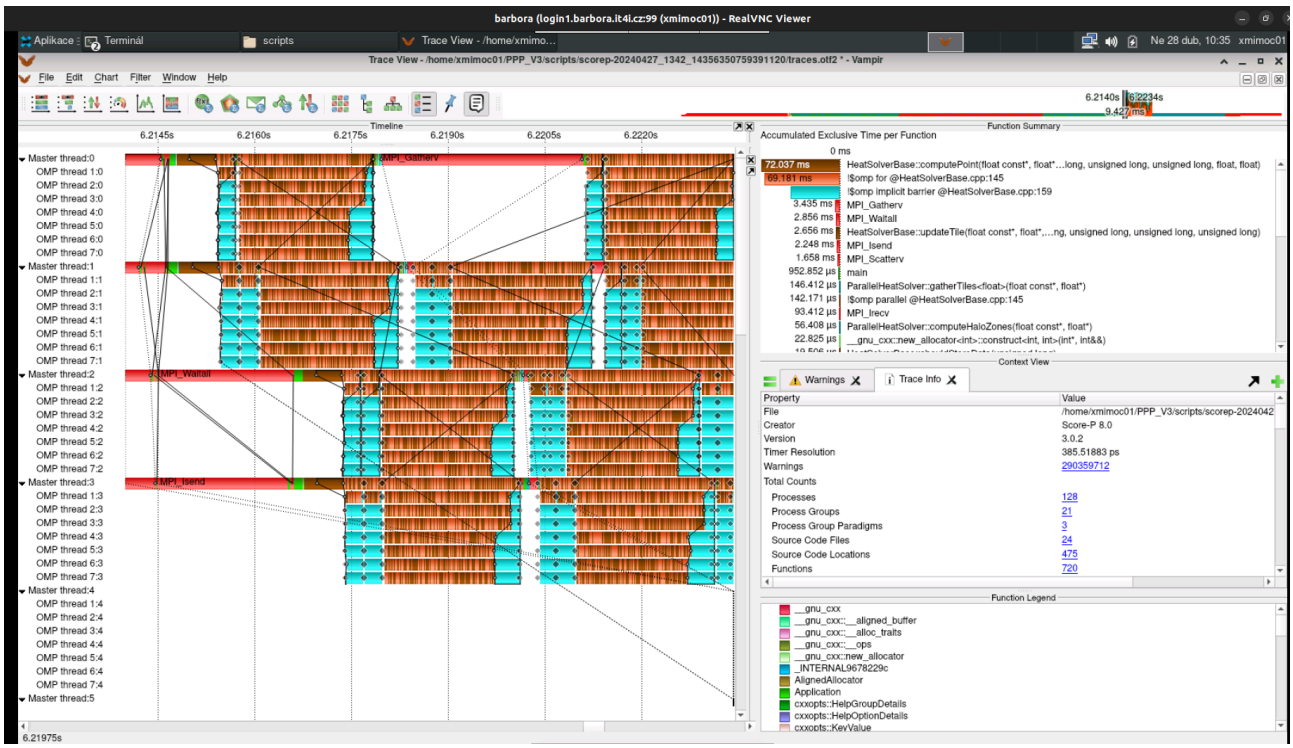
S pomocí nástroje Cube, je možné vidět vytížení jednotlivých procesů v komunikátoru. Ten zobrazuje procesy v kartézské topologii řazené po řádcích. Výrazně nejvytíženější proces je root. To proto, že se stará například o počítání průměrné teploty nebo distribuci dat mezi procesy a sesbírání výsledků. Rovněž mírně větší zátěž lze sledovat na procesech, které se nacházejí uvnitř nebo uprostřed krajů. Nejmenší zátěž mají procesy v rozích topologie. Lze konstatovat, že zátěž mezi procesy není rovnoměrná.



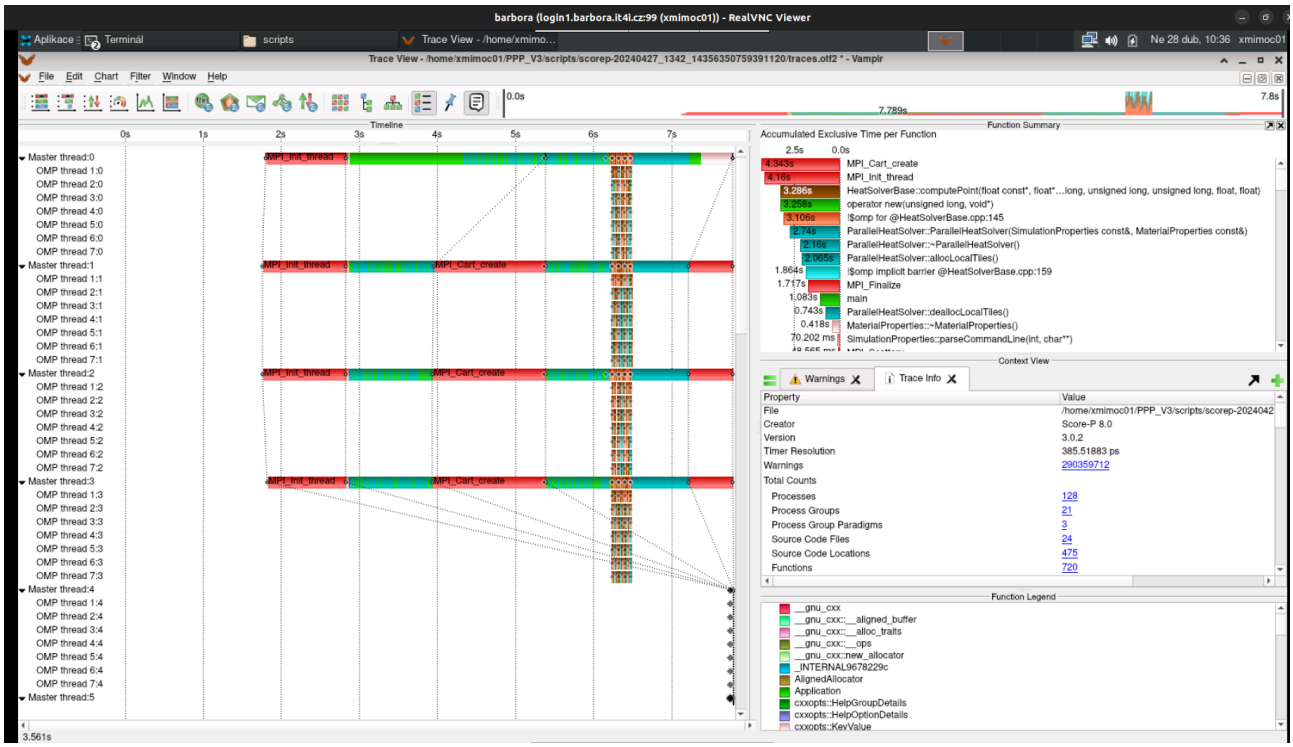
Obrázek 13: Vyhodnocení programem Cube

3.4 Vyhodnocení pomocí Vampir

Na obrázcích lze vidět vyhodnocení v programu Vampir. Na prvním obrázku se nachází přiblížená část programu s výpočtem. Zde lze vidět překrytí komunikace a výpočtu. To je vhodné k urychlení tím způsobem, že je možné počítat zatímco probíhá komunikace. Na druhém obrázku je vidět celkový tok programu. Nejdéle času zabírá vytvoření kartézské topologie pomocí funkce *MPI_Cart_create()*.



Obrázek 14: Vyhodnocení programem Vampir - překrytí komunikace



Obrázek 15: Vyhodnocení programem Vampir - celkový tok programu

4 Závěr

Povedlo se naimplementovat projekt se všemi požadovanými body zadání. Správnost řešení byla postupně testována oproti sekvenční verzi. Nakonec byly spuštěny přiložené scripty a provedeno vyhodnocení pomocí nástroje Vampir a Cube.