Architektura a programování paralelních systémů
Projekt č. 1 - Programování s OpenMP

Marta Čudová, icudova@fit.vutbr.cz
Jiří Jaroš, jarosjir@fit.vutbr.cz

Termín odevzdání: 26. března 2017 23:59:59 (neděle)
Hodnocení: až 10 bodů

1 Úvod

Cílem projektu je osvojit si základní principy programování se sdílenou pamětí s využitím knihovny OpenMP, vyzkoušet si různé principy paralelizace, práce s I/O a ověřit škálování na procesorové NUMA architektuře a Intel Xeon Phi akcelerátoru superpočítače Salomon.

Jako vhodný testovací problém byla zvolena numerická simulace problému šíření tepla. Budeme tedy uvažovat jeden 2D řez procesorovým chladičem věžovité konstrukce, který má měděnou základnu a nosnou konstrukci, ke které je připojeno několik hliníkových žeber. Na styčné ploše chladiče s procesorem dochází k přenosu tepla z procesoru do chladiče. Toto teplo přechází „difuzí“ do žeber chladiče, kde je dále předáváno chladicímu médiu, jímž je zde vzduch. Pro realistickou simulaci uvažujeme, že vzduch proudí laminárně kolmo k ose řezu chladiče (teplo se tedy postupně odvádí pryč ze zkoumané oblasti).

Aby bylo možné analyzovat výsledky simulace, je průběh ohřívání chladiče ukládán do souboru. Tento soubor nám poslouží k vizualizaci výsledků a ověření správnosti výpočtu. Problém však nastává, pokud ukládáme data příliš často. Může tedy snadno nastat situace, kdy je I/O hlavní brzda výpočtu (I/O nelze v OpenMP paralelizovat).

Jak tedy postupovat při řešení projektu?

1. Získejte od Jirky Jaroše přístupové údaje na superpočítač Salomon (pozor, hned první krok potupně odhalí těžké prokrastinátory).
2. Přihlašte se na Salomon, sem překopírujte a rozbalte zdrojáky z wisu, projekt přeložte a spusťte jednoduchý testovací běh, viz kapitolu 10.2.
3. Seznamte se s letos řešeným problémem jak na teoretické úrovni (viz kapitolu 2), tak na té implementační (následující 4 kapitoly).
4. Rámcově pochopte, co se po vás chce (již předimplementovanou funkční sekvenční verzi paralelizujte tak, aby korektně počítala a zároveň dosáhla maximálního výkonu, nic víc).
5. Implementaci proveďte.
6. Zbenchmarkujte svoje řešení připravenými testy jak na procesoru, tak na Xeon Phi akcelerátoru (viz kapitolu 12).
7. Odevzdejte požadované soubory do wisu před deadlinem.

2 Popis numerické metody šíření tepla

Existuje několik metod, jak vyřešit problém šíření tepla numericky. Jmenujme například metody konečných diferencí, konečných elementů, metody hraničních prvků či spektrální metody. My se zaměříme na tech-

niku nejjednodušší (bohužel i nejméně přesnou a nejpomalejší), kterou je metoda označovaná jako FDTD[1] (*Finite Difference Time Domain*), tedy metoda konečných diferencí v čase.

Zkoumanou doménu tedy diskretizujeme pomocí uniformní mřížky velikosti $N \times N$ bodů. Pro jednoduchost uvažujme mřížku čtvercového tvaru. V každém bodě mřížky definujeme potřebné parametry média. V našem případě postačuje tepelná vodivost pro měď, hliník a vzduch. Pokud daný bod domény připadá na vzduch, zavádíme rovněž koeficient perfuze (tedy jak rychle proudí vzduch okolo chladiče a odvádí teplo pryč). Dále pak v každém bodě udržujeme aktuální teplotu chladiče. Na počátku simulace je nastavena klidová teplota (výchozí hodnota je $20°C$). V místě styku chladiče s procesorem je nastavena konstantní teplota ohřívače (výchozí hodnota je $100°C$). Pro úplnost dodejme, že teplota na hranách domény je konstantní (tzv. *Dirichletova podmínka*). Protože simulujeme vývoj teploty na chladiči v čase, diskretizujeme rovněž i časovou osu a simulaci budeme provádět v předem daném počtu kroků.

Představme si nyní, že máme spočítat teplotu v čase $t+1$. K tomuto účelu využijeme jednak znalosti aktuální teploty v okolí zkoumaného bodu a jednak historii vývoje změny teploty v daném bodě v čase. Díky těmto údajům spočteme gradient teploty v prostoru a provedeme dopřednou integraci v čase. K tomuto účelu byla vymyšlena spousta zajímavých metod (např. metoda Runge-Kutta 4. řádu, vícekrokové metody, atd.).

My opět zvolíme tu nejjednodušší techniku a budeme aproximovat gradient v prostoru pomocí jednoduchého (4+4)-okolí. Následující rovnice definuje metodu FDTD 2. řádu v prostoru a 1. řádu v čase:

$$T_{t+1}[i][j] = \frac{T_t[i-1][j] + T_t[i-2][j] + T_t[i+1][j] + T_t[i+2][j] + T_t[i][j-1] + T_t[i][j-2] + T_t[i][j+1] + T_t[i][j+2] + T_t[i][j]}{9.0} \tag{1}$$

Tato rovnice zjednodušeně říká, že teplota bodu v čase $t+1$ ja dána průměrem teplot tohoto bodu a jeho 8 sousedů v čase t. Takto bychom však byli schopni simulovat šíření tepla pouze v homogenním m0diu. My ale chceme použít měď, hliník a vzduch. Heterogenitu proto zavedeme pomocí koeficientu difuze, jenž definuje tepelnou vodivost. Každý člen rovnice je tedy vynásoben normalizovanou hodnotou tepelné vodivosti v daném bodě.

Abychom byli schopni simulovat odvod tepla proudícím vzduchem, je každý bod m0dia odpovídající vzduchu upraven pomocí následujícího vztahu.

$$T_{t+1}[i][j] = (\alpha * T_0) + (1 - \alpha * T_{t+1}([i][j])) \tag{2}$$

kde α definuje relativní rychlost proudění vzduchu vzhledem k délce časového kroku a T_0 je teplota čerstvého vzduchu.

Výpočet nyní probíhá tak, že procházíme jednotlivé body a aktualizujeme teplotu. Při běžné (sekvenční i paralelní) implementaci je nutné využít dvě pole (zdrojové a cílové), tak aby se nové hodnoty počítaly vždy pouze ze starých hodnot a nedocházelo k tomu, že někteří sousedé daného bodu byli již aktualizováni a jiní ne. To by v případě paralelní verze vedlo na nedeterministické chování a každý nový běh by dával odlišné výsledky.

Abychom zjednodušili testování implementace, budeme v každém kroku počítat i průměrnou teplotu v oblasti dané jedním sloupcem bodů kolmým na základnu.

3 Knihovna HDF5 pro práci s velkými daty

HDF5[2] je knihovna (a zároveň označení pro její souborový formát) určená pro rychlé čtení/ukládání vědeckých dat na pevném disku. Jak název napovídá, data jsou v souboru uložena hierarchicky ve stromové struktuře tvořené tzv. skupinami (`groups`) a jednotlivými datasety (`datasets`) uchovávajícími uživatelská data. Datasety mají formát N-rozměrné matice určitého datového typu (`int`, `float`, `string`, ...). V rámci tohoto projektu budeme používat sériovou verzi knihovny HDF5, jenž je určená pro práci na jednom výpočetním uzlu.

HDF5 soubory mají zpravidla koncovku `.h5`. Pro zjištění obsahu takového souboru slouží utilita `h5dump`, např. příkaz

```
h5dump -H soubor.h5
```

vypíše základní informace o souboru a typu uložených dat. Více možných příkazů a pohledů na uložená data naleznete v nápovědě (`h5dump --help`).

[1] `http://www.eecs.wsu.edu/~schneidj/ufdtd/ufdtd.pdf`
[2] `http://www.hdfgroup.org/HDF5/`

4 Vizualizační nástroj VisIt

VisIt[3] je interaktivní (klient-server) vizualizační nástroj určený pro grafické zobrazování vědeckých dat. V tomto projektu slouží VisIt k offline zobrazení průběhu simulace uložené v hdf5 souboru, což může výrazně zjednodušit ladění programu, pomůže lépe pochopit řešený problém a dopady jednotlivých změn provedených ve zdrojovém kódu na vlastnosti výstupu. Jedna z dalších velmi používaných alternativ je např. nástroj *ParaView*[4].

5 High-level popis implementace

Po rozbalení balíčku z WISu se objeví tři hlavní složky - `DataGenerator`, `Sources` a `Scripts`. Ve složce `DataGenerator` je implementovaný generátor vstupních souborů s parametry zkoumané oblasti. Mezi tyto parametry patří např. tvar chladiče, tepelná vodivost mědi a hliníku tvořící samotný chladič, vodivost okolního vzduchu, velikost domény, teplota chladiče a procesoru ve výchozím stavu, atd. Tento program generuje soubor v hdf5 formátu, který jde následně na vstup simulátoru (složka `Sources`).

Ve složce `Sources` je hlavní jádro projektu, simulátor šíření tepla v dané 2D doméně. Obsah tvoří tři hlavní zdrojové soubory: `BasicRoutines.cpp` obsahuje jednoduché pomocné funkce pro tisk nápovědy, zpracování parametrů příkazové řádky, atd. `MaterialProperties.cpp` obsahuje třídu, která načte data vytvořená generátorem z hdf5 souboru do paměti. Nejdůležitější je soubor `proj01.cpp`. Zde se nachází plně funkční sekvenční verze simulátoru a vaším úkolem je v označených úsecích implementovat svoje paralelní řešení. Tento soubor se bude následně odevzdávat do wisu.

V poslední složce `Scripts` jsou PBS scripty, které spustí výpočet na Salomonu pro určitou kombinaci počtu vláken a velikostí domény. Jeho výstupem je CSV soubor s informacemi o daných běžích s naměřenými časy. Vaším úkolem bude na základě tohoto CSV souboru popsat škálovatelnost, efektivitu a zrychlení vašeho řešení pomocí generovaných gnuplot grafů (více v kapitole 12).

Jak pracovat s těmito programy a skripty je popsáno v následujících kapitolách.

5.1 Co je úkolem studenta?

- Student má povoleno editovat **pouze** zdrojový text v souboru `proj01.cpp`. Ten se bude následně odevzdávat do wisu a bude bodován.
- V souboru `proj01.cpp` je povoleno upravit, resp. doplnit implementaci **pouze** těchto dvou funkcí:

  `ParallelHeatDistributionNonOverlapped`

  `ParallelHeatDistributionOverlapped`

  Ostatní kód a funkce **musí** zůstat nedotčeny. V těchto dvou paralelních funkcích je navíc výrazně označena oblast, do které může student implementovat svůj kód. Kód vně této oblasti **musí** zůstat nedotčený.
- Kromě samotné implementace je úkolem změřit škálovatelnost, zrychlení a efektivitu vaší implementace na Salomonu. Ze získaných výsledků vytvořit pomocí předpřipravených skriptů čtyři grafy a zodpovědět dvě jednoduché otázky do texťáku. Podrobnosti jsou popsány v následujících kapitolách.

Suma sumárum, úkolem je upravit, resp. doplnit kód na označených místech dvou paralelních funkcí, vytvořit čtyři grafy z naměřených výsledků a do texťáku zodpovědět dvě jednoduché otázky.

6 Popis sekvenční verze

Sekvenční verze algoritmu je implementovaná ve funkci `SequentialHeatDistribution`, kterou naleznete v souboru `proj01.cpp`. **Tuto verzi neupravujte.** Sekvenční verze slouží jako ukázka, jak korektně spočítat celou simulaci bez použití paralelizace, a zároveň je to odrazový můstek pro bodovanou implementaci dvou paralelních verzí popsaných v následujících kapitolách. Pro lepší orientaci ve zdrojovém

[3] `https://wci.llnl.gov/simulation/computer-codes/visit/executables`
[4] `http://www.paraview.org/`

kódu jsou zde komentáře anotovány pomocí čísla nebo písmena v hranatých závorkách. Toto označení koresponduje s číslem, resp. písmenem odrážky v následujícím popisu. Výpočet sekvenční verze probíhá následovně:

1. Pokud je specifikován název výstupního hdf5 souboru parametrem `outputFileName`, k názvu se připojí suffix `_seq` (těsně před `.h5` koncovku) a dojde k vytvoření souboru. V opačném případě se průběh simulace neukládá.
2. Vytvoří se pomocné pole `tempArray`, které zaručí, že výpočet aktuální iterace bude vycházet pouze z hodnot té předchozí.
3. Hlavní pole `seqResult` i pomocné pole `tempArray` inicializujeme počátečními hodnotami (teplotami) z pole `initTemp` struktury `materialProperties`. Tyto hodnoty byly vytvořeny generátorem vlastností materiálu (složka `DataGenerator`).
4. Pro lepší přehlednost přejmenujeme pole `seqResult` na `newTemp` a `tempArray` na `oldTemp`. Na konci každé iterace budou v `oldTemp` o jednu iteraci starší výsledky oproti `newTemp` (`newTemp` tedy odpovídá času t a `oldTemp` času $t - 1$.
5. Uložíme si aktuální čas pro pozdější výpočet celkové doby běhu simulace.
6. Spustíme samotnou simulaci s `nIterations` kroky (specifikováno parametrem spuštění simulátoru `-n`). V každé iteraci se po řádcích projdou všechny body domény (kromě okrajů, ty zůstávají pevné) a na základě teploty a vlastností aktuálního bodu a jeho osmi sousedů (popsáno v kapitole 2) se jeho teplota aktualizuje. V každé iteraci se tedy provedou tyto podkroky:
   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.
7. 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`). Zde velmi vhodné použít jednu speciální OpenMP pragmu.
8. Aktuální iteraci zapíšeme do souboru. Abychom omezili množství zapisovaných dat, ukládáme pouze ty iterace, jejichž index je delitelný parametrem `diskWriteIntensity`, tedy pokud má `diskWriteIntensity` hodnotu 10, uloží se každá desátá iterace.
9. Prohodíme ukazatele polí `newTemp` a `oldTemp`.
10. Tisk průběhu simulace v procentech a průměrné teploty v prostředním sloupci domény.
11. Tisk výsledného času simulace, jedné iterace atd. V *batch* módu (parametr programu `-b`) se vše tiskne v CSV formátu se středníkem jako oddělovačem.
12. Pokud je celkový počet iterací lichý, jsou poslední výsledky v pomocném poli `tempArray` (dřívější `oldTemp`), proto je překopírujeme do `seqResult` (dřívější `newTemp`).

7 Implementace paralelní verze bez překrytí výpočtu a I/O (hodnoceno až 3 body)

Cílem projektu je paralelizovat sekvenční algoritmus. Projekt má dvě paralelní verze, bez překrytí výpočtu s I/O a s překrytím. Verze bez překrytí (poté, co ji vypracujete) paralelně počítá aktuální krok simulace. Jakmile má dojít k zápisu do souboru, jsou ostatní vlákna pozastavena a zápis je prováděn pouze jedním (master) vláknem. Oproti sekvenční verzi dosáhneme rozumného zrychlení pouze pro nízkou intenzitu zápisu a malé množství zapisovaných dat. Zápis je zde v podstatě sekvenční bottleneck (část α z Amdahlova zákona).

Implementace této verze se nachází ve funkci `ParallelHeatDistributionNonOverlapped` souboru `proj01.cpp`. Tělo funkce tvoří sekvenční verze algoritmu (s drobnými odlišnostmi jako např. koncovka názvu souboru). Vaším úkolem je doplnit do vymezeného úseku kódu OpenMP pragmy takovým způsobem, aby byl výsledek simulace shodný se sekvenční verzí, program pracoval bez chyb a zároveň bylo dosaženo maximálního možného zrychlení výpočtu s využitím daného počtu vláken. Doplňují se tedy **pouze** OpenMP pragmy v **označeném** úseku kódu této funkce (tj. uvnitř bloku `#pragma omp parallel`).

V tuto chvíli je sekvenční jádro algoritmu obaleno jednou `omp parallel` pragmou. To však v některých částech kódu způsobuje chyby díky nekontrolovaným přístupům vláken ke sdíleným zdrojům (tzv. *race conditions*) a tím pádem k nekorektnímu výpočtu. Je tedy nutné definovat, které proměnné jsou sdílené a které privátní (`shared`, `private`, `firstprivate`, apod.), které části kódu smí provádět pouze jedno vlákno (např. sériová verze knihovny hdf5 **není** thread-safe, proto jakékoliv pokusy o paralelizaci jejích operací vedou na nedefinované chování). Na jednom místě je velmi vhodné použít redukci (`pragma reduction`).

V této paralelní verzi **nestartujte** žádné další, popř. vnořené paralelní sekce, jednak je to další zbytečná režie navíc, jednak přijdete o body a hlavně to není vůbec třeba, stačí jedna `#pragma omp paralel` obalující celý výpočet (tak jak už je implementováno ve výchozím stavu, tuto pragmu lze samozřejmě doplnit o vhodné klauzule).

8 Implementace paralelní verze s překrytím výpočtu a I/O (hodnoceno až 5 body)

V předchozí kapitole byla popsána paralelní verze, kde všechna vlákna musí čekat na dokončení zápisu do souboru a až pak mohou pokračovat ve výpočtu. Když si uvědomíme, že se soubory uloženými na pevném disku můžeme pracovat rychlostí stovek megabajtů až několika gigabajtů za sekundu (v případě větších RAID polí a SSD), ale procesor generuje data rychlostí desítek až stovek gigabajtů za sekundu, je jasné, že I/O bude tvořit výrazný bottleneck. Proto je úkolem v této druhé paralelní verzi překrýt výpočet s probíhajícím diskovým zápisem, tj. zapisovat data na disk jedním vláknem a zároveň ostatními vlákny pokračovat ve výpočtu.

Implementace se nachází ve funkci `ParallelHeatDistributionOverlapped` souboru `proj01.cpp`. V této paralelní verzi bude nutné provést více změn do sekvenčního kódu oproti předchozí první paralelní verzi (kde se doplňovaly pouze pragmy), proto je sekvenční kostra tentokrát vynechána, abyste měli větší stupeň volnosti (ale i tato verze bude samozřejmě postavena na té sekvenční).

Hlavním úkolem (a změnou oproti první paralelní verzi) je tedy zajistit, aby se jedno vlákno staralo po celou dobu jenom o zápis na disk a ostatní vlákna řešila výpočet simulace. Je více způsobů, jak obecně rozdělit vlákna do více pracovních skupin, nejpoužívanější jsou sekce (`sections`) a úlohy (`tasks`). V tomto projektu si vyzkoušíte variantu využívající sekce.

Postup řešení s využitím sekcí je následující:

1. Vezměte hotový kód z paralelní verze bez překrytí (pozn.: je důrazně doporučeno implementovat a odladit nejdříve funkci bez překrytí a až poté se pouštět do překrývání) nacházející se v `omp parallel` oblasti a rozdělte ho na tři části - inicializační, výpočetní a I/O (funkce `StoreDataIntoFile`).

2. Po provedení inicializační části (nezapomeňte na *first touch strategy* kvůli NUMA architektuře) vytvořte dvě sekce, do jedné vložte výpočetní část a do druhé I/O. Zajistěte, aby do každé sekce vstoupilo jen **jedno** vlákno. Vlákno, které vstoupí do výpočetní sekce, nastartuje zbytek vláken (celkem max. 23 pro procesory nebo 239 pro akcelerátor). Je tedy povoleno použít jednu další

vnořenou úroveň `#pragma omp parallel`. V I/O sekci poběží vždy pouze jedno vlákno, protože sériová verze hdf5 knihovny není thread-safe.

3. Vytvořte pomocný buffer, do kterého bude výpočetní sekce kopírovat data určená pro zápis do souboru a I/O sekce bude z tohoto bufferu číst. Bez bufferu by byla data přepisována výpočetní sekcí v průběhu probíhajícího zápisu do souboru a dostávali bychom nekorektní výsledky.

4. Výpočetní a I/O sekce se potřebují vzájemně informovat, kdy je buffer plný a může dojít k zápisu do souboru, resp. kdy je zápis dokončen a je možné ho naplnit novými daty. To lze udělat pomocí bool proměnné reprezentující zámek, který bude jedna sekce zamykat a druhá odemykat. Aby byla hodnota zámku koherentní mezi všemi vlákny, je nutné před každým čtením a po každém zápisu do zámku provést `omp flush`. Čekání na odemčení/zamčení zámku implementujte pomocí jednoduchého aktivního čekání (v OpenMP nemá uspávání vláken smysl - jde nám o výkon!).

5. Po skončení výpočtu (poslední iterace) ve výpočetní sekci je nutné informovat I/O sekci, že další data již nepřijdou a může skončit. To můžete implementovat pomocí další bool proměnné, která v I/O sekci zruší aktivní čekání na naplnění bufferu a zároveň dá povel k vyskočení ze sekce.

9 Změření výkonnosti na Xeon Phi bez I/O (hodnoceno až 2 body)

Úkolem za 2b bude naměřit výkonnost řešení na Intel Xeon Phi akcelerátoru. Jelikož I/O je zde mnohem pomalejší než na procesoru (navíc zde není dostupný ramdisk) a tvoří tak výrazný bottleneck, bude se benchmarkovat pouze verze bez překrytí s vypnutým I/O (simulátor bude spouštěn bez `-o` přepínače). Jelikož nemá příliš smysl testovat škálování s nižšími počty vláken (na to máme klasické procesory), budeme se snažit vyždímat maximální výkon. K dispozici je 61 jader na jedné kartě, z nichž na jednom běží operační systém. Na každém jádře mohou běžet zároveň až 4 vlákna, takže celkově budeme benchmarkovat s 60, 120 a 240 vlákny. Zároveň bude omezena velikost testované domény na 256^2 a větší, při nižších velikostech už by nebyla využita všechna vlákna.

Spuštění na Phičku je popsáno v následující sekci.

10 Spuštění projektu

10.1 Spuštění na lokálním PC

Implementaci můžete testovat na libovolném stroji s Linuxem. Je nutné mít nainstalovaný C++ kompilátor, hdf5 knihovnu v sériové verzi a volitelně nástroj VisIt, který vizualizuje vypočtenou simulaci a může velmi usnadnit ladění aplikace. Postup je následující:

1. Ověřte, že máte nainstalován kompilátor jazyka C++ (stačí g++).

2. Stáhněte si aktuálně nejnovější hdf5 knihovnu z této adresy: `https://support.hdfgroup.org/HDF5/release/obtainsrc5110.html`

3. Tento archiv rozbalte a spusťte configure script:

   ```
   ./configure --prefix=CESTA
   ```

   kde CESTA by pro snažší linkování měla být v systémové proměnné `PATH`, nejlépe tedy `/usr/local/` nebo podobně. Následně přeložte knihovnu příkazem `make` a nainstaluje s `make install` (se `sudo` právy, pokud instalujete do systému).

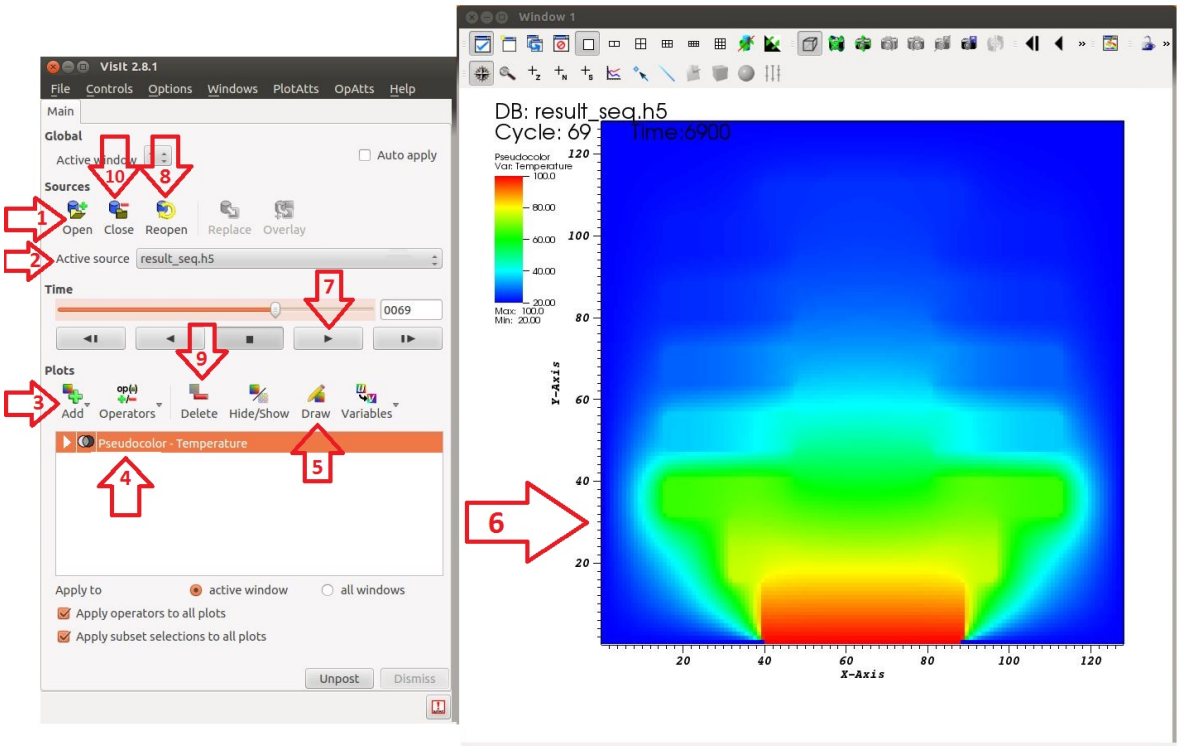
4. Z WISu si stáhněte a rozbalte zdrojové soubory projektu.

5. Makefily ve složkách `Sources` a `DataGenerator` jsou aktuálně nastaveny pro překlad na superpočítači Salomon s využitím Intel kompilátoru. Pro překlad na lokálním PC bude pravděpodobně nutné změnit cestu k hdf5 knihovně v proměnné `HDF5_DIR` (na `CESTA` z configure scriptu), použitý kompilátor (např. na `g++`), `-openmp` přepsat na `-fopenmp`, odstranit `-xhost` a nahradit za `-mavx`. Pokud pracujete ve VirtualBoxu, bude pravděpodobně nutné odstranit `-mavx`, popř. ho zkusit nahradit za `-msse`.

6. Spusťte příkaz `make`, jak ve složce `DataGenerator`, tak ve složce `Sources`. Měly by vzniknout spustitelné soubory `arc_generator` a `arc_proj01`.

7. Nyní zkusíme spustit nějakou jednodušší simulaci. Ve složce `DataGenerator` zadejte `make test`. Vznikne soubor `material.h5`. Následně zadejte ve složce `Sources` příkaz `make test`. Spustí se sériová verze simulace, jejímž výstupem je soubor `result_seq.h5`.

8. Výsledek simulace uložený v souboru `result_seq.h5` nyní zkusíme vizualizovat. Z domovské stránky[5] si stáhněte nástroj VisIt (následující popis odpovídá verzi 2.8.1). Po rozbalení spusťte spustitelný soubor s názvem `visit` nacházející se ve složce `bin`. Otevře se hlavní okno aplikace (obrázek 1, čísla v závorkách v následujícím popisu odpovídají číslům v šipkách na obrázku). Klikněte na tlačítko `Open` (1) v sekci `Sources` a vyberte soubor `result_seq.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 simulace, 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 1: VisIt - Popis grafického uživatelského rozhraní.

10.2 Spuštění na Salomonu

Pro vývoj a ladění aplikace bude pravděpodobně stačit lokální PC, popř. některý ze školních serverů, měření výsledků a časů ale provádějte na Salomonu, který má 1008 výpočetních uzlů, každý s 2×12 jádrovými procesory Intel Xeon architektury Haswell a zhruba polovina uzlů obsahuje po dvojici Intel Xeon Phi akcelerátory (architektura MIC - many integrated cores). Na procesorech je tedy možné škálovat až do 24 vláken a na akcelerátorech až do 120–240, podle aplikace. Více podrobností o Salomonu lze najít v dokumentaci[6].

[5] `https://wci.llnl.gov/simulation/computer-codes/visit/executables`
[6] `https://docs.it4i.cz/salomon`

Na Salomon se přihlásíte přes konzoli v Linuxu (nebo s Putty ve Windows) příkazem

```
ssh -i ssh_key.ppk login@salomon.it4i.cz
```

kde `login` je přihlasovací jméno, které dostanete na lístečku od Jirky Jaroše a `ssh_key.ppk` soubor obsahující ssh klíč pro šifrovanou komunukaci. Se Salomonem je možné komunikovat pouze s pomocí tohoto osobního ssh klíče, bude proto nutné stáhnout si ho z Extranetu ze sekce Training[7], kam se přihlásíte s údaji na lístečku. Více informací o přístupu ke clusteru lze najít zde [8].

Po provedení ssh příkazu a zadání hesla (*passphrase*) jste přihlášeni na login uzlu. Login uzel slouží pouze pro nenáročné operace, jako je přenos souborů přes internet, kompilace programů, atd. Nesmí se zde spouštět žádné dlouhotrvající náročné výpočty (pod pohrůžkou BANu od administrátorů Salomonu), na to slouží výpočetní uzly.

Zdrojové soubory projektu můžete přenést na Salomon např. protokolem SCP (ve Windows např. s WinSCP). Příkaz

```
scp -i ssh_key.ppk myfile login@salomon.it4i.cz:target_folder
```

přenese soubor `myfile` z lokálního PC na Salomon do složky `target_folder` vašeho domovského adresáře. Pro přenos celé složky je nutné doplnit parametr `-r`

```
scp -r -i ssh_key.ppk myfolder login@salomon.it4i.cz:target_folder
```

Opět místo loginu zadejte svoje přihlašovací jméno z lístečku. Alternativně je možné namountovat vzdálený disk ke svému lokálnímu PC. Na svém lokálním PC spusťte příkazy

```
mkdir myfolder
sshfs -o IdentityFile=/complete/path/to/ssh_key.ppk login@salomon.it4i.cz:. myfolder
```

a do složky `myfolder` se zobrazí celý váš home adresář ze Salomonu (ke klíči je tentorát nutné přidat kompletní cestu, tj. výstup příkazu `pwd`). Soubory poté můžete jednoduše kopírovat příkazem `cp` nebo přetahováním myši.

Nyní jsme připojeni a máme překopírovány zdrojov0 soubory projektu na Salomonu. Další krok je překlad. Pro překlad projektu je nutné natáhnout některé moduly, v našem případě Intel kompilátor a pro výpočet na procesoru i knihovnu hdf5, příkazem

```
module load intel/2017.00 HDF5/1.8.16-intel-2017.00
```

Nebo zkráceně:

```
ml intel/2017.00 HDF5/1.8.16-intel-2017.00
```

Nyní lze přeložit projekt příkazem `make` ve složkách `Sources` a `DataGenerator`. Překlad provádějte vždy na login uzlu. Z výpočetních uzlů může překlad trvat o něco déle kvůli ověřování licencí. Na login uzlu můžete zkusit jednoduchý testovací běh příkazem `make test`, popsaný v předchozí části pro lokální PC.

Nyní máme přeloženo a můžeme zkusit něco spustit. Pro testování paralelní verze je nutné se připojit na výpočetní uzel. Ze začátku je vhodné pracovat v interaktivním módu, kdy máme k dispozici konzoli (podobně jako když pracujete třeba na školním Merlinovi, na Salomonu ale máte k dispozici mnohem více výkonu a nemusíte se o něj po danou dobu s nikým dělit) a je možné ručně spouštět jednotlivé běhy s různými nastaveními. Z login uzlu se na výpočetní uzel připojíme příkazem

```
qsub -q qexp -A DD-17-10 -l select=1:ncpus=24 -I
```

Při připojení na uzel s Phi akcelerátorem použijeme příkaz (více o Phičku v následující kapitole)

```
qsub -q qexp -A DD-17-10 -l select=1:ncpus=24:accelerator=True:naccelerators=2 -I
```

[7] `https://extranet.it4i.cz/training`
[8] `https://docs.it4i.cz/salomon/shell-and-data-access/`

Připojení by mělo proběhnout v řádu jednotek až desítek sekund, někdy déle (uzly s akcelerátorem mohou někdy viset ve stavu ready delší dobu a je nutné počkat, než se uzel vyresetuje a připraví). Parametr `-q` specifikuje frontu (v tomto případě expresní `qexp`), `-A` specifikuje název interního it4i projektu, `-l` žádá o výpočetní zdroje, v tomto případě žádáme o 1 uzel a všech 24 jader tohoto uzlu. Nakonec `-I` značí interaktivní mód. Nyní máte po dobu jedné hodiny k dispozici všech 24 jader a 128GB operační paměti daného uzlu. Po uplynutí této doby vám bude plánovačem uzel odebrán (neuložená práce bude ztracena!) a je třeba znovu zažádat o uzel příkazem `qsub...` uvedeným výše a po připojení opět natáhnout požadované moduly. Pro krátké běhy (např. 1 iterace) v debug modu lze využít i login uzel.

Abychom zjistili, kolik úloh máme ve frontě (čekajících nebo spuštěných, i interaktivních), použijeme následující příkaz:

```
qstat -u $USER
```

Přestože Salomon umožňuje pohodlně pracovat vzdáleně s grafickým uživatelským rozhraním přes VNC[9], bude pro začátek jednodušší provádět vizualizaci na svém lokálním PC, kde máte VisIt nainstalovaný. Využijte namountování disku příkazem `sshfs` popsaným výše pro přístup k výstupním hdf5 souborům simulace spočítané na Salomonu. Proces vizualizace pak probíhá totožným způsobem popsaným v části práce na lokálním PC (viz kapitola 10.1).

Pro pokročilejší ladění aplikací lze využít DDT debugger a profiler nástroje Allinea Forge[10], více info na přednášce ARC z 6. března. Pro debugger je nutné překládat bez optimalizací s `-g -O0` a v případě profileru je naopak žádoucí maximální optimalizace s `-O3`.

10.3 Spuštění na Xeon Phi akcelerátoru

Pro překlad na Xeon Phi je nutné použít hdf5 knihovnu zkompilovanou přímo pro tento akcelerátor, která se nalézá ve složce `HDF5_phi`. Cesty v Makefilech jsou nastaveny, stačí zadat `make phi` ve složkách `Sources` a `DataGenerator`. Překlad musí probíhat na uzlu s akcelerátorem (nelze tedy použít login uzel) a nesmí být zároveň načtený modul s hdf5 knihovnou pro procesory. Pokud je modul načtený, lze jej zrušit příkazem

```
module unload HDF5
```

Dále se ubezpečte, že máte načtený modul `intel/2017.00` pomocí příkazu `ml`. Pokud tak není, tak jej načtěte (viz kapitola 10.2).

Pro připojení na uzel s Phi akcelerátorem použijeme příkaz

```
qsub -q qexp -A DD-17-10 -l select=1:ncpus=24:accelerator=True:naccelerators=2 -I
```

Po překladu se můžeme přepnout na akcelerátor (podrobnější popis, co je a co není možné na Phičku provádět, je v dokumentaci[11]) příkazem

```
ssh mic0
```

Od této chvíle budeme pracovat v tzv. nativním módu, tedy bez účasti procesoru, pouze se samotnou kartou (druhou možností je offload mód, kdy část kódu provádí procesor a část akcelerátor, ten ale v rámci tohoto projektu nevyužijeme). Konzolové prostředí je zde očesané na kost a nelze dělat prakticky nic jiného, než spouštět binárky, takže místo načítání modulů musíme ručně zadat cestu ke kompilátoru:

```
export LD_LIBRARY_PATH=/apps/all/icc/2017.0.098-GCC-5.4.0-2.26/
compilers_and_libraries_2017.0.098/linux/compiler/lib/mic:$LD_LIBRARY_PATH
```

Nyní lze spustit binárky `arc_proj01_phi` a `arc_generator_phi` ve složkách `Sources` a `DataGenerator`. Parametry, s jakými lze jednotlivé binární soubory spouštět, jsou rozepsané v následující kapitole 11. Popřípadě se inspirujte soubory `Makefile`.

[9] `https://docs.it4i.cz/general/accessing-the-clusters/graphical-user-interface/vnc/`
[10] `https://docs.it4i.cz/modules-salomon/#debugger`
[11] `https://docs.it4i.cz/salomon/software/intel-xeon-phi`

11 Nastavení a testování implementace

Projekt má dvě části, generátor vlastností materiálu v doméně (složka `DataGenerator`) a samotný simulátor šíření tepla (složka `Sources`). V kapitole 10 byly popsány kroky nutné k rozchození celého projektu, v této kapitole si popíšeme různá nastavení a možnosti spuštění a testování.

Generátor má následující parametry spuštění:

```
-o <string>  - název výstupního hdf5 souboru
-N <integer> - velikost domény (pouze mocniny dvou)
-H <float>   - teplota procesoru
-C <float>   - teplota chladiče a okolního vzduchu v počátečním stavu
-h           - zobrazí nápovědu
```

Hodnoty parametrů si můžete zvolit v rámci testování prakticky libovolně, je však doporučené mít dostatečný rozdíl mezi teplotou procesoru a teplotou chladiče a zároveň počítat na doméně rozumné velikosti mocniny dvou. Nebojte se větších velikostí domén, ale zohledněte přitom počet iterací. Abyste na dokončení výpočtu pak nečekali třeba měsíc. Výstupní soubory simulace by také mohly nabobtnat na stovky megabajtů až gigabajty, a to také po síti nikdo přenášet nechce.

Simulátor šíření tepla má tyto parametry:

```
Povinné argumenty:
  -m <0-2>     - mód simulace (mód 0 - sekvenční verze, mód 1 - paralelní verze
                 s nepřekrytým zápisem do souboru, mód 2 - paralelní verze s překrytým
                 zápisem do souboru)
  -n <integer> - počet iterací (časových kroků) simulace, více == delší běh
  -t <integer> - počet vláken
  -i <string>  - název souboru s vlastnostmi materiálu v doméně (generuje DataGenerator)

Volitelné argumenty:
  -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, takže více == méně I/O)
  -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
```

Počet iterací a hlavně hustotu zápisu do souboru se snažte držet na rozumné hodnotě tak, aby výpočet netrval více než minutu při použití 24 procesorových jader a zároveň jste si s danou velikostí výstupního souboru nezaplnili celý disk.

Výsledky obou paralelních verzí musí být shodné se sekvenční verzí (výsledky je myšlena průměrná teplota v prostředním sloupci a rozložení teplot v doméně). Shodu lze testovat parametrem `-v`, vypsání teplot celé domény na stdout provádí parametr `-d`.

Dejte pozor: některé chyby se projeví jen za určitého počtu vláken a velikosti domény.

12 Ověření škálovaní, zrychlení a efektivity

V rámci projektu je požadováno kromě korektního chování dosáhnout i rozumného zrychlení oproti sekvenční verzi. Parametry a nastavení aplikace, které se budou hodnotit, jsou uvedeny ve skriptech `run.pbs` a `run_phi.sh` ve složce `Scripts`. Po spuštění skriptu na Salomonu příkazem `qsub run.pbs` nebo `qsub run_phi.pbs` získate jako výstup soubory `benchmark.csv` a `benchmark_phi.csv`, kde jsou výsledné časy běhů včetně parametrů spuštění. Pozn.: Po zakomentování řádku `cd $PBS_O_WORKDIR` lze skript spustit v interaktivním módu jako `./run.pbs` na procesoru a `./run_phi.sh` na akcelerátoru.

Obsah csv souborů v tuto chvíli není příliš přehledný, proto s pomocí nástroje Gnuplot vytvoříme z výsledků pěkně vypadající grafy. Ve složce `Plots` je binárka `gnuplot` verze 5.0 a čtyři `.plot` skripty. Překlad spustíme jako

```
./gnuplot *.plot
```

Vzniknou čtyři grafy ve vektorovém formátu: `arc_scaling.svg`, `arc_speedup.svg`, `arc_efficiency.svg` a `arc_phi.svg`. Pokud máte problém otevřít svg formát, lze ve skriptech změnit `set term svg` na `set term png` a `set output 'file.svg'` na `set output 'file.png'`. Do wisu se ale bude odevzdávat vektorový formát!

Jak názvy napovídají, v grafech je vidět škálování, zrychlení a efektivita výpočtu v závislosti na počtu vláken.

13 Odevzdání

Odevzdávají se soubory **proj01.cpp**, čtyři vygenerované grafy zmíněné v předchozí sekci ve **vektorovém** svg formátu a textový soubor s názvem `login.txt`, kde login je váš školní login. V tomto texťáku zodpovězte v 5–10 větách následující dvě otázky:

1. Porovnejte průběh škálování na malých versus velkých doménách a vysvětlete, v čem a proč se liší.
2. Jaká je optimální hustota zápisu na disk (tedy kolik maximálně můžeme zapisovat dat, aby nebyl negativně ovlivněn celkový čas běhu) u překrývané verze běžící na procesoru s 24 vlákny na doméně o velikosti 1024^2? Do texťáku napište jak optimální parametr hustoty zápisu (použitý při `-w`), tak rychlost v MB/s zapisovaných dat. Hodnoty uveďte jak pro zápis na ramdisk (složka `/ramdisk/$PBS_JOBID`), tak pro zápis do `home` adresáře.

Tyto soubory zazipujte formátem zip, pojmenujte vaším loginem a odevzdejte do wisu. Pokud budete mít libovolný dotaz, připomínku nebo návrh, neváhejte se zastavit na konzultaci, případně přispějte do fóra.

14 Odkazy

- `https://computing.llnl.gov/tutorials/openMP/`
- `http://openmp.org/wp/`
- `http://docs.it4i.cz`