FAKULTA INFORMAČNÍCH TECHNOLOGIÍ, VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Praktické paralelní programování (PPP 2023)
Počítačové cvičení č. 8: Profilování a trasování

Jiří Jaroš (jarosjir@fit.vutbr.cz)

1 ÚVOD

Cílem tohoto cvičení je vyzkoušet si práci s profilovacími a trasovacími nástroji Score-P, Scalasca a Vampir. Seznam nejpoužívanějších příkazů je součástí tohoto archivu.
Jako testovací úlohu uvažujme Eratosthenovo síto na prvočísla. Implementace je provedena formou farmy. Farmář rozesílá balíky (interval ohraničený <min,max>) čísel na otestování a zpět přijímá seznam prvočísel. Přijatá prvočísla následně ukládá do souboru. Každý dělník nejprve zašle požadavek o balík dat, následně jej zpracuje a odešle odpověď. Jakmile farmář vyčerpá práci, kterou má, rozešle speciální zprávu o ukončení výpočtu.
Předtím než začnete, prostudujte zdrojový kód a ujistěte se, že chápete jak funguje. Další informace naleznete na adrese `https://www.vi-hps.org/projects/projects.html` nebo v prezentaci zde: `https://juser.fz-juelich.de/record/867978/files/tut115_VI-HPS.pdf`, slide 122 a dále

2 PŘIHLÁŠENÍ NA KAROLÍNU/BARBORU A ALOKACE VÝPOČETNÍHO UZLU

Dnešní cvičení lze vykonat pouze na clusterech Karolina a Barbora, jelikož merlin nemám nainstalovanou profilovací software. Chcete-li pracovat s OpenMP/GCC, použijte následující moduly:

```
ml Scalasca Vampir Cube
```

Chcete-li pracovat s IntelMPI, použijte tyto moduly:

```
ml Score-P
```

3 PŘÍKLAD 1. - ZÁKLADY PROFILOVÁNÍ SE SCORE-P + CUBE.

Cílem tohoto příkladu je udělat si základní přehled o profilování.

1. Nejprve je nutné program přeložit. Využijeme automatickou instrumentaci pomocí Score-P, buď pro OpenMPI/GCC nebo intel. Kompilaci není nutné dělat ručně, scripty pro spuštění obsahují i kompilaci.

   ```
   $ scorep-mpicxx  -O3 farm.cpp -o farm
   $ scorep-mpiicpc -O3 farm.cpp -o farm
   ```

2. Nyní si spustíme program se třemi různými konfiguracemi. Vždy budeme testovat 100M čísel na prvočísla, lišit se ale budeme velikostí balíku, který odesíláme na zpracování. Přirozeně, pro malé bloky práce bude dominovat režie MPI, pro rozumě velké bude práce dobře vyvážena, pro velké balíky již bude docházet k nerovnoměrné distribuci dat.

   Upravte soubor `run-1B.sh` nebo `run-1K.sh` a přidejte následující řádky na konec souboru pro OpenMPI/GCC nebo `run-1BI.sh` nebo `run-1KI.sh` pro intel.

   ```
   SCOREP_EXPERIMENT_DIRECTORY=prof_10k srun ./farm 100000 10
   SCOREP_EXPERIMENT_DIRECTORY=prof_100k srun ./farm 100000 100
   SCOREP_EXPERIMENT_DIRECTORY=prof_1000k srun ./farm 100000 1000
   ```

   Vložte script do fronty, použijte jeden, dva nebo čtyři uzly a výsledky pak porovnejte. Na Barboře volte scripy se přípono B, na Karolině s K

   ```
   $ sbatch run-1B.sh
   $ sbatch run-1K.sh

   $ sbatch run-1BI.sh
   $ sbatch run-1KI.sh
   ```

   Počet uzlů se specifikuje parametrem `-N count` uvnitř job scriptu

   ```
   #SBATCH -N 1
   ```

3. Vypište si základní profil aplikace pro jednotlivé konfigurace pomocí příkazu:

   ```
   $ scorep-score prof_100k/profile.cubex
   $ scorep-score prof_1000k/profile.cubex
   ```

   Kolik procent času trávíte v komunikaci a kolik ve výpočtu?

4. Vypište si základní flat profil pro jednotlivé konfigurace pomocí příkazu:

```
$ scorep-score -r prof_10k/profile.cubex
$ scorep-score -r -s totaltime prof_10k/profile.cubex
```

5. Pokud jste pracovali s Intelem, asi jste si všimli, že výpisech je spousta volání knihoven C++, které nás až tak nezajímají a celkově významně prodlužují profilování. Proto použijeme filtraci. Nejprve si vytvořte soubor `filtre.filt`, viz slide 29. V tomto filtru vyjměte všechny funkce, které obsahují řetězec `std` a `gnu`. Přidejte do scriptu řádek, který filtr vyexportujte:

   ```
   export SCOREP_FILTERING_FILE=filter.filt
   ```

   a proveďte měření znovu. Měli by jste vidět značnou redukci délky výpisu. Podívejte se, které MPI funkce zabírají nejvíce času pro různé velikosti balíků.

6. Nyní si prohlédněte tytéž výsledky v CUBE.

   ```
   $ cube prof_10k/profile.cubex
   $ cube prof_100k/profile.cubex
   $ cube prof_1000k/profile.cubex
   ```

7. Nalezněte funkci, které trvá nejdéle. Které ranky zde tráví většinu času?

8. Prozkoumejte funkci `worker()`. Který rank nepracuje? Prohlédněte si různá zobrazení ve třetím okně.

9. Kolik bylo celkem odesláno bytů? Který rank obdržel nejvíce zpráv?

10. Vyzkoušejte si zobrazení Absolute/Metric root percent/Own root percent.

11. Kolik času tráví každý dělník v MPI a kolik ve výpočtu?

4 Příklad 2. - Pokročilé profilování pomocí Score-P a Cube.

V tomto příkladu si osvojíme možnosti pokročilé profilace, např. použití `PAPI` counterů, profilování paměti, či I/O. Na závěr se podíváme na uživatelskou anotaci programů.

1. Otevřete soubor `run-2.sh` a přidejte do něj potřebné příkazy. Následně použijte jeden, dva nebo čtyři uzly a výsledky pak porovnejte.

2. Exportujte proměnou prostředí, která zajistí sběr PAPI counterů, viz Přednáška 10, slide 34. Použijte countery pro zjištění celkového počtu taktů a zpracovaných instrukcí `PAPI_TOT_INS,PAPI_TOT_CYC`. Seznam dostupných PAPI counterů získáte příkazem `papi_avail | grep Yes` nebo `papi_native_avail`.

   ```
   SCOREP_EXPERIMENT_DIRECTORY=papi_10k srun ./farm 100000 10
   SCOREP_EXPERIMENT_DIRECTORY=papi_100k srun ./farm 100000 100
   SCOREP_EXPERIMENT_DIRECTORY=papi_1000k srun ./farm 100000 1000
   ```

3. Proveďte znovu měření na a následně otevřete CUBE.
4. Podívejte se, kolik instrukcí zpracovaly jednotlivé ranky ve funkci parSieve.
5. Ve které části parSieve tráví farmář nejvíce taktů?
6. Ve které části parSieve tráví dělníci nejvíce taktů?
7. Zobrazte si zdrojový kód.
8. Definujte odvozenou metriku CPI (`metric::PAPI_TOT_CYC() / metric::PAPI_TOT_INS()`). Které funkce mají nejdelší/nejnáročnější instrukce?

Další částí bude ukázka profilování paměti a IO. Jelikož používáme `MPI_IO`, dojde k profilování automaticky.

1. Exportujte proměnnou prostředí, která zajistí profilování paměti (Slide 33).
2. Proveďte jedno měření s největším balíkem dat a prozkoumejte výsledky.

   ```
   SCOREP_EXPERIMENT_DIRECTORY=mem_io_1000k srun ./farm 100000 1000
   ```

3. Prozkoumejte nově naměřené metriky.

Na závěr si ukážeme manuální anotací kódu.

1. Anotujte obě smyčky ve funkci `sieve`, kde první slouží ke zpracování celého tasku a druhá k ověření jednotlivých čísel. Využijte návod na slide 35.
2. Znovu přeložte zdrojový kód

   ```
   $ scorep --user mpiicpc -O3 farm.cpp -o farm
   ```

3. Proveďte jedno měření s největším balíkem dat a prozkoumejte výsledky.

   ```
   SCOREP_EXPERIMENT_DIRECTORY=user_1000k srun ./farm 100000 1000
   ```

4. Ověřte v profilu, kolik času se v ní stráví.
5. Prověřte v Cube nové funkce.

5 Příklad 3. - Pokročilé profilování pomocí Scalasca a Cube.

Úkolem tohoto cvičení je otestovat možnosti nástavby Scalasca pro Score-P. V tuto chvíli není nutné program znovu překládat

1. Upravte soubor `run-3.sh`. Proveďte měření na 1 - 4 uzlech.
2. Proveďte jedno měření s největším balíkem dat a prozkoumejte výsledky.

```
SCOREP_EXPERIMENT_DIRECTORY=scan_1000k scan srun ./farm 100000 1000
```

3. Zobrazte si výsledky získané pomocí analýzy square. Pozor, nyní se zadává jméno složky

   ```
   square scan_1000k
   ```

4. Zjistěte, ve kterém typu komunikací je největší podíl přenesených bytů (P2P, Coll, RMA)
5. Prostudujte vypočtení imbalanci.
6. Rozklikněte položku Time a prozkoumejte čas v MPI a Výpočtu a jednotlivé části

6 Příklad 4. - Trasování s Vampir.

Úkolem posledního cvičení je trasovat běh programu a prozkoumat jeho chování.

1. Upravte soubor `run-4.sh`. Proveďte měření na 1 - 4 uzlech.
2. Proveďte jedno měření s největším balíkem dat a prozkoumejte výsledky.

   ```
   SCOREP_EXPERIMENT_DIRECTORY=trace_10k scan -t srun ./farm 100000 10
   SCOREP_EXPERIMENT_DIRECTORY=trace_100k scan -t srun ./farm 100000 100
   SCOREP_EXPERIMENT_DIRECTORY=trace_1000k scan -t srun ./farm 100000 1000
   ```

3. Pro větší počet procesů bude nutné zvýšit velikost paměti pro Score-P

   ```
   export SCOREP_TOTAL_MEMORY=3G
   ```

4. Nastartujte vampir a otevřete trasovací soubor

   ```
   $ vampir trace_10k/traces.otf2
   $ vampir trace_100k/traces.otf2
   $ vampir trace_1000k/traces.otf2
   ```

5. Prozkoumejte časovou osu, naučte se zoomovat k jednotlivé výměně dat a k prvočíslu.
6. Podívejte se, co provádí Farmář a co dělníci.
7. Obarvěte si region, který složí k výpočtu prvočísel (vaše instrumentace), MPI_IO, atd. - Function summary, barevný čtvereček. Zamyslete se, čím je způsobeno čekání v MPI_Probe.
8. Zobrazte si okno Processing line.
9. Zobrazte si okno Summary timeline.
10. Zobrazte si okno Counter data line, následně pravé tlačítko a Select metric a zobrazte si některou z metrik a prostudujte výstup, např Message Transfer time.

11. Zobrazte si okno Add Message summary a podívejte se na typické velikost zpráv.
12. Zobrazte si okno Process summary a podívejte se kolik času je ve výpočtu, MPI a dalších funkcích.
13. Na závěr si zobrazte funkci Add communcation matrix.