Universität Hamburg
Department Informatik
Wissenschaftliches Rechnen
Dr. J. Kunkel, O. Pola

Übungsblatt 4 zum Praktikum
Paralleles Programmieren
im SoSe 2017
Abgabe: 23.05.2017

1 OpenMP (120 P)

Der OpenMP hat sich als Standard zur Programmierung von gemeinsamen Speicher weit verbreitet (Shared-Memory Programmierung, Global-Address-Space GAS).

Online Kurs. Auch zu OpenMP gibt es wieder einen Onlinekurs unter `http://cluster.wr.informatik.uni-hamburg.de:8000/`. Hinweis: Die Seite ist nur innerhalb des Universitätsnetzwerks zugänglich. Sie können entweder das VPN der Universität oder einen SSH-Forward auf unser Cluster durchführen um Zugriff zu erhalten. Bearbeiten Sie den OpenMP-Kurs.

1.1 Kompilieren

Da Sie für die Entwicklung Ihres Projekts nicht die Onlineplattform nutzen werden ist es wichtig das Sie OpenMP Programme auf dem Cluster kompilieren können. Erstellen Sie ein Makefile[1] um eine der einfachen Aufgaben des Kurses kompilieren kann.

Abgabe:

`1-openmp.zip` Eines Ihrer Programme mit einem passendem Makefile

2 Oprofile (60 P)

Oprofile ist ein leistungsfähiger Kernel-Level Profiler von Linux, dieser unterbricht das Programm nach einer festgelegten Anzahl von Instruktionen und erfasst dann Ereignisse (Events) für den Code an der aktuellen Instruktion. Die Messergebnisse können später erfasst werden und den einzelnen Objekten bzw. Funktionen und auch direkt dem Source-Code und Assembler zugeordnet werden.
Hinweise zur Nutzung von `oprofile` auf der Kommandozeile:

- Bekanntmachung der notwendigen Befehle und Manuals: `module load oprofile`
- Aufzeichnung der Leistung, indem Sie ihre Anwendung mit Oprofile starten: `operf <PROGRAM>`
  - Beachten Sie: Die Ausführung von `operf` auf dem Cluster-Frontend ist nicht sinnvoll (und möglich).
  - Hierbei wird als Standard die Anzahl der Zyklen die der Prozessor für die Verarbeitung der Instruktionen benötigte aufgeichnet.
  - Verfügbare Hardware Counter werden angezeigt mittels `srun /opt/oprofile/1.0.0/bin/ophelp`
  - Diese können bspw. so genutzt werden: `srun operf -e L2_RQSTS:200000:0xff:1:1 <PROGRAM>`
- Auswertung von aufgezeichneter Leistung:
  - `opreport` zeigt die Verteilung auf die einzelnen Objekte (Images) an.
  - `opreport -l` verteilt die Messungen auf die Symbole (Funktionen) der Objekte.
  - `opannotate --source -t 10 <PROGRAM>` Threshold von 10% für die einzelnen Quellcode-Dateien.
- Hinweis: Vergessen Sie nicht Ihre Anwendung mit Debugging-Symbolen zu kompilieren!
- Hinweis: Bitte starten Sie ihr Programm so, dass es ca. 1 Sekunde läuft.
  - Sie können folgenden Shell-Code verwenden um bspw. 1 Million Argumente zu erzeugen: `./PROGRAM $(for x in $(seq 1 1000000) ; do echo -n "$x "; done)`

[1] Schauen Sie sich eine Einführung in Makefiles an, bspw. `https://de.wikipedia.org/wiki/Make`

2.1 OpenMP-Analyse

In dieser Aufgabe soll die Leistung Ihres OpenMP Programms aus Aufgabe 1 analysiert werden. Messen sie zunächst mit den Standard Events die Leistung ihres Programms, wählen sie dann mindestens ein anderes Event das interessant sein könnte.
Welcher Teil ihres Programmms braucht die meiste CPU Zeit? Wieviel Zeit wird im Kernel bzw. anderen Bibliotheken benötigt?

Abgabe:

`2-oprofile.txt` Geben Sie die Resultate Ihrer Messungen mit Erklärung des Outputs ab.

3 Leistungsanalyse mit VampirTrace & Vampir (60 P)

Mit Hilfe von VampirTrace können die Aktivitäten eines Programms in sogenannten Spurdaten aufgezeichnet und später analysiert werden. Einerseits können OpenMP und MPI Aufrufe als auch Funktionsaufrufe Ihres Programms erfasst werden.

Um VampirTrace nutzen zu können müssen Sie Ihr Programm neu kompilieren. Anstelle von `gcc` oder einem anderen Compiler verwenden Sie einen Wrapper für den Compiler: `vtcc -vt:inst compinst -vt:cc [mpicc|gcc]`.
Hierbei werden alle Funktionen Ihres Programms instrumentiert und können somit analysiert werden. Vorsicht, dies kann bei kleineren Funktionen den Overhead drastisch erhöhen. Anstelle von `compinst` können Sie auch `manual` verwenden, in dem Fall werden nur MPI-Aufrufe instrumentiert. Dies reduziert den Overhead deutlich.

Nach der Ausführung ihres Programms wurden Spurdaten mit der Erweiterung `otf` für den Programmlauf erstellt, welche Sie mittels `vampir` visualisieren können. Schalten Sie dafür X-Forwarding für SSH ein.

VampirTrace kennt viele Einstellungsmöglichkeiten über Umgebungsvariablen. Dies kann dazu genutzt werden um ein Profil für den Anwendungslauf zu erstellen. Dafür müssen Sie die Umgebungsvariable `VT_MODE` vor der Programmausführung auf STAT setzen. Beispielsweise in der Bash: `export VT_MODE=STAT`
Es ist ebenso möglich ein Profil aus einem Trace zu erstellen: `otfprofile -i a.otf`.
Mehr Information finden Sie in kompakter Form auf dem CheatSheet: `http://www.vampir.eu/public/files/pdf/vtcheatsheet_a4.pdf`

3.1 Analyse von OpenMP

Vermessen Sie Ihr OpenMP Programm aus Aufgabe 1, denken sie daran, dass es sich um ein OpenMP Programm handelt und Sie daher die oben genannten Aufrufe noch ein wenig anpassen müssen.

3.2 Analyse von MPI-Anwendungen

Vermessen Sie eines ihrer bisherigen MPI Programme für bspw. 8 Prozesse auf zwei Knoten mittels Vampir.

Abgabe:

`3-vampir-omp.pdf` Erstellen Sie ein PDF und fügen Sie Screenshots von ihrer Analyse mit Vampir ein und beschreiben kurz Ihre Erkenntnisse zu Ihrem OpenMP Programm.

`3-vampir-mpi.pdf` Erstellen Sie ein PDF und fügen Sie Screenshots von ihrer Analyse mit Vampir ein und beschreiben kurz Ihre Erkenntnisse zu Ihrem MPI Programm.

4 Likwid (Optional)

Likwid ist ein sehr leichtgewichtiges Werkzeug zur Analyse von verfügbaren Hardware-Countern der CPU. Somit lassen sich bspw. Flop/s oder Speicherdurchsatz der Anwendung ermitteln um Leistungsengpässe in der Programmierung aufzudecken.
Um Likwid nutzen zu können müssen Sie das entsprechende Modul laden.

```
$ module load likwid
```

In einem Batch-Skript wird Likwid so aufgerufen, dass es ein weiteres Programm mit den angegebenen Parametern aufruft.

```
likwid-perfctr -C 1-12 -g <GRUPPE> <PROGRAMM> <PARAMETER>
```

Hardwarebedingt kann bei einem Aufruf immer nur eine Gruppe gemessen werden, beginnen Sie typischerweise mit der Gruppe VIEW.
Auf unserem System gibt es folgende Gruppen:

```
BRANCH: Branch prediction miss rate/ratio
CACHE: Data cache miss rate/ratio
DATA: Load to store ratio
FLOPS_DP: Double Precision MFlops/s
FLOPS_SP: Single Precision MFlops/s
FLOPS_X87: X87 MFlops/s
L2: L2 cache bandwidth in MBytes/s
L2CACHE: L2 cache miss rate/ratio
L3: L3 cache bandwidth in MBytes/s
L3CACHE: L3 cache miss rate/ratio
MEM: Main memory bandwidth
TLB: TLB miss rate/ratio
VIEW: Overview of arithmetic and memory performance
```

Weitere Information finden Sie unter: `http://code.google.com/p/likwid/`
Verwenden Sie Likwid für die Leistungsanalyse eines beliebigen Programms.

Abgabe:

`4-likwid.txt` Dokumentieren Sie Ihre Analyse.

5 Analysen von OpenMP (Optional)

Schreiben Sie ein kleines Benchmark-Programm um etwas mehr über die Leistungsfähigkeit von OpenMP zu erfahren. Stoppen Sie hierfür die Zeit die eine Anzahl von Operationen z.B. Matrixadditionen benötigen und geben Sie den Durchsatz (Operationen pro Sekunde) bei Programmende aus (verwenden Sie hierfür den Befehl `clock_gettime()` um eine genaue Zeit zu ermitteln).
Schachteln Sie zwei Schleifen ineinander und lassen Sie die innere Schleife mittels der Direktive `parallel for` parallelisieren. Ein kleiner Rumpf z.b. Addition einer Zahl wird auch benötigt, damit der Compiler den Code nicht weg optimiert.
Bestimmen Sie mit ähnlichem Aufbau den Zusatzaufwand einmal für Atomare, für serielle Bereiche und für Reduktionen.
Starten Sie das Programm mehrfach mit 1-4 Threads, notieren Sie die Zeiten und erreichte Operationen pro Sekunde. Um Aussagekräftige Ergebnisse zu erreichen sollte das Programm nicht kürzer als 10 Sekunden laufen.
Wenn Sie zusätzlich den Aufwand für die einzelnen Operationen unter OpenMP berechnen wollen, so lassen Sie die selben beiden Schleifen ohne Direktiven laufen und messen Sie die Zeit, die dann benötigt wird. Vergleichen Sie die erzielte Zeit des sequentiellen Codes mit der Zeit des parallelen Codes für 1, 2 und 4 Threads und erstellen Sie Speedup Diagramme.

Abgabe:

`5-benchmark.zip` Ihr Programm mit einem passendem Makefile

6 Julia Mengen (Optional)

Verwenden Sie OpenMP zur Parallelisierung der Berechnung der Julia Mengen (siehe Aufgabenblatt 2).

Abgabe:

`6-julia-mengen-omp.zip` Ihr Programm mit einem passendem Makefile

7 Matrix-Matrix Multiplikation (Optional)

Verwenden Sie OpenMP zur Parallelisierung der Matrix-Matrix Multiplikation (siehe Aufgabenblatt 1 und 2).

Abgabe:

`7-matrix-omp.zip` Ihr Programm mit einem passendem Makefile

8 Leistungsvergleiche von MPI mit OpenMP (Optional)

Falls Sie eines der Programme (z.B. Julia Mengen oder Matrix-Matrix Multiplikation) sowohl mit OpenMP als auch mit MPI geschrieben haben, so vergleichen wir die Leistungsfähigkeit. Hierfür starten Sie die Programme auf nur einem Knoten mit 1-4 Prozessen bzw. Threads. Notieren Sie die erzielten Laufzeiten und Prozesszahlen in Sekunden für ein Programm ihrer Wahl, mit 3 Nachkommastellen in einer Textdatei. Das Programm sollte sinnvolle Zeiten ausgeben, d.h. für einen Prozess sollte die Berechnung schon eigene Minuten dauern. Erstellen Sie ein Speedup-Diagramm für 1-4 Threads bzw. Prozesse mit den Beispieleingaben.

Abgabe:

`8-mpi-vs-omp.pdf` Ihre Grafiken
`8-mpi-vs-omp.txt` Ihre Rohdaten und Beschreibungen