Einige Grundlagen zu OpenMP

Stephanie Friedhoff, Martin Lanser
Mathematisches Institut
Universität zu Köln

22. Juni 2016

Überblick

- Was ist OpenMP?
- Basics
- Das OpenMP fork-join-Modell
- Kompilieren und Ausführen
- Variablen: private und shared
- Parallele for-Schleifen
- Nützliche Links

Was ist OpenMP?

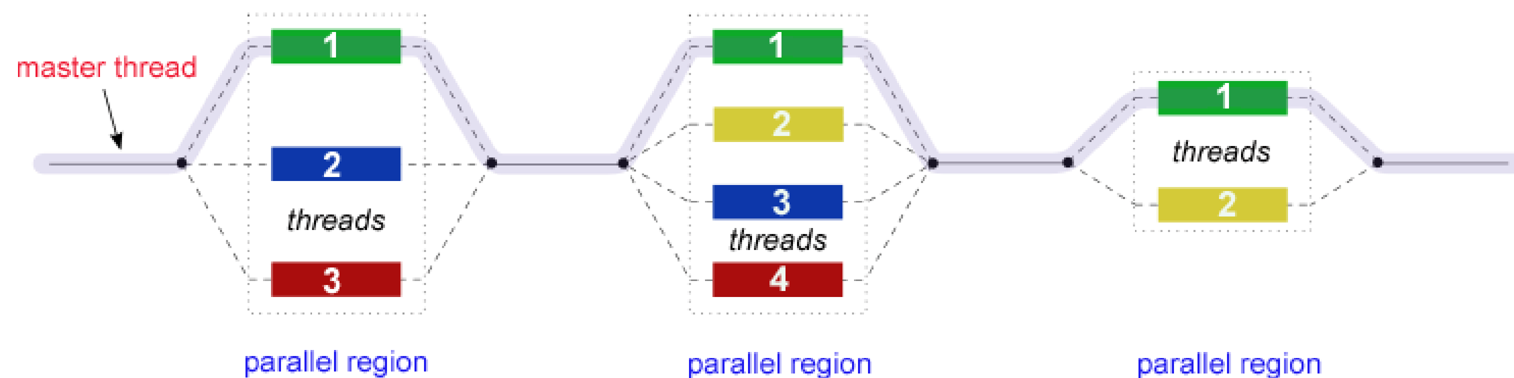
- OpenMP steht für **Open M**ulti-**P**rocessing
- von verschiedenen Herstellern seit 1997 zusammen entwickelt: Intel, GNU, IBM, Oracle etc.
- Schnittstelle zum Einsatz auf **Shared Memory** Systemen
- kann mit MPI zusammen eingesetzt werden (OpenMP auf Knoten, MPI zwischen den Knoten). Nennt sich **hybrid**.
- Aktuelle Version (seit 2015) ist OpenMP 4.5

Basics

- OpenMP parallelisiert Programme auf der Ebene von Schleifen, durch Verteilung der Arbeit auf verschiedene Threads / Aufgaben.
- Die Threads werden automatisch auf die zur Verfügung stehenden Prozessoren verteilt. Man kann jedoch mit einigen Tools (z. B. LIKWID) die Verteilung beeinflussen. Dies nennt sich **pinning**.
- OpenMP kommuniziert nur über von den Threads gemeinsam genutzten Speicher.
- Jeder Thread kann allerdings auch private Daten halten.
- OpenMP ist in den meisten gängigen C, C++ oder Fortran Compilern integriert.
- In den meisten Fällen laufen die Programme, obwohl die OpenMP Bibliothek nicht korrekt installiert oder gar nicht vorhanden ist, da die OpenMP Anweisungen in dem Fall als Kommentar ignoriert werden.

Das OpenMP fork-join-Modell

- OpenMP Programme starten seriell mit einem sogenannten Master-Thread.
- Im Quellcode sind verschiedene parallele Regionen markiert, in denen ein Team von parallelen Threads die Arbeit teilt.
- Am Ende der parallelen Region übernimmt wieder der Master-Thread.



Kompilieren und Ausführen

- Das Kompilieren funktioniert wie bei C, nur . . .
- . . . muss man mithilfe einer Compileroption dem Compiler mitteilen, dass man OpenMP nutzen möchte.
- Diese unterscheidet sich je nach Compiler:

  → gcc, g++, gfortran: **-fopenmp**
  → icc, icpc, ifort: **-openmp**
  → Liste: https://computing.llnl.gov/tutorials/openMP/#Compiling

- Außerdem muss die Bibliothek **omp.h** eingebunden werden.
- Die Ausführung ist wie bei einem seriellen C-Programm.
- OpenMP erhält die Anzahl der parallelen Prozesse über die interne Umgebungsvariable **OMP_NUM_THREADS** die man vor dem Ausführen des Programms mit folgendem Befehl festlegt:

```
export OMP_NUM_THREADS=np
```

Beispiel „Hello world“

```
#include <stdio.h>

// Binde OpenMP Bibliothek ein
#include <omp.h>

int main (int argc, char **argv)
{
    // Hier beginnt eine parallele OpenMP Umgebung
    // Es werden vom Master-Thread OMP_NUM_THREADS viele Threads
    // eroeffnet, die alle Hello world!! schreiben.
    #pragma omp parallel
    {
        printf ("\n Hello world!! \n\n");
    }
    return 0;
}
```

- In C startet eine parallele Region mit **#pragma omp parallel**
- Bei Beginn einer parallelen Region öffnet der Master-Thread (id=0) OMP_NUM_THREAD viele Threads.
- Jeder Thread erhält eine **id**, auslesbar mit **omp_get_thread_num()**
- Am Ende der parallelen Region befindet sich eine Barriere und alle Threads bis auf den Master-Thread werden eliminiert.

Daten: private and shared

- Daten können in OpenMP **private** oder **shared** sein.
- **shared** heißt: Im Speicher liegt **eine Kopie** des Datensatzes, alle Threads lesen und schreiben in diesen Datensatz.
  → **Achtung:** Lese- und Schreibkonflikte möglich!
  → Bei Initialisierung von Variablen (Integer, Double, Arrays etc.) sind diese per default **shared**.
  → **Einsatzgebiet**: Arbeiten mit oder an Arrays: Jeder Thread bearbeitet einen anderen Bereich.
- Werden hingegen Daten explizit als **private** deklariert erhält jeder Thread seine eigene Kopie (keine Konflikte!).
  → Zu Beginn einer parallelen Region wird auf jedem Thread eine private Kopie mit Null initialisiert, am Ende der parallelen Region wird die verbleibende Variable auf dem Master-Thread wieder zurück auf Null gesetzt.
  → **Einsatzgebiet:** Threads berechnen / halten unterschiedliche Ergebnisse (Norm eines Teilvektors, Thread-ID etc.)

Beispiel zu private und shared Daten

```c
#include <stdio.h>
#include <omp.h>

int main (int argc, char **argv)
{
    // Variablen sind hier erst einmal per default "shared"
    int nthreads, threadid;

    // Beginn parallele Region
    // nthreads und threadid werden jetzt private!
    #pragma omp parallel private(nthreads,threadid)
    {
        threadid = omp_get_thread_num();
        printf ("\n I'm Thread %d \n",threadid);

        // Ausgabe der Anzahl Threads nur auf Master-Thread
        if (threadid == 0)
        {
            nthreads = omp_get_num_threads();
            printf ("\n Number of Threads: %d \n",nthreads);
        }
    }
    return 0;
}
```

Parallele for-Schleife

- OpenMP zerlegt eine Schleife in **OMP_NUM_THREADS** viele, etwa gleich lange Teilschleifen und weist jedem Thread eine Teilschleife zur Abarbeitung zu.
- Die Länge der Schleife muss vorab und als **shared** Variable bekannt sein.
- Die Laufvariable hingegen sollte **private** sein.
- Es befindet sich eine **Barriere** am Ende von parallelen for-Loops.
- Befehl: **#pragma omp parallel for**

```
#include <stdio.h>
#include <stdlib.h>
#include <omp.h>

int main()
{
    int n   = 8;
    int a[] = {1,2,3,4,5,6,7,8};
    int b[] = {1,2,3,4,5,6,7,8};
    int c[] = {0,0,0,0,0,0,0,0};
    int i, num;
    int nthreads;

    // parallel for
    #pragma omp parallel for shared(n,a,b,c) private(i,num)
    for (i = 0; i < n; i++)
    {
        c[i] = a[i] + b[i];
    }

    #pragma omp parallel
    {
        num = omp_get_num_threads();
        printf("Num threads: %d\n",num);
    }
```

```
    #pragma omp master
    {
        printf("\n");
        for (i = 0; i < n; i++)
        {
            printf("%d ", c[i]);
        }
    }
    printf("\n");

    return 0;
}
```

- Der Befehl **#pragma omp master** eröffnet einen Bereich, den nur der Master-Thread ausführt.

Nützliche Links

- Offizielle Website

  → `http://openmp.org`

- GNU Project GOMP (Implementierung von OpenMP in der GNU Compiler Collection)

  → `https://gcc.gnu.org/projects/gomp/`