Parallelisierung mit OpenMP

Niklas Wittmer

Seminar "Effiziente Programmierung"
Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

09. November 2017

Ablauf

Motivation

Parallelelisierung von sequentiellen Programmen

- Automatische Parallelisierung
  - Was kann/darf parallelisiert werden?
- MPI
  - Restrukturierung des Codes nötig
- Pthreads
  - Sehr aufwendig

Was ist OpenMP?

- Open Multi-Processing
- Shared-Memory-Parallelisierung
- Parallelisierung durch Compiler-Direktiven
- Fortran, C/C++,
- geeignet für SIMD-Systeme

Was ist OpenMP *nicht*?

- allmächtig
- ein Ersatz für andere Lösungen

Threading Model

- Fork Join

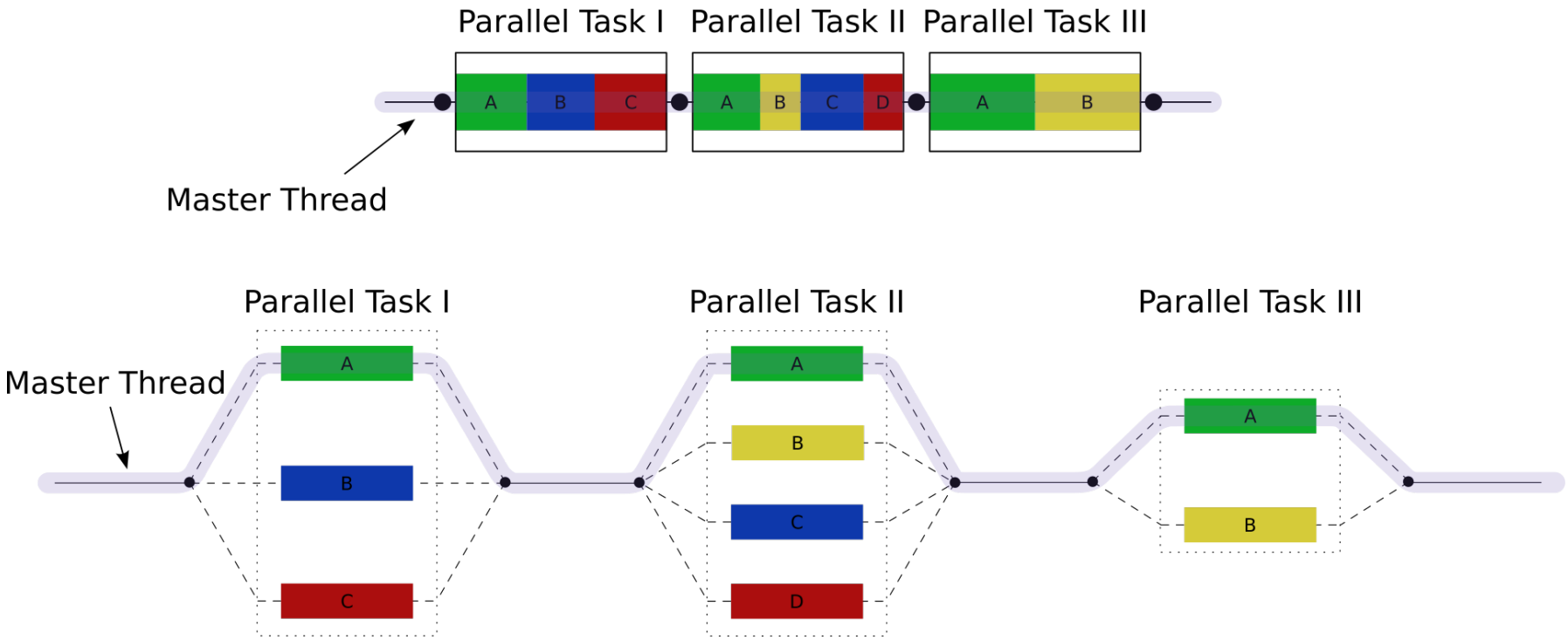

Abbildung: Schematische Darstellung des Fork Join Model (3)

Bestandteile

- Direktiven
  - Konstrukte (engl. *constructs*)
  - Klauseln (engl. *clauses*)
- Laufzeitroutinen
- Umgebungsvariablen

Syntax in C

```
1 #pragma omp directive-name [clause [clause ...]]
2 {
3   . . .
4 }
```

Parallel regions

Definiere eine *Parallel Region* mit 4 Threads

```
1 void main() {
2   #pragma omp parallel num_threads(4)
3   {
4       printf("hello world!\n");
5   }
6 }
```

Ausgabe

```
1 hello world!
2 hello world!
3 hello world!
4 hello world!
```

Work Sharing

Work Sharing einer *for*-Schleife

```
1  void main() {
2    int i;
3    #pragma omp parallel
4    {
5      #pragma omp for
6      for(i=0;i<4;++i) {
7          printf("hello world!(%d)\n", i);
8      }
9    }
10 }
```

Ausgabe

```
1 hello world!(0)
2 hello world!(1)
3 hello world!(2)
4 hello world!(3)
```

Oder aber

```
1 hello world!(2)
2 hello world!(0)
3 hello world!(1)
4 hello world!(3)
```

Work Sharing II

```
1 #pragma omp parallel for schedule(kind [, chunk size])
```

- static
- dynamic (Overhead!)
- guided
- auto

Work Sharing III

Sections

```
1  #pragma omp parallel
2  {
3    #pragma omp sections
4    {
5      #pragma omp section
6      { . . .}
7      #pragma omp section
8      { . . . }
9    }
10 }
```

Synchronisation I

`no wait`-Klausel

```
1  int c = 0, int i;
2  #pragma omp parallel
3  {
4    #pragma omp for nowait
5      for(i=0;i<4;++i) {
6      c++;
7    }
8    printf("c ohne Barrier: %d", c);
9    #pragma omp for
10   for(i=0;i<4;++i) {
11     c++;
12   } // implizites barrier
13   printf("c mit Barrier: %d", c);
14 }
```

Synchronisation II

```
1     int i=0;
2     #pragma omp parallel\
       num_threads(1000)
3     {
4     ++i;
5     }
6
```

`i` hat nach Ausführung einen Wert zwischen 1 und 1000.

Definiere kritischen Abschnitt mit `critical`

```
1     int i=0;
2     #pragma omp parallel\
       num_threads(1000)
3     {
4     #pragma omp critical
5     ++i;
6     }
7
```

Alternativ: `#pragma omp atomic`

Synchronisation III

- `master`
  Folgender Block wird nur vom Master-Thread ausgeführt.
  Kein `barrier` am Ende.
- `single`
  Folgender Block wird von genau einem Thread ausgeführt.
  Andere Threads warten.

Data Sharing I

shared, private Deklarationen

```
1 void main() {
2   int i,j=0;
3   #pragma omp parallel private(i) shared(j) num_threads(2)
4   {
5       i=0;
6       printf("i: %d\n", ++i);
7       printf("j: %d\n", ++j);
8   }
9 }
```

Ausgabe:

```
1 i: 1
2 i: 1
3 j: 1
4 j: 2
```

Default: alle Variablen *shared*, außer Zählvariablen in *for*-Blöcken (Vorsicht bei Verschachtelungen!)

Data Sharing II

firstprivate

```
1  int i = 0;
2  #pragma omp parallel firstprivate(i) num_threads(2)
3  {
4    printf("%d\n",++i);
5  }
6  printf("%d\n", i);
```

Ausgabe

```
1 1
2 1
3 0
```

Data Sharing III

lastprivate

```
1 int i = 0;
2 #pragma omp parallel for lastprivate(i)
3 for(i=0;i<10;++i)
4 {
5   printf("%d",i);
6 }
7 printf("\n%d", i);
```

Ausgabe

```
1 0134298765
2 10
```

Weitere Data-Sharing-Attribute

- `default (private|shared|none)`
- `reduction(op:var)`
- `threadprivate`

Weitere Konstruke

- SIMD
  - Ermöglicht Ausführung von Loops mittels SIMD-Instruktionen
- *Cancellation*
  - Vorzeitiges Verlassen paralleler Regionen
  - `exit, return` etc sind nicht zulässig
- Device
  - Tasks auf andere Devices auslagern
  - kann mit SIMD-Konstrukten verknüpft werden
- Bedingte Parallelisierung mit `IF`

Laufzeitroutinen

- `#include <omp.h>`
- `omp_get_thread_num()`
- `omp_set_num_threads(),omp_get_num_threads()`
- `omp_in_parallel()`
- `omp_set_schedule(),omp_get_schedule()`

Umgebungsvariablen

- `setenv OMP_NUM_THREADS=8`
- `export OMP_SCHEDULE="dynamic"`
- `OMP_THREAD_LIMIT=10 ./executable`
- `OMP_CANCELLATION`

Mehr Features in der OpenMP API Referenz (6) Oder (5) für eine kompakte Übersicht

Vorraussetzungen

- kompatibler Compiler; clang, gcc ...
- OpenMP-Kompilierung muss aktiviert werden
- evtl. Einbindung der Laufzeitbibliothek
- Beispiel gcc: `gcc -fopenmp code.c`

Original

Parallel Region mit zwei gemeinsamen und einer privaten Variable

```
1 int main(void) {
2   int a,b,c;
3   #pragma omp parallel private(c)
4   {
5     do_something(a,b,c);
6   }
7   return 0;
8 }
```

Übersetzung

```
1  static void __omp_func_0(void **__ompc_args) {
2    int *_pp_b,*_pp_a, _p_c;
3
4    _pp_b=(int *)(*__ompc_args);
5    _pp_a=(int *)(*(__ompc_args+1));
6    do_something(*_pp_a,*_pp_b,_p_c);
7  }
8
9  int _ompc_main(void) {
10   int a,b,c;
11   void *__ompc_argv[2];
12
13   *(__ompc_argv)=(void *)(&b);
14   *(__ompc_argv+1)=(void *)(&a);
15
16   _ompc_do_parallel(__ompc_func_0, __ompc_argv);
17   . . .
18 }
```

„Unified Code“

Direktiven werden ignoriert, wenn Compiler-Flag nicht gesetzt.
Verwendung von Laufzeitroutinen

```
1  #ifdef _OPENMP
2    #include <omp.h>
3  #else
4    #define omp_get_thread_num() 0
5  #endif
6  int main() {
7    #pragma omp parallel num_threads(2)
8    {
9      printf("%d\n", omp_get_thread_num());
10   }
11   return 0;
12 }
```

Parallel

```
1 0
2 1
```

Sequentiell

```
1 0
```

Präzedenz

omp-test.c

```
1 omp_set_num_threads(2)
2 #pragma omp parallel if(n > 2) num_threads(4)
3 {
4     printf("hello world!");
5 }
```

Kompilierung und Ausführung

```
1 $ gcc -fopenmp -o omp-test omp-test.c
2 $ OMP_NUM_THREADS=8 ./omp-test
```

Ausgabe?

```
1 hello world!
2 hello world!
3 hello world!
4 hello world!
```

Für `n>2`

Präzedenz

1. `IF`-Klausel
2. `num_threads()`-Klausel
3. Laufzeitroutine `omp_set_num_threads()`
4. Umgebungsvaribale `OMP_NUM_THREADS`
5. Defaultwert, meist Zahl der verfügbaren CPUs

Berücksichtigungen

- Art des Speicherzugriffs
  - Cache Lines
  - False Sharing
- ungleich verteilt Last
- Overhead
  - Sequentieller Overhead
  - Overhead durch Parallelisierung
  - Synchronisations-Overhead

Best Practices

- Barrier-Nutzung optimieren
- Große *Critical Regions* vermeiden
- Möglichst große *Parallel Regions*
- *Parallel Regions* innerhalb von Loops vermeiden
- Last gleichmäßig verteilen
- *False Sharing* vermeiden
- Nutzung von *private*- und *shared*-Konstrukten

häufige Fehler

- Race Conditions
  - sind Bibliotheken thread safe?
  - `no wait`-Konstrukt
  - implizite Barriers
- Annahmen beim Scheduling
- Fehlerhafte Annahmen über Scope von Variablen
  - hilfreich: explizites Deklarieren `default(none)`-Klausel

Parallelisierung einer Matrizenmultiplikation

Performance-Analyse

Version	#CPUs	CPU-Zeit (s)	Verstrichene Zeit (s)	Speedup	Effizienz (%)
Sequentiell	1	18,992	18,997	1	100
Parallel	1	20.492	20,528	0,93	93
	2	21,672	11,124	1,71	85
	4	21,752	5,906	3,22	80
	8	22,392	3,109	6,11	76
	12	20,682	2,546	7,46	62

Vor- und Nachteile

Vorteile

- hoher Abstraktionsgrad
- Code leicht parallelisierbar
- Unified Code

Nachteile

- Speedup von Architektur abhängig
- Erreichen hoher Effizienz mitunter schwierig
- u.U. sogar Verschlechterung der Laufzeit

Zusammenfassung

- Parallelisierung durch Compiler-Direktiven
- Unified Code
- Einfacher Einstieg
- Für Shared-Memory-Systeme
- Keine Low-Level-Kontrolle über Threads
- Kein Ersatz für andere Werkzeuge

Quellen I

(1) Rohit Chandra, Hrsg. *Parallel programming in OpenMP*. San Francisco, CA: Morgan Kaufmann Publishers, 2001. 230 S. ISBN: 978-1-55860-671-5.

(2) Barbara Chapman, Gabriele Jost und Ruud van der Pas. *Using OpenMP: portable shared memory parallel programming*. Scientific and engineering computation. OCLC: ocn145944336. Cambridge, Mass: MIT Press, 2008. 353 S. ISBN: 978-0-262-53302-7 978-0-262-03377-0.

(3) *Fork join - OpenMP - Wikipedia*. URL: `https://en.wikipedia.org/wiki/OpenMP#/media/File:Fork_join.svg` (besucht am 30. 10. 2017).

Quellen II

(4) *OpenMP*. In: *Wikipedia*. Page Version ID: 804095087. 6. Okt. 2017. URL: https://en.wikipedia.org/w/index.php?title=OpenMP&oldid=804095087 (besucht am 30. 10. 2017).

(5) *OpenMP 4.5 API C/C++ Syntax Reference Guide*. URL: http://www.openmp.org/wp-content/uploads/OpenMP-4.5-1115-CPP-web.pdf (besucht am 29. 10. 2017).

(6) *OpenMP 4.5 API Complete Specification*. Nov. 2015. URL: http://www.openmp.org/wp-content/uploads/openmp-4.5.pdf (besucht am 29. 10. 2017).

(7) *openmp - GCC Wiki*. URL: https://gcc.gnu.org/wiki/openmp (besucht am 31. 10. 2017).

Quellen III

(8) *OpenMP Technichal Report 4: OpenMP 5.0 Preview 1*. 11. Okt. 2016. URL: http://www.openmp.org/wp-content/uploads/openmp-tr4.pdf (besucht am 31. 10. 2017).

Neuerungen

- Vorraussichtlich 2018
- Verbesserung und Erweiterung bestehender Features
- OMP Tool Interface
  Soll Entwicklung von portablen Tools für Monitoring und Performance-Analyse von OpenMP-Programmen ermöglichen.
  •Thread States
  •Tracing
- Details in *OpenMP 5.0 Preview* (8)