C Programmierkurs

12. Stunde: Multithreading

Johannes Hayeß & Mirko Seibt

24. Januar 2019

Technische Universität Dresden

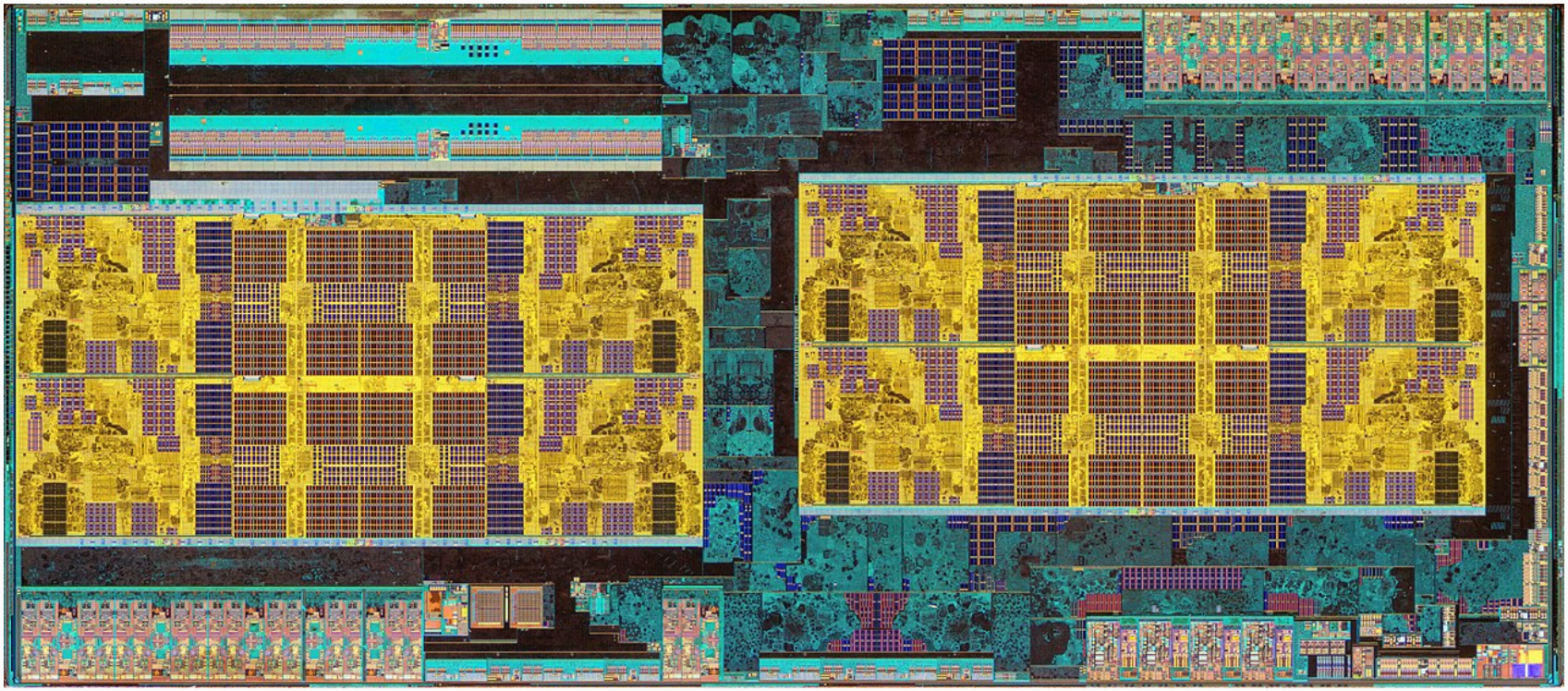
Multithreading - Hardware

Nutzung von Mehrprozessorsystemen

Heutzutage haben Computer zunehmend mehr Prozessorkerne - diese können wir für unsere Programme nutzen um diese zeiteffizienter zu machen!

- Abarbeitung von Programmteilen in unterschiedlichen Threads
  - Threads werden vom BS den Prozessorkernen zugewiesen
  - Threads müssen nicht in der Leistung gleichwertig sein
    - Simultaneous Multithreading
- Verwenden von SIMD-Befehlssatzerweiterungen (tut der Compiler evtl. bereits automatisch)

Moderne Prozessoren - Beispiel AMD Ryzen



Multithreading - C11

threads.h

Bibliothek: **threads.h**
Funktionen:

- **thrd_create** - neuen Thread erstellen
- **thrd_exit** - Thread beenden
- **thrd_join** - Warten das ein Thread beendet wird
- **thrd_detach** - Auf diesen Thread kann nicht gewartet werden
- ...

Datentyp:

- **thrd_start_t - int(*)(void*)**
  Typ der aufrufbaren Funktionen innerhalb eines Threads

thrd_create

Erstellt einen neuen Thread:

```
1  int thrd_create(thrd_t *thr, thrd_start_t func, void *arg);
```

Parameter:

- **thrd_t *thr** - Ort an dem die Thread-ID gespeichert wird
- **thrd_start_t func** - Funktion die ausgeführt wird
- **void *arg** - Argument für die Funktion

thrd_exit

Beendet den laufenden Thread:

```
1  _Noreturn void thrd_exit(int res);
```

Parameter:

- **res** - Ergebnis welches zurückgeben werden soll

thrd_join

Blockt den aktuellen Thread bis der Thread mit der ID **thr** beendet wurde.

```
1 int thrd_join(thrd_t thr, int *res);
```

Parameter:

- **thrd_t thr** - Thread-ID auf die gewartet wird
- **int *res** - Ort an dem der Ergebniscode gespeichert werden soll

Rückgabewert:

- **thrd_success**
- **thrd_error**

thrd_detach

Markiert einen Thread, sodass niemand auf ihn wartet.

```
1 int thrd_detach(thrd_t thr);
2 thrd_detach(thrd_current()); // Beispiel
```

Parameter:

- **thrd_t thr** - Thread-ID
- **thrd_current()** - Gibt uns die ID des aktuellen Threads zurück

Rückgabewert:

- **thrd_success**
- **thrd_error**

VORSICHT!

Threads laufen auf gemeinsamen Speicher!
Greifen mehrere Threads auf die selben Daten zu, kann es zu Problem kommen (z.B. Race Conditions)

Dieses Thema wird ausführlich in der Lehrveranstaltung Betriebssysteme und Sicherheit, Rechnerarchitektur und zum Teil in Datenbanken behandelt.

Für mehr Informationen siehe Kapitel 3.19 Threads in unserer Literaturempfehlung

Multithreading - OpenMP

OpenMP - Konzept

OpenMP ist eine Erweiterung des C-Standards für die vereinfachte Nutzung von Multithreading.

For-Schleifen werden als Basis zur Parallelisierung verwendet.

Kompilieren von C-OpenMP-Code:

```
$ gcc main.c -fopenmp
```

Nachteil: Bindet einen an einen Compiler der OpenMP (evtl. auch nur teilweise) unterstützt

Beispiel - Prallel For

```
1  #include <math.h>
2  int main() {
3    double sin_table[256] = {};
4
5    #pragma omp parallel for
6    for(int n=0; n<size; ++n) {
7      sin_table[n] = sin(2 * M_PI * n / 256);
8    }
9
10   // the table is now initialized
11 }
```

Parallelisiert Ausführung über mehrere Threads

Beispiel - SIMD

```
1  #include <math.h>
2  int main() {
3    double sin_table[256] = {};
4
5    #pragma omp simd
6    for(int n=0; n<size; ++n) {
7      sin_table[n] = sin(2 * M_PI * n / 256);
8    }
9
10   // the table is now initialized
11 }
```

Parallelisiert Ausführung über Vektorverarbeitung

Beispiel - Parallel For & SIMD

```
1  #include <math.h>
2  int main() {
3    double sin_table[256] = {};
4
5    #pragma omp parallel for simd
6    for(int n=0; n<size; ++n) {
7      sin_table[n] = sin(2 * M_PI * n / 256);
8    }
9
10   // the table is now initialized
11 }
```

Kombiniert beide Verfahren

OpenMP - Ordering

`ordered`

Erhält die Reihenfolge in der Ausführung designierter Code-Blöcke bei.

```
1  #pragma omp parallel for ordered
2  for(size_t n=0; n<100; ++n)
3  {
4    encode(packet[n]);
5
6    #pragma omp ordered
7    send(packet[n]);
8  }
```

OpenMP - Scheduling & Verschachtlung

`schedule(<type>)`

static (Standard) Legt Arbeitspensum jedes Threads beim starten des Loops fest

dynamic Jeder Thread arbeitet einen Loopdurchgang ab und fragt danach nach dem nächsten

Dynamic eignet sich wenn der Arbeitsaufwand pro Iteration variiert, oder zusammen mit ordered.

`collapse(<depth>)`

In verschachtelten For-Loops wendet dieser Befehl die OpenMP-Einstellungen auch auf die inneren Loops bis auf die angegebene Tiefe an.

OpenMP - Welches ist besser?

```
#pragma omp parallel for simd collapse(3)
for (size_t i = 0; i < 3; ++i) {
  for (size_t j = 0; j < 3; ++j) {
    for (size_t k = 0; k < 3; ++k) {
      matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
    }
  }
}
```

```
#pragma omp parallel for simd collapse(3) schedule(dynamic)
for (size_t i = 0; i < 3; ++i) {
  for (size_t j = 0; j < 3; ++j) {
    for (size_t k = 0; k < 3; ++k) {
      matrix_c[i][j] += matrix_a[i][k] * matrix_b[k][j];
    }
  }
}
```

Bildrechte

Fritzchens Fritz. *AMD Ryzen 3 1200 Zen core etched*. Lizenz: CC0 1.0. URL: `https://www.flickr.com/photos/130561288@N04/35620962953`.

Textreferenzen

Jens Gustedt. *Modern C*. Lizenz: CC BY-NC-ND 4.0. URL: https://creativecommons.org/licenses/by-nc-nd/4.0/.

Joel Yliluoma. *Guide into OpenMP: Easy multithreading programming for C++*. URL: https://bisqwit.iki.fi/story/howto/openmp/.