Primitive Zeitmessung mit `time`

- Eingabe mit `time cmd`, z.B. `time ./matr` oder `time ls -a`
- `time`-Ausgabe der Form:

```
real    0m.024s ⟵ Abstand zwischen call und finish
user    0m.016s ⟵ CPU-Zeit im User-Mode
sys     0m.016s ⟵ CPU-Zeit im Kernel-Mode
```

- Nachteil: keine selektive Messung möglich

  $\Longrightarrow$ `time` ungeeignet für Performancemessung

Zeitmessung im Code

- Grundidee: Messpunkt `end` - Messpunkt `start` = Dauer
- Messung der Dauer der Berechnung
  $\Longrightarrow$ keine unbegründeten I/O Funktionen im Messbereich
- Abstand zwischen Messpunkten mindestens eine Sekunde
  $\Longrightarrow$ genügend Workload durch z.B. Schleifen
- mindestens Optimierungsstufe 2

Messen eines Zeitpunktes

- Genauigkeit der Messung abhängig von Genauigkeit der Zeitpunkte
- Linearität der Uhrzeit
- `int clock_gettime(clockid_t clk_id, struct timespec *tp)`
  - `@return`: 0 falls erfolgreich, -1 sonst
  - `clk_id`: `clockid_t CLOCK_MONOTONIC`
  - `*tp`: `struct timespec{time_t tv_sec; long tv_nsec;}`
  - $\Longrightarrow$ Sekunden mit: `double sec = tv_sec + 1e-9 * tv_nsec`

Codebeispiel

```
1     #include <time.h>
2     ...
3     struct timespec start;
4     clock_gettime(CLOCK_MONOTONIC, &start);
5     for(int i = 0; i < iterations; ++i)
6         foo();
7     struct timespec end;
8     clock_gettime(CLOCK_MONOTONIC, &end);
9     double time = end.tv_sec - start.tv_sec + 1e-9 *
        (end.tv_nsec - start.tv_nsec);
10    double avg_time = time/iterations;
```

Quizreihe: Zeit (1/2)

Welche Nachteile hat die Nutzung der `CLOCK_REALTIME`?

- ☐ es gibt da diesen Sonntag, an dem meine Berechnung eine Stunde länger braucht...
- ☐ Messung kann durch Änderungen an der Systemuhr stark verfälscht werden
- ☐ keine, die vorgeschriebene Nutzung von `CLOCK_MONOTONIC` ist reine Schikane

Quizreihe: Zeit (2/2)

Welche Nachteile hat die Nutzung der CPU-Zeit (`CLOCK_PROCESS/THREAD_CPUTIME_ID`)?

- ☐ die CPU-Zeit ist nicht identisch mit der real benötigten Zeit
- ☐ die CPU-Zeit ist im Vergleich zur `CLOCK_MONOTONIC` viel ungenauer
- ☐ keine, die vorgeschriebene Nutzung von `CLOCK_MONOTONIC` ist immer noch reine Schikane

Quizreihe: Code-Snippets (1/6)

```
1     #include <stdio.h>
2     #include <time.h>
3
4     int main(int argc, char **argv){
5         struct timespec start;
6         clock_gettime(CLOCK_MONOTONIC, &start);
7         foo();
8         struct timespec end;
9         clock_gettime(CLOCK_MONOTONIC, &end);
10        double time = end.tv_sec - start.tv_sec +
              end.tv_nsec - start.tv_nsec;
11        printf("done after %f seconds", time);
12        return 0;
13    }
```

Quizreihe: Code-Snippets (1/6)

- [ ] Fehler beim Umfang der Zeitmessung
- [ ] Fehler bei der Berechnung der Differenz
- [ ] Fehler bei der Ausgabe des Ergebnisses
- [ ] kein Fehler

Quizreihe: Code-Snippets (2/6)

```
1     #include <stdio.h>
2     #include <time.h>
3
4     int main(int argc, char **argv){
5         struct timespec start;
6         clock_gettime(CLOCK_MONOTONIC, &start);
7         foo();
8         struct timespec end;
9         clock_gettime(CLOCK_MONOTONIC, &end);
10        double time = end.tv_sec - start.tv_sec + 1e-9 *
              (end.tv_nsec - start.tv_nsec);
11        printf("done after %f seconds", time);
12        return 0;
13    }
```

Quizreihe: Code-Snippets (2/6)

- [ ] Fehler beim Umfang der Zeitmessung
- [ ] Fehler bei der Berechnung der Differenz
- [ ] Fehler bei der Ausgabe des Ergebnisses
- [ ] kein Fehler

Quizreihe: Code-Snippets (3/6)

```
1     int main(int argc, char **argv){
2         struct timespec start;
3         clock_gettime(CLOCK_MONOTONIC, &start);
4         printf("Started measurement, doing
              calculation...\n");
5         foo();
6         printf("Ending measurement...\n");
7         struct timespec end;
8         clock_gettime(CLOCK_MONOTONIC, &end);
9         double time = end.tv_sec - start.tv_sec + 1e-9 *
              (end.tv_nsec - start.tv_nsec);
10        printf("done after %f seconds", time);
11        return 0;
12    }
```

Quizreihe: Code-Snippets (3/6)

- ☐ Fehler beim Umfang der Zeitmessung
- ☐ Fehler bei der Berechnung der Differenz
- ☐ Fehler bei der Ausgabe des Ergebnisses
- ☐ kein Fehler (die `#include`'s fehlen aus Platzgründen)

Quizreihe: Code-Snippets (4/6)

```
1   int main(int argc, char **argv){
2       struct timespec start;
3       clock_gettime(CLOCK_MONOTONIC, &start);
4       float res = 0;
5       for (int i = 0; i < 100000000; i++)
6           res = sqrt(5);
7       struct timespec end;
8       clock_gettime(CLOCK_MONOTONIC, &end);
9       double time = end.tv_sec - start.tv_sec + 1e-9 *
            (end.tv_nsec - start.tv_nsec);
10      printf("done after %f seconds, result is %f",
            time, res);
11      return 0;
12  }
```

Quizreihe: Code-Snippets (4/6)

- ☐ Schleife kann wegoptimiert werden
- ☐ zu kurzer Abstand zwischen Messpunkten
- ☐ falsche Schleifensyntax
- ☐ kein Fehler (die `#include`'s fehlen aus Platzgründen)

Quizreihe: Code-Snippets (5/6)

```
1     int main(int argc, char **argv){
2         float input = parse_args(argc, argv);
3         struct timespec start;
4         clock_gettime(CLOCK_MONOTONIC, &start);
5         float res = 0;
6         for (int i = 0; i < 100000000; i++)
7             res += sqrt(input);
8         struct timespec end;
9         clock_gettime(CLOCK_MONOTONIC, &end);
10        double time = end.tv_sec - start.tv_sec + 1e-9 *
              (end.tv_nsec - start.tv_nsec);
11        printf("done after %f seconds, result is %f",
              time, res);
12        return 0;
13    }
```

Quizreihe: Code-Snippets (5/6)

- ☐ Schleife kann trotzdem wegoptimiert werden
- ☐ zu kurzer Abstand zwischen Messpunkten
- ☐ durch Veränderung des Schleifenkörpers wird lange genug gerechnet
- ☐ kein Fehler (die `#include`'s fehlen aus Platzgründen)

Quizreihe: Code-Snippets (6/6)

```
1     int main(int argc, char **argv){
2         float input = parse_args(argc, argv);
3         float res = 0; double time = 0;
4         for (int i = 0; i < 100000000; i++) {
5             // only measure sqrt(), no loop-overhead.
6             struct timespec start;
7             clock_gettime(CLOCK_MONOTONIC, &start);
8             res += sqrt(input);
9             struct timespec end;
10            clock_gettime(CLOCK_MONOTONIC, &end);
11            time += end.tv_sec - start.tv_sec + 1e-9 *
                  (end.tv_nsec - start.tv_nsec);
12        }
13        printf("done after %f seconds, result is %f",
              time, res);
14        return 0; }
```

Quizreihe: Code-Snippets (6/6)

- ☐ loop-overhead kann vernachlässigt werden
- ☐ zu kurzer Abstand zwischen Messpunkten
- ☐ genaueres Ergebnis
- ☐ kein Fehler

Umgebungsbedingungen

- Überprüfung auf Korrektheit
- mindestens drei Wiederholungen der Messung
- exklusive Nutzung der Ressourcen
- Messung auf echter Hardware ohne CPU-Features

Quiz: geeignete Testmaschinen

Womit kann und soll getestet werden?

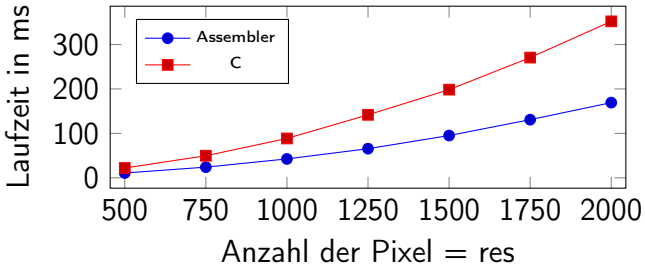
- ☐ eigener Rechner
- ☐ lxhalle
- ☐ VM's, WSL

Dokumentation

- Hintergrund: Reproduzierbarkeit der Ergebnisse
- Setup des Systems (CPU, Speicher, Betriebssystem)
- Kompiliervorgang (Compiler, Version, Optionen)
- Testvorgang (Eingaben, Wiederholungen)
- Aufbereitung der Ergebnisse in Diagramm

Beispieldokumentation

Getestet wurde auf einem System mit einem Intel i7-9700K Prozessor, 3.60GHz, 16 GB Arbeitsspeicher, Ubuntu 20.04, 64 Bit, Linux-Kernel 5.4.0. Kompiliert wurde mit GCC 8.1.0 mit der Option -03.
Die Berechnungen wurden mit Eingabgrößen von 500 bis 2000 jeweils 20 mal durchgeführt und das arithmetische Mittel für jede Eingabegröße wurde in folgendes Diagramm eingetragen:



Zusammenfassung

- Zeitmessung mit `CLOCK_MONOTONIC`
- Vermeidung unnötiger Operationen im Messbereich
- mindestens eine Sekunde Abstand zwischen Messungen
- mindestens Optimierungsstufe 2
- mindestens drei Wiederholungen der Messung
- Bereitstellung maximaler Ressourcen
- möglichst genaue Dokumentation