


PALO Praktikum 3 A3.1:

Anmerkung zu Compiler-Flags:

Bei beiden von uns hat der Intel-Compiler die Prüfsumme extrem verzerrt (etwa Faktor 2), weshalb das Projekt nur mit /O2 optimiert wurde. Optimierungen auf verschiedene Befehlssätze (/arch mit :SSE2, :AVX, etc.) führten auf keinem der beiden Rechnern zu messbaren Verbesserungen.

Durch die Angleichung der Datentypen bei der Array-Initialisierung konnte der Init-Vorgang deutlich beschleunigt werden. Bei beiden Rechnern gibt es hier einen Zuwachs von jeweils 66% und 200%.

```
// initialize matrix a,b,c
starttime = wtime();
for (int i = 0; i < N; i++) {
    float x = i + 1.f;
    // Original
    //a[i] = 1. / (i + 1);
    //b[i] = 2. / (i + 1);
    //c[i] = 3. / (i + 1);
    a[i] = 1.f / x;
    b[i] = 2.f / x;
    c[i] = 3.f / x;
}
elapsedtime_init = wtime() - starttime;
```




Durch das Auslagern der sin(15.0)-Berechnung konnte bei beiden Systemen ebenfalls ein Zuwachs von jeweils 100% und 150% erreicht werden.

```
float sinVal = sin(15.f);
for (int i = 0; i < N; i++) {
    // Original
    //d[i] = ((a[i] / b[i]) / c[i]) * sin(15.0);
    d[i] = ((a[i] / b[i]) / c[i]) * sinVal;
}
```

Die einzige Optimierung die auf keinem der beiden Systeme zu einem Zuwachs geführt hat, ist die Auslagerung der Berechnung von (i + 1) bei der Initialisierung.

```
// initialize matrix a,b,c
starttime = wtime();
for (int i = 0; i < N; i++) {
    float x = i + 1.f;
    // Original
    //a[i] = 1. / (i + 1);
    //b[i] = 2. / (i + 1);
    //c[i] = 3. / (i + 1);
    a[i] = 1.f / x;
    b[i] = 2.f / x;
    c[i] = 3.f / x;
}
elapsedtime_init = wtime() - starttime;
```



Der Gesamt-Zuwachs wurde hier dokumentiert:

Aufgabe	Optimierung	Laufzeit Intel Xeon E3-1231	Zuwachs %	Laufzeit Ryzen 7 4700U	Zuwachs %
3.1	O2 (Baseline)	(Init) 0.963640 Secs, (Calc) 0.592152 Secs, CheckSum 1.000000		(Init) 0.484247 Secs, (Calc) 0.311242 Secs, CheckSum 1.000000	
		(Init) 0.629321 Secs, (Calc) 0.453801 Secs, CheckSum 1.000000		(Init) 0.204687 Secs, (Calc) 0.194386 Secs, CheckSum 1.000000	
		(Init) 0.618746 Secs, (Calc) 0.462227 Secs, CheckSum 1.000000		(Init) 0.206516 Secs, (Calc) 0.174555 Secs, CheckSum 1.000000	
		(Init) 0.615859 Secs, (Calc) 0.452571 Secs, CheckSum 1.000000		(Init) 0.186828 Secs, (Calc) 0.174248 Secs, CheckSum 1.000000	
		(Init) 0.621684 Secs, (Calc) 0.456267 Secs, CheckSum 1.000000		(Init) 0.186164 Secs, (Calc) 0.173502 Secs, CheckSum 1.000000	
	Schleife 1 optimieren durch: 1. => 1.f i + 1 => i + 1.f	(Init) 0.540429 Secs, (Calc) 0.583540 Secs, CheckSum 1.000000	init: 200% calc: 0%	(Init) 0.391119 Secs, (Calc) 0.293618 Secs, CheckSum 1.000000	init: 66% calc: kein Zuwachs
		(Init) 0.220112 Secs, (Calc) 0.467132 Secs, CheckSum 1.000000		(Init) 0.121314 Secs, (Calc) 0.198269 Secs, CheckSum 1.000000	
		(Init) 0.209462 Secs, (Calc) 0.459088 Secs, CheckSum 1.000000		(Init) 0.125163 Secs, (Calc) 0.190453 Secs, CheckSum 1.000000	
		(Init) 0.205370 Secs, (Calc) 0.458464 Secs, CheckSum 1.000000		(Init) 0.109926 Secs, (Calc) 0.173838 Secs, CheckSum 1.000000	
		(Init) 0.198493 Secs, (Calc) 0.455925 Secs, CheckSum 1.000000		(Init) 0.105094 Secs, (Calc) 0.174546 Secs, CheckSum 1.000000	
	Schleife 2 optimieren durch: sin(15.0) => vorher ausrechnen und Ergebnis nur einsetzen	(Init) 0.531858 Secs, (Calc) 0.289164 Secs, CheckSum 1.000000	init: 200% calc: 150%	(Init) 0.391653 Secs, (Calc) 0.172580 Secs, CheckSum 1.000000	init: 66% calc: 100%
		(Init) 0.227085 Secs, (Calc) 0.188759 Secs, CheckSum 1.000000		(Init) 0.117267 Secs, (Calc) 0.092318 Secs, CheckSum 1.000000	
		(Init) 0.232915 Secs, (Calc) 0.200000 Secs, CheckSum 1.000000		(Init) 0.119848 Secs, (Calc) 0.099260 Secs, CheckSum 1.000000	
		(Init) 0.204450 Secs, (Calc) 0.170944 Secs, CheckSum 1.000000		(Init) 0.118414 Secs, (Calc) 0.089957 Secs, CheckSum 1.000000	
		(Init) 0.209564 Secs, (Calc) 0.180036 Secs, CheckSum 1.000000		(Init) 0.117961 Secs, (Calc) 0.083627 Secs, CheckSum 1.000000	
	Schleife 1 optimieren durch: float x = i + 1.f; was danach nunoch eingesetzt werden muss	(Init) 0.536810 Secs, (Calc) 0.283536 Secs, CheckSum 1.000000	init: 200% calc: 150%	(Init) 0.393382 Secs, (Calc) 0.174911 Secs, CheckSum 1.000000	init: 66% calc: 100%
		(Init) 0.218275 Secs, (Calc) 0.192857 Secs, CheckSum 1.000000		(Init) 0.117514 Secs, (Calc) 0.091494 Secs, CheckSum 1.000000	
		(Init) 0.225809 Secs, (Calc) 0.193664 Secs, CheckSum 1.000000		(Init) 0.122796 Secs, (Calc) 0.101204 Secs, CheckSum 1.000000	
		(Init) 0.197357 Secs, (Calc) 0.171380 Secs, CheckSum 1.000000		(Init) 0.128097 Secs, (Calc) 0.094069 Secs, CheckSum 1.000000	
		(Init) 0.205778 Secs, (Calc) 0.177960 Secs, CheckSum 1.000000		(Init) 0.117133 Secs, (Calc) 0.115443 Secs, CheckSum 1.000000	
3.2	Unoptimierte Faktorisierung	Time Elapsed 3.596402 Secs,		Time Elapsed 3.067719 Secs,	
		Time Elapsed 3.538722 Secs,		Time Elapsed 2.974726 Secs,	
		Time Elapsed 3.639669 Secs,		Time Elapsed 2.961791 Secs,	
		Time Elapsed 3.616473 Secs,		Time Elapsed 2.964418 Secs,	
		Time Elapsed 3.646686 Secs,		Time Elapsed 2.967121 Secs,	
	- Array in main() deklariert - init()-Funktion zum einmaligen Füllen des Arrays erstellt - factorial_opt() liest nun nur noch die fertigen Werte aus dem Array aus	Time Elapsed 0.000142 Secs,	35000%	Time Elapsed 0.000091 Secs,	31222%
		Time Elapsed 0.000124 Secs,		Time Elapsed 0.000087 Secs,	
		Time Elapsed 0.000106 Secs,		Time Elapsed 0.000087 Secs,	
		Time Elapsed 0.000106 Secs,		Time Elapsed 0.000087 Secs,	
		Time Elapsed 0.000106 Secs,		Time Elapsed 0.000087 Secs,	
	- alle neuen Funktionen mussten um einen Array-Parameter erweitert werden, um das Array aus main() durch die Methoden durchzureichen	Time Elapsed 0.000106 Secs,		Time Elapsed 0.000087 Secs,	

Anmerkungen A3.2:

In der main() Methode wurde ein int Array von Größe 12 deklariert. Dieses wird in einer init()-Methode einmalig mit Werten von 1!,...,12! gefüllt. Diese Initialisierung dauert 0,000005s und wurde in der Tabelle versehentlich nicht gelistet. Diese Zeit müsste also einmalig zu Beginn des Programmdurchlaufs hinzuaddiert werden. Die Zeiten für die individuellen Schleifendurchläufe bleiben bestehen.

Die factorial_opt()-Methode bekommt das Array als Pointer übergeben und prüft nur, ob der angefragte Wert < 12 ist. Ist er das, wird der entsprechende Wert aus dem Array zurückgegeben, ansonsten wird 0 zurückgegeben.