Pthreads

Was sind Pthreads?

Posix Threads (Akronym für **P**ortable **O**perating **S**ystems **I**nterface) oder abgekürzt Pthreads genannt, ist ein standardisiertes Programmier-Interface, um parallele Programme zu implementieren. Dafür wird eine API bereitgestellt, mit der Threads erzeugt und kontrolliert werden können. Ein Thread ist ein Ausführungsstrang, der je nach Hardwarearchitektur, auf dem der Thread abläuft, neben einen oder mehreren anderen Threads gleichzeitig abgearbeitet werden kann. Die aktuellste Version der Posix API stüzt sich dabei auf den *IEEE Std 1003.1, 2013* Standard und wurde ursprünglich als Synonym für den *IEEE Std. 1003.1-1995* Standard verwendet.

Die Subroutinen der Pthread API können folgendermaßen gruppiert werden:[1]

- Thread Management
- Mutexe
- Condition-Variables
- Synchronisation

Die Posix API ist in vielen Unix-Betriebssystemen wie zum Beispiel FreeBSD, Linux, Mac OS X, Android, etc. vorhanden und typischerweise in einer Bibliothek gebündelt. Unter Microsoft Windows existieren ebenfalls Implementierungen, die entweder als Subsystem betreiben werden können oder über third-party Packages wie *pythreads-w32* auf die existierende Windows API aufgesetzt werden können.

Parallelisieren eines sequentiellen Problems am Beispiel des Laplace-DGS

Da das Laplace Differentialgleichungssystem so arbeitet, dass das Ergebnis einer Iteration als Eingabewert der nächsten Iteration vorliegen muss, handelt es sich bei dem Laplace-DGS um ein sequentielles Problem. Um dieses Problem dennoch parallel abarbeiten zu können, wird die Aufgabe in Teilaufgaben unterteilt, deren Anzahl die der verfügbaren Threads gleicht. Beispielsweise wird bei einer CPU mit vier Kernen das Problem in vier Teilaufgaben aufgeteilt, die dann an vier Threads weitergeben werden. Falls das Problem nicht genau in vier Threads unterteilt werden kann, wird ein weiterer Thread erstellt, der den Rest des Problems übernimmt.

[1] Barney 2017

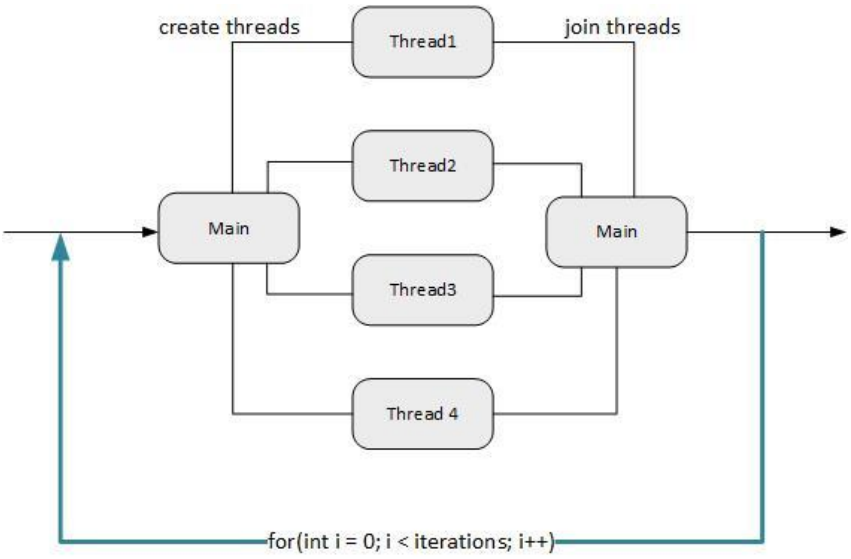

Abbildung 1: Programmablauf mit Threads

Die Berechnungen können somit unabhängig voneinander parallel abgearbeitet werden. Da der Laplace-Algorithmus bei jeder Berechnung mit den Werten der vier nächsten Nachbarn rechnet, müssen im letzten Schritt, nach dem Abarbeiten der Threads die Ergebnisse zwischengespeichert werden. (Schlagwort hierbei ist Single-Instruction Multiple-Data). Dies geschieht beispielsweise über ein weiteres Array.

Implementierung in C++

Bei dem Programmablauf wird die Laplace-Berechnung iterativ aufgerufen. Die Anzahl der Iterationen werden dabei beim Programmstart als Parameter übergeben.

```
unsigned threadNumber = std::thread::hardware_concurrency();
if (single_threaded) threadNumber = 1;
int step_size = rowcol / threadNumber;

vector<std::thread> threads(threadNumber + 1); //+1 for "rest-task"
```

```
// split the array in subtasks that can be executed by a thread
for (int k = 0; k < threadNumber; k += step_size) {
	int start = k + 1;
	int end = k + step_size - 1;

	threads.push_back(std::thread(calculateLaplace, start, end, rowcol, (toggle == 0
? laplace0 : laplace1), (toggle == 0 ? laplace1 : laplace0)));
}
```

Bei jeder Iteration werden Threads erstellt, die den Laplace-Algorithmus für Teilbereiche des Arrays ausführen. Dafür wird zunächst ermittelt, wie viele Threads die Hardware gleichzeitig abarbeiten kann, das Array in Bereiche aufgeteilt und anschließend das Laplace-DGS für die verfügbaren Threads aufgerufen. Die Funktion dazu lautet *calculateLaplace()*. Um das Laplace-DGS zu parallelisieren, wurde die Funktion der Laplace-Berechnung so umgeschrieben, dass zwei Arrays übergeben werden können, ein Input- und ein Output-Array. Die Ergebnisse der Berechnung werden in dem Output-Array gespeichert. Bei jeder Iteration wird das Input- und Output-Array für die Threads vertauscht, sodass der Speicher nicht immer wieder neu allokiert werden muss und die Ergebnisse von der vorherigen Iteration als Ausgangswerte für die nächste Iteration dienen.

```
for (auto &cur_thread : threads) {
	if (cur_thread.joinable()) {
		cur_thread.join();
	}
}

toggle = 1 - toggle;
```

Anschließend wird gewartet, bis jeder Thread seine Aufgabe fertig berechnet hat. Danach wird die Toggle-Variable geändert, sodass danach in der nächsten Iteration die Arrays in einer anderen Reihenfolge übergeben werden. Erst jetzt endet die Iteration und die nächste Iteration startet – insofern das Ende der Iterationen nicht erreicht ist.

Literaturverzeichnis

Barney, Blaise (2017): POSIX Threads Programming. Online verfügbar unter https://computing.llnl.gov/tutorials/pthreads/#PthreadsAPI, zuletzt aktualisiert am 07.03.2017, zuletzt geprüft am 18.04.2017.