Algorithm Engineering Lab Assignment 2

Brian Zahoransky (brian.zahoransky@uni-jena.de)

November 28, 2020

1. What causes false sharing?

False-Sharing entsteht, dann wenn zwei Prozessoren auf die gleiche Cachline schreiben. Dies könnte zum Beipiel folgenden Grund haben. Man legt in einem Programm zwei Variablen des Types int an, in diesem Beispiel a und b und initialisiert diese. Es könnte passieren, dass diese beiden Variablen jetzt in der selben Cachline des Hauptspeichers liegen. Jetzt wird eine parallele Region gestartet. Dabei schreibt ein Thread in a und ein weiterer Thread, welcher auf einem anderen Prozessor ausgeführt wird, schreibt auf b. Zu diesem Zeitpunkt haben beide Prozessoren die Cacheline bereits in ihren eigenen Cache geladen. Nachdem der erste Thread geschrieben hat wird die Cacheline invalid und muss im Cache des anderen Prozessors aktualisiert werden. In dieser Zeit muss der Prozessor warten. Dies kann zu erheblichen Performance-Einbußen führen.

2. How do mutual exclusion constructs prevent race conditions?

Race conditions treten dann auf, wenn mehrere Threads zur gleichen Zeit auf die gleiche Resource zugreifen wollen. Eine race condition bezüglich des Speichers könnte eintreten wenn zwei Prozessoren auf die gleiche Speicheradresse schreiben wollen. Zum Beispiel sollen die Teilsummen mehrerer Threads aufaddiert werden. Thread A und B laden jetzt zufällig zur selben Zeit und damit das gleiche Zwischenergebnis. Jetzt addieren beide Threads ihre Teilergebnisse auf dieses Zwischenergebnis und schreiben dieses zurück in ihren Cache. Jetzt muss die Cacheline, wie in Aufgabe 1 beschreiben, in den anderen Caches aktualisiert werden. Welche der beiden Varianten sich aber durchsetzt ist unklar. Letzen Endes erhalten ist weder so noch so das richtige Ergebnis.

Mutual exclusions verhindern dies. Sie garantieren, dass immer nur ein Thread auf eine kritische Resource zugreift. In eben erläutertem Beispiel läuft dies wie folgt ab. Thread A oder B wollen zur gleichen Zeit ihr Teilergebnis aufaddieren. Bevor Sie das können benötigen sie allerdings Schreibrecht, welches durch das Mutex vergeben wird. Einer der beiden Threads wird die atomare Aktion, des Mutex freischaltens zuerst durchführen. Nun kann er ungestört das Zwischenergebnis aktualisieren. Die Cachlines in den anderen Prozessoren werden aktualisiert und anschließend das Mutex wieder freigeschaltet. Der andere Thread wartet so lange. Danach ist er an der Reihe und durchläuft die gleiche Prozedur.

3. Explain the differences between static and dynamic schedules in OpenMP.

static: Zu Compile-Zeit ist festgeschrieben, wie viele und welche chunks ein Thread ausführen wird. Dies ist sinnvoll, wenn alle chunks ungefähr die selbe Laufzeit haben oder es könnte sinnvoll sein, falls die Arbeit zwar ungleich ist, aber bekannt ist wie viel Arbeit pro chunk anfällt.

dynamic: Die chunks werden dynamisch zur Laufzeit, den verschieden Threads zugeteilt. Dies ist sinnvoll, falls die Arbeit pro chunk bekannt ist aber nicht oder nur sehr schwer statisch gleichmäßig verteilt werden kann. Es wird zudem angewendet, falls zu Programmierzeit nicht klar ist, wie viel Arbeit pro chunk anfällt. Da Threads, die einen chunk abgearbeitet haben, zur Laufzeit ein neuer chunk zugewiesen werden muss, gibt es einen Overhead gegenüber dem static sceduling. Deshalb sollte die static Variante, wann immer sinnvoll bevorzugt werden.

4. What can we do if we've found a solution while running a parallel for loop in OpenMP, but still have many iterations left?

Es wird eine atomic bool-Variable initialisiert und eine atomic Lösungsvariable deklariert. Falls ein Thread wärend der Durchführung eines chunks eine Lösung findet, wird der Wert der bool-Variabe geswitched und der Wert der Lösungsvariable auf die gefundene Lösung gesetzt. Eine if-Abfrage zu beginn jedes chunks überprüft anhand der der bool-Variable, ob bereits eine Lösung gefunden wurde. Falls ja, wird der restliche Code übersprungen, somit werden die verbliebenen chunksn, nach dem finden der Lösung, nur alibi-mäßig durchlaufen. Es wird kein Code mehr abgearbeitet, abgesehen von der if-Abfrage.

5. Explain in your own words how std::atomic::compare_exchange_weak work.