Aufgabe 7: Koordinierung

a) Erklären Sie die Begriffe Spinlock und Barrier.

b) Was bewirken die Befehle `test_and_set` und `fetch_and_increment`?

c) Nennen Sie die vorgestellten Algorithmen für Spinlocks. Was zeichnet diese im einzelnen aus?

Lösungsvorschlag Aufgabe 7:

a) Bei der Bearbeitung von parallelen Aufgaben in UMA und NUMA Architekturen ist die effiziente Koordinierung der einzelnen Threads von grosser Bedeutung. Die Prozessoren kommunizieren über gemeinsame Datenstrukturen. Die Aufgabe der Koordinierung besteht darin die gemeinsamen Datenstrukturen konsistent zu halten. Dabei sind zwei Klassen von Interesse: *Blocking* und *busy-wait*.

Beim *busy-wait* testen die Prozesse wiederholt die gemeinsamen Variablen, um herauszufinden, wann sie weiterarbeiten dürfen. Hier werden die Konstrukte *Spinlock* und *Barriere* unterschieden.

Spinlocks: Koordiniert einen gegenseitigen Ausschluss, in dem genau einem Prozessor erlaubt wird auf den gemeinsamen Speicher zugreifen zu dürfen. Während einer Berechnung kann es sehr häufig zum Gebrauch von Spinlocks kommen.

Barrieren: Kein Prozess kann über einen bestimmten Punkt im Algorithmus hinausgehen, bis nicht sicher gestellt ist, daß alle anderen auch an diesem angekommen sind. Dies ist notwendig, beim Übergang von parallelen in serielle Phasen oder beim Start einer neuen Iteration. Diese Art von Synchronisation kann demzufolge sehr stark zu allgemeinen Laufzeit eines Programms beitragen.

Beim busy-wait kann das Problem eines *Hot Spots* auftauchen. Hierunter versteht man das hÄufige Zugreifen vieler Prozessoren auf eine einzelne Synchronisationsvariable.

b) test_and_set

- merkt sich den alten Wert
- schreibt “belegt” in die Variable
- gibt den alten Wert zurück

Wenn der alte Wert “belegt” war, wartet der Prozess weiter. War der alte Wert “frei”, ist dies das Signal zum Weiterrechnen
Bei jedem test_and_set ist eine Schreiboperation notwendig, die damit in kohärenten Systemen eine Invalidierung nach sich zieht.

fetch_and_increment

- merkt sich den alten Wert
- erhöht den Zähler
- gibt den alten Wert zurück

Ein Thread, der einen fetch_and_increment ausführt bekommt mit dem alten Wert seinen Platz in der Warteschlange mitgeteilt.