High Performance Computing

Günther Wutz

2. Oktober 2012

Einführung

Eine (informale) Definition von Parallelität: verschiedene Prozessoreinheiten (CPUs, ALUs, FPUs, etc.) arbeiten simultan (z. B. parallel) um einen gemeinsamen Task zu lösen.

Warum wollen wir das?

- Manche Applikationen benötigen einen Speedup z. B. Klimavorhersagen, Windtunnel, Motorkonstruktion, Atomkraftwerktests oder Spiele

Herausforderungen von Parallelabarbeitung

- Rießige Speicheranforderungen
- Hohe Durchsatzanforderungen (z. B. Reiseplatformen)
- Verteilte und Kooperative Internetanwendungen
- Datenreplikation und Ausfallstrategien (Rechenclouds)
- Heterogenität und Interoperabilität

Geschwindigkeit

Geschwindigkeit bzw. Performance ist unser **Hauptgrund** um Parallele Programme zu schreiben. Formal ist die `Geschwindigkeitssteigerung für p Prozessoren` folgendermaßen definiert:

$$S_p = \frac{T_1}{T_p}$$

- p ist die Anzahl der Prozessoren
- T_p ist die Laufzeit unserer Applikation auf p Prozessoren
- T_1 ist die Sequenzielle Laufzeit der gleichen Applikation (Laufzeit auf einem Prozessor)

Demzufolge ist S_p der relative Laufzeitvorteil dem man mit p Prozessoren erreichen kann, verglichen mit einer sequenziellen Implementierung.

Den Geschwindigkeitszuwachs, den man für p Prozessoren erwarten kann: Potenziell `p`. Dazu ein Rechenbeispiel: Unsere Applikation benötigt sequenziell 10s zur Ausführung. Wenn wir nun 2 Prozessoren nutzen, könnte das oberste Limit für die Ausführung 5s betragen. Daraus ergibt sich ein Speedup von $S_p = \frac{10s}{5s} = 2$ was unserer eingesetzen Prozessoranzahl entspricht.

Wir werden später noch sehen, dass auch Geschwindigkeitssteigerungen über linearem Zuwachs möglich sind. Diese werden entsprechend *superlinear* genannt.

Für typische Software kann man den Speedup zwischen $1 \leq S_p \leq p$ vermuten. Manche Programme können sogar einen Einbruch der Geschwindigkeit mit Parallel Computing erleben. Nicht alle Applikationen sind für Parallelisierung geeignet. Auch können schlechte Implementierungen für einen schlechten Performancezuwachs schuld sein, obwohl die Applikation ansonsten für Parallelisierung geeignet wäre.

Amdahl's Law (1967)

Amdahl beschrieb 1967 bereits den Geschwindigkeitszuwachs durch Parallelisierung. Er zerlegt ein Programm in zwei Teile, den vollständig sequentiellen Teil (α)und den vollständig Parallelisierbaren Teil $(1 - \alpha)$.

Daraus folgt die parallele Laufzeit auf p Prozessoren:

$$T_p = \alpha T_1 + (1 - \alpha) * \frac{T_1}{p} = T_1 * (\alpha + \frac{(1 - \alpha)}{p})$$

Daraus wiederum kann der Speedup durch obige Formel berechnet werden:

$$S_p = \frac{T_1}{T_p} = \frac{1}{\alpha + \frac{(1-\alpha)}{p}} \leq \frac{1}{\alpha}$$

Daraus folgt: Eine Applikation, die zu 90% perfect parallelisiert werden kann, kann nur bis zum Faktor 10 beschleunigt werden, unabhängig davon, wieviele Prozessoren benutzt werden. Damit bildet $1/\alpha$ die obere Schranke.

Gustafson's Law (1988)

Amdahl nahm in seiner These an, dass der sequentielle Part mit der Problemgröße wächst. Gustafson Ansatz war, dass der sequentielle Teil der meisten Applikationen eine konstante Laufzeit hat.

Sei $T_1(n)$ die sequentielle Laufzeit für die Problemgröße n. So folgt

$$T_1(n) = \tau + v(n)$$

wobei τ der konstante sequentielle Teil und v perfekt parallelisierbar ist. Daraus folgt die Parallele Laufzeit auf p Prozessoren

$$T_p(n) = \tau + \frac{v(n)}{p}$$

Nach Gustafson ist damit eine Speedup

$$S_p(n) = \frac{T_1(n)}{T_p(n)} = \frac{\tau + v(n)}{\tau + \frac{v(n)}{p}} = \frac{\frac{\tau}{v(n)} + 1}{\frac{\tau}{v(n)} + \frac{1}{p}}$$

Wenn man nun die Problemgröße n gegen Unendlich gehen lässt, so wird die Laufzeit v(n) immer größer.

$$\lim_{n \to \infty} S_p(n) = p$$

Solche Applikationen nennt man **Skalierbar**.