Kurs 01727 Parallele Programmierung und Grid Computing Einsendearbeit 1

Aufgabe 1-1 (8 Punkte)

Beantworten Sie die folgenden Fragen:

a) Erläutern Sie die Unterschiede und die Gemeinsamkeiten zwischen einem UMA und einem CC-NUMA System!

b) Erläutern Sie die Ursachen für den Overhead in parallelen Programmen, die einen linearen Speedup verhindern!

Musterlösung

a) UMA steht für Unified Memory Access. Das bedeutet, dass alle Prozessoren eines Systems Zugriff auf den gleichen Speicherbereich haben und auch den gleichen Adressraum verwenden. Die Zugriffszeiten sind dabei auf alle Bereiche des Speichers gleich. Der Zugriff auf den Speicher ist dabei cache-kohärent. Ein System mit einer Multicore-CPU kann als UMA System betrachtet werden.

Auch bei einem CC-NUMA System haben alle Prozessoren Zugriff auf den gemeinsamen Speicher und verwenden den gleichen Adressraum. Allerdings sind die Zugriffszeiten auf den Speicher nicht alle gleich. Daher steht NUMA für Non Unified Memory Access. Das CC steht für cache-kohärent, bei einem CC-NUMA System ist also Cache-Kohärenz gewährt. Die meisten modernen Systeme, die über zwei oder mehr Multicore-CPUs verfügen, sind CC-NUMA Systeme.

b) Der Overhead lässt sich in zwei Teile aufteilen, und zwar den eigentlichen Overhead, für den man drei Ursachen finden kann:

- t_{com}: Die Zeit, die für den Austausch von Daten zwischen Prozessoren verwendet wird
- t_{wait}: Die Wartezeit (z. B. aufgrund ungleicher Auslastung der verschiedenen Prozessoren)
- t_{sync}: Die Zeit, die für die Synchronisierung der verwendeten Prozessoren verwendet wird

und die Setup-Zeit zu Beginn, die sich in zwei Phasen aufteilen lässt:

- t_{place}: Die Zeit, die für die Zuweisung von Aufgaben an einzelne Prozessoren verwendet wird
- t_{start}: Die Zeit, die benötigt wird, um die parallelen Aufgaben auf allen Prozessoren zu starten

Aufgabe 1-2 (8 Punkte)

Gegeben sei eine Anwendung mit einer sequentiellen Laufzeit von m Zeiteinheiten (ZE). Eine parallele Version nutzt einen Master-Prozessor und n Worker-Prozessoren.

Der Master-Prozessor übermittelt zunächst den Worker-Prozessoren ihre Aufgaben. Dies nimmt 1 Zeiteinheit (ZE) in Anspruch. Danach arbeiten die Worker-Prozessoren für $\frac{m}{n} - 2$ ZE, und senden ihre Teilergebnisse in 1 ZE zu dem Master-Prozessor. Zum Schluss vereinigt der Master-Prozessor die Teilergebnisse zu einem Gesamtergebnis in einer Zeit von n ZE.

Berechnen Sie den Speedup (parametrisiert in m) für $n = 2$, $n = m$ und $n = \sqrt{m}$

Musterlösung

Aus dem Aufgabentext leiten wir die sequentiellen und parallelen Laufzeiten t_s und t_p ab, mit

$$t_s = m$$

$$t_p(n) = 1 + \frac{m}{n} - 2 + 1 + n = \frac{m}{n} + n$$

Mit $S(n) = \frac{t_s}{t_p(n)}$ sind die Speedups für $n = 2$,$n = m$,$n = \sqrt{m}$:

$$S(2) = \frac{t_s}{t_p(2)} = \frac{m}{m/2 + 2} = 2 - \frac{8}{m+4}$$

$$S(m) = \frac{t_s}{t_p(m)} = \frac{m}{1+m} = 1 - \frac{1}{m+1}$$

$$S(\sqrt{m}) = \frac{t_s}{t_p(\sqrt{m})} = \frac{m}{2\sqrt{m}} = \frac{\sqrt{m}}{2}$$

Aufgabe 1-3 (6 Punkte)

Angenommen eine Applikation hat eine sequentielle Laufzeit von $t_s = m * log_2 m$. Der serielle Abschnitt, welcher nicht parallelisiert werden kann, nimmt $f * (t_s) = \frac{m}{log_2 m}$ Zeiteinheiten in Anspruch. Was ist der maximal mögliche Speedup (parametrisiert in m) gemäß Amdahl's Gesetz? Wie groß ist dieser maximale Speedup für $m = 2^{20}$?

Musterlösung

Wir erhalten

$$f = \frac{f * t_s}{t_s} = \frac{m/\log_2 m}{m * \log_2 m} = \frac{1}{(\log_2 m)^2}$$

Laut Amdahl's Gesetz ist der maximal mögliche Speedup:

$$\frac{1}{f} = (\log_2 m)^2$$

Für $m = 2^{20}$ist der Speedup $(log_2(2^{20}))^2 = 20^2 = 400$.

Aufgabe 1-4 (8 Punkte)

Im Folgenden soll das Roofline-Modell näher betrachtet werden. Dazu betrachten wir ein System mit einer maximalen Leistung von 700 GFlops/s und einer Speicherbandbreite von 150 bzw. 250 GB/s hat.

a) Betrachten wir zunächst die dgemm-Operation:

$$C \leftarrow \alpha AB + \beta C$$

Hier sind A, B und C zwei Matrizen der Größe $n \times n$, α und β sind Skalare. Berechnen Sie die Arbeit, die Speicherzugriffe und die Intensität für diese Operationen, wenn wir mit float-Werten und mit double-Werten rechnen. Wir gehen davon aus, dass alles in den Cache passt.

b) Zeichnen Sie das Roofline-Modell für dieses System. Zeichnen Sie die Intensität für $n = 1024$ und $n = 32$ für die dgemm operation mit double-Werten in das Diagramm ein.

c) Ist die Applikation eher durch den Speicher oder die Anzahl der Operationen beschränkt?

Musterlösung

a) Die Matrix-Matrix Multiplikation braucht n^3 Multiplikationen und $n^2 * (n-1)$ Additionen. Die Multiplikation mit α wird n^2 mal durchgeführt. Die Multiplikation von β und C benötigt n^2 Multiplikationen. Die beiden Matrizen zu addieren benötigt erneut n^2 Operationen. Insgesammt haben wir also

$$W(n) = 2n^3 - n^2 + n^2 + n^2 + n^2 = 2n^3 + 2n^2$$

Für Die Speicherzugriffe gilt für Float:

$$Q_r \leq 3 * n^2 * 4 = 12n^2$$

und

$$Q_w \leq n^2 * 4$$

also

$$Q \geq 16n^2 \text{ für float}$$

und

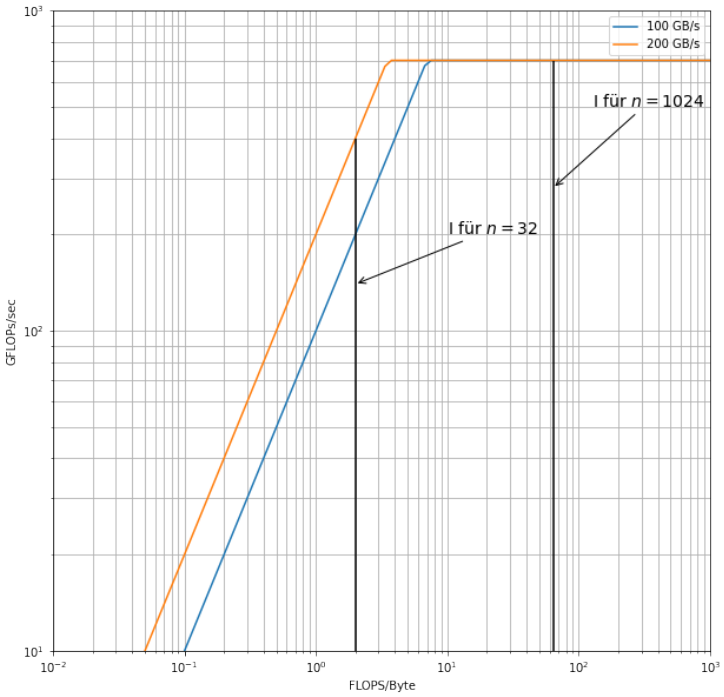
$$Q \geq 32n^2 \text{ für double}$$

Also ist

$$I \leq \frac{2n^3 + 2n^2}{16n^2} = \frac{n+1}{8} \text{ für float}$$

und

$$I \leq \frac{2n^3 + 2n^2}{32n^2} = \frac{n+1}{16} \text{ für float}$$



b)

c) Für $n = 32$ Speicherbegrenz, für $n = 1024$ CPU-Begrenzt (auf beiden Systemen).

Aufgabe 1-5 (10 Punkte)

Das parallele Programm `mergesort` wurde mit OpenMP optimiert. Den Quellcode des Programms sowie das zugehörige Makefile können Sie aus Moodle herunterladen. Wenn Sie über einen OpenMP-kompatiblen Compiler (z.B. gcc) auf Ihrem PC verfügen, können Sie das Programm selbst kompilieren. Alternativ steht Ihnen ein Notebook (scaling.ipynb) in Moodle zur Verfügung. Dieses enthält alle notwendigen Pfadangaben, um das Programm auf Hopper auszuführen. Sie können es direkt auf Hopper hochladen. Sie können dem Programm zwei Parameter übergeben: die Anzahl der Threads und die Anzahl der zu sortierenden Elemente.

Bitte evaluieren Sie sowohl die Strong- als auch die Weak-Scaling-Eigenschaften des Programms. Testen Sie das Programm mit 1, 2, 4, 8, 16 und 32 Threads. Für die Weak-Scaling-Tests starten Sie mit 2^{16} Elementen. Führen Sie die Strong-Scaling-Tests mit 2^{16} und 2^{20} Elementen durch. Bitte visualisieren Sie Ihre Messergebnisse grafisch. Was ist Ihnen dabei aufgefallen? Erläutern Sie Ihre Beobachtungen und Ergebnisse.

Aufgabe 1-6 (10 Punkte)

In den nächsten Kurseinheiten werden wir OpenMP und MPI kennenlernen. Beide Werkzeuge zur parallelen Programmierung beruhen auf der Programmiersprache `C`. Um mit `C` warmzuwerden, wollen wir nun ein wenig in `C` programmieren!

a) Schreiben Sie ein Programm, welches einen Vektor für $n = 10001$ float-Werte alloziert. Der erste Wert soll dabei auf 1 initialisiert werden, die übrigen Werte mit 10^{-9}.

b) Nun addieren Sie die Werte des Vektors in einer Schleife einmal von $0 - (n-1)$ zusammen und einmal von $(n-1) - 0$, also einmal vorwärts und einmal rückwärts.

c) Vergleichen Sie die Ergebnisse. Lassen Sie sich dazu das Ergebnis am besten auf 8 Nachkommastellen genau ausgeben, dies geht z.B. mit `printf("%.8f\n",sum)`. Was fällt ihnen auf? Erläutern Sie ihre Ergebnisse! Welche Probleme glauben sie ergeben sie daraus für die parallele Programmierung?

Musterlösung

```
float *data = (float *)malloc(sizeof(float) * 10001);

data[0] = 1.0;
for (int i = 1; i < 10001; i++) {
    data[i] = 1e-9;
}
float sum1 = 0.0;
for (int i = 0; i < 10001; i++) {
    sum1 += data[i];
}
printf("Result is %.8f \n", sum1);
float sum2 = 0.0;
for (int i = 10000; i >= 0; i--) {
    sum2 += data[i];
}
printf("Result is %.8f \n", sum2);
```

Für die erste Summe wird als *Ergebnis*1 zurückgegeben, für die zweite Summe 1.000010 .

Die Ergebnisse unterscheiden sich also. Der Grund dafür ist, dass die Zahlen im IEEE754-Format dargestellt werden. Während sich die 1 ohne Fehler darstellen lässt, ist eine genaue Darstellung von 10^{-9} nicht möglich, hier kommt es bei der Umrechnung zu einem Fehler von $2.8{*}10^{-17}$. Bei dem ersten Ergebnis kommt das Problem hinzu, dass der Exponent der IEEE754 Zahl -30 ist, bei der 1 ist der Exponent 0. Damit zwei Zahlen miteinander addiert werden können, müssen die Exponenten aneinander angepasst werden. Dabei wird bei 10^{-9} die Mantisse so weit verschoben, dass nur noch Nullen bleiben, so dass zu der 1 zehntausend mal eine Null addiert wird. Addiert man die Zahlen in der anderen Reihenfolge, so werden zunächst die kleinen Zahlen, addiert. Richtig gerechnet sind $10000 * 10^{-9} = 10^{-5}$. In IEEE754 Format hat die Zahl nun einen Exponenten von -20, was sich mit 1 verrechnen lässt.

Wir lernen daraus, dass die Reihenfolge, in der Zahlen miteinander addiert werden können, eine Rolle für das Ergebnis spielt. Dies sollten wir im Hinterkopf behalten, wenn wir parallelisieren, da hier nicht bei allen Aufgaben immer die gleiche Reihenfolge der Operationen garantiert werden kann!