



Parallele Programmierung mit C++ (SS 2017)

Abgabe bis zum 30. Juni 2017, 14:00 Uhr

Lernziele:

- Einfache Parallelisierung mit Hilfe der MPI-Schnittstelle wahlweise mit dem Fork-and-Join- oder Master/Worker-Pattern

Aufgabe 11: Konstellationen von Primzahlen

Zu den offenen Fragen der Zahlentheorie gehört, ob bestimmte aufeinanderfolgende Konstellationen von Primzahlen unendlich oft auftreten und wenn ja, wie sie verteilt sind. Diese Fragen werden gerne empirisch untersucht und lassen sich hervorragend parallelisieren.

Schreiben Sie eine MPI-Anwendung, die ein Intervall der natürlichen Zahlen $[N_1, N_2]$ und eine vorgegebene Primzahlkonstellation $\{n_i\}_{i=1}^{k-1}$ mit $n_i < n_{i+1}$ der Länge $k > 1$ erhält. Gesucht und zurückzuliefern sind dann alle Primzahlkonstellationen $(p, p + n_1, \dots, p + n_{k-1})$ mit $N_1 \leq p \leq N_2$. Primzahlpaare werden beispielsweise mit $\{2\}$ gesucht, Primzahlvierlinge mit $\{2, 6, 8\}$.

Sie können in einem ersten Ansatz davon ausgehen, dass alle beteiligten Prozesse gleich leistungsstark sind, und daher das Gesamtintervall auf die Zahl der Prozesse geeignet aufteilen. Es wäre aber zu begrüßen, wenn Sie sich in Ihrer Lösung von dieser Annahme lösen könnten, indem Sie entsprechend dem Master/Worker-Pattern mit einem Master arbeiten, der kleinere Intervalle verteilt, so dass Prozesse, die schneller rechnen, auch einen größeren Anteil am Gesamtproblem erhalten. Dabei ist es ausdrücklich zulässig, den Prozess mit dem *rank 0* als Master-Prozess zur reinen Koordinierung zu verwenden. (In diesem Fall setzt man die Zahl der MPI-Prozesse um eins höher an, da der koordinierende Prozess kaum Rechenzeit benötigt.)

Hinweise:

Für das Rechnen mit großen Zahlen empfiehlt sich die Verwendung der *GNU Multiple Precision Arithmetic Library*, kurz *GMP*. Die Dokumentation dazu steht direkt auf der Webseite

<http://gmplib.org/> zur Verfügung. Funktionen im Kontext von Primzahlen sind unter dem Abschnitt *Number Theoretic Functions* beschrieben. Mit Hilfe von *mpz_export* und *mpz_import* können Sie diese großen Zahlen in Arrays ganzer Zahlen konvertieren, die sich dann wiederum mit den bekannten Techniken bei MPI übertragen lassen. Beim Zusammenbau Ihres Programms ist dann zusätzlich *-lgmp* mit anzugeben.

Für GMP steht auch eine C++-Schnittstelle zur Verfügung. Hierfür ist **#include** `<gmpxx.h>` notwendig und beim Zusammenbau ist *-lgmpxx -lgmp* anzugeben. Leider bietet die C++-Schnittstelle nur eine Teilmenge der Funktionen der C-Schnittstelle an und hier fehlen insbesondere die für die Aufgabe relevanten zahlentheoretischen Funktionen. Es ist aber möglich, beides zu kombinieren. Wenn mit Hilfe der C++-Schnittstelle mittels *mpz_class p*; die ganzzahlige Variable *p* deklariert wird, dann kann mit *p.get_mpz()* das zugehörige C-Objekt abgefragt und den Funktionen aus der C-Schnittstelle übergeben werden.

Sie sollten in Ihrem ersten Schritt mit der Implementierung einer Funktion beginnen, die ohne Parallelisierung das oben genannte Problem für ein Intervall löst.

Danach sollten Sie die Aufgabenstellung mit Hilfe von MPI parallelisieren. Der Prozess mit dem *rank 0* sollte die Aufgabenstellung der Kommandozeile entnehmen und jeweils das erste Element einer gefundenen Konstellation auf der Standardausgabe ausgeben.

Sowohl auf der Thales als auch unseren Debian-Maschinen steht jeweils OpenMPI zur Verfügung, das sich mit `g++` verwenden lässt. Als Übersetzungskommando ist *mpic++* zu verwenden. Das ausführbare Programm ist mit *mpirun* zu starten. Unter Debian triggert die etwas ältere Fassung der OpenMPI-Bibliothek eine Warnung des *mpic++*, die mit der Option *-Wno-literal-suffix* unterdrückt werden kann.

Verpacken Sie all Ihre Quellen wiederum mit *tar* in ein Archiv und reichen Sie dies ein:

```
tar cvf primes.tar *.*pp [mM]akefile
submit pp 11 primes.tar
```

Viel Erfolg!