1 Bachelor-Thema: Seek-basierte parallele Pseudozufallszahlengeneratoren

1.1 Motivation

Ausgangsgrund waren HPC-Anwendungen am HLRS, die aufgrund Ermangelung von brauchbaren parallelen Zufallszahlengeneratoren bislang auf sequentielle Zufallszahlengeneratoren zurückfallen mussten. Der resultierende sequentielle Laufzeitanteil bei ansonsten gut parallelisierten Anwendungen beeinträchtigt aber wiederum die Gesamtskalierbarkeit zu stark (siehe Amdahlsches Gesetz), um diese Anwendungen effizient auf einem großen Supercomputer wie "*Hawk*" zu nutzen. Dafür müsste dieser Teil auch noch parallelisiert werden. Als Vorarbeit dafür soll diese Bachelor-Arbeit dienen.

1.2 Beschreibung

Einige Anwendungen (bspw. Monte-Carlo Simulationen) benötigen Zufallsdaten, die sie üblicherweise mittels Pseudozufallszahlengeneratoren (kurz PRNG, engl. *Pseudo-random number generator*) erzeugen. Ausgehend von einem Startwert (dem *Seed*, x_0) werden dabei durch eine bestimmte Formel $f(x)$ nacheinander (d. h. meist implizit durch $x_{n+1} = fi(x_n)$) weitere Werte berechnet, die zusammengenommen dann den gewünschten Zufallszahlenstream ergeben. Bei dem Versuch der Parallelisierung eines solchen PRNGs wird oft pro Thread oder Prozess einfach ein zufälliger, individueller Seed-Wert vergeben, in der Hoffnung, dass damit unabhängige Zufallszahlenstreams erzeugt werden. Dafür gibt es aber keinerlei Garantien, sodass auch überlappende Streams entstehen können, die dann teilweise identische Zufallswerte liefern und mit dem daraus resultierenden Bias die Anwendungsergebnisse verzerren oder gar nutzlos machen. Überdies gibt es leider auch keine Reproduzierbarkeit der Daten, z. B. bei Strong-Scaling Experimenten, weil durch diesen Ansatz die Zufallsdaten mit der Anzahl der Streams und damit der Parallelität variieren.

1.3 Ansatz

In dieser Bachelor-Arbeit soll daher ein anderer Ansatz für die Parallelisierung von Zufallszahlengeneratoren verfolgt werden. Dabei soll untersucht werden, welche impliziten PRNG-Formeln auch explizit dargestellt werden können, d. h. $x_n = fe(x_0)$. Mit einer solchen Darstellung wäre es nämlich möglich, den aktuellen Zustand n eines PRNG-Streams um eine beliebige Schrittweite nach vorne zu springen, ohne die dazwischenliegenden Werte zu berechnen. Wählt man dann die Sprungweite pro Thread/Prozess groß genug, kann man sicherstellen, dass die einzelnen Streams sich nicht überlappen. Weiß man darüber hinaus exakt, wie viele Zufallszahlenwerte insgesamt zu erzeugen sind (bspw. bei Strong-Scaling), kann man mittels einer Prefixsumme die parallelen Streams so genau vorspulen, dass diese mathematisch aufeinanderfolgen. Dadurch bekommt man sogar immer reproduzierbar die gleichen Zufallsdaten, unabhängig davon, wie viele Threads oder Prozesse diese Daten parallel erzeugen.

1.4 Aufgaben

In dieser Arbeit sollen mehrere existierende PRNG auf deren Parallelisierungsmöglichkeit nach diesem Ansatz untersucht werden und eine Sammlung von machbaren PPRNGs in Form einer Bibliothek implementiert werden. Darin sollte zumindest ein einfacher PPRNG (z. B. LCG-Typ wie *drand48*) enthalten sein und noch ein besserer PPRNG (wie bspw. *KISS*), der u. a. mit einer größeren Periode überhaupt erst für echte HPC-Anwendungen infrage kommt. Die implementierten PPRNGs sollen auf Korrektheit getestet werde (z. B. entstehen bei Strong-Scaling die gleichen Daten wie beim sequentiellen PRNG?), vermessen und miteinander verglichen werden. Optional kann die Arbeit auch erklären, warum manche Zufallszahlengeneratoren (bspw. kryptografisch-sichere PRNGs) nach diesem Schema nicht parallelisiert werden können.

1.5 Betreuer

Christian Siebert vom *HLRS* (*christian.siebert@hlrs.de*)