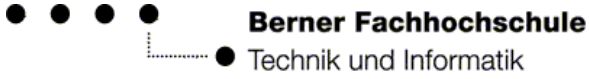

P2PMPI-Beispielprojekt

Aiport Simulation

Ein Beispiel-Projekt in Parallel Computing mit P2P-MPI

von Matthias Blaser[†] und Stefan Heinemann[*]

22. Januar 2012

[†]blasm5@bfh.ch
[*]heins4@bfh.ch

1 Einleitung

Dieses Dokument beschreibt gewisse Aspekte und Überlegungen im Beispielprojekt, welches im Zug des Fachs Parallel Computing an der Berner Fachhochschule entstanden ist.

Der Gebrauch der Software kann in der Datei *README.md* nachgelesen werden. Bisher wurde sie verteilt auf drei Prozessoren getestet - um sie mit mehr oder weniger Prozessoren zu starten muss im Sourcecode der Applikation die Definition der Flughäfen (Namen, Koordinaten auf der Map usw.) und dessen Zuordnung auf die Prozessoren angepasst werden.

2 Messaging

Das Messaging wird von zwei Threads übernommen, die neben der Simulation laufen. Der eine behandelt das Senden und der andere das Empfangen von Messages. Dabei gibt es eine Queue für den Send-Thread (Outgoing-Message-Queue), wo von der Simulation Nachrichten abgelegt werden können die verschickt werden sollen.

Der Ablauf des Messaging ist wie folgt: Falls der Zielflughafen eines Flugzeugs nicht denselben Rank besitzt wie der Flughafen, der vom aktuellen LP[1] verwaltet wird, wird am Ende des Start-Vorgangs zusätzlich zum Scheduling des ARRIVAL-Events eine Message an den LP des Zielflughafen versendet, welche die wichtigsten Daten des Fluges enthält. Dieser LP kreiert dann bei sich selbst ein ARRIVAL-Event. Ab diesem Zeitpunkt ist das Flugzeug beim LP des Zielflughafens bekannt und wird, sobald das ARRIVAL-Event ausgelöst wird, normal abgehandelt. Zudem kann es nun, sobald es in den sichtbaren Bereich des Flughafens fliegt, vom GUI gezeichnet werden.

Beim Flughafen, von dem das Flugzeug gestartet wurde, wird das Flugzeug beim Auslösen des ARRIVAL-Events aus der Simulation entfernt, da es nun nicht mehr von diesem LP behandelt wird.

3 Synchronisierung

Wir haben die Synchronisierung mit Null-Messages implementiert.

3.1 Null-Messages

Gibt es in der Outgoing-Message-Queue keine Messages die verschickt werden sollen, verschickt der Send-Thread automatisch in regelmässigen Abständen an alle anderen LPs eine Null-Message mit Timestamp und Lookahead.

Falls eine Message in der Queue ist, wird nebst dieser ebenfalls eine Null-Message mit demselben Timestamp und Lookahead wie das entsprechende “richtige” Event an die verbleibenden LPs verschickt.

[1] Logischer Prozessor: der einzelne Prozess, der ein Teil der Simulation selbständig durchführt

3.2 Lookahead-Queue

Beim Erhalt einer Message, egal ob Null-Message oder "richtig", wird der Timestamp mit dem Lookahead addiert und in der *LookaheadQueue* abgelegt. Diese ist dazu da, dass die Simulation immer abfragen kann, bis zu welchem Zeitpunkt sie sicher Events verarbeiten darf.

Diese ist so implementiert, dass sie für jeden der "anderen" LPs eine Queue bereitstellt, wo die jeweiligen Lookahead-Werte abgelegt werden. Sie stellt eine Routine bereit, um zu prüfen, ob jede dieser Queues mindestens einen Lookahead-Wert enthält. Ist dies nicht der Fall, muss die Simulation angehalten werden, weil die Ausführung von Events nicht mehr sicher ist.

Sind alle Queues mit mindestens einem Wert gefüllt, kann der aktuell kleinste Lookahead-Wert abgefragt werden (der kleinste Wert aller kleinsten Werte der Queues). Sind nun Events mit einem kleineren Timestamp als der kleinste Lookahead in der lokalen Event-Queue vorhanden, können diese sicher abgearbeitet werden, da sichergestellt ist, dass jeder andere LP seine lokale Zeit mindestens zu diesem Timestamp fortgeschritten hat. Ist dies nicht der Fall (keine lokalen Events vorhanden), wird bis zum Lookahead gewartet und die lokale Simulationszeit Schritt für Schritt erhöht. Ist ein Lookahead einmal erreicht (Simulationszeit entspricht dem sicheren Timestamp aus den Lookahead-Werten), wird er aus den Queues entfernt (und zwar aus allen, da möglicherweise mehrere LPs den selben Wert verschickt haben).

4 GUI

Für die Anzeige der Simulation, also das Zeichnen der Flugzeuge auf der Karte, wird das 2D-Gaming Framework Slick[2] verwendet. Dieses stellt alle nötigen Klassen zur Erstellung und Rendering eines Spiels zur Verfügung und erlaubt eine einfache Ansteuerung des Spielfeldes (Karte).

In der Simulation wird jeweils der Heimatflughafen auf dem Spielfeld zentriert und die Koordinaten der Objekte (Flughafen und Flugzeuge) werden beim Platzieren von den Koordinaten der gesamten Karte auf das Koordinatensystem des Spielfelds umgerechnet. Zudem wird ein Status-Text mit der aktuellen Rank-Nummer, der aktuellen Simulationszeit (CST) und dem kleinsten aktuellen Lookahead-Wert (oder <none>, falls zur Zeit keiner vorhanden ist).

5 Logger

Um nachvollziehen zu können, was genau in der Simulation auf den verschiedenen LPs abläuft, haben wir eine Logger-Klasse implementiert, welche verschiedene Debug-Meldungen in separaten Log-Dateien schreibt. Diese Log-Dateien werden im Verzeichnis */tmp* abgelegt und heissen *airport-<rank>.log*.

[2] http://slick.cokeandcode.com/

6 MPI-Fix

Da für die Slick-Library gewisse Native Libraries gebraucht werden, werden diese ebenfalls beim Start der MPI-Umgebung mitgeschickt. Damit die LPs bei der Ausführung diese Libraries finden, muss beim Starten der Simulation der *java.library.path* an die Java Virtual Machine mitgegeben werden. Da das Ganze aber über die MPI-Umgebung gestartet wird, ist dies nicht ohne weiteres möglich wie z.B. bei der Angabe der Jar-Files, welches einfach über die Umgebungs-Variable *LD_LIBRARY_PATH* gemacht werden kann.

Aus diesem Grund haben wir die Scripts *lib/p2pmpi/bin/p2pmpirun* und *lib/p2pmpi/bin/p2pclient* so angepasst, dass die Variable *java.library.path* beim Aufruf der Java Virtual Machine gesetzt wird. Es ist also erforderlich, dass die Software mit der mitgelieferten P2P-MPI-Version gestartet wird und, falls die Software auf mehrere Hosts verteilt wird, die benutzte P2P-MPI-Version ebenfalls gepatcht ist.