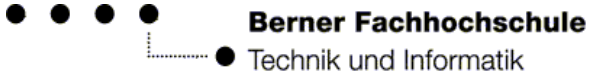

P2PMPI-Beispielprojekt

Aiport Simulation

Ein Beispiel Projekt in Parallel Computing mit P2P-MPI

von Matthias Blaser[†] undStefan Heinemann[*]

22. Januar 2012

[†]blasm5@bfh.ch
[*]heins4@bfh.ch

1 Einleitung

Dieses Dokument beschreibt gewisse Aspekte und Überlegungen im Beispiel Projekt, 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.

2 Messaging

Falls der Zielflughafen eines Flugzeugs nicht denselben Rank besitzt wie der Flughafen, der vom aktuellen LP verwaltet wird, wird am Ende des Start-Vorgangs zusätzlich zum Scheduling des ARRIVAL-Events eine Message an den Zielflughafen versendet, welche die wichtigsten Daten des Flugs enthält. Der 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, gezeichnet werden vom GUI.

Beim Flughafen, von dem das Flugzeug gestartet wird, wird das Flugzeug beim Auslösen des ARRIVAL-Events aus der Simulation entfernt, da es ja jetzt 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 an die verbleibenden LPs ebenfalls eine Null-Message verschickt mit dem selben Timestamp und Lookahead.

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 bereit stellt, wo die jeweiligen Lookahead-Werte abgelegt werden. Sie stellt eine Routine bereit, um zu Prüfen, ob jede dieser Queues mindestens einen Lookahead enthält. Ist dies nicht der Fall,

muss die Simulation angehalten werden, weil die Ausführung von Events nicht mehr sicher ist.

Sind die Queues gefüllt, kann der aktuell kleinste Lookahead-Wert abgefragt werden. Sind nun Events mit einem kleineren Timestamp als der kleine Lookahead vorhanden, können diese sicher abgearbeitet werden. Ist dies nicht der Fall, wird bis zum Lookahead gewartet. Ist ein Lookahead einmal erreicht, wird er aus den Queues entfernt (und zwar aus allen, da möglicherweise mehrere LPs den selben Wert verschickt haben).

4 GUI

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*.

6 MPI-Fix

Da für die Slick-Library gewisse Native Libraries gebraucht werden, werden diese ebenfalls beim Start der MPI-Umgebung mit geschickt. 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 das Script *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.