SIMUL_R PARALLEL

Hardware-in-the-loop Simulation mit Transputern unter Windows

Ronald Ruzicka

SIMUTECH

Wien

Hardware-in-the-loop Simulation hat immer noch den Geruch des experimentellen Probierens an sich, da sie bisher sehr Hardware-abhängig und für den Modellierer unkomfortabel betrieben wurde. SIMUL_R PARALLEL versucht, durch ein modernes, allgemeines Konzept die Problematik der Trennung von Modell und Hardware in den Griff zu bekommen. Exemplarisch wird anhand der Windows Version von SIMUL_R PARALLEL - Oberfläche und Entwicklung unter Windows, Simulation auf Transputern - gezeigt, daß Hardware-in-the-loop Simulation auch modular und wartbar durchgeführt werden kann.

Anforderungen an Hardware-in-the-loop Simulation

Hardware-in-the-loop Simulation (HilS), also das Testen von Hardware, z.B. Steuerungen, durch Anbindung an einen Simulator, z.B. ein Simulationsmodell auf einem Digitalrechner, hat in der heutigen Zeit sehr große Bedeutung. Leider jedoch sind mit HilS (hills, engl. für Hügel) sehr oft "Berge" von Problemen verbunden.

Ein neues Konzept für HilS muß primär versuchen, diese bisherigen Probleme in den Griff zu bekommen. Die Anforderungen an HilS ergeben sich also aus den Erfahrungen, wie bisher HilS betrieben wurde:

- sehr Hardware-abhängig, Modell und Schnittstelle zur Hardware eng vermengt; dies führt bei der Suche nach Fehlern zum häufigen "Suchspiel": Wo steckt der Fehler? (in Hardware und/oder Modell).
- unkomfortabel, da

  das Modell direkt in einer Programmiersprache formuliert ist (fehlende Modellierungsunterstützung),

  oder die Modellierungssprache aus dem gegebenen Anlaß der HilS entwickelt wurde, d.h. notgedrungen eingeschränkt,

  falls eine bestehende Simulationssprache verwendet wurde, eine Speziallösung für eine bestimmte Hardware zur Anwendung gekommen ist. Die HilS-Systeme sind daher oft unübersichtlich und - bei größeren Projekten schwerwiegender - schlecht wartbar.
- die Portierung auf neue, schnellere Hardware erfordert meist völlige Neuentwicklungen.

Daher sollte ein modernes Konzept einer Simulationssprache mit HilS-Fähigkeiten

- eine weitgehende Trennung von Modell und Hardware (siehe [1], Trennung von Modell und Experiment) und damit Hardwareunabhängigkeit ermöglichen und
- eine komfortable Oberfläche zur Modellerstellung und für Simulationsexperimente bieten.

Das Konzept von SIMUL_R PARALLEL

Die für verschiedene Hardware Plattformen verfügbare Simulationssprache SIMUL_R ([2]) ist ein compilierendes System. Die Verwendung von ausführbarem Code - im Gegensatz zu interpretierten Modellen - ist aufgrund der Echtzeitanforderungen an HilS unbedingt nötig.

Ein SIMUL_R Programm mit Modellen wird in C übersetzt und mit einer Library zu einem ausführbaren File gebunden. Modelle können aus mehreren Teilmodellen bestehen, die parallel abgearbeitet werden können. Die Experimente, wie Simulationsläufe, Parametervariationen, Optimierungen, Zeichnungen oder benutzerdefinierte Algorithmen, werden von einem Befehlsinterpreter interaktiv abgearbeitet.

Eine komfortable Entwicklungsoberfläche (z.B. unter MS-Windows), sowie Menüsteuerung und Multiwindowing auf Experimentseite können bei SIMUL_R verwendet werden.

SIMUL_R ist ein offenes System mit mehreren Softwareschnittstellen ([3]). Das *User communications interface* ist eine vordefinierte C-Schnittstelle, die bei Angabe eines Parameters während der Simulation aufgerufen wird.

Damit ist die Basisvoraussetzung für HilS erfüllt: die Einbindungsmöglichkeit von Hardware.

Der modulare Aufbau von SIMUL_R gewährleistet eine schnelle Portierung auf andere Hardware (Modelle können identisch weiter verwendet werden). Die Benutzeroberflächen bleiben bei allen Implementierungen gleich: z.B. weist die reine Windowsversion die gleiche Oberfläche wie SIMUL_R PARALLEL auf Transputern auf, obwohl bei letzterer nur ein File- und Graphikserver unter Windows läuft.

Das Prinzip - der Task

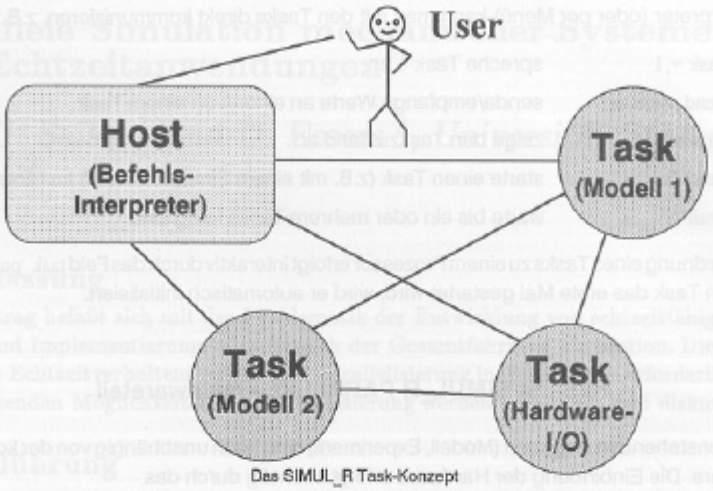
Das Prinzip, das eine portable HilS ermöglicht, wird in SIMUL_R *Task* genannt.

SIMUL_R PARALLEL erlaubt parallele Simulation durch die parallele Abarbeitung der *Modelle* (oder auch I/O-Prozesse mit der angebundenen Hardware) in Tasks.

Ein (virtueller) *Task* ist die Instanz eines Modells auf einem Programm.

Ein *Programm* ist ein fester Hardwareplatz (z.B. ein Prozessor, oder eine I/O-Karte), auf dem ein Task gerechnet werden kann.

Simulationsaufgaben und Hardware-I/O werden also völlig gleichberechtigt als Tasks behandelt.



Das SIMUL_R Task-Konzept

HilS mit SIMUL_R PARALLEL - Modellteil

Der Modell-abhängige Teil der Implementierung eines HilS-Problems in SIMUL_R PARALLEL ist zugleich der Hardware-unabhängige; unabhängig sowohl vom Computersystem, auf dem das Modell gerechnet wird, als auch von der anzuschließenden Hardware selbst.

Ein SIMUL_R Modell besteht aus mehreren Teilmodellen, die mittels als Makros formulierten Befehlen untereinander kommunizieren (diese Befehle beziehen sich immer auf einen Task, repräsentiert durch eine Tasknummer), z.B.:

SEND	sende Werte an einen anderen Task (Modell)
RECEIVE	empfange Werte von einem anderen Task (Modell)
START_TASK	starte ein Modell auf einem anderen *idle* Task (dadurch können Modelle dynamisch hierarchisch strukturiert werden)

Diese Befehle können System-(Modellzustands-)abhängig durchgeführt werden, wodurch große Flexibilität erreicht wird (bei gegebenen Modellen und deren Zuordnung zu Tasks kann man die SEND/RECEIVEs auch automatisch vom Translator im Modell eintragen lassen - hierbei werden die Variablenabhängigkeiten der Modelle untereinander berücksichtigt).

Das Erstellen des Modells erfolgt unter MS-Windows mittels der integrierten Oberfläche SIMMAKE. SIMMAKE bietet unter anderem Projekt-Make-Files und kontextsensitive Hilfe für die SIMUL_R-Modellierungssprache.

Zur Laufzeit kann festgelegt werden, welche Simulationsvariablen von der Hardware gelesen/ beschrieben werden. Pro Task, der HilS-I/O betreibt, gibt es ein .NAM-File, das diese Abhängigkeiten beschreibt (jeweils I/O-Typ, z.B. digital input, und Modellvariable).

Im Interpreter (oder per Menü) kann man mit den Tasks direkt kommunizieren, z.B.:

task = 1	spreche Task 1 an
send, receive	sende/empfange Werte an einen/von einem Task
taskinfo	zeige den Taskzustand an
start &	starte einen Task (z.B. mit einem Simulationslauf) im Hintergrund
wait	warte bis ein oder mehrere Tasks fertig sind

Die Zuordnung eines Tasks zu einem Prozessor erfolgt interaktiv durch das Feld task_processor[]. Falls ein Task das erste Mal gestartet wird, wird er automatisch initialisiert.

HilS mit SIMUL_R PARALLEL - Hardwareteil

Alle obenstehenden Angaben (Modell, Experiment) sind noch unabhängig von der konkreten Hardware. Die Einbindung der Hardware erfolgt einmalig durch das

- Adaptieren der .NAM-Files (Initialisierungstexte für die Hardware, Statusgrößen) und das
- Implementieren des I/O-Tasks. Dieser muß gemäß einem simplen vordefinierten Protokoll reagieren. Er kann sowohl mit den Modellen als auch dem Interpreter kommunizieren, wodurch auch online Daten direkt an die Hardware geschickt werden können. Die Funktionen zum Senden und Empfangen sind in einer Library vorgegeben, es müssen nur Einträge in einer C-Text-Mustervorlage vorgenommen werden.

Zusammenfassung

Das HilS-Konzept von SIMUL_R PARALLEL bietet die geforderte Trennung von Modell und Hardware, also große Hardwareunabhängigkeit. Die MS-Windows/Transputer Implementierung erlaubt die Verwendung von Transputern in einer komfortablen Oberfläche und alle Windowsvorteile (z.B. DDE mit Transputern!). Eine eigene Windowsversion (ohne Transputer) ermöglicht das Testen der Modelle (inklusive I/O). Dadurch kann man mit funktionstüchtigen Modellen auf die Hardware wechseln und somit Fehlerquellen leichter erkennen.

Literatur

[1] Zeigler P.: Theory of Modelling and Simulation. John Wiley, New York, 1976.

[2] Ruzicka R.: Methoden der Simulation auf Parallelrechnern unter SIMUL_R. Proc. 6. Symposium Simulationstechnik, Fortschritte in der Simulationstechnik, Band 1, Vieweg, 1990.

[3] Ruzicka R.: Online Simulation mit SIMUL_R. Proc. 7. Symposium Simulationstechnik, Fortschritte in der Simulationstechnik, Band 4, Vieweg, 1991.