Das Simulationssystem mosis
Parallele Simulation auf PVM - Basis

F. Breitenecker, G. Schuster
Technische Universität Wien, Abt. Simulationstechnik

1. Einleitung

Simulationssprachen dienen zur benutzerfreundlichen Modellbildung und Simulation kontinuierlicher und diskreter Prozesse. Auf dem Gebiet der kontinuierlichen Simulation entwickelten sich Simulationssprachen aus der ,,Simulation“ eines Analogrechners auf einem Digitalrechner. Heute haben Simulationssprachen für kontinuierliche Prozesse Umgebungen verschiedenster Form, weswegen heute Bezeichnungen wie *Simulator*, *Simulationssystem* etc. verwendet werden. Grundlage der Modellbeschreibung sind nach wie vor Systeme von Differentialgleichungen bzw. von Differential-Algebraischen Gleichungen.

An der Abteilung Simulationtechnik des Institutes für Technische Mathematik wurde vor geraumer Zeit in Zusammenarbeit mit dem damaligen Hybrid- bzw. Simulationsrechenzentrum in Forschungsprojekten die Entwicklung von (kontinuierlichen) Simulationssystemen begonnen. Ein Meilenstein in dieser Entwicklung war das System HYBSYS, das sich von einem Hybridrechner-Simulator [1,2] zu einem digitalen Simulator weiterentwickelte ([3]). In diesen Projekten wurde eine Menge von Know-How angesammelt, insbesondere was Methodologie, parallele Strukturen und ,,normierte“ Modelldarstellung anbetrifft. Dieses Know-How floß in das mosis-Projekt [4] ein, einige der erarbeiteten Konzepte und Strukturen konnten zudem erst mit dem heutigen Stand der Softwaretechnik und u.a. mit den Möglichkeiten von PVM sinnvoll und vernünftig implementiert werden.

Mit Hilfe des PVM-Konzeptes wurde im Rahmen eines neuen Simulationsprojektes an der Abteilung für Simulationstechnik als Fortsetzung der bisherigen Simulatorprojekte das Simulationssystem mosis - *modular simulation system* - für die Simulation und Analyse kontinuierlicher Prozesse auf Multiprozessorsystemen entwickelt [4,5]. Dabei wurden das Konzept einer modellunabhängigen Parallelisierung ([6,7]), das Modell-Methode-Experiment - Konzept ([8]) und das Model Interconnection Concept ([5,9]), eine moderne Form der blockorientierten Darstellungsweise von Modellen, implementiert.

Eine Parallelisierung von Simulationsaufgaben (Experimenten) kann in mosis auf verschiedenste Weise erfolgen. Für Parallelisierung erweist sich vor allem das PVM-Konzept als zielführend. Aufgrund der Eigenschaften von PVM kann mosis auch auf Einprozessorsystemen verwendet werden, womit es auch als ,,normale“ Simulationssprache nach CSSL-Norm angesehen werden kann.

2. Struktur und Funktionsweise von mosis

mosis ist ein frei verfügbares (*freeware*) Programmpaket und kann auf vielen verschiedenen Hardwareplattformen eingesetzt werden; die Struktur des dahinterliegenden Computersystems (parallel oder seriell) bleibt dem Benutzer verborgen. Im Falle von Einprozessorsystemen wie z. B. dem Fachbereichsrechner Mathematik-Studierende (fbma.tuwien.ac.at) täuscht das PVM-System eine Parallelrechnerarchitektur vor, die jedoch nur aus einem einzelnen Prozessor besteht: alle Prozesse werden nur auf dem einzigen verfügbaren Prozessor erzeugt. Generell ist es jedoch möglich, verschiedene andere Workstations dynamisch dem PVM-Cluster hinzuzufügen (auch heterogene Parallelrechnerarchitekturen sind hier möglich), um auf diese Weise ein echtes Multiprozessorsystem zu erhalten.

mosis ist ähnlich wie das häufig verwendete ACSL (ebenfalls auf fbma) eine CSSL-Simulationssprache (Continuous System Simulation Language, Standard für Simulationssprachen), basiert aber im Gegensatz zu dieser nicht auf FORTRAN, sondern auf (ANSI-)"C". In CSSL-Sprachen werden für jedes Modell die systembeschreibenden Differentialgleichungen (mit Anfangswerten, Angabe von Beschränkungen, Unstetigkeiten, externen Eingängen und Ausgängen etc.) in *Sections* formuliert. Die *Initial Section* beschreibt Initialisierungen und statische Berechnungen vor einer Simulation. Die *Dynamic Section* regelt die (periodische) Ausgabe von Werten und Übernahme neuer Eingangswerte und etwaige zusätzlichen Berechnungen, z. B. Synchronisation für Realtime - Aufgaben; sie steuert über das Kommunikationsintervall den Start der Integration und bietet sich damit generisch für einen Datenaustausch bei paralleler Simulation an. In der *Derivative Section* werden die systembeschreibenden Differentialgleichungssysteme angegeben, wobei je nach Sprache unterschiedlicher Komfort geboten wird. Die *Discrete Sections* erlauben die Formulierung von *Ereignissen*, die bei zeit- und zustandsabhängigen Modellunstetigkeiten ausgeführt werden sollen (u.a. Strukturen für diskrete Regler). Die *Terminal Section* ermöglicht statische Berechnungen mit den Endwerten eines Simulationslaufes.

Grafische Modellbildungsoberflächen bilden ebenfalls auf diese Modellbeschreibungsstruktur ab, wobei allerdings fast immer nicht alle Möglichkeiten zur Beschreibung (*Sections*) ausgeschöpft werden.

mosis bietet zur Modellbeschreibung all diese Strukturen an. Ein oder mehrere mosis-Modelle (mit einem beliebigen Texteditor erstellt) werden nach "C" übersetzt und kompiliert, danach mit der Laufzeitbibliothek von mosis und von PVM gelinkt, womit ein ausführbares Simulationsprogramm entsteht.

Im Gegensatz zu ACSL ist es in mosis möglich, mehrere Modelle gleichzeitig zu einem Programm zu verbinden, das bei Vorhandensein mehrerer Prozessoren echt parallel abgearbeitet werden kann. Das theoretische Konzept hinter der

Parallelisierung in mosis ist das *Model Interconnection Concept* (*MIC*, [5,9]), das die Entwicklung eines Simulationsmodells aus verschiedenen unabhängigen Modellteilen beschreibt. mosis orientiert sich damit weniger an einer Parallelisierung einer bestehenden Modellbeschreibung, sondern sieht ein Gesamtmodell als ein System von im Prinzip unabhängigen Modellen, die zur Laufzeit geeignet miteinander verbunden werden können, mit jederzeit möglichen Änderungen der Verbindungen und jederzeit möglichem Hinzufügen weiterer Modellteile. Abbildung 1 zeigt, daß diese Verbindungen durchaus auch Rückkoppelungen sein können und daß einzelne Modelle mehrfach (mehrfache *Instanzen*) auftreten können, womit mosis generisch auch jede Art von Teilmodellkonzepten erlaubt.

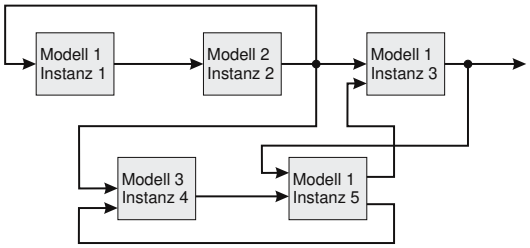

Abb.1: Model Interconnection-Konzept (*MIC*) in mosis

Bestehende große Modelle werden sinnvollerweise durch eine anwendungsorientierte Zerlegung in Teilmodelle strukturiert und parallel abgearbeitet, und nicht primär durch eine Zerlegung des Zustandsraumes. Grundsätzlich erfolgt ein Datenaustausch durch die automatische Synchronisation über das (auch dynamisch änderbare) Kommunikationsintervall der *Dynamic Section*; innerhalb des Kommunikationsintervalles kann zwischen verschiedenen Integrationsalgorithmen, Integrationsschrittweiten und Interpolationen gewählt werden.

Im *MIC* gilt jedoch auch ein wesentlich allgemeinerer Modellbegriff: ein Modell kann neben der ,,herkömmlichen“ Bedeutung z. B. auch durch Daten oder durch ein Interface zu einem externen System (Hardware-in-the-Loop, Man-in-the-Loop, Simulatorkopplung) bestimmt werden. Ein Gesamtmodell kann also aus teilweise sehr verschiedenen ,,Modellen“ bestehen (Abb.2).

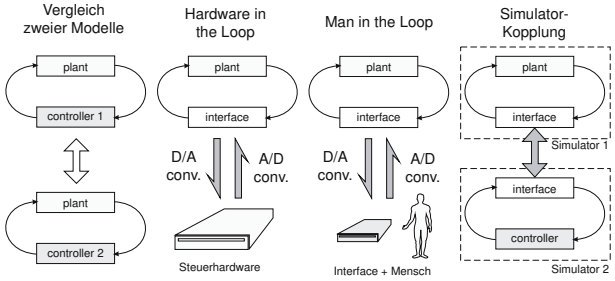

Abb.2: Allgemeiner Modellbegriff im *MIC*

Moderne Simulatoren bieten neben der Zeitbereichsanalyse (der numerischen Lösung der systembeschreibenden Gleichungen) zumindest auch noch die Berechnung eines stationären Zustandes und/oder eine Linearisierung (für eine nachfolgende Frequenzbereichsanalyse) an, wobei allerdings die Integration im Zeitbereich immer die Hauptaufgabe bleibt. mosis baut auf dem *allgemeineren Model-Method-Experiment*-Konzept (*MME*-Konzept) auf, das ein Experiment als die Anwendung einer Methode auf ein oder mehrere Modelle sieht ([8], Abb.3). In diesem Konzept stehen die Integration im Zeitbereich, die Linearisierung, eine symbolische Modellmanipulation etc. als Methoden gleichberechtigt nebeneinander. Jede Methode findet in den in *Sections* strukturierten Modellbeschreibungen die notwendigen Informationen. Das Konzept erlaubt auch eine Rekursivität der Methoden und Experimente und verschiedene Stufen einer modellunabhängigen Parallelisierung ([6,7]).

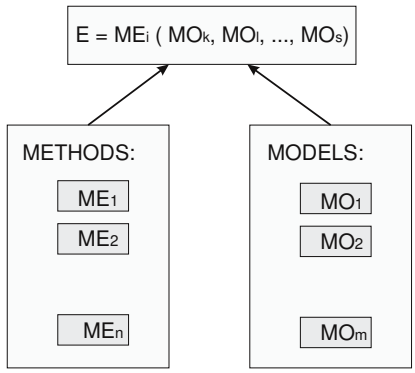

Abb.3: Das *Model-Method-Experiment* - Konzept (*MME* - Konzept)

mosis arbeitet nach diesem Konzept und ermöglicht daher parallele Arbeitsweisen auch auf Ebene von Methoden - einfache Beispiele sind Parameterstudien (mehrfache Simulationsläufe desselben Modelles), Linearisierungen an unterschiedlichen Orten, komplexere sind Monte-Carlo-Simulationen und Sensitivitätsanalysen.

Durch dieses Konzept können in mosis nicht nur verschiedene verbundene Modelle parallel abgearbeitet werden, sondern auch parallel mehrere *Instanzen* eines Modells, ohne zwischen den Modellen Daten auszutauschen (sowie auch Mischformen).

Wie andere CSSL-Simulatoren besteht das gesamte Programmpaket von mosis aus zwei grundlegenden Teilen (Abb.4):

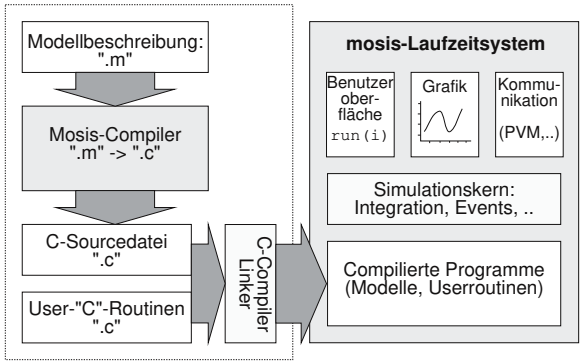

Abb. 4: Softwarestruktur von mosis

- dem Übersetzerteil, der die textmäßig definierten Modellbeschreibungen sowie die benutzerdefinierten "C"-Funktionen in ein ausführbares Programm kompiliert, und
- der Laufzeitoberfläche: hier können Modelle instanziert und Experimente damit durchgeführt werden. Dafür sind eine komfortable Interpretersprache sowie flexible Funktionen zur Datenverwaltung und -ausgabe verfügbar.

3. Anwendungen

mosis ist ein experimentelles System und erlaubt eine Vielfalt von verschiedenen Modellbeschreibungsarten und -techniken. Drei einfache Beispiele sollen Anwendungen in verschiedenen Bereichen zeigen. Zwei davon beschäftigen sich mit ,,parallelen" Modellen, während das dritte eine Methodenparallelisierung zeigt.

3.1 Regelkreis

Dieses aus dem ACSL-Manual stammende Beispiel zeigt einerseits die Grundphilosophie von mosis, nämlich die Betrachung eines Modelles als Verbindung von im Prinzip unabhängigen Teilmodellen, und andererseits, daß der volle CSSL-Standard von mosis die einfache Adaption in anderen CSSL-Sprachen existierender Modelle erlaubt (u.a. umfaßt mosis die erweiterten CSSL-Macro-Features).

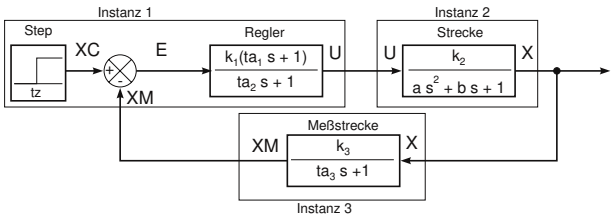

Abb.5: Regelkreis, Modellaufteilung in mosis

Das Modell (Abb.5) besteht aus drei unabhängigen Modellen, nämlich aus einer Strecke, einem Verzögerungsglied zweiter Ordnung (in ACSL CMPXPL-Macro), einer Meßstrecke, einem Verzögerungsglied erster Ordnung (in ACSL REALPL-Macro) und einem Regler, einem Verzögerungsglied erster Ordnung mit D-Anteil (in ACSL LEDLAG-Macro). Der Eingangssprung ist einfach als Funktion definiert, er könnte auch als selbständiges Modell repräsentiert werden.

Übertragungsfunktionen, wie in der Regelungstechnik als Modellbeschreibungsform verwendet, können in mosis durch Macros beschrieben werden (entsprechend den Systemmacros in ACSL). Dabei wird für die Übertragungsfunktion die entsprechende Differentialgleichung definiert. In mosis können Differentialgleichungen sowohl mit dem INTEG-Operator als auch mit dem ,,Ableitungsstrich" angegeben werden, ,,mehrfache" Integrationen sind möglich; zudem können die Differentialgleichungen skalar oder vektoriell sein. Die `step`-Funktion ist als ein `#define`-symbol definiert:

```
#macro cmpxpl(y,p,q,x,ic1,ic2)
  y`=((x)-(y)-(q)*y)/p,ic2,ic1;
#endm
#macro ledlag(out,y,p,q,x,ic)
  out=y+(p)*(x)/q;
  y`=(x-out)/(q),ic;
#endm
#macro realpl(y,p,x,ic)
  y`=(x-(y))/(p),ic;
#endm
#define step(tz)     ((t<tz)?0.0:1.0)
```

Die folgenden unabhängigen Modellbeschreibungen werden gemeinsam übersetzt (nach vorhergehender Ersetzung der Macros) und mit dem Laufzeitsystem gelinkt:

```
model regler(input xm;output u) {
  state y2;
  param tz=0.02, kl=50.0, ta1=0.02, ta2=0.005;
  double xc,h1,e;
  dynamic  {
    derivative  { xc=step(tz); e=xc-xm;
                  ledlag(h1,y2,ta1,ta2,e,0.0);
                  u=k1*h1; } } }

model strecke(input u;output x) {
  state y3;
  param k2=0.5, a=0.012, b=0.2;
  dynamic  {
    derivative  { cmpxpl(y3,a,b,u,0.0,0.0);
                  x=k2*y3;  } } }

model meszstrecke(input x;output xm) {
  state y1;
  param k3=1.0, ta3=0.002;
  dynamic   {
    derivative  { realpl(y1,ta3,x,0.0);
                  xm=k3*y1; } } }
```

Zur Laufzeit werden nun diese drei Modelle *instanziert* und zum Gesamtmodell verbunden. Die Instanzierung legt fest, daß das Modell `regler` auf dem Prozessor mit der Nummer 1 abgearbeitet und mit der Prozeß-ID `r` identifiziert wird, das Modell `strecke` auf dem Prozessor mit der Nummer 2 und mit der Prozeß-ID `s`, etc.

Die Verbindung zum Gesamtmodell erfolgt im `connect` Befehl, wo die Variablen über Prozeß-ID und Namen im Modell identifiziert werden (Owner-Konzept):

```
r = instance ("regler", 1);
s = instance ("strecke", 2);
m = instance ("meszstrecke", 3);
connect(r.u, s.u); connect(s.x, m.x);
connect(m.xm, r.xm);
```

Ein `watch` Befehl legt fest, daß bei folgenden Simulationsläufen die Variablen `u, x` und `xm` gespeichert werden. Der `run` Befehl startet die Simulation mit dem Prozeß `r` und automatisch die verbundenen Prozesse `s` und `m`. PVM erledigt automatisch alle Kommunikation mit einem Standard-Kommunikationsintervall:

```
// Abspeicherung waehrend Simulation
watch(r.u); watch (s.x); watch (m.xm);
// Simulation der Prozesse r, s, m
run(r);
```

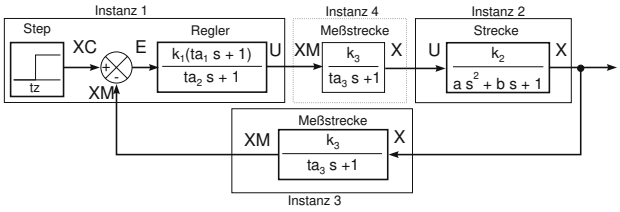

Abb.6: Erweiterter Regelkreis, Modellaufteilung in mosis

Strukturelle Änderungen sind einfach. Die folgenden Befehle fügen eine weitere Meßstrecke zwischen Regler und Strecke (Abb.6):

```
// Loeschen der Verbindung r - s
disconnect(r.u, s.x);
// zweite Instanzierung von "meszstrecke"
m2 = instance ("meszstrecke", 4);
// Hinzufuegen der neuen Verbindungen
connect(r.u, m2.x); connect(m2.xm, s.u);
```

Die Wahl des Kommunikationsintervalles generell erfolgt durch den Benutzer, ebenso die Wahl der Integrationsalgorithmen und deren Schrittweiten und etwaige Inter-

polationen. Zu große Intervalle führen zu Fehlern, im gegenständlichen Fall haben sie den Effekt eines diskreten Reglers. Die Schrittweitenwahl (Anfangsschrittweiten bei schrittweitensteuernden Algorithmen) erfolgt nach den üblichen Kriterien; mit Kommunikationsintervallen kann experimentiert werden, um im Falle einer echt parallelen Simulation die Simulationsdauer zu vermindern.

Wird dieses Beispiel - mit denselben Befehlen - auf einem Einprozessorsystem gerechnet, so wird das Vorhandensein der parallelen Prozessoren simuliert, indem unterschiedliche Prozesse auf dem einem Prozessor arbeiten.

3.2 Diskretisierung einer Partiellen Differentialgleichung

Dieses Beispiel zeigt die Zerlegung eines großen Modelles in sinnvolle Teilmodelle und die Vekor-Features in mosis. Ein schwingendes Seil der Länge L, das an einem Ende fixiert und am anderen Ende periodisch erregt wird, wird in erster Näherung beschrieben durch die partielle Differentialgleichung

$u_{xx}(t,x) = a\, u_{tt}(x,t)$

$u(0,t) = 0,\ u(L,t) = b\, e^{-dt} \sin\omega t,\ u(x,0) = u_x(x,0) = 0$

Diskretisierung der Ortskoordinate in N äquidistante Intervalle erzeugt ein schwach verkoppeltes System von gewöhnlichen Differentialgleichungen zweiter Ordnung für die Linien $u(x_i,t) = u_i(t)$:

$u_i'' = \alpha u_{i-1} - 2\,\alpha u_i + \alpha u_{i+1},\ u_i(0) = u_i'(0) = 0,\ i = 1, .. N-1$

Dieses System kann auf einem Multiprozessorsystem implementiert werden. Dafür wird die Gesamtanzahl der Diskretisierungslinien (z. B. N=800) durch M (=Anzahl der Prozessoren) geteilt: Jeder Prozessor muß nun nur mehr N/M Linien berechnen; nach jedem Kommunikationsschritt / Integrationsschritt werden die Daten mit den benachbarten Prozessen ausgetauscht. Zunächst die Beschreibung des seriellen Modells in mosis, das auch die Grundlage für die parallelen Modelle darstellt:

```
#define N 800
model rope()  {
  int i;  param l=4.0, a=2.0, b=1.0;
  param omega=1.0, d=0.2;
  double ak,ak2,ts,ul;
  state vector U[N],DU[N];
  vector U0[N],DDU[N],DU0[N];
  tend=30.0;   // Laufzeit
  cint=dt=0.005;   // Komm. Schrittweite
  initial { ak=((N+1)*(N+1))/(l*l*a);
  ak2=2*ak;  U0=0;  DU0=0; }
    derivative {
      ul=b*exp(-d*t)*sin(omega*t);
      proc(DDU;U,ul)  // procedural block
        {DDU[0]=ak*U[1]-ak2*U[0];
         for(i=1;i<N;i++) DDU[i]=ak*U[i+1]-
            ak2*U[i]+ak*U[i-1];
         DDU[N-1]=ak*U[N-2]-ak2*U[N-1]+ak*ul; }
      U``=DDU,DU0,DU0;  //  Vektor-Diff.gl. } }
```

Dieses Modell kann parallelisiert werden, indem man drei verschiedene Modelle `rope_first, rope_mid` und `rope_last` definiert, die jeweils die ersten K (=N/M) Linien, jeweils K mittlere Linien (M-2 - mal) und die letzten K Linien berechnen. Von `rope_mid` müssen mehrere *Instanzen* (M-2) zur Laufzeit erzeugt werden (Abb.7). Die Definitionen für diese drei Modelle enthalten nun die Verbindungslinien als Eingänge bzw. Ausgänge:

```
model rope_first(input i1;output o1) { ... }
model rope_mid(input i1,i2;output o1,o2) { ... }
model rope_last(input i1;output o1) { ... }
```

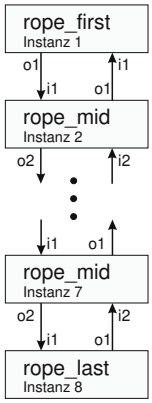

Abb.7: Parallelisierung eines schwach verkoppelten Systems in mosis

Die Eingangs- und Ausgangsgrößen werden in den Modellen über die Zustandsgrößen bestimmt, z.B. in `rope_first`:

```
...
DDU[0]=ak*U[1]-ak2*U[0]+ak*i1; ....;
o1=U[0];
...
```

Zur Laufzeit werden die Modelle instanziert und verbunden (das Modell `rope_mid` wird sechsmal instanziert!). Im Laufzeitsystem wird eine C-ähnliche Interpretersprache verwendet:

```
int p[8],i;
// Eine Instanz auf Prozessor 1
p[0]=instance("rope_first",1);
// 6 Instanzen
for(i=1;i<7;i++) p[i]=instance("rope_mid",1+i);
p[7]=instance("rope_last",8);
... // Setzen der Systemvariablen und Parameter
// Verbinden der Modelle:
connect(p[0].o1,p[1].i1);
connect(p[1].o1,p[0].i1);
for(i=1;i<6;i++) {connect(p[i].o2,p[i+1].i1);
                  connect(p[i+1].o1,p[i].i2); }
connect(p[6].o2,p[7].i1);
connect(p[7].o1,p[6].i2);
// Speichern der jeweils ersten Kurve pro Modellteil
for(i=1;i<8;++) watch(p[i].U[0]);
// Start der Simulation:
run(p[0]);
```

Mit Hilfe dieser Aufteilung auf parallele Prozessoren ist eine beachtliche Geschwindigkeitssteigerung möglich (bis zu einem Faktor 6.5 bei 8 Prozessoren auf einem IBM RS/6000-Cluster). Weitere Beschleunigung (Verbesserung des Speed-Up Faktors ***f***) kann auch durch Vergrößerung des Kommunikationsintervalles erreicht werden, indem z.B. die Kommunikationsschrittweite c_{int} als ein Vielfaches $c_{int} = \gamma h$ der Integrationsschrittweite h gewählt wird:

Anz.Linien / Kommunikation	$c_{int} = h$	$c_{int} = 2h$	$c_{int} = 4h$
N = 600	$f = 5.3$	$f = 5.8$	$f = 6.1$
N = 800	f = 5.7	$f = 6.0$	$f = 6.4$

Dieses Beispiel wurde in der Zeitschrift ***EUROSIM Simulation News Europe*** (***SNE***) als „Parallel Comparison“ ausgeschrieben ([10,11,12]). Es wurden bereits einige interessante Lösungen eingesandt, die hier vorgestellte Lösung mit mosis und PVM stellte sich im Vergleich als effektiv heraus.

3.3 Parametervariation

Dieses Beispiel zeigt eine einfache Methodenparallelisierung. In einem (sehr steifen) Modell wird eine Parameterva-

riation parallelisiert, was zu nahezu linearem Speed-Up führt. Das Modell kommt aus der Festkörperphysik und beschreibt Entstehung und Zerfall von bestimmten Aggregaten in Alkalihaliden unter Elektronenbeschuß und wurde ebenfalls in ***SNE*** als ,,Software Comparison“ ausgeschrieben ([12,13,14]). Die Beschreibung der Gleichungen ist dem mosis-Modell zu entnehmen:

```
model lithium()  // keine E/A-Variablen {
 state r.m,f;
 param kr=1.0, kf=0.1, lf=1000.;
 param dr=0.1, dm=1.0;
 param f0=9.975, m0=1.674, r0=84.99, p=0.0;
  preinitial // Initialisierungen   {
   ialg=8;  // steifer Integrationsalgorithmus
   tend=10.0;   dt=cint=0.001; }
  derivative  {
   r`=-dr*r+kr*m*f, r0;
   m`= dr*r-dm*m+kf*f*f-kr*m*f, m0;
   f`= dr*r+2*dm*m-kr*m*f-2*kf*f*f-lf*f+p, f0;  } }
```

Nach Übersetzungs- und Linkvorgang wird das Modell auf Laufzeitebene initialisiert:

```
// Instanzierung auf irgendeinem Prozessor (-1)
int m; m=instance("lithium",-1);
// Speichern von f und Simulationslauf
watch(m.f); run(m);
```

Eine Variation des Parameters lf von 1.0E2 bis 1.0E4 wird über eine Schleife durchgeführt:

```
double x[7]={ 100, 200, 500, 1000, 2000, 5000, 10000 };
for(i=0;i<7;i++) { m.lf=x[i]; run(m) }; drawcurve(m.f)
```

Eine Parallellisierung dieser Parameterstudie kann durch sieben Instanzierungen des Modelles auf unterschiedlichen Prozessoren ausgeführt werden:

```
// 7 Instanzen
int i; int la[7];
for(i=0;i<7;i++) la[i]=instance("lithium",i+1);
// Abspeichern von f
for(i=0;i<7;i++) watch(la[i].f);
// Parameterzuweisung, parallele Simulation
for(i=0;i<7;i++) { la[i].lf=x[i]; run(la[i]); }
// Zeichnen
for(i=0;i<7;i++) drawcurve(la[i].f);
```

Abbildung 8 zeigt diese Parallelisierung und weist auf die interne Arbeitsweise von mosis in PVM hin: Simulationen werden immer im Hintergrund durchgeführt, kontrolliert vom Master-Prozeß, auf dem das Laufzeitsystem arbeitet. Damit werden im betrachteten Beispiel alle 7 Instanzen gleichzeitig simuliert (parallel), was sich in nahezu linearem Speed-Up auswirkt.

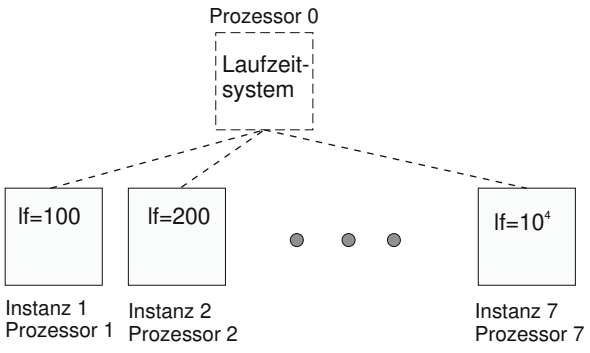

Abb. 8: Parallelisierung einer Parameterstudie in mosis

Wird die parallele Parametervariation bei weniger als 7 Prozessoren versucht, so wird die fehlende Parallelität vom PVM ,,simuliert“, indem einige Prozessoren eben mehr als einen Simulationslauf durchführen müssen.

4. Features von mosis

mosis ist ein experimentelles System. In laufenden Arbeiten werden Ergänzungen und Erweiterungen getestet (und auch verworfen). Die wesentlichen derzeit implementierten Eigenschaften und Features sind:

- CSSL-Beschreibungssprache auf "C"-Basis mit den üblichen Sections. Ein- und Ausgangssignale zur Modellzusammenschaltung. Beliebiger "C"-Code kann in einem Modell definiert werden.
- Modulare Modellbeschreibung: Verschiedene Teile können in verschiedenen Modellen (auch Dateien) beschrieben und zur Laufzeit verbunden werden.
- Einfache Behandlung von Matrizen und Vektoren innerhalb des Modells.
- Rechnet automatisch mit doppeltgenauen Zahlen (*double*).
- Mächtige Macrosprache für einfache Modellbeschreibung sowie Übertragung von ACSL-Modellen nach mosis in Form eines Präcompilers, der eine Übermenge des "C"-Präprozessors darstellt und auch gesondert verwendet werden kann.
- Definition von ODEs von höher als erster Ordnung (automatische Transformation) möglich.
- Komplettes "make"-Utility übernimmt den gesamten Übersetzungsvorgang (Übersetzung nach "C", Compiler, Linker etc.); sehr schnelle Übersetzung.
- Objektorientiertes Simulationskonzept: von einem Modell muß vor der Simulation eine Instanz erzeugt werden, die die Modelldaten enthält. Mehrere Instanzen können gleichzeitig (auch auf Parallelrechnern) berechnet werden (auch von einem einzelnen Modell).
- Die Simulation erfolgt immer im Hintergrund: während des Laufs ist es möglich, den derzeitigen Status (Zeit, Variablen) abzufragen, die Simulation anzuhalten oder neue Instanzen zu erzeugen.
- Dynamische Zusammenschaltung von Modellen auf Laufzeitebene.
- Komfortable Laufzeitinterpretersprache, "C"-ähnlich.
- Kommandofiles können gelesen werden (übernehmen komplexere Befehlsfolgen), Aufruf des Editors und des Betriebssystems aus der Oberfläche heraus.
- Benutzerdefinierte Funktionen: entweder auf "C"-Ebene (wird übersetzt und zum Programm gelinkt) oder in der eingebauten Interpretersprache für Definition von Optimierungsverfahren, Grafik etc.
- Behandlung von Funktionstabellen: Arithmetische Operationen, Einfügen & Löschen von einzelnen Punkten etc.
- Möglichkeit für Hardware-in-the-Loop-Programmierung.
- Verschiedene Integrationsverfahren verfügbar: Euler, Runge-Kutta (2,3,4,6), RK-Fehlberg, Adams-Moulton, Stiff Systems Solver.
- Genaue Suche von zustandsabhängigen Unstetigkeiten im Modell mittels modifizierter Regula falsi.
- Verschiedene Grafikfunktionen: Phasenplot, logarithmische Darstellung etc.; benutzererweiterbar.
- Hilfesystem: Erklärungen zu den Laufzeitbefehlen und Funktionen.
- Initialisierungsfile ermöglicht das automatische Setzen benutzerspezifischer Einstellungen.
- Macros für häufig ausgeführte Befehle können leicht definiert werden.
- Bis zu 32 Grafikfenster darstellbar, Grafiken können auch auf Drucker/Plotter ausgegeben werden (HPGL, PostScript) sowie von anderen Programmen weiterverarbeitet werden.

5. Verfügbarkeit, Dokumentation

mosis ist als experimentelles Simulationssystem gedacht und steht frei zur Verfügung. Durch die parallelen Features und die PVM-Basis ist mosis im Prinzip an UNIX Plattformen gebunden. Allerdings können auch andere Kommunikationssysteme (z.B. Linda) verwendet werden. Vor allem für Lehrzwecke werden parallel zur UNIX-Version auch zwei PC-Versionen unterstützt, bei denen die Parallelisierung durch kooperatives Multitasking simuliert wird.

Folgende Versionen von mosis sind derzeit verfügbar:

- Zwei Versionen für PCs auf Basis von Intel-Prozessoren unter DOS, abhängig vom verwendeten C-Compiler.

Diese beiden Versionen sind im bereits benutzerfertigen Paket verfügbar, das aus Übersetzer, Make-Utility und Laufzeitbibliothek besteht und sofort verwendet werden kann. Beide Versionen unterstützen kooperatives Multitasking, d.h. auch hier können mehrere Modelle ,,parallel“ berechnet werden, was im Hintergrund geschieht:

- 32 Bit-Version unter Watcom C/C++ Version 10.0 mit DOS-Extender: min. 80386, mind. 2 MB RAM und numerischer Koprozessor, VGA-Karte
- 16 Bit-Version unter Borland C Version 3.1: 640 KB RAM, numerischer Koprozessor, VGA

- Die UNIX-Version ist im Sourcecode verfügbar (enthält auch den Source für die PC-Versionen) und benötigt zur Übersetzung einen ANSI-C-Compiler (auch "cc" möglich, falls dieser bestimmte ANSI-Konstruktionen wie "void"-Funktionen etc. versteht); sowie das PVM-System (Version mindestens 3.3.4). Für die Übersetzung werden bereits Makefiles mitgeliefert, die mit den entsprechenden Pfadnamen ausgestattet, lediglich ausgeführt werden müssen. Vorinstalliert ist mosis auf dem
  - Fachbereichsrechner Mathematik-Studierende (fbma; nur ,,virtuelle“ Parallelität), sowie auf dem
  - RS/6000-Cluster des EDV-Zentrums (9 Maschinen unter PVM).
- mosis wurde auch bereits für eine "Cogent XTM"- Transputer-Workstation implementiert, wo sich die Verwendbarkeit von mosis unter einem anderen Kommunikationssystem (Linda) unter Beweis stellte.
- Eine Adaptierung für MS-Windows (Win32s) ist in Vorbereitung.

Derzeit in der Testphase befindet sich eine X Window-Oberfläche für mosis, die eine komfortable Eingabe sowie graphische Ausgabe ermöglicht (bis zu 32 Grafikfenster gleichzeitig). Diese wird in den nächsten Wochen zur Verfügung gestellt werden (auch im Sourcecode).

Dank des modularen Aufbaus des Sourcecodes ist eine Adaption an andere Hardwareumgebungen und Kommunikationssysteme sowie Grafikstandards leicht durchführbar: so muß lediglich ein "C"-File neu geschrieben werden, das die entsprechenden Funktionen enthält; alle anderen Dateien bleiben gleich.

Die komplette Dokumentation für mosis ist ebenfalls in Form einer PostScript-Datei verfügbar. Alle diese Programme und Daten können vom Simulations-Softwareserver `simserv.tuwien.ac.at` (anonymous ftp) unter dem Verzeichnis `software_argesim/mosis` geholt werden. Weitere Informationen zu mosis (Vergleiche, etc.) finden sich in Veröffentlichungen (u.a. in [4,5]) und in ***EUROSIM Simulation News Europe***, herausgegeben von F. Breitenecker und I. Husinsky, sowie im Bericht zum ***Seminar aus Modellbildung und Simulation - Parallele Simulation, mosis“*** *(*EDV-Zentrum, TU Wien, Jänner 1995).

Gewartet wird mosis von der ***ARGE Simulation News***, die auch die Zeitschrift SNE betreut, die ***Seminare aus Modellbildung und Simulation“*** mitveranstaltet, den Server mit Demos von Simulationssoftware (FTP-Server `simserv.tuwien.ac.at`) sowie den Informationsserver `eurosim.tuwien.ac.at` (Gopher Server) betreibt und den Kongreß ***EUROSIM’95*** an der TU Wien ausrichtet (11. - 15. 9. 1995). Bei dieser Tagung wird sich eine spezielle Session mit der Parallelisierung von Simulationsaufgaben beschäftigen. Für mosis ist in der ***ARGE Simulation News*** Herr G. Schuster verantwortlich.

Für weitere Informationen wenden Sie sich an ass.Prof. DI.Dr.F. Breitenecker (fbreiten@email.tuwien.ac.at) oder DI.Dr. Günter Schuster (guenter@ws1.atv.tuwien.ac.at) oder allgemein an argesim@simserv.tuwien.ac.at. Informationen finden sich auch in den READ.ME-Files am Simulations-Server, sowie am Gopher Server von ***EUROSIM*** unter ARGESIM-Software (<URL: gopher://eurosim.tuwien.ac.at>; ab März WWW-Server <URL: http:// eurosim.tuwien.ac.at>). Gedankt sei Herrn Dr. W. Weisz von der Abt. Hochleistungsrechnen des EDV-Zentrums für seine Unterstützung bei der PVM-Installation von mosis.

6. Weitere Entwicklungen

Für die nächste Zeit sind noch einige größere Weiterentwicklungen vorgesehen, wie z.B. : objektorientierte Modellentwicklung, implizite Modelldefinition (DAE-Systeme), halbautomatische Bearbeitung von partiellen Differentialgleichungen, Verbesserung der Kommunikation (PVM-Tuning), Vereinheitlichung der grafischen Oberfläche, Hardware-in-the-Loop Interface, Einbeziehung des VHDL-Standards, Mitentwicklung und Miteinbeziehung des AHDL-Konzeptes

Literatur

[1] W. Kleinert, D. Solar, F. Berger: Status Report on TU Vienna’s Hybrid Time Sharing System. In (Ed. W. Ameling) Proc. 1st European Simulation Congress, Informatik-Fachberichte 71, Springer (1983), 193 - 200.

[2] D. Solar, F. Berger, A. Blauensteiner: HYBSYS - Interactive Simulation Software for a Hybrid Multiple User System. Informatik Fachberichte Bd. 56, Springer (1982), 257 - 265.

[3] F. Breitenecker, D. Solar, I. Husinsky: HYBSYS - a New Simulation System. In (Ed.D.Murray-Smith) Proc. EUROSIM’89 Simulation Congress Edinburgh, SCS Europe Publishing (1989), 275 - 281.

[4] F. Breitenecker, G. Schuster: Modular Parallelization in the Simulation Project HYBSYS: the mosis System. In (Eds. F.Maceri, G. Iazeolla) Proc. EUROSIM’92 Simulation Congress, Elsevier (1992), 129 - 134.

[5] Parallelisierung mit dem Modellverbindungskonzept und Implementierung in der Simulatonssprache mosis. In (Hrsg. A. Sydow) Syposium Simulationstechnik, Fortschritte in der Simulationstechnik Bd. 6, Vieweg (1993), 487 - 490.

[6] G. Schuster, I. Husinsky, F. Breitenecker: Modellunabhängige Parallelisierung in kontinuierlicher Simulation. In (Hrsg. Dj. Tavangarian) Symposium Simulationstechnik, Fortschritte in der Simulationstechnik Bd. 4, Vieweg (1991), 29-39.

[7] F. Breitenecker, G. Schuster, J. Fritscher, I. Husinsky: Parallelisation of Simulation Tasks. In (Hsg.H.P.Zima) Parallel Computation, Lecture Notes in Computer Science vol. 591, Springer (1992), 412 - 425.

[8] F. Breitenecker: Models - Methods - Experiments - A new Structure for Simulations Systems. Mathematics and Computers in Simulation 34 (1992), 1-30.

[9]). G. Schuster, F. Breitenecker,I. Husinsky: The Model Interconnection Concept and ist Effects on Parallelisation of Simulation Models. Proc. ESS 92 - European Simulation Symposium Dresden, SCS Publication San Diego (1992), 93-97.

[10] F. Breitenecker, D. Murray-Smith: Comparison of Parallel Simulation Technique. In (Hrsg. F. Breitenecker, I. Husinsky) EUROSIM Simulation News Europe (SNE) no. 10, Elsevier (1994), 21-22.

[11] F. Breitenecker, I. Husinsky, G. Schuster: Sample Solution: Workstation Cluster under PVM. In (Hrsg. F. Breitenecker, I. Husinsky) EUROSIM Simulation News Europe (SNE) no. 10, Elsevier (1994), 24-25.

[12] I. Husinsky, F. Breitenecker: Comparison of Simulation Software - The EUROSIM Comparisons. In (Eds. F.Maceri, G. Iazeolla) Proc. EUROSIM’92 Simulation Congress, Elsevier (1992), 181 - 186.

[13] W. Husinsky: Comparison 1: Lithium-Cluster Dynamics under Electron Bombardment. In (Hrsg. F. Breitenecker, I. Husinsky) EUROSIM Simulation News Europe (SNE) no. 0 and no.1, ARGE Simulation News TU Vienna (1990/1991), 22 and 25.

[14] F. Breitenecker: Comparison 1: Preliminary Summary of Results. In (Hrsg. F. Breitenecker, I. Husinsky) EUROSIM Simulation News Europe (SNE) no. 6, ARGE Simulation News TU Vienna (1993), 26 - 29.