Informatik-Spektrum (1989) 12: 198–210

Informatik
© Springer-Verlag 1989 Spektrum

Diskrete Simulation – Prinzipien und Probleme der Effizienzsteigerung durch Parallelisierung

F. Mattern und H. Mehl

Universität Kaiserslautern

Zusammenfassung. Simulation ist ein flexibles, aber oft zeitaufwendiges Analysehilfsmittel; eine Effizienzsteigerung ist von großer praktischer Bedeutung. Neben der Verwendung traditioneller Methoden zur Beschleunigung eines Simulationsablaufs denkt man in den letzten Jahren zunehmend an den Einsatz von Parallelrechnern und Multicomputern. Leider stellt sich heraus, daß die meisten Simulationsstrategien kaum oder nur mit Mühe parallelisiert werden können. Im vorliegenden Beitrag werden die verschiedenen traditionellen Simulationsmethoden vorgestellt und auf Parallelisierungsmöglichkeiten hin analysiert. Probleme und Möglichkeiten werden aufgezeigt, mit Literaturreferenzen wird auf weiterführende Untersuchungen verwiesen. In einem späteren Aufsatz sollen darauf aufbauend die neueren Ideen der parallelen diskreten Simulation beschrieben werden, wobei insbesondere auf die verteilte Simulation eingegangen wird.

Schlüsselwörter: diskrete Simulation, ereignisorientierte Simulation, verteilte Simulation, parallele Simulation, Simulationstheorie, Simulationsmethodik

Summary. Simulation is a versatile, but often time-consuming tool; increasing its speed is of great practical importance. In recent years, the use of parallel computer systems and multicomputers is considered in addition to traditional means for speeding up simulations. Unfortunately, most simulation strategies can hardly be parallelized. This article presents the traditional simulation methodologies and analyzes them with respect to potential parallelism. Problems and opportunities are shown, references to more detailed studies are given. In a forthcoming article the theories and methods of parallel and distributed simulation will be described.

Key words: Discrete event simulation, Event-oriented simulation, Distributed simulation, Parallel simulation, Simulation theory, Simulation methodology

Computing Reviews Classification: I.6.1

1. Einleitung

Simulation ist ein wirkungsvolles, in weiten Bereichen der Technik sowie der Natur- und Wirtschaftswissenschaften eingesetztes Hilfsmittel zur Analyse komplexer Systeme. Dabei wird das Verhalten eines realen Systems durch ein *Modell* nachgebildet, das ein vereinfachtes Abbild der komplexen Realität darstellt, sich bezüglich der relevanten Aspekte jedoch weitgehend analog dem realen System verhält. Dieses Modell erlaubt es, Experimente und Messungen durchzuführen, die am realen System zu gefährlich, unmöglich oder zu aufwendig wären. Modelle können materiell existieren, z. B. als maßstabsgerechte Verkleinerung einer Fabrik, oder nur als Programm in einem Rechner vorliegen. Letzteres bietet u. a. den Vorteil, daß relativ einfach verschiedene Modellvarianten untersucht werden können, eventuell bereits vorliegende Daten als Systemparameter in das Simulationsmodell eingebracht werden können und die hohe Geschwindigkeit von Rechnern ausgenutzt werden kann.

Simulation läßt sich als Analysemethode vielfach auch dann noch einsetzen, wenn andere Methoden, etwa eine vollständige mathematische Modellierung komplexer realer Systeme (z. B. der Weltwirtschaft), bereits versagen [50]. Liegt das reale System nicht vor (z. B. weil es erst geplant ist) oder laufen die Vorgänge der Realität zu schnell oder zu langsam ab (wie bei chemischen Reaktionen oder der Entstehung von Galaxien, aber auch bei vielen ökonomischen und sozialwissenschaftlichen Problemen), so stellt die Simulation ebenso wie bei der Analyse von Vorgängen mit irreversiblen Nebeneffekten (z. B. im ökologischen oder militärischen Bereich), „die die Realität nie ertragen könnte" [27], oftmals sogar die einzige Untersuchungsmöglichkeit dar. Typische *Zielrichtungen* der Simulation sind etwa (vgl. auch [54]):

- *Optimierung* des Systemverhaltens (z. B. durch Bestimmung von Engpässen),
- *Entscheidungshilfe* beim Systementwurf und bei der Auswahl unter verschiedenen Entwurfsalternativen,
- *Prognose* des Systemverhaltens (z. B. Wettervorhersage),
- *Überprüfung von Theorien* durch hypothetische Modellbildung eines im Detail unbekannten Systems (z. B. zur Untersuchung kognitiver Prozesse) mit dem Ziel, das System besser zu verstehen,
- *Validierung* eines geplanten Systems (z. B. Simulation eines integrierten Schaltkreises oder eines Mikroprozessors),
- *Modellanimation* zur Veranschaulichung eines komplexen Systemvorgangs.

Viele weitere Einsatzgebiete und Zielrichtungen rechnerbasierter Simulationsmethoden ließen sich nennen. Neben den Vorteilen der Simulation als Analysehilfsmittel sind aber auch *Nachteile* zu nennen [2, 50]:

- Da sich die Realität oftmals nicht exakt im Modell nachbilden läßt, sondern dieses i. a. eine Abstraktion der Realität darstellt, können sich *Ungenauigkeiten* bei den Simulationsergebnissen und ihrer Interpretation ergeben.
- Es ist oft schwierig, den Grad der Übereinstimmung der Simulationsresultate mit der Realität abzuschätzen.
- Simulationen können sehr *rechenzeitaufwendig* sein.

Der zuletzt genannte Punkt ist der Hauptgrund, nach Methoden zu suchen, die es gestatten, eine Simulation verteilt auf mehreren Prozessoren parallel auszuführen. Für den großen Zeitbedarf vieler Simulationen gibt es mehrere Gründe, die unterschiedliche Ansatzpunkte zur Parallelisierung bieten (vgl. auch Abschnitt 3):

- Bei vielen Simulationen handelt es sich um *stochastische* Simulationsexperimente, bei denen einige Parameter Zufallsvariablen einer bestimmten Verteilung darstellen. Um gesicherte und statistisch relevante Ergebnisse zu bekommen, muß – abhängig von der Varianz und der geforderten Genauigkeit – oft eine große Zahl von Einzelexperimenten durchgeführt werden.
- In vielen Fällen, insbesondere bei Optimierungsproblemen und der Bewertung von Entwurfsalternativen, muß eine sehr große *Serie* von Experimenten für unterschiedliche Parameterwerte durchgeführt werden.
- Der Rechenaufwand vieler Modelle wächst nicht proportional, sondern oft quadratisch, kubisch oder sogar exponentiell mit der Modellgröße. Eine Verfeinerung des Modells mit dem Ziel, die Genauigkeit der Ergebnisse geringfügig zu verbessern, führt dann zu einem erheblich größeren Zeitaufwand. In vielen Fällen ist die *nicht-lineare Komplexität* leider probleminhärent.
- Einige Modelle bestehen aus vielen sich gegenseitig beeinflussenden Objekten und sind so *komplex*, daß eine detaillierte Simulation des Verhaltens notgedrungen viel Rechenzeit erfordert.

Als rechenzeitaufwendige Beispiele aus dem Bereich der Informatik seien etwa die Simulation von Kommunikationshardware genannt (mehrere hundert Stunden Rechenzeit für einen Simulationslauf nach [58]) oder die Simulation von VLSI-Chips auf dem Gatterniveau mit bis zu mehreren Monaten Rechenzeit [66]. Da sich die Zahl der Gatter bei VLSI-Chips gegenwärtig etwa alle zwei bis drei Jahre verdoppelt, der empirisch ermittelte Zeitbedarf einer solchen Simulation jedoch eine quadratische bis kubische Größenordnung annimmt [20, 26], kann die Entwicklung der Rechengeschwindigkeit hier nicht Schritt halten. Man muß in diesem Fall von der vollständigen detailgetreuen Simulation auf Gatterebene Abstand nehmen und analog dem VLSI-Entwurf auf hierarchische und modulare Simulationstechniken ausweichen.

Der allgemeine Bedarf an einer Beschleunigung der Simulation ist sehr groß; tatsächlich werden viele Superrechner fast ausschließlich für Simulationsaufgaben im weitesten Sinne verwendet [17]. In einigen Bereichen würde auch erst eine wesentliche Geschwindigkeitssteigerung eine interaktive Realzeitsimulation ermöglichen und so neue interessante Anwendungsmöglichkeiten eröffnen. Da jedoch bei vielen wichtigen Problemen bereits schnellste Rechner und kaum mehr optimierbare Simulationsmodelle und -programme verwendet werden, müssen andere Möglichkeiten zur *Beschleunigung* einer Simulation gesucht werden, zum Beispiel:

- Entwicklung spezieller Hardware für Simulatoren – dies führt jedoch i. a. zu aufwendigen und spezifischen Lösungen, was nur für einige Anwendungsklassen (etwa Logiksimulatoren oder Simulatoren mit Realzeitanforderungen) gerechtfertigt scheint;
- Parallelisierung von Simulationsabläufen.

Die Hoffnungen konzentrieren sich vor allem auf den letzten Punkt, wofür zwei Gründe maßgeblich sind:

1) Komplexe Systeme bestehen aus vielen *autonomen Komponenten*, die miteinander in Wechselwirkung stehen. Anstatt diese Komponenten durch Sequentialisierung in einem einzigen Prozessor zu simulieren, scheint eine Vorgehensweise attraktiver zu sein, bei der jede Komponente in einem eigenen Prozessor simuliert wird. Dieser Hoffnung liegt also die Beobachtung zugrunde, daß komplexe reale Systeme i. a. „inhärent parallel" sind und sich diese natürliche Parallelität einfach ausnutzen lassen sollte.
2) Seit einigen Jahren stehen *kommerzielle Mehrrechnersysteme* zur Verfügung; hierzu gehören Parallelrechner wie das Sequent-Balance-System oder die Butterfly-Maschine, Hypercube-Systeme wie die Intel iPSC-Rechner, Mehrrechnersysteme auf Transputerbasis oder auch nur als „Prozessorfarm" genutzte lokal vernetzte Workstations. Die Connection-Maschine und damit verbundene Ansätze des „massiven Parallelismus" eröffnen weitere Perspektiven.

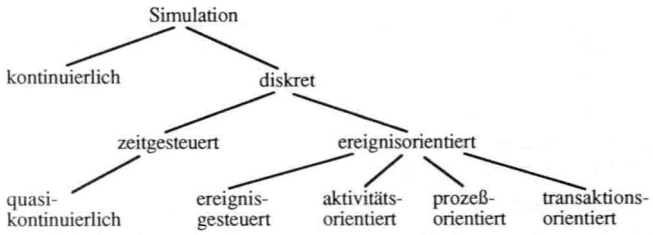

Abb. 1. Klassifikation von Simulationsmethoden

Die große Bedeutung, die einer effizienten Simulationsmethodik in der Praxis zukommt, hat zusammen mit der Verfügbarkeit von Mehrrechnersystemen in den letzten Jahren vor allem an amerikanischen Universitäten eine Reihe von praktisch orientierten Forschungsaktivitäten auf dem Gebiet der parallelen und verteilten Simulation ausgelöst [21, 22, 32, 33, 44, 45, 58, 59, 69]. Erste theoretische Arbeiten gehen allerdings bereits auf die Ansätze von Bryant [8], Chandy und Misra [10, 12–14] sowie Peacock et al. [55, 56] gegen Ende der 70er Jahre zurück. Erst Mitte der 80er Jahre entstand das unkonventionelle, aber interessante Time-Warp-Konzept [31, 34]. Einen guten Literaturüberblick zur verteilten Simulation (wenn auch auf die älteren, bis 1985 erschienenen Veröffentlichungen beschränkt) gibt Kaudel [38]. Eine ausführliche Erläuterung der Problematik verteilter und paralleler Simulation sowie ein Überblick über neue Literatur auf diesem Gebiet soll in einem späteren Beitrag folgen (vgl. dazu auch [47, 59]).

Eingehende Überlegungen und praktische Experimente der jüngsten Zeit zeigen jedoch, daß die vielfach geäußerte Hoffnung, die natürliche Parallelität realer Systeme ließe sich in Simulationssystemen unmittelbar mit Mehrrechnersystemen ausnutzen, einen Trugschluß darstellt. Wie weiter unten gezeigt wird, sind die Prinzipien der *Kausalität* und der *Zeitkonsistenz* verantwortlich dafür, daß die wichtige Klasse der ereignisorientierten Simulationen als *inhärent sequentiell* angesehen werden muß und es zur Parallelisierung besonderer Maßnahmen bedarf. Um dies zu begründen, werden im folgenden zunächst die Prinzipien klassischer Simulationsverfahren unter besonderer Berücksichtigung des ereignisorientierten Paradigmas vorgestellt, bevor in Abschnitt 3 auf die Parallelisierungsproblematik eingegangen wird.

2. Simulationsmethoden

In der Praxis werden für die unterschiedlichsten Aufgaben verschiedene Simulationsmethoden und -sprachen verwendet. Da meist die Entwicklung eines dynamischen Systems in der *Zeit* untersucht wird, spielt die geeignete Modellierung der zeitlichen Abläufe von Systemkomponenten und der Zeit selbst eine wichtige Rolle. Hierfür gibt es eine Reihe von Ansätzen, die sich in verschiedenen *Paradigmen* widerspiegeln. Diese Simulationsparadigmen umfassen Methoden, Strategien, Modellierungsstile und Sichtweisen, wobei zu jedem Paradigma typische Simulationssprachen, prototyphafte Anwendungen und eigene Benutzerklassen gehören (vgl. auch [42, 48, 65]). Abbildung 1 zeigt eine Klassifikation der im folgenden vorgestellten klassischen Simulationsparadigmen.

Zweck einer Simulation ist es, Experimente und Messungen an einem dem realen System in wesentlichen Eigenschaften entsprechenden *Modell* durchzuführen. Die Erstellung des Simulationsmodells ist i.a. nicht trivial und stellt oft mehr eine Kunst als eine Wissenschaft dar [50]. Oft ist der Entscheidungsspielraum bei der Modellierung groß und läßt abhängig von den durchzuführenden Experimenten und den erwarteten Resultaten recht unterschiedliche Ausprägungen zu. Allerdings ist der Modellierer durch die Möglichkeiten der Simulationssprache eingeschränkt, in der das Modell spezifiziert wird. Simulationssprachen prägen bereits durch ihre Konzepte und die ihnen unterlegte *Weltsicht* den prinzipiellen Aufbau des Simulationsmodells.

So wie das reale System i.a. aus Komponenten besteht, die miteinander in Wechselwirkung stehen, setzt sich das Modell aus statischen oder dynamischen *Objekten* zusammen, die sich gegenseitig beeinflussen. Diese Objekte repräsentieren einzelne Systemkomponenten oder auch ganze Klassen von Systemkomponenten. Bei der Simulation wird das Modell ausgeführt, wobei sich der *Zustand* des Modells, der sich aus den Zuständen der einzelnen Objekte zusammensetzt, mit dem Fortschreiten der Simulationszeit verändert. Die *Simulationszeit* entspricht dabei der im realen - also zu simulierenden - System voranschreitenden Zeit, steht jedoch i.a. in keinem Bezug zur *Rechenzeit*, die zur Ausführung des Simulationsmodells benötigt wird. Das Voranschalten der Simulationszeit und die Durchführung der damit verbundenen Zustandsänderungen des Modells können auf unterschiedliche Art erfolgen und charakterisieren wesentlich die verschiedenen in den folgenden Abschnitten vorgestellten Simulationsmethoden.

2.1. Kontinuierliche Simulation

Bei der *kontinuierlichen* Simulation geht man davon aus, daß sich der Zustand des Modells stetig mit der Zeit verändert; man abstrahiert also von den möglichen Unstetigkeitsstellen im Ablauf des realen Systems. Das Systemverhalten wird durch eine Menge verkoppelter Gleichungen (meist Differentialgleichungen) charakterisiert, deren freie Variable die Zeit ist. Ausgehend von einem Anfangszustand läßt sich so der Zustand zu einem späteren Zeitpunkt berechnen. Die Simulation entspricht damit dem Lösen von Differentialgleichungen, wobei meist ein numerischer Ansatz mittels Diskretisierung (Übergang zu einem System von Differenzengleichungen) gewählt wird. Hierfür wurden einige parallele numerische Integrationsalgorithmen entwikkelt [39].

Populär ist die kontinuierliche Simulation vor allem im physikalisch-technischen Bereich, ein typisches Bei-

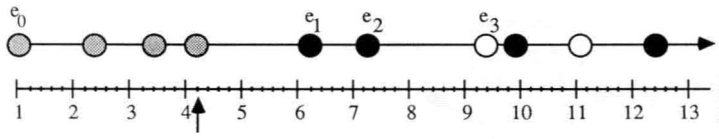

Abb. 2. Ein abstrakter ereignisorientierter Simulationsablauf

spiel stellt die Wettervorhersage dar. Bei derartigen Anwendungen erfolgt i.a. eine homogene Orts- und Zeitdiskretisierung des gesamten Systems. Die effiziente numerische Lösung der dadurch entstehenden großen algebraischen Gleichungssysteme durch Vektor- oder Parallelrechner (z.B. mittels Mehrgittermethoden und hierfür geeigneten Rechnern [64, 67, 68]) ist ein aktuelles Forschungsgebiet der angewandten Mathematik. Im folgenden wird auf die kontinuierliche Simulation nicht weiter eingegangen.

2.2. Diskrete Simulation

Im Unterschied zur kontinuierlichen Simulation erfolgen bei der *diskreten* Simulation die Zustandsänderungen „plötzlich" und sprunghaft zu diskreten Zeitpunkten. Eine solche Zustandsänderung wird durch das Eintreten eines atomaren, d.h. keine Simulationszeit verbrauchenden *Ereignisses* verursacht. Neben dem zentralen Ereignisbegriff sind die Begriffe „Prozeß" und „Aktivität" für eine Charakterisierung der zeitdiskreten Modellierungsvarianten und Simulationsstrategien von besonderer Bedeutung. Ein *Prozeß* stellt eine zeitlich geordnete und inhaltlich zusammengehörige, meist einem bestimmten Simulationsobjekt zugeordnete Folge von Ereignissen dar. Eine *Aktivität* bezeichnet eine zeitlich ausgedehnte, durch ein Anfangs- und Endereignis charakterisierte Operation, die den Zustand eines einzigen Objekts transformiert.

Die diskrete Simulation ist vor allem in der Operations Research und Informatik populär. Typische Beispiele für Ereignisse sind das Einreihen eines Kunden in eine Warteschlange oder bei der Simulation einer Produktionsstraße die Ankunft eines Werkstücks an einer Bearbeitungsstation. Man unterscheidet zeitgesteuerte, ereignisgesteuerte, aktivitätsorientierte, prozeßorientierte und transaktionsorientierte Simulationsstrategien (vgl. Abb. 1), wobei jedoch Kombinationen und Überschneidungen der Ansätze möglich sind. (Der häufig verwendete Begriff *ereignisorientiert* ist etwas diffus; er besagt, daß im Unterschied zur kontinuierlichen Simulation einzelnen Ereignissen eine wesentliche Bedeutung im Simulationsmodell zukommt.) Im folgenden wird nur insoweit auf diese Ansätze eingegangen, als es zum Verständnis der Effizienz- und Parallelisierungsproblematik notwendig ist. Zur genaueren Diskussion weiterer Aspekte sei auf [19, 29, 30, 42, 43] verwiesen.

2.2.1. Ereignisgesteuerte Simulation. Bei der diskreten Simulation ändert sich der Zustand zwischen zwei aufeinanderfolgenden Ereignissen nicht. Daher muß unmittelbar nach dem Stattfinden eines Ereignisses bereits feststehen, welches Ereignis als nächstes eintreten soll. Im allgemeinen stehen jedoch zu einem Zeitpunkt bereits mehr als nur ein einziges zukünftiges Ereignis mit ihren jeweiligen Eintrittszeitpunkten fest - die Ereignisse sind bereits *eingeplant*.

Abbildung 2 skizziert einen abstrakten ereignisorientierten Simulationsablauf. Ereignisse, die bereits stattgefunden haben (z.B. das initiale Urereignis e_0), sind als graue Punkte, die zum momentanen Zeitpunkt 4.2 eingeplanten Ereignisse als schwarze Punkte dargestellt. Später werden weitere, durch weiße Punkte symbolisierte Ereignisse hinzukommen, was aber zum Zeitpunkt 4.2 noch unbekannt ist. So könnte etwa Ereignis e_3 durch e_1 oder e_2 verursacht werden.

Bei der ereignisgesteuerten Simulation hat jedes eingeplante Ereignis einen *Eintrittszeitpunkt* und bewirkt bei seinem Eintritt einen bestimmten Zustandsübergang. Üblicherweise lassen sich Ereignisse zu *Ereignistypen* zusammenfassen, so daß im Simulator für jeden Ereignistyp eine evtl. parametrisierbare Routine bereitgestellt werden kann, die bei Eintreten eines entsprechenden Ereignisses vom Simulator ausgeführt wird. Die eingeplanten Ereignisse werden vom Simulator in einer (üblicherweise nach der Eintrittszeit sortierten) *Ereignisliste* aus sogenannten *Ereignisnotizen* gehalten (vgl. Abb. 3). Da zwischen den Ereignissen nichts „von Interesse" geschieht, kann jeweils die Simulationszeit auf die Eintrittszeit des global nächsten Ereignisses erhöht und das entsprechende Ereignis simuliert werden. Dies geschieht, indem der vorderste Eintrag (d.h. die Ereignisnotiz mit der kleinsten Eintrittszeit) entfernt, die Simulationszeit auf die dort angegebene Eintrittszeit erhöht und die zugehörige Routine ausgeführt wird. Als Folge davon wird der Zustand des Modells verändert und werden evtl. weitere Ereignisse eingeplant.

Durch das Fortschalten der Simulationszeit auf die Eintrittszeit des jeweils nächsten Ereignisses werden ereignislose, evtl. länger dauernde „Totzeiten" übersprungen, „whose passage in the real world we are forced to endure" [19]. Die Zeitfortschaltung geschieht also *ereignisgesteuert*. Von besonderer Bedeutung für Fragen der Beschleunigung und Parallelisierung ist in diesem Zusammenhang die *Dualitäts-Eigenschaft:*

Die Ausführung von Ereignisaktionen kostet Rechenzeit, jedoch keine Simulationszeit. Dagegen benötigt eine durch ein Anfangs- und Endereignis modellierte Aktivität Simulationszeit, jedoch keine Rechenzeit.

Etwas unschärfer, aber provokativer formuliert: Was in der Realität Zeit kostet, kostet bei der Simulation keine Zeit - und umgekehrt. Ein Beispiel mag die Be-

deutung dieses Prinzips unterstreichen: Die Modellierung des Straßenverkehrs einer Stadt erfolge so, daß das Verlassen des Kreuzungsbereichs und die Ankunft eines Autos bei einer Kreuzung Ereignisse darstellen. Die Aktivität eines Autos, d.h. die Fahrt zwischen zwei Kreuzungen, wird also durch das Ankunftsereignis modelliert, welches zur Zeit T für einen Zeitpunkt $T+\Delta t$ eingeplant wird, wobei Δt die Fahrzeit der Strecke für dieses Auto darstellt. In jedem Augenblick sind in der Realität i.a. sehr viele Autos gleichzeitig unterwegs, so daß der Gedanke naheliegt, bei einem Multiprozessor diese „natürliche Parallelität" in der Weise zur Beschleunigung der Simulation auszunutzen, daß (idealerweise) jedes Auto durch einen eigenen Prozessor unterstützt wird. Da die Aktivitäten der Autos im Modell jedoch keine Rechenzeit benötigen, wären die Prozessoren ständig - bis zum Ende der Aktivität - passiv!

Die Nichtberücksichtigung des Dualitätsprinzips ist ein häufiger Irrtum[1] - es ist verführerisch, die Parallelität des realen Systems direkt ausnutzen zu wollen. Das Beispiel zeigt, daß aufgrund der Dualität *Ereignisse* - und nicht Aktivitäten - gleichzeitig ausgeführt werden müssen, wenn bei der ereignisgesteuerten Simulation eine Leistungssteigerung durch Parallelität angestrebt wird. Dies führt jedoch zu weiteren Problemen. So können etwa die Ereignisse e_1 und e_2 aus Abb. 2 nur dann gleichzeitig ausgeführt werden, wenn sie *unabhängig* voneinander sind. Denn ändert e_1 den Zustand in einer Weise, daß davon e_2 beeinflußt wird, so würde eine gleichzeitige Ausführung von e_1 und e_2 i.a. ein anderes (und damit falsches) Resultat liefern als die sequentielle Ausführung von e_1 und e_2. Entsprechendes gilt, wenn e_1 Ereignisse zwischen e_1 und e_2 einplant.

Ohne weitere (semantische) Information können also keine Ereignisse gleichzeitig ausgeführt werden. Deshalb muß die ereignisgesteuerte Simulation, und darauf aufbauende Simulationsstrategien wie die prozeßorientierte Simulation, prinzipiell als *inhärent sequentiell* angesehen werden.

Vielfach wird noch immer behauptet, daß bei vielen Ereignissen die Verwaltung der Ereignisliste sehr aufwendig ist und die ereignisgesteuerte Simulation daher ineffizient (vgl. etwa [51] oder [70])[2]. Tatsächlich würde das Einsortieren einer Ereignisnotiz in eine verkettete Liste entsprechend Abb. 3 $O(n)$ Schritte benötigen, wenn n die (typische) Länge der Ereignisliste bezeichnet. Da im wesentlichen jedoch nur die beiden Operationen „insert" (Einplanen eines Ereignisses) und

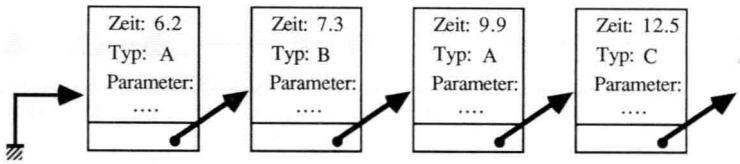

Abb. 3. Beispiel einer Ereignisliste

„delete-min" (Entfernen der „vordersten" Ereignisnotiz) unterstützt werden müssen (eine allgemeine „delete"-Operation ist nur für den meist seltenen Fall erforderlich, wo bereits fest eingeplante Ereignisse wieder gestrichen werden), eignen sich zur Realisierung dieser abstrakten Datenstruktur *priority-queues* [1, 41] mit geringerer Zeitkomplexität wesentlich besser. Nach Jones [37] konnte so bei großen Systemen oft eine Verringerung der Gesamtlaufzeit der Simulation um einen Faktor von bis zu 100 erreicht werden. In den letzten Jahren wurden zahlreiche Untersuchungen zu effizienten Implementierungen von Ereignislisten mittels priority-queues veröffentlicht (vgl. [35] mit weiteren Literaturhinweisen); von den $O(\log n)$-Verfahren scheinen sich am besten ternäre heaps [60] und splay-trees [35, 63] zu eignen. Vor kurzem wurden sogar einfache problembezogene adaptive Hash-Verfahren mit $O(1)$-Zeitkomplexität und sehr niedrigem Proportionalitätsfaktor gefunden [7]. Ein Verständnis geeigneter Implementierungstechniken von Ereignislisten ist bei Untersuchungen zur Effizienzsteigerung von Simulatoren wesentlich, damit die Parallelität nicht an der falschen Stelle angesetzt wird (vgl. Abschnitt 3.3).

Simulationssprachen, die ausschließlich auf der reinen ereignisgesteuerten Simulationsstrategie beruhen, wie frühere Versionen von GASP oder Simscript, werden heute nicht mehr verwendet. Typische moderne Simulationssprachen bieten Strukturierungsmöglichkeiten an, die über die elementare Ereigniseinplanung hinausgehen, und beruhen meist auf den darauf aufbauenden Prozeß- oder Transaktionskonzepten (vgl. 2.2.4 und 2.2.5).

2.2.2. Zeitgesteuerte Simulation. Die zeitgesteuerte Simulation beruht darauf, daß in jedem Simulationsschritt die Simulationszeit um ein vorher festgelegtes *konstantes Zeitinkrement* Δt erhöht wird. Nach jeder Erhöhung der Zeit werden die innerhalb der letzten *Epoche* Δt aufgetretenen Zustandsveränderungen durchgeführt. Ein einfaches Beispiel für dieses Prinzip stellt das bekannte *Game of Life* [23] dar; ernstere, prinzipiell aber auf der gleichen Methode beruhende Anwendungsklassen sind z.B. Gefechtsfeldsimulationen [24, 52] oder volkswirtschaftliche Prognosen und Planspiele. Die zeitgesteuerte Simulation wird oft dann angewendet, wenn die in einer Epoche erfolgten Zustandsveränderungen sich ohne unnötige Detaillierung kaum einzelnen Ereignissen zuordnen lassen, wie etwa bei meteorologischen Modellrechnungen. Nachteilig gegenüber der ereignisgesteuerten Simulation ist vor

[1] Vgl. etwa [71]: „Those processes which occur in parallel in the real system could be simulated in parallel in the distributed simulation system. This approach is esthetically pleasing because it allows the physical components of the world which occur in parallel to be truly executed as such in the simulation."

[2] Gosh [25] schreibt sogar: „Manipulations of this list can often be very expensive and can thus detract from the attractiveness of simulation as a problem-solving tool."

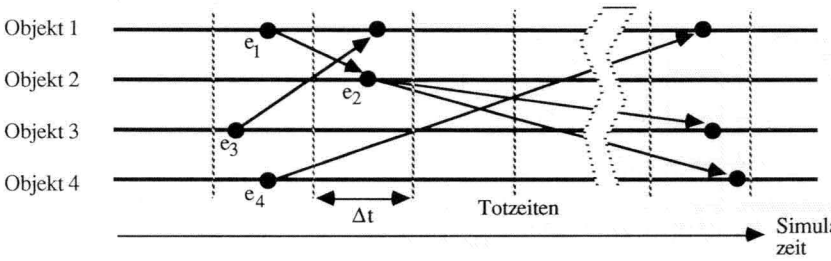

Abb. 4. Zeitgesteuerte Simulation mit ereignisorientiertem Charakter

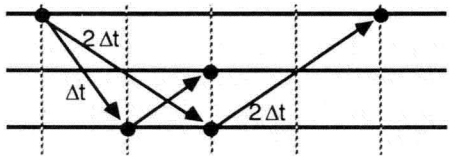

Abb. 5. Zeitliche Mindestgarantien bei der zeitgesteuerten Simulation

allem, daß auch über längere Totzeiten hinweg die Simulationsuhr in einzelnen Δt-Schritten erhöht werden muß.

Die Größe der Zeitepoche Δt ist von entscheidender Bedeutung für Korrektheit, Genauigkeit und Effizienz der Simulation: Sie sollte einerseits klein genug sein, damit Zustandsänderungen eines Objekts sich auf andere Objekte erst in einer der nächsten Zeitepochen auswirken, da sonst die Reihenfolge, in der die Zustandsänderungen der Objekte durchgeführt werden, Einfluß auf die Simulation haben kann. Andererseits sollte sie möglichst groß sein, damit wenige Zustandstransformationen durchgeführt werden müssen und die Simulation durch ein schnelles Voranschalten der Simulationsuhr effizient abläuft. Sinnvolle Größen des Zeitrasters sind stark anwendungsabhängig. So können diese bei der Simulation von VLSI-Chips im Nano- oder Mikrosekundenbereich liegen, für typische Straßenverkehrssimulationen im Sekundenbereich [18], während für die in [24] beschriebene Gefechtsfeldsimulation zehn Minuten angegeben werden.

Ist Δt klein genug und kann der kontinuierliche Verlauf von Modellgrößen damit hinreichend genau approximiert werden, so nimmt die zeitgesteuerte Simulation einen *quasi-kontinuierlichen* Charakter an - einzelnen Ereignissen kommt dann keine wesentliche Bedeutung mehr zu. Dagegen behalten bei der zeitgesteuerten Simulation mit *ereignisorientiertem* Charakter die Ereignisse ihre eigenständige Rolle; hierbei werden alle in einer Epoche liegenden Ereignisse als am Ende der Epoche zusammenfallend betrachtet. Abbildung 4 zeigt anhand eines Zeitdiagramms den möglichen Ablauf einer derartigen Simulation von vier Objekten, wobei die Einplanungs- bzw. Verursachungsrelation durch Pfeile symbolisiert wird. Auch hier gilt, daß z. B. e_3 den (globalen) Zustand nicht so beeinflussen darf, daß ein anderes Ereignis (e_1, e_4) derselben Epoche davon betroffen sein kann. Dies ist garantiert, wenn die Ereignisse nur auf disjunkten lokalen Objektzuständen operieren und Ereignisse nur in späteren Epochen eingeplant werden. In diesem Fall sind die bei verschiedenen Objekten stattfindenden Ereignisse einer Epoche (bzw. die durch sie vermittelten Aktivitäten) unabhängig voneinander - sie können *parallel* ausgeführt werden.

Die Ausnutzung der Parallelität in diesem Sinne erfordert also die Aufteilung des globalen Zustands in disjunkte, lokale Bereiche und die Garantie, daß die Einplanung von Ereignissen frühestens für die nächste Epoche stattfindet. Da Ereignisse immer als am Ende einer Epoche eintretend angesehen werden können, bedeutet dies anschaulich am Zeitdiagramm, daß die zwischen zwei Ereignissen e, e' verlaufenden „Kausalitätspfeile" eine zeitliche Mindestlänge von Δt besitzen (vgl. Abb. 5). Derartige zeitliche Mindestgarantien spielen allgemein bei der Parallelisierung ereignisorientierter Simulationen sowie bei verteilten und parallelen Simulationsverfahren eine zentrale Rolle.

Zur Parallelisierung zeitdiskreter Simulationen fanden einige praktische Untersuchungen statt. So beschreibt Ludwig [46] ein zeitgesteuertes, quasi-kontinuierliches Simulationsexperiment auf dem Erlanger 26-Prozessor-DIRMU-Experimentalsystem zur Lösung der Boltzmann-Gleichung aus dem Bereich der kinetischen Wärmetheorie. Der beste gemessene *speedup* betrug immerhin ca. 16, d. h. mit 26 Prozessoren wurde ein um etwa den Faktor 16 beschleunigter Ablauf gegenüber der Monoprozessorversion erreicht. Bei der in [24] und [52] beschriebenen Gefechtsfeldsimulation wurde bei 8 Prozessoren ein speedup von 5 erreicht.

Zur Leistungsmaximierung sollte idealerweise nicht nur für jedes Objekt ein eigener Prozessor zur Verfügung stehen, der die jeweilige Zustandstransformation für eine Zeitperiode berechnet, sondern es müssen problemspezifische Maßnahmen zur Minimierung des Synchronisationsaufwands und zur Lastverteilung getroffen werden. Abbildung 6 illustriert, daß die Ausführungszeit einer Rechenphase, die einer Zeitepoche entspricht, durch den jeweils am stärksten belasteten Prozessor bestimmt ist. An die Rechenphase schließt sich typischerweise eine Synchronisationsphase an, die dem Datenaustausch und der Erhöhung der Simulationszeit dient. Dies geschieht bei Realisierungen auf Multicomputern oder anderen Systemen mit verteiltem Speicher in Form von Nachrichten (vgl. dazu auch [55]), wodurch ein gewisser Zusatzaufwand verursacht wird.

Im allgemeinen ist eine Beschleunigung der zeitgesteuerten Simulation durch Parallelität nur für be-

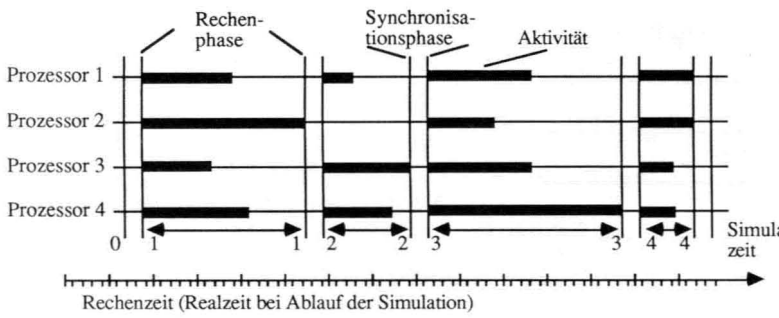

Abb. 6. Phasendauer bei der zeitgesteuerten Simulation

stimmte Anwendungen möglich: Der Zustand sollte sich so auf eine genügend große Zahl unabhängiger Prozesse aufteilen lassen, daß die Rechenlast einigermaßen gleich verteilt ist; gegebenenfalls müssen dazu Teile des Zustands repliziert werden, was wiederum den Synchronisationsaufwand erhöhen kann. Zur Minimierung des Verwaltungsaufwands sollten zudem die Rechenphasen im Vergleich zu den Synchronisationsphasen lang sein. Da typischerweise die Zahl der Prozesse größer als die Zahl der verfügbaren Prozessoren ist, muß eine geeignete Zuordnung („mapping") gefunden werden. Ändert sich das Lastaufkommen dynamisch, so scheint sogar eine dynamische Neuverplanung der Aufgaben („load balancing"), u. U. durch anwendungsbezogene Steuerungsmaßnahmen, notwendig zu sein.

2.2.3. Aktivitätsorientierte Simulation. Als *Aktivität* wird eine Zustandstransformation eines Objekts bezeichnet, die eine gewisse (simulierte) Zeit in Anspruch nimmt; sie ist durch ein Anfangs- und ein Endereignis charakterisiert. Nur diese beiden Ereignisse werden tatsächlich ausgeführt und lösen Aktionen aus, die Zeit dazwischen wird entsprechend der ereignisorientierten Simulationsstrategie übersprungen. Aktivitäten sind also lediglich konzeptuelle Modellierungshilfsmittel. Bei dem oben skizzierten Beispiel einer Verkehrssimulation ist z. B. die Fahrt eines Autos zwischen zwei Kreuzungen oder gegebenenfalls auch das Warten vor einer Ampel eine Aktivität.

Eine Aktivität hat eine *Eintrittsbedingung*, die eine beliebige Boole'sche Bedingung über dem Modellzustand darstellt und mit einer Mindesteintrittszeit verknüpft sein kann, sowie eine (simulierte) *Dauer*, die die Aktivität zeitlich begrenzt. Mit dem Beginn und dem Ende einer Aktivität sind jeweils Aktionen in Form von Ereignisroutinen verbunden, die den Modellzustand verändern und dadurch implizit andere Aktivitäten auslösen können. In Analogie zur ereignisgesteuerten und zeitgesteuerten Simulationsstrategie ließe sich die aktivitätsorientierte Simulationsstrategie daher auch als *zustandsgesteuert* charakterisieren. Die Attraktivität des Ansatzes liegt vor allem darin, daß nicht explizit angegeben werden muß, welches Ereignis welches andere auslöst, und daß im Programmtext Auslösebedingung, Dauer und Aktionen einer Aktivität lokal beeinander stehen.

Bei der aktivitätsorientierten Simulation wird ein Modell durch eine nicht weiter strukturierte Menge von Aktivitäten spezifiziert. Der englische Begriff *activity scanning* bezeichnet treffend die Aufgabe des Simulators: Zyklisch werden die Aktivitäten daraufhin untersucht, welche von ihnen ausgelöst werden können. Da die Zustandsänderung durch das Auslösen einer Aktivität andere, bisher nicht ausführbare Aktivitäten ausführbar machen kann, kann erst nach einem vollständigen Durchgang ohne Zustandsänderung garantiert werden, daß alle möglichen Startereignisse zum aktuellen simulierten Zeitpunkt ausgeführt wurden. Dann erst kann die Simulationszeit auf die Zeit des kleinsten eingeplanten Endereignisses (sofern vorhanden) gesetzt werden und können alle zu diesem Zeitpunkt eingeplanten Endereignisse ausgeführt werden, bevor sich der Zyklus wiederholt.

Die aktivitätsorientierte Simulation läßt sich ohne weiteres mit dem reinen ereignisgesteuerten Simulationsansatz kombinieren [28, 53]. Da das zyklische Durchsuchen und Abtesten der Eintrittsbedingungen zeitaufwendig ist, war die aktivitätsorientierte Simulationsstrategie jedoch nie besonders populär und darauf aufbauende Sprachen wie CSL oder SIMON nicht sehr verbreitet. Allerdings könnten einige Techniken, die bei der entsprechenden Problematik regelbasierter Sprachen und Produktionensysteme im Rahmen der KI-Forschung entdeckt wurden, auch hier Anwendung finden und eine wesentliche Beschleunigung des Simulators bewirken - handelt es sich doch bei der aktivitätsorientierten Simulation um eine sehr frühe (Ende der 50er Jahre entwickelte) Ausprägung der Programmierung mit Kontrollabstraktionen!

Bezüglich einer möglichen *Parallelisierung* gilt grundsätzlich das gleiche wie für die ereignisgesteuerte Simulationsstrategie - die gleichzeitig ausgeführten Ereignisse sollten unabhängig voneinander sein. Da sich jedoch der Anwender wegen der Kontrollabstraktion nicht auf eine bestimmte Ausführungsreihenfolge der ausführbaren Aktivitäten bzw. Ereignisse verlassen kann, scheint ein größeres Potential an Parallelität zu existieren, sofern nach einer Zustandsänderung i.a. mehr als nur eine Aktivität ausführbar ist. Allerdings bedeutet die Tatsache, daß die Eintrittsbedingung für zwei Aktivitäten zu einem Zeitpunkt erfüllt ist, nicht notwendigerweise, daß auch tatsächlich beide Aktivitäten ausgeführt werden: Das jeweilige Anfangsereignis kann den Zustand so ändern, daß die Eintrittsbedingung der anderen Aktivität wieder ungültig wird, sich

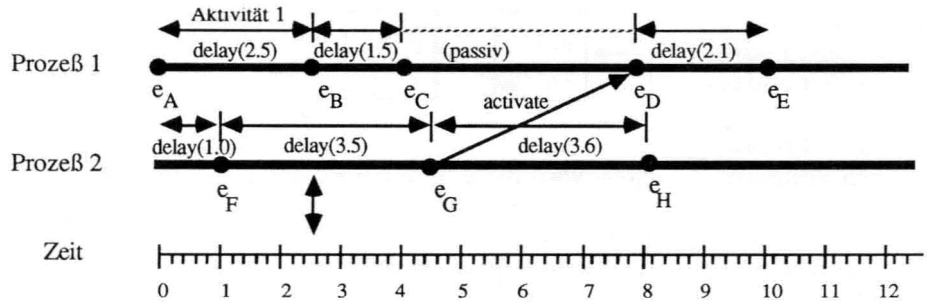

Abb. 7. Zeitlicher Ablauf einer prozeßorientierten Simulation

```
Prozeß_1:        Prozeß_2:
    A;               delay (1.0);
    delay (2.5);     F;
    B;               delay (3.5);
    delay (1.5);     G: {if V=0 then activate Prozeß_1 at 7.9};
    C: {V:=0};       delay (3.6);
    passivate;       H;
    D;
    delay (2.1);
    E;
```

Abb. 8. Prozeßstruktur zu Abb. 7

also die beiden Aktivitäten wechselseitig ausschließen. Bei einem möglichen parallelen Abtesten der Eintrittsbedingung könnten dagegen beide Aktivitäten gleichzeitig ausgeführt werden. Da der Effekt jedoch darauf beruht, daß zu einem virtuellen Zeitpunkt das Prädikat der Eintrittsbedingung nicht wohldefiniert ist (es kann je nach Ausführungsreihenfolge entweder wahr oder falsch sein), kann er daher wohl genauso wie der generelle Nichtdeterminismus der Simulationsstrategie in Kauf genommen werden.

Operieren die parallel ausgeführten Ereignisse auf einem gemeinsamen, globalen Speicher, so kann es auch hierbei zu Konflikten kommen. Bei einem disjunkten, verteilten Speicher entsteht jedoch ein neues Problem - die Eintrittsbedingung einer Aktivität ist dann möglicherweise nicht lokal überprüfbar, da sie vom Zustand anderer, nicht lokaler Objekte abhängt. Die Berechnung eines globalen Prädikats in verteilten Systemen ist jedoch mit Aufwand verbunden und sinnvoll auch nur für monotone Prädikate möglich (also für Prädikate, die nicht mehr falsch werden, nachdem sie einmal wahr wurden) [11].

Obwohl vom Ansatz her in der aktivitätsorientierten Simulationsstrategie ein gewisses Potential an „natürlicher Parallelität" steckt, sind keine speziellen Untersuchungen dazu bekannt.

2.2.4. Prozeßorientierte Simulation. Bei der prozeßorientierten Simulation wird ein Modell aus einer Menge *interagierender Objekte* gebildet. Die Simulationsmethode beruht auf der ereignisgesteuerten oder auch aktivitätsorientierten Simulationsstrategie; sie ist aber durch die zusätzlichen Strukturierungsmöglichkeiten einfacher zu verwenden, da das reale System unmittelbarer nachgebildet werden kann. Ein *Prozeß* stellt eine zusammengehörige, auf ein bestimmtes Objekt bezogene Folge von Ereignissen mit der dadurch vermittelten Sequenz von Aktivitäten dar (vgl. Abb. 7). Da ein Prozeß zusätzlich lokale Daten besitzt, verkörpert er auf diese Weise direkt ein „aktives" Systemobjekt. Ein Prozeß ist z. B. bei einer Verkehrssimulation ein Auto, das nacheinander verschiedene Straßenstücke und Kreuzungen durchfährt. Ankunft und Abfahrt bei einer Kreuzung werden durch Ereignisse repräsentiert (wobei der Prozeß auf die durch passive Datenstrukturen und andere Prozesse realisierte „Umwelt" einwirken kann bzw. durch die Umwelt beeinflußt wird), während die Fahrten zwischen den Kreuzungen lediglich Simulationszeit verbrauchende Aktivitäten darstellen, bei denen der Prozeß nur mit sich selbst beschäftigt ist.

Der Vorteil dieser Methode liegt darin, daß keine isolierten Ereignisroutinen erstellt werden müssen, sondern die Aktionen eines Objekts im Programm textuell zusammen stehen. Außer aus den durch Prozesse modellierten *aktiven* Objekten besteht ein prozeßorientiertes Simulationsmodell i. a. noch aus *passiven* Objekten, die durch globale Daten oder durch geeignet modularisierte Daten (records, Klassen oder in der speziellen Simulationssprache vorgesehene Datenstrukturen) repräsentiert werden. Beim Beispiel der Verkehrssimulation könnten etwa die Straßenkreuzungen als passive Objekte modelliert werden.

Abbildung 8 skizziert die beiden zu Abb. 7 gehörenden Prozesse; jedem Ereignis e_x entspricht dabei die Aktion X, welche jeweils eine Folge von Anweisungen darstellt, die den lokalen und globalen Zustand des Modells verändern kann. (Auch eine direkte Veränderung des lokalen Zustands eines anderen aktiven Objekts ist prinzipiell nicht ausgeschlossen.) Die „ereignislosen" Aktivitätsphasen zwischen den Ereignisaktionen werden hier durch *delay* mit einem Zeitparameter spezifiziert (dies entspricht *hold, work* oder *wait* in den prozeßorientierten Sprachen Simula [6] bzw. Simscript II.5 [40] oder *advance* in GPSS [62]).

Die Ereignisliste besteht stets aus höchstens einer Ereignisnotiz pro Prozeß; Abb. 9 zeigt die Situation zum Zeitpunkt 4.0, nachdem Prozeß 1 „delay (1.5)" im Anschluß an das Ereignis e_B ausgeführt hat. Die Ereignisnotizen enthalten bei der prozeßorientierten Simulation den Programmzähler der als Koroutinen realisierten Prozesse. Durch *delay* plant ein Prozeß also „sein"

Folgeereignis zu einem von ihm bestimmten Zeitpunkt selbst ein. Alternativ dazu kann ein Prozeß durch *passivate* eine Aktivität auch eine unbestimmte Zeit andauern lassen. Er ist in diesem Fall auf eine Fremdaktivierung angewiesen - im Beispiel reaktiviert Prozeß 2 Prozeß 1 zu einem bestimmten zukünftigen Zeitpunkt. (Die zunächst etwas verwirrende Sprechweise, daß ein passiver Prozeß sich in einer Aktivitätsphase befindet, zur Beendigung dieser aber reaktiviert werden muß, beruht auf den in Abschnitt 2.2.1 geschilderten dualen Sichtweisen von Simulations- und Rechenzeit.)

Die bei der aktivitätsorientierten Simulationsstrategie mögliche *zustandsgesteuerte* Auslösung von Aktivitäten ließe sich in der prozeßorientierten Simulation einfach durch eine Anweisung der Art „wait < Bedingung >" erreichen. Weil dies mit den in Abschnitt 2.2.3 geschilderten Effizienzproblemen verbunden ist, verzichten allerdings viele prozeßorientierte Simulationssprachen auf diese für den Anwender bequeme Möglichkeit.

Bei der prozeßorientierten Simulation handelt es sich prinzipiell lediglich um eine strukturierte Fassung der ereignisgesteuerten bzw. aktivitätsorientierten Simulation; bezüglich einer Beschleunigung durch Parallelität gelten daher die Bemerkungen aus Abschnitt 2.2.1 und 2.2.3. Insbesondere darf man keine unmittelbare Effizienzsteigerung erwarten, wenn jeder Prozeß durch einen eigenen Prozessor ausgeführt wird! Abbildungen 7 und 8 illustrieren nochmals die Problematik: Ereignis e_G kann nicht vor e_C ausgeführt werden, da e_C die globale Variable V setzt, wodurch die mit dem Ereignis e_G assoziierten Aktionen beeinflußt werden. Entsprechend ist bei gleichzeitiger Ausführung von e_C und e_G das Ergebnis unbestimmt. Da Prozesse i.a. miteinander interagieren und sich gegenseitig beeinflussen, lassen sich derartige Konflikte ohne eine genaue Analyse der Datenabhängigkeiten und aufwendige Sperrmechanismen grundsätzlich nicht vermeiden. Konsequenterweise würde ein „vorsichtiger" Simulator die Ereignisse streng sequentiell entsprechend der eingeplanten Zeitordnung ausführen - von den parallelen Prozessen wäre dann stets doch nur ein einziger aktiv! Dies zeigt erneut, daß die ereignisorientierte Simulation (und die prozeßorientierte Simulation als eine spezielle Ausprägung davon) prinzipiell als *inhärent sequentiell* angesehen werden muß.

2.2.5. Transaktionsorientierte Simulation. Bei der transaktionsorientierten Simulation geht man davon aus, daß es zwei Arten von Objekten gibt: *mobile* dynamische Objekte, sogenannte *Transaktionen,* und permanente *stationäre* Objekte, oft einfach als *Stationen* bezeichnet. Die Stationen bilden die Knoten eines Graphen, durch den die Transaktionen „fließen". Bei jedem Passieren eines Knotens kann eine Transaktion verzögert werden und eine Zustandstransformation bewirken. Da Transaktionen und Stationen lokale Daten besitzen und zusammengehörige Ereignisse verursachen, wird diese Simulationsmethodik oft als eine Unterklasse der prozeßorientierten Simulation angesehen. Unter anderen baut die wohl am weitesten verbreitete Simulationssprache GPSS [62] auf diesem Modell auf.

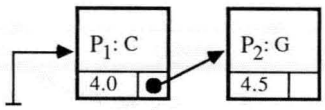

Abb. 9. Ereignisliste bei der prozeßorientierten Simulation

Die transaktionsorientierte Simulation ist besonders gut zur Modellierung von *Warteschlangensystemen* geeignet, wo z.B. Kunden oder Arbeitseinheiten bei einem Fabrikationsprozeß von einer Bedienstation (bzw. Arbeitsstation) zur nächsten weitergereicht werden und es aufgrund unterschiedlicher Bedienzeiten und Konkurrenz bei der Betriebsmittelzuteilung zu Verzögerungen, komplexen Abhängigkeiten und Einwirkungen der Objekte aufeinander kommen kann. Durch diese spezielle Weltsicht lassen sich viele betriebswirtschaftlich interessante Systeme einfach (z.B. durch graphische Flußdiagramme) beschreiben. Die realen Systemkomponenten entsprechen dabei oft unmittelbar den Transaktionen und Stationen, so daß derartige Systeme originalgetreu modelliert werden können. Zur Modellierung und Analyse von Warteschlangensystemen wurden spezifische Modellierungssprachen (z.B. HIT [4], RESQ [61]) entwickelt. Für die Modellierung und Simulation andersgearteter Systeme eignet sich das spezifische Paradigma der transaktionsorientierten Simulation allerdings weniger.

Ein Ansatz zur *Parallelisierung* - und damit zur Beschleunigung der Simulation - scheint darin zu liegen, daß entsprechend dem i.a. relativ direkt modellierten realen System viele Transaktionen gleichzeitig fließen und ggf. mehrere Stationen gleichzeitig arbeiten können. Da zudem alle Transaktionen und Stationen autonome Datenbereiche besitzen, sollten auch die Probleme der Datenabhängigkeit und Zugriffssynchronisation weitgehend wegfallen. Leider stehen einer derartigen Parallelisierung zwei Aspekte entgegen: globale Daten und „schnelle" Transaktionen.

Kommunizieren Stationen oder Transaktionen indirekt über *globale Variablen* miteinander, so sind sie nicht mehr unabhängig voneinander. Würde etwa bei einer sequentiellen Simulation zum simulierten Zeitpunkt 3.5 eine bei einer Station eintreffende Transaktion eine globale Variable V setzen und eine andere Transaktion bei einer anderen Station zum Zeitpunkt 3.7 abhängig vom Wert dieser globalen Variablen eine bestimmte Aktion auslösen (die Transaktion verzögern, an eine andere Station weiterleiten etc.), so würde eine parallele Ausführung, bei der die beiden Aktionen gleichzeitig (oder sogar in umgekehrter Reihenfolge) ausgeführt werden, i.a. ein anderes (also falsches) Ergebnis liefern. Im allgemeinen gilt, daß *eine parallele Simulation nur dann korrekt ist, wenn abhängige Aktionen in der gleichen Reihenfolge wie bei der sequentiellen Simulation - also sequentiell bezüglich der simulierten Zeit*

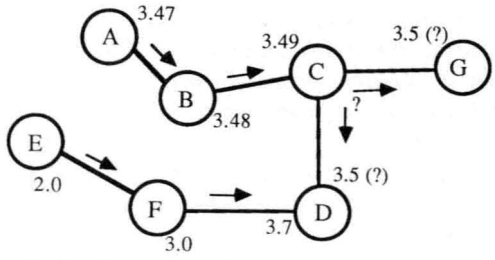

Abb. 10. Ein Netz von Stationen

– *ausgeführt werden*. Das Problem der gegenseitigen Beeinflussung über globale Variablen läßt sich offenbar grundsätzlich vermeiden, wenn globale Daten verboten werden – diese scheinbar drastische Maßnahme ist eine der Voraussetzungen der *verteilten Simulation*.

Auch wenn keine Abhängigkeiten durch globale Daten existieren, können *schnelle Transaktionen* eine effektive Parallelisierung verhindern. Das läßt sich an dem in Abb. 10 skizzierten Netz von Stationen demonstrieren. Zum Zeitpunkt 3.47 startet eine schnelle Transaktion bei Station A, die abhängig von ihrem Zustand, den sie bei Station C hat, von dieser entweder an Station G oder an Station D weitergereicht wird, wo sie jeweils zum Zeitpunkt 3.5 eintreffen kann. Eine andere Transaktion startet bei Station E zum Zeitpunkt 2.0 und trifft zum Zeitpunkt 3.7 bei Station D ein. Da die evtl. früher bei D eintreffende von A ausgehende Transaktion den Zustand in einer Weise verändern kann, daß dies dort die von E initiierte Transaktion beeinflußt, muß bei der Ausführung der zustandstransformierenden Aktionen in jedem Fall die zeitliche Reihenfolge eingehalten werden: Die bei D zum Zeitpunkt 3.7 auszuführende Aktion kann solange nicht ausgeführt werden, wie nicht feststeht, ob dort die von A initiierte Transaktion eintrifft oder nicht. Dies kann i.a. allerdings erst nach Ausführung der Aktionen bei A, B und C entschieden werden.

Das Beispiel zeigt wieder die grundsätzliche Problematik: Ohne weitere Voraussetzung müssen die Aktionen *sequentiell nacheinander* in der vorgegebenen *zeitlichen Ordnung* – die der Reihenfolge bei der sequentiellen Simulation entspricht – ausgeführt werden, damit die Simulation korrekt bleibt. Dies läßt sich auch so auffassen, daß alle Stationen *zeitsynchron* bezüglich der Simulationszeit laufen müssen und daher höchstens die (i.a. seltenen) genau gleichzeitig eingeplanten Aktionen parallel ausgeführt werden können. Hellmold untersucht detailliert Möglichkeiten der Parallelisierung transaktionsorientierter Simulationsmodelle [27] und kommt dabei zu dem ernüchternden Schluß, daß „die naheliegende Vermutung, offensichtlich parallele Systemabläufe unmittelbar auf eine ebensolche Hardwarearchitektur übertragen zu können, in ihrer Einfachheit an den komplizierten Synchronisationsmaßnahmen scheitert".

Schnelle Transaktionen sind offenbar deswegen problematisch, weil durch sie unmittelbare Abhängigkeiten zwischen verschiedenen Stationen entstehen können. (Im Extremfall könnten Transaktionen ohne Zeitverzögerung den gesamten Graphen durchlaufen.) Haben Transaktionen dagegen eine gewisse „Mindestlaufzeit", so sind die wechselseitigen Einwirkungen beschränkt und es können durch den so gewonnenen Spielraum Aktionen bzw. Ereignisse mit (leicht) unterschiedlicher Simulationszeit in beliebiger Reihenfolge (und damit bei disjunkten Zustandsbereichen in realer Zeit gleichzeitig) ausgeführt werden. Letzteres kann man so auffassen, als ob die „lokalen Uhren" einiger Objekte vorgingen; dies ist tatsächlich eine wesentliche Voraussetzung für den effektiven Einsatz der Parallelität [5]! Wie bereits bei der zeitgesteuerten Simulation erkannt wurde, spielen auch hier zeitliche Mindestgarantien eine wichtige Rolle.

3. Auf der Suche nach der Parallelität

Die Aussagen aus Abschnitt 2.2 bezüglich einer Parallelisierung waren grundsätzlich negativer Art. Es zeigte sich mehrfach, daß die ereignisorientierte Simulation einschließlich der populären prozeßorientierten und transaktionsorientierten Ausprägung prinzipiell als inhärent *sequentiell* angesehen werden muß: Eingeplante Ereignisse sind bezüglich ihres Eintrittszeitpunkts linear geordnet, und jedes zeitlich später eintretende Ereignis kann von einem früheren Ereignis abhängen. Wegen der durch die zeitliche Ordnung induzierten Kausalitätsbeziehung müssen die Ereignisse – sofern keine weiteren anwendungsbezogenen Kenntnisse vorliegen – in streng chronologischer Reihenfolge ausgeführt werden.

Trotz dieser generellen Negativaussage existieren allerdings einige Ansatzpunkte zum Einsatz von Parallelität.

3.1. Unabhängige Simulationsläufe

Viele Simulationsläufe erfordern eine ganze *Serie* von Einzelexperimenten (vgl. Abschnitt 1). Die für verschiedene Parameterwerte oder Zufallsfolgen durchgeführten Einzelläufe sind i.a. unabhängig voneinander und können gleichzeitig ausgeführt werden. Diese triviale Art der Parallelisierung vermeidet viele der sonstigen Probleme und ermöglicht sogar einen linearen speedup, sofern genügend unabhängige Einzelexperimente vorliegen. Allerdings läßt sich die Parallelisierungsmethodik nicht anwenden, wenn Simulationsläufe iterativ durchgeführt werden müssen, etwa weil die Ergebnisse eines Laufs in die Planung des nächsten einfließen.

Die Planung und Vorbereitung der Versuchsserie sowie die anschließende Auswertung der simultan anfallenden Daten erfordern geeignete anwendungsbezogene Maßnahmen. Hierfür benötigt man, genauso wie für das automatische Starten und Überwachen der Ein-

zelläufe, eine Unterstützung durch Werkzeuge, etwa analog den mittlerweile unter UNIX entwickelten parallelen ‚make'-Kommandos zum automatischen gleichzeitigen Übersetzen und Generieren unabhängiger Systemkomponenten.

Die Aufteilung des Gesamtexperiments in unabhängige Einzelläufe ist auch eine geeignete Anwendung für ein als „Prozessorfarm" genutztes Mehrrechnersystem, welches aus lokal vernetzten Workstations besteht. Die automatische Zuordnung temporär anderweitig nicht verwendeter Rechner ermöglicht eine sinnvolle Nutzung der hohen Gesamtleistung. Auch hierzu gibt es bereits - unabhängig von Simulationsanwendungen - experimentelle Systeme.

3.2. *Superrechner*

Superrechner werden oft zur Ausführung numerisch aufwendiger Simulationen eingesetzt. Ein Beispiel ist der digitale Windkanal der NASA zur Berechnung komplexer aerodynamischer Gleichungssysteme [49]. In [57] berichtet Petersen von der Verwendung von Superrechnern für große ökonomische Simulationsmodelle, wobei durch *Vektorisierung* der aus über 20000 Gleichungen bestehenden Systeme ein speedup von ca. 2 erreicht werden konnte. Da sich bei dieser speziellen Anwendung die Berechnung der Gleichungen gut in unabhängige Teile aufgliedern läßt, kann durch den Einsatz von Parallelprozessoren eine weitere Beschleunigung ermöglicht werden.

Bei derartigen Anwendungen handelt es sich allerdings um *quasi-kontinuierliche* Simulationen, denen ein mathematisches Modell aus großen Gleichungssystemen zugrunde liegt (vgl. Abschnitt 2.1). Die *ereignisorientierte* Simulation dagegen läßt sich so gut wie *nicht* vektorisieren. In [9] analysieren Chandak und Browne dieses „item of computing folklore" am Beispiel der transaktionsorientierten Simulationsmethode. Sie zeigen, daß eine partielle Vektorisierung (in Form von Transaktionsvektoren) nur unter extremen Einschränkungen möglich ist, die für praktisch keine größere Anwendung erfüllt sind. Reed et al. [59] untersuchten in mehreren Experimenten auf einer Cray X-MP verschiedene Modelle und stellten fest, daß nie mehr als 5% des Codes vektorisierbar waren und sich dadurch keine signifikante Beschleunigung ergab.

3.3. *Funktionale Verteilung*

Außer aus problembezogenen Ereignisroutinen besteht ein ereignisorientierter Simulator i.a. noch aus typischen (rechenzeitverbrauchenden) *Unterstützungsfunktionen*. Hierzu gehören

1) die Generierung von Zufallszahlen,
2) das Akkumulieren von Statistiken,
3) die Durchführung von Ein-/Ausgabe,
4) die Verwaltung der Ereignisliste.

Die Idee der Parallelisierung durch funktionale Verteilung besteht darin, möglichst jeder dieser Funktionen mindestens einen eigenen Prozessor zuzuordnen, der teilweise asynchron zu den problembezogenen Komponenten arbeiten kann [38, 71]. Da die Ereignisroutinen weiterhin sequentiell abgearbeitet werden, ändert sich dadurch aus Anwendersicht am Modell der sequentiellen Simulation nichts.

Die Realisierung der Idee erfordert ein Multiprozessorsystem, in letzter Konsequenz sogar einen hardwaremäßig auf die verschiedenen Aufgaben zugeschnittenen Spezialrechner [3]. Dennoch scheint, auch bei Vernachlässigung des Zusatzaufwands, höchstens ein sehr begrenzter speedup möglich zu sein [59]. Comfort [15, 16] behauptet zwar, daß bis zu 80% der Rechenzeit in den anwendungsunabhängigen Komponenten verbraucht werden, doch beruht diese Zahl auf einem methodischen Fehler: Für die Implementierung der Ereignisliste wird eine verkettete Liste statt eines effizienteren Verfahrens mit wesentlich niedrigerer Zeitkomplexität vorausgesetzt.

Für die ersten drei oben genannten Funktionskomponenten ist ein signifikanter Effizienzgewinn fraglich. Zwar können durch einen eigenen Prozessor Zufallszahlen verschiedener Verteilung „auf Halde" produziert werden, doch i.a. ist eine Zufallszahlenberechnung (insbesondere für die gleichförmige Verteilung im Einheitsintervall) relativ zu den sonstigen Berechnungen nicht aufwendig. So benötigen die üblicherweise verwendeten Zufallsgeneratoren nach der Lehmer'schen linearen Kongruenzmethode nur einige wenige Maschineninstruktionen. Ebenso ist der Gewinn durch eine asynchrone Ausführung von Statistikfunktionen (Mittelwertbildung, Varianzberechnung etc.) zweifelhaft. Eine asynchrone Durchführung der Ein-/Ausgabe kann gegebenenfalls sowieso vom Betriebssystem und vorhandenen E/A-Kanälen vorgenommen werden und verliert bei zunehmender Hauptspeichergröße sogar noch an Bedeutung.

Der angebliche Engpaß bei der Verwaltung der Ereignisliste hat zu einigen Versuchen geführt, diese selbst zu parallelisieren [3, 15, 16, 70]. In [36] und [37] kritisiert Jones diese Ansätze zu Recht - bei Verwendung eines Verfahrens mit logarithmischer Zeitkomplexität (vgl. Abschnitt 2.2.1) ist auch durch Parallelität keine signifikante Beschleunigung mehr erreichbar[3]. Geeignete logarithmische priority-queue-Implementierungen sind spätestens seit dem Erscheinen von Knuths Standardwerk [41] allgemein bekannt. Die Kritik trifft jedoch noch mehr zu, seitdem sogar Implementierungen von Ereignislisten mit linearer Zeitkomplexität bekannt sind [7].

Die Überlegungen zeigen, daß eine signifikante Beschleunigung eines ereignisorientierten Simulators durch funktionale Verteilung kaum möglich scheint.

[3] „How can you beat an $O(\log n)$ algorithm?" (J. Henriksen dazu in [37]). Die Antwort liefert Brown [7]: durch einen $O(1)$-Algorithmus!

3.4. *Unabhängige Ereignisse*

Die negativen Erkenntnisse bezüglich Vektorisierbarkeit und funktionaler Verteilung führen dazu, die Parallelisierbarkeit unabhängiger Ereignisse genauer zu untersuchen. Allerdings ist bereits die Entdeckung der Unabhängigkeit von Ereignissen nicht einfach.

Ein Ereignis A ist *direkt* von Ereignis B abhängig, wenn B das Ereignis A eingeplant hat oder A lesend auf Zustandsvariablen zugreift, die B verändert. Die *indirekte* Abhängigkeit wird durch die Transitivität vermittelt. Zwei Ereignisse sind *unabhängig* voneinander, wenn keines vom anderen direkt oder indirekt abhängt. Im Unterschied zur zeitlichen Ordnung und der durch sie induzierten potentiellen Kausalitätsbeziehung ist die Abhängigkeit lediglich eine *partielle* Ordnung. Gelegentlich wird jedoch übersehen, daß die Unabhängigkeit von Ereignissen noch nicht garantiert, daß diese in beliebiger Reihenfolge oder sogar gleichzeitig simuliert werden können [36]: Wenn zwei (unabhängige) Ereignisse den gleichen Zustandsraum verändern, so kann deren Ausführungsreihenfolge entscheidend sein. In diesem Fall legt die logische Eintrittszeit der Ereignisse die korrekte Semantik fest. Mehrere paarweise unabhängige Ereignisse, die nicht in dieser Konfliktrelation stehen, können dagegen *gleichzeitig* simuliert werden. Im Extremfall wird allerdings nur das wie bei der sequentiellen Simulation eindeutig bestimmte global nächste Ereignis simuliert. Bei der gleichzeitigen Simulation von Ereignissen ist außerdem die Atomarität der Ereignisroutinen zu beachten.

Die Konfliktsituation abhängiger Ereignisse und die Parallelisierungsproblematik erinnert an das concurrency-control-Problem transaktionsorientierter Datenbanksysteme. Allerdings geht es bei der parallelen Simulation nicht nur darum, eine gleichzeitige Ausführung konfliktträchtiger Aktionen zu verhindern, sondern diese in einer korrekten „Reihenfolge" auszuführen, bei gleichzeitiger Maximierung der Parallelität.

Das automatische Erkennen der Unabhängigkeit von Ereignissen ist nicht einfach, andererseits wäre aber eine manuelle Spezifikation der Abhängigkeitsrelation zu umständlich und zu fehleranfällig. Bei der *verteilten Simulation* nutzt man u. a. *disjunkte Zustandsräume* und *zeitliche Mindestgarantien* zur Feststellung der Unabhängigkeit, oder man riskiert bewußt temporäre Verletzungen der Unabhängigkeit und „repariert" gegebenenfalls automatisch die Konsequenzen. Mit diesen Techniken gelingt schließlich - innerhalb gewisser Grenzen - tatsächlich eine Parallelisierung der ereignisorientierten Simulation. Auf die damit verbundenen interessanten Aspekte wird in einem späteren Beitrag ausführlich eingegangen.

Literatur

1. Aho, V.A., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Reading, MA: Addison-Wesley 1983
2. Asshoff, H., Kluwig, H.: Verteilte Simulation diskreter Ereignisse. Diplomarbeit, Universität Bonn und Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin, 1988
3. Barel, M.: The discrete event simulation computer - DESC. Simuletter *15* (2), 9-15 (1984)
4. Beilner, H., Stewing, F.J.: Concepts and Techniques of the Performance Modelling Tool HIT. European Simulation Multiconference Vienna 1987
5. Berry, O., Jefferson, D.: Critical Path Analysis of Distributed Simulation. Proceedings of the Distributed Simulation Conference, San Diego, pp. 57-60. San Diego, CA:SCS 1985
6. Birtwistle, G.M., Dahl, O.J., Myrhaug, B., Nygaard, K.: Simula Begin. Philadelphia, PA Auerbach 1973
7. Brown, R.: Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem. Commun. ACM *31* (10), 1220-1227 (1988)
8. Bryant, R.E.: Simulation on a Distributed System. Proceedings of the 1st International Conference on Distributed Computing Systems, pp. 544-552. Silver Spring, MD:IEEE 1979
9. Chandak, A., Browne, J.C.: Vectorization of Discrete Event Simulation. Proceedings of the International Conference on Parallel Processing, pp. 359-361. Silver Spring, MD:IEEE 1983
10. Chandy, K.M., Holmes, V., Misra, J.: Distributed simulation of networks. Computer Networks *3*, 105-113 (1979)
11. Chandy, K.M., Lamport, L.: Distributed snapshots: Determining global states of distributed systems. ACM Trans. Computer Systems *3* (1), 63-75 (1985)
12. Chandy, K.M., Misra, J.: A Non-Trivial Example of Concurrent Processing: Distributed Simulation. Proceedings of COMPSAC, pp. 822-826. Silver Spring, MD:IEEE 1978
13. Chandy, K.M., Misra, J.: Distributed simulation: A case study in design and verification of distributed programs. IEEE Trans. Software Eng. SE-*5* (5), 440-452 (1979)
14. Chandy, K.M., Misra, J.: Asynchronous distributed simulation via a sequence of parallel computations. Commun. ACM *24* (4), 198-205 (1981)
15. Comfort, J.C.: The Design of a Multi-Microprocessor Based Simulation Computer - I. Proceedings of the 5th Annual Simulation Symposium, pp. 45-53. Silver Spring, MD:IEEE 1982
16. Comfort, J.C.: The simulation of a master-slave event set processor. Simulation *42*, 117-124 (1984)
17. Computer Magazin: Aktuelles Interview: „Cray 2 schon voll?". Computer Magazin Heft *7/8*, 5-6 (1988)
18. Czerny, G.: Die Simulation von Verkehrssystemen, in: Schöne A. Simulation technischer Systeme. Band 3 „Simulation diskreter Systeme", pp. 145-159. München: Hanser 1974
19. Fishman, G.S.: Concepts and Methods in Discrete Event Digital Simulation. New York: John Wiley 1973
20. Franklin, M.A., Wann, D.F., Wong, K.F.: Parallel Machines and Algorithms for Discrete-Event Simulation. Proceedings of the International Conference on Parallel Processing, pp. 449-458. Silver Spring, MD:IEEE 1984
21. Fujimoto, R.M.: Lookahead in Parallel Discrete Event Simulation. Proceedings of the International Conference on Parallel Processing, pp. 34-41. Silver Spring, MD:IEEE 1988
22. Fujimoto, R.M.: Performance Measurements of Distributed Simulation Strategies. Proceedings of the Distributed Simulation Conference, San Diego, pp. 14-20. San Diego, CA:SCS 1988
23. Gardner, M.: Mathematical games - The fantastic combinations of John Conway's new solitaire game of life. Sci. Am. *223* (4), 120-123 (1970)
24. Gilmer, J.B., Hong, J.P.: Replicated State Space Approach for Parallel Simulation. Proceedings of the Winter Simulation Conference, pp. 430-434. Silver Spring, MD:IEEE 1986
25. Gosh, J.B.: Asynchronous Control of Discrete Event Simulation. Proceedings of the 8th Annual Simulation Symposium, pp. 255-263. Silver Spring, MD:IEEE 1985
26. Hahn, W., Anger, H., Hagerer, A., Schuster, B.: Ein Multi-Transputernetz für die Simulation digitaler Systeme. Chip Plus *8*, 18-22 (1988)
27. Hellmold, K.U.: Der Simulationsrechner SIMPLEX. Arbeitsberichte des Instituts für mathematische Maschinen und Datenverarbeitung (Informatik), Erlangen, Bd. 18, No. 4, 1985

28. Hooper, J.W.: Activity scanning and the three-phase approach. Simulation *47*(5), 210–211 (1986)
29. Hooper, J.W.: Strategy-related characteristics of discrete-event languages and models. Simulation *46*(4), 153–159 (1986)
30. Hooper, J.W., Reilly, K.D.: An algorithmic analysis of simulation strategies. Int. J. Computer Infor. Sci. *11*(2), 101–122 (1982)
31. Jefferson, D.R.: Virtual time. ACM Trans. Programm. Languages Systems *7*(3), 404–425 (1985)
32. Jefferson, D.R., Beckmann, B., Hughes, S., Levy, E., Litwin, T., Spagnuolo, J., Havrus, J., Wieland, F., Zimmermann, B.: Implementation of Time Warp on the Caltech Hypercube. Proceedings of the Conference on Distributed Simulation, pp.70–75. San Diego, CA:SCS 1985
33. Jefferson, D.R., Beckmann, B., Wieland, F., Blume, L., Diloreto, M., Houtalas, P., Laroche, P., Sturedevant, K., Typman, J., Van Warren, Wedel, J., Younger, H., Bellenot, S.: Distributed Simulation and the Time Warp Operating System. Proceedings of the 11th ACM Symposium on Operating Systems Principles, pp.77–93. New York: ACM 1987
34. Jefferson, D.R., Sowizral, H.: Fast Concurrent Simulation Using the Time Warp Mechanism. Proceedings of the Conference on Distributed Simulation, pp.63–69. San Diego, CA:SCS 1985
35. Jones, D.W.: An empirical comparison of priority-queue and event-set implementations. Commun. ACM *29*(4), 300–311 (1986)
36. Jones, D.W.: Concurrent Simulation: An Alternative to Distributed Simulation. Proceedings of the Winter Simulation Conference, pp.417–423. Silver Spring, MD:IEEE 1986
37. Jones, D.W., Henriksen, J.O., Pegden, C.D., Sargent, R.G., O'Keefe, R.M., Unger, B.: Implementation of time. Proceedings of the Winter Simulation Conference, pp.409–416. Silver Spring, MD:IEEE 1986
38. Kaudel, F.J.: A literature survey on distributed discrete event simulation. Simuletter *18*(2), 11–21 (1987)
39. Keller, H.B.: Parallele Simulationsmethoden zur Ausführung komplexer Modelle, in: J.Valk (Hrsg.): 18.GI-Jahrestagung, pp.270–278. Informatik-Fachberichte, Band 187. Berlin-Heidelberg-New York: Springer 1988
40. Kiviat, P.J., Markowitz, H.M., Villanueva, R.: SIMSCRIPT II.5 Programming Language. Los Angeles: CACI 1983
41. Knuth, D.E.: The art of computer programming, Vol.III. Reading, MA: Addison-Wesley 1973
42. Kreutzer, W.: System simulation programming styles and languages. Reading, MA: Addison-Wesley 1986
43. Krüger, S.: Simulation - Grundlagen, Techniken, Anwendungen. Berlin: Walter de Gruyter 1975
44. Lomow, G., Cleary, J., Unger, B., West, D.: A Performance Study of Time Warp. Proceedings of the Distributed Simulation Conference, San Diego. San Diego, CA:SCS 1988
45. Lubachevsky, B.D.: Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks. Proceedings of the ACM SIGMETRICS Conference, pp.12–21. New York: ACM 1988
46. Ludwig, T.: Parallelisierung des Monte Carlo Verfahrens nach Bird. Bericht Nr.87/2 des Sonderforschungsbereichs 182 „Multiprozessor- und Netzwerkkonfigurationen", Teilprojekt B1, Universität Erlangen-Nürnberg 1987
47. Misra, J.: Distributed discrete-event simulation. Comput. Surveys *18*(1), 39–65 (1986)
48. Nance, R.E.: The time and state relationship in simulation modeling. Commun. ACM *24*(4), 173–179 (1981)
49. Nasa: NASA numerical aerodynamic simulation facility. Simulation *44*(5), 257–258 (1985)
50. Neelamkavil, F.: Computer Simulation and Modelling. New York: John Wiley 1987
51. Nentwig, D.: Techniken für die Simulation zeitdiskreter Systeme. Fernuniversität Hagen, Informatik Berichte Nr.78, 1988
52. Nicol, D.M.: Performance Issues for Distributed Battlefield Simulations. Proceedings of the Winter Simulation Conference, pp.624–628. Silver Spring, MD:IEEE 1987
53. O'Keefe, R.M.: The three-phase approach: A comment on strategy-related characteristics of discrete-event languages and models. Simulation *47*(5), 208–211 (1986)
54. Page, B., Bölckow, R., Heymann, A., Kadler, R., Liebert, H.: Simulation und moderne Programmiersprachen. Fachberichte Simulation, Band 8. Berlin-Heidelberg-New York: Springer 1988
55. Peacock, K.J., Manning, E.G., Wong, J.W.: Synchronization of distributed simulation using broadcast algorithms. Computer *13*(4), 3–10 (1980)
56. Peacock, K.J., Wong, J.W., Manning, E.G.: Distributed simulation using a network of processors. Computer Networks *3*, 44–56 (1979)
57. Petersen, C.E.: Computer simulation of large-scale econometric models: Project link. Int. J. Supercomputer Applicat. *1*(4), 31–53 (1987)
58. Reed, D.A., Malony, A.: Parallel Discrete Event Simulation: The Chandy Misra Approach. Proceedings of the Distributed Simulation Conference, San Diego, pp.8–13. San Diego, CA:SCS 1988
59. Reed, D.A., Malony, A., McCredie, B.D.: Parallel discrete event simulation using shared memory. IEEE Trans. Software Eng. SE-*14*(4), 541–553 (1988)
60. Reeves, C.M.: Complexity analysis of event set algorithms. Comput. J. *27*(1), 72–79 (1984)
61. Sauer, C.H., MacNair, E.A., Salza, S.: A language for extended queuing network models. IBM J. Res. Dev. *24*(6), 747–755 (1980)
62. Schriber, T.: Simulation using GPSS. New York: John Wiley 1974
63. Sleator, D.D., Tarjan, R.E.: Self-adjusting binary search trees. J. ACM *32*(3), 652–686 (1985)
64. Solchenbach, K., Thole, C.A., Trottenberg, U.: Parallel Multigrid Methods: Implementation on SUPRENUM-like Architectures and Applications. SUPRENUM Report 4. Bonn: SUPRENUM 1987
65. Spriet, J.A., Vansteenkiste, G.C.: Computer-aided Modelling and Simulation. New York: Academic Press 1982
66. Swope, S.M., Fujimoto, R.M.: Optimal Performance of Distributed Simulation Programs. Proceedings of the Winter Simulation Conference, pp.612–617. Silver Spring, MD:IEEE 1987
67. Trottenberg, U.: On the SUPRENUM Conception. Proceedings of the International Conference on Parallel Computing. Amsterdam: North-Holland 1986
68. Trottenberg, U., Wypior, P.: Rechnerarchitekturen für die numerische Simulation auf der Basis superschneller Lösungsverfahren I. GMD-Studien 88, Gesellschaft für Mathematik und Datenverarbeitung, St.Augustin 1984
69. Wagner, D.B., Lazowska, E.D., Bershad, B.N.: Techniques for Efficient Shared Memory Parallel Simulation. Technical Report 88-04-05, Department of Computer Science, University of Washington, Seattle, 1988
70. Wilson, A.: Parallelization of an event driven simulator for computer systems. Simulation *49*(2), 72–78 (1987)
71. Wyatt, D.L., Young, R.E., Sheppard, S.: An Experiment in Microprocessor-based Distributed Digital Simulation. Proceedings of the Winter Simulation Conference, pp.271–277. Silver Spring, MD:IEEE 1983

Eingegangen 25.11.1988; in überarbeiteter Form 2.5.1989

Dipl. Inform. Friedemann Mattern
Horst Mehl
Universität Kaiserslautern
SFB 124: VLSI-Entwurf und Parallelität
Erwin-Schrödinger-Straße
D-6750 Kaiserslautern