
Gerhard Maierhöfer Georg Skorobohatyj

Parallel-TRAPEX

(Ein paralleler adaptiver Algorithmus
zur numerischen Integration; seine Implementierung
für SUPRENUM-artige Architekturen mit SUSI)

Inhaltsverzeichnis

0. Einleitung	1
1. Grundlagen	2
1.1 Numerischer Hintergrund des sequentiellen Trapex	2
1.2 Das SUPRENUM Simulationssystem	4
1.3 Erläuterungen zum Meßverfahren	4
2. Analyse der Problemstellung; Folgerungen	6
2.1 Überlegungen zur Parallelisierung	6
2.2 Prozessoranzahl und erreichbare Beschleunigung	9
2.3 Architekturüberlegungen	12
3. Meßdaten	14
3.1 Beschreibung und Vergleich der parallelen TRAPEX-Varianten .	14
3.2 Betrachtung einiger ausgewählter Beispiele	16
Schlußbetrachtung	32
Anhang	33
Literatur	50

Parallel-TRAPEX

(Ein paralleler adaptiver Algorithmus
zur numerischen Integration; seine Implementierung
für SUPRENUM-artige Architekturen mit SUSI)

G. Maierhöfer G. Skorobohaty

Abstract

This paper describes some ways of transforming a sequential adaptive algorithm for numerical evaluation of an integral (Romberg-Quadrature with polynomial Extrapolation method) to a parallel one, such as have been implemented by the authors. We developed an algorithm which preserves the sequential adaptivity and is capable of running on various architectures, dynamically controlling the number of active processors depending on the problem. To study the time behaviour, we used the simulator SUSI (SUPRENUM SImulator) which is able to simulate SUPRENUM-like architectures. Results are given in part 3.

Key words: Romberg quadrature; numerical integration;
parallel adaptive algorithm;
SUSI; simulation; SUPRENUM; MIMD-Fortran;
computer architecture; granularity; load balancing;
master-slave-principle

Zusammenfassung

Dieser Artikel beschreibt die Übertragung eines sequentiellen adaptiven Algorithmus zur numerischen Berechnung eines Integralwertes (Romberg-Quadratur mit polynomialem Extrapolation) in verschiedene parallele Versionen. Wir entwickelten einen Algorithmus, der nicht nur die sequentielle Adaptivität beibehält, sondern auch auf unterschiedlichen Architekturen lauffähig ist und dynamisch (problemabhängig) die Zahl der aktiven Prozessoren steuert. Um das Zeitverhalten zu untersuchen, verwendeten wir den Simulator SUSI (SUPRENUM SImulator), der SUPRENUM-artige Architekturen simuliert. Meßergebnisse sind in Abschnitt 3 angegeben.

Schlüsselwörter: Romberg-Quadratur; numerische Integration;
Paralleler adaptiver Algorithmus;
SUSI; Simulation; SUPRENUM; MIMD-Fortran;
Rechnerarchitektur; Granularität; Load Balancing;
Master-Slave-Prinzip

0. Einleitung

Der vorliegende Bericht beschreibt eine (auf dem Master-Slave-Prinzip basierende) Parallelisierung eines sequentiellen Algorithmus zur numerischen Berechnung eines Integralwertes (Romberg-Quadratur). Sowohl der sequentielle wie auch der parallele Algorithmus ist adaptiv; d.h. im sequentiellen Fall erfolgt eine problemabhängige Schrittweitensteuerung, zu der im parallelen Fall eine problemabhängige Steuerung der Anzahl der verwendeten Prozessorknoten hinzukommt. Der Algorithmus wurde für eine SUPRENUM-artige Maschine (busgekoppelte Prozessoren, die in Clustern zusammengefaßt werden) implementiert und mit dem SUPRENUM-Simulator (SUSI) getestet. Der gefundene Ansatz ist aber auch auf anderen Architekturen realisierbar (entsprechende Arbeiten werden derzeit durchgeführt). Meßergebnisse liegen sowohl für eine feingranulierte (horizontale) wie auch für eine grobgranulierte (vertikale) Parallelisierung vor. Zum testen wurden die Beispiele des Testset aus [2] verwendet, die eine Teilmenge aus [7] und [11] bilden.

Im ersten Abschnitt wird ein Abriß der zugrunde liegenden numerischen Methode gegeben und der Simulator (kurz) beschrieben. Auf weiterreichende Literatur wird verwiesen. Der zweite Abschnitt beschreibt die Ansätze, die für die Parallelisierung gewählt wurden. In ihm sind auch Überlegungen zur Prozessoranzahl, Effizienz und zu einer möglichen optimalen Architektur enthalten. Im dritten Abschnitt sind die Meßergebnisse zusammengefaßt. Im Anhang A ist der Grundalgorithmus beschrieben. In Anhang B liegt die SUSI-Implementierung für TparE bei. Anhang C zeigt die Graphen der verwendeten Testbeispiele. Sie wurden mit einem modifizierten sequentiellen TRAPEX berechnet und mit dem im ZIB entwickelten GRAZIL [12] geplottet. In diesem Teil sind auch die Bilder 2 bis 6 enthalten.

1. Grundlagen

1.1 Numerischer Hintergrund des sequentiellen Trapex

Das im nachfolgenden Text als TRAPEX bezeichnete Fortran77-Unterprogramm verwendet zur numerischen Integration die von Romberg [15] beschriebene Methode. Die Theorie ist hier soweit skizziert, wie sie — nach unserer Meinung — zum Verständnis der Parallelisierungsproblematik notwendig ist. Der Wert eines Integrales

$$IW := \int_{t_a}^{t_b} f(t) dt \quad \text{im Intervall } [t_a, t_b] \quad \text{und } f(t) \text{ im}$$

Intervall genügend oft differenzierbar, wird durch die Trapez-Summe TS berechnet:

$$TS_1 := \frac{f(t_a) + f(t_e)}{2} * h_1$$

für $i > 1$ $TS_i := \left[\frac{f(t_a) + f(t_e)}{2} + \sum_{j=2}^{n(i)-1} f(t_a + j * h_i) \right] * h_i$

mit $h_1 = \frac{t_e - t_a}{n(i)}$,

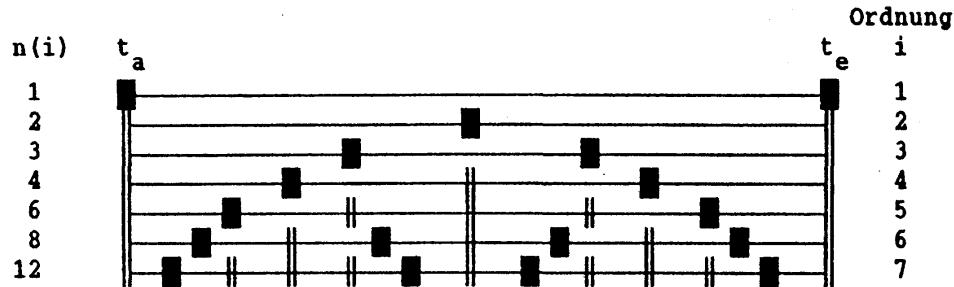
wobei $(n_i)_{i \in N} = 1, 2, 3, 4, 6, 8, 12, \dots$ die (Bulirsch-) Folge von ganzen Zahlen ist, die von Bulirsch [4] als geeignet beschrieben wird. Der Vorteil bei der Verwendung dieser Folge besteht — gegenüber z.B. einer Binärfolge (1, 2, 4, 8, 16, 32, ...) — in der Tatsache, daß ab $i = 4$ Funktionswerte, die bei $j = i - 2$ berechnet wurden, wieder verwendet werden können, wenn TS_i mit Hilfe der in TS_{i-2} schon berechneten Fkt-Werte f ermittelt wird. Dies gilt natürlich auch für die Binärfolge, doch ist die Anzahl der zu berechnenden Fkts ab $i = 4$ wesentlich größer. Die Frage, welches TS_i zur Berechnung des Integralwertes herangezogen wird und wie ein gegebenes Intervall $[a, b]$ optimal in Teilintervalle $[t, t+H]$ schrittweise aufgeteilt werden kann, wird in [5] und das verwendete Konvergenzmodell in [6] behandelt. Die Länge H eines Teilintervall es ist eine Funktion des linken Randpunktes t und der geforderten Genauigkeit, die ein Eingabeparameter des Trapex ist. Es ergibt sich damit eine Aufteilung des Grundintervalles $[a, b]$ mit Teilungspunkten $t_1 = a < t_2 < \dots < t_n = b$ und optimalen $H_k := t_{k+1} - t_k$. Auf jedes Teilintervall $[t_k, t_k + H_k]$ wird nach jeder neuen Berechnung eines TS ein polynomiales Extrapolationsverfahren angewandt und über eine Fehlerabschätzung festgestellt, ob Konvergenz vorliegt und die geforderte Genauigkeit erreicht ist. Für die Polynom-Extrapolation (Aitken-Neville Algorithmus) gilt (siehe auch [6])

die Dreiecksregel:

$$TE_{i,1} = TS_i ; \quad a \leq i \leq 7 , \quad 2 \leq k \leq i$$

$$\text{und } TE_{i,k} = TE_{i,k-1} + \frac{TE_{i,k-1} - TE_{i-1,k-1}}{\left[\frac{n_1}{n_{i-k+1}} \right]^2 - 1}$$

Das beschriebene Verfahren ist adaptiv. Für jedes TS_i wird, falls die geforderte Genauigkeit nicht erreicht wird, mit Hilfe einer Aufwandabschätzung entschieden, ob ein TS_{i+1} berechnet und/oder die Berechnung mit einem kleineren H_k wiederholt wird, bzw. bei erfolgreicher Berechnung des Näherungswertes eines Teilintervall wird überprüft, ob für das nächste eine größere Länge H_{k+1} verwendet werden kann. Bei jedem erfolgreichen oder gescheiterten Berechnen eines Integralwertes über einem Teilintervall wird auch eine optimale Ordnung i ermittelt. Der Algorithmus ist damit in der Lage, entsprechend der lokalen Information (sich ändernder Krümmungsradius des Graphen der Funktion) ein H_k zu wählen, so daß ein optimales i für TS_i gewählt werden kann. Insgesamt ist es eine Optimierungsstrategie, die zum Ziel hat, die Anzahl der zu berechnenden Funktionswerte über das Gesamtintervall zu minimieren, da man bei der Konzeption des Algorithmus davon ausging, daß diese Funktionswerte sequentiell berechnet werden. [Bild 1] zeigt bei welcher Ordnung i schon berechnete Fkt-Werte aus einer schon "früher" berechneten Ordnung $i-2$ wieder verwendet werden können.



bezeichnen FKt-Werte, die neu berechnet werden müssen
schon berechnete Fkt-Werte, die übernommen werden.

[Bild 1]

Man kann daraus ersehen, daß nur für die Ordnung 6 und 7 die Anzahl von 4 Prozessoren für die parallele Funktionsberechnung benötigt wird. Bei den niedrigeren Ordnungen sind (max.) 2 ausreichend.

1.2 Das SUPRENUM Simulationssystem

Das Simulationssystem SUPrenum SImulator (SUSI) besteht aus den Komponenten:

- a) SUSI-Interface
- b) graphische Darstellung
 - I) des dynamischen Ablaufes einer Simulation
 - II) einer Statistik der verwendeten Ressourcen (Prozessor, Bus, Mailbox)
 - III) statische Darstellung der Simulation

Teil a) wurde bei der GMD entwickelt, Teil b) bei der SUPRENUM GmbH. SUSI kann zur Simulation SUPRENUM-artiger Architekturen verwendet werden (d.h. Local Memory, Bus-Verbindungen zwischen den Prozessorknoten, Betriebssystem nach dem Client-Server-Prinzip).

Daten, die von a) geliefert werden, können (mit Hilfe des UNIX-Pipelining) an eines der Programme unter b) übertragen werden. Damit kann ein Simulationslauf in "Realzeit" erfolgen. Beim SUSI des ZIB existiert noch eine Inkonsistenz der Daten, die a) sendet und b) III) benötigt.

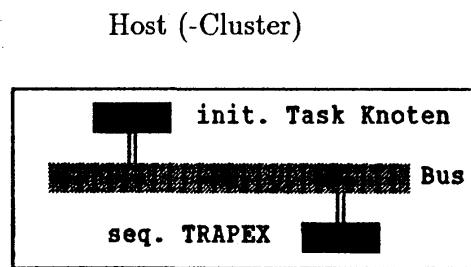
Das SUSI-Interface wird mit dem übersetzten Benutzerprogramm zusammengebunden und stellt dann das Simulationsprogramm dar. Benutzerprogramme sollen auch ohne wesentliche Abänderungen auf der realen SUPRENUM-Maschine lauffähig sein. SUSI gehört damit zur Klasse der "Benutzerprogramme ausführenden" Simulationssysteme.

1.3 Erläuterungen zum Meßverfahren

Daß SUSI keine reale Zeitmessung vollzieht (es können — gewichtete — Statements pro Zeiteinheit gezählt werden), wurde zuerst der seq. TRAPEX-Algorithmus implementiert, um einen Vergleichsmaßstab zu haben.

Bei der SUPRENUM-Maschine gibt es einen Host, das ist ein Prozessor, der die Verbindung zur Außenwelt herstellt und der mit Hilfe einer sogenannten "initialen Task" des Betriebssystems die gewünschte Konfiguration (wie viele Cluster,

wie viele Prozessoren pro Cluster, usw.) für einen Benutzerprogrammlauf bereitstellt. Im Gegensatz dazu können bei SUSI sogenannte Zeilen -und Spalten-Host (-Cluster) [3] definiert werden. Bei unserer Hardware-Konfiguration läuft das sequentielle TRAPEX als Task auf einem separaten Prozessor in einem Host-Cluster, es wird mit Hilfe des Treiberprogrammes und des Message Passing (SEND) mit Daten versorgt (und nicht wie auf einer seq. Maschine über einen Prozedur-Aufruf mit Parameterbereich).



Die initiale Task (des Betriebssystems) und das Treiberprogramm (Usertask) für das sequentielle TRAPEX laufen auf demselben Knoten des Host. Das TRAPEX läuft aber auf einem anderen Knoten innerhalb desselben Host. Damit ist ein Vergleich der durch SUSI angegebenen Simulationszeiten möglich, denn es werden — im sequentiellen Programm — nicht nur die reinen Laufzeiten des Trapex (Anzahl der durchlaufenen Anweisungen) hiermit ermittelt, sondern (wie im parallelen Fall) auch die Zeiten erfaßt, die SUSI zur Kommunikation zwischen dem Host und dem TRAPEXKnoten mißt. Damit ist eine Normierung auf diese Zeitwerte bei den verschiedenen Parallel-TRAPEX-Versionen möglich.

Alle Messungen der verschiedenen Versionen des parallelen TRAPEX wurden für eine Konfiguration einer SUPRENUM Architektur nach [Bild 6] durchgeführt.

2. Analyse der Problemstellung; Folgerungen

2.1 Überlegungen zur Parallelisierung

Im nachfolgenden Text sei unter einem Subintervall (SI) der Teil des zu integrierenden Bereiches verstanden, der einer TRAPEX-Task zur Bearbeitung übergeben wird; unter Teilintervall (TI) solch ein Bereich daraus, den diese Task selbst bearbeitet und unter Restintervall (RI) einen, den sie abgibt.

Eine aus der Integralrechnung folgende Möglichkeit der Parallelisierung besteht darin, den Integralwert für eine Funktion über ein Intervall I als Summe der Werte über n Subintervalle zu berechnen. Bei dem seq. TRAPEX ist diese Unterteilbarkeit ein wesentlicher Bestandteil (Schrittweitensteuerung) des Algorithmus.

Die Analyse des Informationsflusses des sequentiellen TRAPEX [Bild 2] (auf eine detailliertere Darstellung der trapezoidalen Summe analog zu [1] wird hier verzichtet) ergibt folgende Möglichkeiten:

- a) Die Berechnung der Funktionswerte (an den Stützstellen) für die Trapez-Summen erfolgt in einer Schleife. Diese Werte können invariant (also parallel) berechnet werden. Dabei sind folgende Fälle möglich:
 - 1) Es werden alle Fkt-Werte bis zur Ordnung 7 parallel berechnet. Dies kann insbesonders während der Initialisierungsphase der Parameter für die Aufwandabschätzung sinnvoll sein !
 - 2) Je nach vorhandener oder gewünschter Prozessorzahl weitere Fkt-Werte für die Ordnung n ; $2 \leq n \leq 7$.
- b) Aufteilung der Berechnung nach einer Reduktion der Schrittweite h .
- c) Aufteilung der Berechnung nach einem “basic step”.

Zuerst wurde eine Feingranulierung (Paralellisierung der Berechnung der Funktionswerte für die Trapez-Stützstellen Methode a)) untersucht. Dies wird auch in der Literatur als horizontale Parallelisierung beschrieben. Dabei ergab sich ein sehr hohes Message-Aufkommen. Pro Teilintervall-Berechnung müssen maximal 16 Funktionswerte (Ordnung 7) berechnet werden, was zu 34 Messages führt. Bei ca. 100 Teilintervallen entstehen also in der Regel weit über 1000 Messages. Durch diesen Overhead (im Vergleich zur sequentiellen Lösung) an Messages werden Zeitgewinne durch das parallel Rechnen neutralisiert, so daß mit SUSI bei dieser Granulierungsart keine signifikante Beschleunigung gemessen wurde. Es sei darauf hingewiesen, daß die Messages auf einer Inter-Cluster-Verbindung nicht parallel übertragen werden können, damit SUSI nur das Buskonzept des SUPRENUM1-Rechners simuliert werden kann. Andere Verbindungen, z.B. LINKS (Transputer), können nicht simuliert werden.

Als nächstes wurde eine vertikale Parallelisierung (Grobgranulierung) implementiert. Hier wird das zu integrierende Intervall wie oben angedeutet in so viele Subintervalle aufgeteilt, wie CPUs eingesetzt werden können oder sollen. Es wird pro CPU eine TASK initiiert, die aus dem (vollständigen) sequentiellen TRAPEX-Algorithmus, der um Message-Passing- und Subintervall-Optimierungs-Teile erweitert wurde, besteht [Bild 3]. Diese Version des parallelen TRAPEX kann sich statisch an die Hardware adaptieren. Die Knoten- bzw. Task-Anzahl ist ein Eingabeparameter des Algorithmus. Der auf dem Host ablaufende initiale Prozeß initiiert die geforderte Anzahl von (sequentiellen) TRAPEX-Tasks, dabei wird die Zahl vom Benutzer vorgegeben oder sie könnte auch dem initiierenden Prozeß durch das Betriebssystem vorgegeben werden. Auf jedem Prozessor läuft genau eine TRAPEX-Task. Die Zahl der parallel arbeitenden Tasks ändert sich während der gesamten Dauer der Integral-Berechnung nicht. Die Größe der Subintervalle kann entweder über die Anzahl der zur Verfügung stehenden CPUs oder über das Verhältnis L_i/h_{\max} (L_i : Intervallänge; h_{\max} : maximale zulässiges h) bestimmt werden. Terminiert eine TASK, nachdem ein Teilintegralwert erfolgreich berechnet wurde, bleibt diese CPU "unbeschäftigt" (Prozeß und Prozessor im *Busy-Waiting-Zustand* (BW)). Simulationen für Beispiele, bei denen sich in einem Subintervall die Steigungen der Kurve stark ändern, liefern Auslastungs-Diagramme, die zeigen, daß in der Regel eine CPU eine höhere Rechenleistung zu erbringen hat als alle anderen CPUs zusammen. Hier ist ein Ansatzpunkt, um eine stärker problemorientierte Adaptivität des parallelen Algorithmus zu erreichen.

Im sequentiellen TRAPEX-Algorithmus gibt es zwei Stellen, an denen eine Lastverteilung erfolgen kann:

- 1) [Bild 2] Fall b): wenn für ein Teilintervall keine Konvergenz erreicht werden kann, erfolgt eine Anpassung (Verkleinerung) des aktuellen Trapez-Höhenwertes h . Die Schrittweite für ein Teilintervall wird also reduziert. Die Berechnungen des Extrapolationsfehlers für dieses Intervall ergab einen zu großen Wert (implizit gewinnt man eine lokale Kenntnis bzw. Abschätzung über den Kurvenverlauf).
- 2) [Bild 2] Fall c): wenn ein Teilintervall erfolgreich berechnet wurde, wird das nächste Teilintervall aus dem (potentiellen) Restintervall mit einem aktualisierten h -Wert ausgewählt und damit dann weitergerechnet.

Damit entsteht eine dynamische problemorientierte Lastanpassung [Bild 4].

Im Falle 1) kann eine Lastabgabe in dem Sinne erfolgen, daß ein Teil des oder das ganze Restintervall an einen anderen Prozeß zur Berechnung abgegeben wird und nur der verbleibende Teil von der lastabgebenden CPU berechnet wird.

$$L_{RI} = \frac{L_{SI} - \sum(L_{TI})}{c}$$

Wir untersuchten den Einfluß verschiedener Werte von c auf das Verhalten mehrerer Parallel-TRAPEX-Varianten.

Auch im Falle 2) ist die Frage, welchen Wert RI haben soll. Die Abgabe von Restlast kann im Falle 1) unbedingt erfolgen, denn Nichtkonvergenz bedeutet (außer beim ersten Teilintervall), daß erhöhte Rechenleistung anfällt. Dies wird im allgemeinen dann der Fall sein, wenn sich der Krümmungsradius der Kurve stark verkleinert. Diese lokale Information kann verwendet werden, um den Einsatz weiterer CPUs (Tasks) zu initiieren. Im Falle 2) wäre eine Lastabgabe eigentlich nur dann sinnvoll, wenn dem "lastabgabewilligen" Prozeß "bekannt" ist, daß Prozesse mit BW existieren. Die Information "Existenz von BW" ist bei einer Local-Memory-Architektur nur mit Hilfe des Message Passing zu erhalten. Das bedeutet bei einer SUPRENUM-artigen Architektur, daß der sequentielle Anteil im Gesamタルgorithmus steigt, denn es muß eine Synchronisierung zwischen einer die TRAPEX-Tasks überwachenden Task (im folgenden MASTERTRAPEX) und der "lastabgabewilligen" TRAPEX-Task erfolgen, die so lange die Arbeit unterbrechen muß, bis der MASTERTRAPEX geantwortet hat. Dieses Unterbrechen ist sowohl statisch (abzulesen am Implementierungs-Code: zweimalig SEND/RECEIVE Sequenz) wie auch dynamisch (Vergleich von graphischen Anzeigen) erkennbar. Das asynchrone Message Passing kann nicht verwendet werden, denn wenn die TRAPEX-Task eine asynchrone Meldung über die Existenz eines BW-Prozesses zu einem anderen (späteren) Zeitpunkt aufnehmen würde, könnte diese Information schon obsolet sein, weil der ehemals wartende Prozeß ein neues SI erhalten hat!

Im Fall 1) muß eine Zwischenspeicherung der Information über die Lage und Größe des abgegebenen Restintervalles erfolgen

Die Verwaltung der Restintervalle obliegt dem MASTERTRAPEX-Prozeß. Dies ist ein Prozeß, der aufgrund der Local Memory- und Bus- Architektur benötigt wird. Im logischen Sinne ist er ein Monitor wie ihn C.A. Hoare definiert hat [8,9]. Dieser MASTERTRAPEX führt Buch über die Lastverteilung im System der aktiven TRAPEX-Prozesse (auch Knoten genannt) und kennt die Zahl der verfügbaren aber noch nicht belegten CPUs und der aktivierten Prozesse. Ein TRAPEX-Knoten wird BW (oder frei), wenn er ein Subintervall berechnet und dem MASTERTRAPEX den Näherungswert für dieses Subintervall abgeliefert hat. Der MASTERTRAPEX weist einer freien (oder freigewordenen) CPU ein Subintervall, das auf einem Stack liegt, zur Berechnung zu. Liegen Elemente auf dem Stack und existieren noch nicht initialisierte CPUs, werden auf diesen durch den MASTERTRAPEX TRAPEX-Prozesse initialisiert und aktiviert. Einmal aktivierte Prozesse (und damit CPUs) bleiben bis zum Ende der Gesamtberechnung (Termination aller aktiven CPUs erst nachdem der MASTERTRAPEX den Näherungswert an den Host-Knoten abgeliefert hat) zugeteilt. Prinzipiell könnten TRAPEX-Prozesse mit BW jederzeit durch den MASTERTRAPEX zur Termination veranlaßt werden, wenn keine SI auf dem Stack des MASTERTRAPEX liegen. Bei vorzeitiger Termination müßten aber CPUs, die schon einmal initi-

alisiert waren, reinitialisiert werden. Dies ist zeitlich aufwendig. Beobachtungen ergeben, daß die Zahl der Prozesse mit BW-Status gegen Ende der Simulation ansteigt. Alternativ w re denkbar, da  ab der Terminationsgrenze T_g :

$$T_g = \frac{C_{\max} - P_a}{C_{\max}}$$

C_{\max} = maximale Zahl von CPUs, die dem
mit MASTERTRAPEX zur Verfügung steht

$$P_a = \text{Anzahl aktiver Prozesse}$$

Prozesse terminieren und CPUs damit für andere Benutzer frei gegeben werden können. Welchen Wert TG haben sollte, ist aber eine rein heuristische Betrachtung und müsste für eine größere Beispiel-Menge experimentell ermittelt werden.

Mit diesem Verfahren kann eine gleichmäßige Verteilung der (problemabhängigen) Last erreicht werden. Es wird damit die CPU-Leistung problemabhängig dann erhöht, wenn Kurventeile mit kleineren Krümmungsradien zur Berechnung anstehen. Hier wird außer der statischen auch eine dynamische (zur Laufzeit) Adaptivität erreicht. Damit ist der parallele TRAPEX-Algorithmus Hardware- und Problem-adaptiv.

2.2 Prozessoranzahl und erreichbare Beschleunigung

Bei der Untersuchung des Zeitverhaltens wurden zwei Modi untersucht:

- a) saturierter Modus (vergl. auch [13])
 - b) Festgrenzen-Modus

Unter saturiertem Modus wird verstanden, daß der Algorithmus mit n CPUs 1 $\leq n \leq C_s(n)$ beginnt und problemabhängig weitere CPUs aktivieren kann. $C_s(n)$ sei dabei so groß, daß kein Restintervall, das ein TRAPEX-Knoten abgeben will, zwischengespeichert werden muß.

Unter Festgrenzen-Modus wird verstanden, daß der Algorithmus mit n CPUs beginnt und weitere CPUs bis zu einer Höchstzahl m aktivieren kann, so daß gilt: $n \leq m \leq C_{f_{\max}}$ (bei der realen SUPRENUM1-Maschine wird $C_{f_{\max}}$ höchstens gleich 256 sein). n und m werden entsprechend dem zu rechnenden Problem gewählt. Im Gegensatz zum saturierten Modus werden Restintervalle — dann notwendigerweise — zwischengespeichert, wenn keine Prozesse im BW-Zustand existieren, bis eine TRAPEX-Task in den BW-Status geht und neue "Last" aufnehmen kann. Für $n \leq C_{f_{\max}}$ ist $C_{f_{\max}}$ i.a. wesentlich kleiner als $C_s(n)$.

Bei Amdahl muß vorausgesetzt werden, daß die Zahl der ausgeführten Anweisungen in den parallelisierbaren Teilen des Algorithmus sich nicht ändert, wenn man von sequentieller zu paralleler Verarbeitung übergeht. Dies ist bei dem von uns untersuchten Problem nur für die horizontale Parallelisierung der Fall. Bei ihr ändert sich die Zahl der auszuführenden Anweisungen zur Berechnung der Funktionswerte an den Stützstellen nicht, wenn diese Berechnung parallel erfolgt. Da dieser parallelisierbare prozentuale Anteil a für diesen Fall aber sehr klein ist (weil SUSI als Grundlage zur Ermittlung der Simulationszeit die Fortran-Anweisungen pro Sekunde nimmt [16]) und der sequentielle Anteil durch das notwendige Message Passing sogar noch vergrößert wird, wird T_p häufig größer als T_{seq} und die Beschleunigung $B_p < 1$, der parallele Algorithmus ist dann langsamer als der sequentielle.

Bei der vertikalen Parallelisierung wird das Intervall in p Teilintervalle aufgeteilt und jedem der p Prozessoren eines zur Bearbeitung gegeben. In der Regel ist bei äquidistanter Aufteilung des Intervalles keine gleichmäßige Partitionierung des Arbeitsaufwandes gegeben.

Sei A_i die für ein Intervall I auszuführende Anzahl von Instruktionen (Rechenaufwand für das ganze Intervall). Dann wird, wenn

$$TI_p = \frac{I}{p} \quad \text{das } p\text{te Teilintervall}$$

ist, die für ein solches Intervall TI_p zu berechnende Instruktionsanzahl Ai_p nur dann gleich einem

$$Ai_g = \frac{Ai}{p}$$

sein, wenn der Aufwand in jedem TI_p gleich ist, d.h. mit einer äquidistanten Aufteilung des I auch eine Gleichverteilung des Aufwandes erreicht wird.

Die Summe der im Parallel-TRAPEX durch p Prozessoren ausgeführten Anweisungen ist nicht gleich der Zahl, die für den sequentiellen Fall benötigt wird. Die Zahl der Anweisungen in TRAPEX, die ein Prozessor ausführen muß, ist bei gegebener Kurvenform von der Subintervall-Aufteilung abhängig, sowie entscheidend von dem Verhältnis von h/h_{\max} (der initialen Trapezhöhe und dem während der Berechnung zulässigen maximalen H). Man kann also in der Regel nicht davon ausgehen, daß die Summe der Ai der einzelnen "basic step" im sequentiellen Fall gleich der Summe der Ai im parallelen Fall mit p Prozessen ist. Eine ungünstige Wahl der Subintervallgrenzen, z.B. wenn diese mit Null-Stellen der Funktion oder "Nadeln" zusammenfallen, kann dazu führen, daß das parallele TRAPEX scheitert, während das sequentielle über diese Problemstellen "hinwegkommt".

Es zeigt sich, daß für jedes Beispiel b mit $1 \leq b \leq 27$ und b aus dem Testset ein $N_{\text{opt}} > 1$ existiert, so daß die Rechenzeit minimal wird. Es gibt eine optimale initiale CPU-Anzahl für jedes Beispiel aus dem Testset. Derzeit wird dieses n für jedes Testbeispiel pragmatisch ermittelt. Angestrebt wird eine automatische Berechnung der initialen Prozessorzahl. Das Problem dabei ist, daß der Aufwand zur Ermittlung der initialen CPU-Anzahl möglichst gering sein muß, damit bei "einfachen" Kurvenformen (z.B. große Krümmungsradien) dieser in jedem Falle dann sequentielle Anteil an dem Gesamtalgorithmus den Gewinn durch die vertikale Parallelisierung nicht zunichte macht. Es gibt auch Testbeispiele für die gilt: $N_{\text{opt}} < m$.

Die erreichbare Beschleunigung B auf Parallel-Architekturen [10] kann man definieren zu:

$$B_p = \frac{T_{\text{seq}}}{T_p} \quad T_{\text{seq}}, T_p \text{ Ausführungszeit ,}$$

sie ist der Quotient aus zeitlichem Aufwand T_{seq} des sequentiellen Algorithmus A_{seq} zu T_p , der von dem parallelen Algorithmus A_{par} (mit p Prozessoren) benötigte Zeit. Wenn $B < 1$ wird, sollte eine Parallelisierung vermieden werden.

Die Effizienz der Parallelisierung erhält man mit

$$E_p = \frac{B_p}{p} , \quad 0 \leq E_p \leq 1 .$$

Sinnvollerweise wird man fordern, daß E_p möglichst nahe bei 1 liegt. Es wird in der Regel eine mehr als einelementige Menge:

$$E_c = \{E_p \mid E_p \geq c , \text{ mit } 1 < p \leq \text{CPU}_{\max}\} , \quad 0 < c \leq 1$$

geben. Die Frage ist, wie man ein minimales p bestimmt, um ein maximales E_p aus E_c zu erhalten. Welcher Wert für c akzeptabel ist, hängt auch von ökonomischen Faktoren ab.

Nach Amdahl gilt:

$$B_p = \frac{1}{1 - a + \frac{a}{p}} ,$$

wobei a der Anteil in A_{par} (in %) ist, der nicht sequentiell ausgeführt werden muß, und p die Zahl der eingesetzten CPUs ist.

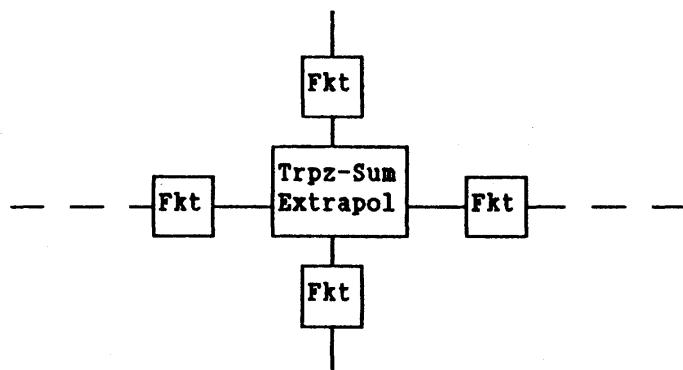
Danach gibt es eine Obergrenze der erreichbaren Beschleunigung, die nur von den Eigenschaften des Algorithmus (genauer seiner Implementierung) abhängt, nicht aber von den Eingabewerten für den Algorithmus !

E_p wäre danach:

$$E_p = \frac{1}{p(1 - a) + a} .$$

2.3 Architekturüberlegungen

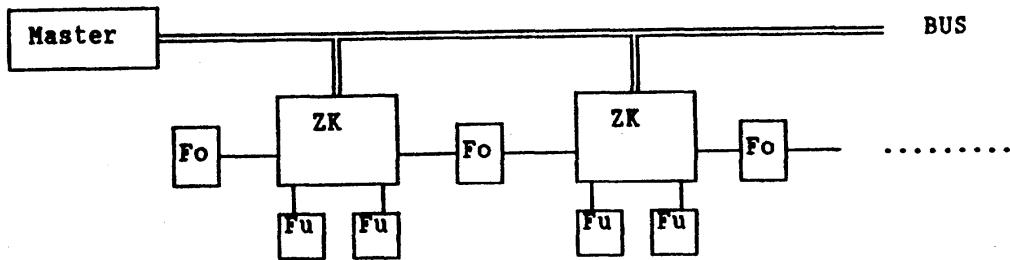
Sowohl bei der sequentiellen wie auch bei den parallelen Versionen des TRAPEX wird die Bulirsch-Folge für die Anzahl der Trapeze in den Verfeinerungsstufen verwendet. Bei ihrer Anwendung werden bis zu einer Ordnung 7 nicht mehr als 4 Funktionswerte gleichzeitig beim Übergang von der n -ten zur $(n+1)$ -ten Ordnung ($1 \leq n \leq 6$) benötigt. Es ist damit ein Quintupel von Prozessoren mit Nearest-Neighbour-Verbindungen ausreichend. Ein Prozessor bearbeitet Extrapolation, Konvergenzüberwachung und Schrittweitensteuerung und vier als Trabanten um ihn gruppierte Prozessoren berechnen die Funktionswerte an den Stützstellen.



Im Gegensatz zum SUPRENUM-Fall müßte bei einer Initiierung einer Task für eine Subintervallberechnung ein Quintupel initialisiert werden. Da die vier Trabantenprozessoren nur für die Stufen der Ordnung 6 und 7 alle gleichzeitig benötigt werden, wäre eine "zeilenweise" Überlappung (und eventueller gleichzeitiger 90 Grad Drehung des gesamten Quintupel) der Fkt-Prozessoren sinnvoll.

Nearest-Neighbour-Verbindungen sind aber für die Übertragung der Start- und Ergebniswerte zwischen TRAPEXMASTER und Quintupel-Zentralknoten nicht optimal. Dafür wäre ein Bus-Architektur besser geeignet. Eine optimale Architektur für die Integralberechnung müßte dann eine Konfiguration entsprechend [Bild 5] - eine heterogene Architektur mit Clustern, deren Prozessoren durch Nearest-Neighbour-Verbindungen kommunizieren, die Cluster aber über Bus-Verbindungen mit dem Master-Prozeß Daten austauschen - sein.

Auf Hardware zur optimalen, parallelen Berechnung der Integralwerte sollte damit folgende Konfiguration definierbar sein:



- Master** Master-Trapex (load balancing)
ZK Zentral-Knoten (Schrittweite, Konvergenz,
 Extrapolation)
Fu Knoten zur Berechnung der Funktionen der
 Ordnung 1 bis 5
Fo Knoten zur Berechnung der Funktionen der
 Ordnung 6 und 7

Man beachte, daß dies eine hybride Lösung im Bereich der Kommunikation darstellt. Die Fo-, Fu-Prozesse könnten über synchrone Nearest-Neighbour-Verbindungen mit dem ZK kommunizieren, während der Master mit den ZK über eine (preiswerte) asynchrone Bus-Verbindung Load-Balancing-Daten austauscht. Synchrone Verbindungen sind zwischen den Fo, Fu und den ZK nur sinnvoll, wenn auf den Knoten ein paralleles Mikro-Tasking zwischen E/A, Adreßrechnung und Fließkomma-Arithmetik hardwaremäßig möglich ist.

3. Meßdaten

3.1 Beschreibung und Vergleich der parallelen TRAPEX-Varianten

TparA: In der initialen Task wird das Grundintervall entsprechend der vorgegebenen CPU-Anzahl in gleich lange Teilintervalle aufgeteilt und jeder CPU nach Initialisierung derselben ein solches zur Bearbeitung übergeben. Als Initialisierung einer CPU wird hier die Zuordnung einer Task zu einer CPU im Sinne von MIMD-FORTRAN mittels NEWTASK bezeichnet. Eine derart eingerichtete Task erhält dann mittels SEND bzw. RECEIVE die Eingabeparameter zur Berechnung eines Näherungswertes, nämlich Anfangs- und Endpunkt des Teilintervall, initiale Schrittweite, eine vorgegebene relative Genauigkeit und weitere. Das Ergebnis wird der initialen Task wieder mittels SEND bzw. RECEIVE übermittelt, die übermittelnde Task wartet dann auf eine Terminationsnachricht von der initialen Task, d.h. eine Nachricht, auf die hin sie sich beendet. Für TparA ist die Terminationsnachricht eigentlich entbehrlich, weil auf einer CPU nur ein Teilintervall abgearbeitet wird; jedoch ist im allgemeinen Fall eine CPU nach Ablieferung eines Ergebnisses frei für eine andere Aufgabe, wovon bei anderen TRAPEX-Varianten auch Gebrauch gemacht wird. Es kann daher unterschieden werden zwischen einer Task im Sinne von MIMD-FORTRAN und einer Task im Sinne eines problemabhängigen Rechenprozesses.

TparB: Bei Reduktion der Schrittweite wird das Teilintervall $[t_n, t_{end}]$ an eine freie CPU abgegeben, falls $t_{end} - t_n > 2 * h_{red}$ gilt, wobei h_{red} die reduzierte Schrittweite ist, die auch als Anfangswert für das Restintervall übergeben wird, d.h., eine Abgabe erfolgt nur, falls mit dieser Schrittweite noch mindestens zwei "basic steps" ausführbar sind. Zur Task-Verwaltung wird bei dieser und den übrigen Trapex-Varianten mit Abgabe von Teilintervallen eine zusätzliche Task MASTER-TRAPEX eingerichtet; diese übernimmt abgegebene Teilintervalle und übergibt sie an freie CPUs. Falls bei der Abgabe eines Teilintervall keine CPU frei ist, wird ein solches auf einem Stack abgelegt und erst später an eine freie CPU übergeben.

Zum Testen, ob sich mit TparB gegenüber TparA tatsächlich eine Beschleunigung erzielen lässt, wurden die Beispiele des Test Set aus [5] herangezogen und jeweils mehrere CPU-Anzahlen (2, 3, 4, 5, 6, 8 und 16) für TRAPEX vorgegeben; für die Aufrufparameter h_{max} und h wurden jeweils die Standardwerte $h = h_{max} = \text{Intervall-Länge} / \text{CPU-Anzahl}$ zugrunde gelegt. Ebenso wurde zum Vergleich der anderen TRAPEX-Versionen verfahren. Bei den Beispielen 13, 14, 15, 16, 21 und 22 ergibt sich eine deutliche Verbesserung und bei den Beispielen 17, 18 und 25 ist die Version TparA günstiger; letzteres ist dadurch erklärlich, daß bei gleichmäßiger Verteilung des Rechenaufwands auf die Teilintervalle der durch den MASTER-TRAPEX bedingte zusätzliche Aufwand sich ungünstig auf die Ausführungszeit

auswirkt.

TparC: Bei dieser Version werden gegenüber TparB noch mehr Teilintervalle abgegeben, und zwar jeweils bei "next basic step". Das verbleibende Restintervall wird dazu in zwei Teile geteilt, wobei der Teilungspunkt t_{an} gemäß

$$\begin{aligned} t_{an} &= 0.5 * (t_n + t_{end}) \quad \text{falls } h \leq t_{an} - t_n \text{ und} \\ t_{an} &= t_n + h \quad \text{sonst ,} \end{aligned}$$

bestimmt ist, d.h. das Restintervall wird halbiert unter Berücksichtigung der zuletzt ermittelten Schrittweite h ; abgegeben wird die zweite Hälfte mit h als Anfangsschrittweite. Hierbei stellt sich das Problem, wie die CPU-Zuordnung geregelt werden soll, falls bei der Abgabe keine CPU frei ist. Es kann analog zu der Abgabe im Falle der Schrittweitenreduktion verfahren oder aber auf die Abgabe ganz verzichtet werden; im zweiten Falle müssen zwischen der betreffenden TRAPEX-Task und dem MASTERTRAPEX Nachrichten ausgetauscht werden, ob ein Teilintervall abgegeben werden kann oder nicht. Eine weitere Möglichkeit besteht darin, die Abgabe durch den MASTERTRAPEX in Abhängigkeit vom Stand der Abarbeitung der TRAPEX-Tasks regeln zu lassen. Auch in diesem Falle müssen zwischen MASTERTRAPEX und TRAPEX-Task diesbezügliche Nachrichten übermittelt werden. Eine solche TRAPEX-Variante wurde auch implementiert und getestet; sie hat sich jedoch für die meisten Beispiele des Test-Set als ungünstiger herausgestellt als die Variante TparC ohne Regelung.

In TparC wird ein Teilintervall bei "next basic step" nur dann abgegeben, falls eine CPU frei ist. Außerdem wird eine Teilintervallabgabe erst gar nicht versucht, falls das Restintervall mit höchsten zwei "basic steps" der zuletzt geschätzten Schrittweite abgearbeitet werden kann; dies dient als Ersatz für eine Regelung durch den MASTERTRAPEX.

Im Vergleich zu TparB ergibt sich für TparC eine Verringerung der Simulationszeiten für die Beispiele 14, 15, 16, 17, 18, 21, 22 bei 2 CPUs, für die Beispiele 2, 3, 14, 22, 23 bei 3 CPUs, für die Beispiele 9, 14, 15, 16 bei 4 CPUs, für die Beispiele 6, 14, 15, 16, 22, 25 bei 5 CPUs, für die Beispiele 14, 15, 16, 17, 22, 24, 25 bei 6 CPUs, für die Beispiele 15, 16, 17, 21, 22, 24, 25 bei 8 CPUs und für die Beispiele 6, 14, 15, 16, 21 und 22 bei 16 CPUs. Vereinzelt erhöht sich auch die Simulationszeit, so für das Beispiel 2 bei 2 CPUs, für die Beispiele 3, 7, 19 und 21 bei 5 CPUs und für das Beispiel 13 bei 3, 5 und 6 CPUs. In den übrigen Fällen ergibt sich kein oder nur ein geringer Unterschied bezüglich der Simulationszeiten.

TparD: Diese Variante unterscheidet sich von TparC dadurch, daß bei "next basic step" auch dann ein Teilintervall abgegeben wird, falls gerade keine CPU frei ist. Solche Teilintervalle werden vom MASTERTRAPEX auf dem Stack

abgelegt; somit ist keine zusätzliche Kommunikation zwischen MASTERTRAPEX und TRAPEX-Task erforderlich, aber ein etwas höherer Verwaltungsaufwand. Der Vergleich mit TparC gibt kein einheitliches Ergebnis und allgemein nur geringe Unterschiede bei den Simulationszeiten, die bei höheren CPU-Zahlen (8, 16) nahezu übereinstimmen.

TparE, TparF: Schließlich wurde überprüft, ob die Bedingung

$$q = \frac{t_{\text{end}} - t_n}{2} < 2$$

für die Abgabe von Teilintervallen bei "next basic step" entbehrlich ist, d.h. bei diesen Varianten wird immer ein Restintervall abgegeben, sofern noch vorhanden. Dieses wird ebenso bestimmt wie bei TparC; es wurden jeweils Varianten mit Stack-Verwaltung (TparE) und Nachrichtenaustausch (TparF) getestet. Die Unterschiede bei den Simulationszeiten erweisen sich gegenüber den Varianten TparC bzw. TparD als recht gering; bei höheren CPU-Zahlen (8, 16) ergibt sich eine leichte Verbesserung. Auch die Varianten TparE und TparF wurden noch miteinander verglichen, was aber wieder kein einheitliches Ergebnis liefert; bei höheren CPU-Zahlen (8, 16) gibt es kaum noch unterschiedliche Simulationszeiten.

3.2 Betrachtung einiger ausgewählter Beispiele

Von Interesse ist hierbei, welcher Zeitgewinn sich durch Erhöhung der CPU-Anzahl ergibt und wie dieser im Vergleich zum sequentiellen TRAPEX ausfällt, wobei zu berücksichtigen ist, da auch im sequentiellen Fall eine Optimierung durch geeignete Wahl der Parameter h_{\max} und h möglich ist. Für die folgenden Beispiele sind jeweils die Simulationszeiten angegeben für das sequentielle Trapex mit der Vorgabe $h_{\max} = h = \text{Intervall-Länge}$ und bei optimaler Wahl der Schrittweite h sowie für die parallelen Varianten TparA, TparB, TparC, TparD und TparE. Unter ACC ist die für die jeweilige CPU-Anzahl erreichte Beschleunigung vermerkt bezogen auf den sequentiellen Fall $h = h_{\max} = \text{Intervall-Länge}$ und unter EFF die zugehörige Effizienz, die definiert als ACC / CPU-Anzahl. In den Tabellen ist als CPU-Anzahl die Anzahl der durch TRAPEX-Tasks belegten CPUs angegeben; zur Bestimmung der Effizienz wurde diese außer bei TparA um eins erhöht, weil die durch den MASTERTRAPEX belegte CPU mit zu berücksichtigen ist.

Beispiel 20

CPU	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	0.42					
SEQmin	0.27 für $h = 1/3 = 1/6 * \text{Intervall-Länge}$					
2	0.245	0.204	0.205	0.205	0.205	0.205
3	0.193	0.199	0.199	0.199	0.177	0.178
4	0.121	0.128	0.128	0.128	0.125	0.125
5	0.105	0.111	0.111	0.111	0.109	0.109
6	0.059	0.063	0.063	0.063	0.063	0.063
7	0.059	0.0645	0.065	0.065	0.063	0.063
8	0.061	0.0666	0.0668	0.0668	0.065	0.065
9	0.059	0.0669	0.067	0.067	0.063	0.063
10	0.057	0.062	0.062	0.062	0.061	0.061
12	0.061	0.065	0.065	0.065	0.065	0.065
14	0.065	0.071	0.071	0.071	0.069	0.069
16	0.067	0.071	0.072	0.072	0.071	0.071
24	0.076	0.086	0.086	0.086	0.08	0.08
32	0.098	0.108	0.109	0.109	0.127	0.127
63	0.218	0.263	0.303	0.304	0.39	0.39

TparA

CPU	2	3	4	5	6	7	8
ACC	1.71	2.18	3.47	4.0	7.12	7.12	6.88
EFF	0.855	0.73	0.87	0.8	(1.19)	(1.02)	0.86

9	10	12	14	16	24	32	63
7.12	7.34	6.885	6.46	6.23	5.53	4.29	1.95
0.79	0.73	0.57	0.46	0.39	0.22	0.13	0.03

Für dieses Beispiel gibt es eine optimale CPU-Anzahl (= 6), bei der die Simulationszeiten bei allen TRAPEX-Varianten minimal sind und nahezu übereinstimmen. Dies liegt daran, daß auch im sequentiellen Fall für $h = 1/6 * \text{Intervall-Länge}$ die minimale Ausführungszeit erreicht wird. Eine genauere Untersuchung ergab, daß im sequentiellen Fall für $h = 1/6 * \text{Intervall-Länge}$ und $h = 1/7 * \text{Intervall-Länge}$ jeweils sieben "basic steps" ausgeführt werden, wobei die Schrittweiten sich nur wenig gegenüber dem Anfangswert ändern. Gegenüber dem sequentiellen Standardfall ergibt sich mit sechs CPUs eine Effizienz von 1.19, gegenüber dem optimalen sequentiellen Fall 0.76. In den Effizienzwert geht daher nicht nur die Effizienz der parallelen Ausführung ein, sondern auch der durch die Parallelisierung bedingte zusätzliche Gewinn durch Verringerung des Rechenaufwandes. Die ermittelten Effizienzwerte können zur Festlegung einer maximalen CPU-Anzahl herangezogen werden, etwa durch die Forderung, daß ein bestimmter Wert nicht unterschritten werden soll. Das Maximum über alle Beispiele des Test-Set kann dann als Empfehlung für die im Rahmen von SUPRENUM für TRAPEX bereitzustellende CPU-Anzahl gelten, soweit diese sich nicht schon aus allgemeinen Betriebsbedingungen ergibt. Das Beispiel zeigt aber auch, daß die maximal verfügbare oder durch Effizienzgrenzen festgelegte CPU-Anzahl nicht notwendig die optimale ist. Zur näherungsweisen Bestimmung derselben erscheint es naheliegend, die durch den Benutzer angegebenen Eingabeparameter h und h_{\max} heranzuziehen, etwa gemäß

$$\begin{aligned} \text{CPU-Anzahl} &= \text{Intervall-Länge}/h_{\max} \\ \text{oder } \text{CPU-Anzahl} &= \text{Intervall-Länge}/h \\ \text{oder } \text{CPU-Anzahl} &= h_{\max}/h \text{ falls } h_{\max} < \text{Intervall-Länge}. \end{aligned}$$

Beispiel 21

CPUs	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	2.18					
SEQmin	2.04 für $h = 1/h = * h_{\max}$, h_{\max} = Intervall-Länge					
2	1.19	1.29	1.36	1.46	1.34	1.35
3	1.74	1.07	0.90	0.846	0.856	0.924
4	1.05	0.778	0.831	0.824	0.737	0.765
5	(0.946)	(0.818)	(0.899)	(0.822)	(0.844)	(0.835)
6	0.908	0.593	0.56	0.56	0.536	0.539
7	0.916	0.545	0.486	0.475	0.443	0.5
8	0.922	0.58	0.47	0.476	0.453	0.454
9	0.83	0.622	0.48	0.473	0.459	0.525
10	(0.52)	(0.49)	(0.384)	(0.418)	(0.402)	(0.393)
12	0.885	0.497	0.362	0.355	0.364	0.376
14	0.891	0.475	0.442	0.435	0.428	(0.436)
16	0.833	0.463	0.386	0.384	0.341	0.348
24	0.719	0.53	0.461	0.458	0.367	0.371
32	0.737	0.537	0.386	0.383	0.336	0.356
63	0.712	0.55	0.405	0.395	0.46	0.495

TparE

CPUs	2	3	4	5	6	7	8
ACC	1.63	2.55	2.96	2.58	4.07	4.92	4.81
EFF	0.54	0.64	0.59	0.43	0.58	0.615	0.53

9	10	12	14	16	24	32	63
4.75	5.42	5.99	5.09	6.39	5.94	6.49	4.74
0.475	0.49	0.46	0.34	0.38	0.24	0.2	0.07

Dieses Beispiel zeigt, daß die Erhöhung der CPU-Anzahl nicht in jedem Falle einen Geschwindigkeitsgewinn liefert. Es gibt für jede TRAPEX-Variante eine optimale CPU-Anzahl, die jedoch aus dem Vergleich mit dem sequentiellen TRAPEX nicht ersichtlich ist. Bei mehr als 16 CPUs steigt offenbar der Zeitaufwand für die Kommunikation im Verhältnis zur Ausführung, wie aus den Effizienzwerten ersichtlich.

Bei diesem Beispiel stellt sich das Problem, welche CPU-Anzahl vorgegeben werden soll, in verschärfter Weise, weil sich bei fünf oder zehn CPUs alle TRAPEX-Varianten mit Fehler beenden. Ursache ist jeweils die TRAPEX-Task, die das Teilintervall $[0.6,0.8]$ erhält; diese beendete sich über den Fehlerausgang JRRED > JRMAX beim ersten "basic step". Das gleiche geschieht beim sequentiellen TRAPEX bei Vorgabe der Integrationsgrenzen $[0.6,0.8]$ und der Schrittweite $h = h_{\max} = 0.2$, d.h. die Schrittweite muß öfter reduziert werden als in TRAPEX vorgesehen. Im sequentiellen Fall kann dies durch geeignete Wahl der Schrittweite durch den Benutzer vermieden werden, z.B. ist $h = 0.1$ geeignet. Allerdings liefert das sequentielle TRAPEX über das Gesamtintervall $[0,1]$ auch für die Schrittweite $h = h_{\max} = 0.2$ das richtige Ergebnis, und zwar offensichtlich deshalb, weil sich TRAPEX - jedenfalls bei diesem h -Wert — derart an die kritische Stelle anpaßt, daß über diese hinweg integriert wird. Dies bedeutet für das parallele TRAPEX, daß die Bestimmung der CPU-Anzahl aus den Eingabeparametern h , h_{\max} und Intervall-Länge nicht hinreichend für die fehlerfreie Beendigung ist. Als Abhilfe bietet sich an, auch die CPU-Anzahl als Eingabeparameter anzusetzen; es sollte aber auch ein Standardwert möglich sein, etwa ein durch das Betriebssystem vorgegebener. Eine weitere Abhilfemöglichkeit besteht darin, die Grenze JRMAX für die Anzahl der Reduktionen heraufzusetzen. Für JRMAX = 12 an Stelle von JRMAX = 5 liefern alle parallelen TRAPEX-Versionen das richtige Ergebnis. Die Begrenzung JRMAX = 5 dient offenbar dem Zweck, den Rechenaufwand bis zur Feststellung, daß keine Konvergenz durch Reduktion der Schrittweite zu erreichen ist, zu begrenzen. Dieser wird also für das parallele Trapex höher angesetzt, verteilt sich aber auch auf mehrere CPUs.

Beispiel 9

CPUs	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	1.13					
SEQmin	1.13 für $h = h_{\max}$ = Intervall-Länge					
2	0.625	0.773	0.765	0.82	0.886	0.784
3	0.47	0.50	0.51	0.513	0.539	0.506
4	0.41	0.452	0.415	0.425	0.422	0.454
5	0.276	0.399	0.341	0.353	0.361	0.348
6	0.281	0.296	0.29	0.282	0.331	0.304
7	0.243	0.285	0.261	0.246	0.251	0.258
8	0.234	0.241	0.253	0.276	0.261	0.234
9	0.219	0.232	0.204	0.204	0.234	0.207
10	0.198	0.242	0.207	0.206	0.21	0.202
12	0.2	0.196	0.195	0.183	0.183	0.18
14	0.196	0.187	0.179	0.181	0.176	0.172
16	0.172	0.178	0.18	0.182	0.172	0.165
24	0.163	0.143	0.143	0.143	0.127	0.133
32	0.16	0.163	0.163	0.163	0.156	0.157
63	0.223	0.269	0.303	0.303	0.394	0.394

TparB

CPUs	2	3	4	5	6	7	8
ACC	1.8	2.4	2.76	4.11	4.02	4.65	4.83
EFF	0.6	0.6	0.55	0.685	0.57	0.58	0.54

9	10	12	14	16	24	32	63
5.16	5.71	5.65	5.77	6.57	6.93	7.06	5.07
0.52	0.52	0.43	0.38	0.39	0.28	0.21	0.08

Für dieses Beispiel ist TparA schon die optimale TRAPEX-Variante, weil auf den initialen Teilintervallen der Rechenaufwand annähernd übereinstimmt. Bei fünf CPUs liefern alle Varianten dasselbe (falsche) Ergebnis 1.0, ohne jedoch einen Fehler anzuzeigen, ebenso das sequentielle TRAPEX für $h = h_{\max} = 1/5 * \text{Intervall-Länge}$ oder $h = 1/5 * h_{\max}$ mit $h_{\max} = \text{Intervall-Länge}$. Der Grund dafür ist, daß für die betreffende Schrittweite h die Extrapolation auf dem ersten Teilintervall nach der zweiten Zeile beendet wird, weil die Differenz zwischen der ersten und zweiten Trapez-Summe gleich null ist, was von TRAPEX als Konvergenz betrachtet wird. Wird jedoch bei den parallelen TRAPEX-Versionen nicht die Standard-Festlegung $h = \text{Intervall-Länge}/5 = 0.2$ getroffen, sondern z.B. $h = 0.1$ für jedes der fünf Teilintervalle, so erhält man das richtige Ergebnis 1.154700538379; auch das sequentielle TRAPEX liefert für $h = 0.1$ das richtige Ergebnis. Daraus folgt, da die Wahl der initialen Schrittweite durch den Benutzer als signifikant zu betrachten ist und im parallelen Fall soweit wie möglich berücksichtigt werden sollte.

Beispiel 22

CPUs	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	2.25					
SEQmin	2.25 für $h = h_{\max} = \text{Intervall-Länge}$					
2	1.59	1.20	1.14	1.19	1.2	1.15
3	1.58	1.42	0.971	0.94	0.978	1.08
4	1.28	0.848	0.801	0.705	0.638	0.72
5	1.50	1.18	0.72	0.672	0.631	0.625
6	1.12	0.578	0.567	0.568	0.563	0.557
7	1.31	1.08	0.6	0.531	0.545	0.608
8	0.882	0.654	0.475	0.475	0.423	0.454
9	1.14	0.892	0.532	0.456	0.459	0.495
10	1.01	0.6	0.448	0.484	0.404	0.433
12	0.785	0.582	0.41	0.412	0.439	0.421
14	0.828	0.492	0.34	0.35	0.381	0.414
16	0.748	0.53	0.349	0.348	0.32	0.325
24	0.721	0.444	0.324	0.323	0.269	0.275
32	0.597	0.477	0.322	0.321	0.288	0.289
63	0.672	0.473	0.388	0.366	0.418	0.424

TparB

CPUs	2	3	4	5	6	7	8
ACC	1.42	1.58	1.76	1.91	3.89	2.08	3.44
EFF	0.47	0.4	0.35	0.31	0.56	0.26	0.38

9	10	12	14	16	24	32	63
2.52	3.75	3.87	4.57	4.25	5.07	4.72	4.76
0.25	0.34	0.3	0.3	0.25	0.2	0.14	0.07

TparB

CPU	2	3	4	5	6	7	8
ACC	1.88	2.3	3.53	3.57	4.0	4.13	5.32
EFF	0.63	0.58	0.71	0.59	0.57	0.52	0.59

9	10	12	14	16	24	32	63
4.9	5.57	5.13	5.91	7.03	8.36	7.81	5.38
0.49	0.51	0.39	0.39	0.41	0.33	0.24	0.08

Bei diesem Beispiel wurde für die Version TparB auch der Einfluß verschiedener Anfangsschrittweiten auf die Simulationszeiten untersucht; dazu wurde jeweils die CPU-Anzahl vorgegeben und $h_{\max} = \text{Teilintervall-Länge} = 2.0/\text{CPU-Anzahl}$ gesetzt. Jede TRAPEX-Task erhielt dieselbe initiale Schrittweite, diese aber wurde variiert. Es zeigte sich, daß die Wahl $h = h_{\max}$ nur für zwei oder sechs CPUs optimal ist, ansonsten aber eher $h = h_{\max}/2$.

Ergebnisse ($h_{\max} = 2.0 / \text{CPU-Anzahl}$)

CPU	h_{\max}	$h_{\max}/2$	$h_{\max}/4$	$h_{\max}/8$	$h_{\max}/16$	$h_{\max}/32$
SEQ	2.25	2.68	2.45	3.06	3.05	2.91
2	1.20	1.36	1.28	1.21	1.29	1.29
3	1.42	0.92	1.48	1.41	1.39	1.42
4	0.85	0.8	0.79	0.72	0.73	0.8
5	1.18	0.72	1.18	1.28	1.19	1.25
6	0.58	0.62	0.66	0.65	0.67	0.75
7	1.08	0.62	0.99	1.14	1.06	1.09
8	0.65	0.51	0.63	0.66	0.69	0.72
10	0.6	0.53	0.64	0.54	0.61	0.61
16	0.53	0.48	0.54	0.65	0.65	0.59

Beispiel 13

CPUs	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	1.06					
SEQmin	1.06 für $h = h_{\max} = \text{Intervall-Länge}$					
2	1.19	0.668	0.702	0.719	0.727	0.718
3	1.12	0.473	0.58	0.551	0.498	0.592
4	0.78	0.77	0.67	0.702	0.415	0.609
5	1.10	0.438	0.452	0.409	0.362	0.442
6	1.01	0.382	0.52	0.381	0.435	0.482
7	0.991	0.364	0.451	0.443	0.452	0.458
8	0.65	0.66	0.576	0.572	0.258	0.264
9	1.07	0.379	0.404	0.398	0.358	0.369
10	0.921	0.315	0.389	0.379	0.38	0.39
12	0.67	0.64	0.51	0.509	0.339	0.344
14	0.935	0.50	0.42	0.413	0.416	0.426
16	0.627	0.65	0.56	0.558	0.386	0.395
24	0.602	0.63	0.53	0.527	0.38	0.387
32	0.588	0.62	0.57	0.564	0.407	0.421
63	0.914	0.609	0.511	0.504	0.635	0.593

TparB

CPUs	2	3	4	5	6	7	8
ACC	1.66	2.34	1.44	2.53	2.9	3.04	1.68
EFF	0.55	0.59	0.29	0.42	0.41	0.38	0.19

9	10	12	14	16	24	32	63
1.0	3.51	1.73	2.21	1.7	1.76	1.78	1.6
0.1	0.32	0.13	0.15	0.1	0.07	0.05	0.03

TparE

CPUs	2	3	4	5	6	7	8
ACC	1.52	2.22	2.57	3.06	2.54	2.45	4.29
EFF	0.51	0.56	0.53	0.51	0.36	0.31	0.48

9	10	12	14	16	24	32	63
3.09	2.91	3.26	2.67	2.87	2.91	2.72	1.74
0.31	0.26	0.25	0.18	0.18	0.12	0.08	0.02

Bei diesem Beispiel gibt es ungünstige CPU-Zahlen (4, 8, 12), für die die Simulationszeit zunimmt im Vergleich zur nächst kleineren, allerdings auch in Abhängigkeit von der TRAPEX-Variante. Maßgeblich ist dafür offenbar die Lage der "Spitze" bezüglich der durch die CPU-Anzahl gegebenen initialen Intervallaufteilung; falls diese am linken Rand eines Teilintervall liegt, wird die Schrittweite öfter reduziert als sonst. Ebenso wie für Beispiel 21 mußte auch hier die Reduktionsgrenze JRMAX erhöht werden, damit in den kritischen Fällen überhaupt ein Ergebnis geliefert wird.

Beispiel 17

CPUs	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	4.365					
SEQmin	4.22 für $h = h_{\max} = \text{Intervall-Länge} / 6$					
2	2.36	3.24	2.78	2.95	3.04	2.79
3	1.73	2.38	2.07	2.13	2.17	2.03
4	1.39	1.62	1.73	1.77	1.75	1.67
5	1.28	1.38	1.42	1.44	1.44	1.42
6	0.98	1.47	1.20	1.20	1.23	1.13
7	0.885	1.305	0.90	0.977	0.968	0.961
8	0.966	1.205	0.829	0.825	0.798	0.78
9	0.707	0.966	0.89	0.884	0.866	0.839
10	0.864	0.967	0.687	0.699	0.716	0.689
12	0.668	0.743	0.663	0.592	0.608	0.613
14	0.551	0.632	0.652	0.639	0.666	0.622
16	0.469	0.553	0.533	0.55	0.55	0.505
24	0.411	0.386	0.431	0.393	0.366	0.393
32	0.315	0.393	0.371	0.3734	0.338	0.334
63	0.24	0.301	0.323	0.319	0.271	0.272

TparB

CPUs	2	3	4	5	6	7	8
ACC	1.43	2.01	2.49	3.03	3.55	4.5	5.46
EFF	0.55	0.59	0.29	0.42	0.41	0.38	0.19

9	10	12	14	16	24	32	63
5.04	6.1	7.18	6.55	7.94	11.93	12.91	16.11
0.5	0.55	0.55	0.44	0.47	0.48	0.39	0.25

Für dieses Beispiel ist TparA die optimale Variante. Alle Varianten liefern mit fünf CPUs ein falsches Ergebnis; es liegt offensichtlich derselbe Effekt vor wie bei Beispiel 9, da für die Schrittweite $h = 0.1$ jeweils das richtige Ergebnis geliefert

wird. Dieses Beispiel legt es nahe, nicht die Teilintervall-Länge als initiale Schrittweite zu nehmen, sondern einen kleineren Wert. Solche Testläufe wurden mit TparA und TparE durchgeführt mit $h = l$, $h = l/2$, $h = l/4$, $h = l/8$, ... $h = l/128$, $l = \text{Intervall-Länge} / \text{CPU-Anzahl}$; die Ergebnisse für TparE sind in der folgenden Tabelle zusammengefaßt, in der jeweils die maximalen (ts_{\max}) und minimalen (ts_{\min}) Simulationszeiten mit den zugehörigen Schrittweiten (h_{\max} bzw. h_{\min}) sowie die Simulationszeit für den Standardfall $h = l/(ts)$ angegeben sind.

CPUs	ts	ts	h_{\max}	ts_{\min}	h_{\min}
2	3.04	3.55	1/16	2.59	1/64
3	2.17	2.28	1/64	1.76	1/16
4	1.75	1.88	1/8	1.33	1/16
5	1.44	1.44	1/2	1.03	1/128
6	1.23	1.23	1	0.9	1/8
7	0.97	1.11	1/128	0.81	1/64
8	0.8	1.02	1/32	0.75	1/8
10	0.72	0.8	1/2	0.62	1/8
16	0.55	0.57	1/16	0.44	1/8
32	0.34	0.45	1/8	0.27	1/4

Wie aus der Tabelle ersichtlich, kann die Ausführungszeit durch optimale Wahl der initialen Schrittweite noch deutlich verbessert werden, wobei der Geschwindigkeitsgewinn in derselben Größenordnung liegt wie bei der Erhöhung der CPU-Anzahl um eins. Bei TparA hat die Variation der initialen Schrittweite keinen Einfluß auf die Simulationszeiten; bei vorgegebener CPU-Anzahl werden jeweils gleich viele "basic steps" durchgeführt und auch gleich viele Funktionsauswertungen. Im sequentiellen Falle ergeben sich bei entsprechenden Testläufen nur geringe Unterschiede bei den Simulationszeiten; offensichtlich greift die initiale Aufteilung des Grundintervalls bei diesem Beispiel nicht so stark in die sequentielle Schrittweitensteuerung über dasselbe ein wie bei den anderen.

Beispiel 1

CPUs	TparA	TparB	TparC	TparD	TparE	TparF
SEQ	0.039					
SEQmin	0.039 für $h = h_{\max} / 4$, h = Intervall-Länge					
2	0.024	0.0275	0.028	0.028	0.029	0.029
3	0.026	0.029	0.029	0.029	0.028	0.028
4	0.024	0.027	0.027	0.027	0.035	0.037
5	0.025	0.029	0.029	0.029	0.029	0.027
6	0.027	0.031	0.031	0.031	0.031	0.031
8	0.031	0.036	0.035	0.035	0.035	0.035
16	0.045	0.051	0.051	0.051	0.052	0.052

Für dieses Beispiel ist TparA optimal mit der CPU-Anzahl 2, weil nur zwei "basic steps" ausgeführt zu werden brauchen, wie ein Vergleichslauf mit dem sequentiellen TRAPEX zeigt. Eine Erhöhung der CPU-Anzahl bedeutet hier nur zusätzlichen Rechen- und Kommunikationsaufwand. Im sequentiellen Fall wird die Schrittweite bei Start mit $h = h_{\max} = 1.0$ einmal auf $h = 0.41$ reduziert, was die etwa höhere Simulationszeit gegenüber der optimalen Wahl $h = 0.25$ erklärt.

Zum saturierten Modus: Aus der Betrachtung der Beispiele ergibt sich, daß aus Effizienzgründen eine Begrenzung der CPU-Anzahl sinnvoll erscheint, wobei allerdings das anzusetzende Maximum vom Beispiel abhängt. Außerdem zeigt sich, daß bei Verwendung der maximal vorgesehenen CPUs die Ausführungszeit nicht notwendig minimal wird, bedingt durch den Eingriff in die Schrittweitensteuerung von TRAPEX durch die Aufteilung des Grundintervalles in mehrere gleich lange Teilintervalle und durch die Festlegung, daß auf jedem Teilintervall mit der Teilintervall-Länge als initialer Schrittweite begonnen wird. Einige Beispiele legen es nahe, zur Festlegung der CPU-Anzahl die Eingabeparameter $t_{end} - t =$ Intervall-Länge und h_{\max} bzw. h zu verwenden. Jedoch ergibt sich daraus nicht in jedem Falle die optimale CPU-Anzahl und im Falle $h_{\max} = h =$ Intervall-Länge keinerlei diesbezügliche Information. Als Lösung dieses Problems bietet sich an, bei der initialen Aufteilung des Grundintervalls in Teilintervalle nicht die maximal mögliche CPU-Anzahl zu verwenden, sondern nur so viele, wie aus den Eingabeparametern h_{\max} bzw. h bestimmbar und während der Programmausführung an kritischen Stellen weitere CPUs heranzuziehen.

Solch eine kritische Stelle ist jedenfalls die Reduktion der Schrittweite, falls noch keine Konvergenz vorliegt oder die Schrittweite in Bezug auf den zu erwartenden Aufwand bis zur Konvergenz als zu groß erscheint (Konvergenzmonitor). Analog

zur Abgabe eines Teilintervall es an eine freie CPU kann noch eine weitere CPU belegt werden, falls vorhanden und alle übrigen CPUs gerade beschäftigt sind. Eine einmal zusätzlich belegte CPU nimmt dann bis zum Programmende an der Lastverteilung teil. Eine weitere Stelle, an der eine CPU belegt werden kann, ist die Abgabe eines Teilintervall es nach einem "basic step". Offen bleibt bei diesem Verfahren aber immer noch die Festlegung einer maximalen CPU-Anzahl für das parallele TRAPEX, soweit diese nicht als SUPRENUM-Betriebs-Parameter anzusehen ist. Naheliegend ist es, nicht von vorneherein ein Maximum anzusetzen, sondern so viele CPUs zuzulassen, wie während der Programmausführung angefordert werden; dies nennen wir den "saturierten Modus". In Hinblick auf die Minimierung der Ausführungszeit stellt sich dann die Frage, ob sich tatsächlich eine Verbesserung durch die Einführung von initialer und maximaler CPU-Anzahl ergibt und ob der saturierte Modus günstiger ist als der entsprechende "Festgrenzenmodus", bei dem die initiale und maximale CPU-Anzahl gleich der sich im saturierten Modus ergebenden maximalen CPU-Anzahl ist. Wegen des Effizienzverlustes bei Erhöhung der CPU-Anzahl stellt sich für den saturierten Modus auch die Frage, ob die erreichte maximale CPU-Anzahl eine minimale Ausführungszeit ergibt bezogen auf die zu Grunde gelegte initiale CPU-Anzahl, d.h. ob nicht doch eine Herabsetzung der maximalen CPU-Anzahl auch im saturierten Modus günstiger ist. Es fehlt allerdings ein Kriterium für diese Begrenzung. Entscheidend für die Minimierung der Ausführungszeit ist aber im saturierten Modus die initiale CPU-Anzahl, wie aus Testläufen mit den Beispielen hervorgeht. Damit stellt sich das Problem, ob die initiale CPU-Anzahl doch anders ermittelt werden sollte als aus den Eingabeparametern h_{\max} , h und Intervall-Länge, etwa aus vorweg zu beschaffenden Informationen über den Graphen der zu integrierenden Funktion.

Ergebnisse: Bei TparE ergibt der saturierte Modus keinen Geschwindigkeitsvorteil für die Beispiele 1, 2, 4, 6, 7, 8, 9, 10, 11, 12, 17, 18, 20, 23, und 25, bei initial ein oder zwei CPUs sogar eine Verschlechterung. Für Beispiel 14 ergibt sich (auch für $CPU_{\text{init}} = 1$) eine Verbesserung, ebenso für Beispiel 16. Bei den übrigen Beispielen ist das Ergebnis je nach CPU-Anzahl unterschiedlich. In TparE kommt eine weitere CPU bei Bedarf jeweils bei der Schrittweitenreduktion und bei der Abgabe eines Teilintervall es am Ende eines "basic step" hinzu. Im übrigen zeigt sich, daß neu hinzukommende CPUs zwar aktuell zur Entlastung beitragen, später aber nicht mehr ausgelastet sind. Bei allen Varianten wird eine einmal initiierte CPU während eines Laufes nicht mehr deaktiviert, in der Hoffnung alle aktivierten CPUs können bis zum Ende des TRAPEX-Laufes mit Last versorgt werden. Hier wäre die Möglichkeit mit der Hilfe eines Idle-Kriteriums (z.B. mehr als die Hälfte der CPUs idled) Deaktivierungen vorzunehmen.

Bei TparF ergibt der saturierte Modus keinen Geschwindigkeitsvorteil für die Beispiele 1, 2, 4, 5, 6, 8, 9, 10, 11, 12, 15 und 23; für die Beispiele 13, 21 und 24 ist der saturierte Modus günstiger als der Festgrenzenmodus; bei den übrigen

Beispielen ist er vereinzelt (Bsp. 9 mit $CPU_{init} = 4$, Bsp. 16 mit $CPU_{init} = 3$, Bsp. 17 mit $CPU_{init} = 1$, Bsp. 25 mit $CPU_{init} = 3$) günstiger. In TparF wird nur bei der Abgabe eines Teilintervall es eine weitere CPU hinzugenommen. Dadurch werden im Vergleich zu TparE im allgemeinen weniger CPUs verwendet. Im übrigen sind die Unterschiede in den Ausführungszeiten zwischen saturierterem und Festgrenzen-Modus recht gering; für 16 CPUs ergeben sich in beiden Modi außer bei den Beispielen 16 und 22 die gleichen Zeiten.

In TparG wird ebenfalls nur bei der Abgabe eines Teilintervall es im Falle der Schrittweitenreduktion eine weitere CPU hinzugenommen, jedoch entfällt der Nachrichtenaustausch zwischen MASTERTRAPEX und TRAPEX-Task bezüglich der Abgabe; das abgegebene Teilintervall wird nötigenfalls auf dem Stack abgelegt. Für die Beispiele 4, 5, 6, 8, 9, 10, 11, 12, 15, 23 und 25 ergibt sich damit jedoch keine Verbesserung gegenüber dem Festgrenzenmodus; eine deutliche Verbesserung zeigt nur das Beispiel 16 ($CPU_{init} = 2, 5, 8, 16$); für einige Beispiele gibt es eine bestimmte CPU-Anzahl, bei der eine Verbesserung auftritt (Bsp. 2 mit $CPU_{init} = 2$, Bsp. 7 mit $CPU_{init} = 1$, Bsp. 17 mit $CPU_{init} = 3$, Bsp. 19 mit $CPU_{init} = 5$, Bsp. 22 mit $CPU_{init} = 5$), der jedoch eine Verschlechterung bei den übrigen CPU-Anzahlen gegenüber steht.

Schlußbetrachtung

Bei einer SUPRENUM-artigen Architektur ist es aufgrund des durch das Message Passing über Busse entstehenden Overheads sinnvoll, eine Grobgranulierung zu verwenden. Auf den Integrationsalgorithmus bezogen bedeutet dies, daß die Partitionierung des Gesamtintervall es in n parallel zu berechnende Teilintervalle bessere Bechleunigungen ergibt, als das parallele Berechnen der Funktionswerte. Allerdings sollte die initiale Partitionierung nicht zu fein sein, denn damit werden zu Beginn sofort viele Prozesse (CPUs) gestartet, die dann im weiteren Verlauf der Berechnung des Integralwertes schwieriger im Busy-Status zu halten sind. Wie auch andere Autoren bemerken, ist eine wie auch immer geartete Lastverteilung wesentlich, um eine ausreichend hohe Effizienz zu erreichen.

Gegenwärtig implementieren wir unseren Algorithmus auf einer Nearest-Neighbour-Architektur (Transputer), und wir werden ihn auch für eine DOOM-Achitektur [14] implementieren, sobald ein Simulator für dieses Architektur verfügbar ist.

Die Autoren danken Herrn U. Nowak für seine Unterstützung und seine wertvollen Hinweise bezüglich numerischer Probleme.

Anhang A

In this part we give a brief overview of the basic algorithm for the so-called TRAPEX-Master.

Basic Algorithm for TRAPEX-Master:

- 1) receive basic data
(number of example, max. numb. of nodes, endpoints of interval,
permissible error)
- 2) determine number of nodes from input parameters, initialize nodes and trans-
fer initial parameters to TRAPEX-nodes
- 3) wait for response from any TRAPEX-node
- 4) if a message has arrived with TAG=3 (in the TRAPEX-Master-mailbox)
 - 4.1) process reduction case
 - 4.2) goto 3)
- 5) if a message has arrived with TAG=7
 - 5.1) process next basic step case
 - 5.2) goto 3)
- 6) if a message has arrived with TAG=4
 - 6.1) process potential solution case
 - 6.2) goto 3)

At this stage the algorithm seems to be infinite. But on the first refinement step we shall introduce termination conditions!

1st Refinement for the basic Algorithm for TRAPEX-Master

- 4.1.1) if all nodes are busy
 - if possible initialize a new node and send initialization data
 - otherwise put load info onto stack
 - otherwise if there is an idling node, send work (just received) to that node
- 4.1.2) if there is an idling node and there is more work determine free node, send work to it, and mark it as busy
 - 5.1.1) if there is an idling node, determine its ID, send load-data (just received) to that node, and mark that node as busy
 - 5.1.2) if all nodes are busy
 - if possible initialize a new node, send initialization data
 - otherwise put load info onto stack (Version TparE)
 - 6.1.1) Check error-flag ($h = 0$)
 - if error send message to the host and terminate all nodes (including TRAPEX-Master)
 - otherwise collect the partial solution, and mark the sending node as to be idle
 - 6.1.2) if all TRAPEX-nodes are idle and there is no load info on stack, send solution to host, and terminate all processes
 - 6.1.3) if there is labour info on stack and there is an idling node, send work to that node, mark that node as busy (now)

When the host node has received the solution or an error indication ($h = 0$) it displays this information and terminates.

Anhang B

This appendix contains code for the monitoring part MASTERTRAPEX and the parallel version of TRAPEX.

```
c ****
c task program masttrap
c ****
c mastertask for control of number of nodes and load balancing
c
c
c
c task external trapex
intrinsic dfloat,min,dint
logical busy(64)
taskid nodeid(64), org, trapx1
integer cpumax,notbusy,nochwork,i,j,nstep,tstep,free,
1      cpuanz,min,dint,bspnr,fanz,tfanz,nwmax
double precision trwork(128,3),alth,esth,aeps,err,h,tan,
1      teulta,teilte,ta,te,mh,li,y,ay,linkspkt,dfloat,scale,tend,
1      error
c ===== start of task control =====
receive(tag=1, sender=org) bspnr,cpumax,cpuanz,ta,te,aeps,
1      err,mh,h
c =====
error = 0.0d0
y = 0.d0
fanz = 0
nstep=0
scale = 1.d0
c scale = aeps
nochwork = 0
nwmax = 0
li=(te-ta)/cpuanz
print*, "Initiale CPU-Anzahl: ",cpuanz
print*, "Laenge der/des init. Intervalle(s):",li
print*, "Laenge von Hmax und H      :",mh,h
print*, "
notbusy = cpuanz
free = cpumax-cpuanz
```

```

        end if
    else
c ===== send work to free node =====
        teilta = linkspkt
        teilte = linkspkt + alth
1200    do 1000 j=1,cpuanz,1
        if (.not.busy(j)) goto 1100
1000    continue
1100    send(tag=9,taskid=nodeid(j)) teilta,teilte,
1        alth,esth,scale
        notbusy = notbusy - 1
        busy(j) = .true.
    end if
    goto 1800
c
c ****
c schedule: collect partial solution and send new labour
c ****
c
400    receive (tag=4, sender=trapx1) tstep,tfanz,ay,err,h
        if (h .eq. 0.d0) then
            send (tag=9999,taskid=nodeid)
            send (tag=5,taskid=org) cpuanz,nstep,fanz,nwmax,y,h
1        error
        stop
    end if
    y = ay + y
    error = error + err
    nstep=nstep+tstep
    fanz=fanz+tfanz
    if (dabs(y) .gt. scale ) scale = dabs(y)
    if ( nochwork .lt. 1) then
        notbusy = notbusy+1
        do 1400 j=1,cpuanz,1
            if (trapx1 .eq. nodeid(j)) then
                busy(j)=.false.
            goto 1410
        end if

1400    continue
1410    if (notbusy.ge.cpuanz) then
        send (tag=9999,taskid=nodeid)
        error = error/dabs(y)
        send (tag=5,taskid=org) cpuanz,nstep,fanz,nwmax,y,h,
1        error

```

```

        stop
    else
        goto 600
    end if
end if
do 1530 j=1,cpuanz,1
    if (nodeid(j).eq.trapx1) goto 1500
1530 continue
1500 teilta=trwork(nochwork,3)
    alth =trwork(nochwork,1)
    teilte =alth+teilta
    esth =trwork(nochwork,2)
1800 if ((notbusy.gt.0).and.(nochwork.gt.0)) then
    do 1600 j=1,cpuanz,1
        if (.not.busy(j)) goto 1700
1600 continue
1700 busy(j)=.true.
    trapx1=nodeid(j)
    notbusy=notbusy-1
    goto 1500
end if
goto 600
c
c **** schedule: work relinquished in case of next basic step ****
c ****
c
2000 receive(tag=7, sender=trapx1) tan,tend,esth
    if (notbusy.gt.0) then
        do 2030 i=1,cpuanz,1
            if (.not. busy(i) ) goto 2040
2030 continue
2040 send(tag=9,taskid=nodeid(i)) tan,tend,tend-tan,esth,
    1 scale
    notbusy=notbusy-1
    busy(i)=.true.
else
c ===== init one more node =====
    if (free .gt. 0) then
        cpuanz = cpuanz + 1
        free = free - 1
        nodeid(cpuanz) = newtask(trapex,cpuanz+1)
        send(tag=2,taskid=nodeid(cpuanz)) bspnr,tan,
    1      tend,aeps,err,tend-tan,esth,scale

```

```

    busy(cpuanz) = .true.
else
c      === stack for collecting further work ====
    nochwork = nochwork + 1
    if (nochwork .gt. nwmax) nwmax = nochwork
    trwork(nochwork,1) = tend-tan
    trwork(nochwork,2) = esth
    trwork(nochwork,3) = tan
    end if
end if
goto 600
stop
end

c
c

c *****
c      task program trapex
c *****
c
c
c      solution of quadrature problems on finite intervals
c      using trapezoidal sums with polynomial extrapolation
c      (Romberg quadrature)
c
c      local relative error control with internal scaling
c
c***** seq. Ver. last change: july 20,'82 *****
c***** par. Ver. last change : june 10, 88 *****
c
c      external f,dfloat
intrinsic dfloat
implicit integer (i-n)
double precision dfloat,alh,scale,neuh,esth
double precision tivala,tivale,tan,altt,redta,redte
taskid org
integer abspnr,afanz,jmi,kmi,diffanz
c
integer nj( 7),incr( 7),nred( 6)
double precision dt( 7),d( 7, 7),a( 7),al( 7, 7)
double precision a1,b1,c,eph,epmach,eps,err,errh,error
double precision f,fc,fcm,fco,fi,fmin,g,h,half,hmax
double precision hmaxu,hr,h1,omj,omjo,one,one1,q,red,ro
double precision safe,t,ta,ten,tend,teps,tn,u,u1
double precision v,w,y,yl,yl1,yn,yr,zero,zq,zq1

```

```

data zero/0.d0/,fmin/1.d-2/,ro/0.25d0/,half/0.5d0/
data safe/0.5d0/,one/1.d0/,one1/1.01d0/,ten/1.d1/
data dt/ 7*0.d0/

c
c stepsize sequence ( due to /1/ )
    data nj/1,2,3,4,6,8,12/
c associated normalized work per unit step
    data a/1.d0,2.d0,4.d0,6.d0,8.d0,12.d0,16.d0/
c
c

c ##### initial Data Transfer from Master-Node #####
c receive (tag=2, sender=org) absprn,t,tend,eps,error,hmax,
1      h,scale
afanz = 0
goto 10008
10005  wait(tag=2,taskid=org,label=10010),(tag=6,taskid=org,
1      label=10020),(tag=9999,taskid=org,label=10000),
1      (tag=9,taskid=org,label=10030)
c
10010  receive (tag=2, sender=org) absprn,t,tend,eps,error,hmax,
1      h,scale
afanz = 0
c
#####
c
10008  tivala=t
tivale=tend
y=0.d0
epmach=2.d-15
c
c maximum column number (1.le.km.le.6)
km=6
c associated maximum row number (2.le.jm.le.7)
jm=km+1
c standard values fixed below
c submitted number of integration steps per interval
nstmax=1000
c maximum permitted number of stepsize reductions
jrmx=12
c initial preparations
eph=ro*eps
error=zeros
omjo=zeros
a1=a(1)

```

```

joh=jmi
tivala=t
tivale=tend
nstep=0
do 10040 j=1,jm-1,1
  incr(j)=0
  nred(j)=0
10040 continue
c =====
10050 incr(jm)=-1
  hmax=dabs(hmax)
  nstep=0
  teps=(dabs(t)+dabs(tend))*epmach
  h1=tend-t
  if(dabs(h1).le.teps) then
    print*,"-----"
    print*,"t-id:",mytaskid(),"vor 15, h1<=teps",h1,teps
    print*,"Reduktions-Startintervall:",redta,redte
    print*,"aktuelles Intervall   :",t,tn
    print*,"-----"
    goto 51
  end if
  q=h1/h
  if(q.le.epmach) then
    print*,"-----"
    print*,"t-id:",mytaskid(),"vor 15, q<=epmach",q,epmach
    print*,"Reduktions-Startintervall:",redta,redte
    print*,"aktuelles Intervall   :",t,tn
    print*,"-----"
    goto 51
  end if
  hmaxu=hmax
  hr=hmax
  zq=f(abspnr,t)
  afanz=afanz+1
  yr=zero
c
15  continue
  if(q.ge.one1) goto 16
  hr=h
  h=h1
16  jred=0
  do 17 k=1,km
17  incr(k)=incr(k)+1

```

```

hmax=dabs(h1)
  if(hmaxu.lt.hmax) hmax=hmaxu
c  goto 20
  goto 18
c =====
10020 receive(tag=6, sender=org) t,tend,hmax,h,
  1           scale,koh,incr,nred
c =====
afanz = 0
y = 0.d0
nstep = 0
jred = 0
teps=(dabs(t)+dabs(tend))*epmach
h1=tend-t
hmaxu = hmax
hmax = dabs(h1)
hr = hmax
if (dabs(hmax).le.teps) then
  print*, "-----"
  print*, "t-id:",mytaskid(),"nach tag=6, hmax<=teps",hmax
  print*, "Reduktions-Startintervall:",redta,redte
  print*, "aktuelles Intervall   :",t,tn
  print*, "-----"
  goto 51
end if
q=h1/h
if(q.le.epmach) then
  print*, "-----"
  print*, "t-id:",mytaskid(),
  1 "nach tag=6, q<=epmach q,h1,tivala,tivale:",q,h1,
  2 tivala,tivale
  print*, "Reduktions-Startintervall:",redta,redte
  print*, "aktuelles Intervall   :",t,tn
  print*, "-----"
  goto 51
end if
if(q.ge.one1) goto 10026
hr= h
h= h1

10026 if(hmaxu.lt.hmax) hmax = hmaxu
joh = koh + 1
zq=f(abspnr,t)
afanz=afanz+1
yr = zero

```

```

18 diffanz=afanz
    redta=t
    redte=t+h
c -----
c ----- trapezoidal sum -----
c -----
20 tn=t+h
    if(h1.eq.h) tn=tend
    fcm=dabs(h)/hmax
    if(fcm.lt.fmin) fcm=fmin
c
    yn=zero
    yl=zero
    do 35 j=1,jm
    m=nj(j)
    g=h/dfloat(m)
    zq1=f(abspnr,t+h)
    afanz=afanz+1
    yl=(zq1+zq)*half
    yr=yl*g
    goto 30
21 if(mod(m,3).eq.0) goto 23
    yl1=yl
    do 22 i=1,m,2
    yl=yl+f(abspnr,t+dfloat(i)*g)
    afanz=afanz+1
22 continue
    yr=yl*g
    goto 30
23 do 24 i=1,m,6
    fi=dfloat(i)
    yn=yn+f(abspnr,t+fi*g)+f(abspnr,tn-fi*g)
24 afanz=afanz+2
    yr=(yn+yl1)*g

c -----
c ----- extrapolation -----
c -----
30 v=dt(1)
    c=yr
    dt(1)=c
    if(j.eq.1) goto 35
    ta=c
    do 31 k=2,j
    jk=j-k+1

```

```

b1=d(j,jk)
w=c-v
u=w/(b1-one)
c=b1*u
v=dt(k)
dt(k)=u
31   ta=u+ta
errh=dabs(u)
yr=ta
ta=dabs(y+yr)
if(ta.lt.scale) ta=scale
err=errh/ta
konv=0
if (err.lt.eps) konv=1
err=err/eph
c -----
c ----- order control -----
c -----
k=j-1
l=j+k
fc=err**one/dfloat(l))
if(fc.lt.fcm) fc=fcm
c ===== optimal order determination
omj=fc*a(j)
if(j.gt.2.and.omj*one1.gt.omjo.or.k.gt.joh) goto 32
ko=k
jo=j
omjo=omj
fco=fc
32   continue
if(j.lt.koh.and.nstep.gt.0.and.tn.lt.tend) goto 35
if(konv.eq.0) goto 33
if(ko.lt.k.or.incr(j).lt.0) goto 40
c ===== possible increase of order
if(nred(ko).gt.0) nred(ko)=nred(ko)-1
if(j.ge.jm) goto 40
fc=fco/al(j,k)
if(fc.lt.fcm) fc=fcm
j1=j+1
if(a(j1)*fc*one1.gt.omjo) goto 40
fco=fc
ko=jo
jo=jo+1
goto 40

```

```

c -----
c ----- convergence monitor -----
c -----
33  red=one/fco
    jk=km
    if(joh.lt.km) jk=joh
    if(k.ge.jk) goto 36
    if(ko.lt.koh) red=al(koh,ko)/fco
34  if(al(jk,ko).lt.fco) goto 36
35  continue
c -----
c ----- stepsize reduction -----
c -----
36  red=red*safe
    alth = h
    h = h * red
    if(nstep.eq.0) goto 38
    nred(koh)=nred(koh)+1
    do 37 l=koh,km
37  incr(l)=-2-nred(koh)
38  jred=jred+1
    if(jred.gt.jrmax) then
        print*, "-----"
        print*, "t-id",mytaskid(),"Fehlerausgang step red"
        jred>jrmax",
        1   jred,jrmax
        print*, "Reduktions-Startintervall:",redta,redte
        print*, "aktuelles Intervall   :",t,tn
        print*, "-----"
        goto 51
    end if
c =====
c   Data Transfer to MASTERTRAPEX (reduction-case)
c =====
neuh = tend - tn
if (neuh .gt. 2.0d0*h) then
    tend = tn
    h1 = tend - t
    tivale=tend
    if (hmax .gt. h1) hmax= h1
    if (testtag(9999)) stop
    send (tag=3,taskid=org) neuh,h,tend
end if
goto 20
c

```

```

c -----
c ----- preparations for next basic step -----
c -----
40  continue
    altt=t
    t=tn
    h1=tend-t
    y=y+yr
    error=error+errh
    zq=zq1
    nstep=nstep+1
    if (nstep.gt.nstmax) then
        print*, "-----"
        print*, "t-id:",mytaskid(),
        1 "nach 40, nstep>nstmax nstep,nstmax:",nstep,nstmax
            print*, "Reduktions-Startintervall:",redta,redte
            print*, "aktueller Intervall   :",t,tn
            print*, "-----"
        goto 51
    end if
c
c -----
c ----- stepsize prediction -----
c -----
    if(fco.ne.fcm) then
        hr=h
    end if
    h=h/fco
    koh=ko
    joh=koh+1
    if(dabs(h).le.teps) then
        print*, "-----"
        print*, "t-id:",mytaskid(),"st-pred h<= teps
        h,teps:",h,teps
        print*, "Reduktions-Startintervall:",redta,redte
        print*, "aktueller Intervall   :",t,tn
        print*, "-----"
        goto 51
    end if
c
41  continue
    if(dabs(h1).le.teps) goto 50
    q=h1/h
    if(q.le.epmach) goto 50

```

```

if (.5d0*h1.le.h) then
    tan=tn+h
    esth = tend-tan
else
    tan=tn+.5d0*h1
    esth= h
end if
c =====
c Data Transfer to MASTERTRAPEX
c (next basic step case)
c =====
if(tend-tan .gt. zero) then
    if (testtag(9999)) stop
    send(tag=7,taskid=org) tan,tend,esth
    tend=tan
    tivale=tend
    h1=tend-t
    q=h1/h
end if
goto 15
c
c solution exit
50   h=hr
     ta=dabs(y)
     if(ta.lt.scale) ta=scale
     hmax=hmaxu
     goto 9999
c
c fail exit
51   h=zero
     hmax=hmaxu
9999  if (testtag(9999)) stop1
       send(taskid=org,tag=4) nstep,afanz,y,error,h
       goto 10005
10000 receive(sender=org,tag=9999)
      stop
c
c **** End TRAPEX-Part ****
c
end

```

Anhang C

Die aus dem Testset oben ausgewählten Beispiele wurden mit einem modifizierten sequentiellen Trapex TPXGRAF berechnet. Die Abzissen-Werte t_n sind die Werte der rechten Ränder der Subintervalle, die TPXGRAF zur Berechnung des Integralwertes verwendet hat. Über der Ordinate sind die Funktionswerte des jeweiligen Beispiels aufgetragen. Für die Interpolation zwischen zwei Werten $F(t_n)$, $F(t_{n+1})$ wurde in GRAZIL der Parameter &Curve * -1 -1 0 0 3 (eine stückweise kubische Hermit'sche Interpolation mit HERMITE FUNCTION EVALUATOR) eingestellt.

Literatur

- [1] Arvin & K.P. Gostelow: *The U-Interpreter; Computer.* Vol. 15, S. 42–49 (1982).
- [2] H.-J. Bauer: *Entwicklung leistungsfähiger Extrapolationscodes.* Diplomarbeit, Universität Heidelberg (1983).
- [3] D. Boles: *Benutzeranleitung des graphischen Generators von HADES-Dateien für das SUPRENUM-Simulationssystem HADESGEN.* Gesellschaft für Mathematik und Datenverarbeitung mbH (GMD), Sankt Augustin (1987).
- [4] R. Bulirsch: *Bemerkungen zur Romberg-Integration.* Numer. Math. 6, S. 6–16 (1964).
- [5] P. Deuflhard & H.J. Bauer: *A Note on Romberg Quadrature.* Preprint Nr.: 169; Universität Heidelberg (1982).
- [6] P. Deuflhard: *Order and Stepsize Control in Extrapolation Methods.* Preprint Nr.: 93, Universität Heidelberg (1980).
- [7] H. Engel: *Numerical Quadrature and Cubature.* Academic Press, London (1980).
- [8] C.A.R. Hoare: *Monitors: An Operating System Structuring Concept.* CACM Vol.17, S. 549 ff (1974).
- [9] C.A.R. Hoare: *Communicating Sequential Processes.* CACM Vol. 21, S. 666–667 (1978).
- [10] Hockney & Jesshope: *Parallel Computer 2.* Adam Hilgers, Bristol and Philadelphia, S. 103 ff (1988).
- [11] D. Kahaner: *Comparison of Numerical Quadrature Formulas.* In J. Rice (Ed.): Mathematical Software; Academic Press, London (1971).
- [12] J. Langendorf & O. Paetsch: *GRAZIL (Graphical ZIB Language).* Konrad-Zuse-Zentrum für Informationstechnik Berlin, Technical Report 87-7 (1987).
- [13] D.M. Nicol & F.H. Willard: *Problem Size, Parallel Architecture, and optimal Speedup.* J. of Parallel and Distributed Computing 5, S. 404–420 (1988)
- [14] E. Odijk & W. Bronnenberg: *Parallel Computing: the object oriented Approach.* Conference Papers: Plenary Sessions and Stream ‘A’, S. 33–39; CON-PAR88, Manchester (1988).

-
- [15] W. Romberg: *Vereinfachte numerische Integration.* DET KNONGELIGE NORSKE VIDENSKABSVERS SELSKABS FORTHANDLINGER. Vol. 28, Nr. 7 (1955).
 - [16] C. Tietz: *Das Benutzer-Interface des SUPRENUM-Simulationssystems.* Gesellschaft für Mathematik und Datenverarbeitung mbH (GMD), Ver. 1.5, S. 9 (1987).

Veröffentlichungen des Konrad-Zuse-Zentrums für Informationstechnik Berlin
Preprints

Dezember 1988

SC 86-1. P. Deuflhard; U. Nowak. *Efficient Numerical Simulation and Identification of Large Chemical Reaction Systems.*

SC 86-2. H. Melenk; W. Neun. *Portable Standard LISP for CRAY X-MP Computers.*

SC 87-1. J. Anderson; W. Galway; R. Kessler; H. Melenk; W. Neun. *The Implementation and Optimization of Portable Standard LISP for the CRAY.*

SC 87-2. Randolph E. Bank; Todd F. Dupont; Harry Yserentant. *The Hierarchical Basis Multigrid Method.* (vergriffen)

SC 87-3. Peter Deuflhard. *Uniqueness Theorems for Stiff ODE Initial Value Problems.*

SC 87-4. Rainer Buhtz. *CGM-Concepts and their Realization.*

SC 87-5. P. Deuflhard. *A Note on Extrapolation Methods for Second Order ODE Systems.*

SC 87-6. Harry Yserentant. *Preconditioning Indefinite Discretization Matrices.*

SC 88-1. Winfried Neun; Herbert Melenk. *Implementation of the LISP-Arbitrary Precision Arithmetic for a Vector Processor.*

SC 88-2. H. Melenk; H.M. Möller; W. Neun. *On Gröbner Bases Computation on a Supercomputer Using REDUCE.*

SC 88-3. J. C. Alexander; B. Fiedler. *Global Decoupling of Coupled Symmetric Oscillators.*

SC 88-4. Herbert Melenk; Winfried Neun. *Parallel Polynomial Operations in the Buchberger Algorithm.*

SC 88-5. P. Deuflhard; P. Leinen; H. Yserentant. *Concepts of an Adaptive Hierarchical Finite Element Code.*

SC 88-6. P. Deuflhard; M. Wulkow. *Computational Treatment of Polyreaction Kinetics by Orthogonal Polynomials of a Discrete Variable.*

SC 88-7. H. Melenk; H. M. Möller; W. Neun. *Symbolic Solution of Large Stationary Chemical Kinetics Problems.*

SC 88-8. Ronald H. W. Hoppe; Ralf Kornhuber. *Multi-Grid Solution of Two Coupled Stefan Equations Arising in Induction Heating of Large Steel Slabs.*

SC 88-9. Ralf Kornhuber; Rainer Roitzsch. *Adaptive Finite-Element-Methoden für konvektions-dominante Randwertprobleme bei partiellen Differentialgleichungen.* (erscheint Feb 1989)

SC 88-10. C. -N. Chow; B. Deng; B. Fiedler. *Homoclinic Bifurcation at Resonant Eigenvalues.* (erscheint Mar 1989)



Veröffentlichungen des Konrad-Zuse-Zentrums für Informationstechnik Berlin
Technical Reports

Dezember 1988

TR 86-1. H.J. Schuster. *Tätigkeitsbericht 1985.* (vergriffen)

TR 87-1. Hubert Busch; Uwe Pöhle; Wolfgang Stech. *CRAY-Handbuch. - Einführung in die Benutzung der CRAY..*

TR 87-2. Herbert Melenk; Winfried Neun. *Portable Standard LISP Implementation for CRAY X-MP Computers. Release of PSL 3.4 for COS.*

TR 87-3. Herbert Melenk; Winfried Neun. *Portable Common LISP Subset Implementation for CRAY X-MP Computers.*

TR 87-4. Herbert Melenk; Winfried Neun. *REDUCE Installation Guide for CRAY 1 / X-MP Systems Running COS Version 3.2.*

TR 87-5. Herbert Melenk; Winfried Neun. *REDUCE Users Guide for the CRAY 1 / X-MP Series Running COS. Version 3.2.*

TR 87-6. Rainer Buhtz; Jens Langendorf; Olaf Paetsch; Danuta Anna Buhtz. *ZUGRIFF - Eine vereinheitlichte Datenspezifikation für graphische Darstellungen und ihre graphische Aufbereitung.*

TR 87-7. J. Langendorf; O. Paetsch. *GRAZIL (Graphical ZIB Language).*

TR 88-1. Rainer Buhtz; Danuta Anna Buhtz. *TDLG 3.1 - Ein interaktives Programm zur Darstellung dreidimensionaler Modelle auf Rastergraphikgeräten.*

TR 88-2. Herbert Melenk; Winfried Neun. *REDUCE User's Guide for the CRAY 1 / CRAY X-MP Series Running UNICOS. Version 3.3.*

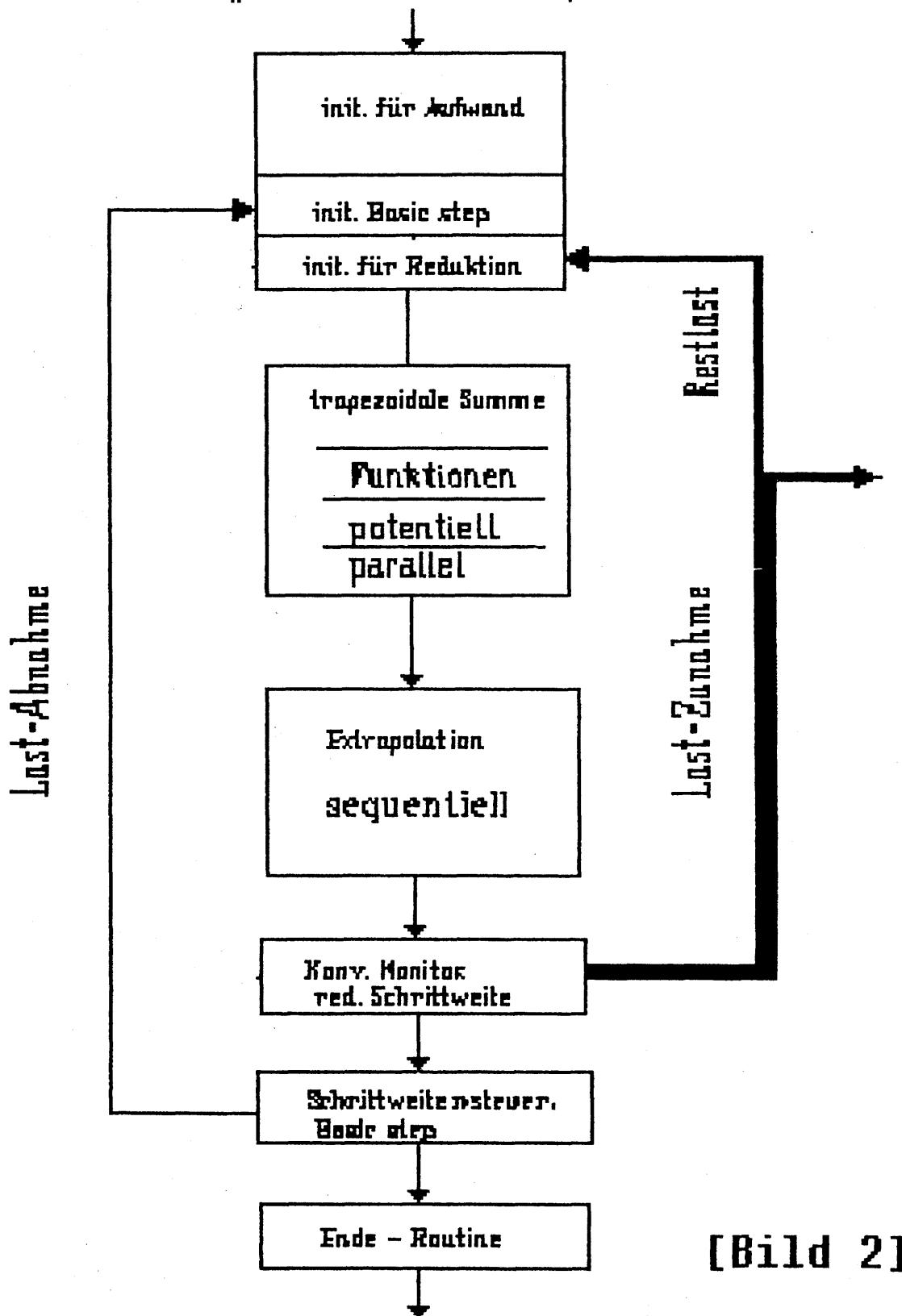
TR 88-3. Herbert Melenk; Winfried Neun. *REDUCE Installation Guide for CRAY 1 / X-MP Systems Running UNICOS. Version 3.3.*

TR 88-4. Danuta Anna Buhtz; Jens Langendorf; OLaf Paetsch. *GRAZIL-3D. Ein graphisches Anwendungsprogramm zur Darstellung von Kurven- und Funktionsverläufen im räumlichen Koordinatensystem.*

TR 88-5. Gerhard Maierhöfer; Georg Skorobohatyj. *Ein paralleler, adaptiver Algorithmus zur numerischen Integration ; seine Implementierung für SUPRENUM-artige Architekturen mit SUSI.*

Informationsfluß

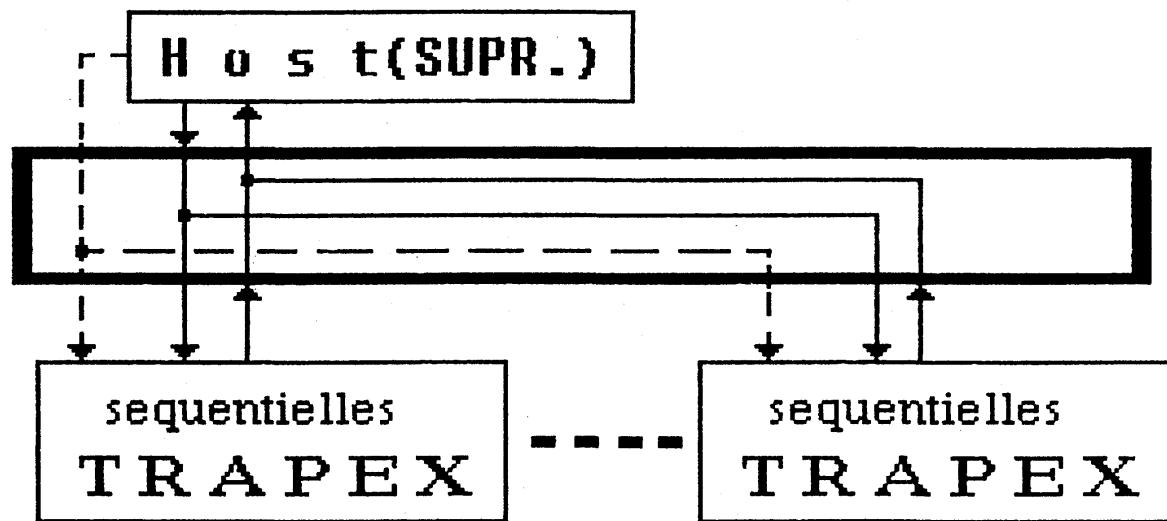
sequentielle Trapex



[Bild 2]

Informationsfluß

statisch paralleler Trapex



— — — — — SUPRENUM-Cluster-Bus
(inter/intra)

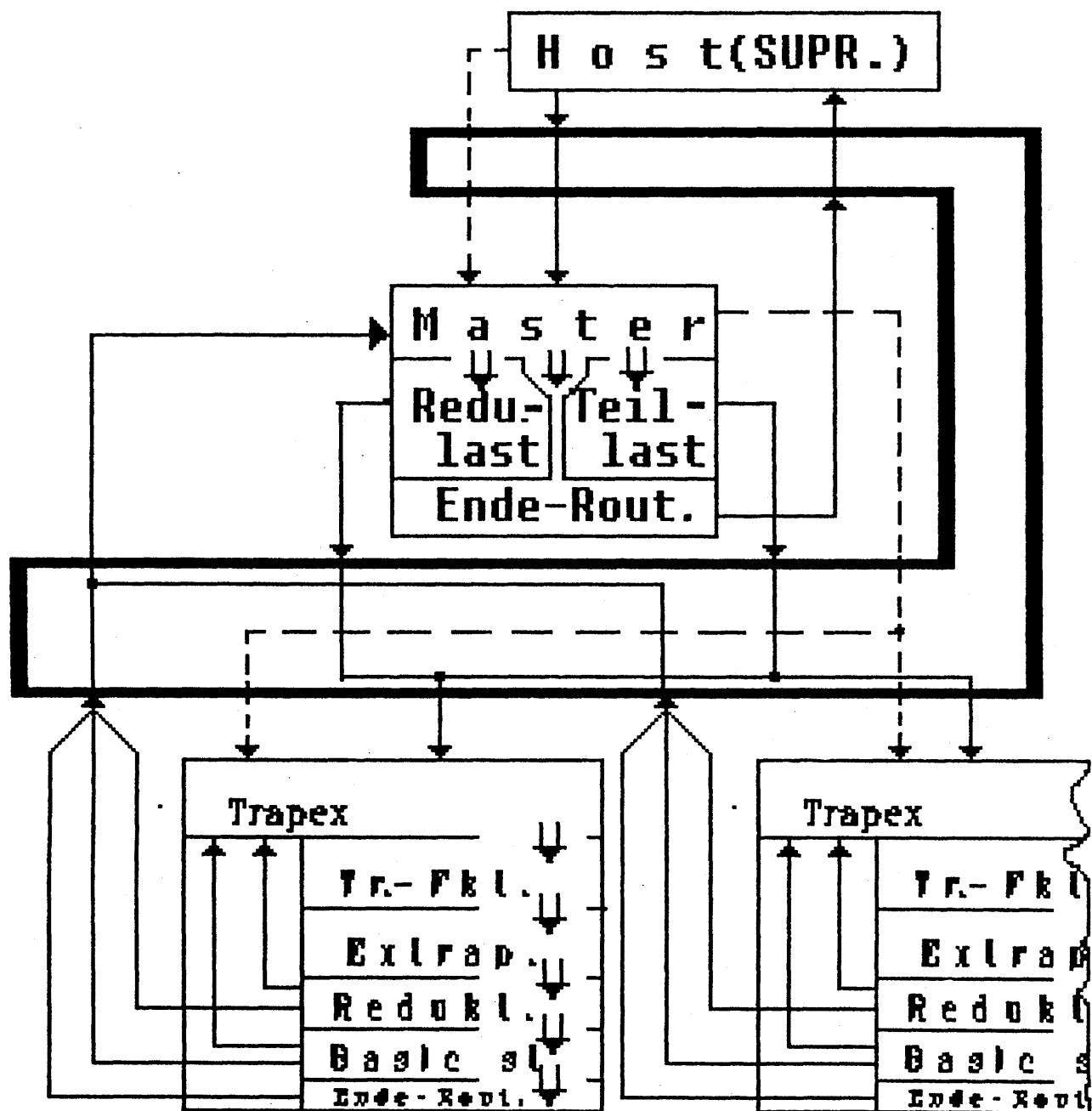
— — — — → Taskinitialisierung

— — — — → Datenfluß-Richtung

[Bild 3]

Informationsfluß

dynamisch paralleles Trapex



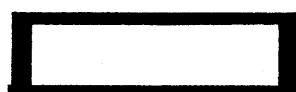
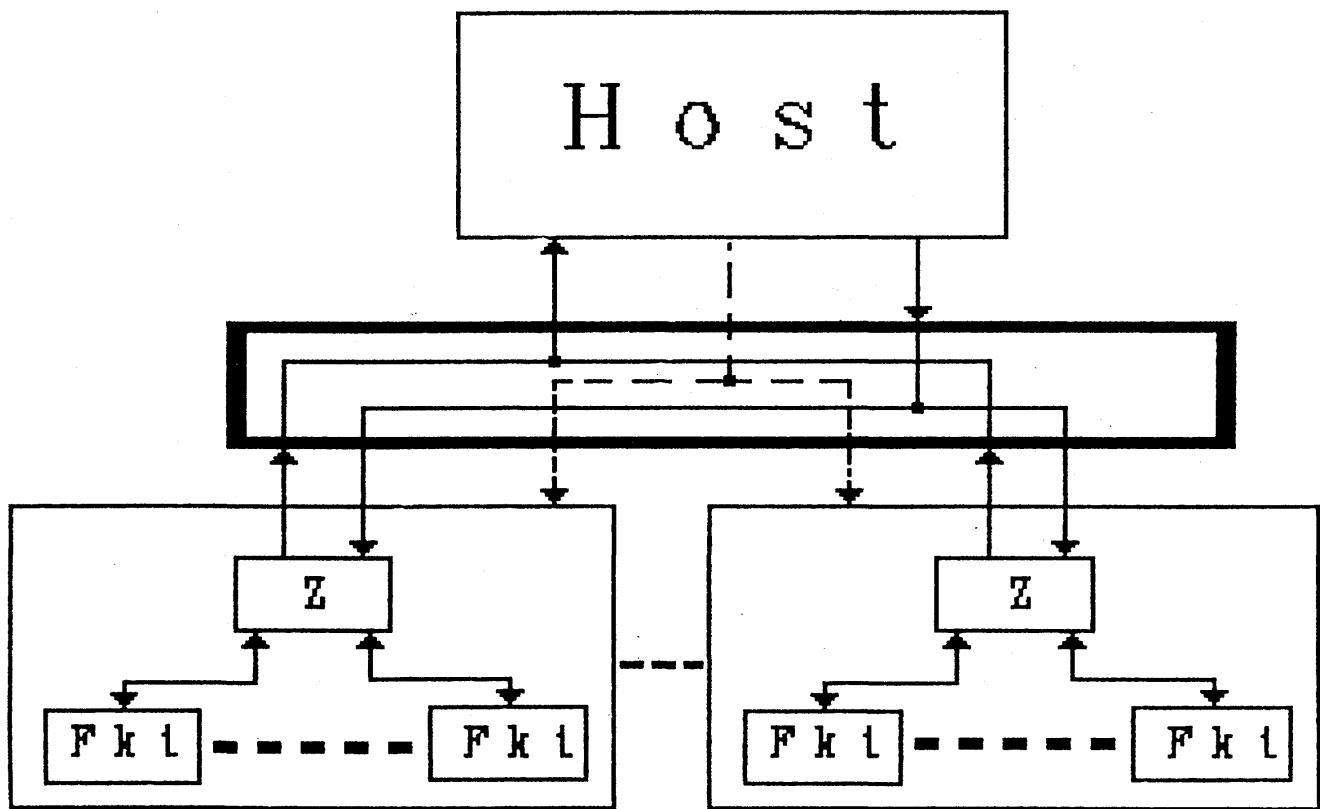
SUPRENUM-Cluster-Bus
(inter/intra)

→ Taskinitialisierung
→ Datenfluß-Richtung

[Bild 4]

Informationsfluß

optimales paralleles Trapex



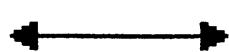
zentraler Bus

Z

Zentralknoten des
n-Tupel

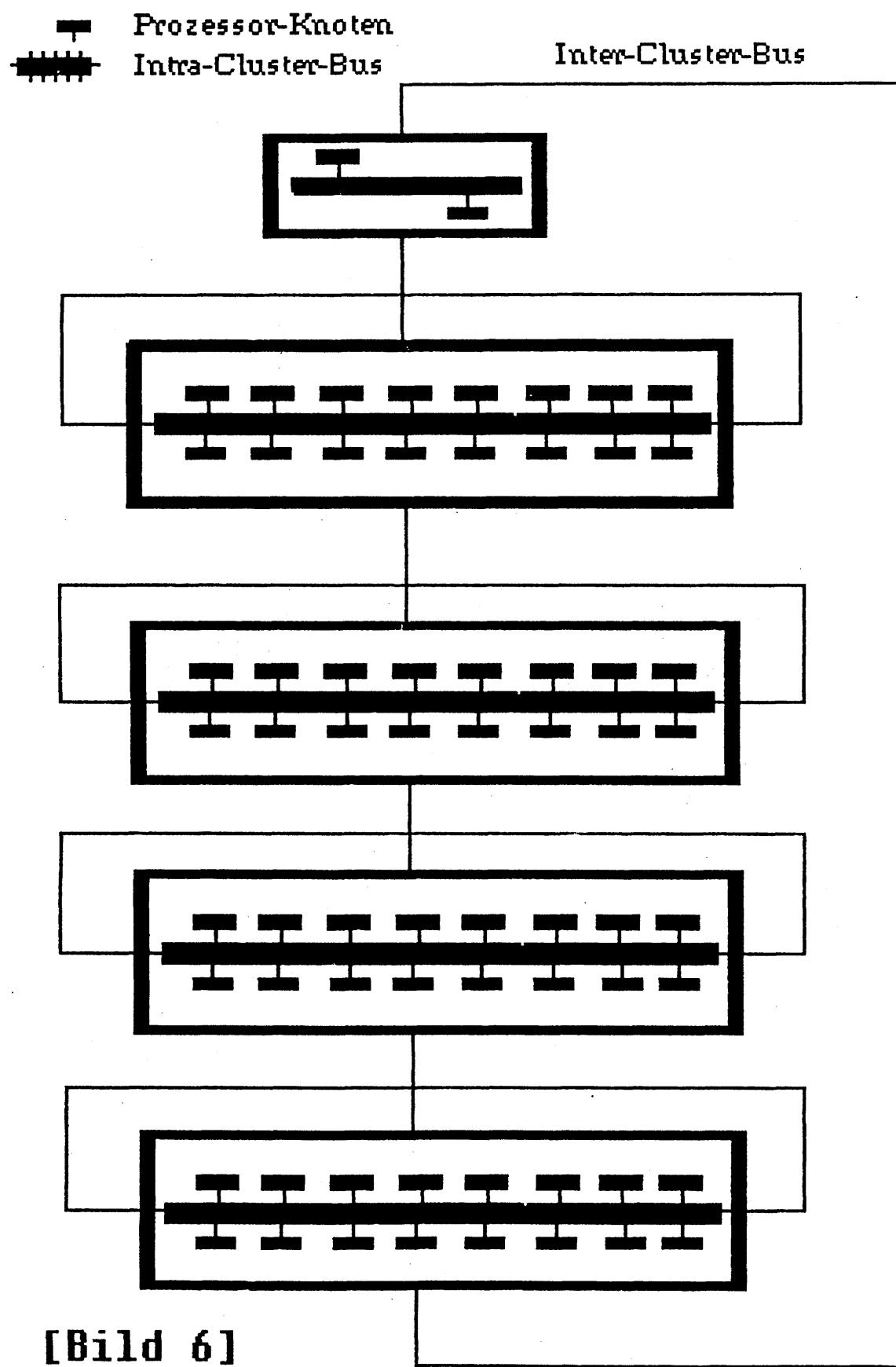
Fkt

einfacher Knoten der
nur Funktionen berech.
bidirektionale
nearest-neighbour



[Bild 5]

SUPRENUM-Architektur



[Bild 6]

>>>>> TPXGRAF Beispiel-Nr: 1 mit seq. TRAPEX <<<<<

$$Y(tn) = e^{tnx}$$

2.72

2.55

2.37

2.2

2.03

$$(U_t^5 - U_t^{1.86}) = Y$$

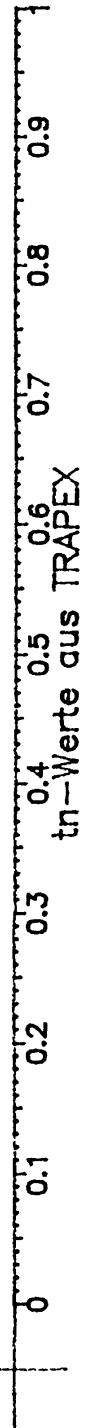
1.69

1.52

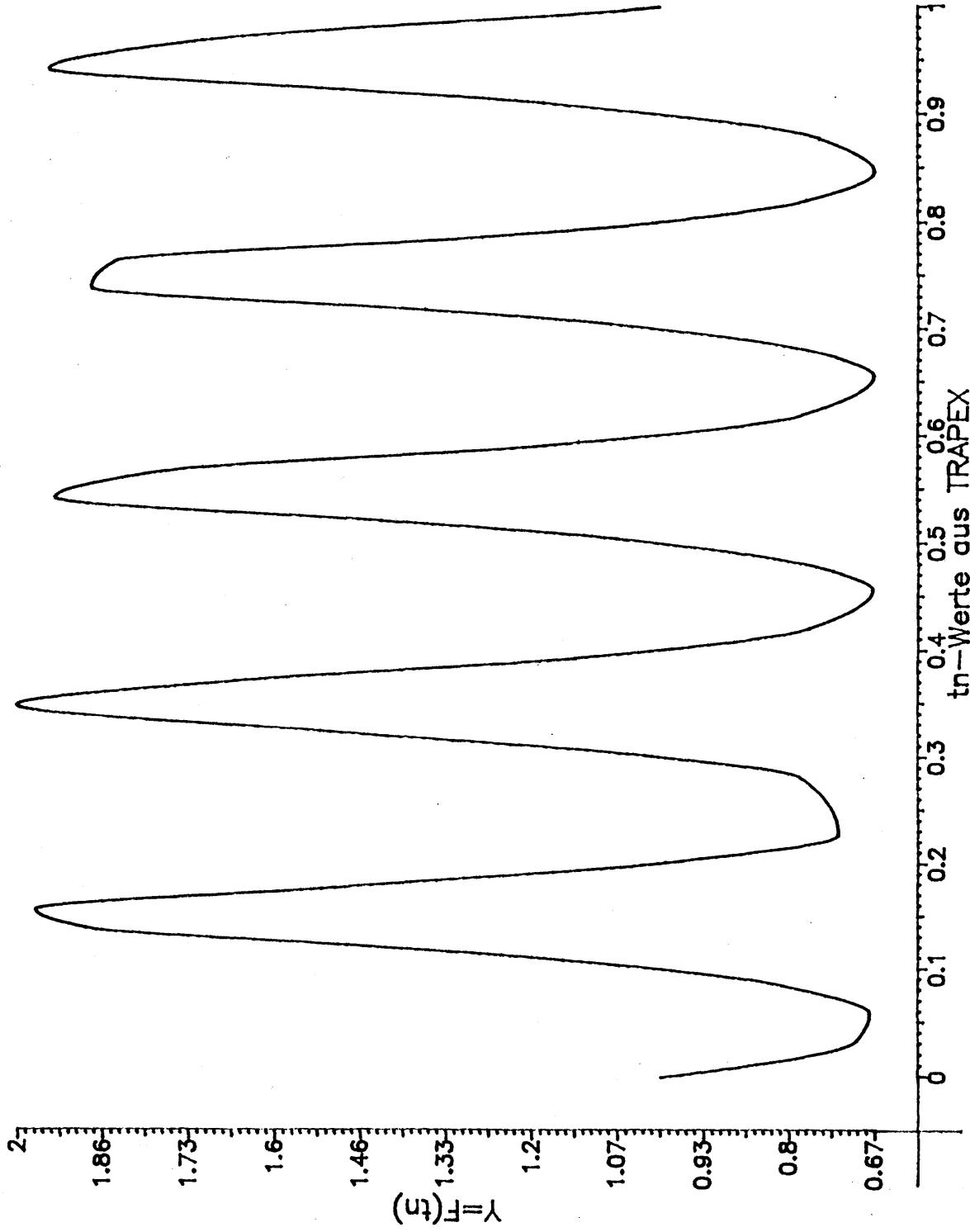
1.34

1.17

1

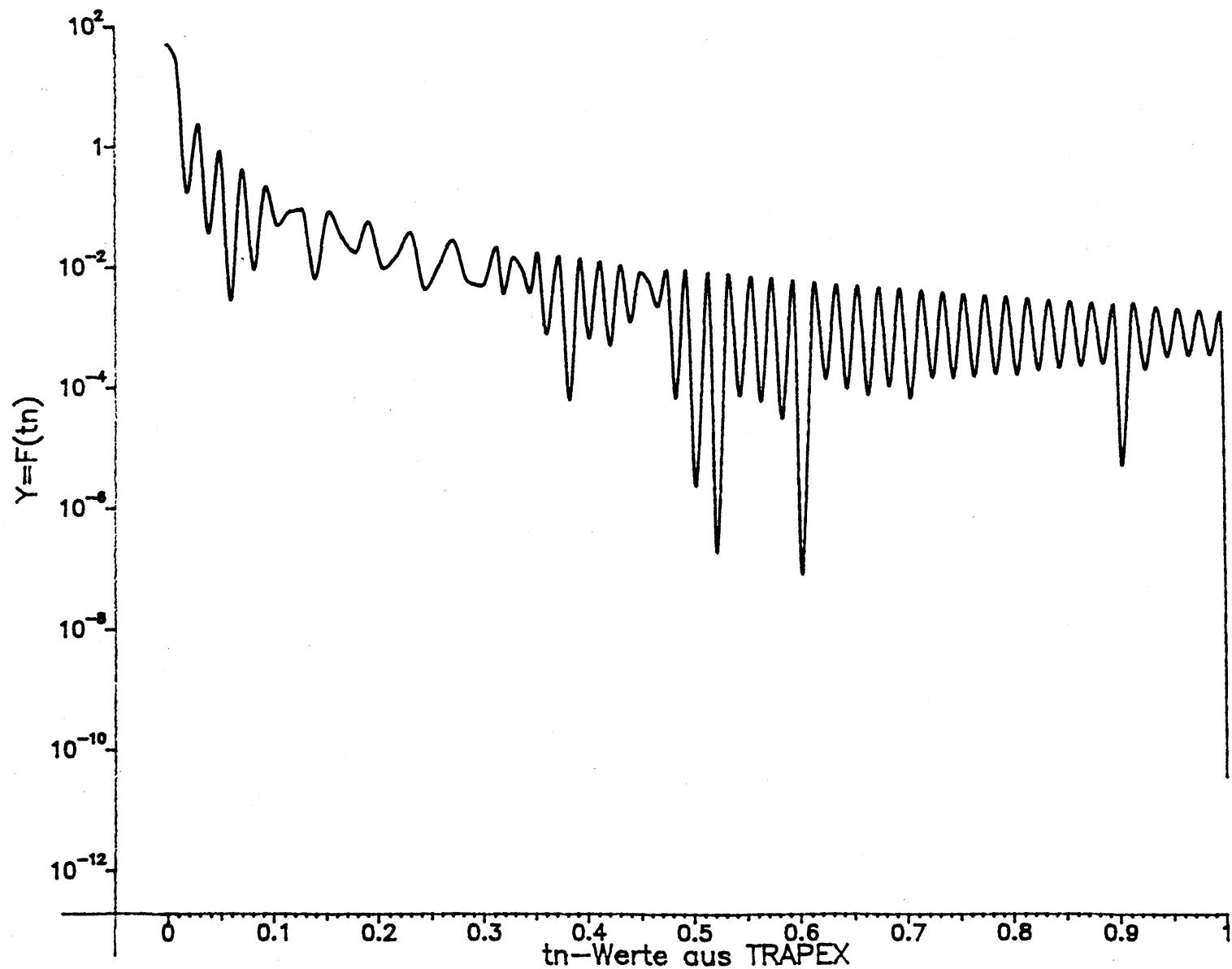


>>> TRAPEZ Beispiel-Nr: 9 mit seg. TRAPEZ
 $y(t_n) = 2 / (\sin(10 \cdot \pi \cdot t_n))$ [0, +1]



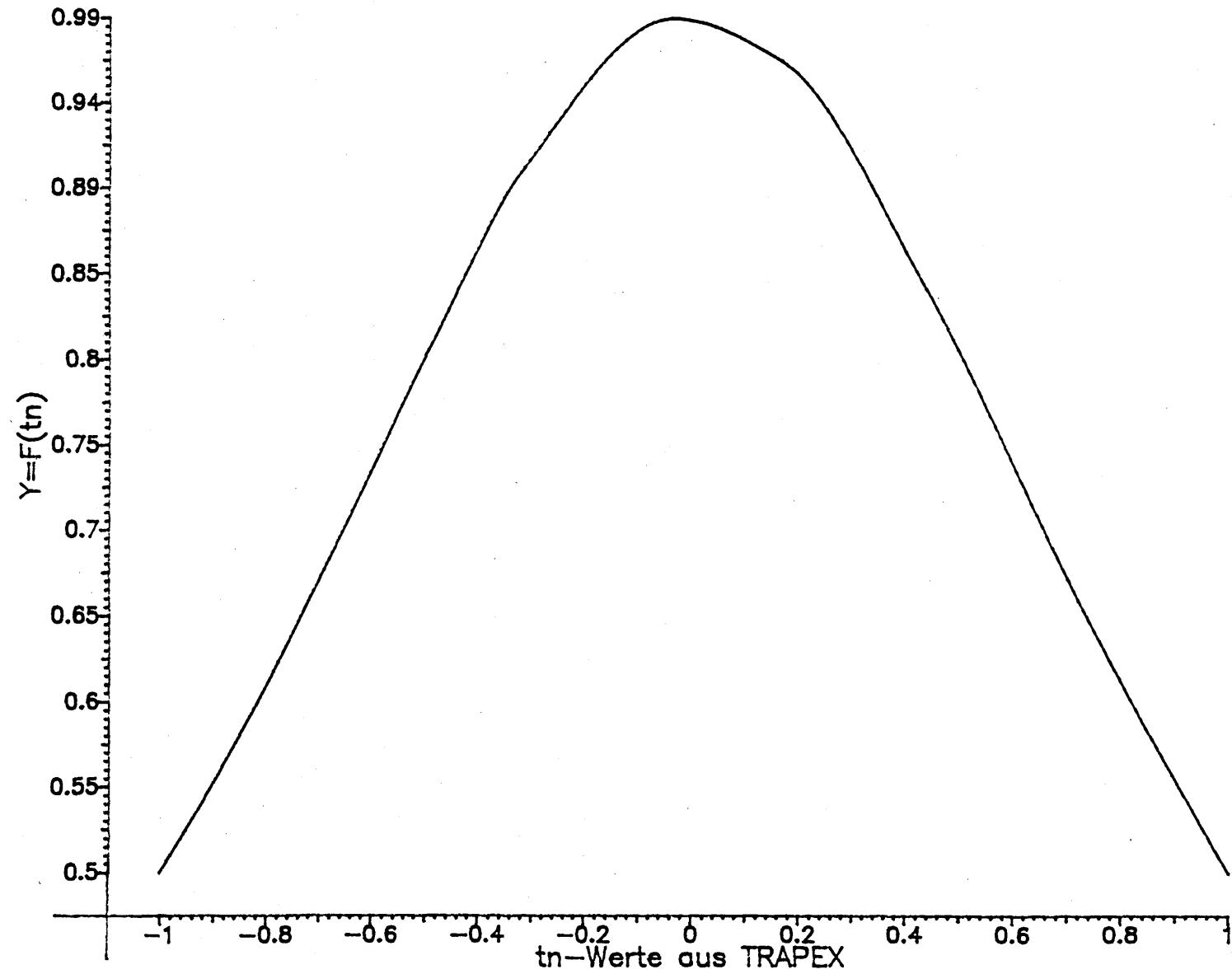
>>>>> TPXGRAF Beispiel-Nr:17 mit seq. TRAPEX <<<<<

$$Y(tn) = \sin(50\pi tn)^2 / 50\pi^2 [0, +1]$$



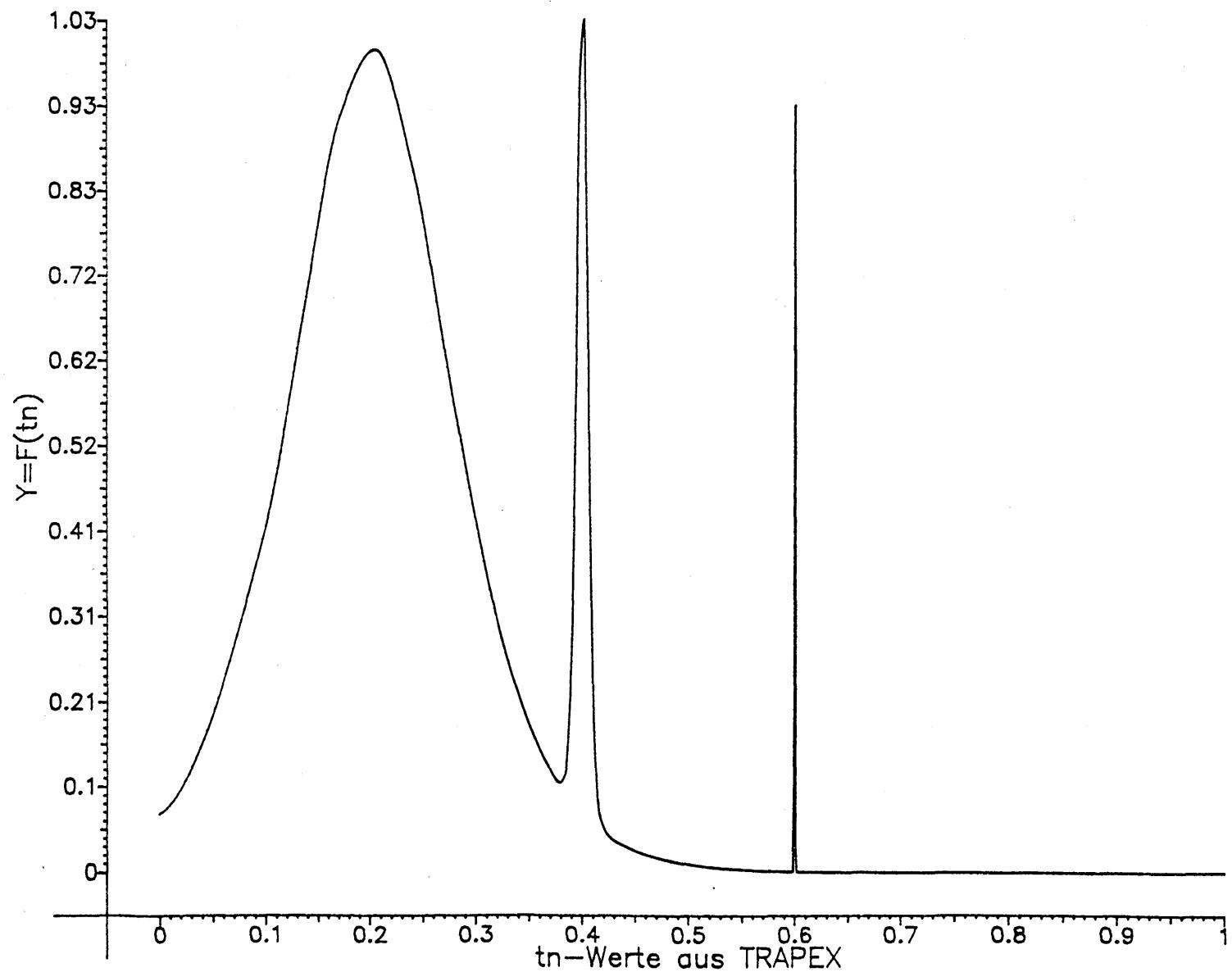
>>>>> TPXGRAF Beispiel-Nr:20 mit seq. TRAPEX <<<<<

$$Y(tn) = 1/(x^{**2} + 1.005) \quad [-1, +1]$$



>>>>> TPXGRAF Beispiel-Nr:21 mit seq. TRAPEX <<<<<

$$Y = (1/\cos(10x-2))^{**}2 + (1/\cos(100x-40))^{**}4 + (1/\cos(1000x-600))^{**}6 \quad [0, 1]$$



>>>>> TPXGRAF Beispiel-Nr:22 mit seq. TRAPEX <<<<<

$$Y(t_n) = 1/(x^{**2} + 10^{**(-4)}) \quad [-1, +1]$$

