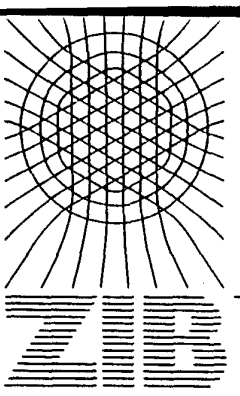
Konrad-Zuse-Zentrum für Informationstechnik Berlin



Gerhard Maierhöfer

Ein paralleler adaptiver Algorithmus für die numerische Integration

Preprint SC 89–8 (November 1989)

Herausgegeben vom
Konrad-Zuse-Zentrum für Informationstechnik Berlin
Heilbronner Strasse 10
1000 Berlin 31
Verantwortlich: Dr. Klaus André
Umschlagsatz und Druck: Rabe KG Buch- und Offsetdruck Berlin

ISSN 0933-7911

Ein paralleler adaptiver Algorithmus für die numerische Integration

Schlüsselwörter: numerischer Algorithmus, Romberg Quadratur,
paralleler adaptiver Algorithmus,
dynamische Lastverteilung und Prozessorzahl,
lokaler Speicher, Nearest-Neighbour-Architektur
Transputer, TDS, OCCAM2

Einleitung

Der vorliegende Bericht beschreibt eine (auf dem Master-Slave-Prinzip basierende) Parallelisierung eines sequentiellen Algorithmus zur numerischen Berechnung eines Integral-Wertes (Romberg-Quadratur). Es wird ein modifiziertes Farming-Modell verwendet. Sowohl der sequentielle wie auch der parallele Algorithmus ist adaptiv.
Unter Adaptivität soll in diesem Fall verstanden werden, daß der Algorithmus zur Laufzeit Aufwandsabschätzungen vornimmt und entsprechend einer vorgegebenen Optimierungsstrategie die Anzahl der Berechungsschritte wählt; d.h. im sequentiellen Fall erfolgt eine problemabhängige Schrittweitensteuerung (Schrittweite bedeutet in diesem Zusammenhang die Länge eines Teil-Intervalles des Intervalles, für das eine Näherung berechnet werden soll), zu der im parallelen Fall eine problemabhängige Steuerung der Anzahl der verwendeten Prozesse (CPUs) hinzukommt. Im parallelen Fall ist es weiterhin wesentlich, daß alle eingesetzten CPUs ständig mit Berechnungen zur Problemstellung (hier numerische Integration) und möglichst wenig mit Interprozeßkommunikation beschäftigt sind. Deshalb muß mindestens eine statische, besser aber dynamische Lastverteilung auf die verschiedenen Rechenprozesse erfolgen. Aus der Problemstellung - man integriert mit numerischen Verfahren, weil eine geschlossene Lösung für das gesuchte Integral (im interessierenden Intervall) nicht bekannt oder nicht möglich ist - ergibt sich, wie hoch der Rechenaufwand pro Teilintervall bei einer beliebigen Intervallunterteilung ist. Ein paralleler adaptiver Algorithmus wird deshalb dynamisch (zur Laufzeit) aus der "lokalen" Aufwands-Ermittlung bzw. -Abschätzung eine Arbeits(Last)-Verteilung vornehmen müssen.

Damit eine dynamische Lastverteilung möglich wird, muß in einer geeigneten Weise eine "Buchführung" darüber erfolgen, welcher Prozeß mit wieviel "Arbeit" beschäftigt ist. In einfachster Form bedeutet dies, eine Buchführung über den Zustand der CPUs, die die Zustände "beschäftigt", "unbeschäftigt" haben können, durchzuführen. Wie die Buchführung erfolgt, hängt nun wesentlich davon ab, welche Architektur (Hardware, Betriebsystem und Implementierungssprache) eingesetzt wird. In [BeEs88] wird ein Algorithmus beschrieben, der auf einer definierten Architektur, einem Hypercube der Dimension 2, läuft und auf einer festen Anzahl von Knoten eine zyklische Lastverteilung anstrebt. Wir betrachten in diesem Artikel Local-Memory-Systeme, bei denen zur Laufzeit sowohl die Zahl der zu verwendenden Knoten geändert und deren Initialisierung veranlaßt werden kann, als auch eine Lastverteilung möglich ist.
Der Algorithmus wurde für busgekoppelte Systeme (SUPRENUM) [MaSk88] wie auch auf Nearest-Neighbour-Architekturen (Transputer) implementiert. Der gefundene Ansatz ist aber auch auf anderen Architekturen realisierbar (entsprechende Arbeiten werden derzeit durchgeführt: objektorientierter Ansatz). Meßergebnisse liegen sowohl für eine feingranulierte (horizontale) wie auch für eine grobgranulierte (vertikale) Parallelisierung vor. Zum Testen wurden die Beispiele des Testset aus [Bau83] verwendet, die eine Teilmenge aus [En80] und [Ka71] bilden.

Numerischer Hintergrund des sequentiellen Trapex

Der im nachfolgenden Text als TRAPEX bezeichnete Algorithmus verwendet zur numerischen Integration die von Romberg [Ro55] beschriebene Methode. Die Theorie ist hier soweit skizziert, wie sie zum Verständnis der Parallelisierungsproblematik notwendig ist.
Der Wert eines Integrales

$$IW := \int_{t_a}^{t_e} f(t)\,dt \quad \text{im Intervall } [\, t_a, t_e \,] \tag{1}$$

und f(t) im Intervall genügend oft differenzierbar, wird durch die Trapez-Summe TS berechnet:

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

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

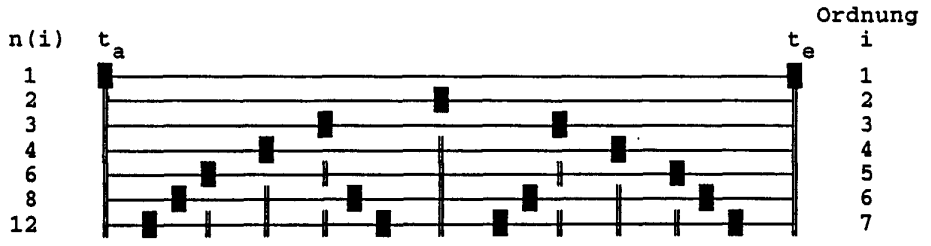
mit $h_i = \frac{t_e - t_a}{n(i)}$, wobei $(n_i)_{i\in N}$= (1,2,3,4,6,8,12, ...) die (Bulirsch-) Folge von ganzen Zahlen ist, die von Bulirsch [Bu64] als geeignet beschrieben wird. Die Frage, welches TS_i zur Berechnung des Integralwertes herangezogen wird und wie ein gegebenes Intervall [a,b] optimal in Teilintervalle [t,t+H] schrtttweise aufgeteilt werden kann , wird in [DeBa82] und das verwendete Konvergenzmodell in [De8Ø] behandelt. Es ensteht eine Aufteilung des Grundintervalles [a,b] mit Teilungspunkten t_1=a < t_2 < .... <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_i ein polynomiales Extrapolationsverfahren angewandt und über eine Fehlerabschätzung festgestellt, ob Konvergenz vorliegt (die geforderte Genauigkeit erreicht ist).
Für die Polynom-Extrapolation (Aitken-Neville Algorithmus) gilt (siehe auch [De8Ø]) die "Dreiecksregel":

$$TE_{i,1} = TS_i \; ; \; 1\le i\le 7, \; 2\le k\le i$$

und

$$TE_{i,k} = TE_{i,k-1} + \frac{TE_{i,k-1} - TE_{i-1,k-1}}{\left[\frac{n_i}{n_{i-k+1}} \right]^2 - 1} \qquad (3)$$

Das beschriebene Verfahren ist adaptiv. Für jedes TS_i^j, jϵ Intervallfolge, wird, in einem "Konvergenzfenster" überprüft, ob "Konvergenz" (die geforderte Genauigkeit) vorliegt, H_k^j reduziert werden muß oder das nächste TS_i^{j+1} berechnet werden kann. Für jedes T_i^j wird eine optimale Ordnung i und ein H_k^j ermittelt. Der Algorithmus adaptiert die H_k^j -Folge entsprechend den lokalen Informatkon (sich ändernder ümmungsradius des Graphen der Funktion). Insgesamt ist es eine Optimierungsstrategie, die zum Ziel hat, die Anzahl der zu berechnenden Funktionswerte über das Gesamtintervall zu minimieren.
[Bild 1] zeigt, bei welcher Ordnung i schon berechnete Fkt-Werte aus einer schon "früher" berechneten Ordnung i-2 wieder verwendet werden können.



[Bild 1] Folge der Funktionswerte jeder Ordnung i

▮ bezeichnen Fkt-Werte, die neu berechnet werden müssen

╫ schon berechnete Fkt-Werte, die übernommen werden.

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.

Parallelisierung

Die erste Parallelisierungsmöglichkeit liefert die Theorie der Integralrechnung. Formel (1) kann modifiziert werden zu:

$$IW := \int_{t_a}^{t_1} \cdots\cdots + \int_{t_{n-1}}^{t_n} + \int_{t_n}^{t_{n+1}} \cdots\cdots + \int_{t_{m-1}}^{t_e} \qquad (4)$$

wobei m die Anzahl zur Verfügung stehender Prozessoren ist
Im Normalfall wird man äquidistante Teilintervalle $t_n - t_{n-1}$ wählen als initiale Aufteilung wählen. Für diese Vorgehensweise ist das Farming-Modell geeignet. Das Problem besteht nun darin, alle m Prozessoren möglichst gleichmäßig auszulasten, denn der Rechenaufwand für die m Teilintervalle ist im allgemeinen nicht gleich hoch. Intervalle mit "glatteren" Kurvenstücken sind aufwandsärmer als solche, bei denen sich der Krümmungsradius stark ändert. Dies ist nicht global entscheidbar. Daher muß ein Prozessor, der einen erhöhten Aufwand feststellt, "Last" abgeben können, vorzugsweise an einen Prozessor, der keinen erhöhten Aufwand feststellt.
Eine Lastabgabe kann in dem Sinne erfolgen, daß ein Teil des Restin-

tervalles oder das ganze Restintervall an einen anderen Prozeß zur Berechnung abgegeben wird und nur der verbleibende Teil von der lastabgebenden CPU berechnet wird.

Sei L_{SI} Subintervall, das ein Prozessr berechnet
L_{TI} Teilintervall(e) das/die ein Proz. berechnet hat;
L_{RI} Restintervall (Last) das abgegeben wird.

Die abzugebende Last ergibt sich dann zu:

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

c kann man als Zurstückelungsfaktor ansehen. Wir untersuchten den Einfluß verschiedener Werte von c auf das Verhalten mehrerer Parallel-TRAPEX-Varianten.

Das Farming-Modell wurde daher so modifiziert, daß jeder Slave gegebenenfalls fortlaufend Teile L_{RI} seines von ihm noch nicht berechneten Subintervalles wieder an den Master abgeben kann. Der Master verteilt diese zurückerhaltenen Intervallstücke auf nicht beschäftigte Slaves, bzw. legt sie auf einen Stack. Erreicht der Stack eine bestimmte Länge, werden weitere Slaves initiiert. Gibt es dagegen zu viele nichtbeschäftigte Prozessoren, können deren Rechenprozesse (durch den Master) termiert werden. Die Zahl der benötigten Prozessoren (die Rechenleistung) kann damit dynamich dem zu lösenden Problem angepaßt werden. Über die Problemdaten wird die Zahl der parallelen Prozesse gesteuert. Diese Form der Parallelisierung sei mit "vertikaler" Parallelisierung bezeichnet.

Weitere Möglichkeiten ("horizontale" Paralelisierung) lassen sich aus Bild 1 bzw. der Formel (3) ableiten. Aus Formel (3) ergibt sich ein Extrapolatiionstableau in Dreiecksform. Zur Berechnug eines $T_{i,k}$ benötigt man ein $T_{i-1,k-1}$ und ein $T_{i,k-1}$. Berechnet man die $T_{i,\emptyset}$ sequentiell, ist auch die Berechnung von $T_{i,k}$ inhärent sequentiell. Da aber bei diesem Verfahren für jede Subintervall-Berechnung (basic step) eine optimale Ordnung i prognostiziert wird und dann in einem Fenster i-1 bis i+1 Konvergenzprüfungen vorgenommen werden, können auch parallel alle Funktionswerte, die für die Ordnung i bzw. i+1 benötigt werden, berechnet werden. Danach ist die parallele Berechnug der $T_{1,\emptyset}$ bis $T_{i,\emptyset}$ bzw. $T_{i+1,\emptyset}$ möglich und damit die parallele Berechnug aller

$T_{j,k}$ $0 \leq j \leq i_{opt}$ einer Spalte k. Für ein $i_{opt}=7$ bedeutet dies, daß 17 Funktionswerte (!) parallel berechnet werden können. Danach werden aber nur noch 7 Prozessoren zur parallelen Berechnug der $T_{1,0}$ bis $T_{7,0}$ benötigt. Außerdem wird für jedes k $1 \leq k \leq 7$ beim Übergang von k auf k+1 ein weiterer Prozessor nicht mehr benötigt. Dieser Aufwand läßt sich nur rechtfertigen, wenn die Inkarnation und die Freigabe von Tasks "billig" ist bzw. in einem Multitasking- Multiuser-System Prozessoren auch anderweitig eingesetzt werden können und bei der Verwaltung der Idle-Tasks nur der Aufwand für die Verwaltung in den Warteschlangen zu Buche schlägt.
Eine ähnliche Problematik tritt auf, wenn nur die Funktionswerte, die für eine Ordnung i benötigt werden, parallel berechnet werden. Aus Bild 1 sieht man, daß dann maximal (für die Ordnungen 6 und 7) 4 Prozessoren parallel arbeiten können. Mit Hilfe von Präzedenzrelationen kann man nachweisen, daß die parallele Berechnung der Extrapolationen der Ordnung i und die Berechnug der Funktionswerte der Ordnung i+1 möglich ist. Der Rechenaufwand zur Berechnung eines $T_{k,k}$ steigt proportional mit der Ordnung k, sodaß für höhere Ordnungen die Wahrscheinlichkeit steigt, daß die Funktionswerte für die Ordnung k+1 vorliegen nachdem $T_{k,k}$ berechnet ist,.

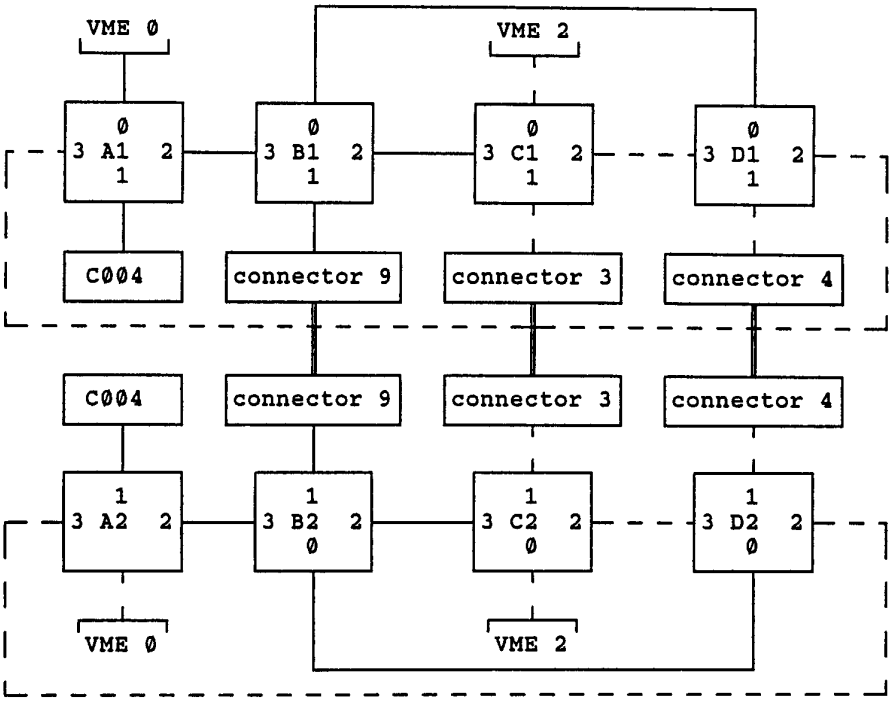
Implementierung

Für die Implementierung des Algorithmus auf gitterartigen Nearest-Neighbour- Architekturen und in OCCAM2, wie sie mit Transputern möglich sind, muß man Kommunikationsprozesse vorsehen, wenn mehr als 5 parallele Rechenprozesse implementiert werden sollen. Diese Kommunikationsprozesse ermöglichen gleichzeitig ein (logisch) asynchrones Message-Passing. Das synchrone Message-Passing der Transputer-Links würde sonst nichtproblembedingte Verzögerungen heraufbeschwören. Da im Transputer zwei Prioritätsstufen hardwaremäßig vorgesehen sind, kann ein (priorisierter) Masterprozeß und ein Rechenprozeß (TRAPEX0) auf demselben Transputer laufen. Über die 4 Links [Bild 2] können 4 weitere TRAPEX-Prozesse direkt mit dem Master kommunizieren. Wird ein weiterer TRAPEX-Prozeß hinzugefügt, wird die Kommunikationslänge grösser 1. Auf allen Zwischenknoten sind (priorisierte) Kommunikationsprozesse implementiert [Bild 3] , die (logisch asynchron) das Message-Passing übernehmen und damit die Rechenprozesse

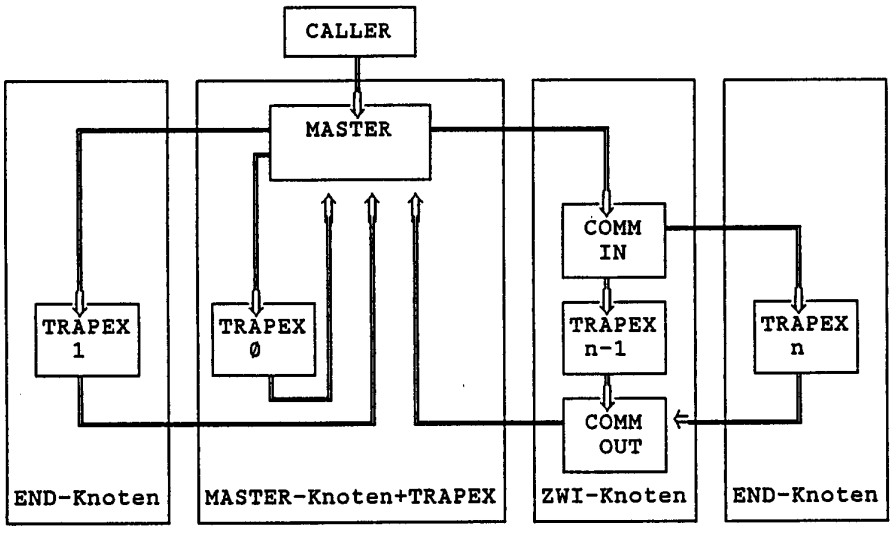
vom Message-Handling befreien.
Der Kommunikatiosprozeß auf dem MASTER-Prozessor hat folgende Struktur:

```
busy :=  TRUE
WHILE busy
  ALT
    in0 ? inbuf0
      process.input ()
    in1 ? inbuf1
      process.input ()
    in2 ? inbuf2
      process.input ()
    in3 ? inbuf3
      process.input ()
    in4 ? inbuf4
      process.input ()
    inmaster ? inbufm
      process.master.input ()
```

Dabei sind IN0 bis IN4 die Eingabekanäle, über die die TRAPEX-Tasks Daten senden und INMASTER ein Kanal, über den der TRAPEXMASTER seine Bereitschaft zur Entgegennahme von Daten signalsiert sowie ein Terminationssignal an den Kommunikationsprozeß absetzt. Diese Kanäle sind ebenso wie Variable im OCCAM-Programm zu deklarieren, bevor sie erwendet werden können. Den Kanälen IN0 und INMASTER wird kein Link zugeordnet, da diese zur Kommunikation zwischen Tasks auf demselben Prozessor dienen; solche Kanäle heißen auch "Soft-Channnels". Der eigentliche Empfangsprozeß besteht im Füllen eines Eingabepuffers; die ALT-KontrollStruktur verlangt für jede Komponente außerdem einen weiteren Prozeß, das ist derjenige, der nach Beendigung der Datenübertragung aktiviert wird. Zu bemerken ist hier, daß ein Prozeß im Sinne von OCCAM als ausführbare Programmeinheit zu verstehen ist, in diesem Sinne sind auch einzelne Anweisungen Prozesse.



[Bild 2] Die *gestrichelten* Linien kennzeichnen von uns *nicht verwendete* Verbindungen



[Bild 3] Datenfluß des parallelem TRAPEX; ab n=6 benötigt man mindestens (n modulo 3)-1 ZWI-Knoten.

Meßergebnisse

Eine ausführliche Darstellung der Meßergebnisse für die vertikale und horizontale Parallelisierung für Beispiele, die schon in [MaSk88] verwendet werden, sind in einem demnächst im ZIB erscheinenden Technical Report (Implementierung des parallelen TRAPEX auf Transputern) enthalten. Es wurden 5 bis 7 Transputer verwendet und damit überwiegend eine Effizienz zwischen 50 - 100% erreicht. Zum Teil sogar darüber hinaus, was in unterschiedlichen H_k^j-Folgen für die parallele bzw. die sequentielle Implementierung begründet liegt. Für die Messungen wurde eine Worst-Case-Situation angenommen, indem immer als maximal zulässiges H die Länge des Startintervalles angesetzt wird. Da bei der parallelen Version die Startintervalle immer kleiner sind als bei der sequentiellen Version, kann dies zu Effizienzsteigerungen über 100% führen.

Vertikale Parallelisierung

Der TRAPEXMMASTER läuft auf Transputer B1 (erstes Board) [Bild 2]; er wird vom CALLER auf dem Host-Transputer mit den Start-Daten versorgt und liefert an diesen zum Schluß die Ergebnisse. Auf den übrigen Transputern und auf dem Host-Transputer läuft jeweils eine TRAPEX-Task, weil der CALLER nach der initialen Datenübertragung an den TRAPEXMASTER bis zur Entgegennahme der Ergebnisse inaktiv ist. Als Vergleich sind auch die Meßwerte für die sequentielle Version (Spalte SEQ) aufgeführt.

TPARC8: mit Kommunikationsprozeß und TRAPEXASTER auf Prozessor 0, ein Transfer-Knoten auf zweitem Board

Bsp	SEQ t [μs]	2 * TRAPEX t [μs]	ACC	3 * TRAPEX t [μs]	ACC	4 * TRAPEX t [μs]	ACC
1	11939	8853	1.35	8792	1.36	8020	1.49
9	353334	143296	2.47	97762	3.61	87221	4.05
13	511875	243305	2.10	155758	3.29	h = 0	-

Bsp	SEQ t [μs]	2 * TRAPEX t [μs]	ACC	3 * TRAPEX t [μs]	ACC	4 * TRAPEX t [μs]	ACC
17	472154	264602	1.78	194822	2.55	166770	2.83
20	49766	22274	2.23	19438	2.56	14158	3.52
21	684812	241604	2.83	172252	3.98	147588	4.64
22	238592	90821	2.63	63496	3.76	44912	5.31

Bsp.	5 * TRAPEX t [μs]	ACC	6 * TRAPEX t [μs]	ACC	7 * TRAPEX t [μs]	ACC
1	8115	1.47	8080	1.48	8094	1.48
9	77722	4.55	58248	6.07	53119	6.65
13	172897	2.96	170999	2.99	188392	2.72
17	147115	3.21	106406	4.44	115186	4.10
20	10666	4.66	8984	5.54	8342	5.97
21	144003	4.76	89059	7.67	81535	8.40
22	48840	4.89	36621	6.52	35996	6.63

Literatur

[Bau83] H.-J. Bauer: Entwicklung leistungsfähiger Extrapolationscodes;Diplomarbeit,Universität Heidelberg 1983

[BeEs88] Berntsen/Espelid: A Parallel Global Adaptiv Quadrature Algorithm for Machines with Hypercube Architecture; Depart.of Informatics University of Bergen;Bergen Norway; Rep. Nr. 27 March 27th 1987

[Bu64] R.Bulirsch: Bemerkungen zur Romberg-IntegrationNumer. Math.6, 6-16 (1964)

[De80] P.Deuflhard: Order and Stepsize Control in Extrapolation Methods Preprint Nr.: 93; Universität Heidelberg 1980

[DeBa82] P.Deuflhard, H.J.Bauer: A Note on Romberg Quadrature, Preprint Nr.: 169; Universität Heidelberg, July 1982

[En80] H. Engel: Numerical Quadrature and Cubature; London Academic Press 1980

[Ka71] D. Kahaner: Comparison of Numerical Quadrature Formulas;in 2. Rice (Ed.): Mathematical Software; London,Academic Press 1971

[MaSk88] Maierhöfer/Skorobohatyj: Ein paralleler adaptiver Algorithmus zur numerischen Integration; seine Implementierung für SUPRENUM- artige Architekturen mit SUSI; ZIB TR88-5 Okt. 1988

[Ro55] W. Romberg: Vereinfachte Numerische Integration. Det Koneglige Norske Videnskabs Forthandlinger; Vol. 28

Veröffentlichungen des Konrad-Zuse-Zentrum für Informationstechnik Berlin
Preprints November 1989

SC 86-1. P. Deuflhard; U. Nowak. *Efficient Numerical Simulation and Identification of Large Chemical Reaction Systems.* (vergriffen) In: Ber. Bunsenges. Phys. Chem., vol. 90, 1986, 940-946
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) In: Numerische Mathematik, 52, 1988, 427-458.
SC 87-3. Peter Deuflhard. *Uniqueness Theorems for Stiff ODE Initial Value Problems.*
SC 87-4. Rainer Buhtz. *CGM-Concepts and their Realizations.*
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.* (vergriffen)
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.* (vergriffen)
SC 88-7. H. Melenk; H. M. Möller; W. Neun. *Symbolic Solution of Large Stationary Chemical Kinetics Problems. I*
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-dominierte Randwertprobleme bei partiellen Differentialgleichungen.*
SC 88-10. S -N. Chow; B. Deng; B. Fiedler. *Homoclinic Bifurcation at Resonant Eigenvalues.*

SC 89-1. Hongyuan Zha. *A Numerical Algorithm for Computing the Restricted Singular Value Decomposition of Matrix Triplets.*
SC 89-2. Hongyuan Zha. *Restricted Singular Value Decomposition of Matrix Triplets.*
SC 89-3. Wu Huamo. *On the Possible Accuracy of TVD Schemes.*
SC 89-4. H. Michael Möller. *Multivariate Rational Interpolation: Reconstruction of Rational Functions.*
SC 89-5. Ralf Kornhuber; Rainer Roitzsch. *On Adaptive Grid Refinement in the Presence of Internal or Boundary Layers.*
SC 89-6. Wu Huamo; Yang Shuli. *MmB-A New Class of Accurate High Resolution Schemes for Conservation Laws in Two Dimensions.*
SC 89-7. U. Budde; M. Wulkow. *Computation of Molecular Weight Distributions for Free Radical Polymerization Systems.*
SC 89-8. Gerhard Maierhöfer. *Ein paralleler adaptiver Algorithmus für die numerische Integration.*

Veröffentlichungen des Konrad-Zuse-Zentrum für Informationstechnik Berlin
Technical Reports November 1989

TR 86-1. H. J. Schuster. *Tätigkeitsbericht (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 3.3*
TR 87-5. Herbert Melenk; Winfried Neun. *REDUCE Users Guide for the CRAY 1 / X-MP Series Running COS. Version 3.3*
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 / CRAY 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. *Parallel-TRAPEX. Ein paralleler, adaptiver Algorithmus zur numerischen Integration ; seine Implementierung für SUPRENUM-artige Architekturen mit SUSI.*

TR 89-1. *CRAY-HANDBUCH. Einführung in die Benutzung der CRAY X-MP unter UNICOS.*
TR 89-2. Peter Deuflhard. *Numerik von Anfangswertmethoden für gewöhnliche Differentialgleichungen.*
TR 89-3. Artur Rudolf Walter. *Ein Finite-Element-Verfahren zur numerischen Lösung von Erhaltungsgleichungen.*
TR 89-4. Rainer Roitzsch. *KASKADE User's Manual.*
TR 89-5. Rainer Roitzsch. *KASKADE Programmer's Manual.*
TR 89-6. Herbert Melenk; Winfried Neun. *Implementation of Portable Standard LISP for the SPARC Processor.*
TR 89-7. Folkmar A. Bornemann. *Adaptive multilevel discretization in time and space for parabolic partial differential equations.*