Untersuchungen zur Parallelverarbeitung mit wissenschaftlich-technischen Berechnungsumgebungen

René Fink

13. Dezember 2007

Gliederung

1. Einleitung
2. Wissenschaftlich-technische Berechnungsumgebungen (SCEs*)
3. Parallelverarbeitung
4. Parallelverarbeitung mit SCEs
5. Analyse von Multi-SCE-Systemen
6. Weiterentwicklung eines Multi-SCE-Systems
7. Anwendungsorientierte Untersuchungen
8. Zusammenfassung

*Scientific and Technical Computing Environment

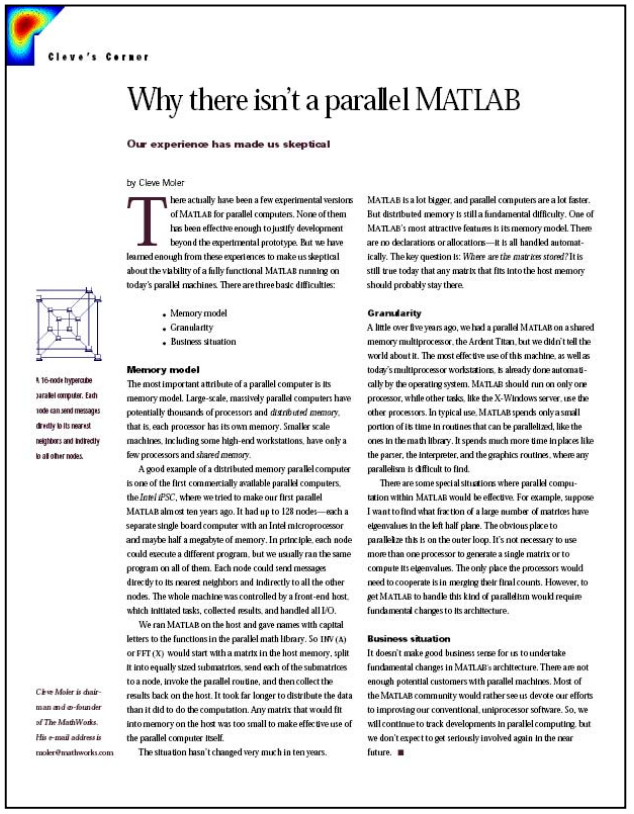
Gliederung

1. Einleitung
2. Wissenschaftlich-technische Berechnungsumgebungen
3. Parallelverarbeitung
4. Parallelverarbeitung mit SCEs
5. Analyse von Multi-SCE-Systemen
6. Weiterentwicklung eines Multi-SCE-Systems
7. Anwendungsorientierte Untersuchungen
8. Zusammenfassung

- SCE: Werkzeug für Ingenieure
- Vertreter: Matlab, Scilab, Octave
- schnelle Entwicklung wissenschaftlich-technischer Programme
- interpretative Entwicklung, Ausführung und Auswertung
- Nachteil: geringe Ausführungsgeschwindigkeit
- Beschleunigung durch Parallelverarbeitung → Zeitgewinn
- SCE-basierte Parallelverarbeitung bisher kaum verbreitet

1 Einleitung
Vorarbeiten zur Thematik

- 80er Jahre: experimentelle parallele Matlab-Versionen
- 1995: DP-Toolbox
- 1995/1996: MultiMATLAB und PT Toolbox
- 1998: Dissertation Pawletta, Multi-SCE-Ansatz
- seit 1998: weiterschreitende Entwicklung, u.a. Matlab DC-Toolbox



Cleve's Corner

Why there isn't a parallel MATLAB

Our experience has made us skeptical

by Cleve Moler

There actually have been a few experimental versions of MATLAB for parallel computers. None of them has been effective enough to justify development beyond the experimental prototype. But we have learned enough from these experiences to make us skeptical about the viability of a fully functional MATLAB running on today's parallel machines. There are three basic difficulties:

- Memory model
- Granularity
- Business situation

Memory model

The most important attribute of a parallel computer is its memory model. Large-scale, massively parallel computers have potentially thousands of processors and *distributed memory*, that is, each processor has its own memory. Smaller scale machines, including some high-end workstations, have only a few processors and *shared memory*.

A good example of a distributed memory parallel computer is one of the first commercially available parallel computers, the *Intel iPSC*, where we tried to make our first parallel MATLAB almost ten years ago. It had up to 128 nodes—each a separate single board computer with an Intel microprocessor and maybe half a megabyte of memory. In principle, each node could execute a different program, but we usually ran the same program on all of them. Each node could send messages directly to its nearest neighbors and indirectly to all the other nodes. The whole machine was controlled by a front-end host, which initiated tasks, collected results, and handled all I/O.

We ran MATLAB on the host and gave names with capital letters to the functions in the parallel math library. So INV(A) or FFT(X) would start with a matrix in the host memory, split it into equally sized submatrices, send each of the submatrices to a node, invoke the parallel routine, and then collect the results back on the host. It took far longer to distribute the data than it did to do the computation. Any matrix that would fit into memory on the host was too small to make effective use of the parallel computer itself.

The situation hasn't changed very much in ten years. MATLAB is a lot bigger, and parallel computers are a lot faster. But distributed memory is still a fundamental difficulty. One of MATLAB's most attractive features is its memory model. There are no declarations or allocations—it is all handled automatically. The key question is: *Where are the matrices stored?* It is still true today that any matrix that fits into the host memory should probably stay there.

Granularity

A little over five years ago, we had a parallel MATLAB on a shared memory multiprocessor, the Ardent Titan, but we didn't tell the world about it. The most effective use of this machine, as well as today's multiprocessor workstations, is already done automatically by the operating system. MATLAB should run on only one processor, while other tasks, like the X-Windows server, use the other processors. In typical use, MATLAB spends only a small portion of its time in routines that can be parallelized, like the ones in the math library. It spends much more time in places like the parser, the interpreter, and the graphics routines, where any parallelism is difficult to find.

There are some special situations where parallel computation within MATLAB would be effective. For example, suppose I want to find what fraction of a large number of matrices have eigenvalues in the left half plane. The obvious place to parallelize this is on the outer loop. It's not necessary to use more than one processor to generate a single matrix or to compute its eigenvalues. The only place the processors would need to cooperate is in merging their final counts. However, to get MATLAB to handle this kind of parallelism would require fundamental changes to its architecture.

Business situation

It doesn't make good business sense for us to undertake fundamental changes in MATLAB's architecture. There are not enough potential customers with parallel machines. Most of the MATLAB community would rather see us devote our efforts to improving our conventional, uniprocessor software. So, we will continue to track developments in parallel computing, but we don't expect to get seriously involved again in the near future. ■

6. 16-node hypercube parallel computer. Each node can send messages directly to its nearest neighbors and indirectly to all other nodes.

Cleve Moler is chairman and co-founder of The MathWorks. His e-mail address is moler@mathworks.com

- Ziel: stärkere Etablierung der SCE-basierten Parallelverarbeitung in ingenieurtechnischen Bereichen
- Methoden:
  - Weiterführung der Arbeiten von Pawletta
  - Systematisierung von Ansätzen
  - Entwicklung von Vergleichskriterien
  - Weiterentwicklung von Systemen
  - Gewinnung anwendungsorientierter Erkenntnisse

Gliederung

1. Einleitung
2. Wissenschaftlich-technische Berechnungsumgebungen
3. Parallelverarbeitung
4. Parallelverarbeitung mit SCEs
5. Analyse von Multi-SCE-Systemen
6. Weiterentwicklung eines Multi-SCE-Systems
7. Anwendungsorientierte Untersuchungen
8. Zusammenfassung

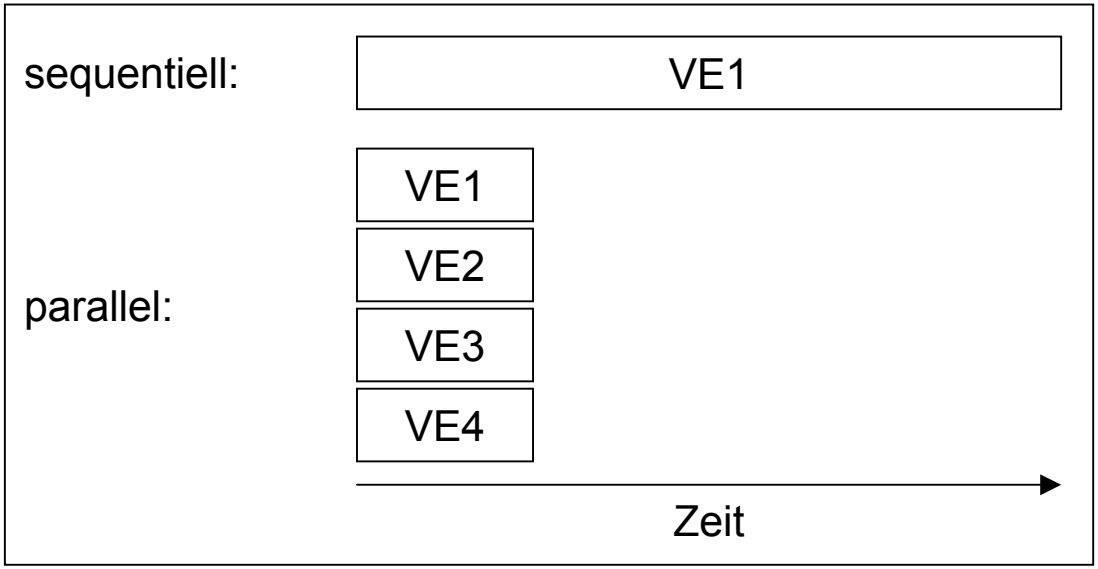
- Scientific and Technical Computing Environment
- Alternative Begriffe:
  - Matlab-like System
  - Computer Numerical System
  - Problem Solving Environment
- Merkmale:
  - interpretative Arbeitsweise
  - datenparallele Programmiersprache
  - integrierte Numerikbibliotheken
  - integrierte Visualisierungsfunktionen

SCE	Lizenz	veröff.
IDL	kommerziell	1977
Gauss	kommerziell	1983
Matlab	kommerziell	1984
Mathematica	kommerziell	1988
O-Matrix	kommerziell	1992
Octave	frei	1993
Rlab	frei	1994
Scilab	frei	1994
Tela	frei	1994
Euler	frei	1996
Yorick	frei	1996

- Codeoptimierung
- Automatische Übersetzung in Maschinencode
- Reimplementierung in compilierbarer Sprache
- SCE-basierte Parallelverarbeitung

Gliederung

- Parallelverarbeitung: zeitgleiche Bearbeitung von Teilproblemen
- Ziel: Problemlösung in kürzerer Zeit



- Gebiete der PV:
  - Architekturen
  - Programmierung
  - Algorithmen

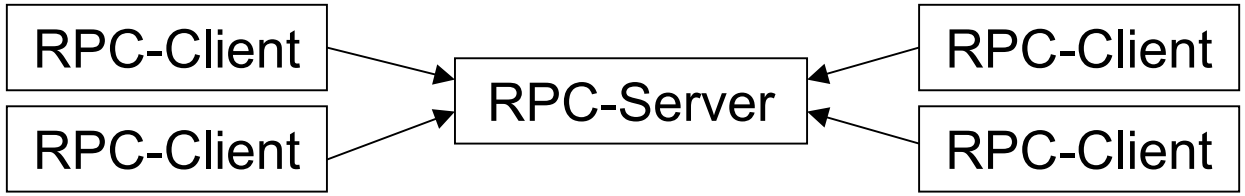
- Aufgaben der Parallelprogrammierung:
  - Zerlegung
  - Instanziierung und Mapping
  - Kommunikation
  - Synchronisation
- Programmiermodell: Schnittstelle zwischen Programmierer und Hardware
- Unterscheidung: implizite/explizite Darstellung der Programmieraufgaben

- Implizites paralleles Programmiermodell: alle Aufgaben werden automatisch erfüllt
  - parallelisierender Compiler
  - Programmierung mit parallelen Bibliotheken
- Explizites paralleles Programmiermodell: einzelne Aufgaben werden durch den Programmierer erfüllt
  - Shared-Memory-Programmierung
  - Message-Passing-Programmierung
  - Remote-Procedure-Call-Programmierung

Programmiermodell	**Zerlegung**	**Mapping**	**Kommunikation**	**Synchronisation**
Shared Memory	explizit	implizit o. explizit	implizit	explizit
Message Passing	explizit	implizit o. explizit	explizit	implizit
RPC	explizit	implizit o. explizit	implizit	implizit

- etablierte parallele Programmiermodelle
- Shared-Memory-Programmierung:
  - Prozesse mit gemeinsamem Speicherbereich
  - Kommunikation implizit über gemeinsamen Speicher
  - Synchronisation explizit
- Message-Passing-Programmierung:
  - Prozesse mit ausschließlich lokalem Speicherbereich
  - Kommunikation explizit
  - Synchronisation implizit durch Kausalordnung

- Ursprung: verteilte Verarbeitung
- Erweiterung lokaler Funktionsaufrufe



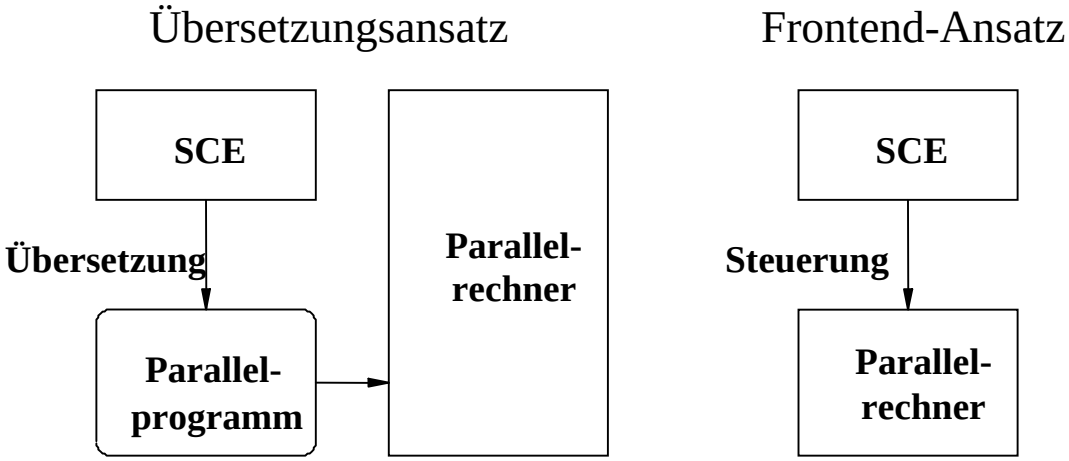
- RPC in Parallelverarbeitung:

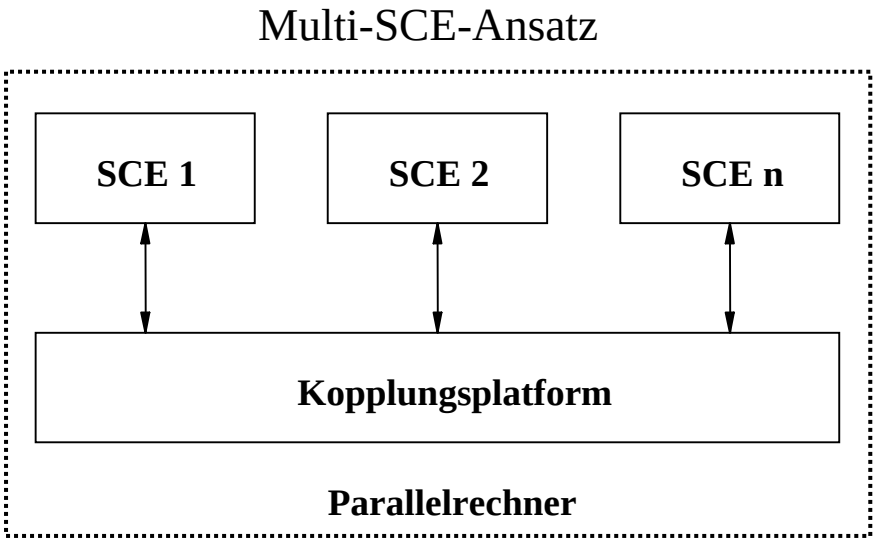
- notwendige RPC-Erweiterungen: asynchrones RPC, vektorielles RPC
- Kommunikation und Synchronisation implizit

Gliederung

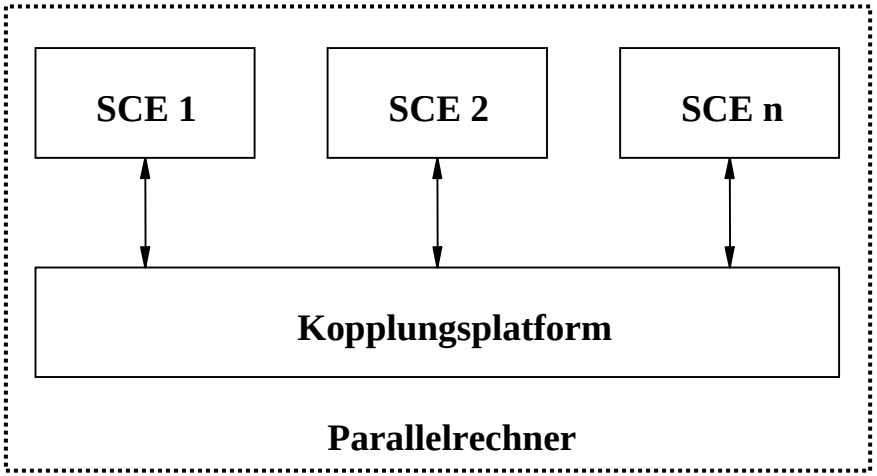
- bekannte Klassifikationen:
  1. Pawletta
  2. Choy, Edelman
  3. Panuganti et al

- neue Klassifikation:
  - Übersetzungsansatz
  - Frontend-Ansatz
  - Multi-SCE-Ansatz





- Übersetzungsansatz:
  - Programmiermodell: parallelisierender Compiler
  - identifizierte Systeme: 4
- Frontend-Ansatz:
  - Programmiermodell: Programmierung mit parallelen Bibliotheken
  - identifizierte Systeme: 3
- Multi-SCE-Ansatz:
  - Programmiermodell: Shared-Memory, Message-Passing, RPC
  - identifizierte Systeme: 28



- dominierender SCE-PV-Ansatz
- überwiegend Matlab-fokussiert
- überwiegend Message-Passing- und RPC-Programmierung
- überwiegend durch Message-Passing-Systeme realisiert
- großes Interesse bei Anwendern
- geeignet für viele ingenieurtechnische Anwendungen

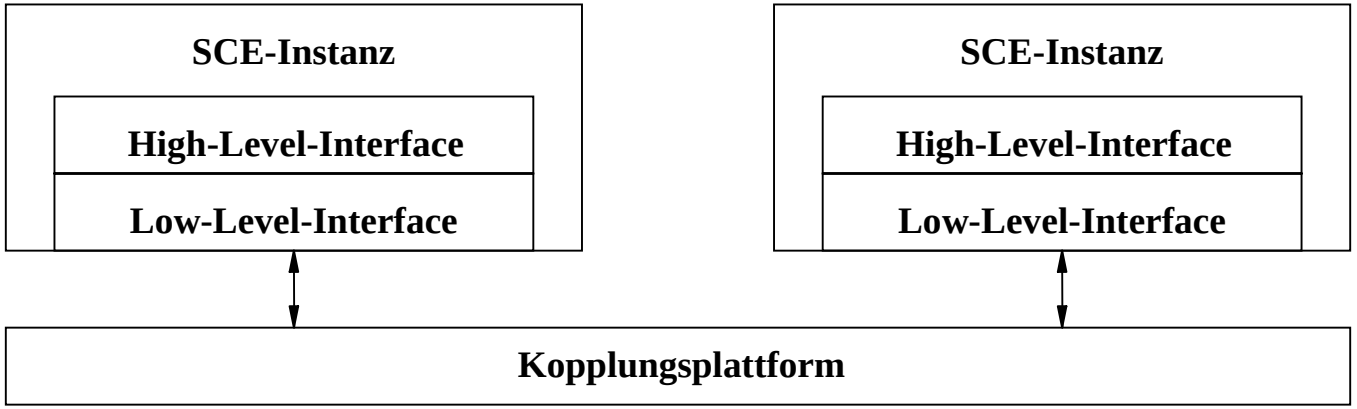
Gliederung

1. Einleitung
2. Wissenschaftlich-technische Berechnungsumgebungen
3. Parallelverarbeitung
4. Parallelverarbeitung mit SCEs
5. Analyse von Multi-SCE-Systemen
6. Weiterentwicklung eines Multi-SCE-Systems
7. Anwendungsorientierte Untersuchungen
8. Zusammenfassung

5 Analyse von Multi-SCE-Systemen
Einleitung

- Vergleich von 8 Multi-SCE-Systemen
- Analyse qualitativer Merkmale
- Untersuchung der Kommunikationsleistung

System	SCE	Programmiermodelle
DP-Toolbox v1.5	Matlab v7.1	Message-Passing
MatlabMPI v1.2	Matlab v7.1	Message-Passing
MPI Toolbox	Matlab v7.1	Message-Passing
Beolab Toolbox	Matlab v7.1	RPC
Parallelization Tk. v1.2	Matlab v7.1	RPC
DC-Toolbox V2.0	Matlab v7.1	Message-Passing, RPC
MPI Toolbox	Octave v2.1	Message-Passing
PVM Toolbox	Scilab v2.7	Message-Passing



- Identifikation SCE-spezifischer Message-Passing-Erweiterungen:
  - Array-Passing
  - Nachrichtenspezifikation durch Stringparameter
  - Array-Scattering und –Gathering
  - SCE-Reduktion
- Identifikation von RPC-Erweiterungen

- Messung der Roundtrip-Zeit
- Adaption des Verfahrens für RPC- und SHM-Programmierung
- Ermittlung von Latenzzeit und Übertragungsrate

System	Kopplungsplattform	Latenzzeit	Übertragungsrate
MPI Toolbox (Matlab)	LAM MPI v7.1	0.09 ms	36.7 MB/s
MPI Toolbox (Octave)	LAM MPI v7.1	0.13 ms	37.2 MB/s
PVM Toolbox (Scilab)	PVM v3.4	0.16 ms	12.0 MB/s
DC-Toolbox (Matlab)	MPICH2 v1.0	0.33 ms	34.1 MB/s
DP-Toolbox (Matlab)	PVM v3.4	1.06 ms	11.9 MB/s
Parallelization Tk. (Matlab)	Matlab engine	25.34 ms	37.2 MB/s
MatlabMPI (Matlab)	Dateisystem	44.77 ms	25.2 MB/s
Beolab Toolbox (Matlab)	Matlab engine	81.78 ms	37.1 MB/s
DC-Toolbox (Matlab)	proprietär	535.23 ms	2.9 MB/s

Gliederung

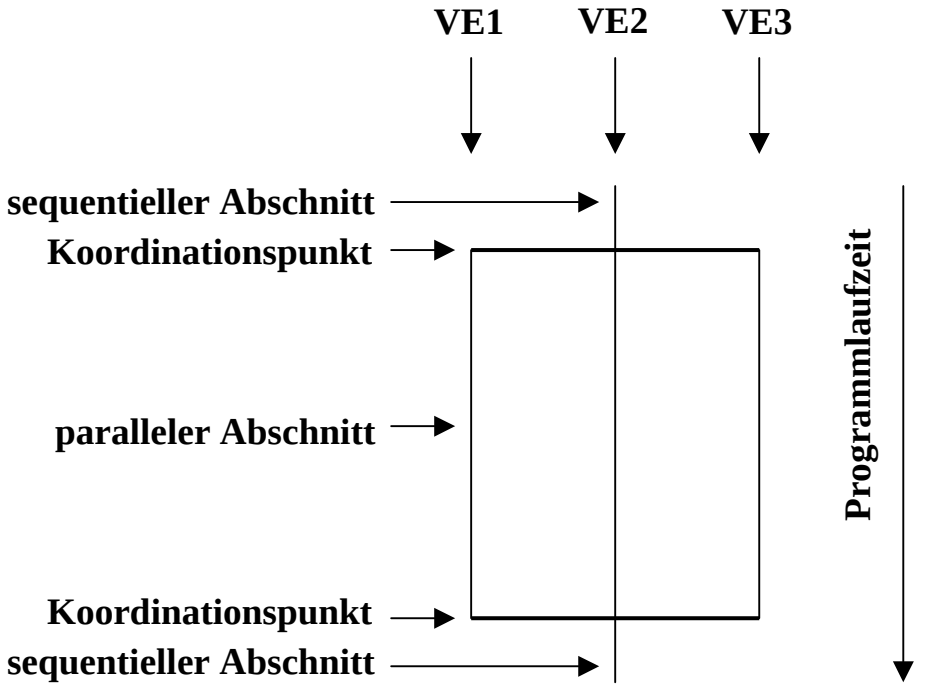
DPMM	**Parallele Matlab-Maschine**				
DPHIGH	**Array-Passing**		**Shared-Arrays**		**Vektorielle RPCs**
DPLOW	**m2pvm**	**m2mpi**	**m2sock**	**m2svipc**	**m2eng**
Kopplungs-plattformen	**PVM**	**MPI**	**Sockets**	**SVIPC**	**Matlab engine**

- Erweiterung des DP-Toolbox-Sets in zwei Richtungen:
  - stabiles System für ingenieurtechnische Anwendungen
  - experimentelle Systeme zu Forschungs- und Lehrzwecken

- DP-1.7:
  - Weiterentwicklung der DP-Toolbox v1.4
  - synchron vektorielle RPC-Programmierung
  - Message-Passing-Programmierung
- DP-MPI:
  - stabile Matlab-MPI-Anbindung
  - Message-Passing-Programmierung
  - Shared-Memory-Programmierung
- weitere Prototypen

Gliederung

- Voraussetzung für Etablierung SCE-basierter PV:
  - geringer Parallelisierungsaufwand
  - hoher Laufzeitgewinn
- beeinflussende Anwendungsmerkmale:
  - Struktur des Ablaufverhaltens
  - Granularität
  - Anwendbarkeit von Programmiermodellen

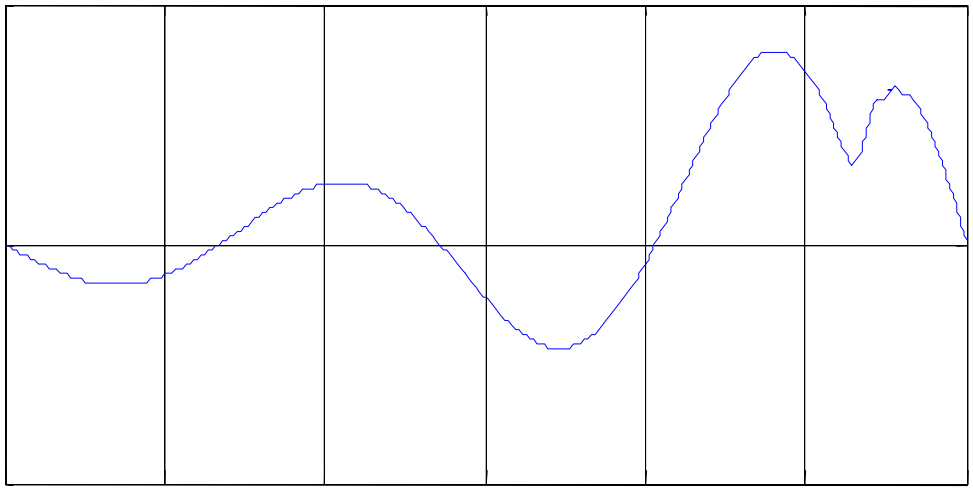


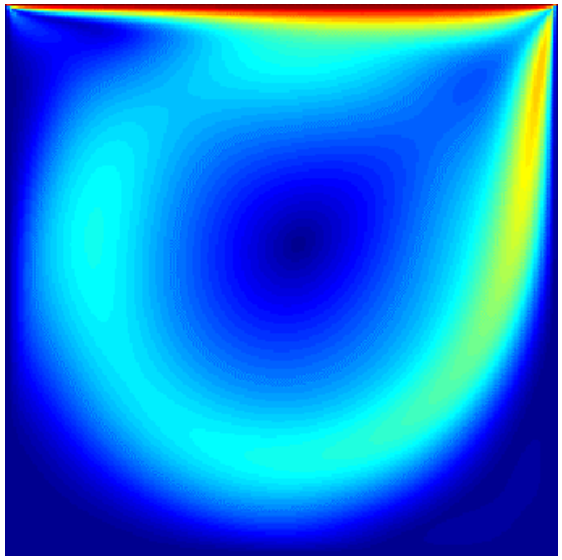
- untersuchte Multi-SCE-Systeme:
  - DC-Toolbox v2.0
  - DP-Toolbox v1.7
  - DP-MPI

- Vergleichskriterien:
  - Parallelisierungsaufwand
  - Laufzeitgewinn

7 Anwendungsorientierte Untersuchungen
Anwendungen

- Parameterstudie eines Feder-Masse-Systems
- Simulation gekoppelter Räuber-Beute-Systeme
- Numerisches Lösen einer partiellen Differentialgleichung
- Strömungssimulation mittels Lattice-Boltzmann-Verfahren
- Parameteroptimierung eines Abgasmodells
- Sicherheitstest eingebetteter Steuerungssoftware





- geringe Unterschiede im Laufzeitgewinn
- große Unterschiede beim Parallelisierungsaufwand
- direkter Vergleich der Anwendungen:
  - geringer Aufwand und hoher Gewinn bei grobgranularen RPC-fähigen Anwendungen
  - hoher Aufwand bei nicht RPC-fähigen Anwendungen
- Voraussetzung für RPC-Anwendung: einfache Programmstrukturen
- Message-Passing oft günstiger bei komplexen Programmstrukturen

Gliederung

8 Zusammenfassung
Durchgeführte Arbeiten

- Entwicklung einer neuen Klassifikation zur SCE-basierten PV
- Einordnung von Ansätzen und Softwaresystemen
- Fokussierung auf den Multi-SCE-Ansatz
- Analyse von 8 Multi-SCE-Systemen
- Weiterentwicklung eines Multi-SCE-Systems
- anwendungsorientierte Untersuchung von 3 Multi-SCE-Systemen

8 Zusammenfassung
Wesentliche Ergebnisse

- Multi-SCE-Ansatz unter Matlab dominierend in SCE-basierter PV
- Message-Passing-Erweiterungen in Multi-SCE-Systemen
- hohe Kommunikationsleistung mit Message-Passing-Bibliotheken
- geringe Unterschiede im Laufzeitverhalten
- große Unterschiede im Parallelisierungsaufwand
- vektorielles RPC geeignet für bestimmte Anwendungstypen
- bei komplexen Strukturen weniger Aufwand mit Message-Passing

8 Zusammenfassung
Ausblick

- aktuelle Hardwareentwicklung: Multikernprozessoren
- Trend in SCE-basierter PV: Verwendung paralleler Numerikbibliotheken
- Multi-SCE-Ansatz wird für einzelne Windows-Workstations interessant

Vielen Dank