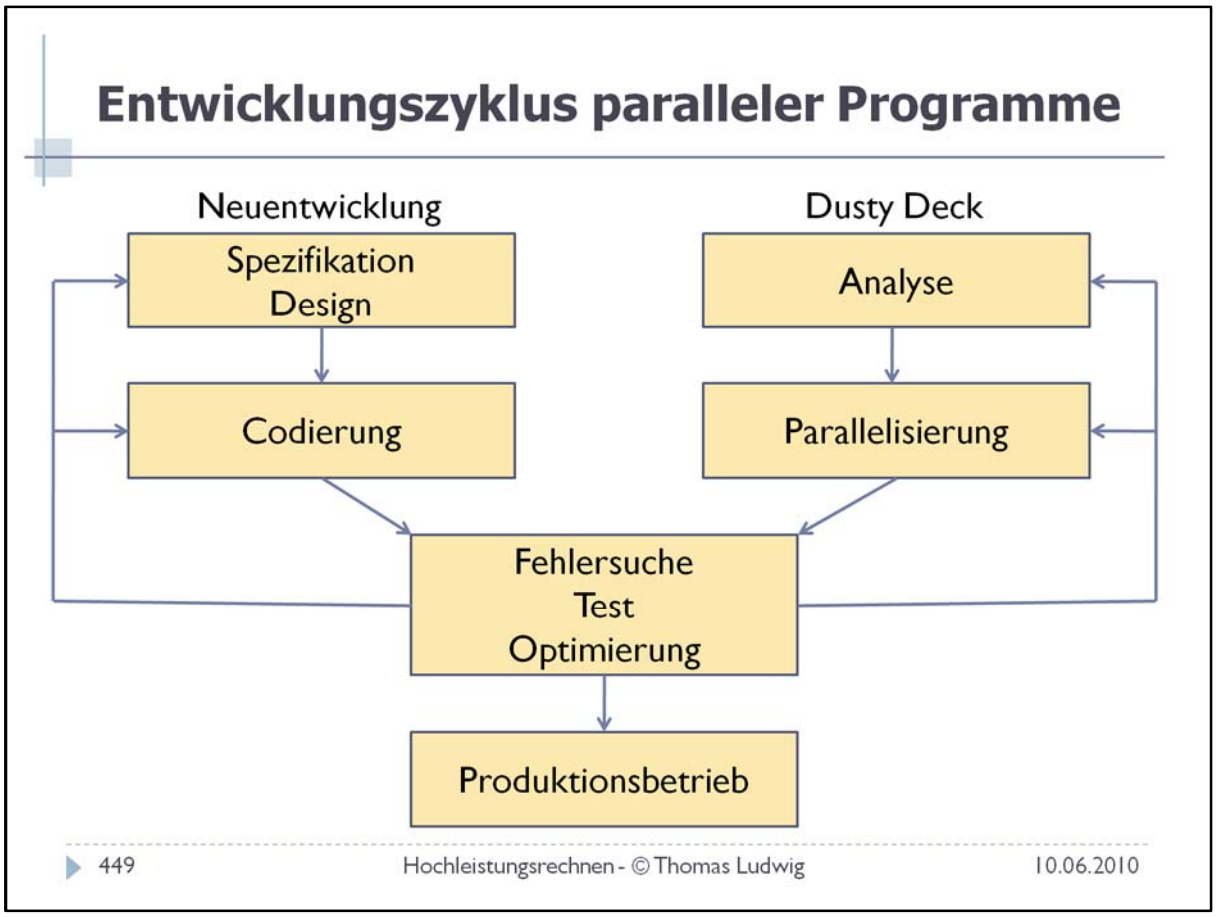
Fehlersuche

- Entwicklungszyklus paralleler Programme
- Fehlersuche
- Häufige Fehlerquellen
- Problemstellungen
- Werkzeugunterstützung
- Laufzeit-Debugger
- Spurbasierte Werkzeuge
- Haltepunktbasierte Werkzeuge
- Konzepte paralleler Debugger
- Ablaufkontrolle und Sicherungspunkte

Siehe: http://en.wikipedia.org/wiki/Debugging

Fehlersuche
Die zehn wichtigsten Fragen

- Welche Schritte umfaßt die Fehlersuche?
- Was sind die häufigsten Fehlerquellen in Programmen?
- Was versteht man unter einer Verklemmung?
- Was versteht man unter einem Überholvorgang?
- Welche Problemstellungen gibt es bei parallelen Prgogrammen?
- Welche Kategorien der Werkzeugunterstützung unterscheiden wir?
- Wie stellen spurbasierte Werkzeuge typischerweise ihre Informationen dar?
- Welche Funktionen bietet ein haltepunktbasierter Debugger?
- Was ist deterministische Ablaufkontrolle und wie funktioniert sie?
- Was ist der Sinn von Sicherungspunkten?



Siehe: http://en.wikipedia.org/wiki/Legacy_code

Als Dusty-Deck (von ´deck´, Lochkartenstapel), oder Legacy-Programme bezeichnet man solche, die schon sehr alt sind, aber immer noch verwendet werden. Die Programmierer sind längst woanders und niemand kennt sich mit dem Code aus. Bei Parallelisierungen gehören die allermeisten Projekte in diese Kategorie.

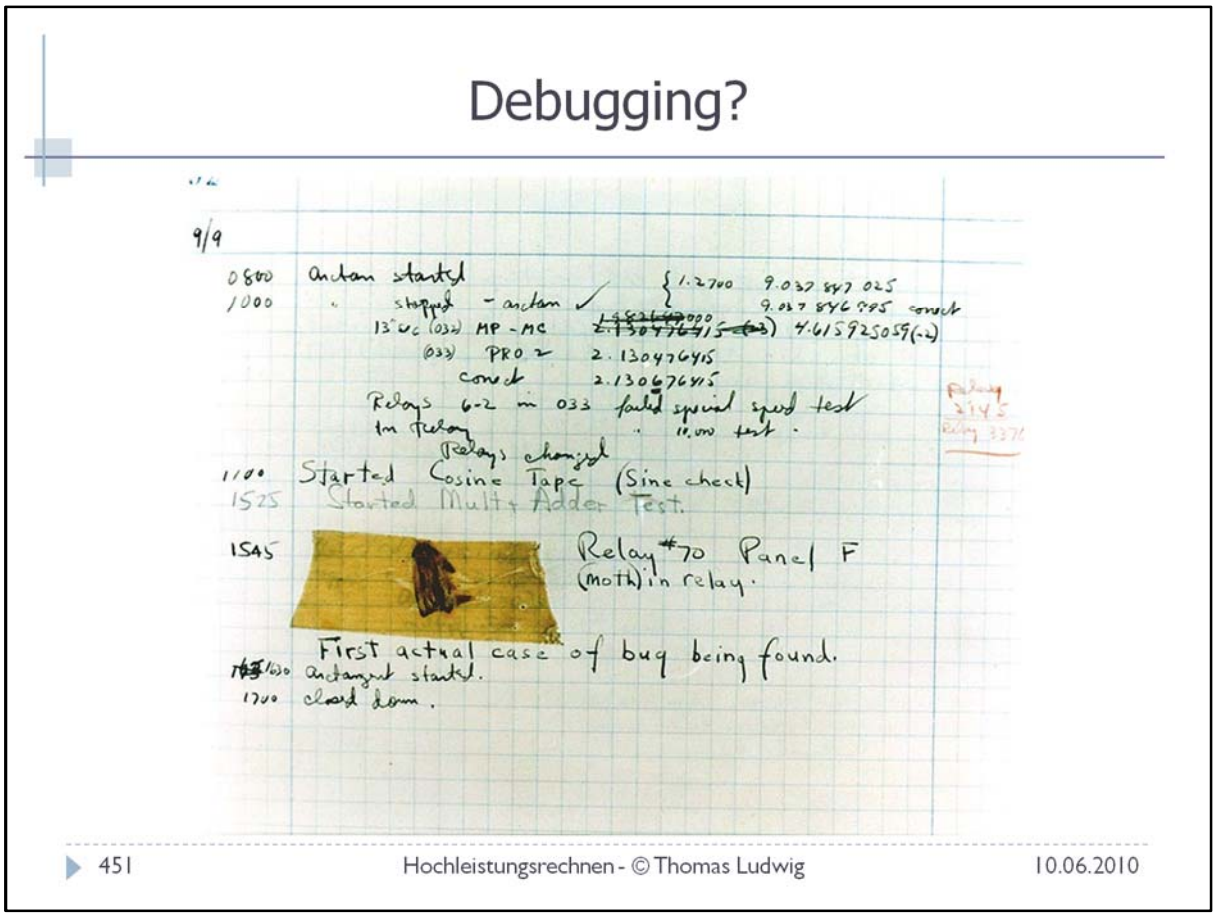
Fehlersuche (Debugging)

Aufspüren von Fehlerzuständen und die Beseitigung ihrer Ursachen

4 Schritte

- Test, Regressionstest
- Erkennen der Fehlerwirkung
- Schließen auf die Fehlerursache
- Beseitigen der Fehlerursache

Siehe: http://en.wikipedia.org/wiki/Regression_test



Quelle: http://en.wikipedia.org/wiki/File:H96566k.jpg#filelinks

The First "Computer Bug" Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1947. The operators affixed the moth to the computer log, with the entry: "First actual case of bug being found". They put out the word that they had "debugged" the machine, thus introducing the term "debugging a computer program".

Beispiel

```
foo (a, b, x, &result);
   /* Ursache: ´&´ vergessen
      Compiler-Warning: pointer from
              integer without a cast */

void foo (int a, int b, int *x,
          int *result)
{  *x = a+b;
   /* segmentation violation */
}
```

Häufigste Fehlerquellen

Sequentielle Programmierung

- Schnittstellenprobleme (Typen, Zeiger auf Parameter, ...)
- Zeiger und dynamische Speicherverwaltung
- Logische und arithmetische Fehler

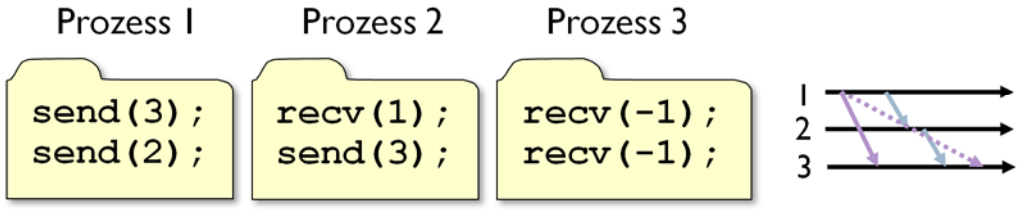
Parallele Programme

- Kommunikationsfehler (Protokolle)
- Überholvorgänge (*races*)
- Verklemmungen (*deadlocks*)

Häufigste Fehlerquellen: Überholvorgänge

Definition: Ein Überholvorgang entsteht durch unsynchronisierte, modifizierende Zugriffe auf gemeinsame Objekte (Adreßbereiche, Nachrichten-puffer)

Beispiel:



Konsequenz: Nichtdeterminismus, Nichtreproduzierbarkeit (schwer feststellbar)

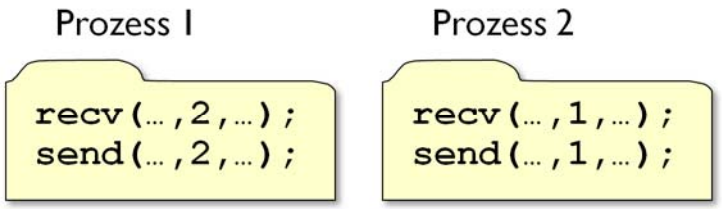
Siehe: http://en.wikipedia.org/wiki/Race_condition

Prozeß 3 empfängt von beliebigen Sender.

Häufigste Fehlerquellen: Verklemmungen

Definition: Bei einer Verklemmung warten Prozesse blockierend auf Ereignisse anderer Prozesse, die auch blockiert sind

Beispiel:



Konsequenz: Programm bleibt hängen (leicht feststellbar)

Siehe: http://en.wikipedia.org/wiki/Deadlock

Kann bei parallelen Programmen leicht passieren, da wir oft nur einen Quellcode verwenden. Der von Prozeß 1 und Prozeß 2 ist also sowieso identisch. Man sieht es aber dem Code nicht so leicht an, da man ihn ja nur einmal vor sich liegen hat und außerdem die Aktualparameter meist Variablen sind.

Problemstellungen

Zusätzlich zu den normalen Problemen der Fehlersuche

- Erkennen einer Fehlerwirkung
- Suchen der Fehlerursache
  - Nichtreproduzierbarkeit der Fehlerwirkung
  - Nichtdeterminismus der Programmausführung
  - Ursache: Zeitabhängigkeit bei der Fehlerursache
- Unübersichtlichkeit: viele Prozesse
- Physische Verteiltheit
- Dynamik: Knoten- und Prozeßmengen variieren potentiell

Erkennen einer Fehlerwirkung

Normalerweise

- Zustand des Programms entspricht nicht der Spezifikation (am Ende / mittendrin)
- Vergleich mit Testdaten

Bei parallelen Programmen

- Ergebnisse nicht nachrechenbar
- Ablauf abhängig von der Prozessorzahl
- Ablauf abhängig von zeitlichen Verhältnissen

=> Falsche Berechnungen schwer erkennbar

Werkzeugunterstützung

- Statische Analyse
- printf() und WRITE
- Laufzeit-Debugger
  - Spurbasierte Werkzeuge
  - Haltepunktbasierte Werkzeuge
- Ablaufkontrolle und Sicherungspunkte

Statische Analyse

Analyse des Programmtextes vor/zur Übersetzungszeit

- Sequentielle Aspekte
  - Strikte Typ- und Parameterprüfung
  - Erweiterte semantische Tests
  - Einsatz spezieller Werkzeuge: lint, insight
  - Gute ANSI-C-Compiler, Option –Wall (alle Warnungen)
- Parallele Aspekte
  - Erkennen möglicher Überholvorgänge
  - Prüfung auf Verklemmungsfreiheit

  = Forschungsthemen (bisher ungelöst)

printf() und WRITE

Die Werkzeuge zur Fehlersuche schlechthin!

Bei parallelen Programmen aber:

- Zuordnung zu einzelnen Prozessen schwierig
  Zeichen- / zeilenweises Mischen möglich
- Bei Netzen: Umleitung in ein gemeinsames Fenster?!
- Sortierung oft unmöglich, da keine globale Zeit
- Korrekte kausale Ordnung der Ausgaben nicht gewährleistet

Laufzeit-Werkzeuge

Automatische Fehlerprüfung zur Laufzeit

Sequentielle Aspekte

- Dynamische Speicherverwaltung

Parallele Aspekte

- Parameterprüfung bei Programmierbibliothek
- *Race*-Erkennung (Forschungsthema)

Vorbereitung der Anwendung (Alternativen)

- Präprozessor und Neuübersetzung
- Binden mit speziell instrumentierter Bibliothek
- Instrumentierung der Binärdatei

Spurbasierte Werkzeuge

Merkmale

- Aufzeichnung relevanter Ereignisse des Programmlaufs
- Betrachtung der Spur durch „Browser“
  offline und auch online möglich
- Im Prinzip: automatisiertes `printf()`

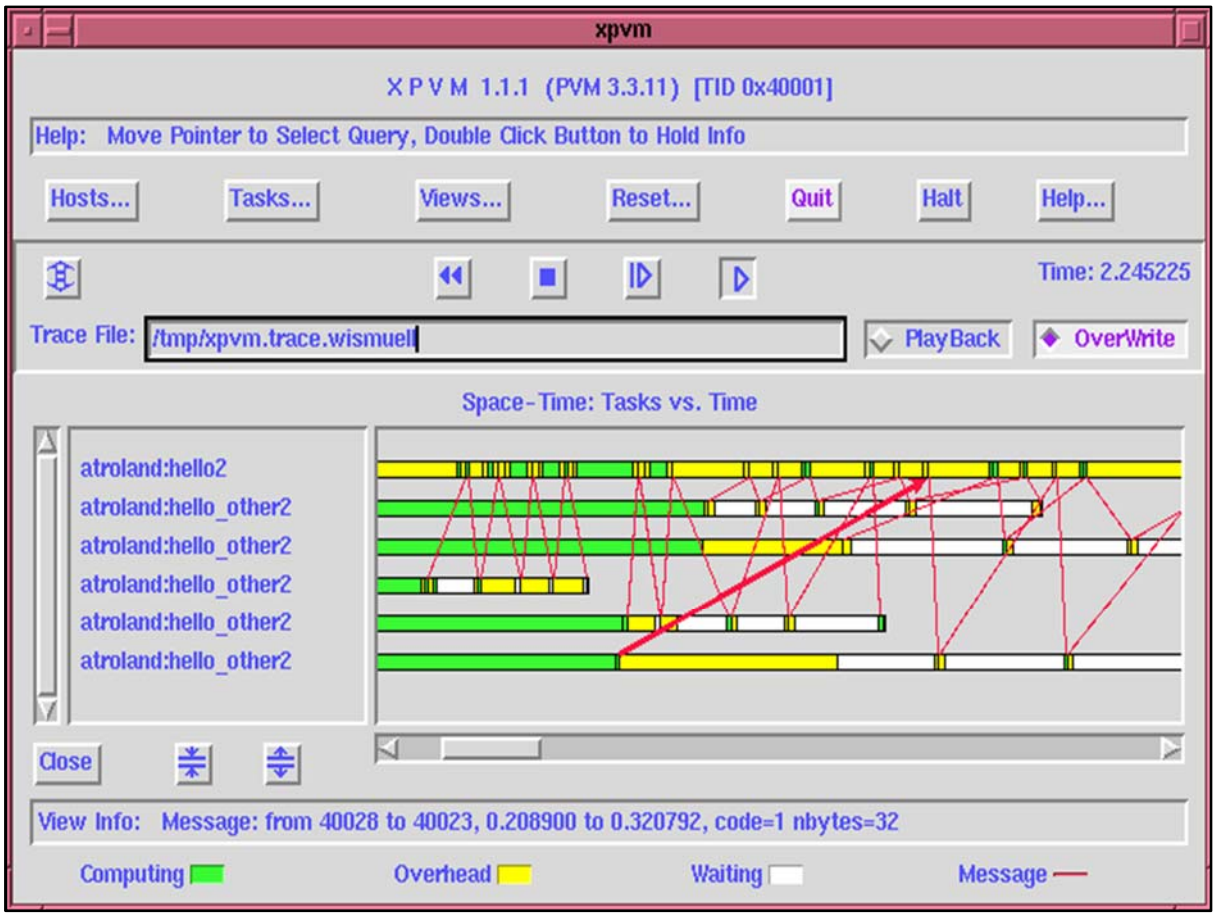
Aufgezeichnete Ereignisse

- Aufruf und Rückkehr der Funktionen der Programmierbibliothek
  Lokale Zeit, Dauer, Parameter
- Zum Teil auch benutzerdefinierte Ereignisse möglich

Spurbasierte Werkzeuge...

Darstellungsarten

- Raum-Zeit-Diagramme, Ganntt-Diagramme
  Darstellung einzelner Prozeßzustände
  Knoten- und/oder prozeßorientierte Darstellung
  Gut: globaler Überblick
  Schlecht: Globale Ordnung meist trügerisch
- Folge von Schnappschüssen
  Darstellung des globalen Zustands zu bestimmten Zeiten



Beispiel: XPVM. In seiner Darstellung sehr typisch für viele andere spurbasierte Werkzeuge.

Spurbasierte Werkzeuge...

Steuerung

- VCR-ähnliche Elemente: Start, Stop, Vor, Zurück, Einzelschritt
- Meist Auswahl relevanter Knoten und Prozesse

Vorbereitung

- Präprozessor und Neuübersetzung
- Binden mit instrumentierter Bibliothek
- Laufzeitoption der Programmierbibliothek

Bewertung

- Für globalen Überblick und zur Überwachung der Kommunikation

Haltepunktbasierte Debugger

Vorgehen

- Anhalten des Programms an interessanten Stellen
- Inspizieren des Programmzustandes
- Fortsetzen (oder Neustart) des Programms

Bei erkanntem Fehler

- Hypothese zur Fehlerursache
- Neustart des Programms und Überprüfen der Hypothese

Haltepunktbasierte Debugger...

Typischer Funktionsumfang

- Anhalten des Programms
  Bedingt und/oder unbedingt
- Inspizieren des Programmzustandes
  Prozeduraufrufkeller, Parameter, Variablen
- Modifikation des Programmzustandes
  Setzen von Variablen, Veränderung des Codes(!)
- Ausführungskontrolle
  - Start und Stop
  - Einzelschritt (Anweisungen, Prozeduren)

Konzepte paralleler Debugger

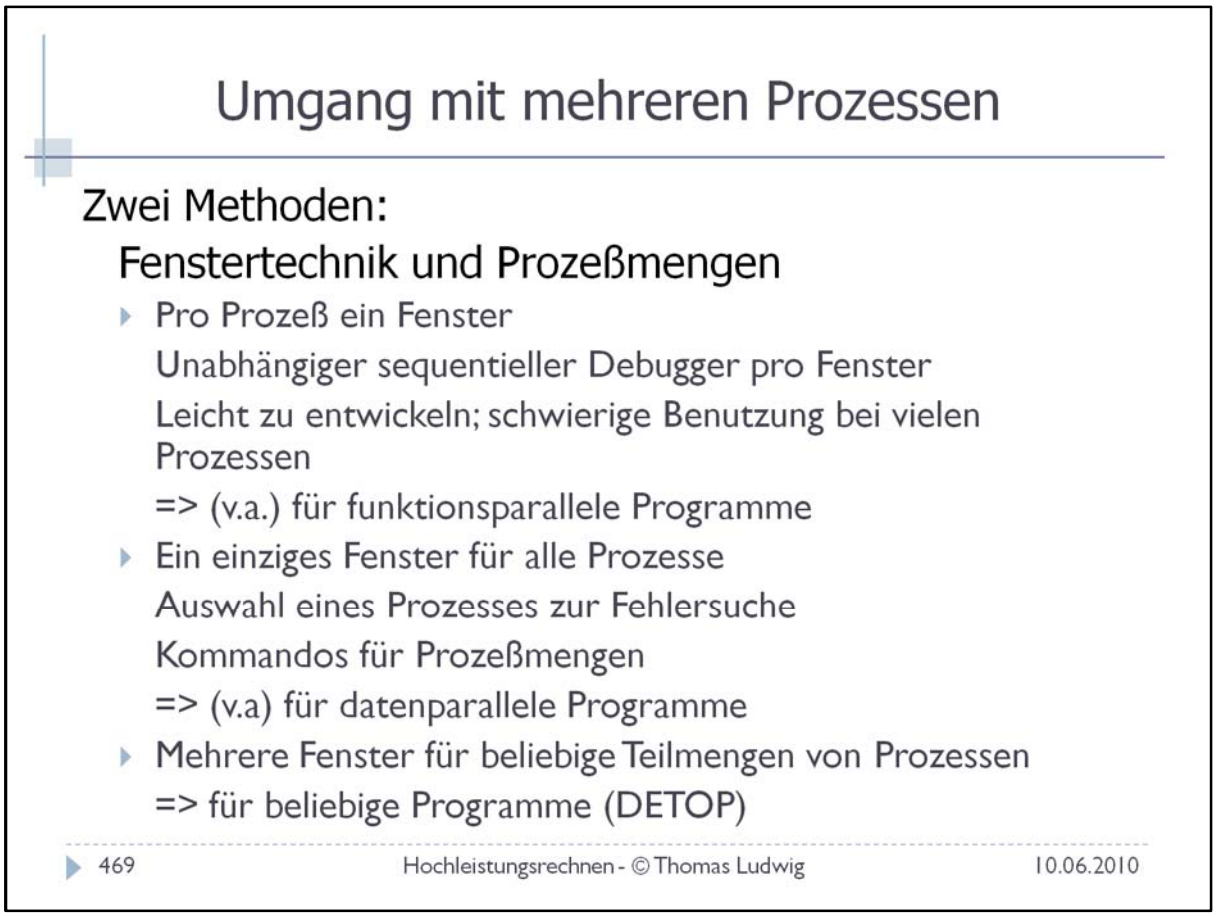
Eigenschaften paralleler Programme

- Mehrere Aktivitätsträger
  Prozesse, evtl. Threads; evtl. mehrere Binärformate
- Dynamik
  Zur Laufzeit Änderungen der Knoten, Prozesse, ...
- Interaktion
  Kommunikation und Synchronisation zwischen Prozessen
- Verteiltheit
  Verteilte Information; kein globaler Systemzustand

Berücksichtigung dieser Eigenschaften sehr unterschiedlich

Kein Standard auf dem Gebiet in Sicht

Es gibt zur Zeit zwei gebräuchliche parallele Debugger:
- The Distributed Debugging Tool DDT der Firma Allinea
- Totalview der Firma Rogue Wave.



DETOP: Debugging Tool für Parallel Programs, Eigenentwicklung an der TU München.

Skalierbarkeit und Dynamik

Problem: Umgang mit höheren Prozeßanzahlen

- Kommandos für Gruppen von Prozessen
- Zusammenfassen identischer Ergebnisse verschiedener Prozesse
- Einsatz geeigneter graphischer Darstellungen

Problem: Debugging dynamisch generierter Prozesse

- Stoppen aller neu erzeugten Prozesse; manuelle Auswahl
- Automatische Auswahl über Mustervergleich mit Zusatzinformation

Interaktion

Überwachung von Kommunikation und Synchronisation

- Möglich durch Haltepunkte auf Bibliotheksfunktionen
- Meist keine weitergehende Unterstützung, wie z.B.
  - Ausgabe wartender Prozesse
  - Status von Nachrichtenwarteschlangen
  - Haltepunkte auf Nachrichten
- Ausweg

  Gleichzeitige Benutzung spurbasierter Werkzeuge (i.a. nur lesender Zugriff) und von Spezialwerkzeugen (message queue manager, mqm)

Verteiltheit

Daten der Anwendung sind verteilt

- Spezialwerkzeuge liefern globale Sicht

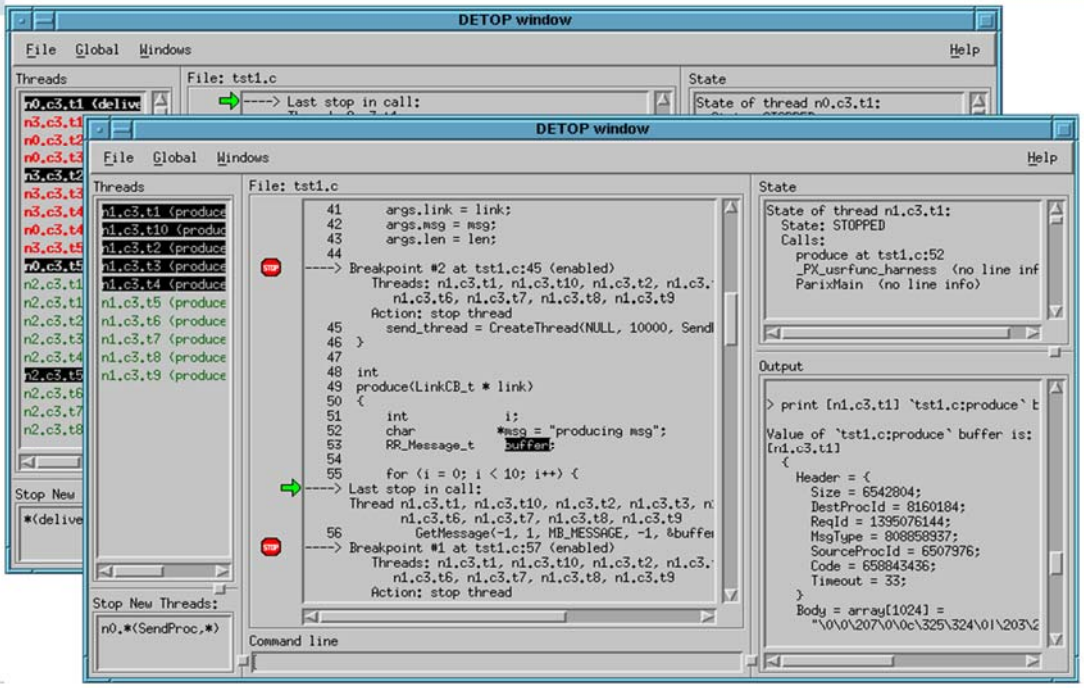
Gemeinsame Daten verteilter Prozesse

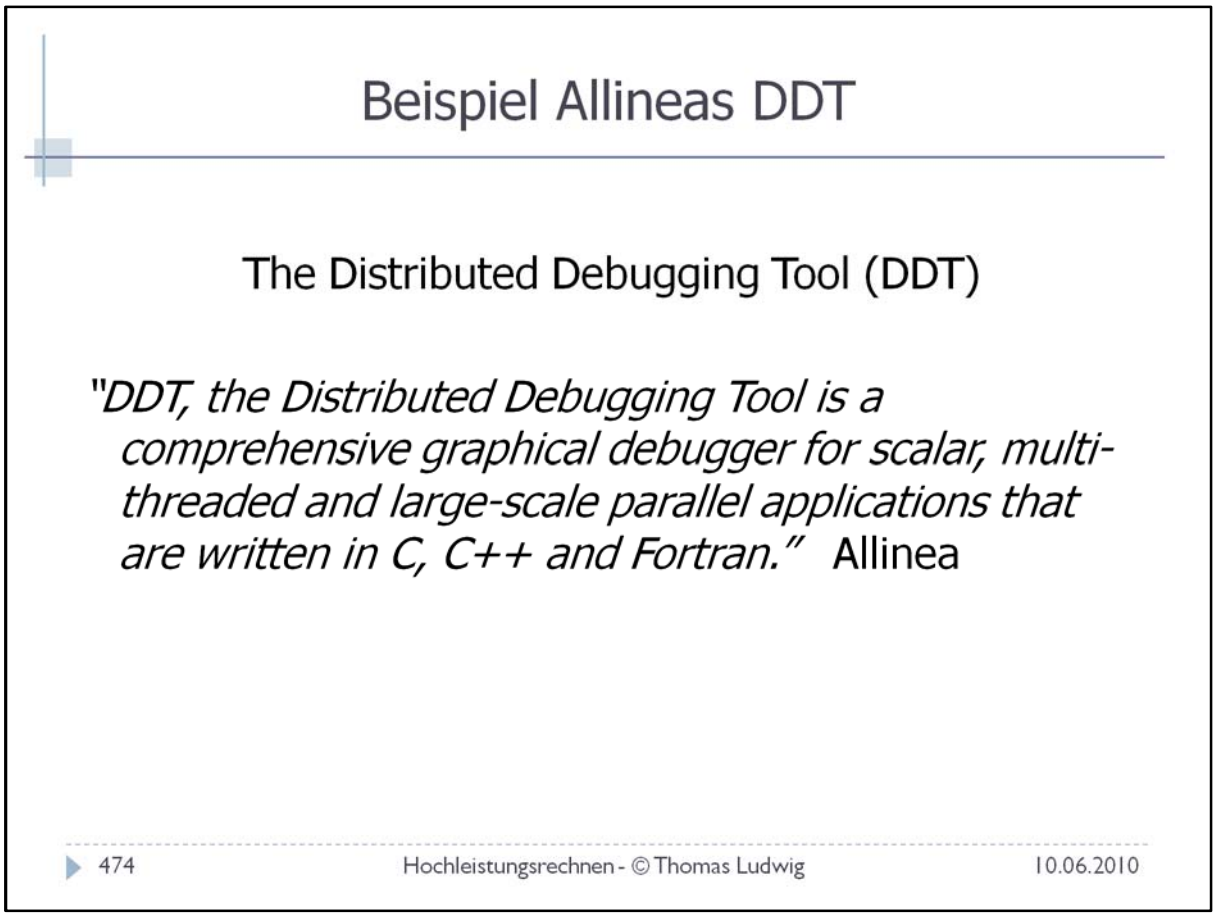
- Beispiele: MPI-Gruppen, gemeinsame Speichersegmente
- Problem: Einfrieren des Zustandes bei Erreichen des Haltepunktes
- Meist nur Anhalten eines Prozesses unterstützt
- Wenn globales Anhalten unterstützt, dann nie sofort(!)
  => Zustandveränderungen sind möglich

Globale Ereigniserkennung (Forschungsthema)

- Verknüpfung von Ereignissen in verschiedenen Prozessen
- Z.B. Ereignisse a und b sind kausal abhängig/unabhängig

Beispiel DETOP





Siehe: http://www.allinea.com/index.php?page=48

“For multi-threaded or OpenMP development DDT allows threads to be controlled individually and collectively, with advanced capabilities to examine data across threads.

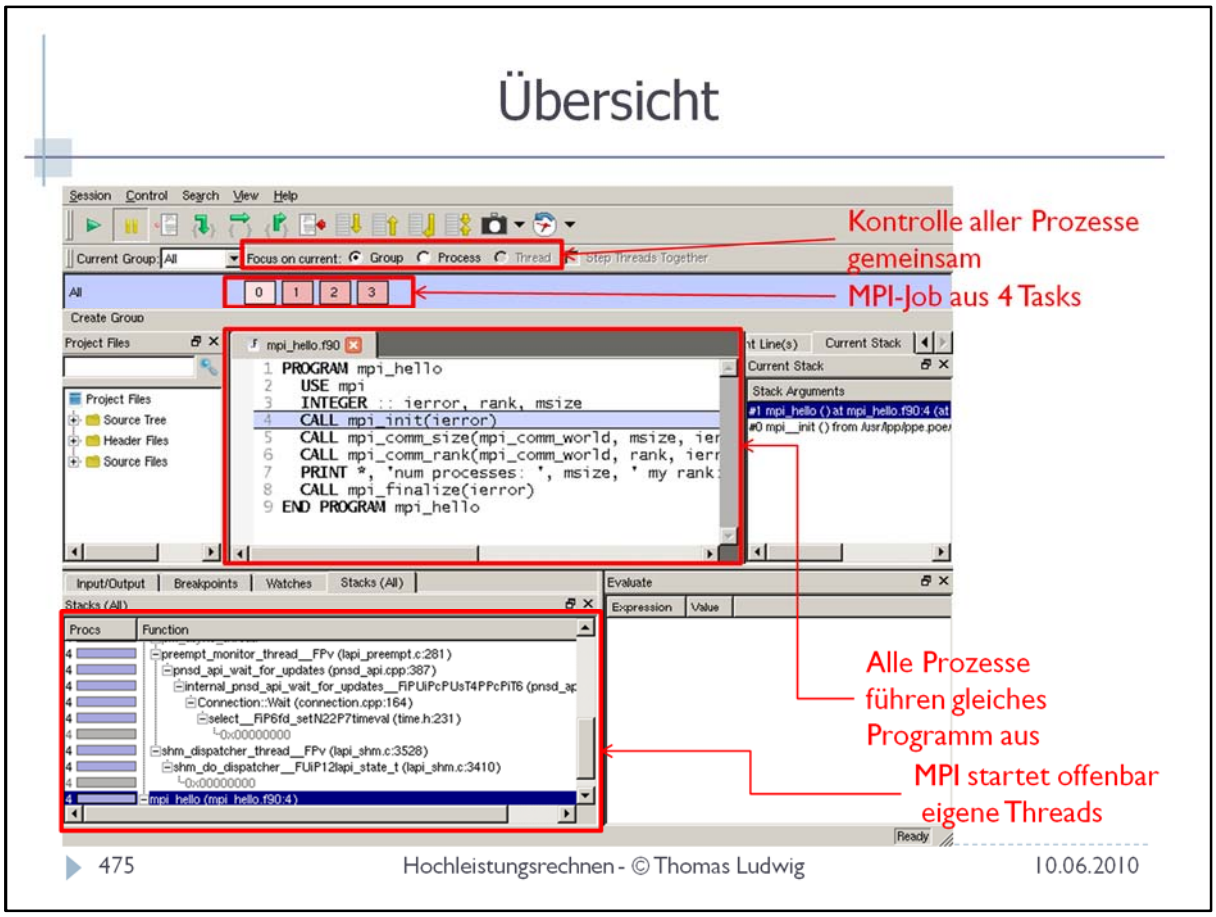
The Parallel Stack Viewer is a unique way to see the program state of all processes and threads at a glance - and developers can easily spot rogue processes or threads, and even define new control groups from it, meaning massively parallel programs are easy to manage.

DDT's interface scales amazingly - providing the same clarity of information at thousands of processes as at a handful - highlighting commonality and differences with summary views and data comparison to focus your attention.

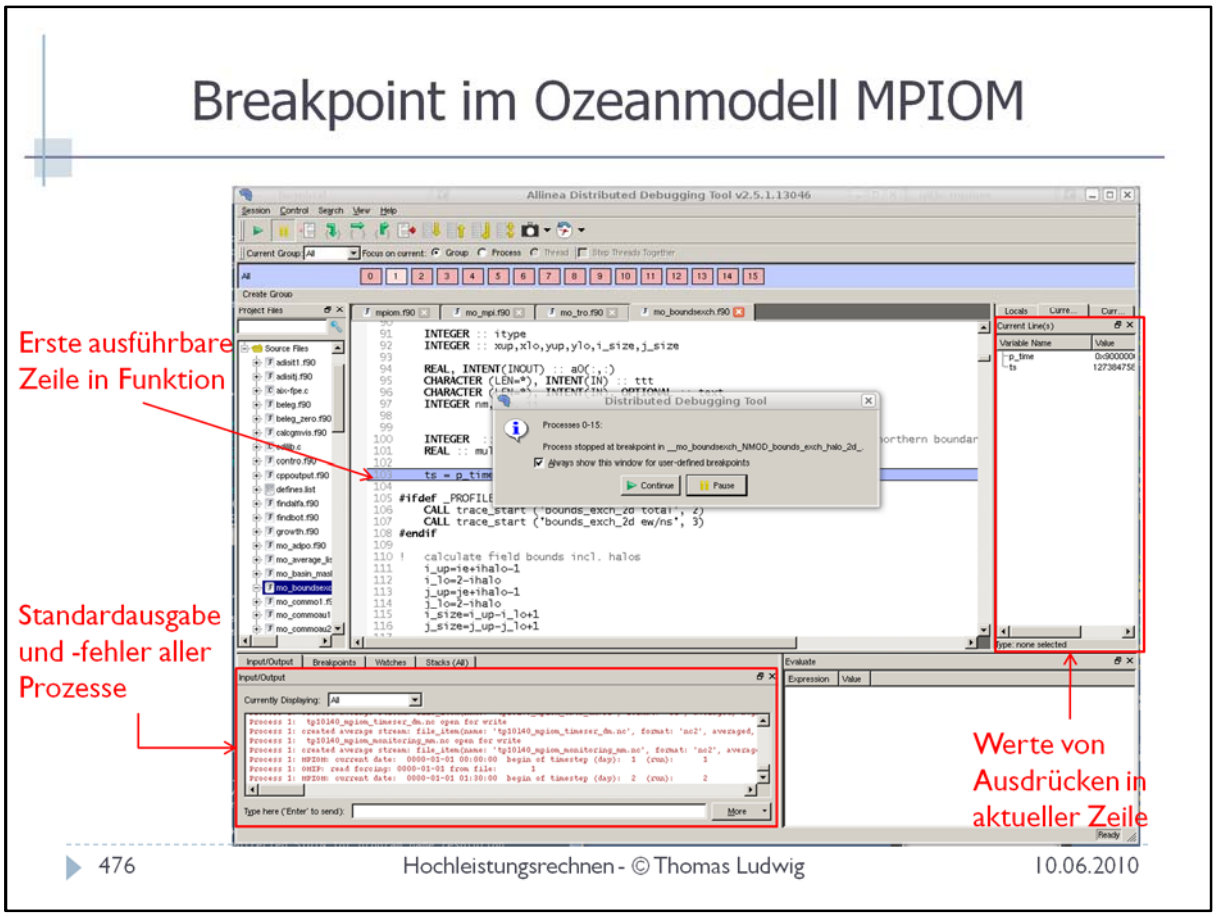
DDT is proven at scale on the most powerful systems - including debugging applications at over 200,000 cores simultaneously.

DDT's advanced memory debugging capability brings tremendous benefits to developers of scalar and parallel applications. DDT can find memory leaks, and detect common memory usage errors before your program crashes.

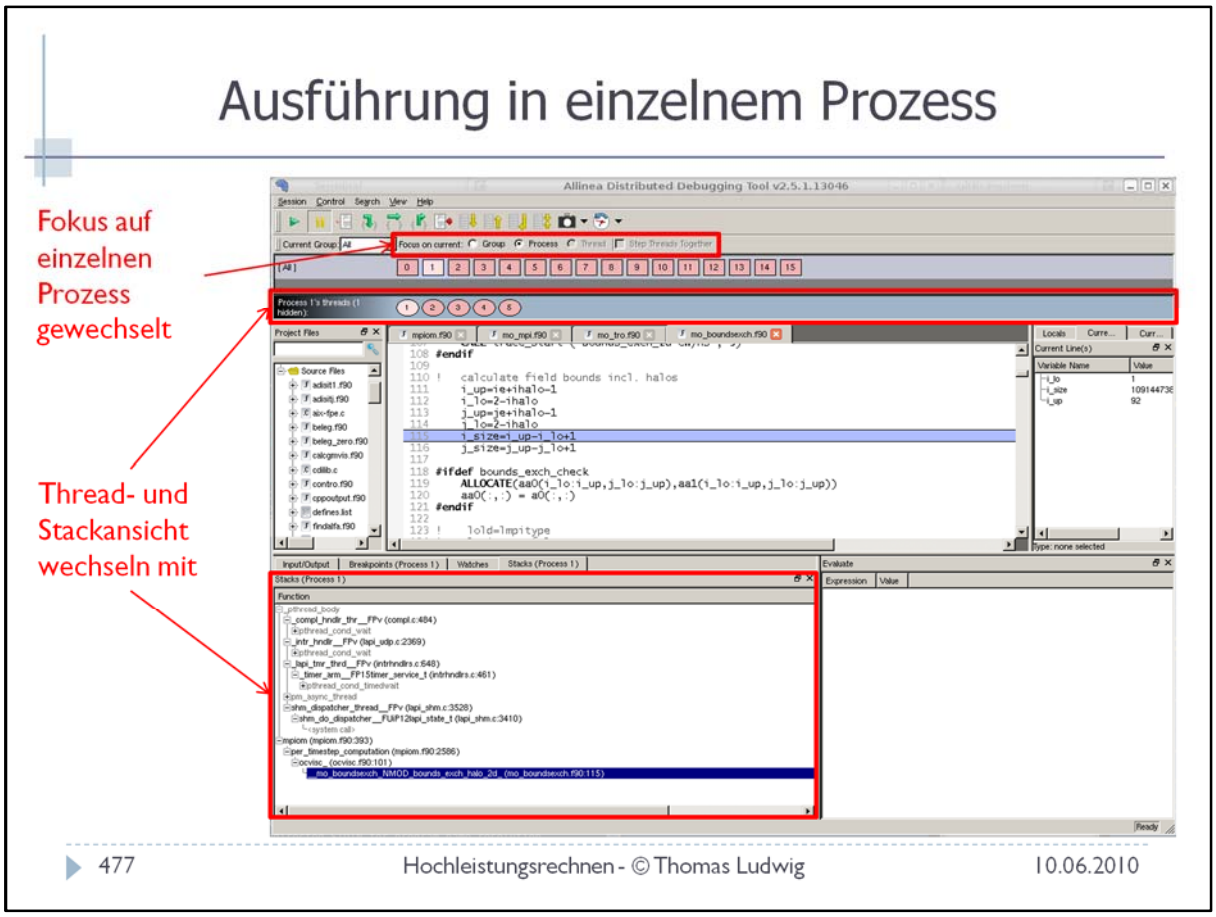
With DDT, you can check a pointer is valid or find the stack when it was allocated - a fantastic boost for any developer. Reading or writing beyond the ends of allocated data can also be detected - instantly.”



Erklärungsbedürftig sind noch: einzelne Kontrollelemente in zweiter Menüleiste (v.l.): Fortfahren, Unterbrechen, Breakpoint setzen, Step-In, Step-Over, Until



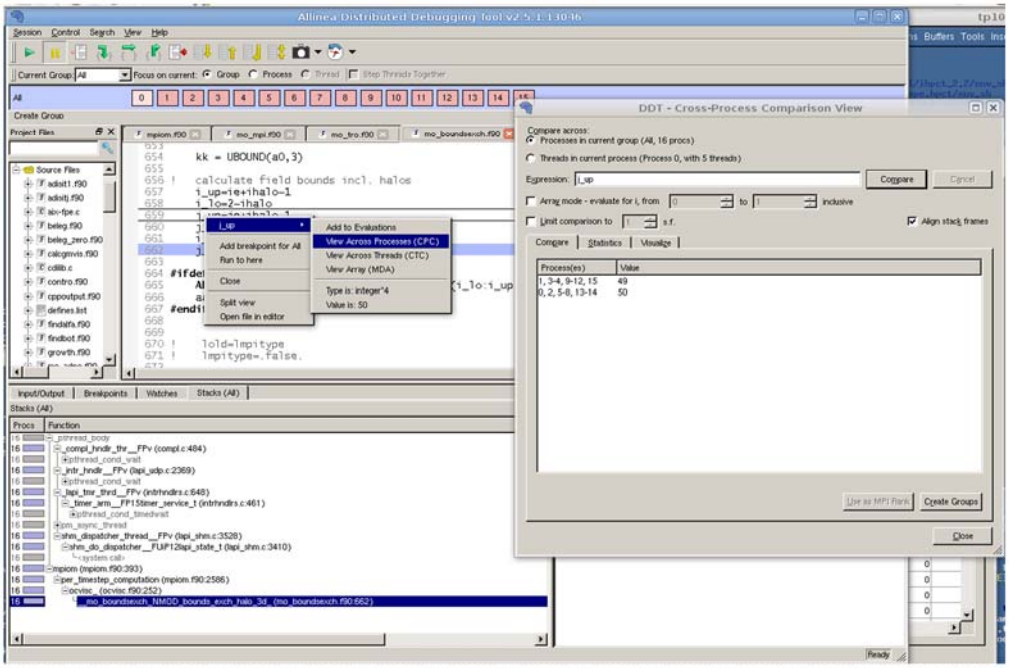
Gezeigt ist ein Lauf von MPIOM mit 16 Prozessen.
Gestoppt wurde durch einen Breakpoint auf die Kommunikationsfunktion boundsexch_halo_2d, es wird die erste ausführbare Zeile fokussiert, die erste Zeile der Funktionsdeklaration ist Fortran-üblich bereits viele Zeilen vorher und deshalb nicht im Bild.
Wie am Dialog in der Bildmitte zu erkennen wartet der Debugger typischerweise einen kurzen Moment bis alle Prozesse in der fokussierten Gruppe am Breakpoint ankommen.



Nennenswert:

- Stackansicht wechselt auf den ausgewählten Prozess
- Fortlaufende Aktualisierung der Ausdrücke in aktueller Zeile
- Automatische Anzeige der auswählbaren Threads unter Prozessen in der Gruppe

Variablenansicht



Die automatische Variable j_up hat auf mehreren Tasks verschiedene Werte die sich in der Vergleichsansicht aufzeigen lassen.
Komplexere Ansichten von Feldvariablen sind möglich (zeitaufwendig).

Deterministische Ablaufkontrolle

Problem des Nichtdeterminismus

- Erzwingen einer deterministischen Abarbeitungsreihenfolge der Kommunikation

  Programm dadurch evtl. verlangsamt

  Varianten der Reihenfolgen systematisch testbar

  (Bei sequentiellen Programmen kein Problem!)

Deterministische Ablaufkontrolle...

Funktionsweise eines Werkzeugs hierzu:

- Erster Programmlauf

  Aufzeichnen der Reihenfolge des Eintreffens von Nachrichten bei Empfängern
- Weitere Programmläufe (*deterministic replay*)

  Verwendung der Informationen aus dem ersten Programmlauf

  Die Reihenfolge, wie Nachrichten beim Empfänger ankommen, wird gesteuert: zu früh eintreffende werden zurückgestellt

Sicherungspunkte

Problem der Zykluszeit

- Bei Fehler muß das Programm vom Anfang wiederholt werden, um nach der Fehlerursache zu suchen

Vorgehensweise

- Zyklisches Erstellen von Sicherungspunkten
- Im Fehlerfall: Auswahl eines geeigneten Sicherungspunktes und Wiederanlauf des Programms von diesem Zeitpunkt aus.
- Evtl gekoppelt mit Ablaufkontrolle

Fehlersuche
Zusammenfassung

- Unterscheide Fehlerursache und Fehlerwirkung
- Typische Fehler paralleler Programme
  - Überholvorgänge, Verklemmungen
- Probleme
  - Nichtreproduzierbarkeit, Nichtdeterminismus
  - Unübersichtlichkeit, physische Verteiltheit, Dynamik
- Spurbasierte Werkzeuge für einen globalen Überblick und Prüfung der Kommunikation
- Haltepunktbasierte Debugger für Detailuntersuchungen
- Ablaufkontrolle beseitigt Nichtdeterminismus
- Sicherungspunkte verkürzen den Testzyklus