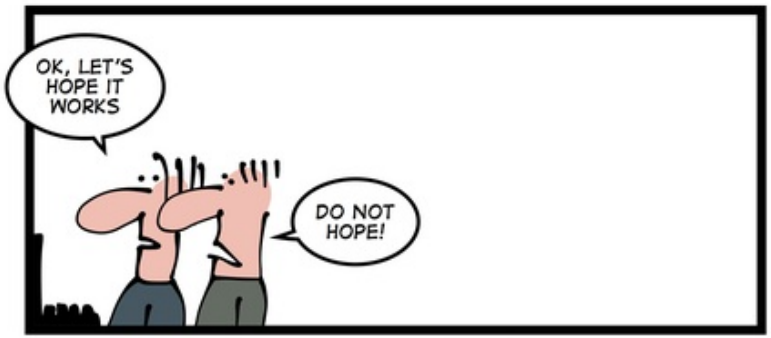
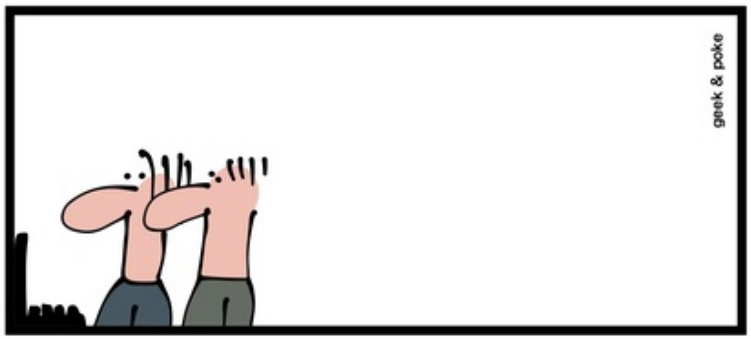
Debugging



NEVER RELY JUST ON HOPE



ALWAYS DO MORE!

http://www.datamation.com/imagesvr_ce/7349/hope.jpg

Paralleles Programmieren für Geowissenschaftler

(sequenzielles) debugging

Herrmann Lehnhart Tim Jammer

Universität Hamburg
Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik, Arbeitsbereich Wissenschaftliches Rechnen

20. April 2017

The easiest bug to catch is the one never written.

Die wichtigsten Methoden, die Debug-Zeiten zu verkürzen

- Sprechende Bezeichner verwenden
- Probleme aufteilen
  - Mache eine simple Sache zur Zeit und mache sie richtig!
  - Funktionen/Subroutines sind kurz
  - Module/Dateien sind kurz

Die wichtigsten Methoden, die Debug-Zeiten zu verkürzen

- Keine globalen Variablen
  - Der Wert von Variablen hängt maximal von einer Datei ab
  - Alle Daten werden explizit übergeben
  - Zusammengehörige Daten zu Typen zusammenfassen

  $\Rightarrow$ Objektorientierte Programmierung
- Keine hartcodierten Array-/Bufferlängen
  - Leider unmöglich mit Fortran-Strings
- Kurz gesagt: Man folge einem vernünftigen Styleguide :-)

Das simpelste Tool

- write
  - ist bei parallelen Anwendungen oft umständlich

Debugger

- Klinken sich in den Ablauf eines Programmes ein
- Können es anhalten, weiterlaufen lassen,
- schrittweise ausführen, Variablen/Code lesen/verändern
  - Der Benutzer kann dem Programm beim Ablaufen zuschauen
- Breakpoints unterbrechen das Programm, wenn eine bestimmte Codezeile erreicht wird
- Watchpoints unterbrechen das Programm, wenn eine Variable sich ändert
- Notwendige Vorbedingung: Debugsymbole
  - gcc oder gfortran erzeugen sie, wenn -g oder -ggdb übergeben wird

gdb

- Aufruf: gdb –args programm [argumente]
  - Oder: gdb programm
- Befehllisten & Beschreibungen: help
- Starten = run, unterbrechen = <Strg>c,
- weiterlaufen = continue
- Step in = step, step over = next, Code anzeigen = list
- Aufrufhierarchie = bt, Variablen ausgeben = print
- Aufrufebene ändern: up, down, frame
- Breakpoint setzen = break, Watchpoint setzen = watch

Was Debugger *nicht* können

- Arrayindizes prüfen
- Freigabe von Speicher überprüfen
- Überprüfen, ob zugegriffener Speicher noch alloziert ist

- Dafür gibt es valgrind.
- valgrind simuliert einen Prozessor, der diese Checks
- durchführt & protokolliert
  - Langsam!
- Kann noch viel mehr, z. B. Cache-Hit-Analyse

valgrind

- Aufruf: valgrind programm [argumente]
- Anzeigen, wo verlorene Blöcke alloziert wurden:
  - –leak-check=full
- Auswählen anderer Tools als memcheck, z. B. cachegrind:
  - –tool=cachegrind

- auch dafür mit debug-optionen (-g) kompilieren

Fragen?