Das Softwaresystem *UG*

Erfahrungen in der Entwicklung und in der Anwendung

Christian Wieners, **TU Chemnitz**

http://cox.iwr.uni-heidelberg.de/∼ug

Grundsätze

Satz 1

Jede Software hat eine maximale Lebensdauer.

Satz 2

Rechenzeit lässt sich in Speicherbedarf transformieren und umgekehrt.

Satz 3

Der Aufwand bei Erweiterungen steigt exponentiell.

Satz 4

Jede Entwicklung dauert länger als vorgesehen.

Komponenten des Softwaresystems *UG*

Der *UG*-Kern wird von den allen Applikationen verwendet:

```
UG/ug/CVS/
      arch/PC/                                  fe/appl/
           SGI/  ...                               pclib/
      configs/
      low/                                      csm/src/
      dev/xif/                                      mesh/
          sif/                                      data/
      parallel/ppif/MPI                             plot/
                   /PVM  ...                    Makefile
               ddd/                             csm3d*
      gm/                                       run.scr
      np/
      ui/                                       df/2d/
      dom/std_domain/                              3d/saltpool/
          lgm_domain/                                 gorleben/   ...
      graphics/                                       gen/appl/
      include/                                            disc/
      lib/                                                est/
      man/                                                misc/
      bin/                                                problems/ ...
```

Architektur(un)abhängigkeit I

Für jede Architektur wird ein machine-file in die makefiles includiert:

```
# ARCH MAKEFILE OPTIONS for Linux-PC (with GCC)

ARCH_TYPE      = __PC__
ARCH_MAKE      = make
ARCH_CC        = gcc
ARCH_C++       = g++
ARCH_LINK      = gcc
ARCH_AR        = ar
ARCH_SUFFIX    =
ARCH_POSTLINK  = true
ARCH_LIBS      = -lm
ARCH_CFLAGS    = -pipe
ARCH_NOOPTIM   = -g -Wall -Wno-unused -Wno-parentheses
ARCH_OPTIM     = -s -O3 -march=i686 -funroll-loops -fstrict-aliasing -ffast-math
ARCH_C++FLAGS  = -fno-strict-prototype
ARCH_LFLAGS    =
ARCH_L++FLAGS  = -lg++
ARCH_ARFLAGS   = rus
ARCH_XINCLUDES =
ARCH_XLIBS     = -L/usr/X11R6/lib -lXaw -lXt -lX11
```

Architektur(un)abhängigkeit II

In `compiler.h` werden architekturspezifische defines gesetzt:

```
#ifdef __PC__
#define ARCHNAME "PC"

/* basic types */
#define SHORT  short
#define INT    int
#define FLOAT  float
#define DOUBLE double
#define COORD  float
#define SCREEN_COORD  float
#define __SWAPBYTES__ 1

/* memory */
#define ALIGNMENT 4                     /* power of 2 and >= sizeof(int) !  */
#define ALIGNMASK 0xFFFFFFFC            /* compatible to alignment          */

/* fortran interfacing */
#define F77SYM(lsym,usym) lsym

/* current time as DOUBLE value */
#undef CURRENT_TIME
#define CURRENT_TIME   (((DOUBLE)clock())/((DOUBLE)CLOCKS_PER_SEC))

#endif
```

Parallelisierung

PPIF – Parallel Processor Interface

In *UG* werden nur eigene parallele Befehle (Broadcast, Send, Receive etc.) verwendet, die im jeweiligen Modul passend abgebildet werden (MPI, PVM, ...).

DDD – Distributed Dynamic Data

Alle parallelen Objekte werden über ein Modul verwaltet, das automatisch passende Interfacelisten bereitstellt, über die die gesamte parallele Numerik implementiert ist. Dieses Modul erlaubt eine effiziente Umverteilung von Basisobjekten mit einer vollständigen Wiederherstellung der konsistenten verketteten Datenstrukturen.

IO – Parallele Aus- und Eingabe

Jeder Prozessor kann seine Daten gleichzeitig auf eine eigene Platte schreiben; der parallele, abgespeicherte Zustand kann dann wiederhergestellt werden. Zur Visualisierung können die parallelen Datenfiles gemerged werden und als sequentieller Datenfile zur Visualisierung verwendet werden.

Speicherverwaltung

Zu Beginn wird der gesamte benötigte Speicher als ein langer Speicherbereich als Heap alloziert. Alle Datenstrukturen sind als verkettete Listen realisiert.

Die Listenelemente werden einzeln aus dem Heap entnommen, und frei werdende Listenelemente werden dort (mit einer hash-Liste sortiert nach Länge des Objekts) wieder gesammelt:

```
void *GetFreelistMemory (HEAP *theHeap, INT size);
INT   PutFreelistMemory (HEAP *theHeap, void *object, INT size);
```

Die freie Liste wird am unteren Ende des Heaps angelegt; wenn neuer Speicher benötigt wird, wird zunächst gesucht, ob die freie Liste ein geeigneten Speicherbereich der benötigten Länge enthält; ansonsten wird (von unten) ein neuer Speicherbereich alloziert freigegeben.

Zur Anforderung eines temporären Speicherbereichs wird am oberen Ende des Heap eine Marke gesetzt und anschließend Speicherplatz alloziert; durch Release wird der gesamte Platz bis zur Marke freigegeben.

```
INT  Mark            (HEAP *theHeap, INT *key);
void *GetMemUsingKey (HEAP *theHeap, INT n, INT key);
INT  Release         (HEAP *theHeap, INT  key);
```

OO-Programing: Classes

Die ganze Numerik ist in einem Klassenkonzept realisiert, indem z. B. alle Glätter das gleiche Interface haben. Für jede Klasse existiert ein Konstruktor, der die Realisierung mehrerer Instanzen derselben Klassen erlaubt.

```
ts.bdf -> nls.newton -> disc.feassemble
                     -> solver.bcgs -> iter.lmgc -> iter.sgs
                                                 -> transfer.fetransfer
                                                 -> solver.bcgs -> iter.ilu
       -> error.residual
       -> transfer.fetransfer
```

Die Klassen sind im script frei konfigurierbar, z. B.

```
npcreate smooth $c jac;          npinit smooth $damp 0.5;

npcreate baseiter $c lu;         npinit baseiter;

npcreate basesolver $c ls;       npinit basesolver $red 0.001 $m 1 $I baseiter;

npcreate transfer $c fetransfer; npinit transfer $x sol;

npcreate lmgc $c lmgc;           npinit lmgc $S smooth smooth basesolver $T transfer;

npcreate mgs $c cg;              npinit mgs $A A $x x $b b $m 8 $red 0.001 $I lmgc;
```

Versionsmanagement

Mit `cvs` (ruft `rcs` auf) kann man file-Versionen verwalten und mergen. Für verteiltes Entwickeln ist es unerlässlich. Wesentlich für die Sicherheit und Zuverlässigkeit ist die Verwaltung einer access-Liste (z. B. `cvs admin -awieners`).

Die Zeile

```
/* RCS string */
static char RCS_ID("$Header$",UG_RCS_STRING);
```

wird durch `cvs ci initug.c`

```
static char RCS_ID("$Header: /home/cvs/UG/ug/initug.c,v 1.32
2000/07/19 09:38:21 klaus Exp $",UG_RCS_STRING);
```

ersetzt; im binary ist es dann enthalten

```
... $^@^@^@^@^@^@^@^@^@$Header: /home/cvs/UG/ug/initug.c,v 1.32
2000/07/19 09:38:21 klaus Exp $$State: UG_VERSION="ug 3.8"
ARCH_VERSION="ARCH_1_0" DIM=2 GRAPE_SUPPORT=OFF
MODEL=MPI NETGEN_SUPPORT=OFF DOM_MODULE=STD_DOMAIN
DEBUG_MODE=ON $^@^@^@^@^@^@^@ ...
```

und kann mit `ugbcheck fe2d` auf Konsistenz überprüft werden.

Debugging

Der Einsatz von `totalview`, `softbench`, `mpirun -gdb` hilft nur bedingt. Zusätzlich haben wir folgende Konzepte realisiert, um Fehler aufzuspüren, die erst nach vielen Gitteradaptionsschritten, vielen Zeitschritten, und bei vielen Prozessoren auftreten:

a) Checkfunktion (`check $a`) zur Überprüpung der Konsistenz der Datenstrukturen.

b) Im Code eingefügte modular bedienbare debug-Funktionen (`debug $gm $3`):

```
IFDEBUG(gm,2)
INT i;  EDGE* edge0 = NULL; EDGE* edge1 = NULL;

if (fatherEdge!=NULL) {
    if (MIDNODE(fatherEdge)!=NULL) {
        edge0 = GetEdge(MIDNODE(fatherEdge),SONNODE(NBNODE(LINK0(fatherEdge))));
        edge1 = GetEdge(MIDNODE(fatherEdge),SONNODE(NBNODE(LINK1(fatherEdge))));
    }
    else
        edge0 = GetEdge(SONNODE(NBNODE(LINK0(fatherEdge))),SONNODE(NBNODE(LINK1(fatherEdge))));
}
assert(edge0==theEdge || edge1==theEdge); /* test whether theEdge lies above fatherEdge */
ENDDEBUG
```

c) Logfiles für jeden Prozessor.

d) Paralleles IO zum Aufsetzen genau in der kritischen Situation.

Dokumentation

Ideal ist eine Dokumentation im Quelltext:

```
/****************************************************************************/
/*D
   level - select another current level

   DESCRIPTION:
   This command changes another current level of the current multigrid.

   level <level> | + | -

.  <level> - go to level <level>
.  +       - go to the next finer level
.  -       - go to the next coarser level

   KEYWORDS:
   multigrid, current
D*/
/****************************************************************************/
```

Mit `doctext` (public domain) lassen sich die Kommentare lesen und in man-pages, html oder latex konvergieren.
Sie sind mit `ugman` / `xugman` von der UNIX-shell oder mit `help` in der *UG*-shell aufrufbar.

Outsourcing

Gittergenerierung

Für komplexe 3D-Geometrien verwenden wir zur Gittergenerierung u. a.

MARC (Hexaeder, kommerziell), *ANSYS* (Tetraeder, kommerziell), *NETGEN* (Tetraeder)

über verschiedene file-Schnittstellen. Bisher können damit nur Polyedergeometrien eingelesen werden; für gekrümmte Oberflächen müssen Parametrisierungen programmiert werden, damit die Geometrieapproximation bei Verfeinerung verbessert wird. Insbesondere ist die Wahl von Toleranzen und die Kantendetektion problematisch.

Visualisierung

Wir verwenden (auch im Parallelen verfügbare) sequentielle file-Schnittstellen zu

TECPLOT (kommerziell), *AVS* (kommerziell) und *DATAEXPLORER*.

In einem großen Projekt wurde u. a. eine binäre file-Schnittstelle zu *GRAPE* entwickelt, das unser paralleles IO verwendet; nur so lassen sich Datensätze mit Millionen von Unbekannte visualisieren.

Open source?

Ideal ist eine Open-Source-Entwicklung unter GNU-Lizenz.

Weit verbreitet sind freie Lizenzen für den nichtkommerziellen Einsatz (*UG*, *NETGEN*); erst nach Eingang der Lizenz wird der Zugriff auf das Programm (oder Teile des Programms) freigegeben.
Damit wird zumindest die Kontrolle ermöglicht, wer sich für das Programm interessiert.
So soll eine mögliche eigene Vermarktung des Programms offen gehalten werden.

Sehr problematisch ist die Entwicklung von Schnittstellen ohne Quelltext (*GRAPE*).

Extrem ungünstig ist ein Licence-server (*AVS*, *ANSYS*).

How to get

Die Software ist über `ftp` erhältlich

```
ftp> cd ug-3.8
250-  This directory contains the latest versions of UG
250-
250-  ug-3.8.tar.gz       gzipped ug-3.8 version (release 1) including all
250-  ugpaper.ps.gz       description of the software concept
250-  applmanual.ps.gz    documentation for applications (gzipped file)
250-  progmanual.ps.gz    module descriptions, commands, overviews,..(gzipped file)
250-  refmanualI.ps.gz    description of C functions Part I (gzipped file)
250-  refmanualII.ps.gz   description of C functions Part II (gzipped file)
250-  licence.ps.gz       postscript version of license form (gzipped file)
250-  licence.ps          postscript version of license form
250-  licencelink.ps.gz   postscript version of license form (link version) (gzipped file)
250-  licencelink.ps      postscript version of license form (link version)
250-
250-  The file ug-3.8.tar.gz contains also the postscript documentation and the license form.
250-
250-  QUESTIONS, COMMENTS, BUG REPORTS:   write email to ug@iwr.uni-heidelberg.de
250-
250 CWD command successful.
ftp>
```

und natürlich durch anklicken auf unserer WWW-Seite ...