Teil 1: Fortran 90
(Dr. Tonnis Pool)

Erste Erfahrungen mit der Version 2.0.0.0 aus Betreuersicht

Inhalt

- CPU-Zeitvergleich von Programmläufen
- Qualität der Dokumentation
- Beispiel: Optimierung durch **loop-splitting**
- Vorhandene Tools für die Konvertierung f77 ⇒f90
- Probleme bei der **Data Representation**
- Fehler und ihre Behebung

CPU-Zeitvergleich der Programmläufe

Randbedingungen:

- Keine Änderungen an der Quelle
- Fortran 77 System: Übersetzung, wie vom Anwender getätigt, meist nur:

```
cf77 -Zv -Wf"-o aggress,inline3" <files>
```

- Fortran 90 System: möglichst optimiert mit inline2 oder inline3:

```
f90 -O "scalar3,vector3,aggress,inline3,inlinefrom=<dir>"
   <files>
```

CPU-Zeit in Sekunden, gemessen mittels timex

Fachgebiet	cf77	f90	Prozentuale Abweichung
Chemie	19.0	18.4	3
Chemie	58.5	57.8	1
Chemie	74.4	38.2	49
CFD [1])	145.5	100.2	31
Elektrotechnik	104.8	92.7	12
Elektrotechnik	**93.2**	**105.8**	**-14 (schlechter)**
Maschinenbau [2])	50.6	34.7	31
	321.1	163.9	49
Physik	127.4	98.3	23
Physik	74.8	72.8	3
Physik	87.0	79.3	9
Physik (Case study)	**48.0**	**58.6**	**-22 (schlechter)**

1) Computational Fluid Dynamics

2) Gleiches Programm, verschiedene Laufzeitparameter

Bemerkungen zum CPU–Zeitvergleich

- Portierung der Fortran 77–Quellen nach Fortran 90 durch Anwender geht **nur sehr schleppend** voran. Es fehlt den Anwendern bis jetzt an **Zeit, Mut und Tools** für notwendigen Veränderungen.

- Obwohl in jedem Semester Kurse zur Fortran 77 Programmierumgebung angeboten werden, wird nur ein Teil der Anwender erreicht. Viele wissen daher zu wenig von den Optimierungsmöglichkeiten. Eine stete Überwachung der MFLOPS-Raten ermöglicht den Beratern, auf Verdacht und stichprobenweise zu überprüfen, ob wenigstens die Option -Zv eingesetzt wird. Einzelne Programme wurden hochoptimiert.

- CF77: Problematisch ist die zweistufige Optimierung und Vektorisierung: teilweise im FPP, teilweise im CFT77. Z.B. ist es den meisten Benutzern nicht auf Anhieb klar, dass das Inlining am Effektivsten vom FPP zu erledigen ist und nicht vom CFT77. Zu allem Überdruss sind die Inlining–Optionen bei FPP und CFT77 ziemlich verschieden.
- CF90: Optimierung und Vektorisierung sind einstufig und können mit wenigen Angaben eingesetzt werden. So auch das Inlining.

Case Study: Warum ist das f90 Executable langsamer als das cf77 Executable?

Messungen mit verschiedenen Compile–Kommandos, in denen keine oder alle Subroutinen–Aufrufe mittels Inlining ersetzt werden:

```
cf77 -Zv -Wf"-o aggress" *.f                              48.18s
cf77 -Zv -Wd"-I grab,mult,dagger,trace" -Wf"-o aggress" *.f48.02s

f90 *.f                                          61.12s
f90 -dp -O aggress,scalar3,vector3 *.f                     60.29s
f90 -dp -O aggress,scalar3,vector3,inline3 *.f                62.04s
f90 -dp -O aggress,scalar3,vector3,inline2 *.f                58.62s
f90 -dp -O aggress,scalar3,vector3,inline2,bl *.f             58.62s
f90 -dp -O aggress,scalar3,vector3,inline1 *.f                58.74s
           mit !DIR$ INLINE ALWAYS in grab,mult,dagger,trace
```

Die f90–Version ist etwa 22% langsamer!!

Merkwürdiges Verhalten bei Option inline3:

- Übersetzungszeit: **16.5 min** statt **1.2 min** bei inline1 **und**
- Ausführungszeit **länger!**

Empfehlungen:

- Für den Optimierungsgewinn reicht es meistens, die Option **inline2** zu nehmen.
- Falls die Compile-Zeit zu lang ist, kann man auf die Option **inline1** ausweichen. Zusätzlich mit den **Perf–Tools** den Inline–Faktor bestimmen und Direktiven anwenden:

f77 perfview report, tabellarisch

Name	Called	Time [s]	Ex %	SMips	V I/L	VMflops
DSLASH	**49228**	**5.31e+01**	**86.3**	**11.8**	**2.4**	**153.7**
			...			
SUM2D	1200	1.93e-01	0.3	12.9	92.8	165.4
			...			
PHASER	330	4.65e-01	0.8	16.6	0.0	14.7
			...			
PBARB	29	5.93e-04	0.0	15.5	4.7	256.6
Totals	98645	4.90e+01	100.0	13.0	3.1	195.4

f90 perfview report, tabellarisch

Name	Called	Time [s]	Ex %	SMips	V I/L	VMflops
DSLASH	**49228**	**4.09e+01**	**83.5**	**11.8**	**0.3**	**199.5.7**
			...			
SUM2D	1200	4.40e-01	0.9	11.7	155.7	210.3
			...			
PHASER	330	8.05e-02	0.2	19.9	0.0	85.0
			...			
PBARB	29	7.92e-04	0.0	19.5	14.1	199.2
Totals	98645	6.16e+01	100.0	12.3	3.6	154.7

Aus dem f90-Compilerprotokoll von DSLASH:

```
1297  49     1 ------       do 200 su2 = 1,4
1298  50     1v -----          do 100 i = 0,vlen-1
1299  51     1v                   dfermi(i,su2)=zero
1300  52     1v ----> 100      continue
1301  53     1 -----> 200   continue

<f90-6003,Scalar> A loop starting at line 1297 was collapsed into
the loop starting at line 1298
<f90-6204,Vector> A loop starting at line 1298 was vectorized.
<f90-8135,Scalar> Loop starting at line 1298 was unrolled 2 times.
```

Aus dem f77-Compilerprotokoll von DSLASH mit der Umwandlung durch den FPP:

```
1316  50.           CDIR@ IVDEP
1317  51.V-------<   DO 100 I = 1, VLEN
1318  52.V           DFERMI(I-1,1) = 0.0
1319  53.V           DFERMI(I-1,2) = 0.0
1320  54.V           DFERMI(I-1,3) = 0.0
1321  55.V           DFERMI(I-1,4) = 0.0
1322  56.V------->100  CONTINUE
```

```
cft77-8004 cf77: VECTOR DSLASH, Line = 51,
File = /tmp/jtmp.001574a/qfd-bulk.my.m, Line = 1317  Loop
starting at line 51 was vectorized.
```

Vergleich der Compiler-Protokolle der vier Subroutinen

Kein Hinweis auf den Grund der unterschiedlichen Performance!

Fazit: Weitergabe an CRI-Entwickler!

Allgemeine Unterschiede bei den Compilern:

CF77

- Der Präprozessor FPP ist oft in der Lage, geschachtelte Schleifen durch einfache zu ersetzen, entweder durch **loop unwinding**, oder durch **loop collapsing**, wobei durch Linearisierung der Feldzugriffe der Speicher konsekutiv durchlaufen wird.
- Bei **Inlining** wird der eingebettete Quellcode protokolliert.

CF90

- Bei komplexeren geschachtelten Schleifen wird, obwohl möglich, kein **loop collapsing** durchgeführt.
- Bei Inlining wird der eingebettete Quellcode nicht protokolliert.

Dokumentation

CF77

- Die benutzten Handbücher haben sich fast überall durch ihre Ausführlichkeit und Klarheit ausgezeichnet, besser als bei der Konkurrenz. Der Stil ist hervorragend.
- Die steifen Ordner sind unhandlich.

CF90

- Klarheit: Im allgemeinen sind die Ausführungen nicht mehr so klar wie von cf77 her gewohnt.
- Fehler: bis jetzt wenige entdeckt. Siehe Fallbeispiele!
- Großer Nachteil der neuen Manuale: Seiten schlecht ausgenutzt.

▶ **Es ist sehr schwer, sich schnell einen Überblick zu verschaffen.**

Craydoc

- Einige Ungereimtheiten, z.B. Fenster, die sich nicht vergrößern lassen.
  Wesentliche Verbesserung am RHRK durch **Insight–Eingliederung** auf SGI-Rechner.

Fehler in der Dokumentation

- Im **CF90 Commands and Directives Reference Manual SR-3901 2.0 S. 37** fehlerhafte Beschreibung der Option "-O inlinefrom":

  The procedure is available in a file or directory **that was compiled** using the -O inlinefrom option.

Es darf natürlich nicht vorher compiliert werden. Außerdem ist die auf dieser Seite erwähnte Direktive **INLINEFROM** nicht vorhanden.

- Im **CF90 Fortran Language Reference Manual, Volume 1, SR-3902 2.0 S. 425**:

  If the FILE= specifier is omitted and the unit is not already connected to a file, the STATUS= specifier must have the value SCRATCH and the unit becomes connected to a file with the unit number appended to the string fort.

Der **STATUS=** specifier muss weggelassen werden, oder den Wert UNKNOWN haben. Hat er den Wert, so wird in $TMPDIR eine Datei angelegt, die bei Programmende gelöscht wird.

Beispiel: Optimierung durch loop-splitting

loop_splitting.f: Beispiel, in dem der CF90–Compiler schlecht abschneidet:

- CFT77 Version: 6.0.5.0
- CF90 Version: 2.0.0.0

CRAY Fortran 90, erste Erfahrungen und GUI's

```
PROGRAM spl
       PARAMETER(nra = 4000, nca = 4000, ncb = 4000)
       INTEGER  i, j
       DIMENSION a(nra, nca), b(nra, nca), c(nra, nca)
       REAL      alpha
c loop_splitting.f
       alpha = 2.34567
       DO j = 1, nca
       DO i = 1, nra
       a(i, j) = i
       b(i, j) = j
       c(i, j) = 1.
       ENDDO
       ENDDO
       CALL split(alpha,a,nra,b,nca,c,ncb)
       WRITE(*,*) c(1000,1000)
       STOP
       END
c
       SUBROUTINE split(alpha,a,nra,b,nca,c,ncb)
       INTEGER nra, nca, ncb, i, j
       REAL    alpha
       DIMENSION a(nra, nca), b(nra, nca), c(nra, nca)
c
       DO j = 1, nca
       DO i = 2, nra
c         Rekursion nur für j = nca
       c(i, nca) = c(i-1, j) + alpha * a(i, j) * b(j, i)
       ENDDO
ENDDO
END
```

Kommando für Datei loop_splitting.f	CPU-Zeit [sec]
cf77 -Zv	1.05
f77 -Zv -Wd" -I split" -Wf" -o aggress"	1.07
f90	5.97
f90 -O aggress,scalar3,vector3	4,90
f90 -O aggress,scalar3,vector3,inline3	4.46

```
PROGRAM spl  ! loop_splitting.f90
INTEGER, PARAMETER :: nra = 4000, nca = 4000
INTEGER, PARAMETER :: ncb = 4000
INTEGER            :: i, j
REAL, DIMENSION(nra, nca) :: a, b, c
REAL                      :: alpha
alpha = 2.34567
DO j = 1, nca
       DO i = 1, nra
       a(i, j) = i
       b(i, j) = j
       c(i, j) = 1.
       ENDDO
ENDDO
CALL split(alpha,a,nra,b,nca,c,ncb)
WRITE(*,*) c(1000,1000)
STOP
END PROGRAM spl
SUBROUTINE split(alpha,a,nra,b,nca,c,ncb)
INTEGER :: nra, nca, ncb, i, j
REAL    :: alpha
REAL, DIMENSION(:,:)  :: a, b, c
DO j = 1, nca
       DO i = 2, nra
       c(i, nca) = c(i-1, j) + alpha * a(i, j) * b(j, i)
       ENDDO
ENDDO
END SUBROUTINE split
```

Kommando für Datei loop_splitting.f90	CPU-Zeit [sec]
`f90`	5.97
`f90 -O aggress,scalar3,vector3,inline`	4.46

Fazit: Die Performance des f77-Compilers beruht auf der Umwandlung, die der Preprozessor FPP bei der Schleife durchführt. Testweise Ersetzung in loop_splitting.f90.

Im folgenden Beispiel: Laufzeit, unabhängig von den gewählten Optionen: 1.12 sec. Offensichtlich muss man hier per Hand optimieren !!

```
DO j = 1, nca
     IF (j .NE. nca) THEN
!DIR$ IVDEP            ! Hier gibt es keine Abhängigkeit.
     DO i = 1, nra - 1
     c(1+i, nca) = c(i,j) + alpha * a(1+i,j) * b(j,1+i)
     ENDDO
     ELSE
!DIR$ NEXTSCALAR  ! Hier ist die Rekursion.
     DO i = 2, nra
     c(i, nca) = c(i-1, j) + alpha * a(i, j) * b(j, i)
     ENDDO
     ENDIF
ENDDO
```

Konversions-Tools

convert.f90

- Das Tool wurde von Metcalf, Autor von "**Fortran 90/95 explained**", geschrieben und kann mittels anonymous ftp kostenfrei von **jkr.cc.rl.ac.uk** bezogen werden (Directory **/pub/MandR**).

- Die Leistung ist enttäuschend.

VAST/77to90

Leistungs-Beschreibung in der July/August 1993 Ausgabe von Fortran Journal. Es leistet mehr als o.g.:

- Eliminierung von GOTO
- Array Syntax anstelle von Schleifen
- MODULE's anstelle von COMMON's

Preis: zeilenanzahlbezogen, z.B. $450,- für 10000 Zeilen
Info: http://www.psrv.com/vast/vast77to90.html

Fortran 77 to 90 Converter

Ein kostenfreies GUI von NAG für deren NAGWare f77 und f90 Tools.
Man braucht folgende beide Tools:

- f77 Tool: Wandelt (Fortran 66 und) Fortran 77 Programme in standardisierte Form um.
- f90 Tool: Wandelt standardisierte Fortran 77 Programme in **free format** Fortran 90 um.
  Weniger Leistung als VAST/77to90.

Preis der Tools im Paket: £ 720,-
Info: http://www.nag.co.uk:80/nagware.html

Data Representation

Die Erläuterungen in CF90 Fortran Language Reference Manual, Volume 3, SR-3905 2.0, Chapter 8 basieren auf den expliziten Werten des KIND Type Parameters. Dies provoziert bei der Spezifikation von Variablen falsche Gewohnheiten.

Für die Spezifikation von REAL-Variablen sind für Fortran 77 und seine Dialekten möglich:

CRAY CFT77	andere Compiler
	REAL*4 r4
REAL r	REAL*8 r8
DOUBLE PRECISION d	REAL*16 r16

Fortran 90 Standard:

```
REAL [ ( [ KIND = ] kind_param ) ] :: varlist
```

wobei der Wert vom KIND-Parameter kind_param die interne Darstellung der Zahlenwerte bestimmt, wie die Längenangabe 4, 8 oder 16 das auch tut.

CRAY Fortran 90 erlaubt:

```
REAL(KIND=4)     :: r1     ! interne Darst. wie r2
REAL(KIND=8)     :: r2     ! CRAY real
REAL(KIND=16)    :: r3     ! CRAY double precision
```

oder

```
REAL(4)       :: r1     ! interne Darst. wie r2
REAL(8)       :: r2     ! CRAY real
REAL(16)      :: r3     ! CRAY double precision
```

Nicht empfehlenswert, weil die _param-Werte compilerabhängig sind.

Besser: Werte bestimmen, und zwar

- **anhand von bestehenden REAL-Zahlvarianten** und der KIND-Intrinsic-Funktion:

```
REAL(KIND(1E0)) :: r2    !CRAY real
REAL(KIND(1D0)) :: r3    !CRAY double precision
```

- oder **anhand der erforderlichen Genauigkeit** einer Anzahl Ziffern der Dezimalzahlen:

```
REAL(SELECTED_REAL_KIND(13))  :: r2  !CRAY real
REAL(SELECTED_REAL_KIND(28))  :: r3  !CRAY double
```

Unterschiedlichen Werte der Intrinsic–Funktion für die Zahlvarianten Integer-Zahl, einfachgenauer und doppeltgenauer Gleitkommazahl für verschiedene Fortran-Compiler:

F-Compiler	NAG f90 IBM RS6000	xlf90 IBM RS6000	f90 SGI SC900	f90 CRAY YMP
int KIND(1)	3	4	4	6
real KIND(1E0)	1	4	4	8
dou KIND(1D0)	2	8	8	16

Geignete Methode:

kind_param-Werte parametrisieren und in einem vereinbaren:

```
INTEGER, PARAMETER   :: short = KIND(1E0)
INTEGER, PARAMETER   :: long = KIND(1D0)
REAL(short)          :: r2
REAL(long)           :: r3
```

oder noch besser

```
INTEGER, PARAMETER   :: short = SELECTED_REAL_KIND(13)
INTEGER, PARAMETER   :: long = SELECTED_REAL_KIND(28)
REAL(short)          :: r2
REAL(long)           :: r3
```

Vorteil

Erfordernisse des Programms werden berücksichtigt, nicht die Compiler–spezifischen Eigenschaften einer Plattform.

Bei einfachgenauen Gleitkommazahlen: Defaulteinstellung ausreichend:

```
REAL   :: r2
```

Fehler-Beispiele

CRAY Fortran 90, erste Erfahrungen und GUI's

- Beim **Compilieren einer .F-Datei** mit der Option -r 0 , um ein Listing zu erzeugen, folgende Meldung:

```
Error - CIF file "minmax.T" points at
  "/usr/people/pool/f90.src/minmax.F" -
  but "/tmp/jtmp.001015a/minmax.f" is expected

"cflist" terminated
```

SPR Number: 101274 , 01-MAR-96
SPR Status: FIX-IN-REVIEW, seit 15-MAY-96

- **F90 Version 2.0 dies**

  with a nonsens error message if -eP (pre-processing only) is specified of the command line.

  SPR Number: 101271, 01-MAR-96
  SPR Status: RELEASED 28-Sep-96, Fixed in version 2.0.1

CRAY Fortran 90, erste Erfahrungen und GUI's

- **F90fe dumps core on inlining module**
  that have been compiled with option modinline. Which means that inlining is NOT available for users.

  SPR Number: 101850 , 22-MAR-96
  SPR Status: RELEASED 28-Sep-96, Fixed in version 2.0.1

- **Bound Checking** (s.u.)

  SPR Number: 102030, 28-MAR-96
  SPR Status: UNDER-INVESTIGATION, seit 29-MAR-96

- **Default Module Search** (s.u.).

  CF90, when compiling modules that contain references to 'modules' defined in other source modules, searches the current directory for any .o files and uses definitions contained therein. Later segldr complains about duplicate definitions in a rather misleading way. If the module-path is specified in the compilations rather than using the (un-documented ?) default search everything works.

  SPR Number: 102888, 29-APR-96
  SPR Status: UNDER-INVESTIGATION

- **CF90 inlining problems**:

  Using -O inline3 ends up in lots of nonsens error messages.

  SPR Number: 105898 , 27-AUG-96
  SPR Status: UNDER-INVESTIGATION

Source zu Fehler 4 (Bound Checking)

```
PROGRAM bounds_check
IMPLICIT NONE
INTEGER :: i, nep, nem, k1, le_20, le20
INTEGER :: leaa, one=1
INTEGER, DIMENSION(20)  :: k
CHARACTER(-20)          :: ch_20
CHARACTER(20)           :: ch20
CHARACTER(*),PARAMETER  :: chaa='aa'
!
DO i = 1, 20
  k(i) = 10*i
ENDDO
!
nep = 3 * 7
nem = -1 * nep
WRITE(*,*) 'nep = ', nep,' nem = ', nem
!
WRITE(*,*) 'k(21)  = ', k(nep)      !No warning message
WRITE(*,*) 'Here I referenced the 21st element of array k,'
WRITE(*,*) 'it was neither declared nor defined,'
WRITE(*,*) 'I used -Rbs: Why is there no warning message?'
```

```
!
le_20 = LEN(ch_20)
le20  = LEN(ch20)
leaa  = LEN(chaa)
WRITE(*,*) 'le_20 = ',le_20,' le20 = ',le20,' leaa = ',leaa
!
ch_20 = ch20(nep:nep)   !No warning message for either side
WRITE(*,*) 'This assignment statement is described on page '
WRITE(*,*) '310 of SR-3902 2.0'
!
WRITE(*,*) 'I referenced the 21st character of string ch20,'
WRITE(*,*) 'it was neither declared nor defined,'
WRITE(*,*) 'and I used -Rbs:'
WRITE(*,*) 'Why is there no warning message?'
WRITE(*,*) 'I defined the 1st character of string ch_20,'
WRITE(*,*) 'it was declared with length 0, and I used -Rbs:'WRITE(*,*)
'Why is there no warning message?'
ch_20(one:one) = ch20(nep:nep)   !No warning message for RHS
WRITE(*,*) 'I referenced the 21st character of string ch20,'
WRITE(*,*) 'it was neither declared nor defined, '
WRITE(*,*) 'I used -Rbs: Why is there no warning message?'
```

```
ch20(nep:nep+2) = '+21'           !No warning message
WRITE(*,*) 'I defined the 21-23rd characters of string '
WRITE(*,*) 'ch20, they were not declared, and I used -Rbs:'
WRITE(*,*) 'Why is there no warning message?'
!
WRITE(*,*) 'ch20(nep:nep+2) = ', ch20(nep:nep+2)
WRITE(*,*) 'ch20(nem:nem+2) = ', ch20(nem:nem+2)
                               !No warning messages
WRITE(*,*) 'I referenced some characters of string ch20,'
WRITE(*,*) 'they were not declared, and I used -Rbs:'
WRITE(*,*) 'Why is there no warning message?'
!
END PROGRAM bounds_check
```

Weitere Problemfälle (durch die -O Option)

```
f90 -r0 -Rabcs -O 0 b_check.f90 && a.out
```

liefert u.a. die Ausgabe k(21) = 0, keine Warnung bzgl. dieser Grenzüberschreitung, dafür aber eine Warnung bzgl. der Verwendung von ch_20(one:one) mit der Bemerkung: Parent string size is 0.

```
f90 -r0 -Rabcs -O 3 b_check.f90 && a.out
```

liefert u.a. die Ausgabe k(21) = -21, eine Warnung bzgl. dieser Grenzüberschreitung, dafür aber keine Warnung bzgl. der Verwendung von ch_20(one:one).

Source zu Fehler 5 (Default Module Search: Variant on Example 4 on page 50-51 of the manual CF90 Commands and Directives Reference Manual SR-3901 2.0*):*

progone.f90:

```
MODULE split
INTEGER  :: k
REAL     :: a
END MODULE split
!
PROGRAM demo
USE split
INTEGER  :: j
j = 3; k = 1; a = 2.0
CALL suba(j)
PRINT *,'j=',j; PRINT *,'k=',k; PRINT *,'a=',a;
END PROGRAM demo
```

progtwo.f90:

```
SUBROUTINE suba(l)
USE split
INTEGER  :: l
l = 4; k = 5
CALL subb(l)
END SUBROUTINE suba
!
SUBROUTINE subb(m)
USE split
INTEGER  :: m
m = 6; a = 7.0
END SUBROUTINE subb
```

Übersetzen mit:

```
f90 -p progone.o progone.f90 progtwo.f90
```

ohne Probleme, aber (z.B. im Makefile)

```
f90 -c progone.f90
f90 -c progtwo.f90
f90 progone.o progtwo.o
```

liefert:

```
ldr-239 f90: CAUTION
More than one primary entry point has been encountered.
Primary entry 'DEMO' in module 'DEMO' from file 'progone.o' was not used.

ldr-290 f90: CAUTION
Duplicate entry point 'SPLIT' was encountered.
Entry in module 'SPLIT' from file 'progone.o' has been used.
Entry in module 'SPLIT' from file 'progone.o' has been ignored.

ldr-290 f90: CAUTION
Duplicate entry point 'DEMO' was encountered.
Entry in module 'DEMO' from file 'progone.o' has been used.
Entry in module 'DEMO' from file 'progone.o' has been ignored.
```

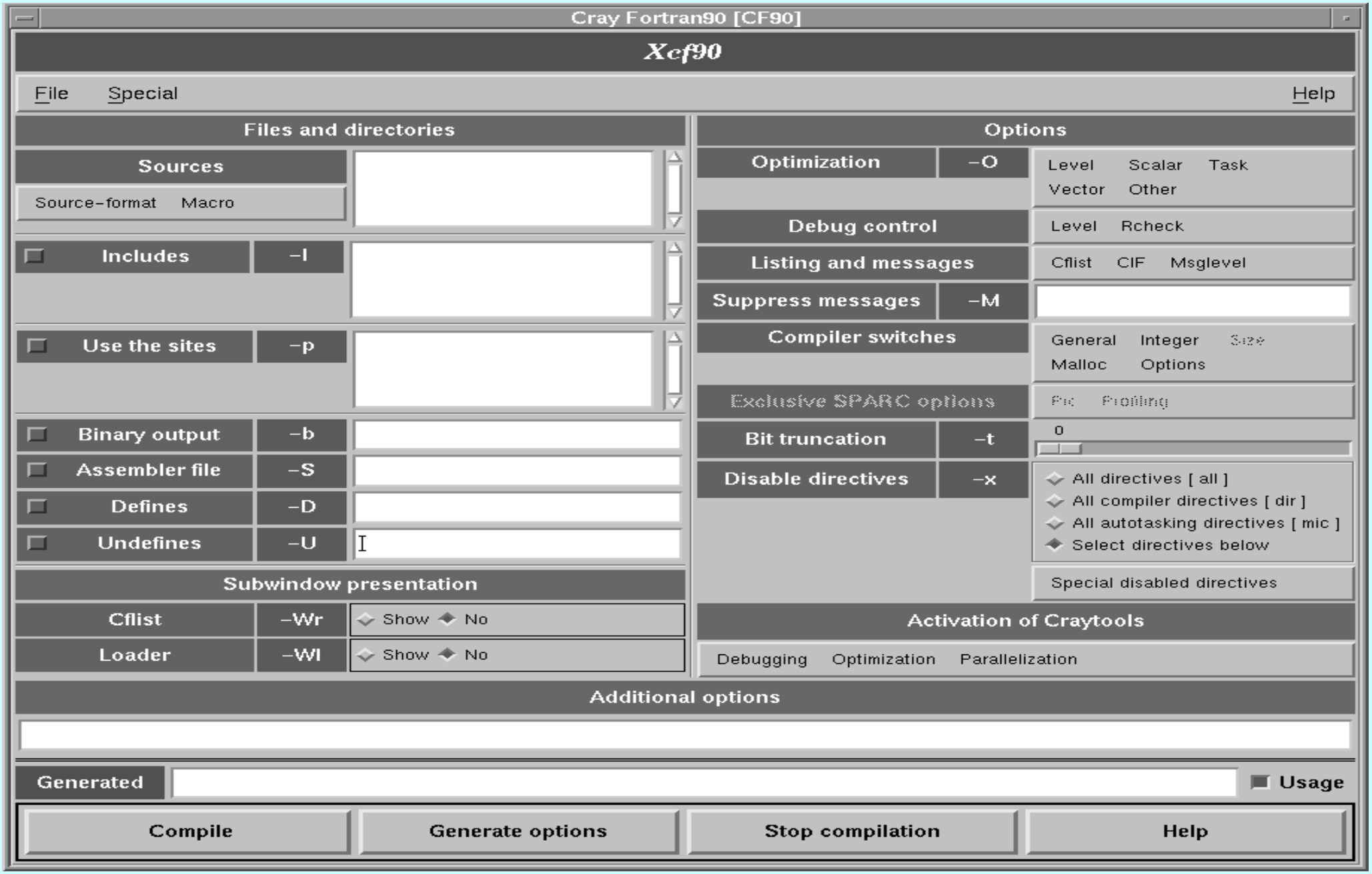
Grund:

Offenbar wird progone.o zweimal durchsucht. Gleiches gilt, wenn die Option -p progone.o verwendet wird. Keine Meldung jedoch bei:

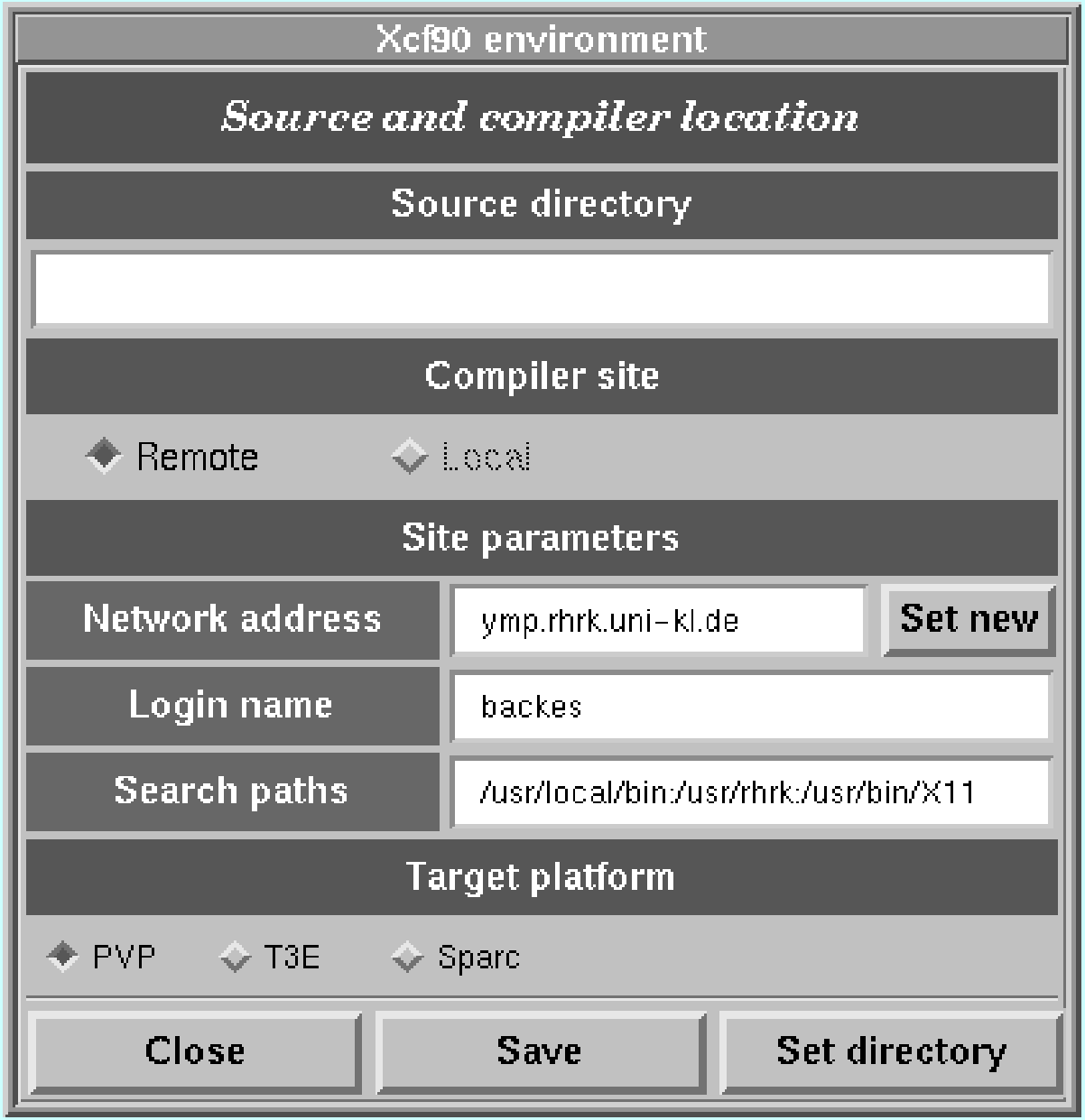
```
f90 -p progone.o progtwo.o
```

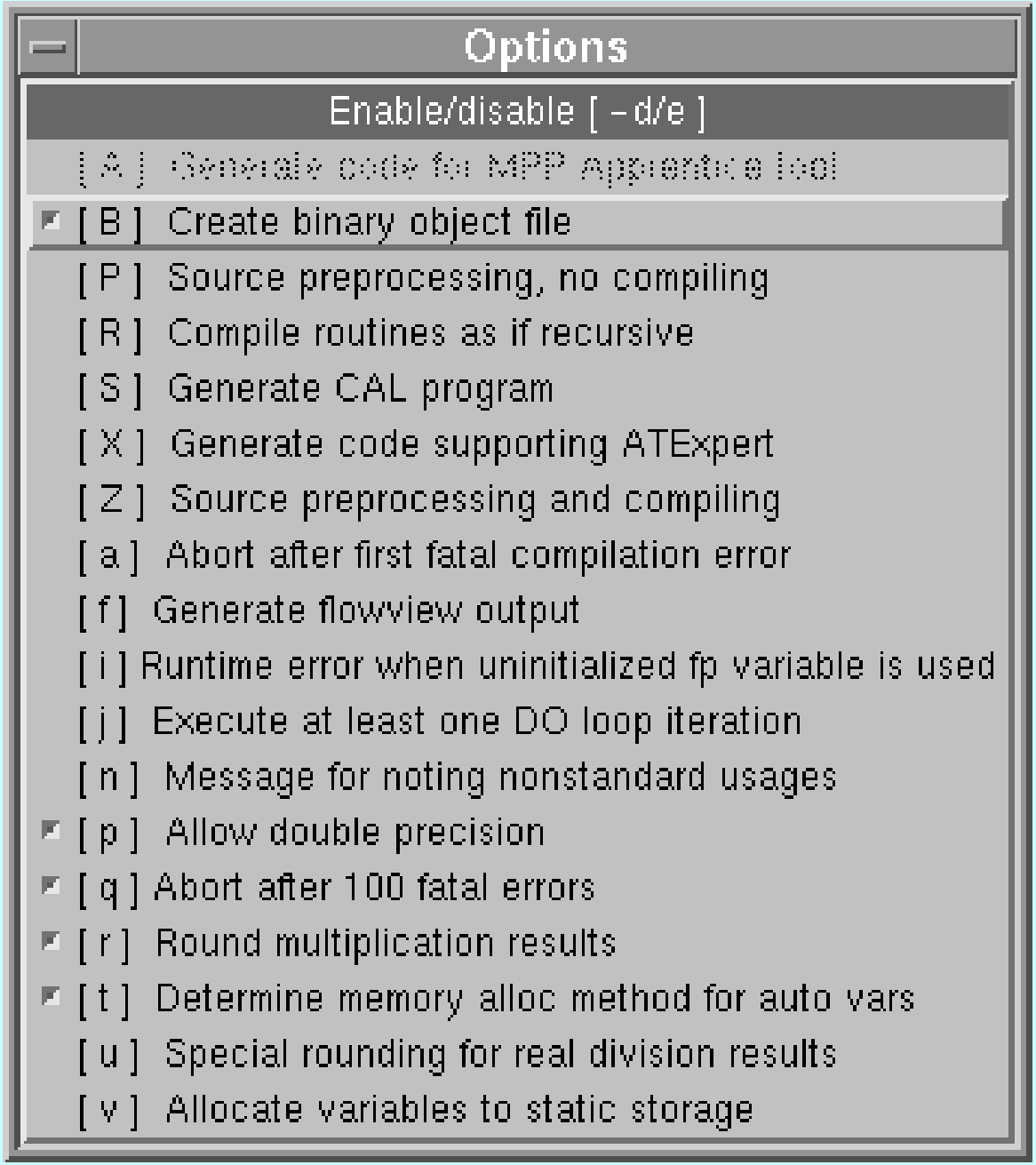
Teil 2: Xcf90
ein GUI zu CRAY f90
(Joachim Backes)

CRAY Fortran 90, erste Erfahrungen und GUI's



CRAY Fortran 90, erste Erfahrungen und GUI's




Options
Enable/disable [-d/e]
[A] Generate code for MPP Apprentice tool
[B] Create binary object file
[P] Source preprocessing, no compiling
[R] Compile routines as if recursive
[S] Generate CAL program
[X] Generate code supporting ATExpert
[Z] Source preprocessing and compiling
[a] Abort after first fatal compilation error
[f] Generate flowview output
[i] Runtime error when uninitialized fp variable is used
[j] Execute at least one DO loop iteration
[n] Message for noting nonstandard usages
[p] Allow double precision
[q] Abort after 100 fatal errors
[r] Round multiplication results
[t] Determine memory alloc method for auto vars
[u] Special rounding for real division results
[v] Allocate variables to static storage

CRAY Fortran 90, erste Erfahrungen und GUI's