THEORY III

- Indirect addressing
- Performance analysis
- General procedure for tuning your code

Indirect addressing

This can prevent vectorization !

directive: indirect addressing

- FORTRAN:

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

i	j
1	3
2	4
3	5
4	6
5	7

- Compiler can not decide whether j can have same values
- simple case: ija is injective
  - use directive

```
!cdir nodep
      do i=1,n
        j=ija(i)
        a(j)=a(j)+b(i)
      end do
```


Non-injective index

i	j
1	3
2	2
3	1
4	3
5	4
6	1
7	1
8	3
9	2
10	3
11	1
12	3

problem

```
a(3)=a(3)+b(1)
a(2)=a(2)+b(2)
a(1)=a(1)+b(3)
a(3)=a(3)+b(4)
a(4)=a(4)+b(5)
a(1)=a(1)+b(6)
a(1)=a(1)+b(7)
a(3)=a(3)+b(8)
a(2)=a(2)+b(9)
a(3)=a(3)+b(10)
a(1)=a(1)+b(11)
a(3)=a(3)+b(12)
```

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

Using directive is not correct!

Non-injective Index (cont.)

- Difficult case
- several potential solutions
  - with setup phase, requires constant index vector
  - without setup phase
    - more difficult
    - several ways, depending on the case

Non-injective Index (cont.)

- **without setup phase**

➢ Dynamic detection of overwriting: Loop is executed as before: results are corrected afterwards

➢ Vector distribution of results: Use additional injective dimensions

➢ Use compiler option: -Wf,-pvctl listvec

When an array element with a vector subscript appears both of left and right hand side of an assignment statement, the statement is vectorized.

X(IX(I)) = X(IX(I)) + ... is vectorized.

Non-injective Index without setup phase

Dynamic detection of overwriting:

Loop is executed as before: results are corrected afterwards

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

```
do i=1,n
  j=ija(i)
  tmp(i)= a(j)+b(i)
enddo
do i=1,n
  j=ija(i)
  a(j)=i
end do
```

Calculate A+B and mark positions

```
do i=1,n
 j=ija(i)
 if(a(j).ne.i) then
  icnt=icnt+1
  list(icnt)=i
 endif
 a(j)=tmp(i)
end do
```

Detect overwritings and put them into a list

```
do k=1,icnt
 i=list(k)
 j=ija(i)
  tmp(i)= a(j)+b(i)
  a(j)=i
end do
```

...

Process list as before, repeat until list empty

Non-injective Index without setup phase

Vector distribution of results:

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

```
dimension a(m), av(512,m)
av=0.0
do i=1,n
 iv = mod(i-1,512)+1
 av(iv,ija(i))=av(iv,ija(i))+b(i)
enddo
do iv=1,512
 do j=1,m
  a(j)=a(j)+av(iv,j)
 end do
end do
```

This approach makes sense only for n >> m !

Non-injective Index without setup phase

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

```
PROGRAM NAME: ex1
FORMAT LIST

  LINE    LOOP     FORTRAN STATEMENT

     1:                  program ex1
     2:                  real, dimension(200) :: a ,b,ija
     3:                  real, dimension(200,4) :: av
     4: +V=====          av=0.0
     5: V------>         do i=1,n
     6: |                  iv = mod(i-1,256)+1
     7: |       S          av(iv,ija(i))=av(iv,ija(i))+b(i)
     8: V------          end do
     9: X------>         do iv=1,256
    10: |+----->           do j=1,m
    11: ||                    a(j)=a(j)+av(iv,j)
    12: |+-----            end do
    13: X------          end do
    14:                  end program
```

Empowered by Innovation NEC

Non-injective Index without setup phase

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

```
PROGRAM NAME: ex1
FORMAT LIST

  LINE    LOOP     FORTRAN STATEMENT

     1:                  program ex1
     2:                  real, dimension(200) :: a ,b,ija
     3:                  real, dimension(200,4) :: av
     4: +V=====          av=0.0
     5:            !CDIR NODEP
     6: V------>         do i=1,n
     7: |                  iv = mod(i-1,256)+1
     8: |                  av(iv,ija(i))=av(iv,ija(i))+b(i)
     9: V------          end do
    10: X------>         do iv=1,256
    11: |+----->           do j=1,m
    12: ||                    a(j)=a(j)+av(iv,j)
    13: |+-----            end do
    14: X------          end do
    15:                  end program
```

Empowered by Innovation NEC

directive: 2d addressing

If: $m \cong n$
Many different values for j
Many overwritings

Choose with setup

Non-injective Index with setup phase

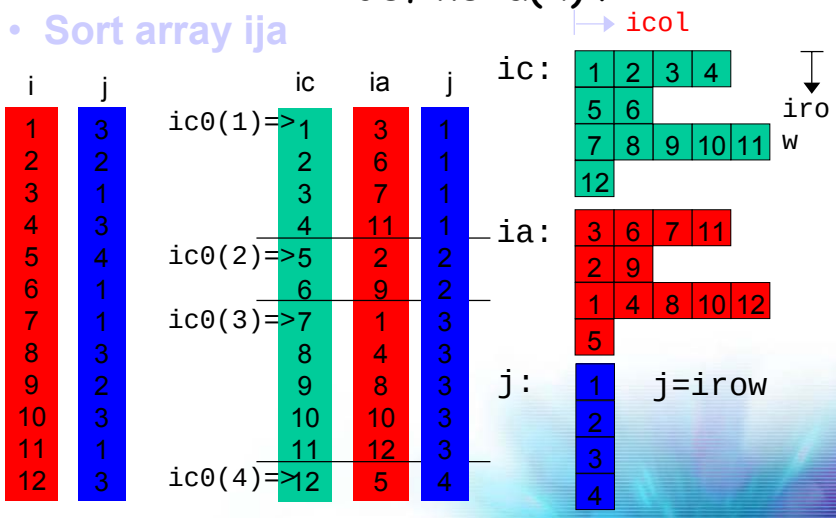
```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

- Sort array ija, if only few values: update for each value
- Use problem-specific structure of ija as Multicoloring or hyperplanes → injective
- JAD: reorder ija into injective chunks and vectorize over the chunks (reordering routine(s) available on request) → injective

indirect addressing (8)

define ia(x) !

- Sort array ija

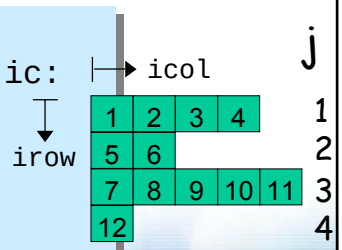


Non-injective Index with setup phase

Sort array
vectorize over icol

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

```
      do irow=1,nrow
        as = 0
        j = irow
V     do icol=1,istop(irow)
        ic=ic0(irow) + icol-1
        as=as+b(ia(ic))
V     end do
        a(j)=a(j)+as
      end do
```



Makes sense for large number of icol

indirect addressing

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

- Method: Multicoloring
  - use problem-specific structure of `ija`
  - Create Groups (Colors) by collecting as many values of `i` as possible - all mapped to different values of `j`

indirect addressing

- **Multicoloring**

i	j
1	3
2	2
3	1
4	3
5	4
6	1
7	1
8	3
9	2
10	3
11	1
12	3

	ic	ia	ja
ic0(1)=>	1	1	3
	2	2	2
	3	3	1
	4	5	4
ic0(2)=>	5	4	3
	6	6	1
	7	9	2
ic0(3)=>	8	7	1
	9	8	3
ic0(4)=>	10	10	3
	11	11	1
ic0(5)=>	12	12	3

icolor →

ic:	1	5	8	10	12
	2	6	9	11	
	3	7			
	4				
ia:	1	4	7	10	12
	2	6	8	11	
	3	9			
	5				
ja:	3	3	1	3	3
	2	2	3	1	
	1	1			
	4				

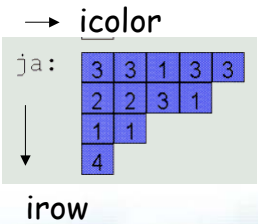
irow

Non-injective Index with setup phase

Multicoloring
vectorize over irow

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

```
      do icolor=1,ncolor
V  ——   do irow=1,istop(icolor)
|         ic=ic0(icolor) + irow-1
|         j=ja(ic)
|         a(j)=a(j)+b(ia(ic))
V  ——   end do
      end do
```



indirect addressing

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```

Method: JAD

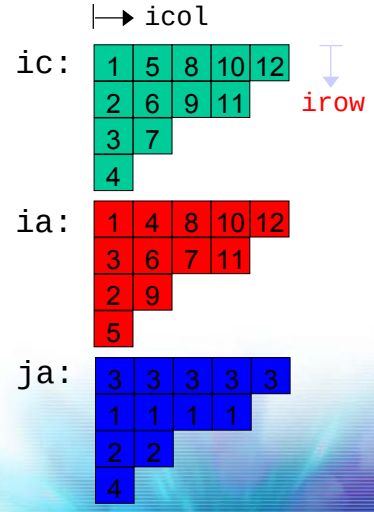
- Find the `j` values with the highest number of `i` elements mapped onto. Create the according number of groups and distribute the remaining `i` values in a injective manner accordingly.

indirect addressing (14)

3 occurs at 5 places, 4 only once

- JAD ordering (1/2)

i	j		ic	ia	ja
1	3	ic0(1)=>	1	1	3
2	2		2	3	1
3	1		3	2	2
4	3		4	5	4
5	4	ic0(2)=>	5	4	3
6	1		6	6	1
7	1		7	9	2
8	3	ic0(3)=>	8	8	3
9	2		9	7	1
10	3	ic0(4)=>	10	10	3
11	1		11	11	1
12	3	ic0(5)=>	12	12	3



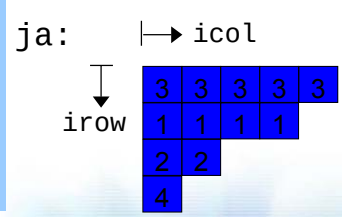
indirect addressing (15)

- JAD ordering (2/2)

outer loop over icol
inner loop over irow:

```
   do icol=1,ncol
V—  do irow=1,istop(icol)
|      ic=ic0(icol)+irow-1
|      j=ja(ic)
|      a(j)=a(j)+b(ia(ic))
V—  end do
   end do
```

```
do i=1,n
  j=ija(i)
  a(j)=a(j)+b(i)
end do
```



directive: 2d addressing

- another case for the nodep:

```
do j = 2, n
  j1 = ind(j  )
  j2 = ind(j-1)
  do i = 2, n
    a(i,j1) = a(i1,j2)*b(i,j) + c(i,j)
  end do
end do
```

- Problem?
- Compiler does it - with !CDIR NODEP

How to measure performance

Analysis of Performance

F_PROGINF and C_PROGINF

- Program Execution Summary based on CPU built-in hardware counters
- Performance: MOPS, MFLOPS, MIPS
- Times: User, Real, System, Vector
- operations: vector, floating point …
- cache info, bank conflicts, …
- multitasking info
- set environment variables in jobscript ksh: export F_PROGINF detail

Analysis of Performance

RECO

F_

C_

De

```
******  Program Information  ******
Real Time (sec)        :        142.373845
User Time (sec)        :        122.693250
Sys  Time (sec)        :          1.331073
Vector Time (sec)      :         65.895359
Inst. Count            :       19621582641.
V. Inst. Count         :        2791820381.
V. Element Count       :      216956074637.
FLOP Count             :       97760445434.
MOPS                   :       1905.449870
MFLOPS                 :        796.787481
VLEN                   :         77.711330
V. Op. Ratio (%)       :         92.801205
Memory Size (MB)       :        496.031250
MIPS                   :        159.923897
I-Cache (sec)          :          1.852328
O-Cache (sec)          :         15.546471
Bank (sec)             :          3.069632

Start Time (date)  :  2002/06/24 11:32:19
End   Time (date)  :  2002/06/24 11:34:42
```

Analysis of Performance

Another tool: prof

- Extended Unix profiler
- link program with –p
- run program (it generates a file mon.out)
- generate profile:
  prof prog_name > profile
- info on:
  - time spent in each subroutine (including library routines)
  - for microtasked routines: time spent in each task
  - F90: number of calls in subroutine

UNIX prof

Output from prof

%Time	Seconds	Cumsecs	#Calls	msec/call	Name	
24.6	373.17	373.17	267320	1.3960	fftstp_	******
20.3	307.71	680.89			dgemm_	*****
7.6	115.10	795.98	87134	1.3209	fftpre_	*
6.7	101.77	897.75			sgemv_	*
4.7	70.70	968.45	53431	1.3232	azzero_	*
3.9	58.46	1026.91	46526	1.2564	fftrot_	
2.5	38.37	1065.28			dgemv_	
2.4	37.12	1102.40	17998	2.0625	matmov_	
2.1	32.24	1134.63	257	125.43	vpsi_	
1.8	26.87	1161.50	141	190.57	rhoofr_	
1.6	23.58	1185.08	275088	0.0857	syssx	
1.5	23.36	1208.45			ew_cdabs	
1.5	22.58	1231.03	142	159.01	eicalc_	
1.3	20.39	1251.42	828	24.63	idcopy_	
1.2	17.64	1269.05	1202784	0.0147	zsctr_	
1.1	16.15	1285.20			MPID_SHMEM_Check_incoming	
1.0	15.81	1301.01	57831	0.2734	mltfft_	
1.0	15.18	1316.19			mpisx_memcpy_ok	
1.0	15.02	1331.21			dcopy_	
0.9	12.93	1344.13			mpisx_memcpy	
0.5	8.27	1352.40	141	58.62	odiis_	

Analysis of Performance

Another tool: loopprof

- Profiler on loop-level
- compile and link program with –loopprof
- run program (it generates a file *.pdf)
- info on:
  - time spent in each loop
  - performance for each loop (MOPS, MFLOPs, Vector Length, …)
  - …

Empowered by Innovation NEC

Analysis of Performance

Another tool: ftrace

```
*-------------------------*
  FLOW TRACE ANALYSIS LIST
*-------------------------*

Execution : Wed Feb 20 11:23:37 2002
Total CPU : 0:26'08"803


PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER. I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN  MISS    MISS      CONF

lti              288   309.642( 19.7)  1075.147  778.6  270.6 87.46  38.5  0.0887  0.3532  0.3303
ltd              287   228.125( 14.5)   794.859  766.3  271.4 87.73  38.2  0.0120  0.2744  0.0000
getflxa2       43624   154.688(  9.9)     3.546 7586.7 1957.1 99.09 159.3  0.0598  7.3855  0.0011
rrtm_rtrn1a_140gp
                3840   148.032(  9.4)    38.550 7649.2 2279.2 99.64 165.6  0.0631  0.0182  0.0751
cloud          22720    79.036(  5.0)     3.479 6151.5 2127.7 99.43 124.8  0.4365  0.2270  0.5416
vdiff          45920    73.244(  4.7)     1.595 6939.4 2982.5 99.31 160.3  1.8788  0.6911  0.3684
…
-----------------------------------------------------------------------------------------------
total       13734088  1568.803(100.0)     0.114 4036.5 1251.0 98.11 127.1 16.3883 24.0532 16.1665
```

Be careful: Overhead

Empowered by Innovation NEC

Analysis of Performance

Another tool: ftrace on loop-level

- Routine- and Loop-based performance analysis
- compile and link program with –ftrace
- run program (it generates a file ftrace.out)
- run command ftrace

```
CALL FTRACE_REGION_BEGIN (“REGION A“)

        DO …
        …
        ENDDO

CALL FTRACE_REGION_END (“REGION A“)
```

Empowered by Innovation NEC

Analysis of Performance

ftrace on loop-level

```
*--------------------------*
  FLOW TRACE ANALYSIS LIST
*--------------------------*
Execution : Mon Jan 29 23:38:35 2001
Total CPU : 0:00'00"004
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN    TIME   MISS    MISS      CONF
sub               1     0.003( 63.7)     2.800   35.7    0.0  0.00   0.0    0.000  0.0000  0.0000  0.0000
main              1     0.002( 36.3)     1.598 1307.7    0.0 97.34 255.9    0.001  0.0002  0.0001  0.0000
-----------------------------------------------------------------------------------------------------------
total             2     0.004(100.0)     2.199  497.9    0.0 92.90 255.9    0.001  0.0002  0.0001  0.0000
REGION_A          1     0.001( 22.7)     0.998 2043.2    0.0 98.08 256.0    0.000  0.0000  0.0000  0.0000
-----------------------------------------------------------------------------------------------------------
```

Be careful: Overhead

Empowered by Innovation NEC

QUESTIONS ?

Empowered by Innovation NEC

SUPER-UX R12.2 CD-ROM (G1AX99E12) Global Contents

Please refer to

General Usage
- User's Guide
- User's Reference Manual

Programming
- Programmer's Guide
- Programmer's Reference Manual

Batch Processing
- NQS User's Guide

Window System
- X Windows System User's Guide
- Xlib Programming Manual
- X Toolkit Programming Manual
- X Window System Programmer's Guide
- Mf/SX User's Guide
- Mf/SX Programmer's Guide
- Mf/SX Style Guide
- Mf/SX Programmer's Reference

Language
- Programming Language Support Reference Manual
- C Programmer's Guide
- FORTRAN90/SX Language Reference Manual
- FORTRAN90/SX Programmer's Guide
- FORTRAN90/SX Multitasking User's Guide
- MPI/SX User's Guide
- Assembly Language Reference Manual
- DBX User's Guide
- PDBX User's Guide
- XDBX User's Guide
- PSUITE User's Guide
- C++/SX Programmer's Guide
- FSA/SX User's Guide

NEC

Tuning your code !

Examples

- *How to prevent vectorization*
- *How to read compiler messages*
- *How to solve*

Code Tuning
ACHIEVING SPEED THROUGH VECTORIZATION

The most important consideration is to maximize the vectorization ratio of the part processed by vector instructions. The next most important point is to maximize the efficiency of generated vector instructions.

V. Op. R > 99% VLEN near 256

Typical Output for PROGINF

```
******  Program Information  ******
  Real Time (sec)        :       4979.666437
  User Time (sec)        :       4776.461378
  Sys  Time (sec)        :        113.101447
  Vector Time (sec)      :       4730.900352
  Inst. Count            :      520073756153.
  V. Inst. Count         :      186315208781.
  V. Element Count       :    38459109019585.
  FLOP Count             :    17025664866477.
  MOPS                   :       8121.675127
  MFLOPS                 :       3564.493360
  MOPS   (concurrent)    :       8121.675183
  MFLOPS (concurrent)    :       3564.493385
  VLEN                   :        206.419590
  V. Op. Ratio (%)       :         99.139639
  Memory Size (MB)       :       1984.000000
  Max Concurrent Proc.   :                 1.
   Conc. Time(>= 1)(sec):        4776.461345
  Event Busy Count       :                 0.
  Event Wait (sec)       :          0.000000
  Lock Busy Count        :                 0.
  Lock Wait (sec)        :          0.000000
  Barrier Busy Count     :                 0.
  Barrier Wait (sec)     :          0.000000
  MIPS                   :        108.882647
  MIPS (concurrent)      :        108.882647
  I-Cache (sec)          :          3.205337
  O-Cache (sec)          :         16.633047
  Bank (sec)             :         95.908582
```

ex
vx
ve

csh
setenv F_PROGINF DETAIL
setenv C_PROGINF DETAIL
setenv MPIPROGINF DETAIL
ksh
export F_PROGINF=DETAIL
export C_PROGINF=DETAIL
export MPIPROGINF=DETAIL

Vector operation rate= 100.0*ve/(ex-vx+ve)

Vectorlength=ve/vx

Typical Output for MPIPROGINF

setenv MPIPROGINF DETAIL

```
Global Data of 4 processes  :          Min [U,R]          Max [U,R]      Average
==========================

Real   Time (sec)           :     1306.891 [0,3]     1306.996 [0,0]     1306.943
User   Time (sec)           :     1283.438 [0,0]     1290.910 [0,3]     1287.639
System Time (sec)           :        3.670 [0,3]       10.421 [0,0]        5.681
Vector Time (sec)           :     1219.680 [0,3]     1230.599 [0,1]     1225.969
Instruction Count           : 108550870867 [0,0] 111212591406 [0,3] 110070143257
Vector Instruction Count    :  38001056352 [0,3]  39420819524 [0,1]  38949507530
Vector Element Count        : 8136381841736 [0,3] 8483381897138 [0,1] 8362206479711
FLOP Count                  : 3151677793786 [0,3] 3291738261371 [0,0] 3242912009747
MOPS                        :     6359.541 [0,3]     6659.359 [0,0]     6549.647
MFLOPS                      :     2441.439 [0,3]     2564.782 [0,0]     2518.576
Average Vector Length       :      214.056 [0,2]      215.382 [0,0]      214.687
Vector Operation Ratio (%)  :       99.108 [0,3]       99.190 [0,0]       99.156
Memory size used (MB)       :     2037.438 [0,1]     2230.656 [0,0]     2131.293
MIPS                        :       84.578 [0,0]       86.151 [0,3]       85.481
Instruction Cache miss (sec):        3.029 [0,3]        3.559 [0,0]        3.331
Operand     Cache miss (sec):        9.788 [0,3]       10.144 [0,0]        9.922
Bank     Conflict Time (sec):       57.121 [0,0]       62.350 [0,3]       60.371
```

```
*--------------------------*
  FLOW TRACE ANALYSIS LIST
*--------------------------*

Execution : Wed Feb 20 11:23:37 2002
Total CPU : 0:26'08"803


PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP  AVER. I-CACHE O-CACHE    BANK
                      TIME[sec](  % )    [msec]               RATIO V.LEN  MISS    MISS      CONF

lti              288   309.642( 19.7)  1075.147  778.6  270.6 87.46  38.5  0.0887  0.3532  0.3303
ltd              287   228.125( 14.5)   794.859  766.3  271.4 87.73  38.2  0.0120  0.2744  0.0000
getflxa2       43624   154.688(  9.9)     3.546 7586.7 1957.1 99.09 159.3  0.0598  7.3855  0.0011
rrtm_rtrn1a_140gp
                3840   148.032(  9.4)    38.550 7649.2 2279.2 99.64 165.6  0.0631  0.0182  0.0751
cloud          22720    79.036(  5.0)     3.479 6151.5 2127.7 99.43 124.8  0.4365  0.2270  0.5416
vdiff          45920    73.244(  4.7)     1.595 6939.4 2982.5 99.31 160.3  1.8788  0.6911  0.3684
cover          22720    47.296(  3.0)     2.082 3814.1  628.3 98.87  77.1  0.1316  0.0623  0.1897
mo_ssortns.orodrag
               45920    38.570(  2.5)     0.840 3069.0  726.9 96.06  95.3  1.3561  2.8082  0.0249
cfdotmc       183680    30.658(  2.0)     0.167 10539.6 2768.4 99.65 160.0  0.1346  0.0013  0.0000
cfintv       3306240    25.504(  1.6)     0.008 9180.9 2115.7 99.77 162.4  0.0007  0.0007  0.0000
rrtm_gasabs1a_140gp
                3840    23.084(  1.5)     6.011 2589.2  849.6 99.22 160.2  0.0490  0.0171  0.3042
physc          45920    19.553(  1.2)     0.426 2361.4  478.1 94.42 159.0  2.9390  1.9469  0.0000
swni           11520    15.952(  1.0)     1.385 6422.6 2988.1 98.73 161.4  0.4019  0.1625  0.0000
cuasc          45440    15.856(  1.0)     0.349 6544.2 1616.7 99.05 161.5  0.5072  0.3636  0.0001
rrtm_taumol3
                3840    14.337(  0.9)     3.734 3130.4  826.1 99.83 160.7  0.0501  0.0066  0.6868
rrtm_taumol5
                3840    13.990(  0.9)     3.643 2918.7  760.9 99.84 160.6  0.0460  0.0027  0.6278
cuadjtq      1133297    12.856(  0.8)     0.011 5915.2 1520.3 99.49 160.0  0.3687  0.1880  0.1932
```

FTRACE: select code

- Measurement of single loops

```
PROGRAM MAIN
INTEGER A(10000)
... other code
CALL FTRACE_REGION_BEGIN("REGION_A")
DO I=1,10000
  A(I)=I
ENDDO
CALL FTRACE_REGION_END("REGION_A")
... other code
STOP
END
```

General procedure for all applications HIGHLY RECOMMENDED

- use setenv F_PROGINF DETAIL, F_FTRACE YES
- Compile and link your code with -ftrace
- Compile with
  -Wf,-L fmtlist map summary transform
- After optimization compile without –ftrace!

Examples

EXAMPLES OF TYPICAL USER CODES

STATEMENTS F77 → F90

Do not use	DIMENSION	REAL, DIMENSION (20) :: help
Do not use	DATA	INTEGER :: a2 = 523
Do not use	PARAMETER	INTEGER, PARAMETER :: par = 5
Do not use	COMMON	MODULES

STATEMENTS F77 → F90

Do not use	EQUIVALENCE	ALLOCATE
Do not use	ENTRY	*Redesign!*
Do not use	ARITHMETIC IF IF (expression) label1, label2, label3 (-1,0,1)	
Do not use	END	END SUBROUTINE, END MODULE…

STATEMENTS F77 → F90

COMPUTED GO TO CASE

```
INTEGER :: value
...
SELECT CASE (value)
CASE (1,3,5,7,9)
   WRITE (*,*) `The value is odd‘
CASE (2,4,6,8,10)
   WRITE (*,*) `The value is even‘
CASE (11:)
   WRITE (*,*) `The value is too high‘
CASE DEFAULT
   WRITE (*,*) `THe value is negative or zero‘
```


STATEMENTS F77 → F90

COMPUTED GO TO CASE

DO 100 i= ….	DO i=1…
…	…
100 CONTINUE	END DO

Example 1

very inefficient due to formatted I/O within in a loop and if-block

```
37: V------>          GP1(:)=0.
38: V------           P(:)=1.
39:
44:
45:                   OPEN(10,file='greenout.d',form='formatted')
51:                   print *,"dGu(40,20,351)=",dGu(40,20+350*IE)
52: V------>          DO 100 IJ1=IJUE+1,IJUE+IJVE
53: |           c      if (GP1(IJ1)<-1.E-16) then
54: |        S         WRITE (10,7776) (IA(IJ1)+10), JA(IJ1), GP1(IJ1)
55: |           c      end if
56: V------      100  END DO
57:                   CLOSE(10)
58:              7776 FORMAT(2I6,1E15.4)
59:                   END
```


Example 2

Is this message important?

```
LINE  LEVEL( NO.): DIAGNOSTIC MESSAGE

73  vec  (   3): Unvectorized loop.
73  vec  (  13): Overhead of loop division is too large.
74  opt  (1017): Subroutine call prevents optimization.
74  vec  (  17): Unvectorizable statement.
78  vec  (   2): Partially vectorized loop.
78  vec  (  25): Work vectors are used. Size=7200byte
82  opt  (1036): Potential feedback - use directive if OK.
82  opt  (1036): Potential feedback - use directive if OK.
82  vec  (  22): Dependency unknown. Unvectorizable dependency is assumed.:dgv
83  opt  (1033): Potential multiple store conflict -- use directive if OK.
84  opt  (1036): Potential feedback - use directive if OK.
84  opt  (1036): Potential feedback - use directive if OK.
84  vec  (  22): Dependency unknown. Unvectorizable dependency is assumed.:dgu
85  opt  (1033): Potential multiple store conflict -- use directive if OK.
88  opt  (1589): Outer loop moved inside inner loop(s).
89  warn (  81): Name "j" is referenced but not defined.
```

Example 2

```
70:                  open(8 ,file='dgld1.d',   form='unformatted')
72:                  close (8)
73: +------>         DO 7099 L=1,IE
74: |                 print *,dGVs(80,L,180)
75: +------     7099 END DO
76: +------>           DO 7010 I=1,IE
77: |+----->            DO 7011 K=1,IE
78: ||V---->             DO 7009 L=1,JE
79: |||                    GKALA=K+(L-1)*IE
80: |||                    KOMP1=modulo(GKALA-J*IE,(IE*JE))
81: |||                    IF (KOMP1==0) KOMP1=IE*JE
82: |||     S              dGv(I,KOMP1)=dGvs(I,K,L)
83: |||     S              dGv(I,KOMP1+IE*JE)=dGv(I,KOMP1)
84: |||     S              dGu(I,KOMP1)=dGus(I,K,L)
85: |||     S              dGu(I,KOMP1+IE*JE)=dGu(I,KOMP1)
86: ||V----     7009     END DO
87: |+-----     7011    END DO
88: +------     7010 END DO
89:                  END
```


Example 2

```
73:            c     DO 7099 L=1,IE
74: V======           print *,dGVs(80,:,180)
75:            c7099 END DO
76: +------>           DO  I=1,IE
77: |+----->            DO  K=1,IE
78: ||V---->             DO  L=1,JE
79: |||                    GKALA=K+(L-1)*IE
80: |||                    KOMP1H(L)=modulo(GKALA-J*IE,(IE*JE))
81: |||                    IF (KOMP1H(L)==0) KOMP1H(L)=IE*JE
82: ||V----              ENDDO
83: ||         c
87: ||V---->             DO 7010 L=1,JE
88: |||                    dGv(I,KOMP1H(L))=dGvs(I,K,L)
89: |||                    dGv(I,KOMP1H(L)+IE*JE)=dGv(I,KOMP1H(L))
90: |||                    dGu(I,KOMP1H(L))=dGus(I,K,L)
91: |||                    dGu(I,KOMP1H(L)+IE*JE)=dGu(I,KOMP1H(L))
92: ||V----     7010     END DO
93: |+-----              END DO
94: +------            END DO
```

Example 3

```
115: V------>          DO 7100 IJ1=1,IJE
116: |                  P(IJ1)=GCG(IJ1)*PI(IJ1)
117: V------      7100 END DO
118: +W=====           neuP(:,:)=0.
119:             C
120: V------>          DO 7060 KL1=IJUE+IJVE+1,IJE
121: |                    K=GIA(KL1)
122: |                    L=GJA(KL1)
123: |        S           neuP(K   ,L   )=P(KL1)
124: |        S           neuP(K   ,L+JE)=P(KL1)
125: |        S           neuP(K+IE,L   )=P(KL1)
126: |        S           neuP(K+IE,L+JE)=P(KL1)
127: V------      7060 END DO
```


Example 3

```
115: V------>          DO 7100 IJ1=1,IJE
116: |                  P(IJ1)=GCG(IJ1)*PI(IJ1)
117: V------      7100 END DO
118: +W=====           neuP(:,:)=0.
119:             !CDIR NODEP
120: V------>          DO 7060 KL1=IJUE+IJVE+1,IJE
121: |                    K=GIA(KL1)
122: |                    L=GJA(KL1)
123: |                    neuP(K   ,L   )=P(KL1)
124: |                    neuP(K   ,L+JE)=P(KL1)
125: |                    neuP(K+IE,L   )=P(KL1)
126: |                    neuP(K+IE,L+JE)=P(KL1)
127: V------      7060 END DO
```

Example 4

What does partially mean?

```
DIAGNOSTIC LIST

  LINE  LEVEL( NO.): DIAGNOSTIC MESSAGE

   200  vec  (   4): Vectorized array expression.
   209  vec  (   2): Partially vectorized loop.
   222  vec  (   2): Partially vectorized loop.
   234  vec  (   2): Partially vectorized loop.
   244  vec  (   1): Vectorized loop.
   257  vec  (   1): Vectorized loop.
   263  vec  (   1): Vectorized loop.
^LSunOS  R5.8 FORTRAN90/SX         Rev.254        Mon Feb 17 14:44:16
2003
```


Example 4

```
233: +------>         DO 7066 L=1,lMax
234: |+----->           DO 7065 K1=kMin,RADIUS_I
235: ||V---->            DO 7064 IJ1=IJUE+1,IJUE+IJVE
236: |||                  I=GIA(IJ1)
237: |||                  J=GJA(IJ1)
238: |||                  K=I+K1+IE
239: |||                  KOMP=l+j-1
240: |||     S            neuGP1(K,KOMP)=neuGP1(K,KOMP)+dGv(I,K,L)*P(IJ1)
241: ||V----     7064    END DO
242: |+-----     7065   END DO
243: +------     7066 END DO
245: +------>         DO 7056 L=lMin,JE
246: |+----->            DO 7055 K1=kMin,RADIUS_I
247: ||V---->             DO 7054 IJ1=IJUE+1,IJUE+IJVE
248: |||                   I=GIA(IJ1)
249: |||                   J=GJA(IJ1)
250: |||                   K=I+K1+IE
251: |||                   KOMP=L+J-1
252: |||     S             neuGP1(K,KOMP)=neuGP1(K,KOMP)+dGv(I,K,L)*P(IJ1
253: ||V----     7054     END DO
254: |+-----     7055    END DO
255: +------     7056 END DO
```

Example 4

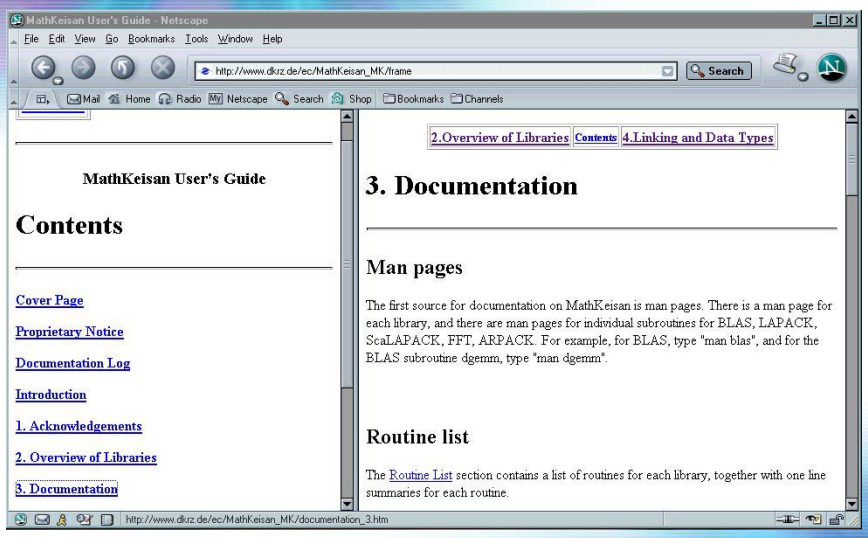
```
232:            !CDIR NODEP
233: X------>         DO 7066 L=1,lMax
234: |+----->            DO 7065 K1=kMin,RADIUS_I
235: ||+---->             DO 7064 IJ1=IJUE+1,IJUE+IJVE
236: |||                     I=GIA(IJ1)
237: |||                     J=GJA(IJ1)
238: |||                     K=I+K1+IE
239: |||                     KOMP=l+j-1
240: |||                     neuGP1(K,KOMP)=neuGP1(K,KOMP)+dGv(I,K,L)*P(IJ1)
241: ||+----     7064        END DO
242: |+-----     7065    END DO
243: X------     7066 END DO
244:            !CDIR NODEP
245: X------>         DO 7056 L=lMin,JE
246: |+----->            DO 7055 K1=kMin,RADIUS_I
247: ||+---->             DO 7054 IJ1=IJUE+1,IJUE+IJVE
248: |||                     I=GIA(IJ1)
249: |||                     J=GJA(IJ1)
250: |||                     K=I+K1+IE
251: |||                     KOMP=L+J-1
252: |||                     neuGP1(K,KOMP)=neuGP1(K,KOMP)+dGv(I,K,L)*P(IJ1)
253: ||+----     7054     END DO
254: |+-----     7055    END DO
255: X------     7056 END DO
```

Libraries for SX-5/6

MathKeisan: Scientific Library for SX-Series

library name	link to
BLAS	-L/opt/MathKeisanx.y.z/lib -lblas
LAPACK	-L/opt/MathKeisanx.y.z/lib -llapack -lblas
FFT	-L/opt/MathKeisanx.y.z/lib -lfft
BLACS	-L/opt/MathKeisanx.y.z/lib -lblacsF90init -lblacs -lblacsF90init -lmpich
ScaLAPACK	-L/opt/MathKeisanx.y.z/lib -lscalapack -lblacsF90init -lblacs -lblacsF90init -lmpich
SOLVER	-L/opt/MathKeisanx.y.z/lib -lsolver -lmetis -lblas
METIS	-L/opt/MathKeisanx.y.z/lib -lmetis_32
	-L/opt/MathKeisanx.y.z/lib -lmetis
ParMETIS	-L/opt/MathKeisanx.y.z/lib -lparmetis_32 -lmpich
	-L/opt/MathKeisanx.y.z/lib -lparmetis -lmpich
ARPACK	-L/opt/MathKeisanx.y.z/lib -larpack -llapack -lblas

Online Documentation (CD)



f90 driver.f libfft.a

The Parallel (OPENMP) I32R32 and I32R64 FFT functionality can be accessed using the following compilation options.

f90 -P openmp driver.f libparfft.a

set the environmental variable OMP_NUM_THREADS to the number of desired parallel Threads.

f90 -ew driver.f libfft_64.a

The Parallel(OPENMP) I64R64 FFT functionality can be accessed using the following compilation options.

f90 -P openmp -ew driver.f libparfft_64.a

set the environmental variable OMP_NUM_THREADS to the number of desired parallel Threads.

NOTE: All FFT's have Parallel Capability (OpenMP) except 1D FFT's.

Online Documentation on FFT

Name	prefixes	description
VECLIB		
_1dfft	c z	1D Complex-Complex FFT, Input/output Complex.
_2dfft	c z	2D Complex-Complex FFT, Input/Output Complex.
_3dfft	c z	3D Complex-Complex FFT, Input/output Complex.
_ffts	c z	Multiple 1D Complex-Complex FFT, Input/output Complex.
_rc1ft	c z	1D Real-Complex, Complex-Real FFT, Input/output Complex.
_rc2ft	c z	2D Real-Complex, Complex-Real FFT, Input/Output Complex.
_rc3ft	c z	3D Complex-Real, Real-Complex FFT, Input/Output Complex.
_rcfts	c z	Multiple 1D Real-Complex, Complex-Real FFT, Input/output Complex.
_1dfft	s d	1D Complex-Complex FFT, input/output Real.
_2dfft	s d	2D Complex-Complex FFT, Input/Output Real.
_3dfft	s d	3D Complex-Complex FFT, Input/output Real.
_ffts	s d	Multiple 1D Complex-Complex FFT, input/output

NEC

man s1dfft

```
S1DFFT(3M)              MLIB Man Pages              S1DFFT(3M)

NAME
     s1dfft, d1dfft - one-dimensional fft - real storage mode

SYNOPSIS
     I64R64 and I32R32

       INTEGER   l, iopt, ier
       REAL      x(l), y(l), work(2*l)
       CALL S1DFFT (x, y, l, work, iopt, ier)

       I32R64

       INTEGER*4 l, iopt, ier
       REAL*8    x(l), y(l), work(2*l)
       CALL D1DFFT (x, y, l, work, iopt, ier)

DESCRIPTION
```


-lparfft_64 -Popenmp

Table 4: Data types for MathKeisan libraries

	Integer and floating point data type		
name	I32R32	I32R64	I64R64
METIS	libmetis_32.a	libmetis.a	libmetis_64.a
ParMETIS	libparmetis_32.a	libparmetis.a	libparmetis_64.a
SOLVER	not available	libsolver.a	libsolver_64.a

Files in column I32R32 + I32R64 of Table 3 are for 32 bit integer data type (Fortran integer*4). The floating point data type is determined by the first letter of the subroutine or function name as follows

- s - 32 bit real (Fortran real*4)
- d - 64 bit real (Fortran real*8)
- c - 32 bit complex (Fortran complex*8)
- z - 64 bit complex (Fortran complex*16)

QUESTIONS ?

Very Important

Check your results –
everytime after optimization

Popular traps

IEEE Format (float0) !!!

- 4 Byte:
  - about 7.2 digits
  - 10^{-38} - 10^{38}
- 8 Byte:
  - about 16 digits
  - 10^{-308} - 10^{308}
- 16 Byte:
  - about 32 digits
  - 10^{-308} - 10^{308}

Default data format

option -dw (4-byte integers, 4-Byte reals), pointers are always 64-bit

some popular traps

- Unexpected zero divides
- insufficient record size for formatted output
- non vectorizing loops

unexpected zero divides

- FORTRAN:

```
program t
real a(100)
data a / 100 * 1.0 /
b = 0.0
do I = 1, 100
  if( a(I) .eq. 0.0 ) a(I) = a(I) / b
end do
print *, a(1)
end
```

- a(i) is constant, compiler will move expression outside of the loop
- Solution:
  - -Wf "-O nomove"

Incorrect assumed vector length

- FORTRAN:

-Wf, -pvctl vwork=stack

```
program t
real a(1000)
call sub(a,1000)
...
end
subroutine sub(a,m)
dimension a(63)
do I = 1, m
  a(I) = float(I)
end do
return
end
```

```
% ./a.out
 ****  96 Loop count is greater than that assumed by the compiler
: loop-count=1000 eln=13 PROG=sub ELN=14(400001584)
                 Called from t ELN=5(4000008f4)
%
```

insufficient record size for formatted output

- error message 186:
  - Formatted record insufficient in WRITE
    ...
- reason: buffer for formatted IO too small
- F_SYSLEN=1024 or more
  - does not hurt

non vectorizing loops

- get detailed information from compiler:
  - -R2 for f90
  - -Wf"-pvctl fullmsg"
  - -Wf"-L fmtlist transform map summary"

Heap versus Stack

Stack
private data in a subroutine (kh)

```
subroutine my_test(a,b,c,d)
integer, dimension(100) :: kh

  … do some stuff …
 do i=1,n
   kh(i)=sqrt(a(i)
 enddo
end subroutine
```

Heap versus Stack

Heap:
Global data in a code

```
subroutine my_test(a,b,c,d)
common / my_data / kh
integer, dimension(100) :: lh

  do some stuff
 do i=1,n
   lh(i)=sqrt(a(i)*kh
 enddo
end subroutine
```

But in F90 do not use common, use modules!

Modules: This will be heap

```
MODULE mo_physc2

  USE mo_parameters

  IMPLICIT NONE

  ! ----------------------------------------------------------------
  !
  ! module *mo_physc2* constants to communicate between the main program
  !                and the physical subroutines (except radiation ones).
  !
  ! ----------------------------------------------------------------

  REAL :: clam            !  *asymptotic mixing length for momentum.
  REAL :: ckap            !  *karman constant.
  REAL :: cb              !  *stability parameter near neutrality.
  REAL :: cc              !  *stability parameter for unstable cases.
  REAL :: cd              !  *stability parameter for stable cases.
  REAL :: cchar           !  *charnock constant.
  REAL :: crgcg           !  *heat capacity x density of the soil.
  REAL :: cdif            !  *thermal diffusivity of soil.
  REAL :: cwcap           !  *soil water content at field capacity wsmax)
```

Summary

- Always use setenv F_PROGINF DETAIL and F_FTRACE YES
- Compile and link your code with –ftrace –Wf,-L fmtlist map summary transform
- Look at the transformation list of bottleneck routines
- After optimization compile without –ftrace!

End of THEORY III