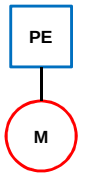
Compiler vectorization

- Fortran90/SX vectorizes automatically
  - if it can identify vectorizable constructs (data parallelism)
- If it cannot vectorize certain code parts :
  - Programmer intervention necessary

```
DO k = kmin, kmax
   DO j = jmin, jmax
      DO i = imin, imax
         wx(i,j,k) = wy(i,j,k) + s1 * wz(i,j,k)
      END DO
   END DO
END DO
```


Performance considerations

- Data access
- Storage order
  - Fortran : column-major




```
…
real, dimension(3, 3) :: x
…
```

x(1,1)	x(2,1)	x(3,1)	x(1,2)	x(2,2)	x(3,2)	x(1,3)	x(2,3)	x(3,3)

```
do j = 1, 3
  do i = 1, 3
    x(i, j) = x(i, j) + …
  enddo
enddo
```

```
do i = 1, 3
  do j = 1, 3
    x(i, j) = x(i, j) + …
  enddo
enddo
```


Compiler listings

- Very helpful for identifying unvectorized code parts
- Understand / control compiler optimizations
- `sxf90 -c -Wf,-L fmtlist summary transform myprog.f90`
  - → file **`myprog.L`** containing
    1. Compiler messages
    2. Transformation listing
    3. Summary of all compiler options
    4. Formatted listing
- `vi myprog.L`
- Go to file end, check for unvectorized parts in formatted listing

Compiler listings

- First part : compiler diagnostics

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

  56  vec  (   1): Vectorized loop.
  61  vec  (   1): Vectorized loop.
  ...
  76  vec  (   3): Unvectorized loop.
  ...
  91  vec  (   4): Vectorized array expression.
```

Compiler listings

- Second part : transformation listing → hands-on sessions
- Third part : Options summary

```
SUMMARY LIST

 PROGRAM : test

 f90 OPTIONS : -C hopt    -Nc        -e a       -e b       -d C       -d D
               -d P       -d R       -e W       -d w       -ftrace    -Nf2003
               -f4        -Ng        -G global
               -I /SX/opt/sxf90/rev482/include
               -P static  -Np        -Npi       -R2        -NS        -sxace
               -NV        -Nw

 f90 DETAILED OPTIONS  :  -NA        -Nadv      -ai        -common global
                          -const_ext -Ncont     -Ndblprecision
                          -dir vec nopar nodebug           -Ndollar   -esc
                          -fusion    -G         -i noerrchk
                          -K         na         nb
                          -L         eject      fmtlist    noinclist  nomap
                          ...
```


Compiler listings

- Fourth part : formatted listing

Vectorisation

```
376: +------>     DO j = 2,je-1
377: |V----->       DO i = 1,ie
378: ||               trf(i,j,ke)=t1o(i,j,ke)/tridsy(i,j,ke,2)
379: |V-----        END DO
380: +------      END DO
```

Loop exchange and vectorisation

```
25: X------> DO i=1,10
26: |+------>       DO j=1,20
27: ||                    a(i,j) = b(i,j) + c(i,j)
28: |+-----         END DO
29: X------ END DO
```

Partial vectorisation

```
32: |V-----> DO j=i,n/2
33: ||              a(j) = b(j) + c(j)
34: ||       S      print *,a(j)
35: |V-----   END DO
```

cf. FORTRAN90/SX Programmer's Guide, chapter 12.6.4

Empowered by Innovation NEC

Performance estimations

"Theoretical peak performance" and what you really get…

Balance analysis

- How fast can a code run, and what limits its performance ?
- Machine balance :
  - $B_m = \frac{Memory\ bandwidth\ [\frac{GB}{s}]}{Peak\ performance\ [\frac{GFlop}{s}]}$ $(\frac{B}{Flop})$
  - SX-ACE : 256 GB/s / 256 Gflop/s = 1 B/F
- Code balance :
  - $B_c = \frac{Memory\ accesses}{Floating-point\ operations}$ $(\frac{B}{Flop})$
- Maximum performance :
  - $P = P_t \times \frac{B_m}{B_c}$

Balance analysis

- Example :

```
DO i=1,n
  v(i) = u(i) + w(i)
END DO
```

- Memory accesses : 2 loads, 1 store = 3
- Floating-point operations : 1
- Code balance 3/1

- P = 64 GFlop/s * 1 / 3 = 21,3 GFlop/s
- Test : P = 10,6 GFlop/s ???
- → only one of two add pipes is used !

Balance analysis

- Example 2 :

```
DO i=1,n
  v(i) = u(i) + s * w(i)
END DO
```

- Memory accesses : 2 loads, 1 store = 3
- Floating-point operations : 2
- Code balance 3/2

- P = 32 GFlop/s * 1 / (3/2) = 21,3 GFlop/s

Empowered by Innovation NEC

Performance analysis

Performance Analysis Tools

Tool	Description	Time Overhead
F_PROGINF C_PROGINF	Environment variables giving SX CPU built-in hardware counters	NO
prof	Unix prof extended to SUPER-UX	YES
loopprof	Profiler on loop level	YES
ftrace	Analyser on routine level	YES
ftrace_region	Analyser on region level	YES

F_PROGINF and C_PROGINF

1) Set environment variables in your jobscript

csh: `setenv F_PROGINF DETAIL`

ksh: `F_PROGINF=DETAIL; export F_PROGINF`

For MPI programs :

csh: `setenv MPIPROGINF ALL_DETAIL`

ksh: `export MPIPROGINF=ALL_DETAIL`

2) Execute the jobscript

F_PROGINF=DETAIL, 1 CPU output

```
                                  ****** Program Information ******
                                  Real Time (sec)      :        76.446357
                UT  =             User Time (sec)      :        19.798240
                                  Sys Time (sec)       :         7.039318
                                  Vector Time (sec)    :        13.360105
                IC  =             Inst. Count          :      2275527965.
SIC = IC -VIC   VIC =             V. Inst. Count       :       310879060.
SVC = SIC+VEC   VEC =             V. Element Count     :     79292000954.
                FC  =             FLOP Count           :     23904445813.
         SVC / UT   =             MOPS                 :      4104.236100
         FC  / UT   =             MFLOPS               :      1207.402590
         VEC / VIC  =             VLEN                 :       255.057388
         VEC / SVC  =             V. Op. Ratio (%)     :        97.582168
                                  Memory Size (MB)     :       128.031250
                                  MIPS                 :       114.935873
                                  I-Cache (sec)        :         0.397157
                                  O-Cache (sec)        :         0.707362
                                  Bank (sec)           :         3.860758
See SUPER-UX Performance Tuning Guide, Chapter 10
                                  Start Time (date)    : 2005/03/16 14:10:32
                                  End Time (date)      : 2005/03/16 14:11:49
```


Ftrace : analyzer on routine level

1) Compile and link program with `-ftrace`

2) Set environment variable in your jobscript

csh: `setenv F_FTRACE YES`

ksh: `export F_FTRACE=YES`

3) Run the job – ftrace output appears on `stderr`

2) Run the job – ftrace output files `ftrace.out*` will be generated

3) Execute the command

```
sxftrace -f ftrace.out
         (> ftrace.txt)
```


Ftrace output

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

Execution : Mon Jan 29 19:54:28 2007
Total CPU : 0:04'40"317


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

      120   42.163( 15.0)    351.356  7443.4  3304.9 98.70 254.1    42.130  0.0233  0.0151  0.0008  occlit
      120   33.109( 11.8)    275.909  8172.8  3835.4 98.63 254.5    33.090  0.0152  0.0107  0.0002  ocmodmom
      240   23.366(  8.3)     97.359  9712.6  3400.8 99.07 254.2    23.252  0.0168  0.0098  0.0040  mo_adpo.ocadpo
        1   23.112(  8.2) 23111.597  3581.6  1631.4 96.52 143.1    20.783  0.0718  0.0050  0.1811  mo_tro.trotest
      120   22.348(  8.0)    186.231  7539.7  3109.1 99.04 216.9    21.561  0.0229  0.0128  0.0009  mo_ocice.ice_dynami
      …
-------------------------------------------------------------------------------------------------------------
   404653  280.317(100.0)      0.693  7265.0  3142.4 98.69 236.4   267.694  0.5444  2.8036  9.8463  total
```

- This is the starting point of program optimisation
- ftrace produces overhead, remove after optimisation !

Ftrace_region : analyzer on loop level

1) Instrument code as follows

```
...
CALL FTRACE_REGION_BEGIN (“REGION A“)

       DO …
       …
       END DO

CALL FTRACE_REGION_END (“REGION A“)
...
```

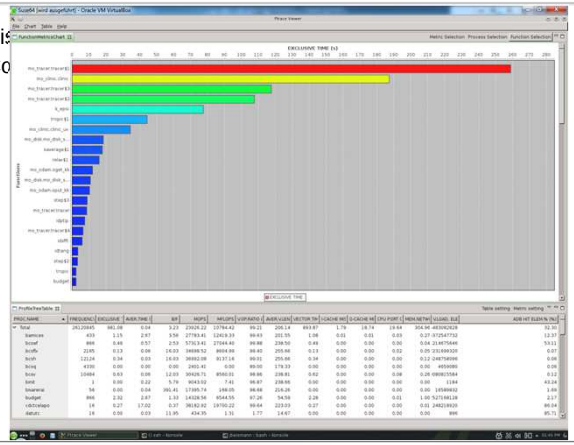
2) Go on as for ftrace

Ftrace_region output

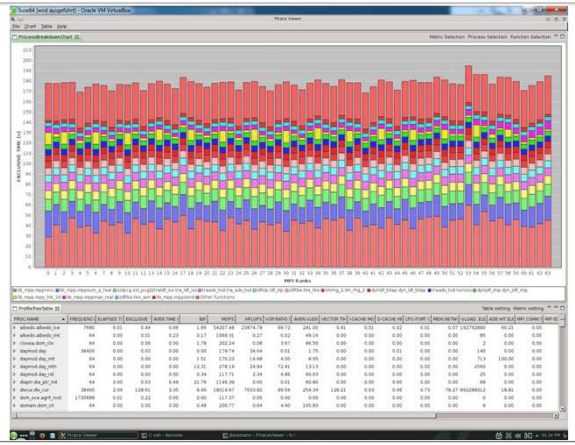
```
*------------------------*
  FLOW TRACE ANALYSIS LIST
*------------------------*
Execution : Mon Jan 29 23:38:35 2001
Total CPU : 0:00'00"004
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME    MOPS MFLOPS V.OP  AVER.   VECTOR I-CACHE O-CACHE    BANK
                      TIME[sec](  % )     [msec]                 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
-------------------------------------------------------------------------------------------------------------
```


Ftrace viewer

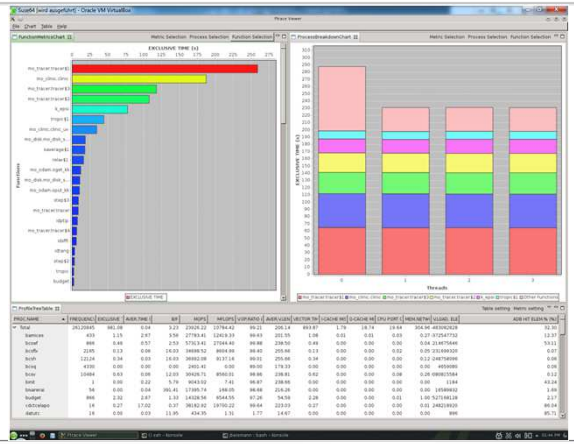
Vis
Co



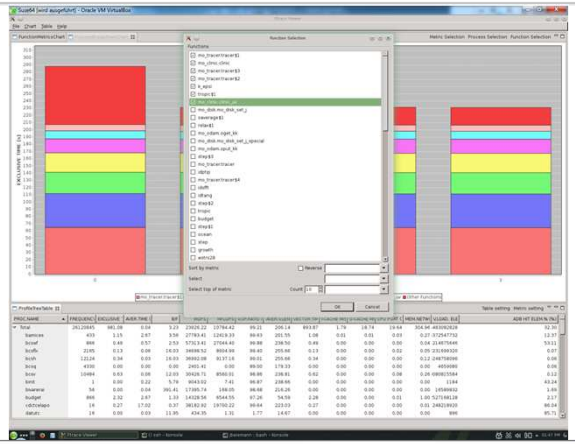
Ftrace viewer



Ftrace viewer



Ftrace viewer



Empowered by Innovation NEC

Improving vectorisation

Basic rules to achieve vector performance

- **Raising Vectorisation Ratio**

  = number of vector instructions / number of execution instructions

- **Improving Vector Instruction Efficiency**

  = effective vector power / maximum (peak) vector power

Empowered by Innovation NEC

Raising vectorisation ratio

…by removing potential dependencies

Example

```
DO J = 1,N
   X(J-1) = X(JW) * Y(J)
   JW     = JW + 1
END DO
```

The compiler does not know if J-1 ≤ JW
If you know it – tell the compiler !

```
!CDIR NODEP(X)
DO J = 1,N
   X(J-1) = X(JW) * Y(J)
   JW     = JW + 1
END DO
```


…by removing potential dependencies

Indirect Adressing

```
DO i=1,n
   j=k(i)
   a(j)=a(j)+b(i)
END DO
```

The compiler does not know if j = k(i) is injective
If you do …

```
!CDIR NODEP
DO i=1,n
   j=k(i)
   a(j)=a(j)+b(i)
END DO
```

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

i	j
1	3
2	2
3	1
4	3
5	4

```
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)
```


…by inline expansion of procedures

```
DO I=1,N
 CALL MAT(A(I),B(I),C(I),D(I),X(I),Y(I))
ENDDO

SUBROUTINE MAT(S,T,U,V,A,B)
 A = S*U+T*V
 B = S*V-U*T
 RETURN
END
```

```
DO I=1,N
 X(I)=A(I)*C(I)+B(I)*D(I)
 Y(I)=A(I)*D(I)-B(I)*C(I)
ENDDO
```

Automatic inlining : `sxf90 … -pi auto …`
Explicit inlining : `sxf90 … -pi exp=<routine name> …`
`sxf90 … -pi expin=<file/directory name>`

…by vectorizing IF blocks

Example 1) f90:

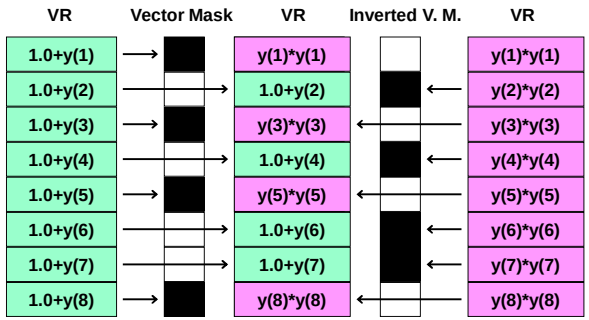
```
WHERE( y > 0.5 )
  x = 1.0 + y
ELSEWHERE
  x = y * y
END WHERE
```

f77:

```
DO i=1,n
  IF ( y(i) .GT. 0.5 ) THEN
    x(i) = 1.0 + y(i)
  ELSE
    x(i) = y(i) * y(i)
  END IF
END DO
```

Can be vectorized using vector mask-registers
Complete loop is computed twice


```
WHERE( y > 0.5 )
  x = 1.0 + y
ELSEWHERE
  x = y * y
END WHERE
```



…by vectorizing IF blocks

Example 2) f90:

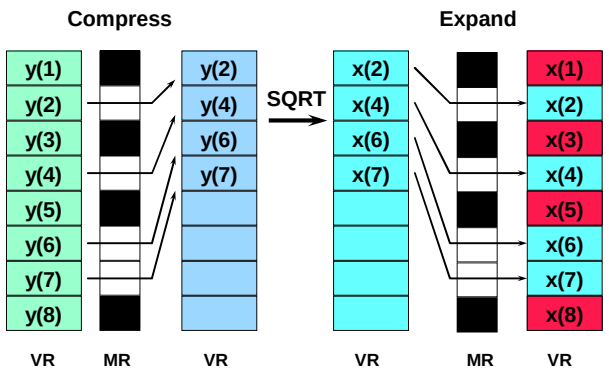
```
WHERE ( y >= 0.0 ) x = SQRT(y)
```

f77:

```
DO i=1,n
  IF( y(i) .GE. 0.0 ) THEN
    x(i) = SQRT( y(i) )
  END IF
END DO
```

To force vectorization method use compiler
- option `sxf90 … -Wf"-pvctl compress"`
- or directive `!CDIR COMPRESS`


```
WHERE( y >= 0.0 ) x = SQRT(y)
```



…by vectorizing IF blocks using list vectors

Example 3)

```
DO I=1,N
 IF (A(I).NE. 0.0) THEN
   C(I) = A(I)*SIN(B(I))
   …
 END IF
END DO
```

```
K=0
DO I=1,N
     IF(A(I).NE.0.0)THEN
       K=K+1
       IX(K)=I
     END IF
END DO

!CDIR NODEP
DO I=1,K
  C(IX(I))=A(IX(I))*SIN(B(IX(I)))
  …
END DO
```


"constant IF"

FORTRAN:

```
do I = 1, n
  if( ifirst .eq. 1 ) then
    a(I) = b(I)
  else
    a(I) = 0.5 * ( b(I) + bold(I) )
  end if
end do
```

solution :
move the if outside of the loop

```
if( ifirst .eq. 1 ) then
  do I = 1, n
    a(I) = b(I)
  end do
else
  do I = 1, n
    a(I) = 0.5 * ( b(I) + bold(I) )
  end do
end if
```


"boundary IF"

FORTRAN:

```
do I = 1, n
  if( i .eq. 1 ) then
    a(I) = b(1)
  else
    a(I) = 0.5 * ( b(I) + b(I-1) )
  end if
end do
```

solution :

```
a(1) = b(1)
do I = 2, n
    a(I) = 0.5 * ( b(I) + b(I-1) )
end do
```

`sxf90 -Chopt`

Basic rules to achieve vector performance

- Raising Vectorisation Ratio

  remove the cause of non-vectorization !

Empowered by Innovation NEC

Important compiler options

An overview – please refer to the User Guides for details

f90 options : **precision promotion**

Command line option :

-ew ***Define basic storage unit as 8 bytes (default: 4 bytes)***

Detailed option :

-Wf, -A <suboption>
Controls precision promotion, taking into account explicit kind specifications

e.g.:
-Wf, -A idbl4 ***promote 4-byte REAL and COMPLEX to 8-byte, except for variables and constants with explicit kind specification***

Command line options : **inlining**

-pi ***Specifies that inline expansion is to be performed for external procedures.***

auto ***Specifies automatic expansion. (default)***

incdir ***Specifies that Fortran source files in the directories specified by -I option are used for inlining.***

nest=n ***Specifies the depth of nests to be expanded automatically (default value:1)***

exp=routine-name[,routine-name]
Specifies that the specified routine name is expanded inline regardless of the specification of automatic expansion.

fullmsg ***All inlining diagnostic messages are output.***

…

Optimisation options F90

Sometimes –C hopt leads to problems. Then go back to –C vopt or
• try to identify the routine and
• switch off specific optimizations for this routine:

With –C hopt: modify default settings

-Wf, … :	
-i errchk	**inline intrinsic functions (with argument check)**
-O nodiv	**Division may not be changed (rec. multiplication)**
-O nomove	**Disable automatic movement of invariants outside of loop**
-O nounroll	**Disable automatic unrolling**
…	
…	

Investigation of Correctness

Use a combination of following f90 options

–Cdebug	Enable debug (suppresses vector and scalar optimizations).
–eC	Enable runtime array bounds checking.
–Cvsafe	Enable only safe vector optimizations.
–Pstack	Disable parallel processing. Local data allocated on the stack.
–Pstack Wf" – init stack=nan heap=nan"	Initialize the stack and heap with NaN.
– Wl"–f nan"	Preset uninitialized common block to nan.

Example

–eC –Pstack Wf" – init stack=nan heap=nan"

- Program will inform and stop when using
  - out of bounds array elements
  - non initialized values
- time overhead, do not use this for production !