Performance of Fortran IO mechanisms

Uwe Küster
Denis Altmann
kuester@hlrs.de

University of Stuttgart
High-Performance Computing Center Stuttgart (HLRS)
www.hlrs.de

general remarks

- IO will be a key feature for large scale instationary problems
- parallel IO will be needed (not addressed here)
- fast and flexibel IO needed
- ASCII and binary IO
- impact of processor performance on IO performance

general remarks

- tested machines are now out-dated; be aware of the relations
  - Itanium 2 1000 MHz with efc
  - NEC SX-5 with f90
  - Pentium III 1000 MHz with ifc
- IO bandwidth understood as useful effective bandwidth
  not as filesize / time
  (formatted output needs more bytes for reaching the same accuracy as unformatted output)
- performance numbers in MB/sec
- tried to use fast IO devices
- two dimensional array with short and long range
- tables on the following slides:
  - write performance
  - read performance

not tested

- direct access IO
- complicated structures mixing different data types
- IO with derived types
- direct access IO
- special vendor techniques
- comparison to C would be interesting because C interacts directly with the operating system;
  not done here

influence of IO hardware

- the IO hardware is not tested here; IO may be memory buffered but this is the users point of view
- non dedicated hardware; competing with other processes
- IO may run on independent IO processors
  e.g. Infiniband sends buffers independently on the processor after a set up phase

IO - example as tested here

```
subroutine write_array_6(string,array,byte,unit)
character(len=*) :: string
real(kind=rk),dimension(:,:)  :: array                    assumed shape array
integer :: ii,imax
integer :: mm,mmax
integer :: unit
integer :: byte                                           string is set for identification
string='write(unit) ((array(mm,ii),mm=1,mmax),ii=1,imax)'
byte=8
mmax=size(array,1);   imax=size(array,2)
  write(unit) ((array(mm,ii),mm=1,mmax),ii=1,imax)
end subroutine write_array_6                              IO instruction
```


Explanation of the measurements

- different kind of small IO procedures
- source code on the slides
- mmax=3 is small
- imax is large enough to guarantee a continous data transfer for more than one second
- tables for write and read look like the following

	Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
write	5,1	0,7	4,3
read	3,1	0,3	2,8

formatted IO

- writes and reads interchangeable ASCII files
- needs **formats**
- formats may appear as character strings and are analyzed at run time --> time consuming
- formats may be very complicated
- list directed **write(unit, *), print *** for simple output; is machine dependent
- list directed **read (unit, *)** reads variable lists separated by blanks; simply to use

formatted IO: list directed

```
do ii=1,imax;
   write(unit,*) (array(mm,ii),mm=1,mmax);
enddo

do ii=1,imax;
   read(unit,*) (array(mm,ii),mm=1,mmax) ;
enddo
```

most flexible way for read

allows list of numbers separated by any number of blanks

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
5,1	0,7	4,3
3,1	0,3	2,8

essentially slower than constant format

formatted IO; character constant format

```
do ii=1,imax;
   write(unit,'(10e15.5)') (array(mm,ii),mm=1,mmax);
enddo

do ii=1,imax;
   read(unit,'(10e15.5)') (array(mm,ii),mm=1,mmax) ;
enddo
```

simple

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
6,1	1,0	5,6
6,6	0,8	6,6

formatted IO: labelled format

```
do ii=1,imax;
   write(unit,100) (array(mm,ii),mm=1,mmax);
enddo;
100  format(10e15.5)

do ii=1,imax;
   read(unit,100) (array(mm,ii),mm=1,mmax) ;
enddo;
```

former way of formatted IO

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
6,1	1,0	5,7
6,8	0,8	6,6

same as constant format

formatted IO: character variable format

```
format='(10e15.5)';
do ii=1,imax;
   write(unit,format) (array(mm,ii),mm=1,mmax);
enddo

do ii=1,imax;
   read(unit,format) (array(mm,ii),mm=1,mmax) ;
enddo
```

dynamic definition of format at run time

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
5,6	0,6	5,2
6,4	0,6	6,2

slower than constant format

unformatted IO

- no complicated data conversion is needed as for formatted IO
- no information loss
- amount of data to be stored is smaller
- output is not readable
- record control words
  - separating records; appear at start and end of the file
  - incompatable with C - IO; but may be handled by C
  - may be suppressed with some compilers
  - in this case the fast access to separated records will be lost
- not standardized
  - difficulties of interchanging data between different systems
  - but defacto possible to write and read identical files on any system
  - problems on interchange between little and big endian systems solvable if all data types have the same size

unformatted IO of short implicit loop, long outer loop

```
do ii=1,imax;
  write(unit) (array(mm,ii),mm=1,mmax);
enddo

do ii=1,imax;
  read(unit) (array(mm,ii),mm=1,mmax);
enddo
```

implicit do lists
short records separated
by record control words

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
31,6	2,1	18,1
31,1	1,8	22,0

unformatted IO of nested implicit loop: (small,large)

```
write(unit) ((array(mm,ii),mm=1,mmax),ii=1,imax)

read(unit) ((array(mm,ii),mm=1,mmax),ii=1,imax)
```

single record
nested in
natural order

special treatment on NEC SX-5

faster than
short records

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
56,3	453524,1	32,3
51,4	470336,9	35,4

unformatted IO of nested implicit loop: (small,large) reversed order

```
write(unit) ((array(mm,ii),ii=1,imax),mm=1,mmax)

read(unit) ((array(mm,ii),ii=1,imax),mm=1,mmax)
```

nested in
nonnatural
order

NEC SX-5 has incredible
low performance

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
59,8	2,3	33,2
52,8	2,0	28,3

for smaller sizes no
problem for Itanium
and PC

unformatted IO of nested implicit loop: (large,small)

```
write(unit)
((array(ii,mm),ii=1,imax),mm=1,mmax)

read(unit)
((array(ii,mm),ii=1,imax),mm=1,mmax)
```

nesting in natural order;
long index first

special treatment on NEC SX-5

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
59,1	461500,9	32,9
52,9	498421,0	36,1

unformatted IO of complete array section

```
write(unit) array(:,:)

read(unit) array(:,:)
```

array section of the whole array;
compiler treats in a different way

no difference to implicit do loop
for Intel

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
58,0	419,1	32,3
51,8	214,8	35,3

unformatted IO of complete array

```
write(unit) array

read(unit) array
```

complete array as buffer

Itanium_2 1 GHz	NEC SX-5	P III 1 GHz
70,9	438,9	103,0
717,1	218,3	125,6

best for all machines
IO may go not to disk

write ordered by performance (Itanium efc)

```
do ii=1,imax; write(unit,*) (array(mm,ii),mm=1,mmax); enddo
```

```
format='(10e15.5)';
do ii=1,imax; write(unit,format) (array(mm,ii),mm=1,mmax); enddo
do ii=1,imax; write(unit,'(10e15.5)') (array(mm,ii),mm=1,mmax); enddo
do ii=1,imax; write(unit,100) (array(mm,ii),mm=1,mmax); enddo; 100 format(10e15.5)
```

```
do ii=1,imax; write(unit) (array(mm,ii),mm=1,mmax); enddo
 write(unit) ((array(mm,ii),mm=1,mmax),ii=1,imax)
 write(unit) array(:,:)
 write(unit) ((array(ii,mm),ii=1,imax),mm=1,mmax)
 write(unit) ((array(mm,ii),ii=1,imax),mm=1,mmax)
 write(unit) array
```

factor
13

read ordered by performance (Itanium efc)

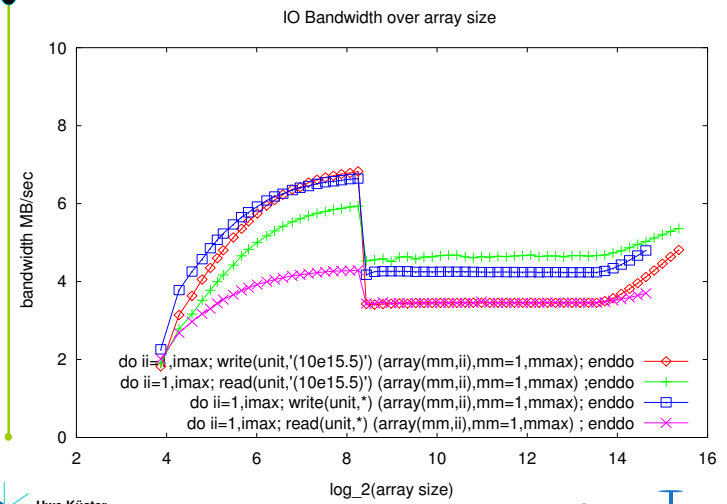
```
do ii=1,imax; read(unit,*) (array(mm,ii),mm=1,mmax) ; enddo
```

```
do ii=1,imax; read(unit,format) (array(mm,ii),mm=1,mmax); enddo
do ii=1,imax; read(unit,'(10e15.5)') (array(mm,ii),mm=1,mmax) ;enddo
do ii=1,imax; read(unit,100) (array(mm,ii),mm=1,mmax) ; enddo;100 format(10e15.5)
```

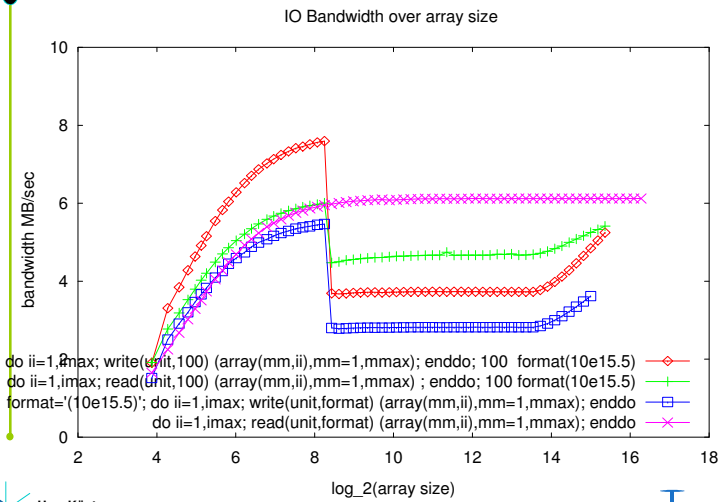
```
do ii=1,imax; read(unit) (array(mm,ii),mm=1,mmax); enddo
 read(unit) ((array(mm,ii),mm=1,mmax),ii=1,imax)
 read(unit) array(:,:)
 read(unit) ((array(mm,ii),ii=1,imax),mm=1,mmax)
 read(unit) ((array(ii,mm),ii=1,imax),mm=1,mmax)
 read(unit) array
```

factor 200 ?

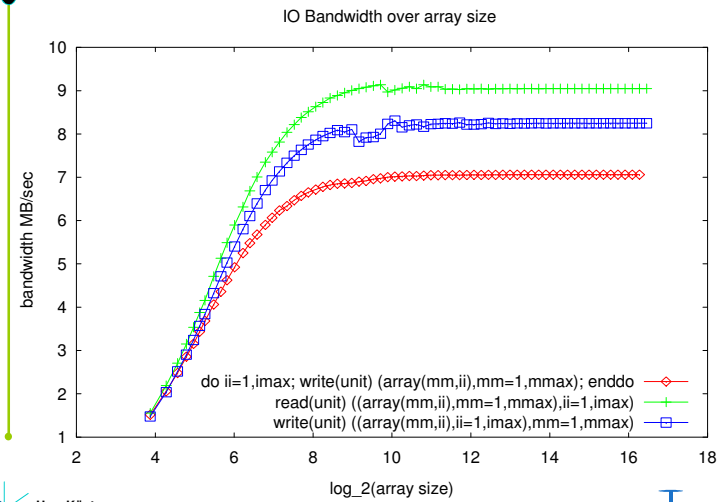
NEC SX-8: Formatted IO (1)



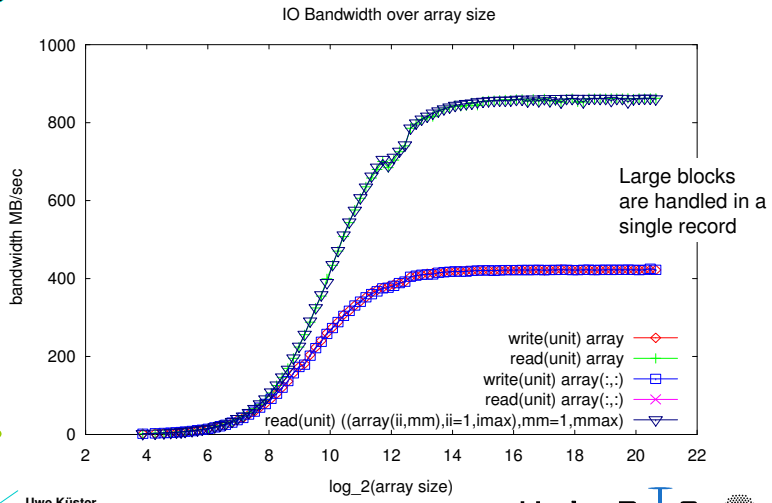
NEC SX-8: Formatted IO (2)



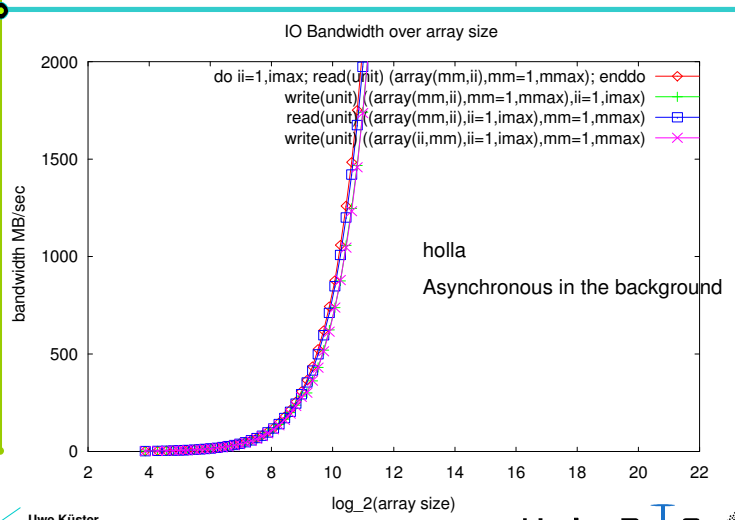
NEC SX-8: Unformatted IO (1)



NEC SX-8: Unformatted IO (2)



NEC SX-8: Unformatted IO (3)



conclusion

- best is unformatted IO for long buffers
- files of this kind may be exchanged between different systems
- performant and interchangeable IO by differentiation between meta data describing the data and pure intrinsic data
  - different files for meta data and binary data (in the same directory)
  - additional files for integers and other intrinsic data
  - avoid writing of derived types

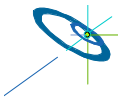
IO example: write

```
meta_file=trim(file)//'.meta'
bin_file=trim(file)//'.bin'
open(unit=meta_unit,file=trim(meta_file))
write(meta_unit,'(a)') trim(bin_file)
write(meta_unit, * ) size(array,1)
write(meta_unit, * ) size(array,2)
close(meta_unit)
open(unit=buffer_unit,file=trim(bin_file))
write(buffer_unit) array
close(buffer_unit)
```

IO example: read

```
open(unit=meta_unit,file=trim(file)//'.meta')
read(meta_unit,'(a)')  bin_file
read(meta_unit, * ) mmax
read(meta_unit, * ) jmax
close(meta_unit)

allocate(array(imax,mmax))
open(unit=buffer_unit,file=trim(bin_file))
read(buffer_unit) array
close(buffer_unit)
```



Thank you