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 flexible 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
- 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

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

<table>
<thead>
<tr>
<th>write</th>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5.1</td>
<td>0.7</td>
<td>4.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>read</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3.1</td>
<td>0.3</td>
<td>2.8</td>
</tr>
</tbody>
</table>

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

```fortran
  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
```

<table>
<thead>
<tr>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.1</td>
<td>0.7</td>
<td>4.3</td>
</tr>
<tr>
<td>3.1</td>
<td>0.3</td>
<td>2.8</td>
</tr>
</tbody>
</table>

Most flexible way for read allows list of numbers separated by any number of blanks.

### formatted IO: character constant format

```fortran
  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
```

<table>
<thead>
<tr>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>1.0</td>
<td>5.6</td>
</tr>
<tr>
<td>6.6</td>
<td>0.8</td>
<td>6.6</td>
</tr>
</tbody>
</table>

### formatted IO: labelled format

```fortran
  do ii=1,imax;
    write(unit,100) (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)

### formatted IO: character variable format

```fortran
  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
```

<table>
<thead>
<tr>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>0.6</td>
<td>5.2</td>
</tr>
<tr>
<td>6.4</td>
<td>0.6</td>
<td>6.2</td>
</tr>
</tbody>
</table>

Dynamic definition of format at run time.

Same as 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
  - incompatible 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

```fortran
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
```

<table>
<thead>
<tr>
<th></th>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>31,6</td>
<td>2,1</td>
<td>18,1</td>
</tr>
<tr>
<td></td>
<td>31,1</td>
<td>1,8</td>
<td>22,0</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>56,3</td>
<td>453524,1</td>
<td>32,3</td>
</tr>
<tr>
<td></td>
<td>51,4</td>
<td>470336,9</td>
<td>35,4</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th></th>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>59,8</td>
<td>2,3</td>
<td>33,2</td>
</tr>
<tr>
<td></td>
<td>52,8</td>
<td>2,0</td>
<td>28,3</td>
</tr>
</tbody>
</table>
unformatted IO of nested implicit loop: (large,small)

```fortran
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

<table>
<thead>
<tr>
<th></th>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>59.1</td>
<td>461500.9</td>
<td>32.9</td>
</tr>
<tr>
<td>L2</td>
<td>52.9</td>
<td>498421.0</td>
<td>36.1</td>
</tr>
</tbody>
</table>

unformatted IO of complete array section

```fortran
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

<table>
<thead>
<tr>
<th></th>
<th>Itanium_2 1 GHz</th>
<th>NEC SX-5</th>
<th>P III 1 GHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>58.0</td>
<td>419.1</td>
<td>32.3</td>
</tr>
<tr>
<td>L2</td>
<td>51.8</td>
<td>214.8</td>
<td>35.3</td>
</tr>
</tbody>
</table>

unformatted IO of complete array

```fortran
write(unit) array
read(unit) array
```

complete array as buffer

```fortran
write(unit) array
read(unit) array
```

complete array as buffer

best for all machines

IO may go not to disk

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

write ordered by performance (Itanium etc)

```fortran
factor
```

```fortran
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
```

factor

```fortran
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
```
read ordered by performance (Itanium etc)

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

factor 200 ?
```

NEC SX-8: Formatted IO (1)

```fortran
io Bandwidth over array size

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

NEC SX-8: Formatted IO (2)

```fortran
io Bandwidth over array size

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

NEC SX-8: Unformatted IO (1)

```fortran
io Bandwidth over array size
```

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

```fortran
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

```fortran
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