1) For WRITE operations:

1. **Write full array to the file instead of each element individually** (speed increases by roughly 30%) :

Example:

(BAD)	do row=1,N write(111,"(1E11.3)") data(row,File_Number)) end do
(BETTER)	write(111,"(1E11.3)") data(1:N,File_Number)

2D Arrays:

(BAD)	do j=1, ny, 1 do i=1, nx, 1 write(111,"(1E11.3)")real(hz(i,j)) end do end do
(BETTER)	write(111,"(1E11.3)") hz(1:nx,1:ny)

2. **Specify RECL**

The sum of the record length (RECL specifier in an OPEN statement) and its overhead is a multiple or divisor of the blocksize, which is device-specific. For example, if the BLOCKSIZE is 8192 then RECL might be 24576 (a multiple of 3) or 1024 (a divisor of 8).

The RECL value should fill blocks as close to capacity as possible (but not over capacity). Such values allow efficient moves, with each operation moving as much data as possible; the least amount of space in the block is wasted. Avoid using values larger than the block capacity, because they create very inefficient moves for the excess data only slightly filling a block (allocating extra memory for the buffer and writing partial blocks are inefficient).

The RECL value unit for formatted files is always 1-byte units. For unformatted files, the RECL unit is 4-byte units, unless you specify the -assume byterecl (Linux) or /assume:byterecl (Windows) option for the ASSUME specifier to request 1-byte units.

2) In general :

1. Avoid Conditionals in Loops (**if** in loops) or use as less as possible

2. For short Loops minimize Loop Overhead **(for n>m)**

(BAD)	`do i = 1, n`
	`  do j = 1, m`
	`  ! (some code)`
	`  end do`
	`end do`

(BETTER)	`do j = 1, m`
	`  do i = 1, n`
	`  ! (some code)`
	`  end do`
	`end do`

3. In arrays of several dimensions when using DO – Loops make first last dimension:

(BAD)	`do i=ICON1,ICON2M`
	`  do j=JCON1,JCON2M`
	`  hz(i,j)=hz(i,j)-dt/(mu_0*dx)*(ey2(i+1,j)-ey2(i,j))&`
	`  & +dt/(mu_0*dy)*(ex2(i,j+1)-ex2(i,j))`
	`  end do`
	`end do`

(BETTER)	`do j=JCON1,JCON2M`
	`  do i=ICON1,ICON2M`
	`  hz(i,j)=hz(i,j)-dt/(mu_0*dx)*(ey2(i+1,j)-ey2(i,j))&`
	`  & +dt/(mu_0*dy)*(ex2(i,j+1)-ex2(i,j))`
	`  end do`
	`end do`