epcc

Overview

- Introduction and Background
- Initial parallelisation with MPI-1
  - MPI-1 benchmarks
- Improvements and parallelisation with MPI-2
  - MPI-2 benchmarks
- Conclusion

- We can calculate the ground state energy of a material using Density Functional Theory.
- For many substances, e.g. semiconductors, we are interested in their excited states.
- To do this, we calculate their self-energy and then use peturbation thoery to calculate the excited energy.
- The serial GW space code was developed to do this.

The GW Space Code

- Although there are many improvements that can be done to the code, it is still very expensive.
- A parallel version of this code has been developed using MPI-1 which has significant gains in terms of computation time.
- But the code is slowed down too much by time-consuming MPI_GATHER and MPI_BCAST calls.

- The culprits are input and output
- several large arrays split across PEs
- each PE deals with a sub-section of the array
- each PE requires a copy of all input data
- all sub-arrays are sent to the master PE which writes out the complete array

MPI-1 Benchmarks

min/PE	min	routine	mostly spent in
64.22	4110	program	-
	3190	MPI_BCAST	-
	276	gwst	MPI_BCASTs
6.67	427	smatrel_in_sigma	MPI_ALLGATHERV(some array), iterim=sum() eq. sxexpv
6.39	409	sigma	MPI_BCAST(some array)
	369	MPI_ALLGATHERV	-
3.23	207	polar	MPI_GATHERV(some array)
	201	MPI_GATHERV	-
1.20	76.9	sfq2	calculation of an inner product
0.63	40	ReOrder	-
0.57	36.2	selfx	sxexpv=sxepv+...
0.29	18.5	greensrt	green=green+gkaux*aux
0.27	17.2	triangle_in_rmt	-

Optimisations

- Input
  - Modification of MPI-1 routines
    - JC's code
    - Reading on all PEs
  - MPI-I/O

- Output
  - Modification of MPI-1 routines
    - JC's code
  - MPI-I/O
    - With basic datatypes
    - With derived datatypes

```
IF(rank.eq.0) THEN
        OPEN file
ELSE
        MPI_RECV(offset)
        REWIND(iunit)
        DO i = offset,offset+enddata
                READ(iunit,REC = i,...)
        ENDDO
ENDIF
MPI_SSEND(offset,rank+1)
```

- Code runs sequentially and needs unformatted input

Improvements to MPI-1

- The main slowdown with input is in broadcasting arrays.
- It is then possible to have all PEs access the file and read in the data at the same time.
- The logical variable pe_all_read is set in the control file
- This must be set to false if the OS locks files when being read.

Timings for pe_all_read

Over all input:

pe_all_read(.false.) = 28.9282

pe_all_read(.true.) = 28.9298

Over the cast1.F90 and cast2.F90 routines, the timings were:

pe_all_read=(.false.) = 28.7398

pe_all_read=(.true.) = 28.8364

Use of MPI-2 for I/O

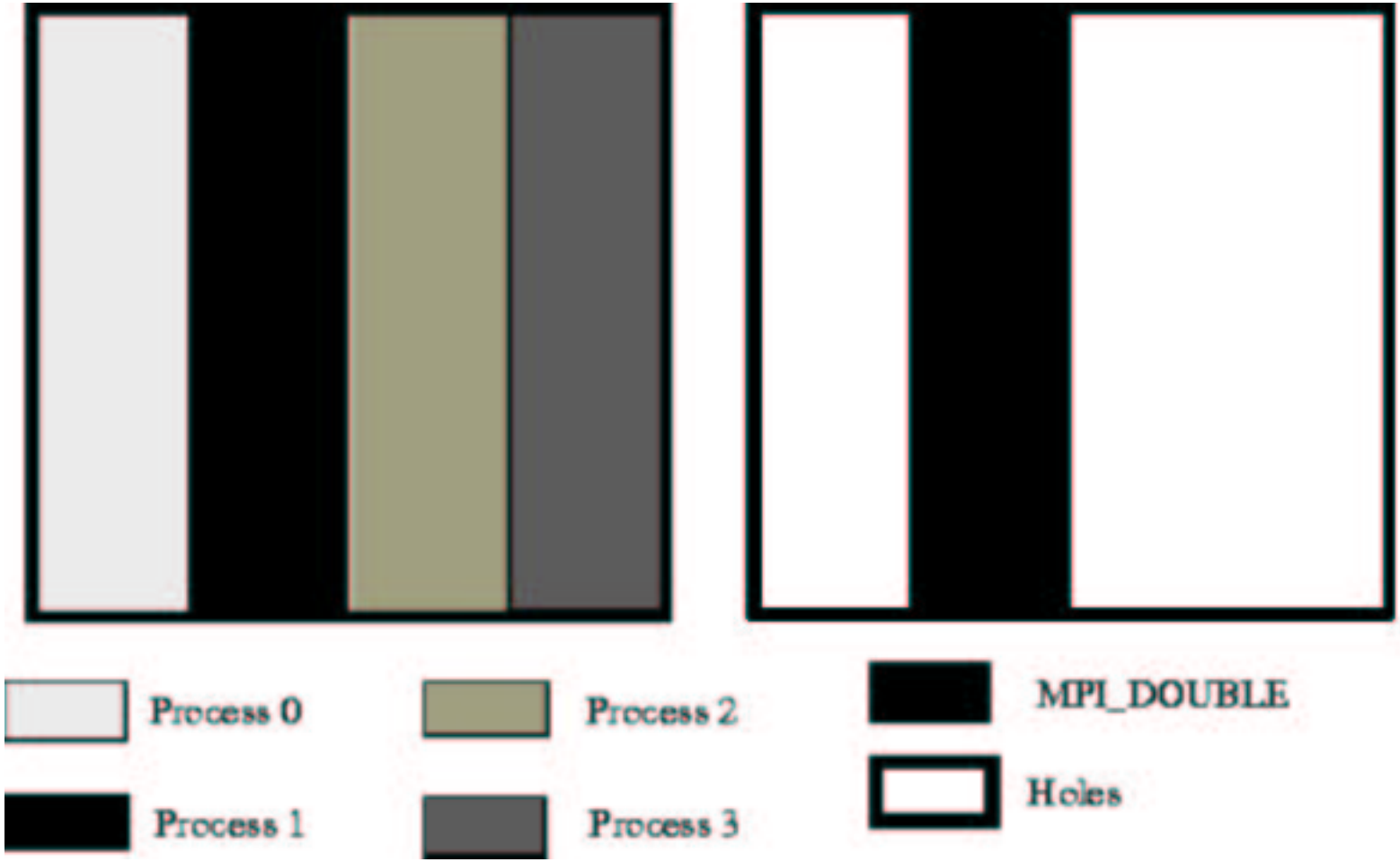
MPI-1

```
CALL MPI_TYPE_STRUCT(...,FINALTYPE,...)
CALL greensrt(...,greensnPE,...)
CALL greensrt(...,greenspPE,...)
CALL MPI_GATHERV(greenspPE,...)
CALL MPI_BARRIER
IF (rank.eq.0) WRITE greenspPE
```

MPI-2

```
CALL MPI_TYPE_CREATE_SUBARRAY(...,filetype,...)
CALL MPI_FILE_SET_VIEW(...,MPI_REAL,filetype,...)
CALL greensrt(...,greensnPE,...)
CALL greensrt(...,greenspPE,...)
CALL MPI_FILE_WRITE_ALL(...,greenspPE,...)
```

Derived Datatypes



Problems with MPI-2

In original code:

Write to record:

rstars%nstars*(it-twgrid%ntmin) + irs

1st read from record:

(kstars%npos*rstars%nstars * it) +
(kstars%npos*(irs-1)) +
kstars%regtopos(kstars%nomem(ik)

2nd read from record 2:

rstars%nstars*(it-twgrid%ntmin) + irs

In MPI-2 code:

Write to record:

irs

1st read from record:

??????

2nd read from record:

irs

MPI-2 benchmarks

Conclusion

- For large arrays, MPI-2 I/O is more efficient
- For small arrays and other data, MPI-1 should be used
- MPI-2 I/O should be easier to implement
- Input of data should be looked at more closely. Currently the cast routines are the most efficient.