EPCC-SS-2001-12: I/O Issues and Benchmarking of the Parallel GW Space code

Scott Fraser

EPCC, University of Edinburgh
University of Edinburgh
email scott@epcc.ed.ac.uk

Introduction

The GW space code is used to calculate the excited states of materials such as semi-conductors. However the code is very computationally intensive, and for realistic problems the execution time is too high. Accordingly a parallel version of the code has been developed using MPI.

While the execution time is lowered somewhat, the code is limited by several I/O routines. Profiling revealed that MPI_BCAST calls prior to writing to file used the most time. This project focuses on these I/O issues with the aim of improving the performance of the code, with both MPI and new MPI-2 methods.

Background Physics

The *GW* approximation proposed by Hedin was that the self-energy operator is given by:

$$\Sigma(r,r';\omega) = \frac{i}{2\pi}\int_{-\infty}^{\infty} d\omega' W(r,r';\omega) \times G(r,r';\omega+\omega')e^{i\omega'\delta} \qquad (1)$$

We can calculate the ground state of a material using density functional theory, and treat the self-energy as a peturbation upon this to obtain a value for the excited state of a substance. This is particularly useful when dealing with semi-conductors as they are insulators in their ground states.

It has been shown that for semiconductors, the *GW* approximation allows computation of the band gaps in excellent agreement with experiment.

The code used here was developed by Reiger *et al.* A parallel version of the code using MPI had previously been developed. This code used a great deal of computational time with expensive I/O procedures, and here these were to be improved by means of new MPI code and also by introducing MPI-2 code.

Improvements to I/O

There were various techniques investigated that could be used to improve I/O performance. These were:

- Use an algorithm with synchronous sends and receives to allow each processor to read the specific part of an unformatted file that it wishes to have, provided that all the data is not required on every processor.
- It is possible to allow all the PEs to access the file and read in the data. This removes the time-consuming broadcasts and two of the subroutines in the input phase.

  Due to the fact that this reduces the portability of the code as some operating systems lock files when they are being read as well as when data is written to them, the code was written with a logical variable pe_all_read set in the control file.

  This determines at runtime whether just one or all PEs should read the code.
- Use of MPI-2: the first method using MPI-2 code to write out temporary files used basic datatypes and is not the most efficient way to write data. The algorithm used the MPI_FILE_WRITE_AT_ALL call to write each part of the array to the correct offset in the file.

  While this algorithm was not fully implemented, an uncomplete version was tested.
- A more thorough investigation of MPI-2 revealed that using derived datatypes has the potential to increase the speed of the I/O processes by an order of magnitude compared to basic datatypes. In this case the implementation is much easier, as there is a new function in MPI-2, MPI_TYPE_CREATE_SUBARRAY which will give each PE a view of a section of an array only. This can then be written directly to file by each PE, rather than having the whole array gathered onto one processor and written to file (see Figure 1).

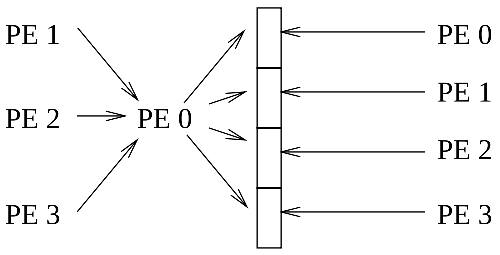

Figure1: Schematic representation of standard MPI (left) and MPI-2 writing to file

Benchmarks

Results are presented for a Cray T3E 1200 (CSAR service at Manchester):

Num/PEs	Time/min	Speedup
Serial	91.83	1
1	99.05	0.93
8	29.29	3.14
16	23.25	3.94
32	21.07	4.36
64	20.72	4.43

The previously developed MPI-1 code was recompiled with the optimisation flags -O3 and -Ounroll2 and benchmarked against the serial code with the same flags.

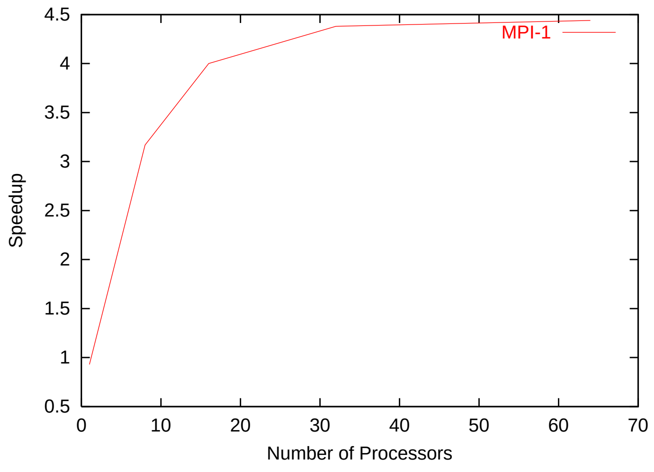

Figure 2: Speedup for MPI-1 Code

The code does not scale well because a lot of the code is inherently sequential, with only the time intensive calculations having been parallelised. Also the time taken for the intensive MPI_BCAST calls increases.

This code would appear to be most efficient when run with between 8 and 16 processors, although some gain can be achieved with th new I/O methods.

For detailed profiling of the code and a discussion of the various improvements to the I/O please see the report at http://www.epcc.ed.ac.uk/ssp/

Acknowledgements

The author would like to thank the EPCC for the opportunity to take part in the SSP. Also thanks to Lorna Smith and Elena Breitmoser who were the supervisors for this project.

References

1. M.Reiger, Steinbeck, I. White, H. Rojas and R.Godby, *Computer Physics Communications*, 117 pp2113-2228, 1999
2. L.Hedin, *Phys. Rev.*, 139 A796, 1965