The Use of PVM with Workstation Clusters in performing Ab initio calculations for Solid State Physics

Deepak Jain

Parsons Power Generation Systems Ltd
(Rolls Royce Power Engineering Group)
&
Department of Physics
The University of Newcastle upon Tyne
Newcastle upon Tyne, UK
Email : D.Jain@newcastle.ac.uk

P.R.Briddon

Department of Physics
The University of Newcastle upon Tyne
Newcastle upon Tyne, UK
Email : Patrick.Briddon@newcastle.ac.uk

Abstract

Over recent years clusters of workstations have began to replace systems of mainframes. This is essentially because of the high performance/cost ratio of these machines for most scientific/industrial applications. For computationally intensive problems these "farms" can be harnessed to behave as a parallel machine, and this can be a way to obtain supercomputing performance at a modest cost.

Here we report the use of such a network within a PVM framework to carry out *ab initio* local density functional calculations. The aim is to be able to predict the properties of defects in semiconductors with a reliable quantitative accuracy. The use and implementation of PVM in this project is discussed with an indication of the speed improvements that have been found and possible directions for further work.

1 Introduction

The applications that we are interested in concern the determination of properties of defects in semiconductors from first principles. This means that given a host material, for example gallium arsenide, and a defect, for example a vacancy or impurity, we should be able to predict a whole range of properties (for example, equilibrium structures, dipole moments, vibrational frequencies, hyperfine couplings etc) only from a knowledge of the atomic numbers of the atoms present. This means that there are no variable parameters in our program which need to be adjusted when moving from one material to another which would need to be taken from experiment. Clearly approximations are necessary, but these are well–defined and are the same for all host materials and for all defects. Our calculations use spin–polarised density functional theory applied in conjunction with *ab inito* norm conserving pseudopotentials. These calculations are very expensive in terms of computer time, but the rapid improvement in hardware has let to them becoming more common in solid state physics and the theoretical study of defects has been revolutionised by them.

In our calculation, the solid is simulated either using a very large unit cell containing around 100 atoms, or as a cluster of up to 300 atoms. The wavefunctions are expanded in terms of Gaussian orbitals, centred on atomic positions and the centres of bonds and other symmetry points in the lattice. The program involves two main operations : (i) the determination of matrix elements of the hamiltonian between Gaussian orbitals and (ii) the diagonalisation of the matrix so generated. We consider that a major advantage of our computational formalism is the fact that only three–centre integrals are required, rather than four-centre integrals which are encountered in quantum chemistry codes based on Hartree Fock theory. In practice this means that for a problem involving N basis functions, both stages (i) and (ii) involve $O(N^3)$ work, although (i) is always the more time–consuming step, taking between 85-99% of the total time of the program. Thus to produce an effective parallel code both parts need to be parallelised. We should point out here that this is considerably different from the situation encountered with quantum chemistry codes in which the N^4 workload implied by the four–centre integrals completely dominates the diagonalisation step.

Implementation using PVM

The calculations described above have only modest memory and disk requirements, with a very large problem fitting into 32MB memory and requiring 40MB disk space. Workstations are therefore ideal platforms for running the program, as such requirements can now be satisfied with only modest expense. We have 6 IBM RS/6000 365 and 150 Hewlett Packard 715 workstations arranged in clusters connected by a bridged off section of the LAN (thin wire ethernet). Machines owned by our research group have a minimum of 32MB; those shared with other groups have 16MB memory.

This hardware arrangement favours a master/slave parallelisation algorithm, particularly if the slave process can be designed to have a small memory requirement. In fact we can achieve this by only linking in a subset of the subroutines of the main code and ensuring that only the bare minimum of data is required by them. This selective linking ensures a relative ease of software maintenance – the source code itself leads little modification, with only the makefile being significantly affected.

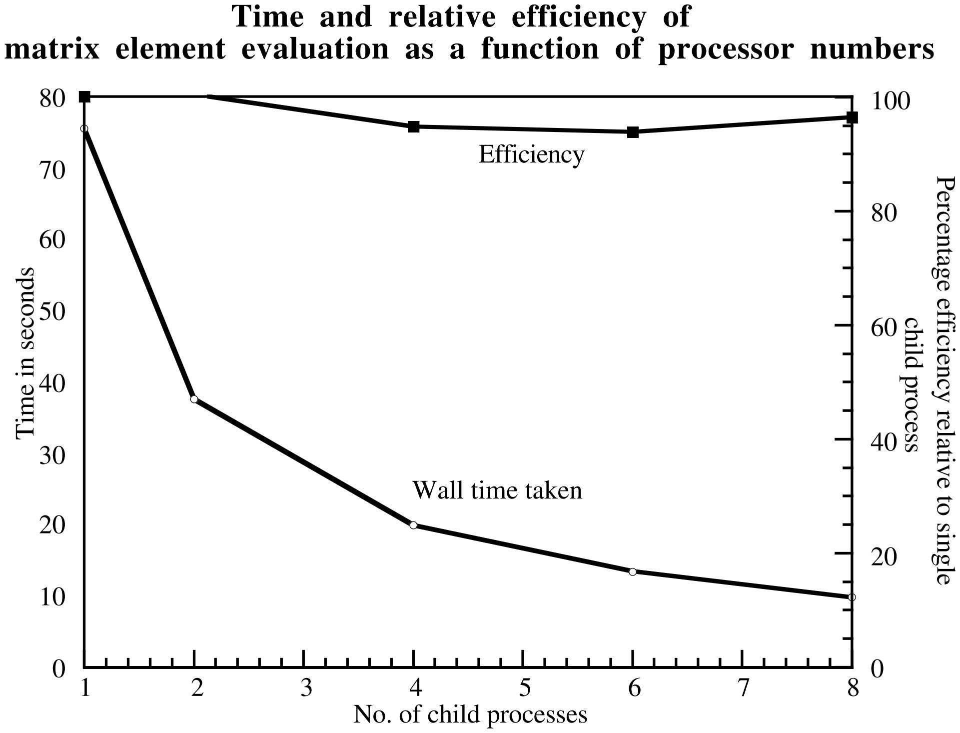
The second decision that was made was to place all of the message handling routines in two subroutines (master and slave). The reason behind this was so we could easily change between using different versions of PVM (with only the commands in these two routines needing to be changed). One possible further advantage is that we can rewrite these routines from the original Fortran to C to enable us to use the PVM routines on machines that have older Fortran compilers that are not compatible with PVM.

The next issue to cover was which routines should first be tackled. For large clusters the most time consuming routines involve the calculation of what are termed multi–centre integrals and contraction with tensors of lower rank. For example, if we introduce basis functions, φ_i, and define a matrix t_{ijk} as the overlap of three of these, then we a significant amount of time is spent, evaluating the quantities (a) $h_{ij} = p_k t_{ijk}$ and (b) $p_k = b_{ij} t_{ijk}$. Typically, we may have 1000 basis functions, so the indices run from 1 to 1000. These contractions need to be evaluated typically twenty times at different stages in the program, with the same matrix t_{ijk} but with different vectors/matrices p and b. Traditionally, this has been implemented by writing the three-centre integrals to disk and reading them in when required. However, with the recent increases in processing power it is now more efficient to recalculate them each time they are needed. The coding is relatively straight forward, with three nested loops surrounding a very large basic–block of code, and the parallelisation of this is relatively easily accomplished.

The simplest algorithm is to split the range of the outermost loop-index (i, say) into groups, g, so that for group g, the index i runs from *ibeg(g)* to *iend(g)*. A group number g is then sent to each slave process which then looks up the start and stop limits on the index of the outer–most loop, *ibeg(g)* to *iend(g),* for this group and performs the calculation for those values of the index. A new group number can then be requested from the master and the calculation repeated with new limits. In this way, the master process continually waits for the slaves to complete the task, and then immediately send them a new group number. This algorithm is automatically load–balancing.

The important questions concern, the number (or equivalently the size) of the groups and the nature of the communication between master and slaves. If operation (a), $h_{ij} = p_k t_{ijk}$, is being implemented, then this produces a matrix of size $O(N^2)$ which needs to be sent back to the master (remember $N \sim 1000$). If operation (b), $p_k = b_{ij} t_{ijk}$, is being implemented only a vector of size $O(N)$ need be communicated after the operation. In both cases the total work done involves $O(N^3)$ floating point operations. Clearly, for the second operation (b), interprocess communication is not an issue, and the vector of results can remain with the slave processes until all the operations are completed, and then each slave sends its own accumulation back to the master process. In process (a), each slave evaluates a number of rows of the matrix h_{ij}. These must then be sent back to the master process. In this case, a smaller number of large groups reduce latency, but make accurate load balancing more difficult. Approximately 10^6 double precision numbers have to be transferred, which makes the properties and quality of the network important for choosing the size of the information packet (and therefore the group). We decided to transfer each part of the resulting matrix as it is generated and have found that timings are not significantly changed and load balancing is quite accurate when using between 50 and 100 groups. In fact, the resulting matrix h_{ij} produced by operation (a) is symmetric, so that only half has to be calculated. As the matrix is split into groups which contain equal numbers of rows, the size of the groups decreases as we reach the end of the algorithm, an effect which aids the load balancing and efficiency of the algorithm and enabling all the slave processes to finish at the same time - rather like filling in a hole with boulders which gradually get smaller so that in the end sand is added and the surface is smooth).

The speed up of these routines is illustrated in the figure below



It is seen that the speed-up factor remains greater than 93% of the maximum linear scaling when using 8 slave processors, and slows little sign of sudden deterioration. The dip at 4 and 6 processors is due to someone working at the console of the workstation during the time the tests were being run.

Future Work

As mentioned in the previous section, multicentre integrals constitute between 85% and 99% of the execution time of a typical production run. The successful parallelisation of these routines is a valuable first step towards a parallel code. The remaining 1 – 15% of the time is spent in matrix multiplications and on the symmetric eigenvalue problem. Work on these is continuing at present, and we would welcome input from anyone with experience with these problems, particularly with the eigenvalue routine. Work is also continuing in making the load balancing algorithm more robust so that machines going down completely or machines suddenly becoming much slower than others (as happens if the load changes suddenly causing massive swapping - a problem we have experienced with machines shared with other groups) should be removed from the virtual machine.