Performance Prediction of Parallel Self Consistent Field Computation

J. Papay, T.J. Atherton, M.J. Zemerly and G.R. Nudd

Parallel Systems Group, Department of Computer Science,
University of Warwick, Coventry, UK
Email: {yuri, tja, jamal, grn}@dcs.warwick.ac.uk

Abstract

This paper presents a methodology for performance prediction of parallel algorithms and illustrates its use on a large scale computational chemistry application. The performance prediction uses a component time characterisation technique which splits up the sequential code into computational components and measures the time for each of them. The parallel algorithm is built from these components by adding communication routines. A "Processor Activity Graph" (PAG) providing a graphical representation of the parallel algorithm runtime behaviour is used for predicting the execution time. For a case study a Self Consistent Field (SCF) computation has been selected which forms the basis of many computational chemistry packages [4,5]. The performance model of SCF computation has been built and the predictions have been compared with the results of measurements. The measurements have been provided on a mesh connected distributed memory parallel computer (128 T800 Parsytec SuperCluster). The prediction error is less than 10%. Performance optimisation of the application has been achieved by reducing the communication overhead and changing the data representation.

1. Introduction

One of the main concerns of parallel computing is to port sequential programs onto a multiprocessor platform. This process requires to take into account the resource limitations of the target machine such as the number of processors, size of memory, latency and bandwidth of the communication network. Performance is one of the key factors that must be taken into account in design, development and tuning of a parallel algorithm. The overall performance is affected by the characteristics of the target machine and the software. Performance prediction allows to change the parallelisation technique and machine parameters and to observe the results of the modifications without porting the code or changing the existing hardware. Typical performance measures for parallel systems are execution time, speedup, system utilisation, throughput, MIPS, MFLOP, etc. Execution time was selected as a performance metric because it is a meaningful measure and widely used in parallel computing. Two approaches are usually used for evaluation of parallelisation techniques (see Figure 1).

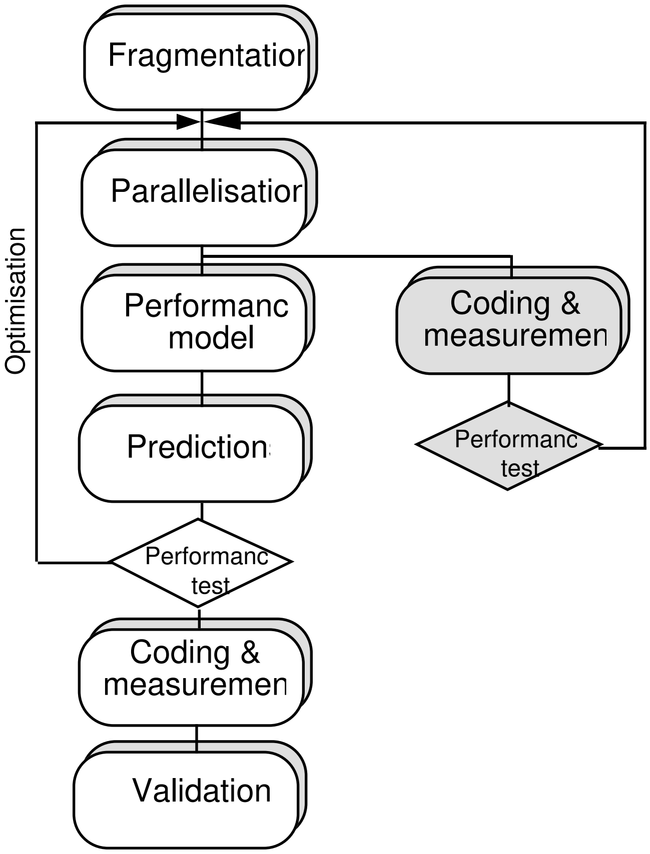

Figure 1 - Parallel algorithm development cycles

The traditional approach (marked by hashing) relies on rewriting the parallel code and all the optimisation experiments are provided on a parallel machine. This approach of trial and error in performance tuning is not efficient since it uses expensive parallel machine time, requires a significant programming effort to modify the parallel code and time to obtain the measurements. A different approach based on early performance prediction is taken in this paper which first develops a performance model of parallel application and uses this model as an experimental testbed for performance optimisation in order to prevent excessive software development costs. The starting point is the fragmentation stage which splits up the sequential code into its components. These are timed and a mathematical expression giving the execution time is derived for each of them. These components and expressions are used later for building the parallel program and its performance model. The second stage is parallelisation which combines the computational modules of the sequential code with the communication modules which are required for parallelisation. The communication models can be derived by benchmarking the parallel machine. The parallelisation does not involve writing the parallel code but requires to derive the algorithm's Processor Activity Graph (PAG). A PAG is a graphical representation of the parallel algorithm and represents the scheduling of computational blocks and communication modules. In the third stage the performance model of the parallel algorithm is built which is based on PAG. The predictions generated by the performance model give the expected execution times of the parallel algorithm for various number of processors. This is an iterative process which carries on until the performance target has been met. Coding and measurements are the most time consuming stages since they require to write the parallel code and to take measurements on a real parallel machine. The validation stage compares the predictions provided by the performance model with the measurements obtained by running the parallel code on a parallel machine.

This paper is organised in the following way: in section 2 a background to performance prediction and computational chemistry is given; section 3 describes the parallelisation of Self Consistent Field computation and presents its performance model; section 4 illustrates the usage of performance model for optimisation.

2. Related Work

An important step in parallel application design is to find out what performance can we expect before we start to develop the actual parallel code. Many researchers characterised parallel systems on the component level (e.g. cache, CPU, etc.) with or without combining the effects of all the components [6]. Hennessy and Patterson provided a comprehensive characterisation of a single micro-processor through characterising its components. Namely, the CPU, the memory hierarchy and the I/O. They argued that the execution time is the most meaningful performance measure for computer systems and they focused their processor characterisation on this measure.

Hockney [7] developed a model for SIMD and MIMD machines based on three program and hardware parameters. The program parameters are: s^s- total number of scalar floating point operations, s^c - total amount of data transferred and q^c- total number of messages. The characteristics of processor, memory system, interconnection network and operating system are given by parameters such as r^s_∞ - asymptotic performance of scalar operations, r^c_∞ - asymptotic communication bandwidth, $n^c_{1/2}$ - half performance message length. The model gives a timing formula which is the sum of times for scalar operations and communications.

Saavedra-Barrera et al. [9] described a way to predict performance by combining benchmarking and hardware characterisation. The idea is to develop and use a machine characteriser which measures the performance of a given system in terms of a Fortran abstract machine. This analysis yields a set of parameters which characterises the system and spotlights its strong and weak points. Each parameter correspondents to the execution time for a primitive operation in Fortran. The authors characterised a large number of machines ranging from workstations to supercomputers.

Basu et al. [1] derived simple analytical execution models for evaluating the performance of parallel message passing systems. They modelled the execution of an algorithm as repetitive steps of computations and communications. Both overlapped and non-overlapped execution models have been analysed. In a non-overlapping model the computations are separated from communications by a zero cost synchronisation. In overlapping models the computation and communication are completely or partially overlapped. Basu et al. used measurements of overlapping fractions between communication and computation from the parallelised code and extrapolated the results to larger systems.

Due to the rapid increase of computing power the computational approach to chemistry has become a powerful alternative to the traditional experiments. Computational Chemistry has become a major consumer of computing resources in academic and industrial applications [3]. The Self-Consistent Field (SCF) computation which forms the heart of many "Ab Initio" Computational Chemistry codes is investigated in this paper. The SCF computation is an approximation of the Schrodinger wave model of particle motion which determines the chemical properties of atoms, the motion of electrons and the interaction between particles. The SCF code has been implemented on various computational platforms such as vector machines and network of workstations [5]. A little work has been done so far on implementing the SCF computation on Distributed Memory MIMD machines [8]. The literature addressing computational chemistry related problems focuses on parallelisation techniques applicable to different parallel platforms. Various data partition techniques for a molecular dynamics program CHARMM have been compared by Hwang et al. [8]. The experimental results obtained on iPSC/860, Paragon, T3D, SP2 parallel machines showed that irregular data partition which takes into account the locality of particles and assigns neighbouring atoms to the same processor provides better performance and scaling properties than regular partitions. The paper compares the advantages and disadvantages o replicated and distributed data partitions and concludes that the distributed data technique although enables to simulate larger problems requires additional programming effort to maintain the distributed data.

Biswas et al. [2] evaluated the implementation of molecular dynamics computation on different parallel platforms: NCube2 and network of DEC 5000 workstations. The authors concluded that this problem since it requires a large number of communications performs better on NCube2. The communication speed for the network of workstations is limited by the bandwidth of Ethernet and also by the inefficient implementation of broadcast in the PVM environment.

This paper addresses the following problems: parallelisation of SCF computation, implementation on the Parsytec SuperCluster, development of an accurate analytical performance model which enables to predict the execution time of the code on various parallel computers and optimisation studies.

3. Parallelisation of the SCF computation

3.1. The sequential code

The sequential SCF computation is an iterative loop which tests the convergence of computation.. The main stages of computation are :

- initialisation
- iterative cycle involves the following steps
    - One Electron Energy Calculation (ONEEL)
    - A Two Electron Energy Calculation (TWOEL)
    - Matrix Diagonalisation (DIAGON) and computation of Density matrix (DENS).
    - output of final results

Two subroutines dominate the computation: evaluation of two-electron integrals and diagonalisation. The former scales as $O(N^4)$, while the latter scales as $O(N^3)$, where N is the number of atoms. The results of profiling the sequential code are presented in Table 1.

Time, s	INIT	ONEEL	TWOEL	DIAG	TDENS
atom=1	4.16	0.47	41.86	1.8	0.05
atom=2	4.53	3.02	831.84	16.64	0.35
atom=4	4.52	16.8	12952	123.76	2.12

Table 1 - Execution time distribution

The computational models have been derived by representing each subroutine by a Software Execution Graph (SEG) and an analytical expression which gives the execution time. Here for illustration purposes performance models of two key subroutines are given, these are ONEEL (Figure 2) and TWOEL (Figure 3)

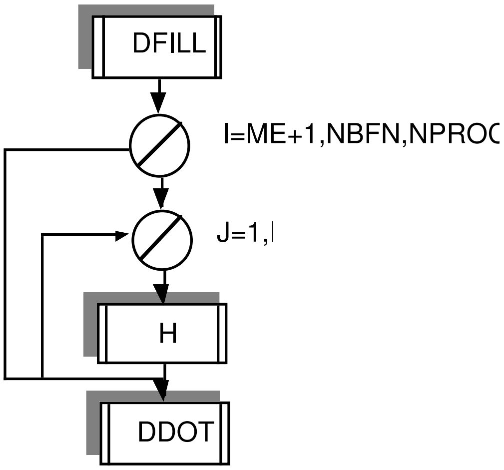

Figure 2 - Software execution graph of ONEEL

The analytical expression for the execution time of ONEEL subroutine is given by (1):

$$T_{ONEEL} = t_{DFILL} + \sum_{I=ME+1,NPROC}^{NBFN} (t_{LOOP1} + \sum_{J=1}^{I} (t_{LOOP2} + t_H)) + t_{DDOT} \qquad (1)$$

where t_{DFILL}, t_{LOOP1}, t_{LOOP2}, t_H and t_{DDOT} are computation modules represented by their execution times.

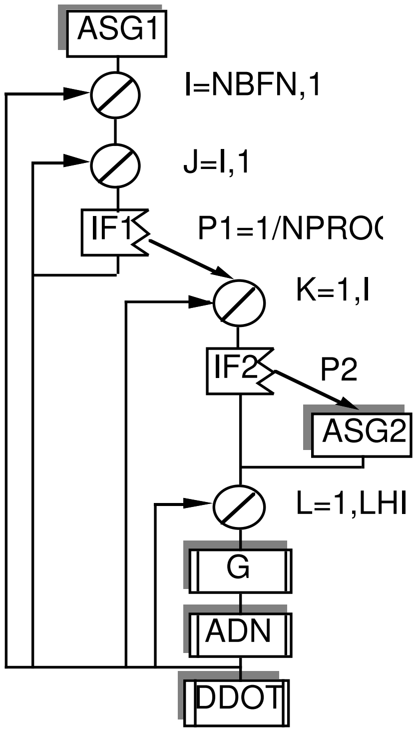

Figure 3 - Software execution graph of TWOEL subroutine

The analytical expression for the execution time of TWOEL subroutine is given by (2):

$$\begin{aligned} T_{TWOEL} = t_{ASG1} + \sum_{I=NBFN}^{1} (t_{LOOP1} + \sum_{J=I}^{1} (t_{LOOP2} + t_{IF1} + P1 * \sum_{K=1}^{I} (t_{LOOP3} + t_{IF2} \\ + (1 - P2) * \sum_{L=1}^{LHI} (t_{LOOP4} + t_{G} + t_{ADN}))))) + t_{DDOT} \end{aligned} \tag{2}$$

where t_{ASG1}, t_{ASG2}, t_{LOOP1}, t_{LOOP2}, t_{IF1}, t_{LOOP3}, t_{IF2}, t_{LOOP4}, t_G, t_{ADN}, t_{DDOT} are computation modules. NPROC is the number of processors, P1 and P2 are probabilities.

3.2. Parallelisation of SCF computation

For parallelisation of the SCF computation a functional decomposition has been selected. The data involved in computation is replicated on each processor. The computational work is represented by modifying the element (I,J) of the Fock matrix which represents the energy level of particles. This operation can be performed in parallel since there is no datadependency between matrix elements. The size of lower triangular Fock matrix is 15N(15N+1)/2, where N is the number of atoms. For the ONEEL computation separate rows of the Fock matrix are selected by each processor. In TWOEL computation the work is distributed by elements of the Fock matrix rather than rows. This decomposition provides better load balancing. The output of parallelisation stage is the Processor Activity Graph (PAG) which is a graphical representation of the parallel computation. The advantages of PAG are the following:

- shows the dynamic behaviour of the parallel algorithm
- shows the overlap between computation and communication, the level of parallelism and load imbalance.
- enables to predict the execution time of the parallel algorithm.

A PAG for a single iteration of parallel SCF computation running on four processors is given in Figure 4.

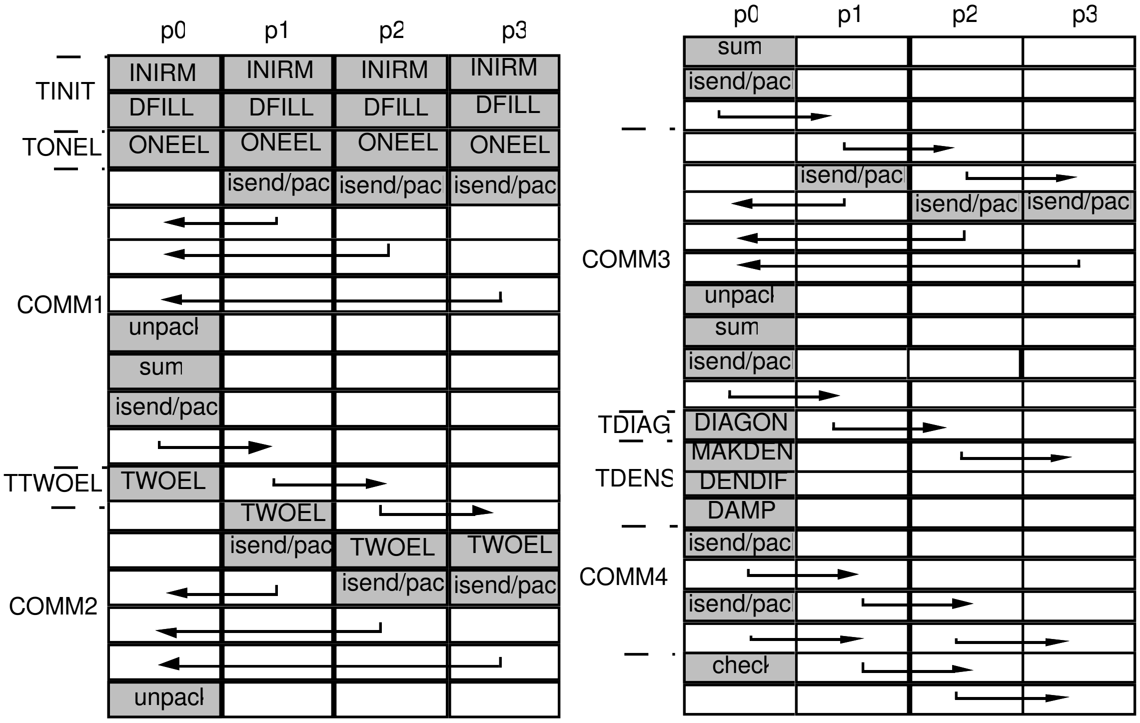

Figure 4 - Processor activity graph of the parallel SCF computation

The PAG shows that communications required for the parallel algorithm are: point-to-point communication and multicast. The analytical models for these communications can be derived by benchmarking the parallel machine.

3.3. Models of PVM communications

Sending a message in PVM requires three steps, these are: initsend, pack and send. receiving processor executes receive and unpack. Initsend (PVMFINITSEND) resets the se buffer every time new data is to be sent. Measurements taken on the Parsytec SuperCluster Initsend showed that it is a constant and equal to t_{isend} = 111.27 μs (with standard deviation of 18.95). The cost of point-to-point communication for the Parsytec SuperCluster can b modelled by equation (3):

$$T_{com}(l,d) = \begin{cases} \alpha_1 + \beta_1 l & l \le p_s \;\&\; d = 1 \\ \alpha_2 + \beta_2 l & l > p_s \;\&\; d = 1 \\ \alpha_3 + \beta_3 l + \gamma_3 d + \delta_3 ld & l \le p_s \;\&\; d > 1 \\ \alpha_4 + \beta_4 l + \gamma_4 d + \delta_4 ld & l > p_s \;\&\; d > 1 \end{cases} \tag{3}$$

where $p_{S,}$ is the packet size (96 bytes), l is the message length in bytes and d is the dista (number of hops between processors).

The parameters of the model for the PVM communications are given in Table 2. The parameters for equation (3) have been obtained from a ping-pong benckmark [7] by lea squares fitting of measured communication times.

	α	β	γ	δ
$l \le p_s$ & $d = 1$	2561.6	0.534	-	-
$l > p_s$ & $d = 1$	2499.03	1.216	-	-
$l \le p_s$ & $d > 1$	2491.31	-0.0001	23.17	0.768
$l > p_s$ & $d > 1$	2267.12	1.834	110.03	0.0039

Table 2 - Parameters of the quiet network communication model

Figure 4 shows the performance of point-to-point communication for short messages (32 96b).

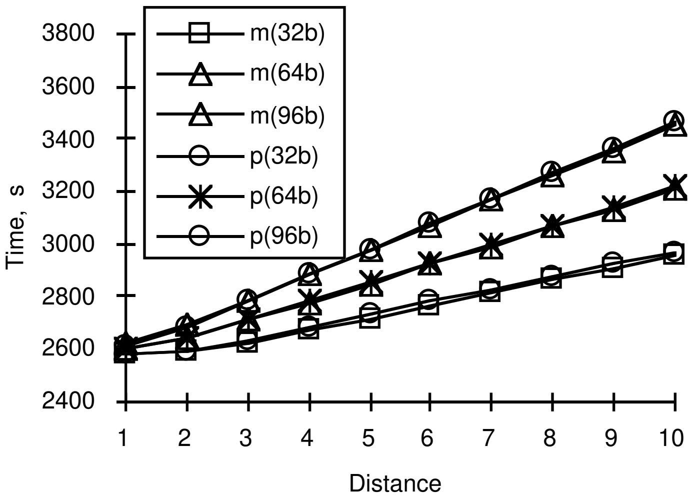

Figure 5 - Measurements (m) and predictions (p) of point-to-point communication for short messages

Figure 6 shows the absolute percentage differences between predicted and measured execution times.

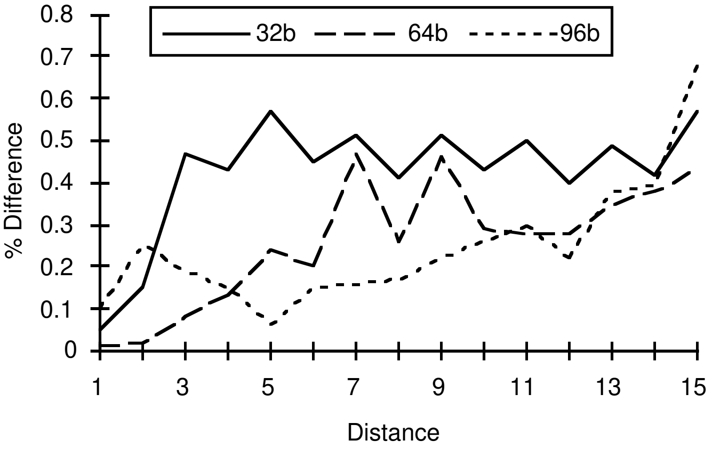

Figure 6 - Absolute percentage differences between predictions and measurements of point-to-point communication for short messages

Figure 7 shows the performance of point-to-point communication for long messages (2Kb 8Kb).

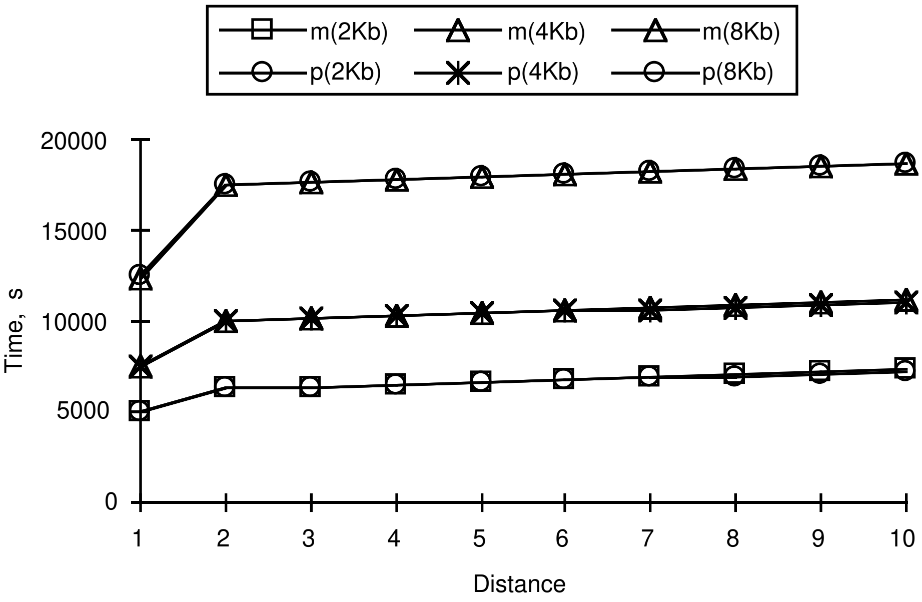

Figure 7 - Measurements and predictions of point-to-point communication for long messages

Figure 8 shows the percentage differences between the predicted and measured times for long messages.

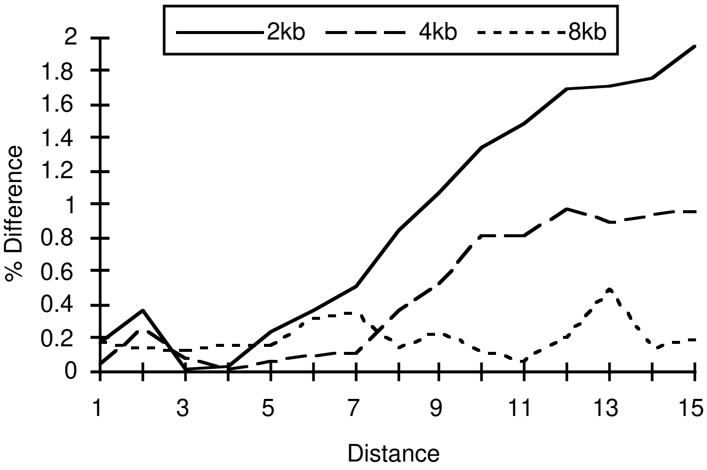

Figure 8 - Absolute percentage difference between predictions and measurements for long messages

Multicast is a frequently used communication pattern in many numerical applications. The analytical model for multicast is given by:

$$T_{mcast}(l, p) = \begin{cases} \alpha_1 + \beta_1 l \log(l) + \gamma_1 p \log(p) + \delta_1 l p \log(l) \log(p) & l \leq p_s \\ \alpha_2 + \beta_2 l \log(l) + \gamma_2 p \log(p) + \delta_2 l p \log(l) \log(p) & l > p_s \end{cases} \quad (4)$$

where l is the message length and p is the number of processors. The parameters for equa (4) have been obtained by least squares fitting of measurements obtained from Parsyt SuperCluster. These parameters are given in Table 3.

	α	β	γ	δ
$l \le p_s$	6617.91	1.87571	125.9	0.132712
$l > p_s$	4743.9	0.123052	239.455	0.0390338

Table 3 - Parameters of the multicast model

Figure 9 gives the measurements and predictions for multicast for different message sizes (3 96b) and number of processors.

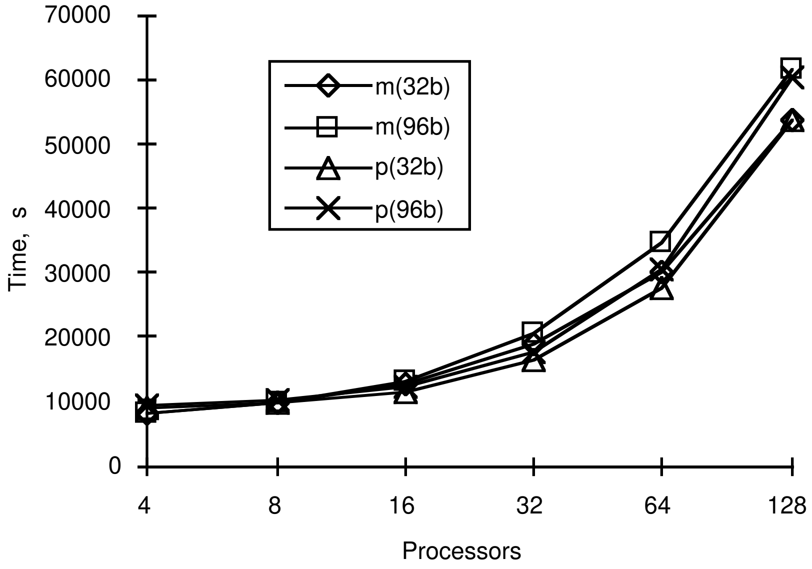

Figure 9 - Measurements (m) and predictions (p) of multicast for short messages

Figure 10 shows the absolute percentage differences between predicted and measured execution times.

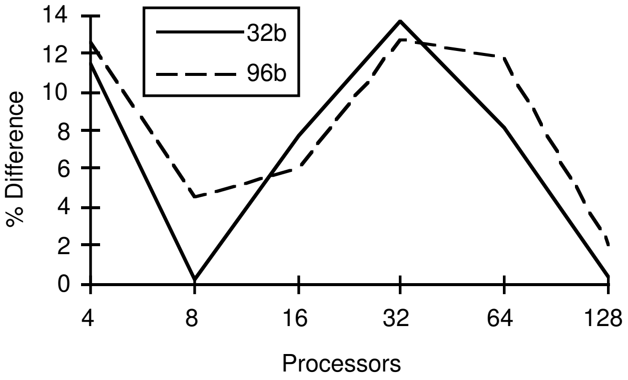

Figure 10 - Absolute percentage differences between predictions and measurements for short messages

Figure 11 shows the measurements and predictions for multicasting long messages (2kb-8k

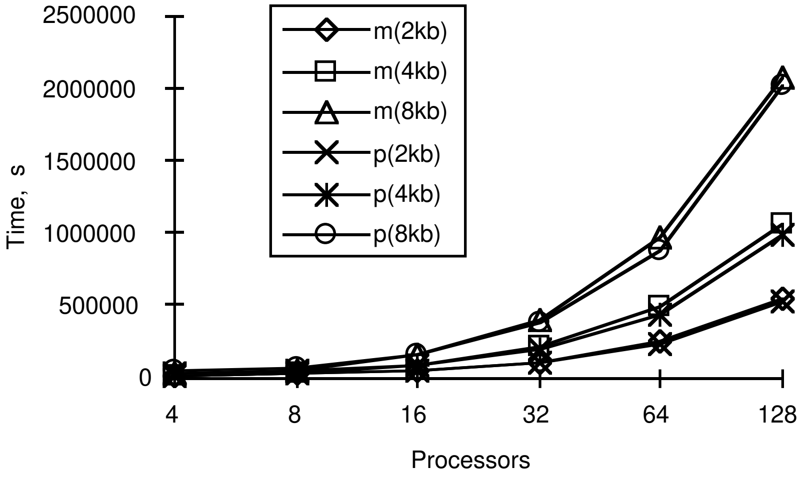

Figure 11 - Measurements (m) and predictions (p) for multicasting long messages

Figure 12 shows the absolute percentage differences between predicted and measured exec times.

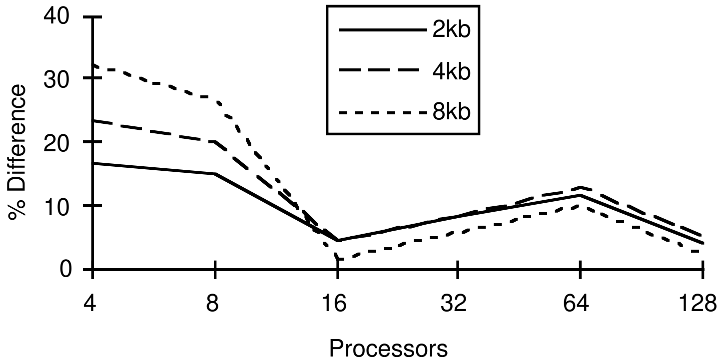

Figure 12 - Absolute percentage differences between predictions and measurements for multicasting long messages

Experimental measurements on packing and unpacking showed that the cost of packing is equal to unpacking. An analytical model for packing/unpacking of double (REAL8) data is given by (5):

$$t_{pack}(l) = 286.91 + 8.2\,l \qquad (5)$$

Figures 5-12 show that the communication models used in this paper are accurate and with a confidence can be used for performance predictions.

3.3. A performance model of parallel SCF computation

The performance model is derived from the Processor Activity Graph (PAG) which is a graphical representation of parallel computation. The execution time of parallel SCF computation is given by analytical expressions as follows

$$\begin{aligned} T_{xp}(N,p) = T_{INIRM} + T_{DFILL} + \text{iteration}^*\,(&T_{ONEEL} + T_{COMM1} + T_{TWOEL} + T_{COMM2} + \\ &T_{COMM3} + T_{DIAG} + T_{DENS} + T_{COMM4}) \end{aligned} \qquad (6)$$

$$T_{COMM1} = t_{coll}(len1, p) + (p-1)^*\,t_{add} + t_{isend} + t_{pack}(len1) + t_{mcast}(len1, p-1) \qquad (7)$$

$$T_{COMM2} = t_{coll}(len1, p) + (p-1) * t_{add} + t_{isend} + t_{pack}(len1) + t_{mcast}(len1, p-1) \qquad (8)$$

$$T_{COMM3} = t_{coll}(len2, p) + (p-1) * nnbfn * t_{add} + t_{isend} + t_{pack}(len2) + t_{mcast}(len2, p-1) \qquad (9)$$

$$T_{DENS} = T_{MAKDEN} + T_{DENDIF} + T_{DAMP} \qquad (10)$$

$$T_{COMM4} = t_{isend} + t_{pack}(len2) + t_{mcast}(len2, p-1) + t_{isend} + t_{pack}(len2) + t_{mcast}(len2, p-1) \qquad (11)$$

where T_{xp} is the execution time of the parallel program, T_{INIRM}, T_{DFILL}, T_{ONEEL}, T_{COMM1}, T_{COMM2}, T_{COMM3}, T_{COMM4}, T_{TWOEL}, T_{DIAG}, T_{DENS}, T_{MAKDEN}, T_{DENDIF}, T_{DAMP} are execution times of subroutines and communication modules, t_{coll} is the cost of collection, len1 (8bytes) and len2 (nnbfn =15*N*(N+1)/2 bytes).

3.4. Validation of the performance model

Validation involves comparison of predictions generated by the performance model with measurements. The SCF code has been ported on Parsytec Supercluster, for communications the PVM/PARIX message passing library had been used. The measurements have been taken on p_0 processor for 1, 2 and 4 atom configurations, the results of measurements and predictions are given in Table 4 and Figure 13.

N\p	1	2	4	8	16	32	64	128
measurement(1	48.43	27.89	17.73	13.11	11.84	13.62	19.89	36.79
prediction(1)	47.88	27.77	17.74	13.33	12.43	14.74	21.66	37.62
measurement(2	856.49	442.16	235.23	132.98	85.75	69.91	78.04	119.43
prediction(2)	853.38	440.78	234.34	132.89	86.47	72.19	83.68	128.72
measurement(4	13099.2	6627.72	3390.96	1779.71	985.76	614.41	486.87	N/A
prediction(4)	13034.47	6601.8	3386.87	1786.67	991.14	625.94	505.56	578.04

Table 4 - Measured and predicted execution times

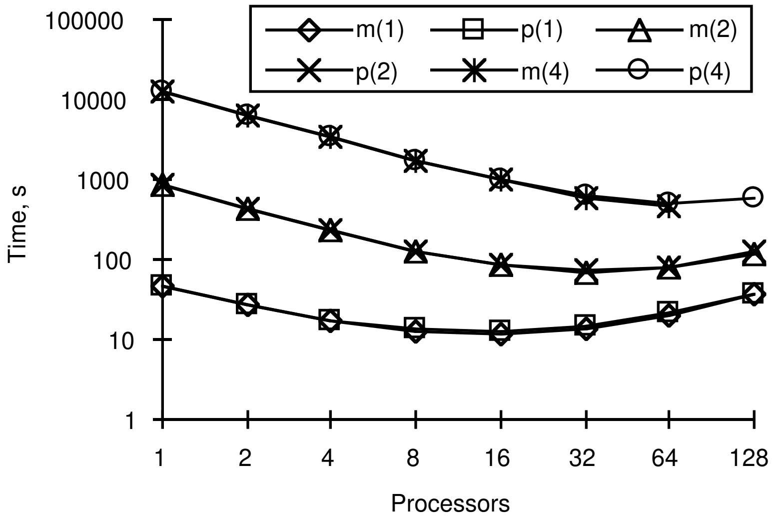

Figure 13 - Execution time measurements (m) and predictions (p) for 1, 2 and 4 atoms

Figure 14 shows the absolute percentage differences between predicted and measured execution times for various number of atoms and processors.

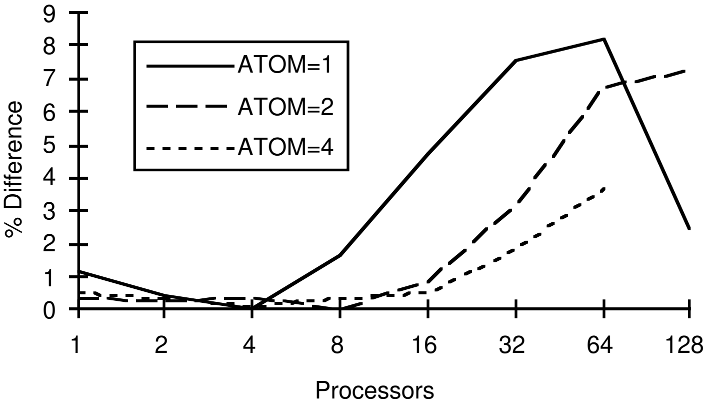

Figure 14 - Absolute percentage difference between measurements and predictions

The comparison of predictions and measurements shows that the accuracy of the performance model is about 8%. Since the accuracy of the performance model is acceptable further optimisations and performance tuning can be provided on the model rather than on the parallel computer. This approach provides fast prototyping, it is flexible and does not require expensive and time consuming experiments.

4. Optimisation of the SCF computation

The optimisation task is concerned with reducing the time the algorithm spends in computations and communications. The computation can be optimised by selecting an appropriate parallelisation strategy which yields good load balancing. The communication time can be reduced by using techniques such as locality of access, avoidance of hotspots in the communication network, overlap of computation and communication. The performance model of SCF computation developed in the previous section enables to investigate various optimisation techniques and to obtain accurate performance predictions without the need to spend time to rewrite the parallel code and taking the measurements. The optimisation technique addressed here intends to reduce the communication overhead by combining messages and modifying the data packing format. Combining messages allows to save on the message startup time which for PVM/PARIX is 1111 μsec. To validate the accuracy of predictions we compare them with measurements of the optimised code (Table 5, Figure 15).

N\p	1	2	4	8	16	32	64	128
measurement(1)	48.43	27.54	17.24	12.33	10.47	11.09	14.65	26.15
prediction(1)	47.88	27.49	17.31	12.6	11.09	12.19	16.64	27.61
measurement(2)	856.49	440.39	232.73	129.05	78.94	57.07	53.29	70.22
prediction(2)	853.38	439.59	232.42	129.5	80.13	59.93	59.55	80.75
measurement(4)	13099.2	6620.82	3381.26	1764.33	958.31	574.27	384.93	N/A
prediction(4)	13034.47	6597.43	3379.64	1773.73	966.75	578.64	412.4	393.1

Table 5 - Measured and predicted execution times for 1,2 and 4 atoms

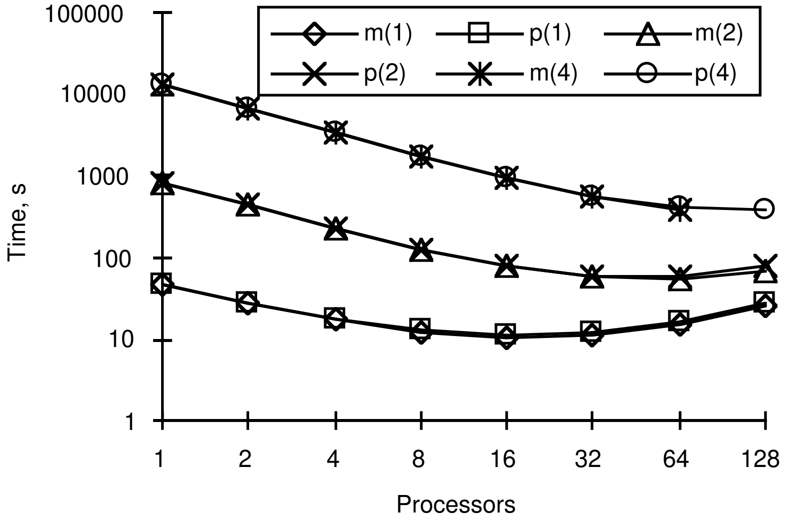

Figure 15 - Execution time measurements (m) and predictions (p)

Figure 16 shows the absolute percentage differences between predicted and measured execution times for various number of atoms and processors.

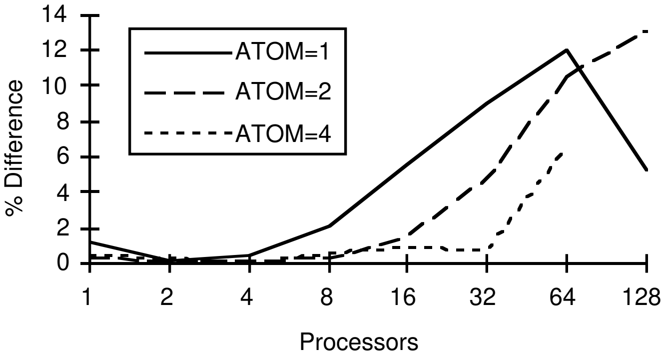

Figure 16 - Absolute percentage difference between the measurements and the predictions for 1,2 and 4 atoms

The validation has shown that the performance model preserved its accuracy and it can be used as a testbed for optimisation studies. The execution time improvements (in percentages) obtained from the optimised SCF computation are shown in Figure 17.

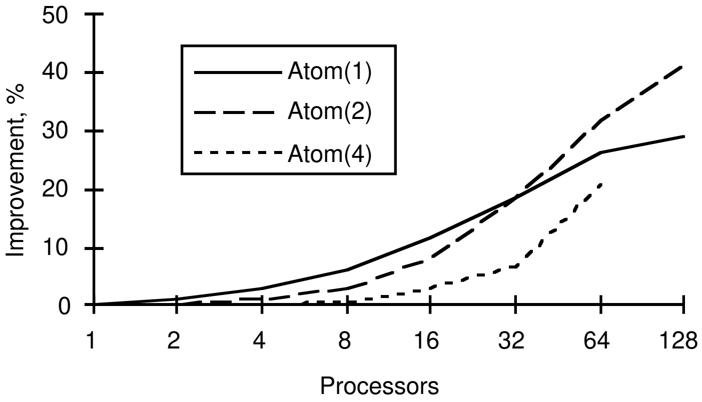

Figure 17 - Performance improvements of the optimised SCF computation

5. Summary

In this paper we presented a methodology for building and optimising large parallel program using early performance prediction. The experience with SCF computation showed tha parallelisation and performance prediction do not require a detailed knowledge of quantu chemistry background. This case study demonstrated that scientific simulations have a lar runtime and require massive computational resources. Direct experimentation on a parall machine is time consuming, expensive and requires a substantial programming effor Therefore it is beneficial to develop a performance model of the application which provides a experimental testbed for optimisations without the need to use expensive computation resources and spend time to modify the parallel program. In this paper a performance model parallel SCF computation has been developed which has been validated by measurements showed 8% error. This performance model has been used for optimisation studies ar predicted performance improvement with an error of 12%.

Acknowledgement

This work is funded by the ESPRIT project 6942 (PEPS) - Performance Evaluation of Parallel Systems. The authors wish to thank Mr. Trevor Chambers, Mr. Michael Stevens (National Physical Laboratory), Dr. Paul Sherwood (Daresbury Laboratory) and the Parallel Systems Group at Warwick University for the valuable discussions of this work.

Bibliography

[1] A. Basu, S. Srinivas, K. G. Kumar, A. Paulraj. A Model for Performance Prediction of Message Passing Multiprocessors Achieving Concurrency by Domain Decomposition. Proc. of the Joint Int. Conf. on Vector and Parallel Processing, (editor) Burkhart, H., Lecture Notes in Computer Science, 457, 10-13 September, Zurich, Switzerland, Springer-Verlag, pp. 75-85, 1990.

[2] R. Biswas, D.E. Turner, A. Zdetis. Comparison of parallel and distributed computing for classical molecular dynamics, Ames Laboratory - USDOE, Department of Physics and Astronomy, Iowa State Uniersity, 1993 (Submitted to Computer Physics Communications)

[3] D.A. Case. Computer Simulations of Protein Dynamics and Thermodynamics. IEEE Computer/CS&E, October 1993, 47-57.

[4] M. F. Guest and J. Kendrick, GAMESS Users Manual. SERC Daresbury Laboratory CCP1/86/1, 1986 .

[5] M.F. Guest et. al. Parallelism in Quantum Chemistry. Parallel Supercomputing, SERC Daresbury Laboratory, 1992.

[6] J. L. Hennessy and D. A. Patterson. Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, 1990.

[7] R. W. Hockney, C.R. Jesshop: Parallel Computers 2, Architectue, Programming and Algorithms , Second edition, Institute of Physics Publishing, Bristol, 1988.

[8] Y.Hwang, R. Das, J.H. Saltz, M. Hodoscek, B. R. Brooks. Parallelizing Molecular Dynamics Programs for Distributed Memory Machines, IEEE Computational Science & Engineering, Summer 1995, 18-29.

[9] R. H. Saavedra-Barrera, A. J. Smith, and E. Miya. Machine characterisation based on an abstract high-level language machine, IEEE Trans. on Computers 38 (12) (1989) 1659-1679.