Load Balancing for the Electronic Structure Program GREMLIN in a Very Heterogenous SSH-Connected WAN-Cluster of UNIX-Type Hosts

Siegfried Höfinger[1]

Dipartimento di Chimica "G. Ciamician",
Universita' degli Studi di Bologna,
Via F. Selmi 2,
I-40126, Bologna, Italy
sh@ciam.unibo.it
http://www.ciam.unibo.it

Abstract. Five far distant machines located at some French, Austrian and Italian research institutions are connected to a WAN-cluster via PVM 3.4.3. The secure shell protocol is used for connection and communication purposes between the different hosts. Operating-Systems, architectures and cpu-performances of all the 5 machines vary from LINUX-2.2.14/INTEL PPro-200MHz, over LINUX-2.2.13/INTEL PII-350MHz, OSF I V5.0/ALPHA EV6-500MHz, IRIX64 6.5/MIPS R10000-195MHz, up to IRIX64 6.5/MIPS R12000-300MHz. An initial benchmark run with the Hartree Fock program GREMLIN reveals a speed difference of roughly a factor 7x between the slowest and the fastest running machine. Taking into account these various speed data within a special dedicated load balancing tool in an initial execution stage of GREMLIN, may lead to a rather well balanced parallel performance and good scaling characteristics for this program if run in such a kind of heterogenous Wide Area Network cluster.

1 Introduction

Computer Science and Industry has made great progress in recent years and as a result of this, the average desktop personal computer as of today has become superior in many aspects to his supercomputer analogues. The other most rapid emerging field has been the internet and internet based technology, and therefore todays probably most potential computing resources might be lying in these huge number of ordinary internet computers, that are accessible in principal to everyone else on the net, but mainly remain idle and serve for minor computational tasks. Scientific research in many areas however suffers from limited access to computational resources and therefore great attention should be payed to development efforts especially focusing on parallel and distributed computing strategies and all the problems connected to them.

One such example for a really demanding scientific discipline is ab initio quantum chemistry, or electronic structure theory, which currently is about to enter the field of mainly application oriented sciences and bio-sciences as well, and thus experiences a never foreseen popularity, which all in all may be due to awarding the Nobel Price in Chemistry to J.A. Pople and W. Kohn in 1998.

In a previous article [1] we introduced one such quantum chemical program, which shall from hereafter be called **GREMLIN**, that solves the *time independent Schrödinger equation* [2] according to the *Hartree Fock Method* [3] [4]. One of the main features of this program had been the capability to execute the most expensive part in it in parallel mode on distributed cluster architectures as well as on shared memory multiprocessor machines [5]. In addition, what makes this application particularly attractive for a distributed computing solution, is its modest fraction in communication time, which on the other hand implies a principal possible extension to a *Wide Area Network* (**WAN**) cluster, where the individual "working" nodes are usually formed form a number of UNIX-type machines[1] of usually hetereogenous architecture and the connection between them is simply realized from the ordinary low-bandwidth/high-latency internet.

Following previous results [1], a properly balanced distribution of the global computational work requires some basic interference with the theoretical concept of recursive *ERI* (Electron Repulsion Integrals) computation [6]. However, taking into account a system inherent, partial inseparability of the net amount of computational work, allows an estimation and decomposition into fairly equal sized fractions of node work, and from this adequate node specific pair lists may be built. The present article intends to describe, how one may extend this concept to an additional consideration of different node performance, since the previous study was based on multiprocessor machines made of equally fast performing CPUs.

1.1 Computational Challenge

Here we briefly want to recall, what makes ab-initio electronic structure calculation a real computational challenge. The main problem lies in the evaluation of ERIs, *the Electron Repulsion Integrals*, which are 6-dimensional, 4-center integrals over the basis functions φ.

$$ERI = \int\limits_{\boldsymbol{r_1}} \int\limits_{\boldsymbol{r_2}} \varphi_i(\boldsymbol{r_1})\varphi_j(\boldsymbol{r_1}) \frac{1}{|\boldsymbol{r_2} - \boldsymbol{r_1}|} \varphi_k(\boldsymbol{r_2})\varphi_l(\boldsymbol{r_2}) \boldsymbol{dr_1 dr_2} \tag{1}$$

and the basis functions φ_i are expanded in a series over *Primitive Gaussians* χ_j

$$\varphi_i(\boldsymbol{r}) = \sum_j d_{i,j}\ \chi_j(\boldsymbol{r}) \ , \tag{2}$$

[1] although PVM 3.4.3 would support WIN32 like OSs as well

which typically are *Cartesian Gaussian Functions* located at some place (A_x, A_y, A_z) in space[2] [7] [8].

$$\chi_j(\boldsymbol{r}) = N_j(x - A_x)^l (y - A_y)^m (z - A_z)^n \; e^{-\alpha_j(\boldsymbol{r}-\boldsymbol{A})^2} \tag{3}$$

Although somewhat reduced from numerical screening, the principal number of ERIs to be considered grows with the 4th power of the number of basis functions, which themselve is proportional to the number of atoms in the molecule. However, since the quality of the employed basis set must be kept high in order to enable quantitative reasoning, the according number of ERIs very soon exceeds conventional RAM and diskspace limits and thus becomes the only limiting factor at all. For example, a simple, small molecule like the amino acid alanine (13 atoms), that has been used as a test molecule throughout this present study, at a basis set description of aug-cc-pVDZ quality [9] [10] (213 basis functions of S, P and D type) leads to a theoretical number of approximately 260 x 10^6 ERIs, which requires about 2.1 GigaByte of either permanent or temporary memory and goes far beyond usual available computational resources.

Fortunately there is partially independence in the mathematical action of these many ERIs and one may solve the problem in a so called "Direct" way, which means, that a certain logical block of related ERIs is first calculated recursively[3], then the action of these block on all the corresponding Fock-matrix elements – from which there luckily are only a number of $(number\ of\ basisfunctions)^2$ – is considered, and then the procedure is repeated and a new block of ERIs overwrites the old one and thus only a small amount of working memory is permanently involved. Further complifying is the fact, that one has to respect a hierarchic structure in spawning the space to the final primitive cartesian gaussian functions χ_j, where, following the notation introduced in [1], a certain center $\boxed{\text{i}}$ refers to an according block of contracted shells → (j)...(k), from which each of them maps onto corresponding intervals of basis functions l...m and the later are expanded from primitive cartesian gaussian functions χ_j as seen from (2). Therefore, after defining a particular centre quartette $\boxed{\text{i1}}\ \boxed{\text{i2}}\ \boxed{\text{i3}}\ \boxed{\text{i4}}$, all the implicit dependencies down to the primitive cartesian gaussians χ_j must be regarded and as a consequence rather granular blocks of integrals must be solved all at once, which becomes the major problem when partitioning the global amount of integrals into equally sized portions.

1.2 Speed Weighted Load Balancing

Concerning parallelization, we follow a common downstream approach and define node specific pair lists, that assign a certain subgroup of centre quartettes

[2] An S-type basis function will consist of primitive gaussians with $l = m = n = 0$, a P-type however of primitives with $l + m + n = 1$, which may be solved at 3 different ways, either $l = 1$ and $m = n = 0$, or $m = 1$ and $l = n = 0$, or $n = 1$ and $l = m = 0$. D-type specification will likewise be $l+m+n = 2$ and similarly F-type $l+m+n = 3$.

[3] All complicated ERI-types ($l+m+n > 0$) may be deduced from the easier computed $(S_i, S_j|S_k, S_l)$ type.

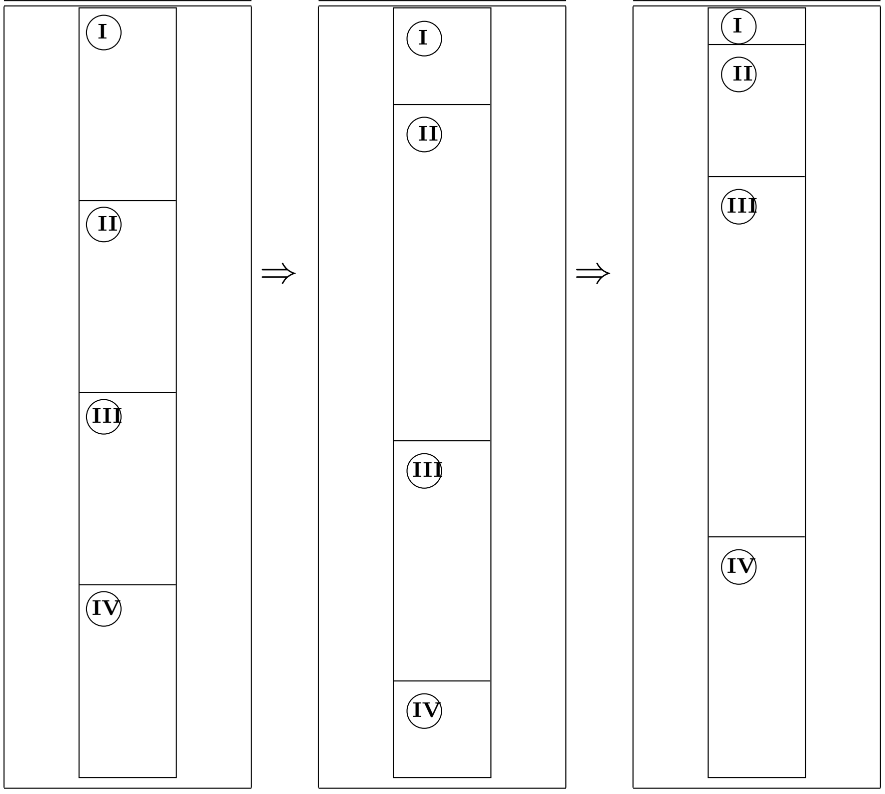
to each of the individual nodes, which then shall work independently on their corresponding partial amount of global computational work. Ideally these pair lists are built in a way, such that each of the nodes needs the same time for executing its according fraction of net work. Suppose the total number of theoretical centre quartettes is represented from the area of a rectangle, like shown in Table 1, and one wants to distribute these many centre quartettes now onto a number of parallel executing nodes, then the simplest method would certainly be an arithmetic mean scheme (left picture in Table 1), where the theoretic number of centre quartettes is devided by the number of nodes and all of them get exactly this arithmetic mean fraction to work on. Due to the fact that several centres may now substantially differ in the number of deducable contracted shells $\rightarrow$ basis functions $\rightarrow$ primitive gaussians, this simple procedure has been shown to not be applicable for the case of distributing global computational work for recursive ERI calculation done in parallel [1]. In fact, instead, one had to take into account all these hierarchic dependencies down to the level of primitive gaussians χ_j in order to be able to estimate the real fraction one particular centre quartette actually had of the global amount of work measured in terms of theoretic establishable quartettes of primitive gaussians now. However, following this pathway led to considerable improvements in parallel performance and the according pair lists of center quartettes may then be symbolized like shown in the medium picture of Table 1. Note, that up to now, the indicated, individual 4 nodes are still considered to all operate at the same CPU speed and despite the fact, that the actual number of centre quartettes each node has to process has become apparently different now, the execution time for each of the nodes is now much more comparable – if not equal – to each other, which stands in great contrast to the arithmetic mean picture.

Going one step further and assuming different node performance next, would change the situation again. For example, let us hypothetically think of a parallel machine, where node II is twice as fast as node I, and node III and IV are running three times and four times as fast as I respectively. Then, we could equally well think of a parallel machine made up of 10 nodes of the speed of type I, divide the global amount of work (measured again at the innermost level of potential primitive gaussian quartettes) into 10 equal sized fractions, and let the fastest node (IV) work on 4 portions of that estimated unit metric, while node III and II get $\frac{3}{10}$ and $\frac{2}{10}$ of the global work and node I will just deal with the remaining $\frac{1}{10}$ of the global amount. The schematic representation of such a kind of partitioning is given in the right picture of Table 1. On the other hand, one could obtain a theoretical speed up factor of 2.5 ($= \frac{10}{4}$) for such a case[4], if at first instance communication time is said to be extremly small and bare serial execution intervals are neglected completely.

[4] compared to the situation where 4 equally fast performing CPUs operate on already load balanced pair lists

Table 1. Comparision between different partitioning schemes of the outermost loop over centre quartettes, represented from the partial areas of the 4 rectangles, that stand for node specific fractions of the global amount of theoretical combinations of centre quartettes. For the Speed Weighted Load Balancing, node II is assumed to be twice as fast as I, and nodes III and IV, are said to be three times and four times as fast as I.

Symbolization of Arithmetic Mean Partitioning of Centre Quartettes With Respect to Equally Fast Performing Nodes	**Symbolization of Load Balanced Partitioning of Centre Quartettes With Respect to Equally Fast Performing Nodes**	**Symbolization of Speed Weighted Load Balanced Partitioning of Centre Quartettes With Respect to Differently Fast Performing Nodes**



1.3 Computational Implementation of Speed Weighted Load Balancing

As already explained within Sect. 1.2, we want to distribute the theoretical number of centre quartettes onto a number of nodes of different CPU performance. For this purpose we implement the following steps:

1. Determine the number of participating hosts (NMB) and their according relative speed factors (SPEED[I]). The speed factor is the relative performance of a particular node related to the slowest CPU. So it will either become 1.0 (weakest node), or greater than 1.0 for faster CPUs.
2. Estimate the net amount of computational work (GLOBAL WORK) at the level of quartettes of primitive gaussians to be considered.
3. Form a unit portion (PORTN) of the dimension of

$$\mathrm{PORTN} = \frac{\mathrm{GLOBAL\ WORK}}{\sum_{\mathrm{I}=1}^{\mathrm{NMB}} \mathrm{SPEED[I]}} \tag{4}$$

4. Loop again over all quartettes of centres and the related contracted shells and basis functions and primitive gaussians either, as if you were calculating GLOBAL WORK, and successively fill the upcoming pair lists for centre quartettes until in the work estimation variable (usually a simple counter, incremented for each new quartette of primitive gaussians) becomes of the size of PORTN*SPEED[I]; then leave the current pair list writing for node I and switch forward to the next node and start with setting up pair lists for this one.

2 Computational Set-Up

In this section we just want to focus on the practical aspects of setting up a **W**ide **A**rea **N**etwork cluster and running **GREMLIN** thereon in the described speed-weighted, load-balanced way.

2.1 WAN Cluster Description

Five university sites at **GUP** LINZ (A) (2 nodes), **RIST**$^{++}$ SALZBURG (A) (1 node), **ICPS** STRASBOURG (F) (1 node) and **G. Ciamician** BOLOGNA (I) (1 node) were connected via the **P**arallel **V**irtual **M**achine (PVM rel.3.4.3) package [11]. One of the remarkable nice features of this release is, that the communication between different, interconnected hosts may be realized with the **s**ecure **sh**ell protocol, that implements RSA authentication with 1024 bit long public/private keys. According to what has been said above, at first one needed to get an overview of the different node perfomance of all the individual hosts involved in the cluster. Therefore an initial benchmark run with the PVM version of **GREMLIN** (1 node at each location seperately) on a very small training

Table 2. Conditioning and description of the individual host performance in the WAN cluster. The data is due to a Hartree/Fock DSCF calculation on glycine/631g with the program **GREMLIN**. (Result: -271.1538 Hartree in 22 iterations)

Physical Location	Architecture/ Clock Speed/ RAM/2L-Cache	Operating System	real [s]	usr [s]	sys [s]	Rel. Speed
node I G.C. BOLOGNA Italy	INTEL Dual PPro 200 MHz 256 MB/512 KB	LINUX 2.2.14	2431	159	0	1.000
node II ICPS STRASBOURG France	MIPS R10000 200 MHz 20 GB/4 MB	IRIX 64 6.5	1186	9	2	1.934
node III GUP LINZ Austria	INTEL PII 350 MHz 128 MB/512 KB	LINUX 2.2.13	1167	60	1	2.054
node IV GUP LINZ Austria	MIPS R12000 300 MHz 20 GB/8 MB	IRIX 64 6.5	767	6	1	2.990
node V RIST SALZBURG Austria	ALPHA EV6 21264 500 MHz 512 MB/4 MB	OSF I V 5.0	341	6	1	6.823

system (glycine/631g, 10 centre, 55 basis functions) was performed, which led to the data shown in Table 2.

The timings were obtained with the simple UNIX-style **time a.out** command. According to the fact, that the PVM version of **GREMLIN** consists of a master-code and a node-code part, and since the node-code part got a different executable name, the mentioned **time**-command could easily distinguish between the parallel and the serial (diagonalization and pre-ERI work) fractions of the program execution. Thus to focus on the sections that really were running in parallel, one simply had to substract the usr+sys timings from the real one and could straightforwardly obtain the relative speed factors shown in Table 3. Note, that node I and III were lacking from special tuned **LAPACK** libraries, so their usr timings became significantly higher.

2.2 Estimation of Network Latency and Communication Time

To get a feeling for the time, that is lost through inter host communication — when nodes are receiving/sending data — we simply measured the bandwidth we got from the different host positions towards those node serving as the master machine in the WAN cluster later on (node III). For the real application of alanine/aug-cc-pVDZ (13 atoms, 213 basis functions) we had to expect a data transfer of the size of 1452 kB per iteration, which results in a net amount of 27.6

Table 3. Speed-Factor and Network-Latency table for the WAN cluster. Speed-Factors represent the relative performance of all the individual hosts in the WAN cluster with respect to the slowest performing CPU. Network bandwidth was obtained from measuring transfer rates between nodes and the future master-machine (node III).

Physical Location	Architecture/ Clock Speed/ RAM/2L-Cache	Operating System	Relative Speed Factor	Network Bandwidth [kB/s]	Exp.Total Comm. Time [s]
node I G.C. BOLOGNA Italy	INTEL Dual PPro 200 MHz 256 MB/512 KB	LINUX 2.2.14	1.000000	166	166
node II ICPS STRASBOURG France	MIPS R10000 200 MHz 20 GB/4 MB	IRIX 64 6.5	1.933617	608	45
node III GUP LINZ Austria	INTEL PII 350 MHz 128 MB/512 KB	LINUX 2.2.13	2.054250	—	—
node IV GUP LINZ Austria	MIPS R12000 300 MHz 20 GB/8 MB	IRIX 64 6.5	2.989474	918	30
node V RIST SALZBURG Austria	ALPHA EV6 21264 500 MHz 512 MB/4 MB	OSF I V 5.0	6.822822	592	47

MB for all the 19 iterations needed throughout the whole calculation. Network transfer rates and estimated total times spent on communication are also shown in Table 3.

3 Discussion

A final calculation of the above mentioned alanine/aug-cc-pVDZ (13 atoms, 213 basis functions) system on a successive increasing WAN cluster was performed and led to the execution timings and according Speed Up factors shown in Table 4. A similar, graphical representation of the Speed Up factors is shown in Fig. 1. Instead of strictly applying *Amdahl' s Law*, Speed Up $\leq \frac{1}{s+\frac{1-s}{N_{cpu}}}$, we tended to simply relate (real-usr) timings to each other, which was estimated to have almost no influence on relative values, and neglectable influence on absolute values.

Comparision of the final column of Table 3 to the 2nd column of Table 4 reveals a neglectable influence of communication time as well.

The 3rd column of Table 4 might be best suited to explain the actual heterogenity of the WAN cluster. In principle there should be one uniform amount of time spent on the diagonalization- and pre-ERI work, which basically is all what is reflected in the Usr Time. However, temporary network bottlenecks, OS-competion for CPU-time, temporary I/O management excess, CPU-time competition from interactive user operation — which all was allowed during program execution — led to that much more realistic, more variational picture.

The plot in Fig. 1 defines the number of machines in a cumulative way from left to the right on the abscissa, thus the always added new hosts are indicated at

Table 4. Execution timings and Speed Up factors for the DSCF Hartree Fock calculation of alanine/aug-cc-pVDZ with **GREMLIN** in a WAN cluster made of 1 to 5 nodes.

WAN Cluster Configuration		Real Time [s]	Usr Time [s]	Sys Time [s]	Theor. Speed Up $\sum$SPEED[I]	Real Speed Up
master	III					
nodes	I	240 061	9 268	3	1.000	1.000
master	III					
nodes	I,II	90 280	9 261	8	2.934	2.847
master	III					
nodes	I,II,III	60 496	9 368	2	4.988	4.516
master	III					
nodes	I,II,III IV	45 014	9 923	3	7.977	6.577
master	III					
nodes	I,II,III IV,V	27 038	9 482	6	14.800	13.116

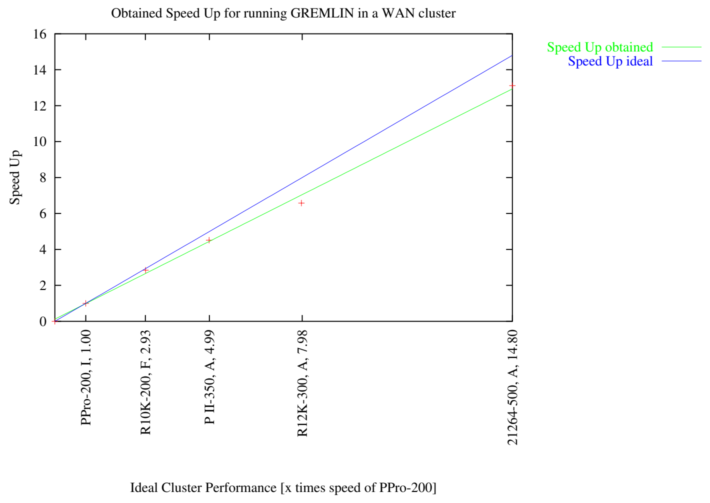


Fig. 1. Representation of the obtained and ideal Speed Up factors for the DSCF Hartree Fock calculation of alanine/aug-cc-pVDZ with **GREMLIN** in a ssh-connected WAN-cluster, made of up to 5 machines.

those final ideal speed level — relative to the slowest node — the cluster should ideally achieve at that very configuration.

3.1 Conclusion

Considering individual node performance in a heterogenous WAN cluster properly, may result in excellent parallel scalability for special dedicated applications, that are characterized from small communication time and large independent node intervals.

Acknowledgement
The author would like to thank Prof. Zinterhof from RIST^{++} Salzburg, Prof. Volkert from GUP Linz and Dr. Romaric David from ICPS Strasbourg for providing access to their supercomputer facilities.

References

1. Höfinger, S., Steinhauser, O., Zinterhof, P.: Performance Analysis and Derived Parallelization Strategy for a SCF Program at the Hartree Fock Level. Lect. Nt. Comp. Sc. **1557** (1999) 163–172
2. Schrödinger, E.: Quantisierung als Eigenwertproblem. Ann. d. Phys. **79**, **80**, **81** (1926)
3. Hartree, D.R.: Proc. Camb. Phil. Soc., **24** (1928) 89
4. Fock, V.: Näherungsmethoden zur Lösung des Quantenmechanischen Mehrkörperproblems. Z. Phys. **61** (1930) 126 **62** (1930) 795
5. Höfinger, S., Steinhauser, O., Zinterhof, P.: Performance Analysis, PVM and MPI Implementation of a DSCF Hartree Fock Program. J. Comp. Inf. Techn. **8** (1) (2000) 19–30
6. Obara, S., Saika, A.: Efficient recursive computation of molecular integrals over Cartesian Gaussian functions. J. Chem. Phys. **84** (7) (1986) 3963–3974
7. Davidson, E.R., Feller, D.: Basis Set Selection for Molecular Calculations. Chem. Rev., **86** (1986) 681–696
8. Shavitt, I.: The Gaussian Function in Calculations of Statistical Mechanics and Quantum Mechanics. Methods in Comp. Phys. ac. New York, **2** (1963) 1–44
9. Dunning Jr., T. H.: J. Chem. Phys. **90** (1989) 1007–1023
10. Woon, D. E., Dunning Jr., T. H.: J. Chem. Phys. **98** (1993) 1358–1371
11. Geist, G., Kohl, J., Manchel, R., Papadopoulos, P.: New Features of PVM 3.4 and Beyond. Hermes Publishing, Paris Sept. (1995) 1–10