Theor Chim Acta (1987) 72: 139–148

Theoretica Chimica Acta

A parallel architecture and programming language for quantum chemistry

James R. Savage*

Myrias Research Corporation 10328 81 Avenue, Edmonton, Alberta, Canada T6E 1X2

(Received November 24, revised and accepted December 30, 1986)

The large gains in computational capability which are required in the future by problems in computational quantum chemistry must come from advances in both parallel architectures and algorithms. The relation between algorithms and architecture is discussed, with examples of non-numerical algorithms for which future architectures should facilitate implementation. The use of time-space complexity trade-offs is discussed. A parallel language extension and architecture targeted towards general numerical and nonnumerical algorithms being developed by Myrias Research Corporation is briefly presented.

Key words: Parallel algorithms—Parallel architectures

1. Introduction

Many scientific researchers estimate that available computing power must increase by about 4 to 6 orders of magnitude, with an associated increase of primary memory size by about 3 orders of magnitude, in order to model systems of present practical interest [1, 2]. In order to perform a self-consistent-field (SCF) Hartree–Fock calculation on a biological molecule such as a protein with 10^3 atoms, using a minimal basis set, an increase in speed of at least 10^6 would be required (assuming the $O(n^4)$ growth in the number of integral evaluations [3] is truncated by the vanishing of integrals involving basis sets whose centres are more than 20 angstroms apart). To achieve results of a reliable nature, significantly larger increases in speed would be required. However, it is estimated that current

* *Present address*: Chion Corporation, Box 4942, Edmonton, Alberta, Canada T6E 5G8

supercomputers (pipelined vector machine architectures) are within an order of magnitude of their maximum achievable speed [4].

Judging from experiences with architectures which have similar synchronization barriers [5, 6], further increases of speed through multi-tasking on several vector machines is likely to give at most another factor of 10 increase in speed, at the expense of propagating hardware restrictions into the user programming model. In order to meet the future demands of scientific researchers, a scalable, programmable, parallel architecture is required. Scalable is taken to mean that there is no a priori limit on the configuration size (number of processing elements), and that performance varies almost linearly with size (provided the problem size is increased in proportion to the configuration size). The memory size must also scale with the configuration size. Programmable is taken to mean that the user has a simple mechanism for expressing the parallelism in his problem. The mechanism must be independent of the configuration size as well as the technology used in implementation. The mechanism must not require the user to be aware of the physical location of parallel tasks or their memory spaces. Further desirable design goals are discussed in [7].

Numerous parallel processors have been, or are being, constructed in an attempt to satisfy the projected needs of the scientific user. However, most parallel processing projects have involved the design of a hardware architecture, followed by attempts to design algorithms which will efficiently map a solution methodology for a given problem onto the hardware architecture. Besides propagating hardware restrictions into the user programming model, resulting in a non-programmable architecture [8], this approach neglects the other major factor involved in significantly decreasing the real time it takes to obtain a solution to a given problem, namely, the use of optimal, or near-optimal algorithms. For example, suppose the size of a given problem is measured by N (N may be the order of a matrix, or the number of basis elements, etc.). A machine which is 100 times faster, but must run an $O(N^2)$ algorithm instead of an $O(N * \log N)$ algorithm to achieve this speed-up, will take a longer time to solve the problem if $N > 1001$, assuming the coefficients are identical.

Algorithm development can have as large an impact on improved performance as new architecture designs. For this reason, it is important that the synergistic relation between algorithms and architectures be well understood, both by the architect and the user developing the algorithms.

In the next section, the relationship between algorithms and architecture is discussed. The last section presents an architectural approach designed to achieve a scalable, programmable architecture which would enable the implementation in a natural, intuitive manner, of algorithms discussed in Sect. 2. Historically, the computational complexity of performing SCF calculations on biological molecules were major influences behind the architectural (language) approach of Sect. 3, and the formation of Myrias Research Corporation to produce an implementation.

2. Algorithms and architectures

An architecture is usually designed to suit the perceived needs of a particular class of algorithms. Typical classes of algorithms include signal and image processing, matrix operations, and artificial intelligence. Architectures for signal processing have tended towards large numbers of small bit-slice processors connected in a regular pattern, such as special purpose fast Fourier transform devices and the Goodyear Massively Parallel Processor (MPP). Architectures for performing fast matrix operations tend to be pipelined vector processors such as those produced by Cray, CDC, Fujitsu, Hitachi, and Floating Point Systems. Artificial intelligence oriented architectures are still in their infancy.

Architectures targeted towards scientific computation are currently predominately pipelined vector processors. However, as will be seen, optimal algorithms for scientific computation are much more diverse than matrix operations, and in fact many are more combinatorial or symbol oriented than numerical. In fact, even some matrix operations, such as those involving sparse matrices, are not well suited to vector architectures [9].

On the other hand, the architecture available to users has a large impact on the algorithms they develop to solve their problems. In fact, the available architectures tend to channel the thought processes which occur in the search for solutions. The channeling process is undoubtedly deeply connected with the relationship between the human thought process and language. To see the connection, consider the following argument.

From a user's viewpoint, an architecture is basically a formal language and a performance model, where the performance model indicates which syntactic constructs in the language are efficient. For example, a vector construct is much more efficient on a pipelined vector processor than a scalar construct is. Furthermore, the efficiency of a vector construct varies with the vector length, with the exact variation depending on the pipeline length of a particular architecture [10]. The user of a pipelined vector processor even has the choice between assembler language constructs which implicitly contain detailed information about the hardware, and higher level constructs such as Fortran with additional vector constructs which save the user the detail at the cost of sacrificing some efficiency.

Linguists have long been aware of the channeling effect of natural language on the human thought process [11]. To some extent, the channeling effect can be observed through the process of translation, the expression of a concept which arose within one language culture in a second, distinct language culture. In a sense, the difficulty of the translation process for a concept is related to the probability that the concept would arise in the second language, without introduction from another language. The more difficult the translation, the less likely the concept would arise naturally. An extreme example is the transfer into English of Chinese philosophical concepts, which is regarded by some as "probably the most complex type of event yet produced in the evolution of the cosmos"[12].

The formal languages and performance models which form the user's model of various computer architectures have much more precision than occurs in natural

languages. However, the channeling process and difficulties in translation are readily evident. For example, a user working with a language (and architecture) which does not support recursive data structures or recursive procedures is not likely to think of recursive solutions to his problems. And even if he does, it may be too difficult to translate into his available architecture, or too inefficient if the translation is possible.

The importance of optimal or near-optimal algorithms should not be overlooked in the design process of a new architecture, as decreasing the real time for problem solutions by many orders of magnitude requires not only a much faster architecture, but an architecture which tends to channel users toward optimal algorithms. Although it is always dangerous to predict the future, it is instructive to investigate the general shape of algorithms and solution methodologies which may play an important role in the future of computational chemistry.

Artificial intelligence algorithms are finding roles in a number of areas. One example is the use of a problem-state language approach in determining an ordered set of chemical structure descriptions from mass spectra and other experimental data [13]. Other examples include the use of combinatorial and graph algorithms for calculating ligand binding [14], discriminating isomeric structures [15], performing similarity searches and structure-activity correlations [16], detecting near equivalence of major substructures in a molecule [17], and drug design [18].

Another interesting example is the use of tree pruning algorithms to select the most important vibrational-rotational states of a molecule for studying its multiphoton dynamics in a laser field [19]. The computational complexity is considerably reduced from the problem of diagonalizing a large matrix of order 7690 to a matrix of order 400. This was done by a careful state selection performed by a branch-and-bound-search, using as the cost of the path between two states a measure which depends on $|E_i - E_j| - h\nu$, where E_i is the energy of state i, and ν is the frequency of the laser.

The computational complexity of molecular dynamics calculations can be considerably reduced by using sorting and searching algorithms to construct and manipulate neighbour lists [20, 21, 22]. The neighbour lists enable the introduction of a number of different time steps, with only interactions between near neighbours being calculated for the smallest time steps and all possible interactions being calculated only for the largest time steps. This is a good example of the use of a time-space complexity trade-off. The decrease in time complexity (measured by the number of operations required to advance a given time interval) is bought at the expense of an increase in the space complexity (the storage, or memory required for the neighbour lists).

It is instructive to look at the efforts being made in other fields to find optimal or near-optimal algorithms for solving partial differential equations. Using methods to decrease the computational complexity of performing the matrix operations which arise is a useful direction, and considerable work has been expended in this area [23]. However, there are a number of techniques being

developed for the solution of partial differential equations which have an affect on the running order of programs at a more global level, dramatically decreasing the number of operations required by effectively decreasing the order of the matrices.

Let $Df = g$ be a differential equation, where D is a differential operator from one type of function space into another. For example, the function spaces are often L^p, Sobolev or Hilbert spaces (and thus infinite dimensional). There are two basic methods for discretizing the differential equation which we will consider. (Another method, asymptotic ray tracing [24], is used predominantly in seismic work). The first, the finite difference (FD) method, discretizes the time-space continuum into a time-space grid, turning the differential operator D into a difference operator. The second, the finite element (FE) method, discretizes the (infinite dimensional) function spaces by approximating them with a finite dimensional subspace [25]. Although FD methods have been used in quantum chemistry [26], the vast majority of *ab initio* methods, such as Hartree-Fock and configuration interaction, are FE methods [27].

The main idea behind attempts to find near-optimal algorithms for the FD and FE methods is to use as much knowledge of the physics as possible in the discretization process, thereby reducing the computational complexity. For example, in FD methods (and FE methods using meshes) the time-space grid can be made spatially non-uniform and dynamic [28–31]. Grid spacings in both the time and space directions are made small in regions where the physics is complicated, such as boundaries, transition regions, shock waves, etc. Moreover, the grid is dynamically adapted as these regions move.

A rough estimate of the change in computational complexity due to the use of such adaptive methods can be obtained as follows. Suppose the problem consists of modelling a 3-dimensional cube of length 1 and which contains a small number of 2-dimensional shock fronts. Let e be the smallest grid spacing required in order to advance the solution another time step with a given accuracy. Then, if $N = 1/e$, the number of operations required using a uniform grid is proportional to N^3, or the number of grid points. If an adaptive grid is used, however, the number of grid points will be $N^{(2+d)}$, where $0 < d < 1$ with d depending on the exact method and the physics. Note that adapting a grid has the flavour of a divide-and-conquer algorithm. Namely: for each point of a grid at a given level, if the physics in the region of that point can be modelled accurately enough with the present spacing, then nothing more need be done. Otherwise, the region of the grid point must be subdivided into a finer grid, whose level is one deeper [32].

In FE methods, the dominant method, except in quantum chemistry, for choosing the finite dimensional subspace is to subdivide the spatial region into a mesh and select a number of polynomials which have compact support on the different elements of the mesh as the basis set [25]. Introducing knowledge of the physics in order to reduce the computational complexity is then done as described above for the FD method. This method has a number of difficulties, such as introducing anisotropic effects into the propagation of waves, which could possibly be avoided

through the use of a more physical choice of basis sets. There is a good historical reason for the solution being channeled towards the mesh approach. By choosing polynomials which have compact support on the mesh elements, the integrals which arise in the FE method are often made less computationally complex. In cases where the integral evaluations are done by numerical quadrature, and are computationally expensive, a slightly modified approach has proven useful [33]. The desired integrals are expressed as linear combinations of basic integrals, which are reduced in number by group theoretic techniques. A computer system for algebraic manipulation is then used to compute expressions for the basic integrals, which are linear combinations of 20 to 26 terms, in this case. These are then programmed by hand.

In SCF calculations, the situation is slightly different. State selection [34], or the desirability of decreasing the basis size n in order to decrease both the number of integrals which must be evaluated ($O(n^4)$) and the matrix operations required (about $O(n^3)$) is still paramount. However, the atoms give positions for the centres of the basis sets (no arbitrary meshes).

3. The myrias PAR DO memory model

The architectural design methodology [8] used by Myrias consisted of doing a detailed analysis of target applications, including possible future algorithms, as above, as well as current algorithms in use. The parallel structure of the algorithms was studied, and a simple but expressive language construct was designed to enable the natural expression of the parallel structures. The language then drove the system architecture design, with the goal of an efficient, cost-effective implementation of a parallel processor which supports that language construct [7].

Any parallel programming language should meet a number of design criteria. The design criteria used by Myrias are the following:

The language must imply a simple, intuitive programming model which is close to the cultural expectations of present programmers, scientists, engineers and other users. There should be a simple physical model for the data flow implied by the language.

The programming model must be independent of the number of processors.

The language must be very expressive, allowing a natural expression of algorithms used in physical modelling, signal processing, combinatorial problems and other cycle-intensive computing tasks.

The conversion costs of present serial programs and algorithms should be minimized.

The parallel construct should be easily grafted onto present serial languages which are used for cycle-intensive problems, such as Fortran and C.

The language construct should allow an efficient implementation with a low performance spread.

An implementation of a high level failure recovery mechanism must be enabled by the language. Otherwise the large number of components in a large configuration would limit the mean time between failure too severely.

The result of the language design done by Myrias is the PAR DO construct and its associated memory semantics. Eliminating global memory semantics eliminates many problems, both in the programming model (synchronization semantics) and in the implementation architecture. The Myrias 4000 system provides the support required for the memory model, including automatic task scheduling and virtual memory management.

The mechanism for expressing the parallelism in a computational problem is done via a language extension, the parallel DO, or PAR DO. The PAR DO memory extension is compatible with many serial programming languages, and has presently been added to C and Fortran 77, producing Myrias Parallel C (MPC) and Myrias Parallel Fortran (MPF). MPF has two additional extensions to increase its power, namely, recursion and dynamic array allocation.

The following is a short description of the user-level model of the PAR DO construct. It is not meant to be a precise language definition, nor is any attempt made to demonstrate how the implementation avoids unnecessary work which might be implied by this model.

When the calculations within a DO loop are independent, the user can indicate that the calculations can be done in parallel by changing the DO keyword to PAR DO. This changes the memory semantics slightly. Each "iteration" now sees the machine state as it was at the beginning of the PAR DO instead of as it was at the end of the previous "iteration". Conceptually, this initial machine state is a parent to many child tasks or loop "iterations". The child tasks may be completely heterogeneous and, conceptually, are done in parallel. Of course, the amount of actual parallelism is restricted by the number of processors available. The mapping of parallel tasks onto available processors is performed automatically by the control firmware, using a sophisticated resource allocation mechanism.

At the end of a PAR DO, all child tasks are merged into one machine state using the following rules:

If no task assigns to a variable (or memory location), then the variable is unchanged.

If one task assigns to a variable, the variable is changed to the assigned value.

If more than one task assigns to a variable, but the values assigned are identical, then the variable is changed to the assigned value.

Otherwise, the value of the variable is undefined. If several tasks assign different values to a variable, there is no natural way to choose which value it should have after merging, and in fact, it might not have any of the assigned values.

Note that there is no communication between sibling tasks. Also, a variable within a PAR DO whose value is undefined at the end of the PAR DO has the behaviour of a local variable.

Intuitively, a PAR DO has the flavour of computing a next state from a previous state. For some, the PAR DO memory model is more natural and intuitive than the serial DO model. For example, consider the effect of the two expressions $A(I) = A(I+1)$ and $A(I) = A(I-1)$ when they occur inside a loop construct. If the construct is a PAR DO, the action is symmetric, causing a left (right) shift respectively. If the construct is a serial DO, the first causes a left shift while the second copies the value of the first element into all the other array positions.

PAR DO's can be combined with recursion. For example, a dot product of two vectors can be done by dividng the two vectors in half, recursing, and summing the resultant dot products. This reduces round-off errors since an operand is involved in only $O(\log n)$ additions instead of $O(n)$. Recursion is also the most convenient method for handling combinatorial problems. Parallel recursion enables a limited simulation of nondeterministic calculations to be performed.

There are no restrictions on the number of "iterations" in a PAR DO, nor on the depth of nesting of PAR DO's and recursive subroutines. There is no need to worry about parallel tasks having different amounts of work to perform or different memory requirements. Causal restrictions are handled without requiring any complicated synchronization semantics. The recursive parallel method (RPM) of programming made possible by MPF subsumes all vector, parallel, and tree-machine architectures. All divide and conquer algorithms are easily expressed using the RPM.

The PAR DO construct gives the user access to parallel processing in a form which is intuitive and easy to use. The author is not aware of any calculations in computational chemistry which cannot be programmed in a natural way with the PAR DO construct. The calculation of integrals is a good example of a large number of parallel, but heterogeneous tasks. A similar comment can be made regarding the creation of integral subroutines by the method discussed in Sect. 2. Some programming examples are given in [7], including game tree searches (related to tree pruning) and sorting.

The Myrias 4000 system has a multilayered architecture [32]. The layers are labeled as follows: hardware, control firmwave, operating software, and programming languages in which application codes are written.

The major technological innovations that made an implementation of MPF possible lie in the control firmware. The main goals for the control firmware were to efficiently and effectively harness the computing power of thousands of low-cost microprocessors, provide detection of and automatic recovery from all system failures, and to accept and run MPF programs

The M4000 control firmware is totally distributed to eliminate performance bottlenecks. Virtual memory management, process management, and resource management are all distributed via a kernel which resides in every processing

element of a configuration. Communication is done through messages and page transfers. The control firmware optimizer collects performance information and adjusts system tuning parameters.

Advantage is taken of the locality of reference which occurs in programs written for virtual memory machines. The hardware is organized as a hierarchy of clusters in a fractal interconnection scheme. Each cluster is composed of smaller, self-similar clusters. The smallest cluster is a single processing element. Pages are cached at different levels of the hardware hierarchy.

The basic processing element consists of a Motorola 68000 microprocessor, 512 Kbytes memory, and a high-speed interface to the board-level bus. Eight processing elements and a service processor are combined on one board. Sixteen boards, a printed circuit backplane (no wirewrap), and two communication boards are combined into a cage. Each communication board has 4 communication modules which are used to interconnect the cages. All buses and communication channels have more than 6 Mbytes/s nominal bandwidth.

The Myrias 4000 system is physically packaged in units of 1024 processors, called Krates. Each Krate has 512 Mbytes of memory and a usable memory bandwidth of 5000 Mbytes/s.

Since typical configurations would consist of 4 to 64 Krates, enough memory is available in the larger configurations for extremely large integral databases, enabling the use of time-space complexity tradeoffs.

References

1. Winkler KA, Norman ML, Norton JL (1986) On the characteristics of a numerical fluid dynamics simulator. In: Matsen FA, Tajima J (eds) Supercomputers: algorithms, architectures and scientific computation. University of Texas Press, Austin, pp 415-429
2. Buzbee BL, Sharp DL (1985) Science 227:591-597
3. Csizmadia IG (1981) Some fundamentals of computational theoretical chemistry. In: Csizmadia IG, Daudel R (eds) Computational theoretical organic chemistry. Reidel, Dordrecht Boston London, pp 1-14
4. Buzbee BL (1985) Applications of MIMD machines, In: Duff IS, Reid JK (eds) Vector and parallel processors in computational science. North-Holland, Amsterdam, pp 1-5
5. Axelrod TS (1986) Parallel Computing 3:129-140
6. Lubeck OM, Frederickson PO, Hiromoto RE, Moore JW (1985) Los Alamos experiences with the HEP computer. In: Kowalik, Janusz S (eds) MIMD computation: HEP supercomputer and applications. MIT Press, Cambridge, pp 331-339
7. Savage JR (to appear) Math Comput Simulation
8. Savage JR (1985) Parallel processing as a language design problem. In: Agerwala T, Freiman C (eds) Proc. 12th Annual International Symposium on Computer Architecture. IEEE, Boston Mass, pp 221-224
9. Duff IS, Reid JK (1982) Comput Phys Commun 26:293-302
10. Lubeck O, Moore J, Mendez R (1985) Computer 18:10-23
11. Steiner G (1975) After Babel: aspects of language and translation. Oxford University Press, Oxford
12. Richards IA (1953) Towards a theory of translating. In: Wright AF (ed) Studies in Chinese thought. University of Chicago Press, Chicago, pp 250-278

13. Lindsay RK, Buchanan BG, Feigenbaum EA, Lederberg J (1980) Applications of artificial intelligence for organic chemistry: the Dendral project. McGraw Hill, New York Toronto London
14. Kuhl FS, Crippen GM, Friesen DK (1984) J Comput Chem 5:24–34
15. Raychaudhury C, Ray SK, Ghosh JJ (1984) J Comput Chem 5:581–588
16. Balaban AT, Mekenyan O, Bonchev D (1985) J Comput Chem 6:538–551
17. Bersohn M, Fujiwara, S, Fujiwara Y (1986) J Comput Chem 7:129–139
18. Golender VE, Rozenblit AB (1983) Logical and combinaotrial algorithms for drug design. Research Studies Press, Letchworth, Hertfordshire, England
19. Chang J, Wyatt RE (1985) Chem Phys Lett 121:307–314
20. van Gunsteren WF, Berensden HJC, Colonna F, Perahia D, Hollenberg JP, Lellouch D (1984) J Comput Chem 5:272–279
21. Sullivan F, Mountain RD, O'Connel J (1985) J Comput Phys 61:138–153
22. Teleman O, Jonsson B (1986) J Comput Chem 7:58–66
23. Pan V (1984) How to multiply matrices faster. Springer, Berlin Heidelberg New York Tokyo
24. Chapman CH, Drummond R (1982) Bull Seismol Soc Am 72:277–317
25. Strang G, Fix GJ (1973) An analysis of the finite element method. Prentice Hall, London
26. Mullally DJ, McIver JW (1983) J Comput Chem 4:552–555
27. Carsky P, Urban, M (1980) Ab initio calculations. Springer, Berlin Heidelberg New York
28. Babuska, I, Chandra J, Flaherty JE (1983) Adaptive computational methods for partial differential equations. SIAM, Philadelphia
29. Smooke MD, Koszykowski ML (1986) SIAM J Sci Stat Comput 7:301–321
30. Glimm J, Marchesin D, McBryan O (1980) J Comput Phys 37:336–354
31. Glimm J, Marchesin D (1981) J Comput Phys 39:179–200
32. Kobos AM, VanKooten RE, Walker MA (1986) Powerful new programming model for parallel computation. In: Rosenblum A (ed) Proceedings of school on relativity, sypersymmetry, and strings. Plenum Press, New York London (to appear)
33. Anderson CM (1979) Eng Maths Appls 5:297–320
34. Shavitt I (1977) The method of configuration interaction. In: Schaefer III HF (ed) Methods of electronic structure theory. Plenum Press, New York London (and references therein)