密度汎関数法プログラム ProteinDF の並列化 -大規模クラスタでの分散並列処理を目指した改良-

吉廣保[†]、上野哲哉[†]、稲葉亨[‡]、柏木浩[†‡*]、佐藤文俊[†]
東京大学生産技術研究所[†]　アドバンスソフト株式会社[‡]　九州工業大学[*]

１．はじめに

生体内で重要な働きをするタンパク質は多くの原子からなる生体高分子であり、基本骨格を作る 20 種類のアミノ酸以外に、ヘムやクロロフィルに代表されるような色素分子が複雑に入り込んでそれぞれのタンパク質固有の機能を発現している。タンパク質の機能を電子レベルで解明するためには、タンパク質全体を分子として扱うことができ、かつ多くの色素に含まれる金属もアミノ酸残基と同程度の精度で求めることができる理論的解析手法であることが望ましい。

そのような観点から、発表者らはこれまで独自にガウス型基底関数を用いた密度汎関数法に基づく大規模分子軌道法プログラム ProteinDF を作成・発展させてきた[1]。密度汎関数法は Kohn-Sham 方程式に基づく分子軌道法の一種で、最大の利点は *ab initio* HF 法と同程度の計算量で、電子相関効果を取り込んだ計算ができることである。そのため、大型で金属錯体を含む系、例えば金属タンパク質などの計算に威力を発揮する。また、開発したプログラムは、計算時間が軌道数の約 2.4 乗で計算できるという良好な結果を得ている[2]。

２．オブジェクト指向技術による並列化と WS クラスタによる並列分散処理

タンパク質の分子軌道計算で問題となるのが、従来の分子軌道法プログラムと比べて、大規模で複雑なプログラムとなってしまうことである。また、プログラム実行時には数値計算過程であらわれる行列等の膨大なデータ量を取り扱わなくてはならない。例えば 10000 軌道の計算を行なうために倍精度浮動小数点型でメモリを確保したとすると、行列１つあたり約 800MB のメモリが必要となる。そのために、これを処理する有効なアルゴリズムを用いなければならず、また実行の制御や計算機資源の把握・管理などが必要となる。しかもプログラムを多人数で共同開発するため、一人一人のプログラマはプログラム全体を把握することは事実上、不可能となる。

これらの困難を克服するため、発表者らが開発した ProteinDF はオブジェクト指向言語 C++でコーディングされている[1]。オブジェクト指向を導入することでプログラム全体を階層構造化させることができ、アルゴリズム、計算、通信といった各機能が互いに独立した構造がとれるようになった(図 1)。この階層化によって、プログラムはシナリオ（アルゴリズム）部、(並列) 計算部、仮想マシン部、通信部に完全に階層化できた。仮想マシンオブジェクトは、並列処理の際に通信を行うための機能も併せ持った方が便利である。これは通信オブジェクトを内包することで実現した。これらの階層は、機能的に全て独立しているので、計算機アーキテクチャや

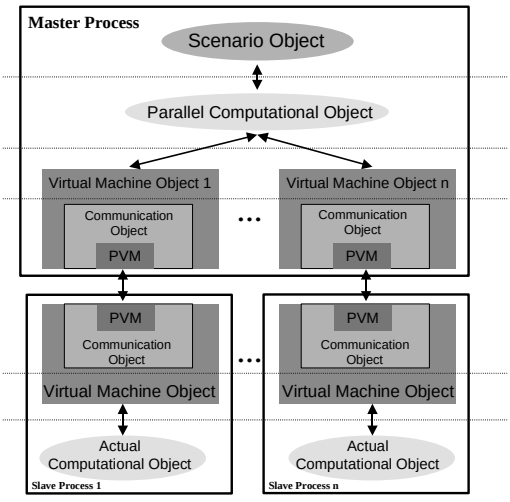

図１：階層化構造

The parallelization of density functional theory program ProteinDF
- Development of distributed parallel processing for large-scale cluster systems -
† Tamotsu Yoshihiro, Tetsuya Ueno, Hiroshi Kashiwagi, Fumitoshi Sato: Institute of Industrial Science, University of Tokyo
‡ Toru Inaba, Hiroshi Kashiwagi: Advance Soft
* Hiroshi Kashiwagi: Kyushu Institute of Technology

通信用ライブラリの多様性に容易に対応できる。これにより、各計算ルーチンが十分に独立し他の計算ルーチンのデータ構造などを考えることなく個別に修正・改良を加えることができ、プログラムの移植性、拡張性が高くなり、様々な並列計算機への対応や最新の技術の取り入れも容易になった。

これまで行ってきた計算結果から、タンパク質の全電子計算への適用を考えると、無理のない時間で計算するためには 10～100GFLOPS 程度の計算能力が必要となることが予想されていた。この計算能力を達成できるシステムは何種類か考えられるが、最近の高性能ワークステーション(WS)の値下がりやネットワークの高速化などから、実現性の高い WS クラスタという分散並列処理システムへの適用を考えてプログラムの並列化を行なった。

３．馬心臓シトクロム *c* の計算

本研究で開発した並列版 ProteinDF を用いて、実際に DEC ALPHA 21264 667MHz WS と DEC ALPHA 21164A 533MHz WS 8 台、500MHz WS 6 台の計 15 台を 100Base-TX の Ethernet で接続した WS クラスタ並列システムを使用し、馬心臓シトクロム *c*（104 残基+ヘム、1738 原子、9600 軌道、17578 補助基底関数）の全電子計算に成功した。59 回の繰り返し計算を要し、繰り返し計算の 1 回にかかる実時間は 77362 秒(21.5 時間)であった[2]。

４．並列計算結果

開発したプログラムが実際にどの程度の効率で並列化されているかを調べるために、シトクロム *c* の 51～81 残基を抜き出した 31 残基ペプチド鎖断片（2867 軌道、5273 補助基底関数）の計算を使用して検証した。

表 1 は、Alpha 21164A 533MHz WS 8 台を 1,4,8 台(表中では system(a), (b), (c))使用してペプチド鎖の計算を行ない、各計算ルーチンと計算全体において最初の SCF 計算にかかる時間を比較し、1 台で計算した時間を基に使用台数による実行速度の倍率と、そのときの並列化の効率を求めたものである。

31 残基ペプチド鎖を 1 台で計算した場合、最初の SCF 計算にかかった時間は 52615 秒であった。4 台使用した場合 14452 秒、8 台使用した場合は 8419 秒となった。実行速度はそれぞれ 3.64 倍、6.25 倍となり、並列化効率はそれぞれ 91.0%、78.1%であった[3]。

31 残基ペプチド鎖計算で得られた詳細なタイミングデータをもとに各ルーチンの並列化の実行性能について検討し、今後より大きな WS クラスタで並列計算を実行するための改良を行った。

発表では、31 残基の計算によって得られる、改良前、改良後の詳細なタイミングデータについて議論し、シトクロム *c* の全電子計算における並列化について報告する。また、さらに大規模なタンパク質の全電子計算に向けて、現在 ProteinDF の地球シミュレータへの移植を行っている。この状況についても報告する予定である。

謝辞

本研究は文部科学省リサーチ・レボリューション計画(RR2002)、IT プログラム「戦略的基盤ソフトウェアの開発」の支援の下に行われた。さらに、地球シミュレータセンターに感謝します。

参考文献

[1] F. Sato, Y. Shigemitsu, I. Okazaki, S. Yahiro, M. Fukue, S. Kozuru, H. Kashiwagi, *Int. J. Quantum Chem.* **63**, 245 (1997).
[2] F. Sato, T. Yoshihiro, M. Era, H. Kashiwagi, *Chem. Phys. Lett.* **341**, 645 (2001).
[3] T. Yoshihiro, F. Sato, H. Kashiwagi, *Chem. Phys. Lett.* **346**, 313 (2001).

表1: 31残基ペプチド鎖の計算で各ルーチンの1SCFにかかる時間と1SCF全体に占める割合と並列化効率

System / Calculation Step	system (a) : 1 processor	system (b) : 4 processors			system (c) : 8 processors		
	elapse time (sec)	elapse time (sec)	speed-up ratio[a]	parallelization efficiency (%)[b]	elapse time (sec)	speed-up ratio[a]	parallelization efficiency (%)[b]
Total Energy Calculation	16,961	4,311	3.93	98.4	2,348	7.22	90.3
Generation of Kohn-Sham Matrix	17,116	4,314	3.97	99.2	2,190	7.82	97.7
Fitting of Electron Density	10,279	3,360	3.06	76.5	1,833	5.61	70.1
Fitting of Exchange Correlation Potential	3,622	1,089	3.33	83.1	548	6.61	82.6
Matrix Diagonalization	2,246	599	3.75	93.7	763	2.94	36.8
Matrix Multiplication	2,391	779	3.07	76.7	737	3.24	40.6
Total	52,615	14,452	3.64	91.0	8,419	6.25	78.1

a: speed-up ratio = elapse time with single processor / elapse time with parallel system
b: parallelization efficiency = (speed-up ratio / number of processors) × 100