社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2007－HPC－111 (14)
2007／8／1

分子軌道計算におけるヘテロクラスタ上での効果的な負荷分散方式

早川 潔* 佐々木 徹** 梅田 宏明***, † 長嶋 雲兵***, †

本稿では，ヘテロクラスタにおける計算資源を有効に活用する分子軌道計算の負荷分散方式について評価・検討する．近年，CPUの急速な進歩とともに，多種多様なPCが登場しており，それら多種多様なPCを使用したヘテロジニアスクラスタが注目されている．また，今後，計算資源がCPUだけではなく，GPUであったり，FPGAを利用したハードウェアエンジンであったりと，さまざまバリエーションのクラスタが登場すると予想される．そのようなヘテロクラスタ環境において，計算プログラムを効率的に動作させる負荷分散法を検討するため，２種類の負荷分散方式（静的負荷分散および動的負荷分散）を評価した．これらの評価結果をもとに，効果的な負荷分散方式を検討する．

Efficacious Load Balancing Methods for Molecular Orbital Calculations on a Heterogeneous Cluster

Kiyoshi Hayakawa, *Tohru Sasaki***, *Hiroaki Umeda****, †, and *Umpei Nagashima****, †

This paper makes a study of load balance methods of molecular orbital calculations which make efficient use of Heterogeneous cluster systems. Recently, different PCs are launched from pillar to post as rapid progress of CPU. In the future, cluster systems which have different types of compute resource, such as GPU, FPGA and so on, may be launched. Under the circumstances, we evaluate 2 types of load balancing methods(static load balancing and dynamic load balancing) of molecular orbital calculations, and discuss about load balance methods which make efficient use of Heterogeneous cluster systems.

1 はじめに

ハイパフォーマンスアプリケーションの１つとして，分子軌道計算が知られている．分子軌道計算法の１つであるHF（Hartree-Fock法）法では，処理全体の約99%がFock行列生成に費やされており，より効率のよいFock行列生成が必要となる．EHPCプロジェクト[1]では，分子軌道計算などのハイパフォーマンスアプリケーションを効率よく処理するために，コンピュータシステムのプラットフォームを提案し，そのプラットフォーム上で効率のよい並列Fock行列生成アルゴリズムも開発されている[2]．

一方，PCの処理速度が急速に上昇し，数年前のPCが陳腐化してしまうため，性能の異なるノードを集めたクラスタが多くなると予想される．また，近年，GPUを汎用CPUの処理に使用したり，アプリケーションをハードウェア化するなど，計算資源が多種多様になっている．このような計算環境に対応すべく，本研究室では，ヘテロジニアス＆コンパクトクラスタのテストベッドとして，EMDCを開発している[3]．EMDCは，異なった性能を持つCPUで構成されており，その環境を利用して，アプリケーションのヘテロジニアスクラスタでの挙動を観測できる．

そこで，本稿では，分子軌道計算プログラムをEMDCで評価する．静的負荷分散法および動的負荷分散法による分子軌道計算をそれぞれ実行し，ヘテロジニアスクラスタでの性能を評価する．また，この性能評価をもとに，より効率よい負荷分散方式を検討する．

*:大阪府立工業高等専門学校 Osaka Prefectural College of Technology
**: (株) アプリオリ・マイクロシステムズ A-Priori Microsystems Ltd
***:産業技術総合研究所 National Institute of Advanced Industrial Science and Technology
†:CREST 科学技術振興機構 CREST Japan Science and Technology Agency

2 分子軌道計算プログラム

EMDC システムで実行する分子軌道計算プログラムは，EHPC プロジェクトで作成されたプログラムを EMDC 用に移植したものである．通信部分は，EHPC とは若干異なる部分があるので，本稿では，通信部分の実装を中心に述べる．

2.1 Fock 行列

分子軌道計算の処理時間の 99%を Fock 行列生成に費やす．従って，この Fock 行列生成処理を並列に処理する．分子軌道計算法の 1 つである Hartree-Fock 法 [4] において，Fock 行列は (1) 式で与えられる．

$$F_{ij} = H_{ij}^{core} + \sum_{k,l}^{N} D_{kl}\{2(ij|kl) - (il|kj)\} \quad (1)$$

ここで，i,j,k,l は縮約シェルのインデックス，D, H^{core} はそれぞれ密度行列および核-電子ハミルトン行列である．また，$(ij|kl)$ は 4 中心の 2 電子積分である．この 2 電子積分は N^4 個の 2 電子積分を計算する必要がある．N は分子によっては数万程度になるため，ほとんどの時間をこの 2 電子積分に費やす．従って，Fock 行列生成の処理（負荷）を分散し，処理時間の短縮を計る．

2.2 処理分担

EMDC では，Fock 行列生成処理を主に 5 種類の処理 [5]（Appli_Control, Compute, Bridge, H_comm, L_comm) に分け，レベル毎に割り振る（図 1 参照）．EMDC の各ノードには，レベルが設定されており，ホストノードがレベル 0，ノード番号 $3n+2$(n は自然数）を持つノードがレベル 1 のノード，$3n+1$ および $3n+3$ をレベル 2 のノードとしている．レベル 0 には，分子軌道計算処理の全体的な制御を行う Appli_Control およびレベル 1 と通信するための H_comm を割り振る．レベル 2 には，2 電子積分計算を行う Compute およびレベル 1 と通信するための L_comm を割り振る．レベル 1 にはレベル 0 と通信するための H_comm，レベル 2 と通信するための L_comm，およびレベル 0 とレベル 2 との通信を中継する Bridge を割り振る．また，レベル 1 にも compute を実行させる場合がある．

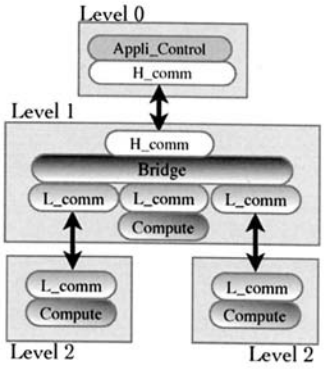

図 1: Fock 行列生成における各ノードの処理分担

2.3 分子軌道計算における通信手順

EMDC で実行した分子軌道計算は，静的負荷分散および動的負荷負荷分散の 2 つの負荷分散方式 [2] である．HF 法では，Fock 行列生成時に，SCF（Self-Consistent Field:自己無撞着場）計算を繰り返す．静的負荷分散では，SCF 計算毎に，Appli_Contorl があらかじめ計算タスクの計算量を決め，全 Compute に一斉に同じ量の計算を実行させる．一方，動的負荷分散では，Appli_Contorl 処理部が計算タスクを細かく分割し，空いている計算ノードに割り当て，計算を実行させる．

静的負荷分散における Fock 行列生成時の通信手順を以下に示す．

1. Appli_Contorl が，あらかじめ，計算タスクの情報を Compute に送る．
2. Appli_Contorl が，SCF 計算ルーチンを開始させるコマンドを Compute へ送る．
3. Compute が SCF 計算ルーチンを実行する．計算が終了したら，全 Compute が参加するバリア同期を行う．Compute は SCF ルーチン終了後，バリア命令を発行する．
4. Appli_Contorl が，Compute のバリア命令発行を確認した後，計算結果を集め，次の SCF 計算を行う．

動的負荷分散における Fock 行列生成時の通信手順を以下に示す（図 2 参照）．

1. Appli_Contorl が動的負荷分散の計算開始コマンド (dlb init) を Bridge を介して Compute に送信する．

2. Compute の計算準備ができたら，Compute が Bridge に計算準備できたことを通知（calc ok）する．

3. Appli_Contorl が Bridge に対して，計算していない Compute が有るかどうか問い合わせる (wait any)．

4. Bridge が calc ok を通知した Compute を調査し，結果を Appli_Contorl に返す (wait result)．

5. Appli_Contorl は Bridge から送られてきた調査結果から，計算していない Compute がない場合には，wait any を繰り返す．計算していない Compute がある場合には，Bridge に 2 電子積分計算を実行させるコマンド (cmd send)，およびそのタスク情報 (task data) を送る．

6. cmd send を受け取った Bridge は計算していない Compute のノード番号を調べ (get pid)，そのノードへ Appli_Contorl から受け取った cmd send および task data を送る．

7. cmd send および task data を受け取った Compute は，その情報をもとに計算を行い，終了したら，再度，calc ok を送る．

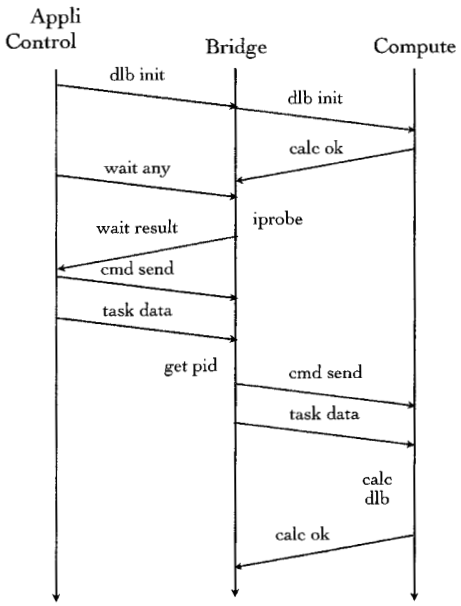

図 2: 動的負荷分散における通信手順

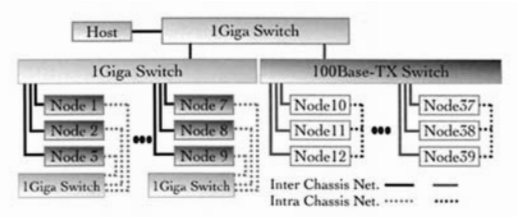

図 3: EMDC システムの構成

3 性能評価

分子軌道計算の性能評価として，C 端末を OH キャップしたグリシンの 5 量体 $(Glycine)_5$ の HF/6-31G(d,p) (ガウス型基底の数は 400) を計算した．文献 [2] に示されているスクリーニングを行い，動的負荷分散で計算した場合と静的負荷分散で計算した場合を比較した．分子軌道プログラムには GAMESS[7] を Appli_Control 処理部として使用した．

3.1 ヘテロジニアス実行環境テストベット EMDC

分子軌道計算に用いた EMDC のシステムを図 3 に示す．本システムは，ヘテロジニアスな実行環境を提供し，その環境下で効率よいアプリケーションを開発するためのテストベットとして開発中である．種々の計算資源をクラスタ化し，その上で効率よいアプリケーションを開発することにより，陳腐化した計算資源でも，それらをクラスタ化し，高速に処理できるシステムを目指す．本システムでは，PentiumM(2.0GHz) 搭載のノードが 9 台，PentiumIII 搭載のノード 30 台，およびホストコンピュータで構成されている．PentiumIII 搭載のノードでは，3 ノードのうち 1 ノードが動作周波数 600MHz の CPU で残りの 2 ノードが 700MHz の CPU である．PentiumIII や PentiumM などの比較的低動作周波数ではあるが低消費電力である CPU を利用して，コンパクトなクラスタも目指している．PentiumM ノードには 1Gbyte(デュアルチャネル)，PentiumIII には 256Mbyte のメモリが搭載されている．HDD は，コンパクトフラッシュメモリ（PentiumM・PentiumIII ノードともに 1Gbyte）を採用している．このことにより，機械稼働部品が減り，より長期運用・低消費電力なシステムになると考えられる．

ネットワーク構成は，Intra-Cahssis Network および Inter-Chassis Network で構成されている．PentiumIII ノードの Intra-Chassis Network は，SCC ボー

ド [6] と呼ばれる低レイテンシ通信用ボードを使用して構築し，PentimuM ノードの Intra-Chassis Network は，Gigabit Ethernet で構築する．ただし，本稿の評価では，SCC の代わりに，Ethernet を使用している．PentiumIII ノードの Inter-Chassis Network は，100Base-TX の Ethernet で構築し，PentiumM ノードの Inter-Chassis Network は，Gigabit Ethernet で構築する．

本評価では，36 台のノード（PentiumIII ノード 30 台，PentiumM ノード 6 台）を使用して行った．各ノードには，LinuxOS(version 2.6) が搭載されている．

3.2 評価方法

本評価では，計算資源を効率的に利用しているかどうかを示す指標として，並列化効率を使用する．ただし，２種類の計算資源を使用しているので，本稿では，PentiumM の性能を PentiumIII の値に換算し，並列化効率を算出する．

3.3 PentiumM ノードの性能評価

EMDC システムを PentiumIII ノードと PentiumM ノードに分け，それぞれで分子軌道計算を実行させた．実行結果を図 4 に示す．図 4 において，「static」は静的負荷分散における分子軌道計算全体の実行時間であり，「dynamic」は動的負荷分散における分子軌道計算全体の実行時間である．

PentiumM ノードの処理速度は，PentiumIII に比べて，静的負荷分散では 3.05 倍，動的負荷分散では 2.83 倍高速であった．したがって，この数値を用いて PenitumM の性能を PentiumIII の台数に換算し，並列化効率を算出する．

3.4 EMDC システムでの評価

EMDC システムでの性能評価として，Compute をレベル 1 で実行しない場合と実行する場合の 2 パターンで並列化効率を測定した．

3.4.1 レベル 1 で Compute を実行しない場合

レベル 1 で Compute を実行しない場合の並列化効率を図 6 および図 5 に示す．図中，Fock は分子軌道計算の Fock 行列生成のみの処理時間から算出した並列化効率であり，Total は，分子軌道計算全体の処理時間から算出した並列化効率である．

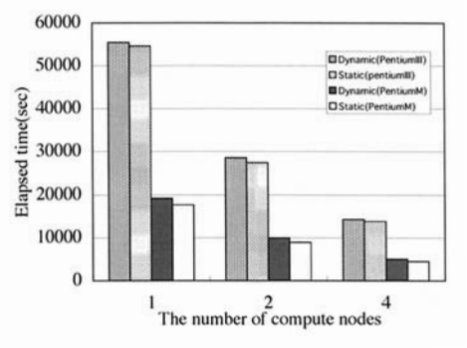

図 4: PentiumIII および PentiumM クラスタにおける α-$helix(Glycine)_5$ の実行時間

静的負荷分散および動的負荷分散において，Compute ２台から２０台までの並列化効率は，ほぼ直線的に減少していった．

PentiumM の Compute が加わると，負荷分散方式によって異なった結果となった．静的負荷分散における Fock の場合，並列化効率が 26.1 台で 0.79，32.2 台で 0.70 と急激に低下した．その原因は，負荷分散のアンバランスにある．静的負荷分散で効率よく実行するには，計算タスクの計算量を正確に見積もって各 Compute に割り当てなければならない．しかし，PentiumM ノードには PentiumIII とほぼ同じ量の計算タスクを割り振った．よって，各 Compute への負荷にアンバランスが生じ，バリア同期で待つ Compute が多くなり，並列化効率が悪化した．

動的負荷分散における Fock の場合，静的負荷分散の場合に比べ減少率は低くなったが，２台から２０台のときと比べると，減少率は高くなった．この減少率上昇の原因は，通信オーバーヘッドによる影響だと推測される．PentiumM は，PentiumIII より多くのタスクを処理するため，PentiumM への通信量が増加し，PentiumIII ２台追加の並列化効率の減少率より高くなってしまったと推測される

3.4.2 レベル 1 で Compute を実行した場合

レベル 1 でも Compute を実行した場合の並列化効率を図 7 および図 8 に示す．

ノード数 3 台での静的負荷分散の並列化効率は，急激に低下したが，４台目以降，ノード 3 台での並列化効率をほぼ同じ値を示した．この原因は，レベ

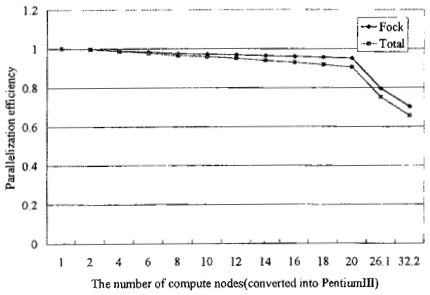

図 5: α-$helix(Glycine)_5$ の並列化効率（静的負荷分散

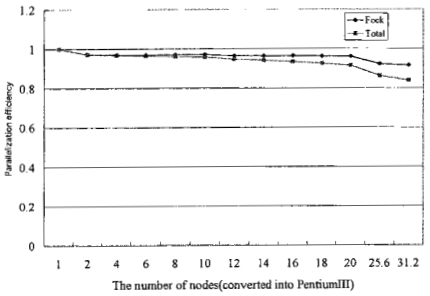

図 6: α-$helix(Glycine)_5$ の並列化効率（動的負荷分散)

ル１のノードで Bridge と Compute とを両方実行していたため，レベル１ノードの Compute 処理に時間かかり，レベル２のノードを待たせてしまったためである．動的負荷分散の場合でも，３台目で急激に低下しているが，静的負荷分散ほどではなかった．動的負荷分散の場合は，計算タスクを細かく分割し，Compute に割り当てているので，静的負荷分散ほど計算タスク処理時間のアンバランスはない．しかし，レベル１の Bridge 処理の通信量が多く，その処理を Compute とともに実行するため，通信時間がかかり，並列化効率の低下を招いた．

動的負荷分散において，レベル１で Compute を走らせない場合に比べて，３台から２４台までの並列化効率の減少率は高かった．この主な原因は，レベル１の Bridge 処理の通信処理が台数が増えるに従って増加するので，その分減少率に影響した．しかし，２４台と２７台で，並列化効率が微増した．

3.5 効率の良い負荷分散方式の検討

静的負荷分散の改良として，プロファイル方式とフィードバック方式が挙げられる．プロファイル方式とは，CPU の性能を前もって計っておき，その性能に応じて，計算量を決めて，分散させる方法である．フィードバック方式は，１回の SCF 計算の処理時間から，ノードの性能を予測し，それに応じて，計算量を決める方式である．

動的負荷分散の改良として，Bridge が計算タスクを動的に割り当てる方法が考えられる．Bridge にある程度大きなタスクをあらかじめ与え，それを Bridge が，細かく分割し，Compute の空き状況をチェックして，タスクを割り振る．本システムでは，別ネットワークで筐体内通信ができるので，筐体数が増えても並列化効率にそれほど影響を与えないと思われる．

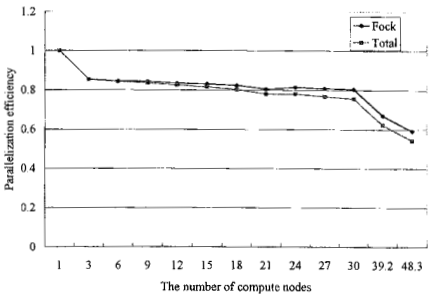

図 7: α-$helix(Glycine)_5$ の並列化効率（静的負荷分散

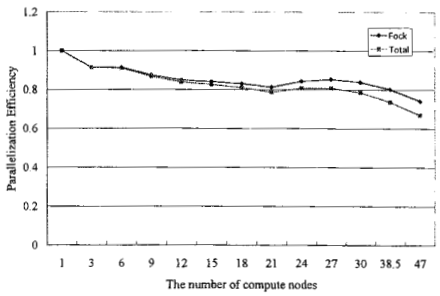

図 8: α-$helix(Glycine)_5$ の並列化効率（動的負荷分散)

4 おわりに

ヘテロジニアスクラスタ環境で電子軌道計算における負荷分散方式を2種類，さらに，レベル1ノードでComputeを実行させるかさせないかで2種類，合計4種類の評価した．静的負荷分散では，計算タスクの負荷がアンバランスだったため，PentiumMノードを有効利用できなかった．一方，動的負荷分散では，そのようなことはなく，静的負荷分散に比べPentiumMを有効利用できた．レベル1ノードでComputeを実行させない場合，動的負荷分散（Fock生成&PentiumIII換算31.2台）の並列化効率は0.91，静的負荷分散（Fock生成&PentiumIII換算32.2台）の並列化効率は0.70となった．レベル1ノードでComputeを実行させた場合，動的負荷分散（Fock生成&PentiumIII換算46.98台）の並列化効率は0.74，静的負荷分散（Fock生成&PentiumIII換算32.2台）の並列化効率は0.59となった．

これらの評価結果より，より効率的な負荷分散方式を検討した．静的負荷分散では，計算タスク量のアンバランスを改善する方法，動的負荷分散では，スケジューリングを担当するノードを増やして筐体内で通信して，通信による計算タスクの影響を少なくする方法を検討した．

今後の課題としては，分子軌道計算の改善・性能評価およびそれらをもとに他のハイパフォーマンスアプリケーションでの開発スタイルを考える予定である．

本稿で検討した負荷分散方式の改良をアプリケーションに施して性能評価したいと考えている．また，レベルを調節して並列化効率の変動を評価したいと考えている．さらに，レベル1を全てPentiumMノードにし，レベル2ノードは全てPentiumIIIノードにし評価したいと考えている．レベル1ノードでBridgeおよびComputeを実行する場合，計算パワーがあるPentiumMにしたほうが，並列化効率も若干よいのではないかと推測される．

他のハイパフォーマンスアプリケーションの開発スタイルとして，研究室内にあるコンパクトクラスタを使って，ソフトウェアでラピッドプロトタイピング型の開発をして，その後，そのソフトをハード化し，より高速に実行することを考えている．

この研究の一部は，平成18年度科学研究補助金（基盤研究C:課題番号18500044）「低消費電力コンパクトクラスタの研究」によって行われた．

参考文献

[1] 佐々木 徹，村上 和彰 :科学計算専用計算機のプラットフォームシステム，日本コンピュータ化学会，Vol. 4 No. 4,pp. 139–145(2005).

[2] 梅田 宏明，稲富 雄一，本田 宏明，長嶋 雲兵：分子軌道計算専用計算機のためのフォック行列並列計算アルゴリズムの開発，日本コンピュータ化学会，Vol. 4 No. 4 ,pp. 179–187(2005).

[3] Kiyoshi Hayakawa, Thoru Sasaki, Hiroaki Umeda, and Umpei Nagashima:A Communication Method for Molecular Orbital Calculations on a Compact Embedded Cluster, Asian Simulation Conference, pp. 357–361(2006)

[4] Helgaker.T., Jorgensen.P., Olsen.J., : Molecular Electronic-Structure Theory, Wiley (2000)

[5] 早川 潔，佐々木 徹，梅田 宏明，長嶋 雲兵，“コンパクトクラスタを利用した分子軌道計算の性能評価”, 情報処理学会研究報告, 2007-HPC-109, pp. 1-6(2007).

[6] Hayakawa, K., Sekigchi, S. :Design and Implementation of a Synchronization and Communication Controller for Cluster Computing Systems. Proc. 4th Intel. Conf. High Performance Computing in Asia-Pacific Region Vol.I, pp. 76–81(2000).

[7] Schmidt, M., Baldridge, K., Boatz, J., Elbert, S., Gordon, M., Jensen, J., Koseki, S., Matsunaga, N., Nguyen, A., Su, S., Windus, T., Dupuis, M., Montgomery, J. :General atomic and molecular electronic structure system. Journal of Computational Chemistry Vol. 14, Issue 11, pp 1347–1363(1993).

[8] 林 徹生, 本田 宏明, 稲富 雄一, 井上 弘士, 村上 和彰, “Cell プロセッサへの分子軌道法プログラムの実装と評価”, 情報処理学会研究報告, 2006-HPC-103, pp. 103-108(2006).

[9] 中島 浩, 中村 宏, 佐藤 三久, 朴 泰祐, 松岡 聡, 高橋 大介, 堀田 義彦, ”高性能計算のための低電力・高密度クラスタ MegaProto”, 情報処理学会論文誌コンピューティングシステム, Vol.46, No. SIG12 (ACS 11), pp. 46–61(2005).