

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

梅田 宏明^{a,b*}, 稲富 雄一^{a,b}, 本田 宏明^c, 長嶋 雲兵^{a,b}

^a 科学技術振興機構 CREST, 〒 332-0012 埼玉県川口市本町 4-1-8

^b 産業技術総合研究所 グリッド 研究センター, 〒 305-8568 茨城県つくば市梅園 1-1-1 中央第二

^c 九州大学 情報基盤センター 研究部門, 〒 812-8581 福岡県福岡市東区箱崎 6-10-1

*e-mail: h-umeda@aist.go.jp

(Received: July 4, 2005; Accepted for publication: August 23, 2005; Published on Web: November 18, 2005)

分子軌道計算専用計算機を効果的に稼働させるための並列フォック行列生成アルゴリズムを開発し、プラットフォームシステム上に実装した。階層構造を持つプラットフォームシステム上ではホスト PC とボード上の専用 LSI 間の通信には非常に大きなコストが必要となるが、多段の動的負荷分散を実装することで通信を局在化しつつ効果的な負荷の分散を実現した。SH4 CPU を 63 個使ったプラットフォーム評価システム上でのベンチマーク計算では 99% を超える非常に高い並列性能を得ることができた。

キーワード: EHPC, 分子軌道計算専用計算機, 多段動的負荷分散, 並列フォック行列生成アルゴリズム, 階層構造を持つプラットフォーム・アーキテクチャ

1 はじめに

非経験的分子軌道計算は分子の物性等の解析に欠かれない計算手法として広く使われてきている。一方で機能性材料の設計や医薬品開発などのように巨大な系を対象とする分子軌道計算には非常に大きな計算機資源が必要となり、大学の研究室などで容易に利用できる安価な PC クラスシステムではこのような大規模計算は大変困難であった。

研究室内外の既存の計算機を有効に活用することのできるグリッド技術は、この問題に対する一つの解決策である。グリッド技術を用いることにより、ネットワーク上にある多くの計算機資源を効果的に利用した、大規模科学技術計算が可能になると考えられる。

計算用途を限定した専用計算機の利用も、大規模計算を実現するための解決策の一つである。分子軌道計算法の一つである HF (Hartree-Fock) 法 [1] ではフォック行列の生成に全計算時間の 9 割以上の時間を費やすことが知られている [2]。フォック行列の生成には分子

サイズの 4 乗程度の計算が必要であるため、対象とする系が巨大になればなるほどフォック行列生成の高速化が重要であり、この部分を専用計算機で高速に実行することができれば全体の計算時間を大幅に短縮することが可能になる。

EHPC (Embedded High Performance Computing) プロジェクト [3] では、フォック行列生成において重要となる二電子積分計算専用の専用ロジック LSI を開発し、プラットフォームシステム [4] 上に多数搭載することによりフォック行列を高速に生成するための分子軌道計算専用計算機システムを構築した。このプラットフォームシステムでは、一般の PC に利用される CPU に比べ低消費電力の専用 LSI を高並列で動作させることにより性能を出すアーキテクチャを採用しているため、アプリケーションプログラムの並列化効率を上げることが本質的な意味を持っている。

一般のユーザがこのシステムを手軽に利用するためには、広く利用されている分子軌道計算アプリケーションをプラットフォームシステム上で動かす必要が

ある。しかしながら、プラットフォームシステムではプロセッサ (PU) 間の通信に強い制約が存在するため、既存の並列フォック行列生成アルゴリズムを用いた並列化では十分な効率が期待できない。

そこで本研究では、プラットフォームシステム上の多数のプロセッサを効率良く利用し、高並列計算を可能とするフォック行列生成アルゴリズムを開発するとともに、それを代表的な非経験的分子軌道計算パッケージの一つである GAMESS[5] に組み込んだ。また専用 LSI の代わりに汎用のプロセッサを利用したプラットフォーム評価システム上でベンチマーク計算を行ない、並列性能を評価した。

2 プラットフォーム評価システム

2.1 ハードウェア

本研究で利用したプラットフォーム評価システム (Figure 1) は専用 LSI の代用として汎用 CPU を搭載した、3 ユニットからなるシステムである。各ユニットは Compact PCI バックプレーンを持ち、PC 互換 Compact PCI ボード一枚と汎用 CPU を搭載した EHPC ボード (Figure 1 左のボード) 7 枚が接続されている [4]。

Figure 2 にプラットフォームシステムの概念図を示す。各ユニットを統轄する Host PC (PC 互換ボード上の PC, Figure 2 の緑色で示した PU [PC-#]) には Pen-

tium3 ボード (Pentium3 400MHz CPU, 256MB メモリ, 20GB HDD) と Pentium2 ボード (Pentium2 266MHz, 64MB メモリ, 20GB HDD) の 2 種類があり、今回利用したシステムは Pentium3 ボードを搭載したユニット 2 台と Pentium2 ボードのユニット 1 台の 3 ユニット構成となっている。これらの Host PC には Linux OS (kernel 2.4) が搭載されており、通常の Linux マシンとして利用できる。また Host PC では 100Base-TX の NIC が利用でき、LAN を通じて TCP/IP 等による通信が可能である。

各ユニットの Compact PCI スロットに挿入された EHPC ボードには 4 個の汎用 CPU (日立製 SH4, 200MHz) が搭載されており、そのうちの 1 つは制御用 PU (Figure 2 の黄色で示された PU [S-#])、残りの 3 個は専用 LSI に代わって二電子積分を実行する計算用 PU (Figure 2 の赤色で示された PU [E-#]) である。これらの SH4 CPU は μ ITRON OS により駆動され、それぞれ 64MB のメモリを利用可能となっている。SH4 プロセッサの倍精度浮動小数点演算は Pentium 等に比べて非常に遅く、乗算 (6 サイクル) でせいぜい 33MHz 程度の演算性能しかない。またボードにはハードディスク等の記憶デバイスは搭載されていない。システム全体としては 63 個 (3 ユニット \times 7 ボード \times 3PU) もの計算用 PU が存在することになり、プラットフォームシステム上での並列実行性能を評価するには十分な数のプロセッサであると言える。



Figure 1. EHPC SH4 processor board (left) and platform system (2-unit in picture).

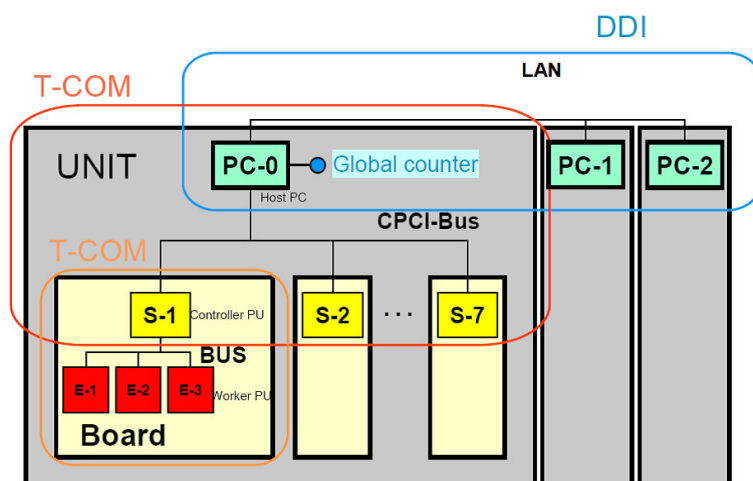


Figure 2. EHPC platform architecture. T-COM library is adopted for intra-unit communication, and DDI library in the GAMESS package for inter-unit communication.

2.2 プロセッサ間通信

ホスト PC 制御用 SH4 計算用 SH4 の通信は EHPC プロジェクトで開発されたプラットフォームシステム用通信ライブラリ Tree-Comm (T-COM) [4] を用いて行なわれる。T-COM では同期的な send/recv や wait_any といった MPI 類似の機能が実装されているが、階層構造を持つプラットフォームシステムのアーキテクチャ (Figure 2) を反映して、木構造を前提とした API になっている (Figure 3)。T-COM API における階層構造のもとでは、計算用 SH4 同士の直接通信はできず、そのボード上の制御用 SH4 に対してのみ通信が可能である。制御用 SH4 もそれら同士の通信はできず、その筐体の PC 又は自らの管理する計算用 SH4 とのみ通信が可能となっている。

Figure 3 に示されるような階層構造を持つシステムでは PC と末端の計算用 SH4、または計算用 SH4 同士のデータのやりとりには中間の制御用 SH4 を経由する必要があるので、小さなデータの転送でも非常に大きなコストが必要になる。特に T-COM API は同期通信を行なうため、中間階層を経由するような通信を効果的に制御するのは容易ではない。このため計算を局所化し、広範囲に影響を与える通信を可能な限り減らすことが求められる。

ユニット内の通信については階層構造による強い制約があるのに対し、ユニット間は TCP/IP ネットワークにより接続されているため MPI[6] 等を用いた自由な通信が可能である。実際 Figure 2 で示されるプラットフォームシステムのホスト PC は広く利用されている Linux システムであり、専用プロセッサを利用しない単なる PC クラスタとしての実行形態であれば、公式に配布されている GAMESS[5] を修正することなく並列計算が可能である。

本研究のターゲットである GAMESS[5] ではプロ

セッサ間の通信 API に DDI (distributed data interface) ライブラリ [7] を採用している。DDI ライブラリは send/recv や broadcast/global sum といった同期通信関数に加えて、get/put などの非同期通信による分散メモリ機構やこれを利用したグローバルカウンタの機構が実装されており、比較的簡単に動的負荷分散による効率の良い並列計算を実現することが可能である。グローバルカウンタ用の関数としてはカウンタを初期化する reset() 関数とカウンタの値を取り出しインクリメントする nextval() 関数が用意されており (Figure 4)、これを用いて動的負荷分散は次のように実現される。単一のタスク ID のみで指定される多数の独立したタスクを考える。この時グローバルカウンタからタスク ID が得られるようプログラムを設計すると、nextval() 関数によりプロセッサは遅延なく次に実行するべきタスクのタスク ID を取得することができる。これによって、プロセッサそれぞれの計算量に応じてタスク ID を取得する回数が動的に変化することとなり、比較的負荷が均一に分散した効率の良い並列計算が可能となる。

3 フォック行列生成のアルゴリズム

最終的なプラットフォームに搭載される専用 LSI は、プログラミングが可能な汎用プロセッサコアと二電子積分専用演算コアを持つマルチコア PU であるため [8]、柔軟なフォック行列生成のアルゴリズムの実装が可能になっている。そこでプラットフォームシステム上においても効率良く実行可能な並列計算アルゴリズムの開発および汎用 PU を用いたプラットフォーム評価システム上での実装を行なった。以下ではフォック行列の計算式および高速化手法である二電子積分のスクリーニングについて解説した後、本研究で開発した行列生成のアルゴリズムについて説明する。

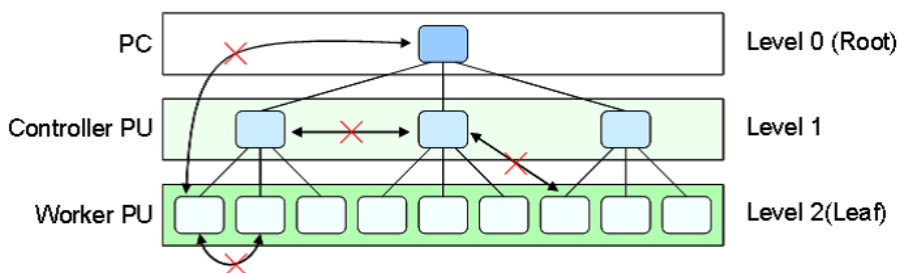


Figure 3. Tree-COM (T-COM) layered architecture. PUs can communicate only with their parent and child PU in a tree.

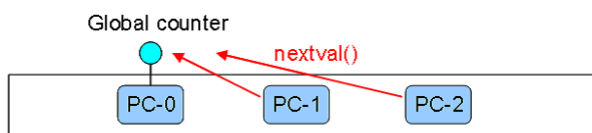


Figure 4. Global counter in DDI library for dynamic load balance. Each PC can access global counter allocated on PC-0 asynchronously through nextval() function.

3.1 フォック行列

非経験的分子軌道計算の一つである HF 法においてフォック行列 F は以下の式で与えられる [1]。

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

ここで i, j, k, l は縮約シェル (contracted shell, CS) のインデックスであり、タンパクなどの大規模分子ではその数 N が数万程度になることもある。 D, H^{core} はそれぞれ密度行列および核-電子ハミルトニアン行列であり、 $(ij|kl)$ は 4 中心の二電子積分である。式 1 に見られるように N^4 個の二電子積分を計算する必要があるため、大規模 HF 計算においてはフォック行列の生成がほとんどの計算時間を占めることとなる。特に二電子積分の計算はコストが大きい計算であり、これを高速に実行するための専用 LSI を作成する意味は大きい。

3.2 スクリーニングによる高速化

フォック行列を高速に計算するために一般的に使われる手法として、二電子積分のスクリーニングがある [1]。これは計算コストの大きい二電子積分の計算をできるだけ減らすために、事前に概算値を算出して不要と判断できる二電子積分の計算を省略する方法であり、本研究では代表的な 3 つの手法を採用した。

(a) 重なり積分によるスクリーニング

4 中心の積分のうち二つのガウス型関数の重なりがほとんどない場合、その関数ペアを含む二電子積分は非常に小さい値しか持たなくなるため、フォック行列への影響も無視できる程小さくなる。このような場合にはその二電子積分の計算を省略することができ、フォック行列の生成を高速化することが可能になる。例えば $N=1000$ 程度の場合、このスクリーニングによ

り生き残る CS ペア数は元々の CS ペア数の半分程度になるため [2, 9]、CS ペアの二重ループで構成される全体の計算量としては 1/4 に削減できることになる。分子サイズが大きくなればスクリーニングの効果は一層大きくなり、さらに効率の良い計算が可能である。

(b) シュワルツの不等式によるスクリーニング

二電子積分 $(ij|kl)$ は、シュワルツの不等式

$$(ij|kl) \leq \sqrt{(ij|ij)}\sqrt{(kl|kl)} \quad (2)$$

によって上限を決定できるため、これによるスクリーニングも可能である。特に CS ペア ij に対する $\sqrt{(ij|ij)}$ の値を事前に一度計算して保存しておけば、HF 法の SCF (self-consistent field, 自己無撞着場) 計算で発生する繰り返しごとの式 2 の評価を高速に実行できる。

(c) 密度行列によるスクリーニング

フォック行列は二電子積分と密度行列の積により求められるため (式 1)、式 2 で求められた二電子積分の上限とそれに対応する密度行列との積が十分小さい値であれば、やはりその二電子積分の計算を省略することができる。このスクリーニング手法は差密度行列によるフォック行列生成アルゴリズムを利用した場合には特に大きな効果を発揮する。SCF の収束につれ差密度が小さくなっていくため閾値以下の要素が急速に増加し、スクリーニングの効果が大きくなるからである。

3.3 生成アルゴリズム

Figure 5 にフォック行列生成の擬似コードを示す。ここで式 1 の核-電子ハミルトニアン行列 H^{core} は SCF の繰り返しに関わらず一定であり、また計算量も $O(N^2)$ と小さいため、事前に計算した行列をホスト PC で加算するものとする。スクリーニングの特徴から CS ペア i, j を一つの CS ペアインデックス ij で表わすと都合が良い。CS ペアインデックスは CS i と j の重なり条件を使ったスクリーニング (a) により生き残ったものだけに対し用意すればよいので、これを構造体の配列 `cs_idx[]` に格納する。また `cs_idx[]` にはシュワルツの不等式によるスクリーニング (b) に必要な $\sqrt{(ij|ij)}$ も格納しておき、`check_schwarz()` 関数においてスクリーニングを実行する。`make_fock()` は与えられた CS ペア対 $IJKL$ に対するフォック行列を計算し配列 $F[]$ に加えていく関数であり、密度行列によるスクリーニングもこの関数内で実装する。

```

0001 if (parallel) broadcast(D[])
0002 do IJ=0, NIJ-1 (screened)
0003   do KL=0, IJ
0004     survive = check_schwarz(cs_idx[IJ], cs_idx[KL])
0005     if (survive) make_fock(F[], D[], cs_idx[IJ], cs_idx[KL])
0006   end do
0007 end do
0008 if (parallel) all_reduce(F[])

```

Figure 5. Pseudocode of Fock matrix construction. N_{IJ} is the number of screened contracted-shell-pair (CS-pair) index, and an array of structure $cs_idx[IJ]$ stores pre-screened CS-pair indices and $\sqrt{(ij|ij)}$ value corresponding to screened index IJ . A Boolean function $check_schwarz()$ examines Schwarz' inequality screening, and a function $make_fock()$ constructs partial Fock matrix for CS-pair ij and kl onto an array $F[]$. $D[]$ is an array of density matrix.

ホスト PC から計算用 PU への通信量を減らすため、密度行列と結果のフォック行列はそれぞれ SCF 繰り返し部分の最初 (Figure 5, line 1) と最後 (Figure 5, line 8) でまとめて転送する。これはプラットフォームシステムの最終的なターゲットである専用 LSI が比較的大きなメモリ領域 (256M-512M バイト) を確保できる設計であり、EHPC プロジェクトでターゲットにしている数千基底程度の分子であれば密度行列及びフォック行列が十分保存できることを前提としている。それに対し本研究で利用した評価システムでは各計算用 PU に 64MB 程度のメモリしか搭載していないこと、ホスト PC のメモリも 64MB と少ないこと、計算用 PU である SH4 プロセッサの浮動小数点演算速度が遅いことなどの問題があり、数百基底の比較的小さな分子を扱うのが現実的である。

数万基底を超えるような超巨大分子を扱う場合には、個々のプロセッサにメモリをあまり必要としない分散メモリ用のアルゴリズム [9, 10] が必須となる。通信のオーバーヘッドが大きい本プラットフォームシステムのような階層構造を持つシステムには不向きであると思われるが、プログラマブルな専用 LSI を利用しているのでそのようなアルゴリズムの採用も十分に可能である。

4 並列化と負荷分散

Figure 5 の擬似コードを並列化するために、本研究では二種類のタスク分割 (TASK_IJKL, TASK_IJ) に対する静的負荷分散 (SLB-IJ, SLB-IJKL)、およびタスク分割 TASK_IJ に対する動的負荷分散 (DLB) の 3 つのコードをプラットフォーム評価システム上の GAMESS

に実装した。

4.1 タスクの分割

並列計算のためには Figure 5 のコードを多数のタスクに分割する必要がある。本研究では二種類のタスク分割 (TASK_IJKL, TASK_IJ) を採用した。

CS ペア対 IJKL をタスク ID としたタスク分割 (TASK_IJKL) では比較的粒度の小さいタスクが非常に多く発生する。この時シュワルツの不等式によるスクリーニングで生き残った IJKL に対するフォック行列の計算 ($make_fock()$) を一つのタスクとして考えると、タスクごとの負荷のばらつきを減らすことができる。一方でタスクの粒度が小さ過ぎるため、タスク分割処理自体のオーバーヘッドが見えてしまう可能性がある。

もう一つのタスク分割 (TASK_IJ) は CS ペア IJ をタスク ID としたタスク分割である。タスクの数は数万程度にはなるため並列化するのに十分なタスク数と考えられる。一方、TASK_IJKL に比べ平均的なタスクの粒度は大きいものの、計算量が kl ループのループ長 ij に依存してしまうためタスクの粒度が不均一になってしまう。

4.2 負荷分散

並列性能を向上させるためには、これらのタスクを各プロセッサに均一に分散させる必要がある。本研究ではタスク ID を均一に分配する形の静的負荷分散とグローバルカウンタを利用した動的負荷分散をそれぞれ採用した。

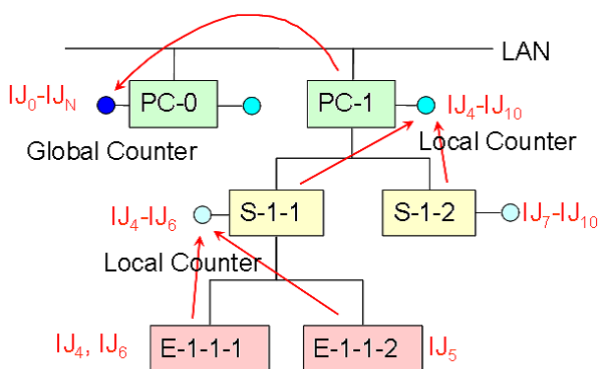


Figure 6. Multi-level dynamic load balance scheme. PC-0 has global counter for task ID, and other host PCs and controller SH4s have task manager with local task counter. TaskIDs IJ_n are allocated to processors in a demand-driven manner.

静的負荷分散 (static load balance, SLB) は並列化において通信回数を最も削減するタスクの分配方法である。各PUで計算するタスクを事前に割り振るため、最初と最後の密度行列およびフォック行列の転送以外に通信の必要はない。一方で静的負荷分散では、計算量に合わせてタスクを均等に分配するために多くの労力が必要となる。シュワルツの不等式によるスクリーニングの影響を事前に考慮するのは非常に困難であるため、本研究ではタスクIDを均等に分配する単純な静的負荷分散を採用した。このような単純な負荷分散では、タスク粒度のばらつきが大きいタスク分割 $TASK_{IJ}$ で計算量の不均一が発生する可能性が高いと思われる。

動的負荷分散 (dynamic load balance) はタスクの分配を動的に行なう方法である。各PUの負荷状況により動的にタスクを分配するため、PU間の負荷の偏りを防ぐことができ効果的に並列計算を実行することができる。一方でタスクID転送のため通信の頻度が増加し、環境によっては通信のオーバーヘッドが並列性能を阻害する可能性がある。また動的負荷分散のためにはタスクの実行状況を管理する必要があり、T-COM APIのように木構造を持っている場合には実装が困難であった。

階層構造を持つプラットフォームシステムで動的負荷分散を実現するために、多段階動的負荷分散機構を実装した (Figure 6)。ホストPC間の動的負荷分散はGAMESSのDDIライブラリが持つ `nextval()` 関数によ

るグローバルカウンタ機構により実現している。ホストPC及び制御用SH4には、より下位のPUにタスクを分配するためのタスクマネージャの機構を実装し、各ツリー内に局在した動的負荷分散を実現した。各タスクマネージャでのタスクの分配は、下位のPUからの要求を受けてタスクを分配するデマンド駆動方式を採用し、負荷の分散を計っている。階層ごとにタスクの粒度 (=分配するタスクIDの数) を変えることで上位層へのタスク要求の頻度を抑え、動的負荷分散による通信オーバーヘッドの影響を最小にした。

動的負荷分散を行なうための通信負荷を考えると、タスクの分割 $TASK_{IJKL}$ では計算時間が短かすぎるため適当ではない。そこでタスク分割 $TASK_{IJ}$ に対してのみ動的負荷分散機構を実装した。 $TASK_{IJ}$ では、先に述べたように IJ が大きい程 KL ループのループ長が長くなる傾向がある。そこで計算量の大きい (= IJ の大きい) タスクから先に分配するようタスクIDを設定した。今回のようなデマンド駆動方式の動的負荷分散の場合には、計算量の大きいタスクを分配することで生まれる負荷の不均衡を計算量の小さいタスクで平均化する効果が期待できる。

動的負荷分散を実行する場合、CSペアの段階で可能な限りスクリーニングすることが重要である。計算用PUで全てのスクリーニングを実行した場合、割り当てられたタスクの半分以上についての計算がスクリーニングによって省略されることになってしまい、タスク要求にかかる通信負荷が非常に高くなってしまからである。CSペアに対して可能なスクリーニングを IJ ループの段階で行なう Figure 5 のアルゴリズムは、この点において動的負荷分散に適したアルゴリズムである。

5 ベンチマーク計算

対象とした計算はC末端をOHキャップしたグリシンの15量体 $(Gly)_{15}$ (Figure 7) のHF/STO-3G計算である。この分子は108個の原子からなり、STO-3Gレベルの計算ではガウス型基底の数 $N=352$ となっている。スクリーニングの条件などについてはGAMESSのデフォルトの値を用いた。スクリーニング後のCSペアの数は $N_{IJ}=14698$ であり、CSペアの数26565個と比較すると半分程度に減少する。

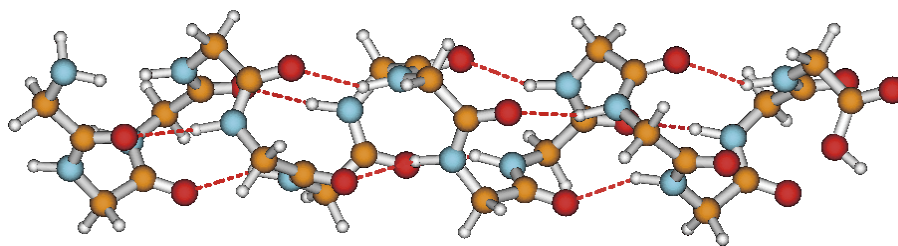


Figure 7. (Glycine)₁₅.

Table 1. Benchmark results on the EHPC platform system with 63 SH4 CPUs. Wfock is the total elapsed time of Fock matrix constructions for 13 SCF cycles, and Wtotal is the total elapsed time of benchmark calculation.

	Platform system (SH4, 200MHz)			PC(Pentium4, 2.8GHz)
	DLB	SLB-IJ	SLB-IJKL	
Wfock /sec	4486.7	5595.3	4837.5	2517.5
Speedups /board	20.724	16.854	19.244	
Speedups /chip	62.562	50.143	57.909	
Ratio	0.993	0.796	0.919	
Wtotal /sec	4659.5	5775.4	4995.6	2526.4
Speedups /board	19.987	16.353	18.664	
Speedups /chip	60.152	48.530	56.105	
Ratio	0.955	0.770	0.891	

プラットフォームシステムのホスト PC 上で動く GAMESS を含むプログラムは全て GNU g77 を利用してコンパイルした。計算時間の大部分を占めるフォック行列の生成部分は計算ボード上で実行されるため、これによる影響はごく小さいものと思われる。ボード上の SH4 用プログラムのコンパイルには日立製コンパイラを利用している。SH4 プロセッサで動作する二電子積分演算ルーチンは、専用 LSI のロジック作成用に用意された小原積分ルーチン [11] であり、汎用プロセッサにとって最適なプログラムとはなっていない。

比較のため、2.8GHz の Pentium4 PC 上でインテルコンパイラによりコンパイルされた公式配付版 GAMESS を実行するベンチマークも行なった。なおこの時の二電子積分の計算には精度を上げるために GAMESS の HONDO オプションを利用している。これは本システムのターゲットとなる巨大分子では精度の高い二電子積分が必要になると考えられるからである。

Table 1 に 3 ユニット (=63PU) のプラットフォーム評価システムおよび Pentium4 PC を利用したベンチマーク計算の結果を示す。DLB は TASK_IJ に対する

動的負荷分散、SLB-IJ, SLB-IJKL はそれぞれ TASK_IJ および TASK_IJKL に対する静的負荷分散の結果である。Wfock は Fock 行列の生成に対する経過時間であり、Wtotal は計算全体に対するものである。Pentium4 PC の結果から 13 回の SCF サイクルの合計でフォック行列の生成に 2500 秒程度かかっており、計算全体の 99% 以上の時間をフォック行列の生成に費やしていることがわかる。Speedup はそれぞれの負荷分散アルゴリズムで 1 つのボード (またはチップ) のみを利用した場合と比較した時の性能向上である。また Ratio は並列化効率であり、リニアな性能向上であれば 1 となる。

3 つの負荷分散の手法を比較すると、動的負荷分散の手法が際だって優れていることがわかる。速度の向上はボード数・PU 数の増加に対してほぼ線形 (並列化効率 99.3%) であり、今後さらにボード数を増加させた場合でも高い性能を与えることが十分期待できる。計算全体でみた場合についても、速度が遅いホスト PC が含まれているにも関わらず高い並列性能を示している。

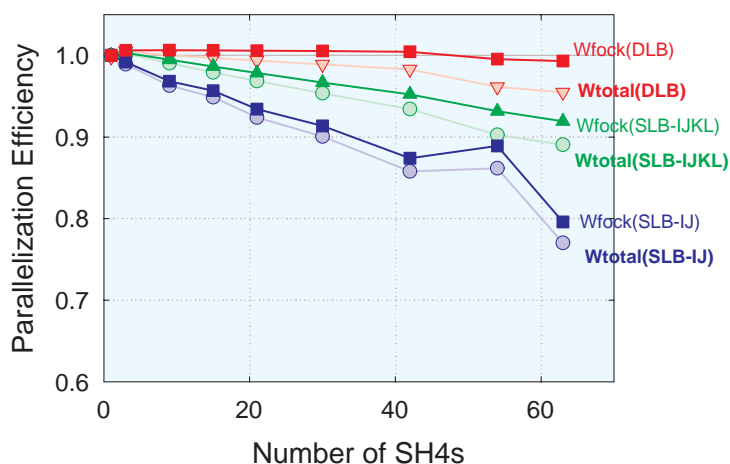


Figure 8. Parallelization Efficiency of three different load-balance schemes on the EHPC platform system with 63 SH4 CPUs. Wfock is the total elapsed time of Fock matrix constructions for 13 SCF cycles, and Wtotal is the total elapsed time of the benchmark calculation.

並列化効率をPUの数に対してプロットしたのが Figure 8 である。動的負荷分散の並列化効率が非常に高いことがよりはっきりと示されている。SLB-IJ ではシュワルツの不等式等によるスクリーニングの影響でタスクの粒度にばらつきが大きく、そのためボードの数によって並列化効率に大きな変動が表われる。一方で SLB-IJKL ではそのような変動は見られていない。これはシュワルツの不等式によるスクリーニング後の IJKL をプロセッサに分配しているため、計算すべき二電子積分が均等に分配されたからである。しかしながら二電子積分計算自体はその積分タイプにより計算量が異なるため、十分な負荷分散は得られていない。Figure 8 において 54 および 63 プロセッサ時に並列性能が悪くなる傾向が負荷分散の方法に関わらず見られているが、これは遅いホスト PC (Pentium2, 266MHz, 64MB メモリ) が利用されたことによる影響と思われる。

今回並列性能の評価に用いた SH4 プロセッサの倍精度浮動小数点演算速度は 33MHz 程度であり、また二電子積分計算アルゴリズムの違いもあって、2.8GHz の Pentium4 に比べ 1/100 程度の速度しかない。これを反映しプラットフォーム評価システム (63PU システム) 全体で、Pentium4 PC の半分程度の性能に留まる結果となってしまう。200MHz で動作する予定の専用 LSI では、専用ロジックを用いることで最終的に SH4 プロセッサの十倍以上の性能が予想されてお

り、1 ユニットで Pentium4 PC の数倍の速度が期待できる。計算 PU の性能が向上すると、タスクひとつの計算時間が減少し、相対的に通信等が並列性能を阻害する可能性も考えられる。しかしながら本研究で開発した多段階の動的負荷分散機構ではその各段階でタスクの粒度を調整することが可能であり、本ベンチマークと同等の並列性能が容易に実現できる。通信環境の変化についても同様の議論が可能であり、多数ユニットを用いたスケーラブルな高速高並列分子軌道計算が実現可能である。

6 おわりに

大規模分子軌道計算においてフォック行列の生成は計算量の多い計算であり、この部分を専用計算機により高速化する意味は非常に大きい。EHPC プロジェクトにより開発された二電子積分計算専用 LSI が組み込まれるプラットフォームシステムは階層的構造を持つアーキテクチャであり、これを考慮した並列化技法の開発が必要であった。本研究ではプラットフォームアーキテクチャの評価システム上にフォック行列生成ルーチンを実装し、並列性能を評価した。我々の開発した多段階の動的負荷分散を利用することで、通信頻度を抑制しつつ負荷を均一に分散させることに成功し、63PU を利用した場合に 99% 以上の非常に高い並列性

能を得ることができた。これは、数百以上の専用 LSI を用いた高速高並列分子軌道計算が可能であることを示している。

このような階層的なネットワーク構造を持つ分散処理システムとしてグリッド RPC モデル [12] がある。本研究で開発された多段動的負荷分散はグリッド RPC モデルにも同様に適用可能であるばかりか、グリッド RPC モデルの中に専用計算機システムを組込むことも可能である。これにより利用者は専用計算機の実在を意識する必要がなくなり、専用計算機の稼働率の向上などの効果も期待できる。

本研究の一部は科学技術振興調整費 総合研究「科学技術計算専用ロジック組込み型プラットフォーム・アーキテクチャに関する研究」(代表 村上和彰 九州大学教授) によるものである。

参考文献

- [1] T. Helgaker, P. Jorgensen, and J. Olsen, *Molecular Electronic-Structure Theory*, Wiley (2000).
- [2] 長嶋雲兵, 計算化学 << 空間系 V >>, 岩波講座 現代工学の基礎, 岩波出版 (2002).
- [3] Embedded High Performance Computing (EHPC) Project, <http://www.ehpc.jp>
村上和彰, 稲垣祐一郎, 上原正光, 大谷康昭, 小原繁, 小関史朗, 佐々木徹, 棚橋隆彦, 中馬寛, 塚田捷, 長嶋雲兵, 中野達也, 情報処理学会研究報告, **2000-HPC-82**, 1 (2000).
長嶋雲兵, 佐々木徹, 大谷泰昭, 上原正光, 塚田捷, 村上和彰, *J. Comput. Chem. Jpn.*, **4**, 131 (2005).
- [4] T. Sasaki, U. Nagashima, *J. Comput. Chem. Jpn.*, **2**, 111 (2003).
佐々木徹, 村上和彰, *J. Comput. Chem. Jpn.*, **4**, 139 (2005).
- [5] M. W. Schmidt, K. K. Baldridge, J. A. Boatz, S. T. Elbert, M. S. Gordon, J. H. Jensen, S. Koseki, N. Matsunaga, K. A. Nguyen, S. J. Su, T. L. Windus, M. Dupuis, and J. A. Montgomery, *J. Comput. Chem.*, **14**, 1347 (1993).
<http://www.msg.ameslab.gov/GAMESS/GAMESS.html>
- [6] The Message Passing Interface (MPI), <http://www-unix.mcs.anl.gov/mpi/>.
- [7] G. D. Fletcher, M. W. Schmidt, B. M. Bode, M. S. Gordon, *Comput. Phys. Commun.*, **128**, 190 (2000).
- [8] 原田宗幸, 中村健太, 桑山庸史, 上原正光, 佐藤比佐夫, 小原繁, 本田宏明, 長嶋雲兵, 稲富雄一, 村上和彰, 情報処理学会研究報告, **2002-ARC-152**, 133 (2003).
中村健太, 本田宏明, 梅田宏明, 小松晴信, 村上和彰, *J. Comput. Chem. Jpn.*, **4**, 155 (2005).
- [9] H. Takashima, S. Yamada, S. Obara, K. Kitamura, S. Inabata, N. Miyakawa, K. Tanabe, and U. Nagashima, *J. Comput. Chem.*, **23**, 1337 (2002).
- [10] I. T. Foster, J. L. Tilson, A. F. Wagner, R. L. Shepard, R. J. Harrison, R. A. Kendall, and R. J. Littlefield, *J. Comput. Chem.*, **17**, 109 (1996).
- [11] S. Obara, A. Saika, *J. Chem. Phys.*, **84**, 3963 (1986).
H. Honda, T. Yamaki, S. Obara, *J. Chem. Phys.*, **117**, 1457 (2002).
小原繁, 本田宏明, *IPJS Symposium Series High Performance Computing Symposium 2003*, **2003(4)**, 71 (2003).
本田宏明, 小原繁, *IPJS Symposium Series High Performance Computing Symposium 2003*, **2003(4)**, 121 (2003).
- [12] Ninf-G, <http://ninf.apgrid.org/>.

Parallel Fock Matrix Construction Algorithm for Molecular Orbital Calculation Specific Computer

Hiroaki UMEDA^{a,b*}, Yuichi INADOMI^{a,b}, Hiroaki HONDA^c and Umpei NAGASHIMA^{a,b}

^aCREST, Japan Science and Technology Agency

4-1-8 Honcho Kawaguchi, Saitama, 332-0012 Japan

^bGrid Technology Research Center, National Institute of Advanced Industrial Science and Technology

Tsukuba Central 2, 1-1-1 Umezono, Tsukuba, Ibaraki, 305-8568 Japan

^cResearch Division, Computing and Communications Center, Kyushu University

6-10-1 Hakozaki, Higashi-Ku, Fukuoka, 812-8581 Japan

**e-mail: h-umeda@aist.go.jp*

Parallel Fock matrix construction algorithm for molecular orbital (MO) calculation specific computer (Figure 1) has been developed. MO-specific computer consists of massive custom processors to calculate two-electron integrals on the EHPC platform system (Figure 2), which have layered architecture (Figure 3). In order to obtain high parallelization efficiency for Fock matrix construction (Figure 5) on the layered architecture, multi-level dynamic load balance scheme (Figure 6) is adopted to get better load balance and to localize communications within a tree structure of Tree-Comm network API. Parallelized Fock matrix construction routine was implemented into GAMESS ab initio program on EHPC platform system, which uses SH4 processor instead of special purpose LSI. Benchmark result on 63 worker SH4 processor system shows extremely high parallelization efficiency (Table 1, Figure 8) on the platform system.

Keywords: EHPC(Embedded High Performance Computing), Molecular orbital calculation specific computer, Multi-level dynamic load balance scheme, Parallel Fock matrix construction algorithm, Platform system with layered architecture