計算機アーキテクチャ　118－8
(1996. 5. 16)

超高速分子軌道計算専用機 MOE のアーキテクチャ

白川 暁[†]　吉井 卓[†]　村上和彰[†]　長嶋雲兵[††]　小原 繁[†††]
網崎孝志[††††]　北村一泰[†††††]　高島 一[†††††]　田辺和俊[††††††]

[†]九州大学 大学院システム情報科学研究科　[††]お茶の水女子大学 理学部
[†††]北海道教育大学釧路校 教育学部　[††††]島根大学 総合理工学部
[†††††]大正製薬株式会社　[††††††]物質工学研究所
E-mail: moe@hososipc.chem.ocha.ac.jp

あらまし　筆者らは，非経験的分子軌道計算を高速に行うことを目的として，超高速分子軌道計算専用機 MOE の開発を行っている．MOE では 2 電子積分値の独立性を利用して，これを並列実行することで高速化を図る．本論文では，MOE のアーキテクチャについて議論している．アーキテクチャ策定に際して，1) 並列化方式および 2) 積和表生成方式について検討を行った．並列化方式としては，演算量が問題サイズ n に対して n^4 あることから，n，n^2，n^3，n^4 の 4 通りの並列化が可能である．これら各並列化方式の価格対性能比を評価した結果，n 並列化を採用している．また，積和表生成方式については，各 PE で自律的にこれを生成する方式を採用した．さらに，本稿では，策定したアーキテクチャ，特に積和演算の部分について述べている．

キーワード　非経験的分子軌道計算，専用計算機，並列処理

The Architecture of a Molecular Orbital calculation Engine (MOE)

Satoru SHIRAKAWA[†]　Takashi YOSHII[†]　Kazuaki MURAKAMI[†]
Umpei NAGASHIMA[††]　Shigeru OBARA[†††]　Takashi AMISAKI[††††]
Kunihiro KITAMURA[†††††]　Hajime TAKASHIMA[†††††]
Kazutoshi TANABE[††††††]

[†]Kyushu University　[††]Ochanomizu University　[†††]Hokkaido University of Education
[††††]Shimane University　[†††††]Taisho Pharmaceutical Co. Ltd.
[††††††]National Institute of Materials and Chemical Reactions

Abstract　The authors are developing a high-performance, special-purpose parallel machine for Molecular Orbital (MO) calculations, called MOE (Molecular Orbital calculation Engine). The sequential execution time is $O(n^4)$ where n is the number of basis functions, and most of the time is spent to the calculations of two-electron integrals. The calculations of two-electron integrals have a lot of parallelism of $O(n^4)$, and therefore the MOE tries to exploit the parallelism. This paper discusses the MOE architecture and examines two important aspects of architecture design: 1) how to exploit the parallelism, and 2) how to create a data structure, called *LABEL*, which is required to calculate two-electron integrals according to Obara's algorithm. The authors conclude that a n-way parallelization is the most cost-effective although there are lots of parallelism of $O(n^4)$. They also conclude that an MIMD operation is adequate to the creation of *LABEL*. The paper further describes the MOE architecture in detail.

key words　molecular orbital calculations, special-purpose computers, parallel processing

1 はじめに

非経験的分子軌道計算は，問題サイズ (基底関数の数) の4乗に比例する計算量と記憶量を必要とする大規模なアプリケーションである．我々は，この分子軌道計算を高速に実行することを目標に専用並列計算機 MOE(*Molecular Orbital calculation Engine*) の開発を進めている [1].

本稿では MOE のアーキテクチャについて述べる．まず，2章で MOE で行う計算内容について述べ，3章で MOE のアーキテクチャを検討する．そして，4章でアーキテクチャ，特に積和演算に関する部分について述べる．

2 MOE の計算内容

MOE はワークステーション，パソコンなどのホスト・コンピュータ (以下，HOST) に SCSI などのホスト・インターフェース (以下，HIF) を介して接続することにより HOST と協調して計算を行う．

非経験的分子軌道計算法として，ハートリーフォック (HF) 法 [4] を用いる．フォック行列の計算式は，

$$F_{rs} = T_{rs} + V_{rs} + \sum_{t=1}^{n}\sum_{u=1}^{n} P_{tu}\left\{g_{rstu} - \frac{1}{2}g_{rtsu}\right\} \quad (1)$$
$$(r,s = 1 \sim n)$$

である．MOE で計算するのは上式の第3項で，これを F'_{rs} とする．n は問題サイズで基底関数の数に等しい．F'_{rs} を計算するためには P_{tu} および g_{rstu} が必要である．$P_{tu}(t,u = 1 \sim n)$ は密度行列と呼ばれるもので HOST から送る．g_{rstu} は2電子積分と呼ばれるもので MOE で計算する．

2電子積分 g_{rstu} の計算には，小原の方法 [2] を用いる．まず，2電子積分 g_{rstu} を計算するためには基底関数の情報が必要となる．この基底関数の情報は HOST から MOE に送られ，MOE はそれから A, B, 積和表 $LABEL$ の3つの行列を生成する．この3つの行列を用いて g_{rstu} を計算する．A, B, $LABEL$ から g_{rstu} を計算するプログラムを図1に，概要を図2に示す．

先に A は基底関数の情報から生成すると述べたが，正確には A の先頭数行 (10 行程度) のみ (以下，A_s) を基底関数の情報から生成する．A の残りの部分は図1に示すプログラムにより求め，A の末尾 10～100 行程度 (以下，A_f) が g_{rstu} になる (g_{rstu} は r,s,t,u $= 1 \sim n$ であるので n^4個存在するが，ある一つの g_{rstu} は 10 ～ 100 個の要素から成る)．行列 A の列方向のインデックス L は g_{rstu} の添字 $rstu$ によって一意に決まる．つまり，A_f のある一列の要素はある特定の $rstu$ に対する g_{rstu} になる．以下，行列 A の1列を表すとき $A(rstu)$ と書く．行列 B も同様である．

A_f の計算は次式を $n^4 \times NN$回行うことにより終了する．

$$A(I,L) = A(I,L) + B(J,L) \times A(K,L) \quad (2)$$

```
    DO 200 M=1, NN
       I=LABEL(1,M)
       J=LABEL(2,M)
       K=LABEL(3,M)
       DO 100 L=1,N
100       A(I,L)=A(I,L)+B(J,L)*A(K,L)
200 CONTINUE
```

図 1: 2 電子積分計算のプログラム

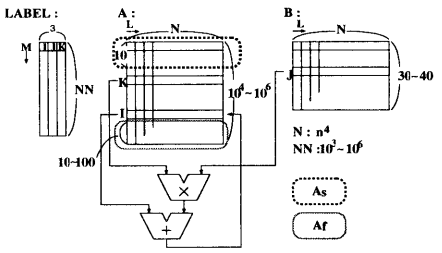

図 2: 2 電子積分計算の概要

MOE では，複数個の PE(Processing Element) を用いて上式の計算を並列に行うことにより計算の高速化を図る．PE の個数を p とし上式の計算を 1 回行うのに 1 ステップかかるとすると，A_f の計算は $n^4/p \times NN$ ステップで行うことができる．

結局，MOE で行う計算部分の処理の流れをまとめると以下のようになる．1) 入力データ (基底関数の情報，密度行列 P) を HOST から受信，2) 基底関数の情報から A_S, B, $LABEL$ を生成，3)A_S, B, $LABEL$ から A_f(2 電子積分値に等しい) を計算，4)2 電子積分値と密度行列 P から F'_{rs} を計算，5)F'_{rs} を HOST へ送信．

3 アーキテクチャの検討

前章で述べた処理の内，次の2つはアーキテクチャの策定に際して十分検討する必要がある．

- 並列化方式: 式2で示した A_f の計算をどう並列化するか?
- $LABEL$ 生成方式: $LABEL$ 生成は単純な積和演算では行えないので，これをどう行うか?

3.1 並列化方式

2 電子積分 g_{rstu} を複数個の PE により並列に計算する．このとき，g_{rstu} をどのように PE に割り当てるかが重要になってくる．また，個々の g_{rstu} がどの F'_{rs} に寄与するかについても注意する必要がある．そこで,『添字 $rstu$ の内，どの添字について並列化するか』について検討する．並列化に際しては，以下の 16 種類の選択肢が存在する．

- 逐次処理 (1 種類)
- n^4並列処理: $rstu$ の全添字について並列化 (1 種類)
- n^3並列処理: rst, rsu, rtu, stu の 3 添字についてそれぞれ並列化 (計 4 種類)
- n^2並列処理: rs, rt, ru, st, su, tu の 2 添字についてそれぞれ並列化 (計 6 種類)
- n 並列処理: r, s, t, u の 1 添字についてそれぞれ並列化 (計 4 種類)

F'_{rs} の計算に要する計算時間およびメモリ量を比較する．計算時間は積和演算の回数で表し，メモリ量は記憶すべきデータの個数で表す．1 個の g_{rstu} の計算には，G 回の積和演算および W個のデータ領域を必要とするものと仮定する．十分な数の PE があると仮定して，プログラム中

の forall の各イタレーションに対して1個の PE を割り当てる．

以下に，逐次処理方式および添字 r に関する n 並列化方式のアルゴリズム，計算時間およびメモリ量を示す．

逐次処理方式

```
for r
  for s
    for t
      for u
        g_rstu計算
        F'_rs := F'_rs + P_tu * g_rstu
        F'_rt := F'_rt − 1/2 * P_eu * g_rstu
      forend
    forend
  forend
forend
```

最内ループの積和回数が $G+2$ 回であり，それが n^4 回繰り返されるので計算時間は $n^4(G+2)$ となる．また，g_{rstu} の計算の作業領域が W 個，$F'_{rs}(r,s=1\sim n)$ が n^2 個，$P_{tu}(t,u=1\sim n)$ が n^2 個それぞれ必要なので，メモリ量は $W+n^2+n^2$ となる．

$n(r)$ 並列処理方式

```
forall r
  for s
    for t
      for u
        g_rstu計算
        F'_rs := F'_rs + P_tu * g_rstu
        F'_rt := F'_rt − 1/2 * P_eu * g_rstu
      forend
    forend
  forend
forallend
```

計算時間は $n^3(G+2)$，メモリ量は全体で $n(W+n+n^2)$ となる．

3.1.1 検討結果

表1に全ての並列化方式の検討結果を示す．各方式の価格対性能比を「PE 数×計算時間」で見てみる．逐次処理方式で $n^4(G+2)$，n^4 並列処理方式では $n^5(G+2)+2n^6$，n^3 並列処理方式では最低で $n^4(G+4)$，n^2 並列処理方式では最低で $n^4(G+2)+n^3$，そして n 並列処理方式では最低で $n^4(G+2)$ となる．これから，$n(r)$ 方式が最も価格対性能比として優れていることがわかる．一方，メモリ量に関しては，実用的な問題サイズから $W=O(10^6)$，$n=O(10^3)$ と仮定した場合，各方式とも $O(10^6)$ となりほぼ互角である．以上から，$n(r)$ 方式を並列化方式として採用することにする．

3.2 *LABEL* 生成方式

LABEL は2電子積分とは異なり，単純な積和演算では生成できない．したがって，*LABEL* 生成可能な機能をどこに設けるかが課題となる．さらに，*LABEL* は積分のタイプによって異なる[1]ので，各 PE が必要とする *LABEL* は等しくはない．そこで，以下の3つの *LABEL* 生成方式について，メモリ量，データ転送量および処理時間を比較する．

- α案:ある PE が親 PE となり，その親 PE が全ての type の *LABEL* を生成してそれを全ての子 PE に一括転送．子 PE はそれを確保しておいて必要な *LABEL* を使う．
- β案:親 PE が全ての type の *LABEL* を生成して，子 PE 毎に必要とするタイプの *LABEL* だけを分割して転送する．
- γ案:(PE には親も子もなく) 各 PE が自分で *LABEL* を生成する．

以下に，各案のアルゴリズムを示す．

α案

```
親が LABEL 生成
forall r
  基底関数の情報 受信 /* O(n) */
  LABEL 受信 /* (10MB) */
  A_s(rstu),B(rstu) 生成 /* step1 O(n^2) */
  for u=1～n
    for t=1～n
      for s=1～n
        P_tu,P_eu受信
        A_s(rstu),B(rstu) 生成 /* step2 O(1) */
        for i=1～NN /* NN は (rstu) に依存 */
          A(rstui),B(rstui) の積和
        forend
        g_rstu := 積和結果
        F'_rs := F'_rs + P_tu * g_rstu
        F'_rt := F'_rt − 1/2 * P_eu * g_rstu
      forend
    forend
  forend
forallend
```

β案

```
親が LABEL 生成
forall r
  基底関数の情報 受信 /* O(n) */
  A_s(rstu),B(rstu) 生成 /* step1 O(n^2) */
  for u=1～n
    for t=1～n
      for s=1～n
        LABEL 受信 /* (1MB) */
        P_tu,P_eu受信
        A_s(rstu),B(rstu) 生成 /* step2 O(1) */
        for i=1～NN /* NN は (rstu) に依存 */
        — 以下，α案と同じ —
```

γ案

```
forall r
  基底関数の情報 受信 */ O(n) */
  A_s(rstu),B(rstu) 生成 /* step1 O(n^2) */
  for u=1～n
    for t=1～n
      for s=1～n
        P_tu,P_eu受信
        A_s(rstu),B(rstu) 生成 /* step2 O(1) */
        LABEL 生成
        for i=1～NN /* NN は (rstu) に依存 */
        — 以下，α案と同じ —
```

3.2.1 検討結果

上で述べた3つの案において *LABEL* の転送量およびメモリ量について比較を行う．表2に比較結果を示す．表2においてα案およびβ案では *LABEL* が，γ案では基底関数の情報が転送の対象となっている．また，α案およびγ案で PE 当りの転送量と全体の転送量が変わらないのは，対象となるデータが各 PE にブロードキャスト可能なためである．また，L というのは子 PE が使う *LABEL* のタイプが変化した回数を表し，最低で0，最大でタイプの総数となる．

次に，各案の処理時間を比較する．図3，4，5に各案の処理の流れをそれぞれ示す．図中の横軸は時間を示しており，通信の場合は転送データ数で，計算の場合は積和演算

[1] 積分のタイプはその積分で用いる基底関数の種類によって決まる．例えば r が s 関数，s が p 関数，t が p 関数，u が s 関数ならばタイプは spps となる．

表 1: 各並列化方式の評価

方式	PE 数	計算時間†	1PE 当りのメモリ量‡	全体のメモリ量‡
逐次処理	1	$n^4(G+2)$	$W+2n^2$	$W+2n^2$
n^4並列処理	n^4	$n(G+2)+2n^2$	$W+4$	$n^4(W+4)$
$n^3(r,s,t)$ 並列処理	n^3	$n(G+4)$	$W+2+2n$	$n^3(W+2+2n)$
$n^3(r,s,u$ 並列処理)	n^3	$n(G+3)+2n^2$	$W+1+2n$	$n^3(W+1+2n)$
$n^3(r,t,u)$ 並列処理	n^3	$n(G+3)+2n^2$	W+1+2n	$n^3(W+1+2n)$
$n^3(s,t,u)$ 並列処理	n^3	$n(G+2)+2n^2$	$W+2+2n$	$n^3(W+2+2n)$
$n^2(r,s)$ 並列処理	n^2	$n^2(G+2)+2n$	$W+1+n+n^2$	$n^2(W+1+n+n^2)$
$n^2(r,t)$ 並列処理	n^2	$n^2(G+2)+2n$	$W+1+n+n^2$	$n^2(W+1+n+n^2)$
$n^2(r,u)$ 並列処理	n^2	$n^2(G+2)+n$	$W+2n$	$n^2(W+2n)$
$n^2(s,t)$ 並列処理	n^2	$n^2(G+2)+2n$	$W+4n$	$n^2(W+4n)$
$n^2(s,u)$ 並列処理	n^2	$n^2(G+3)+n$	$W+2n+n^2$	$n^2(W+2n+n^2)$
$n^2(t,u)$ 並列処理	n^2	$n^2(G+3)+n$	$W+2n+n^2$	$n^2(W+2n+n^2)$
$n(r)$ 並列処理	n	$n^3(G+2)$	$W+n+n^2$	$n(W+n+n^2)$
$n(s)$ 並列処理	n	$n^3(G+2)+n^2$	$W+n+2n^2$	$n(W+n+2n^2)$
$n(t)$ 並列処理	n	$n^3(G+2)+n^2$	$W+n+2n^2$	$n(W+n+2n^2)$
$n(u)$ 並列処理	n	$n^3(G+2)+n^2$	$W+n+n^2$	$n(W+n+n^2)$

†: 積和演算回数. ‡: データ数.

表 2: $LABEL$ についてのメモリおよび転送量の比較

項目	α案	β案	γ案
1 回当りの転送量/PE	10^6	10^5	n
転送の回数/PE	1	$L+1$	1
転送量/PE	10^6	$(L+1)10^5$	n
全体の転送量	10^6	$n(L+1)10^5$	n
メモリ量/PE	10^6	10^5	10^5
全体のメモリ量	$n10^6$	$n10^5$	$n10^5$

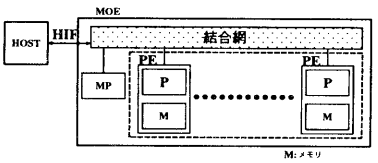

図 6: システム構成

回数でこれを表す．ここでデータ 1 個の転送時間と積和 1 回の計算時間は等しいとする．

α案 $L_a+10^7+n+O(n^2)+n^3(O(1)+NN+2)+n$

β案 $L_a+10^6*(L+1)+n+O(n^2)+n^3(O(1)+NN+2)+n$

γ案 $n+O(n^2)+n^3(O(1)+NN+2)+L_s*(L+1)+n$

L_aは全てのタイプの $LABEL$ を生成するのに要する時間，L_sはある 1 つのタイプの $LABEL$ を生成するのに要する時間をそれぞれ表す．

支配的な項 (n^3の項) の係数が 3 案とも全て等しいので，処理時間はほぼ同等と言える．よって，転送量およびメモリ量がもっとも少ないことからγ 案が他の 2 案よりも優れている．したがって，MOE ではγ案を採用する．

4 MOE のアーキテクチャ

図 6に，MOE のシステム構成を示す．これは以下の特徴を有する．

- 各 PE は自律して動作する (MIMD)．採用したγ案の計算の流れにおいて，for i ループの回数 NNが積分のタイプ ($rstu$) に依存するので各 PE でループ回数が変化する．したがって，全 PE を SIMD で動作させることはできない．
- PE 1 クラスタが 1 つの密度行列 P 用のメモリ (以下，MP) を共有する．PE が SIMD で動くのであれば，各 PE が P_{tu}(密度行列 P の 1 要素) を必要とするタイミングは等しく，P_{tu}を各 PE にブロードキャストすることができる．しかし，PE は MIMD であるので 各 PE が P_{tu}を必要なときに MP から読み出す方式を採ることにする．PE は MIMD で動くのでアクセスのタイミングが PE 間でずれやすいこと，ならびに，ループの 1 イタレーションが長いにも関わらず P_{tu}はその間に 2 個しか必要でないことから，MP の応答速度はそれほど高速でなくても構わない．
- ローカル・メモリ M には高速な SRAM を，MP には大容量の DRAM を使用する．
- 結合網は以下の 4 つの機能を有する．PE 間の通信は全く行う必要がないので，そのための機能は備えない．
  1. ホスト・インターフェース (HIF) から PE へのブロードキャスト:基底関数の情報 ($O(n)$) の転送に使用．
  2. PE から HIF への 1 対 1 通信: $F'_{rs}(s=1\sim n)(O(n))$ の転送に使用．
  3. PE から MP への読出しアクセス: $P_{tu}(O(1))$ の転送に使用．
  4. HIF から MP への書込みアクセス: $P_{tu}(t=1\sim n,\ u=1\sim n)(O(n^2))$ の転送に使用．
- PE の持つべき機能は A_s, B, $LABEL$ 生成および積和演算機能である．

4.1 PE アーキテクチャ

図 7に PE のアーキテクチャを示す．命令セットは以下の通りである．

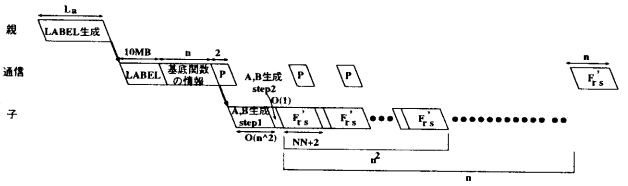

図 3: α案の処理の流れ

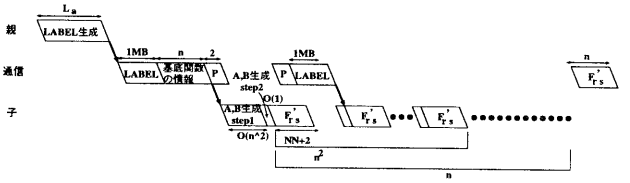

図 4: β案の処理の流れ

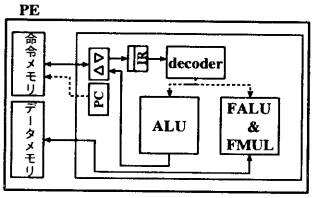

図 7: PE のアーキテクチャ

1. LOAD/STORE 命令 (命令メモリ，データメモリ，MP および HIF にアクセス)
2. 整数演算命令 (レジスタ-レジスタ演算)
3. 浮動小数点積和演算命令 (メモリ-メモリ演算)
4. 分岐命令

上記1および2は主に $LABEL$ 生成に使い，3は $A_s(rstu)$ および $B(rstu)$ の生成，$LABEL$ をプログラムにした積和演算および F'_{rs} の計算に使う．

4.2 $LABEL$ をプログラムとした 2 電子積分計算

浮動小数点積和演算命令は，以下の 3 オペランド形式のメモリ-メモリ演算命令である．

$$opI, J, K$$

op には以下の 4 種類がある．

- LOAD and MUL and ADD (LMA)
- MUL and ADD (MA)
- MUL and ADD and STORE (MAS)
- LOAD and MUL and ADD and STORE (LMAS)

表 3にその動作を示す．

2電子積分計算プログラムは，図8に示すような $LABEL$ の形を採る．図 9にその実行過程を示す．図 9では，命令メモリ (IMEM) およびデータ・メモリ (DMEM) のメモリ・アクセス・レイテンシは 1 クロック・サイクル時間，FMUL の命令発行レイテンシは 1 クロック・サイクル時間，パイプライン段数は 3，FADD の演算レイテンシは 2 クロック・サイクル時間と仮定した．

表 3: 浮動小数点積和演算命令の動作

動作	浮動小数点積和演算命令			
	LMA	MA	MAS	LMAS
LOAD $R_i \leftarrow M(I)$	1			1
LOAD $R_j \leftarrow M(J)$	1	1	1	1
LOAD $R_k \leftarrow M(K)$	1	1	1	1
MUL $tmp1 \leftarrow R_j \times R_k$	2	2	2	2
ADD $tmp2 \leftarrow R_i + tmp1$	3			3
ADD $tmp2 \leftarrow tmp2 + tmp1$		3	3	
STORE $M(I) \leftarrow tmp2$			4	4

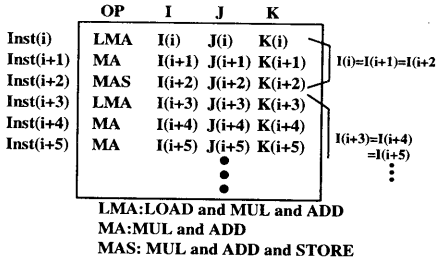

図 8: 2 電子積分計算プログラム

図 9から命令 Inst(i) と命令 Inst(i+1) との間の発行レイテンシ il_i は

$$il_i = \max\{MAC_i * rl_{Mem}, il_{FMUL}, rl_{FADD}\} \quad (3)$$

となる．ここで，

MAC_i:命令 Inst(i) の行うメモリ・アクセス回数で，2~4.

rl_{Mem}:データ・メモリのアクセス・レイテンシ．

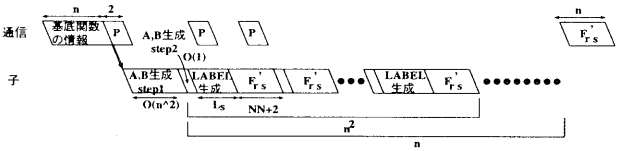

図 5: γ案の処理の流れ

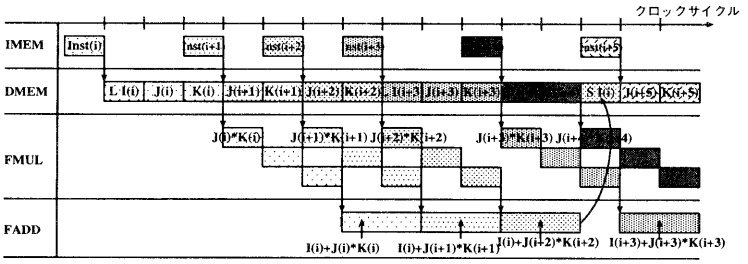

図 9: 2 電子積分計算プログラムの実行過程

il_{FMUL}:FMUL の発行レイテンシ.
rl_{FADD}:FADD の演算レイテンシ.

il_{FMUL}はFMUL をパイプライン化することで低減できるので，上式で支配的なのは $MAC_i * rl_{Mem}$ か rl_{FADD} のどちらかである．rl_{FADD} = 60ns の製品が実際に存在する．また rl_{Mem} = 20ns の SRAM が実際に存在し $MAC_i * rl_{Mem}$=40 ～ 80ns となる．このことから，$MAC_i * rl_{Mem}$ と rl_{FADD} のどちらを支配的にするかは，速度だけでなくハードウェア・コストも考慮して決定しなければならない．

4.3 メモリ・マップ

図 10に命令メモリのプログラム配置，データ・メモリのデータ配置および MP のデータ配置を示す．命令メモリの opI, J, K 部分は $LABEL$ に相当し，$LABEL$ 生成部分を実行することで生成される．

命令長 8 バイト[2] ，$NN = O(10^4)$，$o, p \ll NN$ とすると，命令メモリの容量は $(o+p+NN+2+1)$*(命令長) = $O(100)$K バイトになる．

データ・メモリの容量は基底関数の数 $n = 1000$，データ長 8 バイトとすると $8 * (O(n) + 2 + 100 + 10 + 10^4 + n)$=$O(100)$K バイト程度になる．

上記から各 PE 当りのローカル・メモリは命令，データ合わせて 1MB あれば十分であるので SRAM を使用する．

一方，MP の容量は，基底関数の数 $n = 1000$，データ長 8 バイトとすると $8 * (n^2)$=$O(10)$M バイト程度と大容量になる．そこで，MP にはそれほど高速な応答速度が要求されないことから大容量の DRAM を使用する．

5 おわりに

以上，我々が開発中の MOE のアーキテクチャについて述べた．現在，本稿で述べた PE アーキテクチャに基づいて，これをメモリ混載型 ASIC で実現することを検討している．また，実用レベル規模の問題サイズにおける本並列計算アルゴリズムの動的挙動を解析して，その結果を MOE アーキテクチャに反映することを目的として，MOE プロトタイプを作製している．本 MOE プロトタイプは，100 台規模の PC クラスタとなる予定である．

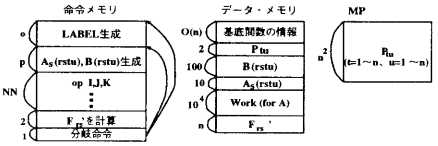

図 10: メモリ・マップ

謝辞 本研究の一部は，科学技術振興調整費「第一原理計算手法に基づくシミュレーション技術に関する研究」，産業技術情報基盤研究開発「ヘテロジニアスネットワークコンピューティング環境における分子シミュレーションシステムの構築」，提案公募型・最先端分野研究開発「分子集合体の構造：ab initio 分子動力学を用いた高速プログラムの開発」，文部省科学研究費試験研究 B「非経験的分子軌道計算の超高速化に関するハードウェア及びソフトウェアの開発」に基づくものである．

参考文献

[1] 長嶋雲兵，小原 繁，村上和彰，吉井 卓，白川 暁，網崎孝志，北村一泰，高島 一，田辺和俊，“超高速分子軌道計算専用機 MOE の開発”，情処研報，ARC-117-16，pp.89-94，1996 年 3 月．

[2] S. Obara and A. Saika, “General recurrence formulas for molecular integrals over Cartesian Gaussian functions”, J.Chem.Phys. Vol.98 NO.3, Aug 1988.

[3] 小原繁，“分子積分と電子相関”，第 32 回分子科学 夏の学校，1992 年．

[4] 兵頭志明，福田敦勇，“分子軌道法”，豊田中央研究所 R&D レビュー Vol.21 NO. 3/4，1987 年 1 月．

[2] 命令長は log(op 数)+log(データ・メモリの語数)*3 ビット．op 数=4，データ・メモリの語数=$O(10K)$ なので，命令長=log 4+log$(10K)$*3= 約 44 ビットとなる．ここでは余裕を持たせて命令長 8 バイトとした．

盛光印刷所