Vol. 44 No. SIG 1(HPS 6) 情報処理学会論文誌：ハイパフォーマンスコンピューティングシステム Jan. 2003

二電子積分計算専用プロセッサ・アーキテクチャの開発

原田宗幸[†1] 中村健太[†1] 桑山庸史[†1]
上原正光[†2] 佐藤比佐夫[†2] 小原繁[†3]
本田宏明[†3] 長嶋雲兵[†4]
稲富雄一[†5] 村上和彰[†1]

筆者らは，非経験的分子軌道計算を高速に処理することを目的とする，二電子積分計算専用プロセッサの開発を行っている．二電子積分計算のアルゴリズムとして採用する小原のアルゴリズムの特徴を活かすことで高速処理が可能となる．本稿では，二電子積分計算専用プロセッサ・アーキテクチャの概要と全体構成について述べ，そのうち漸化計算エンジンの構成法の検討と性能の評価を行う．

Processor Architecture for Molecular Orbital Calculation

Muneyuki Harada,[†1] Kenta Nakamura,[†1] Youji Kuwayama,[†1]
Masamitsu Uehara,[†2] Hisao Sato,[†2] Shigeru Obara,[†3]
Hiroaki Honda,[†3] Unpei Nagashima,[†4] Yuichi Inadomi[†5]
and Kazuaki Murakami[†1]

We are developing a custom processor for *ab initio* Molecular Orbital Calculation to reduce the calculation time. Using characterization of two-electron integrals in the "Obara method," it is possible to reduce the calculation time. This paper describes an outline of processor architecture for Molecular Orbital Calculation. We propose and evaluate organizing method for Recursive Function Engine which a part of custom processor.

1. はじめに

非経験的分子軌道法は，一番近似の粗いハートリー・フォック法を用いた場合でも，用いる基底関数の 4 乗に比例する演算量と補助記憶量を必要とする[1)]．そこで，我々「科学技術計算専用ロジック組込型プラットフォーム・アーキテクチャに関する研究」グループでは，非経験的分子軌道法による分子軌道計算を高速に実行する専用計算機システムの開発に現在取り組んでいる[2)]．

非経験的分子軌道計算において最も時間を要するのが，二電子積分を計算し，フォック行列を生成する部分であるため，大規模計算を高速化する際にはこの部分の並列化，高速化が重要である．現在アーキテクチャの検討を行っている二電子積分計算専用プロセッサでは，非経験的分子軌道法の全計算時間の 95%以上を占める，この二電子積分計算を高速に処理することを目的とする．

二電子積分計算は小原のアルゴリズム[3)]を適用することにより漸化計算で表され，複数の積和演算を並列的に計算することにより高速化できる．この小原のアルゴリズムは大きく分けて，命令レベル並列性が少なく，浮動小数点除算，開平逆数演算，指数関数演算がクリティカル・パスとなっている初期積分計算部分と，多くの命令レベル並列性を持つ漸化計算部分との 2 つの部分に分けられる[4)]．専用プロセッサは，この性質の大きく異なる両方の部分に最適化され，その両方を高速に処理できることが求められる．

そこで本稿では，小原のアルゴリズムの特徴に合わせて，初期積分計算部分を専門に処理する初期積分計算エンジンと，漸化計算部分を専門に処理する漸化計

†1 九州大学
Kyushu University
†2 セイコーエプソン株式会社
Seiko Epson Corporation
†3 北海道教育大学
Hokkaido University of Education
†4 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology
†5 株式会社富士総合研究所
Fuji Research Institute Corporation

算エンジンの 2 つのエンジンから成るプロセッサ・アーキテクチャを提唱し，その上で各エンジンの構成法についての検討を行う．以下，2 章では非経験的分子軌道計算と小原のアルゴリズムの特徴について述べる．次に 3 章では 2 つのエンジンから成る専用プロセッサ・アーキテクチャの概要について述べる．4 章では漸化計算エンジンの構成の検討とその評価を行う．最後に，5 章で本稿をまとめる．

2. 二電子積分計算アルゴリズム

2.1 非経験的分子軌道法

非経験的分子軌道法の解法には，ハートリー・フォック法（HF 法）を用いる[5]．HF 法は式 (2) よりフォック行列（二電子積分寄与部分）を求め，式 (1) よりエネルギー E などを求める．式 (2) の P_{KL} を密度行列，(IJ,KL) を二電子積分と呼ぶ．

$$\sum_{I=1}^{N} F_{IJ}C_{aI} = \sum_{I=1}^{N} S_{IJ}C_{aI}\varepsilon_a \qquad (1)$$

$$F'_{IJ} = \sum_{K=1}^{N}\sum_{L=1}^{N} P_{KL}\{(IJ,KL) - \frac{1}{2}(IK,JL)\} \qquad (2)$$

実際の計算手順を**図 1** に示す．図 1 に示すように，式 (1) の係数行列 C_{aI} が収束するまで手順の (3)〜(5) を繰り返す．**表 1** に HF 法の各部分の計算にかかる時間を示す．表 1 を見て分かるように，二電子積分計算が全体の計算時間の 96%以上を占めている．したがって，非経験的分子軌道法の高速化を実現するには，二電子積分計算をいかに高速に処理するかが重要となる．

2.2 小原のアルゴリズムの特徴

二電子積分計算のアルゴリズムはいくつか知られているが，本プロジェクトでは小原のアルゴリズム[3]を採用する．小原のアルゴリズムの特徴は，まず入力データからいくつかの初期パラメータを計算し，積分タイプ☆(ss,ss) 型の二電子積分（初期積分）を求めたあと，それらを用いて目的の二電子積分まで漸化計算により求める点にある．したがって，小原のアルゴリズムは**図 2** に示すように，初期積分計算部分と漸化計算部分の 2 つに大きく分けられる．

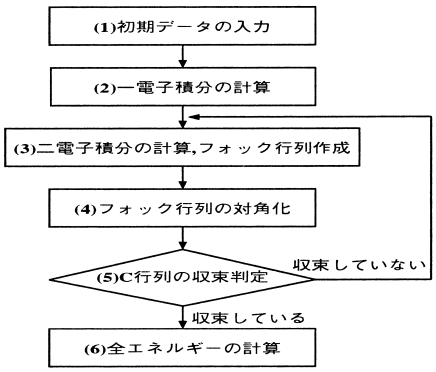

図 1 非経験的分子軌道法の計算手順
Fig. 1 Procedure of *ab initio* Molecular Orbital Calculation.

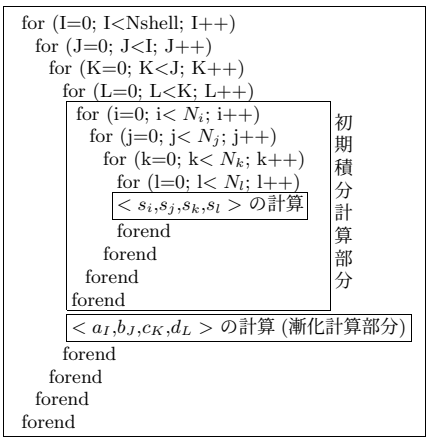

図 2 小原のアルゴリズムのプログラム構造
Fig. 2 Loop structure of the Obara Algorithm.

2.2.1 初期積分計算部分

初期積分計算は図 2 に示すように 4 重のループ構造となる．このループ 1 回あたりの演算数は非常に少なく，命令レベル並列性は低い．さらに，浮動小数点除算，開平逆数演算，指数関数演算という複雑な算術演算を含んでおり，かつそれらの演算がクリティカル・パス上に存在する．したがって，初期積分計算を高速に処理するには，これらの複雑な演算を高速に処理することが重要となる[4]．

2.2.2 漸化計算部分

漸化計算部分は図 2 から見てとれるように，初期積分計算部分が完了したのち，初期積分の計算結果をもとにして最終的に求める二電子積分の値を計算するステップである．そして漸化計算部分は**図 3** に示すように，漸化計算関数（ERI_recursive 関数）と水平移行

☆ 二電子積分を決定する 4 つのガウス型関数それぞれの軌道量子ベクトルを a，b，c，d とする．ある軌道量子ベクトル μ において，ベクトル 3 成分（x 成分，y 成分，z 成分）の和を軌道量子数と呼ぶ．また，軌道量子数が 0，1，2，$\cdots$，の軌道をそれぞれ s 軌道，p 軌道，d 軌道，$\cdots$，と呼ぶ．二電子積分計算はその計算を決定する 4 つの軌道量子ベクトル a，b，c，d ごとの軌道量子数が表す軌道を用いて (pp,ss)，(dd,ss) のように表記される．このように軌道量子数により識別した積分のことを積分タイプと呼ぶ[4]．

表 1 非経験的分子軌道法の計算時間
Table 1 Computation time for some peptide molecules.

ペプチド分子		G	GA	GAQ	GAQM	GAQMY
原子数		10	20	37	58	75
基底関数の数		55	110	207	316	427
計算時間 (s)	初期データの入力	0.1	0.6	4.4	18.9	57.3
	一電子積分の計算	0.1	0.3	1.5	5.0	10.1
	二電子積分の計算，フォック行列作成	**22.9 (96.6%)**	**269.4 (98.7%)**	**1871.0 (98.6%)**	**8482.1 (98.8%)**	**23284.3 (98.6%)**
	フォック行列の対角化	0.2	1.7	11.0	60.9	211.7
	全エネルギーの計算	0.1	0.3	2.2	9.15	27.5
	Total	23.7	272.9	1892.7	8584.9	23614.5

4-31G Basis, using GAMESS, PentiumIII 500 MHz, 512 MB

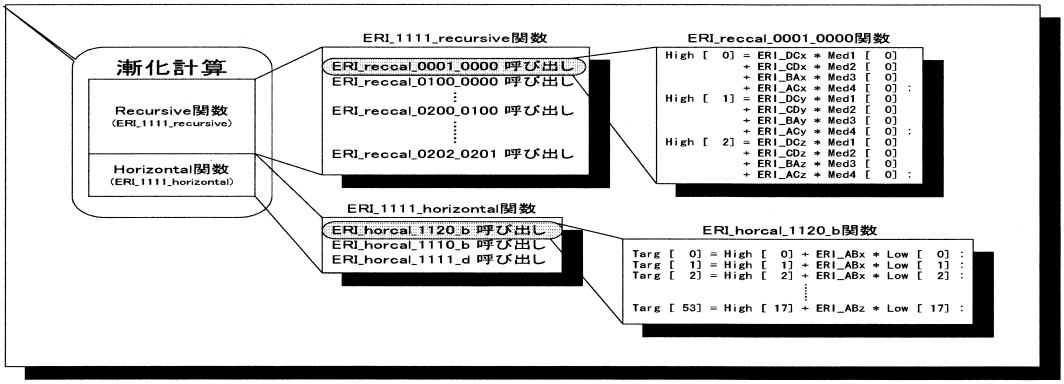

図 3 漸化計算部分の例（積分タイプ (pp,pp)）
Fig. 3 Hierarchical structure of a reccurrence calculation.

関数（ERI_horizontal 関数）から成る．またそれぞれの関数が積和演算の集合である ERI_reccal 関数または，ERI_horcal 関数を呼び出す階層構造を持っている．ただし，積分タイプの種類によっては ERI_horizontal 関数の呼び出しがないものもある．また，積分タイプ (ss,ss) については，初期積分計算で積分タイプ (ss,ss) を計算しているので，漸化計算はない．なお，以下において ERI_reccal 関数と ERI_horcal 関数をサブ関数と呼ぶことにする．漸化計算部分は，サブ関数内における並列度が高く，サブ関数間においても並列度が高い．したがって，漸化計算を高速に処理するにはこれらの並列性を活かすことが重要となる．

3. 専用プロセッサ・アーキテクチャの概要

3.1 検 討 課 題

文献 4) の中で行った評価では，専用プロセッサが初期積分計算部分と漸化計算部分の両方を単一の VLIW（Very Long Instruction Word）プロセッサで処理することを計画していた．そのうえで，

- 浮動小数点除算，開平逆数演算，指数関数演算を高速に処理する専用演算器を設けることで初期積分計算部分を高速化する，
- 積和演算器を多数搭載し，内在する並列性を活用することで漸化計算部分を高速化する，

という方針に基づき，専用演算器と多数の積和演算器，および，すべての演算器に共有されたマルチポート・レジスタファイルから成る VLIW アーキテクチャのプロセッサモデルについての評価を行っている．

しかしながら，2.2 節で述べたように，対象とする二電子積分計算では初期積分計算と漸化計算の性質が大きく異なり，特に 2.2.1 項で述べたように初期積分計算部分の命令レベル並列性が低いため，演算器を多数搭載しても十分な性能向上が得られない．また，単一の VLIW プロセッサで処理するには以下のような VLIW 特有の問題がある．

- レジスタファイルに非常に多くのポートが必要となり，マルチポート・レジスタファイルの面積および遅延により専用プロセッサの性能の上限が抑えられる．
- 命令語長が長いことから命令供給が問題となる．
- コードサイズが大きくなることから，内部メモリに命令を持たせるのに適していない．

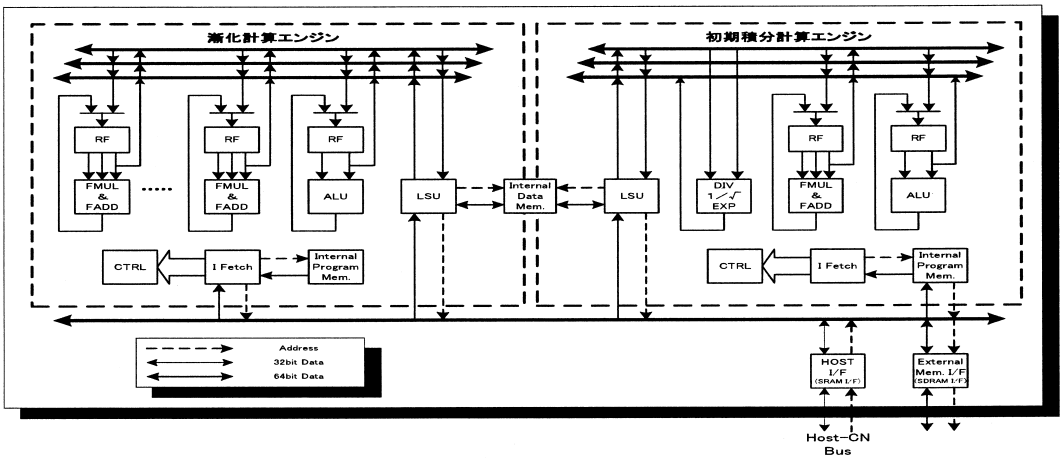

図 4 専用プロセッサ・アーキテクチャのブロック図
Fig. 4 Overview of a custom processor architecture.

これらの問題を解消し，目標とする高い処理能力を得ることができるプロセッサ・アーキテクチャの検討を行った．

3.2 設 計 方 針

専用プロセッサが汎用プロセッサと違う点はいうまでもなく，処理する対象が限定されており，プロセッサ・アーキテクチャを計算対象の特徴に合わせて最適化できる点である．

2 章で述べたように，小原のアルゴリズムは初期積分計算部分と漸化計算部分という 2 つの部分に分けることができる．専用プロセッサが二電子積分計算を高速に処理するために

- 初期積分計算部分には専用演算器を用いる，
- 漸化計算部分は内在する並列性を活用する，

という基本方針には変わりない．

しかし，単一の VLIW プロセッサで処理を行う場合には，3.1 節で述べたような問題があり十分な性能向上が望めない．そこで，専用プロセッサのアーキテクチャを**図 4** に示すように，初期積分計算エンジンと漸化計算エンジンという 2 つのエンジンに分割したチップ・マルチプロセッサで構成とし，初期積分計算部分と漸化計算部分をそれぞれ専用エンジンで実行することで 2 つの計算を同時にオーバラップさせて実行することを検討している．

3.3 全 体 構 成

現在検討中のアーキテクチャは大きく分けて

- 初期積分計算エンジン（IIC Engine）
- 漸化計算エンジン（RC Engine）
- 内部データメモリ（Internal Data Mem.）
- 対ホスト間インタフェース（HOST I/F）
- 対メモリ間インタフェース（External Mem. I/F）

から成る．対象とする演算は倍精度の浮動小数点とし，各演算器，メモリ，インタフェースもそれぞれ 64 ビット幅となる．また単精度の浮動小数点はサポートしない．

3.3.1 初期積分計算エンジン（IIC Engine）

小原のアルゴリズムの初期計算部分を専用に処理する計算エンジンであり，

- 浮動小数点除算，開平逆数演算，指数関数演算を高速に処理する専用演算器（DIV，$1/\sqrt{\ }$，exp）
- 浮動小数点積和演算器（FMUL & FADD）
- ロード・ストアユニット（LSU）
- 内部プログラムメモリ（Internal Program Mem.）

などから成る．

3.3.2 漸化計算エンジン

漸化計算部分を専用に処理する計算エンジンであり，

- 複数の浮動小数点積和演算器（FMUL & FADD）
- ロード・ストアユニット（LSU）
- 内部プログラムメモリ（Internal Program Mem.）

などから成る．なお，漸化計算エンジンに関しては，漸化計算の並列性を活かすために VLIW アーキテクチャを検討している．

3.3.3 内部データメモリ

初期積分計算および漸化計算で必要となるデータを保持するデータメモリである．2 バンク構成となっており，初期積分計算エンジンと漸化計算エンジン間で共有される．また，ダブルバッファ処理を行い，両エンジン間のデータの授受を行う役割も兼ねる．

3.3.4 動 作 原 理

図 2 から見てとれるように，小原のアルゴリズムで

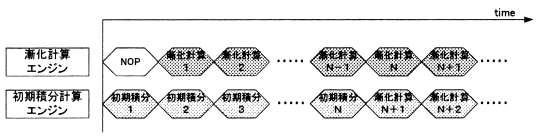

図 5 初期積分計算部と漸化計算部の実行の様子
Fig. 5 Execution flow of IIC engine and RC engine.

は初期積分計算が完了したのち漸化計算が行われる．また，それぞれの二電子積分計算間でのデータ依存は存在しないことから，個々の二電子積分計算は並列に実行可能である．この特徴を活かして，専用プロセッサは**図 5** のように初期積分計算と漸化計算をオーバラップさせて処理する．しかし，初期積分計算と漸化計算の実行時間は積分タイプにより異なる．d 軌道や p 軌道を多く含む積分タイプでは漸化計算の方が実行時間が長く，それ以外の積分タイプでは，初期積分計算の方が実行時間が長い．したがって，先に処理を終了したエンジンがもう一方のエンジンが終了するのを待つことになる．

4. 漸化計算エンジンにおける計算手法の検討

4.1 未解決の課題

3 章では，チップ・マルチプロセッサ構成を採用した小原のアルゴリズムの特徴を活かすプロセッサ・アーキテクチャの全体構成について述べた．しかしながら，漸化計算エンジンは漸化計算の並列性を活かすために VLIW プロセッサとした．したがって，漸化計算エンジンにおいては，3.1 節で述べた問題点のうち，VLIW プロセッサ特有の課題は依然として残っている．従来の手法における漸化計算部分の処理は，多数のサブ関数呼び出しから成る漸化計算をインライン展開し，命令レベル並列性を高めることにより高速化を図っていた．この手法では命令レベル並列性の高い，d 軌道を含むような積分タイプを処理する場合には高い処理能力を得ることができる．一方で漸化計算部分が持つ命令レベル並列性は積分タイプによって異なり，s 軌道と p 軌道のみから成る命令レベル並列性の低い積分タイプを処理する場合には演算器を十分活用できない場合がでてくる．また，命令供給やコードサイズの問題という VLIW プロセッサ特有の課題をかかえている．

4.2 新手法の提案

漸化計算部分は図 3 に示したように，関数呼び出しの階層構造を持つ．4.1 節で述べた課題を解決するために，この特徴を利用し，マイクロ命令を使ったコード圧縮を行う手法を提案する．また，その手法が活用する並列性の違いにより以下の 3 手法に分類し，評価を行った．

手法 FS 命令レベル並列性のみ
手法 FP 関数間 + 命令レベル並列性
手法 SP ステートメント間 + 命令レベル並列性

4.2.1 サブ関数間逐次処理（手法 FS）

漸化計算部分ではサブ関数を繰り返し呼び出す構造をしている．このサブ関数の種類は ERI_reccal 関数が 26 種類，ERI_horcal 関数も 25 種類程度と多くないことから，サブ関数内部の処理をマイクロ命令化し内部メモリに保持することが可能である．そのうえで，マイクロ命令化されたサブ関数を呼び出す漸化計算関数と水平移行関数のみをマクロ命令とし，外部メモリから逐次的に供給することで，先述した命令供給とコードサイズの課題を解消できる．これを “手法 FS” と呼ぶ．

本手法では，1 つのサブ関数内の命令レベル並列性のみしか活用しないことから，演算器の利用効率が上がらないという，従来の手法からの問題点が残る．また，マルチポート・レジスタファイルがかかえる問題も未解決である．しかし，命令コードサイズの問題はこの手法では解決できていない．

4.2.2 サブ関数間並列処理（手法 FP）

漸化計算部分に内在する並列性は命令レベル並列性ばかりではない．サブ関数間にもデータ依存関係がなく，並列実行可能なものが多数存在する．このサブ関数間並列性を活用するのが “手法 FP” である．

手法 FP では関数呼び出しを行うマクロ命令を並列に処理し，対応するマイクロ命令を各サブエンジンに処理させる．つまり，漸化計算エンジン内においても，チップ・マルチプロセッサ構成を採用する手法である．ここで，サブエンジンとは 1 個以上の積和演算器と，1 個以上のロード・ストアユニット，およびそれらのユニットのみからアクセスできるレジスタファイルから構成される実行単位である．

この手法では，1 つのサブ関数を割り当てる演算器の個数が手法 FS よりも少なくなるため，演算器の利用効率が向上することが期待できる．また，サブエンジン 1 個あたりのレジスタファイルのポート数も手法 FS と比べ削減することができる．また，手法 FS では未解決であった命令コードサイズの問題も解決できる．

4.2.3 ステートメント間並列処理（手法 SP）

ステートメントとは，**図 6** に示すように，ERI_reccal 関数においては同じ $Med1[i]$〜$Med4[i]$ を参照する一連の計算であり，ERI_horcal 関数においては同じ $Low[j]$ を参照する一連の計算である．なお，その際に reccal 関数における Low 配列と horcal 関数における $High$

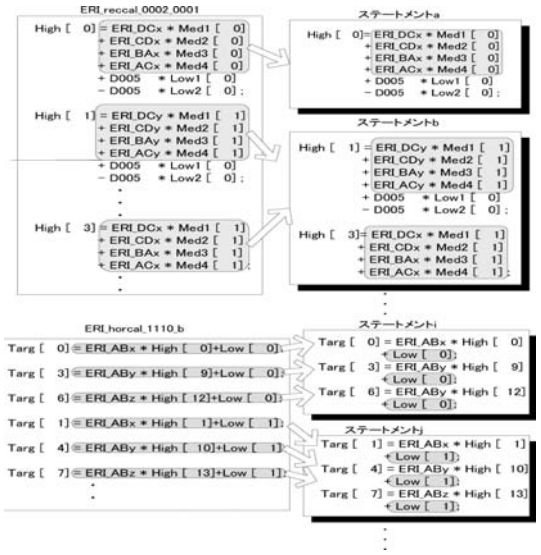

図 6 ステートメント
Fig. 6 Statement.

配列は，各ステートメントごとにロードする．

手法 FP ではサブ関数間の並列性を利用したが，サブ関数は多数のステートメントの集合からなり，そのステートメント間にも並列性が存在する．そこで，ステートメントの出現パターンから規則性を抽出し，19 通りにグループ分けしたものをマイクロ命令とする手法が “手法 SP” である．この手法も FP と同様に，チップ・マルチプロセッサ構成を採用する手法である．

したがって，SP も FP と同様に演算器の利用効率向上，サブエンジン 1 個あたりのレジスタファイルのポート数削減や命令コードサイズ問題の解決が期待できる．さらに FP と比べ，マイクロ命令 1 個あたりの命令サイズが小さく，種類も少ないため，マイクロ命令を保持するのに必要な内部プログラムメモリサイズを小さくすることができる．

しかし，異なるステートメントにおいて参照される reccal 関数における $Low[l]$ および horcal 関数における $High[h]$ の値を破棄するために，データの再利用が行われない．その結果，$Low[l]$ や $High[h]$ の値のロードが FS や FP は 1 回であるのに対して SP では複数回行う必要がある．つまり，FS や FP に比べロードの回数が増加する．

4.3 各手法の性能比較評価

前節で述べた各手法に対しての評価を行う．また，その結果より，漸化計算エンジンの最適な構成法を考察する．なお，サブ関数間やステートメント間の依存関係，並列度は分析可能であるが，今回は，利用する並列度とサブエンジン構成との関連性を調査し，その結果を実際の LSI 開発での指標として利用することを見据えて，シミュレーションで性能評価を行った．

表 2 各機能ユニットのレイテンシ
Table 2 Latency of each functional unit.

	M&A	LSU
Issue Latency	1	1
Result Latency	6	5

表 3 機能ユニットの個数
Table 3 Evaluation models.

Model	Sub Engine	SubEngine 1 個あたりの個数		Total	
		M&A	LSU	M&A	LSU
並列性の違いによる評価					
$FS1,6,6$	1	6	6	6	6
$FP2,3,3$	2	3	3		
$FP3,2,2$	3	2	2		
$FP6,1,1$	6	1	1		
$SP2,3,3$	2	3	3		
$SP3,2,2$	3	2	2		
$SP6,1,1$	6	1	1		
FP におけるサブエンジン構成の違いによる評価					
$FP2,1,1$	2	1	1	2	2
$FP2,2,1$	2	2	1	4	2
$FP2,3,1$	2	3	1	6	2
$FP3,1,1$	3	1	1	3	3
$FP4,1,1$	4	1	1	4	4
$FP6,1,1$	6	1	1	6	6

4.3.1 評価方法

評価方法に関しては，漸化計算部分の実行サイクル数が最小となるように，クリティカル・パス法を拡張したリスト・スケジューリング[6] を行い，その結果による実行サイクル数を求めた．なお，レジスタ数は各サブ関数の処理に必要なデータをすべて保持するのに十分な数を準備すると仮定して評価を行った．また，メモリは各サブエンジンおよび IIC エンジンより毎サイクル，同時アクセス可能と仮定して評価を行った．

4.3.2 評価モデル

評価を行う漸化計算エンジンのモデルとして，**表 2** に示すようなレイテンシを持つ積和演算器（M&A）と，ロード・ストアユニット（LSU），およびレジスタファイルから成る計算エンジンについて考える．先に述べた手法について**表 3** に示す数の機能ユニットを有する各モデルについて評価する．ただし，表中の Model は先頭の要素から，手法名，サブエンジン数，1 クラスタあたりの積和演算器数とロード・ストアユニット数を表すものとする．なお，今回製作する LSI の面積を考慮した際に，積和演算器数が最大でも 6 個程度までしか載せることができないという制約条件が存在するため，今回の評価でも最大 6 個までのモデル

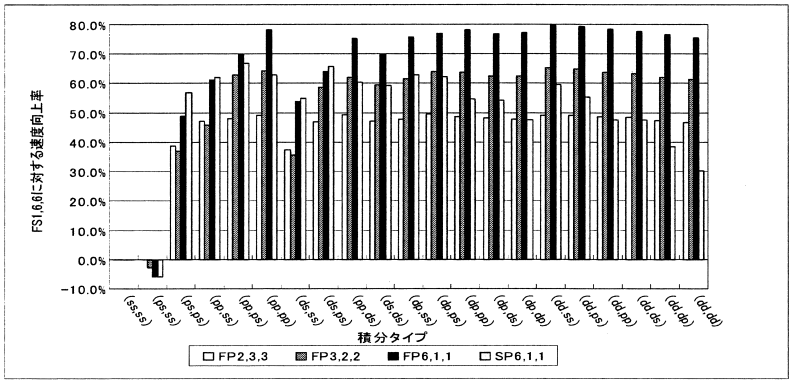

図 7 活用する並列性の違いによる速度向上率比較
Fig. 7 Performance comparison among FS, FP, and SP.

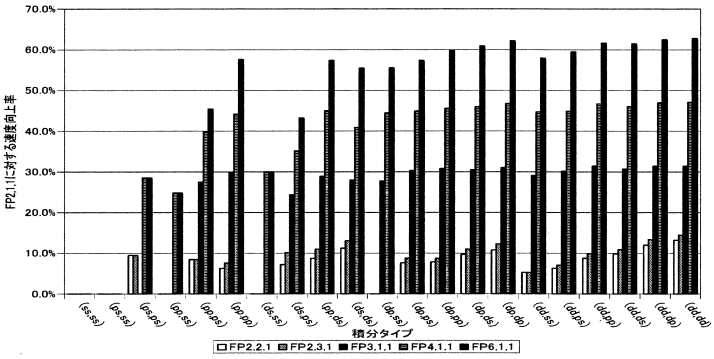

図 8 サブエンジン構成の違いによるの速度向上率比較
Fig. 8 Performance comparison among different organization of FP.

で行った．

4.3.3 評価結果および考察

4.3.3.1 並列性の違いによる評価

総積和演算器の個数および，総ロード・ストアユニットの個数をそれぞれ 6 に固定したときの，すべての手法におけるすべての構成について，性能の違いを評価したのが**図 7** である．ただし手法 SP については，1 サブエンジンあたりの積和演算器の個数が増加しても性能向上が得られなかったことから，$SP6,1,1$ の場合のみグラフに示した．また，グラフは $FS1,6,6$ について正規化しており，$FS1,6,6$ に対する，他の手法の性能向上率を示している．

グラフが示しているように，d 軌道や p 軌道を多く含む積分タイプにおいては，$FP6,1,1$ が最も高い性能向上比を示している．また，実際のクロックサイクル数の差は $SP6,1,1$ に比べ最大で約 50000 サイクルもある．

一方，d 軌道や p 軌道をあまり含まない積分タイプにおいては，$SP6,1,1$ が $FP6,1,1$ よりも若干高い性能向上比を示している．しかし，$SP6,1,1$ と $FP6,1,1$ のクロックサイクル数の差は最大でも 14 サイクルしかない．

これは，SP は FP に比べ，サブ関数内の並列性を活かす手法だが，その性能向上よりもロード回数の増加による性能低下が大きかったといえる．

また，手法 FP についてのみ着目すると，ほとんどの積分タイプで $FP2,3,3$ よりも $FP3,2,2$，$FP3,2,2$ よりも $FP6,1,1$ が性能向上比は高い．これは，総機能ユニット数が同じ場合ならば，サブエンジン 1 個あたりの機能ユニット数を増やして，サブ関数の処理の高速化を図るよりも，サブエンジン数を増やして，サブ関数間の並列性を活かす方が性能向上につながるといえる．

4.3.3.2 手法 FP におけるサブエンジン構成の違いによる評価

手法 FP について $FP2,1,1$ をベースとした場合に，積和演算器を増加させた $FP2,2,1$，$FP2,3,1$ の構成と，サブエンジン数を増加させた $FP3,1,1$,

表 4 内部プログラムメモリサイズ比較
Table 4 Comparison of internal program memory size.

	$FS1,6,6$ に対するメモリサイズ削減率
$FP1,6,6$	0%
$FP2,3,3$	−78.4%
$FP3,2,2$	−158.5%
$FP6,1,1$	−401.2%
$SP6,1,1$	81.3%

$FP4,1,1$，$FP6,1,1$ の構成を比較したのが**図 8** である．

このグラフが示しているように，6 タイプの中では $FP6,1,1$ が高い性能を示している．また，ほとんどの積分タイプにおいて $FP2,2,1$ や $FP2,3,1$ のようにサブエンジン 1 個あたりの積和演算器の数を増やすモデルに比べ，$FP3,1,1$，$FP4,1,1$，$FP6,1,1$ のようにサブエンジンの構成は変えずにサブエンジンの数を増やすモデルの方がより高い性能を示している．また，サブエンジンの数は増やせば増やすほど性能向上が見られる．これは，4.3.3.1 と同様に，サブエンジン 1 個あたりの機能ユニット数を増やして，サブ関数の処理の高速化を図るよりも，サブエンジン数を増やして，サブ関数間の並列性を活かす方が性能向上につながるといえる．

4.3.3.3 内部プログラムメモリサイズ

$FS1,6,6$ に対する各手法の内部プログラムメモリサイズを比較したのが**表 4** である．この表より，$SP6,1,1$ は，FS よりも必要なメモリサイズが 81.3%も減少している．一方，手法 FP は $FP1,6,6$，$FP2,3,3$，$FP3,2,2$，$FP6,1,1$ の順に必要なメモリサイズが増加している．$FP6,1,1$ においては，400%以上も増加している．したがって，内部プログラムメモリサイズの点では SP が他の手法に比べ圧倒的に優れているといえる．

4.3.3.4 考　　察

d 軌道や p 軌道を多く含む積分タイプの計算においては手法 FP が他の手法と比べかなり優れているが，それ以外の場合は若干手法 SP が優れているといえる．また，内部プログラムメモリサイズは圧倒的に手法 SP が優れている．

しかし，実際の分子の計算においては，複数の種類の積分タイプを複数回計算することになる．しかも，今後は d 軌道を含む積分タイプの計算をより多く行うことが一般的になってくることが予想される．その場合，漸化計算の方が初期積分計算よりも時間がかかるので，漸化計算をより高速に処理する必要がある．また，手法 FP がメモリサイズは大きくなるものの，LSI に搭載可能な大きさである．したがって，総実行時間を考慮した場合，d 軌道を多く含む積分タイプで圧倒的に速い FP の方が，望ましいといえる．

また FP の構成法としては 4.3.3.1，4.3.3.2 の結果より，最小構成（$M\&A = 1$，$LSU = 1$）のサブエンジンを多数搭載する構成が最適であるといえる．

5. ま と め

以上の検討から，専用プロセッサを初期積分計算エンジンと漸化計算エンジンから成るアーキテクチャとする全体構成を決定した．また，そのうち漸化計算エンジンについては，少数の積和演算器から成るサブエンジンを多数搭載することで漸化計算部分を高速に処理できることが明らかとなり，今後のプロセッサ・アーキテクチャの仕様を決めるうえでの有用な指針を得た．

今後は面積や消費電力といった制約条件をより詳細に見積もったうえで，制約条件の範囲内で最高の性能を得ることができる初期積分計算エンジンの構成，漸化計算エンジンのサブエンジン数の決定などを行う．また，そのほかのメモリ構成やホスト間との連携についても詰めていき，2002 年度中の論理設計の完了を目標に仕様決定を行っていく．

謝辞 本研究は一部，平成 14 年度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型プラットフォーム・アーキテクチャに関する研究」による．

参 考 文 献

1) 高島　一，山田　想，小原　繁，北村一泰，稲畑深二郎，宮川宣明，田辺和俊，長嶋雲兵：分散メモリ型並列計算機に適した新しい大規模フォック行列生成アルゴリズム—積分カットオフとの関連，*J. Chem. Software*, Vol.6, No.3, pp.85–104 (Jan. 2000).
2) 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：科学技術計算専用ロジック組込み型プラットフォーム・アーキテクチャの開発—プロジェクト全体像，HPC82-1 (Aug. 2000).
3) Obara, S. and Saika, A.: General recurrence formulas for molecular integrals over Cartesian Gaussian function, *J. Chem.Phys.*, Vol.98, No.3 (Aug. 1988).
4) 戸川勝巳，小原　繁，上原正光，佐藤比佐夫，波多江秀典，中村健太，村上和彰：新「小原のアルゴリズム」に基づく二電子積分計算専用 LSI について，HPC85-5 (Mar. 2001).
5) 藤永　茂：入門分子軌道法，講談社 (1990).
6) 村上和彰：スーパースカラ・プロセッサの性能

を最大限に引き出すコンパイラ技術，日経エレクトロニクス，No.521, pp.165–185 (1991).

(平成 14 年 6 月 7 日受付)
(平成 14 年 10 月 3 日採録)

原田 宗幸

2002 年九州大学工学部電気情報工学科卒業．現在，同大学大学院システム情報科学府情報理学専攻修士課程に在学中．科学技術計算専用 LSI 開発の研究に従事．

中村 健太

2001 年九州大学工学部電気情報工学科卒業．現在，同大学大学院システム情報科学府情報理学専攻修士課程に在学中．科学技術計算専用 LSI 開発の研究に従事．

桑山 庸史

2002 年九州大学工学部電気情報工学科卒業．現在，同大学大学院システム情報科学府情報理学専攻修士課程に在学中．科学技術計算専用 LSI 開発の研究に従事．

上原 正光

1972 年静岡大学工学部工業化学科卒業．現在，セイコーエプソン研究開発本部コアテクノロジー開発部材料技術開発室室長．半導体材料工学，機能性有機材料，デバイスシミュレータ，材料開発シミュレータの研究開発に従事．

佐藤比佐夫

1974 年名古屋大学理学部第 2 物理学科卒業．現在，セイコーエプソン（株）半導体事業部 IC 設計部所属．システム LSI の開発に従事．

小原 繁

1980 年北海道大学大学院理学研究科博士課程単位取得退学．博士（理学）．現在，北海道教育大学教育学部釧路校助教授．分子の電子状態の量子化学的研究と量子化学計算アルゴリズムの開発に従事．

本田 宏明

2000 年北海道大学大学院理学研究科博士後期課程修了．博士（理学）．現在，富士総合研究所研究員．量子化学計算分野における理論化学計算や高速電子反発積分計算の研究，科学技術計算専用 LSI 用ソフト開発の研究に従事．

長嶋 雲兵（正会員）

1983 年北海道大学大学院理学研究科博士後期課程化学第二専攻修了．理学博士．同年，岡崎国立共同研究機構分子科学研究所電子計算機センター助手．お茶の水女子大学理学部情報科学科助教授，1996 年同教授を経て，1998 年通産省工業技術院物質工学工業技術研究所基礎部理論化学研究室長．1999 年同産業技術融合領域研究所計算科学研究グループ長，2001 年独立行政法人産業技術総合研究所先端情報計算センター情報基盤研究開発室長，2002 年より同グリッド研究センター統括研究員．筑波大学連携大学院大学教授．計算化学，情報化学，大規模数値計算，広域分散並列処理の研究開発に従事．日本化学会，IEEE，応用数理学会，コンピュータ化学会各会員．

稲富 雄一

1998 年筑波大学大学院博士課程化学研究科化学専攻修了．博士（理学）．現在，独立行政法人産業技術総合研究所グリッド研究センター特別研究員．量子化学計算のための計算手法，ならびに高速分子積分計算のプログラムの開発等に従事．

村上 和彰（正会員）

1984 年京都大学大学院工学研究科情報工学専攻修士課程修了．博士（工学）．現在，九州大学大学院システム情報科学研究院情報理学部門教授．九州大学情報基盤センター教授，九州大学システム LSI 研究センター教授併任．高性能科学技術計算，システム LSI 工学，コンピュータアーキテクチャの研究開発に従事．