情報処理学会論文誌　コンピューティングシステム　Vol. 3　No. 3　209–220 (Sep. 2010)

# ストリームアプリケーション向け 大規模FPGAアレイCUBEの性能評価

吉見真聡[†1]　西川由理[†2]　天野英晴[†2]
三木光範[†1]　廣安知之[†1]　オスカー メンサー[†3]

本論文では，512 個の FPGA から構成される 1 次元接続型 FPGA アレイ CUBE の性能評価を目的に，代表的なストリームアルゴリズムである文字列編集距離の並列アルゴリズムを実行するハードウェアを実装し，評価した結果について述べる．評価では，CUBE におけるロジック資源量に加え，スループットおよび電力性能をマイクロプロセッサのマルチスレッド実行，高性能計算分野での普及が期待されている GPU（NVIDIA 社 GeForce GTX280）および Cell/B.E. と比較した．

# Performance Evaluation of One-dimensional FPGA-cluster CUBE for Stream Applications

Masato Yoshimi,[†1] Yuri Nishikawa,[†2]
Hideharu Amano,[†2] Mitsunori Miki,[†1]
Tomoyuki Hiroyasu[†1] and Oskar Mencer[†3]

This paper reports implementation and evaluation of CUBE, which is a multi-FPGA system which can connect 512 FPGAs in a form of a simple one dimensional array. As the system well suits as stream-oriented application platforms, we evaluated its performance by implementing edit distance computation algorithm, which is a typical stream-oriented algorithm. Performances are compared with Cell/B.E., NVIDIA's GeForce GTX280 and a general multi-core microprocessor. The report also discusses performance efficiency, logic consumption and power efficiency with comparison to other multi-core devices.

## 1. はじめに

科学計算に要するデータ量や演算量は現在でも拡大しており，計算システムの規模は大きくなり続けている．大規模な科学計算には PC クラスタが用いられるようになり，導入やメンテナンスにかかるコストは従来のスーパコンピュータに比べて低減しつつあるものの，計算システムの稼働に関わる電力や空調にまつわる運用コストの問題は依然として大きい．近年では高い演算性能を維持したままマシンの筐体サイズと電力消費を大幅に削減しようとする専用ハードウェアを持つ計算システムの研究もさかんである．なかでも FPGA は，実行するアプリケーションにはある程度の制約を受けるものの，専用ハードウェアを柔軟に構成できる性質から，科学計算用のアクセラレータとして可能性が広く検討されてきた[1)–3)]．そのような状況にあって 2008 年に開発された CUBE は，FPGA が 1 次元接続されたアレイ構造を持つハードウェアである[4)]．CUBE は最大 512 個の Spartan 3 で構成される FPGA アレイを持ち，FPGA 間をデータが通過していく際に，シストリックアレイ計算，パイプライン演算により高速化を図る，優れた電力対性能比を示すアクセラレータとして開発が進められている．

従来の ASIC 製造前のプロトタイピングや高速ネットワークスイッチなどの用法に加え，近年の FPGA は，浮動小数点演算を含む一連の科学シミュレータを単一チップ上に構成できる程度の豊富なロジック資源に加え，乗算器や BlockRAM などのハードマクロを内蔵するようになったことにより，それらを利用した，より複雑なアルゴリズムのハードウェア実装も試みられるようになった[5),6)]．また，多数の FPGA からなる計算システムは，現在までに複数開発されてきた．なかでも，CUBE と類似した構造を持つハードウェアとして，Splash 2[7)] や，PROGRAPE[1)] が広く知られている．今後も FPGA を多数使用して科学計算アクセラレータを構築する傾向は続くことが予測されるため，マルチコアプロセッサや GPU などのメニーコアプロセッサと比較して，整数演算，浮動小数点演算の計算性能，電力性能について，指針となる定量的評価が求められている．

そこで本論文では，CUBE のような FPGA 間の接続が限定された複数の FPGA からな

†1 同志社大学
Doshisha University
†2 慶應義塾大学
Keio University
†3 インペリアルカレッジロンドン
Imperial College London

るシステムの有効性を検証するために，まず整数演算主体のストリームアプリケーションである文字列編集距離を計算するアルゴリズムを実装し，その計算性能について議論する．本論文で扱うアルゴリズムは浮動小数点演算を多用するアプリケーションにはあてはまらないが，編集距離計算は入出力データ量に比べて演算量が多く，豊富な並列性を持つため，計算システムの性能評価の目的で広く使用されているアプリケーションである．また，CUBE が主な対象とする，データに対して逐次的に計算が進むアルゴリズムでもある．評価結果は，近年のマルチコアプロセッサである Intel Core 2 Quad Q9550，Cell/B.E.，および GPU を用いたマルチスレッド実行と比較し，計算性能と消費エネルギーの観点から検討を行った．

## 2. CUBE: 512 FPGA cluster

CUBE は，**図 1** にその構造を示すように，8 × 8 の正方形状に配置された 64 チップの FPGA による CUBE ボードが，最大 8 枚重ねられて構成される並列計算プラットフォームである[4]．各 FPGA 間およびボード間は 64 ビットの信号線で接続され，最大 512 チップの 1 次元 FPGA アレイを構成する．CUBE には FPGA として比較的大きなサイズの Spartan 3 が実装されており，ハードウェアの低価格化と豊富なロジック資源の提供の両立を目標としている．

計算は FPGA アレイ内にデータが順に転送される際に行われる．そのため，実行対象のアルゴリズムがパイプライン，シストリックアレイを構成できる場合には，FPGA のロジック資源が有効に利用され，高い性能が期待される．FPGA 間のデータ転送は FIFO を介して 1 方向に固定されており，アレイ内部でデータを戻すことはできない．そのため，複雑なデータのやりとりを必要とし，複数のメモリへの並列アクセスにより高速化するようなアルゴリズムは不向きである．データ通信に制約が加わる一方で，豊富なロジック資源を持つ CUBE は動画像処理や暗号計算のような，巨大なサイズのデータに対してループレベル並列性を持つストリームアプリケーションで高い効果を発揮すると期待されている．

CUBE のような比較的単純な構造の FPGA アレイは以下の 2 点で利点を持つ．

(1)　ワイヤ長の抑制
接続を隣接 FPGA 間に限定することで，物理的な配線距離を短く保ち，遅延時間の増大を防ぐ．また，FIFO による単純なデータ通信により，通信オーバヘッドを小さく抑える．これらの理由から，FPGA 間もチップ内と同じ 100 [MHz] の動作周波数で通信が行われる．FPGA 間のデータレートは 6.4 [Gbps]（= 100 [MHz] × 64 [bit]）となる．

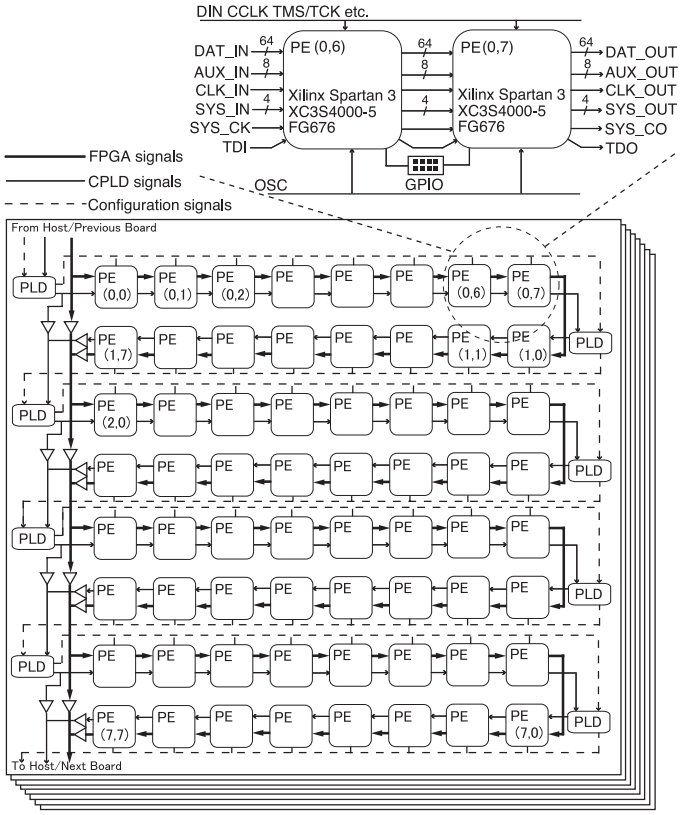


**図 1**　CUBE のアーキテクチャ
Fig. 1　Architecture of CUBE.

(2)　拡張の容易性
FPGA アレイが 1 次元方向にのみに拡張される構造のため，動作周波数やデータ通信網によらないシステムのスケールアップが可能である．

CUBE を用いたアプリケーションとして，1 枚の CUBE ボード（CUBE(1b): 64 FPGA）を用いた $2^{40}$ 文字の文字列探索が報告されている[4]．この例における計算時間は 104 [W] の電力で 1,460 [秒] であり，この時間で計算を完了するには Quad Core Xeon 2.5 [GHz] が 359 CPU 必要になる．このときの CUBE(1b) のエネルギー効率は約 690 倍である．

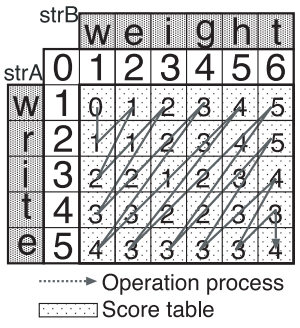


**図 2**　アルゴリズムの概要
Fig. 2　Overview of the algorithm.

## 3. 文字列編集距離

### 3.1 アルゴリズム

編集距離（レーベンシュタイン距離[8]）の計算は，2 つの文字列の類似度を求めるアルゴリズムである．

**図 2** に “write” と “weight” の 2 つの文字列を例としたアルゴリズムの概要を示す．編集距離は，一方の文字列 *strA* に対し，文字の (1) 挿入，(2) 削除，(3) 置換の手順を繰り返し，他方の文字列 *strB* に変換するために要する手順の最小回数のことを指す．この値は，スコアテーブル $c$ の各要素に手順の回数を書き込んでいくことで求められる．文字列長がそれぞれ `lenA` および `lenB` のとき，スコアテーブル $c$ の大きさは $(\texttt{lenA}+1)\times(\texttt{lenB}+1)$ となる．その後，テーブルの要素 $c(i,j)$ の値は，3 つの隣接要素の値 $c(i-1,j-1)$，$c(i,j-1)$，$c(i-1,j)$ を用いて，以下の手順で求められる．

(1)　テーブルの要素 $c(i,0)$ および $c(0,j)$ を，$i$ および $j$ で初期化する．
(2)　*strA* の $i$ 番目の文字と *strB* の $j$ 番目の文字を比較し，等しければ 0，そうでなければ 1 を一時変数 $a$ に代入する．
(3)　$c(i-1,j-1)+a$，$c(i,j-1)+1$，$c(i-1,j)+1$ のうち，最小値を $c(i,j)$ に格納する．

さらに (2)–(3) の手順を繰り返してスコアテーブルを埋めていき，$c(\texttt{lenA},\texttt{lenB})$ の値が 2 つの文字列の編集距離である．

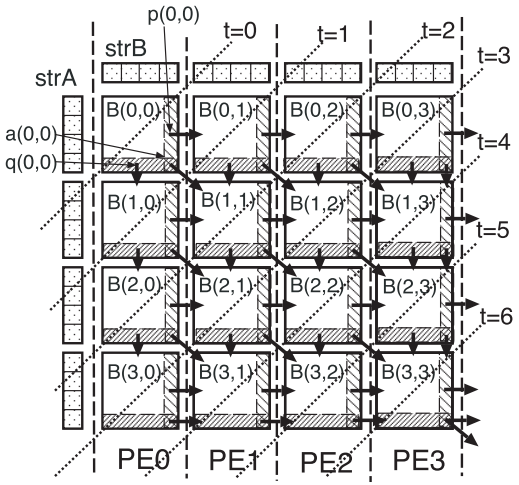


**図 3**　並列アルゴリズムの概要
Fig. 3　Overview of the parallel algorithm.

### 3.2 ブロック分割による並列計算

**図 3** に示すように，編集距離計算のスコアテーブルは，複数のブロックに分割して計算を進めることができる．ブロック $B(i,j)$ は，部分文字列 $strA_i$ と $strB_j$，中間データ $p(i,j-1)$ と $q(i-1,j)$，およびブロックの左上の要素の値 $a(i-1,j-1)$ の 5 つが入力として与えられ，計算の結果である $p(i,j)$，$q(i,j)$，$a(i,j)$ を隣接ブロックの計算の入力として使用する．

このようなブロック分割を行うと，図 3 に示した各破線上のブロックは並列に計算できる．そのため計算資源が複数あれば，それらを使用して高速化を図ることができる．並列計算可能なブロック数は，計算の進展にともなって増減する．この部分文字列どうしのスコアテーブルを計算する一連の処理を，以降 “ブロック計算サイクル” と呼ぶこととする．

## 4. 関 連 研 究

CUBE のような多数の FPGA からなる科学計算向けのアクセラレータは，Splash 2[7] や，多体問題向けシステム PROGRAPE シリーズ[1]，電波天文学やゲノム解析に利用されている FPGA クラスタ BEE2[9] など，複数の開発プロジェクトが進められている．特に Splash 2 は CUBE と類似した計算プラットフォームである．Splash 2 ボードあたり 16 チッ

プの FPGA が 36 ビットの信号線で接続されてリニアシストリックアレイを構成し，さらに各 FPGA がクロスバ接続された構造を持つ．Splash プロジェクトではこのボードが複数接続された計算システムで，多様なアプリケーション実装が報告されてきた．この中には文字列編集距離も含まれており，当時のスーパコンピュータ CRAY-2 と比べて約 330 倍の高速化が確認されている[7]．

また，本論文で実装，評価する文字列編集距離は，画像認識の際のパターンマッチング処理や，遺伝子やアミノ酸配列を対象としたデータマイニング，文字の綴り間違いの検出・修正など，様々なアプリケーションにおける基本的なアルゴリズムとして利用されている．また，様々なレベルで高い並列性を持つアルゴリズムとしても知られており，近年では Cell/B.E. や GPU などのマルチコアプロセッサを対象に，実行速度を競うプログラミングコンテストが開催されている[10),11)]．

編集距離の計算には動的計画法（DP: Dynamic Programming）が用いられ，DP を基礎とする様々なアルゴリズムが開発されてきた．これらの各アルゴリズムについては，枝刈りやビット並列化手法[12),13)] などのアルゴリズム的な改良や，FPGA を用いて複数の比較を同時に行ってスループットを高める工夫[14)] なども試みられている．編集距離計算は広く利用されているアプリケーションであり，問題の大きさや用途，実行環境に応じて様々なチューニング方法が考えられる．たとえば，プログラミングコンテストの結果によると，Cell/B.E. ではビット並列化アルゴリズムの採用や，コア間通信の最適化やループの展開により，枝刈りのできない対象問題においてもサンプルプログラムの最大約 315 倍の高速化が報告されている[15)]．また，GPU を用いた場合でも，対象問題によっては 100 倍を超える高速化がなされている．ビット並列化手法は，スコアテーブルの隣接要素の値の差がたかだか 1 であることを利用してビットレベルの並列性を持つベクトル演算で編集距離計算を実現する手法である．3 章で述べたアルゴリズムでは時間 $O(\mathtt{lenA} \cdot \mathtt{lenB})$ で編集距離が求められるが，$w$ ビットの並列演算が実現できるとすると，ビット並列化手法により計算時間は $O(\lceil \mathtt{lenA}/w \rceil \cdot \mathtt{lenB})$ に短縮できる．Cell/B.E. 向けの実装では SPE が持つ 128 ビットのレジスタを利用し，SPE それぞれで 128 並列で計算を進める方法がとられている．この手法は FPGA でも利用することができ，柔軟なハードウェア構成が可能であることから，他のマルチコア，メニーコアプロセッサと比べて有利に働くと考えられる．このようなチューニング手法は複数のデバイスで共通に使用することができ，それによる計算性能の向上率も同程度であることが見込めるため，高速化において重要と考えられる点は，コア（PE）間のデータ転送の性能およびコア数のような計算資源の量であるといえる．そこで本論文では，演算コアに割り当てる並列アルゴリズムを使用した場合の性能を計測し，特定のアプリケーションやアーキテクチャに固有のチューニングは行わない．FPGA 間の接続が Splash 2 よりも制限された CUBE 上に編集距離計算アルゴリズムを実装し，その性能について近年のマルチコアプロセッサと比較をふまえて議論する．

この評価をもとに，リニアシストリックアレイで計算できるアルゴリズムを複数 FPGA を用いて実行する際の計算性能と消費エネルギーについて検討する．

## 5. 実　　装

### 5.1 CUBE での実装の概要

一定の長さの部分文字列の編集距離を計算するブロック計算モジュールを CUBE 上の各 FPGA 上に実装し，**図 4** の手順で計算が進むシストリックアレイを構成した．1 次元 FPGA アレイである CUBE を用いて計算を行うためには，先頭の FPGA から末尾の FPGA に至るチップ間のデータフローを決定したうえで，各 FPGA 内で行われる計算を定める必要がある．本章では，編集距離計算を FPGA 間，FPGA 内の 2 段階に分け，それぞれについて

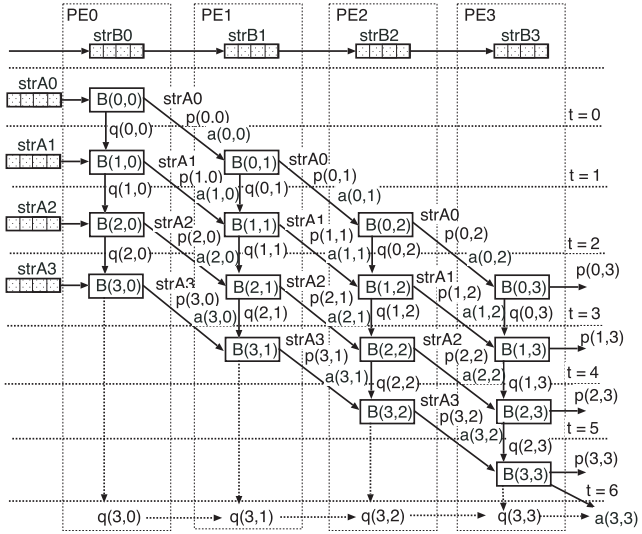


**図 4**　CUBE 上の計算手順とデータフロー
Fig. 4　Computational operation and dataflow on CUBE.

検討する．

### 5.2　FPGA 間のデータフロー

3.2 節で述べたブロック化アルゴリズムは，複数の FPGA に処理を割り当てられる，並列実行に適した問題分割手法である．また，PE 間のデータ転送は 1 方向で，データの出戻りがないストリームアルゴリズムなので，CUBE 上に巨大なシストリックアレイを構成して計算を実行することができる．

図 4 は，4 チップの FPGA で構成される計算システムを用いる場合の FPGA 間のデータフローの例である．FPGA アレイの先頭である “PE0” は，部分文字列 $strB_0$ を保持しており，部分文字列 $strA_0$ から $strA_3$ まで，計算ステージごとに順番に計算が行われる．PE1 以降の FPGA も同様に，第 $i$ 列のブロックの計算を担う．

CUBE の各 FPGA は図 4 のように列方向のブロックの計算を担うハードウェアを構成し．3.2 節で述べた手順における中間データ $p(i,j)$，$a(i,j)$ を隣接 FPGA に転送する．

計算開始時に，部分文字列 $strB_i$ が各 FPGA に転送される．部分文字列長は 1 つの FPGA で計算できる適切なサイズを選択する．部分文字列の数が FPGA 数よりも大きくなってしまう場合は，すべての FPGA を使った一連のブロック計算を複数サイクルに分けて繰り返すことで全体の計算を実現する．

一方，部分文字列の数が FPGA 数よりも少ない場合は，前方に配置されている FPGA で計算が完了し，後方の FPGA は計算結果がバイパスされる．

転送ステージでは，部分文字列 $strA_i$ が CUBE の入力線から入力される．続く計算ステージでは，すべての FPGA が隣接 FPGA や入力線から入力される $strA_i$ と $p_i$ を使用して，ブロックの編集距離計算を行う．各 FPGA は，計算結果 $a_{(i,j)}$ は，それぞれが持っている部分文字列 $strA$ と $p_i$ を隣接 FPGA に転送する．転送データの先頭には次サイクルの計算を指示するフラグが付加され，各 FPGA 間はこのフラグを読み出すことで計算の開始タイミングを知る．転送ステージは必要な時間が固定なので，この方法により各 FPGA で同期をとって計算を進めることができる．末尾の FPGA は，スコア $q_{(3,3)}$ および中間データ $p_{(3,j)}$，$q_{(i,3)}$ を出力線に出力する．中間データは，次のサイクルの計算が必要な場合に使用される．

### 5.3　各 FPGA 内での計算

各 FPGA が行うブロック計算用のハードウェアモジュール LD_thread の構成を**図 5** に示す．各 FPGA は 8 文字の編集距離計算を行う 16 個の LD_unit モジュールを持ち，合計で 128（$= 8 \times 16$）文字を 1 つのブロックとして計算を行う．本実装でのブロック計算サイクルは，128 文字の部分文字列の編集距離を計算する一連の処理のことを指す．CUBE 上

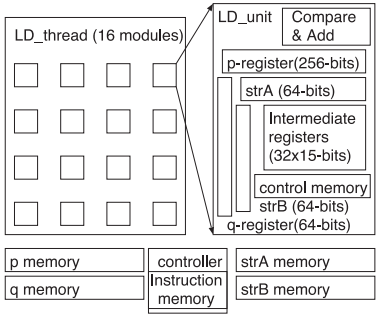


**図 5**　編集距離計算のブロック計算モジュール
Fig. 5　Block computation modules for obtaining edit distance.

の各 FPGA は 100 MHz で動作し，最大 512 個の FPGA を用いて，データが FPGA アレイを通過するまで 64 k 文字の編集距離を計算する．

LD_unit は 8 文字の文字列の編集距離を計算するハードウェアモジュールであり，3 組の比較器と加算器からなる．LD_unit 内部の制御メモリにはマイクロコードが格納されており，計算に使用する部分文字列や中間データを指定するアドレスが記述されている．

LD_unit のコントローラはクロックサイクルごとに制御メモリからマイクロコードを読み出し，適切なデータを要素の計算を行う計算コア部に入力する．CUBE は 100 [MHz] で通信や演算を行うので，遅延の増大を防ぐために計算コア部は，(1) データ読み出し，(2) スコア加算，(3) 比較，(4) データ書き込みの 4 ステージにパイプライン化されている．このとき読み書きされるデータは，中間データ $p$，$q$ および隣接要素のスコアである．LD_unit は $8 \times 8$ のスコアテーブルのブロックを 80 クロックサイクルで計算する．また，文字列や中間データのメモリ位置の指定と読み書きのための 3 クロックサイクルをあわせて，8 文字のブロック計算には 83 クロックサイクル要する．このときスコア計算が 64 回実行されるので，クロックサイクルあたりに実行できるスコアテーブルの要素数（以下，IPC）は $\mathrm{IPC}_u = 64/83 = 0.77$ [Operation/Clock] となる．$\mathrm{IPC}_u$ は LD_unit が 1 クロックサイクルあたりに計算できるスコアテーブルの要素数を示す．同様に，$\mathrm{IPC}_t$ は LD_thread の IPC のことを示す．

LD_thread は 16 モジュールの LD_unit とそれらの制御コントローラを持つハードウェアモジュールである．LD_thread も内部の制御メモリに格納されたマイクロコード（以下，命令）に従う処理が行われる．命令は計算する LD_unit の番号および計算対象となるブロック

位置を指定する．命令が発行された後，計算開始を指示された LD_unit は，計算対象の部分文字列 *strA* と *strB*（各 8 文字），および中間データ $p$，$q$ をメモリから読み出し，計算を実行する．計算の結果は中間データ $p$，$q$ の読み出しアドレスに上書きされる．LD_thread のコントローラは 83 クロックサイクルごとにブロック計算サイクルの命令を発行する．図 3 に示したように，LD_thread において，最初のブロック計算サイクルでは 1 つのブロックのみが計算でき，その結果を使って次のブロック計算サイクルでは 2 つのブロックが並列に計算できる．このように，最初の 15 ブロック計算サイクルは LD_thread 内の LD_unit がすべて使われることはなく，稼働する LD_unit 数がブロック計算サイクルごとに順次増えていく．同様に，LD_thread での計算の最後の 15 ブロック計算サイクルでは稼働する LD_unit 数が減少していく．本実装では，16 モジュールの LD_unit がすべて稼働するブロック計算サイクルが 1 サイクルあるような命令列をコントローラに格納し，LD_thread が 128 文字のブロックを計算することとした．この場合，LD_thread における 128 文字を 1 ブロックとする計算は，8 文字のブロック計算サイクルを 31 回行うことで実行され，$\mathrm{IPC}_t$ は，式 (1) から $\mathrm{IPC}_t = 6.368$ と求められる．

$$\begin{aligned}\mathrm{IPC}_t &= \frac{(\text{スコアテーブルの要素数})}{(\text{計算に要するクロックサイクル数})} \\ &= \frac{128\,[\text{文字}] \times 128\,[\text{文字}]}{83\,[\text{Clock/Block}] \times 31\,[\text{ブロック計算サイクル}]} \\ &= 6.368\,[\text{Operation/Clock}] \end{aligned} \tag{1}$$

このとき $\mathrm{IPC}_u$ の平均は 0.398（$= 6.368/16$）である．

## 6. 評　　価

### 6.1 面積評価

すべてのハードウェアモジュールを Verilog-HDL で実装し，Xilinx 社 ISE10.1i で合成，配置配線を実行した．また，CUBE で実装されている Spartan 3（XC3S4000-5-FG676）を対象 FPGA として選択した．編集距離計算および隣接 FPGA との通信に使用する命令メモリや LD_thread モジュールの $p$-メモリ，$q$-メモリは Spartan 3 の BlockRAM で構成した．

**表 1** に，各モジュールのロジック資源使用量と最大動作周波数を示す．LD_thread が占めるロジック資源の量や動作周波数ともに CUBE の要求仕様を満たしている．

### 6.2 計算性能

CUBE は現在，各 FPGA への回路情報の転送方法が開発中の段階にあるため，100 [MHz] で動作すると仮定して計算性能の評価を行った．

**表 1**　ロジック資源使用量と最大動作周波数
Table 1　Logic utilization and maximum operating frequency.

| | LD_thread | LD_unit | XC3S4000 |
|---|---|---|---|
| Slices | 22,483 | 1,340 | 27,648 |
| FFs | 17,985 | 1,063 | 55,296 |
| LUTs | 42,369 | 2,496 | 55,296 |
| BRAMs | 10 | 1 | 96 |
| Freq. [MHz] | 125.594 | 156.966 | — |

各 FPGA は 128 文字の編集距離を 2,573（$= 83\,[\text{Clock/BlockCycle}] \times 31\,[\text{BlockCycle}]$）クロックサイクルで計算し，ここから生成される 644 [Byte] のデータを 82 クロックサイクルで隣接 FPGA に転送する．このデータ量は図 4 に示されたように PE 間に転送される $p$-メモリ（512 [Byte] = 128 要素 × 4 [Byte]），部分文字列 *strA*（128 [Byte] = 128 文字）のデータに，ヘッダ情報など，制御用の命令を加えたものである．データの送受信はすべての FPGA 間で同時に行われるので，128 文字の計算は，$2{,}573 + 82 = 2{,}655$ クロックサイクルで実行される．

CUBE で編集距離を計算する場合，ブロック数 $n$ が最大 512 までの問題では，図 4 の手順で計算が行われるため，ブロック計算サイクル $2n-1$ 回分の計算時間を要する．$n$ が 512 よりも大きい場合，この計算手順を繰り返すことになる．たとえば，文字列長 128 k（$n = 1{,}024$）の編集距離を求める場合，$(2 \times \mathrm{MaxBlock} - 1) \times (\lceil n/\mathrm{MaxBlock} \rceil)^2 = (2n-1) \times 4 = (2 \times 512 - 1) \times 4 = 4{,}092$ 回分のブロック計算サイクルを要する．このとき，CUBE へ入出力される中間データはホストのメインメモリに格納される．

ホスト PC と CUBE の間では，ブロック計算サイクル（2,655 クロックサイクル）の間に最大で入力 256 [Byte]（= 128 文字 × 2：部分文字列 *strA* と *strB*）と出力 1,024 [Byte]（= 128×2 要素×4 [Byte]：$p$-メモリおよび $q$-メモリの内容）のデータ通信が発生する．ホスト PC-CUBE 間の I/O はユーザが設計できるため，385.687 [Mbps]（((1,024+256) [Byte]×8)/2,655 [ClockCycle] × 100 [MHz]）程度の帯域を持つデータ通信路でホスト PC と CUBE を接続すれば，データ通信がボトルネックになることはない．CUBE を複数回使用する大きな問題の場合は，これに加えて *strB* の部分文字列をあわせて転送する必要があり，ブロック計算サイクル（2,655 クロックサイクル）中の転送データ（入力）が 128 [Byte] 増加する．この場合に要するホスト PC-CUBE 間の転送幅は 424.256 [Mbpz] となる．また，図 4 の手順は複数の編集距離計算をオーバラップさせることで，通信時間を隠蔽できる．

**表 2**　pthread プログラム実行環境
Table 2　Execution environment of pthread program.

| | |
|---|---|
| CPU | Intel Core 2 Quad Q9550 @ 2.83 GHz |
| RAM | 8.0 GByte |
| OS | GNU/Linux 2.6.26-2-amd64 |
| Compiler | gcc-4.1.3(-O3 -lpthread) |

### 6.3 評価環境

文字列編集距離の性能について解析するために，以下の 4 種類の計算システムを使用する．

(1) CUBE に関する 3 種類の計算システム：(a) 単一の Spartan 3，(b) 1 枚の CUBE ボード（CUBE(1b)），(c) 8 枚の CUBE ボードを用いた完全な CUBE システム（CUBE(8b)）で，それぞれが 100 MHz で動作した場合の性能を RTL レベルのシミュレーションで評価した．

(2) Intel Core 2 Quad Q9550：3 章で説明したブロック化アルゴリズムを使用して**表 2** に示す環境で評価した．

(3) Sony ZEGO [16] 搭載 Cell/B.E.：ZEGO は 7 つのコプロセッサ SPE を使用できる．Cell/B.E. での編集距離計算は，プログラミングコンテスト “Cell Challenge 2009” で使用された C 言語のツールキットプログラムで評価した．3 章のブロック計算アルゴリズムを pthread で 7 スレッド生成し，それぞれを SPE に計算させる．SPE 間通信，および SIMD 命令は使用していない．

(4) NVIDIA GeForce GTX280：GPU での編集距離は，プログラミングコンテスト “GPU Challenge 2009” で使用されたツールキットを使用して評価した．

### 6.4 マルチスレッド実行の効果

性能評価に先立ち，PC クラスタなどで最も広く使用されている PC 用マルチコアプロセッサを用いて 3 章のアルゴリズムを実行した場合について性能を評価した．pthread ライブラリを用いてマルチスレッド実行する C 言語プログラムを実装し，表 2 に示す環境で性能を評価を行った．この並列プログラムは，(1) 1 ブロックのサイズ（部分文字列の長さ）および (2) 生成するスレッド数の，2 つのパラメータが実行時に与えられる．また，計算を開始できるブロック番号が格納されるジョブキューが実装されており，生起したスレッドはジョブキューからブロック番号を受け取ってからブロックの計算を開始する．このプログラムについて，生起するスレッド数と 1 度に計算するブロックサイズを変更して計算時間を計測した．

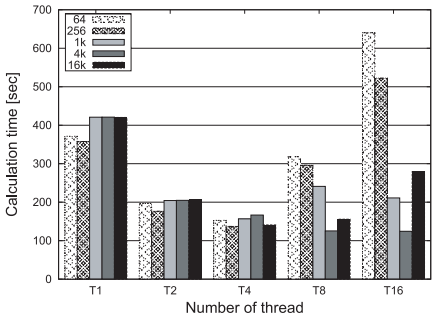


**図 6**　スレッド数とブロックサイズが計算時間に与える影響
Fig. 6　Impact of multithreading and block size on computational time.

まず文字列長を固定して，ブロックサイズとスレッド数が計算時間に与える影響を調べた．**図 6** に 256 k 文字の編集距離の計算時間を，5 種類のブロックサイズで計測した結果を示す．スレッド数が物理コア数よりも少ない T2 以下では，ブロックサイズの影響をあまり受けていない．これは各スレッドが物理コアを占有した状態になっており，プロセスの切替えがほとんど発生しないことが原因であると考えられる．生起スレッドが 1（T1）である場合，スレッド化を行わない逐次型の処理とほぼ同じ性能が得られた．これは，計算を行う生起スレッドに比べ，ジョブキューの処理しか行わない主スレッド処理には計算に時間をほとんど消費せず，スレッド数合計が物理コア数より少ないのでスレッド切替えのオーバヘッドの影響も小さいことが影響していると考えられる．

一方，ブロックサイズが小さいときはスレッド数による性能の変動が特に顕著である．これはブロック計算が短時間で終了して多数の同期処理を頻発させるため，この待ちが性能を悪化させた原因となっていると考えられる．その一方で，同期処理が少ないブロックサイズ 16 k では，8 スレッド（T8）実行よりも 16 スレッド実行の方が性能が悪化している．これは，物理コア数以上のスレッドを生成しているために頻発するスレッドの切替えのオーバヘッドが原因であると考えられる．

ブロックサイズの影響を受けにくく，計算時間が最も短くなるのは，4 スレッド並列実行した場合である．これは実行環境である Core 2 Quad の物理コア数と等しくスレッド切替えが少ないため，計算効率が良いことが原因であると考えられる．

次に，ブロックサイズを 4 k × 4 k 要素に固定して，対象文字列長と計算時間の関係を**図 7** にまとめた．計算時間はアルゴリズムの計算量 $O(N^2)$ に従っており，スレッド数 4 または

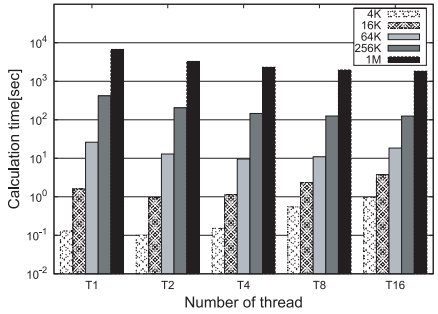


**図 7**　スレッド数と文字列長が計算時間に与える影響
Fig. 7　Impact of multithreading and sequence length on computational time.

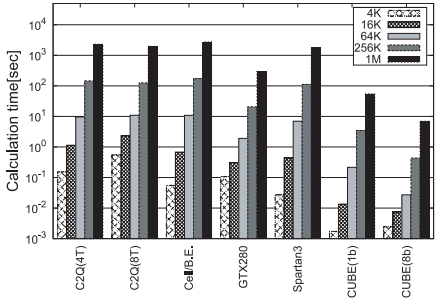


**図 8**　C2Q，Cell/B.E.，GeForce GTX280 と CUBE の計算時間の比較
Fig. 8　Comparison of computational time among C2Q, Cell/B.E., GeForce GTX280 and CUBE.

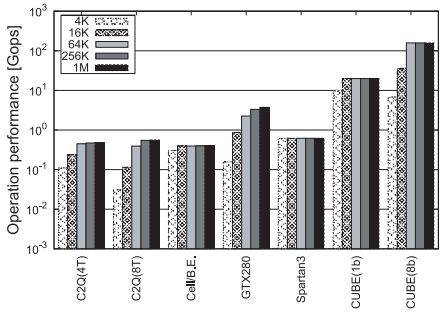


**図 9**　編集距離計算の処理性能
Fig. 9　Performance for computing edit distance.

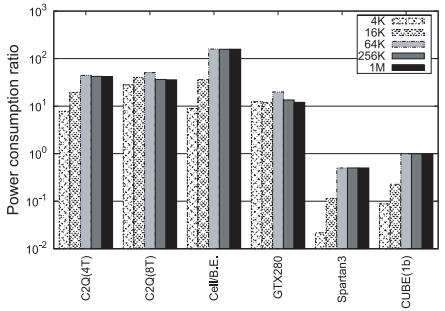


**図 10**　CUBE(8b) を基準とする消費エネルギーの比率
Fig. 10　Power consumption ratio based on CUBE(8b).

8 で計算時間の改善が飽和している．

### 6.5 性能評価

4 つのプラットフォーム上での性能について，以下の 3 つの観点から評価を行い，それらの結果をそれぞれ**図 8**，**図 9**，**図 10** に示す．

(1)　1 つの問題の計算に要する時間（図 8）
計算対象の文字列の長さと計算に要する時間との関係を計測した結果を図 8 に示す．CUBE および Spartan 3 は 1 つの FPGA での処理に要するクロックサイクル数（2,655 クロックサイクル）と動作周波数（100 [MHz]）から対象文字列を処理するために必要な時間を求め，他のシステムはプログラムの実行時間を計測した．

(2)　単位時間あたりの処理性能（図 9）
編集距離計算アルゴリズムは，スコアテーブルの要素を埋める処理の繰返しで占められるため，単位時間あたりに計算される要素数は，計算システムの性能評価として使用できる．この結果は，ランダムな文字列を問題として使用した場合に，1 秒あたりに処理できる要素数を図 8 の結果から計算したものである．この結果から，64 k 文字の編集距離を計算する際，CUBE(8b) は Core2Quad で 4 スレッド並列実行したときと比べ，約 352 倍の性能を示すことが確認された．

(3)　対象文字列長と消費エネルギー [J] の関係（図 10）
CUBE(8b) での処理を基準とする消費エネルギー（消費電力量）[J] の相対値を示す．

**表 3**　各システムの消費電力
Table 3　Power consumption of each system.

| Vendor | Device | Power [W] |
|---|---|---|
| Intel | Core2Quad Q9550 | 95 |
| Sony | ZEGO(BCU-100) Cell/B.E. | 330 |
| NVIDIA | GeForce GTX280 | 236 |
| Imperial | CUBE (8 boards) | 832 |

消費エネルギーは，**表 3** に示す各デバイスやシステムのデータシートから得た消費電力に，対象文字列の計算時間（図 8）を乗じた値を用いた．評価に使用している消費電力は，稼働中に動的に変動する可能性があり，図 10 に示された消費エネルギーは最大値である．たとえば NVIDIA 社の GPU は実行コードによってアイドル時 50 [W] から 200 [W] 程度まで大きな幅が見られることが報告されている[17]．また，Spartan 3 も，FPGA 上に構成されるハードウェアや動作状況によってダイナミック電力が変動する．Intel のプロセッサや Cell/B.E. も各々電力消費を抑える技術が導入されている．

## 7. 議　　論

### 7.1 各プラットフォームにおける性能向上の可能性

図 8，図 9 に示された各プラットフォームにおける編集距離計算の性能は，3.1 節で述べたアルゴリズムで実行しており，枝刈りなど文字列の内容に応じた処理は使用していない．そのため，いずれも計算時間は文字列長 $N$ に応じた $O(N^2)$ となっている．

Cell/B.E. のプログラムは各コアにブロック計算のジョブを発行しているだけなので，文字列長と計算時間の関係は Core2Quad での実行と同様のものになっている．また，Cell/B.E. の処理性能は 300 [Mops] 程度で飽和している．Cell/B.E. のプログラムでは使用してない SIMD 命令やビット並列化が利用できることを考慮すると，単位時間あたりの処理性能は大きくなることが考えられる．さらに，このプログラムは 1 ブロックのサイズが 128 文字で固定されており，通信時間と計算時間の割合やブロック計算に要するオーバヘッドなどは検討されていないため，各 SPE において計算速度を上げる工夫をしたうえで，ブロックサイズを調整することで処理性能の飽和値が上げられると考えられる．

GPU の処理性能は一定値に飽和せず，文字列長に応じて性能向上が見られる．GTX280 は演算コアである SP を 240 基持っており，SP は各々の演算器のパイプラインを埋めるために複数のスレッドを実行する．このことから，文字列長が短い場合は十分な数のスレッドが動作せず処理性能が低く抑えられており，文字列が長くなるにつれて演算パイプラインが埋まり，処理性能が向上していると考えられる．このプログラムの場合も，8 つの SP が共有するローカルメモリのサイズ（16 KB）に合わせてブロックサイズを調整することで飽和値を上げることができると考えられる．

**表 4**　Spartan 3 と CUBE の性能比較
Table 4　Performance comparison between Spartan 3 and CUBE.

| 文字列長 | CUBE(1b) | CUBE(8b) | 8b/1b |
|---|---|---|---|
| 4 k | 15.760 | 11.052 | 0.701 |
| 16 k | 32.252 | 58.783 | 1.823 |
| 64 k | 32.252 | 256.250 | 7.945 |
| 256 k | 32.252 | 256.250 | 7.945 |
| 1 M | 32.252 | 256.250 | 7.945 |

### 7.2 CUBE における処理性能の解析

6 章の評価から，CUBE は編集距離計算のブロック化アルゴリズムについて，他のプラットフォームと比べて優れた性能を示すことが確認された．CUBE の実装は，編集距離のブロック計算を FPGA アレイ上に展開し，その部分においては計算に要する時間を $O(N^2)$ から $O(N)$ に引き下げるものである．この状態では文字列長が大きくなるにつれて処理性能が向上するが，もしも文字列が 1 度のデータ転送に収まらない長さの場合は，FPGA アレイにデータを流す処理を繰り返すことになり，以降は処理性能は飽和してしまう．このときの文字列長は CUBE(1b) では 8 k 文字（$= 128 \times 64$），CUBE(8b) は 64 k 文字（$= 128 \times 512$）である．図 9 に示された値をもとに，単一の Spartan 3 で実行したときの処理性能を 1 としたときの，CUBE(1b) および CUBE(8b) の相対性能を**表 4** に示す．

図 9 に示したように，同じアルゴリズムを使用しているため，単一の Spartan 3 ではマイクロプロセッサ（Core2Quad）と比べて大きな性能差は得られなかったが，CUBE の多数の FPGA に演算を展開することでブロック計算の並列性を活用し，表 4 に示したとおり，64 個の FPGA を用いて約 32 倍，512 個の FPGA を用いて約 256 倍の性能向上が得られることが確認された．また，CUBE(1b) と CUBE(8b) では FPGA の数に比例した性能向上（約 8 倍）が得られていることが明らかになった．

### 7.3 CUBE における処理性能向上手法の検討

編集距離計算は，スコアテーブルの要素を埋める以外の処理が存在しないので，この処理速度が性能を決定する．そのため，CUBE を用いて高速なアルゴリズムを実装するために

は，本実装でスループットを示す IPC について検討する必要がある．

CUBE(8b) で，各 FPGA（LD_thread）が 128 文字のブロックの計算を行い，すべての FPGA が稼働するブロック計算サイクルが 1 サイクルだけある 64 k 文字の計算における IPC を計算すると，式 (2) から 1,581.320 となる．

$$\begin{aligned}\mathrm{IPC} &= \frac{(\text{スコアテーブルの要素数})}{(\text{計算に要するクロックサイクル数})} \\ &= \frac{([\text{文字列長 A}] \times [\text{文字列長 B}])}{([\text{LD\_thread の計算時間}] \times [\text{ブロック計算サイクルの回数}])} \\ &= \frac{(64 \times 1{,}024) \times (64 \times 1{,}024)}{2{,}655 \times 1{,}023} = 1{,}581.320 \end{aligned} \tag{2}$$

このとき，$\mathrm{IPC}_t$ の平均は 3.089（= 1,581.320/512），$\mathrm{IPC}_u$ の平均は 0.193（= 1,581.320/(512 × 16)）である．

同様に，CUBE(1b) の場合は，1 度に計算できる文字列長は 8 k 文字なので，64 k 文字の編集距離を求めるには，64（= 8 × 8）回 FPGA にデータを流す必要がある．このときの IPC は式 (3) より 199.027 となる．

$$\begin{aligned}\mathrm{IPC} &= \frac{(\text{スコアテーブルの要素数})}{(\text{計算に要するクロックサイクル数})} \\ &= \frac{([\text{文字列長 A}] \times [\text{文字列長 B}])}{([\text{LD\_thread の計算時間}] \times [\text{ブロック計算サイクルの回数}] \times [\text{データを流す回数}])} \\ &= \frac{(64 \times 1{,}024) \times (64 \times 1{,}024)}{2{,}655 \times 127 \times 64} = 199.027 \end{aligned} \tag{3}$$

このとき，$\mathrm{IPC}_t$ の平均は 3.110（= 199.027/64），LD_unit の平均 IPC は 0.194（= 199.027/(64 × 16)）である．表 4 に示したように，CUBE(8b) の IPC は CUBE(1b) の 7.945 倍である．

現在の CUBE の実装を改良し，より高い性能を引き出すためには，2 つの方法が考えられる．第 1 は FPGA の特徴を活かして他のデバイスよりも高速な演算を実現する方法であり，第 2 は IPC を上げる工夫をする方法である．

まず，4 章で述べたビット並列化技術を FPGA でも実装する方法が考えられる．ビット並列化は，LD_unit のようにハードウェアモジュールを並列に動作させる手法から，アルゴリズムをベクトルのビット演算に置き換える方法である．FPGA のようなデバイスはビット演算を得意としており，たとえば現在の LD_unit16 モジュールの構成を 128 ビットベクトルの並列計算モジュールに実装を変更した場合，同様の 128 文字ブロックを計算する際には 8 倍程度の性能向上が見込まれる（4 章を参照のこと）．このハードウェアはビット論理演算と加算のみで実装できることから，各 FPGA で処理できる文字列長をより長くすることができるようになり，処理性能の飽和値が高くなることが期待できる．

次に，IPC を上げる方法は，FPGA アレイにデータを流す際の手順を改良することで実現できる．現在の CUBE の実装では，LD_thread 内の 16 モジュールの LD_unit がすべて稼働するブロック計算サイクルは 31 回の計算中で 1 回のみなので，IPC が低くなってしまっている．LD_thread の命令を変更し，より長い文字列長を計算することにして IPC を上げることが考えられる．たとえば，256 文字の計算を行うようにすると，LD_unit は 16 × 16（= 1,024）ブロックの計算を行うことになる．計算開始後，第 15 ブロック計算サイクル（$\sum_{i=1}^{15} i = 120$ ブロック分）までは，従来と同じく順に稼働する LD_unit が増えていくが，第 16 ブロック計算サイクル以降は最後の 15 ブロック計算サイクルを除いてすべての LD_unit が稼働する．最後の 15 ブロック計算サイクル（$\sum_{i=1}^{15} i = 120$ ブロック分）は，開始時とは反対に，稼働する LD_unit が減っていく．計算開始と終了の各 15 ブロック計算時間を除く 49 ブロック計算時間では，すべての LD_unit が稼働するので，このときの $\mathrm{IPC}_t$ は式 (4) より 9.994 となり，$\mathrm{IPC}_u$ の平均は 0.625（= 9.994/16）と，LD_unit 単体と比べた際に，IPC の下落を小さく抑えることができる．

$$\begin{aligned}\mathrm{IPC}_t &= \frac{(\text{スコアテーブルの要素数})}{(\text{計算に要するクロックサイクル数})} = \frac{256\,[\text{文字}] \times 256\,[\text{文字}]}{83\,[\text{Clock/Block}] \times 79\,[\text{BlockCycle}]} \\ &= 9.994\,[\text{Operation/Clock}] \end{aligned} \tag{4}$$

この方法で $\mathrm{IPC}_t$ が式 (1) の約 1.56 倍（= 10.967/6.368）になるので，CUBE の処理性能が上がることが確認された．この場合，命令列が長くなるので，LD_thread の命令格納用のメモリを多く消費することになるが，表 1 より BlockRAM には余裕があるため，このような実装は十分に可能である．このとき，計算の入出力に必要なデータの量は 2 倍になるが，256 文字の 1 ブロックの計算に要する時間も 128 文字のブロック計算のおよそ 2.3 倍（= 72/31）になるので，6.2 節で検討した通信がボトルネックになることはない．また，データを何度も CUBE に投入しなければならない長大な文字列を計算する際には，*strA* は分割せず，*strB* のみを FPGA 数に応じて分割して計算を行うことで，アイドル状態の FPGA を減らすことができる．

これら CUBE を用いて性能向上が可能であり，それらを使用しないで図 10 に示したような省エネルギー性を示していることから，CUBE による編集距離の計算は他のデバイス

と比べて優れた性能を示すことが確認された．

## 8. まとめと今後の展開

本論文では，編集距離計算を 1 次元 FPGA アレイ CUBE へ実装し，計算性能，消費エネルギーについて議論した．データレベルとループレベルの並列性を利用し，CUBE 上の FPGA を用いてシストリックアレイを構成した．評価の結果，x86 のマルチプロセッサ，GPU，Cell/B.E. と比較して，CUBE が良い省エネルギー性を示すことを明らかにした．

今後は，ハードウェアの利用効率を向上させる手法について検討するとともに，CUBE を対象として，浮動小数点演算を含む具体的な科学計算を実装し，CUBE の実用性を示す予定である．

## 参 考 文 献


1) Hamada, T., Fukushige, T., Kawai, A. and Makino, J.: PROGRAPE-1: A Programmable, Multi-Purpose Computer for Many-Body Simulations, *Publications of the Astronomical Society of Japan*, Vol.52, pp.943–954 (2000).
2) Burke, D., Wawrzynek, J., Asanovic, K., Krasnov, A., Schultz, A., Gibeling, G. and Droz, P.Y.: RAMP Blue: Implementation of a Multicore 1008 Processor FPGA System, *Proc. 4th Annual Reconfigurable Systems Summer Institute* (*RSSI'08*) (2008).
3) Osana, Y., Fukushima, T., Yoshimi, M. and Amano, H.: An FPGA-Based Acceleration Method for Metabolic Simulation, *IEICE Trans. Inf. Syst.*, Vol.E87-D, No.8, pp.2029–2037 (2004).
4) Mencer, O., Tsoi, K.H., Craimer, S., Todman, T., Luk, W., Wong, M.Y. and Leong, P.H.W.: CUBE: A 512-FPGA CLUSTER, *Proc. IEEE Southern Programmable Logic Conference* (2009).
5) Yoshimi, M., Nishikawa, Y., Osana, Y., Funahiashi, A., Hiroi, N., Shibata, Y., Yamada, H., Kitano, H. and Amano, H.: Practical Implementation of a Network-Based Stochastic Biochemical Simulation System on an FPGA, *The 18th International Conference on Field Programmable Logic and Applications* (*FPL'08*), pp.663–666 (2008).
6) Morishita, H., Osana, Y., Fujita, N. and Amano, H.: Exploiting Memory Hierarchy for a Computational Fluid Dynamics Accelerator on FPGAs, *Proc. Field Programmable Technology 2008* (*FPT'08*), pp.193–200 (2008).
7) Arnold, J.M., Buell, D.A. and Davis, E.G.: Splash 2, *SPAA '92: Proc. 4th annual ACM symposium on Parallel algorithms and architectures*, New York, NY, USA, pp.316–322, ACM (1992).
8) Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Soviet Physics Doklady*, Vol.10, No.8, pp.707–710 (1966).
9) Chang, C., Wawrzynek, J. and Brodersen, R.W.: BEE2: A High-End Reconfigurable Computing System, *IEEE Design and Test of Computers*, Vol.22, No.2, pp.114–125 (2005).
10) Cell Challenge 2009 実行委員会：先進的計算基盤システムシンポジウム SACSIS2009 併設企画 Cell Challenge 2009. http://www.hpcc.jp/sacsis/2009/cell/
11) GPU Challenge 実行委員会：Cell Challenge 2009 併設企画 GPU Challenge 2009. http://www.hpcc.jp/sacsis/2009/gpu/
12) Myers, G.: A fast bit-vector algorithm for approximate string matching based on dynamic programming, *J. ACM*, Vol.46, No.3, pp.395–415 (1999).
13) Hyyrö, H.: A bit-vector algorithm for computing Levenshtein and Damerau edit distances, *Nordic J. of Computing*, Vol.10, No.1, pp.29–39 (2003).
14) Masuno, S., Maruyama, T., Yamaguchi, Y. and Konagaya, A.: Multiple Sequence Alignment Based on Dynamic Programming Using FPGA, *Transaction on Information and Systems*, Vol.E90-D, No.12, pp.1939–1946 (2007).
15) 桜庭　俊，吉田悠一：Cell プロセッサを用いた編集距離計算の高速化 (2009). http://www.hpcc.jp/sacsis/2009/cell/outputs/pdf/kitei_1.pdf
16) SONY: BCU-100 Computing Unit with Cell/B.E. and RSX. http://pro.sony.com/bbsccms/ext/ZEGO/files/BCU-100_Whitepaper.pdf
17) 長坂　仁，丸山直也，額田　彰，遠藤敏夫，松岡　聡：GPU における性能と消費電力の相関性の解析，情報処理学会研究報告，Vol.2009-HPC-121，No.27，pp.1–5 (SWoPP2009) (2009).


(平成 22 年 1 月 26 日受付)
(平成 22 年 4 月 28 日採録)

**吉見　真聡**（正会員）

平成 16 年慶應義塾大学理工学部情報工学科卒業．平成 21 年同大学大学院理工学研究科開放環境科学専攻後期博士課程修了．博士（工学）．平成 18 年度より日本学術振興会特別研究員（DC1）．現在，同志社大学理工学部助教．リコンフィギャラブルシステム，並列処理，知的システムの研究に従事．電子情報通信学会，人工知能学会各会員．

**西川　由理**（学生会員）

平成 18 年慶應義塾大学理工学部情報工学科卒業．平成 20 年同大学大学院理工学研究科開放環境科学専攻修士課程修了．現在，同大学院理工学研究科開放環境科学専攻博士課程在籍中．平成 20 年度より日本学術振興会特別研究員（DC1）．ハイパフォーマンスコンピューティングとインタコネクトに関する研究に従事．

**天野　英晴**（正会員）

昭和 56 年慶應義塾大学工学部電気工学科卒業．昭和 61 年同大学大学院工学研究科電気工学専攻博士課程修了．工学博士．現在，慶應義塾大学理工学部情報工学科教授．計算機アーキテクチャの研究に従事．

**三木　光範**（正会員）

昭和 25 年生．昭和 53 年大阪市立大学大学院工学研究科博士課程修了，工学博士．大阪市立工業研究所研究員，金沢工業大学助教授を経て，昭和 62 年大阪府立大学工学部航空宇宙工学科助教授，平成 6 年同志社大学工学部知識工学科教授．専門分野は最適設計，並列処理，システム工学，知的システムの設計．現在の研究テーマは，並列分散最適化に基づく知的オフィス環境の創造および知的照明システムの研究．主な著書は『進化する人工物』（オーム社），『工学問題を解決する適応化・知能化・最適化法』（技法堂出版）等．IEEE，人工知能学会，システム制御情報学会，電気学会，建築学会，空気調和・衛生工学会等各会員．知的オフィス環境コンソーシアム会長．

**廣安　知之**（正会員）

平成 9 年早稲田大学大学院理工学研究科後期博士課程修了．早稲田大学理工学部助手，同志社大学工学部助手．知識工学科専任講師，インテリジェント情報工学科准教授を経て，平成 20 年から生命医科学部教授．進化的計算，最適設計，並列処理，設計工学，医療画像工学等の研究に従事．IEEE，電子情報通信学会，計測自動制御学会，人工知能学会，日本機械学会，超並列計算研究会，日本計算工学会各会員．

**オスカー　メンサー**

スタンフォード大学計算機科学・算術グループにて Ph.D.（電気工学）取得．DIGITAL Systems 研究センター，ロックウェル計算機科学研究所，日立中央研究所を経て，ベル研究所計算機科学センター技術スタッフ．Maxeler Technologies 社創立者兼 CEO．現在，インペリアルカレッジロンドン上級講師兼工学・物理科学研究会議（EPSRC）上級フェロー．