メモリバンド幅の利用率を考慮した高性能プロセッサシステムの設計

藤枝 直輝[1,a)]　石垣 良樹[1]　佐藤 清広[1]　市川 周一[1,b)]

概要：本稿では，我々が第 2 回 ARC/CPSY/RECONF 高性能コンピュータシステム設計コンテストに向けて提出した，高性能プロセッサシステムの設計について述べる．本設計では，整数ソート・行列積・ステンシル計算に対応したストリームコプロセッサを搭載し，ボードに搭載された SDRAM のバンド幅を最大限に活用することで，性能の向上を図る．本設計を Terasic 社 DE2-115 ボード上に実装し，第 1 回コンテストにおける決勝問題のデータセットを用いて評価した結果，同ボード向けのリファレンス設計と比較して，137 倍から 591 倍の性能向上を達成した．

Design of a High-Performance Processor System considering the Usage of Memory Bandwidth

Naoki Fujieda[1,a)]　Yoshiki Ishigaki[1]　Kiyohiro Sato[1]　Shuichi Ichikawa[1,b)]

1. はじめに

近年，処理の一部をハードウェア化することで特定の応用に特化した計算機システム，すなわち専用計算システムに対する需要が高まっている．本稿では，第 2 回 ARC/CPSY/RECONF 高性能コンピュータシステム設計コンテスト [1] の提出デザインとして，2014 年 1 月の第 1 回コンテスト [2] に提出した設計 [3] から得られた知見をもとに，我々のチーム “CCS Cerberus” [*1] が提出した，高性能プロセッサシステムの設計について述べる．

第 1 回コンテストに提出した設計においては，1 サイクルに多数のデータを処理する SIMD ユニットをキャッシュ上のデータに対して適用することで処理の高速化を図り，ツールキット付属のデータセットに対して，最大でツールキット比 31.1 倍の性能を示した [3]．しかしながら，コンテストにおける決勝問題のデータセットにおいては，性能比は高々 21.6 倍に留まった [2]．この結果は，データセットの拡大に伴い，キャッシュが有効に利用されなくなったことを示唆している．そのため，むしろ SDRAM のバンド幅を有効に活用することが性能向上に寄与すると考えられる．

この反省をもとに，今回提出したシステムでは，整数ソート・行列積・ステンシル計算に対応したストリームコプロセッサを搭載することを主要な改善点とした．コプロセッサは CPU から要求を受け，SDRAM からのデータの読み出しを要求し，得られたデータをパイプライン演算し，結果を SDRAM へと書き戻す．コプロセッサのインタフェースは，個々のアプリケーション向けの記述が容易となるよう，かつ SDRAM のバンド幅を活用できるように注意した設計をなす．本稿では，2 節で設計システムの全体像，およびコプロセッサのインタフェースについて述べたあと，3 節で個々のアプリケーションに対するコプロセッサの設計と，それを用いたソフトウェアの修正点について述べる．4 節で評価を行い，これらの改善の有効性を確認する．

2. システムの枠組み

図 1 に提出した高性能プロセッサシステムの全体像を示す．システムは Altera 社 Qsys システム統合ツールを用い

[1] 豊橋技術科学大学
Toyohashi University of Technology
[a)] fujieda@ee.tut.ac.jp
[b)] ichikawa@tut.jp
[*1] [si: si: ɛs sɛrbərəs]．CCS は Custom Computing Systems（専用計算システム）の略である．

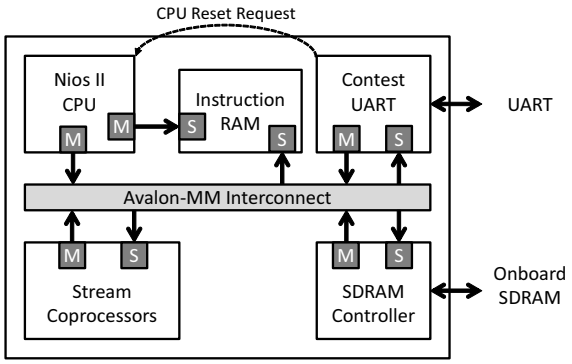

図 1 高性能プロセッサシステムの全体像.

て設計しており，各構成要素は Avalon-MM インタコネクトによって接続される．図中の M，S はそれぞれ Avalon-MM のマスタ，スレーブインタフェースを意味する．

CPU には Nios II/f ソフトプロセッサを利用する．命令は専用の命令メモリ（Instruction RAM）から供給する．データはオンボードの SDRAM から供給するが，データキャッシュを用いてレイテンシを短縮する．プログラムおよび初期データは，コンテストツールキット [1] 付属の UART コントローラ（Contest UART）を用いて，シリアル通信により書き込む．Nios II はソフトリセット信号（CPU Reset Request）を Qsys システム外へエクスポートする機能をもつ [4]．UART コントローラの転送中を示す信号をソフトリセット信号へと接続することで，転送終了とともに CPU が動作を開始する，ツールキット同様の仕様を実現する．

図 1 の左下に示す構成要素が，本システムの性能向上の鍵となるストリームコプロセッサである．**図 2** にコプロセッサ，およびそれらと Avalon-MM インタコネクトとの間のインタフェースの概略を示す．スレーブインタフェースは CPU からアクセスされ，引数（arg0 - arg3）や，どのコプロセッサを動作させるかを示すチップセレクト（cs）といった情報が書き込まれる．また，必要に応じて計算結果（result）が読み出される．マスタインタフェースは SDRAM コントローラにアクセスし，入出力 FIFO との間で必要なアドレス・データのやりとりをする．

コプロセッサ本体は大きく分けて，読み出しアドレス生成部とパイプライン処理および書き込みアドレス生成部とに分けられる．チップセレクトにより選択されたコプロセッサが起動すると，まず読み出しアドレス生成部により，処理に必要なデータに対するアドレスの系列が生成され，入出力 FIFO の読み出しアドレス部（read_addr）に次々に渡される．これにより，SDRAM コントローラによりデータが読み出され，その結果が入出力 FIFO の読み出しデータ部（read_data）へと格納される．このデータをもとにコプロセッサはパイプライン処理を行い，その出力は書き込みアドレスとともに，入出力 FIFO の書き込みアドレス

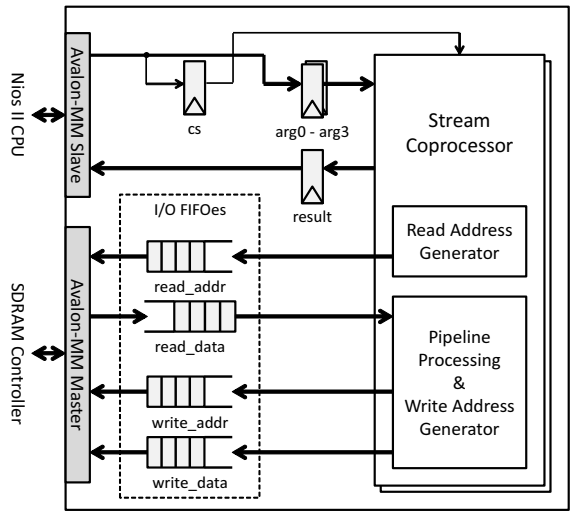

図 2 ストリームコプロセッサ，および Avalon-MM とのインタフェース.

部・データ部（write_addr，write_data）に渡される．全てのデータに対する処理が終了し，かつ全ての FIFO が空になったら，メモリマップされた終了フラグをアサートし，CPU に対し処理の終了を通知する．

コプロセッサの設計例として，整数ソート課題の初期化処理である以下の C プログラムに対応するコプロセッサを考える．

```
for (i = 0; i < n; i++)
    data[i] = init_data[i % (64 * 1024)] + i;
```

読み出しアドレス生成部では，リクエスト生成ごとにインクリメントされるカウンタを用意し，その下位 16 ビットに 4 を乗じる．これを引数として与えられた配列 `init_data` の先頭アドレスに加算することで所望のアドレスを生成する．パイプライン処理部では，データが読み出されるごとにインクリメントされるカウンタを用意し，これを読み出されたデータに加算することで，書き込みデータを生成する．また，それと同時にカウンタの値に 4 を乗じ，配列 `data` の先頭アドレスに加算し，書き込みアドレスを生成する．

SDRAM コントローラへのアクセスや，FIFO のデータが満杯になることによる処理の停止などの制御はインタフェース側で行う．そのため，個々のアプリケーション向けのコプロセッサをシンプルに記述できる．また，読み出し・書き込みの順序を適切に設定することで，SDRAM のバンド幅の利用効率が向上する．

3. アプリケーションの高速化

本節では，個々のアプリケーションに対して，対応するコプロセッサの設計またはソフトウェアの修正点について述べる．下記に挙げるほか，最短路問題を除く 3 つの問題に対しては，データ配列の初期化処理のためのコプロセッサを別途設計している．

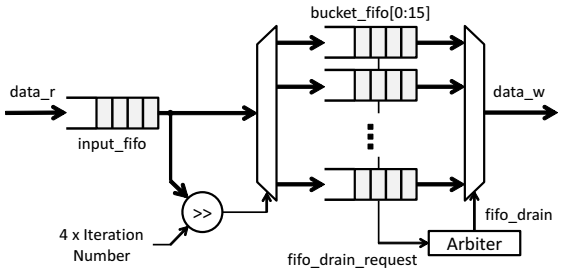

図 3 バケットソートの処理.

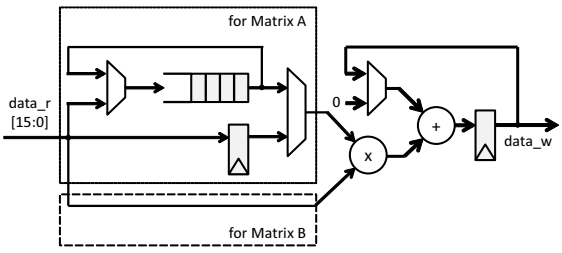

図 4 行列積の処理.

3.1 整数ソート

整数ソートの高速化には，基数ソートの 1 回の繰り返し，すなわち入力データの一部をキーとしたバケットソートに対応するコプロセッサを設計した．**図 3** にバケットソートの処理のブロック図を示す．キーの長さは 4 ビットとする．すなわち，バケットの数は $2^4 = 16$ 個であり，入力データが 28 ビット以下の整数ならば，バケットソートを 8 回繰り返すことで基数ソートが完了する．引数として，入力バケット群，出力バケット群，各バケットに格納されたデータの要素数の配列の先頭アドレス，および繰り返し番号を与える．

コプロセッサは，はじめに各バケットに格納されたデータの要素数を取得する．それをもとに各バケットから要素を読み出し，キーをもとに適切な出力 FIFO （bucket_fifo）へと振り分ける．各出力 FIFO に格納されたデータの個数が閾値を超えると，fifo_drain_request 信号がアサートされる．この信号をもとに，アービタによってどの出力 FIFO から書き込みデータを取り出すかを決定する．出力 FIFO とアービタは，書き込みアドレス列の連続性を高め，SDRAM への書き込みの効率を改善するためのものである．全ての要素を出力バケット群へと書き込んだら，各バケットに振り分けられた要素数を更新して処理を終了する．

このコプロセッサでは，1 回の繰り返しにつき，入力バケットの全要素の読み出し，それと同数の出力バケットへの要素の書き込み，また，データの要素数の配列への 1 回ずつの読み書きを必要とする．そのため，要素数 N の整数配列をソートするには，合計で $16N + 256$ 回の SDRAM へのアクセスを必要とする．

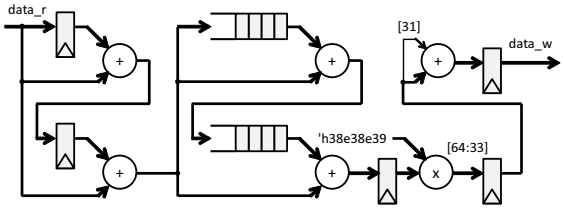

図 5 ステンシル計算の処理.

3.2 行列積

行列積 $C = AB$ の計算の高速化には，**図 4** に示す，出力行列 C を 1 行ごとに計算するコプロセッサを設計した．引数として，入力行列 A と B （B はメモリ上では転置された状態で格納されている），出力行列 C の先頭アドレス，および行列サイズ N を与える．

コプロセッサは，計算する行を k とすると，まず行列 A の k 行目と行列 B の 1 列目の各要素を交互に読み出す．これらの積和を取るとともに，行列 A の要素については FIFO に保存しておく．これにより c_{k1} が計算される．次に，行列 B の 2 列目の要素を順次読み出す．これらと FIFO に保存している行列 A の要素との積和を取り，c_{k2} を計算する．行列 B の全ての列についてこれを繰り返すと，行列 C の k 行目の全ての要素が計算できる．そうしたら，FIFO をクリアし，k をインクリメントして最初に戻る．

このコプロセッサでは，1 行の計算につき，行列 A の 1 行と行列 B の全要素の読み出し，行列 C の 1 行の書き込みを要する．そのため，行列サイズ N の行列積には，合計で $N^3 + 2N^2$ 回の SDRAM へのアクセスを必要とする．

追加の改良として，図 4 に示す演算器を k 個並べ，行列 C を k 行ごとに計算するよう変更できる．これにより，SDRAM へのアクセスは $N^3/k + 2N^2$ 回に削減される．この改良はコンテスト決勝までに実施する予定である．

3.3 ステンシル計算

ステンシル計算の高速化には，図 4 に示す，自要素と隣接 8 要素との平均を計算するコプロセッサを設計した．引数として，入力 A，出力 B，および問題サイズ N を与える．

コプロセッサは，入力 A の全ての要素を順番に読み出す．読み出されたデータに対しては，まず図左側の 2 つの加算器により自要素と左右に隣接する要素との和を取る．この和を FIFO を用いて N サイクルずつ遅延させたのちに図中央の加算器に与えることで，上下に隣接する要素との和算を行う．これにより自要素と隣接 8 要素との合計が計算されるので，これを 9 で除算して（実際には，0x38e38e39 との乗算と 33 ビットの右シフトを行い，負数であれば 1 を加える [5]），所望の値を得る．

このコプロセッサでは，ステンシル計算の 1 回のイテレーションにつき，入力 A の全要素の読み出しと，出力 B の外周を除く要素の書き込みを要する．そのため，行列

表 1 提出システムの性能評価結果.

アプリケーション	Ref [s]	前回設計 [s]	本システム [s]	対 Ref 比	対前回比	予想時間 [s]
整数ソート（N=819200）	36.953	2.332	0.211	175x	11.0x	0.164
行列積（N=320）	59.696	2.135	0.436	137x	4.9x	0.412
ステンシル計算（N=192，I=220）	43.312	2.558	0.221	196x	11.6x	0.201
最短路問題（2048 ノード，8192 エッジ）	28.389	4.232	0.048	591x	88.2x	n/a

サイズ N のステンシル計算には，イテレーション回数を I として，合計で $I(2N^2 - 4N + 4)$ 回の SDRAM へのアクセスを必要とする．

3.4 最短路問題

最短路問題の高速化のための改良は 2 点挙げられる．ひとつは，探索対象を決定するために優先度付き待ち行列を用いることである．もうひとつは，あらかじめエッジの配列をその端点のノード番号でソートしておき，各ノードに対応するエッジの集合を速やかに求めることである．

エッジの配列のソートには，3.1 節で述べた整数ソート向けのコプロセッサが利用できる．まず，各エッジについて，上位 16 ビットにエッジ番号，下位 16 ビットに各端点のノード番号をもつ 2 つの整数を生成し，これらの配列を作成する．次に，この配列の下位 16 ビットに対して基数ソートを行う．ソートされた配列のエッジ番号（すなわち上位 16 ビット）を取り出し，当該エッジを新しいエッジ配列へと順次コピーする．

4. 実装と評価

本節では設計したプロセッサシステムを Terasic 社 DE2-115 ボード上に実装し，第 1 回コンテストにおける決勝問題のデータセット [2] を用いて評価する．論理合成と FPGA への実装には Altera 社 Quartus II 13.1 を使用し，レジスタ・リタイミングの有効化を除く設定はデフォルト設定を用いる．プログラムの実行ファイルは Mentor Graphics 社 Sourcery CodeBench Lite 2013.05 （gcc 4.7.3，binutils 2.23）を用いて生成する．

比較対象として，コンテスト実行委員会が公開している DE2-115 ボード向けのリファレンス設計（Ref）を用いる．リファレンス設計の動作周波数は 40 MHz である．我々のシステムの動作周波数は 80 MHz とする．いずれの設計でも，SDRAM はシステムのクロックと同一の周波数で動作させる．なお，SDRAM コントローラについては，Altera 社より提供されているコントローラが複数バンクへの並列アクセスに対応していなかったため，これに対応したコントローラをスクラッチで実装した．

評価基準としては，シリアル通信により全ての入力データを受け取ってから，計算を完了して全てのデータが出力されるまでの時間を，計算時間と定義する．すなわち計算時間とは，コンテストにおいて定義される実行時間から，入力データの受信に要する時間を除いたものである．

評価結果を**表 1** に示す．参考のため，第 1 回コンテストに提出した設計 [3] の実行時間より求めた計算時間を，前回設計として提示する．また，3 節で求めたコプロセッサが DRAM へアクセスする回数を，周波数 80 MHz で除したものを，予想時間として提示する．表 1 より，本システムは，リファレンス設計に対する性能比が最も高かった最短路問題で 591 倍，最も低かった行列積でも 137 倍の性能向上を達成した．また，最短路問題を除く 3 つのアプリケーションについては，ソフトウェア処理の時間や SDRAM のデータアクセスの待ち時間を一切無視した予想時間と比較しても，最大で 28.7%（整数ソート）の実行時間増にとどまった．このことは，本システムのストリームコプロセッサが SDRAM のメモリバンド幅を十分に活用できていることを意味している．

5. おわりに

本稿では，我々が高性能コンピュータシステム設計コンテストに提出した高性能プロセッサシステム，特にストリームコプロセッサの設計について述べた．評価の結果，リファレンス設計と比較した性能向上比は全てのアプリケーションで 100 倍を超え，高い性能向上を達成した．

決勝デザインの提出に向けては，3.2 節で述べた行列積のコプロセッサの改良のほか，性能に関する種々のパラメータ調整の余地が残されている．今後より高い性能向上が達成できるよう，修正と調整を進めていく．

参考文献

[1] 高性能コンピュータ設計コンテスト実行委員会：The 2nd ARC/CPSY/RECONF High-Performance Computer System Design Contest, (online), available from ⟨http://aquila.is.utsunomiya-u.ac.jp/contest/⟩ (accessed 2014-07-28).
[2] プロセッサ設計コンテスト実行委員会：The 1st IPSJ SIG-ARC High-Performance Processor Design Contest , (online), available from ⟨http://www.arch.cs.titech.ac.jp/contest/⟩ (accessed 2014-07-28).
[3] 藤枝直輝，宇山和輝，市川周一：ソフトプロセッサ向けの SIMD 整数演算ユニットの設計と実装，情報処理学会研究報告 2014-ARC-208，No. 14 (2014).
[4] Altera Corporation: Alternative Nios II Boot Methods, Application Note AN-458-2.2 (2014).
[5] Granlund, T. and Montgomery, P. L.: Division by Invariant Integers Using Multiplication, *Proc. of PLDI 1994*, pp. 61–72 (1994).