Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

大規模再構成可能データパスにおける実行前処理削減方式の検討

片岡広志[†1]　本田宏明[†2]　Farhad Mehdipour[†1]
井上弘士[†1]　村上和彰[†1]

汎用プロセッサと比較して高い演算性能を持ち，特定の処理を低消費エネルギーで実行可能なハードウェアアクセラレータが注目されている．しかしながら，アクセラレータは演算性能に比例したデータ供給能力を必要とするために，提供されるメモリ性能によって演算性能が律速されるという問題が発生し易い．そこで本研究室では要求するメモリバンド幅を削減するアクセラレータとして，大規模再構成可能データパス（LSRDP：Large-Scale Reconfigurable Data-Path）を提案している．LSRDP は高い演算性能を持つが，その実行前処理がオーバーヘッドとなり性能向上を阻害していることも判明している．そこで本稿では，LSRDP の実行前処理の削減手法を検討した．具体的には，プログラムに LSRDP 用のデータ構造を持たせることで実行時の前処理を削減した．また，その際にデータの重複が発生し新たなオーバーヘッドとなったため，ハードウェアによるサポートの検討も行った．提案方式の性能を評価した結果，汎用プロセッサ単体での実行と比較し，熱伝導方程式と振動方程式それぞれについて 109 倍，76 倍の性能向上見積りを得た．

Eliminating overhead preprocessing times for a Large-Scale Reconfigurable Data-Path

hiroshi kataoka,[†1] hiroaki honda,[†2]
Farhad Mehdipour,[†1] koji inoue[†1]
and kazuaki murakami[†1]

Nowadays, hardware accelerators integrating to general purpose processers (GPPs) are increasingly employed to achieve lower power consumption and higher processing speed. However due to impact of memory-wall problem, this kind of acceleration does not always achieve a demanded performance. To resolve this issue, a Large-Scale Reconfigurable Data-Path (LSRDP) has been proposed which is able to reduce the required memory bandwidth. LSRDP consists of a lot of FPUs therefore; it can potentially achieve a very high performance. However, it suffers from long overhead preprocessing time for some attempted applications. In this paper, a software technique has been proposed to eliminate the overhead time required for data rearrangement prior to execution. In addition, a hardware is introduced which is augmented to the architecture and exploits consequent data overlaps to reduce the number of memory accesses for reading input data and alleviate stall time. Our experimental results reveal speedup up to 109 for the architecture exploiting LSRDP.

1. はじめに

近年，科学技術計算などの HPC（High Performance Computing）分野では，汎用プロセッサ (GPP：General Purpose Processor) を用いたスカラー型並列計算機やクラスタシステムが主流となっている．実際に、スーパーコンピュータの性能を示す TOP500[1)] においても，そのほとんどが GPP を用いた構成である．

一方で，プログラム中の高負荷部分においては，GPP 単体では演算性能が一時的に不足することもある．そこで，演算性能に特化したハードウェアであるアクセラレータによって，高負荷部分を加速実行し，システム全体の性能を向上させる研究も注目されている．

しかしながら，アクセラレータは演算性能に比例した高いメモリ性能を要求する．この原因は，多くのアクセラレータが，チップ上に多数の演算器を配置し並列計算を行っている点にある．アクセラレータは，並列に配置された演算器それぞれが入力データを必要とするために大きなメモリバンド幅を要求する．したがって，アクセラレータの演算性能に対してメモリ性能が低い場合に，実効性能がメモリ性能によって抑えられてしまうという問題 (メモリウォール問題[2)]) が発生する．

従来はこの問題に対して，キャッシュメモリ等のオンチップメモリを搭載することで対処している．オンチップメモリは主記憶と比較すると，高速・小容量という特徴を持っている．オンチップメモリは主記憶への参照の局所性を利用し，将来参照されるであろうデータを保持しておくことで，主記憶へのアクセス回数を減らし，主記憶へのアクセスレイテンシを削減している．しかしながら，プログラムを大規模並列で実行すると，オンチップメモリに搭

†1 九州大学大学院 システム情報科学府/研究院
Graduate school / Faculty of Information Science and Electrical Engineering, Kyushu University
†2 九州先端科学技術研究所
Institute of System, Information Technologies and Nanotechnologies

Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

載不可能なほど大量の中間データが発生し，余分なオフチップメモリアクセスが必要になるという問題が発生する．このことから，近年では従来手法による解決策に限界が見え始めている．

本研究室では科学技術計算に特化したアクセラレータとして大規模再構成可能データパス（LSRDP：Large Scale Reconfigurable Data Path）を提案している[3),4)]．LSRDP は，チップ上に多数の演算器を配置し，それらをプログラマブルなネットワークで接続した構成を採る．このアーキテクチャの特徴は，ある演算器の出力を直接他の演算器の入力とすることが可能な点である．したがって，大量の中間結果が生じるような計算においても，中間結果を主記憶に格納することなく実行できる．

また，本研究ではアクセラレータを高速に動作させるために SFQ(Single Flex quantum) を用いた実装を提案している．SFQ は超電導ジョセフソン結合回路を用いており，CMOS 回路と比較して 1mV 程度の非常に小さな電圧で高速に動作する．SFQ 回路を用いることで，LSRDP を作った際に問題となる高速動作による高い消費電力や高密度実装による高発熱の解決が可能であると考えている．

LSRDP の研究開発においては，

- アーキテクチャの検討
- LSRDP コンパイラの開発
- LSRDP 向けアプリケーションの開発
- 性能評価環境構築並びに性能評価

が必須と考えられるが，本研究では性能評価環境の構築に主眼を置いており，さらに，性能評価から得られた知見をフィードバックし，メモリアーキテクチャの検討を行っている．

これまでの性能評価結果[5)] から，実行前処理となる再配置時間が実行時間において支配的であり，性能向上を阻害していることが判明している．再配置とは、プログラムが持つデータ構造を LSRDP の実行に適した構造に変化させることである．

本稿では，LSRDP の性能向上を目標とし，実行前処理の削減手法を提案する．具体的には，事前に LSRDP 用のデータ構造を前提としたプログラムを作成するコードチューニングと，LSRDP による処理の特徴に基づいたハードウェアの追加を行う．

本稿の構成は以下に示す通りである．第 2 節で大規模再構成可能データパスの概要を説明する．第 3 節で，初期の性能評価と判明した問題点について述べ，第 4 節で提案方式の詳細を解説する．第 5 節で提案方式を用いた際の性能評価実験を行い，最後に 6 節でまとめる．

2. 大規模再構成可能データパス

2.1 LSRDP アーキテクチャ

GPP と主記憶にバス経由で LSRDP を付加したシステムの概観を図 1 に示す．LSRDP は Floating Point Unit（FPU）を 2 次元アレイ状に配置した構成となっており，ORN は横に並んだ FPU アレイ（以後，行と呼ぶ）を接続するネットワークである．LSRDP は科学技術計算を対象とするため，演算器は FPU で構成されている．また，ORN はデータフローを自由に変更できるクロスバスイッチ等で構成し，FPU の演算内容と ORN の接続情報を再構成可能としている．ORN を介して，ある FPU の出力を直接他の FPU の入力とすることが可能であるため，データ依存関係のある計算の中間結果をメモリに格納することなく計算可能である．また，ORN によるデータ転送には以下に示す制約が存在する．

- ORN は隣接行間のみデータ転送が可能である
- 非隣接行間では，それらの間に存在する FPU を利用してデータ転送を行う
- 同一行内でのデータ転送は不可能である
- データ転送を行う方向はすべて一意である．

LSRDP はデータ入力用の SB（Streaming Buffer）と，SB への入出力を制御するコントローラ（SMAC：Streaming Memory Access Controller）を持つ．SB は FIFO 動作のバッファであり，LSRDP コアの各入出力ポートは SB を介し 1 つのメモリアクセスコントローラーを使用して DRAM からなる主記憶へとアクセスするという構成を採っている．SMAC は主記憶と SB 間のデータ転送を制御し，データの転送はオフチップのバスを介して行う．この際に DRAM との高速なデータ転送（DRAM ページキャッシュ機構を用いたバースト転送）を行う必要性から，LSRDP の入力は連続なアドレスを持つメモリ空間に保存する必要がある．また，LSRDP からは連続なアドレス空間にデータが出力される．例えば，通常の和の計算：c[i]=a[i]+b[i] (i=1〜M) に対して，GPP 用の通常プログラムであれば，a[ ] と b[ ] とがメモリ上における別領域に置かれ，式を計算する毎に a[i] と b[i] とを個別にロードし DRAM においてランダムアクセスを行うことになる．これに対し LSRDP(幅=WIDTH) による計算の際に DRAM とのバースト転送を使用する場合では，c[i]〜c[i+WIDTH] 分の計算を a[i]〜a[i+WIDTH] と b[i]〜b[i+WIDTH] の入力から行うために a[ ] と b[ ] の並びを a[0],b[0],a[1],b[1],a[2],b[2],… と連続アクセスが可能となるようにデータのパッキングを行う必要がある．

LSRDP を付加したシステムでアプリケーションを実行するには，ソースコードを LSRDP

Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

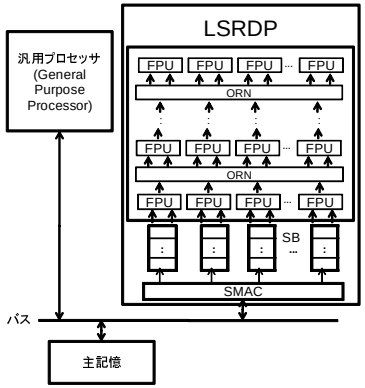

図 1 大規模再構成可能データパス（LSRDP）を含むシステムの概観
Fig. 1 Overall architecture of the SFQ-LSRDP computer

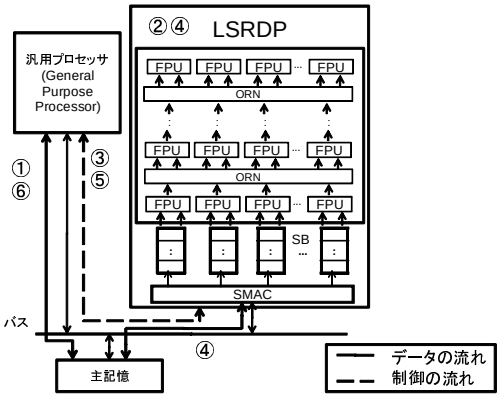

図 2 LSRDP を含むシステムの動作
Fig. 2 LSRDP execution model

で実行する部分と GPP で実行する部分に分割し，それぞれコンパイルする必要がある．LSRDP 部分のコードは元のプログラムよりデータフローグラフ（DFG：Data Flow Graph）を専用のツールを用いて抽出し，マッピングツールを使用して LSRDP に対するマッピングデータへと変換する．なお，アプリケーション内で LSRDP で実行する部分はプログラマが決定するが，プログラマは GPP と LSRDP 上の実装に最適なアルゴリズムを探索する必要がある．

2.2 LSRDP の動作

LSRDP は GPP に対するコプロセッサとして動作する．GPP と LSRDP がオーバーラップ実行しないとすると，その動作は以下のようになる（図 2）．

(1) GPP が LSRDP に入力するデータを主記憶内で整列
(2) LSRDP を再構成
(3) GPP が LSRDP に演算の開始を指示
(4) LSRDP が演算を実行
(5) LSRDP が演算の終了を GPP に報告
(6) GPP が LSRDP の出力データを主記憶内で整列

上記 4. における LSRDP での演算には，入力データの読み込みと，出力データの書き込みが含まれる．LSRDP は主記憶からバーストアクセスによりデータ転送を行うので，演算を実行させる前後に 1.，6. において主記憶内のデータを整列する必要がある．また，LSRDP は 1 クロックサイクルごとにパイプライン処理を行う．このとき，入力されるデータの集合が SB 上に存在する限り LSRDP はストールせずに演算を実行する．しかしながら，SB 上に入力されるデータの集合が存在しない場合は，SB に入力が準備されるまで演算をストールさせる．

2.3 LSRDP の利用による要求メモリバンド幅の抑制

一般的なアクセラレータは，大量の演算器を搭載し並列に動作させることで高い演算性能を実現している．したがって，演算実行時には演算器 1 つに対して 2 つのデータを入力として与える必要がある．更に演算器ごとに異なる入力データを与えているため，演算器の個数に比例して要求するメモリバンド幅が大きくなり，アクセラレータの高い演算性能を抑えてしまう要因となる．これに対し，データーフロー型の計算方式を採用している LSRDP では，主記憶から入力を受け取る演算器の数は少数である．その他の演算器はプログラム内のデータ依存関係を利用し，他の演算器の出力結果を直接入力している．したがって，要求メモリバンド幅を抑えることが可能となる．

Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

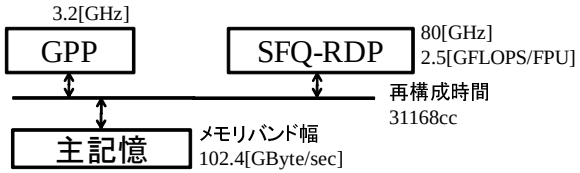

図 3 性能評価の各パラメータ
Fig. 3 Performance evaluation parameters

3. 初期評価実験

3.1 評価環境

提案している LSRDP アーキテクチャについて初期性能評価を行った[5]．評価時のシステムのパラメータを図 3 に示す．対象とするベンチマークプログラムは，2 変数 2 階の偏微分方程式から，熱伝導方程式 (Heat)，振動方程式 (Vib)，Poisson 方程式 (Poi) の 3 つ，量子化学分野のアプリケーションから二電子積分計算 (ERI) を用いた．

評価方法は，まず LSRDP の実行時間をアーキテクチャとベンチマークプログラムのパラメータを用いた数式によりモデル化した．また，GPP における実行時間をサイクルアキュレートなシミュレータで計測し，モデル式によって計算された LSRDP の実行時間と合わせることで，LSRDP を用いたシステムでの性能を予測した．比較対象として，GPP 単体での演算時間をシミュレータにより計測した．

3.2 性能モデリング

2.2 で示した LSRDP の動作に基づいて性能モデリングを行う．GPP と LSRDP がオーバーラップ実行しないとすると, アプリケーションの実行時間 ET は，GPP で実行する部分の実行時間 ET_{GPP}，LSRDP で実行する部分の実行時間 ET_{LSRDP}，LSRDP を利用する際に発生するオーバーヘッド ET_{oh} の和で表される

$$ET = ET_{GPP} + ET_{LSRDP} + ET_{oh} \tag{1}$$

ET_{LSRDP} を演算時間 T_{cal}，メモリアクセスによりストールした時間 T_{st} の和で表す．

$$ET_{LSRDP} = T_{cal} + T_{st} \tag{2}$$

アプリケーションは GPP で実行する部分と，n 回の LSRDP で実行される部分に分割されているとする．LSRDP は，FPU アレイ 1 行ごとのパイプライン処理を行う．1 クロックサイクルごとに 1 つのデータ集合を入力し，パイプライン段数分前の入力に対する演算結果を出力する．したがって，演算時間 T_{cal} は，LSRDP での実行ごとに連続して入力するデータ集合数 C_i と LSRDP の行数 A の和から 1 を引いた値に比例し，LSRDP の動作周波数 f_{LSRDP} に反比例する．

$$T_{cal} = \sum_{i}^{n} \frac{C_i + A - 1}{f_{LSRDP}} \tag{3}$$

LSRDP は連続する入出力の最初と最後のデータ転送のために，それぞれ LSRDP 間のレイテンシ Lat_{mem_LSRDP} 分ストールする必要がある．さらに，LSRDP が要求するメモリバンド幅 BW_{req_i} が主記憶のメモリバンド幅 BW_{mem} より大きいとき，1 つの入力データ集合ごとにそれぞれのメモリバンド幅の比に比例した分ストールする．また，BW_{req_i} が BW_{mem} より小さい場合は，入力データ集合ごとのストールは発生しない．

$$T_{st} = \sum_{i}^{n} \frac{2 \times Lat_{mem\_LSRDP}}{f_{LSRDP}} + (\lceil \frac{BW_{req\_i}}{BW_{mem}} \rceil - 1) \times \frac{C_i}{f_{LSRDP}} \tag{4}$$

LSRDP が要求するメモリバンド幅 BW_{req_i} を，1 クロックサイクルごとに入出力するデータサイズと LSRDP の動作周波数との積で定義する．

$$BW_{req\_i} = (input_i + output_i) \times f_{LSRDP} \tag{5}$$

ET_{oh} は，LSRDP で実行する各部分ごとの再構成時間 T_{rec_i}，GPP・LSRDP 間の通信時間 T_{tra_i}，LSRDP に入出力するデータを整列する時間 T_{sort_i} の和とする．

$$ET_{oh} = \sum_{i=1}^{n} \{T_{rec\_i} + T_{tra\_i} + T_{sort\_i}\} \tag{6}$$

3.3 評価結果と判明した問題点

各ベンチマークプログラムの実行時間を図 4 に示す．グラフ中の実行時間は GPP 単体における実行時間によって正規化されている．ERI を除く各アプリケーションについて，再配置時間 (図 2 の 1,6) が実行時間に対して支配的であった．ERI については GPP による計算部分が支配的となっているが，次いで再配置時間が多くの時間を占めている．これは，LSRDP の高い演算性能による性能向上に加えて，前処理など LSRDP を利用するためのオーバーヘッドによる性能低下が大きいことを示している．

この原因は，図 5 に示すように通常のプログラムが持つデータ構造と LSRDP による処理に適したデータ構造が異なることにある．したがって，主記憶上の別領域に新たに LSRDP

Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

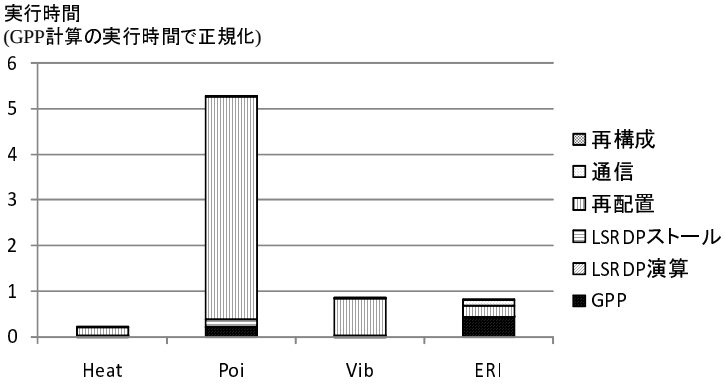

図 4 初期の性能評価結果
Fig. 4 Results of performance evaluation

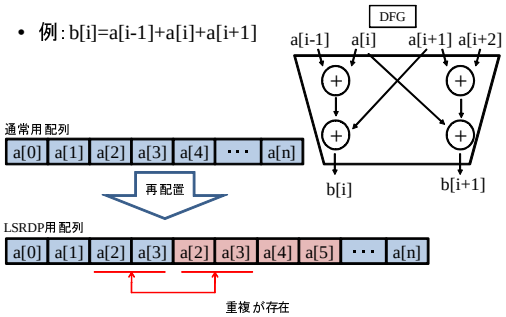

図 5 再配置によるデータ構造の変化
Fig. 5 Data rearrangement for LSRDP

用の配列を作成しなおす再配置の作業をプログラム実行時に行っている．しかしながら，低速な主記憶上で大量のデータの複製を行うためにどうしても処理時間が膨大になってしまう．

さらに，図 5 のような DFG の場合，LSRDP 用の配列はいくつかのデータを重複して持たなくてはならない．これは，LSRDP が連続アクセスで主記憶からデータを読み出しつつ，正しい計算を行うために必要であるが，メモリバンド幅を圧迫して性能を低下させる原因でもある．

4. 実行前処理削減手法

4.1 プログラム内のデータ構造の変更

前節で述べた問題点への対策として，図 5 に示すように，通常のデータ構造ではなく LSRDP による処理に適した構造を用いてプログラムを作成する事とした．この時，LSRDP に入力されるデータが，主記憶上で連続アクセス可能となるようにデータのパッキングを行っている．その結果，実行前処理が不必要となり大部分の実行時間を占めていた再配置の処理の削減が可能と考えられる．一方このデータ構造の変更によって，一部分のデータを重複して持つ必要性がある．したがって，必要メモリ量が増加し，また配列に存在するデータ順を複雑な順で保持する必要が生じる．更には，LSRDP により出力された重複の存在しないデータ列を再び LSRDP の入力とする際に，重複の回復が必要となる．この重複の回復処理については GPP と LSRDP のどちらかで行う場合がそれぞれ考えられるが，今回は LSRDP を使用することとした．そのため，計算対象となる 1 点のデータに対し，LSRDP での処理を差分法計算と並び替えの目的のために 2 回使用することになる．その結果，LSRDP の再構成時間と LSRDP の並び替え処理ならびにデータ転送時間がオーバヘッドとなるが，メモリに対するランダムアクセスの必要が無いためにこれまで支配的であった再配置時間と比較すると小さく，全体の実行時間としては改善が期待できる．他方，今回採用しなかった GPP により重複を回復する場合では結局その処理内容が再配置と同様となり，再び実行時間に対して支配的になってしまうことが予想される．

4.2 ハードウェアの追加

前節で示したソフトウェアのみでの提案手法では，配列に余分な重複が存在するためにそれを処理するためのオーバーヘッドが実行時間に含まれている．そのため，新規ハードウェアの追加によりデータの重複を不要とする手法を検討した．差分方程式を対象としたプログラムでは，一般に 1 つ前の LSRDP での計算の入力データと次の入力が同じ順序で重複し要求される．この特徴を考慮し，連続して入力されてくる重複を持たない入力データストリームを重複を持ったストリームへと変換する事を可能としたハードウエアを考案した（図 6）．このハードウェアは図 1 中の SMAC 内に設置を予定している．図 6 中で DRAM より供給された *input_data* は，下部のバッファを FIFO 動作で移動していく．コントローラはあらかじめ指定されたタイミングでバッファからレジスタへとデータを取り込む．その際にレジスタ上で重複を含んだ 1 列分の入力が揃うことになるために，レジスタから SB へ

Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

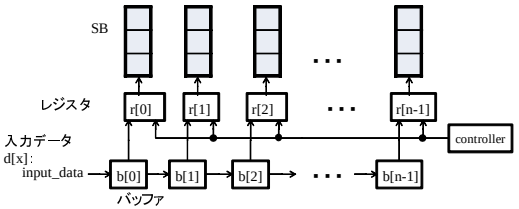

図 6 追加するハードウェアの概要
Fig. 6 The proposed hardware architecture

とデータをそのまま転送する．

性能モデリングでは，3.2 節，式 (4) の T_{st} が変更となる．

$$T'_{st} = \sum_{i}^{n} \frac{2 \times Lat_{mem\_LSRDP}}{f_{LSRDP}} + (\lceil \frac{BW'_{req\_i}}{BW'_{mem}} \rceil - 1 + C_{pad}) \times \frac{C_i}{f_{LSRDP}} \quad (7)$$

$$C_{pad} = interval - \frac{f_{newHW}}{f_{FPU}} \quad (8)$$

$$BW'_{req\_i} = (input_{i\_new} + output_i) \times f_{FPU} \quad (9)$$

$$BW'_{mem} = min(BW_{mem}, BW_{newHW}) \quad (10)$$

ここで，f_{newHW} は追加ハードウェアの動作周波数，$input_{inew}$ は追加ハードウェアを前提とした場合に LSRDP が 1 入力ごとに要求する新しいデータ数，BW_{newHW} は追加ハードウェアにおけるバッファのバンド幅を示している．

$interval$ については，レジスタがバッファからデータを読み込んだ後に再び読み込むまでの時間を示しており，1 回の入力毎にどれだけの入力データを入れ替えるかの個数に対応している．$interval$ が FPU に対する追加ハードウェアの動作周波数の比がよりも大きい場合にはその分のストールが発生する．BW'_{req_i} で示す新規の要求メモリバンド幅は，LSRDP が必要とする新しい入力データ数と出力データ数の輪と FPU の動作周波数の積となる．BW'_{mem} で示すメモリシステムの新規バンド幅は主記憶と追加ハードウェアのバッファのバンド幅の小さいほうとなる．

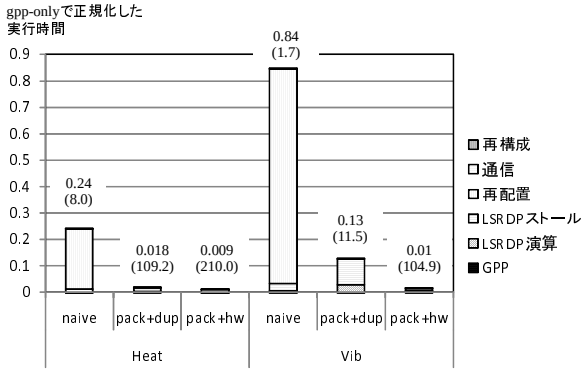

図 7 各評価対象モデルの実行時間と性能
（それぞれのグラフの上の数値は正規化された実行時間，括弧の中の数値は GFLOPS による性能値を示す）
Fig. 7 Results of performance evaluation employing the proposed techniques (the number on each bar represents normalized execution time by GPP and the number inside parenthesis stands for the performance in terms of GFLOPs)

5. 評価実験

5.1 評価環境

前節に示した実行前処理削減手法について評価実験を行った．実験環境としては 3.1 節の図 3 で示した初期評価と同じであるが追加ハードウェアの動作周波数は LSRDP と同一の 80GHz とした．評価対象アプリケーションとして今回は Heat，Vib への適用を行った．評価対象モデルとして，以下の 4 つを考えた．

gpp-only GPP 単体による実行モデル

naive LSRDP への従来手法による実装モデル

pack+dub ソフトウェアの提案手法であるデータのパッキングと重複データ保持を行ったモデル

pack+hw ソフトウェアの提案手法であるデータのパッキングとハードウェアの追加を行ったモデル

5.2 実験結果と考察

実験結果を図 7 に示す．各データは gpp-only の実行時間で正規化している．各データに対し括弧の中に全計算の GFLOPS 値を示した．ソフトウェアのみによる前処理削減提案手

Vol.2009-ARC-186 No.12
Vol.2009-HPC-123 No.12
2009/12/1

法 pack+dup では，naive 実装に比較して，実行時間が Heat, Vib それぞれについて 7%，15%と著しく減少した．これは再配列時間が削減されたためである．実行時間中の他の成分に着目すると，再構成時間や通信時間，LSRDP 計算時間，GPP 時間の全てについてオーバヘッドの処理が増加したため naive 実装に比較して増加している．しかしながらこの増加量は再配置削減の減少量に比較して小さく抑えることが可能との結果となった．Heat に比較して Vib の実行時間の削減比が少ない原因は，Vib は入力データの重複が多くその分のデータ転送のためストール時間が多くなっているためである．

ハードウェアを含めた前処理削減提案手法 pack+hw では pack+dup と比較して，実行時間が Heat, Vib それぞれに対し 52%，11%と，更に減少した．その結果，GPP と比較して 109 倍，76 倍の性能向上となった．これは主記憶から LSRDP へと転送するデータ量が減少したことでストール時間が減少したためである．また，データ重複の回復のために LSRDP を使用する必要がなくなり，LSRDP 演算時間についても減少している．

6. おわりに

本稿では，HPC を対象としたアクセラレータである大規模再構成可能データパス (LSRDP) の概要を解説し，実行時の問題点となっていた実行前処理の大きさについて述べ，その解決手法の提案を行った．その結果，実行前処理をソフトウェアにより直接処理するような実装において支配的なオーバーヘッドであった再配置時間を削減した．また，汎用プロセッサ単体での実行と比較して，熱伝導方程式と振動方程式それぞれについて 109 倍，76 倍の性能向上を達成した．

今回追加したハードウェアでは熱伝導方程式や振動方程式のような，LSRDP での計算において 1 つ前の計算における入力データと，次の入力が同じ順序で重複し要求される場合に有効である．しかしながら，他のプログラムにおいては，入力されるデータは重複しているが要求される順序が異なるような場合も存在する．したがって，そのような場合についてのハードウェアの拡張が必要であり，現在検討を行っている．

参 考 文 献

1) http://www.top500.org/
2) W. Wulf and S. McKee, “Hitting the Memory Wall: Implications of the Obvious,” ACM SIGArch Computer Architecture News, 23 (1):20-24, 1995.
3) 島崎慶太，長野孝昭，本田宏明，ファルハド メディプー，井上弘士，村上和彰，“大規模再構成可能データパスにおけるオンチップ・ネットワーク・アーキテクチャの検討”，情報処理学会研究報告，2007-ARC-173，pp.115-120，2007 年 6 月．
4) N.Takagi, K.Murakami, A.Fujimaki, N.Yoshikawa, K.Inoue, and H.Honda: “Proposal of a Desk-Side Supercomputer with Reconfigurable Data-Paths Using Rapid Single-Flux-Quantum Circuits”, IEICE Trans. Electron, E91-C, pp.350-355 (2008).
5) 片岡 広志, 本田 宏明, Farhad Mehdipour, 井上 弘士, 村上 和彰, “科学技術計算を対象とした大規模再構成可能データパスの性能評価”, 電子情報通信学会研究報告, CPSY2008-35, Vol.108, No.273, pp.35-40, Oct. 2008.

謝辞 下記 CREST プロジェクトにおいてご討論頂いております，名古屋大学大学院情報科学研究科，高木直史教授，高木一義准教授，同じく工学研究科，藤巻朗教授，赤池宏之助教，横浜国立大学工学部，吉川信行教授に感謝いたします．また日頃からご討論頂いております九州大学安浦・村上・松永・井上研究室ならびにシステム LSI 研究センターの諸氏に感謝します．本研究の一部は，科学技術振興事業団（JST）の戦略的創造研究推進事業（CREST）「単一磁束量子回路による再構成可能な低電力高性能プロセッサ」の支援によります．本研究は主に，九州大学情報基盤研究開発センターの研究用計算機システムを利用しました．