Vol.2022-ARC-247 No.10
Vol.2022-SLDM-197 No.10
2022/1/24

小容量キャッシュの追加による マルチバンク一次データ・キャッシュの 疑似デュアルポート化

依田 勝洋[1,a)]　吉川 隆英[1,b)]　塩谷 亮太[2,c)]　五島 正裕[3,d)]

概要：近年，疎行列を扱う処理を中心に不連続なデータデータにアクセスするギャザーロードの高性能化への要求が高まっており，これに対応して L1D キャッシュのマルチバンク化技術が提案されている．しかしながら，マルチバンク化によっても特定バンクへのアクセス競合が発生する場合には十分な性能向上を得られない．そこで，本稿では各バンクにバンクのメモリサイズよりも小さなサブデータアレイキャッシュ（SDAC）の設置を提案する．SDAC のデータ出力ポートは，競合を起こした当該バンクとは異なる隣接バンクの出力ポートとセレクタを介して合流するため，当該バンクは疑似的にデュアルポート化される．そして，アクセス競合が発生した時に SDAC は負けたアクセスが使うはずであったデータとそのアドレスをキャッシュし，再度同じアクセス競合が発生した際に隣接バンクのポートを使って競合アクセスを処理することで競合による性能低下を軽減する．本技術の競合解消の効果を HPC 分野で用いられる大規模疎行列へのアクセス時について見積もるため，HPCG 並びに SuiteSparse Matrix Collection に収録されているいくつかの疎行列に対するアクセス競合数を机上計算した結果，2 ライン程度の SDAC を各バンクに持たせることで競合の回数を最大 33% 削減出来ることが分かった．

1. はじめに

現代の HPC システムをターゲットとした CPU には並列演算能力を高めるために複数の SIMD 演算ユニットを持っている [1], [2], [3], [4]．GPGPU も同様に AI 向けデータ性能強化などの観点より，CPU よりもさらに強力な SIMD 演算ユニットを装備している．

SIMD 演算ユニットに対しては，複数のデータを一度に演算器に届けかつ効率的に捌く必要があるため，L1D キャッシュのマルチバンク化（以下マルチバンク L1D と表記する) が以前より採用されており，連続データアクセスに対しては高い性能を達成してきている．

しかし近年，不連続なデータアクセスをしばしば引き起こす疎行列処理に対する高速化への需要も高まってきている．例えば COVID-19 感染対策に活用され，2021 年ゴードン・ベル賞 COVID-19 研究特別賞を獲得した飛沫・エアロゾルの飛散シミュレーション [5] においても，その中に大規模疎行列を係数とする連立一次方程式の求解処理が含まれている [6]．

このような処理において頻出する不連続なデータへのアクセスに対して，マルチバンク L1D ではバンク衝突によるアクセス競合が多発するため，SIMD 演算ユニットを効率的に活用出来ず，十分な性能が得られないことが多い．

具体的な事例としては，2021 年 11 月に世界最高性能を達成したスーパーコンピュータ「富岳」は，連続データアクセスが多い TOP500 ベンチマーク [7] においては実行効率 84.7% を達成しているが，不連続データアクセスが多い HPCG ベンチマークの実行効率は 3.6% にとどまっている．

そこで本稿ではこのバンク衝突によるアクセス競合の性能影響を軽減するために，特に大規模連立一次方程式の求解に用いられる共役勾配法 (Conjugate Gradient: CG) における疎行列データへの不連続アクセスを対象としてバンク衝突の性能影響を精査した．

そして，この影響を軽減するためにマルチバンク L1D の各バンク（バンク内のメモリの実体をサブデータアレイ（SDA）と表記する）にサブデータアレイキャッシュ（SDAC と表記する）を付加することでバンク衝突の回避

[1] 富士通株式会社
Fujitsu Limited
[2] 東京大学
The University of Tokyo
[3] 国立情報学研究所
National Institute of Informatics
[a)] yoda.katsuhiro@fujitsu.com
[b)] yoshikawa.takah@fujitsu.com
[c)] shioya@ci.i.u-tokyo.ac.jp
[d)] goshima@nii.ac.jp

Vol.2022-ARC-247 No.10
Vol.2022-SLDM-197 No.10
2022/1/24

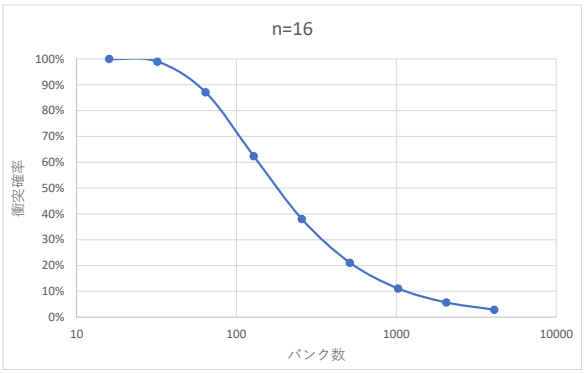

図 1 アクセス数を 16 に固定したときのバンク数と衝突確率の関係．衝突発生確率を下げるにはバンク数を対数オーダーで増やす必要がある．

する方式についての提案を行う．

以下，本稿では第 2 章でバンク衝突の詳細について考察し，第 3 章で解決手段となる SDAC の構成について説明を行い，次いで第 4 章でその効果についての机上計算結果を述べる．最後に第 5 章で全体をまとめる．

2. SpMV とバンク衝突

本章では，バンク衝突の発生頻度，その影響，および本稿の着目点について説明する．

2.1 衝突確率

アクセス数を n としたときのバンク衝突の確率は，バンク数を b とすると

$$1 - \left[\frac{b-1}{b} \times \frac{b-2}{b} \times \frac{b-3}{b} \times \ldots \times \frac{b-(n-1)}{b} \right] \quad (1)$$

と表すことができ [8]，$b = 32, n = 16$ とすると約 99% の確率でバンク衝突が発生する．

また**図 1** に $n = 16$ と固定したときのバンク数と衝突確率の推移をグラフ化した．例えば本稿の最大効果である 33% の衝突数の削減をバンク数を増やして実施しようとすると，256 バンク以上必要となるが，例えば AMD[4] の L1D は 8 バンクであることからこの数字は現実的ではない．

2.2 衝突の影響

次にバンク衝突が CPU の性能に与える影響について述べる．

図 2 に CPU のバックエンドにおける命令の流れとそのチャートを示す．CPU は 2-issue とし $I0$ 命令と $I1$ 命令が同時に実行され両命令の間でバンク衝突が発生し $I0$ が勝った状況を表している．このとき $I1$ 命令は EX ステージ手前で滞留する．すると後続の $I2, I3$ 命令も RR ステージに入れず白抜きのように滞留する．さらに後続の $I4$ 命令も IS ステージに入れず白抜きのように滞留する．そして，バンク衝突が解消されて $I1$ が勝った後，後続命令がパイプラインステージを進む．

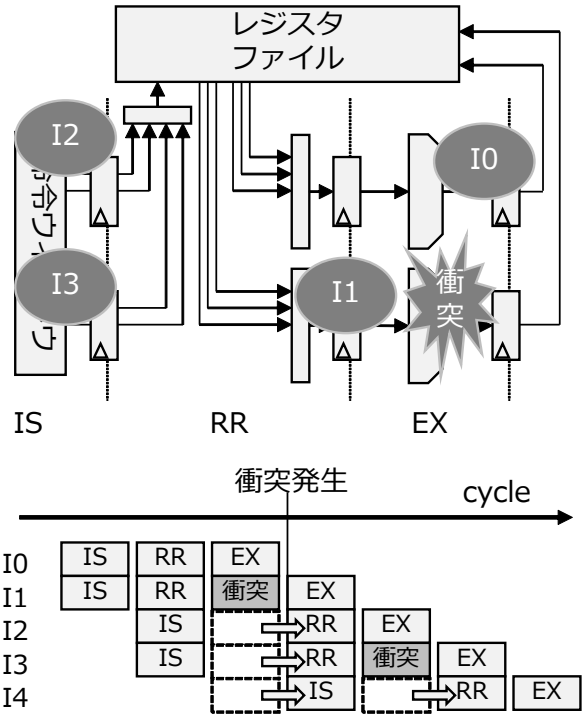

図 2 バンク衝突が発生した場合の後続命令への影響．上部ブロック図の I*は命令の進捗を表す．下部チャートは各命令の実行状態を時間方向に表記したチャートであり I0 が EX ステージの時刻に図上部で衝突が発生した場合，チャート白抜きのように命令の滞留が発生する．

図下部のチャートではさらに $I2, I3$ でもバンク衝突が発生した事象を描いている．この衝突によりさらに後続の $I4$ 命令が RR ステージ手前で滞留し，結果として $I4$ はバンク衝突により 2 サイクルに渡って実行を待たされる．

このように後続命令すべてがバンク衝突の影響を受ける．

2.3 対象アプリケーションでのバンク衝突

本稿では，大規模な疎行列ベクトル積 (**SpMV**: Sparse Matrix-Vector product) を対象アプリケーションとして取り上げる．1 章で述べた HPCG など，CG 法による連立一方程式の求解においては，SpMV が計算の大きな部分を占める．

A を計算対象の疎行列，x を SpMV 計算を行うベクトルと表記する．A は CSR 等の圧縮した形式で与えられ，A の各要素の値を $A.value$，列座標を $A.index$ の組として表現される．

さらに SpMV をプログラムイメージで行ごとの計算に展開すると以下のようになり，$x[A.i[col]]$ がギャザーロードとなる．

```
1  for(col=0;col<NONZEROSINROW;col++)
2    y += A.v[col] * x[A.i[col]];
```

Vol.2022-ARC-247 No.10
Vol.2022-SLDM-197 No.10
2022/1/24

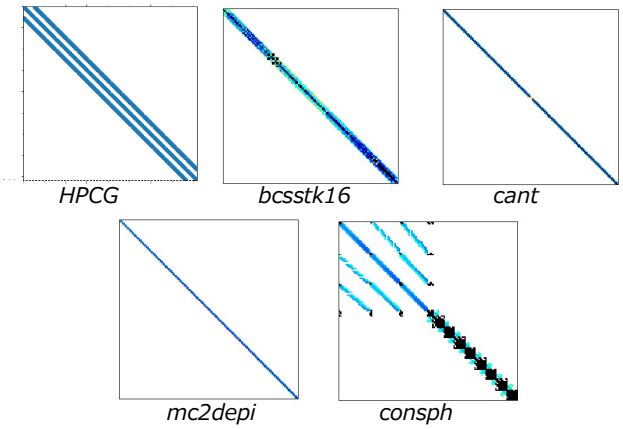

図 3 疎行列の例．縦軸が行，横軸が列を表し，非圧縮形式での非ゼロ要素を色付けしている．白塗りはゼロ要素でありこの比率が大きいことが疎行列の特長である．

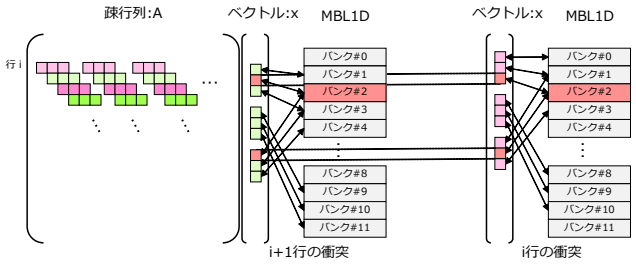

図 4 SpMV 計算の行列イメージとベクトル x の要素が各バンクにどのようにアクセスするかの 1 例を示した図．ベクトル x とそのバンクアクセスの様子を行列 A の i 行目と $i+1$ 行目の計算に関し図で示した．i 行目の計算時にベクトル x はバンク ♯1 とバンク ♯2 で衝突を起こしている．また $i+1$ 行目の計算時にはバンク ♯2 とバンク ♯3 で衝突を起こしている．バンク ♯2 での衝突に着目すると実はベクトル x の同じ要素同士が衝突を起こしている．

この疎行列の特長は，ゼロ要素の数の比率が非常に高いことと対角行列であることである．SuiteSparse Matrix Collection[9] に疎行列のサンプルが多数集められており，[10] はそれらを分類し代表的な疎行列に対してベンチマークを取得している．集められた疎行列のゼロ要素を白，非ゼロ要素をヒートマップで色付けして行列イメージとしたのが**図 3** である．非ゼロ要素が一つあるいは複数の対角線上に並んでいる．

2.4 着目点

一般に SpMV の計算において，$A.value$ と $A.index$ はメモリ上に連続に格納されかつ連続アクセスされるためマルチバンク化によりバンク衝突を起こさない．さらに先のアドレスのデータを取得する単純なプリフェッチャーを設置すればキャッシュミス率を低く抑えることができる．

これに対し x はメモリ上に連続に格納されているが $A.index$ で指定された箇所にアクセスされるため連続アクセスとはならず，バンク衝突が発生する．

図 4 に SpMV 計算時の非ゼロ要素の配置例とバンクアクセス例を示す．

$A.index$ のとびとび度合が直感的に伝わるように A の非ゼロ要素を図中に圧縮しない形式で表現した．対応する x の要素をベクトル x の中に色付けして表現した．図右では i 行目の計算でアクセスされる x の要素を，図中央では $i+1$ 行目の計算でアクセスされる x の要素を色付けした．さらにベクトル x の横にアクセスするバンクを表現した．

通常対角要素は 1 つとは限らず複数の要素から構成されている．そして x の連続した要素は隣り合うバンクに格納され，離れた要素同士が同一バンクに当たることがしばしばある．

そこで本稿では対角成分は次の行の計算の時に 1 列ずれる，という状況に着目した．

例えば図 4 の右に i 行目の計算時のバンクアクセスを重ねて描くことで i 行と $i+1$ 行の非ゼロ要素の位置と使用されるバンクとが重なることがわかる．バンク ♯2 は i 行目と $i+1$ 行目の両方でバンク衝突を起こしており，なおかつ同一の $index$ で衝突を起こしている．

この現象は SpMV の計算が次の行では一つ隣の列の要素を使用し，かつ複数の要素が連続したブロックを形成しているために発生する．前者は図 3 が左上から右下に向かう線状であることを意味し，後者はその線に太さがあることを意味する．

したがって図 4 の例が特殊なのではなく図 3 の他の疎行列内でも発生しうる．

3. 提案手法

本章では提案手法について説明する．提案手法のポイントは 2 つある．

1 つ目は，各 SDA に設置した 2〜8 ラインの SDAC がバンク衝突の負け側を拾うという点である．

2 つ目は，各バンクが SDA 用と SDAC 用の 2 つのポートを持ち，SDA と SDAC の両方のデータを同時に出力する点である．

同様の効果は，SDA をデュアルポート化することで得られる．しかしながら提案手法によれば，回路面積はデュアルポート化より少なくクロスバーはデュアルポート用の時より少ないセレクタ数で実現可能である．その手法について説明する．

3.1 マルチバンク L1D の構成

図 5 にマルチバンク L1D を設置した場合の CPU のバックエンドのブロック図を示す．

バックエンドではスケジューラから各スカラー及びベクトルユニットへ命令が発行される．特にロードストア

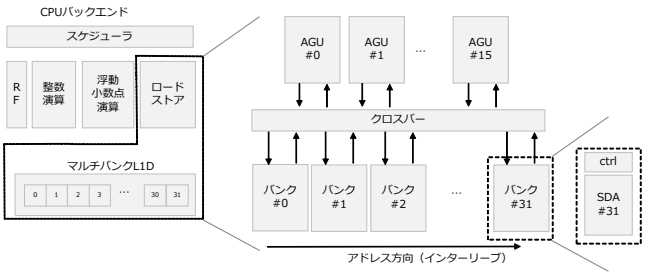

図 5 マルチバンク L1D を実装したときの CPU のバックエンドのブロック図．図左はバックエンドの概要を表しており，その中のロードストアユニット，クロスバー，マルチバンク L1D を拡大したのが図右である．AGU はアドレス生成ユニットでありクロスバーを通じて全バンクと接続している．さらに各バンクは SDA というメモリ本体と制御機構（ctrl）から構成されている．

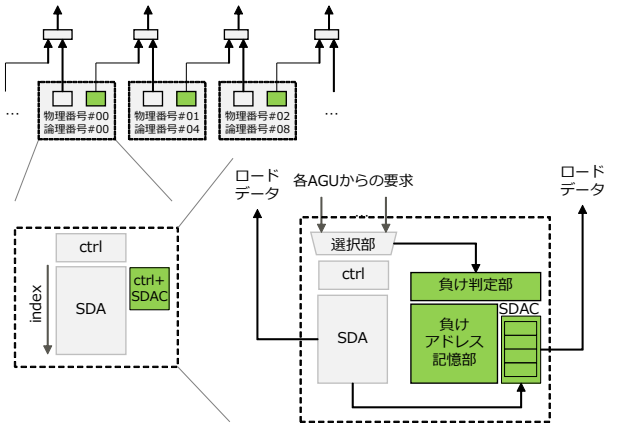

図 6 提案手法のブロック図．上部はポートの共有を物理的に隣接セルにおいて可能とするため各メモリセルの論理および物理番号の配置を変えた工夫である．左は対象となるバンクの 1 つであり，右でその内部に負け判定部，負けアドレス記憶部，およびデータを格納する SDAC 本体を追加したことを表している．さらに SDAC から別系統のポートがでておりこれが図上部でポートを共有している箇所の 2 本目の矢印を表している．

ユニットは（図右）例えば 16 個（以降 VL 長 16 と表記，VL=Vector Length）の AGU（Address Generation Units）を持ち，各 AGU から 32 個のバンクへクロスバー接続される．各バンクは内部に制御部とメモリの実体（SDA:サブデータアレイ）を持つ．アドレスはバンク番号方向に割り振られておりインターリーブと呼ばれる．

3.2 バンク内部の構成

図 6 にブロック図を示す．

図左，従来の各バンクに「(図中，ctrl+SDAC)」を設置する．SDAC 部は 2〜8 程度のラインから構成され複数のアドレス対応したデータを格納する．選択部は従来通りアクセス競合があった場合に各 AGU からのアクセス要求から勝ちとなるアクセスを選択する．

本稿ではさらに 2 位で負けたアクセスのデータを負け判定部に送信する．負け判定部は負けアドレス記憶部を参照し，アドレス情報と一致するアドレスがあった場合には，SDAC ヒットと判断し SDAC から該当するデータを出力する．

そのため AGU にあるアービタから 2 位のアクセス情報を各バンクに送る配線を追加する．そしてもしアドレス情報と一致するアドレスがなかった場合には負けアドレス記憶部に今回の負けアドレス情報を上書き保存し，SDAC の対応するラインのデータも更新する．

そうすることで図 2 の下部の命令の滞留を無くし命令の実行を詰めることができる．

SDAC の容量が SDA の容量と比べて小さくても効果が得られるということは 4 章で述べる．

3.3 ポートの構成

本節では疑似デュアルポート化の工夫について説明する．

図 6 上部のように SDAC を N 個先隣接バンクのポートと共有させる．図は 4 個先バンクと隣接した例である．

より一般的な表現をするためにバンク番号を数列

$$\{0, 1, 2, 3, ..., 31\} \tag{2}$$

とする．まず N の剰余を取って 0 となるものを並べる．N=4 とすると

$$\{0, 4, 8, 12, ..., 28\} \tag{3}$$

となる．最後の 28 は 0 と共有する．次に N の剰余を取って 1 となるものを並べる．

$$\{1, 5, 9, 13, ..., 29\} \tag{4}$$

最後の 29 は 1 と共有する．という作業を N 回繰り返すと

$$\{0, 4, ..., 28\}, \{1, 5, ...29\}, \{2, 6, ..., 30\}, \{3, 7, ..., 31\} \tag{5}$$

となる．メモリの物理番号は（2），論理番号は（5）として並べると図 6 の上部の並びとなる．

この論理番号の離れたバンク同士のポートを共有させることで，2.4 節で述べた対角成分が複数の連続した要素で形成されている場合に，隣接ポートが空いていないという状況を緩和する．

N の値はハードウェア固定である．なぜなら動的可変とした場合デュアルポートのクロスバーと同じとなってしまうからである．N の値は 4 章で決める．

4. 評価

本章では提案手法の評価結果を述べる．

4.1 疎行列の選択

2 章でも述べた SuiteSparse Matrix Collection[9] をもと

表 1　対象とした疎行列の主なパラメータ．種類，行数がなるべく重ならないように抽出した．cant と consph のパラメータは似ているが非ゼロ要素の配置が異なる．

疎行列	行数	種類
HPCG	2,097,152	流体計算等（問題サイズ 128）
bcsstk16	4,884	Structural Problem
cant	62,451	2D/3D Problem
mc2depi	525,825	2D/3D Problem
consph	83,334	2D/3D Problem

表 2　提案手法の評価パラメータ．VL は 8 レーン，2-issue 相当の 16 を中心に振った．ライン数は回路面積に直結する．そのため最低いくつ必要かを測定する．隣接先はポートの共有先である．ハードウェア固定するためシミュレーションで決めておく必要がある．

	値
VL	8,16,32
SDAC ライン数	1,2,3,4,5,6,7,8
隣接先	1,2,3,4,5,6,7,8

に図 3 で示した疎行列について提案手法を評価した．**表 1** に抽出した疎行列の行数と用いられる計算の種類を挙げた．行数は *index* の分布度合に影響すると考えられるため数千から数百万オーダーまで種類を分散して抽出し，種類も固まらないようにした．

表 2 にまとめたように，評価パラメータは VL 長を既存の 16 を中心に振り，SDAC 固有のパラメータである SDAC のライン数は 1 から 8 まで振り，ポートの隣接先は 1 から 8 まで振った．また SDAC の更新は FIFO 置換とした．

4.2　評価方法

評価は hpcg-benchmark[11] のソースコードを改造してデータアクセスログを採取できるようにし，様々な種類の疎行列に対して CG 法処理の実行後にログを解析してアクセス競合回数を数え上げる机上計算で行った．詳細を以下に示す．

1 個の *index* が 1 個の 8-byte データのオフセットを指しており，各バンクは 8-byte 単位でインターリーブされている．そこで *index* に対してバンク数で剰余を取ることでデータのアクセス先バンク番号を計算した．

そして行ごと，VL 長ごとに同一バンクへのアクセスが最大で何重衝突発生したかを計算し，疎行列の全行について合計した．

例えばバンク ♯0 で 3 重衝突，バンク ♯2 で 2 重衝突が発生した場合には，最大を取って 3 重衝突と数えバンク衝突のペナルティを 2 と計算した．

また提案手法をシミュレートするにあたっては，バンク競合が発生しかつ SDAC に前回負けたアクセスのデータがあった場合を SDAC ヒットと数え，さらにこのとき隣接 N 個先のポートが使用されていなければ先ほどのペナルティを 1 減らす，と計算した．

例えばバンク ♯0 で 3 重衝突，バンク ♯2 で 2 重衝突が発生しバンク ♯0 のみ SDAC ヒットとなった場合には，最大 2 重衝突と数えバンク衝突のペナルティを 1 とした．バンク ♯2 のみ SDAC ヒットとなった場合には，最大 3 重衝突はかわらないためバンク衝突のペナルティを 2 とした．

もし SDAC ヒットしなければ提案手法の効果はなかったとして衝突数を変更しなかった．

またアクセス競合が発生した場合には SDAC に負けたアクセスのデータを FIFO 置換して格納した．

この机上計算をもとに表 1 の疎行列に対して，表 2 のパラメータで，総衝突回数が提案手法でどれだけ減ったかを評価した．

4.3　評価結果

まず，表 2 の SDAC ライン数と隣接先のパラメータを決めた．

事前の調査でどちらのパラメータの値が大きいほど性能への効果が大きい傾向にあることがわかっていたため，片方のパラメータを最大値に固定して対象のパラメータを評価した．パラメータの評価結果が**図 7** である．

結果ライン数は図 7 より 2 以上でほぼ効果が飽和した．また隣接バンクとの距離は図 7 より 6 以上離せば効果があることがわかった．そこでライン数は 2，ポートの共有先を 6 つ隣のバンクとして性能を評価した．

次に，決めたパラメータで各疎行列の机上計算を行った．

適用後のバンク衝突の削減比率を示す**図 8** は，提案手法適用前のバンク衝突数を 1（before）として適用後に減った比率を疎行列ごと，VL 長ごとに並べたものである．

HPCG での効果が一番大きく VL=8 のときに約 33% の衝突を回避している．次に回避した比率が高いのは cant であり VL=32 のときに約 29% の衝突を回避している．また mc2depi の効果がないように見える理由は，そもそもバンク衝突がなかったためである．

4.4　回路規模の増加に関する考察

本章では本稿の機能追加による回路規模の増加について考察する．

従来技術の延長線上で提案手法と同等の機能を実現しようとした場合，マルチバンクの各 SDA をデュアルポート化し，それに合わせてクロスバーのポート数を 2 倍とする方法が考えられる．これに対して本稿がどの程度回路規模を削減できているかを説明する．

まず，SDA の面積はデュアルポート化により行と列の方向に 2 倍となるためおおよそ 4 倍となる．これに対し 4.3 章より 1 バンク当たり 2 ラインあれば十分という結果が得られた．

例えば A64FX[1] の L1D キャッシュは 64KB，1 ライン=256byte，VL=8，2-issue である．本稿のシミュレーショ

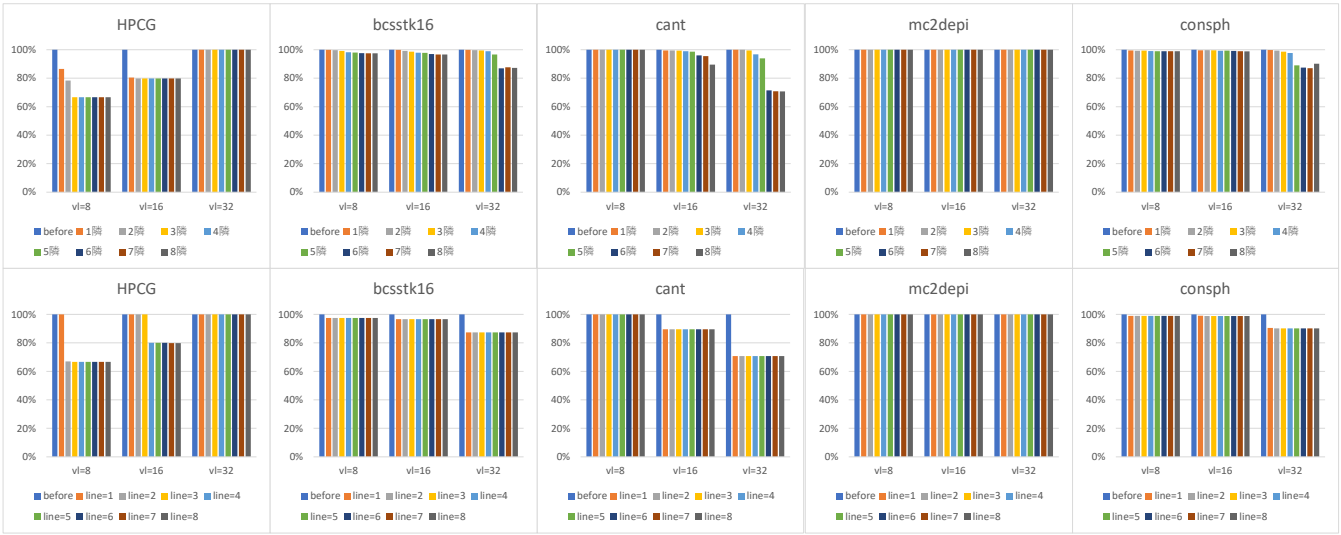

図 7 パラメータ決定のため疎行列ごとに取得した評価結果．上がライン数を最大にして隣接距離を振った結果である．下が隣接距離を最大にしてライン数を振った結果であり，縦軸は before（適用前）の総バンク衝突数を 100% とし，本稿を各パラメータで適用した場合に何 % まで下げることができたかを表している．VL ごとに右に行くほど効果が高い傾向がある．

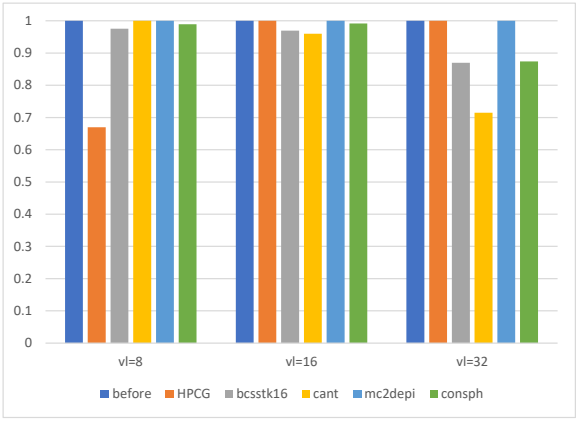

図 8 提案手法なしの場合のバンク衝突数（before）を 100% とした場合の適用後の衝突数の比率．左端の before の棒に比べて低いほど効果が高い．

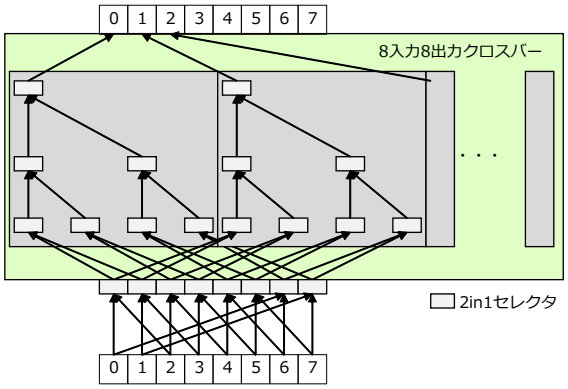

図 10 本稿を適用した場合のセレクタの構成例．8 ポートを例に示している．8 入力から 1 入力を選択するツリー状の構造を 1 つに減らすことができる．その分増える下部のポート共有用のセレクタは従来例の最終段のセレクタの削減で相殺される．

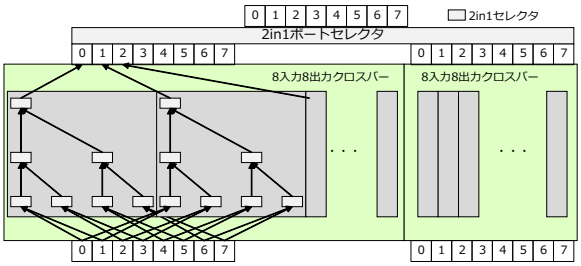

図 9 メモリセルをデュアルポート化した場合のクロスバーセレクタの構成例．8 ポートを例に示している．8 入力から 1 入力を選択するツリー状の構造が 8 ポートの 2 倍分存在する．

ンの VL=16，バンク数=32 に相当する。すると 1 バンク当たり 2KB，8 ラインとなり 1 バンク当たり 2 ラインの増加（25% の増加）となる．

比較すると 4 倍対 1.25 倍となり本稿が有利となる．

次に，セレクタの規模について比較する．

図 9 はデュアルポートメモリ 8 バンクを 8 個の AGU と接続した場合の 2in1 セレクタの実施例である．8 入力 8 出力のクロスバーがデュアルポートのため 2 セットある．その上 AGU のポート選択用の 2in1 セレクタが必要である．

これに対し**図 10** は本稿適用後のクロスバーの実施例である．図下のマルチバンク L1D 側のポートは SDA と SDAC の 2 ポートを持つが隣接ポート（本例では 6 つ隣）とポートを共有するためセレクタは 8 つで済む．その上に 8 入力 8 出力のクロスバーが 1 セットある構成となる．

ポート数を m として 2in1 セレクタの数を比較すると図 9

は $2m^2 - m$，図 10 は $2m$ となりおよそ半分の回路面積となる．

実際，クロスバー内部の 1 ツリー構造はセレクタの段数が進むごとに初期値 1，比率 2 倍の等比級数的にセレクタの数が増加し段数は $log_2 m$ である．等比級数の総和の公式 $S(n)$ に当てはめると図 9 は次のようになる．

$$\begin{aligned} \text{従来技術のセレクタの数} &= S(n)(2m) + m \\ &= \frac{a(1-r^n)}{(1-r)}(2m) + m \\ &= \frac{(1-2^{log_2 m})}{-1}(2m) + m \\ &= 2m^2 - m \end{aligned} \tag{6}$$

これに対し図 10 の場合は次のようになる．

$$\begin{aligned} \text{本稿のセレクタの数} &= S(n)m + m \\ &= m^2 \end{aligned} \tag{7}$$

5. おわりに

本稿では，近年需要が高まっているが，L1D キャッシュのバンク衝突によって実行効率が上がらない，不連続データアクセスを伴う疎行列演算などの処理に対して，バンク衝突の影響を軽減して性能を高めるため，L1D に各バンクにサブデータアレイキャッシュ (SDAC) を付加する提案を行った．

SDAC は L1D キャッシュの各バンクのメモリサイズよりも小さく，専用のデータ出力ポートを持ち，衝突が発生した当該バンクに隣接した隣接 L1D バンクの出力ポートとセレクタで合流することで，当該バンクを擬似的にデュアルポート化する．そして，SDAC はアクセス競合が発生した際に，調停で負けたアクセスのデータとアドレスをキャッシュし，再度同じ競合アクセスが発生したときには，キャッシュされたデータを隣接バンクのポートから出力することで，衝突による性能劣化の低減を実現する．

そして，本技術の競合解消の効果を HPC 分野で用いられる大規模疎行列へのアクセス時について見積もるため，HPCG 並びに SuiteSparse Matrix Collection に収録されているいくつかの疎行列に対するアクセス競合数を机上計算した結果，2 ライン程度の SDAC を各バンクに持たせることで競合の回数を最大 33% 削減出来ることを確認した．

今回の評価では疎行列のデータ格納方式として代表的な CSR を用いて評価を行った．しかし，データアクセスのパタンは格納方式によっても変わるため，今後は CSR 以外の様々な格納方式に対しての効果の検証も行っていく．

参考文献

[1] Yoshida, T.: Fujitsu High Performance CPU for the Post-K Computer, Hot Chips 30 (online), available from ⟨https://www.fujitsu.com/downloads/SUPER/hotchips/20180821hotchips30.pdf⟩ (accessed 2021).
[2] ARM Ltd.: *ARM Architecture Reference Manual Supplement – The Scalable Vector Extension (SVE), for ARMv8-A, issue A.i* (2021).
[3] Intel Corp.: *Intel Architecture Instruction Set Extensions and Future Features Programming Reference* (2020).
[4] AMD Inc.: *Software Optimization Guide for AMD EPYC 7003 Processors* (2020).
[5] Takenouchi, T.: Supercomputer Fugaku wins prize for COVID-19 research, The Asahi Shimbun (online), available from ⟨https://www.asahi.com/ajw/articles/14485778⟩ (accessed 2021).
[6] Tsubokura, M.: HPC for the Prediction and Countermeasures of Virus Droplet Infection in Indoor Environment on the Supercomputer "Fugaku", Kobe University (online), available from ⟨https://arxiv.org/pdf/2110.09769⟩ (accessed 2021).
[7] TOP500.org: HPCG – November 2021, TOP500.org (online), available from ⟨https://www.top500.org/⟩ (accessed 2021).
[8] Tretter, A., Kumar, P. and Thiele, L.: Interleaved multibank scratchpad memories: A probabilistic description of access conflicts, *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, IEEE, pp. 1–6 (2015).
[9] Tim Davis, Texas A&M University, Y. H. Y. L. and Kolodziej., S.: SuiteSparse Matrix Collection, Formerly the University of Florida (online), available from ⟨https://sparse.tamu.edu/⟩ (accessed 2021).
[10] Kreutzer, M., Hager, G., Wellein, G., Fehske, H. and Bishop, A. R.: A unified sparse matrix data format for efficient general sparse matrix-vector multiplication on modern processors with wide SIMD units, *SIAM Journal on Scientific Computing*, Vol. 36, No. 5, pp. C401–C423 (2014).
[11] Dongarra, J., Heroux, M. and Luszczek, P.: hpcg-benchmark, The High Performance Conjugate Gradients (HPCG) Benchmark project (online), available from ⟨https://github.com/hpcg-benchmark/hpcg⟩ (accessed 2021).