Vol.2017-HPC-158 No.15
2017/3/9

疎行列系アプリケーション性能の主記憶遅延増加の影響評価

田邊 昇[†1, a)]　遠藤敏夫[†1, b)]

概要：Intel/Micron 両社が近々の発売をアナウンスしている DIMM 型の 3D Xpoint のように、DRAM と NAND Flash の中間に位置する(中遅延で)大容量なメモリが、PC サーバ等の主記憶に利用される日が近づいている。主記憶の巨大化はページフォルトリスクを減らす一方、アクセス遅延増加により性能劣化が生じる可能性がある。上記メモリ階層は NAND Flash より大幅に低遅延で、主記憶代替利用におけるソフトウェアオーバーヘッドを回避可能であるため、対応可能な用途は広がる可能性が高い。本研究では、主記憶の遅延増加に伴う処理性能へのインパクトについて、多様なアプリケーションに相当する様々な疎行列を用いて調査する。

1. はじめに

ほとんどの計算機の主記憶には非常に長い期間にわたり DRAM が用いられてきた．アクセス遅延時間とビット単価の兼ね合いから，DRAM 以外の選択肢が実質的に無かったためである．

一方，2015 年 7 月に Intel および Micron の両社は 3D Xpoint[1]と称する新型メモリの開発成功と翌 2016 年の市場投入を発表した. 3D Xpoint は DRAM の 10 倍の記憶密度，NAND 型 Flash メモリの 1000 倍のアクセス速度と書込み耐久性を有する．3D Xpoint を搭載するメモリモジュール(DIMM)や，それに対応した Intel 製サーバの 2017 年の出荷もアナウンスされた．CPU からキャッシュライン単位でアクセス可能な主記憶の，大部分の容量を 3D Xpoint が構成する日が近づいてきている．

特に HPC より大きな市場規模を有するクラウドやビッグデータ解析用途の大容量主記憶ニーズにドライブされて，3D Xpoint を主記憶とする計算インフラが増加し，3D Xpoint ベースの主記憶がコストパフォーマンス的に有利な状況になる可能性が高い. 過去にも元来 HPC 用に設計されていなかったサーバ用 CPU が，その高いコストパフォーマンスを原動力に PC クラスタのエンジンとして HPC 用に転用された．その歴史と同様に，上述のような計算インフラが HPC のアプリケーション実行に転用されるという歴史が繰り返される可能性が高まっている．

ただし，3D Xpoint のアクセス遅延は Flash よりは桁違いに速いが DRAM より数倍遅いことが予測されている．IDF2015 における Intel によるプレゼン資料[2]の内容が現時点でも正しければ 250ns 程度となる見込みである．従来のアプリケーションは，そのようなメモリ階層の存在を想定していない．アプリケーションの実行速度は消費エネルギーとも密接に関係するため，どの程度実行速度に影響が出るかを見極めないと，HPC 用途への転用が真に高いコストパフォーマンスを実現できるか現時点ではわからない．何らかの課題が出現するのであれば，その解決への研究も進めていくべきである．

本研究では上記のような記憶階層における劇的な変化を鑑みて，今後 HPC 系アプリケーション，とりわけ重要な柱である疎行列系アプリケーションにおける性能へのインパクトについて評価を行う. そのような評価を行うにあたり，現段階では 3D Xpoint を主記憶とする計算インフラの実機は，主に巨大クラウドベンダーにおける評価用に用いられていると言われており, 筆者を含めた大半の HPC 関係者が使えない状況にある．そこで，当面は仮想的な評価環境を構築する必要がある．

現段階では筆者らも予備評価[3]において試した MARSSx86[4][5]や gem5[6][7]などのサイクルアキュレートなアーキテクチャ研究用シミュレータを用いる方法がある．その方法は非常に正確である反面，あまりに実行時間がかかり過ぎで，HPC のような大規模問題に対応困難で，評価スループットが極端に低くなる．大容量メモリが対応するような大規模データを処理するアプリケーションの網羅的評価はほぼ不可能と言える．

一方，主に不揮発メモリを高度にサポートする OS やデータ解析系アプリの研究用途にツールが開発・利用されており，中でも Quartz[8]が有名である．筆者らはその本格的使用を行うつもりで予備実験や精度検証に多大な労力を傾けた．ところが，少なくとも筆者らの環境では十分な精度が確認できない等の重大な問題があった．そこで筆者らは新たな解決法を考案し，予備評価時とは比較にならないほどの膨大な評価結果を非常に短期間で得ることができた．

本研究の貢献は以下のとおりである．

(1) Quartz より優れたスループット，予測精度，適用範囲，測定環境の柔軟性，環境構築容易性を有する，主記憶遅延増加時のアプリケーション性能予測方法の提案と，その精度評価
(2) 上記手法を応用した主記憶遅延増加時の疎行列系アプリケーション性能の網羅的評価と，それらから得られた知見

本報告の構成は以下のとおりである．まず，2 章では主記憶遅延影響の評価環境やそれらを用いた関連研究を概観する．3 章では，提案する主記憶遅延影響予測手法につい

†1 東京工業大学
Tokyo Institute of Technology
a) tanabe.n.ac@m.titech.ac.jp
b) endo@is.titech.ac.jp

て述べる．4 章では，その精度に関する検証実験を示す．5 章では，疎行列ベクトル積(SpMV)を中心とした各種疎行列系アプリケーション性能の網羅的評価を示す．6 章では，それらから得られた知見について考察し，7 章でまとめる．

2. 関連研究

本章では既存の主記憶遅延影響の評価環境や，それらを用いた関連研究を概観する．

リアルタイムで動作する Intel のハードウェア式エミュレータ HMEP を用いた先行研究[9]-[13]がある．ただしこのハードウェア式エミュレータは Intel 関係者のクローズドな評価環境であり，一般の研究者が容易に利用できるものではない．遅延可変域は 300ns～500ns と狭い．ただし，大規模グラフ解析やデータベースや OS の研究に適用した研究はあるが，HPC 系アプリに適用した例は見当たらない．

サイクルアキュレートな自作シミュレータを用いる方法は主に PCM を想定した研究[14]-[17]が多くなされた．当時の PCM の遅延は楽観的で比較的 DRAM に近く，3D Xpoint の遅延とはかけ離れていると思われる．PCM 等の遅延で HPC 系や BD 系アプリの評価を取った論文[18]も存在するがリード 21ns，ライト 100ns と仮定しているため，3D Xpoint の遅延とはかけ離れていると思われる．また，模擬スループットに問題があると考えられ，本研究のように疎行列の網羅的な評価はなされていない．git にて公開されている MARSSx86 を用いた先行研究としては東芝の Coolchips'16[19]および筆者の予備評価[3]がある．これらは測定スループットが低いため，本研究が目指す網羅的な評価には向かない．

リアルタイムで動作する評価環境としては HP の NVMpro[20]とその改良版の Quartz[8]と東京農工大による改良版[21]がある．東京農工大の改良版はライト遅延をリードとは独立に設定できるが Sandy Bridge のみに対応しており，かつ未公開である．Quartz は NVMpro の進化版であり，git にて公開されている．ただし，筆者らの Haswell ベースの測定環境では動作が不安定である．

3. 提案する主記憶遅延影響予測手法

本章では提案する主記憶遅延影響予測手法と他の既存手法との比較について述べる．

3.1 提案手法

提案する主記憶遅延影響予測手法は，Quartz の基本動作原理を踏襲しつつも，収集するデータ(大きな遅延を有する主記憶におけるアプリケーション性能)を取得することに特化して動作や測定法を簡略化するものである．これにより，HPC や計算機に詳しくない一般ユーザにも自分の手持ちのアプリケーションが遅延に対してどれ位影響が出るのかを手元で気軽に試すことができるようになる．

提案手法は以下の 3 ステップに集約される．

(1) perf コマンドでアプリの LLC ミス数と実行時間を測定
(2) (測定環境毎に 1 度だけ) Intel Memory Latency Checker 等で測定環境の DRAM 遅延を測定
(3)予測実行時間＝現環境の実行時間＋(想定外部メモリ遅延－現環境の DRAM 遅延)×LLC ミス数

以下,本手法を解説する．本手法の最大の特徴は Linux の標準的なディストリビューションに入っている perf コマンドを利用することである．すなわち，改めてインストールする必要がない環境も多いと思われる．もし，ない場合は sudo yum install perf (ubuntu では sudo apt-get install linux-tools-[カーネルの版数])を実行するだけで入る．

perf コマンドは Intel の VTune Amplifier やフリーソフトの PAPI などの本格的な性能評価ツールと同様に，Intel アーキテクチャ CPU の PMC(Performance Monitoring Counter)等を読み書きして実行サイクル数や LLC(Last Level Cache)ミス回数などを手軽に安定して測定できるものである．

前述の測定すべきこと(1)は以下のコマンドで簡単に取得できる．

```
perf stat -e cache-misses 被測定アプリ起動コマンド
```

上記を実行するとエラー出力(何も指定しなければ標準出力の最後)に例えば

```
 Performance counter stats for
'graph500-2.1.4/seq-csr/seq-csr -s 18':
       134,769,394     cache-misses
      21.573263326 seconds time elapsed
```

というような結果が出力される．ここで表示されている cache-misses は LLC においてミスした回数であり，その回数だけキャッシュラインのリプレース動作が CPU において実行されたということを意味する．Quartz が本来出力すべき NVM access という指標はこの値と同程度になることが期待されるが，少なくとも筆者がインストールした環境では，原因は特定できていないが Quartz の方が桁外れに少なかった．どちらがより正確かは 4 章で評価を行う．

ここで，1 回の LLC ミス時にどれ位のペナルティがかかるかは CPU の実装依存になるが，リード遅延とライト遅延がほぼ同一という条件下では遅延という観点では，実験の結果 1 回分のメモリアクセス遅延で良い．

その定性的解釈としてはは，ミスラインの外部メモリへのリード要求を出しつつ，LRU で追い出すラインを決定し，それをライトバッファに挿入し，空きがあれば確実に後ほど外部メモリに追い出されるラインを書き出せるので，LLC の該当キャッシュラインはリードの応答を上書きできる状態に移行できる．つまり「書込みバッファが一杯でないケースが大半」という条件下で CPU のストールをもたらす遅延はリード遅延 1 回分しかかからないと近似できると考えられる．

逆に，提案手法で誤差が入り込む余地は「書込みバッフ

ァが一杯でないケースが大半」という条件が成り立たないアプリケーションの実行時に，書込みバッファ起因のストールをカウントしていないことによる過小評価が発生する．perf list で表示される pre-defined events には，書込みバッファが一杯で発生したストールサイクル数は入っておらず，これを容易に測定することができない．

なお，perf コマンドでは LLC ミスを load と store で区別して測定できるが，ライトバックキャッシュの場合，ロード系命令によって引き起こされた LLC ミスも，ストア系命令によって引き起こされたミスも，上記のリプレース動作は全く同じであるので，特に区別する必要はなく，*cache-misses* だけ測定すれば良い．

上記の測定値から挿入すべき遅延を得るためには評価環境において Intel Memory Latency Checker 等を用いて，一度だけ主記憶遅延を測定しておく必要がある．ダウンロード後に展開して linux フォルダにバイナリ mlc があるのでそこで sudo ./mlc で最初に主記憶遅延が表示される．筆者の環境では本ツールではローカルが 98ns，リモートが 139ns と表示されたが，Quartz に付属の memlat ベンチマークでは 115ns と表示され少しずれがあるものの，大差はない．おそらく後者は 2CPU ソケットに付属するメモリの平均遅延が採用されていると考えられる．

1 スレッド実行の場合は，想定する外部メモリの遅延から測定環境の DRAM 遅延を引いた遅延差に *cache-misses* を乗じたものが挿入すべき遅延である．これを現環境の実行時間に加えたものが提案する推定実行時間である．推定実行時間が想定する外部メモリの遅延に比例する項を含むため,メモリの遅延を変化させたグラフを書けば直線になる．それで正しく予測できているかどうかは 4 章で検証する．

3.2 提案手法と既存手法の比較

以下の 7 つの観点で提案手法と既存手法の比較を行った．表 1 に後述する 7 つの観点からの提案手法と既存手法の比較の概要を示す．

表 1　提案手法と既存手法の比較の概要

比較項目	MARSSx86	Quartz	提案手法
評価スループット	1	1000	10000
適用範囲	×	△	◎
測定環境の柔軟性	×	△	◎
測定環境構築容易性	×	△	◎
調整の必要性	×	△	◎
予測精度	◎	× (→○?)	○
構成の柔軟性	◎	△	×

(1) 評価スループット

MARSSx86 はリアルタイムで動作する Quartz や提案法と比べて約 1 万倍程度の実行時間差がある．リアルタイムという点では Quartz も提案法も同じく分類できるが，評価スループットでは差がある．1 回の測定時間はその場で遅延挿入を行わない分提案法が速い．特に遅延の影響が大きい状況の場合，例えば減速率が 10 倍となるケースでは Quartz は 10 倍＋α(準備時間数秒)の測定時間がかかる．この準備時間は提案法より大きいので非常に短時間で終わる被測定アプリに対する測定時間の効率が悪くなる．さらに，測定する外部メモリ遅延条件数が複数ある場合には Quartz はその回数分だけ設定ファイルを書き換えて実行する必要があるのに対し，提案法は現環境のミス数だけ測定するので必要な測定回数は一桁程度差があり，この差は大きい．

(2) 適用範囲

MARSSx86 のようなサイクルアキュレートシミュレータは上記のように圧倒的に実行時間がかかるため，極めてサイクル数が少ない小さなフットプリントを持つアプリケーションしか評価できない．これは HPC アプリケーションの網羅的評価においては致命的である．Quartz は 2 ソケットの Intel Xeon サーバ上で 1 ソケット分が NVM 用に占有されるため，全コアを使った際の評価ができない．特に Affinity を制御しているアプリケーションとの相性が悪く，1 スレッド実行となったり，アプリが本来意図しない配置になって正しく評価できない．提案法はそのような制約や不安定性が無く，今のところ何でも測定できている．

(3) 測定環境の柔軟性

MARSSx86 は非常に実行時間がかかるため，CPU 性能が高い測定環境が求められる．Quartz は Intel Xeon の 3 バージョン(SandyBridge，IvyBridge，Haswell)しかサポートされておらず，それらの 2 ソケットサーバーが必要である．提案法は perf がインストール可能な Linux マシンがあれば測定と予測が可能である．

(4) 測定環境構築容易性

MARSSx86 の環境構築は実際にやってみるとあまり容易ではない．特に遠隔サーバ上で実行させるには困難を伴う上，アプリを仕込んだ仮想マシンのディスクイメージを作成する等の知識が必要になる．遅延の挿入の設定はリードとライトで遅延を分けない場合，設定ファイルに 1 行入れれば良いだけで簡単である．Quartz は MARSSx86 に比べると楽にインストールできるが，マニュアルを読んだり，多少手間がかかる．提案法は Intel Memory Latency Checker 等をダウンロードする必要があるが，上記は実行ファイル形式なのでインストールは不要である．perf もディストリビューションに入っているので非常に容易で，既にインストールされているケースも多いと考えられる．

(5) 調整の必要性

MARSSx86 は実機で逐次実行時間が数秒程度で終わりつつ，大きい問題と状況が変わらないように問題サイズなどをうまく調整しないと，1 点の測定に数時間と時間がかかりすぎて事実上使えない．Quartz はエポック期間に測定オーバーヘッドを償却できず誤差が広がることもあるので，その確認と，未償却が発生していた場合はエポック期間の調整が必要である．提案法は極めてシンプルなため，調整のような作業は発生しない．

(6) 予測精度

MARSSx86 は CPU 内部での動きを逐一模擬しているので最も予測精度が高い．使い方を間違えなければこれが正解を与えるとすることができる．Quartz と提案手法の予測精度は 4 章で検証する．結果的には実際に観測されている範囲では Quartz の精度が非常に悪い．原因が究明できていないので，何らかの不具合が直れば，原理的には提案法よりはストールの影響を織り込んでいるのでその点に限っては提案法より精度が高くなる潜在的要因がある．

(7) 構成の柔軟性

提案法や Quartz は予測対象の環境の仕様が測定環境の仕様を引きずる．測定環境で外部メモリの遅延だけ変化した環境に限定される．Quartz は CPU ソケット数が 1 つ減ることとアプリケーションに専用 API での記述をすることを引き換えに，遅延の大きなメモリと外部 DRAM が共存する構成の評価が可能である点で，提案法より柔軟性がある．MARSSx86 は存在しないマシン構成でも記述すれば評価可能であり，最も構成の柔軟性は高い．

4. 提案手法の検証

Quartz と提案手法の手元の評価環境において観測されている精度には大きな開きがある．例えば，Parsec ベンチマーク[22]の canneal に simsmall のデータを入力したシングルスレッド実行時間は約 2 秒である．MARRSx86 上で 1000ns の遅延を与えると 1.94 倍に実行時間が延びると予測された．これを正解と考えると約 4 秒弱になるはずである．ところが Quartz 上で 1000ns の遅延を与えた際に 4.6 秒の初期化時間の後，NVM accesses: 5790958 と injected delay in usec: 4806717 を表示し，エミュレーションされた実行時間は約 140 秒であった．MARRSx86 で確認した減速率の正解とは 35 倍もの差がある．これに対して提案手法は同じ条件で実機の主記憶遅延が 98ns とした場合 1.84 倍，115ns とした場合が 1.83 倍と減速率を予測する．MARRSx86 の実行時の条件設定と実機の間にはキャッシュサイズや周波数等に若干の差があることを踏まえても明らかに提案手法の方が圧倒的に高精度であることがわかる．

図 1 に Parsec ベンチマーク[22]の canneal に simsmall, simmedium,simlarge の 3 種類のサイズのデータと 1000ns までの外部メモリ遅延を 115ns から変化させた場合の減速率の変化を，MARRSx86 と提案手法の双方について同じグラフ上に表示したものを示す．サイズごとの対応する 2 本の線が重なっており提案手法が極めて良好な予測結果を出していることが明白である．もし，LLC ミス時に 2 回分の遅延をつけてしまうと直線の傾きが 2 倍急になってしまい MARRSx86 による正解の線から離れてしまうことから，3 章で述べたように 1 回分の遅延をつけることで良いことが実験からも確認できたと言える．

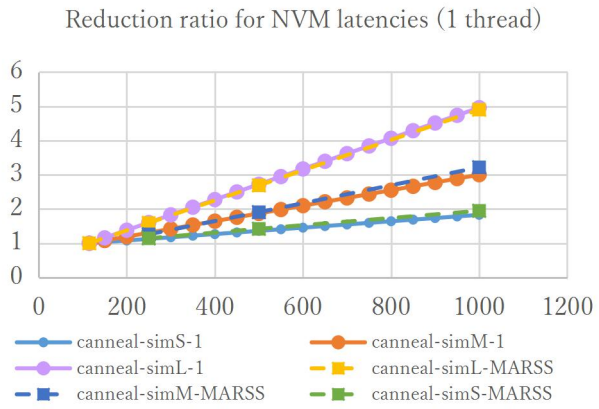


図 1 MARRSx86 と提案手法の結果の整合性

5. 疎行列系アプリへの影響評価への応用

提案手法の手法を応用した主記憶遅延増加時の疎行列系アプリケーション性能の網羅的評価を実施した．本章ではその実験内容と結果について述べる．

5.1 測定環境

表 2 に本実験で用いた測定環境の仕様を示す．DRAM 遅延は local,remote 分を Intel Memory Latency Checker，平均値は Quartz 付属の memlat による実測値を示している．キャッシュミス時に発生する 1Read+1Write のアクセス比率における DRAM 最大バンド幅は Intel Memory Latency Checker による実測値を示した．

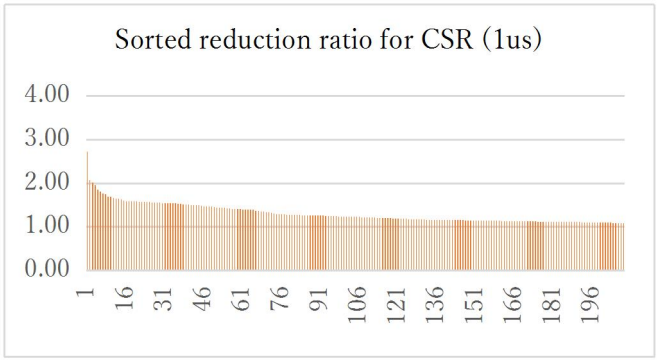
表 2 測定環境の仕様

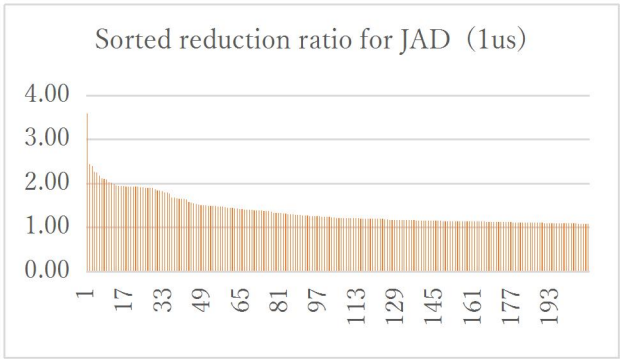
CPU name	Intel(R) Xeon(R) CPU E5-2697 v3
freq / RAM	2599.985 MHz / 125.65 GB
#CPU processors	28 (= 2 packages x 15 cores x 1 SMTs)
COMPILER	GCC (GNU C Compiler) version 4.8.5
Memory Latency	local: 98ns, remote: 139ns, average:
Memory Bandwidth	1Read+1Write (MAX) : 102.9GB/s

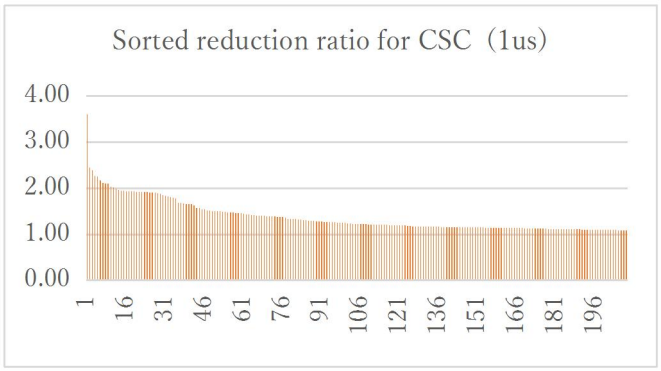
5.2 実験に用いたワークロード

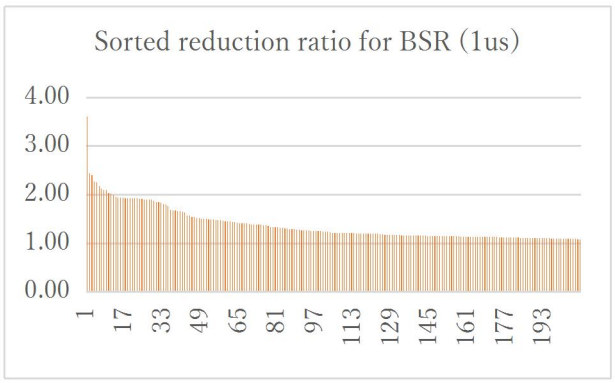
本実験で用いた疎行列系アプリケーションのワークロードの概要を示す．

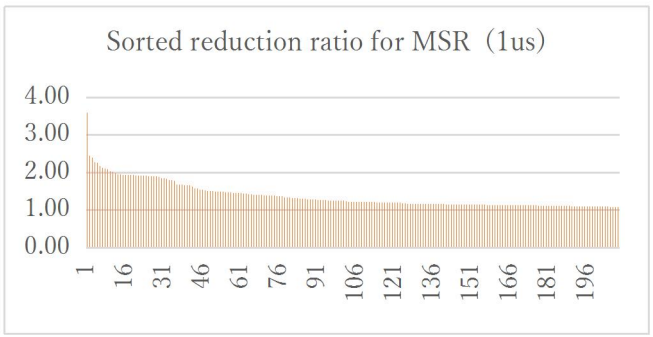
(1) 反復解法ライブラリ LIS-1.7.21[23]の spmvtest5 を用いた 10 種の格納方式における，フロリダ大学疎行列コレクション[24]から得た 208 種の行列に対する SpMV
(2) 反復解法ライブラリ LIS の hpcg_spmvtest を用いた 9 種の前処理法における 27 点ステンシル行列の SpMV
(3) 反復解法ライブラリ LIS の hpcg_kernel を用いた 2 種のソルバ(CG,BiCG)における，27 点ステンシル行列の連立一次方程式求解
(4) Graph500 ベンチマーク[25]の各種実装(参照実装の seq-csr，omp-csr，グラフ CREST による実装[26][27]の旧版 NETAL-BD13，新版 pBFS-v7.00)
(5) Parsec2.1 ベンチマーク[22]の LSI 配置配線問題 canneal のシミュレーション用データ 3 サイズ simsmall, simmedium, simlarge

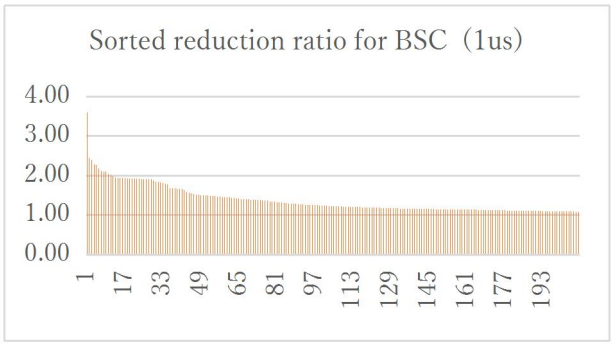


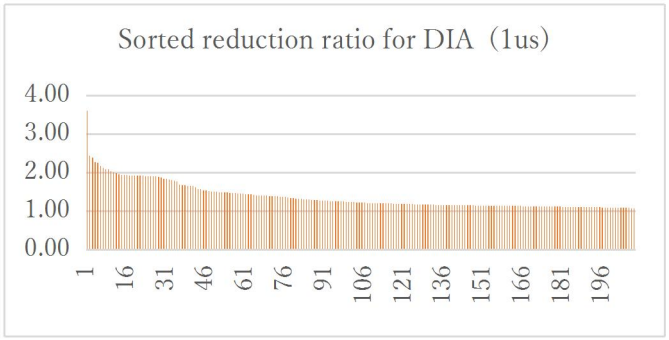


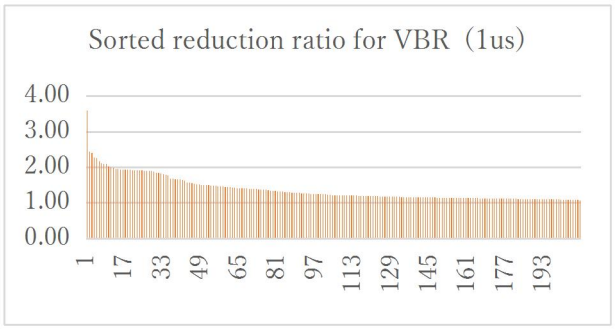


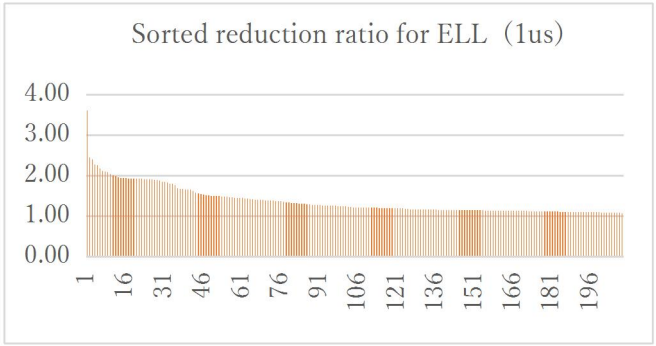












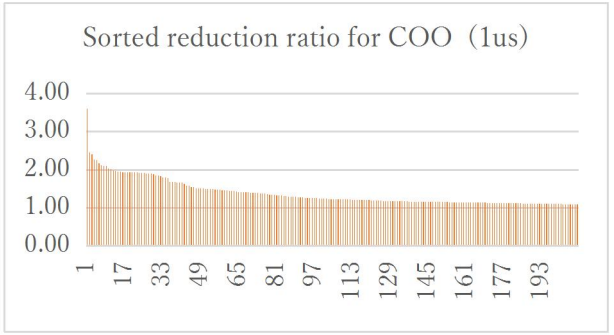


図 2　LIS の spmvtest5 を用いた 10 種の格納方式における 208 種の正方行列に対する SpMV の予測減速率(遅延 1 μ 秒)

5.3 LIS を用いた各種格納方式の SpMV

図 2 に LIS の spmvtest5 を用いた 10 種の格納方式における，フロリダ大学疎行列コレクションから抜粋した 208 種の正方行列に対する SpMV の提案手法による予測減速率をソートした結果を示す．メモリ遅延 1 マイクロ秒における値を表示した．横軸には順位,縦軸に予測減速率を取っている．分布の傾向はすべての格納方式に対して概ね同じであるが，上位の予測減速率が格納方法によって若干異なることが判った．全体的に格納方式 CSR がやや遅延に強い傾向を示している．

測定スループットの観点では，遅延方向に 10 程度の測定点に対応する実験を 1 つの測定で代表し，MARSSx86 や Quartz による測定点で 20800 点に相当する測定結果が，実行用および csv ファイル生成用にスクリプトを書いて 10 グループに分けて目視検査をしながらやって 2 日弱で得ることができた．MARSSx86 ならば図 2 のまばらな 10 点ほどの測定に約 1 週間かかっていたことと比較すると，数万倍の測定スループットを実感することができた．

5.4 LIS を用いた HPCG 用疎行列の SpMV

図 3 に LIS の hpcg_spmvtest を用いた 27 点ステンシル行列の 10 種の格納方式における SpMV の予測減速率を示す．この結果で注目すべきはワークロード(1)の 208 種の疎行列には見られなかったような高い減速率がこの行列には表れている点である．HPCG ベンチマークは Top500 の後継とされる HPC 業界における重要なベンチマークであるため，格納方式を誤ると非常に高い遅延感度が発生してしまうという今回得られた知見は重要と思われる．

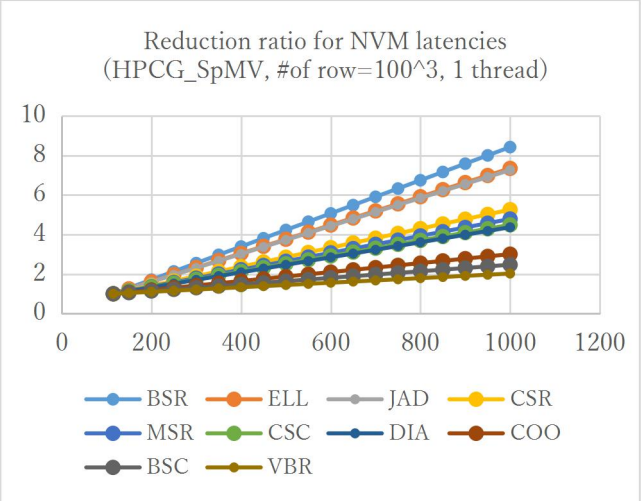


図 3　LIS の hpcg_spmvtest を用いた 10 種の格納方式における SpMV の予測減速率

図 4 に LIS の hpcg_spmvtest を用いた 10 種の格納方式における SpMV の予測実行時間と減速率の関係を示す．実行時間が短いものほど遅延による減速率も小さいので，比較的単純に格納形式を選択できるが，実行時間的に最適なものと遅延による減速率の面で最適なものは異なることが判った．つまり，概ね減速率が低いものを選んでいれば良いが実行時間で 2 倍位差が出ることがあるので他の候補が最適でないか注意した方が良いと言える．

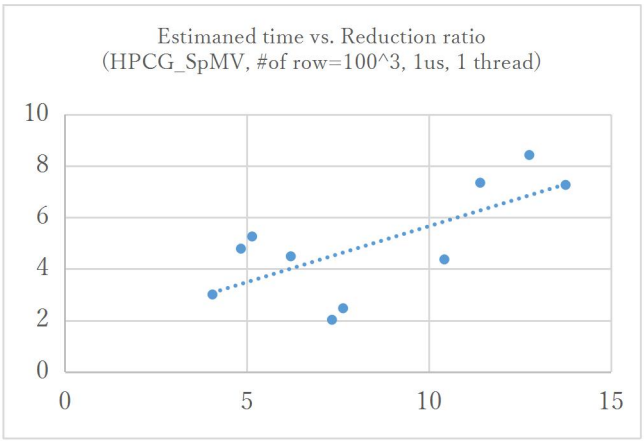


図 4　LIS の hpcg_spmvtest を用いた 10 種の格納方式における SpMV の予測実行時間と減速率の関係

5.5 LIS を用いた HPCG 用疎行列の連立方程式求解

図 5 に LIS を用いた HPCG 用疎行列の連立方程式求解の予測減速率を示す．前処理法によっては遅延に対して非常に敏感であることが判った．ソルバを CG から BiCG に変えた場合も同様に測定したが同じような傾向を示した。前処理 SAINV が非常に長い実行時間がかかったが，収束しなかったわけではなく，反復回数が少ない代わりに前処理時間が長くかかるアルゴリズムのようである．

HPCG ベンチマークにおいて前処理アルゴリズムは Gauss-Seidel smoother と指定されているので選択の余地は無いが，同様の構造の疎行列を係数とする連立一次方程式求解が必要な場合は前処理の選択を遅延の観点からも慎重に行うべきであるという知見が得られた．

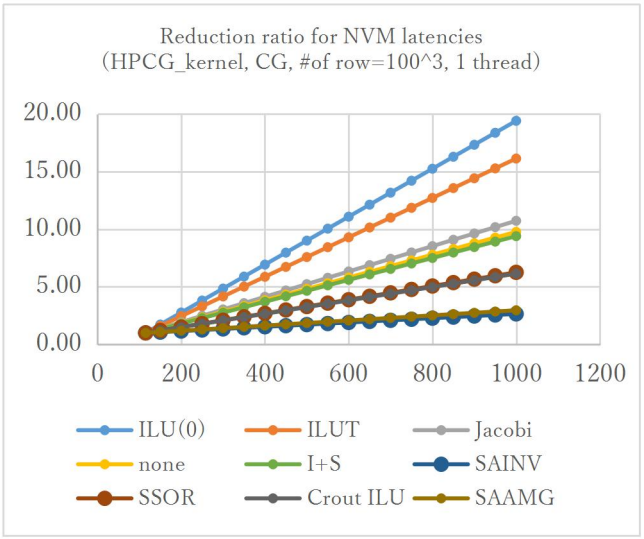


図 5　LIS を用いた HPCG 用疎行列の連立方程式求解の 9 種の前処理方式における減速率

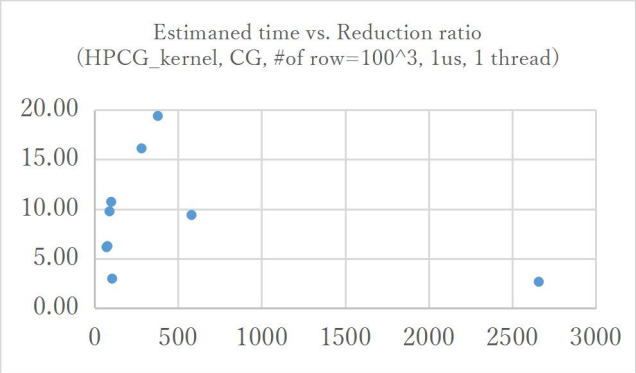


図 6　LIS の hpcg_kernel を用いた 9 種の前処理方式における連立一次方程式求解の予測実行時間と減速率の関係

図 6 は実行時間を横軸に取ったものである．減速率のみで最小の SAINV を選択すると一番時間がかかる選択になっているので注意が必要であることが判る．

5.6 各種 graph500 実装

図 7 に Graph500 の各種実装における予測減速率を示す．グラフ CREST 実装の新版 pBFS-v7.00，旧版 NETAL-BD13 の比較では減速率の観点からは旧版の方が遅延への耐性がやや強いことが判った．ワークロード(1)の SpMV より遅延への感度がかなり高いことがわかる．

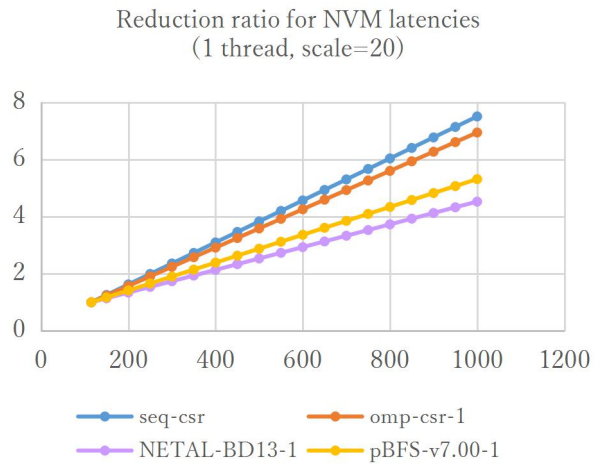


図 7　Graph500 の各種実装における予測減速率

5.7 Parsec2.1 ベンチマーク canneal

Parsec2.1 ベンチマーク canneal については既に精度検証のために図 2 に示したような予測減速率が得られた．

6. 得られた知見と考察

本章では前章で得られた結果を踏まえ，いくつかの角度からの考察を行う．

6.1 疎行列の特徴と減速率の関係

図 8，図 9，図 10 に示すように疎行列の表面的な特徴量(行数，非零要素数，行当たり非零要素数)と減速率の間には有意な相関関係が見出せなかった．図には格納形式 CSR しか表示していないが他の格納形式でも同様である．

しかし，格納方式 ELL を除く他の格納方式では，減速率が多く出る行列名は比較的限られていることが判った．表 3 に格納方式毎の減速率上位 10 個の疎行列名，表 4 に ELL 以外の格納方式で上位に頻出する疎行列名とその特徴量を示す．これらの行列の非零要素配置には格納方式にあまり依存しない遅延の影響を強く受ける性質が内在していると考えられる．特に gsm_106857 については殆どの格納法で遅延時間への影響が 1 位であり，この疎行列を深く調べることが遅延時間への影響が大きくなる条件の解明に有望と考えられる．本報告ではそのような行列の特定までを行ない，なぜこれらの行列では遅延時間の影響が大きくなるかの原因の究明は今後の課題とする．

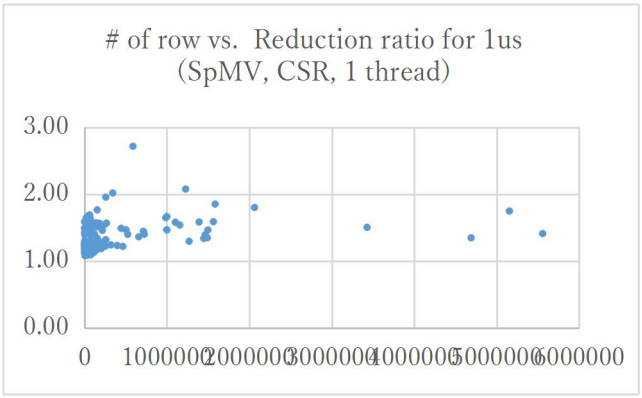


図 8　SpMV における行数と予測減速率の関係

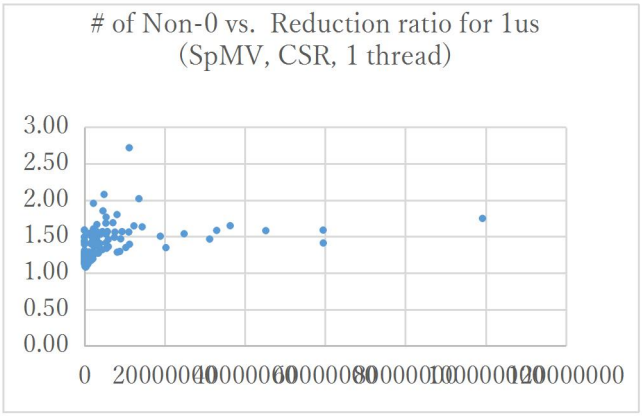


図 9　SpMV における非零要素数と予測減速率の関係

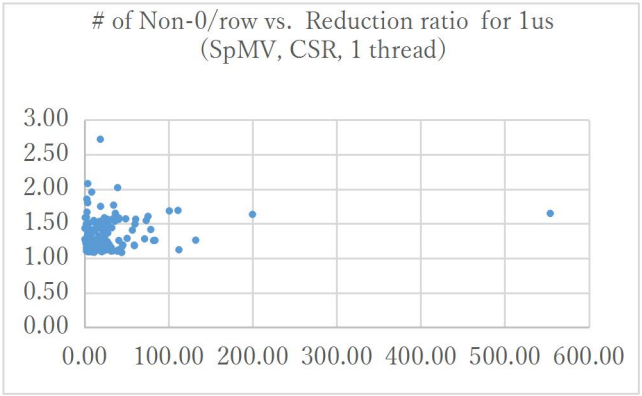


図 10　SpMV における非零要素数/行と予測減速率の関係

表 3　ELL を除く格納方式毎の減速率上位 10 個の疎行列名

1:CSR	2:CSC	3:MSR	4:DIA	6:JAD	7:BSR	8:BSC	9:VBR	10:COO
gsm_106857	gsm_106857	gsm_106857	gsm_106857	Serena	gsm_106857	gsm_106857	gsm_106857	gsm_106857
thermal2	F1	thermal2	thermal2	kkt_power	offshore	poisson3Db	thermal2	thermal2
F1	thermal2	F1	F1	F1	poisson3Db	thermal2	offshore	cop20k_A
offshore	offshore	offshore	offshore	nd24k	thermal2	F1	kkt_power	offshore
G3_circuit	kkt_power	G3_circuit	G3_circuit	Hook_1498	kkt_power	offshore	F1	kkt_power
kkt_power	cage15	kkt_power	kkt_power	thermal2	sme3Db	kkt_power	G3_circuit	poisson3Db
para-7	crankseg_2	para-7	para-7	offshore	G3_circuit	cage15	webbase-1M	G3_circuit
cage15	crankseg_1	cage15	cage15	dielFilterV2real	F1	cop20k_A	poisson3Db	F1
crankseg_2	para-7	crankseg_2	crankseg_2	dielFilterV3real	para-7	para-7	para-7	para-7
crankseg_1	nd24k	crankseg_1	crankseg_1	gsm_106857	human_gene1	crankseg_2	cop20k_A	cage15

表 4　ELL 以外の格納方式で減速率上位に頻出する疎行列名とその特徴量

	# of row	#of Non-0	NNZ/row
gsm_106857	589446	11174185	18.96
thermal2	1228045	4904179	3.99
F1	343791	13590452	39.53
offshore	259789	2251231	8.67
G3_circuit	1585478	4623152	2.92
kkt_power	2063494	8130343	3.94
para-7	155924	5416358	34.74
cage15	5154859	99199551	19.24
crankseg_2	63838	7106348	111.32
poisson3Db	85623	2374949	27.74

表 5　ELL の格納方式でのみ上位になる疎行列名とその特徴量

	# of row	#of Non-0	NNZ/row
bloweya	30004	80005	2.67
a0nsdsil	80016	200021	2.50
mult_dcop_02	25187	193276	7.67
c-72	84064	395811	4.71
shermanACb	18510	145149	7.84
matrix-new_3	125329	2678750	21.37
circuit_2	4510	21199	4.70
c-61	43618	176817	4.05
c-68	64810	315408	4.87
ckt11752_dc_1	49702	333029	6.70

6.2 LLC ミス率と減速率の関係

図 11 に示すように LLC ミス率と減速率の間には格納方式 ELL を除く他の格納方式では, LLC ミス率の 2 乗で近似される減速率付近に予測された減速率が分布することが判った．一方，図 12 に示すように ELL に他方式と異なる異様な現象が見つかった．具体的には，ミス率 5%(低め)付近に減速率 5〜7 倍を示すグラフ上に赤丸で囲った垂直に立つグループが明らかに存在する．表 5 に ELL の格納方式でのみ上位になる疎行列名とその特徴量を示す．これらの行列の非零要素配置には ELL の格納方式との組み合わせでのみ遅延の影響を強く受ける性質が内在していると考えられる．本報告ではそのような行列と格納法の組み合わせの特定までを行ない，なぜこれらの行列と ELL の組み合わせでは遅延時間の影響が大きくなるかの原因の究明は今後の課題とする．

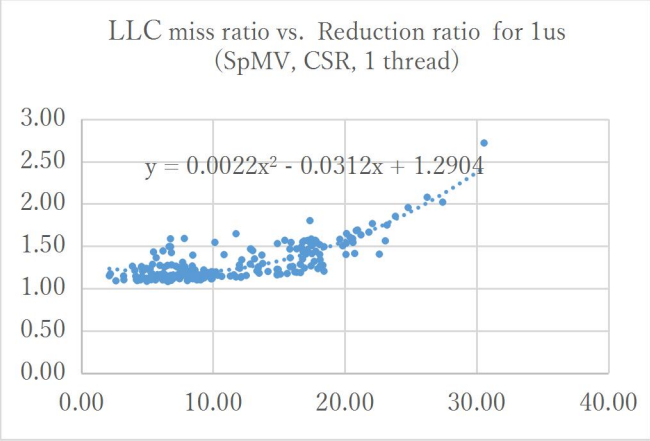


図 11　格納方式 CSR での SpMV における LLC ミス率と予測減速率の関係

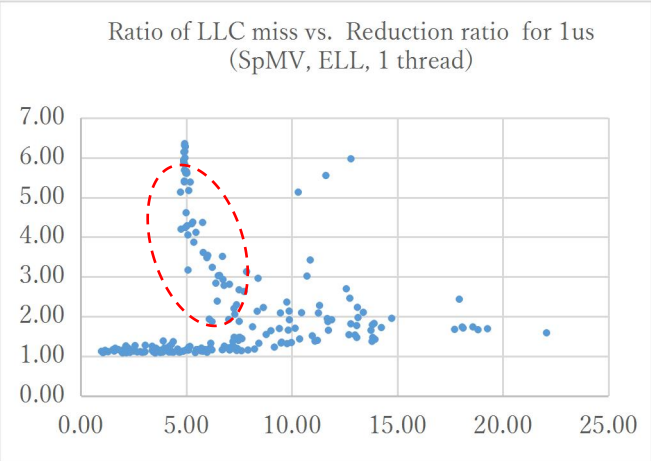


図 12　格納方式 ELL での SpMV における LLC ミス率と予測減速率の関係

6.3 秒あたりの LLC ミス数とバンド幅と減速率の関係

図 13 に各種アプリケーションのグループにおける秒あたりの LLC ミス数から逆算される主記憶が DRAM の場合の要求バンド幅を示す．要求バンド幅の逆算法は簡単であり，ミス回数にリード 64 バイト，ライト 64 バイトの合計 128 バイトを乗ずることで得られる．

今回の探索範囲においてはシングルスレッドでの測定であることもあり，Intel Memory Latency Checker で測定されたリード対ライトの比が 1 の時の測定環境の最大バンド幅である 102.9GB/s を遥かに下回るリクエストしかキャッシュラインリプレースの際には発生しない．

実際にはプリフェッチャが発生するメモリアクセス要求も存在し，バンド幅を消費すると考えられるが，少なくとも今回の探索範囲ではメモリバンド幅が足りない状況にはなっていないと考えられる．つまり，処理性能は純粋に CPU の処理速度(ロード・ストア命令実行に伴う遅延も含む)が律速する状況での測定が行われていると考えられる．

一方，大きなスレッド数でマルチスレッド動作をさせた場合や，GPU や Xeon Phi のように演算器が大量にある環境では測定環境の最大バンド幅以上のメモリアクセス要求が発行され，メモリバンド幅律速のモードで性能が決まる状況になりうる．ただし，本実験環境において最大スレッド数である 28 スレッド動作でさせた場合，並列化効率が 100%を仮定しても秒あたりの LLC ミス数が 28 倍に増加するにとどまる．その場合でも表　図 13 に示されるように上記の 102.9GB/s を上回ることはないことがわかる．

3D Xpoint の実効バンド幅が DRAM の 6 割に落ちるのであれば，HPCG_kernel を ILU(0)の前処理の条件において 28 スレッドで遅延が DRAM と同等な時に予測される要求 72.9GB/s＞61.7GB/s＝102.9GB/s×0.6 となり，ようやくバンド幅律速のモードに入る可能性があると考えられる．ところが，実際には 3D Xpoint はメモリアクセス遅延が 250ns に延びると予想され，それに伴ってスレッド当たりの処理時間がシングルスレッド時同様に 3.74 倍に延びると考えられるので，アクセス要求頻度はその逆数倍に減少するため，バンド幅律速にはならないと予測される．

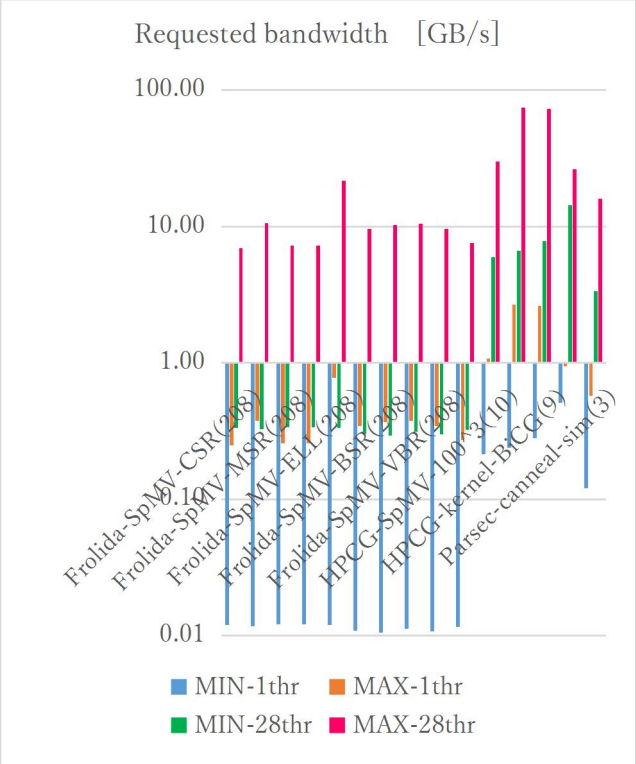


図 13　アプリケーションのグループにおける主記憶が DRAM の場合の要求バンド幅の最大値と最小値

6.4 アプリケーション横断での減速率の比較

遅延の影響の受けやすさは遅延 vs.減速率グラフの傾きである「秒あたりの LLC ミス数」で表現することができる．このような単純な指標はアプリケーションの種類が全く異なっていても平等に取得・比較ができる．図 14 に本実験で条件の異なるアプリケーションのグループにおける単位時間あたりの LLC ミス数の最大値(オレンジ)と最小値(青)を示す．グループ名の最後の括弧の中の数字はそのグループに属するアプリケーションの個数を意味する．例えば Frolida-SpMV-CSR は格納形式が CSR の場合のフロリダ疎行列コレクション 208 種類のグループを意味する．通常，同種のアプリケーションの中で格納形式のみ違うもの等の比較は実行時間で単純に比較しやすいが，異種のアプリケーション(例えばフロリダ疎行列コレクションの SpMV と graph500)の間の比較は難しい．ところが，単位時間あたりの LLC ミス数の分布範囲で比較すると，graph500 の方が大きい(遅延の影響を受けやすい)ところに分布することがわかる．

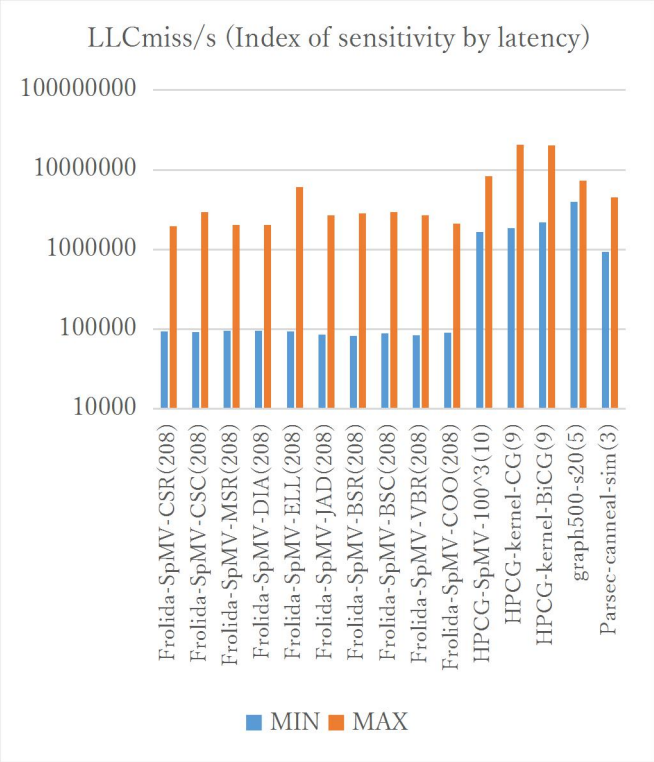


図 14　アプリケーションのグループにおける単位時間あたりの LLC ミス数の最大値と最小値

7. まとめ

本報告では Quartz より優れたスループット，予測精度，適用範囲，測定環境の柔軟性，環境構築容易性を有する，主記憶遅延増加時のアプリケーション性能予測方法の提案と，その精度評価を行なった．perf コマンドを用いた非常に簡潔な測定により，精度良くメモリ遅延時間のアプリケーション実行時間への影響を予測できる．

さらに，上記手法を応用した主記憶遅延増加時の疎行列系アプリケーション性能の網羅的評価を実施し，それらから得られた様々な知見について述べた．

本研究において抽出された遅延時間の影響が大きくなる格納法と行列の組合せに対して，いかなるメカニズムでそのような現象が起きるのかについて解明することが，今後の課題である．本報告では疎行列系の HPC アプリケーションを中心にメモリ遅延の影響の評価を行なったが，密行列系および粒子系の HPC アプリケーションや AI ・ビッグデータ解析系アプリケーションに関する評価は今後の課題である．

また，本報告では汎用 CPU 上での評価しか対応できていないが，GPU 環境においてもキャッシュのミス数をカウントする方法は存在するため，GPU のデバイスメモリに 3D Xpoint を用いた場合の評価も原理的には可能と考えられる．その実施は今後の課題である．

謝辞　本研究は，科学技術振興機構戦略的創造研究推進事業(JST CREST)の研究課題「ポストペタスケール時代のメモリ階層の深化に対応するソフトウェア技術」の支援を受けている．Graph500 の先端的実装を実験用にご提供いただいた九州大学藤澤克樹教授に感謝いたします．

参考文献

[1] Intel. 3D Xpoint Technology. http://www.intelsalestraining.com/infographics/memory/3DXPoint c.pdf, (参照 2016-11-20).
[2] Intel. IDF15 におけるスライドの抜粋. http://www.3dnews.ru/919364, (参照 2016-11-20).
[3] 田邊昇，遠藤敏夫．中遅延大容量メモリ階層出現のインパクトと新たな対応に関する初期検討．第 157 回ハイパフォーマンスコンピューティング研究会，2016, Vol.2016-HPC-157,No.11.
[4] Avadh Patel, Furat Afram, Shunfei Chen, and Kanad Ghose. MARSS: a full system simulator for multicore x86 CPUs. In Proceedings of the 48th Design Automation Conference (DAC '11). ACM Press, 2011, p.1050-1055.
[5] MARSSx86 - Micro-ARchitectural and System Simulator for x86-based Systems. http://marss86.org/~marss86/index.php/Home, (参照 2016-02-20).
[6] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The gem5 simulator. SIGARCH Computer Architecture News Vol.39, No.2 (August 2011), p.1-7.
[7] The gem5 Simulator: A modular platform for computer-system architecture research. http://gem5.org/Main_Page, (参照 2016-02-20).
[8] Haris Volos, Guilherme Magalhaes, Cherkasova, and Jun Li. Quartz: A Lightweight Performance Emulator for Persistent Memory Software. In Proceedings of the 16th Annual Middleware Conference (Middleware '15), ACM Press, 2015, p.37-49.
[9] Jasmina Malicevic, Subramanya Dulloor, Narayanan Sundaram, Nadathur Satish, Jeff Jackson, and Willy Zwaenepoel. Exploiting NVM in large-scale graph analytics. In Proceedings of the 3rd Workshop on Interactions of NVM/FLASH with Operating Systems and Workloads (INFLOW '15). ACM Press, 2015, Article 2.
[10] Joy Arulraj, Andrew Pavlo, and Subramanya R. Dulloor. Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. In Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). ACM Press, 2015, p.707-722.
[11] Subramanya R. Dulloor, Sanjay Kumar, Anil Keshavamurthy, Philip Lantz, Dheeraj Reddy, Rajesh Sankaran, and Jeff Jackson. 2014. System software for persistent memory. In Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14). ACM Press, 2015, Article 15.
[12] Ismail Oukid, Daniel Booss, Wolfgang Lehner, Peter Bumbulis, and Thomas Willhalm. SOFORT: a hybrid SCM-DRAM storage engine for fast data recovery. In Proceedings of the Tenth International Workshop on Data Management on New Hardware (DaMoN '14), ACM Press, 2015, Article 8.
[13] Yiying Zhang, Jian Yang, Amirsaman Memaripour, and Steven Swanson. Mojim: A Reliable and Highly-Available Non-Volatile Memory System. SIGARCH Comput. Archit. News 43, 1 (March 2015), ACM Press, p.3-18.
[14] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory

Technology. Proc. 36th Int'l Symp. Computer Architecture (ISCA 09), ACM Press, 2009, pp. 14-23.
[15] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. Proc. 36th Int'l Symp. Computer Architecture (ISCA 09), ACM Press, p. 2-13.
[16] Moinuddin K. Qureshi, John Karidis, Michele Franceschini, Vijayalakshmi Srinivasan, Luis Lastras, and Bulent Abali. Enhancing Lifetime and Security of Phase Change Memories via Start-Gap Wear Leveling. Proc. 42nd Ann. Int'l Symp. Microarchitecture (MICRO-42), ACM Press, 2009, pp. 14-23.
[17] M.K. Qureshi, V. Srinivasan, and J.A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. Proc. 36th Int'l Symp. Computer Architecture (ISCA 09), ACM Press, 2009, p. 24-33.
[18] Amoghavarsha Suresh, Pietro Cicotti, and Laura Carrington. Evaluation of emerging memory technologies for HPC, data intensive applications. IEEE International Conference on Cluster Computing (CLUSTER 2014), 2014, p.239-247.
[19] Yusuke Shirota, Shiyo Yoshimura, Satoshi Shirai, and Tatsunori Kanai. Powering-off DRAM with Aggressive Page-out to Storage-class Memory in Low Power Virtual Memory System. Proceedings of COOL Chips XIX, 2016.
[20] Dipanjan Sengupta, Qi Wang, Haris Volos, Ludmila Cherkasova, Jun Li, Guilherme Magalhaes, and Karsten Schwan. A Framework for Emulating Non-Volatile Memory Systemswith Different Performance Characteristics. In Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering (ICPE '15). ACM Press, 2015, p.317-320.
[21] 小柴篤史, 広渕崇宏, 高野了成, 並木美太郎. Read/Write レイテンシの違いを考慮した不揮発性メモリのソフトウェアエミュレータ. 第 28 回コンピュータシステム・シンポジウム (ComSys2016), 2016, p.28-34.
[22] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: characterization and architectural implications. In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08). ACM Press, 2008, p.72-81.
[23] 反復解法ライブラリ LIS. http://www.ssisc.org/lis/
[24] Tim Davis. The SuiteSparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/
[25] Graph500. http://www.graph500.org/
[26] Yuichiro Yasui, Katsuki Fujisawa and Kazushige Goto. NUMA-optimized parallel breadth-first search on multicore single-node system, *2013 IEEE International Conference on Big Data*, 2013, pp. 394-402.
[27] Yuichiro Yasui and Katsuki Fujisawa. Fast and scalable NUMA-based thread parallel breadth-first search. 2015 International Conference on High Performance Computing & Simulation (HPCS), 2015, pp. 377-385.