Vol.2017-HPC-160 No.12
2017/7/26

Intel Xeon Phi における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：米国の次世代スーパーコンピュータ Aurora には DIMM 型の 3D Xpoint を主記憶にした次世代 Intel Xeon Phi が利用されることがアナウンスされている。その類似環境は次世代 HPC インフラとして有望な候補となる。3D Xpoint は DRAM より大容量である一方で遅延が大きく、耐久性の制約から DRAM キャッシュとの併用が必須である。現世代の Xeon Phi はオンパッケージ DRAM をキャッシュに設定可能だが、対応する stall 関連の性能カウンタが無いため既存の遅延変動ツールを移植することが困難である。本研究では、そのような制約を有する Intel Xeon Phi の主記憶の遅延増加に伴う処理性能への影響の予測手法を提案し、多様なアプリケーションによって評価を行なう。

1. はじめに

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．2018 年完成予定の米国の次世代スーパーコンピュータ Aurora には DIMM 型の 3D Xpoint を主記憶にした次世代 Intel Xeon Phi が利用されることがアナウンスされている。その類似環境は次世代 HPC インフラとして有望な候補となる。

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．

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，多様なアプリケーション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]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，高遅延主記憶のソフトウェアエミュレータ Quartz[6]が公開されている．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[3]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ことを報告[7]した．

ただし，上記の報告で提案した測定法においては，Out-of-Order 機構を有する CPU 上での Outstanding なメモリアクセス多重度を無視する近似が導入されており，多数スレッドでの動作時に誤差が拡大することが予想される．このため多数スレッドで動作させることがほぼ必須な Xeon Phi のようなメニーコア型 CPU を用いる環境においては上記誤差の解消方法が望まれる．

さらに，上記の報告で提案した測定法においては，Xeon Phi KNL の DRAM キャッシュを用いないモードはそのまま適用可能だが，3D Xpoint の耐久性不足を補うために必要な DRAM キャッシュを有する構成に対応できていなった．

さらに，上記の報告で示した疎行列系アプリケーションの遅延感度は大容量な DRAM キャッシュが無い場合の評価結果になっているため，DRAM キャッシュを用いることがほぼ必須な 3D Xpoint の評価としては厳しい条件での評価となっていた．

本研究は上記のような課題を解決し，DRAM キャッシュを用いる設定がなされた Xeon Phi KNL 上でのアプリケーションの遅延感度測定法を開発する．バンド幅制約の効果が混在しがちな stream を多数スレッドで動作させるようなアプリケーションにも対応可能な遅延感度測定の実現を目指す．

本研究の貢献は以下のとおりである．

(1) Xeon Phi KNL の DRAM キャッシュを用いるモード上で動作する二種類(CAS カウント方式および Outstanding 補正方式)の主記憶遅延感度評価手法の提案．
(2) Out-of-Order 機構を有する CPU(Skylake および Xeon Phi KNL)上での DRAM への Outstanding なメモリアクセス多重度の測定法とその Stream における値の実測，それらを用いた主記憶遅延感度評価の精度改善方法の提案．
(3) DRAM キャッシュを設定した Xeon Phi KNL 上の疎行列ベクトル積の遅延感度評価と，それらから得られた 3D Xpoint を主記憶とする際のより精度の高い知見．

本報告の構成は以下のとおりである．まず，2 章では本報告が扱う研究対象について述べる．3 章では，既存の遅延感度測定法の代表例を概観する．4 章では，除去を検討

する誤差要因について述べる．5 章では，提案する主記憶遅延影響予測手法について述べる．6 章では，性能評価を示す．7 章でまとめる．

2. Xeon Phi と遅延感度

本章では本報告が扱う研究対象について述べる．本報告の動機を形成している一因として，米国 ANL に 2018 年に納入が予定されている米国の次世代スーパーコンピュータ Aurora が存在する．Aurora は Intel Xeon Phi KNL(Knights Landing)の後継となるメニーコア型 CPU で構成されるとともに，主記憶に Intel/Micron が開発した 3D Xpoint を搭載した DIMM を主記憶に持つことがアナウンスされている．

3D Xpoint は DRAM の約 10 倍の集積度を有するため，安価に大容量主記憶を構築できる．一方，遅延時間と耐久性は DRAM より大幅に劣る．

このため，アプリケーションの遅延感度を測定する意義がある．耐久性が DRAM のように無限とみなせないため，実機での使用においては DRAM 階層によって 3D Xpoint へのアクセス頻度を大幅に抑制する必要がある．つまり，DRAM キャッシュの併用がほぼ必須であると考えられる．

現在入手可能で最新の Intel Xeon Phi としては Intel Xeon Phi KNL があり，評価環境として筆者らはこれを用いることを考える．Xeon Phi KNL は 64～72 個の SIMD 拡張された Atom ベースのコアを用いる．それらのコアは Out-of-Order 型であり，バイナリレベルで Xeon と互換性があり，Linux 等の OS がブート可能である．それらに分散配置された共有 L2 キャッシュとオンパッケージに 8 チャネルで合計 16MB の高バンド幅な MCDRAM をメモリシステムとして内蔵する．この MCDRAM は 1/4，1/2 または全体の容量を Embedded DRAM Cache (EDC)に設定して L3 キャッシュとしても利用可能である．よって，3D Xpoint へのアクセス頻度を大幅に抑制する目的で必須となる DRAM キャッシュをハードウェアとして設定可能である点で注目し，本研究では評価環境としての利用を探求する．

マルチコア型 CPU の Xeon 系列と比較して制約的な特徴としては，Out-of-Order 等のハードウェア資源が少なく，コア単体の性能的には大幅に劣る．このため，KNL の利用においては Xeon 系より多くのスレッドを走らせないで利用すると性能面で負けてしまうことが多い．また，KNL の性能カウンタは Xeon 系とは互換性に乏しく，KNL にはあって Xeon 系には無い物や，その逆のケースもある．

3. 既存の遅延感度測定法

本章では既存の代表的な主記憶遅延影響予測手法について述べる．

3.1 ソフトウェアシミュレータ TaskSim

BSC および Samsung のチームは STT-MRAM ベースの主記憶遅延影響の評価[8]を 2016 年に発表した．その際には厳選した HPC アプリケーションのメインループについて，インストゥルメントツール Valgrind を用いてある程度フィルタリングされたメモリアクセストレースを抽出し，ソフトウェアシミュレータ TaskSim[9]の速度優先モードである mem モードに投入している．この mem モードはキャッシュのシミュレートはしているが，Out-of-Order(OoO)パイプラインのシミュレートしておらず正確性に欠ける．その実行時間の大きな割合を占めると思われる TaskSim の mem モードは実機の 1561 倍の時間がかかることが報告[9]されている．疎行列処理のような入力データ依存のカーネルをこの手法で評価しようとすると，例えば実機で 2 日間かかるようなアプリケーション群の評価は TaskSim の実行時間だけで約十年間かかってしまう．実際にはメモリアクセストレース抽出は TaskSim の mem モード実行以上に時間がかかる可能性が高く，この手法を多様かつ大きな疎行列処理等の評価に用いるということは現実的ではない．

3.2 ハードウェアエミュレータ HMEP

上記のような実行時間の問題が無いのがハードウェアエミュレータである．リアルタイムで動作する Intel のハードウェアエミュレータ HMEP を用いた先行研究[10]-[14]がある．ただし HMEP 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．設定した遅延により実際に性能が低下してしまうので評価の能率は必ずしも最高ではない．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．このアプローチは CPU ごとにハードウェアを開発する必要があるが，本研究が対象とする Xeon Phi の後継機に対応するハードウェア式エミュレータが開発されたという情報は現在のところ見当たらない．

3.3 ソフトウェアエミュレータ Quartz

リアルタイムで動作する評価環境としてはソフトウェアエミュレータ Quartz があり git にて公開されているため関係者以外でも試すことができる．ただし，特定型番の Xeon(SandyBridge，IbyBridge，Haswell)の 2 ソケットサーバのみサポートされている．筆者らの Haswell ベースの測定環境では残念ながら精度も低く，動作が不安定であった．実験ごとに初期時間が 5 秒ほどかかり，短時間のアプリの評価を大量にこなすには能率が悪い．

Quartz は原理的には L2 ミスに伴う CPU ストールの性能カウンタからメモリアクセス分を予測する性能モデルを用いて，メモリ遅延増加に伴うストールを予測し，人工的に遅延を挿入する．ところが，本研究が対象とする Xeon Phi には対応するストール関係の性能カウンタが存在しないため，その戦略での Quartz 移植はできない．

3.4 性能モデルと perf を組合せて用いる手法

筆者らは本研究に先立ち，将来にわたって広範なプラットフォーム上で，安定的に動作しつつ，網羅的な評価も可

能な程度に非常に高いスループットで，ユーザが手元のアプリケーションの遅延感度を測定可能な，簡易的な遅延感度測定法を 2017 年 3 月に提案[7]し，疎行列系カーネルの網羅的性能評価を行った．動作原理は Quartz とは異なり，Last Level Cache(LLC)ミス数の性能カウンタ値をメモリアクセス数に近似し，LLC ミス数にストール増加が直線的に増加する単純な線形性能モデルを用いて，上記の優れた特性を実現した．

その報告で探求する課題はアプリケーション毎の純粋な遅延感度であるため，バンド幅律速やその効果が混ざった状況を極力排除した状況 1 スレッド)での測定を行った．これによりアプリ間の遅延感度の大小を比較することが可能になった．コア数が少ない CPU 環境では，そのような状況での使い方をするケースも少なくないと考えられる．

しかしながら，本研究が対象とする Xeon Phi のようなメニーコア型 CPU においては，多数スレッドで走らせないとマルチコア型 CPU の性能に負けてしまうことが少なくないため，ハンド幅の影響が混ざっている状態での利用における遅延感度を知るニーズが存在する．そのニーズに応えるためには上記提案の優れた特性の一部を削ってでも，Xeon Phi などのメニーコア型 CPU に対象を絞った補正手法の開発が望まれる．

4. メニーコア向けに除去を検討する誤差要因

論文[7]の提案手法は OoO 機構の一部の効果(投機的なメモリアクセスによる並列メモリアクセスの効果)を省略する近似を用いている．メニーコアでありがちな利用状況(多数のスレッドで実行してバンド幅の影響も大きい状況)下においてはこの誤差が増大するおそれがあるため，検討が必要である．

論文[7]の評価環境の CPU(Haswell)には存在しないが，それより新しい世代の Xeon である Skylake においては L2 ミスおよび L3 ミスに伴う Demand リードの投機的実行数を累積カウントする性能カウンタが存在する．そのカウント値を実行サイクル数で除算したものがその期間の平均 Outstanding リード数となる．予備評価としてサーバ向けではない 4 コア(Hyper Thread 数は各コア 2)の Skylake(Core i7-6700K,4.00GHz)を用いた環境で配列サイズを L3 キャッシュより十分に大きい 80M に設定した stream ベンチマークを 1 スレッドと 4 スレッドで実行した際の上記カウンタから算出した 100ms 周期の平均 Outstanding リード数の時系列変化をそれぞれ図 1 および図 2 に示す．

図 1 および図 2 より，1 スレッド実行の場合は DRAM への Outstanding リード数が非常に少なく論文[7]の提案手法の誤差が小さいことが判ったが，4 スレッド実行時には誤差が大幅に増えることがわかる．本評価環境はメモリチャネル数が多いサーバではないためメモリバンド幅が小さいため，少ないスレッド数で Outstanding リード数が増加しやすい分，現象が明瞭に観測できる．

論文[7]の提案手法は stream のように主にバンド幅律速で性能が決まるアプリケーションでマルチスレッド実行した際には，上記の Outstanding リード数を 0 と近似したことによるメモリアクセス数や遅延感度を過剰にカウントしてしまう誤差が生じる．4 スレッドの stream の平均 Outstanding リード数は 1.15 であり，これの他に 1 アクセスが先行して実行されていると考えると，2.15 倍に遅延感度を過剰にカウントすることになる．メニーコア CPU ではこの状況が加速されると考えられるので，この誤差を除去する評価手法が必要である．

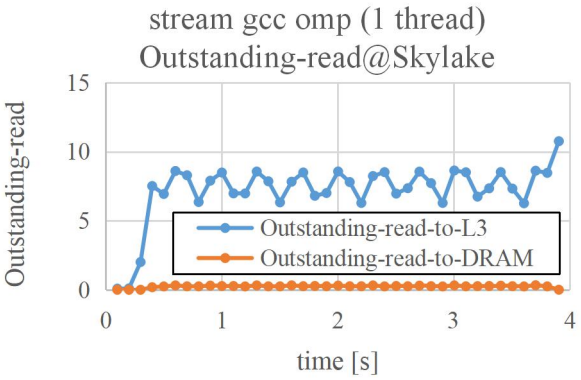

図 1　Skylake 上の stream 実行時の Outstanding リード数の時系列変化(1 スレッド時)

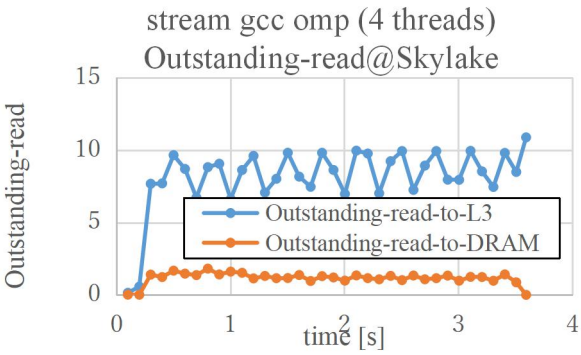

図 2　Skylake 上の stream 実行時の Outstanding リード数の時系列変化(4 スレッド時)

5. 提案手法

本章では提案する Xeon Phi 上の主記憶遅延影響予測手法について述べる．

5.1 提案手法(1) DIMM アクセス数測定法

Xeon Phi KNL における DRAM キャッシュ(EDC)を設定しない状態での LLC ミス数は L2 ミス数に相当し，論文[7]に記載した perf の predefined event である cach_misses を用いた方法がそのまま利用可能である．しかしながら，3D Xpoint の耐久性対策として併用が予想される DRAM キャッシュ(EDC)を設定した状態における LLC ミス数の測定は，上記の方法ではできない．

一方，Intel(R) Xeon Phi(TM) Processor Performance Monitoring Reference Manual Volume 2: Events[15] (以下イベントマニュアルと呼ぶ)によれば，Xeon Phi KNL 上には外部 DIMM アクセス数に相当する DRAM 制御信号(CAS)を直接測定可能な性能カウンタが存在することが確認できる．これは EDC の設定状況によらず用いることができる上，OoO 機構の全ての効果(投機的なメモリアクセスによる並列メモリアクセスの効果を含む)が反映された結果の DIMM アクセス数である．

KNL には EDC が 6 個存在するため，その 6 個すべてに関して適切な perf コマンドで CAS をカウントし，合算することで全 DIMM アクセス数が求まると考えられる．以上の測定方針に基づいて提案する DIMM アクセス数測定 perf コマンドは以下のとおりである．

```
sudo perf stat -r 10 -a -e
uncore_imc_0/event=0x03,umask=0x03,name=CAS0/ -e
uncore_imc_1/event=0x03,umask=0x03,name=CAS1/ -e
uncore_imc_2/event=0x03,umask=0x03,name=CAS2/ -e
uncore_imc_3/event=0x03,umask=0x03,name=CAS3/ -e
uncore_imc_4/event=0x03,umask=0x03,name=CAS4/ -e
uncore_imc_5/event=0x03,umask=0x03,name=CAS5/ アプリコマンド
```

この提案コマンドで測定した DIMM アクセス数を，論文[7]に記載した perf を用いた LLC ミス数を用いた簡易的遅延感度測定法の cache_misses と置き換えることで，投機的なメモリアクセスによる並列メモリアクセスの効果が反映された遅延感度を測定できる．

ただし，KNL 上に 6 個存在するメモリコントローラブロック(IMC)内に存在する DRAM 制御信号をカウントするカウンタであるためか，筆者の環境では root 権限で全プロセスのメモリアクセスの合計(perf の-a オプション付き)でしか測定できないという制約があった．このため他のジョブ投入を排除することが必要で，OS ノイズなどの影響を受けやすく，測定においては(perf の-r オプション付きで)繰り返し数を増やしてばらつきも測定することが望ましい．これは誤差が増加する測定法であり，その解消のために測定スループットを 1/10 程度に低下させてしまうという欠点がある．

5.2 提案手法(2) EDC ミス数測定法

前述のとおり DRAM キャッシュ(EDC)を設定した状態における LLC ミス数の測定は，論文[7]に記載した方法ではできない．よって，本節ではその測定法を提案する．

KNL には EDC が 8 個存在するため，その 8 個すべてに関して適切な perf コマンドで EDC ミス数をカウントし，合算することで LLC ミス数を求めることができる．イベント名や Umask 値はイベントマニュアル[15]中に公開されているものを指定する．以上の測定方針に基づいて提案する EDC ミス数測定 perf コマンドは以下のとおりである．

```
sudo perf stat -r 10 –a -e
uncore_edc_uclk_0/event=0x02,umask=0x1C,name=EDC0-misses/ -e
uncore_edc_uclk_1/event=0x02,umask=0x1C,name=EDC1-misses/ -e
uncore_edc_uclk_2/event=0x02,umask=0x1C,name=EDC2-misses/ -e
uncore_edc_uclk_3/event=0x02,umask=0x1C,name=EDC3-misses/ -e
uncore_edc_uclk_4/event=0x02,umask=0x1C,name=EDC4-misses/ -e
uncore_edc_uclk_5/event=0x02,umask=0x1C,name=EDC5-misses/ -e
uncore_edc_uclk_6/event=0x02,umask=0x1C,name=EDC6-misses/ -e
uncore_edc_uclk_7/event=0x02,umask=0x1C,name=EDC7-misses/ アプリコマンド
```

perf コマンドのバージョンや Linux カーネルのバージョンの組合せによっては動作しない可能性の確認はできていないが，筆者の環境においては上記のコマンドで測定が可能であった．

この提案コマンドで測定した LLC ミス数を論文[7]に記載した perf を用いた LLC ミス数を用いた簡易的遅延感度測定法の cache_misses と置き換えることで，論文[7]と同等の精度，すなわち OoO 機構の一部の効果(投機的なメモリアクセスによる並列メモリアクセスの効果)を省略する近似を用いた精度の遅延感度評価を行なうことができる．

上記提案を用いた KNL 上でのシングルスレッドでの評価により，バンド幅律速でない状況下での KNL 上のアプリの遅延感度が得られる．

5.3 提案手法(3) Outstanding リード数測定法

提案手法(2)における KNL 上の誤差を補正する手段として，投機的なメモリアクセス数の測定法を提案する．イベントマニュアル[15]によると KNL には前章で紹介した Skylake と異なり，Outstanding リードに関するカウンタは 1 種類しか取れない．

このカウンタは root 権限を必要としないため高スループットで評価が可能である．Xeon 系とは大きく異なり，Offcore_Response Event (名称 OFFCORE_RESP，event select=0xB7)であり，Umask=0x01 とした上で，さらに MSR_OFFCORE_RESP0 レジスタに細かい測定条件を設定することで測れるので，単に event select と Umask の組で指定できた他のケースと perf コマンドの形が異なるので注意が必要である．DIMM への Outstanding リードの場合以下の bit を 1 とする． Request type として bit0 (DEMAND_DATA_RD), Response type として bit23 (DRAM_NEAR), bit24 (DRAM_FAR), bit31 (SNOOP_NONE), bit32 (NO_SNOOP_NEEDED), Outstanding requests として bit38 (OUTSTANDING)

以上の測定方針に基づいて提案する DIMM への Outstanding リード数を測定する perf コマンドは以下のとおりである．

```
perf stat -e cycles -e
cpu/event=0xB7,umask=0x01,offcore_rsp=0x4181800001,name=OUTSTANDING_RD_DRAM/ アプリコマンド
```

上記のコマンドで得られる *OUTSTANDING_RD_DRAM* の値を *cycles* で除算したものが DRAM への平均 Outstanding リード数となる．

5.4 提案手法(4) Xeon Phi 上の Outstanding 補正付遅延感度測定法

提案手法(2)(3)を組合せることで Outstanding リードの並列実行に伴う遅延感度誤差の補正方法を提案する．DRAM への Outstanding リードは厳密には 100%並列実行されるわけではない．アクセスするアドレス列によって所定のサイクル数ずらされる．例えばバンクコンフリクトがあれば所定のサイクル数待たされ逐次化される．メモリコントローラの作りによって理論的なメモリレベル並列度と実際に引き出されるメモリレベル並列度の高低は生じうる．一方，メモリ遅延を大幅に増加させると，メモリにアドレスを送信したり，リードデータを受信するのに必要なサイクル数に対して、メモリアクセス遅延のサイクル数が大幅に大きくなる状況が生まれる．高バンド幅提供のためには 3D Xpoint 等のメモリ側もバンク数を豊富に設定している可能性が高い．このため，メモリコントローラ的には高遅延メモリに対しては Outstanding リード要求を効率的に並列実行可能になると予測される．よって，DRAM への Outstanding リードは高遅延メモリではほぼ 100%並列実行されると仮定し，提案手法(2)で得られる遅延感度を平均 Outstanding リード数で除算することで補正を行うことを提案する．

6. 性能評価

提案手法の手法を応用した主記憶遅延増加時のアプリケーション性能の評価を実施した．さらに，提案補正法の元となる KNL 上での Outstanding リード数についても評価を行なった．本章ではその実験内容と結果について述べる．

6.1 測定環境

表 1 に本実験で用いた測定環境の仕様を示す．

表 1 測定環境の仕様

CPU name	Intel(R) Xeon Phi(R) 7210 (KNL: Knights Landing)
Frequency	Base:1.3GHz, Turbo boost:1.5GHz
#CPU processors	64 ores x 4SMTs
Cahce	L1: 32KB/core, L2: 32MB(分散配置), L3(EDC):8GB
MCDRAM	MAX 16KB (この半分の 8GB を EDC として使用)
DIMM	Type: DDR4-2133(DIMM)×6ch, Latency: 175ns, Capacity: 192GB Bandwidth: 1Read+1Write : 70.7GB/s / MAX
COMPILER	Intel icc ver. 17.0.1 (for workload (1)(2)) gcc (GNU C Compiler) version 4.8.5

DRAM 遅延は local,remote 分を Intel Memory Latency Checker，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．

6.2 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[16]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[17]から得た 208 種の行列に対する SpMV．DRAM キャッシュの有無の差を観察するために論文[7]と共通のもの．1 スレッド実行．

(2) Stream ベンチマーク．マルチスレッド時にバンド幅が主要な性能律速要因になるアプリケーション．配列サイズを 2 倍ずつ増加させ，フットプリントが最大 24GB になるまで変化させ，64 スレッド実行．

6.3 LIS を用いた各種格納方式の SpMV

図 3 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．メモリ遅延は 300ns, 500ns, 750ns, 1000ns の 4 種類についてプロットした．この実験は提案方式(2)のカウンタを用い，繰り返し数 10 回に設定して 134 時間(6 日弱)の実行時間を要した．BSC と Samsung のチームが 2016 年発表した方法で同様の結果を得る場合は TaskSim の OoO パイプラインを模擬しない MEM モードの実行時間だけで 1 遅延につき 30 年，4 遅延だと 120 年かかる規模の実験である．

図 3 の結果は論文[7]にて発表した Haswell ベースで DRAM キャッシュが無い構成の 1000ns における各々同じ条件の減速率より概ね半分程度に緩和されている．論文[7]においても SpMV は遅延感度が比較的低いと予測されていたが，3D Xpoint 利用時により近い構成ではさらに感度が低いことが示された．

格納方式 DIA および ELL と，それ以外で明らかに違う傾向を示していることがわかる．DIA および ELL は 208 種のうち全てが最後まで完走したわけではなく，一部の行列は Segmentation fault を表示して異常終了した．異常終了しないまでも遅延感度が 20 近くに異常に上昇しているケースがある．DIA は特に感度が上がっているものが多い．

ELL の場合は密に近い行があると殆ど密行列と同等のメモリ領域を必要としてしまうため，このような激しい現象が起きてしまったと考えられる．異常終了しないまでも遅延感度が 20 近くに異常に上昇しているケースが見受けられる．密行列並みにフットプリントが上昇した結果，遅延感度が上がっているケースと考えられる．このことは，疎行列より密行列の方が遅延感度が高くなる可能性を示唆しているとも考えられる．

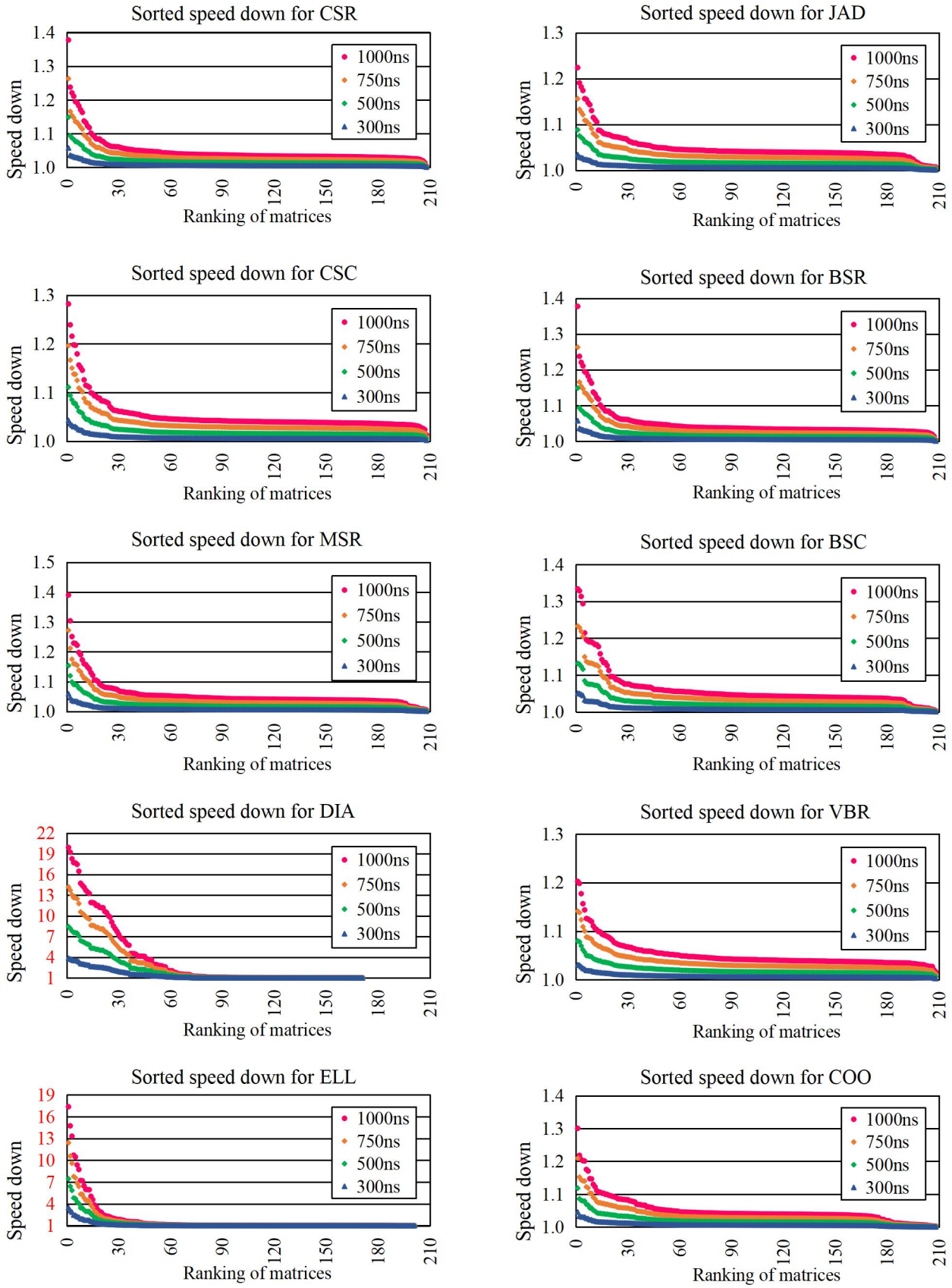

図 3　KNL 上での LIS の spmvtest5 を用いた 10 種の格納方式における 208 種の正方行列に対する SpMV の予測減速率

6.4 Stream ベンチマーク

マルチスレッド時にバンド幅が主要な律速要因になるアプリケーションとして，Stream ベンチマークを用いる．Stream はバンド幅律速のアプリケーション性能モデルである Roofline モデルの基礎となっている重要なベンチマークである．8GB もの大容量の DRAM キャッシュを有する測定環境であるため，配列サイズを 2 倍ずつ増加させ，フットプリントが最大 24GB になるまで変化させた．64 スレッド実行した際の主記憶遅延による平均 Outstanding リード数の時系列変化を図 4 に示す. 測定には提案手法(3)を-I 100 オプション(100ms おきの周期での測定指定)付きで用いた．

終わり際に一瞬上がるがバンド幅を計測するメインループを実行中は 2 程度までしか上がらない．メニーコア上でも当初恐れていたほど Outstanding リードの並列実行による大きな誤差は無いのではないかと感じられるデータが得られた. この値は DRAM キャッシュのサイズを超えても大きな変動は無い．

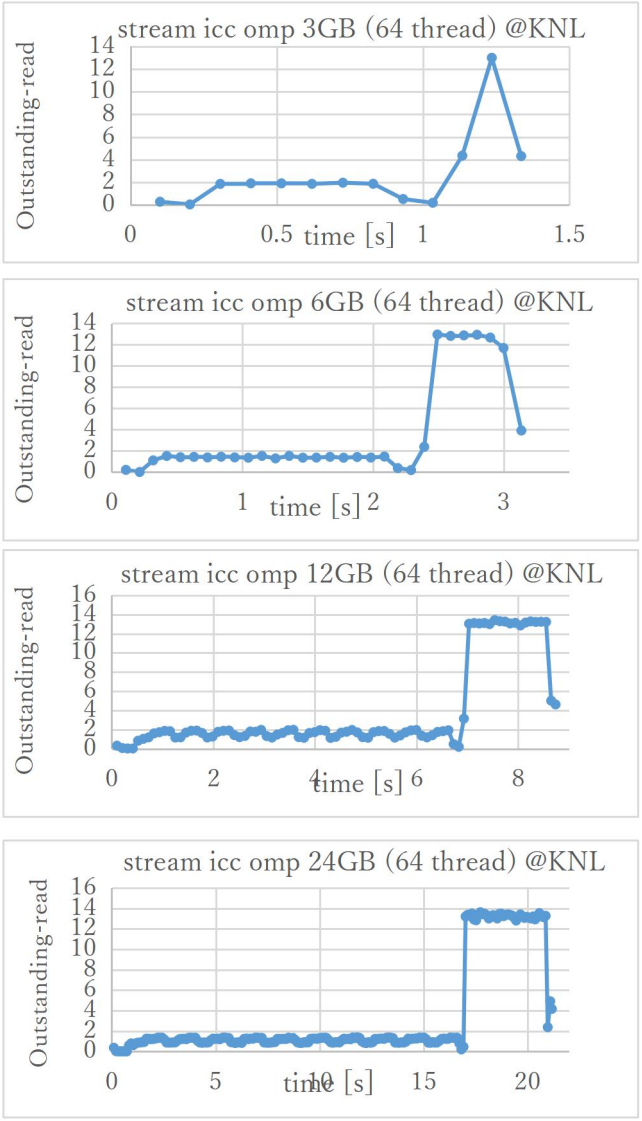

図 4　KNL 上での Stream の 64 スレッドによる実行時の Outstanding リード数の時系列変化

図 4 に示されるように，アプリケーションが複数(今回の場合は大別して 2 つ)のフェーズから成り立っている場合は Outstanding リード数にも時間軸上でのムラがある. 正確性を期すのであればそれを考慮した補正も考える余地があることが判った．時系列で採取可能であるので，最も正確にするには周期ごとに補正すれば良い．しかし，今回の実験で示されているように通常あまり誤差が大きくないようであれば，そこまで厳密に補正しなくても傾向はつかめると考えられる．

7. まとめ

本報告では Intel Xeon Phi KNL 上で実行可能な主記憶遅延感度評価法を提案した．そのうちの 1 つを使って筆者らによる前回の発表において示した疎行列ベクトル積の LIS とフロリダ行列 208 種を組み合わせた実験と同等なものを DRAM キャッシュ付の状態で実行した．その結果，DRAM キャッシュの無い状態の Haswell ベースの環境よりも半分程度まで遅延感度は下がることを観測した．BSC と Samsung のチームが 2016 年発表した方法で同様の結果を得る場合は TaskSim の OoO パイプラインを模擬しない MEM モードの実行時間だけで 1 遅延につき 30 年，4 遅延だと 120 年かかると予想される．

さらにメニーコアで重要性が高まるマルチスレッド実行における Outstanding リード数の増加を KNL 上で Stream の 64 スレッド実行により評価した．データサイズによらずにその値は 2 程度であるので，補正は必要ながら当初思っていたほどは大きくないことを確認した．

今後は提案した二つの類似手法の優劣を実験により評価することが課題である．優れた方法を絞り込んだ後により多くの種類の大きなフットプリントを有するアプリケーションの評価を行なう予定である．

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．

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] 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.
[7] 田邊昇，遠藤敏夫. 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. 第 158 回ハイパフォーマンスコンピューティング研究会，2017, Vol.2017-HPC-158,No.15.
[8] Kazi Asifuzzaman, Milan Pavlovic, Milan Radulovic, David Zaragoza, Ohseong Kwon, Kyung-Chang Ryoo, and Petar Radojkovi?. Performance Impact of a Slower Main Memory: A case study of STT-MRAM in HPC. In Proceedings of the Second International Symposium on Memory Systems (MEMSYS '16), 2016, p.40-49.
[9] Alejandro Rico, Felipe Cabarcas, Carlos Villavieja, Milan Pavlovic, Augusto Vega, Yoav Etsion, Alex Ramirez, and Mateo Valero. On the simulation of large-scale architectures using multiple application abstraction levels. 2015. ACM Trans. Archit. Code Optim. 8, 4, Article 36.
[10] 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.
[11] 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.
[12] 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.
[13] 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.
[14] 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.
[15] Intel(R) Xeon Phi(TM) Processor Performance Monitoring Reference Manual Volume 2: Events
[16] 反復解法ライブラリ LIS. http://www.ssisc.org/lis/
[17] Tim Davis. The SuiteSparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/