Vol.2020-DPS-185 No.8
2020/12/22

クラウドコンピューティングを支える仮想計算機技術の性能分析

島谷 隼生[1]　佐藤 将也[1]　谷口 秀夫[1]

概要：クラウドコンピューティングは，仮想計算機技術を用いており，その利用は拡大している．このクラウドコンピューティングのシステムは，マルチコアの CPU を複数搭載した計算機システムを利用している．この計算機システムは，NUMA 構成であり，プログラムのメモリ操作速度はプログラムを実行するコアとメモリ配置の関係に依存し一定でない．しかし，今日のクラウドコンピューティングでは，この性質を考慮していないと考えられる．本稿では，NUMA 環境を利用して実現された仮想計算機上の性能分析を述べ，その影響を明らかにし，この性質を考慮した負荷分散の必要性を述べる．

キーワード：クラウドコンピューティング，仮想計算機，NUMA

1. はじめに

クラウドコンピューティングにおける仮想計算機（VM：Virtual Machine）技術の利用が拡大している．このようなクラウドコンピューティングのシステムでは，マルチコアの CPU を複数搭載した計算機が利用される．この計算機の多くは，NUMA（Non-Uniform Memory Access）構成を採用している [1].

NUMA 構成の計算機では，プロセッサとプロセッサのアクセスするメモリが近い場合はメモリ操作が速くなり，遠い場合はメモリ操作が遅くなる．このため，プログラムのメモリ操作速度は，プログラムの走行コアと格納位置の関係に依存し一定でない [2]．この特性を考慮したプログラムの走行コアと格納位置の最適化に関する研究がされている．Lepers[3] は，NUMA 上で動作するプログラム性能について，走行コアとメモリ配置，およびプロセッサ間を繋ぐバスのバンド幅を考慮した分析を行っている．NUMA 環境上の VM でプログラムを実行する場合，VM は物理計算機の NUMA 構成を認識しない [1], [4] ため，プログラムのメモリ操作速度は，VMM（Virtual Machine Monitor）によるコアとメモリの割り当てに依存する．そこで，VMM によるコアとメモリの割り当てを最適化する研究がされている [4], [5], [6], [7]．しかし，これらの研究では，VM が使用するコアとメモリの関係がプログラムのメモリ操作速度に与える影響について，VMM のメモリ格納位置を含めた分析がされていない．

[1] 岡山大学 大学院自然科学研究科

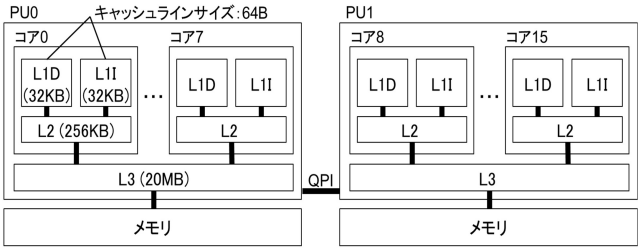

図 1　ハードウェア構成例

本稿では，NUMA 環境を利用して実現された仮想計算機上におけるプログラムのメモリ操作速度の性能分析について述べる．また，分析結果から VMM，VM 上の OS（Operating System），および AP（Application Program）のプログラムのメモリ格納位置と走行コアの組み合わせによるメモリ操作速度の違いを明らかにし，これを考慮した負荷分散の必要性を述べる．

2. NUMA 構成の計算機

クラウドコンピューティングのシステムでは，マルチコアの CPU を複数搭載した NUMA 構成の計算機を利用している．このような NUMA 構成の計算機の例として，Intel Xeon E5-2630 v3 を 2 基搭載したハードウェア構成例を**図 1** に示す．プロセッサ（PU）は，8 コアを有し，各コアに 32 KB の L1 データキャッシュ（L1D），L1 命令キャッシュ（L1I），および 256 KB の L2 キャッシュを保持しており，1PU 内の全コアで 20 MB の L3 キャッシュを共有する．また，PU 間は，QPI（QuickPath Interconnect）を介

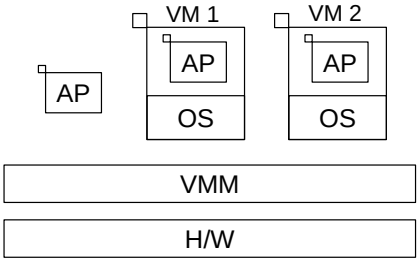

図 2　仮想計算機の構成

して接続されており，他 PU に接続されたメモリは，QPI を介して操作できる．ただし，このメモリ（リモートメモリ）を操作する場合，QPI を介することによるオーバヘッドが生じ，自 PU に接続されたメモリ（ローカルメモリ）への操作に比べて，メモリ操作速度が遅くなる．

3. 仮想計算機

3.1 基本構造

VM の基本構成を**図 2** に示す．VM は，ハードウェア資源を VMM が抽象化することにより実現される．また，1 つの VMM 上で複数の VM が動作できる．VMM は，VM に対して，CPU，メモリ，入出力デバイス等を提供する．物理計算機資源（外部記憶装置，NIC，および CPU レジスタ等）へのアクセスが必要となる処理が VM 上で発生した場合は，VM から VMM へ制御が遷移し，必要に応じてエミュレーションを行う．なお，VM から VMM へ制御が遷移した場合，VMM の処理は VM が実行されているコアで実行される．

VMM は，Linux の KVM[8] や FreeBSD の bhyve[9] のように，OS に VM 機能を追加した形で実装される場合が多い．したがって，VMM 上において，VM はプロセスとして実現され，AP プロセスも走行し得る．

3.2 プログラムの格納位置

NUMA 環境で VM を構築する場合の仮想計算機の構成を**図 3** に示す．

VMM は，メモリの若番アドレス域，つまり PU0 側のメモリ領域に格納される．

VM 上の OS と AP は，メモリの使用状況によって，PU0 側のメモリ領域，PU1 側のメモリ領域，あるいは両方にまたがって格納される．例えば，1 つ目の VM1 がメモリに格納されており，PU0 側のメモリ領域に十分な空きがない場合，2 つ目の VM2 は，PU1 側のメモリ領域に格納される．

3.3 問題点

図 3 に示したように，NUMA 環境での仮想計算機では，プログラムの格納位置と走行コアの関係により，プログラムのメモリ操作速度は異なる．したがって，同じ処理を行っても処理時間に長短が生じる問題がある．

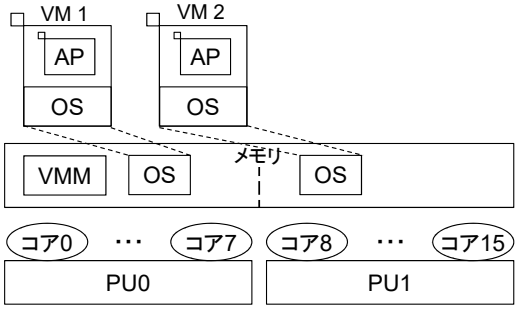

図 3　NUMA 環境での仮想計算機

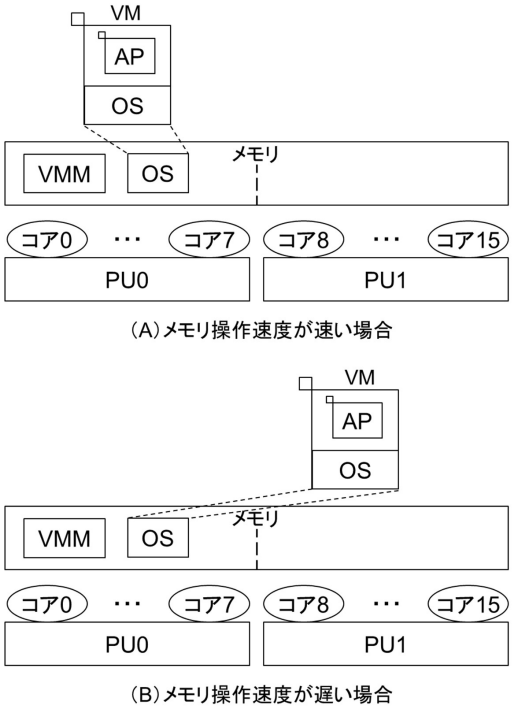

図 4　効率的な負荷分散の例

例として，VM2 上の OS や AP が PU0 側のメモリ領域に格納された場合を考える．VM2 の走行コアが PU0 側である場合，VM 上で動作する OS や AP のメモリ操作はリモートメモリへの操作となり遅い．一方，VMM のメモリ操作はローカルメモリへの操作となり速い．また，VM2 の走行コアが PU1 側である場合，VM 上で動作する OS や AP のメモリ操作は速く，VMM のメモリ操作は遅い．

したがって，メモリ操作速度が速くなるように効率的に負荷分散するためには，VM 上の OS や AP を格納したメモリ領域，VM の走行コア，および VMM のメモリ領域との関係を考慮する必要がある．例として，VM 上のプログラムのメモリ操作が速くなる場合と遅くなる場合を**図 4** に示す．（A）は，VM 上の OS や AP が格納されたメモリ領域と VM の走行コアが PU0 側であり，VM 上で動作する OS や AP と VMM のメモリ操作がローカルメモリへの操作となるため，VM 上のプログラムのメモリアクセス速度が速い．（B）は，VM 上の OS や AP が格納されたメモリ

表 1　AP や OS に関する条件

プログラムの格納位置	プログラムの走行コア	APの走行位置
タイプ1:PU0側(16 GB内)	タイプA:PU0のコア	タイプX:VMM上で走行
タイプ2:PU1側(16 GB外)	タイプB:PU1のコア	タイプY:VMのOS上で走行

表 2　評価種別

タイプ	評価条件の組み合わせ
0A	1-A-X(=2-B-X): PU0側に格納したプログラムをPU0のコアを用いてVMM上で実行
0B	1-B-X(=1-A-X): PU0側に格納したプログラムをPU1のコアを用いてVMM上で実行
1A	1-A-Y: PU0側に格納したプログラムをPU0のコアを用いてVMのOS上で実行
1B	1-B-Y: PU0側に格納したプログラムをPU1のコアを用いてVMのOS上で実行
2A	2-A-Y: PU1側に格納したプログラムをPU0のコアを用いてVMのOS上で実行
2B	2-B-Y: PU1側に格納したプログラムをPU1のコアを用いてVMのOS上で実行

領域と VM の走行コアがそれぞれ PU0 側と PU1 側であり，VM 上で動作する OS や AP と VMM のメモリ操作がリモートメモリへの操作となるため，VM 上のプログラムのメモリアクセス速度が遅い．

4. 性能の分析

4.1 評価観点と評価環境

NUMA 環境上の VM では, プログラム走行時のメモリ操作速度は，以下の 2 つの組み合わせによって変化する．
(1) AP，OS，および VMM のプログラムがメモリ上のどこに格納されているか
(2) AP，OS，および VMM のプログラムがどのコアで走行されるか
これらを踏まえ，評価種別を決定する．VMM のプログラムは，PU0 側のメモリ領域に格納される．そこで，AP と OS について，プログラムの格納位置とプログラムの走行コア，さらに AP の走行位置が VMM か VM 上の OS か否かを加えて，評価種別を用意する．AP や OS に関する条件を**表 1** に示し，その組み合わせとして評価種別を**表 2** に示す．AP と OS のプログラムのメモリ格納位置と走行コアの関係より，(1-A-X) の組み合わせは (2-B-X) の組み合わせと同じ関係になり，(1-B-X) の組み合わせは (2-A-X) の組み合わせと同じ関係になる．また，評価種別の様子を**図 5** に示す．プログラムの走行コアがタイプ A の時にはコア 7 を使用し，タイプ B の時にはコア 15 を使用する．また，タイプ 1A が図 4（A）で示した場合に該当し，タイプ 1B が図 4（B）で示した場合に該当する．

AP の処理は 2 種類とした．この評価用プログラムの処理流れを**図 6** に示す．処理 P は，mmap() システムコールを用いて AP 領域に指定された size 分のバッファを確保し，AP が 4 B データを offset 間隔でバッファの終端まで書き込む処理を n 回繰り返す．処理 Q は，buf_get() システムコールを用いて OS 領域に指定された size 分のバッファを確保し，buf_write() システムコールを繰り返し発行して OS が 4 B データを offset 間隔でバッファの終端まで書き込む処理を n 回繰り返す．

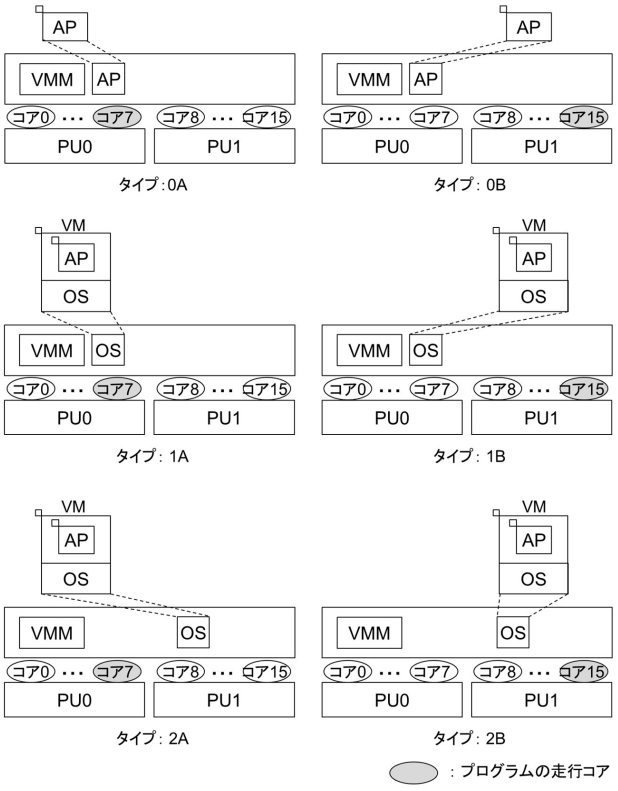

図 5　評価種別の様子

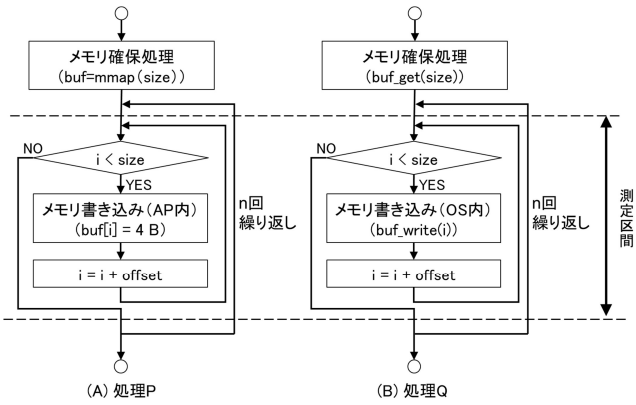

図 6　評価用プログラム

評価環境を**表 3** に示す．評価に用いた計算機は，物理 8 コアを搭載した Intel Xeon CPU E5-2630 v3 を 2 基搭載しており，図 1 と同じ構成である．VMM は bhyve である．

4.2 評価結果と考察

4.2.1 VMM 上のプログラムのメモリ操作速度

VMM 上で AP を走行させた場合（タイプ 0A とタイプ 0B），つまり FreeBSD の OS 上で評価用プログラムを走行させた場合として，データ書き込み間隔（図 6 の offset）が

表 3 評価環境

物理計算機の構成	
CPU	Intel Xeon E5-2630 v3 2.40 GHz (8 コア× 2 ソケット) L1 キャッシュ：32 KB L2 キャッシュ：256 KB L3 キャッシュ：20 MB キャッシュラインサイズ：64 B
メモリ	DDR4 32 GB
OS	FreeBSD 11.3-RELEASE
VMM	bhyve
VM の構成	
CPU	1 コア
メモリ	1 GB
OS	FreeBSD 11.3-RELEASE

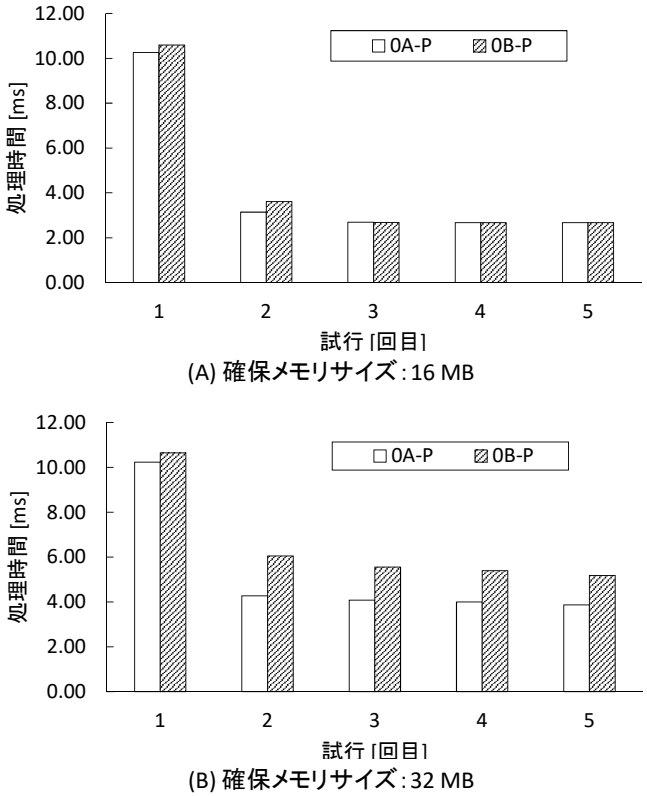

図 7 VMM を介さない場合のメモリ書き込み処理時間

64 B（キャッシュラインサイズ）の時の処理時間を**図 7** に示す．なお（A）(B) とも，メモリ書き込み回数は，同数（確保メモリサイズが 16 MB の場合の回数）である．図 7 より，以下のことがわかる．

(1) 確保メモリサイズに関わらず，1 回目の処理時間が 2 回目以降よりも長い．これは，データ書き込み時に，1 回目はページ例外が発生するものの，2 回目以降はページ例外が発生しないためである．このため，2 回目以降の処理時間は同様である．

(2) 0A-P と 0B-P に関わらず，2 回目以降の処理時間について，確保メモリサイズが 32 MB の場合は確保メモリサイズが 16 MB の場合より長い．これは，確保メモリサイズが 16 MB の場合はデータ書き込みが L3 キャッシュ（20 MB）にヒットするが，確保メモリサイズが 32 MB の場合はデータ書き込みが L3 キャッシュ（20 MB）にヒットしないためである．なお，1 回目については，確保メモリサイズの大小に関係なく L3 キャッシュ（20 MB）にヒットしないため，処理時間差はない．

(3) 確保メモリサイズが 32 MB の場合，0B-P の処理時間は 0A-P の処理時間より長い．例えば，2 回目では，0B-P は 6.05 ms で 0A-P は 4.27 ms であり，1.78 ms（約 1.42 倍）長い．これは，確保メモリサイズが 32 MB の場合は 2 回目以降でもデータ書き込みが L3 キャッシュ（20 MB）にヒットせず，0B-P ではリモートメモリへの操作になるためである．この現象は，確保メモリサイズが 16 MB の場合でも 1 回目で見られるが，2 回目以降はデータ書き込みが L3 キャッシュ（20MB）にヒットするため，0B-P の処理時間と 0A-P の処理時間は同様になる．

上記のことから，ページ例外が発生しないと仮定してメモリ操作速度に着目すると，0A-P の 2 回目以降の処理時間が最も短く約 2.68 ms であり，0B-P の 2 回目以降の処理時間が最も長く約 5.57 ms である．したがって，プログラムのメモリ格納位置と走行コアの関係により，最大 2.07 倍の性能差が発生する．

4.2.2 VM の OS 上のプログラムのメモリ操作速度

VM の OS 上で AP を走行させた場合（タイプ 1A，タイプ 1B，タイプ 2A，およびタイプ 2B），つまり VMM の管理する VM の OS 上で評価用プログラムを走行させた場合として，データ書き込み間隔が 64 B，確保メモリサイズが 32 MB の時の処理時間を**図 8** に示す．なお，比較のために図 7 の処理時間を併記する．図 8 より以下のことがわかる．

(1) 試行回数に関わらず，1B-P の処理時間は 1A-P の処理時間より長い．最も処理時間差が大きい場合として，2 回目では，1B-P の処理時間が 1A-P の処理時間よりも 3.66 ミリ秒（約 1.42 倍) 長い．これは，1A-P はローカルメモリへの操作となり高速であるが，1B-P はリモートメモリへの操作となり低速であるためである．これは 2A-P と 2B-P でも同様である．また，これは 0B-P と 0A-P の関係と同様であり，VM の OS 上のプログラムも，VMM 上のプログラムと同様の性能差が生じている．

(2) 評価用プログラムに関わらず，1 回目の処理時間は，タイプ 1A が最も短い．これは，タイプ 1A では，VMM，VM 上の OS，および AP のメモリ操作がローカルメモリへの操作になるためである．

(3) 処理 Q の評価用プログラムを用いた場合，1 回目と 2 回目以降の処理時間差がない．これは，OS 領域における動的メモリ確保によって確保したメモリには，すでに物理ページが割り当てられており，1 回目のデータ書き込み時にページ例外が発生しないためである．

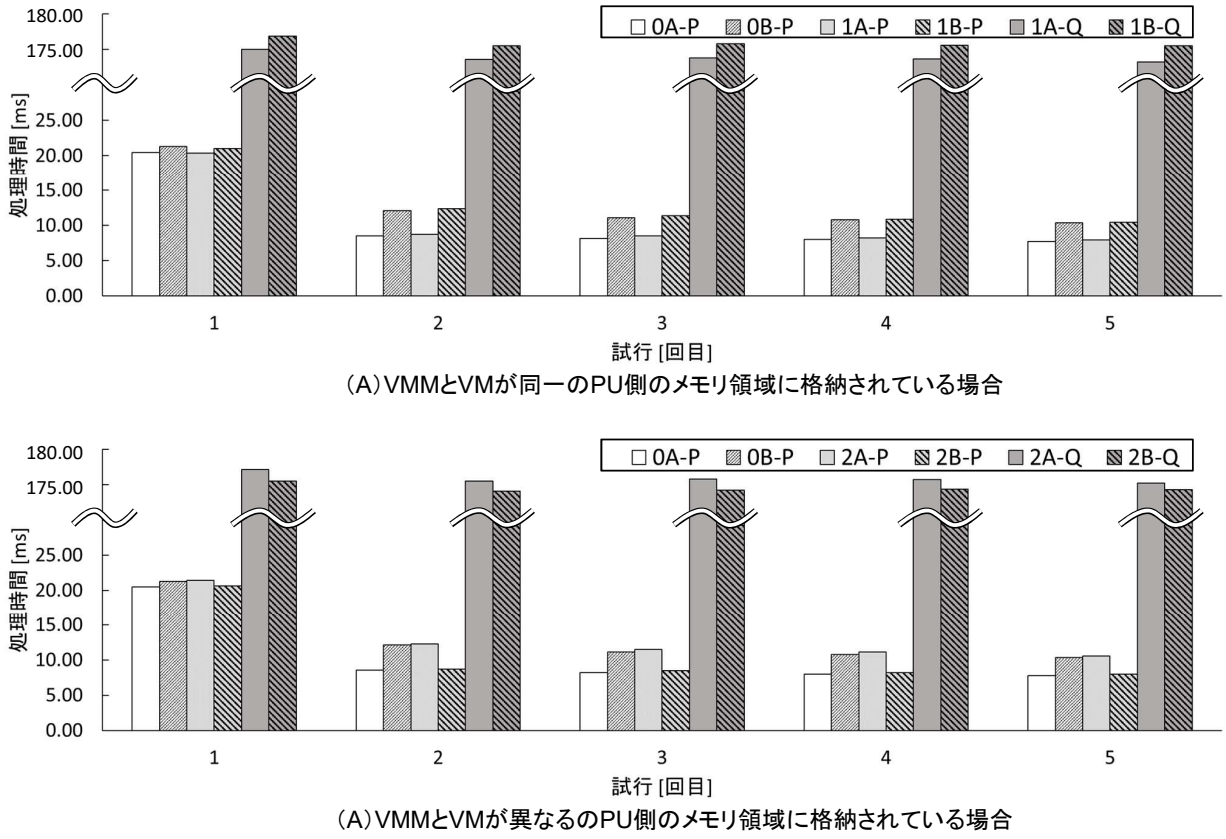

図 8　VM を介する場合のメモリ書き込み処理時間

(4) タイプに関わらず, 処理 Q の処理時間は処理 P の処理時間よりも長い．例えば，2A-Q と 2A-P の差は約 164.20 ms であり，2B-Q と 2B-P の差は約 165.90 ms とほぼ同じである．これは，処理 Q では，システムコール呼び出しに伴いコンテキストスイッチが発生しているためである．ただし，処理 Q においても，(1) で述べた性能差が生じている．

上記のことから，コンテキストスイッチの有無に関わらず，VM の OS 上でプログラムを走行させた場合においても，プログラム格納位置と走行コアの関係によるメモリ操作速度への影響が存在する．その中でも差が大きくなる場合に着目すると，1A-P の 2 回目の処理時間が約 8.73 ms であり，1B-P の 2 回目の処理時間が約 12.4 ms である．したがって，プログラムのメモリ格納位置と走行コアの関係により，約 1.41 倍の性能差が発生する．このことから，走行するプログラムが VMM 上にあるか VM 上にあるかに関わらず，同じメモリ操作処理を実行する場合は，プログラムのメモリ格納位置と走行コアの関係により，性能差が発生することがわかる．

4.2.3　キャッシュによる影響

4.2.1 項の (3) から，L3 キャッシュにヒットする場合，プログラムの格納位置と走行コアの遠近によるメモリ書き込み時間の差がなくなることがわかった．したがって，L1 キャッシュあるいは L2 キャッシュにヒットする場合

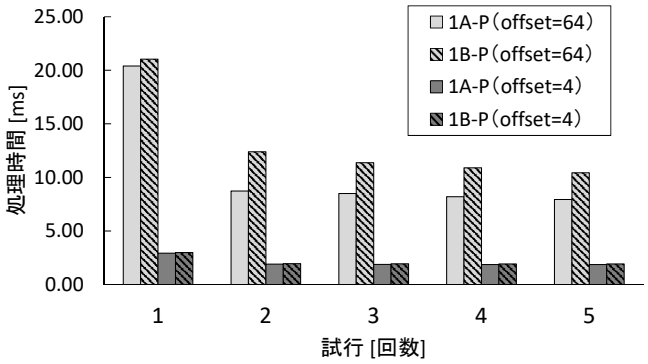

図 9　書き込み間隔の違いによるメモリ書き込み処理時間の比較

も，メモリ書き込み時間の差がなくなることが予想される．図 6(A) において，データ書き込み間隔が 64 B と 4 B の場合の処理時間を**図 9** に示す．図 9 より，以下のことがわかる．

(1) offset=4 の場合，1A-P と 1B-P のメモリ書き込み処理時間差はない．これは，メモリ書き込みが L1 キャッシュあるいは L2 キャッシュにヒットするためである．つまり，データ書き込み間隔がキャッシュラインサイズよりも小さい場合にキャッシュヒットが発生していると考えられる．このため，キャッシュラインサイズ間隔でメモリ書き込みを行う場合よりも実メモリ操作の回数が減少する．また，プログラムの格納位置と走行コアが遠い場合，実メ

モリ操作の処理時間が増加するため，キャッシュヒットの効果が大きくなる．

(2) 1B-P(offset=64) の 1 回目の処理時間は，21.03 ms である．これは，VM 上のプログラムの格納位置と走行コアが遠く，かつページ例外処理が発生し，かつ CPU キャッシュミスによる実メモリ操作が発生しているため，最悪ケースの処理時間といえる．また，1A-P(offset=1) の 2 回目以降の処理時間の平均は，1.93 ms である．これは，VM 上のプログラムの格納位置と走行コアが近く，かつページ例外処理が発生せず，かつキャッシュヒットによって実メモリ操作の回数が減少しているため，最良ケースの処理時間といえる．これらから，VM 上のプログラムのメモリ操作時間は，この最良ケースと最悪ケースの区間内の時間に収まることがわかる．

5. おわりに

本稿では，NUMA 環境において，VMM，VM 上の OS，および AP のプログラムの格納位置と走行コアの関係によるプログラムのメモリ操作速度の違いを明らかにし，この性質を考慮した負荷分散の必要性を述べた．分析の結果から，VMM 上で AP を走行させた場合，プログラムのメモリ格納位置と走行コアの関係により，最大 2.07 倍の性能差が発生することがわかった．また，VM の OS 上で AP を走行させた場合，プログラムのメモリ格納位置と走行コアの関係により，最大 1.41 倍の性能差が発生することがわかった．このことから，VM 実行環境においても，NUMA の特性を考慮したプログラム配置による負荷分散が必要であることを示した．さらに，プログラムのメモリ格納位置と走行コアが遠い場合，キャッシュヒットによる効果が大きくなることを確認した．

謝辞 本研究の一部は，JSPS KAKENHI 18K11244，および共同研究（株式会社富士通研究所）による．

参考文献

[1] Song Wu, Huahua Sun, Like Zhou, Qingtian Gan, and-Hai Jin.: vProbe: Scheduling virtual machines on numasystems. *InProceedings of IEEE International Conference on Cluster Computing(CLUSTER)*, pp. 70–79 (2016).

[2] Christoph Lameter.: NUMA (Non-Uniform Memory Access): An Overview, *ACM Queue*, 11(7):40:40-40:51 (2013).

[3] Baptiste Lepers, Vivien Quéma, Alexandra Fedorova.: Thread and Memory Placement on NUMA Systems: Asymmetry Matters, *2015 USENIX Annual Technical Conference (USENIX ATC'15)*, pp. 277-289 (2015).

[4] 林 遼, 味曽野 雅史, 品川 高廣: ゲスト OS における NUMA 対応スケジューリング, 情報処理学会研究報告システムソフトウェアとオペレーティング・システム,Vol.2020-OS-148, No.11, pp. 1-7 (2020).

[5] Bao Bui, Djob Mvondo, Boris Teabe, et al.: When extended para-virtualization (xpv) meets numa, *In Proceedings of the Fourteenth EuroSys Conference 2019(EuroSys'19)*, pp. 1-15 (2019).

[6] Jia Rao, Kun Wang, Xiaobo Zhou, and Cheng-ZhongXu.: Optimizing virtual machine scheduling in NUMA multicore systems, *In Proceedings of 19th InternationalSymposium on High Performance Computer Architec-ture (HPCA)*, pp. 306–317 (2013).

[7] Gauthier Voron, Gaël Thomas, Vivien Quéma, and Pierre Sens.2017.: An Interface to Implement NUMA Policies in the XenHypervisor, *In Proceedings of the Twelfth European Conferenceon Computer Systems (EuroSys'17)* pp. 453–467 (2017).

[8] KVM:MainPage—KVM,(online), available from ⟨https://www.linux-kvm.org/page/Main_Page⟩ (accessed 2020-11-25).

[9] Dexter, M.: BSD Hypervisor, (online), available from ⟨https://bhyve.org/⟩(accessed 2020-11-25)