Vol.2019-HPC-172 No.13
2019/12/18

ムーア則終焉直後に向けた高性能汎用計算機アーキテクチャの初期検討

田邊昇 [1]　田浦健次朗 [1]

概要：2025 年頃に半導体集積度向上が停止してムーア則が終焉する説が有力と言われている．一方，スーパーコンピュータやクラウドに代表される高性能汎用計算インフラの電力あたりの性能向上要求は留まるところを知らない．ムーア則終焉直後の時点で，量子コンピュータが全ての計算インフラを置き換える状態に成熟するとは考えにくく，何らかの形で従来型インフラの延命をはかる必要がある．ムーア則終焉直後には，ユーザ側に十分なソフトウェア的な対応をさせることは困難で，多くのアプリケーションの全面的な書き換え等を抑制した形で，終焉の影響を軽減させつつ性能向上を実現することが望まれる．本研究はそのようなニーズに対応する方策を考察する．本稿ではその方策として有望と思われる想定アーキテクチャを考察し，McIM(Massive cores in Memory side)というコンセプトを提案する．さらに，McIM を有効活用する例として配列間接参照の高速化アルゴリズムも提案する．

キーワード：ポストムーア，FLOPS to BYTES，スーパーコンピュータ，クラウド，ビッグデータ解析，In memory computing，Near data processing，DIMMnet，NVDIMM，階層記憶，Storage class memory，Memory disaggregation，疎行列，グラフ処理，間接参照，Gather，Prefetch，Helper thread，Many cores，Massive Threads

1. はじめに

半導体の集積度向上速度を示すムーアの法則は既に進化が鈍っており，2025 年頃には半導体の集積度向上が停止することが有望視されている．一方，スーパーコンピュータやクラウドに代表される高性能汎用計算インフラの性能や，電力削減の要求は留まるところを知らない．ビッグデータ解析のニーズや AI の特性上，データ量の大きさが質的向上に寄与するため，処理すべきデータ量の上昇が止まらないということが計算インフラへのニーズの背景になっている．

CPU の性能向上の歴史においては，冷却の限界から周波数を増やす戦略は先に破綻した．ムーア則が継続している間は CPU チップ内のコア数を増やす戦略によって性能向上は順調に達成されてきた．ムーア則が終焉すると，その戦略も限界を迎える．その後の性能向上ニーズに応えていくためには，半導体集積度向上に依存しない戦略を生み出していく必要がある．

これまでの CPU とは全く別のデバイス技術や動作原理に基盤を置く量子コンピュータの開発競争も激化しており，様々な研究成果が出てきている．ゲート方式の量子コンピュータでは従来システムを汎用的に置き換えるには 10〜20 年といった長い期間がかかると予測されている．

つまり，ムーア則が終焉してしまう今から 5,6 年先から，ゲート型量子コンピュータへの置き換えができるほどに成熟するまで，およそ 10 年間程度の手詰まり期間が出現する可能性が高い．その手詰まり期間に対して，現状では実現可能性が高く，有望な解決策が十分に提示されていない．本研究はそのような中期的課題の解決の一助になる技術の創出を目指している．

一方，ムーア則終焉直後には，ユーザ側に十分なソフトウェア的な対応をさせることは困難で，多くのアプリケーションの全面的な書き換え等を抑制した形で，終焉の影響を軽減させつつ性能向上を実現することが望まれる．

本研究はそのようなニーズに対応する方策を考察する．本稿ではその方策として有望と思われる想定アーキテクチャを考察し，McIM(Massive cores in Memory side)というコンセプトやそれに付随するいくつかの補助的なアイディアを提案する．さらに，McIM を有効活用する例として配列間接参照の高速化アルゴリズムも提案する．

2. ムーア則終焉問題と既存の取り組み

本章ではムーア則終焉問題と既存の取り組みの方向性について述べる．

2.1 ムーア則終焉問題

半導体の集積度向上が 1.5 年で 2 倍になるというムーアの法則は既にそのサイクルが鈍っており，近い将来に半導体の集積度向上が停止することが確実視されている．時期には諸説あるものの，2025 年頃には論理回路向けの半導体の集積度が向上しなくなる説が有望視されている．

一方，Flash 等の不揮発メモリに関しては平面方向ではなく積層数方向やセル内のアナログ的状態の多ビット化といった論理回路の集積度とは異なる方向に進化を続けており，ムーア則を超越している状態にある．

2.2 既存の取り組みの方向性

- **FLOPS to BYTES 主義**

  FLOPS to BYTES とは，SC16 のパネルディスカッション[1]等において松岡・理研 R-CCS センター長が

1 東京大学
University of Tokyo

提唱したコンセプトである．ムーア則に強く依存するピーク演算能力ではなく，メモリ容量とバンド幅とアルゴリズムで性能を上げることが重要とする考え方である．

- **カンブリア爆発主義**

  カンブリア爆発とは「多様性」を特徴とする本来生物学の言葉であるが，計算機の世界では「多様な」問題ごとに専用コンピュータを開発し，得意なことは得意なマシンに任せるという考え方を意味する．Deep Learning に特化した TPU[2]のような ASIC ベースでの専用プロセッサが代表例であるが，広い意味では FPGA などを用いたリコンフィギャラブルコンピューティングはこのカテゴリに入る．これらは用途限定な高性能計算機を提供する考え方ではあるが，汎用計算機を構築するための思想ではない．

- **量子コンピュータ至上主義**

  これまでの CPU とは全く別のデバイス技術や動作原理に基盤を置く量子コンピュータの開発競争も激化しており，様々な研究成果が出てきている．しかしながら，最初に D-WAVE[3]として商品化されたアニーリング方式では原理上適用できる問題が限定的であるという問題を抱えており，カンブリア爆発の一構成要素に位置づけられる．つまり，従来型計算機システムの完全な置き換え候補にはならない．

  一方，極めて特殊な課題設定ながら，実在するゲート方式の量子コンピュータによってスーパーコンピュータより大幅に高速な処理があることの実証[4]が Google 社の研究者によって Nature 誌上で発表された．しかしながら，現在の量子コンピュータは CMOS ベースの現在の計算機システムのように洗練された技術ではない．増幅の無い世界でノイズの中から信号を探さねばならない問題の克服や，極低温装置の必要性からの脱却や，メモリやストレージを含む計算機システムとして汎用的な従来システムを完全に置き換えるには少なくとも 10～20 年といった長い期間がかかると予測されている．

3. 対策の設計方針

本研究は大前提として 2028 年頃の汎用高性能計算機を対象とする．本研究が採用する設計方針は以下の通りである．

- **基本的な方向性**

  FLOPS to BYTES 主義，すなわち従来型の計算機システムをベースとして，チップ内のコア数(FLOPS に対応)を増やす戦略が取れない状態で，バンド幅(BYTES に対応)の有効活用などの別の方法で性能向上を実現する方向性に基づく．前述の 3 つの方向性の中では 2028 年頃の汎用高性能に唯一到達しうると考える．

- **実現性の重視**

  ムーア則が終焉するとされる 2025 年までほぼ 5 年しか猶予は無い．このため，あまりあてにならない多くの新デバイス技術に頼るのではなく，比較的保守的かつ商業性も含めて実現性の高いデバイス技術でも効果が出せる選択を重視する．

- **汎用性･ニーズの多様性の重視**

  限定したアプリケーションだけを対象に究極的な効率化を目指す専用化(カンブリア爆発主義)ではなく，多様なユーザに対して，基本的には汎用的な共通のハードウェアでその多様なニーズに応えつつ，現実的な性能を提供することを重視する．アプリケーションによって必要とされる資源(演算，容量，バンド幅，遅延)のポートフォリオが異なるので，必要なアプリには必要な資源を可能な限り融通できるようにすることを目指す．

- **大容量性の重視**

  従来のスーパーコンピュータ設計において，メモリ容量はバンド幅と比べてさほど重視されてこなかった．HPC コミュニティ以外の社会全体へのインパクトを考慮し，本研究はクラウドもターゲットと設定する．今後の社会的ニーズ・アプリケーションのトレンドと，不揮発メモリのムーア則を超越した昨今の進化を踏まえ，メモリ容量が必要なユーザには大きくとれるようにすることを目指す．

- **多数の Fat ノードの提供**

  発見した新素材を効率よく製造することが産業の競争力強化に直結するため，パラメータサーベイを行って製造プロセスの最適化[a]を図るアレイジョブのニーズが高まってくると予測している．従来，メモリフットプリント全体のインメモリ化を確保するために多数の小メモリノードを用いていた実用アプリケーションに対して，低遅延な位置に集約された多数の汎用コアや不揮発メモリの大容量性の恩恵を活用した Fat ノードを提供することで，単一ノード上での OpenMP 並列化だけでも多くの実用規模の実行ができる状況を目指す．

- **ユーザへの負担の抑制**

  従来型の計算機システムをベースとすることにより，汎用性を確保するとともに，ユーザのプログラミングモデルに激変が起きないよう，言語処理系やライブラリ等のシステムソフトウェアレベルで極力ハードウェアの違いや最適化の手間を吸収し，使い易い状況を維持することを目指す．

a) 青色発光ダイオードで著名なノーベル賞学者の天野教授の講演において，この点が強調されていた．

4. 着目するアプリケーション

本章では上記の方針に基づき本研究において着目するアプリケーションに関して述べる．FLOPS to BYTES 主義に基づき，かつ社会的ニーズも高いアプリケーションの代表例は，疎行列を扱うアプリケーションである．

疎行列を扱うアプリケーションには配列の間接参照が欠かせないが，現在の計算機のキャッシュアーキテクチャとは非常に相性が悪い．よって，疎行列処理を伴うアプリケーションの実行時にはピーク性能の 1%程度しか出ないのが普通である．例えば，スーパーコンピュータの性能競争において用いられる指標の一つである HPCG ベンチマーク[5][6]は通常ピーク性能の 1%程度しか出ない．つまり，実効メモリバンド幅を上げることでこれまで遊んでいた 99%の演算器のいくらかを稼働状態に移行することが可能になり，ムーア則終焉により演算器を増設できなくても性能を向上できる余地が大きい．

非常に多くの HPC アプリケーションが間接参照を伴う．疎行列を係数行列とする連立一次方程式求解が最も典型的であるが，粒子法のコードの中にもカットオフ距離内にある粒子を選別する過程などで間接参照が現れる．

一方，ビッグデータ解析の一種であるグラフ解析も疎行列処理であり，係数行列の非零要素配置のランダム性は HPC 系より高いためキャッシュとの相性はさらに悪い．

本研究では少なくとも上述のアプリケーションにおいて大きな改善が得られることを目標に，アーキテクチャやアルゴリズムの工夫を試みる．

5. McIM の提案

本章では前述の問題の対策として，上記の方針に基づき筆者らが提案する McIM (Massive Cores in Memory side)に関して述べる．

5.1 McIM の基本概念

McIM は以下の三項目を中核とする高性能汎用計算機の設計思想である．

- **big.LITTLE 技術をシステムレベルに拡張**
  本来の big.LITTLE[7][8]技術は ARM のモバイル用 CPU チップの設計思想である．これは周波数や電源電圧や Out-of-Order 機構の有無の違いなどに起因する特性が異なる 2 種類以上の共通 ISA に準拠したコアを適宜使い分ける．この考え方をシステムレベルに拡張して，多様なニーズに対し最適な電力性能を提供する．
- **コア増設場所はメモリ側**
  電力重視の LITTLE コアをメモリ側のコントローラ内に多数配置する．ムーア則終焉に伴いパッケージ内に収容できないコアの新しい置き場所としてメモリ側のコントローラを利用することで，In memory computing や Near data processing を促進する．
- **全てのコアは OS 稼働可能な CPU**
  Processing in Memory（PIM)の提案は配線層数が少ないメモリ用半導体プロセス上で単純な演算器とその最低限の不随回路を付加するものであることが多い．一方，McIM ではロジック用半導体プロセス上で LITTLE コアも OS を稼働可能な CPU を集積し，ノード内に同時に共存する複数のユーザがコア単位で予約確保可能とする．

5.2 McIM の実施形態

McIM の基本概念により以下に示すような実施形態が想定できる．

(1) 想定されるコア数

例えば 2016 年に出荷された Xeon Phi KNL は 72 個の In order コア(ハードスレッド数 4)を集積して 8 個の MCDRAM と合わせて 245W であった．Xeon Phi KNL の 1 コアは 4 個の LITTLE コアと同等レベルと考える．2025 年頃までは減速しながらも集積度向上が続くため，KNL の 4～8 倍程度のコア数を持つ Xeon Phi が実現可能と考えられる．2028 年にはその 1/4～1/8 程度の個数のコアに当たる 300 コア程度を 1 本の DIMM 上に配置することが可能と予測する．

上記コア数と電力制約は関連性がある．現時点での DIMM スロットの上限電力は 20W であるが，McIM を採用する場合は電源を強化する必要があるかもしれない．しかし，GPU も進化の途中では PCIexpress バス規格の電源上限を大幅に超えることが多かったように，電源の強化は必要ならば可能である．

(2) LITTLE コアの置き場所のバリエーション

McIM の LITTLE コアはメモリ側のコントローラロジックの中に配置される．具体的には以下の 4 つの例が考えられる．

- **Hybrid Memory Cube (HMC)のロジックベース**
  HMC の積層 DRAM を積み上げるロジック部分
- **Active Memory Buffer(AMB)型 DIMM の AMB チップ**
  FB-DIMM の DIMM 上に配置される AMB
- **DIMM 型 Storage class memory(SCM)のコントローラ**
  Intel 社の Optane DC Persistent Memory[9][10]のような SCM を記憶媒体とした DIMM 上に搭載されるコントローラ
- **サーバ用マザーボード上の Buffer on Board (BoB)**
  サーバ上で DIMM の本数を多く設置可能にするためのマザーボード上の Buffer on Board (BoB)

(3) 適用システムごとのバリエーション

McIM を適用するシステムとしては，図 1 および図 2 に示す以下の 2 種類を想定している．ニーズやコストの観点から，適する実装形態に差が出る．

- ハイエンドスーパーコンピュータ
- クラウド

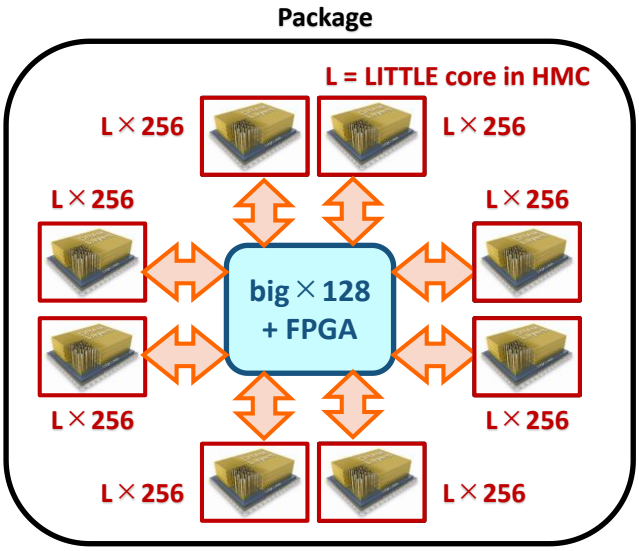

図 1　McIM のスーパーコンピュータへの適用例

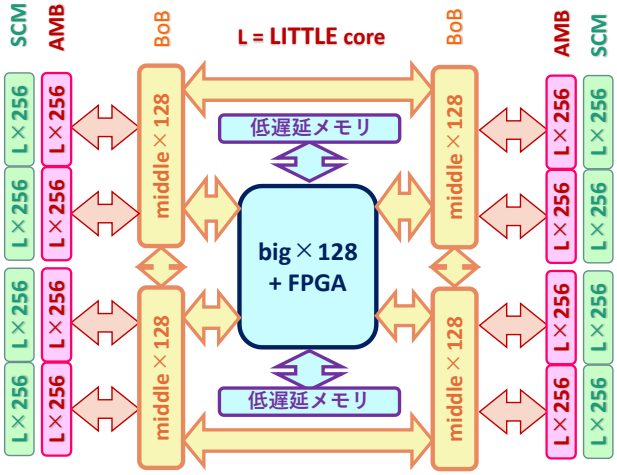

図 2　McIM のクラウドへの適用例

5.3 McIM の基本概念により期待される効果

McIM の基本概念により以下に示すような効果が期待できる．本報告ではカバーしないが，今後に予定される評価においてはこれらが評価項目の候補となる．

- **電力効率の向上**
  McIM は自然に In memory computing や Near data processing となる．これらの手法を取り入れるなどしてメモリ側の LITTLE コアを適切に利用することにより，データの移動による性能低下や電力浪費を抑制できる．さらに，使い切れない資源は積極的に他人に利用させることにより，システム全体として電力を含む資源の有効活用を図ることができる．
- **演算ネックのアプリ向けの FLOPS の向上**
  HPL やニューラルネットの学習などの密行列型アプリケーションの実行時には，多数のメモリモジュール内にある多数の LITTLE コアを確保して浮動小数演算器を稼働状態に設定することにより， ルーフラインモデルの演算器ネック領域の性能限界を引き上げることができる．通常，サーバ型ノードの DIMM スロット数は CPU ソケット当たり 12～24 本程度ある．このため，big コアが搭載される CPU ソケット側のコア数の 1/4～1/8 程度の LITTLE コアが DIMM 上に搭載された場合，2～8 倍程度のノード当たりピーク FLOPS 向上を期待できる．この効果は半導体の進化サイクルの 1～3 回分の延命効果に相当すると考えられる．
- **平均アクセス遅延の短縮**
  サーバの CPU(big コア)の LLC(Last Level Cache)にミスが生じて主記憶の DIMM に対するアクセスが発生する場合，メモリバッファチップを通過するためアクセス遅延が数十 ns のオーダーで大きくなる．一方，McIM においてニアメモリの LITTLE コアに一部の処理をオフロードした場合は，対応するメモリアクセスが LITTLE コアによるものに置き換わるため，DRAM チップが直結されていることから数十 ns のオーダーの遅延が削減される．遅延が CPU の実行性能に与える影響は大きいことが報告[ ]されており，上記による性能向上の効果が期待できる．
- **コア間通信遅延の短縮**
  McIM によればノード内に従来と比べて多くのコアをノード内に実装できる．同じコア数を複数ノードに実装して Infinibannd などのネットワークを用いて結合させるより，小さい遅延時間でコア間のやり取りが可能なため性能がスケールしやすい．メモリアクセスの遅延は Infiniband の通信遅延より概ね 1/10 のオーダーであるため，この効果は大きいと期待できる．共有メモリとして一貫性を維持させる場合においても，維持制御にかかる遅延時間は大幅に低下するため，Infiniband などを用いた分散共有メモリよりも，McIM のようにメモリチャネルを用いた共有メモリの方が性能をスケールしやすいと期待できる．
- **メモリ容量に比例した後付けの処理能力向上**
  サーバ設置後のピーク性能向上は通常困難である．一方処理すべきデータ量が増える場合は実行時間も延びるので，処理能力もそれに応じて向上できることが望ましい．McIM を適用する場合は装着する DIMM の本数(メモリ容量)に比例したピーク演算能力の向上が自然となされる．
- **コア数拡張による実効バンド幅の拡張**
  McIM の LITTLE コアには小容量ながらキャッシュが搭載される．大量の LITTLE コアを予約確保すれば，低遅延高バンド幅なメモリの容量を増量することが

できる．既に冷却面で限外に近い On パッケージに搭載できる高バンド幅 SRAM の容量には限界があるが，それと同様の効果を分散配置によって冷却の問題を回避しつつ，ある程度期待できる．

- **メモリ階層を制御するコアを自然に分散配置可能**
  階層メモリにおいては隣接階層間で局所性を考慮して適切なタイミングで適切なブロックのデータ移動を行うことで実効性能が改善できる．McIM を各メモリ階層に適用することで，上記の移動処理を管理するコアを自然に分散配置することが可能になる．
- **冷却の限界突破と効率化**
  McIM は本来 big のパッケージに入れたかったコア(発熱体)を多数の DIMM 上に分散配置する．この分散化によって，冷却限界に起因するダークシリコン問題(増設コアを同時に稼働できない)を超越している．冷却は発熱体の密度を一か所に集中させると困難度が増すので，冷却コストを削減する効果が期待できる．

5.4 McIM 実装上の補助的アイディア

McIM の基本概念に加えて．実装上の幾つかの補助的なアイディアを組み合わせることにより，望ましい効果を得られる可能性がある．本節ではそのような補助的アイディアの例を示す．

- **一貫性維持プロトコルが通る対称的メモリチャネル**

例えば Intel Xeon は専用チャネル(QPI や UPI)で互いに接続することで CPU 間のキャッシュ一貫性維持を行える CC-NUMA(Cache Coherent Non-Uniform Memory Architecture)を提供できる．プログラミングモデルをユーザフレンドリーなものとするためには，McIM 上でも共有メモリイメージを提供し，キャッシュ一貫性維持を実現できることが望ましい．具体的には DIMM 等のメモリチャネルを QPI 等と等価にも動作させることで仮想的な CC-NUMA のハードウェアイメージを実現することができる．

McIM 上のキャッシュ一貫性維持の効率的実現においては DIMM チャネルが仮想的に QPI 等としても動作する必要がある．現在の DIMM のチャネルはチャネルの片側しかマスタになれないため不便である．McIM が効率的に一貫性維持を行えるようにするためには CPU 側の実装においては HMC のように両側がマスタになれるよう拡張した DIMM チャネルプロトコルを導入することが望ましい．必要に応じて NVDIMM 規格ができたように，将来の DIMM 規格にチャネルの対称性も採用されることを推奨したい．

- **不必要な資源の電源断**

管理の簡略化の観点から，McIM の LITTLE コア内資源はコアの粒度で予約可能とするのが自然である．しかし，利用状況によってはコアが余ったり，アプリケーションによっては一部のコア内資源(例えば浮動小数演算)は使わないこともあり，電力浪費の可能性がある．コアの個数が多くなるので，これらの有休資源については予約管理ソフトウェアの管理下で電源を切断可能とすることで節電の効果が大きいと期待される．

- **Massive Narrow Memory Channel (MNMC)**

McIM はメモリ側の多数のコアによって，DRAM 側に十分なアクセス要求を発生させることが可能である．よってメモリ側もその要求に応えられる高性能なものにしないとバランスが取れず，McIM の能力を十分に活かせない．通常の DIMM はキャッシュラインサイズ(Intel の場合は 64 バイト)ごとに 1 つのアドレスを与えるバーストメモリアクセス要求を効率的に処理できる．ところが，FLOPS to BYTES 主義に基づき着目している疎行列処理などで頻出する不連続アクセス要求を効率的に処理できない．例えば，64 バイトのキャッシュラインに 8 バイトのデータを Gather するためには，DRAM はライン当たり 8 回のアドレスを受け取る必要がある．一方，DIMM 上には多数の DRAM チップが搭載されており，原理的には同時に DRAM チップ数と同数のアドレスを受け取り，DRAM チップごとに連続しないアドレスを並列にアクセスするように制御することも不可能ではない．上記のポリシーで細い(例えば 9bit)幅の DRAM チップを常時 8 バーストで動作させるならば，通常の DIMM の 8 倍のスループットで 64 バイトのキャッシュラインに 8 個の 8 バイトのデータを Gather できる．

- **Scatter/Gather 命令のキャッシュバイパス**

例えば Intel の AVX512 にあるベクトル Gather の命令で空間的局所性が無い 8 個の 8 バイトデータの Gather をすると，異なる 64 バイトのキャッシュラインを 8 回読み取る．前述のメモリシステムに 8 バイト単位の細粒度な多数のメモリアクセス要求を効率よく供給するために，LITTLE コアの SIMD 型ベクトル Scatter/Gather 命令はキャッシュにアクセスするのではなく，DRAM コントローラにメモリアクセス要求(アドレス列に相当する SIMD レジスタ上のインデックスベクトル)またはアドレス変換後の物理アドレスベクトルを直接転送し，そちらで DRAM アクセスを実行させ，Gather の応答はキャッシュに受けるべきである．これによりキャッシュラインの消費や DRAM バンド幅消費を大幅に抑制できる．間接参照のうち空間的局所性があるものは big コア＋大きめのキャッシュ，無いものは LITTLE コア＋メモリコントローラ直結とすみ分けて使えば副作用もほとんどないと考えられる．

6. 配列間接参照の高速化アルゴリズム

FLOPS to BYTES のコンセプトでは演算能力ではなくバンド幅やメモリ容量やアルゴリズムで高速化を実現する．本章では，前述の McIM を用いたシステム上での上記コンセプトの実施例として McIM 上での配列間接参照の高速化アルゴリズムを示す．

6.1 配列間接参照とその課題

配列間接参照は現在のキャッシュベース CPU において

十分な高速化がなされていない典型的な Open problem である．HPC 分野では大規模な疎行列を係数行列とする連立一次方程式求解における明白なボトルネックである．この抜本的解決は多くの HPC ユーザが切望している．近年ではビッグデータ解析の一つであるグラフ処理において同様の課題が問題視されており，高速化に関しては社会的要請が強い．配列間接参照のコード例を図 3 に示す．

```
double A[1000], B[1000], c;
int Index[1000], i;
for(I = 0; i < 1000; i++){
c = c + A[i] * B[Index[i]];
}
```

図 3　配列間接参照を含むコード例

間接参照は配列のインデックスが配列になっている状態を指す．図 3 の例では配列 B のインデックスが配列 Index になっている．

配列 Index の中身は自由に定義できるため，通常連続しない整数が入っている．すると B[Index[i]]と B[Index[i+1]]は連続しない位置にアドレスされるため高い確率で別のキャッシュライン上にマップされる．こうして本来 1000 個の double を読めば済むはずのところを 8000 個分読まねばならず，キャッシュやメモリバンド幅を浪費する．さらに配列 B に関しては過去のアドレス列から将来のアドレス列を予測できないので CPU のハードウェアプリフェッチャは正しいラインをプリフェッチできない．それらの結果，キャッシュミスが雪崩的に発生し，連続アクセスのためプリフェッチが効いてキャッシュに入っていた A や Index の値までキャッシュから追い出される．このような悪循環が起こり，連続アクセスで済んだ場合と比較して圧倒的に性能低下し，電力も浪費してしまうという課題があった．

6.2　提案する配列間接参照アルゴリズムの設計方針

上記のような課題を解決すべく，McIM を用いて配列間接参照の適切なプリフェッチを実現する方法を設計する．その設計方針を以下に示す．

(1) できるだけ多数の DIMM を確保して並列にアクセスすることで，DRAM バンド幅がボトルネックになるのを軽減する．

(2) LITTLE コア上に Helper thread を立ち上げ，DIMM のバンド幅を十分に引き出せる適切な個数の LITTLE コアで並列にアクセスする．

(3) LITTLE コアが他の DIMM を無秩序にアクセスするのを予防して，通信トラフィックや一貫性維持が複雑になり性能低下するのを防ぐ．

(4) 間接参照したデータを用いた演算が必要で，それを LITTLE コアにさせると別の DIMM からのデータ(例えば図 3 の配列 A)の移動が必要な場合，(3)を達成すべく能力が高い big コアに演算を任せる．

(5) 再利用性が高いアクセスは big コアのキャッシュで活用し再利用性ありキャッシュセクタで加速する．

(6) 再利用性が低いアクセスは LITTLE コアで Gather することで，使わないデータを big コアに移動しないようにデータ量を削減する．

(7) big コアは LITTLE コアが Gather したデータを適切なタイミングでプリフェッチし，使うデータだけを自分のキャッシュ上に移動して遅延を隠蔽する．

6.3　提案する配列間接参照アルゴリズム

上記のような設計方針で McIM を用いた配列間接参照アルゴリズムを以下に示す．データの流れの模式図を図 4 に示す．

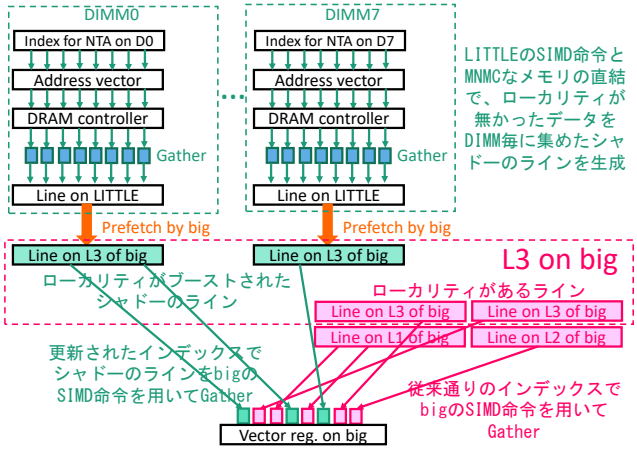

図 4　提案アルゴリズムによるデータの流れ

(1) 前処理部

0. 事前にアドレス→(ノード番号,DIMM 番号)のマッピングを調査、mapping 関数を確定

1. 再利用性が有るデータと無いデータの二つの集合に分類(分類基準は※に後述)

2. 再利用性が無いデータを(ノード番号,DIMM 番号)毎に分類

3. 参加する全 DIMM 毎に LITTLE 側に立てるヘルパースレッド毎の Gather 用 Index 配列を生成、big 側のプリフェッチ用 Index 配列も生成

4. 再利用性が無いデータを LITTLE 側で Gather されるシャドーデータ配列にマップし，big 側にある対応する元 Index 配列がシャドーを指すように更新

(2) 実行部

5. 再利用性が無いデータを 3 の index を用いて LITTLE 側ヘルパースレッドが各 DIMM 上のシャドー配列に Gather を開始

6. 5 が終わっている頃合い(遅延係数 d1)を見計らい big 側が 3 の index を用いて再利用性無し指定プリフェッチを開始

7. 6 が終わっている頃合い(遅延係数 d2)を見計らい big は 4 で更新後の index を元に SIMD 型 Gather 命令で間接ロードし、そのヒット率を計測

8. 7 のヒット率が向上するように、必要に応じて遅延係

数 d1 および d2 を調整

※再利用性に基づく分類基準の例

- 簡易版：疎行列のインデックスである場合、対角位置の付近のアドレスは再利用性が高い可能性が高いと仮定して分類
- 平凡版：Reuse distance を計測し、適当な距離の閾値で二分割
- 推奨版：ラインサイズを組み込んだ変形(ラインサイズ 64 バイトの場合，アドレスの下位 6bit をマスク)を施した Reuse distance を計測し、適当な距離の閾値で二分割
- 精密版：ラインサイズやライン数やリプレースアルゴリズムを組み込んだキャッシュシミュレーションを行い、適当なヒット率の閾値で二分割

7. 予備評価

本報告では提案アルゴリズムの実装や提案アーキテクチャ(McIM や補助的アイディア)の様々な効果に関する評価は行わず，今後の課題とする．本章では FLOPS to BYTES の観点から間接参照によって非常に効率が低下していると考え着目したアプリケーションである HPCG ベンチマークを題材に，手始めに行った潜在的な効率改善の余地の大きさやメモリアクセスの挙動に関する評価を報告する．

評価に用いた環境を以下の**表 1** に示す．

表 1　評価環境

マシン名	東京大学 Reedbush-U
CPU	Intel Xeon E5-2695 v4 (Broadwell) 18core 2.1GHz×2CPU (ピーク性能：1209.6 GFlops)
メモリ	容量 256GB (ピークバンド幅 153.6 GB/sec)
QPI	v1.1×2 本 (ピークバンド幅 38.4GB/s)
コンパイラ	Intel icpc
プロファイラ	Linux perf
アプリケーション	HPCG3.0 Reference code (As is) HPCG3.0 Intel MKL 版 (xhpcg-avx2)
実行時パラメータ	-n192 -t60

7.1 HPCG の現状の性能と効率

HPCG の Reference code(As is)[6]の単一ノード上で観測した最大の GFLOPS 値は 18 スレッド時に 1.05GFLOPS であった．このバージョンはソースコードが存在するものの最適化状況が低く，計算機の専門家ではないスーパーコンピュータの一般ユーザが書いて，最適化ができていない状態のコードを模していると考えられる．よって，必ずしも性能ボトルネックが最適化版と一致しない可能性がある．

一方，Intel MKL 版の HPCG[ ]の単一ノード上の最大 GFLOPS 値は 23 スレッド時に 18.3GFLOPS であった．上記 Reference code(As is)版との性能差が 17.4 倍もあることや，ソースコードを公開していない(Intel 社が秘匿したいだけの価値があると推定できる)点も併せて考えると，最適化のプロによって究極的にチューニングされた状態のプログラムと考えられる．

一方，ハードウェア的にはピーク性能が 1209.6GFLOPS であるため，Intel MKL 版の効率は 1.5%に過ぎない．演算器の FLOPS は 98.5%が余っている状態と言える．よって，演算器を増設しなくても性能低下要因を除去できれば性能向上する余地があり，効率向上の余地はかなり大きい．つまり，FLOPS to BYTES のコンセプトに基づく提案手法によって改善を目指す例題としての適性をサポートする結果と言える．

7.2 HPCG のスケーラビリティ特性

Intel MKL 版の HPCG には実行すると GFLOPS 値を出力するので，その値を用いて Intel MKL 版の HPCG の GFLOPS 値を，スレッド数を変化させつつ計測した．その結果を図 5 に示す．スレッド数 23 において最大値を示すが，その後スレッド数を増やしても徐々に減り，32 以上になると大きく減るという少し複雑な形状を示した．計算環境は 18 コア 2 ソケットなので 18 スレッド(ソケット間の通信路が細かったり，通信が多い場合)または 36 スレッド(ソケット間の通信路が太かったり，通信が少ない場合)でピークを迎えると予想していたがそうはならなかった．23 でピークになる理由は次のメモリアクセス特性に関連して考察する．

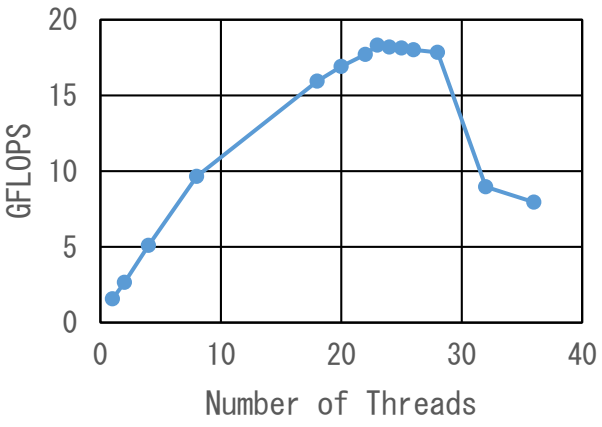

図 6　Intel MKL 版 HPCG のスケーラビリティ特性

7.3 HPCG のメモリアクセス特性

Intel MKL 版の HPCG にはソースコードが付属していないが Linux の perf コマンドはアプリケーションのバイナリだけで解析が可能なので，これを用いて Intel MKL 版の HPCG のメモリアクセス時系列特性を，スレッド数を変化させつつ計測した．

LLC(Last Level Cache)のミスを perf のプリデファインドなイベント cache-misses を計測周期 100ms の設定で計測した結果を図 6 に示す．イベント cache-misses はオンデマンドなミス，つまりプリフェッチに伴うメモリアクセスを除いたキャッシュラインのリプレース回数に相当する．

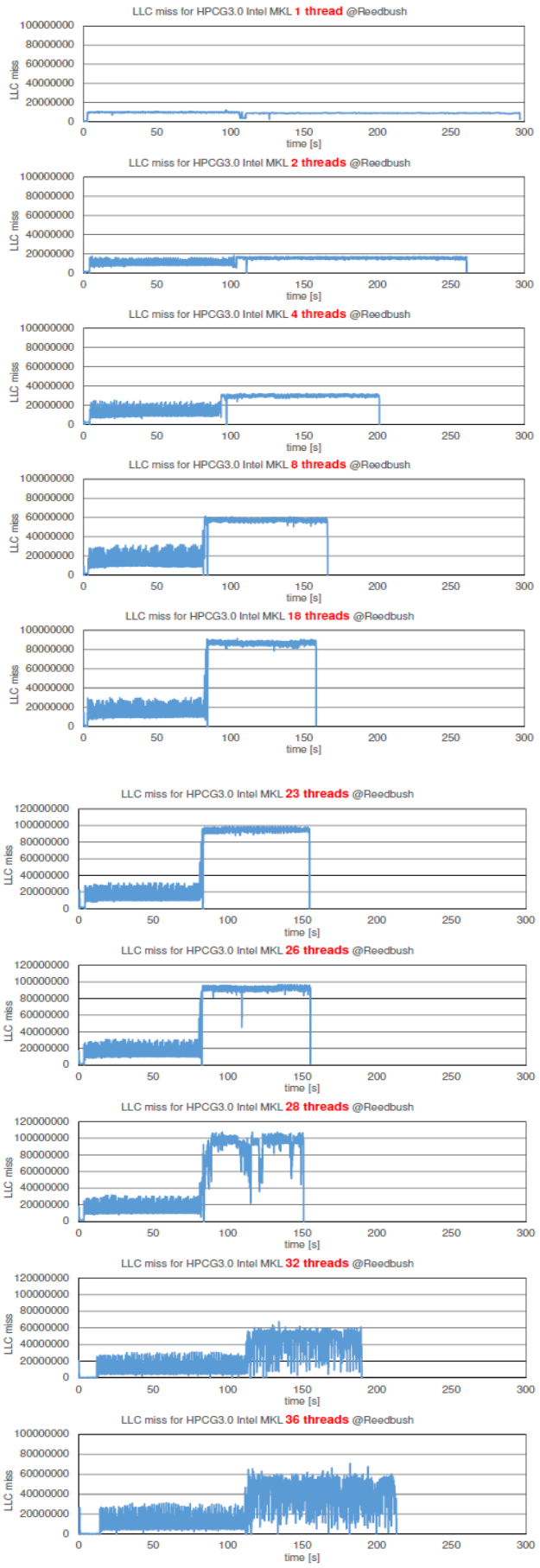

図 6　Intel MKL 版 HPCG のメモリアクセス時系列特性

結果全体において非常に多数の LLC ミスが継続的に発生しており，プリフェッチがうまく機能していないことがわかる．1 回のミスで 64 バイトのラインのライトとリードが 1 回ずつ発生するので，1 周期(100ms)内の LLC ミス数に 128 バイトを乗じ，0.1 秒で割った数字が LLC ミスに伴う要求メモリバンド幅を意味する． 23 スレッド時が最も短時間で処理が終わり，最大 GFLOPS 値がレポートされたが，上記の換算で得られた GFLOPS 値計測期間(グラフの後半の山の部分)の平均要求バンド幅は 120.3GB/s に達した．この値はハードウェアのピークバンド幅 153.6GB/s より少し少ないが，差分はプリフェッチのバンド幅にあたるのか，QPI のバンド幅がこの GFLOPS に対してバランスした状態にあるのかのいずれかと考えられる．18 スレッドを超えると compact 指定でも割り当てられるメモリ領域が 2 ソケットにまたがるため，23 スレッド時には 5 スレッド分が QPI をまたいでアクセスしている状態になる．1CPU 時のコア当たりバンド幅は 153.6GB/s/18 コア＝8.53GB/s/コアであり，QPI2 本で賄えるコア数は 38.4 GB/s/8.53＝4.5 コアとなる．つまり 22.5 コアで QPI 側が溢れることが予想される．スレッド数が 23 を超えると GFLOPS 値計測期間の波形に下側へのスパイクが入るようになり，スレッド数を増やすほど顕著になり，性能低下が酷くなる．スパイクは QPI を溢れさせるアクセスが発生し，結果的にストールが発生してメモリアクセス頻度や演算実行が低下している瞬間を意味していると考えられる．

McIM を用いた間接参照のアルゴリズム設計においては，多くの DIMM を参加させる際に LITTLE コアから別の DIMM をアクセスしない方針で設計しているが，上述の測定結果はこの方針の正しさをサポートしていると考える．McIM は仮想的な QPI をメモリチャネル上に張ることになるので，そこを溢れさせるような使い方は，まさに上述の QPI 溢れを起こして性能低下が起きた状況と対応するためである．

今回の計測は間接参照の際にキャッシュラインには有効データが少ない状態でアクセスしている．一方，McIM を用いた間接参照のアルゴリズムは Gather により有効データしかメモリバンド幅を消費しない．McIM の複数の LITTLE コアがメモリコントローラを溢れさせる程度までリード要求を発生させることができ，そこにメモリコントローラに Massive Narrow Memory Channel (MNMC)を採用した場合，従来の 8 倍の実効バンド幅が出るため，そこはボトルネックにはならないと考えられる．LITTLE コアのキャッシュ上のラインに MNMC が Gather した有効データ密度の濃いデータが，big コアからのプリフェッチ要求で DIMM のメモリチャネルを経由して big コアのキャッシュに転送される．この転送も 1/8 のデータ量になっているのでボトルネックにはならないと考えられる．つまり，DRAM から big のキャッシュまでの経路で一貫して 8 倍のレートで転送が

実行されることが期待されるので，本実験のようにメモリバンド幅やQPIバンド幅で律速されていたアプリケーションは約8倍の処理速度を示すことが期待される．

さらに，本実験のように従来のHPCGの実行状況はオンデマンドなLLCミスが発生して遅延ペナルティを課せられている状態での実行になるが，提案アルゴリズムがうまく機能すればbigコアのプリフェッチが継続的に成功する動作モードになるため,より高い性能を示す可能性がある．

8. 関連研究

間接参照の高速化に関する研究としては，古くは筆者の研究であるDIMM内でのGather回路を有するDIMMnet-2[11]やHMC内のGather回路[12][13]があり，どちらもメモリサイドでのGatherであり，本研究のルーツとなる研究である．これらはDMA的回路を想定しているが，本報告の案はメモリ側のマルチコアのSIMD型Gather命令をベースとしており,メモリコントローラへの直結で並列に不連続アクセスが実行される点でより高いスループットが期待できる．最近では早大によるマルチコアのコア内のGather回路[14]を設ける研究があるが，こちらはCPUサイドのGatherという点が異なる．上記は細粒度なDRAMアクセスが発生するが，早大案のように通常のDIMMを用いていると絶対性能が確保できない．本報告で提案しているMassive Narrow Memory Channel (MNMC)のようなものとの組み合わせが必要になる．一方，富岳用CPUの設計においても間接参照の高速化の取り組みがされており，京には無かったCombined Gather[15][16]の機能が追加された．L1の128Bのブロック内からレジスタへの間接ロードのバンド幅を倍増している．このような演算器に近い側の改良は真のボトルネック(主記憶の細粒度なアクセスバンド幅)とは違う部分での改良になるので，メモリバンド幅が京のCPUの16倍に向上した効果に比べて，この機構単体の効果の寄与分は薄いと考えられる．

9. おわりに

本報告ではムーア則が終焉後，量子コンピュータ等が十分に洗練して従来型計算機システムを置き換えるまでの期間に性能向上を果たすための技術として，McIM(Massive cores in Memory side)というアーキテクチャを提案した．McIMの基本コンセプトによる8つの観点からの予想される効果についても論じた．共有メモリ環境の提供や不連続アクセスの効率化を行うためのMcIMに付随するいくつかの補助的なアイディアも提案した．

さらに，McIMを有効活用する例として配列間接参照の高速化アルゴリズムも提案した．予備評価として，配列間接参照に課題があるとされている現状のHPCGを計測し，演算器を増設しなくても得られる潜在的な性能改善幅の大きさや，間接参照性能改善の重要性および提案アルゴリズムの方針の正しさをサポートする実験結果を得た．

今後は提案アルゴリズムの試験的実装や，提案アーキテクチャ(McIMや補助的アイディア)の様々な効果に関する評価を進める予定である．

謝辞 この成果は，国立研究開発法人新エネルギー・産業技術総合開発機構（ＮＥＤＯ）の委託業務の結果得られたものである．

参考文献

[1] Hisa Ando. “SC16 - スパコンはポストムーアの時代をどう乗り切るのか？”, https://news.mynavi.jp/article/20161219-sc16_pmes/, (参照 2019-11-17).
[2] Norman P. Jouppi et al. (86 people) In-Datacenter Performance Analysis of a Tensor Processing Unit, Proceedings of the 44th Annual International Symposium on Computer Architecture(ISCA'17), 2017.
[3] D-WAVE 日本. http://dwavejapan.com/, (参照 2019-11-17).
[4] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. Nature No.574, p.505-510, 2019
[5] “HPCG Benchmark”, http://www.hpcg-benchmark.org/, (参照 2019-11-17).
[6] “hpcg-benchmark/hpcg”, https://github.com/hpcg-benchmark/hpcg//, (参照 2019-11-17).
[7] “ARM big.LITTLE”. https://www.arm.com/why-arm/technologies/big-little, (参照 2019-11-17).
[8] “big.LITTLE Technology: The Future of Mobile”. https://www.arm.com/files/pdf/big_LITTLE_Technology_the_Future_of_Mobile.pdf, (参照 2019-11-17).
[9] Intel Corp. “インテル® Optane™ DC Persistent Memory”, https://www.intel.co.jp/content/www/jp/ja/architecture-and-technology/optane-dc-persistent-memory.html, (参照 2019-11-17).
[10] Intel Corp. Intel® Optane™ DC Persistent Memory Product Brief,https://www.intel.co.jp/content/dam/www/public/us/en/documents/product-briefs/optane-dc-persistent-memory-brief.pdf, (参照 2019-11-17).
[11] 田邊昇, 安藤宏, 箱崎博孝, 土肥康孝, 中條拓伯, 天野英晴. プリフェッチ機能を有するメモリモジュールによるPC上での間接参照の高速化, 情報処理学会論文誌コンピューティングシステム(ACS) , 2005, Vol.46, No.SIG12(ACS11), p.1-12.
[12] Noboru Tanabe, Boonyasitpichai Nuttapon, Hironori Nakajo, Yuka Ogawa, Junko Kogou, Masami Takata, Kazuki Joe. A memory accelerator with gather functions for bandwidth-bound irregular applications, Proceedings of the 1st Workshop on Irregular Applications: Architectures and Algorithms. 2011.
[13] 田邊昇, 堀喬裕, Boonyasitpichai Nuttapon, 中條拓伯. Gather機能を有するHybrid Memory CubeのFPGAを用いた予備評価, 研究報告ハイパフォーマンスコンピューティング（HPC) , 2012-HPC-133, No.6, p.1-10
[14] 柏俣智哉, 北村俊明, 木村啓二, 笠原博徳. DMAのカスケード接続による間接ロードの高速化, 研究報告システム・アーキテクチャ（ARC）,2019-ARC-234, No.16, p.1-6
[15] Hisa Ando. SC18 - 全容が見えてきた富士通のポスト「京」スパコン. https://news.mynavi.jp/article/20181207-737074/, (参照 2019-11-17).
[16] 新庄直樹. スーパーコンピュータ「富岳」の開発. SS研HPCフォーラム, 2019, http://www.ssken.gr.jp/MAINSITE/event/2019/20190820-hpcf/lecture-04/20190820_HPCF_shinjo.pdf, (参照 2019-11-17).