Vol.2020-HPC-173 No.5
2020/3/16

将来システムのコデザインのためのCPUシミュレータによるMPIリプレイ環境および性能推定手法の検討

辻 美和子[1,a)]　佐藤 三久[1]

概要：将来システムのアプリケーションにおける性能推定のためにはしばしば CPU シミュレータが用いられる．HPC アプリケーションの多くは MPI などにより記述された並列アプリケーションであるが，膨大な計算量を必要とする CPU シミュレータを複数実行し，並列アプリケーションをシミュレートすることは現実的ではない．本研究では，CPU シミュレータ上で並列アプリケーションの演算性能を大まかに推定するために，逐次実行時に並列実行時のアプリケーションの振る舞いを再現するためのツールである MPI アプリケーションリプレイライブラリを用いることを提案する．また，実験によりオーバーヘッドなどの評価を行った．MPI リプレイライブラリは，受信バッファに受信データではなくさきに取得して置いたファイルから読み込んだデータを格納するため，データ移動のふるまいが変化し，とくにアプリケーションの必要メモリ量とキャッシュサイズが等しいときは，CPU シミュレータにより推定される実行時間に変化がみられた．一方，キャッシュサイズが十分な場合や，まったく足りない場合には，比較的良好な推定が可能であった．

1. はじめに

コンピュータシミュレーションは現代の科学や産業において欠かすことのできない技術であり，複雑化・高度化するシミュレーションソフトウエアは，より高い計算性能の HPC システムを必要とする．さらに，近年需要が高まりつつある人工知能，機械学習，およびソサエティ 5.0 アプリケーションにおいても，同様に高い計算性能が要求されると考えられる．一方で，従来から言われているように，微細化技術による半導体の性能向上は限界に近付きつつある．そのようななかで，科学的成果の早期創出のためにはアプリケーションとアーキテクチャの協調設計であるコデザインは欠かすことができない [1]．代表的なコデザインのアプローチのひとつは，いろいろなハードウエアのパラメータをさまざまに変化させながらシステムシミュレータを用いてアプリケーションの性能を推定し，電力などの制限のもとでアプリケーションの性能を最大化するシステムを探索することである [5]．

とくに CPU シミュレータなどをもちいたノード単体での演算性能評価は，大規模実行時の通信性能評価とともに，性能推定において大きな比重を占める．CPU シミュレータ上での演算性能予測とアプリケーションチューニングは，システムインストール後に速やかな成果創出にも重要な役割を果たす．しかし，多くの HPC アプリケーションは並列実行を想定して MPI などの通信インターフェース・ライブラリを用いて記述されており，一般的にそのままでは CPU シミュレータ上では動作しないため，ノード単体での演算性能推定のためにはソースコードの書き換えが必要になる．アプリケーションによっては他のノードの実行結果に依存して別のノードでの実行パスが変化するものもあり，単純にプロセス数を 1 に設定しても並列実行時の挙動が再現できないものもあるため，大規模実行時を想定した単体ノードで実行可能なカーネルの切り出しにはしばしば高い労力や，ターゲットアプリケーションに対する知識が必要となる．

著者らは，MPI で記述された並列アプリケーションを，ソースコードを変更することなく，ノード単体性能評価時に並列実行時の特定のランクでの振る舞いを再現できるようにするために，ノード単体性能評価のための MPI リプレイ環境を作成した [6]．MPI リプレイ環境は，現在利用可能な大規模実システムを用いてアプリケーションを並列に実行し，通信プロファイルとメッセージバッファを取得し，次にプロトタイプなどの将来 CPU 上でこれらを読み込みながらアプリケーションのリプレイを行って単体ノード演算性能評価を可能とするものである．

本研究では，単体ノードリプレイをプロトタイプ CPU

[1] 理化学研究所計算科学研究センター
RIKEN Center for Computational Science
[a)] miwako.tsuji@riken.jp

(実 CPU) ではなく，CPU シミュレータ上で行うことを提案し，オーバーヘッドや評価精度について検討する．本稿は以下のように構成される：まず，2 章で，本研究の背景となる MPI リプレイ環境について述べる．続いて，3 章で，CPU シミュレータ上での MPI リプレイ環境について述べ，4 章で実験と性能評価を行う．最後に，5 章で結論を述べる．

2. 研究背景 MPI リプレイ環境

本節は，ノード単体性能評価のための MPI リプレイ環境 [6] について概要を述べる．並列実行を想定したアプリケーションは，単純にプロセス数を 1 に設定するだけでは動作しなかったり，動作したとしてもプログラムの挙動が並列実行時とは大きく変化してしまうことがある．そこで，MPI リプレイ環境として，MPI アプリケーションの任意のプロセスランクを単体ノードで再現するためのライブラリセットを開発した．

MPI リプレイ環境は Sandia National Laboratories で開発されている HPC システムのコデザインのためのツールキットである Structural Simulation Toolkit (SST)[3] を拡張して開発された．SST キットに含まれる SST/macro は，分散システムのためのネットワークシミュレータであり MPI アプリケーションの通信性能をオンライン・オフライン 2 つの手法で推定することができる．オフラインシミュレーションは，実機で取得した通信ログに基づいてネットワークシミュレーションを行う．

MPI リプレイ環境では，実機での実行時に通信ログに追加して通信バッファを取得し，単体ノードリプレイ時には取得した通信メッセージを読み込みながらリプレイを行う．これにより，例えばランク 0 が `MPI_Bcast` した初期パラメータを用いて計算を行うランク 1 の振る舞いがランク 1 のみでリプレイすることができる．

図 1 に，実環境におけるトレースおよびメッセージバッファの取得 (左) とノード単体におけるリプレイ (右) を示す．まず，実環境におけるトレースおよびメッセージバッファの取得時には，アプリケーションにおける各プロセスは，MPI 関数呼出しごとに，実際に MPI 通信を行うとともに，MPI の送信先やバッファサイズなどの引数を記録し，受送信バッファ内容をストレージなどに書き込む．このために，通信にログ取得のための機能を追加した MPI 関数が，OpenMPI や MVAPICH などのデフォルトの MPI 関数に上書きして用いられる．次に，ノード単体環境においては，MPI 関数呼出ごとに，実際には通信を行わず，必要な引数や受信バッファの内容を，ストレージから読み込み，アプリケーションリプレイを実行する．そのため，リプレイされるプロセスは，受信バッファの内容でプログラムの振る舞いが分岐する場合なども，並列実行時とまったく同様の挙動をとることができる．

図 2 に，トレース・バッファ取得用，およびリプレイ用のバイナリの作成方法を示す．ソースコードの書き換えは不要で，リンクするライブラリを変更するだけで，トレース・バッファ取得用，およびリプレイ用のバイナリが作成できる．MPI 関数の置き換えは，一般的な MPI の実装が MPI 関数を上書き可能なウィークシンボルとして実装していることを利用して行われる．一般的な MPI 実装においては，置き換え可能な MPI 関数の内部で，実際に通信を行う PMPI 関数が呼び出される．PMPI 関数の前後に記録のためのコードを追加した同名の MPI 関数を実装し，アプリケーションにリンクすることで，並列実行時の通信ログおよびメッセージバッファの取得を行う．同様に，PMPI 関数を呼び出さず，ストレージから通信バッファを読み込む機能を追加した MPI 関数を含むリプレイ用のライブラリをアプリケーションにリンクすることで，単体ノードにおけるリプレイが可能となる．

3. CPU シミュレータ上での実行

著者らの以前の研究報告 [6] では，MPI リプレイライブラリを用いたノード単体性能評価について，オーバーヘッドや推定精度を実ノードを用いて評価した．これは，新たな大規模実システムがインストール中に，少数のプロトタイプ CPU を用いたチューニングや性能評価が可能である，ことを想定した評価であった．本稿では，より早期におけるノード単体性能評価やチューニング，さらにはプロセッサ設計にノード単体性能評価を用いることを想定し，CPU シミュレータ上でのノード単体性能評価について検討する．また，詳細な性能情報を取得可能な CPU シミュレータにより，通信バッファをファイルからの読み込みに置き換えた場合に起こり得る演算性能や推定実行時間の変化について議論する．

3.1 gem5 CPU シミュレータ

本稿では，CPU シミュレータとして，計算システムのアーキテクチャ研究のためのモジュラプラットフォームである gem5 シミュレータを用いる [2]．gem5 は，OS そのものをシミュレートするフルシステム (FS) モードと，システムコールについてはエミュレーションするシステムコールエミュレーション (SE) モードがある．本稿の目的は，アプリケーションのノード単体上での演算性能を評価することであるため，SE モードを用いる．

3.2 gem5 シミュレータ向けの評価用バイナリの作成

単体ノードリプレイを，実ノードではなくシミュレータで行う場合においても，リプレイ用のバイナリを作成方法は実ノードの場合とほぼ同様である．ただし，gem5 シミュレータではダイナミックライブラリが使用できないため，すべてのライブラリが静的にリンクされている必要が

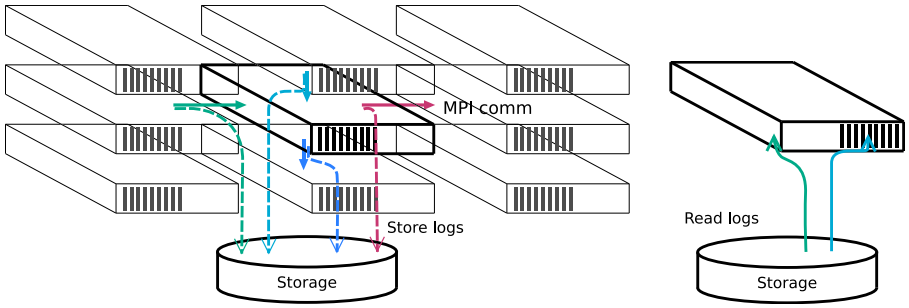

図 1　実環境におけるトレース・メッセージバッファの取得 (左) とノード単体におけるリプレイ (右)

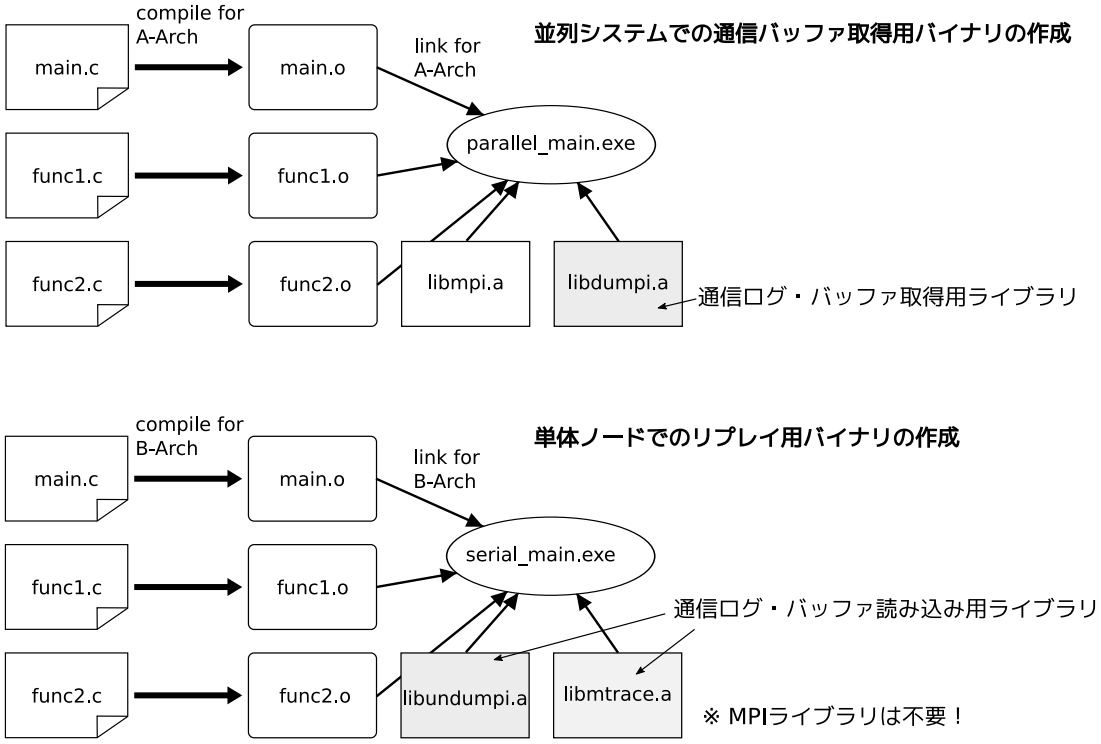

図 2　トレース・バッファ取得用，およびリプレイ用のバイナリの作成

ある．

4. 実験

以前の研究 [6] においては，単体ノードリプレイ時の正確さやバッファの読み込みのオーバーヘッドについて検討した．本研究では，シミュレータ上での実行における読み込みオーバーヘッド，およびシミュレータが提供するパフォーマンス情報を用いた，通信バッファをファイル読み込みに置き換えることによる性能推定結果の変化などに焦点をあてる．

4.1 実験環境

本実験では，以下の 5 つの実行環境について，（推定）実行時間について議論する：

表 1　The specification of the Oakforest PACS

CPU	Intel Xeon Phi 7250 (KNL), 68 core, 1.4 GHz
Memory	96GB(DDR) + 16GB(MCDRAM)
Network	Intel Omni-Path Network, 100 Gpbs
Compiler	intel/2018.1.163
Option(N)	-O3 -xMIC-AVX512
Option(T)	-O3 -xMIC-AVX512
MPI library	impi/2018.1.163
OS	CentOS 7

N	Normal	トレース取得を行わない実機での並列実行
T	Trace	トレース取得を行う実機での並列実行
R	Relay	単体ノードでの単体リプレイ
G	Gem5	Gem5 での単体リプレイ
S	Serial	マニュアルで逐次化したバイナリの Gem5 実行

並列実環境としては，東京大学と筑波大学が共同して運

表 2　The specification of 単体ノード

CPU	Intel Gold 6146, 16 core, 3.2GHz
Memory	32GB DDR4
Network	-
Storage	NFS
Compiler	intel/2018.1.163
Option (R)	-O3 -xCORE-AVX512
Option (G)	-O3
Option (S)	-O3
OS	CentOS 7.5

表 3　The specification of gem5

Version	gem5.opt 2.0
CPU	1.0GHz
Memory	8GB

営する Oakforest-PACS (OFP) を用いた．OFP の諸元を**表 1** に示す．単体ノード環境および gem5 シミュレータ実行環境としては，**表 2** に示す計算機を用いた．すべてのコンパイルに，OFP にインストールされているインテルコンパイラ群を用いた．gem5 シミュレータ (**表 3**) は X86 向けのデフォルトのパラメータを用いた．またシミュレーションはシステムコールについてはエミュレーションするシステムコールエミュレーション (SE) モードで行われた．

4.2　テスト問題

本稿では例題として，単純化したステンシル計算 および Nas Parallel Benchmarks (NPB) [4] を考えた．

ステンシル計算では以下に示すように，$N \times N$ の二次元のデータをプロセスに分散し，各プロセスは深さ 1 の袖領域を上下左右のプロセスと交換しながら計算を進める．境界条件は周期的境界条件とする．メモリ的に連続な袖領域はそのまま送り，ストライドアクセスとなる袖領域についてはいったん別のバッファにコピーして連続にした上で送信するものとした．

```
n = N/sqrt(nprocs);

for(iter=0; iter<niter; iter++){
  memcpy(&(b[0][0]), &(a[0][0]),
                  (n+2)*(n+2)*sizeof(double));
  // exchange halo
  send_halo(a, n, myrank, nprocs, uprank,
                      dwrank, lerank, rirank);
  // calc
  for(i=1; i<n; i++){
    for(j=1; j<n; j++){
      b[i][j] += (c0*a[i][j-1] + c1*a[i][j+1]
              + c2*a[i-1][j] + c3*a[i+1][j]);
    }}
  diff=0.0;
  for(i=1; i<n; i++){
    for(j=1; j<n; j++){
      diff +=  (b[i][j] - a[i][j])
                          * (b[i][j] - a[i][j]);
    }}
  MPI_Allreduce(&diff, &diff2, 1, MPI_DOUBLE,
                       MPI_SUM, MPI_COMM_WORLD);
}
```

4.3　実験結果:ステンシル計算

MPI 通信をファイル読み込みに置き換えた場合にシミュレーションされた推定演算時間の正確さについて調査するために，上述のステンシルベンチマークを用いて，Gem5 で通信バッファファイルを読み込みながら単体リプレイする場合と，Gem5 上で MPI 関数を同名の中身が空の関数で置き換えた場合とを比較した．ステンシルベンチマークの場合，受信バッファの内容が後の計算の振る舞いに影響を与えることはないため，単純に MPI 関数を空関数で置き換えることにより，逐次化を行った．ただし，MPI_Comm_size で取得していたプロセス数についてはハードコーディングで与えた．

全体の問題サイズ $N = 1024, 1200$ として $4 \times 4 = 16$ プロセスに分散して実行するケースを想定した．$N = 1024$ のとき各プロセスは double 型の $(1024 \times 1024)/(4 \times 4)$ のブロックを `a[][]`，`b[][]` の 2 つ分保持する．よって，1 プロセスあたりのメモリ量は，おおよそ

$$8 \times 2 \times (1024 \times 1024)/(4 \times 4) = 1\text{MB}$$

である．同様に，$N = 1200$ のときは，おおよそ 1.37MB である．

本実験では，MPI 通信をファイル読み込みに置き換えた場合のキャッシュまわりにおける挙動の変化によって推定時間がどう影響をうけるかを調査するため，gem5 シミュレータの入力パラメータを，キャッシュがほぼゼロの場合 (L2=1KB)，ぎりぎりの場合 (L2=1MB)，十分なキャッシュがある場合 (L2=2MB, 4MB) の 3 種類に変化させ，MPI 通信をファイル読み込みに置き換えた場合と，MPI 関数を何も行わない空の関数に置き換えた場合のシミュレーションを行った．本実験では，コンパイラによって必要以上に最適化されて，演算が行われないコードを生成するのを防ぐために，コンパイルオプションは `-O0` とした．gem5 シミュレータの入力パラメータは以下である：

```
./gem5/build/X86/gem5.opt \
./gem5/configs/example/se.py \
--mem-size=8192MB  --cpu-type=DerivO3CPU --caches \
```

```
--l2cache --l2_size=<size> -c ./a.out
```

図 3–4 に，$N = 1024$ のときの MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の推定実行時間とその比率を示す．図から，どのキャッシュサイズの場合も，MPI 通信をファイル読み込みに置き換えた場合は，MPI 通信を単純に省略した場合と比較して，推定実行時間が長い．これはファイル読み込みそのもののオーバヘッドが乗っているからであると考えられる．ファイル読み込みのキャッシュへの影響が排除できると考えられる場合，すなわちキャッシュ容量が極端に少ない場合 (1KB) と，容量に余裕がある場合 (2MB, 4MB) と比較して，キャッシュ容量がぎりぎりの場合 (1MB) では推定された実行時間の差が大きい．1KB,2MB,4MB では，2 ～ 3 パーセントの差であったのに対して，1MB では 14 パーセント程度の差が生じた．これは，通信をファイル読み込みに置き換えることにで，性能推定結果に 10 パーセント以上の変化が生じてしまうことを示す．

図 5–6 に，$N = 1200$ のときの MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の推定実行時間とその比率を示す．このとき 1 プロセスあたりのメモリ容量は 1.34MB であり，本実験で考慮した gem5 のキャッシュ容量にはクリティカルな容量は存在しないため，推定された実行時間はファイル読み込みの有無に係わらず，数パーセントの差に収まった．

図 7–8 に，MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) のシミュレーションに要する時間 (ホスト時間) について示す．gem5 シミュレータにおける SE モードでは，ファイル IO のシミュレーションは行わないため，MPI 通信をファイル読み込みに置き換えた場合でも，ホスト時間が増加することはなく，同様の所要時間でシミュレーションを行うことができる．

図 9 – 14 に，gem5 シミュレータの出力する統計情報から，N=1024 で MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の，L1D ミス，L2 ミス，L2 ミス・レイテンシおよびその比率を示す．図 3–4 で示した実行時間と同様に，プロセスの必要とするメモリ量が L2 キャッシュ容量に近いとき (L2=1M)，MPI 通信をファイル読み込みに置き換えた場合と省略した場合で，L2 キャッシュミス数およびそれに起因するレイテンシが大きく変化している．なお，gem5 のデフォルトのレイテンシ情報は ticks 単位で出力されるため，1000 で割った数がサイクル数である．

容量を変化させていない L1D ミス数や，L2 サイズが十分に大きいか小さいとき，MPI 通信の置き換え方法によって，統計情報が大きく変化することはなかった．

4.4 実験結果:NPB

次に，Nas Parallel Benchmarks (NPB) を用いて，シミュレータの実行時間を MPI 通信をファイル読み込みに置き換えた場合のオーバーヘッドを評価した．

表 4 NPBs, Class and the number of processes

App	Class	Nprocs
BT	A	64
CG	A	8
EP	A	8
FT	A	8
IS	B	8
LU	A	16
MG	B	16
SP	A	64

表 4 に，NPB の各ベンチマークのクラスとプロセス数を示す．

図 15–16 に，NPB の各ベンチマークのトレース取得を行わない実機での並列実行 (N)，トレース取得を行う実機での並列実行 (T)，単体ノードでの単体リプレイ (R)，gem5 (L2=2MB) での単体リプレイ (G) および，gem5 シミュレータそのものの実行時間 (gem5-host) および，N を 1 としたときとの比率を示す．なお，図 1–3 で示したように，これらの実験はまったく性能の異なる CPU 上で実行されており，シミュレータや実ノードリプレイの正確さを評価するものではない．本表の目的は，コデザイン時の使い易さの観点から，シミュレータや単体実ノードを用いたリプレイがどの程度の時間を要するものかを一次近似的に示すことである．表からキャッシュの挙動までシミュレートする場合には実際の実行と比較して $O(10^5)$ の時間がかかっており，1 秒程度のアプリケーションのシミュレーションに数時間を要することがわかる．ただし，前述のステンシルアプリケーションの実行と同様に，MPI 通信をファイルの読み込みに置き換えたことによるオーバーヘッドはごくわずかであった．

ステンシルアプリケーションでの実験により，MPI リプレイ環境による性能推定は，各プロセスのデータサイズと想定するアーキテクチャのキャッシュサイズが近い値であるとき，精度が下がる懸念があることが示唆された．そこで，NPB の各アプリケーションについて L2 キャッシュサイズのパラメータを変化されたときの推定実行時間を調査した．**図 17** に，実験結果を示す．BT，LU，SP などのようにキャッシュサイズの増加に対して推定実行時間が短調に減少しているアプリケーションや，EP，FT，IS，MG のようにキャッシュサイズが極端に小さい場合以外は推定実行時間が安定しているアプリケーションについては，プロセスあたりのデータサイズが想定キャッシュサイズとは大きく異なると考えられ，MPI リプレイ環境と CPU シミュレータによる推定精度が低下する可能性は低いと考えられる．CG の場合のみ，想定キャッシュサイズが 2 ～ 4MB の間で大きく性能が変化したことから，MPI リプレイ環境

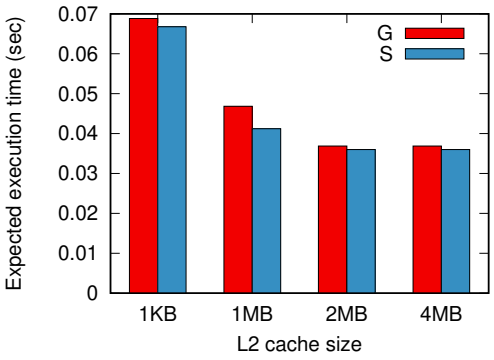

図 3 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の推定実行時間

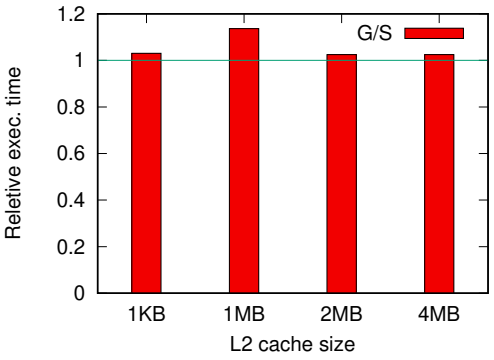

図 4 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の推定実行時間比率 (G/S)

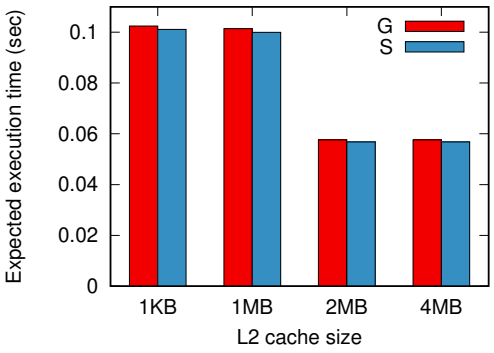

図 5 N=1200,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の推定実行時間

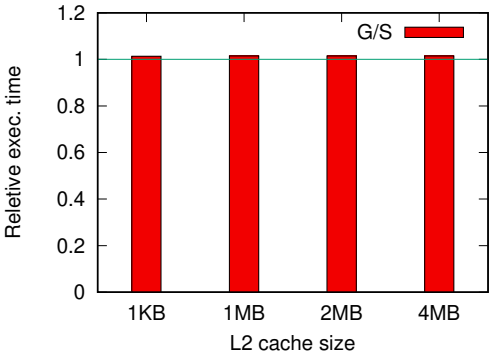

図 6 N=1200,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の推定実行時間比率 (G/S)

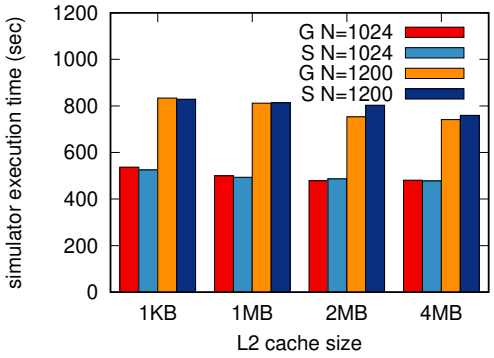

図 7 MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) のシミュレーション時間時間

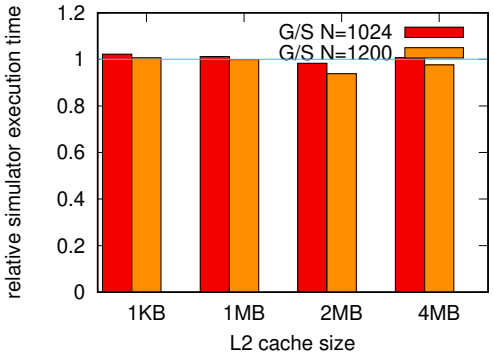

図 8 MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) のシミュレーション時間比率 (G/S)

と CPU シミュレータによる推定精度が低下している懸念があると考えられる．当然ながら，プロセスが必要とするデータサイズは問題サイズとプロセス数によって変化するため，本実験における必要データサイズが一般的なものであるとはいいきれないが，本実験の結果から，複数のアプリケーションを複数のデータセットで横断的に調査した場合，精度の低下はおそらくその一部にしか見られないと考えられる．なお，表 4 に示した本実験におけるデータサイズとプロセス数は，各アプリケーションが 1 ～ 数秒で終了するように設定した．

5. おわりに

本稿では，並列アプリケーションの単体ノード上での性能評価のための MPI リプレイ環境を，実単体ノードではなく，CPU シミュレータ上で利用することを提案し，その際のオーバーヘッドや正確さについて検証した．これにより，シミュレータ専用のコードをマニュアルで作成することなく，さまざまな並列アプリケーションがシミュレータ上で実行することが可能となる．

単純なステンシルアプリケーションによる検討の結果，

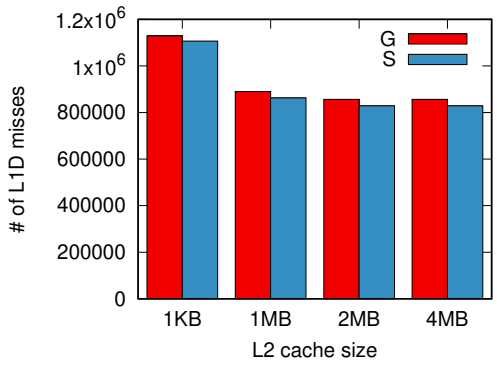

図 9 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の L1D ミス数

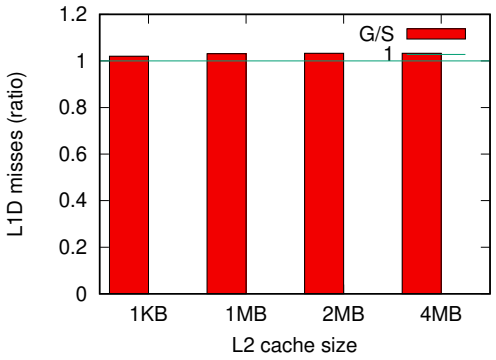

図 10 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の L1D ミス数比率 (G/S)

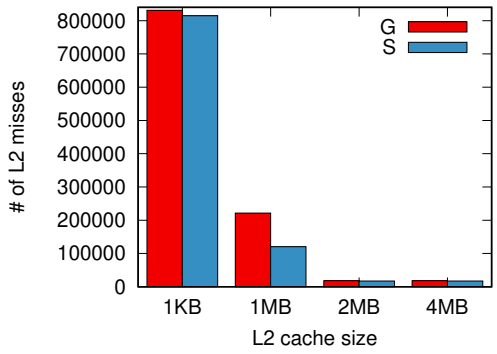

図 11 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の L2 ミス数

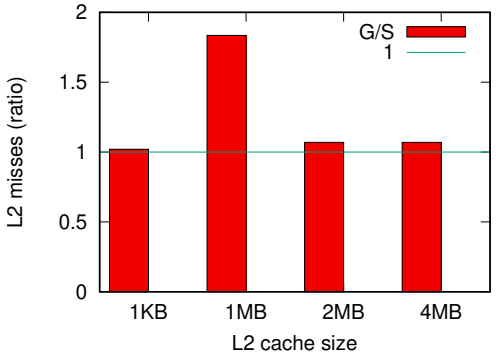

図 12 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の L2 ミス数比率 (G/S)

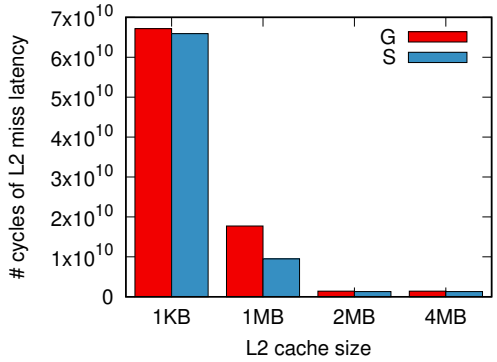

図 13 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の L2 ミス・レイテンシ

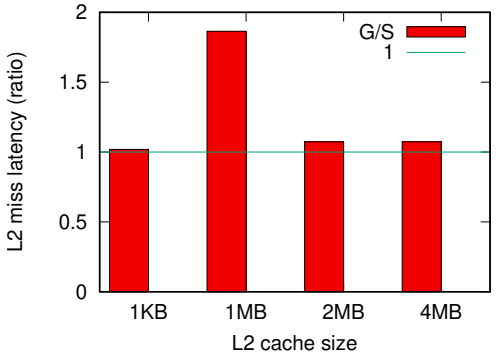

図 14 N=1024,
MPI 通信をファイル読み込みに置き換えた場合 (G) と省略した場合 (S) の L2 ミス・レイテンシ (G/S)

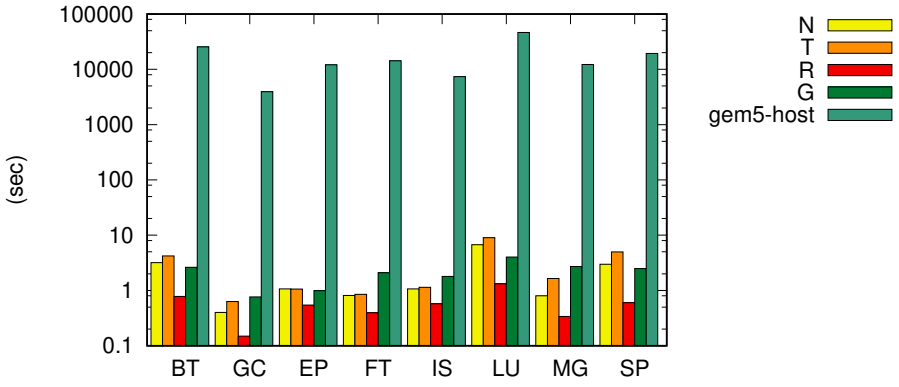

図 15　NPB の実行時間・推定実行時間・シミュレータのホスト時間

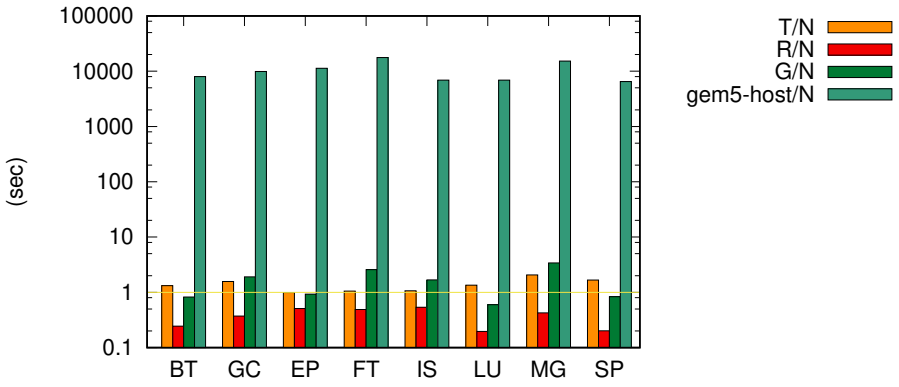

図 16　NPB の実行時間・推定実行時間・シミュレータのホスト時間 (N との比率)

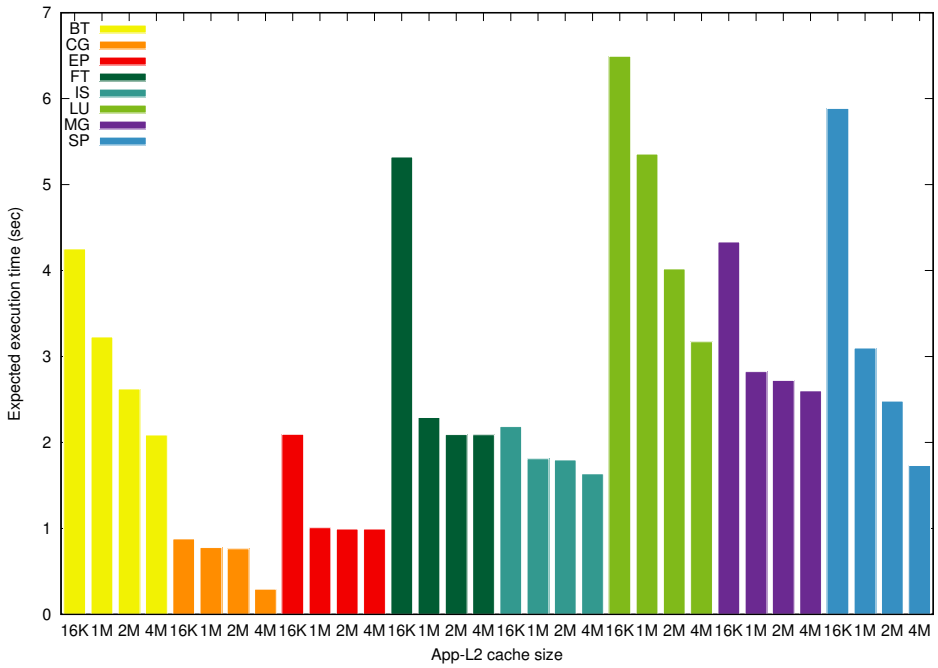

図 17　NPB の推定実行時間と L2 サイズ

MPI 通信をファイル読み込みに置き換えることは，キャッシュ利用を変化させ，アプリケーションの必要メモリ量と L2 キャッシュの容量が近いとき，実行時間の推定結果が比較的大きく変化することがわかった．キャッシュの容量がアプリケーションの必要メモリ量と比較して十分に大きい・もしくは小さい場合には，MPI リプレイを行わない場合とほぼ同様の推定実行時間を得ることができた．

また，ステンシル計算および NPB を用いて gem5 シミュ

レータ上でのリプレイに必要な時間について調査した．gem5 シミュレータの SE モードでは，IO はシミュレートしないため，MPI 通信をファイル読み込みに置き換えた場合でも，シミュレーションそのものにかかる時間はほとんど変化しなかった．ただし，もともと gem5 シミュレータによるシミュレーションは，実 CPU との実行と比較して $O(10^5)$ 倍のシミュレーション時間を要するため，アプリケーションによっては単純なリプレイでは現実的な時間内での単体性能推定が不可能なこともあると考えられる．

今後の課題としては，MPI 通信をファイル読み込みに置き換えることで大きく性能が変化する場合とそうでない場合の判定方法を確立し，実行時間の修正のための方法論を検討すること，CPU シミュレータによる演算時間推定とネットワークシミュレータによる通信時間推定を組み合わせて全体性能を推定する枠組みを作成することが挙げられる．

参考文献

[1] R. F. Barrett, S. Borkar, S. S. Dosanjh, S. D. Hammond, M. A. Heroux, X. S. Hu, and J. Luitjens. On the role of co-design in high performance computing. In *Transition of HPC Towards Exascale Computing*, pp. 141–155, 2013.
[2] gem5: The gem5 simulator system. http://www.gem5.org/.
[3] G. Hendry and A. Rodrigues. SST: A simulator for exascale co-design. In *ASCR/ASC Exascale Research Conference*, pp. –, 2012.
[4] NAS Parallel Benchmarks. http://www.nas.nasa.gov/publications/npb.html.
[5] M. Sato. Codesign for fugaku, invited talk, HPCAsia2020, 2020.
[6] 辻美和子, 佐藤三久. ノード単体性能評価のための MPI アプリケーションリプレイ環境の作成. 情報処理学会研究報告, No. 2015-HPC-150, p. On Line. 情報処理学会, 2015.