Vol.2016-HPC-153 No.6
2016/3/1

SCAMP: MPI疑似通信プロファイルによるネットワーク性能推定手法の提案

辻 美和子[1,a)]　李 珍泌[1]　朴 泰祐[2]　佐藤 三久[1]

概要：大規模並列システムのネットワークの協調設計のためには，仮想的なネットワーク上においてアプリケーションの性能を見積もる必要がある．性能推定の方法として，まず実際に並列実行を行い，得られた通信プロファイルを用いてオフライン・シミュレーションを行う手法は，平易に行えることから，しばしば用いられる．しかしながら，対象とする仮想システムの規模が現存のシステムよりもはるかに大規模な場合，通信プロファイルを直接利用することができない．本稿では，利用可能な実システムが小規模な場合でも，得られた少数のプロファイルを加工してスード・プロファイルと呼ばれる擬似的なプロファイルを大量に生成し，これに基づいて想定する大規模並列システムにおける性能を推定する方法 SCAlable Mpi Profiler (SCAMP) を提案する．

1. はじめに

近年の大規模並列システムの設計においては，システムとアプリケーションの協調設計 (Co-design) というコンセプトがしばしば用いられている [1]．協調設計は，システムとアプリケーションの双方向的なアプローチであり，システムがアプリケーションの要求を可能な限り満足するように設計される一方で，アプリケーションはシステムに向けに最大限に最適化される．協調設計の過程では，より良いシステムを追求するために，複数のアプリケーションの性能が候補システム上で推定・評価される．性能評価の方法のひとつは，既存のシステム上でアプリケーションを実行してプロファイルを取得し，これに基づいてオフライン・シミュレーションを行い仮想システムでの振る舞いを推し測ることである．

しかしながら，対象とする仮想システムの規模が現存のシステムよりもはるかに大規模な場合，通信プロファイルを直接利用することができない．本稿では，利用可能な実システムが小規模な場合でも，得られた少数のプロファイルを加工してスード・プロファイルと呼ばれる擬似的なプロファイルを大量に生成し，これに基づいて想定する大規模並列システムにおける性能を推定する方法 SCAlable Mpi Profiler (SCAMP) を提案する．

1 理化学研究所計算科学研究機構
RIKEN Advanced Institute for Computational Science
2 筑波大学
Univeristy of Tsukuba
a) miwako.tsuji@riken.jp

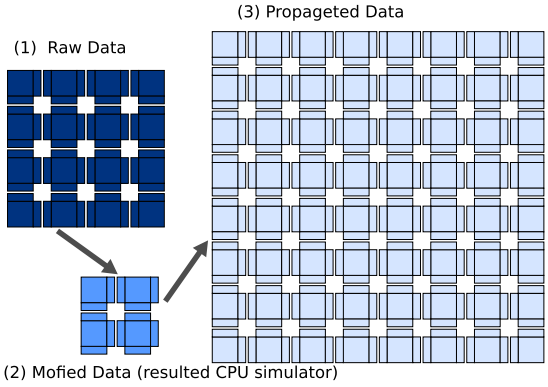


図 1　提案手法の全体像の概要

本稿の構成は以下である：2 章で研究の全体像と本稿で提案する MPI 疑似通信プロファイルを用いた大規模アプリケーションの性能予測について伸べる．3 章で提案手法について実験的に検証する．4 章で提案手法の課題について議論する．5 章で結論を述べる．

2. MPI 疑似通信プロファイルを用いた大規模アプリケーションの性能予測

本節では，研究の全体像を示すとともに，本稿で焦点を当てる MPI 疑似通信プロファイルによるネットワーク性能推定手法 SCAlable Mpi Profiler (SCAMP) について述べる．

2.1 概要

将来の並列システムにおけるアプリケーションの性能を

ⓒ 2016 Information Processing Society of Japan

```
#define XSIZE 1000

np = MPI_Comm_size();
myid = MPI_Comm_rank();
xsize = XSIZE / np;
ux = malloc(sizeof(double), xsize + 2);
uux = malloc(sizeof(double), xsize + 2);
MPI_Send(&ux[1],1,MPI_DOUBLE,..,(myid-1+np)%np,.);
MPI_Send(&ux[xsize],1, MPI_DOUBLE,..,(myid+1)%np,.);
MPI_Recv(&ux[xsize+1],1,MPI_DOUBLE,..,(myid+1)%np,.);
MPI_Recv(&ux[0],1,MPI_DOUBLE,...,(myid-1+np)%np,.);
for(i=1; i<=xsize; i++)
    uux[i]=(ux[i-1]+ux[i+1])-2.0*u[i];
for(i=1; i<=xsize; i++)
    ux[i]=uux[i];
```

図 2 MPI アプリケーションのプログラミング例．1 次元のステンシル計算を並列化する例．プロセス数やランク番号に関わる変数を斜体で，斜体文字から派生する変数を太字で示す．

予測するためには，演算時間および通信時間を見積もる必要がある．**図 1** に本研究で提案するアプリケーション全体性能の推定手法の概要を示す．まず，(1) 小規模なシステムにおいてアプリケーションを実行し，通信プロファイルを得る．プロファイルは，ランクごとに得られるものとする．(2) シミュレータ等を用いて，仮想システムの 1 プロセスにおけるアプリケーションの演算性能を見積り，通信プロファイルにおける通信と通信の間の演算部のタイムスタンプを修正する．続いて，(3) 得られた少数の通信プロファイルから，ソースコードの解析を通じて，送信先・受信元などの値を置き換え，多数のプロファイルを生成する．このようにして得られた仮想的なプロファイルをスード・プロファイルと呼ぶ．スード・プロファイルを用いてネットワークシミュレータを実行し，ネットワーク性能を推定する．

(2) の仮想システムにおける演算時間の見積りのために，著者らは，MPI アプリケーションの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 [6] を提案している．本稿では，通信性能推定に焦点をあて，(3) のように，スード・プロファイルを生成し，それに基づいてネットワーク性能を推定することを提案する．以降では，提案手法を SCAlable Mpi Profiler (SCAMP) と呼ぶ．

多くの MPI アプリケーションにおいて，性能に影響を与えるパラメータは，プロセス数やランク番号などから機械的に決定することができると考えられる．**図 2** に，1 次元のステンシル計算を 1 次元分割して並列化した MPI アプリケーションのソースコードの例を示す．図では，プロセス数やランク番号に関わる変数を斜体で，斜体文字から派生する変数が太字で示されている．この例のように，送信先，受信元のランク番号，バッファのアドレスなどを機械的に決定することができれば，小規模実行から得られた実プロファイルを元に大規模実行用の仮想プロファイルを生成することができる．ただし，本稿では，スード・プロファイルを用いたシミュレーションの有用性について検討することに主眼を置き，まずはスード・プロファイルを自動ではなく手動で生成する．

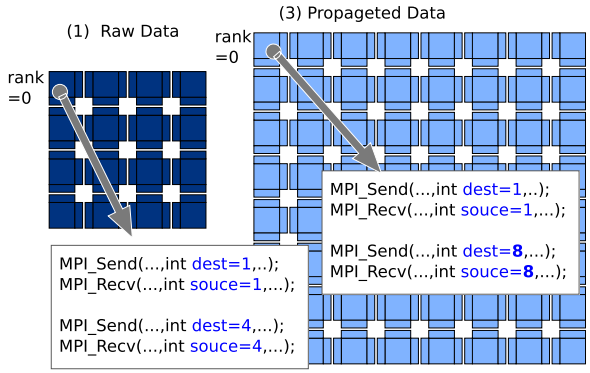


図 4 MPI 擬似通信プロファイル作成の概念図．二次元のステンシル計算 (非周期的) なアプリケーションの例．

2.2 MPI 疑似通信プロファイルによるネットワーク性能推定手法の提案

本稿では，スード・プロファイルを用いたネットワーク性能推定手法 SCAMP の有効性を検証するために，図 1 に示したうち，(1) 実プロファイルより直接 (3) スード・プロファイル を手動生成し，これを用いたシミュレーションを行った．

ネットワークシミュレータとしては，Sandia National Laboratories で開発されている SST/macro を用いた [3]．SST/macro は，分散メモリシステムのためのマクロスケールシミュレータであり，実際のアプリケーションをシミュレータ上で実行するオンライン・シミュレーションと，実機上の実行から得られた MPI 関数呼び出しのプロファイルから通信性能をシミュレーションするオフライン・シミュレーションが可能である．実システムにおける通信プロファイルの取得においては，同ツールのサブセットとして提供されている dumpi ライブラリが用いられる．一般的な MPI の実装における MPI 関数は，実際に通信を行う PMPI 関数のラッパーであり，ウィークシンボルとして提供される．dumpi ライブラリでは，MPI 関数をプロファイル取得と PMPI 関数の呼び出しにを行う関数に置き換えて実行する．よって，ユーザーは，プロファイルの取得のためにソースコードを書き換える必要がない．

図 3 に，dumpi によって得られたプロファイルの例を示す．MPI 関数名，開始時刻，終了時刻，送信先，受信元，要素数，データタイプ，タグ，コミュニケータの種類などが記録され，提供される．これらのプロファイルから，大規模システムを想定したスード・プロファイルを作成する．

```
MPI_Send entering at walltime 8835524.347071757, cputime 0.271717400 seconds in thread 0.
int count=1048576
MPI_Datatype datatype=9 (MPI_INT)
int dest=1
int tag=0
MPI_Comm comm=2 (MPI_COMM_WORLD)
MPI_Send returning at walltime 8835524.347971034, cputime 0.272616000 seconds in thread 0.
MPI_Recv entering at walltime 8835524.348989074, cputime 0.272629700 seconds in thread 0.
int count=1048576
MPI_Datatype datatype=9 (MPI_INT)
int source=1
int tag=0
MPI_Comm comm=2 (MPI_COMM_WORLD)
MPI_Status status=<IGNORED>
MPI_Recv returning at walltime 8835524.349915164, cputime 0.273555000 seconds in thread 0.
```

図 3 dumpi が提供する MPI プロファイルの例 (バイナリで得られた出力をテキスト化)

MPI 擬似通信プロファイル作成の概念図を**図 4** に示す．図の例では，非周期的境界条件を持つ 2 次元ステンシル計算を，2 次元に分割して $4 \times 4 = 16$ プロセスで実行し，得られたプロファイルから大規模な $8 \times 8 = 64$ プロセスを想定したスード・プロファイルを作成している．左上に位置する rank $= 0$ は，右隣および下隣のノードと袖領域を交換する．このときの通信相手は，4×4 では rank $= 1$ および 4 になり，8×8 では rank $= 1$ および 8 になる．よって，この場合は，4×4 の実行で得られた通信プロファイルから，source もしくは dest が 4 になっている部分を探して，8 に置き換えれば良い．

上述のように，スード・プロファイル作成においては，プロセス数や問題規模の変化に対して，MPI 関数の通信相手や通信量などの引数がどのように変化するかを考慮する必要がある．よって，これらの引数が機械的に決定できないアプリケーションに対しては，提案する手法でのスード・プロファイル作成およびネットワーク性能推定は難しい．しかしながら，図 2 に示したような，通信時間推定に影響を与えるようなパラメータが，比較的単純なルールに基づいて決定されるアプリケーションは多数存在すると考えられる．

スード・プロファイルの生成は，将来的にはソースコードの解析を通して自動的に行えるようにしたいと考えているが，本稿では，スード・プロファイル作成とそれに基づくオフライン・シミュレーションという提案の枠組みの妥当性を検証するために，比較的単純なアプリケーションを対象として，手動で置き換えルールを作成し，スード・プロファイルの生成を行う．

3. 実験

3.1 SST/macro パラメータ

すべての実プロファイル取得は，京コンピュータ上で行った．また，スード・プロファイルによるシミュレーションは京コンピュータのパラメータに基づく仮想的なシステムで行った．ただし，シミュレーション実行時間を短縮するため，一部のパラメータは変更した．**図 5** に SST/macro のパラメータを示す．また，京コンピュータの諸元を**表 1** に示す．

```
node_cores = 8
node_mem_latency = 200ns
node_mem_bandwidth = 64Gbps
node_frequency = 2.0e9

topology_name = torus
topology_geometry = ...

launch_app1_start = 0ms

network_name = switch
network_node_per_switch = 1
network_switch_type = infinite_buffer
packet_switch_MTU = 8192
packet_switch_bandwidth_n2r = 100gbytes/sec
network_bandwidth_link = 5.0gbytes/sec
network_hop_latency = 100ns

nic_name = packet
nic_injector = infinite_buffer
nic_ejector = infinite_buffer
```

図 5 SST/macro のパラメータ

表 1 京コンピュータ諸元

CPU	Fujitsu SPARC64VIIIfx, 8 core, 2.00 GHz
Memory	16GB , 64GB/s
Cache	L1:32+32KB/core, L2:6MB/core
Network	Tofu Interconnect 5GB/s x 2

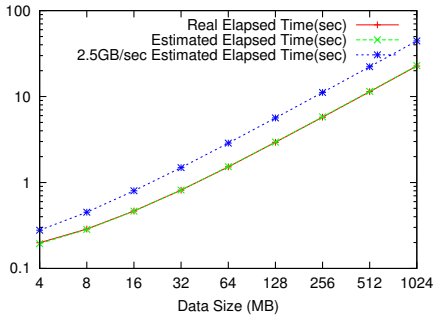


図 6 Pingpong の実実行時間と SST/macro による推定実行時間

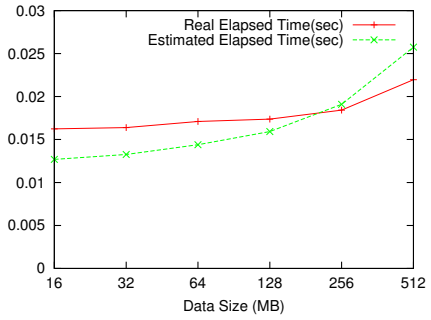


図 7 Allreduce の実実行時間と SST/macro による推定実行時間

ここで，SST/macro においては，ネットワークトポロジとして，ノードではなくスイッチのトポロジを指定する必要がある [5]．また，京コンピュータの MTU は ∼2K であるが，MTU サイズの小さいシミュレーションは非常に時間がかかるため，8K とした．network_hop_latency は，後述の ping-pong 実験で，比較的小さいサイズのデータを送ったときの実機の計測と結果があうように調整した．

3.2 予備実験

3.2.1 ping-pong

提案手法の妥当性を検証する前に，dumpi と SST/macro の妥当性を検証するために，ping-pong アプリケーションの実実行およびシミュレータ結果を比較した．ping-pong アプリケーションは，2 ノードによる Send/Recv の 100 回の繰り返しである．Recv もしくは Send の後に 1 ミリ秒の sleep を挿入した．受送信するデータのサイズは 4 から 1024 MB まで変化させた．実実行の時間は，rank = 0 における MPI_Init の呼び出し時刻と MPI_Finalize の終了時刻の差とした．ping-pong はスレッド並列を用いないアプリケーションであるが，1 ノードに 1 プロセスを配した．

図 6 に，実実行時間と SST/macro での推定時間を示す．なお，実実行時刻は，dumpi のログ取得のオーバーヘッドを含む．これは，dumpi のログから行われる SST/macro による実行時間見積りも，このオーバーヘッドの修正を行っていないためである．前述のように，シミュレーションで用いるレイテンシのパラメータは，実実行時間と合うように実験的に決定された．図で 2.5GB/sec の凡例がついているのは，SST/macro の入力パラメータのうち，ネットワークバンド幅を半分の 2.5GB/sec にした場合のシミュレーション結果である．SST/macro は，ネットワークバンド幅に律速される受信や送信に関しては，適切な推定が行えることがわかった．

3.2.2 Allreduce

次ぎに，集合通信を調査した．要素数 1 の MPI_Allreduce (MPI_SUM) と 1 ミリ秒の sleep を 100 回繰り返した．プロセス数は $n = 16 \sim 512$ とし各プロセス数に関して 10 回の試行を行い，平均時間を算出した．ネットワークトポロジは，1 次元トーラス $n \times 1 \times 1$ とした．**図 7** に結果を示す．図から，SST/macro は，プロセス数が小さいときは実際よりも速く，プロセス数が大きいときは実際よりも遅く，実行時間を見積りをする．この一因としては，SST/macro が，シミュレーション時に京コンピュータにおける集団通信のアルゴリズムを考慮していないことが挙げられる．

表 2 各ベンチマークの主要な MPI 関数の呼び出し関数

	EP (class.np)	
	B.4	C.16
Allreduce	4	4
	FT	
	B.4	C.16
Alltoall	22	22
Reduce	20	20
	CG	
	C.16	C.64
Send	13984	19988
Irecv	13984	19988
Wait	13984	19988
	CCS-QCD (class.np)	
	2.64	3.512
Sendrecv	12036	12036
Barrier	2006	2006
Allreduce	3012	3012

3.3 実験

3.4 アプリケーションの通信特性とスード・プロファイル生成手法

本実験ではベンチマークとして NAS Parallel Benchmarks (NPB) 3.3.1 の CG, EP, FT (以下，NPB-CG, NPB-EP, NPB-FT) [2]，および Fiber Miniapp Suite から CCS-QCD [4] を用いた．**表 2** に，これらのベンチマーク内で呼び出される主要な MPI 関数を示す．

以下では，それぞれのベンチマークに関して SCAMP で性能を推定した．また，推定時と相当する問題サイズおよびプロセス数を実際に実行し，結果を比較した．すべての実験は，1 ノードあたり 1 プロセスを実行した．NPB-EP, NPB-FT, NPB-CG はスレッド並列化されてい

ない．CCS-QCD は OpenMP によりスレッド並列化されており，1 プロセスあたり 8 スレッドが用いられる．

3.4.1 NPB-EP

NPB-EP

NPB-EP は，乱数の生成を行うベンチマークである．NPB-EP のクラス B を 4 プロセスで実行したプロファイルから，16, 256, および 4096 プロセスのスード・プロファイルを作成した．表 2 に示すように，NPB-EP は 4 回の MPI_Allreduce を行う．すべてのランクが同じ通信を行うため，スード・プロファイルの作成においては，無作為に選んだオリジナル・プロファイルを，推定対象のプロセス数の分だけ複製した．

NPB-EP・クラス B・プロセス数 4 を，16 プロセス分のスード・プロファイルに水増しした場合，NPB-EP・クラス C・16 プロセス と，256 プロセス分のスード・プロファイルに水増しした場合，NPB-EP・クラス D・256 プロセス と，4096 プロセス分のスード・プロファイルに水増しした場合，NPB-EP・クラス E・4096 プロセス と，それぞれほぼ同等になるはずである．

図 9 に，これらの NPB-EP のそれぞれのクラス・ノード数の実際の実行時間と，SCAMP による推定実行時間を示す．**図 10** に，図 9 の 30 秒付近を拡大した図を示す．いずれの場合も SCAMP によって，適切な実行時間予測がなされている．

NPB-FT

NPB-FT は，三次元空間を一次元もしくは二次元に分割して，高速フーリエ変換 (FFT) を行う．二次元分割の場合，MPI_Comm_split によって，列方向および行方向に分割したコミュニケータを作成し，これらの内部で MPI_Alltoall を行う．スード・プロファイルの作成においては，プロセス数の変化によって変化する MPI_Comm_split の分割パターンを置き換えたほか，Alltoall で受信・送信するデータの個数は，Alltoall を実行するコミュニケータの大きさに半比例して少なくなる点を反映した．

空間サイズ $2048 \times 256 \times 256$ を 4×4 プロセスで実行して得られた実プロファイルから，空間サイズ $2048 \times 512 \times 512$ を 8×8 プロセスで実行したと仮定してスード・プロファイルを作成した．また，実際に同サイズ，同プロセス数の実験行い，実実行時間と推定実行時間とを比較した．結果を**図 11** に示す．なお，繰り返し回数は 5 に固定した．

実実行においても推定においても，プロセス数の増加にともない，実行時間が増加しているが，提案手法による推定のほうがより増加率は小さかった．

図 13 に，実実行において，MPI_Alltoall と MPI_Alltoall の間の経過時間を各ランク番号ごとに調査し，もっとも経過時間が大きかったものと小さかったものの差を調査した結果を示す．図に示されるように，プロセス数が少ない場合は MPI_Alltoall 間の時間のランクごとのばらつきが少なく，プロセス数が多い場合は MPI_Alltoall 間の時間のノードごとのばらつきが大きくなっている．このような実行時間のばらつきは集団通信の待ち時間となって性能に影響するが，本実験ではそのような実行時間の分散の大きさを考慮していない．また，**図 14** に，各ランクにおけるある MPI_Alltoall と MPI_Alltoall の間の時間を示した．図に示すように，プロセス数と問題サイズが大きくなると，プロセスあたりの三次元形状が変化することから，演算時間も全体に増加するが，今回の実験では，スード・プロファイル作成の際にこのような演算時間の変化を性能推定に取り入れていないため，推定時間が実実行時間よりも短くなったと考えられる．演算時間の修正に関しては，2.1 で述べたように，将来的にはネットワーク性能推定の前の処理として行う予定である．

NPB-CG

NPB-CG は，疎行列の固有値を共役勾配法で求めるベンチマークである．行列を一次元もしくは二次元に分割する．表 2 に示すように，MPI_Irecv/Send/Wait を使用する．スード・プロファイルの作成にあたっては，MPI_Irecv/Send/Wait の回数が，列あたりの分割数 n_{col} に対して $\log_2(n_{col})$ で変化する点に留意する必要がある．

本実験では，NPB-CG のクラス C を 16 プロセスで実行して，プロファイルを得た後，これを 64 プロセス分および 256 プロセス分に水増しした．$16 = 4 \times 4$ プロセスから $64 = 8 \times 8$ プロセスにプロセス数が増加した場合，16 プロセスのときに 2 セット行っていた Irecv/Send/Wait を 3 セットに増やす必要がある．同様に，$256 = 16 \times 16$ プロセスに増やす場合には，Irecv/Send/Wait を 4 セットに増やす必要がある．ここでは，受信先，送信元，データサイズを再計算するとともに，新たな MPI 関数の挿入にともない，**図 8** に示すように，Wait 終了から新たに挿入した Irecv 開始までの間，および Irecv-Send 間，Send-Wait 間の時間を実行時間のそれで補正し，以後のタイムスタンプをすべてずらすことで，スード・プロファイルを作成した．

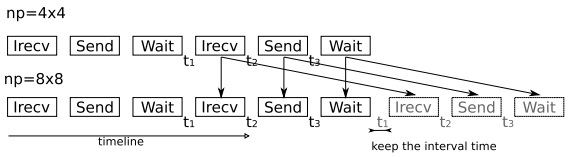
図 12 に，NPB-CG の SCAMP による推定実行時間を示す．NPB-CG は問題クラスごとに列数はもちろん，行列の非ゼロ要素が変化し，スード・プロファイル生成によってプロセス数を増加させた仮想実行に対応する NPB クラスは存在しない．よって，実実行時間との比較は行わない．

CCS-QCD

CCS-QCD は格子 QCD 計算を行うベンチマークである．本実験では，CCS-QCD のうちで周期的境界条件のステンシル計算を行う BiCGStab 計算部のみを考慮した．SST/macro によるシミュレーションでは，該当部分のプロファイルのみを抜き出して実行した．QCD では四次元の格子を扱うが，このうち空間方向を示す格子が三次元分割される．CCS-QCD の問題クラスは，クラス 2 および 3 を用いた．格子点数は，前者が $32 \times 32 \times 32 \times 32$ で，

プロファイル

```
MPI_Irecv entering at walltime 10000.000 ..
..  1-st
MPI_Irecv returning at walltime 10010.000 ..
MPI_Send entering at walltime 10020.000 ..
..
MPI_Send returning at walltime 10030.000 ..
MPI_Wait entering at walltime 10040.000 ..
..
MPI_Wait returning at walltime 10050.000 ..
MPI_Irecv entering at walltime 10060.000 ..
..  // 2-nd
MPI_Irecv returning at walltime 10070.000 ..
MPI_Send entering at walltime 10080.000 ..
..
MPI_Send returning at walltime 10090.000 ..
MPI_Wait entering at walltime 10100.000 ..
..
MPI_Wait returning at walltime 10110.000 ..
MPI_Irecv entering at walltime 10200.000 ..
..  // 1-st'
```



スード・プロファイル

```
MPI_Irecv entering at walltime 10000.000 ..
..  // 1-st
MPI_Irecv returning at walltime 10010.000 ..
MPI_Send entering at walltime 10020.000 ..
..
MPI_Send returning at walltime 10030.000 ..
MPI_Wait entering at walltime 10040.000 ..
..
MPI_Wait returning at walltime 10050.000 ..
MPI_Irecv entering at walltime 10060.000 ..
..  // 2-nd
MPI_Irecv returning at walltime 10070.000 ..
MPI_Send entering at walltime 10080.000 ..
..
MPI_Send returning at walltime 10090.000 ..
MPI_Wait entering at walltime 10100.000 ..
..
MPI_Wait returning at walltime 10110.000 ..
MPI_Irecv entering at walltime 10120.000 ..
..  // 3-rd
MPI_Irecv returning at walltime 10130.000 ..
MPI_Send entering at walltime 10140.000 ..
..
MPI_Send returning at walltime 10150.000 ..
MPI_Wait entering at walltime 10160.000 ..
..
MPI_Wait returning at walltime 10170.000 ..
MPI_Irecv entering at walltime 10270.000 ..
..  // 1-st'
```

図 8 NPB-CG におけるスード・プロファイル生成

後者が $64 \times 64 \times 64 \times 32$ である．プロセス数はそれぞれ $64 = 4 \times 4 \times 4$ および $512 = 8 \times 8 \times 8$ である．これらのノードどうしは 3 次元のトーラス上に配置される．比較のためにすべての問題クラスにおいて，繰り返し回数は 1000 に固定した．

スード・プロファイルの生成においては，クラス 2 で得られたプロファイルに対して，各空間方向の隣接通信の通信相手のランク番号，および MPI_Sendrecv のタグの置き換えを行った．

結果を**図 15** に示す．NPB-FT のときと同様に，プロセス数増加にともなう演算時間の分散の増大などが考慮されていないため，推定実行時間のほうが実際の実行時間に比べて短くなっている．

4. 考察

4.1 提案手法の適用範囲

本研究では，機械的に予測可能な通信パターンを持つアプリケーションが多く存在すると仮定して，少数のプロセスでアプリケーションを実行して得られた通信プロファイルから大量のスード・プロファイルを生成し，それに基づいて通信性能推定を行うことを提案した．しかし，すべてのアプリケーションが，機械的に予測可能な通信パターンで実行させるとは限らない．

本節では，Fiber Miniapp Suite に集められているアプリケーション・ベンチマークにおける通信関数の使用状況を調査した．結果を**表 3** に示す．MPI_Comm_rank, MPI_Comm_size, MPI_Wtime などの通信を行わない関数は無視している．また，初期化のみ呼び出される関数，カーネル部で呼び出される関数，存在するが呼び出されない関数を区別しない．スード・プロファイルの機械的な生成が困難であることが予測される MPI 関数は，MPI_Allgatherv などの プロセスごとに可変長のデータを扱う関数群であるが，表からこれらの関数はあまり用いられていないことがわかる．

4.2 スード・プロファイル生成の自動化

本稿では，著者らがアプリケーションを解析し，手動でスード・プロファイルの生成を行ったが，さまざまなアプリ

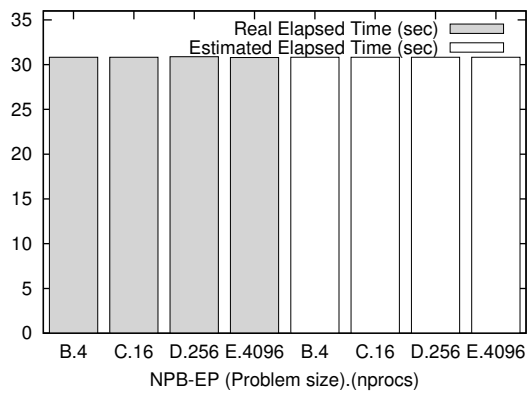


図 9　NPB-EP の実実行時間と推定実行時間

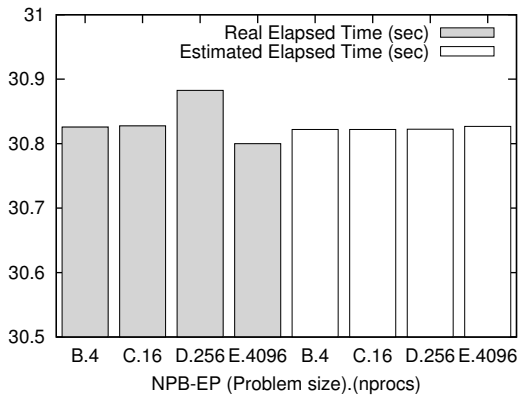


図 10　NPB-EP の実実行時間と推定実行時間 (拡大)

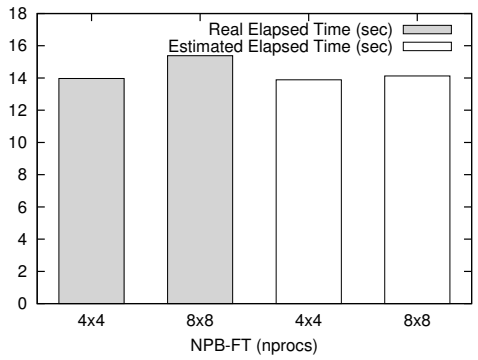


図 11　NPB-FT の実実行時間と推定実行時間
プロセス数 4x4 の問題サイズ 2048x256x256,
プロセス数 8x8 の問題サイズ 2048x512x512

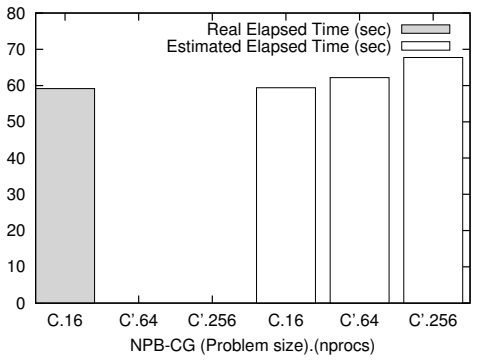


図 12　NPB-CG の実実行時間と推定実行時間

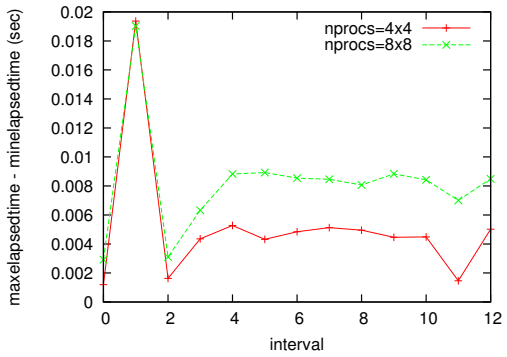


図 13　NPB-FT の alltoall 間のプロセスごとの実行時間について，実行時間がだった最大だったプロセスと最小だったプロセスの差を調べた

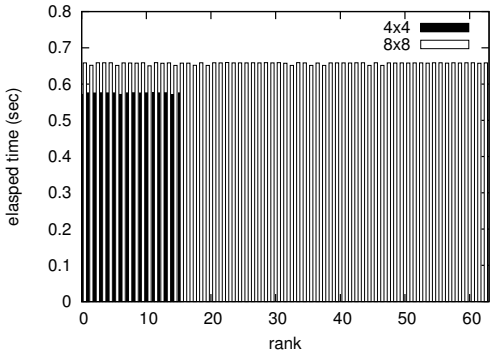


図 14　NPB-FT のあるの alltoall 間における各プロセスの実行時間

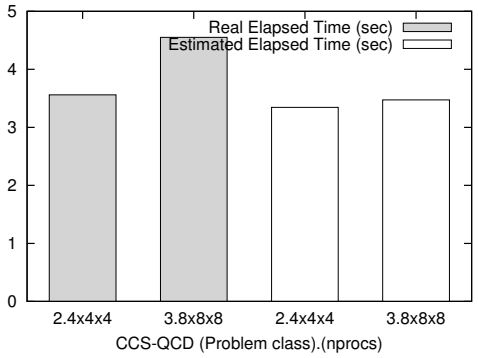


図 15　CCS-QCD の実実行時間と推定実行時間

ⓒ 2016 Information Processing Society of Japan

表 3 Fiber Miniapp Suite の各アプリケーションベンチマークにおいて使用されている MPI 関数の種類

Application	CCS-QCD	FFVC-MINI	NICAM-DC-MINI	mVMC-mini	NGS Analyzer-MINI	NTChem-MINI	FFB-MINI
Verson	1.2.1	1.0.1	1.0	1.0	1.0.1	1.1	1.0.1
Send		o					
Bcast	o	o	o	o		o	
Isend	o	o	o			o	o
Irecv	o	o	o			o	o
SendRecv	o						
Wait	o	o				o	
Waitall		o	o				o
Barrier	o	o	o		o	o	o
Reduce		o	o			o	o
Gather		o		o			
Allreduce	o	o		o		o	o
Allgather		o					
Scatter				o			
Gatherv		o					
Allgatherv		o					

ケーションを平易に解析できるようにするためには，スード・プロファイルの生成は自動的に行うことが必要とされる．そこで，将来的には，コンパイラの定数伝搬などの機能を利用して，MPI_Comm_rank や MPI_Comm_size で得られた値をソースコードがどのように利用しているかを解析することで，スード・プロファイルの自動生成を行う予定である．

5. おわりに

本稿では，アプリケーションを小規模なシステムで並列実行し，得られたプロファイルを加工してスード・プロファイルと呼ばれる擬似的なプロファイルを大量に生成し，これに基づいて想定する大規模並列システムにおける性能を推定する方法を提案した．アプリケーションベンチマークを用いた提案手法の実験的検証を行った．提案手法は，プロセス数が大きくなったときの

- 演算時間そのものの増加
- ランクごとの演算時間のばらつきやそれによる通信開始待ち時間の増加

などが再現できない．よって，提案手法は，全体の実行時間推定は実実行時と比較して楽観的な数値を与える傾向があるものの，仮想システムにおける通信性能の傾向を調査できることがわかった．今後の課題として，

- 通信性能推定の前の段階の処理として，演算時間の修正や予測を組み込む
- スード・プロファイル生成の自動化

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, J. Luitjens, S. G. Parker, J. Shalf, and L. Tang. On the role of co-design in. high performance computing. In *Advances in Parallel Computing /Volume 24: Transition of HPC Towards Exascale Computing*, pp. 141 – 155. 2013.

[2] N. P. Benchmarks. http://www.nas.nasa.gov/ publications/npb.html.

[3] G. Hendry and A. Rodrigues. SST: A simulator for exascale co-design. In *ASCR/ASC Exascale Research Conference*, pp. –, 2012.

[4] RIKEN Advanced Institute for Computational Science. Fiber Miniapp Suite http://fiber-miniapp.github.io/.

[5] Sandia National Labs. SST/macro 4.0: User's manual, 2015.

[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.

ⓒ 2016 Information Processing Society of Japan