Vol.2014-HPC-144 No.10
2014/5/27

メニーコアクラスタにおけるジョブ間の性能干渉について

堀 敦史[1,a)]　亀山 豊久[1,b)]　並木 美太郎[2,c)]　石川 裕[1,3,d)]

概要：メニーコア CPU とマルチコア CPU が混在するような計算ノードから構成されるクラスタにおいて、2 つのジョブを混在させた場合のジョブ間の性能干渉について調べた結果について報告する。ジョブの組み合わせとしては、メニーコア CPU とマルチコア CPU の両方それぞれに別なジョブを投入した場合と、メニーコア CPU あるいはマルチコア CPU の片方に 2 つのジョブを投入した場合について調査した。その結果、マルチコア CPU（Intel Xeon）上のジョブは、メニーコア CPU（Intel Xeon Phi）上のジョブのノード間通信の影響を受けることが判明した。一方、メニーコア CPU 上の並列ジョブは、マルチコア CPU ジョブの通信の有無に関わらずマルチコア CPU ジョブにより大きな影響を受けることが判明した。また、メニーコア CPU 上での 2 つのジョブ実行は、場合によっては数%程度にまで性能が落ち込むことが判明した。残念ながら、この現象を解明するまでに至らなかった。今後の研究が望まれる。

Job Performance Interference on Many-core and Multicore Cluster

Atsuhi Hori[1,a)]　Toyohisa Kameyama[1,b)]　Mitaro Namiki[2,c)]　Yutaka Ishikawa[1,3,d)]

Abstract: Job performance interference on a heterogeneous cluster consisting of multi-core CPUs and many-core CPUs is reported. All possible combinations of two jobs running on many-core CPUs and multi-core CPUs, many-core CPUs only, and multi-core CPUs only are investigated. As a result, paralle job performance running on multi-core CPUs are affected by the communication invoked by the job running on many-core CPUs. However, parallel job performance running on many-core CPUs can be affected by the jobs running on many-core CPUs with or without communication. When two jobs running on many-core CPUs, its performance is degraded to few percent of a single job performance. The reason explaning this phenomenon is left for future work.

1. 研究の背景

我々はより高性能なクラスタ型並列計算機として図 1 に示すようなアーキテクチャ上でのシステムソフトウェアの研究を進めている。図 1 に示されるように、クラスタを構成する計算ノードは、現在広く使われているマルチコア CPU、Intel 社製の Xeon Phi[6], [12] に代表されるメニーコア CPU、および高速なネットワークが主要なコンポーネントとなっている。これらは通常 PCI などの I/O バスにより接続されている。

マルチコア CPU と同様に、メニーコア CPU は独自のメモリを持ち、単独でアプリケーションの実行が可能である。Xeon Phi（以下、”Phi” と記す) による混在型計算ノードでは、Intel 社により、1) マルチコア CPU 上の計算、2) offloading などによるマルチコアとメニーコアの双方を用いた計算、3) メニーコア CPU 上の計算の 3 種類のジョブ実行がサポートされている [6], [12]。我々も独自にメニーコア CPU とマルチコア CPU の両方を用いた計算環境として、[2], [16] を提案している。

大規模なクラスタにおいては、複数のユーザが同時にジョブを投入して使用する場合が大半である。図 1 のようなクラスタにおいて、メニーコア CPU のみ、あるいはマルチコア CPU のみを用いるジョブを走らせるのでは、もう

[1] 独立行政法人理化学研究所計算科学研究機構
RIKEN AICS
[2] 東京農工大学
Tokyo University of Agriculture and Technology
[3] 東京大学情報基盤センター
Information Technology Center, University of Tokyo
[a)] ahori@riken.jp
[b)] kameyama@riken.jp
[c)] namiki@cc.tuat.ac.jp
[d)] ishikawa@is.s.u-tokyo.ac.jp

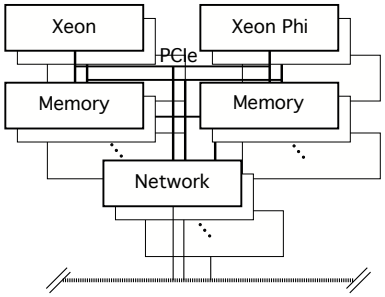

図 1 メニーコア混在型計算ノード

一方の CPU が遊んでしまう。クラスタ全体のジョブのスループットを向上させるという観点からは、例えばメニーコア CPU のみを用いるジョブが走っていた時に、マルチコア CPU のみを用いるジョブを走行させることで、ジョブのスループットを向上させることができる。また、京コンピュータ [14], [15] のようにメニーコア CPU 上のフォアグラウンドジョブ実行中に、バックグラウンドジョブとしてマルチコア CPU 上でその前後で実行するジョブのファイルステージングをするような使い方も考えられる。

このように計算ノード上で複数のジョブを実行する場合に問題となるのが、互いの性能干渉である。例えば、図 1 において、マルチコア CPU とメニーコア CPU のそれぞれで別な並列ジョブを実行させた場合、単一のネットワークデバイスを共有することになり、性能干渉が生じる可能性がある。このように、計算ノード上に複数のジョブを混在させた結果、性能干渉が生じるとジョブの実行時間が、他のジョブの影響を受けて変化する。

多くのバッチスケジューリングの運用において、短い実行時間のジョブの方が長い実行時間のジョブよりも優先的に実行されることが多い。また、ジョブ投入時に宣言された実行時間以内に実行が終了しないと実行が打ち切られてしまうような運用も多い。このため、ジョブの実行時間がなんらかの影響により変化するような実行環境は好ましくない。

Weinberg と Snavely は、彼らが提案する symbiotic な空間分割ジョブスケジューリング において性能干渉が及ぼす場合に、ジョブスケジューラは以下に示す 3 つの要件を挙げている [13]。

Accountability スケジューリングの結果として、ジョブの実行時間 (およびその結果としてのコスト) に変化が生じる場合は、ユーザが納得できなければならない

Familiarity ユーザが指定するスケジューリングのパラメータはユーザにとって馴染みのあるものでまけれればならない

Simplicity ユーザへの質問は単純で簡潔でなければならない

本研究では、図 1 のようなクラスタにおいて、さまざまなパターンで複数のジョブを投入し、互いにどのような性能干渉を及ぼすかについて実験をおこなった結果について報告する。

2. 実験

実験に用いたクラスタは、表 1 に示す仕様を持つ計算ノードが 8 台で構成されるクラスタを用いた。投入するジョブは NAS Parallel Benchmark プログラム [8] (以下、NPB と記す) と、別途作成した通信のみをおこなう MPI プログラムである。このプログラムは、図 2 で示されるように、`MPI_Send` と `MPI_Recv` を対向して通信しあう。

表 1 実験に用いた計算ノードの仕様

マルチコア CPU	Intel Xeon E5-2650-v2 8 cores 2.60GHz (IvyBridge-EP) 2 sockets
メニーコア CPU	Xeon Phi 5110P (60 cores)
ネットワーク	HCA: Mellanox MT27500 ConnectX3 Switch: Mellanox SX6036
ソフトウェア	CentOS release 6.5 uOS Version: 2.6.38.8+mpss3.2 SCore Version: 7.0.2-改 [9]

性能干渉を調べるにあたり、性能を測る方をフォアグラウンドジョブと呼び、干渉のための別なジョブを走らせる。この干渉させる方をバックグラウンドジョブと呼ぶ。実行時間は、常にバックグラウンドジョブの方が長くなるようにした。NPB プログラムは `MPI_Init` の直後から、`MPI_Finalize` の直前までを指定回数だけ繰り返すように新たなループを入れ込み、この実行時間の制約を満たすようにした。表 2 の実験に用いた NPB プログラムの問題サイズとプロセス数 (“ノード数” × “ノード内プロセス数”) を示す。基本的にクラス C を用いたが、Phi の IS だけはメモリ不足で動かなかったのでクラス W とした。Xeon においてはクラス D の実行も可能であったが、より通信の影響を受け易いと考えられるクラス C とした。また、以下の実験において Phi 上で走るプログラムは Intel MPI を用いた。Xeon 上で走る MPI プログラムは MPICH-SCore[9][*1]を用い、バックエンドコンパイラは Intel コンパイラを使用した。Phi はコア当たり 4 つの HyperThreading が可能であり、最大 240 の仮想コアを持つが、Intel MPI におけるノードあたりのプロセス数に SCIF[5] の制限があり、結果的にノード当たり 64 プロセスとした。また、Phi 上で MPI プログラムを実行すると Xeon 側のデーモンプロセスがかなりの負荷になるため、その影響を避けるために Xeon 上のプロセス数は 8 とした。

以下の実験では、想定されているマルチコア CPU とメニーコア CPU を持つ計算ノードを持つクラスタ上で、2 つのジョブを投入した時に考えられる全ての組み合わせ、す

[*1] SCore は CentOS 6.5 に対応するように変更を加えたもの。

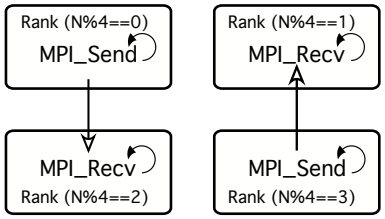

図 2 通信ジョブの処理内容

表 2 実験に用いた NPB プログラムの問題サイズ

	EP	CG	FT	IS	LU	MG
Xeon (8x8)	C	C	C	C	C	C
Phi (8x64)	C	C	C	W	C	C

なわち、

- マルチコアジョブとメニーコアジョブの間での性能干渉
- マルチコア上のふたつのジョブ間での性能干渉
- メニーコア上のふたつのジョブ間での性能干渉

の評価をおこなう。

2.1 マルチコアジョブとメニーコアジョブ間の性能干渉

2.1.1 マルチコアジョブへの性能干渉

表 3 に、フォアグラウンドジョブ（マルチコア Xeon）およびバックグラウンドジョブ（メニーコア Xeon Phi）共に図 2 に示した通信ジョブを走らせた時の干渉率を示す。ここで干渉率とは、バックグラウンドジョブがない時の性能に対するバックグラウンドジョブがあった時の割合と定義される。この数字が “1.00” の場合は、バックグラウンドの影響を全く受けなかったことを示し、例えば “0.50” の場合は、バックグラウンドの影響を受けて、性能が半分に低下したことを表す。表中には、バックグラウンドジョブがない状態での通信ジョブが計測したバンド幅も併せて載せてある。通信ジョブは、通信メッセージ長を 1KB、8KB、64KB、512KB、4MB と変化させた。表中の各行は、行の先頭に示したメッセージ長の通信をバックグラウンドでおこなった場合のフォアグラウンドの性能干渉率を示す。

この表から、Xeon 上のフォアグラウンドジョブのメッセージ長が長い程、Phi 上のバックグラウンドジョブの影響を受け易いことが分かる。しかし、一方、バックグラウンドジョブのメッセージ長の変化の影響はほとんどないことが分かる。

表 3 MPI 通信性能の干渉率（FG:Comm/Xeon、BG:Comm/Phi）

	1KB	8KB	64KB	512KB	4096KB
BW [MB/s]	1703	5395	5540	5556	5611
1KB	0.95	0.97	0.71	0.65	0.49
8KB	0.93	0.95	0.73	0.64	0.49
64KB	0.92	0.96	0.74	0.63	0.49
512KB	0.93	0.95	0.74	0.64	0.47
4096KB	0.93	0.95	0.72	0.64	0.49

表 4 は、フォアグラウンドジョブ、バックグラウンドジョブの双方で NPB を走らせた時の干渉率である（以下、本論文における干渉率の表の見方については表 3 と同じである）。EP はほとんど通信をしないプログラムなので、バックグラウンドジョブの影響を全く受けていないことが分かる。これは、換言すれば、Xeon においては通信以外の影響はほとんどないということを意味する。一方、遅延の影響を受け易い CG、通信が大きな比率を占める FT や IS は、バックグラウンドジョブの影響を受けていることが分かる。しかしながら、表 3 では最大約半分まで性能が低下していたのに対し、NPB の場合では最大で 6% の性能低下しか観測できなかった。この点に関しては、次節で議論したい。

表 4 NPB の性能干渉率（FG:NPB/Xeon、BG:NPB/Phi）

	EP	CG	FT	IS	LU	MG
EP	1.00	1.00	1.00	1.00	1.00	1.00
CG	1.00	0.97	0.97	0.95	0.99	0.98
FT	1.00	0.97	0.94	0.96	1.00	0.98
IS	1.00	0.98	0.99	0.97	1.00	1.00
MG	1.00	0.99	0.98	0.96	1.00	0.99

表 5 は、フォアグラウンドジョブとして NPB を、バックグラウンドジョブとして通信ジョブを走らせた時の干渉率である。干渉の結果、通信ジョブが 4MB のメッセージで通信した場合に最大で 10% の性能低下を見ることができた。表 4 と同様に表 3 で見られたような大きな影響はみられなかった。これは、NPB の多くのプログラムが、計算と通信のフェーズに分かれて実行されており、バックグラウンドの影響を受けるのは通信の部分だけであるため、プログラムの実行時間に対する影響が少なくなったと考えることができる。

表 5 NPB と通信ジョブの性能干渉率（FG:NPB/Xeon, BG:Comm/Phi）

	EP	CG	FT	IS	LU	MG
1KB	1.00	0.99	0.99	0.98	1.00	0.99
8KB	1.00	0.99	0.99	0.99	1.00	1.00
64KB	1.00	0.97	0.95	0.96	0.99	0.98
512KB	1.00	0.92	0.91	0.93	0.97	0.96
4096KB	1.00	0.92	0.90	0.92	0.98	0.96

表 6 は、表 5 の逆、つまり、フォアグラウンドを通信ジョブ、バックグラウンドを NPB とした時の干渉率である。

2.1.2 メニーコアジョブへの性能干渉

前節とは逆に、フォアグラウンドを Phi にした場合の干渉率の結果を表 7 に示す。Xeon の場合（表 3）と比べるとその挙動は大きく異なる。例えば、バックグラウンドジョブの通信メッセージサイズが 1KB の時、その影響はほとんど見られない。また、表 3 では、フォアグラウンドジョ

表 6 通信ジョブと NPB の性能干渉率（FG:Comm/Xeon, BG:NPB/Phi）

	1KB	8KB	64KB	512KB	4096KB
EP	1.00	1.00	1.00	1.00	1.00
CG	0.86	0.86	0.86	0.86	0.86
FT	0.84	0.81	0.81	0.81	0.81
IS	0.96	0.93	0.95	0.95	0.95
LU	0.95	0.95	0.95	0.95	0.95
MG	0.92	0.90	0.90	0.90	0.90

ブが 4MB のメッセージで通信していた場合に、バックグラウンドジョブの影響を最も強く受けていたが、今回は、バックグラウンドで 4MB のメッセージ通信していた時に最も影響を受けている（ただし、フォアグラウンドが 1KB のメッセージ通信をおこなっていた場合を除く）。バックグラウンドが 512KB の時だけ他の場合に比べ影響が特異的に少ないが、この現象については今だ説明できていない。

表 7 MPI 通信性能の干渉（FG:Comm/Phi、BG:Comm/Xeon）

	1KB	8KB	64KB	512KB	4096KB
BW [MB/s]	110	192	1054	1818	3589
1KB	1.01	0.97	1.01	1.02	1.02
8KB	0.97	0.84	0.83	0.81	0.83
64KB	0.91	0.80	0.80	0.81	0.81
512KB	1.01	0.91	0.90	0.90	0.91
4096KB	0.91	0.65	0.66	0.63	0.64

参考として、Xeon と Phi の MPI の性能を Intel MPI Benchmarks[7] の pingpong で計測した結果を図 3 に示す。Phi の場合のバンド幅の最大は 2GB/s 程度しかない。これは、表 3 と表 7 のバックグラウンドジョブがない時のバンド幅を比べても、Phi のバンド幅がかなり少ないことが分かる。この特性は、Xeon と Phi を結ぶ PCIe バスの問題 [10] と Phi のコア性能に起因すると考えられる。

先の表 4 同様に、Phi 上の NPB プログラムが Xeon 上のバックグラウンドジョブにどれだけ影響を受けるかについても実験をおこなったが、もともと Phi 上の実行が Xeon の場合に比べてバラツキが 1 割程度と大きく、その誤差を考慮すると有意とは言えないデータしか得ることができなかった。

表 8 は Xeon と Phi のぞれぞれにおける NPB の性能値（Mop/s total）である。この表から分かるように Phi の性能は Xeon に比べ数倍遅い（ただし、IS に関しては表 2 に示したように問題サイズが違うため直接は比較できない）。Phi においては、MPI 通信性能の低さ（図 3）、アプリケーションの絶対性能の遅さ（表 8）の 2 つの理由から、Xeon 上のジョブの影響を大きく受けなかったものと考えられる。

表 9 と表 10 に、通信ジョブと NPB の性能干渉について計測した結果を示す。特に表 10 において、バックグラウンドジョブとして NPB を走らせた時に、バックグラウンドジョブがない時よりも性能が 5 割程度向上しているケースがある（EP、メッセージサイズが 64KB の時）。

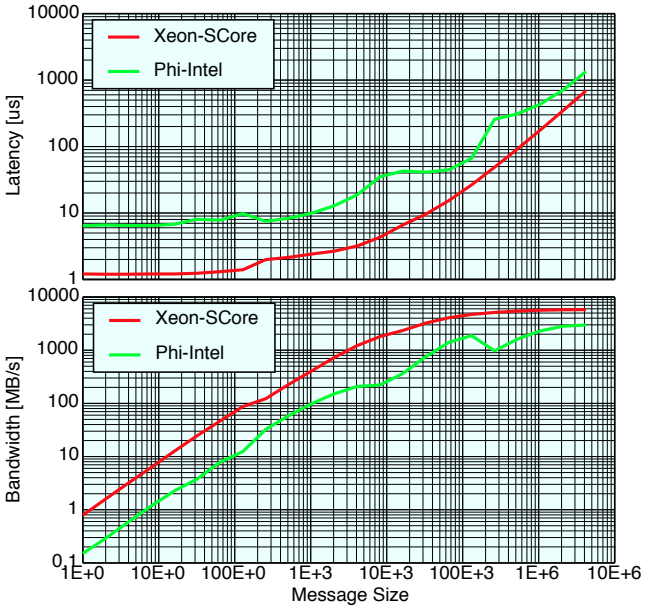

図 3 Intel MPI Benchmark, PingPong 性能の比較

表 8 NPB の性能 (MOP/s Total)

	EP	CG	FT	IS	LU	MG
Xeon	5217	35045	66457	2586	146651	128991
Phi	2258	9470	23285	60	48875	79432

表 9 通信ジョブと NPB の性能干渉率（FG:NPB/Phi, BG:Comm/Xeon）

	EP	CG	FT	IS	LU	MG
1KB	1.00	1.01	0.98	1.04	1.00	1.00
8KB	1.00	0.95	0.91	0.95	1.00	0.98
64KB	1.00	1.00	0.97	1.02	1.00	0.99
512KB	1.00	0.97	0.93	0.97	1.00	0.98
4096KB	1.00	0.95	0.91	0.94	0.99	0.97

表 10 通信ジョブと NPB の性能干渉率（FG:Comm/Phi, BG:NPB/Xeon）

	1KB	8KB	64KB	512KB	4096KB
EP	1.01	1.13	1.47	0.97	0.99
CG	0.98	0.96	1.29	0.90	0.86
FT	0.97	0.92	1.24	0.85	0.80
IS	0.91	0.85	1.16	0.82	0.77
LU	0.98	1.11	1.41	0.97	0.95
MG	0.99	1.06	1.40	0.95	0.92

この現象を調べるために、Phi の IMB の pingpong の性能を、Xeon 上に NPB のバックグラウンドジョブを走らせた状態で計測した結果を図 4 に示す。上のグラフでは対数スケールのため差が分かり難いため、下のグラフに一部を拡大し、Y 軸を線形スケールとして示してある。Xeon 上に NPB を走らせた時に、Phi 上の IMB ベンチマークの性能が 5 割程度（メッセージサイズが 16KB の時、NPB

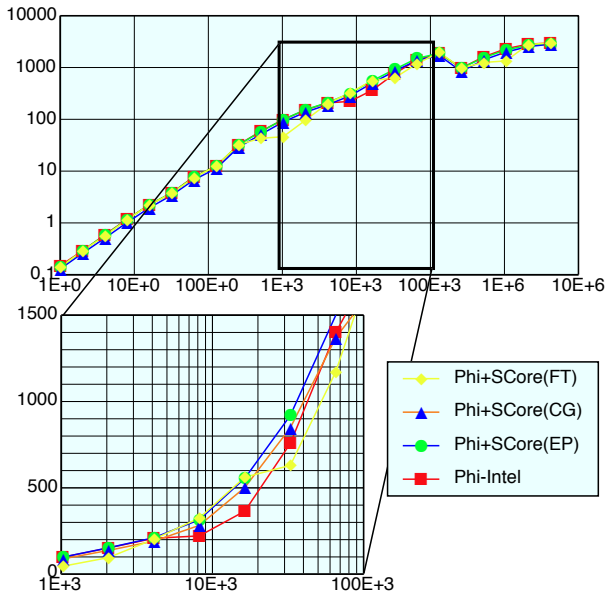

図 4 Intel MPI Benchmark, PingPong 性能の比較（バックグラウンドジョブがあった場合）

の FT をバックグラウンドジョブとして実行した時）良くなっている。恐らく、これと同じ現象が表 10 の計測時にも起きていると考えられる。

特に奇妙なのは、バックグラウンドジョブとして EP を走らせている時でも Phi の性能が向上する点である。EP においては通信はほとんど発生しないにも関わらず Xeon の影響を受けていることが、表 10 からも図 4 からも見て取れる。

2.2 マルチコアジョブ間の性能干渉

本節では、マルチコア CPU 上に 2 つのジョブを走らせた場合の性能干渉についての実験結果を示す。表 11 は Xeon 上に、NPB をフォアグラウンドジョブとし、通信プログラム（ノード毎に 2 プロセス）を走らせた結果である。本実験に用いた計算ノードには Xeon が 2 ソケットあるので、それぞれの CPU　ソケットをジョブに割り当てた。

表 11 NPB と通信ジョブの性能干渉率（FG:NPB/Xeon、BG:Comm/Xeon）

	EP	CG	FT	IS	LU	MG
1KB	1.0	1.01	0.98	1.04	1.00	1.00
8KB	1.00	0.95	0.91	0.95	1.00	0.98
64KB	1.00	1.00	0.97	1.02	1.00	0.99
512KB	1.00	0.97	0.93	0.97	1.00	0.98
4096KB	1.00	0.95	0.91	0.94	0.99	0.97

結果は、最大で 9% の性能低下（NPB:CG、通信メッセージ長 4,096KB）であった。IS の時に、バックグラウンドジョブがない時よりも性能が良い結果が得られているが、この理由が定かではないが、再現性があることを確認している。

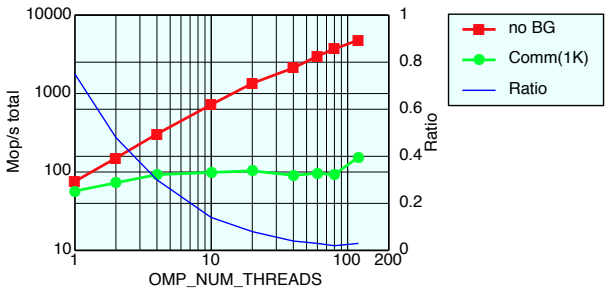

図 5 OpenMP-CG.A ジョブと通信ジョブ（1KB）との性能干渉

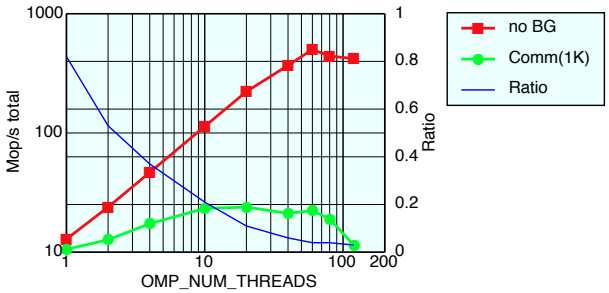

図 6 OpenMP-IS.A ジョブと通信ジョブ（1KB）との性能干渉

2.3 メニーコアジョブ間の性能干渉

前節の Xeon と同様の実験をメニーコアの Phi 上でもおこなった結果が表 12 である。バックグラウンドジョブの通信メッセージ長が 1KB の時に著しい速度低下が見られたため、これ以上のメッセージ長での実験を諦めた。

表 12 NPB と通信ジョブの性能干渉率（FG:NPB/Phi、BG:Comm/Phi）

	EP	CG	FT	IS	LU	MG
1	0.04	0.02	0.05	0.03	0.04	0.04

この現象が MPI に起因するものでないことを確認するために、OpenMP 版の NPB（CG.A および IS.A）を用いて同様の実験をおこなった結果が図 5 および図 6 である。この実験では、通信プログラムは 2 ノードを用いて実行され、OpenMP プログラムはそのうちのひとつのノードで実行した。このグラフの横軸は OpenMP のスレッド数、縦軸が NPB の性能（Mop/s total）である。グラフ中 “no BG” とあるのはバックグラウンドジョブがない時の単体性能のグラフであり、”Comm(1K)” とあるのがバックグラウンドの通信プログラムが 1KB のメッセージで通信した場合の NPB の性能である。”Ratio”（右側の目盛）は、バックグラウンドジョブの干渉がない時の性能を 1 としたときの性能低下率を表している。

これらのグラフから、バックグラウンドに通信ジョブがあった場合に、OpenMP のスレッド数に比例して性能が伸びていないのが分かる。バックグラウンドジョブがあると、CG.A の性能はスレッド数に依らず一定であり、IS.A の場合は 10 スレッド以上で性能は頭打ちとなり、80 スレッドからは性能が低下している。両者とも 100 スレッド

の時の性能はバックグラウンドジョブがない場合に比べ数十倍も遅くなっている。これは表 12 の結果を裏付けるものと考えられる。

Xeon Phi の 1 次キャッシュのサイズは 32KB（データ）、2 次キャッシュのサイズは 512KB（コア当たり）である [4]。例えば、CG のクラス A では、長さ 1,400 の 1 次元配列（倍精度）を 7 つ程持っているが、これの合計は 100KB に満たない。また、通信プログラムにおいては 1KB の送受信バッファ（送信と受信で別なコア）しかない。これらの数値からは、マルチスレッド化によりキャッシュが溢れ、そのために性能が低下したとは考え難い。逆に、マルチスレッド化により、2 次キャッシュの総量が増え、速度が向上しても不思議ではない。従い、この速度の低下にはキャッシュ以外のなんらかの要素が関与しているものと考えられる。

3. 関連研究

ジョブの “co-location” に関する研究は比較的古くからおこなわれており、論文も多い。ここでは最近の主な研究成果について述べる。

Breslow らは、計算ノードを共有するように配置された走行中のジョブの性能をモニタすることで、性能の干渉を定期的に計測し、干渉が大きい場合に共有する片方のジョブの実行を一時的に中断する方式を提案し、これによりシステム全体のジョブのスループットとエネルギー消費を 10-20% 向上させることができるとしている [3]。

Raghunath らは、Infiniband をネットワークに持つ x86 クラスタにおいて、通常の並列ジョブとファイル I/O の通信スレッドが同時に走行する際の性能干渉を Infiniband が有する QoS 機能 [1] を用い、並列ジョブへの性能干渉を減らすことができたとしている [11]。

これらの手法は、本研究が対象とするようなメニーコア混在型のクラスタの場合にも適応可能と考えられる。しかしながら、本稿で調査をおこなった通常の並列アプリケーションジョブ（NPB）をメニーコア CPU とマルチコア CPU のそれぞれで走らせた場合の性能干渉は 5% 程度であった。混在型アーキテクチャの場合は、原理的にはネットワークデバイスのみ共有されている。一方、従来のマルチコア CPU のみから構成さえるクラスタでは、ネットワークデバイスだけでなく、メモリやキャッシュの共有などにより、干渉がより強く出る可能性がある。残念ながら本稿では、メニーコア CPU 上のジョブの性能干渉が想定外の結果となったため、有意な結論を導き出すことができなかった。

4. まとめと今後について

メニーコア CPU とマルチコア CPU から構成される計算ノードを持つクラスタにおいて、2 つのジョブを同時に実行した際に生じる可能性のある互いの性能干渉について調査をおこなった。このような構成では、性能干渉はネットワークデバイスの共有に起因すると考えられる。

マルチコア CPU 上の並列ジョブの性能は、メニーコア CPU 上の並列ジョブの影響を受けるが、その影響は通信が生じる場合に限定されていることが判明し、ネットワークデバイスの共有に起因するものと考えられる。NPB のような普通の並列アプリケーションでは、実行時間に対する通信時間の割合はそれほど大きくないと考えられ、計測した結果では 10%程度の性能低下にとどまった。一方、常に通信をおこなうようなジョブが混在した場合には、性能は半分程度までに低下することが判明した。また、マルチコア CPU 上で 2 つのジョブ、NPB と通信ジョブを走らせた場合の互いの干渉による性能低下も最大 6% 程度であった。このため、例えば並列アプリケーションとファイルステージングジョブを同時に走らせても大きな問題になることはないと考えられる。

一方、メニーコア CPU 上のジョブはマルチコア CPU 上のジョブの影響を、上記のマルチコア CPU の場合とは大きく異なる形で受けることが判明した。特に、メニーコア CPU 上の並列ジョブの性能は、マルチコア CPU のジョブに通信がほとんど発生しないような場合でも大きく影響を受けることが判明した。これは性能干渉はネットワークデバイスに起因するという仮定に反する。さらに、その影響は、干渉による性能低下ばかりでなく性能が向上する現象も観測された。

特筆すべきは、メニーコア CPU 上で 2 つのジョブを走らせた場合の性能低下である。通信ジョブを走らせることで、NPB ベンチマークの性能は 1/20 以下にまで低下した。一般に、スケジューリングなどで資源を共有する場合は、共有する主体の数に反比例して性能が低下する分に、共有により発生するオーバヘッドが上乗せされて性能が低下すると考えられる。もしこのオーバヘッドが非常に大きい場合は、共有するのではなく、互いに独立して実行されるべきであろう。今回のメニーコア CPU 上の 2 つのジョブによる性能干渉の度合いは、共有すべきではないという結論を導き出すものである。

今後の研究の焦点は、今回の調査では因果関係が解明できなかった部分、すなわち、

- メニーコア CPU 上の通信性能がマルチコア上のジョブの影響を受ける因果関係
- メニーコア CPU 上に複数のジョブが走行した場合、大きく性能が低下する因果関係の解明

の 2 点になろう。

謝辞 本研究は, 科学技術振興機構 (JST) の戦略的創造研究推進事業「CREST」における研究領域「ポストペタスケール高性能計算に資するシステムソフトウェア技術の創出」によるものである.

参考文献

[1] Alfaro, F., Sanchez, J. and Duato, J.: QoS in InfiniBand subnetworks, *Parallel and Distributed Systems, IEEE Transactions on*, Vol. 15, No. 9, pp. 810–823 (online), DOI: 10.1109/TPDS.2004.46 (2004).

[2] Atsuhi, H., Akio, S., Mitaro, N., Mikiko, S., Go, F., Yuishi, T. and Yutaka, I.: メニーコア用 Agent プログラミング環境の提案，*IPSJ SIG Notes*, Vol. 2013, No. 32, pp. 1–8（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009588152/en/⟩ (2013).

[3] Breslow, A. D., Tiwari, A., Schulz, M., Carrington, L., Tang, L. and Mars, J.: Enabling Fair Pricing on HPC Systems with Node Sharing, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA, ACM, pp. 37:1–37:12 (online), DOI: 10.1145/2503210.2503256 (2013).

[4] Fang, J., Varbanescu, A. L., Sips, H., Zhang, L., Che, Y. and Xu, C.: Benchmarking Intel Xeon Phi to Guide Kernel Design, Technical Report PDS Technical Report PDS-2013-005, Delft University of Technology (2013).

[5] Intel Corp.: *Intel Many Integrated Core Symmetric Communications Interface (SCIF) User Guide*, Revision 1.0 edition (2012).

[6] Intel Corp.: *Intel Xeon Phi Coprocessor System Software Developers Guide*, Revision 2.0.3 edition (2012).

[7] Intel Corp.: *Intel MPI Benchmarks – User Guide and Methodology Description*, Revision 3.2.4 edition (2013).

[8] NASA: NAS Parallel Benchmarks. http://www.nas.nasa.gov/publications/npb.html.

[9] PC Cluster Consortium: SCore. http://www.pccluster.org/.

[10] Potluri, S., Bureddy, D., Hamidouche, K., Venkatesh, A., Kandalla, K., Subramoni, H. and Panda, D. K. D.: MVAPICH-PRISM: A Proxy-based Communication Framework Using InfiniBand and SCIF for Intel MIC Clusters, *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, SC '13, New York, NY, USA, ACM, pp. 54:1–54:11 (online), DOI: 10.1145/2503210.2503288 (2013).

[11] Rajachandrasekar, R., Jaswani, J., Subramoni, H. and Panda, D. K.: Minimizing Network Contention in InfiniBand Clusters with a QoS-Aware Data-Staging Framework, *Proceedings of the 2012 IEEE International Conference on Cluster Computing*, CLUSTER '12, Washington, DC, USA, IEEE Computer Society, pp. 329–336 (online), DOI: 10.1109/CLUSTER.2012.90 (2012).

[12] Reinders, J.: An Overview of Programming for Intel Xeon processors and Intel Xeon Phi coprocessors (2012).

[13] Weinberg, J. and Snavely, A.: User-guided Symbiotic Space-sharing of Real Workloads, *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS '06, New York, NY, USA, ACM, pp. 345–352 (online), DOI: 10.1145/1183401.1183450 (2006).

[14] 宇野篤也，庄司文由，横川三津夫：ファイルステージングのあるジョブスケジューリングの評価，*IPSJ SIG Notes*, Vol. 2012, No. 22, pp. 1–6（オンライン），入手先 ⟨http://ci.nii.ac.jp/naid/110009453388/en/⟩ (2012).

[15] 宮崎博行，草野義博，新庄直樹，庄司文由，横川三津夫，渡邊貞：スーパーコンピュータ「京」の概要 (2012).

[16] 深沢　豪，佐藤未来子，吉永一美，辻田祐一，島田明男，堀　敦史，並木美太郎：メニーコア混在型並列計算機向け大域仮想アドレス空間モデル Multiple PVAS の提案，情報処理学会第 141 回ハイパフォーマンスコンピューティング研究発表会，Vol. 2013, No. 7, pp. 1–10 (2013).