非ブロッキング集団通信による通信隠蔽技術の Fujitsu FX100 における効果の検証

研究代表者： 南里 豪志 （九州大学）
共同研究者： 荻野 正雄 （名古屋大学）

プロセス並列プログラムにおける通信の標準規格である Message Passing Interface (MPI) で定義されている非ブロッキング通信は、あるプロセス間の通信と、それらのプロセス上での計算を並行して進行させることにより、見かけ上、通信に要する時間の一部を計算時間で隠蔽することを可能とする。このうち、MPI-3.0 規格において採用された非ブロッキング集団通信は、多くの並列プログラムで頻出する、複数のプロセス間でのデータ複製や集約のような定型通信である集団通信について、通信時間の隠ぺいを図るものである。集団通信は、計算機の大規模化に伴って所要時間が増大するため、並列プログラムのスケーラビリティ向上の手段として、この非ブロッキング集団通信が期待されている。非ブロッキング集団通信は、その集団通信に参加する各プロセスが計算を行っている間に、集団通信を完了させるための通信アルゴリズムを推進することで、通信時間を隠蔽するため、この、集団通信アルゴリズムの推進手法が、隠蔽効果に大きく影響する。最も一般的に用いられている推進手法は、プログレススレッドと呼ばれるスレッドを別途起動するものである。これは、計算スレッドと CPU コアを取り合うので、計算時間に影響する。一方、名古屋大学に導入されている Fujitsu PRIMEHPC FX100 では、プログレススレッド専用のアシスタントコアを用意しており、計算時間に影響しない。

そこで本研究では、これらの推進手法による非ブロッキング集団通信の通信隠蔽効果の相違を調査した。調査には、図 1 に示す簡単なプログラムにより、ハイブリッド並列プログラムにおける非ブロッキング集団通信の効果を計測した。このプログラムでは、通信量 `M` と計算量 `N` を実行時に指定し、これらのパラメータを使って、以下の時間をそれぞれ計測することにより、非ブロッキング集団通信使用の有無による並列プログラムの性能への影響の見積もりが可能となる。

`Tcomm` ： ブロッキング集団通信の所要時間

`Tnbcomm` ：非ブロッキング集団通信の所要時間

`Tcomp` ： 計算時間

```
get_param(&M, &N);
MPI_Comm_size(&procs);
ts = MPI_Wtime();
MPI_Alltoall(M);
Tcomm = MPI_Wtime() - ts;
ts = MPI_Wtime();
MPI_Ialltoall(M); MPI_Wait();
Tnbcomm = MPI_Wtime() - ts;
ts = MPI_Wtime();
do_comp_S(N, procs);
Tcomp = MPI_Wtime() - ts;
ts = MPI_Wtime();
MPI_Alltoall(M);
do_comp_S(N, procs);
Tblock = MPI_Wtime() - ts;
ts = MPI_Wtime();
MPI_Ialltoall(M);
do_comp_S(N, procs);
MPI_Wait();
Tovlp = MPI_Wtime() - ts;
```

図 1: ハイブリッド並列プログラムにおける非ブロッキング集団通信の効果を計測するベンチマークプログラム (Alltoall 版))

`Tblock` ： ブロッキング集団通信と計算を連続して実行した場合の所要時間

`Tovlp` ：非ブロッキング集団通信と計算を並行して実行した場合の所要時間

図 1 から呼ばれる計算関数 `do_comp_`S のプログラムを図 2 に示す。関数名、および関数内の `schedule` 節における S には、OpenMP におけるスケジューリングポリシーを記述する。今回の計測では、static と dynamic、それぞれのスケジューリングポリシーについて計測する。これは、特に Fully Subscribed において、スケジューリングポリシーによる性能の変化を確認するためである。

実験に使用した計算機環境は、九州大学情報基盤研究開発センターの Fujitsu PRIMERGY CX400 と、名古屋大学情報基盤センターの Fujitsu PRIMEHPC FX100 である。

```
void do_comp_S(int N, int procs)
{
  int nn = N / procs;
#pragma omp parallel for private(j,k) schedule(S)
  for (i = 0; i < nn; i++)
    for (k = 0; k < N; k++)
      for (j = 0; j < N; j++)
        c[i*N+j] += a[i*N+k] * b[k*N+j];
}
```

図 2: do_comp_S 関数

Fujitsu PRIMERGY CX400 では、MPI ライブラリとして MVAPICH2 2.2 を、C コンパイラとして GCC を、それぞれ使用する。

MVAPICH2 では、プログレススレッドを有効にするための環境変数として、以下を指定する。

```
MV2_SMP_USE_CMA=0
MV2_ENABLE_AFFINITY=0
MV2_ASYNC_PROGRESS=1
```

一方、Fujitsu PRIMEHPC FX100 では、MPI ライブラリ、C コンパイラとも、富士通社製のものを使用する。この MPI ライブラリでは、mpiexec コマンドに以下のオプションを追加することにより、アシスタントコア上に非ブロッキング集団通信用のプログレススレッドを起動し、利用できる。

```
--mca opal_progress_thread_mode モード番号
```

ここでモード番号とは、プログレススレッドの動作モードを指定するもので、以下から選択する。

mode 1 指定した区間のみプログレススレッドが動作する。区間内では MPI 関数を呼び出すことが出来ない。

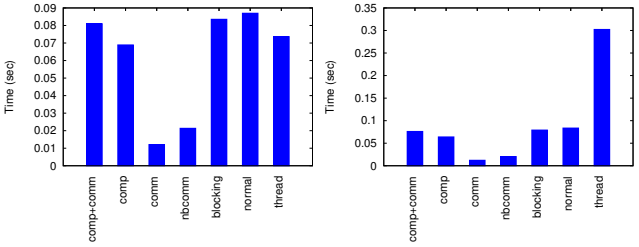
mode 2 指定した区間のみプログレススレッドが動作する。区間内で MPI 関数を呼び出すことが出来る。

mode 3 非ブロッキング集団通信が呼ばれると自動的にプログレススレッドが動作する。

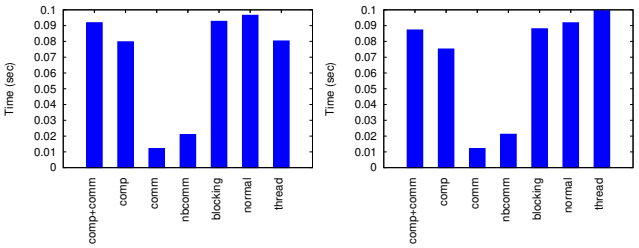
mode 1 および mode 2 の場合、図 1 の MPI_Ialltoall と MPI_Wait に挟まれた do_comp_S の呼び出し部分をプログレススレッドの動作区間として指定する。

Fujitsu PRIMERGY CX400 で Alltoall 関数によるベンチマークプログラムの性能を計測した結果を、図 3 に示す。通信量と計算量のパラメータはそれぞれ M=131072 および N=1024 としている。

一方、Fujitsu PRIMERGY CX400 で Alltoall 関数によるベンチマークプログラムの性能を計測した結果を、



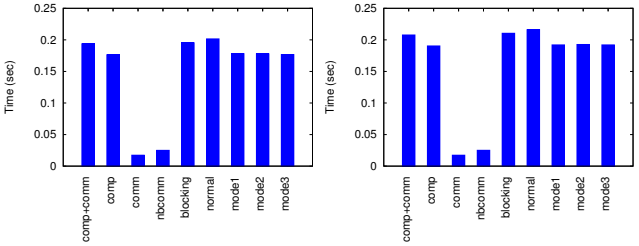
(a) Spare, Static (b) Full, Static



(c) Spare, Dynamic (d) Full, Dynamic

図 3: Alltoall with MVAPICH2 (32nodes, 32procs, M=131072, N=1024)

図 4 に示す。通信量と計算量のパラメータはそれぞれ M=131072 および N=1536 としている。



(a) Static (b) Dynamic

図 4: Alltoall on Fujitsu PRIMEHPC FX100 (48nodes, 48procs, M=131072, N=1536)

平成 29 年度名古屋大学 HPC 計算科学連携研究プロジェクト 成果

南里豪志, 大島聡史, 小野謙二, 非ブロッキング集団通信の通信隠蔽効果に関する調査, 第 162 回ハイパフォーマンスコンピューティング研究会, 2017.