2008－HPC－115 (3)
2008／5／20

ハイブリッド並列化したIDR(s)法の計算時間に対するプロセス数とスレッド数の組み合わせ依存性について

[1)] 馬場 慎也, [1)] 尾上 勇介, [2)] 南里 豪志, [2)] 藤野 清次
[1)] 九州大学大学院, [2)] 九州大学情報基盤研究開発センター

MPI と OpenMP のハイブリッド並列モデルで並列化されたプログラムの性能は, 使用する CPU コア数が同じであっても, 選択されたプロセス数とスレッド数の組み合わせに依存する. 本稿では, 連立一次方程式解法の一つである IDR(s) 法を対象として, ハイブリッド並列化手法の性能評価を行い, 最適なプロセス数とスレッド数の組み合わせについての考察を行う.

Dependence on combination with number of processes and threads for computation times of hybrid-parallel version of IDR(s) Method

[1)]SHINYA BABA, [1)]YUSUKE ONOUE, [2)]TAKESHI NANRI, [2)]SEIJI FUJINO
[1)]Kyushu University, [2)]Reseach Institute for Information Technology, Kyushu University

The performance of hybrid-parallel programs with MPI and OpenMP depends on the chosen combination of the numbers of processes and threads even on the same number of CPU cores. In this study, the performance of IDR(s) method, an iterative method for sparse linear systems, parallelized with hybrid is evaluated. Also, the dependency of the performance on the numbers of processes and threads are discussed.

1 はじめに

近年, 科学技術計算を行う計算機環境として, 共有メモリ型の並列計算機をネットワークで接続した階層的な並列計算機が広く用いられる. このようなアーキテクチャの特性を活かすために, 分散メモリ型の並列化手法と共有メモリ型の並列化手法を組み合わせたハイブリッド並列化手法が用いられる. ハイブリッド並列化手法の性能は, 使用するプロセス数とスレッド数の組み合わせに依存する. そのため, より良い性能を達成するためには, 計算機環境やプログラム, データの特性に応じた組み合わせの選択が必要である.

階層的な並列計算機が普及するにつれ, スレッド並列化を実行する環境が身近になり, ハイブリッド並列化の研究も盛んである. 陳らの研究 [2] では, ハイブリッド並列化した行列の固有値解法の性能評価を行っている. この研究では, ハイブリッド並列化することで計算速度を維持したまま, メモリ使用量を減らしている. また, 中島の研究 [5] では, ハイブリッド並列化と MPI(Message Passing Interface)[1, 7] のみを用いた Flat MPI 型の並列化の性能比較を行っている. この研究では, 地球シミュレータ上で前処理付き CG 法の性能評価を行い, 使用するノード数が小さい場合は Flat MPI 型が有効であるが, ノード数が大きくなるとハイブリッド並列化の方が効果的であるという結果が出た.

本稿では新しく開発された連立一次方程式解法の一つである IDR(s) 法を対象とする. IDR(s) 法をハイブリッド並列化し, 様々なプロセス数とスレッド数の組み合わせによる性能の変化を調査する. また, 複数の並列計算機で実験を行い, その特性の違いについて考察する.

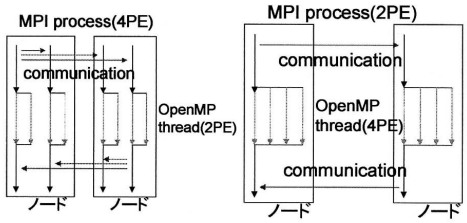
2 並列化手法について

近年, 特にマルチコア CPU を搭載した PC のような共有メモリ型並列計算機の普及に伴い, 分散メモリ型並列計算機の計算ノードに共有メモリ型並列計算機を用いる階層型の並列計算機が用いられる.

階層型の並列計算機における並列化手法として, 分散メモリ型とハイブリッド型がある. このうち分散メモリ型とは, 独立したメモリ空間を持つプロセス単位で並列処理を行う並列化手法である. この手法の事実上の標準として MPI が利用されている. 一方, ハイブリッド型とは, 分散メモリ型並列化手法における各プロセスの処理をさらに複数のスレッドに分割して並列化処理を行うものである. スレッドに分割する手段としては, 主に自動並列化, もしくは OpenMP[9]

が用いられる．特に OpenMP は自動並列化に比べて，指示文を挿入することでより詳細な並列化が可能であり，MPI と比べても並列化が容易である．

階層型の並列計算機上で分散メモリ型の並列化手法を用いる場合，階層型でない分散メモリ型並列計算機と同じプログラムを利用できるという利点がある．しかし，全 CPU でプロセスが実行するため，プロセス間通信の影響が大きく，効率的な並列化が行えない可能性がある．一方，ハイブリッド型には，スレッド並列化を行うことでプロセス間通信のコストを削減することができ，プログラムの所要時間を減らせるという利点がある．図 1 に 2 ノード，8 コア使用した場合のハイブリッド並列化の例を示す．図 1(a) が 4 プロセス，2 スレッドで並列化したもの，図 1(b) が 2 プロセス，4 スレッドで並列化したものである．スレッド数を増やすことで，ノードをまたいだプロセス間通信を少なくできる．



(a) 4 プロセス× 2 スレッド　(b) 2 プロセス× 4 スレッド

図 1: ハイブリッド並列化におけるプロセス数とスレッド数の組み合わせの例

3　IDR(s) 法の並列化

3.1　IDR(s) 法について

IDR(s) 法 [6, 8] は反復法の一つである．反復法とは，大規模で疎な行列を係数行列に持つ連立一次方程式の解法の一つである．IDR(s) 法は，計算時間について，従来から用いられている BiCGSafe 法 [3] などの反復法と遜色がない程度に優れている．また，行列の性質によっては，既存の解法に比べて収束性の良さが伺える．

IDR(s) 法には，従来の反復法にも含まれる疎行列・ベクトル積や内積などの計算に加えて，密行列の積などの従来とは異なる種類の計算が含まれているのが特徴である．本研究ではこの特徴に注目し，ハイブリッド並列化した IDR(s) 法の性能評価を行うことにした．ちなみに，IDR(s) 法の並列化の研究として，OpenMP によるスレッド並列化については [4] で報告されているが，ハイブリッド並列化に関する研究は報告されていない．

以下に IDR(s) 法の 1 回の反復に含まれる主な計算を示す．このうち，(1) と (2) は一重ループの計算，(3) と (4) は二重ループの計算である．

(1) ベクトルの加減算

(2) ベクトルの内積

(3) 密行列・ベクトル積

(4) 疎行列・ベクトル積

以下に IDR(s) 法のアルゴリズムを示す．アルゴリズム中の記号は [6] に準ずる．上記の四つの計算に対応する部分に (1) から (4) と記す．

IDR(s) 法のアルゴリズム

Let $\boldsymbol{x}_0$ be an initial guess, and put $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
  For $n = 0, \ldots, s-1$ Do
    $\boldsymbol{v}_n = A\boldsymbol{r}_n$
    $\omega_n = \dfrac{(\boldsymbol{v}_n, \boldsymbol{r}_n)}{(\boldsymbol{v}_n, \boldsymbol{v}_n)}$
    $\boldsymbol{q}_n = \omega_n \boldsymbol{r}_n, \quad \boldsymbol{e}_n = -\omega_n \boldsymbol{v}_n,$
    $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \boldsymbol{e}_n, \ \boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{q}_n$
  End Do
  $E_s = (\boldsymbol{e}_{s-1} \cdots \boldsymbol{e}_0), \ \ Q_s = (\boldsymbol{q}_{s-1} \cdots \boldsymbol{q}_0)$
  Do $n = s, s+1, \ldots$
    Solve $\boldsymbol{c}_n$ from $P^T E_n \boldsymbol{c}_n = P^T \boldsymbol{r}_n$
    $\boldsymbol{v}_n = \boldsymbol{r}_n - E_n \boldsymbol{c}_n \quad (1), (3)$
    If $\mathrm{mod}(n, s+1) = s$ then
      $\boldsymbol{t}_n = A\boldsymbol{v}_n \quad (4)$
      $\omega_n = \dfrac{(\boldsymbol{t}_n, \boldsymbol{v}_n)}{(\boldsymbol{t}_n, \boldsymbol{t}_n)} \quad (2)$
      $\boldsymbol{e}_n = -E_n \boldsymbol{c}_n - \omega_n \boldsymbol{t}_n \quad (1), (2), (3)$
      $\boldsymbol{q}_n = -Q_n \boldsymbol{c}_n + \omega_n \boldsymbol{v}_n \quad (1), (2), (3)$
    Else
      $\boldsymbol{q}_n = -Q_n \boldsymbol{c}_n + \omega_n \boldsymbol{v}_n \quad (1), (2), (3)$
      $\boldsymbol{e}_n = -A\boldsymbol{q}_n \quad (4)$
    End If
    $\boldsymbol{r}_{n+1} = \boldsymbol{r}_n + \boldsymbol{e}_n \quad (1)$
    $\boldsymbol{x}_{n+1} = \boldsymbol{x}_n + \boldsymbol{q}_n \quad (1)$
    if $||\boldsymbol{r}_{n+1}||_2 / ||\boldsymbol{r}_0||_2 \leq \epsilon$ then stop
    $E_n = (\boldsymbol{e}_{n-1} \cdots \boldsymbol{e}_{n-s}), \ Q_n = (\boldsymbol{q}_{n-1} \cdots \boldsymbol{q}_{n-s})$
  End Do

3.2　IDR(s) 法のハイブリッド並列化

前節に示した四つの計算それぞれについて，その中の一つのループを対象にハイブリッド並列化する．まず，プロセス

並列化では, 各プロセスが各自に割り当てられた繰り返しのみを実行するようにループを書き換える. さらにそのループに OpenMP の指示文を挿入し, スレッド並列化する.

二重ループである密行列・ベクトル積と疎行列・ベクトル積については, ループ長が長い方のループを並列化の対象とする. まず, ハイブリッド並列化した密行列・ベクトル積を Algorithm(1) に示す. ここで, istart と iend は各プロセス毎に分割したループの開始と終了位置である. Algorithm(1) において, 外側のループ長 s は IDR(s) 法のパラメータであり, 一般に用いられる値は高々10 である. 一方, 内側ループの並列化前のループ長は計算対象となる行列の次元数であり, 一般に数千以上の値となるため, 内側ループを並列化の対象とする.

一方, ハイブリッド並列化した疎行列・ベクトル積を Algorithm(2) に示す. Algorithm(2) では, 並列化前の外側のループ長が次元数であるのに対して, 内側ループのループ長は各列の非零要素数である. 疎行列では非零要素数が次元数に対して非常に少ないため, ハイブリッド並列化の対象として外側のループを選択する.

Algorithm(1) 密行列・ベクトル積のハイブリッド並列化

```
do i = 1, s
  tmp = 0.0d0
  !$ omp parallel do reduction(+ : tmp)
  do j = istart, iend
    tmp = tmp + A(i,j) * c(j)
  end do
  b(i) = tmp
end do
```

Algorithm(2) 疎行列・ベクトル積のハイブリッド並列化

```
call MPI_ALLREDUCE(p, ptmp, ncol, ···)
!$ omp parallel do private(i, j, temp)
do i = istart, iend
  temp = 0.0d0
  do j = colptr(i), colptr(i + 1) − 1
    temp = temp + val(j) * ptmp(rowind(j))
  end do
  y(i) = temp
end do
```

上記のハイブリッド並列化に伴い, 3種類のプロセス間通信を挿入する. まず, 反復計算に入る前に, MPI_BCAST を用いて計算に用いる値を全プロセスに渡す. 次に, スカラー値である ω_n や収束判定値を求めるために MPI_ALLREDUCE を用いている. 三つ目に, 反復計算内では, 疎行列・ベクトル積に用いるベクトル p が各プロセスで分散して計算されるため, 全プロセスで値を共有するために MPI_ALLREDUCE を用いる. MPI_ALLREDUCE における転送データのサイズは行列の次元数と同じである. なお, MPI_ALLGATHER ではなく MPI_ALLREDUCE に用いているのは, 最後のプロセスに割り当てられる部分ベクトルだけ大きさが異なるためである.

4 数値実験

4.1 計算機環境と計算条件

数値実験は, 九州大学情報基盤研究開発センターに設置された日立 SR11000(以下, SR と記述) と FUJITSU PRIMEQUEST 580(以下, PQ) と FUJITSU PRIMERGY RX200S3(下, PG) で行った. 各計算機は共有メモリ型の並列計算機のクラスタである. 表 1 に計算機環境を示す.

表 1: 計算機環境

計算機名	SR	PQ	PG
CPU	IBM POWER5	Itanium2	Xeon
クロック周波数	1.9GHz	1.6GHz	3.0GHz
ノード内コア数	16	64	4
使用コア数	64	256	128
ネットワーク	専用クロスバーネットワーク	InfiniBand	InfiniBand

プログラムは Fortran90 で実装し, 計算は全て倍精度浮動小数点演算で行った. コンパイラは SR では日立最適化 Fortran90, PQ, PG では Fujitsu Fortran Compiler を使用した. 最適化オプションは, SR では-64 -Oss -omp を使用し, PQ と PG では-Kfast,OMP を使用した.

反復計算における計算条件を以下に示す. 初期近似解 $\boldsymbol{x}_0$ は全て零とし, 最大反復回数は次元数と同じにした. また, 今回の実験では, IDR(s) 法のパラメータ s として 3 を選択した. これはほとんどの場合でこの値により, 高速に計算できることが経験的に分かっているためである.

実験では, スレッド数, プロセス数どちらも 1, 2, 4,··· のように 2 のべき乗で, スレッド数が表 1 に示す各計算機のノード内コア数以下, かつプロセス数 × スレッド数が使用コア数以下となる全ての組み合わせについて計算時間を測定した. また, それぞれの組み合わせで計測を三回行い, その平均を計測結果として用いた. 実験では, Florida 大学が公開している疎行列データベース [10] にある行列 XENON1, XENON2 を入力データとして使用した.

表 2: 実非対称行列の主な特徴

行列名	解析分野	次元数	非零要素数	平均非零要素数
XENON1	sodelite crystals	48,600	1,181,120	24.30
XENON2	同上	157,464	3,866,688	24.56

4.2 実験結果

以下に各計算機での実験結果を示す．ここで，使用したプロセス数とスレッド数の組み合わせを（プロセス数 × スレッド数）と表す．

表 3 に各計算機での行列 XENON1, XENON2 における IDR(s) 法の所要時間を示す．また，行列 XENON1 について，図 2 に IDR(s) 法における各計算機での通信時間と計算時間の図を示す．ここで，通信時間とは，MPI_BCAST 及び MPI_ALLREDUCE に要した時間である．一方，計算時間とは，所要時間から通信時間を除いた時間のことである．

まず行列 XENON1 について，各計算機で最も高速だったプロセス数とスレッド数の組み合わせは，SR で（1×16），PQ で（4×8），PG で（8×4）であった．また，行列 XENON2 でもほぼ同じだが，PQ では（2×16）が最も高速であった．使用コア数が一定の場合には，SR, PG ではできるだけスレッド数を増やした方が高速となった．一方，PQ ではプロセス数とスレッド数を適度に組み合わせた方が高速であった．

また，(1×1), (1×2) のようにプロセス数を固定してスレッド数を増やした場合の傾向を見ると，SR のノード内 (16 並列以下) での並列化では，所要時間は減少する．しかし，ノードをまたぐと (32 並列以上)，必ずしも減少するとは限らない．PQ では，ノード内 (64 並列以下) での並列化でも，あるスレッド数以上で性能向上が見られなくった．PG では，ノード内外での並列化で，スレッド数を増やすことでほぼ所要時間は減少する．

一方，(1×1), (2×1) のようにスレッド数を固定してプロセス数を増やした場合は，SR のノード内 (16 並列以下) での並列化では，ほぼ所要時間が減少する．しかし，ノードをまたぐと (32 並列以上)，並列化の効果が見られなくなる．PQ では，ノード内 (64 並列以下) での並列化でも，ある程度プロセス数を増やすと通信時間が増え，所要時間は遅くなる．PG では，ノードをまたいでも (8 並列以上)，少ないながらも性能向上が見られるが，プロセス数 16 以上で逆に遅くなる．

5 考察

プロセス数とスレッド数の組み合わせに対する性能の傾向は，SR, PG ではほぼ同様であり，使用コア数が一定であれば，できるだけスレッド数を増やした場合が一番高速で

表 3: IDR(s) 法の所要時間 [秒]

(a) 計算機 SR, XENON1

		スレッド数				
		1	2	4	8	16
プロセス数	1	11.57	6.91	3.66	2.13	**1.58**
	2	7.11	3.92	2.56	1.96	2.04
	4	3.93	2.71	2.03	3.17	2.59
	8	2.85	2.47	3.08	3.79	
	16	2.93	3.50	4.05		
	32	4.67	4.84			
	64	5.14				

(b) 計算機 SR, XENON2

		スレッド数				
		1	2	4	8	16
プロセス数	1	48.39	24.48	16.57	8.78	**6.43**
	2	25.83	14.98	9.72	7.70	7.83
	4	17.20	9.00	7.34	11.45	8.77
	8	10.12	10.52	11.54	13.24	
	16	10.53	12.00	15.56		
	32	14.00	14.49			
	64	17.22				

(c) 計算機 PQ, XENON1

		スレッド数						
		1	2	4	8	16	32	64
プロセス数	1	23.67	13.96	10.10	9.61	8.90	8.70	12.30
	2	12.04	8.23	6.71	6.06	6.04	6.81	11.30
	4	8.33	7.72	5.65	**5.25**	5.42	8.25	13.37
	8	9.74	7.49	5.36	5.51	7.18	10.18	
	16	12.56	8.44	6.63	7.75	10.09		
	32	16.32	11.94	9.55	11.92			
	64	21.61	17.37	13.91				
	128	35.03	22.93					
	256	51.29						

(d) 計算機 PQ, XENON2

		スレッド数						
		1	2	4	8	16	32	64
プロセス数	1	132.02	81.88	42.13	34.09	33.97	28.97	36.98
	2	72.06	48.76	36.74	22.93	**19.82**	24.69	38.56
	4	68.04	40.13	23.71	23.42	21.68	39.16	45.33
	8	88.73	40.78	27.18	23.77	42.93	44.99	
	16	76.37	49.94	36.65	33.05	52.59		
	32	87.25	48.07	44.04	52.05			
	64	135.88	72.58	80.53				
	128	163.67	108.37					
	256	313.97						

(e) 計算機 PG, XENON1

		スレッド数		
		1	2	4
プロセス数	1	19.43	20.00	10.69
	2	17.75	10.73	6.49
	4	15.90	8.01	5.42
	8	12.76	6.79	**5.21**
	16	14.25	7.61	5.87
	32	19.48	11.73	10.28
	64	27.22	15.26	
	128	31.15		

(f) 計算機 PG, XENON2

		スレッド数		
		1	2	4
プロセス数	1	95.57	84.96	49.07
	2	97.75	58.40	32.78
	4	81.13	38.56	26.46
	8	69.49	36.56	**25.16**
	16	77.80	44.01	28.03
	32	91.11	43.97	33.51
	64	106.92	57.79	
	128	125.52		

あった. これは, スレッド数を増やすことで MPI 通信時間を減らすことができるためである. しかし, PQ では他の二機種と異なり, (4×8) のようにプロセス数とスレッド数を適度に組み合わせると高速になるという傾向が見られた. この理由としては, スレッド数を増やすことで, MPI 通信時間を減らすことができる反面, メモリを共有しているため, キャッシュのヒット率が悪くなったためだと考えられる. この傾向は図 2 で計算時間がスレッド数に応じて増加することでも確認できる.

また, ノードをまたぐことで並列化の効果が見られなくなる場合があるが, この傾向は, ノード内の MPI 通信とノード間の MPI 通信の性能の違いによると考えられる.

また, PQ におけるスレッド数と通信時間の関係として, 例えばプロセス数が 1 の場合でもスレッド数を増やすと通信時間の割合が増えるなど, 原因不明の現象が見られる. この点については現在調査中である.

二つの行列の間では, ほとんど性能の傾向に違いが見られなかった. 計算時間に注目すると, 疎行列・ベクトル積の計算に多くの時間がかかっており, この計算時間は行列の非零要素数に依存している. また, MPI 通信時間は通信するデータのサイズ, つまり行列の次元数に依存している. 表 2 からも分かるように, 今回実験に用いた行列ではどちらも次元数に対する非零要素数の比率が同じであるため, 同じ傾向になったと考えられる. 今後は, 異なる比率の行列の実験が必要である.

6 まとめと今後の課題

本稿では, ハイブリッド並列化した IDR(s) 法について, プロセス数とスレッド数の組み合わせを変えて, その性能評価を行った. そして, その組み合わせがプログラムの所要時間に与える影響を観測した. 計算機 SR と PG では, できるだけスレッド数を増やすことで所要時間を少なくできる. 一方, PQ では適度にプロセス並列化とスレッド並列化を組み合わせることで高速化できることが分かった. また, 使用コア数を増やしても, ノードをまたぐと MPI 通信時間が増大し, 並列化の効果が得られなくなることが分かった.

今後は, 計算機の特性や行列の特徴を意識して並列化手法の性能評価を行っていく. また, プロセス並列化によるキャッシュのヒット率の向上やスレッド並列化によるアクセス競合に関しても調査を行っていく. そこから, ハイブリッド並列化における最適なプロセス数とスレッド数を自動選択するための評価指標の確立を目指す.

参考文献

[1] 青山幸也: 並列プログラミング入門 MPI 版, 理研スパコン・システム講習会配布テキスト, 2007.

[2] 陳華偉, 仲田晋, 石原昌文, 山田進, 萩原一郎: ハイブリッド記述による固有値解析の並列処理の性能評価, Transactions of JSCES, Paper No.20040014.

[3] 藤野清次, 藤原牧, 吉田正浩: 準残差の最小化に基づく BiCGSafe 法の収束性について, Trans. of JSCES, Paper No.20050028, 2005.

[4] 草場健一郎, 藤野清次, 尾上勇介: 行列のブロックサイクリック分割に基づく IDR(s) 法の並列性評価, 情報処理学会研究報告, 2008-HPC-114, pp.193-198, 北海道大学, 2008.

[5] K. Nakajima: "OpenMP/MPI Hybrid vs. Flat MPI on the Earth Simulatar: Parallel Iterative Solvers for Finite Element Method", Lecture Notes in Computer Science 2858, pp.486-499, Springer, 2003.

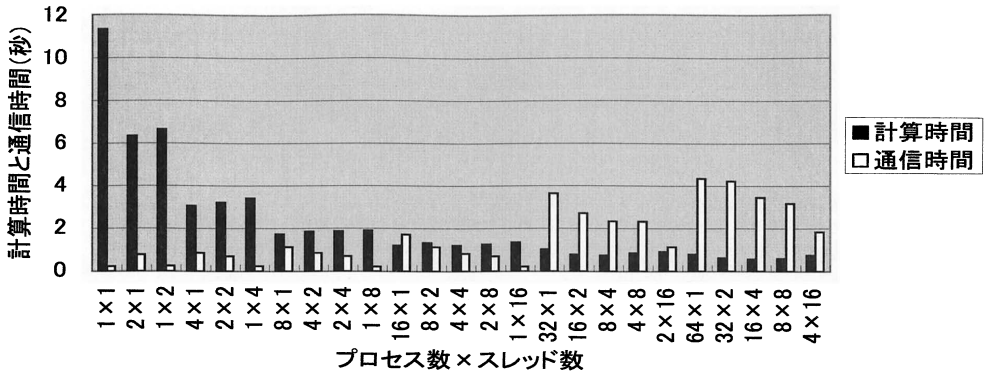
[6] 尾上勇介, 藤野清次, 中嶋徳正: IDR(s) 法の算法の概要と収束性評価, 日本応用数理学会 2007 年度講演予稿集, pp.126-127, 北海道大学, 2007.

[7] P. Pacheco, 秋葉博: MPI 並列プログラミング, 培風館, 2001.

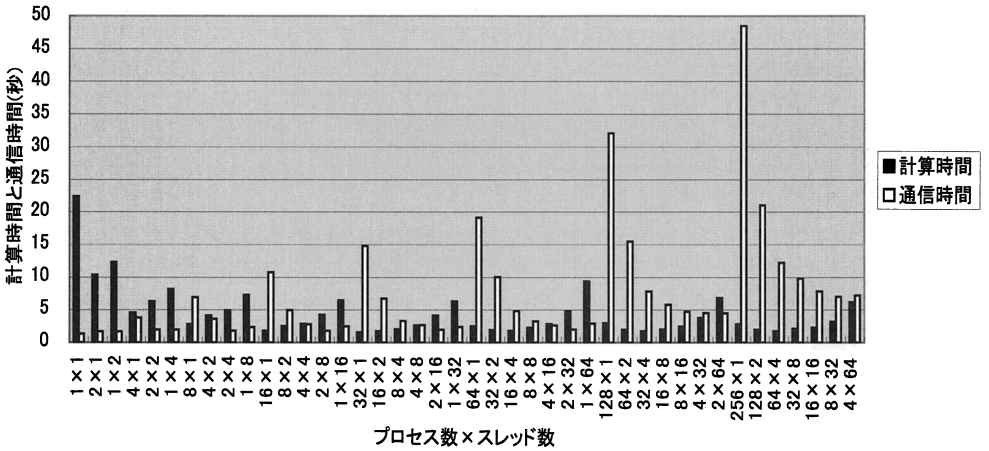
[8] P. Sonneveld and M.B. van Gijzen, "IDR(s): a family of simple and fast algorithms for solving large nonsymmetric systems of linear equations",Tecnical Report 07-07,Department of Mathmatical Analysis, Delft, The Netherlands, 2007.

[9] 牛島省: OpenMP による並列プログラミングと数値計算法, 丸善, 2006.

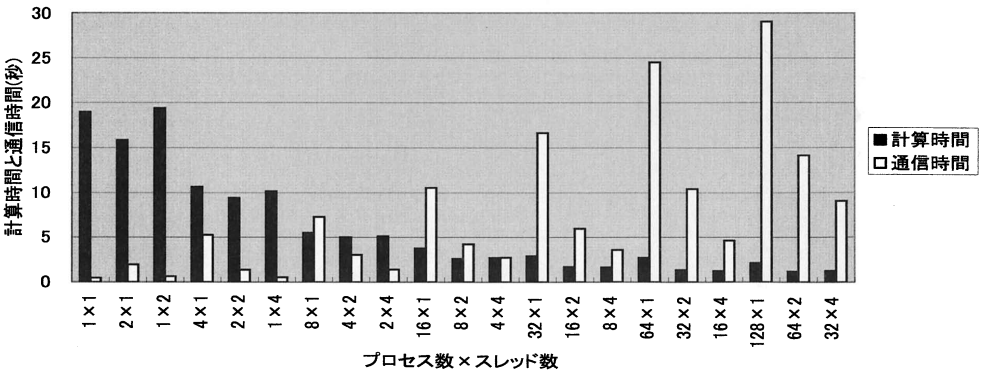
[10] University of Florida Sparse Matrix Collection: http://www.cise.ufl.edu/research/sparse/matrices/index.htm



(a) SR



(b) PQ



(c) PG

図 2: 行列 XENON1 における各計算機での計算時間と通信時間 [秒]