グリッド環境におけるScaLAPACKのための GAによるスケジューリング

廣安 知之[†], 三木 光範[†], 斉藤 宏樹[††], 谷村 勇輔[††], Jack Dongarra [‡]

[†] 同志社大学工学部 [††] 同志社大学大学院

[‡] University of Tennessee/Oak Ridge National Laboratory

グリッド上の計算資源を利用して，効率的に分散・並列アプリケーションを実行するためには，アプリケーションの特性に合わせて，適切な計算資源を選び出す必要がある．本研究では，並列線形ライブラリ ScaLAPACK の一関数を対象に，グリッド計算資源の最適配置について検討する．このようなスケジューリング問題は，局所解を多く持つ NP 困難な大域的最適化問題となることが分かっている．本研究では，スケジューリング手法に Genetic Algorithm (GA) を利用して，異なる種類のグリッド資源パターンを想定して最適スケジューリングを決定した．先行研究と比較したところ，ノード数が多く，また計算資源の性能差が大きく，制約が多く存在するようなグリッド上で，GA が有効な結果を示した．

Static Scheduling for ScaLAPACK on the Grid Using Genetic Algorithm

Tomoyuki HIROYASU[†], Mitsunori MIKI[†], Hiroki SAITO[††], Yusuke TANIMURA[††], Jack Dongarra [‡]

[†] Knowledge Engineering Dept., Doshisha University

[††] Graduate Student, Doshisha University

[‡] University of Tennessee/Oak Ridge National Laborator

To execute the parallel and distributed applications efficiently on the Computational Grid, the proper nodes should be selected with the consideration of characteristics of applications. In this paper, the linear algebra library, ScaLAPACK, is targeted and the optimum selection of nodes of the computational Grid is discussed. In this study, Genetic Algorithm is utilized to derive the optimum scheduling. Compared to the results of the former studies, it is pointed out that Genetic Algorithm shows the good result on the Grid where there are many nodes, there are huge performance gaps among the nodes, and there are many constraints.

1 はじめに

グリッド上の計算資源を利用して，効率的に分散・並列アプリケーションを実行するためには，アプリケーションの特性に合わせて，適切な計算資源を選び出す必要がある．現在，このようなスケジューリング問題を目的とした研究が多くなされており[1]，局所解を多く持つ NP 困難な大域的最適化問題となることが分かっている[2]．

本研究では，並列線形ライブラリ ScaLAPACK の一関数を対象に，グリッド計算資源の最適配置について検討する．YarKhan らの研究[3]では，スケジューリング手法に Ad-Hoc greedy やシミュレーテッドアニーリング (Simulated Annealing，以降 SA)[4] が最適化手法として用いられているが，特定の計算環境で実験を行った結果しか示されておらず，さらにスケジューリングのオーバーヘッドが問題となっている．

本研究では，スケジューリング手法に遺伝的アルゴリズム (Genetic Algorithm，以降 GA)[5] を適用し，異なる種類のグリッド資源パターンを想定して各手法を比較する．グリッドの資源パターンとしては，資源の規模や各資源の性能の均質／非均質性等が考えられる．特に，計算資源の規模が大きくなり環境が複雑になると，Ad-Hoc greedy や SA では解けない問題において，GA が有効な解を導き出すと考えられる．スケジューリングを行うアプリケーションは，並列実行が可能な数値計算ライブラリの ScaLAPACK[6] である．

本研究での検討は，すべてシミュレーション上のもので，実際の結果との比較は行っていない．これについては今後の課題とする．

2 ScaLAPACK

2.1 概要

ScaLAPACK (Scalable LAPACK) は，共有メモリ用の線形計算ライブラリである LAPACK (Linear Algebra PACKage) を PVM や MPI を使用して分散並列実行できるように開発された分散メモリ用の高性能数値計算ライブラリであり，密行列の分解，線形方程式，線形最小二乗問題，固有値問題，特異値問題を解くことが可能である．通信層に BLACS (Basic Linear Algebra Communication Subprogram) を使用しており，係数行列をプロセスグリッド (P,Q) に基づいてブロックサイクリックに分割し，複数のプロセッサにブロードキャストして LU 分解・QR 分解による行列演算を行うことができる．

図 1 に $N \times N$ の係数行列を $(P,Q) = 2 \times 3$ のプロセスグリッドに基づいてブロックサイクリックにデータ分割する例を示す．まず係数行列を一つのプロセスに割り当てられる最小の行列サイズ，ブロックサイズ (NB) に分割する．そして列方向に 2 つのプロセスを，行方向には 3 つのプロセスをプロセスグリッド 2×3 に従って，繰り返し順に割り当てることで各プロセスへのデータ分割が行える．この時，ScaLAPACK のパラメータ N，NB，(P,Q) の値とプロセスを割り当てる計算資源の性能によって，データ分割に要する処理時間やデータ分配する通信時間が変化し，行列演算の実行時間が変動する．

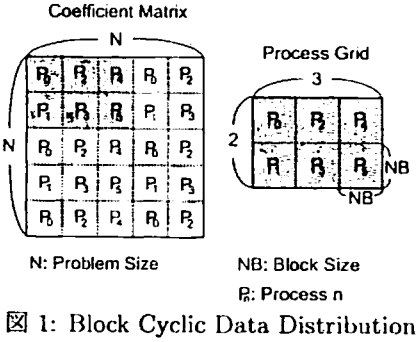


図 1: Block Cyclic Data Distribution

2.2 メモリ制約

実際に ScaLAPACK の行列演算を実行する場合には，計算量に適したメモリ容量が必要となる．メモリ容量が不足した場合にはスワップが発生し計算時間が極めて長くなる．N 次元連立一次方程式を解くためには，$N \times N$ の行列要素を保持するメモリ容量が必要となるので，必要な総メモリ容量は，次元数 N によって式 (1) で表される．

$$Total\,memory[byte] = N^2 \times 8 \qquad (1)$$

3 ScaLAPACK のコスト見積もり関数

実際のアプリケーションの実行時間を予測するためには，そのアプリケーションの実行時間を予測するコスト見積もり関数が必要となる．

現在，グリッドでアプリケーションの実行と開発を行うために技術とツールを提供する Grid Application Development Software (GrADS) Project[7] は，ScaLAPACK の実行時間を予測するルーチンを所有する．この関数は，ScaLAPACK に関するパラメータと計算資源の情報を入力として受け取り，それらの入力値において right-looking LU 分解を行う場合の予想実行時間を計算し出力として返す関数である．計算している見積もりコストは，3 つの重要な LU 分解のフェーズであり，計算による分解フェーズ，通信による分配フェーズ，計算と通信による更新フェーズを繰り返し行った場合の予測実行時間である．以降に，ScaLAPACK の見積もり関数に入力する情報の詳細について説明する．

3.1 ScaLAPACK に関するパラメータ

本来 ScaLAPACK に関するパラメータは実行するユーザが入力して決定することになるが，本研究では表 1 のように定義した．問題サイズは計算資源のメモリ容量で扱える範囲のサイズに設定し，NB は 80 で固定した．P,Q 比については，見積もり関数で扱えるプロセスグリッドが 1 次元であることから，(P,Q) =1 ×使用ノード数となる．

表 1: ScaLAPACK parameters

Parameter	Value
Problem size (N)	1000〜15000, 25000〜37000
Block size (NB)	80
(P,Q)	1 × Numbre of Nodes

3.2 計算資源情報

入力する計算資源情報は，NWS (Network Weather Service) [8] や MDS (Globus Metacomputing Directory Service) [9] を利用して取得される情報を想定している．各ノードにおいて利用できる CPU の性能とメモリ容量，各ノード間のネットワークの状態 (バンド幅，レイテンシ) が入力として必要であり，これらは NWS から得られ，図 2 に示すように配列に保持される．なお CPU の性能は CPU 処理速度 (MHz) と CPU の利用可能率 (0.0〜1.0) の積で計算される．

4 グリッド環境における ScaLAPACK を実行するためのスケジューリング

グリッド上で効率的に ScaLAPACK を実行するためには，ScaLAPACK の特性に合う適切な計算

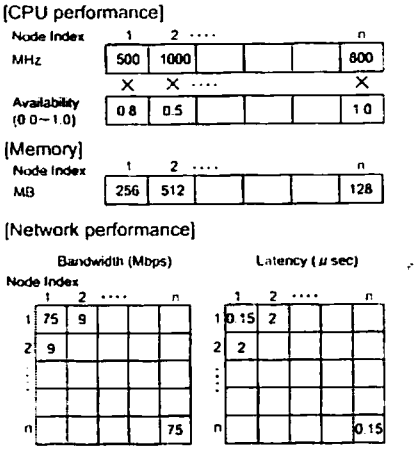


図 2: Information of Resources

資源が必要となる．GrADS Project では，ScaLAPACK のコスト見積もり関数をスケジューリングの目的関数として利用し，より短い時間でグリッド上で ScaLAPACK を実行するために，最適な計算資源を示すスケジュールの生成を試みている．図 3 に，そのスケジューリングの概略図を示す．MDS や NWS からグリッド上の計算資源情報を取得し，Ad-Hoc greedy や SA 等の最適化手法でスケジュールの探索を行う．その際には，ScaLAPACK のコスト見積もり関数へスケジュール情報と計算資源情報を入力し，出力される見積もりコストの値を利用する．

本研究では探索手法に GA を用いたスケジューリングを提案し，文献[3] で用いられている Ad-Hoc greedy や SA との性能比較を試みる．最適化するスケジュール情報は，ScaLAPACK を実行するために使用する計算資源の情報である．本研究では図 4 に示すように，利用可能な計算資源のノードインデックスを保持した配列を用意し，その計算資源のノードインデックスに基づいて選び出した配列をスケジュール情報とする．この一次元配列により，一次元のプロセスグリッドに割り当てる使用ノードを決定することになる．

4.1 従来のスケジューリング手法

GA の比較対象となる Ad-Hoc greedy と SA のアルゴリズムについて説明する．

4.1.1 Ad-Hoc greedy

Ad-Hoc greedy のアルゴリズムの概要は，以下のようになる．

Step1 クラスタの選択

利用可能なクラスタの中から，一つ選択する．

Step2 最速 CPU ノードの選択

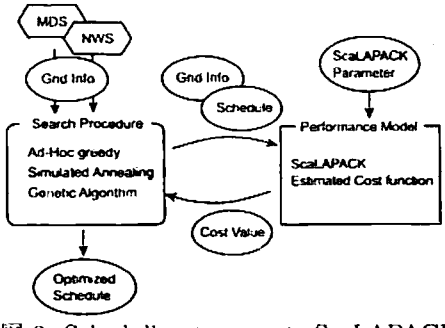


図 3: Scheduling to execute ScaLAPACK

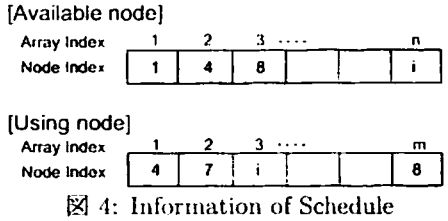


図 4: Information of Schedule

クラスタの中から最速の CPU 性能をもつノードを選択する．

Step3 計算ノードの追加

選択されているノードとのバンド幅をもとに，全ての利用可能ノードから最もネットワークバンド幅の大きいノードを選択し，追加する．

Step4 メモリ制約

使用ノード集合が必要なメモリ容量を満たしていない場合には，Step3 へ戻り，満たしている場合は Step5 へ進む．

Step5 見積もり時間の計算

生成された使用ノードの順序と組み合わせにより見積もり時間を計算する．

Step6 終了判定

得られる最適なスケジュールの見積もり時間が一定期間更新されなくなれば探索を終了して Step1 へ戻る．それ以外は Step2 へ戻る．

ノード間のネットワークのバンド幅に注目して探索を行うアルゴリズムであり，リモート上のノードよりもローカルに存在するノード，クラスタ内のノードを使用する探索法である．これは，ScaLAPACK の特徴に基づいたルールベースの探索法であり，アプリケーションの特徴に基づいて開発された．

4.1.2 SA

SA のアルゴリズムの概要は，以下のようになる．

Step1 利用可能ノード数の取得

利用可能なノード数 N を，MDS と NWS の情報から得る．

Step2 仮想ノード数の初期化

探索するノード数を固定するため，仮想ノード数（$vmsize = 1$）を設定する．

Step3 メモリ制約

問題サイズと仮想ノード数から必要となるメモリ容量を計算し，メモリ制約を満たすノード数 R を求める．

Step4 SA 探索判定

$vmsize > R$ の場合は $vmsize = vmsize + 1$ として，Step3 へ戻る．それ以外は Step5 へ進む．

Step5 初期状態の生成

R 台のノードからランダムに $vmsize$ の数だけノードを選択し，スケジュールを初期化し，見積もり時間を計算する．

Step6 次状態の生成

スケジュールからランダムに 1 台のノードを取り除き，残りの資源（$R - vmsize$）台からランダムに 1 台ノードを選択し，スケジュールのノード順序をランダムに変更して見積もり時間を計算する．

Step7 受理判定

次状態の見積もり時間が短くなった場合は，次状態を受理して遷移し，長くなった場合は Metropolis 基準 $exp^{(-dE/kT)}$ に従い，見積もり時間を E とした受理判定を行う．

Step8 温度の冷却

アニーリングステップ数がクーリング期間で割り切れる場合は，温度 T を冷却する．

Step9 SA 終了判定

見積もり時間が安定するか，温度が最低温度になるまで Step6 へ戻る．そうでない場合は，Step10 へ進む．

Step10 終了判定

$vmsize < N$ の場合は $vmsize = vmsize + 1$ とし，Step2 へ戻る．そうでない場合は探索を終了する．

SA は，Ad-Hoc greedy のように計算資源の特徴に基づいて探索を行うのではなく，確率的な探索を行う方法として用いられている．コスト見積もり関数に存在する局所解を回避するために実装された．本研究ではこれら二つの手法を実装し，次節で提案する GA と比較する．

4.2 GA によるスケジューリング手法

GA は，生物の進化を工学的にモデル化した最適化手法の一つであり，環境により適応した生物，すなわち目的関数に対して最適値を与えるような個体（解）を計算機上で生成することで，解探索を進める手法である．ScaLAPACK の実行時間を見積もるために必要となる計算資源のスケジュールを GA の個体として表現して，最適な個体を選び出す．以下に GA のアルゴリズムの概要を示す．

Step1 初期母集団生成・評価

初期母集団をランダムに生成する．

Step2 評価・エリート保存

母集団の全個体を評価し，評価値の最も良い個体をエリートとして保存する．

Step3 選択

ランキングに基づいたトーナメント選択により，母集団とエリート個体から個体を復元抽出する．

Step4 交叉・突然変異

2 個体を母集団から非復元抽出してランダムに交叉させ，生成した子個体に対してビット反転による突然変異を行う．生成された子集団を母集団と無条件で入れ換える．

Step5 評価・エリート保存

母集団の全個体を評価し，評価値の最も良い個体が保存されたエリート個体の評価値よりも良好な場合は，その個体をエリート個体として保存する．

Step6 終了判定

終了世代に達していない場合は，Step3 へ戻る

個体のコーディング方法と評価方法にはスケジューリング問題に特化したアルゴリズムを用いる．次節にその詳細を説明する．

4.2.1 個体のコーディング

使用する計算資源の組み合わせを個体の設計変数値で表現する．グリッドで使用できるノード 1 台を 1 設計変数とし，1 設計変数を 1 ビットの遺伝子で表現する．そのビットが 0 であればそのノードを使用せず，1 であれば使用するものとする．これより，使用する計算資源の組み合わせを個体の遺伝子の値によって決定することができる．選んだ計算資源の使用順序は，次節に記すように評価時に決定する．

4.2.2 個体の評価方法

個体の評価方法は以下の通りである．本手法では 1 個体を評価するために 2 度の評価を行い，ノードの使用順序を探索するアルゴリズムとなっている．

1. 個体の遺伝子座の小さい値から順に探索し，対応するノードが使用されていれば，スケジューリング情報に順に格納し，評価値 E_1 を求める．
2. 1 で評価したスケジュールのノードの使用順序を，ランダムに変更する．そして，再び評価値を計算し，評価値 E_2 とする．
3. E_1 と E_2 を比較して良好な評価値をその個体の評価値とする．

また評価関数には，ペナルティ関数法を適用し，メモリの制約条件を満たさないスケジュールを生成した個体にはペナルティ値を加える．式 (2) に，ペナルティ関数 F_ρ を示し，式 (3) にペナルティ値を計算する関数を示す．F_ρ の値が評価値 E となる．

$$F_\rho = Estimated\,Cost + g(\rho) \tag{2}$$

$$g(\rho) = \rho(N - (Total\,memory[byte]/8)^{1/4}) \tag{3}$$

5 数値実験

本研究で実装したGAによるスケジューリングの性能を検討するため，Ad-Hoc greedy，SAを用いたスケジューリング手法との比較実験を行う．実験内容は，複数の計算資源から構成されるグリッド上で，ScaLAPACKの実行時間の見積もりが最小となるスケジュールを探索することである．大きく分けて実験では，文献[3]で示されているグリッド，本研究で想定した小規模な計算資源から構成されるグリッド，また大規模な計算資源から構成されるグリッドの合計3つのグリッド計算環境を想定して行う．

5.1 パラメータ

実験に用いたGA，SAのパラメータを表2，表3のように決定した[10, 11]．

表2: GA parameters

Parameter	Value
Population Size	100
Chromosome Length	Number of nodes (=L)
Selection Method	Tournament
Tournament Size	4
Crossover Method	Two-Point Crossover
Crossover Rate	0.6
Mutation Rate	1/L
Number of Elites	1
Number of Generations	500
ρ (penalty parameter)	300, 500

表3: SA parameters

Parameter	Value
Max. temperature	150
Min. temperature	4.3
Number of Annealing Steps	1.0×10^4
Cooling Function	$a(T) = \alpha T$
Cooling Rate	0.9648
Cooling Period	100

5.2 グリッド環境1による性能比較

文献[3]をもとに，同じグリッド上の計算資源でスケジューリングを行った．実験に用いた計算資源の静的情報を表4に示す．表4はテネシー大学のTORCクラスタとMSCクラスタ，イリノイ大学のOPUSクラスタの性能を示している．なお，大学間のネットワーク速度を9Mbps，2μsecとして，TORCクラスタとOPUSクラスタ間のネットワーク速度を20Mps，0.3μsecとした．なお，文献[3]では表4の環境において問題サイズ14000が最大であったため，利用可能なメモリの容量に対して，40%の容量をScaLAPACKで使用するように設定した．またペナルティのパラメータρは300とした．Ad-Hoc greedyはルールベースによる探索であるため試行は1回とし，SAとGAについては確率的探索であるため20回の試行を行い，見積もりコストの中央値において比較する．図5に各問題サイズにおける結果を示す．

図5の結果から，提案するGAは問題サイズが13000，14000の場合においてAd-Hoc greedyよりも良く，SAとほぼ同等の性能を示していることが確認できる．Ad-Hoc greedyの性能が，問題サイズが大きい場合に悪くなっているのは，ノード順序を考慮していない探索アルゴリズムにある．表5に，問題サイズが14000における各手法により得られたある1試行の使用ノード順序の結果を示す．問題サイズが14000の場合には，メモリ制約により全計算資源ノードを使用することになり，各手法が全15ノードを使用している．しかし，使用順序は異なり，得られた見積もりコストが大きく異なっていることが確認できる．

また，問題サイズが12000以下でGAの性能が悪くなっているのは，図6に示している各手法が生成したスケジュール結果から，選択したノードにあることが確認できる．図6を見ると，GAが選択したノードは他の手法が選択したノードと大きく異なっている．GAが選んでいるノードの総メモリ容量は2304MBであり，必要メモリ容量の2307.8MBを満たしていない．これは，制約付近の設定したペナルティ値が極めて低い値であったため，制約を満たしていない個体がエリート個体として生存する結果となった．これはペナルティ関数の設定が適切でなかった可能性がある．

表4: Static Information of Resources

	TORC	MSC	OPUS
Number of Nodes	3	3	9
CPU Speed (MHz)	550	933	450
CPU Availability	0.9	0.9	0.9
Bandwidth (Mbps)	75	75	75
Latency (μsec)	0.15	0.15	0.15
Memory (MB)	512	512	256

表5: Order of Nodes and Cost (N=14000)

	Node order	Cost
Ad-Hoc	5,3,4,0,1,2,6,7,8,9,10,11,12,13,14	1507.1
SA	9,7,11,13,8,14,4,5,3,1,0,2,6,12,10	1292.1
GA	5,4,8,7,11,10,12,14,13,6,9,0,1,2,3	1235.5

5.3 グリッド環境2による性能比較

3ノードからなる2つのクラスタと，9ノードからなる1つのクラスタから構成される小規模計算資源を，計算資源の全ノードのネットワークのバンド幅とCPU速度の標準偏差により，4つのパターンに分類して各手法の性能比較を行う．それぞれの計算資源のパターンを表6に示す．

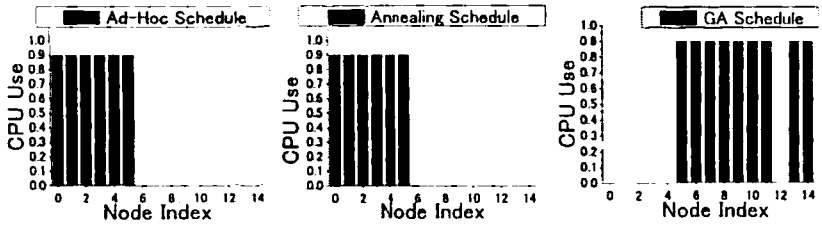


図 6: Selection of Nodes (N=11000)

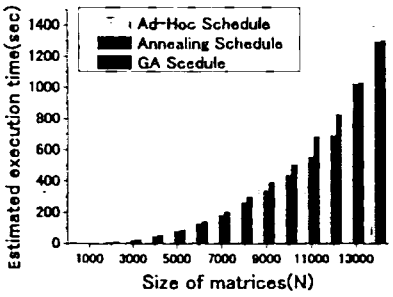


図 5: Estimated execution times for SA and Ad-Hoc greedy, GA schedules

表 6: Pattern of Resources

Pattern	Bandwidth Gap	CPU Speed Gap
1	Small ($SD < 50$)	Small ($SD < 150$)
2	Large ($SD > 500$)	Small
3	Small	Large ($SD > 300$)
4	Large	Large

SD : Standard Deviation

メモリについては，4 パターン全て 256MB の均一性能とし，利用可能なメモリ容量の 50%を ScaLAPACK で使用するように設定した．また ρ は 300 とした．図 7 に実験結果を示す．4 つの実験において，問題サイズが 13000 以上になると Ad-Hoc greedy のコストが著しく悪くなり，GA と SA の手法が良好である．これは，5.2 節で示したように，Ad-Hoc greedy が計算資源の使用順序を考慮した探索を行っていないからである．

5.4 グリッド環境 3 による性能比較

文献 [3] では，大規模計算資源からなる環境での実験が示されていない．そこで本研究では，合計 100 ノード，10 個のクラスタからなる大規模計算資源による実験を行った．表 6 に示す 4 つの分類に従って，計算資源の値を設定した．ScaLAPCK で使用するメモリ容量を 50%とし，ρ は 500 とした．図 8 に実験結果を示す．

大規模計算資源においては，問題サイズが大きい場合に，GA が最も良い性能を示していることが確認できる．特に，CPU とネットワークの性能差が大きいパターン 4 のグリッドにおいては，見積もりコストの時間差が大きく，GA が最も良好な結果を示した．問題サイズ 31000 以上で SA の探索が悪くなるのは，使用ノード数が増加することで使用ノードの組み合わせや使用順序のパターンが増加し，SA が局所解へ陥りやすくなったためである．図 9 に，パターン 4 の実験の問題サイズ 37000 のある 1 試行の探索履歴を示す．図 9 の探索履歴から，SA の探索が途中から停滞していることが確認できる．GA の見積もりコストが極端に下がっているのは，メモリ制約を満たしていなかったスケジュールから，制約を満たすスケジュールが生成されたことで，ペナルティによる値が無くなったことを示している．パターン 4 において，問題サイズが 35000 の場合に得られたスケジュールを図 10 に示す．図 10 から，各手法で使用しているノードの組み合わせが異なっていることが確認できる．GA の性能が問題サイズ 30000 以下の場合において悪い結果を示すのは，5.2 での結果と同様に，ペナルティ値の設定が適切で無かったためである．

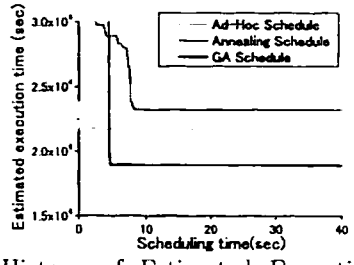
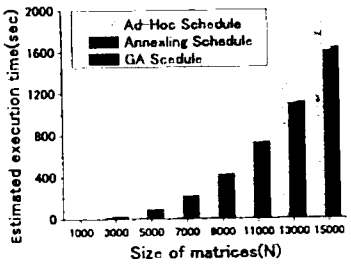


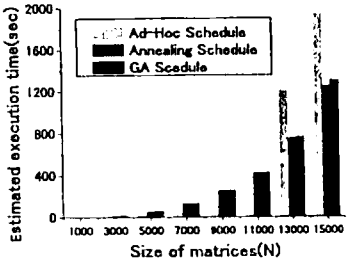
図 9: History of Estimated Execution Time (N=37000)

5.5 スケジューリング時間による性能比較

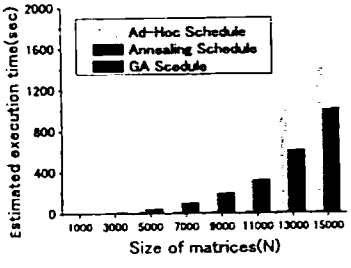
スケジューリングによって求めた最適な見積もり時間と，その時間を生成するために要したスケジューリング時間の和をスケジューリングに要し



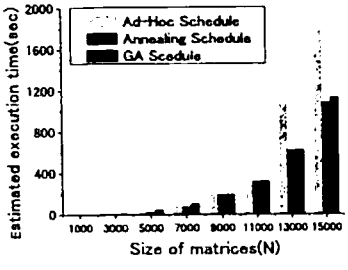
(a) Pattern1



(b) Pattern2

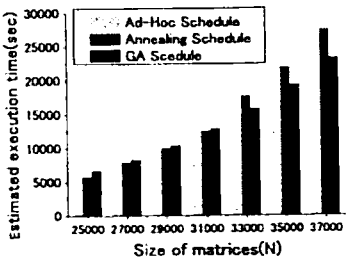


(c) Pattern3

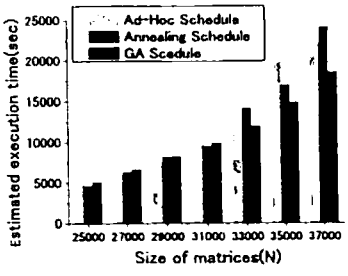


(d) Pattern4

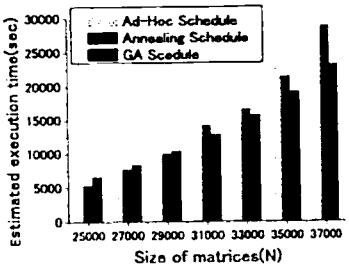
図 7: Estimated Execution times for the GA and SA, Ad-Hoc greedy on small-scale of resources



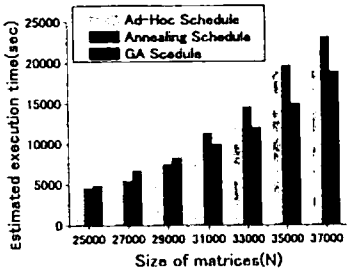
(a) Pattern1



(b) Pattern2



(c) Pattern3



(d) Pattern4

図 8: Estimated Execution times for the GA and SA, Ad-Hoc greedy on large-scale of resources

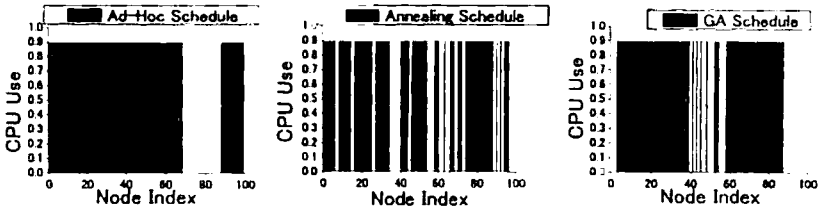
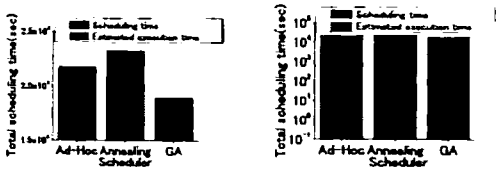


図 10: Selection of Nodes (N=35000)

た総合時間とし，その時間による性能比較を行う．5.4 節に示したパターン 4 の大規模計算資源におけるグリッド上で，問題サイズ 37000 で実行した場合における各手法のスケジューリングに要した総合時間を測定した．各手法の実行には CPU が PentiumIV 1.6GHz，メモリが 512MB の Linux が動作する PC で行った．図 11 に，測定したスケジューリング時間の結果を示す．図 11 から，GA が Ad-Hoc greedy や SA よりも短い時間で良好な結果を導き出していることが確認できる．



(a) Linear (b) Log

図 11: Total Scheduling time (N=37000)

6 まとめ

本研究では，ScaLAPACK の性能モデルを利用して，数種類の計算資源のパターンで，静的情報によるスケジューリングの検討を行った．GA によるスケジューリング手法は，大規模計算資源によるグリッド上において，従来の Ad-Hoc greedy や SA による手法よりも良好な結果を示すことが多く，スケジューリング時間も SA より短くなった．しかし GA の問題点としてペナルティ値の設定が挙げられ，今後はあらゆる問題サイズで良好な結果を示すような値を検討する必要がある．そして本研究をもとに，静的情報だけでなく，動的情報を利用したスケジューリング手法に GA を適用することが今後の課題である．また，生成された最適なスケジュールを用いて，実環境で ScaLAPACK を実行する必要がある．

参考文献

1) Francine Berman. High-performance schedulers. *The Grid: Blueprit for a New Computing Infrastructure*, pp. 279-309, 1999.

2) J Ullman. Np-complete scheduling. *Journal of Computer and System Sciences*, pp. 384-393, 1975.

3) Asim YarKhan, Jack J. Dongarra. Experiments with scheduling using simulated annealing in grid environment. 2002.

4) Kirkpatrick, S., Gelett Jr. C. D., , Vecchi, M. P. Optimization by simulated annealing. *Science*, Vol. 220, No. 4598, pp. 671-680, 1983.

5) D.E.Goldberg. *Genetic Algorithms in Search Optimization and Machine Learnig*. Addison-Wesley, 1989.

6) L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: a linear algebra library for message-passing computers. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997)*, p. 15 (electronic), Philadelphia, PA, USA, 1997. Society for Industrial and Applied Mathematics.

7) Francine Berman, Andrew Chien, Keith Cooper, Jack Dongarra, Ian Foster, Dennis Gannon, Lennart Johnsson, Ken Kennedy, Carl Kesselman, John Mellor-Crummey, Dan Reed, Linda Torczon, and Rich Wolski. The GrADS Project: Software support for high-level Grid application development. *The International Journal of High Performance Computing Applications*, Vol. 15, No. 4, pp. 327-344, 2001.

8) Neil T. Spring Rich Wolski and Jim Hayes. The network weather service: a distributed resource performance forecasting service for metacomputing. *Future Generation Computer Systems*, pp. 757-768, 1999.

9) I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, Vol. 2, No. 11, pp. 115-128, 1997.

10) 伊庭斉志. 遺伝的アルゴリズムの基礎. オーム社, 1994.

11) 小西健三, 瀧和男. 温度並列シミュレーテッドアニーリング法の評価ー lsi ブロック配置問題に関してー. 情報処理学会 DA シンポジウム'94, pp. 223-228, 1994.