jh160002-NAJ

分散メモリ型スーパーコンピュータにおける 直接法と反復法の並列化行列解法の研究

若谷彰良（甲南大学）

本研究は，SPCG 法をはじめとする反復法と軸選択付き LDU 分解による直接法がベクトル型及びスーパースカラ型の分散メモリ型スーパーコンピュータで高い並列性と実行性能を実現する方法を確立することを目的としている．nested-dissection オーダリングを採用した直接法は，非同期並列演算を行うことで 100 万自由度規模の問題を，マルチコアシステムの 1 ノード内で 64G バイトの主記憶を用いて 50%以上の並列効率を実現できることを確認した．また複数ノードを利用するためのコードの拡張を開始するとともに FETI 領域分割法での直接法利用の有用性の検証を行った．反復法においては SPCG 法に対する 5 つの実装方法を検討し，ベクトルアーキテクチャの分散型スーパーコンピュータにおいては，ノード間通信を極力削減した手法が，収束性能を加味した総合的な評価において最適であることが分かった．半導体ドリフト拡散モデルの定常問題を実アプリケーションとして直接法と反復法の二種の行列解法の有効性を検証した．

1. 共同研究に関する情報

(1) 共同研究を実施した拠点名

大阪大学

(2) 共同研究分野

- ◉ 超大規模数値計算系応用分野
- ○ 超大規模データ処理系応用分野
- ○ 超大容量ネットワーク技術分野
- ○ 超大規模情報システム関連研究分野

(3) 参加研究者の役割分担

・若谷 彰良（甲南大学・知能情報学部）
研究統括・並列化反復法計算手法

・小田中 紳二（大阪大学・サイバーメディアセンター）
半導体シミュレーション

・鈴木 厚（大阪大学・サイバーメディアセンター）並列化直接法計算手法

・鍾 菁廣（大阪大学・サイバーメディアセンター）数値シミュレーション

2. 研究の目的と意義

現代のスーパーコンピュータは共有メモリ構造を持つマルチコアシステムのクラスタで構成されている.ここにはキャッシュメモリ，主記憶，ネットワークを介した分散記憶の 3 種類のアクセス速度の異なるメモリ階層がある．1 つのプロセッサの内部は 4 から 16 程度のマルチコアを持ち，1 ノードの論理演算速度は 256GFlop/s 程度になっている．クラスタ全体は 1,000 ノードを超えるが，実アプリケーションでは，30 ノード程度での高い並列効率が必要である.ノード内メモリバンド幅とノード間通信バンド幅の差は大きいが，この環境でも演算速度を保てるように最適なタスク配置と新規アルゴリズムの導入により通信コストの低減を図る．またノード内での演算器の効率的な利用を実現する必要がある．偏微分方程式の離散化で得られる対称行列及び非対称行列の問題に対する反復法及び直接法の並列化手法を研究し，各種スーパーコンピュータ上で評価することによって，メニーコア時代に適応した並列化行列解法を提供するとともに，対称行列及び非対称行列を用いた大規模半導体シミュレーションで有用性を検証することを目的とする．

特に，反復法においては Splitting-Up 作用素を用いた前処理付き共役勾配法の実装を主な対象とし，直接法においては Dissection 法による疎行列のグラフ分割と密行列部分のブロック化によって得られた LDU 分解のタスクの非同期実行とデータ配置の最適化を主な対象として取り組んでいる．

3. 当拠点公募型共同研究として実施した意義

数値シミュレーションを分散メモリ型スーパーコンピューティングに適用する場合，一般にノード間通信バンド幅はノード内メモリバンド幅よりも小さいので，問題自体を領域分割し，分割された問題を各ノードに割り当てる領域分割法が利用され，各領域で高速な行列解法が必要とされる．そこで，この行列解法としては，直接法か反復法を選択することになる．このため，両方の解法の研究が必要

である．さらに，分散メモリ環境でも，領域分割を用いない“フラット”な行列解法も必要であり，本研究では反復法でこれを行う．

行列解法の反復法には，ICCG (Incomplete Cholesky Conjugate Gradient)法がしばしば用いられるが，収束性能を保ったままで，単純に並列化することが難しいものとされている．一方，前述の通り，Splitting-Up 作用素を前処理として用いた共役勾配法(SPCG 法)は，ICCG 法と同等程度の収束性能を持ち，また，並列実行に適したアルゴリズムである．しかし，前処理の際に，行列の分散形態を変更する配列再分散が必要となり，その通信コストが性能を律速することになり，並列化効果を低下させるので，分散メモリシステムに適したアルゴリズムの変更および最適なタスクの配置を提案することが必要となる．

また，直接法は疎行列のオーダリングにより並列演算が可能な部分を抜き出すことで共有メモリ環境での並列計算を実現する．Schur 補行列の再帰的構成に基づくため，分散環境では境界の自由度の自乗に比例するデータを転送する必要が生じる．このため大規模分散環境では直接法単体ではなく，領域分割の内部境界の自由度に関する部分構造反復法と組み合わせることが必要となる．

そこで，本研究では，SPCG 法をはじめとする反復法及び部分構造反復法の直接法のアルゴリズム研究を行うために，並列アルゴリズムの研究者に数値シミュレーションの研究者を加えたチームを構成して，拠点の異なるスーパーコンピュータ上で評価するため，公募型共同研究として実施することに意義があると言える．

4. 前年度までに得られた研究成果の概要

該当なし

5. 今年度の研究成果の詳細

直接法と SPCG 法をはじめとする反復法の研究成果について詳述する．

本研究では東京大学，京都大学，大阪大学の共同利用・共同研究拠点の 3 種の異なるアーキテクチャの計算機を利用した．東京大学のシステムは Fujitsu FX-10 で，マルチコアスーパースカラー型 CPU，SPARC64IXfx 1 基を 1 ノードとするクラスタシステムであり，1 ノードあたり 16 コアを有し 236.5Flop/s の論理性能を持つ．京都大学のシステムは Cray XC30 でマルチコアスーパスカラー型 CPU，Intel Xeon E5-2695v3 を 2 基 1 ノードとするクラスタシステムであり，1 ノードあたり 28 コアを有し 1,030.4Flop/s の論理性能を持つ．大阪大学のシステムは NEC SX-ACE でマルチコアベクトル型 CPU，1 基を 1 ノードとするクラスタシステムであり，ノードあたり 4 コアを有し 256.0 GFlop/s の論理性能を持つ．以下，単一ノード内の性能比較においては CPU 名，Fujitsu SPARC64IXfx，Intel Xeon E5v3，NEC SX-ACE を用い，複数ノードでの性能比較では，計算機システム名 FX10，XC30，SX-ACE を用いる．

5.1 直接法解法

有限差分法や有限体積法，有限要素法により得られる連立一次方程式の係数行列は大規模な疎行列である．疎行列の LDU 分解を実行すると，行列成分で 0 であったものが，分解後に値をもつようになる．これを Fill-in と呼ぶが，その出現は行列の非零要素のパターンと分解手続きの順序に依存する．自由度の添字の順を並べ替えること(オーダリング)で，Fill-in パターンを最小化することができる．

帯行列として捉えることで帯幅を最小にする Cuthill-McKee 法はベクトル演算器に適しているが，LDU 分解は本質的に逐次演算であり，並列演算はできない．また，対象とする領域で定数でない物理係数を持つ問題から得られる大きな条件数の行列では，浮動小数点解像度内での演算の安定化と精度を保つための軸選択が必須である．これは，オーダリングが入力の数値データに依存することを意味する．Cuthill-McKee 法ではオーダリングは非零要素の結合関係のみを考慮しており，軸選択を導入すると帯幅を最小化する手法の特性を完全に崩してしまうため軸選択を取り込むことができない．一方，nested-dissection 法は領域分割のアイディアに基づき，係数行列を 2 つの疎行列とその境界部分の和に再帰的に分割する方法である．境界部分はサイズの小さな密行列となり，軸選択の適用ができる．また 2 分木の層を多くとると複数の独立した疎行列を得ることができるため，LDU 分解の並列化が可能になる．これをマルチフロンタル法と呼ぶ．また，内部の境界は密行列であるため，BLAS level 3 のサブルーチンを用いて，最近の，十分なキャッシュメモリ容量を持つスーパースカラ型の演算器を有効に活用することができる．

直接法解法ソフトウェア Dissection は METIS あるい

は SCOTCH によるグラフオーダリングから，いくつかの疎行列の部分行列と，密行列への分解を得る．一つの部分疎行列のサイズが 200 から 500 程度になるように 2 分木の層の数を設定する．2 分木の葉の層では部分行列が多数あるため，部分行列のサイズのばらつきに起因する負荷分散に注意すれば，並列化そのものは容易である．2 分木の根の層では，部分行列数が並列稼働可能なコア数よりも少なくなるため，部分行列そのものの並列化が必須になる．これは，ブロック化を導入することで実現する．またブロック化により，行列と行列の積の演算の BLAS level 3 DGEMM を活用することができる．

本研究課題でターゲットとする，マルチコアスカラ型スーパーコンピュータの共有メモリノードでは，図 1 に示すように，同種の直接法解法である Pardiso, MUMPS と比較して計算速度，並列効率とも優位な性能が得られた．

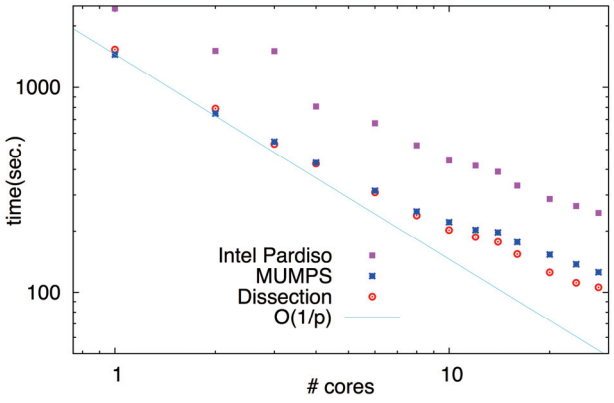

図 1. 100 万自由度 FEM 行列の並列効率の比較

コードは，インテル社製の C++ コンパイラを持いて翻訳し，MKL ライブラリに含まれる単一コア用の逐次版の BLAS ライブラリをリンクしている．

テストに使用した行列は非圧縮性流体の Navier-Stokes 方程式を P2/P1 要素により離散化を行い，定常解を求める非線形 Newton 反復の内部の線形ソルバーの係数行列として得られるものであり，1,032,183 自由度，97,961,089 の非零要素を有する．また圧力変数は定数分の自由度を含むため，行列は特異であり，1 次元分の核を持つ．Pardiso はインテル社による実装によるもので，MKL ライブラリに含まれるものであるが，特異な行列を判定する機能を持たないため，LDU 分解の演算は終了するが解の誤差は 10^{-3} 程度であり，求解はできていない．

Dissection 直接法では 1 基あたり 14 コアを持つ Intel Xeon E5v3 CPU を 2 基，合計 64GB の主記憶を持つ 1 ノードの演算資源で，57GB を使用して，100 秒強で LDU 分解が可能である．

SX-ACE は演算器がベクトル型であるため，行列演算はベクトル化を行う必要があるが，これは Mathkeisan ライブラリに含まれるシングルコア用の逐次版の BLAS ライブラリをリンクする．

ブロック化を行った LDU 分解は，タスク間の依存関係を保って非同期に並列実行を行うことが可能である．この非同期処理により，プロセスの空き時間はほとんど零になる．タスク割当は，POSIX threads ライブラリによって実装しているため，UNIX システムを採用している SX-ACE でも問題なく実行できる．図 2,3 に Intel Xeon と SX-ACE での各タスクの並列非同期実行の様子を示す．

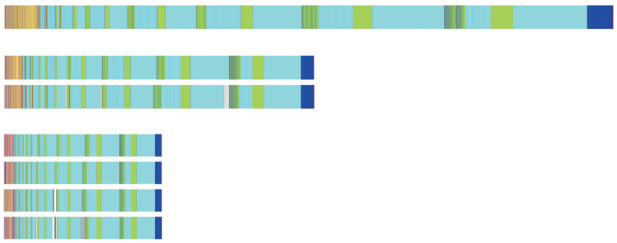

図 2. Intel Xeon E5v3 での並列非同期タスク実行

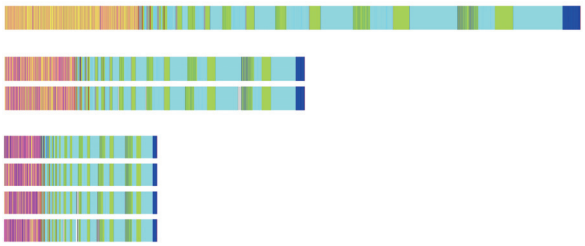

図 3. NEC SX-ACE 最適化 前

密行列演算部分での DGEMM ルーチンの演算実行性能を表 1 に示す．Intel Xeon E5v3 は動的なクロック制御を行うため，ピーク性能はコアの稼働数に線形に比例しないが，Xeon E5v3, SX-ACE とも，65% 程度の実行性能が得られている．ところが，疎行列部分の演算（図 3 で，黄色と紫色の部分）は SX-ACE では，Intel Xeon に比較して，4 倍程度

表 1. BLAS3 DGEMM ルーチン実効性能

# of cores	Intel Xeon	NEC SX-ACE
1	36.35	44.85
2	36.27	43.76
4	34.06	41.31
8	31.23	
16	25.25	
24	24.38	
28	22.90	

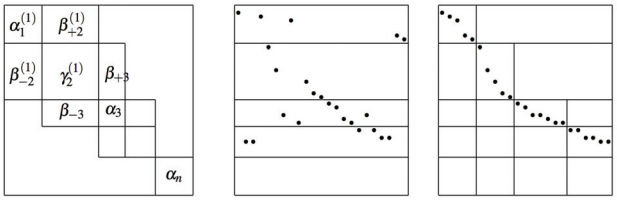

図 4. 部分問題疎行列 LDU 分解

の時間が必要である.疎行列演算ルーチンのループ長さが短いため，ベクトル演算器が活用できないことがその原因である．疎行列を LDU 分解した後に，2 分木の上位の層との関わりを表す疎なデータ構造を持つ非対角ブロック同士の掛け算により，Schur 補行列を更新するが，これは疎なデータ構造向けに修正した DGEMM ルーチンにより実行している.演算量を最適化するため，疎行列パターンによる並べ替えと零でない成分を包み込むブロック化を行っている(図 4)．しかし，この操作は主記憶のデータの移動を行うのみで浮動小数点演算器は稼働しない.

ベクトル機では，零成分の演算が無駄になることを許容して,密行列成分からなる DGEMM ルーチンに置き換えると高速化が達成できることが分かった（図 5).

また，紫色の疎行列 LDU 分解のタスクと黄色の DGEMM 演算は，連続するタスクとして交互に実行することが効果的である.これは，疎行列 LDU 分解は主記憶への頻繁なアクセスを必要とするが，DGEMM 演算はデータの再利用が可能であるため，主記憶へのアクセス要求が低い．全てのコアが LDU 分解を実行するとメモリ競合が起きるが,異なるタスクの交互実行によりコア間でのメモリアクセスへの競合を緩和することができる．スーパースカラ型の CPU では，十分なキャッシュメモリにより隠蔽できているが，ベクトル型の CPU ではキャッメモリの容量が少ないため，この最適化も必要であった.

図 6 に Fujitsu SPARC64IXfx，Intel Xeon E5v3，NEC SX-ACE での Dissection 直接法の性能比較を示す．FX10 の 1 ノードでのユーザー利用可能主記憶が 28GB であるため，

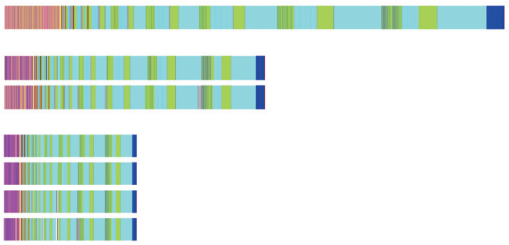

図 5. NEC SX-ACE 最適化 後

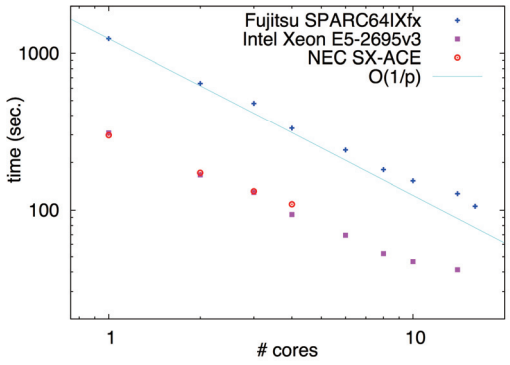

図 6. 3 種の CPU アーキテクチャでの性能比較

645,928 自由度，60,863,284 の非零要素数を持つ Stokes 方程式の対称行列を用いた．SPARC64IXfx は 1 コアあたり，14.86GFlop/s のピーク性能を持つが，1/3 程度の実行性能であった．これは Dissection コードが，シングルコアでの逐次型のサイズの小さな DGEMM ルーチンを非同期に実行することで待ち時間を減らし，演算効率を高める実装を採用しているのに対し，SPARC64IXfx 向けの SSL2 ライブラリに含まれる DGEMM が複数コアでのサイズの大きな行列の演算を目的に最適化されているためだと考えられる.並列効率は 16 コアで 73%と高効率である.

5.2 反復法解法

一方，SPCG (Splitting-Up Conjugate Gradient)法は，前処理に Splitting-Up オペレータを用いた共役勾配法で，対称行列に対しては ICCG (Incomplete Cholesky Conjugate Gradient)法と同程度の収束性能を持つとともに，並列化に向いた反復手法であり，前処理部分と CG 部分からなる.CG 部分では行列ベクトル積の計算及び内積計算で MPI 通信が必要となるが，そのデータ量はわずかである．一方，前処理部分では 3 次元の各方向で逐次の三重対角行列解法が必要となるので，1 次元分散をしていても，配列の再分散が必要となり，通信に必要なデータ量は配列全体となる．そこで，配列再分散を行う方法を含め，5 つの手法での実行性能，全体性能を検討した．なお，5 つの内，2 つの方法はアルゴリズムの変更を伴うので，収束性能が異なる．また，収束性能が劣る代わりに実行性能が優れる場合があるので,その両方を加味した性能,すなわち,実行時間に対する収束性能を全体性能として，評価に加える.

配列再分散を用いた手法(redist)に続く 2 番目の方法(pipe)は，プロセス間の通信をパイプライン化したものである．分散されたベクトル間で通信を用いて逐次に実行さ

れるが，このベクトルは 2 次元上にあるので，順次にパイプライン実行することで並列化が可能である．ただし，1 データ毎に通信するのは効率が悪いので，1 次元分のデータをまとめてパイプライン化することで効率化を図る．3 番目の方法(p-scheme)は，分散されている次元の三重対角行列に P-scheme 法を用いて並列化する．これは，三重対角行列を解くための前進代入と後退代入のいずれも，3 ステップの計算に分解し，2 番目のステップでパイプラインの通信を用いるものである．通信は 2 次元空間をまとめて行うが，この部分の通信隠蔽ができないので，プロセス間通信が多くなると，コストが高くなるが，通信起動回数は最小化できる．なお，P-scheme 法の演算量自体は通常の前進代入および後進代入の 1.5 倍程度増加する．以降の方法は分散している次元での前処理にプロセス間通信を用いない方法である．4 番目の方法(NoZSolve)は分散している次元での三重対角行列を解かない方法である．つまり，3 つの次元のうち 2 次元分だけを解くことで前処理の収束性能が低下するが，その分のプロセス間通信を含む計算時間が削減できる．最後の方法(ISPCG)は，分散している次元の三重対角行列は，分散しているデータだけを用いて前進代入と後進代入して解くものである．

図 7 に，ベクトル演算器をもつ SX-ACE における，384×384×384 のデータに対する 8 ノード(32 コア)での全体性能を示す．データ分散は Z 方向のみをコア間で分散し，32 プロセスを生成する．

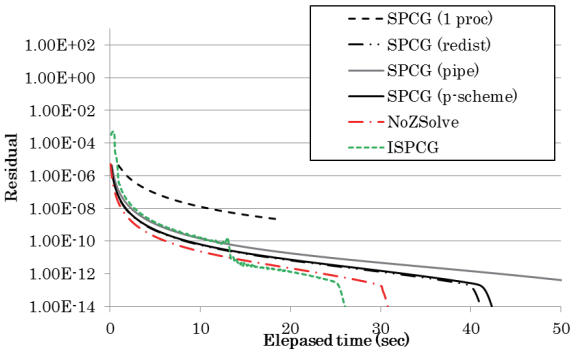

図 7. SPCG 全体性能 (SX-ACE)

図から分かるように，実行時間の中盤あたりまでは，NoZSolve の残差の減少が一番高く，その後，ISPCG の残差減少が上回る．したがって，Z 方向の三重対角行列を解かないもしくは不十分に解く手法が，正しく解く方法よりも全体性能を上回っていることが分かり，アルゴリズムを変えてでも通信を削減することの有用性を示している．なお，他のサイズ，192×192×192，96×96×96 でも，本結果と

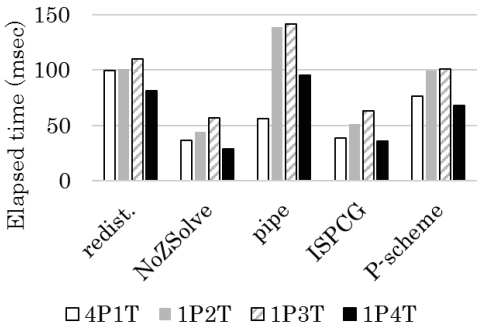

図 8. SPCG 性能比較 (SX-ACE)

ほぼ同様の傾向を示していた．

図 8 に，ノード内のスレッド数の変化が実行性能に与える影響について示す．384×384×384 のデータサイズにおいてノード数は 16 とする．なお，aPbT は，4 個のコアを持つ各ノードに a 個の MPI プロセスと b 個の OpenMP スレッドに割り当てることを示す．

SPCG (pipe)における 1P4T では，0 番のスレッド(コア)が通信している間，1 から 3 番のスレッド(コア)は停止して計算をしていないのに対し，4P1T であれば，各コアは通信も計算も両方行うので，他のコアの通信のために停止することは無い．また，1P4T の方が通信間に行うコア当りのベクトル長が短くなり，演算性能が下がることが考えられる．以上のような理由で OpenMP を用いない 4P1T の方が優れていると考えられる．他の通信が無い手法に対しては，SX-ACE のようなキャッシュを前提にしていないシステムなので，4P1T と 1P4T の差が無いのは当然であり，通信があっても，SPCG (redist)における通信量は 4P1T も 1P4T も同じである，また，SPCG (p-scheme)では通信起動回数が少ないので，差が出にくくなっていると考えらる．なお，データサイズ及びノード数を変えても同様の傾向を示していた．詳細は割愛する．

次に，研究の後半で行った，ベクトル演算器を持たない Cray XC30 と FX10 での反復法の実験結果について述べる．図 9 に，Cray XC30 における 128×128×1024 のデータに対する 32 ノード(512 コア)での全体性能を示す．データ分散は Z 方向 (1024) のみをコア間で分散し，512 プロセスを生成する．SPCG (redist)と ISPCG は割愛する．

SX-ACE の場合と同様に NoZSolve の性能が最も高いが，SPCG (p-scheme)はかなり低速になっている．演算性能と比較した場合のコア当りのメモリバンド幅は，SX-ACE よ

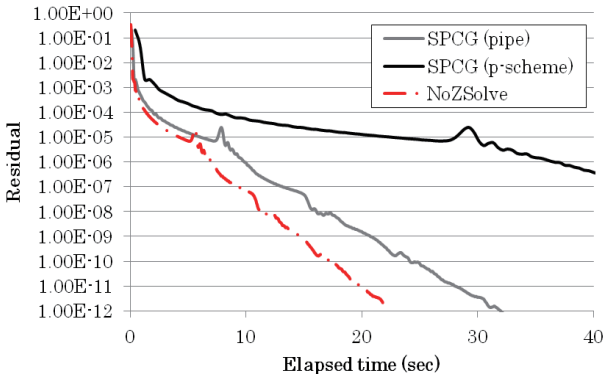

図 9. SPCG 全体性能（Cray XC30）

りも Cray XC30 の方が低い．よって，SPCG 法のようなメモリ IO バウンドな計算は，キャッシュがヒットしない場合，性能の低下が著しい．特に，SPCG (p-scheme)は，元の三重対角行列解法よりも多くの演算が必要となるので，より性能が低下しているものと考えられる．

図 10 に，ノード内のスレッド数の変化が実行性能に与える影響について示す．同じデータサイズにおいてノード数は 32 とし，ノード内の 16 コア利用する．よって，全体として 512 コアを利用する場合で計測し，aPbT は，ノード内に a/32 個の MPI プロセスを生成し，各 MPI プロセスに b 個の OpenMP プロセスを生成して，b 個のコアで実行することを示し，MPI 通信は OpenMP スレッドの 0 番目のスレッドが担当するようにしている．

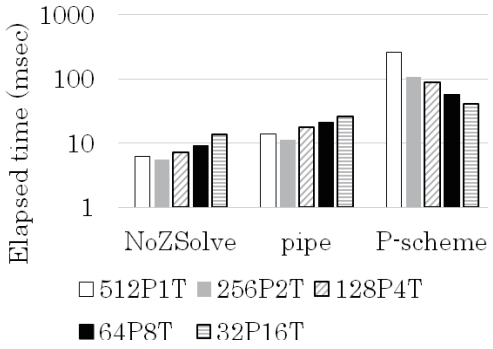

図 10. SPCG 性能比較（Cray XC30）

NoZSolve と SPCG (pipe)は 256P2T で最も実行時間が短くなった．一般に，スレッド間でメモリを共有することで生じるキャッシュヒット率低下が少なければ，MPI プロセス数を少なくして MPI 通信を削減することよって性能向上が図られる．CG 部分では通信はあるが，NoZSolve では前処理部分では通信が無く，SPCG (pipe)は計算とオーバーラップできるので通信コストは少ない．よって，その最適の組み合わせが 256P2T であると考えられる．

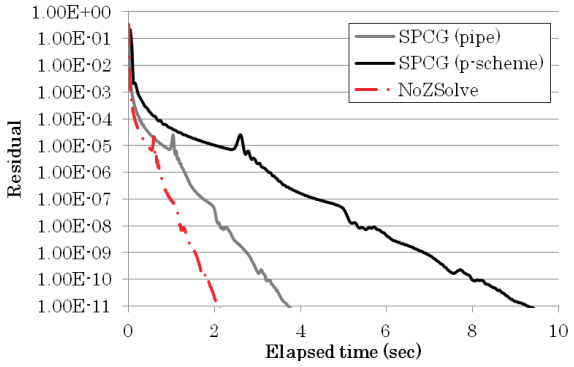

図 11. SPCG 全体性能（FX10）

一方，SPCG (p-scheme)は 512P1T が最も低速であった．SPCG (p-scheme)は MPI 通信を逐次に行う部分があるが，その通信コストは大きく，MPI プロセスの個数に比例したコスト増が計算時間に影響を与えていると考えられる．

図 11 に，富士通 FX10 における，Cray XC30 と同じ条件での全体性能を示す．結果は Cray XC30 とほぼ同様となる．すなわち，NoZSolve の性能が最も高いが，SPCG (p-scheme)はかなり低速になっており，メモリ IO バウンドな計算は，キャッシュがヒットしない場合，性能の低下が著しい．Cray XC30 及び FX10 のいずれも，ノード内を OpenMP でフルにマルチスレッドで実装するよりも，MPI プロセスと混合することで最適となっている．なお，演算時間自体は Cray XC30 に比べ，1 桁程度速くなっている．

5.3 半導体シミュレーションでの高精度解法の必要性

半導体ドリフト拡散モデルによる定常状態を記述する方程式系はポテンシャル，電子密度，及び正孔密度に関する 2 階偏微分楕円型方程式の非線形結合で表される．指数関数を用いた Slotboom 変換により熱平衡状態からの変位を Gummel マップで反復計算することで定常状態を求めることができる．非線形性は Gummel マップにより線形化されるが，得られる電子密度，及び正孔密度に関する離散化行列は非対称となる．Dissection 直接法をドリフト拡散コードに組み込み，NEC SX-ACE の 1 ノードを用いて検証を行った．

実際のデバイス構造として用いられる Bulk 型 Si n-MOSFET(図 12)において計算を行った．ゲート印加電圧を Vg=0.0V，ドレイン印加電圧を Vd=0.2V に設定した際の正孔密度分布を図 13 に示す(この結果は Dissection 直接法によるものである)．Si n-MOSFET においては，導電に寄与するキャリアは電子であるため，正孔は少数キャリア

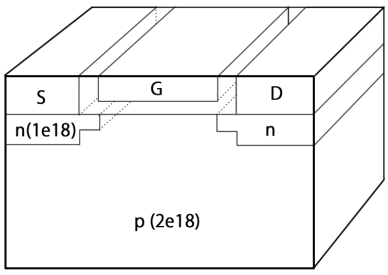

図 12. 3 次元 Bulk 型 Si n-MOSFET

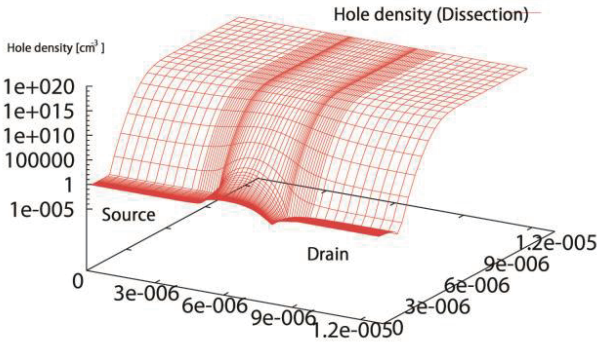

図 13. Bulk 型 Si n-MOSFET の正孔密度分布

となる．ドレイン印加電圧がゲート印加電圧より高い時，ドレイン近傍に空乏層(電子も正孔もほとんど存在しない領域)が出来る．

各方程式の離散化行列はそれぞれ，302,600 の自由度，1,820,696 の非零要素を有する．離散化行列は非対称であるので BiCGSTAB 法に Splitting-up 前処理を適用する(SPBiCGSTAB 法)．収束判定は，相対残差で 10^{-10} としている．図 14 に収束履歴を示す．前処理が効果的に働き反復計算は効率良く収束していることがわかる．ところが，空乏層付近の正孔密度の一次元分布(図 15)をみるとドレイン近傍の正孔密度分布は Dissection 直接法では解けているが，SPBiCGSTAB 法では解けていないことが分かる．

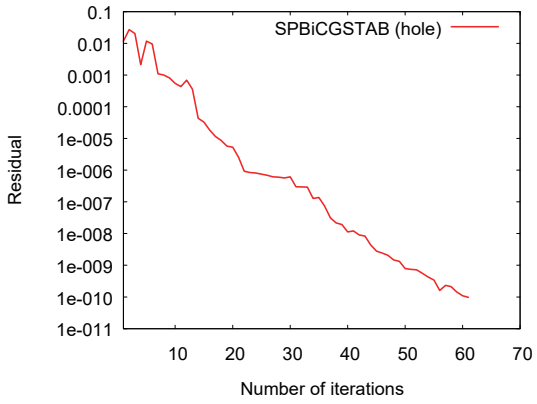

図 14. SPBiCGSTAB 法の相対残差による収束履歴

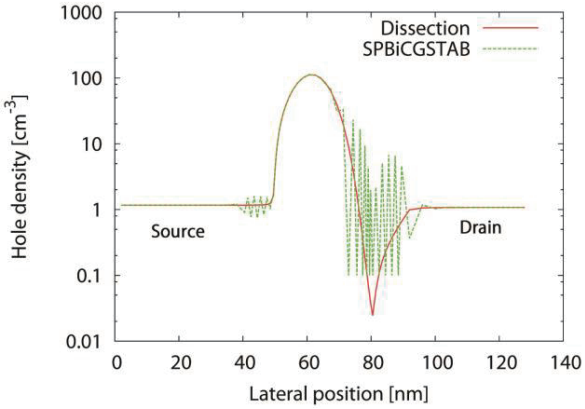

図 15. 正孔密度分布の行列解法による差異

これにより，少数キャリアの計算において Dissection 直接法の有効性が分かった．

6. 今後の展望

SPCG 法をはじめとする反復法と分散メモリ環境での直接法と部分構造反復法の部分問題解法としての利用の進捗状況と展望について述べる．

Dissection 直接法は 100 万自由度規模の疎行列を約 64GB の主記憶を用いて LDU 分解することが可能であるが，より大きな行列の解を得るためには分散計算環境を利用する必要がある．これには二つの方法がある．nested-dissection 法の 2 分木構造を複数ノードに割り当てる方法と，部分構造反復法などの領域分割法の部分問題ソルバーとして Dissection 直接法を用い，内部人工境界の問題を反復法により求解する方法である．

Dissection コードの分散メモリ環境への拡張では，4 ノード規模が最大となると思われる．これは直接法の計算量が未知数のおおよそ 2.5 乗程度になることからくる制約である．2 分木の根の位置にある Schur 補行列は左側と右側の 2 分木のタスクを分散し，左右の結果を足合わせることで並列処理は可能になる．しかしながら，図 2,3,5 から分かるようにアンバラスな 2 分木分割から生じたタスクの不均一さは LDU 分解のクリティカルパス以外のタスクの非同期実行により解消されている．分散メモリ環境で並列効率を保つためには，別のノードで実行されたタスクのデータを MPI の GET/PUT による非同期な受信と送信でアクセスすると共に，あるノード内にマッピングされているタスク実行が終了した際に別のノードではどのタスクが実行されていて，クリティカルパス外のどのタスクが待ち状態にあるかを効率的に知るアルゴリズムを確立する必

表 2. FETI 法による分散環境での並列実行

decomp.	LDU (sec.)	# iter.	time (sec.)	memory
2 × 2 × 2	19.3	52	421.6	32.37GB ×8
3 × 3 × 3	4.10	157	205.5	6.95GB ×27
4 × 4 × 4	1.75	64	27.9	2.75GB ×64

要がある

領域分割法の部分ソルバーとして Dissection 直接法を用いるアプローチでは，フランス，パリ第六大学，リオンス研究所・ONERA の François-Xavier Roux 教授が開発した FETI 領域分割法と，Fortran コードを用いて，弾性体問題に P2 有限要素を用いて離散化した，500 万自由度の問題を Cray XC30 の 16 ノード 224 から 448 コアを使用して計算をおこなった．複合材料の問題をモデル化した，弾性係数に 100 倍程度のジャンプを持つ問題である．表 2 に領域分割数と部分領域の LDU 分解にかかった時間，FETI 法の反復回数，計算時間，メモリ使用量を示す．それぞれの座標軸を 3 等分する分割の場合，領域間の境界が異なる複合材料を跨いでしまうため，FETI 法の反復計算が多くなる.問題に合った領域分割の形状を用いる必要があるが，大規模問題の効率的解法として期待できる．

一方，反復法に関しては，SPCG 法に対して，ベクトル演算器を持つ SX-ACE，ベクトル演算器を持たない Cray XC30 及び FX10 での実行結果およびその評価を行った．ここまでの評価によって，ICCG 法に代わる SPCG 法での分散メモリ型スーパーコンピュータでの実装は一定の成果を得たと考えられるが，配列再分散による通信時間をいかに削減するかが課題である．そこで，SPCG 法の実装方法として考えた 5 つの手法，すなわち，redist，pipe，p-scheme，NoZSolve，ISPCG の中で，通信（および計算）を最も削減した NoZSolve は，収束性能では他の方法では劣るが，いずれにおいても総合性能で優位ある．また，ノード内に OpenMP によるマルチスレッドを導入するか否かに関しては，最適なハイブリッド構成が存在し，計算機構成毎に決定する必要があることが分かった．現在のスーパーコンピュータは GPU などのメニーコアプロセッサを搭載することが多くなり，今後，そのようなアーキテクチャにおける最適な実装方法を検討することが課題となる．

ドリフト拡散モデルでは，NPN 型の半導体モデルの条件数が非常に大きくなることが知られており，反復法の収束が難しくなる．直接法での解法の可能性を検討したい．

7. 研究成果リスト

(1) 学術論文

(2) 国際会議プロシーディングス

1. Akiyoshi Wakatani, “Evaluation of Splitting-Up Conjugate Gradient Method on GPUs,” in Proc. of the 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, pp. 435-439, 2016.
2. Atsushi Suzuki, François-Xavier Roux, ″Implementation of a parallel sparse direct solver on vector architecture″, Sustained Simulation Performance 2016, M.M. Resch, W.Bez, E. Focht, N. Patel, H. Kobayashi eds. Springer, pp.99-108, 2016.
3. Akiyoshi Wakatani, ″An Incomplete Splitting-Up Conjugate Gradient Method for Parallel Computing″, the 2nd International Conference on Computer and Information Sciences, pp. 224-233, 2016.

(3) 会議発表(口頭，ポスター等)

1. Atsushi Suzuki, Francois-Xavier Roux, ″Dissection：A parallel direct solver in multi-core environment for sparse matrices by finite elements″, 計算工学講演会論文集 Vol.21, D-9-5, 2 pages, 2016.
2. 鍾菁廣, 小田中紳二,″不完全 HV 分解を伴った CG 法の並列計算″, 日本応用数理学会 2016 年度年会, pp. 238-239, 2016.
3. Atsushi Suzuki, Francois-Xavier Roux, ″Dissection：A direct solver with kernel detection for finite element matrices on multi-core supercomputers″, PMAA16, 2016.（口頭発表のみ）
4. 鈴木 厚, ″Dissection 直接法並列解法と FreeFem++での活用″, 日本応用数理学会 2016 年度年会, pp. 34-45, 2016
6. Shohiro Sho, ″A parallel semiconductor device simulation on SX-ACE″, NEC user group meeting, Osaka university, Japan, 2016.（口頭発表のみ）
7. 鍾菁廣, ″Restricted additive Schwarz 法の半導体におけるドリフト拡散方程式系への拡張″, 常微分方程式の数値解法とその周辺, 2016.（口頭発表のみ）

(4) その他（特許，プレス発表，著書等）