社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2008－HPC－118 (22)
2008／12／17

メニーコア向け並列有限要素法の実装と性能評価

荻野正雄† 河合浩志‡ 塩谷隆二§ 吉村忍‡

近年のマルチコア CPU で構成された PC クラスタや大型並列計算機上における並列有限要素解析の実行性能を改善するためには，従来のノード間あるいはノード内 CPU 間だけではなく，CPU 内コア間における並列効率が重要となってくる．本稿では，反復型領域分割法 (DDM) に基づく並列有限要素法として，キャッシュ階層を有効活用する実装を行うことで，並列効率の改善を示す．特に，DDM の各反復において右辺問題となる部分領域問題に対して，LDL 分解結果をメモリ保存して再利用する DS 型，毎回 LDL 分解を実施する DSF 型，反復法を用いる ISF 型を実装し，CPU 内コア間並列効率を評価する．

Implementation and Performance Evaluation of Parallel FEM for Many-Core Processors

Masao OGINO† Hiroshi KAWAI‡ Ryuji SHIOYA§ Shinobu YOSHIMURA‡

To improve the total performance of a PC cluster or a supercomputer using multi-core scalar CPU, not only the parallel efficiency between computational nodes or CPUs, but also prcessor cores of CPU is becoming more important than ever before. In this report, we propose a new design and programming style to optimize the performance of a parallel finite element code based on an iterative domain decomposition method on a multi-core CPU node by changing the design from memory access intensive approach to matrix storage-free one.

1 緒言

大規模有限要素解析を行なうためには，PC クラスタや超並列計算機 (Massively Parallel Processors: MPP) などの高速計算機が必要となる．近年，これらの多くがスカラー型並列計算機となっているが，これらが用いるスカラープロセッサに関して，その CPU チップ内部に複数のプロセッサコアを有するマルチコア CPU が急速に普及しつつある．また，CPU あたりコア数のさらなる増加 (メニーコア化 [1]) が将来的に見込まれている．これにより，従来のようにノード間あるいはノード内 CPU 間での並列性能を向上させるだけでなく，今後は CPU 内コア間の並列性能を引き出すことが重要と言える．コア間の並列性能を引き出すには，メモリバンド幅の制約を避けるため，コアごとに用意されたキャッシュメモリを有効利用できるようなプログラミング設計が必要となる．

一方，反復型領域分割法または領域分割法 (Domain Decomposition Method: DDM) は，有限要素法 (Finite Element Method: FEM) の並列化のための有効な手法の一つである．一般に DDM では，解析領域全体を複数の部分領域 (Subdomain) に分割し，これら領域ごとに仮の境界条件を与えてそれぞれ有限要素解析を行なう．そして，仮に与えられた境界条件による領域間の不釣合を反復法により補正していく．また，多様な並列計算機上において並列効率を高める実装方法として階層型領域分割法 (Hierarchical Domain Decomposition Method: HDDM)[2] が提案され，さらに BDD(Balancing Domain Decomposition)[3] や FETI-DP(Dual-Primal Finite Element Tearing and Interconnecting)[4] などの効率的な DDM 向け前処理法が開発されている．特に，BDD を大規模解析に適用するための手法として BDD-DIAG 法 [5] が提案され，2 億自由度規模構造解析などの実績 [6] が示されている．

本稿では，マルチコア CPU を搭載した PC クラスタや MPP 上において，DDM ソルバー，および BDD-DIAG 前処理を施した BDD-DIAG ソルバーの演算性能を効率化するための第一歩として，DDM における主要な計算負荷である領域ごと FEM 計算のマルチコア CPU 向け性能最適化を試みる．

DDM では，領域分割数が領域間不釣合いの収束性及び領域ごとの FEM 計算時間に影響するため，最適分割数が存在する．経験的に，総計算時間が最適化されるように，領域サイズはかなり小さめにとられる．従って，領域ごとの FEM 計算では直接法ソルバー (Direct Solver) が用いられることが多い．また，DDM の各反復過程では領域ごとの問題は右辺項のみが変化する右辺問題となるため，領域ごとの剛性行列および LDL 分解行列を最初に 1 度作成してメインメモリ上に保存することで，以降の反復過程ではそれらをメインメモリから読み出し，前進代入および後退消去のみで解を得ることができる．ここでは，この手法を Matrix Storage 型と呼ぶ．Matrix Storage 型では各 DDM 反復における計算量が小さいために計算時間の短縮が期待できるが，マルチコア CPU を用いる場合にはメモリアクセス部分が性能のボトルネックになっている．

これに対し，本稿ではこれを Matrix Storage-Free 型へと変更することを提案する．DDM 反復において，剛性行列などを毎回作成して FEM 計算を行うようにし，それらを全てキャッシュメモリ上で実行可能とすること

†九州大学
Kyushu University
‡東京大学
University of Tokyo
§東洋大学
Toyo University

で，CPU 内コア間の並列効率改善を可能とする．以降では，剛性行列および LDL 分解行列をメインメモリ上に記憶する実装を Direct Storage(DS) 型，毎回剛性行列作成と直接法による FEM 計算を行う実装を Direct Storage-Free(DSF) 型と呼ぶ．さらに，反復法によって FEM 計算を行う Iterative Storage-Free(ISF) 型もあわせて実装する．これにより，マルチコア CPU を用いた PC 上において，コア間並列効率向上を試みる．その結果として，DS と比較して実行時間の短縮と大幅な省メモリ化を同時に実現することを目指す．

以下ではまず，DDM について説明し，続いて本稿で提案する Matrix Storage-Free 型実装について述べる．次に，数値実験例としてマルチコア CPU を用いて各実装方法の性能評価を行う．

2 DDMアルゴリズム

本章では，3 次元構造解析を想定した場合における DDM アルゴリズムについて説明する．

DDM は解析領域を重なりのない部分領域に分割し，静的縮約を行った領域間境界自由度を反復法で解くアルゴリズムを基本としている．次の線形な連立一次方程式を考える．

$$Ku = f \tag{1}$$

ただし，K は剛性行列，u は未知変位ベクトル，及び f は荷重ベクトルである．次に，解析領域 Ω を重なりのない N 個の部分領域 Ω_1,...,Ω_N に分割し，領域間境界を $\Gamma = \cup_{i=1}^{N} \partial\Omega_i$ とする．ここで，R_i を全体領域から部分領域 Ω_i に自由度を制限する 0-1 行列とする．このとき，u_i を Ω_i における変位ベクトルとすると，$u_i = R_i u$ により得られる．剛性行列 K も同様に，$K = \sum_{i=1}^{N} R_i^T K_i R_i$ となる．

次に，u_i を領域内部自由度に関する u_{Ii} と領域間境界上自由度に関する u_{Bi} に分ける．

$$u_i = \begin{bmatrix} u_{Ii} \\ u_{Bi} \end{bmatrix} \tag{2}$$

また，領域 Ω_i に関する K_i，f_i，R_i は，

$$K_i = \begin{bmatrix} K_{IIi} & K_{IBi} \\ K_{IBi}^T & K_{BBi} \end{bmatrix}, \tag{3}$$

$$f_i = \begin{bmatrix} f_{Ii} \\ f_{Bi} \end{bmatrix}, R_i = \begin{bmatrix} R_{Ii} & 0 \\ 0 & R_{Bi} \end{bmatrix} \tag{4}$$

これより，式 (1) は次のように表される．

$$\begin{bmatrix} K_{II1} & \cdots & 0 & K_{IB1}R_{B1} \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & K_{IIN} & K_{IBN}R_{BN} \\ R_{B1}^T K_{IB1}^T & \cdots & R_{BN}^T K_{IBN}^T & K_{BB} \end{bmatrix} \begin{bmatrix} u_{I1} \\ \vdots \\ u_{IN} \\ u_B \end{bmatrix} = \begin{bmatrix} f_{I1} \\ \vdots \\ f_{IN} \\ f_B \end{bmatrix} \tag{5}$$

ただし，

$$K_{BB} = \sum_{i=1}^{N} R_{Bi}^T K_{BBi} R_{Bi}, \quad f_B = \sum_{i=1}^{N} R_{Bi}^T f_{Bi} \tag{6}$$

式 (5) より，部分領域内部の自由度を静的縮約することで部分領域間自由度に関する次式が得られる．

$$Su_B = g \tag{7}$$

ただし，

$$S = \sum_{i=1}^{N} R_{Bi}^T S_i R_{Bi}, \tag{8}$$

$$S_i = K_{BBi} - K_{IBi}^T K_{IIi}^{-1} K_{IBi} \tag{9}$$

$$g = \sum_{i=1}^{N} R_{Bi}^T (f_{Bi} - K_{IBi}^T K_{IIi}^{-1} f_{Ii}) \tag{10}$$

であり，S はシュアコンプリメント行列と呼ばれる．式 (7) を解くことで，領域間境界上自由度の変位解 u_B が得られる．領域内部自由度の変位解については，u_B を用いて次式により得られる．

$$u_{Ii} = K_{IIi}^{-1} (f_{Ii} - K_{IBi} R_{Bi} u_B), \quad i = 1,..., N \tag{11}$$

本稿で取り扱う構造解析では一般的に S は対称正定値となるため，式 (7) に前処理付き共役勾配 (Preconditioned Conjugate Gradient: PCG) 法を適用する．

ここで，PCG 法のアルゴリズムでは係数行列とベクトルの積を求める必要があるが，シュアコンプリメント行列 S は式 (8) のような形をしていることから，S を陽に作成するのは困難となる．しかし，

$$y = Sp \tag{12}$$

に対して，

$$\begin{bmatrix} x_i \\ \cdot \end{bmatrix} = \begin{bmatrix} K_{IIi} & K_{IBi} \\ K_{IBi}^T & K_{BBi} \end{bmatrix} \begin{bmatrix} 0 \\ -R_{Bi}p \end{bmatrix} \tag{13}$$

$$\begin{bmatrix} w_i \\ \cdot \end{bmatrix} = \begin{bmatrix} K_{IIi} & 0 \\ 0 & I \end{bmatrix}^{-1} \begin{bmatrix} x_i \\ \cdot \end{bmatrix} \tag{14}$$

$$\begin{bmatrix} \cdot \\ y_i \end{bmatrix} = \begin{bmatrix} K_{IIi} & K_{IBi} \\ K_{IBi}^T & K_{BBi} \end{bmatrix} \begin{bmatrix} w_i \\ R_{Bi}p \end{bmatrix} \tag{15}$$

$$y = \sum_{i=1}^{N} R_{Bi}^T y_i \tag{16}$$

は等価な処理であるため，S を陽に作成することなく，同様の演算結果を得ることが可能である．これらの式はそれぞれ，式 (13) はディリクレ境界条件として領域間境界に与えられた仮の強制変位境界条件の処理，式 (14) は領域内部自由度に関する有限要素解析，式 (15) は反力の計算，式 (16) は領域間境界上における反力の不釣合いの計算を意味する．DDM においては，式 (14)-(16) が主な計算負荷であり，これらを領域 FEM 計算と呼ぶ．

3 マルチコアCPU向けの実装

ここでは，Matrix Storage 型の実装である DS，および本稿で Matrix Storage-Free 型の実装となる DSF 並びに ISF について述べる．

3.1 Direct Storage (DS)

DDM では領域ごとに，式 (13) と式 (15) で表される剛性行列 K_i を用いた行列-ベクトル積，式 (14) で表される K_{IIi} を係数行列とした連立方程式求解が必要となるが，これらの行列は線形問題である式 (1) を解く DDM 反復中では変化しない．よって DS 実装では以下のように設計される．
(反復 1 回目)

1. 要素剛性評価と全体化を行い K_i を作成し，式 (13) を処理
2. ディリクレ境界条件処理を行って得られる K_{IIi} を LDL 分解し，前進代入と後退消去で式 (14) を処理
3. K_i を読み出し，式 (15) を処理

(反復 2 回目以降)

1. K_i を読み出し，式 (13) を処理
2. ${K_{IIi}}^{-1}$ を読み出し，式 (14) を処理
3. K_i を読み出し，式 (15) を処理

これにより，反復 2 回目以降の計算量を削減できるが，メモリ使用量は増加する．さらに，マルチコア CPU 上ではメモリアクセス部分が性能ボトルネックとなってくる．これを解決するために，各種行列を記憶しない実装を考える．

3.2 Direct Storage-Free (DSF)

DSF 実装では，DS と異なって DDM 反復ごとに剛性行列の作成と LDL 分解を行う．

1. 要素剛性評価と全体化を行い K_i を作成し，式 (13) を処理
2. ディリクレ境界条件処理を行って得られる K_{IIi} を LDL 分解し，前進代入と後退消去で式 (14) を処理
3. 要素剛性評価と全体化を行い K_i を作成し，式 (15) を処理

これにより，各反復における計算量は増加するが，領域サイズをある程度小さくすることで，比較的最近のマイクロプロセッサであれば，これらすべての計算をプロセッサコアごとのキャッシュメモリ上で行うことが可能である．なお，このときメインメモリから読み出すのは，要素コネクティビティや節点座標値などの領域に関するメッシュデータのみである．

3.3 Iterative Storage-Free (ISF)

ISF 実装では，DSF と同様に DDM 反復ごとに剛性行列の作成を行うが，式 (14) を反復法を用いて解く．

1. 要素剛性評価と全体化を行い K_i を作成し，式 (13) を処理
2. ディリクレ境界条件処理を行って得られる K_{IIi} を用いて，反復法で式 (14) を処理
3. 要素剛性評価と全体化を行い K_i を作成し，式 (15) を処理

ここで，3 次元構造解析では K_{IIi} は対称正定値となるため，PCG 法を用いる．反復法を用いるため，DSF に比べて領域サイズに対する汎用性が高い実装となる．なお，本稿では反復法が収束するまでの総計算量の観点から，前処理法として SSOR 前処理を用いた．さらに，計算速度向上のために，行列ベクトル積をベクトル・ベクトル演算に置き換える Eisenstat 技法を用いている．

3.4 要素剛性評価方法

プログラム設計を DS 型から DSF 型や ISF 型へ変更する際には，要素剛性行列の評価と全体剛性行列への全体化を DDM 反復ごとに行うことになる．特に，ディリクレ境界条件処理を行う式 (13) と反力計算を行う式 (15) で 2 回行う必要があり，さらに領域サイズを小さくするほど，直接法や反復法による式 (14) の求解に対してその比重が大きくなることが予想される．よって，DSF 型と ISF 型の実装においては，これらの高速化が重要となってくる．

本稿では，等方性材料の線形弾性問題に対して 3 次元ソリッド 4 面体 2 次要素を用いた場合に限定し，体積座標公式を用いた直接積分を行うことで，ガウスの数値積分など汎用的実装を行った場合に比べて演算量の削減を行った．

3.5 マルチメディア拡張命令の利用

DSF 型や ISF 型では，DDM における領域計算のほとんどがキャッシュメモリ上で実行されることが期待できる．よって，マルチメディア拡張命令 [7] を有するスカラープロセッサ上では，これを有効利用することで，計算速度の大幅な向上が期待できる．本稿では，Intel コンパイラや GCC などのマルチメディア拡張命令向けベクトル化をサポートするコンパイラを用い，ループの自動ベクトル化を行なう [8]．また，必要に応じてコンパイラディレクティブを挿入する．なお，ベクトル化が行われやすいように，ループの入れ替えやアンローリングを行った．

4 数値実験例

ここでは DDM を，DS 型，本稿で提案した DSF 型並びに ISF 型で実装し，マルチコア CPU 上における性能評価を行った．式 (7) に対する前処理としては BDD-DIAG 法を用いた．なお，性能評価に用いた計算機は Intel Xeon E5345 (quad-core x 2CPU, 2.33GHz, L2-4MB) である．コンパイラは Intel C/C++ Compiler 10.1.015 を用いた．並列実装は Flat MPI で行い，MPICH2-1.0.5 を用いた．また，解析モデルは Fig.1 に示す約 100 万自由度の線形弾性問題を用いている．

4.1 領域計算の性能評価

初めに，領域あたりの自由度数を変化させた場合における，領域 FEM 計算 (式 (13)-(15)) に要した計算時間について Fig.2 に示す．それぞれ，Xeon 上で 1 コア逐

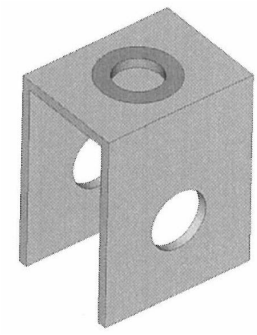

Fig. 1: Benchmark model with 1M DOFs

次処理，4 コア並列処理，及び 8 コア並列処理を行ったときの，1 つの領域 FEM 計算に要した時間の平均値を表している．ただし，DS 型では剛性行列作成や LDL 分解行列はメインメモリ上のものを利用する反復 2 回目以降における値である．また，領域計算の実装による性能差を明らかにするために BDD-DIAG 前処理は行っていない．

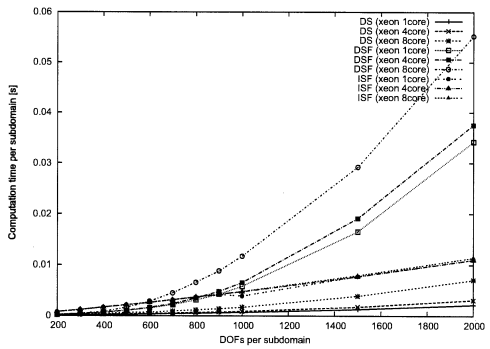

Fig. 2: Subdomain DOFs vs computation time per one subdomain FEM calculation using Xeon

図より，最も計算量が少ない DS が常に最速であったが，領域自由度が 400 以下では DSF は DS と同程度の計算時間であった．よって，領域自由度を十分に小さくできる場合は，メインメモリをほとんど使用しない DSF が有効であると言える．しかし，DSF では毎回直接法による連立方程式求解を行うため，領域自由度に応じて計算量が $O(N^{7/3})$ で増えることから計算時間が大幅に増加している．一方，反復法を用いる ISF では計算量は $O(N^{4/3})$ のため，領域自由度が 1,000 以上では DSF に比べて高速であった．

また，CPU 内コアを全て用いた場合，DS と DSF では領域自由度の増加に伴って，1 コアしか用いない場合に比べて計算時間が遅くなっている．これはキャッシュ溢れが生じたためと考えられる．特に，Xeon で 8 コア (2CPU) 使用した場合はその影響が大きい．一方，領域計算時のメモリ使用量が最も少ない ISF ではマルチコアを使用してもほとんど性能に差が生じず，領域自由度が 5,000 程度まではマルチコアによる影響は見られなかった．よって，領域自由度を大きく必要がある場合は，キャッシュ溢れを起こす閾値が大きい ISF が有効であると言える．

次に，8 コア使用時における，領域自由度に対する CG 法 1 反復に要した時間を Fig.3 に示す．図より，領域自由度が十分に小さいときは DSF を用いることで DS と同程度の性能が得られ，領域自由度が大きい場合は ISF を用いることで DS の 2 倍程度の計算時間であった．よって，提案手法は従来手法に比べてマルチコア CPU 向けに適していると言える．

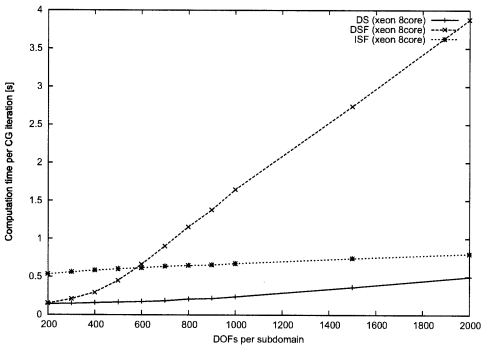

Fig. 3: Subdomain DOFs vs computation time per one CG iteration using eight cores of Xeon

4.2 BDD-DIAG 法の特徴

次に，本研究で用いる BDD-DIAG 法の特徴について示す．BDD-DIAG は理論的に，領域自由度を一定に保った場合，全体モデル自由度を増加させても一定の収束性が期待できる手法であり，大規模解析において実用性が高い手法である．

Fig.4 に領域自由度に対する BDD-DIAG の収束性を表しており，縦軸は相対残差ノルムが 6 桁小さくなった時点における反復回数である．ここで，BDD-DIAG はコースグリッド修正法に基づいているため，領域自由度の増加はコースグリッドが粗くなることを意味している．図より，領域自由度の増加に応じて反復回数が増加しているが，増加の割合は直線的であり，領域自由度を大きくせざるを得ない場合においても有効な手法であると言える．

Fig.5 に Matrix Storage 型 (DS) と Matrix Storage-Free 型 (DSF,ISF) のメモリ使用量を示す．領域の剛性行列や LDL 分解行列をメインメモリ上に記録する DS では，領域自由度の増加に伴って大きくメモリ使用量が増加している．よって，大規模解析に DS を用いる場合は，計算時間よりもメモリ使用量を重視した最適分割数を考える必要性がでてくる．一方，DSF や ISF では領域ごとの行列を記録しないため，計算時間を最適化する領域分割数を選択可能となる．なお，領域自由度が小さいときにメモリ使用量が大きくなっているのは BDD-DIAG 前処理行列によるものである．

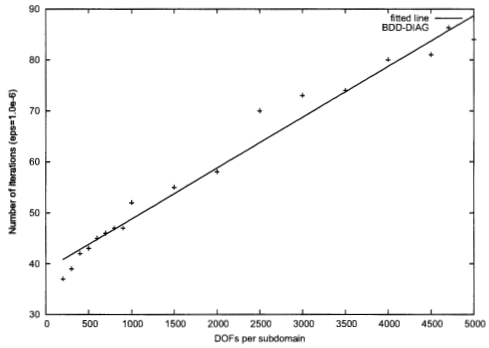

Fig. 4: Subdomain DOFs vs iteration counts using the BDD-DIAG method

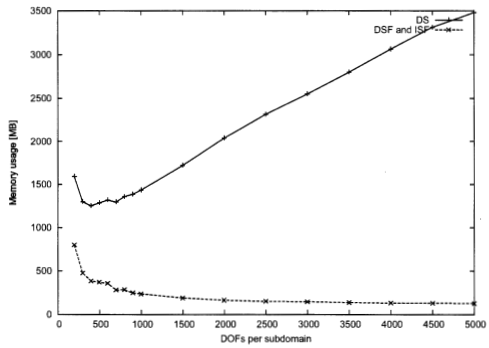

Fig. 5: Subdomain DOFs vs memory usage of the BDD-DIAG method

4.3 BDD-DIAG 法の性能評価

ここでは，本手法の実用性を評価するために，BDD-DIAG を用いた計算結果を示す．領域自由度を変化させたときの BDD-DIAG の総計算時間を，DS，DSF，ISF を用いた場合について Figs.6-8 に示す．DDM 反復あたりの計算時間性能と同じ傾向が見られ，Fig.6 より，DS では 4 コア使用時までは高速化が行われるものの，8 コア使用しても性能はほとんど改善されていない．一方，Fig.7 より，DSF では領域自由度が十分に小さいときは，8 コア使用時でも高い並列効率が得られている．また，Fig.8 より，ISF では領域自由度に依らずに高い並列効率が得られた．

最後に，8 コア使用時の計算時間の比較を Fig.9 に示す．並列効率は悪いが総計算時間としては計算量が少ない DS が最も速かった．しかし，計算量を大幅に増やした DSF と ISF でもキャッシュメモリ上の範囲であれば同程度の性能が得られることが分かった．

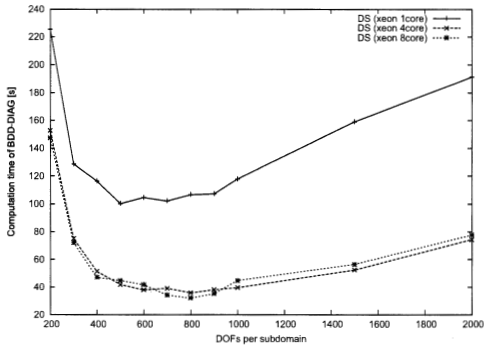

Fig. 6: Subdomain DOFs vs total computation time by Direct Storage type

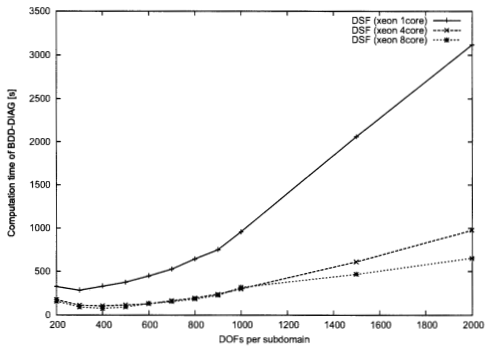

Fig. 7: Subdomain DOFs vs total computation time by Direct Storage-Free type

5 結論

将来のメニーコア CPU に向けた並列有限要素法として，Matrix Storage-Free 型の反復型領域分割法の実装を行った．領域分割法の主な計算負荷である領域 FEM 計算に関して，剛性行列や LDL 分解などの演算結果をメインメモリ上に記録して再利用することで計算量を減らす実装に比べて，計算量は増えるが全てキャッシュメモリ上の範囲内で毎回演算する実装を行うことで，CPU 内コアにおいても高い並列効率を得ることに成功した．特に，領域 FEM 計算として，領域あたり自由度を小さくできる場合は直接法に基づく実装を行い，領域あたり自由度を大きくする場合は反復法に基づく実装を行えば良い事が得られた．

今後は，その他のマルチコアアーキテクチャ上で性能評価を行うことで，提案手法の有効性を示していく．

参考文献

[1] 安藤秀樹, 新世代マイクロプロセッサアーキテクチャ 5–チップ・マルチプロセッサ, 情報処理, Vol. **46**, No. 10, pp. 1124–1130, 2005.

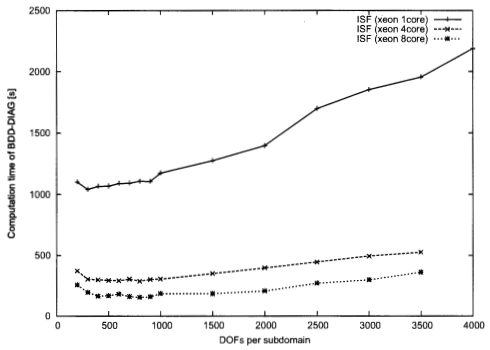

Fig. 8: Subdomain DOFs vs total computation time by Iterative Storage-Free type

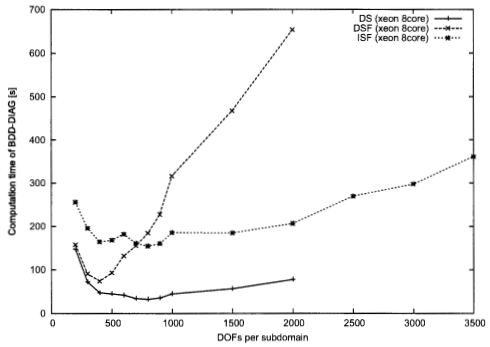

Fig. 9: Subdomain DOFs vs total computation time using eight cores of Xeon

[2] Yagawa, G. and Shioya, R.: Parallel Finite Elements on A Massively Parallel Computer with Domain Decomposition, *Computing Systems in Engineering*, Vol. **4**, pp. 495–503, 1993.

[3] Mandel, J.: Balancing Domain Decomposition, *Communications on Numerical Methods in Engineering*, Vol. **9**, pp. 223–241, 1993.

[4] Farhat, C., Lesoinne, M., and Pierson, P.K.: A Scalable Dual-Primal Domain Decomposition Method, *Numerical Linear Algebra with Applications*, Vol. **7**, pp. 687–714, 2000.

[5] Ogino, M., Shioya, R., and Kanayama, H.: An Inexact Balancing Preconditioner for Large-Scale Structural Analysis, *Journal of Computational Science and Technology*, Vol. **2**, No. 1, pp. 150–161, 2008.

[6] Ogino, M., Shioya, R., Kawai, H., and Yoshimura, S.: Seismic Response Analysis of Nuclear Pressure Vessel Model with ADVENTURE System on the Earth Simulator, *Journal of The Earth Simulator*, Vol. **2**, pp. 41–54, 2005.

[7] Slingerland, N.T. and Smith, A.J.: Multimedia Extension for General Purpose Microprocessor: A Survey, *Microprocessors and Microsystems*, Vol. **29**, pp. 225–246, 2005.

[8] Bik, A.J.C: The Software Vectorization Handbook, *Intel Press*, 2004.