境界要素法論文集 Vol. 20 (2003年12月), 論文 No. 18-031017 JASCOME

共有メモリ計算機における境界積分方程式法の並列化について

ON THE PARALLELISATION OF BIEM FOR SHARED MEMORY COMPUTERS

西村 直志[1)], 大谷 佳広[2)]

Naoshi NISHIMURA and Yoshihiro OTANI

1) 京都大学学術情報メディアセンター (〒606-8501 京都市左京区吉田本町, E-mail: nchml@media.kyoto-u.ac.jp)
2) 京都大学大学院工学研究科 (〒606-8501 京都市左京区吉田本町, E-mail: otani@mbox.kudpc.kyoto-u.ac.jp)

This paper discusses parallelisation strategies of BIEM for a large shared memory parallel computer. The conventional BIEM for two dimentional Laplace's equation with GMRES is parallelised with the help of OpenMP. The performances of the OpenMP and MPI parallelised codes are compared on a large SMP machine. Also the hybridisation of an MPI code with either OpenMP or automatic parallelisation is tested. It is concluded that all these approaches work equally well, while the parallelisation with OpenMP has a merit of simplicity.

Key Words: Parallelisation, SMP, OpenMP, MPI

1. はじめに

近年、PC clusterの普及にともなって、分散メモリ型並列計算機が身近なものとなった。一方、大型計算機の分野においては、地球シミュレータに代表されるベクトル並列計算機の流れに加えて、大規模SMP nodeを結合したスカラ並列計算機に移行しようとする流れが出てきている。例えば、京都大学学術情報メディアセンターにおいては、2004年3月に、従来のベクトル並列計算機Fujitsu VPP800を、スカラ並列計算機HPC2500に置き換える事が決定している[(1)]。京都大学の新システムは、128個のCPUからなる11台のSMP nodeをクロスバによって結合したものとなる。各node内のCPUは512GBのメモリを共有する事ができる。従来、メモリアクセスの競合などの理由からあまり大きなSMP機を作る事は現実的でないといわれていたが、最近の技術の向上により、今ではこの程度の規模のものが可能になっている。また、software的には、MPI[(2)]によるprocess並列に加え、OpenMP[(3)]や自動並列によるthread並列が提供される。今後運用が開始されると、node内では共有メモリ向きの並列化が、またnodeを跨いでは分散メモリ向きの並列化が行なわれるようになるものと考えられる。

従来、多くのPC cluster userは、あまり大きなSMPを扱う機会がなく、並列化はMPI (ないしはPVM)によって実現してきたものと考えられる。しかし、上述のような現状に鑑み、thread並列による並列計算を行ない得る環境が、ますます身近なものになるものと予想される。特に、thread並列のde facto standardであるOpenMPを用いた並列化は、逐次コードにdirectiveを書き込むだけで簡単に実現でき、MPIを用いた並列化より更に容易であるので、今後ますます普及するものと考えられる。

一方、境界積分方程式法(境界要素法)は、有限要素法や差分法の並列化に比べて、領域分割を行なう必要がない、機械的に分割するだけで良好なロードバランスを実現する事が可能な場合が多いなどの特徴を有している。このため、境界積分方程式法の並列化は一般に容易であり、境界積分方程式法は並列計算に向いた手法であるという事ができる。これまで多くの並列化に関する研究が行なわれて来ているが[(4, 5)]、その多くは分散メモリ計算機に対するものである。

このような状況を踏まえて、本研究では大型SMP機における境界積分方程式法の並列化法について検討する。具体的には、2次元Laplace方程式の外部問題における直接法の境界積分方程式法を、反復法であるGMRESによって解く事を想定し、これをOpenMPや自動並列によって並列化した時の計算効率を調べる。また、これを従来のMPIによる結果と比較する。更にMPIとOpenMPのハイブリッドによる計算効率についても検討する。

2. 境界積分方程式法

考える領域Dは、自分自身と交わらない境界∂Dの外側であるとする(Fig.1)。外部領域Dにおける2次元Laplace方程式のNeumann問題

$$\Delta u = 0 \quad \text{in } D \tag{1}$$

$$\frac{\partial u}{\partial n} = q \quad \text{on } \partial D \tag{2}$$

$$u(\mathbf{x}) = O(\log|\mathbf{x}|) + o(1) \quad \text{as } |\mathbf{x}| \to \infty \tag{3}$$

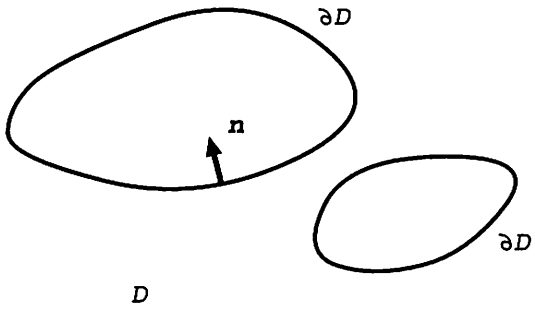

Fig. 1　領域

を考える。ここに q は ∂D 上で与えられた関数であり、$\mathbf{n}$ は ∂D の外向き単位法線ベクトルである。

この問題の解は、積分方程式

$$\frac{u(\mathbf{x})}{2}+\int_{\partial D}\frac{\partial G(\mathbf{x}-\mathbf{y})}{\partial n_y}u(\mathbf{y})dS_y \quad = \quad \int_{\partial D}G(\mathbf{x}-\mathbf{y})q(\mathbf{y})dS_y \qquad \mathbf{x}\in\partial D \tag{4}$$

を解く事に帰着される。ここに、G は Laplace 方程式の基本解

$$G(\mathbf{x})=-\frac{1}{2\pi}\log|\mathbf{x}| \tag{5}$$

である。

式 (4) における u を境界要素補間し、選点法によって離散化すれば、

$$A\mathbf{x}=\mathbf{b} \tag{6}$$

の形の代数方程式が得られる。これを適当な方法で数値的に解けば、元の境界値問題の解が構成できる。本論文では、式 (6) の解法として、反復解法の一つである GMRES[6, 7] を用いる。

3. 境界積分方程式法の並列化

前節で述べた境界積分方程式法は、係数行列 A と定数ベクトル $\mathbf{b}$ の作成、及び、GMRES による求解の 2 段階に分かれる。A、$\mathbf{b}$ の作成については前節で述べた通りである。一方、非対称行列を有する線形方程式の反復解法である GMRES[6, 7] については、簡単のために前処理を含まないアルゴリズムを書き出してみると以下のようになる。

$n=-1$

$\mathbf{r}_0=\mathbf{b}-A\mathbf{x}_0$ (7)

$\gamma=\|\mathbf{r}_0\|$ (8)

$\mathbf{v}_1=\mathbf{r}_0/\gamma$ (9)

$E_{-1}=\gamma$

do while $E_n\geq\epsilon$

$n=n+1$

$\mathbf{w}_n=A\mathbf{v}_n$ (10)

$h_{in}=\mathbf{w}_n\cdot\mathbf{v}_i \quad (i=1,2,\ldots,n)$ (11)

$$\mathbf{V}_{n+1}=\mathbf{w}_n-\sum_{i=1}^{n}h_{in}\mathbf{v}_i \tag{12}$$

$h_{n+1\,n}=\|\mathbf{V}_{n+1}\|$ (13)

$\mathbf{v}_{n+1}=\mathbf{V}_{n+1}/h_{n+1\,n}$ (14)

$\min_{y_i}\|\gamma\delta_{i1}-\sum_{j=1}^{n}h_{ij}y_j\|$ を解く (15)

$(i=1,2,\ldots,n)$

$E_n=|\gamma Q^n_{n+1\,1}|$

enddo

$$\mathbf{x}_n=\mathbf{x}_0+\sum_{i=1}^{n}\mathbf{v}_i y_i \tag{16}$$

3.1. 分散メモリ環境における並列化

まず、境界積分方程式法を分散メモリ環境において並列化する事を考える。この際、標準的な GMRES の並列化を考慮すれば次のような方法が自然である [5]。まず、係数行列は

$$A=\begin{pmatrix} A_0 \\ \hline A_1 \\ \hline \vdots \\ \hline A_{n_p-1} \end{pmatrix} \tag{17}$$

のように行方向に均等に分割し、i 番目の process に A_i を割り当てる ($i=0\ldots n_p-1$)。ここに n_p は process 数である。各 process は割り当てられた係数行列成分のみ計算し、記憶する。また、定数ベクトル $\mathbf{b}$ についても同様に分割し、各 process は割り当てられた成分のみ計算して記憶する。次に、GMRES においても、任意の N ベクトルを

$$\begin{pmatrix} \text{process 0} \\ \hline \text{process 1} \\ \hline \vdots \\ \hline \text{process } n_p-1 \end{pmatrix} \tag{18}$$

のように n_p 個のブロック分割する。この時、上記の GMRES の各ステップは次のように分類する事ができる。

1. 式 (17) の分割に従って、通信なしに自然に並列化できる部分: (9), (12), (14), (16)。
2. 通信を要する部分: (8), (10), (11), (13)。
3. あまり大きな計算に (なって) はならない部分: (15)

このうち、1. は各 process で並列実行、2. は各 process の寄与分を並列計算後に通信、3. の部分は比較的小さい計算なので、全 proces で同じ計算を各々実行する事によって十分な並列化効率が得られる。CERFACS の GMRES[8] のような reverse communication を用いた code を用いると、以上を MPI で実現する事は非常に容易である [5]。

3.2. 共有メモリ環境における並列化

同じ問題を共有メモリ環境において並列化する場合は次のようになる。まず、係数行列は各 thread で共有すればよく、

分割する必要はない。ただ、計算においては i 番目の thread に A_i に相当する部分を計算させる事によって work share すれば良い。定数ベクトル $\mathbf{b}$ についても同様である。GMRES においても、上記の 1., 2. は各 thread で work share、3. の部分は master thread のみで実行すれば良い。これを OpenMP で実現する場合、逐次 code に数行の directive を書き加えるだけで良く、並列化に要する手間は MPI による並列化よりも更に少ない。

3.3. 分散共有メモリ環境における並列化

分散共有メモリ環境における並列化を行なった場合、各 process で実行する行列・ベクトルに関する演算は、更に thread 並列により work share する事ができる。このような 2 階層の並列化は、一旦 MPI で並列化した code に OpenMP の directive を書き加える事によって簡単に実現できる。また、MPI の code に自動並列化を用いる事によって更に簡便に実現する事も可能である。

4. 計算例

以上述べてきた並列化の効果を京都大学学術情報 media center の HPC2500 で検証した。構成は 96 個の Sparc64V を CPU とする SMP 1 ノードである。

対象とする領域は Fig.2 に示すように、半径 a、中心間隔 $2.1a$ の $10\times10 = 100$ 個の円孔の外部である。境界条件として、Neumann data

$$\frac{\partial u}{\partial n} = n_2 \quad (\text{法線ベクトルの第 2 成分})$$

を与えた。各円孔の表面は 100 個の境界要素に分割したので、全自由度は 10,000 である。

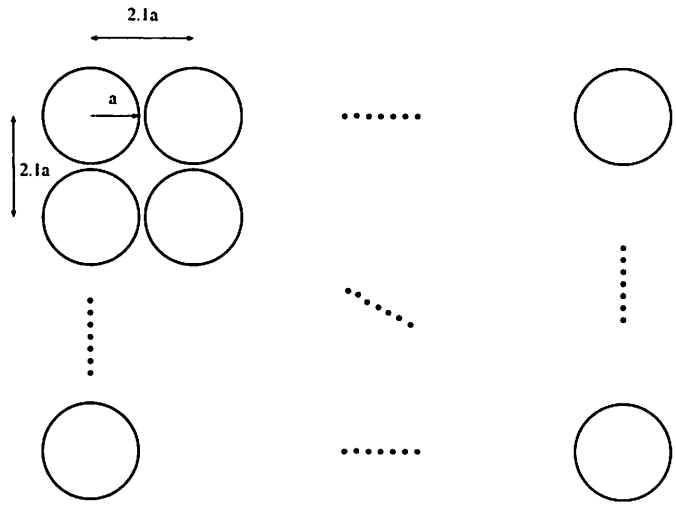

Fig. 2　円孔群

離散化においては区分一定形状関数を用い、積分は全て解析的に行なった。また使用した GMRES は CERFACS の code である [8]。収束判定条件は残差 < 残差の初期値 $\times10^{-5}$ とし、Krylov 空間の次元は 10 とした。繰返し回数は 11 であった。

Fig.3 には、MPI で並列化した時と OpenMP で並列化した時の計算時間を示した。システムの制限により、OpenMP では 90 並列まで、MPI では 80 並列までとした。結果には大差はなく、使用した環境では process 並列も thread 並列も効果はほぼ同程度である事がわかる。Fig.4 には、Fig.3 の結果

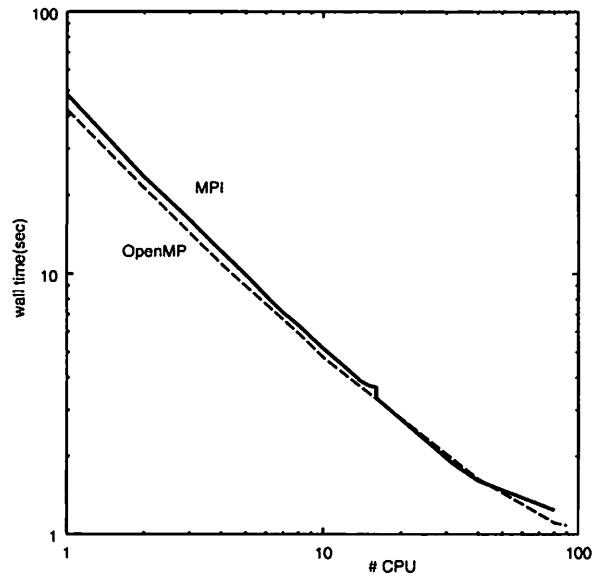

Fig. 3　MPI 及び OpenMP による計算時間

の Speedup (計算時間/1CPU での計算時間) を示した。MPI の結果が CPU 数 16 で不連続であるのは、それより少ない CPU では会話型処理を行なったからである。それ以外の結果は全てバッチ処理による。図より MPI の speedup の方が OpenMP よりやや大きい値を示している事がわかるが、いずれにせよ 30 CPU 程度を越えるとメモリアクセスの競合が問題になり始める事がわかる (SMP では MPI の通信もメモリへの読み書きで実現している)。

Table.1 には MPI-OpenMP ハイブリッド並列時の計算時間を示す。使用した全 CPU 数は 32 で固定した。使用した環境では MPI の process 数を増やす方が OpenMP の thread 数を増やすより効果的であるという結果となった。同様な傾向は 16CPU を用いた計算においても見られた。MPI の process 数が少ないと、並列化の効果に見合わない overhead が発生するためにこのような結果となると解釈する事もできよう。Table.2 には、Table.1 と同様な比較を MPI-自動並列

Table 1　MPI-OpenMP ハイブリッド並列時の計算時間

# process × # thread	wall time(sec)
1×32	3.036
2×16	2.460
4×8	2.239
8×4	1.998
16×2	1.862
32×1	1.592

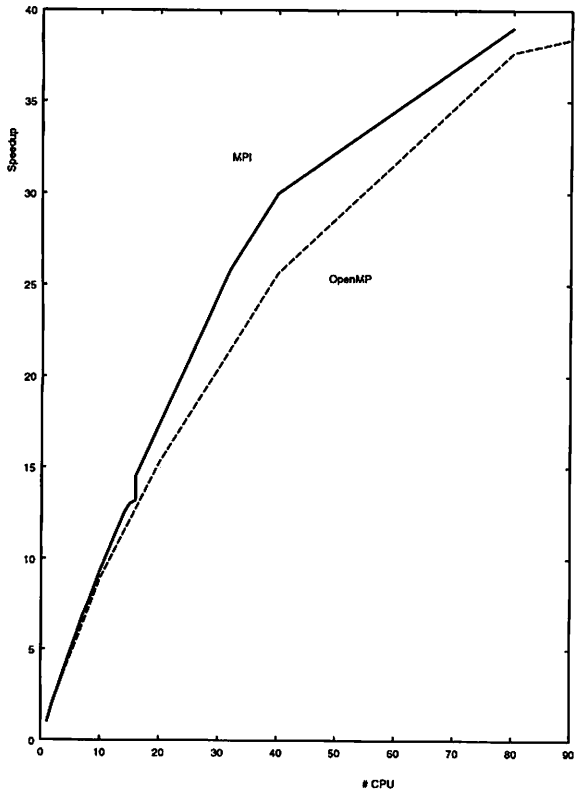

Fig. 4　MPI 及び OpenMP による speedup

ハイブリッドによって行なった結果を示す。使用した富士通製の compiler では自動並列と OpenMP は独立したもので、自動並列を OpenMP への preprocessor として実現しているわけではない事に注意する。結果は手で directive を書き込んだ MPI-OpenMP の場合に比べるといずれも遅くなっているが、単に compile 時に option をつけるだけで並列化を実現できる事を考えると、自動並列化も十分実用価値はあるものと言えよう。

Table 2　MPI-自動並列ハイブリッド時の計算時間

# process × # thread	wall time(sec)
1×32	3.738
2×16	3.041
4×8	2.686
8×4	2.523
16×2	2.178
32×1	1.626

5. 結言

- 共有メモリ型計算機における境界積分方程式法の並列化を検討し、OpenMP による並列化を行なった。この結果、従来分散メモリ型の計算機で開発してきた MPI による並列化よりかなり簡単に並列化が実現でき、しかも並列化効率も MPI によるものと殆んど変わらない事が分かった。また、MPI と OpenMP の併用による並列化も可能である事が確認できた。ただし、ハイブリッド並列に実用的な意味があるのは SMP クラスタ (grid を含む) においてであり、今後、そのような環境におけるテストを行なう予定である。
- 境界積分方程式法は並列計算機向きであると同時にベクトル計算機向きの解法であるともいえる。このため、ベクトル並列計算機において非常に高速な解析が可能である。一般にベクトル並列計算機においては、ベクトル長を長く取りつつ分散メモリ型の並列化を行ない、スカラ並列計算機ではキャッシュミスを少なくする事を意識しつつ並列化を行なうが、境界積分方程式法の場合は、これらの要求が必ずしも対立しないので、本論文で使用した MPI の code はベクトル並列計算機においてもそのまま高い演算効率を示す。実際、本論文ではスカラ並列計算機において最大 90CPU を用いた計算を行なったが、そのいずれもが、京都大学学術情報 media center の今回リプレースされるスーパーコンピュータである vpp800 の 5 台並列の結果 (0.97sec) を越える事はできなかった。

参考文献

(1) 金澤正憲 (2003), 超並列スーパーコンピュータへの移行 — VPP800 から HPC2500 —, 京都大学学術情報メディアセンター全国共同利用版広報、**2**, 260–264.

(2) **http://www-unix.mcs.anl.gov/mpi/**

(3) **http://www.openmp.org/specs/**

(4) 例えば、Engineering Analysis with Boundary Elements, **18-3** (1996), **19-1** (1997).

(5) 樫山和男, 西村直志, 牛島省 (2003), 並列計算法入門, 丸善

(6) Saad Y and Schultz MH (1986), A generalized minimum residual algorithm for solving nonsymmetric linear systems, *SIAM. J. Sci. Stat. Comput.*, **7**, 856–869.

(7) 藤野清次, 張 紹良 (1996), 反復法の数理, 朝倉書店.

(8) **http://www.cerfacs.fr/algor/Softs/GMRES**