ハイパフォーマンス
コンピューティング 73－4
（1998．10．9）

自動分散型オブジェクト指向シミュレーションシステムの実行効率

上原 均　　畠山正行
茨城大学工学部情報工学科
〒 316-8511 茨城県日立市中成沢町 4-12-1

本報告では，我々が以前に開発した自動分散化システム ADS によって，逐次型オブジェクト指向シミュレーションシステムから自動生成される自動分散型オブジェクト指向シミュレーションシステムの実行効率を検証する．この検証では数値流体力学のオブジェクト指向シミュレーションシステム 2 例を自動分散化してその計算時間等を計測し，分散型オブジェクト指向シミュレーションシステム構築時のユーザへの負担を重視して検証を行った．この検証結果と自動分散化でユーザの負担を大きく軽減している点を考慮して，自動分散型オブジェクト指向シミュレーションシステムが十分な実行効率を達成している，という結論を得た．

Performance of Automatic Distributed Object-Oriented Simulation System

Hitoshi Uehara　　Masayuki Hatakeyama
Department of Computer & Information Sciences, Faculty of Engineering, Ibaraki University
4-12-1 Nakanarusawa-cho, Hitachi-city, 316-8511 Japan

We have developed the automatic distribution system ADS that can automatically generate the automatic distributed object-oriented simulation system from sequential one. The aim of this report is to verify the performance of the automatic distributed object-oriented simulation system generated by ADS. In this verification, we have generated two samples of the automatic distributed object-oriented computational fluid dynamics simulation system, and have measured these performances. We have verified the performance from the viewpoints of the user's load and others. From considering the reduction of the user's load by ADS and the verification-result, we have concluded that automatic distributed object-oriented simulation system have realized the sufficient performance.

1　はじめに

科学技術計算の分野では，近年オブジェクト指向分析/設計に基づいたオブジェクト指向シミュレーションシステム（以降，OOSS と略）の構築例が確実に増加しつつある [1, 2]．また OOSS の「高い開発効率」「高い再利用性」「実行時の柔軟性」等の特長から，この傾向が今後も継続することが予測できる．

これら OOSS の特長は特に開発時に発揮されるが，実用的な運用，特に大規模計算でのそれを行うには従来の手続き型システム（Fortran プログラム等）と同様にシステムの分散/並列化が必要である．ただしシミュレーションを利用しているユーザ（以降，ユーザと記す）の目的はシミュレーション結果を得ることであり，分散/並列化は単なる実用的な実行効率を得るだけの手段に過ぎない．そのため，ユーザは自分自身でシステムを分散/並列化することを嫌い，計算機側でシステムを分散/並列化すること（完全自動化）を理想と考えている．しかし完全自動化は実現困難 [3] なので，ユーザの負担を十分に軽減した（半）自動分散/並列化こそがユーザの理想に近く，かつ達成可能なものであると考える．

我々はこのようにユーザの負担を軽減して逐次型 OOSS から分散型 OOSS を自動生成できるシステムとして，自動分散化システム ADS を開発した [4, 5, 6]．そしてこの ADS で自動生成された分散型 OOSS のことを自動分散型 OOSS と呼んでいる．この ADS を利用することで分散型 OOSS 構築時のユーザへの負担を軽減できることは既に判明している [4, 5, 6] が，それとは別に ADS で自動生成される自動分散型 OOSS の実行効率も実用的観点から見て重要なポイントである．

そこで本報告では，ADS が生成する自動分散型 OOSS の実行効率を検証する．具体的には数値流体力学の OOSS2 例を ADS で自動分散化し，その実行効率を検証する．この検証では実行効率だけではな

く，それを実現する際のユーザへの負担も重視した．これは分散/並列化ソフトウェア技術は実行効率の観点だけではなく，ユーザの負担や開発期間等を含む分散/並列型システムの開発・実行過程全体を通して総合的に評価されるべきである，という観点にたつものである．この実行効率とそれを実現する際のユーザへの負担の比を考慮して，自動分散型 OOSS の実行効率が良好であることを本報告で示す．

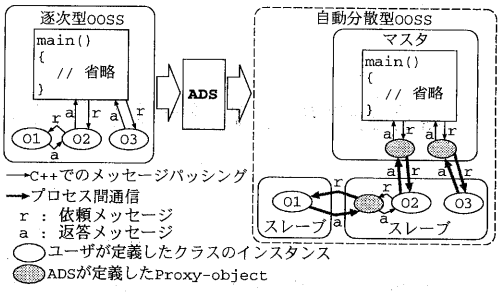

図 1: ADS によるシステム構造の変更

2 OOSS と自動分散化

本章では自動分散型 OOSS の実行効率検証のための前提知識として，OOSS とその自動分散化システム ADS，逐次型 OOSS から ADS によって生成される自動分散型 OOSS の概要を述べる．ただし OOSS の具体例は文献 [1, 2]，ADS の詳細説明は文献 [4, 5, 6] の参照を前提に簡略化している．

まず OOSS については我々は，OMT 法 [7] 等のオブジェクト指向分析/設計方法論に基づいて構築され，かつオブジェクトのカプセル化/情報隠蔽が十分されているもの，と緩やかに定義している．そして自動分散化の研究ではオブジェクトの粒度が分散計算に適していることを条件として追加し，独立計算を行う計算要素（＝オブジェクト）間でメッセージパッシングしながらシステム全体の処理が進む陽解法に類するシステム構造の OOSS を対象としている．なお OOSS の記述言語は C++を想定している．

この OOSS の根本にはオブジェクト指向パラダイムがあるが，このオブジェクト指向パラダイムにおけるオブジェクトがデータ（＝属性）とその処理（＝メソッド）を集約した自己完備な計算主体 [8] であり，分散/ 並列計算に適したメッセージパッシングをインタフェースとして備えることから，オブジェクト指向パラダイムはオブジェクト単位の分散/並列計算に適していると文献 [8][9] で指摘されている．また分散型アーキテクチャとオブジェクト指向アーキテクチャの類似点が多いことも文献 [9] で指摘されている．そしてこのような特長を持つオブジェクト指向に基づいた OOSS にも同様にオブジェクト単位での分散/並列計算への適性が指摘できる．

この OOSS の分散/並列計算への適性に着目して，OOSS の自動分散化を図ったのが自動分散化システム ADS [4, 5, 6] である．ADS では拡張型 proxy-object 方式で OOSS の自動分散化を実現している．この拡張型 proxy-object 方式は，通常の分散オブジェクト間での通信処理詳細を隠蔽するだけの proxy-objetct[10][11] を機能拡張したもので，1) 分散オブジェクトの自動配置，2) メソッドの並行実行を実現する非同期通信，3) 並行処理の同期制御，4) 通信デッドロックの回避，等の様々な分散計算に必要な機能を proxy-object 内部で実現している．JAVA の分散実行系である HORB[11] の horbc では通常の proxy-object クラス生成が主な機能であるが，ADS では機能拡張された proxy-object を生成して更にそれを逐次型 OOSS へ自動的に組み込むことでマスタ・スレーブ方式の自動分散型 OOSS を自動生成する [4, 5, 6]．また proxy-object のみでは実現できない分散システムに必要な処理も同時に ADS が自動分散型 OOSS に組み込んでいる．

この ADS による逐次型 OOSS から自動分散型 OOSS へのシステム構成変更の例を図 1に示す．この例の逐次型 OOSS は三つのオブジェクトとそれ以外の要素（この例では main() 関数）から構成されるが，自動分散型 OOSS では proxy-object によるオブジェクトへのアクセスの仲介が追加されている．この proxy-object 内部で分散型システムに必要な分散処理（分散通信処理や分散配置等）を実現することで OOSS の自動分散化を実現している．

またこの自動分散化処理時のクラス定義変更の例を図 2に示す．この例で示すようにユーザが逐次型 OOSS 用に定義したクラスが，自動分散型 OOSS ではそのクラス名を変えられている．そしてクラス名を変えられたユーザ定義クラスの代わりに proxy-object クラスが定義されている．このクラス名の変更によって逐次型 OOSS ソースコードを大幅変更すること無く proxy-object を導入でき，逐次型 OOSS

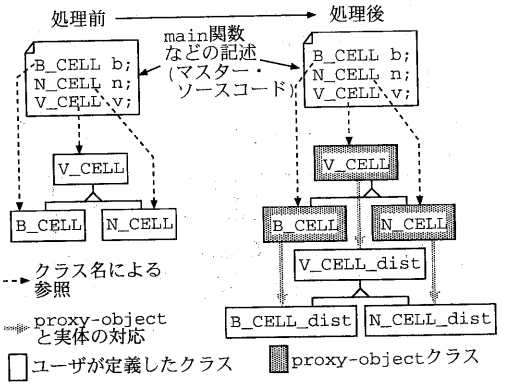

図 2: ADS によるクラス名変更

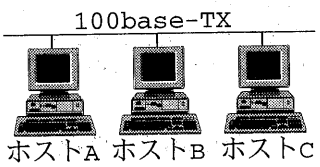

図 3: 実行効率を検証した計算機環境

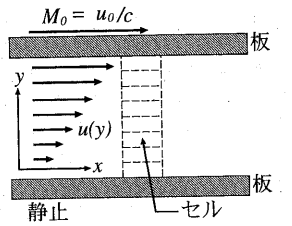

図 4: 一次元流れ問題（Couette 流）

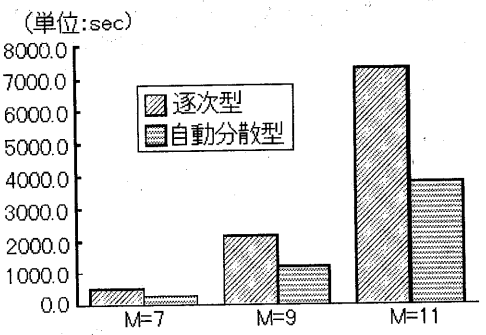

図 5: DVO 法 OOSS での計算時間の比較

からの自動分散型 OOSS へのスムーズな自動変換を実現している．またこのクラス名変更と同様に逐次型 OOSS のクラス定義ファイルの名前も変更し，proxy-object クラスの定義ファイルを生成している．ファイル名が変更された元のファイルは proxy-object クラスの定義ファイル中に読み込まれる．

3 自動分散型 OOSS の実行効率

本章では，数値流体力学での分散計算に適した離散化速度座標法（Discrete Velocity Ordinate scheme, 以降 DVO 法と略）[12] の OOSS と通常は分散計算に適さないと考えられる Roe 法 [13] の OOSS の二つを自動分散化して，その実行効率を調査する．この調査での動作環境は図 3のような SPARCStation20 三台を接続した LAN で，マスタをホスト A，スレーブをホスト B，C に一つずつ配置した．なお比較対象の逐次型 OOSS はホスト B で駆動した．

3.1 離散化速度座標法 OOSS での検証

ここで扱う DVO 法の OOSS では図 4に示す一次元流れ問題（Couette 流）を解く．この OOSS では計算対象の一次元空間を微小空間（セル）に区切って数値計算を実現しており，OOSS ではこのセルをオブジェクトとしている．この DVO 法全体の計算負荷と通信負荷は DVO 法で用いる離散速度分布における各座標軸毎のメッシュ数 M とセル数 N で定まり，計算負荷は $O(N \cdot M^6)$，通信負荷は $O(N \cdot M^3)$ で計算負荷が通信負荷より十分大きいので DVO 法は分散計算に適している．

この DVO 法 OOSS でのクラス構成は既に図 2に示しており，流体セルを N_CELL, 境界を B_CELL, それらのスーパークラス V_CELL を定義している．この DVO 法 OOSS に対する ADS による自動分散化処理はメソッド並列実行についての対話処理を含めて約 1 分程度で終了した．

この DVO 法 OOSS を流体セル数 10，境界セル数 2，マッハ数 3.0，クヌーゼン数 Kn=0.1,dt=0.0025 という条件で，離散速度空間のメッシュ数 M を 7, 9, 11 と変化させて，逐次型と自動分散型各々で 10 回ずつ試行した．各セルは Burnett の解 [14] で初期近似値を与えている．なお境界条件として上下の両境界で Y 方向の Flux バランスを取っている．終了条件は規定の回数 (20 回) 数値計算処理を繰り返すこととした．この条件下での自動分散型では，スレーブ 1 に図 4の Couette 流の下半分を表すオブジェクト群が，スレーブ 2 に残りの上半分を表すオブジェ

表 1: DVO 法 OOSS での数値計算の所要時間

	M=7	M=9	M=11
逐次型	23.79	108.29	364.33
自動分散型	13.56	58.77	190.13

(単位：sec/1step)

表 2: DVO 法 OOSS での通信の所要時間

	M=7	M=9	M=11
逐次型	0.05	0.14	0.20
自動分散型 (並行処理)	0.22	0.28	0.44
自動分散型 (逐次処理)	0.72	0.77	0.90

(単位：sec/1step)

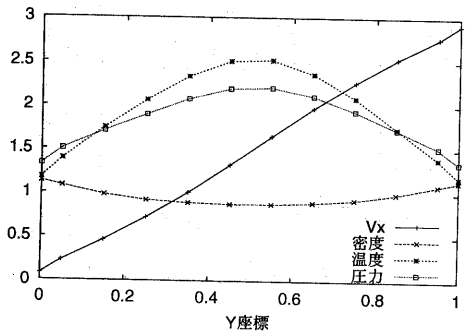

図 6: DVO 法 OOSS の計算結果

クト群が自動配置された．この実行の平均計算時間は図 5のとおりであり，自動分散型ではスレーブ数 2 での実行で逐次型と比べて計算時間の約 40–48%短縮，実行速度の約 1.7–1.9 倍の高速化を実現した．

この DVO 法 OOSS での計算負荷と通信負荷をその処理に要した時間で示す（表 1，表 2）．オブジェクト間通信は離散速度分布 F の転送だけなので，その所要時間を示す．自動分散型 OOSS ではその通信処理を並行処理しているが，逐次処理した場合の通信時間も示す．逐次型と自動分散型 (逐次処理) の通信時間の差から分散通信オーバーヘッドは約 0.70sec 前後と推定されるが，並行処理の場合には約 0.20sec 前後まで抑えられている．総合的に見て計算時間の短縮に比べて通信時間の増大は問題にならない．

また同条件で実行した手続き型システムと比較して，この DVO 法 OOSS では記憶領域で約 1.2 倍，計算時間は約 1.03 倍必要であることを確認した．この記憶領域の増加はクラス定義の効率化で低減できる．

なお自動分散型 DVO 法 OOSS でパラメータを変えずに定常状態 (ループ回数 2000) まで計算した結果を図 6に示す．この結果は逐次型での結果と一致し，

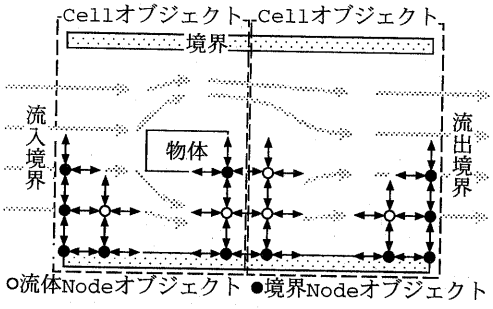

図 7: 二次元流れ問題（領域分割済み）

自動分散化による誤差は無い．この結果は Burnett の解と比較すると誤差があるが，これは逐次型 DVO 法 OOSS に内在する計算誤差であり，ADS による自動分散化に起因するものではない．

3.2 Roe 法 OOSS での検証

ここでは図 7のように縦横 $N \times M$ の 2 次元数値風洞を計算する Roe 法 OOSS を自動分散化する．この Roe 法 OOSS では図 7に見られる直交格子の交点を Node オブジェクトとしてモデル化し，各オブジェクトでの数値計算には Roe 法を用いている．この Roe 法 OOSS は格子数 20,000 点の計算で，効率化された手続き型システムの約 1.06 倍の計算時間が必要であるが，最適化すれば効率化された手続き型システムとほぼ同等な実行効率が達成できる [15].

この Roe 法 OOSS 全体の計算負荷と通信負荷は同じオーダー ($O(N \times M)$) であり，そのままでは分散計算に適さないので，自動分散化する前に図 7のように領域分割して分散通信負荷を軽減する．分散計算での領域分割は通信負荷を軽減するための一般的な技法であり，この Roe 法 OOSS のように細粒度な計算主体間で直接相互に分散通信させる方がむしろ稀であろう．

ここでは分割した領域を Cell クラスとして定義し，数値風洞内で近接する Node オブジェクト群を Cell オブジェクトに集約する．Cell オブジェクトを跨いだ Node オブジェクト間通信は Cell オブジェクト間で分散通信することで分散計算の効率向上を図る．領域を分割するにはオブジェクトの通信範囲等のユーザの数値シミュレーションに関する専門知識が不可欠なので自動化はほぼ不可能であり，この領域分割はユーザが行わなければならない．しかし従

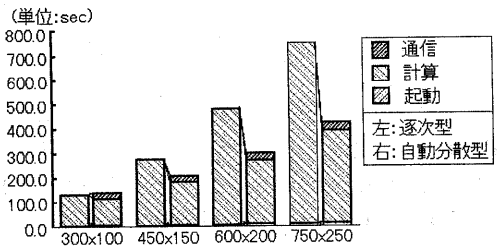

図 8: Roe 法 OOSS での計算時間の比較

来の PVM 等でのそれと違って分散処理の記述は全て ADS で自動生成されるので，集約オブジェクトを定義してそれで Node オブジェクトを管理するようにするだけの簡単な作業である．実際，この Roe 法 OOSS での領域分割型への変更は約 90 分程度で完了した．この領域分割した Roe 法 OOSS ソースコードを ADS で自動分散化し，その所要時間は DVO 法 OOSS と同様に約 1 分程度であった．

この自動分散化した Roe 法 OOSS を，縦横比 1:3 の風洞に円柱状の物体を挿入し，その風洞自体を均等に 2 つに領域分割する設定で駆動させた．この際の計算パラメータはマッハ数 Mach=3.0，dx=0.01，dy=0.01，レイノルズ数 Re=500，dt=0.0003 で，初期状態は密度 $\rho = 1.0$，速度 $u = 1.0$，$v = 0.0$，エネルギー $e = 1.0$ とした．そして境界条件は X=0.0 は一定，それ以外の境界は隣接する流体側のグリッドと等しいとし，物体境界は u, v 共に 0.0 でその他の物理量は隣接する流体側と等しいとした．なお終了条件は規定のループ回数 (100 回) 数値計算を繰り返すものとした．この条件下での自動分散型 OOSS では Cell オブジェクトの二つの内一つがスレーブ 1 に，もう一つがスレーブ 2 に配置された．この条件下での逐次型および自動分散型の平均実行時間 (各 10 回試行) は図 8に示した通りで，ここでは内訳として起動，数値計算，通信各々に要した処理時間も示す．この結果から 300×100 では自動分散型は逐次型とほぼ同じ程度だが，750×250 の状態で逐次型と比べて計算時間の約 44%短縮，実行速度の約 1.78 倍高速化を実現している．分散通信オーバーヘッドで通信時間は増加しており，自動分散型の起動時間もプロセス数の多い分だけ若干増加しているが，それらによる計算時間の増加以上に数値計算時間が短縮できているので，総合的な実行効率は向上している．

なお 450×150 の風洞を同じ計算パラメータでほぼ定常状態 (ループ回数 100,000) になるまで計算した結果の密度分布を図 9に示す．この結果は逐次型の結果と一致し，自動分散化による数値誤差は無い．

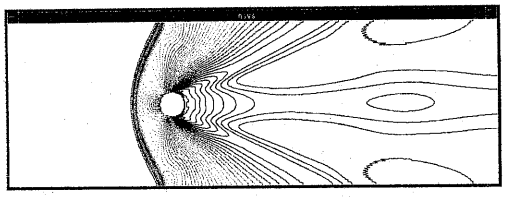

図 9: Roe 法 OOSS の実行結果一例

4 考察

自動分散型 OOSS の総合的な実行効率は，3章での DVO 法 OOSS，Roe 法 OOSS の両方でスレーブ数 2 の状態で逐次型 OOSS の約 1.7–1.9 倍という実行速度を実現したことから，良好な実行効率を達成していると判断する．なお計算負荷が通信負荷より十分大きい限り，スレーブ数を増加させることで更なる実行効率向上も望める．

また逐次型 OOSS の実行効率が手続き型システムのそれとほぼ同等である点，自動分散型 OOSS がスレーブ数からみて良好な実行効率を実現している点から，自動分散型 OOSS が手続き型の分散型システムに実行効率で大きく劣ることは考えにくい．

次に詳細を検討すると，まずオブジェクトの分散配置は特に問題はない．自動分散型 OOSS ではオブジェクトは負荷均等に，かつ計算空間で隣接するものには同じスレーブに配置する．3.1節の配置がその好例である．ただし負荷が一部へ集中したり動的変動する場合には問題があるが，どちらも OOSS では発生しにくいので問題となることは少ないと判断した．

そして通信オーバヘッドは 3章の実測から判断して，数値計算時間の短縮に比べて通信時間の増大が十分小さいので問題ない．同一スレーブにあるオブジェクト間の通信も proxy-object が仲介するが，それによる処理の遅れが 500 万回のメソッド実行で約 3 秒と十分小さいので問題ない．

また proxy-object のデータサイズは 3章の DVO 法で 88byte，Roe 法で 80byte 程度と，分散計算が必要な計算規模の OOSS での使用メモリ（数 Mbyte 以上）等より十分小さいので，proxy-object によるメモリ消費は問題にならない．

なお最適化については自動分散化処理時の対話処理で並行処理の効率化を図れる．ただし我々の経験では逐次型OOSSソースコードを整理して並行性を高めたほうが，結果的にはより良好に最適化できた．

ここまでは実行効率について考察してきたが，この実行効率を実現するためのユーザの負担も考察する．まず自動分散型OOSSの基となる逐次型OOSSソースコードであるが，これは通常のC++ソースコードなので，その作成のための特別な負担は無い．そしてこの逐次型から自動分散型を生成するADSの駆動も3章で述べたように数分程度で終了する．従来のPVM等を用いた開発には数日から数週間の期間を要したことを考えると，この開発期間短縮には大きな実用的価値がある．またADSによる自動分散化処理時におけるメソッド並列実行のための対話処理では分散システムの知識は必要ない．同様に領域分割が必要な場合でも分散処理の記述はADSで全て生成されるので，ユーザには分散システムの知識は要求されない．これらのことを総合的に考慮して，ADSによる自動分散化処理におけるユーザの負担はかなり軽いと判断した．なおこれらの詳細については文献[4, 5, 6]も参照願いたい．

以上のことをまとめて，実行効率とそれを実現する際のユーザへの負担の比という観点から考えて，自動分散型OOSSの実行効率には重大といえる問題点は無いと判断した．勿論より一層の効率向上が望ましいが，3章の例や個々の機能テストを通して，ユーザの負担を大幅に軽減する自動分散化のメリットを打ち消す程の実行効率上の重大な問題点は見い出されていない．

5 まとめと今後の課題

本報告では，我々が開発したOOSSの自動分散化システムADSが生成する自動分散型OOSSの実行効率を実例で検証し，良好な実行効率が得られることを示した．この実行効率はユーザ自身が最適化した分散型システムのそれよりは劣るかもしれないが，実行効率とそれを実現する際のユーザへの負担の比を考慮して自動分散型OOSSは十分な実行効率を達成していると判断した．なお我々の行ったサーベイの範囲ではADSのように，逐次型OOSSから一般的なLAN環境下で駆動する分散型OOSSを分単位の作業時間で実現でき，かつ本報告で述べたような実行効率を達成できるOOSSの自動分散化技術は見い出せていない．

今後の課題としては，分散オブジェクトの動的初期配置の効率を向上させる手法，および多次元問題への対応の強化が挙げられる．通信方式の最適化としてInspector/Executor法[3]の導入も検討する．

参考文献

[1] 日本機械学会. 第6回計算力学講演会講演論文集. 日本機械学会, 1993.

[2] Satya N. Atluri, Genki Yagawa, and Thomas A. Cruse, (eds). *Computational Mechanics'95*, Vol. 1, chapter 2, pp. 14–68. 1995.

[3] 湯浅太一, 安村通晃, 中田登志之（編）. はじめての並列プログラミング. 共立出版, 1998.

[4] 上原均, 畠山正行. 自動分散型オブジェクト指向シミュレーションシステムの生成. 情報処理学会研究報告 97-HPC-67, pp. 103–108, 1997.

[5] 上原均, 畠山正行. 自動分散型オブジェクト指向数値計算プログラムの生成システム. 情報処理学会研究報告 97-SE-117, pp. 69–76, 1997.

[6] 上原均, 畠山正行. オブジェクト指向シミュレーションの自動分散化の実現と評価. 日本シミュレーション学会論文誌「シミュレーション」, 1998年12月（予定，採録決定済み）.

[7] James Rumbaugh, 羽生田栄一（監訳）. オブジェクト指向方法論OMT. トッパン, 1992.

[8] 所 真理雄, 松岡 聡, 垂水 浩幸（編）. オブジェクト指向コンピューティング, 岩波書店, 1993.

[9] P. Wegner, 尾内理紀夫（訳）. はやわかりオブジェクト指向. 共立出版, 1992.

[10] Marc Shapiro. Structure and encapsulation in distributed systems : the proxy principle. *ICDCS*, pp. 198–204, 1986.

[11] S. Hirano. HORB: Distributed Execution of Java Programs. *World Wide Computing and its Applications*, pp. 29–42, 1997.

[12] M. Hatakeyama, S. Sugano, and A. Takaada. Applications of function approximation methods to computational schemes for discrete boltzmann equation. *Rarefied Gas Dynamics 19*, Vol. 1, pp. 64–70, 1995.

[13] 棚橋隆彦. 計算流体力学. アイピーシー, 1994.

[14] Zhong, X. and Koura, K. Comparison of solutions of the burnett equations, navier-stokes equations, and dsmc for couette flow. *Rarefied Gas Dynamics 19*, Vol. 1, pp. 354–360, 1995.

[15] 鈴木 俊人. オブジェクト指向に基づく数値流体シミュレーション. 茨城大学大学院理工学研究科修士学位論文, 1998.