タスク並列スクリプト言語用アプリケーション層ライブラリの実現

西川雄彦[†]　阪口裕輔[†]　田中一毅[†,☆]
大野和彦[†]　中島浩[††]

近年，複雑な物理計算を要する処理などにおいてPflops以上の計算能力が求められている．そのため，インターネット，Grid，P2P，汎用プロセッサなどを用いて100万台規模の並列処理を実現するためのタスク並列スクリプト言語MegaScriptの開発が進められている．MegaScriptは設計上，非常に柔軟で強力な並列記述機能を提供する．その反面，学術計算などを目的とするエンドユーザは並列処理の知識が乏しく，使いこなせないおそれがある．

本稿では，需要が高く，並列に処理することで性能の向上をはたせるアルゴリズムをMegaScriptのライブラリとして実装することで，並列処理に関する知識の乏しいエンドユーザでも簡単に並列処理を実現できるようにした．

実装したアプリケーション層ライブラリの性能評価を行った結果，ライブラリが実用レベルに達していることを確認する事ができた．

Application Layer Libraries for a Parallel Script Language

Takehiko Nishikawa,[†] Yusuke Sakaguchi,[†] Kazuki Tanaka,[†,☆]
Kazuhiko Ohno[†] and Hiroshi Nakashima[††]

Some applications, such as physical computations, requires great computing power over Pflops. A parallel script language *MegaScript* is under development to meet such demands. MegaScript provides very flexible and powerful faculty for parallelization, and can handle mega-scale parallel processing. However, most end-users in these application fields will not have enough knowledge in parallel processing and bringing out high performance using MegaScript is difficult for them.

Therefore, we implemented generic algorithms a MegaScript libraries for such end-users.

We evaluated these Libraries by applying to some problems. The result shows the libraries can be applied to practical use.

1. はじめに

ゲノム情報/生命工学などのライフサイエンス，環境・気象シミュレーションなどの複雑な物理系が絡み合う系，大規模な都市における地震や山火事などの災害シミュレーションなどでは1Pflops以上の計算能力が期待されている．そのため，100万台規模のプロセッサを用いたメガスケールコンピューティングが必要とされている．現在世界最速のスパコンである地球シミュレータは5120CPUを使って35.8Tflopsを達成しているが，膨大なコストと巨大な施設が必要になるという問題点があり，スパコンの延長でPflopsを実現するのは極めて困難である．このためコモディティな技術を用いた「低電力化とモデリング技術によるメガスケールコンピューティング」の研究が行われている．メガスケールコンピューティング環境は，性能の異なる計算機やネットワークの集合としてシステムを構築するため，それらの資源を効率よく利用する実行システムが必要である．そこで，メガスケールコンピューティング向けの言語としてタスク並列スクリプト言語MegaScriptを開発している[1)]．

MegaScriptは設計上，非常に柔軟で強力な並列記述機能を提供する．その反面，学術計算などを目的とするエンドユーザは並列処理の知識に乏しく，この機能を使いこなせないおそれがある．

そこで我々は，エンドユーザにとって需要が高く，また並列に動作させることで性能の向上をはたせる機能をライブラリとして設計・実装することで，並列処

† 三重大学
Mie University
†† 豊橋技術科学大学
Toyohashi University of Technology
☆ 現在，株式会社　昭和システムエンジニアリング
Presently with Showa System Engineering Corporation

理に関する知識を必要とすることなく簡単に並列処理プログラムが作成可能となるようにした．

2. タスク並列スクリプト言語 MegaScript の概要

メガスケールの並列性を持つプログラムを一から記述するのは非常に困難である．その解決策として，逐次プログラムや部分問題を並列化した小規模な並列プログラムを組み合わせることにより大規模並列性を引き出す「多重並列モデル」が考えられている．

MegaScript は逐次プログラムや並列プログラムをタスクとして扱い、複数のタスクを制御して並列実行するタスク並列言語である．各タスクは独立性の高い処理モジュールであり,「ストリーム」と呼ばれる論理的な通信路を用いることでタスク間のデータ送受信を行うことができる．

現在の MegaScript 処理系はマスターホスト 1 台と，それ以外のスレーブホストとで計算機ネットワークを構築する[2)]．マスターホストでは，Ruby インタプリタを内蔵した MegaScript ランタイムが起動し，実行するスクリプトの内容に従ってタスクをスレーブホストに起動する．

MegaScript はメガスケール規模でのプログラミングを実現可能とし，広域に分散し，複数の並列計算機を含むヘテロな環境下での動作を目指している．

2.1 アプリケーション層ライブラリ

文献[1)] に記載されているように，MegaScript は設計上，非常に柔軟で強力な並列記述機能を提供する．その反面，学術計算などを目的とするエンドユーザは並列処理の知識に乏しく，この機能を使いこなせないおそれがある．そこで，エンドユーザにとって需要が高く，また並列に動作させることで性能の向上をはたせる機能をライブラリとして設計・実装することで，並列処理に関する知識を必要とすることなく簡単に並列処理プログラムが作成可能となる．本稿では，パラメータサーベイ・ライブラリ，GA ライブラリ，木探索ライブラリの 3 つのアプリケーション層ライブラリを設計・実装し評価を行った．

3. ライブラリの設計

3.1 パラメータサーベイ・ライブラリ

3.1.1 パラメータサーベイの概要

パラメータサーベイとは，大量の同じタスクを異なるパラメータで実行することで，個々のタスクが与えたパラメータに対してどのように挙動するかを求めることを目的とする計算である．この計算は数値解析，

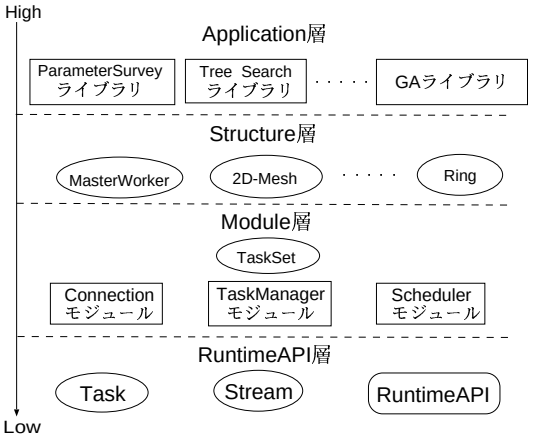

図 1 ライブラリ階層構造

天文学，物理学分野，原子炉のシミュレーションなどで発生する．パラメータサーベイで扱える問題は，各タスク間に依存関係がなく並列処理の効果を引き出しやすい．しかし，実際にパラメータサーベイを並列に行うには，多数のタスクを複数の計算機上でプロセスとして立ち上げる必要がある．さらに，各計算機上の計算量が均等になるようにタスクを分配しなければ，途中で仕事をしない計算機が生じて効率が低下してしまう欠点もある．

3.1.2 パラメータサーベイ・ライブラリの設計

基本的な動作は以下のようになっている．

(1) 並列動作させるタスク (計算タスク) のパラメータ設定，計算タスクの出力を処理する後処理タスクの設定をユーザが行う．

(2) 設定された内容に従って，ライブラリが計算タスクを複数の計算機に割り当てて並列に実行し，結果を後処理タスクに渡す．

ここで，各計算ホストの処理する計算量の差から生じる処理効率の低下を防ぐために動的負荷分散を行う必要がある．

3.1.3 パラメータサーベイ・ライブラリの実装

動的負荷分散機能を MegaScript で記述するとタスクネットワークは図 2 のようになる．

このタスクネットワークでは，MegaScript はマスタータスクキューを持っており，ユーザがインターフェース部で設定した計算タスクをこのキューに格納している．各計算ホストへは，ユーザが指定した計算タスクではなく管理タスクを配置する．この管理タスクが計算ホスト上での計算タスクを管理する．管理タスクは MegaScript に計算タスクの要求を行いマスタータスクキューから計算タスクを送信してもらい，

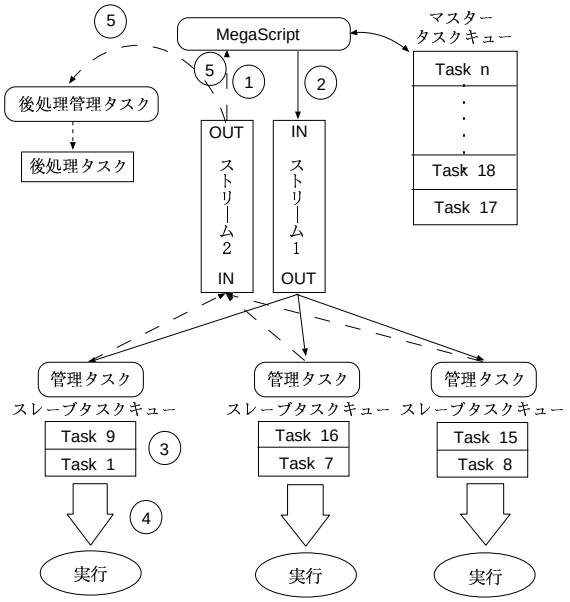

図 2 動的負荷分散を実現するタスクネットワーク

受信したものを実行する．実行後は結果を後処理タスクへ送信し，次の計算タスクの要求を行う．このように動作させることで，計算ホストは常に処理を行っている状態になり，計算タスクの負荷のばらつきによる処理量の偏りをなくすことができる．

3.2 GA ライブラリ

3.2.1 GA の概要

遺伝的アルゴリズム (GA)[3] は，自然界の生物進化に着想を得て，そのメカニズムを 工学的にモデル化した最適化手法であり，組み合わせ最適化問題やスケジューリング問題など，他の手法では解を求めることが困難な問題を解くために利用されている．

3.2.2 GA ライブラリの設計

GA の問題点の一つとして，高い計算負荷による計算時間の増大がある．これを解決する方法として，並列計算機を利用した GA の並列化に関する研究が数多く報告されている．並列化手法の一つである島モデルの通信は，各島間において一定の間隔で一部の個体データを送信するため，他の手法と比べて通信量が少なく，高い並列化効率が期待できる．また，GA の一連の処理を一つのタスクとし，それを複数起動させることで簡単に並列化が可能となる．さらに，それぞれの計算機において大きな負荷の差が発生しにくく，計算機の利用効率が高い．以上の利点を考慮して，島モデルを用いて設計を行う．

処理の流れは，移住操作を除いて単一母集団 GA と同様であり，また各分割個体集団に対する GA の処理は同一のものである．そのため，移住操作を加えた GA 処理タスクを一つ定義し，そのタスクを複数個生成することで並列化が可能となる．移住に関しては，タスク間通信を可能とするストリームを用いて行う．

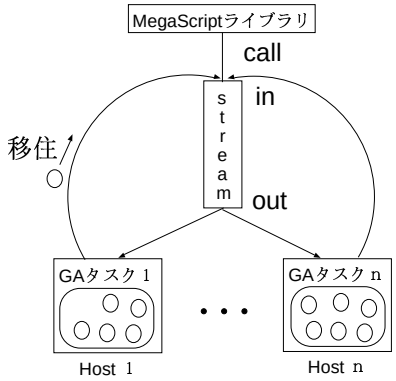

図 3 MegaScript を用いた島モデル

ライブラリの構成としては，タスクの生成と制御を行う MegaScript ライブラリと，GA の処理を行う C のプログラムからなる．MegaScript ライブラリはタスク，ストリームの定義を行い，タスクネットワーク構造を構築する．GA 処理プログラムは MegaScript によりタスクとして生成され，各ホストにおいて独立して処理を行う．

GA の処理においては，問題に依存する部分があり，初期個体集団の生成，型変換，評価といった処理は問題固有のものである．そのため，ユーザ側に適用する問題に応じてコールバック関数としてこれらの処理を定義してもらう必要がある．その関数群をライブラリ側の GA プログラムとリンクさせることで，GA タスクプログラムが構成される．ライブラリ側の GA プログラムでは移住を含めた並列化に関する部分と，問題依存しない操作に関する部分を定義しておく．そのため，ユーザ側には問題固有の操作を記述してもらうだけで並列処理に関する知識や細かな制御を意識することなく並列化が可能である．

3.2.3 GA ライブラリの実装

島モデルを MegaScript で記述したモデルを図 3 に示す．

MegaScript では，ストリームに送信されたデータは出力端に接続されているタスク全てにマルチキャストされるため，1 対 1 通信ができない．よって，ストリームの出力端に全タスクをつなぐことで送信側はどのタスクに対してもデータを送ることができるようにした．また，入力側に関しても全タスクを接続し，一

つのストリームを全タスクで共有する形で通信を行う．

移住時の通信に関しては，ストリームを介して行う．並列に動作している各タスクにおいて移住を行う周期に達すると，それぞれが標準入出力ライブラリの関数を用いてストリームに個体情報を流す．

問題点として，送受信時のオーバヘッド増大による並列化効率の低下がある．これは，個体送信時に 1 個体づつ送信しているため通信回数が大幅に増えたことによる．これを解決する方法として，GA 処理プログラムに Wrapper プログラムをかぶせ，送信データを宛先ごとにまとめてからストリームに流すようにする．

以上のような操作により，ユーザは意識することなく通信回数の大幅な減少が可能となりオーバヘッドの低減がはかられる．

3.3 木探索ライブラリ

3.3.1 木探索ライブラリの概要

木探索問題は比較的問題サイズが大きく並列に動作させることで性能の向上が期待できる．このため MegaScript のアプリケーション層ライブラリの一つとして有用であると考えられる．設計した木探索ライブラリは，問題サイズが大きくなりやすいゲーム木探索問題を対象としている．

3.3.2 木探索ライブラリの設計

基本的にパラメータサーベイ・ライブラリと同様のタスクネットワークを構築する．図 2 の MegaScript 部分では，ゲーム木探索の流れを制御する．管理タスク部分では，ストリームから流れてきたゲームの局面を受け取り，その局面の子供ないし評価値を MegaScript へ返す展開機能を受け持つ．

局面を展開するとき，すべての手を展開するたびにホストプログラムへ返していては．先読みする深さを増やすごとに爆発的に通信が増加し，実行時間に悪影響を及ぼす．この悪影響を最小限にするため，管理タスクに，ある一定の深さに達するとそこから探索打ち切り深さまで逐次処理を行う逐次切り替え機能を追加した．

逐次切り替え深さはライブラリ側で設定を行い，ユーザが自由に変更可能である．

3.3.3 木探索ライブラリの実装

並列 $\alpha\beta$ カットの仕組みを図 4 に示す．マスターホストでは，木構造を管理しており，各スレーブホストからの評価値を基に $\alpha\beta$ カットアルゴリズムを用いてマスターホスト上で管理している木の上層部の枝刈りを行っている．

逐次処理に切り替え後は，スレーブホスト内でローカルな枝刈りを行っている．しかし，スレーブホスト内のローカルな枝刈りでは，他のスレーブホストに $\alpha\beta$ カットの評価値が伝搬されないので，枝刈り効率が低下する問題がある．これを改善するため，あるスレーブホストで打ち切り深さまで探索が完了すると $\alpha\beta$ カットの評価値をホストプログラム経由で他のスレーブホストに送信する．他のスレーブホストでは，送られてきた $\alpha\beta$ カットの評価値を閾値として用い，探索不要な部分の枝刈りを行う．これにより枝刈り効率の低下を抑えることができる．

例えば，図 4 において，スレーブホスト 0 上で (1) のようなローカルな枝刈りが行われる．次に，スレーブホスト 0 での木探索の評価値が送信され，マスターホスト上で (2) ような枝刈りが行われる．このとき刈り取られなかった部分は，評価値とともにスレーブホストへ送られる．この場合，スレーブホスト 1 上で，送られてきた評価値を基に (3) のような枝刈りが行われる．以上のように並列 $\alpha\beta$ カットを行っている．

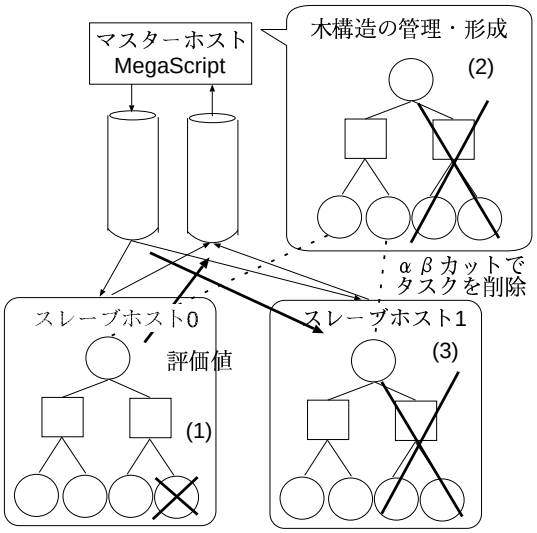

図 4 並列 $\alpha\beta$ カットの仕組み

4. 評　　価

4.1 評価環境

評価用の実行環境として，表 1 に示す PC9 台を Gigabit Ethernet で接続した PC クラスタを利用した．

4.2 パラメータサーベイ・ライブラリの評価

4.2.1 評価内容

実装したパラメータサーベイ・ライブラリの実行性能を評価するために，SimpleScalar[4] 上で gcc シミュレーションを行った．SimpleScalar は out of order 実行のプロセッサをモデル化したもので，パラメータを

表 1　評価用 PC 性能

PC	Dell OptiPlex GX260
CPU	Pentium4 2.4GHz
Memory	512MB
OS	RedHat Linux 7.3
Network	Gigabit Ethernet

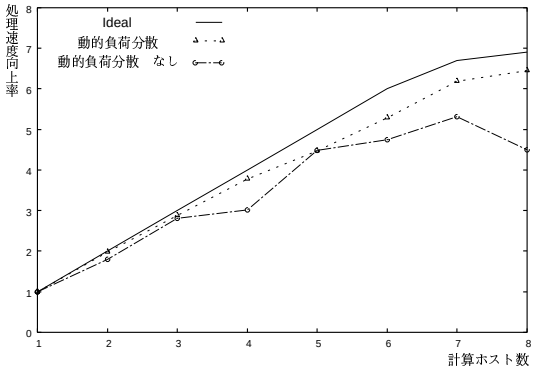

図 5　評価結果：速度向上率グラフ

変更することで性能見積もりを行うシミュレータである．このシミュレータはプロセッサ研究の分野で広く使われている．この評価で，以下の項目について確認する．

- 逐次処理と比較した場合の処理速度向上率
- 実用レベルのシミュレーションを行えるか否か

評価条件は以下の通りである．

- 1000〜8000 行のソースコードファイル各 4，計 32 ファイルをコンパイル (32 タスク)
  - 動的負荷分散を行った場合
  - 動的負荷分散を行わなかった場合

4.2.2　評 価 結 果

評価結果を図 5 に示す．Ideal を示す線が傾いているのは，問題サイズの組み合わせにより完全に均等に分割できないためである．動的負荷分散を行わずに実行した場合，計算ホスト数を増やしても実行性能が低下することが確認された．一方，動的負荷分散を行った場合は計算ホスト数の増加に伴い，性能向上が得られ，実用に耐えうる実行性能があることが確認できた．

4.3　GA ライブラリの評価

4.3.1　評 価 内 容

実装した GA ライブラリの性能評価を行うためにナップサック問題 (KP) と巡回セールスマン問題 (TSP) を用い，複数のパラメータにおいて評価を行う．その結果より，速度向上率について検証する．

評価に用いた GA のパラメータを表 2 に示す．

4.3.2　評 価 結 果

単一個体集団の GA を逐次で処理したときの解と同品質のものが得られるように移住パラメータを設定し，速度向上率の比較を行う．

表 2　GA パラメータ

GA parameter	KP	TSP
baggage ｜ town num	1000	100
population size	4000	4000
crossover rate	0.9	0.9
mutation rate	0.001	0.01
generation num	1000	1000

結果を図 6 のグラフに示す．8 並列時に 7 倍程度の速度向上率を得られることが確認できたが，計算機台数の増加に伴って並列化効率は低下している．

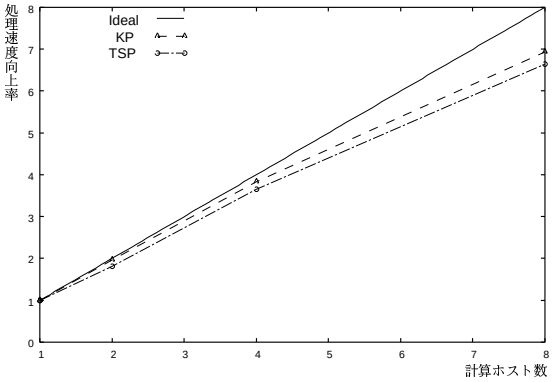

図 6　速度向上率グラフ

この結果より，比較的小規模の並列動作においては並列化効率は 80 %を越えるものとなっており，実装したライブラリの有用性を証明することができた．しかしながら，並列規模の増大に伴って並列化効率が低下していっているため，計算機台数の多い大規模な環境下においては通信時のオーバヘッドなどにより，並列化効率の低下は著しいものになると考えられる．これは，現在採用している通信方法のようにマルチキャストを利用した通信では大規模並列においては非効率的である．MegaScript が実現するような環境下において優れた並列動作を行うには，今後，通信方法について改良していく必要がある．

4.4　木探索ライブラリの評価

4.4.1　評 価 内 容

評価対象としてオセロを取り上げ，初期局面からの一定手数の先読みを行いミニマックス法・$\alpha\beta$ カットアルゴリズムを用いて最良着手を決定する．逐次切り替え深さを 2 と設定し (ルートノードは深さ 0 にしている)，現深さが 2 に達すると逐次処理を行うようにした．

4.4.2 評価結果

速度向上率と枝刈り効率をグラフ化したものをそれぞれ図 7，図 8 に示す．

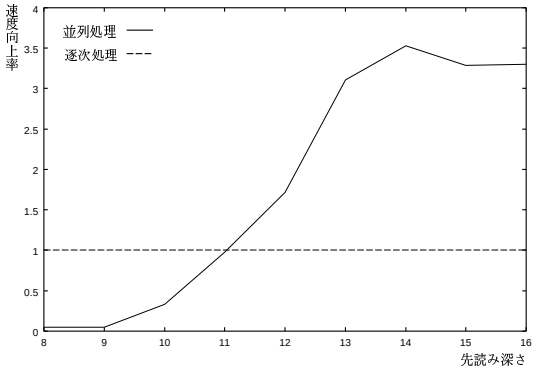

図 7 速度向上率

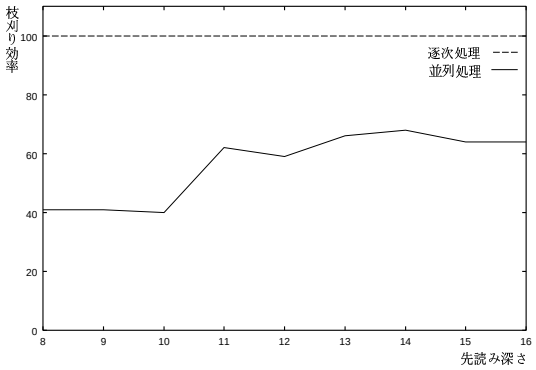

図 8 枝刈り効率

9 台で処理を行った結果，最大で逐次処理の約 3.5 倍の速度が出た．図 7 において，先読み深さが 14 以降で処理効率が低下している原因として，タスク数は深さ 2 の時点の枝数になるので一定だが，タスクの粒度は深さ 2 から打ち切り深さに相当するので計算量に差が生じ，処理効率が低下したと考えられる．また，並列度に対し速度向上があまり高くないが，これは図 8 で見られるように枝刈り効率が低いためである．したがって枝刈り効率を改善することで，さらに速度を向上させられる可能性がある．枝刈り効率が低い原因として，$\alpha\beta$ カットアルゴリズムは逐次性を持つアルゴリズムなので，並列探索を行った時に，本来刈られるはずの枝まで探索してしまったと考えられる．

5. おわりに

本稿では，タスク並列スクリプト言語用アプリケーション層ライブラリを設計･実装し，その評価を行った，

パラメータサーベイ･ライブラリの性能評価では，実際にプロセッサ研究分野で使われている SimpleScalar を用いて評価を行った．その結果，実用レベルのシミュレーションが行えることを確認した．また，動的負荷分散機能がうまく機能していることも確認できた．

GA ライブラリの性能評価では，ライブラリに対し問題に依存した逐次の処理をリンクするだけで並列化を実現でき，8 並列で 6〜7 倍程度の速度向上率を得ることができた．しかしながら，計算機台数の増加に伴い並列化効率が低下していっており，その原因である通信時のオーバヘッドの低減をはかる必要がある．GA のように複数のタスクとの 1 対 1 通信が必要な場合，マルチキャストによる通信方法では通信効率の点から不利である．そのため，通信効率のよい大規模並列を実現するためには言語側において通信方法の改善を検討してもらう必要がある．

木探索ライブラリの性能評価では，サンプルプログラムとして用意したのがオセロだけであることやゲーム木探索以外の木探索に対応していないことなどまだ至らない点はある．よって今後サンプルプログラムをより多く用意し，結果を測定していきライブラリとしての完成度を上げていかなければならない．

謝辞 本研究は，科学技術振興事業団･戦略的基礎研究「低電力化とモデリング技術によるメガスケールコンピューティング」による．

参考文献

1) 大塚 保紀，深野 佑公，西里 一史，大野 和彦，中島 浩：タスク並列スクリプト MegaScript の構想，先進的計算基盤システムシンポジウム SACSIS2003，pp.73-76(2003)．
2) 西里 一史，大野 和彦，中島 浩：タスク並列スクリプト言語 MegaScript のランタイムシステムの設計と実装，情報処理学会研究報告，2003-HPC-95，pp.119-124(2003)．
3) 三宮 信夫，喜多 一，玉置 久，岩本 貴司：遺伝アルゴリズムと最適化，朝倉書店 (1998)．
4) SimpleScalar 公式サイト http://www.simplescalar.com/
5) 大塚 保紀，大野 和彦，中島 浩：タスク並列スクリプト言語 MegaScript によるタスク動作モデル記述，情報処理学会研究報告，2003-HPC-95，pp.113-118(2003)．
6) 原 真一郎，まつもとゆきひろ：Ruby プログラミング入門，Ohmsha (2000)．