MegaScript における多次元タスク配列の設計と実装

増田 嵩[†] 三田明宏[†,††] 松本真樹[†]
大野和彦[†] 佐々木敬泰[†] 近藤利夫[†]

1. はじめに

近年，遺伝子解析や環境シミュレーション等複雑な物理計算を要する分野で，非常に高い計算性能が要求される．これには大規模な並列処理が必要とされる．

そこで，我々は大規模並列処理を容易に実現するためのタスク並列スクリプト言語 MegaScript[1] を開発している．MegaScript にはいくつかの API クラスが用意されているが，マルチパラメータスイープ等を記述するには冗長なコードが必要となる．従って，これらを容易に記述する為の API の設計と実装を行った．

2. 背景

2.1 MegaScript

MegaScript はワークフローモデルに基づく並列言語であり，外部プログラムであるタスクを繋ぎ合わせることで並列処理を行う．タスク間のデータ通信はストリームと呼ばれる通信路を用いる事で実現する．

MegaScript は Ruby ベースの言語であり，API クラスとしてタスクやストリームを表す Task や Stream を用意している．また，複数のタスクを扱う際に初期化を簡単にする為のクラスとして TaskArray がある．例として，プロセッサシミュレータ SimpleScalar でプロセッサの動作シミュレーションを行う処理を記述したコードを**図 1** に示す．sim-outorder は SimpleScalar の実行ファイル名であり，オプションで分岐予測ミスのレイテンシの設定を行っている．例では，1 から 5 までの各値を設定値とした 5 個のタスクを生成している．また，引数の最後にある exefile はシミュレーションに使用する実行ファイル名である．このように，TaskArray を使うと類似したタスクの集合を簡単に記述する事が出来る．

```
t1 = TaskArray.new(5, "sim-outorder", "-fetch:mplat", 1..5, "exefile")
```

図 1 TaskArray の例

2.2 Ruby

Ruby はオブジェクト指向スクリプト言語であり可読性を重視した構文が特徴である．また，プログラムコード中では**図 2** のようなメソッド呼び出しの後にブロックを付加したブロック付きメソッド呼び出しと呼ばれる継続の一種がよく使用される．この例では，ファイルをオープンしてブロックを実行するのではなく，ファイルをオープンするメソッドにブロックを渡してメソッド中でブロックを実行させる．従って，ファイルをオープンして実行する処理をブロックで渡す事が出来る．

```
File.open("data.txt") do |fp|
  while line = fp.gets
    print line
  end
end
```

図 2 ブロック付きメソッド呼び出しの例

Ruby では組み込みライブラリとしてブロックをコンテキストと共にオブジェクト化する Proc クラスや，可変長配列である Array クラス，連想配列である Hash クラス等を利用できる．

3. 提案手法

従来の TaskArray は一次元配列のみ対応しており，マルチパラメータスイープ等の記述が煩雑である．そこで本手法では，多次元配列に対応した TaskArray（以下、多次元対応 TaskArray）を提案する．

3.1 設計方針

MegaScript では 100 万タスク規模の並列処理を想定している．その為，従来の TaskArray ではタスクをそのまま保持するのではなく，生成に必要な情報を保持し必要に応じてタスクを生成する縮約と呼ばれる方法[2] を取っている．多次元対応 TaskArray では従来より扱うタスク数が大きくなることが想定されるので，従来の TaskArray に比べ縮約の必要性が高い．従って，縮約を考慮した設計，実装を行った．

従来の TaskArray でマルチパラメータスイープを記述するには無理に一次元配列で書く必要があり煩

† 三重大学大学院工学研究科情報工学専攻
†† 現在，三菱電機メカトロニクスソフトウェア株式会社

```
./sim-outorder -cache:dl1:32:32:1 -cache:dl2:64:32:1 file
./sim-outorder -cache:dl1:32:32:2 -cache:dl2:64:32:2 file
./sim-outorder -cache:dl1:32:32:4 -cache:dl2:64:32:4 file
...
./sim-outorder -cache:dl1:32:32:32 -cache:dl2:64:32:32 file
./sim-outorder -cache:dl1:32:64:1 -cache:dl2:64:64:1 file
...
./sim-outorder -cache:dl1:1024:128:16 -cache:dl2:2048:128:16 file
./sim-outorder -cache:dl1:1024:128:32 -cache:dl2:2048:128:32 file
```

図 3 SimpleScalar の実行例

```
t1 = TaskArray.new(6 * 3 * 6, "sim-outorder",
  proc {|i| "-cache:dl1:#{2 ** (5 + i / (3 * 6))}:" +
            "#{2 ** (5 + i / 6 % 3)}:" +
            "#{2 ** (i % 6)}"},
  proc {|i| "-cache:dl2:#{2 ** (6 + i / (3 * 6))}:" +
            "#{2 ** (5 + i / 6 % 3)}:" +
            "#{2 ** (i % 6)}"},
  "file")
```

図 4 従来の TaskArray での記法

```
t1 = TaskArray.new([6, 3, 6], "sim-outorder") do |i, j, k|
  ["-cache:dl1:#{2 ** (5 + i)}:#{2 ** (5 + j)}:#{2 ** k}",
   "-cache:dl2:#{2 ** (6 + i)}:#{2 ** (5 + j)}:#{2 ** k}",
   "file"]
end
```

図 5 多次元対応 TaskArray での記法

雑なコードになる．例として，**図 3** のように L1・L2 キャッシュの大きさを変えながら SimpleScalar を実行する場合を考える．キャッシュの大きさはオプションで三つの値を与える事で指定でき，これらの値を変化させる事で三次元のパラメータスイープを行っている．パラメータは，L1 キャッシュの場合それぞれ 32〜1024，32〜128，1〜32 の範囲の 2 のべき乗を取る．これを従来の TaskArray を用いて記述すると**図 4** のようになる．従来手法では TaskArray のコンストラクタに，配列の要素数とプログラム名，タスクに与える引数（ここではオプション）を指定する．この時，オプションで指定する数値は Proc クラスでそれぞれ計算している．従来の TaskArray では一次元配列にしか対応していない為，多次元配列における添え字を得る為にはその都度一次元配列の添え字を元に計算する必要がある．この為，コードの可読性が悪化したり，バグの原因になる等の問題がある．本手法ではこれらの問題を改善する為に，より簡単にマルチパラメータスイープ等を記述できる記法の設計も行った．

3.2 記述方法の検証

本手法では多次元対応 TaskArray クラスの設計に加え，より簡単にマルチパラメータスイープを記述する為の記法を設計した．これを図 4 と等価のタスクを生成するコードで示す．

本手法を用いると**図 5** のように記述できる．本手法では生成時に各次元での配列の要素数を格納した配列リテラルとプログラム名を指定する．また，オプションはブロックで指定する．このブロックはタスクを生成する度に実行され，そのタスクに対する多次元配列の添え字が引数として与えられる．この実行結果から得られる配列をそのタスクの引数として格納する．

本手法を使用する事で，ユーザが配列の要素数から多次元配列での添え字を計算する必要がなくなる．従って，タスクに与える引数をより簡単に指定でき，可読性の向上と共にバグの発生の抑制になる．

3.3 実　　装

従来の TaskArray は一次元配列であり，縮約できないタスクだけを Hash クラスを用いて保持する事でメモリ使用量の削減を行っている．また，Ruby の多次元配列は C 言語と同様の多次元配列ではなく，一次元配列に一次元配列の参照を格納していく入れ子構造となっている．その為，従来の TaskArray と同じ手法でタスクの初期化を行う事が困難である．また，縮約した TaskArray の部分配列において実行ファイル名やタスクの引数の変更を行う場合，縮約されていないタスクが多く生成されてしまう．この結果，縮約の効率上も問題がある．

多次元対応 TaskArray ではこれらの問題を解決する為に Proc クラスのカリー化を行った．これにより，複数の引数を取る Proc オブジェクトを，一つの引数を取り Proc クラスを返す Proc オブジェクトに変換している．多次元対応 TaskArray ではタスクの引数を指定するのにブロックを使っており，多次元対応 TaskArray の内部ではこのブロックを Proc クラスで保持している．この Proc クラスをカリー化する事でタスクの初期化が容易になる．また，カリー化により Proc クラスに渡す引数の変更や Proc クラスの差し替えが可能になった．その結果，縮約されていないタスクの縮約が可能となり，縮約効率の向上が出来た．

4. おわりに

本稿では，タスク並列スクリプト言語 MegaScript において多次元配列に対応した TaskArray の設計と実装について述べた．その結果，マルチパラメータスイープの記述が容易に行え可読性を向上できた．

今後の課題として縮約で用いる閾値を決定する必要がある．また，メモリ使用量やオーバーヘッドに関して従来の TaskArray との比較評価を行っていく．

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言語 MegaScript の構想, *SACSIS2003*, pp. 73–76 (2003).
2) 三田明宏, 仲貴幸, 松本真樹, 大野和彦, 佐々木敬泰, 近藤利夫: MegaScript における大規模ワークフロー縮約機構の設計, 情処研報 2011-HPC-130, pp. 1–9 (2011).