特集
マルチコアを活かすお手軽並列プログラミング

1 マルチコア計算機と基本的な並列化技法

松崎 公紀（東京大学）
武市 正人（東京大学）

マルチプロセッサからマルチコアへ

マルチコアプロセッサとは，1つのプロセッサの中に複数のコアと呼ばれる計算ユニットを持つようなプロセッサである．それまでは手間をかけて並列計算機環境を用意する必要があったのだが，マルチコアプロセッサの発展によって簡単に並列計算機として動作する計算機が手に入るようになった．

マルチコアプロセッサが利用される以前から，並列計算を行うための計算機環境を実現する手法として次の2つの手法が利用されてきた．1つは，1つの計算機に複数のプロセッサを搭載する手法(マルチプロセッサ)であり，もう1つは，複数のノード(計算機)をネットワークによって接続し全体を1つの計算機として利用する手法である．マルチコア計算機は，1つの計算機の中に計算ユニットであるコアが複数あるという点でマルチプロセッサに近い．

マルチプロセッサなどの1つのノードからなる並列計算機のモデルとして，Symmetric Multi-Processing（SMP）がよく利用される．SMPは，1つのメモリと複数の計算ユニットを複数持つような計算機モデルである．SMPでは，複数の計算ユニットは機能などの点で区別されず対称的に扱われ，各計算ユニットは共有されているメモリをアクセスして並列に計算を行う．代表的なWindows，Mac OS X，Linuxなどの OSは，このSMPモデルに対応している．

実際は，ひとことでマルチコアプロセッサといっても，吉瀬氏の解説記事（本誌 pp.1403-1406）にて解説されているようにコアの種類やキャッシュの構造などの点でさまざまな形式のものがある．本特集記事では，現在入手しやすい以下のような計算機を対象とする．まず，マルチコアプロセッサとして，Intel社の Coreアーキテクチャや AMD社の Phenomアーキテクチャのような，同一のコアからなるものを対象とする．また，プロセッサを1つないし複数持つような計算機1台を対象とし，PCクラスタのようなネットワークによってつながれた複数の計算機からなる環境は考えない．

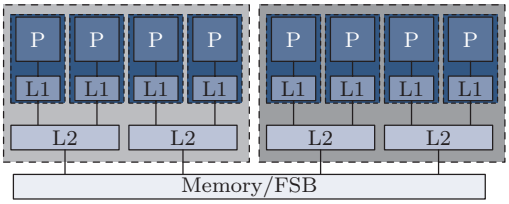

図-1　デュアル Core2Quad 計算機における計算ユニットとキャッシュ／メモリの階層的な構造

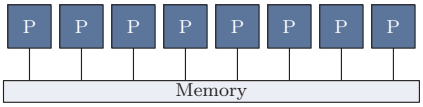

図-2　図-1のマルチコア計算機に対する簡易的なモデル

本特集の対象となるマルチコア計算機とそれに対する簡易的な SMPモデルについて1つ具体例を挙げておこう．例として，4つのコアを持つ Core2Quadプロセッサを2つ持つような計算機を考える．この計算機におけるコアやキャッシュ／メモリは図-1に示すような階層的な構造を持っている．これに対して，図-2に示すようなフラットな8つの計算ユニットとメモリからなる計算機として考えるというのが SMPによるモデルである．

上に例示した代表的な OSでは，マルチコアプロセッサを持つ計算機はこのような SMPとして捉えられている☆1．したがって，ユーザがマルチコア計算機上で並列プログラムを作成する場合には，図-2に示される比

☆1 Windows や Linux では，スレッドがどのコアの上で実行可能かを指定する方法もある．これを使うと計算機に合わせたプログラムを作成できるが，本特集の「お手軽さ」の範囲外とする．

較的簡単な SMP モデルの上で作成すれば大抵の場合は十分であろう．

2 つの重要なお手軽並列化手法

並列プログラミングを行うための重要な考え方に，データ並列（data parallelism）とタスク並列（task parallelism）の 2 つがある．データ並列は，多数のデータに対して独立な同様の処理を同時に適用して並列に計算する手法である．数値計算分野の多くのアプリケーションや Google 社の MapReduce などは，このデータ並列の手法が適用されている．一方，タスク並列は，独立性の高い機能をコンポーネントとして実装し，それぞれ独立に並列に計算を行う手法である．オブジェクト指向プログラミングにおける独立した複数のオブジェクトなどには，このタスク並列の手法が適用できる．

これらの並列化の考え方を比較的容易に実現するための手法として，それぞれ Parallel-For ベースの方法と Fork-Join ベースの方法がある．

❒ Parallel-For ベースの方法

データ並列の考え方によってプログラムを並列化する一番簡単な方法は，プログラム中の for 文のうち処理が独立しているものを並列に計算を行う適切なライブラリで置き換えるものである．この方法による並列化をここでは Parallel-For ベースの方法と呼ぶ．

たとえば，配列 a の値の 2 乗和を求める次のプログラム

```
SIMPLE-LOOP(a, n)
1 for i = 1 to n
2   do s ← s + a[i] × a[i]
3 return s
```

において，値の 2 乗を計算する部分はそれぞれ独立しており，また合計を求める計算は + の結合性によって並列化できる．したがって，独立した計算を行う部分を map（parallel-for や do-all とも呼ばれる）に，合計を求める計算を reduce（sum，accumulate とも呼ばれる）にそれぞれ置き換えることで，並列に計算を行うプログラムを得ることができる（図 -3）．

```
MAP-AND-REDUCE(a, n)
1 map(i ∈ [1..n] について並列に,
        b[i] ← a[i] × a[i] を実行)
2 s ← reudce(b を +で合計)
3 return s
```

上記の map や reduce は，対象となるデータを適度なサイズに分割して並列に計算される．得られるプログラムの効率はその分割サイズによって多少変化するが，もとの for 文の繰り返し回数が大きい方が並列化による効果が大きい．大きい繰り返しがない場合には，ネストした繰り返しを展開することで並列化による効果を稼ぐことができる場合がある．

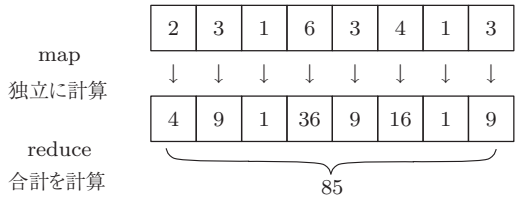

図 -3　Parallel-For ベースでの配列の 2 乗和の計算

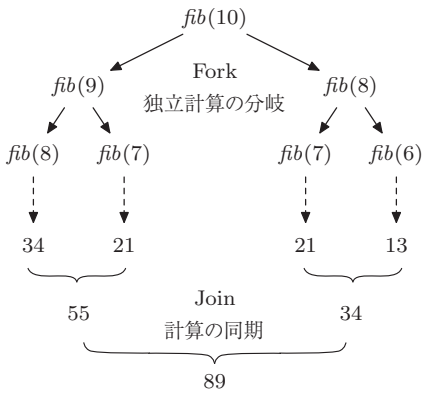

図 -4　Fork-Join ベースでの Fibonacci 数の計算

❒ Fork-Join ベースの方法

タスク並列の考え方によってプログラムを並列化する一番簡単な方法は，プログラム中の関数呼び出しで処理が独立しているものを同時に実行できるようにすることである．この方法による並列化をここでは Fork-Join ベースの方法と呼ぶ．

たとえば，Fibonacci 数を求める次の素朴なプログラム

```
FIB-SEQ(n)
1 if n ≤ 1
2   then return 1
3   else return FIB-SEQ(n − 1) + FIB-SEQ(n − 2)
```

の else 部の 2 つの関数呼び出しはそれぞれ独立している．すなわち，この 2 つの関数呼び出しを並列に行っても計算結果は変わらない．したがって，これらの 2 つの関数呼び出しを同時に行うことができるように指示することで，プログラムを並列化することができる（図 -4）．

```
FIB-FJ(n)
1 if n ≤ 1
2   then return 1
3   else x ← fork(FIB-FJ(n − 1))
4        y ← fork(FIB-FJ(n − 2))
5        join          // 同期
6        return x + y
```

この Fork-Join による並列化手法は，もとの逐次プログラムが再帰関数によって構築されている場合に適用可

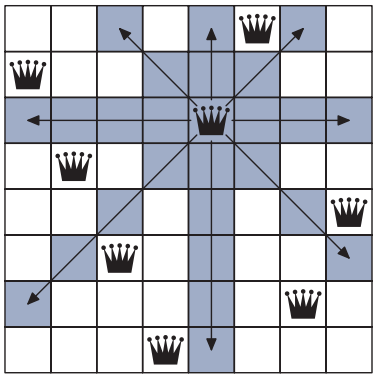

図 -5　8-Queens パズルの 1 つの解とクイーンの効き

能な場合が多い．ただし，同時に計算するための仕組みとしてスレッドが使われることが多いので，生成されるスレッドの数が多くなりすぎないように処理の分岐する程度を制御することが効率の面で重要となる．たとえば上の Fib-FJ 関数において，最後まで処理を分岐させると多数のスレッドが生成されることになり効率が悪い．適切な回数だけ分岐した後は逐次関数のみ呼び出すように切り替えることで効率が改善する．

N-Queens 問題

本特集では，お手軽並列プログラミングを支援するさまざまなライブラリを紹介する．プログラムの理解や比較を容易にするため，共通問題として N-Queens 問題を使う．

❒ 問題定義

まず，よく知られた 8-Queens パズルを紹介する．8-Queens パズルは，チェスの 8×8 の盤面の上に 8 つのクイーンを，互いに効き（移動範囲）の上にないように配置するパズルである．クイーンは，縦・横・ななめの 8 方向のいずれかに何マスでも進むことができる．**図 -5** に 8-Queens パズルの 1 つの解を示す．

この 8-Queens パズルのクイーンの数および盤面をそれぞれ N 個および $N \times N$ に一般化する（N-Queens パズル）．このとき，N が 4 以上であれば，そのようなクイーンの配置が存在することが知られている．本特集での共通問題である N-Queens 問題は，N-Queens パズルの解の数を（回転や対称移動によって一致するものも区別して）数えるという問題である．N-Queens パズルの解の個数は盤面のサイズが増えると非常に大きくなる（**表 -1**）．2008 年 10 月の時点で N=25 までの解の個数が計算されている [1]．

この問題は，問題の定義が簡単でよく知られていることと並列化が比較的容易であることから，並列計算のベンチマークプログラムとしても利用されている．たとえば，N=24 の解は 2004 年に吉瀬らによって 34 ノードの PC クラスタを利用して求められた [2]．過去最大の N=25 の解は 2005 年にグリッド環境を利用して求められたものである [3]．

盤面のサイズ	N-Queens 問題の解
4	2
5	10
6	4
7	40
8	92
⋮	⋮
16	14,772,512
⋮	⋮
20	39,029,188,884
⋮	⋮
24	227,514,171,973,736
25	2,207,893,435,808,352

表 -1　N-Queens 問題の解（一部）

❒ 逐次解法

まず，N-Queens 問題を解く逐次アルゴリズムを示す．N-Queens 問題を解く 1 つのアルゴリズムは，後もどり法（Backtrack）によるものである．これは，第 1 列目，第 2 列目と順番に制約を満たすように盤面にクイーンを置いていき，もし最終列まで置くことができた場合には解の個数を 1 増やす，もしそれ以上置くことができない場合には 1 つ前の列に戻って別の置き方を調べる，という方法である．

このような逐次アルゴリズムは，再帰関数を用いることで簡単に記述することができる．以下の疑似コードの関数 NQ-Seq が受け取る引数は，盤面のサイズ（n），次にクイーンを置く列（x），クイーンを置いた位置を保持する配列（ys），である．

```
NQ-Seq(n, x, ys)
 1 if n = x
 2   then return 1
 3   else for y = 1 to n
 4          do cnt ← 0
 5             isOK ← True
 6             for i = 1 to x − 1
 7               do if (x, y) と (i, ys[i]) の間に効きがある
 8                    then isOK ← False
 9             if isOK = True
10               then ys[x] ← y
11                    cnt ← cnt + NQ-Seq(n, x + 1, ys)
12        return cnt
```

次の NQ-Seq2 は，クイーンを置いた位置の配列を渡すのではなく，横（r）とななめ上（u）とななめ下（l）の効

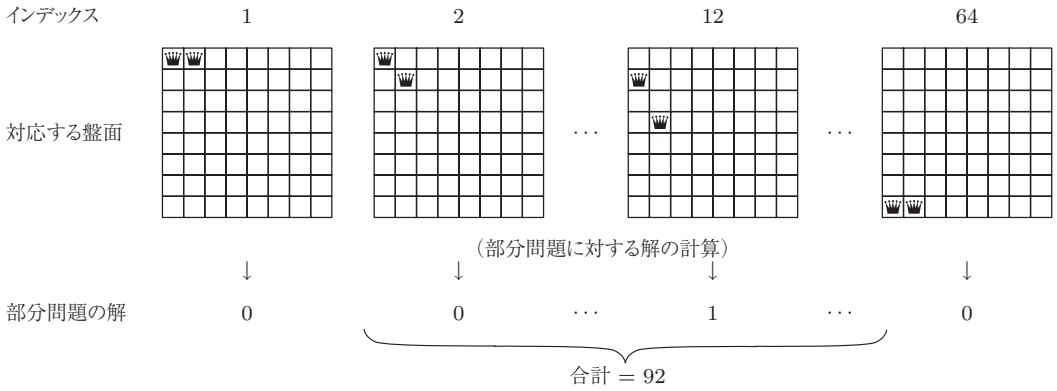

図 -6　Parallel-For 手法による 8-Queens 問題の並列解法

きを渡すという方法による再帰関数である．

```
NQ-SEQ2(n, x, r, u, l)
 1 if n = x
 2   then return 1
 3   else cnt ← 0
 4        u' ← u を 1 つ上にずらす
 5        l' ← l を 1 つ下にずらす
 6        for y ∈ (r, u', l' のいずれにも効きがない)
 7          do r'' ← r の位置 y に効きを追加
 8             u'' ← u' の位置 y に効きを追加
 9             l'' ← l' の位置 y に効きを追加
10             cnt ← cnt + NQ-SEQ2(n, x + 1, r'', u'', l'')
11        return cnt
```

N=24 の場合の解を求めた吉瀬らによるプログラム [2] では，2 つ目のアルゴリズム NQ-SEQ2 の再帰関数をループで実現することで高速化したものが使われている．実際に効率の良い並列プログラムを作成するには，このように逐次計算部分の高速化を行うことが非常に重要であるが，本稿では並列化手法について着目するため議論の範囲外とする．

以下では，再帰関数 NQ-SEC をベースとして，前述した 2 つの並列化手法を利用することでどのように並列プログラムを得ることができるかを示す．

❒ Parallel-For ベースの並列解法

上記の逐次プログラムを開始点として，Parallel-For ベースの並列化手法によってどのように並列化を行うかを示す．この解法によるプログラムの動作の概略は図 -6 に示される．

手順 1．Parallel-For で走査するデータの準備

まず，Parallel-For によって走査する大きなデータを準備する．N-Queens 問題において NQ-SEC 関数の中の for 文はその繰り返し回数が n（盤面のサイズ）であり，並列化の効果を得るにはやや不十分である．そこで，ここでは端から 2 列分だけ展開を行い N^2 のサイズの for 文を作ることにする☆2．簡単のため，端の 2 列の時点で制約を満たさないものについてもとりあえずデータに含めることにする．

手順 2．各データに対する処理のため処理を書き換え

手順 1 で準備したデータに合わせて必要があれば処理を書き換える．今回の並列化では，端 2 列を展開することに伴う変更が必要となる．

```
NQ-PART(n, i)
1 ys[1] ← ((i − 1)/n) + 1
2 ys[2] ← ((i − 1)%n) + 1
3 if (1, ys[1]) と (2, ys[2]) に効きがある
4   then return 0
5   else return NQ-SEQ(n, 3, ys)
```

この関数は，インデックス i から第 1 列目と第 2 列目に置くべき位置を計算した後，2 つのクイーンが効きの位置にないかを判定し，続きの逐次プログラムを呼び出している．

手順 3．ライブラリの並列化を適用

ライブラリのサポートなどを利用して，手順 1 のデータを走査する for 文を並列化する．また，総和を求める部分についても並列化できるならばさらに行う．

```
NQ-PF(n)
1 a ← n × n のサイズの array
2 map(i ∈ [1..n × n] について並列に,
        a[i] ← NQ-PART(n, i) を実行)
3 s ← reudce(a を + で合計)
4 return s
```

❒ Fork-Join ベースの並列解法

次に，逐次プログラムを開始点として，Fork-Join ベースの並列化手法によってどのように並列化を行うかを示す．この解法によるプログラムの動作の概略は図 -7 に示される．

☆2　2 コアの計算機であればこのような展開を行わなくても十分に並列化の効果が得られるかもしれない．逆に，64CPU を使って N=24 の場合を計算した吉瀬らの実装では，効率良く計算するため，端から 4 列を展開して，この時点で制約を満たさないものを除外することで 75516 個の部分問題に分割していた．

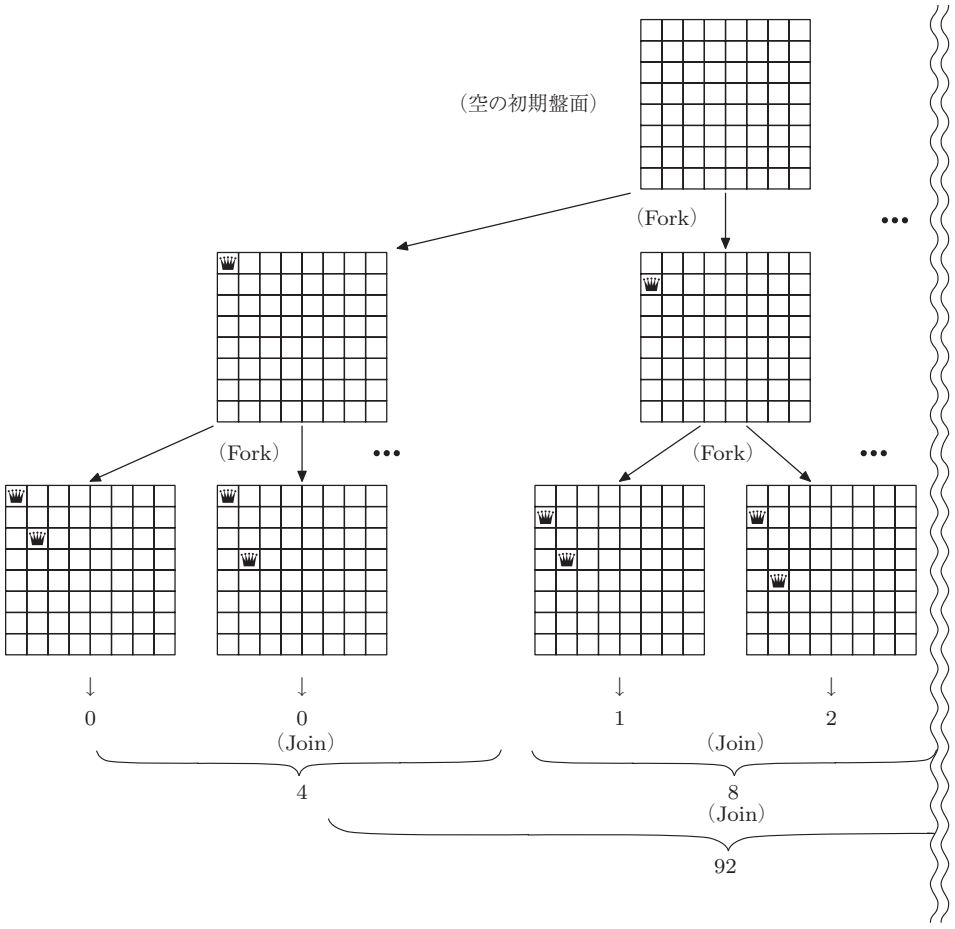

図 -7　Fork-Join 手法による 8-Queens 問題の並列解法

手順 1．独立な関数呼び出しを見つける

まず，プログラム中の関数呼び出しについて複数の関数呼び出し間の依存関係を調べ，独立した関数呼び出しを見つける．上記の NQ-Sec 関数の場合は，for 文内の再帰呼び出しは独立しているので，これをそのまま並列化の対象とする．

手順 2．ライブラリの並列化を適用

次に，手順 1 で見つけた関数呼び出しの部分について，同時に計算ができるようにライブラリのサポートを使ってプログラムを書き換える．多くの場合は，単純に関数呼び出しを置き換えた上で，すべての処理が終わるのを待つ（同期する）ところにコードを追加することになる．

```
NQ-FJ(n, x, ys)
 1 if n = x
 2   then return 1
 3   else for y = 1 to n
 4           do 置けるかどうかのチェック (省略)
 5              if isOK = True
 6                then ys[i] ← y
 7                     cnt[y] ← fork(NQ-FJ(n, x + 1, ys))
 8        join
 9        return cnt の合計
```

手順 3．並列部分／逐次部分の調整

手順 2 で得られた並列プログラムで望ましい並列効果が得られない場合でも，性能のチューニングができる場合がある．もとのプログラムが再帰関数である場合において，最後の関数呼び出しまで fork/join を行うと生成されるスレッドの数が多くなりすぎ性能が出なくなりやすい．そのような場合は，適切な数の分岐後に逐次の計算に切り替えることで効率が改善できる．たとえば，3 列目以降について逐次の計算に切り替える改良を行った（1 行目）プログラムは次のようになる．

```
NQ-FJ2(n, x, ys)
 1 if x = 3 ∨ x = n
 2   then return NQ-Seq(n, x, ys)
 3   else for y = 1 to n
 4           do 置けるかどうかのチェック (省略)
 5              if isOK = True
 6                then ys[i] ← y
 7                     cnt[y] ← fork(NQ-FJ2(n, x + 1, ys))
 8        join
 9        return cnt の合計
```

お手軽並列プログラミングをサポートするライブラリたち

最後に，容易に並列プログラムを作成することをサポートするいくつかのライブラリについて概観する．

容易に並列プログラムを作成するためのプログラミング言語やライブラリの研究は，1990 年ごろから行

われてきている．その中でも重要かつ有名なものは，NESL[4] および Cilk[5] である．これらの並列プログラミング言語の研究は，その後の並列言語や並列ライブラリの研究に大きな影響を与えている．

NESL[4] は，CMU の Blelloch のグループによって開発されたデータ並列プログラミング言語である．NESL の特徴はリストの内包表記を効率的に並列計算する点である．リストの内包表記は，たとえばリスト xs の 2 乗和を求めるのに $\sum[x^2|x \in xs]$ と書く表記法である．リストの内包表記を用いることで，ユーザは簡潔にプログラムを記述することができ，また得られたプログラムは並列に実行することができる．NESL では，標準的な map や reduce に加えて scan（prefix-sums）と呼ばれる並列アルゴリズムがあり，多くのアルゴリズムが NESL の枠組みで記述できる．

Cilk[5] は，MIT の Leiserson のグループによって開発された Fork-Join ベースの並列プログラミング言語であり，C 言語に Fork-Join（とその拡張）を実現するための少数のプリミティブを追加したものである．Cilk における複数のスレッドの実行スケジューリング手法はワークスティーリングと呼ばれ，さまざまな並列計算ライブラリのスケジューリングに用いられている．

表 -2 に現在もしくは近い将来に比較的容易に入手することができる並列プログラミングライブラリの特徴についてまとめた．これまでに開発された並列プログラム作成のためのライブラリの多くは C＋＋ が対象言語となっている．解説 2 の OpenMP，libstdc＋＋ parallel mode，解説 4 の Intel Thread Building Blocks，SkeTo，その他 .NET などは C＋＋ 言語（OpenMP，.NET は C なども）を対象とする Parallel-For ベースの並列プログラミングライブラリである．Java 言語については，解説 3 の Java 1.7 で拡張される予定の java.util.concurrent パッケージが Fork-Join ベースによる並列プログラミングをサポートする．また，近年スクリプト言語や関数型プログラミング言語の普及に伴い，これらに対する簡単な並列プログラミングライブラリの作成も積極的に行われている．解説 5 の dRuby のほか，Parallel Python や Parallel Haskell などを利用することで Fork-Join ベースの並列プログラミングを容易に行うことができる．

これらの並列プログラミングライブラリを利用することで，マルチコア計算機をより効果的に利用できるようになると期待する．

	言語	Par-For	Fork-Join	複数 PC
OpenMP	C++など	○	△	×
並列 libstdc++	C++	○	×	×
Java 1.7	Java	△	○	×
Intel TBB*1	C++	○	△	×
SkeTo	C++	○	×	○
dRuby	Ruby	△ *2	○	○
.NET TPL*3	C++など	○	△	×
Parallel Python	Python	×	○	○
Parallel Haskell	Haskell	△	○	×

○はそのライブラリで主に対象とされている並列化手法，△はサポートされているが中心的な機能でないもの，×はサポートされていない機能をそれぞれ示す．

*1: Intel Thread Building Blocks
*2: Rinda を併用することで可能
*3: Task Parallel Library（.NET 3.5 以降）

表 -2　現在もしくは近い将来に入手できる並列プログラミングライブラリの特徴

参考文献

1）Sloane, N. J. A. : Number of Ways of Placing n Nonattacking Queens on $n \times n$ Board, In the On-Line Encyclopedia of Integer Sequences (OEIS), available from http://www.research.break att.com/~njas/sequences/A000170 (2008).
2）吉瀬謙二，片桐孝洋，本多弘樹，弓場敏嗣：PC クラスタを用いた N-queens 問題の求解，電子情報通信学会論文誌 D-I, 情報・システム，I- 情報処理，Vol.87, No.12, pp.1145-1148 (2004).
3）Caromel, D., Costanzo, A. and Mathieu, C. : Peer-to-peer for Computational Grids : Mixing Clusters and Desktop Machines, Parallel Computing, Vol.33, No.4-5, pp.275-288 (2007).
4）Blelloch, G. E., Chatterjee, S., Hardwick, J. C., Sipelstein, J. and Zagha, M. : Implementation of a Portable Nested Data-Parallel Language, Journal of Parallel and Distributed Computing, Vol.21, No.1, pp.4-14 (1994).
5）Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H. and Zhou, Y. : Cilk : An Efficient Multithreaded Runtime System, Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, pp.207-216 (1995).

（平成 20 年 10 月 29 日受付）

松崎　公紀（正会員）　kmatsu@ipl.t.u-tokyo.ac.jp

1979 年生．2001 年東京大学工学部計数工学科卒業．2003 年同大学院情報理工学系研究科修士課程修了．2005 年同研究科博士課程中退．同年より同研究科助手，2007 年より助教となり現在に至る．博士（情報理工学）．並列プログラミング，アルゴリズム導出などに興味を持つ．日本ソフトウェア科学会会員．

武市　正人（正会員）　takeichi@mist.i.u-tokyo.ac.jp

1972 年東京大学工学部助手，講師，電気通信大学講師，助教授，東京大学工学部助教授，を経て 1993 年東京大学大学院工学系研究科教授，2001 年より同大学情報理工学系研究科教授，現在に至る．2003 年より日本学術会議会員，現在に至る．工学博士．プログラミング言語，関数プログラミング，言語処理システムの研究・教育に従事．日本ソフトウェア科学会，ACM 各会員．