2009－GI－21 (13)
2009／3／9

並列局面探索における待ち行列の負荷分散

柳 圭二郎, 柴原 一友, 但馬 康宏, 小谷 善行
東京農工大学 工学府情報工学専攻

概要 並列局面探索において改善を目指すときに，タスクの待ち行列の負荷分散が考えられる．そこで，優先度を決定するアルゴリズムを他分野の並列環境でも用いられている「深さ別待ち行列」「深さ別優先度付き待ち行列」「重み付き待ち行列」の三種類を並列局面探索に適用し，基準となるプログラムにそれぞれ実装した形での対局によって比較した． Cell において実装し，比較用の Intel 系 CPU における逐次プログラムに対して，どれだけ勝率が改善されたかを示す．初期状態において 200 対局 46.5%に対し，「重み付き待ち行列」は 46%と変わらなかったが，改善した「深さ別待ち行列」は 53.5%，「優先度付き待ち行列は」50.1%となった．

Load-balancing of Queue in Parallel Game-Tree Search

Keijirou YANAGI, Kazutomo SHIBAHARA, Yasuhiro TAJIMA, Yoshiyuki KOTANI
Tokyo University of Agriculture and Technology

Abstract When aiming at the improvement in the parallel game-tree search, the load-balancing of the queue of the task is thought. Then, three kinds "Queue according to depth", "Queue with priority according to depth", and "Queue with weight" were set, and algorithms that decided priority were compared by the shape mounted on the program that became a standard respectively play. It mounts in Cell, and how much the chance of success was improved to system CPU Intel for the comparison is shown. Improved "Queue according to depth" became 53.5% and "Queue with priority" 50.1% though "Queue with weight" was not different from 46% in the initial state compared with 200 play 46.5%.

1．はじめに

将棋における並列局面探索の多くは Intel 系などのマルチコア CPU 環境が主流である．それに対し，異なるコアの PPU 1 個と SPU 6 個が混在する CPU である”Cell Broadband Engine（以下 Cell）”は，比較的安価なため CPU の数を増加させやすく，また独自のメモリ環境において最適化を行うことで高速化が可能という特徴がある．並列局面探索を実装した研究対象はいまだ少ないため，Cell におけるゲーム木の探索は Intel 系のマルチコア環境よりも適している可能性がある．そこで本研究では，独特な環境でも行えるゲーム木探索の高速化を目的とし，並列局面探索の効率化を目指す．

すでにこれまでに筆者らは Cell と，Intel 系に代表される他の CPU における並列局面探索の比較を行っている．並列局面探索におけるタスクを定義し，それに基づいて既存の将棋プログラムを並列化して Cell に実装した．これと基となったプログラムと対局実験を行った．まずは対局の勝率における比較を行っている．この Cell における並列将棋プログラムと，それのもとになった Intel 系シングルコアにおける逐次プログラムの対局実験を行ったところ，Cell の勝率は 200 回対局で 46.5%となり従来の Intel 系 CPU のシングルコアに及ばない強さであった．ここから Cell 上における探索の改善点を探した．

まず，並列研究の一部ではキャッシュを用いることで使用 CPU 数以上の高速化を得られる可能

性があるとされている．このことから局面探索におけるキャッシュの役割であるトランスポジションテーブルに並列化の効果が大きいのではないかと考えた．この並列トランスポジションテーブルの主なアルゴリズムとしてTDSAB[1]があるが，ここで待ち行列の優先度の設定も探索効率の向上に効果があるのではないかとされている．よって並列環境における探索の効率化には待ち行列の改良が良いと推論した．

また，Cell 上における構造上の問題点として、固有メモリの有効利用が必要であった．メモリの少なさは探索の効率に直結し，タスク配分による負荷分散の影響が大きく反映される．この小さいメモリ環境を改善するにあたって，タスク配分を決める待ち行列の改良が効果あると考えた．

これらの理由により，Cell の独自な環境においても実現できる効率化のために，待ち行列の改良によって並列環境の負荷分散を行った．

タスク配分の負荷分散の解決方法として，既存の並列手法に対して待ち行列の構造を変更した．この構造を設計するにあたって，他分野などで用いられている既存の待ち行列を並列探索に適用した．比較した待ち行列の構造は，”深さ別待ち行列”，”深さ別優先度付き待ち行列”，”可能手数による重み付き待ち行列”の三種類である．これらの比較実験として，先の対局実験において用いた並列プログラムに改良として加え，対局した．各アルゴリズムで 200 回対局し，その結果，深さ別待ち行列が 53.5%の勝率を得た．

さらに勝率以外の要素を調べるために局面探索における枝刈りの回数と，並列処理において各プロセッサが何もしない空き時間を比較した．3 種類の待ち行列の構造においてどれが効率的か調べ，それぞれの要素が影響しているか考察した．

1．2　環境としての Cell

プロセッサ環境としての Cell Broadband Engine（以下 Cell）は PLAYSTATION3®などに用いられていて，現在の Intel 系などと比べて比較的安価であることから，個人ユーザにも扱えるコア数が多い．PPU と SPU という異なるコアを持つ構造であること，記憶域を共通メモリ(256MB)と固有メモリ(256KB)において小さいながらも二種類扱える独自のメモリ構造などが挙げられる．（図１．２）

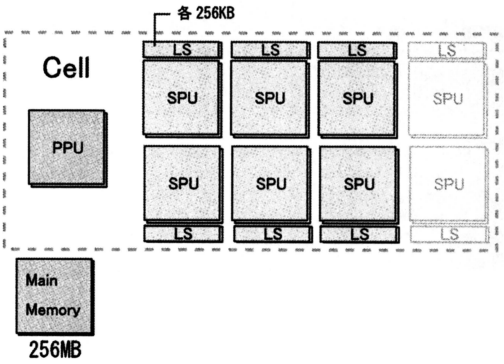

図１．２　Cell の構成図

また，Cell でのゲーム研究として，モンテカルロ法の研究[7]など並列化しやすいものの実装例はあるが，min-max 法に基づく局面探索は少ない．本研究においては Cell という環境が探索において新たな可能性があるかを探すという意味も持つ．

1．3　基本となる並列局面探索

並列局面探索における待ち行列によるタスクの受け渡しの基本型をここで説明する．（図 1）

まず，PPU から始まるプログラムは複数 SPU を起動させる．SPU の最初の一つは局面探索におけるルートノードを受け取り，Produce 機能からスタートする．Produce の機能はタスクをバッファに格納することである．

さて，残りすべての SPU は Consume 機能からスタートする．Consume 機能はバッファからタスクを取り出し，αβの再帰部分を呼ぶことである．αβの再帰は当然αβの展開部分を呼び，展開しながら末端まで再帰し続ける．このαβの展開において，次ノードを展開したのち Produce を呼び，タスクを待ち行列にエンキューする．ここで他のプロセッサがビジー状態のときは自分で取り出して逐次のように処理することができる．

末端までたどり着いたら以降は再帰を辿ってきた順番に返し，Consume 機能まで戻る．ここで値の更新や枝刈りを行い，今回のタスクが終了となる．Consume を抜けた SPU は while でバッ

ファにタスクがある限り Consume をループする.
以上が基本の並列動作となる.

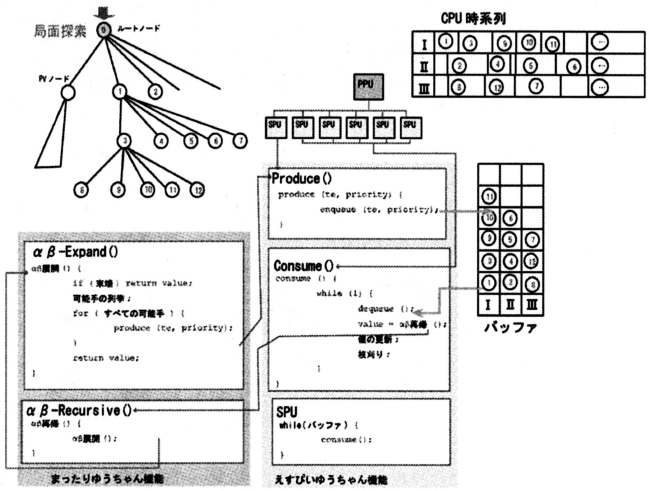

図１　基本の並列動作

２．待ち行列における優先度の改良

並列環境での局面探索は PV-Splitting[3]や YBWC[4]に代表されるような手法では，探索の順序が大きく枝刈りに係るとして PV ノードを先に探索する．このことから，タスクの負荷分散を考えるときに，処理の順序である待ち行列の形が大きく関係する．ここではタスク分散の待ち行列における構造によって処理の効率，ひいては枝刈りの回数増加からの探索速度上昇，結果としてゲームにおける強さの向上を目指す．

２．１　深さ別待ち行列

「深さ別待ち行列」とは，深さごとに待ち行列を用意し，優先度はその深さにおいてのオーダリングを用いたものである．

並列環境で深さ別に待ち行列を用意するとき，プロセッサ数だけ用意する必要がある．つまり，深さの上限×プロセッサ数だけ待ち行列が必要になる．これらにおいて，深さについて順番に回るラウンドロビン方式でタスクを取り出している．

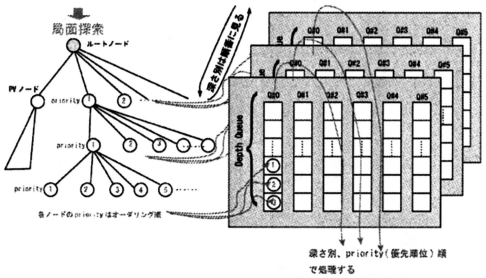

図２．１．１　深さ別待ち行列

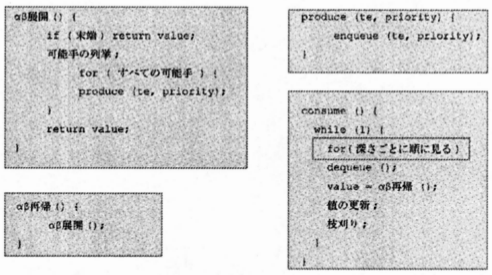

図２．１．２　深さ別待ち行列の疑似コード

２．２　深さ別優先度付き待ち行列

TDSAB の研究[1]において，優先度付き待ち行列が用いられている．

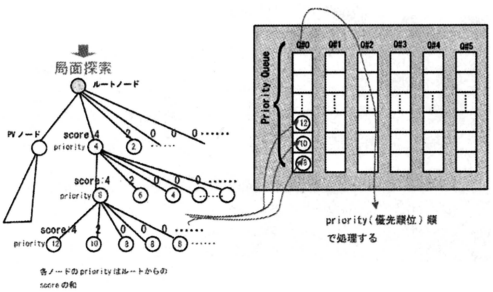

図２．２．１　優先度付き待ち行列

```
αβ展開 (score) {
    if ( 末端 ) return value;
    可能手の列挙；
    priority = score + 定数（オーダリング）
    for ( すべての可能手 ) {
        produce (te, priority);
    }
    return value;
}

αβ再帰 () {
    αβ展開 (priority);
}

produce (te, priority) {
    enqueue (te, priority);
}

consume () {
    while (1) {
        // 優先度順に取り出す
        dequeue ();
        value = αβ再帰 ();
        値の更新；
        枝刈り；
    }
}
```

図２．２．２　優先度付き待ち行列の疑似コード

ここでは，２．１の深さ別であるバッファに保存し，加えて優先度を付随して用いることにした．

まず，各局面探索の深さにおいて前回の反復深化によるムーブオーダリングを行う．このときにオーダリングの値を良い順から定数(可能手数：nとしたとき，良いものから n, n-1, n-2, …, 1の定数)を score として記録する．

次に子ノードを展開する．このときに各ノードに対してルートノードからの score を加算し，その値を優先度（priority）とする．

priority = score +オーダリングによる定数　(式１)

その結果求まる優先度をノードに付加し，深さ別の待ち行列に配置する．

２．３　重み付き待ち行列

ここでの「重み付き待ち行列」とは，重みを優先度としてそのノードにおける可能手数と近似して計算したものである．　基本的なタスクの大きさは可能手の広がりであり，この可能手数≒優先度として扱う．ここではタスクが大きいものほど他の CPU が待つ時間が長いと考え，タスクが大きいものほど先に処理をする必要があるとする．よって，優先度が大きいものから処理される順番になる．

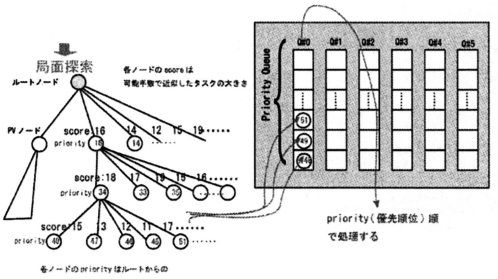

図２．３．１　重み付き待ち行列

```
αβ展開 (score) {
    if ( 末端 ) return value;
    可能手の列挙；
    priority = score + 定数（可能手数）
    for ( すべての可能手 ) {
        produce (te, priority);
    }
    return value;
}

αβ再帰 () {
    αβ展開 (priority);
}

produce (te, priority) {
    enqueue (te, priority);
}

consume () {
    while (1) {
        // 優先度順に取り出す
        dequeue ();
        value = αβ再帰 ();
        値の更新；
        枝刈り；
    }
}
```

図２．３．２　重み付き待ち行列の疑似コード

３．対局実験による検証

Cell において２章におけるアルゴリズムの実装をし，対局実験を行う．

３．１　対局環境

対局に用いるのは本研究室で開発されている将棋プログラム「まったりゆうちゃん」である．これは逐次で動作する将棋プログラムであるが，これを並列環境である Cell に移植し，「えすぴいゆうちゃん」とした．以下のプログラムはこのえすぴいゆうちゃんの基本となる状態(Base)に各アルゴリズム

を加えたものである.

3．2　対局設定

一手3秒時間打ち切りで 200 回対局を行い，勝率＝(勝ち数 / 試合数)として計測した.

組み合わせとしては移植前の Intel 系プロセッサで逐次動作するまったりゆうちゃん(Original)を相手とし，その勝率が上昇するかどうかをみる.

基本となる並列プログラムを"Base"とし，それぞれ深さ別待ち行列を"Depth"，深さ別優先度付き待ち行列を"Priority"，重み付き待ち行列を"Weight"として次の表に結果を示す.

3．3　結果

対局の結果を以下の表に示す.

表3．1　対局結果（Base 対 Original）

	CPU	勝ち数	勝率
Base	Cell	93	46.5%
Original	Intel	107	53.5%
計		200	

表3．2　対局結果（Depth 対 Original）

	CPU	勝ち数	勝率
Depth	Cell	107	53.5%
Original	Intel	93	46.5%
計		200	

表3．3　対局結果（Priority 対 Original）

	CPU	勝ち数	勝率
Priority	Cell	102	51.0%
Original	Intel	98	49.0%
計		200	

表3．4　対局結果（Weight 対 Original）

	CPU	勝ち数	勝率
Weight	Cell	92	46.0%
Original	Intel	108	54.0%
計		200	

3．4　結果のまとめ

対局実験では Base の勝率が 46.5%から，各種変更した値として，Depth が 53.5%，Priority が 51.0%，Weight が 46.0%という結果になった.

4．枝刈りと CPU 空き時間の比較

並列処理において複数のプロセッサが100%稼働することは難しい．しかし，無駄のない負荷分散が行えていれば空き時間が減少するはずである.

4．1　枝刈り回数の比較実験

局面探索の最も大きい効率化は枝刈りの回数が増加することである．とくに並列環境であれば重複探索を抑えることで全体的な探索速度が上昇すると思われる.

各待ち行列の構造に対し，一手3秒制限の探索中に枝刈りが何回発生しているか比較した．値はそれぞれ 100 対局中の発生回数を調べ平均化したものである.

結果は表4．1に示す.

4．2　空き時間の比較実験

各待ち行列の構造に対し，一手3秒制限の探索中に各プロセッサが何もしていない状態（空き時間）を比較した．値はそれぞれ 100 対局中の発生回数を調べ平均化したものである.

表4．2に結果を示す.

表４．１　一手３秒探索中においての枝刈り発生回数

	Base	Depth	Priority	Weight
枝刈り回数(回/一手)	123,115	117,407	120,948	115,711

表４．２　一手３秒探索中においての各 CPU の空き時間

	Base	Depth	Priority	Weight
空き時間[s]	0.532	0.476	0.523	0.510

４．３　結果のまとめ

枝刈りでは一手３秒における探索の平均として，Base が 123,115 ，Depth が 117,407，Priority が 120,948，Weight が 115,711 になった．

空き時間では同様に一手 3 秒として Base が 0.532[ms]，Depth が 0.476[ms]，Priority が 0.523[ms]，Weight が 0.510[ms]となった．

５．考察

対局の結果はあまり大きく差がついた値ではないが，Cell で Intel 系の逐次プログラムにようやく追いついた状態である．しかし，Intel 系のマルチコア並列プログラムと比較したらもっと厳しい数字になっていただろう．単純な並列化のみでは速度は上昇しても強さは追いつくことが難しい．これはメモリが限定されていることも影響が大きく，単純な探索速度だけが強さではないということが言える．並列な Intel 系との性能比較が今後に必要であり，並列化による効率上昇の度合では Cell にも有利な点があると思われる．

次に，結果から得られる待ち行列の構造の違いにおいて，枝刈り率は探索のゲーム木が変わってくるために単純な数だけでは比較しづらい．ただ，3 つのうちでは Priority が最も高いのだが，これに関しては Base の方が高い値を示している．これは探索中のゲーム木において効率よく枝刈りされるときは木の浅い方，つまりルートに近い方から刈られるため，枝刈り回数が少ないとも考えられる．末端の方で小さい枝刈りを数多く行っても効率は良くないと思われる．

これらの理由を考察する．深さ別待ち行列は計算量を省くために計算結果のテーブルをあらかじめメモリに確保しておくので効率は良いと思われるが，メモリの有効利用という観点からは外れてくる．優先度付き待ち行列は，何を優先度として計算するかという選択の幅広いので，有効そうな要素が挙げられれば今後に一番発展すると思われる構造である．現在はムーブオーダリングとルートからのパスであるが，今回の深さ別待ち行列の効果を考慮すると，深さを重視した計算にするとより効率的ではないかと考えられる．今回，重み付き待ち行列は可能手数という値があまり影響を与えないことから，優先度の計算として組み込んでいくのが良いのではないかと考える．

Cell にはまだ台数増加性によりクラスタ化，Cell Blade サーバの利用を含め速度は上昇が見込めるので，メモリの扱いについて効率の良い構造を用いることで Intel 系マルチコアなどのほかの並列環境にも対抗できる可能性はあると思われる．

６．おわりに

本章では，Cell の環境における局面探索の速度を向上させるための待ち行列の改良についての結論を述べる．

Cell における台数増加性やメモリなど独特の環境を用いて，局面探索を効率化できるかを目指した．Cell による並列のゲーム木探索をする一例として実験を行った．

待ち行列の性能の計測からは，Base を元に”深さ別待ち行列”，”優先度付き待ち行列”，”重み付き待ち行列”の構造を比較した．

まずは Base に”深さ別”を加えた構造と最初に用いた逐次プログラム(Original)とを再対局させ，53%の勝率まで引き上げた．これにより五分以上の強さとなったといえる．さらに，細かい要素の調査として，探索時の枝刈り回数とプロセッサの空き時間を計測して比較した．その結果，今回の

三つの手法では優先度付き待ち行列が 1 手 3 秒あたりの枝刈り回数 120,948 回で最も良かったが，プロセッサの空き時間の比較では深さ別待ち行列が 1 手 3 秒あたり 0.476 秒の空き時間という結果になった．両者とも Base からの改善という意味では効果を上げている．優先度付き待ち行列には優先度の設定と計算方法という選択の幅が大きくあり，今後のさらなる改善に可能性がある．

参考文献

[1] Akihiro Kishimoto, Transposition Table Driven Scheduling for Two-Player Games, M.sc. Thesis, University of Alberta, 2002

[2] J.W.Romeinm, A.Plaat, H.E.Bal, and J.Schaeffer. In 16th National Conference on Artificial Intelligence (AAAI'99), pages 725-731, Orlando, 1999.

[3] T. A. Marsland and M. S. Campbell. Parallel Search of Strongly Ordered Game Trees, ACM Computing Surveys, 14(4): pp533-551, 1982.

[4] R.Feldmann, P.Mysliwietz, and B.Monien. Game tree search on a massively parallel system, Advances in Computer Chess 7, pp203-219, 1993

[5] Mark G. Brockington. Asynchronous Parallel Game-Tree Search, PhD thesis, University of Alberta, Edmonton, Canada, 1998

[6] 柳 圭二郎, 柴原 一友, 但馬 康宏, 小谷 善行：並列局面探索における待ち行列の負荷分散, Game Programming Workshop 2007, pp152-155, 2007

[7] 加藤 英樹, 竹内 郁雄：並列 MC/UCT アルゴリズムの実装, Game Programming Workshop 2007, pp23-29, 2007

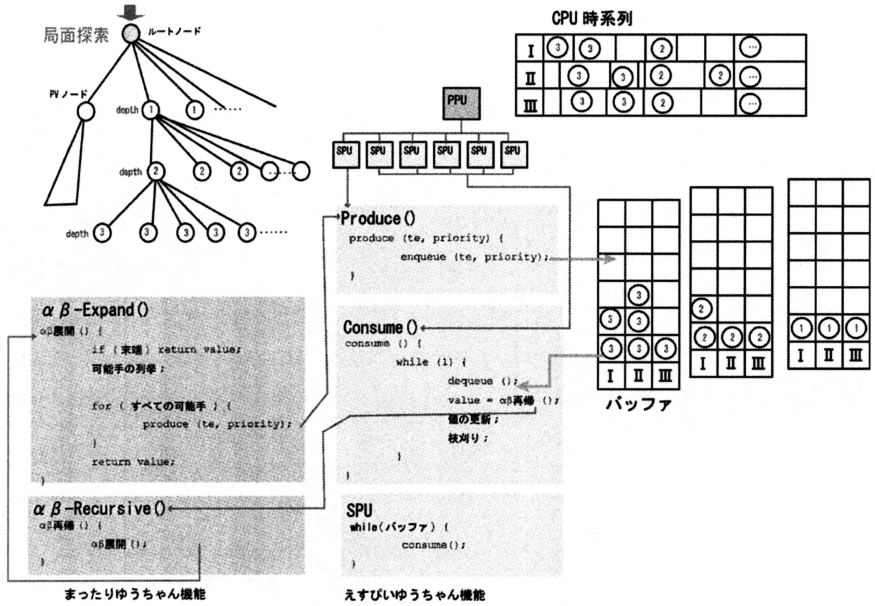

図 1　深さ別待ち行列の全体動作

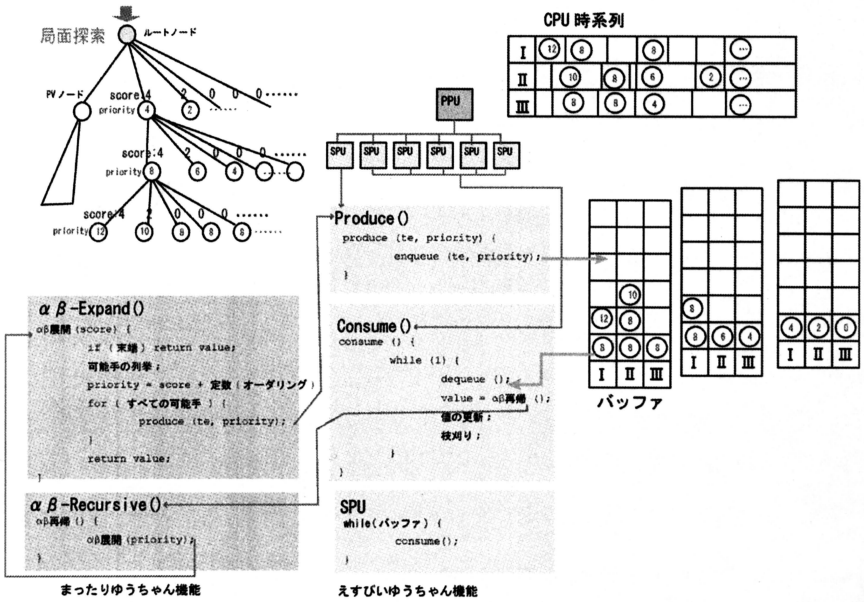

図２　深さ別優先度付き待ち行列の全体動作

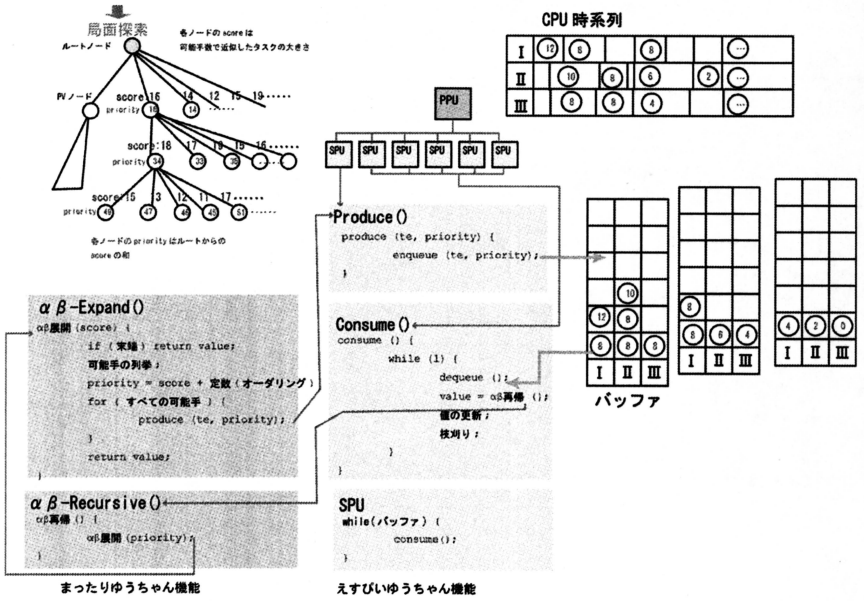

図３　重み付き待ち行列の全体動作