ヘテロジニアスマルチプロセッサのためのタスク分散手法

三 好 健 文[†1] 笹 田 耕 一[†1]

ヘテロジニアスマルチプロセッサは異種のコアの組み合わせにより構成されるプロセッサである．プログラムの並列性に着目し，個々の特徴に適した割り当てによって省スペース，低消費電力で高い計算性能を得ることができる．しかし，本来記述したい動作に加え，効率良く動作させるために考慮しなければならないことが多く，プログラミングが困難である．この解決のために，ヘテロジニアスマルチプロセッサ向けに容易にプログラムを記述することができるランタイムフレームワーク及び，これをバックエンドとする自動並列化手法について検討する．提案するランタイムフレームワークは，効率良く計算資源を利用するために，与えられたプログラムを粗粒度のタスクに分割し，静的あるいは動的にスケジューリングする．また，タスク単位でのプログラムの分割は，粗粒度自動並列化手法によるコード生成との親和性が高い．

本稿では提案手法であるランタイムフレームワークと自動並列化手法について述べる．また，Cell B.E. を対象に提案するフレームワークを実装し，実行結果を評価した．提案手法に基づいて実行することで，静的なタスクと動的なタスクの協調により効率の良い並列実行を実現しプログラムの実行時間は 93%になった．

A Study of Task Distribution Framework for Heterogeneous Multi Processor

Takefumi Miyoshi[†1] and Koichi Sasada[†1]

In this paper task distribution framework for heterogeneous multi processor in order to program easily is described.

Heterogeneous multi processor consists of processing cores which have variant processor architectures and properties. It pays attention to the parallelism of the program, and a high calculation performance can be obtained with small chip size and low power consumption by the allocation that is appropriate for an individual feature of each core. However, it is necessary to consider to operate efficiently in addition to the operation that wants to be described originally, and the programming is difficult.

In order to solve this, run-time framework for programming easily onto heterogeneous multi processor and automatic parallelization method are proposed. The proposed framework divides given programs into coarse grained tasks, and assigns them to cores statically and/or dynamically in order to use them efficiently. Moreover, the division of the program into tasks is high compatibility with the coarse grained automatic parallelization method.

In this paper, proposed frame work and automatic parallelization method are described at first. Then, an implementation of the framework onto Cell B.E. is shown as an example, and it is evaluated. By the proposed framework, calculation time is reduced by 93%.

1. はじめに

ヘテロジニアスマルチプロセッサは，アーキテクチャや命令セットの違いなどによって異なる特性を持つコアを複数組み合わせたプロセッサであり，個々のコアの特徴を活用することで低消費電力，省スペースで高い計算処理能力を得ることが期待できる．一方で，効率の良い利用のためには，プログラマ自身が，実現したい動作の記述に加え，その並列化や，並列化した部分プログラム同士の同期，コアの特徴を考慮した割り当てを考慮しなければならない．これは，大変困難で時間のかかる作業である．同一の機能や特性を持つ複数のコアによってもホモジニアスマルチプロセッサでも，プロセッサ内のコア数の増加に伴い，性能にばらつきがでてくることが考えられる．また，プロセッサ内での位置により外部デバイスやメモリとの通信コストの差異が生じる．たとえば Feature-Packing[1] は，多数の同一のプロセッサコアによるホモジニアスマルチプロセッサであるが，幾つかのコアでそれ以外の多数のコアを管理し，計算を実行させるという使い方が考えられている．これらの場合にも，効率の良い計算を行うためにヘテロジニアスプロセッサと同様，個々の特性を考慮したタスクの分割が必要となる．

ヘテロジニアスマルチプロセッサを効率良く利用するプログラミング方式として，マスタ–スレーブ方式がある．これは，豊富な制御命令やメモリアクセス命令を持つコアをマスタコア，特殊な機能や高い計算性

†1 東京大学大学院情報理工学系研究科創造情報学専攻
Dept. of Creative Informatics, The Univ. of Tokyo

能を持つコアをスレーブコアとして用いることで，全体として高い処理性能を得ることができる．このマスタ–スレーブ方式で効率良くプログラムを実行するためには，スレーブコアの高い演算性能の活用することは勿論のこと，マスタコアでの制御集中により生じるボトルネックやコア間のデータ通信コストを考慮する必要がある．

本稿では，ヘテロジニアスマルチプロセッサに対して効率の良いプログラムを容易に記述することができるように，ランタイムフレームワークによってサポートすることを考える．

提案するランタイムフレームワークでは，プログラムをデータ授受のパタン化によるタスクに分割して取り扱う．プロセッサ内の各コアの特徴を活用するためには，事前にタスクのコストや特徴を評価し，適切なコアへ割り当てることができると嬉しい．しかし，プログラム中のタスクは静的に解析できるものばかりではないため，動的な決定が必要不可欠である．提案するフレームワークでは，キューによるタスクプールを用いて，静的なタスクと動的なタスクを協調して効率良く実行する．

また，プログラムをタスクに分割してスケジューリング可能であるため，粗粒度自動並列化との親和性が高い．本稿では，このフレームワークをバックエンドとする自動並列化手法の構成手法について検討する．提案する自動並列化手法ではマクロタスクフローグラフに基づいた制御フロー解析およびデータフロー解析を用いることで，各コアに対し静的に割り当てるタスクと動的に割り当て実行するタスクを生成する．

本稿では，まず，ヘテロジニアスマルチプロセッサについて述べ，次に提案するランタイムフレームワークと粗粒度自動並列化手法について述べる．最後に提案するランタイムフレームワークを Cell Broadband Engine(Cell B.E.)[2] 上に実装する例と，タスクの実行結果について示す．

2. ランタイムフレームワーク

提案するランタイムフレームワークは，与えられたプログラムを粗粒度に分割したタスクとして管理，実行する．タスク単位でのプログラムの実行モデル及びタスク間でのデータ授受をパタン化することで，容易に効率の良いプログラムを実現することができる．

プログラムを粗粒度のタスクに分割することで，自動並列化手法や，スレーブコアの少いメモリ領域を効率良く利用するためのスケジューリング手法と高い親和性を持つことが期待できる．

本節では，まず実行単位であるタスクの定義について述べ，次に，ランタイムフレームワークにおけるスケジューリングモデルについて説明する．また，動的なタスク生成に関するプログラミングモデルについて述べる．

```
struct task{
  タスクの ID
  タスクの型（関数ポインタの識別）
  タスクの実行状態
  タスクの状態管理変数
  パラメタ
   入力データ格納先アドレス
   入力データサイズ
   出力データ格納先アドレス
   出力データサイズ
};
```

図 1 タスクの定義
Fig. 1 A task definition of the proposed framework

2.1 タスクの定義

提案するフレームワークでは，実行単位であるタスクの実行モデル及びタスク間のデータ授受をパタン化することによって容易なプログラミングを実現する．タスクは，タスクを実行する関数へのポインタと，実行時パラメタで定義される．擬似的な C 言語で記述したタスクの定義を**図 1** に示す．

フレームワーク中でタスクは静的なタスクと動的なタスクのいづれかとして取り扱われる．静的なタスクはプログラマあるいはコンパイラによって実行前に割り当てるべきコアや，実行順序が決定されるタスクであり，動的なタスクを実行するコアの割り当ては実行時に動的に決定される．

動的なタスクは 2.2 節で述べるようにタスクプールで管理される．そのため，タスク自体のサイズはあまり大きくしたくない．パラメタは，入出力のデータの格納先アドレスおよびそれらのサイズを保持できる程度とする．ただし，タスクへの入出力データが小さい場合にはこのフィールドに直接値を代入することもでき，この場合，データ転送オーバヘッドを削減することができる．

2.2 タスクスケジューリング

静的にコアへの割り当てを決定することが可能なタスクは，タスクの管理や割り当てに要するオーバヘッドなく効率良い実行が可能である．しかし，一般的にはすべての処理の割り当てを静的に決定することは困難であり，実行時に割り当て決定する必要がある．そのため，各コアのタスク実行のばらつきによる空き時間を効率良く利用することができるように，静的なタスクと動的なタスクを組み合わせる．静的なタスクはソースコードに埋め込まれ，コンパイル時に割り当てと実行するタイミングが決定される．動的なタスクは，キューによるタスクプールによって管理され，計算コアへの割り当てと実行するタイミングは実行時に決定される．

ソースコードに埋め込まれる静的なタスクと動的なタスクのスケジューリングについて述べる．**図 2** にスケジューリングの例を示す．図 2 中の四角のノードはタスクを表わし，ノード内のラベルはタスクの型である．図 2 には，A，B と X の型のタスクがある．タス

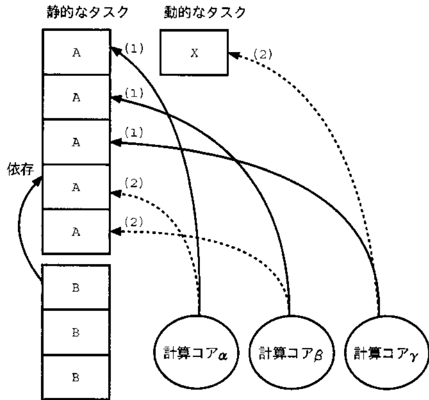

図 2 タスクスケジューリングの例
Fig. 2 An example of task scheduling

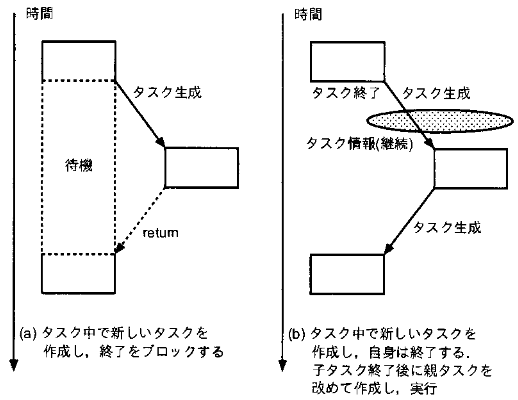

図 3 動的なタスクの生成モデル
Fig. 3 The model of dynamic child task generation

ク型 B のタスク群はタスク型 A のタスク群に依存するとし，タスク型 X のタスクはいつでも実行することができるとする．図中 (1) の辺で示すようにすべての計算コアがタスク A を実行後，次のタスクを実行しようとすると計算コアが一つ空いてしまう．動的なスケジューリングと協調することで，図中 (2) の辺で示すように，計算コア α，β でタスク A を計算コア γ に動的なタスク X を実行させることができる．

動的なタスクの取得，実行手法として，次の 2 つの手法が考えられる．

(1) 各スレーブコアが独自のタスクプールを管理し，自律的にタスクを奪い合う

(2) マスタコアがタスクプールを管理し，スレーブコアはマスタコアへタスクを問い合わせる

各スレーブコアが自律的にタスクを奪い合う手法では，マスタコアの計算資源の消費や共有するタスクプールへのアクセス競合による待ち時間がなく，効率良いタスク割当てが実現できる．しかし，その一方で，各計算コアが貧弱である場合には，計算コアが自律的にタスクを生成，取得するためのコストが大きい．従って，アーキテクチャによって，いづれの手法を選択するか検討する必要がある．

2.3 動的なタスクの実行モデル

動的なタスクはタスクキューに追加され，空いているコアによって取得され実行される．したがって，タスク中で新しいタスクを動的に生成する時，**図 3**(a) のように新しいタスクの終了をブロックするとデッドロックが生じる．これを回避するために，動的なタスクを生成する場合には，タスクの生成後に自身は終了しなければならない．再帰的なプログラムを記述するためには，図 3(b) に示すように，継続を用いて明示的に記述する必要がある．

```
int array(int *a, int *b, int size)
{
  volatile int x, y;
  for(x = 0; x < size; x++) ;
  for(y = 0; y < size; y++) ;
}
```

図 4 例題プログラム
Fig. 4 An example program

3. 粗粒度自動並列化

タスク単位で分割したプログラムを各コアに割り当てて実行する提案手法では，粗粒度自動並列化で扱う単位をそのまま取り扱うことができる．本稿では，マクロタスクフローグラフに基づく粗粒度自動並列化手法の CoCo[3] をベースとして，提案フレームワークをバックエンドとするタスクのコードの生成について述べる．

3.1 CoCo による粗粒度自動並列化

CoCo[3] はマクロタスクフローグラフに基づく自動粗粒度並列化を実現する．

マクロタスクフローグラフは，プログラム中の粗粒度でのフロー解析を行うことを目的として，複数の基本ブロックからなるマクロタスクを単位として得られるフローグラフである．マクロタスクでは，if や for によって囲まれる基本ブロックを一つの単位とすることで内部の構造を隠蔽することができる．例えば，**図 4** に示すプログラムからは，**図 5** に示すマクロタスクフローグラフが得られる．ここで，丸のノードが基本ブロックを四角の枠がマクロタスクを示している．

CoCo ではマクロタスクフローグラフを生成し，制御依存およびデータ依存の解析結果に基づいて粗粒度の自動並列化を実現する．生成するのは，OpenMP のディレクティブを含むコードである．並列実行可能な個々のマクロタスクは，switch–case 文によってス

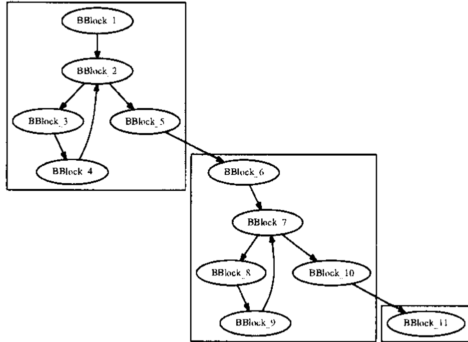

図 5　図 4 から得られるマクロタスクフローグラフの例
Fig. 5　An example of macro task flow graph generated from fig. 4

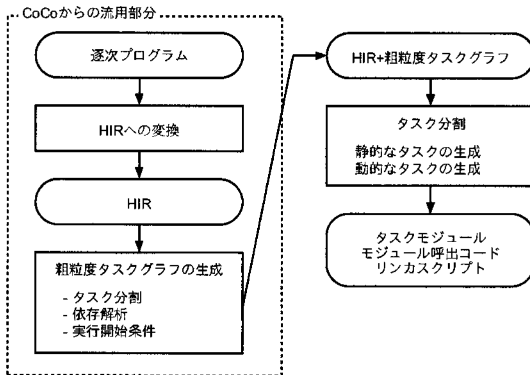

図 6　提案フレームワーク向け自動並列化
Fig. 6　A diagram of propsed automatic parallelization method based on CoCo

レッドに分割され，並列に実行される．

3.2　コード生成

CoCo における OpenMP 向けのコード生成部分を変更し，提案フレームワーク向けのコードを生成する．**図 6** に自動並列化のダイグラムを示す．図中，点線部分で囲んだ部分が CoCo からの流用部分を示している．提案するフレームワークをバックエンドとする場合，CoCo で switch–case 文を用いてスレッドに分割したマクロタスクをそれぞれタスクモジュールとして切り出すとよい．切り出したタスクモジュールは，その実行条件を判定し，静的なタスクと動的なタスクに分割される．静的なタスクは，計算コアのソースコードに埋めこまれ，動的なタスクは，タスクプールに格納されるコードとして生成される．

4.　実装と評価

提案するフレームワークを Cell B.E.[2] を対象として実装する例を示す．また，簡単な例により，タスクスケジューリングによる並列実行の結果を示し，実行時間を評価する．

4.1　Cell B.E. への実装

Cell B.E. は汎用プロセッサである PPE と 128bit の SIMD プロセッサである SPE から成る．PPE は汎用プロセッサならではの十分な制御命令，例外処理機構などを持ち，またメインメモリに直接アクセスすることができる．SPE は，SIMD 命令による高効率の演算を実行することが可能であるが，一方でメインメモリへ直接アクセスすることができない等，PPE に比べ制約が大きい．従って，PPE をマスタコア，SPE をスレーブコアとした場合の提案フレームワークの実装例について述べる．ここで，SPE の処理能力を考慮するとタスクプールのキュー操作は PPE で実行する方が良い．

4.2　起動と初期化

Cell B.E. では，PPE を用いて SPE のバイナリイメージのロードを行う必要がある．一般に SPE のローカルストレージは，実現したいプログラムに対して十分な大きさではない．従ってタスクモジュールをオーバレイを用いて SPE へロードし，起動する．オーバレイによって実行する全てのタスクのバイナリイメージを仮想的に一度にロードすることができる．起動後に初期設定として，各 SPE に割り当てられたグローバルアドレス空間上のアドレスおよびキューのアドレスを通知する．

4.2.1　タスクスケジューリング

SPE に埋めこんだ静的なタスクと PPE で管理される動的なタスクのスケジューリングの例を**図 7** に示す．SPE は，まずソースコード中に埋め込まれた静的なタスクを実行する．タスクの実行終了や，他のタスクとの同期待ちによる空き時間に，PPE にあるタスクプールからタスクを取得し，実行する．ここで，スレーブコアである SPE は複数個あるため，PPE へのアクセスは排他的に取り扱う必要がある．また，同期を行うためのバリア変数へのアクセスにも排他的な処理が必要である．

Cell B.E. では，PPE–SPE 間のデータ転送は，SPE の持つ MFC を経由して実現される．MFC を用いた排他制御は，アトミック転送命令を用いて実現することができる．アトミック転送命令はコストの高い処理であるため，タスクの取得にはロック変数を用いた排他制御を行う．ロック変数へのアクセスのみにアトミック転送命令を用い，タスクのデータ授受には単に DMA 転送命令を用いることでオーバヘッドの小さいデータ通信を実現することができる．

4.3　スケジューリング結果

動的なタスクと静的なタスクの協調によって，プログラムを効率良く実行することができること確認する．例題として，4 つのタスク型を考える．素数を計算するタスク型 “PRIME” と，これに依存するタスク型 “B” 及び，これらと依存関係にないタスク型 “X” と

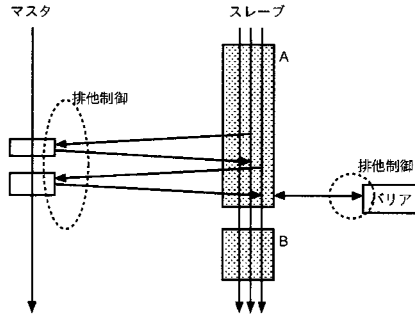

図 7　PPE でタスク管理をする静的タスクと動的タスクのスケジューラ
Fig. 7　Dynamic and static tasks Scheduler managed by PPE

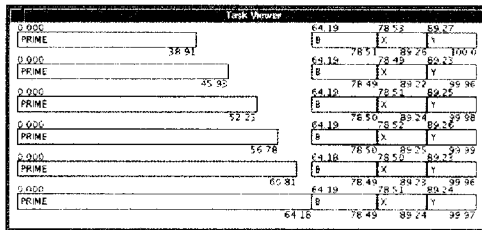

図 8　静的なスケジューリングを行った場合
Fig. 8　The result of executing task by static scheduling

"Y" である．素数の計算を行うタスク型 "PRIME" は，60000 までの素数を求める計算であり，静的なタスクとして，各コアのインデックスに応じて割り当てられている．したがって，"PRIME" の各コアに対する計算負荷は自然な不均一さを持つ．タスク型 "B" は 400000 回，"X" および "Y" は 300000 回の定数回の空ループである．実行環境は，Sony Playstation 3, Linux cell 2.6.16-ps3 であり，SPE は 6 基使用した．

これを，一般的なプログラミング手法によって静的に割りあてた場合のタスクの実行結果を**図 8** に示す．各行は計算コアである SPE を，横軸は時間を示している．各ブロックはブロック内に記したタスクの実行を示し，その開始時刻と終了時刻がブロックの左上，右下にそれぞれ記されている．時刻は，すべてのタスクが完了した時刻を 100 として正規化している．

一方**図 9** は提案するランタイムフレームによって静的および動的にタスクを割り当て実行した結果を示している．図 9 より提案するランタイムフレームでは静的なタスクと動的なタスクを効率良く並列に実行できていることが分かる．また総実行時間は，静的に実行した場合が 20.169m 秒，提案フレームーアクに基づき動的なスケジューリングと協調した場合 18.764m 秒であり，提案手法による実行では，実行時間は 93%になった．

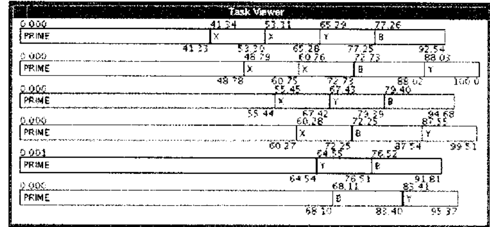

図 9　提案フレームワークにより静的/動的スケジューリングを行った場合
Fig. 9　The result of executing task by static and dynamic scheduling on the proposed framework

5. 関連研究

関連研究として並列化ライブラリやフレームワークを挙げる．また，ヘテロジニアスプロセッサの例として挙げた Cell B.E. を対象としたプログラミングモデルやライブラリについて述べる．

5.1 並列化ライブラリ/フレームワーク

並列プログラミングをサポートするフレームワークとして，OpenMP[4] と MPI[5] がある．OpenMP は，共有メモリ型の並列プログラミングの共有 API である．サポートするコンパイラも多く，バックエンドとして OpenMP を用いる自動並列化コンパイラも多い[6]．一方，MPI は，分散計算資源を扱うフレームワークである．MPI では分散する計算資源を扱うために，コア間の協調動作をメッセージとして取り扱う．しかしプログラマが明示的に記述しなければならないためプログラムは難しい．さらに，ヘテロジニアスプロセッサのように不均質なコアを考慮する場合，各コア性能の偏りを考慮する必要があり期待する性能を得ることが難しい．

文献 7) では，ヘテロジニアスプロセッサ向けの効率の良いタスク分割を実現する手法を提案している．しかし，扱うタスクの対象が等しい大きさのタスクに限定されている．また動的に分割を決定することが困難である．

動的なタスク分割をサポートするライブラリ/フレームワークとして，Cilk[8] や，Intel Threding Building Block[9] がある．Cilk[8] は，C 言語をベースとしたマルチスレッド並列プログラミングのためのランタイムシステムである．各スレッドのタスクを計算資源の余っているスレッドが取得し，実行することで効率の良い動的なタスク分割を実現している．また，Intel Threading Building Block[9] は，並列プログラミングをサポートする C++ベースのテンプレートライブラリである．プログラマは明示的なスレッド生成や同期を意識することなく本来実現したいプログラム設計に注力することができる．動的なタスクのスケジューリング手法に提案手法は大きく影響を受けている．しかし，これらはヘテロジニアスマルチプロセッサを考慮

しているものではなく，特にメモリ操作や制御命令の機能が弱いコアがあるような場合には，効率の良い計算性能を得ることが期待できない．

5.2 Cell B.E. 向けライブラリ

Cell B.E. プロセッサを効率良く利用するためのコンパイラによる最適化手法は，文献10) に詳しい．容易なプログラミングを可能にするライブラリやプログラミングモデルには，SPURS[11] や CTK[12] がある．

また，CellVM[13] は，Cell B.E. プロセッサにおいて，SPE と PPE のデータ転送やメモリ量の制限をプログラマに隠蔽することを目的とした Java ベースの仮想マシン環境であり，PPE で実行される SellVM と，SPE で実行される CoreVM から成る．ShellVM では，Java でスレッド分割された各スレッドを CoreVM に割り当て，SPE を効率良く利用した演算を実現する．しかし，Java のスレッド単位で CoreVM へ部分プログラムを割り当てるため柔軟に部分プログラムを切り出し割り当てることが困難である．

6. まとめ

本稿では，静的なタスクと動的なタスクの協調した動作を実現するスケジューラをもつランタイムフレームワークと，それをバックエンドとする自動並列化手法を提案した．提案したフレームワークでは，簡単なタスク定義と，キューによるタスク管理手法によって静的なタスクと動的なタスクを協調して効率良く実行することができる．また，粗粒度自動並列化によるコード生成手法について述べた．提案するフレームワークを Cell B.E. 上で実現し，具体的な実装手法および，動的なタスク割り当てによる実行を確認した．

今後の課題としては，提案したフレームワークのもとで，異なるプロセッサの特徴を静的，動的な解析によって活用する手法について検討することが考えられる．また，自動並列化による効率の良いコード生成について検討する．特に実用的な自動並列化を考える場合，データ並列性を抽出し，データ転送コストとのオーバヘッドを考慮した割り当て実行を行うことが重要である．

さらに，他のヘテロジニアスマルチプロセッサやホモジニアスなメニーコアへ提案手法を適用し，その評価を行うことも今後の課題である．

参考文献

1) 小林良太郎, 吉瀬謙二. 多機能メニーコアを実現するアーキテクチャ技術 feature-packing の構想 (inventive and creative architecture 特別セッション i). 情報処理学会研究報告. 計算機アーキテクチャ研究会報告, Vol. 2007, No. 115, pp. 11–15, 20071121.
2) IBM Systems and Technology Group. Cell Broadband Engine Programming Handbook Version 1.1, 4 2007.
3) 池田倫久, VANNGO TAU, 田中雅俊, 福岡岳穂, 片桐孝洋, 本多弘樹, 弓場敏嗣. 粗粒度並列化コンパイラ coco の開発 (並列処理のためのシステム). 情報処理学会研究報告. [ハイパフォーマンスコンピューティング], Vol. 2004, No.38, pp. 37–42, 20040413.
4) Leonardo Dagum and Ramesh Menon. Openmp: An industry-standard api for shared-memory programming. *IEEE Computational Science and Engineering*, Vol.05, No.1, pp. 46–55, 1998.
5) William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Comput.*, Vol.22, No.6, pp. 789–828, 1996.
6) GNU Project. Gomp - an openmp implementation for gcc. *http://gcc.gnu.org/projects/gomp/*.
7) Cyril Banino, Olivier Beaumont, Larry Carter, Jeanne Ferrante, Arnaud Legrand, and Yves Robert. Scheduling strategies for master-slave tasking on heterogeneous processor platforms. *IEEE Transactions on Parallel and Distributed Systems*, Vol. 15, No. 4, pp. 319–330, 2004.
8) Robert D. Blumofe, Christopher F. Joerg, Bradley C. Kuszmaul, Charles E. Leiserson, Keith H. Randall, and Yuli Zhou. Cilk: an efficient multithreaded runtime system. *SIGPLAN Not.*, Vol.30, No.8, pp. 207–216, 1995.
9) James Reinders, (菅原清文監訳). インテル スレッディング・ビルディング・ブロック – マルチコア時代の C++並列プログラミング. オライリー・ジャパン, 2 2008.
10) Peng Wu, Tao Zhang, DanielA. Prener, Peng Zhao, JaniceC. Shepherd, Michael Gschwind, Byoungro So, Zehra Sura, AlexandreE. Eichenberger, Kathryn O'Brien, Kevin O'Brien, Tong Chen, Amy Wang, and PeterH. Oden. Optimizing compiler for the cell processor. In *PACT '5: Proceedings of the 14th International Conference on Parallel Architectures and Compilation Techniques*, pp. 161–172, Washington, DC, USA, 2005. IEEE Computer Society.
11) 井上敬介. Cell プロセッサ向け実行環境 (spu centric execution model). SACSIS 2006 - 先進的計算基盤システムシンポジウム, 5 2006.
12) http://cell.fixstars.com/ctk/index.php. Ctk: Cell toolkit library.
13) Albert Noll, Andreas Gal, and Michael Franz. Cellvm: A homogeneous virtual machine runtime system for a heterogeneous single-chip multiprocessor. *Technical report, School of Information and Computer Science, University of California, Irvine*, 11 2006.