9F−6

KL1 負荷分散ライブラリの試作

市吉 伸行
(株) 三菱総合研究所 応用技術部 *

1 はじめに — 負荷分散ライブラリとは

現在、並列論理型言語 KL1 向けに試作している負荷分散ライブラリについて述べる。

多くの並列プログラムは、問題依存の局所的・逐次的なプロセスないし手続きを特定の静的ないし動的な負荷分散方式でプロセッサに割り当てる構造をしている。負荷分散ライブラリは、この両者を明確に分離し、典型的な幾つかの負荷分散方式を再利用可能なユーティリティとして提供しようとするものである。ユーザは、対象問題の解法を記述し、ライブラリの中の適当な負荷分散方式と組み合わせることによって並列マシン上で問題を解くことができる。

負荷分散ライブラリには以下のような効用が期待される。

- 並列プログラムの生産性の向上

  負荷分散方式の設計・実装・評価・改良の手間は、並列プログラム開発労力の中の一定の部分を占める。負荷分散ライブラリを用いると、それが選択・評価・再選択（またはライブラリコードの改良）という、差分的なオーバヘッドで済むようになる。

- 並列プログラムの性能の基準点

  問題解法を記述して、負荷分散ライブラリと組み合わせて得られる性能は、ある意味で性能の基準点と見なすことができる。それを出発点として、問題に適したよりよい並列化方式を開発し、何倍の性能が得られたという尺度でもって、開発した並列化方式の評価を行なうことができる。

- 負荷分散方式に関する技術の蓄積

  負荷分散方式を研究開発した場合、公開の負荷分散ライブラリに加えることにより、性能測定の追試、技術の蓄積、相互評価、他人による改良などが期待できる。

これらは理想であって、実際には、解きたい問題に適した負荷分散方式が大抵見つかるほど密にライブラリを備えるのは簡単ではないであろうし、また一般的インタフェースを与えるために実行効率がある程度犠牲になるという問題もあろう。

2 負荷分散ライブラリの方式

前述のように、並列プログラムは概念的に問題依存部分と負荷分散部分とに分けて考えることができるが、実際のプログラムのソースコード上においては、この両者がはっきりと分離しているとは限らない。負荷分散ライブラリを実現するには、これらの適切な分離の仕方を考える必要がある。ここでは今回の負荷分散ライブラリの対象言語である並列論理型言語 KL1 に即して考えてみる。

2.1 負荷分散プラグマ

KL1 言語は並行論理型言語 GHC をベースとしており、GHC のゴール（手続き呼び出しに相当）にプラグマと呼ばれる標記を付加することによって、ゴール単位の負荷分散を指定することができる [2]。例えば、N 個のプロセスを巡回的にプロセッサに割当てる述語（手続き）を次のように書くことができる。

```
fork(N,PE) :- N > 0 |
    spawn @node(PE),
    NextPE := PE + 1,
    fork(N-1,NextPE).
```

ここで、spawn プロセスを実行すべきプロセッサを @node(PE) というプラグマが指定している。[1]

KL1 言語では、負荷分散を指定しないプログラムをまず作成し、それに対してプラグマの付け方を工夫することで負荷分散方式の実験をすることができる。その際、プログラムの正しさがプラグマの付け方に左右されないのが、KL1 の優れた特徴である。

2.2 プロセッサ番号サーバ方式

KL1 プログラムにおいて負荷分散を直接指定する部分はプラグマという形で、他の部分から明確に分離されているが、再利用可能な形でそれを抽出してライブラリ化することは困難である。通常、実際のプログラム・ソースの中でプラグマの現れる回数は少なく、むしろ、プロセッサ番号の計算方法の記述の方が量が多い。そこで、後者の部分のみをカプセル化したサーバを用意することが考えられる。この方式では、プログラム開始初期にサーバを起動し、負荷分散ポイントでサーバにプロセッサ番号を要求することになる。例えば、前出のプログラム断片は次のようになる。

```
fork(N,PNS) :- true |
    PNS =[get(PE)|NewPNS],
    spawn @node(PE),
    fork(N-1,NewPNS).
```

本体部分のソースコードを変えずに、呼び出すサーバを入れ替えることによって、いろいろな負荷分散を実現することが容易なのはこの方式の長所である。この方式でユーティリティ化したものとして、多重レベル負荷バランサ [1] がある。

プロセッサ番号サーバ方式は、親ゴールが子ゴールの行き先を指定するという形の負荷分散には有効だが、そのような枠組では自然に実現できないような負荷分散方式には適用できないのが難点である。（例えば、各プロセッサに常駐しているプロセスの負荷をバランスさせるように、データを動的に分散したい、など）

2.3 テンプレート方式

これまで、プログラムから負荷分散部分を取り出すことを考えてきたが、逆に問題記述部分を取り出すことが考えられる。すなわち、並列プログラムのテンプレートをライブラリで用意し、ユーザは問題記述部分をテンプレートの穴に埋めるという方式である。今回の負荷分散ライブラリではこの方式を採用した。

先ほどの巡回割付けの例では、

---

*Development of An Experimental Load Distribution Library for KL1
Nobuyuki ICHIYOSHI, Mitsubishi Research Institute, Inc.

[1] 「プロセッサ」と呼んでいるが、プロセッシング・ノードは密結合クラスタであってもよい。

```
fork(N,PNS) :- true |
    PNS =[get(PE)|NewPNS],
    $:spawn @node(PE),
    fork(N-1,NewPNS).
```

というテンプレートをライブラリ側が用意し（$: は不定の外部モジュール呼び出しを示す）、一方、ユーザは spawn 述語のみを記述してライブラリと結合する。テンプレート方式では、問題依存部分は局所的な処理であり、ユーザプログラムには負荷分散プラグマは現れない。[2]

テンプレート方式の短所は、負荷分散方式を選ぶと問題解法の大枠が決まってしまい、柔軟性に欠けることである。極端な場合、問題解法毎に負荷分散方式を用意する必要があろう。

3 負荷分散ライブラリの内容

現在試作中のライブラリには、以下のような負荷分散方式が含まれる予定である。

ネットワーク生成

- 完全結合ネットワーク, K-ary N-cube ネットワーク

  それぞれの論理トポロジーを持つネットワークの生成。ユーザは、ノードプロセス、論理ノードのプロセッサへのマッピング関数などを記述する。

静的負荷割付け

- プロセスマッパー

  プロセス種類と定義、プロセス間の結合、プロセスのプロセッサへのマッピング、などをユーザが与えると、それに従ったプロセスネットワークを実現する。

- 分散ハッシュ表

  部分ハッシュ表に分割することによってハッシュ表を並列化したもの。

動的負荷割付け

- スタック分割方式負荷バランサ

  暇になったプロセッサが他のプロセッサから、スタックに積まれた部分問題を分け与えてもらう形式の負荷バランサ [3]。問題を部分問題に分割する展開述語、解に達したかどうかを判定する述語などをユーザが定義する。

ヒューリスティック探索

- 並列深さ優先分枝限定法
- 並列 A*

  これらでは、状態の展開述語などの他にヒューリスティック関数もユーザが定義する。

3.1 負荷分散ライブラリの作り

ネットワーク生成ユーティリティやプロセスマッパーは、非常に一般的なツールであり、複雑な負荷分散方式を試作する時のインフラとして用いることができる。実際、負荷分散ライブラリ自身、ネットワーク生成を最下位として、下位の負荷分散方式の上に、より特化された上位の負荷分散方式を積み上げる形に構成されている（図 1）。

[2]場合によっては、問題記述部分は逐次言語で書くことも考えられる。それがプロセッシング・ノード内で自動並列化されることはあり得る。

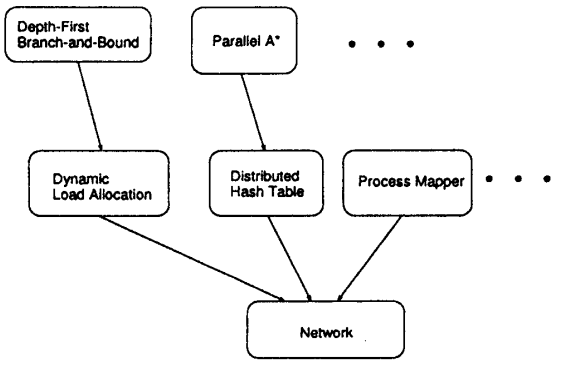

図 1: 負荷分散ライブラリの構造

問題記述部分との結合方式は、目的に応じて静的／動的リンクを選べるように、リンク制御ファイルの指定によって希望の方式の実行ファイルを生成できるようにしている。[3]

呼び出しインタフェースは、一般に、

$$LDU{:}\texttt{create}(PSC,Param,C,R)$$

の形をしている。ここで、LDUは負荷分散ユーティリティ名、PSCは問題依存コードのベクタ（動的結合の場合。静的結合の場合は不要）、$Param$はユーティリティへのパラメータ、Cは制御用ストリーム（主に、制御情報のブロードキャストに用いる）、Rは監視用ストリーム（解収集にも用いる）である。例えば、

```
network:create({Node,Split,Combine,Map},cube(K,N),C,R)
```

4 おわりに

試作した負荷分散ライブラリは今年度末に公開する予定である。本研究は第五世代コンピュータ・プロジェクトの一環として行なわれた。支援および助言を与えて下さった内田俊一研究部長、新田克己第２研究室長、近山隆第１研究室長、ならびに負荷バランサの開発に貢献された三菱電機・古市昌一氏に感謝いたします。

参考文献

[1] M. Furuichi, K. Taki, and N. Ichiyoshi. A multi-level load balancing scheme for or-parallel exhaustive search programs on the Multi-PSI. In *Proceedings of PPoPP'90*, pp. 50–59, 1990.

[2] K. Ueda and T. Chikayama. Design of the kernel language for the parallel inference machine. *The Computer Journal*, 33(6):494–500, 1990.

[3] 古市昌一, 中島克人, 中島 浩, 市吉伸行. スタック分割動的負荷分散方式とマルチ PSI 上での評価. 電子情報通信学会コンピュータシステム研究会, 1991/7.

[3]現在の KL1 システムでは、外部モジュール呼び出しを不定にしたコンパイルができないので、「リンク制御ファイル」で不定の外部呼び出しの解決を記述し、そのファイルからライブラリ・ソースコードをインクルードするようにしている。このため、静的結合のためにはユーザがライブラリ・ソースにアクセスできる必要がある。