TR-I-0319

用例主導型機械翻訳における負荷分散による並列検索手法の検討

"A Study on the Parallel Implementation by Data Distribution in Example-based Machine Translation"

Takashi Okada

1993.3

概要

用例主導型機械翻訳では、大量の用例データからの用例検索処理の高速化が課題である。そこで今回、負荷分散による用例の並列検索の検討とその実現を行った。本項では、並列化手法を、動的/静的負荷分散、及び、並列機構を考慮したデータ分配型/プロセス分配型負荷分散(筆者の提案する区分)に分けて検討した。結果として、疎結合型マシンでプロセッサ数倍、密結合型マシンで約4倍の用例検索処理の高速化を実現した。

ATR自動翻訳電話研究所

ATR Interpreting Telephony Research Laboratories

１．はじめに

文献[１]の用例主導型機械翻訳（ＥＢＭＴ）では、入力文と類似な用例（原言語表現とその訳の対）を検索し、それを利用して翻訳を行う。この翻訳過程において、全用例について用例単位に入力文との属性距離計算を実行する類似検索処理が、翻訳時間上のボトルネックになっている。したがって、この類似検索処理の高速化が問題となり、その解決手段として、以下の二通りが考えられる。

（アプローチ１）並列計算機上での負荷分散による並列化（全件検索）

（アプローチ２）検索対象の動的絞り込み手法による検索（部分検索）

そこで今回、「アプローチ１」を選択し、ＭＩＭＤ型並列計算機上で、負荷分散による並列化手法の検討・実験をし、類似検索処理の高速化を実現した。以下で、その成果報告をする。

２．　用例主導型機械翻訳システムと類似検索処理の概要

２．１　用例主導型機械翻訳（ＥＢＭＴ）システム

今回の類似検索処理の高速化の実験で使用するＥＢＭＴシステムは、「名詞句”Ｎ１のＮ２”」の翻訳システムであり、全体の処理フローを図１に示す。このＥＢＭＴシステムは、三つの処理過程（解析・用例検索・変換）からなり、用例検索では千～十万件の用例からなる用例データベースと３千件程度の類語からなるシソーラス・データベースを使用する。用例は、「国際会議の登録」に関する対話の日英対訳データベースから抽出したもので、シソーラスは、大野，浜西の体系に準拠していて、その階層は三段階である。

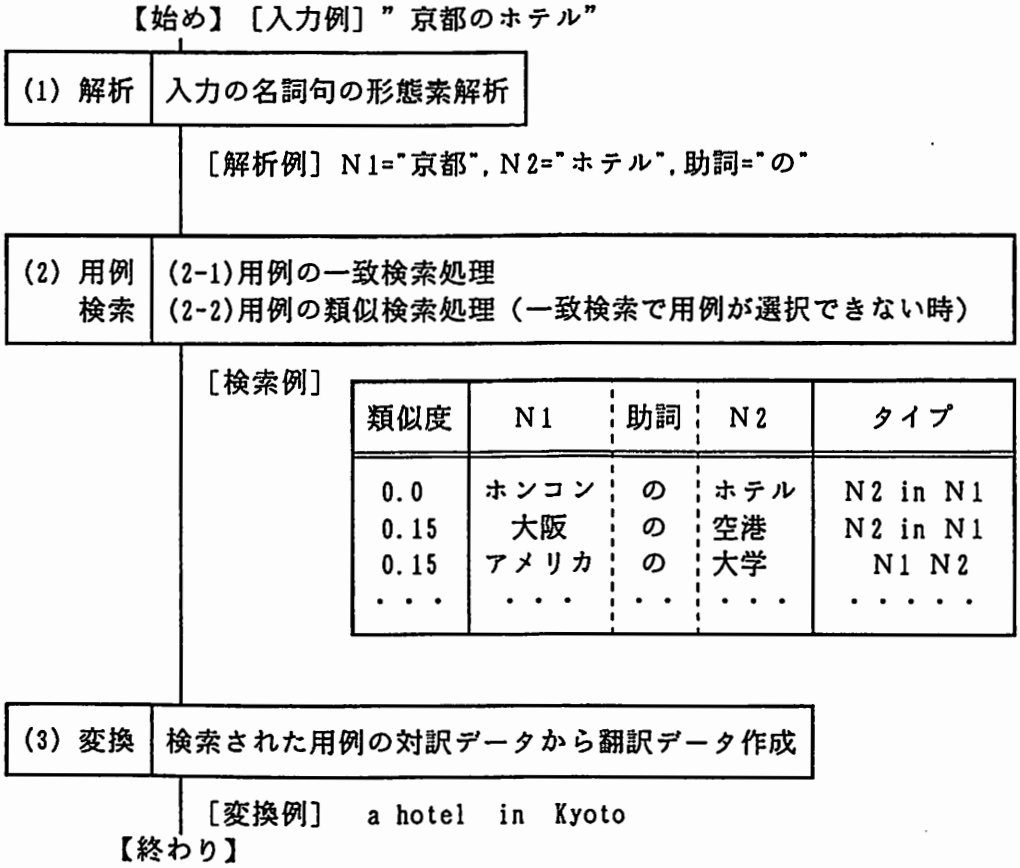


図１．ＥＢＭＴシステムでの「名詞句”Ｎ１のＮ２”」の翻訳処理過程

2．2　類似検索処理の概略

類似検索処理は、用例検索部において一致検索処理で用例が選択できない場合に実行される処理で、用例と入力文とで属性距離計算を行い、それによって算出された類似度を比較して類似用例を選択する処理である。この処理の検索キーワードは類似度であり、それは入力の”N1のN2”と各用例間との属性距離計算で決まるため、全用例の類似度の算出を翻訳のたびに実行する必要がある。1回の属性距離計算にかかる処理時間は小さいが、用例数回（千～十万のオーダー）実行するため、類似検索処理全体に大きな処理時間を費やす。それゆえ、類似検索処理がＥＢＭＴシステムの処理時間上のボトル・ネックになっている。

次に、類似検索処理の処理内容（図2）、及び、属性距離計算の方法と処理実行時の特徴を、具体的に説明する。

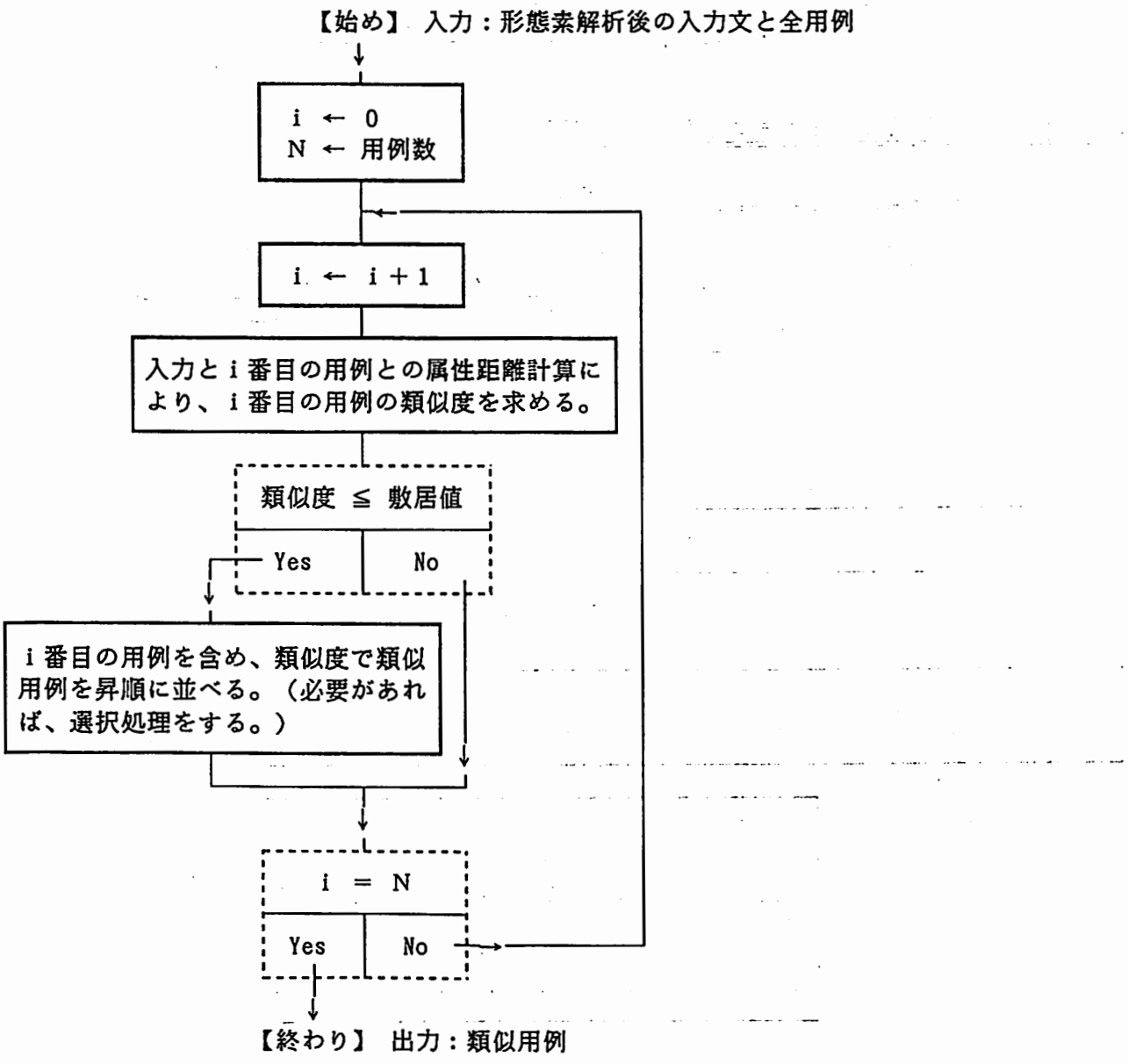


図2．類似検索処理の処理フロー図

【属性距離計算】

属性として、「品詞」・「接辞」・「助詞」・「類語」を用い、「品詞」・「接辞」・「助詞」の距離は、等しければ距離を0（ｒｉ＝０）、異なっていれば距離を１（ｒｉ＝１）とする。「類語」は、角川類語辞典を使用し、そのシソーラスは基本的に三段階のクラスから構成されている。（基本的に、類語コードは三桁の数値）よって、「類語」の距離は、（１ － 一致する階層数／３）で求める。そして、各属性距離を加算する時、各属性距離には予め決められた重み（ｗｉ）が付加され、その合計を類似度とする。

また、重み（ｗｉ）は、属性値の持つ翻訳時の変換の相違におけるばらつきを考慮したもので、この値は、用例データのロード時に全て計算され、類似検索処理の実行前に参照される。

$$\text{類似度} = \sum_{i} wi \times ri$$

注）ｗｉ：ｉ番目の属性の重み　ｒｉ：ｉ番目の属性の距離（「類語」のみ 0,1/3,2/3,1、その他 0,1）

$$wi = \sqrt{\sum_{j} (kj/n)^2}$$

注）ｎ：該当する用例数　ｋｊ：変換後のものが共通な用例数

【処理上の特徴】

(1) １用例当たりの類似検索処理時間が小さい。
（Intel社「ipsc/2 」のLisp環境上で、平均 ４．１ × １０$^{-2}$ sec ）
（Sequent社「Symmetry S2000 」のLisp環境上で、平均 １．７ × １０$^{-3}$ sec ）

(2) 用例のデータ量があらかじめ決まっていて、そのデータ量が十分大きい。

(3) 処理において、他の用例の類似検索処理と同期関係がなく、用例が個々独立に処理される。

類似検索処理の処理上の特徴から、類似検索処理の高速化を目的として、負荷分散による並列処理を検討するのが適当であることが分かる。マルチプロセッサ・システムは、疎結合型と密結合型の２種類に分類できる。疎結合型では、Ｉｎｔｅｌ社「ipsc/2」を、密結合型では、Ｓｅｑｕｅｎｔ社「Symmetry S2000 」を使用して、負荷分散による並列化を考えていく。

３．　ＭＩＭＤ型並列計算機上での負荷分散のためのプログラム並列化手法

まず最初に、マルチプロセッサ・システム上での負荷分散によるプログラム並列化のための方法論が、理論的にカテゴリー化されていないため、疎結合型と密結合型の並列化機構を考慮して、負荷分散によるプログラム並列化を分類し、典型的ないくつかの方法論にまとめてみる。

３．１　ＭＩＭＤ型並列計算機の特徴

疎結合型では、Ｉｎｔｅｌ社「ipsc/2」を、密結合型では、Ｓｅｑｕｅｎｔ社「Symmetry S2000」を参考にして、マルチプロセッサ・システムの主な特徴を説明する。それらのマルチプロセッサ・システムの概略図を、図３に示す。

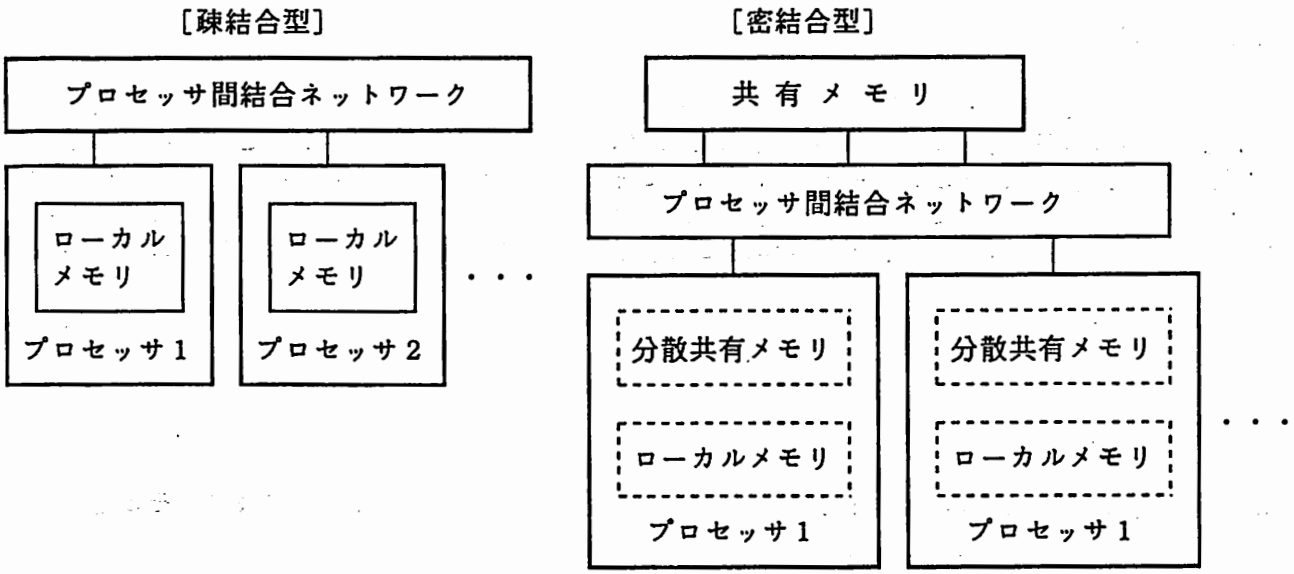


図3．マルチプロセッサ・システムの概略図

疎結合型：分散メモリ・メッセージ交換方式

【特徴】

- プロセッサ単位に、ローカルメモリを持つプロセスが静的に割り当てられている。
- プロセス間の同期処理（グローバル・デッドロックの危険が伴う）・共有データの取り扱いは、メッセージ交換方式で行うため、比較的大きなオーバーヘッドがともない、且つ、他プロセスの能動的な制御を実行するのが難しい。

密結合型：共有メモリ方式

【特徴】

- 共有変数機能（共有メモリ上の変数）とそれに対する排他制御機能（lock･unlock）がある。
- 高々数十程度のプロセッサによるマルチプロセッサ機構とプロセス制御機能（プロセス作成・起動・中止・続行・終了）がある。
- アクセス時間は、共有変数よりもローカル変数の方が小さい。

3．2．　負荷分散のためのプログラム並列化手法の分類

一般に、プログラムの並列化には、次の二種類がある。

- 負荷分散（データ分散）：データを分割して複数のプロセッサに割り振り、並列実行する方法
- 機能分散（コード分散）：コードを分割して複数のプロセッサに割り振り、並列実行する方法

以下では、「負荷分散（データ分散）」のみを具体的に検討する。負荷分散のためのプログラム並列化をＭＩＭＤ型並列計算機上で実現するため、並列化機構上でのデータとプロセスとの関係から、以下の二つに分類した。（筆者が考案した分類）

- データ分配型　：プロセッサ上のプロセスへデータを分配して並列実行する負荷分散

- プロセス分配型：個々のデータ（または、データの塊）を分割所有するプロセスを作成し、実行可能なプロセッサがそれを手分けして並列実行する負荷分散

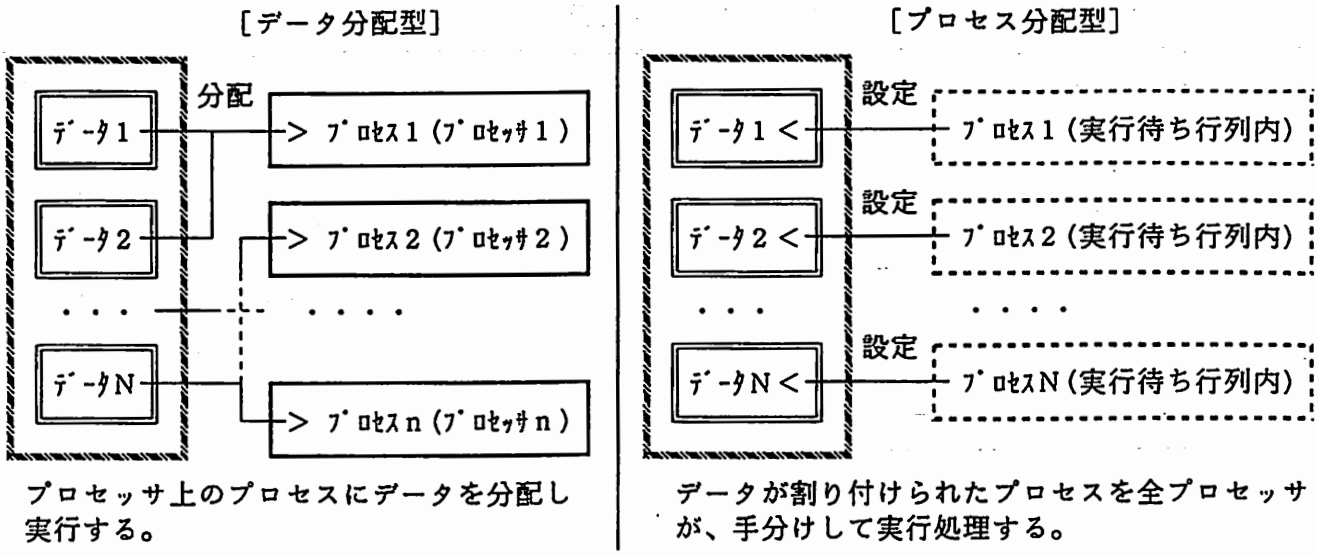


図4．ＭＩＭＤ型並列計算機の並列化機構を考慮した分類（データ分配型とプロセス分配型）

また、通常、負荷分散は、データ処理を分散させるタイミング（実行前・実行中）から、静的負荷分散と動的負荷分散とに分類される。動的負荷分散のデータ分配型は、１個のデータ分配管理用プロセス（ｍａｓｔｅｒプロセス）と１個以上のデータ受信用プロセス（ｗｏｒｋｅｒプロセス）とで構成され、並列実行時に、それらのプロセス間でデータの授受が行われる。このデータ送受信のタイミングの違いから、同期型と非同期型の二種類に分類できる。

図５に、負荷分散のためのプログラム並列化手法における分類をまとめて示す。以下では、それらに対応したプログラム並列化方式を説明する。

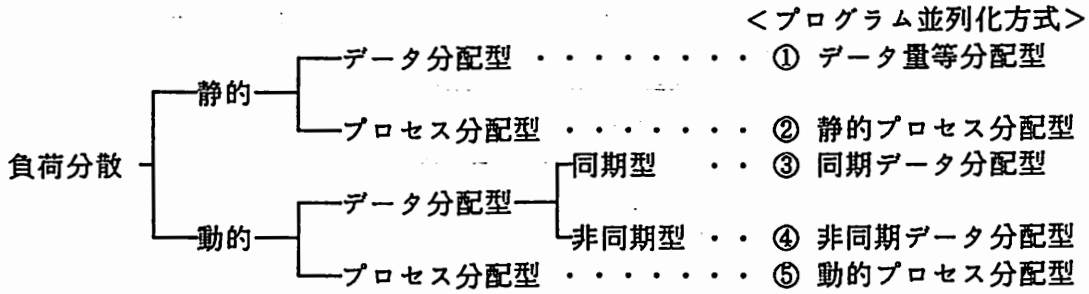
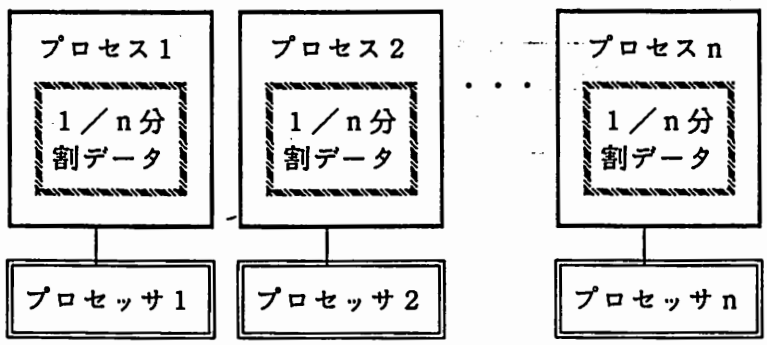


図５．負荷分散のためのプログラム並列化手法における分類

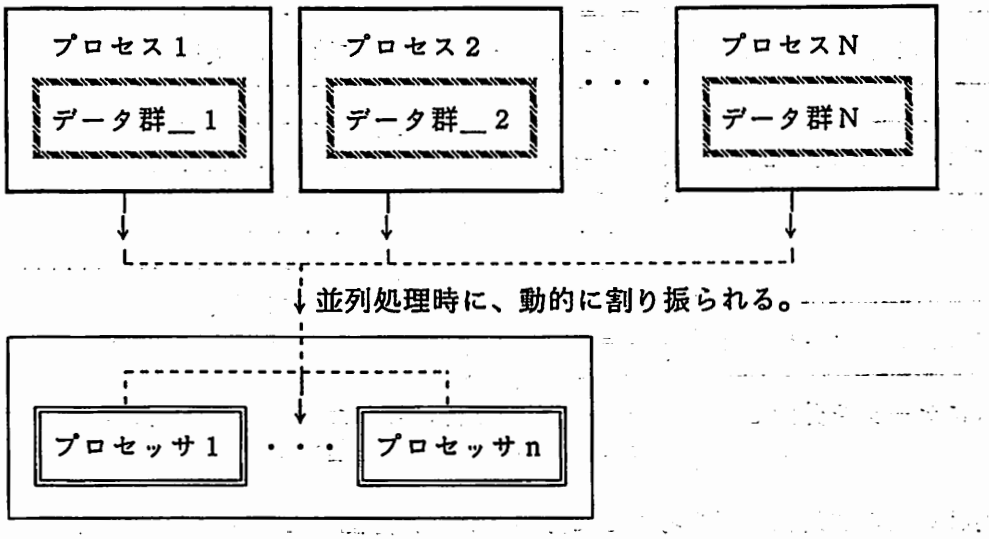
［プログラム並列化方式：　① データ量等分配型　］



説　明

全プロセスの処理時間が等しくなるように、あらかじめプロセッサ数個に等分割したデータを、すべてのプロセッサ上のプロセスが、互いに独立に並列処理する。

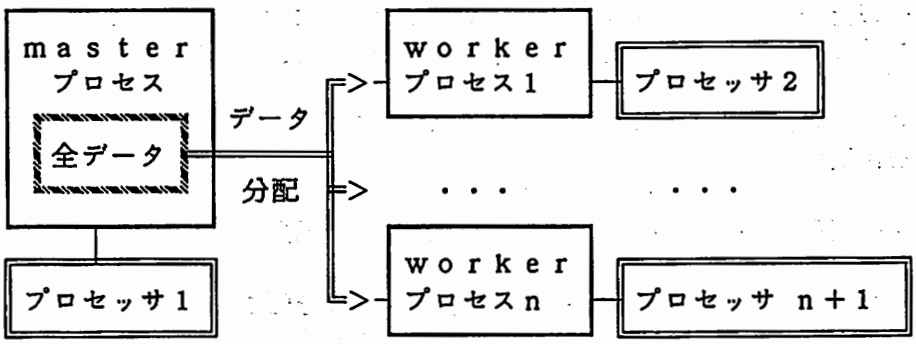
［プログラム並列化方式： ② 静的プロセス分配型 ］



説　明

すべてのプロセッサの処理時間が均等になるように、あらかじめ適当な数にプロセスを分割しておき、並列処理時に、すべてのプロセッサがそれらのプロセスを手分けして処理していく。

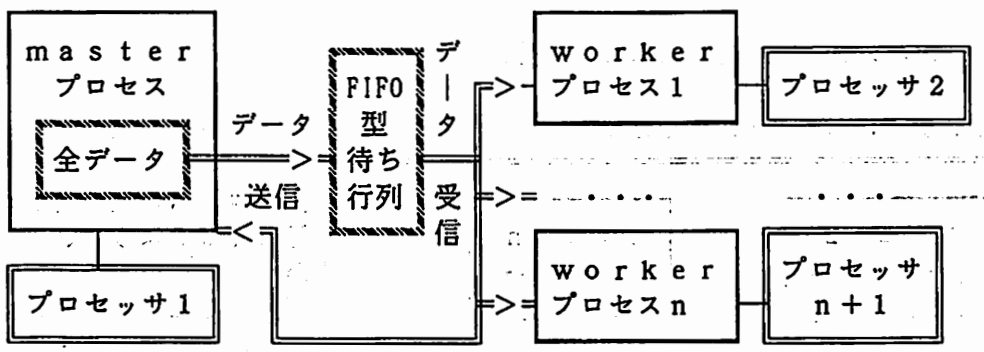
［プログラム並列化方式： ③ 同期データ分配型 ］



説　明

並列処理中に、ｍａｓｔｅｒプロセスが、全ｗｏｒｋｅｒプロセスと直接同期をとってデータを分配し、ｗｏｒｋｅｒプロセスにデータ処理させる。

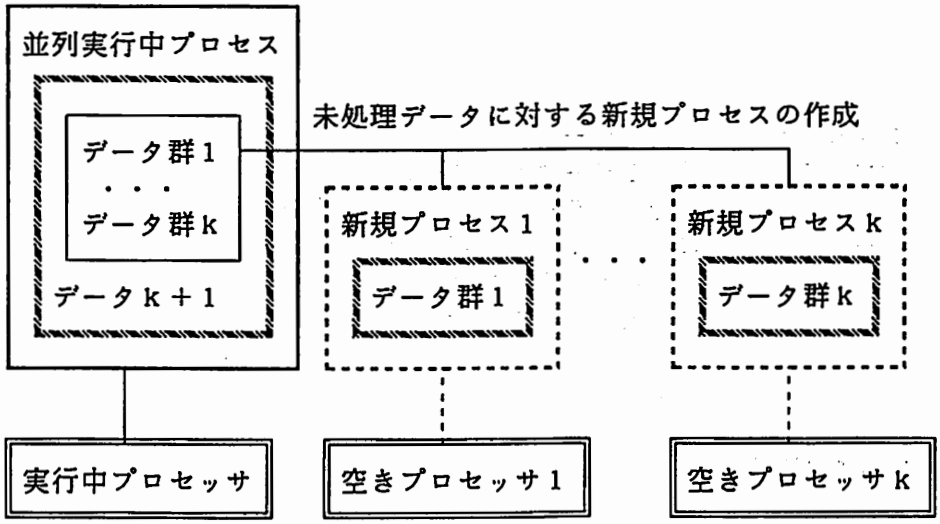
［プログラム並列化方式： ④ 非同期データ分配型 ］



説　明

並列処理中に、ｍａｓｔｅｒプロセスが、データ、または、その塊を媒体（FIFO型待ち行列）に送信し、ｗｏｒｋｅｒプロセスが、それを非同期に受信し、データ処理を実行する。また、ｍａｓｔｅｒプロセスもすべてのデータを送信した後、ｗｏｒｋｅｒプロセスと同じ処理を実行する。

［プログラム並列化方式： ⑤ 動的プロセス分配型 ］



説　明

並列処理中に、プロセッサ上のプロセスが、データ処理の合間に、空きプロセッサを検索する処理を実行する。もし、空きプロセッサを見つけたら、自プロセス担当の未処理データの一部を処理するプロセスを作成する。その新しいプロセスを、空きプロセッサが処理する。

上述したプログラム並列化方式とそのデータ割り振り方法の一覧を、表１に示す。

表１．プログラム並列化方式とそのデータ割り振り方法

	プログラム並列化方式	データ割り振り方法
静的負荷分散	データ量等分配型 (静的データ分配型) ①	プロセッサ数個の等量データを作成し、並列処理の開始直後に、それらを各プロセッサ上のプロセスに割り振る。
	静的プロセス分配型 ②	並列処理開始以前に、データ量とデータ処理内容によるデータ区分に従って、データを分割し、おのおのにプロセスを割りつけておく。
動的負荷分散	同期データ分配型 ③	並列実行中に、ｍａｓｔｅｒプロセスが、すべてのｗｏｒｋｅｒプロセスへのデータ分配を、直接管理する。
	非同期データ分配型 ④	並列実行中に、ｍａｓｔｅｒプロセスがデータ、または、その塊を受信先プロセスの指定をしないでｍａｉｌｂｏｘに送信し、ｗｏｒｋｅｒプロセスが、それを非同期に受信する。また、ｍａｓｔｅｒプロセスもデータ送信後、ｗｏｒｋｅｒプロセスと同一の処理を行う。
	動的プロセス分配型 ⑤	並列実行中に、空きプロセッサを検知したプロセスが、自プロセス担当の未処理データの一部を処理するプロセスを作成する。

次に、プログラム並列化方式の処理時間の概算式を述べる。尚、「⑤動的プロセス分配型」は、並列実行時の空きプロセッサの発生状況やデータ分割・プロセス作成にかかるオーバーヘッドに大きく依存するため、ここでは省略する。

[以下で使用する記号の説明]

D　　：処理データ量
n　　：使用可能な最大プロセッサ数（動的なデータ分配型の場合、workerﾌﾟﾛｾｽ数＝ｎ－１）
Tseq　：コード内の並列化できる処理部分の１データ当りの平均実行時間
Tsynch：コード内の並列化不可能な処理部分の１データ当りの平均実行時間
Tto　：データのプロセスへの割り付けに於ける１データ当りの平均ｵｰﾊﾞｰﾍｯﾄﾞ時間
Tpro　：プロセス制御（作成・起動・終了）に於ける１プロセス当りの平均ｵｰﾊﾞｰﾍｯﾄﾞ時間
T'pro　：プロセス制御（起動・終了のみ）に於ける１プロセス当りの平均ｵｰﾊﾞｰﾍｯﾄﾞ時間

[処理時間の概算式]

・ シーケンシャル版　：　$Tseq \times D$　　$+ Tsynch \times D$

① データ量等分配型　：　$\max_{i=1,n}(Tseq_i) \times D/n + Tpro + Tsynch \times D$

③ 同期データ分配型　：　$Tseq \times D/(n-1) + Tpro + Tsynch \times D$

④ 非同期データ分配型　：　$Tseq \times D/n + Tto \times D/n + Tpro + Tsynch \times D$

② 静的プロセス分配型　：　分割数が小さい（高々、プロセッサ数の数倍程度：ｈ×ｎ）場合、

$$\max_{i=1,n}\left(\sum_{j=1,h} Tseq_{i,j}\right) \times D/(h \times n) + T'pro \times h + Tsynch \times D$$

分割数（Ｋ）が大きい場合、

$$Tseq \times D/n + T'pro \times K/n + Tsynch \times D$$

注）「①データ量等分配型」・「②静的プロセス分配型」の場合、分割されたデータの処理時間がすべて同じである保証がないため、１データ当りの平均処理時間を個別に取り扱う。

注）「③同期データ分配型」の場合、「Ｔto」は、ｍａｓｔｅｒプロセスの各ｗｏｒｋｅｒプロセス用の個別変数への値更新操作のみであるので無視できるものとした。

注）「④非同期データ分配型」の場合、具体的に言えば、［資料－２］の式ｇ１から、$Tseq = t_{ave}$，

$Tto = \dfrac{Tsend + Treceive}{m} + \Delta t$　という置き換えで導くことができる。

4．　「用例検索処理」の高速化実験とその結果

4．1　疎結合型マシン（Intel社「ipsc/2」）

［A］特徴と性能

＜特徴＞

- ローカル変数機能（各要素プロセッサ内の個別メモリ上の変数）
- ハイパーキューブ方式で要素プロセッサを結合したマルチプロセッサ機構
- ＣＰＵは、ｉ８０３８６（Vector Processer）で、そのキャッシュ・メモリのサイズは、８Ｋバイトである。また、各プロセッサ上で、ＮＸ／２というＯＳが走っていて、それらが要素プロセッサ間のメッセージ応答やファイル・アクセス、及び、並列処理用Lisp環境を個々に管理している。

注）要素プロセッサ：ローカルメモリの付随したプロセッサで、一個のプロセスが対応する。

＜性能＞

- プロセスへの関数実行指示の平均時間　　：０．１８ sec
- 非同期のメッセージ交換の平均時間　　：送信側：０．０６ sec 受信側：０．０８ sec
- 同期のメッセージ交換の平均時間　　：送信側：０．０８ sec 受信側：０．０８ sec
- １００００回ｄｏ－ｌｏｏｐ処理の平均時間：５．７２ sec
- 処理結果の回収（join）処理の方法は、［資料－１］より、階段方式による非同期型のメッセージ交換が効果的である。

［B］プログラム並列化方式の実現性

- 実現に支障のないもの

  ① データ量等分配型

- 実現不可能なもの

  ② 静的プロセス分配型　：プロセスがプロセッサ固定で、プロセスの作成が不可能
  ⑤ 動的プロセス分配型　：プロセスがプロセッサ固定で、プロセスの作成が不可能

- 実現できるが、並列効果を望めない。

  ③ 同期データ分配型　　：プロセス間の応答はメッセージ交換のみで、これで同期処理やデータ分配を行うと、大きなオーバーヘッドがかかる。
  ④ 非同期データ分配型　：プロセス間の応答はメッセージ交換のみで、この方式の媒体（FIFO型待ち行列）を使ったデータ送受信処理を実現するのは難しい。

［C］実験結果

Intel社「ipsc/2」の「ipsc/2 lisp Version 1.3」(Common LISP + IPSC/2 Concurrent Constructs)という並列処理用Lisp環境上で、「①データ量等分配型」のプログラム並列化方式を実現し、用例検索処理の負荷分散による高速化実験を行った。尚、用例データベース内には２５５０件の用例しかなかったため、その以上の用例数を必要とする場合、それら２５５０件を複写して使用した。

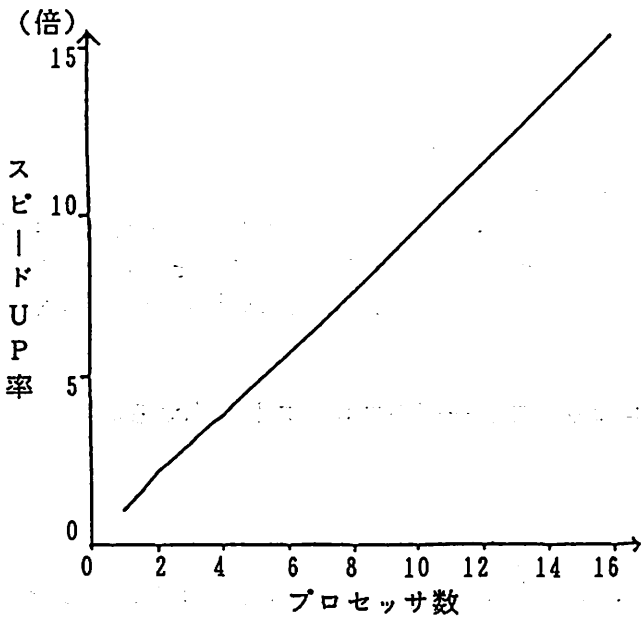


図５．用例データ数＝５０００件での処理
（疎結合型 Intel社「ipsc/2」）

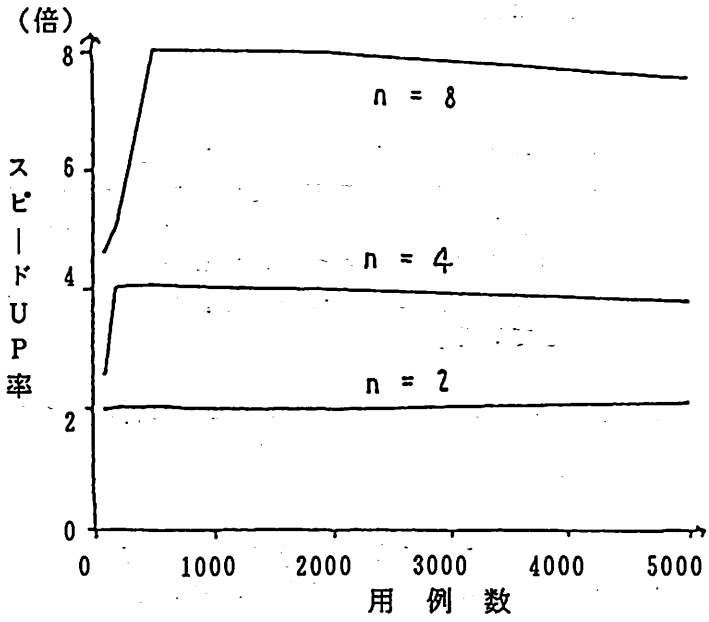


図６．プロセッサ数＝２，４，８での処理
（疎結合型 Intel社「ipsc/2」）

表２．「用例データ数＝５０００件での処理」の処理時間（CPU sec）一覧

プログラム並列化方式	１プロセッサ時	８プロセッサ時
① データ量等分配型	２０６．１２ sec	２７．２１ sec

実験の結果、図５に示すように、用例数＝５０００件の時、プロセッサ数倍に近いスピードＵＰを実現できた。また、図６では、用例数が１０００件以上で、プロセッサ数倍に近いスピードＵＰを実現できることが分かる。これらのことから、次の二点が言える。

（１）「①データ量等分配型」で、且つ、用例数が約１０００件以上の用例検索処理において、並列処理のためのオーバーヘッド時間が、相対的に十分小さい。

（２）「①データ量等分配型」で、且つ、用例数が約１０００件以上の用例検索処理において、すべての並列処理プロセスの処理時間が、近似している。

［まとめ］

１０００件以上の用例に対する用例検索処理において、「①データ量等分配型」で、常に、プロセッサ数倍に近い高速化を実現できた。

4．2　密結合型マシン（Sequent社「Symmetry S2000 」）

［A］特徴と性能

＜特徴＞

- 共有変数機能（共有メモリ上の変数）
- 排他制御機能（ｌｏｃｋ・ｕｎｌｏｃｋ）
- プロセス管理機能
- 高々数十個程度のマルチプロセッサ機構（ＣＰＵは、ｉ８０４８６）で、ＤＹＮＩＸというＯＳがこのマルチプロセッサ機構上で走っていて、この並列処理用Lisp環境を管理している。

＜性能＞

- プロセス起動／終了の平均時間　　：１．５　×１０$^{-2}$ sec
  （１プロセッサ使用時の１０プロセスでの平均時間）
- ｌｏｃｋ／ｕｎｌｏｃｋの平均時間　　：０．６５ × １０$^{-2}$ sec <- sleepy-lock
  ０．１７ × １０$^{-2}$ sec <- spin-lock
- １００００回ｄｏ－ｌｏｏｐ処理の平均時間：１．６　×１０$^{-2}$ sec
- プロセス数、または、プロセッサ数を増加させると、１プロセスあたりのプロセス制御のオーバーヘッド時間が増加する。
- アクセス・スピードは、共有変数よりもローカル変数の方が速い。プロセッサ数が増加するほど、その傾向か増加する。
- １用例あたりの用例検索処理時間が、媒体（FIFO型待ち行列）の排他制御（lock/unlock）にかかる時間より十分小さいので、「④非同期データ分配型」では、［資料－２］で求めたデータ送信単位を使用する。

［B］プログラム並列化方式の実現性

- 実現に支障のないもの

  ① データ量等分配型　，　② 静的プロセス分配型，　③ 同期データ分配型，
  ④ 非同期データ分配型，　⑤ 動的プロセス分配型

［C］実験結果

Sequent社「Symmetry S2000 」の「Allegro Clip」(Version 3.0.3) という並列処理用Lisp環境上で、上述したすべてのプログラム並列化方式を実現し、用例検索処理の負荷分散による高速化実験を行った。また、この実験では、用例はすべて共有メモリ上に置き、プロセスに分配、または、分割されるデータとしては、アドレスやデータ領域指定のインデックスを使用した。

表３．「用例データ数＝１００００件での処理」の処理時間（CPU sec）一覧

プログラム並列化方式	１プロセッサ時	８プロセッサ時
① データ量等分配型	１５．９３ sec	４．４２ sec
② 静的プロセス分配型	１５．３３ sec	１６．６２ sec
③ 同期データ分配型	－	４．４８ sec
④ 非同期データ分配型	－	５．６３ sec
⑤ 動的プロセス分配型	１５．２８ sec	４．８５ sec

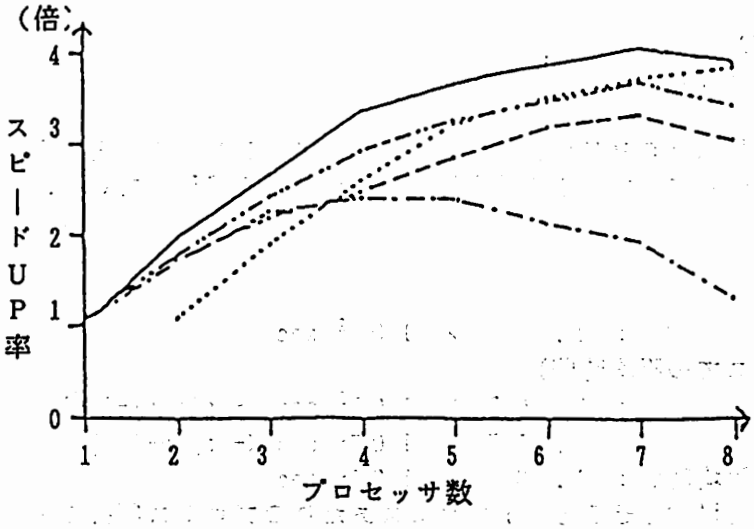


図７．用例データ数＝１００００件での処理
（密結合型 Sequent社「Symmetry S2000」）

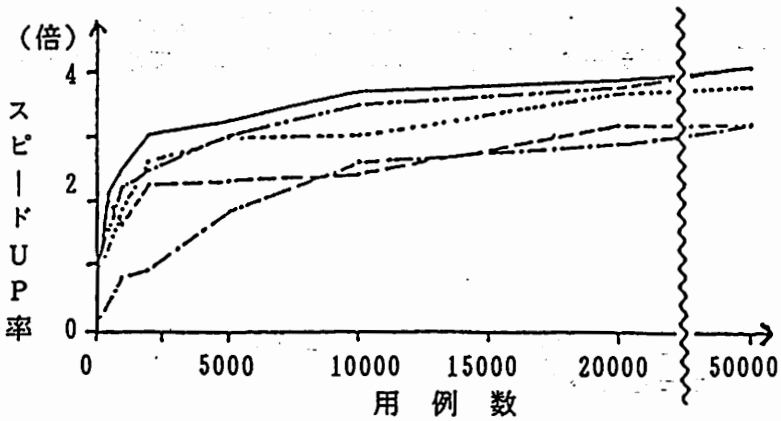


図８．プロセッサ数＝８での処理
（密結合型 Sequent社「Symmetry S2000」）

注）実験での「⑤動的プロセス分配型」のプロセス数は、プロセッサ数×８個とした。

実験の結果、図７・図８に示すように、並列処理時のオーバーヘッドが最も小さい「①データ量等分配型」が、常に、最も高いスピードＵＰ率を示した。また、どのプログラム並列化方式も、並列処理時のオーバーヘッドのため、スピードＵＰ率が頭打ちの状態になった。

［まとめ］

「データ量等分配型」の並列化方式が、常に、最も高いスピードＵＰ率を示し、そのスピードＵＰ率は、高々、４倍程度である。スピードＵＰ率が４倍程度に留まるのは、類似検索処理上での共有メモリ・アクセスに時間がかかっているからである。これは、各プロセスが頻繁に共有メモリをアクセスするために、プロセス間のバス競合によるアクセス渋滞が発生するからである。

5．　類似検索処理の負荷分散による並列化の並列効果についての考察

文献［3］［4］［5］より、負荷分散における並列効果（ここでは、スピードＵＰ）の要因として、次の二つの主要な要因を考える。尚、以下の議論では、並列実行前に負荷分散にかかるオーバーヘッド時間を除外して考える。

［要因－ａ］使用可能なプロセッサのアイドル時間（休止・待ち時間）を最小にする。

［要因－ｂ］負荷分散にかかるオーバーヘッド時間を最小にする。

並列効果（ここでは、スピードＵＰ）の第一義的目的は、［要因－ａ］である。また、［要因－ｂ］について、並列実行時のオーバーヘッドは、静的負荷分散より動的負荷分散の方が大きいことが容易に分かる。これより、静的負荷分散のプログラム並列化方式で［要因－ａ］が満たされる場合、静的負荷分散による方式が最も効果的であると言える。もし、［要因－ａ］が満たされない場合、動的負荷分散のプログラム並列化方式の範囲内で［要因－ｂ］を満たす方式を選択する必要がある。

［要因－ａ］を決定づける主要なものを、いくつか挙げる。

(1) データ処理時間の分散
(2) データ領域内でのデータ処理時間のかたより
(3) データ数
(4) プロセッサ数

［要因－ｂ］を決定づける主要なものを、いくつか挙げる。

(1) 並列処理機構の性能
(2) データ数
(3) プロセッサ数

次に、類似検索処理の場合、「①データ量等分配型」のプログラム並列化方式が最適であるのは、用例のデータ量を等分割しただけで、［要因－ａ］を満足するからである。［要因－ａ］を満足する根拠として、１データあたりの処理時間が小さいだけでなく、以下のものが挙げられる。

［根拠］

データ領域内で、１データ当りの処理時間が、独立で、かつ、予想される処理時間の期待値が等しいため、中心極限定理により、データ数が多いほど、等量のデータ数を処理するプロセスの実行時間のばらつきが、相対的に小さくなる。

注）文献［７］を参考に、データ処理時間のばらつきが正規分布をしている場合のデータ数（ｎ）と全体の処理時間（Ｗ）ののばらつきとの関係を理論的に求める。

中心極限定理より、母集団の分布の形に関係なく、平均μ，分散σ^2 が与えられると、標本 Ｘ１，Ｘ２，・・・，Ｘｎ に基づく標本平均Ｘの標本分布は、正規分布 Ｎ（μ，σ^2/n）となる。
即ち、Ｘは、母平均μを中心として分布し、しかも－これが重要なのであるが－μに集中する傾向がデータ数（ｎ）と共に顕著になっている。
それゆえ、全体のデータ処理時間（Ｗ）は、$n \cdot \overline{X}$であるから、その標本分布は、

正規分布 Ｎ（ Ｗ0，σ_w^2 ）　　$\therefore$ Ｗ0 ＝ ｎ・μ　　$\sigma_w = \sqrt{n} \cdot \sigma$

となる。これより、データ数（ｎ）が増加するに従って、全体の処理時間（Ｗ）の分散（σ_w）は、相対的に小さくなる。

6. まとめ

類似検索処理の負荷分散による並列化の最適な高速化手法は、「データ量等分配型」のプログラム並列化方式である。その内容を、以下で述べる。

【手続き】

並列計算機上で、すべてのプロセッサ対応プロセスに対し、各プロセスのローカル・メモリ上にデータ量を等分割した用例をあらかじめロードしておいて並列実行する静的負荷分散による並列化方式

【並列効果】

疎結合型マシンでプロセッサ数倍、密結合型マシンで約4倍の高速化

7. 今後

今回、検索対象の用例数が処理時間を遅らせる原因である類似検索処理において、負荷分散による並列処理が効果的であることが示された。今後は、本研究で取り扱わなかった検索対象の動的絞り込み手法による検索（部分検索）を検討する必要がある。

参考文献

[1] 隅田英一郎・飯田仁 「用例主導型機械翻訳」，自然言語処理 82－5，1991．3．15

[2] Carriero and Gelern 「How to write Parallel Programs」

[3] 新田淳 他，「大規模ＤＢ／ＤＣの高性能・高信頼化方式の考察」，アドバンストデータベースシステムのシンポジュウム，1885，12月

[4] 笠原博徳 著「並列処理技術」，コロナ社，P41-P45

[5] 益田隆司 他，「データベース及び知識ベースを対象としたストリーム指向型並列処理系の共有メモリ・マシン上への実現」，アドバンストデータベースシステムのシンポジュウム，1885，12月

[6] 牧野都治 著「待ち行列の応用」，森北出版，数学ライブラリー12

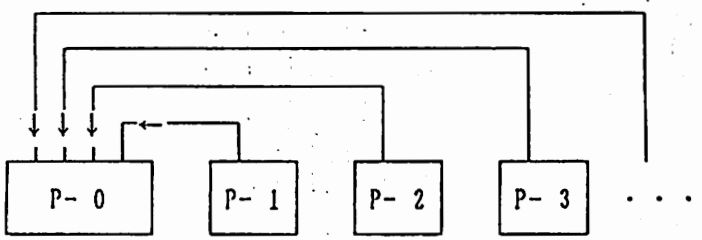
[7] RAMON E. HENKEL 著「統計的検査」，朝倉書店，人間科学の統計学6

資料 — 1

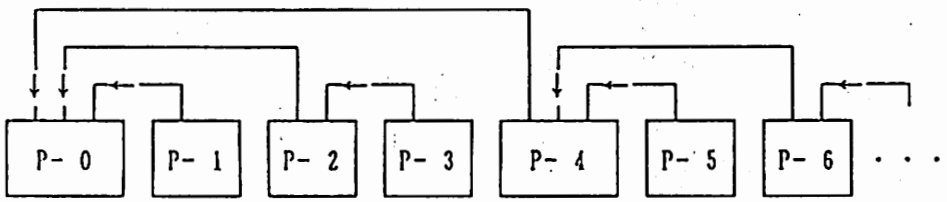
Intel社「ipsc/2」の「ipsc/2 lisp Version 1.3」という並列処理用Lisp環境上における
メッセージ交換による処理結果の回収（join）処理の性能テスト
（1プロセスが、他のすべてのプロセスから1Kバイトの文字列データを収集するメッセージ交換処理）

処理結果の回収のためのメッセージ交換方法として、2種類を設定した。

1）シンプル方式：単純に、1個のプロセス(P- 0)に、他プロセスからのメッセージ送信を集中させる。

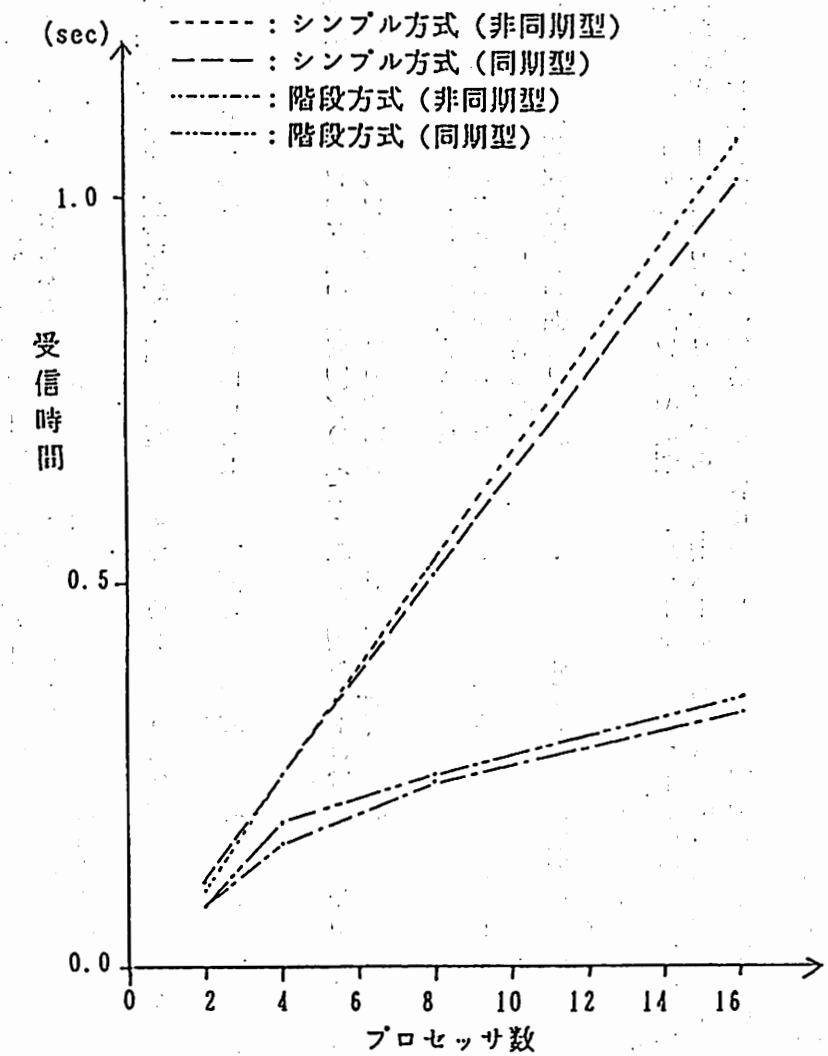


2）階段方式：iを0～［$\log_2 n$］まで、(奇数$*2^i$)番目から(奇数$*2^i-1$)番目へのメッセージ送信を繰り返す。（奇数：1,3,5,・・・）



注）P－n：n番目のプロセス（n：0,1,2,・・・）

○ プロセス（P-0）のメッセージ受信時間



資料 — 2

「④非同期データ分配型」では、すべてのプロセスが競合して、媒体（FIFO型待ち行列）にアクセス（データ送受信）する。媒体とのデータ送受信を効率的に行うため、データを複数個まとめて送受信する方法を取り、そのデータ数を『データ送信単位』として取り扱う。データ送信単位の大小は、以下のような問題を引き起こす。

『データ送信単位』が小さい。　＜－＞　媒体に、空き状態が発生しやすくなり、データ送受信の回数が増加する。

『データ送信単位』が大きい。　＜－＞　ｗｏｒｋｅｒプロセスのデータ処理終了のタイミングにズレが生じ、プロセッサのアイドル時間が増える。

これより、適切なデータ送信単位（ｍ）の値が必要となる。以下で、媒体（FIFO型待ち行列）をｍａｉｌｂｏｘと呼び、適切なデータ送信単位（ｍ）の決定条件を提示して、データ送信単位（ｍ）の算出方法を、理論的に求める。

まず、負荷分散における効果的な並列効果のための二つの要因から、５つの制約条件を提示する。

［要因１］使用可能なプロセッサのアイドル時間（休止・待ち時間）を最小にする。

- 制約条件ａ＞　ｍａｉｌｂｏｘの空状態の回避
- 制約条件ｂ＞　各プロセスのデータ処理終了タイミングのズレが小さいことの保証
  （１回のデータ受信によるデータ処理時間が、データ処理全体からみて十分小さいことの保証）
- 制約条件ｃ＞　全ｗｏｒｋｅｒプロセスが最低１個のデータを処理することの保証

［要因２］負荷分散にかかるオーバーヘッド時間を最小にする。

- 制約条件ｄ＞　ｍａｉｌｂｏｘアクセスに於けるプロセスの平衡状態（使用率＜１）の保証
- 制約条件ｅ＞　データ送信／受信の回数をなるべく少なくする。

これらの制約条件を数式で表現するために、ｍａｉｌｂｏｘを介したｍａｓｔｅｒ／ｗｏｒｋｅｒプロセスによるデータ送受信をモデル化し、次の二つのアプローチを行う。

（Ａ）　プロセスのアクセス頻度から、ｍａｉｌｂｏｘの空状態の回避条件を求める。
（Ｂ）　プロセス間のアクセス競合を考慮したｍａｉｌｂｏｘへのデータ送信時間（受信時間）を求める。

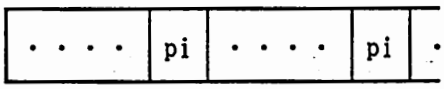
注）以下で使用する記号の説明

Ｄ	：全データ数
ｎ	：プロセッサ数（＝ｗｏｒｋｅｒプロセス数＋１）
ｍ	：送信データ単位（１回の送信で送られるデータ数）
⊿ｔ	：送信データ作成時間（全てのアクセス範囲情報作成の１データ当りの消費時間、又は、データ・リスト作成の１データ当りの消費時間）
Ｔsend	：ｍａｓｔｅｒプロセスからｍａｉｌｂｏｘへのデータ送信時間（Ｔsend ≒ Ｔreceive）
Ｔreceive	：ｗｏｒｋｅｒプロセスのｍａｉｌｂｏｘからのデータ受信時間
Ｔ０	：単一プロセスでのデータ送信／データ受信の平均時間（プロセス競合のない場合）
t_{ave}	：一個のデータの平均処理時間

【 （A） プロセスのアクセス頻度から、ｍａｉｌｂｏｘの空状態の回避条件を求める。 】

ｍａｉｌｂｏｘの空状態の回避を考える場合、ｍａｉｌｂｏｘへのアクセス頻度を問題にする。以下で、アクセス待ち行列を使って、この問題を考えていく。

［アクセス待ち行列］



プロセスｉ（ｐｉ）のアクセス頻度を、

順番：１２３・・・ｋ　(k+1)・・・・ｍ　・・・　　ａｉ ＝ １／アクセス間隔ｉ　・・・　式ｈ１

とすると、

ｍａｓｔｅｒプロセスは、ｍａｉｌｂｏｘへのデータ格納処理を行い、ｗｏｒｋｅｒプロセスは、ｍａｉｌｂｏｘからのデータ搬出処理を行うため、ｍａｉｌｂｏｘの空状態の回避のためには、次の式が成り立つ必要がある。

全ｗｏｒｋｅｒプロセスのアクセス頻度 ＜ ｍａｓｔｅｒプロセスのアクセス頻度

$$(n-1) \times a_{worker} < a_{master}$$

式ｈ１より、

（ｎ－１）×masterﾌﾟﾛｾｽのアクセス間隔 ＜ workerﾌﾟﾛｾｽのアクセス間隔

$$(n-1) \times (\Delta t \times m + Tsend) < m \times t_{ave} + Treceive$$

$$m \times (t_{ave} - (n-1) \times \Delta t) > (n-1) \times Tsend - Treceive \quad \cdots 式h2$$

Ｔｓｅｎｄ ≒ Ｔｒｅｃｅｉｖｅ，ｎ ≧ ２ により、

$$(n-1) \times Tsend - Treceive \fallingdotseq (n-2) \times Tsend \geqq 0 \quad \cdots 式h3$$

【 （B） プロセス間のアクセス競合を考慮したｍａｉｌｂｏｘへのデータ送信時間（受信時間）を求める 】

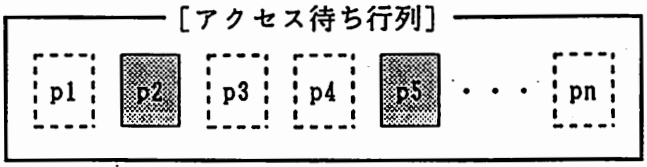
データ送信時間（Ｔｓｅｎｄ）において、ｍａｉｌｂｏｘへのプロセス競合による遅延の問題を、高々プロセッサ数個の待ち行列問題としてとらえ、プロセッサ数個のコインを投げる問題と同等と考えて、この理論標本分布を、二項分布とみなす。

［今回必要とする二項分布の方程式］

０＜ｐ＜１，ｑ＝ｐ－１，Ｎ：試行回数とすると、

$$(p+q)^N = \sum_{i=0}^{N} {}_NC_i p^i q^{N-i} = 1$$

［アクセス待ち行列］



注）ｐｉ：ｉ番目のプロセス □：待ち状態のプロセス

平均値：$\sum_{i=0}^{N} i \cdot {}_NC_i p^i q^{N-i} = N \cdot (p+q)^{N-1} \cdot p = N \cdot p$　　分散：Ｎ・ｐ・ｑ

注） データ受信時間：ここでは、データ送信時間と同じものとして取り扱う。

「④非同期データ分配型」では、ｍａｓｔｅｒプロセスの処理内容の相違（データ分配・データ処理）から、処理状態を二つに分け、データ送信時間（Ｔsend）を考える。

［状態－１］　ｍａｓｔｅｒプロセスのｍａｉｌｂｏｘへのデータ分配処理時で、ｍａｓｔｅｒプロセスは常時、ｍａｉｌｂｏｘをアクセスしているものとして取り扱い、ｗｏｒｋｅｒプロセスと別途に取り扱う。この時のｗｏｒｋｅｒプロセスのｍａｉｌｂｏｘへのアクセス待ち状態は、二項分布をなすものと見なす。

［状態－２］　ｍａｓｔｅｒプロセスのｍａｉｌｂｏｘへのデータ分配処理終了後で、全プロセスがｍａｉｌｂｏｘを均等にアクセスしているものと見なし、全プロセスのｍａｉｌｂｏｘへのアクセス待ち状態は、二項分布をなすものと見なす。

（Ｂ－１）状態－１（ｍａｓｔｅｒプロセスのｍａｉｌｂｏｘへのデータ分配処理時）の場合

ｍａｓｔｅｒプロセスのデータ送信処理は、常時、ｍａｉｌｂｏｘをアクセスしているとみなし、ｍａｉｌｂｏｘへのアクセス確率（ｐ）を１とする。ｗｏｒｋｅｒプロセスのｍａｉｌｂｏｘへのアクセス確率（ｐ）をデータ受信時間／（１回のデータ送信でのデータ処理時間）＝Ｔreceive／（ｍ・t_{ave}）とする。

$$\text{アクセス待ち行列の列平均数（k）} = \sum_{i=0}^{n-1} (i+1) \cdot {}_{n-1}C_i p^i q^{n-1-i}$$

$$= \sum_{i=0}^{n-1} i \cdot {}_{n-1}C_i p^i q^{n-1-i} + \sum_{i=0}^{n-1} {}_{n-1}C_i p^i q^{n-1-i}$$

$$= (n-1) \cdot p + 1$$

$$= (n-1) \frac{Treceive}{m \cdot t_{ave}} + 1$$

よって、Ｔreceive ≒ （ｋ＋１）Ｔ0 より

$$Treceive \fallingdotseq \frac{2 \cdot T0}{1 - \frac{(n-1) \cdot T0}{m \cdot t_{ave}}} \qquad \cdots \text{式h4}$$

（Ｂ－２）状態－２（ｍａｓｔｅｒプロセスのｍａｉｌｂｏｘへのデータ分配後の全プロセスによるデータ処理時）の場合

全プロセスのｍａｉｌｂｏｘへのアクセスは、平衡状態を保ち、アクセス確率（ｐ）をデータ受信時間／（１回のデータ送信によるデータ処理時間）＝Ｔreceive／（ｍ・t_{ave}）とする。

$$\text{アクセス待ち行列の列平均数（k）} = \sum_{i=0}^{n} i \cdot {}_nC_i p^i q^{n-i}$$

$$= n \cdot p$$

$$= \frac{n \cdot Treceive}{m \cdot t_{ave}}$$

よって、Treceive ≒ （k＋1）T0 より

$$Treceive \fallingdotseq \frac{T0}{1-\frac{n \cdot T0}{m \cdot t_{ave}}} \qquad \cdots \text{式h5}$$

以下で、上述の二つのアプローチの結果（式ｈ１～式ｈ５）から、制約条件ａ～ｅを式に展開する。

・ 制約条件ａ＞ ｍａｉｌｂｏｘの空状態の回避

式ｈ２・式ｈ３により、プロセッサ数（ｎ）と送信データ単位（ｍ）が次のような制約を受ける。

◆ プロセッサ数（ｎ）の上限値に対する制約

状態－１に於て、一個のデータについて、平均処理時間（t_{ave}）が、ｗｏｒｋｅｒプロセス数分の送信データを作成する時間（（ｎ－１）×⊿ｔ）よりも大きくなければ成らない。

$$t_{ave} > (n-1) \times \Delta t \qquad \cdots \text{式f1}$$

◆ 送信データ単位（ｍ）の下限値に対する制約

状態－１に於て、ｍａｓｔｅｒプロセスのｗｏｒｋｅｒプロセス数個のデータ送信にかかる全体の処理時間よりも、ｗｏｒｋｅｒプロセスのｍａｉｌｂｏｘアクセス間隔（１送信データ単位のデータ受信インターバル）の方が大きくなければならない。

$$m > \frac{(n-2) \times Tsend}{t_{ave} - (n-1) \times \Delta t}$$

式ｈ４，Tsend≒Treceive より、

$$m > \frac{(n-2) \times 2 \times T0}{t_{ave} - (n-1) \times \Delta t} + (n-1) \times \frac{T0}{t_{ave}} \qquad \cdots \text{式f2-1}$$

・ 制約条件ｂ＞ 各プロセスのデータ処理終了タイミングのズレが小さいことの保証
（１回のデータ受信によるデータ処理時間が、データ処理全体からみて十分小さいことの保証）

この制約条件を、「最大のズレの時間が、並列処理時間の高々１％程度であればよい」という条件に置き換えて考えると、送信データ単位（ｍ）が次のような制約を受ける。

◆ 送信データ単位（ｍ）に対する制約

プロセス内のデータ処理終了時間のズレの最大（Ｍａｘ）は、１回のデータ受信のよる処理にかかる処理時間（Ｌａｇ＝ｍ×t_{ave}＋Treceive）である。データ処理全体の時間（Ｔ）は、式ｇ１を使う。すると、そのズレ比率の逆数（Ｌ）は、

$$L = \frac{T}{Lag} \fallingdotseq \frac{D}{n}\left(t_{ave} + \frac{Tsend + Treceive}{m} + \Delta t\right) \times \frac{1}{m \times t_{ave} + Treceive}$$

この場合、Tsend, Treceive は、$m \times t_{ave}$ より十分小さい場合の問題であり、又、Δt は無視できるので、

$$\fallingdotseq \frac{D}{n \times m}$$

となり、この比率（L）の逆数がズレの比率になるから、以下のような条件式になる。

$$\frac{D}{n \times m} > 100 \qquad \cdots \text{式f3}$$

・ 制約条件c＞ 全ｗｏｒｋｅｒプロセスが最低1個のデータを処理することの保証

送信回数が、ｗｏｒｋｅｒプロセス数以上あればよいので、以下のようになる。

$$(n-1) \times m \leqq D \qquad \cdots \text{式f4}$$

・ 制約条件d＞ ｍａｉｌｂｏｘアクセスに於けるプロセスの平衡状態（使用率＜1）の保証

アクセス待ち行列系の平衡状態を維持する（使用率＜1）ための条件として、n≠2の場合、式f2－1を十分条件として利用できるが、特に、状態－2に於て、n＝2の場合（1個のｍａｓｔｅｒプロセスと1個のｗｏｒｋｅｒプロセスの場合）、式h5により、送信データ単位（m）が以下のような制約の追加を受ける。

◆ 送信データ単位（m）の下限値に対する制約（追加）

$$m > \frac{n \times T0}{t_{ave}} \qquad \cdots \text{式f2－2}$$

・ 制約条件e＞ データ送信／受信の回数をなるべく少なくする。

データ受信回数を減らすには、送信データ単位（m）を最大にすればよい。

$$m = MAX(\text{m値の許される範囲}) \qquad \cdots \text{式f5}$$

また、これら6つの制約条件式の内、式f1・式f2－1・式f2－2・式f4を必要条件として扱い、式f3・式f5は、処理のスピードＵＰ率（速度向上率）の安定化のために、おおまかな目安として付加した条件であり、他に単位データ当りの処理時間のばらつきなどを加味して、最適な値を決定付ける必要があるものであり、ここでは、この二式を十分条件として扱う。

したがって、データ送信単位（m）を算出方法は、以下のようになる。

【 データ送信単位（m）の算出方法 】

［入力］：プロセッサ数（n），平均データ処理時間（t_{ave}），データ量（D），
送信データ作成時間（⊿t），１プロセスでのデータ送信／受信時間の平均（T0）

［出力］：データ送信単位（m），変更されたプロセッサ数（n），エラー（実行不可能）の有無

［処理］

・ step1 => プロセッサ数nが、式ｆ１を満たすか？

・・・ 満たさない場合、プロセッサ数nの変更 or エラー終了

・ step2 => 式ｆ２－１・式ｆ２－２を満たす最小の送信データ単位（m）を設定

・ step3 => プロセッサ数（n）・送信データ単位（m）・データ量（D）が、式ｆ４を満たすか？

・・・ 満たさない場合、プロセッサ数nの変更 or エラー終了

・ step4 => プロセッサ数（n）・送信データ単位（m）・データ量（D）が、式ｆ３を満たすか？

・・・ 満たさない場合、警告

・・・ 満たす場合、式ｆ５により、送信データ単位（m）を、D／（１００×n）を超えない最大の整数値に変更する。

データ送信単位（m）の算出において、最も並列効果の高い最適な値を求める為には、このデータ送信単位（m）の算出方法内で、処理データに対して、「平均データ処理時間」・「データ量」以外に、「単位データの処理時間の分散（ばらつき）」・「データ領域に於ける単位データの処理時間の分布状態」などを考慮する必要がある。

最後に、「④非同期データ分配型」の並列処理部分のデータ処理時間の期待値を求める。期待値を求めるにあたって、計算を簡単にするために、次の二点を前提とする。

前提１：データ受信時間（Treceive）・データ送信時間（Tsend）は、ｍａｓｔｅｒプロセスの処理状態に関わらず一定であるとみなす。

前提２：全プロセスの実行時間を全て同じとみなす。（＝Ｔ１）また、全データ数（D）は、送信データ単位（m）で割り切れるものとみなす。

これより、「④非同期データ分配型」の並列処理実行部分の実行時間（Ｔ１）の期待値は、以下の通りになる。

$$
\begin{cases}
\Delta t \times m + Tsend & + (m \times t_{ave} + Treceive) \times k1 = T1 \\
(\Delta t \times m + Tsend) \times 2 & + (m \times t_{ave} + Treceive) \times k2 = T1 \\
\cdots & \cdots \\
(\Delta t \times m + Tsend) \times (n-1) & + (m \times t_{ave} + Treceive) \times kn-1 = T1 \\
(\Delta t \times m + Tsend) \times \dfrac{D}{m} & + (m \times t_{ave} + Treceive) \times kn = T1 \\
\end{cases}
$$

$$k1 + k2 + k3 \quad \cdots\cdots \quad + kn-1 + kn = \frac{D}{m}$$

注）ｋｉ（ｉ＝1～n-1）：各ｗｏｒｋｅｒプロセスのデータ受信回数

注）ｋｎ：ｍａｓｔｅｒプロセス内のｗｏｒｋｅｒプロセス処理でのデータ受信回数

$$T1 = \underbrace{\left(\frac{n-1}{2} + \frac{D}{n \times m}\right)(\Delta t \times m + Tsend)}_{\text{実行開始時の遅れによる時間 \& masterプロセスのデータ分配処理時間}} + \underbrace{\frac{D}{n}\left(t_{ave} + \frac{Treceive}{m}\right)}_{\text{データ処理時間}}$$

データ数が十分大きいので（Ｄ》０）、

$$T1 \fallingdotseq \frac{D}{n}\left(t_{ave} + \frac{Tsend + Treceive}{m} + \Delta t\right) \quad \cdots\cdots \quad 式g1$$

式ｈ５，Ｔsend≒Ｔreceive より、

$$T1 \fallingdotseq \frac{D}{n}\left(t_{ave} + \frac{2 \times T0}{m - \dfrac{n \times T0}{t_{ave}}} + \Delta t\right) \quad \cdots \quad 式g2$$