システムソフトウェアと
オペレーティング・システム 65－8
（1994．7．21）

# 超並列計算機におけるデータ転送の最適化について

國貞 勝弘　　村上和彰

九州大学 大学院総合理工学研究科 情報システム学専攻

〒816 福岡県春日市春日公園 6-1

*E-mail: {kunisada, murakami}@is.kyushu-u.ac.jp*

あるプログラムを並列計算機上で，並列実行する場合には，データの分割・配置，データ転送のためのコード生成・スケジューリング，といったことが必要になる．現在我々は，科学技術計算プログラムを対象にして，「コンパイラ（および，ランタイム・ソフトウェアならびにプログラム自身）によるデータ分割／配置およびデータ転送の最適化」について検討を行なっている．本稿ではそのうち，「コンパイラによる静的なデータ転送の最適化」について検討する．一般的な問題の定義と，今後の研究対象としてのより単純化した問題の定義を行う．

# Optimization of Data Transfers on Massively Parallel Processors

Katsuhiro Kunisada　Kazuaki Murakami

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
Kasuga-shi, Fukuoka 816 Japan

*E-mail: {kunisada, murakami}@is.kyushu-u.ac.jp*

Data decompositions and data transfers are essential to execute a parallel program on a parallel processors. We are investigating into a optimization method of data decompositions and transfers. In this paper, we examine a statically optimization of data transfers by compiler. We present general definition of the problem, and a more simple problem definition as future target.

# 1 はじめに

1個のプログラムを（超）並列計算機上で並列実行する場合，

- **計算**（*computation*）の**分割**（*partitioning, decomposition*），および，分割された計算（一般に，**タスク**（*task*）と呼ぶ）のプロセッサへのスケジューリング（各タスクをどのプロセッサで，どの時点で実行するかを決定すること）
- **データの分割**（*partitioning, decomposition*），および，分割されたデータのメモリへの**配置**（*allocation*）
- **データ転送**（*data transfer, data movement*）のための**コード生成**（*code generation*）およびスケジューリング

といった操作が一般に必要である（図1参照）．

これらの操作は，対象とする（超）並列計算機が，

- **集中共有メモリ**（*centralized shared memory*）／**UMA**（*Uniform Memory Access*）モデル：物理的に一ヶ所にグローバル・メモリとして集中配置されたメモリを，プロセッサが通常のRead/Write操作（LOAD/STORE，等）によりアドレス指定で直接的に読み書きすることで共有する．共有メモリ上の共有変数に対する読み書きという形でデータ授受を行なう．すべてのプロセッサから同一時間で集中共有メモリにアクセス可能である．
- **分散メモリ**（*distributed memory*）モデル：メモリをある単位（たとえば，プロセッサ）毎にローカル・メモリとして分散配置する．
  - **分散共有メモリ**（*distributed shared memory*）／**NUMA**（*NonUniform Memory Access*）モデル：プロセッサが，それ自身のローカル・メモリだけでなく他のプロセッサに配置されたメモリ（リモート・メモリ）までも，通常のRead/Write操作（LOAD/STORE, PUT/GET，等）によりアドレス指定で直接的に読み書きすることで共有する．共有メモリ上の共有変数に対する読み書きという形でデータ授受を行なう．各プロセッサから見た共有メモリの距離はアドレスにより遠近が生じ，アクセス時間も異なる．
  - **メッセージ交換**（*message passing*）**NORA**（*NO Remote Access*）モデル：プロセッサが通常のRead/Write操作によりアドレス指定で直接的に読み書きできるのは，それ自身のローカル・メモリだけで，リモート・メモリに対しては出来ない．すなわち，共有メモリ上の共有変数に対する読み書きという形でデータ授受を行なうのではなく，メッセージの交換という形でデータ授受を行なう．

のいずれでも必要である．

また，これらの操作は，

- プログラム実行前（**静的**）に，
  - **プログラマが**，および／あるいは，
  - コンパイラが

  行なう，および／あるいは，
- プログラム実行時（**動的**）に，
  - **プログラム自身が**，
  - **ランタイム・ソフトウェアが**，および／あるいは，
  - **ハードウェアが**（それ自身の論理で）行なう

ことが選択肢として可能である．

さらに，これらの操作は，相互に密接に関わり合っている．すなわち，ある操作の結果が他の操作に影響を与え，その逆もまたしかりである（「鶏と卵」の関係）．

現在，我々は，科学技術計算プログラムを対象にして，『コンパイラ（および，ランタイム・ソフトウェアならびにプログラム自身）によるデータ分割／配置およびデータ転送の最適化』について検討を行なっている [5]．本稿ではそのうち，『コンパイラによる静的なデータ転送の最適化』について検討する．

まず，2章で，我々が現在検討している『コンパイラによる静的なデータ分割／配置およびデータ転送の最適化』問題を一般化した場合の定義を与え

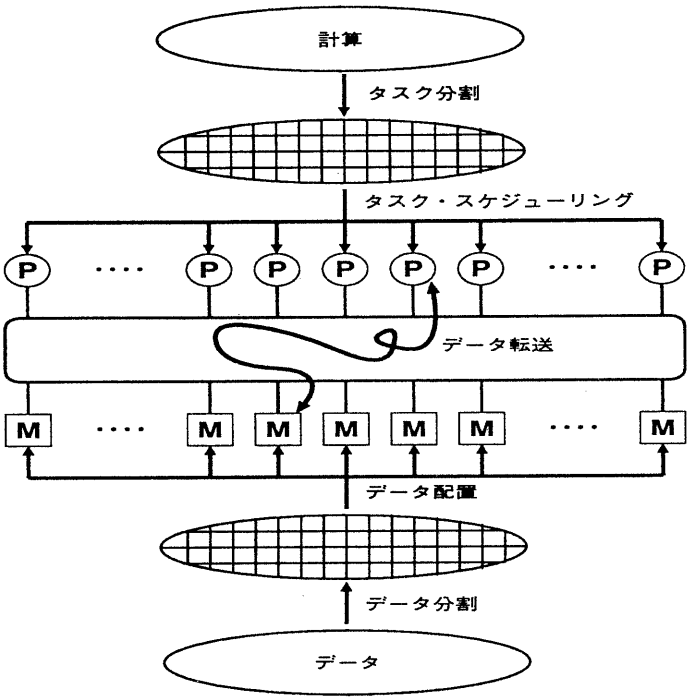


図 1: 並列実行に必要な操作

る．3章で，本稿で取り上げる問題に関連する研究を紹介する．そして，4章で，我々が現在解こうとしている問題を定義する．

## 2 一般化された問題の定義

### 2.1 データ分割／配置の最適化

「コンパイラによる静的なデータ分割／配置の最適化」問題を一般化すると，次のように定義できる．

すなわち，プログラムおよびデータが与えられた時に，プログラム実行時間を最小とするように，データ分割／配置

$$\delta : \mathcal{D} \rightarrow \mathcal{M} \times \mathcal{T}$$

を求めることである．ここで，

- $\mathcal{D}$:データを要素とするデータ空間（*data space*）
- $\mathcal{M}$：メモリ・モジュールを要素とするメモリ空間（*memory space*）
- $\mathcal{T}$：時空間（*time sapece*）

である．

メモリ空間 $\mathcal{M}$ の各要素は必ずしも物理的なメモリ・モジュールに対応している必要はない．物理的なメモリ・モジュールと1対1に対応していない場合は，単に「データ分割」と呼ぶ．

プログラムの実行中，データ配置が一切変わらない場合は，データ分割／配置

$$\delta : \mathcal{D} \rightarrow \mathcal{M}$$

を求めればよい．

### 2.2 データ転送の最適化

「コンパイラによる静的なデータ転送の最適化」問題を一般化すると，次のように定義できる．

すなわち，タスク・スケジュール

$$\rho : \mathcal{C} \rightarrow \mathcal{P} \times \mathcal{T}$$

および，データ分割／配置$\delta$が与えられた時，プログラム実行時間を最小とするように，データ転送

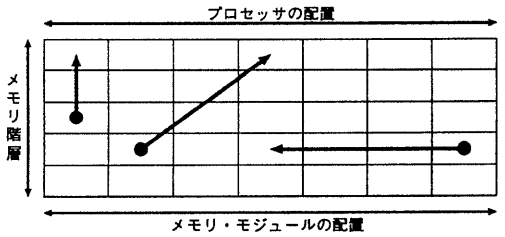


図 2: どこからどこへ?

コードを生成し，スケジュール

$$\sigma : \mathcal{CC} \rightarrow \mathcal{P} \times \mathcal{T}$$

を求めることである．ここで，

- $\mathcal{C}$：計算を要素とする**計算空間**（*computation space*）
- $\mathcal{CC}$：$\mathcal{C}$に生成したデータ転送コードを加えた結果得られる計算空間
- $\mathcal{P}$：プロセッサを要素とする**プロセッサ空間**（*processor space*）

である．

データ転送コードの生成およびスケジューリングに当たっては，一般に以下の変数を決定する必要がある．

- **誰が転送するのか?**：データの生産者，消費者，あるいは，第3者が転送するのか?
- いつ転送するのか?
- どのデータを転送するのか?
- どこからどこへ転送するのか?：水平方向にはプロセッサおよびメモリ・モジュールの配置をとり，垂直方向にはメモリ階層（レジスタ，キャッシュ，メモリ，等）をとるような2次元空間（図2参照）において，どの点からどの点へ転送するのか?
- どのような経路で転送するのか?

## 3 関連研究

1個のプログラムの並列実行を対象にした「コンパイラによる静的なデータ転送の最適化」については，これまであまり研究が行なわれていない．

「データ転送の最適化」に関する研究は，そのほとんどが逐次プログラムを対象にした「ハードウェアによる動的なデータ転送」に関するもので，しかもプリフェッチング（*prefetching*：つまり，データの消費者が当該データの転送を行う）に限られている [10].

対象は逐次プログラムだが,「コンパイラによる静的なプリフェッチングの最適化」に関する研究がいくつかある [11,16,22]. たとえば，Mowry と Monica と Gupta [22] は，配列添字がループ制御変数のアフィン関数であるようなループ・ネストに対して，「どのデータをいつ，メモリからキャッシュにプリフェッチすれば良いか」を決定するアルゴリズムを提案している．プリフェッチすべきデータの決定には再利用性解析（局所性解析の一部）を行い，配列要素に対する個々の参照が**時間再利用性**（*temporal reuse*），**空間再利用性**（*spatial reuse*），および，**グループ再利用性**（*group reuse*）を有しているか否かを判定する．そして，キャッシュミスを起こす可能性の高い参照に対応する配列要素をプリフェッチする．また，その時期は，ソフトウェア・パイプライニングを適用することで当該配列要素が使用されるよりも，$\left\lceil \frac{l}{s} \right\rceil$ イタレーション前（$s$ はループ・ボディ中で最も短いパスの長さ，$l$はプリフェッチ・レイテンシ）としている．

さらに，科学技術計算プログラムの並列実行を対象にした「コンパイラによる静的なプリフェッチングの最適化」についても，少しだが研究が行なわれている [12,13]. たとえば，Gornish と Granston と Veidenbaum [13] は，NUMA モデルを仮定して，個々のループ・ネストに対して「どのデータを（出来るだけ早い時点で）いつ，リモート・メモリからローカル・メモリ（あるいは，そのキャッシュ）にプリフェッチすれば良いか」を決定するアルゴリズムを提案している．プリフェッチすべきデータの決定には再利用性解析を行い，その時期の決定はデータ依存および制御依存に関する制約条件を満足する範囲で最も早い時点としている．

以上のすべての研究が，プリフェッチング，つまりデータの消費者が当該データの転送を行う場合に限られている．プリフェッチングは，*Owner-Computes Rule*（代入式の左辺のデータを所有するプロセッサが，当該代入式の右辺の計算を行なう）と整合性が良い．しかし，データが生産されるよりも前に当該

データをプリフェッチするわけにはいかないので，データ依存関係が存在する場合はプリフェッチの適用範囲が狭くなる．そこで，プリフェッチングとは逆のアイデアであるポストストアリング（*poststoring*），すなわち，データの生産者が当該データを（消費者に）転送するという選択肢も検討の価値があろう．なお，ポストストアリングも，逆依存関係が存在する場合は適用範囲が狭くなるが，逆依存関係は真のデータ依存関係とは異なり変数名前替えで消去することが可能である．

また，並列計算機では逐次計算機よりも，データを「どこからどこへ」転送するかに関する選択肢も広い．逐次計算機の場合，

- プリフェッチング：
  - （ローカル）メモリから：
    - （ローカル）キャッシュへ
    - （ローカル）レジスタへ
  - （ローカル）キャッシュから：
    - （ローカル）レジスタへ

といった選択肢しか存在しなかったが，並列計算機の場合は，

- プリフェッチング：
  - リモート・メモリから：
    - ローカル・メモリへ
    - ローカル・キャッシュへ
    - ローカル・レジスタへ
  - リモート・キャッシュから：
    - ローカル・メモリへ
    - ローカル・キャッシュへ
    - ローカル・レジスタへ
  - リモート・レジスタから：
    - ローカル・メモリへ
    - ローカル・キャッシュへ
    - ローカル・レジスタへ
- ポストストアリング：
  - ローカル・メモリから：
    - リモート・メモリへ
    - リモート・キャッシュへ
    - リモート・レジスタへ
  - ローカル・キャッシュから：
    - リモート・メモリへ
    - リモート・キャッシュへ
    - リモート・レジスタへ
  - ローカル・レジスタから：
    - リモート・メモリへ
    - リモート・キャッシュへ
    - リモート・レジスタへ

等の選択肢が加わる（図 2参照）．したがって，「どこからどこへ」の決定も極めて重要となろう．

さらに，相互結合網のルーティングが固定的でない場合は，コンパイラによるデータ転送の経路選択の最適化も重要となる．

# 4 問題の定義

ここでは，我々が現在検討しているデータ転送問題の決定変数について考察する．

## 4.1 どのデータを転送するか

$D$は，ある場所からある場所へと転送される，もしくは，あるプロセッサから参照される，全部で$k$個の要素を持つプログラム中のデータの集合である．$D = \{d_1, \cdots, d_k\}$である．$P_i(i = 1, \cdots, n)$をあるプロセッサ，$D_{P_i}$を$P_i$がもつデータの集合とすると，

$$D \subseteq D_{all} = D_{P_1} \cup D_{P_2} \cdots \cup D_{P_n}$$

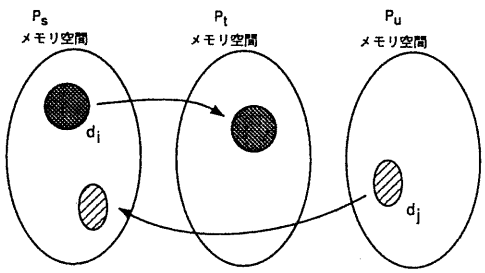


図 3: データの移動

ただし，この$D$を決定するときには，データの依存関係や，転送先のメモリの空き容量の大きさなどの制約条件を満たさなければならない．

転送の対象となるデータは，放っておいても転送されるもの，つまりプログラム中で，明示的に転送(集中共有メモリの場合には，グローバルメモリの参照の場合もある)されると書かれてあるものである．

### 4.2 誰が転送するか

$w$は集合$D$の要素$d_i$を，生産者，または，消費者主導の転送のどちらかであるか指定する．

$$w : D \longrightarrow \{\text{生産者}, \text{消費者}\}$$

### 4.3 いつ転送するか

本稿では，データの転送が行われるのはそのデータが使用される前である，とする．つまり，データのプリフェッチングについてのみ考える．現時点ではポストストアリングについては考慮しない．

$\tau$は，データを移動するプログラムのポイントである．通常$\tau$が示すプログラムポイントにデータ転送命令を挿入することになる．$start$を，プログラムの最初のポイント，$in_use$をもとのプログラムで実際にデータが転送(参照)されるポイントであるとすると，

$$start \leq \tau(d_i) < in\_use$$

となる．これは，データの転送が，プログラムの最初のポイントから，実際にローカルメモリに転送されるよりも前のポイントの間で行われることを示す．データの転送がデータアクセスレイテンシ分だけ前であれば，もっともデータを無駄なくアクセスできることになるが，データがどこにあるかによって，データアクセスレイテンシは，変わってくる．

制約条件として，データの依存関係がある．データの転送を行った後で，そのデータが現実を反映していない(データの値が書き変わっている)ということがあってはならない．したがって，依存関係の影響を受けない時期にデータの転送を行うことが必要になる．

### 4.4 どこからどこに転送するか

各プロセッサには，それぞれ2階層のメモリ階層構造(通常のローカルメモリとキャッシュメモリ)が存在しているとする．ここで，$P$をプロセッサの集合，$H$はメモリ階層の集合とすると，$(P = \{P_1, \cdots, P_n\}, H = \{H_1, H_2\})$

$$r_{from} \in W = P \times H$$

$$r_{to} \in W$$

である(図2参照)．また，キャッシュメモリにある場合の方が当然データアクセスレイテンシは小さいので，できるだけキャッシュメモリにデータを転送する．

## 5 おわりに

本稿では，コンパイラによる静的なデータ転送の最適化についての検討を行った．一般的なデータ転送の最適化の問題の定義を行い，問題を解くためには以下の5つの変数の決定を行う必要があると述べた．つまり，(1)誰がデータを転送するか，(2)いつ転送するか，(3)どのデータを転送するか，(4)どこからどこへ転送するか，(5)どのような経路で転送するか，である．

最後に，今後の研究対象として，より特殊なデータ転送の最適化問題の定義を行った．今後の課題として，ここで定義した問題について，それを解くためのアルゴリズムを求め，さらにその評価を行うことを挙げることができる．

## 謝辞

日頃ご討論頂く，九州大学 大学院総合理工学研究科 安浦寛人 教授，岩井原瑞穂 助手，ならびに，山家 陽 氏をはじめとする安浦研究室の諸氏に感謝します．

本研究は一部，文部省科学研究費補助金 重点領域研究「超並列原理に基づく情報処理体系」による．

## 参考文献

[1] 國貞勝弘，村上和彰，“超並列計算機におけるデータ転送の最適化について，” 情処研報，OS–65–8，1994年7月．

[2] 進藤達也，岩下英俊，土肥実久，萩原純一，金城ショーン，“Twisted Data Layout，” 並列処理シンポジウム *JSPP'94* 論文集，pp.161–168，1994年5月．

[3] 本多弘樹, "自動並列化コンパイラ," 情報処理, vol.34, no.9, pp.1150–1157, 1993年9月.

[4] 村上和彰, 山家 陽, "高性能プラットフォーム要素マイクロプロセッサ・アーキテクチャ — 方式検討 —," 情処研報, ARC–101–20, 1993年8月.

[5] 山家 陽, 村上和彰, "超並列計算機におけるデータ分割／配置の最適化について," 情処研報, OS–65–7, 1994年7月.

[6] 吉田明正, 前田誠司, 尾形 航, 笠原博徳, "マルチグレイン並列処理におけるデータローカライゼーション手法," 並列処理シンポジウム *JSPP'94* 論文集, pp.145–152, 1994年5月.

[7] Amarasinghe, S. P. and Lam, M. S., "Communication Optimization and Code Generation for Distributed Memory Machines," *Proc. 1993 Conf. on Programming Language Design and Implementation*, pp.126–138, Jun. 1993.

[8] Anderson, J. M. and Lam, M. S., "Global Optimization for Parallelism and Locality on Scalable Parallel Machines," *Proc. 1993 Conf. on Programming Language Design and Implementation*, pp.112–125, Jun. 1993.

[9] Balasundaram, V., Fox, G., Kennedy, K., and Kremer, U., "A Static Performance Estimator to Guide Data Partitioning Decisions," *Proc. 3rd Symp. on Principles & Practice of Parallel Programming*, pp.213–223, Apr. 1991.

[10] Baer, J.-L. and Chen, T.-F., "An Effective On-Chip Preloading Scheme To Reduce Data Access Penalty," *Proc. Supercomputing'91*, pp.176–186, Nov. 1991.

[11] Callahan, D., Kennedy, K., and Porterfield, A., "Software Prefetching," *Proc. 4th Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.40–52, Apr. 1991.

[12] Gallivan, K., Jalby, W., and Gannon, D., "On the Problem of Optimizing Data Transfers for Complex Memory Systems," *Proc. 1988 Int. Conf. on Supercomputing*, pp.238–253, Jul. 1988.

[13] Gornish, E. H., Granston, E. D., and Veidenbaum, A. V., "Compiler-directed Data Prefetching in Multiprocessors with Memory Hierarchies," *Proc. 1990 Int. Conf. on Supercomputing*, pp.354–368, Jun. 1990.

[14] Gupta, M. and Banerjee, P., "Demonstration of Automatic Data Partitioning Techniques for Parallelizing Compilers on Multicomputers," *IEEE Trans. on Parallel and Distributed Systems*, vol.3, no.2, pp.179–193, Mar. 1992.

[15] Hudak, D. E. and Abraham S. G., "Compiler Techniques fot Data Partitioning of Sequentially Iterated Parallel Loops," *Proc. 1990 Int. Conf. on Supercomputing*, pp.187–200, Jun. 1990.

[16] Klaiber, A. C. and Levy, H. M., "An Architecture for Software-Controlled Data Prefetching," *Proc. 18th Ann. Int. Symp. on Computer Architecture*, pp.43–53, May 1991.

[17] Knobe, K., Lukas, J. D., and Dally, W. J., "Dynamic Alignment on Distributed Memory Systems," *Proc. 3rd Workshop on Compilers for Parallel Computers*, pp.394–404, Jul. 1992.

[18] Lee, R. L., Yew, P.-C., and Lawrie, D. H., "Data Prefetching in Shared Memory Multiprocessors," *Proc. 1987 Int. Conf. on Parallel Processing*, pp.28–31, Aug. 1987.

[19] Li, J. and Chen, M., "Index Domain Alignment: Minimizing Cost of Cross-Referencing Between Distributed Arrays," *Proc. Frontiers'90: The 3rd Symp. on the Frontiers of Massively Parallel Computation*, pp.424–433, Oct. 1990.

[20] Li, J. and Chen, M., "Compiling Communication-Efficient Programs for Massively Parallel Machines," *IEEE Trans.*

*on Parallel and Distributed Systems*, vol.2, no.3, pp.361–376, Jul. 1991.

[21] Mace, M., *Memory Storage Patterns in Parallel Processing*, Kluwer Academic Publishers, 1987.

[22] Mowry, T. C., Lam, M. S., and Gupta, A., "Design and Evaluation of a Compiler Algorithm for Prefetching," *Proc. 5th Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.62–73, Oct. 1992.

[23] Ramanujam, J. and Sadayappan, P., "Compile-Time Techniques for Data Distribution in Distributed Memory Machines," *IEEE Trans. on Parallel and Distributed Systems*, vol.2, no.4, pp.472–482, Oct. 1991.

[24] Rogers, A. and Li, K., "Software Support for Speculative Loads," *Proc. 5th Conf. on Architectural Support for Programming Languages and Operating Systems*, pp.38–50, Oct. 1992.

[25] Tullsen, D. M and Eggers, S. J., "Limitations of Cache Prefetching on a Bus-Based Multiprocessor," *Proc. 20th Ann. Int. Symp. on Computer Architecture*, pp.278–288, May 1993.