行政院國家科學委員會專題研究計畫　成果報告

平行編譯器中不規則資料重新分佈之訊息產生與通訊最佳化研究(2/2)

計畫類別：個別型計畫
計畫編號：NSC94-2213-E-216-002-
執行期間：94 年 08 月 01 日至 95 年 07 月 31 日
執行單位：中華大學資訊工程學系

計畫主持人：許慶賢

計畫參與人員：蔡秉儒、翁銘遠、藍朝陽

報告類型：完整報告
報告附件：出席國際會議研究心得報告及發表論文
處理方式：本計畫可公開查詢

中　華　民　國 95 年 10 月 24 日

行政院國家科學委員會補助專題研究計畫 ■ **成 果 報 告**
□ **期中進度報告**

平行編譯器中不規則資料重新分佈之訊息產生與通訊最佳化研究(2/2)

計畫類別：■ 個別型計畫　　□ 整合型計畫
計畫編號：NSC　94－2213－E－216－002－
執行期間：　94 年 08 月 01 日至 95 年 07 月 31 日
執行單位：　中華大學資訊工程學系

計畫主持人：許慶賢 副教授
共同主持人：
計畫參與人員：蔡秉儒、翁銘遠、藍朝陽 中華大學資訊工程學系研究生

成果報告類型(依經費核定清單規定繳交)：□精簡報告　■完整報告

本成果報告包括以下應繳交之附件：
□赴國外出差或研習心得報告一份
□赴大陸地區出差或研習心得報告一份
■出席國際學術會議心得報告及發表之論文各一份
□國際合作研究計畫國外研究報告書一份

處理方式：除產學合作研究計畫、提升產業技術及人才培育研究計畫、列管計畫及下列情形者外，得立即公開查詢
□涉及專利或其他智慧財產權，□一年□二年後可公開查詢

中　華　民　國　94　年　10　月　14　日

行政院國家科學委員會專題研究計畫結案報告

平行編譯器中不規則資料重新分佈之訊息產生與通訊最佳化研究

中文摘要

本報告是有關於在分散式記憶體環境下對 GEN_BLOCK 動態式資料重新配置技術之描述。在這一個計畫中，我們研發了對 GEN_BLOCK 資料重新配置的通訊排程演算法，*Two-Phase Degree-Reduction* (*TPDR*)，以及對估算通訊成本做較佳化的演算法，*Local Message Reduction*(*LMR*)。*TPDR* 是由兩個排程方法所組成的演算法：第一個是本計畫所發展的排程演算法，優先處理需傳送最多訊息或接收最多訊息的處理器；第二個演算法運用著色理論和修正著色排程機制，找出最後兩個步驟中，訊息適合擺放的位置，以此協助完成第一個排程演算法未完成的排程步驟。*LMR* 針對網路環境中的傳輸速度與本機傳送速度的差異，給予每一筆通訊較佳通訊成本，幫助排程演算法降低資料重新配置的時間，是一個適用於任何通訊排程演算法的方法。在本計畫中，由於 *TPDR* 有效降低整體排程步驟的通訊時間並且避免處理器互相競爭傳送訊息的優先順序，且 *LMR* 提供比以往更能反應出真實傳輸時間的通訊成本理論模組，因此有助於提升平行編譯器運算平行程式的整體效能。

關鍵詞：GEN_BLOCK、動態式資料重新配置、分散式記憶體、排程演算法、通訊成本理論模組、平行編譯器。

Design and Implementation of Resource Allocation and Job Scheduling for Supporting SPMD Programs on Heterogeneous Parallel Computing Networks

Abstract

This report presents the development and implementation of runtime GEN_BLOCK data redistribution scheduling algorithm for irregular data redistribution on distributed memory computing environments. In this project, we developed an irregular array redistribution scheduling algorithm, *two-phase degree-reduction* (*TPDR*) and a method to provide better cost when computing cost of each communication, *local message reduction* (*LMR*). *TPDR* is consisted of two parts: the first part schedules messages which are from processors that have most messages to be sent or received; the second part is used to schedule the left two scheduling steps that can't be completed in first part using coloring method and an adjustable coloring mechanism. *LMR* estimates the difference of speed ratio between a node transmitting data to others and to itself. After the difference is estimated, *LMR* lowers cost of communications that a node transmits to itself. It helps scheduling algorithm reducing redistribution time. It is an algorithm independent method. In this project, *TPDR* scheduling algorithm and *LMR* can facilitate performance improvement on parallel compiler environments because of helping shortening overall communication time, avoiding node contention and giving better theoretical module of communication cost when performing GEN_BLOCK data redistribution.

Keywords: GEN_BLOCK, Dynamic Data Redistribution, Distributed Memory, Scheduling Algorithm, Theoretical module of communication cost, Parallel Compiler.

一、緣由與目的

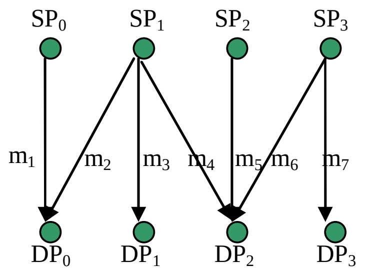
在分散式記憶體環境裡，大部分的科學運算程式在執行的過程中，需要經常改變資料配置方式，以便配合不同運算階段中對處理器的資料需求。在平行程式重新配置資料的過程中，處理器需要經常移動大量的資料已達到平行程式的資料配置需求。然而花費的通訊成本過高時會影響程式整體的執行效能，因此，動態的資料配置技術在分散式記憶統環境中，是一個重要的研究方向。過去，由於分散式記憶體環境都是由相同能力的電腦組成，因此，資料重新配置研究主要著重在規則的資料重新配置上。隨著網路的發達，利用網際網路上的電腦進行平行運算的研究越來越受到重視，因此需要針對不同能力的電腦組成的分散式記憶體環境開發不規則的資料重新配置通訊演算法，但目前仍然少有針對不規則的資料重新配置型態所發展的演算法。在本計畫中，我們將在分散式記憶體環境下，針對不規則的資料分配情形提出高效率的通訊排程演算法。主要方向為研發最少排程步驟的排程方法，並尋找和可以利用著色理論縮短整體排程時間的排程方式。接著，我們針對環境中不同的訊息傳輸速度，研發出通訊成本理論模組，給予不同類型的訊息不同的通訊成本。在設計排程演算法的同時我們也將設計一套理論分析模組和資料配置參數產生器，以便比較理論上和實際上的排程結果是否互相對應。

二、研究方法與成果

由於 GEN_BLOCK 運算程式要求分配不同大小的連續資料區塊給各個處理器，不容易有效率地執行訊息封裝和資料重新配置(Data Redistribution)。為了降低資料重新配置所花費的通訊時間，發展有效率的資料排程演算法是必需的。

資料重新配置時，處理器與資料通訊的關係是由兩個資料配置參數(Data Distribution scheme)所形成。將這個關係表示成二分圖就如圖一所示，SP_0 到 SP_3 為二分圖中的一個節點集合(Vertex Set)，代表資料未重新分配時的處理器集合；DP_0 到 DP_3 則是二分圖中的另一個節點集合，代表資料重新分配後的處理器集合；m_1 到 m_7 是二分圖中的邊(Edge)，代表處理器之間傳送訊息的關係。

本計畫提出的資料重新配置之排程演算法為 *Two-Phase Degree Reduction* (*TPDR*)，主要分成兩個部份。第一個部分先對需要傳送或接受最多通訊訊息的處理器做通訊排程，如圖一中的 SP_1 和 DP_2，也就是先處理二分圖中與擁有最大 *Degree* 的節點(處理器)有關係的邊(通訊訊息)。



圖一、處理器與資料通訊的關係。

第一個部分會循序降低最大 *Degree* 的值，每降低最大 *Degree* 1 時，演算法便能排出一個排程步驟。當最大 *Degree* 的值小於等於 2 時，*TPDR* 排程演算法將進入第二個部份。第二部份使用著色理論的概念將第一部分尚未排入排程步驟的通訊訊息塗上兩種顏色以便分成兩個邊集合(Edge Set)分別排入兩個排程步驟中。在排程演算法的第二部份中也採用顏色修正機制，當通訊訊息的顏色改變可降低通訊成本且不發生衝突時使用。

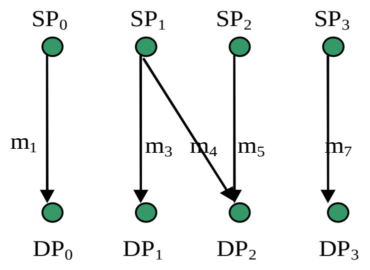
TPDR 排程演算法的第一部分分成三步，分述如下：

第一步：在二分圖中，找出目前 *Degree* 值是最大的節點，並且依照仍需傳送或接收通訊訊息的總量節點排序。假設有 *k* 個節點的 *Degree* 值是最大的，且目前最大 *Degree* 值是 *d*，則對處理器排序完後得到一集合$<V_{i1}, V_{i2}, \ldots, V_{ik}>$。

第二步：由步驟一排序節點的結果中，逐一挑出最大總量的節點，將他們仍需傳送或接收的最小量通訊訊息排入目前的第 *d* 個排程步驟中。

第三步：當 *Degree* 最大值大於 2 時，重複步驟一和步驟二。

圖一中的最大 *Degree* 值為 3，在第一步中，SP_1和 DP_2首先依照通訊訊息的傳輸量被排序。假設 m_2 和 m_6 分別為 SP_1 和 DP_2 所要傳送和接收的最小通訊訊息，m_2 和 m_6 將會被排入排程步驟 3 中。此時，圖一中的二分圖將變成由 m1、m3、m4、m5 和 m7 所組成的二分圖，如圖二所示。



圖二、挑選出 m_2 和 m_6 排入排程步驟 3 以後的二分圖。由於目前最大 *Degree* 為 2，所以剩餘的通訊訊息將由 *TPDR* 排程演算法的第二部份進行排程。

因為最大 *Degree* 值為 2，排程演算法進入利用著色理論來排通訊訊息的第二部份。將 m_1、m_3、m_5 和 m_7 塗上紅色，m_4 塗上藍色。紅色和藍色分別代表排程步驟 1 和 2，因此可得到排程結果，如圖三。

在著色的過程中，假如顏色修正機制發現 m_4 放置在第一個排程步驟且 m_3 和 m_5 放再第二個排程步驟後，會使得整體的傳送時間減少的話，則 m_4 應塗成紅色，m_3 和 m_5 應塗成藍色，其排程結果便如圖四所示。

Schedule Table	
Step 1	m_1、m_3、m_5、m_7
Step 2	m_4
Step 3	m_2、m_6

圖三、經由 TPDR 排程演算法排程後所得到的結果。

Schedule Table	
Step 1	m_1、m_4、m_7
Step 2	m_3、m_5
Step 3	m_2、m_6

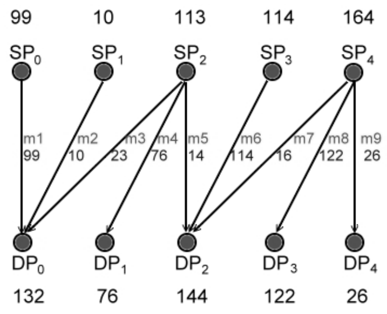
圖四、顏色修正機制更換了 m_4 以及 m_3 和 m_5 在 Step 1 和 Step 2 的位置。

我們針對不同的資料配置情形和各個排程演算法進行分析，發現排程演算法是否容易找到最佳或花費最少排程時間的排程結果與資料配置情形有關聯。當我們設定每個處理器取得的資料為平均值的正負百分之三十時，也就是當平行程式要求資料分配的情形為資料較平均地分配在各個處理器時，通訊排程演算法較不易找到接近最佳解的通訊排程結果。然而當資料明顯地被集中在某個或是某些少數處理器時，通訊排程演算法大部分都可以找到花費較少的排程結果。然而在比較這些結果後，本計畫中提出的排程演算法明顯地勝出其他演算法許多。

LMR 的通訊成本理論模組提供傳輸速率比來改善資料重新分配的效率，是獨立於排程演算法的技術。其主要精神在去除節點傳送訊息給自己時被過於考慮的通訊成本，以反應出實際在機器上執行的時間。因此，*LMR* 先測量節點傳送訊息給其他節點以及給自己的速率，將其差異表現在估算的通訊成本中，譬如，節點傳送訊息給自己的，其通訊成本應比傳給其他節點的低，不能視為同一通訊類型。*LMR* 提供兩者的速率比，用來降低本地端傳送的成本或增加外地傳送的成本，以突顯兩者不同。讓排程演算法先得知每個通訊其合理的通訊成本後在進行排程，會有較佳的排程結果。圖五結合 *TPDR*

與 *LMR*，並說明考慮速率比的重要性。

圖五(a)是一個二分圖，說明資料重新分配前後，資料來源端與目的端的關係，因最大分支為三，如 SP_2、DP_2 與 SP_4，最小排程步驟有三步。圖五(b)是 *TPDR* 的排程結果，結果顯示重新分配資料的成本為 297。若考慮實際通訊的時間，本地端通訊應給予較低的通訊成本，則其成本應為 211，如圖五(c)所示，為排程後才考慮速率比的排程結果。圖五(d)則事先考慮速率比，在改善通訊成本後，讓 *TPDR* 演算法進行排程。結果顯示重新分配資料的成本變成更低的 155，較圖五(c)的 211 少了 56。這個例子說明排程演算法應重視本地端與外地端通訊成本的差異，改善成本後再進行排程，有助於提高資料重新分配的效率。我們在實驗部分發現 *LMR* 不僅能改善 *TPDR*，更能改善其他排程演算法，降低資料重新分配所需的時間，是一個獨立於排程演算法的技術。



(a)

Schedule Table		Cost
Step 1	m_3、m_6、m_8	122 (m_8)
Step 2	m_1、m_5、m_9	99 (m_1)
Step 3	m_2、m_4、m_7	76 (m_4)
Total cost = 297		

(b)

Schedule Table		Cost
Step 1	m_3、m_6、m_8	122 (m_8)
Step 2	**m_1、m_5、m_9**	**13** (m_1)
Step 3	m_2、m_4、m_7	76 (m_4)
Total cost = **211**		

(c)

Schedule Table		Cost
Step 1	m_1、m_4、m_6、m_8	122 (m_8)
Step 2	m_3、m_7	23 (m_3)
Step 3	m_2、m_5、m_9	10 (m_2)
Total cost = 155		

(d)

圖五、(a)資料重新分配之二分圖。(b)*TPDR* 的排程結果，通訊成本為 297。(c)利用 *LMR* 分析圖五(b)能反應真實傳送時間的通訊成本，即降低 m_1、m_5 與 m_9 三個本地端通訊的成本，變成 211。(d)排程前利用 *LMR* 改善通訊成本，讓 *TPDR* 排程後得到較圖五(c)更低的通訊成本，155。

三、結果與討論

下面我們歸納本計畫主要的成果:

- 完成訊息產生與封裝技術實作 - 為了知道確切的通訊訊息交換量，我們完成了訊

息索引計算涵式用來產生訊息，並完成封裝技術以便快速傳送訊息。

- 完成Interval graph 之演算法實作 - 我們完成Interval graph之演算法實作，可用來判斷通訊訊息的最少排程步驟。
- 完成GEN_BLOCK資料重新分配之排程演算法實作 - 針對最小排程步驟分析可行的排程演算法，找出最大*Degree*與最小步驟的值是相等的。因此以循序降低最大*Degree*進行通訊排程，並完成演算法與實作。
- 完成著色理論嵌入不規則排程的模組 - 利用著色理論得到的排程雖然是最小排程步驟，但無法得到最低通訊量的排程結果，因此嵌入*TPDR*排程演算法中，利用其特性排程不需要在乎通訊量的通訊訊息。
- 完成List scheduling algorithm[22]以及Divide-and-conquer scheduling algorithm [20]之實作 - 為了證實本計畫開發的排程演算法為有效率的排程演算法，需實做其他排程演算法以便進行實驗。
- 完成用於分析通訊排程步驟與訊息長度成本的理論模組 - 為了比較排程演算法的優缺，我們完成了通訊成本的理論模組，用以判斷排程結果的好壞。
- 完成解決訊息傳送可能引起競爭之研究 - 通訊訊息再被傳送或接收的過程中易引起處理器互相競爭，增加通訊成本。我們的排程演算法成功地避免了傳送訊息時可能引起的通訊競爭情形。
- 完成資料配置參數產生器 - 為了模擬實際的資料配置參數，我們實做了一個資料配置產生器，產生資料集中於某些處理器和平均分散在各個處理器上的資料配置參數。
- 完成傳輸速率比的理論模組 - 為了讓通訊成本能確實反應出傳送資料花費的時間，我們完成了傳輸速率比的理論模組，避免排程演算法處理到不正確的訊息。
- 完成網路傳輸速率測試程式之實作 - 未提供傳輸速率比的理論模組測試數據，需要測試網路環境中的真實傳輸速度。
- 完成實驗數據分析模組 - 為了比較 *LMR* 改善排程演算法的程度，我們提出分析模組，分析改善前後的優劣。
- 執行本計畫所發表之相關論文列舉如下

1. Ching-Hsien Hsu, Shih-Chang Chen and Chao-Yang Lan, "Scheduling Contention-Free Irregular Redistribution in Parallelizing Compilers," Accepted, The *Journal of Supercomputing*, Kluwer Academic Publisher, 2006. // TPDR 演算法
2. Ching-Hsien Hsu, Min-Hao Chen, Chao-Tung Yang and Kuan-Ching Li, “ Optimizing

Communications of Dynamic Data Redistribution on Symmetrical Matrices in Parallelizing Compilers,” *IEEE Transactions on Parallel and Distributed Systems*, Vol. 17, No. 11, pp. -, Nov. 2006. // 對稱矩陣資料重新分配技術

3. Ching-Hsien Hsu, “Sparse Matrix Block-Cyclic Realignment on Distributed Memory Machines," The *Journal of Supercomputing*, Vol. 33, No. 3, pp. 175-196, September 2005, Kluwer Academic Publisher. (SCI, EI, NSC 93-2213-E-216-029) // 稀疏矩陣資料重新分配技術
4. Ching-Hsien Hsu, Chao-Yang Lan and Shih-Chang Chen, “Optimizing Scheduling Stability for Runtime Data Alignment,” *Embedded System Optimization - Lecture Notes in Computer Science*, Vol. 4097, pp. 825-835, Springer-Verlag, Aug. 2006. // Local Message Reduction (局部最佳化)通訊排程演算法
5. Ching-Hsien Hsu, Shih-Chang Chen, Chao-Yang Lan, Chao-Tung Yang and Kuan-Ching Li, “Scheduling Convex Bipartite Communications Towards Efficient GEN_BLOCK Transformations,” *Parallel and Distributed Processing and Applications - Lecture Notes in Computer Science*, Vol. 3758, pp. 419-424, Springer-Verlag, Nov. 2005. (ISPA’05) // 利用二分圖與著色理論的排程演算法
6. Chang Wu Yu, Ching-Hsien Hsu, Kun-Ming Yu, C.-K. Liang and Chun-I Chen, “Irregular Redistribution Scheduling by Partitioning Messages,” *Computer Systems Architecture - Lecture Notes in Computer Science*, Vol. 3740, pp. 295-309, Springer-Verlag, Oct. 2005. // 利用訊息分割的通訊排程技術
7. Shih-Chang Chen (博士班學生), Ching-Hsien Hsu, Chao-Yang Lan, Chao-Tung Yang and Kuan-Ching Li, "Efficient Communication Scheduling Methods for Irregular Array Redistribution in Parallelizing Compilers," Parallel Computing Technologies - *Lecture Notes in Computer Science*, Vol. 3606, pp. 216-225, Springer-Verlag, Sep. 2005. (PaCT’05) // Contention free 排程技術

四、計劃成果自評

本計畫之研究成果相當豐碩。第一年度、在平行環境下對不規則的資料重新配置做無處理器競爭之排程演算法這一個研究主題上共計發表三篇研討會論文以及二篇期刊論文。第二年度、在 LMR 技術的主題上，我們也發表了兩篇國際論文。目前正在整理這一兩年的研究成果，將投稿至 IEEE Transactions on Parallel and Distributed Systems. 目前初稿已完成 80%預計 11 月底以前可以投出去。雖然計畫已經結束，但是我們也發覺這一個題目還有一些有趣的問題可以深入研究。這些問題預計可以寫成 1 至 2 篇期刊論文。不論未來是否有執行相關計畫，這些題目我們都會繼續做下去。本計畫有豐碩的研究成果，感謝國科會給予機會。下一個年度，我們將更加努力，爭取經費建立更完備的研究環境。另外，對於參與研究計畫執行同學的認真，本人亦表達肯定與感謝。

五、參考文獻

1. G. Bandera and E.L. Zapata, “Sparse Matrix Block-Cyclic Redistribution,” *Proceeding of IEEE Int'l. Parallel Processing Symposium* (IPPS'99), San Juan, Puerto Rico, April 1999.
2. Frederic Desprez, Jack Dongarra and Antoine Petitet, “Scheduling Block-Cyclic Data redistribution,” *IEEE Trans. on PDS*, vol. 9, no. 2, pp. 192-205, Feb. 1998.
3. Minyi Guo, “Communication Generation for Irregular Codes,” *The Journal of Supercomputing*, vol. 25, no. 3, pp. 199-214, 2003.
4. Minyi Guo and I. Nakata, “A Framework for Efficient Array Redistribution on Distributed Memory Multicomputers,” *The Journal of Supercomputing*, vol. 20, no. 3, pp. 243-265, 2001.
5. Minyi Guo, I. Nakata and Y. Yamashita, “Contention-Free Communication Scheduling for Array Redistribution,” *Parallel Computing*, vol. 26, no.8, pp. 1325-1343, 2000.
6. Minyi Guo, I. Nakata and Y. Yamashita, “An Efficient Data Distribution Technique for Distributed Memory Parallel Computers,” *JSPP'97*, pp.189-196, 1997.
7. Minyi Guo, Yi Pan and Zhen Liu, “Symbolic Communication Set Generation for Irregular Parallel Applications,” *The Journal of Supercomputing*, vol. 25, pp. 199-214, 2003.
8. C.-H Hsu, S.-W Bai, Y.-C Chung and C.-S Yang, “A Generalized Basic-Cycle Calculation Method for Efficient Array Redistribution,” *IEEE TPDS*, vol. 11, no. 12, pp. 1201-1216, Dec. 2000.
9. Ching-Hsien Hsu, Chao-Yang Lan and Shih-Chang Chen, “Optimizing Scheduling Stability for Runtime Data Alignment,” *Embedded System Optimization - Lecture Notes in Computer Science*, Vol. 4097, pp. 825-835, Springer-Verlag, Aug. 2006. (ESO’ 06) (SCI Expanded, NSC92-2213-E-216-029)
10. Ching-Hsien Hsu, Chao-Yang Lan and Shih-Chang Chen, "Scheduling Contention-Free Irregular Redistribution in Parallelizing Compilers," Accepted, The *Journal of Supercomputing*, Kluwer Academic Publisher. (SCI, EI, NSC93-2213-E-216-028, NCHC-KING-010200)
11. C.-H Hsu, Dong-Lin Yang, Yeh-Ching Chung and Chyi-Ren Dow, “A Generalized Processor Mapping Technique for Array Redistribution,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, vol. 7, pp. 743-757, July 2001.
12. Edgar T. Kalns, and Lionel M. Ni, “Processor Mapping Technique Toward Efficient Data Redistribution,” *IEEE Trans. on PDS*, vol. 6, no. 12, December 1995.

13. S. D. Kaushik, C. H. Huang, J. Ramanujam and P. Sadayappan, “Multiphase data redistribution: Modeling and evaluation,” *Proceeding of IPPS*’95, pp. 441-445, 1995.
14. S. Lee, H. Yook, M. Koo and M. Park, “Processor reordering algorithms toward efficient GEN_BLOCK redistribution,” *Proceedings of the ACM symposium on Applied computing*, 2001.
15. Y. W. Lim, Prashanth B. Bhat and Viktor and K. Prasanna, “Efficient Algorithms for Block-Cyclic Redistribution of Arrays,” *Algorithmica*, vol. 24, no. 3-4, pp. 298-330, 1999.
16. Neungsoo Park, Viktor K. Prasanna and Cauligi S. Raghavendra, “Efficient Algorithms for Block-Cyclic Data redistribution Between Processor Sets,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, No. 12, pp.1217-1240, Dec. 1999.
17. Antoine P. Petitet and Jack J. Dongarra, “Algorithmic Redistribution Methods for Block-Cyclic Decompositions,” *IEEE Trans. on PDS*, vol. 10, no. 12, pp. 1201-1216, Dec. 1999.
18. L. Prylli and B. Touranchean, “Fast runtime block cyclic data redistribution on multiprocessors,” *Journal of Parallel and Distributed Computing,* vol. 45, pp. 63-72, Aug. 1997.
19. S. Ramaswamy, B. Simons, and P. Banerjee, “Optimization for Efficient Data redistribution on Distributed Memory Multicomputers,” *Journal of Parallel and Distributed Computing,* vol. 38, pp. 217-228, 1996.
20. Akiyoshi Wakatani and Michael Wolfe, “Optimization of Data redistribution for Distributed Memory Multicomputers,” short communication, *Parallel Computing*, vol. 21, no. 9, pp. 1485-1490, September 1995.
21. Hui Wang, Minyi Guo and Daming Wei, "Divide-and-conquer Algorithm for Irregular Redistributions in Parallelizing Compilers”, *The Journal of Supercomputing*, vol. 29, no. 2, 2004.
22. Hui Wang, Minyi Guo and Wenxi Chen, “An Efficient Algorithm for Irregular Redistribution in Parallelizing Compilers,” *Proceedings of 2003 International Symposium on Parallel and Distributed Processing with Applications*, LNCS 2745, 2003.
23. H.-G. Yook and Myung-Soon Park, “Scheduling GEN_BLOCK Array Redistribution,” Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems, November, 1999.

行政院所屬各機關人員出國報告書提要

撰寫時間： 95 年 10 月 24 日

姓名	許慶賢	服務機關名稱	中華大學資工系	連絡電話、電子信箱	03-5186410 chh@chu.edu.tw
出生日期	62 年 2 月 23 日		職稱	副教授	
出席國際會議名稱	Eighth International Conference on Parallel Computing Technologies (PaCT’05), September 5 -9 2005.				
到達國家及地點	**Krasnoyarsk, Russia**		出國期間	自 95 年 09 月 03 日 迄 95 年 09 月 11 日	
內容提要	這一次在俄羅斯所舉行的國際學術研討會議共計五天。第一天上午由 **Thomas L. Casavant** 博士針對 Genomics and Biomedical Applications for Parallel Computing 以及 Fine Grained Parallelism in Spatial Dynamics Simulation 主題發表精闢的演說作為研討會的開始。第二天許多重要的研究成果分為兩個平行的場次進行論文發表。本人選擇了 Architecture and Infrastructure 場次聽取報告。晚上本人亦參加酒會，並且與幾位國外學者及中國、香港教授交換意見。第三天本人在上午聽取了 Data and Information Management 相關研究，同時獲悉許多新興起的研究主題，並了解目前國外大多數學者主要的研究方向。當天下午發表我們的論文，本人亦參與大會所舉辦的晚宴。並且與幾位外國學者認識，交流，合影留念。會議第四天是一天的 Social Program。會議最後一天，本人選擇與這一次論文較為相近的 Scheduling, Fault Tolerance and Mapping 以及分散式計算研究聽取論文發表，並且把握最後一天的機會與國外的教授認識，希望能夠讓他們加深對台灣研究的印象。五天下來，本人聽了許多優秀的論文發表。這些研究所涵蓋的主題包含有：網格系統技術、工作排程、網格計算、網格資料庫以及無線網路等等熱門的研究課題。此次的國際學術研討會議有許多知名學者的參與，讓每一位參加這個會議的人士都能夠得到國際上最新的技術與資訊。是一次非常成功的學術研討會。參加本次的國際學術研討會議，感受良多。讓本人見識到許多國際知名的研究學者以及專業人才，得以與之交流。讓本人與其他教授面對面暢談所學領域的種種問題。看了眾多研究成果以及聽了數篇專題演講，最後，本人認為，會議所安排的會場以及邀請的講席等，都相當的不錯，覺得會議舉辦得很成功，值得我們學習。				
出席人所屬機關審核意見					
層轉機關審核意見					
研考會處理意見					

(出席 PaCT-05 研討會所發表支論文)

Efficient Communication Scheduling Methods for Irregular Array Redistribution in Parallelizing Compilers[1]

Shih-Chang Chen, Ching-Hsien Hsu[2] and Chao-Yang Lan

Department of Computer Science and Information Engineering
Chung Hua University, Hsinchu, Taiwan 300, R.O.C.

chh@chu.edu.tw

Abstract. Many scientific problems have been solved on distributed memory multi-computers. For solving those problems, efficient data redistribution algorithm is necessary. Irregular array redistribution has been paid attention recently since it can distribute different size of data segment to processors according to their own computation ability. It's also the reason why it has been kept an eye on load balance. *High Performance Fortran Version 2* (HPF2) provides *GEN_BLOCK* (*generalized block*) distribution format which facilitates *generalized block* distributions. In this paper, we present a *two-phase degree-reduction* (*TPDR*) method for scheduling HPF2 irregular array redistribution. In a bipartite graph representation, the first phase schedules communications of processors that with degree greater than two. Every communication step will be scheduled after a degree-reduction iteration. The second phase schedules all messages of processors that with degree-2 and degree-1 using an adjustable coloring mechanism. An extended algorithm based on *TPDR* is also presented in this paper. Effectiveness of the proposed methods not only avoids node contention but also shortens the overall communication length. To evaluate the performance of our methods, we have implemented both algorithms along with the divide-and-conquer algorithm. The simulation results show improvement of communication costs. The proposed methods are also practicable due to their low algorithmic complexity.

Keywords: Irregular redistribution, communication scheduling, `GEN_BLOCK`, degree-reduction

1. Introduction

Parallel computing systems have been used to solve complex scientific problems using their powerful computational ability. Dealing with those kinds of problems, systems must process large number of data. For this reason, keeping load balancing is important. In order to achieve a good performance of load balancing, using an appropriate data distribution scheme [8] when processing different phase of application is necessary. In general, data distribution can be classified into regular and irregular. The regular distribution usually employs `BLOCK`, `CYCLIC`, or `BLOCK-CYCLIC(`*c*`)` to specify array decomposition. The irregular distribution uses user-defined functions to specify unevenly array distribution.

To map unequal sized continuous segments of array onto processors, High Performance Fortran version 2 (HPF2) provides `GEN_BLOCK` distribution format which facilitates generalized block distributions. `GEN_BLOCK` allows unequal sized data segments of an array to be mapped onto processors. This makes it possible to let different processors dealing with appropriate data quantity according to their computation ability.

In some algorithms, an array distribution that is well-suited for one phase may not be good for a subsequent phase in terms of performance. Array redistribution is needed when applications running from one sub-algorithm to another during run-time [6]. Therefore, many data parallel programming languages support run-time primitives for changing a program's array decomposition. Since array redistribution is performed at run-time, there is a performance trade-off between the efficiency of the new data decomposition for a subsequent phase of an algorithm and the cost of redistributing array among processors. Thus efficient methods for performing array redistribution are of great importance for the development of distributed memory compilers for those languages.

Communication scheduling is one of the most important issues on developing runtime array redistribution techniques. In this paper, we present a two-phase degree reduction (*TPDR*) algorithm to efficiently perform `GEN_BLOCK` array redistribution. The main idea of the two-phase degree reduction method is to schedules communications of processors that with degree greater than two in the first phase (degree reduction phase).

[1] The work of this paper was supported by NSC, National Science Council of Taiwan, under grant number NSC-93-2213-E-216-029.

[2] The correspondence address

Every communication step will be scheduled after a degree-reduction iteration. The second phase (coloring phase) schedules all messages of processors that with degree-2 and degree-1 using an adjustable coloring mechanism. Based on the *TPDR* method, we also present an extended *TPDR* algorithm (*E-TPDR*). The proposed techniques have the following characteristics:

- It is a simple method with low algorithmic complexity to perform GEN_BLOCK array redistribution.
- The two-phase degree reduction technique can avoid node contentions while performing irregular array redistribution.
- The two-phase degree reduction method is a single pass scheduling technique. It does not need to re-schedule / re-allocate messages. Therefore, it is applicable to different processor groups without increasing the scheduling complexity.

The rest of this paper is organized as follows. In Section 2, a brief survey of related work will be presented. In section 3, we will introduce an example of GEN_BLOCK array redistribution as preliminary. Section 4 presents two communication scheduling algorithms for irregular redistribution problem. The performance analysis and simulation results will be presented in section 5. Finally, the conclusions will be given in section 6.

2. Related Work

Many methods for performing array redistribution have been presented in the literature. These researches are usually developed for regular or irregular problems [5] in multi-computer compiler techniques or runtime support techniques. We briefly describe the related works in these two aspects.

Techniques for regular array redistribution, in general, can be classified into two approaches: the communication sets identification techniques and communication optimizations. The former includes the *PITFALLS* [17] and the *ScaLAPACK* [16] methods for index sets generation; Park *et al.* [14] devised algorithms for BLOCK-CYCLIC Data redistribution between processor sets; Dongarra *et al.* [15] proposed algorithmic redistribution methods for BLOCK-CYCLIC decompositions; Zapata *et al.* [1] proposed parallel sparse redistribution code for BLOCK-CYCLIC data redistribution based on *CRS* structure. The *Generalized Basic-Cycle Calculation* method was presented in [3].

Techniques for communication optimizations category, in general, provide different approaches to reduce the communication overheads in a redistribution operation. Examples are the processor mapping techniques [10, 12, 4] for minimizing data transmission overheads, the multiphase redistribution strategy [11] for reducing message startup cost, the communication scheduling approaches [2, 7, 13, 21] for avoiding node contention and the strip mining approach [18] for overlapping communication and computational overheads.

On irregular array redistribution, some work has concentrated on the indexing and message generation while some has addressed on the communication efficiency. Guo *et al.* [9] presented a symbolic analysis method for communication set generation and to reduce communication cost of irregular array redistribution. On communication efficiency, Lee *et al.* [12] presented a logical processor reordering algorithm on irregular array redistribution. Four algorithms were discussed in this work for reducing communication cost. Guo *et al.* [19, 20] proposed a divide-and-conquer algorithm for performing irregular array redistribution. In this method, communication messages are first divided into groups using Neighbor Message Set (NMS), messages have the same sender or receiver; the communication steps will be scheduled after those NMSs are merged according to the relationship of contention. In [21], a relocation algorithm was proposed by Yook and Park. The relocation algorithm consists of two scheduling phases, the list scheduling phase and the relocation phase. The list scheduling phase sorts global messages and allocates them into communication steps in decreasing order. Because of conventional sorting operation, list scheduling indeed performs well in term of algorithmic complexity. If a contention happened, the relocation phase will perform a serial of re-schedule operations. While algorithm flow goes to the relocation phase, it has to allocate an appropriate location for the messages that can't be scheduled at that moment. This leads to high scheduling overheads and degrades the performance of a redistribution algorithm.

3. GEN_BLOCK Array Redistribution

In this section, we will present some properties of irregular array redistribution. To simplify the presentation, notations and terminologies used in this paper are prior defined as follows.

Definition 1: Given an irregular GEN_BLOCK redistribution on a one-dimension array $A[1:N]$ over P processors, the *source processors* of array elements $A[1:N]$ is denoted as SP_i; the *destination processors* of array elements $A[1:N]$ is denoted as DP_i where $0 \le i \le P-1$.

Definition 2: A bipartite graph $G = (V, E)$ is used to represent the communications of an irregular array redistribution on $A[1:N]$ over P processors. Vertices of G are used to represent the source and destination processors. Edge e_{ij} in G denotes the message sent from SP_i to DP_j, where $e_{ij} \in$ E. $|E|$ is given as the total number of communication messages through the redistribution.

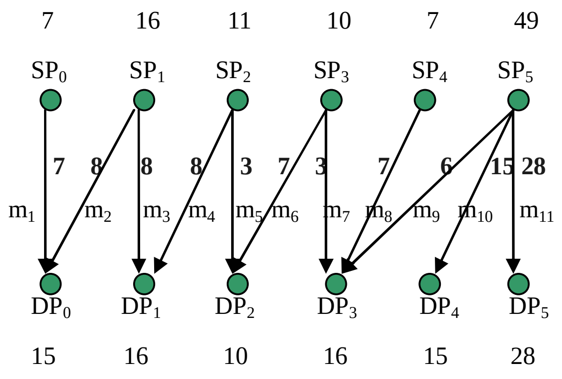
Figure 1(a) shows an example of redistributing two GEN_BLOCK distributions on an array $A[1:100]$. Distributions I and II are mapped to source processors and destination processors, respectively. The communications between source and destination processor sets are depicted in Figure 1(b). There are totally eleven communication messages ($|E|=11$), m_1, m_2, m_3…, m_{11} among processors involved in the redistribution. In general, to avoid conflict communication or node contention, a processor can only send one message to destination processors at a communication step. Similarly, one can only receive a message from source processors at any communication step. In other words, messages sent or received by the same processor can't be scheduled in the same communication step. The messages which can not be scheduled in the same communication step are called conflict tuple [19]. For instance, $\{m_1, m_2\}$ is a conflict tuple since they have common destination processor DP_0; $\{m_2, m_3\}$ is also a conflict tuple because of the common source processor SP_1. Figure 1(c) shows a simple schedule for this example.

Unlike regular problem, there is no repetition communication pattern in irregular GEN_BLOCK array redistribution. It is also noticed that if SP_i sends messages to DP_{j-1} and DP_{j+1}, the communication between SP_i and DP_j must exist, where $0 \le i, j \le P-1$. This result was mentioned as the consecutive communication property [12].

Distribution I (Source Processor)						
SP	SP_0	SP_1	SP_2	SP_3	SP_4	SP_5
Size	7	16	11	10	7	49

Distribution II (Destination Processor)						
DP	DP_0	DP_1	DP_2	DP_3	DP_4	DP_5
Size	15	16	10	16	15	28

(a)



(b)

Schedule Table	
Step 1	m_1、m_3、m_5、m_7、m_{10}
Step 2	m_2、m_4、m_6、m_8、m_{11}
Step 3	m_9

(c)

Figure 1: An example of irregular array redistribution. (a) The source and destination distributions. (b) Bipartite representation of communications between source and destination processors. (c) A simple schedule of irregular array redistribution.

4. Communication Scheduling

The communication time depends on total number of communication steps and the length of these steps. In general, the message startup cost is direct proportional to the number of communication steps. The length of these steps determines the data transmission overheads. A minimal steps scheduling can be obtained using the coloring mechanism. However, there are two drawbacks in this method; it can not minimize total size of communication steps; the graph coloring algorithmic complexity is often high. In the following subsections, we will present two low complexity and high availability scheduling methods.

4.1 The Two-Phase Degree Reduction Method

The Two-Phase Degree Reduction (*TPDR*) method consists of two parts. The first part schedules communications of processors that with degree greater than two. In a bipartite graph representation, the *TPDR* reduces the degree of maximum degree nodes by one in each reduction iteration. The second part schedules all messages of processors that with degree-2 and degree-1 using an adjustable coloring mechanism. The degree reduction is performed as follows.

Step1: Sort the vertices that with maximum degree d by total size of messages in decreasing order. Assume there are k nodes with degree d. The sorted vertices would be $<V_{i1}, V_{i2}, \ldots, V_{ik}>$.

Step2: Schedule the minimum message $m_j = \min\{m_1, m_2, \ldots, m_d\}$ into step d for vertices $V_{i1}, V_{i2}, \ldots, V_{ik}$, where $1 \le j \le d$.

Step3: Maximum degree $d = d$-1. Repeat Steps 1 and 2.

We first use an example to demonstrate the degree reduction technique without considering the coloring scheme. Figure 2(a) shows the communication patterns initially. The source `GEN_BLOCK` distribution is of (7, 10, 4, 18, 7, 18, 36). The destination `GEN_BLOCK` distribution is of (10, 14, 18, 14, 14, 12, 18). The redistribution is carried out over seven processors with maximum degree 3. Therefore, the communications can be scheduled in three steps. According to the above description in step 1, there are two nodes with degree 3, SP_6 and DP_1.

The total message size of SP_6 (36) is greater than DP_1 (14). Thus, SP_6 is the first candidate to select a minimum message (m_{11}) of it into step 3. A similar selection is then performed on DP_1. Since m_5 is the minimum message of DP_1 at present, therefore, m_5 is scheduled into step 3 as well. As messages m_{11} and m_5 are removed from the bipartite graph, adjacent nodes of edges m_{11} and m_5, i.e., SP_6, DP_4, DP_1 and SP_3 should update their total message size. After the degree reduction iteration, the maximum degree of the bipartite graph will become 2. Figures 2(b) to 2(d) show this scenario. Figures 2(e) and 2(f) show the similar process of above on degree = 2 bipartite graph. In Figure 2(e), vertices SP_6, SP_5, SP_4, SP_1, DP_3, DP_2, DP_1 and DP_0 have the maximum degree 2 and are candidates to schedule their messages into step 2. According to the degree reduction method, m_{12}, m_{10} and m_7 are scheduled in order. The next message to be selected is m_8. However, both messages of DP_3 will result node contention (one with SP_4 and one with SP_5) if we are going to schedule one of DP_3's messages. This means that the degree reduction method might not reduce degree-2 edges completely when the degree is 2.

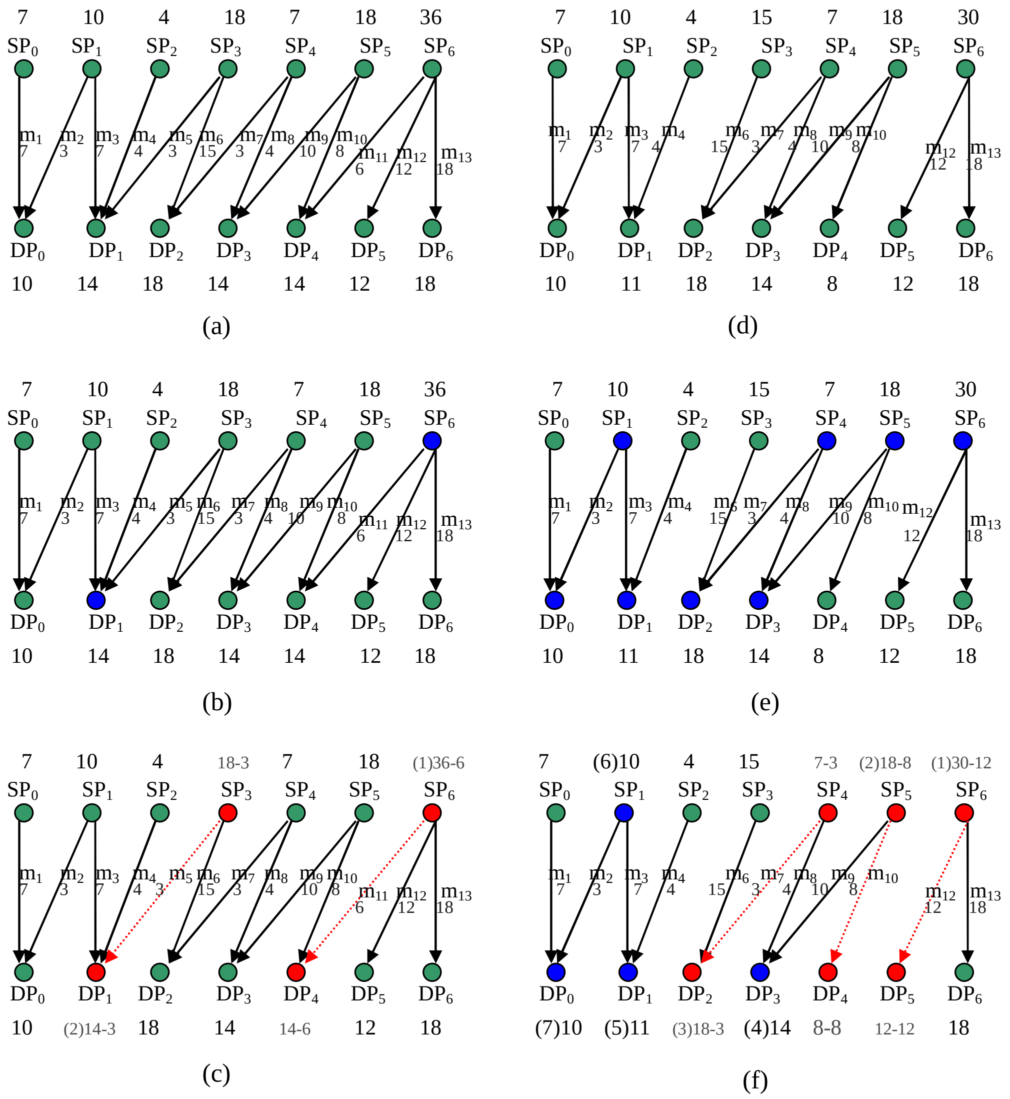
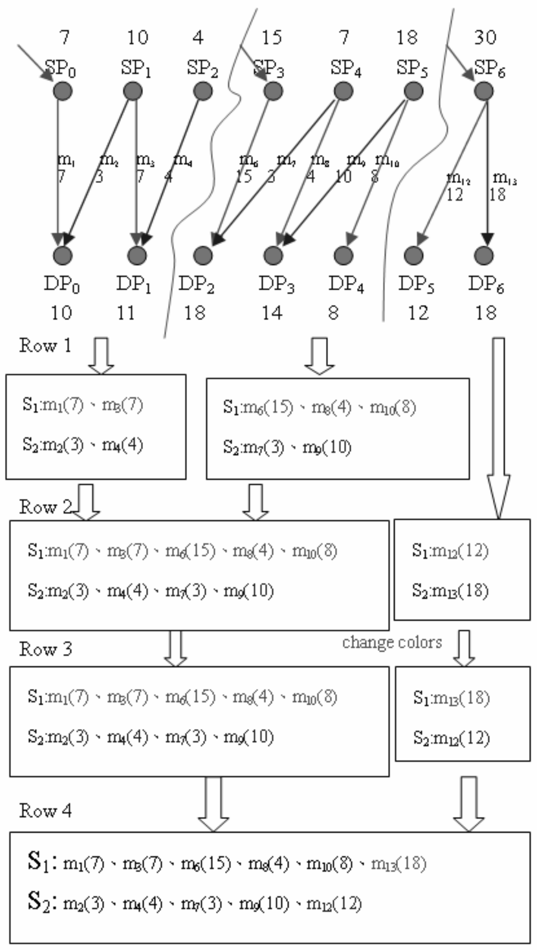

7 10 4 18 7 18 36
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7 m2 3 m3 7 m4 4 m5 3 m6 15 m7 3 m8 4 m9 10 m10 8 m11 6 m12 12 m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
10 14 18 14 14 12 18
(a)
7 10 4 15 7 18 30
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7 m2 3 m3 7 m4 4 m6 15 m7 3 m8 4 m9 10 m10 8 m12 12 m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
10 11 18 14 8 12 18
(d)
7 10 4 18 7 18 36
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7 m2 3 m3 7 m4 4 m5 3 m6 15 m7 3 m8 4 m9 10 m10 8 m11 6 m12 12 m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
10 14 18 14 14 12 18
(b)
7 10 4 15 7 18 30
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7 m2 3 m3 7 m4 4 m6 15 m7 3 m8 4 m9 10 m10 8 m12 12 m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
10 11 18 14 8 12 18
(e)
7 10 4 18-3 7 18 (1)36-6
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7 m2 3 m3 7 m4 4 m5 3 m6 15 m7 3 m8 4 m9 10 m10 8 m11 6 m12 12 m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
10 (2)14-3 18 14 14-6 12 18
(c)
7 (6)10 4 15 7-3 (2)18-8 (1)30-12
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7 m2 3 m3 7 m4 4 m6 15 m7 3 m8 4 m9 10 m10 8 m12 12 m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
(7)10 (5)11 (3)18-3 (4)14 8-8 12-12 18
(f)

Figure 2: The process of degree reduction (a) A bipartition graph showing communications between source processors and destination processors. (b) SP_6 and DP_1 have the maximum degree 3, they are marked blue. (c) m_{11} and m_5 are scheduled. Total message size of adjacent nodes of edges m_{11} and m_5 (SP_6, DP_4, DP_1 and SP_3) should be updated. (d) m_{11} and m_5 are removed from the bipartite graph. The maximum degree is 2 after degree reduction. (e) SP_6, SP_5, SP_4, SP_1, DP_3, DP_2, DP_1 and DP_0 have the maximum degree 2, they are marked blue. (f) m_{12}, m_{10}, m_7, m_2 and m_4 are scheduled. Adjacent nodes of edges m_{12}, m_{10}, m_7, m_2 and m_4 (SP_6, DP_5, SP_5, DP_4, DP_2, SP_4,…) should be updated. After remove messages m_7 and m_{10}, the degree of DP_3 can't be reduced.

To avoid the above situation, an adjustable coloring mechanism to schedule degree-2 and degree-1 communications in bipartite graph can be applied.

Since the consecutive edges must be scheduled into two steps, there is no need to care about the size of messages. That means we don't have to schedule the large messages together on purpose.



7 10 4 15 7 18 30
SP0 SP1 SP2 SP3 SP4 SP5 SP6
m1 7
m2 3
m3 7
m4 4
m6 15
m7 3
m8 4
m9 10
m10 8
m12 12
m13 18
DP0 DP1 DP2 DP3 DP4 DP5 DP6
10 11 18 14 8 12 18
Row 1
S1:m1(7)、m3(7)
S2:m2(3)、m4(4)
S1:m6(15)、m8(4)、m10(8)
S2:m7(3)、m9(10)
Row 2
S1:m1(7)、m3(7)、m6(15)、m8(4)、m10(8)
S2:m2(3)、m4(4)、m7(3)、m9(10)
S1:m12(12)
S2:m13(18)
Row 3
change colors
S1:m1(7)、m3(7)、m6(15)、m8(4)、m10(8)
S2:m2(3)、m4(4)、m7(3)、m9(10)
S1:m13(18)
S2:m12(12)
Row 4
S1: m1(7)、m3(7)、m6(15)、m8(4)、m10(8)、m13(18)
S2: m2(3)、m4(4)、m7(3)、m9(10)、m12(12)

Figure 3: Adjustable coloring mechanism for scheduling degree-2 and degree 1 communications. The bipartite graph has three *CS*s. Row 1 is the schedule tables of CS_1 and CS_2. Row 2 is the schedule table consisted of CS_1 and CS_2, and a schedule table of CS_3. Row 3 is the same as Row 2 but the schedule table of CS_3. The messages m_{12} and m_{13} are exchanged. Row 4 is the schedule table consisted of CS_1、CS_2 and CS_3.

Let's consider again the example in Figure 2(e). Figure 3 demonstrates scheduling of the coloring phase for communication steps 1 and 2. To facilitate our illustration, we denote each connected component in G' as a *Consecutive Section* (*CS*). In Figure 3, there are three *Consecutive Sections*, the CS_1 is consisted of four messages m_1, m_2, m_3 and m_4; the CS_2 is consisted of five messages m_6, m_7, m_8, m_9 and m_{10}; the CS_3 is consisted of two messages m_{12} and m_{13}. A simple coloring scheme is to use two colors on adjacency edges alternatively. For example, we first color m_1, m_6 and m_{12} red; then, color m_2, m_7 and m_{13} blue; and so on. The scheduling results for CS_1 and CS_2 are given as shown in Row 1. Row 2 shows the merging result of CS_1 with CS_2 and the schedule of CS_3. In Row 2, messages m_6 (15) and m_{13} (18) dominate the communication time at steps 1 and 2, respectively. This results total communication cost = 33. If we change the order of steps 1 and 2 in CS_3, it becomes m_{13} dominates the communication time in step 1 and m_{12} dominates the communication time in step 2. This will result total communication cost = 30. Therefore, the colors of two steps in CS_3 are exchanged in Row 3 for less communication cost. Row 4 shows communication scheduling of the adjustable coloring phase for degree-2 and degree-1 communications. The complete scheduling for this example is shown in Figure 4.

S_1: m1(7) 、 m3(7) 、 m6(15) 、 m8(4) 、 m10(8) 、 m13(18)

S_2: m2(3) 、 m4(4) 、 m7(3) 、 m9(10) 、 m12(12)

S_3: m11(6) 、 m5(3)

Figure 4: The scheduling of communications for the example in Figure 2.

The main idea of the two-phase degree reduction technique is to reduce degree of nodes by one for those nodes that have maximum degree at each reduction iteration. The following lemma states this property.

Lemma 1: Given a bipartite graph $G = (V, E)$ denotes the communications of irregular array redistribution. Let d be the maximum degree of vertices v, for all $v \in V$, a bipartite graph G' that with maximum degree 2 will be resulted after performing d-2 times degree-reduction iterations.

Proof Sketch:

We use an example of degree 3 to demonstrate the above statement. For the cases of bipartite graphs with degree larger than 3, the proof can be established in a similar way.

According to the characteristic of communications in irregular array redistribution we noticed that two communication links in a bipartite graph will not intersect with each other. Figure 5 shows a typical example of bipartite graph with maximum degree node is 3. There are three contiguous nodes have maximum degree 3, SP_{i-1}, SP_i and SP_{i+1}. SP_i has out-degree 3, i.e., it has three outgoing messages, m_{k-1}, m_k and m_{k+1} for destination processors DP_{j-1}, DP_j and DP_{j+1}, respectively. The worst case for SP_i is to incur communication conflicts with its neighboring nodes and induces itself can not schedule any communication message at the same communication step. For this scenario, we assume that if m_{k-2} and m_{k+2} are schedule at communication step 3 (the current maximum degree is 3), this will result messages m_{k-1} and m_{k+1} can not be scheduled in the same communication step (because they have common destination processor). Even so, the degree reduction operation will select m_k of SP_i into step 3.

Enlarge the maximum degree = $d > 2$, i.e., SP_i will send messages to DP_j, DP_{j+1}…, DP_{j+d-2} and DP_{j+d-1}. Because no two communication links would be intersected, SP_i has at most two communications go into conflicts with other processors (DP_j and DP_{j+d-1}). Therefore, the *TPDR* algorithm can schedule one of the remaining d-2 messages of SP_i into the same communication step. Therefore, we conclude that the *TPDR* algorithm reduces node degree by one in each degree reduction iteration. That is, a bipartite graph G' with maximum degree 2 will be resulted after performing d-2 times degree-reduction iterations. ■

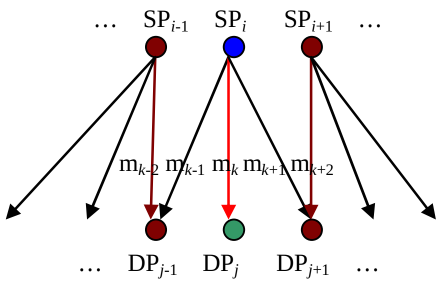


Figure 5: Contiguous nodes have the same maximum degree.

The algorithm of the two-phase degree reduction method is given as follows.

```
Algorithm two_phase_degree_reduction (P, GEN_BLOCK)
1.    generating messages;
2.    while ( messages != null)
3.    {
4.        step = d = maximal degree;
5.        while ( step > 2 )
6.        {
7.            find_mark_blue(d);
              // marking degree-d nodes blue
8.            sort_marked_blue_node();
              // sorting blue nodes in decreasing order by message size
9.            while(marked node !=null)
10.           {
11.               choose_marked_node();
                  // selecting a marked node, assume v
12.               pick_minimal_message();
```

```
                // schedule the minimum message (assume e_ij) of v into step d
13.             mark_red();
                // mark adjacent vertices of edge e_ij as red to avoid conflict communication
14.             update schedule(step, message );
15.         }
16.         step--;
17.     }
18.     coloring_message();
        // color the remaining messages with maximal degree is 2
19.     merge_Schedule();
        // merge schedules of consecutive sections into communication steps 1 and 2
20.     update schedule(step, message);
21. }
end_of_two_phase_degree_reduction
```

4.2 Extended TPDR

Based on *TPDR*, we present an extended two-phase degree reduction (*E-TPDR*) algorithm. An edge-complement operation is added in the degree-reduction phase. As the *TPDR* algorithm stated, the original degree-reduction operation only schedules degree-*k* nodes' messages into communication step *k*. This might not fully utilize the available space in step *k* and remains heavy communications in the previous steps (less than *k*). Therefore, a principle for adding extra messages into these steps is to select the maximum message that is smaller than the length of current step and with un-marked adjacent vertices.

The key concept of this modification is to schedule messages into communication steps during reduction phase as many as possible. Because the additional scheduled messages are with smaller message size than the current step length, the edge-complement operation will not influence the cost of original scheduling from *TPDR*. Figure 6 shows the communication schedule of the example given in Figure 2 using *E-TPDR*. Although this example does not reflect lower total cost of *E-TPDR*, section 5 will demonstrate the improvement of *E-TPDR* method from the simulation results.

S_1: m_1(7), m_3(7), m_6(15), m_9(10), m_{13}(18)

S_2: m_4(4), m_7(3), m_{10}(8), m_{12}(12)

S_3: m_{11}(6), m_5(3), m_8(4), m_2(3)

Figure 6: The *E-TPDR* scheduling of communications for the example in Figure 2.

The algorithm of the extended two-phase degree reduction method is given as follows.

```
Algorithm E_TPDR (P, GEN_BLOCK)
1.  generating messages;
2.  while ( messages != null)
3.  {
4.      step = d = maximal degree;
5.      while (step > 2 )
6.      {
7.          find_mark_blue(d);
            // marking degree-d nodes blue
8.          sort_marked_blue_node();
            // sorting blue nodes in decreasing order by message size
9.          while(marked node !=null)
10.         {
11.             choose_marked_node();
                // selecting a marked node, assume v
12.             pick_minimal_message();
                // schedule the minimum message (assume e_ij) of v into step d
13.             mark_red();
                // mark adjacent vertices of edge e_ij as red to avoid conflict communication
14.             update schedule(step, message );
15.             step_length(d);
                // determine the length L of current step
16.             sort_small_message();
                // sort all messages with unmarked adjacent vertices and message size is smaller than L
17.             pick_message();
                // schedule the selected messages into current step if there is no contention
18.             mark_red();
                // mark adjacent vertices of the scheduled edges as red
19.             update schedule( step, message );
20.             }
21.             step--;
22.         }
23.         coloring_message();
            // color the remaining messages with maximal degree is 2
24.         merge_Schedule();
            // merge schedules of consecutive sections into communication steps 1 and 2
25.         update schedule(step, message);
26.    }
end_of_ E_TPDR
```

4.3 Algorithmic Complexity Analysis

Definition 3: Given a bipartite graph *G* to represent the communications of an irregular array redistribution, *G'* and |*E'*| are denoted as the graph and the number of edges in *G'* after edges are removed in degree reduction phase by *TPDR*, respectively.

TPDR algorithm can be divided into three stages. The first part is to schedule communication messages of processors who's degree is greater than 2. Because the scheduling in this stage is size-oriented, the time complexity of this stage is O(*dP*log*P*), where *d* is the maximum degree of processors. The second stage is to schedule remaining messages after the degree reduction phase using the coloring scheme. Since we need to color all remaining edges once, according to definition 3, the time complexity in this stage is O(|*E'*|). The last stage is to combine scheduled tables of consecutive sections. This can be done in O(*logS*) via the winner tree data structure, where *S* is the number of consecutive sections.

For the *E-TPDR* algorithm, the time complexity of the first stage in worst case is O(*dE*log*E*), which is different from *E-TPDR* algorithm. The other stages have the same complexities as *TPDR* algorithm.

5. Performance Evaluation

To evaluate the performance of the proposed methods, we have implemented the *TPDR* and *E-TPDR* along with the divide-and-conquer algorithm [19]. The performance simulation is discussed in two classes, even GEN_BLOCK and uneven GEN_BLOCK distributions. In even GEN_BLOCK distribution, each processor owns similar size of data. The communication cost will not be dominated by specific processor, because the size of messages between processors could be very close. Contrast to even distribution, few processors might be allocated grand volume of data in uneven distribution. Since array elements could be centralized to some specific processors, it is also possible for those processors to have the maximum degree of communications. Therefore, the communication cost will be dominated by these processors. To accomplish an optimal scheduling, it is obvious that even distribution case is more difficult than uneven distribution. This observation was comprehended by that communication cost could be determined by one processor that with maximum degree or maximum total message size in uneven distribution; consequently, it leads high probability to achieve a schedule that has the same cost as the processor's total message size.

To determine the redistribution is on even GEN_BLOCK or uneven GEN_BLOCK, we define upper and lower bounds of data size in GEN_BLOCK distribution. Given an irregular array redistribution on *A*[1:*N*] over *P* processors, the average block size will be *N*/*P*. In even distribution, the range of upper and lower bounds is set to ±30%. Thus, size of data blocks could be 130% *N*/*P* ~ 70% *N*/*P*. In uneven distribution, the range of upper and lower bounds is set to ±100%. Thus, size of data blocks could be 200% *N*/*P* ~ 1.

5.1 Simulation A

Simulation A is carried out to examine the performance of *TPDR* and *E-TPDR* algorithms on uneven cases. We use a random generator to generate 10,000 test data sets.

Figure 7 shows the comparisons of *TPDR* algorithm and the *divide-and-conquer* (*DC*) algorithm. We run tests by different processor numbers from 4 to 24. In 10,000 samples, the number of cases of *TPDR* better than *DC*, *DC* better than *TPDR* and the same are counted. When the number of processors is 4, there are lots of cases both algorithms have the same result. This is because that the size of messages could be larger when number of processors is less. It's easier to derive schedules that have minimum size of total communication steps. When the number of processors becomes numerous, the *TPDR* provides significant improvements generally. This phenomenon can be explained by the statistic of degree-2 nodes for 10,000 test samples as shown in Figure 8. When number of processors is large, size of data blocks in these processors are relative small. Therefore, processors have lower possibility to have high degree of communication links. In other words, the number of degree-2 nodes increases largely. Since the *TPDR* uses an optimal adjustable coloring mechanism for scheduling degree-2 and degree-1 communications, therefore, we expect that *TPDR* performs better when the number of degree-2 nodes is large. This observation also matches the simulation results in Figures 7 and 9.

Figure 9 gives the comparisons of the *E-TPDR* algorithm and the divide-and-conquer algorithm. When the number of processors is 4, there are about 60% cases has the same result by both algorithms. Similar to the previous observations, the *E-TPDR* performs well when number of processors is numerous.

5.2 Simulation B

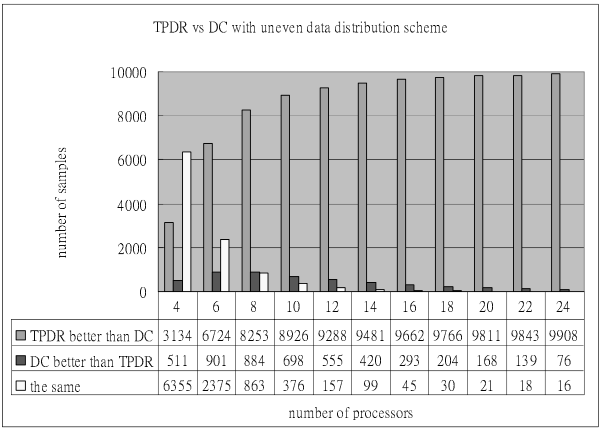
Simulation B is carried out to examine the performance of *TPDR* and *E-TPDR* algorithms on even cases. We also use the random generator to produce 10,000 data sets for this test.

Figures 10 and 11 show the performance comparisons of *TPDR* and *DC*, *E-TPDR* and *DC*, respectively. Overall speaking, we have similar observations as those described in Figures 7 and 9. The *E-TPDR* performs better than *TPDR*. When number of processors is large, the *TPDR* and *E-TPDR* both provide significant improvements. Compare to the results in uneven cases (simulation A), the ratio of our algorithms outperform the *DC* algorithm become lower. We move our focus to Figure 12 to explain this phenomenon. Figure 12 gives the statistics of different degree nodes for even distribution test data. We observed that there is no nodes with degree higher than 4. In other words, the maximum degree of nodes of these 10,000 test samples is 3. On this aspect, the *DC* algorithm and the *TPDR* methods have more cases that are the same. This is also why the *TPDR* and *E-TPDR* have better ratio from 99% to 93%.

From the above performance analysis and simulation results, we have the following remarks.

Remark 1: The *TPDR* and *E-TPDR* algorithms perform better in uneven cases than in even cases.

Remark 2: The *TPDR* and *E-TPDR* algorithms perform well when number of processors is large.



	4	6	8	10	12	14	16	18	20	22	24
▨ TPDR better than DC	3134	6724	8253	8926	9288	9481	9662	9766	9811	9843	9908
■ DC better than TPDR	511	901	884	698	555	420	293	204	168	139	76
□ the same	6355	2375	863	376	157	99	45	30	21	18	16

Figure 7: The *E-TPDR* scheduling of communications for the example in Figure 2.

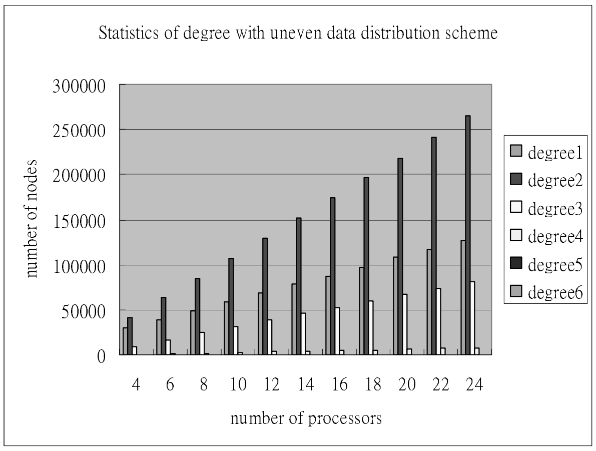
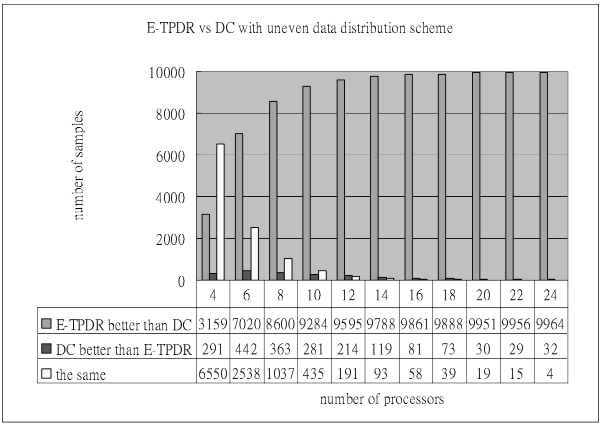
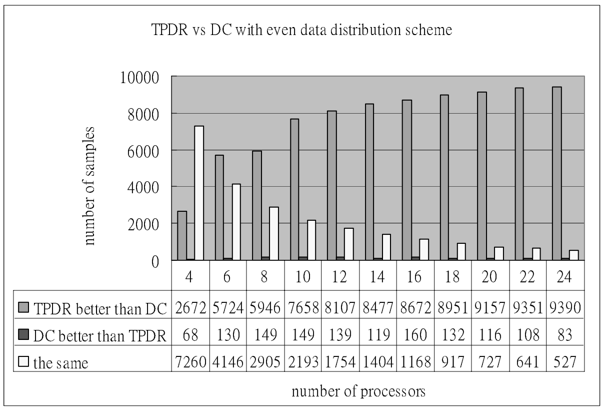


Figure 8: Statistic of degree-2 nodes in 10,000 uneven `GEN_BLOCK` test samples.



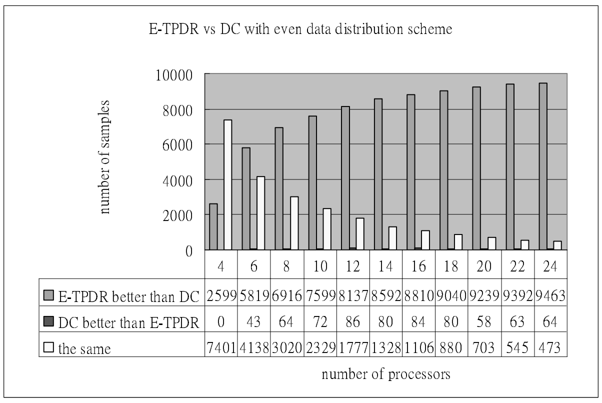
	4	6	8	10	12	14	16	18	20	22	24
▨ E-TPDR better than DC	3159	7020	8600	9284	9595	9788	9861	9888	9951	9956	9964
■ DC better than E-TPDR	291	442	363	281	214	119	81	73	30	29	32
□ the same	6550	2538	1037	435	191	93	58	39	19	15	4

Figure 9: Comparison of the *E-TPDR* method and the *divide-and-conquer* algorithm on 10,000 uneven data sets (1 MB).



	4	6	8	10	12	14	16	18	20	22	24
▨ TPDR better than DC	2672	5724	5946	7658	8107	8477	8672	8951	9157	9351	9390
■ DC better than TPDR	68	130	149	149	139	119	160	132	116	108	83
□ the same	7260	4146	2905	2193	1754	1404	1168	917	727	641	527

Figure 10: Comparison of the *TPDR* method and the *divide-and-conquer* algorithm on 10,000 even `GEN_BLOCK` test samples (1 MB).



	4	6	8	10	12	14	16	18	20	22	24
▨ E-TPDR better than DC	2599	5819	6916	7599	8137	8592	8810	9040	9239	9392	9463
■ DC better than E-TPDR	0	43	64	72	86	80	84	80	58	63	64
□ the same	7401	4138	3020	2329	1777	1328	1106	880	703	545	473

Figure 11: Comparison of the *E-TPDR* method and the *divide-and-conquer* algorithm on 10,000 even `GEN_BLOCK` test samples (1 MB).

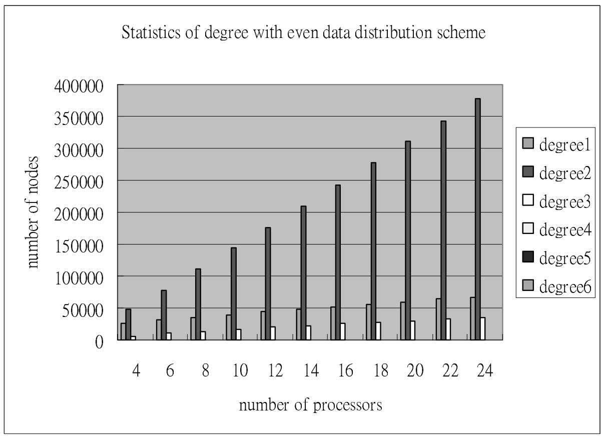


Figure 12: Statistic of degree-2 nodes in 10,000 even GEN_BLOCK test samples.

6. Conclusions

In this paper, we have presented a *two-phase degree-reduction* (*TPDR*) scheduling technique to efficiently perform HPF2 irregular array redistribution on distributed memory multi-computer. The *TPDR* is a simple method with low algorithmic complexity to perform GEN_BLOCK array redistribution. An extended algorithm based on *TPDR* is also presented. Effectiveness of the proposed methods not only avoids node contention but also shortens the overall communication length. The simulation results show improvement of communication costs and high practicability on different processor hierarchy.

In HPF, it supports array redistribution with arbitrary source and destination processor sets. The technique developed in this paper assumes that the source and the destination processor sets are the same. In the future, we will study efficient methods for array redistribution with arbitrary source and destination processor sets. Besides, the issues of scheduling irregular problems on grid system and considering network communication latency in heterogeneous environments are also interesting and will be investigated. Also, we will also study realistic applications and analyze their performance.

References

[1] G. Bandera and E.L. Zapata, “Sparse Matrix Block-Cyclic Redistribution,” *Proceeding of IEEE Int'l. Parallel Processing Symposium* (IPPS'99), San Juan, Puerto Rico, April 1999.

[2] Frederic Desprez, Jack Dongarra and Antoine Petitet, “Scheduling Block-Cyclic Data redistribution,” *IEEE Trans. on PDS*, vol. 9, no. 2, pp. 192-205, Feb. 1998.

[3] C.-H Hsu, S.-W Bai, Y.-C Chung and C.-S Yang, “A Generalized Basic-Cycle Calculation Method for Efficient Array Redistribution,” *IEEE TPDS*, vol. 11, no. 12, pp. 1201-1216, Dec. 2000.

[4] C.-H Hsu, Dong-Lin Yang, Yeh-Ching Chung and Chyi-Ren Dow, “A Generalized Processor Mapping Technique for Array Redistribution,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 12, vol. 7, pp. 743-757, July 2001.

[5] Minyi Guo, “Communication Generation for Irregular Codes,” *The Journal of Supercomputing*, vol. 25, no. 3, pp. 199-214, 2003.

[6] Minyi Guo and I. Nakata, “A Framework for Efficient Array Redistribution on Distributed Memory Multicomputers,” *The Journal of Supercomputing*, vol. 20, no. 3, pp. 243-265, 2001.

[7] Minyi Guo, I. Nakata and Y. Yamashita, “Contention-Free Communication Scheduling for Array Redistribution,” *Parallel Computing*, vol. 26, no.8, pp. 1325-1343, 2000.

[8] Minyi Guo, I. Nakata and Y. Yamashita, “An Efficient Data Distribution Technique for Distributed Memory Parallel Computers,” *JSPP'97*, pp.189-196, 1997.

[9] Minyi Guo, Yi Pan and Zhen Liu, “Symbolic Communication Set Generation for Irregular Parallel Applications,” *The Journal of Supercomputing*, vol. 25, pp. 199-214, 2003.

[10] Edgar T. Kalns, and Lionel M. Ni, “Processor Mapping Technique Toward Efficient Data Redistribution,” *IEEE Trans. on PDS*, vol. 6, no. 12, December 1995.

[11] S. D. Kaushik, C. H. Huang, J. Ramanujam and P. Sadayappan, “Multiphase data redistribution: Modeling and evaluation,” *Proceeding of IPPS*’95, pp. 441-445, 1995.

[12] S. Lee, H. Yook, M. Koo and M. Park, “Processor reordering algorithms toward efficient GEN_BLOCK redistribution,” *Proceedings of the ACM symposium on Applied computing*, 2001.

[13] Y. W. Lim, Prashanth B. Bhat and Viktor and K. Prasanna, “Efficient Algorithms for Block-Cyclic Redistribution of Arrays,” *Algorithmica*, vol. 24, no. 3-4, pp. 298-330, 1999.

[14] Neungsoo Park, Viktor K. Prasanna and Cauligi S. Raghavendra, “Efficient Algorithms for Block-Cyclic Data

redistribution Between Processor Sets," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, No. 12, pp.1217-1240, Dec. 1999.

[15] Antoine P. Petitet and Jack J. Dongarra, "Algorithmic Redistribution Methods for Block-Cyclic Decompositions," *IEEE Trans. on PDS*, vol. 10, no. 12, pp. 1201-1216, Dec. 1999.

[16] L. Prylli and B. Touranchean, "Fast runtime block cyclic data redistribution on multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 45, pp. 63-72, Aug. 1997.

[17] S. Ramaswamy, B. Simons, and P. Banerjee, "Optimization for Efficient Data redistribution on Distributed Memory Multicomputers," *Journal of Parallel and Distributed Computing*, vol. 38, pp. 217-228, 1996.

[18] Akiyoshi Wakatani and Michael Wolfe, "Optimization of Data redistribution for Distributed Memory Multicomputers," short communication, *Parallel Computing*, vol. 21, no. 9, pp. 1485-1490, September 1995.

[19] Hui Wang, Minyi Guo and Daming Wei, "Divide-and-conquer Algorithm for Irregular Redistributions in Parallelizing Compilers", *The Journal of Supercomputing*, vol. 29, no. 2, 2004.

[20] Hui Wang, Minyi Guo and Wenxi Chen, "An Efficient Algorithm for Irregular Redistribution in Parallelizing Compilers," *Proceedings of 2003 International Symposium on Parallel and Distributed Processing with Applications*, LNCS 2745, 2003.

[21] H.-G. Yook and Myung-Soon Park, "Scheduling GEN_BLOCK Array Redistribution," *Proceedings of the IASTED International Conference Parallel and Distributed Computing and Systems*, November, 1999.