プロセッサ間通信に特化したGUIプログラミング環境

Programming Environment Specified for Interprocessor Communications Based on Graphical User Interface

酒寄 保隆[†]
Yasutaka Sakayori

三浦 元喜[†]
Motoki Miura

田中 二郎[††]
Jiro Tanaka

†筑波大学大学院博士課程工学研究科
Doctoral Program in Engineering,
University of Tsukuba

††筑波大学電子・情報工学系
Institute of Information Science and Electronics,
University of Tsukuba

概 要

本論文では，メッセージパッシングパラダイムに基づく並列プログラムで必要になる，プロセッサ間通信を記述するためのプログラミング環境であるビジュアルプログラミングシステムGRIXの概要と，そのプロセッサ間通信記述に特化されたGUIベースの操作法について述べる．本論文で述べられる操作法により，GRIXの入力はよりプロセッサ間通信を直観的に入力することが可能になるとともに，その入力によって生成されたプログラムは，小規模並列実行環境から超並列計算機等のあらゆる環境に対応することも可能となる．

1 序　論

我々は，メッセージパッシングのパラダイムの基での並列プログラム中で必要となるプロセッサ間通信を効率的に実行可能にし，かつ複雑になりがちなプロセッサ間通信のテキスト表現をできる限り避けて入力することができるシステムとして，ビジュアルプログラミングシステム*GRIX* [1] [2]を提案している．

プロセッサ間通信は，並列プログラムを構築する際に最もその記述に注意が必要となる要素である．並列計算機のピークパフォーマンス (1台当たりの*PE*(Processing Element) の処理能力 × PEの台数) をロスさせるのは，各PE間での同期処理を引き起こすプロセッサ間通信が大きな原因である．効率の悪いコーディングでは頻繁に同期が引き起こされ，折角の計算機の能力も発揮できないことになる．またプロセッサ間通信を記述する手段は，*PVM* [3]や*MPI* [4]などの並列実行環境の仕様にそったものや，各々の計算機に用意されたネイティブな通信手段など多種多様に存在する．その上ハイパフォーマンスを求める性格からも，その仕様は複雑になりがちである．このようなプロセッサ間通信の記述に関する，仕様の多種多様性・複雑性はユーザにとって歓迎されるものではない．

例としてPVMを用いたプログラムでは，最も単純な送受信でさえデータの格納 (`pvm_pkint` 等)，送信 (`pvm_send`)，受信 (`pvm_recv`)，取り出し (`pvm_upkint` 等) の手順を踏まなくてはならない．またそれぞれの関数に必要な引数に関しても，その順番や内容など複雑なものが多い．この傾向はネイティブな関数の仕様になるほど強くなる．

我々はこの点からくるプログラマの不幸を憂い，以下の主張をする．

- 引数の何番目には何を書くとか，通信するためには前もって何をするなどといった手続きは，ユーザにとって冗長な情報である．
- 「誰が」「どこに」「何を」通信するという情報のみでプロセッサ間通信の実現が理想的である．

我々はこの主張に対する有効な解の1つとして，ビジュアルプログラミングによるプロセッサ間通信の実現を見出した．ここでいうビジュアルプログラミングとは，テキスト表現ではなく図やアニメーションといった人間にとってより直観的な表現を用いてプログラミングを行う方法である．ビジュアルな表現でプロセッサ間通信を直観的に記述でき，またハイパフォーマンスを維持する．これがGRIXの起因であり目的である．

2 ビジュアルプログラミングシステムGRIX

GRIXが使われるのは，SPMD型のプログラムを作成している段階で，プロセッサ間通信のコーディングが必要

とされる場面である．GRIX システムは以下に示すようなコンセプトで, その処理系の構築を行った．

- GUI(Graphical User Interface) に基づいた入力
- ポータビリティの高い最適化処理
- 最適化結果の GUI 出力
- 任意の実行環境に対するコード生成

GUI に基づいた入力機構を用意することにより，ユーザは直観的な入力が可能になる．その入力情報を, あらゆる環境に対応できるポータビリティの高い最適化を施すことにより, 任意の実行環境に対応する自動コード生成が可能となる．これは将来的にどんな環境が現れても, それ用の処理系を従来のものに加えるだけで, 対応が可能になる．そして最適化結果も GUI 出力することにより，ユーザに対する最適化結果と生成されるコードの直観的な理解, 及びデバッグの際のサポートも可能になる．

図 1　内部処理系の関係

以上のようなコンセプトの基でシステムを設計したことにより，GRIX の内部処理系は，その処理の特徴から, GUI エンジン，最適化エンジン，コード生成エンジンの 3 つのサブシステムに大分される (図 1)．GUI エンジンは，ユーザの入力によって得られた情報をファイル化し, 最適化エンジンに渡す．最適化エンジンはそのファイルから得られた情報を基に, 通信時間の短縮と自動コード生成のための最適化を施す．そして最適化によって変更された情報を示す，GUI 出力用とコード生成用のファイルを生成する．GUI エンジンは, そのファイルから得られた最適化結果を GUI 表示する．この最適化結果の GUI 表示により, ユーザに対する直観的な理解を助け, 生成されるコードの実行状況の理解やデバッグなどの補助をする．またコード生成用のファイルは, コード生成エンジン内部に複数の環境それぞれに対応する変換器を用意することにより, 各々の環境に対応したコードを自動生成する．このコード生成用のファイルを保存しておくことにより, 将来新しい環境用のコード生成エンジンを加えた際に, 従来の環境用のコードを速やかに新しい環境用に変換することができる．

3　プロセッサ間通信に特化したインターフェイス

本論文では，GRIX の大きな特徴である，プロセッサ間通信に特化された GUI インターフェイスについて述べる．

我々は，このビジュアルインターフェイスを設計するに当り，入力の際のユーザの描くイメージを重要視した．最も基本的な, 1 次元のノード ID を前提とした入力例で, GRIX の基本ビジュアルインターフェイスについて説明する．

3.1　絶対的視点からの入力

プロセッサ間通信の送受信関係を入力する際，ユーザがイメージするその状況の一つに, 並列計算を行う全プロセッサを視点にいれたものが考えられる．つまり, 並列実行プロセッサ台数があらかじめ分かっていて, それを前提にプロセッサ間通信の入力を行っていく場合である．この様な状況下での入力を, 並列計算をする計算機, または計算機の集合全体を入力の単位とし, 各々のプロセッサに割り振られるノード ID を固定値で用いることが可能であるため, **絶対的視点からの入力** (図 2) と呼ぶことにする．

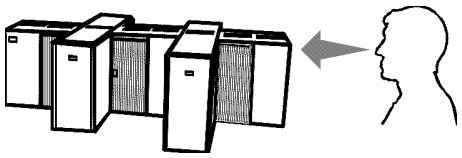

図 2　絶対的視点

まずユーザがシステム側に表示するノードの数を通知すると, システムは図 3左に示す初期画面を表示する．この図の中の円はプロセッサを表すノードで, 縦に並ぶノード群はプロセッサの集合, 同じ列の横に並ぶノードは同じプロセッサを示し, 左側が送信, 右側が受信ノードとなる．送信側のノードの脇に表示された番号は, 0 から始められたプロセッサの相対ノード番号を示している．ユーザはこの画面上に, 送信ノードから受信ノードへマウスドラッグする操作で矢印を描いていく．この操作は, 入力画面上に矢印を描く最も基本的な動作であり, ユーザはどんなプロセッサ間通信の関係でも, この操作で自然に記述できる (図 3右)．

マウスドラッグによる矢印の描画のみでも，送受信関係の入力は可能であるが, GRIX には，しばしば並列プログラミング中に現れる通信パターンを記述しやすくするために，またその他の状況にも簡単に対処できるように, 様々な入力用のオプションが用意されている．Broadcast や Scatter などの 1 対多の通信の入力を例として挙げると．まずユーザは図 4左上の様に，1 つの送信ノードと複数の受信ノードをマウスの範囲選択等を用いて選択する．そしてメニューからの操作 (図 4右), もしくはショートカットキーを押すことにより, 図 4左下の様に描画される．この様なオプションを用いることにより, 1 本 1 本矢

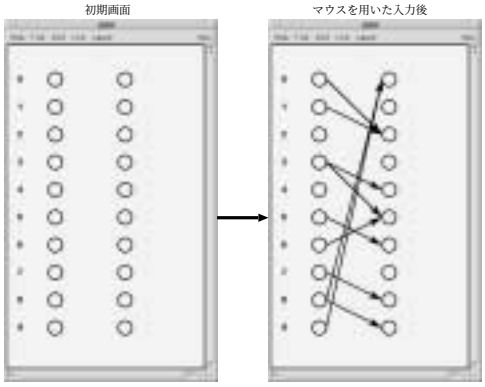

図 3　絶対的視点からの入力

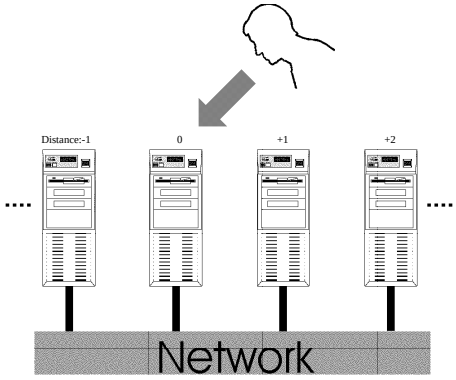

図 5　相対的視点

印を描いていく必要がなくなり, すばやい入力が可能となる.

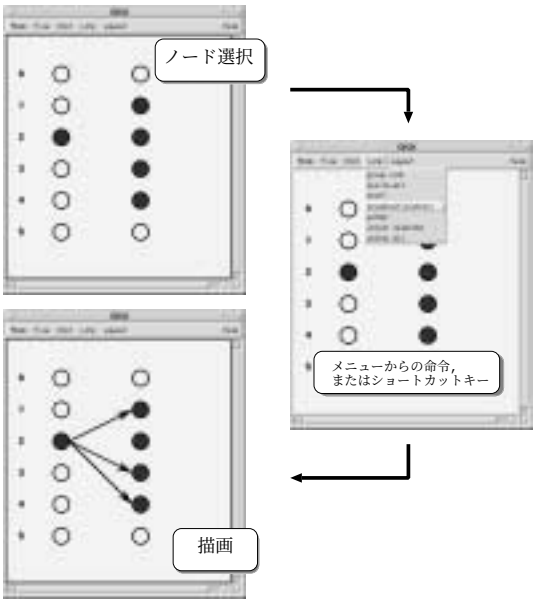

図 4　入力オプションの例

3.2　相対的視点からの入力

プロセッサ間通信をユーザがイメージする際には, 絶対的視点のようなイメージではなく, 全プロセッサの中の一つに注目した視点で考える状況がある. 例えば, 「ID が偶数のプロセッサは -2,+1,+3 のプロセッサにデータを送信する」というような Scatter のイメージをユーザが持った場合である. この状況下での入力を, 絶対的視点に対し, 並列計算をする計算機の集合の 1 つのプロセッサに着目し, 他のプロセッサとの関係は相対値によって表現されることから, **相対的視点の入力** (図 5) と呼ぶことにする.

この例の入力の手順を図 6に示す. まずユーザは, ユーザが告知した数のノードが表示された初期画面 (図 6左上) の中から, 1 つの視点の中心となるノードを選択する. この中心のノードを **Owner** と呼ぶことにする. ユーザが Owner を選択すると, システムは Owner を強調色で表現し, 送信側のノードの脇に Owner からの各々への相対距離を表示する (図 6右上). この相対距離の表示は, ユーザが Owner を選び直す度に, それにあった距離を示す値に書き直される. この様に入力画面上に Owner を選択した後に, 絶対的視点からの入力と同様にユーザは入力を行う (図 6左下). ではどの Owner がアクティブであるかを入力するにはどうすればよいか. この例の場合「ID が偶数のプロセッサ」がアクティブである. アクティブノードを定義する際には, メニューの「`Edit → define active nodes`」を選択することから始まる. ユーザはここから「`even`」を選ぶことにより, この入力が完了する (図 6右下). システムにはデフォルトで`all`(全てアクティブ), `1 node`(ある 1 つのノード), `even`(ID が偶数), `odd`(ID が奇数) の 4 つが条件の選択肢として用意されているが, これら以外の条件を与えたい場合には「`customize`」を選択し, ユーザ自身がシステムが出すダイアログの中に, 自ノード ID を示す変数「`id`, または `ID`」と四則演算子, 論理演算子, 及び定数を用いた条件式で新たに定義することになる (ex. `((id /2) % 2) == 0`). なおアクティブノードの定義は, 通信パターンの入力の前後どちらでも構わない. またこの操作を行わないと, システムは全てがアクティブノードだとして処理を進める.

以上が Scatter の入力例だが, 逆に Gather を入力したい場合には, 最初の Owner 選択の際にユーザが受信側のノードを選択することで可能となる. つまりこの視点の入力では, ユーザが選択した Owner 以外のノードからの送信, または Owner 以外のノードへの受信が許可されない制約が発生する.

この様な入力の後, システムは通信衝突の回避などのスケジューリングを行い, その結果の実行コードと, 入力画

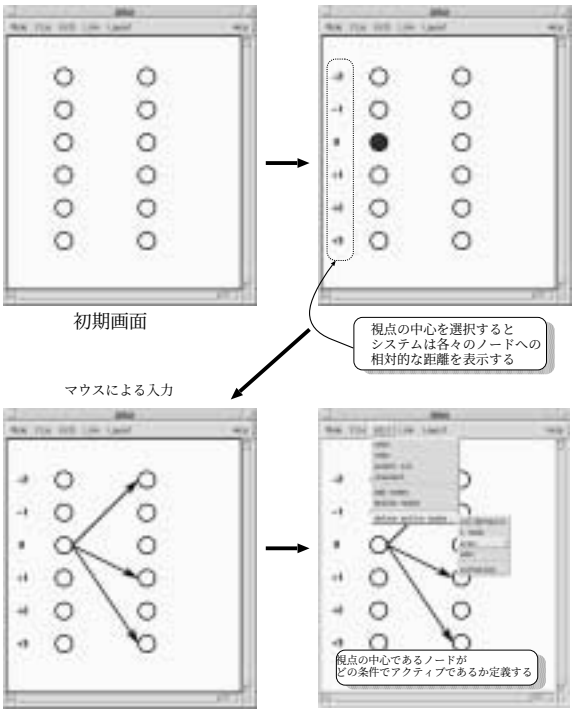

図 6　相対的視点からの入力手順

面と同様なビジュアルな出力を行う．入力後の処理については, 文献 [1] [2]を参照されたい.

4 関連研究

プロセッサ間通信記述に特化したことにより，GRIX は従来にない並列計算用の GUI ツールである.

他の並列計算用の GUI ツールには，まず P. Newton, J. C. Browne らの, *CODE* [5]というビジュアル並列プログラミング言語がある．また文献 [6]の中で, Newton は同様なビジュアル並列プログラミング言語である *HeNCE* [7]と CODE との比較を行っている．CODE, HeNCE とも, 並列プログラムを記述するという特徴では, GRIX と同様なビジュアルプログラミングシステムであるが, 我々はプログラム全体ではなく, 並列計算固有のプロセッサ間通信にビジュアルプログラミングを適用している．また CODE, HeNCE は，ビジュアルプログラミング全体に, 様々なアイコンを用いているのも特徴である.

また PVM プログラムを制御・可視化する，*XPVM* [8]という PVM 用のツールがある．XPVM はプロセッサ間通信をコントロールする点で, GRIX と共通の特徴を持つが, 我々は特定の環境に特化したシステムではなく, あらゆる環境に柔軟に対応できるシステムを目指している.

これ以外にも, 並列プログラム実行時の各プロセッサの負荷を図解的に表示・制御する, GUI を出力重視で用いている *ParaGraph* [9]などのパフォーマンスモニタがある.

5 結　論

プロセッサ間通信は, その記述に関する仕様が数多く存在し, また複雑なものが多い．このような状況下でよりスムーズなプログラミングを行う手段として, ビジュアルプログラミングを採用することは非常に有効である． そこで本論文では, ユーザフレンドリで, かつ高速なプロセッサ間通信を実現するための環境である, プロセッサ間通信に特化したビジュアルプログラミングシステム GRIX の概要を述べ, その特徴的なビジュアルインターフェイスについて説明した．GRIX は, そのプロセッサ間通信に特化されたビジュアルインターフェイスを有することにより, 直観的に各プロセッサ間の送受信を記述することができる．このことにより, ユーザは複雑で煩雑な通信処理記述の仕様を理解するといった, 煩わしい仕事から解放されることになる.

現在の GRIX システムはプロセッサ間通信の記述用に完全に独立した構成を持っている．今後, プロセッサ間通信以外のプログラム編集を行う既存のテキストエディタと連動させていくことで, よりインタラクティブなプログラミング環境を構築していくことが可能になる.

参考文献

[1] Yasutaka Sakayori, Motoki Miura and Jiro Tanaka: *GRIX: Visual Programming System for Interprocessor Communications*, Proceedings of the 10th IASTED International Conference PDCS '98, pp.503-508, Oct. 1998

[2] 酒寄保隆: プロセッサ間通信記述用のビジュアルプログラミングシステムの提案, 筑波大学大学院博士課程工学研究科修士論文, 1999 年 3 月

[3] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Mancheck and Vaidy Sunderam: *PVM*, The MIT Press, 1994

[4] William Gropp, Ewing Lusk and Anthony Skjellum: *USING MPI*, The MIT Press, 1994

[5] Peter Newton and James C. Browne: *The CODE 2.0 Graphical Parallel Programming Language*, Proceedings of ACM International Conference on Supercomputing, July, 1992

[6] P.Newton: *Visual Programming and Parallel Computing*, Delivered at Workshop on Environments and Tools for Parallel Scientific Computing, May, 1994.

[7] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek, and V. S. Sunderam: *Graphical Development Tools for Network-Based Concurrent Supercomputing*, Proceedings of Supercomputing 91, pp.435-444, 1991

[8] G. A. Geist, A. Beguelin, J. J. Dongarra, W. Jiang, R. Manchek, and V. S. Sunderam: *PVM 3 Users Guide and Reference Manual*, Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, 1993

[9] Michael T.Heath: *ParaGraph: A Tool for Visualizing Paformance of Parallel Programs*, Univ of Illinois, Jennifer Etheridge Finger, Oak Ridge National Laboratory, June 1994