様式Ｃ－１９

科学研究費助成事業（科学研究費補助金）研究成果報告書

平成２４年５月２５日現在

機関番号：87103
研究種目：基盤研究(B)
研究期間：平成２１年度～平成２３年度
課題番号：21300011
研究課題名（和文）
細粒度マルチスレッド処理原理による言語処理系および並列分散 OS 構成法の研究
研究課題名（英文）
Research on Fine-grain Multithreading Language and its Operating System
研究代表者
雨宮　真人（Makoto Amamiya）
財団法人九州先端科学技術研究所・生活支援情報技術研究室・特別研究員
研究者番号：　90202697

研究成果の概要（和文）：細粒度マルチスレッド処理原理による並列プログラミング技術を開発し，その有効性の検証を行なった．具体的には，(1)並列ストリーム処理に焦点を当てた言語処理系を開発し，多重ループの並列展開手法を開拓し，(2)商用マルチコアプロセッサおよびクラスタマシン上にランタイムシステムを開発し，(3)データ依存性が複雑に絡む多重ループ処理をベンチマークとして選び細粒度マルチスレッド処理方式による並列ストリーム展開効果を評価した．

研究成果の概要（英文）：We did the research on parallel programming language and its operating system on the basis of fine-grain multithreading principle. First, we developed the multithread-based parallelizing technique and applied it to the stream processing. Second, we developed the runtime system of the language on the multi-core processor and PC-cluster machine. And third, we showed the enough performance enhancement could be exploited by our approach, by evaluating the parallelized version of the nested loop programs with complicated data dependency structure, on the runtime system.

交付決定額

（金額単位：円）

	直接経費	間接経費	合　計
平成 21 年度	4700000	1410000	6110000
平成 22 年度	4300000	1290000	5590000
平成 23 年度	4600000	1380000	5980000
総　計	13600000	4080000	17680000

研究分野：総合領域
科研費の分科・細目：情報学・ソフトウェア
キーワード：並列コンピュータ，言語処理系，並列分散 OS，細粒度マルチスレッド，ストリーム処理

１．研究開始当初の背景

コンピュータのハードウェアに関して，大規模数値演算主体のスーパーコンピュータのみならず汎用のパーソナルコンピュータにおいても並列コンピュータが一般的という時代が到来しつつある．しかし，そのソフトウェアに関しては，日常的に使いこなされる技術にまで成熟しているとは言い難く，並列効果の得られるソフトウェアはごく限られている．

我々は，並列処理を基本とするコンピュータをアーキテクチュアレベルから再構築することを目指し，“スレッドの排他的（走りきり）実行とスレッド間継続実行（continuation）”の概念による細粒度マルチスレッド処理アーキテクチャ（Fuce アーキテクチャとよぶ）およびその上でのソフトウェア方式の確立をめざして研究を行い，我々

の提案する Fuce アーキテクチャが並列分散処理にきわめて効果的であることを明らかにしてきた．それと同時にこの方式を活かすために，細粒度マルチスレッド処理の概念に基づくソフトウェア方式をさらに実用レベルで確立することの重要性を認識した．

２．研究の目的

本研究ではソフトウェア開発に重点を置いた研究を進めることによって，我々の提案する細粒度マルチスレッド処理方式の効果をより明確にする．具体的には，Fuce アーキテクチャ上で，言語処理系およびその実行環境を開発し，その性能評価を通して，細粒度マルチスレッド処理原理によるソフトウェア構成法の実用化可能性を検証する．特に，本研究の焦点は Fuce 上のソフトウェアを構築する上で必須である細粒度マルチスレッド処理原理による並列プログラミング技術を開発することにある．

３．研究の方法

(1) 言語処理系については CML (Continuation-based Multithreading Language)をソース言語とし，C 言語に変換する処理系を開発する．CML は継続概念によるスレッドを単位とし，イベント駆動や要求駆動の並行プロセスを低レベルで記述することができる．従来データ依存性のために並列実行が困難であった C 言語の逐次プログラムに対し，ストリーム（パイプライン）並列性を抽出することにより効率的に並列化する技法を開発する．

(2) OS 研究においては排他的細粒度マルチスレッドの実行制御を徹底させる OS 構造を究明する．対象マシンとして既存の商用プロセッサを想定し，その上に Fuce ランタイムシステムを開発して検証／評価実験を行う．

(3) 応用に関しては，科学技術計算などの並列分散ベンチマークプログラムをいくつか選定して選んで CML で記述しこれを Fuce ランタイムシステム上で走行させ，記述性と性能を評価する．

４．研究成果

言語処理系に関しては特に並列ストリーム処理に着目し，CML ストリーム処理による多重ループの並列展開手法を開拓した．

商用マシン上での Fuce-OS の開発に関しては，Unix-OS 上に Fuce ランタイムシステムを開発し，さらに，MPI を利用して複数ノードマシン対応に拡張した．

評価に関しては，データ依存性が複雑に絡む多重ループ処理のベンチマークとして ISC'07 ループと Levenstein distance アルゴリズムを選び，そのストリーム展開効果を Fuce ランタイムシステム上で評価した．

また，FUce の応用として位置づけているマルチエージェントシステムならびにシェルスクリプトの並列化に関する研究を進めた．

以下これらについて詳細を述べる．

(1) CML 言語とストリーム処理環境

繰り越し依存性をもつ多重ループの並列化に関して，細粒度マルチスレッディングモデルに基づく非同期ハンドシェイク式のストリームプログラミングを応用することで，繰り越し依存型多重ループの並列展開が容易に行なえる．

CML は基本的にほぼ ANSI 標準の C 言語をそのまま踏襲しており，ストリーム処理向きに若干の変更と拡張を行っている．

① CML 言語のストリーム処理機能

ストリーム処理の手法を用い，データ並列化とは異なるアプローチの並列化方法によって，複雑な手順を踏まなくとも簡単に波頭計算の恩恵が得られる．

CML はストリーム処理に関する以下の構文をもつ．

チャネル構造体

チャネルと呼ぶ基本的にデータ 1 個分の容量の中継記憶領域を用いる．

プロセス定義

CML におけるプロセスは Fuce 実行モデルのスレッドと基本的に同じである．

インスタンス生成

演算子 new を用いてプロセスを生成し，そのデータ領域とインスタンスを確保する．

ストリームの敷線

チャネル構造体 from にストリームデータ送出元のプロセス ID をセットし，to に受取先のプロセス ID をセットする．

プロセスの起動

プロセスの起動時に，チャネル名を実引数として呼び出す．ストリーム送出側なら<+>，受取側なら<->のモード指示子を付す．

② ストリームの基本動作

書き手プロセス w が，ストリーム要素 x の値を生成し，読み手プロセス r に伝える動作を永続的に繰り返す．CML 記述を以下に示す．

```
/＊＊ CML Program ＊＊/
process main() {
  w = new W(); r = new R();
  ch->from = w; ch->to = r;
  w(ch<+>);     // wのrecur-cont発行
  r(ch<->);     // rのrecur-cont発行
  cont w;       // wのinit-cont発行
}
process W(chan *ch) <2> {
  // xの値を決める計算
  ch->value = x;
  cont ch->to;      // rのtrigger-cont発行
  recur;
}
```

```
process R(chan *ch) <2> {
   x = ch->value;
   cont ch->from;       // wのack-cont発行
   // xの値を決める計算
   recur;
}
```

ストリーム要素は, チャネル ch を中継して w から r へ伝えられる．プロセス w, r の起動は，継続命令 cont によって制御される．

③ ストリームの動的伸張

CML Program のプロセス R に W の動作を加えてプロセス RW として記述し直す：

```
process RW(chan *in, chan *out) <3> {
      int x;
      in ?? x;        // チャネルから読み込む.
      out !! x;       // チャネルへ書き込む.
      recur;
}
```

次に，プロセス main 中のプロセス R の生成･起動部分をプロセス Create に置き換える：

```
process main() {
      int w = new W(ch<+>);
      new create(ch<->);
      cont w;
}
process create(chan *ch, int j) <2> {
      darea chan ch1;
      j++;
      new RW(ch<*>, ch1<+>, j);
      new create(ch1<->, j);
}
```

このようにすると，RW と Create の関係が２つのプロセス間のストリーム処理の関係となり，プロセス W がデータを生成し続ける限り次々に新たなプロセス RW が生成されていき，ストリームが伸張していく．

④ 多重ループの並列展開

２重ループの並列展開

数値ペアを２次元配列の要素と見なし，各時刻における全てのプロセス RW の受け持つ数値ペアに注目すると，ストリーム上で同時に参照する要素は例えば波頭計算などの２重内ループで参照する要素と全く同一と見なすことができる．

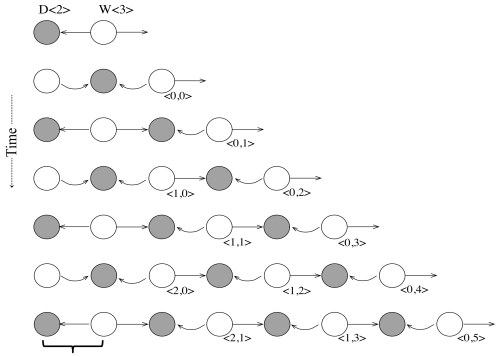

図１ データ依存を解消したストリーム動作

繰越し依存のある２重ループの並列展開

図１にこのストリーム処理の様子を示す．動作中または停止中のプロセスをそれぞれ白，黒の丸で表している．ループフェーズ間のデータ依存はストリーム処理の動作では自然に解消している．

３重ループの並列展開

２重ループのストリーム展開法を少し拡張することで，容易に３重ループを並列展開することができる．この様子を図２に示す．

さらに，同様に考えて，ストリーム上の各プロセスから新規にストリームが伸張していくようにすれば，n 重ループにも対応可能である．

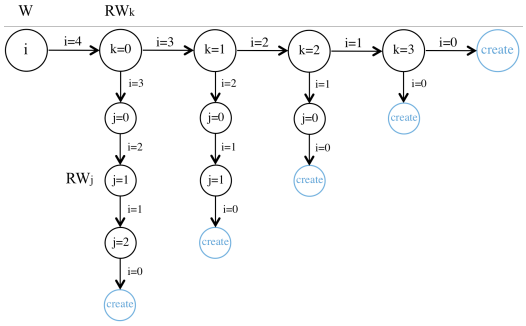

図２　３重ループのストリーム展開

鏡像による並列性の補強

図１自身とこれの鏡像を結合し図３のようなストリームを考えると，データ供給プロセス W は A モード，Bモード(鏡像)と起動ごとに動作モードを変更し，左右交互にデータを送信しながらストリームが左右両方に伸張していく．このことによって同時動作プロセス数が倍加し理想状態となる．

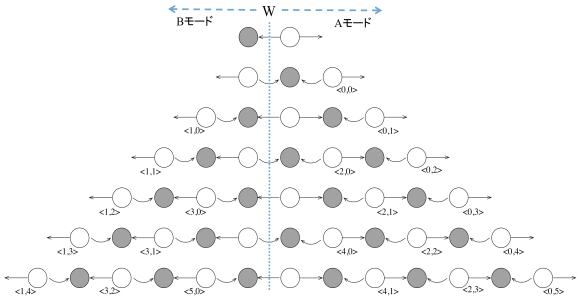

図３　鏡像

(2) 細粒度マルチスレッド実行環境
－ Fuce ランタイムシステムの実装 －

既存マシン上への Fuce-OS の実装法を検討した結果, Unix-OS 上に Fuce ランタイムシステムを開発・実装すること最適かつ効果的であると判断した．

① 共有メモリ型アーキテクチャ向け実装
－ マルチコアプロセッサへの実装 －

Fuce プロセッサの基本部分は ACM (Activation Control Memory)と TAC (Thread Activation Controller), および, 複数の TE (Thread Execution)からなる.

ACM と TAC

ACM は走りきりスレッドの発火･実行制御を管理する. ACM には実行中の関数(関数インスタンス)とその内部のスレッドの情報全てが保持されている.図４に ACM の概要を示す.

TAC はスレッド内部で発行される継続シグナル（例えば CML の cont）などの ACM 操作を処理し，実行可能スレッドのレディーキューへの投入，レディーキューからの実行待ちスレッドの取出しと TE の起動を行なう.

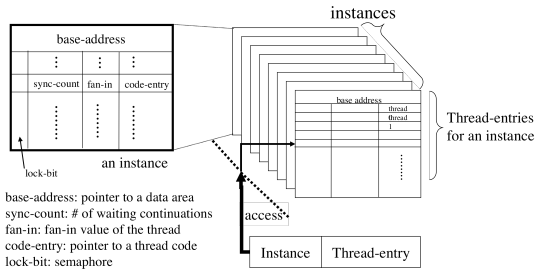

図４　ACM

TE プロセッサ

TE プロセッサはスレッドを実行する. TE を複数搭載することで複数スレッドの同時実行を可能としている.

TE と TAC の融合

TAC の機能を TE へ融合し，各 TE が独立して ACM アクセスを行うようにする. 各 TE には，アイドル状態に陥るのを防ぐために，別の TE のキューから実行待ちスレッド要素を‘盗む’ことができるようにする.

複数の TE とそれらが共通してアクセスする ACM の様子を図５に示す. この TE&TAC ルーチンを pthread (OS スレッド)として生成することで，各 TE は既存のマルチコアプロセッサ上で並行に動作可能となる.

継続命令の処理

CML の cont 命令は，継続スレッドの id を引数としてランタイムシステム内の関数 cont()を呼ぶことで処理する.

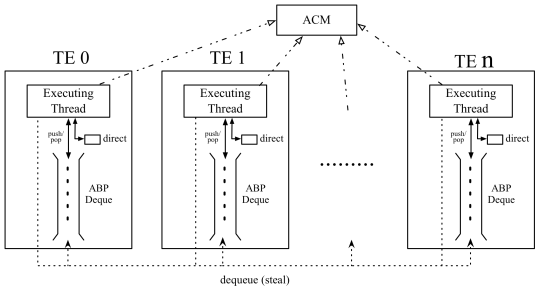

図５　複数の TE と ACM

② 分散メモリアーキテクチャ向け拡張

MPI を利用して Fuce ランタイムシステムを複数ノードマシン対応に拡張する.

ノードへのプロセス動的配置

プロセスの各ノードへの配置については, ACM のインスタンス ID を拡張し, プロセスグループ ID を設ける. ACM でのスレッドの同定は，プロセスグループ ID，インスタンス ID，スレッド ID の 3 つ組で行う.

MPI 専用プロセス

プロセスグループの数が増加するとそれだけノード間通信も増えるため，MPI のプロセスはプロセスグループ数に比例した個数を用意するのが自然な発想である(図６). しかし，MPI (OpenMPI v1.4.3) の仕様上の制限から独立に動作する MPI 送信用プロセス，受信用プロセスを複数動作させることは不可能であった. このためデータの送受信は単一プロセス内で MPI 関数 MPI_Sendrecv()によって一手に引き受けるという実装にせざるを得ない. また，MPI プロセスが通信処理のために頻繁にブロックされこれを実行している TE も同時にサスペンドしてしまうため，他の実行可能スレッドの実行開始を遅らせてしまう. これを防ぐため，MPI プロセスを動作させる TE では通常のスレッドをスケジュールしないようにしている.

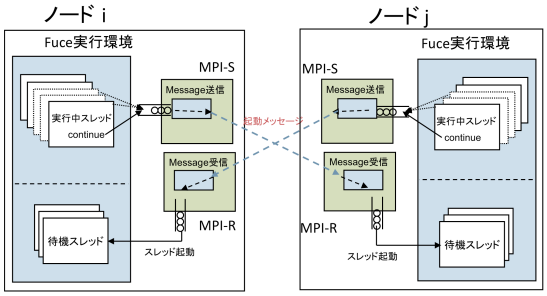

図６　理想的なノード間通信

ユーザプロセスから MPI プロセスへのデータ送信

MPI プロセスへのアクセスは排他制御が必要であり，このため ACM の lock-bit を操作する Fuce 命令 trylock()/unlock()を実装する.

MPI プロセスからユーザプロセスへのデータ送信

MPI プロセスは隣接ノードから継続先プロセス ID およびデータを受信すると, ACM へアクセスして対象プロセスのデータ領域へデータを書き込み cont()関数を直接呼び出す.

(3) 性能評価

① ベンチマークプログラム

ISC'07 ループ

ループ間でデータ依存が複雑に絡む多重ループのストリーム展開のアイデアを最初に思いつき，ISC'07 で発表した例題である.

Levenstein distance

異なる文字列の類似性を判定するアルゴ

リズムである．この種のアルゴリズムは近年，DNA，RNA 解析含む生物情報学などで利用されており，並列化による実行性能の向上は重要である．また，他の DP アルゴリズムもこれとほぼ同様のタイプであり，一般の DP アルゴリズムもストリーム処理方式による並列展開が可能になると考える．

② マルチコアプロセッサ上の性能評価

ISC'07 ループと Levenstein distance のストリーム処理スケーラビリティを AMD Opteron 8222，3.0GHz を 8 基搭載（合計 16 コア）した共有メモリ型 SMP マシンを用いて計測した．タイル化（ブロック化）を施し，1 プロセス内で 256 要素をまとめて計算するようにしたプログラムを実行した．入力データサイズは 8Kx8K の配列である．

実測結果を図 7 に示す．このグラフは逐次実行(元のループプログラム)に対するストリーム処理プログラムの性能向上率を表している．プロセッサ数が 8 個まではよく性能向上を得ているが，それ以降では頭打ち或は急激に悪化している．ストリーム処理では本質的にキャッシュの効果が薄く，メモリバス帯域を圧迫しているためであると思われる．

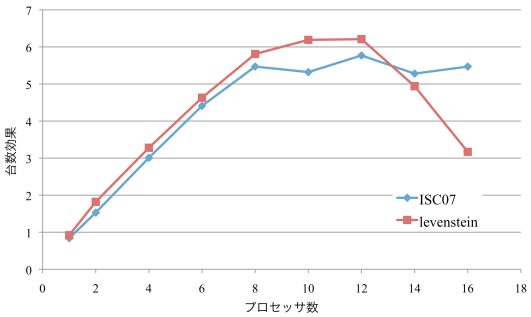

図 7　マルチコアプロセッサにおける速度向上

③ マルチノードマシン上の性能評価

isc07 および levenstein を入力配列データサイズ 6Kx16K，タイルサイズ 256(配列要素 256 個)としてストリーム化したものを用いた．実験環境には Intel Xeon 3.3GHz（6 コア CPU）を 2 台(計 12CPU コア)搭載した Linux マシンを 2 台利用した．(台数を 2 に限っているのは，後述するように分散メモリマシン上では台数効果がほとんど得られないという結果になってしまったため台数を増やしても意味がないと判断したためである.)

図 8 に実験結果を示す．図中，isc07，levenstein となっているものは共有メモリ型のプログラムとして実行し，接頭語 mpi が付いているものは分散メモリ版のプログラム，接尾語 in が付いているものは単一マシン内で MPI ノードを 2 つ実行しているもの，接尾語 ext となっているものは 2 台のマシン上に MPI ノードをそれぞれ 1 つ配置し，計 2 ノードを実行したものである．共有メモリ型の isc07 は 5 倍弱，levenstein は 10 倍程度で性能が頭打ちになっている（②の結果と比べると実行環境（CPU）が異なるために若干異なっているが傾向は同じである).

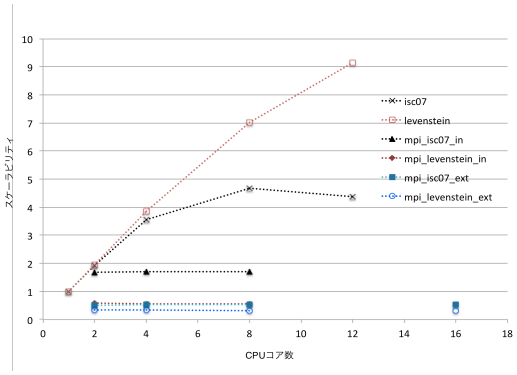

図 8　CPU コア数に対する速度向上

分散メモリ型の性能は共有メモリ型と比較しても，著しく悪く，CPU コア数に対してほとんど台数効果が得られないという結果になった．

この理由を考察する．(2)の② 節でも触れたように，MPI を利用したデータの送受信では，送信処理と受信処理を単一スレッド内にまとめる必要があるため，それぞれを独立プロセスとして並行動作させることが困難であり，送信すべきデータが次々と発生しても送信処理だけを優先して実行することができない．ストリーム処理ではこの状態は致命的となる．ストリームの先頭のプロセスはデータ供給を絶えず行っているが，このプロセスが属するプロセスグループからの隣接ノードへのデータ送信が滞るとストリーム先頭プロセスも直ちに動作不能となり，データの供給速度が著しく低下してしまう．そのため，ストリーム全体が不活性化し，並行動作しているプロセスが極端に減ってしまうと考えられる．また，データ送信が滞るということは，それを受信するノードにもデータがあまり届かないことを意味する．したがって，受信側ノード内での並行動作可能なプロセス数も限られてしまう．

データの送信側，受信側ともにノード内の並行動作可能なプロセス数が低下してしまうため，ノード内の共有メモリ型実行部分の並列度(TE 数)を上げても性能上の効果はあがらないという結果になってしまった．

MPI を用いない他の実装法をいろいろ試みたが上記問題を解決することはほとんど困難であった．結局，ソフトウェア実装によってデータ転送プロセスと内部処理プロセスを並行して行なわせることには無理があり，別途，データ転送・管理専用のハードウェアを備える必要があるとの結論に達した．

(4) その他応用
本研究の主題である細粒度マルチスレッド処理の概念に基づくソフトウェア方式の研究とは現在のところ直接結びつかないが，その応用として位置づけているマルチエージェントシステムならびにシェルスクリプトの並列化に関する研究を行なった．

５．主な発表論文等
（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計７件）
① S.Amamiya, M.Amamiya, R.Hasegawa, H.Fujita : A continuation-based non-interruptible multithreading processor architecture, Journal of Supercomputing, 査読有, vol.47, 2009, 228-252.
② T.Mine, K.Kimura, S.Amamiya, K.Takahashi, M.Amamiya : Agent-Community-Network-Based Secure Collaboration Support System , LNBIP,Springer, 査読有, Vol.25,No.1, 2009, 234-255
③ H.Yu, T.Mine, M.Amamiya : Agent-Community-based P2P semantic MyPortal information retrieval system architecture , Journal of Embedded Computing , 査読有, Vol.3,No.1, 2009, 63-75
④ T.Matsuzaki, N. Urashima, M.Amamiya : Thread Control Mechanism for Multithreading Processor, Proc. The 2010 International Conference on Computer Design (CDES'10), 査読有, 2010, 81-87
⑤ M.Koshimura, H.Nabeshima, H.Fujita, R.Hasegawa : Solving Open Job-Shop Scheduling Problems by SAT Encoding , IEICE TRANS.on Information and Systems,査読有,Vol.E93-D, No.8, 2010, 2316-2318
⑥ Haibo Yu, T.Mine, M.Amamiya : Towards User Intent Based Searching, Proc. 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 査読有, 2011, 1400 - 1407
⑦ S. Amamiya, M. Amamiya : Stream Processing Approach on Fuce System for Parallelizing Nested Loops with Data Dependency,Proc. of The 2012 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'12), 査読有, 2012, to appear

〔学会発表〕（計５件）
① 雨宮聡史, 長谷川隆三, 雨宮真人：ストリーム処理方式を用いた繰越し依存型多重ループの並列展開法，情報処理学会プログラミング研究会（2009-5-(6), 2010年3月
② 雨宮聡史, 雨宮真人：分散メモリアーキテクチャ向け Fuce ランタイムシステムとその上でのストリーム処理の評価，情報処理学会プログラミング研究会（2010--5-(4), 2011 年 4 月
③ 峯恒憲：P2P 型情報検索のためのユーザフィードバックの活用とクエリ送信先学習手法の評価，電子情報通信学会技術研究報告(人工知能と知識処理), 2010 年 6 月
④ 峯恒憲:セキュアで頑健なコミュニティ指向マルチエージェントフレームワーク，電子情報通信学会技術研究報告(CSP 研究会), 2010 年 7 月
⑤ 前田直人, 水谷泰治：PCクラスタ環境下におけるシェルスクリプトの半自動並列化手法の提案と評価，第73回情報処理学会全国大会，2011年3月

〔産業財産権〕

○ 取得状況（計２件）

名称：Network node machine and information network system
発明者：M.Amamiya, et.al.
権利者：Fujitsu Limited
種類：特許
番号：US 7,975,291
取得年月日：July 5, 2011
国内外の別：国外

名称:ネットワークシステム、情報処理装置、情報転送方法、プログラム及び記録媒体
発明者：雨宮真人，他
権利者：九州大学
種類：特許
番号：特許第 4806774 号
取得年月日：2011 年 8 月 23 日
国内外の別：国内

６．研究組織
(1)研究代表者
雨宮 真人（AMAMIYA MAKOTO）
財団法人九州先端科学技術研究所・特別研究員
研究者番号：90202697

(2)研究分担者
長谷川 隆三（HASEGAWA RYUZOU）
九州大学・システム情報科学研究院・教授
研究者番号：20274483
藤田 博（FUJITA HIROSHI）
九州大学・システム情報科学研究院・准教授
研究者番号：70284552
峯 恒憲（TSUNENORI MINE）
九州大学・システム情報科学研究院・准教授
研究者番号：30243851
越村 三幸（KOSHIMURA MIYUKI）
九州大学・システム情報科学研究院・助教
研究者番号：30274492
水谷 泰治（MIZUTANI YASUHARU）
大阪工業大学・情報科学部・講師
研究者番号：10411414