(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第4295915号
(P4295915)

(45)発行日　平成21年7月15日(2009.7.15)　(24)登録日　平成21年4月17日(2009.4.17)

(51)Int.Cl.　　F I
G06F 9/50 (2006.01)　　G06F 9/46 465Z
G06F 9/45 (2006.01)　　G06F 9/44 322A
G06F 11/28 (2006.01)　　G06F 11/28 340C

請求項の数 44　(全 23 頁)

(21)出願番号　特願2000-517338 (P2000-517338)
(86)(22)出願日　平成10年10月21日 (1998.10.21)
(65)公表番号　特表2001-521218 (P2001-521218A)
(43)公表日　平成13年11月6日 (2001.11.6)
(86)国際出願番号　PCT/US1998/022261
(87)国際公開番号　WO1999/021085
(87)国際公開日　平成11年4月29日 (1999.4.29)
審査請求日　平成17年10月19日 (2005.10.19)
(31)優先権主張番号　08/954,843
(32)優先日　平成9年10月21日 (1997.10.21)
(33)優先権主張国　米国 (US)

(73)特許権者　500184268
エフティーエル　システムズ　インコーポレイテッド
アメリカ合衆国　ミネソタ州　５５９０２　ロチェスター　グリーンヴィュー　ドライヴ　サウス　ウェスト　１６２０
(74)代理人　100059959
弁理士　中村　稔
(74)代理人　100067013
弁理士　大塚　文昭
(74)代理人　100082005
弁理士　熊倉　禎男
(74)代理人　100065189
弁理士　宍戸　嘉一

最終頁に続く

(54)【発明の名称】コンピュータプログラムとハードウェアモデルを並列にコンパイレーション、シミュレーション、及び実行するためのコンパイラ指向装置

(57)【特許請求の範囲】
【請求項１】
分散型のコンパイラ指向のデータベースを実行するためにプログラム化された少なくとも一つのノードを持ち、コンパイレーション（コンパイル）作動モード、シミュレーション作動モード、及び／又は、ソフトウェア実行作動モードを含む並列プロセッサシステムであって、
少なくとも二つのプロセッサと、
少なくとも一つのプロセッサに接続され、ローカルなメモリであるか又は共有されたメモリであるかのどちらかであるメモリと、
前記少なくとも二つのプロセッサと前記メモリとを接続するための相互接続機構と、
再構成可能なロジックエレメントとを備え、
キャッシュ及び共有メモリ内の実行可能なプロセッサ命令、又は前記再構成可能なロジックエレメント内のコンパイル・プロファイリング・デバックに関連する機能の追加を通じて、前記プログラム化されたノードを組み込み、前記分散型のコンパイラ指向のデータベース上で一つ又はそれ以上の下記に示す機能を実行する複数のクライアントプログラムを含み、
前記下記の機能として、
完全ソースコードファイルを分析したり、又はフラッグメントをコンパイラ指向データベースの表示に翻訳することを指令することで、ソースコードの変更が前記データベースへの変更を生じさせるという、前記データベースに対するソースコードの分析機能と、

新たな又は改訂されたエントリーを生成するため、以前に分析された情報を精緻化（エラボレーション：elaboration)又はインライニング(in-lining)する機能と、

ハードウェア記述言語又は最適コンパイルの分野の技術を用いて、前記分散型のコンパイラ指向データベースに含まれている情報を変換し、より効率的なシミュレーション、より効率的な実行、又はより観測可能な実行を生成する最適化のための機能と、

分散型の全体分析と書き直し変換を用いて、前記分散型のコンパイラ指向データベースに含まれている情報を変換し、プログラム可能なプロセッサと再構成可能なロジックエレメントとの組合せ上で直接実行される表示を生成するためのコードの生成・集約・結合機能と、

ローカルメモリ、共有メモリ、及び再構成可能論理をロードし実行を制御するために、コンパイラ指向データベースからの記号情報及び制御情報を利用するオペレーション機能と、

実行、区切点、設定状態及び読出し状態を提供するデバッグ機能と、

外部のツール又はユーザに対しシミュレーション／実行の結果をアクセス可能とするためにシミュレーション／実行をプロファイリング(profiling)又はイベント記録する機能と、

インタフェースを通して、クライアントプログラムが、既存オブジェクトのコピーを作成、要求、更新し、そしてコンパイラ及びシミュレーション／実行に関連するデータベースオブジェクトのコレクションを解除し解放する機能を含む、ことを特徴とする並列プロセッサシステム。

【請求項２】

前記少なくとも二つのプロセッサの少なくとも一方が、前記少なくとも二つのプロセッサの少なくとも一方の指示ストリームからの指示を登録するために、レジスタの動作により指名された再構成可能なロジック実行パイプラインにより増強されていることを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項３】

前記クライアントが、少なくとも二つの通信するオペレーティングシステムプロセッサの間で、追加アドレス可能メモリ、ソフトウェア保護を提供し且つ目標ソフトウェア実行環境を模倣（エミュレート）するように作動することを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項４】

前記クライアントが、少なくとも二つの通信するオペレーティングシステムプロセッサの間で、追加アドレス可能メモリ、ソフトウェア保護を提供し且つ目標ソフトウェア実行環境を模倣（エミュレート）するように作動することを特徴とする請求項２に記載の並列プロセッサシステム。

【請求項５】

前記複数のクライアントが、さらに変化し再分割されたシミュレーション／実行を、プロセッサと再構成可能なロジック装置との間で実行するために、前記データベース内の増分且つ一貫した変化を通じてソースコードを変更し、ソースコード内の変化が、シミュレーション／実行に関する変更要求を満たすために、シミュレーション／実行の状態を保存することを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項６】

前記シミュレーション／実行を中断する段階が、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを含むことを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項７】

前記複数のクライアントが、さらに変化し再分割されたシミュレーション／実行を、プロセッサと再構成可能なロジック装置との間で実行するために、前記データベース内の増分且つ一貫した変化を通じてソースコードを変更し、最適化されたコードシーケンスとデ

バッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを保存することを特徴とする請求項２に記載の並列プロセッサシステム。

【請求項８】

前記シミュレーション／実行を中断する段階が、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを含むことを特徴とする請求項７に記載の並列プロセッサシステム。

【請求項９】

あるノード上に常駐する情報とある遠隔ノード上にのみ存在する情報とを関係付けるデータベース上に含まれる関係情報が、ある遠隔代理レコードによって表され、前記遠隔代理レコードが、キャッシュされたコピーの寿命を制限するために、メモリの利用可能性と整合性を使って前記遠隔ノード上の前記情報を一貫してキャッシュすることを可能にすることを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項１０】

あるノード上に常駐する情報とある遠隔ノード上にのみ存在する情報とを関係付けるデータベース上に含まれる関係情報が、ある遠隔代理レコードによって表され、前記遠隔代理レコードが、キャッシュされたコピーの寿命を制限するために、メモリの利用可能性と整合性を使って前記遠隔ノード上の前記情報を一貫してキャッシュすることを可能にすることを特徴とする請求項２に記載の並列プロセッサシステム。

【請求項１１】

データベース情報の間における分析、精緻化、実行時間の従属関係が、コンパイレーション、シミュレーション、又は実行のクライアント動作を同一又は他のノード上にトリガーすることを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項１２】

データベース情報の間における分析、精緻化、実行時間の従属関係が、コンパイレーション、シミュレーション、又は実行のクライアント動作を同一又は他のノード上にトリガーすることを特徴とする請求項２に記載の並列プロセッサシステム。

【請求項１３】

精緻化クライアントの作動により、コード生成の以前に、精緻化されたデータベースオブジェクトを作成することができ、精緻化の結果生じたデータベースオブジェクトが、分析クライアントの作動から生じたデータベースオブジェクト又は精緻化クライアントの以前の作動により生成されたオブジェクトを再使用することを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項１４】

精緻化クライアントの作動により、コード生成の以前に、精緻化されたデータベースオブジェクトを作成することができ、精緻化の結果生じたデータベースオブジェクトが、分析クライアントの作動から生じたデータベースオブジェクト又は精緻化クライアントの以前の作動により生成されたオブジェクトを再使用することを特徴とする請求項２に記載の並列プロセッサシステム。

【請求項１５】

分散型のコンパイラ指向のデータベースを実行するためにプログラム化された少なくとも一つのノードを持ち、コンパイレーション作動モード、シミュレーション作動モード、及び／又は、ソフトウェア実行作動モードを含む並列プロセッサシステムであって、

少なくとも二つのプロセッサと、

少なくとも一つのプロセッサに接続され、ローカルなメモリであるか又は共有されたメモリであるかのどちらかであるメモリと、

前記少なくとも二つのプロセッサと前記メモリとを接続するための相互接続機構とを備え、

前記プログラム化されたノードを組み込み、前記分散型のコンパイラ指向のデータベー

ス上で一つ又はそれ以上の下記に示す機能を実行する複数のクライアントプログラムを含み、
前記下記の機能として、
完全ソースコードファイルを分析したり、又はフラッグメントをコンパイラ指向データベースの表示に翻訳することを指令することで、ソースコードの変更が前記データベースへの変更を生じさせるという、前記データベースに対するソースコードの分析機能と、
新たな又は改訂されたエントリーを生成するために、以前に分析された情報を精緻化（エラボレーション：elaboration）又はインライニング(in-lining)する機能と、
ハードウェア記述言語又は最適コンパイルの分野の技術を用いて、前記分散型のコンパイラ指向データベースに含まれている情報を変換し、より効率的なシミュレーション、より効率的な実行、又はより観測可能な実行を生成する最適化のための機能と、
分散型の全体分析と書き直し変換を用いて、前記分散型のコンパイラ指向データベースに含まれている情報を変換し、プログラム実行可能なプロセッサと再構成可能なロジックエレメントとの組合せ上で直接実行される表示を生成するための、コードの生成・集約・結合機能と、
ローカルメモリ、共有メモリ、及び再構成可能論理をロードし実行を制御するために、コンパイラ指向データベースからの記号情報及び制御情報を利用するオペレーション機能と、
実行、区切点、設定状態及び読出し状態を提供するデバッグ機能と、
外部のツール又はユーザーに対しシミュレーション／実行の結果をアクセス可能とするためにシミュレーション／実行をプロファイリング(profiling)又はイベント記録する機能と、
クライアントプログラムが、既存オブジェクトのコピーを作成、要求、更新し、そしてコンパイラ及びシミュレーション／実行に関連するデータベースオブジェクトのコレクションを解除し解放する機能を含む、ことを特徴とする並列プロセッサシステム。
【請求項１６】
クライアントが、少なくとも二つの通信するオペレーティングシステムプロセッサの間で、追加アドレス可能メモリ、ソフトウェア保護を提供し且つ目標ソフトウェア実行環境を模倣（エミュレート）するように作動することを特徴とする請求項１５に記載の並列プロセッサシステム。
【請求項１７】
前記複数のクライアントが、さらに変化し再分割されたシミュレーション／実行を、プロセッサと再構成可能なロジック装置との間で実行するために、前記データベース内の増分且つ一貫した変化を通じてソースコードを変更し、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを保存することを特徴とする請求項１５に記載の並列プロセッサシステム。
【請求項１８】
シミュレーション／実行を中断する段階が、フラッグメントコードを再構成し且つ状態をマッピングし、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを含むことを特徴とする請求項１７に記載の並列プロセッサシステム。
【請求項１９】
あるノード上に常駐する情報とある遠隔ノード上にのみ存在する情報とを関係付けるデータベース上に含まれる関係情報が、ある遠隔代理レコードによって表され、前記遠隔代理レコードが、キャッシュされたコピーの寿命を制限するために、メモリの利用可能性と整合性を使って前記遠隔ノード上の前記情報を一貫してキャッシュすることを可能にすることを特徴とする請求項１５に記載の並列プロセッサシステム。

【請求項２０】
データベース情報の間における分析、精緻化、実行時間の従属関係が、コンパイレーション、シミュレーション又は実行のクライアント動作を同一又は他のノード上にトリガーし、その結果、様々なノード上の情報やクライアントの位置、並びに当該ノードの構成要素に左右されることなく、前記データベースが前記トリガーをその関係のあるコンパイレーション、シミュレーション又は実行に連結することを特徴とする請求項１５に記載の並列プロセッサシステム。
【請求項２１】
精緻化クライアントの作動により、コード生成の以前に、精緻化されたデータベースオブジェクトを作成することができ、精緻化の結果生じたデータベースオブジェクトが、分析クライアントの作動から生じたデータベースオブジェクト又は精緻化クライアントの以前の作動により生成されたオブジェクトを再使用することを特徴とする請求項１５に記載の並列プロセッサシステム。
【請求項２２】
前記少なくとも二つのプロセッサの少なくとも一方が、前記少なくとも二つのプロセッサの少なくとも一方の指示ストリームからの指示を登録するために、レジスタの動作により指名された再構成可能なロジック実行パイプラインにより増強されていることを特徴とする請求項１５に記載の並列プロセッサシステム。
【請求項２３】
クライアントが、少なくとも二つの通信するオペレーティングシステムプロセッサの間で、追加アドレス可能メモリ、ソフトウェア保護を提供し且つ目標ソフトウェア実行環境を模倣（エミュレート）するように作動することを特徴とする請求項２２に記載の並列プロセッサシステム。
【請求項２４】
前記複数のクライアントが、さらに変化し再分割されたシミュレーション／実行を、プロセッサと再構成可能なロジック装置との間で実行するために、前記データベース内の増分且つ一貫した変化を通じてソースコードを変更し、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを保存することを特徴とする請求項２２に記載の並列プロセッサシステム。
【請求項２５】
シミュレーション／実行を中断する段階が、フラッグメントコードを再構成し且つ状態をマッピングし、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを含むことを特徴とする請求項２４に記載の並列プロセッサシステム。
【請求項２６】
あるノード上に常駐する情報とある遠隔ノード上にのみ存在する情報とを関係付けるデータベース上に含まれる関係情報が、ある遠隔代理レコードによって表され、前記遠隔代理レコードが、キャッシュされたコピーの寿命を制限するために、メモリの利用可能性と整合性を使って前記遠隔ノード上の前記情報を一貫してキャッシュすることを可能にすることを特徴とする請求項２２に記載の並列プロセッサシステム。
【請求項２７】
データベース情報の間における分析、精緻化、実行時間の従属関係が、コンパイレーション、シミュレーション又は実行のクライアント動作を同一又は他のノード上にトリガーすることを特徴とする請求項２２に記載の並列プロセッサシステム。
【請求項２８】
精緻化クライアントの作動により、コード生成の以前に、精緻化されたデータベースオブジェクトを作成することができ、精緻化の結果生じたデータベースオブジェクトが、分

析クライアントの作動から生じたデータベースオブジェクト又は精緻化クライアントの以前の作動により生成されたオブジェクトを再使用することを特徴とする請求項２２に記載の並列プロセッサシステム。

【請求項２９】

前記少なくとも二つのプロセッサの少なくとも一方が、前記少なくとも二つのプロセッサの指示ストリームから固定アドレスへマップされた指示の動作により指名された再構成可能なロジック実行パイプラインにより増強されていることを特徴とする請求項１５に記載の並列プロセッサシステム。

【請求項３０】

クライアントが、少なくとも二つの通信するオペレーティングシステムプロセッサの間で、追加アドレス可能メモリ、ソフトウェア保護を提供し且つ目標ソフトウェア実行環境を模倣（エミュレート）するように作動することを特徴とする請求項２９に記載の並列プロセッサシステム。

【請求項３１】

前記複数のクライアントが、さらに変化し再分割されたシミュレーション／実行を、プロセッサと再構成可能なロジック装置との間で実行するために、前記データベース内の増分且つ一貫した変化を通じてソースコードを変更し、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを保存することを特徴とする請求項２９に記載の並列プロセッサシステム。

【請求項３２】

シミュレーション／実行を中断する段階が、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを含む必要なシミュレーション／実行の中間状態を生成することを特徴とする請求項３１に記載の並列プロセッサシステム。

【請求項３３】

あるノード上に常駐する情報とある遠隔ノード上にのみ存在する情報とを関係付けるデータベース上に含まれる関係情報が、ある遠隔代理レコードによって表され、前記遠隔代理レコードが、キャッシュされたコピーの寿命を制限するために、メモリの利用可能性と整合性を使って前記遠隔ノード上の前記情報を一貫してキャッシュすることを可能にすることを特徴とする請求項２９に記載の並列プロセッサシステム。

【請求項３４】

データベース情報の間における分析、精緻化、実行時間の従属関係が、コンパイレーション、シミュレーション又は実行のクライアント動作を同一又は他のノード上にトリガーすることを特徴とする請求項２９に記載の並列プロセッサシステム。

【請求項３５】

精緻化クライアントの作動により、コード生成の以前に、精緻化されたデータベースオブジェクトを作成することができ、精緻化の結果生じたデータベースオブジェクトが、分析クライアントの作動から生じたデータベースオブジェクト又は精緻化クライアントの以前の作動により生成されたオブジェクトを再使用することを特徴とする請求項２９に記載の並列プロセッサシステム。

【請求項３６】

前記少なくとも二つのプロセッサの少なくとも一方が、前記少なくとも二つのプロセッサの指示ストリームから固定アドレスへマップされた指示の動作により指名された再構成可能なロジック実行パイプラインにより増強されていることを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項３７】

クライアントが、少なくとも二つの通信するオペレーティングシステムプロセッサの間

で、追加アドレス可能メモリ、ソフトウェア保護を提供し且つ目標ソフトウェア実行環境を模倣（エミュレート）するように作動することを特徴とする請求項３６に記載の並列プロセッサシステム。

【請求項３８】

前記複数のクライアントが、さらに変化し再分割されたシミュレーション／実行を、プロセッサと再構成可能なロジック装置との間で実行するために、前記データベース内の増分且つ一貫した変化を通してソースコードを変更し、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを保存することを特徴とする請求項３６に記載の並列プロセッサシステム。

【請求項３９】

シミュレーション／実行を中断する段階が、最適化されたコードシーケンスとデバッグコードシーケンス間のスタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするため、コンパイラ指向データベース、コード生成器及びリンカーに関するオペレーションを含むことを特徴とする請求項３８に記載の並列プロセッサシステム。

【請求項４０】

あるノード上に常駐する情報とある遠隔ノード上にのみ存在する情報とを関係付けるデータベース上に含まれる関係情報が、ある遠隔代理レコードによって表され、前記遠隔代理レコードが、キャッシュされたコピーの寿命を制限するために、メモリの利用可能性と整合性を使って前記遠隔ノード上の前記情報を一貫してキャッシュすることを可能にすることを特徴とする請求項３６に記載の並列プロセッサシステム。

【請求項４１】

データベース情報の間における分析、精緻化、実行時間の従属関係が、コンパイレーション、シミュレーション又は実行のクライアント動作を同一又は他のノード上にトリガーすることを特徴とする請求項３６に記載の並列プロセッサシステム。

【請求項４２】

精緻化クライアントの作動により、コード生成の以前に、精緻化されたデータベースオブジェクトを作成することができ、精緻化の結果生じたデータベースオブジェクトが、分析クライアントの作動から生じたデータベースオブジェクト又は精緻化クライアントの以前の作動により生成されたオブジェクトを再使用することを特徴とする請求項３６に記載の並列プロセッサシステム。

【請求項４３】

前記少なくとも二つのプロセッサの各々が、構成可能な指示セットアーキテクチャをサポートし、前記少なくとも二つのプロセッサの内の一つが、再構成可能な論理実行パイプラインにより増強されていることを特徴とする請求項１に記載の並列プロセッサシステム。

【請求項４４】

前記少なくとも二つのプロセッサの各々が、再構成可能な指示セットアーキテクチャをサポートし、前記少なくとも二つのプロセッサの内の一つが、再構成可能な論理実行パイプラインにより増強されていることを特徴とする請求項１５に記載の並列プロセッサシステム。

【発明の詳細な説明】

【０００１】

（発明の背景）

コンピュータープログラムとハードウェアモデルのコンパイレーション、シミュレーション及び実行を二つ以上の処理ノード内に分散させると、プログラム／モデルの容量の増加と、シミュレーション／実行の時間の削減という二つの大きな利点が得られる。コンパイルし、シミュレートし／実行できるプログラム／モデルの規模と複雑さは、利用可能な処理リソースのみならず、メモリを追加することにより増す。シミュレーション／実行の時

間は、多数の処理ノードによりコンパイレーション、シミュレーション及び実行を同時に行うことだけでなく、プログラム／モデルを表す部分的にコンパイルされた中間手段への、最適化のためのアクセスをする機会によっても減少する。

【０００２】

処理ノードは、ある共通メモリを共有する一つ又はそれ以上の汎用プロセッサで構成される。処理ノードの随意的な構成要素として、プロセッサ専用メモリ、単一プロセッサ専用の又は二つ以上のプロセッサに共有される様々なレベルのメモリキャシング、単一プロセッサ専用の又は二つ以上のプロセッサに共有される再構成可能な論理を挙げることができる。処理ノードは、従来のアドレス翻訳ハードウェア及びソフトウェアにより、物理的なメモリ上にマップされた一つ又はそれ以上の固有仮想アドレススペースをサポートする。処理ノードは、再構成可能な論理アレイを加えた共有メモリマルチプロセッサとみなすことができる。

【０００３】

処理ノード（及び共有メモリマルチプロセッサ）は、約１２個までのプロセッサを含む構成として容易に構築されるが、追加プロセッサは共有メモリへ接続して追加されるので、共有メモリを求めて競合が起こり、各プロセッサの効率は低下する。従って、規模が大きく強力なコンピューターシステムはしばしば、こうした二つ以上の処理ノードをポイントツーポイント又はマルチキャストメッセージプロトコルを使って連結させることによって、作成される。ポイントツーポイントメッセージプロトコルは１ユニットの情報（メッセージ）を、一つの処理ノード上のエージェントから同じ処理ノード又は別の処理ノード上のエージェントへ伝達する。マルチキャストメッセージ・プロトコルにより、一つの処理ノード上のエージェントから同じ処理ノード又は別の処理ノード上の一つ又はそれ以上のエージェントへの通信が行われる。エージェント機能は、プロセッサ上で作動するソフトウェアか、又は再構成可能な論理アレイに組み込まれるか又は関連付けされるハードウェアかのどちらかの形で具体化される。こうしたエージェントにより、コンパイレーション、シミュレーション又は実行の構成要素が具体化される。

【０００４】

コンパイレーション、シミュレーション及び実行は、プロセッサの実行可能命令（キャッシュ及びメモリ中に現れる）及び論理構成（再構成可能な論理要素中に現れる）の形で具体化された、密接に相互関連する作動モードとして生まれたものと見なすことができる。コンパイレーションにより、一つ又はそれ以上のコンピュータープログラム及び／又はハードウェアモデルは、プロセッサの実行可能命令及び論理構成情報に変換される。実行可能命令及び論理構成により表された動作は、次にシミュレーション及び／又は実行として評価される。一般的な使い方では、シミュレーションはしばしばハードウェアモデルの評価を指し、実行はコンピュータープログラムの評価を指す。ハードウェア／ソフトウェアの共同設計に見られるように、ハードウェア記述言語（ＶＨＤＬ及びＶｅｒｉｌｏｇ）が盛んに使われるようになったので、シミュレーションと実行とは区別がほとんどできない作動モードとなり下記のような扱いがなされている。

【０００５】

記号デバッグ、プロフィーリング、フォールト挿入、選択的イベント追跡、ライブラリのダイナミックリンキング、（利用可能なリソース又は新しい情報に基づく）実行可能命令のインクレメンタルな最適化、プログラミングインターフェースのような、プログラム及びモデル上で起こる作動のインクレメンタルなモードを収容するためには、実行／シミュレーションの下でプログラム／モデルをインクレメンタルに修正する必要があり、コンパイレーションモードと実行／シミュレーションモードとを密接に連結することが有益である。固定された実行リソースが前提である場合、こうした密接な連結によりシミュレーション／実行の時間が削減される。

【０００６】

コンパイレーションは通常、実行／シミュレーションの作動モードに達する以前に、二つ以上の中間ファイル（パイプによる実際又はシミュレートされたメモリ中のファイル）を

使って一方向のパイプライン中に配列される。共通の中間ファイルは、中間最適化表示と、テキストアッセンブリコードと、再配置可能なバイナリと、実行可能なファイルとを含んでいる。ハードウェアモデルのコンパイレーションをプログラムに翻訳し、次にこのプログラムをプログラミング言語専用のコンパイラによりコンパイルする場合、多くのシミュレーターはプログラム言語中間手段を導入する。最適化コンパイラによっては、１２個ものファイル中間手段を利用するものがある。

【０００７】

コンパイレーションのフェーズ間における一方向の通信にファイルのような装置を使うと、情報をコンパイレーション作動モードの最終段階から初期段階へ素早く効率的に戻すことが阻害される。例えば、より最適なシミュレーション／実行をロードするために、実行可能なプロセッサ指示を共有メモリ内に配置する後置コンパイラの機能又は再構成可能な論理アレイ内の論理機能は、フォールスシェアリング又は再構成可能な論理ピンの競合を検出できるが、このフォールスシェアリングや論理ピンの競合は、初期のコンパイレーション機能（この場合はマッピングとスケジューリングの機能）を部分的に再実行することにより、非常に効率的に処理される。

【０００８】

ファイルは、コンパイレーションの段階間の非常に大雑把な通信メカニズムでもある。実質的な情報は一般にファイル中間手段中に存在するが、この中間手段は、シミュレーションに対するローカルな変化とは関係を持っていない。従ってコンパイレーションあるいは再コンパイレーションでは、所要の作動に必要な情報よりも実質的に多くの情報を扱わなければならない。こうした追加作業は時間がかかるため、実行／シミュレーションの段階へ達するには長時間を要する。

【０００９】

コンパイレーションの作動モードがすべての中間手段を中間ファイルのシーケンス内でなくメモリ内に保持している数少ないケースを調査文献で見ると、そうした保持は単一プロセッサのメモリ内で行われている。コンパイラ作動中に全中間手段へ全体的にアクセスすると、実質的な実行／シミュレーション性能が向上するが、いかなる単一プロセッサも一般に物理的に存在するメモリと同様にアドレス可能な範囲が制限されている。従ってこうした方法では、コンパイラの作動変更やシミュレーション／実行の新たらしい装置を目標として新しいエージェントを導入することが難しくなり、単一プロセッサ上でコンパイルされるプログラム又はモデルのサイズが制限される。

【００１０】

既存のコンパイラ文献とコンパイラ製造環境の中では、コンパイレーションの単一フェーズを加速するために、共有メモリのマルチプロセッサを使ってコンパイレーションを並列に実行するか、又はソースファイルを個別の関連実行可能命令に編成しその後に２進数のシーケンシャルリンケージを単一の実行可能命令にするようになっている。共有メモリのマルチプロセッサ上で単一のコンパイレーションフェーズを加速するコンパイレーションは、研究目的には非常に適しているが全体のコンパイレーション又はインクレメンタルな再コンパイレーションの遅れを減らすことに直接応用することができない。孤立したマルチファイルプログラム又はモデルから成る各ファイルをコンパイルしても、ファイル間の情報の流れにより更に最適な実行可能命令を作り出すことはできない。例えば、あるファイル内に存在する機能の本体は、その本体が第二ファイルのコンパイルの一部としてテキストの形で含まれていなければ、別のファイル（しばしばインライニングと呼ばれる）中で呼び出しを受けた場所で組み込んで利用することができない。単一ファイルにテキストの形として含まれる情報が多くなるにつれ、ファイルサイズも大きくなり、コンパイルされる全体のプログラム又はモデルサイズ、即ち、コンパイルに必要な作業の総量が結果的に限られてしまう（コンパイル中に二回以上同じ情報が分析されるので）。

【００１１】

１９９０年、要約データタイプ（クラス）の中間表示例を使って分析型ハードウェア記述言語モデルを表示する研究報告が発表された。メモリアドレス（ポインタ）が事例間の関

係を記述している。例えば、中間表示例のシーケンスはそれぞれが次の例へポインターを有し、リンクされたリストを形成する。この報告は、中間表示を二つ以上のノードに亘って分割すること、コンパイラの分析フェーズの表示を更に統合すること、の何れについても述べてはいない。

【００１２】

１９９１年、並列中間表示を備えた共有メモリ又はメッセージベースの並列プロセッサを使って、コンパイル、シミュレーション、実行に関する実行可能性を調べた研究報告が更に発表された。この報告は、ノードを表すフィールド及び特定のノード上の中間表示アドレスを表すフィールドから成る組（レコード）を使って分析形式の中間表示内に各ポインターを配置することにより、中間コンパイラ表示を分散することを示唆している。この報告は、インクレメンタルなコンパイレーションに関する複雑さと可能な手法を調べたものである。

【００１３】

１９９３年の研究報告は、１９９１年の報告を進展させ分散型のポスト分析の中間表示について述べているが、中間コンパイレーション内におけるインプリメンテーションの詳細と、ポストエラボレーション及びポストオプティマイゼーション（インライニング）再配分のプロセスについては触れてない。この報告は、複数のコンパイレーションフェーズ、シミュレーション又は実行に亘る単一のコンパイラ指向データベースについても、並列データベース表示についても触れていない。

【００１５】

要するに、コンパイラ及びシミュレーション／実行の作動モードを備えた装置が望ましく、本装置により、プロセッサ間におけるシミュレーション／実行とコンパイルのために必要な特定の情報への全体的なアクセスが効果的に行われるようになり、ノードのような一つ又はそれ以上の処理ノードに関する随意的な再構成可能な論理とメモリを利用できるようになる。こうした装置及び作動モードならば、全体の最適化とインクレメンタルな再コンパイレーションの機会を提供しつつ、単一ノード上でのコンパイレーションよりも大規模なコンパイレーション及びシミュレーション／実行ができるようになり、これによりシミュレーション／実行と同様にコンパイルに必要な時間が減少する。

【００１６】

（発明の概要）

その後の報告は、クライアントを備えた分散型のコンパイラ指向データベースを開示し、

・ソースアナライザ（コンパイラ構成要素）と、
・エラボレータ（コンパイラ構成要素）と、
・オプティマイザ（コンパイラ構成要素）と、
・コード生成器（コンパイラ構成要素）と、
・アセンブラ（コンパイラ構成要素）と、
・リンカー（コンパイラ構成要素）と、
・実行時間システム（シミュレーション／実行構成要素）と、
・デバッガー（シミュレーション／実行構成要素）と、
・プロフィーラ（シミュレーション／実行構成要素）と、
・イベントログ（シミュレーション／実行構成要素）と、
・図表ツール（様々なフェーズの構成要素）と、

をクライアントとして挙げている。

本報告は、複数のノードを有するコンピューター上のコンパイレーション及びシミュレーション／実行にまたがる単一のコンパイラ指向データベースの概念を導入したものである。

本発明が開示するコンパイラ指向データベース及びクライアント装置は、コンピュータープログラムとハードウェアモデルについてコンパイレーションモードと、シミュレーションモードと、実行モードとを含んだ効率的且つ密接に統合された作動モードとを提供す

る。本発明は並列コンピューターシステムの一つ又はそれ以上のノードを利用し、各ノードは、再構成可能な論理装置により随意的に増強された単一のプロセッサ又は共有メモリのマルチプロセッサである。本発明の利点は、コンパイレーション又はシミュレーション／実行の作動に関連した中間情報への全体的なアクセスを提供しながら、単一ノードの場合よりもはるかに大規模なコンパイル能力を提供できる点である。この全体的なアクセスにより、コンピュータープログラム及び／又はハードウェアモデルを（再）コンパイル及びシミュレート／実行するのに必要な時間がかなり減るものである。

【００１７】

（好適な実施の形態の説明）

本項は、発明者にとって現在は既知である本発明の最良の実施の形態を述べるが、構成要素の機能を達成するための代替手段を当てることにより本発明を実施する多くの関連する実施の形態があることを、当業者であれば理解できる。

基本ハードウェア装置を図１，２，３に示す。図１は装置全体の構成を示す。図２は随意ブロック９の詳細を示す。図３はブロック１及びブロック２内の随意的な再構成可能論理を示す。

【００１８】

図１において、ブロック１から８は従来のキャッシュコヒーレントマルチプロセッサを表す。ブロック２，４，６を省略すると、ハードウェア装置は単一のプロセッサノードの形となり、ブロック１，３，５，７，８を含む。この従来型ユニプロセッサにメッセージインターフェース（ブロック１０）を加え、ＩＥＥＥ標準１５９６「スケーラブルコヒーレントインターフェース」のようなプロトコルを使う従来型の大規模並列処理計算機を提供する。当業者に周知の従来手段を使うグラフィカルユーザーインターフェース、ローカルエリアネットワーク、ディスクサブシステムのような従来のＩ／Ｏ装置が存在する（図示せず）と仮定する。

【００１９】

再構成可能な論理ブロックであるブロック２２は、従来型ユニプロセッサ、共有メモリのマルチプロセッサ、又は上記の大規模並列処理プロセッサに随意的に挿入される。独立型装置として様々なバリエーションの再構成可能な論理ブロックが利用でき、個別のセルの動作と相互接続の両者が変更（再構成）されるが、装置はシステム中へインストールされ、周期を基準にシステム中へインストールされることもある。

コンパイラ指向データベースによりコンパイレーションとミュレーション／実行をサポートするために、一つ又はそれ以上の再構成可能論理ブロックが、共有メモリインターフェース（ブロック９）、メッセージベースインターフェース（ブロック９）を用いるか又は一つ又はそれ以上のプロセッサ（図３のブロック３０及び２２）が利用できる実行パイプラインの一部として、システムに随意的に組み込まれる。一般性を失うことなく、一つ又はそれ以上の再構成可能論理ブロック（ブロック２１）を、コンパイラ及び／又はコントローラが認識できる従来型のメモリアレイ、固定論理装置（ＡＤ変換器等）に置き換えることができる。

【００２０】

図９は、ローカルな相互接続（ブロック２０から７へ）への共有メモリ又はメッセージ伝達インターフェースを通してハードウェア装置に組み込まれた一つ（又はそれ以上）の再構成可能論理装置（ブロック２１）の内部構成を詳しく示している。ブロック９とブロック２１の動作と相互接続を再構成するために、再構成の情報がブロック９から書き込まれる（又は読み取られる）ように、一つ又はそれ以上のアドレスが相互接続（ブロック７）メモリマップ又はＩ／Ｏマップにマップされる。そのような相互接続情報を書き込む（及び読み取る）ための一つの手段では、各再構成可能論理ゲート／要素の構成及び相互接続ポイントに特定のメモリアドレスをブロック２１内で割り当てる。アドレスから書き込まれた（又は読み取られた）値は、ゲート／要素の作動又は相互接続のルーチングを決定する。代替手段では二つのアドレス、即ち、特定のゲート／要素又は相互接続のルーチングポイントを規定するために用いられるアドレスと、前記ゲート／要素又は相互接続のルー

チングポイントの値を規定するためのアドレスとを関連付けることが行われる。もしくは、同じ構成情報がメッセージ相互接続（ブロック１３）とメッセージインターフェース（ブロック１０）を通じてプロセッサにより書き込まれて（読み取られて）もよい。論理ブロック９を再構成する別の手段として、マッピングＩ／Ｏ及びメモリマップＩ／Ｏ装置の広範な履歴から、共有メモリマルチプロセッサ又は大規模並列プロセッサシステムアーキテクチャに至るまで、様々なものが当業者に知られている。

【００２１】

再構成可能論理（ブロック９）が一つ又はそれ以上のプロセッサで実行されるソフトウェアにより一旦構成されると、インターフェースブロック９が使えて、再構成可能論理（ブロック９）は、共有メモリ及び大規模並列プロセッサが従来から使っている同じ形態の相互プロセスと相互プロセッサの通信を使い、（組み込まれた）プロセスの動作を評価できるようになる。これらの仕組みは当業者に周知であり、ポイントツーポイントやマルチキャストのメッセージ、割り込み、及び共有メモリへの一貫したアクセス等を挙げることができる。本質的には、一旦再構成されたブロック９は、プログラムが固定され高度に並列なマルチタスクプロセッサとしてシステムアーキテクチャに関わる。

【００２２】

もしくは、上記の再構成可能な論理ブロック（ブロック９）は、図３に示すように一つ又はそれ以上のプロセッサ（ブロック１）の指示実行パイプラインに組み込まれていてもよい。一つ又はそれ以上の指示セットエンコーディング（一般的に「オプコード」として知られている）は、オプショナルコプロセッサ、相互接続、アービタ、再構成可能な論理ブロック（ブロック９及び３０）により実行される機能のために取って置かれる。プロセッサ内で指示ストリームを実行中にオプコードが登場すると、オプコード、拡張オプコード（もしあれば）、入力オペランドはプロセッサ（ブロック１）により取り出され、コプロセッサインターフェース、相互接続、アービタを経由し、適当に構成された再構成可能論理ブロック（ブロック９）に送られ、再構成可能な論理ブロックは、プロセッサが供給した情報で示される作動を実行し、後の適当な時点で作動開始したプロセッサ（ブロック１）へ完了状態とオプション結果を返却するので、当該プロセッサは最初の指示を完了することができる（指示を外すことも可能）。アービタとインターフェースの詳細は、６８００ファミリーコプロセッサインターフェース又は密接に組み込まれた専用インターフェースのような設計に見られるように、当業者には周知である。

【００２３】

再構成可能な論理ブロックにより実行された指示は、再構成可能な論理ブロック、アクセスメモリ、コントローラ又は再構成可能な論理ブロック内に組み込まれている装置内のゲート／要素動作又は相互接続ルーチングを再構成するか、又は再構成可能な論理ブロック内の以前の作動から生じている読み取り状態を再構成するために使われる。共有メモリ及びブロック９のメッセージベースの使用に関して言えば、コンパイラが従来のプロセッサのためにファイル（又はメモリ）内に指示を出す場合、コンパイラ指向データベースのクライアントは、インストールされた再構成可能な論理ブロック（ハードワイアで接続された構成状態を通した能力を含む）を認識する段階と、適切な再構成情報を作成する段階とに従来通り責任を負っている。

【００２４】

次に基本ハードウェア装置のメモリ（ブロック８）、キャッシュ（ブロック３と５）、最終プロセッサ（ブロック１）は図４に示すように、分散型コンパイラ指向データベースを実行させるようにプログラムされ、クライアント装置と関連付けされる。次に最終装置は、より短時間でより大規模なプログラム／ハードウェアモデルをコンパイル及びシミュレーション／実行するために、シミュレーション／実行の作動モードだけでなくコンパイレーションモードでも使われる。

【００２５】

分散型コンパイラ指向データベースは、本発明のコンパイレーション及びシミュレーション／実行作動モードに特に適した中間表示データベースオブジェクトのコレクションを保

持している。データベースオブジェクトの実例としては、リテラル識別子「ｉ」を表すオブジェクト、宣言、式、並列ステートメント、エラボレートされたプロセス、エラボレートされたドライバ、現にシミュレート／実行されているプロセスの瞬間的な状態、相互プロセスメッセージの待ち行列の時間／数値変更の履歴、又はあるインターバルに亘って共有される変数又は実行中のプログラム及び／又はモデルによって読み取り又は書き込みを行うために開かれているユーザー定義ファイルの内容等を挙げることができる。データベースは、複数の処理ノードのメモリ、スワップスペース、ファイルシステムの間に分散される。各オブジェクトは、分散型コンパイラ指向データベースに亘って一貫性がある固有の観測可能値を有し、この値は現在どの計算ノードが基準であるかによらず一貫している。

【００２６】

分散型コンパイラ指向データベースを実行するための一般的な手法は、汎用並列データベース又は分散型共有メモリに関する文献から当業者には周知である。本発明の固有の態様は、汎用データベースシステムには存在しない（ブロック４０から５０の例に示すような）コンパイラ作動モード及びシミュレーション／実行作動モードのための装置と、コンパイル作動モード及びシミュレーション／実行作動モードに必要な特定のコンパイラ又はシミュレーション／実行オブジェクトをキャッシュするための装置とを含んでいる。これとは対照的に、分散型共有メモリシステムでは、コンパイル時間表示の単一宣言領域内の宣言ツリーのような、意味論からすれば意味はあるが隣接しないオブジェクトのコレクションではなく、長さと細分性間隔とが一定の隣接するアドレススペースへの遠隔アクセスが提供される。

【００２７】

分散型コンパイラデータベースインターフェースでは、プログラミング言語インターフェース（ブロック５１）が提供され、このインターフェースを通じ（ブロック４０から５０の例に示すような）クライアントは、既存オブジェクトのコピーを作成し、要求し、細かな更新をし、コンパイラ及びシミュレーション／実行に関連するデータベースオブジェクトのコレクションを解除し解放する。このようなプログラミング言語インターフェースを実行する一般的な手段は、以前に公表された文献により当業者に周知であり、ＶＨＤＬハードウェア記述言語を実行するユニプロセッサシステム用のＡＩＲＥ／ＣＥプログラミング言語インターフェースのためのものを挙げることができる。

コンパイラ指向データベースとの接続に適した幾つかのコンパイラ及びシミュレーション／実行クライアントを評価するために、図示の処理ノード（ブロック５２）内のブロック４０から５２で示したクライアントについて以下の段落で述べる。

【００２８】

しかし、図示の全てのクライアントが所定の処理ノード上に存在する必要はなく、又コンパイル又はシミュレーション／実行に関連する別のクライアントが一つ又はそれ以上のノード上のコンパイラ指向データベースと接続されていても差し支えないことに注意されたい。特にノード次第で、ユーザーインターフェース・クライアントを含む傾向のノードもあれば、エラボレータ、オプティマイザ、又はコード生成器のような背景処理クライアントを含む傾向のノードもある。

【００２９】

ソースコードアナライザ（ブロック４０）は、プログラム又はハードウェアモデルを構成するフラッグメント又は完全ソースコードファイルをコンパイラ指向データベースの表示に翻訳することを指令する分散型コンパイラ指向データベースインターフェース（ブロック５１）から指令を受ける。特定のファイルを分析する間に従属関係に出会うと、例えば別の宣言領域を参照するＶＨＤＬ使用クローズに出会う場合であるが、コンパイラ指向データベースが使用済み宣言及び関連データベースオブジェクトのローカルコピーを提供するまで、分析を遅らせることが必要となる。後続の遅れを最小にするため、ソースアナライザがある計算ノード上で、ある宣言領域を使用すると、参照されたこの宣言領域はソース分析を行っている他のノードへ順向的に移送されるということが起こる。こうしたキ

ャッシングの発見的手法は、作動中に学習された情報に指令された言語スペシフィック且つユーザースペシフィックなものであり、分散型コンパイラ指向データベースの持続的な形式を使って記憶される。

【００３０】

エラボレータ（ブロック４１）は分散型コンパイラ指向データベースから取り出した以前の分析済み情報を書き直し、設計とサブプログラム／機能の階層を選択的に平坦化するか、又は、サブプログラム／機能の定義とそのプログラム又はハードウェアモデルの他の部分内の呼出個所との間の拘束を伝播するようになっている。エラボレータは、データベース全体に亘る可視性と、分析前後で同じものを保持するデータベースオブジェクトのコレクションを再利用する能力とを使用する。例えば、アナライザは、例が作成されるか又はサブプログラム／機能が呼び出されるポイントに影響を受けてタイプとサイズが様々に変化する宣言セットから、タイプとサイズがプロセス中で一定している宣言セットを分離する。次にエラボレータは、メモリとオブジェクトを作成／維持するためのオーバーヘッドを減少させるために、分散したコンパイラ指向データベースを利用して、特定のタイプ又はサイズを有するオブジェクトのための新しくエラボレートされたデータベースオブジェクトを作成する。エラボレータクライアントは、別のノードにおける構成要素例、オブジェクト例、又はサブプログラム階層の二次的なエラボレーションを作成するために分散データベースを活用する。分析の間に見積もられ、データベースと共に記憶されエラボレーションツリーへ繰り返し伝播された複雑な距離を使いながら、エラボレータは、別の処理ノード上にエラボレーションを作成するために、発見的手法を使用し、エラボレーションの結果がプロセッサ向けのプログラム又は設計の良好な第一パス区分となるようにする。

【００３１】

オプティマイザクライアント（ブロック４２）は、発明者が以前の文献〔ウィリス９１、ウィリス９２〕で説明したような分散型の全体分析と書き直し変換とを可能にするために、設計データベースへの全体的な可視性を利用する。全体的に分析されエラボレートされたシミュレーション／実行データベースへの選択的且つ全体的な可視性がない場合、オプティマイザは不完全な情報を使って判断しなければならず、結果は一般に不完全な最適化となる。データベースを通じて他のオプティマイザと直接通信すれば、サブプログラムのデータフロー分析と同様に中間の最適化分析を共有できる。開示された分散コンパイラ指向データベースがない場合、個別の並列コンパイレーションで実行されるオプティマイザは、共通の構成要素とサブプログラムに関する分析を頻繁に作成し直さなければならない。

【００３２】

マッピングクライアント（ブロック４３）は、並列エラボレーションの間に開始された処理ノードへのワークロードの割り当てと、各処理ノード上の複数の実行スレッド（プロセス）に関する準静的なスケジュール評価とを繰り返し行う〔ウィリス９５〕。コンパイラ指向データベース中で注釈を付けたクリティカルパス分析は、有効且つ準静的なマッピング及びスケジューリングを達成することのできる全体的な視点を再度提供する。シミュレーション又は実行のワークロードを抱える各処理ノード上のマッピングクライアントは、シミュレーション又は実行の速度が最大になると予測されるロードバランスに達するまで、ワークロードを交換し合う。こうした全体的なコンパイル時間情報がない場合、シミュレーション／実行の環境は実行時間のロードバランスに左右され、結果的に実行時間のオーバーヘッドが大きくなりシミュレーション／実行の性能が低下する。

【００３３】

コード生成クライアント（ブロック４４）は、擬似コード情報、プロセッサ実行可能情報、又は再構成可能な論理構成情報を、コード生成器のアドレス・スペース、同じ処理ノード上の関連するオペレーティングシステムのプロセス又は後で実行される従来のファイルへ直接発することにより、マッピング及びスケジューリングを追跡する。コード生成器又は密接に連結されたプロセスとしての同じオペレーティングシステムのプロセスへ実行可能な構成情報を直接発することにより、分散型コンパイラ指向データベース中で起こる変

化に対し、迅速でローカル化され且つインクレメンタルな応答が可能になる。変化は、記号デバッグ、プロフィーリングコードの挿入、故意のフォールトの挿入（フォールトシミュレーション）、又はシミュレーション／実行に影響を与えるデータベース上の他のデルタに関する区切点で起こる。

【００３４】

リンカー／ローダー（ブロック４５）は、記号値のアドレスから実際のメモリ中のアドレスを導き出すために、キャッシュ性能を改良するためにコードフラッグメントを突き止めるために、キャッシュ制御指示を実行可能命令中に挿入するために、アドレススペースであり、またコード生成器のターゲットの中へ外部ライブラリをリンク／ロードするために、コンパイラ指向データベースとコード生成器の両者と密接に働く。共通のコンパイラ指向データベースを使うことにより、ファイル又は一方向のパイプラインを通して通信する従来型の個別のオペレーティングシステムプロセスでは実現することができなかった迅速、かつきめ細かな協調をコード生成器とスケジューラーの間に提供することができる。

【００３５】

実行／シミュレートクライアント（ブロック４６）は、ローカルメモリ、共有メモリ、及び再構成可能論理を適切にロードし実行を制御するために、コンパイラ指向データベースからの記号情報と制御情報との両者を利用する。実行／シミュレートクライアントは同様に、コード生成器とリンカー／ローダーがターゲットとして使用するアドレス及び全体的なコンパイラ指向データベースへのきめ細かなアクセスから利点を得る。現にシミュレートされているプログラム又は設計の修正を要求するプログラミングインターフェースは、シミュレート／実行アドレススペースからの呼出により容易にしかも効率的に適応でき、結果的にコンパイラ指向データベースが変更されインクレメンタルな再コンパイルコマンドがコード生成器とリンカー／ローダークライアントに送られる。このような密接な修正は、コンパイレーション環境と実行環境とを同時に作動せず双方向の通信をしない従来型の環境では実現できなかったものである。従って、従来の環境では所要修正をできるようにするために最初に発生したコードは実質的な柔軟性を必要とし、それ故にシミュレーション／実行の性能が下がっていた。制御指令は、ユーザーインターフェイス又はデバッガーで発生し、コンパイラ指向データベースを通じて、並列コンパイレーション及びシミュレーション／実行に関与する各処理ノード上の各実行／シミュレートのクライアントと一貫した通信を行う。

【００３６】

デバッガークライアント（ブロック４７）は、実行と区切点を指令し状態を設定し状態を読み取るために、ユーザーとコンパイラ指向データベースの間でインターフェイスする。従来のデバッガーは実行可能命令に直接到達するが、コンパイラ指向データベースの手法の場合、デバッガーは（適当なユーザーインターフェイスを備えた）一方の処理ノードからコンパイラ指向データベース中に適当な変化を作り、ソースコードアナライザがコード生成器とリンカー／ローダーを通して見ることのできるデータベース経由でコマンドを開始することができる。相対的に僅かなコードが所定のシミュレーション／実行の作動中にデバッグされるので、この手法により殆どのコードは最大性能を狙って生成することができる。デバッグが必要な場合、準最適な性能であるが最初のソースコードと良好な相関を有する実行可能命令から成る領域を生成するために、適当なフラッグメントコードを再構成でき状態をマッピングすることができる。コンパイラ指向データベース中の全情報の助けを得るコード生成器とリンカーは、スタックエリア、信号待ち行列エリア、静的エリアの上に状態をマップするために最適化されたコードシーケンスとデバッグコードシーケンスの間に装着されている。従来型の記号表だけでガイドされた実行可能画像上で直接作動するデバッガーの場合は、このようなマッピングは困難又は不可能である。

【００３７】

デバッグシミュレーションは複雑な最適化／マッピング／スケジューリングの変換であり、この変換では可能な場合にローカルな「サイクル駆動」モードでハードウェアモデルを実行することが試みられ、ローカルなタイミング詳細ではない情報従属情報が保存される

。デバッグ機能、プロフィーリング機能、ユーザーインターフェース機能がシミュレーションのフルタイミング部分からサイクル駆動コードのエリアに入ると、ハードウェアモデルの可視的意味を保存するには、サイクル駆動状態からフルタイミング意味論へローカルコード及び状態を変換する必要があるが、この変換はコンパイラ指向データベースを即座に入手することにより実行できる。一般にタイミング状態は、デバッグ用のフルタイミング状態を生成するために、シミュレーションを既知の状態から新たなフルタイミングコードへとローカルに再実行することにより生成される。このようなデバッグ指向のインクレメンタルなタイミング能力は、実質的なタイミング詳細の選定された部分のみがハードウェアモデルの性能にとって重要であるような、そうした実質的なタイミング詳細を有するライブラリをコンパイルする場合には、性能上の重要なソースである。このような状況は、ＶＨＤＬのＶＩＴＡＬタイミングプリミティブをコンパイルしシミュレートする際にしばしば起こる。デバッグ後、プロフィーリングや他のインターフェースは問題のコード領域での動作を止め、シミュレーション性能を高めるために、高性能コードインプリメンテーションへの復帰がなされる。実行可能命令へのこうした迅速でローカルな変更は、データとコマンド通信の両者に関係するきめ細かなコンパイラ指向データベースの構造に左右される。

【００３８】

プロフィーラクライアント、ブロック４８は、総計シミュレーションと実行動作を調べるための手段をユーザーに与える。これは、通常は少数の実行可能命令／シミュレーションの領域を詳細に調べるデバッガーと好対照をなす。プロフィーリングは、特定のシミュレーション／実行の作動により実際に実行されたコードパスウェイ、各実行可能命令／シミュレーションの領域で費やされた時間に関する報告をし、特定状態に割り当てられた値の総計特性を考慮することも行う。複雑なプロフィーリング規準が設けられている場合は特に、コンパイラ指向データベースによりもたらされた、実行可能命令／シミュレーションを迅速に変更する能力が非常に重要である。

【００３９】

ユーザーインターフェースクライアント、ブロック４９により、ユーザーはコンパイレーション及び実行／シミュレーションの全プロセスを制御できるようになる。ここでもコンパイラ指向データベースを通じ他のクライアントとインターフェースすることにより、非常に多様なユーザーインターフェースが他のクライアントインターフェースに直接左右されることなく作成される。これにより、コンパイラ制御インターフェース、コマンドインタープリタ、概略表示スキーム、波形ディスプレイ、解析ツールを個別に作成することが容易になる。情報プロトコルとコマンドプロトコルはコンパイラ指向データベースのソケットによって定義され、他の特定のクライアントはインストールされていないので、ユーザーインターフェースの作成により、非常に簡素化されたインターフェースが得られ、ユーザーインターフェースの作成コストが下がるか又は機能が高められるかの何れかの利点が得られる。

【００４０】

ファイルインターフェースクライアント、ブロック５０によって、メッセージ相互接続に亘ってファイルＩ／Ｏを分散させることと、実行／シミュレーションで得られたファイルＩ／Ｏを一貫して再アセンブリことができるようになる。コンパイラ指向データベースを通じてファイルアクセスすると、合成タイプのＩ／Ｏの実行時間合成に対してユーザーが定義したタイプに応じて最適なＩ／Ｏルーチンを生成することが容易になり、ファイルインターフェースは、ハードウェア上で実行され混ざり合っている読み取りと書き込みを、個別で互換性のないデータタイプの符号化を用いて直接処理しなくて済む。重要な作動ステップは、通信インターフェースも現に送信されている情報についての意味論的な情報を有していることであり、これは、コンパイレーションと実行／シミュレーションを分離している従来型の環境に欠けていたことである。

【００４１】

上記クライアントは、新しい固有の装置と作動モードとを示しているが、これは従来の粗

い一方向のファイル（又はパイプライン）のリンケージではなく、コンパイラ指向データベースを通じてクライアントを連結することによりできるようになったのである。並列データベースの設計やこうしたコンパイラ指向データベースの構造に携わる当業者の中には、従来技術を延長するだけで実質的技術を引き出せる者もいるであろう。しかし、論点を明確にするため以下のセクションで、分散型のコンパイラ指向データベースを実行する手段について述べる。

【００４２】

図５から９に、分散型のコンパイラ指向データベースを実行する一つの手段を示す。こうしたデータベースを実行する際の最も重要な問題の一つは、最初に作成されたか又は現に他のノード上に「常駐する」オブジェクトを参照する効率的な手段を提供することである。過去の手法は、直接又は単一メモリアドレス内で各ポインタを<ノード、アドレス>の組として指定する段階を含んでいた。第一の手法では、組の中のノードデジグネータが現在のノードに対応する否か、又は遠隔ノードが必要なのか否かを判断するために、データベース中の各ポインタへアクセスするのが遅れてしまう。第一の手法の場合、各ポインタはサイズがメモリアドレスの２倍となり、データベースの多くはポインタで構成されるので、データベースを表すのに必要なメモリは実質的に二倍になる。第二の手法では、スペーススペナルティはなくなるが、ポインタによりアドレスされるアドレス範囲に対する全体のデータベースのサイズが制限される。第二の手法では、必要なオブジェクトがローカルに一旦キャッシュされた後に誤ったアドレスをバックパッチできるようにする非常に複雑で、マシンに左右されるトラップハンドラーも必要となる。トラップハンドラーはコードの重要部分であり、当該部分内では非ローカルな応答に対して待機する間に妨害が起こらないので、コールバックは、非ローカルなオブジェクトが到着した後にパッチが起こるようにするための実用的要件となる。上記の欠点を考慮し第三の手法を本発明の好適な実施の形態で開示する。

【００４３】

データベース中の各オブジェクトは特定の中間表示タイプ（ブロック６３のような）と関連付けられている。関連付けの手段には、クラス定義に対するインプリメンテーション内部ポインタ（例えば、C++「ｖｐｔｒ」）、タイプを表す明確な整数、又は列挙法数値を始めとし、様々なものがある。ＩＲオブジェクトタイプを参照することにより、有効な演算子と内部データのセットが定義される。共通のＩＲオブジェクトタイプの例としては、リテラル、宣言、コレクション、名称、式、ステートメント、ブロック構成等が含まれる〔ウィリス９６〕。従来のコンパイラが内部で共通に利用できるＩＲタイプ以外で、分散型のコンパイラ指向データベースに必要な追加ＩＲタイプとして、遠隔オブジェクト代理、相互クライアントコマンド、相互クライアント応答、シミュレーション／実行状態（シミュレーション／実行内におけるスタックフレーム、静的エリア、通信に関する表示を含む）を挙げることができる。

【００４４】

多くのポインタはノードに対してローカルなオブジェクトを参照する（ブロック６１で表されるローカルオブジェクトを指すメモリアドレス、ブロック６４のように）。少数のポインタは一般に、遠隔（又は代理）オブジェクトにより、他のノード（ブロック６２で表されるタイプ１の遠隔オブジェクトを指すブロック６５のメモリアドレスのように）上にあるオブジェクトを参照する。第一の種類の遠隔オブジェクトには、現に表されている遠隔オブジェクトの実際のタイプ（ブロック７０）、基準コピーが存在するノード（ブロック７１）、遠隔ノード上のメモリアドレス（ブロック７２）が含まれている。従って、ポインタの大多数で、単一メモリアドレスに必要な最低限のメモリを占有することになる。少数のオフノード基準が占有する記憶装置のサイズは、<ノード、アドレス>の手法のサイズ（ブロック６２に示すように）のほぼ２倍である。

【００４５】

図６で、実際のオブジェクトタイプ（ブロック７０）に対して適切な、タイプ１の遠隔オブジェクト（ブロック６９）に対する参照（又は方法呼出）が行われる場合、分散型のコ

ンパイラ指向データベースは、遠隔（又は代理）オブジェクトをタイプ２基準に変更するが、このタイプ２基準では、ローカルに「キャッシュされた」コピー（ブロック８７）に対するポインタがオブジェクトタイプ（ブロック７０）に取って代わる。一つのメモリだけ置き換えることにより、キャッシュされたコピーは、ローカルなノード上に常駐しているかの如くローカルに機能する。使用パターンは勿論、参照されたオブジェクトのタイプ、利用可能なメモリと帯域巾に基づき、分散型のコンパイラ指向データベースは、直に請求されたオブジェクト以上のローカルにキャッシュされたコピーを取り出すためにドメイン情報を使用する。例えば、宣言領域に対して選択した基準により、一つの宣言ではなく宣言領域の全内容を転送することが始まる。第二の例では、ある宣言を取り出すことにより、宣言のタイプを検索することが始まる（及びタイプの定義を繰り返し検索することが始まる）。同じ様な方法で、分散型のコンパイラ指向データベースは、ローカルメモリを得るため又は「キャッシュの一貫性」のオーバーヘッドを減らすために、遠隔オブジェクトをタイプ１へと逆に切り換えることにより、キャッシュされたコピーを削除することを選択する。

【００４６】

図７に示すようにオブジェクトの中には、更新のためにローカルにキャッシュされるものがあるが、このような場合、一つのコピーのみが如何なる時点でも変化することができ、他の全オブジェクトは一貫性のある状態に保たれてなければならない。例えばユーザーインターフェースをサポートする処理ノードはコマンドストリーム（リスト）に書き込みを行うが、このリストはコマンドを実行する全ての処理ノードに対して一貫して可視的でなければならない。図７に示すように、あるタイプ２の遠隔オブジェクトは、別のタイプ２の遠隔オブジェクト（ブロック１１２と１１３とでブロック１１９を表す）を表す。第二のタイプ２の遠隔オブジェクトは順次、ローカルなメモリアドレス（図示）であるか、又は第三のノード（図示せず）上のオブジェクトを表すためにノードＩＤとメモリアドレス（ブロック１１７と１１８）とを使用する。

【００４７】

一貫性を維持するために、分散型のコンパイラ指向データベースは図８に示すように、データベースオブジェクトを第一の非ローカルコピーに関連づけしなければならない。ノード上の多くのオブジェクトはローカルに参照されるだけなので、こうした情報をデータベースの全オブジェクトに詰め込むことは（論理的には正しいであろうが）スペースの点で効率がよくないであろう。第一の非ローカルコピーのノード識別子を突き止める手段に代わって、最初のオブジェクトを所有している処理ノード上に書き込み状態とメモリアドレス（遠隔ノードの）を維持する。フル関連ハードウェア又はソフトウェアハッシュ表を始めとする、このような多くの手段が当業者に知られている。

【００４８】

多くのキャッシングスキームは当業者に既知であり、上記のスキームでは、コンパイレーション用に一貫性のある分散型データベースを実行する一つの手段のみを説明した。キャッシングの文献から多くの他の手法を、発明全体に必要なキャッシングメカニズムを実行する手段として適合させることができる。

【００４９】

幾つかのオブジェクト、例えば、言語又は環境によって定義された所定記号のような、整数リテラル、浮動小数点リテラル、文字リテラル、ストリングリテラル、他の正規のオブジェクトは一般に、処理ノード毎に一つの表示しか持っていない。例えば、整数３については一つの表示しかない。更新のために、このようなオブジェクトの一貫性を保つことは必要でない。従って、分散型のコンパイラ指向データベースの最適化は、非ローカルなキャッシングを実行することではなく、正規のオブジェクトをローカルな表示にマッピングすることによって行われる。こうした手段を図９に示す。

【００５０】

時間が経過すると、データベース中のオブジェクトへ到達することができなくなる。可能であり安全な場合、こうしたオブジェクトの全てのキャッシュされたコピーと最初のオブ

ジェクトを削除することが役に立つ。データベースの作動は、当業者に既知の基準カウント又は不要情報コレクションの他の形式の何れかを維持することで加速される。

【００５１】

図１０は、コード生成器（ブロック４４）と、リンカー／ローダー（ブロック４５）と、実行／シミュレーション（ブロック４６）の修正された形式を示す。クライアントとしての同じアドレススペースへコンパイルするのではなく、実行可能命令／シミュレーションは独立した作動システムプロセス中に存在する（ブロック１６２）。コード生成器（ブロック４４）、リンカー／ローダー（ブロック４５）、実行／シミュレーション（ブロック４６）の制御装置は、当業者には周知のメカニズムである、共有メモリメッセージ、相互プロセス通信メッセージ、又は作動システムメッセージ（１６３、１６４、１６５として示す）のような機構を通して、ノード上の個別のシミュレーション／実行作動システムプロセスを読み取り及び書き込みを行う。

コンパイラ指向データベースクライアントを実際の実行可能命令から分離すると、幾つかの重要な利点が得られる。第一の利点は、オペレーティングシステムが各プロセスに提供した全仮想アドレススペースを、データベース及びクライアント（一つのプロセス）とシミュレーション／実行可能命令（別のプロセス）の両者が利用できる点である。こうした分割により、古いアーキテクチャと２０億バイトの仮想メモリ領域に制限されたオペレーティングシステムを収容することができ、新しいアーキテクチャとオペレーティングシステム（関連するメモリの増大は伴うが）上で拡張アドレス表示をしなくて済む。第二の利点は、分割することにより、シミュレーション／実行の誤作動があってもデータベースとクライアントに関連するコード又はデータの構造を直ちに変更することはできないために、システムの保全性が高まる。最後に挙げる利点は、区分することによりコード又はモデルのセキュリティの問題が扱われることであり、このセキュリティ問題は、ライブラリ又は構成要素ハードウェアモデルを受信し、解読し、コンパイルし、より抽象的なコンパイラ指向データベース表示へ直接アクセスを試みる他のプログラム又はコードとリンクする場合に起こる可能性がある（恐らくリバースエンジニアリングが理由である。）適当なオペレーティングシステム防護手段を組み込めば、図１０のデュアルプロセス手法により、こうしたリバースエンジニアリングは非常に複雑なものとなる。

本発明の原理を好適な実施の形態で例示し説明したが、こうした原則を逸脱することなく本発明の配列や細部を変更できることが当業者には明らかである。そのような変更は全て、請求の範囲の範疇と精神に含まれるものである。

【図面の簡単な説明】

【図１】　随意的な再構成可能論理ブロックを有する二つの共有メモリマルチプロセッサノードを含む基本ハードウェア装置の実施例を示す。

【図２】　基本ハードウェア装置内の随意的な独立型の再構成可能論理ブロックの実施例を示す。

【図３】　一つ又はそれ以上のプロセッサの一部と接続されている随意的な再構成可能論理ブロックの実施例を示す。

【図４】　事例クライアントと分散型のコンパイラ指向データベース装置の関係を示す。

【図５】　データベースオブジェクトが、ローカルに分解された第二データベースオブジェクトと、コンパイラ指向データベース装置内の遠隔ノード上だけに存在する第四オブジェクトを参照する第三代理データベースオブジェクトとを参照する手段を示す。

【図６】　代理データベースオブジェクトが、遠隔オブジェクトのローカルコピーをコンパイラ指向データベース装置内にキャッシュする手段を示す。

【図７】　コンパイラ指向データベース装置内の複数の遠隔代理データベースオブジェクトに一貫性を提供するための手段を示す。

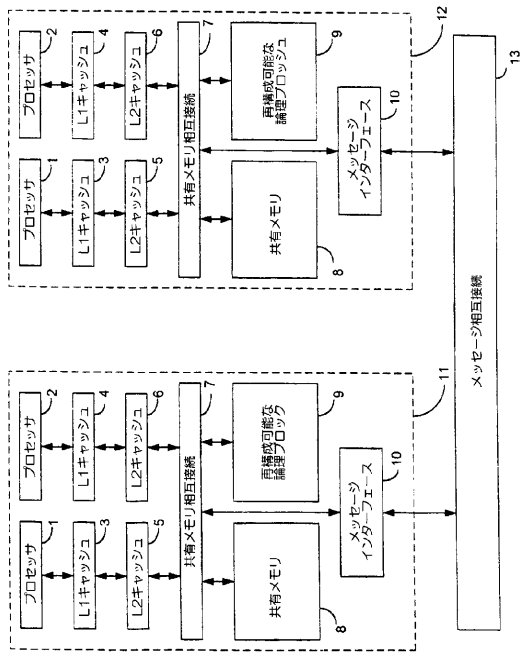
【図８】　特定オブジェクトの第一の非ローカルコピーをコンパイラ指向データベース装置内に配置するための手段を示す。

【図９】　正規に定義されたオブジェクトのローカル表示をコンパイラ指向データベース装置内に配置する手段を示す。

【図１０】　クライアント機能を二つ又はそれ以上の作動システムプロセスに分ける手段を示す。

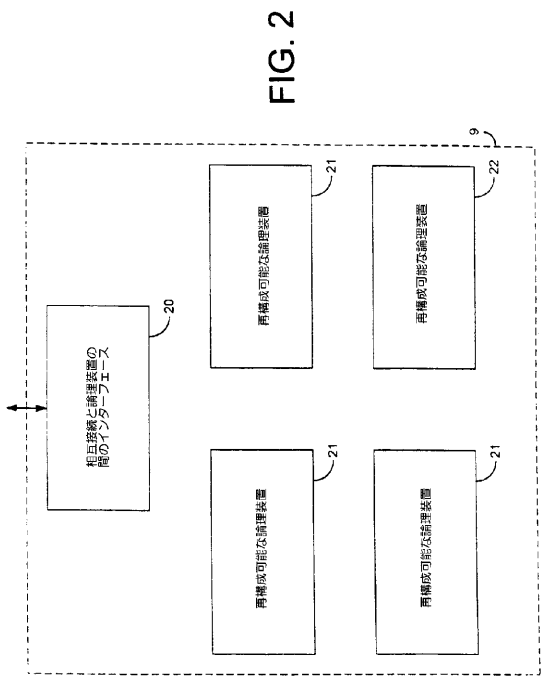
【図１】

FIG. 1



【図２】

FIG. 2



【図３】

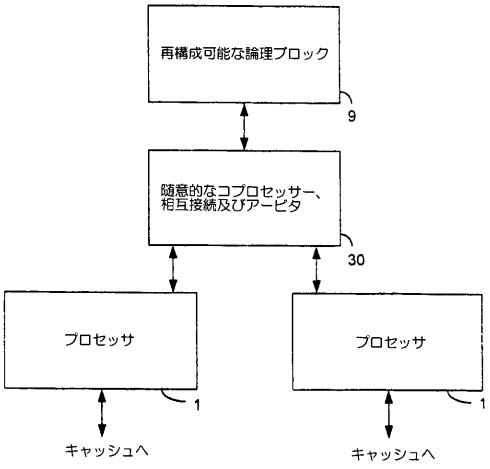
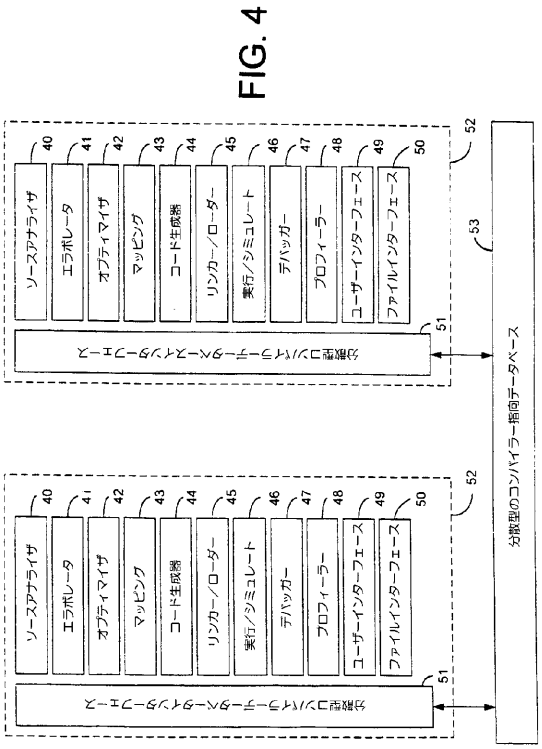


FIG. 3

【図４】

FIG. 4



【図５】

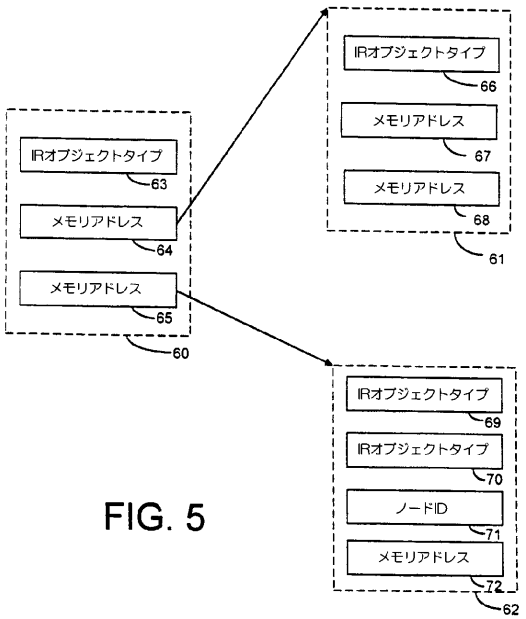


FIG. 5

【図６】

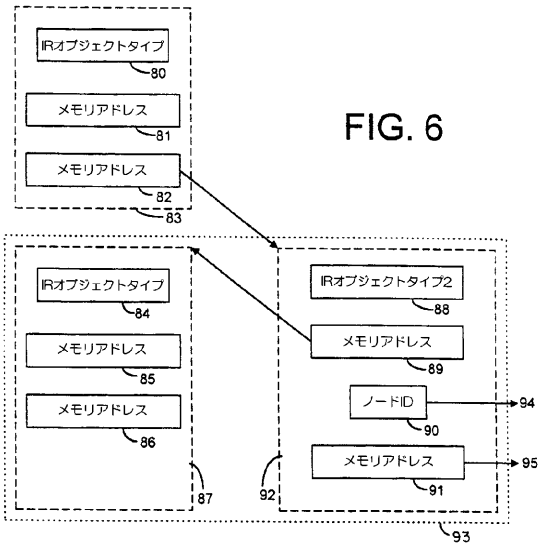


FIG. 6

【図７】

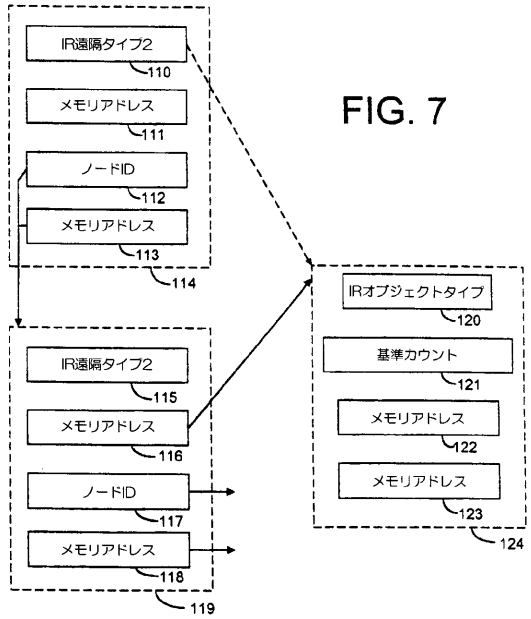
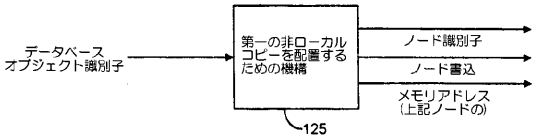


FIG. 7

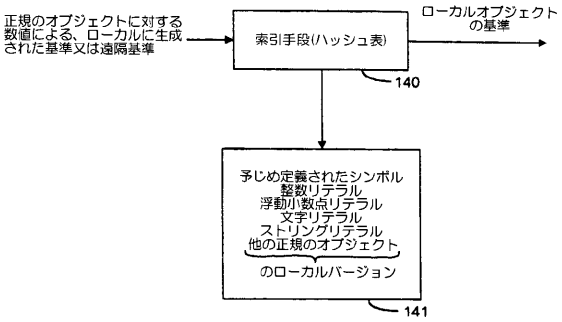
【図８】

FIG. 8



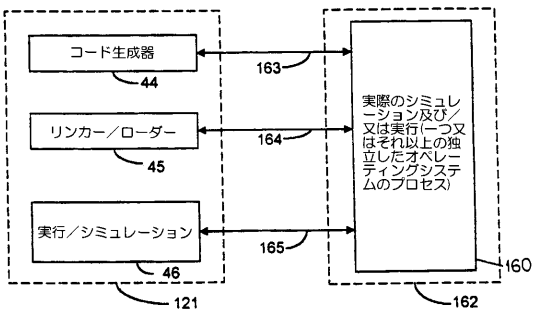
【図９】

FIG. 9



【図１０】

FIG. 10



フロントページの続き

(74)代理人 100096194
弁理士 竹内 英人
(74)代理人 100074228
弁理士 今城 俊夫
(74)代理人 100084009
弁理士 小川 信夫
(74)代理人 100082821
弁理士 村社 厚夫
(74)代理人 100086771
弁理士 西島 孝喜
(74)代理人 100084663
弁理士 箱田 篤
(72)発明者 ウィーリス ジョン シー
アメリカ合衆国 ミネソタ州 ５５９０６ ロチェスター シエラ レーン サウス ウェスト ９２４
(72)発明者 ニューシューツ ロバート エヌ
アメリカ合衆国 ミネソタ州 ５５９０６ ロチェスター ベアード レーン ノース イースト ９３１

審査官 殿川 雅也

(56)参考文献 米国特許第０４４９３０２９（ＵＳ，Ａ）
米国特許第０４６６１９０１（ＵＳ，Ａ）
米国特許第０５３９０３２０（ＵＳ，Ａ）
欧州特許出願公開第００７７２１４０（ＥＰ，Ａ１）
名古屋彰，新しい並列処理アーキテクチャとその設計技術，NTT R&D，日本，社団法人電気通信協会，１９９７年 2月１０日，Vol. 46，No. 2，第１５３頁乃至第１５８頁

(58)調査した分野(Int.Cl.，ＤＢ名)
G06F 9/45
G06F 9/46 - 9/54
G06F 11/28