(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第3870112号
(P3870112)
(45)発行日　平成19年1月17日(2007.1.17)　(24)登録日　平成18年10月20日(2006.10.20)

(51)Int.Cl.　F I
G06F 9/45 (2006.01)　G06F 9/44 322F

請求項の数 21 (全 25 頁)

(21)出願番号　特願2002-69221(P2002-69221)
(22)出願日　平成14年3月13日(2002.3.13)
(65)公開番号　特開2003-280919(P2003-280919A)
(43)公開日　平成15年10月3日(2003.10.3)
審査請求日　平成15年1月8日(2003.1.8)

(73)特許権者　390009531
インターナショナル・ビジネス・マシーンズ・コーポレーション
ＩＮＴＥＲＮＡＴＩＯＮＡＬ　ＢＵＳＩＮＥＳＳ　ＭＡＳＣＨＩＮＥＳ　ＣＯＲＰＯＲＡＴＩＯＮ
アメリカ合衆国１０５０４　ニューヨーク州　アーモンク　ニュー　オーチャード　ロード
(74)代理人　100086243
弁理士　坂口　博
(74)代理人　100091568
弁理士　市位　嘉宏

最終頁に続く

(54)【発明の名称】コンパイル方法、コンパイル装置、及びコンパイル用プログラム

(57)【特許請求の範囲】
【請求項１】
コンピュータが、記憶手段に格納されているソース・プログラムに基づき、プログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入された第１の中間表現プログラムを作成し、記憶手段に格納するステップ、
コンピュータが、記憶手段に格納されている前記第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出し、記憶手段に格納するステップ、
コンピュータが、記憶手段に格納されている前記ブロックについて、各命令をノードとし、依存関係にある命令のノード間にアークを設定するとともに、Ｓ－ＰＥＩの後にＨ－ＰＥＩ又は別のＳ－ＰＥＩを実行しなければならないという命令間の依存（以下、「例外依存」と言う。）の関係にある両命令のノード間のアーク（以下、「例外依存アーク」と言う。）を、例外依存以外の依存関係にある命令のノード間のアーク（以下、「非例外依存アーク」と言う。）に対して区別した第１の依存グラフ情報を作成し、記憶手段に格納するステップ、
コンピュータが、記憶手段に格納されている前記第１の依存グラフ情報に基づき、実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断するステップ、
コンピュータが、Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実

行された場合の方が例外依存アークを経由して実行された場合より有利と判断したときは、記憶手段に格納されている前記第１の依存グラフ情報に基き、該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成し、記憶手段に格納するステップ、

コンピュータが、記憶手段に格納されている前記第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成し、記憶手段に格納するステップ、及び

コンピュータが、記憶手段に格納されている前記第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成し、記憶手段に格納するステップ、

を有していることを特徴とするコンパイル方法。

【請求項２】

前記実行時間に係る有利性基準とは、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準であることを特徴とする請求項１記載のコンパイル方法。

【請求項３】

前記スケジューリングは、ブロック内のプログラムのクリティカル・パスを最小とするリスト・スケジューリングであることを特徴とする請求項１記載のコンパイル方法。

【請求項４】

非例外依存にはデータ依存又は制御依存が含まれることを特徴とする請求項１記載のコンパイル方法。

【請求項５】

遅くとも前記スケジューリングより前に実行される第１の追加ステップを備え、

該第１の追加ステップでは、記憶手段に記憶されている前記第２の依存グラフ情報及び第１の中間表現プログラム部分に基づき、コンピュータが、

前記投機実行がなされるように先端が別のノードに変更された例外依存アークの先端の当該別のノードに係る命令を番兵命令とし、

投機実行対象のＨ－ＰＥＩが、その例外発生を抑制された状態で前記番兵命令が実行されたときには、投機実行の対象となった命令列を、例外発生を抑制することなく再実行するように、前記第１の中間表現プログラム部分内の命令列を生成し、記憶手段に格納する、

ことを特徴とする請求項１記載のコンパイル方法。

【請求項６】

遅くとも前記スケジューリングより前に実行される第２の追加ステップを備え、

該第２の追加ステップでは、記憶手段に記憶されている前記第２の依存グラフ情報及び第１の中間表現プログラム部分に基づき、コンピュータが、

前記投機実行がなされるように先端が別のノードに変更された例外依存アークの先端の当該別のノードに係る命令をチェック命令とし、

投機実行の対象となった命令列のコピーを回復コードとして前記チェック命令の後に配置し、

これにより前記チェック命令が、その実行時には、投機実行対象のＨ－ＰＥＩにおいて例外発生が抑制されたか否かを判定し、該判定が正であれば、前記回復コードが実行されるように、前記第１の中間表現プログラム部分内の命令列を生成し、記憶手段に格納する、

ことを特徴とする請求項１記載のコンパイル方法。

【請求項７】

前記第１の依存グラフ情報には、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへ向けて設定された例外依存アーク（以下、この例外依存アークを「Ｓ－ＰＥＩ間例外依存アーク」と言う。）の設定情報が含まれ、

前記コンパイル方法は、遅くとも前記スケジューリングより前に実行される第３の追加ステップを備え、
前記第３の追加ステップでは、コンピュータが、
Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、該Ｓ－ＰＥＩ間例外依存アークを除去し、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、該Ｓ－ＰＥＩ間例外依存アークをそのまま残すように、記憶手段に格納されている前記第２の依存グラフ情報を変更する、
ことを特徴とする請求項１記載のコンパイル方法。
【請求項８】
記憶手段に格納されているソース・プログラムに基づき、プログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入された第１の中間表現プログラムを作成し、記憶手段に格納する手段、
記憶手段に格納されている前記第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出し、記憶手段に格納する手段、
記憶手段に格納されている前記ブロックについて、各命令をノードとし、依存関係にある命令のノード間にアークを設定するとともに、Ｓ－ＰＥＩの後にＨ－ＰＥＩ又は別のＳ－ＰＥＩを実行しなければならないという命令間の依存（以下、「例外依存」と言う。）の関係にある両命令のノード間のアーク（以下、「例外依存アーク」と言う。）を、例外依存以外の依存関係にある命令のノード間のアーク（以下、「非例外依存アーク」と言う。）に対して区別した第１の依存グラフ情報を作成し、記憶手段に格納する手段、
記憶手段に格納されている前記第１の依存グラフ情報に基づき、実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断する手段、
Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成し、記憶手段に格納する手段、
記憶手段に格納されている前記第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成し、記憶手段に格納する手段、及び
記憶手段に格納されている前記第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成し、記憶手段に格納する手段
を有していることを特徴とするコンパイル装置。
【請求項９】
前記実行時間に係る有利性基準とは、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準であることを特徴とする請求項８記載のコンパイル装置。
【請求項１０】
前記スケジューリングは、ブロック内のプログラムのクリティカル・パスを最小とするリスト・スケジューリングであることを特徴とする請求項８記載のコンパイル装置。
【請求項１１】
非例外依存にはデータ依存又は制御依存が含まれることを特徴とする請求項８記載のコンパイル装置。
【請求項１２】
前記投機実行がなされるように先端が別のノードに変更された例外依存アークの先端の当該別のノードに係る命令を番兵命令とし、投機実行対象のＨ－ＰＥＩが、その例外発生

を抑制された状態で前記番兵命令が実行されたときには、投機実行の対象となった命令列を、例外発生を抑制することなく再実行するように、記憶手段に格納されている前記第１の中間表現プログラム部分内の命令列を生成し、記憶手段に格納する手段、
を有していることを特徴とする請求項８記載のコンパイル装置。
【請求項１３】
前記投機実行がなされるように先端が別のノードに変更された例外依存アークの先端の当該別のノードに係る命令をチェック命令とし、投機実行の対象となった命令列のコピーを回復コードとして前記チェック命令の後に配置し、これにより前記チェック命令が、その実行時には、投機実行対象のＨ－ＰＥＩにおいて例外発生が抑制されたか否かを判定し、該判定が正であれば、前記回復コードが実行されるように、記憶手段に記憶されている前記第１の中間表現プログラム部分内の命令列を生成し、記憶手段に格納する手段、
を有していることを特徴とする請求項８記載のコンパイル装置。
【請求項１４】
記憶手段に格納されている前記第１の依存グラフ情報に対し、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへ向けて設定された例外依存アーク（以下、この例外依存アークを「Ｓ－ＰＥＩ間例外依存アーク」と言う。）の設定情報を含ませる手段、及び
Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、該Ｓ－ＰＥＩ間例外依存アークを除去し、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、該Ｓ－ＰＥＩ間例外依存アークをそのまま残すように、記憶手段に格納されている前記第２の依存グラフ情報を変更する手段、
を有していることを特徴とする請求項８記載のコンパイル装置。
【請求項１５】
記憶手段に格納されているソース・プログラムに基づき、プログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入された第１の中間表現プログラムを作成し、記憶手段に格納するステップ、
記憶手段に格納されている前記第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出し、記憶手段に格納するステップ、
記憶手段に格納されている前記ブロックについて、各命令をノードとし、依存関係にある命令のノード間にアークを設定するとともに、Ｓ－ＰＥＩの後にＨ－ＰＥＩ又は別のＳ－ＰＥＩを実行しなければならないという命令間の依存（以下、「例外依存」と言う。）の関係にある両命令のノード間のアーク（以下、「例外依存アーク」と言う。）を、例外依存以外の依存関係にある命令のノード間のアーク（以下、「非例外依存アーク」と言う。）に対して区別した第１の依存グラフ情報を作成し、記憶手段に格納するステップ、
記憶手段に格納されている前記第１の依存グラフ情報に基づき、実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断するステップ、
Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成し、記憶手段に格納するステップ、
記憶手段に格納されている前記第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成し、記憶手段に格納するステップ、及び
記憶手段に格納されている前記第２の中間表現プログラムに基づいてオブジェクト・プ

ログラムを作成し、記憶手段に格納するステップ、
をコンピュータに実行させるためのコンパイル用プログラム。
【請求項１６】
前記実行時間に係る有利性基準とは、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準であることを特徴とする請求項１５記載のコンパイル用プログラム。
【請求項１７】
前記スケジューリングは、ブロック内のプログラムのクリティカル・パスを最小とするリスト・スケジューリングであることを特徴とする請求項１５記載のコンパイル用プログラム。
【請求項１８】
非例外依存にはデータ依存又は制御依存が含まれることを特徴とする請求項１５記載のコンパイル用プログラム。
【請求項１９】
前記コンパイル用プログラムは、遅くとも前記スケジューリングより前に実行される第１の追加ステップをさらにコンピュータに実行させるものであり、
該第１の追加ステップでは、
前記投機実行がなされるように先端が別のノードに変更された例外依存アークの先端の当該別のノードに係る命令を番兵命令とし、
投機実行対象のＨ－ＰＥＩが、その例外発生を抑制された状態で前記番兵命令が実行されたときには、投機実行の対象となった命令列を、例外発生を抑制することなく再実行するように、記憶手段に格納されている前記第１の中間表現プログラム部分内の命令列を生成し、記憶手段に格納する、
ことを特徴とする請求項１５記載のコンパイル用プログラム。
【請求項２０】
前記コンパイル用プログラムは、遅くとも前記スケジューリングより前に実行される第２の追加ステップをさらにコンピュータに実行させるものであり、
該第２の追加ステップでは、
前記投機実行がなされるように先端が別のノードに変更された例外依存アークの先端の当該別のノードに係る命令をチェック命令とし、
投機実行の対象となった命令列のコピーを回復コードとして前記チェック命令の後に配置し、
これにより前記チェック命令が、その実行時には、投機実行対象のＨ－ＰＥＩにおいて例外発生が抑制されたか否かを判定し、該判定が正であれば、前記回復コードが実行されるように、記憶手段に記憶されている前記第１の中間表現プログラム部分内の命令列を生成し、記憶手段に格納する、
ことを特徴とする請求項１５記載のコンパイル用プログラム。
【請求項２１】
前記第１の依存グラフ情報には、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへ向けて設定された例外依存アーク（以下、この例外依存アークを「Ｓ－ＰＥＩ間例外依存アーク」と言う。）の設定情報が含まれ、
前記コンパイル用プログラムは、遅くとも前記スケジューリングより前に実行される第３及び第４の追加ステップをさらにコンピュータに実行させるものであり、
前記第３の追加ステップでは、
Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、該Ｓ－ＰＥＩ間例外依存アークを除去し、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、該Ｓ－ＰＥＩ間例外依存アークをそのまま残すように、記憶手段に記憶されている前記第２の依存グラフ情報を変更する、
ことを特徴とする請求項１５記載のコンパイル用プログラム。
【発明の詳細な説明】
【０００１】

【発明の属する技術分野】
本発明は、コンパイル装置、コンパイル方法、及びコンパイル用プログラムに係り、詳しくは依存を緩和して投機実行を適格化するコンパイル装置、コンパイル方法、及びコンパイル用プログラムに関する。
【０００２】
【従来の技術】
Ｊａｖａのような型安全(ｔｙｐｅ　ｓａｆｅ）な言語は、不正なメモリ・アクセスを起こさない、プログラマが意図しない異常なプログラム終了を起こさない、という性質を保証する。したがって、型安全な言語では、不正なメモリ・アクセスなどは行われず、結果として、プログラムがクラッシュするような実行はされない。このような型安全な言語がもつこれらの性質は、セキュリティの面からもプログラム言語において近年急速に重要性が増している。これらの性質を保証するためには、不正なアドレスを伴ったメモリアクセス命令、ＴＬＢ(Ｔｒａｎｓｌａｔｉｏｎ　Ｌｏｏｋａｓｉｄｅ　Ｂｕｆｆｅｒ）ミス、などハードウェア例外によってプログラムを異常に終了させる可能性がある命令（ｈａｒｄｗａｒｅ－ｉｎｉｔｉａｔｅｄ　ｐｏｔｅｎｔｉａｌｌｙ　ｅｘｃｅｐｔｉｏｎ　ｉｎｓｔｒｕｃｔｉｏｎｓ：Ｈ－ＰＥＩ）の前に、プログラムを停止しないこと保証する例外チェックの命令（ｓｏｆｔｗａｒｅ－ｉｎｉｔｉａｔｅｄ　ｐｏｔｅｎｔｉａｌｌｙ　ｅｘｃｅｐｔｉｏｎ　ｉｎｓｔｒｕｃｔｉｏｎｓ：Ｓ－ＰＥＩ）を挿入しなければならない。つまり、例外チェック命令の後にメモリアクセス命令を実行しなければならない、という命令間の依存が発生する。
【０００３】
このような依存関係や、一般にプログラムが持つデータ依存及び制御依存は、命令の実行順序を交換するような最適化を阻害する。このような依存を緩和して命令実行の順序を交換可能にする投機的実行が知られている。投機的実行は、プログラムの実行が決定する前にある命令を実行することである。このとき、不正な値を伴ってメモリアクセス命令が実行されることもあるので、例外発生を抑制した実行をシステムでサポートする必要がある。
【０００４】
型安全な言語では例外チェック命令が数多く存在するので、ＣやＦｏｒｔｒａｎのような型安全でない言語に比べて、投機実行を適用する機会が多い。しかし、型安全な言語の例外チェック命令に関する投機的実行を、プログラムの実行時間が短縮することを保証しながら効率よく適用する方法は知られていない。
【０００５】
また、プレサイス例外セマンティクス（Ｐｒｅｃｉｓｅ　ｅｘｃｅｐｔｉｏｎ　ｓｅｍａｎｔｉｃｓ）を要求する言語の場合は、Ｓ－ＰＥＩの発生順序がプログラム最適化前後でも変わらないことを保証しなければならないので、一般にＳ－ＰＥＩの実行順序を交換する最適化は単純ではない。
【０００６】
投機実行［先行文献１： M. D. Smith, M. S. Lam, and M. A. Horowitz. Boosting Beyond StaticScheduling in a Superscalar Processor. In Proceedings of the 17th AnnualInternational Symposium on Computer Architecture, pp. 344-354, 1990.］は、実際の実行が正しく行われるかどうかが決定する前に、早期に命令を実行することによって、プログラムの実行を高速化する方式として知られている。投機実行には、大きく２個の方法が知られている。一つは、制御投機（ｃｏｎｔｒｏｌ　ｓｐｅｃｕｌａｔｉｏｎ［先行文献２： S. A. Mahlke, W.Y. Chen, R. A. Bringmann, R. E. Hank, W. W. Hwu, B.R. Rau, and M. S. Schlansker. Sentinel scheduling: A model forcompiler-controlled speculative execution. ACM Transactions on ComputerSystems, 11(4), pp. 376-408, 1993.］）である。これは、分岐命令とその後ろに連続する命令との制御依存を緩和する。コンパイラが、分岐命令を越えて頻繁に実行される命令を移動することを可能にする。もう一つは、データ投機（ｄａｔａ　ｓｐｅｃｕｌａｔｉｏｎ［先行文献３：D. M. Gall

agher and W. Y. Chen and S. A. Mahlke and J. C. Gyllenhaaland W. W. Hwu. Dynamic memory disambiguation using the memory conflict buffer.In Proceedings of International Conference on Architectural Support forProgramming Languages and Operating Systems, pp. 183-193, 1994.］）である。これは、メモリストア命令とメモリロード命令との間のデータ依存を緩和する。コンパイラが、メモリストア命令とメモリロード命令が別のアドレスをアクセスすると仮定して、メモリストア命令を越えてロード命令と後続する命令を移動する。

【０００７】

これらの投機実行の研究は、型安全でない言語であるＣやＦｏｒｔｒａｎを対象としている。型安全な言語を対象とした投機実行の研究は、唯一［先行文献４：M. Arnold, M. S. Hsiao, U. Kremer, and B. Ryder. Exploring theinteraction between Java's implicitly thrown exceptions and instruct］がある。この研究では、以下の手順でＳ－ＰＥＩを越えた投機実行を行う。

１．プログラムに存在するＳ－ＰＥＩを、比較命令と分岐命令に分解する。

２．制御の入り口が一つで出口が複数となるスーパー・ブロック（ｓｕｐｅｒｂｌｏｃｋ）を形成する。

３．命令をノードとし、データ依存、制御依存の２個をアークとする、依存グラフを生成する。制御依存は、分岐命令から後続の全ての命令に持つことになる。

４．スーパー・ブロック（ｓｕｐｅｒ　ｂｌｏｃｋ）内で、ジェネラル・パーコレーション（ｇｅｎｅｒａｌ　ｐｅｒｃｏｌａｔｉｏｎ）を用いて分岐命令を越えた命令移動を行う。分岐命令を越えてＨ－ＰＥＩを移動した場合には、投機実行可能命令に変換する、分岐命令を越えて命令を移動した際には、依存グラフ上の制御依存は取り除かれる。

５．依存グラフ上で、命令レイテンシなどを考慮してリスト・スケジューリング（ｌｉｓｔ　ｓｃｈｅｄｕｌｉｎｇ）を行う。

【０００８】

先行文献４の方法では、１．において、Ｓ－ＰＥＩを、通常の条件分岐に用いられる比較命令と分岐命令に変換する。そのため、２．以下で分岐命令に対する従来のスーパー・ブロック・スケジューリング（ｓｕｐｅｒ　ｂｌｏｃｋ　ｓｃｈｅｄｕｌｉｎｇ）のフレームワークを使うことができる、という利点がある。実際の適用例を、図１に示す。図１において、１番上の四角形内の例示プログラムはＪａｖａ言語で記述されており、上から２番～４番の四角形内のプログラムはアセンブリ言語で記述されている。コンパイラは、与えられたソース・プログラムから、中間表現を作成する。その後、上記の手順に沿って、投機実行可能なコードを生成する。まず、Ｓ－ＰＥＩを比較命令と分岐命令に分解する。ここでは、ヌルチェック命令（ｎｕｌｌｃｈｅｃｋ）が、比較命令（ｃｍｐ）と分岐命令（ｊｍｐ）に分解されている。その後、スーパー・ブロックを形成し、依存グラフを形成する。さらに、スーパー・ブロック内で分岐命令を越えて命令を移動する。ここでは、Ｎ５，６，７がＮ４の分岐命令を越えて移動可能である。この際、Ｎ６はＨ－ＰＥＩなので、例外発生を抑制した投機ロード（ｓｐｅｃｕｌａｔｉｖｅ　ｌｏａｄ）命令に変換する。（例外発生を抑制するとは、例外の発生を所定のハードウェアに通知しないことを意味する。ただし、例外の発生を、ハードウェアに通知しない代わりに、ソフト上のメモが行われる。）。さらに、Ｎ４からＮ５，６，７への制御依存を除去する。Ｎ８を、投機実行の例外状態のチェックのための番兵(ｓｅｎｔｉｎｅｌ）命令とするので、Ｎ８は分岐命令を越えて移動することはできない。この例では、Ｎ６’で例外発生が抑制された状態でＮ８を実行した際に、Ｎ６’から例外発生を抑制することなく命令列を再実行する。最後に、命令レイテンシ（ｉｎｓｔｒｕｃｔｉｏｎ　ｌａｔｅｎｃｙ）を考慮してリスト・スケジューリング（ｌｉｓｔ　ｓｃｈｅｄｕｌｉｎｇ）を行う。なお、以降の図面関連説明では、命令レイテンシは、ロード命令（ｌｄ）及び投機的ロード命令（ｌｄ．ｓ）が３、その他の命令は１、であると仮定する。図２において、左側の依存グラフは図１のチェック命令分解前のコンパイラの中間表現に対応する依存グラフ、右側の依存グラフは図１のリスト・スケジューリング後のコンパイラの中間表現に対応する依存グラフである。

【０００９】
また、アーク除去による投機実行を採用する別の公知の方式としてＧＰＤＧ（Ｇｕａｒｄｅｄ　Ｐｒｏｇｒａｍ　Ｄｅｐｅｎｄｅｎｃｅ　Ｇｒａｐｈ）［先行文献５：古関、小松、深澤．命令レベル並列アーキテクチャのための大域的コードスケジューリング手法とその評価、JointSymposium on Parallel Processing 1994, pp. 1-8, 1994］がある。ＧＰＤＧは、命令をノードとし、制御依存、データ依存、及びリソース依存をアークとする、グラフから構成される。グラフによってプログラムの実行時間を表現できる、制御依存アークの除去による制御投機（ｃｏｎｔｒｏｌ　ｓｐｅｃｕｌａｔｉｏｎ）を扱うことができる。しかし、例外による制約を区別して表現していない。
【００１０】
例外が発生する順序を緩和する研究として、［先行文献６：Manish Gupta, Jong-Deok Choi, and Michael Hind. Optimizing Javaprograms in the presence of exceptions. In Proceedings of the 14th EuropeanConference on Object-Oriented Programming (ECOOP'00), pp. 422-446, 2000.］がある。この研究では、エンクロージング例外ハンドラー（ｅｎｃｌｏｓｉｎｇ　ｅｘｃｅｐｔｉｏｎ　ｈａｎｄｌｅｒ）の生存性（ｌｉｖｅｎｅｓｓ）を調べ、副作用がある命令と、Ｓ－ＰＥＩの実行順序を交換する最適化、Ｓ－ＰＥＩを発生する命令とチェックする命令とを分割しＳ－ＰＥＩの発生順序を正しく保つことでＳ－ＰＥＩのチェック順序を交換可能にする最適化、について述べている。
【００１１】
同様に例外を発生する順序を緩和する公知技術［先行文献７：稲垣、小松．例外を起こす可能性のある命令の投機実行機構、及び投機的例外処理機構の設置方法、Docketnumber JA9-2000-0285］では、Ｓ－ＰＥＩによる例外を発生する命令とチェックする命令とを分割し、クリティカル・パスを短縮している。Ｓ－ＰＥＩによる例外をチェックする命令とＨ－ＰＥＩの間に依然依存関係は残る。
【００１２】
【発明が解決しようとする課題】
先行文献４に記載された従来技術の問題点は次の通りである。
（ａ）Ｓ－ＰＥＩを汎用の比較命令と分岐命令とに分解するため、ブロック内の分岐の数が増え、プログラムの表現形式を複雑にし、コンパイル時間を増加させる可能性がある。図１の例では、元のプログラムには存在しなかった分岐命令が中間表現に新たに生成されてしまっている。これは、ジェネラル・パーコレーションの際に分岐命令を越えた命令移動を考慮する必要を生じる、などコンパイル時間を増加させる可能性を引き起こす。
（ｂ）Ｈ－ＰＥＩは元々Ｓ－ＰＥＩにのみ依存を持つ。しかしＳ－ＰＥＩが汎用の分岐命令に分解されることで、データ依存と制御依存とに関するグラフを作成する際に、後続の全ての命令と依存を持つことになり、不要な制御依存が生成される。図１の例では、元々のｎｕｌｌｃｈｅｃｋ命令は後続のｌｄ　ｒ５＝［ｒ４］のみをガードするためのものである。しかし、一般の分岐命令に変換されたことで、後続の全ての命令に対して制御依存が生成されたことによって、不要な依存を引き起こす（図２の左側の依存グラフ）。
（ｃ）正確な実行時間を見積もっていないので、実行時間を短縮しない投機的命令移動を起こすことがある。図１の例では、ジェネラル・パーコレーションによる命令移動は、命令レイテンシを考慮したリスト・スケジューリングの前に行われているので、移動可能な命令は全て移動されてしまう。したがって、この移動が実行時間を短縮できるのかどうかは、保証されていない。
（ｄ）Ｓ－ＰＥＩを汎用の比較命令と分岐命令とに分解すると、Ｓ－ＰＥＩ命令の実行順序交換は分岐命令の実行順序交換になるので、Ｓ－ＰＥＩ命令自体の実行順序交換の最適化が困難になる。
【００１３】
本発明は、上記（ａ）～（ｄ）の問題点を克服するコンパイル装置、コンパイル方法、及びコンパイル用プログラムを提供することである。
【００１４】

【課題を解決するための手段】
本発明のコンパイル方法は次のステップを有している。
・ソース・プログラムから第１の中間表現プログラムを作成するステップであって、該第１の中間表現プログラムではプログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入されているステップ
・第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出するステップ
・該ブロックについての第１の依存グラフ情報を作成するステップであって、該第１の依存グラフ情報では、各命令をノードとして、依存関係にあるノード間にアークが設定され、アークは、例外依存に係るアーク（以下、「例外依存アーク」と言う。）が、例外依存以外の依存に係るアーク（以下、「非例外依存アーク」と言う。）に対して区別されているステップ
・実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断するステップ
・Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成するステップ
・第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成するステップ
・第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成するステップ
【００１５】
実行時間に係る有利性基準とは、例えば、プログラム全体の実行に要する時間に係る基準や、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準である。もし投機実行のための命令移動を行うならば、プログラム全体の実行に要する時間が短縮化されたり、Ｈ－ＰＥＩの最早実行開始時間が早まるときには、投機実行のための命令移動は有利と判断される。スケジューリングは、リスト・スケジューリングに限定されない。スケジューリングは、典型的に、リスト・スケジューリングであるが、これに限定されず、当業者にとり自明の種々のスケジューリングを採用できる。リスト・スケジューリング以外のその他のスケジューリングとしては、例えば、整数線形計画法を使ったスケジューリングである。好ましくは、スケジューリングは、ブロック内のプログラムのクリティカル・パスを最小とするリスト・スケジューリングである。クリティカル・パスは例えば各命令の命令レイテンシに基づいて算出される。また、非例外依存には、例えば、データ依存及び／又は制御依存が含まれるが、さらに、リソース依存が含まれてもよい。
【００１６】
本発明のコンパイル方法は、特に、型安全な（ｔｙｐｅ　ｓａｆｅ）なプログラム言語のコンパイル処理において優れた性能を発揮する。なお、「型安全な言語」とは、本明細書において、決められた演算しか行わない、すなわち不正なメモリ・アクセスは行わないプログラム言語として定義される。「型安全な言語」には、例えば、Ｊａｖａ、Ｓｃｈｅｍｅ、ＣＬＵ、Ｍｏｄｕｌａ－２、Ｍｏｄｕｌａ－３、Ｏｂｅｒｏｎ、Ｐａｓｃａｌなどがある。
【００１７】
Ｈ－ＰＥＩ及びその後ろに連続する１個以上の命令を含む命令列が、Ｈ－ＰＥＩの最早実行開始時間などの実行時間に係る有利性判断に基づいて投機実行のための移動が行われることにより、有利な投機実行を保証しつつ、Ｈ－ＰＥＩについての投機実行を実施でき

る。
【００１８】
本発明のコンパイル方法には、さらに、次の特徴を付加することができる。
（ａ）遅くともスケジューリングより前に実行される第１の追加ステップを追加する。該第１の追加ステップでは、投機実行の実施のために先端を変更された例外依存アークの先端のノードに係る命令を番兵命令とし、投機実行対象のＨ－ＰＥＩが、その例外発生を抑制された状態で番兵命令が実行されたときには、投機実行の対象となった命令列を、例外発生を抑制することなく再実行するように、第１の中間表現プログラム部分内の命令列を生成する。
（ｂ）遅くともスケジューリングより前に実行される第２の追加ステップを追加する。該第２の追加ステップでは、投機実行の実施のために先端を変更された例外依存アークの先端のノードに係る命令をチェック命令とし、投機実行の対象となった命令列のコピーを回復コードとしてチェック命令の後に配置し、チェック命令では、投機実行対象のＨ－ＰＥＩにおいて例外発生が抑制されたか否かを判定し、該判定が正であれば、回復コードを実施するように、第１の中間表現プログラム部分内の命令列を生成する。
（ｃ）第１の依存グラフ情報には、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへ例外依存アーク（以下、この例外依存アークを「Ｓ－ＰＥＩ間例外依存アーク」と言う。）の設定情報を含ませる。遅くともスケジューリングより前に実行される第３及び第４の追加ステップを追加する。第３の追加ステップでは、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、該Ｓ－ＰＥＩ間例外依存アークを除去し、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、該Ｓ－ＰＥＩ間例外依存アークをそのまま残す。第４の追加ステップでは、第２の依存グラフ情報に、第３の追加ステップにおけるＳ－ＰＥＩ間例外依存アークの変更を含ませる。
【００１９】
「例外の種類」の「例外」とは、例外命令（例：図１２のＮ１，Ｎ４，Ｎ７のｎｕｌｌｃｈｅｃｋ）が発生する例外を意味する。Ｓ－ＰＥＩが関与する例外の種類には、入力値チェック、配列インデックス・チェックなどがあり、Ｓ－ＰＥＩ間例外依存アークの両端のノードに係るＳ－ＰＥＩが同一であれば、当然に例外の種類が同一と判断されるだけでなく、Ｓ－ＰＥＩ間例外依存アークの両端のノードに係るＳ－ＰＥＩが異なっていても、両Ｓ－ＰＥＩが、共に、例えば、入力チェックに係るものであるなどであれば、例外の種類が同一と判断される。
【００２０】
本発明のコンパイル装置は次のものを有している。
・ソース・プログラムから第１の中間表現プログラムを作成する手段であって、該第１の中間表現プログラムではプログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入されている手段
・第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出する手段
・該ブロックについての第１の依存グラフ情報を作成する手段であって、該第１の依存グラフ情報では、各命令をノードとして、依存関係にあるノード間にアークが設定され、アークは、例外依存に係るアーク（以下、「例外依存アーク」と言う。）が、例外依存以外の依存に係るアーク（以下、「非例外依存アーク」と言う。）に対して区別されている手段
・実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断する手段
・Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥ

Ｉのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成する手段
・第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成する手段
・第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成する手段
【００２１】
本発明のコンパイル用プログラムはコンピュータに次のステップを実行させる。
・ソース・プログラムから第１の中間表現プログラムを作成するステップであって、該第１の中間表現プログラムではプログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入されているステップ
・第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出するステップ
・該ブロックについての第１の依存グラフ情報を作成するステップであって、該第１の依存グラフ情報では、各命令をノードとして、依存関係にあるノード間にアークが設定され、アークは、例外依存に係るアーク（以下、「例外依存アーク」と言う。）が、例外依存以外の依存に係るアーク（以下、「非例外依存アーク」と言う。）に対して区別されているステップ
・実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断するステップ
・Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成するステップ
・第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成するステップ
・第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成するステップ
【００２２】
本発明のコンパイル用プログラムは、型安全な言語で記述されている必要はない。コンパイル用プログラム自体の言語は、Ｊａｖａのような型安全な言語であっても、また、Ｃ言語のような型安全でない言語であってもよいとする。
【００２３】
【発明の実施の形態】
以下、発明の実施の形態について図面を参照して説明する。
図３はＳ－ＰＥＩを越えた投機実行を行う処理手順のフローチャートである。Ｓ１０では、制御の入り口が一つで出口が一つ以上となるブロック（ｂｌｏｃｋ）を形成する。なお、ブロックには、例えば、基本ブロック（ｂａｓｉｃ　ｂｌｏｃｋ）、スーパー・ブロック（ｓｕｐｅｒ　ｂｌｏｃｋ）、及びハイパー・ブロック（ｈｙｐｅｒｂｌｏｃｋ）などがある。これらブロックでは、入り口は１個である。出口は、基本ブロックでは１個、スーパー・ブロック及びハイパー・ブロックでは１個以上となっている。ハイパー・ブロックは内部に合流部をもつ。合流部とは、例えば、ｉｆ～ｔｈｅｎ～ｅｌｓｅの次の命令のように、制御がｔｈｅｎ部及びｅｌｓｅ部から共通に流れる命令である。また、２以上の出口とは、ブロック内の中間部にｂｒａｎｃｈなどの命令が存在することにより形成される。概念上、スーパー・ブロックは基本ブロックを包摂する。Ｓ１０におけるブロック

は、基本ブロック、スーパー・ブロック、及びハイパー・ブロックのどれでもよい。Ｓ１１では、命令をノードとし、データ依存、制御依存、及び例外依存の３つをアークとする依存グラフを生成する。例外依存は、Ｓ－ＰＥＩとＨ－ＰＥＩ間に生成する。プレサイス例外セマンティクス（Ｐｒｅｃｉｓｅ　ｅｘｃｅｐｔｉｏｎ　ｓｅｍａｎｔｉｃｓ）を要求する言語の場合は、例外の発生順序を保証するために、Ｓ－ＰＥＩとＳ－ＰＥＩの間にも例外依存を生成する。Ｓ１２では、Ｓ－ＰＥＩからの例外依存をもつＨ－ＰＥＩノードについて、データ依存、制御依存、及び例外依存のそれぞれによって決定される最早実行開始時間を比較する。例外依存によって決定される最早実行開始時間が他より遅い場合にのみ、投機的命令移動を適用して、例外依存のアークを番兵命令へ張り替える。本発明は、投機実行の例外状態の復帰方式として、回復コード（ｒｅｃｏｖｅｒｙ　ｃｏｄｅ）無しの場合にも、回復コード有りの場合にも適用可能である。回復コード有りの場合には、ここで回復コードも生成する。Ｓ１３では、Ｓ－ＰＥＩからの例外依存を持つＳ－ＰＥＩノードについて、アークの両端のノードで発生する例外の種類が同じならばアークを除去する。Ｓ１４では、依存グラフ上で、命令レイテンシなどを考慮してリスト・スケジューリング（ｌｉｓｔ　ｓｃｈｅｄｕｌｉｎｇ）を行う。

【００２４】

図４は例外依存アークを生成する手順をプログラム記述形式で示している。図４において、ｓｒｃはソース（ｓｏｕｒｃｅ）、ｄｓｔはデスティネーション（ｄｅｓｔｉｎａｔｉｏｎ）を意味する。ｆｏｒｅａｃｈは、（　）内の集合に含まれる各要素について１個ずつ｛　｝内の処理を実施することを意味する。ブロック内の各Ｈ－ＰＥＩ命令について、そのソース・オペランドをデスティネーション・オペランドとしてもつ先行の命令が調べ上げられ、そのデスティネーション・オペランドから到達するソース・オペランドを持つ命令がＳ－ＰＥＩ命令であるならば、そのＳ－ＰＥＩ命令とＨ－ＰＥＩ命令との間に例外依存アークを張る。図４の下３行の命令列の意義は次の通りである。Ｈ－ＰＥＩへ例外依存を張るＳ－ＰＥＩは１個とは限らない。そのため、データ依存をたどりながら、Ｈ－ＰＥＩで使用される値を生成する命令（ｎｅｗ／ｎｅｗａｒｒａｙ／ｇｅｔｆｉｅｌｄ／ｇｅｔｓｔａｔｉｃなどのオブジェクトを生成する命令）に到達するまで存在するＳ－ＰＥＩを調べる必要がある。対象のＨ－ＰＥＩで使用されるオペランドを生成する命令に到達するまでデータ依存をたどるのが、この３行の命令列である。

【００２５】

プレサイス例外セマンティクスを要求する言語の場合は、例外発生の順序を決定するためにＳ－ＰＥＩとＳ－ＰＥＩの間にも例外依存アークを生成する。これは以下の手順で生成可能である。図５は例外発生の順序を決定するためにＳ－ＰＥＩとＳ－ＰＥＩの間にも例外依存アークを生成する手順をプログラム記述形式で示している。ブロック内の各命令について、逆戻り順（前の命令から後ろの命令の方へ。ｒｅｖｅｒｓｅ　ｐｏｓｔ　ｏｒｄｅｒ）で命令が調べ上げられ、Ｓ－ＰＥＩ命令に行き当たると、それをｉｓとし、ｉｓより後ろの各命令について逆戻り順で、Ｓ－ＰＥＩ命令を探し、最初に探し当てたＳ－ＰＥＩ命令をｉｅとして、ｉｓとｉｅとの間に例外依存アークを張る。

【００２６】

図６は本発明を適用したコンパイラによる処理過程の説明図、図７は図６の処理過程に対応して作成される依存グラフである。図６の一番上の四角内のＪａｖａのプログラムは、コンパイラが処理するソース・プログラムである。Ｊａｖａのソース・プログラムは、型安全な言語で記述されたソース・プログラムとして本発明の説明のために選択されている。コンパイラは、ソース・プログラムから中間表現のプログラム（このプログラムより図３のＳ１０のブロック抽出を行ったものが例えば図６の上から２番目の四角内のプログラム部分となる。）を作成する。この中間表現プログラムでは、Ｈ－ＰＥＩ（Ｎ５のｌｄ）に対して、プログラムが異常終了しないことを保証するＳ－ＰＥＩ（Ｎ３のｎｕｌｌｃｈｅｃｋ）がこのＨ－ＰＥＩより前に挿入される。中間表現プログラム部分に対して図４及び図５の処理が実行されることにより、図７の左側の依存グラフが作成される。この依存グラフにおいて、Ｓ－ＰＥＩからの例外依存を持つＨ－ＰＥＩのノードについて、デー

タ依存、制御依存、例外依存から決定される最早実行開始時間を求める。これは、依存グラフ上で、グラフの始点から当該ノードまでアークが持つ重みによって決まる最早実行開始時間を蓄積していくことより求められる。アークの重みは、データ依存及び制御依存では命令のレイテンシに等しい。例外依存のアークの重みは０である。例では、Ｎ５について依存から制約される時間を求めると、データ依存によって決定される最早実行開始時間は５(＝１＋３＋１）であり、例外依存によって決定される最早実行開始時間は４＝(１＋３＋０）、であることが分かる。例外依存によって決定される最早実行開始時間の方が速いので、例外依存による制約を除去しても、この命令の実行開始時間は早くならないことが分かる。したがって、この場合、投機的命令移動を適用しない。不必要な投機的命令移動は、不要なデータ・キャッシュのｍｉｓｓを引き起こす、ＴＬＢ(Ｔｒａｎｓｌａｔｉｏｎ　Ｌｏｏｋａｓｉｄｅ　Ｂｕｆｆｅｒ）ミスによる命令再実行のオーバヘッド、など性能を落とす原因を引き起こすので、避けなければならない。次に、図７の左側の依存グラフにおいて、交換可能なＳ－ＰＥＩノード、すなわち、Ｓ－ＰＥＩからの例外依存を持つＳ－ＰＥＩのノードを探す。この２個のノードが発生する例外の種類が同じで、例外を捕捉した時点で観測できる状態を変更する命令がこの２個のノード間に無いならば、例外依存アークを除去できる。例外を捕捉した時点で観測できる状態を変更する命令とは、例外ハンドラの中で参照される変数への書き込み、配列・インスタンス変数・クラス変数への書き込みなどである。配列・インスタンス変数・クラス変数への書き込みについては、前述した先行文献６のように、ｅｎｃｌｏｓｉｎｇ　ｅｘｃｅｐｔｉｏｎ　ｈａｎｄｌｅｒの生存性（ｌｉｖｅｎｅｓｓ）を調べてより正確に判断することもできる。この例ではＳ－ＰＥＩが１個しかないので、交換はできない。こうして、図７の右側の依存グラフが作成される。最後に、図７の右側の依存グラフに基づいて命令レイテンシを考慮したリスト・スケジューリングを行って、図６の上から３番目の中間表現を得る。この３番目の中間表現は、２番目の中間表現に対して、Ｎ３とＮ４との順番が入れ替わっているが、Ｎ３の命令レイテンシは０であるので、特に入れ替えなくてもよい。リスト・スケジューリングは、例えば、プログラム全体の実行に要する時間を短縮させること及び／又は各命令の最早実行開始時間が早まることを目的に、実施される。

【００２７】

別のソース・プログラムに本発明に準拠するコンパイラを使用したときのコンパイルの実施例を説明する。関連図は図８及び図９である。図８はコンパイラによる処理過程の説明図、図９は図８の処理過程に対応して作成される依存グラフである。この実施例では、Ｊａｖａプログラムにおいて、投機実行の例外状態の復帰方式として回復コード（ｒｅｃｏｖｅｒｙ　ｃｏｄｅ）無しの場合を示す。前述しているが、回復コード無しの場合とは、自前で回復コードを作成せずに、作成済みとなっている汎用のコードを使用する場合を意味し、自前での回復コードの作成は省略する。まず、ブロック（この例では基本ブロック）を形成する。図８の第１段階中間表現は、このブロックにおける命令リストを示している。次に、第１段階中間表現に基づいて、データ依存、制御依存、及び例外依存からなる依存グラフ（図９の左側の依存グラフ）を形成する。この依存グラフを生成する際に、例外依存は、Ｓ－ＰＥＩとＳ－ＰＥＩ、Ｓ－ＰＥＩとＨ－ＰＥＩの間に生成する。この例では、Ｓ－ＰＥＩとＳ－ＰＥＩの例外依存をＮ１→Ｎ３（Ｎ１のｎｅｗａｒｒａｙ命令も例外を生成する可能性があるＳ－ＰＥＩ命令である）に、Ｓ－ＰＥＩとＨ－ＰＥＩの依存をＮ３→Ｎ６に生成する。次に、Ｓ－ＰＥＩノードからの例外依存を持つＨ－ＰＥＩノードについて、データ依存、制御依存、及び例外依存から決定される最早実行開始時間を求める。この場合は、Ｎ６について依存から制約される時間を求めると、データ依存によって決定される最早実行開始時間は３、例外依存によって決定される最早実行開始時間は４、であることが分かる。例外依存によって決定される最早実行開始時間の方が遅いので、例外依存による制約を除去すると、この命令の実行開始時間は早くなることが分かる。したがって、この場合投機的命令移動を適用する。ここでは、Ｎ６とＮ７を投機的命令移動の対象として、Ｎ８を番兵（ｓｅｎｔｉｎｅｌ）命令とする。Ｈ－ＰＥＩであるＮ６を投機的命令移動したので、Ｎ６において、ｌｏａｄ命令（ｌｄ）をｓｐｅｃｕｌａｔｉｖｅ

ｌｏａｄ命令（ｌｄ．ｓ）に変換する。さらに、Ｎ３からＮ６への例外依存を除去して、Ｎ３からｓｅｎｔｉｎｅｌのＮ８に張り直す（図９の右側の依存グラフを参照。）。この例では、Ｎ１→Ｎ３にＳ－ＰＥＩ間のアークが存在するが、Ｎ１が発生する例外とＮ３が発生する例外は異なるので、アークを除去することはできない。こうして、図９の右側依存グラフに対応する、図８の第２段階中間表現が作成される。最後に、第２段階中間表現を、命令レイテンシを考慮してリスト・スケジューリングすることにより、図８の第３段階中間表現が完成する。この例では、Ｎ６で例外発生が抑制された状態でＮ８を実行したときは、Ｎ６から例外発生を抑制することなく命令列を再実行する。このように、回復コード無しの場合は、後述の図１０の第２段階中間表現のＮ１０～Ｎ１２のようなリカバリ（回復）専用のコードを用意しない代わりに、所定の命令（例：図８のＮ８のｒｅｔ）にリカバリ時の機能を適宜、付与して、リカバリ対策を施している。クリティカル・パス長（ブロック内の最終の命令に到達する最長経路長であって、経路長は命令レイテンシのに基づいて算出する。）は、図８の左側の依存グラフでは、８＝(１＋３＋０＋３＋１）であったが、右側の依存グラフでは、７＝(１＋１＋１＋３＋１）に短縮される。

【００２８】

次に、投機実行の例外状態の復帰方式として回復コード有りの場合のＪａｖａプログラムのコンパイル例について説明する。関連図は図１０及び図１１である。まず、ブロック（この例では基本ブロック）を形成して、図１０の第１段階中間表現を得る。次に、この第１段階中間表現に基づいてデータ依存、制御依存、及び例外依存からなる依存グラフ（図１１の左側の依存グラフ）を形成する。この依存グラフを生成する際に、例外依存は、Ｓ－ＰＥＩとＳ－ＰＥＩ、Ｓ－ＰＥＩとＨ－ＰＥＩの間に生成する。この例では、Ｓ－ＰＥＩとＳ－ＰＥＩの例外依存をＮ１→Ｎ３に、Ｓ－ＰＥＩとＨ－ＰＥＩの依存をＮ３→Ｎ６に生成する。次に、Ｓ－ＰＥＩからの例外依存を持つＨ－ＰＥＩノードについて、データ依存、制御依存、及び例外依存から決定される最早実行開始時間を求める。この場合は、Ｎ６について依存から制約される時間を求めると、データ依存によって決定される最早実行開始時間は３、例外依存によって決定される最早実行開始時間は４、であることが分かる。例外依存によって決定される最早実行開始時間の方が遅いので、例外依存による制約を除去すると、この命令の実行開始時間は早くなることが分かる。したがって、この場合、投機的命令移動を適用する。ここでは、Ｎ６とＮ７を投機的命令移動の対象とする。Ｈ－ＰＥＩであるＮ６を投機的命令移動したので、Ｎ６において、ｌｏａｄ命令（ｌｄ）をｓｐｅｃｕｌａｔｉｖｅ　ｌｏａｄ命令（ｌｄ．ｓ）に変換する。ｓｐｅｃｕｌａｔｉｖｅ　ｌｏａｄ命令で例外が抑制されたかどうか調べ、ｒｅｃｏｖｅｒｙ　ｃｏｄｅを実行するかどうかを判断するｃｈｋ命令（ｃｈｋ．ｓ）をＮ９として追加する。さらに、投機的命令移動の対象としたＮ６とＮ７をコピーして、回復コードのためにＮ１０，Ｎ１１として追加する。この際ｓｐｅｃｕｌａｔｉｖｅ　ｌｏａｄ命令は通常のｌｏａｄ命令に変換される。最後に、回復コードから戻るためのｂｒａｎｃｈ命令（ｂ）をＮ１２として追加する。アークに関しては、Ｎ３からＮ６への例外依存を除去して、Ｎ３からＮ９に張り直す。Ｎ９では、例外が抑制されたかどうかを調べるためにｔ６を読むので、Ｎ７からＮ９へデータ依存のアークを張る。回復コード内のＮ１０はｔ４を読むので、Ｎ５からＮ１０へデータ依存のアークを張る。Ｎ１１で定義されたｔ６はＮ８で読まれるので、Ｎ１１からＮ８へデータ依存のアークを張る。さらに、ｃｈｋ命令が正しい位置にｓｃｈｅｄｕｌｅされるように、Ｎ９からＮ８へ制御依存のアークを張る。ここでは、ｃｈｋ命令のレイテンシを０とすると、アークの重みは０となる。この例では、Ｎ１→Ｎ３にＳ－ＰＥＩ間のアークが存在するが、Ｎ１が発生する例外とＮ３が発生する例外は異なるので、アークを除去することはできない。こうして、図１１の右側の依存グラフに対応する図１０の第２段階中間表現が完成する．最後に、命令レイテンシを考慮してリスト・スケジューリングを行い、図１０の第３段階中間表現を得る．ただし、Ｎ１０，Ｎ１１，Ｎ１２から構成される回復コードはまれにしか実行されないと考えられるので、Ｎ１～９からなる頻繁に実行されるコードと分けてリスト・スケジューリングを適用する。さらに頻繁に実行されるコードのリスト・スケジューリングの際には、回復コードから流入するアークを無

視する。本発明適用前に８＝(１＋３＋０＋３＋１）であったクリティカル・パス長は、７＝(１＋１＋１＋３＋１）に短縮されている。

【００２９】

次に、ＪａｖａプログラムのコンパイルにおいてＳ－ＰＥＩの実行順序を交換できる例を示す。関連図は図１２及び図１３である．コンパイラは、図１２のＪａｖａプログラムのコンパイルを行う。まずブロック（この例では基本ブロック）を形成して、図１２の第１段階中間表現を得る。次に、データ依存、制御依存、及び例外依存からなる依存グラフ（図１３の左側依存グラフ）を形成する。依存グラフを生成する際に、例外依存は、Ｓ－ＰＥＩとＳ－ＰＥＩ、Ｓ－ＰＥＩとＨ－ＰＥＩの間に生成する。この例では、Ｓ－ＰＥＩとＳ－ＰＥＩの例外依存をＮ１→Ｎ４→Ｎ７に、Ｓ－ＰＥＩとＨ－ＰＥＩの依存をＮ１→Ｎ３，Ｎ４→Ｎ６，Ｎ７→Ｎ９に生成する。次に、Ｓ－ＰＥＩからの例外依存を持つＨ－ＰＥＩノードについて、データ依存、制御依存、及び例外依存から決定される最早実行開始時間を求める。この場合は、Ｎ３，Ｎ６，Ｎ９ともに、データ依存によって決定される最早実行開始時間は例外依存によって決定される最早実行開始時間より遅いことが分かる。したがって、この場合投機的命令移動を適用しない。この例では、Ｎ１→Ｎ４→Ｎ７にＳ－ＰＥＩ間のアークが存在し、Ｎ１が発生する例外とＮ４が発生する例外とＮ７が発生する例外が同じＮｕｌｌｐｏｉｎｔｅｒＥｘｃｅｐｔｉｏｎで、この３つのノード間に例外を捕捉した時点で観測できる状態を変更する命令は存在しないので、例外依存によるアークを除去する。このような処理を経て変更された依存グラフが図１３の右側の依存グラフであり、この依存グラフに対応するプログラムが図１２の第２段階中間表現である。最後に、第２段階中間表現を基に、命令レイテンシを考慮してリスト・スケジューリングを行い、第３段階中間表現を得る。クリティカル・パス長は、第１段階中間表現では、１０＝(０＋１＋３＋０＋０＋３＋１＋１＋１）｛引数ａ→Ｎ２→Ｎ３→Ｎ４→Ｎ７→Ｎ９→Ｎ１０→Ｎ１１→Ｎ１２→Ｎ１３｝であるのに対し、リスト・スケジューリング後の第３段階中間表現では、９＝(０＋１＋３＋１＋３＋１）｛引数ａ→Ｎ２→Ｎ３→Ｎ５→Ｎ６→Ｎ１２→Ｎ１３｝に短縮される。

【００３０】

所定のＪａｖａソース・プログラムに対し投機実行の例外状態の復帰方式として回復コード有りのコンパイルを本発明準拠のコンパイラと従来方式のコンパイラとで行った場合、ＳＰＥＣｊｖｍ９８ベンチマークのｃｏｍｐｒｅｓｓ，ｍｐｅｇａｕｄｉｏ，ｄｂで約８％の性能向上が得られた。ＳＰＥＣｊｖｍ９８ベンチマークを実行した場合、１０７０個のメソッドが実行された。このときのＤＡＧ内のノードとアークの数を図１４に示す。従来方式では、Ｓ－ＰＥＩ命令でＰＤＧを分割し、後続ＰＤＧと例外ハンドラへアークを張った。本発明準拠のコンパイラの方が、ＰＤＧノード、合計アーク数の両方において、従来方式より総数が少ないことが分かる。したがって、コンパイル時に要求されるメモリ量が少ない。また、１個のＰＤＧノードが含む命令数が約４倍多い。このことより、従来技術において本発明と同様の効果を得るためには、トレース・スケジューリング（ｔｒａｃｅ　ｓｃｈｅｄｕｌｉｎｇ）やパーコレーション・スケジューリング（ｐｅｒｃｏｌａｔｉｏｎ　ｓｃｈｅｄｕｌｉｎｇ）のようなより時間のかかるグローバル・コード・モーション（ｇｌｏｂａｌ　ｃｏｄｅ　ｍｏｔｉｏｎ）を適用しなければならない。このことからも本発明はコンパイル時の効率を上げることが可能になる。

【００３１】

図１５はコンパイル装置２０の機能ブロック図である。第１の中間表現プログラム作成手段２４は、Ｊａｖａなどの型安全な言語で記述されたソース・プログラムより第１の中間表現プログラムを作成する。第１の中間表現プログラムでは、プログラムを異常に終了させる可能性がある命令としてのＳ－ＰＥＩ（例：図８の第１段階中間表現のＮ６）の前に、プログラムを停止しないことを保証する例外チェックの命令としてのＳ－ＰＥＩ（例：図８の第１段階中間表現のＮ３）が挿入されている。ブロック抽出手段２５は、第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロック（例：図８の第１段階中間表現）を抽出する。第１の依存グラフ情報作成手段２６は、ブロック抽出手

段２５が抽出したブロックの中間表現プログラム部分についての第１の依存グラフ情報（例：図９の左側の依存グラフに係る情報）を作成する。第１の依存グラフ情報では、各命令をノードとして、依存関係にあるノード間にアークが設定され、アークは、例外依存に係るアークとしての例外依存アーク（例：図９において軸部が点線の矢印）が、データ依存や制御依存などの例外依存以外の依存に係るアークとしての非例外依存アークに対して区別されている。Ｓ－ＰＥＩ間例外依存アーク設定手段２７は、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへの例外依存アークとしてのＳ－ＰＥＩ間例外依存アーク（例：図９の左側の依存グラフにおけるＮ１からＮ３への矢印）の設定情報を、第１の依存グラフ情報作成手段２６における第１の依存グラフ情報に付加する。有利性判断手段３０は、第１の依存グラフ情報作成手段２６が作成した第１の依存グラフ情報において、実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合とこの例外依存アークを経由しないで実行された場合とでどちらが有利かを判断する。実行時間に係る有利性基準とは、例えば、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準であり、Ｈ－ＰＥＩの最早実行開始時間が、例外依存アークを経由して実行されるときよりも、例外依存アークを経由しないで実行されるときの方が、早まるならば、投機実行のための命令移動を実施すべきと判断する。

【００３２】

第２の依存グラフ情報作成手段３１は、Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より、実行時間に係る有利性基準に照らして有利との判断があったときは、このＨ－ＰＥＩのノード及びその後ろに連続する幾つかのノードに対応の命令の列が投機実行されるように例外依存アークの先端を別のノードへ変更し、この変更に対応する第２の依存グラフ情報（例：図９の右側の依存グラフに係る情報）を作成する。第１の復帰設定手段３４及び第２の復帰設定手段３５は、中間表現プログラム部分の各ブロックに対してはいずれか一方のみが関与する。すなわち、コンパイル装置２０は、第１の復帰設定手段３４及び第２の復帰設定手段３５の一方のみを装備してもよいし、両方を装備して、或るブロックに対しては第１の復帰設定手段３４を関与させ、また、別のブロックに対しては第２の復帰設定手段３５を関与させることもできる。第１の復帰設定手段３４は中間表現プログラム部分のブロックに対して回復コード無しの処理を行うものであり、第２の復帰設定手段３５は中間表現プログラム部分のブロックに対して回復コード有りの処理を行うものである。第１の復帰設定手段３４は、有利性判断手段３０の判断の結果、投機実行の実施のために先端を変更された例外依存アークの先端のノード（例：図９の右側依存グラフのＮ８）に係る命令を番兵命令とし、投機実行対象のＨ－ＰＥＩ（例：図９の右側依存グラフのＮ６）が、その例外発生を抑制された状態で番兵命令が実行されたときには、投機実行の対象となった命令列（例：図９の右側依存グラフのＮ６，Ｎ７）を、例外発生を抑制することなく、再実行するように、第１の中間表現プログラム部分内（例：図８の第２段階中間表現）の命令列を編集する。なお、図８の第２段階中間表現のＮ８は、図８の第１段階中間表現のＮ８と同様に、ｒｅｔ（ｒｅｔｕｒｎ）と同一表現となっているが、番兵命令としての機能を付与されたｒｅｔとなっている。また、第２の復帰設定手段３５は、有利性判断手段３０の判断の結果、投機実行の実施のために先端を変更された例外依存アークについて、その変更後例外依存アークの先端のノード（図１１の右側依存グラフのＮ９）に係る命令をチェック命令とし、投機実行の対象となった命令列のコピーを回復コードとしてチェック命令の後に配置する（例：図１１の右側依存グラフのＮ１０，Ｎ１１）。ただし、このコピーでは、投機命令へ変更した命令（例：図１０のＮ６のｌｄ．ｓ）は、元の通常の命令（例：ｌｄ）へ戻す。ブロック抽出手段２５は第１の中間表現プログラム部分内（例：図１０の第２段階中間表現）の命令列について編集し、この編集では、チェック命令（例：図１１の右側依存グラフのＮ９）において、投機実行対象のＳ－ＰＥＩ（例：Ｎ６）において例外発生が抑制されたか否かを判定し、この判定が正であれば、回復コード（例：Ｎ１０，Ｎ１１）を実施する。Ｓ－ＰＥＩ間例外依存アーク変更手段３６は、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、このＳ－Ｐ

ＥＩ間例外依存アークを除去し（例：図１３の左右依存グラフにおけるＮ４→Ｎ７に注目）、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、このＳ－ＰＥＩ間例外依存アークをそのまま残すように（例：図１３の左右依存グラフにおけるＮ２→Ｎ４に注目）、第２の依存グラフ情報を変更する（例：図１３の右側依存グラフ）。第２の中間表現プログラム作成手段３７は、第１の中間表現プログラム部分をリスト・スケジューリングした中間表現プログラム部分としての第２の中間表現プログラム部分（例：図８、図１０、及び図１３のリスト・スケジューリング後のコンパイラの中間表現）から成る全体の第２の中間表現プログラムを作成する。リスト・スケジューリングは、例えば、ブロック内のプログラムのクリティカル・パスが最小となる観点で行われる。オブジェクト・プログラム作成手段３８は、第２の中間表現プログラム作成手段３７が作成した第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成する。

【００３３】

まとめとして本発明のコンパイル装置及びコンパイル用プログラムの構成に関して以下の事項を開示する。

（１）ソース・プログラムから第１の中間表現プログラムを作成する手段であって、該第１の中間表現プログラムではプログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入されている手段、

第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出する手段、

該ブロックについての第１の依存グラフ情報を作成する手段であって、該第１の依存グラフ情報では、各命令をノードとして、依存関係にあるノード間にアークが設定され、アークは、例外依存に係るアーク（以下、「例外依存アーク」と言う。）が、例外依存以外の依存に係るアーク（以下、「非例外依存アーク」と言う。）に対して区別されている手段、

実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断する手段、

Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成する手段、及び

第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成する手段、及び

第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成する手段

を有しているコンパイル装置。

（２）実行時間に係る有利性基準とは、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準である（１）記載のコンパイル装置。

（３）スケジューリングは、ブロック内のプログラムのクリティカル・パスを最小とするリスト・スケジューリングである（１）又は（２）記載のコンパイル装置。

（４）非例外依存にはデータ依存及び／又は制御依存が含まれる（１）～（３）のいずれかに記載のコンパイル装置。

（５）投機実行の実施のために先端を変更された例外依存アークの先端のノードに係る命令を番兵命令とし、投機実行対象のＨ－ＰＥＩが、その例外発生を抑制された状態で番兵命令が実行されたときには、投機実行の対象となった命令列を、例外発生を抑制することなく再実行するように、第１の中間表現プログラム部分内の命令列を生成する手段、

を有している（１）～（４）のいずれかに記載のコンパイル装置。

【００３４】
（６）投機実行の実施のために先端を変更された例外依存アークの先端のノードに係る命令をチェック命令とし、投機実行の対象となった命令列のコピーを回復コードとしてチェック命令の後に配置し、チェック命令では、投機実行対象のＨ－ＰＥＩにおいて例外発生が抑制されたか否かを判定し、該判定が正であれば、回復コードを実施するように、第１の中間表現プログラム部分内の命令列を生成する手段、
を有している（１）～（４）のいずれかに記載のコンパイル装置。
（７）第１の依存グラフ情報には、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへ例外依存アーク（以下、この例外依存アークを「Ｓ－ＰＥＩ間例外依存アーク」と言う。）の設定情報を含ませる手段、及び
　Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、該Ｓ－ＰＥＩ間例外依存アークを除去し、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、該Ｓ－ＰＥＩ間例外依存アークをそのまま残すように、第２の依存グラフ情報を変更する手段、
を有している（１）～（７）のいずれかに記載のコンパイル装置。
（８）ソース・プログラムから第１の中間表現プログラムを作成するステップであって、該第１の中間表現プログラムではプログラムを異常に終了させる可能性がある命令（以下、「Ｈ－ＰＥＩ」と言う。）の前に、プログラムを停止しないことを保証する例外チェックの命令（以下、「Ｓ－ＰＥＩ」と言う。）が挿入されているステップ、
　第１の中間表現プログラムにおいて記述順の連続する複数個の命令を含むブロックを抽出するステップ、
　該ブロックについての第１の依存グラフ情報を作成するステップであって、該第１の依存グラフ情報では、各命令をノードとして、依存関係にあるノード間にアークが設定され、アークは、例外依存に係るアーク（以下、「例外依存アーク」と言う。）が、例外依存以外の依存に係るアーク（以下、「非例外依存アーク」と言う。）に対して区別されているステップ、
　実行時間に係る有利性基準に照らしてＨ－ＰＥＩのノードについてそれが例外依存アークを経由して実行された場合と該例外依存アークを経由しないで実行された場合とでどちらが有利かを判断するステップ、
　Ｈ－ＰＥＩのノードについてそれが例外依存アークを経由しないで実行された場合の方が例外依存アークを経由して実行された場合より有利との判断があったときは該Ｈ－ＰＥＩのノードとその後ろに連続する１個以上のノードとを含む命令列が投機実行されるように例外依存アークの先端を別のノードへ変更した第２の依存グラフ情報を作成するステップ、及び
　第２の依存グラフ情報に対応する中間表現プログラム部分（以下、該中間表現プログラム部分を「第１の中間表現プログラム部分」と言う。）をスケジューリングした中間表現プログラム部分（以下、該中間表現プログラム部分を「第２の中間表現プログラム部分」と言う。）を備える第２の中間表現プログラムを作成するステップ、及び
　第２の中間表現プログラムに基づいてオブジェクト・プログラムを作成するステップ、
をコンピュータに実行させるためのコンパイル用プログラム。
（９）実行時間に係る有利性基準とは、Ｈ－ＰＥＩの最早実行開始時間に係る有利性基準である、ステップをコンピュータに実行させるための（８）記載のコンパイル用プログラム。
（１０）スケジューリングは、ブロック内のプログラムのクリティカル・パスを最小とするリスト・スケジューリングである、ステップをコンピュータに実行させるための（８）又は（９）記載のコンパイル用プログラム。
【００３５】
（１１）非例外依存にはデータ依存及び／又は制御依存が含まれる、ステップをコンピュータに実行させるための（８）～（１０）のいずれかに記載のコンパイル用プログラム。
（１２）遅くともスケジューリングより前に実行される第１の追加ステップを追加し、

該第１の追加ステップでは、
投機実行の実施のために先端を変更された例外依存アークの先端のノードに係る命令を番兵命令とし、
投機実行対象のＨ－ＰＥＩが、その例外発生を抑制された状態で番兵命令が実行されたときには、投機実行の対象となった命令列を、例外発生を抑制することなく再実行するように、第１の中間表現プログラム部分内の命令列を生成する、
ステップをコンピュータに実行させるための（８）～（１１）のいずれかに記載のコンパイル用プログラム。
（１３）遅くともスケジューリングより前に実行される第２の追加ステップを追加し、
該第２の追加ステップでは、
投機実行の実施のために先端を変更された例外依存アークの先端のノードに係る命令をチェック命令とし、
投機実行の対象となった命令列のコピーを回復コードとしてチェック命令の後に配置し、
チェック命令では、投機実行対象のＨ－ＰＥＩにおいて例外発生が抑制されたか否かを判定し、
該判定が正であれば、回復コードを実施するように、第１の中間表現プログラム部分内の命令列を生成する、
ステップをコンピュータに実行させるための（８）～（１１）のいずれかに記載のコンパイル用プログラム。
（１４）第１の依存グラフ情報には、Ｓ－ＰＥＩからそのＳ－ＰＥＩの次に順番の早いＳ－ＰＥＩへ例外依存アーク（以下、この例外依存アークを「Ｓ－ＰＥＩ間例外依存アーク」と言う。）の設定情報を含ませ、
遅くともスケジューリングより前に実行される第３及び第４の追加ステップを追加し、
第３の追加ステップでは、
Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が同一である場合には、該Ｓ－ＰＥＩ間例外依存アークを除去し、また、Ｓ－ＰＥＩ間例外依存アークの両端ノードに係る例外の種類が異なる場合には、該Ｓ－ＰＥＩ間例外依存アークをそのまま残し、
第４の追加ステップでは、
第２の依存グラフ情報に、第３の追加ステップにおけるＳ－ＰＥＩ間例外依存アークの変更を含ませる、
ステップをコンピュータに実行させるための（８）～（１３）のいずれかに記載のコンパイル用プログラム。
【００３６】
【発明の効果】
こうして、例外依存に対して的確な投機実行を行うコンパイル装置及びコンパイル方法が提供される。
【図面の簡単な説明】
【図１】先行技術におけるコンパイル過程の中間表現の説明図である。
【図２】図１の中間表現に対応する依存グラフを示すである。
【図３】Ｓ－ＰＥＩを越えた投機実行を行う処理手順のフローチャートである。
【図４】例外依存アークを生成する手順をプログラム記述形式で示している図である。
【図５】例外発生の順序を決定するためにＳ－ＰＥＩとＳ－ＰＥＩの間にも例外依存アークを生成する手順をプログラム記述形式で示している図である。
【図６】第１のソース・プログラムに対して、本発明を適用したコンパイラによるコンパイルを実行したときの中間表現の説明図である。
【図７】図６の中間表現に対応する依存グラフを示すである。
【図８】第２のソース・プログラムに対して、本発明を適用したコンパイラによるコンパイルを回復コード無しで実行したときの中間表現の説明図である。
【図９】図８の中間表現に対応する依存グラフを示すである。

【図１０】第２のソース・プログラムに対して、本発明を適用したコンパイラによるコンパイルを回復コード有りで実行したときの中間表現の説明図である。
【図１１】図１０の中間表現に対応する依存グラフを示すである。
【図１２】第３のソース・プログラムに対して、本発明を適用したコンパイラによるコンパイルを回復コード有りで実行したときの中間表現の説明図である。
【図１３】図１２の中間表現に対応する依存グラフを示すである。
【図１４】ベンチマークテストにおける従来のコンパイラと本発明を適用するコンパイラとの性能対比図である。
【図１５】コンパイル装置の機能ブロック図である。
【符号の説明】
２０　　コンパイル装置
２４　　第１の中間表現プログラム作成手段
２５　　ブロック抽出手段
２６　　第１の依存グラフ情報作成手段
２７　　Ｓ－ＰＥＩ間例外依存アーク設定手段
３０　　有利性判断手段
３１　　第２の依存グラフ情報作成手段
３４　　第１の復帰設定手段
３５　　第２の復帰設定手段
３６　　Ｓ－ＰＥＩ間例外依存アーク変更手段
３７　　第２の中間表現プログラム作成手段
３８　　オブジェクト・プログラム作成手段

【図１】

Javaのソースプログラム

```
int foo() {
  Cls1 a = this.field;
  Cls2 i = a.field2 + 1;
  return i;
}
```

チェック命令分解前のコンパイラの中間表現

```
add       r2 = r1, 16  // r1 = this
ld        r3 = [r2]
nullcheck r3
add       r4 = r3, 24
ld        r5 = [r4]
add       r6 = r5, 1
ret       r6
```

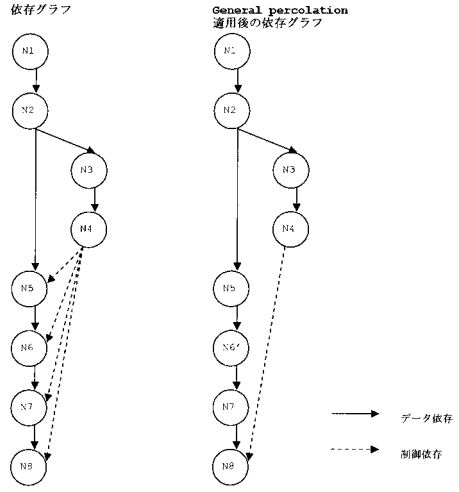
チェック命令分解後のコンパイラの中間表現

```
N1 :add     r2 = r1, 16  // r1 = this
N2 :ld      r3 = [r2]
N3 :cmp     cr1 = r3, 0
N4 :jmp     cr1, eq, ThrowNullException
N5 :add     r4 = r3, 24
N6 :ld      r5 = [r4]
N7 :add     r6 = r5, 1
N8 :ret     r6
```

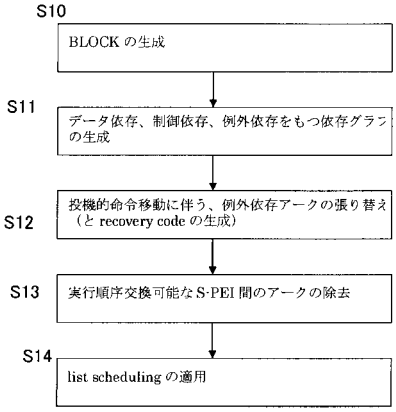
List scheduling後のコンパイラの中間表現

```
N1 :add     r2 = r1, 16  // r1 = this
N2 :ld      r3 = [r2]
N3 :cmp     cr1 = r3, 0
N5 :add     r4 = r3, 24
N6':ld.s    r5 = [r4]    // speculative load
N7 :add     r6 = r5, 1
N4 :jmp     cr1, eq, ThrowNullException
N8 :ret     r6           // sentinel
```

【図２】



【図３】



【図４】

```
ih<入力>:   H-PEI 命令
is, id:     命令
uset:       src オペランドの集合
u, uc:      src オペランド
d:          dst オペランド

uset = src operands(ih)
foreach (u ⊂ uset) {
  foreach (d ⊂ Pred(u)) {
    foreach (uc ⊂ Succ(d)) {
      is = instruction(uc)
      if (is が S-PEI である) {
        is と ih の間に例外依存アークを張る
      }
    }
    id = instruction(d)
    if (id が、new/newarray/getfield/getstatic などのオブジェクトを生成する命令でない) {
      uset = uset ∪ src operands(id)
    }
  }
}
```

【図５】

```
依存グラフ
G<入力>:
命令
is, ie, i:
v[# of instructions(G)]: boolean

v = F
foreach (i ⊂ instruction by reverse post order(G)) {
  if (visited[i]) { continue }
  visited[i] = T;
  if (i が S-PEI である) {
    is = I
    break
  }
}

foreach (ie ⊂ succeeding instruction by reverse post order(is)) {
  if (visited[ie]) { continue }
  visited[ie] = T;
  if (ie が S-PEI である) {
    is と ie の間に例外依存アークを張る
    is = ie
  }
}
```

【図６】

Java のソースプログラム

```
int foo() {
  Cls1 a = this.field;
  Cls2 i = a.field2 + 1;
  return i;
}
```

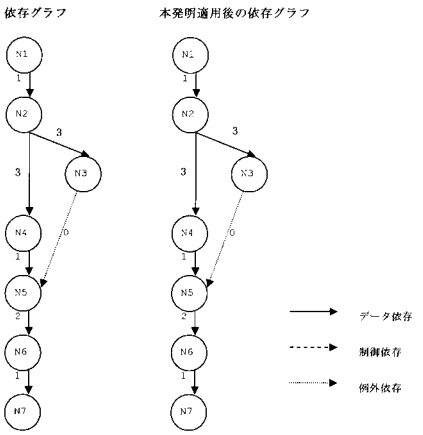
コンパイラの中間表現

```
N1:add        r2 = r1, 16  // r1 = this
N2:ld         r3 = [r2]
N3:nullcheck  r3
N4:add        r4 = r3, 24
N5:ld         r5 = [r4]
N6:add        r6 = r5, 1
N7:ret        r6
```

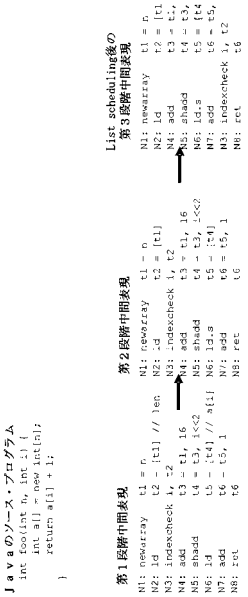
List scheduling 後のコンパイラの中間表現

```
N1:add        r2 = r1, 16  // r1 = this
N2:ld         r3 = [r2]
N4:add        r4 = r3, 24
N3:nullcheck  r3
N5:ld         r5 = [r4]
N6:add        r6 = r5, 1
N7:ret        r6
```

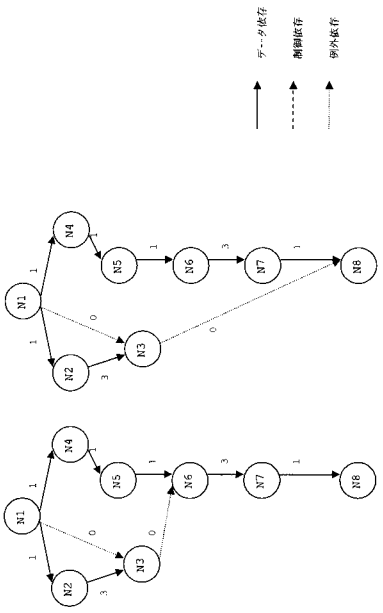
【図７】


依存グラフ
本発明適用後の依存グラフ
N1
N2
N3
N4
N5
N6
N7
データ依存
制御依存
例外依存

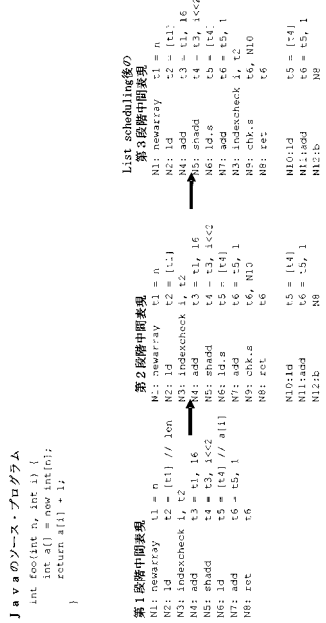
【図８】


Ｊａｖａのソース・プログラム
int foo(int n, int i) {
int a[] = new int[n];
return a[i] + 1;
}
第１段階中間表現
N1: newarray t1 = n,
N2: ld t2 = [t1] // len
N3: indexcheck i, t2,
N4: add t3 = t1, 16
N5: shadd t4 = t3, i<<2
N6: ld t5 = [t4] // a[i]
N7: add t6 = t5, 1
N8: ret t6
第２段階中間表現
N1: newarray t1 = n
N2: ld t2 = [t1] // len
N3: indexcheck i, t2, N8
N4: add t3 = t1, 16
N5: shadd t4 = t3, i<<2
N6: ld.a t5 = [t4] // a[i]
N7: add t6 = t5, 1
N8: ret t6
List scheduling後の
第３段階中間表現
N1: newarray t1 = n
N2: ld t2 = [t1]
N4: add t3 = t1, 16
N5: shadd t4 = t3, i<<2
N6: ld.a t5 = [t4]
N3: indexcheck i, t2, N8
N7: add t6 = t5, 1
N8: ret t6

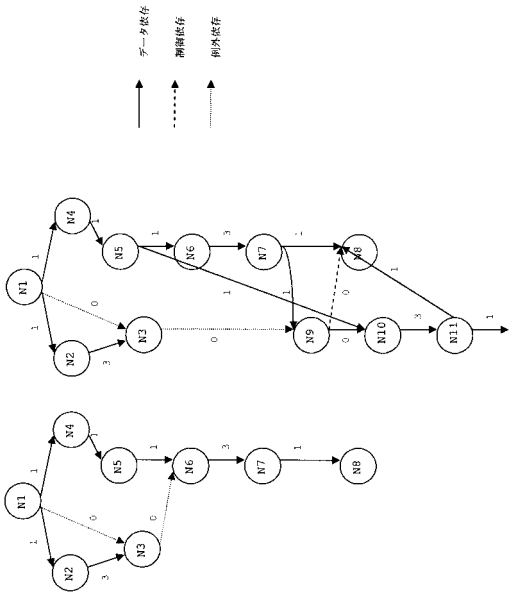
【図９】


データ依存
制御依存
例外依存
N1
N2
N3
N4
N5
N6
N7
N8

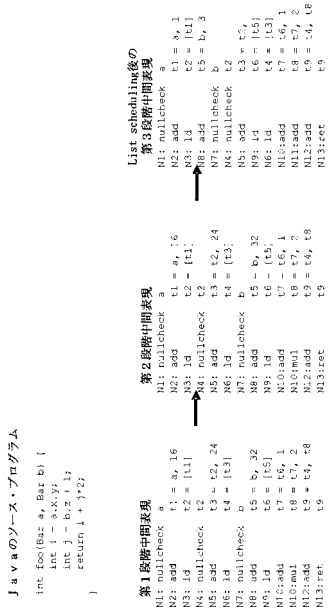
【図１０】


Ｊａｖａのソース・プログラム
int foo(int n, int i) {
int a[] = new int[n];
return a[i] + 1;
}
第１段階中間表現
N1: newarray t1 = n
N2: ld t2 = [t1] // len
N3: indexcheck i, t2, N8
N4: add t3 = t1, 16
N5: shadd t4 = t3, i<<2
N6: ld t5 = [t4] // a[i]
N7: add t6 = t5, 1
N8: ret t6
第２段階中間表現
N1: newarray t1 = n
N2: ld t2 = [t1]
N4: add t3 = t1, 16
N5: shadd t4 = t3, i<<2
N6: ld.a t5 = [t4]
N3: indexcheck i, t2, N10
N9: chk.a t6, N10
N8: ret t6
N10:ld t5 = [t4]
N11:add t6 = t5, 1
N12:b N8
List scheduling後の
第３段階中間表現
N1: newarray t1 = n
N2: ld t2 = [t1]
N4: add t3 = t1, 16
N5: shadd t4 = t3, i<<2
N6: ld.a t5 = [t4]
N3: indexcheck i, t2, N12
N7: add t6 = t5, 1
N8: ret t6
N10:ld t5 = [t4]
N11:add t6 = t5, 1
N12:b N8

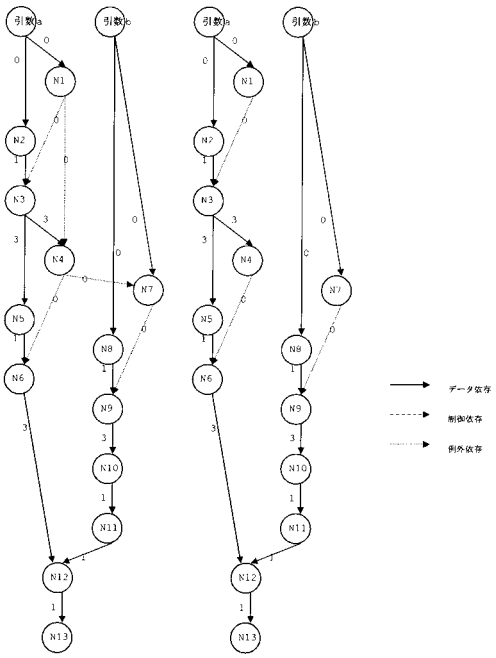
【図１１】



【図１２】



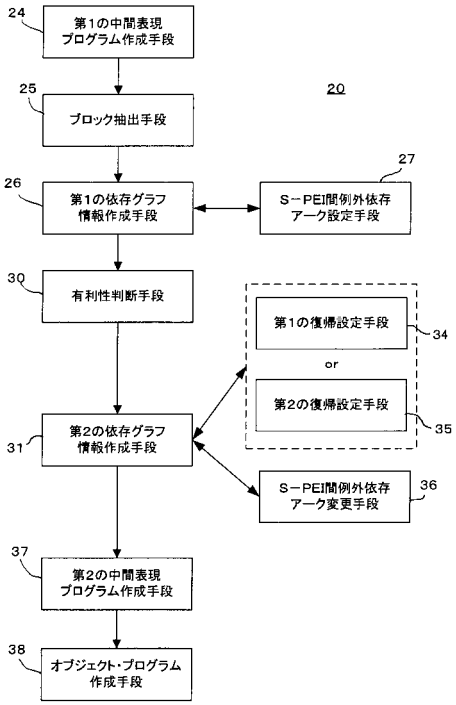
【図１３】



【図１４】

	本発明適用のコンパイラ	従来方式のコンパイラ
PDGノードの数	37327	142679
１つのPDGノードが含む平均命令数。	4.71	1.23
PDGノード間のアーク数	54669	262791
PDGノード内の命令ノード間のアーク数	145952	80190

【図１５】


24
第1の中間表現
プログラム作成手段
25
ブロック抽出手段
20
27
26
第1の依存グラフ
情報作成手段
S－PEI間例外依存
アーク設定手段
30
有利性判断手段
第1の復帰設定手段
34
or
第2の復帰設定手段
35
31
第2の依存グラフ
情報作成手段
S－PEI間例外依存
アーク変更手段
36
37
第2の中間表現
プログラム作成手段
38
オブジェクト・プログラム
作成手段

フロントページの続き

(72)発明者 石崎 一明
神奈川県大和市下鶴間１６２３番地１４ 日本アイ・ビー・エム株式会社 東京基礎研究所内
(72)発明者 稲垣 達氏
神奈川県大和市下鶴間１６２３番地１４ 日本アイ・ビー・エム株式会社 東京基礎研究所内
(72)発明者 小松 秀昭
神奈川県大和市下鶴間１６２３番地１４ 日本アイ・ビー・エム株式会社 東京基礎研究所内

審査官 久保 光宏

(56)参考文献 特許第３７０７７２７（ＪＰ，Ｂ２）
特開２０００－６６８９８（ＪＰ，Ａ）
Carole Dulong, 「具体例によるIA-64アーキテクチャの紹介」, 情報処理, 日本, 社団法人情報処理学会, １９９８年１２月１５日, Vol.39, No.12, pp.1225-1232, ISSN:0447-8053
中西知嘉子・他, 「パス選択によるソフトウェアパイプライニング」, 電子情報通信学会論文誌, 日本, 社団法人電子情報通信学会, １９９７年 9月２５日, Vol.J80-D-I, No.9, pp.774-786, ISSN:0915-1915

(58)調査した分野(Int.Cl., ＤＢ名)
G06F9/45,
G06F9/38,
CSDB(日本国特許庁),
JSTPlus(JDream2)