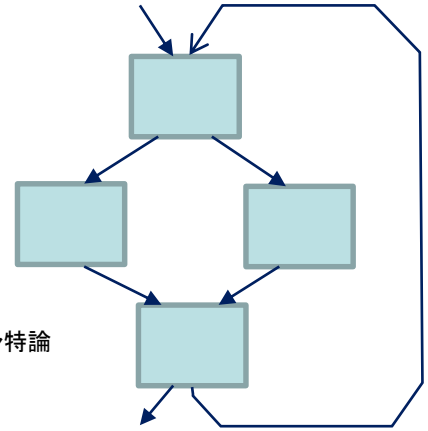
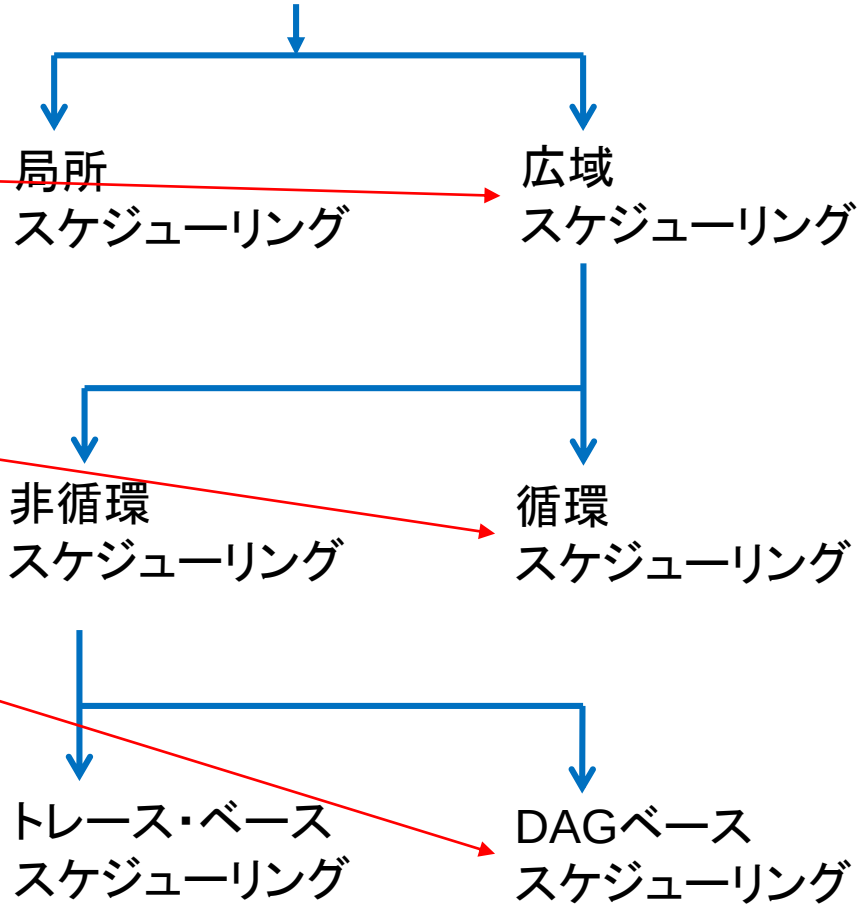
内容

- 非循環スケジューリング
  - パーコレーション・スケジューリング[Nicolau 1985, Aiken 1988]
  - トレース・スケジューリング[Fisher 1981]
  - スーパブロック・スケジューリング[Chang 1991]
- ループ最適化（循環スケジューリング）
  - ループ・アンローリング
  - ソフトウェア・パイプライニング[Rau 1981, Lam 1988]

スケジューリング・アルゴリズムの分類

- 基本ブロック内
- 基本ブロックを超える
- ループがない
- ループ
- 単一制御フロー
- 複数制御フロー





マイクロコード圧縮技術

- 局所スケジューリングの後、空いているスロットを広域スケジューリングにより埋める
  - e.g., [Tokoro 1981]

- 隣接する基本ブロック間での基本命令移動を繰り返す
  - 結果的に、遠い命令が移動される
  - 欠点
    - 広域的視野の欠落
    - コード全体の利益になっているかどうかはわからない

パーコレーション・スケジューリング [Nicolau 1985, Aiken 1988]

- 上方命令移動のみ

- 合流点を超える命令移動
- 分岐を超える命令移動による単一化
- 合流点を超える分岐命令の移動
  - 制御フローグラフの変形が起こる
  - 非常に複雑

トレース･スケジューリング
[Fisher 1981]

- 商用化
  - Multiflowマシン
  - 500K行のCコード
  - RISCに負ける

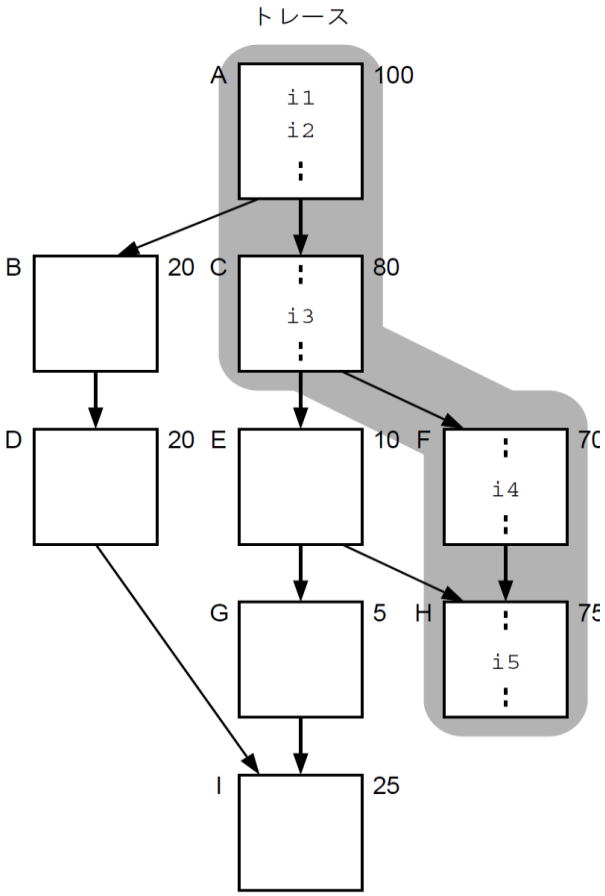
- 特徴
  - 複数のBBからなるトレースを単一のBBのように扱う
  - 広域的視野：命令移動の利益が計算できる
    - 後の広域スケジューリング手法に多大なる影響を与えた
  - 数値計算に有効
  - 補償コードによりトレース外に大きなペナルティを与える
  - 複雑なブックキーピング

TSのアルゴリズム

- トレースの選択
  - プロファイル・ベース
  - 頻繁に実行されるトレースを選択
- トレースのスケジューリング
  - リスト・スケジューリング
- ブックキーピング[Ellis 1986]
  - トレース外に補償コードを挿入

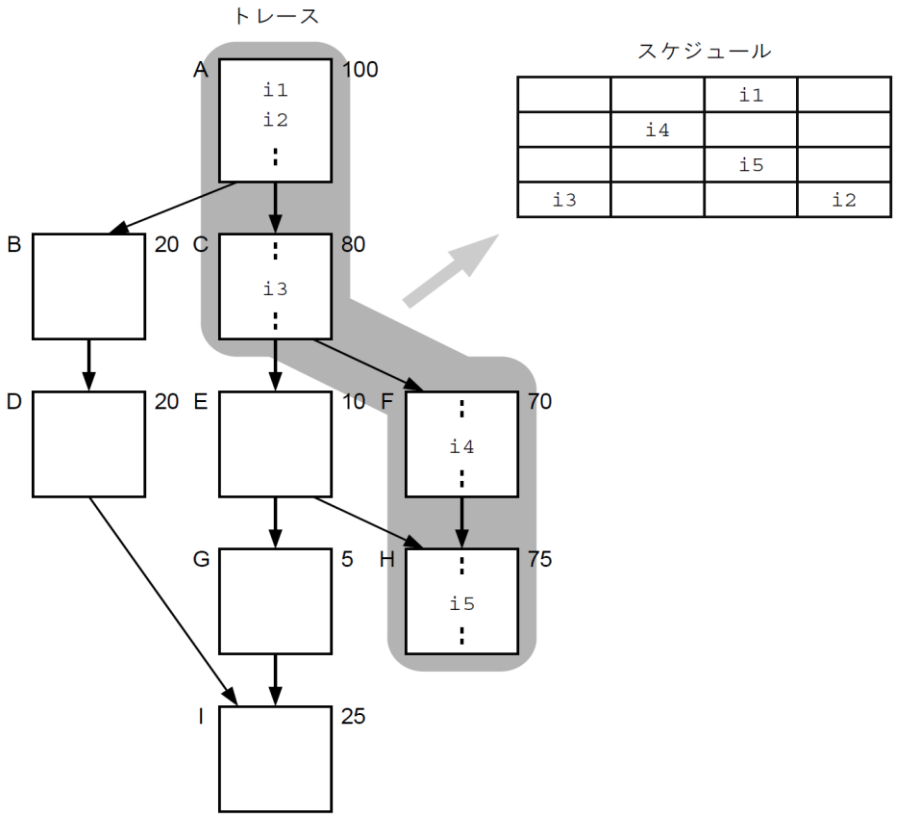
トレースの選択

- シードを選択
  - 実行頻度が最も高いBB
- トレースを上下に成長
  - 実行頻度が高い方に向かって
- 成長を停止
  - call/return
  - ループ境界
  - 既スケジュールブロック



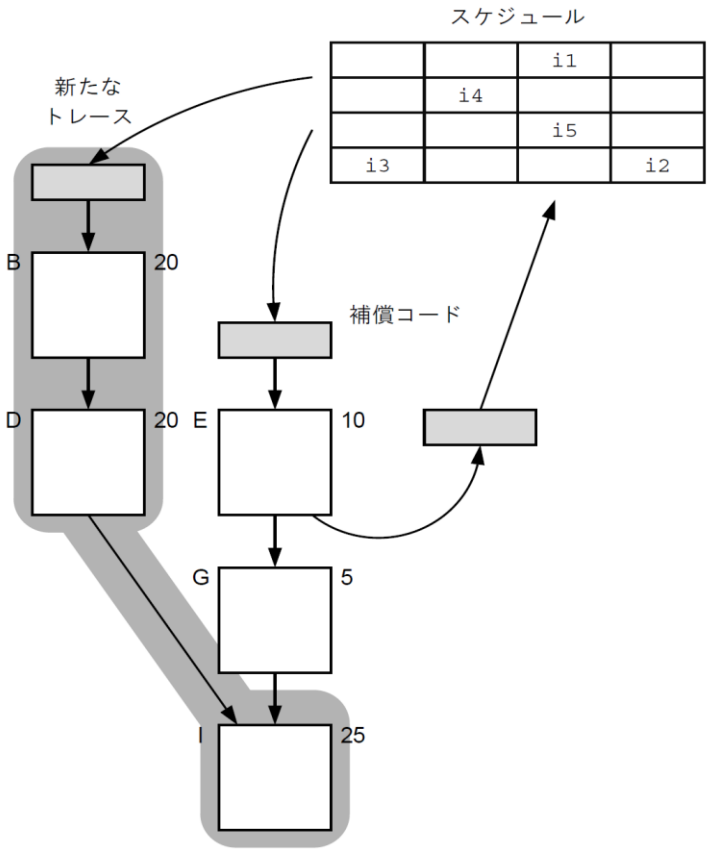
トレースのスケジューリング

- リスト・スケジューリング（＋レジスタ割り当て）



ブックキーピング

- スケジュールされたコードをフローグラフに戻す
- 補償コードを挿入[Ellis 1986]
  - 分岐点を越える上/下方移動
  - 合流点を越える上方移動
  - 分岐も移動
- 分岐点を越える上方移動（投機）では、レジスタ割り当て変更
- 欠点
  - 複雑なブックキーピング
  - トレース外コードにペナルティ
  - コード量増加

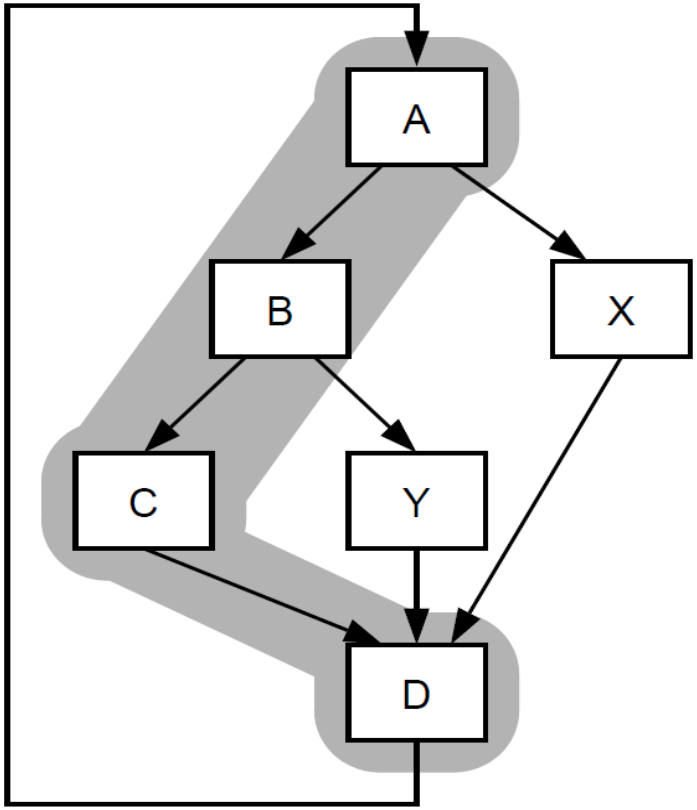


スーパブロック・スケジューリング [Chang 1991]

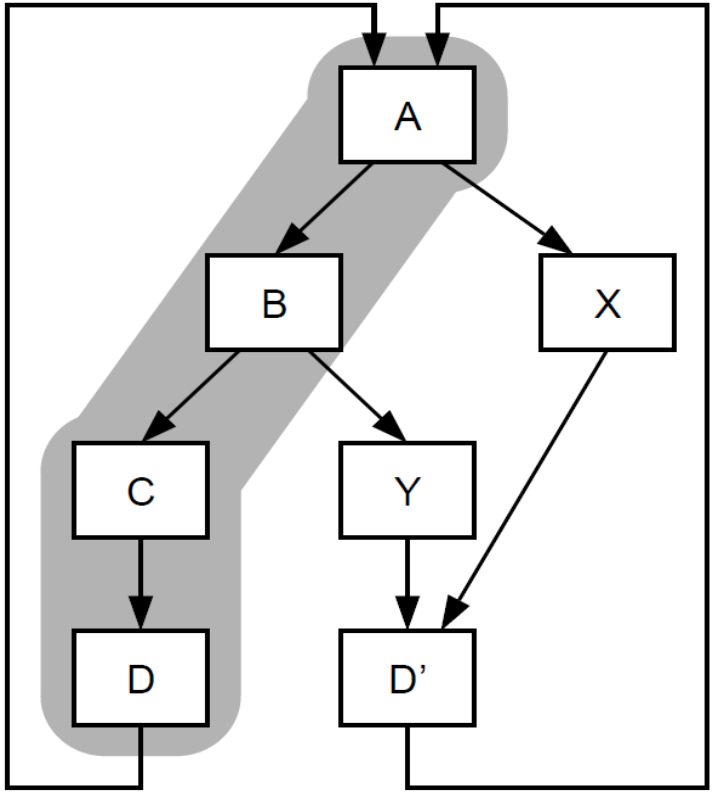
- IMPACTコンパイラ
  - University of Illinois
  - 92K行のCコード

- 特徴
  - トレース・ベース
  - スーパブロックを形成することで、ブックキーピングを簡単化
    - 合流点以降のBBを複写することで、合流点を削除
    - 上方命令移動のみ
  - 種々の最適化が容易
  - BB複写→コード量増加
    - 不効率←移動されないかもしれないコードも複写される

スーパブロック形成



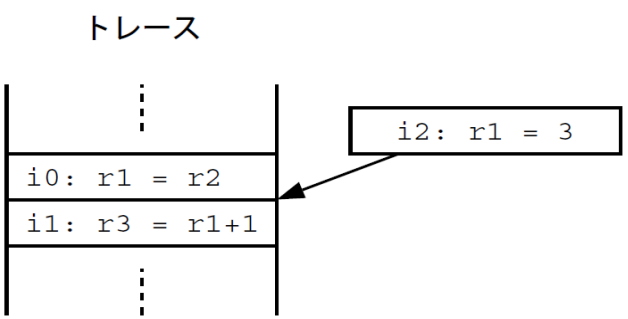
トレース選択



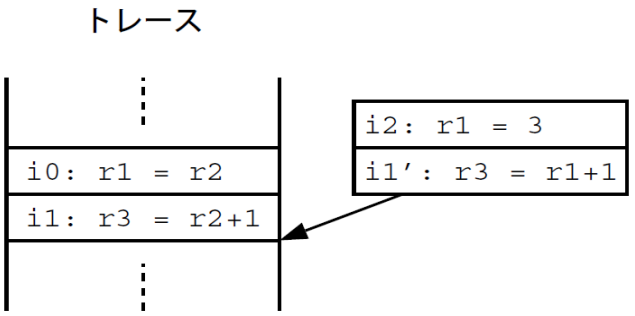
末尾複写

最適化の例:コピー伝搬

- copy→use依存を切断
  - コピー元を代入
- TSでは複雑



コピー伝搬前

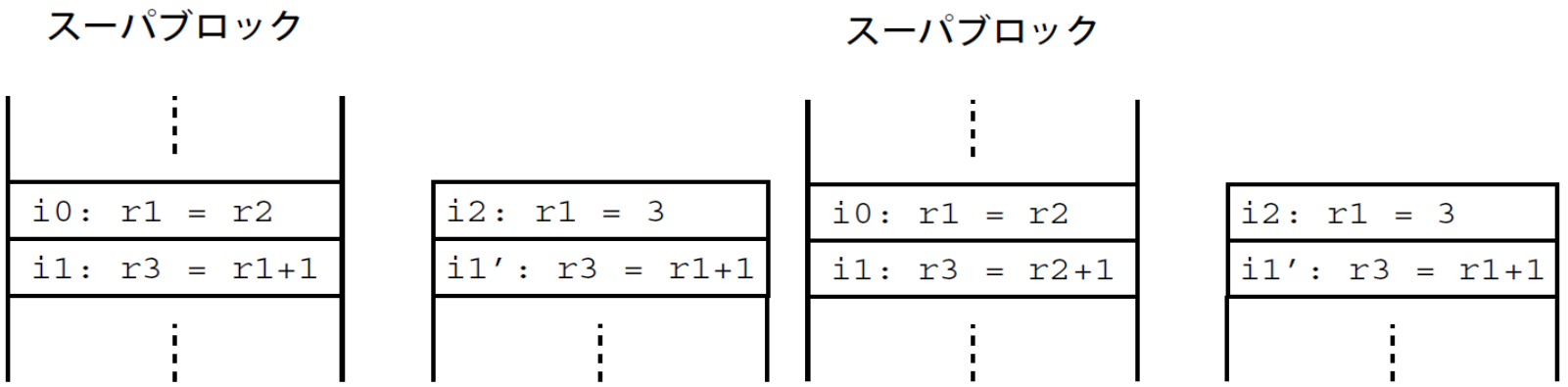


コピー伝搬後

  - 合流点の移動
  - 補償コードの挿入

最適化の例:コピー伝搬(cont'd)

- スーパブロックでは簡単



コピー伝搬前　　コピー伝搬後

内容

- 非循環スケジューリング
  - パーコレーション・スケジューリング[Nicolau 1985, Aiken 1988]
  - トレース・スケジューリング[Fisher 1981]
  - スーパブロック・スケジューリング[Chang 1991]

- ループ最適化（循環スケジューリング）
  - ループ・アンローリング
  - ソフトウェア・パイプライニング[Rau 1981, Lam 1988]

ループ・アンローリング

- ソース・コード

```
int i, x[1000], s;
for (i = 0; i < 1000; i++)
   x[i] = x[i] + s;
```

- アセンブリ・コード

```
    Loop:
i1:     r4 = load 0(r1)
        nop
i2:     r4 = r4 + r3
i3:     store 0(r1) = r4
i4:     r1 = r1 + 4
i5:     branch Loop if r1!=r2
        nop
```

  – ループ・ボディに並列性はない

2回アンロール

- コンパイラにループ･レベル並列性を露出

```
    Loop:
i11:   r4 = load 0(r1)      #1回目の始まり
       nop
i12:   r4 = r4 + r3
i13:   store 0(r1) = r4
i14:   r1 = r1 + 4
i21:   r4 = load 0(r1)      #2回目の始まり
       nop
i22:   r4 = r4 + r3
i23:   store 0(r1) = r4
i24:   r1 = r1 + 4
i5:    branch Loop if r1!=r2
       nop
```

最適化

i11: r14=load 0(r1)	i21: r24=load 4(r1)
i24: r1=r1+8	
i12: r14=r14+r3	i22: r24=r24+r3
i13: store -8(r1)=r14	i25: branch Loop if r1!=r2
	i23: store -4(r1)=r24

- レジスタ・リネーミング

誘導変数に関する最適化

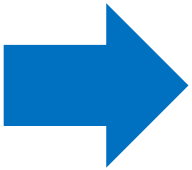
- 誘導変数：ループを回るごとに定数だけ増加または減少される変数

```
Loop:
        r1 = r1 + 4
        beq ... Loop
```

- 最適化
  - 削除、代入

誘導変数の削除

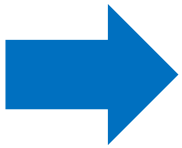
```
…
r1 = r1 + 4
…
r1 = r1 + 4
```



```
…
…
…
r1 = r1 + 8
```

誘導変数の代入

```
...
store 0(r1) = r4
r1 = r1 + 4
r4 = load 0(r1)
...
store 0(r1) = r4
r1 = r1 + 4
```



```
r41 = r4
r4 = load 4(r1)
...
r1 = r1 + 8
store -8(r1) = r41
...
store -4(r1) = r4
```

最適化

i11: r14=load 0(r1)	i21: r24=load 4(r1)
i24: r1=r1+8	
i12: r14=r14+r3	i22: r24=r24+r3
i13: store -8(r1)=r14	i25: branch Loop if r1!=r2
	i23: store -4(r1)=r24

- レジスタ・リネーミング
- 誘導変数の代入と削除
  - 誘導変数：ループを回る度に一定値だけ増加または減少する変数
- リスト・スケジューリング
- 性能向上
  - 7サイクル/イタレーション→2.5サイクル/イタレーション

長所と短所

- 長所
  - ループ・レベル並列を利用
  - 最適化が容易
  - スケジューリングが容易（ループを非循環グラフに変換）

- 短所
  - コード量増加
  - 何回アンロールすれば良いか、スケジュールしてみないとわからない
    - 発見的に決定
  - 一般に、最初と最後のサイクル周辺で並列度が低くなる

内容

- 非循環スケジューリング
  - パーコレーション・スケジューリング[Nicolau 1985, Aiken 1988]
  - トレース・スケジューリング[Fisher 1981]
  - スーパブロック・スケジューリング[Chang 1991]
- ループ最適化（循環スケジューリング）
  - ループ・アンローリング
  - ソフトウェア・パイプライニング[Rau 1981, Lam 1988]

ループ

```
int i;
float a[N], b, c;
for (i = 0; i < N; i++)
   a[i] = a[i] * b + c;
```

1イタレーションのコード

ライン	コード
0	load
1	fmul
2	
3	fadd
4	
5	store

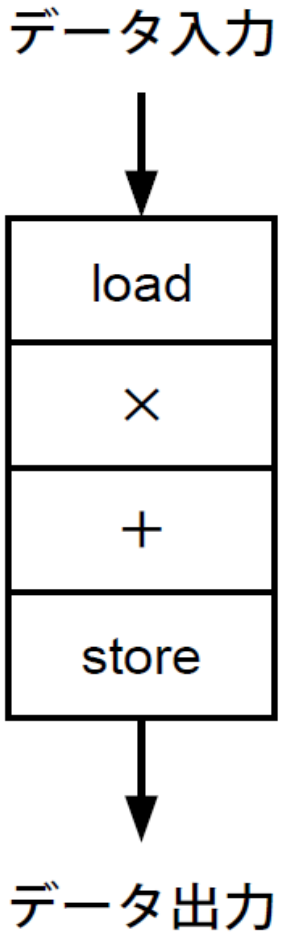
ソフトウェア・パイプライニング

8イタレーションのループの実行

サイクル	イタレーション							
	0	1	2	3				
0	load							
1	fmul	load						
2		fmul	load					
3	fadd		fmul	load				
4		fadd		fmul	load			
5	store		fadd		fmul	load		
6		store		fadd		fmul	load	
7			store		fadd		fmul	load
8				store		fadd		fmul
9					store		fadd	
10						store		fadd
11							store	
12								store

あるイタレーションの実行を
その前のイタレーションの実行が
終了する前に開始する

ハードウェア・イメージ



パイプライン化された
データパス

ソフトウェア・パイプライニング

8イタレーションのループの実行

サイクル	イタレーション							
	0	1	2	3	4	5	6	7
0	load							
1	fmul	load						
2		fmul	load					
3	fadd		fmul	load				
4		fadd		fmul	load			
5	store		fadd		fmul	load		
6		store		fadd		fmul	load	
7			store		fadd		fmul	load
8				store		fadd		fmul
9					store		fadd	
10						store		fadd
11							store	
12								store

これが繰り返されている

SPのコード

ライン		LOAD	FMUL	FADD	STORE	JUMP
0		load				
1		load	fmul			
2		load	fmul			
3		load	fmul	fadd		
4		load	fmul	fadd	store	
5	Loop:	load	fmul	fadd	store	jump Loop
6			fmul	fadd	store	
7				fadd	store	
8				fadd	store	
9					store	
10					store	

- 5: カネール
- 0～4: プロローグ
- 6～10: エピローグ

・開始間隔II(initiation interval)=1

ループ・アンローリングとの比較

- SPの方がLUより並列度が高い
  - オーバヘッドが小さい
  - LUではアンロールされたループ・イタレーション間の並列性しか利用できないが、SPにはそのような制限はない
- SPの方がコード量増加が少ない
  - LUではアンロール回数倍に増加
  - SPではプロローグとエピローグのみ

まとめ(1)

- 非循環スケジューリング
  - パーコレーション・スケジューリング
    - 隣接BB間での命令移動を繰り返す
    - 広域的視野を欠く
  - トレース・スケジューリング
    - 頻繁に実行されるパスを1つのBBのようにしてスケジュール
    - 広域的視野による最適化
    - 非常に複雑
  - スーパブロック・スケジューリング
    - TSを末尾複写により簡単化
    - 古典的最適化[Aho 1986]が容易に行える

まとめ(2)

- ループ最適化
  - ループ・アンローリング
    - ループレベル並列をコンパイラに露出
    - 古典的最適化の適用
  - ソフトウェア・パイプライニング
    - あるイタレーションの実行を、それが終了する前に次のイタレーションの実行を開始する
    - 1イタレーションの実行時間は短くならないが、スループットは高くなる
    - Intel Itanium

期末試験

- 1/17
- A4用紙1枚（表裏）のメモ持ち込み可

- 過去問をHPにアップロード