コンパイラ演習
第 10 回
(2011/12/15)

中村 晃一　野瀬 貴史　前田 俊行
秋山 茂樹　池尻 拓朗
鈴木 友博　渡邊 裕貴
潮田 資秀
小酒井 隆広
山下 諒蔵　佐藤 春旗
大山 恵弘　佐藤 秀明
住井 英二郎

今回の内容

- ループ最適化
- 並列化

ループ最適化とは

- ループの構造を変形する最適化
  - キャッシュ効率向上
  - 並列性向上
  - などの効果がある
- 今回は第 8 回に検出方法を示した Do-All 型ループのみを対象とする

Do-All型ループとは

- i = a, a + d, a + 2d, ..., a + (n-1)d と回す形のループ
  - a,d,m はループ実行中は定数でなければならない
  - ループボディに i への代入が含まれていてはならない

```
for (i = a; i < m; i += d) {
    ...
    ...
    ...
}
```

ループの正規化

- i = 0, 1, 2, ... と回す形のループへ
  あらかじめ変換しておくと以後が楽になる

```
for (i = a; i < m; i += d) {
    ...
    ... i ...
    ...
}
```

```
for (i = 0; i < (m-a+d)/d; i++) {
    ...
    ... a + i*d ...
    ...
}
```

完全ネストと不完全ネスト

完全ネスト

```
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        ...
    }
}
```

不完全ネスト

```
for (i = 0; i < m; i++) {
    ...
    for (j = 0; j < n; j++) {
        ...
    }
    ...
}
```

- 完全ネストではループ全体を丸ごと最適化する事が容易
- 不完全ネストの場合は
  (基本的に) 内側のループから個別に最適化を行う

- これからまずループ変換の例を示します
  - Loop Unrolling
  - Loop Interchange
  - Loop Fusion/Distribution
- その後必要な解析について説明します

Loop Unrolling

- 効果
  - ジャンプのオーバヘッド削減
  - スケジューリングの効果向上
- 副作用
  - レジスタ負荷増大
  - コードサイズ増大

```
for (i = 0; i < n; i++) {
    .. a[i] ..
}
```

```
for (i = 0; i < n; i+=3) {
    .. a[i] ..
    .. a[i+1] ..
    .. a[i+2] ..
}
for (i = i-3; i < n; i++) {
    .. a[i] ..
}
```

Loop Interchange

- 効果
  - キャッシュ効率が向上する
    (配列のレイアウト・サイズに依る)

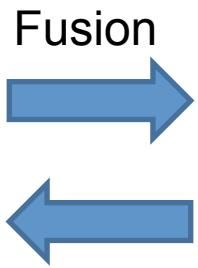
```
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        .. a[j][i]
    }
}
```

```
for (j = 0; j < n; j++) {
    for (i = 0; i < m; i++) {
        .. a[j][i] ..
    }
}
```

Loop Fusion/Distribution

- Fusionの効果
  - ループのオーバヘッド除去
- Distributionの効果
  - 参照の局所性を高める

```
for (i = 0; i < m; i++) {
    ... // statement 1
}
for (i = 0; i < m; i++) {
    ... // statement 2
}
```




```
for (i = 0; i < m; i++) {
    ... // statement 1
    ... // statement 2
}
```

ループ依存検査

- ループの構造を
  好き勝手に変換してよいわけではない
- スケジューリングの回に登場した
  以下の依存関係にある命令の順序を
  変えてはならない
  - Write after Write依存
  - Write after Read依存
  - Read after Write依存

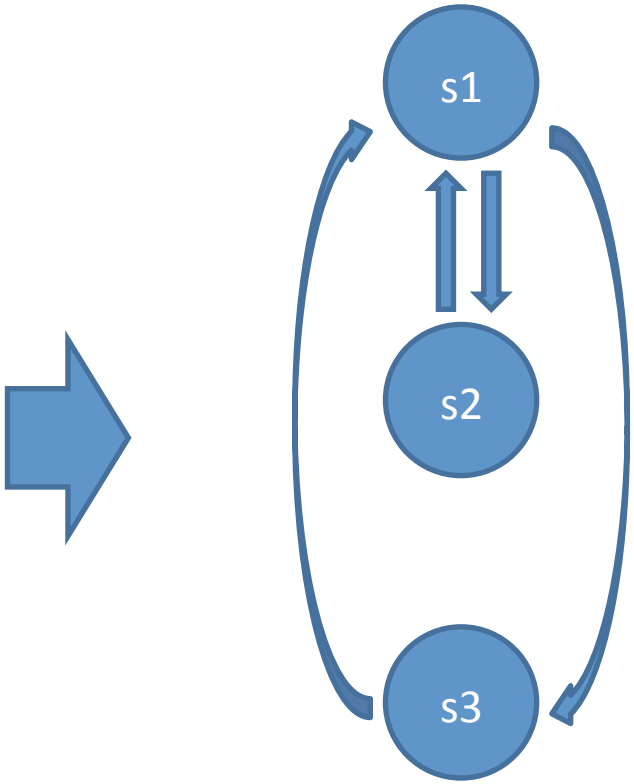
ループ独立依存とループ繰越し依存

- ループ独立依存
  - 同じイテレーション内での依存
  - ループ構造のみの変形によっては変化しない
- ループ繰越し依存
  - 異なるイテレーション間での依存
  - ループ構造の変形によって変わりうる

ループ依存グラフ

- 各命令間の依存関係をグラフにしたもの
  - 以降、先行するステートメントがグラフ上で上にあると考える

```
for (i = 0; i < m; i++) {
    s1: t = 2*a[i]
    s2: a[i+1] = b[i] + t
    s3: c[i] = b[i] + 3*t
}
```



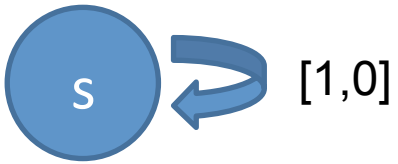
Loop Unrolling ができる条件

- イテレーションの実行順序を変更しないので常に実行可能

Loop Interchange ができる条件

- **依存距離**を考える。
  - **依存距離**: 配列アクセスに使用される
    ループインデックス [i,j] の差分

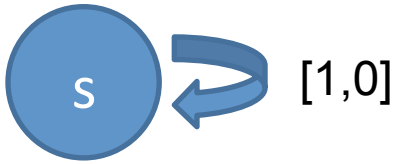
```
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        s: a[i+1][j] = a[i][j] + b[i][j]
    }
}
```



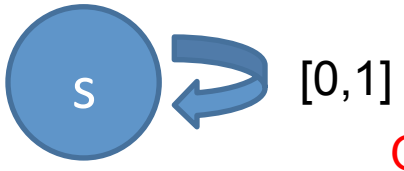
- 変換後、全ての依存距離の
「最も左の0でない要素が正」
でなければならない

Loop Interchange ができる例

```
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        s: a[i+1][j] = a[i][j] + b[i][j]
    }
}
```



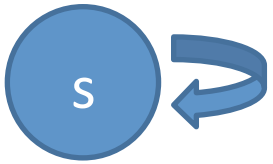
```
for (j = 0; j < n; j++) {
    for (i = 0; i < m; i++) {
        s: a[i+1][j] = a[i][j] + b[i][j]
    }
}
```



OK

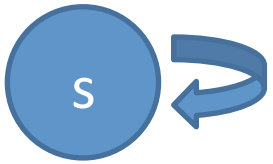
Loop Interchangeができない例

```
for (i = 0; i < m; i++) {
    for (j = 0; j < n; j++) {
        s: a[i+1][j] = a[i][j+1] + b[i][j]
    }
}
```



[1,-1]

```
for (j = 0; j < n; j++) {
    for (i = 0; i < m; i++) {
        s: a[i+1][j] = a[i][j+1] + b[i][j]
    }
}
```

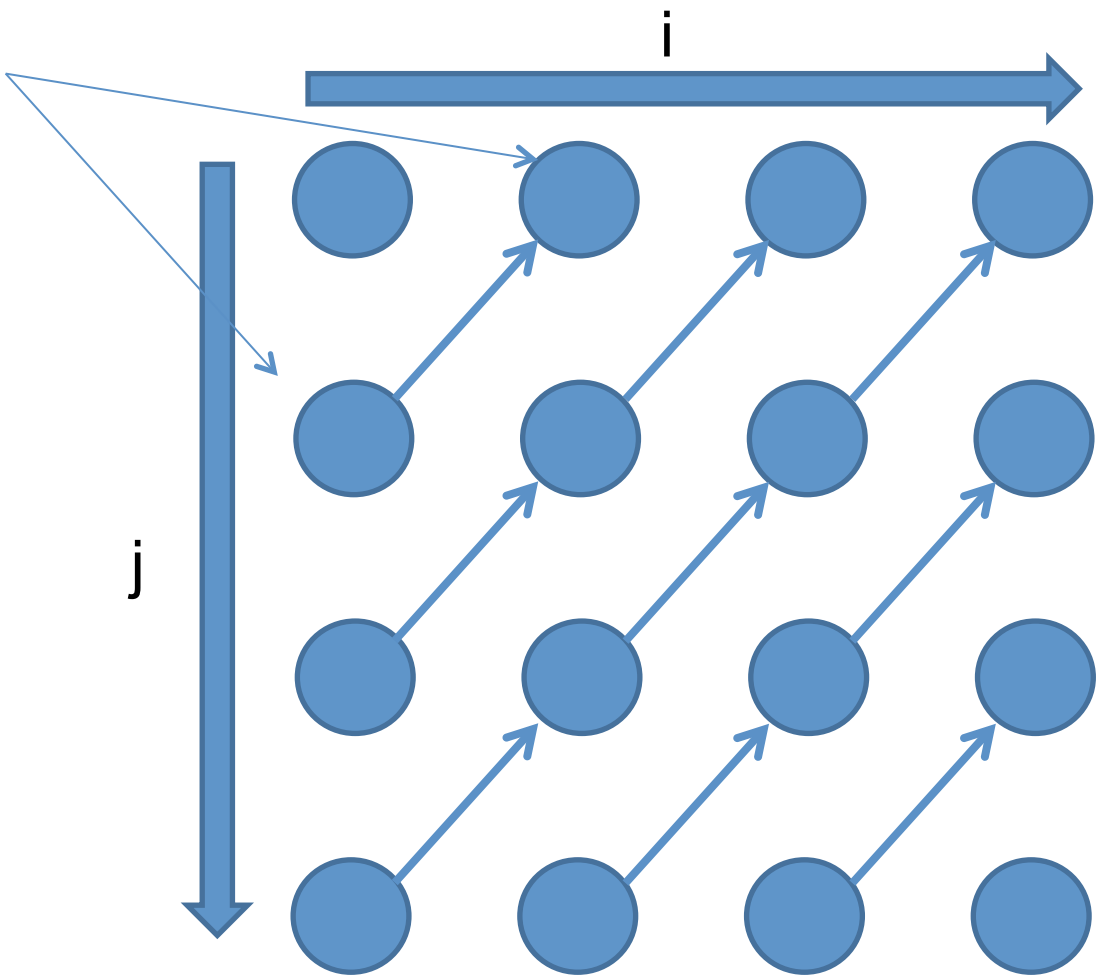


[-1,1]

NG

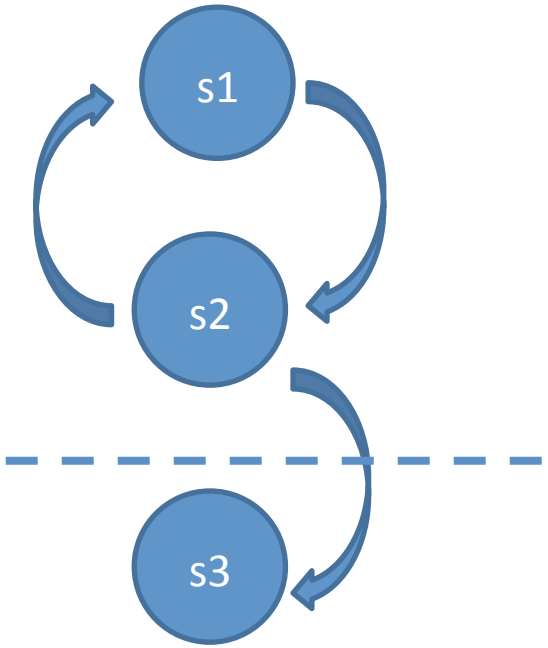
Loop Interchangeができない例の図

iから回すか
jから回すかで
アクセス順序が
変わってしまう



Loop Distribution ができる条件

- 依存グラフの上向きの辺を
  切らないければ良い




```
for (i = 0; i < m; i++) {
    s1: t = 2*a[i+1]
    s2: a[i] = b[i] + t
    s3: c[i] = b[i] + a[i]
}
```

```
for (i = 0; i < m; i++) {
    s1: t = 2*a[i+1]
    s2: a[i] = b[i] + t
}
for (i = 0; i < m; i++) {
    s3: c[i] = b[i] + a[i]
}
```

並列計算の分類

- 命令レベル並列
- データ並列計算
  - ベクトル計算
  - SIMD
- タスク並列計算
  - スレッド並列
  - プロセス並列

自動並列化の技術

- ループ並列化
  - Do-All型ループからベクトル演算・SIMD演算命令を使用するプログラムへ変換
- データ並列モデル
  - 型によってデータ並列性を明示する
- スレッド並列化
  - (プログラム)依存グラフを解析し並列化する

- 今回はループ並列化とデータ並列モデルからの変換を扱います

ループ並列化

```
for (i = 0; i < m; i++) {
    c[i] = a[i] * b[i+1]
}
```

```
c[0:m] = a[0:m] * b[1:m+1]
```

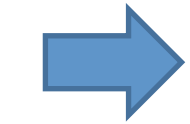
- a[x:y] で a[x]..., a[y-1] をまとめたベクトル変数を表す
- ベクトル変数をオペランドに持つ式は
  ベクトル演算を表す

- 変換できる条件
  - ループ繰越し依存が存在しない

Scalar Expansion

- スカラ変数を配列変数へと拡張

```
for (i = 0; i < m; i++) {
    t = a[i] * b[i+1]
    c[i] = 2 * t
}
```




```
for (i = 0; i < m; i++) {
    t[i] = a[i] * b[i+1]
    c[i] = 2 * t[i]
}
t = t[m-1]
```

```
t[0:m] = a[0:m] * b[1:m+1]
c[0:m] = 2 * t[0:m]
t = t[m-1]
```

- 変換できる条件
  - 全ての t の使用に到達する定義が
    同一イテレーション内にあるときに変換が可能

Loop strip mining

- SIMD演算器のベクトル長に fit させる為のループ変換
  - 評価順序を変えないので常に実行可能

```
for (i = 0; i < m; i++) {
    c[i] = a[i] * b[i+1]
}
```

```
for (i = 0; i < m; i+=4) {
    for (j = 0; j < 4; j++) {
        c[i+j] = a[i+j] * b[i+j+1]
    }
}
i = m
```

```
for (i = 0; i < m; i+=4) {
    c[i:i+4] = a[i:i*4] * b[i+1:i+5]
}
i = m
```

型によるデータ並列性

- ベクトル型とその演算

```
...
let p = v_add (v_scalar t dir) src in
let norm' = v_scalar (-1.0) norm in
opt_some (t, p, norm', obj)
...
```

- リスト型とその演算

```
let intersect_opts  = list_map (calc_intersect_obj src dir) and_net in
let intersect_opts' = list_filter (all_objs_contain_opt and_net) intersect_opts in
find_first_intersect intersect_opts'
```

共通課題

- 二つの共通課題のうち
  一つ以上を解いてください

課題 1

- ループ変換のうち最低一つを実装せよ
  - MinCaml もしくは自作コンパイラに実装せよ
- 変換結果を例示し
  最適化の効果について議論せよ

課題 2

- 既存の言語処理系の並列化技術 (もしくは並列化に関する言語設計) について論文を読み、要約せよ

- 例えば以下の論文でもよい
  - Z. Bozkus et al. “Fortran 90D/HPF compiler for distributed memory MIMD computers: design, implementation, and performance results.”, In *Proceedings of the 1993 ACM/IEEE conference on Supercomputing* (Supercomputing '93).
  - Chakravarty, Manuel et al. “Nepal — Nested Data Parallelism in Haskell”, Euro-Par 2001 Parallel Processing, Vol. 21050, pp. 524-534, 2001.
  - Philippe Charles et al. “X10: an object-oriented approach to non-uniform cluster computing. ”, *SIGPLAN Not.* 40, Vol. 10, pp. 519-538 (October 2005).
  - B.L. Chamberlain et al. “Parallel Programmability and the Chapel Language”, International Journal of High Performance Computing Application, Vol. 21, No. 3, pp. 291-312 (August 2007).
  - Carlson W, Draper J.M, Culler D.E, Yelick K, Brooks E and Warren K “Introduction to UPC and Language Specification”.

課題 2 (続き)

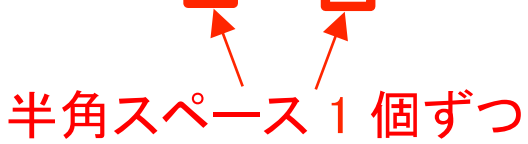
- Allen, Chase, Hallett, Luchangco, Maessen, Ryu, Steele, Tobin-Hochstadt, “The Fortress Language Specification version 1.0”
- D. Blumofe et al. “Cilk: An Efficient Multithreaded Runtime System”, Proceedings of the Fifth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pp. 207-216 (July 1995).
- など

コンパイラ係向け課題

- 自作コンパイラに自動並列化機能を実装せよ
  - この課題に限っては
    実機でなくシミュレータで動作すればよい

課題の提出先と締め切り

- 提出先: compiler-report-2011@yl.is.s.u-tokyo.ac.jp
- 締め切り: 3 週間後 (2012/1/5) の午後 1 時 (JST)
  - コンパイラ係向け課題締切 : 2012/2/27
- Subject: **`Report␣10␣`** <学籍番号: 5 桁>



  - 例 : **`Report 10 11099`**
- 本文にも氏名と学籍番号を明記のこと

◆ 質問は compiler-query-2011@yl.is.s.u-tokyo.ac.jp まで