Bab 9

Local Optimization dan Data Flow Analysis

Sub-CPMK yang Dicakup dalam Bab Ini:

- **Sub-CPMK 4.3:** Menganalisis dan mengoptimasi kode tingkat lokal

9.1 Pendahuluan Data-Flow Analysis

Data-Flow Analysis adalah teknik untuk mengumpulkan informasi tentang bagaimana nilai-nilai (*definitions*, *uses*) mengalir melalui Control Flow Graph (CFG) sebuah program.

9.1.1 Komponen Analisis

- **Transfer Function**: Aturan yang menjelaskan bagaimana instruksi dalam sebuah blok mengubah informasi aliran data.
- **Meet Operation**: Aturan untuk menggabungkan informasi dari beberapa jalur yang bertemu di satu *node*.

9.1.2 Arah Analisis

Analisis dapat dilakukan secara **Forward** (dari entry ke exit, misal: *Reaching Definitions*) atau **Backward** (dari exit ke entry, misal: *Live Variable Analysis*).

9.2 Reaching Definitions Analysis

Analisis ***Reaching Definitions*** menentukan definisi variabel mana yang mungkin "mencapai" titik tertentu dalam kode tanpa diubah oleh definisi lain di tengah jalan.

9.2.1 Aplikasi

Informasi ini sangat krusial untuk:

- *Constant Propagation* skala global.
- Mendeteksi penggunaan variabel yang mungkin belum diinisialisasi.

9.2.2 Persamaan Aliran Data

Untuk setiap blok B:

$$OUT[B] = GEN[B] \cup (IN[B] - KILL[B])$$

Di mana GEN adalah definisi baru di dalam blok dan $KILL$ adalah definisi lain dari variabel yang sama di blok lain.

9.3 Live Variable Analysis dan DCE

Analisis ***Live Variable*** menentukan apakah sebuah variabel masih "hidup" (*live*), artinya nilainya mungkin akan dibaca lagi di masa depan sebelum program berakhir atau variabel tersebut di-assign ulang.

9.3.1 Analisis Mundur (Backward Analysis)

Analisis ini dimulai dari titik keluar program dan merambat mundur ke atas. Variabel di-mark sebagai *live* jika ia dibaca dalam instruksi saat ini.

9.3.2 Dead Code Elimination (DCE)

Jika hasil dari sebuah operasi (*assignment*) disimpan ke variabel yang sudah tidak lagi *live* (mati), maka instruksi tersebut dapat dihapus dengan aman karena tidak mempengaruhi luaran program.

```
x = 10;      // x live?
y = 20;      // y live?
print(x);    // x dibaca, y mati
// y = 20 dapat dihapus!
```

9.4 Available Expressions dan CSE

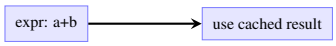
Ekspresi $x + y$ disebut ***Available*** di titik p jika sudah pernah dihitung sebelumnya dan nilai x maupun y belum berubah sejak penghitungan tersebut.

9.4.1 Common Subexpression Elimination (CSE)

Jika sebuah ekspresi sudah *available*, kompilator akan mengganti perhitungan ulang ekspresi tersebut dengan nilai yang sudah ada di variabel perantara.

```
// Sebelum CSE
t1 = a + b
t2 = c * d
t3 = a + b  // Perhitungan berulang

// Sesudah CSE
t1 = a + b
t2 = c * d
t3 = t1
```



Gambar 9.1: Konsep eliminasi sub-ekspresi umum

9.5 Algebraic Simplification dan Strength Reduction

9.5.1 Penyederhanaan Aljabar

Menggunakan identitas matematika untuk menyederhanakan kode:

- $x + 0 \Rightarrow x$
- $x \times 1 \Rightarrow x$
- $x \times 0 \Rightarrow 0$

9.5.2 Strength Reduction

Mengganti operasi yang "mahal" (membutuhkan banyak siklus CPU) dengan operasi yang lebih "murah":

- $x \times 2 \Rightarrow x + x$ atau $x \ll 1$
- $x^2 \Rightarrow x \times x$
- $x/4 \Rightarrow x \gg 2$

Teknik ini sangat efektif terutama di dalam loop yang dieksekusi jutaan kali.

9.6 Pipeline dan Interaksi Optimasi

Optimasi kompilator tidak berjalan sekali saja, melainkan dalam beberapa lintasan (*passes*). Satu optimasi seringkali menjadi pemicu untuk optimasi lainnya.

9.6.1 Optimization Loop

Karena satu perubahan sering memicu peluang baru, optimizer biasanya bekerja dalam sebuah loop iteratif:

```
while (terjadi perubahan):
    lakukan constant folding
    lakukan copy propagation
    lakukan dead code elimination
```

9.6.2 Trade-off

Optimasi yang lebih agresif membutuhkan waktu kompilasi yang lebih lama. Kompilator modern menyediakan level optimasi (seperti `-O1`, `-O2`, `-O3` pada GCC) untuk memberikan fleksibilitas bagi pemrogram.

Aktivitas Pembelajaran

1. **Algebraic Optimization**: Implementasikan berbagai algebraic optimizations.
2. **Constant Propagation**: Bangun constant propagation analyzer.
3. **Copy Propagation**: Implementasikan copy propagation algorithm.
4. **Dead Code Elimination**: Identifikasi dan hapus dead code.
5. **CSE**: Implementasikan common subexpression elimination.

Latihan dan Refleksi

1. Identifikasi semua algebraic optimizations yang mungkin dalam potongan kode!
2. Implementasikan constant propagation untuk program dengan multiple assignments!
3. Analisis copy propagation chains dan handling conflicts!

4. Identifikasi dead code dalam program dengan complex control flow!

5. Implementasikan CSE untuk expressions dengan multiple operands!

6. **Refleksi**: Bagaimana local optimizations mempengaruhi performance compiler?

Asesmen (Evaluasi Kinerja)

Instrumen Penilaian untuk Sub-CPMK 4.3

A. Pilihan Ganda

1. Strength reduction mengganti:
   (a) Operasi mahal dengan operasi murah
   (b) Konstanta dengan variabel
   (c) Variabel dengan konstanta
   (d) Dead code dengan live code

2. Constant propagation:
   (a) Menyebarkan nilai konstanta
   (b) Menghapus konstanta
   (c) Mengidentifikasi konstanta
   (d) Mengoptimasi loops

3. Dead code elimination menghapus:
   (a) Instruksi yang tidak digunakan
   (b) Instruksi yang lambat
   (c) Instruksi yang error
   (d) Semua instruksi

B. Essay

1. Jelaskan implementasi complete local optimization pipeline dengan semua teknik yang dibahas!

2. Analisis trade-off antara berbagai local optimizations dalam terms of performance vs complexity!

Rubrik Penilaian: Lihat Lampiran A

Checklist Pencapaian Kompetensi

Centang item berikut setelah Anda yakin telah menguasainya:

☐ Saya dapat menganalisis dan mengoptimasi kode tingkat lokal

☐ Saya dapat mengimplementasikan algebraic optimizations

☐ Saya dapat melakukan constant propagation

☐ Saya dapat menerapkan copy propagation

☐ Saya dapat mengidentifikasi dan menghapus dead code

☐ Saya dapat melakukan common subexpression elimination

Rangkuman

Bab ini membahas local optimization dan data flow analysis, termasuk algebraic optimizations, constant propagation, copy propagation, dead code elimination, dan common subexpression elimination. Mahasiswa belajar membangun optimization pipeline yang efektif.

Poin Kunci:

- Local optimizations bekerja dalam satu basic block
- Algebraic optimizations menyederhanakan ekspresi matematis
- Constant propagation menyebarkan nilai konstanta
- Copy propagation menghilangkan variabel perantara
- Dead code elimination menghapus instruksi yang tidak berguna
- CSE menghilangkan perhitungan berulang

Kata Kunci: ***Local Optimization***, ***Algebraic Optimization***, ***Constant Propagation***, ***Copy Propagation***, ***Dead Code Elimination***, ***Common Subexpression Elimination***, ***Data Flow Analysis***