Введение в теорию трансляторов (практика 3)

Василий Шамповов, Intel Computer Vision, Intel Corporation

Loop-invariant code motion

- Компиляторная оптимизация, выносящая вычисления, инвариантные к итерациям цикла, за рамки цикла
- Часто используемая аббревиатура - LICM

```
double val = 5.0;
constexpr size_t SIZE = 1000;
double ARR[SIZE];

double tmp = 0;
for (int i = 0; i < SIZE; i++)
{
    tmp = 2 * val;
    ARR[i] = i + tmp;
}
```

LICM →

```
double val = 5.0;
constexpr size_t SIZE = 1000;
double ARR[SIZE];

double tmp = 2 * val;
for (int i = 0; i < SIZE; i++)
{
    ARR[i] = i + tmp;
}
```

Задание

1. Построить LLVM + clang на Linux
2. Скомпилировать каждый из файлов *licm_5.cpp, licm_10.cpp, licm_20.cpp, licm_div_5.cpp, licm_div_10.cpp, licm_div_20.cpp* с помощью clang в двух версиях:
   - -O0 (то есть без оптимизаций)
   - -O1 (то есть с оптимизациями, включающими в себя LICM)
3. Изучить ассемблер и LLVM-IR код скомпилированных программ в предыдущих случаях, отметить существенные отличия и объяснить их
4. Руками в коде LLVM отключить LICM-проход компилятора (редактировать не больше нескольких строк кода) и пересобрать LLVM + clang
5. Снова компилировать файлы из шага 2 с -O1, получая третьи версии файлов. Изучить ассемблер и LLVM-IR код, объяснить отличия от первых двух версий.
6. Для всех трех версий всех скомпилированных программ сделать измерения времени выполнения согласно инструкциям: https://llvm.org/docs/Benchmarking.html. Требуются все метрики, выдаваемыми при выполнении команды perf stat (исполненой через cset shield)
7. Объяснить различия в метриках и наблюдаемые эффекты. Учитывать аппаратную специфику. Полезный ресурс - https://www.agner.org/optimize/instruction_tables.pdf