Независимая от компилятора библиотека точной сборки мусора для языка C++

Моисеенко Евгений
Даниил Березун

СПбГУ
JetBrains Research

5 апреля 2017 г.

Динамическое управление памятью

- Ручное управление памятью
  - Утечки памяти
  - Висячие указатели
  - Повторное освобождение памяти
- Автоматическое управление памятью
  - Подсчёт ссылок
    - Не обрабатываются циклические ссылки
    - Непредсказуемые задержки
  - **Трассирующая сборка мусора**

Мотивация

- Устраняет целый класс возможных ошибок
- Поддержка структур данных с циклическими ссылками
- Упрощает программирование lock-free структур данных
- Упрощает взаимодействие с управляемыми языками

Подходы к сборке мусора в C++

- Умные указатели
  - std::unique_ptr<T>
  - std::shared_ptr<T>
  - std::weak_ptr<T>
- “Прозрачная” (transparent) сборка мусора
  - Boehm GC
- Модификация компилятора
  - C++/CLI

Цели

- **Независимость от компилятора**
- **Точная** сборка мусора
- Возможность совмещать с другими методами управления памятью
- Поддержка объектной модели C++
- Сжатие кучи
- Расширяемость

Трассирующая сборка мусора для C++

```
1     struct Node {
2         int value_;
3         gc_ptr<Node> next_;
4
5         Node(int value, const gc_ptr<Node>& next) {
6             value_ = value;
7             next_  = next;
8         }
9     };
10
11    gc_ptr<Node> create_list(int n) {
12        if (n == 0) {
13            return nullptr;
14        }
15
16        gc_ptr<Node> head = gc_new<Node>(n, nullptr);
17        for (int i = n-1; i > 0; --i) {
18            gc_ptr<Node> next = head;
19            head = gc_new<Node>(i, next);
20        }
21        return head;
22    }
```

Задачи

Задачи

- Поддержание корневого множества

Задачи

- Поддержание корневого множества
  - Динамическая регистрация корней
  - Проверка в конструкторе gc_ptr<T>

Задачи

- Поддержание корневого множества
  - Динамическая регистрация корней
  - Проверка в конструкторе gc_ptr<T>
- Генерация метаинформации

Задачи

- Поддержание корневого множества
  - Динамическая регистрация корней
  - Проверка в конструкторе gc_ptr<T>
- Генерация метаинформации
  - Вычисление смещений gc_ptr внутри объекта
  - Протокол взаимодействия gc_ptr и gc_new

Задачи

- Поддержание корневого множества
  - Динамическая регистрация корней
  - Проверка в конструкторе gc_ptr<T>
- Генерация метаинформации
  - Вычисление смещений gc_ptr внутри объекта
  - Протокол взаимодействия gc_ptr и gc_new
- Собственная реализация кучи

Задачи

- Поддержание корневого множества
  - Динамическая регистрация корней
  - Проверка в конструкторе gc_ptr<T>
- Генерация метаинформации
  - Вычисление смещений gc_ptr внутри объекта
  - Протокол взаимодействия gc_ptr и gc_new
- Собственная реализация кучи
- Реализация остановки мира

Задачи

- Поддержание корневого множества
  - Динамическая регистрация корней
  - Проверка в конструкторе gc_ptr<T>
- Генерация метаинформации
  - Вычисление смещений gc_ptr внутри объекта
  - Протокол взаимодействия gc_ptr и gc_new
- Собственная реализация кучи
- Реализация остановки мира
  - Регистрация управляемых потоков
  - Использование сигналов Linux для инициации сборки
  - gc_unsafe секции кода

Закрепление объектов: мотивация

- ”Сырые” указатели на управляемые объекты
  - this
  - third-party code

Закрепление объектов

- Класс gc_pin<T>

```
1   struct A { ... };
2
3   int f(A* pA);
4
5   int g() {
6       gc_ptr<A> pA  = gc_new<A>();
7       gc_pin<A> pin = pA.pin();
8
9       return f(*pin);
10  }
```

Закрепление объектов

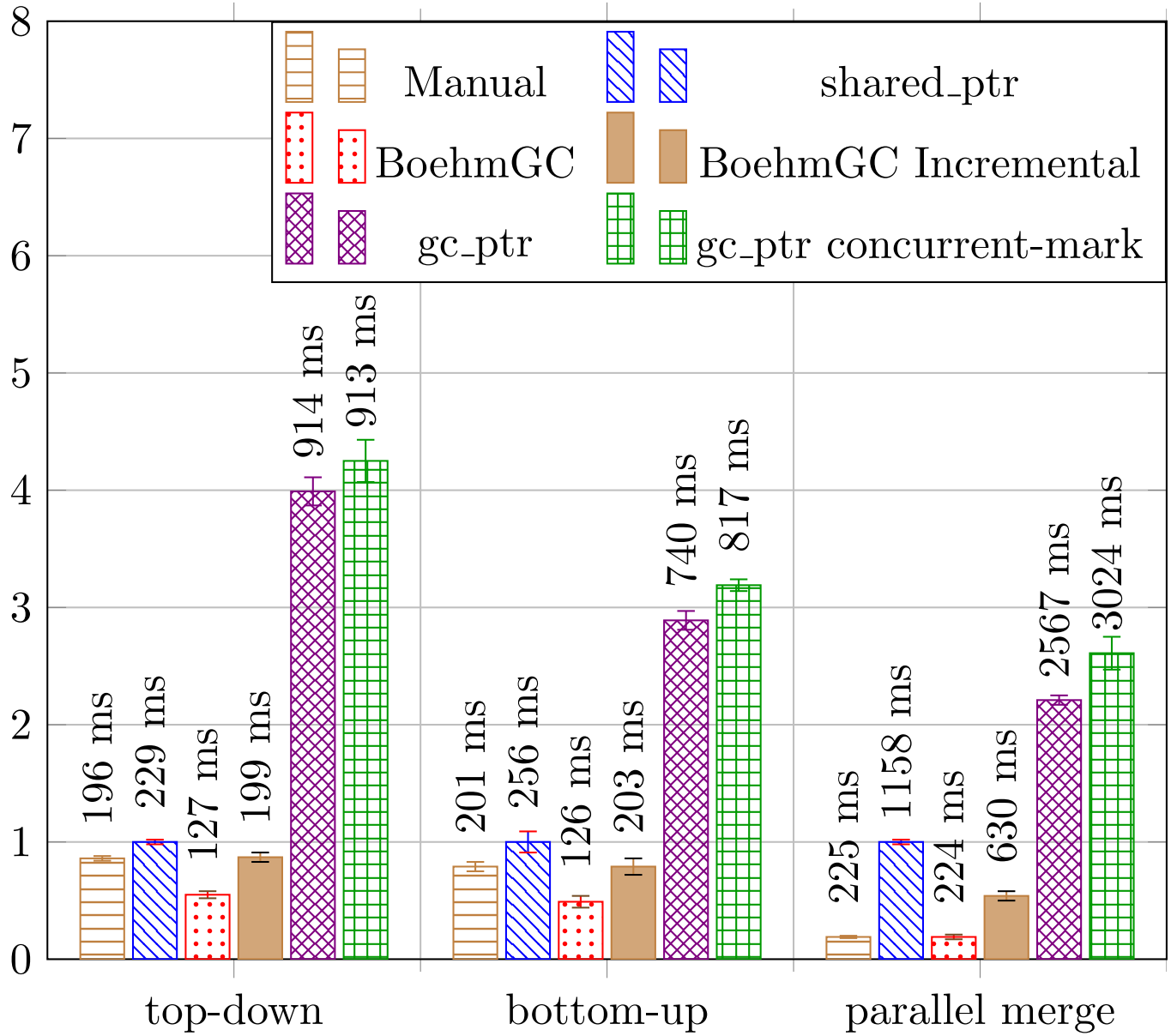
- Класс gc_pin<T>

```
1   struct A {
2       int f();
3   };
4
5   int g() {
6       gc_ptr<A> pA  = gc_new<A>();
7
8       return pA->f();
9   }
```

Дополнительные возможности

- Сжатие кучи при высокой фрагментации
- Параллельная (concurrent) маркировка
- Параллельное (parallel) освобождение/сжатие

Эксперименты



Результаты

- Точная сборка мусора для C++ на уровне библиотеки
- Сборщик мусора может сосуществовать с другими методами управления памятью
- Наш подход следует парадигме ”not pay for what you don’t use”
- Большая часть кода платформо-независимая
- Пользователь должен соблюдать ряд соглашений
- Текущая реализация привносит значительные накладные расходы
- Исходный код: `https://github.com/eucpp/allocgc`