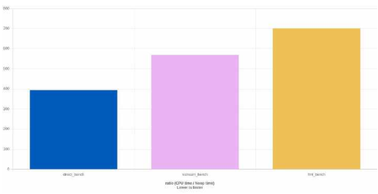
ИССЛЕДУЕМ STD::FORMAT

Александр Романов
младший инженер-программист
в отделе инструментов разработки и компиляторов

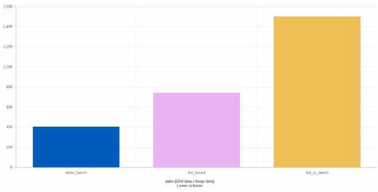
У нового std::format много преимуществ: удобство, безопасность и более высокая производительность, чем у старых способов форматирования строк. Моя жизнь была прекрасна и полна надежд, пока я не увидел один бенчмарк, где format оказался медленнее всех. Как же так? Неужели «устаревший» std::stringstream или даже operator+ все еще работают лучше? В письме расскажу, как исследовал производительность форматирования и поделюсь полученными результатами.

Несколько недель назад мне попался на глаза один интересный бенчмарк. На форматировании строк std::format обогнали все кому не лень:

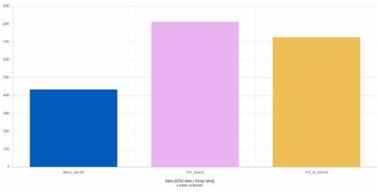


Больше материалов о разработке на «плюсах» ищите в блоге YADRO на Хабре

Однако простое сложение строк все еще доминирует. Почему так? В то время как stringstream обязан учитывать локаль, сложение просто конкатенирует строки, и этот эффект не компенсируется даже лишними выделениями памяти. Со std::format все уже не так просто. Может быть, виновато лишнее выделение памяти? Не беда, используем std::format_to и будем выводить результат в заранее аллоцированную строку. Ой...



Стало еще хуже, но и этот результат объясним. Да, лишние аллокации ушли, но std::back_inserter вызывает std::string::push_back на каждый символ в строке. Эта операция обязана проверять, что мы не вылезаем за границы аллоцированной памяти. Такая проверка замедляет форматирование в разы сильнее, чем аллокации. Исправив наш пример,мы получим намного более приемлемый результат:



На quickbench это сделать нельзя, буду запускать локально. Приведу результаты сборки компилятором gcc 14.3.0 с оптимизацией -O3, запускал на i5-1235U.

```
Маленькие (size() <= 15) строки
stream_bench: 323 ns
format_bench: 233 ns
fmt_format_bench: 165 ns
add_bench: 132 ns

Большие (size() >= 33) строки
stream_bench: 502 ns
format_bench: 401 ns
fmt_format_bench: 376 ns
add_bench: 234 ns
```

Видим, что fmt::format работает стабильно быстрее стандартного аналога, но все еще проигрывает сложению. На этом этапе я готов отдать победу оператору сложения строк. Однако все мои замеры относились исключительно к конкатенации строк (задаче, для которой был создан operator+) и не включали форматирование чисел и выравнивание вывода. Если же мы начнем форматировать double-числа, то результаты сильно изменятся:

```
format_bench: 346 ns
fmt_format_bench: 183 ns
add_bench: 249 ns
```

std::format все еще проигрывает сложению, но fmt::format вырывается на уверенное первое место. Отмечу, что я использовал fmt 10.2.1, а в последней версии 12.0.0 форматирование чисел с плавающей точкой ускорили еще на 60%.

```
format_bench: 346 ns
fmt_format_bench: 183 ns
add_bench: 249 ns
```

Разумеется, все приведенные замеры — не безусловная истина, а лишь промежуточные результаты. Но я попробую сделать выводы:

- Конкатенируйте строки при помощи сложения. Форматируйте произвольные типы при помощи format.
- К моему большому сожалению, std::format отстает от fmt::format как в скорости, так и в функциональности.
- Аккуратно используйте format_to в местах, где важна производительность, и проверяйте, какой эффект он оказал.