Optymalizacja kodu

vs. Intuicja (średnio) zaawansowanego programisty

Po pierwsze

- Intuicja **nie** działa
- Najpierw pomiar. *Zawsze!*

Po drugie

- (średnio) zaawansowani programiści lubią mikro-optymalizację
- Jeśli chce się *bawić* w mikro-optymalizacje,
  trzeba umieć **czytać *asemblera***

- mov eax, ebx -> eax = ebx
- mov eax, [rdi] -> eax =*rdi
- mov eax, [rdi+rsi*4] -> eax = rdi[rsi*4]

Co nie tak z tą intuicją?

- Np.: „*if* to skok, skok jest wolny, pousuwam *if'y*”
- *godbolt.org*

Co nie tak z tą intuicją?

- „każdy obiekt to narzut”
- „lepiej używać ‚gołych’ wskaźników”
- *Kod*

- *Ale naprawdę „lepiej nie nadużywać shared_ptr”: Kod*

Co nie tak z tą intuicją?

- „każdy obiekt to narzut”
- „wywalę wektory, tablice będą szybsze”
- *Kod*

- *Kod*
- int[][] != vector<vector<int>>

Co nie tak z tą intuicją?

- „wiem co tu kompilator zrobi”
- *x * 2*
- *x * 65553*

- *Choć historia jest prawdziwa: -march=i486 -m32*

Co nie tak z tą intuicją?

- „a, kompilator jakoś sobie z tym poradzi”
- *const*

- *więcej:*
  CppCon 2017: Matt Godbolt
  “What Has My Compiler Done for Me Lately?
  Unbolting the Compiler's Lid”

Co dalej?

- Nie popadać w nadmierny pesymizm…
- …ale i optymizm
- Najpierw pomiar. *Zawsze!*

Wnioski

- Warto używać gotowego kodu
- Warto ufać kompilatorowi
- Warto umieć *czytać* asemblera
  (patrz „zasada ograniczonego zaufania”)
- Trzeba znać architekturę na którą się mikro-optymalizuje
- Najpierw *ładny/dobry* kod, potem (ewentualna) optymalizacja
- Dobre mieć testy funkcjonalne (np. jednostkowe)
- Pomiar pierwszy!

Gdzie intuicja może pomagać

- Operacje I/O
- new
- mutex
- wywołania systemowe (w tym std::cout)
- wielopiętrowe wskazania (a->b->c)
  - w tym ukryte w strukturach (std::vector<std::vector<…>>)

Wskazówki (dla C++ i nie tylko)

- Pokochać *const*
- Polubić operacje „na wartościach” i „lokalność danych”
- Nie nadużywać *std::shared_ptr*
- Optymalizować kod tam gdzie to potrzebne
- Ale też nie robić kodu „nie optymalnego”
- Mierzyć, mierzyć oraz mierzyć

Dziękuję za uwagę!