



MINISTÉRIO DA
CIÊNCIA, TECNOLOGIA
E INOVAÇÃO



Profiling e otimização em códigos C/C++

Parte 2

Professor: Hiago Mayk G. de A. Rocha
E-mail: mayk@lncc.br

Agenda

- Introdução
- gcov
- gprof
- valgrind
- perf

Agenda

- **Introdução**
- gcov
- gprof
- valgrind
- perf

O que é *Profiling*

- *Profiling* é o processo de analisar o comportamento de execução (*runtime*) de um programa para entender o seu comportamento:
 - Quantas vezes cada função foi chamada?
 - Quanto tempo cada função consome?
 - Qual função chamou qual?
 - Como está o consumo de memória?

Por que *profiling* é essencial em otimização?

- Sem *profiling*, otimização é adivinhação.
- *Profiling* permite:
 - Decisões baseadas em dados
 - Validação de hipóteses
 - Comparação antes/depois de otimizações

Tipos de *profiling*

- Tipos diferentes de *profiling* respondem perguntas diferentes:
 - **Code Coverage** → O que realmente executa?
 - **Flat Profile** → Onde o tempo é gasto?
 - **Call Graph** → Como o tempo é gasto?

Tipos de *profiling*

- **Code Coverage:**

- Mede quantas vezes cada linha de código foi executada:
 - Linhas nunca executadas
 - Linhas executadas poucas vezes
 - Linhas críticas (executadas muitas vezes)
- Serve para:
 - Identificar código morto
 - Localizar regiões críticas para otimização

“Cobertura ajuda a focar otimização onde realmente executa.”

Tipos de *profiling*

- ***Flat Profile:***

- Mede quanto tempo o programa gasta em cada função, de forma independente:
 - Tempo total por função
 - Percentual do tempo total
 - Número de chamadas
- Serve para:
 - Identificar *hot spots* (funções mais custosas)
 - Decidir onde otimizar primeiro
 - Evitar otimizar código que quase não executa

“90% do tempo está em 2 funções.”

Tipos de *profiling*

- ***Call Graph:***

- Mede qual função chamou qual função, e quantas vezes, formando uma hierarquia:
 - Relação entre funções
 - Caminhos de execução
 - Frequência de chamadas
- Serve para:
 - Entender estrutura do programa
 - Detectar chamadas desnecessárias
 - Identificar funções simples chamadas milhões de vezes

“Uma função barata é chamada bilhões de vezes e se torna cara.”

Ferramentas disponíveis

- Existem diversas ferramentas de *profiling* amplamente utilizadas em HPC:
 - **Análise de desempenho:**
 - perf, Intel VTune Profiler, HPCToolkit, Score-P
 - **Paralelismo e comunicação:**
 - Intel Trace Analyzer and Collector (ITAC), TAU Performance System
 - **Memória e acesso a dados:**
 - Valgrind (Cachegrind), Massif
 - **Visualização e análise pós-execução:**
 - Paraver, Vampir
 - **Aceleradores (GPUs):**
 - Nsight Systems / Nsight Compute, rocprof / rocTracer

Agenda

- Introdução
- **gcov**
- gprof
- valgrind
- perf

gcov

- O gcov é a ferramenta de análise de cobertura de código do GNU Compiler Collection (GCC).
- gcov não mede tempo, ele mede frequência de execução:
 - Linhas executadas milhões de vezes → candidatas a otimização
 - Linhas raramente executadas → baixo retorno otimizar

gcov: flags do GCC

- **-fprofile-arcs:**
 - Ativa a coleta de dados de fluxo de controle:
 - Quantas vezes cada *branch* foi tomado
 - Quantas vezes *loops* iteraram
 - Quantos saltos ocorreram
- **-ftest-coverage:**
 - Ativa a instrumentação para cobertura de código por linha.

```
gcc -fprofile-arcs -ftest-coverage matmul.c
```

Agenda

- Introdução
- gcov
- **gprof**
- valgrind
- perf

gprof

- O gprof é a ferramenta de profiling do GNU Compiler Collection (GCC).
- O funcionamento do gprof tem duas partes:
 - Instrumentação de funções:
 - O compilador insere chamadas automáticas
 - Conta quantas vezes cada função é chamada
 - Amostragem de tempo:
 - O sistema coleta amostras do contador de programa (PC)
 - Estima quanto tempo foi gasto em cada função

Agenda

- Introdução
- gcov
- gprof
- **valgrind**
- perf

Valgrind

- O Valgrind é uma ferramenta de análise dinâmica de programas.
- Ele executa o seu programa dentro de uma máquina virtual e observa:
 - Uso de memória
 - Acessos inválidos
 - Vazamentos (*leaks*)
 - Uso de valores não inicializados
- Não é apenas um *profiler*:
 - Verifica correção
 - Simula comportamento
 - Rastreia uso real de recursos

Valgrind

- **Execução muito mais lenta:**
 - Cada instrução do programa é analisada
 - Cada acesso à memória é validado
 - O programa não executa direto no *hardware*
- **Algumas ferramentas internas:**
 - **memcheck:** rastreia memória alocada dinamicamente, *leaks*, consumo
 - **cachegrind:** simula o uso da cache (*hit* e *miss*)
 - **callgrind:** gera o grafo de chamadas (podemos usar o kcachegrind para visualização)

Agenda

- Introdução
- gcov
- gprof
- valgrind
- **perf**

Linux Perf

- Perf é uma ferramenta do Linux que mede eventos de *hardware* enquanto o programa executa.
- Ele usa os *Performance Monitoring Units* (PMUs) do processador:
 - Ciclos de CPU
 - Instruções executadas
 - Cache misses (L1, L2, LLC)
 - Branches errados
 - *Page faults*
 - *Context switches*
- Não é estimativa, são contadores reais do *hardware*

Referências e materiais adicionais

- PROGRAMA DE VERÃO SD02-II. 29/01/25 – Profiling e otimização em códigos C/C++ (14h–18h). YouTube, 2025. Disponível em: <https://www.youtube.com/live/OTAUEx197CE> Acesso em: 19 jan. 2026.
- BORIN, Edson. LNCC14 – Material da disciplina. Universidade Estadual de Campinas (UNICAMP). Disponível em: <https://www.ic.unicamp.br/~edson/disciplinas/lnc14/>. Acesso em: 19 jan. 2026.