

# Coisas para saber antes de fazer o seu próprio Benchmarks Game

---

Alfredo Goldman - IME - USP

Elisa Uhura - IME - USP

Sarita Mazzini Bruschi - ICMC - USP

# Agenda

Motivação

Conceitos importantes

Ferramentas

Desenvolvimento

Parte 1

Parte 2

Conclusão

# Motivação

O que fazer para melhorar desempenho das aplicações?

- Paralelizar

Mas antes de paralelizar...

- Avaliar o desempenho
- Comparar os resultados

*Computer Language Benchmarks Game*

# Conceitos Importantes

O que é desempenho?

- Quantidade de trabalho útil realizado por um computador
- Deve ser mensurável

Pode ser

- Absoluto: tempo de execução, latência, vazão (*throughput*), consumo energético
- Relativo: composição de várias métricas (*Passmark rating*)

# Conceitos importantes

Variação de fatores que influenciam o desempenho

*Benchmark:*

- Programa (ou conjunto de programas) desenvolvidos especialmente para medir o desempenho

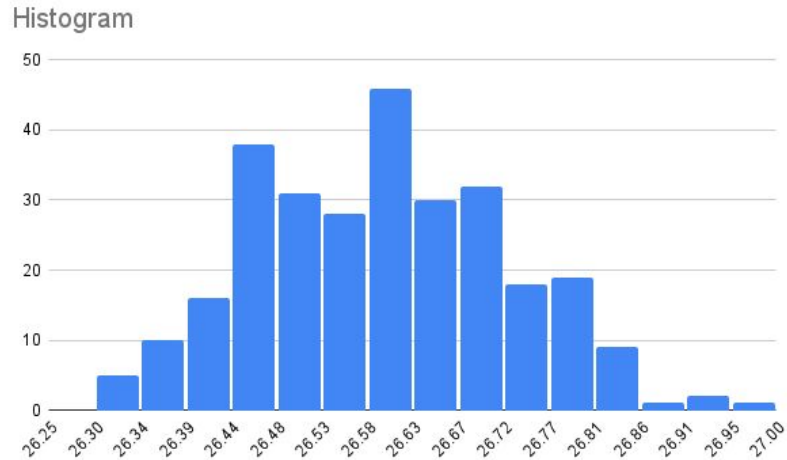
Fatores que influenciam o desempenho:

- Hardware: arquitetura (ISA, velocidade), hierarquia de memória
- Software: estrutura do código, compilador/interpretador, sistema operacional

# Conceitos Importantes

## Análise

- Precisão dos resultados: média, desvio padrão, intervalo de confiança
- Importante verificar se os dados seguem um padrão de distribuição



# Ferramentas

Time:

real: tempo total entre o início e o fim da execução

user: tempo gasto pelas instruções do programa

sys: tempo usado nas chamadas ao sistema (*kernel*)

Vantagem: não é invasivo

# Ferramentas

## Perf

```
1 Performance counter stats for './binary-trees 15':
2
3           268,19 msec task-clock          #    0,960 CPUs utilized
4           30      context-switches      #    0,112 K/sec
5           0       cpu-migrations        #    0,000 K/sec
6           731     page-faults           #    0,003 M/sec
7 <not supported>    cycles
8 <not supported>    instructions
9 <not supported>    branches
10 <not supported>   branch-misses
11
12           0,279237198 seconds time elapsed
13
14           0,264584000 seconds user
15           0,003949000 seconds sys|
```



# Ferramentas

## Intel VTune Profiler

The screenshot displays the Intel VTune Profiler interface. On the left is the Project Navigator showing a tree view of a 'sample (matrix)' project with sub-items r00hs, r001ue, r002ps, r003ue, r004ue, r005ue, r006ue, and r007ue. The main window is titled 'Microarchitecture Exploration' and shows a list of performance metrics for the selected item 'r001ue'. The 'Elapsed Time' is 38.004s. The metrics table is as follows:

Metric	Value	Unit
Clockticks	947,323,000,000	
Instructions Retired	318,851,000,000	
CPI Rate	2.971	of Pipeline Slots
MUX Reliability	0.990	
Retiring	16.2%	of Pipeline Slots
Front-End Bound	2.5%	of Pipeline Slots
Bad Speculation	0.1%	of Pipeline Slots
Back-End Bound	81.2%	of Pipeline Slots
Memory Bound	72.8%	of Pipeline Slots
L1 Bound	4.0%	of Clockticks
L2 Bound	2.0%	of Clockticks
L3 Bound	9.6%	of Clockticks
Contested Accesses	0.0%	of Clockticks
Data Sharing	2.0%	of Clockticks
L3 Latency	2.0%	of Clockticks
SQ Full	0.0%	of Clockticks
DRAM Bound	65.5%	of Clockticks
Memory Bandwidth	47.1%	of Clockticks
Memory Latency	43.2%	of Clockticks
Store Bound	0.0%	of Clockticks
Core Bound	8.4%	of Pipeline Slots
Divider	0.0%	of Clockticks
Port Utilization	9.4%	of Clockticks
Cycles of 0 Ports Utilized	66.6%	of Clockticks
Cycles of 1 Port Utilized	12.9%	of Clockticks
Cycles of 2 Ports Utilized	10.4%	of Clockticks
Cycles of 3+ Ports Utilized	9.1%	of Clockticks
Vector Capacity Usage (FPU)	25.0%	
Average CPU Frequency	3.2 GHz	
Total Thread Count	10	
Paused Time	0s	

Below the metrics is a diagram labeled 'µPipe' representing pipeline inefficiencies. It shows a bar chart with three segments: a large orange segment for '72.8% - Memory Bound', a green segment for '16.2% - Retiring', and a small grey segment for '8.4% - Core Bound'. A text box to the right explains: 'The metric value is high. This can indicate that the significant fraction of execution pipeline slots could be stalled due to demand memory load and stores. Use Memory Access analysis to have the metric breakdown by memory hierarchy, memory bandwidth information, correlation by...'. Below the diagram, another text box states: 'This diagram represents inefficiencies in CPU usage. Treat it as a pipe with an output flow equal to the "pipe efficiency" ratio: (Actual Instructions Retired)/(Maximum Possible Instruction Retired). If there are pipeline stalls decreasing the pipe efficiency, the pipe shape gets more narrow.'

# Parte 1

Benchmark: binary-trees

Objetivo:

- Comparar o tempo de execução nas linguagens C, Go e Python

Métrica:

- Tempo de execução (podem ser outras)

Tabulação dos dados

Análise dos dados

# Parte 2

Benchmark: Mandelbrot

Objetivos:

- Relógios do sistema
- Flags de compilação/execução
- Compiladores e Interpretadores alternativos
- Uso de compilação *just-in-time* (JIT)
- Profiling
- Soma de verificação
- Diferença de desempenho entre linguagens

# Código

Todos os códigos está disponíveis em:

<https://github.com/elisauhura/ERAD-benchmarks>