

SuperComputação

Aula 04 – Profiling

2021 – Engenharia

André Filipe de Moraes Batista <andrefmb@insper.edu.br>

Solução de alto desempenho

1. Algoritmos eficientes

2. Implementação eficiente

- Cache, paralelismo de instrução
- Linguagem de programação adequada

3. Paralelismo

Medição de desempenho

Nossa otimização não funcionou, por que?

- Medir quanto tempo cada função demora?
- Nossa função ficou mais rápida? Se sim, quanto? Se não, por que?
- **Como medir "quantidade de trabalho feito"?**

Medição de tempo

Profiling

"Análise de um programa durante sua execução para determinar seu consumo de memória e/ou tempo."

Podemos responder duas importantes perguntas:

- onde o programa consome mais recursos?
- onde devo concentrar meus esforços de otimização?

HPC Linux Engineer

at Jump Trading ([View all jobs](#))

Chicago, New York, Singapore, London

This rare hands-on individual will be highly skilled in the details and nuances of managing Linux environments with a strong software development background necessary to support uniquely customized systems.

What you'll do:

- Assist in designing, implementing, and maintaining high performance file storage systems.
- Implement and support performance monitoring and fault monitoring systems
- Monitors systems and storage performance, up to and including network components
- Compiles, packages, installs and upgrades software and operating system components
- Creates scripts and uses tools to automates frequently performed tasks
- Develops, improves, documents and promotes standard operating procedures
- Develop and monitor the tools used to maintain a production computing environment
- Other duties as assigned or needed

What you'll need:

- High proficiency with at least one of the following languages: Python, Ruby, Go, C
- Extensive experience profiling and debugging application stacks
- Extensive experience ~~designing~~ building, and maintaining complicated, interdependent, and distributed systems
- Experience with system configuration management tools (Cfengine, Salt, Puppet, Ansible, etc.)
- Experience in an academic research computing (HPC) background
- A compulsion to perform root cause analysis
- Reliable and predictable availability



Atividade prática

Seção "Preparação" (5 minutos)

1. Preparar ambiente para acompanhar uma sessão de profiling com o professor.

KCachegrind

The screenshot displays the KCachegrind application window titled "callgrind.out.7030 [./euclid] — KCachegrind". The interface includes a menu bar (File, View, Go, Settings, Help) and a toolbar with icons for Open, Back, Forward, Up, Relative, Cycle Detection, Relative to Parent, Shorten Templates, and Instruction Fetch.

Flat Profile:

Self	Called	Function	Location
.00	0.00	(0)	ld-2.27.so
.96	0.00	_start	euclid
.96	0.00	(below main)	libc-2.27.so: libc-start.c
.74	0.43	main	euclid: euclid.cpp, std_vector.h,
.34	1.44	std::ostream& std::ostream::<...>	libstdc++.so.6.0.25
.11	0.11	std::num_put<char, std::ostre...	libstdc++.so.6.0.25
.98	3.53	std::ostreambuf_iterator<char...	libstdc++.so.6.0.25
.73	1.47	0x0000000000d9c40	libstdc++.so.6.0.25
.09	1.39	vsnprintf	libc-2.27.so: vsnprintf.c
.92	7.39	vfprintf	libc-2.27.so: vfprintf.c, printf-pa
.31	0.17	__printf_fp	libc-2.27.so: printf_fp.c
.15	26.45	__printf_fp_l	libc-2.27.so: printf_fp.c, localeir
.69	6.55	hack_digit	libc-2.27.so: printf_fp.c
.08	0.33	__gnu_cxx::stdio_sync_filebuf<...	libstdc++.so.6.0.25
.74	4.13	fwrite	libc-2.27.so: iofwrite.c, libioP.h
.53	7.53	__mpn_divrem	libc-2.27.so: divrem.c
.33	1.48	std::basic_ostream<char, std::>	libstdc++.so.6.0.25

Source Code:

```

# Ir Co Source
219 0.00 { return _M_extract(__f); }
    2.94 240 call(s) to 'std::istream& std::istream::_M_extract<double>(double&) (libstdc++...
941 0.00 if (this->_M_impl._M_finish != this->_M_impl._M_end_of_storage)
41 0.00 for (int i = 0; i < n; i++) {
943 0.00     _Alloc_traits::construct(this->_M_impl, this->_M_impl._M_finish,
945 0.00         ++this->_M_impl._M_finish;
136 0.00     { ::new((void *)__p)_Up(std::forward<Args>(_args)...); }
32 0.00     for (int i = 0; i < n; i++) {
948 0.00         _M_realloc_insert(end(), __x);
    0.01 16 call(s) to 'void std::vector<double, std::allocator<double>>::_M_realloc_insert<d...
25 0.00 int main() {
  
```

Profile Part:

Profile Part	Incl.	Self	Called	Comment
--------------	-------	------	--------	---------

Parts: Callee, Call Graph, All Callees, Caller Map, Machine Code

At the bottom, a status bar indicates: "callgrind.out.7030 [1] - Total Instruction Fetch Cost: 51 844 483".

Demonstração



Atividade prática

Profiling na prática

1. Usar o KCachegrind para analisar nossa tentativa de otimização
2. Fazer novas otimizações e medir seu desempenho

Fechamento

Pontos importantes:

- Entrada/Saída custa caro.
- Implementações diferentes do mesmo algoritmo podem ter desempenho diferente
- Detalhes finos só são visíveis com ferramentas de profiling

Insper

www.insper.edu.br