

Profiling: técnicas e ferramentas

Profa. Andrea Charão
DLSC/CT/UFSM

Profiling

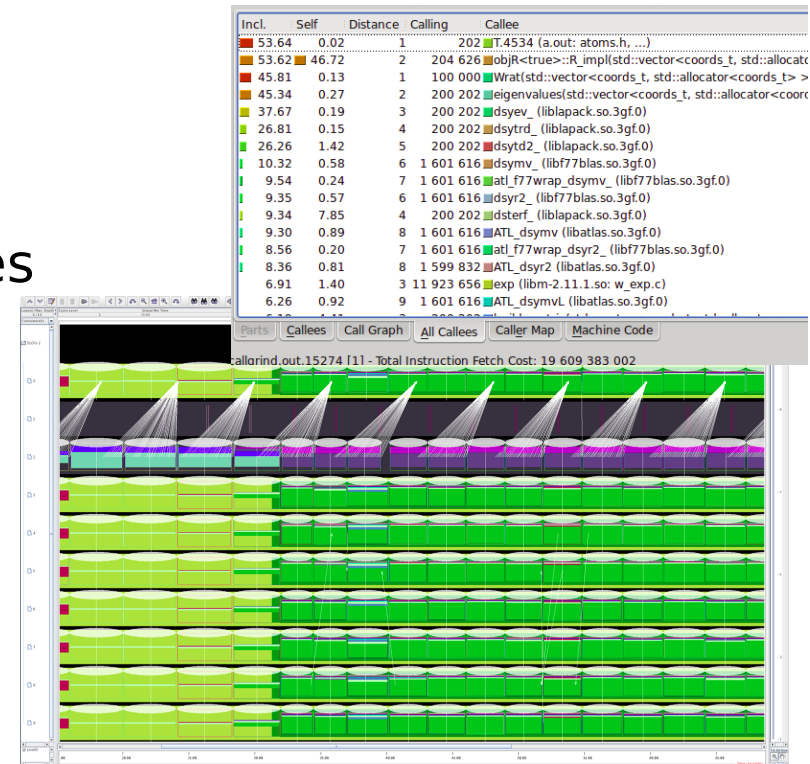
- Extração do perfil da execução de um programa

- Pode incluir, por exemplo:

- ◆ Tempo gasto na execução do programa inteiro e de cada subprograma
- ◆ Número de chamadas de funções ou métodos
- ◆ Grafo de chamadas
- ◆ Diagrama espaço-tempo (Gantt chart)
- ◆ Etc.

Each sample counts as 0.01 seconds.

%	cumulative	self	calls	self	total	name
time	seconds	seconds		ms/call	ms/call	
52.39	0.67	0.67	160014	0.00	0.00	broadcast_msg(char*)
17.99	0.90	0.23	48	4.80	9.38	catch_up_msg(int, int)
17.20	1.12	0.22	4815505	0.00	0.00	send_msg(char*, int)
7.04	1.21	0.09	10354	0.01	0.12	exec_cmds(SDL_Surface*)
0.78	1.22	0.01	160062	0.00	0.00	parsemsg(char*)
0.78	1.23	0.01	160014	0.00	0.00	formatandaddtomsgbuff(char*)
0.78	1.24	0.01	160014	0.00	0.00	draw(int, int, int, int)
0.78	1.25	0.01	10354	0.00	0.00	parse_cmds()
0.78	1.26	0.01	10354	0.00	0.00	net_tick()



Como funciona o profiling

Técnica: Amostragem

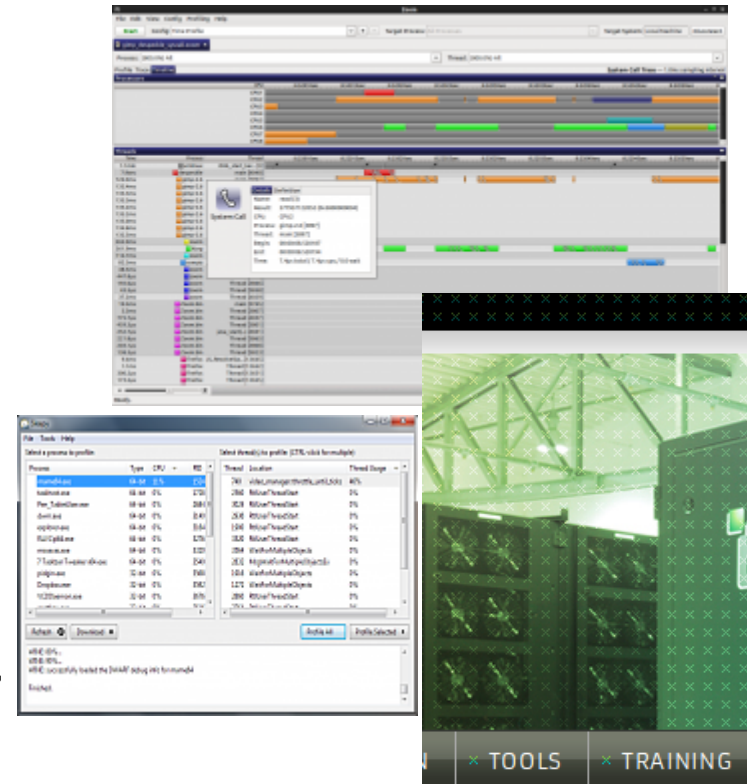
- interrupção periódica do programa para coleta de dados sobre a execução
- tende a ser pouco intrusiva
- menos precisa

Técnica: Instrumentação

- adição de instruções no programa para coleta de dados sobre a execução
- pode ser muito intrusiva
- mais precisa

Profilers

- Dependência (SO, compilador...)
- Análise “post-mortem” ou durante execução
- Suporte (ou não) a concorrência/paralelismo
- Exemplos:
 - Gprof, Kprof, Zoom, VerySleepy, Intel Vtune, etc.
 - Ferramentas avançadas: TAU, Score-P, PAPI, etc.



SCORE-P

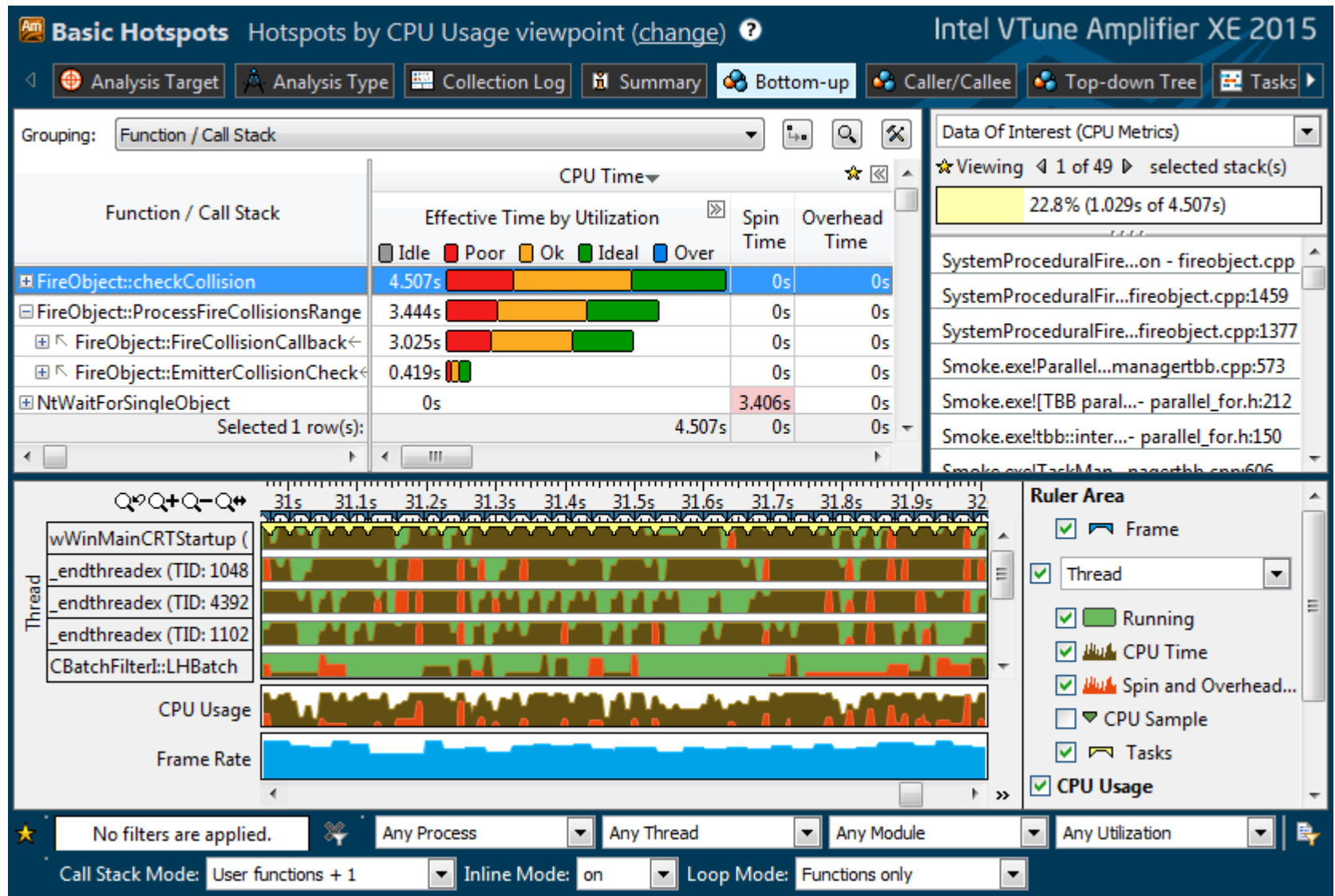
Scalable Performance Codes

Exemplo: Gprof

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
52.39	0.67	0.67	160014	0.00	0.00	broadcast_msg(char*)
17.99	0.90	0.23	48	4.80	9.38	catch_up_msg(int, int)
17.20	1.12	0.22	4815505	0.00	0.00	send_msg(char*, int)
7.04	1.21	0.09	10354	0.01	0.12	exec_cmds(SDL_Surface*)
0.78	1.22	0.01	160062	0.00	0.00	parsemsg(char*)
0.78	1.23	0.01	160014	0.00	0.00	formatandaddtomsgbuff(char*)
0.78	1.24	0.01	160014	0.00	0.00	draw(int, int, int, int, int)
0.78	1.25	0.01	10354	0.00	0.00	parse_cmds()
0.78	1.26	0.01	10354	0.00	0.00	net_tick()

Exemplo: Intel Vtune Amplifier



Exemplo: JProfiler

Biggest Objects References Time Inspections Graph

Object groups:

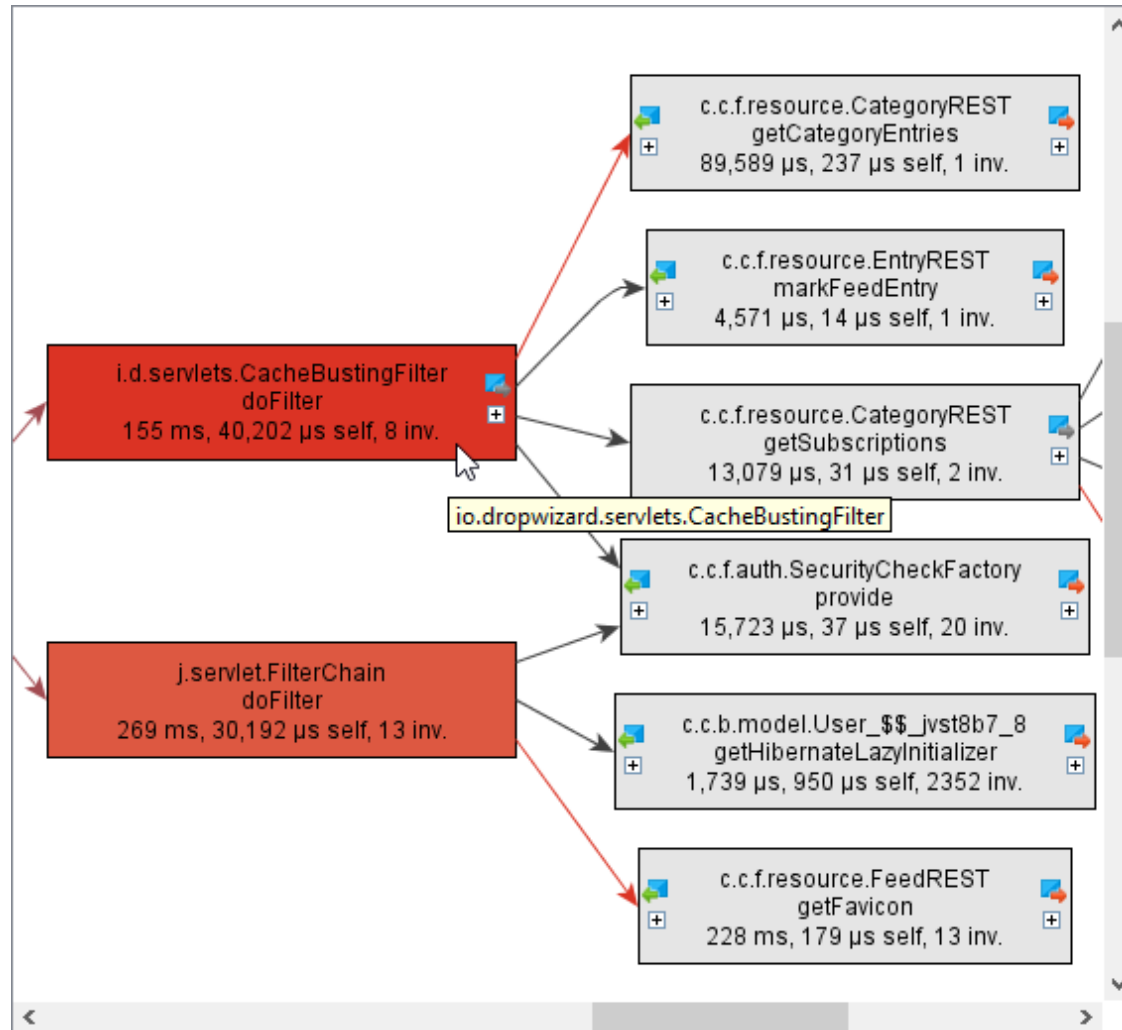
Priority	ThreadLocal fields	Instance Count ▾	Shallow Size	
10	field threadLocalAutoAnalyzerNameCache of org.jvnet.hk2.i...	6	288 bytes	^
12	static field decoder of java.lang.StringCoding	6	240 bytes	
17	static field dateFormats of org.glassfish.jersey.message.inte...	6	144 bytes	
13	static field _recyclerRef of com.fasterxml.jackson.core.JsonF...	5	200 bytes	
18	field cache of sun.nio.cs.ThreadLocalCoders\$1	4	128 bytes	
19	field cache of sun.nio.cs.ThreadLocalCoders\$2	4	128 bytes	
20	static field _threadEncoder of com.fasterxml.jackson.core.io....	3	120 bytes	
22	field threadLocalAnnotationCache of org.jvnet.hk2.internal.P...	1	48 bytes	
23	static field tba of com.sun.xml.internal.stream.util.ThreadLo...	1	40 bytes	▾

Current object set: 5 instances of java.lang.ref.SoftReference
3 selection steps, 200 bytes shallow size, [Calculate retained and deep sizes](#) [Use retained obje](#)

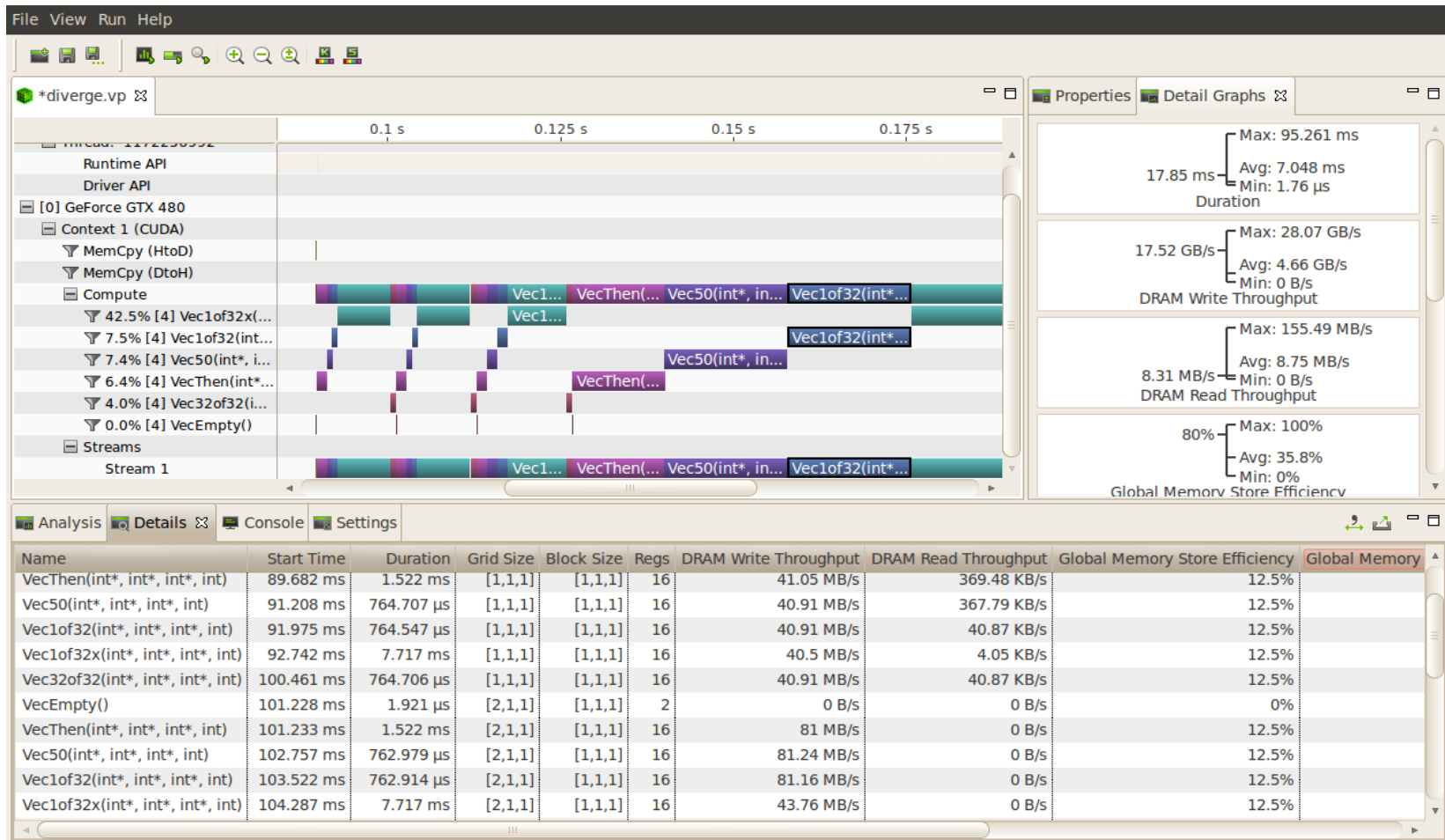
Outgoing references ▾ Use ... ▾ Apply filter ... ▾

	Object	Retained size	Shallow Size	Allocation Time (h:m:s)
[-]	java.lang.ref.SoftRefe... timestamp = 332811159 + queue (declared by java.lang.ref.Reference) → java.lang.ref.ReferenceQueue\$Null (0x39e3) + referent (declared by java.lang.ref.Reference) → com.fasterxml.jackson.core.util.BufferRecycler (0x61372)	16,160 bytes	40 bytes	n/a
[+]	java.lang.ref.SoftRefe...	16,160 bytes	40 bytes	n/a
[+]	java.lang.ref.SoftRefe...	16,160 bytes	40 bytes	n/a
[+]	java.lang.ref.SoftRefe...	16,160 bytes	40 bytes	n/a
[+]	java.lang.ref.SoftRefe...	16,160 bytes	40 bytes	n/a

Exemplo: JProfiler



Exemplo: NVIDIA Visual Profiler



Considerações finais

- Ferramentas avançadas de suporte ao desenvolvimento + otimização
- Um programa, diferentes profilers
 - Resultados podem variar

The screenshot shows the ACM Digital Library interface. At the top, the logo for ACM Digital Library is on the left, and 'Universidade Federal de Santa Maria (UFSM)' is on the right. A search bar and links for 'SIGN IN' and 'SIGN UP' are also present. The main content area displays the article 'Evaluating the accuracy of Java profilers' by Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. It includes a PDF icon, a '2010 Article' label, and a 'Bibliometrics' section showing download statistics. The 'Published in' section lists the proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation. A right sidebar contains 'Tools and Resources' such as 'Request Permissions', 'TOC Service', 'Export Formats', and 'Upcoming Conference'. At the bottom, there are links for 'Recent authors with related interests' and 'Concepts in this article', along with a 'powered by IBM Watson' logo.