

Profiling

Dr.-Ing. Volkmar Sieh

Department Informatik 4
Verteilte Systeme und Betriebssysteme
Friedrich-Alexander-Universität Erlangen-Nürnberg

WS 2019/2020



Profiling unabdingbar, da höchstmögliche Performance erwünscht.

Aber: normales Profiling schwierig:

- Profiling-Option verhindert i.A. Compiler-Optimierungen (z.B. Inlining)
- Compiler-Optionen für die Code-Generierung u.U. unabdingbar (z.B. für die Generierung der JIT-Code-Schnipsel)
- für das Profiling eingefügter Code verfälscht ggf. das Profiling-Ergebnis
- Profiling von JIT-generiertem Code generell unmöglich

normales Profiling unmöglich => eigenes Profiling notwendig



Zeitmessungen:

`gettimeofday`: POSIX-System-Call

Genauigkeit: ca. 1 Mikrosekunde
(aber: System-Call, jeder Aufruf teuer)

`rdtsc`: CPU-Instruktion

Genauigkeit: ca. 1 Takt
(aber: Out-of-Order-Execution, variabler Takt, siehe:
„How to Benchmark Code Execution Times on Intel
IA-32 and IA-64 Instruction Set Architectures”)

Vorsicht: VM wird ggf. (mehrfach) verdrängt!



Event-Zählungen:

Beispiele

- Anzahl der ausgeführten Instruktionen
- Anzahl der ausgeführten Basisblöcke
- ...



Statistisches Profiling:

zu profilendes Programm setzt Signal-Handler für SIGPROF-Signal:

```
struct sig_action sa;  
  
sa.sa_sigaction = sigprof_handler;  
sa.sa_flags = SA_RESTART | SA_SIGINFO;  
sigemptyset(&sa.sa_mask);  
sigaction(SIGPROF, &sa, NULL);
```



... und startet selbst Profiling-Timer

```
struct itimerval it;

it.it_interval.tv_sec = 0;
it.it_interval.tv_usec = 1000000 / 31;
it.it_value.tv_sec = 0;
it.it_value.tv_usec = 1000000 / 31;

setitimer(ITIMER_PROF, &it, NULL);
```



... und kann dann im periodisch aufgerufenen Signal-Handler nachschauen, wohin der `%rip` der VM gerade zeigt (Beispiel für Linux/x86_64):

```
void
sigprof_handler(int sig, siginfo_t *si, void *uc)
{
    mcontext_t *regs = &uc->uc_mcontext;
    int enosave = errno;

    fprintf(stderr, "PROFILE_□%016lx\n",
            regs->gregs[REG_RIP]);

    errno = enosave;
}
```



Nach einem (längeren) Lauf der VM kann man die statistischen Profile-Daten in das Disassembler-Listing der VM eintragen. Beispiel:

```
1840      0040819d <chip_intel_80686_klamath_cache2_line>
    ...
    24/ 1% 004081f9: ... movslq %edx,%rcx
    8/ 0% 004081fc: ... imul  $0x50,%rcx,%rax
    31/ 1% 00408200: ... lea   0xedee90(%rax,%r8,1),%rax
    30/ 1% 00408208: ... add   %rdi,%rax
    40/ 2% 0040820b: ... cmp   %rsi,(%rax)
1129/61% 0040820e: ... jne   408225 ...
    4/ 0% 00408210: ... lea   0xedefd0(%rdi,%r8,1),%rdx
    69/ 3% 00408218: ... movzbl (%rdx),%esi
280/15% 0040821b: ... mov   0x7fea00(%rcx,%rsi,4),%cl
    32/ 1% 00408222: ... mov   %cl,(%rdx)
    20/ 1% 00408224: ... retq
    20/ 1% 00408225: ... inc   %edx
    57/ 3% 00408227: ... cmp   $0x4,%edx
    3/ 0% 0040822a: ... jne   4081f9 ...
    0/ 0% 0040822c: ... jmp   4081b3 ...
```

