ЛАБОРАТОРИЯ
ИНФОРМАЦИОННЫХ
ТЕХНОЛОГИЙ
имени М.Г. Мещерякова

Профилирование пакета программ SpdRoot

Дидоренко Алексей Викторович
(стажёр-исследователь ЛИТ ОИЯИ, didorenko@jinr.ru)

Бадмаев Ананда Аюрович
(магистрант СПбГУ, st055663@student.spbu.ru)

Цели и задачи

Целью работы было выявление узких мест в исходном коде пакета программ SpdRoot. Для достижения поставленной цели необходимо было выполнить ряд **задач**:

- изучить принципы запуска процессов генерации и реконструкции событий SPD
- выбрать программный инструментарий для поиска узких мест в коде SpdRoot
- изучить основы работы с выбранным программным инструментарием
- провести серию экспериментов и выявить проблемные места в коде SpdRoot

Метод поиска - профилирование

Профилирование используется для наблюдения за выполнением программы для сбора данных о различных аспектах, таких как:

- временная сложность;
- пространственная сложность.

Цель профилирования — выявить узкие места или области, где программа может быть оптимизирована для повышения ее эффективности и производительности.

Профилирование может быть:

- статическим (анализ кода программы без ее выполнения)
- динамическим (отслеживает программу во время её выполнения)

perf как инструмент для анализа производительности ПО

perf - инструмент динамического профилирования, который предназначен для систем на базе Linux. Преимущества:

- простой интерфейс командной строки
- отсутствие требования к перекомпилированию анализируемого ПО, по сравнению с такими утилитами профилирования, как gprof.

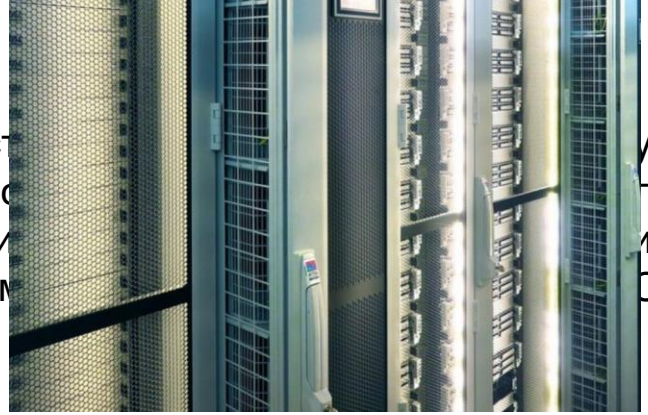
Для анализа производительности SpdRoot использовались такие perf - команды как:

- perf top
- perf record
- perf report

```
[alxdid@ncx104 ~]$ perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data file
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   c2c             Shared Data C2C/HITM Analyzer.
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   ftrace          simple wrapper for kernel's ftrace functionality
   inject          Filter to augment the events stream with additional information
   kallsyms        Searches running kernel for symbols
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   version         display the version of perf binary
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.
```

Описание эксперимента

Все эксперименты проводились на NICA - класт[illegible]усков SpdRoot.
Количество генерируемых событий = 100. Для с[illegible]готовлен bash - скрипт, включающий в себя perf - команды запи[illegible]и программы, для извлечения имён функций, занимающих максим[illegible]CPU.



```
spdroot.py -l -b -q "simu.C(100)"
spdroot.py -l -b -q reco.C
```

```
#!/bin/bash
echo "execution started.."
while [ true ]; do
    echo "running.."
    perf record -a sleep 1
    perf report --stdio --field-separator=';' --quiet | sed -n '1p' >> "$1"
done
```

Полученные результаты

Events	Simulation			Reconstruction		
	Function	% CPU	Time (s)	Function	% CPU	Time (s)
100	**FairMCApplication::Stepping**	**19**	178	**RKTrackRep::RKPropagate**	**7**	396
	TGeoIterator::Next	28		SpdFieldMap1_8::Approx_0	7	
	G4Region::BelongsTo	63		TGeoNavigator::Safety	8	
	G4ElasticHadrNucleusHE::HadrNucDifferCrSec	18		TGeoManager::CountLevels	3	
	G4hPairProductionModel::ComputeDMicroscopicCrossSection	40		TGeoIterator::Next	15	
	G4VRangeToEnergyConverter::ConvertCutToKineticEnergy	22		MaterialEffects::dEdxBrems	15	
	TGeoManager::CountLevels	16		TGeoHMatrix::Multiply	7	
	SpdFieldMap1_8::Approx_0	28		**__mul_avx**	**29**	

Заключение

- Изучены принципы запуска макросов SpdRoot
- Выбрана и изучена утилита для анализа производительности изучаемого ПО
- Обнаружены функции, которые использовали большой процент CPU д (Stepping, RKPropagate, BelongsTo и пр.)
- Найдена функция mul_avx, которая использовала достаточно высокий процент CPU при торможении процесса реконструкции событий.

Результаты являются достаточно грубыми, так как требуется больше статистических данных при разных количествах событий.
Полученные результаты могут быть применены как основа для дальнейшего устранения узких мест процессов симуляции и реконструкции. В случае отсутствия возможности кардинального ускорения реконструкции, будет рассмотрен вариант разработки нового алгоритма.

Спасибо за внимание!