Новосибирский государственный университет
Факультет информационных технологий
Кафедра параллельных вычислений

Эффективное программирование современных микропроцессоров и мультипроцессоров

Профилирование программ

Преподаватели:
Киреев С.Е.
Калгин К.В.

Профилирование

- Профилирование – сбор характеристик работы программы

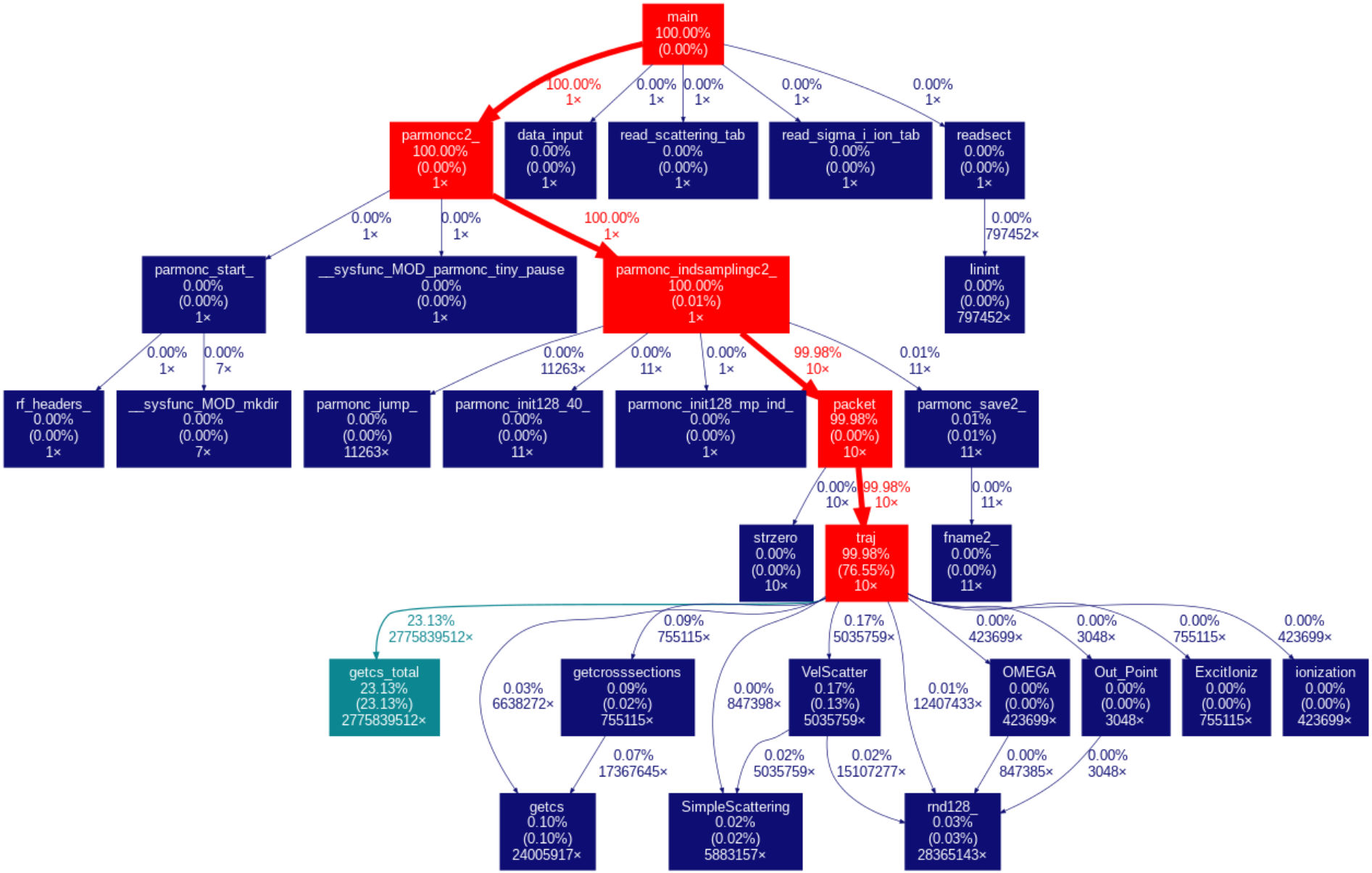
- Цели профилирования
  - Оценка эффективности работы программы или вычислительной системы
  - Определение критических участков программы (hotspots) или компонентов вычислительной системы

Результаты профилирования

- Определение путей исполнения
  - Покрытие кода, пути срабатывания переходов, граф вызовов, количество исполнений, …
- Произошедшие события
  - Такты, исполненные команды, чтение/запись памяти, кэш-промахи/попадания, (не)предсказанные переходы, …
- Потребление ресурсов
  - Время, ядра, память, сеть, электроэнергия, …

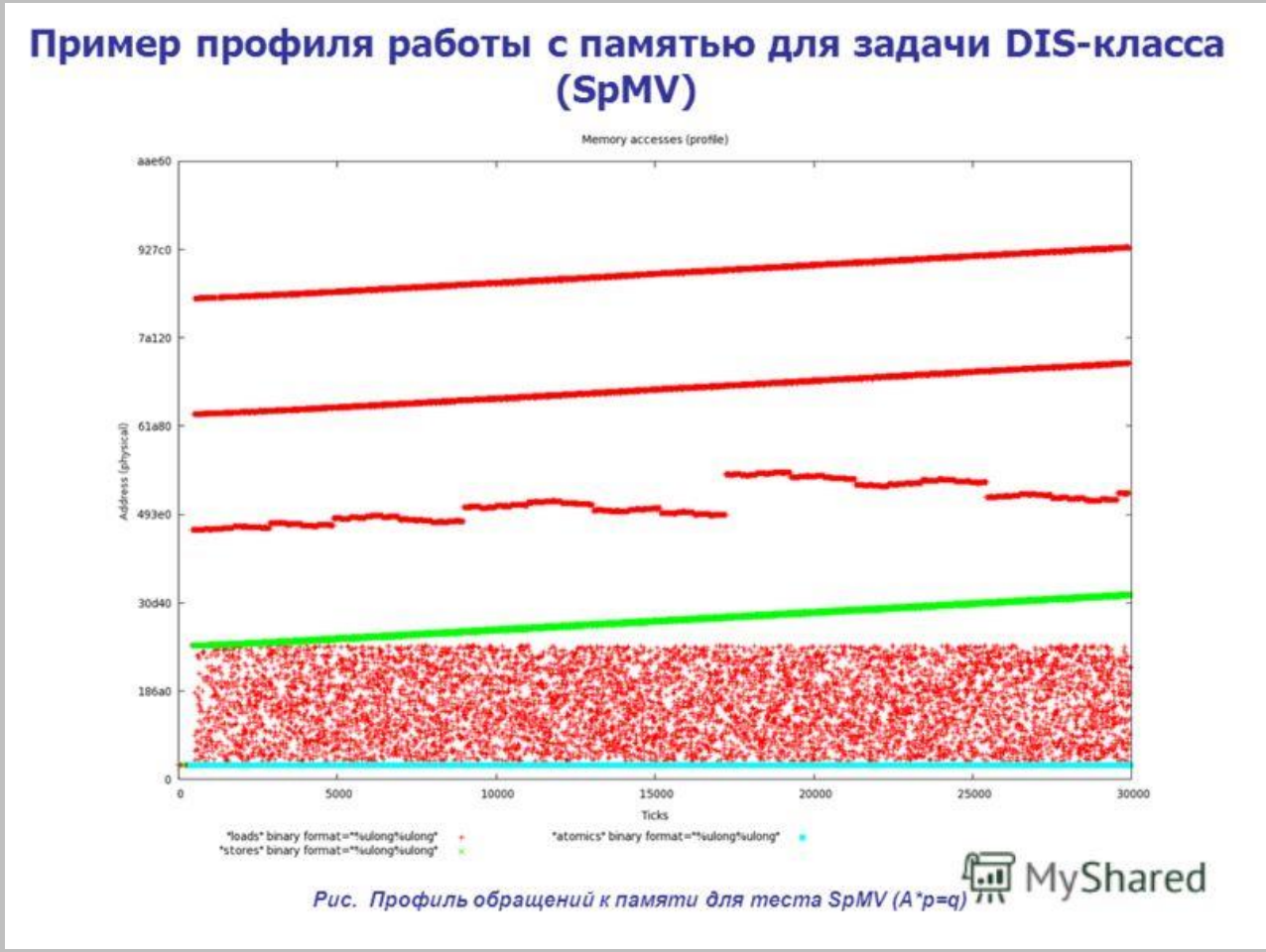
Результаты профилирования

- **Представление результатов**
  - **Аннотированный листинг**
    - Отображение информации на исходный код
  - **Профиль исполнения**
    - Потребление ресурсов/количество событий с течением времени
  - **Специальные представления**
    - Граф вызовов подпрограмм (call graph, flame graph)
    - Диаграмма передачи данных между процессами
    - …
- **Детализация информации**
  - Вся программа, модули, подпрограммы, строки кода, оператора языка, машинные команды, …
- **Анализ результатов**
  - Оценка (хорошо/плохо)
  - Обнаружение «узких мест» в программе (hotspots, critical path)
  - Определение возможных причин
  - Советы по улучшению
  - …


main
100.00%
(0.00%)
100.00%
1×
0.00%
1×
0.00%
1×
0.00%
1×
0.00%
1×
parmoncc2_
100.00%
(0.00%)
1×
data_input
0.00%
(0.00%)
1×
read_scattering_tab
0.00%
(0.00%)
1×
read_sigma_i_ion_tab
0.00%
(0.00%)
1×
readsect
0.00%
(0.00%)
1×
0.00%
1×
0.00%
1×
100.00%
1×
0.00%
797452×
parmonc_start_
0.00%
(0.00%)
1×
__sysfunc_MOD_parmonc_tiny_pause
0.00%
(0.00%)
1×
parmonc_indsamplingc2_
100.00%
(0.01%)
1×
linint
0.00%
(0.00%)
797452×
0.00%
1×
0.00%
7×
0.00%
11263×
0.00%
11×
0.00%
1×
99.98%
10×
0.01%
11×
rf_headers_
0.00%
(0.00%)
1×
__sysfunc_MOD_mkdir
0.00%
(0.00%)
7×
parmonc_jump_
0.00%
(0.00%)
11263×
parmonc_init128_40_
0.00%
(0.00%)
11×
parmonc_init128_mp_ind_
0.00%
(0.00%)
1×
packet
99.98%
(0.00%)
10×
parmonc_save2_
0.01%
(0.01%)
11×
0.00%
10×
99.98%
10×
0.00%
11×
strzero
0.00%
(0.00%)
10×
traj
99.98%
(76.55%)
10×
fname2_
0.00%
(0.00%)
11×
23.13%
2775839512×
0.09%
755115×
0.17%
5035759×
0.00%
423699×
0.00%
3048×
0.00%
755115×
0.00%
423699×
getcs_total
23.13%
(23.13%)
2775839512×
0.03%
6638272×
getcrosssections
0.09%
(0.02%)
755115×
0.00%
847398×
VelScatter
0.17%
(0.13%)
5035759×
0.01%
12407433×
OMEGA
0.00%
(0.00%)
423699×
Out_Point
0.00%
(0.00%)
3048×
ExcitIoniz
0.00%
(0.00%)
755115×
ionization
0.00%
(0.00%)
423699×
0.07%
17367645×
0.02%
5035759×
0.02%
15107277×
0.00%
847385×
0.00%
3048×
getcs
0.10%
(0.10%)
24005917×
SimpleScattering
0.02%
(0.02%)
5883157×
rnd128_
0.03%
(0.03%)
28365143×

Результаты профилирования

- Профиль обращений к памяти



Способы профилирования

- **Иструментирование кода** – внесение в программу дополнительного кода для сбора нужной информации
  - + Позволяет получить точные значения некоторых характеристик
  - + Можно сделать вручную, без дополнительных средств
  - – Сильное влияние на временные характеристики программы
- **Обработка событий** – код профилировщика вызывается при срабатывании определенных событий (new, delete, call, return, …)
  - + Позволяет получить точные значения некоторых характеристик
  - – Заметное влияние на временные характеристики программы
- **Статистическое профилирование (sampling)** – остановка по системным/аппаратным прерываниям и накопление статистики на точках останова
  - + Малое влияние на характеристики программы
  - – Результаты профилирования приблизительны
- **Эмуляция выполнения программы**
  - + Позволяет получить точные значения некоторых характеристик
  - + Повторяемость результата
  - – Значительно большее время работы программы
  - – Не учитывается «реальная» ситуация

Препятствия к профилированию

- **Оптимизация программы компилятором**
  - Компилятор удаляет команды, заменяет на другие, меняет команды местами, …
- **Динамическое исполнение кода в процессоре**
  - Процессор меняет порядок исполнения команд, выполняет команды параллельно, …

Возможные проблемы

- **Нет информации об исходном коде, есть только ассемблер**
  - Компиляция с символьной информацией: **-g**
  - Предоставить исходный код (если поддерживается)

- **Неправильно определяется граф вызовов**
  - Запретить inline: **-fno-inline**
  - Обязать использовать **%rbp**: **-fno-omit-frame-pointer**
  - Использовать Hardware Last Branch Record facility: **perf --callgraph lbr**

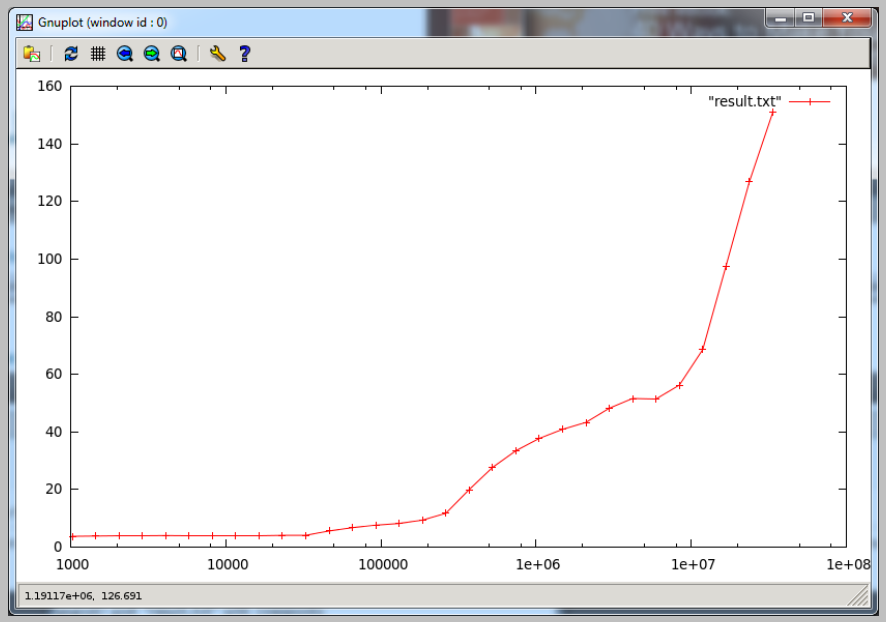
- **Собрана маленькая статистика, много событий пропущено**
  - Собирается слишком много событий одновременно
  - Посторонние прерывания (NMI watchdog, …)
  - …

Профилировщики

- gprof, gcov
- perf
  (https://github.com/torvalds/linux/tree/master/tools/perf)
- Oprofile
  (http://oprofile.sourceforge.net)
- Intel Vtune Profiler
  (https://software.intel.com/en-us/vtune)
- Intel Advisor
  (https://software.intel.com/en-us/advisor)
- AMD µProf
  (https://developer.amd.com/amd-uprof/)
- Optick: C++ Profiler For Games
  (https://github.com/bombomby/optick)
- Dtrace
  (http://dtrace.org/blogs/about/, https://github.com/opendtrace)
- Valgrind
  (http://valgrind.org)
- Google perftools
  (https://github.com/gperftools/gperftools)
- …

Тестируемая программа

- Программа: определение времени доступа к элементам массива в случайном порядке в зависимости от размера массива
- Этапы работы:
  - Заполнение массива
    - a[j] = rand()
  - Обход массива
    - for (i=…) j = A[j]



gprof

- Компиляция программы
  - $g++ -O1 **-g -pg** -o cache cache.c
  - Создаётся файл: cache
- Запуск программы (много раз)
  - $./cache
  - Создается файл: gmon.out
- Получение профиля программы и графа вызовов
  - $gprof ./cache >cache.profile.txt
  - Создается файл: cache.profile.txt
- Получение аннотированного листинга
  - $gprof -A cache >cache.source.txt
  - Создается файл: cache.source.txt

gcov – тест покрытия кода

- Компиляция программы
  - $g++ -O1 **--coverage** -o cache cache.c
  - Создаются файлы: cache, cache.gcno
- Запуск программы (много раз)
  - $./cache
  - Создаётся файл: cache.gcda
- Получение аннотированного листинга
  - $gcov -b cache.c
  - Создаётся файл: cache.c.gcov

Оптимизация в gcc с помощью профилирования

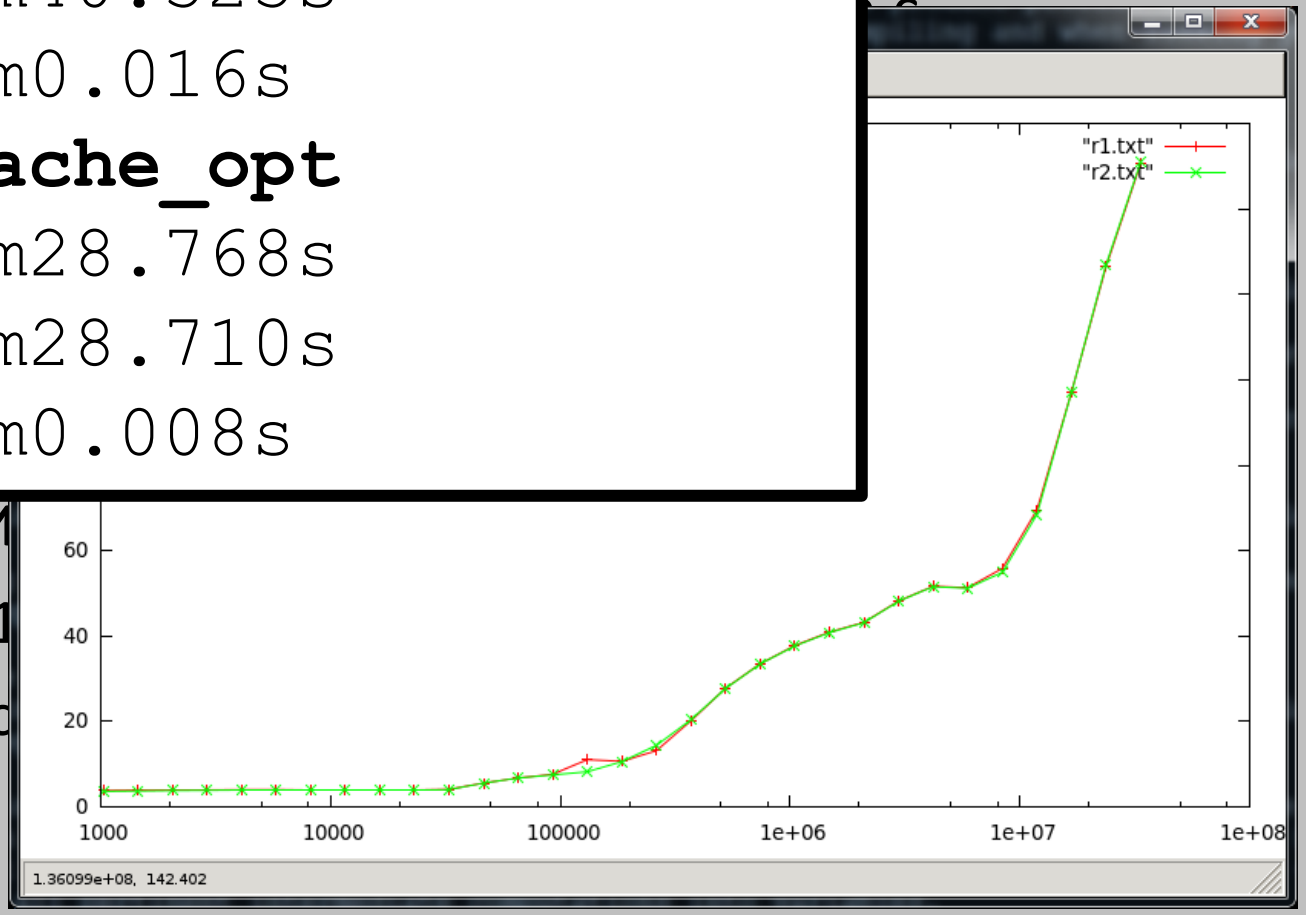
- Компиляция программы
  - $g++ -O1 **-fprofile-generate** -o cache cache.c
  - Создаётся файл: cache
- Запуск программы (много раз)
  - $./cache
  - Создаётся файл: cache.gcda
- Компиляция программы с оптимизацией
  - $ g++ -O1 **-fprofile-use** -o cache_opt cache.c
  - Создается файл: cache_opt

Оптимизация в gcc с помощью

Пример (хороший)

```
$time ./cache
real    0m40.409s
user    0m40.323s
sys     0m0.016s
$time ./cache_opt
real    0m28.768s
user    0m28.710s
sys     0m0.008s
```

- Компиляци
  - $ g++ -O1
  - Создаетс



Оптимизация в gcc с помощью

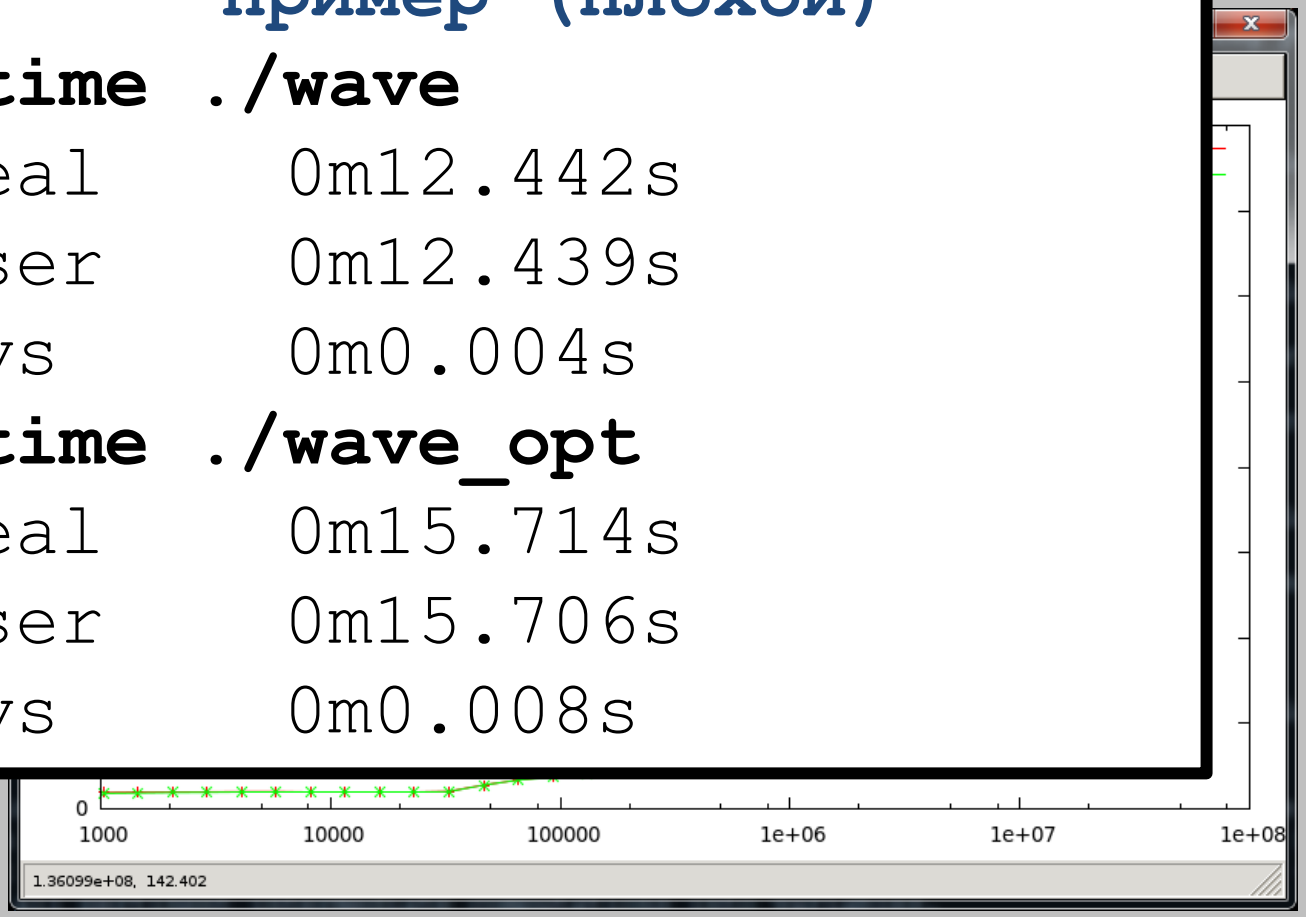
Пример (хороший)

```
$time ./cache
real    0m40.409s
user
sys
$time .
real
user
sys
```

Пример (плохой)

```
$time ./wave
real    0m12.442s
user    0m12.439s
sys     0m0.004s
$time ./wave_opt
real    0m15.714s
user    0m15.706s
sys     0m0.008s
```

- Компиля
  - $ g++
  - Созд



perf

- Позволяет получать аппаратные и программные счетчики производительности системы:

192.168.12.31 - PuTTY

```
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                               [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend      [Hardware event]
  instructions                                       [Hardware event]
  cache-references                                   [Hardware event]
  cache-misses                                       [Hardware event]
  branch-instructions OR branches                    [Hardware event]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]

  cpu-clock                                          [Software event]
  task-clock                                         [Software event]
  page-faults OR faults                              [Software event]
  minor-faults                                       [Software event]
  major-faults                                       [Software event]
  context-switches OR cs                             [Software event]
  cpu-migrations OR migrations                       [Software event]
  alignment-faults                                   [Software event]
  emulation-faults                                   [Software event]

  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
  L1-dcache-stores                                   [Hardware cache event]
  L1-dcache-store-misses                             [Hardware cache event]
  L1-dcache-prefetches                               [Hardware cache event]
  L1-dcache-prefetch-misses                          [Hardware cache event]
  L1-icache-loads                                    [Hardware cache event]
  L1-icache-load-misses                              [Hardware cache event]
  L1-icache-prefetches                               [Hardware cache event]
:
```

perf

- Компиляция программы
  - $g++ -O1 **-g** -o cache cache.c
  - Создаётся файл: cache
- Запуск программы с профилированием
  - $perf record -g -e cycles -e cache-misses -- ./cache
  - Создается файл: perf.data
- Получение профиля программы и графа вызовов
  - $perf report
- Получение аннотированного листинга
  - $perf annotate

perf

- Анализ программы во время выполнения
  - sudo perf top -e cycles -e cache-misses
- Статистика выполнения программы
  - perf stat ./executable
  - perf stat -e LLC-loads -e LLC-load-misses ./executable

kireev@mineral: ~/work/pic3d_common/simple

```
kireev@mineral:~/work/pic3d_common/simple$ perf stat ./pic3d_simple
Parameters read from file: 'pic3d.par'
Mesh: 200 x 200 x 200 / 1 x 1 x 1   Particles: 1000000   Steps: 30 (10,0)
Domain: 1.000000 x 1.000000 x 1.000000   Time step: 0.000100
Particles initial shape: CIRCLE:XY   radius: 0.200000
Particles initial velocity: CIRCLE:XY
Particles mass: 0.500000   Sun mass: 0.500000
Poisson solver threshold: 1.000000e-02   acceleration parameter: 1.000000
   0 : c: 5.5790054821e-18   p: 3.2755969266e-17   e: -2.5168325877e+00
  10 : c: 7.6034943874e-18   p: 1.2999034034e-16   e: -2.5287723065e+00
  20 : c: 7.5136612230e-18   p: 2.4019049068e-16   e: -2.5387413561e+00
  30 : c: 6.4655685721e-18   p: 2.8593021611e-16   e: -2.5479554329e+00
Wall time init: 1.198043   count: 6.688023   total: 7.886066

 Performance counter stats for './pic3d_simple':

       7899,617714      task-clock (msec)         #    1,000 CPUs utilized
                 6      context-switches          #    0,001 K/sec
                 0      cpu-migrations            #    0,000 K/sec
           160 717      page-faults               #    0,020 M/sec
    29 868 299 970      cycles                    #    3,781 GHz
    44 951 539 131      instructions              #    1,50  insn per cycle
     3 373 107 551      branches                  #  426,996 M/sec
         8 817 298      branch-misses             #    0,26% of all branches

       7,899793330 seconds time elapsed

kireev@mineral:~/work/pic3d_common/simple$
```

нения

ecutable

нения

kireev@mineral: ~/work/pic3d_common/simple

```
kireev@mineral:~/work/pic3d_common/simple$ perf stat ./pic3d_simple
Parameters read from file: 'pic3d.par'
Mesh: 200 x 200 x 200 / 1 x 1 x 1   Particles: 1000000   Steps: 30 (10,0)
Domain: 1.000000 x 1.000000 x 1.000000   Time step: 0.000100
Particles initial shape: CIRCLE:XY   radius: 0.200000
Particles initial velocity: CIRCLE:XY
Particles mass: 0.500000   Sun mass: 0.500000
Poisson solver threshold: 1.000000e-02   acceleration parameter: 1.000000
   0 : c: 5.5790054821e-18   p: 3.2755969266e-17   e: -2.5168325877e+00
  10 : c: 7.6034943874e-18   p: 1.2999034034e-16   e: -2.5287723065e+00
  20 : c: 7.5136612230e-18   p: 2.4019049068e-16   e: -2.5387413561e+00
  30 : c: 6.4655685721e-18   p: 2.8593021611e-16   e: -2.5479554329e+00
```

kireev@mineral: ~/work/pic3d_common/simple

```
kireev@mineral:~/work/pic3d_common/simple$ perf stat -e LLC-loads -e LLC-load-misses ./pic3d_simple
Parameters read from file: 'pic3d.par'
Mesh: 200 x 200 x 200 / 1 x 1 x 1   Particles: 1000000   Steps: 30 (10,0)
Domain: 1.000000 x 1.000000 x 1.000000   Time step: 0.000100
Particles initial shape: CIRCLE:XY   radius: 0.200000
Particles initial velocity: CIRCLE:XY
Particles mass: 0.500000   Sun mass: 0.500000
Poisson solver threshold: 1.000000e-02   acceleration parameter: 1.000000
   0 : c: 5.5790054821e-18   p: 3.2755969266e-17   e: -2.5168325877e+00
  10 : c: 7.6034943874e-18   p: 1.2999034034e-16   e: -2.5287723065e+00
  20 : c: 7.5136612230e-18   p: 2.4019049068e-16   e: -2.5387413561e+00
  30 : c: 6.4655685721e-18   p: 2.8593021611e-16   e: -2.5479554329e+00
Wall time init: 1.180656   count: 6.795973   total: 7.976629

 Performance counter stats for './pic3d_simple':

        46 096 550      LLC-loads
        21 292 479      LLC-load-misses          #   46,19% of all LL-cache hits

       7,990322783 seconds time elapsed

kireev@mineral:~/work/pic3d_common/simple$
```

OProfile

- Позволяет получать аппаратные счетчики производительности процессора:

kireev@xeon: ~/work/profiling

```
kireev@xeon:~/work/profiling$ ophelp | head -n 30
oprofile: available events for CPU type "Intel Westmere microarchitecture"

See Intel Architecture Developer's Manual Volume 3B, Appendix A and
Intel Architecture Optimization Reference Manual (730795-001)

For architectures using unit masks, you may be able to specify
unit masks by name.  See 'opcontrol' or 'operf' man page for more details.

CPU_CLK_UNHALTED: (counter: all)
        Clock cycles when not halted (min count: 6000)
INST_RETIRED: (counter: all)
        number of instructions retired (min count: 6000)
LLC_MISSES: (counter: all)
        Last level cache demand requests from this core that missed the LLC (min count: 6000)
        Unit masks (default 0x41)
        ----------
        0x41: No unit mask
LLC_REFS: (counter: all)
        Last level cache demand requests from this core (min count: 6000)
        Unit masks (default 0x4f)
        ----------
        0x4f: No unit mask
BR_INST_RETIRED: (counter: all)
        number of branch instructions retired (min count: 500)
BR_MISS_PRED_RETIRED: (counter: all)
        number of mispredicted branches retired (precise) (min count: 500)
LOAD_BLOCK: (counter: all)
        Loads that partially overlap an earlier store (min count: 200000)
        Unit masks (default 0x2)
        ----------
kireev@xeon:~/work/profiling$
```

OProfile

- Получение списка аппаратных счетчиков
  - $ophelp
- Компиляция программы
  - $g++ -O1 -g -o cache cache.c
- Запуск программы и профилирование
  - $operf -e CPU_CLK_UNHALTED:100000:0:0:1 ./cache
  - Создаётся каталог: ./oprofile_data
- Получение информации об исполнении программы
  - $opreport ./cache - профиль по модулям
  - $opreport -l ./cache - профиль по функциям
  - $opreport -c ./cache - граф вызовов
  - $opannotate -s ./cache - аннотированный исходный код
  - $opannotate -a ./cache - аннотированный ассемблерный код
  - $opgprof - генерация gmon.out для gprof

OProfile

- Пример использования:

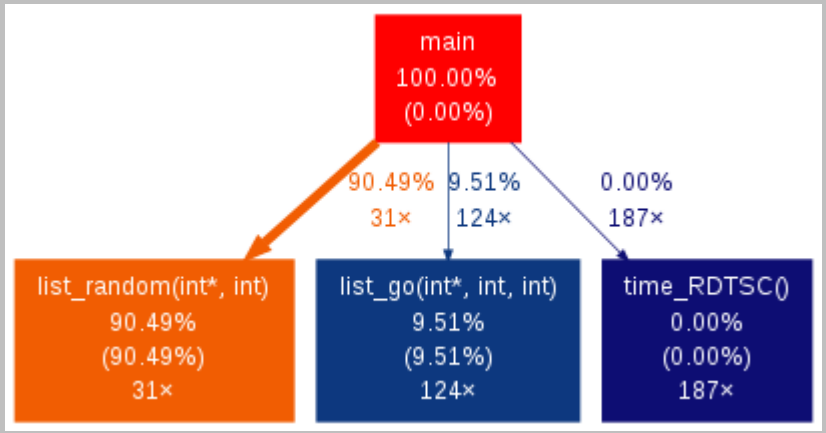
```
$operf -e \
CPU_CLK_UNHALTED:100000:0:1:1,\
LLC_REFS:100000:0x4f:0:1:1,\
LLC_MISSES:100000:0x41:0:1:1 ./cache
```

- Ссылки:
  - Сайт: http://oprofile.sourceforge.net
  - W. Cohen, Tuning programs with OProfile // Wide Open Magazine, 2004, p. 53–62. http://people.redhat.com/wcohen/Oprofile.pdf

gprof2dot: рисование графа вызовов

- Сайт:
  - https://code.google.com/p/jrfonseca/wiki/Gprof2Dot

- Пример:
  - gprof ./cache | ./gprof2dot.py -f prof -n0 -e0 | \
    dot -Tpng -o callgraph.png

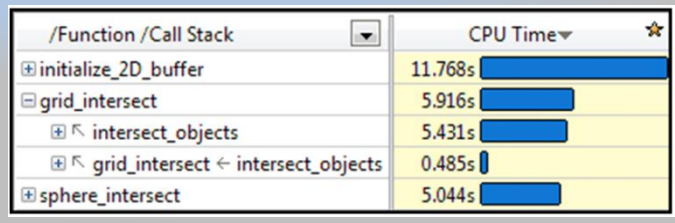


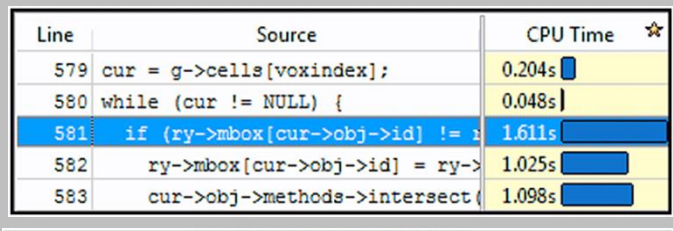
Продукты Intel для анализа программ

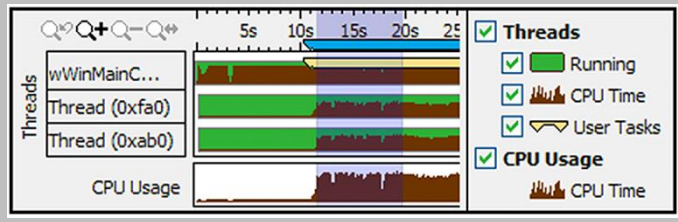
- Intel® Vtune™ Profiler
  - https://software.intel.com/en-us/vtune
- Intel® Advisor
  - https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/advisor.html
- Intel® Inspector
  - https://software.intel.com/content/www/us/en/develop/tools/oneapi/components/inspector.html

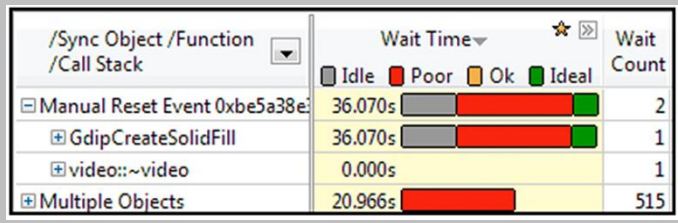
Intel VTune Profiler

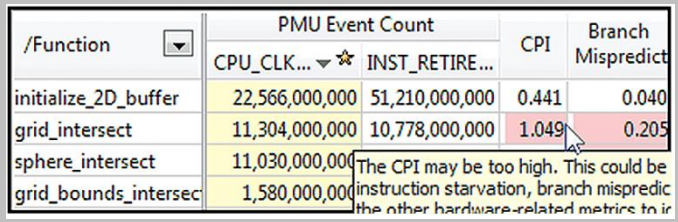
- Получение информации с точностью до команд ассемблера
- Построение графа вызовов
- Получение значений аппаратных счетчиков производительности (Intel)
- Динамика загрузки ядер процессора
- Динамика работы многопоточных программ (ожидание на блокировках)
- Roofline-анализ
- Подсказки вариантов оптимизации
- Удобный графический интерфейс
- Поддержка только процессоров Intel
- …еще много плюсов…
- Доступен бесплатно:
  https://software.intel.com/en-us/vtune











Intel VTune Profiler

Basic Hotspots Hotspots viewpoint (change) Intel VTune Amplifier XE 2013

Collection Log | Summary | Bottom-up | Caller/Callee | Top-down Tree | Tasks and Frames | Graphics

Grouping: Function / Call Stack

Function / Call Stack	CPU Time
FireObject::checkCollision	9.376s
FireObject::ProcessFireCollisionsRange	8.310s
FireObject::FireCollisionCallback ← ParallelForBody::operato	6.896s
FireObject::EmitterCollisionCheck ← FireObject::UpdateRang	1.414s
NtWaitForSingleObject	3.764s
Selected 1 row(s):	8.310s

CPU Function/CPU Stack - CPU Time

Viewing 1 of 58 selected stack(s)

24.9% (2.069s of 8.310s)

SystemProceduralFire.DLL!FireObject::Pr...
SystemProceduralFire.DLL!FireObject::Fi...
Smoke.exe!ParallelForBody::operator()+...
Smoke.exe![TBB parallel_for on class Para...
Smoke.exe!TaskManagerTBB::ParallelFo...

26.6s 26.7s 26.8s 26.9s 27s 27.1s 27.2s 27.3s 27.4s 27.5s 27.6s 27.7s

Thread
wWinMainCRTStartup (
endthreadex (0x9e4)
endthreadex (0xef8)
endthreadex (0x2518)

CPU Usage

Frame Rate

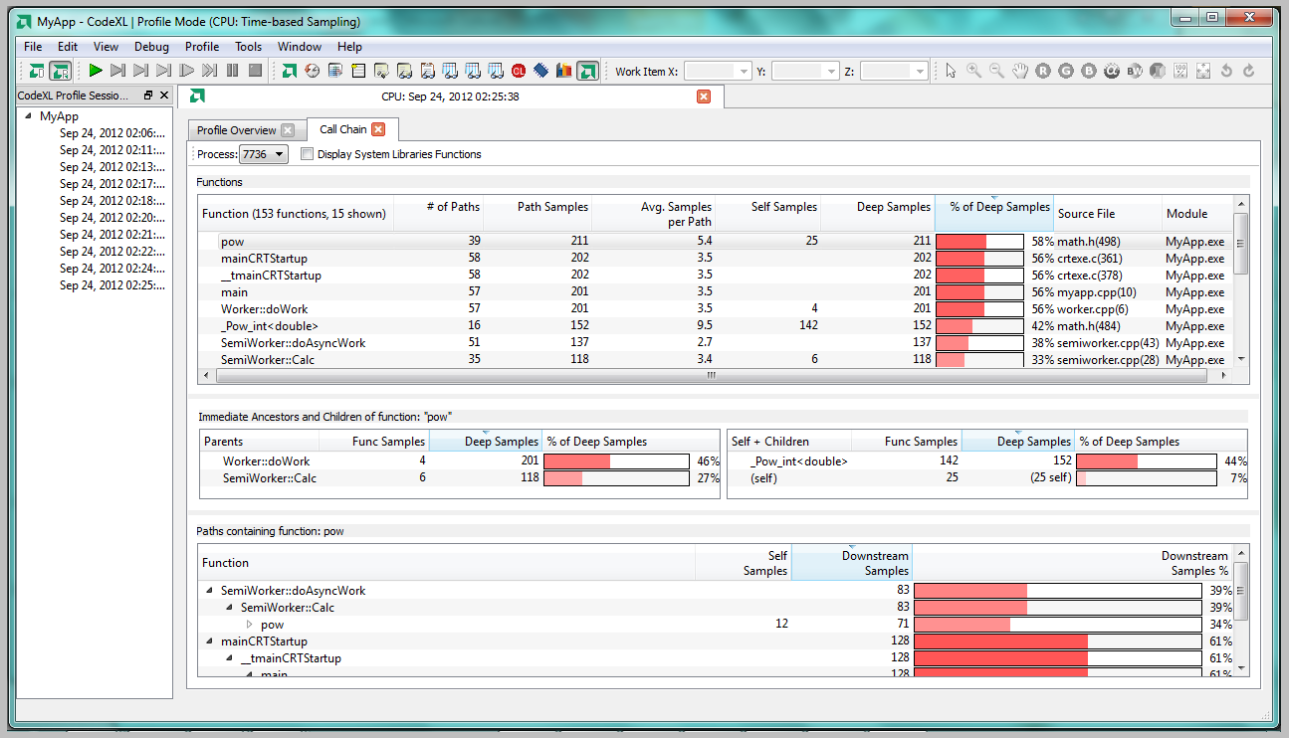
Ruler Area
User Marks
Frame
Thread
Running
CPU Time
Tasks
CPU Usage

No filters are applied. | Any Process | Any Thread | Any Module

Call Stack Mode: Only user functions | Inline Mode: on | Loop Mode: Functions only

AMD CodeXL

- Набор инструментов разработки программ, включающий CPU/GPU-профилировщик, GPU-отладчик, статистический анализатор ядра OpenCL.



- В свободном доступе
- Сайт: https://github.com/GPUOpen-Tools/CodeXL

PAPI – Performance Application Programming Interface

- Интерфейс (библиотека) для доступа к счетчикам производительности на различных современных микропроцессорах. Позволяет встраивать анализ производительности в разрабатываемые программы.

- Доступна к использованию на языках C/C++, Fortran, Java, Matlab, …

- На базе PAPI реализовано множество программных систем для изучения производительности программ.

- Сайт: https://icl.utk.edu/papi/, https://bitbucket.org/icl/papi/
  - Пример использования: http://www.drdobbs.com/tools/performance-monitoring-with-papi/184406109

```
int main() {

  long long values[3];
  int event_set = PAPI_NULL;

  PAPI_library_init(PAPI_VER_CURRENT);
  PAPI_create_eventset(&event_set);

  PAPI_add_event(event_set, PAPI_BR_CN  );
  PAPI_add_event(event_set, PAPI_BR_MSP );
  PAPI_add_event(event_set, PAPI_BR_PRC );

  std::vector<int> x(n);

  for (size_t n = 100; n<=1000; n+=100) {

    std::generate(x.begin(),x.end(),std::rand);

    PAPI_start(event_set);

    std::sort(x.begin(),x.end());

    PAPI_stop(event_set, values);

    std::cout << values[0] << " " << values[1] << " " << values[2] << std::endl;
  }

  PAPI_cleanup_eventset(event_set);
  PAPI_destroy_eventset(&event_set);
  return 0;
}
```

Intel Pin

- Средство для динамического инструментирования кода для архитектур IA-32, x86-64 и MIC.
- Позволяет создавать средства динамического анализа программ.

- Сайт: https://software.intel.com/en-us/articles/pin-a-dynamic-binary-instrumentation-tool