Андрей Хорохорин

Разработка анализатора производительности программ по трассе исполнения с учётом микроархитектурных особенностей процессорного комплекса

Выпускная квалификационная работа

Научный руководитель: Е.М. Линский

10.06.2025

Факультет математики и компьютерных наук СПбГУ
Программа «Современное программирование»

Постановка задачи

При разработке производительных процессорных ядер для оценки полезности нововведений постоянно прибегают к измерению на бенчмарках.

Пример HWT трассы:

```
< 5 > [00800051fc] lh      a4, 2(a4)        RD[0e]=000000000000000f
< 5 > [0080005200] bne     s9, a4, pc - 12
< 6 > [00800051f4] c.ld    a5, 0(a5)        RD[0f]=00000000800136b8
```

Главная цель: облегчить анализ производительности и поиск проблемных мест при изменении как аппаратной, так и программной части.

Постановка задачи

При разработке производительных процессорных ядер для оценки полезности нововведений постоянно прибегают к измерению на бенчмарках.

Пример HWT трассы:

```
< 5 > [00800051fc] lh      a4, 2(a4)        RD[0e]=000000000000000f
< 5 > [0080005200] bne     s9, a4, pc - 12
< 6 > [00800051f4] c.ld    a5, 0(a5)        RD[0f]=00000000800136b8
```

Главная цель: облегчить анализ производительности и поиск проблемных мест при изменении как аппаратной, так и программной части.

- подсказывать причину и место инцидента

Постановка задачи

При разработке производительных процессорных ядер для оценки полезности нововведений постоянно прибегают к измерению на бенчмарках.

Пример HWT трассы:

```
< 5 > [00800051fc] lh      a4, 2(a4)        RD[0e]=000000000000000f
< 5 > [0080005200] bne     s9, a4, pc - 12
< 6 > [00800051f4] c.ld    a5, 0(a5)        RD[0f]=00000000800136b8
```

Главная цель: облегчить анализ производительности и поиск проблемных мест при изменении как аппаратной, так и программной части.

- подсказывать причину и место инцидента
- возможность работать без ОС

Постановка задачи

При разработке производительных процессорных ядер для оценки полезности нововведений постоянно прибегают к измерению на бенчмарках.

Пример HWT трассы:

```
< 5 > [00800051fc] lh      a4, 2(a4)        RD[0e]=000000000000000f
< 5 > [0080005200] bne     s9, a4, pc - 12
< 6 > [00800051f4] c.ld    a5, 0(a5)        RD[0f]=00000000800136b8
```

Главная цель: облегчить анализ производительности и поиск проблемных мест при изменении как аппаратной, так и программной части.

- подсказывать причину и место инцидента
- возможность работать без ОС
- учёт микроархитектурных особенностей при анализе

Постановка задачи

При разработке производительных процессорных ядер для оценки полезности нововведений постоянно прибегают к измерению на бенчмарках.

Пример HWT трассы:

```
< 5 > [00800051fc] lh      a4, 2(a4)        RD[0e]=000000000000000f
< 5 > [0080005200] bne     s9, a4, pc - 12
< 6 > [00800051f4] c.ld    a5, 0(a5)        RD[0f]=00000000800136b8
```

Главная цель: облегчить анализ производительности и поиск проблемных мест при изменении как аппаратной, так и программной части.

- подсказывать причину и место инцидента
- возможность работать без ОС
- учёт микроархитектурных особенностей при анализе
- минимальные искажения динамики исполнения

Постановка задачи

При разработке производительных процессорных ядер для оценки полезности нововведений постоянно прибегают к измерению на бенчмарках.

Пример HWT трассы:

```
< 5 > [00800051fc] lh      a4, 2(a4)        RD[0e]=000000000000000f
< 5 > [0080005200] bne     s9, a4, pc - 12
< 6 > [00800051f4] c.ld    a5, 0(a5)        RD[0f]=00000000800136b8
```

Главная цель: облегчить анализ производительности и поиск проблемных мест при изменении как аппаратной, так и программной части.

- подсказывать причину и место инцидента
- возможность работать без ОС
- учёт микроархитектурных особенностей при анализе
- минимальные искажения динамики исполнения
- удобство использования

Обзор средств профилировки программ

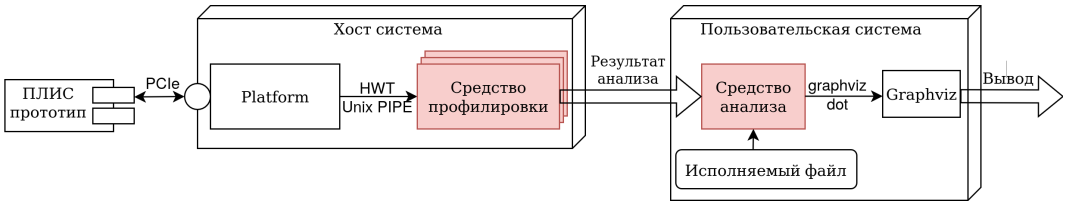
- HPM — *Hardware Performance Monitors (Аппаратные счётчики производительности)*
  Пример: Perf, Intel V-Tune
- Сэмплирующие
  Пример: Perf, Google Performance Tools
- Программные потактовые симуляторы
  Пример: Gem5

Обзор средств профилировки программ

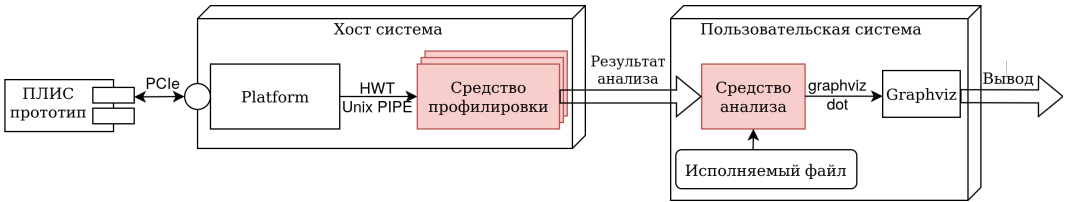
- HPM — *Hardware Performance Monitors (Аппаратные счётчики производительности)*
  Пример: Perf, Intel V-Tune
- Сэмплирующие
  Пример: Perf, Google Performance Tools
- Программные потактовые симуляторы
  Пример: Gem5

И всё не очень подходит в нашей ситуации!

Общая схема решения

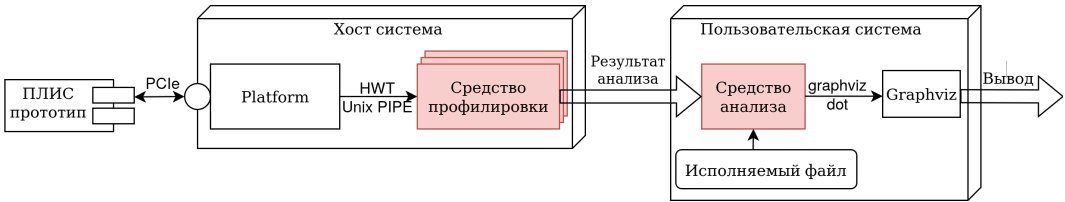


Общая схема решения



- анализ должен работать достаточно быстро, чтобы не тормозить FPGA

Общая схема решения



- анализ должен работать достаточно быстро, чтобы не тормозить FPGA
- бенчмарк запускается один раз, трасса не записывается на диск

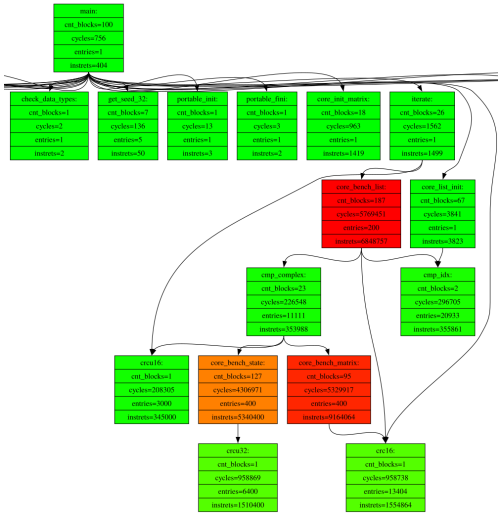
Пример применения

Проблема: очки бенчмарка coremark колеблются на $\sim 2\%$ в зависимости от длины строки с флагами компиляции.

За основу возьмём два исполняемых файла (первый лучше второго). Запускаем для обоих средство профилировки, получая два файла с профилями исполнения.

Пример применения

Выводим статистику по функциям:

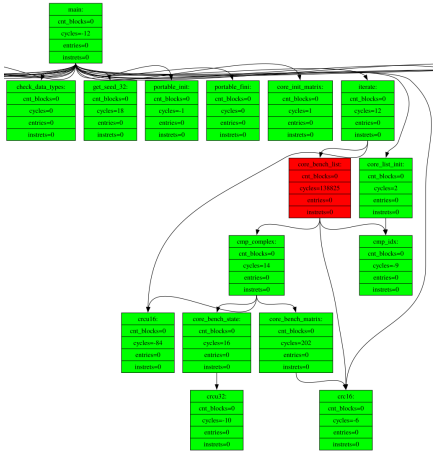


Пример применения

Сравнивать большое количество метрик на двух графах достаточно не удобно, поэтому можно воспользоваться режим для определения различий.

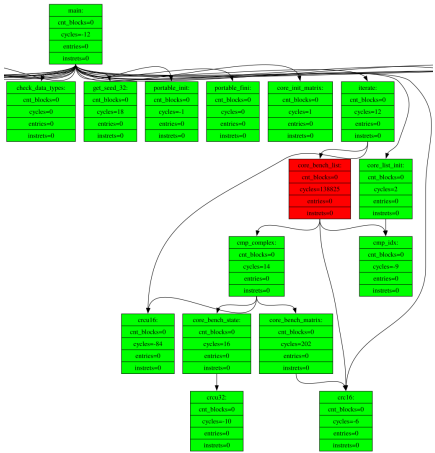
Пример применения

Сравнивать большое количество метрик на двух графах достаточно не удобно, поэтому можно воспользоваться режим для определения различий.



Пример применения

Сравнивать большое количество метрик на двух графах достаточно не удобно, поэтому можно воспользоваться режим для определения различий.



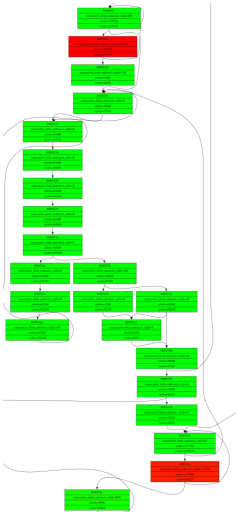
core_bench_list:
cycles=138825
entries=0
instrets=0
executed_nops=0
broken_timestamps=-69
cnt_insns=0
misaligned_branch_target_stalls=-2
misaligned_sfb_stalls=0
consecutive_fetch_redictects_stalls=393307
alu_data_stalls=-166
load_stalls=-268336
late_invokes=-453
late_dependency_stalls=215
mispredicted_branches=-46
mispredicted_branch_stalls=-276
hazard_stalls=-48

Сужаем область поиска

Выведем карту базовых блоков заданной функции.

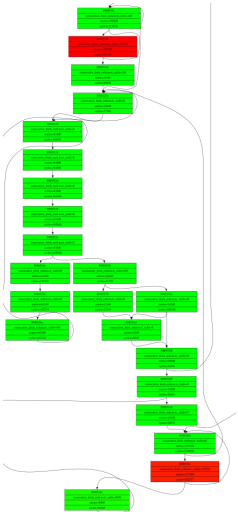
Сужаем область поиска

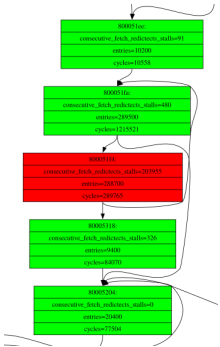
Выведем карту базовых блоков заданной функции.



Сужаем область поиска

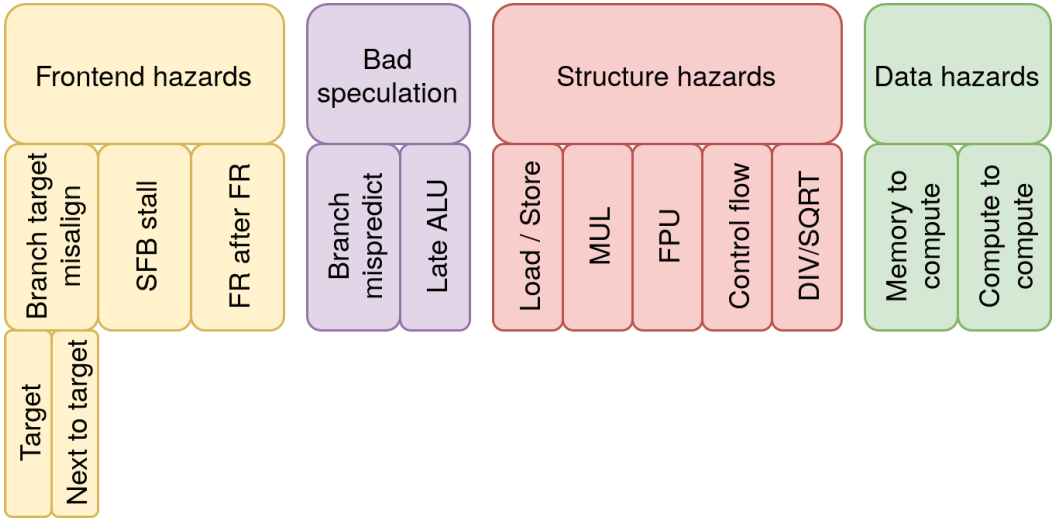
Выведем карту базовых блоков заданной функции.






```
800052f0:       ld a1, 8(a5)
800052f2:       lbu s6, 0(a1)
800052f6:       beq s7, s6, 80005204
800052fa:       ld a5, 0(a5)
800052fc:       bnez a5, 800052f0
```

Метрики


Frontend hazards
Branch target misalign
Target
Next to target
SFB stall
FR after FR
Bad speculation
Branch mispredict
Late ALU
Structure hazards
Load / Store
MUL
FPU
Control flow
DIV/SQRT
Data hazards
Memory to compute
Compute to compute

Пример: Late ALU

Рис. 4: Схема рассматриваемого конвейера

Инструкция исполняется как *late*, если:

- На момент начала исполнения существует неразрешённая зависимость по данным.
- Через 3 такта зависимость по данным *почти точно* будет разрешена.

Пример: Late ALU

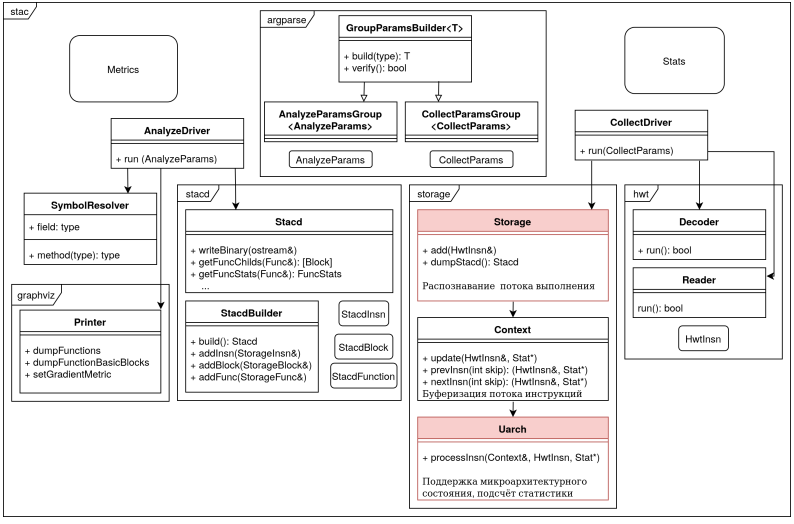
```
ld x1, addr1
ld x2, addr2
addi x1, x1, 1 (late)
addi x1, x1, 2 (late)
addi x1, x1, 3 (late)
addi x2, x2, 1
addi x2, x2, 2
addi x2, x2, 3
/* x1, x2 вычислено */
sd x1, addr1
sd x2, addr2
```

Рис. 5: Код без задержек

```
ld x1, addr1
ld x2, addr2
addi x1, x1, 1 (late)
addi x2, x2, 1 (late)
addi x1, x1, 2 (late)
addi x2, x2, 2 (late)
addi x1, x1, 3 (late)
addi x2, x2, 3 (late)
/* x1 и x2 будут вычислены через 3 такта */
sd x1, addr1
sd x2, addr2
```

Рис. 6: Код с задержкой

Архитектура разработанного средства



Результаты работы

В результате разработан инструмент, который:

- пригоден для поиска мест и причин снижения производительности

Результаты работы

В результате разработан инструмент, который:

- пригоден для поиска мест и причин снижения производительности
- в большинстве сценариев способен обрабатывать данные без потери скорости выполнения на FPGA

Результаты работы

В результате разработан инструмент, который:

- пригоден для поиска мест и причин снижения производительности
- в большинстве сценариев способен обрабатывать данные без потери скорости выполнения на FPGA
- требует лишь один запуск теста производительности для сбора всех метрик

Результаты работы

В результате разработан инструмент, который:

- пригоден для поиска мест и причин снижения производительности
- в большинстве сценариев способен обрабатывать данные без потери скорости выполнения на FPGA
- требует лишь один запуск теста производительности для сбора всех метрик
- значительно более точен, чем профилировка известными средствами

Результаты работы

В результате разработан инструмент, который:

- пригоден для поиска мест и причин снижения производительности
- в большинстве сценариев способен обрабатывать данные без потери скорости выполнения на FPGA
- требует лишь один запуск теста производительности для сбора всех метрик
- значительно более точен, чем профилировка известными средствами
- расширяем как в сторону добавления других микроархитектур, так и в сторону смены формата трассы.

Спасибо за внимание!

Ссылка на репозиторий с работой: github.com/khaser/stac

Ссылка на репозиторий со слайдами: github.com/khaser/stac-report

Рис. 7: QR-код ссылки на репозиторий с работой

Возможности для дальнейшей работы

- поддержка трассировки процесса, запущенного под linux
- поддержка формата трассы, с большим количеством видимых стадий выполнения инструкций
- улучшения в user experience.
  - переход на QT/GTK, интерактивность
  - возможность вводить пользовательские метрики в виде формул над существующими.