Московский физико-технический институт (государственный университет)
Физтех-школа радиотехники и компьютерных технологий(ФРКТ)
Кафедра информатики и вычислительной техники

Развитие генератора случайных тестов для верификации механизма передачи операндов в конвейере МП семейства Эльбрус

Выпускная квалификационная работа
(бакалаврская работа)

Студент: Балаховский Ф.М.
Научный руководитель: Фролов П.В.

Москва, 2023

Генератор тестов конвейера исполнения инструкций

Генератор формирует тестовые программы на языке ассемблер – случайные последовательности широких команд, в которых передача данных между отдельными инструкциями осуществляется через байпас.

Цель работы

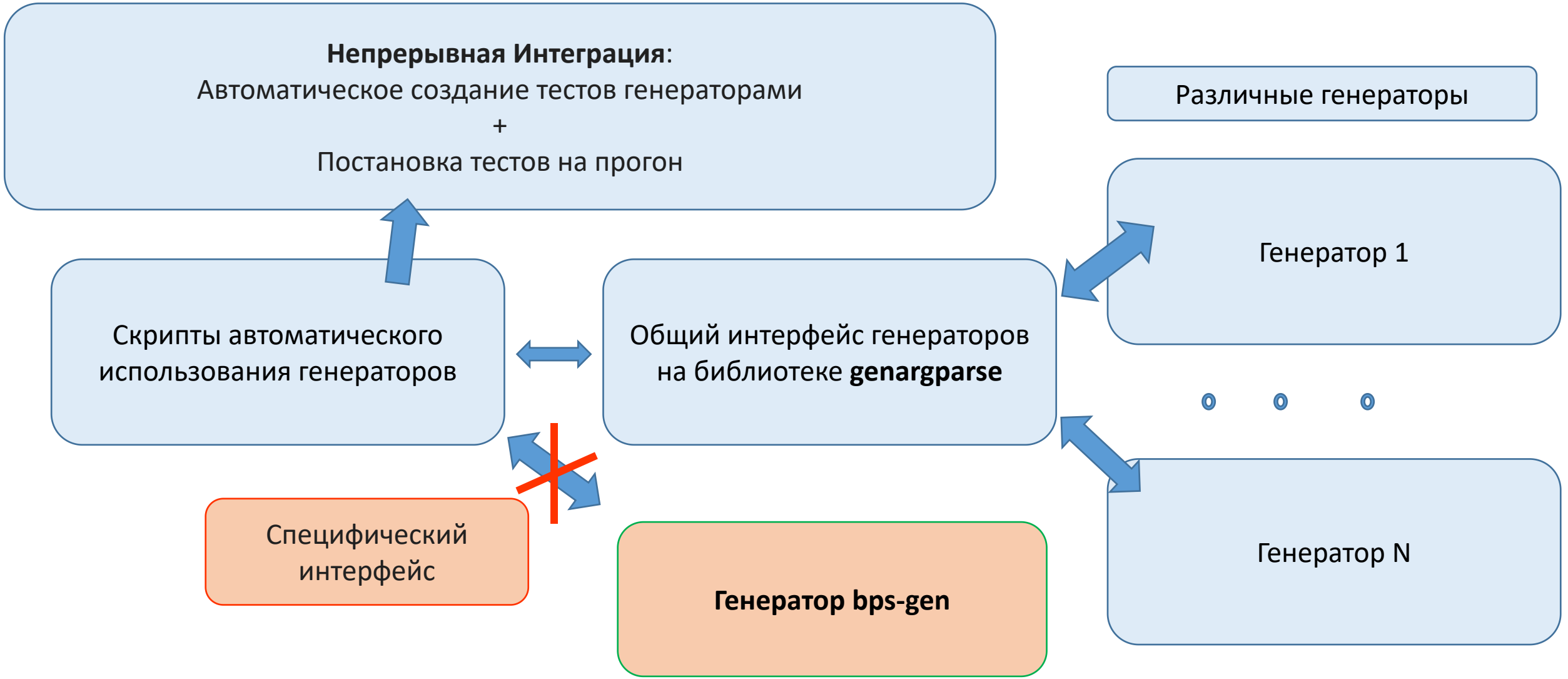
Развитие генератора тестов конвейера исполнения инструкций в микропроцессорах семейства Эльбрус

Недостатки генератора:

1. Специфический интерфейс строки запуска;

2. Организация проверок параметров запуска в коде программы.

Недостаток 1: Специфический интерфейс



Недостаток 1 : специфический интерфейс

часть специфических параметров и общие опции:
как ключи командной строки

часть специфических параметров:
через отдельный файл

```
balakh_f@irc-219 /export/data/verif/balakh_f/bps-old-trunk/run $ ../install/bin/bps_gener --help
BPS_GEN Options:
  -T [ --target ] arg (=8c2)           Processor version[4c, 8c, 1c+, 8c2,
                                       12c, 2c3]
  -W [ --wd_dbl_set ]                  Set wd_dbl flag
  -N [ --num_vliw ] arg (=1000)        Number of long instructions[min: 10
                                       max: 100000]
  -C [ --checkpoints_range ] arg (=20) Checkpoint spacing[min: 20]
  -S [ --seed ] arg (=42)              Seed generation
  --path-config arg                    Path to the configuration file
  -o [ --outfile_prefix ] arg (=bps_gen_test)
                                       Test file name prefix
  --path-share arg                     Path to the share dir
  -v [ --verbose ]                     Print what are doing now
  -V [ --version ]                     Program version
  -h [ --help ]                        Help screen
  -A [ --algorithm ] arg (=rd)         Algorithm[rd, dp]
  -M [ --chan_mask ] arg               Channel mask
  --trace                              Trace
  --nself_check                        Test without selfcheck
  --ntime_check                        Test without timecheck
  --loop                               Test with loop
  --lcount arg (=3)                    Loop iteration count(size of data array
                                       lcount * N)
  -m [ --model ] arg (=default)        Generation model: [default, custom].
                                       Custom is userlike test.
```

```
{
    "channel_1" : {
          "used_iset" : ["C7", "C1"],
          "unused_iset" : ["C7", "C1"],
          "instructions" : [
              {"mnem" : "add*",  "percent" : "10"},
              {"mnem" : "subs",  "percent" : "20"}

          ]
    },

    "channel_2" : {
          "instructions" : [
              {"mnem" : "sub*",  "percent" : "100"}
          ]
    }
}
```

параметры передаваемые генератору

Общие опции для всех генераторов

Специфические параметры для конкретного генератора

через ключи в строке запуска

через отдельный файл сценария

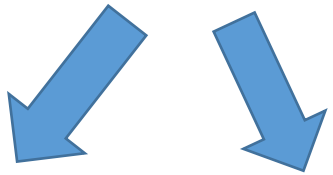
```
General Options:
  -h [ --help ]                         show help message and exit
  --doc arg                             show information about given
                                        and exit
  -s [ --seed ] arg                     initial random seed
  --version                             show version and exit
  --dest arg (="./")                    directory to store tests
  --keep-tmp                            keep temporary files
  --scenarios                           show available test generati
                                        scenarios and exit
  --share-path arg (="/export/data/verif/balakh_f/bps-trunk/install/
                                        share path (for debugging)
  --target-socket arg                   target socket
  --target-iset arg                     target iset
```

```
wd_dbl_set                      Set wd_dbl flag
num_vliw   arg (=1000)          Number of long instructions[min: 10
                                        max: 100000]
checkpoints_range   arg (=20)   Checkpoint spacing[min: 20]

path-share arg                  Path to the share dir

algorithm   arg (=rd)           Algorithm[rd, dp]
chan_mask   arg                 Channel mask
trace                           Trace
nself_check                     Test without selfcheck
loop                            Test with loop
lcount  arg (=3)                Loop iteration count(size of data array lo
```

Недостаток 2:организация проверки параметров запуска

Организация	Последствия	Недостатки
хранит данные частично в преобразованном к некоторому типу виде	тип внутри частично гарантирует корректность опции, частично за это отвечает код проверки	трудно контролировать как именно ответственность за корректность хранящейся опции разделяется между кодом проверки и типом, выбранным для хранения опции
содержит в себе коды проверки каждого параметра	возможно повреждение чужого кода проверки, при добавление новой опции с проверкой	код проверки другой опции будет трудно отлаживать, без знания предназначения этой опции
запускает эти коды одновременно при парсинге и загрузке хранлища	в случае отладки вам придётся проследить, как сложно указатель на ваш код проверки передаётся через класс option_description и вызывается вместе с другими	трудности отладки в случае нарушения работы проверки одной из опций

Задача разбилась на две:



1.Сформировать стандартный интерфейс запуска для использования генератора автоматической системой запуска

2. Реорганизовать хранение и проверку опций: с целью упращения отладки, повышения устойчивости к ошибкам и масштабируемости программы

Задача 1: формирование стандартного интерфейса

Скрипты автоматического использования генераторов

Общий интерфейс генераторов на библиотеке **genargparse**

Генератор bps-gen

Все специфические для генератора опции передаются в отдельном **файле сценария**

Общие опции **через ключи** команды запуска

```
General Options:
  -h [ --help ]                         show help message and exit
  --doc arg                             show information about given scenario
                                        and exit
  -s [ --seed ] arg                     initial random seed
  --version                             show version and exit
  --dest arg (="./")                    directory to store tests
  --keep-tmp                            keep temporary files
  --scenarios                           show available test generation
                                        scenarios and exit
  --share-path arg (="/export/data/verif/balakh_f/bps-trunk/install/share/bpsgen/0.2")
                                        share path (for debugging)
  --target-socket arg                   target socket
  --target-iset arg                     target iset
```

```
{
"doc" :{
    "target_isets" : ["5", "6"],
    "brief" : "rand_scenario"
},
"loop" : true ,
"lcount" : 2 ,
"target" : "2c3",
"wd_dbl_set" : true,
"num_vliw" : 1000,
"checkpoints_range" : 30,
"algorithm" : "rd",
"chan_mask" : 13,
"verbose" : false,
"nself_check" : "false",
"ntime_check" : "false",
"conf" : {
            "channel_1" : {
                  "used_iset" : ["C7", "C1"],
                  "unused_iset" : ["C7", "C1"],
                  "instructions" : [
                      {"mnem" : "add*",  "percent" : "10"},
                      {"mnem" : "subs",  "percent" : "20"}

                  ]
            },

            "channel_2" : {
                  "instructions" : [
                      {"mnem" : "sub*",  "percent" : "100"}
```

Использование Библиотеки genargparse

Загрузка дерева конфигурации
cfg -- дерево конфигурации; argc, argv -- параметры
содержащие строку запуска программы

```
int main (int argc, const char* argv [])
try
{
    LOG_INFO("START GENERATION", "my_dop_arg")
    boost::property_tree::ptree cfg;

    if (!genargparse::parse(argc, argv, cfg)) {
```

```
std::string tmp_wd_dbl_str = cfg.get("wd_dbl_set", std::string("false"));
```

Извлечение значения параметра из дерева конфигурации

Задача 2: реогранизация проверок опций и хранилища опций

1. Использование библиотеки genargparse

2. Вынос проверки опции за пределы класса

```
//vliw_number option initialization
std::size_t tmp_vliw_num;
boost::optional<std::size_t> opt_vliw_num = cfg.get_optional<std::size_t>("num_vliw");
if(!cfg.count("num_vliw"))
{
  tmp_vliw_num = 1000;
}
else if(!opt_vliw_num)
{
  throw  std::runtime_error(std::string("value of num_vliw is unconversiable to std::size_t")
}
else if((*opt_vliw_num) < bps_gen::c_min_inst__ || (*opt_vliw_num) >  bps_gen::c_max_inst__)
{
  throw  std::runtime_error(std::string("value of num_vliw is too big, or too small"));
}
else
{
  tmp_vliw_num = (*opt_vliw_num);
}
const std::size_t vliw_number    = tmp_vliw_num;
```

Задача 2: реогранизация проверок опций и хранилища опций

Недостатки	Измения	Результат
трудно контролировать как именно ответственность за корректность хранящейся опции разделяется между кодом проверки и типом, выбранным для хранения опции	хранилище НЕ контролирует то что передаётся ввиде опции, хранит только передаваемые строки ответственность за корректность опции берёт на себя код, распалагающейся между извлечением опции и её применением	Программу можно более контролируем расширять, добавляя в неё новые опции.
код проверки другой опции будет трудно отлаживать, без знания предназначения этой опции	код проверки НЕ связан никакой структурой с имеющимися кодами проверки других опций	Устойчивость к ошибкам: Добавление новых опций с проверками имеет меньше шанс повредить старые коды проверки
трудности отладки в случае нарушения работы проверки одной из опций	код проверки НЕ передаётся как указатель через структуру option_description, а явно располагается после извлечения опции	Лёгкость отладки: Код проверки проще отлаживать.

Проверка опций : как было

```cpp
int main(int argc, char** argv)
{
try {
    LOG_INFO("START GENERATION")

    bps_gen::Option_parser user_options;

    user_options.init(argc, argv);
```

```cpp
void
Option_parser::init(int argc, char** argv)
{
    po::store(po::parse_command_line(argc, argv, desc__), init_options__);
    po::notify(init_options__);
}
```

```cpp
Option_parser::Option_parser()
{
    Declare_opt();
}
```

Класс Option_parser -- реализует извлечение, хранение и **проверку** опций

Почти **одновременная** проверка опций при создании хранилища опций

```cpp
void
Option_parser::Declare_opt()
{
    desc__.add_options()
      ("target,T", po::value<std::string>()->notifier(std::bind(&Option_parser::check_opt_target__, this, std::placeholders::
ult_value(c_ver_8c2_def__),
       (std::string("Processor version[") + c_ver_4c_def__ + std::string(", ") + c_ver_8c_def__ + std::string(", ") +
           c_ver_1cp_def__ +  std::string(", ") + c_ver_8c2_def__ + std::string(", ") +
           c_ver_12c_def__ +  std::string(", ") + c_ver_2c3_def__ + "]").c_str())
      ("wd_dbl_set,W", "Set wd_dbl flag")
      ("num_vliw,N", po::value<std::size_t>()->default_value(1000)->notifier(std::bind(&Option_parser::check_opt_num_vliw__,
:placeholders::_1)),
           (std::string("Number of long instructions[min: ") + std::to_string(c_min_inst__) + std::string(" max: ") + std::to
max_inst__) +"]").c_str())
      ("checkpoints_range,C", po::value<std::size_t>()->default_value(20)->notifier(std::bind(&Option_parser::check_opt_check
ge__, this, std::placeholders::_1)),
```

Проверка опций : как было : проблемы

Программист будущий внедряет новую функциональность в генератор

Программист знает функциональность которую добавляет => знает суть проверки **новой опции**

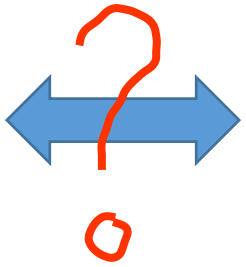
Для новой функциональности требуется:
новая опция

Программист не знает другую функциональность => не знает суть проверки **другой опции**

проверка **новой опции** встраивается в класс: приходится протаскивать ограничения на опцию и их проверку в класс

Если что-то сломается?

В классе есть **другая опция** другой функциональности



Класс Option_parser

Результат работы

1. Интерфейс генератора приведен к унифицированному в отделе виду, что позволяет включить его в общий цикл автоматического использования.

2. Реорганизовано хранение и проверка опций: использована общая в отделе библиотек genargparse,
программу теперь легче расширять, она более устойчива к ошибкам и её проще отлаживать.