Инструкция по загрузке, настройке и логированию GPU-BURN

0. Prelude - Подготовка

```
# Установка требуемых пакетов
sudo apt update
sudo apt install -y git g++ make htop nvidia-cuda-toolkit
# Если NVIDIA драйвер не установлен:
# sudo apt install -y nvidia-driver-580-open
```

```
# Загрузка gpu-burn с GitHub
cd ~
# ORIGINAL GPU-BURN:
git clone https://github.com/wilicc/gpu-burn.git
# WITH RU COMMENTS GPU-BURN:
# git clone https://github.com/Br41nfck/gpu-burn.git
```

1. Setup - Настройка GPU-BURN

Настройка Makefile

Узнаём версию CUDA:

```
> nvcc -V
```

Пример вывода:

```
user@linux_test_machine:~$ nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2023 NVIDIA Corporation
Built on Fri_Jan__6_16:45:21_PST_2023
Cuda compilation tools, release 12.0, V12.0.140
Build cuda_12.0.r12.0/compiler.32267302_0
```

Искомая строка:
`Cuda compilation tools, release 12.0, V12.0.140`

Отсюда берём версию CUDA (переменная CUDA_VERSION): `12.0.140`
(БЕЗ УКАЗАНИЯ V!)

Узнаём какие архитектуры поддерживает CUDA:

```
nvcc --list-gpu-code
```

Далее из приведённой ниже таблицы указываем вычислительную возможность, в зависимости от архитектуры GPU (имеется в названии) и поддержки CUDA (переменная COMPUTE):

Архитектура GPU	Вычислительные возможности
Blackwell	sm_100, sm_100a, sm_101, sm_101a, sm_120, sm_120a
Hopper	sm_90, sm_90a
Ada	sm_89
Ampere	sm_80, sm_86, sm_87 (Orin)
Turing	sm_75
Volta	sm_70, sm_72 (Xavier)
Pascal	sm_60, sm_61, sm_62
Maxwell	sm_50, sm_52, sm_53
Kepler	sm_30, sm_35, sm_37
Fermi	sm_20

По итогу, имеем 2 значения (COMPUTE, CUDA_VERSION), которые нужно заменить в файле Makefile:

Пример

Возьмём COMPUTE = 90 и CUDA_VERSION = 12.0.140

Значения по умолчанию:

```
COMPUTE      ?= 50
CUDA_VERSION ?= 11.8.0
```

Можно заменить через `nano` :

```
nano Makefile
```

Или вставить команду:

```
# Заменяем COMPUTE ?= 50 на 90 (пример)
sed -i 's/^\(COMPUTE[[:space:]]*?=[[:space:]]*\)50$/\190/'
Makefile

# Заменяем CUDA_VERSION ?= 11.8.0 на 12.0.140 (пример)
sed -i 's/^\(CUDA_VERSION[[:space:]]*?=
[[:space:]]*\)11\.8\.0$/\112.0.140/' Makefile
```

Настройка gpu_burn-drv.cpp

Далее нужно внести изменения в файле gpu_burn-drv.cpp:

Заменить:

```
GPU_Test(int dev, bool doubles, bool tensors, const char
*kernelFile)
        : d_devNumber(dev), d_doubles(doubles),
d_tensors(tensors), d_kernelFile(kernelFile){
        checkError(cuDeviceGet(&d_dev, d_devNumber));
        checkError(cuCtxCreate(&d_ctx, 0, d_dev));

        bind();

        // checkError(cublasInit());
        checkError(cublasCreate(&d_cublas), "init");

        if (d_tensors)
            checkError(cublasSetMathMode(d_cublas,
CUBLAS_TENSOR_OP_MATH));

        checkError(cuMemAllocHost((void **)&d_faultyElemsHost,
sizeof(int)));
        d_error = 0;

        g_running = true;

        struct sigaction action;
        memset(&action, 0, sizeof(struct sigaction));
        action.sa_handler = termHandler;
        sigaction(SIGTERM, &action, NULL);
    }
```

На следующий:

```
GPU_Test(int dev, bool doubles, bool tensors, const char
*kernelFile)
        : d_devNumber(dev), d_doubles(doubles),
d_tensors(tensors), d_kernelFile(kernelFile){
```

```
        checkError(cuDeviceGet(&d_dev, d_devNumber));
        checkError(cuDevicePrimaryCtxRetain(&d_ctx, d_dev));
        checkError(cuCtxSetCurrent(d_ctx));

        bind();

        // checkError(cublasInit());
        checkError(cublasCreate(&d_cublas), "init");

        if (d_tensors)
            checkError(cublasSetMathMode(d_cublas, CUBLAS_TENSOR_OP_MATH));

        checkError(cuMemAllocHost((void **)&d_faultyElemsHost, sizeof(int)));
        d_error = 0;

        g_running = true;

        struct sigaction action;
        memset(&action, 0, sizeof(struct sigaction));
        action.sa_handler = termHandler;
        sigaction(SIGTERM, &action, NULL);
}
```

Или командой sed:

```
sed -i '/GPU_Test(int dev, bool doubles, bool tensors, const char \*kernelFile)/,/^    }/c\
GPU_Test(int dev, bool doubles, bool tensors, const char *kernelFile): d_devNumber(dev), d_doubles(doubles), d_tensors(tensors), d_kernelFile(kernelFile) {\
checkError(cuDeviceGet(&d_dev, d_devNumber));\
/\=\=\=  Инициализация CUDA-контекста \=\=\=/;\
checkError(cuDevicePrimaryCtxRetain(&d_ctx, d_dev));\
checkError(cuCtxSetCurrent(d_ctx));\
bind();\
/\ checkError(cublasInit\(\)\);/;\
checkError(cublasCreate(&d_cublas), "init");\
if (d_tensors)\
checkError(cublasSetMathMode(d_cublas, CUBLAS_TENSOR_OP_MATH);\
checkError(cuMemAllocHost((void **)&d_faultyElemsHost, sizeof(int)));\
d_error = 0;\
g_running = true;\
struct sigaction action;\
memset(&action, 0, sizeof(struct sigaction));\
action.sa_handler = termHandler;\
sigaction(SIGTERM, &action, NULL);\
```

```
}'
```

Сохраняем файлы

Компиляция и запуск GPU-BURN

Переходим в директорию gpu-burn:
`cd ~/gpu-burn`

Удаляем старые файлы:

```
rm -f *.ptx *.o
make clean
```

Собираем программу:

```
make
```

Запускаем программу для теста на исправность:

```
# -m 100% - 100%-ное использование памяти GPU
# 300 - длительность теста в секундах (5 минут)
./gpu-burn -m 100% 300
```

Если всё хорошо, переходим на следующий этап

2. Run - Скрипт для GPU-BURN

Далее используем скрипт start_gpuburn.sh (в папке gpuburn):

```
# Длительность теста можно изменить изменив значение TEST_DURATION (в секундах) в файле start_gpuburn.sh
# nano start_gpuburn.sh

# Даём права на запуск скрипта
sudo chmod +x
# Запускаем
sh ./start_gpuburn.sh
```

В домашней папке будет создана папка:
`/home/user/gpu_burn_logs`

В которой будут лежать файлы вида:

```
```

```
gpu_burn_<Дата_время>.log
nvidia-smi_<Дата_время>.log
```

Просмотреть содержимое файла можно командой:
`cat <Имя_файла>`

Автоматически отслеживать файл на наличие нового содержимого:
`tail -f <Имя_файла>`

3. Parse - Парсим логи

Файл `nvidia-smi_<Дата_время>.log` будет содержать таблицы из вывода команды `nvidia-smi`, что не удобно для анализа, поэтому нужно запустить скрипт `nvidia-smi-table-parser.py` (лежит в папке gpuburn, актуальная версия скрипта находится по адресу: GitLab: Linux Scripts), который распарсит логи в удобный формат .xlsx (Excel).

Для корректной работы скрипта, нужно установить требуемые библиотеки:

```
pip3 install pandas openpyxl
```

Файл для парсинга и скрипт, должны быть в одной и той же папке:

```
# Если файл находится в папке `/home/user/gpu_burn_logs`
# то перемещаем скрипт в эту папку и запускаем его
# из папки `/home/user/gpu_burn_logs`
cp nvidia-smi-table-parser.py /home/user/gpu_burn_logs
# Или наоборот
```

Далее, запускаем скрипт указывая версию python, python-файл и файл для парсинга:

```
# Пример запуска скрипта nvidia-smi-table-parser.py:
python3 nvidia-smi-table-parser.py nvidia-smi_<Дата_время>.log
```

На выходе получаем файл: `gpu_metrics.xlsx`

Открыть можно через Excel для Windows или OpenOffice на Linux.

4. Copy - Копирование папки на локальный/сетевой диск

Если копирование на Windows:
Win + R → powershell

```
# Копирование файлов рекурсивно на локальный ПК через ssh
pscp.exe -r user@192.168.50.62:/home/user/Logs e:\Logs\
```

где:
`-r` - рекурсивное копирование файлов (самой папки)
`user` - имя пользователя
`192.168.50.62` - IP-адрес ПК
`:/home/user/Logs` - папка на удалённом ПК
`e:\Logs\` - папка на локальном ПК (должна быть создана)

Если копирование на Linux:

```
# Копирование файлов рекурсивно на локальный ПК через ssh
scp -r /home/user/Logs user@192.168.50.62:/e/Logs
```

где:
`-r` - рекурсивное копирование файлов (самой папки)
`user` - имя пользователя
`192.168.50.62` - IP-адрес ПК
`/home/user/Logs` - папка на локальном ПК
`:e/Logs` - папка на удалённом ПК (должна быть создана)