0.1 Методика

Первым делом нужно произвести обновление или установку следующих модулей и библиотек

- autoconf Утилита для создания скриптов настройки (конфигурации)
- build-essential позволяет скомпилировать и установить программное обеспечение из исходного кода
- ca-certificates это пакет сертификатов, необходимых для установки и использования надежных сертификатов корневых УЦ (Сертификационных центров), которые используются в различных приложениях и службах в системе Linux
- gfortran это компилятор языка программирования Fortran, который является частью свободно распространяемого пакета GCC
- libblas3 библиотека низкоуровневых алгоритмов линейной алгебры, которая используется для ускорения вычислений в различных приложениях
- libc6 Она является одной из самых важных системных библиотек в Linux, так как содержит функции, необходимые многим приложениям для работы с файловой системой, сокетами, процессами и т.д
- libfftw3-dev это пакет разработчика, содержащий заголовочные файлы, библиотеки и другие файлы, необходимые для создания программ, использующих библиотеку FFTW. Которая в свою очередь является одной из самых быстрых и наиболее известных библиотек для быстрого преобразования Фурье
- libgcc-s1 это библиотека, которая является частью GCC - GNU Compiler Collection и содержит поддержку низкоуровневых функций, используемых при компиляции программ на языке C и C++
- liblapack-dev LAPACK - это высокопроизводительная библиотека, осуществляющая численные операции над матрицами, такие как решение систем линейных уравнений, вычисление собственных значений и векторов, а также решение задач линейной алгебры

Это нужно так как эти библиотеки и программы необходим Quantum Espresso для сборки. Далее стоит установить модули, что потребуются для параллельных вычислений, что очень необходимо, ведь вычисления бенчмарков требует огромного количества ресурсов, а без парализации получить достаточно большую мощность сложно.

Следующим шагом скачиваем программный пакет из официального репозитория разработчик программы GitHub

```
wget https://gitlab.com/QEF/q-e/-/archive/qe-7.2/q-e-qe-7.2.tar.gz
```

Распаковываем архив

```
tar -zxvf q-e-qe-7.2.tar.gz
```

Заходим в появившуюся папку

```
cd q-e-qe-7.2
```

Далее следует весьма важный этап, конфигурации пакета

```
OPTFLAGS="-O3 -march=native -DNDEBUG"
./configure FFLAGS="$OPTFLAGS" CFLAGS="$OPTFLAGS" \
                                --enable-parallel
```

Смысл этой конфигурации заключается в том, что первая строчка задаёт оптимизацию для компилятора. -O3 указывает, что нужно использовать максимальный уровень оптимизации, чтобы программа была максимально быстро действенна. Этот уровень оптимизации удаляет повторяющиеся операции, убирает неиспользуемый код и существенно уменьшает размер генерируемого исполняемого файла. Следующий флаг -march=native делает так, чтобы оптимизация учитывала архитектуру конкретного процессора. -DNDEBUG определяет макросы компиляции, связанные с выводом отладочной информации чтобы этот вывод отключить. В следующей строке мы присваиваем эти флаги языкам Фортран и С. И последний флаг –enable-parallel разрешает использование параллельных вычислений в программе, тем самым ускоряя её.

```
make pw
make install
```

Этими командами запускаем установку основной программы пакета, а затем и все остальные вместе с ней.

Программный пакет установлен, но нам ещё потребуется пара инструментов. Первый это проектировщик perf скучивающийся следующей командой

```
apt-get install linux-tools-common \
    linux-tools-generic linux-tools-‘uname -r‘
```

Эта команда устанавливает инструменты производительности Linux, такие как perf, trace-cmd, ftrace и др. для текущей версии ядра Linux. Далее

```
git clone git://perfmon2.git.sourceforge.net/gitroot/perfmon2/libpfm4
cd libpfm4
make
```

С помощью этих команд производится скачивание из Git репозитория библиотеки libpfm4 позволяющее просматривать коды ивентов процессора. Когда скачивание и установка завершиться, нужно перейти в папку exampls и запустить showevtinfo. Программа возвращает текстовый документ внутри которого перечислено множество различных событий процессора, среди них нудно найти те, что связаны с операциями с плавающей точке Получив выбрав имя и маску. В этой же директории запускам

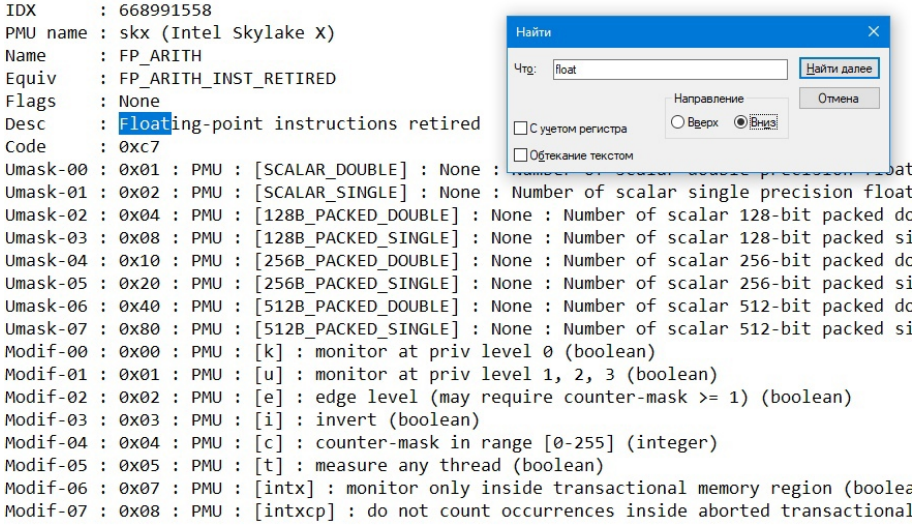

Рис. 0.1: Поиск событий на floating-point

программу с выбранной строкой

```
./check_events FP_ARITH:SCALAR_SINGLE
```

Это лишь пример выбора имени и строки. Результатом действия программы будем информация о событие и среди этой информации можно будет найти строку Codes, где и находится нужная мне информация о событии.

Практически вся подготовительная работа выполнена. Осталось загрузить бенчмарк.

```
wget https://github.com/QEF/benchmarks/tree/master/AUSURF112
```

После этого папка с бенчмарком появится в папке, где вы располагаетесь. Осталось написать Slurm скрипт. Основные параметры таковы

- ntasks – число процессов
- cpus-per-task – количество ядер CPU выделяемых на процесс
- gpus – количество GPU
- nodes – Минимальное выделяемое количество узлов
- ntasks-per-node – количество процессов на каждый узел
- gpus-per-task – количество GPU на каждый запускаемый процесс
- gpus-per-node – количество GPU на каждый выделяемый узел

```
#!/bin/bash
#SBATCH --job-name="QuantumEspresso-CPU"
#SBATCH --ntasks=16
#SBATCH --cpus-per-task=1
#SBATCH --nodes=1

module load openmpi
module load  fftw/v3.3.10

perf stat -e r5301c7 mpirun /home/ienedomolkin/newQE/q-e-qe-7.2/bin/pw.x -i ausurf.in
```

Рис. 0.2: Пример запуска задачи

Теперь делая запуски на разных предпочтительных узлах. Это делается чтобы проанализировать работу выбранного объекта. Так как задача состоит в том, чтобы качественно протестировать данные, нужно учитывать тот факт, что в процессоре количество физических ядер не всегда равно количество вычислительных. Поэтому гипотетически может произойти такая ситуация, что на одном физическом ядре, в одном из его логических ядер будет происходить решение твоей задачи, а на другом другой задачи. Это может оказать некоторое влияние на получаемый результат. Поэтому чтобы минимизировать возможные влияния ограничим возможности CPU и скажем, что каждое ядро будет обрабатывать только один процесс.

Получив данные, нужно провести анализ масштабируемости задачи. Как сильно распараллеливание влияет на скорость вычислений. Это может ответить на вопрос целесообразности увеличения распараллеливания, так как в один момент затраты на разделение процесса могут стать больше нежели ускорение от этого. Поэтому для некоторых задач будет не эффективно использование нескольких вычислительных узлов.

Так же стоит проанализировать данные о количестве операций с плавающей точкой. Сравнив эти показатели с номинальными процессора может узнать насколько эффективно работает программный пакет. Хотя конечно не только количество операций с плавающей точкой решает, что лучше. Следующим шагом, опираясь на стоимость оборудования стоит узнать, за какую цену можно купить единицу мощности.

Итоговый ответ даётся на основании количества операций с плавающей точкой в секунду и собственной самого маленького времени вычислений, но и стоит упомянуть стоимость единицы мощности на эффективность, так как некоторые люди основывают свой выбор основываясь на этом пункте.