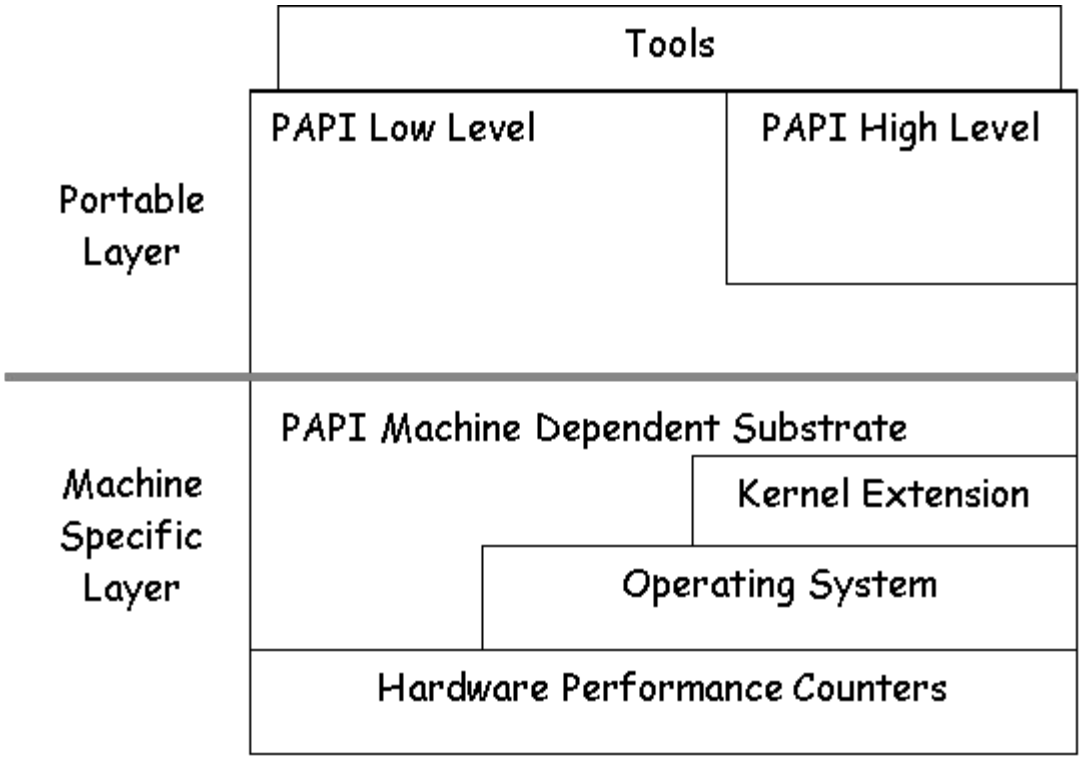
Средства и системы параллельного программирования

Семинар #3. PAPI

PAPI



“ PAPI attempts to report per-process values, but there are a number of ways in which these might vary from what you would expect “

Установка и использование PAPI

https://github.com/icl-utk-edu/papi/releases/tag/papi-7-2-0b1-t - релиз последней версии

cd papi-7.2.0b1/

Установка: (подробный мануал лежит в INSTALL.txt)

./configure
make
make install (*опционально*)

gcc prog.c -lpapi -o prog -I/path/to/papi -L/path/to/papi

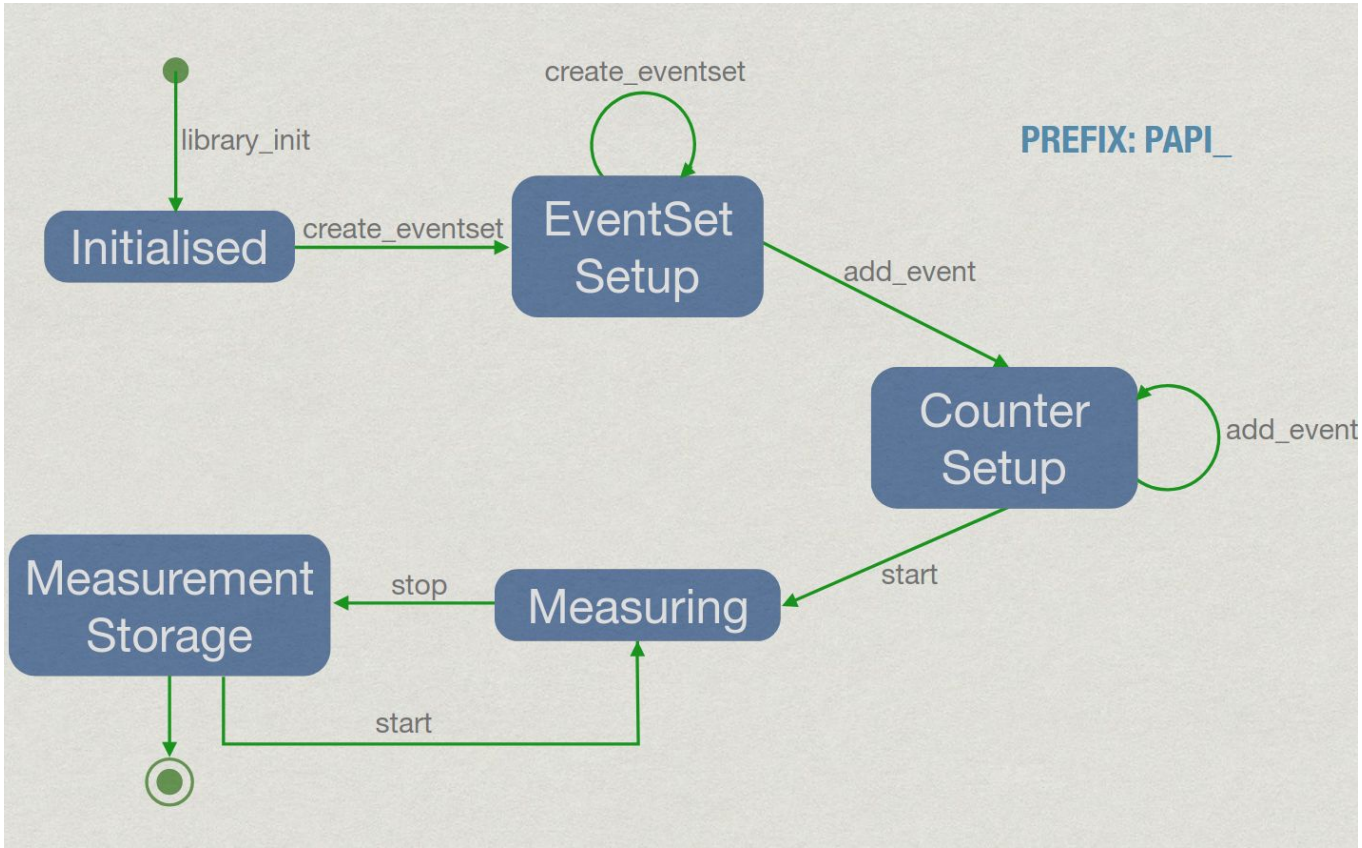
Paranoid

```
config.status: creating config.h
config.status: executing genpapifdef commands
configure: WARNING:

*******************************************************************************
* Insufficient permissions for accessing any hardware counters.               *
* Your current paranoid level is 4.                                           *
* Set /proc/sys/kernel/perf_event_paranoid to 2 (or less) or run as root. *
*                                                                             *
* Example:                                                                    *
* sudo sh -c "echo 2 > /proc/sys/kernel/perf_event_paranoid"                  *
*******************************************************************************
```

sudo sysctl kernel.perf_event_paranoid=-1

Общая структура Low-level PAPI



Native & Preset events

Native events - события, определенные архитектурой процессора

Preset events - события, введенные в интерфейсе PAPI и каким-либо образом использующие Native events

papi_avail (papi_native_avail) - просмотр доступных preset (native) событий

papi_avail -e SOME_EVENT - детальный просмотр события

papi_decode - утилита для детального просмотра всех preset событий

papi_decode -a - то же самое, но только для доступных событий

Требуемые функции при работе с low-level API

```
1.  int PAPI_library_init(int version); // инициализация библиотеки

2.  int PAPI_create_eventset (int *EventSet); // инициализация EventSet

3.  int PAPI_add_event(int *EventSet, int EventCode);
    int PAPI_add_events(int *EventSet, int *EventCode, int number); // добавление Event в EventSet

4.  int PAPI_start(int EventSet); // начало подсчёта event в EventSet

5.  int PAPI_read(int EventSet, long_long *values); // чтение текущих значений с counters в values
    int PAPI_accum(int EventSet, long_long *values);// то же самое, но с прибавлением

6.  int PAPI_stop(int EventSet, long_long *values); // остановка чтения, чтение значений с counters в values

7.  int PAPI_reset (int EventSet); //очистка текущих значений в counters

8.  int PAPI_cleanup_eventset(int *EventSet); // удаление всех event из EventSet

9.  int PAPI_destroy_eventset(int *EventSet); // освобождение данных, выделенных под EventSet. Можно вызывать
    только для пустого EventSet

10. int PAPI_shutdown(); // освобождение всех ресурсов, выделенных PAPI
```

Работа с событиями

- `int PAPI_event_code_to_name(int EventCode, char *EventName);`
- `int PAPI_event_name_to_code(char *EventName, int *EventCode);`
- `int PAPI_query_event(int EventCode); // проверка возможности сбора данного Event на данной архитектуре (0 = OK)`

В качестве EventName можно использовать и Preset и Native events

High-level API

```
int PAPI_flops_rate(int event, float *rtime, float *ptime, long long
* flpops, float *mflops);
```

```
// Составная PAPI-команда, позволяющая снять с разработчика
работу с Event
```

*rtime -- total realtime since the first PAPI_flop_rate() call

*ptime -- total process time since the first PAPI_flop_rate() call

*flpins -- total floating point instructions since the first PAPI_flop_rate() call

*mflops -- Mflops/s achieved since the latest PAPI_flop_rate() call

Многопоточность & PAPI

```
int PAPI_thread_init (unsigned long int (*handle)(), int flag); //инициализация поддержки многопоточности, вызывать сразу после library_init
```

в Pthreads: PAPI_thread_init(pthread_self, 0);

```
int PAPI_register_thread (void); //внутри функции, выполняющейся в потоке

int PAPI_unregister_thread (void); //не собирать PAPI для этого потока
```

Overflow

```
int PAPI_overflow (int EventSet, int EventCode, int threshold, int flags, PAPI_overflow_handler_t handler); // возможность выполнить действие, прописанное в handler при достижении числа событий Event установленного threshold
```

```
typedef void (*PAPI_overflow_handler_t) (int EventSet, void *address, long long overflow_vector, void *context);
```

Вывод papi_avail

```
dlichman@dimPC:~$ papi_avail
Available PAPI preset and user defined events plus hardware information.
--------------------------------------------------------------------------------
PAPI version             : 7.2.0.0
Operating system         : Linux 5.8.0-59-generic
Vendor string and code   : GenuineIntel (1, 0x1)
Model string and code    : 11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz (140,
0x8c)
CPU revision             : 1.000000
CPUID                    : Family/Model/Stepping 6/140/1, 0x06/0x8c/0x01
CPU Max MHz              : 4700
CPU Min MHz              : 400
Total cores              : 8
SMT threads per core     : 2
Cores per socket         : 4
Sockets                  : 1
Cores per NUMA region    : 8
NUMA regions             : 1
Running in a VM          : no
Number Hardware Counters : 19
Max Multiplex Counters   : 384
Fast counter read (rdpmc): yes
--------------------------------------------------------------------------------
```

Мультиплексирование

С помощью мультиплексирования реализуется возможность сбора большего числа событий, чем числа counters

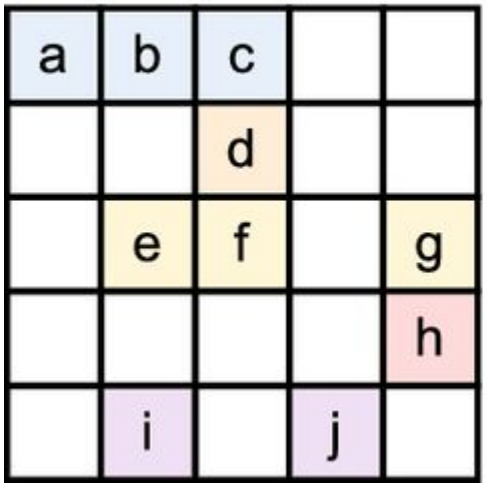
papi_multiplex_cost - утилита для просмотра накладных расходов от использования мультиплексирования на выбранной архитектуре

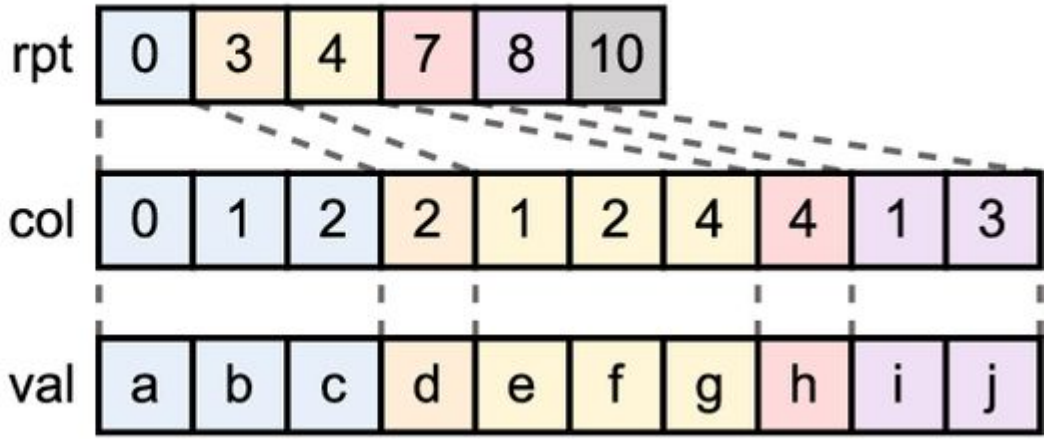
```
int PAPI_multiplex_init (void); // инициализация мультиплексирования

int PAPI_set_multiplex(int *EventSet); // настройка EventSet на мультиплексирование
```

Задание

Граф в формате CSR





Задание #3

Реализовать следующие алгоритмы работы с графом в формате CSR:

1) Определение вершины с наибольшим суммарным весом инцидентных ребер, ведущих к вершинам с чётными номерами.
2) Определение вершины с наибольшим рангом, где ранг считается по формуле:

$$Rank(vertex) = \sum_{i=0}^{N\_inc\_edges} w_{edge_i} * W_{vert_i},$$

где N_inc_edges - число инцидентных вершине $vertex$ рёбер, w_{edge_i} - вес i-го ребра, а вес вершины W_{vert_i} определяется по формуле

$$W(vertex) = \sum_{j=0}^{N\_inc\_edges} w_{edge_j} * N\_inc\_edges_{vert_j}$$

Запрещается явное хранение W в виде какой-либо структуры или массива

Подсчитать и сравнить на двух алгоритмах показатели событий PAPI_L1_TCM, PAPI_L2_TCM (при наличии), также подсчитать любой native event на выбор. Составить *небольшой* отчет.

Набор тестовых графов в CSR будет предоставлен.

Дедлайн: (ИЗМЕНЕНО) 8.10, 20.10

Большая формула и tex

$$Rank(vertex) = \sum_{i=0}^{N\_inc\_edges} w_{edge_i} * W_{vert_i},$$

где N_inc_edges - число инциндентных вершине $vertex$ рёбер, w_{edge_i} - вес i-го ребра, а вес вершины W_{vert_i} определяется по формуле

$$W(vertex) = \sum_{j=0}^{N\_inc\_edges} w_{edge_j} * N\_inc\_edges_{vert_j}$$

$$Rank(vertex) = \sum _{i = 0} ^{N\_inc\_edges} w_{edge_{i}} * W_{vert_{i}} ,$$

где \(N_inc_edges\) - число инциндентных вершине \(vertex\) рёбер, \(w_{edge_{i}}\) - вес \(i\)-го ребра, а вес вершины \(W_{vert_{i}}\) определяется по формуле

$$W(vertex) = \sum _{j = 0} ^{N\_inc\_edges} w_{edge_{j}} * N\_inc\_edges_{vert_{j}} $$