**ОТЧЕТ ПО ПРАКТИКЕ**

тема: «Оптимизация анализа генома на базе программы picard-tools»

Выполнила

студентка гр.53504/2

И.И. Закирова

Руководитель

З.Д.Льянов

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2015 г.

# Предметная область

Данная работа посвящена исследованию алгоритмов, применяемых в биоинформатике. Для получения базы знаний в это предметной области был пройден курс “Введение в биоинформатику”[1] от Санкт-Петербургского Государственного университета. Задание было получено и выполнялось в компании EPAM.

# Постановка задачи

В представленной работе необходимо оптимизировать класс EstimateLibraryComplexity в инcтрументе Picard Tools[2]. Программа Picard Tools разработана на языке Java в Broad Institute и представляет собой набор инструментов командной строки для анализа генома. Входной геном хранится в bam-файле. Алгоритмы по расчету метрик организованы в классы. Каждая метрика может запускаться отдельно и использовать различные входные параметры. Для класса EstimateLibraryComplexity необходимо задать входной bam-файл, и выходной файл, референсный геном для расчетов не требуется.

# Ход работы

Алгоритм условно можно разбить на следующие части:

1. Чтение входной информации и ее первичный анализ
2. Сортировка входных данных
3. Последовательный проход по отсортированным данным, расчет метрик
4. Запись в результирующий файл

В данной работе оптимизации были подвергнуты две первые части. На этих этапах введена многопоточность. Для этого добавлен следующий код:

**final** ExecutorService service = Executors.*newFixedThreadPool*(NUMBER\_OF\_THREADS);

**final** BlockingQueue<List<Object[]>> queue = **new** LinkedBlockingQueue<List<Object[]>>(

QUEUE\_CAPACITY);

**final** Semaphore sem = **new** Semaphore(NUMBER\_OF\_THREADS+1);

**final** List<SortingCollection<PairedReadSequence>> sorters = **new** ArrayList<SortingCollection<PairedReadSequence>>(NUMBER\_OF\_THREADS);

**for**(**int** i=0; i<NUMBER\_OF\_THREADS; i++){

**final** **int** k=i;

service.execute(**new** Runnable() {

@Override

**public** **void** run() {

SortingCollection<PairedReadSequence> sorter = SortingCollection.*newInstance*(PairedReadSequence.**class**,

**new** PairedReadCodec(),

**new** PairedReadComparator(),

(**int**)(MAX\_RECORDS\_IN\_RAM\*(1.0+0.20\*k)),

TMP\_DIR);

**while** (**true**) {

**try** {

**final** List<Object[]> tmpPairs = queue.take();

**if**(tmpPairs.isEmpty()){

sorters.add(sorter);

log.info("Thread is finished.");

**return**;

}

sem.acquire();

**for** (Object[] objects : tmpPairs) {

PairedReadSequence prs = (PairedReadSequence) objects[0];

SAMRecord rec = (SAMRecord) objects[1];

// Read passes quality check if both ends meet the mean quality criteria

**final** **boolean** passesQualityCheck = passesQualityCheck(rec.getReadBases(),

rec.getBaseQualities(),

MIN\_IDENTICAL\_BASES,

MIN\_MEAN\_QUALITY);

prs.qualityOk = prs.qualityOk && passesQualityCheck;

// Get the bases and restore them to their original orientation if necessary

**final** **byte**[] bases = rec.getReadBases();

**if** (rec.getReadNegativeStrandFlag()) SequenceUtil.*reverseComplement*(bases);

**if** (rec.getFirstOfPairFlag()) {

prs.read1 = bases;

} **else** {

prs.read2 = bases;

}

**if** (prs.read1 != **null** && prs.read2 != **null** && prs.qualityOk) {

sorter.add(prs);

}

progress.record(rec);

}

sem.release();

}**catch** (InterruptedException e1) {

e1.printStackTrace();

}

}

}

});}

…

**new** Thread(**new** Runnable() {

**public** **void** run() {

**final** Map<String, PairedReadSequence> pendingByName = **new** HashMap<String, PairedReadSequence>();

readGroups.addAll(in.getFileHeader().getReadGroups());

List<Object[]> pairs = **new** ArrayList<Object[]>(MAX\_PAIRS);

**while** (**true**) {

**try** {

**final** List<SAMRecord> tmpRecords = queueRec.take();

….//Анализ входных данных. Заполнение очереди прочтениями из входного файла и парами ридов

}**catch** (InterruptedException e1) {

e1.printStackTrace();

}}}

}).start();

В результате чтение обработка и сортировка входных данных имеет следующий вид:

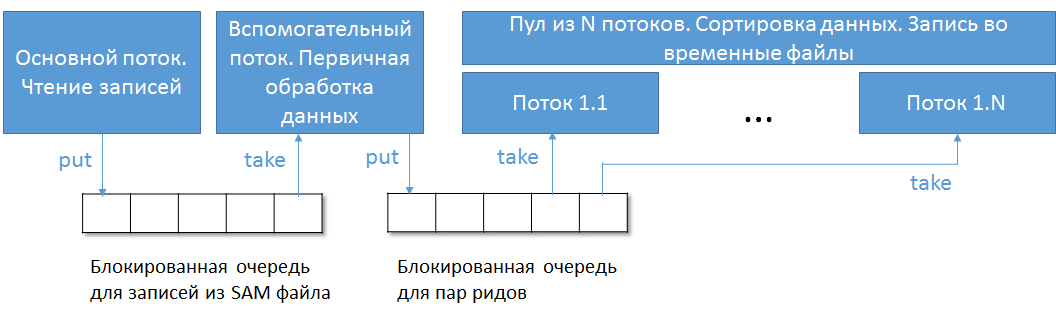


Рисунок 1 Модифицированная схема чтения записей и сортировки ридов

# Результаты

Испытания проводились на файле other.bam. До модификаций расчет метрики EstimateLibraryComplexity выполнялся за 3 минуты 37 секунд. После распараллеливания алгоритма на первых этапах была проведена серия испытаний с различным количеством потоков для сортировки данных. Результаты представлены в Табл.1

Таблица 1

|  |  |  |
| --- | --- | --- |
| NUMBER\_OF\_THREADS | Time(min) | Time(sec) |
| 1 | 3 | 3 |
| 2 | 2 | 50 |
| 3 | 2 | 38 |
| 4 | 2 | 28 |
| 5 | 2 | 44 |
| 6 | 2 | 49 |

Наиболее эффективным является создание 4х потоков для сортировки данных (в рамках исследуемого оборудования).

Также были проведены испытания с разным размером блоков, помещаемых в очередь. Их длина измеряется в количестве записей или пар ридов.

|  |  |  |
| --- | --- | --- |
| MAX\_PAIRS | Time(min) | Time(sec) |
| 50 | 2 | 67 |
| 100 | 2 | 57 |
| 500 | 2 | 24 |
| 1000 | 2 | 32 |
| 5000 | 2 | 60 |

Влияние параметра MAX\_PAIRS не так велико, как влияние числа потоков, однако тоже сказывается на производительности. Наилучших результатов удалось добиться при значении MAX\_PAIRS=500.

Таким образом с помощью преобразований удалось ускорить расчет метрики приблизительно на 30%.

Во время контрольного запуска метрики были также сняты показания с динамического анализатора Visual VM

# Вывод

В результате работы были оптимизирован расчет метрики EstimateLibraryComplexity. Расчет метрики ускорился приблизительно на 30%.