



Средства распараллеливания в Java 1.7 (jsr166у...)

Михаил Пономаренко,
Tech Lead компании Sigma Ukraine



немного истории

- до java 1.5 были
 - wait
 - notify
 - synchronized
- в 1.5 - jsr166
 - java.util.concurrent
 - Future, ThreadExecutor
 - ConcurrentHashMap - асинхронные итераторы

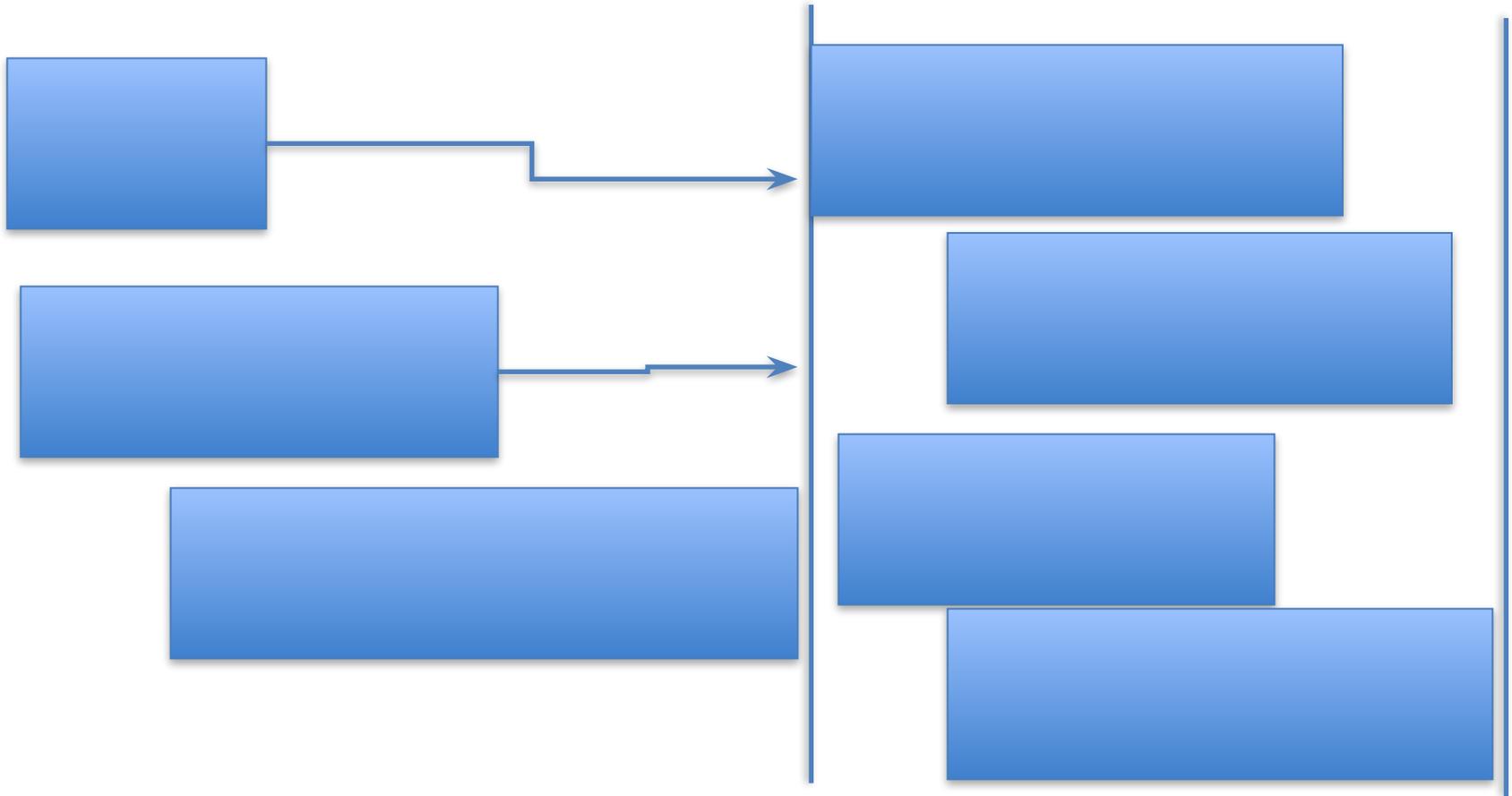


Немного истории

- В 1.6 - jsr166x
 - BlockingDeque
 - ConcurrentNavigableMap, NavigableMap
- В 1.7 - jsr166y
 - ForkJoinPool
 - Phaser
 - *ParallelArray (jsr166y extra)*



java.util.concurrent.Phaser





java.util.concurrent.Phaser

```
void startTasks(List<Runnable> tasks, final int iterations) {  
    final Phaser phaser = new Phaser() {  
        protected boolean onAdvance(int phase, int registeredParties) {  
            return phase >= iterations || registeredParties == 0;    }  
    };
```

```
    phaser.register();  
    for (final Runnable task : tasks) {  
        phaser.register();  
        new Thread() {    public void run() {  
            do {  
                task.run();  
                phaser.arriveAndAwaitAdvance();  
            } while (!phaser.isTerminated());  
        } }.start();  
    }  
    phaser.arriveAndDeregister(); // deregister self, don't wait  
}
```



Fork Join – рекурсивная декомпозиция

- если задача маленькая - посчитать
- если большая разбить и посчитать рекурсивно



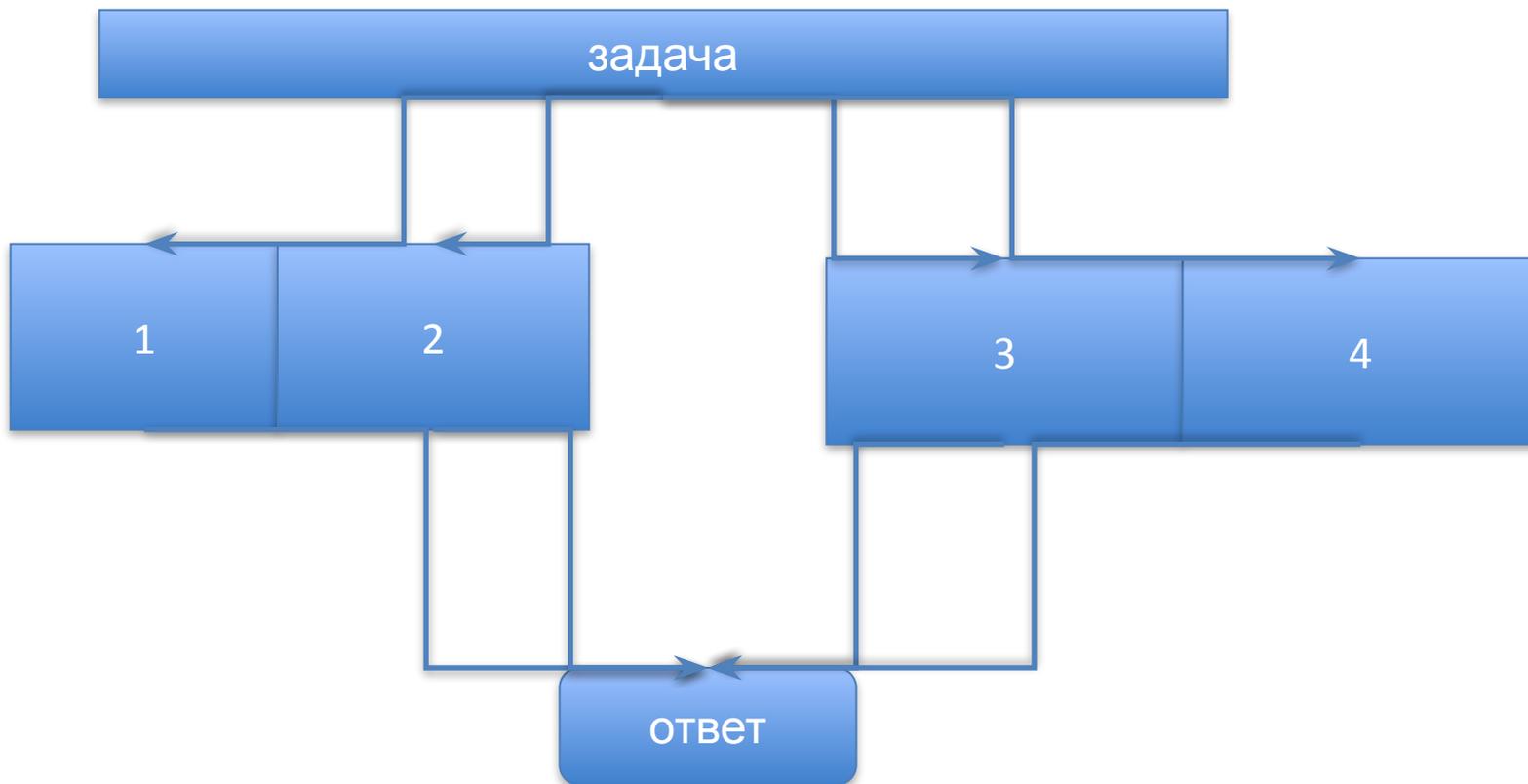


Fork Join – рекурсивная декомпозиция

- JDK7 дает возможность дробить мелко
- Минимум взаимодействия



ForkJoin





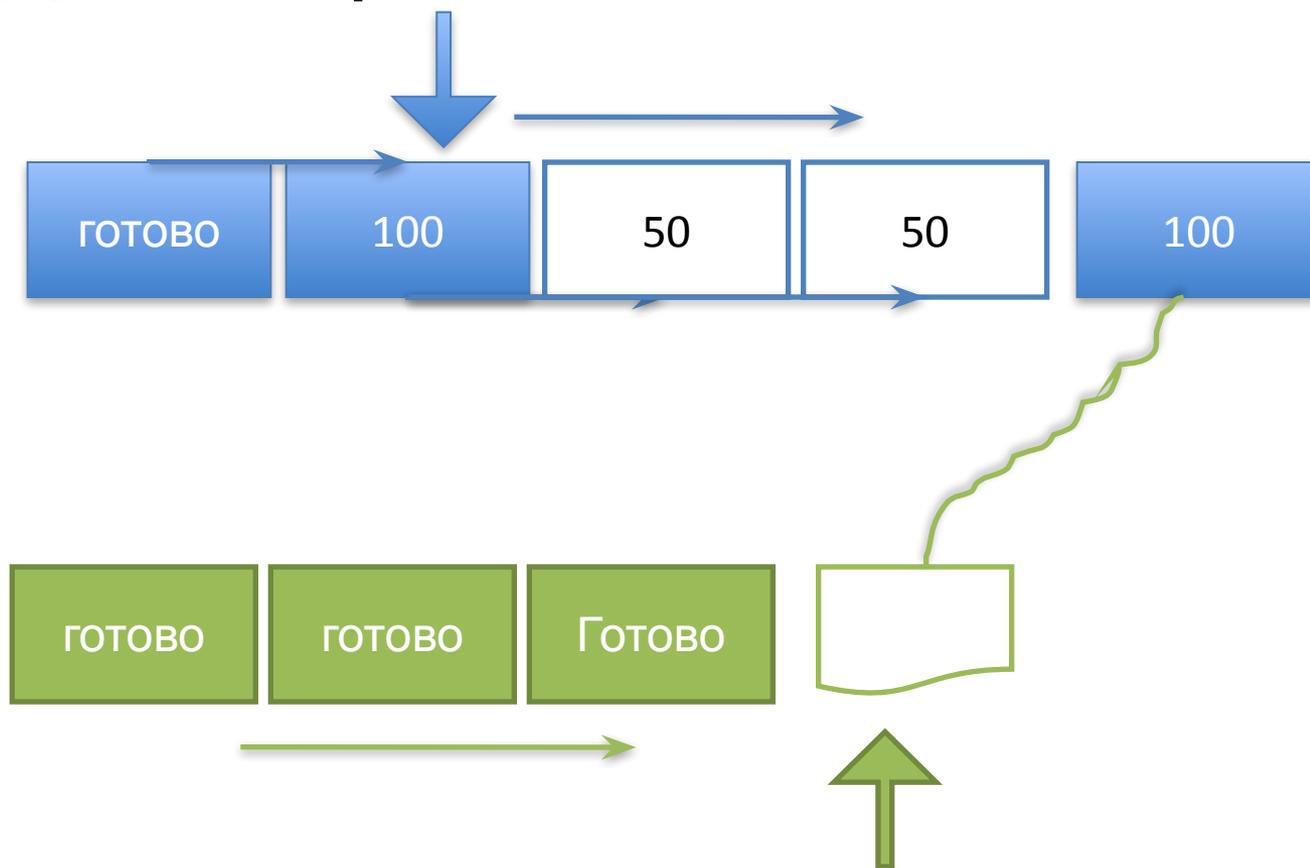
Детали реализации

- дополнительной синхронизации не требуется
- старые разбиения "больше"

Поэтому:

1. У каждого потока свой дек задач
2. поток выполнения берет задачу из дека
3. задача добавляет подзадачи в дек или производит вычисления
4. если задачи кончились - "украсть" задачу у другого потока

Детали реализации





ForkJoinTask<V>

- protected abstract boolean exec()
- ForkJoinTask<V> fork()
 - Не ждет
- public final V join()
 - То же но без исключений
- public static void invokeAll(ForkJoinTask<?>... tasks))
- Invoke = fork(); join();



RecursiveAction

- extends ForkJoinTask<Void>
- **protected void compute()**
 - Посчитать
 - Поделить
 - Вызвать `invokeAll`
 - Сделать `join`
- Нужно что то сделать, но нет возвращаемого значения
- `RecursiveTask` – есть возвращаемое значение – его вернет `join`

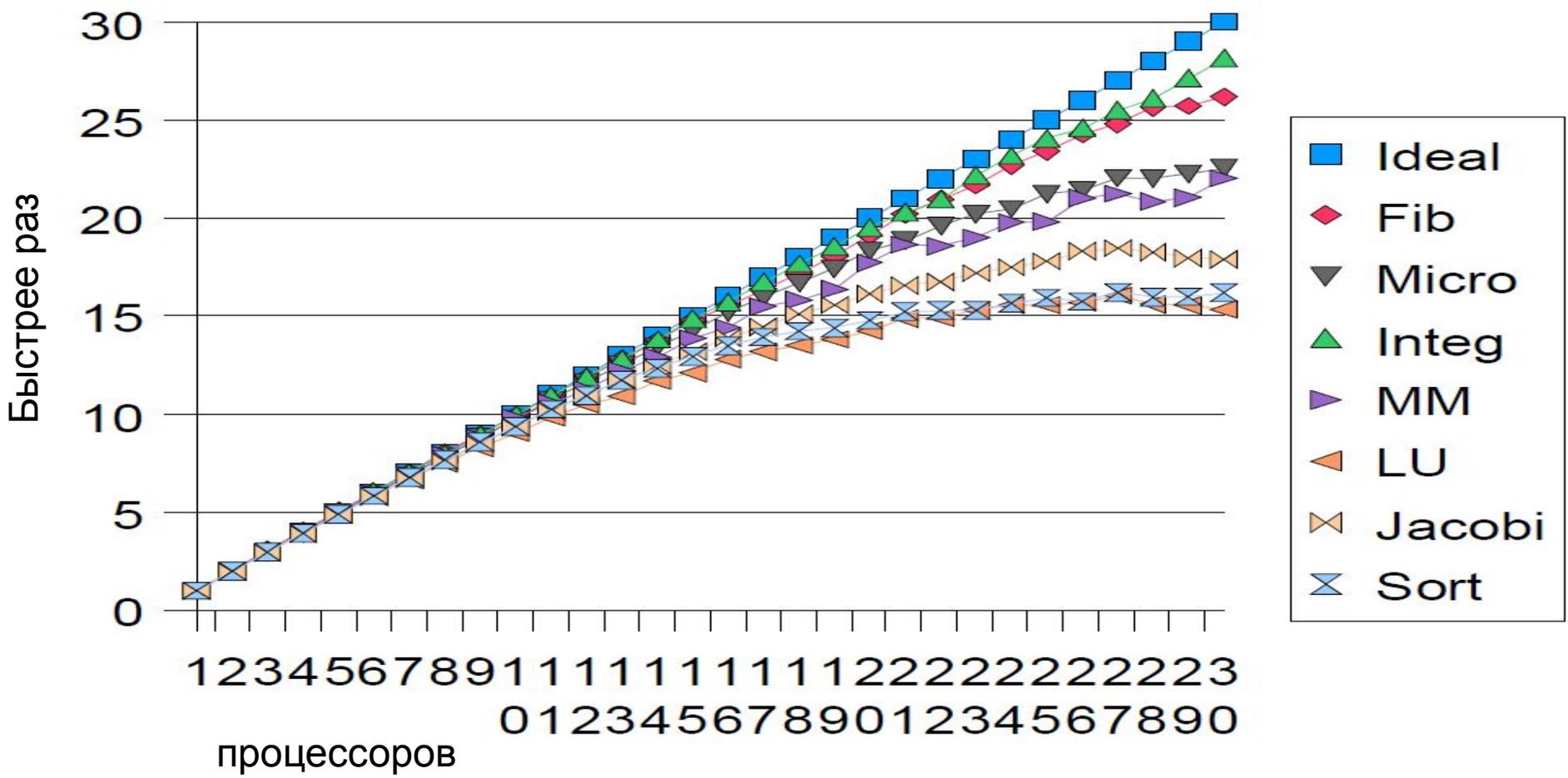


Примеры Doug Lea

Пример	Описание
Fib	Фибоначчи - подсчет числа фибоначчи для 47. Менше 13 считается влоб
Integrate	Интегрирование - суммирование численного интеграла от -47 до 48 для $(2i-1)x^{(2i-1)}$
Micro	Поиск лучшего хода в настольной игре. Прогноз на 4 хода вперед
Sort	Сортировка слиянием 100 миллионов чисел
MM	Умножение двух матриц 2048x2048. double
LU	lu-разложенные невыраженной матрицы 4096x4096. double
Jacobi	Итерационная релаксация сетки с ограничениями. Усреднение по 100 соседям. матрица 4096x4096



Результаты Doug Lea





Пример

- 2^{28} произвольных чисел double (2 гб)

```
ForkJoinPool pool = new ForkJoinPool(p);  
SinCosHuge task = new  
SinCosHuge(randomData);  
pool.execute(task);  
Double rz = task.join();
```



Пример

```
import java.util.concurrent.RecursiveTask;
public class SinCosHuge extends RecursiveTask<Double> {
    ...
    protected Double compute() {
        if (to - from < threshold) {
            double rz = 0;
            for (int i = from; i < to; i++) {
                rz += Math.sin(data[i]) + Math.atan(data[i]);
            }
            return rz;
        } else {
            int i = (from + to) / 2;
            SinCosHuge right = new SinCosHuge(data, from, i);
            SinCosHuge left = new SinCosHuge(data, i, to);
            invokeAll(left, right);
            return right.join() + left.join();
        }
    }
    ...
}
```



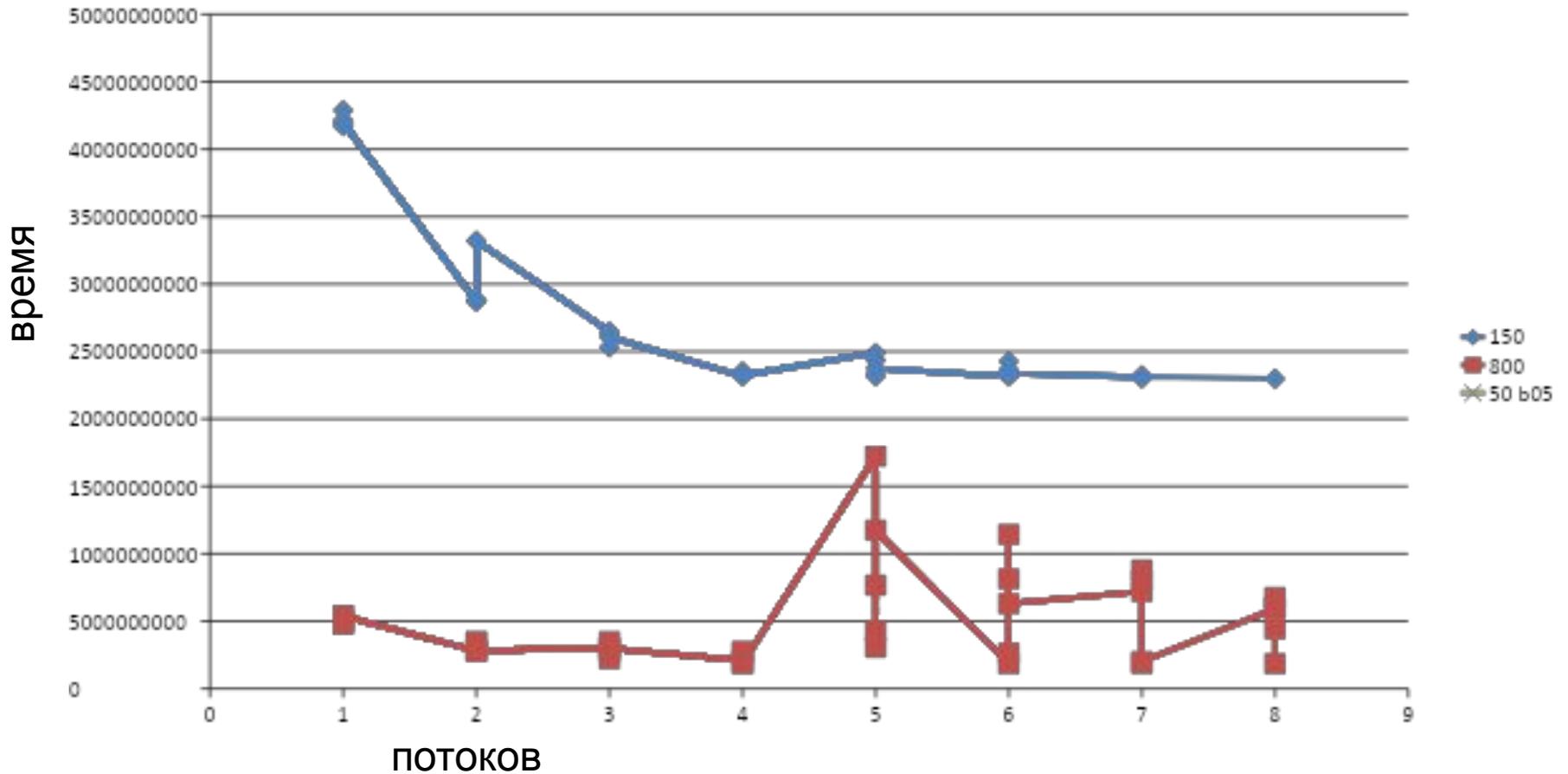
Мои измерения

Загрузка процессора 100%
температура



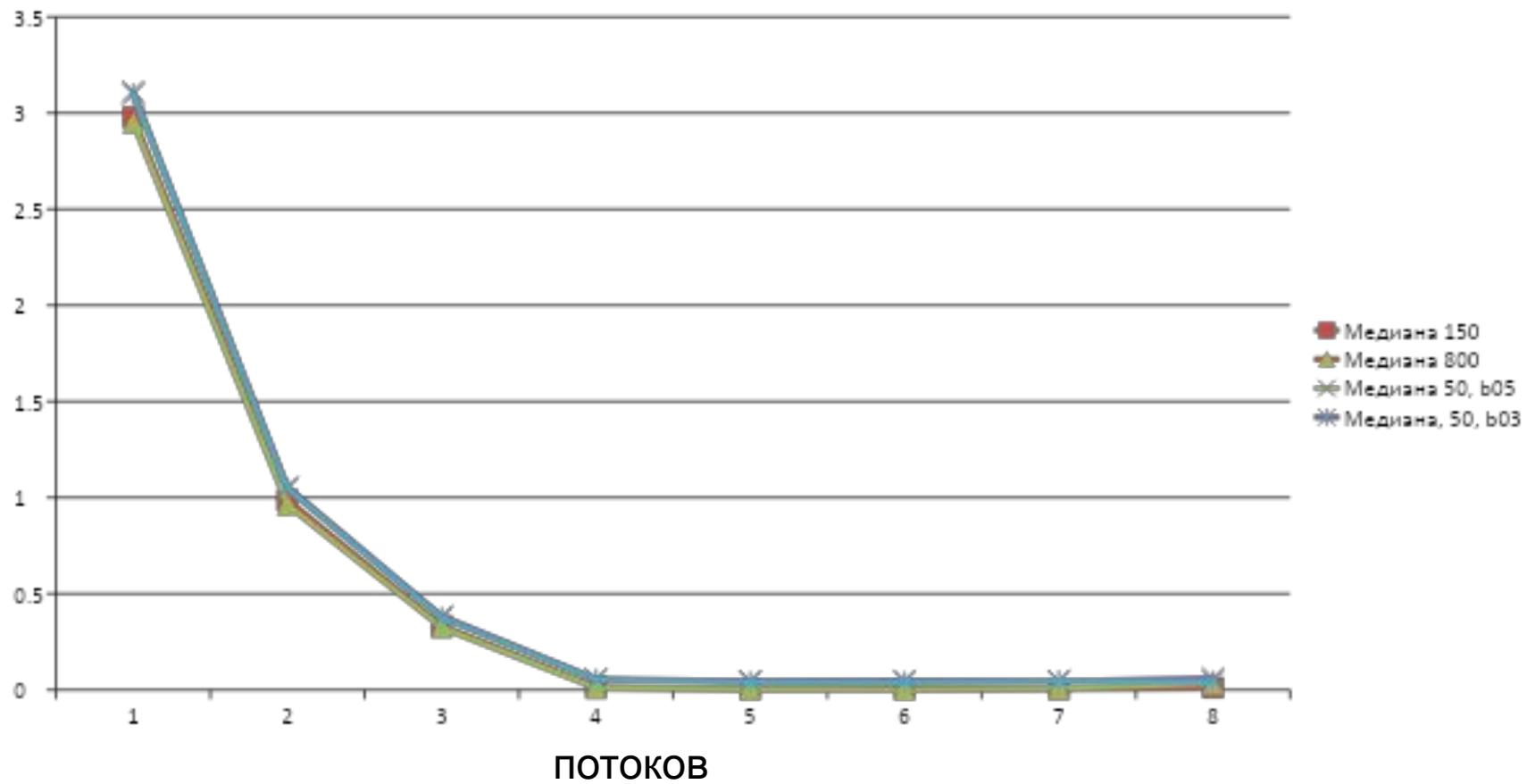


Мои измерения, время выполнения.

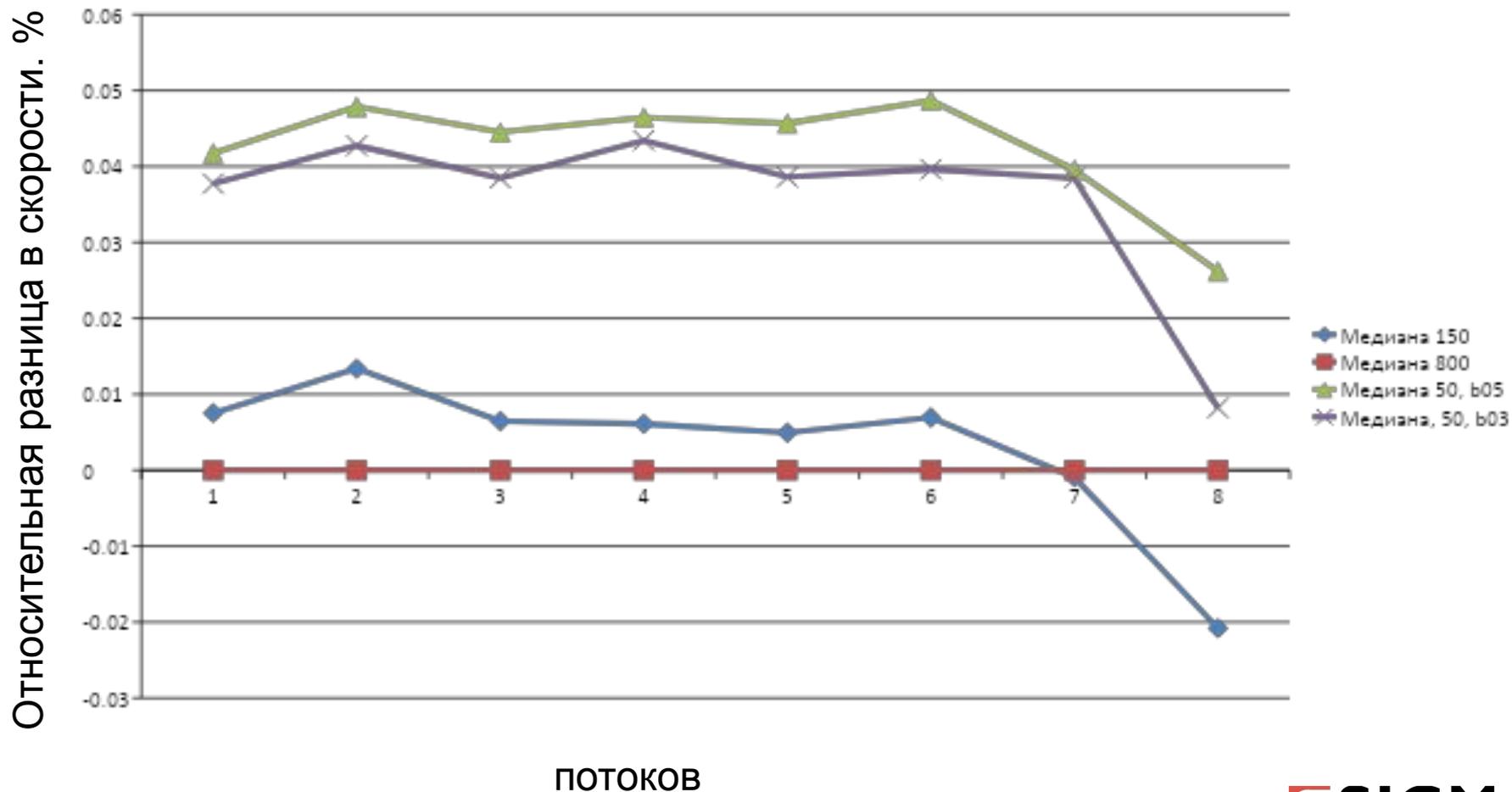




Во сколько раз медленнее минус один



Мои измерения — относительно самого быстрого





А если по старому?

- ThreadPoolExecutor
- Result
 - add(double)
 - waitDone
 - fork

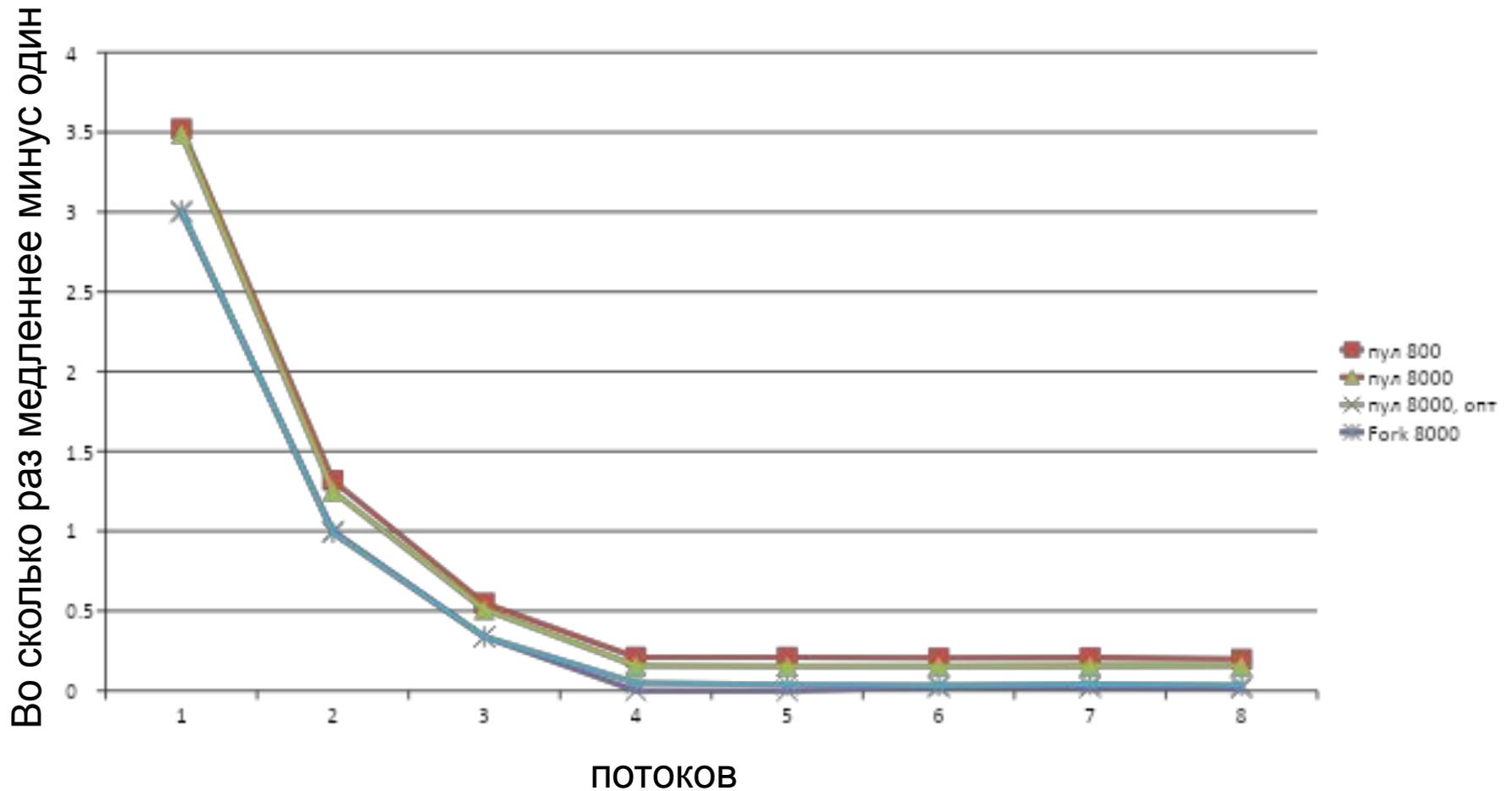
```
ThreadPoolExecutor tpe = new ThreadPoolExecutor(p, p, 10,SECONDS, workQueue);  
Result rz = new Result();  
tpe.submit(new SinCosHugePool(randomData, tpe, rz));  
rz.waitDone();
```



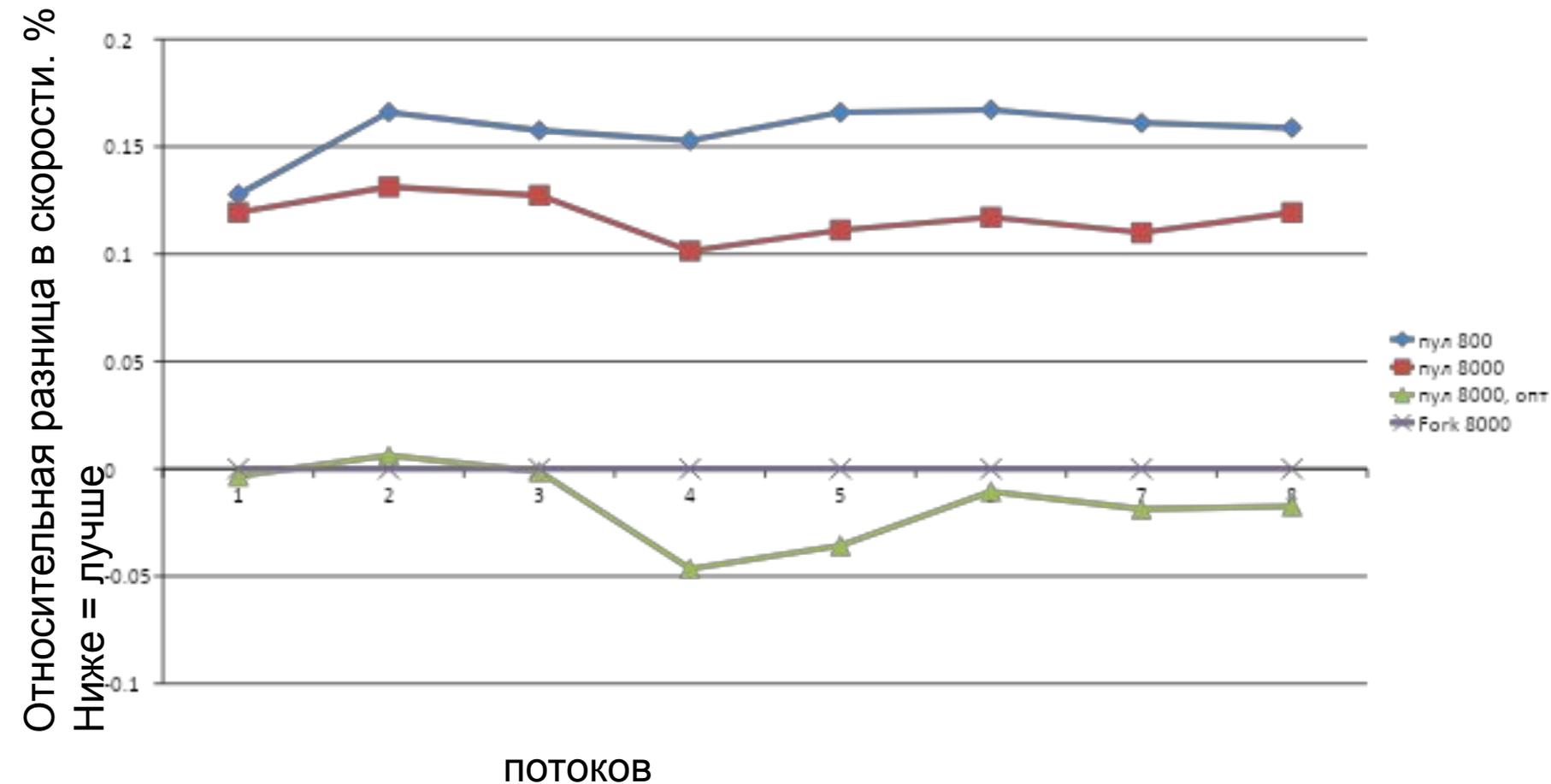
Runnable.run

```
if (to - from < treshold) {
    double rz = 0;
    for (int i = from; i < to; i++) {
        rz += Math.sin(data[i]) + Math.atan(data[i]);
    }
    result.add(rz);
} else {
    int i = (from + to) / 2;
    SinCosHugePool right = new SinCosHugePool(data, from, i, executor,result);
    SinCosHugePool left = new SinCosHugePool(data, i, to, executor,result);
    result.fork();
    executor.execute(left);
    executor.execute(right);
}
```

А если по старому?



А если по старому





ParallelArray

- Судя по всему HE попадет в JDK 1.7, но исходники доступны
- MapReduce в пределах одной машины



Пример IBM

```
ParallelArray<Student> students = new ParallelArray<Student>(fjPool, data);
double bestGpa = students.withFilter(isSenior).withMapping(selectGpa).max();

public class Student {
    String name;  int graduationYear;  double gpa;
}

static final Ops.Predicate<Student> isSenior = new Ops.Predicate<Student>() {
    public boolean op(Student s) { return s.graduationYear == Student.THIS_YEAR;  }
};

static final Ops.ObjectToDouble<Student> selectGpa = new Ops.
ObjectToDouble<Student>() {
    public double op(Student student) { return student.gpa;  }
};
```

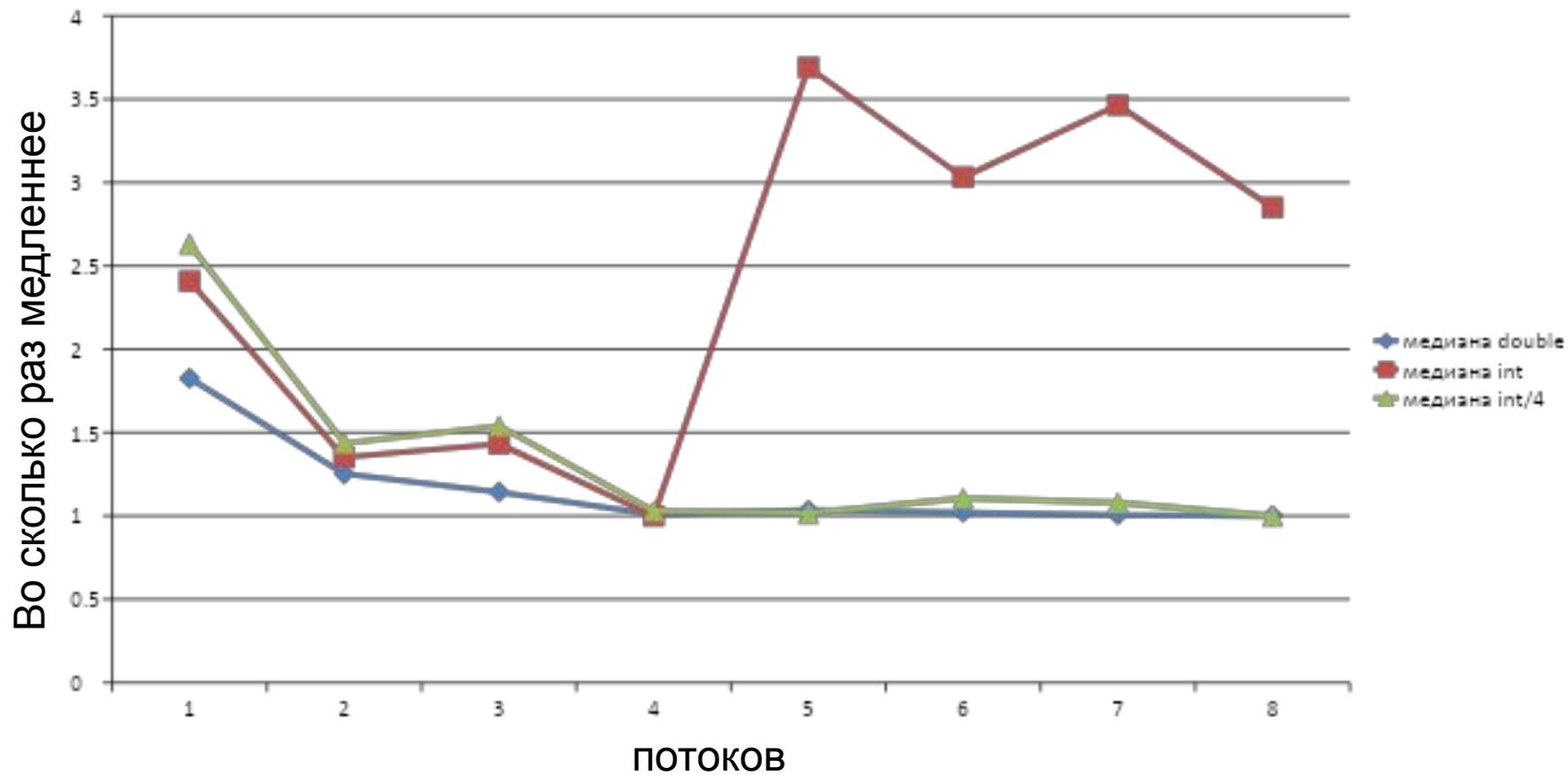


Мой пример

- `new long[16384 * 16384/8]` - 1 гб рабочей памяти,
- `double[16384 * 16384/2]` – 6гб рабочей

```
import jsr166y.ForkJoinPool;
import extra166y.ParallelLongArray;
.....
long[] randomData = new long[16384 * 16384/8];
ForkJoinPool pool = new ForkJoinPool(p);
ParallelLongArray arr = ParallelLongArray.createUsingHandoff(randomData, pool);
int uniqueCount = arr.allUniqueElements().size();
ParallelLongArray.SummaryStatistics summary = arr.summary();
```

Результаты





Откуда начать

- Concurrency JSR-166 Interest Site:
<http://g.oswego.edu/dl/concurrency-interest/>