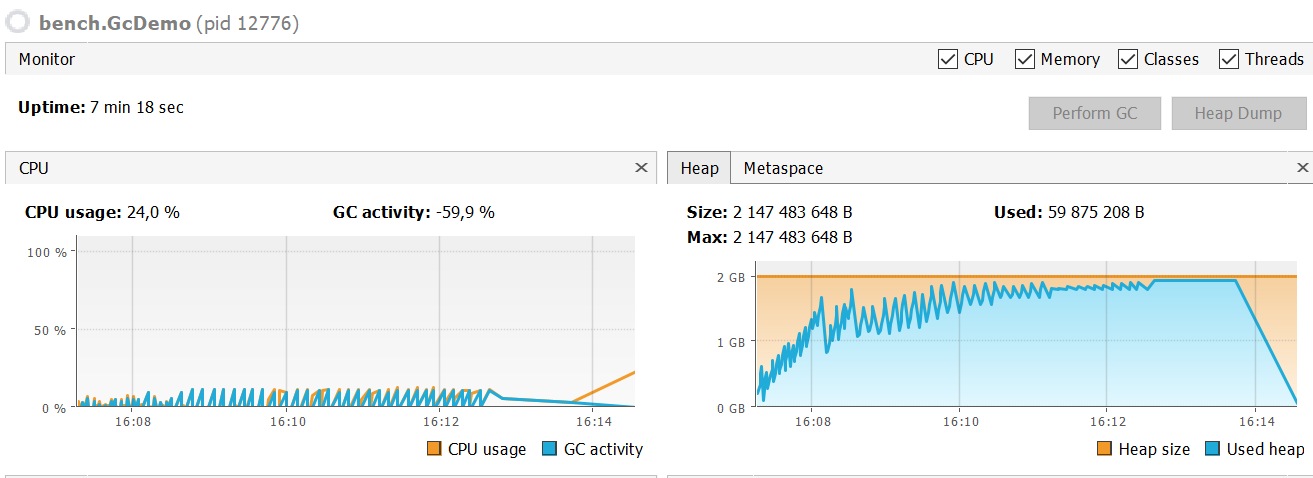
Настройки JVM:  
*-Xms2048m  
-Xmx2048m  
-XX:+HeapDumpOnOutOfMemoryError  
-XX:HeapDumpPath=./GC\_Log  
-verbose:gc  
-Xlog:gc\**

*Список GC:  
-XX:+UseSerialGC  
-XX:+UseParallelGC  
-XX:+UseG1GC  
-XX:+UnlockExperimentalVMOptions -XX:+UseZGC //Доступен на Windows начиная с JDK 14*

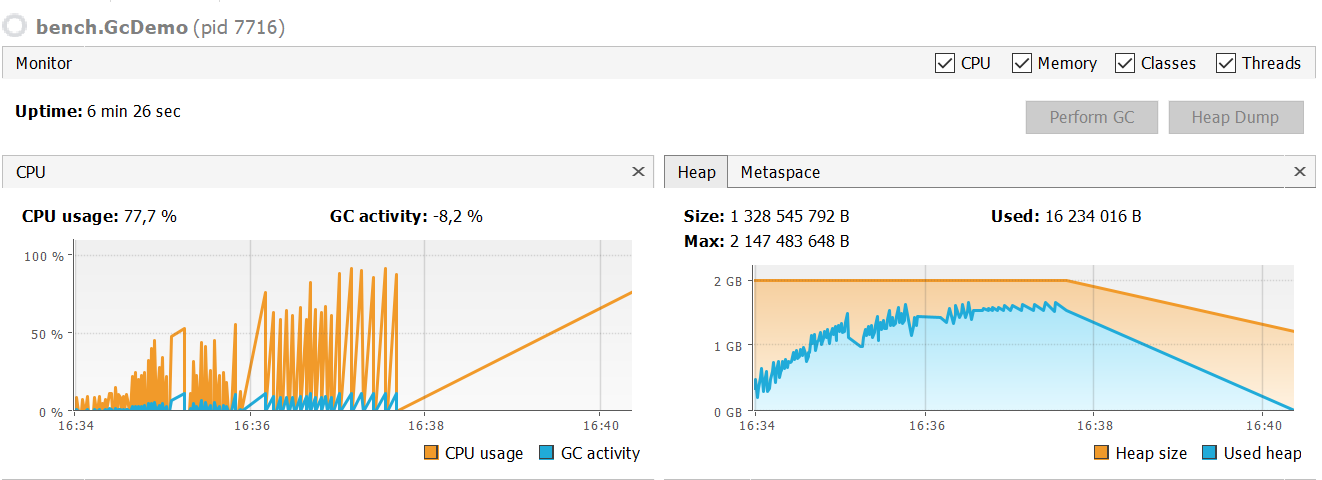
*SerialGC*

**

**java.lang.OutOfMemoryError: Java heap space**

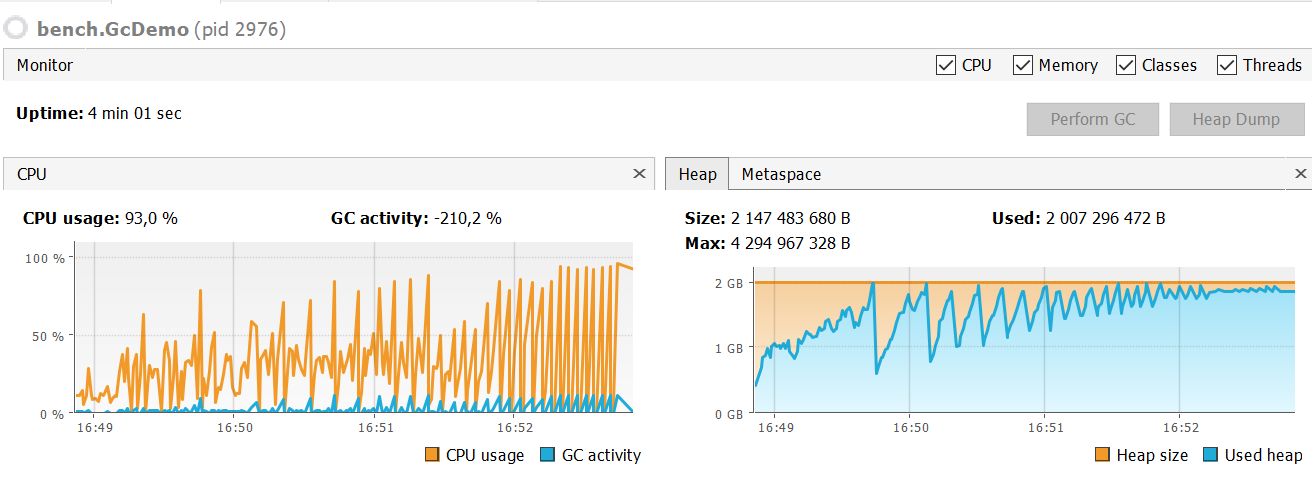
Закончилось место для аллокации объектов

*ParallelGC*

**

**java.lang.OutOfMemoryError: GC overhead limit exceeded**Время, затраченное на сборку мусора, > полезного времени работы приложения

*G1GC*

**

**java.lang.OutOfMemoryError: Java heap space**

Закончилось место для аллокации объектов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Время работы до получения *OutOfMemoryError* | Количество минорных сборок | Время затраченное на минорные сборки (в секундах) | Количество мажорных сборок | Время затраченное на мажорные сборки(в секундах) |
| *SerialGC* | 7м 18с | 26 | 7.043 | 37 | 178.794 |
| *ParallelGC* | *6м 26с* | *46* | *13.838* | *24* | *112.388* |
| *G1GC* | 4м 1с | 204 | 26.963 | 31 | 52.203 |
| *ZGC* | Не тестировался, т.к доступен для Windows начиная с 14 JDK. На ПК установлена 11ая, потестировать не получилось | | | | |

Выводы:

Наилучшим вариантом в плане производительности показал себя **G1** – у него гораздо меньше времени ушло на сборки в % от общего времени работы приложения. Но 2 ГБ памяти для него маловато. Не смотря на то, что по времени работы до OOM он уступает двум другим GC, стоит учитывать, что Serial/Parallel гораздо больше времени тратят на Stop-The-World (Де-факто приложение не работает), а Parallel при этом еще и сильно утилизирует CPU.  
  
**Serial** является хорошим вариантом для машин с низкими вычислительными мощностями.

**Parallel** представляет собой улучшенный вариант Serial, но при запуске на слабой машине последние 2 минуты (до падения с OOM) можно провести в Stop-The-World’e.  
  
Согласно документации от Oracle, и конкретно, в статье «**Memory Management in the Java HotSpot™ Virtual Machine»** указано следующее:

The ergonomics described in the previous section lead to automatic garbage collector, virtual machine, and heap size selections that are reasonable for a large percentage of applications. Thus, the initial recommendation for selecting and configuring a garbage collector is to do nothing! That is, do not specify usage of a particular garbage collector, etc. Let the system make automatic choices based on the platform and operating system on which your application is running. Then test your application. If its performance is acceptable, with sufficiently high throughput and sufficiently low pause times, you are done. You don’t need to troubleshoot or modify garbage collector options.  
  
То есть, JVM самостоятельно выберет подходящую конфигурацию GC для вашего приложения.   
В моем случае, по информации снятой с помощью JConsole, наиболее эффективным считается **G1.**

