Memory mapped parallel BAM file access API for high throughput sequence analysis informatics

Timothy A. Pitman[1], Xiaomeng Huang[1], Gabor T. Marth[1], Yi Qiao[1]*

1 UTAH Center for Genetic Discovery, Eccles Institute of Human Genetics, University of Utah, Salt Lake City, UT 84112

* Corresponding Author
Yi Qiao, PhD
Director, Research & Science
UTAH Center for Genetic Discovery
Eccles Institute of Human Genetics
University of Utah School of Medicine
15 N 2030 E, Rom 7150
Email: yi.qiao@genetics.utah.edu

RUNNING TITLE: high throughput parallel BAM access library: mmbam

ABSTRACT

In precision medicine, genomic data needs to be processed as fast as possible to arrive at treatment decisions in a timely fashion. We developed mmbam, a library to allow sequence analysis informatics software to access raw sequencing data stored in BAM files extremely fast. Taking advantage of memory mapped file access and parallel data processing, we demonstrate that analysis software ported to mmbam consistently outperforms their stock versions. Open source and freely available, we envision that mmbam will enable a new generation of high-performance informatics tools for precision medicine.

INTRODUCTION

High throughput, genome wide next generation sequencing (NGS) have revolutionized precision medicine. As an example, NGS has now been implemented as a routine diagnostic modality in many pediatric subspecialty clinics for critically ill children admitted into the neonatal intensive care unit or pediatric intensive care unit(Elliott et al. 2019; Petrikin et al. 2015). And increasingly, genomics-guided precision medicine is helping advanced cancer patients who have exhausted standard-of-care options(Schwartzberg et al. 2017). In these settings, the amount of data analyzed is small compared to large cohort studies, involving usually one to a few tumor samples and the paired normal sample from the same patient. However, the analysis turnaround is of critical importance. For example, it is the longstanding standard of care practice to initiate definitive therapy within 2 to 4 weeks after pediatric cancer patients receive biopsy / resection surgery(Ross et al. 2021). Furthermore, after the optimal treatment is identified, it still takes significant time to coordinate treatment access due to e.g. drug acquisition, compassionate care approval, clinical trial enrollment, or insurance authorization. It is therefore significant that the informatics analysis tasks, which have surpassed sequencing as the primary bottleneck, are to be as fast as current computer hardware can make possible.

The BAM file format(Li et al. 2009) is the current *de facto* standard for storing sequencing data generated from NGS experiments. BAM files are the most common starting place for various downstream analyses. The BAM format is the compressed, binary version of the SAM format, which we designed as part of the 1000 Genomes Project(1000 Genomes Project Consortium et al. 2015) to reconcile the once many different formats of storing sequencing data. Subsequently, software libraries are created to provide APIs to access the sequencing reads contained in a BAM file. HTSLIB(Bonfield et al. 2021) is the file access layer from Samtools(Li et al. 2009), the software developed to perform many SAM / BAM file related operations. The main focus of HTSLIB is to provide high level abstractions so that the programming interfaces stay the same regardless of the underlying file format (be it SAM, BAM, or CRAM) or storage and transport media (local files, HTTP URLs, or cloud storage). BamTools(Barnett et al. 2011) and SeqLib(Wala and Beroukhim 2017) focus on modern C++ API designs for ease of programming. While BamTools implements its own BAM parsing logic, SeqLib integrates HTSLIB as the access layer. These libraries perform file access in a single-threaded manner (HTSLIB does support multi-threaded decompression, but not file reading), which is a reasonable design choice 1) when informatics analysis is compute bottlenecked, therefore cannot benefit from faster data access; or that 2) the file storage and transport media are incapable of high performance, e.g. when the files are served over network attached storages equipped with

hard disk drives. However, they are unable to take advantage of file I/O parallelization and higher data bus bandwidth supported by modern hardware, such as nonvolatile memory express solid state drives (NVME SSDs).

We propose a new approach for parallelized BAM file access. We developed mmbam, which uses memory mapped I/O and the bam file index (BAI) for parallel data reading, and takes the scatter / gather programming paradigm to parallelize computation tasks over many different genomic regions. Since the majority of sequence analysis informatics analyses (e.g. quality control, various types of mutation calling) involve reading already indexed BAM files, mmbam has the potential to significantly shorten end-to-end analysis turnaround. Mmbam is freely available at https://gitlab.com/yiq/mmbam/.

RESULTS

Parallel data processing architecture using mmbam. Figure 1 demonstrates the architecture we envision software using mmbam will take. At the hardware level, the compute servers will be equipped with fast storage capable of parallel access. Since data of only one patient needs to be analyzed on one server at a time, this hardware configuration is readily available both locally and on the cloud. The API design facilitates a pattern where analysis tasks over different genomic regions can occur independently and simultaneously. Using the memory map system call supported by the Linux operating system, common for high throughput computing environments, parallel reading from the storage media occurs transparently and with minimal overhead. Once the data corresponding to a particular genomic region is loaded into memory, decompression and data processing occurs in parallel to other regions, producing regional results which can then be combined to generate the final, global results. We describe example applications that fit this programming style below.

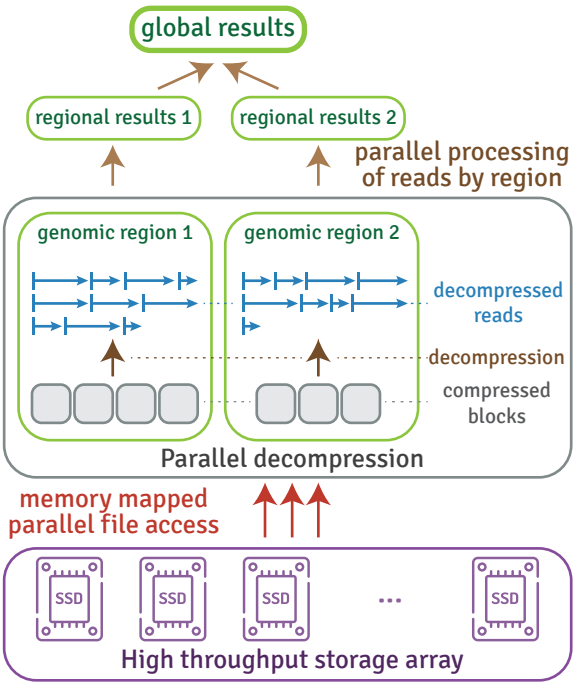

Figure 1, mmbam data processing architecture. Mmbam enables parallel processing of alignment data via memory mapped file access, and utilizes the scatter / gather paradigm to parallelize computation tasks across many genomic regions before combining the regional results to produce global results.

Proof-of-concept implementation of samtools flagstats and performance benchmarking. The first sample program we ported to mmbam is the utility in samtools called flagstats. Flagstats iterates over the entire BAM file, updating statistics (e.g., number of reads failed QC) with the flags field of each read, and finally printing the statistics when all reads are processed. Using mmbam, it is possible to compute separate statistics for each 16 kilo-bases window across the entire genome, followed by calculating the sum of the results from all windows. Samtools does have the ability to parallelize decompression but can only read data with one thread at a time. With a single thread, mmbam based

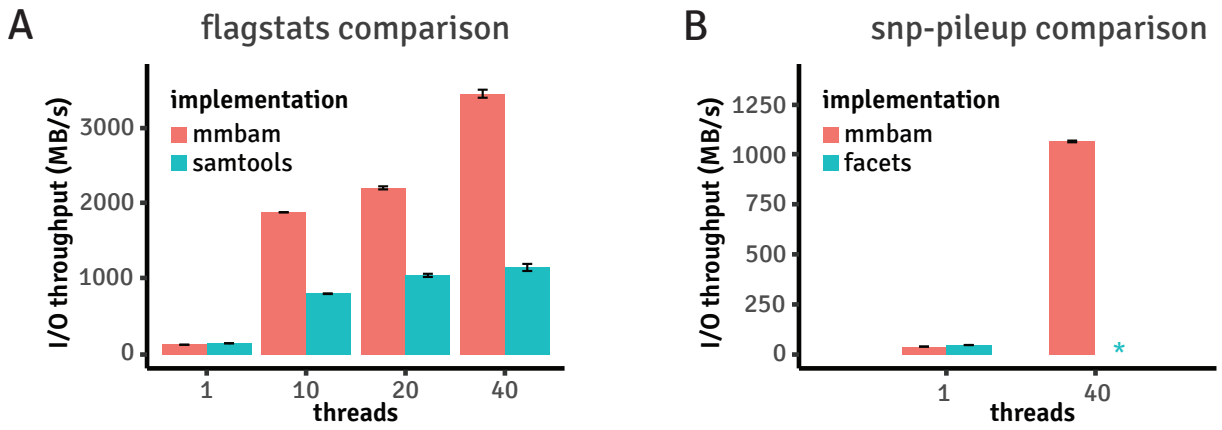

Figure 2, Performance benchmark results of sample programs ported to mmbam vs their stock versions. A) Performance benchmarking on the flagstats algorithm implemented with mmbam vs. its original implementation found in Samtools. **B)** Performance benchmarking on the snp-pileup utility program, part of the FACETS copy number variation detection algorithm, reimplemented using mmbam vs its original implementation. * The original version of snp-pileup does not support multi-threading.

flagstats shows a similar performance as samtools (**Figure 2A**. 133.79MB/s for mmbam and 148.92MB/s for samtools). However, while samtools benefits little from more than 10 threads (808.33MB/s, 1051.33MB/s, and 1155.99MB/s with 10, 20, and 40 threads), mmbam allows for a much better scaling (1886.27MB/s, 2209.31MB/s, and 3454.36MB/s with 10, 20, and 40 threads). Full timing observations are described in **Supplemental Table 1**. The mmbam version of flagstats produces identical results compared to the original version. Other algorithms that can potentially be implemented in a similar fashion include but are not limited to read counting per fixed genome windows for CNV detection and transcript abundance counting per gene in RNAseq data analysis.

Reimplementation of a real-world, widely used program and performance benchmarking. The second sample program we ported to mmbam is a utility found in the somatic copy number variant detection algorithm FACETS(Shen and Seshan 2016) called "snp-pileup". FACETS utilizes the coverage and variant allele fraction at population polymorphic sites in the tumor and paired normal sequencing data to estimate the absolute copy number and loss-of-heterozygosity status of the tumor sample. Snp-pileup takes as input a set of BAM files and a VCF file (commonly the dbSNP published human common polymorphic sites(Sherry et al. 1999)), and iterates over positions in the VCF file. At each position, it pulls all the reads from the BAM files overlapping with the position, and extracts the sequencing coverage and variant allele fraction information. Different from the flagstats example which parallelizes over multiple, non-overlapping genomic windows, we ported snp-pileup to parallelize over each variant position. As shown in **Figure 2B**, mmbam snp-pileup with 40 cores parallelization achieved over 1 Gigabyte per second data processing throughput (1039MB/s) with mmbam's built-in multiple input pileup engine, roughly 22 times faster than the original implementation (47.5MB/s) which does not support multithreading. Consequently, using the Genome In A Bottle(Zook et al. 2016) HG002 and HG004 Illumina 2x250bp BAM files (243 Gigabytes of data combined), a two samples joint snp-pileup can be finished in 4 minutes (mmbam), compared to 1 hour 27 minutes (FACETS stock version). Full timing observations are described in **Supplemental Table 2**. The mmbam version of snp-pileup produces functionally identical results (different results were observed at 143 out of 28.5 million positions, or a rate of 0.000005) compared

to the original version. Other algorithms that can potentially be implemented in a similar fashion include single cell sequencing data genotyping and variant calling.

DISCUSSION

In this manuscript we present mmbam, an API and library for accessing sequence alignments in BAM files with a high degree of parallelization and performance. We achieve this by taking advantage of parallel file access supported by modern storage hardware, and facilitated by the memory map function offered by the Linux / Unix operating system. Our emphasis on the scatter / gather programming style allows parallelizable computation tasks to be easily implemented. As a demonstration of mmbam's utility, we ported various types of sequence analysis informatics algorithms, which have shown consistently higher performance than their original implementations. We summarized the expected runtimes on typical datasets using software implemented with mmbam vs their stock versions in **Table 1**.

Data Set	Typical Size	Durations (flagstats)		Durations (snp-pileup)	
		samtools	mmbam	facets	mmbam
200X WES (1 sample)	32GB	28s	9.5s	11.5m	31.5s
60X WGS (1 sample)	200GB	2m57s	59.3s	1h11m51s	3m17.1s
120X WGS (tumor/normal pair)	600GB	8m51.5s	2m57.9s	3h35m34s	9m51s

Table 1. Expected duration of running the stock versions of flagstats and snp-pileup vs the versions implemented using mmbam on representative data set types with typical data sizes. Expected durations are calculated based on the best performance observed from the benchmarking experiment.

Our results highlight the level of performance obtainable when data is available on locally attached high performance file storage (e.g. NVME SSD Raid0), and the computation parallelized over many threads. High performance local file storage has not been a popular configuration for high performance compute environments because the datasets to be analyzed are often too big to fit on a single computer. Albeit true for traditional, large cohort studies (e.g. the 1000 Genome Project , or The Cancer Genome Atlas), this is no longer a limiting factor for small sample-size analyses commonly found in precision medicine settings (e.g. tumor and paired normal samples from the same patient). Given the benefit of fast turnaround, we envision a new analysis infrastructure model where one or a few servers, either locally managed or instantiated on the cloud, are used to perform extremely fast analysis for the patient actively investigated in a precision medicine program. These servers will be equipped with fast storage large enough to hold the raw and intermediate data for the analysis of one patient case. After the initial data transfer into the server, which can be pipelined together with unavoidable, compute-bottlenecked analysis steps (e.g. read mapping) to hide latency, all subsequent analyses can benefit from fast data access and parallel computing. When all analysis tasks are finished, the results are then uploaded to central file storage (slower but with larger capacity), and all data relevant to this patient can be ejected from the local storage to prepare the server for the next case. Contrary to the currently common infrastructure where compute nodes are connected by a shared file system on which analyses occur, this model offers better analysis turnaround for individual cases, without compromising storage cost-effectiveness.

Beyond accelerating data access on traditional hardware, mmbam offers the opportunity to take advantage of hardware accelerators for further performance gains. As an example, all the sample programs we presented in this manuscript spend a significant amount of CPU time in data decompression. Since the bgzip blocks are gzip specification compliant, their decompression can be offloaded onto field programmable gate array (FPGA) devices for better throughput. In order to fully take advantage of the 16GB/s PCI-E 3.0 x16 cpu-to-device data transfer bandwidth (or even higher bandwidth enabled by future PCI-E standards), parallel reading of BAM bgzip blocks is a must. Alternatively, if an algorithm is compute-intensive but massively parallelizable (e.g. variant calling), it can potentially be implemented on the graphics processing units (GPUs). Parallel reading and decompressing the bgzip blocks on the CPU will ensure that the GPU is not data starved. These techniques can be combined together to produce the next generation sequence analysis informatics applications to support rapid precision medicine analysis turnaround so that we can arrive at treatment decisions much sooner, or perform more comprehensive analyses, than what is currently possible.

In conclusion, we have developed a high throughput NGS data access library -- mmbam -- taking advantage of parallelization to achieve exceptional performance. Our work enables many types of sequence analysis software to be accelerated significantly, which in turn benefit time sensitive clinical / research applications such as precision medicine. We demonstrated that existing algorithms can be ported to mmbam, resulting in much higher data processing throughput without altering the results. Our code is open source and publicly available. We plan to actively maintain the project, incorporating further improvements and developing new features according to feedback from the user community.

METHODS

1. Core API design principles.

We designed our API according to the following principles.

1. The library interface is in a C-style data structure + functions that operates on these data structures.
2. The data structures are implemented as C structs. The struct members are aligned to the byte arrangement of the records in a BAM file or a BAM index (BAI) file they represent. The members are intended to be accessed by client programs directly.
3. Functions that perform more sophisticated operations on the data structures are named as <struct_name>_<operation_name> for better code readability.
4. Modern C++11 and C++14 features are used whenever it makes sense. This includes
    a. Smart pointers for automatic memory management
    b. Iterator types in mmbam often support the ***range-for*** syntax
    c. STL algorithms are preferred over traditional loops
    d. Auto type deduction are used whenever possible
5. The implementation of mmbam uses Intel Thread Building Block (libtbb) for parallelization.

2. Benchmark experiments environment

We performed all benchmark experiments on the Amazon Web Service (AWS) cloud, using a c5d.24xlarge instance and ubuntu 20.04 operating system (AMI ID: ami-03d5c68bab01f3496). This instance type has 96 logic cores, and 192 gigabytes of system memory. Input files are placed on a raid 0 logical volume created from the 4 NVME SSD devices available on this instance, with the following linux commands:

- `mdadm --create /dev/md0 --level=stripe --raid-devices=4 /dev/nvme{1,2,3,4}n1`
- `mkfs -t ext4 /dev/md0`

Because the linux operating system kernel caches all file read operations into unused system memory, the subsequent experiments reading the same files will not incur actual disk operations. To maintain the same conditions between experiments, the kernel caches are manually dropped using the following linux command

- `echo 3 > /proc/sys/vm/drop_caches`

Elapsed times are measured using the "gnu time" utility shipped with the operating system.

3. Flagstats benchmark experiment

We used the Genome In A Bottle (GIAB)(Zook et al. 2016) Ashkenazim Trio HG002 Illumina 2x250bp GECh38 BAM file for the benchmarking experiment. After downloading the BAM file, we used samtools(Li et al. 2009) 1.13 to filter out unmapped reads, as they are placed at the end of the BAM file and cannot be parallel accessed. Note that programs implemented with mmbam will have the ability to access these reads, albeit without multithreading benefits. However because this particular BAM file contains a large number (roughly 4GB) of unmapped reads, the performance benchmark will be skewed if they are included. The experiments were carried out with 1, 10, 20, 40 core configurations. Each configuration is repeated three times to obtain standard deviations.

4. Snp-pileup benchmark experiment

We used the GIAB Ashkenazim Trio HG002 and HG004 Illumina 2x250bp GRCh38 BAM files for the benchmarking experiment to emulate a tumor / normal sample pair, as needed by snp-pileup. We used the VCF recommended by the snp-pileup documentation (ftp://ftp.ncbi.nlm.nih.gov/snp/organisms/human_9606/VCF/00-common_all.vcf.gz) for sites of known polymorphism. We further used the '-x' option when running the stock version of snp-pileup because the multiple pileup engine in mmbam does not currently re-adjust base qualities of overlapping mates. We ran the stock snp-pileup with one thread, repeated three times to obtain standard deviations. No further runs were performed since the stock snp-pileup does not support multi-threading. We ran the snp-pileup ported to mmbam with both 1 and 80 core configurations, each repeated three times to obtain standard deviations.

DATA ACCESS

All data used for benchmark experiments are publicly available. Specifically:

GIAB Ashkenazim Trio HG002 and HG004 Illumina 2x250bp novoalign GRCh38 BAM files are available at

- https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG002_NA24385_son/NIST_Illumina_2x250bps/novoalign_bams/
- https://ftp-trace.ncbi.nlm.nih.gov/ReferenceSamples/giab/data/AshkenazimTrio/HG004_NA24143_mother/NIST_Illumina_2x250bps/novoalign_bams/

Known polymorphism sites VCF used for the snp-pileup benchmark experiments are available at

- ftp://ftp.ncbi.nlm.nih.gov/snp/organisms/human_9606/VCF/00-common_all.vcf.gz

COMPETING INTEREST STATEMENT

The authors declare no competing interests.

ACKNOWLEDGEMENTS

A.P., X.H., G.M., and Y.Q. are supported by NIH Grant U24CA209999, and an internal grant from the Center for Genomic Medicine for the development of computational pipelines supporting precision oncology.

AUTHOR CONTRIBUTIONS

Y.Q. conceived the project, contributed to the development of software code, and designed and performed benchmark experiments. A.P. contributed to the design, development, and testing of the software code. X.H. performed data analysis on the benchmarking experiment results. G.M. contributed to the design of the project, and provided support. All authors read and edited the manuscript.

REFERENCES

1000 Genomes Project Consortium, Auton A, Brooks LD, Durbin RM, Garrison EP, Kang HM, Korbel JO, Marchini JL, McCarthy S, McVean GA, et al. 2015. A global reference for human genetic variation. *Nature* **526**: 68–74.

Barnett DW, Garrison EK, Quinlan AR, Strömberg MP, Marth GT. 2011. BamTools: a C++ API and toolkit for analyzing and managing BAM files. *Bioinformatics* **27**: 1691–1692.

Bonfield JK, Marshall J, Danecek P, Li H, Ohan V, Whitwham A, Keane T, Davies RM. 2021. HTSlib: C library for reading/writing high-throughput sequencing data. *Gigascience* **10**. http://dx.doi.org/10.1093/gigascience/giab007.

Elliott AM, du Souich C, Lehman A, Guella I, Evans DM, Candido T, Tooman L, Armstrong L, Clarke L, Gibson W, et al. 2019. RAPIDOMICS: rapid genome-wide sequencing in a neonatal intensive care unit-successes and challenges. *Eur J Pediatr* **178**: 1207–1218.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, 1000 Genome Project Data Processing Subgroup. 2009. The Sequence Alignment/Map format and SAMtools. *Bioinformatics* **25**: 2078–2079.

Petrikin JE, Willig LK, Smith LD, Kingsmore SF. 2015. Rapid whole genome sequencing and precision neonatology. *Semin Perinatol* **39**: 623–631.

Ross A, Gomez O, Wang X, Lu Z, Abdelhafeez H, Davidoff AM, Talbot L, Murphy AJ. 2021. Timing of adjuvant chemotherapy after laparotomy for Wilms tumor and neuroblastoma. *Pediatr Surg Int*. http://dx.doi.org/10.1007/s00383-021-04968-1.

Schwartzberg L, Kim ES, Liu D, Schrag D. 2017. Precision Oncology: Who, How, What, When, and When Not? *Am Soc Clin Oncol Educ Book* **37**: 160–169.

Shen R, Seshan VE. 2016. FACETS: allele-specific copy number and clonal heterogeneity analysis tool for high-throughput DNA sequencing. *Nucleic Acids Res* **44**: e131.

Sherry ST, Ward M, Sirotkin K. 1999. dbSNP-database for single nucleotide polymorphisms and other classes of minor genetic variation. *Genome Res* **9**: 677–679.

Wala J, Beroukhim R. 2017. SeqLib: a C ++ API for rapid BAM manipulation, sequence alignment and sequence assembly. *Bioinformatics* **33**: 751–753.

Zook JM, Catoe D, McDaniel J, Vang L, Spies N, Sidow A, Weng Z, Liu Y, Mason CE, Alexander N, et al. 2016. Extensive sequencing of seven human genomes to characterize benchmark reference materials. *Sci Data* **3**: 160025.

Supplemental Table 1, Benchmark results for samtools flagstats and mmbam flagstats

implementation	# threads	pass #	Time elapsed (s)	IO bandwidth (MB/s)	File size (MB)
samtools	1	1	777	148.92	115712
samtools	10	1	144.7	799.67	115712
samtools	20	1	115.53	1001.58	115712
samtools	40	1	110.91	1043.30	115712
samtools	1	2	777	148.92	115712
samtools	10	2	143.49	806.41	115712
samtools	20	2	106.72	1084.26	115712
samtools	40	2	96.1	1204.08	115712
samtools	1	3	777	148.92	115712
samtools	10	3	141.3	818.91	115712
samtools	20	3	108.33	1068.14	115712
samtools	40	3	94.8	1220.59	115712
mmbam	1	1	866.53	133.53	115712
mmbam	10	1	95.08	1217.00	115712
mmbam	20	1	52.93	2186.13	115712
mmbam	40	1	32.3	3582.41	115712
mmbam	1	2	864.16	133.90	115712
mmbam	10	2	95.13	1216.36	115712
mmbam	20	2	52.96	2184.89	115712
mmbam	40	2	34.09	3394.31	115712
mmbam	1	3	863.87	133.95	115712
mmbam	10	3	94.95	1218.66	115712
mmbam	20	3	51.27	2256.91	115712
mmbam	40	3	34.17	3386.36	115712

Supplemental Table 2, Benchmark results between snp-pileup stock version and mmbam version

implementation	# of threads	pass #	Time elapsed (s)	IO bandwidth (MB/s)	File size (MB)
facets	1	1	5238	47.50515464	248832
facets	1	2	5228	47.59602142	248832
facets	1	3	5238	47.50515464	248832
mmbam	1	1	6278	39.63555272	248832
mmbam	1	2	6532	38.09430496	248832
mmbam	1	3	6499	38.28773657	248832
mmbam	40	1	238.45	1043.539526	248832
mmbam	40	2	239.89	1037.275418	248832
mmbam	40	3	240	1036.8	248832