	Juicer	Hi-C Pro	HiFive	HiCdat	hiclib
Processes raw sequence data?	✔	✔	✖	✔	✔
Chimeric reads	✔	✔	✖	✖	✔
Normalization	✔	✔	✔	✔	✔
Can handle large data sets	✔	✔	✔	✖	✖
Compact file format	✔	✖	✔	✖	✖
Diploid map capability	✔	✔	✖	✖	✖
Multi-scale map generation	✔	✖	✖	✖	✖
A/B compartment annotation	✔	✖	✖	✔	✔
Contact domain annotation	✔	✖	✖	✖	✖
Peak annotation	✔	✖	✖	✖	✖
Motif discovery	✔	✖	✖	✖	✖

Table S1. **Comparison of Hi-C pipelines,** Related to Table 1. Hi-C Pro (Servant et al., 2015) can handle the large datasets now typical of Hi-C experiments and can generate diploid maps given a SNP list but does not offer a compact file format or a method for multiple resolution generation in one step. HiFive (Suria et al., 2015) has a compact file format but does not start from raw sequencing data. HiCdat (Schmid et al., 2015) does not allow for chimeric reads, which are common to Hi-C experiments, and cannot handle large datasets. Hiclib is a Python library, not a one-click pipeline, that requires comfort with computer programming, and has not been evaluated by peer review.

Table S2. Hi-C File Format, Related to Figure 1. Exact layout of the compressed *.hic* file format used to store the contact maps at multiple resolutions. See Excel table.

Supplemental Experimental Procedures

In this paper, we describe Juicer, a one-click system for automatic Hi-C data analysis and feature annotation. The tools that comprise the system were largely described in the supplement to our 2014 Cell paper (Rao and Huntley et al., 2014). Here, we will summarize the pipeline and detail some small refinements made since that paper was published.

Sequence data to normalized Hi-C contact matrices. The first portion of the pipeline, where raw sequence data is converted to a .hic file containing the normalized Hi-C contact matrices, proceeds exactly as described in the supplement to (Rao and Huntley et al., 2014). The raw sequence data is aligned in chunks in parallel as single end reads using an updated version of BWA, *bwa mem* (Li, 2013). Each chunk is then sorted by read name and the single end alignments are paired again. Ambiguous chimeric reads (those that align to more than two places in the genome) are discarded. Each read is assigned its corresponding restriction fragment based on the restriction enzyme. The read pairs are then sorted by position and the chunks are merged together into one large file for duplicate and near-duplicate removal. The result serves as input to the Juicebox tools "pre" command, which bins the reads at multiple different resolutions and normalizes each matrix at each resolution. The matrices are stored in the ".hic" file format, a compressed binary format that allows Juicebox fast random access to different matrices at different resolutions. The .hic file format is described in detail below. For extensive explanation about the above steps, please refer to the supplement to (Rao and Huntley et al., 2014).

Arrowhead. The Arrowhead matrix transformation described in (Rao and Huntley et al., 2014) annotates genome-wide contact domains. The original algorithm was implemented in MATLAB; the Juicer implementation is in Java. With the exception of minor bug fixes and stability improvements, the details of the algorithm remain the same. As described by (Rao and Huntley et al., 2014), the Arrowhead transformation for annotating contact domains is defined as $A_{i,i+d} = (M^*_{i,i-d} - M^*_{i,i+d}) / (M^*_{i,i-d} + M^*_{i,i+d})$. This is equivalent to calculating (1 - observed/expected) for *(i,i+d)*, where the expected is the average of values at *(i,i+d)* and *(i,i-d)*. Positive values for $A_{i,i+d}$ indicate locus *(i-d)* is within the domain, whereas negative values indicate locus *(i+d)* is within the domain; values near zero indicate both loci are within or outside a domain. As in the original algorithm, the Juicer implementation leverages dynamic programming to calculate the domains in $O(n^2)$ time. Upon a successful run of the Arrowhead algorithm, a file containing the contact domains will be produced in the output directory that can be loaded into Juicebox.

HiCCUPS. HiCCUPS (Hi-C Computational Unbiased Peak Search) is a sophisticated local peak caller that integrates information from multiple local neighborhoods in its peak calling procedure. The original algorithm was implemented using Python/pyCUDA; the Juicer implementation is in Java/JCUDA. With the exception of minor bug fixes and stability improvements, the details of the algorithm remain the same. As described by (Rao and Huntley et al., 2014), HiCCUPS examines each pixel in a Hi-C contact matrix and identifies those with enriched contact frequencies relative to local neighborhoods (pixels to its lower-left, pixels to its left and right, pixels above and below, and pixels within a "donut" surrounding the pixel of interest), after accounting for larger structural features and correcting for multiple hypothesis testing. Juicer provides a JCUDA-based (Yan et al., 2009) implementation of the HiCCUPS GPGPU algorithm.

Clustering of enriched pixels found at 5KB, 10KB, and 25KB is conducted as described by (Rao and Huntley et al., 2014). The Juicer implementation allows extensive customization of parameters involved in locating enriched peaks, such as FDR thresholds and local window widths, and parameters involved in post processing enriched peaks, such as the initial clustering radius and observed/expected thresholds for filtering of peaks.

Upon a successful run of the HiCCUPS implementation, a directory will be created containing:

- Separate enriched pixel lists for every resolution specified
- Separate FDR threshold lists for every resolution specified
- Separate post-processed loop lists for every resolution specified
- A merged post-processed loop list

Motif Finder. Motif Finder determines the unique and inferred motifs responsible for creating a chromatin loop in the manner described in (Rao and Huntley et al., 2014) and validated using CRISPR Cas-9 editing by (Sanborn and Rao et al., 2015). By default, Motif Finder supports motifs for hg19, hg38, mm9, and mm10 using the M1 motif (Schmidt et al., 2012) and FIMO (Grant et al., 2011). Motif Finder requires a set of ChIP-Seq tracks in locating unique and inferred CTCF motifs for given loop list. (Rao and Huntley et al., 2014) describe in detail the process of using CTCF, RAD21, and SMC3 ChIP-Seq tracks to locate unique and inferred CTCF motifs.

Potential CTCF motifs across provided genomes are available at *http://hicfiles.s3.amazonaws.com/internal/motifs/*GENOME_ID.*motifs.txt* (e.g. *http://hicfiles.s3.amazonaws.com/internal/motifs/hg19.motifs.txt*). Motif Finder can also accept custom provided motif lists following FIMO output format (Grant et al., 2011).

The original algorithm used bedtools and STORM (Schones et al., 2007). The Juicer implementation includes a Java port of the required bedtools functions (merge and intersect; Quinlan and Hall 2010). FIMO (Grant et al., 2011) was used instead of STORM for improved capability in finding accurate motifs. Motif Finder results were validated biologically by (Sanborn and Rao et al. PNAS 2015).

APA. APA (Aggregate Peak Analysis) tests the aggregate enrichment of an entire set of putative peaks as described by (Rao and Huntley et al., 2014). APA is especially useful for assessing low-resolution Hi-C maps, where individual peaks may be difficult to discern, but where the aggregate signal should be detectable if the peak set is reliable and the Hi-C experiment was successful. The original algorithm was implemented using Python; the Juicer implementation is in Java. With the exception of minor stability improvements, the details of the algorithm remain the same.

Other improvements. HiCCUPSDiff is a Java wrapper to the HiCCUPS algorithm that allows users to find significant differences in loop lists given the original .hic files and respective loop lists. As detailed in (Rao and Huntley et al., 2014), differential loops are only called if a loop is annotated for one file, no overlapping loop was identified in the other file, and the peak pixel displays less than 1.3-fold enrichment over all local neighborhoods in the other file. We have also added our script for creating diploid Hi-C maps from a variant call format (VCF) file containing phased SNPs. The details of the diploid algorithm are unchanged from (Rao and Huntley et al., 2014).

Cluster implementation. We compared the performance of three different cluster systems running the Juicer pipeline for processing Hi-C data: Amazon Web Services (AWS) Intel-based OpenLava cluster, Rice IBM Power8-based SLURM cluster, and Broad Institute Intel-based Univa GE (UGER) cluster. As detailed in Table 1, these clusters are each using different operating systems, hardware configurations, and cluster management software; performance is determined by unique combination of software and hardware in each system. In particular, AWS is a dedicated system, whereas the performance of the clusters at Rice and Broad is affected by how many other users are sharing resources. We provide the code for Juicer on all three systems in our GitHub repository.

We used the public IMR90 dataset from (Rao and Huntley et al., 2014), which contains more than 1.5 billion paired-end reads. We generated contact maps down to 5kb resolution together with the list of contact domains returned by Arrowhead and the list of loops returned by HiCCUPS.

Juicer first splits the reads into subsets of 90 million reads each, and aligns them in parallel to the human genome reference. Ligations are also counted during the alignment phase. After alignment, each pair is merged into single sorted files and these are then merged into one large file. In the next step, duplicates are removed in parallel. The Hi-C Creation phase consists of using the resulting file to calculate statistics and create the normalized contact maps, which are stored in the .hic file. Features are then annotated using the .hic file as input. On systems without GPUs, only contact domains are annotated.

To illustrate the capabilities of Juicer, we used it to re-analyze all 15,976,316,772 paired-end reads (7.3 TB) from GM12878 human lymphoblastoid cells generated in a recent Hi-C study by our lab (Rao and Huntley et al., 2014). We processed this dataset using different aligners (*bwa sw* (Li et al., 2010) and *bwa mem* (Li,

2013)) and different reference genomes (hg19 and hg38), generating a map containing ~8.9B contacts in each case.

Beta testing. The three cluster implementations of Juicer have been available since January 2016 on GitHub for beta testing. In the past month, we have also established a Google groups forum for answering questions. There were over 50 different users generating over 150 different posts, indicating a high level of interest and user adoption of our software. Beta users have been able to successfully run Juicer on all three different cluster implementations. We used the feedback we received through the forum to improve Juicer functionality and fix bugs.

Comparison to other Hi-C pipelines. There have been several Hi-C pipelines published recently (Castellano et al., 2015; Schmid et al., 2015; Servant et al., 2015; Suria et al., 2015). Most will map reads to a genome (though some do not properly handle chimeric reads), bin the Hi-C contacts at a single resolution, and normalize the resulting Hi-C contact map. However, the output of the pipelines is usually a single matrix at a single resolution in sparse or dense text format, which takes up a lot of space on disk and requires rerunning the pipeline to produce Hi-C contact maps at different resolutions. Most also cannot handle the large datasets that are now becoming standard for a Hi-C experiment. Finally, none of the other pipelines offer automatic feature annotation. Table S1 lists some of the other pipelines and their capabilities.

Specialized file format. The .hic file format was created by Jim Robinson for Juicebox (Durand and Robinson et al., 2016) and is specially designed to provide fast random access to any contact matrix at any resolution. The footer of a .hic file contains pointers to all the matrices, together with the size (in bytes) of the matrix. To quickly access a matrix at a particular resolution, the reader looks up the pointer in the footer then reads all the data from that matrix into main memory. Data is also stored in "blocks" of adjacent contacts, enabling fast visualization in two dimensions. This design makes it possible to visualize billions of Hi-C contacts quickly and zoom in to extremely high resolution in real time in Juicebox. All of the subsequent tools in the Juicer pipeline run on .hic files.

Table S2 contains the file format. The header portion starts with a field identifying the file as .hic format and establishing the version number (currently 8). The master index position (the location in bytes in the file where the reader can look up the position of all the matrices) follows. The subsequent fields of genomeID, attribute dictionary, chromosome dictionary, base-pair resolution dictionary, and fragment resolution dictionary are important meta-data used to properly read and render the matrices.

The body of the .hic file contains the data for every chromosome-chromosome combination; if there are N chromosomes, there will be N^2 matrices represented. Each matrix starts with its unique chromosome-chromosome combination, followed by the number of resolutions stored. Then for each resolution, there's a header portion with the metadata necessary to read the matrix followed by the matrix data, stored in block format to ensure good locality. That is, each matrix is subdivided into sub-matrices, called blocks. The "block column count" is the number of columns for the block grid. The "block bin count" is the size of each block in bins, or pixels. Since the blocks are square, their size is blockBinCount x blockBinCount. Finally, each block contains the cell data: the actual count data for that block at that resolution, stored in sparse format.

The footer of the .hic file format contains pointers to each matrix, ensuring fast lookup and direct access to the data without reading through the entire file. It also contains the expected value vectors, which enable calculation of observed/expected for all the matrices. Following the expected value vectors are the normalized expected value vectors, used for the observed/expected for normalized matrices, and the normalization vectors. There are two main types of normalization: VC, or vanilla coverage, and KR, or Knight-Ruiz balanced. The normalization vectors are used with the count to calculate and display normalized matrices.

The code for reading and writing the .hic file format is available at the Juicebox GitHub repository; consult the class DatasetReaderV2 in package juicebox.data for reading and the class Preprocessor in package juicebox.tools.utils.original for writing.

Availability. Juicer is an open source project available at github.com/theaidenlab/juicer. All the .hic files from our recent publications have been added to their corresponding GEO accession.

Supplemental References

Durand, N. C., Robinson, J.T., Shamim, M.S., Machol, I., Mesirov, J.P., Lander, E.S., and Aiden, E.L. (2016). Juicebox: a visualization system for Hi-C contact maps with unlimited zoom. Cell Systems.

Li H. (2013). Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. arXiv:1303.3997v1 [q-bio.GN].

Li H. and Durbin R. (2010) Fast and accurate long-read alignment with Burrows-Wheeler Transform. Bioinformatics, Epub. [PMID: 20080505]

Quinlan, A. R., and Hall, I.M. (2010). BEDTools: a flexible suite of utilities for comparing genomic features. Bioinformatics, 26(6), 841-842.

Schones, D. E., Smith, A.D., and Zhang, M.Q. (2007). Statistical significance of cis-regulatory modules. BMC bioinformatics 8(1), 19.

Yan, Y., Grossman, M., and Sarkar, V. (2009). JCUDA: A programmer-friendly interface for accelerating Java programs with CUDA. Euro-Par 2009 Parallel Processing. Springer Berlin Heidelberg, 887-899.