# Celera Assembler:

# Prototype Chunk Graph Builder Manual

Version 1.0

*Clark Mobarry*

## 1 Overview

The Celera Chunk Graph Builder is the module of the Celera Assembler that implements the graph reduction algorithms discussed in "Toward Simplifying and Accurately Formulating Fragment Assembly" by Eugene Myers. The CGB digests the output from the overlap detector module and constructs a graph to represent the interrelationships between the fragment reads. The CGB maintains a database of fragments reads and fragment overlaps in terms of graph data structures.

## 2 Memory Usage

The memory usage of the CGB entirely self-contained. Currently, the memory usage of the prototype is 16 bytes per fragment overlap C-structure and 56 bytes per fragment read C-structure. (The C-structure for an overlap needs only 13 bytes or less, and the C-structure for the fragment reads needs 55 bytes or less, but padding is necessary for 64 bit address alignment.) We anticipate that haploid genomes will have about 10 overlaps per fragment and diploid genomes will have about 20 overlaps per fragment. Thus, the prototype's memory usage scales at about 376 bytes per fragment. The Human genome will be sequenced with 70 million fragment reads will need about 27 GBytes of virtual memory in the working set of virtual pages for the current prototype.

## 3 Interface

### 3.1 Command Line Interface

The command line interface for the CGB is

```
$ unitigger -v max_verts [-fl-a] datastore dataset
```

The integer *max_verts* is an array length sufficient for the new and old fragment reads. The *datastore* is a directory with the check-point information. The *dataset* is file name prefix to specify an input file named *dataset*.ovl. There is an output file names *dataset*.cgb and *dataset*.cga.. The input file is batch of ADT, BRC, IDT, IJN, OFG, OVL, and RPT messages. The output file is composed of ADT, IDT, IJN, and RPT pass-through messages and newly created CHK messages. The "-f" option clears the *datastore* to prepare for a new job. The "-a" option appends the new *dataset* to the check-point information.

## 3.2 Input files

The input files are *.ovl message files described in "CELERA ASSEMBLER LOGICAL ARCHITECTURE" (ProtoDesign.rtf), "PROTOTYPING I/O CONVENTIONS" (ProtoSpec.rtf), and "Celera Assembler: Prototype Assembler Message Routines and API" (ProtoMesg.rtf).

## 3.3 Persistent Store

A persistent store for the overlap records is to be implemented. Currently the store is just a checkpoint/restart facility. The store is composed of a directory with the following four ~~five~~ binary files:

- Fglobals The global scalar information.
- Fverts An array of the fragment read information.
- Fedges. An array of the fixed-length fragment overlap information.
- Fsource A segmented character array containing the simulator source strings.
- ~~Fdeltas The variable length fragment overlap information.~~

## 3.4 Output files

The output files are three types: (1) output message files to assembler components downstream from the CGB as a *.cgb file, (2) statistical diagnostic information as a *.cga file, and optionally (3) visual plots such as reduced graphs for human understanding.

The statistical information produced is:

- Fragment graph vertex types
  - All
  - Singleton
  - Inter-chunk
  - Intra-chunk
  - Contained
- Fragment graph edge types
  - All
  - Inter-chunk
  - Intra-chunk
  - Removed by transitivity
  - Removed by containment
  - Contained
- Chunk graph vertex types
  - Singleton
  - Non-singleton
- Fragment graph vertex types by chunk membership
  - Fragments in singleton chunks
  - Contained fragments in singleton chunks

    - Fragments in non-singleton chunks
    - Contained fragments in non-singleton chunks
    - Inter-chunk fragments in non-singleton chunks
    - Intra-chunk fragments in non-singleton chunks
- Portion of  genome covered by
    - Bases of singleton chunks
    - Bases of non-singleton chunks
- Portion of sampled bases in
    - singleton chunks
    - non-singleton chunks
    - essential fragments
    - contained fragments

- Histograms of chunk lengths using ρ (ρ is zero iff for a singleton chunk)
- Histogram of chunk length for CGW
    - Length in bases of singleton chunks
    - Length in bases of non-singleton chunks
- Histogram of statistical unique/repeat classification versus the number of fragments
    - True uniques determined by simulator information
    - True repeats determined by simulator information
    - At different coverage levels
- Histogram of the number of edges per vertex.
- Histogram of  (max(npx,nsx)/(npx+nsx)) for each vertex.
    - All chunks
    - Repeat chunks only
    - Unique chunks only
    - Ambiguous chunks only
    - Near a repeat boundary (<40 bp)
-

The reduced graphs are produced in the "term representation" for the graph visualization tool DaVinci (file:/usr/local/src/DaVinci/daVinci_V2.1/docs/overviewF.html). The most important feature for the CGB visualization is that all edges must be defined when a vertex is defined. This implies that a list of edges adjacent to a vertex must be available by the time that the graph files are written.

## 4 Design

See cds/AS/doc/Assembler/Designs/CGBModule.rtf (the design document) for a description of the algorithms used. Also, see the documentation for the Celera Assembler messaging routines in cds/AS/doc/Assembler/SoftwareModules/ProtoMesg.rtf. The prototype chunk graph builder is written using ANSI C. Regression tests are supported by UNIX diff on ASCII files.

### 4.1 Incremental Operation

The human genome is large, about 3.5 Gbp = 1 GBytes. The Celera assembler will expect 10 times coverage sampling of the genome on average. Thus, there will be an average of 9 fragment prefix overlaps and 9 fragment suffix overlaps. This means there will be 18 edges adjacent to each vertex on average. In addition, we assume that there will be 450-550 bp per fragment read. The assembler needs to ingest about 200000 fragments per day staring 99/04. Thus, the Celera assembler must be built for incremental operation.

### 4.2 Universal Truths

A guiding principle for the Celera Assembler is the definitions must come before reference. Likewise, all references must be removed before the definition.

### 4.3 Checkpoint/Incremental Input/Surveys

The CGB operates in an incremental mode by accepting a *.ovl file and a checkpoint of the labeled fragment read and overlap arrays.

## 4.4 Chunk graph analyzer

Currently the chunk graph analyzer is part of the CGB executable. In particular, some questions to be answered are:

- We are concerned about how the chunk graph evolves as the coverage increases.
- What is the compression factor for the fragment overlap graph by using the chunk graph?
- How does the unique/repeat classification statistic improve as the coverage increases, for example 1x, 2x, 4x, 6x, 8x, 10x coverage. The quality of the classification statistic is judged by using the information from the simulator. If any overlaps in the chunk are from non-adjacent fragments, then the chunk is considered truly a repeat.
- How consistent are the mate link distances (10% or 20%) between individual fragment read in two chunks with the distances computed with the chunks?
- What is the effect of using 20-mers instead for 24-mers?
- How does the number of overlaps, time for execution, and other statistics vary with coverage and genome size for the optimized executables? In particular, find the scaling for OVL and CGB using the 1/1000th and 1/100th human simulations in the incremental execution mode.

### 4.5 Optimization

The execution performance of the CGB will probably be enhanced by using conformal arrays rather than an array of structures for the fragment/vertex information. Unlike the edge data, loops over the vertices usually use only a small amount of the vertex information. This means that we are guaranteed to waste a significant portion of the main memory to cache bandwidth on unused data.

Currently the CGB does not currently use multiple threads.

## 5 Limitations

If two fragment reads overlap each other with zero overhang, then the CGB arbitrarily chooses the newer fragment as contained in the older fragment (contained_vertex_marking).

~~The IID of zero is a sentinel for no mate.~~

To allow fragment delete messages to be deferable, the fragment IIDs should not be reused. ~~Any fragment delete messages invoke a reassessment of contained and transitively removed edges.~~

The chunk classifier could conceivably produce a cyclic chunk. The graph transversal routine currently assumes that each non-singular chunk has two ends.

Do we allow a fragment to overlap with itself?

There are violations of transitivity of containment edges. That is to stay there are "contained" fragments that have containment edges only from other "contained" fragments. We should investigate modifying the definition of "contained".

## 6 Status

Here is the "to do" list:

- Process the BRC messages and compare with the simulator annotations when the mutation rate is zero.
- Produce the abr and bbr fields in the CHK messages and compile statistics versus simulator information whether the fragment read is truly a unique or repeat.
- Resolve the "not-found" contained fragments due to the non-transitivity of containment.
- The scaling to large problem sizes is still an issue.
- The edge data should be accessed through inline routines.
- All functions need comments of the arguments and parameters.
- All functions should have about the same amount of code as two columns of enscript output
- Must finish the output for the chunk graph walker, the chunk branch point information is not being produced.
- Perform a preliminary scaling analysis for the elapsed time of each phase in the

computation versus problem size and type.

- Need to put the delta encoding into the overlap store.
- Should count the number of repeat chunks with branch points.
- Design a Chunk Graph Analyzer
    - Take the output of the ORA as input to the CGB to produce a chunk graph with no inter-chunk edges.
    - Check to see that the chunk graph is consistent with the layout from the simulator.
    - Check every overlap to see if it is consistent with the current chunk graph. In particular, the labeling of each edge and the labeling of the edge's endpoints.
    - Make sure that the behavior is correct for circular chunks.
- Does the CGB need to produce a range of possible coordinates for the 5' and 3' tips of the fragment reads in a chunk?
- Consider a check point scheme for power failures.
- Implement an AS_DELETED status for fragment reads and an AS_REMOVED_BY_DELETION for fragment overlaps.
- Chimera fragment detection.
- Crappy fragment detection.
- Are the multiple overlaps between a pair of fragments?
- Check the behavior for multi-edges between a pair of vertices.
- Remove the "-v max_verts" command line.
- Use realloc to resize the queue in the graph traversal subroutine.

## 7 Component Architecture and Unit Dependencies

The source comprises a directory with make, header, and source files:

- Makefile,
- AS_CGB_all.h: the catch all header file,
- AS_CGB_main.c: the program driver,
- AS_CGB_proto.c: the fragment graph building routines,
- AS_CGB_cgb.[ch]: the chunk graph building routines,
- AS_CGB_io.c: the input file reading routine,
- AS_CGB_store.[ch]: the check pointing routines and store initialization,
- AS_CGB_cga.[ch]: the chunk graph analysis routines,
- AS_CGB_histo.[ch]: the fancy histogramming routines,
- AS_CGB_transverse.[ch]: the breadth first search through the graph to file the weakly connected components.

The imported include files are

- Cds.h : The bit length specific type definitions for integers .

- AS_MSG_pmesg.h : The assembler's prototype I/O routines.
- AS_UTL_Var.h: The variable length array package. The routines used are CreateVA_Type, DeleteVA_Type, GetType, TouchType, and macros VA_DEF, and VA_TYPE.

Authors:
Clark Mobarry
Created : 12/14/98
Last revised: 07/05/07 Split of the manual from the design document.