Celera Assembler:

Prototype Chunk Graph Builder Design

Version 1.0

Clark Mobarry

Abstract

This document describes the architecture of the chunk graph builder (CGB).

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 fragment overlap graph to represent the interrelationships between the fragment reads. The fragment reads (OFG records) and fragment overlaps (OVL records) form the vertices and edges of the fragment overlap graph. The fragment reads have adjacent overlaps of two types: those that are 3' (suffix) overlaps and those that are 5' (prefix) overlaps. The vertices are dual ported, one port for the prefix overlaps and one port for the suffix overlaps. The edges are labeled as a whole as dovetail or containment, and the edges are labeled at both ends with overlap orientation information. "The essential property of a dovetail edge is that when one of its arrows is directed into a read, the prefix of the read is in the overlap, and when directed out, the suffix of the read is in the overlap."

The CGB module has a series of graph transformation functions. The initial fragment overlap graph is formed from all of the fragment reads and fragment overlaps. This graph has much more complexity than necessary to represent all possible assemblies of the fragments. The initial graph (G0) is transformed into three intermediate graphs by marking the vertices and edges. The G1 graph has all "deleted" and "contained" vertices and their adjacent edges marked. The remaining vertices are considered essential. The G2 graph has all remaining edges marked as essential or as transitively removable edges. The G3 graph has all essential edges marked as inter-chunk or intra-chunk edges, and the essential vertices marked as a single fragment chunk, an inter-chunk vertex, or an intra-chunk vertex.

2 Design

The human genome is large, about 3.5 Gbp. The Celera assembler will expect 10 times coverage sampling of the genome on average. Thus, there will be on average 9 fragment

prefix overlaps and 9 fragment suffix overlaps. This means there will be 18 edges adjacent to a vertex on average. In addition, we assume that there will be 450 to 550 base pairs per fragment read. The assembler needs to ingest about 200000 fragments per day staring 99/04. The complete data set will have about 70 million fragment reads. Thus, the Celera assembler must be built for incremental operation.

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

The transitive edge removal graph operation of the CGB requires an adjacency-list data structure. ~~For incremental operation, the adjacency list of new fragments will house the overlap edges to older fragments. This will minimize editing of the adjacency lists of older fragments.~~

> UNIVERSAL TRUTH: The CGB will assume that the IIDs are issued in order by the Gatekeeper module of the Celera Assembler and that a new batch of fragments and edges will include all overlaps between the fragments and the old fragments.

The CGW module seems to require (1) an undirected graph data structure or (2) the edges must be ordered along the chunk. This is because the fragment reads and overlaps arrive in random order. A linear chain of the essential edges represented by arbitrary directed edges with random directions would be hard to walk.

The current design of the CGB has components that scale as:

- (N_V) : operations on the vertices. This is a very common loop type.
- (N_V N_A) log (N_V N_A) : a sort over the edges. The G0 graph computation uses a sort to compute the edge adjacency for each vertex.
- (N_V N_A) : operations on the edges. The G1 graph computation uses this loop to mark the edges adjacent to contained fragment. This is a very common loop type.
- (N_V N_A N_A) : operations on triangles. The G2 graph computation uses loop to mark the transitively removable edges.
- N_V^2 : incremental processing tasks. Assuming that batches are roughly equal in size, some house keeping tasks will have quadratic dependency.

2.1 OVL file input

The input stream is processed one file at a time. The OFG messages are appended to the end of the current vertex array, rather than inserted into an array where the internal identifier (IID) is used as an index. The array index in to the vertex array is the VID of the fragment read. This disassociation of the fragment IID from the array index (VID) will allow vertex reordering for memory locality. The CGB maintains a permutation between the assembler IIDs in the fragment store and the chunk graph vertex ids. This

permutation is a time dependent thing that uses the current best guess for the locality of the vertex. The new OVL messages are likewise appended to the end of the current edge array. The directed pair of IIDs for the new fragment overlaps is mapped to the appropriate VIDs. The new BRC messages are stored temporarily in an array. The IIDs are mapped to VIDs and then the information is placed in the vertex array.

2.2 G0 graph: Undirected graph of all fragment reads and fragment overlaps

The most important data structures are the internal representation of a fragment read as a graph vertex and the fragment overlap as a graph edge.

The CGB overlap edge records has some arbitrary choices. In the diagrams below it is assumed that the A-fragment is on top of the B-fragment, and that overhang of the A-fragment is on the left and the overhang of the B fragment is on the left. This is the convention is consistent with the output of the overlap detector module. The overhang of the A-fragment in base pairs is denoted "ahg" and measures the number of base pairs of the A-fragment that is not in the overlap region. The overhang of the B-fragment is denoted "bhg". By convention, the graphical representation of sequences are by arrows that have the prefix at the DNA 5' end, and the suffix (arrow tip) at the DNA 3' end. For dovetail overlaps, the flag "asx"/"bsx" is true if the suffix of the A/B fragment read is in the overlap.

Dovetail overlap edge types by convention always have ahg>0 and bhg>0. This allows four possible dovetail overlap edge orientations which correspond to the four possible values of the pair (asx,bsx): ND (normal dovetail), AD (anti-normal dovetail), ID (innie dovetail), and OD (outtie dovetail). The AD orientation is introduced to facilitate graphics. Note that if we allow the A- and B-fragments to be interchanged then the AD type is unnecessary.

Normal dovetail overlap:

```
 ND A     --------->        asx=T, ahg>0
 .  B        --------->     bsx=F, bhg>0
```

Anti-normal dovetail overlap:

```
AD A^c  <---------         asx=F, ahg>0
.  B^c     <---------      bsx=T, bhg>0
```

Innie dovetail overlap:

```
ID A    ---------->        asx=T, ahg>0
.  B^c     <---------      bsx=T, bhg>0
```

Outtie dovetail overlap:

```
OD A^c  <--------          asx=F, ahg>0
.  B        --------->     bsx=F, bhg>0
```

The (..)^c notation denotes the Watson-Crick complement sequence.

The overlap detector does not emit anti-normal overlaps since they are redundant with normal overlaps if we are allowed to swap the A- and B-fragments. This classification scheme is extended to containment overlaps by allowing the B-fragment to have non-

positive overhangs. The assignment of the suffix flag is made as if the B-fragment is extended out of the overlap region on the B-end.

Containment overlap edge types by convention always have ahg>=0 and bhg <= 0. In addition, there are four possible containment overlap edges. This allows four possible containment overlap edge orientations which correspond to the four possible values of the pair (asx,bsx): NC (normal containment), AC (anti-normal containment), IC (innie containment), and OC (outtie containment). Contained overlap edge types where the dots (…) graphically represent a non-positive overhang.

Normal containment overlap:

```
NC A    ------------>    asx=T, ahg>=0
.  B       ------>...    bsx=F, bhg<=0
```

Anit-normal containment overlap:

```
AC A^c  <------------    asx=F, ahg>=0
.  B^c     <------...    bsx=T, bhg<=0
```

Innie containment overlap:

```
IC A    ------------>    asx=T, ahg>=0
.  B^c     <------...    bsx=T, bhg<=0
```

Outtie containment overlap:

```
OC A^c  <------------    asx=F, ahg>=0
.  B       ------>...    bsx=F, bhg<=0
```

Note that the AC and OC edge types are redundant with NC and IC edge type, respectively. However, the overlap detector emits both innie-containment and outtie-containment overlaps.

ASSERTION: A dovetail overlap has both overhangs positive.

ASSERTION: A containment overlap is modeled as an edge with an overhang of zero or negative length in base pairs.

ASSERTION: Only one of the two overhangs of an overlap can be negative.

ASSERTION: For overlaps where both overhangs are zero, the symmetry with be broken by the IID of the fragment reads. The fragment read with the smaller (earlier) IID contains the fragment read with the larger IID.

2.3 G1 graph: Contained fragment read and Containment overlap marking

It is important to emphasize that fragments are not labeled as contained by the overlap detector module. The deciding information is containment overlaps. That is, being a

"contained" fragment read is a derived quality from the existence of a containment overlap to that fragment read. The fragment reads that are not marked as contained are considered essential. The fragment overlaps that are not marked containment are considered possibly essential. After the contained fragments are found, every remaining overlap edge, that is not marked as deleted or containment, to a contained fragment is marked as removed by containment.

We have an unresolved issue due to non-transitivity of containment.
Suppose the edges E are directed edges composed of two disjoint sets, ED and EC, where ED are the dovetail overlaps and EC are the containment overlaps reported by the overlap detector module.

The vertices V are composed of two disjoint sets: (1) the contained fragments (VC) that are those vertices pointed to by at least one EC edge, and (2) the remaining fragments (V-VC).

There is an implementation issue caused of the non-transitivity of containment overlaps. That is V1 contains V2, and V2 contains V3, does not imply V1 contains V3. This is sometimes caused by a lousy overlap between V1 and V2. An implementation problem for the chunk graph walker is that members of V-VC can have mate-links to the members of VC that do not have EC edges from members of V-VC. Call the members of VC that do not have EC edges from members of V-VC as VI. It turns out that some members of VI naturally fit into the dovetail graph when the lousy containment edges are ignored. In addition, it would be preferable for the chunk graph builder if every contained vertex has a direct containment edge from the essential vertices.

The graph reduction can be refined by noting that EC is composed of two disjoint sets. Define ER as the members of EC that have both ends in VC. These edges were marked for removal in the original scheme. Suppose we mark them for removal as a separate step. Then the relaxed containment marking algorithm is:

1. The edges E are directed edges composed of two disjoint sets, ED and EC, where ED are the dovetail overlaps and EC are the containment overlaps reported by the overlap detector module.
2. The vertices V are composed of two disjoint sets: (1) the contained fragments (VC) that are those vertices pointed to by at least one EC edge, and (2) the remaining fragments (V-VC).
3. The edges E are directed edges composed of three disjoint sets, ED, EC-ER, and ER, where ER are the members of EC that join two members of VC.
4. The vertices V are composed of two disjoint sets: (1) the relaxed contained fragments (VR) that are those vertices pointed to by at least one (E-ER)-(EC-ER) edge, and (2) the remaining essential fragments (V-VR).

To recap, all members of VR are hanging off a V-VR vertex by an EC-ER edge, and all the members of V-VR are connected by edges of E-EC.

2.4 G2 graph: Essential overlap marking

Describe transitive edge removal marking here. We are using Gene Myers' algorithm.

We are considering Granger Sutton's transitive edge removal algorithm. When a contained fragment read is parallel to its container fragment read, a NC and AC edge pair are recorded. When a contained fragment read is anti-parallel to its container fragment read, an IC and OC edge pair are recorded.

The directed graph design would allow edge flipping. All edge that has the A-fragment IID smaller than the B-fragment IID are flipped. The edge flipping of dovetail and contained edges is possible do to the internal representation of those overlap types. The data for the A-fragment and the B-fragment are simply exchanged, then the flipped edge is put into the adjacency list of the new A-fragment.

2.5 G3 graph: Chunk graph creation

The purpose of the chunk graph representation is to explicitly reduce the complexity of the graphical representation of the possible assemblies of the genome.

Describe how the vertices and edges are classified for their position in a chunk.

Describe how the chunk graph traverser finds the number of base pairs in a chunk.
The graph (V,E) is composed of a set of vertices (V) and a set of edges (E). The number of vertices is N_V and the number of edges is N_E. A specific vertex is indexed as V[I]

A chunk has a linearly linked set of essential edges. The number of vertices is N_{VC} and the number of edges is N_{EC}. A non-cyclic chunk has $N_{EC} = N_{VC} - 1$. The $N_{B,}$ number of base pairs in fragment read chunk, is
N_B= V[1].length + sum(1,N_{VC}-1) (E[k,k+1].bhg),
And
N_B = V[N_VC].length + sum(1,N_{VC}-1) (E[k,k+1].ahg).
Note that N_VC counts all contained fragment reads for the chunk. This means that contained fragments are potentially over counted in the statistics.

The number of base pairs in a chunk with three fragment reads is
(the length of the A fragment) + (the length of the B fragment) - (the length of the overlap between A and B).
Describe how the chunk traverser finds the start points within a chunk.

ASSERTION: The 3' and 5' offsets of a contained fragment are within the 3' and 5' offsets of the containing fragment.

Assume that all of the intra-chunk fragments have been stored in the array of chunk data structures. The chunk overlap edges are determined by examining the inter-chunk fragment overlap edges of the A-fragment and B-fragment. The chunk id of the fragment at the other end of the inter-chunk edge will be available. For the A-fragment, if 5po < 3po then follow the fragment prefix overlaps, otherwise the suffix overlaps. For the B-fragment, if 5po < 3po then follow the fragment suffix overlaps.

2.6 Statistical Unique/Repeat Classification

The global arrival rate of fragment reads is e_g = N_V / N_B. The arrival rate of fragment reads local to a chunk is e_l = N_VC / ρ, where r is the length of the chunk in base pairs for the purposes of the coverage statistic. We are computing ρ as
$\rho\_0 = (1/2) \Sigma_{k=1,N\_EC}$ (E[k,k+1].ahg + E[k,k+1].bhg)
The coverage statistic is
C = e_g ρ - (N_VC - 1) ln(2)
A singleton chunk has by definition ρ = 0 and C = 0.

We also tried computing ρ as ρ_0 + (V[1].length +V[N_VC].length)/2 - (the total number of bp sampled in the chunk including contains)/N_VC

3 Chunk Graph Analysis

A measure of the false positives and false negatives for the coverage statistics is available using the simulator information. Each fragment read from the simulator has the positions of the 5' and 3' tops of the fragments. For each fragment overlap, we check to see if the overlap "ahg" and "bhg" are consistent with the true positions of the fragment read's position in the genome. If the overlap is not consistent, then the overlap was do to a repeat or was spurious. In particular for each edge in a chunk,

A.dir = (A.end > A.bgn ? +1 : -1), and
B.dir = (B.end > B.bgn ? +1 : -1).

These are the direction flags of the fragment reads relative to the global coordinates of the genome. The global coordinates of the A-ward and B-ward tips of the A- and B-fragments of edge E are

A.atip = (E.asx ? A.bgn : A.end)
A.btip = (E.asx ? A.end : A.bgn)
B.atip = (E.bsx ? B.end : B.bgn)
B.btip = (E.bsx ? B.bgn : B.end)

A test for consistency is

(B.atip - A.atip) == A.dir * E.ahg +- error
(B.btip - A.btip) == B.dir * E.bhg +- error

The CGB accepts increment input by writing a checkpoint file of the labeled fragment read and overlap arrays. 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 Global Data Structures

The memory usage of the CGB is dominated by the arrays defined and allocated in the `main` routine.

```
int novl;   /* The number of overlap records read. */
int nofg;   /* The number of fragment records read. */
int nidt;   /* The number of distance records read. */
int nedge;  /* The number of edges in use. */
int nvert;  /* The number of vertices in use. */

Tedge edges[MAXEDGES];/* The internal representation of the
                                    overlaps. */

Tvertex verts[MAXREADID];/* The internal representation of
                                      the fragment reads. */
```

BranchMesg thebrc[MAXVERTS]

```
typedef struct {
  int32        avx,bvx;   /* should be unsigned, currently
zero biased */
  int16       ahg,bhg;
```

```
  int8        asx,bsx;  /* can be but a bit */
  int8        nes;      /* The edge labeling */
  int8         padding;  /* Pad to 8 words == one D cache
line */
} Tedge;


int vertex_visited[MAXVERTS]; /* for graph transversal */
int vertex_tmp[MAXVERTS];
IntFrag_ID afr_to_avx[MAXREADID]; /* This array is used
to find the current vertex id (avx) from a known fragment
read IID. */
```

5 Optimization

The execution performance of the CGB will be enhanced by using conformal arrays rather than an array of structures for the 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.

This CGB does not currently use multiple threads.

6 Limitations

The maximum allocation necessary to store the old and new fragments and overlaps must be specified by the "-v maxverts" command line option.

If two fragment reads overlap each other with zero overhang, then it is arbitrarily chosen that the newer fragment is marked as contained in the older fragment (contained_vertex_marking).

To allow fragment delete messages to be deferable, the fragment IIDs should not be reused. The delete messages will 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 itsself?

Authors:
Clark Mobarry
Created : 12/14/98
Last revised: 07/05/07 CMM changed the vertex data structure to be an array of structures.

$Source: /cvsroot/wgs-assembler/doc/Designs/ChunkGraphBuilder.rtf,v $
$Revision: 1.1.1.1 $
$Date: 2004/04/14 13:42:40 $
$Name: $
$Author: catmandew $
$Log: ChunkGraphBuilder.rtf,v $
Revision 1.1.1.1 2004/04/14 13:42:40 catmandew
Initial import

Revision 1.1 2004/02/10 14:10:23 dewim
Initial release0 of the assembler source code.
Compiles on aix, tru64/osf, and linux.
Runs a006 on aix but not on tru64/osf or linux.

$Source: /cvsroot/wgs-assembler/doc/Designs/ChunkGraphBuilder.rtf,v $
$Revision: 1.1.1.1 $
$Date: 2004/04/14 13:42:40 $
$Name: $
$Author: catmandew $
Revision 1.4 1999/03/10 18:19:05 cmobarry
$Source: /cvsroot/wgs-assembler/doc/Designs/ChunkGraphBuilder.rtf,v $
$Revision: 1.1.1.1 $
$Date: 2004/04/14 13:42:40 $
$Name: $
$Author: catmandew $
*** empty log message ***
$Source: /cvsroot/wgs-assembler/doc/Designs/ChunkGraphBuilder.rtf,v $
$Revision: 1.1.1.1 $
$Date: 2004/04/14 13:42:40 $
$Name: $
$Author: catmandew $