Chapter 14. APEG generation

Authors: *Soonhoi Ha*

Since all code generation domains depends on the SDF domain, and the same routine is needed by a specialized loop scheduler in the SDF domain ($PTOLEMY/src/domains/sdf/loopScheduler), the source of APEG generation is placed in $PTOLEMY/src/domains/sdf/kernel.

An APEG graph (an ExpandedGraph class) consists of EGNodes and EGGates. Class EGNode represents an object corresponding to an invocation of a DataFlowStar (DataFlowStar is a base class of SDFStar class). An EGNode has a list of EGGates. EGGate class is similar to PortHole class in the respect that it is an object for connection between EGNodes. Between two EGGates, there exists an EGArc object. All connections in an APEG graph is homogeneous. If there is a sample rate change on an arc in the SDF program graph, the arc is mapped to several homogeneous arcs. APEG generation routines are defined as member methods of the ExpandedGraph class.

Refer to "Class ExpandedGraph" on page 14-5, for he main discussion of APEG generation.

14.1 Class EGArc

Class EGArc contains the information of (1) sample rate of the arc and (2) the initial delay on the arc.

`EGArc(int` *`arc_samples`*, int *`arc_delay`*);

The constructor requires two arguments for sample rate and the number of initial delays on the arc.

```
int samples();
int delay();
```

These functions return the sample rate of the arc, and the initial delay on the arc. We can increase the sample rate of the arc using the following method

`void addSamples(int` *`increments`*);

There is no protected members in Class EGArc.

14.2 Class EGGate

Class EGGate is a terminal in an EGNode for connection with other EGNodes. A list of EGGates will become a member of EGNode, called `ancestors` or `descendants` based on the direction of connection.

14.2.1 EGGate public members

`EGGate(EGNode*` *`parent`*, PortHole* *`pPort`*);

Is a constructor. The first argument is the EGNode that this EGGate belongs to, and the second argument is the corresponding porthole of the original SDF graph.

`const PortHole* aliasedPort();`
`const char* name() const;`

The above methods returns the corresponding porthole of the original SDF graph, the name of the porthole.

`int isItInput();`

Returns TRUE or FALSE based on whether the corresponding porthole is an input or not.

`void allocateArc(EGGate*` *`dest`*, int *`no_samples`*, int *`no_delay`*);

The method creates a connection between this EGGate and the first argument by allocating an arc with information from the second and the third arguments. It should be called once per connection.

`int samples();`
`int delay();`
`void addSamples(int` *`increments`*);

These methods call the corresponding methods of the EGArc class if an arc was already allocated by `allocateArc`.

`EGGate* farGate();`
`EGNode* farEndNode();`
`DataFlowStar* farEndMaster();`
`int farEndInvocation();`

The above methods query information about the other side of the connection: EGGate, EGNode, the original DataFlowStar that the EGNode points to, and the invocation number of the EGNode.

`StringList printMe();`

It prints the information of the arc allocated: the sample rate and the initial delay.

`void setProperty(PortHole*` *`pPort`*, int *`index`*);

This method sets the pointer to the corresponding porthole of the original SDF graph and the index of the EGGate. Since multiple EGGates in an EGNode may be mapped to the same porthole in the original SDF graph, we order the EGGates by indices.

`void setLink(EGGateLink*` *`p`*);
`EGGateLink* getLink();`

Since the list of EGGates is maintained as a derived class of DoubleLinkList, an EGGate is assigned an EGGateLink that is derived from the DoubleLink class. These methods set and get the assigned EGGateLink.

`void hideMe(int` *`flag`*);

If the initial delay is greater than or equal to the sample rate in an EGArc, the precedence relationship between the source and the destination of the arc disappears while not removing the arc from the APEG. This method removes this EGGate from the access list of EGGates (`ancestors` or `descendants`), and stores it in the list of hidden EGGates (`hiddenGates`) of the parent EGNode . If the argument flag is NULL, it calls the same method for the EGGate of the other side of connection. By default, the flag is NULL.

`virtual ~EGGate();`

Is a virtual destructor that deletes the allocated arc, removes itself from the list of EGGates.

14.2.2 Class EGGateList

This class, derived from DoubleLinkList, contains a list of EGGates. An EGGate is assigned to an EGGateLink and the EGGateList class accesses an EGGate through the assigned EGGateLink.

The following ordering is maintained in the precedence list: entries for the same far-end EGNode occur together (one after another), and they occur in order of increasing invocation number. Entries for the same invocation occur in increasing order of the number of delays on the arc.

class EGGateLink

`EGGateLink(EGGate*` *e*);

The constructor has an argument for an EGGate.

`EGGate* gate();`
`EGGateLink* nextLink();`

These methods return the corresponding EGGate and the next link in the parent list.

`void removeMeFromList();`

Removes this link from the parent list.

EGGateList public members

Class EGGateList has a default constructor.

`void initialize();`

This method deletes all EGGates in the list and initialize the list. It is called inside the destructor.

`DoubleLink* createLink(EGGate*` *e*);

Creates an EGGateLink for the argument EGGate.

`void insertGate(EGGate*` *e*, int *update*);

This method insert a new EGGate into the proper position in the precedence list. The update parameter indicates whether or not to update the arc data if an EGGate with the same far-end EGNode and delay, already exists. If *update* is 0, the argument EGGate will be deleted if redundant. If 1, the arc information of the existing EGGate will be updated (sample rate will be increased). When we insert an EGGate to the `descendants` list of the parent EGNode, we set *update* to be 1. If the EGGate will be added to the `ancestors,` the variable is set 0.

```
StringList printMe();
```
Prints the list of EGGates.

Iterator for EGGateList

Class EGGateLinkIter is derived from class DoubleLinkIter. The constructor has an argument of the reference to a constant EGGateList object. It returns EGGates. This class has a special method to return the next EGGate connected to a new `farEndMaster` that is different from the argument DataFlowStar.

```
EGGate* nextMaster(DataFlowStar* master);
```

14.3 Class EGNode

Class EGNode is a node in an APEG, corresponding to an invocation of a DataFlowStar in the original SDF graph. The constructor has two arguments: the first argument is the pointer to the original Star of which it is an invocation, and the second argument represents the invocation number. The default value for the invocation number is 1. It has a virtual destructor that does nothing in this class.

An EGNode maintains three public lists of EGGates: `ancestors`, `descendants`, and `hiddenGates` .

14.3.1 Other EGNode public members

Invocations of the same DataFlowStar are linked together.

```
void setNextInvoc(EGNode* next);
EGNode* getNextInvoc();
EGNode* getInvocation(int i);
void setInvocationNumber(int i);
int invocationNumber();
```
The first two methods sets and gets the next invocation EGNode. The third method searches through the linked list starting from the current EGNode to return the invocation with the argument invocation number. If the argument is less than the invocation number of the current EGNode, returns 0. The other methods sets and gets the invocation number of the current EGNode.

```
void deleteInvocChain();
```
Deletes all EGNodes linked together starting from the current EGNode. This method is usually called at the EGNode of the first invocation.

```
StringList printMe();
StringList printShort();
```
These methods print the name and the invocation number. In the first method, the `ances-tors` and `descendants` lists are also printed.

```
DataFlowStar* myMaster();
```
Returns the original DataFlowStar of which the current EGNode is an invocation.

```
int root();
```

This method returns TRUE or FALSE, based on whether this node is a root of the APEG. A node is a root if it either has no ancestors, or if each arc in the ancestor list has enough delay on it.

`EGGate* makeArc(EGNode*` *`dest`*, int *`samples`*, int *`delay`*);

Create a connection from this node to the first argument node. A pair of EGGates and an EGArc are allocated in this method. This EGNode is assumed to be the source of the connection.

```
void resetVisit();
void beingVisited();
int alreadyVisited();
```

The above methods manipulates a flag for traversal algorithms: resets to 0, sets to 1, or queries the flag.

```
void claimSticky();
int sticky();
```

These methods manipulates another flag to indicate that the invocations of the same DataFlowStar may not be scheduled into different processors since there is a strong interdependency between them. The first method sets the flag and the second queries the flag.

14.3.2 EGNodeList

Class EGNodeList is derived from class DoubleLinkList.

`void append(EGNode*` *`node`*);
`void insert(EGNode*` *`node`*);

These methods appends or inserts the argument EGNode to the list.

```
EGNode* takeFromFront();
EGNode* headNode();
```

The above methods both returns the first EGNode in the list. The first method removes the node from the list while the second method does not.

There is a iterator class for the EGNodeList class, called EGNodeListIter. It returns the EGNodes.

14.4 Class ExpandedGraph

Class ExpandedGraph has a constructor with no argument and a virtual destructor that deletes all EGNodes in the graph.

The major method to generate an APEG is

`virtual int createMe(Galaxy&` *`galaxy`*, int *`selfLoopFlag`*);

The first argument is the original SDF galaxy of which the pointer will be stored in a protected member `myGal`. The second argument enforces to make arcs between invocations of the same star regardless of the dependency. The procedure of APEG generation is as follows.

5. Initialize the APEG graph.

`virtual void initialize();`

Does nothing here, but will be redefined in the derived class if necessary.

6. Allocate all invocations (EGNodes) of the blocks in the original SDF graph. Keep the list of the first invocations of all blocks in the protected member `masters.`

`virtual EGNode *newNode(DataFlowStar*` *`star`*, int *`invoc_index`*);

Is used to create an invocation of a DataFlowStar given as the first argument. The second argument is the invocation number of the node. This method is virtual since the derived ExpandedGraph class may have derived classes from the EGNode class.

7. For each star in the original SDF graph,

(3-1) Make connections between invocations of the star if any one of the conditions is met: *`selfLoopFlag`* is set in the second argument, the star has internal states, the star accesses past values on its portholes, or the star is a wormhole. The connection made in this stage does not indicate the flow path of samples, but the precedence relation of two EGNodes. Therefore, EGGates associated with this connection are not associated with portholes in the original SDF graph. If the connections are made, the `claimSticky` method of EGNode class is called for each invocation EGNode. If any such connection is made, the APEG is said not-parallelizable as a whole: A protected member, `parallelizable,` is set FALSE.

(3-2) For each input porthole, get the far-side output porthole and make connections between invocations of two DataFlowStars. A connection in the original SDF graph may be mapped to several connections in the APEG since the APEG is homogeneous.

8. Find the root nodes in the APEG and stored in its protected member `sources.`

`void insertSource(EGNode*` *`node`*);

Inserts the argument EGNode into the source list, `sources,` of the graph.

All protected members are explained above.

14.4.1 Other ExpandedGraph public members

`int numNodes();`
This method returns the number of total nodes in the APEG.

`virtual StringList display();`
Displays all EGNodes by calling `printMe` method of EGNode class.

`virtual void removeArcsWithDelay();`
This method hide all connections that have delays on them. When an APEG is created, the number of initial delays on an arc, if exists, is always greater than or equal to the sample rate of the arc. Therefore, this method is used to make the APEG actually acyclic.

14.4.2 Iterators for ExpandedGraph

There are three types of iterators associated with an ExpandedGraph: EGMasterIter, EGSourceIter, and EGIter. As its name suggests, EGMasterIter returns the EGNodes in `masters` list of the graph. EGSourceIter returns the EGNodes in `sources` list of the graph. Finally, EGIter returns all EGNodes of the ExpandedGraph.

EGMasterIter and EGSourceIter are derived from EGNodeListIter. EGIter, however, is not derived from any class. Instead, EGIter uses EGMasterIter to get the first invocation of each DataFlowStar in the original SDF graph and traverse the linked list of invocations. Thus invocations are traversed master by master.