15.10 Hu Level Scheduler

Hu's level scheduling algorithm is a simple list scheduling algorithm, in which a node is assigned a fixed priority. No communication overhead is considered in the scheduling procedure. The code lies in `$PTOLEMY/src/domains/cg/HuScheduler`. All classes, except the HuScheduler class in this directory, are derived from the classes for the dynamic level schedulers.

15.10.1 Class HuNode

Class HuNode represents a node in the APEG for Hu's level scheduling algorithm. It is derived from the DLNode class so that it has the same constructors. The level (or priority) of a node does not depend on the communication overhead.

```
int getLevel();
```

Just returns `StaticLevel` of the node.

A HuNode has two private variables to indicate the available time of the node (or the time the node becomes runnable) and the index of the processor on which the node wants to be assigned. The latter is usually set to the index of the processor that its immediate ancestor is assigned. There are five public methods to manipulate these private variables.

```
int availTime();
void setAvailTime(int t);
void setAvailTime();
void setPreferredProc(int i);
```

The first three methods get and set the available time of the node. If no argument is given in `setAvailTime`, the available time is set to the earliest time when all ancestors are completed. The last two methods get and set the index of the processor on which the node is preferred to be scheduled.

15.10.2 Class HuGraph

Class HuGraph is the input APEG for Hu's level scheduler. It redefines three virtual methods of its parent classes.

```
EGNode* newNode(DataFlowStar* s, int invoc);
```

Creates a HuNode as a node in the APEG.

```
void resetNodes();
```

This method resets the variables of the HuNodes: visit flag, `waitNum`, the available time, and the index of the preferred processor.

```
void sortedInsert(EGNodeList& nlist, ParNode* n, int flag);
```

In the ParGraph class, this method sorts the nodes in order of decreasing `StaticLevel` of nodes. Now, we redefine it to sort the nodes in order of increasing available time first, and decreasing the static level next.

15.10.3 Class HuScheduler

Class HuScheduler, derived from the ParScheduler class, is parallel to the DLScheduler class in its definition.

```
HuScheduler(MultiTarget* t, const char* log);
```

The constructor has two arguments: one for the multiprocessor target and the other for the log file name.

The HuScheduler class has a pointer to the HuParProcs object that will provide the details of the Hu's level scheduling algorithm.

```
HuParProcs* parSched;
```

This is the protected member to point to the HuParProcs object. That object is created in the following method:

```
void setUpProcs(int num);
```

This method first calls `ParScheduler::setUpProcs` and next creates a HuParProcs object. The HuParProcs is deallocated in the destructor.

```
int scheduleIt();
```

The scheduling procedure is exactly same as that of the Dynamic Level Scheduler except that the actual scheduling routines are provided by a HuParProcs object rather than a DLParProcs object. Refer to the `scheduleIt` method of class DLScheduler . Also note that the runnable nodes in this scheduling algorithm are sorted by their available time first.

```
StringList displaySchedule();
```

Displays the scheduling result textually.

15.10.4 Class HuParProcs

Class HuParProcs is derived from class DLParProcs so that it has the same constructor. While many scheduling methods defined in the DLParProcs class are inherited, some virtual methods are redefined to realize different scheduling decisions. For example, it does not consider the communication overhead to determine the processor that can schedule a node earliest. And it does not schedule communication resources. Another big difference is that Hu's level scheduling algorithm has a notion of global time clock. No node can be scheduled ahead of the global time. At each scheduling step, the global time is the same as the available time of the node at the head of the list of runnable nodes.

```
void fireNode(DLNode* n);
```

This redefined protected method sets the available time and index of the preferred processor of the descendants, if they are runnable after node *n* is completed. This is done before putting them into the list of runnable nodes (`sortedInsert` method of the HuGraph class).

```
void scheduleSmall(DLNode* n);
```

When this method is called, the node *n* is one of the earliest runnable nodes. We examine a processor that could schedule the node at the same time as the available time of the node. If the node is at the wormhole boundary, we examine the first processor only. If the node should be assigned to the same processor on which any earlier invocations were already assigned, we examine that processor whether it can schedule the node at that time or not. If no processor is found, we increase the available time of the node to the earliest time when any processor can schedule it, and put the node back into the list of runnable nodes. If we find a processor to schedule the node at the available time of the node, we assign and fire the node, then update the variables of the HuGraph. Recall that no communication overhead is considered.

Top Up Prev Next Bottom Contents Index Search
