Chapter 14. HW/SW Partitioning

Author: Choonseung Lee, Hyunok Oh, and Soonhoi Ha

Hardware-Software partitioning for embedded systems gives us an optimal architecture for a given application by minimizing the cost of the system configuration with satisfying the constraints such as cost, power consumption, etc. PeaCE provides not only automatic partitioning but also manual partitioning capability for a given architecture model.

In the partitioning step, PeaCE solves the following three problems.

1. Component selection: determine the processing elements to be used. For hardware modules, an implementation of each module should be selected.
2. Partitioning and scheduling: partition the input tasks into processing elements and perform a static scheduling to estimate the execution time of tasks.
3. Performance evaluation: evaluate the quality of solution and check whether design constraints are met.

Note that component selection and partitioning are performed at the same time to make the step a “cosynthesis step” rather than a “partitioning” step. The cosynthesis framework implemented in PeaCE is extensible and adaptable. It is a fast heuristic that is made of an iteration loop of three steps that attack the sub-problems separately. It can partition the multi-mode multi-task applications, taking into account resource sharing and loop structure. There is no limitation on the number of processing elements.

A good thing on the cosynthesis framework of PeaCE is that it can be run independently since it uses interface files for input and output. It implies that any other algorithm can be integrated into PeaCE if it uses the same interface files. Our aim is to support various cosynthesis (or partitioning) algorithms in PeaCE. Then the PeaCE can be used as a research environment to compare the various algorithms.

14.1 Inputs to the Partitioning Step

The inputs to the partitioning step are three interface files generated from the previous graph analysis step in the design flow. They can be found in $HOME/PEACE_SYSTEMS/DesingName/ directory (Figure 14-1). The following is the short description of these input files: detailed explanation is in Chapter 11.

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DesignName.xml</td>
<td>Represents the input task graph topology after looping transformation.</td>
</tr>
<tr>
<td>DesignName_TimeCost.xml</td>
<td>Represents three attributes, the execution time, power consumption and hardware area of all nodes of a task on each processor (SW) or FPGA (HW). If a node cannot be scheduled on a certain resource, the execution time is INFINITE value (-1). Note that because the partitioning algorithm in PeaCE is only considered to...</td>
</tr>
</tbody>
</table>
execution time value, other attributes are ignored at the current release.

| DesignName_mode.xml | Represents the real-time constraints of given application. The constraints are composed of the task period and deadline. These attributes are used for real-time schedulability analysis. PeaCE can support multi-mode application and each mode has its own constraints. |

---

**Figure 14-1 Input interface files for the partitioning step**

### 14.2 Hardware-Software Partitioning in PeaCE Design Flow

The target for the partitioning step is the Cosynthesis target that has the following target parameters as shown in Figure 14-2.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>display?</td>
<td>YES</td>
<td>INT</td>
</tr>
<tr>
<td>compile?</td>
<td>YES</td>
<td>INT</td>
</tr>
<tr>
<td>run?</td>
<td>YES</td>
<td>INT</td>
</tr>
<tr>
<td>debug?</td>
<td>NO</td>
<td>INT</td>
</tr>
<tr>
<td>profile?</td>
<td>YES</td>
<td>INT</td>
</tr>
<tr>
<td>profile block?</td>
<td>YES</td>
<td>INT</td>
</tr>
<tr>
<td>compile options</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>link options</td>
<td>STRING</td>
<td></td>
</tr>
<tr>
<td>multi mode multi task</td>
<td>YES</td>
<td>STRING</td>
</tr>
<tr>
<td>Deadline</td>
<td>10000</td>
<td>FLOAT</td>
</tr>
<tr>
<td>Schedulability</td>
<td>Schedule-Based</td>
<td>STRING</td>
</tr>
<tr>
<td>Partition</td>
<td>Overlapped</td>
<td>STRING</td>
</tr>
</tbody>
</table>

**Figure 14-2 The partitioning target parameters**

The most important parameters are “multi mode multi task”, “Schedulability” and “Partition”. 303
multi mode multi task

- Default: YES
- Description: It is for multi-task applications. If you want to partition the single task system, you must say “No”.

Partition

- Default: Overlapped
- Description: It represents the method of task partitioning. In PeaCE kernel, two algorithms are implemented, “Independent” and “Overlapped”. Under the “Independent” scheme, all tasks are partitioned independently assuming that each task monopolizes all resources. Then each task tries to minimize the schedule length so that utilization of the processors becomes highly unbalanced: the fastest processor will be heavily utilized. It will not be good for multi-processor systems when more than one task can run concurrently. The “Overlapped” partitioning scheme solves this problem by assigning a weight to each processor based on the current utilization of the processors. We partition the highest priority task first. After a task is partitioned, it computes the utilization of each processor. If the utilization of a processor is high, we lower the performance of the processor before partitioning the next task. “Overlapped” is default value in PeaCE.

![Partition parameter](image)

**Figure 14-3 Partition parameter**

Schedulability

- Default: Schedule-Based
- Description: “Schedulability” parameter selects the real-time schedulability analysis method of the given application. There are also two values: Utilization and Schedule-based. The “Utilization” represents the utilization-based schedulability test which is suitable only when a task monopolizes the whole system. It should be paired with “Independent” partitioning. For example, if utilization of two tasks is defined as $U(\text{Task}_1) = 0.6$, $U(\text{Task}_2) = 0.5$, then, the whole system utilization is 1.1 (0.6+0.5), which fails the schedulability test. This method is not directly applicable for multi-processor environment. Therefore you should use the “Schedule-Based” method for multi-processor system. The “Schedule-Based” method is to construct a static schedule of tasks at compile time after the TM (Timed Multitasking) model. If there exists a static schedule that meets all timing requirements, we consider the schedulability test to be passed. This scheme is paired with the “Overlapped” partitioning option. Note that the schedule-based test does not guarantee the timing correctness if the static schedule is not maintained at run time: so it is not safely applicable for RTOS based system.

![Schedulability parameter](image)

**Figure 14-4 “Schedulability” parameter**
While the current implementation has two options for the schedulability test, we will include another one in the next release for more general multi-tasking system.

### 14.2.1 Example: DIVX player

In this section, we use the DIVX player example and summarize the procedure of the partitioning step.

1. First open the schematic of schematic/Peace/Demo/Dataflow/DIVX then you can see three task nodes as shown in Figure 14-5:

   - **AviReader task**
     - Parse the video and audio data frame from the AVI file.
   - **H263FRDivx task**
     - H263 decoder algorithm specified by the fractional dataflow model.
   - **MADStream task**
     - MAD MP3 algorithm

![Figure 14-5 DIVX player design schematic](image)

2. Before run the partitioning step, please make sure that three input files are prepared (Figure 14-1).

3. Click the **archi** tab and confirm that the number of processing elements is the same as that in the TimeCost.xml file. For the DIVX player example, Set the names of processor and ASIC as follows: CPU processor is

305
“arm926ej-s” and ASIC is “FPGA”. And then click “set architecture” button in “arch” tab (Figure 14-6). If you see the confirmation window message, press OK.

4. Now click the “partition” tab. You have two partitioning options: automatic and manual partitioning. First of all, you should set the partition target parameter. And then just press “Run” button for automatic partitioning. Otherwise, if you press the “manual partition” button, you can see the selectable check-list table representing the candidate processing elements and all nodes be partitioned as shown in Figure 14-7. You can select on HW or SW for each function block. For an infeasible mapping, the check-box is disabled. After you check the list finish your manual partition, click “Set”.

Figure 14-6 Architecture description

Figure 14-7 In case of mapping “FixCBPIDCTBlock122” to hardware only
5. The partitioning result will be showed up in a Gantt chart as shown in Figure 14-8 for the manual partition of Figure 14-7.

![Figure 14-8 The Gantt chart in case of mapping “FixCBPIDCTBlockI22” to hardware only](image)

For automatic partitioning option, the Gantt chart of Figure 14-9 will be shown as the partition result for DIVX player with “Overlapped” and “Schedule-Based” parameters.

![Figure 14-9 The Gantt chart of automatic partitioning](image)

At the same time PeaCE generates the output interface file: `DesignName_sched.xml"`. It represents how function blocks are partitioned and scheduled for each processing element. The schedule file name is
DesignName_sched.xml is located at $HOME/PEACE_SYSTEMS/DesignName/ directory. If you want to know the description in detail, see the ‘Interface files’ chapter.

![Image of DesignName_sched.xml file]

**Figure 14-10 The generated DesignName_sched.xml file**

### 14.3 How to partition for randomly generated graph

Generally, PeaCE needs three input XML files as described in section 14.1. We also support a functionality to partition the randomly generated graph for research purpose. It means that you can use our partitioning algorithm without drawing an SPDF graph in PeaCE. Image how terrible it will be if you should draw the graph in Hae to evaluate the partitioning algorithm!

For this reason, we provide the sweet tool, XML converter, to convert a randomly generated graph or user-defined graph to generate the three input files. If you want to know what is the pre-defined format and how to convert the pre-defined file to XML files, see “Utility” section.

After convert the text graph file to the XML files, you **should do** the followings:

- Change each XML file name to the name with DesignName_ as its prefix.
- Copy the XML files to $HOME/PEACE_SYSTEMS/DesignName/ directory.
- Create a project with the DesignName. You do not need anything except architecture selection in this project.
- Must make an initial architecture with the same number of processing element as the number of processing elements used in randomly generated DesignName_TimeCost.xml.
- Click “set architecture”
- Do partition
14.4 How to integrate your partitioning algorithm

PeaCE supports a way to integrate an external partitioning algorithm into our design flow. Since each design step of PeaCE is modularized by XML description, your partitioning algorithm can be easily integrated in our design flow if it can deal with three input files described in Chapter 11.

Figure 14-11 shows the target in the partition tab for importing an external partitioning algorithm. Note that you may not be able to see the Gantt chart after your partitioning unless you generate the sched.xml as the partitioning result.

To integrate your partitioning algorithm, first your code files should be copied in the path below:
- SPEACE/src/domains/bp/targets/BPCosynthesisExternalTarget/your_directory

Then modify the ‘BPCosynthesisExternalTarget.cc’ in the following directory:
- SPEACE/src/domains/bp/targets/

BPCosynthesisExternalTarget.cc file should be modified as follows:

```
add your algorithm to a candidate partitioning method in the target parameter called “Partition”. Initially, no algorithm is registered so that the Partition parameter is set “NONE”. If you add your algorithm as the following line, “YourAlgo” will be shown as a candidate partition method in the parameter list box.

addState(pPartitionMethod.setState("Partition", this, "None ##None,YourAlgo.", "your description"));

- append your execution file name.
```

```
int BPCosynthesisExternalTarget :: choosePartitioner () {
   ....
   } else if (((const char*)pPartitionMethod)[0]=='Y') { // External algorithm
      algorithmdir << "your_directory";
      command << "your_execution_file_name";
   }
```

Note that ‘Y’ indicates the initial letter of your partition algorithm name that you registered in the previous step. After modifying the .cc file, you should compile and restart the ptcl window.

- cd SPEACE/src/domains/bp/targets
- make depend
- cd $PEACE/obj.SPTARCH/domains/bp/targets
- make install
- cd $PEACE/obj.SPTARCH/ptcl
- make install (for ptcl) or make ptcl.debug (for gdb)

Now restart your ptcl and click ‘run’. Then, you will see that your algorithm imported in PeaCE.