The PartiCore Quick Guide

Version 1.0

Yifan Wu

y.wu@sssup.it

2011-10-06

Contents

Chapter 1

Introduction

This document describes the characteristics of PartiCore, a software package for partitioning parallel real-time applications on a multi-core platform. The tool has been developed at the Retis Lab (`http://retis.sssup.it/`) of the Scuola Superiore Sant'Anna and is available at

`http://particore.sssup.it/`

1.1 What is PartiCore

PartiCore is a tool for partitioning parallel real-time applications on a set of reservations available on a multi-core platform. It consists of a C++ library and a GUI-based tool. The tool allows a user to specify the structure of a real-time application in terms of tasks subject to precedence constraints. This is done by a very intuitive graphical interface that allows the user to create tasks, position them with drag and place operations, define precedence constraints between task pairs, and delete tasks or precedence links. Once the application graph has been defined, the user can specify a few timing parameters, as the computation time for each task and the period and the relative deadline of the application. Context switch cost can also be specified and it is used to take overhead into account in the optimization procedure. The tool shows the timeline representation, which represents the individual executions starting at the minimum possible time, assuming as many cores as needed. Once the application and its parameters have been entered, the user can run the algorithm for partitioning the graph into a set of flows (a flow is a subset of tasks allocated on a single reservation and sequentially executed on a single core).

Two optimization criteria can be selected to minimize the total bandwidth demanded by the application or the number of required cores (fragmentation). Partitioning can be performed using an exhaustive branch and bound search (precise, but slow) or by a heuristic algorithm (faster but not optimal). Note that for large applications consisting of more the 15 tasks, the branch and bound

search is too expensive, hence the heuristic search should be selected. The tool also allows creating/removing manual partitions.

When a partition is found, the tool identifies the various flows with different colors, and for each flow it shows (in a window below the application graph) the corresponding demand bound function and supply function, with the associated (α, Δ) parameters computed by the algorithm.

1.2 System model

A real-time application Γ is modeled as a set of n tasks with given precedence constraints, specified as a Directed Acyclic Graph (DAG). The application is sporadic, meaning that it is cyclically activated with a minimum inter-arrival time T (also referred to as period) and must complete within a given relative deadline D, which can be less than or equal to T.

Each task τ_i is a portion of code that cannot be parallelized and must be executed sequentially. A task τ_i can be preempted at any time and is characterized by a known worst-case execution time (WCET) $C_i > 0$. A task τ_i is also assigned an absolute deadline d_i and an activation time a_i relative to the activation of the first task of the application, which is also the activation time of the application, denoted by a. This means that τ_i must execute in $[a + a_i, a + d_i]$. Tasks are scheduled by the Earliest Deadline First (EDF) scheduling algorithm [3].

Using PartiCore, the application Γ is partitioned into m flows. Each flow F_k is a subset of tasks, i.e., $F_k \subseteq \Gamma$, allocated on a virtual processor VP_k, which is an abstraction of a sequential machine achieved through a resource reservation mechanism. The virtual processor VP_k is characterized by a bandwidth $\alpha_k \leq 1$ and a maximum service delay $\Delta_k \geq 0$, and hence is also called alpha-delta server.

There are two methods for creating a partition: Branch-and-Bound Search and Heuristic Search. Two optimization criteria can be selected to create a partition:

- *Total bandwidth* B is the overall bandwidth requirement of all flows, that is
$$B = \sum_{k=1}^{m} B_k = \sum_{k=1}^{m} \left(\alpha_k + 2\sigma \frac{1 - \alpha_k}{\Delta_k} \right). \tag{1.1}$$
- *Fragmentation* β is an index that reveals the degree of fragmentation of all flows, given by
$$\beta = \max_{k=1,\ldots,m} \frac{\sum_{i=k}^{m} B_i}{B_k}, \tag{1.2}$$
where the bandwidths $B_1, \ldots, B_m$ are assumed to be ordered by non-increasing values.

The theoretical foundations of the methodology used by PartiCore can be found in [1].

Chapter 2

Getting Started

To use PartiCore, uncompress the downloaded zip file to any directory. For example, if the path to the unzipped PartiCore package is $PARTICORE$, the inner folder structure is the following:

```
$PARTICORE$/
├── PartiCore/
├── Examples/
├── UnitTest/
├── GUI/
│   ├── Python/
│   └── Windows/
└── QuickGuide/
```

- PartiCore/ contains the source code of the library. To use the library, simply add the containing files to a C++ project and include the particore.h file in the code:

```
#include "particore.h"
```

- Examples/ contains several examples.
- UnitTest/ contains a bundle of unit tests for the PartiCore library, using Google Test framework.
- GUI/ is the graphical user interface that helps the user build, modify, analyze an application and its partitions. The GUI tool is a browser/server-based web application. Most of its functions could be achieved using only the HTML/Javascript files on the client side, which are located in $PARTICORE$/GUI/Python/particore/ and $PARTICORE$/GUI/Windows/particore/, as two identical copies. Simply open in the browser the index.html file contained in either of the above folder to use the client-side functions alone.

  To use more complex functions like Branch-and-Bound Search or Heuristic Partition, a CGI server must be started. There are two ways to do that:

  - Python/ contains a simple Python CGI server. Any operating system having Python interpreter[1] installed is qualified. To use it, type the following command line in the terminal:

    python webserver.py

  - Windows/ is for a window user that does not have Python installed. It contains a windows executable which has been compiled using pyinstaller. To use it, double click webserver.exe file.

  In either way, the index.html file will be opened in the browser automatically. Otherwise, manually open the following link in the brower:

  `http://localhost:9000/particore/index.html`

- QuickGuide/ contains this Quick Guide.

[1] It has only been tested on Python 2.6.

Chapter 3

Using PartiCore

This chapter explains how to use the PartiCore library and program with it in C++ language.
To use the library, simply add the files in $PARTICORE$/PartiCore/ to a C++ project and include the particore.h file in the user's code:

```
#include "particore.h"
```

Since the whole library is wrapped in a namespace called *particore*, the following line could be added in the user's preference:

```
using ::particore::Application;
```

3.1 Building an application

3.1.1 Creation

The following code creates a simple application:

```
Application app;
app.Create("[[[10],[15]],[[0,1]],[[0,1]],[40,0,40,0.1]]");
```

This code first makes an instance of the Application class, and then builds the application with two tasks whose WCETs are 10 and 15. The first task is the predecessor of the second task. The application is partitioned into one flow containing both tasks.

3.1.2 Application definition format

The string passed into Application::Create(**const** string& str) function is the definition of application to be created in the format as Listing 3.1.
The whole definition is wrapped in a pair of square brackets, as line 1 and line 7. Inside, there are 5 groups of definition, each within a pair of square brackets, as

Listing 3.1: Definition format of an application

```
1 [
2     [[WCET,X,Y], ...],
3     [[predecessor_index,successor_index], ...],
4     [[task_index,...], ...],
5     [D, a, T, sigma],
6     [mode1, mode2, mode3, mode4]
7 ]
```

shown from line 2 to line 6. Notice that, groups of definition must be separated by commas, but not necessarily by line breaks.
Line 2 is the definition of tasks. Tasks are wrapped in square brackets and separated by commas. For each task, WCET is the worst-case execution time, and the optional X, Y are the coordinates in the GUI tool. The ID of a task will be the prefix "Task" plus its index, which is 0-based and decided by its position in the definition list. That is, tasks will be given IDs as "Task0", "Task1", "Task2", and so forth.
Line 2 defines the immediate precedence relations between tasks. Each precedence is specified as [predecessor_index, successor_index], where predecessor_index and successor_index is the index of the predecessor and successor, respectively. For instance, [0, 1] means "Task0" is the predecessor of "Task1".
Line 3 defines the partition of the application. For each flow, wrap the indices of its containing tasks in a pair of square brackets. Leave blank the inner content of the [ ] to specify no predefined partition.
Line 4 specifies the application parameters, that is the relative deadline, activation time, period, server context switch cost.
Line 5 specifies the modes that will be used for Branch-and-Bound Search or Heuristic Partition:

- mode1 specifies the choice of deadline assignment method
- mode2 specifies the choice of optimization goal
- mode3 specifies the choice of the heuristic method
- mode4 specifies the choice of the search mode

They will be further explained later. Notice that, Line 5 is optional. When it is omitted, the application will use default values for all the modes.

3.1.3 Loading from a file

To avoid the verbose string of the application definition in source code, PartiCore gives an option to move the definition into a plain text file and load it using code as:

```
app.Load(filename);
```

3.1.4 Initialization

After creation, the application has to be initialized before further manipulation:

```
app.Initialize();
```

The initialization mainly performs the following jobs:

- Find the topological order
- Find the critical path
- Calculate the sequential computation time
- Calculate the parallel computation time

Notice that, the application must be initialized again after it is modified.

3.2 Partitioning an application

3.2.1 Partitioning methods

There are several ways to partition an application. The easiest one is to specify the partition while creating the application, as shown in Section 3.1.1. Another way is to manually partition after the application is created. See $PARTICORE$/Examples/manual_partition.cpp for an example.
To find a partition that minimize a certain criterion, two methods can be used: Branch-and-Bound Search or Heuristic Partition. Both will be explained in Section 3.3.

3.2.2 Calculating the reservation parameters

For a partitioned application, the reservation parameters (α, Δ) of each virtual processor can be calculated. But before that, tasks must be assigned with proper deadlines and activation times.
In PartiCore, two methods are supported for deadline setting:

- *CHETTO*: the method originally proposed by Chetto-Silly-Bouchentouf [2], and adapted to work on multi-core systems
- *CHETTO**: a modified version of *CHETTO* that is proposed in [1]

During the creation of application, the choice of deadline setting method could be specified in the application definition by setting mode1 in Listing 3.1 to 1 (for CHETTO_STAR) or 2 (for CHETTO). After creation, it can still be

changed by evaluating Application::mode_deadline_setting with CHETTO or CHETTO_STAR.
The following code snippet uses *CHETTO**, and sets the deadlines of all tasks, and activation times afterwards. Notice that, activation times must be assigned after setting deadlines.

```
app.mode_deadline_setting = particore::CHETTO_STAR;
app.SetDeadlines();
app.SetActivations();
```

Upon setting deadlines and activations of all tasks, the reservation parameters of all the virtual processors could be calculated using the following code:

```
app.ComputeReservationParameters();
```

While calculating the reservation parameters, the criteria of the partition (the cost) are evaluated as well, saved in Application::total_bandwidth for *Total bandwidth* and Application::beta for *Fragmentation*, respectively.

3.3 Finding an partition

3.3.1 Setting optimization goal

As described in Section 1.2, two optimization goals are supported in ParticCore: *Total bandwidth* B and *Fragmentation* β, which can be set by evaluating Application::mode_optimization_goal with TOTAL_BANDWIDTH or BETA, respectively. The following code sets the optimization goal as *Fragmentation*:

```
app.mode_optimization_goal = particore::BETA;
```

The choice of optimization goal can also be specified in the application definition, by setting mode2 in Listing 3.1 to 1 (for BETA) or 2 (for TOTAL_BANDWIDTH).

3.3.2 Branch-and-Bound Search

The partition that minimizes the intended optimization goal can be found using Branch-and-Bound Search:

```
app.ComputeOptimalPartitioning();
```

To reduce the search time, the maximum allowed number of flows m_{max} can be set as a pruning condition, using the variable Application::max_flow_num. It is by default equal to 0, meaning that this pruning condition is not used.
The search result is saved in Application::search_result, which stores all the partitioning ways that give the same optimal cost. It could be output using:

```
app.search_result.Print(cout);
```

The Branch-and-Bound Search can be performed while a subset of tasks are already fixed to a certain number of flows, which are called fixed flows. That means, this subset of tasks will not participate in the search procedure. However, other tasks are allowed being put into the fixed flows. To enable this search method, set Application::mode_search to EXHAUSTIVE_FIXED_FLOW. See $PARTICORE$/Examples/search_with_fixed_flows.cpp for an example.

3.3.3 Heuristic Partition

The Branch-and-Bound Search method is extremely time consuming when the number of tasks is large. In that case, heuristic methods are preferred. In this version, 4 heuristic methods are supported:

- MULTI_CRITICAL_PATH_FIT_ALL_CP first puts the critical path in a flow. In the second step, it tries to fit the critical path in the remaining graph into one of the existing flows. If not possible, a new flow is created. This continues until M_{low} flows are created. In the third step, the remaining tasks are selected by decreasing computation times and put in the existing flows using Best Fit policy. Refer to the heuristic algorithm H1 in [1] for details.
- MULTI_CRITICAL_PATH differs from the above heuristic method in that at the second step, the critical path of the remaining graph is directly put into a new flow.
- SINGLE_CRITICAL_PATH is similar with H1 with the only difference that it directly goes from the first step to the third step, ignoring the second step completely. Refer to the heuristic algorithm H2 in [1] for details.
- NAIF builds flows by putting the remaining tasks one by one using Next Fit policy. Refer to the Naif algorithm in [1] for details.

The choice of heuristics can be set by evaluating Application::mode_heuristic with the above listed names in uppercase. The following code is an example that sets the heuristic method to SINGLE_CRITICAL_PATH and then finds the partition using this method.

```
app.mode_heuristic = particore::SINGLE_CRITICAL_PATH;
app.HeuristicPartition();
```

Chapter 4

The GUI-based Tool

4.1 The main interface

The GUI-based tool of PartiCore is a browser/server-based web application that helps the user build, modify, analyze an application and its partition. Fig. 4.1 shows the main interface.

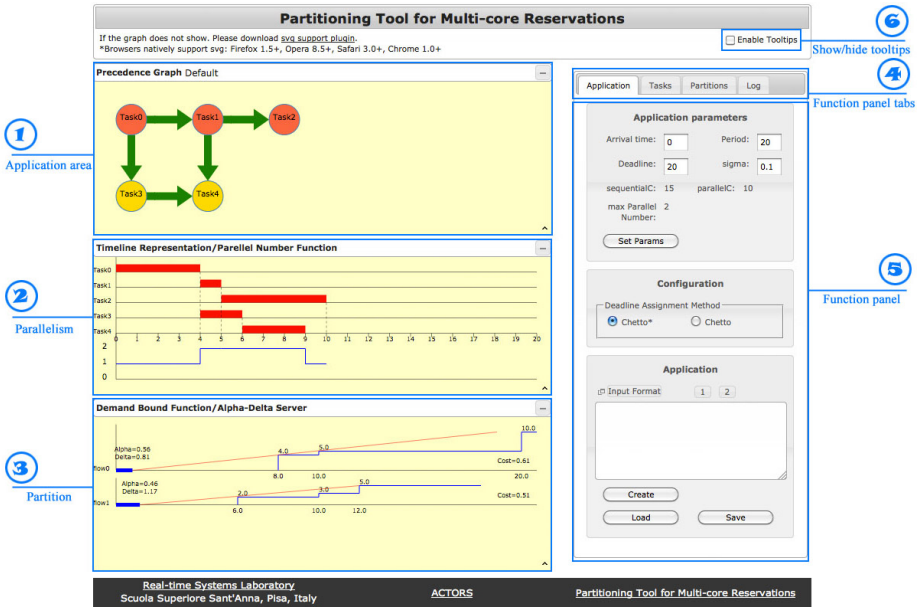


Figure 4.1: Graphical User Interface of PartiCore.

(1) **Application area** is the window to show and select tasks and precedence relations. The name of the application can be changed by editing the top bar of the window.

(2) **Parallelism** shows the timeline representation of the application and its corresponding parallel number function.

(3) **Partition** shows the demand bound function of each flow and the (α, Δ) server of its hosting virtual processor.

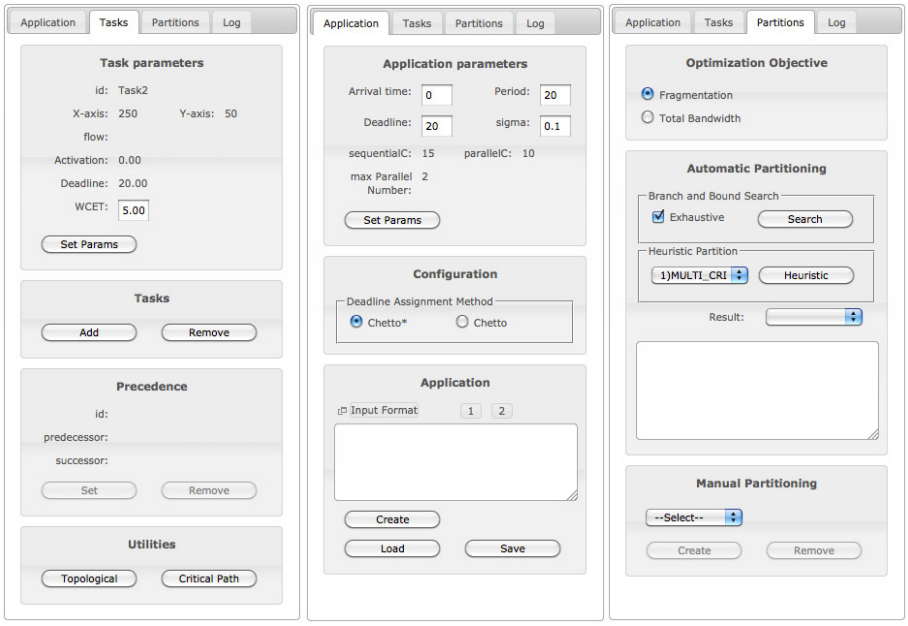
(4) **Function panel tabs** allows switching between different function panels, that are Application, Tasks, Partitions, and Log.

(5) **Function panel** allows manipulation of the application, e.g., adding/removing tasks, setting parameters, finding optimal partition, etc.

(6) **Show/hide tooltips** toggles the tooltips appearance.

4.2 Basic usage

The operation of the tool can be performed mainly through the function panels.



(a) Tasks panel. (b) Application panel. (c) Partitions panel.

Figure 4.2: Function panels.

4.2.1 Tasks panel

The tool allows the user to specify the structure of a real-time application in terms of tasks, subject to precedence constraints. This is done by allowing

the user create tasks, position them with drag and place operations, define precedence constraints between task pairs, and delete tasks or precedence links. Tasks and precedence links can be created by proper buttons available in the Tasks panel shown in Fig. 4.2a. Once the application graph has been defined, the user can specify the computation time for each task, by selecting the task with the mouse in the application area, filling the corresponding field in the Tasks panel, and clicking the Set Params button.

4.2.2 Application panel

Then, the user can specify the timing parameters of the entire application (activation time, period, and relative deadline), by filling the proper fields in the Application panel (see Fig. 4.2b). An additional field allows specifying the context switch cost (σ) to be used when computing the overall bandwidth requirements. The tool also constructs the timeline representation, which clearly visualizes the individual executions starting at the minimum possible time, assuming as many cores as needed.
The application can be saved by clicking the Save button, where the application will be translated into text in the format as Listing 3.1, and put in the textbox. If the server script of the tool has been started, a popup window will show up for the user to download a file containing this text. Otherwise, manually copy the text into a file to save the application.
To load an application, click the Load button and upload the file containing the application definition. This requires server support as well. An alternative way is to copy the text of the application definition into the textbox, and click the Create button. It is also possible to directly create one of the two pre-defined applications, by clicking button 1 or 2.

4.2.3 Partitions panel

Once the application and its parameters have been entered, the user can run the algorithm for partitioning the application into a set of flows, each of which is individually allocated to a virtual processor, characterized by (α, Δ). This is done from the Partitions panel illustrated in Fig. 4.2c.
Before running the partitioning algorithm, the optimization goal has to be selected, between minimizing the total bandwidth and minimizing the fragmentation. Then, the partition can be performed using Branch-and-Bound Search (the Search button) or by Heuristic Partition (the Heuristic button). Different heuristic methods can be selected from the drop-down list.
Note that for large application consisting of more than 15 tasks, Branch-and-Bound Search is too expensive, hence deselect the 'Exhaustive' checkbox to search with fixed flows. The tool allows creating/removing manual partitions, just by selecting a subset of nodes with mouse in the application area, and clicking the Create/Remove button in the Partitions panel.
The result of running the partitioning algorithms is shown in the textbox of the Partitions panel, and listed in the drop-down list beside 'Result' label. For

Brand-and-Bound search, there might be multiple partitioning ways that give the same optimal cost. Select from the drop-down list to view a specific partition in the application area.
When a partition is selected, the tool identifies the various flows with different colors, and for each flow it shows the corresponding demand bound function and supply function, with the associated (α, Δ) parameters computed by the algorithm.

References

[1] Giorgio Buttazzo, Enrico Bini, and Yifan Wu. Partitioning parallel applications on multiprocessor reservations. *IEEE Transactions on Industrial Informatics*, 7(2):302–315, May 2011.

[2] H. Chetto, M. Silly, and T. Bouchentouf. Dynamic scheduling of real-time tasks under precedence constraints. *Real-Time Systems*, 2(3):181–194, September 1990.

[3] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the Association for Computing Machinery*, 20(1), 1973.