Manual

Tommaso Costanzo

October 2019

Contents

1 Introduction

What follows is a list of the possible commands for rplsh. For each command it's indicated in **bold** the mandatory parts, in *italic* the particular value for the user and between square brackets [] the optional parts.

In the following, *identifier* will indicate an user defined name, created simply by doing *new_identifier* = *pattern*. If the identifier has several possible pattern implementations, shown by the **show** command, it's possible to refer to a single one of them by doing *identifier*[*integer*], indicating between brackets the desired index of the possible patterns.

2 Verbs

- **show** *identifier* [**by** *parameters*]: shows a list of possible implementations of the identifier. If one or more parameters are given, the list is ordered according to them and also the values of the parameters are shown. If there is more than one parameter, they must be separated by a comma.
- **set** *parameter* **with** *number*: sets the desired global parameter with the numerical value given.
- **annotate** *identifier* **with** *parameter value* [**as** *static/dynamic*]: updates the parameter value of the identifier with the newly given. For the **grain** parameter, you can also specify if the scheduling should be static or dynamic; if a value is given, the default is dynamic.
- **rewrite** *identifier* **with** *rewriting rules*: applies the given rules to the identifier. If there is more than one rule, they must be separated by commas. It's possible to apply all the rules indicating **allrules** as rewriting rule.
- **optimize** *identifier* **with** *optimization rules*: applies the given optimization to the identifier. If there is more than one rule, they must be separated by commas.
- **history** [*identifier*]: prints on screen all the valid commands given in the current session. If an identifier is given, only the commands relative to the identifier are printed.
- **import "***file***"**: imports a source file. The path for the file must be between "".
- **gencode** *identifier* [**as** *name* **in** *dir*]: generates a fastflow code that implements the pattern of the identifier; if a name is given, that will be the name of the source file; if a directory is given, the file will be saved in that directory.
- **expand** *identifier1* [**in** *identifier2*]: when used on an identifier that has other identifiers inside, the internal part is expanded to show what those identifiers are. For example, with a = seq(), b = pipe(a), c = farm(b), using expand c instead of farm(b) with the show command we obtain farm(pipe(a)). If used with a second identifier, only the second identifier will contain the expanded form.
- **add** *identifier1* **in** *identifier2*: used to add source and/or drain patterns to another identifier. The second identifier will become a pipe with the source/drain and the original pattern.
- **load "***file***"** [*boolean*]: imports a text file with a list of commands for rplsh and then executes them. If the boolean value is given and it is true, the commands are printed while executed.

3 Patterns

- **seq** ([*number* [**,** *boolean*]]): builds a sequential pattern for an identifier. If a number is given, the service time of the pattern is set to that value, otherwise is set to 1. If a bolean value is given, it sets the datap non functional parameter.

- **source** ([*number*]): builds a source pattern for an identifier. If a number is given, the service time of the pattern is set to that value, otherwise is set to 1.

- **drain** ([*number*]): builds a drain pattern for an identifier. If a number is given, the service time of the pattern is set to that value, otherwise is set to 1.

- **comp** (*pattern* [**,** *pattern*]): builds a sequential composition of two or more other patterns for an identifier.

- **pipe** (*pattern* [**,** *pattern*]): builds a pipe of two or more other patterns for an identifier.

- **farm** (*pattern* [**,** *integer*]): builds a farm of the given pattern for an identifier. If a number is given, that becomes the number of workers; by default, the number of workers is 1.

- **map** (*pattern* [**,** *integer*]): builds a map of the given pattern for an identifier. If a number is given, that becomes the number of workers; by default, the number of workers is 1.

- **reduce** (*pattern* [**,** *integer*]): builds a reduce of the given pattern for an identifier. If a number is given, that becomes the number of workers; by default, the number of workers is 1.

- **divide_conquer** (*pattern* [**,** *integer*]): builds a divide and conquer of the given pattern for an identifier. If a number is given, that becomes the number of workers; by default, the number of workers is 1.

4 Rewriting rules

These rules can be used only with the **rewrite** verb.

- **farmintro**: puts the given identifier inside a new farm.
- **farmelim**: removes the outermost farm from the identifier. It does not effect internal farms.
- **pipeintro**: p sostituisce un comp con pipe inside a new pipe.
- **pipeelim**: removes the outermost pipe from the identifier. It does not effect internal pipes.
- **pipeassoc**: if there are two pipes, one inside the other, moves the internal pipe from left to right or vice versa.
- **compassoc**: if there are two comps, one inside the other, moves the internal comp from left to right or vice versa.
- **mapofcomp**: transforms a sequential composition of two maps into a map of sequential compositions.
- **compofmap**: transforms a map of sequential compositions into a sequential composition of maps.
- **mapofpipe**: transforms a pipe of two maps into a map of a pipe.
- **pipeofmap**: transforms a map of a pipe into a pipe of two maps.
- **mapelim**: removes the outermost map from the identifier. It does not effect internal maps.
- **reduceelim**: removes the outermost reduce from the identifier. It does not effect internal reduces.
- **mapmapelim**: combines a map of map into a single map.
- **farmfarmelim**: combines a farm of farm into a single farm.
- **compdel**: removes the sequential composition if inside the comp there is just one element.
- **pipedel**: removes the pipe if there is just one element inside.
- **dctomap**: replace a divide and conquer node with a map node. Assumptions: the divide function splits the container in half, the merge function reassembles the two half, the condition is true if the size of the container is less of or equal to 1.
- **maptodc**: replace a map node with a divide and conquer node. Assumptions: the divide function splits the container in half, the merge function reassembles the two half, the condition is true if the size of the container is less of or equal to 1.

5 Optimization rules

This rules works even if the patterns that have to be optimized are inside another pattern. For example, a map(farm(farm(x))) can be optimized with the farmfarmopt rule, but the rewrite rule farmelim would not work.

- **farmopt**: chooses the optimal number of workers for the farm considering the service time of the workers and the one of the emitter.
- **pipeopt**: changes the internal structure of the pipe trying to optimize completion time, resources, ecc.
- **mapopt**: chooses the optimal number of workers for the map considering the service time of scatter, gather and workers.
- **reduceopt**: chooses the optimal number of workers for the reduce considering the size of the input.
- **dcopt**: chooses the optimal number of workers for the divide and conquer.
- **maxresources**: if possible, reduces the amount of resources used until it's equal to the maximum set by the user with the environmental parameter.
- **twotier**: works only on map and reduce patterns: it substitutes the non sequential pattern inside the map/reduce with a sequential composition to adhere to the two tier model.
- **farmfarmopt**: if there are two consecutive farms they are merged into a single one.
- **mapmapopt**: if there are two consecutive maps they are merged into a single one.

6 Non functional parameters

This parameters can only be used for the **show** and **annotate** verbs.

- **servicetime**: used to show the service time of different pattern structures for the same problem or to annotate a pattern.
- **latency**: used to show the latency of different pattern structures for the same problem or to annotate a pattern.
- **pardegree**: used to show the degree of parallelism of different pattern structures for the same problem or to annotate a pattern.
- **compltime**: used to show the completion time of different pattern structures for the same problem or to annotate a pattern.
- **resources**: used to show the number of resources needed for different pattern structures for the same problem or to annotate the number of resources available.
- **datap**: used to annotate that a sequential wrapper can be used as a functional parameter of data parallel skeletons, like map or reduce.
- **grain**: used to annotate map or reduce; it specifies the size of the grain in the parallel for loop and the type of scheduling (static or dynamic).
- **dc_capable**: used to annotate that a sequential wrapper can be used inside a divide and conquer skeleton.

7 Environmental parameters

All this parameters can only be used with the **set** verb.

- **emitter_time**: used to define the service time of the emitter.
- **collector_time**: used to define the service time of the collector.
- **scatter_time**: used to define the service time of the scatter.
- **gather_time**: used to define the service time of the gather.
- **dimension**: used to define the dimension of input. Affects only completion time.
- **inputsize**: used to define the input size of the program. Affects service time of wrappers annotated with "datap".
- **resources**: used to define the maximum number of resources available.