Preesm Documentation

Workflow Tasks Documentation

Last update for Preesm version 2.13.0

Desnos Karol, Nezan Jean-François, Pelcat Maxime
(contact: contact@preesm.org)

August 22, 2018

Contents

Acronyms

DAG Directed Acyclic Graph. 9, 13

Fifo First-In, First-Out queue. 6, 25

IBSDF Interface Based Synchronous Dataflow (SDF). 4, 5, 6, 8, 9

MEG Memory Exclusion Graph. 13, 14, 15, 16, 19, 20, 21

PE Processing Element. 19, 22

PiSDF Parameterized and Interfaced SDF. 4, 8, 9

SDF Synchronous Dataflow. 2, 4, 5, 6, 7, 8, 9, 23, 25

SDF3 SDF For Free. 23

How to Read this Document

<table>
<tr><th>Graphic Element</th><th>Brief Description</th></tr>
<tr><td rowspan="3">TaskName
Input1 Output1
Input2 ...
...</td><td>Description of the purpose of the workflow task in one sentence.</td></tr>
<tr><td>Plugin identifier</td></tr>
<tr><td><code>ID associated to the workflow task. In order to use the presented workflow task, add a new task to a workflow using Preesm, edit the property of the new workflow task, and set the "plugin identifier" field of the "Basic" property tab with the value given in this cell.</code></td></tr>
</table>

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">Param1</td><td colspan="2">Description of what this parameter does.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td>value1</td><td>Description of the effect of this parameter value.</td></tr>
<tr><td>value2</td><td>Description of the effect of this parameter value.</td></tr>
</table>

Description
Detailed description of the workflow task including references to associated papers (if any).

Documented Errors
None

Graph transformation

Static PiMM to IBSDF

Graphic Element	Brief Description
StaticPiMM2SDF PiMM SDF scenario	Transforms a static PiSDF Graph into an equivalent IBSDF graph. **Plugin identifier** `org.ietr.preesm.experiment.pimm2sdf.StaticPiMM2SDFTask`

Parameters	Description
None	

Description

In Preesm, since version 2.0.0, the Parameterized and Interfaced SDF (PiSDF) model of computation is used as the frontend model in the graphical editor of dataflow graphs. This model makes it possible to design dynamically reconfigurable dataflow graphs where the value of parameters, and production/consumption rates depending on them, might change during the execution of the application.
In former versions, the Interface Based SDF (IBSDF) model of computation was used as the front end model for application design. Contrary to the PiSDF, the IBSDF is a static model of computation where production and consumption rates of actors is fixed at compile-time.
The purpose of this workflow task is to transform a static PiSDF graph into an equivalent IBSDF graph. A static PiSDF graph is a PiSDF graph where dynamic reconfiguration features of the PiSDF model of computation are not used.

See also: IBSDF [15], PiSDF [5]

Documented Errors

None

Hierarchy Flattening

<table>
<tr><th>Graphic Element</th><th>Brief Description</th></tr>
<tr><td rowspan="3">HierarchyFlattening
SDF SDF</td><td>Transforms a hierarchical IBSDF graph into an equivalent SDF graph.</td></tr>
<tr><td>Plugin identifier</td></tr>
<tr><td><code>org.ietr.preesm.plugin.transforms.flathierarchy</code></td></tr>
</table>

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="5">depth</td><td colspan="2">This parameter is used to select the number of hierarchy levels that will be flattened by the workflow task.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td>0</td><td>The input IBSDF graph is copied to the output port of the workflow task with no modification.</td></tr>
<tr><td>$n \in \mathbb{N}^+$</td><td>The first n levels of the hierarchical IBSDF graph are flattened.</td></tr>
<tr><td>$n < 0$</td><td>All levels are flattened (up to $2^{31} - 1$)</td></tr>
</table>

Description
The purpose of this workflow task is to flatten several levels of the hierarchy of an IBSDF graph and produce an equivalent SDF graph. A hierarchical IBSDF graph is a graph where the internal behavior of some actors is described using another IBSDF subgraph instead of a C header file. When applying this transformation, hierarchical IBSDF actors of the first n levels of hierarchy are replaced with the actors of the IBSDF subgraph with which these hierarchical actors are associated. **See also:** IBSDF [15]

Documented Errors
`Inconsistent Hierarchy, graph can't be flattened` Flattening of the IBSDF graph was aborted because one of the graph composing the application, at the top level or deeper in the hierarchy, was not consistent. **See also:** Graph consistency [11].

Single-Rate Transformation

Graphic Element	Brief Description
Single-rate Transformation SDF SDF	Transforms an SDF graph into an equivalent single-rate SDF graph. **Plugin identifier** `org.ietr.preesm.plugin.transforms.sdf2hsdf`

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">ExplodeImplore-Suppr</td><td colspan="2">(Deprecated: use at your own risks)
This parameter makes it possible to remove most of the explode and implode actors that are inserted in the graph during the single-rate transformation. The resulting SDF graph is an ill-constructed graph where a single data input/output port of an actor may be connected to several First-In, First-Out queues (Fifos).</td></tr>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>false</code></td><td>(default) The suppression of explode/implode special actors is not activated.</td></tr>
<tr><td><code>true</code></td><td>The suppression of explode/implode special actors is activated.</td></tr>
</table>

Description
The purpose of this task is to transform an SDF graph — which is actually an IBSDF graph in Preesm— into an equivalent single-rate SDF graph. A single-rate SDF graph is a graph where each actor of the original SDF graph is duplicated as many times as its number of firings per iteration of the original graph. The purpose of this transformation is to reveal of the implicit data-parallelism of the original SDF graph. Special actors, called *explode* and *implode* actors, may be automatically inserted in the single-rate SDF graph resulting from this transformation. The purpose of these actors is to distribute (resp. gather) data-tokens produced (resp. consumed) on a single input (resp. output) port of an actor in order to send them to several consumer actors (resp. to receive them from several producer actors). **See also:** Single-rate transformation [16], Special actors [7]

Documented Errors
`Graph not valid, not schedulable` Single-rate transformation of the SDF graph was aborted because the top level was not consistent, or it was consistent but did not contained enough delays — i.e. initial data tokens — to make it schedulable. **See also:** Graph consistency [11].

Data-parallel Transformation

Graphic Element	Brief Description
Data-parallel Transform SDF SDF INFO	Detect wheter an SDF graph is data-parallel and provide its data-parallel equivalent Single-Rate SDF and its re-timing information.
	Plugin identifier
	`fi.abo.preesm.dataparallel.DataParallel`

Parameters	Description
None	

Description

An SDF graph is data-parallel when for each actor of the SDF graph, all of its instances can be executed at the same time. For instance, all strictly acyclic SDF graphs are data-parallel. This task increases the scope of this analysis to generic SDF graphs.
The task analyses an input SDF graph and reports if it is data-parallel by analysing each strongly connected component of the graph. If a strongly connected component requires a re-timing transformation for it to be data-parallel, then the transformation is carried out such that the corresponding strongly connected component in the output single-rate SDF (see Section. 2.3) is data-parallel. The re-timing transformation modifies the delays in the original SDF graph such that the final single-rate SDF output is data-parallel.
However, if a strongly connected component of the SDF is not data-parallel, then the plugin reports the actors of this strongly connected component. In this case, the strongly connected component at the output single-rate SDF graph is same as that of the single-rate transformation on the original SDF.
The data-structure INFO describes mapping of delays in original FIFO to delays in the transformed SDF. This non-trivial initialization of delays in FIFOs is represented using SDF-like graphs where FIFO initialization function is represented as an actor. The data-structure INFO is provided for sake of completion and is not being used by other plugins. The data-structure INFO can change in future based on the design of the initialization of the FIFOs.
See also: Implementation details [9]

Documented Errors

`DAGComputationBug`
: There is a bug in implementation due to incorrect assumption. Report the bug by opening an issue and attaching the graph that caused it.

Graph Exporters

SDF Exporter

Graphic Element	Brief Description
SDF Exporter SDF	Create a new `*.graphml` file containing the exported SDF graph.
	Plugin identifier
	`org.ietr.preesm.plugin.exportXml.sdf4jgml`

Parameters	Description	
path	Path of the directory within which the exported `*.graphml` file will be created. If the specified directory does not exist, it will be created.	
	Value	**Effect**
	`path/in/proj`	Path within the Preesm project containing the workflow where the "SDF Exporter" task is instantiated. Even if the workflow of a Preesm project A is executed with a scenario from a different project B, the `*.graphml` file will be generated within the specified directory of project A. Exported SDF graphs will be named automatically, usually using the same name as the original SDF graph processed by the workflow. If a graph with this name already exists in the given path, it will be overwritten. Example: `Algo/generated/singlerate`
	`path/in/proj/` `name.graphml`	Path within the Preesm project containing the workflow where the "SDF Exporter" task is instantiated. Even if the workflow of a Preesm project A is executed with a scenario from a different project B, the `*.graphml` file will be generated within the specified directory of project A. Exported SDF graph will be named using the string with the `graphml` extension at the end of the given path. If a graph with this name already exists in the given path, it will be overwritten. Example: `Algo/generated/singlerate/myexport.graphml`

Description

The purpose of this task is to create a new `*.graphml` file containing where the exported IBSDF graph will be written. The exported graph can then be visualized and exported using the former Preesm graph editor for IBSDF graph, which was replaced with the PiSDF graph editor since version 2.0.0 (See 2.1).
This task is generally used to export intermediary graphs generated at different step of a workflow execution. For example, to visualize the SDF graph resulting from the flattening of an IBSDF graph (See 2.2), or to understand the parallelism that was exposed by the single-rate transformation (See 2.3).

Documented Errors

`Path <given path> is not a valid path for export. <reason>`
The value set for parameter *path* is not a valid path in the project.

DAG Exporter

Graphic Element	Brief Description
DAG Exporter DAG	Create a new *.graphml file containing the exported Directed Acyclic Graph (DAG).
	Plugin identifier
	org.ietr.preesm.mapper.exporter.DAGExportTransform

Parameters	Description
path	See SDF Exporter task 3.1
openFile	*(Deprecated)*

Description
The purpose of this task is to create a new *.graphml file containing where the exported DAG will be written. The exported graph can then be visualized and exported using the former PREESM graph editor for IBSDF graph, which was replaced with the PiSDF graph editor since version 2.0.0 (See 2.1). This task is generally used to export intermediary graphs generated by the mapping and scheduling tasks of the workflow (See 4).

Documented Errors
Path <given path> is not a valid path for export. <reason> The value set for parameter *path* is not a valid path in the project.

Static Mapping Scheduling

List Scheduler

Graphic Element	Brief Description
List Scheduler architecture — ABC scenario — DAG SDF	Map and schedule the dataflow graph on the architecture. **Plugin identifier** `org.ietr.preesm.plugin.mapper.listscheduling`

Parameters	Description	
Check	Specify whether an automated verification of the mapping validity should be performed. This verification should be activated when trying to develop a new scheduling algorithm, but can be deactivated for Preesm users.	
	Value	**Effect**
	`True`	Activate the verification.
	`false`	Deactivate verification.
balanceLoads	Specify whether the mapping and scheduling algorithm should try to distribute the computational load fairly among cores. Computational load is measured as the amount of time during which a processing element is executing an actor.	
	Value	**Effect**
	`true`	Load balancing will be an objective of the mapping algorithm.
	`false`	Load balancing will not be an objective of the mapping algorithm. Only minimizing the graph iteration latency will be.
edgeSchedType	Specify which strategy the scheduling algorithm should use to order the communications.	
	Value	**Effect**
	`Simple`	*description missing*
	`Switcher`	*description missing*
	`Advanced`	*description missing*

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="8">simulatorType</td><td colspan="2">Control the accuracy/complexity tradeoff of the simulator used to minimize latency of the scheduling algorithm.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td><code>LooselyTimed</code></td><td>The loosely-timed ABC that accounts for task times of operators and transfer costs on PN and CN. However, it does not consider transfer contention (ignoring the difference between Parallel and Contention Nodes in the S-LAM architecture).</td></tr>
<tr><td><code>ApproximatelyTimed</code></td><td>The approximately-timed ABC that associates each inter-core contention node with a constant rate and simulates contentions on CNs.</td></tr>
<tr><td><code>AccuratelyTimed</code></td><td>The accurately-timed ABC that includes the set-up time necessary to initialize a parallel transfer controller such as Texas Instruments Enhanced Direct Memory Access (EDMA). This set-up time is scheduled in the core which triggers the transfer.</td></tr>
<tr><td><code>InfiniteHomogeneous</code></td><td>The infinite homogeneous ABC which is a special ABC that performs an algorithm execution simulation on a homogeneous architecture containing an infinite number of cores with main type. It may be noted that for this study, the main core type of an S-LAM architecture is defined in the input scenario. This ABC enables the extraction of the critical path of the graph.</td></tr>
<tr><td><code>CommConten</code></td><td>description missing</td></tr>
<tr><td><code>DynamicQueuing</code></td><td>description missing</td></tr>
</table>

Description
Schedules the algorithm on the architecture using the Kwok List scheduling heuristic. This implementation of the scheduling algorithm is based on the ABC scheduler. **See also:** List scheduling [10], ABC scheduler [14].

Documented Errors
None

FAST Scheduler

Graphic Element	Brief Description
FAST Scheduler architecture ABC scenario DAG SDF	Map and schedule the dataflow graph on the architecture using an iterative algorithm. **Plugin identifier** `org.ietr.preesm.plugin.mapper.fast`

Parameters	Description
Check	see List Scheduler (Section 4.1).
balanceLoads	see List Scheduler (Section 4.1).
edgeSchedType	see List Scheduler (Section 4.1).
simulatorType	see List Scheduler (Section 4.1).
fastTime	Specify for how long the FAST algorithm will iteratively search for better a scheduling, if not already interrupted manually by the user. **Value** / **Effect** $n \in \mathbb{N}^*$: Timeout value in seconds.
fastLocal-SearchTime	*Description missing.* **Value** / **Effect** $n \in \mathbb{N}^*$: Time in seconds.
displaySolutions	Specify whether Gantt diagrams of intermediate schedules found iteratively by the algorithm should be displayed. **Value** / **Effect** `true` : Intermediate Gantt diagrams are displayed. May be slow for large applications. `false` : Intermediate Gantt diagrams are not displayed.

Description
Schedules the algorithm on the architecture using the Kwok FAST scheduling heuristic. This implementation of the scheduling algorithm is based on the ABC scheduler. **See also:** FAST scheduling [10], ABC scheduler [14].

Documented Errors
None

Memory optimization

MEG Builder

<table>
<tr><th>Graphic Element</th><th>Brief Description</th></tr>
<tr><td rowspan="3">MEG Builder
DAG MemEx
scenario</td><td>Builds the Memory Exclusion Graph (MEG) modeling the memory allocation constraints.</td></tr>
<tr><td>Plugin identifier</td></tr>
<tr><td><code>org.ietr.preesm.memory.exclusiongraph.MemoryExclusionGraphBuilder</code></td></tr>
</table>

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">Verbose</td><td colspan="2">How verbose will this task be during its execution. In verbose mode, the task will log the start and completion time of the build, as well as characteristics (number of memory objects, density of exclusions) of the produced MEG.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td><code>false</code></td><td>(Default) The task will not log information.</td></tr>
<tr><td><code>true</code></td><td>The task will log build and MEG information.</td></tr>
<tr><td rowspan="4">supprForkJoin</td><td colspan="2">(Deprecated) When activated, the DAG used to build the MEG is pre-processed to remove all fork/join (aka. explode/implode) actors. Compatibility of the produced MEG with other workflow tasks is not guaranteed, and known to be non-functional for code generation tasks.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td><code>false</code></td><td>(Default) Feature is not activated.</td></tr>
<tr><td><code>true</code></td><td>Feature is activated.</td></tr>
</table>

Description
The memory allocation technique used in Preesm is based on a Memory Exclusion Graph (MEG). A MEG is a graph whose vertices model the memory objects that must be allocated in memory in order to run the generated code. In the current version of Preesm, each of these memory objects corresponds either to an edge of the Directed Acyclic Graph (DAG) or to a buffer corresponding to ”delays” of the graph that store data between executions of a schedule. In the MEG, two memory objects are linked by an edge (called an exclusion) if they can not be allocated in overlapping memory spaces. **See also:** MEG [2].

Documented Errors
None

MEG Update with Scheduling Information

Graphic Element	Brief Description
MEG Updater DAG MemEx MemEx	Relax memory allocation constraints of the MEG using scheduling information. **Plugin identifier** `org.ietr.preesm.memory.exclusiongraph.MemExUpdater`

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">Verbose</td><td colspan="2">How verbose will this task be during its execution. In verbose mode, the task will log the start and completion time of the update, as well as characteristics (number of memory objects, density of exclusions) of the MEGs both before and after the update.</td></tr>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>false</code></td><td>(Default) The task will not log information.</td></tr>
<tr><td><code>true</code></td><td>The task will log build and MEG information.</td></tr>
<tr><td>supprForkJoin</td><td colspan="2">(Deprecated) See MEG Builder 5.1.</td></tr>
<tr><td rowspan="4">Update with MemObject lifetime</td><td colspan="2">Specify what kind of precedence information should be used when updating the MEG.</td></tr>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>false</code></td><td>(Default) Only data precedence and scheduling order are taken into account to update the MEG.</td></tr>
<tr><td><code>true</code></td><td>Update the MEG with precedence and timing information from the schedule. This option will produce a valid allocation only if the runtime of the actors is constant and identical to the one used by the scheduler. Small variations of the actors runtime may corrupt the memory allocation.</td></tr>
</table>

Description
The MEG used in Preesm can be updated with scheduling information to remove exclusions between memory objects and make better allocations possible. **See also:** MEG update [3].

Documented Errors
None

Memory Bounds Estimator

Graphic Element	Brief Description
Memory Bounds Estimator MemEx	Compute bounds of the amount of memory needed to allocate the MEG
	Plugin identifier
	`org.ietr.preesm.memory.bounds.MemoryBoundsEstimator`

Parameters	Description	
Verbose	How verbose will this task be during its execution. In verbose mode, the task will log the name of the used solver, the start and completion time of the bound estimation algorithm. Computed bounds are always logged, even if the verbose parameter is set to `false`.	
	Value	**Effect**
	`false`	(Default) The task will not log information.
	`true`	The task will log build and MEG information.
Solver	Specify which algorithm is used to compute the lower bound.	
	Value	**Effect**
	`Heuristic`	(Default) Heuristic algorithm described in [2] is used. This technique find an approximate solution.
	`Ostergard`	Östergård's algorithm [12] is used. This technique finds an optimal solution, but has a potentially exponential complexity.
	`Yamaguchi`	Yamaguchi et al.'s algorithm [18] is used. This technique finds an optimal solution, but has a potentially exponential complexity.

Description
The analysis technique presented in [2] can be used in Preesm to derive bounds for the amount of memory that can be allocated for an application. The upper bound corresponds to the worst memory allocation possible for an application. The lower bound is a theoretical value that limits the minimum amount of memory that can be allocated. By definition, the lower bound is not always reachable, which means that it might be impossible to find an allocation with this optimal amount of memory. The minimum bound is found by solving the Maximum Weight Clique problem on the MEG. This task provides a convenient way to evaluate the quality of a memory allocation. **See also:** Memory Bounds [2, 3].

Documented Errors
None

Serial Memory Bounds Estimator

Graphic Element	Brief Description
Serial Memory Bounds MEGs	Compute bounds of the amount of memory needed to allocate the MEGs
	Plugin identifier
	`org.ietr.preesm.memory.bounds.SerialMemoryBounds-Estimator`

Description
This task computes the memory bounds (see Memory Bound Estimator Task 5.3) for several MEGs, like the one produced by the Memory Allocation task 5.6.

Buffer Merging: Memory Script Runner

Graphic Element	Brief Description
Memory Scripts DAG MemEx scenario MemEx	Executes the memory scripts associated to actors and merge buffers.
	Plugin identifier
	`org.ietr.preesm.memory.script.MemoryScriptTask`

Parameters	Description	
Check	Verification policy used when checking the applicability of the memory scripts written by the developer and associated to the actor. More information on forbidden buffer matching patterns in [7].	
	Value	**Effect**
	`Thorough`	Will generate error messages with a detailed description of the source of the error. This policy should be used when writting memory scripts for the first time.
	`Fast`	All errors in memory script are still detected, but error messages are less verbose. This verification policy is faster than the `Thorough` policy.
	`None`	No verification is performed. Use this policy to speed up workflow execution once all memory scripts have been validated.
Data alignment	Option used to force the allocation of buffers with aligned addresses. The data alignment property should always have the same value as the one set in the properties of the *Memory Allocation* task (See 5.6).	
Log Path	Specify whether, and where, a log of the buffer matching optimization should be generated. Generated log are in the markdown format, and provide information on all matches created by scripts as well as which match could be applied by the optimization process.	
	Value	**Effect**
	`path/file.txt`	The path given in this property is relative to the "Code generation directory" defined in the executed scenario.
	empty	No log will be generated.
Verbose	Verbosity of the workflow task.	
	Value	**Effect**
	`True`	The workflow task will be verbose in the console.
	`False`	The workflow task will be more quiet in the console.

Description
Executes the memory scripts associated to actors and merge buffers. The purpose of the memory scripts is to allow Preesm to allocate input and output buffers of certain actors in overlapping memory range. **See also:** Buffer merging [7]

Documented Errors
None

Memory Allocation

<table>
<tr><th>Graphic Element</th><th>Brief Description</th></tr>
<tr><td rowspan="3">Memory Allocation
MemEx MEGs</td><td>Perform the memory allocation for the given MEG.</td></tr>
<tr><td>Plugin identifier</td></tr>
<tr><td><code>org.ietr.preesm.memory.allocation.MemoryAllocatorTask</code></td></tr>
</table>

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="4">Verbose</td><td colspan="2">Verbosity of the task.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td><code>True</code></td><td>Detailed statistics of the allocation process are logged.</td></tr>
<tr><td><code>False</code></td><td>Logged information is kept to a minimum.</td></tr>
<tr><td rowspan="6">Allocator(s)</td><td colspan="2">Specify which memory allocation algorithm(s) should be used. If the string value of the parameters contains several algorithm names, all will be executed one by one.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td><code>Basic</code></td><td>Each memory object is allocated in a dedicated memory space. Memory allocated for a given object is not reused for other.</td></tr>
<tr><td><code>BestFit</code></td><td>Memory objects are allocated one by one; allocating each object to the available space in memory whose size is the closest to the size of the allocated object. If MEG exclusions permit it, memory allocated for a memory object may be reused for others.</td></tr>
<tr><td><code>FirstFit</code></td><td>Memory objects are allocated one by one; allocating each object to the first available space in memory whose size is the large enough to allocate the object. If MEG exclusions permit it, memory allocated for a memory object may be reused for others.</td></tr>
<tr><td><code>DeGreef</code></td><td>Algorithm adapted from [1]. If MEG exclusions permit it, memory allocated for a memory object may be reused for others.</td></tr>
<tr><td rowspan="6">Distribution</td><td colspan="2">Specify which memory architecture should be used to allocate the memory.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td><code>SharedOnly</code></td><td>(Default) All memory objects are allocated in a single memory bank accessible to all Processing Elements (PEs).</td></tr>
<tr><td><code>Distributed-Only</code></td><td>Each PE is associated to a private memory bank that no other PE can access. (Currently not supported by code generation.)</td></tr>
<tr><td><code>Mixed</code></td><td>Both private memory banks and a shared memory can be used for allocating memory.</td></tr>
<tr><td><code>MixedMerged</code></td><td>Same as mixed, but the memory allocation algorithm favors buffer merging over memory distribution.</td></tr>
</table>

<table>
<tr><th>Parameters</th><th>Description</th></tr>
<tr><td>Best/First Fit order</td><td>When using <code>FirstFit</code> or <code>BestFit</code> memory allocators, this parameter specifies in which order the memory objects will be fed to the allocation algorithm. If the string value associated to the parameters contains several order names, all will be executed one by one.
<table>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>ApproxStableSet</code></td><td>Memory objects are sorted into disjoint stable sets. Stable sets are formed one after the other, each with the largest possible number of object. Memory objects are fed to the allocator set by set and in the largest first order within each stable set.</td></tr>
<tr><td><code>ExactStableSet</code></td><td>Similar to "ApproxStableSet". Stable set are built using an exact algorithm instead of a heuristic.</td></tr>
<tr><td>LargestFirst</td><td>Memory objects are allocated in decreasing order of their size.</td></tr>
<tr><td><code>Shuffle</code></td><td>Memory objects are allocated in a random order. Using the "Nb of Shuffling Tested" parameter, it is possible to test several random orders and only keep the best memory allocation.</td></tr>
<tr><td><code>Scheduling</code></td><td>Memory objects are allocated in scheduling order of their "birth". The "birth" of a memory object is the instant when its memory would be allocated by a dynamic allocator. This option can be used to mimic the behavior of a dynamic allocator. (Only available for MEGs updated with scheduling information).</td></tr>
</table></td></tr>
<tr><td>Data alignment</td><td>Option used to force the allocation of buffers (i.e. Memory objects) with aligned addresses. The data alignment property should always have the same value as the one set in the properties of the Memory Scripts task (See 5.5).
<table>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>None</code></td><td>No special care is taken to align the buffers in memory.</td></tr>
<tr><td><code>Data</code></td><td>All buffers are aligned on addresses that are multiples of their size. For example, a 4 bytes integer is aligned on 4 bytes address.</td></tr>
<tr><td><code>Fixed:=</code>n</td><td>Where $n \in \mathbb{N}^*$. This forces the allocation algorithm to align all buffers on addresses that are multiples of n bytes.</td></tr>
</table></td></tr>
<tr><td>Nb of Shuffling Tested</td><td>Number of random order tested when using the <code>Shuffle</code> value for the Best/First Fit order parameter.
<table>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td>$n \in \mathbb{N}^*$</td><td>Number of random order.</td></tr>
</table></td></tr>
<tr><td>Merge broadcasts</td><td>(Deprecated) Merge memory objects corresponding to outputs of Broadcast actors. This feature was replaced by the more generic Memory Scripts.</td></tr>
</table>

Description
Workflow task responsible for allocating the memory objects of the given MEG. **See also:** Memory Allocation Algorithms [3], Distributed Memory Allocation [6], Broadcast Merging [4].

Documented Errors

`The obtained allocation was not valid because mutually exclusive memory objects have overlapping address ranges. The allocator is not working.`
`<List of memory objects>`
When checking the result of a memory allocation, two memory objects linked with an exclusion in the MEG were allocated in overlapping memory spaces. The error is caused by an invalid memory allocation algorithm and should be corrected in the source code.

`The obtained allocation was not valid because there were unaligned memory objects. The allocator is not working.`
`<List of memory objects>`
When checking the result of a memory allocation, some memory objects were found not to respect the *Dala alignment* parameter. The error is caused by an invalid memory allocation algorithm and should be corrected in the source code.

Code Generation

Static Code Generation

Graphic Element	Brief Description
Code Generation MEGs DAG scenario architecture	Generate code for the application deployment resulting from the workflow execution. **Plugin identifier** `org.ietr.preesm.codegen.xtend.task.CodegenTask`

Parameters	Description
Printer	Specify which printer should be used to generate code. Printers are defined in Preesm source code using an extension mechanism that make it possible to define a single printer name for several targeted architecture. Hence, depending on the type of PEs declared in the architecture model, Preesm will automatically select the associated printer class, if it exists.

Value	Effect
`C`	Print C code and shared-memory based communications. Currently compatible with `x86`, `c6678`, and `arm` architectures.
`InstrumentedC`	Print C code instrumented with profiling code, and shared-memory based communications. Currently compatible with `x86`, `c6678` architectures.
`XML`	Print XML code with all informations used by other printers to print code. Compatible with `x86`, `c6678`.

Description
This workflow task is responsible for generating code for the application deployment resulting from the workflow execution.

Documented Errors
None

Exporter/Importer for Third-Party Dataflow Frameworks

SDF3 Exporter

Graphic Element	Brief Description
SDF3 Exporter SDF architecture scenario	Export a `*.xml` file conforming the SDF For Free (SDF3) format. **Plugin identifier** `org.ietr.preesm.algorithm.exportSdf3Xml.Sdf3Exporter`

Parameters	Description	
path	Path of the exported `*.xml` file. If the specified directory does not exist, it will not be created.	
	Value	**Effect**
	`path/in/proj/name.xml`	Path within the Preesm project containing the workflow where the "SDF3 Exporter" task is instantiated. Exported SDF graph will be named using the string with the `xml` extension at the end of the given path. If a graph with this name already exists in the given path, it will be overwritten. Example: `Code/generated/sdf3/myexport.xml`

Description
This task generates SDF3 code modeling the given SDF graph. SDF modeling in SDF3 follow the specification introduced by Stuijk et al. in [17]. **Known Limitations:** Here is a list of known limitations of the SDF3 importation process: Only SDF graphs can be imported, Actors of the SDF cannot be implemented on more than one processor type, Timings cannot depend on parameters since SDF3 does not support parameterized SDF.

Documented Errors
None

SDF3 Importer

DIF Exporter

Promela Exporter

Graphic Element	Brief Description
Promela Exporter SDF scenario	Generate a `*.pml` file modeling the given SDF graph. **Plugin identifier** `org.ietr.preesm.algorithm.exportPromela.PromelaExporter`

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">path</td><td colspan="2">Path of the exported <code>*.pml</code> file. If the specified directory does not exist, it will not be created.</td></tr>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>path/in/proj/name.pml</code></td><td>Path within the PREESM project containing the workflow where the "Promela Exporter" task is instantiated.
Exported SDF graph will be named using the string with the <code>pml</code> extension at the end of the given path. If a graph with this name already exists in the given path, it will be overwritten.
Example: <code>Code/generated/promela/myexport.pml</code></td></tr>
<tr><td rowspan="4">FIFO allocation policy</td><td colspan="2">This parameter is used to select how FIFOs will be allocated in memory in the generated Promela code.</td></tr>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>Separated</code></td><td>(Default) Each FIFO is assumed to be allocated in a dedicated, separate memory space. The total amount of memory needed to run the application is the sum of the maximum number of data tokens stored in all FIFOs during an SDF graph iteration.</td></tr>
<tr><td><code>Shared</code></td><td>All FIFOs are assumed to be allocated in a shared memory space. The total amount of memory needed to run the application is the maximum number of data tokens stored in all FIFOs during an SDF graph iteration.</td></tr>
<tr><td rowspan="4">Synchronous production/ consumption</td><td colspan="2">This parameter is used to select how SDF actor firings are modeled in the generated <code>*.pml</code> file</td></tr>
<tr><th>Value</th><th>Effect</th></tr>
<tr><td><code>true</code></td><td>(Default) When an SDF actor is executed, data tokens are consumed on its input ports and produced on its output ports simultaneously. This means that produced and consumed tokens are not present in input and output FIFOs simultaneously.</td></tr>
<tr><td><code>false</code></td><td>When an SDF actor is executed, data tokens are consumed on its input ports and later produced on its output ports in two separate, non-simultaneous steps.</td></tr>
</table>

Description
This task generates Promela code modeling the given SDF graph. SDF modeling in Promela follow the specification introduced by Geilen et al. in [8].

Documented Errors
None

Analysis

Gantt Display

Graphic Element	Brief Description
Gantt Display scenario ABC	Displays the result of a mapping/scheduling algorithm as a Gantt diagram. **Plugin identifier** `org.ietr.preesm.plugin.mapper.plot`

Parameters	Description
None	

Description
When executed, this workflow task opens a new tab in PREESM where the result of a mapping/scheduling workflow task is displayed. The tab itslef consists of three subtabs containing: a Gantt diagram, statistics on the computational load of each core, and a speedup assessment chart. **See also:** Speedup assessment chart [13].

Documented Errors
None

Gantt Exporter

<table>
<tr><th>Graphic Element</th><th>Brief Description</th></tr>
<tr><td rowspan="3">Gantt Exporter
ABC
scenario</td><td>This task exports scheduling results as a *.pgantt file that can be viewed using the ganttDisplay viewer.</td></tr>
<tr><td>Plugin identifier</td></tr>
<tr><td>org.ietr.preesm.stats.exporter.StatsExporterTask</td></tr>
</table>

<table>
<tr><th>Parameters</th><th colspan="2">Description</th></tr>
<tr><td rowspan="3">path</td><td colspan="2">Path of the exported *.pgantt file. If the specified directory does not exist, it will not be created.</td></tr>
<tr><td>Value</td><td>Effect</td></tr>
<tr><td>/path/in/proj</td><td>Path within the PREESM project containing the workflow where the "Gantt Exporter" task is instantiated.
Exported Gantt will be named as follows: /path/in/proj/<scenario_name>_stats.pgantt. If a graph with this name already exists in the given path, it will be overwritten.</td></tr>
</table>

Description
This task exports scheduling results as a *.pgantt file that can be viewed using the *ganttDisplay* viewer. The exported *.pgantt file uses the XML syntax. **See also:** ganttDisplay project: https://github.com/preesm/preesm-apps/tree/master/GanttDisplay.

Documented Errors
None

Memory Bounds Estimator

See Section 5.3.

Bibliography

[1] E. De Greef, F. Catthoor, and H. De Man. Array placement for storage size reduction in embedded multimedia systems. *ASAP*, 1997.

[2] K. Desnos, M. Pelcat, J.-F. Nezan, and S. Aridhi. Memory bounds for the distributed execution of a hierarchical synchronous data-flow graph. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XII), 2012 International Conference on*, 2012.

[3] K. Desnos, M. Pelcat, J.-F. Nezan, and S. Aridhi. Pre-and post-scheduling memory allocation strategies on MPSoCs. In *Electronic System Level Synthesis Conference (ESLsyn)*, 2013.

[4] K. Desnos, M. Pelcat, J.-F. Nezan, and S. Aridhi. Memory analysis and optimized allocation of dataflow applications on shared-memory MPSoCs. *Journal of Signal Processing Systems, Springer*, 2014.

[5] K. Desnos, M. Pelcat, J.-F. Nezan, S.S. Bhattacharyya, and S. Aridhi. PiMM: Parameterized and interfaced dataflow meta-model for MPSoCs runtime reconfiguration. In *Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS XIII), 2013 International Conference on*, pages 41–48. IEEE, 2013.

[6] Karol Desnos, Maxime Pelcat, Jean-François Nezan, and Slaheddine Aridhi. Distributed memory allocation technique for synchronous dataflow graphs. In *Signal Processing System (SiPS), Workshop on*, pages 1–6. IEEE, 2016.

[7] Karol Desnos, Maxime Pelcat, Jean-François Nezan, and Slaheddine Aridhi. On memory reuse between inputs and outputs of dataflow actors. *ACM Transactions on Embedded Computing Systems*, 15(30):25, January 2016.

[8] Marc Geilen, Twan Basten, and Sander Stuijk. Minimising buffer requirements of synchronous dataflow graphs with model checking. In *Design Automation Conference*, pages 819–824, NY, USA, 2005. ACM.

[9] Sudeep Kanur, Johan Lilius, and Johan Ersfolk. Detecting data-parallel synchronous dataflow graphs. Technical Report 1184, 2017.

[10] Yu-Kwong Kwok. High-performance algorithms for compile-time scheduling of parallel processors. 1997.

[11] E.A. Lee and D.G. Messerschmitt. Synchronous data flow. *Proceedings of the IEEE*, 75(9):1235 – 1245, sept. 1987.

[12] Patric R. J. Östergård. A new algorithm for the maximum-weight clique problem. *Nordic J. of Computing*, 8(4):424–436, December 2001.

[13] Maxime Pelcat. *Prototypage Rapide et Génération de Code pour DSP Multi-Coeurs Appliqués à la Couche Physique des Stations de Base 3GPP LTE*. PhD thesis, INSA de Rennes, 2010.

[14] Maxime Pelcat, Pierrick Menuet, Slaheddine Aridhi, and Jean-François Nezan. Scalable compile-time scheduler for multi-core architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1552–1555. European Design and Automation Association, 2009.

[15] J. Piat, S.S. Bhattacharyya, and M. Raulet. Interface-based hierarchy for synchronous data-flow graphs. In *SiPS Proceedings*, 2009.

[16] J.L. Pino, S.S. Bhattacharyya, and E.A. Lee. *A hierarchical multiprocessor scheduling framework for synchronous dataflow graphs.* Electronics Research Laboratory, College of Engineering, University of California, 1995.

[17] S. Stuijk, M. Geilen, and T. Basten. Sdf3: Sdf for free. In *Sixth International Conference on Application of Concurrency to System Design (ACSD'06)*, pages 276–278, June 2006.

[18] K. Yamaguchi and S. Masuda. A new exact algorithm for the maximum weight clique problem. In *23rd International Conference on Circuit/Systems, Computers and Communications (ITC-CSCC'08)*, 2008.