A SchedMCore primer

SchedMCore

Eric NOULARD - Eric.Noulard@onera.fr
Claire PAGETTI - Claire.Pagetti@onera.fr

ONERA
THE FRENCH AEROSPACE LAB
http://www.onera.fr/dtim

ETR'2015 in Rennes

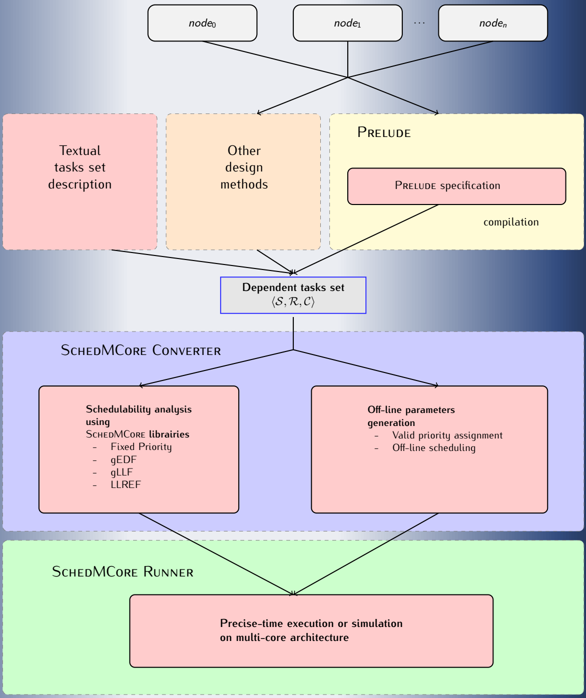
August, 26th 2015

http://sites.onera.fr/schedmcore

https://forge.onera.fr/projects/schedmcore

Plan

SchedMCore overall framework



Contributors

- ONERA
  - Eric Noulard
  - Claire Pagetti
  - Wolfgang Puffistch (former Post-Doc)
  - Luca Santinelli
- LIFL
  - Julien Forget
- ISAE
  - Alexandre Hamez
- Former Students
  - Alessandra Melani (Intern)
  - Julie Baro (Intern)
  - Adrien Charles (Intern)
  - Mikel Cordovilla (PhD)

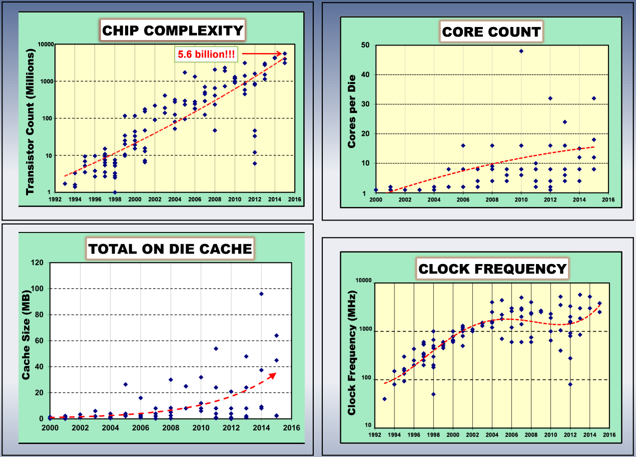
Plan

So many thanks to many contributors

- thanks to Marc Boyer for allowing me to borrow many slides,
- thanks to Pierre-Loïc Garoche and Xavier Thirioux for being polite when I scream about Lustre Compiler,
- thanks to many early SchedMCore users who contributed to its development,
- thanks to many co-coworkers for many fruitful discussions,
- thanks to many coffee breaks,
- thanks to many lawyers who did not think about trademarking/copyrighting/whatevering the word "many"

This presentation could not have been made without them...

Multi-/Many- core are there already



see http://isscc.org/doc/2014/2014_Trends.pdf

Cache memory

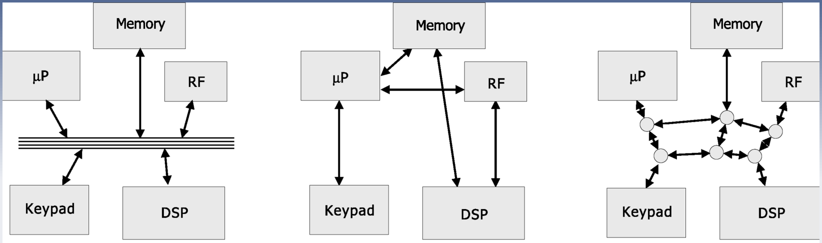
More cores, and more cache

- cache consumes few energy
- cache is efficient

But...

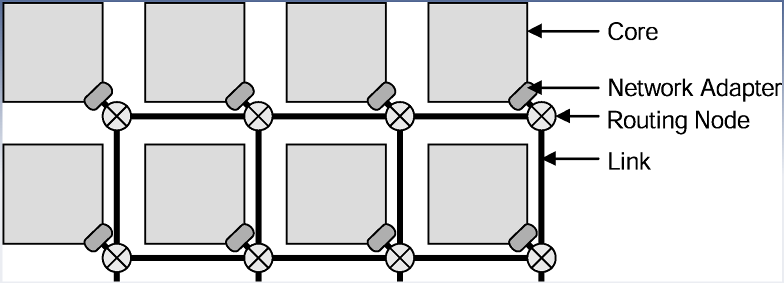
- how to ensure cache coherency with 32 cores ?
- why ?
- local cache or local memory ?
- implicit or explicit communications ?
  - message passing vs shared memory
- an old/new programming way

From bus to NoC



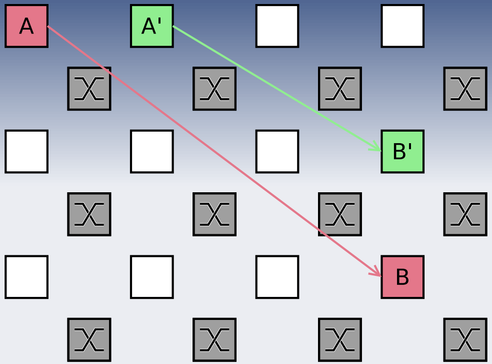
- Bus : shared resource
- Point-to-point : does not scale
- NoC :
  - set of shared resources
  - allow parallel communications

A common vocabulary



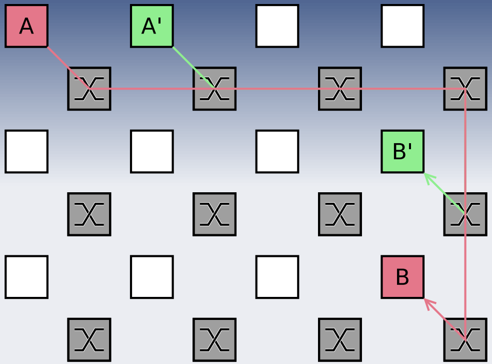
- Core/tile : could be also IO/RAM
  - write/read messages
- Network adapter
  - fragment/reassemble messages into packets
  - send/receive packets
  - flow control
- Routing node : commutation element
  - send/receive flits (≈ 64bits)
  - also flow control

Routing



Routing :

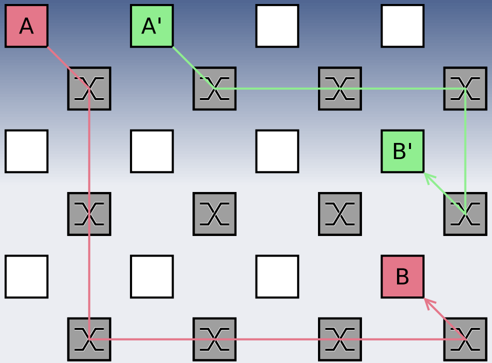
Routing



Routing :

- XY : follows the row first, then moves along the column
  Note : reverse communication uses another path

Routing



Routing :

- XY : follows the row first, then moves along the column
  Note : reverse communication uses another path
- Source routing : source set the path in the header

Routing

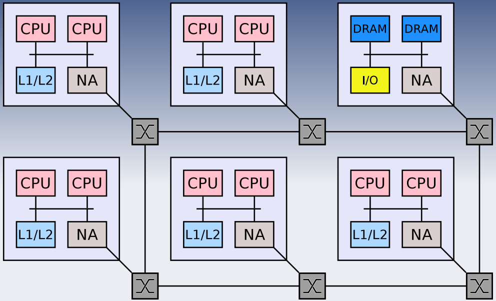
Routing :

- XY : follows the row first, then moves along the column
  Note : reverse communication uses another path
- Source routing : source set the path in the header
- Adaptative :
  - route computed "on the fly"
  - minimize link/router load
  - research only ?

NoC brings network topics

The NoC on many-core brings the "usual" network issue : contention, forwarding policies (store & forward, wormhole, virtual circuit...), this is the work of network expert.

Tile-based solutions



- Initial architecture : MIT, 2007
- Tile :
  - local multi-core
  - DRAM, I/O...
- NoC between tiles
- Hierarchical design

  ⇒ multi-core interferences + NoC interferences

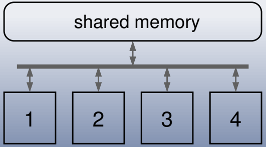
Example of Tiled architectures

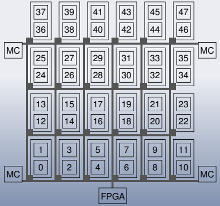
- Intel SCC research processor
  - experimental processor
  - 24 tiles
  - 2 cores per tile
  - 2Tb/s bisection bandwidth
  - explicit message passing (but virtual global addressing)
- Tilera Gx and now Mx processor http://www.tilera.com/
  - COTS solution
  - from 9 up 100 tiles, 1 core per tile.
  - 3-level coherent cache architecture
  - High level programming models (Linux SMP POSIX thread, ZOL, Baremetal)
- Kalray MPPA http://www.kalray.eu/kalray/products
  - COTS solution
  - 16 tiles of 16 cores leading to 256 core chip
  - Shared memory within the 16-core Tile and explicit message passing among tiles.
  - High level programming models (Restricted POSIX, Baremetal, OpenCL)

Multi-Many-Cores

Today : Multi-cores
A few complex cores
Shared on-chip bus
Shared memory
Mostly well understood
Common clock
Communication immediate

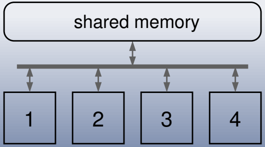
Tomorrow : Many-cores
Lots of simple cores
On-chip network
Message passing
How to use efficiently/safely ?
Clock synchronization
Communication takes time

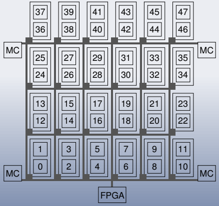




Multi-Many-Cores

Today : Multi-cores	**Tomorrow : Many-cores**
A few complex cores	Lots of simple cores
Shared on-chip bus	On-chip network
Shared memory	Message passing
Mostly well understood	How to use efficiently/safely ?
Common clock	Clock synchronization
Communication immediate	Communication takes time

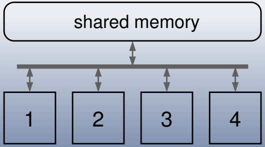


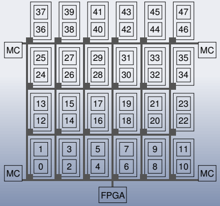


Multi-Many-Cores

Today : Multi-cores	**Tomorrow : Many-cores**
A few complex cores	Lots of simple cores
Shared on-chip bus	On-chip network
Shared memory	Message passing
Mostly well understood	How to use efficiently/safely ?
Common clock	Clock synchronization
Communication immediate	Communication takes time

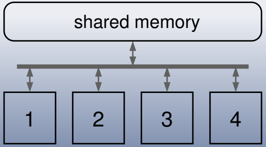
The sliced execution model, see[2]

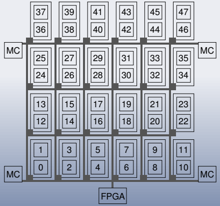




Multi-Many-Cores

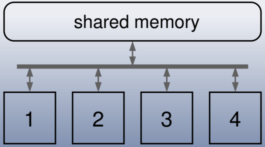
Today : Multi-cores	**Tomorrow : Many-cores**
A few complex cores	Lots of simple cores
Shared on-chip bus	On-chip network
Shared memory	Message passing
Mostly well understood	How to use efficiently/safely ?
Common clock	Clock synchronization
Communication immediate	Communication takes time

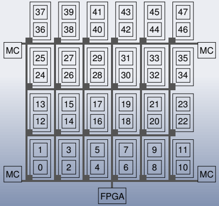




Multi-Many-Cores

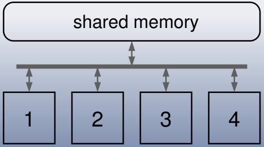
Today : Multi-cores	**Tomorrow : Many-cores**
A few complex cores	Lots of simple cores
Shared on-chip bus	On-chip network
Shared memory	Message passing
Mostly well understood	How to use efficiently/safely ?
Common clock	Clock synchronization
Communication immediate	Communication takes time

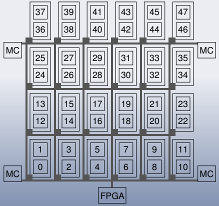




Multi-Many-Cores

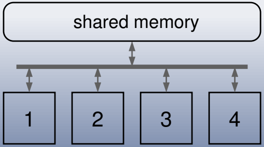
Today : Multi-cores	**Tomorrow : Many-cores**
A few complex cores	Lots of simple cores
Shared on-chip bus	On-chip network
Shared memory	Message passing
Mostly well understood	How to use efficiently/safely ?
Common clock	Clock synchronization
Communication immediate	Communication takes time

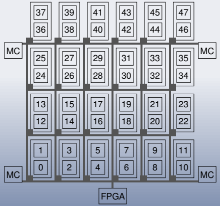




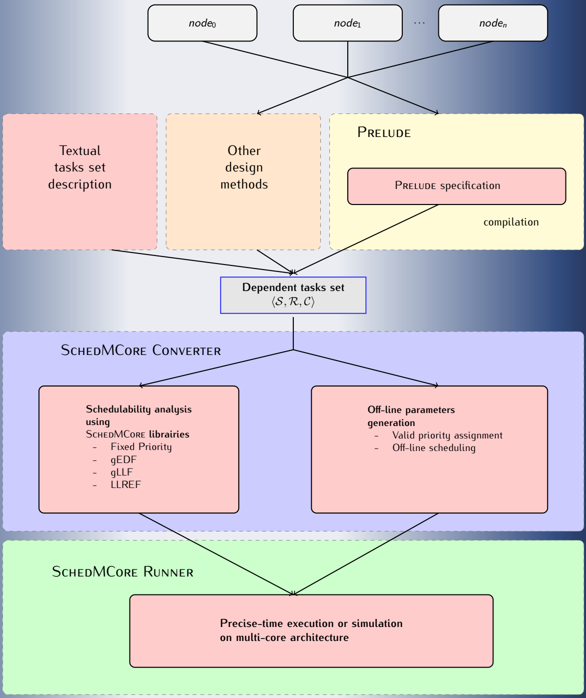
Multi-Many-Cores

Today : Multi-cores	Tomorrow : Many-cores
A few complex cores	Lots of simple cores
Shared on-chip bus	On-chip network
Shared memory	Message passing
Mostly well understood	How to use efficiently/safely ?
Common clock	Clock synchronization
Communication immediate	Communication takes time

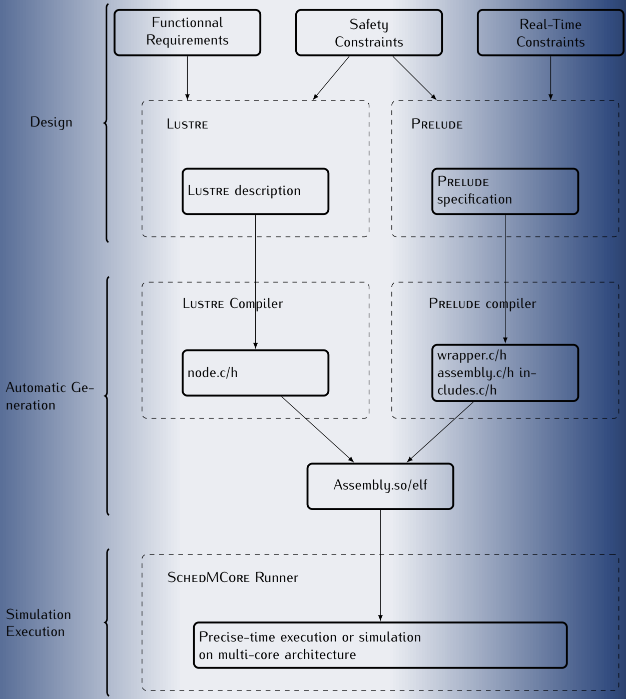




SchedMCore overall framework



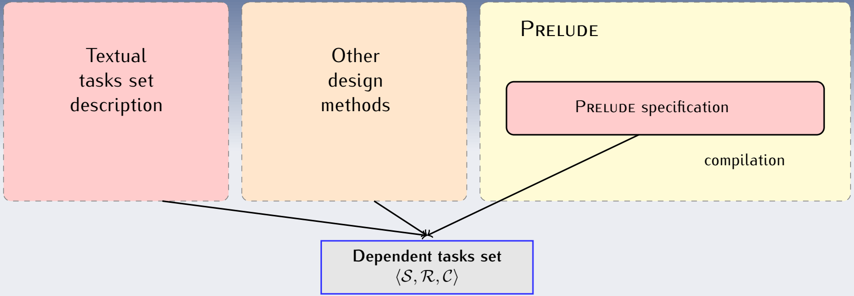
Lustre/Prelude/SchedMCore



Plan

1. Overview
2. The Multi-/Many-core era
3. **SchedMCore Converter**
4. SchedMCore Runner

A reference task model I



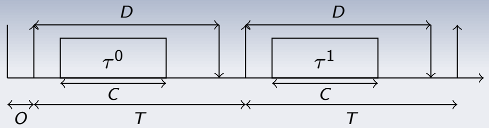
Extendable set of inputs

SchedMCore tools take as input a reference task model which is general enough to be an output of possibly several system modeling tools (Prelude, bare text file, . . .).

A reference task model II

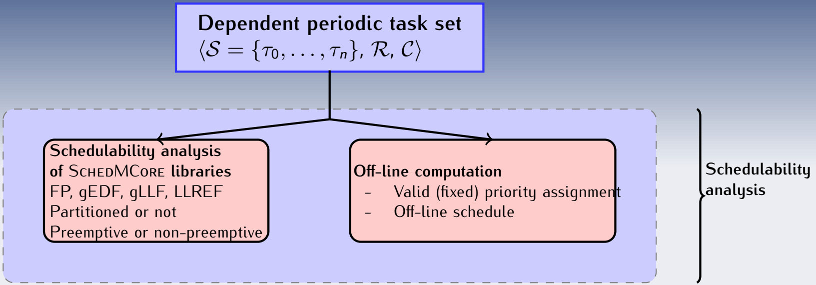
SchedMCore toolset takes as input a set of concurrent periodic and dependent communicating tasks $\langle \mathcal{S}, \mathcal{R}, \mathcal{C}[, \mathcal{M}] \rangle$:

- $\mathcal{S} = \{\tau_j = (T_j, O_j, D_j, C_j)\}_{j=1,\dots n}$ is a finite periodic task set. τ_j^i is the i^{th} job of τ_j ;



- $\mathcal{R}$ is the precedence relation, defined as a set of repetitive job precedence patterns
- $\mathcal{C}$ is the communication function, it tells where each task instance writes or reads its data from (buffer or message).
- $\mathcal{M}$ is an optional partial mapping function which may indicate task placement (on a particular core).

Multiprocessor schedulability analysis : SchedMCore Converter



SchedMCore Converter

A tool for the schedulability analysis of [non]-preemptive global and/or partitioned policies.

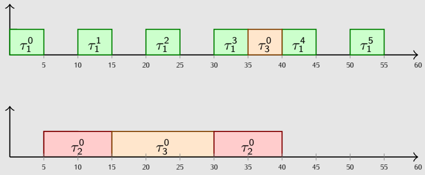
- encoding of the schedulability analysis or the off-line computation as an equivalent configuration automaton ;
- generation of C or Uppaal [or Fiacre] programs for the exploration.

Sequence of configurations

τ_i	T_i	O_i	D_i	C_i	$\mathcal{R}$
τ_1	10	0	10	5	$\tau_1 \xrightarrow{(0,0)} \tau_2$
τ_2	30	0	30	10	$\tau_2 \xrightarrow{(0,0)} \tau_3$
τ_3	60	1	60	20	$\tau_1 \xrightarrow{(0,0)} \tau_3$

policy : FP
with $\tau_1 < \tau_2 < \tau_3$
2 processors

time	0	1	5	10	30	61
τ_1	(10, 0, 10, 5)	(9, 0, 9, 4)	(5, 0, 5, 0)	(0, 0, 0, 0) $\rightarrow$ (10, 0, 10, 5)	(0, 0, 0, 0) $\rightarrow$ (10, 0, 10, 5)	(9, 0, 9, 4)
τ_2	(30, 0, 30, 10)	(29, 0, 29, 10)	(25, 0, 25, 10)	(20, 0, 20, 5)	(0, 0, 0, 0) $\rightarrow$ (30, 0, 30, 10)	(29, 0, 29, 10)
τ_3	(60, 1, 60, 20)	(60, 0, 60, 20)	(56, 0, 56, 20)	(51, 0, 51, 20)	(21, 0, 21, 5)	(0, 0, 0, 0) $\rightarrow$ (60, 0, 60, 20)

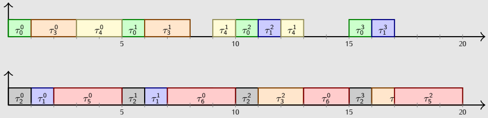


Off-line (optimal) schedule

Generation of a valid schedule on a platform $\mathcal{C}$ made up of p processors.

- Produce of a configuration automaton (available only in Uppaal) ;
- Search a cycle in the sequence of configurations ;
- Combinational explosion ;

	T	C	D	O
τ_0	5	1	1	0
τ_1	5	1	2	0
τ_2	5	1	1	0
τ_3	5	2	3	0
τ_4	8	2	5	0
τ_5	8	3	7	0
τ_6	20	5	19	0




```
# The following line(s) describe the tasks using 1 line per task on each line one find:
# Task "task name" Period  WCET Deadline ReleaseDate
Task "tau 0"  5 1 1 0
Task "tau 1"  5 1 2 0
Task "tau 2"  5 1 1 0
Task "tau 3"  5 2 3 0
Task "tau 4"  8 2 5 0
Task "tau 5"  8 3 7 0
Task "tau 6" 20 5 19 0
```

SchedMCore task file

A simple textual file [9] may be used to describe a task set. E.g., for the task set : $\mathcal{S} = \{\tau_0 = (0,5,5,1), \tau_1 = (0,5,5,1), \tau_2 = (1,5,5,1), \tau_3 = (1,10,10,1), \tau_4 = (1,10,10,1), \tau_5 = (1,20,20,1)\}$ and the associated precedence constraints $\mathcal{R} = \{(\tau_1, \{(0,0)\}, \tau_0), (\tau_1, \{(0,0),(1,0)\}, \tau_3)\}$ the file is :

```
TFF-2.0
# Task "Name" T  C O  (D)
Task   "Tau0" 5  1 0  (5)
Task   "Tau1" 5  1 0  (5)
Task   "Tau2" 5  1 1  (5)
Task   "Tau3" 10 1 1  (10)
Task   "Tau4" 10 1 1  (10)
Task   "Tau5" 20 1 1  (20)
# Dependency "pred" "succs" words
Dependency   "Tau1" "Tau0"  (0:0)
Dependency   "Tau1" "Tau3"  (0:0,1:0)
```

Comments are beginning with # and expand until the end of the line. A task description begins with the Task keyword followed by the name of the task, its period, its WCET (Worst Case Execution Time), its deadline and finally its release date/offset. A precedence constraint begins with Dependency followed by the name of predecessor and successor tasks and the dependency words [5] that define this contraint. The communicatoin scheme $\mathcal{C}$ is not described in this file.

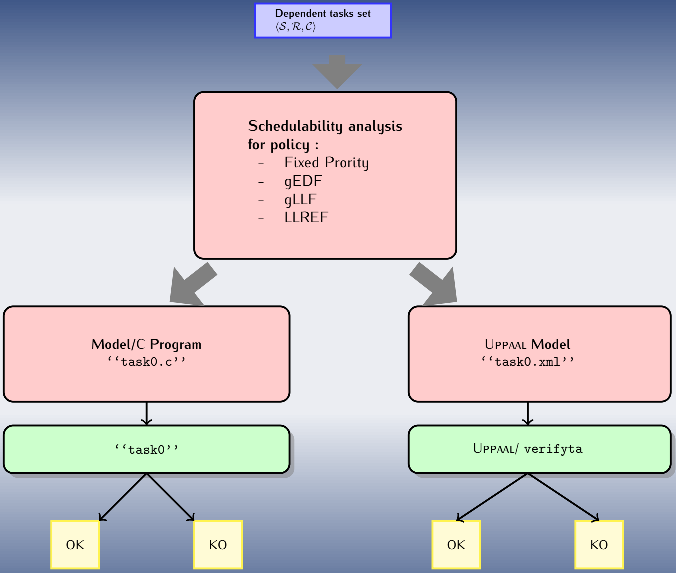
SchedMCore Converter command line I

The SchedMCore Converter tool transforms a task model description into a formally analyzable model in C or Uppaal. In order to use the `lsmc_converter` executable, one can play with several options :

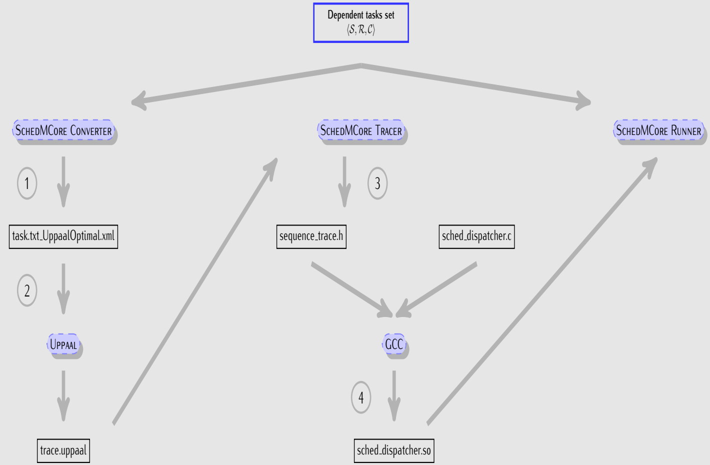
- c : [int] number of processors/cores ;
- m : [Uppaal |C |all] model type to generate Uppaal, C or both (all) ;
- l : [string] name of the input file, if it's a Prelude shared library ;
- t : [string] name of the input file, if it's a textual file ;
- p : [FP|gEDF |gLLF |LLREF |optimalFP|optimal|all] scheduling policy ;
- d : [determinist|undeterminist] deterministic or undeterministic version (only for policy which required it and only in Uppaal).

The -h option of the `lsmc_converter` command gives a complete description of the options and default values.

SchedMCore Converter command line II

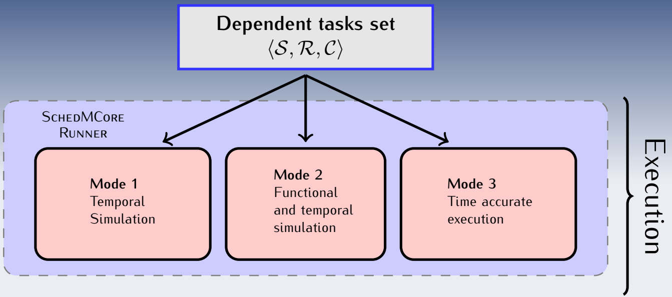


lsmc_tracer workflow



Plan

The runner modes



- mode 1 and 2 are almost the same and shall be used with lsmc_run−nort.
  The only difference is whether if some user functional code is provided or not. Textual task files leads to mode 1 whereas prelude library leads to mode 2.
- mode 3 requires some privileges which can be checked with lsmc_checkCapabilities.

The SchedMCore Runner command line

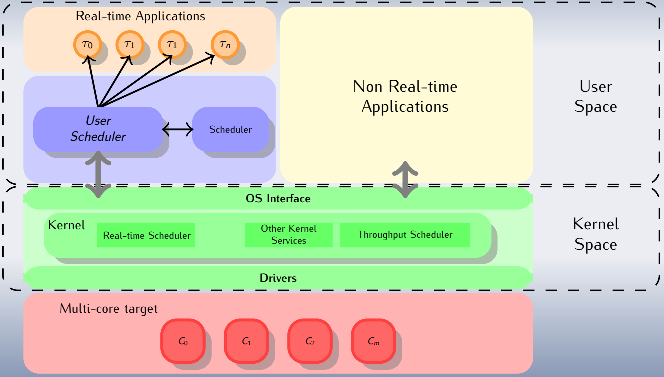
```
$ lsmc_run-nort -h
lsmc_run v1.2
SchedMCore runner tool from the SchedMCore toolset by ONERA
Usage: lsmc_run [-v<level>] [-t <taskfile>|-l <preludeLibFile>] -s <schedulerLibFile>
[-c <nbcore>]
The runner may be used to run a set of tasks described in a tasks file or Prelude library
  -h, --help              Print help and exit
  -V, --version           Print version and exit
  -v, --verbose=STRING    verbose mask (level)  (default=‘0x21’)
  -c, --nb-core=INT       the number of processor core(s)  (default=‘2’)
  -s, --scheduler=STRING  the scheduler to be used  (default=‘edf’)
  -p, --policy=STRING     same as --scheduler   (default=‘edf’)
  -b, --basetime=INT      the base period (in micro-seconds) used for
                            execution  (default=‘1000000’)
  -m, --maxtick=INT       the maximum tick for execution  (default=‘0’)
  -B, --burn              burn cycles when scheduling task set files (default=off)
  -r, --runtime=INT       set affinity of runtime threads to a specific core mask  (default=‘1’)
  -C, --coremask=INT      use only cores in mask  (default=‘-1’)

 Mode: lsmc runner
  lsmc_run [-v=[level]] -t <taskfile> -s <schedulerLibFile> [-c <nbcore> ]
  Run tasks specified in an lsmc task file.
  Available options for this mode are:
  -t, --tasks-file=STRING  the file containing the tasks description (mandatory)
 Mode: prelude runner
  lsmc_run [-v=[level]] -l <preludeLibFile> -s <schedulerLibFile> [-c <nbcore>]
  Run tasks specified in a prelude library file.
  Available options for this mode are:
  -l, --preludelib=STRING  the prelude library containing the tasks description (mandatory)
```

Application execution principles

Layered execution

The execution of an application on a machine equipped with an operating system usually look this way.



Scheduling is an OS decision

Scheduler

The scheduler is traditionally a kernel level task which makes the decision concerning which task runs on which processors.

1. the kernel can preempt or block any user task
2. the scheduler usually implements scheduling classes, i.e. on Linux you have :
   - SCHED_OTHER : the standard round-robin time-sharing policy
   - SCHED_BATCH : for "batch" style execution of processes
   - SCHED_IDLE : for running very low priority background jobs.
   - SCHED_FIFO : a first-in, first-out policy
   - SCHED_RR : a round-robin policy
   - SCHED_DEADLINE : deadline-oriented (Patch http://www.evidence.eu.com/sched_deadline.html)

Scheduling is a kernel activity

The consequence is that when you want to introduce a new scheduling policy you have to work in the kernel.

Scheduling at user level

User level Scheduler

Implements scheduling in userland (not in kernel) built on top of some predictable kernel scheduler.

We assume we have a kernel scheduler with the following properties :

- fixed-priority scheduler with at least 5 priority levels
- preemptive scheduler

Good news

This fits with the specifications of the POSIX SCHED_FIFO scheduler. see : sched_setscheduler (2).

We need more RTOS primitives

Usual RT programming requirement

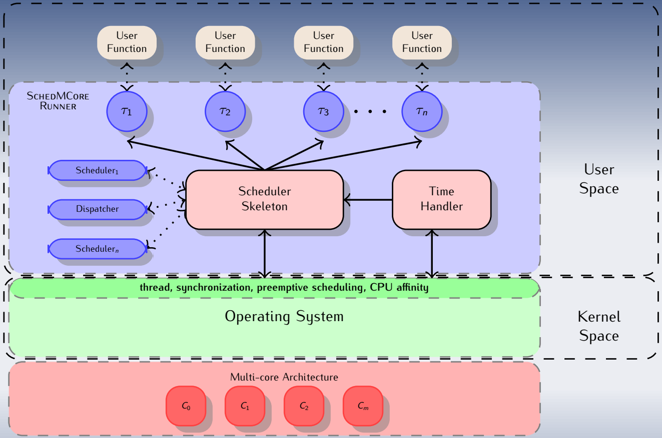
Any serious real-time programming environment should cope with real-time building blocks features

- real-time preemptive scheduling : on POSIX see sched_setscheduler (2)
- processor affinity : on Linux see sched_setaffinity (2)
- interrupt isolation/affinity : on Linux see cat /proc/ interrupts
- physical memory lock : on POSIX see mlockall (2)
- inter-process (or thread) synchronization : on POSIX see e.g. pthread_mutex_lock(P)

SchedMCore runner solution

A user level real-time scheduler which makes it easy to design and implement new real-time policy including task dependencies as a first-class scheduling parameter.

The runner architecture



SchedMCore objectives

Main objective

Enable research experiments on multi-core/multi-processors real-time scheduling from the design of the application to its real-time execution.

SchedMCore objectives

Main objective

Enable research experiments on multi-core/multi-processors real-time scheduling from the design of the application to its real-time execution.

SchedMCore objectives

Main objective

Enable research experiments on multi-core/multi-processors real-time scheduling from the design of the application to its real-time execution.

Some secondary (but important) goals :

1. ease of use
2. modular ⇝ unix way : combine SchedMCore parts or use them independently
3. extensible
4. portable
5. reusable design for real-time embedding

SchedMCore objectives

Main objective

Enable research experiments on multi-core/multi-processors real-time scheduling from the design of the application to its real-time execution.

Some secondary (but important) goals :

1. ease of use
2. modular ⇝ unix way : combine SchedMCore parts or use them independently
3. extensible
4. portable
5. reusable design for real-time embedding

Reusable design

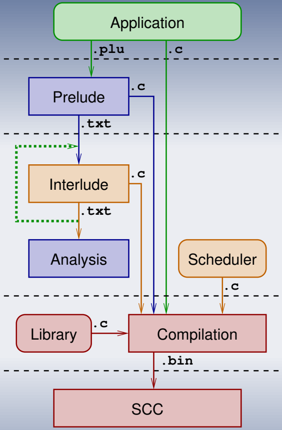
SchedMCore framework is NOT an ready-to-embed environment however its design should be reusable for that purpose and should not have left implementation detail aside since we **do execute** the functional code.

Reusable parts : Intel SCC example [11]

Design reuse

Many ideas and code from SchedMCore and Prelude have been re-used in order to go from design to execution on Intel SCC many-core.

- User provides application, can modify mapping
- Prelude and schedulability analysis (SchedMCore Converter) are generic
- `Interlude` and scheduler (ideas borrowed from SchedMCore Runner) target-specific, but portable
- Library, and compilation target-specific



Going generic

This approach has been genericized (SCC, TI C6678, Tilera Gx36, ...) in [12].

Read the source Luke ! !

Open Source software

SchedMCore, Prelude and Lustre compilers are open source softwares (GPL+LGPL). Go, download, compile, patch, contribute.

- SchedMCore
  - Home site : http://sites.onera.fr/schedmcore
  - Forge : https://forge.onera.fr/projects/schedmcore
    Read-only login : schedmcore – passwd : schedmcore.
  - SVN : https://svn.onera.fr/schedmcore/trunk
  - Bibliography : http://sites.onera.fr/schedmcore/biblio
  - RTFM Please : One may find an on-going draft version of the SchedMCore manual in the schedmcore source : schedmcore/documentation/manual.
    Some technical aspects are described in LSMC technical notes in the schedmcore source : schedmcore/documentation/technotes
- Prelude
  - Home site : http://www.lifl.fr/~forget/prelude.html
  - Forge : http://forge.onera.fr/prelude
    Read-only login : prelude – passwd : prelude.
  - Prelude SVN : https://svn.onera.fr/Prelude/Prelude/trunk
- Lustre Compiler https://cavale.enseeiht.fr/redmine/projects/lustrec

References I

Julie Baro, Frédéric Boniol, Mikel Cordovilla, Eric Noulard, and Claire Pagetti.
Off-line (optimal) multiprocessor scheduling of dependent periodic tasks.
In Proceedings of the 27th ACM Symposium on Applied Computing (SAC'2012), March 2012.
http://www.acm.org/conferences/sac/sac2012/.

Frédéric Boniol, Hugues Cassé, Eric Noulard, and Claire Pagetti.
Deterministic execution model on cots hardware.
In Architecture of Computing Systems - ARCS2012, Lecture Notes in Computer Science. Springer, 2012.
http://www.arcs2012.tum.de/.

Mikel Cordovilla, Frédéric Boniol, Julien Forget, Eric Noulard, and Claire Pagetti.
Developing critical embedded systems on multicore architectures : the prelude-schedmcore toolset.
In 19th International Conference on Real-Time and Network Systems (RTNS 2011), Nantes, France, September 29-30 2011. IRCCyN lab, IRCCyN lab.
http://rtns2011.irccyn.ec-nantes.fr/.

Mikel Cordovilla, Frédéric Boniol, Eric Noulard, and Claire Pagetti.
Multiprocessor schedulability analyzer.
In Proceedings of the 26th ACM Symposium of Applied Computing (SAC'2011), March 2011.
http://sites.onera.fr/schedmcore/sites/sites.onera.fr.schedmcore/files/2011_MSA_cordovilla.pdf.

Julien Forget, Emmanuel Grolleau, Claire Pagetti, and Pascal Richard.
Dynamic priority scheduling of periodic tasks with extended precedences.
In IEEE International Conference on Emerging Technology and Factory Automation (ETFA'11), Toulouse, France, 2011.
http://www.lifl.fr/~forget/docs/forget-ETFA-2011.pdf.

References II

Alessandra Melani, Eric Noulard, and Luca Santinelli.
Learning from probabilities : Dependences within real-time systems.
In Proceedings of the 18th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA'2013), September 2013.
http://www.etfa2013.org/.

Mikel Cordovilla Mesonero.
Environnement de développement d'applications multipériodiques sur plateforme multicœur. La boîte à outil SchedMCore.
PhD thesis, Université de Toulouse, 2012.
http://tel.archives-ouvertes.fr/tel-00720709.

Claire Pagetti, David Saussié, Romain Gratia, Eric Noulard, and Pierre Siron.
The ROSACE case study : From simulink specification to multi/many-core execution.
In Proceedings of the 20th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'2014), April 2014.
http://2014.rtas.org/.

Wolfgang Puffitsch and Alessandra Melani.
Schedmcore task file format.
SchedMCore technical note LSMC-TN002, ONERA/DTIM, 2013.

Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti.
Explicit precedence constraints in safety-critical java.
In Proceedings of the 11th International Workshop on Java Technologies for Real-time and Embedded Systems (JTRES'2013), October 2013.
http://jtres2013.atego.com/.

References III

Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti.
Mapping a multi-rate synchronous language to a many-core processor.
http://www.cister.isep.ipp.pt/rtas2013/.

Wolfgang Puffitsch, Eric Noulard, and Claire Pagetti.
Off-line mapping of multi-rate dependent task sets to many-core platforms.
Real-Time Systems, 51(5) :526–565, September 2015.
http://dx.doi.org/10.1007/s11241-015-9232-1.

Perspectives and on-going work I

More features

Enrich the scheduling analyses features : more on partitioned and non-preemptive scheduling, connect to other real-time oriented languages like Prelude, include a cost model for task communication, tighter connection with other ONERA formal analysis tools, support classical MIF/MAF scheduling analysis out-of-the box...

Go on real Many-core hardware

This is already **done on Intel SCC** see forthcoming publication [11] accepted for RTAS'2013. Nevertheless, we should definitely go further in order to generalize the approach for other many-core (Kalray MPPA, Tilera Gx, ...)

Perspectives and on-going work II

Help with probabilistic WCET evaluations

This is an on-going joint-work (with Luca Santinelli and Alessandra Melani).

- ease dynamic user-function loading (any C function may be linked-in for RT execution)
- enhance the SchedMCore Runner with precise timing trace with LTTNG (Linux-only) in order to collect statistical samples and provides entries for probabilistic WCET evaluation.

Industrialize

Find industrial partners which may be interested to include our knowledge and tools into their industrial product.