Zero-Jitter Task Chains via Algebraic Rings

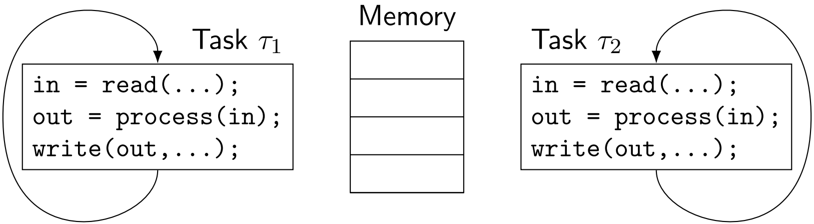
Enrico Bini, Paolo Pazzaglia, Martina Maggio

University of Turin

USC, 03/03/2023

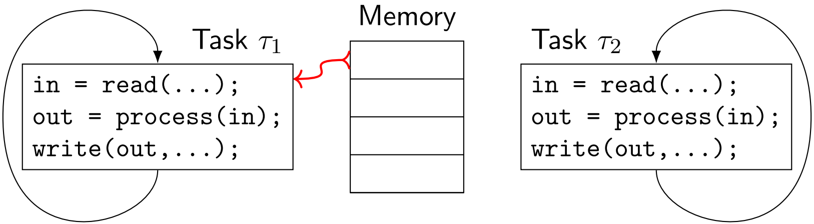
Outline

Motivation of shared memory communication



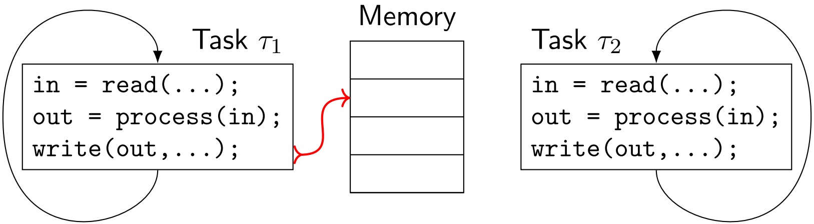
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today

Motivation of shared memory communication



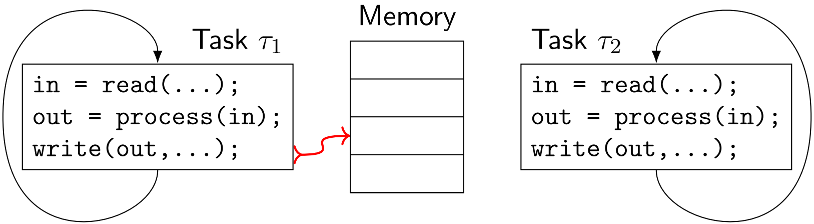
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"...then

Motivation of shared memory communication



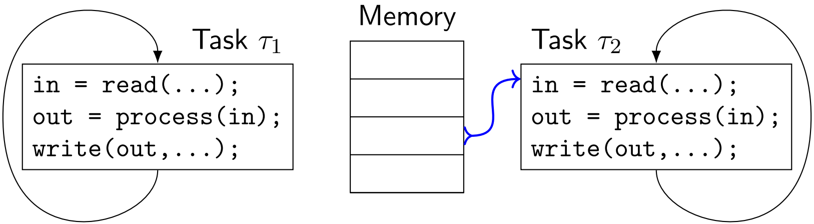
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"... then

Motivation of shared memory communication



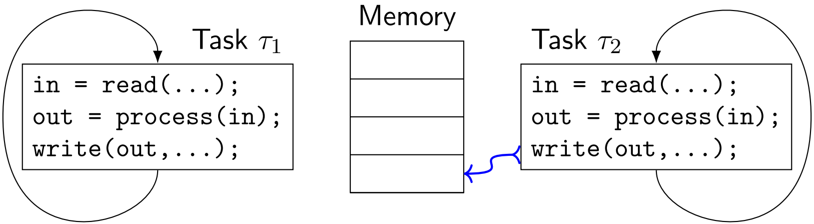
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV/LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"...then

Motivation of shared memory communication



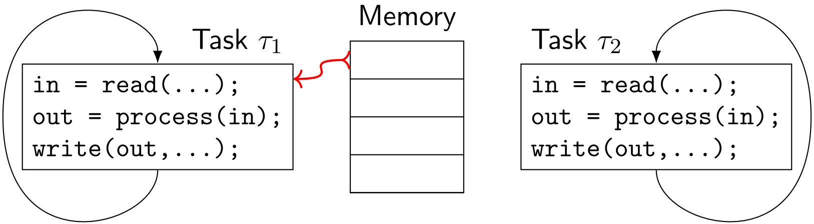
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"...then

Motivation of shared memory communication



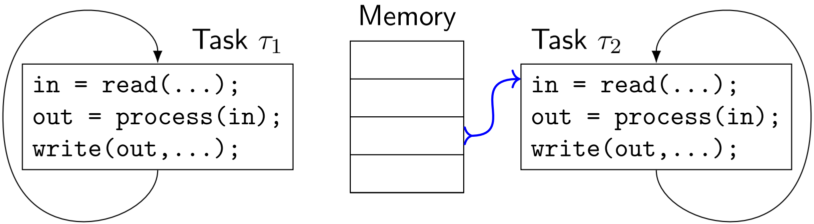
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"…then

Motivation of shared memory communication



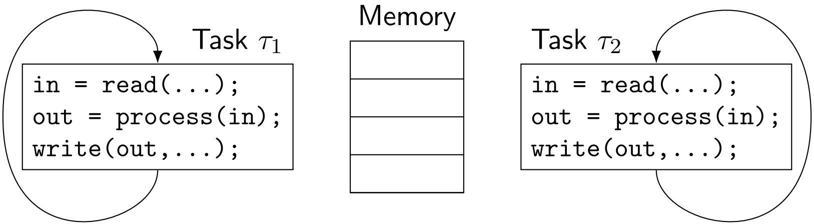
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"... then

Motivation of shared memory communication



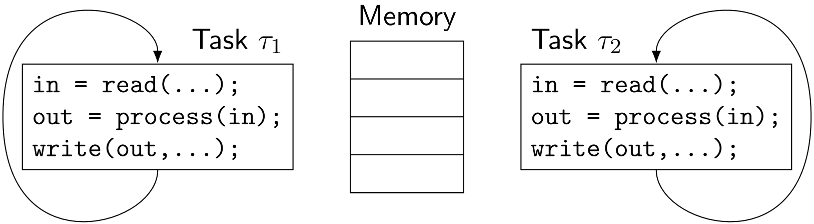
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic $\Rightarrow$ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"... then

Motivation of shared memory communication



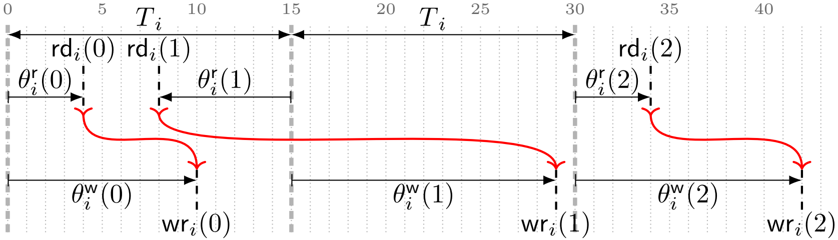
- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic ⇒ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"... then
  2. "Let us analyze how good/bad things can go" ... and

Motivation of shared memory communication



- Communication through shared memory is very popular in automotive applications
  - pros: very efficient (just a `MOV`/`LOAD` instruction)
  - cons: very basic ⇒ risk of inconsistent data
- The story presented today
  1. "Let the guys (tasks) go, write and read in freedom (no timestamping, no message queues, or other higher level fancy mechanisms)"... then
  2. "Let us analyze how good/bad things can go" ...and
  3. "Let's mitigate (or prevent) issues, if any, if feasible"

Temporal model of a task τ_i



- A task, denoted by τ_i, is composed by recurrent jobs
  - $\mathbb{J}_i$ denotes the set of jobs and is equal to $\mathbb{Z}$
- Every job $j \in \mathbb{J}_i$ reads, processes, and then writes data (curvy arrow)
- For each job $j \in \mathbb{J}_i$, read/write instants are relative to the *period* T_i

$$\mathsf{rd}_i(j) = j\,T_i + \theta^{\mathsf{r}}_i(j) \qquad \textit{read instant of } \tau_i \text{ job } j$$

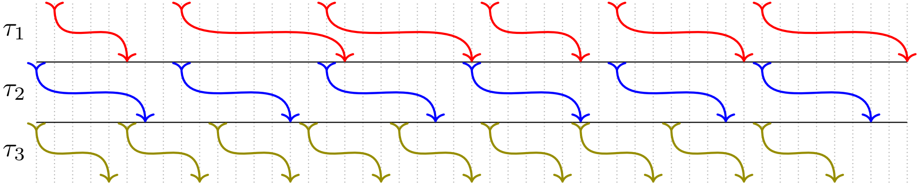
$$\mathsf{wr}_i(j) = j\,T_i + \theta^{\mathsf{w}}_i(j) \qquad \textit{write instant of } \tau_i \text{ job } j$$

- with bounded phasings (if constant with j, then zero jitter, nice)

$\theta^{\mathsf{r}}_i(j)$, *phasing of read instants* $\theta^{\mathsf{w}}_i(j)$, *phasing of write instants*

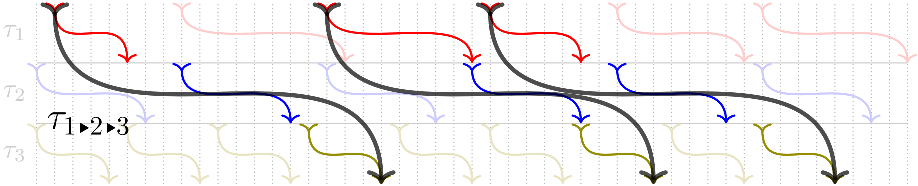
Outline

Model of chains



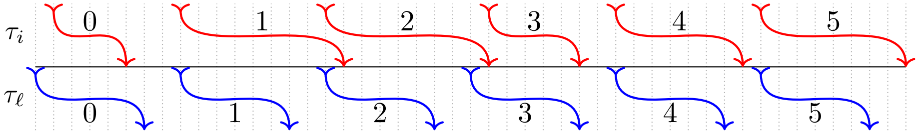
- A chains is the concatenations of tasks (a stack of curvy arrows)
  - a task reads what the preceding task (if any) has written
- Chains perform **the same operations** of tasks, because
  - chains read (the first task does), process (through task processing and communication), and write (the last task does)
  - chains are recurrent

Model of chains



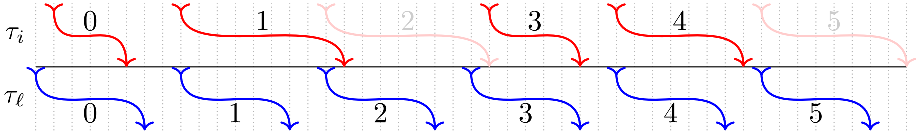
- A chains is the concatenations of tasks (a stack of curvy arrows)
  - a task reads what the preceding task (if any) has written
- Chains perform **the same operations** of tasks, because
  - chains read (the first task does), process (through task processing and communication), and write (the last task does)
  - chains are recurrent
- We **extend the model**, notations and terminology of tasks to chains
- (Recursive) definition of a *chain*
  1. A *task* τ_i is a chain (denoted by the same τ_i)
  2. The *concatenation* of chains τ_i and τ_ℓ is a chain denoted by $\tau_{i\blacktriangleright\ell}$ (reads "tau i to ℓ")
- This research is about
  - determining the parameters of $\tau_{i\blacktriangleright\ell}$ **from the composing chains** τ_i and τ_ℓ

Jobs of the composition of chains: from $\mathbb{J}_i$ and $\mathbb{J}_\ell$ to $\mathbb{J}_{i\blacktriangleright\ell}$



- The jobs of $\tau_{i\blacktriangleright\ell}$ are a subset $\mathbb{J}_{i\blacktriangleright\ell} \subseteq \mathbb{J}_i \times \mathbb{J}_\ell$. A job (j_i, j_ℓ) belongs to $\mathbb{J}_{i\blacktriangleright\ell}$ if and only if

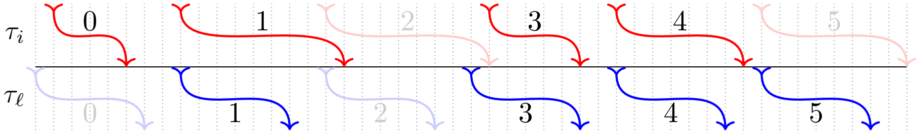
Jobs of the composition of chains: from $\mathbb{J}_i$ and $\mathbb{J}_\ell$ to $\mathbb{J}_{i \blacktriangleright \ell}$



- The jobs of $\tau_{i \blacktriangleright \ell}$ are a subset $\mathbb{J}_{i \blacktriangleright \ell} \subseteq \mathbb{J}_i \times \mathbb{J}_\ell$. A job (j_i, j_ℓ) belongs to $\mathbb{J}_{i \blacktriangleright \ell}$ if and only if
  1. $j_i \in \mathbb{J}_i$ is the **last job to write** for $j_\ell \in \mathbb{J}_\ell$

$$j_i = \max\{j \in \mathbb{J}_i : \mathsf{wr}_i(j) \leqslant \mathsf{rd}_\ell(j_\ell)\}$$

Jobs of the composition of chains: from $\mathbb{J}_i$ and $\mathbb{J}_\ell$ to $\mathbb{J}_{i\blacktriangleright\ell}$



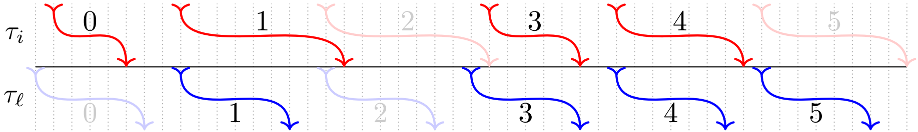
- The jobs of $\tau_{i\blacktriangleright\ell}$ are a subset $\mathbb{J}_{i\blacktriangleright\ell} \subseteq \mathbb{J}_i \times \mathbb{J}_\ell$. A job (j_i, j_ℓ) belongs to $\mathbb{J}_{i\blacktriangleright\ell}$ if and only if
  1. $j_i \in \mathbb{J}_i$ is the **last job to write** for $j_\ell \in \mathbb{J}_\ell$

$$j_i = \max\{j \in \mathbb{J}_i : \mathsf{wr}_i(j) \leqslant \mathsf{rd}_\ell(j_\ell)\}$$

  2. $j_\ell \in \mathbb{J}_\ell$ is the **first job to read** from $j_i \in \mathbb{J}_i$

$$j_\ell = \min\{j \in \mathbb{J}_\ell : \mathsf{wr}_i(j_i) \leqslant \mathsf{rd}_\ell(j)\}.$$

Jobs of the composition of chains: from $\mathbb{J}_i$ and $\mathbb{J}_\ell$ to $\mathbb{J}_{i \blacktriangleright \ell}$



- The jobs of $\tau_{i \blacktriangleright \ell}$ are a subset $\mathbb{J}_{i \blacktriangleright \ell} \subseteq \mathbb{J}_i \times \mathbb{J}_\ell$. A job (j_i, j_ℓ) belongs to $\mathbb{J}_{i \blacktriangleright \ell}$ if and only if
  1. $j_i \in \mathbb{J}_i$ is the **last job to write** for $j_\ell \in \mathbb{J}_\ell$

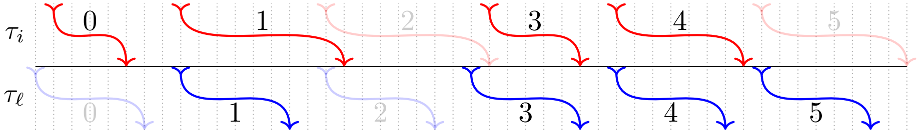
$$j_i = \max\{j \in \mathbb{J}_i : \mathsf{wr}_i(j) \leqslant \mathsf{rd}_\ell(j_\ell)\}$$

  2. $j_\ell \in \mathbb{J}_\ell$ is the **first job to read** from $j_i \in \mathbb{J}_i$

$$j_\ell = \min\{j \in \mathbb{J}_\ell : \mathsf{wr}_i(j_i) \leqslant \mathsf{rd}_\ell(j)\}.$$

- called "last-to-first" semantic: widely used, linked to Sample-and-Hold

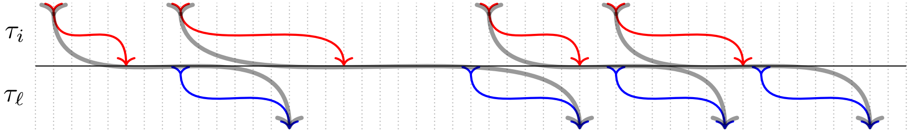
Jobs $\mathbb{J}_{i \blacktriangleright \ell}$ of a chain $\tau_{i \blacktriangleright \ell}$ are isomorphic to $\mathbb{Z}$



- Theorem: $\mathbb{J}_{i \blacktriangleright \ell}$ is isomorphic to $\mathbb{Z}$ (by induction)
  1. (base case) if τ_i is a task, easy
  2. (inductive step, sketch) ...
- We interchangeably use $(j_i, j_\ell) \in \mathbb{J}_{i \blacktriangleright \ell}$ or $j_{i \blacktriangleright \ell} \in \mathbb{Z}$
  - when using $(j_i, j_\ell) \in \mathbb{J}_{i \blacktriangleright \ell}$, we mean to underline the writer/reader jobs $j_i \in \mathbb{J}_i$ and $j_\ell \in \mathbb{J}_\ell$
  - when using $j_{i \blacktriangleright \ell} \in \mathbb{Z}$, we mean to highlight the position of the job $j_{i \blacktriangleright \ell}$ w.r.t. other earlier/later jobs in $\mathbb{J}_{i \blacktriangleright \ell}$
- The set $\mathbb{J}_{i \blacktriangleright \ell}$ is **totally ordered**

$(j_i, j_\ell) \in \mathbb{J}_{i \blacktriangleright \ell}$	$j_{i \blacktriangleright \ell} \in \mathbb{Z}$
$(0, 1)$	0
$(1, 3)$	1
$(3, 4)$	2
$(4, 5)$	3

Read/write instants of $\tau_{i \blacktriangleright \ell}$



- For any $(j_i, j_\ell) \in \mathbb{J}_{i \blacktriangleright \ell}$
  - the read instant is $\mathsf{rd}_{i \blacktriangleright \ell}(j_i, j_\ell) = \mathsf{rd}_i(j_i)$
  - the write instant is $\mathsf{wr}_{i \blacktriangleright \ell}(j_i, j_\ell) = \mathsf{wr}_\ell(j_\ell)$
- Period of $\tau_{i \blacktriangleright \ell}$
  - (existence) Is there any period $T_{i \blacktriangleright \ell}$ that for any $j \in \mathbb{J}_{i \blacktriangleright \ell}$ allows us writing?

$$\mathsf{rd}_{i \blacktriangleright \ell}(j) = j\,T_{i \blacktriangleright \ell} + \theta^{\mathsf{r}}_{i \blacktriangleright \ell}(j)$$
$$\mathsf{wr}_{i \blacktriangleright \ell}(j) = j\,T_{i \blacktriangleright \ell} + \theta^{\mathsf{w}}_{i \blacktriangleright \ell}(j)$$

    with bounded phasings $\theta^{\mathsf{r}}_{i \blacktriangleright \ell}(j)$ and $\theta^{\mathsf{w}}_{i \blacktriangleright \ell}(j)$
  - If it exists, what is the relation of $T_{i \blacktriangleright \ell}$ with T_i and T_ℓ? (question to the audience)
- In this research, we investigate these questions in the case of *Logic Execution Time (LET)*

Outline

LET tasks

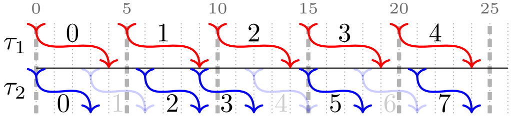
- In LET, reads and writes happen at pre-determined instants, **indep. of schedule**

$$\text{phasings are constant} \qquad \forall j \in \mathbb{J}_i, \qquad \theta_i^{\mathsf{r}}(j) = \theta_i^{\mathsf{r}}, \quad \theta_i^{\mathsf{w}}(j) = \theta_i^{\mathsf{w}}$$

- Logic Execution Time (LET): the execution time is logic (and **constant**)
  - it is independent of scheduling decisions
  - it becomes a constraint for the scheduler
- LET requires a (lightweight) copying mechanism
- LET eliminates the jitter
- LET introduces some additional delay
- What happens to chains of LET tasks? Are they LET chains (with zero jitter)?

Chain of 2 LET tasks

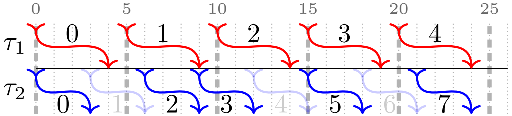
	T_i	θ_i^{r}	θ_i^{w}
τ_1	5	0	4
τ_2	3	0	3



$(j_1, j_2) \in \mathbb{J}_{1\blacktriangleright 2}$	$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$
$(0, 2)$	0	0	9
$(1, 3)$	1	0	7
$(2, 5)$	2	0	8
$(3, 7)$	3	0	9

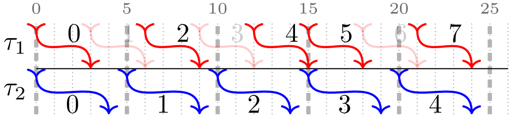
Chain of 2 LET tasks

	T_i	θ^{r}_i	θ^{w}_i
τ_1	5	0	4
τ_2	3	0	3



$(j_1, j_2) \in \mathbb{J}_{1\blacktriangleright 2}$	$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$
$(0, 2)$	0	0	9
$(1, 3)$	1	0	7
$(2, 5)$	2	0	8
$(3, 7)$	3	0	9

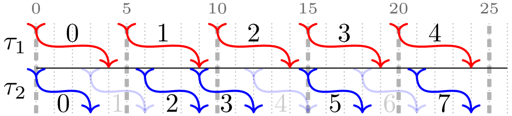
	T_i	θ^{r}_i	θ^{w}_i
τ_1	3	0	3
τ_2	5	0	4



$(j_1, j_2) \in \mathbb{J}_{1\blacktriangleright 2}$	$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$
$(-1, 0)$	0	-3	4
$(0, 1)$	1	-5	4
$(2, 2)$	2	-4	4
$(4, 3)$	3	-3	4

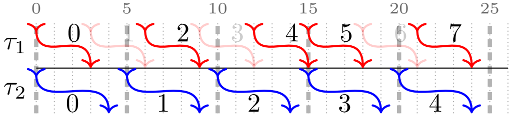
Chain of 2 LET tasks

	T_i	θ^{r}_i	θ^{w}_i
τ_1	5	0	4
τ_2	3	0	3



$(j_1, j_2) \in \mathbb{J}_{1\blacktriangleright 2}$	$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$
$(0, 2)$	0	0	9
$(1, 3)$	1	0	7
$(2, 5)$	2	0	8
$(3, 7)$	3	0	9

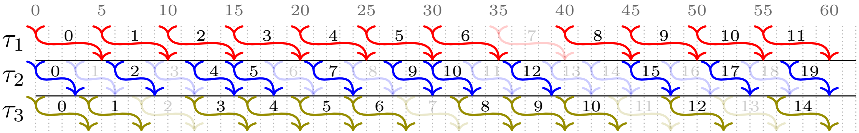
	T_i	θ^{r}_i	θ^{w}_i
τ_1	3	0	3
τ_2	5	0	4



$(j_1, j_2) \in \mathbb{J}_{1\blacktriangleright 2}$	$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$
$(-1, 0)$	0	-3	4
$(0, 1)$	1	-5	4
$(2, 2)$	2	-4	4
$(4, 3)$	3	-3	4

- $T_{1\blacktriangleright 2} = \max\{T_1, T_2\}$, we cannot expect any lower value
- the phasings are not constant: a chain of 2 LET tasks is not a LET chain
  - however, we have a fix for this

Chain of 3 LET tasks



$(j_1, j_2, j_3) \in \mathbb{J}_{1\blacktriangleright 2\blacktriangleright 3}$
$(0, 2, 3)$
$(1, 4, 4)$
$(2, 5, 5)$
$(3, 7, 6)$
$(4, 9, 8)$
$(5, 10, 9)$
$(6, 12, 10)$
$(8, 15, 12)$
$(9, 17, 14)$
$(10, 19, 15)$
$(11, 20, 16)$

- Example with $T_1 = 5$, $T_2 = 3$, $T_3 = 4$ illustrated above
- The pattern repeats every $\mathrm{lcm}(T_1, T_2, T_3) = 60$
- One job of the larger period task ($j_1 = 7$ above) is canceled!!
- Only 11 jobs in $\mathbb{J}_{1\blacktriangleright 2\blacktriangleright 3}$ released every 60

$$T_{1\blacktriangleright 2\blacktriangleright 3} = \frac{60}{11} > 5 = \max\{T_1, T_2, T_3\}.$$

Outline

1. Model of tasks
2. Composing tasks into chains
3. Logic Execution Time (LET)
4. Analysis of chains of LET tasks

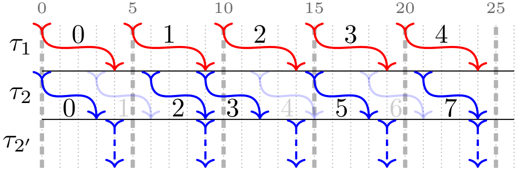
Contribution

- Summary of discoveries, so far
  1. Communication through shared memory is popular in embedded systems
  2. Timing of communication needs to be studies
  3. LET eliminates jitter for tasks
  4. However, chains of LET tasks may be slower than the slowest task (bad)
- Our contribution
  - When composing two tasks, we add a third task that regularizes the pattern, making the chain phasings constant
    - such a composition preserves the period of the largest task
  - Chains of LET tasks can then be composed arbitrarily, preserving the constant phasing (zero-jitter)

Making a zero jitter chain of 2 LET tasks: add a copier task

- if $T_1 \geqslant T_2$ then we add $\tau_{2'}$ after τ_2

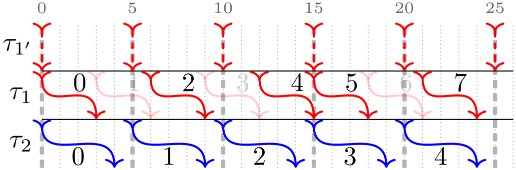
	T_i	θ^{r}_i	θ^{w}_i
τ_1	5	0	4
τ_2	3	0	3
$\tau_{2'}$	5	9	9



$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2\blacktriangleright 2'}$
0	0	9	9
1	0	7	9
2	0	8	9
3	0	9	9

- if $T_1 \leqslant T_2$ then we add $\tau_{1'}$ before τ_1

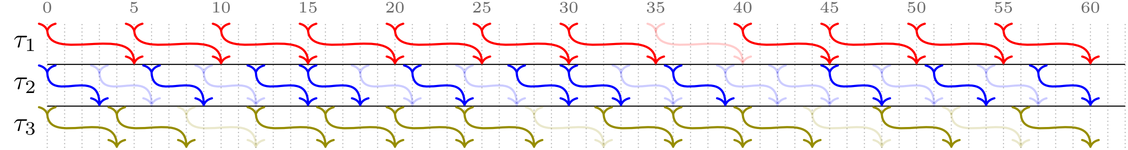
	T_i	θ^{r}_i	θ^{w}_i
$\tau_{1'}$	5	-5	-5
τ_1	3	0	3
τ_2	5	0	4



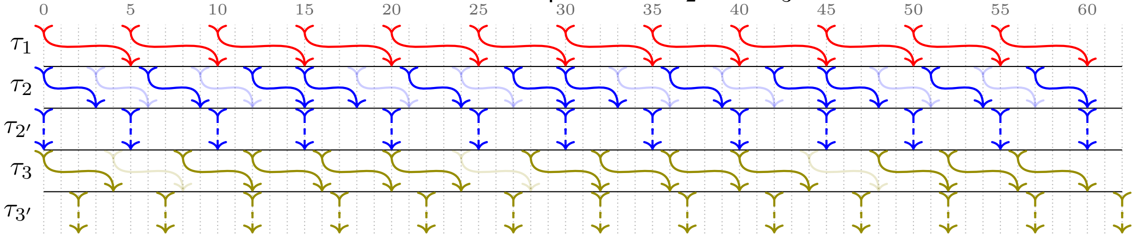
$j_{1\blacktriangleright 2}$	$\theta^{\mathsf{r}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1\blacktriangleright 2}(j_{1\blacktriangleright 2})$	$\theta^{\mathsf{w}}_{1'\blacktriangleright 2\blacktriangleright 2}$
0	-3	4	-5
1	-5	4	-5
2	-4	4	-5
3	-3	4	-5

Chain of 3 LET tasks (revised)

without the extra task



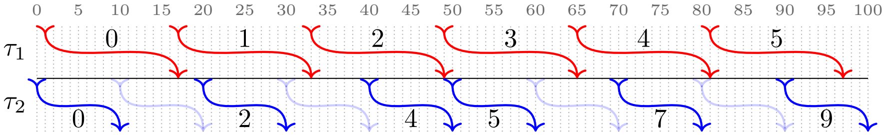
with the extra copier tasks $\tau_{2'}$ and $\tau_{3'}$



- Now 12 jobs in $\mathbb{J}_{1 \blacktriangleright 2 \blacktriangleright 2' \blacktriangleright 3 \blacktriangleright 3'}$ exists in $\mathrm{lcm}(T_1, T_2, T_3) = 60$
- The period then is: $T_{1 \blacktriangleright 2 \blacktriangleright 2' \blacktriangleright 3 \blacktriangleright 3'} = \max\{T_1, T_2, T_3\} = 5$
- Phasings are constant (zero jitter): $\theta^{\mathsf{r}}_{1 \blacktriangleright 2 \blacktriangleright 2' \blacktriangleright 3 \blacktriangleright 3'} = 0$, $\theta^{\mathsf{w}}_{1 \blacktriangleright 2 \blacktriangleright 2' \blacktriangleright 3 \blacktriangleright 3'} = 17$

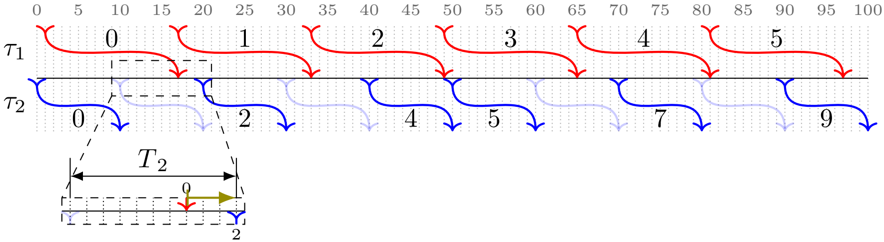
Parameters of the copier task

- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$



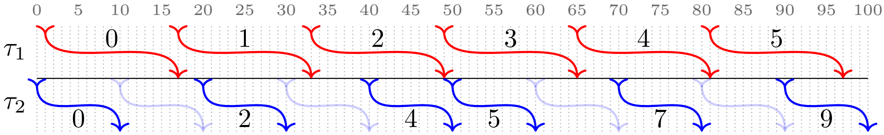
Parameters of the copier task

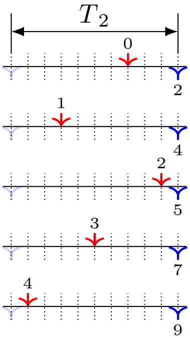
- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$



Parameters of the copier task

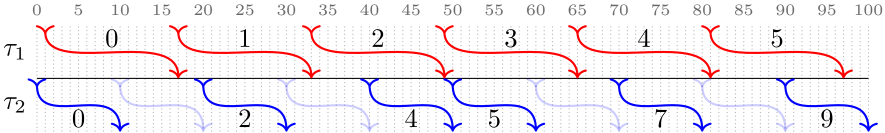
- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$

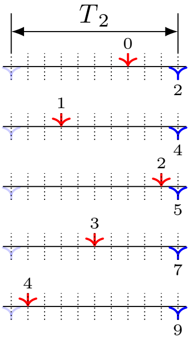


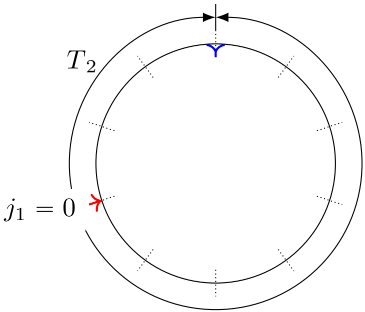


Parameters of the copier task

- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$

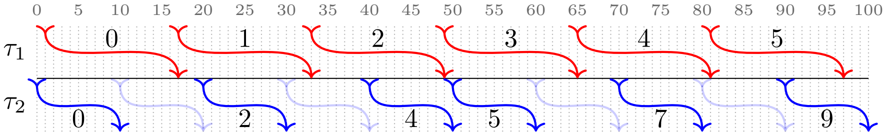


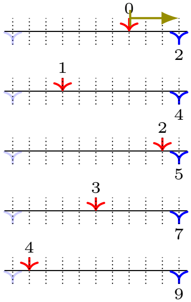


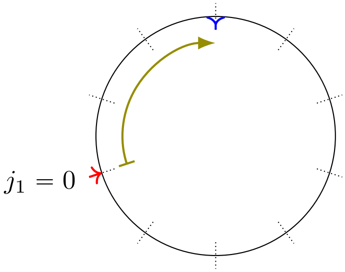


Parameters of the copier task

- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$

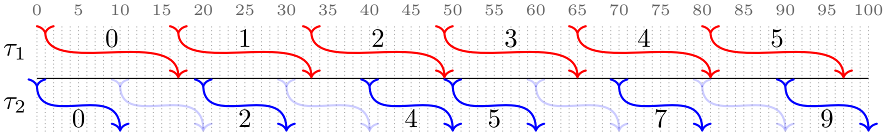


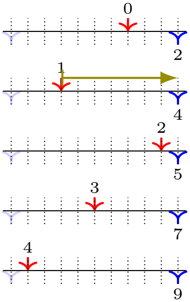


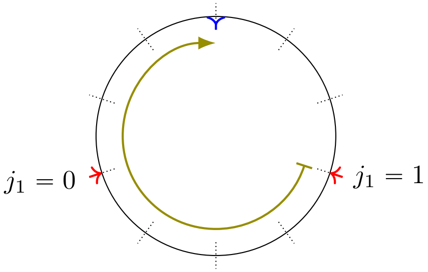


Parameters of the copier task

- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$

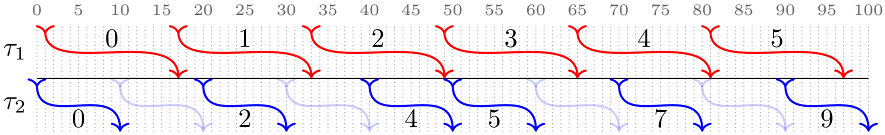


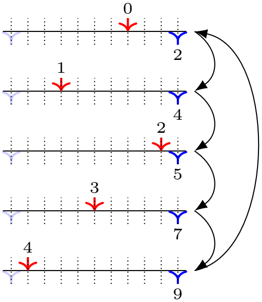


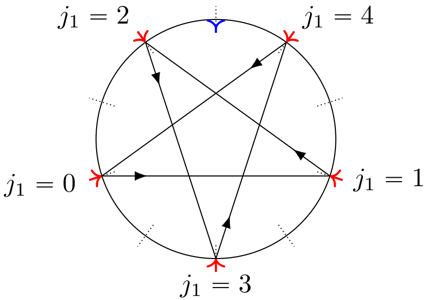


Parameters of the copier task

- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$

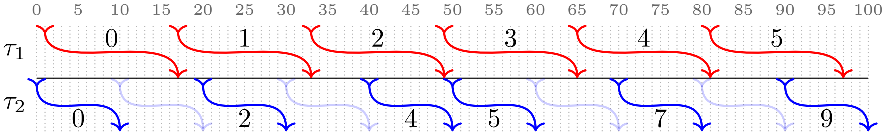






Parameters of the copier task

- Need to find the right phasing of the copier task
- Through enumeration or (better) via modular arithmetic. Example: $T_1 = 16$, $T_2 = 10$



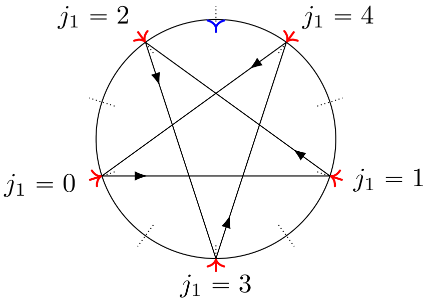
$\theta_2^{\mathsf{r}}(j_2) - \theta_1^{\mathsf{w}}(j_1) : 3, 7, 1, 5, 9, 3, 7, \ldots$

$\theta_2^{\mathsf{r}}(j_2) - \theta_1^{\mathsf{w}}(j_1) = 1 + G(1 - j_1 p_1 \mod p_2)$

with $G = \mathrm{GCD}(T_1, T_2)$

$T_1 = p_1 G$, $T_2 = p_2 G$

Example: $G = 2, p_1 = 8, p_2 = 5$



“Beautiful math... but just complicated: a short summary?”

- Easy case of two LET tasks with
  - no read offset: $\theta^{\mathsf{r}}_1 = \theta^{\mathsf{r}}_2 = 0$
  - deadline equal to period: $\theta^{\mathsf{w}}_1 = T_1$, $\theta^{\mathsf{w}}_2 = T_2$
- $G = \gcd(T_1, T_2)$ and p_1, p_2 such that $T_1 = p_1 G$, $T_2 = p_2 G$. Notice that $\gcd(p_1, p_2) = 1$
- $\lfloor x \rceil_m = x - \left\lfloor \frac{x}{m} \right\rfloor m$ [modulo operator, possibly over reals]

case $T_1 \geqslant T_2$

- $\mathbb{J}_{1\blacktriangleright 2} = \left\{ \left(j_1, \left\lceil \frac{(j_1+1)T_1}{T_2} \right\rceil \right) : j_1 \in \mathbb{J}_1 \right\} \equiv \mathbb{J}_1$
- $\theta^{\mathsf{r}}_2\left(\left\lceil \frac{(j_1+1)T_1}{T_2} \right\rceil\right) - \theta^{\mathsf{w}}_1(j_1) = \lfloor -(j_1+1)p_1 \rceil_{p_2} G$
- max $\tau_{1\blacktriangleright 2}$ latency $= T_1 + 2T_2 - G$
  - when $j_1 \equiv p_1^{-1} - 1 \mod p_2$
- min $\tau_{1\blacktriangleright 2}$ latency $= T_1 + T_2$
  - when $j_1 \equiv -1 \mod p_2$

case $T_2 \geqslant T_1$

- $\mathbb{J}_{1\blacktriangleright 2} = \left\{ \left(\left\lfloor \frac{j_2 T_2}{T_1} \right\rfloor - 1, j_2 \right) : j_2 \in \mathbb{J}_2 \right\} \equiv \mathbb{J}_2$
- $\theta^{\mathsf{r}}_2(j_2) - \theta^{\mathsf{w}}_1\left(\left\lfloor \frac{j_2 T_2}{T_1} \right\rfloor - 1\right) = \lfloor j_2 p_2 \rceil_{p_1} G$
- max $\tau_{1\blacktriangleright 2}$ latency $= 2T_1 + T_2 - G$
  - when $j_2 \equiv -p_2^{-1} \mod p_1$
- min $\tau_{1\blacktriangleright 2}$ latency $= T_1 + T_2$
  - when $j_2 \equiv 0 \mod p_1$

Conclusions

- Summary
  - proposed a common model for tasks and concatenation of tasks (chains)
  - recalled the LET task model
  - realized that chains of LET tasks
    - may have jitter
    - may have period larger than the largest among tasks
  - full characterization of a chain of two LET tasks via modular arithmetic
  - demonstrated that the introduction of a copier task makes a zero jitter chain
- The presented material is under submission to IEEE Transactions on Computers