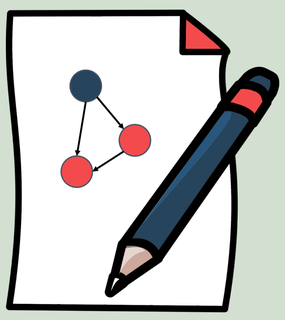
HGT: an open-source framework for simulating parallel real-time tasks

Ignacio Sañudo, Paolo Burgio and Marko Bertogna
University of Modena, Italy

The problem

Hard to properly validadate academic research because **even interested industrial players cannot share much information** about the system environment, nor the source code of their applications, due to NDA and IPR restrictions.

El 3 de mayo en Madrid - Francisco de Goya



The **HiPeRT Generator Tool (*HGT*)** is an open-source software framework that helps researchers creating synthetic yet realistic test cases, starting from behavioral description of application

Our approach

We follow a Model Driven Development (*MDD*) approach that, starting from text files providing a high level behavioral description of applications, generates code.

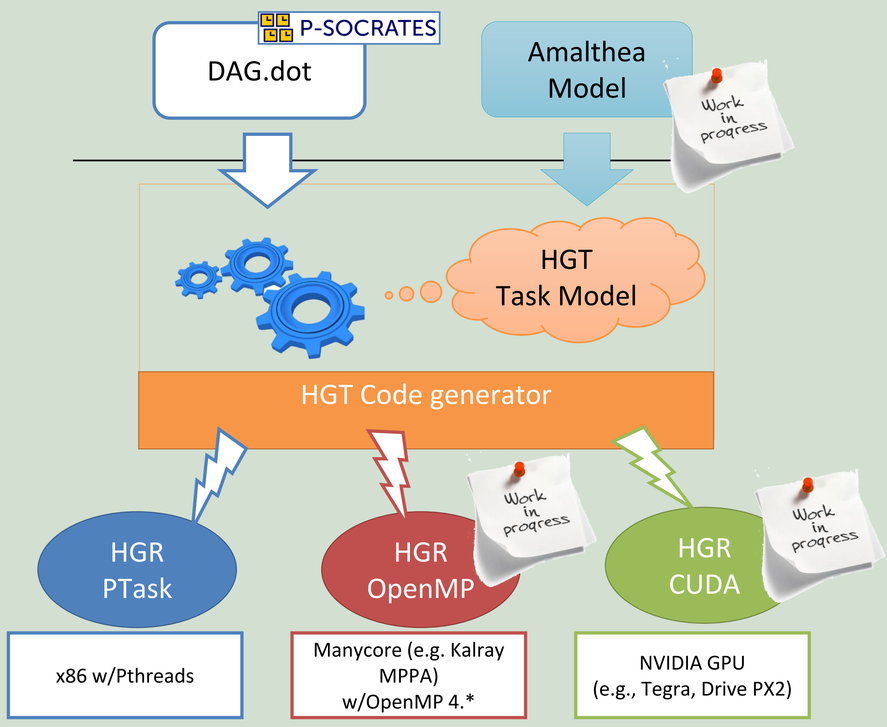
HGT receives as input a set of task dependencies, timing and memory constraints represented by a Directed Acyclic Graph (***DAG***). Then, the constraints are parsed into an internal model and then transformed into ANSI C code, that may be executed in different target platforms

We validate the correctness and accuracy of our tool by emulating the behavior of a real-time application specified using the UpScale DAG representation, produced in the P-SOCRATES FP7 project, and running on a x86-based system.

Alpha-version on GitHub

https://github.com/nachoSO/hipert.hg

Structure

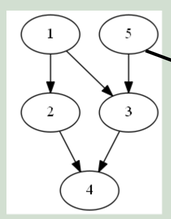


Frontend

```
digraph G2 {
    0 [period=2000, priority=60, deadline=2000, map=1]
    0 [label="tdg_id=0", style=invis]
    0 [label="maxI=0", style=invis]
    0 [label="maxT=14", style=invis]
    1 [MIET="10.48", MEET="10.65", MAET="10.63", MEM="100", UNIT="B"]
    2 [MIET="10.21", MEET="10.91", MAET="10.34", MEM="100", UNIT="B"]
    3 [MIET="10.42", MEET="10.82", MAET="10.66", MEM="100", UNIT="B"]
    4 [MIET="10.38", MEET="10.12", MAET="10.89", MEM="100", UNIT="B"]
    5 [MIET="10.84", MEET="10.86", MAET="10.21", MEM="100", UNIT="B"]
    1 -> 2
    1 -> 3
    2 -> 4
    3 -> 4
    5 -> 4
}
```

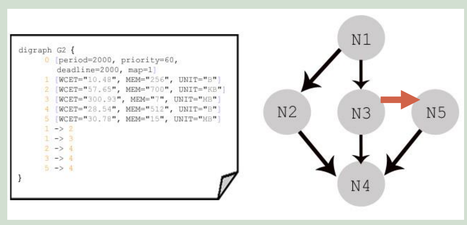
P-SOCRATES

Core



- Worst case execution time
- Worst case memory access time
- Period. Deadline…

Backend



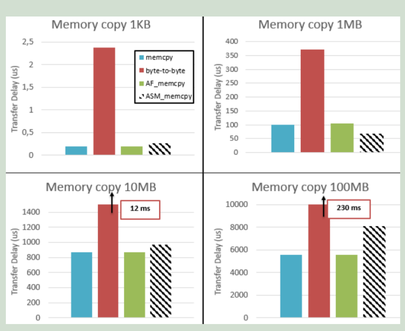

```
void *G1_1(){
    //printf("Hi im the node G1_1\n");  //1 [miet="10.381", meet="10.381"

    pthread_mutex_trylock(&lock_start1);

    volatile char *ptr_dst = (char *)malloc(G1_data[0]->data_size);
    hgr_PREM_compute_node(G1_data[0],ptr_dst); //Mem phase

    hgr_release_dependency(&G1_1_2);
    hgr_release_dependency(&G1_1_3);

    var_condition1++;
    free(ptr_dst);
    hgr_exit();

    return 0;
}
```

Simulation

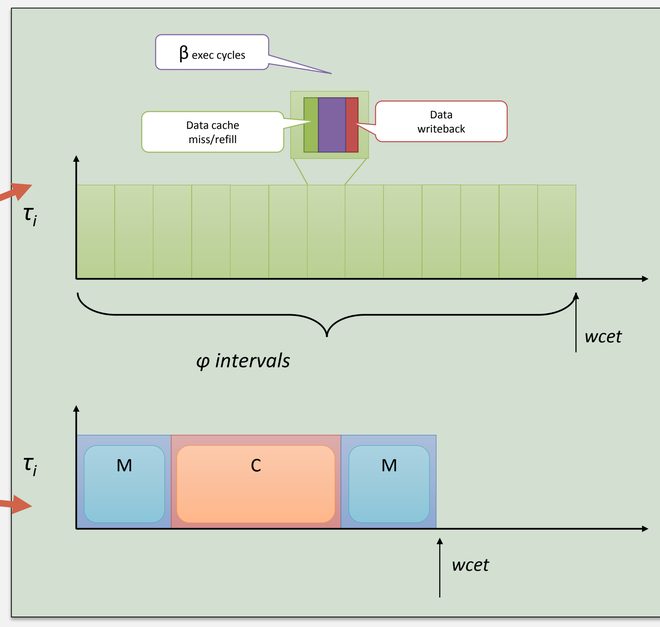
The front-end provides a DAG where **each node is characterized by a worst-case execution time, specified in time-units, and a memory access size**, specified in bytes.

- ✓ **Sparse model**, instead, we decided to evenly divide the memory accesses into multiple sequential blocks
- ✓ **PRedictable Execution Model (PREM).** *Memory* phases are implemented using a single *memcpy* of corresponding size, followed by an *Execution* phase lasting for the specified WCET (and an optional copyput phase).

We measured the timing accuracy in simulating application behavior



	PHI	512	1024	10240	102400	1024k	10240k	102400k
1 ms	CLOCK	6,10	10,27	89,37	889,25	9598,57	96614,45	966825,65
	ASM	2,70	3,97	11,39	32,60	78,19	1667,25	17589,36
10 ms	CLOCK	0,90	1,82	10,75	86,05	879,84	9572,47	96624,53
	ASM	3,71	2,15	2,47	9,47	13,73	76,47	1664,93
100 ms	CLOCK	0,11	0,23	1,22	8,61	83,31	866,89	9569,64
	ASM	0,62	0,48	0,79	1,07	8,19	11,60	73,29
500 ms	CLOCK	0,05	0,09	0,33	2,07	16,72	141,41	1833,61
	ASM	0,39	0,33	0,34	0,48	1,87	15,41	17,22
1 sec	CLOCK	0,03	0,05	0,19	1,06	8,41	82,54	866,99
	ASM	0,40	0,39	0,22	0,34	0,95	8,05	10,97
2 sec	CLOCK	0,00	0,01	0,07	0,58	4,31	46,79	383,48
	ASM	0,41	0,20	0,23	0,28	0,54	4,21	38,18



The number of blocks for a node is computed as

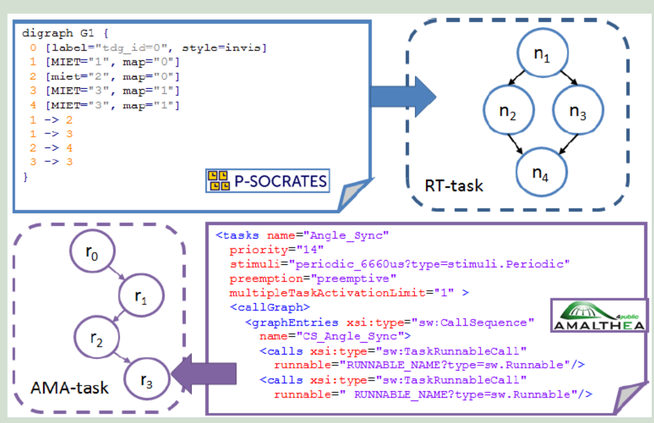
$$\Phi_{i,j} = \frac{M_{i,j}}{granularity}$$

Similarly, the worst-case execution time of the considered node is accordingly distributed among the blocks, so that each block has an execution time of $\beta_{i,j}$, where

$$\beta_{i,j} = C_{i,j}/\Phi_{i,j}$$

Next steps

- ✓ Extend the front-end to be compatible with Amalthea
- ✓ Port the back-end on top of the OpenMP supported by Kalray MPPA manycore and CUDA for embedded GPU-based platforms (e.g., NVDIA Tegras)



This Project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement: 688860.