Course assignment: Automated solving of task allocation and scheduling problems with arithmetic constraint solving

2007-05-29 Martin Fränzle, Andreas Eggers

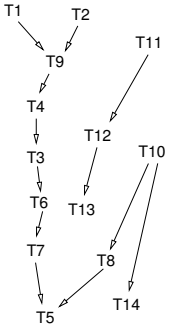
1 Overview

The goal of this course assignment is to build a tool that reads in a rather user-friendly input language for task allocation and scheduling problems and converts them to arithmetic SAT problems. These formulae can then be solved by HySAT in order to retrieve a schedule or to find out that no such schedule exists.

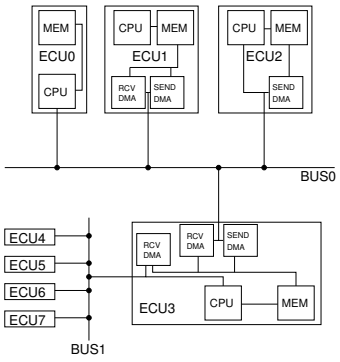
2 System description

2.1 Tasks

Tasks are executed only once. Each task has a start time at which it becomes ready for execution and a hard deadline after which the result of its execution becomes worthless, i.e. the task *must* finish before the deadline is exceeded. Tasks can depend on each other such that a task uses the results of its predecessors. A task may thus start if and only if it is ready (i.e. after its start time) and all its inputs are available in the electronic control unit (ECU) on which it is to be executed. This may necessitate the transmission of results from one ECU to another. Depending on which ECU a task is executed, its worst case execution time (WCET) may vary, e.g. because an ECU offers floating-point arithmetic while another one offers only an emulation of it. Tasks can be restricted to specific ECUs, e.g. because these are connected to sensors or actuators.



2.2 Hardware



Systems consist of ECUs on which the tasks can be executed and busses that can connect two or more ECUs with each other. An ECU may be connected to an arbitrary number of busses. Each of the ECUs is single-threaded, i.e. can execute no more than one task at a time. The results of a task that is executed on an ECU can be stored locally in the ECU or have to be relocated if the task that depends on the results runs on a different ECU. This transmission is performed on a bus that connects the source and target CPUs directly. The duration of this communication depends on the speed of the bus and the amount of data that has to be transferred. During a transmission the bus is blocked by the transmission task just like an ECU is blocked by a normal task when executing it. Although ECUs are single threaded, they can have bus interfaces that allow backgrounding of communication processes. If e.g. an ECU has a dedicated bus gateway listening on one of its busses for incoming messages, the ECU itself can execute a task while receiving a message from that bus at the same time. ECUs that are neither sender nor receiver in a transmission are not influenced by it, except for the fact that they cannot use the bus during this time. No broadcasting and multicasting transmissions are considered, neither are transmissions that use an ECU as a gateway to relay messages from one bus to another.

2.3 Scheduling

Tasks are scheduled without preemption, i.e. if a task has started to run, the ECU is blocked during the WCET of that task on that ECU. The schedule and the overall deployment of tasks is to be determined a priory by the tool that results from this assignment.

3 Input language

3.1 Syntax

```
system           ::=  busses ecus tasks communications
busses           ::=  "%BUSSES" {bus_declaration}*
ecus             ::=  "%ECUS"   {ecu_declaration}+
task             ::=  "%TASKS"  {task_declaration}+
communications   ::=  "%COM"    {com_declaration}*
bus_declaration  ::=  "bus" bus_id "{" "delay" "=" number ";" "}" ";"
ecu_declaration  ::=  "ecu" ecu_id "{"
                             { bus_id "{"
                                "receive" "=" ["foreground" | "background"] ";"
                                "send"    "=" ["foreground" | "background"] ";"
                               "}" ";"
                             }*
                      "}" ";"
task_declaration ::=  "task" task_id "{"
                             "starttime" "=" number ";"
                             "wcet" "{" { ecu_id "=" number ";" }+ "}" ";"
                             "deadline" "=" number ";"
                       "}" ";"
com_declaration  ::=  "com" "(" task_id "->" task_id ")" "=" number ";"
task_id          ::=  "t_" {[a-zA-Z0-9_]}+
bus_id           ::=  "b_" {[a-zA-Z0-9_]}+
ecu_id           ::=  "e_" {[a-zA-Z0-9_]}+
number           ::=  {[0-9]}+
```

In addition to that `"--"` introduces a comment which continues up to the end of the line.

3.2 Example

The following input file describes a system with two busses, three ECUs, and four tasks. *Syntax errors – if any – are unintentional. The syntax given above should be used as a base for the parser.*

```
1  %BUSSES
2  bus b_1 {
3      -- time per data unit
4      delay = 1;
5  };
6  bus b_2 {
7      -- slower than b_1
8      delay = 2;
9  };
10
11 %ECUS
12 ecu e_1 {
13     b_1 {
14         receive = background;
15         send = foreground;
16     };
17     b_2 {
18         receive = background;
19         send = background;
20     };
21
22 };
23 ecu e_2 {
24     b_1 {
25         receive = foreground;
26         send = foreground;
27     };
28     b_2 {
```

```
29          receive = background;
30          send = background;
31      };
32  };
33  ecu e_3 {
34      b_1 {
35          receive = background;
36          send = background;
37      };
38  };
39
40  -- e_1 -> e_2: * over b_1:
41  --       fully synchronized
42  --       communication, i.e. both
43  --       ECUs need to be active
44  --       during communication.
45  --     * over b_2: both can
46  --       execute other tasks
47  --       during transmission
48  -- e_1 -> e_3: e_1 needs to be
49  --       active, e_3 can do other
50  --       stuff.
51  -- e_3 -> e_2: e_3 can do other
52  --       stuff but e_2 must receive
53  --       actively.
54
55  %TASKS
56  task t_1 {
57      starttime = 0;
58      wcet {
59          -- runs only on e1!
60          e_1 = 100;
61      };
62      deadline = 500;
63  };
64  task t_2 {
65      starttime = 0;
66      wcet {
67          e_1 = 240;
68          e_2 = 70;  -- much faster
69          e_3 = 350; -- than here
70      };
71      deadline = 390;
72  };
73  task t_3 {
74      starttime = 100; -- ready @100
75      wcet {
76          -- runs only on e1 and e3
77          e_1 = 1030;
78          e_3 = 350;
79      };
80      deadline = 3000;
81  };
82  task t_4 {
83      starttime = 0;
84      wcet {
85          e_1 = 330;
86          e_2 = 120;
87          e_3 = 900;
88      };
89      deadline = 1500;
90  };
91
92  %COM
93  -- Dependencies between the tasks:
94  -- E.g. com(a -> b) = c;
95  -- a must send data to b, which takes
96  -- c data units multiplied with the
97  -- delay per data unit which is given
98  -- in the bus declaration of the bus
99  -- that is used. Only when this
100 -- communication has been finished
101 -- successfully, task b can start.
```

```
102
103 -- The communication may not start
104 -- before task a has been completed.
105 -- If a and b are allocated on the
106 -- same processor, c becomes zero.
107
108 -- If this communication is a
109 -- foreground process on a cpu, this
110 -- means that the cpu cannot execute
111 -- anything else when performing
112 -- the communication task.
113
114 com(t_1 -> t_3) = 90;
115 com(t_1 -> t_2) = 100;
116 com(t_3 -> t_4) = 20;
```

3.3 Semantics

From the descriptions in section 2 and the comments in the preceding example, the semantics of the input language should hopefully be quite clear.

The first section declares the names of the busses and their speed by assigning a delay value. Later this delay is multiplied with the number of data units given in the `"%COM"` section, e.g. `com(t_1 -> t_3)` uses $90 \cdot 1$ time units using bus `b_1`. If however `t_1` and `t_3` are executed on the same ECU, no bus communication is necessary and thus the delay is zero. For this example this would only be possible on ECU `e_1` because this is the only ECU that both tasks can run on. It is however probably a bad choice because `t_3` has a very high WCET on that ECU. Luckily the solver has to make these considerations later...

The second section introduces the ECUs and their bus connections. If an ECU is connected to a bus, both, the `receive` and the `send` specification must be given. If an ECU has some extra logic that allows to run bus communications in the background without needing the full attention of the ECU, this is described by using the `background` keyword. If a CPU cannot execute a task other than e.g. receiving data from the bus `b_1`, in the `b_1` section the line `receive = foreground` would be given. The ECU is only connected to the busses for which such a definition is given.

The third section describes the tasks that have to be scheduled. Each task has a `starttime` (the task is not ready before the value supplied in this field), at least one WCET, and a deadline. In the `wcet` section all worst case execution times are given, i.e. if an ECU does not occur there, the task cannot be allocated to run on that ECU.

The last section describes the communication dependencies between tasks. Communication is always performed after a task's execution has finished. In case the results are not needed on a different ECU, this communication takes no time, otherwise it is calculated as describes above. All dependencies have to be specified in this section. The task `t_1` does e.g. not depend on any other task, thus it never occurs on the right hand side of the `->` operator.

4 Static scheduling

After parsing a system that is encoded in the way described above, your tool will have to build a constraint system. This constraint system shall be satisfiable if and only if a schedule exists that violates none of the properties described above. Furthermore, the satisfying valuation itself shall include an encoding of a feasible schedule and allocation mapping for the system. This means that it must be possible to read from the encoding for all tasks on which ECU they start at which time and when they stop. Ideally the tool would parse the output of the solver and generate a more readable description that contains this data.

5 Assignment

It is expected that you write a tool which

- parses the input format as described in this document,
- has some suitable error handling in case of syntax errors
- generates a constraint system which enforces the given properties and causes a satisfying solution to be a feasible schedule for the system that does violate neither deadlines nor other properties mentioned above, and
- writes the resulting formula encoded for HySAT to a file.

Please design a few (at least two) benchmarks to demonstrate the capabilities of your tool. It is absolutely okay to invent random architectures or tasks, but it would of course also be nice to try building a "realistic" example like an airbag system with e.g. deceleration sensors sending their information to some control units that invoke tasks to check whether the airbag may be inflated e.g. by checking the current speed and finally call the task that actually causes the explosion to begin.

If you encounter difficulties with the complexity of this task, you may decide to drop some of the requirements, e.g. opt for having always exactly one bus in the system. These simplifications – depending on the amount of work you cut off by introducing them – may reduce the grade you earn for the course work. It is expected that you describe such decisions in your final report explicitly and clearly. We encourage you however to try to finish the task in its entirety.

If you find any parts of the descriptions unclear or some behaviour to be unspecified, feel free to make suitable decisions that fit in with the overall idea of this assignment. Please explain such decisions in the final report.

In case of questions or comments, especially if you want to back up your decisions, feel free to contact us.

6 Report Requirements

Your work on this assignment must be documented by a report. The report should comprise the following:

- A short introduction.
- A detailed description of your approach including explanations on how the properties described in this document are mapped to constraints in the resulting formula.
- Clear statements if you opted to simplify certain requirements.
- The example system descriptions and derived variants that demonstrate your tool's abilities. Along with these: the generated HySAT input files and the resulting schedules or unschedulability results, respectively.
- A conclusion in which you should summarize your findings obtained on this project.

Report Form

The report must be handed in *on paper*. In case this is impractical, you should ask Michael for permission for an electronic only submission. Deadlines apply in the same way.

Further, the various input files, output files, the tool along with its well-documented sources, etc. must be packed into a single *zip* or *tar.gz* file for each report and sent to MF (`Fraenzle@informatik.uni-oldenburg.de`) by email. After unpacking, the various files must be clearly identifiable.

The report must have a front page identifying the course, the assignment and the participating students.

The report on the assignment is expected to be around 10 pages and should not exceed 15 pages. To this you may add appendices.

General

The report for this assignment must be handed in at the *IMM reception, building 321* no later than **Friday, July 11, 2008 at 12.00**. This is a *hard deadline* — no reports handed in later will be considered. Also the files must be sent before the deadline.

Please note:

- The assessment will be based upon the presentation of your work in the reports.
- The reports *must be signed* by all participants. Unless you state otherwise, the signatures are understood to confirm that all participants agree to have contributed equally to the project.
- Any collaboration with other groups on smaller parts of the assignments must be declared and clearly identified. Collaboration on major parts is not acceptable.

Also note that clarifications, FAQs, and practical details may be put on the *course project page* found via the course home page. You should consult this if you encounter problems.