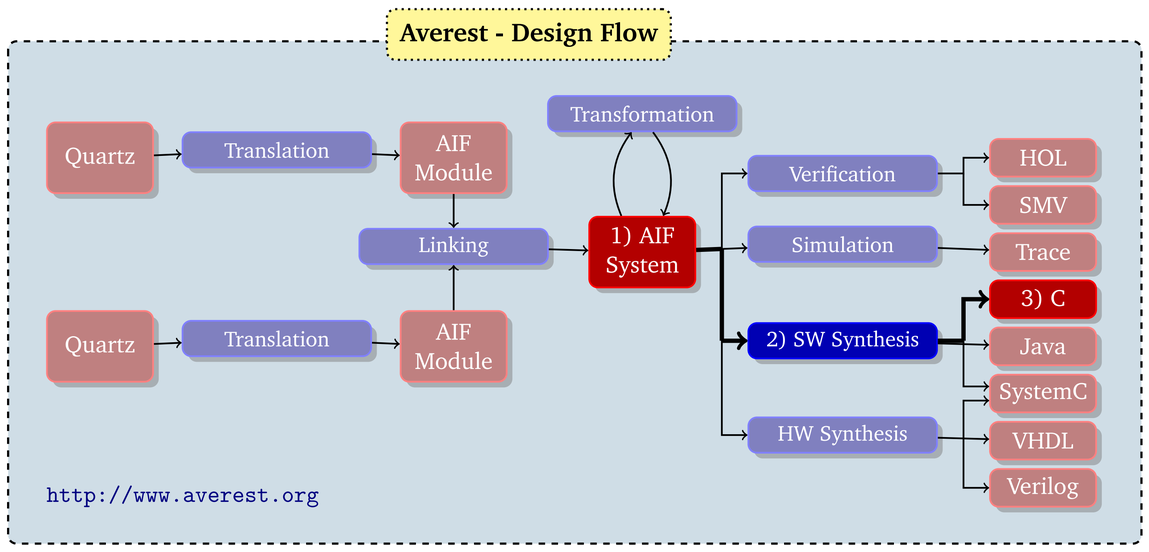
Multithreaded Code from Synchronous Programs: Extracting Independent Threads for OpenMP

Daniel Baudisch, Jens Brandt, Klaus Schneider



1) AIF System

1.1) AIF System

Interface:
Inputs: $start$
Output: z_1, z_2
Locals: x_1, x_2, y_1, y_2

Guarded Actions:

$$
\begin{aligned}
start &\Rightarrow l_1 = \text{true} \\
l_1 &\Rightarrow \text{next}(l_1) = \text{true} \\
l_1 &\Rightarrow x_1 = \text{true} \\
l_1 &\Rightarrow x_2 = \text{true} \\
l_1 \wedge x_1 &\Rightarrow y_1 = \text{true} \\
l_1 \wedge x_2 &\Rightarrow y_2 = \text{true} \\
l_1 \wedge \neg x_2 &\Rightarrow z_1 = \text{true} \\
l_1 \wedge y_1 \wedge x_2 &\Rightarrow z_1 = \text{true} \\
l_1 \wedge y_1 \wedge y_2 &\Rightarrow z_2 = \text{true}
\end{aligned}
$$

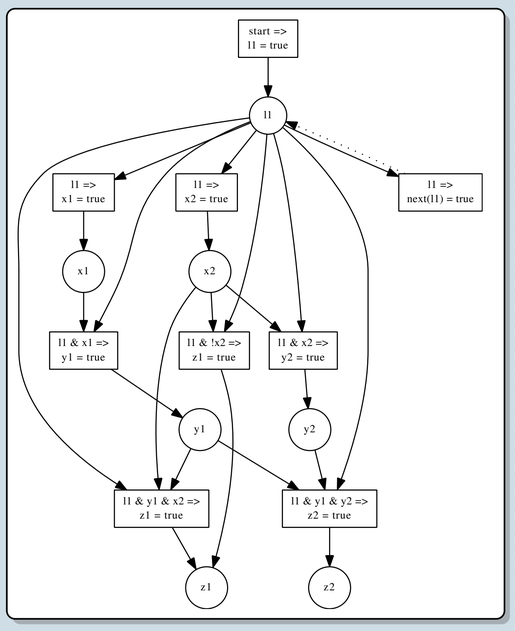
Behavior similar to a (clocked) sequential circuit. Each clock:

1. read inputs,
2. execute all actions,
3. write outputs.

Execution of actions in software:

- theoretically: concurrently, in zero time
- practically: find a sequence.

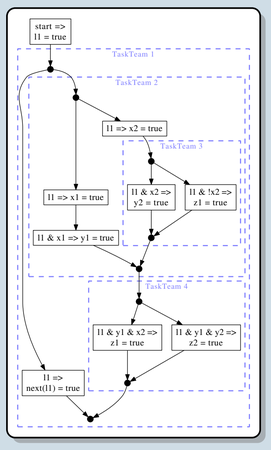
1.2) Action Dependency Graph (ADG)



- partial ordering over guarded actions
- required to sequentialize execution

2) Software Synthesis

2.2b) Interleaved Dependencies



OpenMP does not support interleaved dependencies. Can be handled by

- Duplicating actions (as shown in 2.2a)))
- Moving up the barrier (as shown in figure above)

2.1) Translation to Task Structure

Idea: Split dependency graph vertically and create group of actions such that each group can be executed as a single thread. The created groups must not communicate within a single macro step.

Definition of *Task Structure*:

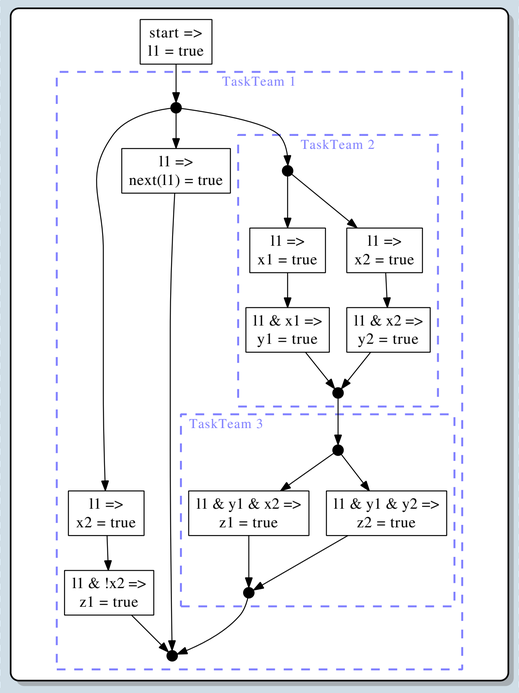
- TaskAction ::= $\gamma \Rightarrow A$ | TaskTeam
- TaskTeam ::= (Task TaskTeam) | Task
- Task ::= (TaskAction Task) | TaskAction
- TaskStructure ::= Task

Translation of ADG to Task Structure:

- A set of *open tasks* marks potential places, where actions can be appended.
- Determine all actions that can fire and schedule them to the open tasks. Synchronization may be required.

(See paper for a detailed description.)

2.2a) Task Structure



Each task team contains a list of tasks. Each task can be a sequence of actions or a task team. The begin (end) of a task team represents a fork (join) of the given set of tasks.

3) Multi-Threaded C

3) Generating C Code using OpenMP

```
bool start, l1, _next_l1, x1, x2, y1, y2, z1, z2;

while(true) {
  // read inputs
  read(start);
  // execute actions
  if(start) l1=true;
  #pragma omp parallel sections \
          reduction(|:x2) reduction(|:z1) {
    #pragma omp section
    { if(l1) _next_l1 = C(); }
    #pragma omp section
    { if(l1) x2 = B(); if(l1 && !x2) z1 = F(); }
    #pragma omp section {
      #pragma omp parallel sections {
        #pragma omp section
        { if(l1) x1 = A(); if(l1 && x1) y1 = D(); }
        #pragma omp section
        { if(l1) x2 = B(); if(l1 && x2) y2 = E(); }
      }
      #pragma omp parallel sections {
        #pragma omp section
        { if(l1 && y1 && x2) z1 = G(); }
        #pragma omp section
        { if(l1 && y1 && y2) z2 = H(); }
  } } }
  // write outputs
  write(z1);
  write(z2);
}
```

- Each task team represents a fork and is translated using the *sections* directive.
- Each task in a task team is translated as a thread using the *section* directive.
- Each task action can be translated as a conditional C statement

Alternative: Pipelining Approach

Basic Idea: Partion system into stages, i.e. split ADG "horizontally" instead of "vertically".
⇒ *"Multithreaded Code from Synchronous Programs: Generating Software Pipelines for OpenMP"* (MBMV'10)

TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

(es) Embedded Systems Group
Department of Computer Science - University of Kaiserslautern