Dispatching Domains for Multiprocessor Platforms and their Representation in Ada

Alan Burns and Andy Wellings

Motivation

- Multiprocessor and multicore platforms are becoming widespread
- For real-time systems the control of affinities is as important as the control of priorities
- Ada05 allows, but does not support, multiprocessor execution

Contents of Talk

- Background
- Scheduling – State of Art
- Basic support
- Current definition of support
- Example
- Conclusions

Background

- Identical, homogeneous, symmetric processors - SMPs, MPSoCs
- Assume basic OS support
- Ideas developed at IRTAW14 – thanks to all participants
- Proposal focused by ARG
- Presentation matches current proposal *not* paper!

Scheduling – State of Art

- Single processor scheduling is well understood
- Fixed Priority and EDF are mature technologies – and supported by Ada
- Multiprocessor state of art much less certain
- For example:
  - EDF is not an optimal scheme
  - Rate/Deadline monotonic priority ordering is not optimal

Multiprocessor Scheduling

- Two basic approaches:
  1. Partitioned – allocate all tasks to particular processors
  2. Global – allow tasks to migrate during execution.
- Partitioning is really ‘bin packing’ followed by single processor scheduling
- Global is potentially more effective, but increased overheads, probably does not scale and many open research questions

Basic Model

- Want to support partitioned and controlled global scheduling
- A fully flexible model is not justified at this time
- View all CPUs as a sequence 1..Number_Of_CPUs (not a set)
- Define Dispatching Domains to be slice of this sequence

Example

- 16 CPUs, denoted by 1..16
- 4 Dispatching Domains: 1..4, 5..8, 9..12 & 13..16, or
- 3 Domains: 1..1, 2..12, 13..16

Dispatching Domains

- All CPUs must be in exactly one Dispatching Domain
- All tasks (essentially) fixed to a single domain
- A task many be globally scheduling within its domain (can run on any CPU from within the domain), or
- A task can be fixed to just one CPU

Language Model

- Need to representing CPUs and Dispatching Domains
- Ideally these are done before any tasks execute
- But Ada's model of computation does not allow this *pre-execution* phase
- So a means of creating dispatching domains must be provided, and
- There needs to be a default initial domain for the environmental task.

Language Model

- In this talk:
  1. CPUs are just ordered integers – ie not sets
  2. Dispatching Domains are slices
  3. All dispatching domains have the same dispatching policies
  4. Interrupts may also have infinities
  5. Simpler than the model in the paper

Representing CPUs

```
package System.Multiprocessors is
  pragma Preelaborate;
  type CPU_Range is range 0 .. <implementation-defined>;
  Not_A_Specific_CPU : constant CPU_Range := 0;
  subtype CPU is CPU_Range range 1 .. CPU_Range'last;
  function Number_Of_CPUs return CPU;
  -- always returns the same value
end System.Multiprocessors;
```

Representing DDs

```
with Ada.Real_Time;
package System.Multiprocessors.Dispatching_Domains is
   pragma Preelaborate;
   Dispatching_Domain_Error : exception;
   type Dispatching_Domain is private;
   System_Dispatching_Domain : constant Dispatching_Domain;
   function Create(First,Last : CPU) return Dispatching_Domain;
   function Get_First_CPU(DD : Dispatching_Domain) return CPU;
   function Get_Last_CPU(DD : Dispatching_Domain) return  CPU;
```

Representing DDs

```
    function Get_Dispatching_Domain(T : Task_Id := Current_Task)
             return Dispatching_Domain;
    procedure Assign_Task(
        DD  : in out Dispatching_Domain; P : in CPU_Range;
        T : in Task_Id := Current_Task);
    procedure Set_CPU(P:CPU_Range; T : Task_Id := Current_Task);
    function Get_CPU(T:Task_Id := Current_Task) return CPU_Range;
    procedure Delay_Until_And_Set_CPU(
        Delay_Until_Time : in Ada.Real_Time.Time; P : in CPU_Range);
private
  -- not defined by the language
end System.Multiprocessors.Dispatching_Domains;
```

Pragmas

```
pragma CPU (expression);

pragma Dispatching_Domain (expression);
```

Ravenscar

1. No use of dispatching domains
2. All tasks statically partitioned
3. Task make use of pragma CPU
4. Each processor is advised to have its own set of ready queues

Interrupt Affinities

```
function Get_CPU(I: Interrupt_Id) return CPU_Range;

   -- The function Get_CPU returns the processor on which the
   -- handler for I is executed.

   -- If the handler can execute on more than one processor the
   -- value Not_A_Specific_CPU is returned.
```

Protected Objects

- Care must be taken with ceilings
- Real locks are needed
- Deadlocks etc are possible
- Execute PO code non-preemptively is one effective model

Example

- First using default dispatching domain
- A task executes on CPU 1 for this first 1.7ms, with a deadline of 5ms
- It then executed on CPU 2 with a deadline of 20ms
- Scheduling is via EDF
- Uses a Timer to switch CPUs

PO Spec.

```
protected Switcher is
  procedure Register(ID : Task_ID; E : Time_Span);
  procedure Handler(TM :in out Timer);
private
  Client : Task_ID;
  Extended_Deadline : Time_Span;
end Switcher;
```

Task Spec.

```
task Split is
  pragma Relative_Deadline(Milliseconds(5));
  pragma Priority (15); -- computed from deadline of task
  pragma CPU(1);
  pragma Dispatching_Domain(System_Dispatching_Domain);
end Split.
```

PO Body

```
protected body Switcher is
  procedure Register(ID : Task_ID; E : Time_Span) is
  begin
    Client := ID;
    Extended_Deadline := E;
  end Register;
```

PO Body

```
  procedure Handler(TM :in out Timer) is
    New_Deadline : Deadline;
  begin
    New_Deadline := Get_Deadline(Client);
    Set_Deadline(New_Deadline + Extended_Deadline,Client);
      -- extends deadline by fixed amount passed in as E
    Set_CPU(2,Client);
  end Handler;
end Switcher;
```

Task Body

```
task body Split is
  ID : Task_ID := Current_Task;
  Switch : Timer(ID'Access);
  Next : Time;
  First_Phase : Time_Span := Microseconds(1700);
  Period : Time_Span := Milliseconds(20); -- equal to full deadline
  First_Deadline : Time_Span := Milliseconds(5);
  Temp : Boolean;
begin
  Switcher.Register(ID,Period-First_Deadline);
  Next := Ada.Real_Time.Clock;
```

Task Body

```
  loop
    Switch.Set_Handler(First_Phase,Switcher.Handler'Access);

    -- code of application

    Next := Next + Period;
    Switch.Cancel_Handler(Temp); -- to cope with task
                                 -- completing early (ie < 1.7ms)
    Set_Deadline(Next+First_Deadline);
    Delay_Until_And_Set_CPU(Next,1);
  end loop
end Split;
```

Conclusions

- Historically, Ada has always taken a neutral position on multiprocessor implementations.
- On the one hand, it tries to define its semantics so that they are valid on a multiprocessor.
- On the other hand, it provides no direct support for allowing a task set to be partitioned.
- This talk has presented a set of facilities that is being considered for Ada2012.