Session Summary: Beyond Ada 2005

Chair: Jorge Real
Rapporteur: Stephen Michell

1. Session Goals :

To consider future directions in computing, and what changes would be required for Ada to effectively use new features.

Related Papers

1. *Beyond Ada2005: Allocating Tasks to Processors in SMP Systems*; A.J.Wellings and A. Burns.
2. *Suggestions for Stream Based Parallel Systems in Ada*; M. Ward and N.C. Audsley

2. Stream-Based Parallelism

Neil Audsley gave a look at a possible future in computation based on massive parallel architectures. The presentation began with current architectures, including

a) single CPU with L1, L2 cache and memory, and
b) double CPU with L1, L2 cache and memory, cache coherence at L2 memories

The presentation noted that these architectures are unscalable beyond a few (4-6) CPUs, because the replication of processors (CPU) on each chip, separation of L2 cache onto dedicated chips, distances and switching speeds of circuits when using multiple chips increase delays and power requirements.

An alternative view was presented that was called “System on a chip". Such a system has:

- Heterogeneous CPUs,
- Non uniform memory,
- Special devices.

It is expected that this will soon be followed by "Network on a chip", which consists of:

- Multiple systems-on-chip connected by networks
- No common notion of time
- Packet switched network

An example of such a system has been developed by the authors, that amounts to "Field Programmable System on a chip". Such a system is highly reprogrammable and can be reprogrammed in milliseconds, using an almost Ravenscar compliant system.

The authors identified some issues for the Ravenscar Tasking Profile.

Open Issue 1.1: Lack of a shared lock for Protected Entries

This was expressed as Ravenscar's restriction to a single entry per protected object or a single caller task per entry, but discussion highlighted that the problem is fundamental in Ada's specification of protected operations.

Protected functions in Ada permit a shared access to a protected object, but lack any synchronization. Protected entries provide synchronization, but lack the ability for a collective release of waiting tasks and each released task maintains a sequential lock. The need in highly parallel systems is to release collections of tasks that will read their dedicated data and not update protected data, hence behaving as a function once released.

The Ravenscar restrictions of a single entry and a single queue element per entry exacerbate this problem. It was agreed that this was a problem that requires a proposal to the Ada Rapporteur Group to solve these issues. Solutions could resemble a *pragma Simultaneous_Release*, or the addition of functions that block to protected objects.

The workshop agreed that this deserved further study.

2 Synchronous Multiprocessing

Andy Wellings presented a summary of the paper “Beyond Ada2005: Allocating Tasks to Processors in SMP Systems” and then lead a discussion on the topic through an interactive slide presentation. The paper, presentation and discussion assume a model of a shared memory multiprocessor environment and additions required for Ada 2005 to better support such an environment.

The author noted that Ada nominally addresses the multiprocessor environment, but assumes that there is an OS-level or implementation-level of support that simplifies the view of multiprocessing to make it seamless. Specifically, the paper notes that Ada is currently silent on how the runtime maps tasks

to specific processors, and proposes the use of pragmas to let an application guide such mappings.

The authors claim that better schedulability can be obtained by supporting static allocation of tasks to CPUs. They also claim that the approach is not scalable to multicore architectures that are Non Uniform Memory Access (NUMA).The authors also note that there is still no standardization of support for SMP in OS community, which affects any choices that Ada makes because Ada implementations may rely upon services that are not supplied, or may make choices that differ significantly from those eventually chosen by an OS. The challenge is to provide set of mechanisms that can be both expressive enough to support a wide range of application requirements, yet be implemented on a wide range of OS's.

Platform variability is a very significant issue for multiprocessor systems. An assumption is made that a concurrent program running on a SMP system will often not be the only program executing, that the hardware resources available to it will not be constant throughout the execution of a single execution, and that some processors may have capabilities or interfaces that are not available to other processors. For example,

a) An underlying operating system may dynamically change set of processors allocated to a program during execution and may or may not inform the executing program of such changes.
b) There may be hardware registers, interfaces to the external environment or interrupts available to some processors but not to all.

It is hoped that such changes would be done in a safe manner, but at present there is no language mechanism to manage these issues. The workshop decided that the minimal level of support that a program requires is to be able to determine how many processors are available to it. A proposed Ada service is shown in paper [1].

Another issue raised was that Ada 2005's support of task groups should interoperate with processor affinity. An extension of *Set_Affinity* to a task group would be useful. Another issue raised was that some aspects of memory maps may be processor-specific, and that ways to specify memory affinities should be considered. There were no specific set of calls proposed to provide such capability.

Throughout the presentation and discussion, there were a number of "Open Issues" that were raised and discussed.

Open Issue 2.1: Should the mapping of tasks be by-partition?

There was general agreement that this was the desired model.

Open Issue 2.2: Should there be Affinity Inheritance?

There was some discussion but no strong conclusion. It was generally agreed that such a model would work, in that nested tasks would start with the same processor affinity and could explicitly change that affinity with a call. It was noted that a pragma, such as *pragma priority* could be used for static affinity control.

Open Issue 2.3: Dispatching policies

There was agreement that dispatching policies must be partition-wide. A discussion was held about specific Ada dispatching policies and how they would be affected by the SMP model.

a) Dispatching policy *FIFO_Within_Priorities* should imply that a task can be migrated between its allocated processors whenever it is preempted.

b) Dispatching policy *Non_ Preemptive_ FIFO_ Within_Priorities* should mean that a task, once dispatched to a processor, will not be migrated from that processor while it is still executable (because it cannot be preempted).

c) The meaning of the dispatching policy *EDF_-Across_Priorities* is unclear if the tasks assigned to the priority range have a disjoint set of processors.

d) This raises the need for a new dispatching policy, *FIFO_Within_Priorities_Without_Migration*, where a preempted task cannot be migrated from the processor from which it was preempted while it is still runnable.

The discussion also considered the ramifications of affinity to scheduling policies. The ARM view of priorities states that high priority tasks ready for execution should always be executed in preference to lower priority tasks. Examples were given where a high priority task executing on a single processor (say HI with affinity {A}) could preempt a medium priority task (say MED on A with affinity {A, B}).

Open Issue 2.4: Interrupt handlers, Protected Objects & Tasks.

Ada's nominal mapping of interrupts is to protected objects, but tasks also often initiate and complete interrupt-level operations. If interrupts are processor-specific, there must be a way to map protected objects and tasks to the processor. An alternative procedure *Set_Handler* was proposed that would include the affinity mapping, but it was noted that task-processor affinity could also be a requirement. A

further complication would result if a single task called 2 protected objects that had different affinities.

The workshop decided that this was an area of interest and for further study.

Open Issue 2.5: Consistent notion of time.

Timers and relative delay were discussed and considered to be consistent. Absolute notions of time could be a problem, but should be satisfactory within a single partition. CPU time, however, could be problematic as processors may not all have the same clock speeds, and reduction or increase on the processor set could hinder calculations that optimize CPU-time.

It was agreed that at a minimum should be standardized for symmetric multiprocessing with static processor allocations.

Open Issue 2.6: Is it important how an OS manages SMP's?

The consensus was that Ada programs sit above OS implementations and cannot rely upon specifics of the OS-to-processor decisions.

Open Issue 2.7: Mapping Tasks to Processors:

The next discussion considered the mapping of tasks to processors. There was a general consensus that the mapping should be task-based, as opposed to partition-based. There was also sentiment that such a mapping should include mapping of data-specific regions, cache description and interrupts to processors. The following mapping choices for tasks-processors were enumerated,

1) Task → Processor
2) Task → {Processor}
3) {Task} → Processor
4) {Task} → {Processor}
5) {Task} → {Processor} + return to same processor.

It was agreed that the mapping proposal enumerated above is a reasonable beginning, but that pragmas should be included for the static mappings and memory mappings should be considered.

The workshop noted that affinity and preemption can lead to cascading preemptions. A case in point,

- HI on A, MED with affinity {A, B},
  LOW on any.
- HI preempts, but can't preempt LOW because LOW is on a processor for which HI has no affinity,
- HI therefore preempts MED which must then preempt LOW.

Other scenarios can be constructed where priority inversion occurs, i.e. HI preempts MED but MED cannot preempt LOW because MED has no affinity for the processor executing LOW.

Round-robin scheduling was discussed, and it was concluded that as long as all tasks participating in the round-robin at the same priority level had identical processor affinities, placing a task that has just finished its quantum at the end of the queue for all processors in the affinity set would suffice.

EDF was thought to be generally ok, but will cause preemption cascading. Further research is required.

Open Issue 2.8: What happens if OS removes a processor?

This is a serious issue if the processor causes significant perturbations in the affinity set of some tasks, such as giving a task a null affinity set. The call-back notification discussed earlier may suffice, as long as there was prior notification of the removal so that tasks could synchronously change their affinity sets.

Open Issue 2.9: Asynchronous Task Control and Affinity

There was a proposal to be able to add to *Aynchronous_Task_Control* the ability to change affinity. This proposal received insufficient support.

Open Issue 2.10: Protected Objects and Processor Affinities.

There are some significant issues in giving protected objects affinities. The requirement is clear since processor-specific mappings such as interrupts and registers may be utilized with no task thread, or may be called by a task without affinity for the processor in question. This is most likely if the implementation had proxy execution of protected entries, and a task with the wrong processor affinity tried to execute a protected entry on behalf of another task. There is, however, no current concept of *Protected_Object_ID* similar to *Task_ID* to build such a mapping.

It was noted that the existing Ada pragma *Attach_Handler* requires extension to include processor information where applicable. Similarly, a pragma to provide affinity could provide static affinities for protected objects.

It was decided that this topic needed further research.

3. Conclusions

As the session wrapped up, it was decided to continue developing proposals for the next workshop, and for the Ada Rapporteur Group to consider as they are developed.