ISSN (Print) : 0974-6846
Indian Journal of Science and Technology, *Vol 9(11), DOI: 10.17485/ijst/2016/v9i11/87833,* March *2016* ISSN (Online) : 0974-5645

Is Ada a good candidate for multiprocessor systems?

Negin Mahani*

Zarand Higher Education Complex, Computer Engineering Department, Shahid Bahonar University, Kerman, Iran; Negin.Mahani @uk.ac.ir

Abstract

Background/Objectives: Choosing a proper programming language for optimizing performance of multiprocessor systems is a trite object in the recent data processing era. **Methods/Statistical Analysis:** This research investigates Ada concomitant features for multicore programming. We have done several experiments to back the step that Ada is one of the best candidates for multiprocessor systems and this language is childbearing to be improved for multicore prospectus. **Finding:** By analyzing the simulation result we can say that Ada new features for supporting multicore programming are useful and they have improved the simulation speed. Thus it's better to schedule our tasks and set task affinity by a determined scenario using Ada new libraries rather than giving it simply to the underlying operation system. **Conclusion/ Application:** We have done some simulation on different platforms with different models to evaluate the speed up gained using new library packages useful for multicore programming in Ada language.

Keywords: Ada Programming Language, Concurrent and Multicore Programming, Multicore and Multiprocessor Systems, Parallel

1. Introduction

Ada programming language with a long history of existence already proved by large amount of sensitive real time, embedded and concurrent applications and inseparably supports concurrent programming with tasks.

For example, Vision Systems Ltd. of Australia lower used Ada instead of C to reduce their bugs from 1.25% to 0.36%, also they improve their testing time up to 55%. For the first time Rational Software's 1995 study compared the Ada and C programs. Verdix, had performed 50 percent of code lines in C and 50 percent in Ada. The Ada code in comparison to C reduces the defects and the overall development cost by 50% and 35% respectively. This sign bestow to academia as well. Previous researches, reported that the efficiency of programmers in their project on real-time and embedded software application are more efficient by utilizing Ada. Indeed, no one accomplished the coding remarkable, when using C. In other words, Three-quarters of the rank finished it by Ada, even proposed a partial part of the code (less than 20 percent). Also, it can be founded that with improving the skills and increasing the experience, the efficiency of Ada would be increased rapidly.

Programmer productiveness is always an interest, so tools that force it easier to compose excellent programs are efficient. The productivity and efficiency of coding are affected by the skillful of the language in huge programming solutions. Ada is to comprehend vie to various concurrent languages/frameworks, however, possession inherent support for task should spouse it more instinctive for various programmers to inscribe parallel programming in which increase the performance rate and rapids the program development.

Multiprocessor platforms are suitable model for more large embedded real-time development programs. Ada assigns its tasks to be accomplished on such systems, but in later versions, it does not clear support them clearly. The modern revise of Ada programming language know supply convenience for localize and programming duties on SMP system. The concept of a CPU and

* *Author for correspondence*

Dispatching-Domain are entered. These properties and specifications have been summarized in this paper till recently.

The original need for assisting the execution of Ada duties on multiple systematic processors symmetric multiprocessors is to control the mapping of duties to system CPU. In order to specify tasks to processors, two following methods are presented:

- Fully subdivided – each duty is specified to a single system CPU on which all its tasks should be ran.
- Universal – all duties are ran on all system CPU, tasks can migrate during performance.

Ada supports both schemes. For global scheduling, tasks are limited to a subset of the valid processors; and for partitioned scheduling;

The program can clearly turn a task's affinity and so, it causes to be moved at run-time1–4.

Dispatching domain shows a block of systems CPU with universal programming happens. A duty is called to be related to a dispatching zone and it can be bound to an especial processor. Most of the concentration with multiple core systems has been on the relating and dedicating of tasks to cores. It is essential for languages to be performed by static and dynamic models of this allocation simultaneously. However, as the number of cores grows and platforms become more inharmonious it would be essential to recognize and maintain parallel performance. There are distinct platforms such as one, limited and several cores. Most languages were determined for single processors. A 'limited cores' are based on number fewer than the inherent number of tasks, which are existed in the system. In these structures, the advantages of parallel run can be identified clearly. Tasks are mapped statically or dramatically to its inside system. In the simulation procedure there are regularly more duties than cores and so adequate operation of the hardware is created. In many cases that there are more ones than duties, the parallelism should only be utilized if parallelism within duties is recognized or utilized5.

This study investigates Ada programming language concurrent features and does some simulation experiments with models on different platforms to show if Ada is a good choice for multicore programming in the modern era. Section 2 of this paper introduced Ada programming language, its technical features and GNAT compiler briefly. Section 3 focuses on related works to analyzing Ada for multicore programming. Section 4 introduces Ada concurrent features. Descriptions of multiprocessor systems are in Section 5. Next section talks about Ada 2012 and multiprocessor systems relationships. Section 7 contains our implemented Ada models. The simulation results are in the last section and the final section is the conclusion.

2. Ada 2012 Advantages besides the Barriers for using it

The advantages of Ada 2012 are classified in categories which are named below:

- Introduction of dynamic contracts (Assert pragma…).
- More flexible and powerful form of expressions (if, case, …).
- Structure and visibility control (in/in out parameters in functions…).
- Development to the normal or high quality library.
- Increasing the efficiency, tasking and real-time.

All of the variability's are in the Real-Time procedure. Novel parcels are used for the revision of duties and budgeting on CPUs and the monitoring of time consumed in interrupts. In addition, there are extra features for non-preemptive dispatching, duty barriers and suspension projects.

The reasons listed below for not using Ada with all of these advantages are largely cultural:

- It is covered by its defense background.
- It works in a smaller universe.
- It lacks cool.

Programmers prefer to use languages, due to their stylish, however more that they like to be part of a community with plenty of noise and novelty and Ada doesn't propose that propose in relation to the various platforms. At the other extreme, Ada provides a task built option into core languages, which needs to know more languages. Some of them mentioned that it is easier to apply parallel programming[6]. Table 1 has presented some significant technical features of Ada programming language, classified by their application field.

Table 1. Technical features of Ada programming language

Useful for software coding	Useful for software coding and high level hardware description	Useful for hardware coding and in common with hard ware description languages
Structured programming	Pragmas such as Assert, …	Packages
Strong typing	Compile isolation	Generic components
Encapsulation	Up/down development	Task
Exception handling	Abstraction	Verification capability
Portability	Reusability	Embedded systems programming
Modularity	Multiple inheritance using interfaces and the ability to make calls using prefixed notation	Specification and implementation isolation
Physical and logical problem isolation	More flexible access types with anonymous types, more control over null and constant, and downward closures via access to subprogram types	Real time processing, the Ravenscar profile, various new scheduling polices, timers and execution time budget control
Enhanced structure and visibility control by the introduction of limited with and private with clauses and by an extended form of return statement	Various extensions to the standard library such as the introduction of operations on vectors and matrices, further operations on times and dates, and operations on wide wide characters; and especially a comprehensive library for the manipulation of containers of various kinds	
Readability		
Overloading		

3. Related Works

Karl Nyberg applied Ada's language with multicore hardware in 2007. They compared the sequential and duty-based extension of both applications (word counting and calculating checksums). On the word count algorithm, with increasing the number of tasks, the elapsed time decreased. Since file I/O effects on performance, its improvement did not vary linearly. The control program used fewer I/O activities and so demonstrated a reduction in used time that increased more linearly as more duties were utilized. The tasking overhead limited total throughput to about 50%[6]. In this paper the features and facilities of Ada 2012 program are discussed. Besides, it's great to consider about what's variability's since 2007 so we have been motivated to do some experiments using the new facilities in the last version of Ada.

Other research has been conducted in the context of direct synthesis of Ada code to hardware where these parallel calls should be directly mapped in the FPGA. It was concluded from them that, now it was not considered to be needed. Some other recent research about system level hardware design and simulation with Ada is done and had some unique results. The support that Ada gives for codes operation and performance on multiple CPU system was the subject of various papers[7–14]. There are lots of references that shows doing research on parallel programming in Ada programming language since long time ago[15–17].

4. Ada 2012 useful Packages for Leveraging Multiprocessor Systems

The Ada packages designed for supporting multiprocessor domain allow system processors to be divided into a share set of non-overlapping 'sendoff zone'. One sendoff zone domain is called to be the platform sendoff zone; the environmental duty and any derived from that duty are dedicated to the 'platform' sendoff zone. New sendoff zone can be created by subprograms. Tasks should be dedicated and specified to just one sendoff zone or limited to more than one processor. System, Multiprocessor is illustrated in Figure 1. The necessary comments are put near related lines.

```
-- The first package just defines a parent unit for all microprocessor facilities, and give a range
for an integer representation of each CPU.

package System. Multiprocessors is

pragma Preelaborate (Multiprocessors);

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; -- the CPUs are actually numbered
from one

function Number_of_CPUs return CPU; -- the number of CPUs is a constant and reflects the
number of available processors at system start up

end System. Multiprocessors;
```

Figure 1. Package system. Multiprocessor specification.

The second package called System Multiprocessors. Dispatching Domains provides the support for Dispatching Domains as its name shows. The specification of this package is presented in Figure 2.

```
with Ada.Real_Time;

package System.Multiprocessors.Dispatching_Domains is

...

type Dispatching_Domain (<>) is limited private;

System_Dispatching_Domain : constant Dispatching_Domain; --initially all CPUs are in
System_Dispatching_Domain

function Create(First, Last : CPU) return Dispatching_Domain;

-- removes specified CPUs from System_Dispatching_Domain

function Get_First_CPU(Domain : Dispatching_Domain) return CPU;

function Get_Last_CPU(Domain : Dispatching_Domain) return CPU;

function Get_Dispatching_Domain(T : Task_Id:= Current_Task)

return Dispatching_Domain;

procedure Assign_Task(Domain : in out Dispatching_Domain;

CPU : in CPU_Range :=Not_A_Specific_CPU;

T : in Task_Id:= Current_Task);

procedure Set_CPU(CPU : in CPU_Range; T : in Task_Id:= Current_Task);

function Get_CPU(T : in Task_Id := Current_Task) return CPU_Range;

procedure Delay_Until_And_Set_CPU(Delay_Until_Time : in Ada.Real_Time.Time; CPU : in
CPU_Range);

private

... -- not specified by the language

end System.Multiprocessors.Dispatching_Domains;
```

Figure 2. Package system. Multiprocessors. Dispatching_Domains specification[7].

5. Multiprocessor Systems Programming

C and C++ do not support multitasking inherently and it cannot be added as a postscript. The languages such as Simula-67, PL-I and Algol-68, all contained specifications for controlling parallel performance on multiple CPU systems. Also, Ada is a significant and important language in in multiprocessing issues. The Ada-83 version of the program has a good-extended program of multithreading (named "tasking" in Ada) with high-stage readable constructs. Other versions of this language also improved this feature.

Java has some low-level and error-prone facilities for multithreading. Per Brinch-Hansen's, accused the designers of Java of launching out multiple years of projects in concurrency; however Java's facilities in that area are certainly disappointing[18]. There are useful researches on different aspects of multiprocessor systems which are more than this discussion[19–22].

6. Ada 2012 and Multiprocessor Systems: Most Important Features

In the recent years programming multicore systems is an interesting objective. As Robert Dewar notes in his EE Times commentary, "There's nothing new about multicore mania". Decades of experience have been accumulated in using Ada to deal with the problem of writing programs that run effectively on machines using more than one processor".

Some researchers are not considering and negligible this experience and searching for new methods to do multiple procedure systems for cultural reasons. Ada programmers will be utilizing Ada to construct high-extension multiprocessor CPU approaches. Some other features are discussed in[23–26].

6.1 Sendoff Zone in Ada 2012

Ada 2012 support duty dependence by providing Sendoff Zone[9,27,28]. This language expresses Sendoff Zone as static specification. So, creating changes involving migration of processors from one Sendoff Zone to another is not able to be used. Reconstruction Sendoff Zone when the underlying hardware change is not supported either. Supporting dynamic would be better than static one to address these situations. It should be considered that in this version of Ada 2012, the set of processors are fixed for its whole time, so for applying the variations and to achieve dynamic response, the commands should be sent to the language out of the language.

6.2 Preference, Deadline and Tendency

Preference, latest time and tendency can be varied dynamically (Set_ { Preference, Deadline, processor}). Ada 2012 provides the facility which makes it possible to vary deadline and tendency after a delay (Delay_Until_And_Set_ { latest time, processor}), however, it is not able to tune more than one of these features, which causes incorrect scheduling and so, the risk of error increases[29].

In the operation process, a duty is ran inherits the operation priority that is more possible to be activated (Ada 2012 RM 9.2) had at the time the operation was stated. While tryst, the duty passing the all call inherits the priority of the entry procedure. In the process of a protected operation on a constant object, a duty inherits the ceiling priority of the constant object (9.5, D.3, D.1).

Proposed Ada program, although provides sendoff zones, imposes a single programming policy in all zones. So, preference assignments in all CPU systems can be universally coherent. By following this method, succession of a preference is valid.

7. Ada Models

We have implemented a concurrent model that utilizes four basic operations. This program calculates the result of addition, subtraction, multiplication and division on four pair constant arrays of integer numbers. Each pair of the arrays applies to one specific type of arithmetic operation to eliminate data dependency and resource control problems in data sharing between concurrent tasks. Also to see the side effects of data dependency on task execution speed we have done the operations with a shared pair of arrays, again. We have implemented the source code in three different models regarding the number of the tasks involved. The first model consist of a simple task to do all arithmetic operations, the second model consist of two task to do half of the operations each task and the number of the tasks in the last model is equal to the number of operations (one basic operation each task). The following figures illustrate the Ada models described above explicitly (Figures 3, 4 and 5).

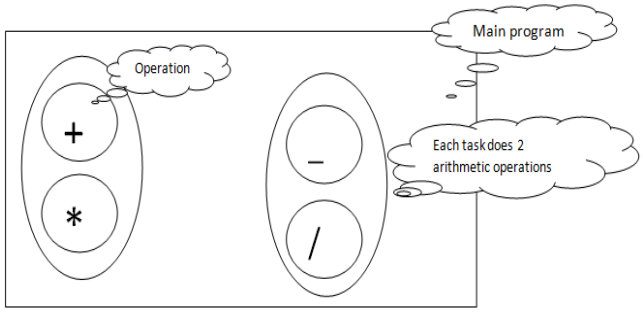

Figure 4. Ada Model #2 (containing 2 tasks).

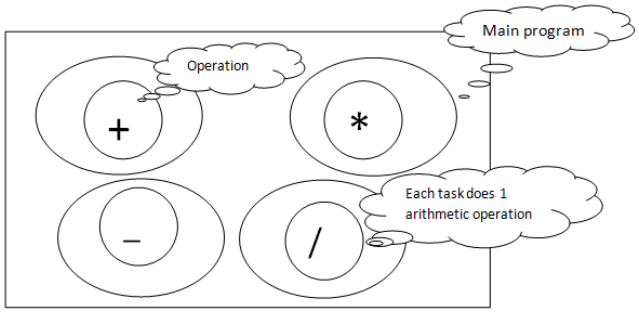

Figure 5. Ada Model #3 (containing 4 tasks).

8. Simulation Results

We have done some simulation on different platforms with different models to evaluate the speed up gained using new library packages useful for multicore programming in Ada language. We have done the simulation on the windows based platforms with different number of cores (1 core, two cores and 4 cores) using GNAT 2012 compiler on Windows. We have measured the execution time using Ada.Execution_Time package facilities (Tables 2 to 5).

Table 2. Execution time of different task number on different core number (whit out using multi-processor package facilities – without data dependency)

Task # and Core#	Execution time
1 & 1	9.87
2 & 2	7.47
4 & 4	5.62

Table 3. Execution time of different task number on different core number (using multi-processor package facilities – without data dependency)

Task # and Core#	Execution time
1 & 1	9.87
2 & 2	5.99
4 & 4	4.11

Table 4. Execution time of different task number on different core number (without using multi-processor package facilities – with data dependency)

Task # and Core#	Execution time
1 & 1	11.65
2 & 2	10.93
4 & 4	9.23

Table 5. Execution time of different task number on different core number (using multi-processor package facilities – with data dependency)

Task # and Core#	Execution time
1 & 1	11.65
2 & 2	10.16
4 & 4	9.01

9. Conclusion

A concurrent program consists of elements which effects on each other directly or by the shared references. Ada can model each of these components, which named a task. Straight relation among two duties can be reached by a rendezvous. Shared references are traced to a constant and fixed object (which provides mutual deprivation). Ada tasks are allocated to multiprocessor CPU system, to multiple fields on a single CPU system or to multi inside CPU system of a only one CPU, depending on the compiler and goal surroundings. It should be noted that it is not necessary to change with variation of environments.

It is obvious that it is required that writing program with the features of easily and efficiently leverage parallel hardware. For Ada programming language it equals with procedure that the writer can designate code which should be executed in parallel. This paper was about Ada multicore programming challenges with the new multiprocessor architectures. The simultaneous support for multi procedure CPU system in Ada 2012 is sendoff zones for task dependencies. It matches the support typically proposed by multi procedure CPU acting systems. In systems with non-SMP configuration, a lot of possibilities are existed for each programming codes which tries to provide the efficient production and predicable codes. In some cases intelligent compilers can remove these difficulties, however, it is similar to that some summarized delegation of the system will require to be existing at the program stage.

Ada 2012 gives good extra features for multi-processor codes and now GNAT provides a main share of it. A few pragmas like Volatile were improved to create the life of Ada applicants on multiple procedure CPU system to be convenient, by adding some cost to performance fine, and a little job on compiler vendor's system. Since the mentioned ability and facilities are not supported by action systems the area of modifications to new sendoff policies has not been paid more interesting.

By analyzing the simulation result we can say that Ada new features for supporting multicore programming are useful and they have improved the simulation speed. Thus it's better to schedule our tasks and set task affinity by a determined scenario using Ada new libraries rather than giving it simply to the underlying operation system. In overall, it can be mentioned that Ada is a proper and sufficient language for multicore chips, which its advantages and priority to the other programs were identified.

10. References

1. Ali H, Pinho M. A parallel programming model for Ada. SIGAda. 2011 Dec; 31(3):19–26.
2. Burns A. Programming languages for real-time applications executing on parallel hardware, Reliable Software Technologies - Ada-Europe. Springer Berlin Heidelberg. Lecture Notes in Computer Science; 2011; 6652:193–5.
3. Chun R, Lichota R, Perry B, Sabha N. Synthesis of parallel Ada code from a knowledge base of rules. Proceedings of the Third IEEE Symposium on Parallel and Distributed Processing; Dallas, TX. 1991 Dec 2-5. p. 600–7.
4. Jha R. Parallel Ada: Issues in programming and implementation. Proceedings of the Fourth International Workshop on Real-time Ada Issues, IRTAW '90. 1990; 10(9):126–32.
5. Burns A. Parallel Ada- A requirement for Ada 2020. Ada Letters. 2013 Aug; 33(2):9–13.
6. Nyberg K. Multi-Core + Multi-Tasking = Multi-Opportunity? SIGAda '07 Proceedings of the 2007 ACM International Conference on SIG Ada Annual International Conference. ACM SIGAda Ada Letters - SIGAda '07. New York, NY, USA. 2007 Dec; 27(3):79–82.
7. Burns A, Wellings AJ. Support for multiprocessor platforms. Ada Letters. 2013 Apr; 33(1):9–14.

8. Burns A, Wellings AJ. Multiprocessor systems: Session summary. Ada Letters. Proceedings of the 14th International Workshop on Real-Time Ada Issues (IRTAW 14). 2010 Apr; 30(1):147–51.
9. Lee S, Hong S. Analysis of time records on digital forensics. Indian Journal of Science and Technology. 2015 Apr; 8(S7):365–72.
10. Real J, Mitchell S. Beyond Ada 2005: Session report. Proceedings of IRTAW 13, Ada Letters. 2007; 27(2):124–6.
11. Ruiz J. Towards a Ravenscar extension for multiprocessor systems. Ada Letters. Proceedings of the 14th International Workshop on Real-Time Ada Issues (IRTAW 14). 2010 Apr; 30(1):86–90.
12. Ward M, Audsley NC. Suggestions for stream based parallel systems in Ada. Proceedings of IRTAW 13, Ada Letters. 2007 Aug; (2):82–7.
13. Wellings AJ, Burns A. Beyond Ada 2005: Allocating tasks to processors in SMP systems. Proceedings of IRTAW 13, Ada Letters. 2007 Aug; 27(2):75–81.
14. Wellings AJ, Malik AH, Audsley NC, Burns A. Ada and cc-NUMA architectures. What can be achieved with Ada 2005? Ada Letters – Proceedings of the 14th International Workshop on Real-Time Ada Issues (IRTAW 14). 2010 Apr; 30(1):125–34.
15. Hind M, Schonberg E. Efficient loop-level parallelism in Ada. Tri-Ada; 1991. p. 166–79.
16. Mayer HG, Jahnichen S. The data-parallel Ada run-time system, simulation and empirical results. Proceedings of Seventh International Parallel Processing Symposium; Newport, CA. 1993 Apr 13-16. p. 621–7.
17. Park EK, Anderson PB, Dardy HD. An Ada interface for massively parallel system. Fourteenth Annual International on Computer Software and Applications Conference. COMPSAC 90; Chicago, IL. 1990 Oct 31-Nov 2. p. 430–5.
18. Pandharpurkar NG, Ravi V. Design of BIST using self-checking circuits for multipliers. Indian Journal of Science and Technology. 2015 Aug; 8(19):1–7.
19. Andersson B, Bletsas K. Sporadic multiprocessor scheduling with few preemptions. Euromicro Conference on Real-Time Systems (ECRTS); Prague. 2008 Jul 2-4. p. 243–52.
20. Andersson B, Jonsson J. Fixed-priority preemptive multiprocessor scheduling: to partition or not to partition. Proceedings of the International Conference on Real-Time Computing Systems and Applications; Cheju Island. 2000. p. 337–46.
21. Burns A, Davis RI, Wang P, Zhang F. Partitioned EDF scheduling for multiprocessors using a C = D scheme. Proceedings of 18th International Conference on Real-Time and Network Systems (RTNS); 2010. p. 169–78.
22. Burns A, Wellings AJ. Dispatching domains for multiprocessor platforms and their representation in Ada. J. Real and T. Vardanega. Proceedings of Reliable Software Technologies - Ada-Europe 2010. 2010; 6106: p. 41–53.
23. Juan A, Stephen M. Session summary: Concurrency issues. Ada Letters. 2013 Jun; 33(1):150–6.
24. Barros A, Pinho LM. Revisiting transactions in Ada. 2013 Apr; 33(1):84–92.
25. Vardanega T, Harbour M, Pinho LM. Session Summary: Language and Distribution Issues. Ada Letters. Proceedings of the 14th International Real-Time Ada Workshop. 2010 Apr; 30(1):152–61.
26. Ruiz F. Going real-time with Ada 2012 and GNAT. Ada Letters. 2011. p. 1–6.
27. Taft T. Multicore programming in ParaSail. Ada-Europe. 16th International Conference on Reliable Software Technologies, Lecture Notes in Computer Science (LNCS); 2011. p. 196–200.
28. Afkar A, Mahmoodi-Kaleibar M, Paykani A. Geometry optimization of double wishbone suspension system via genetic algorithm for handling improvement. Journal of Vibro Engineering. 2012 Jun; 14(2):827–37.
29. Wellings AJ, Malik AH, Audsley NC, Burns A. Ada and cc-NUMA architectures. What can be achieved with Ada 2005? Ada Letters. Proceedings of the 14th International Workshop on Real-Time Ada Issues. 2010 Apr; 30(1):125–34.