Multi-Core + Multi-Tasking = Multi-Opportunity?

Karl Nyberg
Grebyn Corporation
P. O. Box 47
Sterling, VA 20167-0047
1-703-406-4161
karl@nyberg.net

ABSTRACT

In this paper we look at the opportunity provided by the introduction of multicore chips to leveraging Ada's multitasking capabilities, with an eye to expanding the influence of Ada in this environment. The Sun Microsystems Sun Fire T1000 running Solaris is used as a platform for investigation of Ada in a multicore environment. Some sample applications are described and evaluated. Guidance on developing multi-tasking applications is also given. Opportunities for the Ada community to leverage this hardware shift to expand Ada's scope of application are presented.

Categories and Subject Descriptors: C.1.4 [**Parallel Architectures**] Multi-Core Processors; D.1.3 [**Concurrent Programming**]: Parallel programming.

General Terms: Measurement, Performance, Design.

Keywords
Multi-Core, Tasking, Parallel Execution.

1. INTRODUCTION

For many years, the semiconductor industry has been increasing clock speed on delivered chips to provide greater performance. It seemed that the industry was about to run into physical limitations of the underlying semiconductor material being used to continue this trend when manufacturers began considering the creation of chips that contained multiple (albeit slower, to manage power consumption and heat dissipation [1, 2]) computing cores as a way to continue increasing overall performance.

This paper looks into exploiting the opportunity provided by a paradigm shift to multi-core computing by leveraging the use of Ada's built-in tasking mechanism and expanding the influence of Ada in this environment. The technical focus is specifically on the Sun Microsystems Sun Fire T1000, containing an 8 core Niagara 1 CPU with quad strands, effectively creating a system with 32 virtual CPUs. Opportunities are identified for the Ada community to leverage the shift to parallel programming to increase the extent of Ada's use as well as identification of impediments to such increase in the Ada community.

2. CHASING PERFORMANCE

2.1 Prior Approach – Increase Clock Speed

2.1.1 Commodity Hardware

The chip wars of the end of the twentieth century and the beginning of the twenty-first were all about speed. Some times there were tweaks (64 bit CPUS, additional on-die memory, etc.), but generally it was faster, faster, faster and who had bragging rights to the fastest chip.

2.1.2 Serial Languages - Single-Threaded Applications

Faster chips allowed single-threaded applications written in serial languages to continue to achieve "acceptable" performance by technology refreshes. This allowed industry to sweep problems with application performance under the rug by continually upgrading hardware as demands increased. This is not to say that there aren't multi-threaded applications (such as operating systems, database engines and other transactional systems), even in serial languages, just that such applications weren't necessarily multi-threaded applications at heart.

2.1.3 Commodity Programmers

By allowing clock speed increases to solve performance problems, industry has been able to utilize commodity programmers – those with an ability to think linearly and serially, rather than in a parallel manner. These programmers have come to depend on future clock speed improvements to increase performance and have hit a performance wall as such improvements have subsided.

2.2 New Approach – Increase Computations

2.2.1 Special Purpose (for a while) Hardware

Not everybody is going to want to pay the premium for a multicore chip, especially when their underlying application (be it a commodity operating system or other application) doesn't take sufficient advantage of the capability of the equipment to justify the expense. However, as time progresses and operating systems and applications evolve to take advantage of the technology, and, as production volumes and commoditization kicks in to reduce prices, the "popularity" of such chips will be more widespread.

2.2.2 Concurrent Languagesand Multi-Threaded Applications

For Ada, multi-tasking through the use of multiple parallel processors is not a new idea [3]. In fact, concurrent variants of other programming languages (C, Fortran, Pascal, Modula II) have also existed for a number of years. However, for some of these languages, the addition of concurrent features has only been optimized for special purposes hardware [5]. Even Java has a comparable level of feature support for concurrency to that of Ada [6], although it doesn't seem to be widely applied.

Oftentimes, parallelization is accomplished indirectly through the use of "add-on" capabilities and libraries, such as OpenMP [7].

2.2.3 Thread-Aware Programmers

Academia [8] is just beginning an NSF-funded curriculum study targeting parallel programming. Some parts of industry [9] thinks it will be five to ten years before a parallel programming model emerges and that the tools necessary for multicore programming is "still in the dark ages" [10].

3. SUN FIRE T1000

The Sun Fire T1000 is one of Sun Microsystems' "Cool Threads" line of systems. The machines are centered around their multi-core Niagara chip, designed to minimize power consumption requirements and maximize a new metric, coined "SWAP" (Space, Watts and Performance). The machine runs standard Solaris and has had a version of Linux ported to it, comes with a single CPU chip containing up to 8 cores, one floating point unit, four gigabit Ethernet connections, and a fully loaded 1U rack mount system with two disk drives consumes only 142 watts.

In order to introduce the new product to the marketplace, Sun ran an "Open Performance Contest" [11] wherein it provided evaluation systems for 60 day periods, and awarded systems based upon a set of evaluation criteria to individuals, companies, and universities based upon those evaluations. A system priced at $14,465 was awarded based upon our evaluation [12] developed with an Ada application.

4. SAMPLE APPLICATIONS

I had hoped to find more, but simple searching and sorting [13] and the game of Life [14] weren't highly motivating, so these simple replacements for UNIX utilities wd ("word count") and sum (checksum) were created. The tables below show the execution times (as reported, user time, system time and elapsed time), with performance relative to the single task implementation and the serial implementation (the identical Ada code for calculating results, implemented in a loop rather than via tasks).

It is instructive to note that the sum command, requiring a greater amount of computation than the wc command, was able to effectively have a greater utilization of the available CPU with increased tasking although both results show the overall limitations due to interleaved file system operations.

4.1 The UNIX "wc" Command

Tasks	User Time	System Time	Elapsed Time	Relative to 1 task	Relative to serial
1	1002.0	5.0	1007.0	100.00	30.78
2	1059.0	5.0	534.0	199.25	58.43
3	1069.0	6.0	359.0	299.44	86.91
4	1078.0	6.0	278.0	389.93	112.23
5	1077.0	6.0	219.0	494.52	142.47
10	1128.0	6.0	121.0	937.19	257.85
15	1185.0	6.0	85.0	1401.18	367.06
20	1291.0	7.0	73.0	1778.08	427.40
25	1395.0	7.0	64.0	2190.63	487.50
30	1530.0	8.0	56.0	2746.43	557.14
32	1576.0	8.0	56.0	2828.57	557.14

4.2 The UNIX "sum" Command

Tasks	User Time	System Time	Elapsed Time	Relative to 1 task	Relative to serial
1	130.0	5.0	135.0	100.00	100.00
2	130.0	6.0	68.0	200.00	198.53
3	130.0	6.0	46.0	295.65	293.48
4	130.0	6.0	34.0	400.00	397.06
5	130.0	5.0	27.0	500.00	500.00
10	135.0	5.0	14.0	1000.00	964.29
15	143.0	6.0	10.0	1490.00	1350.00
20	172.0	6.0	9.0	1977.78	1500.00
25	201.0	6.0	9.0	2300.00	1500.00
30	234.0	6.0	9.0	2666.67	1500.00
32	243.0	6.0	8.0	3112.50	1687.50

5. PRACTICAL CONSIDERATIONS

There are a number of practical considerations in development of multi-tasking applications. They probably apply just as much in a multi-core environment as anywhere. These have to do with implementing functionality, exception management (or the lack thereof), memory management and task scheduling.

5.1 Implementing Functionality

The applications described in this paper have been made parallel, resulting in multiple instances of what is effectively the same algorithm being applied in either a linear or recursive fashion. This homogeneity of tasks has allowed the desired functionality to be developed in a serial fashion (thus focusing on the application and not upon the tasking interactions) and then distributed across multiple processors. Development of an application with non-homogeneous tasks would of necessity have to be performed in a different manner.

5.2 Exception Management

One of the inevitable problems that occur during the development of a tasking-based application is the raising of an exception within one or more of the tasks. Failing to provide an exception handler for the subsidiary tasks causes these tasks to become non-responsive. Using a debugger, manual tracing and / or a "catch all" exception handler within each task will help to identify the location of the problem and resolve this situation.

5.3 Memory Management

With multiple tasks operating, it is necessary to control access to variables within the program. This could be considered a drawback to the serialized implementation followed by recasting in a parallel environment when the temporal scope of variables is not adequately addressed. We found it sufficient to move such variables into the scope of the task under execution. In other instances it may be necessary to utilize capabilities provided by protected types. Given our limited sort of "pattern" during this effort, it is difficult to speak with any more certainty on this issue.

5.4 Task Scheduling

Another situation under which tasks become non-responsive is due to an unfair scheduling in the operating system. If one task were to perform an I/O operation and thus be thrown out of a ready-to-run queue, other tasks that are performing computations could effectively "hog" the underlying CPU. In our applications, we never saw this behavior in part because we never ran with more subsidiary tasks (32) than we knew we had underlying processors as our goal was to maximize the use of the CPU for computationally intensive applications.

6. OPPORTUNITY

The Ada tasking model maps quite nicely (some would say "intuitive" [15]) for application execution in multi-core enviornments. The opportunity afforder by the introduction of multi-core systems is not unlike the paradigm shift to a software economy over twenty years ago with other hardware changes [16, 17]. Care should be taken to implement the application in such a way as to utilize not only the underlying computational power in the available system, but other capabilities (memory, file processing, etc.) in order to fully maximize utilization of the investment in the system.

Other efforts have targeted Ada to both general purpose and application specifichardware, particularly in the parallel programming / concurrent applications domain [18, 19]. Ada is already making some inroads into applications with parallel / concurrent opportunities, such as astrophyscis [20], but would be a natural fit for other areas such as non-defense embedded systems (communications systems, routers, etc.) as well as high performance computing [21].

7. CHALLENGES

The use of Ada has been hindered by many issues during its lifetime. Some of these have included: poor performance of early compilers, the defense "taint", lack of "coolness". Even today, just looking at the sponsors and presenters at recent SIGAda conferences, the industry is primarily contractors and academics – a community selling to one another. Most of the software that is written in Ada is for limited quantity, albeit large scale, deployment. There may be hundreds of Boeing 777s or 787s, but there are nothing like the nearly 5 billion (yes, Billion) devices running Java software [22].

8. ACKNOWLEDGMENTS / DISCLOSURE

The author would like to acknowledge Sun Microsystems, who contributed the computing equipment for this effort, and in which the author maintains an investment.

9. REFERENCES

[1] http://www.webservices.org/weblog/patrick_leonard/the_multi_core_dilemma – The Multi-Core Dilemma

[2] http://www.devx.com/enterprise/Article/34588/1954 - Do Newer Processors Equate to Slower Applications?

[3] – Linnig, Michael and Forinash, Donna, "Ada Tasking and Parallel Processors", ACM 0-9079329-9/89/0010-0426.

[4] White, James B. III "Performance Issues of Scientific Programming in Ada 95", Annual Internaction Conference on Ada, Proceedings of the conference on Tri-Ada '97, St. Louis, MO, 1997. ACM 0-89791-981-5.

[5] – Brosgol, Benjamin – "A Comparison of the Concurrency Features of Ada 95 and Java", SIGAda '98, p. 175 – 192, November 1998, Washington, DC.

[6] – Marowka, Ami – "Parallel Computing on Any Desktop", *Communications of the ACM*, September 2007, Vol 50, No. 9, p 75 – 78.

[7] http://http://news.uns.purdue.edu/x/2007b/070807PaiComputer.html

[8] http://www.eetimes.com

[9] – via http://www.eetimes.com – keynote from Multicore Expo in Santa Clara, March / April 2007.

[10] http://www.sun.com/tryandbuy/prm/perf/winners.jsp - Sun Open Performance Contest Winners.

[11] – http://www.grebyn.com/t1000 - Grebyn Corporation's T1000 Evaluation.

[12] – Cohen, Norman; "Parallel Quicksort: An Exploration of Concurrent Programming in Ada", p II.2.61 – II.2.68.

[13] – Levine, Gertrude – "The Game of Life with Ada Tasks", *Ada Letters,* Nov/Dec 1997, Volume 17, Num. 6, p. 19 – 31,.

[14] – Sanden, Bo I. "Intiutive Multitasking in Ada 2005", Crosstalk Aug. 2006.

[15] – Welbourne, Porter "Software economy in a rapidly changing hardware environment", Journal of Medical Systems, Volume 8, Numbers 1-2 / April 1984.

[16] – Stephen Zeigler et al, "Ada for the Intel 432 Microcomputer", IEEE Computer, pp.47 – 56, June 1981.

[17] – Schonberg, Edith; Schonberg, Edmond; "Highly Parallel Ada – Ada on an Ultracomputer", p 58 – 70.

[18] – Goforth, Andre; Collard, Philippe; and Marquardt, Matthew - "Performance Measurement of Parallel Ada: An Applications Based Approach", *Ada Letters Special Edition*, *Volume X, Number 3.* p 38 – 58.

[19] – Stift, Martin J. "Astrophysical Software Engineering in Ada", Institut fur Astronimie, Turkenscahnzstr 17 A-1180

[20] – Collard, Philippe; Goforth, Andre; Marquardt, Matthew – "Ada As A Parallel Language for High Performance Computers: Experience and Results", ACM – 089791-409-0/90/1200-346. p. 346 – 351.

[21] – Schwartz, Jonathan - http://blogs.sun.com/jonathan/entry/i_believe_in_network_clients