Occam: A primary parallel programming language

Masoud Nosrati *
Young Researchers Club,
Kermanshah Branch,
Islamic Azad University,
Kermanshah, Iran

Ronak Karimi
Young Researchers Club,
Kermanshah Branch,
Islamic Azad University,
Kermanshah, Iran

minibigs_m@yahoo.co.uk rk_respina_67@yahoo.com

Abstract: In this paper, we are going to return to early parallel programming language Occam and investigate its features, hardware, concepts, versions and concurrency. Due to this, we will begin with the development history of this program. Then, the Transputer as the hardware that Occam is using it will be investigated as pioneering microprocessor architecture of the 1980s, featuring integrated memory and serial communication links, intended for parallel computing. Different versions of Occam including Occam 1, Occam 2, Occam 2.1 and Occam π are going to be talked in follow. Characteristics and features on these versions are talked in details.

Key word: Occam • programming languages • parallel programming • early programming languages

I. INTRODUCTION

Occam was developed by David May [1] at Inmos Limited, Bristol, England. It is a concurrent programming language that builds on the Communicating Sequential Processes (CSP) process algebra [2]. A more careful treatment of CSP is in Hoare’s book on CSP [3]. Using CSP as a basis, the researchers at Inmos developed an Occam concurrency model. From the Occam model, they developed the programming language Occam. The name is derived from William of Occam, a thirteenth century philosopher. Occam’s Razor or the ancient philosophical principle of “keep things simple,” is attributed to William. A primary goal of the Occam language is to keep the language simple, hence the name [10].

Occam works on Transputer. The Transputer was pioneering microprocessor architecture of the 1980s, featuring integrated memory and serial communication links, intended for parallel computing. It was designed and produced by Inmos, a British semiconductor company based in Bristol [5]. In other words, it is in the form of a very large scale integration (VLSI) integrated chip (IC) [6] [7]. The Transputer (Inmos part number T800) is a 32-bit microprocessor (20 MHz clock) that provides 10 MIPS (million instructions per second) and 2.0 MFLOPS (million floating point operations per second) processing power with 4K bytes of fast static RAM (Random Access Memory) and concurrent communication capability all on a single chip. Though Occam is a high-level language, it can be viewed as the assembly language for the Transputer. Unlike most microprocessors, e. g., the M68000, the definition of the operations of the Transputer is in terms of the Occam model and not machine language. Because the Transputer is designed to execute Occam, the compiler can generate very efficient and compact machine code. Besides being a high performance microprocessor (half the speed of a VAX 8600), the Transputer has on its chip four (4) serial bi-directional links (each 20 Megabits per second) to provide concurrent message passing to other Transputers. The “channels” in the Occam language are mapped to these hardware links which connect by way of twisted pairs of wires to other Transputers. The Transputer hardware supports concurrency by scheduling (time-slicing), in round-robin fashion, an arbitrary number of Occam concurrent processes. The language and the hardware are so designed that an Occam program consisting of a collection of concurrent processes may execute on one Transputer (via time slicing between the different concurrent processes) or be spread over many Transputers with little or no change in the Occam code. Therefore, the designer can develop his or her Occam program on one Transputer, and if higher performance is required, can spread the Occam processes over a network of interconnected Transputers. The original Transputer (T414), having no floating point unit and only 2 Kbytes of RAM, became available in 1985. Two years later, the T800 Transputer was introduced and is
used quite widely in many vendor products. Inmos is currently developing a new, faster version of the Transputer called the “T9000,” which is scheduled to be available the middle of 1992. The T9000 will be a 150 MIPS microprocessor with a 20 MFLOPS floating point unit. The four links of the T800 will be replaced by more “virtual” links, with each link’s speed at 100 Megabits per second. Memory on chip will be increased from 4 K bytes to 16 K bytes and will include memory mapping and memory protection [10].

Occam has different versions that we are going to introduce them with the distinguished features of each version in below:

**Occam 1**

It was distributed in 1983 by Inmos for research laboratories and universities. This version had many forms. The most important of them are [10]:

- Portakit: It was a FORTRAN source program with low speed.
- VAX VMS: This version corrected most of the errors of the Portakit and was distributed to universities.

**Occam 2**

It was distributed in 1986. In this version, the Occam model of concurrency has not changed; however, Inmos has added many features which we have come to expect in a modern high-level programming language, notably types and type checking. For the numerical programmer, important new features added were floating point arithmetic and multi-dimensional arrays. In 1988, David May added protocols on the Occam channels, user defined functions and “include” files to Occam 2 [8].

**Occam 2.1**

It was the last of the series of Occam language developments contributed by INMOS. Defined in 1994, it was influenced by an earlier proposal for an Occam 3 language (also referred to as “occam91” during its early development) created by Geoff Barrett at INMOS in the early 1990s. A revised Reference Manual describing Occam 3 was distributed for community comment [4], but the language was never fully implemented in a compiler [2]. Occam 2.1 introduced several new features to Occam 2, including:

- Named data types (DATA TYPE x IS y)
- Named Records
- Packed Records
- Relaxation of some of the type conversion rules
- New operators (e.g. BYTESIN)
- Channel retyping and channel arrays
- Ability to return fixed-length array from function.

**Occam-π**

It is the common name for the Occam variant implemented by later versions of KRoC, the Kent Retargetable Occam Compiler. The addition of the symbol “π” (pi) to the Occam name is an allusion to the fact that KRoC Occam includes several ideas inspired by the Pi-calculus [9]. It contains a significant number of extensions to the Occam 2.1 compiler, for example:

- Nested protocols
- Run-time process creation
- Mobile channels, data, and processes
- Recursion
- Protocol inheritance
- Array constructors
- Extended rendezvous
II. OCCAM CONCURRENCY

The Occam model supports concurrency, i.e., true parallelism on several processors or simulated parallelism on one processor by way of time-slicing. The model is based on concurrent processes. In Occam, communication between concurrent processes is achieved by passing messages along point to point channels. Point to point means that the channel’s source and destination must be at one point or reside in one concurrent process. Below, process P1 can send a message by way of channel C to process P2 [10].

![Fig.1 Process P1 sends a message along channel C to process P2](image)

To alleviate many problems caused by interference when sharing variables between concurrent processes, all communication between concurrent processes in Occam must be by way of channels. Hence, there are no shared variables in Occam. (Actually, two concurrent processes can both read a variable, but one cannot read while the other writes into a shared variable.) Therefore, Occam reflects the message-passing model of parallel computation that is supported by the Transputer hardware with its local memory, i.e., no global memory, and communication links. The communication on an Occam channel is synchronous. When either the sender or the receiver arrives at the proper place in the code, the first to arrive waits for the other. Once synchronized, the message is transferred between the two, and then they continue executing. This action is similar to an Ada rendezvous. A consequence of this style of communication is that it provides no automatic buffering. If buffering is desired, an intermediate process may be inserted between the two processes. One advantage of this scheme is that one language feature performs both synchronization and message passing [10].

III. CONCLUSION

In this paper, we introduced the Occam programming language as a good choice for learning and real world parallel programming in the early stages. We saw that, the Occam programming language is a viable language to express concurrency. It is especially valuable for teaching the concepts of parallel algorithms in the message passing paradigm. One of Occam’s redeeming aspects is its simplicity. One only needs to compare it to other programming languages that support concurrency, e.g., Ada, to be convinced of Occam’s simplicity. The Occam concurrency model makes it easy to think and reason about an Occam program as a network of concurrent processes which only passes messages, i.e., and the programmer does not have to deal with shared variables and their associated problems. In Occam, one expresses concurrency explicitly at the statement level, i.e., the PAR construct, while in other languages, e.g., Ada, concurrency can only be expressed implicitly at the procedural level. In a language like Ada, the “heavy” machinery to support concurrency tends to discourage the programmer from using concurrency. With Occam, the PAR is a natural and easy construct to use [10].

REFERENCES


