Preface

This is a book about designing and writing certain sorts of parallel program. As parallel computers become more common, and as common computers become more parallel, it is important that the programmer cast off the shackles of sequential thought. Although the principal virtue of parallel computers would appear to be their speed, this book is not greatly concerned with writing programs in parallel to make them go faster. Rather, in this book what is important about parallel programs is that they can be simpler to write and understand than sequential programs which achieve the same effect.

It used to be that writing a program meant finding some strict sequence of steps to achieve the desired end. This odd constraint is a legacy of the time when programs were written only to be executed by computers, and computers were machines that did one thing at a time. Programmers knew that any attempt to get such a machine to take on two tasks at once was to give themselves the additional burden of making sure that these tasks did not interfere with each other.

The occam language, intended by its designers as the assembly language of the INMOS transputer[12], takes the revolutionary step of enforcing the independence of parts of the program. An occam programmer can start from the assumption that the parts of a program cannot interfere with each other. Unless the programmer makes explicit provision to the contrary, parallel processes in an occam program are guaranteed to be entirely unaware of each other.

Theoretical models like CSP[8] and CCS[9] describe parallel computation using only *synchronization*: no process engaged in a synchronizing communication may proceed until its partners are simultaneously doing so. The handling of parallelism in occam is based on these models: the job of programming is making the otherwise independent processes in a program co-operate by explicit communication. As it happens, programs written in this way are quite likely to be suitable for execution on computers made of several processors, but that is incidental to our purpose in this book.

The occam programmer is encouraged to think of process creation, and of synchronization and scheduling operations, as being as cheap as any other 'primitive' actions. Indeed, even on a conventional processor, neither process creation nor scheduling need be any more expensive than, say, procedure invocation. This startling scale of costs gives the programmer much greater freedom of expression, and leads to an unaccustomed programming style. If you think the book is about anything, we would prefer you to think it is about this style.

The language used in this book, occam 2, supplements the program structures of the earlier prototype occam languages with some concrete data structuring. To the sequential sub-language of occam is added a conventional system of data types, which makes it easier to read and write programs and easier for a compiler to check that the programmer's intentions are properly implemented. The system of protocols on communications between parallel processes is the natural extension of this type scheme to the parallel sub-language.

This book begins with an introduction to occam 2, intended to make the rest of the work intelligible to anyone who is already able to write and understand sequential programs. Several short examples exercise some occam 2 idioms illustrating the way the language can be used. The bulk of the text is devoted to developing a number of sizeable examples. Most of these programs solve the same problems as those in the earlier *Programming in occam*[5], although the differences between the two languages alter the emphasis; some of the examples have been substantially reworked and expanded.

At the back of the book there is a concise summary of the occam 2 language, drawn from material in INMOS's *occam 2 Reference Manual*[1], which is also published in this series. The complete source code of the larger examples is reproduced; we do not expect that anyone might want to read hundreds of lines of code, but their presence may make it easier to settle ambiguities earlier in the presentation. A short bibliography contains some suggestions for related reading.

*   *   *

Many will find that their conversations have contributed to this book, among them Steven Ericsson Zenith and David May of INMOS, Geoff Barrett, Jim Davies and Andrew Kay of the Programming Research Group. To these and others whom we have negligently omitted, we are grateful for their time, patience, and insights. Such inaccuracies and obfuscations as you will find herein we were, as our colleagues could testify, able to concoct entirely unaided.

Oxford — G.J.
March, 1988 — M.H.G.
occam-book@uk.ac.oxford.prg