COBWEB-2: Structured Specification of a Wafer-Scale Supercomputer

by

Paul Anderson*, Chris Hankin**, Paul Kelly**, Peter Osmon* and Malcolm Shute***.

* Computer Science Department, The City University, Northampton Square, London EC1 VOHB, UK.
** Department of Computing, Imperial College of Science and Technology, 180 Queen's Gate, London SW7 2BZ, UK.
*** Microelectronics Unit, Middlesex Polytechnic, Bounds Green Road, London N11 2NQ, UK.

1. Introduction

This paper describes the COBWEB-2 machine, with particular emphasis on the role of formal specification in the architecture's development.

COBWEB-2 aims to execute programs written in purely-functional programming languages (for example ALFL [Hudak & Kranz 83] and Miranda [Turner 85]). The project hopes to derive very high performance from a combination of powerful compiler optimisations and implementation using whole-wafer integration - thereby enjoying massive parallelism and, by avoiding chip-to-chip communications, high sequential speed.

The graph reduction approach to parallel evaluation of functional programs is shown to give particularly good utilisation of working elements in a fault-tolerant wafer-scale processor array.

The work presented here has its roots in the COBWEB-1 design, reported in [Hankin, Osmon & Shute 85], but differs in several radical respects. COBWEB-2 has substantially more random-access memory distributed through the wafer and a more subtle memory management system alleviates the program library access bottleneck. COBWEB-2 also has a new fault-tolerant communications mechanism, supplanting Catt's spiral, which was employed in COBWEB-1.

We start with a review of functional programming languages, compilation, and the run-time program representation - program graphs decorated with director strings and annotations to control parallel activity. An abstract machine executing this representation is defined, and used as
the root of the specification-driven development of the architecture. The refinement of this specification has the structure represented below:

```
| Normal-order reduction-based abstract machine running the machine code. |
| Token-based specification, showing structure sharing |
| Parallel graph-reduction abstract machine. |
| Parallel abstract machine with eager parameter distribution. |
| Engineering model, pool of virtual reduction machines. |
| Distributed, copying, virtual memory system with garbage collection. |
| Fault-tolerant, packet-switched communications network. |
| Wafer-scale integration. |
```

This development is mirrored in the structure of the paper as follows. After section 2 has introduced functional languages, showing how programs are compiled into the machine's run-time representation, section 3 introduces term-rewriting systems, and uses the notation to define a normal-order reducer for COBWEB-2's machine code. Section 4 describes an extension to the notation to describe graph-reduction, and this is used in section 5 to specify the parallel graph-reduction abstract machine which forms a specification for the engineering model. Section 5 describes the engineering model itself, and section 7 shows how this separates the implementation into independent aspects - virtual processor management, memory management, and inter-processor communications. Finally, section 8 discusses how wafer-scale integration makes the design feasible, and why it is particularly well-suited to a graph-reduction model of parallel computation.

2. Compilation and Program Representation

Functional, or applicative, programming languages concentrate on data inter-dependencies, making potential parallelism implicit. Their built-in determinacy avoids many parallel programming pitfalls, and their simple algebraic properties facilitate rigorous software development as well as sophisticated compilers.

Programming languages like Miranda [Turner 85], and ALFL [Hudak & Kranz 83] have "lazy" semantics - that is their semantics is based on call-by-name parameter passing. This contrasts with most current fast functional language implementations, such as [Cardelli 84], or [Hope 86],