Transputer-Based Experiments with the ZAPP Architecture.

D.L. McBurney and M.R. Sleep
Declarative Systems Project,
University of East Anglia,
Norwich NR4 7TJ, England.

ABSTRACT.

We report experiments with a parallel architecture called ZAPP[2] simulated on several connected INMOS transputers. Besides the usual synthetic benchmarks (eg Nfib), our experiments covered a range of applications including matrix multiply, heuristic search and the 0-1 knapsack optimisation problem. Some of these applications cannot be adequately supported by the original ZAPP model. We report the modifications we made to ZAPP to accommodate them.

One experiment involved 40 transputers; we obtained a stopwatch speed of over one million function calls per second, corresponding to a relative speedup over a single simulated ZAPP element of 39.9, and a real performance improvement over a single transputer running the same algorithm directly programmed in OCCAM of more than 15. A similar experiment for matrix multiply confirmed that real improvements were obtained using ZAPP techniques.

Experiments with less structured problems, such as heuristic search and the 0-1 knapsack problem, revealed that the longer a problem took to solve using sequential implementation, the more likely it was to benefit from parallel solution using ZAPP techniques.

1.0 Introduction.

Exploring the potential of parallelism is receiving increasing attention [1, 4, 7]. There are a number of reasons for this interest:

a. Some applications, such as speech and image processing, require large amounts of computational power which sequential computing cannot satisfy. In the field of general purpose computing, knowledge based applications will also need powerful engines if acceptable performance is required.

b. Many problems have solutions which are most naturally expressed as static or dynamic networks of communicating processes. For example, control systems for industrial plants can be decomposed into a set of data collection processes, data integration and filtering processes, decision making processes, and effector processes (ie those which are responsible for realising the decisions made).

c. Very Large Scale Integration technology enables us to produce large numbers of computing elements at low cost, perhaps on a single large chip or wafer.

1.1 Architecture Background: Good results have been obtained by careful machine-oriented programming of large SIMD (Single Instruction Multiple Data) machines for specific applications and the current generation of supercomputers is based on this experience.

The search for more general purpose MIMD (Multiple Instruction Multiple Data) parallel architectures has been less successful, particularly in achieving performance which would improve as the number of computing elements grows. Early experiments gave disappointing results, even after significant programming effort was deployed. Successful exploitation of parallelism has usually been associated with
limiting both the degree of parallelism and the range of applications.

1.2 The Paradigm Approach: One difficulty in realising general purpose parallelism is that it is not in general sufficient to take a program designed for a sequential machine and recompile it for a parallel machine. Although progress has been made on automatic extraction of parallelism from sequential programs, it is in general necessary to re-think the problem in order to obtain an effective parallel solution.

Some rather abstract paradigms on which parallel solutions may be based have proved useful over a range of applications. Perhaps the simplest example is the divide and conquer paradigm, which is easy to understand and familiar to most programmers. In this paper we describe a parallel architecture (called ZAPP) for Divide and Conquer solutions, and report early experiments with a transputer-based implementation of ZAPP.

1.3 Plan of paper: Section 2 introduces the process tree interpretation of Divide and Conquer algorithms on which ZAPP is based. Section 3 describes the basic principle of ZAPP operation. Section 4 describes very briefly the OCCAM/Tranputer realisation of ZAPP we used in our experiments. Section 5 describes the experiments performed, and section 6 presents the experimental results. Section 7 contains the main conclusions.

2.0 The Divide and Conquer approach to parallelism.

A number of useful paradigms exist for obtaining parallel solutions to problems. These include pipelining, systolic networks, and parallel array processing. More recent paradigms include the connection paradigms of Hillis[6] and others. Here we will be concerned with the Divide and Conquer paradigm. This may be illustrated by the following function, which is written in a functional notation:

\[ \text{Nfib} (n) = \begin{cases} 1 & \text{if } n < 2 \\ \text{Nfib} (n-1) + \text{Nfib} (n-2) + 1 & \text{otherwise} \end{cases} \]

Operationally, this equation may be interpreted as a rewrite rule. For example Nfib(6) rewrites to (Nfib(5)) + (Nfib(4)) + 1. Parallel rewriting creates a number of sub-expressions which grows (initially exponentially) with time if all available sub-expressions are rewritten at each step.

Nfib is an example of the application of the Divide and Conquer paradigm for problem solving. This paradigm can be expressed by the D_C function defined as follows:

\[
\text{def } \text{D_C} (\text{primitive}, \text{divide}, \text{combine}, \text{solve}) = f \\
\text{where } f(x) = \begin{cases} \text{primitive}(x) & \text{if } \text{primitive}(x) \\ \text{solve}(x) & \text{else} \\ \text{combine}(\text{map}(f, \text{divide}(x))) & \text{fi} \end{cases} \\
\text{x is a problem description.} \\
\text{primitive} (x) \text{ returns true if } x \text{ can be solved directly, otherwise it returns false.} \\
\]