JOINT ROYAL MELBOURNE INSTITUTE OF TECHNOLOGY AND
COMMONWEALTH SCIENTIFIC AND INDUSTRIAL RESEARCH ORGANISATION
PARALLEL SYSTEMS ARCHITECTURE PROJECT

Parallel Programming: Easy As Π

TR 118 084 R

G.K. Egan

Digital Systems and Computer Engineering
Department of Communication and Electrical Engineering
Royal Melbourne Institute of Technology
124 Latrobe St
Melbourne 3000

Version 1.0 Original Document 1/6/89

ABSTRACT:

In a recent book entitled "Programming Parallel Processors" the computation of Π was used to explore the ease, or otherwise, of programming current generation parallel computer systems. In the book Babb mentions the language SISAL but does not explore its use on dataflow or other architectures; this short report remedies that omission.

1. INTRODUCTION

A recent book by Robert Babb (Ed.) [2] used the computation of Π to explore the ease, or otherwise, of programming current generation parallel computer systems. In the book Babb mentions the language SISAL [5] but does not explore its use on conventional computer systems or other systems such as dataflow multiprocessors.

The report presents a solution to the computation of Π expressed in SISAL and gives the execution times for a Sun 3/260 workstation, Encore Multimax multiprocessor and for interest the CSIRAC II dataflow multiprocessor [1][6]. No modification of the SISAL source or any other programmer intervention was required to achieve the results presented.

2. COMPUTING Π BY NUMERICAL INTEGRATION

The method used by Babb was to integrate the function $f(x) = 1/(1+x^2)$ over the interval 0 to 1 to obtain Π /4. A number of methods may be used to perform this integration the simplest being rectangular integration.

2.1 FORTRAN and SISAL Programs

A FORTRAN program which expresses this method is:

```
      PROGRAM PI
C
      DOUBLE PRECISION PION4,DX
      INTEGER RECS
C
      RECS = 1000000
      DX = 1.0 / RECS
      PION4 = 0.0
C
      DO 100 I=1,RECS
      PION4 = PION4 + DX / (1.0 + (DX * I) ** 2 )
100   CONTINUE
      WRITE(6, *) 4.0 * PION4
C
      END
```

The equivalent SISAL program is:

```
% compute Pi by rectangular integration
define pi
function pi(returns real)
        let
             Rectangles := 1000000;
             Dx := 1.0 / Real(Rectangles);
        in
             4.0 * for r in 1 , Rectangles
                       returns value of sum Dx / (1.0 + sqr(Dx*Real(r)))
             end for
        end let
end function
```

3. RESULTS

The SISAL and FORTRAN programs were run on a Sun 3/260 (68020/68881) and an Encore Multimax (32332/32xxx). The results for these systems and simulation results for SISAL on the CSIRAC II dataflow multiprocessor are presented in the following sections.

3.1 Conventional Machines

The results presented here were obtained using the standard f77 FORTRAN compiler released with the Sun 3/260 and Encore Multimax, and the Optimising SISAL Compiler (OSC) [3] from the Colorado State University. The FORTRAN compilations were performed with and without optimisation. Default optimisation was used for SISAL i.e. full optimisation without any parameter tuning. Table 3.1.1 below gives the results for a single CPU and Figure 3.1.1 gives the runtimes for a varying number of workers (~CPUs) for SISAL on an Encore Multimax.

	Time (Sec.)		
System	f77	f77 -O	OSC
Sun 3/260	42.4	24.5	27.8
Multimax	30.4	30.4	34.1

Table 3.1.1 FORTRAN and SISAL Results for a Single CPU

Figure 3.1.1 Runtime versus Workers (~CPUs) for SISAL on an Encore Multimax

It can be seen that the runtimes obtained using SISAL compare favourably with those obtained using FORTRAN on a single CPU. In addition a near linear speedup was obtained using SISAL on the Encore Multimax yielding a runtime of 1.77 Seconds for 20 CPUs (96.5% of ideal). There were 5 floating point instructions in the inner loop of 12 instructions yielding 0.14 MFLOPS and 0.34 MIPS respectively for each CPU. It is expected that the Encore floating point processor option would significantly improve these instruction rates as the accepted instruction rate for each Encore 32332 based CPU is 2 MIPs; this was verified for the non-floating point instructions in the inner loop. Similar speedups for other programmes written in SISAL are presented in [4]. The Encore Parallelising FORTRAN compiler (EPF) was not available at the time of writing and the single CPU FORTRAN results are presented here only as a baseline for the SISAL results.

3.2 CSIRAC II Dataflow Multiprocessor

An alternative formulation of the solution is to use a divide and conquer scheme to perform the integration. A SISAL program implementing a divide and conquer scheme is given below.

```
% compute pi by recursive binary subdivision of integration interval
define recursive
function recursive(returns real)
function Area(D: integer; dx, L,R: real returns real)
  let
    Mid := (L + R) * 0.5;
    NewD := D / 2;
  in
    if D = 0 then
      (R - L) /(1.0 + L * L)
    else
     Area(NewD, dx, L, Mid) + Area(NewD, dx, Mid, R)
    end if
  end let
end function
  let
    Rectangles := 1000;
    dx := 1.0 / Real(Rectangles);
  in
    4.0*Area(Rectangles,dx,A,B)
  end let
end function
```

It is not intended to compare the results for CSIRAC II with the conventional processors of the previous section but to augment Babb's results for dataflow systems with those of CSIRAC II.

The expected performance by simulation of a CSIRAC II Dataflow multiprocessor when performing the integration over 1000 rectangles is given in Figure 3.2.1 below. The left hand graph depicts workload distribution and the right hand graph the number of processors active in the computation.

Figure 3.2.1 Workload Distribution and Processor Activity (128 processor CSIRAC II)

Conclusion

This short report sets out to augment Babb [2] by presenting the results obtained using the SISAL language on current conventional computer systems, and the results which may be expected on a dataflow multiprocessor. While not presuming to be an exhaustive demonstration of the superiority of implicit parallel programming using languages such as SISAL and ID[7][8], the ease with which the results were obtained may be contrasted with the explicit style required by some other language and parallel programming environments examples of which are contained in the Appendix.

Acknowledgements

The RMIT/CSIRO Parallel Systems Architecture Project is jointly funded by the Commonwealth Scientific and Industrial Scientific Organisation (CSIRO) and the Royal Melbourne Institute of Technology.

References

[1] D. Abramson and Egan G.K, "Design Considerations for a High Performance Dataflow Multiprocessor", TR 122073R, Department of Communication and Electrical Engineering, Royal Melbourne Institute of Technology, 1988. *Presented at the Workshop on Dataflow Computing: A Status Report, Eilat, Israel 1989*

[2] R.G. Babb, "Programming Parallel Processors", Addison-Wesley, 1988.

[3] D.C. Cann and R.R. Oldehoeft, "Compilation Techniques for High Performance Applicative Computation", Technical Report CS-89-108, Colorado State University, May 1989.

[4] D.C. Cann, "High Performance Parallel Applicative Computation", Technical Report CS-89-104, Colorado State University, Feb.1989.

[5] McGraw et al, "SISAL: Streams and Iteration in a Single Assignment Language, Language Reference Manual", Lawrence Livermore National Laboratories, M146.

[6] N. Webb, "Implementing an Applicative Language for the RMIT/CSIRO Dataflow Machine", Department of Computer Science, Royal Melbourne Institute of Technology, *M.App.Sci. Thesis in preparation*, 1989.

[7] R. Nikhil, K. Pringali and Arvind, "Id Nouveau", Computation Structures Group Memo 265, Massachusetts Institute of Technology, Laboratory for Computer Science.

[8] P. Whiting, "IDA: A Dataflow Programming Language", TR112 075 R, Department of Communication and Electrical Engineering, Royal Melbourne Institute of Technology, 1988.

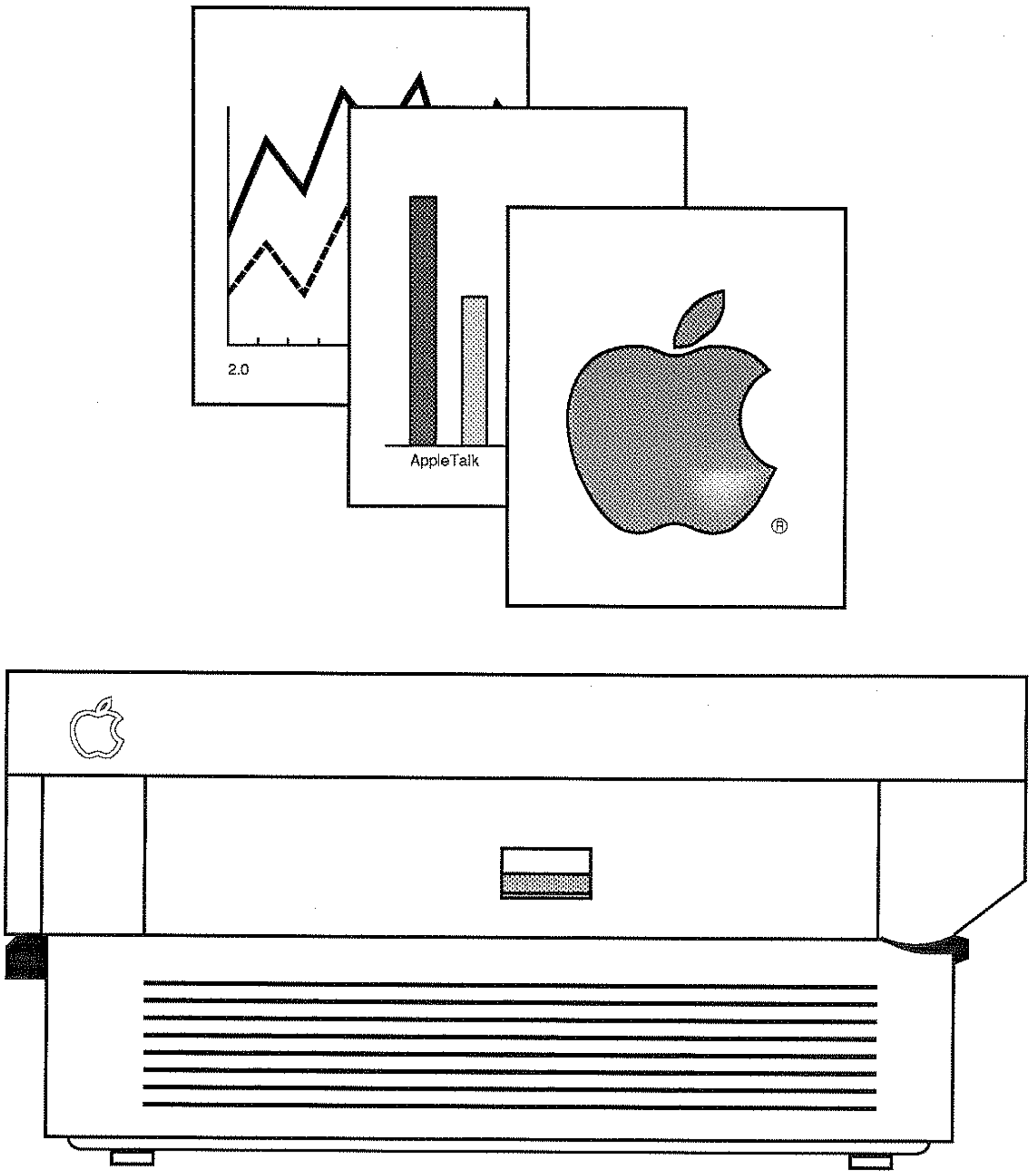
APPENDIX

Examples drawn from "Programming Parallel Processors", Addison-Wesley, 1988.

Sequent Balance (FORTRAN)

FPS T Series Parallel Processor (Occam)

Big Ears



81381