Submission for SC97
A Predictably Portable Process Farm Skeleton
extended abstract

Mohammad Hamdan*, Greg Michaelson and Peter King

Department of Computing and Electrical Engineering,
Heriot-Watt University, Riccarton, Edinburgh EH14 4AS, UK

{hamdan, greg, pjbk}@cee.hw.ac.uk

May 15, 1997

Abstract

Communications libraries like MPI enable architecture independent parallel program development but cannot alone guarantee performance consistency across different architectures. Here an algorithmic skeleton for a process farm is discussed and an architecture independent performance model is developed for it. The skeleton has been ported from a Meiko Computing Surface to a Cray T3D and a Fujitsu AP1000, demonstrating consistent behaviour. Preliminary performance prediction accuracy on the Cray T3D is around 90%, on a test application based on arbitrary length arithmetic. (The performance model is currently being instantiated for the other systems. This will be reported in the final paper.)

1 Introduction

It is well known that parallelism adds an additional level of difficulty to software development. Following Cole's characterisation[3], algorithmic skeletons have been recognised widely as a valuable basis for parallel software construction. A skeleton abstracts a control structure which may be instantiated subsequently with specific functions to carry out specific tasks. Common parallel skeletons include the process farm, which applies a function to each member of a data set, and divide and conquer which splits a data set, applies a function to each sub-set and then accumulates the results back together.

Parallel skeleton construction is greatly eased by communications libraries like MPI which enable the elaboration of architecture independent process topologies. Nonetheless, the performance of a skeleton on a particular parallel

*Author for correspondence

system depends on the underlying processing and communication characteristics. Simply moving a skeleton from one system to another is no guarantee of performance consistency.

We are investigating an approach to skeleton portability based on parameterised performance models[4]. The models are instantiated for a new architecture using appropriate instrumentation. Here we discuss the development of a performance model for a process farm and evaluate the predictability of its portability across 3 parallel systems.

2 The process farm

A process farm is a widely used construct for data parallelism[2]. A farmer process has access to a pool of worker processes, each of which runs the same function. The farmer distributes data to workers and collects results back. The effect is to apply the function to every data item.

There are a variety of topologies for process farms including linear lists and trees of workers. However these require relatively low level intercommunication and incur overheads for intermediate message passing between workers. Instead, we have taken advantage of MPI's flexibility to avoid commitment to a concrete topology, and have implemented a broadcast farm. The farmer communicates directly with each worker through virtual connections and we rely upon the MPI implementation on each system to allocate processes to processors.

It might appear that this compounds the unpredictability of performance on an individual system, let alone across systems. However, it is testimony to MPI's robustness that we have found consistent behaviour for the basic skeleton in early experiments on three different architectures, under a dummy processing load, as shown in Figure 1.

The farm has a generic communication harness which enables transmission of arbitrary data structures based on a subset of Standard ML's types: lists, tuples, integers, reals, booleans and strings. The sender flattens a structure to a byte sequence, determined by its type. No type information is held in the flattened form as the receiver already knows the structure's type. This is then used to parse the byte sequence to reconstruct the structure.

It may be appropriate to vary the number of sub-structures sent to a worker, to modify the granularity of processing. Thus several flattened sub-structures may be assembled into a single message for transmission in one step. We refer to a sub-structure as a work packet.

3 The performance model

To derive a performance model for a skeleton, the skeleton's performance must be broken down into its principal components with respect to communication and computation times involved. Figure 2 shows the processor farm components that are relevant to the performance model.

Combining the principal components yields a general performance model

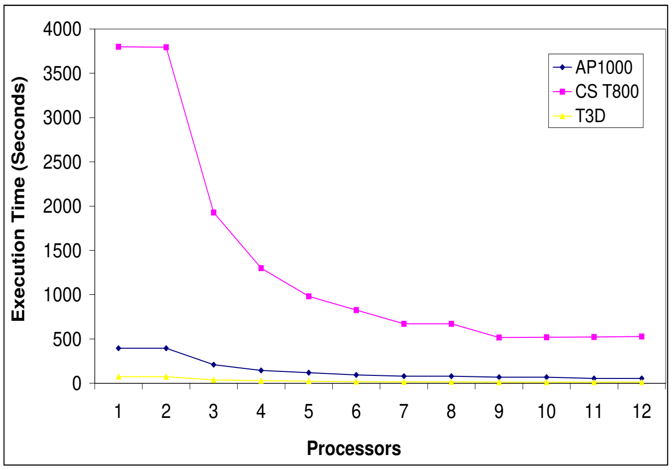

Figure 1: Processor farm behaviour on Cray T3D, Fujitsu AP100 and Meiko CS with T800s.

for the process farm skeleton.

$$T_{Farm} = \begin{cases} T_{Setup} + N * T_{Comp} & \text{if } P = 1 \\ T_{Setup} + S * T_{Worker} + \left\lceil \frac{N}{CU} \right\rceil * T_{Comm} & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} T_{Setup} &= T_{LoadProgram} + T_{Initialisation} \\ T_{Worker} &= CU * T_{Comp} \\ T_{Comm} &= T_{Send} + T_{Recv} + T_{Flatten} + T_{Unflatten} \end{aligned}$$

$$S = \left\lceil \frac{N}{CU * (P-1)} \right\rceil$$

T_{Farm} is the predicted execution time of the process farm. T_{Setup} is the time that is taken to allocate processes to processors and set them going. T_{Comp} is the time taken to process a single packet. T_{Worker} is the computation time used by a worker for a single message. T_{Comm} is the time taken to organise the data into messages for transmission between process and perform that transmission.

P is the number of processes, N is the total number of elements, CU is the clustering unit (the number of packets combined in a single message). S stands for the total number of steps the farmer has to go through for processing all work packets. A single step is to distribute a work packet to each worker then collect the results back.

Our model is similar to those of other workers, for example Bratvold [1] and Darlington et al [5].

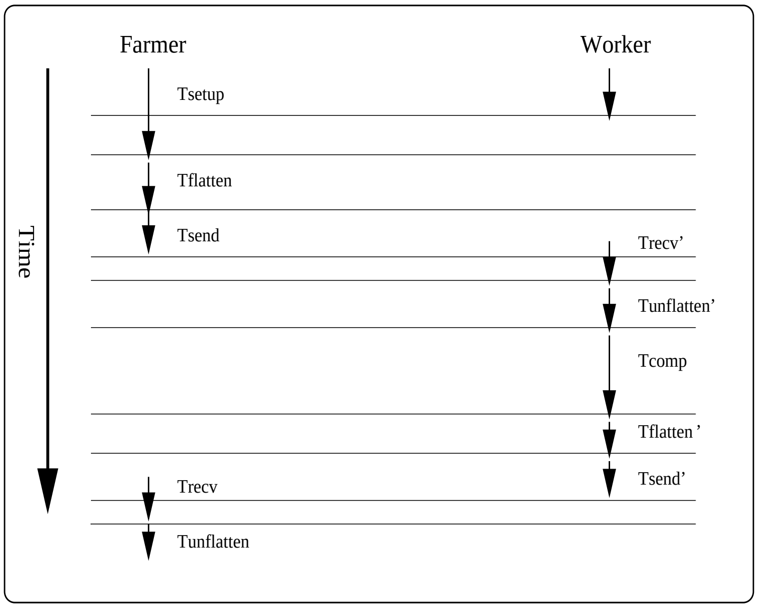

Figure 2: Processor farm skeleton timing relationships.

The model needs to be instantiated with measurements taken from the target architecture. T_{Send}, the time to transmit a packet from the farmer, T_{recv}, the time to receive a packet at the farmer, and $T_{Flatten}$ and $T_{Unflatten}$, the times for flattening and un-flattening the data structures are all measured directly. In future, we intend to derive formulas for these times from the properties of the data structures to be passed.

4 Evaluation

The process farm is being tested through arithmetic on Arbitrarily Large Numbers (ALN)[6]. Here numbers are represented as lists of decimal digits. Arithmetic is based on mutually recursive functions, ultimately incrementing and decrementing numbers. This provides a simple but flexible basis for varying both the communication and processing loads. Our implementation follows a "little endian" convention of storing the least significant digit of a number at the head of its list.

Figure 3 shows the measured versus predicted execution time for ALN vector addition on the Cray T3D with $CU = 1$. Here there were 500 numbers in each vector and each number had 3 digits:

This evaluation shows good correspondence between the predicted and actual performance, to around 90% accuracy.

5 Conclusion

These preliminary findings suggest that our process farm is robust across different systems and its performance model provides accurate predictions of be-

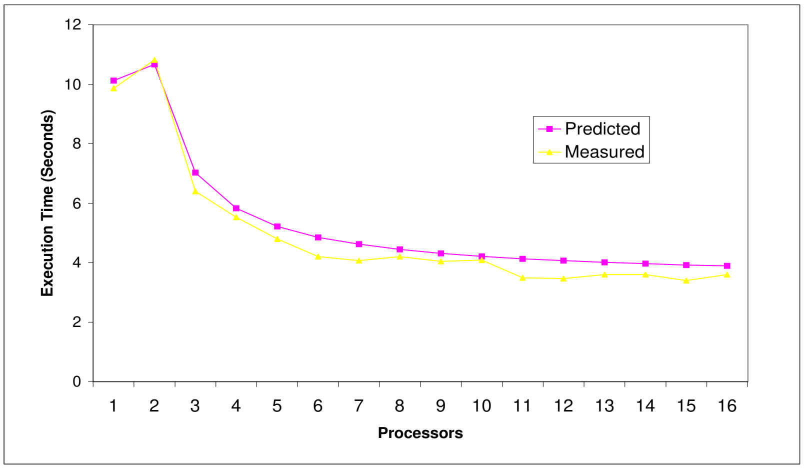

Figure 3: Measured versus predicted execution time of ALN on Cray T3D with CU=1.

haviour.

(For the final paper, we will provide evaluation on the Fujitsu AP1000 and the Meiko Computing Surface, and we will discuss the general accuracy of the model.)

6 Acknowledgments

We would like to thank the Edinburgh Parallel Computing Centre for access to their Cray T3D and the Imperial College Fujitsu Parallel Computing Research Centre for access to their AP1000.

We would also like to thank the British Council and Yarmouk University for supporting this research.

Our Meiko Computing Surface was funded by SERC grant GR/G30522.

References

[1] T. Bratvold. *Skeleton-based Parallelisation of Functional Programmes.* PhD thesis, Dept. of Computing and Electrical Engineering, Heriot-Watt University, 1994.

[2] Paul. D. Coddington. The Application of Transputer Arrays to Scientific Problems. In John Hulskamp, editor, *Australian Transputer and OCCAM user Group Conference Proceedings*, pages 5–10, Jun 1988.

[3] M. I. Cole. *Algorithmic Skeletons: Structured Management of Parallel Computation.* Pitman/MIT, London, 1989.

[4] G. Michaelson et al. Parallel Implementations from Functional Prototype Instrumentation. In F.J. Peters G.R. Joubert, D. Trystram and D.J. Evans, editors, *Parallel Computing: Trends and Applications, Proceedings of the International Conference ParCo93*, pages 631–634. Elsevier Science B. V., September 1994.

[5] J. Darlington et al. Structured Parallel Programming. In *Proceedings of Massively Parallel Programming Models Conference*, Berlin, September 1993.

[6] Peter Robinson. From ml to c via modula-3: an approach to teaching programming. In Mark Woodman, editor, *Programming Language Choice*, pages 149–169, UK, 1996. International Thomson Computer Press.