Chapter 8

Conclusions and Directions for Future Research

8.1 Thesis Summary

A model of parallel computation (HOPP) based on the functional style and the Bird-Meertens Formalism has been investigated in this thesis. In particular, the thesis has focussed on developing a realistic cost model for HOPP. The model has been targeted at distributed-memory MIMD machines in which the costs of interprocessor communications tend to be significant. Consequently, sufficient effort has been directed at minimising these costs. The motivations for choosing BMF as the basis for the model of parallel computation lie in the advantages it offers and includes the following:

- It provides a high level of abstraction, therefore relieving the programmer of handling the low-level details involved in parallel programming.
- Only one program is written irrespective of the target architecture, which enhances the portability of the program.
- The model comprises of a fixed repertoire of constructs, most of which are implicitly parallel. Any potential parallelism in the program can only be expressed through these constructs. It should be possible to predict the cost

of execution of such a program on a given architecture. Also, once such an estimate has been obtained for a given architecture, it can be re-used for different programs expressed in terms of these constructs.

- The set of implicitly parallel constructs are either already defined, or are easily definable in most functional languages, thereby removing the need for learning any new programming technique in order to use the model.

The implicitly parallel constructs are higher-order functions which are based on the functions in BMF. The basic set of higher-order functions in BMF has been extended by incorporating some additional useful functions. These additional functions are not *new*, but are well-known functions for which definitions in terms of the functions in BMF have been provided, along with cost-effective parallel implementations on four target topologies. This extended set of implicitly parallel functions is referred to as the set of *recognised* functions. A program is expressed in terms of these constructs, typically as a composition of instances of recognised functions. For each recognised function, a parallel implementation is defined on a given target parallel machine topology, and the cost corresponding to that parallel implementation is derived. Associated with each target topology is a cost model which describes the possible methods for implementing a given program in parallel, together with the corresponding cost estimates. A cost analysis is performed on the program, using the analytical cost model that has been developed for the chosen target architecture. The analyser considers the costs associated with the various possible implementations and selects the one that results in the least cost. Parallel code can then be generated for the selected implementation, which, in turn, can be executed on the parallel machine.

The performance of the recognised functions has been studied on four target architectures, viz. hypercube, 2-D torus, tree and linear array. The performance of a majority of these functions was found to be most efficient on the hypercube and hence an analyser has been implemented for this topology. Three example programs were analysed and the chosen parallel implementations were executed on a transputer-based machine configured as a hypercube. In all the cases, the predictions of the analyser have been confirmed by the actual implementations. The 2-D torus follows the hypercube as the one on which the functions can be efficiently implemented. However, due to the poor connectivity of the torus, several possible parallel implementations for a program would probably be inefficient. A hypercube embedding in the torus was proposed as a possible approach in overcoming this inefficiency. A possibility then would be to compute two sets of costs associated with implementing a program on the torus:

- With the torus configured as a hypercube - this would use the defined hypercube embedding and analyse the cost of the program with respect to the embedded hypercube topology.

- With the torus retained as such - this would analyse the cost of the program by considering the possible implementations on the torus itself.

The least-cost implementation could then be chosen.

The tree and the linear array proved to be quite inefficient and were not investigated further. It is possible to develop cost models for other target topologies, thereby extending the range of architectures catered for by the HOPP model. For each such architecture, a parallel implementation must be provided for all the recognised functions, together with the corresponding costs. The cost model must describe the techniques for handling the implementation of nested recognised functions and must provide cost estimates for any data distribution functions which may be involved in the implementation of programs.

8.2 Contributions of Thesis

The main contributions of this thesis can be summarised as follows.

- The development of the HOPP model which provides an extended set of predefined implicitly parallel constructs called *recognised functions*, in an effort to make the task of writing a wider class of programs more feasible.
- Cost-effective parallel implementations have been devised for the functions in this extended set on four different interconnect topologies. The cost associated with each such implementation has also been derived.
- A hierarchical cost model has been developed for these topologies. The cost model for the hypercube topology has been implemented in the form of an analysis program. The analyser considers the costs of the parallel implementations for the recognised functions and their arguments, in the selection of a cost-effective parallel implementation for the program. It also considers the costs associated with a number of possible parallel implementations for the program before selecting a cost-effective one.
- The performance of the model has been illustrated by implementing three example programs. The predictions of the analyser have been confirmed by the experimental results, in almost all the cases.

8.3 Limitations

There are some limitations to the analyser in its current form. Most of these can be overcome by the incorporation of some additional features in the analyser. However, there are some limitations to the model arising from the nature of the chosen constructs and cost analysis, and these cannot be wished away. They constitute the trade-offs involved in obtaining a simple parallel programming model that provides a high level of abstraction. These limitations are discussed below.

- The main limitation is that only *regular* problems can be effectively parallelised using this approach. In this context, a *regular* problem is one whose performance does not depend on the nature of the input data and one which also has a predictable communication structure. For problems which are not *regular* in this sense, the analyser cannot be guaranteed to make performance predictions that will reflect practical behaviour. This limitation arises partly due to the use of compile-time techniques to predict performance and also due to the nature of the constructs available for expressing parallelism.

- The fixed repertoire of constructs for expressing programs does limit the programmer. Although this set comprises of functions which should allow the expression of a number of types of data-parallel operations, situations could arise when this becomes difficult. For example, it may be difficult to express a program based on an array data structure with array operations.

- The analyser in its current form requires more assistance from the programmer than is actually necessary, e.g. it requires estimates of the sizes of input data structures and the costs of sequential functions. These demands on the programmer could be removed by extending the analyser so that it incorporates profiling capabilities. The analyser also lacks type inference and requires the programmer to explicitly supply the base types of input lists and also the input and output types of sequential functions. This could also be automated.

- The analyser cannot detect certain types of transformations of the input lists by sequential functions (See Section 5.1). Consequently, the estimates of the sizes of the corresponding output lists would be inaccurate. This in turn would result in inaccuracies in the cost estimates for subsequent phases of the program. The analyser could be extended so that some of these transformations are detected. Again, this would involve incorporating type inference in the analyser. However, due to the compile-time techniques which are used, a transformation of an input list resulting in an output list of different size, cannot be detected by the analyser.

8.4 Avenues for Further Research

The research reported in this thesis opens several avenues for further investigation.

- An immediate possibility would be the construction of a compiler to automatically generate parallel code for the implementation selected by the analyser. Such a compiler would be targeted at a parallel machine like the Meiko Computing Surface or the Cray-T3D. The design of the compiler could be such that it generates code in a high-level language such as *C*, with the code for the appropriate communication constructs being generated. Standard communication harnesses such as MPI could be used. This would have a double advantage in that the issues involved in generating efficient machine code would be transferred to standard *C* compilers and the portability of the program would also be enhanced. The onus of generating efficient *C* code is, however, still on the compiler. Another option is to design the compiler such that it generates code in some intermediate form, which in turn could be executed on the parallel machine. Some method of inserting the communication constructs would have to be devised.

- The cost model was studied on four different target topologies, of which only two were found to be cost-effective. The model could be extended to cater for other interconnect topologies, such as, fat-trees [Lei85].

- The hypercube embedding in the 2-D torus can be implemented. An analyser to predict execution costs and select cost-effective implementations on the torus can then be realised.

- The scheme only considers parallel implementations for recognised functions that do not occur within sequential functions. Any occurrences of recognised functions within sequential functions are implemented sequentially. A possible extension would be to consider the exploitation of potential parallelism for recognised functions nested within sequential functions. This would possibly involve the formulation of rules that describe when such exploitation of parallelism is allowed, so that semantic inconsistencies are not introduced into the program.

- The cost model is hierarchical but it considers parallel implementations for nested recognised functions only up to the first three levels. This could be extended to handle more levels, if the situation arises. Alternatively, the number of recognised functions implemented in parallel could still be three, but instead of simply choosing the first three recognised functions for analysis, parallel implementations involving recognised functions in other levels could be analysed.

- The search tree constructed by the analyser is exhaustive in the sense that it considers all the possible parallel implementations described by the model, for every phase in the program, before selecting the least-cost implementation. This causes an exponential increase in the size of the search tree with increase in the number of phases in the program. However, this is probably not required since many of the implementations would perform too poorly to merit consideration. Heuristics could be designed to eliminate such poor implementations at the analysis stage itself, in order to reduce the search space. This would probably involve a combination of local minimisation and inter-phase elimination techniques.

- The limitations of the analyser discussed in Section 8.3 could be overcome by extending the analyser to incorporate the required capabilities.

- The analyser cannot handle problems where the performance of sequential functions depends on the actual *value* of the input data. In other words, the cases where the cost of a sequential function varies for different data values, cannot be handled. This arises due to the limitation of *regularity* that is imposed on the nature of the problem that can be analysed. However, the introduction of profiling techniques in the analyser could be used to overcome the problem to some extent. Profiling could be used to obtain estimates of the values of each data set. This could, in turn, be used by the analyser to predict execution costs and select a cost-effective parallel implementation for that data set. The implementation of a phase containing a sequential function whose cost is a function of the input value, would probably involve placing different numbers of data items on processors, in order to achieve load balance. The analyser could be extended to handle such cases, thereby making it less restrictive.

- Program transformations can be employed to generate efficient parallel implementations for programs. Programs written in the functional style naturally lend themselves to transformations. The cost model could be used to predict the costs of different versions of the program, obtained by using semantics-preserving transformations. The parallel implementation corresponding to the least-cost version could then be selected for execution.

- More example programs from different scientific disciplines could be tested on the scheme. This could assist in identifying more useful functions which could be incorporated in the set of recognised functions.
- The model is currently based on distributed-memory machines. It would be interesting to develop a cost model for shared-memory machines.