氏　　　名	高井昌彰（たかい　よし　あき）
授与学位	工学博士
学位授与年月日	昭和63年3月25日
学位授与の根拠法規	学位規則第5条第1項
研究科，専攻の名称	東北大学大学院工学研究科 （博士課程）情報工学専攻
学位論文題目	A General−Purpose Pipeline System for the Function−Level Programing Language （関数型言語指向汎用パイプライン処理方式に関する研究）
指導教官	東北大学教授　重井　芳治
論文審査委員	東北大学教授　重井　芳治　　東北大学教授　木村　正行 東北大学教授　野口　正一　　東北大学助教授　中村　維男

論　文　内　容　要　旨

Chapter 1 INTRODUCTION

There has been considerable interest in functional programming, which has great possibility to enhance lucidity and maintainability of programs, and further, to increase programmers' productivity. Function−level programming language (FP) proposed by J. Backus is one of purely functional languages. Even though functional programming has great potential for relieving "software crisis", there has been a major drawback that functional programs run slowly on conventional von Neumann machines.

So far, aiming at realization of ideal function−level computing, many efforts have been paid to find out non−von Neumann, parallel reduction machines which can execute functional programs efficiently. From an architectural point of view, most of the functional language−based, parallel machines already proposed aim at developing fine−grained parallelism in a rich MIMD architecture. On the other hand, today's successful supercomputers, in which the well−known pipeline architecture is from the cost−reducing design technique, have mainly been developed in the area of numerical computations.

In this thesis, as another challenge to more cost−effective function−level computing, the author tries to join the two quite different fields of computer architectures together:

the language–based, sophisticated parallel reduction machnes, and the traditional pipelined supercomputers. The goal of this thesis is to establish the systematic methods for realizing a brand–new MISD machine based on the overlapping between the graph reduction and the pipelined supercomputing. This will be achieved by means of answering the three how's: (1) how the function applications dealing with list–structured data are executed in a pipeline (to be developed in Chapter 2), (2) how the reducible function applications are detected and issued to the pipeline (to be developed in Chapter 3), and (3) how the efficiency of pipelining function–level programs can be improved (to be developed in Chapter 4).

Chapter 2 PIPELINING RECURSIVE FUNCTION

Pipelining can be characterized in terms of linearity. Therefore, there exists a *natural applicability of the pipeline processing to linear recursive functions*: structural primitive functions dealing with list–structured data (objects) in FP. A linear recursion generally consists of two temporally successive: "unfolding phase" in which an object is analyzed in forward direction by both selection and decision operations, and "folding phase" in which a modified version is synthesized in backward direction by construct operations.

The most natural way to execute the linear recursive functions through a pipeline is to assign each instance of recursion to a different segment of the pipeline. In this case, however, both unfolding and folding of linear recursion must be achieved concurrently, from the general limitation of a pipeline: one–way single data stream. This requirement means that the folding should be done in each segment of the pipeline, while executing the unfolding at the same level of recursion.

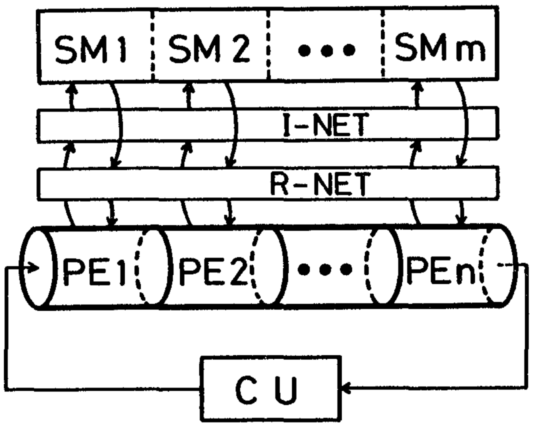

Fig.1 List processing–oriented general–purpose pipeline.

Fig 1 shows a general architecture of a list processing-oriented general-purpose pipeline. Here, the SM means a structure memory composed of independent banks of heap memory. List-structured data are carefully allocated into the memory banks. Then the PE is an autonomous processing element. The PEs are connected linearly, and construct an asynchronous linear pipeline. The functions of each PE are assigned dynamicly,which results in a general-purpose pipeline. Moreover, each of PEs is also connected to an arbitrary memory bank through the interconnection networks. According to the data processing scheme, this pipeline can be regarded as being in an MISD architecture.

At the implementation level, the folding in parallel with unfolding is realized by using the notion of "pipeline cons" which stems from the "lenient cons" to achieve the stream parallelism in dataflow computers. The notion of the pipeline cons is very essential to the pipeline execution of recursive function dealing with list-structured data. To make parallelism in creating one list-cell and in writing one pointer into the car field of the cell and the other pointer the cdr field, a well-known "cons" operation can be basically decomposed into three more primitive operations: "get-cell", "write-car", and "write-cdr". In the pipeline cons, while both get-cell and write-car to the newly generated cell are executed at the same time, the write-cdr to the cell is executed later by another pipeline cons. Therefore, in a list construction using the pipeline cons, a list structure is always built from the head toward the tail of the list.

Chapter 3 CONTROL SCHEME OF REDUCTION

To deal with more complex structures in the pipeline architecture, a graph reduction technique is introduced into the function-level computing. That is, based on a view point that any kind of recursion is the combination of linear one,the pipeline is regarded as a graph reducer dedicated to linear recursions. The formal meanings of FP, on the other hand, are given by the pure reduction system, called FFP. In FFP, all the functional forms, which play crucial roles in the function-level programming, are represented as sequences of objects through the rule of "metacomposition". Therefore, to relate the functional forms to their pipeline processing through the notion of the metacomposition rule, the whole FP programs are represented as list-structures, like a style of objects.

Here, two kinds of list-cells are employed to construct graph structures representing FP programs with objects. One is an "object cell" constructing a sequence of objects or functional forms, while the other is an "application cell" indicating the "function application". Program evaluation is carried out by reconstructing the graph structure according to the evaluation rules, towards a constant expession without application cells.

Based on the data dependencies among the newly generated application cells, functional forms can be classified into two types: a serial type and a parallel type. In the case of reduction of the serial type functional form, the firstly generated application cell is the root which needs all evaluated results of the other application cells generated afterword, because of the pipeline execution of linear recursive functions. Hence, the addresses of newly generated application cells must be pushed into the LIFO stack (reduction stack) one after another until the first step of reduction is completed.

In the case of the parallel type, on the other hand, newly generated application cells have no data dependencies among them. Therefore, the function applications represented by them can be executed immediately in the reduction engine. It is still necessary, however, to allocate new reduction stacks each time, in preparation for the next step of reductions. Consequently, the application cell management by the multiple reduction stacks linked in a tree structure can be very suitable for pipeline execution of reduction of all functional forms. Moreover, by using this control schem, the semantics of function–level computing has been naturally extended to include not only eager but also lazy evaluations.

Chapter 4 IMPLEMENTATION AND EVALUATION

The degree of the concurrency within a task (redex) depends on the granularity of each instance assigned to a different segment, as well as the depth of recursion. Therfore, it is required to obtain as much speedup in the pipeline as possible that each segment is assigned sufficient processes enough to fill up the pipeline, in addition to that the pipeline consists of many segments.

On the other hand, the latency for accessing structure memory will bring heavy degradation of throughput of pipelined reduction engine. The best method for solving the latency problem is time multiplexing of segments of the pipeline. That is, some tasks are issued to the pipeline concurrently, and whenever a memory access is required, the segment switches to the process of another ready task.

Hiding latency in this way requires high–level parallelism in a program. In FP programs, parallelism among tasks is dominated by the parallel functional forms:"eager composition", "construction", and "apply–to–all". The "eager composition" is for AND (stream) parallelism, and the rest for OR parallelism. The degrees of the AND/OR parallelism are characterized by the number of parallel tasks and the granularity of those.

In addition to the implementation of a prototype based on a tightly coupled multi–microprocessor system, the total performance of the reduction machine is evaluated via simulations on the register transfer level.

In general, OR parallelism is relatively easy to exploit in MIMD machines. As for the pipelined reduction machine to be an MIMD machines. As for the pipelined reduction machine to be an MISD architecture, however, the granularity of each OR parallel component strongly dominates the improvement of processing time. Therefore, to what extent each task in a given application program fills up the pipeline is a major and critical issue in the performance improvement of this machine.

In the case of AND parallelism, on the other hand, the overhead from the sequential task generation is hidden by the stream parallelism among the tasks. But, a larger object to be a stream will bring about the speedup saturation more easily. Because tasks with larger objects generate more requests for accessing the structure memory, which results in a longer waiting queue for the memory service. That is, too much parallelism causes the structure memory to be saturated, and prolongs the total processing time. Therefore, to achieve the maximum improvement on speedup in the system, we must keep the structure memory not to be saturated with so many memory access requests. It can be said that the power source of the pipelined reduction engine is the parallelism inherent in a task, and the parallelism among tasks, on the other hand, works as "lubrication oil" for the system.

Chapter 5 CONCLUSIONS

In conclusion, the MISD reduction machine based on the general-purpose pipeline is one of the most promising candidates for cost-effective function-level computing featured by crude granularity of parallelism.

Although the linear expansion of recursions is the principle of pipelining recursive functions, the general algorithm to generate a set of processes being carried out within the segments has not been brought to view yet. Like a conventional vector processors, an "intelligent compiler" will be needed to detect concurrency from the source FP programs given and to produce as large-grained primitive functions as possible.

Furthermore, a load balancing mechanism will make it possible that many inevitable fine-grained primitive functions unable to expand are processed throughout the pipeline segments efficiently. The intelligent compiler and dynamic load balancing will be the keys to further viability of the reduction machine based on a general-purpose pipeline.

審 査 結 果 の 要 旨

ソフトウェアの生産性向上と並列処理の可能性の点から，関数型言語ＦＰがJ. Backus により提案されているが，このプログラミング言語を効率よく処理するための計算機アーキテクチャが必要である。そのためのハードウェア機構として，汎用パイプラインが提案され，研究がなされている。しかし，ＦＰが効率よく実行されるためには，関数のもつ性質を十分に活用する計算モデルアーキテクチャの整合が必要であり，この問題について従来の研究では必ずしも十分なものとなっていない。特に，計算モデルの選択とアーキテクチャの調整が重要な研究課題である。著者は，この点に着目して，ＦＰによるプログラムの処理を行なう場合，汎用パイプラインがＦＰの特徴を十分吸収できるグラフ簡約機械として動作できることを理論的，及びシステム構成の立場から導いた。本論文は，これらの研究成果をまとめたもので，全文５章よりなる。

第１章は序論である。

第２章では，ＦＰの説明を行うに当り，その言語の持つ再帰的性質を積極的に抽出し，これを用いて再帰的関数の展開に汎用パイプライン処理方式を活用できることを示している。特に，線形再帰的関数が一般の再帰的関数の基本になるとの立場から，線形再帰的関数の処理をパイプライン上で効率よく行えることを示している。

第３章では，ＦＰの複雑なプログラムを処理する場合の実行制御機構を示している。データとしてのオブジェクトと共にＦＰプログラムの構造をグラフで表現し，これを計算モデルとしてグラフ簡約による処理を行っている。グラフ簡約機械は，グラフ簡約を終えると第２章で述べた再帰的関数の展開をパイプライン上で行っている。この方式は，ＦＰプログラムの実行を汎用パイプラインで行えることを示すもので，重要な知見である。

第４章はグラフ簡約機械の性能評価である。ここで簡約項の粒度に対応する簡約関数の再帰の深さとパイプラインのステージ数とが等しくなるとき，性能が最大となることを導いている。また，簡約項の処理の並列度により，性能がさらに向上することも示している。これは，実用上，有益な成果である。

第５章は結論である。

以上要するに本論文は，関数型言語ＦＰによるプログラムをグラフ簡約の手法を用いて，汎用パイプライン上で処理するための方法論について研究し，関数型言語プログラムの実行の効率を十分に高める方法を与えたもので，情報工学に寄与するところが少なくない。

よって，本論文は工学博士の学位論文として合格と認める。