Parallel Processing of Sequential Programs

Some Thoughts

Utpal Banerjee

University of California, Irvine

Given

- A program consisting of a set of operations with a fixed sequential order of execution
- A machine capable of executing multiple operations simultaneously in some fashion

Goal

- Transform the program in such a way that it runs on the given machine utilizing its resources to the fullest extent possible, while producing the same output as the original code.

Method

- Manual program transformation can be extremely time consuming and does not produce portable performance.
- Programmers should focus on implementation of correct algorithms and let the compiler transform the code to take advantage of parallel devices.

Status

- Despite many years of development, even automatic vectorization is only partially effective today.
- Today's compilers succeed only on about 30% of the loops from real applications that could be vectorized.

Belief

- We *can* improve the status of automatic parallelization if we put our mind to it.

Basic Steps

- Dependence Analysis
- Program Transformations

Dependence Analysis

- An operation q *depends* on an earlier operation p in the given program, if p *must* be executed before q to ensure that q uses or stores the right value.

- Find all dependence relations in the program.

- If it cannot be established that an operation q does NOT depend on an operation p, then it must be assumed that it does.

- Some dependences may be eliminated by various transformations of the code.

- The problem of finding dependences is well-understood when operations deal with array variables in loops, where the subscripts are linear functions of loop index variables, known at compile time.

- Since subscripts in real programs are usually simple, one often gets away with a simple algorithm for dependence testing.

- The ultimate weapon here is integer programming. If an algorithm claims to be accurate for the general linear case, it is probably a variation of a known integer programming algorithm in disguise.

- Dependence checking is hard when we deal with pointer variables and other complex data structures.

- Dependence analysis can be performed statically by a compiler or dynamically during program execution. The fusion and trade-offs between compiler and run-time approaches are not well understood.

Program Transformation

- A transformation of the program is *valid*, if the transformed program has the same output as the original code. The program after a valid transformation is *equivalent* to the original program.

- A transformation is valid if it respects all dependence relations.

- Change the given program by a sequence of valid transformations to a form that displays sets of mutually independent operations, such that each set can be executed by the given machine simultaneously.

- Try to maximize the utilization of machine resources.

- Several program transformations have been discovered over the years and many of them have been studied extensively.

- Although many transformations have been used by actual compilers, they are usually applied in an ad hoc way.

- It is not known what a necessary and sufficient set of transformations would be for a given type of parallelization.

- Cost models used by compilers to determine whether the transformed code runs faster than the original are often grossly inaccurate.

- User directives may guide or force the compiler to make the right decisions, but they can easily become counter productive.

- Although pattern matching of loops containing recurrences etc. is implemented in today's compilers, there is a need for more general algorithmic substitutions where the compiler recognizes the essence of a given algorithm as being embedded in a given code fragment.

Three Classes of Transformations

- Trace Scheduling
- Unimodular Transformations
- Echelon Transformations

Trace Scheduling

- Given: An acyclic graph of operation nodes (assignment and conditional branch operations)

- Finds: An equivalent acyclic graph of instruction nodes, where an instruction typically contains several operations that can be executed simultaneously.

Unimodular Matrix

- Square integer matrix with determinant 1 or -1
- An m X m unimodular matrix takes an m-vector with integer components into another m-vector with integer components.
- All identity matrices are unimodular.
- The product of two m X m unimodular matrices is an m X m unimodular matrix.
- The inverse of a unimodular matrix is also unimodular.

Unimodular Transformation

- Given: A perfect nest L of m sequential loops

- Finds: An equivalent perfect nest L’ of m loops, with the help of an m X m unimodular matrix U.

- A unimodular matrix U can always be found such that the outermost loop in L’ is sequential and all of the (m - 1) inner loops are parallel.

- In the worst possible case, each 'parallel' loop will have just one iteration.

- There are infinitely many unimodular matrices that will work. Ideally, one finds a matrix that minimizes the number of iterations of the outermost sequential loop.

- Based on the nature of dependence relations between iterations of the loop nest, there may exist a valid unimodular transformation L -> L' such that L' has some outermost parallel loops, one sequential loop, and then all parallel inner loops.

- Special Unimodular Transformations
    - Wavefront Method (just another name)
    - Loop Interchange
    - Loop Permutation
    - Loop Skewing

- Unimodular Transformations can be precisely described, and they can be easily combined. It is not clear how to pick the optimal ones and how to combine such transformations with more traditional ones for profit.

Echelon Transformation

- Given: A perfect nest L of m sequential loops

- Finds: An equivalent perfect nest L’ of (m + r) loops, where r is the rank of the distance matrix of L, such that the m outermost loops are parallel.

General Comment

- It is not clear how echelon transformations interact with other transformations.

Computer Science and Physics

- Work of M. H. Halstead at Purdue

- Papers dealing with race detection invoke the light cone of the Special Theory of Relativity and Heisenberg's Uncertainty Principle.

- Noether's Theorem

Emmy Noether

- Mathematician, born in Germany in 1882, died in US in 1935.

Noether's Theorem (stated for kids):

- In a physical system, a symmetry of some kind usually implies a law of conservation.

Examples

- The fact that the laws of physics do not vary with location in space translates into the law of conservation of linear momentum.

- The fact that the laws are invariant with respect to time gives the law of conservation of energy.

Heineken's Uncertainty Principle:

On a Sunday morning, it is impossible to remember exactly how many beers you had the night before.