Parallel Computation

Fatih Doğaç

June 2024

1 Abstraction

A **parallel computer** is one that can perform multiple operations simultaneously. Parallel computers may solve certain problems much faster than **sequential computers**, which can only do a single operation at a time.
But even the sequential ones are designed to use some parallelism as they execute individual instructions. So in this report, i am going to focus on massive parallelism whereby a huge number (think of millions or more) of processing elements are actively participating in a single computation .

2 Introduction

Parallel computation is an important area in computational theory dealing with the optimization of computations using parallelism. In other words, it studies how to obtain speedup in the performance of a program by executing processes concurrently. By breaking down complex problems into smaller, concurrently executable tasks, parallel computation aims to significantly reduce processing time and handle larger datasets more effectively.
The theoretical foundations of parallel computation are closely connected to complexity theory, which studies how hard different problems are to solve. In this context, there are important concepts like P vs. NC (Nick's Class) and P-complete problems.
For a brief explanation, P represents problems that can be solved quickly (in polynomial time) by a regular computer, while NC represents problems that can be solved quickly using many computers working together. We will talk about them later in the report.

3 Uniform Boolean Circuits

One of the most popular models in theoretical work on parallel algorithms is called the Parallel Random Access Machine or PRAM. In the PRAM model, idealized processors with a simple instruction set patterned on actual computers interact via a shared memory.

3.1 Boolean Circuits

Boolean circuits are networks composed of basic logical gates such as AND, OR, and NOT. These circuits process binary inputs (0s and 1s) to produce a desired binary output. They are used to solve computational problems by performing sequences of logical operations.
A family of Boolean circuits $C_n : n \in \mathbb{N}$ is polynomial-time uniform if there exists a deterministic Turing machine M, such that

- M runs in polynomial time
- for all $n \in \mathbb{N}$, M outputs a description of C_n on input 1^n

In the Boolean circuit model of a parallel computer, we take each gate to be an individual processor, so we define the **processor complexity** of a Boolean circuit to be its **size**.
We consider each processor to compute its function in a single time step, so we define the **parallel time complexity** of a Boolean circuit to be its **depth**, or the longest distance from an input variable to the output gate.
And we consider the simultaneous size and depth of a single circuit family in order to identify how many processors we need in order to achieve a particular parallel time complexity or vice versa.
Say that a language has simultaneous size–depth circuit complexity at most (f(n), g(n)) if a uniform circuit family exists for that language with size complexity f(n) and depth complexity g(n).

3.1.1 An example

Let A be the language over {0,1} consisting of all strings with an odd number of 1s.
We can test membership in A by computing the parity function. We can implement the two-input parity gate $x \oplus y$ with the standard AND, OR, and NOT operations as $(x \wedge \neg y) \vee (\neg x \wedge y)$ Let the inputs to the circuit be $x_1, ..., x_n$. One way to get a circuit for the parity function is to construct gates g_i whereby $g_1 = x_1$ and $g_i = x_i \oplus g_{i-1}$ for $i <= n$. This construction uses O(n) size and depth.
Parity function is a Boolean function whose value is one if and only if the input vector has an odd number of ones.

3.2 The Class NC

The complexity class NC (Nick's Class) plays a crucial role in the study of parallel computation within theoretical computer science. This class encapsulates problems that can be efficiently solved using parallel algorithms, highlighting their practical and theoretical significance. The class NC is the set of languages decidable in parallel time T(n, p(n)) = $O(log^{O(1)}n)$ with $p(n) = O(n^{O(1)})$ processors.

For i ¿= 1, let NC^i be the class of languages that can be decided by a uniform family of circuits with polynomial size and $O(log^i n)$ depth.
Let NC be the class of languages that are in NC^i for some i.
Functions that are computed by such circuit families are called NC^i **computable or NC computable.**

We explore the relationship of these complexity classes with other classes of languages we have encountered. First, we make a connection between Turing machine space and circuit depth. Problems that are solvable in logarithmic depth (it can be computed in a very shallow computation) are also solvable in **logarithmic space** (it can be computed using a very small amount of memory). Conversely, problems that are solvable in logarithmic space, even nondeterministically, are solvable in logarithmic squared depth.

Logarithmic squared depth refers to the depth of a circuit or computational process where the depth grows proportional to the square of the logarithm of the input size. For a circuit or computational process operating on input of size n, if the depth of the circuit is $O((log_n)^2)$ then it is said to have logarithmic squared depth.

3.3 P-Completeness

The class P is the set of all languages L that are decidable in polynomial time.

A language B is **P-complete** if

- $B \in P$, and
- every A in P is log space reducible to B.

A problem is P-complete if it is as "hard" as the hardest problems in the class P. If you can efficiently solve a P-complete problem, you can efficiently solve any problem in P.

4 Conclusion

In conclusion, the landscape of parallel computing is rich with opportunities and challenges, particularly in the context of P-completeness and NC complexity classes. As evidenced by this report, the theoretical foundations of parallel algorithms offer insights into the inherent difficulty of solving problems efficiently in parallel, as well as strategies for navigating these complexities. Understanding P-completeness provides a lens through which we can gauge the inherent difficulty of problems in parallel settings, while NC algorithms offer practical approaches for designing efficient parallel solutions.

5 References

- https://en.wikipedia.org/wiki/P-complete
- Introduction to the theory of computation by Michael Sipser 3rd edition
- https://athena.nitc.ac.in/ kmurali/complex/PCompleteBook.pdf