Optimal Speedup on a Low-Degree Multi-Core Parallel Architecture (LoPRAM)

Reza Dorrigiv, Alejandro López-Ortiz, Alejandro Salinger
David R. Cheriton School of Computer Science, University of Waterloo

Over the last five years, microprocessor manufacturers have released plans for a rapidly increasing number of cores per microprocessor, with upwards of 64 or 128 cores by 2015. In this setting, a sequential RAM computer no longer accurately reflects the architecture on which algorithms are being executed.

We propose a new model of low degree parallelism (LoPRAM) which better reflects recent multicore architectural advances.

We argue that the number of processors available is best described in terms of degree rather than the implicit \( O(n) \) of the classic PRAM model.

The LoPRAM model supports a high level of abstraction that simplifies the design and analysis of parallel programs.

We show that in many instances our model naturally leads to work-optimal parallel algorithms via simple modifications to sequential algorithms. In contrast, in the PRAM model the design, analysis and implementation of work-optimal algorithms for \( O(n) \) processors proved to be one of the biggest challenges in practice for their adoption.

Parallel Models of Computation

The dominant model for past theoretical research on parallel computation is the PRAM, which generally assumed \( O(n) \) processors working synchronously with zero communication delay and infinite bandwidth among them. If the number of processors available in practice was smaller, the \( O(n) \) processor solution could be emulated using Brent’s Lemma.

The PRAM, while fruitful from a theoretical perspective, proved unrealistic and various attempts were made to refine it in a way that would better align to what could effectively be achieved in practice.

Among the practical drawbacks of the PRAM are the cost of synchronization; the cost of interprocessor communication; the cost-effectiveness of a massively parallel machine; and the enormous difficulty in developing and implementing work-optimal algorithms (i.e. linear speedup) for \( O(n) \) processors.

Recent developments in multicore architectures have brought back the possibility of parallel architectures in practice which has revived the study of parallel algorithms.

To the best of our knowledge the assumption of a logarithmic level of parallelism as well as its theoretical implications had yet to be noted in the literature.

The LoPRAM Model

The core of a LoPRAM is a PRAM with \( p=O(\log n) \) processors running in Multiple-Instruction Multiple-Data (MIMD) mode.

The read and write model is generally assumed to be Concurrent-Read Exclusively-Write (CREW).

To support this model, semaphores and automatic serialization on shared variables are available, in a transparent form to the programmer.

Thread Model

Parallelism is specified through the use of threads. The model provides standard threads and PAL-threads (Parallel ALgorithmic threads).

PAL-threads are executed at a rate determined by the scheduler.

• If there are any PAL-threads pending, at least one of them must be actively executing.

• Pending PAL-threads are activated in a manner consistent with order of creation as resources become available, in a fashion reminiscent of work stealing.

• Once a thread has been activated, it remains active just like a standard thread (to avoid potential deadlock).

The scheduler inserts thread requests into an ordered tree. Threads are scheduled in a combination of parallel breadth-first and depth-first order. When a thread issues calls for its children, it enters a wait state and children threads are executed by available cores, in order of creation. When all available cores are executing a thread, each core executes the subtree rooted at its thread in depth-first order. If cores become available again, threads are assigned to them in the order given by the preorder traversal of the tree.

For example, consider a parallel implementation of Megersort with suitable C extensions for the LoPRAM:

```c
void merge_sort2(int *data, int i, int j, int m, int n, int *tmp)
{
    int l1 = i, l2 = m + 1, l3 = i, l4 = m + 1;
    for (int k = i; k <= m; k++)
    {
        if (data[k] < tmp[l2])
        {
            l3 = (l3 < l1) ? l3 : l1;
            l4 = (l4 < l2) ? l4 : l2;
        }
        else
        {
            l3 = (l3 < l1) ? l3 : l1;
            l4 = (l4 < l2) ? l4 : l2;
        }
    }
    for (int k = i; k <= m; k++)
    {
        if (data[k] < tmp[l2])
        {
            data[l3] = tmp[l2++], l3++;
        }
        else
        {
            data[l4] = tmp[l2++], l4++;
        }
    }
}
```

The LoPRAM model supports a high level of abstraction that simplifies the design and analysis of parallel programs. In contrast, in the PRAM model the design, analysis and implementation of work-optimal algorithms for \( O(n) \) processors proved to be one of the biggest challenges in practice for their adoption.

We argue that the number of processors available is best described in terms of degree rather than the implicit \( O(n) \) of the classic PRAM model.

The LoPRAM model supports a high level of abstraction that simplifies the design and analysis of parallel programs. In contrast, in the PRAM model the design, analysis and implementation of work-optimal algorithms for \( O(n) \) processors proved to be one of the biggest challenges in practice for their adoption.

We introduce a new model for parallel computation that is faithful to current architectures, avoids many of the pitfalls of the PRAM model, and allows for significant classes of problems to be parallelized with little effort.

We provide the first formal argument for the observation that it is easier to develop work optimal parallel programs when the degree of parallelism is small.

The main contribution of this work is the combination of a series of established facts, as well as new observations and lemmas into a novel model which is simple, effective and better reflects the state of current parallelism.

Contact information

Alejandro Salinger
ajsalinger@uwaterloo.ca
David R. Cheriton School of Computer Science
University of Waterloo

Work-Optimal Algorithms

We show that Divide-and-Conquer and Dynamic Programming allow for ready parallelization under the LoPRAM model.

Divide-and-Conquer

Consider a recursive divide-and-conquer sequential algorithm whose time complexity is given by:

\[
T(n) = aT(n/b) + f(n)
\]

By the master theorem, \( T(n) \) is such that:

- if \( f(n) = O(n^{\log_b a}) \)
- if \( f(n) = \Theta(n^{\log_b a}) \)
- if \( f(n) = \Omega(n^{\log_b a}) \) but \( f(n) \leq n^{\log_b a} \cdot \log^k(n) \) for some \( k \leq 0 \)

In a parallel execution of the algorithm, recursive calls are handled in a different way. At this point, each core solves a subproblem of size \( n \log \log n \). Let the time complexity of the subproblem be \( T(d) \).

The parallel time can be written as:

\[
T_{p}(d) = T\left(\frac{n}{p}\right) + \sum_{i=0}^{\log_{b}p} f\left(\frac{n}{b^{i}}\right)
\]

We first consider algorithms whose merging phase is strictly sequential. We obtain the following master theorem for parallel divide and conquer algorithms in the LoPRAM:

\[
T_{\text{parallel}}(n) = T(n) + O(n) \log(n)
\]

If the merge step can be parallelized the third case becomes \( T_{p}(n) = O(n/p) \), i.e. optimal speedup, as long as \( p = O(\log n) \). This implies that optimal parallel algorithms can be readily derived for important divide and conquer algorithms such as Megersort, Matrix multiplication, Delaunay triangulation, Polygon triangulation and Convex hull, among others.

Dynamic Programming

We seek a general procedure such that given the DP specification of a problem, it generates a scheduling strategy to schedule it in parallel. Let the specification be:

\[
M[x] = \begin{cases} f(x) & \text{if } g(x) = 0 \\ \ell(M[x_{x \times y}]) & \text{otherwise} \end{cases}
\]

The dependency graph is a DAG, where vertices correspond to subproblems and edges indicate dependencies between subproblems. The goal is to compute this DAG in parallel.

Conclusions