Parallel Unification: Practical Complexity

David Powers http://www.cs.flinders.edu.au/people/DMWPowers.html
Department of Computer Science, David.Powers@flinders.edu.au
The Flinders University of South Australia powers@acm.org

Parallel Unification

The research discussed in this presentation arises in the context of Parallel Theorem Proving and Logic Programming. These paradigms amount to searching a tree in which a path from root to leaf represents a set of choices (implemented as unification) identifying a potential solution.

Examining alternative paths in parallel exploits so-called *OR-parallelism* whilst searching at different levels in parallel amounts to *AND-parallelism.* In addition, it is possible to parallelize the unification process, or a lazy evaluation involving delayed unification or bookkeeping [8].

Interestingly, unification has proven a problem both for sequential and parallel implementation. The standard unification algorithms used in Prolog systems are not sound, and the straightforward fixes lead to exponential execution time. There are, however, linear and near-linear time sound sequential algorithms. Unfortunately, even the unsound algorithms don't parallelize well, and exhibit worst case linear performance.

This has lead to parallelization of unification being eschewed, although the pathological cases are rare. But unification has usefully been parallelized [1] and the pathological cases have been characterized [8].

Parallel Hardware

Theoretical characterization of the complexity of parallel unification and reasoning systems rends to rely implicitly or explicitly on a PRAM model. The quicksort algorithm is a standard Prolog demonstration with expected $N \log N$ sequential performance. A simulated parallel reasoning system gave $\log N$ performance with its implicit PRAM model and a PRAM algorithm was derived from it [9]. Although there are now several such $O(\log N)$ algorithms around (and the derived QuickSort and RadixSort algorithms have amongst the lowest constants), there are actually $O(\log^2 N)$ algorithms which will execute faster in any achievable implementation! Ironically, there are also O(1) algorithms [2]which are even worse!! We ignore these.

The paper will discuss what these orders really mean and the factors which are hidden by the current models of parallelism. Indeed, the serial characterizations also hide many constants and are not truly scalable either! We characterize algorithms in terms of the memory bandwidth or buswidth, *w,* and note that a general sorting task involving N items will scale only while $w >= \log N$ — with *PRAM* **or** *Sequential RAM.* With a constant w we can achieve $O(wN)$ bit-level cost giving $O(w\log^2 N)$ bit-level delay [3]. Note that any parallel architecture whose connectivity and bandwidth doesn't scale as $\log N$ will incur multiplicative factor(s) of $\log N$.

Indeed, it has long been observed that sorting networks provide the most efficient known solution to the PRAM's arbitrary permutation problem [3], avoiding the typical probabilistic routing or blocking behaviour [4,6]. Furthermore, with Batcher's bitonic sort an $O(\log^2 N)$ bit-level latency may be realized (viz. measured in terms of gate-delays). Although there are claims that $o(N\log^2 N)$ hardware complexity can be reduced to $o(N\log N)$, these assume a constant bandwidth $w >= \log N$ [7], and thus hide another factor of $\log N$ and don't change the order of the gate count. Suprisingly, *parallel comparisons* actually *slow* bitonic sort by $\log N$ in terms of gate-delays, as they can be pipelined in a *serial* implementation.

Considering the limiting speed of light and the constant size of processing elements, path-lengths of $o(N^{1/3})$ provide a hard physical limitation, which is significant once propagation times and processor clock cycles are of the same order, such as when 300Mhz processors communicate at 300Mm/s in a 1m cube. Gate fanins and fanouts which are functions of N [5] should be viewed with suspicion, and a sound definition of bit-level cost and delay makes them constant. But *sequential* machines face these limits too...

References

1. J Barklund (90), *Parallel Unification,* **PhD Thesis,** UPMAIL, Uppsala University, Sweden
2. Y-C Chen & Q-T Chen (94), *Constant Time Sorting on Reconfigurable Meshes*, **IEEE-C 43**:6 749-751
3. MV Chien & AY Oruç (94), *Adaptive Binary Sorting Schemes and Associated Interconnection Networks,***IEEE-PDS 5**:6 561-572
4. SA Felperin, L Gravano, GD Pifarré & JLC Sanz (91), *Routing Techniques for Massively Parallel Communication,* **Proc. IEEE 79**:488-489
5. MJ Forsell (94), *Are multiport memories physically feasible?,* **CAN 22**:4 47-54
6. J Kimm & CR Das (94), *Hypercube Communication Delay with Wormhole Routing*, **IEEE-C 43**:7 806-814
7. S Lee & M Lu (94), *New Self-Routing Permutation Networks,* **IEEE-C 43:**11 1319-1323
8. DMW Powers (91), *Parallel and Efficient Implementation of the Compartmentalized Connection Graph Proof Procedure*, in B. Fronhoefer & G. Wrightson (eds), **Parallelism in Inference Systems**, **LNAI 50**, Springer-Verlag.
9. DMW Powers (91), *Parallelized QuickSort and RadixSort with Optimal Speedup*, **Procedings of the Parallel Computing Technologies Conference (PACT-91)**, Novosobirsk, World Scientific.