Chapter 6

Conclusions

The first implementation of a parallel *Chinook* demonstrated how closely the game tree branching factor is linked with the speedups that can be obtained with tree decomposition algorithms, such as Principal Variation Splitting (PVSplit). The narrow checkers game trees have a branching factor that is over an order of magnitude smaller than in chess game trees. With a branching factor of 2.84, the degree of parallelism offered by checkers game trees is quickly overloaded by even a moderate number of parallel searchers. For 16 searchers, there are over 5 times as many searchers as there are units of parallel work available at a split node. Consequently, PVSplit suffers from starvation.

To reduce the amount of starvation, Principal Variation Frontier Splitting (PVFSplit) was developed. PVFSplit attempts to create more parallel work by allowing each node on the current variation to be a split node. Experiments with PVSplit and PVFSplit on a test suite of positions, using a BBN TC2000 shared memory parallel computer, show that PVFSplit achieves 52% better speedups than PVSplit (with 16 searchers) by lowering the idle time associated with starvation. Additional support for load balancing through straggler preemption further improves the speedup to a high of 3.32 with 16 searchers. In absolute terms, the speedups achieved with PVFSplit are low.

The experience of developing *ParaChinook* has resulted in three main observations. First, a relatively high branching factor in the application domain cannot be taken for granted. One can choose the application, but one usually cannot choose the properties of that domain. Any discussion of parallel performance must be put in the context of the natural degree of parallelism of the application domain. To present speedups without mentioning the branching factor is to present a partial picture. Second, it is important to note how many of the subtrees rooted at a node require a substantial amount of effort to search. Some of the branches may represent poor move choices that lead to quick refutations. These subtrees are considerably smaller than the other more viable subtrees, and thus contain less work. It is speculated that for checkers, there are about 2 viable moves per node on average, even if the actual branching

factor is, say, 8. A high branching factor does not guarantee large amounts of parallel work. Third, any PVSplit-based will suffer from the same trade-off between starvation and search overhead due to the branch-and-bound nature of Alpha-Beta search. Furthermore, there are a limited number of split nodes available in a single variation, therefore a future version of *ParaChinook* may have to use a non-depth first search strategy to find enough split nodes to get acceptable speedups.

6.1 A Future *ParaChinook*: Ideas and a Paper Design

Experimental results show that the current version of *ParaChinook* cannot profitably use more than about 8 searchers before the speedup curve begins to level off. Given the trend towards more processors in parallel computers, *ParaChinook* has to be updated too. Even with PVFSplit, the main reason for poor speedups is searcher starvation.

However, PVFSplit should not be abandoned altogether. On the one hand, the depth first search nature of PVFSplit is appealing. Its single controller is easier to debug than a fully distributed control structure and the low control-related storage requirement[1] is an important advantage when the total node count of the game trees is in the millions. On the other hand, a single variation splitting framework cannot provide enough split nodes to eliminate starvation when the trees are narrow. A compromise on pure depth first search *may* be sufficient to achieve acceptable performance.

The storage overhead of best first search and other non-depth first search strategies can be reduced if only the top few plies of the search tree are expanded in this manner. Recall that with the DBTE algorithms, the nodes of top plies of the game tree are queued in memory and are given to a pool of parallel searchers. A similar processor model can be adopted within *ParaChinook*.

The individual searchers can be clustered into groups of 4 or 8 with a single controller per cluster. The main *ParaChinook* program builds the upper plies of the search tree in memory in a non-depth first manner so that many split nodes are generated. A subtree is given to a cluster of searchers who search it to the nominal depth using PVFSplit. Because the subtrees given to the clusters of searchers are near the root of the tree, they are coarse-grained units of work, which will result in better parallel performance. Because the clusters of searchers build

---

[1] Of course, data structures such as transposition tables do require large amounts of storage. However, they are supporting data structures and are not essential to the control aspects of the tree search.

the last several plies of the tree depth first, the low control-storage requirements of the current *ParaChinook* is largely maintained. Furthermore, clustering the searchers reduces the number of entities contending for a concurrent data structure, such as a work queue. In effect, a cluster of searchers executing PVFSplit becomes a parallel static evaluation function from the point of view of the main *ParaChinook*.

The hybrid design may not be the best choice for a new *ParaChinook*, but it is clear that small changes to the basic PVSplit algorithm, in the same vein as PVFSplit, will be insufficient to improve the speedups on narrow game trees. However, PVFSplit with a small number of searchers can be an useful building block for other parallel search algorithms.

6.2 Final Comment: Quality Versus Quantity

The method of quantifying performance in this thesis has been the parallel speedup. As already touched upon, speedup has both advantages and disadvantages as a performance metric. It is simple to understand, but its simplicity does not capture other important aspects of performance.

For example, although a speedup of 3.32 with 16 searchers is low in absolute terms, it does translate to over 1.5 extra plies of search depth in the same amount of time. Furthermore, since the branching factor is 2.84, the speedup value is greater than the theoretical ideal speedup with PVSplit: speedup equal to the branching factor.

The ideal performance metric would be the improvement in the quality of move choice that comes with parallelism. Since the goal is to improve the game playing program's strength, the decision quality should be the bottom line. Towards that, the potential of solving the game of checkers and creating a perfect checkers oracle is intriguing. Using the endgame databases, a future version of *Chinook* may be able to judge the quality improvement that results from parallelism. However, that remains a future research project.