LESS: Loop Nest Execution Strategies for Spatial Architectures

Amalee Wilson
University of Alabama at Birmingham
amalee@uab.edu

Swapna Raj
Intel
swapna.raj@intel.com

Kermin Fleming
Intel
kermin.fleming@intel.com

1 INTRODUCTION

Conventional architectures are challenged to reach exascale power and performance levels, but if properly utilized, reconfigurable spatial architectures, e.g. FPGAs, can be part of the solution to this issue. However, such hardware is useful only if they can be programmed effectively, preferably with minimal overhead for domain experts. Thus, tools which advise users on how to exploit new architectures must be made. LESS aims to fill a critical gap in the software development process, by using data from other profiling tools to suggest optimized offloading strategies for existing programs. By automatically analyzing these results, the Loop Nest Execution Strategy for Spatial Architectures tool (LESS) recommends an optimized execution strategy, which considers both relative performance to overall program execution and the performance-area tradeoff of implementing a loop on the spatial architecture.

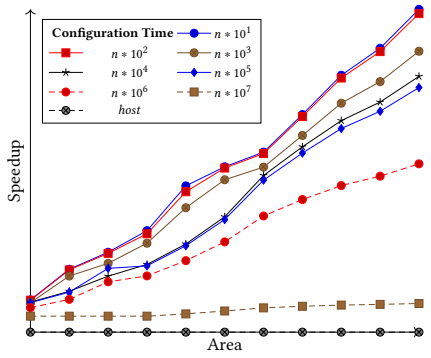

Figure 1: Estimated speedup of the Nekbone workload on the accelerator vs host, plotted with varied area and configuration times.

2 ALGORITHM

LESS views program execution as a hierarchy of nested loops. Our approach uses dynamic programming by finding the best offloading strategy for each outer loop nest in the program, recording this result, and combining these approaches to arrive at an overall execution strategy for the program. Within the spatial architecture, parallelism is obtained by implementing multiple instances of loops within the hierarchy, with the assumption that these copies will execute in parallel. Spatial architectures may also be reconfigured, allowing loop nests to be separately implemented, which can potentially increase acceleration. LESS attempts to balance the resources assigned to each loop, that is the number of loop bodies implemented, according to the relative importance of the loop, its implementation cost, and the cost to reconfigure the resource.

When analyzing each outermost loop nest, LESS makes two decisions: 1. Whether to offload child loops nests independently or as part of their parents, and 2. How to offload the loops that are chosen to be accelerated. These choices are made at each level of nesting to produce an offloading strategy for each loop nest. From this combination of approaches, a whole-program implementation is produced, and an estimation of the speedup when using this strategy is provided to the user. Although the search space for an optimal implementation is larger, LESS prunes this space with its dynamic programming inspired approach.

2.1 Offloading Child Loop Nests

When deciding whether to offload child loop nests independently, LESS finds the best offloading strategy for a whole loop implementation that includes the parent loop and its child loop nests. After estimating the performance of this whole loop implementation, LESS then determines the best offloading strategy for each child loop nest and estimates the performance of offloading the child loops independently. For each of these two strategies, the configuration times are taken into account, with the total time spent on configuration typically higher for the second strategy. LESS then chooses the best of each of these approaches, with the aim of minimizing total time spent executing the loop.

2.2 Choosing the Best Offloading Strategy

As mentioned in the previous section, LESS must choose the best offloading strategy for both the whole loop implementation and the independent child loop implementations. To do this we begin by unrolling the loop nest such that the compute fabric is exhausted. Then the marginal value of each loop is calculated by estimating the resultant execution time of the loop nest when a copy of loop is removed from the resource. This difference is then divided by the estimated area of the loop. Then a copy of the loop with the smallest marginal value is removed, and the process is repeated until the loop nest conforms to the area constraint of the resource. This approach was chosen because it allows a wider exploration of the space of possibilities of offloading strategies.

3 RESULTS

Preliminary results show that our approach may offer better execution strategies for loop nests with large child loop nests, which can benefit from being offloading independently. Figure 1 shows the relationship between area and speedup, as well as the relationship between configuration time and speedup. In this graph it is clear that the cost of total configuration time for this workload, even with multiple configurations, is offset by the speedup gained by the approach. In fact, as the cost of configuration grows, it must become relatively high before there is a large decrease in performance. Some experiments have been conducted on different workloads, but these will be explored more thoroughly later.