Superword Level Parallelism

- While different processors vary in the type and number of multimedia instructions offered, at the core of each is a set of short SIMD operations. These instructions operate concurrently on data that are packed in a single register or memory location.

- Complicated loop transformation techniques such as loop fission and scalar expansion are required to parallelize loops that are only partially vectorizable. Consequently, no commercial compiler currently implements this functionality.

- In the same way that loop unrolling translates loop level parallelism into ILP, vector parallelism can be transformed into SLP. This realizaiton allows for the parallelization of vectorizable loops using the same basic block analysis. As a result, our algorithm does not require any of the complicated loop transformations typically associated with vectorization.

- The core of our algorithm begins by locating statements with adjacent memory references and packing them into groups of size two. From this initial seed, more groups are discovered based on the active set of packed data.

- All groups are then merged into larger clusters of a size *consistent with the superword datapath width*.

- Finally, a new schedule is produced for each basic block, where groups of packed statements are replaced with SIMD instructions.

1 Loop Unrolling

- In order to ensure full utilization of the superword datapath in the presence of a vectorizable loop, the unroll factor must be customized to the data sizes used within the loop. For example, a vectorizable loop containing 16-bit values should be unrolled 8 times for a 128-bit datapath.

- Our system currently unrolls loops based on the smallest data type present.

2 Alignment Analysis

- Alignment analysis determines the alignment of memory accesses with respect to a certain superword datapath width. For architectures that do not support unaligned memory accesses, alignment analysis can greatly improve the performance of our system.

3 Pre-optimization

- This ensures that parallelism is not extracted from computation that would otherwise be eliminated.

- Optimization include constant propagation, copy propagation, dead code elimination, common subexpression elimination, loop-invariant code motion, and redundant load/store elimination.

4 Identifying Adjacent Memory References

- Statements containing adjacent memory references are the first candidates for packing. Adjacency is determined using both alignment information and array analysis.

- When located, the first occurrence of each pair is added to the PackSet.

  Definition 1. A *Pack* is an n-tuple, where elements are independent isomorphic statements in a basic blocks.

  Definition 2. A *PackSet* is a set of *Packs*.

  Definition 3. A *Pair* is a *Pack* of size two, where the first statement is considered the left element, and the second statement is considered the right element.

- As an intermediate step, a statement us allowed to belong to two Pairs as long as it is the left element in one Pair and right in the other Pair. This allows the Combination phase to easily merge Pairs into larger Packs.

5 Extending the PackSet

- New candidates can either:
  - Produce needed source operands in packed form, or
  - Use existing packed data as source operands.
- For two statements to be packable, they must meet the following criteria:
  - The statements are isomorphic.
  - The statements are independent.
  - The left statement is not already packed in a left position.
  - The right statement is not already packed in a right position.
  - Alignment information is consistent.
  - Execution time of the new parallel operation is estimated to be less than the sequential version.

6 Combination

- Two groups can be combined when the left statement of one is the same as the right statement of the other.

7 Scheduling

- Dependence analysis before packing ensures that statements within a group can be executed safely in parallel. However, it may be the case that executing two groups produces a dependence violation.
- The scheduling phase begins by scheduling statements based on their order in the original basic block.
- If scheduling is ever inhibited by the presence of a cycle, the group containing the earliest unscheduled statement is split apart.