intel® Look Inside™

Extending LoopVectorizer:

OpenMP4.5 SIMD and Outer Loop Auto-Vectorization

Vectorizer Team **(presenter: Hideki Saito)**
Intel Corporation

LLVM Developer Conference 2016/11/03

Legal Disclaimers

- Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.
- Relative performance is calculated by assigning a baseline value of 1.0 to one benchmark result, and then dividing the actual benchmark result for the baseline platform into each of the specific benchmark results of each of the other platforms, and assigning them a relative performance number that correlates with the performance improvements reported.
- Intel does not control or audit the design or implementation of third party benchmarks or Web sites referenced in this document. Intel encourages all of its customers to visit the referenced Web sites or others where similar performance benchmarks are reported and confirm whether the referenced benchmarks are accurate and reflect performance of systems available for purchase.
- Intel® Hyper-Threading Technology Available on select Intel® Xeon® processors. Requires an Intel® HT Technology-enabled system. Consult your PC manufacturer. Performance will vary depending on the specific hardware and software used. For more information including details on which processors support HT Technology, visit http://www.intel.com/info/hyperthreading.
- Intel® Turbo Boost Technology requires a Platform with a processor with Intel Turbo Boost Technology capability. Intel Turbo Boost Technology performance varies depending on hardware, software and overall system configuration. Check with your platform manufacturer on whether your system delivers Intel Turbo Boost Technology. For more information, see http://www.intel.com/technology/turboboost
- Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor series, not across different processor sequences. See http://www.intel.com/products/processor_number for details. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications. All dates and products specified are for planning purposes only and are subject to change without notice
- Intel product plans in this presentation do not constitute Intel plan of record product roadmaps. Please contact your Intel representative to obtain Intel's current plan of record product roadmaps. Product plans, dates, and specifications are preliminary and subject to change without notice
- Copyright © 2014 Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon and Xeon logo , Xeon Phi and Xeon Phi logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries. All dates and products specified are for planning purposes only and are subject to change without notice.
- *Other names and brands may be claimed as the property of others.

Optimization Notice

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

Key Takeaways

1. We plan to teach LLVM to vectorize Outer-loops
   a. Explicitly via OpenMP4.5 and also automatically.
   b. Use it for vectorizing Functions
   c. Design also for Loop+SLP vectorization, Outer+Inner Loop vectorization, and more
2. Do so by extending LLVM's existing Innermost Loop Vectorizer
   a. Leveraging past and ongoing efforts as well as future maintenance
   b. Introducing Vectorization Plan to model (multiple) potential candidates
3. This long-term challenge just started; Collaboration welcome!

Related Talks

- Scalable Vectorization (Day 1, 11:15-noon) by Graham Hunter and Amara Emerson
  - Vectorizer's output for SVE arch
  - Predication
  - Gather/scatter
- Representing composite SIMD operations in LLVM-IR (Day 1, 5-5:45pm, BoF) by Elena Demikhovsky
  - Vectorizer's output
- RV: A Unified Region Vectorizer for LLVM (Day2, 3:45-4:45pm, Poster) by Simon Moll
  - Vectorizer as a building block

Should be complementary to each other
and can co-exist/collaborate with this work.

Objectives/Agenda

- Goal Setting for Multi-Year Project
- Rationale
- Compare and Contrast
- Execution Plan
- Call for Participation

Background and Goal Setting

We'd like to build up incrementally,
but towards an ambitious goal.

Today's LLVM Vectorizers

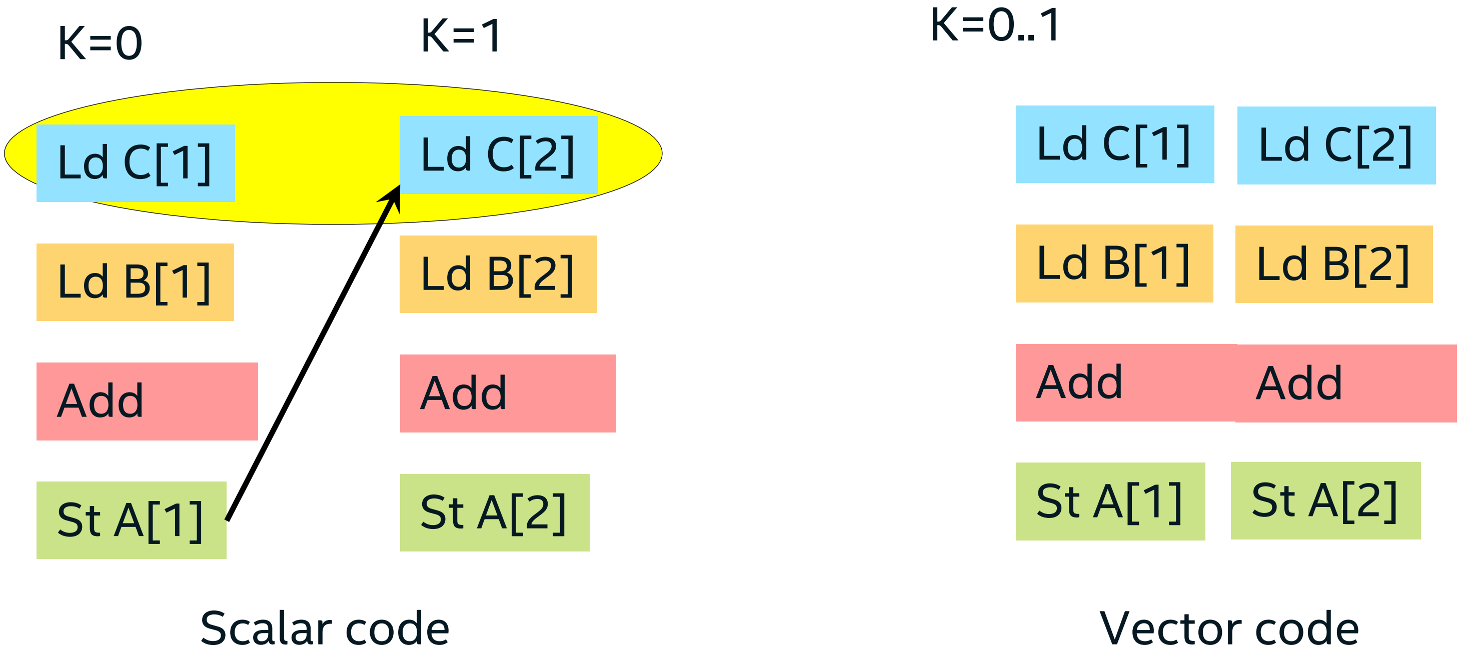
- LoopVectorize for innermost loop vectorization
- SLPVectorize for "unrolled" (or similar) code
- LoadStoreVectorizer for GPU memrefs
- BBVectorize (replaced by SLPVectorize)

- Various (non-)proprietary OpenCL implementations
- RV/WfV projects (U-Saarland)
- …

We fully understand how this happens, but this isn't necessarily ideal.

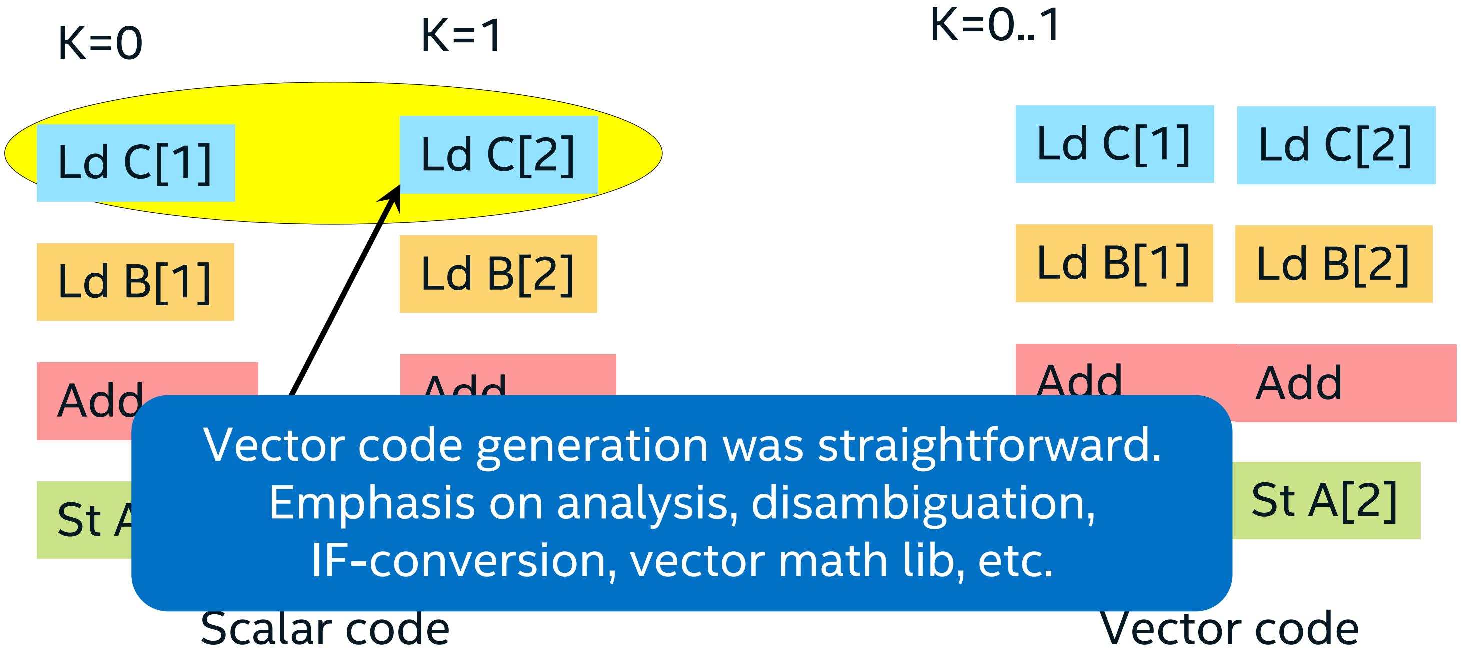
Vectorization Yesterday

```
for (k=0; k<N; k++)
  A[k] = B[k] + C[k];
```



Vectorization Yesterday

```
        for (k=0; k<N; k++)
          A[k] = B[k] + C[k];
```



K=0

Ld C[1]

Ld B[1]

Add

St A

K=1

Ld C[2]

Ld B[2]

Add

K=0..1

Ld C[1] Ld C[2]

Ld B[1] Ld B[2]

Add Add

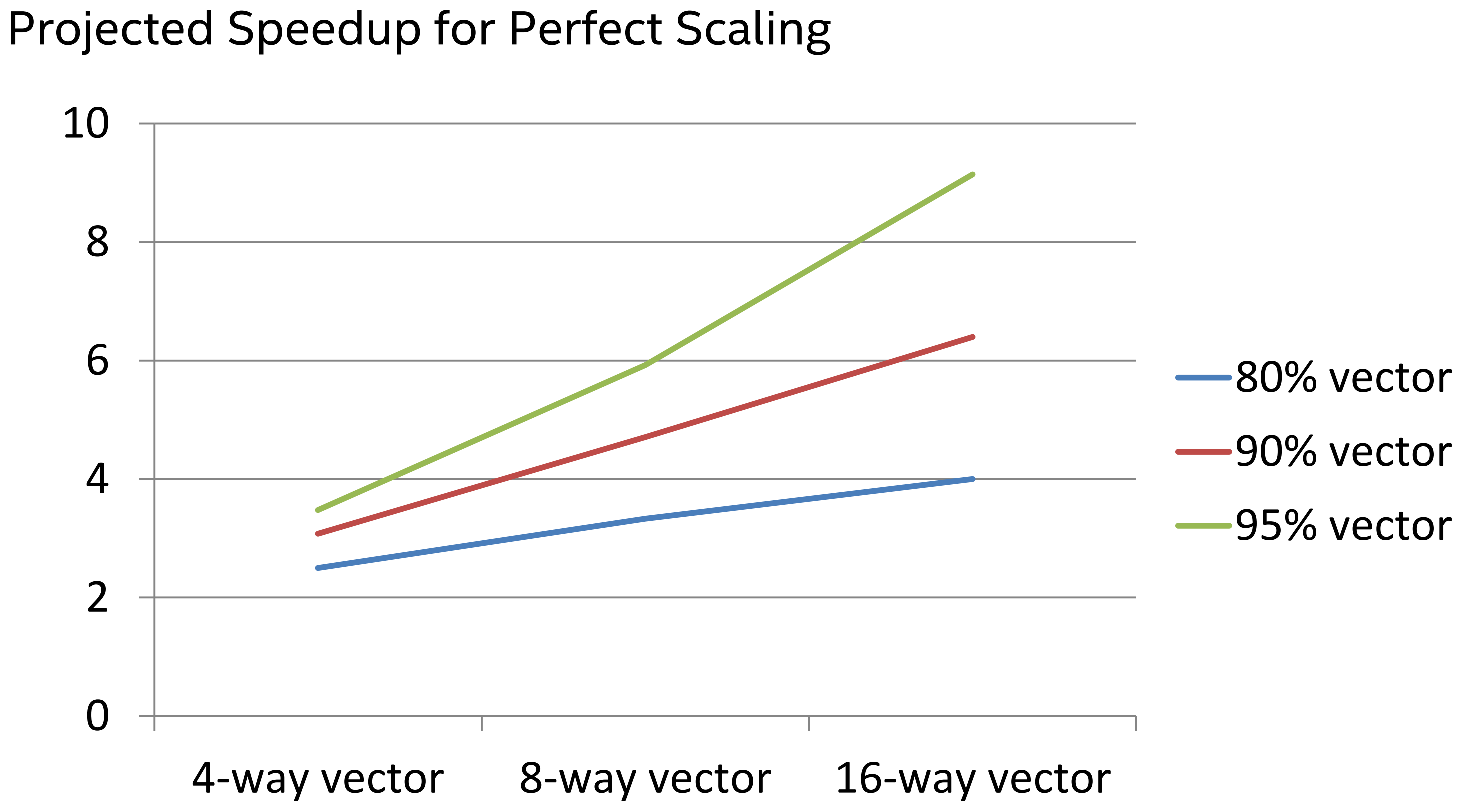
St A[2]

Scalar code

Vector code

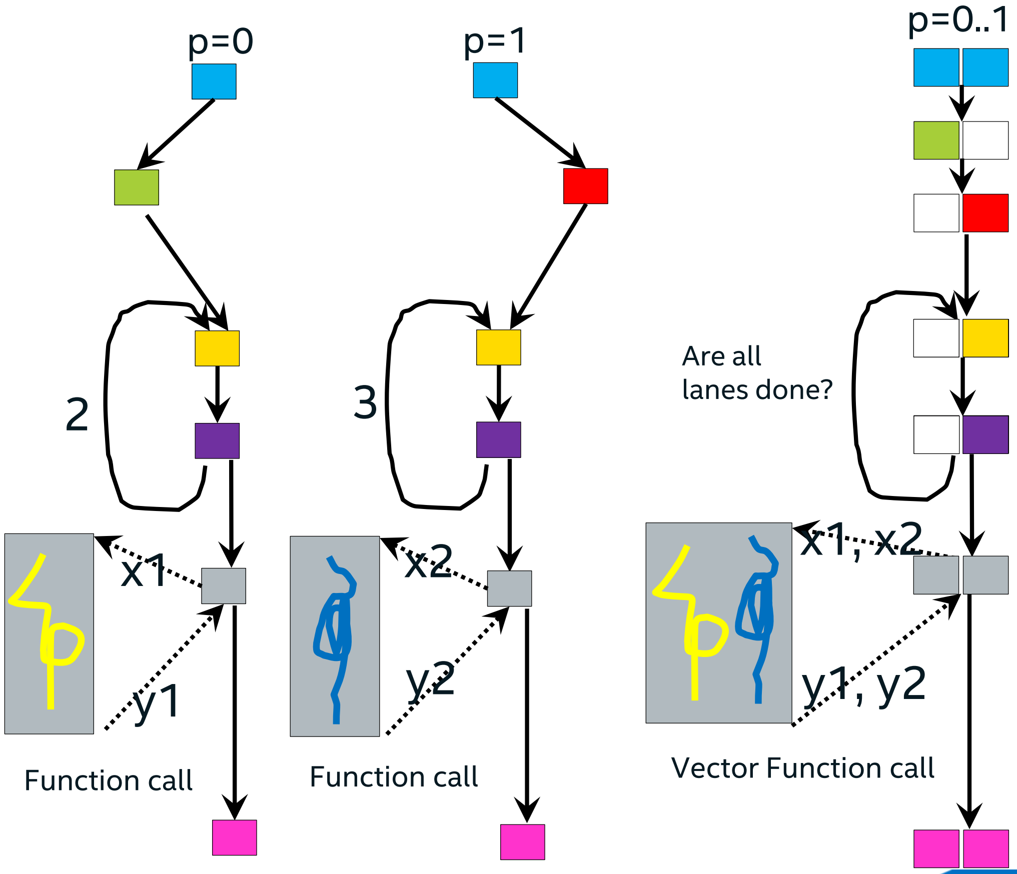
Vector code generation was straightforward.
Emphasis on analysis, disambiguation,
IF-conversion, vector math lib, etc.

Need to Vectorize More w/ Longer Vector



Vectorization Today

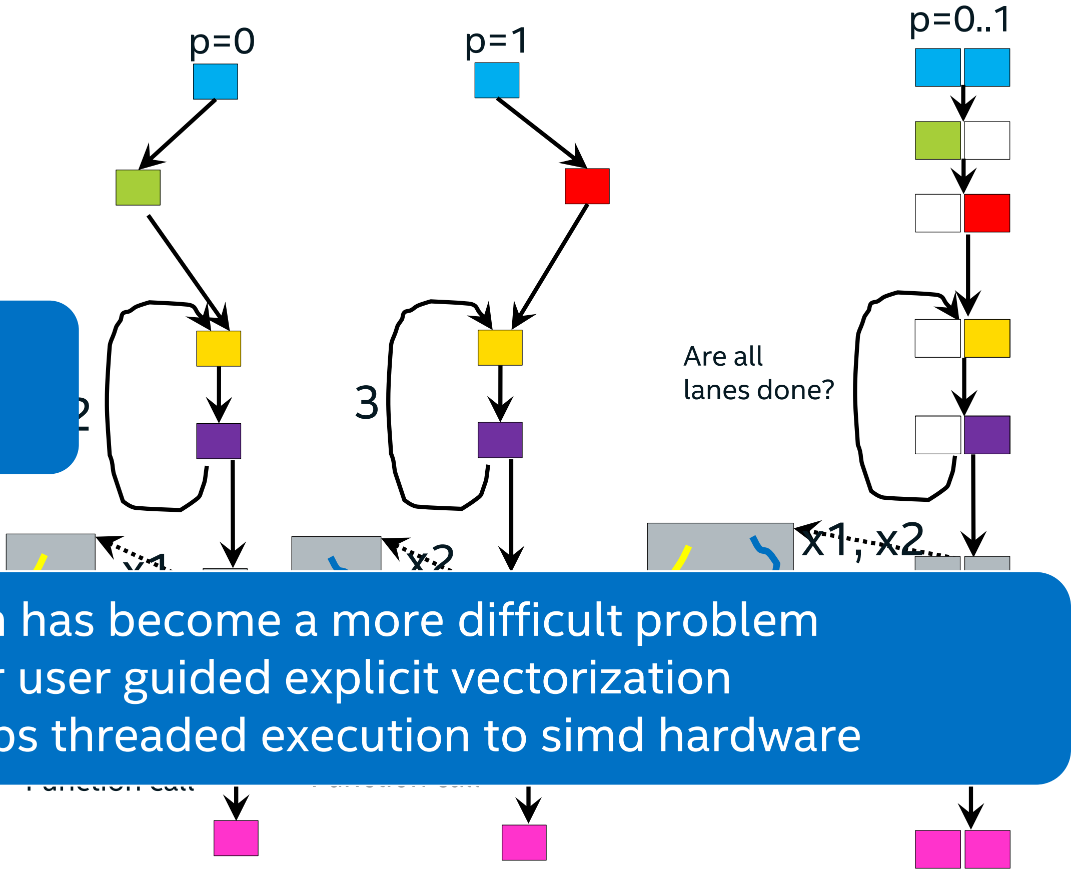
```
#pragma omp simd reduction(+:....)
for(p=0; p<N; p++) {
   // Blue work
   if(...) {
      // Green work
   } else {
      // Red work
   }
   while(...) {
      // Gold work
      // Purple work
   }
   y = foo (x);
   Pink work
}
```



Vectorization Today

```
#pragma omp simd reduction(+:....)
for(p=0; p<N; p++) {
   // Blue work
   if(...) {
      // Green work
   } else {
      // Re
   }
   while(...) {
      // Gold work
}
```

Two fundamental problems
- ✓ Data divergence
- ✓ Control divergence



Vector code generation has become a more difficult problem
Increasing need for user guided explicit vectorization
Explicit vectorization maps threaded execution to simd hardware

[See WPMVP2016-Keynote-xtian-.pdf https://sites.google.com/site/wpmvp2016/talk for more info]

Vector Programming Tomorrow

- Nested vectorization (iterations from inner- and outer-loops plus SLP forming a single vector) [Zhou and Xue, CGO16, “mixed SIMD” talked about mixing intra-/inter-iteration parallelism.]
- More Idiomatic Patterns (Compress, Expand, Histogram [Demikhovsky, LLVM-US’13, AVX512], Last Value from Conditional Assignment)
- Any other ways to guide SIMD intrinsic programmers to high level language programming
- We even talk about how to support C++EH inside vector context
  - FP speculation safety is already part of today’s state-of-the-art. ☺

Programmer Demands:

Vectorization with all standard programming constructs inside.

Performance on-par with (or close enough to) SIMD intrinsic code.

Straightforward enough coding to maintain.

Ambitious Goal

- **One** vectorizer for loops, SLP (including load/store coalescing), function, and multi-level (or nested) vectorization
  - Demand for vectorizer is getting complex. Can't afford to reinvent/maintain multiple similar things.
  - Evaluate trade-offs between vectorizing one way versus another (e.g., inner- versus outer- loop vectorization) – using an abstract Vectorization Plan
- Extensible for further extensions of standards (OpenMP, OpenCL, C++, …) + various (non-)proprietary needs
- Pick and choose (i.e., customize) "features" to control compile time, code size, and target needs.
  - Vectorizer for JIT (e.g. OpenCL) has different requirements from static compilation
  - Vector-heavy targets needs higher functionality to achieve more %vector_coverage and better scalability

OpenMP4.5 as a good milestone

- Innermost loop vectorization
- Outer loop vectorization
- Function Vectorization
  - Similarity to “kernel vec” in OpenCL, WfV, etc.
- No SLP Vectorization yet
- Actively extended to meet programmer needs

- Multi-vendor + multi-platform to compare
- One vectorizer can accommodate both OpenMP SIMD and auto-vec
- Try designing the solution w/ further extensions in mind
  - Outer loop auto-vec, Nested Vectorization, and more

Compare and Contrast

- Inner- versus Outer-Vectorization
- Function- versus Loop-Vectorization
- Auto- versus Explicit-Vectorization

Innermost loop versus Outer-loop

```
// vectorize here
for (i=ilb; i<iub; i++){
   .....
}
```

- Non-loop control flow can be IF-converted/masked.
- Inner loop control flow needs massaging and then IF-convert+mask

```
// vectorize here
for( i=ilb; i<iub; i++){
   .....
   for (j=jlb(i); j<jub(i); j++){
      while(cond(i,j)) { ... }
      if (...) break;
   }
}
```

More about outer-loop vec

```
// vectorize here
for( i=ilb; i<iub; i++){
  .....
  for (j=jlb(i); j<jub(i); j++){
     while(cond(i,j)) { … }
     if (…) break;
  }
}
```

```
julb = hmin(jlb(i));
juub = hmax(jub(i));
cont1 = T;
for (j=julb; j<juub; j++){
   if (jlb(i) <= j && j < jub(i) && cont1) {
     cont2 = cond(i,j);
     while(hor(cont2)) {
       if (cont2) {
         …
         cont2 = cond(i,j);
       }
     }
     if (…) cont1=F;
     if (!hor(cont1)) break;
   }
}
```

Loop until last element finish looping

Optimization Notice

Loop versus Function

```
// vectorize here
for( i=ilb; i<iub; i++){
  .....
  for (j=jlb(i); j<jub(i); j++){
     while(cond(i,j)) { … }
     if (…) break;
  }
}
```

Essentially the same if loop body is extracted

```
// vectorize here
float foo(float x, int i, int *p) {
  .....
  for (j=jlb(i); j<jub(i); j++){
     while(cond(i,j)) { … }
     if (…) break;
  }
}
```

Essentially the same if explicit loop is created.

Loop versus Function (cont)

Loop Vectorization via Function Vectorizer

- Outline loop body into a function
- Vectorize
- Inline it back

High overhead even when we know a loop needs to be vectorized. Even more so for auto-vec.

Function Vectorization via Loop Vectorizer

- Create loop around function body + fix call interface
- Vectorize

Low overhead esp. when we know a function needs to be vectorized.

Auto- versus Explicit- Vectoriztion

- Typical auto-vectorization is ---- vectorize when planets align.
  - One bad thing ➔ do not vectorize
- Operating principle for Explicit Vectorization is the converse.
  - Vectorize ➔ unless a good justification why scalar is better exists: enclosed in `ordered simd` block, function call w/o vector function mapping, best emulation sequence is scalar code, etc.
  - Serialization w/o good justification is considered lack of robustness.
- Both should be implemented in the same framework.
  - Explicit vectorization helps auto-vectorizer more robust
  - Auto-vectorization helps explicit vectorizer work better w/o optional tuning clauses.

Auto- versus Explicit- Vectoriztion (cont)

- Explicit vectorization
  - could use more compile time and code size, esp. in static compilation (user opted-in)
  - Can proactively transform IR (instructed to vectorize)

- Auto vectorization
  - Shouldn’t spend a lot of compile time just to decide whether xform should kick-in or not (subject to stricter time/size ROI than opt-in)
  - Shouldn’t start to modify IR until estimated gain is known.

This is where OpenCL vectorizer (that we know) and WfV aren’t nice about.

It’s Feasible to Build One Loop Vectorizer

that can support

- Innermost loop auto- and explcit-vectorization
- Outer loop auto- and explcit-vectorization
- Function vectorization
- SLP-awareness

Execution Plan

Major Milestones towards OpenMP4.5

- Clang Front End to parse directives and create IR
  - Please attend Xinmin Tian's talk at the 3rd LLVM-HPC Workshop at SC'16 (Nov 14th) https://llvm-hpc3-workshop.github.io/
- Convert Function Vectorization to Loop-Vectorization
  - https://reviews.llvm.org/D22792
- Outer Loop Vectorization Support
  - RFC: http://lists.llvm.org/pipermail/llvm-dev/2016-September/105057.html
- Vectorization Plan
- Robust handling of uniform/linear/private/reduction, etc.
  - Make sure upstream transformation won't lose them or leave them inconsistent
  - POD (plain old data) and non-POD types
  - User-defined reduction is a big challenge

Major Building Blocks Needed for Outer-Loop Vectorization

1. Basic Block abstraction, Instruction abstraction
   - Need to massage inner loop control flow before deciding to vectorize
2. Uniformilty/Linearity/Privateness analysis
   - A loop-invariant value is uniform; Converse holds for innermost loops, but not for outer loops, in general.
3. Actual massaging of inner loop control flow
4. Robust predication/masking that works on massaged inner loop
5. Facility to compare inner loop vectorization cost model and outer loop vectorization cost model
6. Abstract Vectorization Plan is a good place to store them
7. Non-POD (arrays, structs) privatization is must-have in explicit vectorization.

Steps to Apply to LoopVectorize

Our plan is a gradual, incremental development to best leverage the efforts already invested, and actively ongoing, in the existing innermost Loop Vectorizer.

1. Introduce Vectorization Plan (NFC patch).
   VPlan models current decisions and transformations.
2. Let LoopVectorize retain uniform control flow.
   Less masking/blending can lead to better perf.
3. Vectorize Outer Loop if Inner Loop has uniform control flow.
4. Vectorize Outer Loop with Inner Loop control flow massaging.

Step 1: Introduce the Vectorization Plan

- 1st patch designed to be an NFC
- Migrate from the current flow of

  Legal → Cost → Transform

  where Cost tries to predict what Transform will do, and both are confined to a single fixed candidate assumed to be branch-free, to a flow of

  Legal → Plan(s) → Cost → Transform

  where a *Plan* provides both *Cost* and *Transform* for the resulting vector loop
- Evaluate compile-time and memory overheads, early pruning and other potential optimizations and clean-ups

Step 2: Retain Uniform Control-Flow

- Current innermost Loop Vectorizer if-converts all branches inside vectorized loop, inherently assuming resulting vector loop will be branch-free
- When vectorizing an outer-loop, resulting vector loop will necessarily contain branches – those that control internal loops
- This step teaches the innermost Loop Vectorizer to handle uniform branches inside its vector loop
- Evaluate the impact of the optimization, enabled under a flag

Steps 3 and 4: Extend LV to handle Outer Loops

- Extend the innermost Loop Vectorizer to handle outer loops, by transforming nested divergent loops into uniform loops; selection of best candidate loop to vectorize facilitated by Vectorization Plan
- At-first driven by directives; later guided by cost model
- At-first handle simple cases, e.g., being vectorized by GCC; later extended to support , e.g., OpenMP 4.x patterns

Summary

- Presented multi-year vectorizer enhancement goals towards Outer Loop Vectorization and beyond
- Jaxutaposed various vectorization approaches
- Incremental development on existing LoopVectorize to leverage past and ongoing efforts
- **One** vectorizer for OpenMP4.5 SIMD functionality and outer loop auto-vectorization (and more)

Call To Action

Participate in the discussion, design, development, code review, experiments, bug fixing, etc.

Project has a lot of work for many people to collaborate. We'd like to have coordinated development to minimize/eliminate redundancy and maximize productivity.

Contact us if you are interested in working on this.

Key Takeaways

1. We plan to teach LLVM to vectorize Outer-loops
   a. Explicitly via OpenMP4.5 and also automatically.
   b. Use it for vectorizing Functions
   c. Design also for Loop+SLP vectorization, Outer+Inner Loop vectorization, and more
2. Do so by extending LLVM's existing Innermost Loop Vectorizer
   a. Leveraging past and ongoing efforts as well as future maintenance
   b. Introducing Vectorization Plan to model (multiple) potential candidates
3. This long-term challenge just started; Collaboration welcome!

intel®