Outer-Loop Vectorization Legality Analysis for LLVM

One Step Closer to a Production Vectorizer, the Region Vectorizer

Stefanos Baziotis
NEC Corporation and University of Athens
users.uoa.gr/~sdi1600105/
stefanos.baziotis@gmail.com

Thanks to Simon Moll for guiding me through this research!
Still in Early Development

- You can follow it on [github.com/baziotis/llvm-project/tree/feature/lda](https://github.com/baziotis/llvm-project/tree/feature/lda)
Still in Early Development

- You can follow it on github.com/baziotis/llvm-project/tree/feature/lda
- Possibly already contains pieces of novel work
Interface

const LoopDependence getDependenceInfo(const Loop &L) const;
Interface

```cpp
const LoopDependence getDependenceInfo(const Loop &L) const;
```

Future:

- Cache results
Interface

```cpp
const LoopDependence getDependenceInfo(const Loop &L) const;
```

Future:

- Cache results
- Provide finer-grained interface (e.g. pairs of instructions)
Related Work

- A lot of time was dedicated on how to deduce dependence vectors
  - Direction / Distance vectors
Related Work

- A lot of time was dedicated on how to deduce dependence vectors
  - Direction / Distance vectors
- Later I found out that most of this work was already published in [1]

Reflecting Dependence Vectors

```java
for (int j = 0; j < N; ++j)
    A[j] = A[j + 1];
```

- Distance *from* the read *to* the write
Reflecting Dependence Vectors

```c
for (int j = 0; j < N; ++j)
    A[j] = A[j + 1];
```

- Distance *from* the read *to* the write
- When this distance is negative, we have an anti-dependence in memory access space
  - The read happens before the write
Reflecting Dependence Vectors

```java
for (int j = 0; j < N; ++j)
    A[j] = A[j + 1];
```

- Distance *from* the read *to* the write
- When this distance is negative, we have an anti-dependence in memory access space
  - The read happens before the write
- In iteration space, we have to reflect the vector
Reflecting Dependence Vectors

```c
for (int j = 0; j < N; ++j)
    A[j] = A[j + 1];
```

- Distance *from* the read *to* the write
- When this distance is negative, we have an anti-dependence in memory access space
  - The read happens before the write
- In iteration space, we have to reflect the vector
- This idea extends to N dimensions
  - When the memory access vector “looks” to previous iterations
Analyzing N-Dimensional Loop Nests

Recent work:

- Take all the pairs of dimensions with the dimension you’re vectorizing.
Analyzing N-Dimensional Loop Nests

Recent work:

- Take all the pairs of dimensions with the dimension you’re vectorizing.
- If at least one causes dependence, the vectorization factor is the distance in the dimension you’re vectorizing.
  - Falls back to 2D tests.
Recent work:

- Take all the pairs of dimensions with the dimension you’re vectorizing.
- If at least one causes dependence, the vectorization factor is the distance in the dimension you’re vectorizing.
  - Falls back to 2D tests.
- Otherwise, it’s vectorizable for any factor

Analyzing N-Dimensional Loop Nests
Analyzing Imperfect Loop Nests

- Easy to analyze a perfect loop nest
Analyzing Imperfect Loop Nests

- Easy to analyze a perfect loop nest
- View the imperfect nest as a tree
Analyzing Imperfect Loop Nests

- Easy to analyze a perfect loop nest
- View the imperfect nest as a tree
- Analyze each path to a leaf as a perfect loop nest
Analyzing Imperfect Loop Nests

- Easy to analyze a perfect loop nest
- View the imperfect nest as a tree
- Analyze each path to a leaf as a perfect loop nest
- Take the minimum vectorization factor
Future!

- Cost-Modeling (with Machine Learning)
- Analysis for Tensorization
- Loop Transformation Framework
Thank you!

Stefanos Baziotis
NEC Corporation and University of Athens
users.uoa.gr/~sdi1600105/
stefanos.baziotis@gmail.com