optview2
View and improve optimizations

Ofek Shilon
Istra Research
August 2021
What if the compiler could tell us...

• I couldn’t inline that function
  ...because I had only its declaration.

• I’m re-evaluating these loop boundaries on every iteration
  ...because I couldn’t prove they stay fixed.

• I’m doing tons of memory accesses that might be redundant
  ...because in theory two unrelated variables might refer to the same memory location. You modified one, so I had to reload the other from memory.
Raw optimization data is exposed

- **Clang/gcc**: `-Rpass` code.cc:4:25: remark: foo inlined into bar [-Rpass=inline]
- **Intel**: `-qopt-report=[0..5]`
- **Microsoft (very partial)**: `/Qpar-report, /Qvec-report`
Presentation Matters.

• Focus on Clang, over Linux
• opt-viewer, 2016 work by Adam Nemet (Apple), Hal Finkel (Argonne)
Opt-Viewer Usage

• Build with
  
  ```bash
  -fsave-optimization-record: opt.yaml files are generated, by default in the obj folder.
  ```

• Generate htmls:
  
  ```bash
  opt-viewer.py
  --output-dir <htmls folder>
  --source-dir <repo>
  <yamls folder>
  ```
opt-viewer.py Sample Output

```
REG OneToFifty IntIndex;

IntLoc = IntPar11 + 5;
Array1Par[IntLoc] = IntPar12;
Array1Par[IntLoc+1] = Array1Par[IntLoc];
Array1Par[IntLoc+30] = IntLoc;
for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
    Array2Par[IntLoc][IntIndex] = IntLoc;
    for (IntIndex = IntLoc; IntIndex <= (IntLoc+1); ++IntIndex)
        Array2Par[IntLoc][IntLoc-1] = IntLoc;
```

Hotness (PGO)

<table>
<thead>
<tr>
<th>Hotness</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>52%</td>
<td><code>loop-deleted</code></td>
</tr>
<tr>
<td>100%</td>
<td><code>loop-vecorize</code></td>
</tr>
<tr>
<td>52%</td>
<td><code>loop-idiom</code></td>
</tr>
<tr>
<td>17%</td>
<td><code>gvn</code></td>
</tr>
<tr>
<td>17%</td>
<td><code>gvn</code></td>
</tr>
<tr>
<td>34%</td>
<td><code>gvn</code></td>
</tr>
<tr>
<td>17%</td>
<td><code>gvn</code></td>
</tr>
</tbody>
</table>

Inlining context

- Proc0
- Proc8
- Proc0
- Proc0
- Proc0
- Proc8
- Proc0
Great work, still not much traction

https://www.youtube.com/watch?v=qq0q1hfzidg
Great work, still not much traction
Why?

• Heavy
  • High I/O
  • High memory
  • >1G htmls

• Designed for compiler writers
  • Mostly non actionable to developers
Introducing optview2

- [https://github.com/OfekShilon/optview2](https://github.com/OfekShilon/optview2)
Target Developers, Not Compiler Authors

• Denoise:
  • Collect only optimization failures
  • Ignore system headers
  • Display a single entry per type/source loc (in index)
  • ...
  • \( \sim 1.5M \text{ lines } \Rightarrow 22K \text{ lines} \)
• Don’t mention ‘passes’
• Make the index sortable, resizable & pageable
• Abridged function names
• ...
<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
<th>Function</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>/home/feb/src/opencv-clean/modules/ml/test/test_mtests.cpp</td>
<td>MissedDetails</td>
<td>__cxx_global_array_dtor.44</td>
<td>loop not vectorized</td>
</tr>
<tr>
<td>/home/feb/src/opencv-clean/modules/ml/test/test_mtests.cpp</td>
<td>MissedDetails</td>
<td>__cxx_global_array_dtor.51</td>
<td>loop not vectorized</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>MissedDetails</td>
<td>std::function_handler::finalize::Connector::M_Invoke</td>
<td>loop not vectorized</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>TooCostly</td>
<td>ade::CallbackConnector::finalize</td>
<td>std::function&lt;void&gt;();operator::CallbackConnector::finalize():: finalize because too costly to inline (cost=310, threshold=100)</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>TooCostly</td>
<td>ade::CallbackConnector::finalize</td>
<td>std::function&lt;void&gt;();operator::CallbackConnector::finalize():: finalize because too costly to inline (cost=280, threshold=100)</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>TooCostly</td>
<td>ade::CallbackConnector::finalize</td>
<td>std::function&lt;void&gt;();operator::CallbackConnector::finalize():: finalize because too costly to inline (cost=280, threshold=100)</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>TooCostly</td>
<td>ade::CallbackConnector::finalize</td>
<td>std::function&lt;void&gt;();operator::CallbackConnector::finalize():: finalize because too costly to inline (cost=280, threshold=100)</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>NoDefinition</td>
<td>ade::ICommChannel::BufferDesc::BufferDesc</td>
<td>ade::MemoryDescriptorRef::MemoryDescriptorRef will not be inlined into ade::ICommChannel::BufferDesc::BufferDesc because its definition is unavailable</td>
</tr>
<tr>
<td>src/direction/adef-0.1.1/if/sources/adef/include/adef/common</td>
<td>TooCostly</td>
<td>ade::ExecutionEngine::addPass</td>
<td>ade:ExecutionEngine::addPass::addPass::checkCycles: char const* is not inlined into</td>
</tr>
</tbody>
</table>
Demo 1: inlining

```
#include <iostream>

int main() {
    int a = 5;
    int b = 3;
    int c = a + b;
    return 0;
}
```

• cvChangeSeqBlock will not be inlined into `cv::SeqIterator<CvSeq>::operator++` because its definition is unavailable

• Dirty fix:
  • Move cvChangeSeqBlock from datastructs.cpp to types_c.h
  • Copy some macros - CV_SEQ_ELEM, CV_WRITE_SEQ_ELEM_VAR CV_WRITE_SEQ_ELEM
  • Remove declaration from core_c.h

• Less relevant if you’re building with LTO
Demo 2:

```c
for (int i = 0; i < histSize; ++i)
    tileHist[i] += redistBatch;
```

• What if `redistBatch` references an element of `tileHist`?
  • *Aliasing*
  • Forces reloading `redistBatch` on every loop iteration
  • Compilers are remarkably bad at alias analysis.
Aliasing – the silent killer

• Distribution of 22K remarks in a C++ project:
Aliasing II: What can we do?

• LTO ?  ❌

• __restrict__  ✔
  - Non standard, tells the compiler *an argument* doesn’t alias with anything else

• __attribute__((pure)), __attribute__((const))  ❌
  - Non standard, tell the compiler *a function* doesn’t write/read global state.
  - Potentially resolve ‘clobbered by call’ opt failures,
  - No successes seen in the wild yet.

• Strict-Aliasing  ✔
  - The only C++ standard compliant way.

Detour
Aliasing III: Types and Strict-Aliasing

• “Strict aliasing is an assumption made by the compiler, that objects of different types will never refer to the same memory location (i.e. alias each other.)”

  Mike Acton  [https://cellperformance.beyond3d.com/articles/2006/06/understanding-strict-aliasing.html](https://cellperformance.beyond3d.com/articles/2006/06/understanding-strict-aliasing.html)

• Allowed optimization. on by default for -O2 +
• char* is an exception – always allowed to alias with any type.
• Can be disabled with –fno-strict-aliasing (pessimization!)
• In the example above – all vars are int.
Aliasing IV: Forcing type difference

- `typedef / using` ✗
- `Inherit int?` ✗
  - “I considered ... to allow derivation from built-in classes ... However, I restrained myself. ... the C conversion rules are so chaotic that pretending that int, short, etc., are well-behaved ordinary classes is not going to work. They are either C compatible, or they obey the relatively well-behaved C++ rules for classes, but not both.”
  - Bjarne Stroustrup, The Design and Evolution of C++, §15.11.3.

- **Strong-Typedefs** ✓
  - Discussion typically motivated by type safety, not optimization
  - `boost/serialization/strong_typedef.hpp` comes close.
#define BOOST_STRONG_TYPEDEF(T, D)  
struct D : boost::totally_ordered1< D, boost::totally_ordered2< D, T > > {  
  T t;
  explicit D(const T& t_) : t(t_) {}  
  D() : t() {}  
  D(const D & t_) : t(t_.t) {}  
  D& operator=(const D& rhs) { t = rhs.t; return *this; }  
  D& operator=(const T& rhs) { t = rhs; return *this; }  
  operator const T&() const { return t; }  
  operator T&() const { return t; }  
  bool operator==(const D& rhs) const { return t == rhs.t; }  
  bool operator<(const D& rhs) const { return t < rhs.t; }  
};
#define CUSTOM_STRONG_TYPEDEF(T, D)
struct D
{
    T t;
    D(const T t_) : t(t_) {}
    D(){}
    D(const D & t_) : t(t_.t) {}
    D & operator=(const D & rhs) { t = rhs.t; return *this; }
    D & operator=(const T & rhs) { t = rhs; return *this; }
    operator const T & () const {return t; }
    operator T & () { return t; }
};
Aliasing V: Impact

```cpp
cv::AutoBuffer<int> _tileHist(histSize);
int* tileHist = _tileHist.data();
std::fill(tileHist, tileHist + histSize, 0);
```

```cpp
CUSTOM_STRONG_TYPEDEF(int, tHistDat)
cv::AutoBuffer<tHistDat> _tileHist(histSize);
tHistDat* tileHist = _tileHist.data();
std::fill(tileHist, tileHist + histSize, 0);
```

// redistribute clipped pixels
int redistBatch = clipped / histSize;
int residual = clipped - redistBatch * histSize;

```cpp
for (int i = 0; i < histSize; ++i)
    tileHist[i] += redistBatch;
```

...failed to move load with loop-invariant address because the loop may invalidate its value...

Detour
Aliasing V: Impact

Detour
Impact

class CV_EXPORTS Mat
{
    ...
    //! pointer to the data
    uchar* data;
    tDat data;
};

struct tDat
{
    uchar* t;
    tDat(uchar* const & t_) : t(t_) {}
    tDat(){}
    tDat(const tDat & t_) : t(t_.t){}
    tDat(tDat & t_) : t(t_.t){}
    tDat & operator=(tDat & rhs) { t = rhs.t; return *this;}
    tDat & operator=(uchar* rhs) { t = rhs; return *this;}
    operator uchar* () const { return t; }
    operator const uchar* () { return t; }
    bool operator==(uchar* rhs) const {return t == rhs;}
    bool operator==(const uchar* rhs) const {return t == rhs;}
    bool operator==(tDat& rhs) const {return t == rhs.t;}
    bool operator==(const tDat& rhs) const {return t == rhs.t;}
    bool operator==(nullptr_t) const { return t == nullptr;}
    bool operator!=(uchar* rhs) const {return t != rhs;}
    bool operator!=(tDat& rhs) const {return t != rhs.t;}
    bool operator!=(const tDat& rhs) const {return t != rhs.t;}
    uchar* operator+(long i) const { return t + i; }
    uchar* operator+=(long i) { t += i; return t;}
    uchar* operator-(long i) { t -= i; return t;}
    uchar& operator[](long i) { return t[i];}
    uchar operator[](long i) const { return t[i];}
    template<typename T> operator T*() const { return (T*)(t); }
};
Impact

https://github.com/opencv/opencv/wiki/HowToUsePerfTests

$ cd ~/src/opencv/modules/ts/misc/

$ python3 ./run.py ~/src/opencv-clean/build/ -w ~/logs-opencv-clean

$ python3 ./run.py ~/src/opencv-opt/build/ -w ~/logs-opencv-opt

$ python3 ./summary.py ~/logs-opencv-clean/core* ~/logs-opencv-opt/core*
   -u mks -m median

• Seems substantial.

• Will publish more detailed steps in the optview2 github page (and possibly fork opencv)
Demo 3: loop hoisting

```cpp
for (int i = rowStart; i <= boundaries.end; ++i)
    // failed to move load with loop-invariant address because the loop may invalidate its value
    // load of type i32 not eliminated in favor of load because it is clobbered by store
    // load of type i32 not eliminated because it is clobbered by store

if (L2gradient)
    // failed to move load with loop-invariant address because the loop may invalidate its value
    // load of type i32 not eliminated in favor of load because it is clobbered by store
    // load of type i32 not eliminated because it is clobbered by store
```
Demo 3: loop hoisting

```c
const int end = boundaries.end;
const bool localL2gradient = L2gradient;

for (int i = rowStart; i <= end; ++i)
{
    // 64 spills 1 folded spills 115 reloads 18 folded reloads generated in loop

    if (localL2gradient)
    {
        int j = 0, width = src.cols * cn;
    }
}
```
PGO integration

• “Hotness” column, can be used for sorting or filtering
• Demo

• Not necessary. If you have profiling data – use that to narrow your search.
@mattgodbolt Have you considered adding opt-viewer (LLVM) support to Compiler Explorer? Seems useful: github.com/androm3da/optv...

not as yet no. But looks like a great idea
GCC work

- [https://github.com/davidmalcolm/gcc-opt-viewer](https://github.com/davidmalcolm/gcc-opt-viewer)

- **YAML -> JSON**
  - Actually a better choice
  - [https://stackoverflow.com/questions/27743711/can-i-speedup-yaml](https://stackoverflow.com/questions/27743711/can-i-speedup-yaml)

- Worse starting point
- Active only during 2018
Conclusions applicable to other compilers?

•?
Future work

• Enhance filtering,
• Consume binary optimization remarks,
• Reduce run time & memory,
• Adapt (possibly integrate) gcc opt-viewer,
• Support code annotations,
• Script code quality improvements,
• Report LLVM bugs

• Lots of help needed 😊
• https://github.com/OfekShilon/optview2
https://github.com/OfekShilon/optview2

- ofekshilon@gmail.com
- Questions?
Bkp
$ clang -O3 -o /tmp/v.o -c /tmp/v.c -fsave-optimization-record
$ llvm-opt-report /tmp/v.yaml > /tmp/v.lst
$ cat /tmp/v.lst

• https://reviews.llvm.org/D25262
<table>
<thead>
<tr>
<th>Name of Test</th>
<th>core clean</th>
<th>core opt</th>
<th>core opt vs core clean</th>
</tr>
</thead>
<tbody>
<tr>
<td>subtract::BinaryOpTest::&lt;(1920x1080, 165C4)</td>
<td>3189.750</td>
<td>3163.450</td>
<td>1.01</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 8UC1)</td>
<td>143.450</td>
<td>163.900</td>
<td>1.18</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 8SC1)</td>
<td>91.400</td>
<td>86.600</td>
<td>1.06</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 165C1)</td>
<td>78.100</td>
<td>79.880</td>
<td>1.01</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 32SC1)</td>
<td>68.800</td>
<td>55.980</td>
<td>1.09</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 32FC1)</td>
<td>59.950</td>
<td>56.100</td>
<td>1.07</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 165C2)</td>
<td>170.000</td>
<td>162.700</td>
<td>1.04</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 8UC3)</td>
<td>361.100</td>
<td>291.200</td>
<td>1.24</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 165C3)</td>
<td>262.750</td>
<td>249.200</td>
<td>1.05</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 8UC4)</td>
<td>485.900</td>
<td>416.250</td>
<td>1.17</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(640x480, 165C4)</td>
<td>338.100</td>
<td>333.800</td>
<td>1.01</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 8UC1)</td>
<td>380.050</td>
<td>265.100</td>
<td>1.43</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 8SC1)</td>
<td>343.700</td>
<td>262.100</td>
<td>1.31</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 165C1)</td>
<td>319.800</td>
<td>233.100</td>
<td>1.37</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 32SC1)</td>
<td>189.000</td>
<td>196.300</td>
<td>1.05</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 32FC1)</td>
<td>187.000</td>
<td>196.300</td>
<td>1.05</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 165C2)</td>
<td>630.350</td>
<td>536.000</td>
<td>1.18</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 8UC3)</td>
<td>925.650</td>
<td>821.550</td>
<td>1.13</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 165C3)</td>
<td>942.100</td>
<td>921.600</td>
<td>1.02</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 8UC4)</td>
<td>1486.450</td>
<td>1663.900</td>
<td>1.14</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1280x720, 165C4)</td>
<td>1130.700</td>
<td>1259.200</td>
<td>1.12</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 8UC1)</td>
<td>765.650</td>
<td>591.050</td>
<td>1.31</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 8SC1)</td>
<td>713.450</td>
<td>689.800</td>
<td>1.06</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 165C1)</td>
<td>615.650</td>
<td>627.650</td>
<td>1.03</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 32SC1)</td>
<td>762.300</td>
<td>759.900</td>
<td>1.01</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 32FC1)</td>
<td>768.750</td>
<td>795.300</td>
<td>1.05</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 165C2)</td>
<td>1311.400</td>
<td>1327.700</td>
<td>1.01</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 8UC3)</td>
<td>2252.000</td>
<td>2382.750</td>
<td>1.09</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 165C3)</td>
<td>2587.250</td>
<td>2258.700</td>
<td>1.15</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 8UC4)</td>
<td>3094.700</td>
<td>3235.900</td>
<td>1.06</td>
</tr>
<tr>
<td>subtractScalarDouble::BinaryOpTest::&lt;(1920x1080, 165C4)</td>
<td>3333.300</td>
<td>2812.950</td>
<td>1.18</td>
</tr>
</tbody>
</table>