Getting Started with Automatic Compiler Vectorization

David Apostal
UND CSci 532 Guest Lecture
Sept 14, 2017
Parallellism is Key to Performance

• Types of parallelism
  – Task-based (MPI)
  – Threads (OpenMP, pthreads)
  – Vector processing
Scalar Processing

• Processor uses one element from each array on each loop iteration.

```c
#define N 32
float a[N], b[N], c[N];
for (int i = 0; i < N; i++) {
    a[i] = b[i] + c[i];
}
```
Vector Processing

• Compiler unrolls the loop body a number of times.

```c
#define N 32
float a[N], b[N], c[N];
for (int i = 0; i < N; i=i+4) {
    a[i] = b[i] + c[i];
    a[i+1] = b[i+1] + c[i+1];
    a[i+2] = b[i+2] + c[i+2];
    a[i+3] = b[i+3] + c[i+3];
}
```

• Processor performs the same operation on a number of scalars (simultaneously)
Techniques

- Automatic Compiler Vectorization
- Compiler hints (directives / # pragmas)
- SIMD / Vector intrinsics
- Assembler instructions
Requirements / Guidelines

- Compiler optimization –O2 or higher
- Loop trip count known at runtime.
- A single entry and single exit
- Straight-line code
- Inner-most loop
- No function calls
Manage the Compiler Report

• Vectorization is enable by default
• -qopt-report-phase=vec
• -qopt-report=[1-5]
  – Provides increasing levels of detail
• -qopt-report-routine:func1, func2
Exercise

Test.cpp

```c
#include <math.h>
void foo(float *theta, float *sth) {
    for (int I = 0; I < 128; i++) {
        sth[i] = sin(theta[i] + 3.1415927);
    }
}
```

$ icpc -qopt-report-phase=vec \
   -qopt-report=5 \
   -qopt-report-routine:foo \
   -c \
   Test.cpp$
Test.optrpt (1)

Begin optimization report for: foo(float *, float *)

Report from: Vector optimizations [vec]

LOOP BEGIN at T1.cpp(3,5)
<Multiversioned v1>
... remark #15300: LOOP WAS VECTORIZED
...
LOOP END

LOOP BEGIN at T1.cpp(3,5)
<Multiversioned v2>
remark #15304: loop was not vectorized: non-vectorizable loop instance from multiversioning
LOOP END
Aliasing

• Add a command-line option to tell compiler that function arguments are not aliased.

```
$ icpc -qopt-report-phase=vec \
   -qopt-report=5 \
   -qopt-report-routine:foo \
   -c \
   -fargument-noalias \
   Test.cpp
```

• Results in a single vectorized version of loop.
Test.optrpt (2)

Begin optimization report for: foo(float *, float *)

Report from: Vector optimizations [vec]

LOOP BEGIN at T1.cpp(3,5)
remark #15389: vectorization support: reference theta has unaligned access [ T1.cpp(4,18) ]
remark #15389: vectorization support: reference sth has unaligned access [ T1.cpp(4,9) ]
remark #15381: vectorization support: unaligned access used inside loop body
remark #15305: vectorization support: vector length 4
remark #15309: vectorization support: normalized vectorization overhead 0.081
remark #15417: vectorization support: number of FP up converts: single precision to double precision 1 [ T1.cpp(4,18) ]
remark #15418: vectorization support: number of FP down converts: double precision to single precision 1 [ T1.cpp(4,9) ]
remark #15300: LOOP WAS VECTORIZED
remark #15450: unmasked unaligned unit stride loads: 1
remark #15451: unmasked unaligned unit stride stores: 1
remark #15475: --- begin vector loop cost summary ---
remark #15488: --- end vector loop cost summary ---
LOOP END
Alignment

• Tell the compiler that arrays are aligned.

```c
#include <math.h>
void foo(float *theta, float *sth) {
    __assume_aligned(theta, 32);
    __assume_aligned(sth, 32);
    for (int I = 0; I < 128; i++) {
        sth[i] = sin(theta[i] + 3.1415927);
    }
}
```
LOOP BEGIN at T1.cpp(3,5)
remark #15388: vectorization support: reference theta has aligned access [ T1.cpp(6,18) ]
remark #15388: vectorization support: reference sth has aligned access [ T1.cpp(6,9) ]
remark #15305: vectorization support: vector length 8
remark #15309: vectorization support: normalized vectorization overhead 0.006
remark #15417: vectorization support: number of FP up converts: single precision to double precision 1 [ T1.cpp(6,18) ]
remark #15418: vectorization support: number of FP down converts: double precision to single precision 1 [ T1.cpp(6,9) ]
remark #15300: LOOP WAS VECTORIZED
remark #15448: unmasked aligned unit stride loads: 1
remark #15449: unmasked aligned unit stride stores: 1
remark #15475: --- begin vector loop cost summary ---
remark #15476: scalar loop cost: 113
remark #15477: vector loop cost: 20.000
remark #15478: estimated potential speedup: 5.640
remark #15482: vectorized math library calls: 1
remark #15487: type converts: 2
remark #15488: --- end vector loop cost summary ---
LOOP END
Avoid type conversions

• Specify float (f) on function call and literal value.

```c
#include <math.h>
void foo(float *theta, float *sth) {
    __assume_aligned(theta, 32);
    __assume_aligned(sth, 32);
    for (int I = 0; I < 128; i++) {
        sth[i] = sinf(theta[i] + 3.1415927f);
    }
}
```
LOOP BEGIN at T1.cpp(5,5)
remark #15388: vectorization support: reference theta has aligned access  [ T1.cpp(6,18) ]
remark #15388: vectorization support: reference sth has aligned access  [ T1.cpp(6,9) ]
remark #15305: vectorization support: vector length 8
remark #15309: vectorization support: normalized vectorization overhead 0.013
remark #15300: LOOP WAS VECTORIZED
remark #15448: unmasked aligned unit stride loads: 1
remark #15449: unmasked aligned unit stride stores: 1
remark #15475: --- begin vector loop cost summary ---
remark #15476: scalar loop cost: 110
remark #15477: vector loop cost: 9.870
remark #15478: estimated potential speedup: 11.130
remark #15482: vectorized math library calls: 1
remark #15488: --- end vector loop cost summary ---
LOOP END
Resources

• https://software.intel.com/en-us/articles/vectorization-essential
• https://software.intel.com/en-us/mkl