Compiler Optimization

[redacted]

```
library(ggplot2)
library(tidyverse)
```

Job Submission

The following script submits a Slurm job to development queue then:

- Print the hostname for each task
- Write a system topology XML using `lstopo`
- Compile 3 variants on the batch host
- Assign 4 tasks to each variant and write out `srun --multi-prog` configuration file
- Invoke `srun` to run the program
- Cleanup outputs into one file per variant

```
#!/usr/bin/env fish
#SBATCH -J hw1_compiler    # Job name
#SBATCH -o hw1_compiler.o%j       # Name of stdout output file
#SBATCH -e hw1_compiler.e%j       # Name of stderr error file
#SBATCH -p development     # Queue (partition) name
#SBATCH -N 1               # Total # of nodes (must be 1 for serial)
#SBATCH -n 12
#SBATCH -t 00:20:00        # Run time (hh:mm:ss)
#SBATCH --mail-type=all    # Send email at begin and end of job
[redacted]

set -x LANG en_US.UTF-8
set CC icc
set CFLAGS -xcore-avx512 -qopt-report-phase=vec,ipo,loop -qopt-report=3
set fish_trace 1

function compile -w $CC
    $CC $CFLAGS $argv
end

echo "JOBID="$SLURM_JOB_ID
srun --label hostname

module load intel

lstopo -f topo.xml

compile -O0 -ggdb -o vector.dbg vector.c
compile -O1 -o vector.o1.exec \
            -qopt-report-file=vector.o1.optrpt \
            vector.c
compile -O2 -o vector.o2.exec \
            -qopt-report-file=vector.o2.optrpt \
            vector.c
compile -O2 -o vector.o2.novec.exec \
            -qopt-report-file=vector.o2.novec.optrpt \
            -no-vec  \
            vector.c

echo "## AUTO GENERATED DO NOT EDIT" > job.prog.conf
set -l task_rank 0
set -l nrep 4

for f in *.exec
    echo $task_rank"-"(math $task_rank + $nrep - 1) $f >> job.prog.conf
    set task_rank (math $task_rank + $nrep)
end

srun --output hw1_compiler.%j.%t.out \
        -c1 -n$task_rank --mem-bind=local \
        --multi-prog job.prog.conf

set -l i 0
for f in *.exec
    set -l fout (string replace -r .exec\$ .out $f)
    set -l fin hw1_compiler.$SLURM_JOBID.(seq $i (math $i + $nrep - 1)).out
    cat $fin > $fout
    rm $fin

    set i (math $i + $nrep)
end
```

```
sbatch job.sbatch.fish
```

```
## JOBID=4958671
##  0: c202-013.frontera.tacc.utexas.edu
##  4: c202-013.frontera.tacc.utexas.edu
##  9: c202-013.frontera.tacc.utexas.edu
##  8: c202-013.frontera.tacc.utexas.edu
##  3: c202-013.frontera.tacc.utexas.edu
##  6: c202-013.frontera.tacc.utexas.edu
##  2: c202-013.frontera.tacc.utexas.edu
##  1: c202-013.frontera.tacc.utexas.edu
##  7: c202-013.frontera.tacc.utexas.edu
## 10: c202-013.frontera.tacc.utexas.edu
##  5: c202-013.frontera.tacc.utexas.edu
## 11: c202-013.frontera.tacc.utexas.edu
```

Setup Information

We are working in `/work2/07468/mfu/frontera/sds335fall2022/exercises/hw1_compiler`.

Variable sizes are calculated as follows:

```
gdb --batch \
    -ex "b main" -ex "r" \
    -ex "rwatch i" -ex "c" \
    -ex "printf \"Variable Sizes:|a: %u KiB|B: %u KiB|c: %u KiB\", sizeof(a)>>10, sizeof(B)>>10, sizeof
        vector.dbg | grep "Variable Sizes"  |  sed "s/|/\n/g"
```

```
## Variable Sizes:
## a: 8 KiB
## B: 8192 KiB
## c: 8 KiB
```

```
echo "Cache Information:"
lstopo -i topo.xml --only cache | head -n5
```

```
## Cache Information:
## L3 L#0 (39MB)
## L2 L#0 (1024KB)
## L1d L#0 (32KB)
## L1i L#0 (32KB)
## L2 L#1 (1024KB)
```

We see the system has 39MB of L3 cache so all of the variables can fit in L3.

Target Code

The optimization target is a triple nested loop as follows:

```
// Compute a = B * c
for (n = 0; n < iter; n++)
{
a[0] = 0.;
for (j = 0; j < N; j++)
    for (i = 0; i < N; i++)
    a[i] = a[i] + B[i][j] * c[j];
}
```

The outer loop is to ensure the computation takes long enough to measure the performance difference. The timing would be too short to measure accurate if we only run the inner loops once.

The 2 innermost loops computes a dot product of matrix B and vector c, in a column-major (non-sequential) fashion.

Compiler Optimization Reports

We used the following two flags to generate the reports:

- `-qopt-report-phase=vec,ipo,loop` which enables report for vectorization, interprocedural optimization, and loop optimization
- `-qopt-report=3` which generates level 3 details.

By inspecting the following report:

```
cat vector.o2.optrpt.short
```

```
## Intel(R) Advisor can now assist with vectorization and show optimization
##   report messages with your source code.
## See "https://software.intel.com/en-us/intel-advisor-xe" for details.
##
## Intel(R) C Intel(R) 64 Compiler for applications running on Intel(R) 64, Version 19.1.1.217 Build 20
##
## Compiler options: -xcore-avx512 -qopt-report-phase=vec,ipo,loop -qopt-report=3 -O2 -o vector.o2.exec
##
##     Report from: Interprocedural optimizations [ipo]
##
## ...
##
##
##     Report from: Loop nest & Vector optimizations [loop, vec]
##
##
## ...
##
## LOOP BEGIN at vector.c(36,3)
##    remark #25096: Loop Interchange not done due to: Imperfect Loop Nest (Either at Source or due to
##    remark #25451: Advice: Loop Interchange, if possible, might help loopnest. Suggested Permutation
##    remark #15542: loop was not vectorized: inner loop was already vectorized
##
##    LOOP BEGIN at vector.c(40,7)
##       remark #25444: Loopnest Interchanged: ( 1 2 ) --> ( 2 1 )
##       remark #15542: loop was not vectorized: inner loop was already vectorized   [ vector.c(40,7) ]
##
##       LOOP BEGIN at vector.c(39,5)
##          remark #25085: Preprocess Loopnests: Moving Out Load and Store    [ vector.c(41,9) ]
##          remark #15301: PERMUTED LOOP WAS VECTORIZED
##          remark #26013: Compiler has chosen to target XMM/YMM vector. Try using -qopt-zmm-usage=high
##          remark #15448: unmasked aligned unit stride loads: 2
##          remark #15475: --- begin vector cost summary ---
##          remark #15476: scalar cost: 7
##          remark #15477: vector cost: 1.750
##          remark #15478: estimated potential speedup: 3.950
##          remark #15488: --- end vector cost summary ---
##          remark #25015: Estimate of max trip count of loop=32
##       LOOP END
##    LOOP END
## LOOP END
##
## ...===========================
```

We see that:

- The outermost loop on line 36 was kept as is
- The 2 innermost loops on lines 39 and 40 were reordered so now memory access is sequential
- The innermost loop were then vectorized.

This is equivalent to:

```
// Compute a = B * c
for (n = 0; n < iter; n++)
{
a[0] = 0.;
for (i = 0; i < N; i++)
#pragma simd
    for (j = 0; j < N; j++)
    a[i] = a[i] + B[i][j] * c[j];
}
```

In this new reordered code, access patterns become:

- `a` is a scalar with respect to the inner (vectorized) loop.
- `B` is a 2D array accessed in sequentially (row-major).
- `c` is a 1D array accessed in sequentially.

This is better than the original code because:

- The output variable `a` is now a scalar (reduction) so cache can be shared among cores.
- Variable `B` was accessed in column-major (stride N) so it cannot be vectorized or efficiently cached.
- Variable `c` was accessed as a scalar and now it is a vectorized read, should not be a big difference.

Timing Results

We parse the timing output into a CSV file.

```
awk -vOFS=, '
    BEGIN {print "flavor", "time"}
    BEGINFILE {gsub("^vector\\\\.|\\\\.out$", "", FILENAME) }
    /run time/ { print FILENAME, $NF+0 }' *.out > timing.csv
```

```
timing <- read_csv("timing.csv")
```

```
## Rows: 12 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): flavor
## dbl (1): time
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
timing %>%
    group_by(flavor) %>%
    summarize(mean_time = mean(time), min_time = min(time)) %>%
    arrange(mean_time) %>%
    knitr::kable(
        longtable = TRUE,
        col.names = c("Flavor", "Mean Time (s)", "Min Time (s)"),
        caption = "Timing Results"
    )
```

Table 1: Timing Results

Flavor	Mean Time (s)	Min Time (s)
vector.o2.out	3.775673	3.64380
vector.o2.novec.out	7.421145	7.22009
vector.o1.out	32.960575	32.85370

We see that the no-vec variant, which disables vectorization, is slower than the -O2 counterpart by about 1 fold. This is because without vectorization, the compiler only emits scalar instructions, which processes one element at a time as opposed to 4 elements at a time.

If we compare the -O2 -no-vec and -O1 variants:

```
diff vector.o1.optrpt vector.o2.novec.optrpt || true
```

```
## 7c7
## < Compiler options: -xcore-avx512 -qopt-report-phase=vec,ipo,loop -qopt-report=3 -O1 -o vector.o1.exe
## ---
## > Compiler options: -xcore-avx512 -qopt-report-phase=vec,ipo,loop -qopt-report=3 -O2 -o vector.o2.no
## 39c39,73
## < remark #15320: routine skipped: loop optimizations disabled
## ---
## >
## > LOOP BEGIN at vector.c(23,3)
## >    remark #15541: outer loop was not auto-vectorized: consider using SIMD directive
## >
## >    LOOP BEGIN at vector.c(27,5)
## >       remark #15540: loop was not vectorized: auto-vectorization is disabled with -no-vec flag
## >       remark #25438: unrolled without remainder by 2
## >    LOOP END
## > LOOP END
## >
## > LOOP BEGIN at vector.c(36,3)
## >    remark #25096: Loop Interchange not done due to: Imperfect Loop Nest (Either at Source or due to
## >    remark #25451: Advice: Loop Interchange, if possible, might help loopnest. Suggested Permutation
## >    remark #15541: outer loop was not auto-vectorized: consider using SIMD directive
## >
## >    LOOP BEGIN at vector.c(40,7)
## >       remark #25444: Loopnest Interchanged: ( 1 2 ) --> ( 2 1 )
## >       remark #15541: outer loop was not auto-vectorized: consider using SIMD directive   [ vector.
## >
## >       LOOP BEGIN at vector.c(39,5)
## >          remark #25085: Preprocess Loopnests: Moving Out Load and Store    [ vector.c(41,9) ]
## >          remark #15540: loop was not vectorized: auto-vectorization is disabled with -no-vec flag
## >          remark #25438: unrolled without remainder by 2
## >          remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
## >          remark #25457: Number of partial sums replaced: 1
## >       LOOP END
## >    LOOP END
## > LOOP END
## >
## > LOOP BEGIN at vector.c(51,3)
## >    remark #15540: loop was not vectorized: auto-vectorization is disabled with -no-vec flag
## >    remark #25438: unrolled without remainder by 2
## >    remark #25456: Number of Array Refs Scalar Replaced In Loop: 1
## >    remark #25457: Number of partial sums replaced: 1
## > LOOP END
```

We see that the -O1 lack 2 additional optimizations that can cause the observed 4x slowdown: - Despite both using scalar instructions, -O2 enabled loop unrolling that can reduce the number of instructions and loop overhead. - The previously mentioned loop permutation was not applied to -O1 variant.