A Kishore Kumar
@akcube:matrix.org
Github: **@akcube**
Email: a.kishorekumar@students.iiit.ac.in, kishash22@gmail.com
Pursuing **B.Tech Honors in Computer Science & Engineering @ IIIT - Hyderabad**

Background Information

EDUCATIONAL BACKGROUND

I'm currently pursuing a B.Tech in Honors in Computer Science and Engineering at the International Institute of Information Technology, Hyderabad (IIIT - H). I'm in my second year and the relevant courses I have taken are Computer Systems and Organization (CSO), Operating Systems & Networking (OSN) and Software Programming for Performance (SPP). I have also audited the course on Compilers. I am a part of my university's **C**omputer **S**ystems **G**roup Lab and conduct research on parallelizing algorithms in the bioinformatics space and compilers.

PROGRAMMING BACKGROUND & INTERESTS

OSS projects I have contributed to include maintaining the websites of two of my university's club websites and helping host a discord bot. I would like to get more involved with the world of OSS through GSoC. I enjoy taking part in optimization contests and love working on low-level, HPC code. More on this in *previous work*.

INTEREST IN PROJECT

This project focuses heavily on aiding compiler codegen and making conditions favorable for SIMD parallelization. I've spent a lot of time pouring over Agner Fog's instruction tables and just love to play around with SIMD and inline assembly to see how small changes can have drastic performance changes (example: temporal stores vs non-temporal stores). I have also gone through some material from LLVM's kaleidoscope tutorial previously. The project fits both my interests in HPC and Compilers and I feel like I will learn a lot from this project.

PREVIOUS WORK

1. I was introduced to the world of HPC 2 years back when I took the Speller problem (Implementing a fast hashtable) seriously and competed against others. I held the rank #1 spot for around 2 weeks. Now, even after 2 years I hold the #7 rank on the leaderboard. I

learnt a lot of things about SIMD, cache coherence and spent enough time micro-optimizing to learn not to do it again :)

2. As a pet project I decided to start working on my own implementation of a BLAS conforming library called KBLAS. I wrote my own benchmark library, to generate test data, verification data and benchmark code. I used ideas of data parallelization (SIMD intrinsics for vectorization and OpenMP for thread-parallelization) to beat the BLIS library and CBLAS by good margins. It also contains code related to roofline-analysis where I wrote benchmarks to extract the maximum possible bandwidth from my system & tuning the Stream benchmark. I was able to beat even my tuned Stream benchmark by ~5GBPS using SIMD reads and compiler tricks.

   I documented my entire journey and relevant benchmark data here in my Notion site. The graphs which are labeled "Final implementation of xyz" are the final results I was able to obtain. Although I did rely on some architecture specific information such as hardcoded cache sizes I intend on making it more easily configurable in the future. *(Note: This is an ongoing project and will be updated whenever I find time)*

3. I have also worked on modifying the MIT xv6-riscv operating system as part of my OSN course to implement different types of schedulers. I had to solve multiple thread synchronization issues while implementing this project.

PLANS BEYOND GSoC

I'll be honest and admit that I found out about GSoC and HPX pretty late in the process. HPC and parallelization is my favorite field to explore and work with in Computer Science. I discovered HPX recently, but instantly liked and agreed with many of the points in the ParalleX model of parallelization. Latency due to data transfer and thread sync barriers are two huge obstacles I have encountered in the past and I really like how ParalleX suggests using constraint based synchronization and latency hiding to solve these problems. I would like to learn more about them and work on implementing real code to demonstrate the usability of these techniques. The to-do list here is massive and I'd love to keep working on checking off more boxes even after GSoC. The few people I have interacted with made me feel welcome and helped me out a lot despite my last-minute showing. I find the community in general very supportive and would definitely love to keep working with them.

SELF ASSESSMENT

- C++ : 5/5 | C / C++ are my primary languages
- C++ Standard Library : 5/5
- Boost C++ Libraries : 2.5/5

  I have limited experience using the boost library but I'm willing to put in the time to learn and get familiar with it.

- Git distributed source code control system : 5/5 | Daily driver

Familiar software development environment: **CLion**

Familiar software documentation tool: **Doxygen**

PARALLEL MATRIX MULTIPLICATION

https://github.com/akcube/hpx_matmul

Conduct a thorough Performance Analysis on HPX Parallel Algorithms (and optimize)

19th April 2022

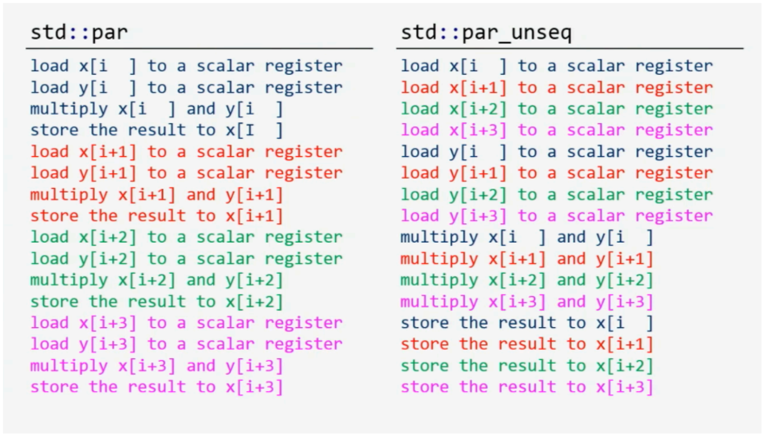
PROBLEM ABSTRACT

C++ 17 introduced the std::execution::par_unseq execution policy. C++ 20 followed up on this with std::execution::unseq. These execution policies provide us the guarantee that we can interleave the execution of multiple element access function calls in the same thread.

The image on the right is from Bryce Lelbach's talk in cppcon 2016.

To quote P0024R2,

"Vectorization-unsafe standard library functions may not be invoked by user code called from parallel_vector_execution_policy



```
std::par                              std::par_unseq
load x[i  ] to a scalar register      load x[i  ] to a scalar register
load y[i  ] to a scalar register      load x[i+1] to a scalar register
multiply x[i  ] and y[i  ]            load x[i+2] to a scalar register
store the result to x[I  ]            load x[i+3] to a scalar register
load x[i+1] to a scalar register      load y[i  ] to a scalar register
load y[i+1] to a scalar register      load y[i+1] to a scalar register
multiply x[i+1] and y[i+1]            load y[i+2] to a scalar register
store the result to x[i+1]            load y[i+3] to a scalar register
load x[i+2] to a scalar register      multiply x[i  ] and y[i  ]
load y[i+2] to a scalar register      multiply x[i+1] and y[i+1]
multiply x[i+2] and y[i+2]            multiply x[i+2] and y[i+2]
store the result to x[i+2]            multiply x[i+3] and y[i+3]
load x[i+3] to a scalar register      store the result to x[i  ]
load y[i+3] to a scalar register      store the result to x[i+1]
multiply x[i+3] and y[i+3]            store the result to x[i+2]
store the result to x[i+3]            store the result to x[i+3]
```

algorithms." (par_vec was since renamed to par_unseq)

Parallel algorithms called with this execution policy have the power to auto-vectorize the code passed to them. The problem I intend on solving is implementing two new policies hpx::par_unseq and hpx::unseq which should internally exploit this guarantee given to them to get the compiler to auto-vectorize code whenever possible.

WHY IS THIS SOMETHING WE SHOULD CARE ABOUT?

Using vector intrinsics can give a massive boost in performance on modern CPUs. Data parallel loops with dependencies that are consistent with the wavefront model can be parallelized via SIMD. Consider the following loop,

```
for(int i=0; i<n; i++){
        x = f(A[i+1]);
        A[i] = g(x, B[i]);
}
```

Such a loop cannot be parallelized over threads due to no guarantees on execution order. However, since it is consistent with the wavefront model we can vectorize such a loop and gain large performance gains. Not vectorizing is essentially leaving a major part of the performance we could gain from the CPU die dormant.

TARGET

The primary goal here is **auto-vectorization.** Different compilers have different loop optimization passes and these optimizers often work differently. Clang works using the LLVM backend, GCC has its own backend written from scratch and ICC is developed by intel but it recently adopted using LLVM backend as well. I do not have much experience with MSVC but am willing to learn the required details.

Compilers try to recognize vector patterns in code, weight pros and cons and only if they recognize that the given code can be (provably) vectorized without affecting the serial-outcome will they vectorize the code. Our job is to provide the compiler with as many hints as possible to aid them recognize these patterns.

SOLUTION

Auto loop transformations are compiler dependent, hardware dependent and also task dependent. Getting compilers to generate good vector code by auto-vectorization is truly a mystic art :) But I believe profiling and analyzing instructions generated will be a good first step.

1. **Profiling and analyzing:** Each of these compilers take hints via different syntax and use them differently. I plan on first running several micro-benchmarks to test each of the above-mentioned 4 compilers for their auto-vectorization capabilities and how they respond to different hints. I believe that profiling and analyzing the assembly they generate using tools like perf report and Intel VTune will give insightful data about how the compilers implement auto-vectorization for different hints and tasks. I will also run them against hand-typed SIMD versions of tasks if necessary to compare peak performance.
2. **Removing arch enemies of auto-vectorization:** There are certain specific patterns in code / reasons why compilers do not auto-vectorize code that looks very "vectorize-able" to a human.
    a. One of the biggest offenders is aliasing, specifically pointer aliasing. We can add some checks before the loop body to check for such aliasing (on random access iterators) and also (if acceptable) use the restrict keyword (compiler specific) to provide the compiler with information that the pointer arrays are independent.
    b. Another common enemy is memory alignment. We can try out adding loops in the beginning / end which when possible perform the computations using scalar operations till we reach a 16/32-bit boundary separately before entering the main loop. We can then hint to the compiler that this memory is always aligned to some memory boundary using compiler specific hints which aid vectorization. Compilers usually can do this themselves, but to make sure we can use the profiles we have from step 1 to make accurate decisions.

3. **Implement the execution policy:** I plan on keeping a draft PR open where I will be pushing commits, so that the community (and especially the mentors) are aware of what I am working on and can give me feedback on the same thread. I will post profile results, take feedback and use this accumulated information to implement the actual execution policy.

4. **Dabble with OpenMP & explicit vectorization libraries:** There was concern raised about using OpenMP previously as it required users to compile with -fopenmp. However, OpenMP is still one of the best ways to enforce the compiler to vectorize loops which we know can be vectorized. In the last part of the project I hope to work with and test both OpenMP and explicit vectorization libraries like VC and std::experimental::simd as suggested by @srinivasyadav227 on the IRC.

SPECIFICATIONS

Availability: Willing to spend **40 hours a week** or more post May 7th. Final semester examinations at my university conclude on May 7th. I have summer holidays after this and have no other obligations to attend apart from GSoC should I be given this opportunity.

Location / Timezone: During my vacations I will remain in India (UTC +5:30) until May 29th (beginning of community bonding period, after which I will be working from Bahrain (UTC +3:00) till around the end of August (2 month period). After this I will move back to Hyderabad, India (UTC +5:30).

Communication: I don't have any personal choices related to communication platforms. I have accounts on most platforms and wouldn't mind making a new one if required.

MILESTONES

The following are the milestones I hope to achieve over the course of this project.

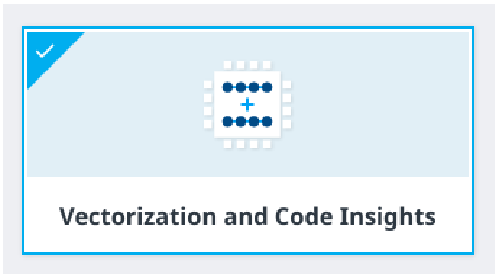
Phase 1

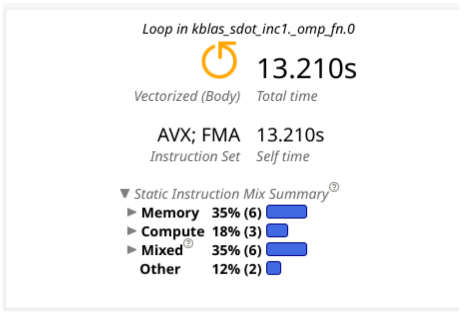
For the first phase of evaluations, I plan to have completed the following tasks.

1. Learn the compiler specific syntax and other idiosyncrasies of the MSVC compiler.

2. Profile and analyze the instructions generated by the different compilers in response to provided hints on various workloads. The goal is to start off with basic operations like dot product, copy etc. and move on to more HPX targeted workloads. I will use perf and Vtune primarily to assess output. Vtune and Intel Advisor often give valuable information about useful hints that can be inserted in code.

Advisor & VTune:



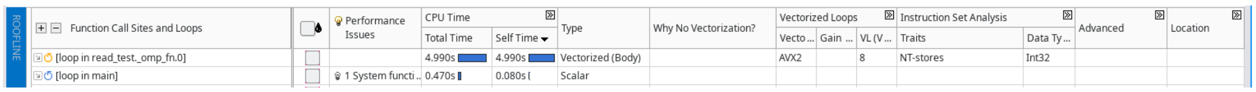

Loop in kblas_sdot_inc1._omp_fn.0

Vectorized (Body) 13.210s Total time

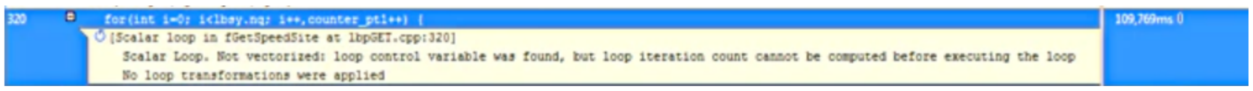
AVX; FMA Instruction Set 13.210s Self time

▼ Static Instruction Mix Summary

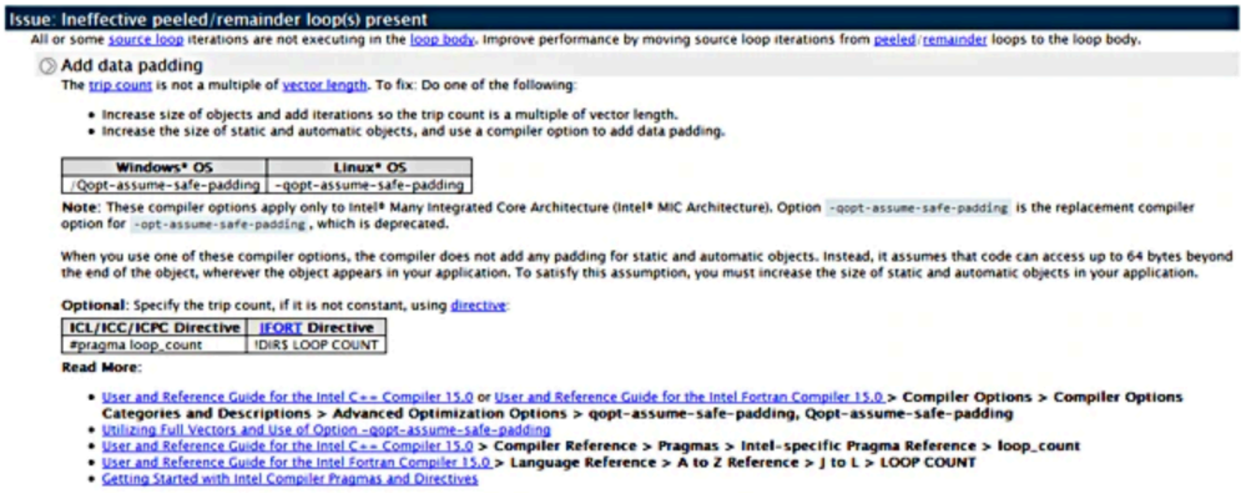
▶ Memory	35% (6)
▶ Compute	18% (3)
▶ Mixed	35% (6)
Other	12% (2)


ROOFLINE

Function Call Sites and Loops		Performance Issues	CPU Time		Type	Why No Vectorization?	Vectorized Loops			Instruction Set Analysis		Advanced	Location
			Total Time	Self Time			Vecto ...	Gain ...	VL (V ...	Traits	Data Ty ...		
[loop in read_test._omp_fn.0]			4.990s	4.990s	Vectorized (Body)		AVX2		8	NT-stores	Int32		
[loop in main]		1 System functi ..	0.470s	0.080s	Scalar								



```
320    for(int i=0; i<lbsy.nq; i++,counter_ptl++) {                                        109,769ms 0
       [Scalar loop in fGetSpeedSite at lbpGET.cpp:320]
         Scalar Loop. Not vectorized: loop control variable was found, but loop iteration count cannot be computed before executing the loop
         No loop transformations were applied
```



Issue: Ineffective peeled/remainder loop(s) present

All or some source loop iterations are not executing in the loop body. Improve performance by moving source loop iterations from peeled/remainder loops to the loop body.

Add data padding

The trip count is not a multiple of vector length. To fix: Do one of the following:

- Increase size of objects and add iterations so the trip count is a multiple of vector length.
- Increase the size of static and automatic objects, and use a compiler option to add data padding.

Windows* OS	Linux* OS
/Qopt-assume-safe-padding	-qopt-assume-safe-padding

Note: These compiler options apply only to Intel® Many Integrated Core Architecture (Intel® MIC Architecture). Option `-qopt-assume-safe-padding` is the replacement compiler option for `-opt-assume-safe-padding`, which is deprecated.

When you use one of these compiler options, the compiler does not add any padding for static and automatic objects. Instead, it assumes that code can access up to 64 bytes beyond the end of the object, wherever the object appears in your application. To satisfy this assumption, you must increase the size of static and automatic objects in your application.

Optional: Specify the trip count, if it is not constant, using directive:

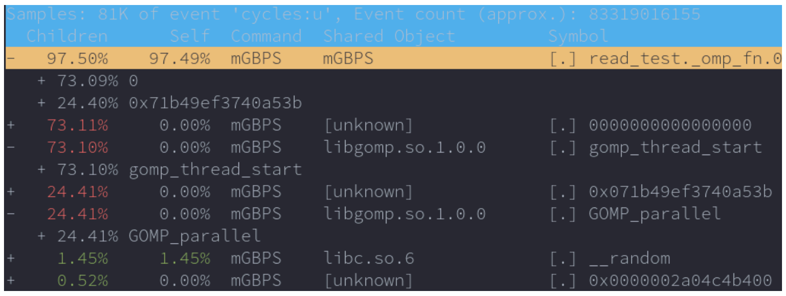
ICL/ICC/ICPC Directive	IFORT Directive
#pragma loop_count	!DIR$ LOOP COUNT

Read More:

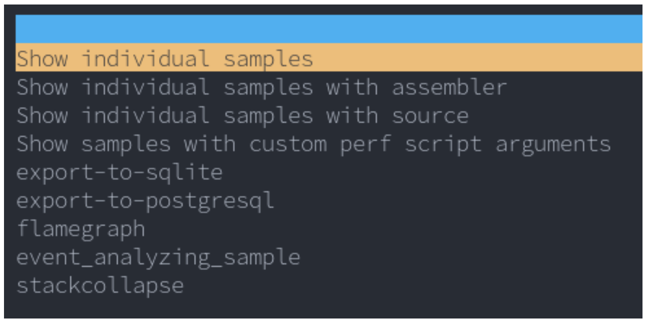
- User and Reference Guide for the Intel C++ Compiler 15.0 or User and Reference Guide for the Intel Fortran Compiler 15.0 > **Compiler Options > Compiler Options Categories and Descriptions > Advanced Optimization Options > qopt-assume-safe-padding, Qopt-assume-safe-padding**
- Utilizing Full Vectors and Use of Option -qopt-assume-safe-padding
- User and Reference Guide for the Intel C++ Compiler 15.0 > **Compiler Reference > Pragmas > Intel-specific Pragma Reference > loop_count**
- User and Reference Guide for the Intel Fortran Compiler 15.0 > **Language Reference > A to Z Reference > J to L > LOOP COUNT**
- Getting Started with Intel Compiler Pragmas and Directives

While the advice given by this tool is somewhat specific to the Intel compiler, the same principles can be extended to work for helping the other compilers auto-vectorize workloads as well. The advice given by these two tools is pretty much *exactly* what we need. Pragmas and other hints to help the compiler auto vectorize some code.

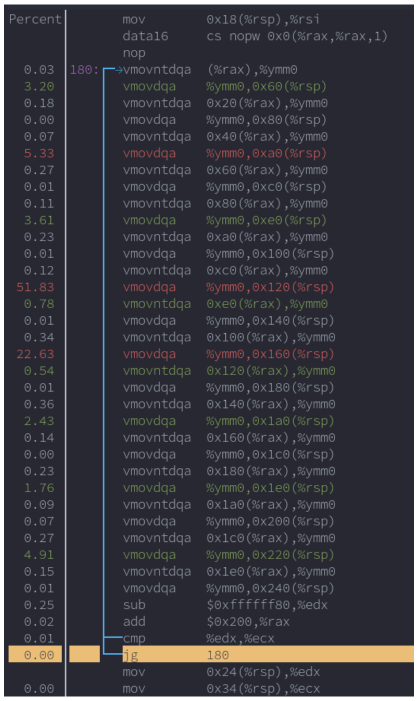
perf on the other hand is able to (with very low overhead) annotate the assembly and quickly let me explore the generated assembly for different functions. + it fits in my terminal :)



```
Samples: 81K of event 'cycles:u', Event count (approx.): 83319016155
  Children      Self  Command  Shared Object         Symbol
-   97.50%    97.49%  mGBPS    mGBPS                 [.] read_test._omp_fn.0
   + 73.09% 0
   + 24.40% 0x71b49ef3740a53b
+   73.11%     0.00%  mGBPS    [unknown]             [.] 0000000000000000
-   73.10%     0.00%  mGBPS    libgomp.so.1.0.0      [.] gomp_thread_start
   + 73.10% gomp_thread_start
+   24.41%     0.00%  mGBPS    [unknown]             [.] 0x071b49ef3740a53b
-   24.41%     0.00%  mGBPS    libgomp.so.1.0.0      [.] GOMP_parallel
   + 24.41% GOMP_parallel
+    1.45%     1.45%  mGBPS    libc.so.6             [.] __random
+    0.52%     0.00%  mGBPS    [unknown]             [.] 0x0000002a04c4b400
```



```
Show individual samples
Show individual samples with assembler
Show individual samples with source
Show samples with custom perf script arguments
export-to-sqlite
export-to-postgresql
flamegraph
event_analyzing_sample
stackcollapse
```



```
Percent│         mov        0x18(%rsp),%rsi
       │         data16     cs nopw 0x0(%rax,%rax,1)
       │         nop
  0.03 │180: ┌─→vmovntdqa  (%rax),%ymm0
  3.20 │     │  vmovdqa    %ymm0,0x60(%rsp)
  0.18 │     │  vmovntdqa  0x20(%rax),%ymm0
  0.00 │     │  vmovdqa    %ymm0,0x80(%rsp)
  0.07 │     │  vmovntdqa  0x40(%rax),%ymm0
  5.33 │     │  vmovdqa    %ymm0,0xa0(%rsp)
  0.27 │     │  vmovntdqa  0x60(%rax),%ymm0
  0.01 │     │  vmovdqa    %ymm0,0xc0(%rsp)
  0.11 │     │  vmovntdqa  0x80(%rax),%ymm0
  3.61 │     │  vmovdqa    %ymm0,0xe0(%rsp)
  0.23 │     │  vmovntdqa  0xa0(%rax),%ymm0
  0.01 │     │  vmovdqa    %ymm0,0x100(%rsp)
  0.12 │     │  vmovntdqa  0xc0(%rax),%ymm0
 51.83 │     │  vmovdqa    %ymm0,0x120(%rsp)
  0.78 │     │  vmovntdqa  0xe0(%rax),%ymm0
  0.01 │     │  vmovdqa    %ymm0,0x140(%rsp)
  0.34 │     │  vmovntdqa  0x100(%rax),%ymm0
 22.63 │     │  vmovdqa    %ymm0,0x160(%rsp)
  0.54 │     │  vmovntdqa  0x120(%rax),%ymm0
  0.01 │     │  vmovdqa    %ymm0,0x180(%rsp)
  0.36 │     │  vmovntdqa  0x140(%rax),%ymm0
  2.43 │     │  vmovdqa    %ymm0,0x1a0(%rsp)
  0.14 │     │  vmovntdqa  0x160(%rax),%ymm0
  0.00 │     │  vmovdqa    %ymm0,0x1c0(%rsp)
  0.23 │     │  vmovntdqa  0x180(%rax),%ymm0
  1.76 │     │  vmovdqa    %ymm0,0x1e0(%rsp)
  0.09 │     │  vmovntdqa  0x1a0(%rax),%ymm0
  0.07 │     │  vmovdqa    %ymm0,0x200(%rsp)
  0.27 │     │  vmovntdqa  0x1c0(%rax),%ymm0
  4.91 │     │  vmovdqa    %ymm0,0x220(%rsp)
  0.15 │     │  vmovntdqa  0x1e0(%rax),%ymm0
  0.01 │     │  vmovdqa    %ymm0,0x240(%rsp)
  0.25 │     │  sub        $0xffffff80,%edx
  0.02 │     │  add        $0x200,%rax
  0.01 │     ├──cmp        %edx,%ecx
  0.00 │     └──jg         180
       │         mov        0x24(%rsp),%edx
  0.00 │         mov        0x34(%rsp),%ecx
```

3. Use this data to implement the necessary template overloads for HPX algorithms. The starting point is for_each. After this I plan on taking community feedback on the implementation and further improving it before continuing to work on other algorithms. The next algorithms on my to-do list are: sort and transform. By this time I'm sure the mentors and the community will be able to further guide me on what other algorithms to work on next.

Final evaluations

For the final evaluations, I will choose either of the following two paths based on mentor suggestions. I will either continue implementing more algorithms from the to-do list or I will start exploring the effects of OpenMP directives and libraries like VC to explore the data parallel transforms we can make on input to aid auto-vectorization. Disclaimer, while I am comfortable with OpenMP I do not have any experience working with such a library but am definitely interested in spending as much time as required to understand and work with such a library.

Timeline

An attempt to make-up: Admittedly I have not spent as much time as I'd have liked interacting with the community. I have gone over half the lectures in the summer series and played around with the examples in the documentation. This is not much. It is my fault for not being aware of GSoC related news. I have until the 20th of May before the accepted projects get announced and university end-semester exams end on 7th May. I intend on using this in-between period to further learn and interact with the community and submit micro-patches to small issues to better familiarize myself with the codebase.

Week 1 [May 20th - May 28th]

1. During this community bonding period I primarily wish to talk to the mentors and other contributors about which algorithms are the most important to implement auto-vectorization hints for first and which tasks they think are good candidates for analyzing compiler responses to hints.

2. Familiarize myself with any new tools or libraries that I have been suggested to learn by other members of the community

3. Make a to-do list which is a concise representation of the information accumulated during the interactions and a Kanban board :) I find using them boosts my productivity a lot.

Week 2 [May 29th - June 4th]

I plan on testing all 4 compilers (and more if suggested) on various workloads with different hints and writing a **detailed analysis report** which might also serve as a resource to many others who might work on this project after me and simultaneously push any findings to the draft PR so that the mentors and other contributors are kept in the loop and can give valuable feedback.

Week 3 [June 5th - June 11th]

I will continue with testing the compilers but this time the focus will be on the HPX codebase. Optimizing the first function is always the hardest, and `for_each` is definitely abstract enough to include a lot of hurdles and discoveries that I will have to account for. Familiarizing myself with what I will be coding is the primary goal here.

Week 4 [June 12th - June 18th]

I will begin implementing the HPX execution policy for `par_unseq` for `for_each` in small incremental commits and document my progress along the way. Ideally I will finish this by the end of the week with satisfactory auto-vectorization results.

Week 5 [June 19th - June 25th]

If all goes well and both my work is completed and the community gives a positive response, I'll switch to implementing the `unseq` policy for the same function. I'll give myself a little free space here to collect feedback from the community and time to implement the suggested changes.

Week 6 [June 26th - July 2nd]

1. Create tests and documentation for both the execution policies created.

2. Finalize any last changes and polish up the code.

Week 7 [July 3rd - July 9th]

1. Perform any additional tests / analysis required for implementing the policies for sort or transform. The decision is up to what the community needs next.

2. Start implementing auto-vectorization hints for the second function.

Week 8 [July 10th - July 16th]

Continue working on implementing par_unseq and unseq execution policies for the second function.

Week 9 [July 17th - July 23rd]

Write tests and relevant documentation for the implementation of the second function (current idea: sort). Also use this period to implement any suggested changes / ponder over feedback received.

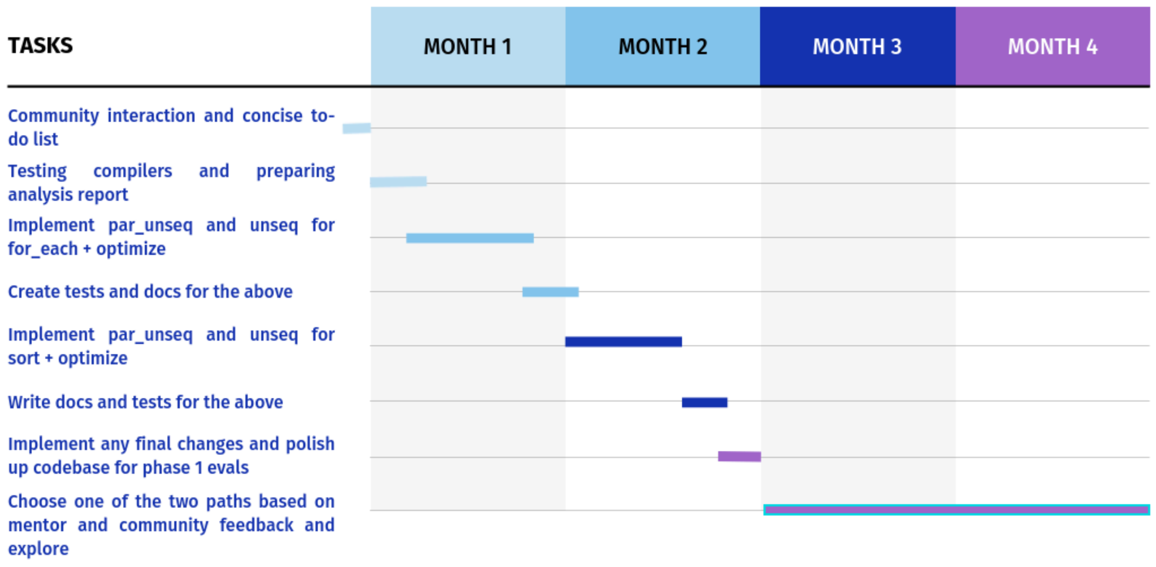
Week 10 [July 24th - July 30th]

Polish up all code written during the duration of this project, implement any last-minute requested changes and make sure everything is working for phase 1 evals. This week will also serve as a buffer week in case things don't go according to plan in any of the previous weeks and things get shifted by a week.

Note: In an ideal world I'd attempt to implement transform as well, but at the very least I'd like to be done with 2 implementations completely.

July 25th - Sept 4th

Implement a third function and choose between either of the two paths mentioned in my milestones.


TASKS
MONTH 1
MONTH 2
MONTH 3
MONTH 4
Community interaction and concise to-do list
Testing compilers and preparing analysis report
Implement par_unseq and unseq for for_each + optimize
Create tests and docs for the above
Implement par_unseq and unseq for sort + optimize
Write docs and tests for the above
Implement any final changes and polish up codebase for phase 1 evals
Choose one of the two paths based on mentor and community feedback and explore