Project 2 ispc

Group 12

Shuo Fan Yuwei Huang

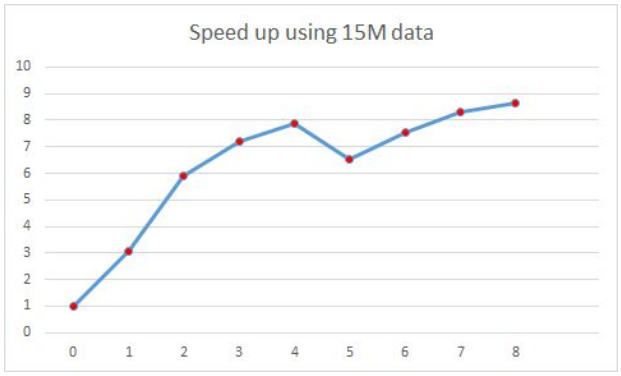
Nirali Shah Careena Braganza

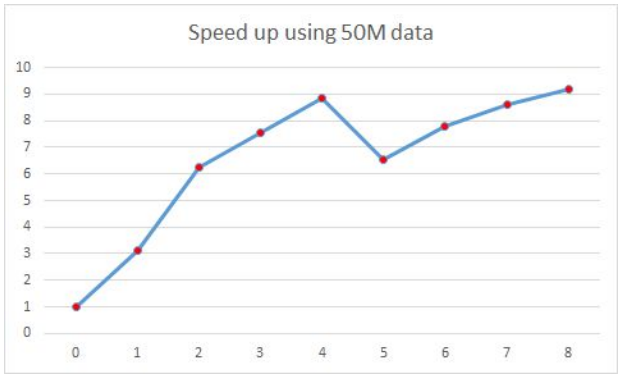
We use three size(15 million, 50 million, 80 million) of random real numbers (not integer) between 0 and 5 as input. And we use Newton method to compute sqrt. We use four sections to discuss this project.

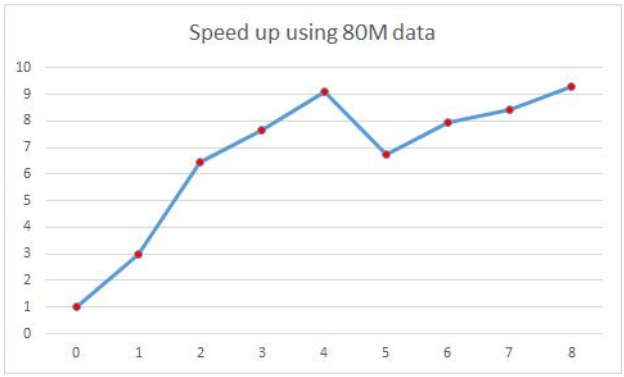
Experiment 1:
Performance using target AVX
Platform CPU: Intel Core I5, dual-core with 4 threads

Using avx2-i32x16 target, time unit ms

15m Data	seq	Single core	2 task	3 task	4 task	5 task	6 task	7 task	8 task	Avx 8 wide
Time (ms)	355.199	115.163	60.562	49.894	45.686	55.658	47.610	43.384	44.545	48.616
speedup	1	3.084	5.926	7.187	7.896	6.512	7.551	8.298	8.643	7.306
50m Data										
Time (ms)	1198.950	384.847	193.910	161.898	136.582	185.256	156.826	143.291	137.058	164.647
speedup	1	3.115	6.261	7.550	8.869	6.542	7.809	8.602	9.208	7.282
80m Data										
Time (ms)	2030.674	684.056	316.741	252.957	211.724	289.273	242.961	229.102	205.641	269.192
speedup	1	2.969	6.452	7.680	9.123	6.741	7.926	8.447	9.272	7.544







In these graph, X axis means number of tasks we launch, 0 task means sequential. Y axis means the speedup.

From the result, we can see that on the AVX target, the ISPC implementation has a quite good speedup than the sequential version. The largest speedup can be up to 9 times than the sequential version when using multi-core. When using single core, the ISPC implementation speedup is about 3 times.

The speedup of single core ISPC to the sequential implementation is due to the SIMD parallelization. So the speed up due to the SIMD is about 3 times.

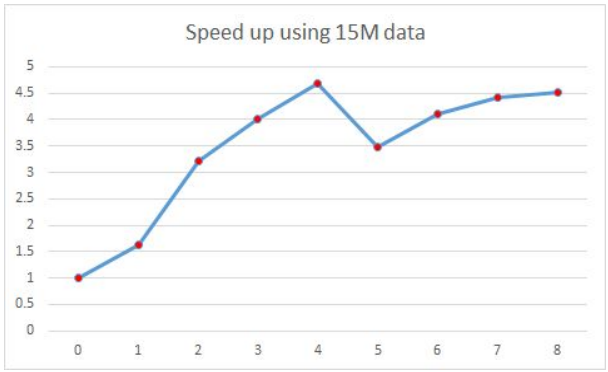
Ispc also provides facilities for parallel execution across multiple cores though an asynchronous function call mechanism via the launch keyword. A function called with launch executes as an asynchronous task, often on another core in the system.

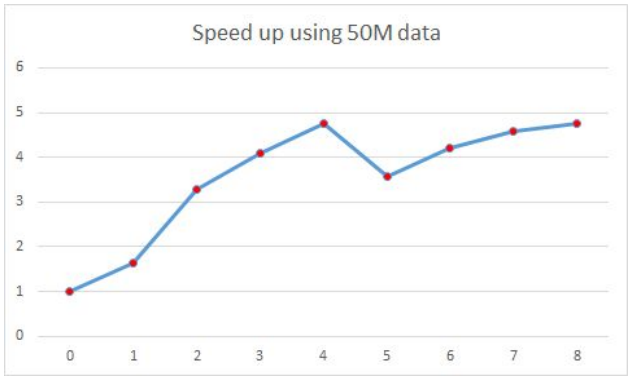
The speedup of multi-task ISPC to the single-task ISPC implementation is due to the multi-core parallelization. The speedup due to multi-core parallelization is depends on how many tasks we launch and how many hardware cores we have. The speedup due to multi-core parallelization is varied from 2 times to 3.1 times depending on the number of tasks we launched. When we launched 4 or 8 tasks, we can get the most speedup due to multi-core parallelization, about 3.1 times.

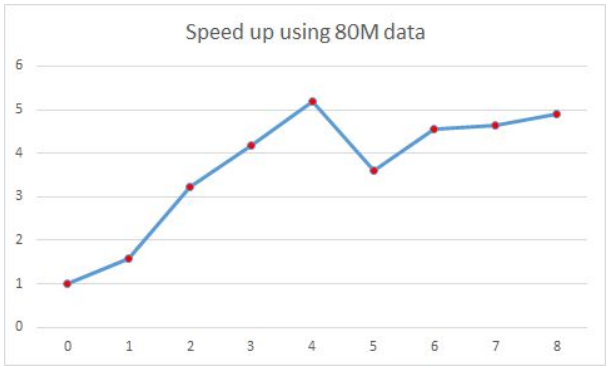
When we extended our code into more threads, we test the performance using 2,3,4,5,6,7,8 threads and produce a graph. In the graph, we can find that when the number of threads less than 4, the speedup is linear. But when the tasks are more than 4, the speedup is not linear. (we can clearly see there is a drop when # of task equals to 5) The performance using 5,6,7 tasks even has worse performance than 4 tasks. We think this is because we only have 2 cores (4 threads) on the machine. When the number of tasks less than 4, we can assign each task to each core equally. When we launch more tasks, we can use more resource of our CPU, so we can get the linear speedup. But when the number of tasks are more than 4, the tasks cannot be assigned to the 4 cores we have equally, so the performance may decrease. This may due to the unbalancing workload,some cores have more tasks so it takes longer time. We think that means when we only have 2 cores (4 threads) on our CPU, we may not get better performance if we launch tasks more than 4. So the number of tasks we launched should depend on the CPU cores we have.

Experiment 2:
Performance using target SSE
Platform CPU: Intel Core I5, dual-core with 4 threads
Using see2 target, time unit ms

15m	seq	Single core	2 task	3 task	4 task	5 task	6 task	7 task	8 task	Avx 8 wide
time	380.480	232.865	118.871	95.574	80.486	109.065	91.280	86.327	84.052	52.525
speedup	1	1.634	3.221	4.020	4.692	3.480	4.112	4.417	4.520	7.244
50m										
time	1259.85	771.10	402.209	316.223	272.575	356.337	298.984	276.748	266.991	168.498
speedup	1	1.634	3.270	4.101	4.746	3.565	4.218	4.579	4.765	7.677
80m										
time	2212.283	1406.974	639.805	505.541	400.207	593.521	478.588	438.121	408.693	260.379
speedup	1	1.572	3.224	4.168	5.183	3.593	4.552	4.648	4.902	7.821







In these graph, X axis means number of tasks we launch, 0 task means sequential. Y axis means the speedup.

When we use the SSE platform, we can get the similar result to the result on the AVX platform. The speedup is quite good and the largest speedup can be up to 5.1 times when using multi-core. When using single core, the ISPC speedup is about 1.6 times.

The speedup of single core ISPC to the sequential implementation is due to the SIMD parallelization. So the speed up due to the SIMD is about 1.6 times.

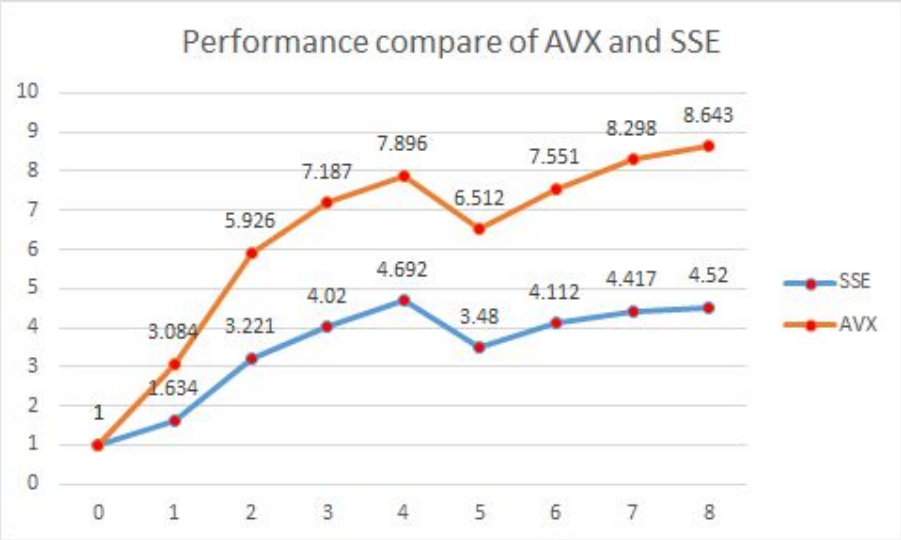
The speedup of multi-task ISPC to the single-task ISPC implementation is due to the multi-core parallelization. It varies from 2 times to 3 times.

Also when use more threads, the speedup when number of tasks less than 4 is linear. When number of tasks more than four, the speedup is not linear. The result is similar to the result of AVX implementation, the renson and conclusion should also be the same.

Experiment 3:
Performance compare of SSE target and AVX target
Platform CPU: Intel Core I5, dual-core with 4 threads
Using 15M dataset

In this table, the size of dataset is 15M.

	seq	Single core	2 task	3 task	4 task	5 task	6 task	7 task	8 task
SSE Speedup	1	1.634	3.221	4.020	4.692	3.480	4.112	4.417	4.520
AVX Speedup	1	3.084	5.926	7.187	7.896	6.512	7.551	8.298	8.643



From the result, we can see the AVX target has obviously better performance than the SSE target. When using the same number of tasks, the speedup of AVX is nearly 2 times more than the SSE target.
We checked the resources of these instruction set and found that AVX instruction set is a newer version that only be supported by new CPU released after 2011 while the SSE instruction set is older and supported by early 2000s era CPUs. The new instruction set should have some new instructions and optimizations, so it should have better performance than the old target.

Experiment 4:
Performance compare of ISPC implementation and AVX intrinsics
Platform CPU: Intel Core I5, dual-core with 4 threads
Using 15M dataset

	Sequential	Single core ISPC	AVX instrinsics 8 wide
Time (ms)	355.199	115.163	48.616
speedup	1	3.084	7.306

In this experiment, we can see when using the AVX intrinsics to implement sqrt function also has better performance than using ISPC to implement the sqrt function, the AVX intrinsics implementation is about 2.43 times faster than the ISPC implementation. That means the speedup is also depends on the different version of codes.

Here is the screenshot of the result.

```
This is sequential version vout
Elapsed processor cycles serial run :  970.711 million cycles
Sqeuential Sqrt Run time: 376.989ms

This is single cpu core(no tasks launched) vout
Time of single core run :       598.441 million cycles
Single Core(no tasks launched) Run time: 232.412ms
Speedup : 1.622

This is Multiple Cores(with 6 tasks)vout
Time of multiple core run :     232.683 million cycles
Multiple Cores(with 6 tasks) Run time: 90.365ms
Speedup : 4.172

This is the AVX intrinsics...
Time of 8-wide AVX run :        127.430 million cycles
AVX intrinsic with 6 tasks) Run time: 49.502ms
 Speedup : 7.616
```