SMID instruction and Vector Class Library from Anger Fog

Reference:
https://www.youtube.com/watch?v=TKjYdLIMTrI&list=PLKK11LigqithMn_3ipTSSTZdbLHSh5Iy3&index=3

Single Instruction Multiple Data can be considered as instructions that can operate elements reside in a large register at once. This can greatly improve the performance.

Vector Class Library

Compiler intrinsic can be used to take advantage on these registers. However, the syntax is difficult to read and maintain. VCL provide SIMD vectors of 128 bits, 256 bits, or 512 bits for int from 8 bits to 64 bits and 32/64 bits floating point operation. The instruction sets can emulate AVX512 using two 256 bits registers.

Compiler Settings

gcc -march=native -Q --help=target | grep march
Check the default option selected by the gcc compiler

https://stackoverflow.com/questions/52653025/why-is-march-native-used-so-rarely
Forum about -march option in GCC

export CPLUS_INCLUDE_PATH="/home/erebus/VCL"
Add default include path to C++ compiler

Additionally, the program needs to be compiled in 64 bits and the C++ standard should be set as c++17

g++ -std=c++17 -m64 -march=native -o binaryname filename

Simple Example

This program does a simple reduction to calculate the sum of all elements in an array which has size is not a multiple of the vector size.

```
#include <iostream>
#include <vectorclass.h>

const int datasize = 134;
const int vectorsize = 8;
const int regularpart = datasize & (-vectorsize); // = 128
// (AND - ing with -vectorsize will round down to the nearest
// lower multiple of vectorsize . This works only if vectorsize
// is a power of 2)
```

Set the data size and calculate the part that is the multiple of vector size.

Method 1

Handling the remaining data with smaller vector size

```
int i;
int mydata[datasize];
for (i = 0; i < datasize; ++i) // initialize mydata
    mydata[i] = i;

Vec8i sum1(0), temp;
int sum = 0;

// loop for 8 numbers at a time
for (i = 0; i < regularpart; i += vectorsize)
{
    temp.load(mydata + i); // load 8 elements
    sum1 += temp;          // add 8 elements
}

sum = horizontal_add(sum1); // sum of first 128 numbers

if (datasize - i >= 4)
{
    // get four more numbers
    Vec4i sum2;
    sum2.load(mydata + i);
    i += 4;
    sum += horizontal_add(sum2);
}
// loop for the remaining 2 numbers
for (; i < datasize; i++)
{
    sum += mydata[i];
}
```

Method 2

Use partial load for the last vector

```
int i;
int mydata[datasize];
for (i = 0; i < datasize; ++i) // initialize mydata
    mydata[i] = i;

Vec8i sum1(0), temp;

// loop for 8 numbers at a time
for (int i = 0; i < regularpart; i += vectorsize)
{
    temp.load(mydata + i); // load 8 elements
    sum1 += temp;          // add 8 elements
}
// load the last 6 elements
temp.load_partial(datasize - regularpart, mydata + regularpart);
sum1 += temp;                    // add last 6 elements
int sum = horizontal_add(sum1); // vector sum
```

Method 3

Read past the end of the array and ignore excess data

```
int i;
int mydata[datasize];
for (i = 0; i < datasize; ++i) // initialize mydata
    mydata[i] = i;

Vec8i sum1(0), temp;

// loop for 8 numbers at a time , reading 136 numbers
for (int i = 0; i < datasize; i += vectorsize)
{
    temp.load(mydata + i); // load 8 elements
    if (datasize - i < vectorsize)
    {
        // set excess data to zero
        // ( this may be faster than load_partial )
        temp.cutoff(datasize - i);
    }
    sum1 += temp; // add 8 elements
}
int sum = horizontal_add(sum1); // vector sum
```

Method 4

Make array bigger and set excess data to zero

```
const int arraysize = (datasize + vectorsize - 1) & (-vectorsize); // = 136
int i;
int mydata[arraysize];

for (i = 0; i < datasize; ++i) // initialize mydata
    mydata[i] = i;

// set excess data to zero
for (i = datasize; i < arraysize; ++i)
    mydata[i] = 0;

Vec8i sum1(0), temp;

// loop for 8 numbers at a time , reading 136 numbers
for (i = 0; i < arraysize; i += vectorsize)
{
    temp.load(mydata + i); // load 8 elements
    sum1 += temp;          // add 8 elements
}
int sum = horizontal_add(sum1); // vector sum
```