Vectorizing common HEP analysis algorithms

Jaydeep Nandi
Jim Pivarski
David Lange

Contents

Scalar Pogramming and Vectorization

Scalar Programming

- Works on scalar (individual elements) values of an array.
- Applies an operation over the elements via a loop, usually a **for-loop** or **while** loop.
- Low-level, and tedious to write.
- Virtually all programming languages support scalar programming.

Scalar Programming

- **For example:** If we wish to add two compatible vectors (of same shape and castable type) ***A*** and ***B*** together and store it in **C**, a typical scalar program (in Python) would look like:

```
for i in range(len(A)):
    C[i] = A[i] + B[i]
```

- It operates on each element of the arrays sequentially.

What is Vectorization?

- Converting a scalar program to a vector program.
- It is also known as Array Programming.
- Wikipedia Definition: "In computer science, **array programming languages** (also known as vector or **multidimensional** languages) generalize operations on scalars to apply transparently to vectors, matrices, and higher-dimensional arrays."
- Allows applying an operation on multiple data items simultaneously

Array Languages

- Some languages and libraries support vectorization by default. Examples include:
  - **Python: Numpy, Scipy etc.**
  - **MATLAB**
  - **GNU OCTAVE**
  - **R etc.**
- Usually they compute the vectors under the hood as efficient "**C**" or "**FORTRAN**" implementations.

Array Languages

- Recollect the addition of all elements of two vectors to give a new vector, which we implemented as a scalar code:

```
for i in range(len(A)):
    C[i] = A[i] + B[i]
```

- In an array language, it is quite simple to write:

```
C = A + B
```

So why Vectorize?

- Simple to write; no need of writing loops incessantly.
- Expressive, from a mathematical point of view.
- But those are not the major reason for vectorising code.
  - Most modern CPUs and GPUs provide support for vectorised code, which runs very efficiently, operating in a SIMD style.
  - Can take advantage of SSE (Streaming SIMD Extensions) and AVX instructions, which operate on 4, 8 or more data simultaneously.
  - GPUs take it even further: Can operate on a large number, typically in thousands, of data at once. (More on it later!)

Autovectorization

- Modern compilers allow the vectorization of simple sequential loops into efficient SIMD instructions for CPU, typically through a compiler switch.
- Some compilers which support this:
  - **GCC**: Supports through switch ***–ftree-vectorize*** or **–O3** level optimization.
  - **ICC:** From Intel. Reported to be better than GCC at vectorising code.
  - **MSVC and others.**
- They check if a loop is safe to be vectorised, and if it is, they vectorise the code. The list of such possible vectorizable loops fo GCC can be found here.

Autovectorization

- As an example, consider again the addition of two vectors. A simple c loop that does is :

```
for ( int i=0; i<length(A); i++)
     C[i] = A[i] + B[i];
```

- The corresponding assembly dump (relevant part) is:

```
vmovdqu xmm0, XMMWORD PTR [r9+rax]
add edx, 1
vinserti128 ymm0, ymm0, XMMWORD PTR [r9+16+rax], 0x1
vpaddd ymm0, ymm0, YMMWORD PTR [r13+0+rax]
vmovups XMMWORD PTR [rcx+rax], xmm0
```

- Note the vector instructions in the assembly (**vmovdqu**, **vinserti** etc)

Autovectorization

- Advantages:
  - Sufficiently simpler than writing SSE or AVX instructions.
  - Easy to debug.
  - Portable.
- There is however, a major disadvantage. They cannot vectorise a loop if there is a loop carried dependency.

Barriers to Autovectorization

- The major barrier is a loop carried dependency. They are variables whose particular value depends on the order of the execution.
- As an example consider the problem of finding the max value of all elements in an array **A**.

```
for (int i=0; i<N; i++)
    maxval = max(maxval, A[i]);
```

- The value in maxval at any instance depends on the order of loop execution. So, it's a loop carried dependency, and the compiler cannot autovectorize it.

Barriers to Autovectorization

- The assembly output of the loop is

```
mov edx, DWORD PTR [rax]
add rax, 4
cmp ecx, edx
jl .L12
cmp rax, rsi
jne .L11
mov edx, ecx
```

- Note the absence of vector instructions in the assembly, indicating an unvectorized loop.
- This can be suitably vectorised, but it will be dealt in a later section (please bear with it!)

- Todo next:
  - Vectorization and GPUs
  - Examples
    - Product
    - Local Reduction
  - Link to markdown report
  - Conclude