Theoretical and practical limitations of some real-world computer

Theoretical limitations

In my case, we are using AMD Ryzen 7 5700U with Radion graphics. The AMD Ryzen 7 5700U with Radeon Graphics. The AMD Ryzen 7 5700U is part of the Zen 2 architecture based on the x86_64 instruction set. It is a mobile processor designed for laptops and ultrabooks. The CPU has 8 cores and supports simultaneous multithreading (SMT), allowing for a total of 16 threads. It has a base clock frequency of 1800 MHz and can boost up to a maximum frequency of 4369.9209 MHz. The CPU supports various instruction set extensions such as SSE, SSE2, SSE4.1, SSE4.2, AVX, AVX2, and BMI2, enabling efficient multimedia processing and floating-point operations. It also includes a floating-point unit (FPU) for performing floating-point arithmetic operations. The CPU has 2 threads per core.

CPU clock frequency	1800 MHz
Instructions per cycle	4
Cores	8
Threads per core	2
Number of floating operations of CPU	89.6 GFLOPS
L2 cache	4mb
L3 cache	8mb

Practical limitations

This code is used for calculating FLOPs of the CPU

```cpp
#include <iostream>
#include <vector>
#include <thread>
#include <chrono>

void perform_operations(long long start, long long end, double &result)
{
   for (long long i = start; i < end; ++i) {
       result += (i * 0.1) / (i + 1.0);
   }
}

int main() {
   auto start_time = std::chrono::high_resolution_clock::now();
   const long long max_iterations = 5e10;
   const unsigned num_threads = std::thread::hardware_concurrency();

   // Each thread will store its result here
   std::vector<double> results(num_threads, 0.0);
   // Start all threads
   std::vector<std::thread> threads;
   for (unsigned i = 0; i < num_threads; ++i) {
       long long start = i * (max_iterations / num_threads);
       long long end = (i + 1) * (max_iterations / num_threads);
       threads.push_back(std::thread(perform_operations, start, end,
std::ref(results[i])));
   }
   // Wait for all threads to finish
   for (auto &thread : threads) {
       thread.join();
   }
   // Combine the results from each thread
   double result = 0.0;
   for (const auto &partial_result : results) {
       result += partial_result;
   }

   auto end_time = std::chrono::high_resolution_clock::now();
   std::chrono::duration<double> duration = end_time - start_time;

   std::cout << "Result: " << result << std::endl;
   std::cout << "Time taken: " << duration.count() << "s" << std::endl;
   // Estimating the FLOPS
   long long total_operations = 3 * max_iterations; // we perform 3 
operations per iteration
   double flops = total_operations / duration.count();
   std::cout << "Estimated FLOPS: " << flops << std::endl;

   return 0;
}
```

This bash script (appendix 1) is used for retrieving useful information. In our case, we ran the compiler with the following flags:

```
g++ -mavx2 -mfma -O3 test.cc -o test
```

We can analyze the assembly code (appendix 2).
Code is using the hardware concurrency feature of the std::thread library, which suggests that it's trying to use all available cores on the machine for the computation. After creating the threads, it's joining them all (waiting for them to finish) and then seems to be adding up some kind of results stored in memory, possibly returned by the threads. It ends with a computation result which is returned.
We can see our foo function code:

```
foo(long long):
...
    call    std::thread::hardware_concurrency()  ; Determine the number 
of threads that can potentially run concurrently.
    vxorps  xmm0, xmm0, xmm0                     ; Set xmm0 to zero.
    mov     ebx, eax                             ; Save the result of 
the previous function call in ebx.
    test    rbx, rbx                             ; Test whether rbx is 
zero.
...
    cqo                                          ; Sign-extend rax into 
rdx:rax.
    idiv    rbx                                  ; Divide rdx:rax by 
rbx.
...
    vaddsd  xmm0, xmm0, QWORD PTR [r14]          ; Add the value pointed 
to by r14 to xmm0.
...
```

And

```
perform_operations(long long, long long, double&):
    cmp     rdi, rsi                        ; Compare the two long long
parameters.
    jge     .L5                             ; Jump if the first parameter is
not less than the second one.
    vmovsd  xmm2, QWORD PTR [rdx]           ; Load the double parameter into
xmm2.
    vmovsd  xmm4, QWORD PTR .LC0[rip]       ; Load a constant value into
xmm4.
    vxorps  xmm5, xmm5, xmm5                ; Set xmm5 to zero.
    vmovsd  xmm3, QWORD PTR .LC1[rip]       ; Load a constant value into
xmm3.
.L3:
    vcvtsi2sd       xmm0, xmm5, rdi         ; Convert rdi to a double, and
store it in xmm0.
    add     rdi, 1                          ; Increment rdi.
    vmulsd  xmm1, xmm0, xmm4                ; Multiply xmm0 by xmm4 and
store the result in xmm1.
    vaddsd  xmm0, xmm0, xmm3                ; Add xmm3 to xmm0.
    vdivsd  xmm0, xmm1, xmm0                ; Divide xmm1 by xmm0 and store
the result in xmm0.
    vaddsd  xmm2, xmm2, xmm0                ; Add xmm0 to xmm2.
    cmp     rsi, rdi                        ; Compare rsi and rdi.
    jne     .L3                             ; Repeat the loop if they are
not equal.
    vmovsd  QWORD PTR [rdx], xmm2           ; Store the result from xmm2
back to the double variable.
.L5:
    ret                                    ; Return from the function.
```

After looking into the code we can say that program operates as intended.

After execution, we got this:

```
cpu MHz         : 1800.000 MHz
Result: 5e+09
Time taken: 9.472s
Estimated FLOPS: 1.58263e+10
        Command being timed: "./test"
        User time (seconds): 153.79
        System time (seconds): 0.16
        Percent of CPU this job got: 1263%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:09.47
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 3936
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 180
        Voluntary context switches: 6
        Involuntary context switches: 29569
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
Result: 5e+09
Time taken: 11.6459s
Estimated FLOPS: 1.28178e+10

 Performance counter stats for './test':

   440,029,311,219      cycles
   400,615,955,976      instructions              #    0.91  insn per
cycle
        83,636,824      cache-references
        26,217,941      cache-misses              #   31.347 % of all
cache refs
               168      faults
             1,033      migrations

      14.976474908 seconds time elapsed

     157.693824000 seconds user
       0.261334000 seconds sys
```

We can see that we have 10 GFLOPS. That means that the program used only one core for program execution.

Comparison with CP solution

Here we have been using the CP3a solution. After running it we got. This results.

```
cpu MHz         : 1829.511 MHz
Elapsed time: 6.63833 s
```

```
FLOPS: 3.0103e+09
        Command being timed: "./cp"
        User time (seconds): 8.62
        System time (seconds): 0.04
        Percent of CPU this job got: 99%
        Elapsed (wall clock) time (h:mm:ss or m:ss): 0:08.67
        Average shared text size (kbytes): 0
        Average unshared data size (kbytes): 0
        Average stack size (kbytes): 0
        Average total size (kbytes): 0
        Maximum resident set size (kbytes): 32636
        Average resident set size (kbytes): 0
        Major (requiring I/O) page faults: 0
        Minor (reclaiming a frame) page faults: 7506
        Voluntary context switches: 1
        Involuntary context switches: 145
        Swaps: 0
        File system inputs: 0
        File system outputs: 0
        Socket messages sent: 0
        Socket messages received: 0
        Signals delivered: 0
        Page size (bytes): 4096
        Exit status: 0
Elapsed time: 2.69469 s
FLOPS: 4.49177e+09

 Performance counter stats for './cp':

    13,351,601,245      cycles
    44,526,202,829      instructions              #    3.33  insn per
cycle
       525,431,478      cache-references
        13,258,797      cache-misses              #    2.523 % of all
cache refs
             7,495      faults
                79      migrations

       3.536560769 seconds time elapsed

       3.500115000 seconds user
       0.027937000 seconds sys
```

Comparing to the previous part we can see a significant drop in the performance as only 5.8 GFLOPS is used. Nearly 2 times drop in performance.
Where several code parts can explain performance drop:

1) First is memory access:

```
.L60:
  vcvtss2sd xmm0, xmm3, DWORD PTR [rax]
  add rax, 4
  vaddsd xmm1, xmm1, xmm0
  cmp rbx, rax
  jne .L60
```

   this kind of memory accesses could potentially cause slowdowns because of the cost of reading data from memory. This cost is especially high if the data is not in the cache, causing a cache miss.

2) Code has a number of busted loops that contribute to a performance drop:

```
.L6:
  mov rbx, QWORD PTR [rsp-8]
  vmulsd xmm6, xmm7, QWORD PTR [r10+r15*8]
  xor esi, esi
  ...
  ...
  ...
  cmp r15, rbx
  jne .L6
```

3) Arithmetic operations:

```
.L4:
  vcvtss2sd xmm0, xmm3, DWORD PTR [r15]
  vsubsd xmm8, xmm0, xmm5
  vcvtss2sd xmm0, xmm3, DWORD PTR [r15+rdx*4]
  vsubsd xmm0, xmm0, xmm4
  add r15, 4
  vfmadd231sd xmm2, xmm8, xmm0
  cmp rcx, r15
  jne .L4
```

   Code contains such complex operations as multiplication and division, which are difficult for CPU.

Sources

[1] https://www.amd.com/en/products/apu/amd-ryzen-7-5700u

[2]
https://stackoverflow.com/questions/6289745/how-to-compute-the-theoretical-peak-performance-of-cpu#:~:text=here%20the%20formula%3A,number%20of%20CPUs%20per%20node)
[3] https://lhcb.github.io/developkit-lessons/first-development-steps/06a-perf.html
[4] https://preshing.com/20110723/finding-bottlenecks-by-random-breaking/

Appendix 1

```
#!/bin/bash

# Compile the C++ test
g++ -mavx2 -mfma -O3 test.cc -o test

# Get the CPU frequency
CPU_FREQ=$(cat /proc/cpuinfo | grep Hz)

echo "CPU Frequency: $CPU_FREQ MHz"

# Run the test and measure the time
/usr/bin/time -v ./test

# Collect some statistics
sudo perf stat -e
cycles,instructions,cache-references,cache-misses,faults,migrations
./test
```

Appendix 2

```
perform_operations(long long, long long, double&):
        cmp     rdi, rsi
        jge     .L53
        vmovsd  xmm2, QWORD PTR [rdx]
        vmovsd  xmm4, QWORD PTR .LC0[rip]
        vxorps  xmm5, xmm5, xmm5
        vmovsd  xmm3, QWORD PTR .LC1[rip]
.L3:
        vcvtsi2sd       xmm0, xmm5, rdi
        add     rdi, 1
        vmulsd  xmm1, xmm0, xmm4
        vaddsd  xmm0, xmm0, xmm3
        vdivsd  xmm0, xmm1, xmm0
        vaddsd  xmm2, xmm2, xmm0
        cmp     rsi, rdi
        jne     .L3
        vmovsd  QWORD PTR [rdx], xmm2
.L5:
        ret
.LC2:
        .string "vector::_M_realloc_insert"
foo(long long):
        push    r15
        push    r14
        push    r13
        push    r12
        push    rbp
        mov     rbp, rdi
        push    rbx
        sub     rsp, 88
        call    std::thread::hardware_concurrency()
        vxorps  xmm0, xmm0, xmm0
        mov     ebx, eax
        test    rbx, rbx
        je      .L45
        lea     r14, [0+rbx*8]
        mov     rdi, r14
        mov     QWORD PTR [rsp+16], r14
        call    operator new(unsigned long)
        mov     rdx, r14
        xor     esi, esi
        mov     rdi, rax
        lea     r15, [rax+r14]
        mov     QWORD PTR [rsp+24], rax
        mov     r12, rax
        mov     r14, rax
        call    memset
        mov     rax, rbp
        vpxor   xmm0, xmm0, xmm0
        xor     ebp, ebp
        cqo
        vmovdqa XMMWORD PTR [rsp+48], xmm0
        idiv    rbx
        mov     QWORD PTR [rsp+64], 0
        xor     ebx, ebx
        mov     QWORD PTR [rsp+8], rax
        jmp     .L56
.L49:
        mov     QWORD PTR [rbx], 0
        mov     rax, QWORD PTR [rsp+32]
        add     rbx, 8
        add     r12, 8
        mov     QWORD PTR [rbx-8], rax
        mov     QWORD PTR [rsp+56], rbx
        cmp     r15, r12
        je      .L90
.L56:
        mov     rax, QWORD PTR [rsp+8]
        mov     edi, 40
        mov     r13, rbp
        mov     QWORD PTR [rsp+32], 0
        add     rbp, rax
        call    operator new(unsigned long)
        mov     QWORD PTR [rax+8], r12
        mov     edx, OFFSET FLAT:_ZNSt6thread24_M_thread_deps_never_runEv
        lea     rsi, [rsp+40]
        lea     rdi, [rsp+32]
        mov     QWORD PTR [rax], OFFSET FLAT:vtable for
std::thread::_State_impl<std::thread::_Invoker<std::tuple<void (*)(long long,
long long, double&), long long, long long, std::reference_wrapper<double> > >
>+16
        mov     QWORD PTR [rax+16], rbp
        mov     QWORD PTR [rax+24], r13
        mov     QWORD PTR [rax+32], OFFSET FLAT:perform_operations(long long,
long long, double&)
        mov     QWORD PTR [rsp+40], rax
        call
std::thread::_M_start_thread(std::unique_ptr<std::thread::_State,
std::default_delete<std::thread::_State> >, void (*)())
        mov     rdi, QWORD PTR [rsp+40]
        test    rdi, rdi
        je      .L48
        mov     rax, QWORD PTR [rdi]
        call    [QWORD PTR [rax+8]]
.L48:
        cmp     QWORD PTR [rsp+64], rbx
        jne     .L49
        lea     rdx, [rsp+32]
        mov     rsi, rbx
        lea     rdi, [rsp+48]
        call    void std::vector<std::thread, std::allocator<std::thread>
>::_M_realloc_insert<std::thread>(__gnu_cxx::__normal_iterator<std::thread*,
std::vector<std::thread, std::allocator<std::thread> > >, std::thread&&)
        cmp     QWORD PTR [rsp+32], 0
        jne     .L61
        add     r12, 8
        mov     rbx, QWORD PTR [rsp+56]
        cmp     r15, r12
        jne     .L56
.L90:
        mov     r12, QWORD PTR [rsp+48]
        cmp     r12, rbx
        je      .L57
        mov     rbp, r12
.L58:
        mov     rdi, rbp
        call    std::thread::join()
        add     rbp, 8
        cmp     rbx, rbp
        jne     .L58
.L57:
        vxorps  xmm0, xmm0, xmm0
.L59:
        vcvtss2sd       xmm0, xmm0, xmm0
        vaddsd  xmm0, xmm0, QWORD PTR [r14]
        add     r14, 8
        vcvtsd2ss       xmm0, xmm0, xmm0
        cmp     r15, r14
        jne     .L59
        cmp     r12, rbx
        je      .L60
        mov     rax, r12
.L62:
        cmp     QWORD PTR [rax], 0
        jne     .L61
        add     rax, 8
        cmp     rax, rbx
        jne     .L62
```

```
.L60:
        test    r12, r12
        je      .L68
        mov     rsi, QWORD PTR [rsp+64]
        mov     rdi, r12
        vmovss  DWORD PTR [rsp+8], xmm0
        sub     rsi, r12
        call    operator delete(void*, unsigned long)
        vmovss  xmm0, DWORD PTR [rsp+8]
.L68:
        mov     rsi, QWORD PTR [rsp+16]
        mov     rdi, QWORD PTR [rsp+24]
        vmovss  DWORD PTR [rsp+8], xmm0
        call    operator delete(void*, unsigned long)
        vmovss  xmm0, DWORD PTR [rsp+8]
.L45:
        add     rsp, 88
        pop     rbx
        pop     rbp
        pop     r12
        pop     r13
        pop     r14
        pop     r15
        ret
.L61:
        call    std::terminate()
        mov     rbx, rax
        jmp     .L63
        mov     rbx, rax
        jmp     .L51
        mov     rbx, rax
        vzeroupper
        jmp     .L53
foo(long long) [clone .cold]:
.LC7:
        .string "Result: "
.LC8:
        .string "Time taken: "
.LC9:
        .string "s"
.LC11:
        .string "Estimated FLOPS: "
```

```
main:
        push    rbx
        sub     rsp, 16
        call    std::chrono::_V2::system_clock::now()
        movabs  rdi, 50000000000
        mov     rbx, rax
        call    foo(long long)
        vmovss  DWORD PTR [rsp+12], xmm0
        call    std::chrono::_V2::system_clock::now()
        mov     edx, 8
        mov     esi, OFFSET FLAT:.LC7
        mov     edi, OFFSET FLAT:_ZSt4cout
        sub     rax, rbx
        vxorps  xmm1, xmm1, xmm1
        vcvtsi2sd       xmm1, xmm1, rax
        vdivsd  xmm2, xmm1, QWORD PTR .LC6[rip]
        vmovsd  QWORD PTR [rsp], xmm2
        call    std::basic_ostream<char, std::char_traits<char> >&
std::__ostream_insert<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&, char const*, long)
        vmovss  xmm0, DWORD PTR [rsp+12]
        mov     edi, OFFSET FLAT:_ZSt4cout
        vcvtss2sd       xmm0, xmm0, xmm0
        call    std::basic_ostream<char, std::char_traits<char> >&
std::basic_ostream<char, std::char_traits<char> >::_M_insert<double>(double)
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >&
std::endl<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&) [clone .isra.0]
        mov     edx, 12
        mov     esi, OFFSET FLAT:.LC8
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >&
std::__ostream_insert<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&, char const*, long)
        vmovsd  xmm0, QWORD PTR [rsp]
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >&
std::basic_ostream<char, std::char_traits<char> >::_M_insert<double>(double)
        mov     edx, 1
        mov     esi, OFFSET FLAT:.LC9
        mov     rbx, rax
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >&
std::__ostream_insert<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&, char const*, long)
        mov     rdi, rbx
        call    std::basic_ostream<char, std::char_traits<char> >&
std::endl<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&) [clone .isra.0]
        mov     edx, 17
        mov     esi, OFFSET FLAT:.LC11
        mov     edi, OFFSET FLAT:_ZSt4cout
        vmovsd  xmm0, QWORD PTR .LC10[rip]
        vdivsd  xmm0, xmm0, QWORD PTR [rsp]
        vmovsd  QWORD PTR [rsp], xmm0
        call    std::basic_ostream<char, std::char_traits<char> >&
std::__ostream_insert<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&, char const*, long)
        vmovsd  xmm0, QWORD PTR [rsp]
        mov     edi, OFFSET FLAT:_ZSt4cout
        call    std::basic_ostream<char, std::char_traits<char> >&
std::basic_ostream<char, std::char_traits<char> >::_M_insert<double>(double)
        mov     rdi, rax
        call    std::basic_ostream<char, std::char_traits<char> >&
std::endl<char, std::char_traits<char> >(std::basic_ostream<char,
std::char_traits<char> >&) [clone .isra.0]
        add     rsp, 16
        xor     eax, eax
        pop     rbx
        ret
typeinfo name for
std::thread::_State_impl<std::thread::_Invoker<std::tuple<void (*)(long long,
long long, double&), long long, long long, std::reference_wrapper<double> > >
>:
        .string
"NSt6thread11_State_implINS_8_InvokerISt5tupleIJPFvxxRdExxSt17reference_wrappe
rIdEEEEEEE"
typeinfo for std::thread::_State_impl<std::thread::_Invoker<std::tuple<void
(*)(long long, long long, double&), long long, long long,
std::reference_wrapper<double> > > >:
        .quad   vtable for __cxxabiv1::__si_class_type_info+16
        .quad   typeinfo name for
std::thread::_State_impl<std::thread::_Invoker<std::tuple<void (*)(long long,
long long, double&), long long, long long, std::reference_wrapper<double> > >
>
        .quad   typeinfo for std::thread::_State
vtable for std::thread::_State_impl<std::thread::_Invoker<std::tuple<void
(*)(long long, long long, double&), long long, long long,
std::reference_wrapper<double> > > >:
        .quad   0
        .quad   typeinfo for
std::thread::_State_impl<std::thread::_Invoker<std::tuple<void (*)(long long,
long long, double&), long long, long long, std::reference_wrapper<double> > >
>
        .quad   std::thread::_State_impl<std::thread::_Invoker<std::tuple<void
(*)(long long, long long, double&), long long, long long,
std::reference_wrapper<double> > > >::~_State_impl() [complete object
destructor]
        .quad   std::thread::_State_impl<std::thread::_Invoker<std::tuple<void
(*)(long long, long long, double&), long long, long long,
std::reference_wrapper<double> > > >::~_State_impl() [deleting destructor]
        .quad   std::thread::_State_impl<std::thread::_Invoker<std::tuple<void
(*)(long long, long long, double&), long long, long long,
std::reference_wrapper<double> > > >::_M_run()
.LC0:
        .long   -1717986918
        .long   1069128089
.LC1:
        .long   0
        .long   1072693248
.LC6:
        .long   0
        .long   1104006501
.LC10:
        .long   771751936
        .long   1111586393
```