Hashing as fast and as much as possible

Uzochi Dimkpa
High-Performance Computing
Prof. E. Saule

The project idea itself

How many hashes can I calculate in as little time as possible (1 sec)?

How best can I utilize computer architecture to improve throughout (hashes / sec)?

Why this project idea?

- (Relatively) simple to understand

- Straightforward optimizations goals
  - Throughput
  - Memory management

- More **focus where it needs to be** (Proper coding practices & optimizations)

Why do we care about this topic?

- Hashing is very widely used throughout the field of general computer science
  - **Cybersecurity** (file verification/transport)
  - **Cryptography** (protect file content)
  - **Search/Storage algorithms** (data indexing, storage data structure; Bloom filter)
  - **Web3** (blockchain, smart contracts, NFTs, etc)

- Produces a **large range** of **unique (semi-meaningless) data**
  - Calculating hashes can be (variably) trivial and consistent
  - **Measuring performance is straightforward**

The **Gold Standard**!

The first model

A wild guess

- Just needed get it done

- **no optimization flags;**
- somewhat-intelligent **memory management** (All I know is `delete[]`)

- Expecting:

1,000,000 hashes/sec

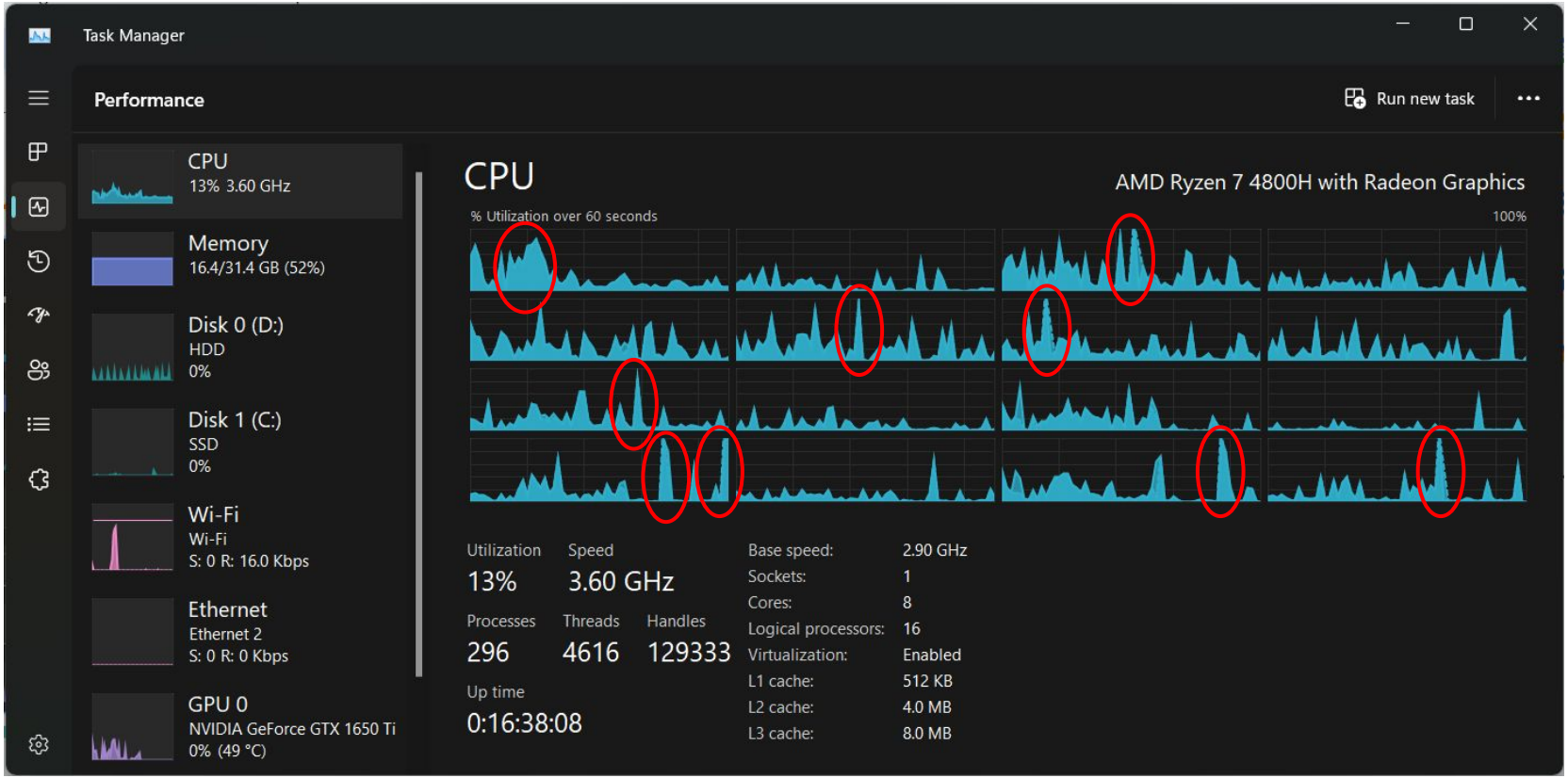
Completely off

Of course

```
input message:
abc

input digest:
4b eb 1 f7 6f 1 ec 5e 78 2f d0 d5 9f c1 d6 f7


Found a working input message!
Second Pre-Image Resistance has been broken!
Here is an input that produces a matching hash:
!!!   [--->   abc   <---]   !!!
along with its matching digest:
+=+   [4b eb 1 f7 6f 1 ec 5e 78 2f d0 d5 9f c1 d6 f7]   +=+
It took 17944236 guess(es)


Time elapsed (s): 180.037
# Guesses / sec: 99669.9
```

- Off by **a factor of 10**
- **Readjusting as necessary**

Work-sharing?



Architecture 1

CPU

Optimizations

The second model

More well-informed

- Seemed to follow the same pattern as the gold standard
  - **Performance multiplied** (roughly) by **# of cores**

- Expecting:

  160,000 hashes/sec

Optimizations include:

- Macros in place of functions
  - Maybe the **performance adds up a little over time?**

- Pthreads
  - **Assuming OpenMP is slower** (I don’t know if this is the case)

Half off

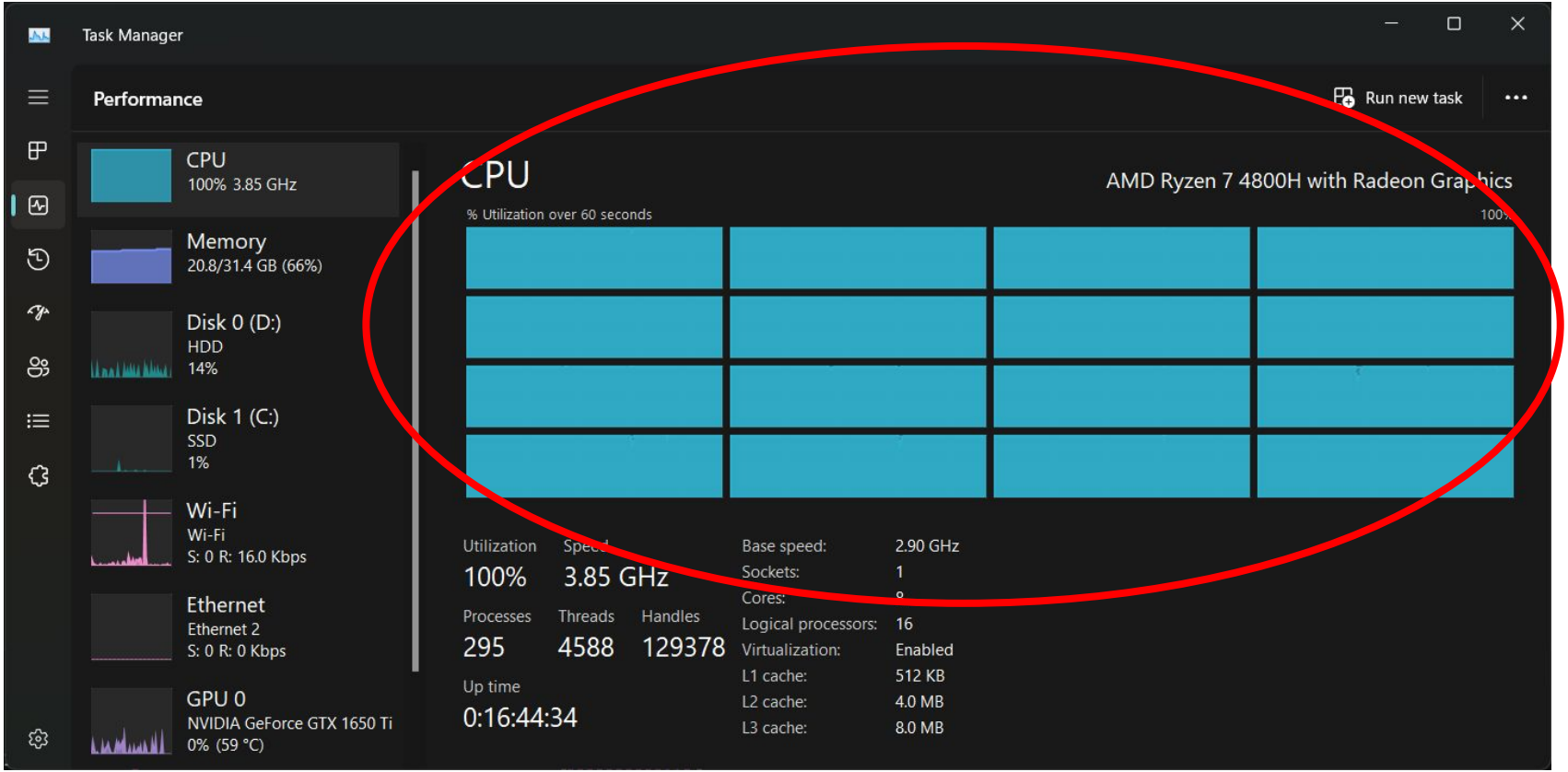
Could be several reasons

main_opt_.out

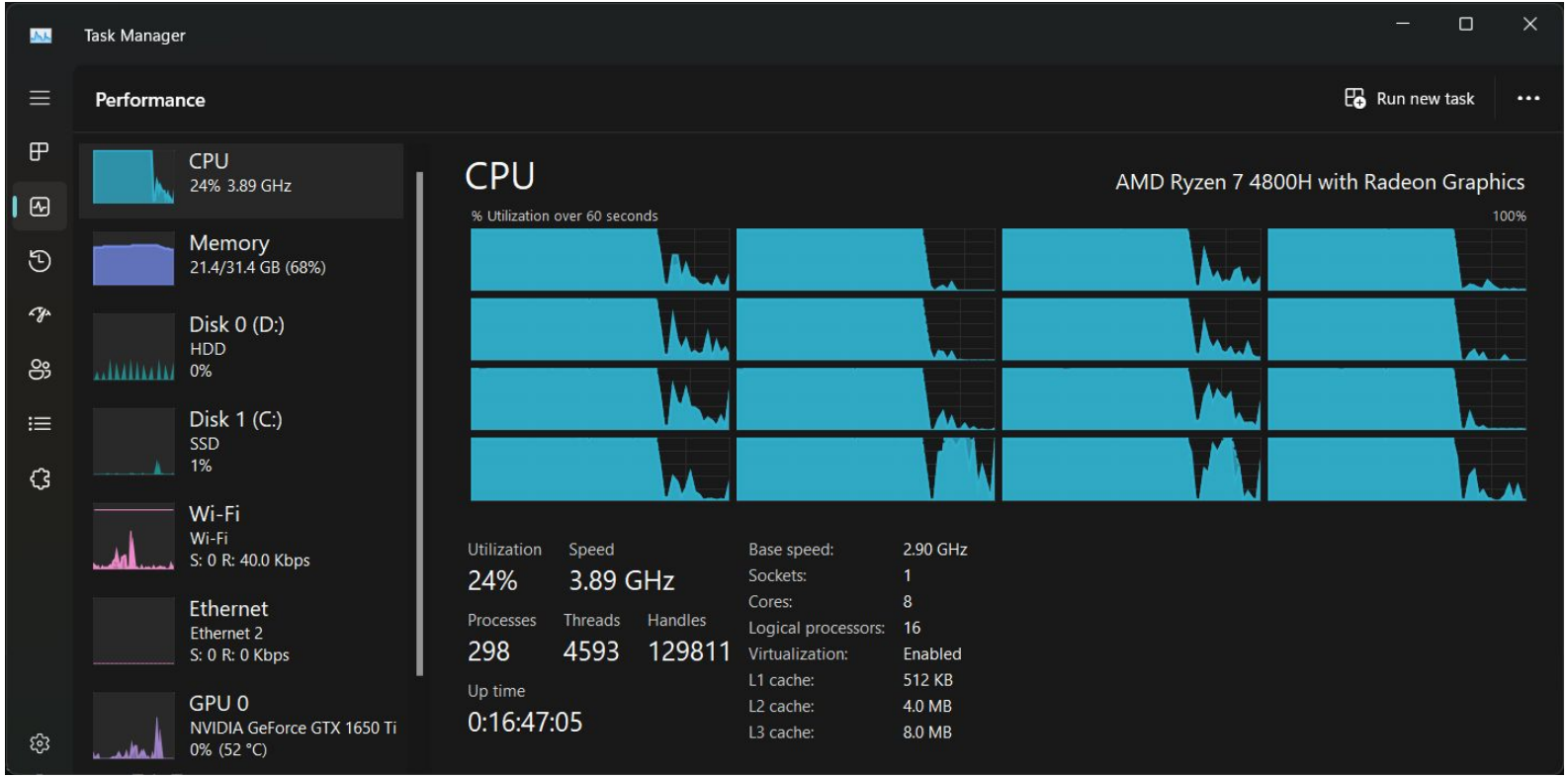
```
It took 291042921 guess(es)


Time elapsed (s): 360.488
# Guesses / sec: 807358

```

- Too much **type-casting**?
- Could I have used **intrinsics**?
- Possibly **rework bit-shifts**?
- Bad **memory management**?
- Are **strings** just **slow**?

A great picture



Another great picture



Architecture 2

GPU

Optimizations

The third model

Another wild guess

- Seemed to follow the same pattern as the gold standard
-
- Meaning: **performance multiplied** (roughly) by **# of cores**

- Expecting:

1,000,000 hashes/sec

Optimizations include:

- **CUDA**
  - GPUs are great at throughput, which is **exactly what I need**
- **Shared memory**
  - Input message, message digest, sine constants, bit-shift amount per round
  - By far the **best optimization**
- Macros
  - Variables, functions
- Memory management
  - The **main clincher** in working with larger messages; took the **most time** to fix
- Parallel populating array with data
  - Multithreaded (in one case); still **paying a small price** of **syncing threads**
- Less type-casting
  - Everything is basically an **unsigned char** or **unsigned int**

```
msgLength: 3, paddedLength: 512
CUDA error: no error
CUDA error: no error
CUDA error: no error
CUDA error: no error

guessFound on thread 75!

numGuesses : 1646
```

Nifty

To be expected; but **my architecture 2 report is wrong**

```
Thread 4095 made 2387guesses!


Time elapsed (s): 5.87285
# Guesses / sec: 1.42424e+06
```

- Make **cuRAND state generation local**
- Most definitely hit a **sweet spot**; unsure **why**
  - GPU performance **seems dependent on message length**
  - Possibly due to GPU's **memory management system?**

No picture...

404

Not found

sorry!

So,

what was

the whole point?

I chose a simple problem on purpose

- I enjoyed the project!
  - Challenging
  - Good work-pace for **understanding code-optimization**

- Mostly scratched the surface
  - I’ve looked at **other implementations**
  - My work is mostly surface-level, basic in comparison

- Piqued my interests
  - In memory management, optimizing GPU throughput; now that I’m doing it, **I kinda get it**