GPU DOCK: GPU Acceleration for Molecular Docking

Saad Amin, John Irwin

UCSF Dept. of Pharmaceutical Chemistry, 1700 4th St, San Francisco CA 94158-2330 and Mission San Jose High School, 41717 Palm Ave, Fremont, CA 94539

Contact: saadamin245@gmail.com

Project Overview

- Molecular docking is a very computationally expensive process.
- Current versions of DOCK attempt to amortize docking times by docking many ligands in parallel.
- However, docking an individual ligand can be parallelized, making it a perfect candidate for GPU acceleration.
- We'll discuss how we implemented GPU acceleration in DOCK 3.8 and propose a future, more efficient docking algorithm

What Even is a GPU?

- GPUs are hardware devices separate from the CPU that can execute the same set of instructions on many different sets of data in parallel.
- For instance, suppose you had two arrays you wanted to add together and store the result in a third array.
- A CPU implementation would loop over each element and execute the exact same set of instructions on each element *but with different sets of data*.
- A GPU would launch multiple threads that would execute the add instruction in parallel in *lock-step execution*, meaning that all add instructions are executed at the same time.
  - Please note this is an oversimplification on how GPUs work.
  - See figure 1 for more details.
- Docking is often repeated sets of instructions executed on different sets of data, e.g. matching and scoring.
- Since GPUs are well suited to these kinds of tasks, docking is a good candidate for GPU acceleration.

How We GPU-ified DOCK

- We needed to port DOCK 3.8 over from C++ to make it easier to work with.
  - Also, CUDA Fortran was unable to compile DOCK 3.8.
- As we migrated over to C++, we took the opportunity to optimize slow sections of DOCK 3.8.
  - We'll refer to the C++ optimized version of DOCK 3.8 as C++ DOCK.
  - C++ DOCK doesn't have any program behavior changes but is much faster than DOCK 3.8, especially when it comes to matching
- We iteratively replaced parts of the Fortran code with a C++ version and verified the program along the way using OUTDOCK and the mol2 output files.
- We first ported scoring over to C++, which we then GPU-ified.
  - Our scoring algorithm is shown in figure 2. It scores all sets of a ligand for a particular match in parallel and sends the data back to the CPU.
  - We initially saw a speedup of 23% compared to the single core version of C++ DOCK.

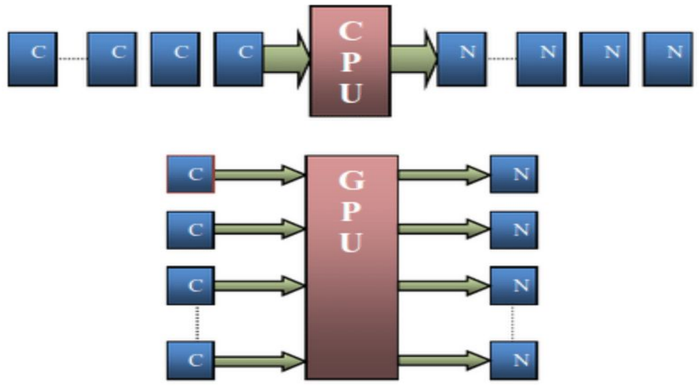

Figure 1

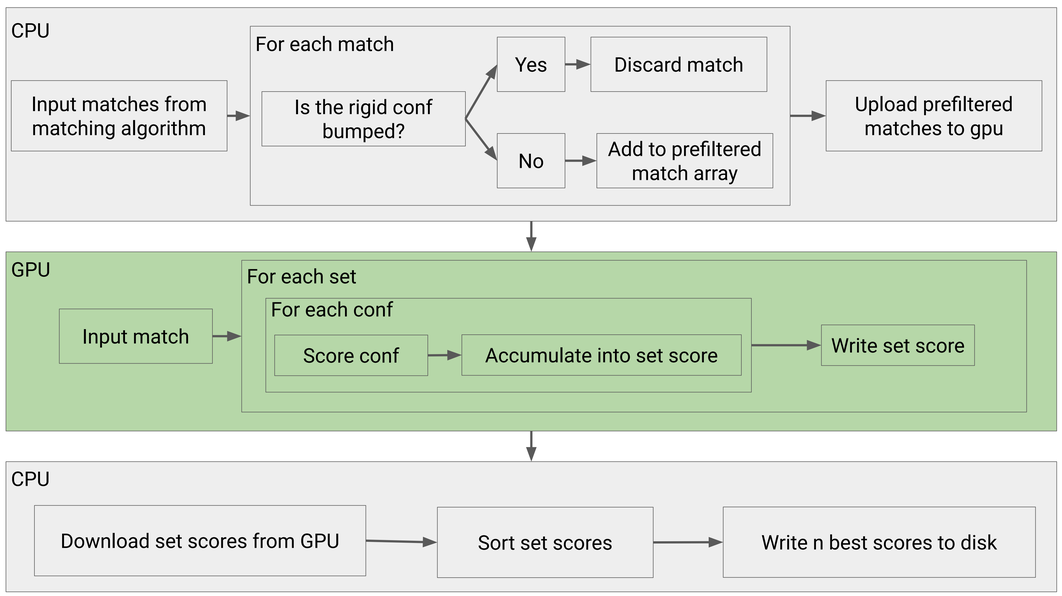

Figure 2

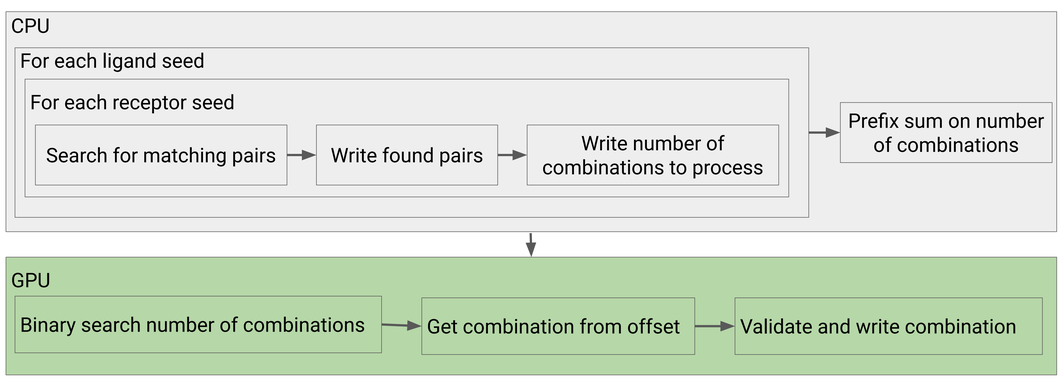

Figure 3

- After our modification, we moved towards optimizing matching, as that took 95% of docking time, especially with high match counts.
  - Validating match combinations is the most expensive part of matching, requiring immense bandwidth.
  - Our current approach uses one thread/combination and achieves a massive speed up from 500 seconds total docking time down to 300 seconds.
  - You can see the algorithm in figure 3.
- A lot of the 300 second docking time came from passing data from C++ code back to Fortran code, and then processing the data with slower Fortran code.
  - By fully integrating Mol2 output into C++, we got GPU DOCK down to 190 seconds
  - A large portion of time is still spent reading from files. Reading the H17P050-N-laa dataset took 44 seconds and reading can only be done serially, which becomes a problem for any DOCK implementation

How Does the Performance Compare?

- For our benchmarks, we ran the H17P050-N-laa dataset with various DOCK backends. Here are our results:

Name	CPU Threads Used	Docking Time (Seconds)
DOCK 3.8	1	2300
C++ DOCK	1	950
C++ DOCK	16	115
GPU DOCK	1	190

- Notably, C++ DOCK's optimization allows it to speed way ahead of DOCK 3.8, even surpassing GPU DOCK in the multithreaded test.
  - GPU DOCK is still early in the process of being ported to the GPU and there still is a lot of room for optimization.

Future Plans

- In the future, we plan to transition more towards a plugin-based design. Ideally, any DOCK user should be able to write their own subroutines for docking.
  - The C++ DOCK backend could become a built-in plugin, and GPU DOCK could become a separate plugin the user could select to use.
  - This way, people who don't have CUDA installed can still benefit from the optimizations of C++ DOCK.
- We aim to do end-to-end docking on the GPU.
  - We can split matching and scoring into parallel substages.
  - Each substage takes information from the previous stage and creates new information that is passed to the next