

NCHC Open Hackathon

Dream Chaser
Final

2024/12/04

Dream Chaser

Leader



Gao Quan-Ze
高銓澤
CGU 醫學



Member 1

Cheng-Yu Ma
馬誠佑
CGU 人工智慧



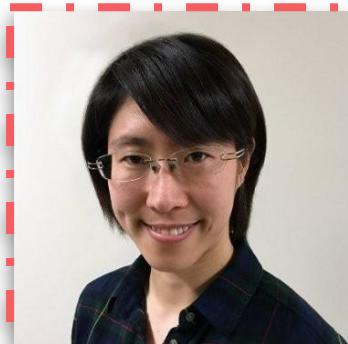
Member 2

Chi-Ching Lee
李季青
CGU 資訊工程



Mentor 1

Anthony Chang
張安政
NVIDIA DevTech



Mentor 2

Ying-Ja Chen
陳映嘉
NVIDIA Solution Architect

OmegaDocker

Introduction

- OmegaDocker is an oligomer-supported and gpu-accelerated docking program under LGPL.
- The first (current) version of OmegaDocker is based on AutoDock-GPU v1.5.4 developed by the Forli lab at Scripps Research.
- AutoDock-GPU v1.5.4 is the GPU version of AutoDock v4.2.6.

Algorithmic motif

Genetic algorithm (GA), Adadelta

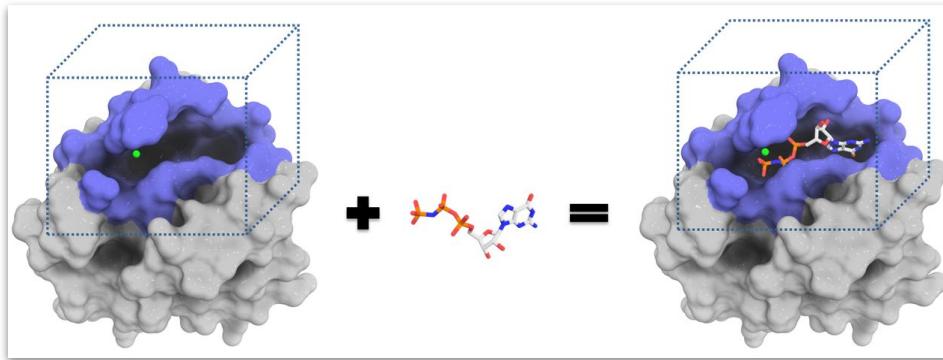
Libraries

CUDA, GMP

Programming Language

C/C++

Docking



Due to the computation complexity,
only the small compound is flexible.

DockThor

GMMSB

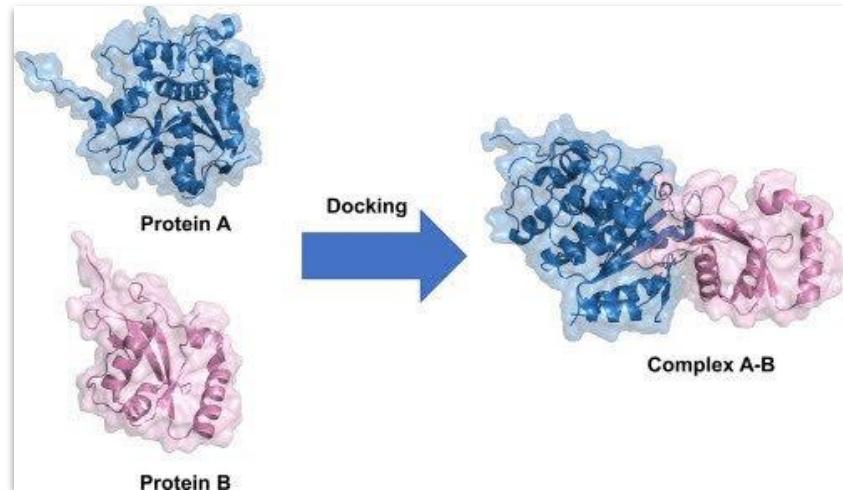
<https://dockthor.lncc.br/v2/>

But the demand for docking with
miRNAs or aptamers is increasing.

Computational Methods in Drug Discovery and
Development

Sep. 2024, Sadettin Yavuz Ugurlu

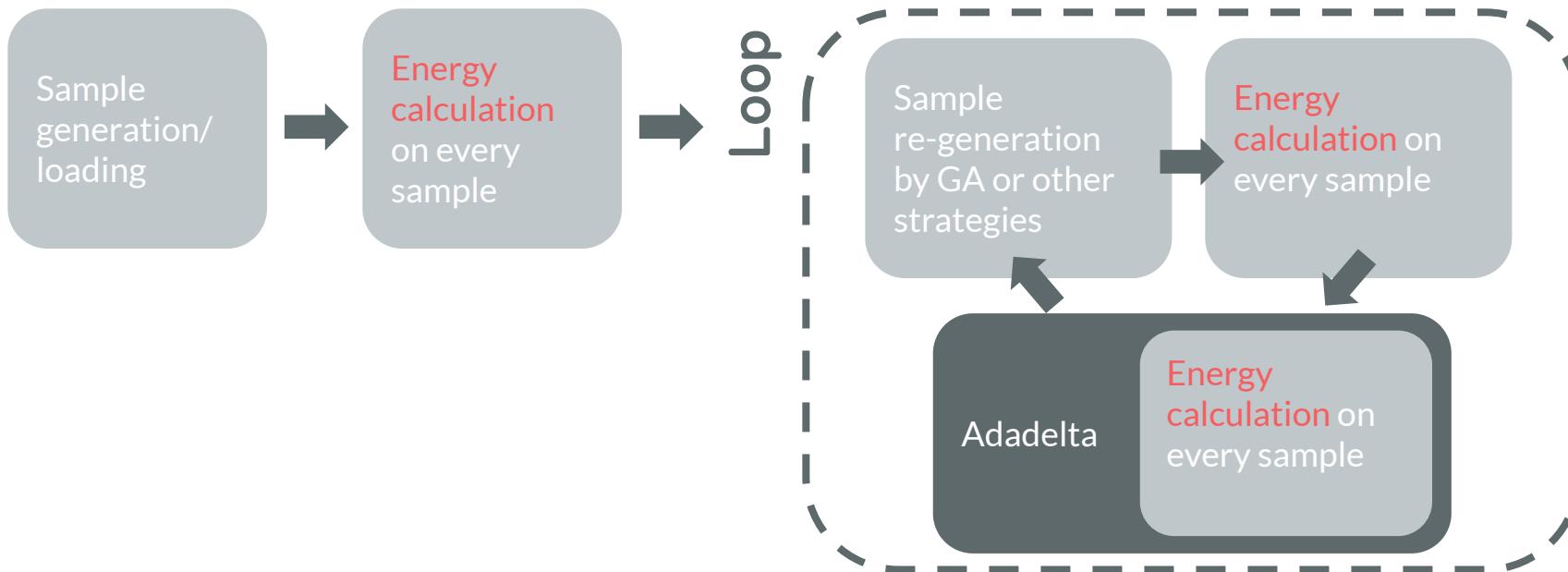
<https://chemrxiv.org/engage/chemrxiv/article-details/66f4016a12ff75c3a16b28c2>



OmegaDocker

The most time-consuming function

Energy calculation function



Initial Goal

Acceleration test of tensor core applying

Accelerating a Molecular Docking Application by Leveraging Modern Heterogeneous Computing Systems

GABIN SCHIEFFER

Degree Programme in Computer Science and Engineering
Date: April 16, 2023

Supervisor: Ivy B. Peng
Examiner: Stefano Markidis
School of Electrical Engineering and Computer Science
Swedish title: Accelererering av en Molekylär Dockningsapplikation genom att Utnyttja Moderna Heterogena Datorsystem

arXiv:2410.10447v1 [cs.DC] 14 Oct 2024

Accelerating Drug Discovery in AutoDock-GPU with Tensor Cores

Gabin Schieffer and Ivy Peng^(✉)

KTH Royal Institute of Technology, Stockholm, Sweden
`{gabins, ivybopeng}@kth.se`

Abstract. In drug discovery, molecular docking aims at characterizing the binding of a drug-like molecule to a macromolecule. AutoDock-GPU, a state-of-the-art docking software, estimates the geometrical conformation of a docked ligand-protein complex by minimizing a scoring function. Our profiling results indicate that the current reduction operation that is heavily used in the scoring function is sub-optimal. Thus, we developed a method to accelerate the sum reduction of four-element vectors using matrix operations on NVIDIA Tensor Cores. We integrated the new reduction operation into AutoDock-GPU and evaluated it on multiple chemical complexes on three GPUs. Our results show that our method for reduction operation is 4-7 times faster than the AutoDock-GPU baseline. We also evaluated the impact of our method on the overall simulation time in the real-world docking simulation and achieved a 27% improvement on the average docking time.

Keywords: Molecular docking · AutoDock · GPU · Tensor Core · Drug Discovery

1 Introduction

The pharmacological effect of a drug is generally induced by the binding of a drug molecule to a specific protein target. Thus, characterizing the ability of binding is crucial for drug discovery. Once a target for a disease is identified, tens of millions of chemical compounds, or *ligands*, will go through high-throughput screening. For such vast search space, virtual screening that leverages computational approaches is becoming increasingly important for accelerating the process and reducing the high cost required in experimental screenings [125]. In particular, structure-based virtual screening software uses molecular docking tools to test a molecule drug candidate for binding a protein target (receptor). In recent COVID-19 research, high-performance virtual screening software has been used in combating the pandemic [3].

A typical molecular docking job consists of evaluating a large number of ligands, each as an independent docking task. Further distributing individual docking tasks onto high-performance computing (HPC) systems, with multi-core CPU or GPUs, can significantly accelerate docking, e.g., AutoDock-GPU reports 350-fold speedup over single-threaded implementation [812]. AutoDock is widely

<https://arxiv.org/abs/2410.10447>

Initial Goal

Acceleration test of tensor core applying

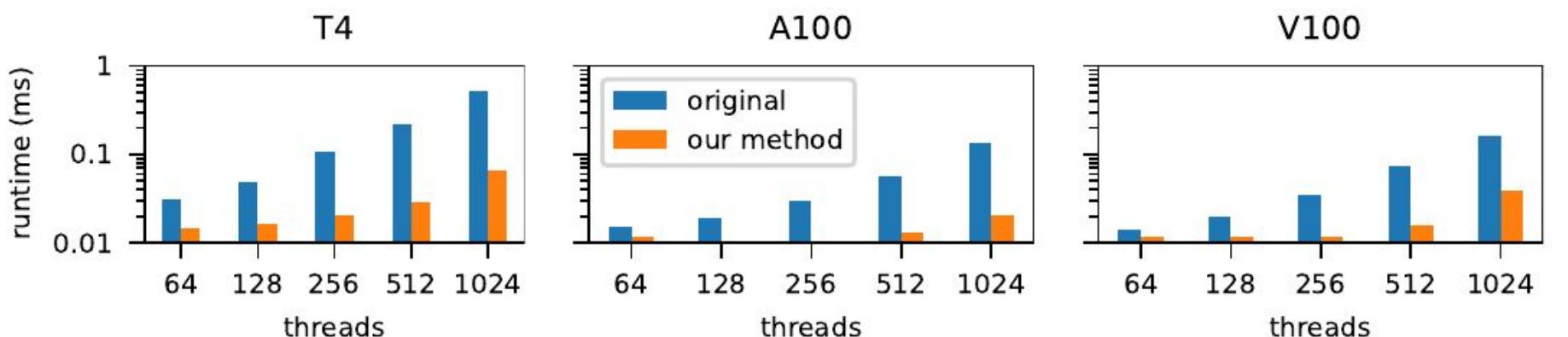
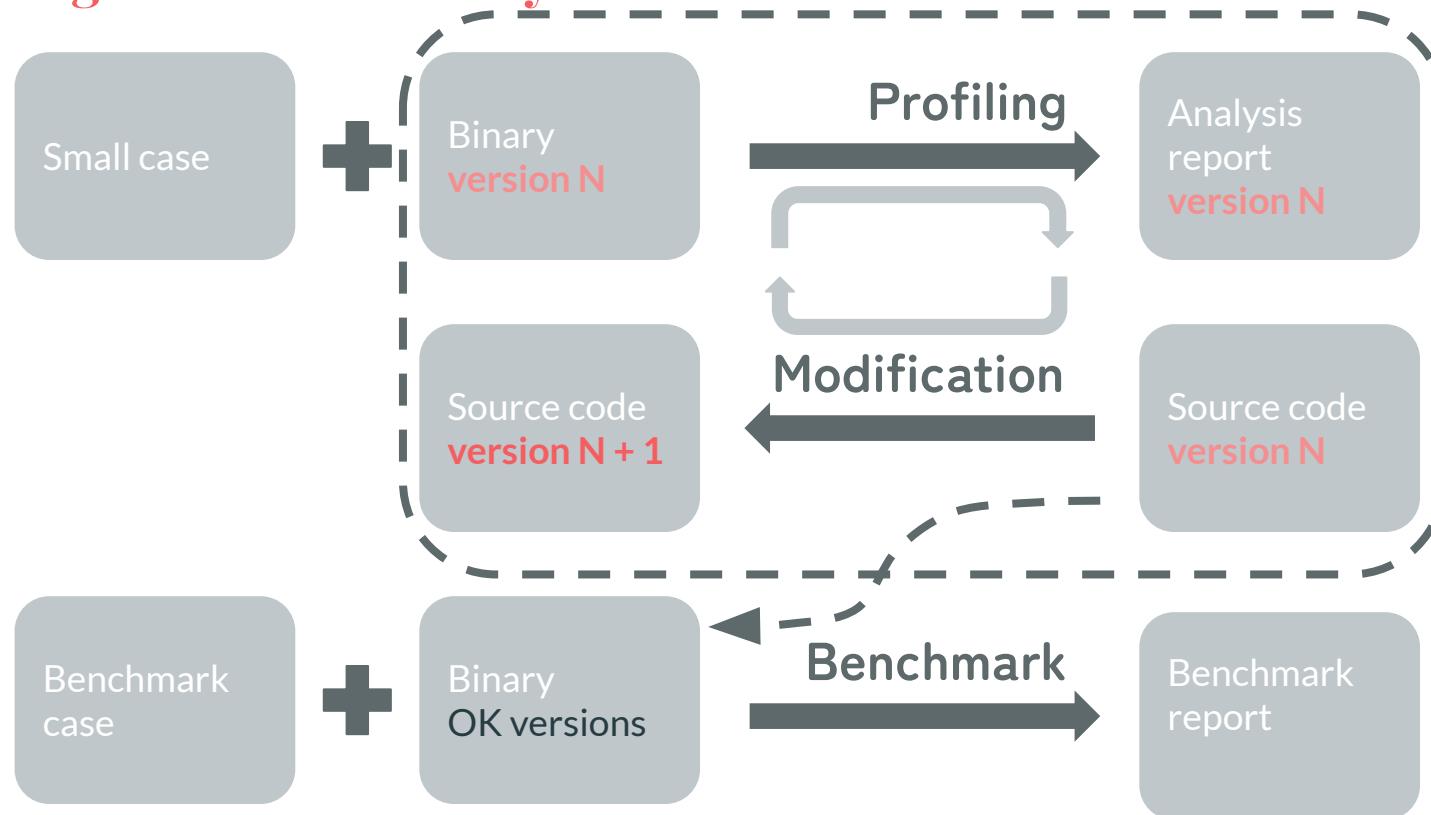


Fig. 5: Average runtime of the two versions of the test reduction kernel on three generations of NVIDIA GPUs: T4, A100, and V100.

Actual Strategy

Profiling – Modification Cycle



Overall Improvement

Overall Improvement

Size of blockDim

1.38

Algorithm modification

1.18

Conditional statements

1.02

printf() in kernel

1.01

Prefetch

N/A

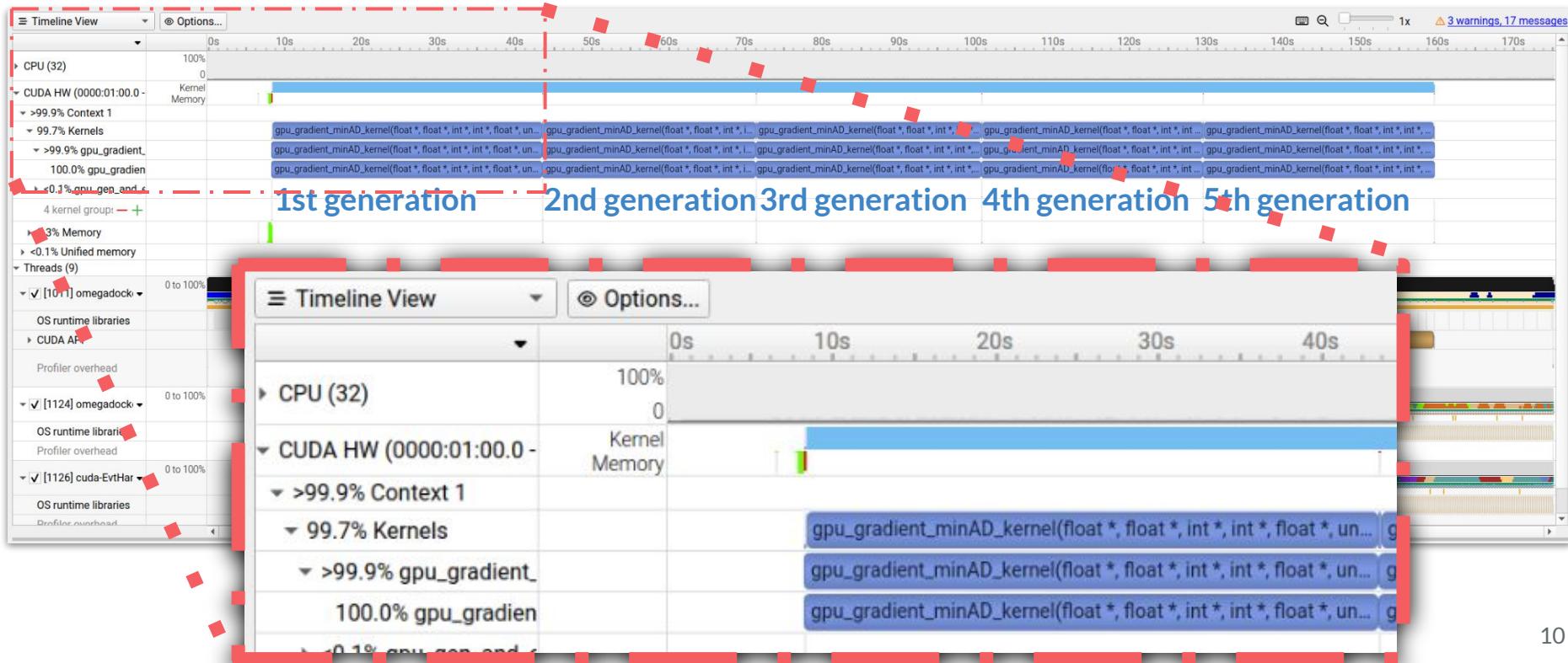
7.79

Remove unnecessary
atomic operations

4.68

Profiling of OmegaDocker

Nsight Systems 2024.5.1



Profiling of OmegaDocker

Nsight Compute 2024.5.1

Run with root permission

The screenshot shows the Nsight Compute 2024.5.1 interface with a report titled "base_NsightComputeReport2.ncu-report". The "Result" section displays performance metrics for a kernel named "728 - gpu_gradient_minAD_kernel". The metrics include:

Size	Time	Cycles	GPU	SM Frequency	Process	Attributes
(160, 1, 1)x(128, 1, 1)	15.83 s	22,323,384,802	1 - NVIDIA RTX A6000	1.41 Ghz	[4084] omegadocker_gpu_128wi	...

The "Summary" tab is selected, showing a table of results:

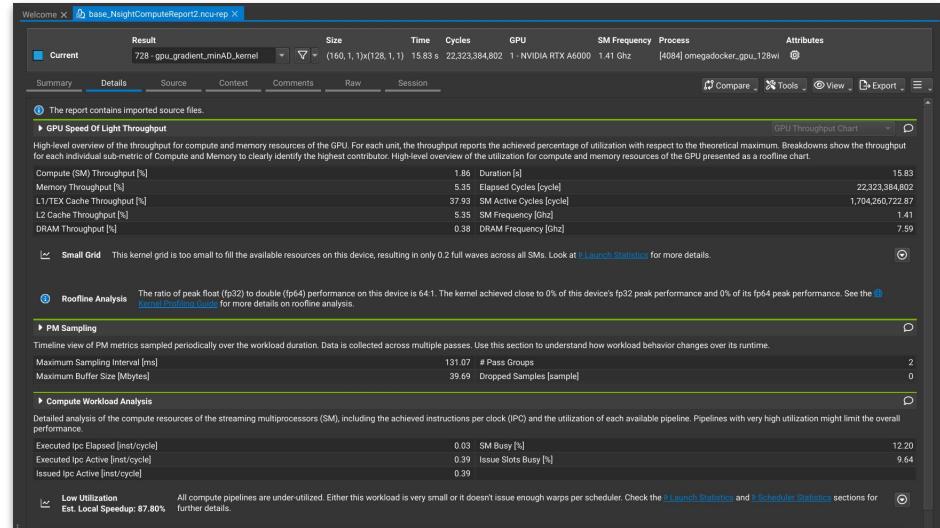
ID	Estimated Speedup	Function Name	Demangled Name	Duration (2.92062e+10)	Runtime Improvement (2.43521e+10)	Compute Throughput	Memory Throughput	# Registers	Grid Size	Block Size
0	83.15	gpu_gradient_min...	gpu_gradient_min...	15.83	13.16	1.86	5.35	64	160, 1, ...	128, 1, ...
1	83.66	gpu_gradient_minA...	gpu_gradient_minA...	13.37	11.19	2.38	6.87	64	160, 1, ...	128, 1, ...

A tooltip message at the bottom left of the table area states: "This table shows all results in the report. Use the column headers to sort the results in this report. Double-click a result to see detailed metrics. Double-click on demangled names to rename it."

Profiling of OmegaDocker

Nsight Compute 2024.5.1

Run with root permission



Potential Performance Bottlenecks

Size of blockDim

- Original: 128 (Original source code up limit: 256)
- Tested: 256, 512, 1024 (Source code modified)

► Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide memory latency. Achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	66.67
Theoretical Active Warps per SM [warp]	32
Achieved Occupancy [%]	11.24
Achieved Active Warps Per SM [warp]	5.39



active war
ability to h
66.67
32
65.52
31.45

Performance ratio

- 1.38x

Potential Performance Bottlenecks

printf() in kernel

- Original: For an important function in other docking stage
- Modified: Create a new kernel for it.

```
if(threadIdx.x == 0 && blockIdx.x == 0)
{
    printf("COOR1 %d\n", iteration_cnt);
    for(uint i = 0; i < cData.dockpars.num_of_atoms; i++)
    {
        // printf("ATOM1 %12u %12.6f %12.6f %12.6f\n", i, calc_coords[i].x, calc_coords[i].y, calc_
        printf("ATOM1 %12u %12.6f %12.6f %12.6f\n", i, cData.pKerconst_conform_ref_coords_const[3*i]
    }
    printf("DONE1 %d\n", iteration_cnt);
}
__syncthreads();
```

Performance ratio

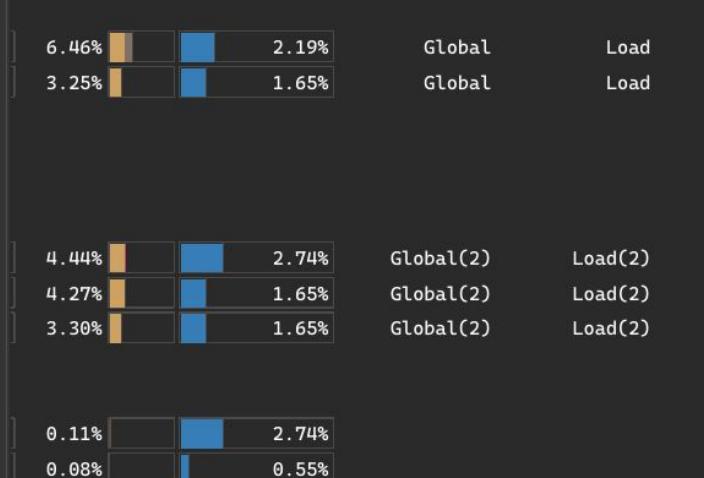
- 1.01x

Potential Performance Bottlenecks

Prefetch

- Original: Many global variables IO

```
// sans =>  
uint32_t atom1_id = cData.pKerconst_intracontrib_intraE_contributors_const[2*contributor_counte  
uint32_t atom2_id = cData.pKerconst_intracontrib_intraE_contributors_const[2*contributor_counte  
// <== sans  
  
// Calculating vector components of vector going  
// from first atom's to second atom's coordinates  
float subx = calc_coords[atom1_id].x - calc_coords[atom2_id].x;  
float suby = calc_coords[atom1_id].y - calc_coords[atom2_id].y;  
float subz = calc_coords[atom1_id].z - calc_coords[atom2_id].z;  
  
// Calculating atomic_distance  
float dist = sqrt(subx*subx + suby*suby + subz*subz);  
float atomic_distance = dist * cData.dockpars.grid_spacing;
```



Potential Performance Bottlenecks

Prefetch

The screenshot shows a blog post on the NVIDIA Developer website. The title is "Boosting Application Performance with GPU Memory Prefetching". It was posted on March 23, 2022, by Rob Van der Wijngaart and Fred Oh. The post has 10 likes and 7 discussions. The content includes a large image of a 3D surface plot with a grid, colored in shades of green and blue, representing data being processed.

The screenshot continues the blog post. It discusses synchronization within loops and asynchronous loads into shared memory. A code snippet is provided:

```
#include <cuda_pipeline_primitives.h>
__shared__ double v[PDIST* BLOCKDIM];
for (int k=PDIST; k<imax) { // fill the prefetch buffer asynchronously
    __pipeline_memcpy_async(&vsmem(k), &aarr[threadIdx.x+k* BLOCKDIM], 8);
    __pipeline_commit();
}
for (int i=threadIdx.x, ctr=0; i<imax; i+= BLOCKDIM, ctr++) {
    __pipeline_wait_prior(PDIST-1); // wait on needed prefetch value
    ctr_mod = ctr%PDIST;
    locvar = vsmem(ctr_mod);
    if (i<imax-PDIST* BLOCKDIM) { // prefetch one new value
        __pipeline_memcpy_async(&vsmem(ctr_mod), &aarr[i+PDIST* BLOCKDIM], 8);
        __pipeline_commit();
    }
    // more instructions using locvar, for example, transcendentals
}
```

Text below the code explains: "As each `__pipeline_wait_prior` instruction must be matched by a `__pipeline_commit` instruction, we put the latter inside the loop that prefills the prefetch buffer, before entering the main computational loop, to keep bookkeeping of matching instruction pairs simple."

Performance results

Figure 1 shows, for various prefetch distances, the performance improvement of a kernel taken from a financial

Performance ratio

- N/A

Potential Performance Bottlenecks

Conditional statements

```
bool notDoubleH = !(  
    atom1_type_vdw_hb == 0 ||  
    atom1_type_vdw_hb == 1 ||  
    atom1_type_vdw_hb == 2 ||  
    atom1_type_vdw_hb == ATYPE_HG_IDX  
)  
&&  
(  
    atom2_type_vdw_hb == 0 ||  
    atom2_type_vdw_hb == 1 ||  
    atom2_type_vdw_hb == 2 ||  
    atom2_type_vdw_hb == ATYPE_HG_IDX  
)
```

9.46% 7.55% Global Load



```
bool notDoubleH = (atom1_type_vdw_hb > 2 && atom1_type_vdw_hb != ATYPE_HG_IDX)  
||  
(atom2_type_vdw_hb > 2 && atom2_type_vdw_hb != ATYPE_HG_IDX);
```

48 2.94% 2.54%

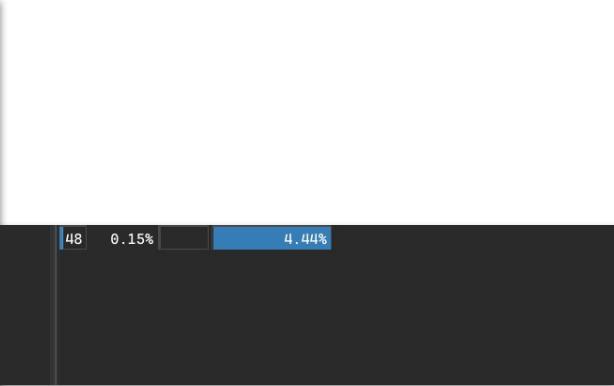
Potential Performance Bottlenecks

Conditional statements

```
if  
{  
    if  
        (atom1_type_vdw_hb == ATYPE(CG_IDX) ||  
         (atom1_type_vdw_hb == ATYPE(HG_IDX) ||  
         (atom1_type_vdw_hb == ATYPE(NG_IDX) ||  
         (atom1_type_vdw_hb == ATYPE(OG_IDX)  
     )  
     &&  
    (atom2_type_vdw_hb == ATYPE(G0_IDX)  
 )  
||  
    (atom1_type_vdw_hb == ATYPE_G0_IDX  
 )  
    &&  
    (atom2_type_vdw_hb == ATYPE(CG_IDX) ||  
     (atom2_type_vdw_hb == ATYPE(HG_IDX) ||  
     (atom2_type_vdw_hb == ATYPE(NG_IDX) ||  
     (atom2_type_vdw_hb == ATYPE(OG_IDX)  
 )  
 )
```



```
if  
{  
    if  
        (atom1_type_vdw_hb > ATYPE_G0_IDX && atom2_type_vdw_hb == ATYPE_G0_IDX)  
        ||  
        (atom1_type_vdw_hb == ATYPE_G0_IDX && atom2_type_vdw_hb > ATYPE_G0_IDX)
```



Performance ratio

- 1.02x

Potential Performance Bottlenecks

Remove unnecessary atomic operations

```
ATOMICSUBI32(&gradient[atom1_id].x, priv_intra_gradient_x);  
ATOMICSUBI32(&gradient[atom1_id].y, priv_intra_gradient_y);  
ATOMICSUBI32(&gradient[atom1_id].z, priv_intra_gradient_z);
```

Time

8.67 s

```
ATOMICADDI32(&gradient[atom2_id].x, priv_intra_gradient_x);  
ATOMICADDI32(&gradient[atom2_id].y, priv_intra_gradient_y);  
ATOMICADDI32(&gradient[atom2_id].z, priv_intra_gradient_z);
```

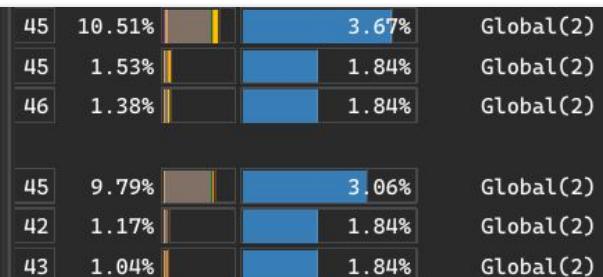
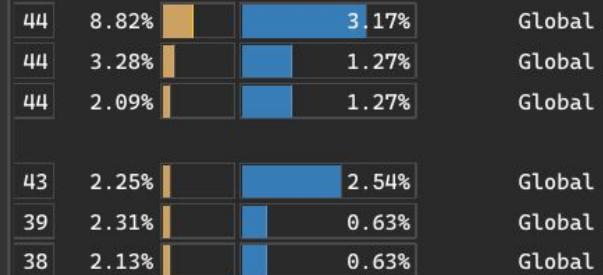
```
gradient[atom1_id].x == priv_intra_gradient_x;  
gradient[atom1_id].y == priv_intra_gradient_y;  
gradient[atom1_id].z == priv_intra_gradient_z;
```



Time

1.21 s

```
gradient[atom2_id].x += priv_intra_gradient_x;  
gradient[atom2_id].y += priv_intra_gradient_y;  
gradient[atom2_id].z += priv_intra_gradient_z;
```



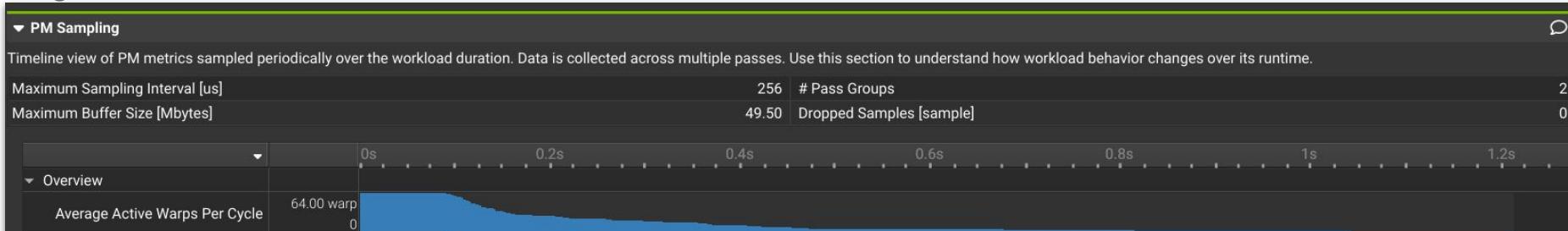
Performance ratio

- 4.68x

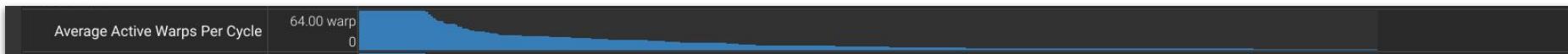
Potential Performance Bottlenecks

Algorithm modification

Original



Modified Test 1



Modified Test 2



Performance ratio

- 1.00x ~ 1.18x

Overall Improvement

Overall Improvement

Size of blockDim

1.38

Algorithm modification

1.18

Conditional statements

1.02

printf() in kernel

1.01

Prefetch

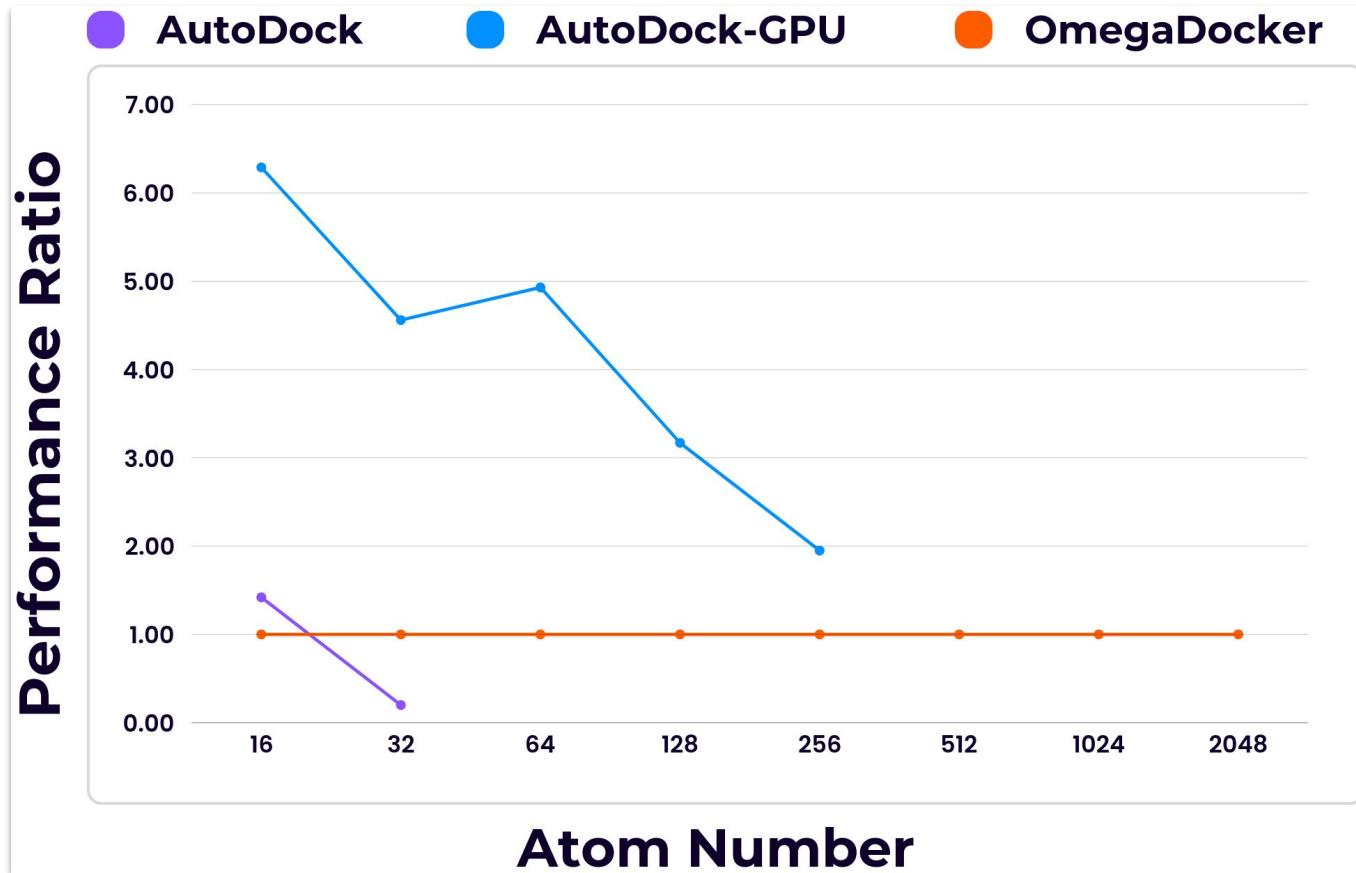
N/A

7.79

Remove unnecessary
atomic operations

4.68

Availability and Performance



Not Everything, Not yet.

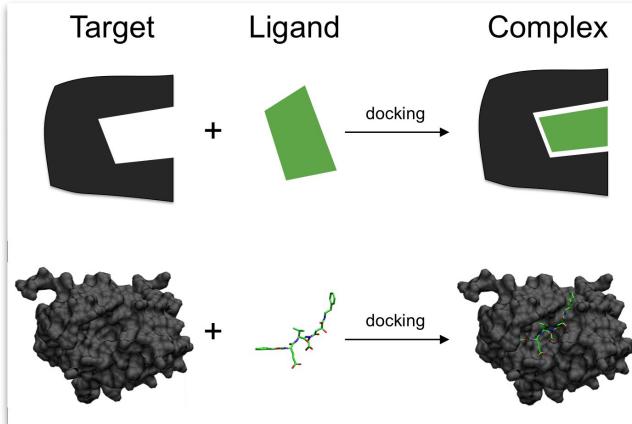
Further improvement

- Prefetch
- Other tricks for accelerating the IO of global memory
- Tensor Cores utilization

Learning never ends

- Nsight Compute
- Profiling tricks
- Advanced CUDA programming

分子對接



Dream Chaser 團隊成員長庚大學醫學系高鈺澤博士、人工智能學系馬誠佑老師、與資工系李季青老師，解除對接軟體 AutoDock 的原子數量限制，並保持穩定的效能。

— NVIDIA Mentors: Anthony Chang, Ying-Ja Chen

分子對接是加速藥物開發不可或缺的重要工具，有效的電腦計算結果，能節省大量後期生物實驗與臨床試驗的費用與時間。然而就算在電腦的輔助之下，藥物開發的平均時長約十年，且 2024 年六月的研究指出，2018 年的平均藥物開發成本約為 8.793 億美元，並且逐年上升。近年除了小分子藥物以外，較小的生物分子如 miRNA 與 aptamer 對於調節細胞生長與功能的研究需求也是與日俱增，由於其可能成為潛在的抗癌新星，讓許多學者投入相關研究。然而目前支援 GPU 的分子對接軟體有藥物原子數量上限的限制，使得相關研究難以順利進行。Dream Chaser 團隊與 NVIDIA 成員合作，在解除原子數量限制的情況下，透過 CUDA 加速獲得穩定的效能。

