석 사 학 위 논 문
Master's Thesis

실수연산의 계산 가능성, 효율, 실제사례

Real Computation:
from Computability via Efficiency to Practice

2021

황 지 만 (黃 智 萬 Hwang, Jiman)

한 국 과 학 기 술 원

Korea Advanced Institute of Science and Technology

석 사 학 위 논 문

실수연산의 계산 가능성, 효율, 실제사례

2021

황 지 만

한 국 과 학 기 술 원

전산학부

실수연산의 계산 가능성, 효율, 실제사례

황 지 만

위 논문은 한국과학기술원 석사학위논문으로
학위논문 심사위원회의 심사를 통과하였음

2021년 5월 31일

심사위원장	Ziegler Martin	(인)
심 사 위 원	Svetlana Selivanova	(인)
심 사 위 원	한 종 인	(인)

Real Computation: from Computability via Efficiency to Practice

Jiman Hwang

Advisor: Martin Ziegler

A dissertation submitted to the faculty of
Korea Advanced Institute of Science and Technology in
partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Daejeon, Korea
May 31, 2021

Approved by

Martin Ziegler
Professor of School of Computing

The study was conducted in accordance with Code of Research Ethics[1].

[1] Declaration of Ethical Conduct in Research: I, as a graduate student of Korea Advanced Institute of Science and Technology, hereby declare that I have not committed any act that may damage the credibility of my research. This includes, but is not limited to, falsification, thesis written by someone else, distortion of research findings, and plagiarism. I confirm that my thesis contains honest conclusions based on my own careful research under the guidance of my advisor.

MCS 황지만. 실수연산의 계산 가능성, 효율, 실제사례. 전산학부 . 2021년. 36+iv 쪽. 지도교수: 지글러마틴. (영문 논문)
Jiman Hwang. Real Computation:
from Computability via Efficiency to Practice. School of Computing . 2021. 36+iv pages. Advisor: Martin Ziegler. (Text in English)

초 록

이 논문에서는 실수연산에 관한 세 가지 주제를 다룬다. 첫째, 유클리드 공간상의 콤팩트 집합을 정확히 표현하기 위한 추상데이터형을 제안한다. 이 추상데이터형으로 한 점이 콤팩트 셋에 포함되는지 여부를 알 수 있다. 또한 콤팩트 집합이 평면상에 정의되는 경우, 픽셀 단위로 정확한 그림을 그릴 수 있다. 둘째, 100 비트급 크기의 부동소숫점 곱셈을 부동소숫점장치로 가속하는 방법을 알아본다. 널리 쓰이는 `MPFR` 소프트웨어 라이브러리보다 부동소숫점장치로 행렬 곱셈과 다항식 곱셈을 하면 최소 네 배 빠른 것으로 실험결과 확인되었다. 셋째, 공기청정기에서 정화된 공기가 사람들에게 효과적으로 전달되도록 하는 출구 모양과 설치 높이를 시뮬레이션을 통해 알아낸다. 그 결과, 기존 프로토타입에 대비해 PM2.5 밀도가 $50\mu g/m^3$이하인 영역이 공기청정기 주변으로 형성됨을 확인하였다.

핵 심 낱 말 실수 연산, 콤팩트 집합, 곱셈, 모의실험, 미세먼지

Abstract

We address three subjects regarding real computation. Firstly, we introduce an abstract data type (ADT) for representing compact subsets in Euclidean space in Exact Real Computation. Given a point, it provides membership check up to the desired precision. Furthermore, the visualization is supported for 2D compact subsets. Secondly, we explain a method to accelerate multiplication of 100-bit precision floating-points via Floating-Point Unit (FPU). The experiment shows that it is at least 2-4 times faster than `MPFR` to multiply matrices and polynomials, respectively. Lastly, we run a simulation to design an outlet shape and installation height for an air purifier so that the clean air flows to the average human effectively. Our simulation has improved the efficiency of the original prototype so that it successfully forms an area of PM2.5 density less than $50\mu g/m^3$ around itself.

Keywords real computation, compact set, multiplication, simulation, fine dust

Contents

List of Tables

List of Figures

Chapter 1. Introduction

Real-valued models arise in many applications, including Physics, Engineering, Artificial Intelligence, etc. While the real numbers form a continuous space in Mathematics, a computer works on discrete spaces. This discrepancy makes it hard to deal with real numbers on a computer, leaving the design choice between accuracy and computation speed depending on the purpose of the computation. High accuracy is very difficult to achieve while it is every important e.g. for safety-critical applications such as Particle Physics. At the same time, it is also vital to keep the computations feasible, and carefully balancing between high accuracy and fast computation.

Chapter 2 explains how to represent compact subsets in Euclidean spaces with any prescribed precision. This is done based on an Exact Real Computation package called `iRRAM` [2]. Written in `C++`, it performs real computations within any given rounding error. Chapter 3 discusses how to increase the efficiency of multiplication performance in `iRRAM` when balancing between the accuracy and the speed by utilizing multiple floating-points. Lastly, Chapter 4 explores an application of real computation where a simulation is used to improve a purifier, exploiting fast computations.

Chapter 2. Compact Euclidean Subsets as Computable Data Type

It is desired to manipulate some geometric objects in Euclidean space such as a triangle, a sphere, or even a function on $\mathbb{R}^n$ via software. Some software (e.g. *Mathematica* and *Geogebra*) support it for the education and research purpose. However, these suffer from error-prone computations under the hood because the data structure for containing real numbers has a finite precision. As a result, one may get a wrong answer, for example, from a membership test of a point near the boundary of a shape in Euclidean space.

In this chapter we present an abstract data type (ADT) which takes the characteristic function for a bounded and closed subset $\mathcal{C}$ of Euclidean space—thereby compact— in Exact Real Computation(ERC). It provides two functionalities: 1) testing if a given point is in $\mathcal{C}$ and 2) drawing a figure of $\mathcal{C}$ if $\mathcal{C} \subset \mathbb{R}^2$. Moreover, handling a homotopy of the form $f : [0,1]^m \to \mathbb{R}^n$ as a compact set is explained because finding the characteristic function is often cumbersome.

Section 2.1 explains the necessity and brief internals of ERC. Section 2.2 introduces ADT for compact subsets, which takes a characteristic function as an input. Section 2.3 describes another ADT for compact subsets, which takes a homotopy of the form $f : [0,1]^m \to \mathbb{R}^n$ as an input.

All contents in this Chapter are implemented in `iRRAMx`, an `iRRAM` [2] extension. See the full code at https://github.com/realcomputation/iRRAMx.

2.1 Background

A mathematical method of representing a bounded and closed set $\mathcal{C} \subset \mathbb{R}^n$, or equivalently compact subset, is to define a characteristic function $\chi : \mathbb{R}^n \to \{0,1\}$ where $\chi(x) = 1$ if $x \in \mathcal{C}$ and $\chi(x) = 0$ otherwise. For example, the closed ball of radius π centered at the origin is defined as $\chi(x) = 1$ if $|x| \leq \pi$ and $\chi(x) = 0$ otherwise.

When it comes to computer realm where only discrete memory cells are provided, it is not trivial to embed a function defined on $\mathbb{R}^n$. For instance, when $|y| = \pi$, the characteristic function χ above produces $\chi(y) = 1$. Suppose we implemented a function χ' to approximate χ such that $\chi'(x) = 1$ if $|x| \leq 3.14159$ and $\chi'(x) = 0$ otherwise, using IEEE [11] 64-bit floating-point. It follows that $\chi'(z) = 0$ for $|z| = 3.141591$, which exhibits the failure of the approximation. This is due to the lack of precision when manipulating real numbers. The approximated boundary 3.14159 isn't precise enough to express $\pi = 3.14159265359\cdots$, hence it raises a round-off error.

The problem is that a round-off error may take place over the entire computation and influence the real world seriously [7]. As the necessity of more precise computation is observed, researchers have developed software for high precision such as `MPFR` [1], which provides an arbitrary precision. Instead of having a fixed precision, the programmer adjusts it so that the final result gets more accurate. Some applications are inspecting mathematical constants [9], or mathematical Physics [8].

However, extending the available precision doesn't guarantee that the final result is solid. Logistic map [17] is a good example that exhibits a chaotic aspect and fails to acquire the correct value. Moreover, numerical analysts take advantage of partial pivoting for Gaussian elimination [10] to alleviate the round-off error even if each element of a matrix has an arbitrary precision.

ERC came up with the concept of an arbitrary round-off error. A programmer is able to set the upper bound of the round-off error without a problem-specific workaround such as Gaussian elimination. A common way of achieving it is the interval arithmetic. Instead of working with a concrete value $a \in \mathbb{R}$, we maintain an interval $[l, r] \ni a$ where l and r are expressible with a finite precision, often dyadic. It follows that all arithmetic operations are defined on the intervals. For example, we can define the addition of $[l_1, r_1]$ and $[l_2, r_2]$ as $[l_1 + l_2, r_1 + r_2]$ which still include the true value. The error is the half of the length, which is likely to increase over a sequence of computations.

The computing procedure is done iteratively. Initially a fixed precision is used for the boundary of each interval. If the error after the entire calculations is greater than the desired one, then we start it over with a higher precision. This iteration is repeated until the final error is less than the target.

However, this approach needs a caution because comparing two real number is not decidable [5, Theorem 4.1.16]. Assume $a > b$ is tested where a and b are numbers represented by intervals. If a and b are distinct, then the computation will finish within finite steps. However, if they're the same, then the computation never stop.

One workaround is to use two conditions in parallel. For a positive dyadic $\epsilon \ll 1$, we test $a > b$ and $a < b + \epsilon$ in parallel. Regardless of the equality of a and b, either condition must hold. Also, we'll get accurate result as $\epsilon \to 0$. Although we cannot say $a = b$ definitely in practice, ϵ is adjustable at the cost of more computations so that we can confirm that the test result is correct within a certain error bound.

There are software for ERC such as `iRRAM` [2] and `core2` [4] which utilize the interval arithmetic. In addition, `reallib` [3] is another one that has implemented ERC in a hybrid way by adding a symbolic approach on top of interval arithmetic. Having `C++` as the base code, we used `iRRAM` to express real numbers in this paper.

2.2 `Compact`

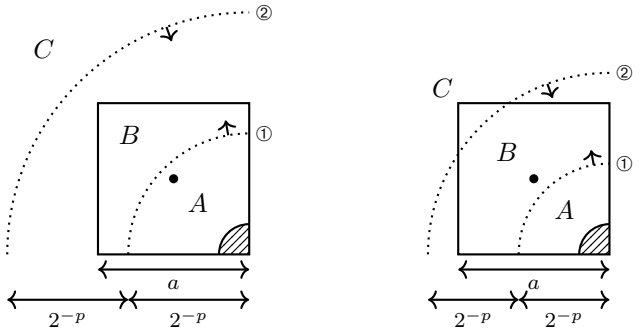
As a base object, `Compact` is an ADT for a compact set in Euclidean space, which takes a characteristic function as an initialization parameter. Since comparing two real numbers is not decidable, a naive characteristic function $\chi : \mathbb{R}^n \to \{0, 1\}$ may fail to compute $\chi(x)$ for x on the boundary of the compact set $\mathcal{C}$. Therefore, `Compact` accepts a multivalued characteristic function of the following properties [6, Definition 2.2.7].

$$\chi : \mathbb{R}^n \times \mathbb{N} \rightrightarrows \{0, 1\}$$
$$\chi(x, p) = \begin{cases} 1 & \text{if } B(x, 2^{-p}) \cap \mathcal{C} \neq \emptyset \\ 0 & \text{if } B\left(x, 2^{-p+1}\right) \cap \mathcal{C} = \emptyset \\ 0 \text{ or } 1 & \text{otherwise} \end{cases} \tag{2.1}$$

where $B(x, r)$ is the open ball of radius r centered at x. For example, the unit sphere at the origin is represented by

$$\chi : \mathbb{R}^3 \times \mathbb{N} \rightrightarrows \{0, 1\}$$
$$\chi(x, p) = \begin{cases} 1 & \text{if } |x| < 1 + 2^{-p} \\ 0 & \text{if } |x| > 1 + 2^{-p+1} \\ 0 \text{ or } 1 & \text{otherwise} \end{cases}$$

or equivalently as code



(a) a proper precision (b) an improper precision

Figure 2.1: Deciding the precision of drawing an image. □ and ▨ illustrate a pixel and a part of the compact set, respectively. The pixel center ● is tested on which side it is, based on ① and ② in parallel. The pixel is marked as black if it's on area A, white if it's on area C, and either black or white on area B. Both (a) and (b) must be colored by black because the pixel contains a part of the compact set. (a) shows an adequate setting of the precision p in which ● is on area A. (b), however, is not configured properly in which ● is on area B, possibly failing to color the pixel.

UNITSPHERE(x, p)
1 $d = \sqrt{x_1^2 + x_2^2 + x_3^2}$
2 **return** CHOOSE$(d > 2^{-p}, d < 2^{-p+1})$

where CHOOSE$(expr_1, expr_2)$ computes $expr_1$ and $expr_2$ in parallel and returns j when $expr_j$ turns out to be true first. In the implementation, we use two condition in parallel as above example.

The second parameter $p \in \mathbb{N}$ in (2.1) works as the precision controller. We acquire a more accurate result as $p \to \infty$. It is user's responsibility to make a characteristic function that agrees with (2.1).

The membership check is trivially supported by directly computing χ.

2.2.1 Drawing 2D images

The other functionality is drawing the computer image of the compact set on real plane. Every pixel that overlaps the compact set is colored by black, while the others are either white or black. To this end, we'll check if each pixel center is in the compact set or not with an appropriate precision.

If the precision is too low, we'll only get a trivial image such as all-black one that definitely covers the compact set. If the precision is too high, some pixel may not cover the compact set as shown in Fig 2.1.b.

Given the pixel size $a \in \mathbb{R}$, we must find p such that $2^{-p} > a\sqrt{2}/2$ so that the pixel center lies on area A, shown in Fig 2.1. Thus, we pick up $p := \lfloor 1/2 - \log_2 a \rfloor$ to draw the image as precise as possible.

Equipped with this decision strategy, DRAWCOM takes the characteristic function χ, the image width(the number of pixels), and the region $[x_1, x_2] \times [y_1, y_2]$ to draw and outputs the image as a two-dimensional binary array. Each cell having 1 indicates the corresponding pixel is colored.

```
DRAWCOM(χ, imgWidth, x1, x2, y1, y2)
 1  pixelSize = (x2 − x1)/imgWidth
 2  imgHeight = ⌈(y2 − y1) /pixelSize⌉
 3  p = ⌊1/2 − log2 pixelSize⌋
 4  image = new array[1..imgWidth][1..imgHeight]
 5  for i = 1 to imgWidth
 6      for j = 1 to imgHeight
 7          u = x1 + pixelSize × i − pixelSize/2
 8          v = y1 + pixelSize × j − pixelSize/2
 9          image[i][j] = χ((u, v), p)
10  return image
```

Line 1 and 2 computes the side length of a pixel and the image height respectively. DRAWCOM doesn't provide scaling along a single axis, hence the image height is determined by the image width. Line 3 computes the precision in accordance with the aforementioned explanation. Line 4 prepares the two-dimensional array to contain the image. For each pixel(cell), we compute the coordinate of its center, (u, v), on line 7-8 and test the membership on line 9.

See Fig 2.7 for an example.

2.3 `Homotopy`

`Compact` demands the characteristic function that satisfies (2.1) from the programmer, which is often unhandy. To give you an example, suppose we want to manipulate the compact set $f([0,1])$ such that $f : [0,1] \to \mathbb{R}^2$ and $f(t) = (t,t)$ via `Compact`. An intuitive approach to design the characteristic function is to compute the distance from an arbitrary point $(u, v) \in \mathbb{R}^2$ to f. First, we partition $\mathbb{R}^2$ into three areas by $y = x$ and $y = x + 2$. Then, we compute the distance between (u, v) and one of $(0, 0)$, $(1, 1)$ and f—by the perpendicular distance— depending on the area that contains (u, v). Splitting cases is already a hassle even for this simple example, not to mention more complicated ones such as $f(t) = (\sin t, \cos t)$.

To remove this inconvenience, we introduce the class `Homotopy`, which takes a computable homotopy of the form $f : [0,1]^m \to \mathbb{R}^n$ as an initialization parameter and represents the compact set $f([0,1]^m)$. Like `Compact`, checking the membership and drawing 2D image are available.

2.3.1 Membership check

Although we'll describe the membership checking function as code, not as a mathematical formula, both of them do exactly the same task. For this reason, let χ denote the membership check function. Initialized with a computable $f : [0,1]^m \to \mathbb{R}^n$, χ follows the property (2.1) with $\mathcal{C} = f([0,1]^m)$. However, the algorithm works differently as the characteristic function is not given.

Let us show the entire process to compute χ as code and then elaborate each line over a few subsections. The main strategy is the divide-and-conquer.

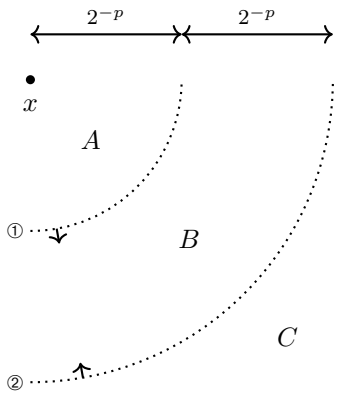


Figure 2.2: Membership check of x for the compact set $\mathcal{C} = f(H_R)$. The two condtions of χ are drawn as ① and ②, which partition the Euclidian space into three areas. If there exists $z \in \mathcal{C}$ on area A, $\chi(x) = 1$. If all $z \in \mathcal{C}$ are on area C, $\chi(x) = 0$. Otherwise, $\chi(x) = 0$ or 1, randomly. The two conditions are tested in parallel. If ② turns out to be true first, then there exists $z \in \mathcal{C}$ on area A or B, thereby it's safe to state $\chi(x) = 1$. If ① turns out to be true first, then there exists $z \in \mathcal{C}$ on area B or C, thereby it's safe to state $\chi(x) = 0$.

MEMBER(f, x, p, R)

```
1  Get a hyperrectangle H_{R'} that includes f(H_R).
2  r = |(center of H_{R'}) − (corner of H_{R'})|
3  d = |(center of H_{R'}) − x|
4  if CHOOSE(d − r > 2^{−p}, d − r < 2^{−p} + 2^{−p−2}) == 0
5      return 0
6  elseif CHOOSE(d + r > 2^{−p+1} − 2^{−p−2}, d + r < 2^{−p+1}) == 1
7      return 1
8  for each b ∈ {0,1}^m
9      if MEMBER(f, x, p, R_b) == 1
10         return 1
11 return 0
```

MEMBER takes a computable $f : [0,1]^m \to \mathbb{R}^n$, a point $x \in \mathbb{R}^n$, precision p, and a $m \times 2$ matrix R. R corresponds to the hypercube

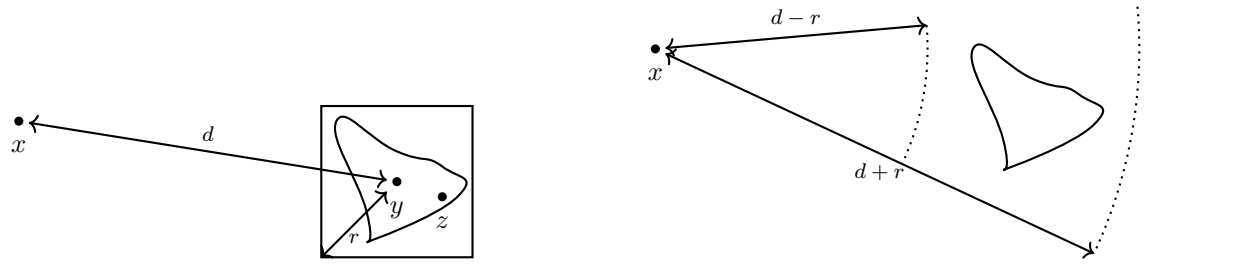
$$H_R := \prod_{j=1}^{m} [r_{j1}, r_{j2}] = [r_{11}, r_{12}] \times \cdots \times [r_{m1}, r_{m2}] \tag{2.2}$$

MEMBER(f, x, p, R) returns $\chi(x, p)$ for the compact set $\mathcal{C} = f(H_R)$. Refer to Fig 2.2. Thus, calling MEMBER(f, x, p, U) is equivalent to compute $\chi(x, p)$ for the target compact set $\mathcal{C} = f([0,1]^m)$ where $u_{j1} = 0, u_{j2} = 1$ for all j, which corresponds to $[0,1]^m$.

Line 1 computes $f(H_R)$ via the interval arithmetic, resulting in a hyperrectangle $H_{R'} \supset f(H_R)$.

Line 2-7 test the conditions ① and ② in Fig 2.2 over $H_{R'}$. If it is outside of ①, return 0. If it is inside of ②, return 1. See Base cases below for the detail.

If the two conditions above were not useful, we divide $R \subset \mathbb{R}^m$ into 2^m hypercubes of the same size. For each hypercube R_b, line 9 checks if there exists any R_b for which the membership test returns 1. If it exists, it means there exists $z \in \mathcal{C}$ on area A or B, hence we return 1 immediately on line 10. If all



(a) Compact set in a hyperrectangle (b) The lower and upper bounds of the distance to compact set

Figure 2.3: Distance to the compact set $f(H_R) =$ ▷. In (a), ▷ is included in a hyperrectangle □. x is the queried point for the membership test. y is the center of □. z is an arbitrary point on ▷. d is the distance from x and y. r is the distance from y to a corner of □. (b) shows that $d - r \leq |x - z| \leq d + r$ for all $z \in$ ▷.

sub-member testing returned 0, this indicates all $z \in \mathcal{C}$ is on area B or C, thus we return 0. See Dividing into sub-hypercubes below for the detail.

Base cases

Line 2-7 in MEMBER are explained here, in which it tries to decide the answer given $H_{R'}$. The idea is to find the upper and the lower bound of $|x - z|$ for $z \in f(H_R)$. Then, the lower bound is used to test if $f(H_R)$ is outside of ① in Fig 2.2. Similarly, the upper bound is used to test if $f(H_R)$ is inside of ②.

In Fig 2.3.a, an arbitrary $z \in f(H_R)$ is bounded by d and r. By the triangle inequality,

$$|x - z| = |x - y + y - z| \leq |x - y| + |y - z| \leq d + r$$

Also, by the reverse triangle inequality,

$$|x - z| = |x - y - (z - y)| \geq |x - y| - |z - y| \geq d - r$$

Therefore

$$d - r \leq |x - z| \leq d + r$$

as illustrated in Fig 2.3.b.

To this end, two more conditions are necessary as shown in Fig 2.4. The procedure will run forever if only ① is compared with the lower bound and they're the same. Therefore, ③ is added and used along with ① in parallel to resolve the decidability problem on comparing the lower bound. Similarly ④ is added for the upper bound.

Fig 2.5 shows all cases. If either (a) or (b) is true, then we immediately return 0 or 1. Otherwise the $f(H_R)$ is split further.

Dividing into sub-hypercubes

Line 8-11 in MEMBER are executed if none of the base cases above is satisfied. Fig 2.5.c exhibits a situation where MEMBER returns either 0 or 1 because $2^{-p} < |x - z| < 2^{-p+1}$ for all $z \in$ ▷. However, it is possible ③ was chosen when comparing the lower bound, meaning there exists $z \in$ ▷ with $|x - z| < 2^{-p}$, where MEMBER must return 1.

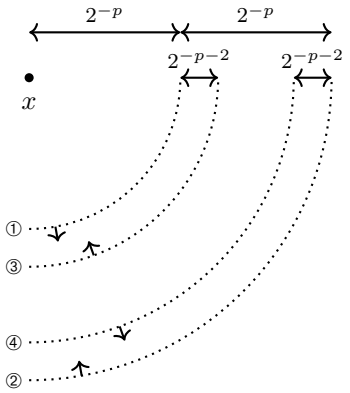


Figure 2.4: Conditions for the lower and upper bounds. Condition ③ and ④ are added to resolve the decidability problem. ① and ② are used to test the lower bound $d - r$, while ③ and ④ to test the upper bound $d + r$.

1. f has a modulus of continuity $\mu(k)$. That is, $\|z - z'\| \leq 2^{-\mu(k)} \rightarrow \|f(z) - f(z')\| \leq 2^{-k}$ for all $z, z' \in \mathrm{dom}(f)$.
2. It takes $T_f(p)$ time to compute $f(z)$ where z has the precision p.
3. T_f is an increasing function.

Table 2.1: Assumptions for the bit-cost complexity analysis

To make this happen, we divide the compact set $f(H_R)$ into smaller subsets. $H_R \subset \mathbb{R}^m$ is a hypercube, whose interval for j's dimension is $[r_{j1}, r_{j2}]$. We divide it by half for each dimension, hence we acquire 2^m sub-hypercubes of equal size.

On the line 8 in MEMBER, b selects which sub-hypercube to use for each iteration. We define $m \times 2$ matrix R_b that contains the interval information on its rows.

$$R_b := \begin{bmatrix} r_{11} + \frac{r_{12}-r_{11}}{2} b_1 & \frac{r_{11}+r_{12}}{2} + \frac{r_{12}-r_{11}}{2} b_1 \\ r_{21} + \frac{r_{22}-r_{21}}{2} b_2 & \frac{r_{21}+r_{22}}{2} + \frac{r_{22}-r_{21}}{2} b_2 \\ \vdots & \vdots \\ r_{m1} + \frac{r_{m2}-r_{m1}}{2} b_m & \frac{r_{m1}+r_{m2}}{2} + \frac{r_{m2}-r_{m1}}{2} b_m \end{bmatrix}$$

The j's dimension of H_R is $[r_{j1}, r_{j2}]$, which is splitted into $[r_{j1}, (r_{j1} + r_{j2})/2]$ and $[(r_{j1} + r_{j2})/2, r_{j2}]$. b_j chooses either of them. If $b_j = 0$, we pick up $[r_{j1}, (r_{j1} + r_{j2})/2]$, otherwise $[(r_{j1} + r_{j2})/2, r_{j2}]$. The combined expression is $\left[r_{j1} + \frac{r_{j2}-r_{j1}}{2} b_j, \frac{r_{j1}+r_{j2}}{2} + \frac{r_{j2}-r_{j1}}{2} b_j\right]$, which corresponds to the j's row of R_b.

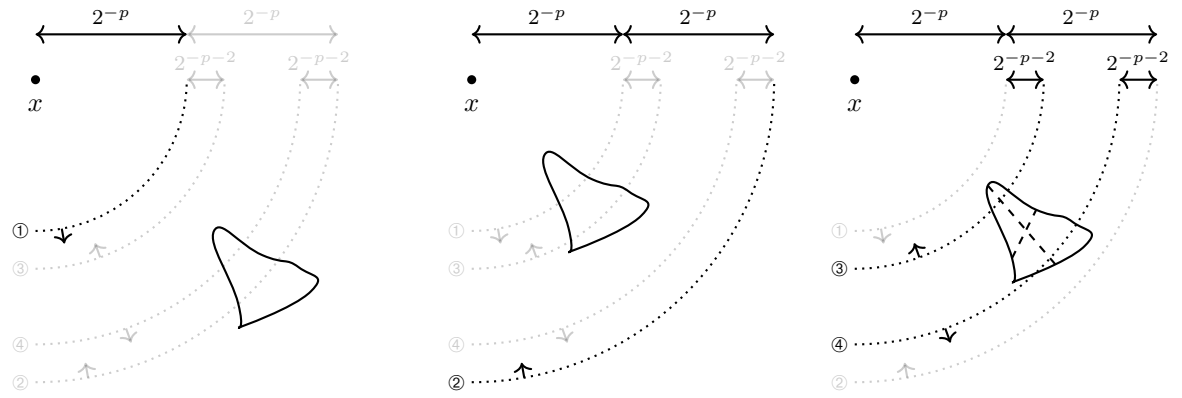
For each iteration, membership check done for $f(H_{R_b})$ on line 9. If any membership test returns 1, return 1 immediately. If not, return 0.

Time complexity

We analyze the worst case of MEMBER. Assume Table 2.1.

In the worst case, the situation in Fig 2.5.c keeps taking place until every divided compact subset becomes small enough to meet one of the base cases, shown in Fig 2.5.a and Fig 2.5. Fig 2.6 illustrates such subset.

MEMBER is called with $H_R =$ (the unit hypercube). Assume that the dividing occurred with depth of q. At the depth j, MEMBER is called at most 2^{jm} times, each taking $O(T_f(j))$. Therefore the total



(a) ① is satisfied (b) ② is satisfied (c) ③ and ④ are satisfied.

Figure 2.5: Possible cases of checking the boundaries of a compact set ▷. The tests are done in sequential : CHOOSE(①,③) for the lower bound, and then CHOOSE(②,④) for the upper bound. If ① turns out to be true as shown in (a), then return 0 (line 4-5 in MEMBER). If ③ is satisfied before ①, then test the upper bound. If ② is satisfied first as shown in (b), then return 1 (line 6-7 in MEMBER). If ④ is satisfied before ② as shown in (c), then we divide the compact set into smaller subsets and repeat this process for each subset. ▷ is divided by ⋰.

time complexity $T(f,p)$ is

$$T\left(f,p\right)=\sum_{j=1}^{q}2^{jm}O\left(T_f\left(j\right)\right)$$

Since T_f is increasing,

$$T\left(f,p\right)=O\left(T_f\left(q\right)\right)\frac{2^{qm+1}-1}{2^m-1}=O\left(T_f\left(q\right)2^{qm}\right) \tag{2.3}$$

A divided hypercube $H_R\subset\mathbb{R}^m$ at the depth q has a diagonal length $2^{-q}\sqrt{m}$. Let $2^{-q-1}\sqrt{m}\leq 2^{-\mu(k)}$ and we'll define k properly. By the definition of the modulus of continuity, $\left|f\left(z\right)-f\left(z'\right)\right|\leq 2^{-k}$ for all $z,z'\in\mathrm{dom}\left(f\right)$. Fig 2.6 states that every divided subset is included in a ball of radius $2^{-p-2}/\sqrt{n}$. If $2^{-k}\leq 2^{-p-2}/\sqrt{n}$, then Fig 2.5.c does not happen. Thus, let

$$k:=\left\lceil p+2+\frac{1}{2}\log_2 n\right\rceil$$

Similarly let

$$q:=\left\lceil \mu\left(k\right)-1+\frac{1}{2}\log_2 m\right\rceil$$

to satisfy $2^{-q-1}\sqrt{m}\leq 2^{-\mu(k)}$.

(2.3) becomes

$$T\left(f,p\right)=O\left(T_f\left(q\right)m^{m/2}2^{\mu(k)}\right)$$

Space complexity

Since MEMBER doesn't store a variable, maintained within the same depth, the space complexity is

$$O\left(q\right)=O\left(\mu\left(p+2+\frac{1}{2}\log_2 n\right)+\log_2 m\right)$$

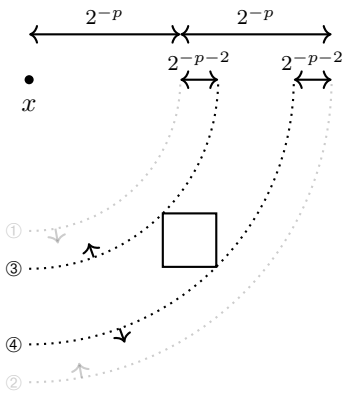


Figure 2.6: The smallest bound of divided compact subset □ for the procedure termination. □ is inscribed between ③ and ④ by the opposite corners. The two corners and x are on the same line.

2.3.2 Drawing 2D images

Similar to MEMBER, we use divide-and-conquer on drawing.

DRAWHOM$(f, R, x_1, x_2, y_1, y_2, image)$

1 Get a hyperrectangle $H_{R'}$ that includes $f(H_R)$.
2 $pixelSize = (x_2 - x_1)/imgWidth$
3 **if** $H_{R'} \cap [x_1, x_2] \times [y_1, y_2] \neq \emptyset$
4 **if** there exists a hypercube of side length *pixelSize* that includes $H_{R'}$
5 Set 1s to 2×2 cells in *image* that overlap $H_{R'}$.
6 **else**
7 **for** each $b \in \{0, 1\}^m$
8 DRAWHOM$(f, R_b, x_1, x_2, y_1, y_2, image)$

DRAWHOM takes a computable $f : [0, 1]^m \to \mathbb{R}^n$, a $m \times 2$ matrix R as input. R corresponds to the hypercube H_R 2.2. It also takes the range to draw $[x_1, x_2] \times [y_1, y_2]$, and a two-dimentional binary array *image*. The output is stored in *image*. A programmer must call DRAWHOM$(f, U, x_1, x_2, y_1, y_2, image)$ where $u_{j1} = 0, u_{j2} = 1$ for all j, which corresponds to $[0, 1]^m$, and *image* must be initialized 0 for all cells, meaning all white pixels. We assume that the image width and height are already determined in the same way `Compact` does.

Line 1 gets a hyperrectangle $H_{R'}$ that includes the target compact set $f(H_R)$.

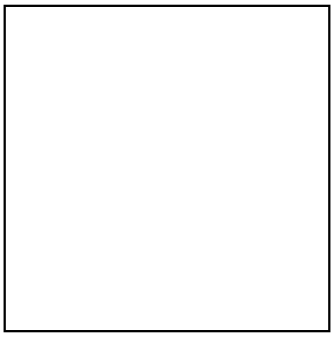
Line 2 computes the side length of a pixel.

Line 3 checks if $H_{R'}$ overlaps the target area $[x_1, x_2] \times [y_1, y_2]$. If not, no drawing is necessary. When checking the boundary, we add additional conditions to resolve the decidability problem.

Line 4 checks if $H_{R'}$ is small enough to be included in a hypercube of side length *pixelSize*. If this is satisfied, $f(H_R)$ must reside 2×2 pixels at most. Thus, We color the corresponding pixels.

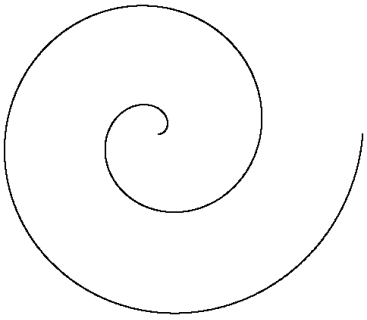
If $H_{R'}$ is not small enough, we split and delve into each subsets on line 7-8.

See Fig 2.7 for examples.



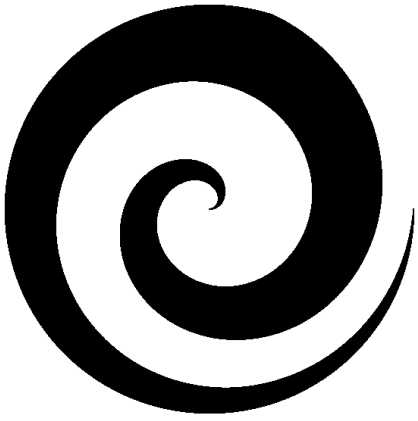
(a) `Compact` (Unit square)

(b) `Homotopy` 1 (Prolate cycloid)



(c) `Homotopy` 2 (Whirl 1)

(d) `Homotopy` 3 (Filled sin func.)



(e) `Homotopy` 4 (Whirl 2)

Figure 2.7: Examples of drawn images. (a) $f : [0,1]^2 \times \mathbb{N} \rightrightarrows \{0,1\}$, $f(u,v,p) = 1$ if the distance from (x,y) to unit rectangle is less than 2^{-p}. (b) $f : [0,1] \to \mathbb{R}^2$, $f(t) = (16\pi t + 3\sin(8\pi t), 3\cos(8\pi t))$. (c) $f : [0,1] \to \mathbb{R}^2$, $f(t) = (8\pi t\cos(8\pi t), 8\pi t\sin(8\pi t))$. (d) $f : [0,1]^2 \to \mathbb{R}^2$, $f(u,v) = (6\pi u, v\sin(6\pi u))$. (e) $f : [0,1]^2 \to \mathbb{R}^2$, $f(u,v) = (r\cos\theta, r\sin\theta)$ where $r = 4\pi u$ and $\theta = 4\pi u(2v+3)/5$.

Time complexity

We assume 2.1 again for the analysis and use a similar approach from MEMBER. We also assume that DRAWHOM ends at the maximum q the depth in the worst case. A divided hypercube $H_R \subset \mathbb{R}^m$ at the depth q has a diagonal length $2^{-q}\sqrt{m}$. Let $2^{-q-1}\sqrt{m} \leq 2^{-\mu(k)}$ and we'll define k properly. By the definition of the modulus of continuity, $|f(z) - f(z')| \leq 2^{-k}$ for all $z, z' \in \mathrm{dom}(f)$. Line 4 states that every divided subset is included in a ball of radius *pixelSize*. If $2^{-k} \leq \frac{\Delta x}{W}$ where $\Delta x = x_2 - x_1$ and W is the image width(number of pixels), then no more split takes place. Thus, define k as

$$k := \left\lceil -\log_2 \frac{\Delta x}{W} \right\rceil$$

Similarly let

$$q := \left\lceil \mu(k) - 1 + \frac{1}{2}\log_2 m \right\rceil$$

to satisfy $2^{-q-1}\sqrt{m} \leq 2^{-\mu(k)}$.

Then, the time complexity of DRAWHOM $T(f, p)$ is the same with that of MEMBER but k.

$$T(f, p) = O\left(T_f(q)\, m^{m/2} 2^{\mu(k)}\right)$$

Space complexity

Like MEMBER, DRAWHOM doesn't store a variable, maintained within the same depth. Therefore, the space complexity is

$$O(q) = O\left(\mu\left(-\log_2 \frac{\Delta x}{W}\right) + \log_2 m\right)$$

2.4 Conclusion

Euclidean compact subsets are managable via `Compact` and `Homotopy` in Exact Real Computation. Both of them commonly provide two functionalities: 1) given a point $x \in \mathbb{R}^n$, output if x is in a certain compact set, and 2) drawing the image of a certain compact set on real plane. `Compact` takes a characteristic function as an input, and a programmer should manually make sure that the characteristic function satisfies some conditions. However, `Homotopy` takes a computable homotopy $f : [0, 1]^m \to \mathbb{R}^n$, avoiding the cumbersome step above.

Chapter 3. Accelerating Intermediate-Precision Arithmetic

Being fundamental operations in many fields, real computations include, but are not limited to, Numerical Analysis, some areas of Artificial Intelligence, etc. They often demand computations in various forms such as matrices, polynomials and and so on. However, all these high-level operations are implemented in the four arithmetic operations, which implies that whenever they get improved, so is the overall speed of the computation.

This chapter concerns the multiplication performance of the real computation packages in practice such as `iRRAM` and `MPFR`, which often demands a lot of time. Any speed improvements are thus very valuable, and we aim at achieving them at the level of the basic arithmetic operations. Section 3.1 introduces the background. Section 3.2 inspects the mechanism how the acceleration is feasible. Section 3.3 exhibits the improvement in a quantitative manner.

3.1 Background

Modern computers deal with some mathematical concepts in limited ways. For instance, `int` as a primitive data structure stores an integer up to a finite bound. Working with the integers beyond it requires an additional effort. Established in 1985, on the other hand, IEEE [11] introduced 32-bit and 64-bit floating-point for the real numbers. Having 23 and 53 bits precision and supported by hardware such as FPU [16], they have been widely used in engineering where fast real computations are demanded. For example, Tensorflow [20] and Pytorch [21] utilize 32-bit floating-point as the default primitive data structure.

However, the floating-points above suffer from the insufficient precision [18, 19, 26]. This is where an intermediate-precision comes in. We call a precision **intermediate** if it's more than 53 bits but less than 640 bits, **high** if larger than 640 bits, and **low** otherwise.

Equipped with many popular math software such as *Matlab*, *Mathematica* and *Geogebra*, intermediate-precision provide a more reliable computation. Some hardware have implemented intermediate-precision. 128-bit floating-point computation is offered by IBM in various instruction set architectures [12, 13, 14] and by *RISC-V* system [15]. Moreover, *x87* chipset [16], planted on the most recent PCs, also provides 80-bit floating-point computation as an extension of 64-bit one.

Meanwhile, the intermediate-precision plays an important role in high precision computations. Many advanced algorithms for the high-precision multiplication including *Karatsuba* [22, 23], *Toom-Cook* [24], *Schönhage–Strassen* [25] take advantage of divide-and-conquer strategy in which a long mantissa is split into shorter ones of an intermediate size. As multiplication is at the crux of other operations such as matrix or polynomial multiplications, we explain how to accelerate the multiplication of two intermediate-precision numbers and observe its effect via experiments.

3.1.1 Data structure for arbitrary precision

There are two major data structures to manipulate arbitrary precision numbers: multi-digit and multi-component. We explore how these are defined on Sec 3.1.2, followed by their properties on Sec 3.1.3 and 3.1.4.

Table 3.1: Comparison of Floating-point, Multi-digit and Multi-component Floating-point, Multi-digit and Multi-component. The sign of each data structure is omitted. Multi-digit has a arbitrary long fractional part, while multi-component consists of many floating-points. We call $\texttt{frac}_j$ in multi-digit a limb and $\texttt{frac}_j$ in multi-component a component

	Floating-point	Multi-digit	Multi-component
Illustration	frac | exp p q	$\texttt{frac}_1$ | $\cdots$ | $\texttt{frac}_{n+1}$ | exp p p q	$\sum_{j=1}^{n}$ $\texttt{frac}_j$ | $\texttt{exp}_j$ p q
Formula	$\texttt{frac} \times 2^{\texttt{exp}}$	$(\texttt{frac}_1 \| \| \cdots \| \|\texttt{frac}_{n+1}) \times 2^{\texttt{exp}}$	$\sum_j \texttt{frac}_j \times 2^{\texttt{exp}_j}$
Precision	Fixed $(\texttt{p} = 24, 53, \cdots)$	Arbitrary$(\texttt{np})$	Arbitrary$(\texttt{np})$
Implementation	Hardware	MPFR	DD, QD$(\texttt{n} = 2, 4)$
Mul. algo.	Long	Long, Toom-Cook, Schönhage–Strassen	Long

3.1.2 Multi-digit vs Multi-component

Table 3.1 shows their properties along with those of the canonical floating-point.

Inheriting the concept of separating fractional and exponent part from the floating-point, multi-digit and multi-component are considered extensions of the floating-point. Multi-digit simply extends the length of fractional part to an arbitrary length via an array of unsigned integers, called **limbs**, each of size p. For example, a binary number $\overbrace{1010\cdots10}^{128}$ is equal to $0.\overbrace{1010\cdots10}^{128} \times 2^{128}$, $((\overbrace{1010\cdots10}^{64}, \overbrace{1010\cdots10}^{64}), 128)$. Note that many implementation use a little larger storage than needed for performance aspect. In the illustration p bits were additionally used to contain a np-bit number. See Sec 3.1.3 for the details. On the other hand, multi-component utilizes multiple identical floating-points, called **components**, representing a real number as an unevaluated sum of them. To give you an example, the aforementioned binary number $\overbrace{1010\cdots10}^{128}$ is equal to $0.\overbrace{1010\cdots1}^{53} \times 2^{128} + 0.\overbrace{1010\cdots1}^{53} \times 2^{74} + 0.\overbrace{1010\cdots1}^{19} \times 2^{20}$, hence we store $(\overbrace{1010\cdots1}^{53}, 128)$, $(\overbrace{1010\cdots1}^{53}, 74)$ and $(\overbrace{1010\cdots1}^{19}, 20)$ separately. Note that both examples may slightly vary depending on the implementation.

The traditional floating-point supports only fixed precision(p). Meanwhile, both multi-digit and multi-component can deliver an arbitrary precision number(np) by either increasing the number of elements for fractional part or using more components.

There are several hardware for the floating-point as we enumerated in the Sec 3.1. MPFR [1] is a representative implementation of multi-digit on which many software including but not limited to the ones in the Sec 3.1 depend. Multi-component has been implemented for some fixed number of component such as DD and QD [27], etc. [28, 29, 30]

It is important to construct efficient algorithms for data structures. We explore multiplication algorithms in this paper. Multiplying two floating-points is already wired on circuit, thereby, no software-level algorithm is necessary. Meanwhile, there are some algorithms for the high-precision multiplication based on multi-digit such as the ones in Sec 3.1, while multi-component doesn't have such an algorithm. This difference comes from the simplicity of their structures. The fractional part of multi-digit is easily partitioned into several limbs of a constant-size like 64bits, giving algorithm designers more chance of finding an efficient algorithm. In contrast, multi-component doesn't show such an alignment property, giving less chance of achieving the same goal.

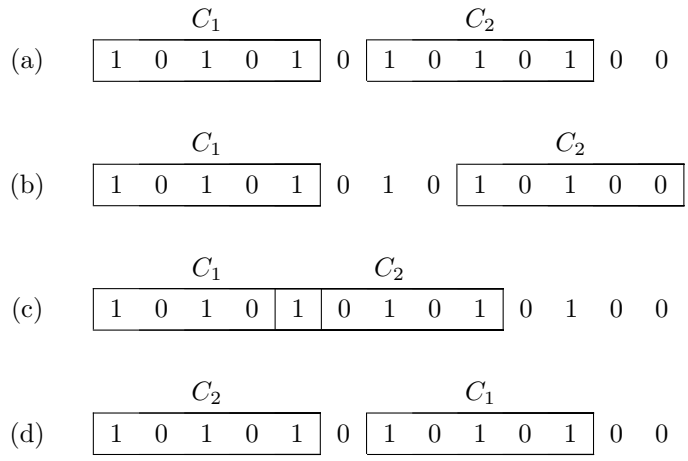


Figure 3.1: Correct and incorrect component assignments of two components for a binary number $\texttt{x} = \texttt{1010101010100}$. We assume that a component can at most 5 bits for explanation. (a) correctly assigned the two components, while the rests didn't. (b) doesn't express the best approximation to $\texttt{x}$. (c) has an overlap between C_1 and C_2. In (d), the components are not sorted in descending order.

3.1.3 Position of leading 1 in multi-digit

When representing non-zero number, each data structure above traces where the most significant 1 is for optimizing the consecutive computations. The canonical floating-point has no interest as it has only one component in which the leading 1 always lies on the first position of the fractional part. Moreover, whenever a computation takes place, hardware will maintain this property implicitly. Similar to the canonical floating-point, multi-component maintains the leading 1 for each component in the same way where the canonical floating-point does. That is, the fractional part of every component starts with 1.

Multi-digit, however, doesn't trace the exact position of it for performance issue. For instance, $x = \texttt{0.}\overbrace{1\cdots1}^{\texttt{128}}$ is represented as $((\overbrace{1\cdots1}^{\texttt{64}}, \overbrace{1\cdots1}^{\texttt{64}}), \texttt{0})$. If we want to have the fractional part always start with 1, then $x - \texttt{0.1}$ must be $((\overbrace{1\cdots1}^{\texttt{64}}, \overbrace{1\cdots10}^{\texttt{63}}), -\texttt{1})$, acquired by shifting each element for the fractional part. If there are n elements in the fractional part, then the addition will take $O(n)$ time.

For this reason, multi-digit often utilize one more limb. If we want to represent a number of precision $\texttt{np}$, where $\texttt{p}$ is the size of a limb, then we use $\texttt{n+1}$ limbs and maintain the most significant integer as non-zero. This sloppy condition prevents the aforementioned problem and gives more chance to correctly round to the desired precision because it has longer fractions than required.

3.1.4 Descending order and non-overlap of multi-component

A multi-component keep making its components sorted and not overlapped to its neighbor component to express the best approximation to the available precision. Example of correct and incorrect component assignments are shown in Fig 3.1.

3.2 Long multiplication analysis

Although the advanced multiplication algorithms run faster than the long multiplication does asymptotically, we must conceive the overhead in practice. This is why the long multiplication is used for the

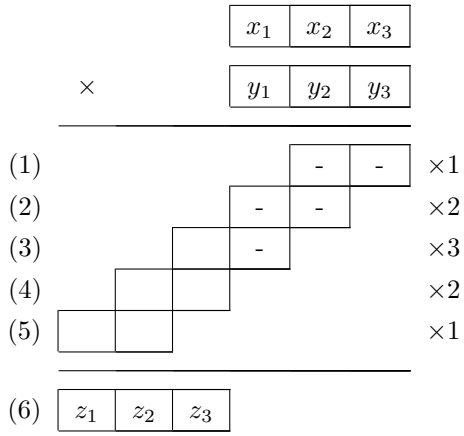


Figure 3.2: Multiplication procedure of multi-digit. x, y and z are contained in 3-limb multi-digits where each limb has p-bit precision. (1)= x_3y_3. (2)= $x_2y_3 + x_3y_2$. (3)= $x_1y_3 + x_2y_2 + x_3y_1$. (4)= $x_1y_2 + x_2y_1$. (5)= x_1y_1. (6) = (3)+(4)+(5). [-] is not computed.

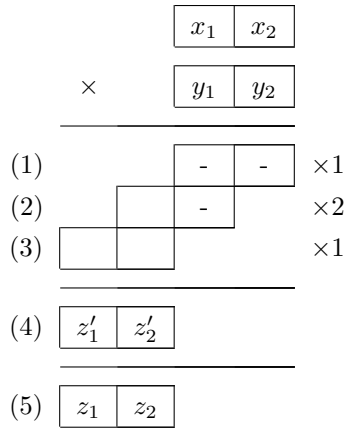


Figure 3.3: Multiplication procedure of multi-component. x, y and z are contained in 2-component multi-digits where each component has p-bit precision. Note that $x = x_1 + x_2$ and $y = y_1 + y_2$. (1)= x_2y_2. (2)= $x_1y_2 + x_2y_1$. (3)= x_1y_1. (4)=(2)+(3). (4) is renormalized into (5). [-] is not computed.

intermediate-precision. The long multiplication works similarly on both multi-digit and multi-component. Yet, the number of necessary primitive operations are different.

Sec 3.2.1 describes how the multiplication is done for each data structure. Sec 3.2.2 explains how many bits are lost when a multiplication takes place. Sec 3.2.3 inspects the performance by counting the number of primitive operations.

3.2.1 Long multiplication

We explore the base case. Let x and y be $2p$-bit numbers where p is the length of a limb and a component. We want to compute $z = xy$ with multi-digit and multi-component, respectively.

Multi-digit uses 3 limbs to contain a $2p$-bit number as explained in Sec 3.1.3. Fig 3.2 illustrates the computation procedure of z. The limb-wise multiplications are drawn as (1), $\cdots$, (5), which are summed up to make z on the last line. Note that we don't need to compute [-] s because they are too small to affect the final 3-limb result z.

On the other hand, multi-component spends two components to contain a $2p$-bit number. The

multiplication procedure looks similar except the last stage, shown in Fig 3.3. z_1' and z_2' may overlap with each other, violating one of the maintenance rules of multi-component. To resolve this issue, we must renormalize (Sec 3.1 in [27]) (4), resulting in (5).

3.2.2 Correctness

The multiplication on multi-digit produces the best approximation. That is, if x and y in the Fig 3.2 are exact, then z contains the most significant $2p$ bits of xy.

However, multi-component loses the last two bits. Assume that x and y in the Fig 3.3 are exact and that $|x|, |y| \in [0.5, 1)$ without loss of generality. By the properties in Sec 3.1.4, $|x_1|, |x_1| < 2^{-p}$. The absolute error $xy - z$ is

$$|xy - z| = |x_0y_1 - x_0 \otimes y_1 + x_1y_0 - x_1 \otimes y_0 + x_1y_1|$$

where $a \otimes b$ is the most significant p bits of ab. By the triangular inequality,

$$|xy - z| \leq |x_0y_1 - x_0 \otimes y_1| + |x_1y_0 - x_1 \otimes y_0| + |x_1y_1|$$

$|x_0y_1 - x_0 \otimes y_1|$ is the least significant p bits of x_0y_1, which is less than 2^{-2p}. Thus,

$$|xy - z| < 2^{-2p} + 2^{-2p} + 2^{-2p} < 2^{-2p+2}$$

Therefore, we lose only the last two bits, reducing the number of primitive operations.

3.2.3 Performance

There are three primitive operations in Fig 3.2 and Fig 3.3 : 1) $\square \times \square \to \square\square$, 2) $\square \times \square \to \square$ and 3) $\square + \square \to \square$. 1) multiplies two limbs (components) and produces one 2-limb (2-component) number without losing any information. 2) also multiplies two limbs(components), but it retains only the significant one. 3) adds two limbs(components) and produces one limb(component), in which a carry is assumed to be handled without a hassle with the help of hardware such as the instruction `ADC` or `FADD` in `x86`.

We now consider the general case where the multiplication is applied on np-bit numbers. Multi-digit uses $n+1$ limbs and multi-component uses n components to a np-bit number.

On the multi-digit multiplication, we compute x_iy_j of type 1) for $i + j \leq n + 1$, incurring $1 + 2 + \cdots + n = n(n+1)/2$ multiplications. Also we need x_iy_j of type 2) for $i + j = n + 2$, incurring $n + 1$ multiplications. Lastly, $2 + 4 + \cdots + 2n = n^2 + n$ additions of type 3) are necessary to get z.

On the multi-component multiplication, x_iy_j of type 1) for $i+j \leq n$ of type 1) are needed, meaning $1 + 2 + \cdots + (n-1) = n(n-1)/2$ multiplications. x_iy_j of type 2) for $i + j = n + 1$ incurs n multiplications. z' is acquired by adding these x_iy_j, requiring $2 + 4 + \cdots + (2n-2) = n^2 - n$ additions of type 3). Renormalizing z' requires $3(n-1) + 3(n-2) = 6n - 9$ additions of type 3), considering a `Quick_Two_Sum()`(Alg 3 in [27]) takes one additions and two subtractions.

Table 3.2 summarizes the number of primitive operations for each data structure. Since `(a)-(b)` is positive for all operations when $n = 2$, multi-component must outperform multi-digit for computing $2p$-bit number.

However, this is not true when we increase the precision. `(a)-(b)` for operation 3) is $-4n + 9$, implying that multi-digit will run faster than multi-component does as $n \to \infty$. Since we aim the improvement in practice, we have conducted experiments to figure out the break-even point between the two data structures.

Table 3.2: The number of primitive operations on multiplying two np-bit numbers. The $+(6n-9)$ at operation 3) on multi-component is for the renormalization.

Operation	(a) Multi-digit	(b)Multi-component	(a)-(b)
1) $\square \times \square \to \square\square$	$n(n+1)/2$	$n(n-1)/2$	n
2) $\square \times \square \to \square\,\square$	$n+1$	n	1
3) $\square + \square \to \square\,\square$	n^2+n	$\left(n^2-n\right)+(6n-9)$	$-4n+9$

3.3 Experiments

It is difficult to measure the performance of a single multiplication. To see the performance difference, we've tested matrix and polynomial multiplications, which are widely used operations in Engineering.

In the matrix multiplication test, we generate two $n \times n$ matrices A and B with each element having k-bit mantissa, and compute AB. We compare the elapsed time taken by multi-digit and that by multi-component.

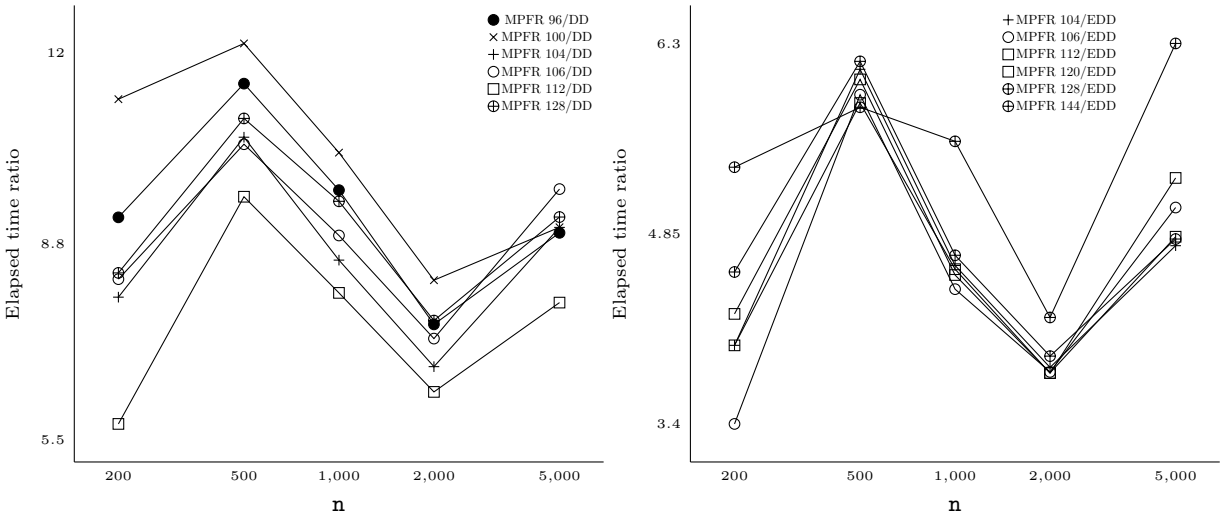
We've used the popular library `MPFR` for multi-digit which provides an arbitrary precision. Multi-component is provided with some fixed precisions — 106 and 212 bits by `DD` and `QD`, respectively, although the actual precision is reduced for a reason explained in Sec 3.2.2. Also we tried the modified `DD`, called `Extended Double-Double(EDD)`. `DD` utilizes IEEE `double` as a primitive data structure. Supported by the *x87* chipset, we've replaced it with `extended double` that offers 64-bit mantissa, so that `EDD` provides 128 bits in total.

The `MPFR` was tested with changing precision that corresponds to those of `DD`, `EDD` and `QD`, roughly $k \approx 106, 128$ and 212 bits.

The PC specs used in the experiments are *Intel i7-7700, 16GB and Ubuntu 18 x64* and compiled with *gcc7.5*.

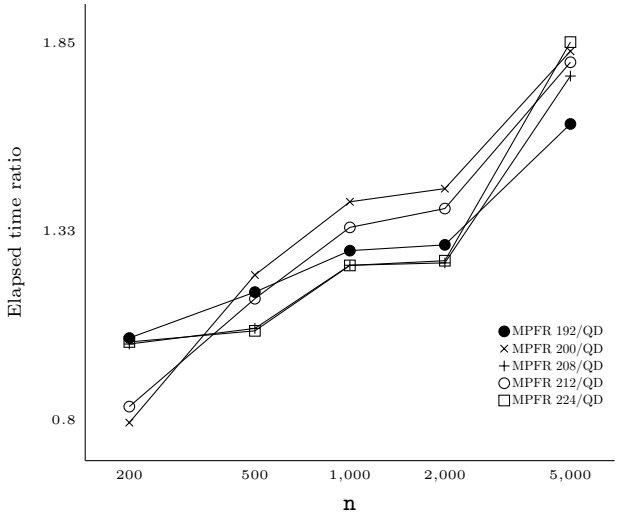
Fig 3.4 summarizes the result. See Table 6.1 for the full data. We've plotted the elapsed time ratio to compare the performances more easily. `DD` outperformed `MPFR` around its precision. Since `MPFR` is designed to offer an arbitrary precision, there is some overhead such as branch instructions, which gives rise to even more performance gap. `EDD` is faster than `MPFR` but it's less efficient than `DD`. When it comes to `QD`, multi-component isn't faster than multi-digit. Neither `MPFR` nor `QD` has shown a clear dominant result.

The polynomial multiplication test has been conducted in a similar way. We generate two polynomials $p(x)$ and $q(x)$ of order n with each coefficient having k-bit mantissa, and compute AB. We compare the elapsed time taken by multi-digit and that by multi-component.



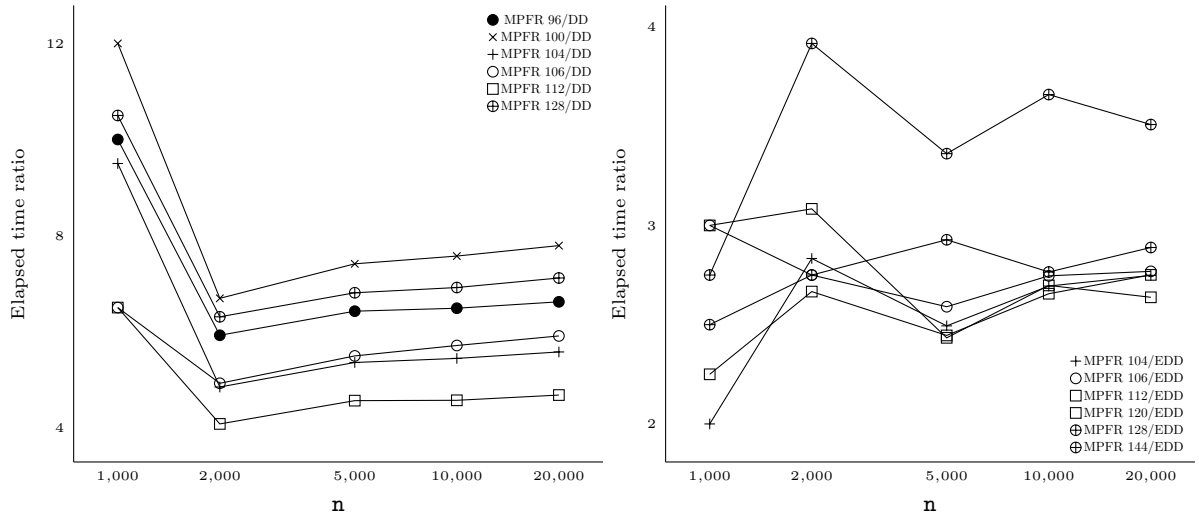
(a) `MPFR` / `DD`

(b) `MPFR` / `EDD`



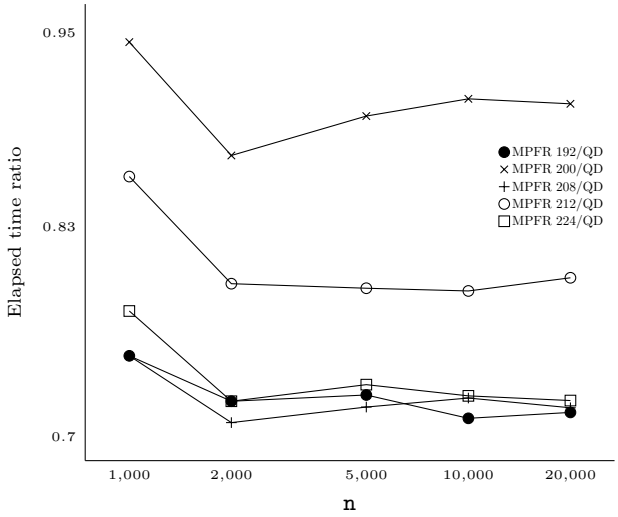
(c) `MPFR` / `QD`

Figure 3.4: Matrix multiplication speed comparison of `MPFR`, `DD`, `EDD`, and `QD`. (a) shows the elapsed time ratio of `MPFR` to `DD`. Regardless of the various precisions, `DD` was faster than `MPFR` from five to twelve times. (b) illustrates a similar aspect regarding `EDD` but the ratio is reduced to $(3.4, 6.3)$. (c) shows the elapsed time ratio of `MPFR` to `QD`. The ratios were plotted around 1, meaning no sizable performance difference.



(a) MPFR / DD

(b) MPFR / EDD



(c) MPFR / QD

Figure 3.5: Polynomial multiplication speed comparison of MPFR, DD, EDD, and QD. (a) shows the elapsed time ratio of MPFR to DD. Regardless of the various precisions, DD was faster than MPFR from four to twelve times. (b) illustrates a similar aspect regarding EDD but the ratio is reduced to $(2, 4)$. (c) shows the elapsed time ratio of MPFR to QD. The ratios were plotted below 1, meaning that MPFR computes faster than QD.

Fig 3.5 illustrates a similar aspect to that of the matrix test. DD and EDD outperformed MPFR around their precisions, while QD underperformed MPFR.

Judging by the two tests above, **the break-even point is $k \approx 212$ bits**

3.3.1 Why EDD is less efficient than DD

It is worthwhile to discuss the result of EDD. There are difference between DD and EDD when implementing the primitive operation 1) in Sec 3.2.3. DD utilizes `Fused Multiply-Subtract(FMS)` for 1), incurring a little cost. However, EDD lacks FMS or a similar instruction to achieve the same goal. Thus, the implementation should combine other instructions, which takes a high cost.

3.4 Conclusion

Multi-digit and multi-component treat a number differently. Multi-digit assigns a subsequent space to store the fractional part of a number, while multi-component utilizes several floating-points. On multiplication, multi-component outperforms multi-digit up to the precision about 212 bits. Therefore, if we replace the most prevalent library for the real computation, `MPFR` or `iRRAM`, with `DD` or `EDD`, then the computation runs faster at least four or two times.

Chapter 4. Simulating Fine Dust Air Purifier

For encouraging collaborations and convergence researches, the project named *A localized air purification system in the city to prevent pedestrians from the exposure to fine dust* had been conducted from July 2019 to June 2020 in collaboration of the three laboratories in KAIST: Environment Back To Environment Lab, Maturepolis Lab, and Complexity and Real Computation Lab. As the title implies, the goal was to design a purifier for improving the air quality in metropolises. The first two labs worked on figuring out how to convert the contaminated air into the clean one considering the efficiency and economics, while the third lab took a responsibility of distributing the clean air.

Section 4.1 explains why air quality matters and what considerations are relevant. Section 4.2 introduces the purifier prototype designed by Prof. Jongin Han, which we analyzed by computational software, as described in Section 4.3.

4.1 Background

Since the mankind has appeared on the earth, the population became higher than ever before with the power of the industrialization [31, 32, 33]. Unfortunately, this phenomenon gives rise to several environmental problems including higher temperature, contaminated river, etc. Among them, we focus on the air pollution, which affects human's health intensively [34] and deliberately [35].

There are two main strategies to manage it: 1) remove the pollution source and 2) improve the air quality around humans. The first approach sounds ideal, yet it is often impractical for some reasons. For example, consider a factory that generates sulfur dioxide(SO_2) while the production. The factory will be reluctant to install a preventing facility if it has no economical benefit [37] or there is a technical issue. Perhaps a government isn't willing to force the factory to do it for a political reason. On the other hand, purifying the air around people is free from such problems. This is why we aim to make and place an air purifier in the middle of city.

When it comes to the purification mechanism, traditional solutions such as a High-Efficiency Particulate Air(HEPA) filter [36] are discouraged in favor of a wet scrubber [38] system, which consists of machine and water. The absence of a chemical compound gives us high economic feasibility and easy maintenance. Furthermore, the scale-up is done without much hassle because of its simple structure.

Having these merits, the clean air shelter was introduced, planned to be placed on sidewalks and on the top of buildings.

4.2 Clean air shelter

Clean air shelter is an air purifier in form of a wet scrubber. Fig 4.1 shows the original experimentally designed prototype. The polluted air above the machine goes into the machine along with water from the pipe. Then, the impeller shatters the water into droplets, which absorb the particulate matters inside the polluted air into themselves. The heavy droplets fall into the tank in the bottom, while the fresh air goes out through the gap between the wet chamber and the tank.

Requiring water and electricity only, clean air shelter provides a shelter in which people take clean air without the repeated filter replacement. Fig 4.2 illustrates the PM density distribution around the

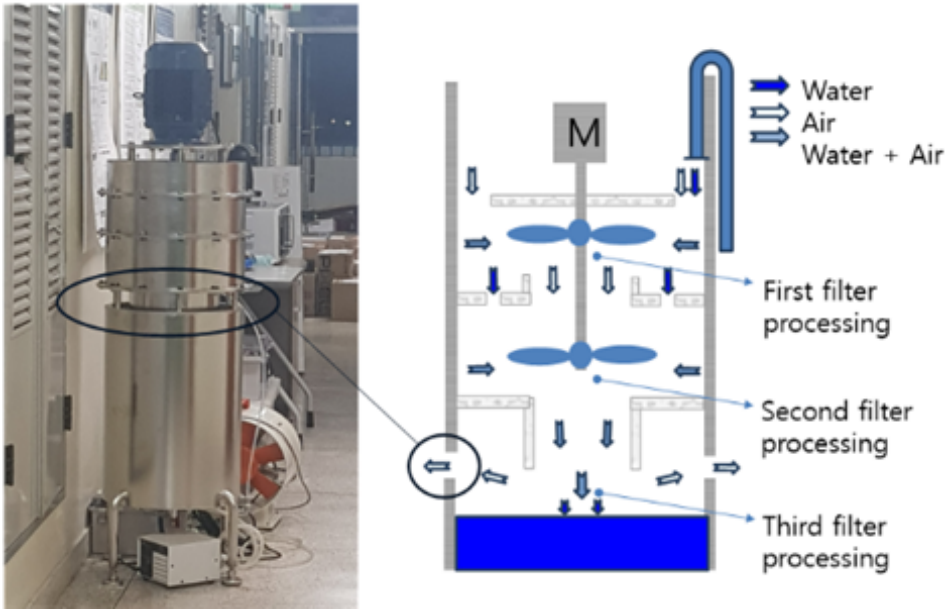


Figure 4.1: The prototype of clean air shelter designed by Prof. Jongin Han. Air enters the open top, goes through the purification process, and exits through the passage at the middle of the machine.

initial prototype. It emits the cleaned air upwards, thereby affecting people indirectly. An improvement is available by guiding the clean air to be emitted toward people directly. It also increases the efficiency because the cleaned air will not likely go through the filter process again.

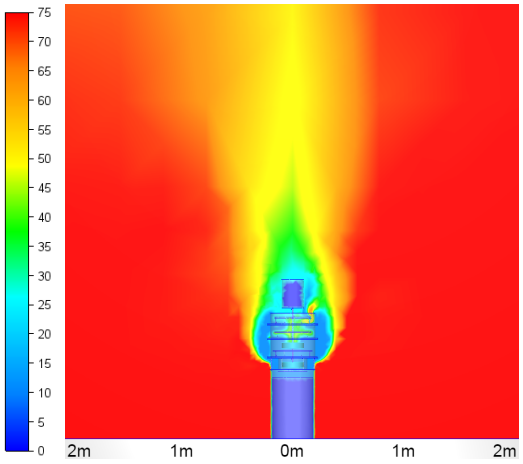


Figure 4.2: The particulate matter(PM) density in $\mu g/m^3$. The PM density less than $30\mu g/m^3$ is considered good. Bad if it is larger than $75\mu g/m^3$ or more.

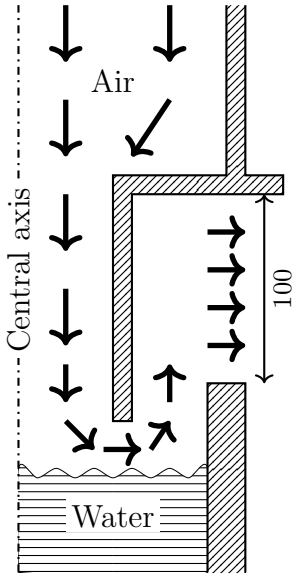
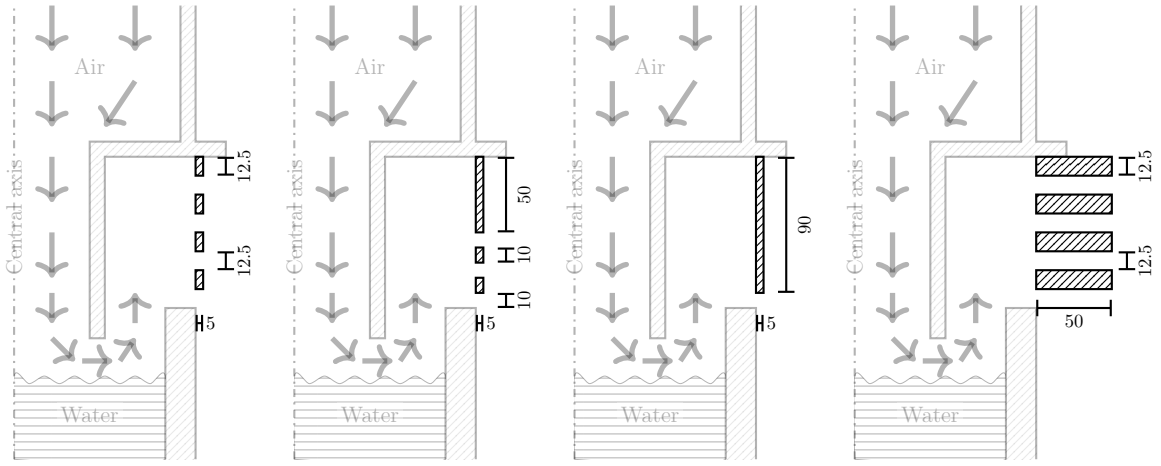


Figure 4.3: The section view of the prototype. The air passes the outlet of height 100mm.

The way to improve it is modifying the outlet shape, which was too wide (100mm in Fig 4.3) to spread the air horizontally. We've decided to attach a cover to make it have a narrower exit as either redesigning from the scratch or modifying the prototype would raise a high cost. Which shape of a cover would successfully form the clean air? The next section provides the answer.



(a) Model 1 (b) Model 2 (c) Model 3 (d) Model 4

Figure 4.4: Cover models(mm)

4.3 Simulation

Before manufacturing covers, we take advantage of a computer simulator to save fund. The goal is to design an outlet of the clean air shelter with two metrics, PM density drop and clean area size. The bigger these quantities are, the better performance is indicated.

The software for this virtual experiment are *Autodesk Inventor 2020* for modeling the purifier and the covers and *Autodesk CFD 2019* for running simulations.

4.3.1 Independent variables

There are two indepenent variables: cover model and height.

The cover must be designed to emit air horizontally, meaning that the outlet size is essentially less than that of the original prototype. However, we shouldn't go too far since a reflux is more likely to take place. Four models are introduced in Fig 4.4, which costs little material.

Model 1 has four gaps of height 12.5mm, while Model 2 and 3 have two and one gaps of height 10mm. That is, we decreased the total gap height by about a half of the previous model. Similar to Model 1, Model 4 has the same gap positions and heights formed by the prolonged cover size (50mm).

On the other hand, the height of the machine affects the distribution of air. Therefore, for each model, we've tested three different heights, 0m, 1m and 2m, from the ground by lifting the purifier so as to find a better installation height.

4.3.2 Simulation environment

Fig 4.5 illustrates the configuration. The clean air shelter is placed on the ground. Fine dust is generated on the boundary of a cylinder having 6m radius and 8m height. Note that the cylinder is hypothetical. There is no wall that surrounds the purifier. This configuration makes sense because we design an outdoor product and dust will come from somewhere instead of generated inside or near the machine. The specifications are as follow

- PM density of fine dust: $75\mu g/m^3$ (PM2.5 Interim target-1, 24-hour in [39])

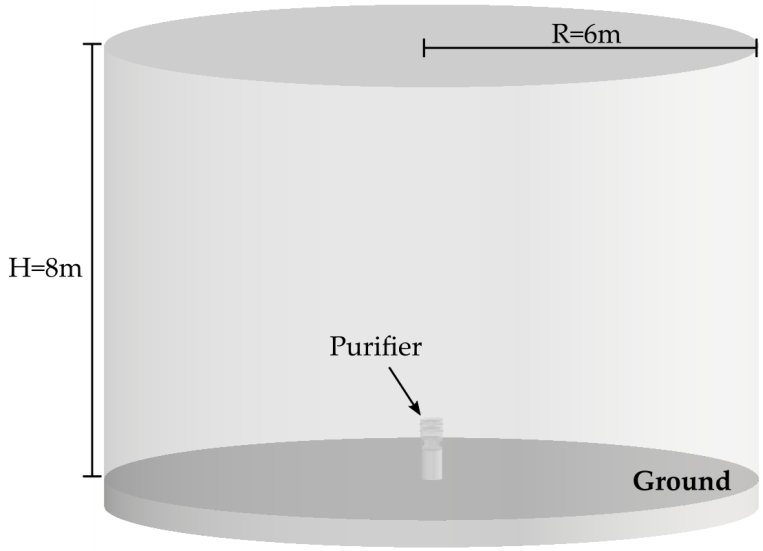


Figure 4.5: The virtual experiment setup for testing performance of the clean air shelter

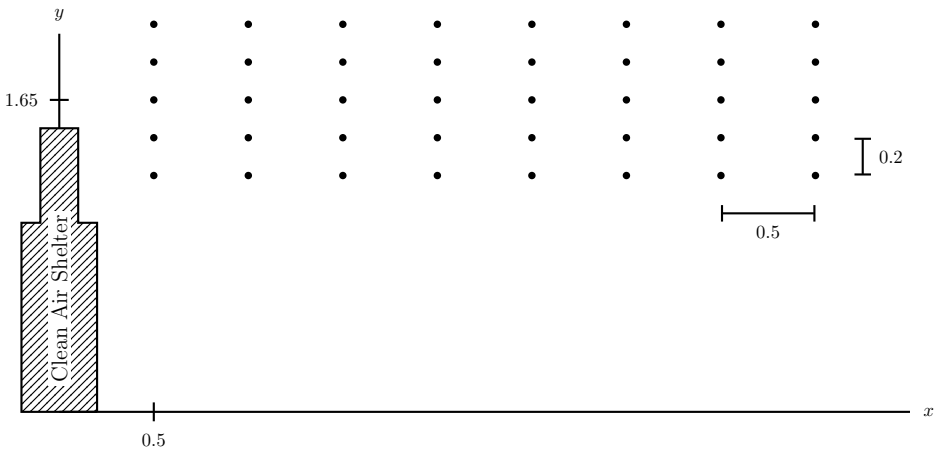


Figure 4.6: PM sensors(m)

- PM density of purified air: $10\mu g/m^3$
- Volumetric flow rate of motor: $50m^3/min$
- Fan speed: 1700RPM

The PM sensors are positioned in the grid formation, shown in Fig 4.6. There are five rows centered on the 1.65m height, which is intended and will be explained in Sec 4.3.3. Spaced by 0.5m, there are eight columns to measure the PM density over different distances from the machine.

4.3.3 Results and analysis

The PM density distributions are visualized in Fig 4.7. Model 1 shows the similar result to that of the original prototype, hence we skipped the lifting tests for model 1. Model 2 successfully emitted the air horizontally, although the PM density was not improved below the outlet when it was lifted. This is because the air flows upwards just before going out and the cover in model 2 didn't change direction clearly. Having only one outlet passage, model 3 pushes the air to the downward. When lifted by 1m, it

Table 4.1: Total score for models(μg/m^3). Smaller is better.

Models	Weighted average
Model 1 lifted by 0m	7.094
Model 2 lifted by 0m	7.091
Model 2 lifted by 1m	6.103
Model 2 lifted by 2m	7.087
Model 3 lifted by 0m	7.083
Model 3 lifted by 1m	7.039
Model 3 lifted by 2m	7.060
Model 4 lifted by 0m	4.788
Model 4 lifted by 1m	4.676
Model 4 lifted by 2m	5.037

formed a clean-air space below the machine. Unlike other models, model 4 pushed the air horizontally, filling the space below even in the lifted cases.

We define the total score as the weighted average of the sensor readings to pick up the best option.

$$(\text{Total score}) = \frac{1}{40} \sum_{i,j} w(y_i) \cdot \text{PM}(x_j, y_i)$$

where $\texttt{PM}(x_j, y_i)$ refers to the sensor reading at (x_j, y_i) and w is the weight function which depends on the height of people as they're beneficiaries. We take the height of total population of Korea in 2015, which has the mean of 165.42cm and the standard deviation 9.174cm (sizekorea.kr). Assuming it is normally distributed, we define w as the probability density function(PDF) that has the same mean and standard deviation.

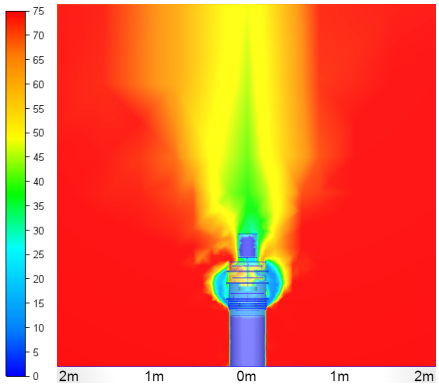
$$w(y) = f\left(\frac{y - 1.65}{0.09174}\right) \quad (y \text{ in m})$$

where $f(x)$ is the PDF of normal distribution.

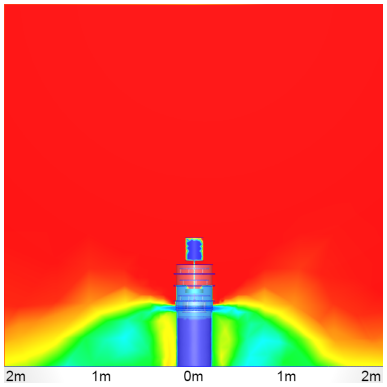
Note that the weight function isn't affected by the horizontal distance from the clean air shelter. Also, since the PM density is better when it has lower value, so is to total score. Table 4.1 summarizes the total score for each model. Check Table 6.3 and Table 6.4 in chapter 6 for the full data.

Having the minimum total score, the **model 4 lifted by 1m** is the best case.

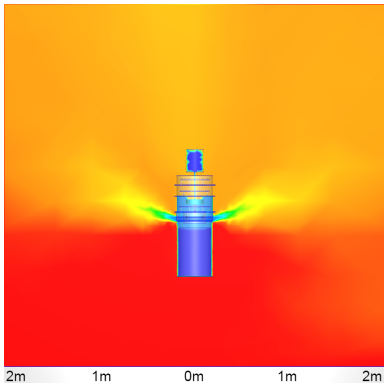
So how large is the created clean area? If PM density less than 50μg/m^3 (PM2.5 Interim target-2, 24-hour in [39]) is considered to be fresh, then **the best case makes a circular clean area of radius 3.7m** (Fig 4.8). In other words, when we install the clean air shelters in a row, spaced by 3.7m, a clean air passage of width 7.4m is generated.



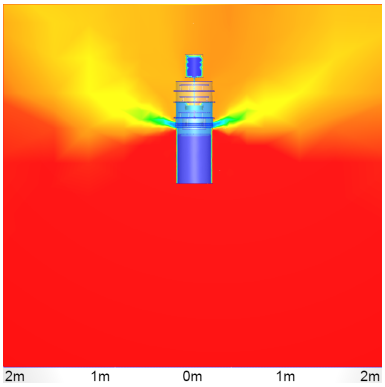
(a) Model 1 lifted by 0m



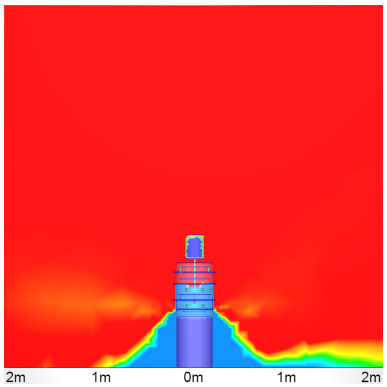
(b) Model 2 lifted by 0m



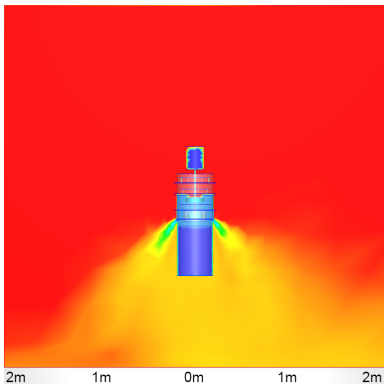
(c) Model 2 lifted by 1m



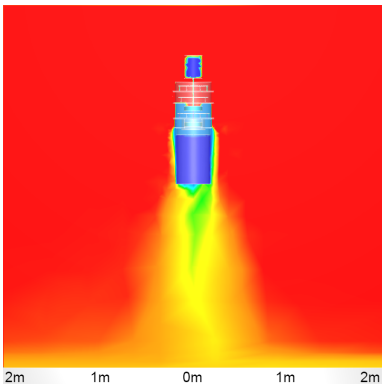
(d) Model 2 lifted by 2m



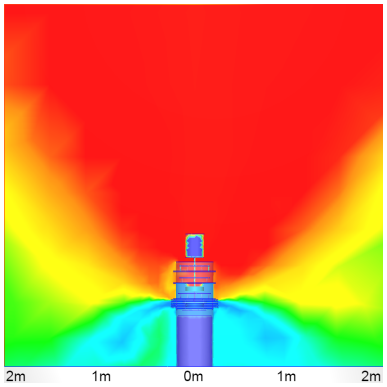
(e) Model 3 lifted by 0m



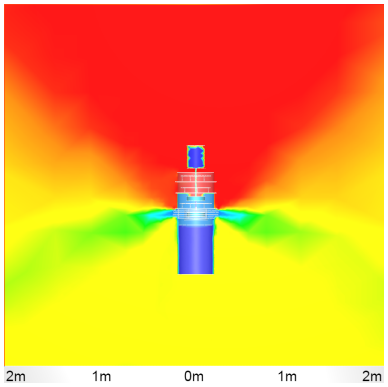
(f) Model 3 lifted by 1m



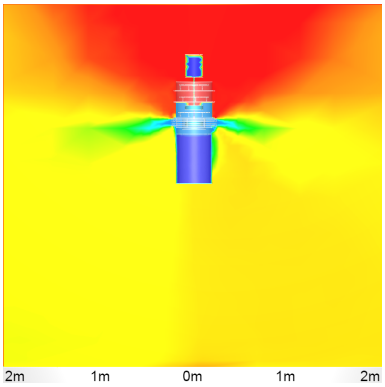
(g) Model 3 lifted by 2m



(h) Model 4 lifted by 0m



(i) Model 4 lifted by 1m



(j) Model 4 lifted by 2m

Figure 4.7: PM density distributions the models($\mu g/m^3$)

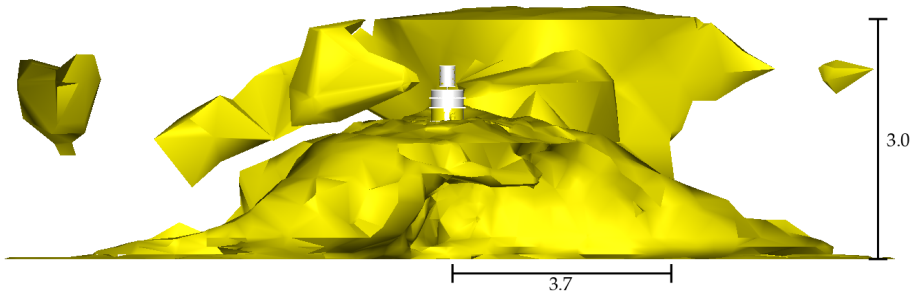


Figure 4.8: Clean area(m), model 5 lifted by 1m

4.4 Conclusion

Unlike the traditional systems, clean air shelter demands only water and electricity. The initially proposed prototype distributed the purified air upwards, as shown in the simulation. Considering economics, we suggested a specially crafted cover to the outlet, based on the simulation result. Model 4, acquired by attaching the four-piece cover to the original prototype, showed the least weighted PM density when it was lifted by 1m among other models and heights. It provides a clean area of radius 3.7m in which the PM density is under $50\mu g/m^3$ when that of the fine dust is $75\mu g/m^3$.

Chapter 5. Summary

We've explored the diverse aspects of the real computation throughout three topics. In Chapter 2, compact subsets in Euclidean spaces were managed by introducing two new abstract data types in `iRRAM` — an Exact Real Computation package. Chapter 3 analyzed and exhibited that the current real computation in `iRRAM` can be done more efficiently. Chapter 4 went over an application of real computation in practice, where the exit of purifier was designed by a simulation.

Chapter 2 was presented at KSC2020 with the title *Compact Subsets in Exact Real Computation*. Chapter 2 was presented at CCA2020 and KMS2020 with the titles *Happy Birthday, iRRAM! Considerations for the Future of Exact Real Computation* and *Faster multi-precision computation by hybridization between hardware and software*, respectively.

Chapter 6. Appendix

Table 6.1: Matrix multiplication elapsed time ratio. DD or QD is faster than MPFR if the ratio is greater than 1.

mat.size	200	500	1000	2000	5000
MPFR96/DD	9.237	11.485	9.694	7.437	8.976
MPFR100/DD	11.224	12.161	10.325	8.180	9.070
MPFR104/DD	7.895	10.584	8.518	6.726	9.068
MPFR106/DD	8.197	10.466	8.932	7.198	9.712
MPFR112/DD	5.763	9.583	7.966	6.302	7.803
MPFR128/DD	8.303	10.900	9.506	7.502	9.242
MPFR192/QD	1.027	1.155	1.270	1.286	1.624
MPFR200/QD	0.791	1.202	1.407	1.443	1.826
MPFR208/QD	1.010	1.053	1.230	1.236	1.757
MPFR212/QD	0.835	1.136	1.335	1.388	1.796
MPFR224/QD	1.016	1.047	1.229	1.242	1.852

Table 6.2: Polynomial multiplication elapsed time ratio. DD or QD is faster than MPFR if the ratio is greater than 1.

poly.order	1000	2000	5000	10000	20000
MPFR96/DD	10.000	5.923	6.425	6.486	6.619
MPFR100/DD	12.000	6.692	7.411	7.572	7.791
MPFR104/DD	9.500	4.846	5.356	5.441	5.576
MPFR106/DD	6.500	4.923	5.493	5.710	5.909
MPFR112/DD	6.500	4.077	4.562	4.569	4.678
MPFR128/DD	10.500	6.308	6.808	6.917	7.116
MPFR192/QD	0.750	0.722	0.726	0.711	0.715
MPFR200/QD	0.944	0.874	0.899	0.909	0.906
MPFR208/QD	0.750	0.709	0.718	0.724	0.718
MPFR212/QD	0.861	0.795	0.792	0.790	0.798
MPFR224/QD	0.778	0.722	0.732	0.725	0.722

Table 6.3: PM density sensor readings(μg/m^3). Smaller is better.

(x,y)	Model 1	Model 2 lifted by			Model 3 lifted by			Model 4 lifted by		
	0m	0m	1m	2m	0m	1m	2m	0m	1m	2m
(0.5,1.25)	74.943	74.992	74.985	74.995	75.000	55.743	66.912	75.000	49.959	53.991
(0.5,1.45)	74.821	74.996	73.658	74.994	75.000	61.700	68.844	74.999	46.126	54.276
(0.5,1.65)	74.949	74.999	50.757	74.982	75.000	74.743	72.158	74.942	26.837	54.376
(0.5,1.85)	75.000	74.998	56.532	74.934	75.000	74.906	74.167	74.913	73.053	54.565
(0.5,2.05)	72.801	74.998	62.727	74.935	75.000	75.000	75.000	74.865	74.535	54.716
(1.0,1.25)	74.999	73.995	72.879	74.953	74.856	67.373	74.925	71.031	46.859	54.017
(1.0,1.45)	74.998	74.826	70.431	74.933	74.958	71.517	74.844	72.849	42.972	54.159
(1.0,1.65)	75.000	74.978	62.116	74.904	74.995	74.042	74.981	74.881	47.191	54.040
(1.0,1.85)	74.993	74.996	54.256	74.978	74.997	74.782	74.970	75.000	59.157	54.127
(1.0,2.05)	75.000	74.998	58.605	75.000	74.993	74.933	75.000	75.000	64.697	52.117
(1.5,1.25)	74.998	74.381	70.130	74.962	74.643	71.916	74.979	52.187	47.232	53.844
(1.5,1.45)	74.994	74.626	68.768	74.939	74.793	74.252	74.992	54.183	47.312	53.901
(1.5,1.65)	74.990	74.782	65.567	74.977	74.889	74.484	74.977	61.386	51.956	53.590
(1.5,1.85)	74.994	74.907	61.942	75.000	74.969	74.836	74.984	67.686	56.808	53.231
(1.5,2.05)	74.998	74.960	60.410	75.000	74.971	74.880	74.995	71.903	59.339	51.903
(2.0,1.25)	74.999	74.778	69.528	74.948	74.631	73.769	75.000	40.646	48.717	53.243
(2.0,1.45)	74.998	74.851	69.189	74.890	74.591	74.253	75.000	43.111	49.950	53.032
(2.0,1.65)	74.995	74.933	66.336	74.845	74.704	74.949	74.992	46.976	53.682	52.763
(2.0,1.85)	74.990	74.947	64.651	74.528	74.859	74.941	74.992	50.601	54.140	52.621
(2.0,2.05)	74.988	74.911	62.966	74.211	74.979	74.932	74.992	55.409	54.598	52.479
(2.5,1.25)	75.000	74.857	68.441	74.926	74.677	74.003	74.995	36.918	50.942	52.671
(2.5,1.45)	75.000	74.957	69.192	74.892	74.548	74.720	75.000	37.099	51.831	52.463
(2.5,1.65)	74.999	74.993	68.248	74.831	74.536	74.960	75.000	36.956	52.184	52.586
(2.5,1.85)	74.999	74.978	66.597	74.551	74.676	74.996	75.000	39.993	52.648	52.559
(2.5,2.05)	74.999	74.960	64.904	74.238	74.816	74.986	75.000	43.536	53.119	52.438
(3.0,1.25)	75.000	74.718	68.460	74.963	75.000	72.711	74.982	36.664	51.479	52.521
(3.0,1.45)	74.999	74.992	68.778	74.947	74.928	74.700	75.000	36.761	52.326	52.342
(3.0,1.65)	74.999	74.991	67.835	74.886	74.872	74.806	75.000	36.647	52.680	52.532
(3.0,1.85)	74.999	74.981	66.848	74.824	74.841	74.881	75.000	36.525	53.020	52.740
(3.0,2.05)	75.000	74.971	65.861	74.762	74.811	74.955	75.000	36.403	53.360	52.949
(3.5,1.25)	75.000	74.659	67.402	74.955	75.000	71.835	74.970	36.668	51.788	52.782
(3.5,1.45)	75.000	74.992	67.378	74.944	75.000	74.025	74.991	36.695	52.372	52.615
(3.5,1.65)	75.000	75.000	67.147	74.959	75.000	74.421	74.995	36.451	52.535	52.798
(3.5,1.85)	75.000	75.000	66.890	74.976	75.000	74.743	74.999	36.207	52.687	53.008
(3.5,2.05)	75.000	74.992	66.067	74.932	74.962	74.873	75.000	36.058	52.984	53.216
(4.0,1.25)	75.000	74.964	67.853	74.957	74.994	72.511	74.984	36.663	51.977	52.884
(4.0,1.45)	75.000	74.998	66.853	74.941	74.995	73.820	74.986	36.658	52.453	52.886
(4.0,1.65)	75.000	75.000	66.145	74.953	75.000	73.913	74.988	36.514	52.377	53.228
(4.0,1.85)	75.000	75.000	65.954	74.970	75.000	74.239	74.992	36.272	52.529	53.450
(4.0,2.05)	75.000	75.000	65.763	74.987	75.000	74.565	74.996	36.031	52.681	53.672

Table 6.4: PM density subtotal and total score(μg/m^3). Smaller is better.

weighted sum	Model 1	Model 2 lifted by			Model 3 lifted by			Model 4 lifted by		
	0m	0m	1m	2m	0m	1m	2m	0m	1m	2m
for y=1.25	0.015	0.015	0.014	0.015	0.015	0.014	0.014	0.009	0.010	0.010
for y=1.45	20.090	20.071	18.564	20.079	20.056	19.392	19.884	13.141	13.241	14.257
for y=1.65	239.086	238.984	204.901	238.849	238.713	237.646	237.954	161.303	155.201	169.736
for y=1.85	24.547	24.540	20.607	24.497	24.521	24.479	24.511	17.069	18.576	17.441
for y=2.05	0.022	0.022	0.018	0.022	0.022	0.022	0.022	0.016	0.017	0.015
Total score	7.094	7.091	6.103	7.087	7.083	7.039	7.060	4.788	4.676	5.037

Bibliography

[1] Fousse, Laurent, et al. "MPFR: A multiple-precision binary floating-point library with correct rounding." ACM Transactions on Mathematical Software (TOMS) 33.2 (2007): 13-es.

[2] Müller, Norbert Th. "The iRRAM: Exact arithmetic in C++." International Workshop on Computability and Complexity in Analysis. Springer, Berlin, Heidelberg, 2000.

[3] Lambov, Branimir. "RealLib: An efficient implementation of exact real arithmetic." Mathematical Structures in Computer Science 17.1 (2007): 81-98.

[4] Yu, Jihun, et al. "The design of Core 2: A library for exact numeric computation in geometry and algebra." International Congress on Mathematical Software. Springer, Berlin, Heidelberg, 2010.

[5] Weihrauch, Klaus. Computable analysis: an introduction. Springer Science & Business Media, 2000.

[6] M. Braverman, Computational Complexity of Euclidean Sets: Hyperbolic Julia Sets are Poly-Time Computable, Proc. CCA 2004., Electr. Notes Theor. Comput. Sci. 120: 17-30 (2005)

[7] Lions, Jacques-Louis. "Flight 501 failure." Report by the Inquiry Board 190 (1996).

[8] Bailey, David H., and Jonathan M. Borwein. "High-precision arithmetic in mathematical physics." Mathematics 3.2 (2015): 337-367.

[9] Pathria, R. K. "A statistical study of randomness among the first 10,000 digits of π." Mathematics of Computation 16.78 (1962): 188-197.

[10] Chu, Eleanor, and Alan George. "Gaussian elimination with partial pivoting and load balancing on a multiprocessor." Parallel Computing 5.1-2 (1987): 65-74.

[11] Stevenson, David. "IEEE 754-1985 IEEE Standard for Binary Floating-Point Arithmetic." 20pp, IEEE, July (1985).

[12] Schwarz, Eric M., and Christopher A. Krygowski. "The S/390 G5 floating-point unit." IBM Journal of Research and Development 43.5.6 (1999): 707-721.

[13] Gerwig, Guenter, et al. "The IBM eServer z990 floating-point unit." IBM journal of Research and Development 48.3.4 (2004): 311-322.

[14] Sadasivam, Satish Kumar, et al. "IBM Power9 processor architecture." IEEE Micro 37.2 (2017): 40-51.

[15] Waterman, Andrew, et al. "The risc-v instruction set manual, volume i: Base user-level isa." EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62 116 (2011).

[16] Russinoff, David M. "x87 Instructions." Formal Verification of Floating-Point Hardware Design. Springer, Cham, 2019. 227-231.

[17] May, Robert M. "Simple mathematical models with very complicated dynamics." The Theory of Chaotic Attractors (2004): 85-93.

[18] Galias, Zbigniew. "The dangers of rounding errors for simulations and analysis of nonlinear circuits and systems? and how to avoid them." IEEE Circuits and Systems Magazine 13.3 (2013): 35-52.

[19] Hu, Tianli, and Shijun Liao. "On the risks of using double precision in numerical simulations of spatio-temporal chaos." Journal of Computational Physics 418 (2020): 109629.

[20] Abadi, Martín, et al. "Tensorflow: A system for large-scale machine learning." 12th USENIX symposium on operating systems design and implementation (OSDI 16). 2016.

[21] Paszke, Adam, et al. "Pytorch: An imperative style, high-performance deep learning library." arXiv preprint arXiv:1912.01703 (2019).

[22] Karatsuba, Anatolii Alekseevich, and Yu P. Ofman. "Multiplication of many-digital numbers by automatic computers." Doklady Akademii Nauk. Vol. 145. No. 2. Russian Academy of Sciences, 1962.

[23] Karatsuba, Anatolii Alexeevich. "The complexity of computations." Proceedings of the Steklov Institute of Mathematics-Interperiodica Translation 211 (1995): 169-183.

[24] Cook, Stephen A., and Stål O. Aanderaa. "On the minimum computation time of functions." Transactions of the American Mathematical Society 142 (1969): 291-314.

[25] Schönhage, Arnold, and Volker Strassen. "Rapid multiplication of large numbers." Computing 7.3 (1971): 281-292.

[26] Bailey, David H., Roberto Barrio, and Jonathan M. Borwein. "High-precision computation: Mathematical physics and dynamics." Applied Mathematics and Computation 218.20 (2012): 10106-10121.

[27] Hida, Yozo, Xiaoye S. Li, and David H. Bailey. "Library for double-double and quad-double arithmetic." NERSC Division, Lawrence Berkeley National Laboratory (2007): 19.

[28] Dekker, Theodorus Jozef. "A floating-point technique for extending the available precision." Numerische Mathematik 18.3 (1971): 224-242.

[29] Priest, Douglas M. On properties of floating point arithmetics: numerical stability and the cost of accurate computations. Diss. University of California, Berkeley, 1992.

[30] Shewchuk, Jonathan Richard. "Adaptive precision floating-point arithmetic and fast robust geometric predicates." Discrete & Computational Geometry 18.3 (1997): 305-363.

[31] Kunzmann, Klaus R., and Michael Wegener. "The pattern of urbanization in Western Europe." Ekistics (1991): 282-291.

[32] Ritchie, Hannah, and Max Roser. "Urbanization." Our world in data (2018).

[33] Scott, Allen J. "Industrialization and urbanization: a geographical agenda." Annals of the Association of American Geographers 76.1 (1986): 25-37.

[34] Brunekreef, Bert, and Stephen T. Holgate. "Air pollution and health." The lancet 360.9341 (2002): 1233-1242.

[35] Cohen, Aaron J., et al. "Estimates and 25-year trends of the global burden of disease attributable to ambient air pollution: an analysis of data from the Global Burden of Diseases Study 2015." The Lancet 389.10082 (2017): 1907-1918.

[36] Payet, S., et al. "Penetration and pressure drop of a HEPA filter during loading with submicron liquid particles." Journal of Aerosol Science 23.7 (1992): 723-735.

[37] Rooney, Charles. "Economics of Pollution Prevention= How Wmte Reduction Pays." Pollution Prevention Review 261 (1993).

[38] Kim, H. T., et al. "Particle removal efficiency of gravitational wet scrubber considering diffusion, interception, and impaction." Environmental engineering science 18.2 (2001): 125-136.

[39] World Health Organization (WHO). "WHO Air Quality Guidelines for Particulate Matter." Ozone, Nitrogen Dioxide and Sulfur Dioxid (2005).

Acknowledgment

Finishing up one of my life stages as a student, I cannot help but show my gratitude to Prof. Dr. Martin Ziegler, who hired and has guided me as a mentor. I've learned altruism as well as profound knowledge from him. In addition, Dr. Svetlana Selivanova has taught me thoroughly and kindly, spending her precious time on revising this thesis and presentation slides. I leave a big thanks to her on this small paragraph.

Furthermore, I would like to mention the lab members. Dr. Alexander Stoimenov gave great talks on the computational knot theory, which inspired me. Dr. Sewon Park gently introduced the compact subset topic, developed and archived in Chapter 2. Donghyun Lim and Hyunwoo Lee also shed light on whatever I was struggling with. Jihoon Hyun reminded me that I had a passion like his.

마지막으로 저의 여정을 도와주신 가족분들께 고마움을 표합니다. 30년이 넘도록 보답이라곤 제대로 해본 적이 없는 저에게, 그래도 자식이라고 무한한 사랑을 베풀어 주신 부모님 고맙습니다. 또 언제나 웃으며 반겨주시는 큰이모네 가족분들과 막내이모도 모두 고맙습니다. 베푸신 은혜에 얼른 보답하도록 노력하겠습니다. 서현아 승주야 너네는 커서 잘 살았으면 좋겠구나.

Curriculum Vitae

Name : Jiman Hwang

Date of Birth : July 1, 1990

Educations

2009. 2. – 2017. 2. 성균관대 기계공학부, 컴퓨터공학 (학사)

2019. 9. – 2021. 8. 한국과학기술원 전산학부 (석사)

Career

1990. 7. – 2021. 6. 경력도 없고 망했네

Academic Activities

1. **Jiman Hwang**, Svetlana Selivanova, Martin Ziegler *Happy Birthday, iRRAM! Considerations for the Future of Exact Real Computation*, CCA 2020 (Sep.10, virtual, originally planned for Bologna, Italy)
2. **Jiman Hwang**, Svetlana Selivanova, Martin Ziegler *Faster Multi-Precision Computation by Hybridization between Hardware and Software*, 2020 KMS Annual Meeting) (Oct.23, virtual)

Publications

1. **Jiman Hwang**, and Sewon Park. *Compact Subsets in Exact Real Computation.* KSC2020: 1104-1106.