

**NHẬN XÉT CỦA NGƯỜI HƯỚNG DẪN**

**NHẬN XÉT CỦA NGƯỜI PHẢN BIỆN**

## TÓM TẮT

Tên đề tài: Hệ thống truyền thông một chiều với độ lỗi và độ trễ thấp.

Sinh viên thực hiện: Nguyễn Vũ Khánh Uy

Số thẻ SV: 102210240      Lớp: 21TCLC\_DT3

Đề tài tập trung vào việc thiết kế hệ thống truyền thông một chiều (UDP) với độ lỗi thấp bằng cách sử dụng kỹ thuật sửa lỗi chuyển tiếp (FEC) trên kênh xóa (erasure channel), cụ thể là mã sửa lỗi Reed-Solomon trên trường GF(65537). Để đạt được tốc độ mã hóa, giải mã nhanh, hệ thống áp dụng thuật toán mã hóa giải mã với độ phức tạp thấp  $O(N \log N)$  và các kỹ thuật xử lý song song (SIMD, SPMD) trên CPU và GPU, cụ thể là trình biên dịch ISPC cho CPU Intel hỗ trợ AVX2-i32x8 và framework CUDA cho GPU Nvidia kiến trúc Ampere. Phiên bản CPU hướng tới doanh nghiệp có nhu cầu gửi một chiều nhiều dữ liệu nhỏ với độ trễ thấp, tốc độ mã hóa 405 MB/s, giải mã 22 MB/s. Phiên bản GPU hướng tới doanh nghiệp có nhu cầu gửi dữ liệu một chiều với thông lượng cao tốc độ mã hóa 1.1 GB/s, giải mã 61 MB/s. Với mạng có tỷ lệ mất gói cao là 20% (tính cả gói lỗi bị xóa sau khi checksum) và thông lượng lớn (đủ lớn để tỷ lệ lỗi, mất gói trở thành ngẫu nhiên), hệ thống đảm bảo tỷ lệ mất gói khi truyền 1 TB dữ liệu là nhỏ hơn  $10^{-6}$  với thông lượng truyền tối đa bằng một nửa thông lượng của mạng. Đề tài kỳ vọng sẽ giải quyết được các bài toán đặc thù về truyền thông một chiều của doanh nghiệp như truyền thông một chiều từ mạng SCADA ra mạng OFFICE của tập đoàn điện lực Việt Nam – EVN, cùng với đó là đánh giá được khả năng xử lý song song của phần cứng hiện đại.

## NHIỆM VỤ ĐỒ ÁN TỐT NGHIỆP

Họ tên sinh viên: Nguyễn Vũ Khánh Uy

Số thẻ sinh viên: 102210240

Lớp: 21TCLC\_DT3    Khoa: Công Nghệ Thông Tin    Ngành: An toàn thông tin

1. Tên đề tài đồ án:

Hệ thống truyền thông một chiều với độ lỗi và độ trễ thấp.

2. Đề tài thuộc diện: ☐ Có ký kết thỏa thuận sở hữu trí tuệ đối với kết quả thực hiện

3. Các số liệu và dữ liệu ban đầu: Không.

4. Nội dung các phần thuyết minh và tính toán:

- Mở đầu: Giới thiệu tổng quan, mục đích, phạm vi, hướng nghiên cứu và bố cục đề tài.
- Chương 1: Cơ sở lý thuyết – Trình bày cơ bản thuật toán sửa lỗi, thuật toán song song và kiến trúc CPU, GPU.
- Chương 2: Thiết kế và tối ưu hệ thống – Trình bày chi tiết về cách thiết kế và tối ưu thuật toán trong hệ thống.
- Chương 3: Kết quả và kiểm thử.
- Kết luận: So sánh và đánh giá các kết quả đạt được, chỉ ra những hạn chế còn tồn tại và đề xuất hướng phát triển.

5. Các bản vẽ, đồ thị ( ghi rõ các loại và kích thước bản vẽ ): Không.

6. Họ tên người hướng dẫn: ThS. Nguyễn Thê Xuân Ly

7. Ngày giao nhiệm vụ đồ án: 12/04/2025

8. Ngày hoàn thành đồ án: 19/06/2025

Đà Nẵng, ngày 19 tháng 6 năm 2025

Trưởng Bộ môn .....

Người hướng dẫn

## LỜI NÓI ĐẦU

Trong quá trình học tập và nghiên cứu tại Trường Đại học Bách Khoa – Đại học Đà Nẵng, em đã được trang bị những kiến thức chuyên môn vững chắc, cũng như có cơ hội rèn luyện kỹ năng tư duy, phân tích và giải quyết vấn đề. Đặc biệt, đồ án tốt nghiệp là cơ hội quý báu để em vận dụng những kiến thức đã học vào thực tế, từ đó nâng cao năng lực chuyên môn và tích lũy kinh nghiệm thực tiễn.

Em xin chân thành gửi lời cảm ơn sâu sắc đến quý thầy cô Trường Đại học Bách Khoa – Đại học Đà Nẵng, những người đã tận tình giảng dạy, truyền đạt kiến thức và đồng hành cùng em suốt chặng đường đại học.

Em đặc biệt biết ơn thầy Nguyễn Thế Xuân Ly, người đã trực tiếp hướng dẫn, hỗ trợ em trong suốt quá trình thực hiện đồ án. Những góp ý và chỉ dẫn quý báu của thầy đã giúp em định hướng rõ ràng hơn và hoàn thiện tốt đề tài này.

Em cũng xin gửi lời cảm ơn chân thành đến gia đình và bạn bè – những người luôn bên cạnh, động viên và tạo điều kiện thuận lợi nhất cho em trong học tập và cuộc sống.

## **CAM ĐOAN**

Tôi xin cam đoan đề tài “Hệ thống truyền thông một chiều với độ lỗi và độ trễ thấp” này do chính tôi thực hiện, dưới sự hướng dẫn trực tiếp của thầy Nguyễn Thế Xuân Ly, không sao chép bất kỳ nguồn nào khác và chưa từng được công khai từ trước. Các tham khảo dùng trong đồ án đều được trích dẫn rõ ràng tên tác giả, tên công trình, thời gian, địa điểm công bố. Nếu có những sao chép không hợp lệ, vi phạm, tôi xin chịu hoàn toàn trách nhiệm.

Sinh viên thực hiện

## MỤC LỤC

TÓM TẮT .....	
LỜI NÓI ĐẦU .....	i
CAM ĐOAN .....	ii
MỤC LỤC .....	iii
DANH SÁCH CÁC BẢNG, HÌNH VẼ .....	v
DANH SÁCH CÁC KÝ HIỆU, CHỮ VIẾT TẮT .....	vii
MỞ ĐẦU .....	1
CHƯƠNG 1: CƠ SỞ LÝ THUYẾT .....	3
1.1. Truyền thông một chiều .....	3
1.1.1. Tổng quan bài toán truyền thông một chiều của tập đoàn EVN .....	3
1.1.2. Mô hình chi tiết về hệ thống truyền thông của tập đoàn EVN .....	4
1.1.3. Tổng quan về mã sửa lỗi, hướng giải quyết cho bài toán truyền thông một chiều tin cậy .....	5
1.2. Mã sửa lỗi Reed-Solomon (Erasure code) .....	6
1.2.1. Tổng quan về mã sửa lỗi Reed-Solomon (Erasure code) .....	6
1.2.2. Trường $GF(65537)$ và biến đổi số Fermat (Fermat Number Transform) .....	8
1.2.3. Mã sửa lỗi Reed-Solomon (Erasure code) trên trường $GF(65537)$ .....	9
1.3. Thiết kế thuật toán song song .....	12
1.3.1. Tổng quan về thiết kế thuật toán song song .....	12
1.3.2. Thuật toán biến đổi số Fermat song song .....	13
1.3.3. Mã sửa lỗi Reed-Solomon (Erasure code) song song .....	15
1.4. Công nghệ và kiến trúc phần cứng sử dụng .....	16
1.4.1. Công nghệ sử dụng .....	16
1.4.2. Kiến trúc cơ bản và lập trình song song trên CPU .....	16

1.4.3. Kiến trúc cơ bản và lập trình song song trên GPU .....	18
<b>CHƯƠNG 2: THIẾT KẾ VÀ TỐI ƯU HỆ THỐNG .....</b>	<b>21</b>
2.1. Thiết kế hệ thống .....	21
2.1.1. Thiết kế cấu trúc gói tin và chi tiết hơn về mã hóa giải mã .....	21
2.1.2. Xây dựng và tối ưu xử lý mã hóa giải mã trên CPU .....	22
2.1.3. Xây dựng và tối ưu xử lý mã hóa giải mã trên GPU .....	23
2.1.4. Xây dựng ứng dụng.....	27
<b>CHƯƠNG 3: KẾT QUẢ VÀ KIỂM THỬ .....</b>	<b>29</b>
<b>KẾT LUẬN .....</b>	<b>38</b>
<b>TÀI LIỆU THAM KHẢO.....</b>	



## DANH SÁCH CÁC BẢNG, HÌNH VẼ

Hình 1.1 Mô hình hệ thống truyền thông tập đoàn EVN .....	4
Hình 1.2 Phần cứng ETAP-2003 .....	4
Hình 1.3 Biểu diễn hình học của mã lặp lại .....	5
Hình 1.4 Cơ bản về mã sửa lỗi Reed-Solomon (erasure code) .....	7
Hình 1.5 Ví dụ về 8 điểm dừng trong FFT so với FNT .....	9
Hình 1.6 Thuật toán cây tích con (Subproduct Tree) .....	11
Hình 1.7 Quá trình sinh ra 8 điểm trong thuật toán FNT qua các tầng (mô hình sinh nhóm tuần hoàn) .....	14
Hình 1.8 Quá trình sinh ra 8 điểm trong thuật toán FNT qua các tầng (mô hình công việc trên mảng) .....	14
Hình 1.9 Công nghệ sử dụng .....	16
Hình 1.10 Kiến trúc cơ bản một nhân CPU (Intel Skylake), nguồn [2] .....	17
Hình 1.11 Kiến trúc cơ bản của một SM trong GPU V100, nguồn [2] .....	18
Hình 1.12 Mô hình xử lý cơ bản trong kiến trúc Cuda .....	20
Hình 2.1 Cách truy cập bộ nhớ tối ưu, đảm bảo các luồng truy cập bộ nhớ cạnh nhau ở mỗi bước .....	24
Hình 2.2 Tỷ lệ cache hit L1, L2 sau khi truy cập bộ nhớ theo cách trên, đo bằng công cụ Nsight Compute .....	25
Hình 2.3 Tỷ lệ thời gian sao chép dữ liệu giữa CPU và GPU so với cả hệ thống, đo bằng công cụ Nsight System .....	26
Hình 2.4 Thời gian chạy của một hàm FNT chỉ còn 40 micro giây, đo bằng công cụ Nsight System .....	26
Hình 2.5 Tình trạng các kernel, tỉ lệ lấp đầy Warp, tỉ lệ GPU active mỗi chu kì, tỉ lệ câu lệnh của SM mỗi chu kì, đo bằng Nsight System .....	27
Hình 3.1 Giao diện console cơ bản của hệ thống .....	29
Hình 3.2 Lệnh tăng kích thước bộ nhớ đệm để lưu các gói tin đến và đi .....	30
Hình 3.3 Mô hình mạng cơ bản trên GNS3 .....	30
Hình 3.4 Thử gửi một lần 10 MB/s dùng công cụ iperf3 .....	31
Hình 3.5 Thử gửi 100 lần thông lượng 1 MB/s, phần đầu, dùng công cụ iperf3 .....	31
Hình 3.6 Thử gửi 100 lần thông lượng 1 MB/s, phần sau, dùng công cụ iperf3 .....	31

Hình 3.7 Thử gửi 30 lần thông lượng 2 MB/s dùng công cụ iperf3.....	32
Hình 3.8 Thử gửi 30 lần thông lượng 2 MB/s phần bên người gửi.....	33
Hình 3.9 Chỉnh tỉ lệ mất gói mạng thành 20% trong GNS3 .....	34
Hình 3.10 Chỉnh tỉ lệ lỗi gói mạng thành 5% trong GNS3 .....	34
Hình 3.11 Gửi file 500 KB 30 lần bằng hệ thống .....	34
Hình 3.12 Nhận file 500 KB 30 lần bằng hệ thống.....	35
Hình 3.13 Đo lại tỉ lệ lỗi mạng sau khi đã chỉnh GNS3 dùng iperf3 .....	36

## DANH SÁCH CÁC KÝ HIỆU, CHỮ VIẾT TẮT

Từ	Viết tắt của
CPU	Central Processing Unit
GPU	Graphics Processing Unit
UDP	User Datagram Protocol
SIMD	Single instruction, multiple data
SPMD	Single Program, Multiple Data
FFT	Fast Fourier transform
FNT	Fermat Number Transform
GF	Galois Field
ECC	Error-Correcting Code
FEC	Forward Error Correction
TBB	Thread Building Block (Intel OneAPI)
JNI	Java Native Interface
JVM	Java Virtual Machine
CAS	Compare And Swap

## MỞ ĐẦU

### 1. Mục đích thực hiện đề tài

Các hệ thống truyền thông hiện nay xây dựng dựa trên các giao thức kết nối hai chiều (TCP, rUDP) đã chứng minh độ hiệu quả cao và độ lỗi thấp nhờ vào việc phát hiện lỗi bằng checksum và gửi lại gói tin lỗi.

Tuy nhiên có một số bài toán đặc thù yêu cầu khả năng giao tiếp một chiều như:

- Trích xuất dữ liệu từ hệ thống điều khiển mạng lưới điện (SCADA) ra hệ thống mạng văn phòng (OFFICE) của tập đoàn EVN: Hệ thống điều khiển trung tâm trong mạng SCADA của EVN chỉ huy vận hành hơn 200 trạm biến áp và 180 nhà máy điện (chỉ tính trong khu vực miền Trung – Tây Nguyên). Hệ thống SCADA là hệ thống trọng yếu ảnh hưởng đến an ninh năng lượng Việt Nam. Mạng SCADA được thiết kế cô lập với môi trường mạng bên ngoài, dữ liệu trích xuất ra từ SCADA sẽ được đi qua hệ thống DATA DIODE giúp chặn một chiều, đảm bảo SCADA an toàn trước các cuộc tấn công từ bên ngoài. Tuy nhiên truyền thông một chiều thì tỷ lệ lỗi, mất gói cao. Việc xây dựng hệ thống truyền thông một chiều từ mạng SCADA ra mạng OFFICE với độ lỗi thấp là vấn đề cấp thiết.
- Các ứng dụng truyền thông đa phương tiện, thời gian thực cần độ trễ thấp và tính ổn định cao: Các ứng dụng gọi video (Zalo), game online (AOE4) cần độ trễ thấp, đôi khi có thể chấp nhận mất khung hình đã quá trễ để người dùng có được cập nhật mới nhất, đảm bảo trải nghiệm mượt mà. Việc dùng các giao thức truyền thông như TCP có thể ảnh hưởng đến độ trễ do quá trình đợi SYN-ACK hay việc truyền lại gói tin khi bị mất. Vì vậy việc xây dựng hệ thống truyền thông một chiều với độ lỗi và độ trễ thấp là cần thiết.

Việc dùng các kỹ thuật sửa lỗi chuyển tiếp (Forward Error Correction), mã sửa lỗi (Error-Correcting Code) để giảm thiểu lỗi khi truyền thông một chiều đã được nghiên cứu mạnh mẽ trong 50 năm qua. Vấn đề chính của các kỹ thuật này là khả năng sửa lỗi càng mạnh thì độ phức tạp thuật toán càng cao, hạn chế tính khả thi trong ứng dụng. Tuy nhiên những năm gần đây, việc phát triển AI đã thúc đẩy sự phát triển mạnh mẽ của các kỹ thuật tính toán song song (SIMD, SPMD) trên CPU và GPU. Với khả năng tính toán của phần cứng hiện tại, việc xây dựng hệ thống truyền thông một chiều với độ lỗi và độ trễ thấp là khả thi.

## 2. Mục tiêu đề tài

- Xây dựng thành công hệ thống truyền thông một chiều với độ lỗi và độ trễ thấp.
- Đánh giá được khả năng xử lý song song của phần cứng hiện đại.

## 3. Đối tượng và phạm vi đề tài

Các công ty hoặc người dùng có nhu cầu sử dụng hệ thống truyền thông 1 chiều, có khả năng sở hữu CPU Intel hỗ trợ các câu lệnh AVX2-32ix8 (Core I5 12400) hoặc GPU Nvidia kiến trúc Ampere (RTX 3060).

## 4. Phương pháp thực hiện

- Đánh giá khả năng chịu lỗi: Mô phỏng mạng GNS3 và cài đặt hệ thống trên 2 máy ảo, chỉnh các thiết lập độ lỗi mạng và thử với các file kích thước khác nhau, thu thập số liệu và tối ưu chương trình.
- Đánh giá độ trễ: Cài đặt và chạy chương trình trên CPU Intel, GPU Nvidia, đo thời gian xử lý và tính thông lượng.

## 5. Cấu trúc đồ án tốt nghiệp

- Các bài toán thực tế và ứng dụng của đề tài.
- Ý tưởng cơ bản mã sửa lỗi Reed-Solomon.
- Cài đặt Reed-Solomon (erasure code) với độ phức tạp thấp  $O(N \log N)$  bằng Fermat Number Transform.
- Kiến trúc CPU Intel hỗ trợ AVX2-32ix8 và GPU Nvidia Ampere.
- Tối ưu thuật toán trên CPU Intel (Core I5 12400) và GPU Nvidia Ampere (RTX 3060).
- Thiết kế cấu trúc gói tin.
- Thiết kế hệ thống (console app).
- Thử nghiệm hệ thống khi mô phỏng trên GNS3.

## CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

### 1.1. Truyền thông một chiều

#### 1.1.1. Tổng quan bài toán truyền thông một chiều của tập đoàn EVN

Trung tâm Điều độ hệ thống điện miền Trung là đơn vị thuộc Tập đoàn điện lực Việt Nam, có nhiệm vụ chỉ huy, vận hành lưới điện 220kV, 110kV và các NMD ở khu vực 13 miền Trung – Tây Nguyên. Để phục vụ vận hành hệ thống điện, Trung tâm được Tập đoàn điện lực Việt Nam trang bị hệ thống SCADA có nhiệm vụ giám sát, điều khiển và thu thập dữ liệu của các Trạm biến áp, nhà máy điện trong khu vực. Ước tính đến thời điểm hiện tại, hệ thống SCADA tại Trung tâm kết nối với khoảng 200 trạm biến áp, 180 nhà máy điện trong phạm vi 13 tỉnh miền Trung - Tây nguyên.

Hệ thống SCADA tại Trung tâm Điều độ hệ thống điện miền Trung được xếp loại là hệ thống thông tin trọng yếu do có thể ảnh hưởng đến an ninh năng lượng của lưới điện miền Trung nói riêng và lưới điện Việt Nam nói chung. Vì vậy, việc đảm bảo an toàn thông tin cho hệ thống này là cực kỳ quan trọng và được đặt lên hàng đầu.

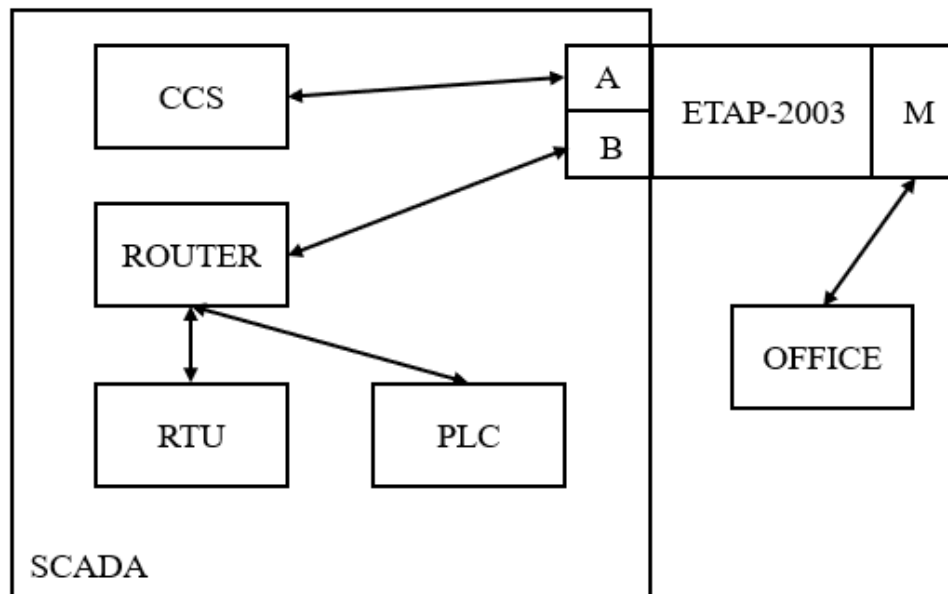
Hệ thống mạng điều khiển công nghiệp (ICS – Industrial Control System) có tầm quan trọng rất lớn, phục vụ việc vận hành lưới điện an toàn, tin cậy. Bên cạnh việc thu thập số liệu để phục vụ vận hành, hệ thống cần hỗ trợ truy xuất dữ liệu từ hệ thống ICS ra mạng văn phòng để phục vụ báo cáo, tính toán, phân tích, lập kế hoạch. Trong trường hợp này, Việc trang bị tường lửa là cần thiết để tránh bị tấn công từ hệ thống mạng văn phòng vào hệ thống ICS. Tuy nhiên, các thiết bị này vẫn tiềm ẩn nguy cơ bị tấn công thông qua việc thiết lập các chính sách không chặt chẽ, cũng như tiềm ẩn các lỗ hổng bảo mật trong thiết bị tường lửa. Từ thực trạng nêu trên, Trung tâm đề xuất đề tài “Tìm hiểu hệ điều hành linux và xây dựng hệ thống truyền số liệu 1 chiều để truyền số liệu giữa mạng công nghiệp và mạng văn phòng”.

Dữ liệu sẽ được truyền theo 1 chiều về mặt vật lý, đảm bảo an toàn tuyệt đối khi truyền số liệu từ mạng công nghiệp ra mạng văn phòng.

Đầu vào: Các file dữ liệu, file ảnh, file video kích thước lớn từ hệ thống mạng công nghiệp. Đầu ra: Dữ liệu được xuất ra từ hệ thống mạng công nghiệp đảm bảo tính toàn vẹn, có tỉ lệ lỗi thấp, tốc độ truyền dữ liệu cao. Hệ thống truyền số liệu hoạt động ổn định trong thời gian dài.

### 1.1.2. Mô hình chi tiết về hệ thống truyền thông của tập đoàn EVN

Mạng công nghiệp SCADA và mạng văn phòng OFFICE là hai mạng biệt lập, được ngăn cách với nhau bởi thiết bị phân cứng ETAP-2003 giúp ngăn chặn sự xâm nhập của mạng bên ngoài và cho phép dữ liệu từ bên trong mạng công nghiệp đi ra để phân tích. Cụ thể, thiết bị phân cứng ETAP-2003 chỉ sao chép tất cả gói tin được gửi trong một đầu, giữa 2 cổng A và B, là mạng SCADA và gửi ra mạng đầu còn lại, cổng M, là mạng OFFICE.



Hình 1.1 Mô hình hệ thống truyền thông tập đoàn EVN



Hình 1.2 Phần cứng ETAP-2003

Trong mạng SCADA, giao tiếp thông qua giao thức UDP, UDP multicast giữa máy chủ điều khiển trung tâm, các máy điều khiển trạm biến áp và nhà máy điện.

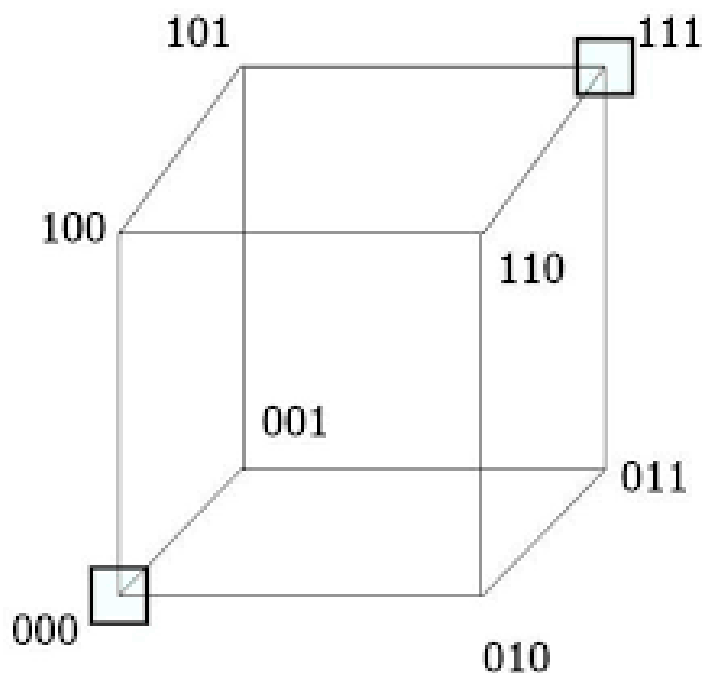
Mô hình truyền thông đặc thù một chiều từ mạng SCADA ra mạng OFFICE thông qua phần cứng ETAP-2003 dẫn đến khả năng mất, lỗi gói cao. Giả sử tỷ lệ mất gói chỉ 1% thì tỉ lệ gửi và nhận đủ cả 70 gói tin chỉ chưa tới 50%.

### 1.1.3. Tổng quan về mã sửa lỗi, hướng giải quyết cho bài toán truyền thông một chiều tin cậy

Khi hệ thống truyền thông với thông lượng phù hợp, tương đương thông lượng của mạng thì tỉ lệ lỗi, mất gói là ngẫu nhiên. Với điều kiện trên, việc sử dụng các mã sửa lỗi để đảm bảo truyền thông tin cậy là khả thi.

Ý tưởng chính của mã sửa lỗi (ECC) hay sửa lỗi chuyển tiếp (FEC) là sinh ra các bit dư thừa dựa vào một thuật toán chạy trên bit gốc, thuật toán được thống nhất giữa bên gửi và bên nhận, sau đó gửi cả bit gốc và bit thừa đi. Bên nhận khi nhận được dữ liệu sẽ cố suy lại bit gốc hoặc ít nhất là phát hiện lỗi dữ liệu với một tỷ lệ cao.

Ví dụ về mã sửa lỗi đơn giản là “Mã lặp lại”, người gửi sẽ gửi 1 bit 3 lần, người nhận sẽ dựa vào 2 bit giống nhau trong 3 bit để dự đoán bit gốc, tuy nhiên với mã đơn giản trên thì lượng bit dư thừa sẽ lớn, ảnh hưởng đến thông lượng tối đa, tỉ lệ dự đoán đúng bit gốc cũng không quá cao và không có khả năng chịu được lỗi mất bit.



Hình 1.3 Biểu diễn hình học của mã lặp lại



Việc nghiên cứu áp dụng các mã sửa lỗi để đảm bảo truyền thông tin cậy đã được nghiên cứu mạnh mẽ trong 50 năm gần đây, hiện nay có rất nhiều loại mã sửa lỗi, mỗi loại mã đều có điểm mạnh riêng về thông lượng, khả năng chống chịu lỗi, loại kênh truyền, loại lỗi.. Để chọn mã sửa lỗi phù hợp cho bài toán truyền thông, cần xem xét về tính chất của kênh truyền và tỉ lệ lỗi, loại lỗi, yêu cầu về chống chịu lỗi.

Bài toán của tập đoàn EVN yêu cầu truyền thông qua mạng riêng SCADA sử dụng UDP. Với mạng không dây bình thường, tỉ lệ lỗi bit trung bình dưới  $10^{-6}$  % và tỉ lệ mất gói dưới 5 % [11], với mạng riêng của tập đoàn EVN, có thể kì vọng tỉ lệ lỗi và mất gói thấp hơn, hoặc ít nhất là ổn định hơn. Nhận thấy tỉ lệ lỗi gói thấp hơn nhiều so với tỉ lệ mất gói, mỗi gói tin có thể sử dụng checksum để kiểm tra độ toàn vẹn của gói, nếu gói lỗi thì coi như gói đó mất, điều đó cũng không tăng tỉ lệ mất gói lên đáng kể. Với việc sử dụng checksum và xóa gói tin bị lỗi, kênh truyền bây giờ sẽ có tính chất kênh xóa (erasure channel). Với tính chất trên, có thể sử dụng các mã sửa lỗi thiết kế riêng cho kênh xóa để có khả năng chống chịu lỗi cao và thông lượng lớn. Vì hệ thống cần truyền thông qua giao thức UDP nên gói tin cần có một kích cỡ phù hợp, không thể quá nhỏ vì sẽ ảnh hưởng đến tốc độ thực tế và độ lỗi của kênh truyền, do router phải chạy thuật toán định tuyến cho nhiều gói tin nhỏ thay vì cho chung một gói tin lớn. Ngoài ra, các vị trí các gói bị mất thường hay ở cạnh nhau do lý do chính gây mất gói là sự tắc nghẽn ở các router, dạng lỗi này gọi là lỗi liên tiếp (burst error). Từ những yếu tố kể trên, mã sửa lỗi được chọn phải có độ dài mã lớn, có khả năng chống chịu lỗi liên tiếp tốt. Vì vậy, hệ thống được thiết kế sử dụng mã Reed-Solomon, phiên bản mã xóa (erasure code), trên trường  $GF(65537)$  cho phép sinh từ mã gốc dài tới 32768 kí hiệu (symbol), mỗi kí hiệu gồm 2 byte, mỗi gói tin có thể thiết kế độ lớn 1024 byte gồm 512 kí hiệu, hệ thống sẽ gửi thêm 50% kí hiệu dư thừa để đảm bảo gần như chắc chắn khôi phục lại được bộ kí hiệu gốc. Ngoài ra, mã Reed-Solomon (erasure code) còn là một trong những mã chống chịu lỗi liên tiếp mạnh nhất, với tỉ lệ kí hiệu khôi phục được so với kí hiệu dư thừa cao nhất, mã hiện tại có thuật toán mã hóa giải mã với độ phức tạp khá tốt  $O(N\log N)$ .

## 1.2. Mã sửa lỗi Reed-Solomon (Erasure code)

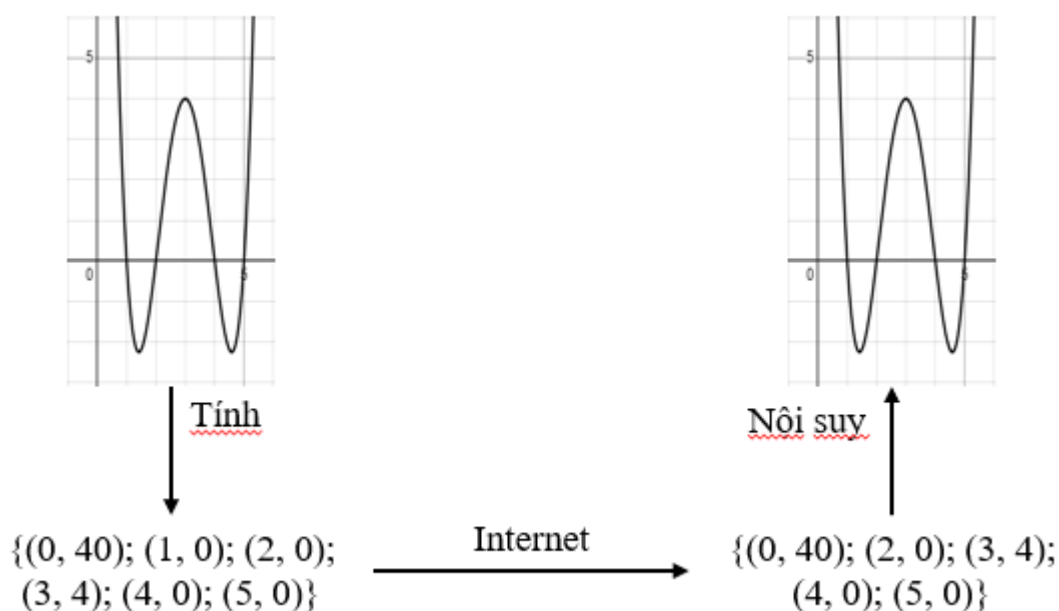
### 1.2.1. Tổng quan về mã sửa lỗi Reed-Solomon (Erasure code)

Mã Reed-Solomon, phiên bản mã xóa, xử lý trên một khối dữ liệu có độ dài cố định, mỗi khối chứa  $k$  kí hiệu, từ  $k$  kí hiệu ban đầu sẽ sinh ra  $n$  kí hiệu,  $t = n - k$  là số lượng kí hiệu dư thừa. Chỉ cần có  $k$  kí hiệu bất kì và vị trí của  $k$  kí hiệu đó trong  $n$  kí hiệu khi sinh thì có thể phục hồi  $t$  kí hiệu còn lại.

Mã xóa Reed-Solomon hoạt động dựa trên tính chất về nội suy đa thức. Cho  $n + 1$  điểm dữ liệu  $(x_i, y_i)$ ,  $0 \leq i \leq n$ , tồn tại một và chỉ một đa thức bậc cao nhất  $n$  nội suy các điểm dữ liệu. Đa thức  $p(x) = a_0 + \dots + a_n x^n$  nội suy  $n + 1$  điểm dữ liệu khi  $p(x_i) = y_i$  với mọi  $0 \leq i \leq n$ .

Mã xóa Reed-Solomon (không hệ thống) coi khối dữ liệu ban đầu là đa thức bậc  $k$ , bên gửi sinh  $n$  điểm khác nhau từ đa thức đó và gửi đi, bên nhận nếu nhận đủ  $k$  điểm bất kì sẽ có thể nội suy lại đa thức ban đầu, là dữ liệu ban đầu.

VD: Dữ liệu ban đầu gồm 5 điểm  $\{1; -12; 49; -78; 40\}$ , đa thức bậc 4  $p(x) = x^4 - 12x^3 + 49x^2 - 78x + 40$ , tính giá trị  $p(x)$  tại 6 điểm  $\{(0, 40); (1, 0); (2, 0); (3, 4); (4, 0); (5, 0)\}$ , bên nhận chỉ cần nhận được 5 điểm bất kì trong 6 điểm trên là có thể suy ra lại dữ liệu ban đầu.



Hình 1.4 Cơ bản về mã sửa lỗi Reed-Solomon (erasure code)

Tuy nhiên, thuật toán đơn giản như trên chỉ là ý tưởng cơ bản, có 2 vấn đề. Thứ nhất, vì với đa thức lớn bậc vài nghìn thì giá trị của đa thức cũng không có giới hạn, có thể lên tới cả triệu tỷ chữ số. Thứ hai, chưa có thuật toán đủ hiệu quả để tính nhiều điểm hay nội suy lại đa thức từ tập điểm bất kì, độ phức tạp thuật tính đa điểm hay nội suy tổng quát tốt nhất hiện tại là  $O(N^2)$ . Với các điểm dữ liệu đặc biệt, thuộc trường hữu hạn  $F_p$  với  $p$  là số nguyên tố Fermat,  $p = 2^{2^q} + 1$ , thì tính chất nội suy đa thức vẫn đúng và tồn tại thuật toán hiệu quả với độ phức tạp  $O(N \log N)$  được nêu trong bài báo [1], hệ thống trong đề tài thiết kế theo thuật toán này và được cài đặt lại để xử lý song song.

### 1.2.2. Trường $GF(65537)$ và biến đổi số Fermat (Fermat Number Transform)

Trường Galois (GF) hay trường hữu hạn là cấu trúc toán học rất hữu dụng được ứng dụng nhiều trong mật mã học và lý thuyết thông tin. Lý do là vì trường hữu hạn giúp kiểm soát giá trị dữ liệu trong một khoảng và tồn tại rất nhiều thuật toán hiệu quả với tập các phần tử trong trường.

Với  $p$  là một số nguyên tố, tất cả số nguyên không âm bé hơn  $p$  tạo thành một trường hữu hạn, các phép toán giữa các phần tử trong trường  $a_i, a_j \in GF(p)$  (cộng, trừ, nhân, chia số khác 0) chia lấy dư cho  $p$  đều thuộc trường  $GF(p)$ . Trong trường  $GF(p)$ , thứ tự (order) của phần tử được sắp xếp theo bậc mũ thấp nhất của phần tử đó đồng dư với 1 khi chưa lấy dư cho  $p$ , phần tử có thứ tự  $p - 1$  gọi là nghiệm nguyên thủy (primitive root). Nghiệm nguyên thủy có thể sinh ra mọi phần tử khác (trừ phần tử 0) trong trường bằng các bậc mũ từ 0 đến  $p - 1$ . Ví dụ 3 là nghiệm nguyên thủy trong  $GF(65537)$  vì  $3^{65536} \equiv 1 \pmod{65537}$ ,  $3^i \neq 1$  với mọi  $0 \leq i < 65536$ .

Trong trường hữu hạn, biến đổi Fourier (Fourier transform) vẫn có ý nghĩa tương tự với trên số thực hay số phức, được nêu trong bài báo [9]. Với  $r$  là nghiệm nguyên thủy của  $GF(p)$ , vector  $a = (a_0, \dots, a_{n-1})$  kích thước  $n$  với các phần tử trong  $GF(p)$ ,  $A = (A_0, \dots, A_{n-1}) = DFT(a)$ , có các công thức:

$$DFT: A_j = \sum_{i=0}^{n-1} a_i r^{ij}$$

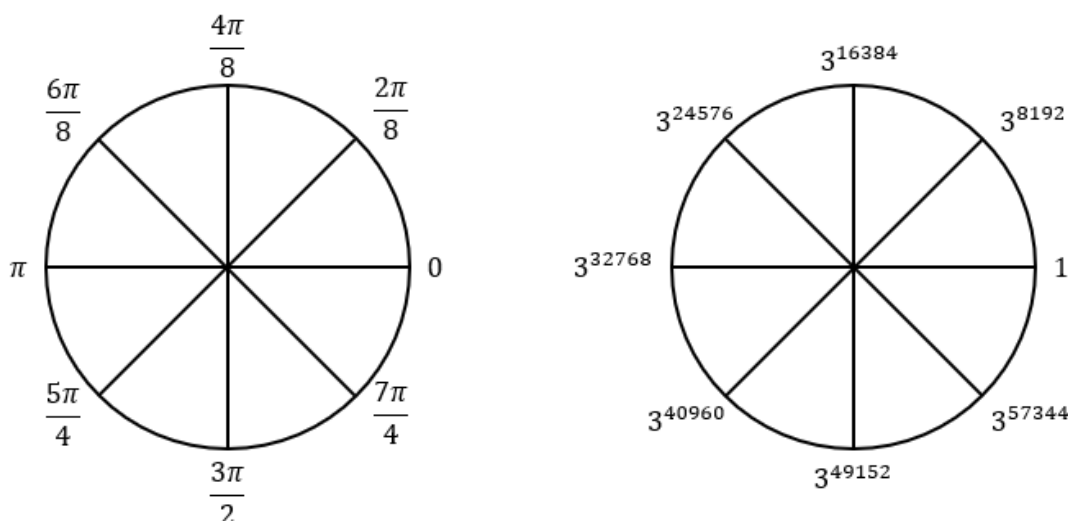
$$DFT^{-1}: a_j = \frac{1}{n} \sum_{i=0}^{n-1} A_i r^{-ij}$$

Biến đổi Fourier có nhiều ý nghĩa, trong đề tài chỉ tập trung về ý nghĩa tính (biến đổi xuôi) và nội suy (biến đổi ngược). Trong công thức xuôi, vector  $a$  tương tự như hệ số đa thức, các  $r^j$  tương tự như các điểm trong trường hữu hạn vì  $r$  là nghiệm nguyên thủy nên  $r^j$  có thể tạo ra các điểm với mỗi  $j$ , vậy nên  $r^{ij}$  giống như các bậc mũ của điểm đó. Tương tự với công thức biến đổi ngược.

Thuật toán biến đổi Fourier nhanh (Fast Fourier Transform) trên bộ số phức nằm cách đều trong vòng tròn đơn vị là thuật toán hiệu quả nhất dùng để tính DFT,  $DFT^{-1}$  với độ phức tạp tốt  $O(N \log N)$ . Để áp dụng thuật toán tương tự FFT vào các phần tử trong  $GF(p)$  cần tìm hiểu hơn về cách hoạt động của FFT. Thuật toán FFT đạt được độ phức tạp tốt nhờ tận dụng tính chất đối xứng chẵn của hàm đa thức và tính chất của bộ số phức đó. Cụ thể hơn về bộ số phức, bộ số phức được dùng cho FFT phải có dạng là

một nhóm tuần hoàn dưới phép nhân và kích thước nhóm đó phải là bội số lũy thừa của 2 hay cụ thể hơn thì tồn tại nhóm tuần hoàn con trong nhóm đó có kích thước đúng bằng một nửa. Trường  $GF(p)$  với  $p$  là số nguyên tố bản chất cũng đã là phiên bản chặt hơn của nhóm tuần hoàn dưới phép nhân. Vậy để áp dụng FFT trên  $GF(p)$  thì trường cần có số lượng phần tử là bội số lũy thừa của 2, cụ thể thì  $p - 1$  phải là bội số lũy thừa của 2, vì vậy nên chọn  $p = 2^{2^4} + 1 = 65537$ , số  $p$  này được gọi là số Fermat (Fermat Number), thuật toán FFT trên  $GF(65537)$  gọi là Fermat Number Transform (FNT).

Mỗi khi dùng biến đổi số Fermat xuôi để sinh ra vector  $n$  phần tử thì dùng các phần tử trong  $GF(65537)$  có dạng  $3^{i \frac{65536}{n}}$  vì 3 là nghiệm nguyên thủy. Tương tự dùng  $3^{-i \frac{65536}{n}}$  hay  $21846^{i \frac{65536}{n}}$  cho biến đổi ngược vì 21846 là nghịch đảo của 3 trong  $GF(65537)$ .



Hình 1.5 Ví dụ về 8 điểm dùng trong FFT so với FNT

Chọn  $p = 65537$  vẫn có một vấn đề nhỏ, chỉ mỗi phần tử 65536 không phải số 16-bit, vẫn có thể xử lý bằng việc tính toán chia trường hợp cẩn thận và thiết kế cấu trúc gói tin phù hợp, vấn đề này sẽ được thảo luận sau. Hiện giờ cứ coi như mỗi phần tử đều 16-bit.

### 1.2.3. Mã sửa lỗi Reed-Solomon (Erasure code) trên trường $GF(65537)$

Thuật toán mã hóa rất đơn giản. Từ dữ liệu ban đầu chia thành các mảnh nhỏ, chia tiếp mảnh nhỏ ra làm  $k$  kí hiệu,  $k$  là lũy thừa của 2, mỗi kí hiệu 16-bit, đó là đa thức ban đầu. Có thể cần thêm phần đệm (padding) vào dữ liệu ban đầu để đảm bảo các tính chất trên. Tính giá trị đa thức bằng biến đổi số Fermat tại  $n$  điểm để thu được  $n$  kí hiệu. Độ phức tạp thuật toán mã hóa  $O(N \log N)$ .

Thuật toán giải mã thì phức tạp hơn nhiều. Tương tự mã hóa, cũng sẽ giải mã theo các mảnh nhỏ. Trong mỗi mảnh, sẽ khôi phục lại được đa thức gốc nếu thu được  $k$  kí hiệu từ  $n$  kí hiệu gửi đi, các kí hiệu là vị trí và giá trị các điểm của đa thức gốc. Giả sử  $k$  điểm đó được cho bởi vector vị trí  $x = (x_0, \dots, x_{k-1})$  và vector giá trị  $y = (y_0, \dots, y_{k-1})$ , cần tìm lại vector đa thức  $p = (p_0, \dots, p_{k-1})$  là dữ liệu ban đầu.

Thuật toán nội suy trong giải mã sẽ được tối ưu từ nội suy Lagrange, công thức:

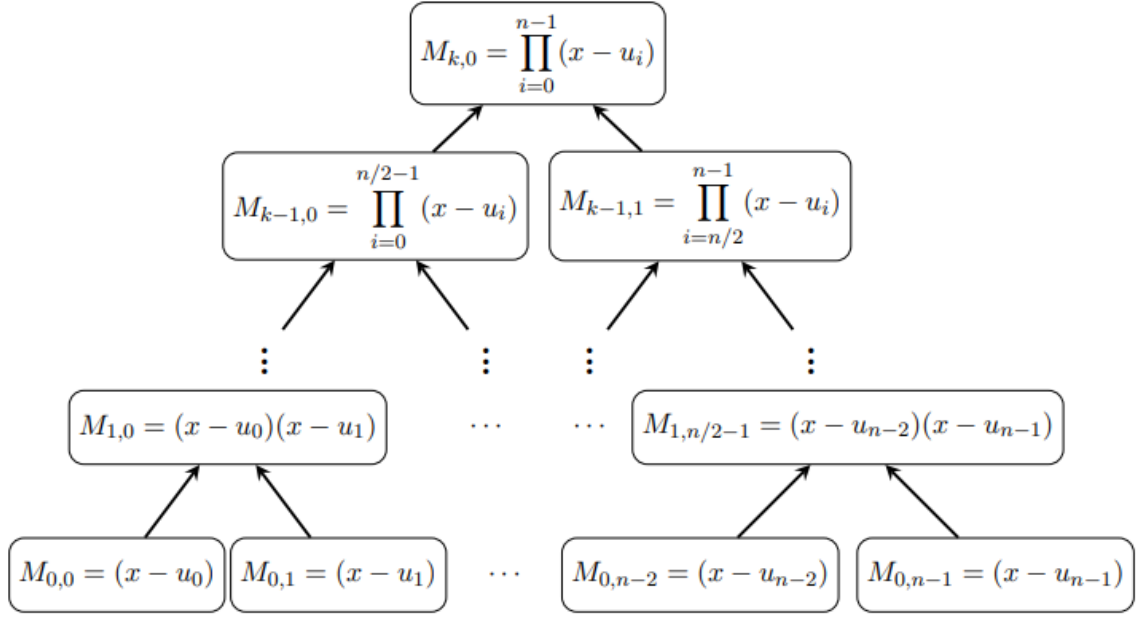
$$p(x) = \sum_{i=0}^{k-1} \left( y_i * \prod_{\substack{j \neq i \\ 0 \leq j \leq k-1}} \frac{x - x_j}{x_i - x_j} \right)$$

Đặt  $A(x) = \prod_{0 \leq j \leq k-1} (x - x_j)$ ,  $A_i(x) = \prod_{\substack{j \neq i \\ 0 \leq j \leq k-1}} (x - x_j) = \frac{A(x)}{x - x_i}$ , biến đổi công thức nội suy:

$$p(x) = \sum_{i=0}^{k-1} y_i * \frac{A_i(x)}{A_i(x_i)} = A(x) \sum_{i=0}^{k-1} \frac{y_i}{x - x_i} \frac{1}{A_i(x_i)}$$

Ý tưởng của những biến đổi trên là đưa công thức về những phần có thể tính hiệu quả được, cụ thể là phần  $A(x)$  và phần  $\sum_{i=0}^{k-1} \frac{y_i}{x - x_i} \frac{1}{A_i(x_i)}$ . Sau đó nhân hai đa thức đó lại tạo thành  $P(x)$ .

Phần  $A(x)$  cách tính tổng quát nhất là cứ giữ một tập hợp các đa thức rồi cứ lần lượt lấy các đa thức bậc thấp nhất nhân lại với nhau. Tuy nhiên trong bài toán các đa thức nhận được ban đầu đều cùng bậc với nhau, vậy nên đơn giản và hiệu quả hơn nữa là dùng kĩ thuật cây tích con (Subproduct Tree), ý tưởng cơ bản là cứ bỏ các đa thức vào một mảng, giả sử mảng gồm  $n$  đa thức và nhân hai đa thức cạnh nhau thành một đa thức mới, thu được mảng mới gồm  $\frac{n}{2}$  đa thức, cứ lặp lại như vậy đến khi còn lại một đa thức, đó là  $A(x)$ . Độ phức tạp của đoạn này là  $O(N \log^2 N)$  nhưng trong bài toán nếu coi số lượng gói tin cần nhận là hằng số thì độ phức tạp trở thành  $O(N \log N)$ . Kĩ thuật cây tích con có thể tìm hiểu thêm trên bài báo [7].



Hình 1.6 Thuật toán cây tích con (Subproduct Tree)

Phần  $\sum_{i=0}^{k-1} \frac{y_i}{x-x_i}$  ý tưởng là sẽ tính  $n_i = \frac{y_i}{A_i(x_i)}$  trước, sau đó tính  $\sum_{i=0}^{k-1} \frac{n_i}{x-x_i}$ . Phần này có thể tính được nhờ vào tính chất đặc biệt khi đạo hàm  $A(x)$  và biến đổi với chuỗi Taylor trên hàm giải tích (analytic function)  $\frac{1}{c-x}$ .

Xét đạo hàm  $A(x)$ :

$$A'(x) = (A(x))' = \left( \prod_{0 \leq j \leq k-1} (x - x_j) \right)' = \sum_{i=0}^{k-1} \prod_{0 \leq j \leq k-1, j \neq i} (x - x_j) = \sum_{i=0}^{k-1} A_i(x)$$

Nhận xét,  $A_i(x) = 0 \forall x = x_j, 0 \leq j \leq k-1, i \neq j$ . Vậy  $A'(x) = A_i(x) \forall x = x_i, 0 \leq i \leq k-1$ . Đây là kết quả quan trọng vì  $A(x)$  đã tính được ở trên và tính đạo hàm  $A(x)$  cũng rất dễ,  $x_i \in GF(65537)$  và 65537 (thực tế hệ thống thiết kế với  $k = 32768, n = 65536$ ) cũng không phải số lớn nên có thể dùng FNT để tính tất cả giá trị rồi lọc ra các giá trị  $A_i(x_i)$ . Vậy các giá trị  $n_i$  đã có thể tính với độ phức tạp  $O(N \log N)$ .

Xét chuỗi Taylor của hàm giải tích:  $\frac{1}{c-x} = \sum_{j=0}^{\infty} c^{-j-1} x^j$ . Vậy  $\frac{n_i}{x-x_i} = -\sum_{j=0}^{\infty} n_i x_i^{-j-1} x^j$ . Vì  $\sum_{i=0}^{k-1} \frac{n_i}{x-x_i} = \frac{P(x)}{A(x)}$  và  $P(x)$  là đa thức bậc  $k-1$  nên chỉ cần quan tâm đến bậc  $n-1$  trong kết quả của  $\frac{P(x)}{A(x)}$ , tương đương chia lấy dư cho  $x^n$ .

$$\sum_{i=0}^{k-1} \frac{n_i}{x - x_i} \bmod x^n = - \sum_{i=0}^{k-1} \sum_{j=0}^{n-1} n_i x_i^{-i-1} x^j = - \sum_{j=0}^{n-1} x^j \sum_{i=0}^{k-1} n_i x_i^{-j-1}$$

Với  $r$  là nghiệm nguyên thủy nên  $x_i = r^{z_i}$  với  $z_i < n$ , thay vào và sắp xếp theo  $z_i$  thì phần  $\sum_{i=0}^{k-1} n_i x_i^{-j-1}$  trở thành như tính giá trị đa thức  $N(x) = \sum_{i=0}^{k-1} n_i x^{z_i}$  bậc  $n-1$  tại  $n$  điểm  $r^{-j-1}$  hay  $(r^{-1}, \dots, r^{-n})$ . Phần tính giá trị tại  $n$  điểm đó có thể tính bằng FNT, tính dùng thuật tương tự FNT ngược nhưng bước cuối không chia  $n$ , sau khi tính xong các giá trị đó thế vào lại công thức sẽ là  $n$  hệ số đầu của đa thức  $\frac{P(x)}{A(x)}$  cần tìm. Độ phức tạp của phần này là  $O(N \log N)$ .

$$\sum_{i=0}^{k-1} \frac{n_i}{x - x_i} \bmod x^n = \sum_{j=0}^{n-1} -N(r^{-j-1}) x^j$$

Cuối cùng vì  $P(x)$  bậc  $k-1$  nên công thức để khôi phục  $P(x)$  với những phần đã tính được ở trên là:

$$P(x) = \left( (A(x) \bmod x^k) \left( \frac{P(x)}{A(x)} \bmod x^k \right) \right) \bmod x^k$$

Đó là kết thúc của quá trình giải mã,  $P(x)$  chính là dữ liệu ban đầu, độ phức tạp thuật toán giải mã là  $O(N \log N)$ .

### 1.3. Thiết kế thuật toán song song

#### 1.3.1. Tổng quan về thiết kế thuật toán song song

Thuật toán song song là thuật toán mà mỗi bước có nhiều phép tính có thể thực hiện song song, không có sự phụ thuộc về thứ tự.

Thiết kế thuật toán song song thường là tìm cách loại bỏ sự phụ thuộc giữa các phép tính trong thuật toán tuần tự truyền thống hay phức tạp hơn là thiết kế thuật toán mới có sự phụ thuộc giữa các phép tính ít hơn nhưng vẫn đạt được kết quả như cũ, trong một số trường hợp chỉ cần xấp xỉ kết quả cũ.

Độ phức tạp thuật toán song song được đánh giá dựa trên 2 yếu tố:  $W$  – Tổng khối lượng đơn vị công việc cần xử lý trong thuật toán, tương tự như độ phức tạp thuật toán,  $D$  – Độ dài đường đi dài nhất công việc tuần tự trong thuật toán, tương tự như đường găng trên sơ đồ lịch trình. Sau khi xác định các yếu tố trên, độ phức tạp của thuật toán song song là  $O(\frac{W}{P} + D)$ ,  $P$  là kí hiệu cho số bộ xử lý. VD: Thuật toán có  $W = O(N \log N)$ ,  $D = O(\log N)$  thì độ phức tạp thuật toán song song là  $O(\frac{N \log N}{P} + \log N)$ .

Trong đề tài chủ yếu sẽ trình bày thiết kế thuật toán song song cho những phần không quá rõ ràng cách xử lý song song, những thuật toán có dạng quá rõ ràng như  $A \rightarrow B, B_i = f(A_i) \forall i$ , thì sẽ chỉ đề cập đến độ phức tạp song song.

### 1.3.2. Thuật toán biến đổi số Fermat song song

Hiện tại các thư viện lớn như FFTW (CPU) hay CuFFT (GPU) đều chưa hỗ trợ biến đổi số Fermat trên GF(65537), vậy nên việc tự cài đặt hiệu quả thuật toán FNT song song thay vì dùng các thư viện khác sẽ mang lại nhiều lợi ích về sự linh hoạt và tối ưu riêng phù hợp điều kiện bài toán.

Thuật toán song song sẽ thiết kế dựa vào tối ưu thuật toán FFT cổ điển. Thuật toán FFT cổ điển (chia để trị) gồm 2 giai đoạn chính.

Giai đoạn một mục đích là sắp xếp hệ số đa thức theo một thứ tự phù hợp. Cụ thể, chia đa thức thành 2 đa thức, một đa thức chứa bậc lẻ và một đa thức chứa bậc chẵn, đa thức chứa bậc chẵn sắp xếp qua bên trái và lẻ qua bên phải, đặt  $x_n = x^2$  và tiếp tục quá trình đó với mỗi đa thức đến khi bậc đa thức là 0. VD: Với  $n = 8$  và FFT trên vector  $a = (a_0, \dots, a_7)$ , sau giai đoạn một thu được vector  $b = (a_0, a_4, a_2, a_6, a_1, a_5, a_3, a_7)$ .

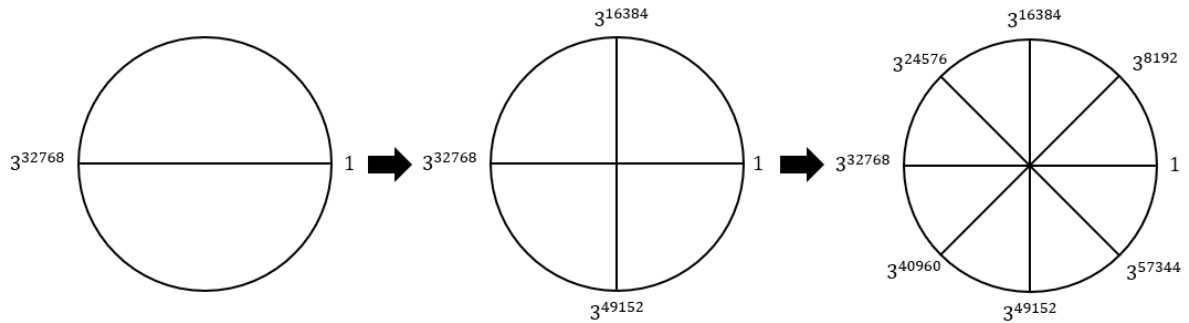
Để cài đặt thuật toán song song cho giai đoạn một, cần nhận xét về vị trí của các phần tử sau khi sắp xếp. Cụ thể, gọi vị trí của một phần tử trong vector ban đầu là  $x$  ( $x < n$ ), gọi  $rev\_x$  là giá trị khi đảo ngược vị trí các bit trong  $x$ , sau đó sắp xếp các phần tử theo  $rev\_x$  từ nhỏ tới lớn. Vì chọn  $p = 65537$  và chỉ FFT trên các vector có kích thước là lũy thừa của 2, vậy nên chỉ có thể có 16 cấu hình kích thước  $n$ , có thể tính trước tất cả các cách sắp xếp với 16 cấu hình này để tăng tốc giai đoạn một. Vậy nếu tính sẵn 16 cách sắp xếp các cấu hình kích thước, giai đoạn một bây giờ trở thành thuật toán song song đơn giản, độ phức tạp thuật toán song song  $O\left(\frac{N}{P}\right)$ .

Giai đoạn hai mục đích là tính dần giá trị của đa thức tại các điểm trên nhóm tuần hoàn con, dần dần sinh ra và tính đủ các điểm cần tính. Cụ thể, thuật toán giai đoạn hai sẽ lặp  $\log(n)$  lần, với  $n$  là kích thước vector kết quả, ở lần lặp thứ  $i$  sẽ tính trên  $\frac{n}{2^{i+1}}$  mảnh,  $0 \leq i < \log(n)$ , trong mỗi mảnh sẽ gộp giá trị các điểm của mảnh lần lặp trước và sinh ra thêm điểm mới cho mảnh hiện tại. Nếu chỉ song song theo từng mảnh thì càng lên tầng trên sẽ càng ít mảnh dẫn đến không tối ưu. Nhận thấy mỗi tầng của thuật toán số công việc cần làm khá ổn định, luôn có  $\frac{n}{2}$  công việc ở phần gộp giá trị và sinh thêm điểm mới, vậy nên cài đặt song song được theo từng công việc được thì sẽ rất tối ưu. Để song song theo từng công việc thì ở mỗi công việc cần biết đang tính thuộc mảnh nào, đang ở vị trí thứ mấy trong mảnh và hệ số xoay của vị trí đó trong mảnh. Hệ số xoay của một

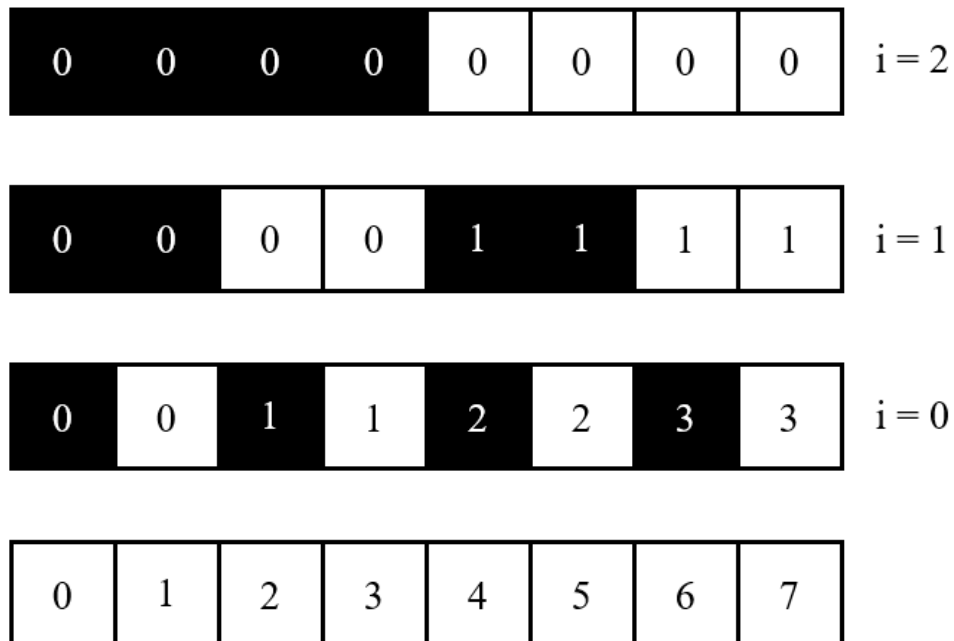


vị trí trong một mảnh kích thước bất kì có thể được tính sẵn vì cũng chỉ có 16 cấu hình kích thước của mảnh. Xét về độ phức tạp,  $D = O(\log N)$  vì phải tính theo từng tầng,  $W = O(N \log N)$ . Vậy độ phức tạp thuật toán song song giai đoạn hai là  $O(\frac{N \log N}{P} + \log N)$ .

Thuật toán FNT còn một bước cuối là chia kết quả cho  $n$  nếu muốn tính nghịch đảo, bước cuối này đơn giản và độ phức tạp như bước một. Vậy độ phức tạp của thuật toán FNT là  $O(\frac{N \log N}{P} + \log N)$ .



Hình 1.7 Quá trình sinh ra 8 điểm trong thuật toán FNT qua các tầng (mô hình sinh nhóm tuần hoàn)



Hình 1.8 Quá trình sinh ra 8 điểm trong thuật toán FNT qua các tầng (mô hình công việc trên mảng)

### 1.3.3. Mã sửa lỗi Reed-Solomon (Erasure code) song song

Khi đã cài đặt được thuật toán FNT song song thì các bước ở mã hóa giải mã đều trở thành thuật toán song song đơn giản, ngoại trừ bước xây dựng  $A(x)$  ở phần giải mã.

Trước khi nói về thuật toán xây dựng  $A(x)$  hiệu quả, cần xem xét về cách nhân hai đa thức hiệu quả bằng FNT vì nhân đa thức là công việc chính khi xây dựng  $A(x)$ . Cụ thể, cho hai đa thức bậc  $n - 1$ ,  $P_1(x)$ ,  $P_2(x)$ , đa thức kết quả  $P(x) = P_1(x)P_2(x)$  sẽ có bậc  $2n - 2$ . Vậy có thể dùng FNT để tính giá trị  $P_1(x)$ ,  $P_2(x)$  tại  $2n - 1$  điểm dữ liệu rồi nhân từng giá trị lại với nhau,  $2n - 1$  điểm giá trị sau khi nhân cũng là giá trị của  $P(x)$ , có thể dùng để nội suy lại  $P(x)$  bằng  $FNT^{-1}$ . Nếu bậc của đa thức đủ lớn thì thuật toán nhân sẽ hiệu quả vì tận dụng được thuật toán FNT song song đã đề cập ở trên. Độ phức tạp thuật toán nhân đa thức song song tương tự thuật toán FNT,  $O(\frac{N \log N}{P} + \log N)$ .

$$P_1(x) = a_0 + \dots + a_{n-1}x^{n-1}$$

$$P_2(x) = b_0 + \dots + b_{n-1}x^{n-1}$$

$$P(x) = P_1(x)P_2(x) = FNT^{-1}(FNT(P_1(x)) * FNT(P_2(x)))$$

Vì hệ thống truyền thông qua giao thức UDP, nên các đa thức xử lý ở phần xây dựng  $A(x)$  luôn có bậc đủ lớn, chứa tích của các đa thức bậc một của cả gói tin, các đa thức này được tính trước, chỉ cần biết số thứ tự của gói tin trong mảnh là có thể suy ra đa thức. Trong hệ thống xây dựng mã với  $k = 32768$ ,  $n = 65536$ , gói tin chứa 512 kí hiệu, nghĩa là mỗi đa thức của gói tin bậc 1024, chỉ cần nhận 64 gói tin trong 128 gói tin là có tái tạo mảnh ban đầu.

Nhận thấy càng lên tầng cao của cây tích con khi xây dựng  $A(x)$  thì số nhánh cần tính càng giảm đi một nửa nhưng kích thước của các đa thức cần nhân cũng tăng gấp đôi. Vì các đa thức này đủ lớn nên có thể dùng FNT để nhân các đa thức lớn này song song hiệu quả, số lượng công việc song song ở mỗi tầng vẫn luôn giữ ổn định.

Độ phức tạp thuật toán mã hóa song song là độ phức tạp của thuật toán FNT  $O(\frac{N \log N}{P} + \log N)$ . Độ phức tạp thuật toán giải mã song song là độ phức tạp của quá trình xây dựng  $A(x)$ , nếu coi số lượng gói tin cần nhận là hằng số (64 gói tin, cây tích con gồm 5 tầng), độ phức tạp quá trình xây dựng  $A(x)$  sẽ là độ phức tạp của một tầng trên cây tích con,  $W = O(N \log N)$ ,  $D = O(\log N)$ , độ phức tạp thuật toán giải mã song song là  $O(\frac{N \log N}{P} + \log N)$ .

## 1.4. Công nghệ và kiến trúc phần cứng sử dụng

### 1.4.1. Công nghệ sử dụng

Hệ thống sử dụng ngôn ngữ C/C++ để lập trình thuật toán song song trên CPU và GPU, sử dụng ngôn ngữ Java để lập trình phản ứng dụng.

Để lập trình xử lý đa luồng trên CPU hiệu quả, hệ thống sử dụng thư viện oneAPI Threading Building Blocks của Intel.

Hệ thống được cài đặt để chạy tốt trên hệ điều hành Ubuntu 22.04 LTS, trên Windows vẫn có thể sử dụng nhưng chạy không quá tốt do WDDM driver (đặc biệt là WDDM 2.7) làm hạn chế khả năng xử lý song song của GPU, bản GPU trên Windows 10 sẽ có hiệu năng thấp hơn trên Linux khoảng 6 lần.

Để lập trình xử lý song song trên CPU, hệ thống sử dụng trình biên dịch ISPC giúp dễ dàng viết chương trình SPMD hiệu năng cao chạy trên CPU Intel hoặc GPU Intel Xe. Trình biên dịch ISPC cung cấp một tập câu lệnh có cú pháp bậc cao, dễ đọc hơn cú pháp của SIMD để người lập trình có thể dễ dàng viết chương trình song song trên CPU Intel và khi biên dịch sẽ dịch ra lại mã C/C++ với cú pháp SIMD. Câu lệnh biên dịch bởi trình biên dịch ISPC có hiệu năng rất cao, tiệm cận với khi viết cú pháp SIMD bình thường. ISPC là trình biên dịch SIMD có hiệu năng cao nhất theo CppCon 2016.

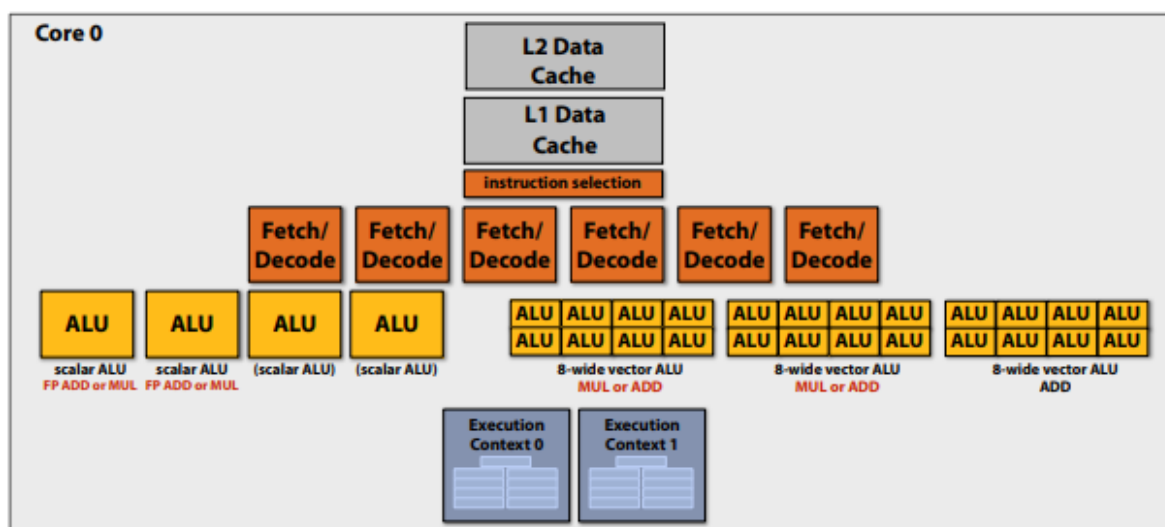
Để lập trình xử lý song song trên GPU, hệ thống sử dụng CUDA giúp dễ dàng viết chương trình SPMD hiệu năng cao chạy trên GPU Nvidia, với cấu trúc chương trình tương tự ISPC. CUDA là một hệ sinh thái gồm kiến trúc trừu tượng, trình biên dịch, công cụ lập trình, đo lường, tối ưu, hỗ trợ phát triển các chương trình trên GPU Nvidia. CUDA và GPU Nvidia hiện nay dẫn đầu trong lĩnh vực tính toán hiệu năng cao.



Hình 1.9 Công nghệ sử dụng

### 1.4.2. Kiến trúc cơ bản và lập trình song song trên CPU

Để cài đặt thuật toán song song hiệu quả trên CPU, cần hiểu rõ về kiến trúc của CPU, sau đây là kiến trúc một nhân Intel Skylake, cơ bản không quá khác biệt so với CPU i5 12400 kiến trúc Intel Alderlake dùng trong hệ thống:



Hình 1.10 Kiến trúc cơ bản một nhân CPU (Intel Skylake), nguồn [2]

Một nhân CPU có hỗ trợ công nghệ siêu phân luồng (Hyper-Threading) sẽ có hai luồng phần cứng (hardware thread), hệ điều hành khi sử dụng CPU sẽ nhìn thấy gấp đôi số nhân (execution context) có thể sử dụng, mục đích của luồng phần cứng này là để thay đổi qua lại trong lúc đợi đọc toán tử từ bộ nhớ giúp hạn chế thời gian chết của CPU, luồng phần cứng khác với luồng phần mềm hay luồng hệ điều hành (OS thread) ở chỗ là nó có riêng bộ thanh ghi, cho phép quá trình chuyển đổi qua lại giữa hai luồng cực kì hiệu quả, không cần thông qua lời gọi hệ thống (system call) để tránh xung đột như luồng hệ điều hành. Cache (L1, L2, L3) cũng là thành phần giúp giảm thời gian đọc dữ liệu từ bộ nhớ (RAM). Sau đây là bảng so sánh thời gian đọc toán tử từ các vùng bộ nhớ khác nhau, nguồn [2]:

Đỗ trễ khi đọc dữ liệu từ bộ nhớ của kiến trúc Kabylake, đơn vị: số chu kì CPU	
L1 cache	4
L2 cache	12
L3 cache	38
DRAM	248

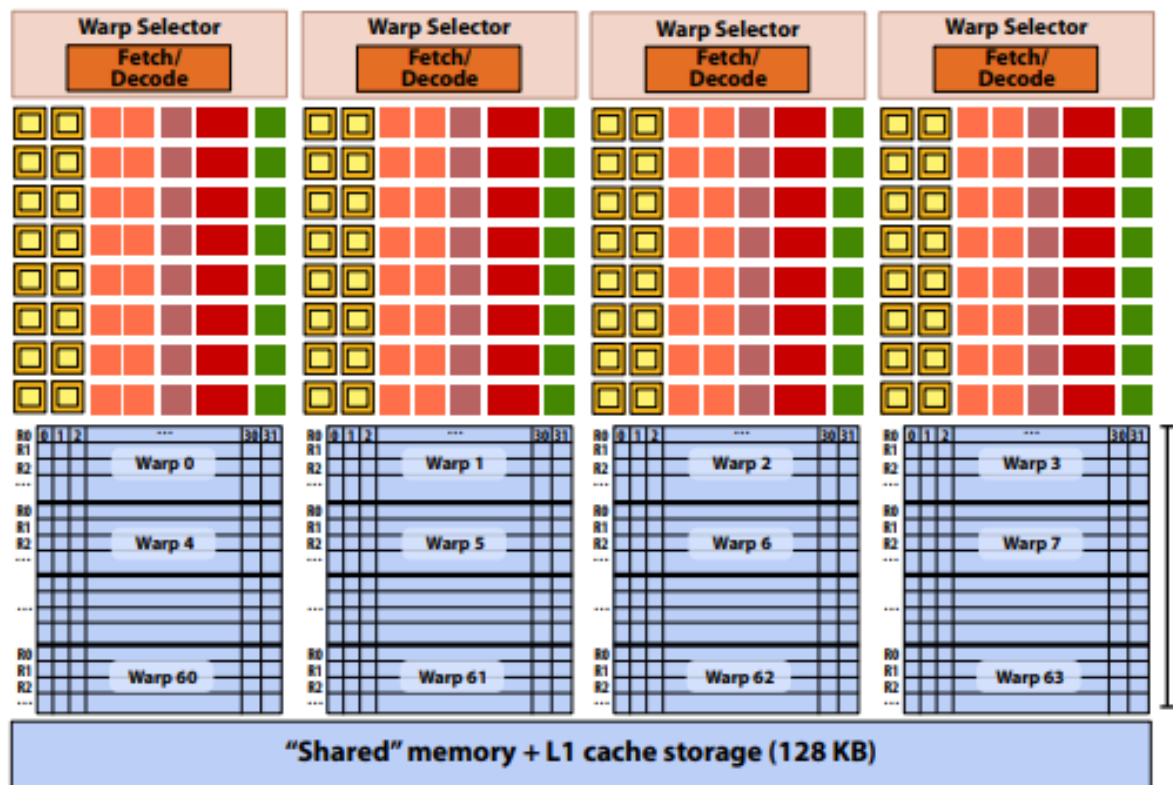
Mỗi chu kì CPU, bộ chọn câu lệnh (instruction selection) sẽ có thể chọn một câu lệnh đã sẵn sàng thực thi từ một trong hai luồng phần cứng, câu lệnh được bộ đọc, giải mã (Fetch / Decode) lệnh đọc để phân giải câu lệnh thành các câu lệnh của CPU để thực thi. Nếu câu lệnh có dạng vô hướng (scalar) thì có thể được xử lý bằng các bộ xử lý phép

tính vô hướng ALU (bên trái), nếu câu lệnh có dạng vector thì sẽ được xử lý bằng các bộ xử lý phép tính vector ALU (bên phải). CPU hiện đại còn có thể có nhiều nhân, cụ thể như CPU sử dụng để cài đặt hệ thống, Intel Core I5 12400 có 6 nhân 12 luồng, hỗ trợ AVX2-32x8 nghĩa là mỗi nhân có vector ALU có thể tính vector 8 phần tử 32-bit một lúc. Với việc cài đặt hợp lý thì có thể tăng tốc so với thuật toán tuần tự đơn luồng từ 32 đến 64 lần.

Cài đặt thuật toán song song trên CPU là tìm cách để viết câu lệnh để có thể sử dụng các bộ xử lý phép tính vector ALU (SIMD) trong mỗi nhân hiệu quả, kết hợp xử lý đa nhân, đa luồng để tận dụng hết sức mạnh và hạn chế thời gian chết của CPU.

#### 1.4.3. Kiến trúc cơ bản và lập trình song song trên GPU

Để cài đặt thuật toán song song hiệu quả trên GPU, cần hiểu rõ về kiến trúc của GPU, sau đây là cơ bản một SM của GPU V100 kiến trúc Volta, mỗi SM lại được chia làm 4 nhân con, một GPU thường gồm vài chục đến hơn 100 SM, cơ bản không quá khác biệt so với GPU RTX 3060 kiến trúc Ampere dùng trong hệ thống, tuy nhiên sự khác biệt giữa các đời GPU là nhiều hơn so với giữa các đời CPU.



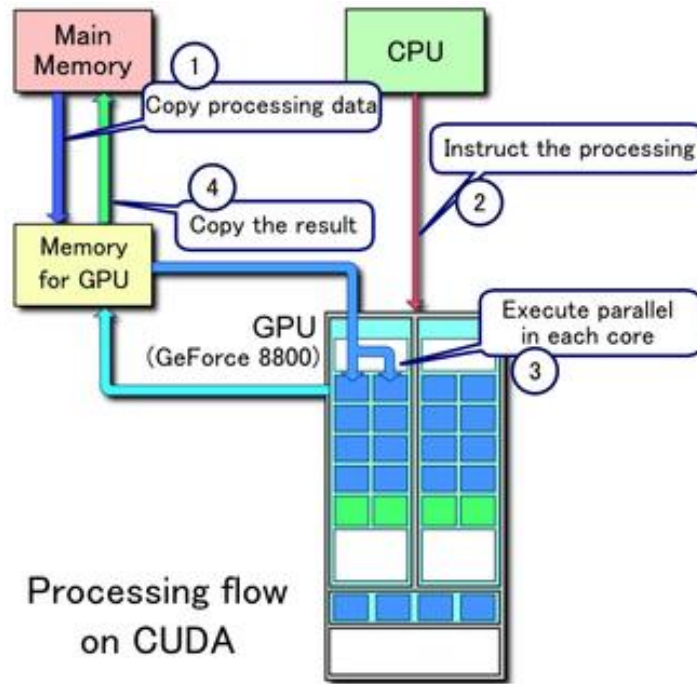
Hình 1.11 Kiến trúc cơ bản của một SM trong GPU V100, nguồn [2]

Tương tự nhân CPU, SM trong GPU cũng có luồng phân cứng, đây là cấp độ nhỏ nhất về tính toán trong GPU, tuy nhiên GPU không đổi qua lại giữa từng luồng mà 32

luồng được nhóm lại thành một nhóm gọi là Warp, GPU sẽ đổi qua lại giữa các Warp giúp hạn chế thời gian chết. Ở mỗi chu kỳ GPU, bộ chọn Warp (Warp Selector) sẽ chọn một Warp đã sẵn sàng để thực thi, ở các dòng GPU kiến trúc Volta về trước, các luồng trong Warp đều dùng chung một bộ đếm chương trình (Program Counter), điều đó làm cho Warp khá giống luồng phần cứng CPU còn luồng phần cứng lại giống làn SIMD trong CPU, tuy nhiên ở các kiến trúc hiện đại thì CPU và GPU đã tương đồng về các khái niệm hơn. Trong mỗi SM cũng có cache L1 giúp giảm thiểu thời gian khi đọc từ bộ nhớ VRAM nhiều lần, các SM trong GPU cũng có cache L2 dùng chung để tiếp tục giảm thời gian truy cập bộ nhớ. Tuy nhiên kích thước cache mỗi nhân hay mỗi luồng của GPU thấp hơn nhiều so với CPU, gấp vài chục đến vài trăm lần, vì vậy để giảm thiểu thời gian chết, mỗi SM có số lượng Warp, luồng phần cứng cao hơn nhiều so với CPU. Cụ thể SM của GPU thường có thể chứa từ 32 tới 64 Warp để thay đổi qua lại so với chỉ 2 luồng phần cứng mỗi nhân trên CPU.

Như đã đề cập, từ các dòng GPU kiến trúc Ampere (sử dụng trong hệ thống) về sau thì mỗi luồng phần cứng trong Warp đã có bộ đếm chương trình riêng, điều này giúp tăng hiệu năng khi gặp các điều kiện rẽ nhánh phân kỳ (branch divergence) và khi không đủ bộ tính toán. Cụ thể về không đủ bộ tính toán, GPU hiện đại được tối ưu cho các phép tính dấu phẩy động 32-bit (FP32) trong khi hệ thống chủ yếu sử dụng kiểu số nguyên 32-bit (INT32). GPU RTX 3060 sử dụng trong hệ thống có 128 nhân Cuda mỗi SM và có tất cả 28 SM, vậy có tất cả 3584 nhân Cuda, mọi nhân Cuda đều có thể thực hiện một phép tính FP32 nhưng không phải nhân nào cũng có thể thực hiện phép tính trên INT32, cụ thể là chỉ một nửa nhân Cuda có thể thực hiện phép tính trên INT32. Tuy nhiên điều này chỉ là vấn đề khi chương trình cài đặt bị giới hạn về số phép tính, hầu hết chương trình song song đều bị giới hạn về thông lượng bộ nhớ.

Lập trình GPU Nvidia là lập trình theo kiến trúc trừu tượng Cuda. Trong Cuda, CPU gọi là Host và GPU gọi là Device. Các công việc có tính tuần tự thường sẽ thực hiện trên Host và các công việc có tính song song sẽ thực hiện trên Device. Người lập trình tìm cách chia chương trình thành các bước có thể xử lý song song, phần mã nguồn để thực hiện công việc song song ở mỗi bước gọi là kernel. Đảm bảo các kernel phát sinh đủ nhiều số lượng công việc song song để có thể lấp đầy các Warp, hạn chế thời gian chết của GPU.



Hình 1.12 Mô hình xử lý cơ bản trong kiến trúc Cuda

Host và Device có bộ nhớ riêng, cần phải thực hiện các lệnh sao chép dữ liệu giữa hai phía trước và sau khi tính toán, các sao chép này tốn rất nhiều thời gian. Vì vậy các lệnh sao chép này có thể là bất đồng bộ nếu dùng bộ nhớ không phân trang, các kernel chạy ở Device cũng bất đồng bộ với Host, cho phép Host thực hiện các câu lệnh tuần tự mà không phải đợi kernel. Host vẫn có thể đồng bộ với Device bằng một số câu lệnh đặc biệt.

## CHƯƠNG 2: THIẾT KẾ VÀ TỐI ƯU HỆ THỐNG

### 2.1. Thiết kế hệ thống

#### 2.1.1. Thiết kế cấu trúc gói tin và chi tiết hơn về mã hóa giải mã

Như đã đề cập tổng quan ở phần trên, hệ thống sẽ mã hóa giải mã với  $n = 65536$ ,  $k = 32768$  để đảm bảo tỉ lệ lỗi gói ngẫu nhiên cực thấp, đổi lại hệ thống chỉ có thể sử dụng tối đa một nửa thông lượng của mạng để gửi dữ liệu, một nửa còn lại là để gửi những gói tin dư thừa để đảm bảo khả năng sửa lỗi.

Một vấn đề nữa là khi tính toán trên GF(65537) dữ liệu sẽ nằm trong khoảng 0 đến 65536, riêng phần tử 65536 là số 17-bit. Vậy cần xử lý ngoại lệ số 65536 phù hợp trong các phép tính cũng như thiết kế gói tin.

Khi gửi dữ liệu trên mạng dùng giao thức UDP, tốt nhất nên gửi các gói tin đủ lớn để tránh việc router phải định tuyến nhiều lần cho các gói tin nhỏ nhưng cũng phải nhỏ hơn đơn vị vận chuyển nhỏ nhất của phần cứng trong mạng (MTU), MTU của mạng là 1500 byte, vậy gói tin nên chứa khoảng 1024 byte (chưa kèm các header). Vậy gói tin nên được thiết kế để chứa khoảng 512 kí hiệu, mỗi kí hiệu 17-bit. Sau đây là thiết kế gói tin (không bao gồm các trường mặc định như địa chỉ nguồn đích của UDP), các trường được sắp xếp theo thứ tự:

<i>Trường</i>	<i>Kích thước (byte)</i>	<i>Ý nghĩa của trường</i>
<i>file_id</i>	<i>16</i>	<i>UUID của file gửi để bên nhận nhóm các mảnh lại thành file gốc.</i>
<i>file_byte_cnt</i>	<i>8</i>	<i>Kích thước theo byte ban đầu của file.</i>
<i>chunk_id</i>	<i>8</i>	<i>Số thứ tự của mảnh trong file.</i>
<i>id</i>	<i>1</i>	<i>Số thứ tự của gói trong mảnh, chỉ có tối đa 128 gói tin trong một mảnh.</i>
<i>data</i>	<i>1024</i>	<i>Dữ liệu của gói, gồm 512 kí hiệu, mỗi kí hiệu 17-bit, trường này chỉ chứa 16 bit đầu của kí hiệu.</i>
<i>sub_data</i>	<i>64</i>	<i>Chứa 1 bit cuối của 512 kí hiệu trên.</i>



<i>checksum</i>	<i>16</i>	<i>Mã băm MD5 của các trường trong gói tin để bên nhận kiểm tra tính toàn vẹn.</i>
-----------------	-----------	--

Vậy gói tin có tổng kích thước 1137 byte, kèm thêm vào chục byte mặc định của gói tin UDP nữa thì gói vẫn dưới MTU 1500 byte, gói có kích vẫn vẫn đủ lớn để hạn chế áp lực cho router.

Như đã đề cập tổng quan ở phần trên, tỉ lệ lỗi gói khi truyền qua mạng là cực thấp, chủ yếu chỉ lỗi nếu dùng mạng không dây, vậy nên cách xử lý của hệ thống là đưa dạng kênh truyền về dạng kênh xóa (erasure channel), vậy nên thuật toán giải mã bao giờ cũng làm việc với đa thức ít nhất tạo từ 512 kí hiệu, đa thức đó bậc 1024. Để tăng tốc quá trình xây dựng đa thức  $A(x)$  trong phần giải mã, có thể tính trước các đa thức này vì các đa thức này chỉ phụ thuộc vào vị trí gói tin trong mảnh, bên nhận được gói tin chỉ cần kiểm tra tính toàn vẹn rồi nạp đa thức phù hợp để xây dựng  $A(x)$ .

### 2.1.2. Xây dựng và tối ưu xử lý mã hóa giải mã trên CPU

CPU có lượng cache mỗi nhân khá lớn, tốc độ tính toán trên số nguyên 32-bit cao, độ trễ truy cập bộ nhớ thấp hơn nhiều so với GPU. Xây dựng và tối ưu mã hóa giải mã trên CPU rất đơn giản, chỉ cần làm giống như lý thuyết xử lý song song ở trên với các câu lệnh SIMD, chưa cần tới xử lý đa nhân đa luồng là đã đạt tốc độ mã hóa 90 MB/s, giải mã 4 MB/s.

Để tận dụng được hết tất cả các nhân của CPU, có thể sinh ra các luồng hệ điều hành (OS thread) cho mỗi quá trình mã hóa giải mã trên mảnh. Mỗi luồng là một đoạn chương trình được báo cho hệ điều hành có thể thực hiện không cần quan tâm tới thứ tự so với luồng khác sau khi được cho phép chạy, hệ điều hành có thể tự do sắp xếp các luồng hệ điều hành này vào các luồng phần cứng mỗi nhân trên CPU để tận dụng được hết khả năng xử lý của CPU. Tuy nhiên việc tạo các luồng hệ điều hành này cần dùng lời gọi hệ thống (system call) và cần khởi tạo vùng bộ nhớ riêng cho luồng nên khá tốn kém nếu tạo nhiều luồng, thường chỉ nên tạo số luồng khoảng gấp đôi số nhân của CPU. Nếu tự cài đặt một nhóm các luồng (thread pool) và kéo công việc trong các hàng đợi ra để xử lý thì việc đồng bộ các công việc trong luồng này hiệu quả cũng là một vấn đề khó, các công việc phải được cài đặt theo kiểu luồng người dùng (user thread), có cơ chế kích hoạt sự kiện đồng bộ hay đẩy vào nhóm các luồng khi muốn tiếp tục thực thi song song. Chưa kể là các công việc có thời gian thực thi có thể không bằng nhau, nếu dùng nhiều hàng đợi khác nhau thì phải có cơ chế phân bổ công việc hiệu quả vào các hàng đợi, cơ bản độ khó phần này sẽ rất cao nếu muốn tự cài đặt hiệu quả và đi rất xa

vấn đề đang cần giải quyết. Vậy nên để xử lý đa nhân đa luồng hiệu quả, hệ thống sử dụng thư viện Threading Building Blocks (TBB) của Intel.

Sử dụng thư viện TBB để quản lý các luồng mã hóa giải mã riêng, trong mỗi luồng giải mã còn quản lý thêm các luồng cho quá trình nhân các đa thức con tại mỗi tầng khi xây dựng  $A(x)$ . Kết quả hệ thống đạt được tốc độ mã hóa 400 MB/s, giải mã 21 MB/s.

Một tối ưu nhỏ nữa là về phép tính chia lấy dư được sử dụng trong các phép tính của GF(65537). Cụ thể, phép chia và phép chia lấy dư là các phép tính rất tốn thời gian, phép chia thời gian chạy tốn gấp khoảng 10 lần phép cộng và phép nhân, phép chia lấy dư lại được cài đặt dựa trên phép chia, phép nhân và phép trừ.  $A \bmod M = A - \lfloor \frac{A}{M} \rfloor \cdot M$ . Vậy nếu có thể thực hiện phép chia lấy dư bằng ít hơn 10 phép tính cộng, trừ, nhân hay các phép toán trên bit thì hiệu năng sẽ tăng. Các phép tính nhân, chia cho các số là lũy thừa của 2 cũng thay bằng phép toán dịch bit. Các phép toán cộng trừ với các số nhỏ hơn 65537 cũng có thể thay chia lấy dư bằng lệnh điều kiện và một phép tính để điều chỉnh kết quả cho đúng.

Phép chia lấy dư số nguyên 32-bit với số 65537 có thể được tối ưu để thời gian chạy tốt hơn. Cụ thể, gọi  $x$  là số nguyên cần tính chia lấy dư,  $low$  là số nguyên có giá trị bằng 16-bit thấp của  $x$ ,  $high$  là số nguyên có giá trị bằng 16-bit cao của  $x$ :

$$low = x \& 0xFFFF$$

$$high = x \gg 16$$

$$x = low + (high \ll 16)$$

$$x \equiv low + (high \ll 16) \pmod{65537}$$

$$x \equiv low + high * 65536 \pmod{65537}$$

$$65536 \equiv -1 \pmod{65537}$$

$$x \equiv low - high \pmod{65537}$$

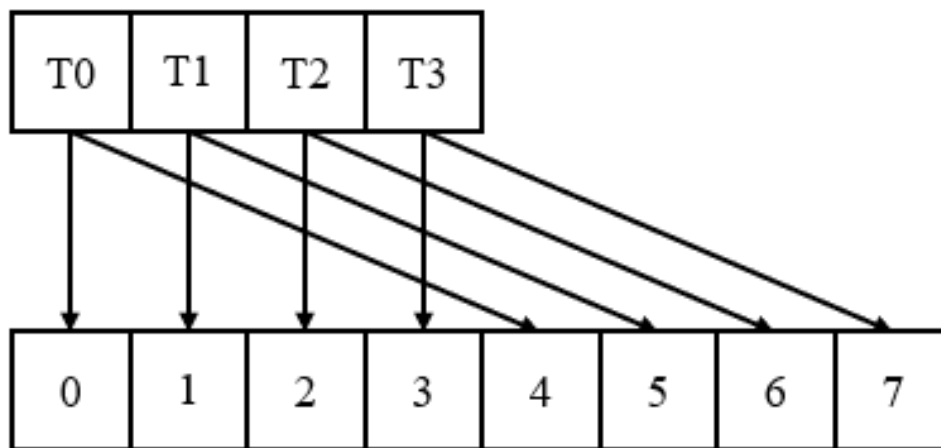
Với cách tính chia lấy dư  $x \equiv low - high \pmod{65537}$  và các thay đổi về một số phép cộng trừ số nhỏ như trên, hệ thống đạt được tốc độ mã hóa 420 MB/s, giải mã 22 MB/s.

### 2.1.3. Xây dựng và tối ưu xử lý mã hóa giải mã trên GPU

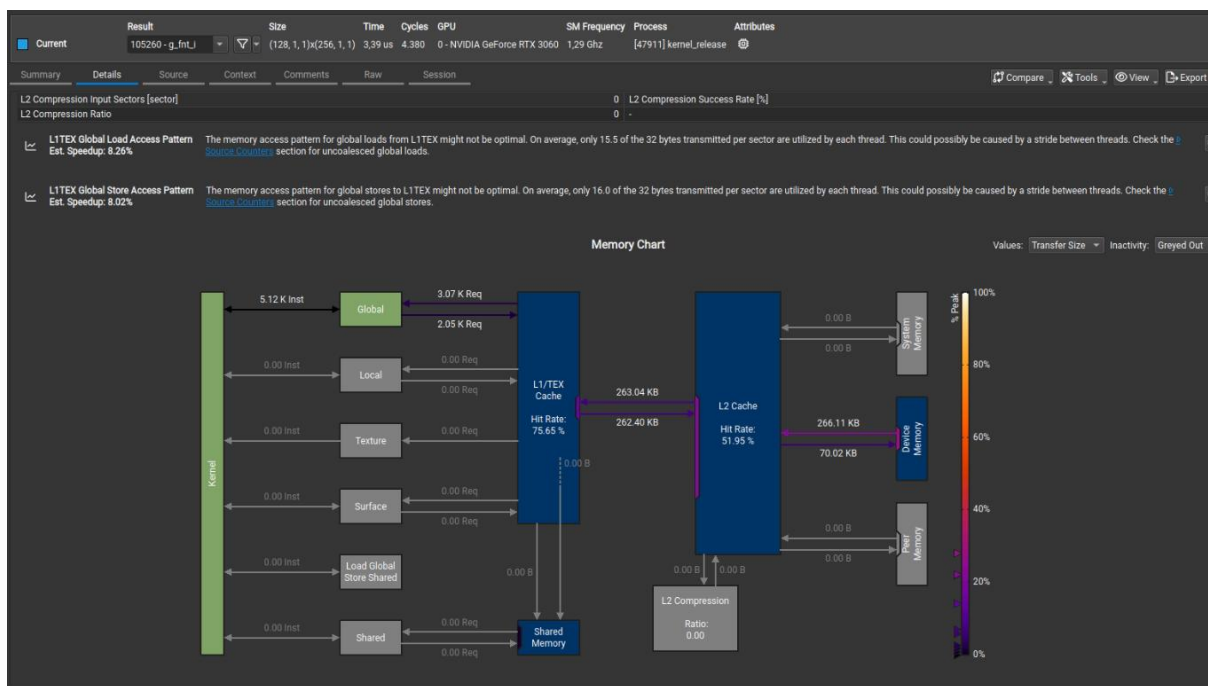
Trên GPU quá trình tối ưu thuật toán song song trở nên khó khăn hơn nhiều. Với cài đặt như lý thuyết, hệ thống chỉ có thể đạt được tốc độ mã hóa 100Mb/s và giải mã 7 Mb/s. Các cải tiến được đề cập sau đây chỉ là một phần nhỏ đã được thử nghiệm và có

hiệu quả tốt nhất, có quá nhiều thử nghiệm không hiệu quả nên không thể đề cập hết. Chiến thuật tối ưu thuật toán hiệu quả trên GPU là cài đặt theo hướng dẫn của các tài liệu chính thức về Cuda, kèm với đó là liên tục đo lường bằng các công cụ được hỗ trợ bởi Cuda (Nsight System, Nsight Compute) để tối ưu cài đặt.

Trên GPU, kích thước cache mỗi luồng rất nhỏ. Cụ thể với RTX 3060, mỗi SM chỉ có 128kb cache L1 trong khi chứa tới 48 Warp là 1536 luồng phần cứng, vậy mỗi luồng chỉ có 85-byte cache. Trong RTX 3060, một dòng cache (cache line) là 128-byte, nếu cách truy cập bộ nhớ không được thiết kế để tối ưu cho cache thì việc có lấp đầy các Warp cũng không đem lại nhiều lợi ích vì khi đổi qua Warp mới lệnh đọc có thể làm mất dữ liệu trong cache của Warp cũ. Cách truy cập bộ nhớ tối ưu nhất là đảm bảo cho các luồng truy cập các vị trí bộ nhớ gần nhau tại một thời điểm. Cụ thể, giả sử một công việc chạy trên N luồng, có M phép tính trên mảng M phần tử, vậy luồng thứ i nên lặp và tính phép tính trên mảng ở các vị trí i, i + N, i + 2N... cho đến M, vì trong mỗi lần lặp N luồng luôn đọc N vị trí gần nhau nên sẽ tận dụng tốt được cache.



Hình 2.1 Cách truy cập bộ nhớ tối ưu, đảm bảo các luồng truy cập bộ nhớ cạnh nhau ở mỗi bước

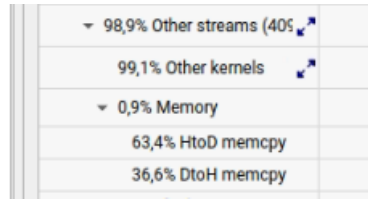


Hình 2.2 Tỷ lệ cache hit L1, L2 sau khi truy cập bộ nhớ theo cách trên, đo bằng công cụ Nsight Compute

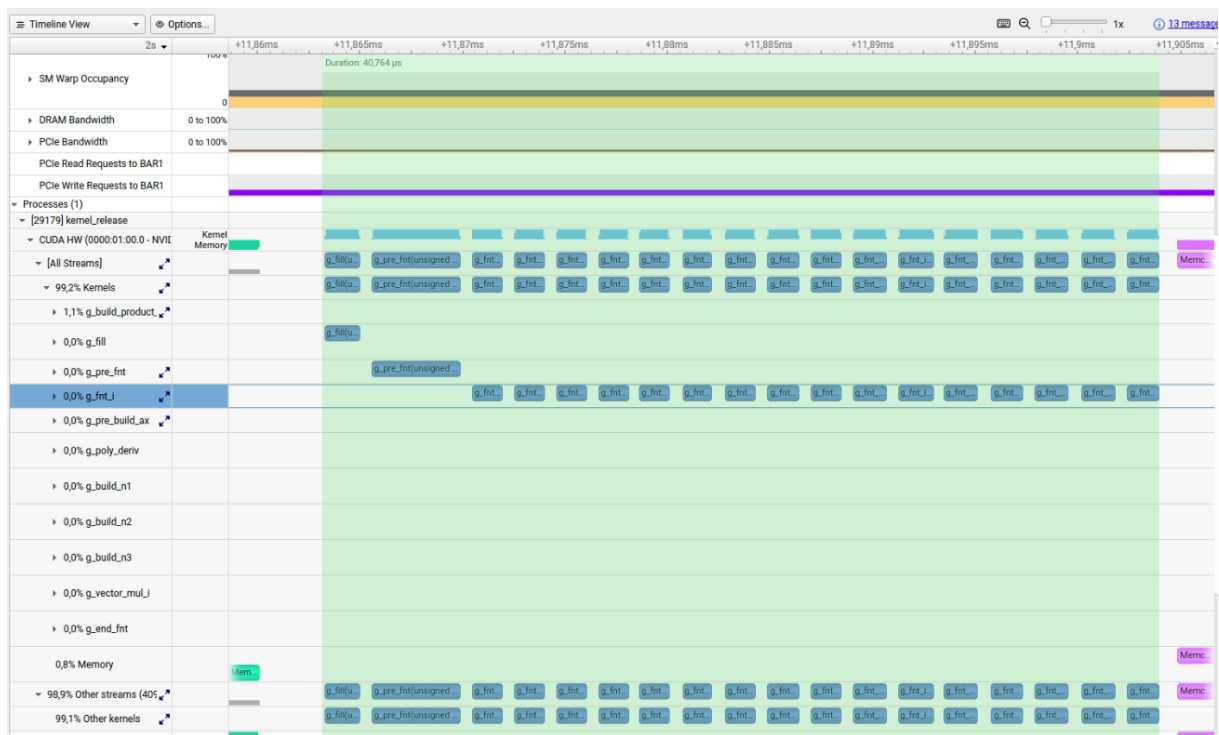
Trên GPU, tốc độ sao chép và cấp phát bộ nhớ là một vấn đề lớn, làm giới hạn hiệu năng thực tế của GPU. Cuda có hỗ trợ các lệnh xử lý bất đồng bộ giữa Host và Device để giúp hạn chế thời gian chết nhưng cần sử dụng bộ nhớ không phân trang. Cụ thể, trong máy tính hiện đại, hệ điều hành có thể lưu các biến của chương trình đang chạy trên RAM hoặc chỉ lưu địa chỉ của vùng nhớ đó trên RAM còn biến lưu xuống đĩa, cách làm này có nhiều lợi ích như hạn chế sự phân mảnh và giúp hệ điều hành có thể chạy nhiều chương trình cùng lúc hơn. Khi muốn sao chép dữ liệu giữa CPU và GPU bất đồng bộ thì GPU sẽ được phép tự truy cập vào bộ nhớ RAM để lấy dữ liệu (DMA), tuy nhiên với vùng nhớ phân trang thì địa chỉ CPU đưa cho GPU lúc đầu có thể không đúng nữa, vậy nên cần bộ nhớ không phân trang. Việc cấp phát vùng nhớ không phân trang không phải đơn giản, tốn rất nhiều thời gian.

Vậy nên trong hệ thống sẽ quy định sẵn số lượng mảnh mã hóa giải mã tối đa có thể xử lý cùng lúc rồi cấp phát bộ nhớ từ đầu sau đó tái sử dụng các vùng nhớ đó. Các luồng muốn mã hóa giải mã trước hết phải đăng ký với hệ thống và đợi để được phát vé quy định vị trí các vùng nhớ để sử dụng, luồng mã hóa giải mã sau khi chạy xong phải trả lại vé cho hệ thống để các luồng khác sử dụng, sẽ có 2 loại vé cho mã hóa và giải mã. Một vé sẽ quy định vị trí vùng nhớ không phân trang trên CPU và các vùng nhớ cần có để chạy thuật toán tùy theo loại trên GPU. Khi một luồng có vé thì phải sao chép dữ liệu đầu vào của thuật toán vào vùng nhớ không phân trang dành cho đầu vào theo vé,

sau đó chạy thuật toán, thuật toán chỉ được sử dụng các vùng nhớ GPU theo vẽ, sau khi chạy xong thì phải sao chép dữ liệu đầu ra vào vùng nhớ không phân trang trên CPU dành cho đầu ra theo vẽ, cuối cùng thì trả lại vé cho hệ thống.



Hình 2.3 Tỷ lệ thời gian sao chép dữ liệu giữa CPU và GPU so với cả hệ thống, đo bằng công cụ Nsight System



Hình 2.4 Thời gian chạy của một hàm FNT chỉ còn 40 micro giây, đo bằng công cụ Nsight System



gần như tương đương với việc gọi hàm trong cùng chương trình nếu không cần sao chép dữ liệu.

Xử lý đa luồng trong hệ thống cũng là phần quan trọng, cụ thể là phần xử lý quản lý các vé chứa thông tin vùng nhớ được sử dụng trong thuật toán và phần xử lý kết hợp lại file gốc ở bên nhận. Các phần trên đều cần xử lý đa luồng để tận dụng được khả năng xử lý của CPU về sau nhưng lại bị chặn (Blocking) nhiều ở giai đoạn đầu. Cụ thể các luồng chạy thuật toán khi đã hết vé phải đợi các luồng thuật toán khác hoàn trả vé rồi mới xử lý tiếp được, các luồng kết hợp file khi đến phải đợi một luồng nào đó khởi tạo vị trí lưu file trong bộ nhớ, đăng ký với hệ thống và sinh ra luồng theo dõi để kiểm tra tiến trình file và giải phóng bộ nhớ sau cùng. Java từ phiên bản 21 trở đi có hỗ trợ tính năng luồng ảo (Virtual Thread), có thể hiểu tương tự như TBB trong phần thuật toán CPU, so với TBB thì luồng ảo của Java là dạng luồng người dùng (User Thread) tập trung xử lý các công việc bị chặn nhiều hiệu quả, cùng với đó là tự động sinh thêm luồng hệ điều hành (OS Thread) khi cần CPU.

Theo dõi và cập nhật tình trạng gửi và nhận file hiệu quả cũng là vấn đề cần giải quyết, cụ thể cần cập nhập và theo dõi các biến lưu tình trạng của file hiệu quả. Phương pháp cơ bản là dùng khóa (Lock) để đảm bảo an toàn trong môi trường đa luồng chỉ tốt nếu luồng bị chặn lâu, vì dùng khóa sẽ khiến luồng hiện tại chuyển sang trạng thái đợi và chờ đến khi tới lượt lấy khóa, quá trình này cực kì tốn kém trên luồng hệ điều hành (có thể cần context switch) và cho dù đỡ tốn hơn trên luồng ảo Java nhưng vẫn cần hạn chế. Hầu như mọi CPU hiện đại (từ 2013) đều hỗ trợ các câu lệnh CAS (compare-and-swap), các câu lệnh CAS giúp tạo ra các kiểu dữ liệu có thể đọc và cập nhật nhanh và an toàn trong môi trường đa luồng, trong Java có hỗ trợ sử dụng các câu lệnh này. Bản chất của CAS tương tự như dùng khóa xoay (spin lock), tận dụng việc các câu lệnh đọc và cập nhật trên các kiểu dữ liệu này rất nhanh, câu lệnh CAS sẽ liên tục thử so sánh và cập nhật đến khi thành công, vì các câu lệnh này nhanh nên số lần thử đến khi thành công sẽ thấp. Các câu lệnh CAS được cài đặt ở tầng CPU nên chi phí cho mỗi câu lệnh rất thấp, chỉ hơn 15% so với một câu lệnh đọc dữ liệu bị cache miss và không làm luồng bị chặn.

## CHƯƠNG 3: KẾT QUẢ VÀ KIỂM THỬ

### 3.1. Kết quả

Hoàn thành được hệ thống, phần ứng dụng chạy trên màn hình console gồm một tính năng chính là gửi file:

```
Send port: 6969
Receive port: 8888
OK.
Save path: /home/captain3060/Desktop/Projects/PBL7_DUT_ECC_CUDA/rs_app/received/
Setup: -s {destination_ip} {destination_port}
Send: -f {file_path}
Quit: -q
Init process completed!
-s 127.0.0.1 8888
OK.
-f /home/captain3060/Downloads/code_1.100.2-1747260578_amd64.deb
File code_1.100.2-1747260578_amd64.deb will be send as id 7c875e0f-a892-443b-be45-b76eca44d369
File 7c875e0f-a892-443b-be45-b76eca44d369 send completed.
File 7c875e0f-a892-443b-be45-b76eca44d369 original name is code_1.100.2-1747260578_amd64.deb
File 7c875e0f-a892-443b-be45-b76eca44d369 receive completed.
```

Hình 3.1 Giao diện console cơ bản của hệ thống

Người quản trị hệ thống cần bật hệ thống lên ở cả máy chủ nguồn và đích, làm theo các hướng dẫn của hệ thống. Đầu tiên là chọn cổng nhận file và cổng gửi file, như trên hình chọn cổng 6969 để gửi và 8888 để nhận. Sau đó người quản trị có thể cài đặt địa chỉ muốn gửi file đến, như trên hình chọn 127.0.0.1 và cổng 8888 là muốn gửi nội bộ trong máy, từ 6969 đến 8888. Cuối cùng người quản trị có thể gửi file, địa chỉ sẽ là địa chỉ gần nhất được cài đặt, nếu chưa cài đặt địa chỉ đích thì hệ thống sẽ thông báo yêu cầu cài đặt và bỏ qua, tương tự với lỗi cú pháp hệ thống cũng sẽ thông báo và bỏ qua. Sau khi chạy lệnh gửi file sẽ có thông báo file gửi thành công, bên nhận khi đã nhận đủ dữ liệu để tái tạo file thì sẽ lưu ở địa chỉ được thiết lập sẵn như trên và có thông báo, như trên vì gửi cùng máy nên sẽ có cả thông báo gửi thành công và nhận thành công.

Lưu ý rằng người quản trị hệ thống phải đảm bảo file gửi không quá một nửa thông lượng của mạng vì hệ thống hiện tại không có tính năng kiểm soát thông lượng gửi đi, cùng với đó thiết lập kích thước bộ nhớ đệm gấp đôi kích thước file gửi. Phần thông lượng mạng có thể dựa vào thông lượng lý thuyết của nhà mạng cung cấp. Phần bộ nhớ đệm có thể thiết lập bằng câu lệnh sau (1G cho nhận và 1G cho gửi):



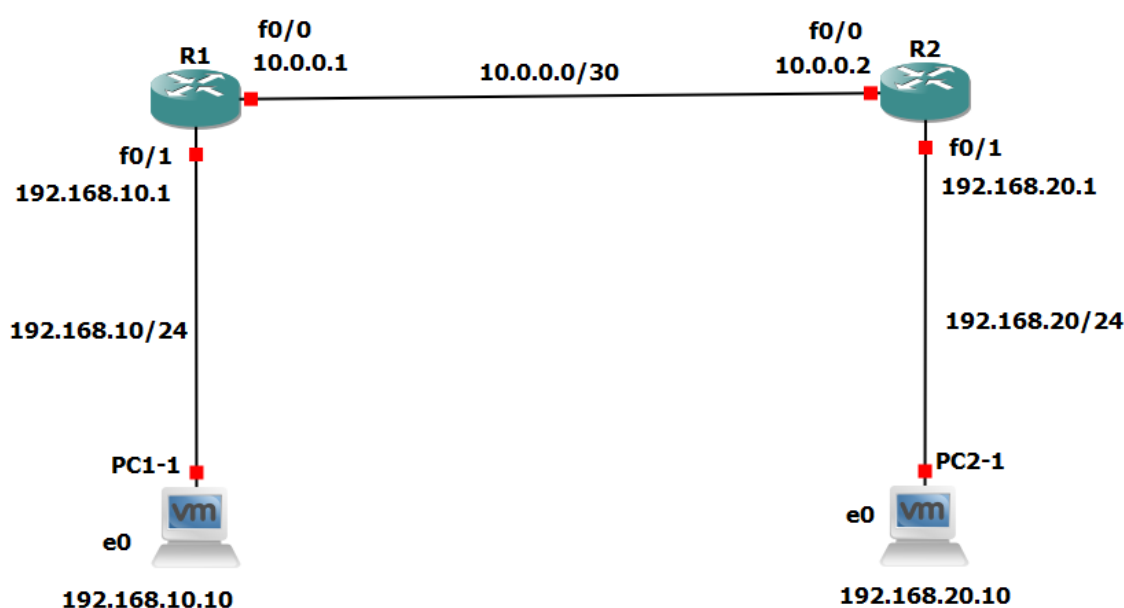
```
sudo sysctl -w net.core.rmem_max=1073741824
sudo sysctl -w net.core.rmem_default=1073741824
sudo sysctl -w net.core.wmem_max=1073741824
sudo sysctl -w net.core.wmem_default=1073741824
```

Hình 3.2 Lệnh tăng kích thước bộ nhớ đệm để lưu các gói tin đến và đi

## 3.2. Kiểm thử hệ thống

### 3.2.1. Kiểm thử về khả năng chống chịu lỗi

Để mô phỏng mô hình mạng và tỉ lệ mất lỗi gói, sử dụng công cụ GNS3, mô hình mạng sẽ xây dựng đơn giản như sau:



Hình 3.3 Mô hình mạng cơ bản trên GNS3

Máy PC1-1 và PC2-1 trên sẽ là các máy ảo ubuntu 22.04 cấu hình 1 core 2GB ram, được liên kết với GNS3, có thể sử dụng mạng trong GNS3. Các router R1 và R2 được cấu hình để có thể chịu được thông lượng 10MB/s, tuy nhiên mạng mô phỏng trong GNS3 sẽ yếu hơn chỉ số được cấu hình.

Cụ thể khi đo nhanh bằng công cụ iperf3, thử gửi UDP 1 lần với thông lượng 10 MB/s, mỗi gói 1024 byte, tỉ lệ mất đã là 4%.

```
pc@pc-virtual-machine:~$ iperf3 -c 192.168.20.10 -u -b 80M -l 1024 -t 1
Connecting to host 192.168.20.10, port 5201
[ 5] local 192.168.10.10 port 56533 connected to 192.168.20.10 port 5201
[ ID] Interval           Transfer     Bitrate        Total Datagrams
[ 5]  0.00-1.00      sec    9.53 MBytes    79.9 Mbits/sec    9754
- - - - -
[ ID] Interval           Transfer     Bitrate        Jitter    Lost/Total Datagrams
[ 5]  0.00-1.00      sec    9.53 MBytes    79.9 Mbits/sec    0.000 ms    0/9754 (0%) sender
[ 5]  0.00-10.37     sec    9.14 MBytes    7.39 Mbits/sec    0.949 ms   390/9751 (4%) receiver
iperf Done.
```

Hình 3.4 Thử gửi một lần 10 MB/s dùng công cụ iperf3

Thử gửi 100 lần với thông lượng chỉ 1MB/s tỉ lệ mất gói là 0%.

```
pc@pc-virtual-machine:~$ iperf3 -c 192.168.20.10 -u -b 8M -l 1024 -t 100
Connecting to host 192.168.20.10, port 5201
[ 5] local 192.168.10.10 port 55955 connected to 192.168.20.10 port 5201
[ ID] Interval           Transfer     Bitrate        Total Datagrams
[ 5]  0.00-1.00      sec    976 KBytes    7.99 Mbits/sec    976
[ 5]  1.00-2.00      sec    977 KBytes    8.00 Mbits/sec    977
[ 5]  2.00-3.00      sec    976 KBytes    8.00 Mbits/sec    976
[ 5]  3.00-4.00      sec    977 KBytes    8.00 Mbits/sec    977
[ 5]  4.00-5.00      sec    976 KBytes    8.00 Mbits/sec    976
[ 5]  5.00-6.00      sec    977 KBytes    8.00 Mbits/sec    977
[ 5]  6.00-7.00      sec    977 KBytes    8.00 Mbits/sec    977
[ 5]  7.00-8.00      sec    976 KBytes    8.00 Mbits/sec    976
[ 5]  8.00-9.00      sec    977 KBytes    8.00 Mbits/sec    977
[ 5]  9.00-10.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 10.00-11.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 11.00-12.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 12.00-13.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 13.00-14.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 14.00-15.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 15.00-16.00     sec    977 KBytes    8.00 Mbits/sec    977
```

Hình 3.5 Thử gửi 100 lần thông lượng 1 MB/s, phần đầu, dùng công cụ iperf3

```
[ 5] 88.00-89.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 89.00-90.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 90.00-91.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 91.00-92.00     sec    976 KBytes    7.99 Mbits/sec    976
[ 5] 92.00-93.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 93.00-94.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 94.00-95.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 95.00-96.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 96.00-97.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 97.00-98.00     sec    977 KBytes    8.00 Mbits/sec    977
[ 5] 98.00-99.00     sec    976 KBytes    8.00 Mbits/sec    976
[ 5] 99.00-100.00    sec    977 KBytes    8.00 Mbits/sec    977
- - - - -
[ ID] Interval           Transfer     Bitrate        Jitter    Lost/Total Datagrams
[ 5]  0.00-100.00    sec    95.4 MBytes    8.00 Mbits/sec    0.000 ms    0/97656 (0%) sender
[ 5]  0.00-100.08    sec    95.4 MBytes    7.99 Mbits/sec    1.524 ms   0/97641 (0%) receiver
```

Hình 3.6 Thử gửi 100 lần thông lượng 1 MB/s, phần sau, dùng công cụ iperf3

Thử 30 lần với thông lượng 2 MB/s tỉ lệ mất gói đã lên đến 34%.

```
pc@pc-virtual-machine:~$ iperf3 -c 192.168.20.10 -u -b 16M -l 1024 -t 30
Connecting to host 192.168.20.10, port 5201
[ 5] local 192.168.10.10 port 37173 connected to 192.168.20.10 port 5201
[ ID] Interval            Transfer           Bitrate           Total Datagrams
[ 5] 0.00-1.00      sec  1.91 MBytes      16.0 Mbits/sec     1952
[ 5] 1.00-2.00      sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 2.00-3.00      sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 3.00-4.00      sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 4.00-5.00      sec  1.91 MBytes      16.0 Mbits/sec     1952
[ 5] 5.00-6.00      sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 6.00-7.00      sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 7.00-8.00      sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 8.00-9.00      sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 9.00-10.00     sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 10.00-11.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 11.00-12.00    sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 12.00-13.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 13.00-14.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 14.00-15.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 15.00-16.00    sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 16.00-17.00    sec  1.91 MBytes      16.0 Mbits/sec     1952
[ 5] 17.00-18.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 18.00-19.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 19.00-20.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 20.00-21.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 21.00-22.00    sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 22.00-23.00    sec  1.91 MBytes      16.0 Mbits/sec     1952
[ 5] 23.00-24.00    sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 24.00-25.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 25.00-26.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 26.00-27.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 27.00-28.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
[ 5] 28.00-29.00    sec  1.91 MBytes      16.0 Mbits/sec     1954
[ 5] 29.00-30.00    sec  1.91 MBytes      16.0 Mbits/sec     1953
- - - - -
[ ID] Interval            Transfer           Bitrate           Jitter      Lost/Total Datagrams
[ 5] 0.00-30.00      sec  57.2 MBytes      16.0 Mbits/sec     0.000 ms    0/58593 (0%) sender
[ 5] 0.00-40.79      sec  37.8 MBytes      7.78 Mbits/sec     1.597 ms    19738/58487 (34%) receiver
```

Hình 3.7 Thử gửi 30 lần thông lượng 2 MB/s dùng công cụ iperf3

Chuyển sang bên người nhận, các lần gửi cuối bắt đầu có hiện tượng mất gói nhiều, kết luận đó là do tắc nghẽn vì đã quá thông lượng thực tế của GNS3.

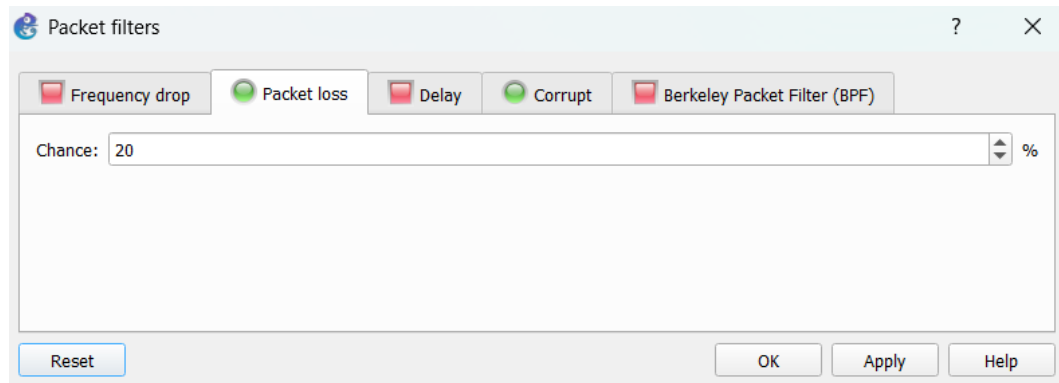
ID	Interval	Transfer	Bitrate	Jitter	Lost/Total Datagrams
[ 5]	0.00-1.00	sec 928 KBytes	7.60 Mb/s	1.931 ms	0/928 (0%)
[ 5]	1.00-2.00	sec 1.01 MBytes	8.50 Mb/s	1.072 ms	0/1037 (0%)
[ 5]	2.00-3.00	sec 1008 KBytes	8.26 Mb/s	1.061 ms	0/1008 (0%)
[ 5]	3.00-4.00	sec 1.00 MBytes	8.37 Mb/s	2.213 ms	0/1025 (0%)
[ 5]	4.00-5.00	sec 1023 KBytes	8.40 Mb/s	1.009 ms	0/1023 (0%)
[ 5]	5.00-6.00	sec 1.02 MBytes	8.52 Mb/s	1.066 ms	0/1040 (0%)
[ 5]	6.00-7.00	sec 1.00 MBytes	8.39 Mb/s	0.993 ms	0/1024 (0%)
[ 5]	7.00-8.00	sec 1007 KBytes	8.24 Mb/s	1.882 ms	0/1007 (0%)
[ 5]	8.00-9.00	sec 1.01 MBytes	8.53 Mb/s	1.046 ms	0/1039 (0%)
[ 5]	9.00-10.00	sec 1.00 MBytes	8.39 Mb/s	0.996 ms	0/1024 (0%)
[ 5]	10.00-11.00	sec 979 KBytes	8.02 Mb/s	1.120 ms	0/979 (0%)
[ 5]	11.00-12.00	sec 951 KBytes	7.79 Mb/s	1.015 ms	0/951 (0%)
[ 5]	12.00-13.00	sec 952 KBytes	7.80 Mb/s	1.045 ms	0/952 (0%)
[ 5]	13.00-14.00	sec 940 KBytes	7.70 Mb/s	1.142 ms	0/940 (0%)
[ 5]	14.00-15.00	sec 952 KBytes	7.80 Mb/s	0.985 ms	0/952 (0%)
[ 5]	15.00-16.00	sec 940 KBytes	7.70 Mb/s	1.664 ms	0/940 (0%)
[ 5]	16.00-17.00	sec 936 KBytes	7.67 Mb/s	1.079 ms	0/936 (0%)
[ 5]	17.00-18.00	sec 952 KBytes	7.80 Mb/s	1.032 ms	0/952 (0%)
[ 5]	18.00-19.00	sec 940 KBytes	7.70 Mb/s	1.312 ms	0/940 (0%)
[ 5]	19.00-20.00	sec 936 KBytes	7.67 Mb/s	1.032 ms	0/936 (0%)
[ 5]	20.00-21.00	sec 952 KBytes	7.80 Mb/s	0.984 ms	475/1427 (33%)
[ 5]	21.00-22.00	sec 956 KBytes	7.83 Mb/s	2.628 ms	1111/2067 (54%)
[ 5]	22.00-23.00	sec 936 KBytes	7.67 Mb/s	1.219 ms	958/1894 (51%)
[ 5]	23.00-24.00	sec 953 KBytes	7.80 Mb/s	1.962 ms	958/1911 (50%)
[ 5]	24.00-25.00	sec 940 KBytes	7.70 Mb/s	3.117 ms	1100/2040 (54%)
[ 5]	25.00-26.00	sec 967 KBytes	7.93 Mb/s	0.976 ms	945/1912 (49%)
[ 5]	26.00-27.00	sec 940 KBytes	7.70 Mb/s	4.349 ms	1117/2057 (54%)
[ 5]	27.00-28.00	sec 952 KBytes	7.80 Mb/s	1.075 ms	940/1892 (50%)
[ 5]	28.00-29.00	sec 952 KBytes	7.80 Mb/s	1.049 ms	955/1907 (50%)
[ 5]	29.00-30.00	sec 948 KBytes	7.76 Mb/s	2.291 ms	1122/2070 (54%)
[ 5]	30.00-31.00	sec 908 KBytes	7.44 Mb/s	1.795 ms	949/1857 (51%)
[ 5]	31.00-32.00	sec 904 KBytes	7.41 Mb/s	1.118 ms	931/1835 (51%)
[ 5]	32.00-33.00	sec 888 KBytes	7.27 Mb/s	1.082 ms	949/1837 (52%)
[ 5]	33.00-34.00	sec 892 KBytes	7.31 Mb/s	1.241 ms	958/1850 (52%)
[ 5]	34.00-35.00	sec 888 KBytes	7.27 Mb/s	1.588 ms	954/1842 (52%)
[ 5]	35.00-36.00	sec 876 KBytes	7.18 Mb/s	2.730 ms	940/1816 (52%)
[ 5]	36.00-37.00	sec 904 KBytes	7.41 Mb/s	1.443 ms	938/1842 (51%)
[ 5]	37.00-38.00	sec 876 KBytes	7.18 Mb/s	2.182 ms	943/1819 (52%)
[ 5]	38.00-39.00	sec 904 KBytes	7.41 Mb/s	1.630 ms	944/1848 (51%)
[ 5]	39.00-40.00	sec 888 KBytes	7.27 Mb/s	1.339 ms	929/1817 (51%)
[ 5]	40.00-40.79	sec 692 KBytes	7.14 Mb/s	1.597 ms	622/1314 (47%)
-----					
ID	Interval	Transfer	Bitrate	Jitter	Lost/Total Datagrams
[ 5]	0.00-40.79	sec 37.8 MBytes	7.78 Mb/s	1.597 ms	19738/58487 (34%) receiver

Hình 3.8 Thử gửi 30 lần thông lượng 2 MB/s phân bên người gửi

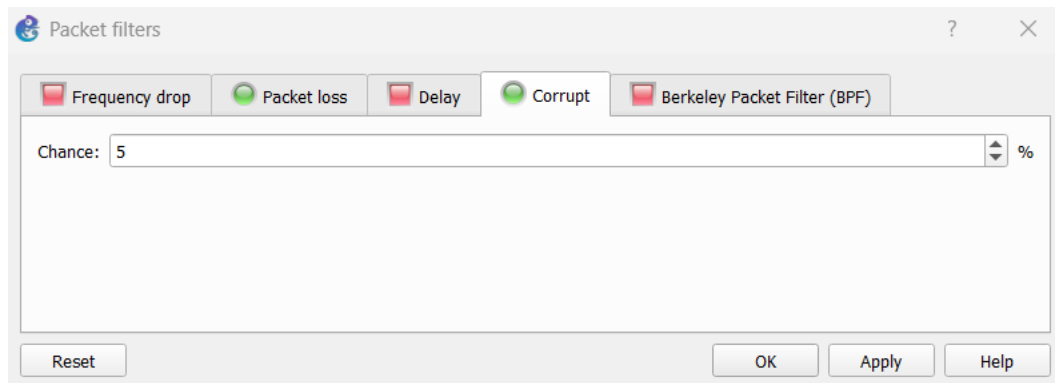
Sau các thử nghiệm trên thì xác định được thông lượng thực sự của mạng chỉ khoảng 1 MB/s (8 Kbit/s), đó mới chỉ là đo lường gửi các gói tin, khi gửi file thì cho dù chỉ mất một mảnh nhỏ đã không thể tái tạo lại được file gốc. Vậy các thử nghiệm của hệ thống sẽ chỉ thực hiện với file khoảng 500 KB.

Thử gửi file khoảng 525 KB bằng hệ thống 30 lần, tỉ lệ lỗi file 0%. Thử chỉnh tỉ lệ mất gói lên 20%, tỉ lệ lỗi gói lên 5%, thử lại 30 lần, tỉ lệ lỗi file vẫn 0%.

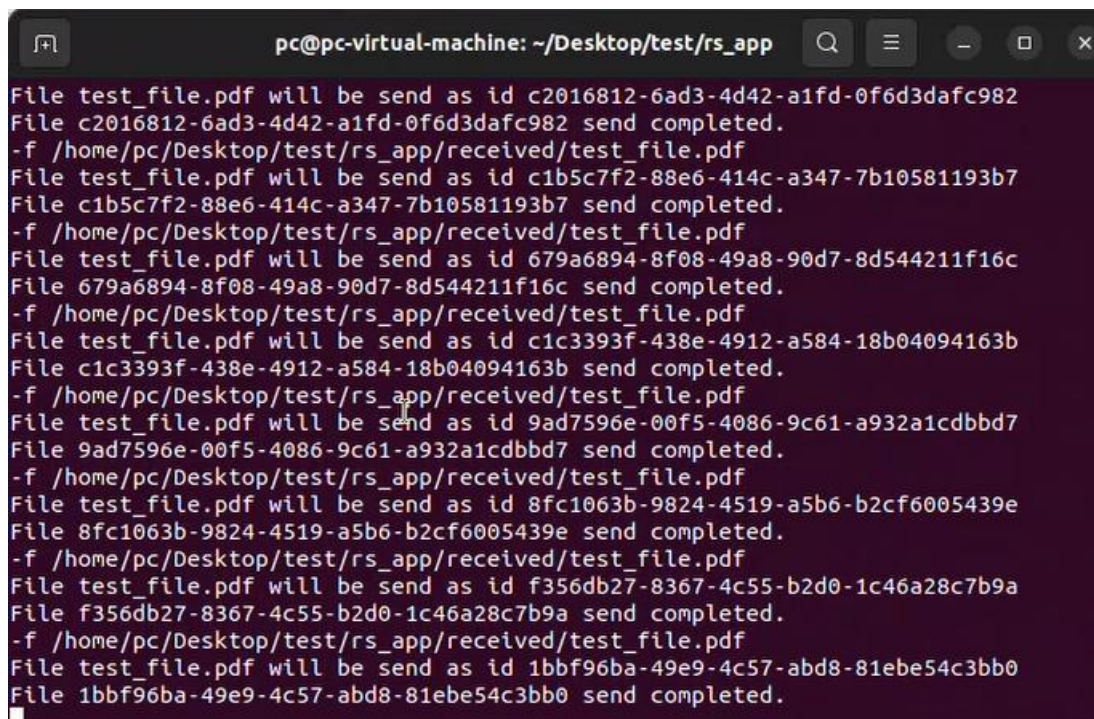




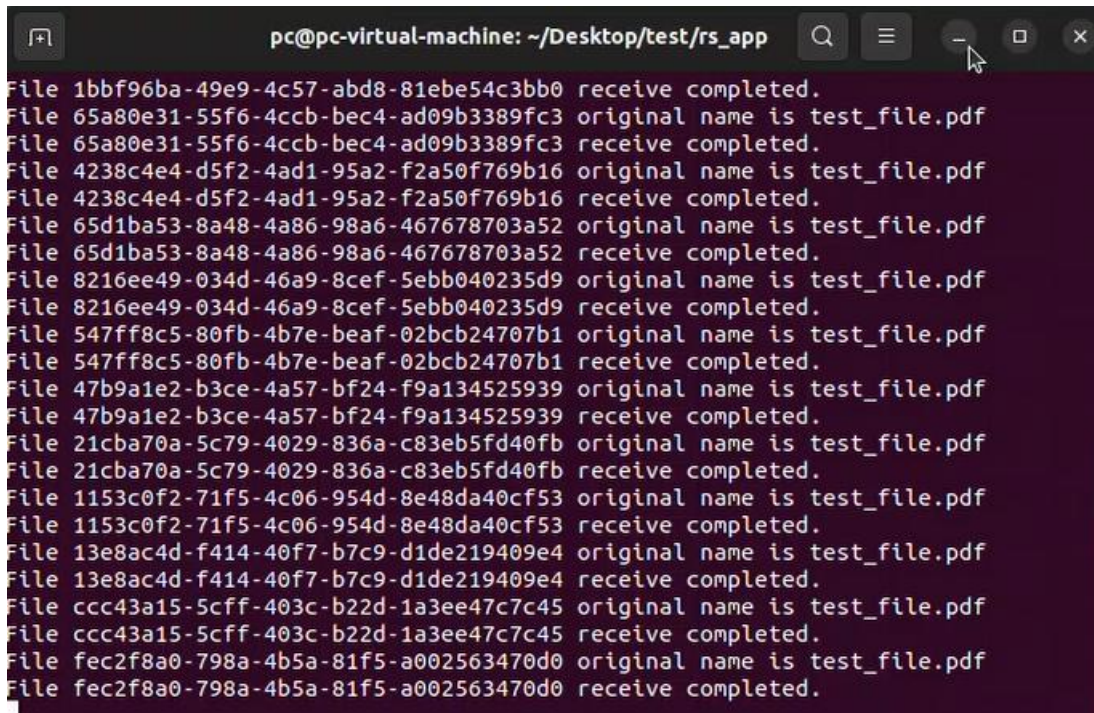
Hình 3.9 Chỉnh tỉ lệ mất gói mạng thành 20% trong GNS3



Hình 3.10 Chỉnh tỉ lệ lỗi gói mạng thành 5% trong GNS3



Hình 3.11 Gửi file 500 KB 30 lần bằng hệ thống



```
pc@pc-virtual-machine: ~/Desktop/test/rs_app
File 1bbf96ba-49e9-4c57-abd8-81ebe54c3bb0 receive completed.
File 65a80e31-55f6-4ccb-bec4-ad09b3389fc3 original name is test_file.pdf
File 65a80e31-55f6-4ccb-bec4-ad09b3389fc3 receive completed.
File 4238c4e4-d5f2-4ad1-95a2-f2a50f769b16 original name is test_file.pdf
File 4238c4e4-d5f2-4ad1-95a2-f2a50f769b16 receive completed.
File 65d1ba53-8a48-4a86-98a6-467678703a52 original name is test_file.pdf
File 65d1ba53-8a48-4a86-98a6-467678703a52 receive completed.
File 8216ee49-034d-46a9-8cef-5ebb040235d9 original name is test_file.pdf
File 8216ee49-034d-46a9-8cef-5ebb040235d9 receive completed.
File 547ff8c5-80fb-4b7e-beaf-02bcb24707b1 original name is test_file.pdf
File 547ff8c5-80fb-4b7e-beaf-02bcb24707b1 receive completed.
File 47b9a1e2-b3ce-4a57-bf24-f9a134525939 original name is test_file.pdf
File 47b9a1e2-b3ce-4a57-bf24-f9a134525939 receive completed.
File 21cba70a-5c79-4029-836a-c83eb5fd40fb original name is test_file.pdf
File 21cba70a-5c79-4029-836a-c83eb5fd40fb receive completed.
File 1153c0f2-71f5-4c06-954d-8e48da40cf53 original name is test_file.pdf
File 1153c0f2-71f5-4c06-954d-8e48da40cf53 receive completed.
File 13e8ac4d-f414-40f7-b7c9-d1de219409e4 original name is test_file.pdf
File 13e8ac4d-f414-40f7-b7c9-d1de219409e4 receive completed.
File ccc43a15-5cff-403c-b22d-1a3ee47c7c45 original name is test_file.pdf
File ccc43a15-5cff-403c-b22d-1a3ee47c7c45 receive completed.
File fec2f8a0-798a-4b5a-81f5-a002563470d0 original name is test_file.pdf
File fec2f8a0-798a-4b5a-81f5-a002563470d0 receive completed.
```

Hình 3.12 Nhận file 500 KB 30 lần bằng hệ thống

Thử đo lại tỉ lệ lỗi với iperf3 trên mạng sau khi đã chỉnh, cũng gửi với 500KB 30 lần, tỉ lệ lỗi tương đương với tỉ lệ lỗi được mô phỏng bởi GNS3 là 27%.

```
pc@pc-virtual-machine:~/Desktop$ iperf3 -c 192.168.20.10 -u -b 4096K -l 1024 -t 30
Connecting to host 192.168.20.10, port 5201
[ 5] local 192.168.10.10 port 53069 connected to 192.168.20.10 port 5201
[ ID] Interval            Transfer        Bitrate          Total Datagrams
[ 5] 0.00-1.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 1.00-2.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 2.00-3.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 3.00-4.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 4.00-5.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 5.00-6.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 6.00-7.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 7.00-8.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 8.00-9.00 sec        500 KBytes     4.10 Mbits/sec   500
[ 5] 9.00-10.00 sec       500 KBytes     4.10 Mbits/sec   500
[ 5] 10.00-11.00 sec      500 KBytes     4.10 Mbits/sec   500
[ 5] 11.00-12.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 12.00-13.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 13.00-14.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 14.00-15.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 15.00-16.00 sec     500 KBytes     4.08 Mbits/sec   500
[ 5] 16.00-17.00 sec     500 KBytes     4.11 Mbits/sec   500
[ 5] 17.00-18.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 18.00-19.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 19.00-20.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 20.00-21.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 21.00-22.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 22.00-23.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 23.00-24.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 24.00-25.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 25.00-26.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 26.00-27.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 27.00-28.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 28.00-29.00 sec     500 KBytes     4.10 Mbits/sec   500
[ 5] 29.00-30.00 sec     500 KBytes     4.10 Mbits/sec   500
-----
[ ID] Interval            Transfer        Bitrate          Jitter    Lost/Total Datagrams
[ 5] 0.00-30.00 sec      14.6 MBytes     4.10 Mbits/sec   0.000 ms  0/15000 (0%) sender
[ 5] 0.00-30.00 sec      10.7 MBytes     3.00 Mbits/sec   3.332 ms 3993/14991 (27%) receiver

iperf Done.
```

Hình 3.13 Đo lại tỉ lệ lỗi mạng sau khi đã chỉnh GNS3 dùng iperf3

Vậy đã chứng minh được hệ thống có khả năng chịu lỗi tốt trong môi trường có thông lượng mạng phù hợp, tỉ lệ lỗi ngẫu nhiên.

### 3.2.2. Kiểm thử về thông lượng

Thông lượng của thuật toán mã hóa giải mã được tính bằng cách sinh các test ngẫu nhiên, cho thuật toán chạy và ghi lại thời gian, từ đó suy ra thông lượng, cuối cùng sẽ có trình chấm để kiểm tra kết quả chạy của thuật toán. Kết quả như đã đề cập ở những phần trên, bản CPU có thông lượng mã hóa 400 MB/s giải mã 20 MB/s, bản GPU có thông lượng mã hóa 1.1 GB/s giải mã 60 MB/s.

Thông lượng của cả hệ thống sẽ được đo bằng cách gửi file trong trường hợp lý tưởng là mạng có thông lượng cao và bộ nhớ đệm cao, cụ thể là gửi trong cùng máy từ cổng này sang cổng khác và cấp cho bộ nhớ đệm 1G gửi 1G nhận. Hiện tại chưa có phần kiểm thử chính xác về thông lượng của hệ thống nhưng qua trải nghiệm thực tế thì ở bản GPU hệ thống thông lượng cực kì sát với thuật toán, phần CPU thì chậm hơn khoảng

50%. Video quay lại trải nghiệm khi gửi nhận file 100MB trong cùng một máy từ cổng này qua cổng khác, bản GPU, cấu hình máy là i5 12400 và RTX 3060 [12].



## KẾT LUẬN

Kết quả và đóng góp:

- Đã xây dựng thành công hệ thống truyền thông một chiều với khả năng chịu lỗi và thông lượng tốt. Phát triển nền tảng cơ bản để tập đoàn EVN và các đối tượng có nhu cầu truyền thông một chiều tin cậy có thể nghiên cứu áp dụng.
- Đánh giá cơ bản về khả năng xử lý song song của phần cứng hiện đại. CPU hiện đại có khả năng xử lý song song rất tốt và cấu trúc bộ nhớ mỗi nhân nhiều, dễ dàng cài đặt được thuật toán song song tối ưu. GPU hiện đại có khả năng xử lý song song tốt hơn CPU nhưng tốt hơn bao nhiêu phụ thuộc vào hai yếu tố, loại phép tính và nhu cầu sử dụng bộ nhớ. Cụ thể GPU hiện đại được thiết kế để tính nhanh các phép tính dấu phẩy động 32-bit (FP32) và bị nghẽn cổ chai chính là ở thông lượng bộ nhớ.

Hạn chế và hướng phát triển:

- Bài toán lúc đầu mục tiêu là hướng đến giải quyết vấn đề của tập toàn EVN nhưng khi phát triển lại tập trung vào bài toán tổng quát gửi file một chiều tin cậy trên môi trường mạng nên cũng không quá sát với bài toán của tập đoàn.
- Hệ thống vẫn còn thiếu một số tính năng nhỏ như giới hạn thông lượng, gửi file đến nhiều đối tượng cùng lúc (multicast). Giao diện của hệ thống vẫn chưa quá trực quan.
- Phần thuật toán song song trên GPU cũng cần thời gian nghiên cứu để tối ưu thêm, hiện tại chỉ nhanh khoảng gấp 3 lần bản CPU cũng chưa phải kết quả quá ấn tượng so với chi phí bỏ ra.

## TÀI LIỆU THAM KHẢO

- [1] A. Soro and J. Lacan, “FNT-based Reed–Solomon erasure codes” in *2010 7th IEEE Consumer Communications and Networking Conference*, Las Vegas, NV, USA, 2010, pp. 1–5, doi: 10.1109/CCNC.2010.5421749.
- [2] K. Fatahalian, "CS149: Parallel Computing," [Online course], Stanford University, Stanford, CA, USA, 2024. [Accessed: Jun. 7, 2025]. Available: <https://gfxcourses.stanford.edu/cs149/fall24>.
- [3] M. Pharr and W. R. Mark, ISPC: A SPMD compiler for high-performance CPU programming, [Online]. Available: <https://ispc.github.io/ispc.html> [Accessed: Jun. 7, 2025].
- [4] NVIDIA Corporation, CUDA C++ Programming Guide, Version 12.9, [Online]. Available: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/> [Accessed: Jun. 7, 2025].
- [5] "Fast Fourier transform," CP-Algorithms, [Online]. Available: <https://cp-algorithms.com/algebra/fft.html>. [Accessed: Jun. 7, 2025].
- [6] N. H. Vũ, "Biến đổi Fourier nhanh – FFT," VNOI Wiki, [Online]. Available: <https://wiki.vnoi.info/algo/trick/FFT>. [Accessed: Jun. 7, 2025].
- [7] J. Gauthier, Fast Multipoint Evaluation On n Arbitrary Points, MSc. thesis, Dept. of Mathematics, Simon Fraser Univ., Burnaby, BC, Canada, 2017. [Online]. Available: <https://www.cecm.sfu.ca/CAG/theses/justine.pdf>. [Accessed: Jun. 7, 2025].
- [8] J. McKernan, 18.703 Modern Algebra, Spring 2013, Massachusetts Institute of Technology, 2013. [Online]. Available: <https://ocw.mit.edu/courses/18-703-modern-algebra-spring-2013/>. [Accessed: Jun. 7, 2025].
- [9] J. M. Pollard, "The fast Fourier transform in a finite field," *Mathematics of Computation*, vol. 25, no. 114, pp. 365–374, 1971. DOI: 10.1090/S0025-5718-1971-0301966-0.
- [10] Wikipedia contributors, “Compare-and-swap,” Wikipedia, The Free Encyclopedia, [Online]. Available: <https://en.wikipedia.org/wiki/Compare-and-swap>. [Accessed: Jun. 7, 2025].

[11] Wikipedia contributors, “Packet loss,” Wikipedia, The Free Encyclopedia, [Online]. Available: [https://en.wikipedia.org/wiki/Packet\\_loss](https://en.wikipedia.org/wiki/Packet_loss). [Accessed: Jun. 7, 2025].

[12] U. Nguyen, Demo Video – Final Project on Parallel Processing, personal video, Google Drive, Jun. 2025. [Online]. Available: <https://drive.google.com/file/d/1J8mvpULf2tqIhwdmzzFB3uwXuhCWSwNz/view?usp=sharing> [Accessed: Jun. 18, 2025].