BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC NÔNG LÂM THÀNH PHỐ HỒ CHÍ MINH

TIỂU LUẬN TỐT NGHIỆP

NGHIÊN CỨU VỀ XỬ LÝ SONG SONG TRONG GIS VÀ XÂY DỰNG ỨNG DỤNG SONG SONG HÓA THUẬT TOÁN ĐỊNH DÒNG CHẢY TRÊN BỀ MẶT

Sinh viên thực hiện: TRẦN CÔNG HUẤN

Ngành: HỆ THỐNG THÔNG TIN MÔI TRƯỜNG

Niên khóa: 2010 – 2014

Thành phố Hồ Chí Minh, Tháng 6 năm 2014

NGHIÊN CỨU VỀ XỬ LÝ SONG SONG TRONG GIS VÀ XÂY DỰNG ỨNG DỤNG SONG SONG HÓA THUẬT TOÁN ĐỊNH DÒNG CHẢY TRÊN BỀ MẶT

Tác giả

TRẦN CÔNG HUẤN

Giáo viên hướng dẫn:

ThS. KHƯU MINH CẢNH

Tháng 6 năm 2014

LỜI CẢM ƠN

Trong quá trình thực hiện đề tài này, tôi đã nhận được sự hướng dẫn và giúp đỡ của thầy hướng dẫn, quý thầy cô thuộc khoa môi trường tài nguyên nói chung và bộ môn thông tin địa lý ứng dụng nói riêng thuộc trường đại học Nông Lâm Thành phố Hồ Chí Minh. Qua đây tôi xin gửi lời cảm ơn chân thành tới:

- ThS.Khưu Minh Cảnh, công tác tại Trung tâm Ứng dụng Hệ thống Thông tin Địa lý – Sở Khoa học và Công nghệ TP.HCM, người đã hướng dẫn trực tiếp tôi trong quá trình làm đề tài này.
- Thầy Phó giáo sư, tiến sĩ Nguyễn Kim Lợi, chủ nhiệm Bộ môn Thông tin địa lý ứng dụng - Trường Đại học Nông lâm Tp HCM.
- KS. Lê Hoàng Tú Kĩ sư GIS - Nghiên cứu viên, Trung tâm Nghiên cứu Biến đổi Khí hậu, Trường Đại học Nông Lâm TP.HCM.
- KS. Nguyễn Duy Liêm Kĩ sư GIS - Giảng viên Khoa Môi trường – Tài nguyên, Trường Đại học Nông Lâm TP.HCM.
- Đặc biệt tôi xin cảm ơn đến ba mẹ đã sinh thành, nuôi nấng giáo dục tôi đi đến ngày hôm nay. Gia đình và bạn bè luôn động viên giúp đỡ, tạo điều kiện thuận lợi cho tôi trong quá trình học tập cũng như trong thời gian thực hiện đề tài, đồng thời cho tôi cũng gửi lời cảm ơn sâu sắc tới Sở Khoa Học và Công Nghệ Thành Phố Hồ Chí mình đã tạo điều kiện cho tôi thực tập, giúp đỡ, cung cấp số liệu giúp tôi thực hiện tốt đề tài này.

TP.HCM, ngày... tháng... năm 2014

Trần Công Huấn

Bộ môn Tài nguyên và GIS

Khoa Môi trường & Tài nguyên

Trường Đại học Nông Lâm Tp.Hồ Chí Minh

TÓM TẮT

Đề tài nghiên cứu "Nghiên cứu về xử lý song song trong GIS và xây dựng ứng dụng song song hóa thuật toán định dòng chảy trên bề mặt" được thực hiện và hoàn thành tại Phòng kỹ thuật – Trung tâm ứng dụng Hệ Thống Thông Tin Địa Lý – Sở Khoa học và Công nghệ Thành phố Hồ Chí Minh. Thời gian thực hiện từ 2/3/2014 đến 31/5/2014. Gồm các nội dung như sau:

- Tìm hiểu về các mô hình, công nghệ, họ máy tính song song.
- Tìm hiểu về các thuật toán song song, cấu trúc các thuật toán song song.
- Tìm hiểu về thuật toán Floyd – Warshall và thuật toán tìm tích lũy dòng chảy.
- Tìm hiểu về các công cụ tìm dòng trong ArcGis.
- Tìm hiểu về thuật toán xác định hướng dòng chảy theo D8 và cài đặt trên ngôn ngữ C sharp.
- Tìm hiểu về các dạng mô hình dữ liệu của DEM.
- Tìm hiểu về gói phát triển ArcEngine của ArcGis trên Visual studio. Trên nền đó xây dựng các công cụ hỗ trợ hiển thị, cập nhật, phân tích dữ liệu và chuyển dạng dữ liệu.

Các kết quả thu được:

- Xây dựng được công cụ xác định hướng dòng chảy (theo D8) và tính tích lũy của dòng chảy trên bề mặt.
- Ứng dụng khai thác tốc độ tối đa của máy tính và cho ra kết quả nhanh nhất có thể nhằm tiết kiệm thời gian cho người sử dụng.
- Xây dựng các công cụ chuyển dữ liệu dạng file text sang dạng raster và hiển thị dữ liệu raster lên form ứng dụng.

MỤC LỤC

DANH MỤC TỪ VIẾT TẮT

GIS: Geographic Information System (Hệ thống thông tin địa lý).

ESRI: Economic and Social Research Institute (Viện nghiên cứu hệ thống môi trường).

TP.HCM: Thành phố Hồ Chí Minh.

DEM: Digital Elevation Models (Mô hình độ cao số).

CPU: Central Processing Uint (Bộ xử lý trung tâm).

Thuật toán D8 (Thuật toán xác định hướng dòng chảy đơn).

DANH MỤC BẢNG BIỂU

DANH MỤC HÌNH ẢNH

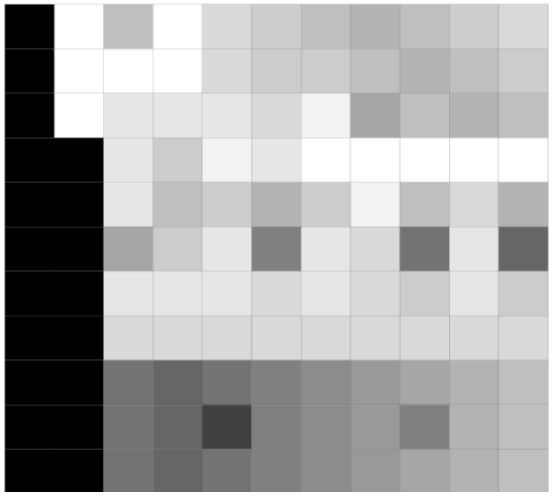
PHẦN 1. MỞ ĐẦU

1.1. Tính cấp thiết của đề tài

Mô hình độ cao số (viết tắt là DEM) là một nguồn thông tin quan trọng trong các ứng dụng GIS. Nó được sử dụng rộng rãi cho mô hình thủy văn bề mặt bao gồm phân định tự động của khu vực lưu vực, mô hình xói mòn hoặc khai thác hệ thống thoát nước tự động. Tất cả những tính toán này có liên quan đến việc xác định hướng dòng chảy, sau đó tính toán dòng chảy tích lũy. Hơn nữa tính toán tích lũy dòng chảy là đặc biệt quan trọng để điều khiển lượng nước, lượng carbon, chất dinh dưỡng và trầm tích dòng chảy trên bề mặt địa hình trong lưu vực.

0	20	15	20	17	16	15	14	15	16	17
0	20	20	20	17	16	16	15	14	15	16
0	20	18	18	18	17	19	13	15	14	15
0	0	18	16	19	18	20	20	20	20	20
0	0	18	15	16	14	16	19	15	17	14
0	0	13	16	18	10	18	17	9	18	8
0	0	18	18	18	17	18	17	16	18	16
0	0	17	17	17	17	17	17	17	17	17
0	0	9	8	9	10	11	12	13	14	15
0	0	9	8	5	10	11	12	10	14	15
0	0	9	8	9	10	11	12	13	14	15

Hình 1.1: DEM là một raster



Hình 1.2: Cấp độ xám của DEM

Nghiên cứu khoa học địa lý ngày càng phát triển với các bộ dữ liệu ngày càng lớn từ vệ tinh hoặc máy bay LIDAR cho kết quả tốt hơn. DEM có lợi thế để cho ta kết quả chính xác hơn khi phân chia ranh giới các khu vực cụ thể hoặc dùng để mô phỏng. Mặt khác các phép toán tuần tự có thể theo không kịp, hơn thế các nhà lập trình cần tính toán thời gian tương thích với vòng lặp phân tích cụ thể từng bước tính toán, giải pháp được đặt ra là cần sử dụng thuật toán song song để tối ưu khả năng tính toán và đưa ra những kết quả chính xác nhất.

Với những lý do nêu trên, tôi đã đề xuất phương pháp tính toán song song sự tích lũy dòng chảy cho hệ thống thoát nước được xây dựng từ một DEM lớn cho đề tài của mình. Đề tài: "Nghiên cứu về xử lý song song trong GIS và xây dựng ứng dụng song song hóa thuật toán định dòng chảy trên bề mặt".

1.2. Mục tiêu nghiên cứu của đề tài

Thiết kế và xây dựng một ứng dụng sử dụng thuật toán để tìm dòng chảy trên lưu vực, ứng dụng này được lập trình xử lý song song bằng ngôn ngữ C# trong môi trường Visual Studio. Mục tiêu cụ thể của đề tài như sau như sau:

- Tìm hiểu về xử lý song song, các thuật ngữ trong tính toán song song.
- Lập trình xử lý song song bằng ngôn ngữ C# trong môi trường Visual Studio.
- Tìm hiểu về các thuật toán phân tích dòng chảy.
- Cài đặt một thuật toán tính toán song song về phân tích dòng chảy trong địa hình.

1.3. Đối tượng và phạm vi nghiên cứu

1.3.1. Đối tượng nghiên cứu

Đối tượng nghiên cứu là địa hình, độ dốc, dòng chảy (mưa, lũ, vỡ đê đập tạo thành), code xử lý song song, thuật toán.

1.3.2. Phạm vi nghiên cứu

- Toán học bao gồm toán rời rạc, lý thuyết đồ thị và toán hình học. Cụ thể là:
  - Các lý thuyết khái niệm đồ thị và một số khái niệm cơ bản của đại số về hướng dòng chảy.
  - Cơ sở thống kê phân loại dữ liệu.
  - Thống kê phân tích dữ liệu tương quan.
  - Khảo sát các thuật toán dòng chảy đơn và đa (D8, D16,…)
- Lập trình bao gồm: Coding trên C shapre, xử lý song song D8, sử dụng thư viện tính toán .NET Framwork 4.0.

PHẦN 2. TỔNG QUAN VỀ THUẬT TOÁN VÀ TÍNH TOÁN SONG SONG

2.1. Đại cương về tính toán song song

2.1.1. Một số khái niệm và thuật ngữ

- **Tính toán song song hay xử lý song song (Parallel Computing/Parallel Processing):** là quá trình xử lý thông tin trong đó nhấn mạn việc nhiều đơn vị dữ liệu được xử lý đồng thời bởi một hay nhiều bộ xử lý để giải quyết một bài toán.
- **Siêu máy tính:** là những máy tính đa năng thông thường nhưng có tốc độ tính toán vô cùng lớn. Tất cả các siêu máy tính hiện nay đều là những máy tính song song. Chúng chia làm hai loại. Loại thứ nhất – máy tính song song dựa trên bộ vi xử lý – được thiết kết với rất nhiều bộ vi xử lý có tốc độ xử lý vừa phải. Loại thứ hai – siêu máy tính truyền thống (supercomputer) – ít bộ vi xử lý hơn nhưng tốc độ của mỗi bộ xử lý đó lại cực cao.
- **Song song về dữ liệu (data parallelism):** là cơ chế sử dụng nhiều đơn vị xử lý thực hiện cùng một thao tác trên nhiều đơn vị dữ liệu.
- **Song song điều khiển (control parallelism):** là cơ chế trong đó nhiều thao tác khác nhau tác động lên nhiều đơn vị dữ liệu khác nhau một cách đồng thời.
- **Dây chuyền (Pipelining):** là cơ chết chia công việc thành nhiều chặng nối tiếp nhau, mỗi chặng được thực hiện bởi một bộ phận khác nhau. Đầu ra của bộ phận này là đầu vào của bộ phận tiếp theo.
- **Tăng tốc (speedup):** Tăng tốc của thuật toán song song là tỷ số giữa thời gian thực hiện trong tình huống xấu nhất của thuật toán tuần tự tốt nhất và thời gian thực thiện cùng công việc đó của thuật toán song song.

$$\text{Tăng tốc} = \frac{\begin{array}{c}Thời\ gian\ thực\ hiện\ trong\ tình\ huống\ xấu\ nhất\ của \\ thuật\ toán\ tuần\ tự\ nhanh\ nhất\end{array}}{\begin{array}{c}Thời\ gian\ thực\ hiện\ trong\ tình\ huống\ xấu\ nhất\ của \\ thuật\ toán\ song\ song\ đang\ xét\end{array}}$$

Liên quan đến tăng tốc, năng 1967 Amdahl nêu ra định lý sau đây:

Định lý Amdahl. Gọi f là tỷ lệ thao tác tuần tử trên tổng số thao tác phải làm, trong đó $0 \leq f \leq 1$. Tăng tốc tối đa S của một máy tính song song với p bộ vi xử lý luôn nhỏ hơn.

$$S \leq \frac{1}{f + (1-f)/p}$$

- **Hiệu quả (Efficiency)** của thuật toán song song được tính bằng Tăng tốc / Số bộ xử lý tham gia tính toán
- **Giá (cost)** của một quá trình tính toán trên PRAM

Giá = thời gian tính × Số lượng bộ xử lý tham gia tính,

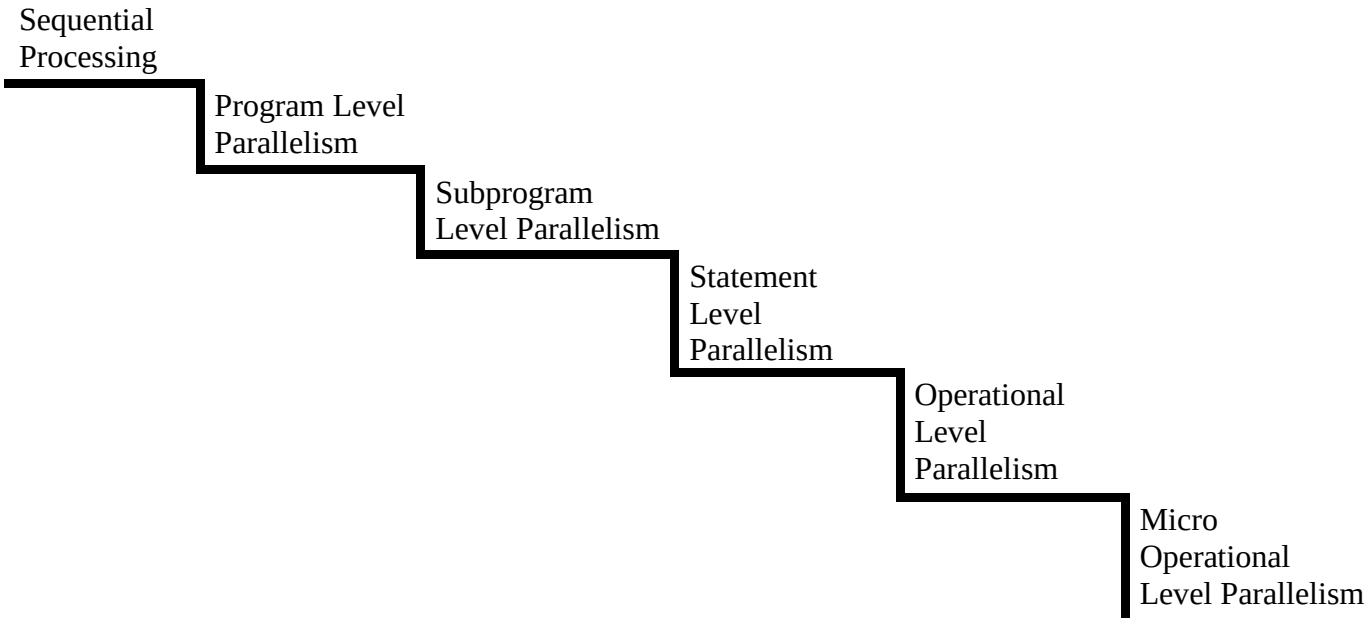
Trong đó thời gian tính chính là số phép toán cơ bản (với giả thiết rằng việc thực hiện một phép toán cơ bản tiêu tốn một đơn vị thời gian).

2.1.2. Các mức độ song song (Level of parallelism)

Việc thực hiện song song hóa có thể chia ra thành nhiều mức độ khác nhau. Chẳng hạn, nếu có 10 công việc (jobs) khác nhau về nội dung và đôi một là độc lập, thì ta có thể giao việc thực hiện 10 việc cho 10 máy khác nhau thực hiện. Đây là mức độ song song cao nhất vì các công việc có thể thực hiện độc lập. Thông thường người ta gọi mức độ song song này *mức chương trình song song* (program level). Trong quá trình thực hiện mỗi một công việc, để tăng hiệu quả thực hiện chương trình, ta lại tiếp tục song song hóa. Chia việc thực hiện mỗi chương trình ra thành các công đoạn (tasks). Các công đoạn này lại có thể thực hiện một cách song song và tổ hợp các kết quả của chúng cho ta kết quả cần tìm. Mỗi công đoạn như vậy được thực hiện bởi một chương trình con và các chương trình con này được thực hiện song song. Ta gọi mức độ song song này là *mức song song chương trình con* (subprogram level). Trong mỗi chương trình cũng như chương trình con lại có hang loạt các câu lệnh (statement) cần thực hiện. Chúng ta lại nghĩ đến thực hiện song song các thao tác này, và đó là *mức độ song song theo thao tác.* Thông thường mỗi thao tác lại bao gồm hàng loạt các thao tác nhỏ hơn (micro-operation). Chẳng hạn, xét thao tác cộng nội dung hai biến A và B và cất nó vào C (C = A+B). Thao tác này có thể thực hiện như sau:

1. Load A to accumulator
2. Add B to accumulator
3. Store the content of the accumulator to variable C

Nếu các thao tác nhỏ có thể thực hiện song song, thì đó là *mức độ song song thao tác nhỏ* (micro-operation lever parallelism). Có thể hình dung các mức độ song song trên hình 2.1.2. sau đây…



Hình 2.1.2. Các mức độ song song

Mức độ song song chương trình và chương trình con là dễ hiểu và không cần giải thích thêm. Để giải thích mức độ song song câu lệnh, ta xét ví dụ sau.

Ví dụ: Xét thuận toán:

For i:=1 to n do

x[i]:= x[i]+1;

Thời gian của thực hiện tuần tự: O(n)

Ta có thể thực hiện công việc này một cách song song. Giao công việc này cho n bộ xử lý P_1, P_2, … P_n. Bộ xử lý P_i thực hiện x[i]:= x[i]+1. Do cách bộ xử lý thực hiện đồng thời nên thời gian của thực hiện song song: O(1). Thuật toán này có thể mô tả như sau

For i:=1 to n do in parallel

x[i]:= x[i]+1

end Parallel

Mức độ song song này là mức độ song song câu lệnh.

Xét một ví dụ khác:

```
S = 0;
for i = 1 to n do
        s = s + x[i];
```

(Thuật toán này thực hiện việc tính tổng s = x[1]+ x[2]+ … + x[n]).

Dễ thấy là thuật toán này không thể song song hoá tương tự như trong ví dụ trước. Ta có nhiều cách song song hoá thuật toán này để thu được thuật toán có thời gian O(log n).

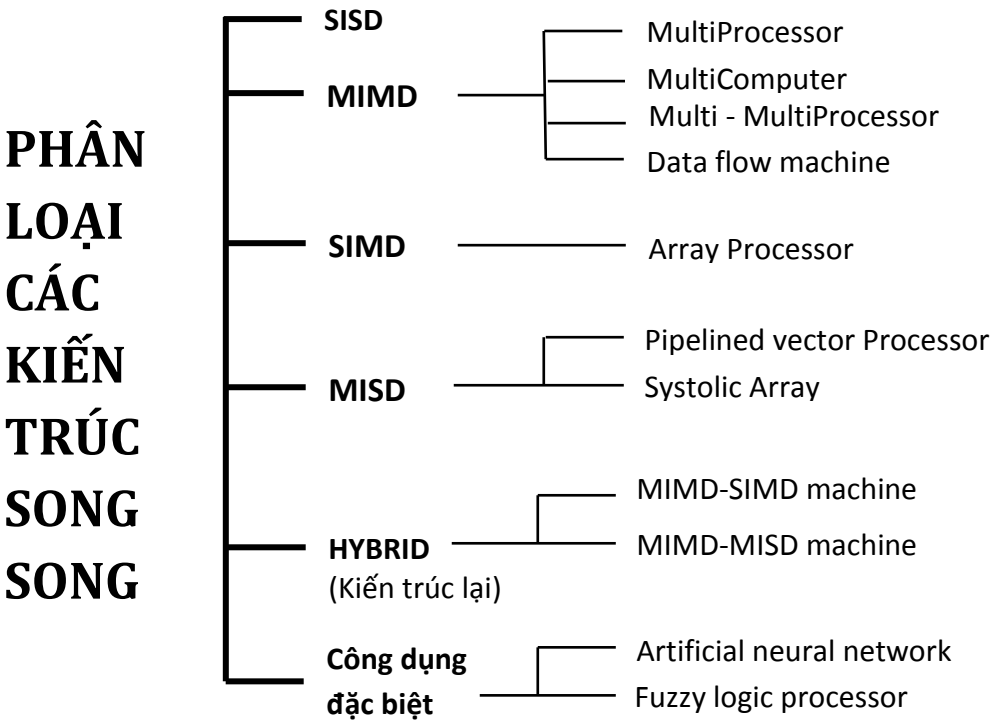
Mức độ song song thao tác cũng được thực hiện tương tự. Chẳng hạn, xét câu lệnh:

```
Y = A(i) + B(j) + C(k)
```

Trong câu lệnh này ta phải tính A(i), B(j), C(k) và sau đó cộng các kết quả để cất giữ vào Y. Việc tính A(i), B(j), C(k) có thể thực hiện song song trên ba bộ xử lý, và quá trình này được gọi là mức độ song song thao tác.

2.1.3. Phân loại các kiến trúc song song

Một trong những phân loại hay được nhắc tới là của Flynn – 1972. Michael Flynn phân các kiến trúc máy tính thành bốn loại dựa trên tương tác với lệnh (instruction) và dữ liệu (data):



Hình 2.1.3. Phân loại kiến trúc song song

- **SISD** (Single Instruction stream, Single Data stream). Đây chính là kiến trúc tuần tự Von Neuman, trong đó tại mỗi thời điểm chỉ một lệnh được thực hiện.
- **MIMD** (Multiple Instruction stream, Multiple Data stream). Cho phép nhiều lệnh khác nhau có thể đồng thời xử lý nhiều dữ liệu khác nhau trong cùng một thời điểm.
- **SIMD** (Single Instruction stream, Multiple Data stream) – còn gọi là kiến trúc xử lý mảng. Cho phép một lệnh được thực hiện đồng thời trên các dữ liệu khác nhau. Đại diện cho mô hình này là máy Connection Machine CM-200 của IBM.
- **MISD** (Multiple Instruction stream, Single Data stream). Kiến trúc này cho phép một vài lệnh cùng thao tác trên một dữ liệu.
- **HYBRID** là kiến trúc lai, có những đặc điểm của cả ba loại trên. Loại công dụng đặc biệt: có sử dụng những tiến bộ về mạng nơron và lý thuyết mờ trong thiết kế.

2.1.4. Mô hình SIMD (PRAM)

Mô hình kiến trúc này còn được gọi là PRAM (Parallel Random Access Machine – Máy tính song song truy cập bộ nhớ ngẫu nhiên). Trong mô hình này, N bộ xử lý cùng chia sẻ bộ nhớ chung. Mô hình PRAM chia thành 3 lớp nhỏ:

- **EREW** (Exclusive Read, Exclusive Write): độc quyền đọc, độc quyền ghi. Không cho phép 2 bộ xử lý đọc hoặc ghi đồng thời vào cùng một ô nhớ.
- **CREW** (Concurrent Read Exclusive Write): đọc đồng thời, ghi độc quyền. Các bô xử lý có thể đọc đồng thời, nhưng không được phép ghi đồng thời vào một ô nhớ.
- **ERCW** (Exclusive Read Concurent Write): đọc độc quyền, ghi đồng thời. Các bộ xử lý có thể ghi đồng thời, nhưng không được phép đọc đồng thời từ một ô nhớ.
- **CRCW** (Concurent Read Concurent Write): đọc/ghi đồng thời. Các bô xử lý có thể đồng thời đọc ghi vào một ô nhớ.

Việc cho phép nhiều bộ xử lý đồng thời đọc một ô nhớ không khó. Nhưng thiết kế cấu trúc ghi đồng thời thì phức tạp hơn. Khi có nhiều bô xử lý đồng thời thực hiện việc ghi vào cùng một vị trí, thì cần giải quyết xung độ này như thế nào. Có 3 cách giải quyết cơ bản:

- ECR (Equality Conflict Resolution): Chỉ thực hiện ghi nếu tất cả các bộ xử lý đều cùng ghi một giá trị như nhau;
- PCR (Priority Conflict Resolution): Trong cách giải quyết này, mỗi bộ xử lý có một chỉ số ưu tiên, và giá trị mà bộ xử lý có số thứ tự ưu tiên cao nhất sẽ được ghi.
- ACR (Arbitrary Conflict Resolution): Trong cách giải quyết này, khi có nhiều bộ xử lý cùng ghi vào một vị trí, một cách ngẫu nhiên sẽ có một trong số chúng thực hiện được việc ghi.

Sức mạnh của máy tính là tăng dần theo thứ tự trên. Có nghĩa là EREW yếu nhất, rồi đến CREW… Thật vậy, xét bài toán tìm kiếm sau. *"Một tệp có n khoản mục được lưu trữ không có thứ tự. Cần tìm một mục x trong tệp (để thực hiện xóa, sửa…)"*

Với máy tính tuần tự, việc này tốn thời gian cỡ Θ(n).

Trên máy tính EREW với k bộ xử lý, việc này được thực hiện theo các bước sau:

1. Quảng bá giá trị x cho các bộ xử lý.
2. Tệp được chia làm k phần và phân cho mỗi bộ xử lý tìm kiếm trên một phần.

Gọi các bộ xử lý là $P_0, P_1, P_2, \dots, P_{k-1}$. Trong thao tác quảng bá giá trị x cho các bộ xử lý thực hiện như sau:

- P_0 đọc x và báo cho P_1
- P_0 và P_1 báo cho P_2 và P_3
- P_0, P_1, P_2, P_3 báo cho P_4, P_5, P_6, P_7

Rõ ràng là thao tác này tốn $\mathbf{\log k}$ đơn vị thời gian.

Trên k phần, mỗi phần có n/k mục, các bộ xử lý tiến hành việc tìm kiếm như trên các máy tính tuần tự. Do đó, thời gian thực hiện sẽ là $O(n/k)$
Tổng cộng độ phức tạp tính toán là: $\mathbf{O(n/k) + O(\log k)}$

Trong khi đó, cũng bài toán trên nếu sử dụng máy tính CREW rõ ràng ta không phải thực hiện bước quảng bá x cho các bộ xử lý vì chúng có thể cùng một lúc đọc x từ ô nhớ. Do đó độ phức tạp tính toán giảm xuống chỉ còn: $\mathbf{O(n/k)}$. Mặt khác, nếu x xuất hiện nhiều lần trong tệp, sẽ có nhiều bộ xử lý muốn ghi vào vùng chứa kết quả. Điều này chỉ thực hiển được trên mô hình CRCW.

Mặt dù yếu nhất, nhưng công nghệ chế tạo phổ biến hiện nay lại là các máy EREW, còn các máy CRCW, CREW thì đắt và khó chế tạo. Do đó ta phải nghĩ tới việc dùng EREW để mô phỏng các kiến trúc còn lại như trình bày dưới đây.

2.1.5. Dùng công nghệ EREW mô phỏng các kiến trúc CRCW, CREW

Mô phỏng việc cho phép Đọc đồng thời: Tính năng *Đọc đồng thời* được mô phỏng trên EREW bằng cách sử dụng thao tác quảng bá giá trị cần đọc cho các bộ xử lý. Ở trên chúng ta đã thấy rằng thao tác này đòi hỏi thời gian thực hiện là $\mathbf{O(\log n)}$.

Mô phỏng việc cho phép Ghi đồng thời: Giả sử chúng ta quy định trằng, các bộ xử lý chỉ được phép ghi nếu các giá trị cần ghi trùng nhau. Như vậy thao tác ghi đồng thời được tiến hành trên máy mô phỏng như sau:

1. Kiểm tra xem n giá trị cần ghi có trùng nhau không.
2. Nếu trùng nhau thì tiến hành thao tác ghi. Nếu không thì dừng.

Thao tác thứ 2 chỉ tốn thời gian cỡ hằng số. Thao tác kiểm tra thứ nhất có thể diễn ra như sau, với a_i giá trị thứ i.

- for $i \leftarrow 1$ to $(n/2)$ do in parallel

  if $a_i = a_{i+n/4}$ then $b_i \coloneqq$ true

  else $b_i \coloneqq$ false ;

- for $i \leftarrow 1$ to $(n/4)$ do in parallel

  if $\left(a_i = a_{i+n/4}\right)$ and $(b_i \coloneqq \text{true})$

  and $\left(b_{i+n/4} \coloneqq \text{true}\right)$

  then $b_i \coloneqq$ true else $b_i \coloneqq$ false ;

... ...

Ta thấy sau $\log n$ bước, quá trình kiểm tra kết thúc. Như vậy bước kiểm tra đòi hỏi thời gian cỡ $\log n$ và đó cũng là giá phải trả cho việc mô phỏng.

2.1.6. Họ máy MIND

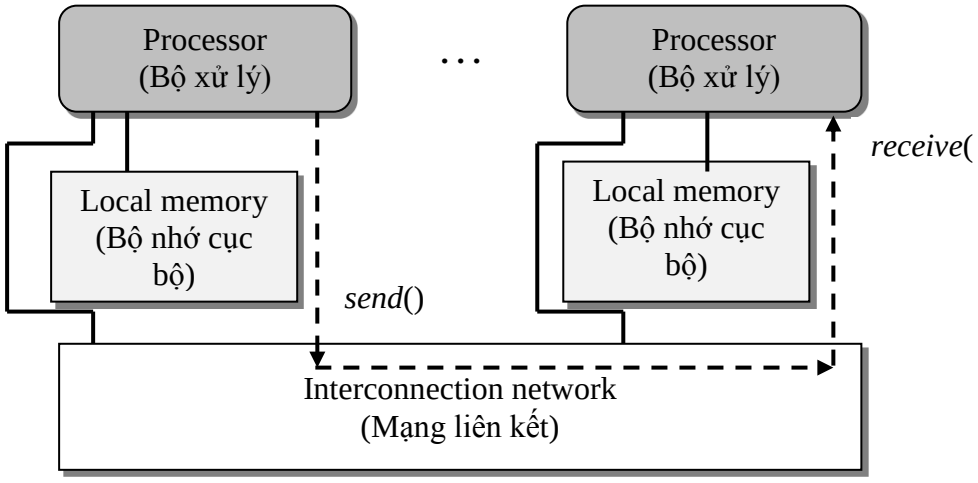
Trong các mục trên chúng ta đã xem xét họ máy SIMD, hay còn được gọi là PRAM. Sau đây chúng ta xem xét một kiến trúc song song khác – lớp máy MIMD. Lớp này phân thành 3 loại:

- Hệ đa xử lý với bộ nhớ phân tán
- Hệ đa xử lý dùng chung bộ nhớ
- Hệ đa xử lý với bộ nhớ dùng chung phân tán

2.1.6.1. Hệ đa xử lý với bộ nhớ phân tán (*Multi processor system with Distributed Memory*)

Đây chính hệ song song gồm nhiều máy tính kết nối thành mạng (multicomputer system). Hình 2.1.6.1 mô tả kiến trúc của hệ thống. Một vài đặc điểm của hệ thống:

- Các bộ xử lý chỉ được quyền truy cập vùng nhớ cục bộ của mình.
- Liên kết giữa các bộ xử lý được thực hiện theo mô hình *Chuyển thông báo* (message passing)
- Hệ thống có quy mô lớn, có thể lên đến hàng chục ngàn bộ xử lý. Khi đó bộ xử lý quá lớn có thể làm cho đường truyền mạng trở nên quá tải.
- Kỹ thuật lập trình khá phức tạp, tương ứng với các môi trường lập trình Message Passing như PVM, MPI.
- Còn được gọi dưới một tên khác là hệ NORMA (no remote memory access model: mô hình không cho phép truy cập vùng nhớ ở xa). Ta thấy mỗi bộ xử lý có một *vùng nhớ cục bộ* riêng của mình (local memory) và chỉ được quyền truy cập vào đó, nó là *vùng nhớ ở xa* (remote memory) đối với các bộ xử lý khác và cũng không được quyền truy cập vào đây.

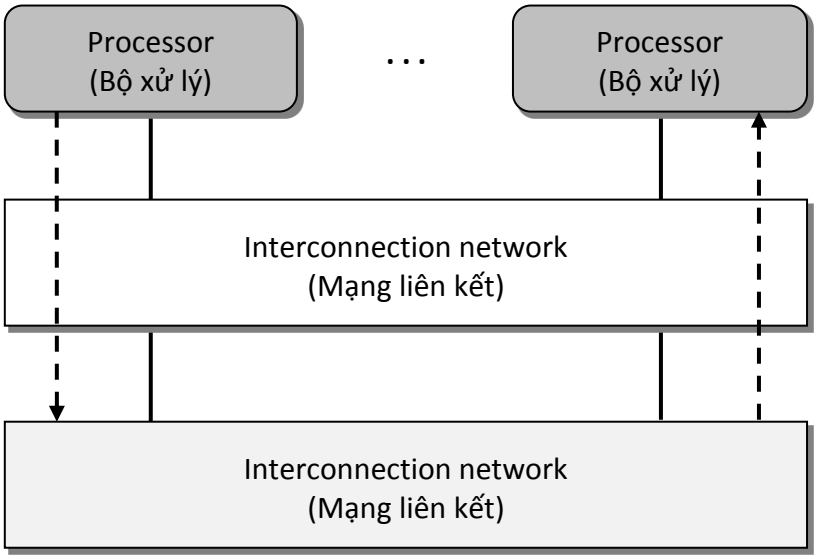


Hình 2.1.6.1: Hệ đa xử lý với bộ nhớ phân tán

2.1.6.2. Hệ đa xử lý dùng chung bộ nhớ (Multi processor system with Shared Memory)

Đây là các máy tính lớn với nhiều bộ xử lý (multi processor) hoạt động theo cơ chế *Đa xử lý đối xứng SMP (symmetric multi processing)*. Kiến trúc của hệ thống được mô tả trong hình 2.2.6.2 Một số đặc điểm của hệ thống:

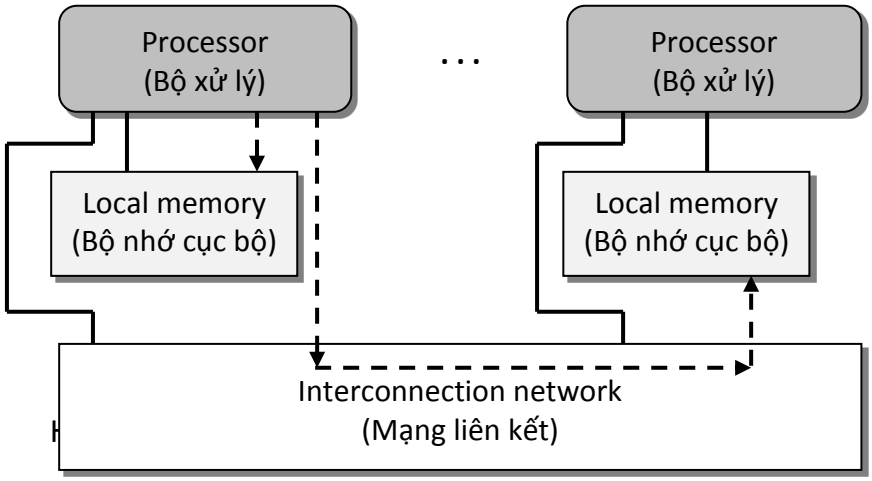
- Các bộ xử lý có thể truy nhập toàn bộ vùng nhớ dùng chung.
- Liên lạc giữa các bộ xử lý được thực hiện thông qua vùng nhớ dùng chung. Giả sử bộ xử lý P_1 muốn gửi dữ liệu cho P_2, dữ liệu đó sẽ được ghi vào vùng nhớ dùng chung rồi báo địa chỉ cho P_2. P_2 sẽ đọc tại địa chỉ đó để lấy dữ liệu về.
- Quy mô của hệ thống tương đối nhỏ, chỉ cỡ vài trăm bộ xử lý.
- Kỹ thuật lập trình dễ dàng hơn so với mô hình trên.
- Còn được gọi dưới một tên khác là hệ UMA (uniform memory access model: quản lý và đánh đại chỉ toàn bộ vùng nhớ dùng chung theo một dạng địa chỉ thống nhất).



Hình 2.1.6.2: Hệ đa xử lý dùng chung bộ nhớ

Ta sẽ thấy hệ NUMA dưới đây tuy cũng quản lý được toàn bộ vùng nhớ dùng chung nhưng không dùng một mà dùng nhiều dạng địa chỉ.

2.1.6.3. Hệ đa xử lý với bộ nhớ dùng chung phân tán (Multi processor system with distributed shared memory)



Hình 2.1.6.3: Hệ đa xử lý với bộ nhớ dùng chung phân tích

- Về mặt vật lý, mỗi bộ xử lý đều có một vùng nhớ cục bộ của mình.
- Sự truy cập tới các vùng nhớ khác nhau được thực hiện nhờ cơ chế truy nhập mạng. Nhờ vậy, tất cả các vùng nhớ cục bộ đều được gom lại và đánh địa chỉ như một vùng nhớ logic duy nhất. Các bộ xử lý đều có thể truy nhập mọi địa chỉ trong vùng nhớ chung đó.
- Còn được gọi dưới môt tên khác là hệ NUMA (non uniform memory access model: quản lý và đánh địa chỉ vùng nhớ chung theo nhiều dạng địa chỉ). Ta có thể thấy mỗi bộ xử lý, ngoài *vùng nhớ cục bộ* (local memory) của mình, còn có thể truy nhập vào các *vùng nhớ ở xa* (remote memory: là vùng nhớ cục bộ của các bộ xử lý khác). Hai vùng nhớ này được quản lý theo hai chế độ địa chỉ khác nhau.

2.1.7. Ngôn ngữ mô tả thuật toán song song

Trước hết, toán tử gán có dạng: Variable = expression.

Ở đây expression sẽ được tính và kết quả sẽ được cất giữ vào variable. Câu lệnh if có dạng:

if cond then

s_1

s_2

…

Else

s'_1

s'_2

…

Endif

Trong cú pháp này cond là điều kiện logic. Nếu điều kiện là đúng thì s_1, s_2, … được thực hiện, trái lại s'_1, s'_2, … được thực hiện.

Vòng lặp For có dạng sau đây:

For variable = s to e step h

s_1

s_2

…

end for

Các lệnh s_1, s_2,… được thực hiện với các giá trị của variable lần lượt là s, s + h, s + 2h, … cho đến khi s + k.h > e.

Chúng ta sử dụng cả vòng lặp while dưới dạng sau:

While cond do

s_1

s_2

…

end while

Các lệnh s_1, s_2,… được lặp lại chừng nào điều kiện cond còn là true.

Việc thực hiện song song được mô tả nhờ lệnh For-in-Parallel. Câu lệnh này có thể viết theo hai cấu trúc khác nhau.

Cấu trúc 1.

For variable = 1 to n do in parallel

s_1

s_2

...

End parallel

Đồng thời, mỗi một trong n bộ xử lý đều thực hiện cùng một dãy các lệnh s_1, s_2,...

Biến chạy trong vòng For thể hiện chỉ số của các bộ xử lý.

Cấu trúc 2.

For $x \in$ S do in parallel

s_1

s_2

...

End parallel

Trong cấu trúc này dãy lệnh s_1, s_2,… được tất cả các bộ xử lý với chỉ số trong S thực hiện đồng thời.

Giả sử x, y, z là các biến trong bộ nhớ toàn cục và ta viết:

x = y +z

Trong thuật toán song song, Công việc này sẽ được bộ xử lý tiến hành như sau:

1. Đọc nội dung biến y và gọi nó là v_1 (v_1 là biến trong bộ nhớ địa phương của bộ xử lý);
2. Đọc nội dung của biến z và gọi nó là v_2 (v_2 là biến trong bộ nhớ địa phương của bộ xử lý);
3. $v_3 = v_1 + v_2$ (v_3 là biến trong bộ nhớ địa phương của bộ xử lý):
4. Ghi giá trị của v_3 vào biến toàn cục x.

2.2. Các mô hình tính toán song song và minh họa

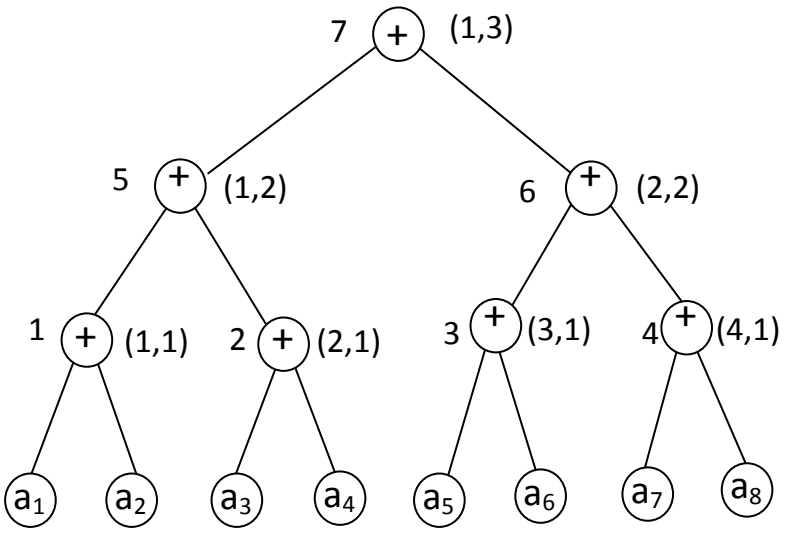
Mỗi giải thuật song song bao giờ cũng được thiết kết với giả thiết sử dụng một kiến trúc nhất định của máy tính song song. Ta gọi điều đó là mô hình tính toán song song. Có rất nhiều mô hình khác nhau và một số mô hình cơ bản là:

2.2.1. Mô hình cây nhị phân (Bimary Tree Model)

Quá trình xử lý vấn đề đặt ra được biểu diễn dưới dạng một cây nhị phân, trong đó mỗi nút trung gian (nút không phải là lá) có hai con. Mỗi nút trung gian biểu diễn một thao tác. Các thao tác ở trên cùng một mức được thực hiện song song. Mô hình

cây nhị phân đôi khi còn được gọi là mô hình đồ thị có hướng không có chu trình (DAG model).

Xét một ví dụ minh họa. Giả sử ta cần tính tổng của 8 số. Ta cần vẽ một cây nhị phân với mức ở độ cao thấp nhất gồm 8 lá tương ứng với 8 số đã cho. Các nút trung gian biểu diễn việc cộng hai số tương ứng với hai con của nó. Hình 2.2.1 dưới đây cho ta thấy quá trình này:



Hình 2.2.1.1: Mô hình cây nhị phân cộng 8 số

Giả sử có 4 bô xử lý có thể sử dụng. Ta cần lập lịch phân việc thực hiện các thao tác ở các nút trung gian cho các bộ xử lý này. Việc lập lịch được thực hiện bởi hàm SCH. Hàm này được gán cho mỗi trung gian cặp có thứ tự (p, t), trong đó p là chỉ số của bộ xử lý còn t là thời điểm mà thao tác này được thực hiện. Các nút trung gian được đánh số bởi 1, 2, …, 7. Bảng dưới đây cho giá trị của hàm SCH.

SCH	Ý Nghĩa
SCH(1) = (1, 1)	Bộ xử lý 1 thực hiện tại thời điểm 1
SCH(2) = (2, 1)	Bộ xử lý 2 thực hiện tại thời điểm 1
SCH(3) = (3, 1)	Bộ xử lý 3 thực hiện tại thời điểm 1
SCH(4) = (4, 1)	Bộ xử lý 4 thực hiện tại thời điểm 1
SCH(5) = (1, 2)	Bộ xử lý 1 thực hiện tại thời điểm 2
SCH(6) = (2, 2)	Bộ xử lý 2 thực hiện tại thời điểm 2
SCH(7) = (1, 3)	Bộ xử lý 1 thực hiện tại thời điểm 3

Theo lịch này ở khoảng thời gian t = 0 đến t = 1, các công việc sau đây được thực hiện song song

Bộ xử lý 1 cộng a_1 với a_2.

Bộ xử lý 2 cộng a_3 với a_4.

Bộ xử lý 3 cộng a_5 với a_6.

Bộ xử lý 4 cộng a_7 với a_8.

Các giá trị của $(a_1 + a_2), (a_3 + a_4), (a_5 + a_6)$ và $(a_7 + a_8)$ là đã biết. Trong khoảng thời gian từ $t = 1$ đến $t = 2$, các công việc sau đây được thực hiện song song

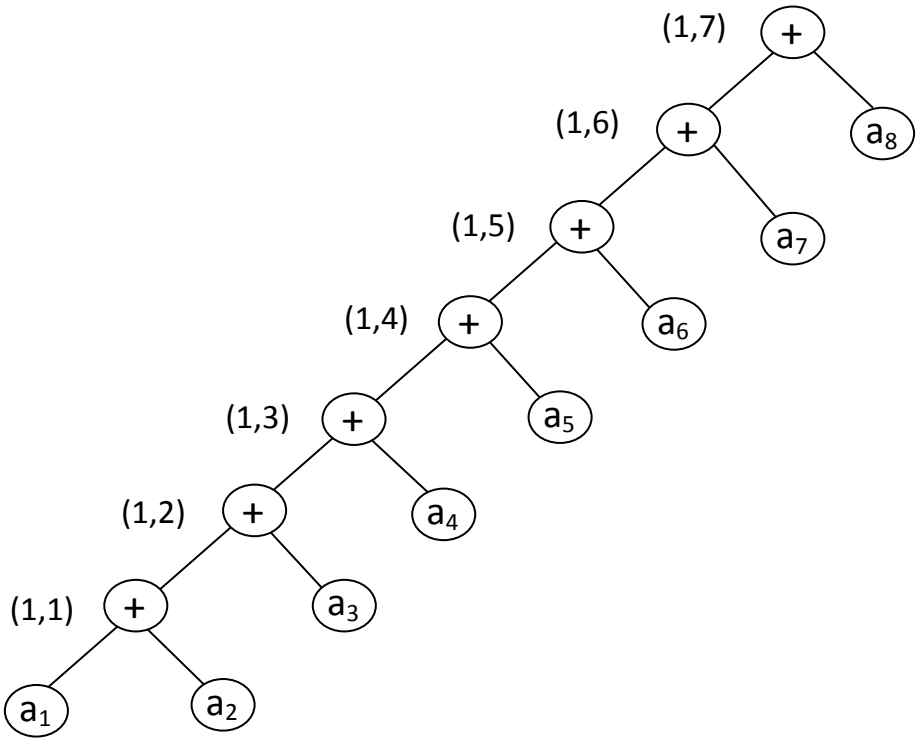
Bộ xử lý 1 cộng $(a_1 + a_2)$ và $(a_3 + a_4)$.

Bộ xử lý 2 cộng $(a_5 + a_6)$ và $(a_7 + a_8)$.

Bây giờ các giá trị của $(a_1 + a_2 + a_3 + a_4)$ và $(a_5 + a_6 + a_7 + a_8)$ là đã biết. Trong khoảng thời gian từ $t = 2$ đến $t = 3$, việc cộng $(a_1 + a_2 + a_3 + a_4)$ và $(a_5 + a_6 + a_7 + a_8)$ được thực hiện bởi bộ xử lý 1. Công việc được thực hiện giống như theo cách biểu diễn:

$$[(a_1 + a_2) + (a_3 + a_4)] + [(a_5 + a_6) + (a_7 + a_8)].$$

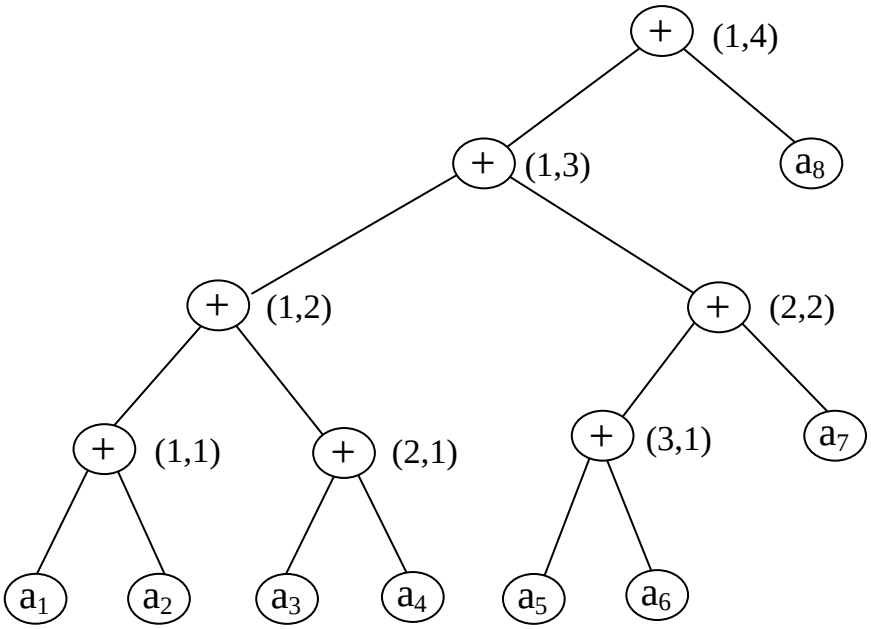
Như vậy việc cộng 8 số sử dụng 4 bộ xử lý có thể thực hiện xong sau 3 đơn vị thời gian. Một cây nhị phân đầy đủ n lá có độ cao không quá $\log n$, vì thế việc cộng n số có thể thực hiện nhờ sử dụng $[n/2]$ bộ xử lý sau thời gian $\log n$.Giả sử chỉ có một bộ xử lý để thực hiện tính toán. Trong trường hợp này thủ tục tính toán có thể biểu diễn bởi sơ đồ cây cho trong hình 2.2.1.2 sau:



Hình 2.2.1.2: Cây nhị phân thực hiện tính toán tuần tự

Giá trị của hàm SCH ghi bên cạnh mỗi nút trung gian. Thời gian thực hiện là 7. Nếu phải cộng n số thì thời gian sẽ là $n - 1$.

Khi chỉ có 3 bộ xử lý để thực hiện công việc, mô hình cây nhị phân để thực hiện cộng 8 số có thể mô tả trong hình 2.2.1.3, trong đó thời gian thực hiện là 4.



Hình 2.2.3: Tính tổng với 3 bộ xử lý

2.2.2. Mô hình mạng

Khi chúng ta tiến hành song song hóa một hệ thống, điều đương nhiên là cần sử dụng nhiều bộ xử lý. Vì vậy, ta có thể giả thiết rằng các bộ xử lý hoạt động độc lập và truyền tin cho nhau. Để thực hiện việc truyền tin, các bộ xử lý cần được nối với nhau sử dụng các kênh nối vật lý. Mô hình như vậy được gọi là mô hình mạng. Trong mô hình mạng, các bộ xử lý được nối với nhau bởi các kênh vật lý và thông thường người ta giả thiết rằng:

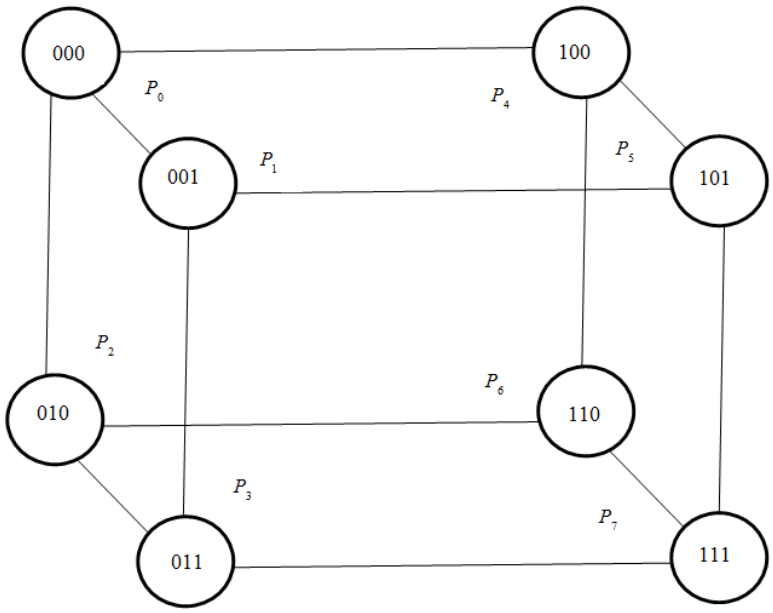
1. Mỗi bộ xử lý có bộ nhớ riêng đủ lớn, mà ta sẽ gọi đó là bộ nhớ địa phương. Chương trình cần được thực hiện trên bộ xử lý và dữ liệu vào, dữ liệu ra đều được giữ trên bộ nhớ này.
2. Không có bộ nhớ dùng chung cho tất các các bộ xử lý.
3. Các bộ xử lý được kết nối trực tiếp nhờ các kênh vật lý theo một kết cấu tôpô được gọi là tôpô mạng (network topology). Tôpô mạng là một đồ thị có các đỉnh tương ứng với các bộ xử lý và các cạnh tương ứng với các kênh kết nối vật lý. Các kênh kết nối cho phép truyền tin theo mội chiều hoặc hai chiều.
4. Nếu hai bộ xử lý là kề nhau (nghĩa là có kênh nối trực tiếp giữa chúng) thì dữ liệu có thể truyền từ bộ xử lý này sang bộ xử lý khác.
5. Trong một nhịp thời gian (clock cycle), mỗi bộ xử lý đều thực hiện được một thao tác cơ bản.
6. Mỗi bộ xử lý có thể truyền dữ liệu cho bất cứ bộ xử lý nào kề với nó trong một đơn vị thời gian.

Trong một thuật toán được thiết kế cho mô hình mạng, 3 vấn đề sau đây cần được chỉ rõ:

1) Cấu trúc tôpô của mạng;
2) Phân bổ đầu vào (Input Configuration);
3) Phân bổ đầu ra (Output Configuration);

- **Tôpô của mạng:** Rất tiếc là không có một cấu trúc tôpô nào thích hợp cho mọi loại toán. Vì thế cần phải khảo sát bài toán và đưa ra cấu trúc tôpô thích hợp. Cấu trúc tôpô có thường dùng là: đồ thị vòng, cây, đồ thị siêu hộp,…

- **Phân bổ đầu vào:** Trong thuật toán được thiết kế cho mô hình mạng để giải quyết một bài toán, dữ liệu cần được phân bổ cho một số bộ xử lý. Phân bổ này được gọi là cấu hình đầu vào.
- **Phân bổ đầu ra:** Phân bổ đầu ra cho một số bộ xử lý được gọi là cấu hình đầu ra.
- **Mạng liên kết siêu hộp (Hypercube/d-cube connection):** Ta có thể định nghĩa d-cube một cách đệ qui. 0-cube là một bộ xử lý. 1- cube là mạng gồm hai bộ xử lý được nối với nhau. 2-cube là mạng gồm bốn bộ xử lí được nối với nhau theo sơ đồ có dạng một hình vuông. 2-cube có thể xây dựng từ 1-cube. Thật vậy, từ hai 1-cube: P_0-P_1 và P_2-P_3, thực hiện nối P_0 với P_2 và P_1 với P_3. Ta có 3-cube là cặp gồm hai 2-cube $P_0P_1P_2P_3$ và $P_4P_5P_6P_7$ được bổ sung các kênh nối giữa các cặp máy:



P_0 và P_4; P_1 và P_5; P_2 và P_6; P_3 và P_7.

Hình 2.2.2.1: Mạng 3-cube

Tổng quát, d-cube được xác định bởi một cặp gồm hai (d-1) – cube trong đó các đỉnh tương ứng của chúng được nối với nhau. Như vậy d-cube gồm 2^d đỉnh trong đó các đỉnh được gán nhãn bởi các số nhị phân có độ dài d. hai đỉnh được nối với nhau khi và chỉ khi hai xâu nhãn của chúng chỉ khác nhau đúng một vị trí.

Từ định nghĩa của d-cube trực tiếp suy ra một số tính chất của d-cube sau đây.

- **Tính chất 1.** Sơ đồ nối mạng d-cube là đồ thị chính qui bậc d.
- **Tính chất 2**. Khoảng cách từ đỉnh a=($a_1a_2…a_k$) và b=($b_1b_2…b_k$) bằng số lượng bit mà tại đó a và b là khác nhau.
- **Tính chất 3.** Trong mạng siêu hộp n nút (với n = 2^k) dữ liệu có thể truyền từ nút này đến nút khác sau thời gian không quá log(n).

Thực vậy, do hai nhãn của 2 đỉnh bất kỳ khác nhau không quá log n bit nên từ tính chất 2 suy ra luôn tồn tại đường đi độ dài không quá log n giữa chúng.

- **Tính chất 4.** Gọi e(k) là số lượng cạnh của một k-cube với n = 2^k đỉnh, ta có E(k) = k x 2^{k-1} =(log n) n/2

Từ cách xác định k-cube ta có công thức đệ qui

$E(k) = 2e(k-1) + 2^{k-1}$

$=2(2e(k-2) + 2^{k-2}) + 2^{k-1}$

$=2^2e(k-2) + 2x2^{k-1}$

= …

$=2^{k-1}e(1) + (k-1)x2^{k-1}$, do e(1) = 1

$=kx2^{k-1}$.

Bảng dưới đây cho ta so sánh một số đặc trưng của k-cube với đồ thị đầy đủ

Số đỉnh (n)	Log n	Số cạnh của đồ thị đầy đủ (n(n-1)/2)	Số cạnh của k-cube
2	1	1	1
16	4	120	32
64	6	2016	192
1024	10	523776	5120

Bảng 2.2.2: so sánh một số đặc trưng của k- cute với đồ thị đầy đủ

Khi các nút được nối với nhau giống như đồ thị đầy đủ n đỉnh ta có số lượng cạnh là n(n-1)/2. Trong đồ thị đầy đủ, do hai đỉnh bất kỳ luôn được nối với nhau, nên dữ liệu có thể truyền từ máy này sang máy kia sau 1 đơn vị thời gian. Tuy nhiên do số lượng cạnh quá lớn, mà cạnh ở đây tương ứng với các kênh vật lý nối các máy, nên điều đó dẫn đến chi phí xây dựng mạng quá lớn. Nếu ta nối mạng kiểu siêu hộp, số

lượng cạnh sử dụng chỉ là (log n) n/2 và ta vẫn có thể truyền tin từ máy này sang máy kia với thời gian không quá log n.

Bây giờ ta tìm hiểu việc thiết kế thuật toán cho cấu trúc tôpô k-cube trong mạng kết nối trực tiếp. Xét thuật toán tìm số nhỏ nhất trong một mảng số cho trước.

Giả sử các phần tử của mảng là

$a_0a_1a_2\ldots a_{n-1}$,

Thuật toán tuần tự để giải quyết bài toán là

Small = a_0

For i = 1 to n - 1 do

If small > a_i then

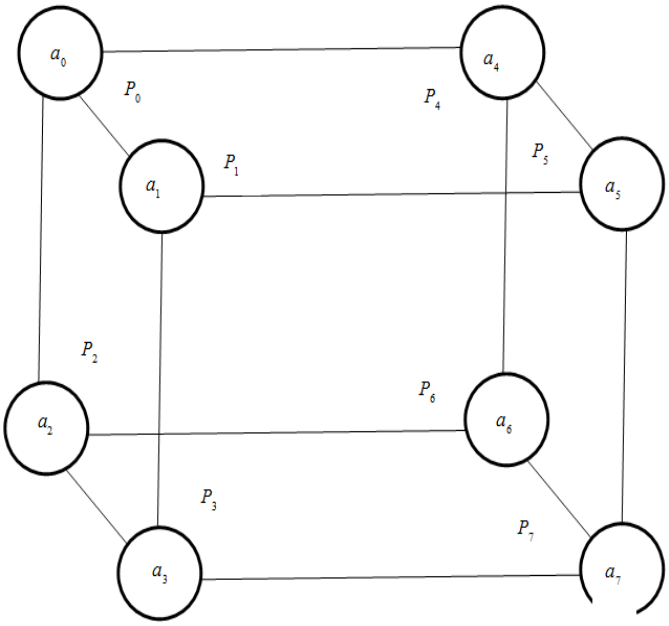
Small = a_i

Endif

endfor

Thuật toán tuần tự đòi hỏi thời gian tính là O(n). Sử dụng mạng kết nối các bộ sử lý dạng k-cube ta có thể thiết kế thuật toán với thời gian tính O(log n). Ta bắt đầu từ việc mô tả phân bổ đầu vào.

- **Phân bổ đầu vào.** Ta phân bổ *n* số cho n bộ xử lý. Để đơn giản, ta giả thiết là $n=2^k$. Khởi đầu, số a_i ở trong bộ nhớ của bộ xử lý P_i ($0 \le i \le n-1$). Điều đó thể hiện trong hình sau:



Hình 2.2.2.2: Phân bổ đầu vào

- **Phân bổ đầu ra**. Chúng ta muốn cất giữ kết quả (số nhỏ nhất trong mảng) ở P_0 Xử lý. Theo định nghĩa, k-cube là cặp gồm hai (k-1)-cube, trong đó Pi nối với $P_{i+2^{(k-1)}}$. Bây giờ ta sẽ truyền nội dung của $P_{i+2^{(k-1)}}$ cho P_i ($0 \le i < 2^{k-1}-1$). Như vậy mỗi bộ xử lý P_i đều có thông tin về hai số ai và $a_{i+2^{(k-1)}}$ ($i = 0, 1, \ldots, 2^{k-1}$). Tiếp theo, mỗi bộ xử lý P_i nãy sẽ tiến hành so sánh hai số a_i và $a_{i+2^{(k-1)}}$ và cất giữ số nhỏ hơn vào a_i. Sau thao tác này số nhỏ nhất trong mảng đã cho sẽ nằm trong mảng

  $a_0a_1a_2 \ldots a_{(n/2)-1}$

Quá trình sẽ được lặp lại trên (k-1)-cube bao gồm các bộ xử lý P_0, P_1, .., $P_{(n/2)-1}$. Lặp lại thủ tục này k lần, số nhỏ nhất sẽ được cất giữ trong P_0.

2.2.3. Thuật toán k-cube-Min

- **Phân bổ đầu vào:** Mỗi bộ xử lý P_i chứa một số a_i ở bộ nhớ địa phương của nó (i = 0,1,2, ..., n-1). Giả thiết là $n = 2^k$.
- **Phân bổ đầu ra:** Số nhỏ nhất trong mảng a_0, a_1, ..., a_{n-1} cất giữ vào a_0.

1. For d = k - 1 to 1 step - 1 do
2. M = 2^d
3. For i = 0 to m -1 do in parallel
4. Bộ xử lý P_{i+m} truyền dữ liệu a_{i+m} cho bộ xử lý P_i
5. Bộ xử lý P_i nhận dữ liệu a_{i+m} truyền tới từ P_{i+m} và cất giữ vào b_i trong bộ nhớ địa phương
6. Bộ xử lý P_i thực hiện tính $a_i = \min \{a_i, b_i\}$
7. end parallel
8. end for

Do phải thực hiện vòng lặp for (bước 1 đến 8) nên thuật toán làm việc với thời gian O(k), cũng nghĩa là với thời gian O(log n). Trong thủ tục mô tả ở trên các bước 4 và 5 mô tả việc truyền tin giữa các bộ xử lý. Thời gian truyền tin thường là rất lớn so với thời gian thực hiện các tính toán. Vì thế thuật toán vừa trình bày nói chung là không hiệu quả.

Một số cấu trúc tôpô mạng thông dụng nhất.

- Mạng hình lưới (Mesh Network): còn được gọi là mô hình lưới trong đó các bộ xử lý $P_{(i,j)}$ được bố trí tại các nút của một mạng lưới đường truyền.
- Mạng lưới 3-chiều là tập hợp các lưới 2-chiều trong đó các nút tương ứng được kết nối. Tương tự như vậy mạng lưới k-chiều là tập hợp các lưới (k-1)-chiều với sự kết nối các bộ xử lý tương ứng.
- Mạng liên kết dịch chuyển đầy đủ (perfect shuffle connection): gọi n là số bộ xử lý, bộ xử lý P_i được nối với P_j trong đó

$$j = \begin{cases} 2i \leftarrow 0 \leq n/2 - 1 \\ 2i + 1 - n \leftarrow n/2 \leq i \leq n - 1 \end{cases}$$

- Mạng hình tháp (Pyramid Network): Mạng hình tháp được xây dựng như một cây có gốc. Gốc chứa một bộ xử lý. Ở mức tiếp theo có 4 bộ xử lý được nối với nhau theo dạng của lưới 2-chiều và cả 4 nút đều là con của gốc. Mỗi một nút ở mức 1 đều có 4 con ở mức 2. Tất cả các nút con ở mức 2 được kết nối với nhau theo dạng lưới. Tiếp tục như vậy hình tháp có thể xây dựng đến độ cao cần thiết.

2.2.4. Thuật toán song song tính tích ma trận

Bài toán tính tích ma trận giữ vị trí nổi bật trong hàng loạt các bài toán quan trọng. Nếu A là ma trận kích thước *m*x*n*, B là ma trận kích thước *n*x*p*, thì tích của hai ma trận A và B là ma trận C (ký hiệu là C = AB) có kích thước *m*x*p*. Phần tử dòng trên giao của dòng *i* và cột *j* của C ký hiệu là C(*i*,*j*) là bằng tích vô hướng của dòng thứ *i* của A với cột thứ *j* của ma trận B.

$$C(i,j) = (a_{i1}, a_{i2}, \dots, a_{in}) \begin{pmatrix} b_{1j} \\ b_{2j} \\ \vdots \\ b_{nj} \end{pmatrix} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

Thuật toán có thể mô tả trong sơ đồ sau

Thuật toán tính tích ma trận

Input: Các ma trận A và B

Output: Ma trận tích C

BEGIN

```
1. For i = 1 to m do in parallel
2.       For j = 1 to p do in parallel
3.             Evaluate C(i,j)
4.       End Parallel
5. End Parallel
END
```

Bước 3 trong thuật toán cần được chi tiết hoá. Việc tính C(*i,j*) chính là tính tích vô hướng của hai vectơ. Ta có thể thực hiện điều đó trong đoạn chương trình sau, trong đó mỗi bộ xử lý sẽ sử dụng biến địa phương tạm thời T[1:*n*] để cất giữ giá trị *A(i,k)*B(k,j)*.

```
3.1   For k = 1 to n do in parallel
3.2         T(k) = A(i,k)*B(k,j)
3.3   End parallel
3.4   P = n/2
3.5   While p>0 do
3.6         For k=1 to p do parallel
3.7               T(k) = T(2k-1) + T(2k)
3.8         End parallel
3.9         P = p/2
3.10  End while
3.11  C(i,j) = T(1)
```

- **Phân tích độ phức tạp.**

Đoạn chương trình trên gồm hai phần. Trong phần thứ nhất các giá trị A(i,k)*B(k,j) được tính trước. Sau đó, trong phần thứ hai các giá trị này sẽ được cộng dồn để thu được giá trị C(i,j). Biến địa phương tạm thời T[1:n] đươc sử dụng để thực hiện việc cộng và nhân này. Trong đoạn chương trình này, các bước 3.1 – 3.3 có thời gian tính là O(1) và đòi hỏi O(*n*) bộ xử lý. Các bước từ 3.4 đến 3.11 giống hệt thuật toán tính tổng SUM, có thời gian tính là O(log *n*) sử dụng O(*n*) bộ xử lý. Bước 3 nằm trong 2 vòng lặp song song lồng nhau, vì thế thời gian tính của thuật toán là O(log *n*) đòi hỏi O(*mnp*) bộ xử lý.

Trong trường hợp riêng khi A và B là các ma trận vuông cấp n, thuật toán có thời gian tính là O(log n) và sử dụng O(n^3) bộ xử lý. Giá trị của A(i,j) cần thiết để tính C(i,1), C(i,2), …, C(i,n). Do C(i,1), C(i,2), …, C(i,n) được tính song song tại các nhóm bộ xử lý khác nhau, A(i,j) sẽ bị đọc đồng thời bởi nhiều bộ xử lý. Do đó thuậttoán trình bày trên đòi hỏi CREW PRAM.

2.2.5. Đánh giá hiệu quả của thuật toán song song

Hiệu quả của thuật toán song song được đánh giá thông qua 3 yếu tố sau:

1. Thời gian tính (Time Complexity);
2. Số lượng bộ xử lý đòi hỏi sử dụng (Processor Complexity);
3. Cấu hình máy cần sử dụng.

Ví dụ, hiệu quả của 3 thuật toán đề nghị trong mục trước được trong bảng sau đây

Thuật toán	**Thời gian tính**	**Số lượng bộ xử lý**	**Mô hình PRAM**
Boolean-AND	$O(1)$	$O(n)$	ERCW
Boolean-AND-1	$O(1)$	$O(n)$	ERCW-ECR
Boolean-AND-2	$O(\log_2 n)$	$O(n)$	ERCW

Bảng 2.2.5: So sánh thời gian của 3 thuật toán Boolean-AND, Boolean-AND – 1, Boolean-AND - 2

Để đánh giá cận của độ phức tạp chúng ta sử dụng ký hiệu so sánh tiệm cận sau

1. $T(n) = O(f(n))$ nếu tìm được các số dương c và n_0 sao cho $T(n) < c\ f(n)$ với mọi $n > n_0$
2. $T(n) = \Omega(f(n))$ nếu tìm được các số dương c và n_0 sao cho $T(n) < c\ f(n)$ với mọi $n > n_0$
3. $T(n) = ©(f(n))$ nếu tìm được các số dương c_1, c_2 và n_0 sao cho $c_1\ f(n) < T(n) < c_2\ f(n)$ với mọi $n > n_0$

Hai đặc trưng quan trọng khác cần khảo sát khi phân tích thuật toán song song là tăng tốc (speedup) và hiệu suất (efficiency).

- **Tăng tốc và Hiệu suất**. Xét bài toán mà đối với nó thuật toán tuần tự tốt nhất có thời gian tính là T_s. Giả sử ta có thuật toán song song giải bài toán đó có thời gian tính là T_p với bộ xử lý là P. Ta định nghĩa

  Tăng tốc = T_s/T_p;

  Hiệu suất = $T_s/(PT_p)$.

Tăng tốc nhiều nhất là bằng số lượng bộ xử lý, và chúng ta luôn cố tìm cách đạt được tăng tốc gần với số bộ xử lý. Trên thực tế chúng ta cố gắng đạt tăng tốc cao nhất đối với một số lượng hạn chế bộ xử lý hiện hữu. Hiệu suất cung cấp cho chúng ta thước do hiệu quả sử dụng các bộ xử lý. Đo độ hiệu quả trong lĩnh vực song song càng bị phức tạp bởi đòi hỏi giải đáp "Liệu việc giải bài toán ứng dụng sẽ nhanh hơn bao nhiêu lần khi nó được thực hiện trên máy tính song song?". Nghĩa là, liệu chúng ta thu được lời ích gì khi sử dụng song song hoá? Điều đó sẽ được giải đáp thông qua khái niệm tăng tốc.

Tăng tốc = thời gian tính tuần tự / thời gian tính song song.

Hiện tại có rất nhiều cách định nghĩa thời gian tính tuần tự và song song. Điều đó dẫn đến 5 định nghĩa khác nhau về tăng tốc. Đó là:

- Tăng tốc tương đối (relative speedup);
- Tăng tốc thực tế (real speedup);
- Tăng tốc tuyệt đối (absolute speedup);
- Tăng tốc tiệm cận thực tế (asymptopic real speedup);
- Tăng tốc tiệm cận tương đối (asymptopic relative speedup)

Sahni và Thanvantri đã khảo sát kỹ các độ đo này trong bài báo "*Performance Metrices: Keeping the Focus on Runtime*, IEE-PDT, 1996, 43-46". Dưới đây là sơ lược về các khái niệm này.

- **Tăng tốc tương đối (relative speedup).** Thời gian tính tuần tự được xác định như thời gian tính của thuật toán song song khi nó chạy trên một bộ xử lý của máy tính song song. Như vậy, tăng tốc tương đối thu được từ thuật toán A khi giải bài toán với bộ dữ liệu *I* kích thước *n* sử dụng *p* bộ xử lý sẽ là

$$Tăng\ tốc\ tương\ đối\ (n,p) = \frac{Thời\ gian\ giải\ I\ sử\ dụng\ thuật\ toán\ A\ với\ 1\ bộ\ xử\ lý}{Thời\ gian\ giải\ I\ sử\ dụng\ thuật\ toán\ A\ với\ p\ bộ\ xử\ lý}$$

Tăng tốc tương đối của hệ thống song song không phải là số cố định. Chính xác hơn, nó phụ thuộc vào kích thước của bài toán và số lượng bộ xử lý. Tăng tốc tương đối là hàm của *n* và *p*. Cố định *n* chúng ta có thể vẽ đường cong biểu diễn sự phụ thuộc của tăng tốc tương đối vào số lượng bộ xử lý *p*. Điều đó cho phép phân tích hiệu quả của thuật toán khi số lượng bộ xử lý gia tăng. Tương tự như vậy chúng ta có thể cố định *p* và vẽ đường cong biểu thị sự phụ thuộc của tăng tốc tương đối vào *n*. Điều đó cho phép khảo sát dáng điệu của thuật toán khi kích thích dữ liệu tăng. Sử dụng 2 đường cong này chúng ta có thể tìm ra điểm mà tại đó tăng tốc tương đối là lớn nhất. Ta gọi đó là tăng tốc tương đối lớn nhất. Tăng tốc tương đối trung bình, tăng tốc tương đối nhỏ nhất, tăng tốc tương đối mong đợi được xác định một cách tương tự.

- **Tăng tốc thực tế (real speedup).** Trong độ đo này, thời gian tính song song được so sánh với thời gian tính của thuật toán tuần tự nhanh nhất đối với vấn đề ứng dụng cần giải quyết. Thời gian tính của thuật toán tuần tự nhanh nhất trên một bộ xử lý của máy tính song song được sử dụng. Do đối với nhiều bài toán hoặc là ta còn chưa biết thuật toán nhanh nhất, hoặc là không có thuật toán nào là nhanh nhất đối với mọi bộ dữ liệu, nên thời gian của thuật toán thường được sử dụng trong thực tiễn ứng dụng sẽ được sử dụng thay cho thời gian tính của thuật toán nhanh nhất. Tăng tốc thu được sẽ được gọi là tăng tốc thực tế.

$$Tăng\ tốc\ thực\ tế\ (n,p) = \frac{Thời\ gian\ giải\ sử\ dụng\ thuật\ toán\ tuần\ tự\ tốt\ nhất}{Thời\ gian\ giải\ sử\ dụng\ p\ bộ\ xử\ lý}$$

Ta có thể định nghĩa tăng tốc thực tế lớn nhất, tăng tốc thực tế trung bình, tăng tốc thực tế nhỏ nhất, tăng tốc thực tế mong đợi giống như những định nghĩa các khái niệm này đối với tăng tốc tương đối.

- **Tăng tốc tuyệt đối (absolute speedup):** Trong một cách định nghĩa tăng tốc khác, thời gian tính song song được so sánh với thời gian tính của thuật toán tuần tự nhanh nhất trên máy tính tuần tự nhanh nhất. Cũng giống như trong trường hợp tăng tốc thực tế, so sánh được tiến hành đối với thuật toán được sử dụng trong thực tiễn.

$$Tăng\ tốc\ tuyệt\ đối\ (n,p) = \frac{Thời\ gian\ giả\ i\ sử\ dụng\ thuật\ toán\ tuần\ tự\ tốt\ nhất\ trên\ bộ\ xử\ lý\ nhanh\ nhất}{Thời\ gian\ giả\ i\ sử\ dụng\ thuật\ toán\ A\ với\ p\ bộ\ xử\ lý}$$

Định nghĩa tăng tốc tuyệt đối được mở rộng thành tăng tốc tuyệt đối lớn nhất, tăng tốc tuyệt đối nhỏ nhất, tăng tốc tuyệt đối trung bình, tăng tốc tuyệt đối mong đợi.

- **Tăng tốc tiệm cận thực tế (asymptopic real speedup):** Giả sử *S(n)* là độ phức tạp tiệm cận của thuật toán tuần tự tốt nhất đối với bài toán và *P(n)* là độ phức tạp tiệm cận của thuật toán song song A với giả thiết là máy tính song song có đủ số bộ xử lý mà thuật toán song song đòi hỏi, thời gian tính tiệm cận thực tế được xác định bởi

$$Tăng\ tốc\ tiệm\ cận\ thực\ tế\ (n,p) = \frac{S(n)}{T(n)}$$

- **Tăng tốc tiệm cận tương đối (asymptopic relative speedup).** Khái niệm này khác với tăng tốc tiệm cận thực tế ở chỗ trong việc xác định độ phức tạp tuần tự ta sử dụng độ phức tạp thời gian tiệm cận của thuật toán song song khi chạy trên một bộ xử lý.
- **Chi phí tăng tốc hoá (Cost Normalized Speedup).** Thông thường ngoài việc cần biết hệ thống song song thực hiện nhanh như thế nào, ta cũng cần biết cách đo chi phí phải trả để đạt được điều đó. Vì thế, ta đưa ra định nghĩa *Chi phí tăng tốc chuẩn hoá (CNS):*

$$CNS\ (n,p) = \frac{Speedup\ (n,p)}{(Chi\ phí\ cho\ hệ\ thống\ song\ song)/(Chi\ phí\ cho\ hệ\ thống\ tuần\ tự)}$$

Để tính CNS, ngoài tăng tốc, chúng ta cần biết chi phí cho các hệ thống song song và tuần tự. Chi phí cho hệ thống song cần tính đến cả chi phí cho phần cứng lẫn phần mềm. Và rõ ràng chi phí này phụ thuộc vào rất nhiều yếu tố và việc tính nó

không phải là vấn đề dễ dàng. Chi phí cho hệ thống tuần tự phụ thuộc vào dạng tăng tốc mà ta sử dụng. Như vậy, nếu biết loại tăng tốc nào được chọn, chúng ta biết hệ thống tuần tự nào được sử dụng.

- **Hiệu suất.** Hiệu suất là độ do rất gần với tăng tốc. Như chúng ta đã thấy, theo định nghĩa hiệu suất là tỷ số giữa tăng tốc và số lượng bộ xử lý *p*. Phụ thuộc vào loại tăng tốc được lựa chọn mà ta có các khái niệm hiệu suất khác nhau.

Còn nhiều cách định nghĩa hiệu suất khác có thể thấy trong tài liệu về tính toán song song. Chẳng hạn, có tác giả định nghĩa hiệu suất như là tỷ số *wa/we*, trong đó *wa* là công việc đạt được (accomplish work) bởi thuật toán song song còn *we* (expended work) là công việc tiêu sài bởi thuật toán. Ta xác định công việc đạt được bởi thuật toán song song là công việc mà một thuật toán tuần tự tốt nhất đạt được, còn công việc tiêu sài bởi thuật toán song song là tích của thời gian tính song song, tốc độ (*s*) của một bộ xử lý, và số bộ xử lý (*p*). Ta có

we = (thời gian tính song song) x *s* x *p*;

wa = (thời gian tính tuần tự tốt nhất) x *s*

we/wa = (thời gian tính tuần tự tốt nhất) / (*p* x thời gian tính song song)

Hiệu suất này chính là bằng tăng tốc thực tế chia cho *p*. Vì vậy, *wa/we* chính là hiệu suất thực tế. Tương tự như vậy, nếu wa được xác định như là công việc thực hiện bởi thuật toán song song khi nó chạy trên một bộ xử lý, thì *wa/we* chính là hiệu quả tương đối (tức là bằng tăng tốc tương đối chia cho số bộ xử lý).

Trong một cách định nghĩa khác, hiệu suất được định nghĩa thông qua việc xác định công việc vô ích (wasted work) *ww* = *we* – *wa*. Khi đó hiệu suất là:

$$\text{Efficiency} = \frac{wa}{we} = \frac{wa}{ww + wa} = \frac{1}{1 + ww/wa}$$

Tóm lại, khi bàn về hiệu quả của thuật toán song song chúng ta cần quan tâm đến các thông số sau:

- T_s = thời gian tính của thuật toán tuần tự tốt nhất

- T_p=thời gian tính của thuật toán song song sử dụng p bộ xử lý
- Tăng tốc $S_p=T_s/T_p$
- Hiệu quả sử dụng $E_p = S_p/p$

Chú ý rằng

- Nếu hiệu quả của thuật toán song song là 100% (E_p=1) thì ta có tăng tốc là phụ thuộc tuyến tính vào p ($S_p = p$)
- Hiệu quả sử dụng không bao giờ đạt 100% bởi vì:
  - Chi phí phải trả cho việc đồng bộ hoá hoặc truyền thông giữa các bộ xử lý;
  - Sự phân tải gần tối ưu giữa các bộ xử lý.

Chúng ta xét luật Amdahl để phân tích hiệu quả của thuật toán song song

Luật Amdahl

Luật nói lên rằng sự cải thiện hiệu quả thu được bằng cách cài đặt song song bị giới hạn bởi bộ phận không thể song song hoá. Xét ứng dụng đòi hỏi T đơn vị thời gian khi được thực hiện tuần tự trên một bộ xử lý. Khi ứng dụng được song song hoá, ta giả sử rằng thông số α thể hiện bộ phận không thể song song hoá được (bộ phận tuần tự)

Với giả thiết là p bộ xử lý được sử dụng trong tính toán song song, thời gian thực hiện được xác định bởi biểu thức sau (giả thiết rằng trong phần của ứng dụng được song song hoá, tăng tốc đạt mức tối đa):

$$T_p = \left(\alpha + \frac{1-\alpha}{p}\right).T$$

Luật Amdahl: Tăng tốc có thể đạt được bởi sử dụng p bộ xử lý là:

$$S_p = \frac{T}{T_p} = \frac{1}{\alpha + \frac{1-\alpha}{p}}$$

Để ý rằng: Nếu ta giả thiết rằng bộ phận tuần tự α (còn gọi là bộ phận Amdahl) là cố định, thì tăng tốc, ngay cả khi có thể sử dụng một số lượng không hạn chế bộ xử lý, bị

chặn bởi 1/α. Chẳng hạn, nếu α=10%, thì tăng tốc cực đại là 10, ngay cả khi chúng ta có thể sử dụng một số lượng không hạn chế bộ xử lý.

Bình luận thêm về luật Amdahl

- Bộ phận α trong luật Amdahl trong thực tế là phụ thuộc vào kích thước bài toán *n* và số lương bộ xử lý *p*.
- Một thuật toán song song hiệu quả phải có:
- $\alpha(n,p) \to 0\ khi\ n \to \infty$
- Trong những tính huống như vậy, ngay cả khi số lượng bộ xử lý *p* là cố định, chúng ra vẫn có thể đạt được tăng tốc tuyến tính nhờ việc chọn kích thước dữ liệu lớn:

$$S_p = \frac{T}{T_p} = \frac{p}{1 + (p+1)\alpha(n,p)} \to p\ khi\ n \to \infty$$

- Trong thực tế, kích thước của bài toán mà ta có thể xử lý được đối với một vấn đề ứng dụng là bị chặn bởi bộ nhớ của máy tính song song.

Để minh hoạ cho luật Amdahl, ta xét ví đơn giản sau đây.

Ví dụ: Tính số Pi (π)

Xét thuật toán song song để tính giá trị của π nhờ việc tính tích phân số

$$\pi = \int_0^1 \frac{4}{1 + x^2} dx$$

Thuật toán song song

- Mỗi bộ xử lý sẽ tính toán trên tập gồm khoảng *n/p* điểm được phân bố cho mỗi bộ xử lý một cách có chu kỳ.
- Cuối cùng ta giả sử rằng giá trị của π sẽ được gom tụ lại từ *p* bộ xử lý bởi cơ chế đồng bộ.

Thuật toán song song tính Pi

```
parallelComputepi (){
id = my_processor_id ();
nprocs = number_of_processors ();
h = 1.0/n;
sum = 0.0;
for (int i = id; i < n; i = i + nprocs){
x = h*(i - 0.5);
sum = sum + 4.0/(1 + x * x);
}
localpi = sum*h;
use_tree_based_combining_for_critical_section ();
pi = pi _ localpi;
end_critical_section();
}
```

Phân tích thuật toán song song tính Pi

- Giả sử rằng việc tính π được thực hiện trên *n* điểm.
- Thuật toán tuần tự đòi hỏi thực hiện 6 phép toán (2 phép nhân, 1 phép chia, 3 phép cộng) đối với mỗi điểm trên trục X. Vì vậy, đối với *n* điểm, số lượng phép toán mà thuật toán tuần tự phải thực hiện là:

$$T_s = 6n$$

- Thuật toán song song sử dụng *p* bộ xử lý, mỗi bộ xử lý tính toán trên tập gồm *m* điểm được phân bố cho các bộ xử lý một cách có chu kỳ.
- Rõ rang $m \leq 1 + n/p$ (đúng cả cho khi *p* không chia hết *n*). Thời gian tính của thuật toán song song để tính các giá trị địa phương của π là:

$$T_p = 6m = 6\frac{n}{p} + 6$$

- Để gom tụ các giá trị địa phương của π ta sử dụng mô hình cây và điều đó đòi hỏi thời gian là $\log_2 p$
- Tổng thời gian của thuật toán song song kể cả tính toán lẫn gom tụ là:

$$T_p = 6\frac{n}{p} + 6 + \log p$$

- Tăng tốc của thuật toán song song là:

$$S_p = \frac{T_s}{T_p} = \frac{6n}{6\frac{n}{p} + 6 + \log p}$$

- Bộ phận Amdahl đối với thuật toán song song này có thể xác định nhờ viết lại đẳng thức trên dưới dạng:

$$S_p = \frac{p}{1 + \frac{p}{n} + \frac{p\log p}{6n}} \quad => \quad S_p = \frac{p}{1 + (p-1)\alpha(n,p)}$$

- Vì vậy bộ phận Amdahl α(*n*,*p*) là:

$$\alpha(n,p) = \frac{p}{(p-1)n} + \frac{p\log p}{6n(p-1)}$$

- Thuật toán song song vừa trình bày là hiệu quả, bởi vì:

$$\alpha(n,p) \to 0 \; khi \; n \to \infty \; đối \; với \; p \; cố \; định.$$

2.3. Tính toán song song trong .NET và minh họa

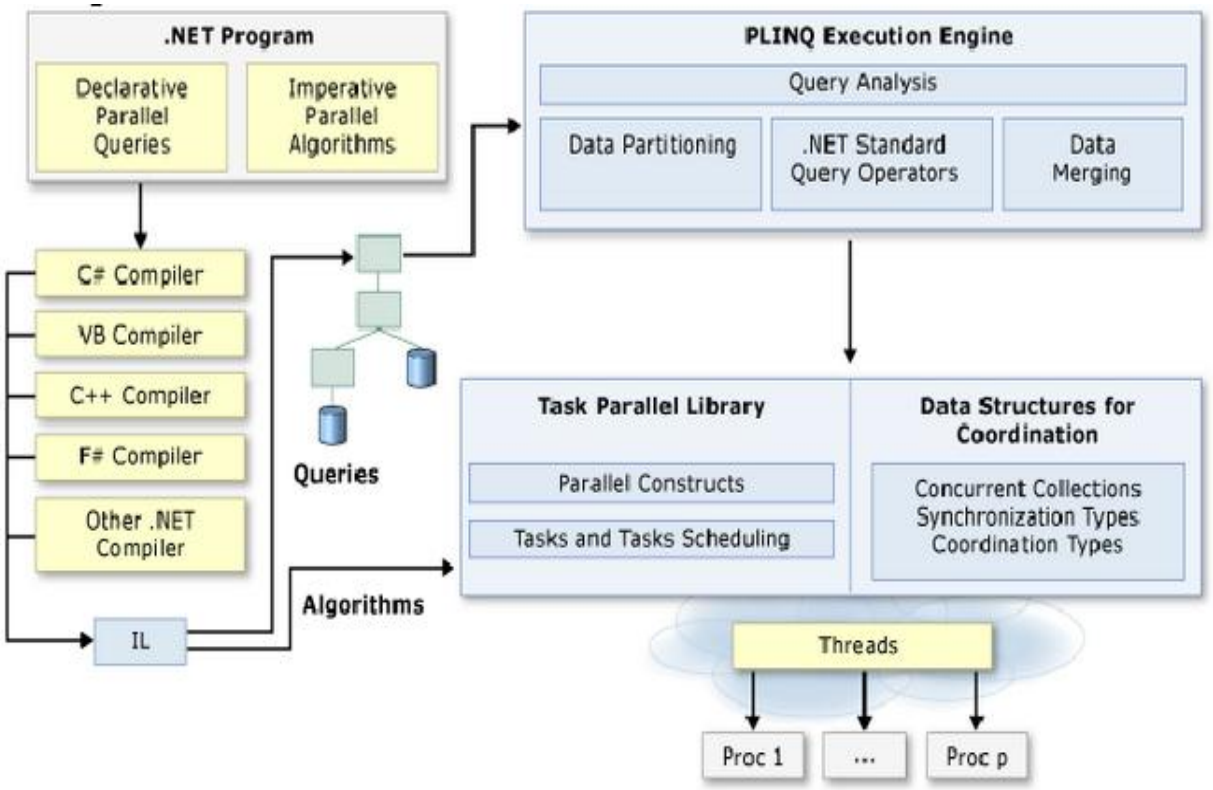
Hệ thống cần xử lý các bước :

- Phần chia chương trình thành các công việc con (các task)
- Phân tích sự phụ thuộc giữa các task
- Định thời gian làm việc cho các task

O Ưu điểm của tính toán song song trong .NET

- Đạt tốc độ cao hơn so với lập trình theo mô hình lập trình tuần tự.
- Tận dụng được tối đa hiệu năng mà các bộ xử lý đa nhân đa lõi đang được sử dụng.

- Nhược điểm của tính toán song song trong .NET:
  - Việc lập trình khó khăn,phức tạp..



Hình 2.3: Mô tả thuật toán song song trong .NET Framework 4.0

Ý nghĩa từ hình 2.3 có thể giải thích ngắn gọn như sau:

- Chương trình .NET Framwork 4.0 được viết trên các ngôn ngữ khác nhau như C#, VB, C++, F#, ...

- Chương trình cần lấy dữ liệu từ database, nó sẽ truy vấn đến database thông qua LINQ, sau đó được đưa đến các thư viện chứa các Task (Task Parallel Library – viết tắt là TPL). Tại đây chương trình sẽ được chia nhỏ thành các tác vụ và được lập lịch hoạt động (Task Sheduling). Kết quả ta được các Threads và các Threads này sẽ được xử lý đồng thời.
- Nếu chương trình không cần lấy dữ liệu từ database, ta sẽ thao tác trực tiếp và trực tiếp tạo ra các task.

2.3.1. Task

Như đã trình bày ở trên, mô hình lập trình song song sẽ chia nhỏ chương trình ra thành các tác vụ nhỏ (Task). Để sử dụng Task ta cần thêm hai không gian tên:

- System.threading
- System.threading.task

.NET Framework 4.0 cung cấp cho ta rất nhiều cách để khai báo và khởi tạo một Task:

```
// use an Action delegate and a named method
  Task task1 = new Task(new Action(printMessage));
  // use a anonymous delegate
  Task task2 = new Task(delegate { printMessage(); });
  // use a lambda expression and a named method
  Task task3 = new Task(() => printMessage());
  // use a lambda expression and an anonymous method
  Task task4 = new Task(() => { printMessage(); });
```

Mỗi một Task khi khai báo và khởi tạo sẽ được máy tính thực hiện song song đồng thời. Tuy nhiên ta cũng có thể định thời gian hoạt động cho các task.

- Hủy bỏ một task: ta sử dụng phương thức CancellationTokenSource để hủy bỏ một task. Việc hủy bỏ một tác vụ được sử dụng đến khi chương trình đang hoạt động mà ta muốn can thiệp và dừng 1 tác dụng nào đó lại.
- Tác vụ chờ: khi ta không muốn một tác vụ nào đấy thực hiện ngay, mà chờ đợi 1 khoảng thời gian nhất định. Ta sẽ sử dụng phương thức Thread.Sleep() và Thread.SleepWait().
- Ta cũng có thể bắt 1 Task thực hiện sau khi tất cả các task khác đã thực hiện xong hoặc chờ đợi sau bao lâu bằng việc sử dụng phương thức Task.Wait();

2.3.2. Vòng lặp song song (Parallel Loops)

Với chương trình tuần tự, khi các vòng lặp thông thường như for, while, do while chương trình sẽ thực hiện tuần tự từng vòng lặp cho đến khi kết thúc, trong khi đó CPU đang rảnh rỗi thì vẫn phải đợi thực hiện xong một vòng lặp rồi mới thực hiện vòng lặp tiếp theo, vì thế nó sẽ làm tốn thời gian của hệ thống.

Mô hình lập trình song song sử dụng các vòng lặp song song (Parallel Loops) để khắc phục nhược điểm của lập trình tuần tự như đã nêu trên, các vòng lặp ở đây sẽ được CPU thực hiện đồng thời, làm tăng hiệu năng tính toán và giảm thời gian của hệ thống.

- Phương thức Parallel.For() với 3 tham số đầu vào, tham số đầu vào chỉ chỉ số bắt đầu tác vụ, tham số thứ 2 chỉ số kết thúc và mục thứ 3 chỉ tác vụ cần thực hiện.

```
Parallel.For(0, 10, index =>
{
 Console.WriteLine("Task ID {0} processing index: {1}",
 Task.CurrentId, index);
  });
```

- Vòng lặp ForEach(): sử dụng phương thức Parallel.ForEach(), vòng lặp này linh hoạt hơn so với vòng lặp trước nhờ việc cho phép cải biên giá trị đầu vào. Xét ví dụ:

```
List<string> dataList = new List<string> {
        "the", "quick", "brown", "fox", "jumps", "etc" };
 Parallel.ForEach(dataList, item =>
{
 Console.WriteLine("Item {0} has {1} characters",
      item, item.Length);
 });
```

Cấu trúc câu lệnh trên khá giống với cấu trúc câu lệnh của lập trình tuần tự. Tuy nhiên ta không nên hiểu lầm các vòng lặp này sẽ được thực hiện một các tuần tự. Nó chỉ biểu diễn dưới dạng này. Còn việc thực hiện song song là do trình biên dịch và CPU đảm nhiệm.

2.3.3. Parallel LINQ

Parallel LINQ được biết đến như LINQ thông thường nhưng điểm khác biệt là việc thực hiện song song các câu truy vấn LINQ đến cơ sở dữ liệu. Từ đó các tác vụ được xử lý đồng thời.

2.4. Thuật toán Floyd – Warshall và bài toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh trên đồ thị

Thuật toán Floyd – Warshall hay còn gọi là Floyd, Roy – Floyd là thuật toán tìm đường đi ngắn nhất giữa mọi cặp đỉnh trên đồ thị sau một lần chạy thuật toán. Một tính chất nữa là Floyd – Warshall có thể chạy trên đồ thị có các cạnh có trọng số có thể âm. Tuy nhiên lưu ý là đồ thị không được có vòng (cycle) nào có tổng các cạnh là âm, nếu có vòng như vậy ta không thể tìm đường đi ngắn nhất (mỗi lần đi qua vòng này, độ dài quãng đường lại giảm, nên ta có thể đi vô hạn lần).

Thuật toán Floyd – Warshall so sánh tất cả các đường đi có thể giữa từng cặp đỉnh. Nó là một dạng của quy hoạch động (Dynamic Programming). Nếu đặt hàm adj(i, j, k) là đường đi ngắn nhất từ i đến j, chỉ dùng đến các đỉnh trong tập {1, 2, 3, ..., k}. Giả sử muốn tính adj(i, j, k + 1). Với mỗi cặp đỉnh i và j, đường đi ngắn nhất có thể là: (1) đường đi chỉ sử dụng các đỉnh trong tập {1,..., k} hoặc (2) đường đi từ i đến k + 1 rồi từ k + 1 đến j, cũng chỉ sử dụng các đỉnh trong tập {1, ..., k}. Do vậy:

Trường hợp cơ bản: adj(i, j, 0) = w(i, j);

Đệ quy: `adj(i,j,k+1) = min{adj(i,j,k), adj(i,k+1, k) + adj(k+1, j, k)}`

Đây là đoạn pseudocode của Floy - Warshall

```
for (int i = 0; i < n; i++)
{
for (int j = 0; j < n; j++)
   {
for (int k = 0; k < n; k++)
            {
adj[i][j] = min(adj[i][j], adj[i][k] + adj[k][j]);

            }
     }
 }
```

Độ phức tạp của thuật toán Floyd là $O(n^3)$.

Đầu vào của thuật toán là đồ thị liên thông G = (V, E), V = {1, 2, 3, ... ,n}, có trọng số w(i, j) > 0 với mọi cung (i, j). Đầu ra của thuật toán là ma trận D = [d(i, j)] là chiều dài đường đi ngắn nhất từ i đến j với mọi cặp (i, j).

Dựa vào thuật toán trên, tôi áp dụng vào thuật toán tìm dòng tích lũy cho bài toán của mình. Phương pháp của tôi như sau:

Phương pháp:

Bước 1: Khởi tạo ma trận xuất phát (D_0): $D_0 = [D_0(i, j)]$

Trong đó:

- d_0 (i, j) = w(i, j) nếu tồn tại cung (i, j). Cụ thể nếu tồn tại hướng (p, q) bằng 1 trong 8 hướng có thể chảy là hướng số 1, 2, 4, 8 , 16, 32, 64, 128 thì trường hợp này hướng (p, q) ta gán bằng 1.
- d_0 (i, j) = +∞ nếu không tồn tại cung (i, j). Ngược lại với trường hợp trên thì hướng (q, p) ta gán bằng +∞.

Đặc biệt nếu không có khuyên tại i thì $d_0(i, i)$ cũng bằng +∞.

Gán k = 0;

Bước 2: Kiểm tra kết thúc: Nếu k = n thì kết thúc. D = D_n là ma trận chứa vị trí tích lũy. Ta chỉ cần chia vị trí lấy tọa độ: (i / số cột) là tọa độ dòng và chia lấy dư (i % số

cột) để lấy tọa độ cột. sau đó ta chỉ cần đếm lại số ô để lấy kết quả tích lũy. Ngược lại k = k + 1. Sang bước 3.

Bước 3: Tính ma trận D_k theo D_{k-1}

Với mọi cặp (i, j) i = 1.... n, j = 1.... n. Thực hiện:

Nếu $d_{k-1}(i, j) > d_{k-1}(i, k) + d_{k-1}(k, j)$ thì đặt $d_k(i, j) = d_{k-1}(i, k) + d_{k-1}(k, j)$. Ngược lại đặt $d_k(i, j) = d_{k-1}(i, j)$.

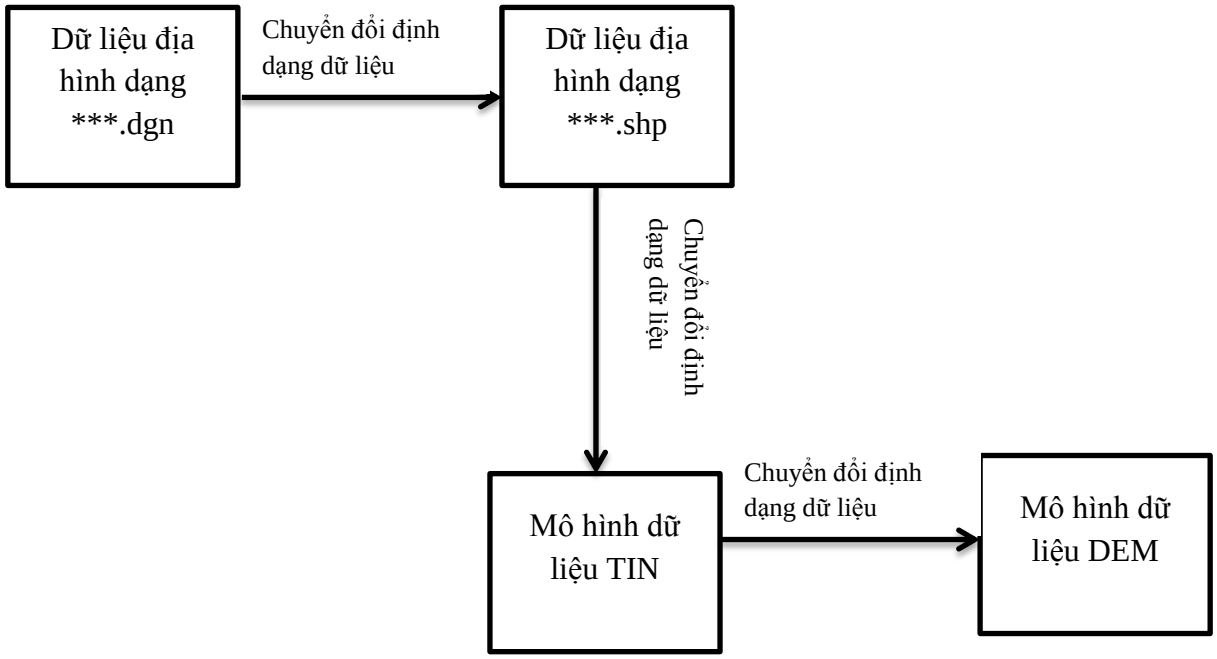
Quay lại bước 2 và kết thúc thuật toán.

PHẦN 3. DỮ LIỆU, NỘI DUNG VÀ PHƯƠNG PHÁP NGHIÊN CỨU

3.1. Dữ liệu

3.1.1. Mô hình dữ liệu DEM

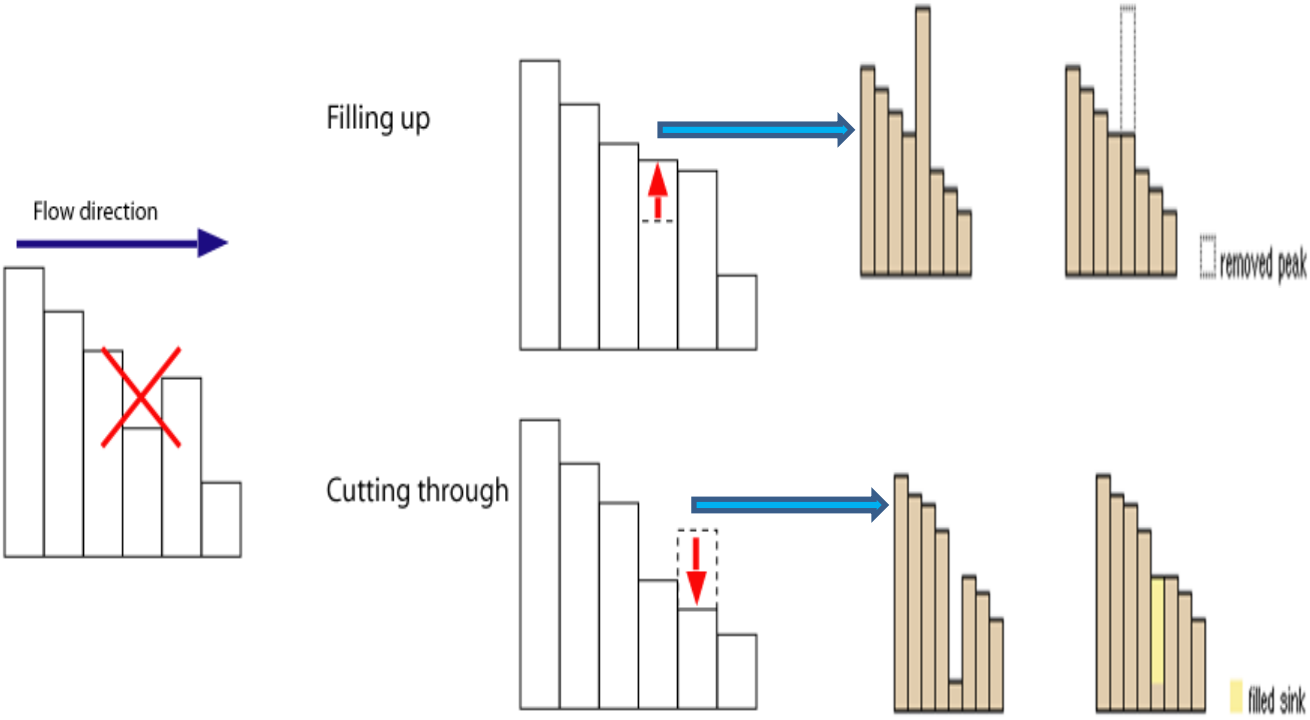
Mô hình DEM có thể được xây dựng từ bản đồ địa hình, kích thước ô lưới phụ thuộc vào quy mô lưu vực và tỷ lệ bản đồ địa hình, tuy nhiên, thường dao động trong khoảng từ 10 – 100m. Trong trường hợp nếu mô hình số hóa độ cao đã có sẵn thì trước khi sử dụng cần kiểm tra tính phù hợp của các thông tin cơ bản, bao gồm kích thước ô lưới và hệ quy chiếu.



Hình 3.1.1.1. Quy trình chuyển đổi DEM

Mô hình dữ liệu DEM cũng có thể tải về từ phần mềm Global Mapper với dữ liệu số được cập nhật trên toàn thế giới.

Sau khi đã có mô hình DEM ta cần hiệu chỉnh một số điểm lỗi về những điểm lồi hoặc lõm không mong muốn như hình dưới đây.

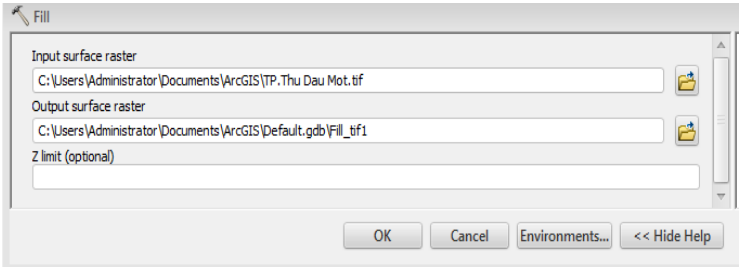


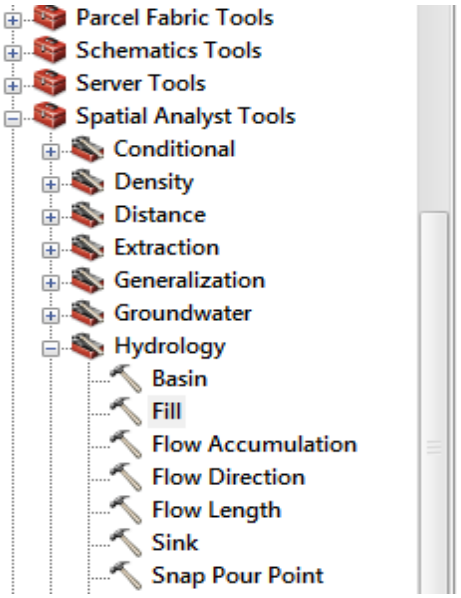
Hình 3.1.1.2: Những điểm lỗi có thể có của DEM

- Sink: phải lấp đầy => filled sink
- Peak: phải xóa đi => removed peak

Để làm điều này ta dùng hàn "Fill" trong công cụ Hydrology của ArcGis:

Xuất hiện bảng:



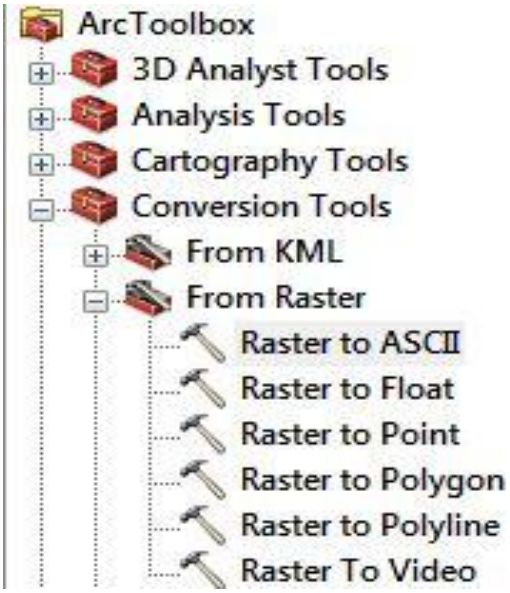


Hình 3.1.1.3: Hàm fill trong ArcToolbox

- Trong mục input suface raster ta nhấn  và chọn dữ liệu DEM cần hiệu chỉnh.
- Tiếp theo phần output suface raster ta cũng nhấn vào  và chọn vị trí lưu dữ liệu DEM đã hiệu chỉnh và đặt tên cho nó. Tiếp theo nhấn OK để bắt đầu hiệu chỉnh DEM.

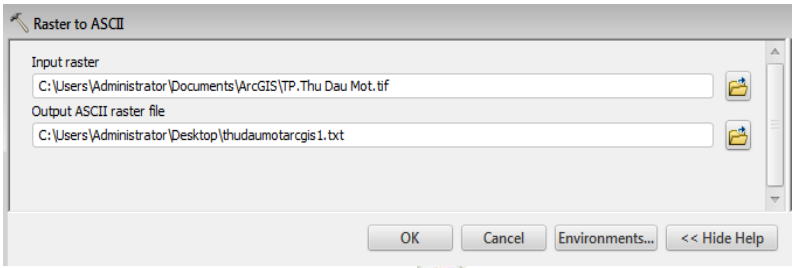
3.1.2. File text độ cao

Mỗi pixel trong mô hình dữ liệu DEM cho ta độ cao số của lưu vực tại điểm đó. Và để lấy độ cao đó ra ta dùng công cụ Raster to ASCII trong Arc toolbox của ArcGis.



Hình 3.1.2.1: Công cụ Raster to Ascii trong ArcToolbox

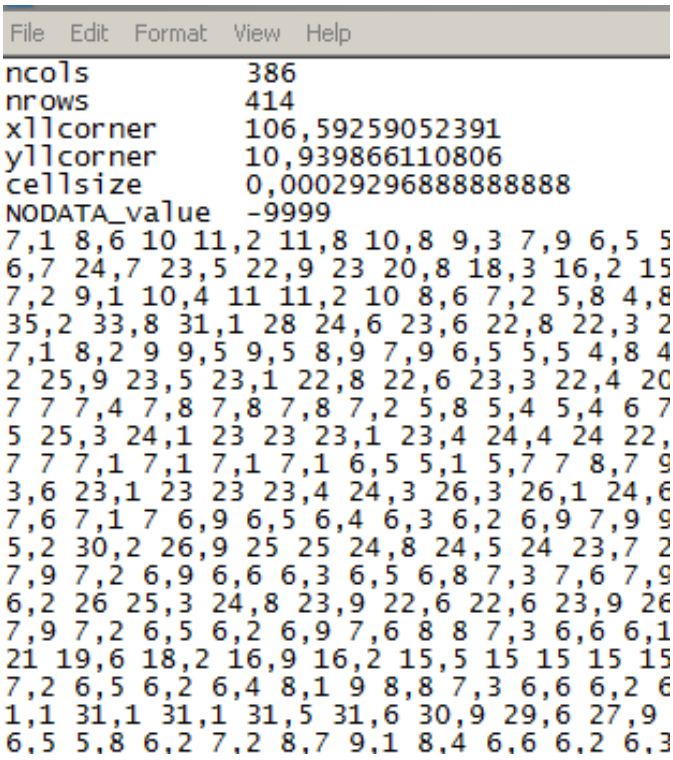
Xuất hiện bảng:



- Trong mục input raster ta nhấn  và chọn dữ liệu DEM cần xử lý.
- Tiếp theo phần output ASCII raster file ta cũng nhấn vào  và chọn vị trí lưu dữ liệu độ cao dạng text file và đặt tên cho nó. Tiếp theo nhấn OK

Kết quả ta được text file với các thông tin sau:

- Với ncols là số cột, nrows số dòng.
- Xllcorner, yllcorner: là tọa độ ví trí điểm nằm ở góc dưới bên trái của DEM.
- Cellsize: là kích thước của mỗi ô.
- NODATA_value: là những giá trị không được tính được gán bằng "-9999" hay nói các khác đó chính là ranh giới của lưu vực.
- Các số bên dưới mô tả độ cao của DEM ở từng vị trí và được phân các nhau bằng khoảng trắng.



Hình 3.1.2.2: Text file dùng để xử lý

3.2. Thuật toán định dòng chảy trên bề mặt địa hình

3.2.1. Giới thiệu thuật toán phân tích dòng chảy D8

3.2.1.1. Xác định hướng dòng chảy

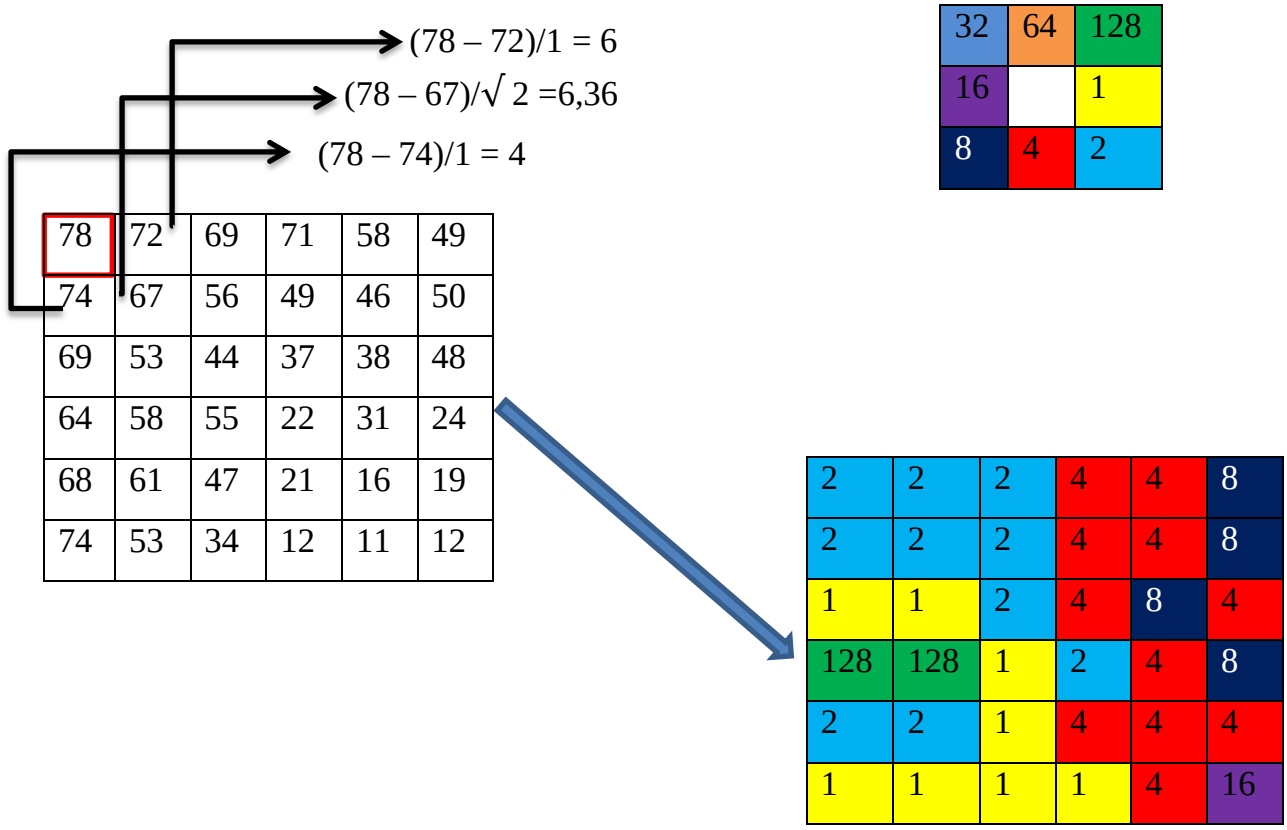
Hướng dòng chảy của một pixel bất kỳ được so sánh trên cơ sở so sánh độ cao chênh của điểm đó với 8 điểm xung quanh.

$$\Delta \mathbf{h}_i = (H - H_i)/ D_i$$

Trong đó Δh_i là độ cao chênh. H là độ cao tại pixel cần xác định hướng dòng chảy, H_i là độ cao của pixel lân cận, D_i là khoảng cách giữa 2 pixel trên (1 hoặc $\sqrt{2}$)

Hướng dòng chảy được xác định là hướng tới điểm có độ cao chênh là lớn nhất. Quá trình tính toán được lặp lại để xác định hướng dòng chảy cho toàn bộ các điểm trong lưu vực.

Để đơn giản chúng ta cùng xét bảng ví dụ sau:



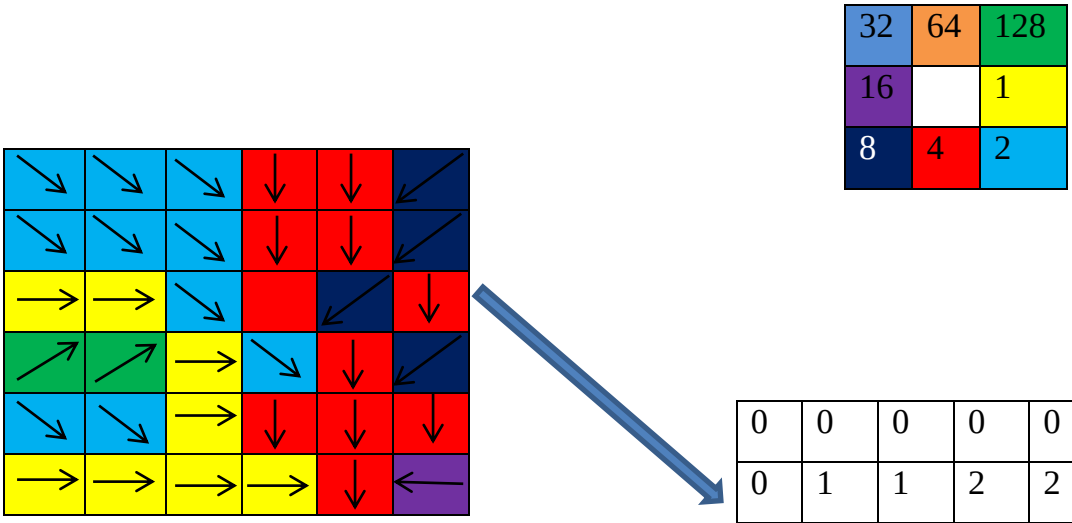
Bảng 3.2.1.1: Hướng dòng chảy tính trên lưu vực

Vậy theo thuật toán D8 ta biết được ô 78 chảy về ô 67 (có giá trị lớn nhất trong các ô xung quanh) nên giá trị sẽ là 2, tương tự ta tính tất cả các ô còn lại được bảng 3.2.1.1)

3.2.1.2. Tính toán sự tích lũy dòng chảy

Sự tích lũy dòng chảy cho một ô nào đó trong khu vực trên nền mô hình DEM được xác định bằng cách tính tổng số ô lưới tập trung nước về ô đó theo hướng dòng chảy.

Xét tiếp ví dụ vừa rồi, khi đã tính được hướng của từng ô thì nhiệm vụ tính toàn tích lũy dòng chảy trở nên đơn giản hơn, ngồi cộng tất cả các ô có chảy về ta có kết quả như bảng 3.2.1.2



0	0	0	0	0	0
0	1	1	2	2	0
0	3	7	5	4	0
0	0	0	20	0	1
0	0	0	1	24	0
0	2	4	7	35	1

Bảng 3.2.1.2: Sự tích lũy dòng chảy trên lưu vực

Kết quả ta tìm được bảng tích lũy 3.2.1.2, các con số thể hiện sự tích lũy dòng chảy ở từng ô, mỗi ô có tất cả bao nhiêu ô chảy về.

3.2.2. Giới thiệu các công cụ tìm dòng trong ArcGIS

3.2.2.1. ArcSWAT

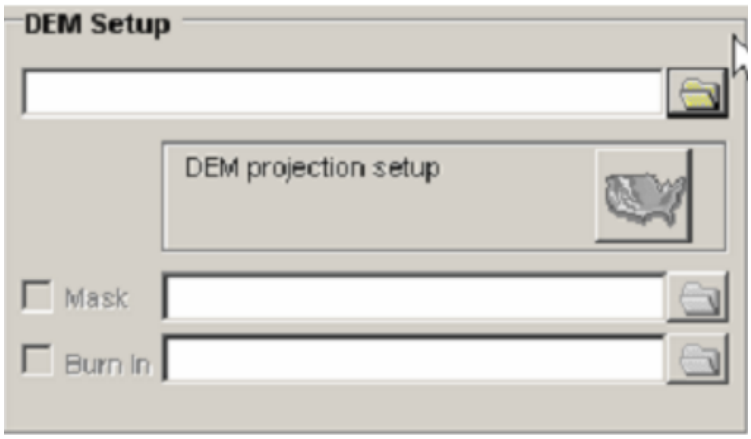
- Tổng quan
  - Swat là tên viết tắt của cụm từ "Soil and Water Assessment Tool"
  - Mục đích: Mô phỏng chất và lượng tài nguyên nước mặt, nước ngầm. Dự báo tác động môi trường của hoạt động sử dụng đất đai và biến đổi khí hậu.
- Tiến trình tìm dòng trong ArcSWAT

Trong ArcGis mở menu "SWAT Project Setup" nhấn chuột vào dòng lệnh "New Swat Project". Trong cửa sổ giao diện Project Setup, đặt Project Directory chỉ đến thư mục trong đĩa cứng. Mục "Swat project geodatabase" sẽ tự động chuyển thành "lakefork.mdb" và cơ sở địa lý dạng raster sẽ có tên "RasterStore.mdb". mục Swat

Parameter sẽ tự kết nối đến cơ sở dữ liệu. Sau đó nhấn OK sẽ xuất hiện bảng với các mục sau:

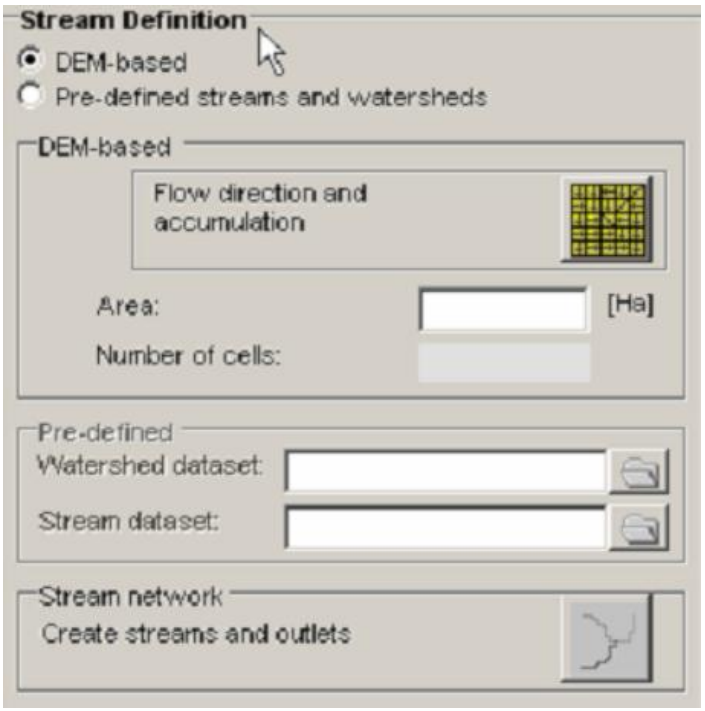
Mục DEM Setup:

- Load the DEM: Lấy DEM từ ổ đĩa hay từ ArcMap vào chương trình.
- DEM projection setup: Khai báo đơn vị độ cao thành mét.
- Define Mask: Định nghĩa khu vực cần mô phỏng.
- Burn in a stream network: sử dụng mạng lưới dòng chảy có sẵn để mô tả lưu vực.

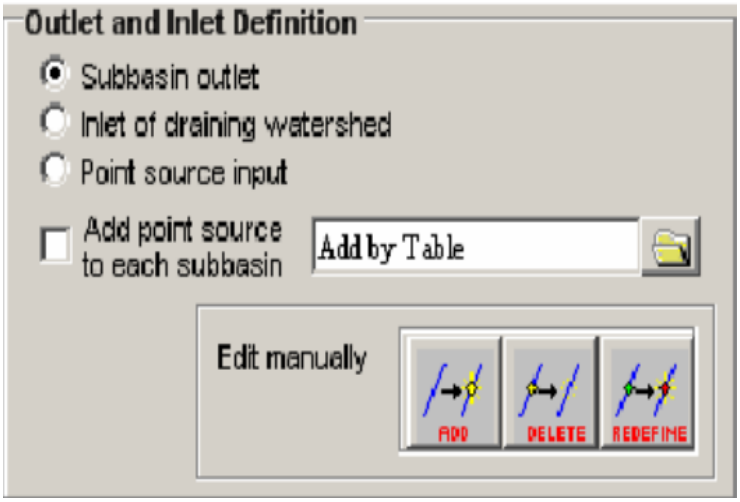


Mục Stream Definition:

- DEM – based: sử dụng DEM để mô phỏng lưu vực.
- Flow direction and accumulation: Tính toán hướng dòng chảy và dòng chảy tích lũy.
- Area: Thiết lập ngưỡng diện tích tiểu lưu vực.
- Stream network: Tạm mạng lưới dòng chảy, cửa xả.
- Pre-defined streams and watershed: sử dụng mạng lưới tiểu lưu vực, dòng chảy đã có sẵn.

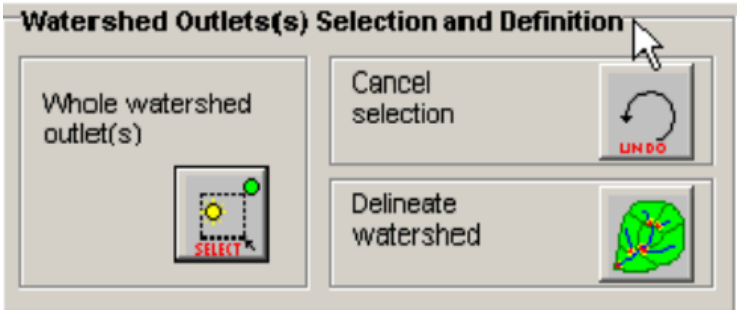


Mục Outlet and Inlet Definition:



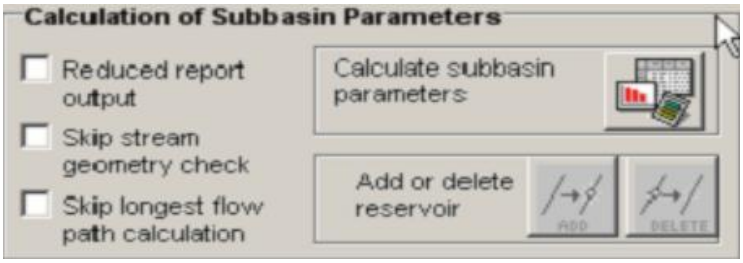
- Chọn subbasin outlet.
- Add by Table: sử dụng bảng để thêm cửa xả.
- Add, Delete, Redefine: Thêm, xóa, di chuyển cửa xả thủ công.

Mục Watershed Outlets (s) Selection and Definition:



- Whole watershed outlet (s): lựa chọn một hay nhiều cửa xả lưu vực.
- Cancel selection: Hủy việc lựa chọn cửa xả.
- Delineate watershed: Tiến hành phân định lưu vực.

Mục calculation of Subbasin Parameters:



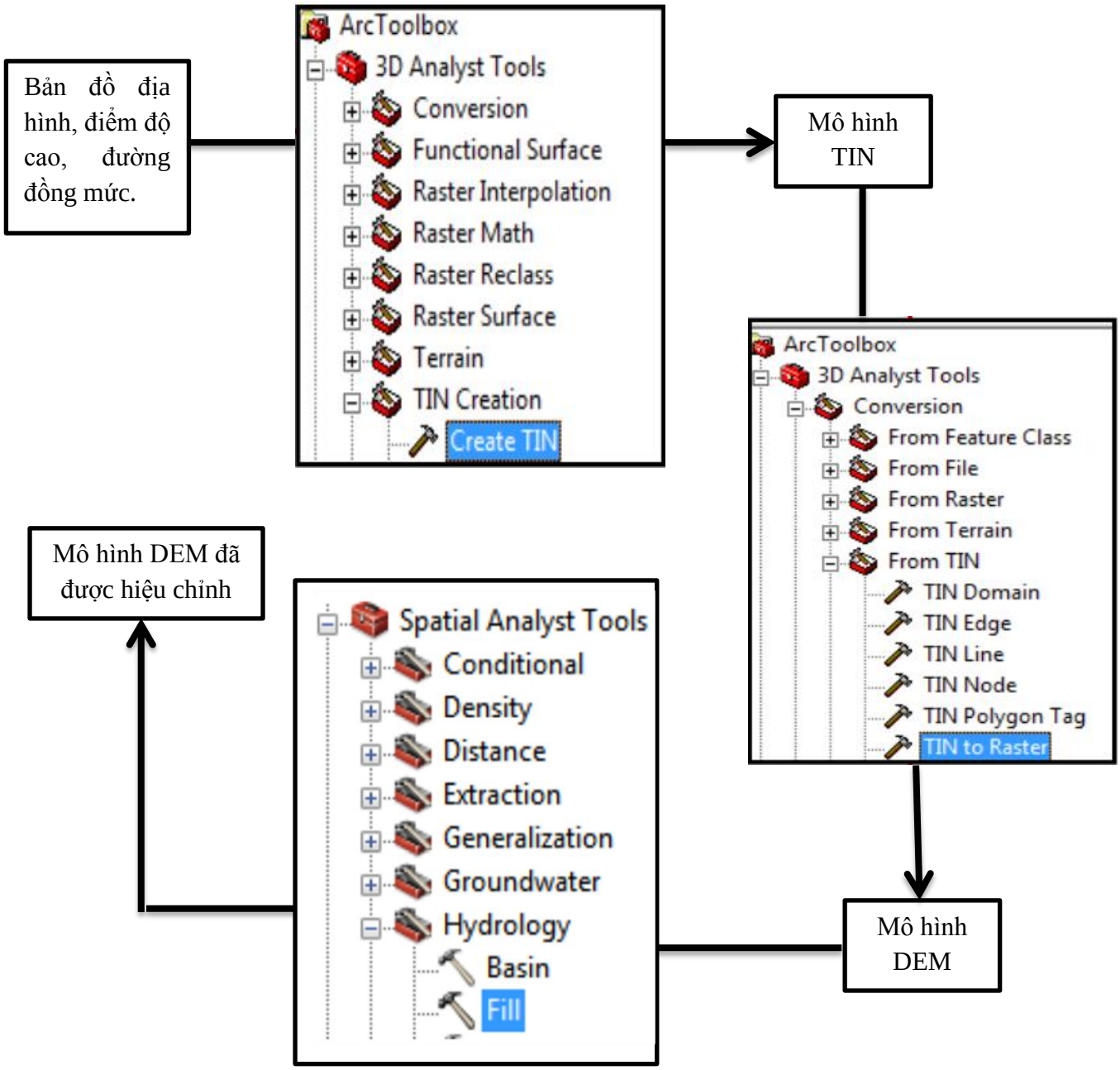
- Reduced report output: bỏ qua thống kê địa hình tiểu lưu vực.
- Skip stream geometry check: Bỏ qua việc kiểm tra hình học của dòng chảy.
- Skip longest flow path calculation: bỏ qua tính toán đường dòng chảy dài nhất.
- Calculate subbasin parameters: tính toán các thông số của tiểu lưu vực.

Hoàn tất quá trình phân định lưu vực, nhấn nút Exit để hoàn tất toàn bộ quá trình phân định lưu vực và xem kết quả.

3.2.2.2. Bộ công cụ tìm dòng chảy tích lũy trong ArcGIS

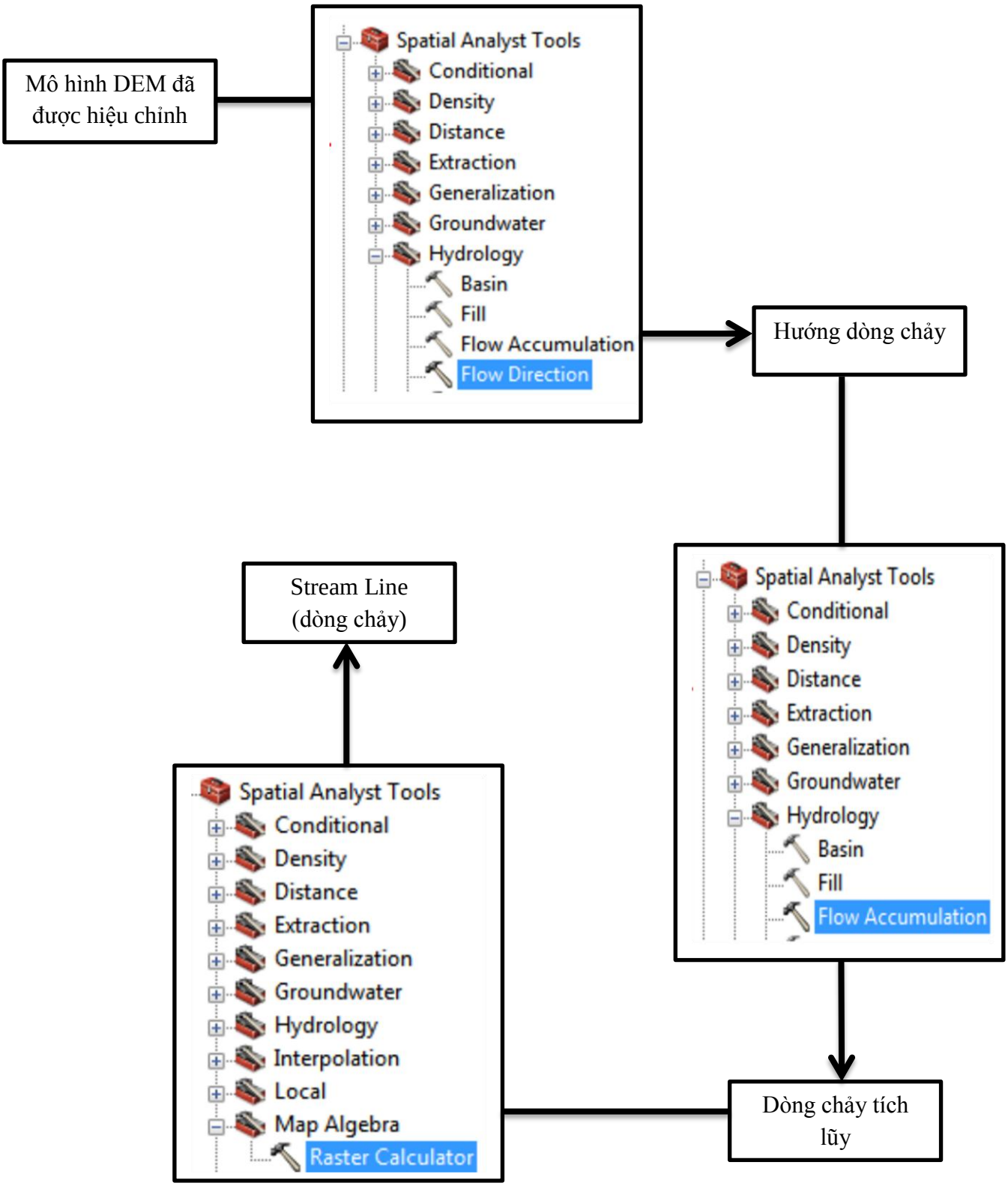
Trong ArcGIS để tìm dòng chảy tích lũy, cần thực hiện tuần tự các bước sau:

- **Bước 1:** Tạo DEM từ bản đồ địa hình và hiệu chỉnh



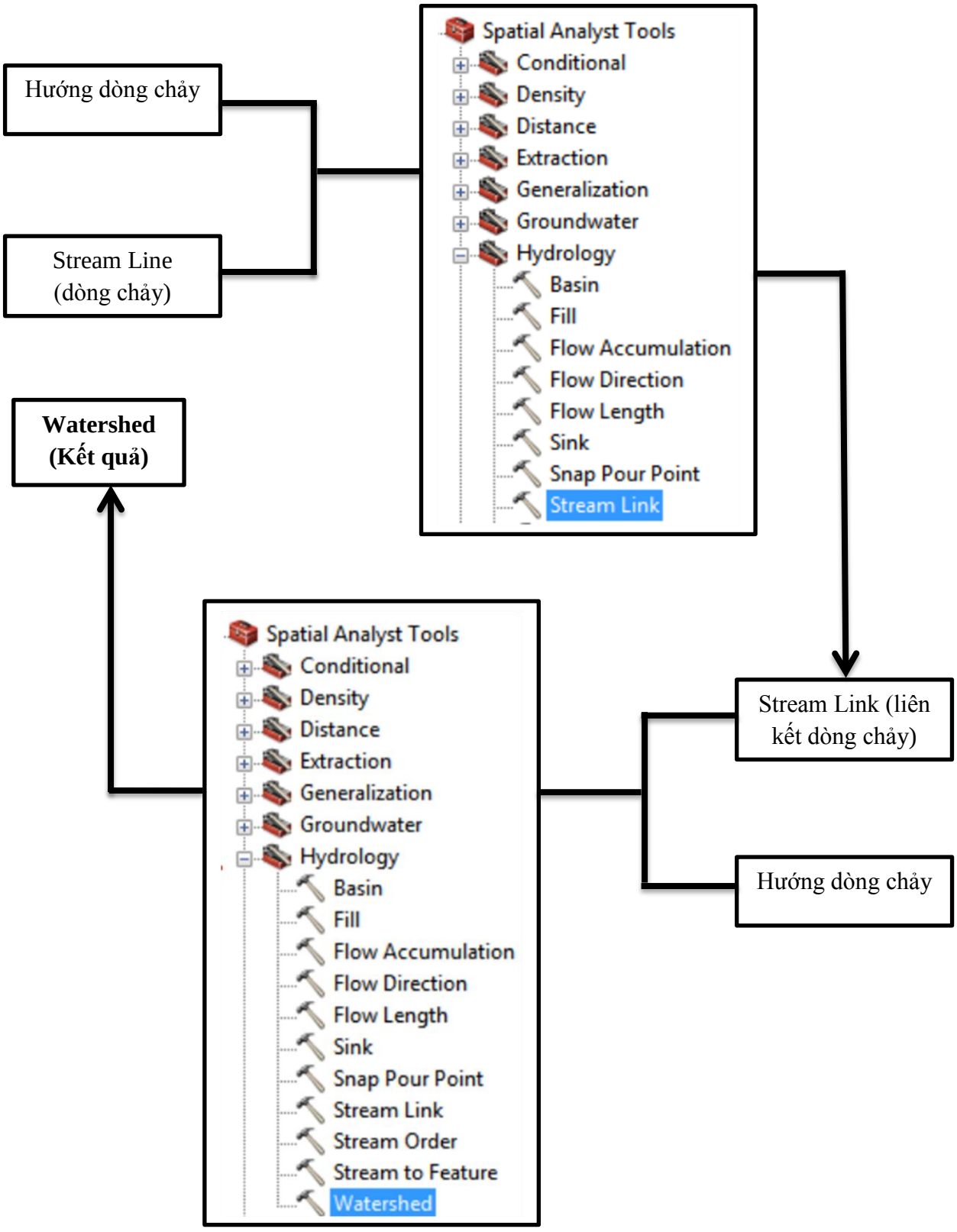
Hình 3.2.2.2.1: Sơ đồ tạo DEM từ bản đồ địa hình trong ArcGis

- **Bước 2:** Tính toán hướng dòng chảy, dòng chảy tích lũy và tìm dòng



Hình 3.2.2.2.2: Sơ đồ tính hướng, tích lũy và tìm dòng trong ArcGis

- **Bước 3:** Stream link/ Snap Pour Point (liên kết dòng) và watershed:



Hình 3.2.2.2.3 Sơ đồ tìm liên kết dòng và cửa xả trong ArcGis

- **Nhận xét:** Để thực hiện việc tính toán tích lũy dòng chảy bằng các công cụ tính toán trên là khá phức tạp. Bên cạnh đó các công cụ trên chỉ tính toán được các DEM nhỏ và tiểu lưu vực, việc áp dụng cho các DEM, lưu vực lớn cho ta kết quả không hiệu quả cao hay mất nhiều thời gian và không điều chỉnh được kết quả như ý muốn.

3.3. Cài đặt thuật toán D8 (tuần tự)

Khi đã có dữ kiệu truyền vào là file text độ cao ta bắt đầu thực hiện tuần tự các bước cài đặt thuật toán như sau:

3.3.1. Đọc dữ liệu (đọc file text độ cao)

Ở bước này, ta thực hiện đọc file text và lấy ra các dữ liệu cần sử dụng như số dòng, số cột, độ cao của từng ô trong mảng,..

```
// sử dụng Open File Dialog để lấy đường dẫn của file text
   if (openFileDialog1.ShowDialog() == DialogResult.OK)
   {
//đường dẫn file text được lưu vào label1
  label1.Text = openFileDialog1.FileName;
//sử dụng textbox1 để hiển thị file text vừa đọc
  textBox1.Text = File.ReadAllText(label1.Text);
}
//tạo chuỗi reco lưu dữ liệu từ file text từ đường dẫn
label1
   string[] reco =
System.IO.File.ReadAllLines(label1.Text);
//vì cấu tạo của file text là dòng 1 và dòng 2 lưu số cột
và số dòng
nên ta sẽ lấy 2 dòng đó ra
   var dong1 = reco[0];
   var dong2 = reco[1];
// cắt các kí tự cuối của dòng và chuyển dạng chuỗi sang
dạng int để lấy kết quả

   var a1 = dong1.Substring(dong1.Length - 5);
   var a2 = dong2.Substring(dong2.Length - 5);
   col = Convert.ToInt16(a1.Trim());
   row = Convert.ToInt16(a2.Trim());
```

Khi đã có số dòng và số cột ta resize lại các mảng đã khai báo để tính trước đó:

```
//resize lại các mảng
if (row > 0 & col > 0)
   {
       docao = new double[row, col];
       huong = new double[row, col];
       tichluy = new string[row, col];
       ketqua = new int [row,col];
    }
```

Để cho việc tính toán được trở nên dễ dàng hơn tôi sẽ tạo một file text mới và lưu dữ liệu độ cao vào đó:

```
//khởi tạo vòng lặp chạy từ dòng thứ 6 đến dòng row(số dòng
của dữ liệu) + 6
for (int h = 6; h < row + 6; h++)
{
//sử dụng mảng "mt" để lưu dữ liệu
  mt = mt + reco[h] + Environment.NewLine;
}
//sử dụng phương thức SteamWrite để tạo và ghi kết quả vào
file text trong đường dẫn bên dưới
using (StreamWriter writer = new
StreamWriter("D:\\Matrix_dem.txt"))
{
  writer.Write(mt);
  writer.WriteLine();
}
```

3.3.2. Xác định hướng dòng chảy theo D8

Khi đã có file text lưu dữ liệu riêng, ta sẽ dùng nó để tính hướng dòng chảy theo D8. Trước hết ta cần đọc file text đã lưu ở bước trước đó:

```
//đọc file text theo từng dòng với đường dẫn đã lưu
string[] text =
System.IO.File.ReadAllLines("D:\\Matrix_dem.txt");
//lặp để cắt từng vị trí của dữ liệu trong file text
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
```

```
{
var na = text[i];
string[] pt = na.Split(' ');
// chuyển tất cả dữ liệu về dạng "int"
int yy = Convert.ToInt16(pt[j]);
//lưu dữ liệu vào mảng docao
docao[i, j] = yy;
}
}
```

Khi có mảng docao ta bắt đầu tính hướng của từng ô:

Theo như thuật toán tính hướng dòng chảy D8, ô ở vị trí (i, j) sẽ chảy về ô nào có độ cao nhỏ nhất trong 8 ô xung quanh nó tức ô có giá trị tuyệt đối độ cao chênh lớn nhất trong 7 ô còn lại. Tôi sẽ xét từng trường hợp, điều kiện ràng buộc là hướng chảy không được vượt ngoài mảng. VD: trường hợp hướng bằng 1 tức tại vị trí đó ô sẽ chảy sang bên phải nên điều kiện ràng buộc là số cột cộng 1 (j + 1) phải nhỏ hơn số cột của mảng. Bắt đầu tính giá trị của trường hợp này.

```
max = Math.Max(max, (array1[i, j] - array1[i, j + 1]));
```

Khi có giá trị tại trường hợp đó, ta so sánh với 7 trường hợp còn lại, nếu từ ô (i, j) trừ ô (i, j + 1) đạt giá trị lớn nhất thì trả ra hướng tại ô đó bằng 1.

```
if (max == (array1[i, j] - array1[i, j + 1]))
direct = 1;
```

Tương tự cho các trường hợp còn lại, các trường hợp hướng 2, 8, 32, 128 là hướng chéo nên sẽ chia cho căn 2. Sau đây là code ví dụ cụ thể của một số hướng.

```
for (int i = 0; i < row; i++) //lặp voi moi dong
{
for (int j = 0; j < col; j++)//lặp voi moi cot
{
    double giatri = docao[i, j];
// Theo D8 có 8 hướng và trường hợp ko giá trị “-9999”, tạo
biến direct để lưu hướng, nếu ô đó không nằm trong trường
hợp nào thì direct bằng 0
    int direct = 0;
//Tạo biến max để tìm giá trị lớn nhất khi chảy qua từng ô
    double max = 0;
//xet 9 truong hop:
if (docao[i, j] == -9999)
```

```
{
// với trường hợp này thì ta giữ nguyên, ko tính
direct = -9999;
}
 //trường hợp direct bằng 32(điều kiện của trường hợp này
là cột j - 1, i - 1 phải lớn hơn 0, tức là giá trị không
vượt qua ngoài mảng đó)
if ((i - 1 >= 0) && (j - 1 >= 0) && docao[i - 1, j - 1] !=
-9999)
{
//trong trường hợp này thì ô tính ra có giá trị lớn nhất là
ô nằm ở góc trên bên tay trái (tức ô i - 1, j - 1)
max = Math.Max(max, (docao[i, j] - (docao[i - 1, j - 1]))
/Math.Sqrt(2));
if (max == (docao[i, j] - docao[i - 1, j - 1]) /
Math.Sqrt(2))
// thỏa điều kiện thì in ra direct bằng 32
     direct = 32;                    }
//trường hợp direct bằng 8(điều kiện i + 1 không vượt ra
ngoài dòng và j - 1 lớn hơn 0)
if ((i + 1 < row) && (j - 1 >= 0) && docao[i + 1, j - 1] !=
-9999) //8
{
//trong trường hợp này thì ô tính ra có giá trị lớn nhất là
ô nằm ở góc dưới bên tay trái (tức ô i + 1, j - 1)
max = Math.Max(max, (docao[i, j] - docao[i + 1, j - 1]) /
Math.Sqrt(2));
if (max == (docao[i, j] - docao[i + 1, j - 1]) /
Math.Sqrt(2))
// thỏa điều kiện trên thì in ra direct bằng 8
direct = 8;
}
//Tương tự với các trường hợp direct còn lại ta tính được
direct của hướng 1, 2, 4, 16, 64, 128
//có kết quả ta in direct vào mảng huong[,]

huong[i, j] = direct;
//kiem lai sau khi doc xong, in lên hướng lên textbox để
kiểm tra:
System.Text.StringBuilder s = new
System.Text.StringBuilder();
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
```

```
    {
       s.Append(huong[i, j] + " ");
          }
       s.AppendLine();
     }
     textBox1.Text = s.ToString();
}
```

3.3.3. Tính toán tích lũy dòng chảy (D8)

Trong phần tìm dòng tích lũy này, để cho bài làm thêm đa dạng tôi sẽ không dùng thuật toán Floyd (sẽ dành cho tính toán song song) thay vào đó tôi sẽ sử dụng thuật toán đệ quy để tính. phương pháp của tôi như sau:

Tôi sẽ sử dụng 2 bước để tính tích lũy. Bước 1: Tôi đếm số ô và gán số ô đó cho từng ô để biết vị trí, cùng với đó tôi cũng dựa vào mảng hướng để tìm xem tại ô (i, j) đó có những vị trí ô xung quanh nào chảy về. Khi đã biết được có vị trí nào chảy về tôi sẽ xử lý chuỗi đó bằng cách chia vị trí cho số cột để lấy tọa độ dòng của những vị trí đó, chia vị trí lấy dư cho cột để lấy tọa độ cột của vị trí đó. Sau bước này tôi đã có tập hợp tất cả các vị trí chảy về ô (i, j) bất kỳ. Sang bước 2 tôi chỉ việc lặp để đếm số ô chảy về để lấy tích lũy. Cụ thể như sau:

Bước 1: Tính từng ô xem có những ô nào chảy về:

```
//tạo biến đếm vitri với giá trị ban đầu là 0

  int vitri = 0;
//khởi tạo mảng tichluy và gán cho mỗi ô có giá trị ban đầu
là “”
for (int i = 0; i < row; i++)
for (int j = 0; j < col; j++)
  tichluy[i, j] = "";
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
{
//giá trị vitri được bắt đầu đếm, giá trị sau mỗi ô được
tăng lên 1 đơn vị
vitri = vitri + 1;
//tiếp tục chia thành 9 trường hợp
```

```
if (huong[i, j] == -9999) tichluy[i, j] = " "; //với trường
hợp hướng bằng "-9999" thì tichluy không được tính
if (huong[i, j] == 32) tichluy[i - 1, j - 1] = tichluy[i -
1, j - 1] + " " + vitri.ToString(); // Trường hợp hướng
bằng 32 thì theo lý thuyết tại ô 32 đó chảy lên ô phía bên
trên góc trái (i – 1, j – 1) nên tại ô phía trên đó sẽ được
cộng thêm ô 32 vừa chảy về
if (huong[i, j] == 8) tichluy[i + 1, j - 1] = tichluy[i +
1, j - 1] + " " + vitri.ToString(); // Trường hợp hướng
bằng 8 thì tại ô bằng 8 đó chảy xuống ô phía dưới góc trái
(i + 1, j – 1) nên tại ô phía trên đó sẽ được cộng thêm ô 8
vừa chảy về
//Tính tương tự với các trường hợp hướng còn lại 1, 2, 4,
16, 64, 128

}
}
```

Khi đã có được giá trị ở bước trên ta cần xử lý chuỗi của từng ô xem mỗi vị trí trước đó còn có ô nào chảy về hay không, ta dùng hàm đệ quy:

```
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
{
//gọi hàm xử lý tích lũy
tichluy[i, j] = xulytichluy(i, j, row, col);
}
}
//xử lý tích lũy
public string xulytichluy(int i, int j, int row, int col)
{
int toadox, toadoy;
string kq = (" " + tichluy[i, j] + " ").Trim();
if (String.ReferenceEquals(kq, ""))
{
kq = "";
}
else
{
string[] dayvitri = kq.Split(' ');
foreach (string viri in dayvitri)
{
if (!String.ReferenceEquals(viri, ""))
```

```
{
toadox = Convert.ToInt16(viri) / col; // chia để lấy tọa độ
dòng
toadoy = Convert.ToInt16(viri) % col; // chia lấy dư để lấy
tọa độ cột
kq = xulytichluy(toadox, toadoy, row, col) + " " +
kq.Trim();
}
}
}
  return kq;
}
```

Bước 2: Cộng các vị trí lại để lấy kết quả: khi đã biết mỗi ô trong tích lũy có những vị trí nào chảy về, ta lấy chuỗi đó và tính xem mỗi ô có tất cả bao nhiêu vị trí chảy về.

```
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
{
  int dodai = 0;
//dùng hàm Trim cắt khoảng trắng đầu và cuối chuỗi
string tmp = tichluy[i, j].Trim();
if (!String.ReferenceEquals((" " + tichluy[i, j] + "
").Trim(), ""))
{
  int soluongtrung = 0;
//đếm số lượng chuỗi
string[] chuoi = (tichluy[i, j].Trim()).Split(' ');
//với mỗi chuoicon, đếm số lượng chuỗi trùng:
foreach (string chuoicon in chuoi)
{
String temp = " " + tmp.Trim() + " ";
temp = temp.Replace(" " + chuoicon + " ", " a ");
//nếu chuoicon còn nằm trong chuoi tmp thì:
if (tmp.IndexOf(" " + chuoicon + " ") >= 0)
{
string[] chuoi1 = temp.Split('a');
soluongtrung += chuoi1.Length - 2;
```

```
//nếu chuỗi con nào tính rồi thì không tính nữa để tránh
lặp
tmp = tmp.Replace(" " + chuoicon + " ", " - ");
}
}
dodai = chuoi.Length - soluongtrung;
}
ketqua[i, j] = dodai;
}
Console.WriteLine();
}
//In ra kết quả tính được
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
{
Console.Write(ketqua[i, j] + " ");
}
Console.WriteLine();
}
```

3.4. Tại sao phải cài đặt thuật toán song song

Để trả lời cho câu hỏi tại sao phải cài đặt thuật toán song song tôi đã tìm hiểu và xây đựng ứng dụng để có thể dễ dàng so sánh sự khác biệt của 2 thuật toán này, ứng dụng đó như sau:

Ứng dụng thứ nhất sử dụng thuật toán tuần tự để tính tích ma trận dựa vào 2 ma trận được tạo ngẫu nhiên, giá trị từ 0 đến 100, với số dòng và số cột được người dùng tự nhập vào. Sau đó tính toán thời gian chạy của ứng dụng, sau đây là chương trình được viết cụ thể:

```
string k = textBox1.Text; //nhập vào số dòng
string k1 = textBox2.Text; //nhập vào số cột
bool result = false;
bool result1 = false;

result = int.TryParse(k, out hang); //lấy ra số dòng
result1 = int.TryParse(k1, out cot); // lấy ra số cột
// khởi tạo các mảng 2 chiều với số dòng, số cột vừa nhập
vào
```

```
double [,] array = new double[hang, cot]; //mảng để lưu kết quả
double[,] array1 = new double[hang, cot]; //mảng 1 ngẫu nhiên lưu các giá trị ngẫu nhiên để lấy tích
double[,] array2 = new double[hang, cot]; //mảng 2 ngẫu nhiên lưu các giá trị ngẫu nhiên để lấy tích

for (int i = 0; i < hang; i++)
{
 for (int j = 0; j < cot; j++)
  {
  array1[i, j] = random.Next(100); //tạo một mảng với số ngẫu nhiên có giá trị từ 0 đến 100, tương tự với mảng array2
  array2[i, j] = random.Next(100);
  }
Stopwatch sw = Stopwatch.StartNew();// gán biến bắt đầu đếm thời gian của chuong trình
// Bắt đầu lấy tích 2 ma trận
for (int i = 0; i < hang; i++) //voi moi dong
{
for (int j = 0; j < cot; j++)//voi moi cot
{
for (int k = 0; k < cot; k++)
{
  array[i, j] = array1[i, k] * array2[k, j]; // lấy tích của array1 và array2 lưu vào mảng array
}
Console.Write(array[i, j] + " ");
}
Console.WriteLine("------");
}
sw.Stop();//dừng đồng hồ và tính thời gian thực hiện chương trình
TimeSpan ts = sw.Elapsed;
MessageBox.Show("Thời gian thực hiện là:" + ts.Minutes + "phút" + ts.Seconds + "giây");
} // kết thúc chương trình
```

Để so sánh với thuật toán này tôi cũng tạo 1 ứng dụng giống như vậy, cũng tính tích ma trận từ 2 ma trận ngẫu nhiên với số dòng và số cột được

người dùng nhập vào, nhưng ứng dụng này được viết trên .NET 4.0 và dùng thuật toán song song. Cụ thể ứng dụng như sau:

```
Console.Write("Hay nhap vao so so hang: ");
  int hang = int.Parse(Console.ReadLine());// lấy ra số
hàng
Console.Write("Hay nhap vao so so cot: ");
   int cot = int.Parse(Console.ReadLine()); //lấy ra số cột
//tương tự ta cũng tạo mảng array1 và array2 để lưu ma trận
ngẫu nhiên, array để lưu kết quả tích 2 ma trận
double[,] array = new double[hang, cot];
double[,] array1 = new double[hang, cot];
double[,] array2 = new double[hang, cot];
//lặp for lấy ngẫu nhiên số từ 1 đến 100 vào 2 mảng array1
và array2
System.Random random = new System.Random();//lấy ra hàm lấy
số ngẫu nhiên

Parallel.For(0, hang, i =>
{
for (int j = 0; j < cot; j++)
{
array1[i, j] = random.Next(100); // lấy số ngẫu nhiên vào
array1
array2[i, j] = random.Next(100); // lấy số ngẫu nhiên vào
array2
}
Console.WriteLine();
});
// sau đó ta bắt đầu lấy tích 2 ma trận
Stopwatch sw = Stopwatch.StartNew(); //bấm đồng đồ đo thời
gian thực hiện

Parallel.For(0, hang, i => //voi moi dong
{
for (int j = 0; j < cot; j++)
{
for (int k = 0; k < cot; k++)
{
 array[i, j] = array1[i, k] * array2[k, j];
}
Console.Write(array[i, j] + " "); //in kết quả ra màn hình

}
Console.WriteLine("------");
});
sw.Stop();// dừng đồng hồ
TimeSpan ts = sw.Elapsed;
```

```
Console.Write("Thoi gian thuc hien song song la:" +
ts.Seconds + "s");//in ra số giây thực hiện
Console.ReadLine();
```

So sánh điểm khác biệt của 2 ứng dụng trên:

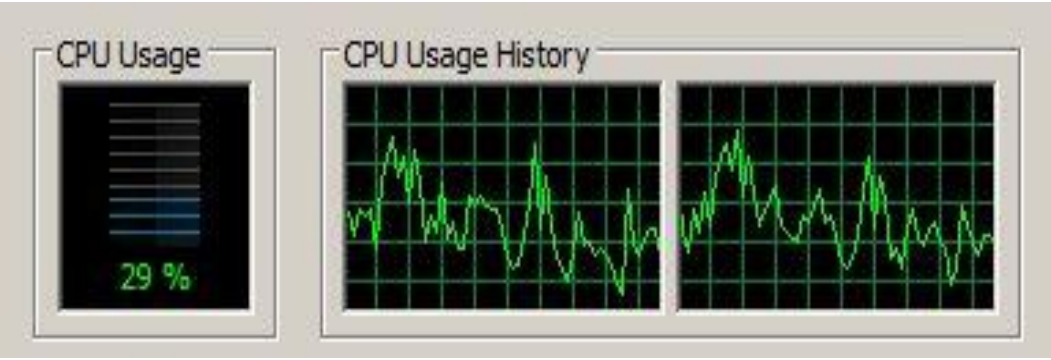
- Thời gian: ta cùng xem bảng thống kê so sánh thời gian chạy của 2 ứng dụng trong bảng dưới đây:

Số dòng, số cột	Thời gian chạy (phút/ giây)	
	Thuật toán tuần tự	Thuật toán song song
100 x 100	8 giây	2 giây
300 x 300	1 phút 22 giây	22 giây
500 x 500	3 phút 46 giây	1 phút 5 giây
1000 x 1000	15 phút 9 giây	4 phút 14 giây

Bảng 3.4: Thống kê thời gian của 2 phép toán tuần tự và song song

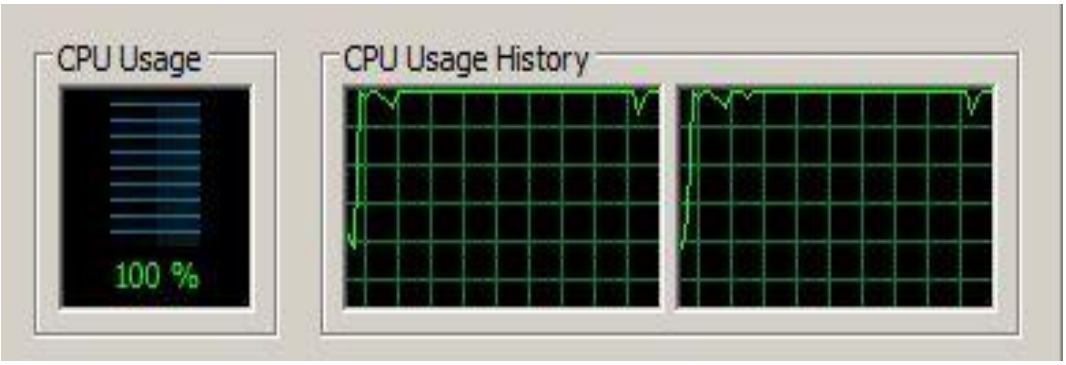
Bảng thống kê thời gian thực hiện của 2 phép toán trên cho thấy thời gian thực hiện cùng một bài toán của 2 phép toán là rất khác biệt, ưu thế nghiêng hẳn về phép toán song song, với tốc độ nhanh hơn phép toán tuần tự rất nhiều lần.

- Hiệu suất CPU



Hình 3.4.1: Hiệu suất CPU khi thực hiện phép toán tuần tự

Hiệu suất CPU khi thực hiện phép toán tuần tự ở mức trung bình, biểu đồ cho thấy tốc độ máy chạy chưa thực sự ổn định, chưa thực sự khai thác được tối đa khả năng của CPU.



Hình 3.4.2: Hiệu suất CPU khi thực hiện phép toán song song

Trong hình 3.4.2 cho ta thấy trong trường hợp này, phép toán song song đã sử dụng tất cả khả năng mà CPU có thể để thực hiện phép toán, biểu đồ tăng đột biến và giữ nguyên ở mức cao nhất. Điều này chứng tỏ vì sao phép toán song song lại nhanh hơn, tốn ít thời gian hơn so với phép toán tuần tự.

- Debug:

Để dễ dàng nhận thấy sự thay đổi trong debug, tôi đã tạo ra mảng nhỏ 3x3 để tìm xem nó có sự khác nhau như thế nào. Sau đây là debug trong thuật toán song song tôi đã nhìn thấy:

```
3055 1820 5460 ------ hàng 2
2115 1260 3780 ------ hàng 1
1081 644 1932 ------ hàng 3
```

Chúc ta đã có thể dễ dàng nhận thấy đã có sự xáo trộn ở đây, các dòng không còn được in theo thứ tự nữa, nó đã được xáo trộn trùy theo kết quả đã tính được. Vì được thực hiện cùng lúc trên nhiều nhân nên dòng nào thực hiện xong sẽ được in ra ngay.

- **Kết luận:** Với những so sánh trên , có thể thấy phép toán song song có những ưu điểm vượt trội so với thuật toán tuần tự nhất là đối với những dữ liệu lớn, những gói dữ liệu mà ta có thể sẽ phải ngồi chờ hàng giờ đồng hồ để đợi kết quả thì đây, phương pháp xử lý song song sẽ là một giải pháp mới và cũng là thách thức không nhỏ cho những là lập trình viên (đối với những bài toán phức tạp).

3.5. Cài đặt thuật toán song song D8

Trong phương pháp xử lý song song, điểm khác biệt lớn nhất với phương pháp tuần tự là phương pháp song song sẽ chia nhỏ dữ liệu ra để tính. Xét dữ liệu như trong hình 3.5.1, nếu phương pháp tuần tự sẽ sử dụng cả mảng lớn để tính lần lượt từng ô trong mảng thì phương pháp song song sẽ chia thành 2 hay nhiều mảng con để tính toán, điều này giúp cho việc nhiều CPU có thể thực hiện đồng thời trong cùng một thời điểm, giúp cho việc tính toán dữ liệu lớn được thực hiện nhanh hơn. Với lập trình .NET Microsoft cung cấp cho ta một công cụ lập trình không quá phức tạp trong việc tính toán song song. Từ thuật toán tuần tự tìm tích lũy dòng chảy tôi dễ dàng phát triển lên thuật toán tìm tích lũy dòng chảy song song cho bài của mình. Việc cài đặt thuật toán cũng bao gồm các bước :

78	72	69	71	58	49
74	67	56	49	46	50
69	53	44	37	38	48
64	58	55	22	31	24
68	61	47	21	16	19
74	53	34	12	11	12

Hình 3.5.1: Chia dữ liệu thành 4 mảng con

3.5.1. Đọc dữ liệu

Trước tiên, việc đọc dữ liệu, tôi cũng đọc file text và lấy ra số dòng số cột. Sau đó tôi sẽ cắt dữ liệu độ cao ra lưu vào file text riêng ra để tính toán. Tôi thực hiện như sau:

```
// sử dụng Open File Dialog để lấy đường dẫn của file text

if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
//đường dẫn file text được lưu vào label1
label1.Text = openFileDialog1.FileName;
//sử dụng textbox1 để hiển thị file text vừa đọc
textBox1.Text = File.ReadAllText(label1.Text);
}
//tạo chuỗi reco lưu dữ liệu từ file text từ đường dẫn
label1
string[] reco = System.IO.File.ReadAllLines(label1.Text);
//vì cấu tạo của file text là dòng 1 và dòng 2 lưu số cột
và số dòng
nên ta sẽ lấy 2 dòng đó ra
```

```
var dong1 = reco[0];
var dong2 = reco[1];
// cắt các kí tự cuối của dòng và chuyển dạng chuỗi sang
dạng int để lấy kết quả
var a1 = dong1.Substring(dong1.Length - 5);
var a2 = dong2.Substring(dong2.Length - 5);
col = Convert.ToInt16(a1.Trim());// lấy ra số cột
row = Convert.ToInt16(a2.Trim());// lấy ra số dòng
```

Khi đã có được số dòng và số cột ta resize lại các mảng cần tính toán đã khai báo trước đó:

```
if (row > 0 & col > 0)
{
docao = new double[row, col];
huong = new double[row, col];
tichluy = new string[row, col];
}
```

Để dễ dàng cho việc tính toán, tôi sẽ lưu những giá trị độ cao vào file text riêng.

```
for (int h = 6; h < row + 6; h++) //cho biến chạy từ dòng 0
đến số dòng để lấy ra dữ liệu cần tính toán(chạy từ 6 vì 6
hàng đầu tiên lưu số dòng, số cột, ..., dữ liệu được lưu
bắt đầu từ dòng thứ 6)

{
mt = mt + reco[h] + Environment.NewLine; //tạo mảng mt lưu
dữ liệu độ cao
}
//Sử dụng phương thức streamwrite để lưu mảng mt vào đĩa
cứng
using (StreamWriter writer = new
StreamWriter("D:\\Matrix_dem.txt")) //đường dẫn file text
là Đĩa cứng D, file text có tên “Matrix_dem.txt”
{
writer.Write(mt1); //ghi mảng mt vào file text
writer.WriteLine();
}
```

3.5.2. Xác định song song hướng dòng chảy theo D8

Giống như tuần tự, ta cũng đọc file text đã ghi trước đó và bắt đầu tính hướng dòng chảy. Cụ thể như sau:

```
//đọc file text lưu mảng 1 theo từng dòng với đường dẫn đã
lưu
string[] text =
System.IO.File.ReadAllLines("D:\\Matrix_dem.txt");
//lặp để cắt từng vị trí của dữ liệu trong file text
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
{
var na = text[i];
string[] pt = na.Split(' ');
  int yy = Convert.ToInt16(pt[j]);
  docao[i, j] = yy; //lưu dữ liệu vào docao[,] mà ta đã
khai báo trước đó
}
});
//bắt đầu tính hướng cho mảng thứ nhất. Cách tính giống như
thuật toán tuần tự nên tôi sẽ trình bày chứ không giải
thích thêm nữa, thuật toán của tôi như sau:
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
{
  double max = 0;
  double giatri = docao[i, j];
  int direct = 0;
//xet 8 truong hop:
if (docao[i, j] == -9999)
{
direct = -9999;
}
if ((i - 1 >= 0) && (j - 1 >= 0) && docao[i - 1, j - 1] !=
-9999)//32
{
max = Math.Max(max, (docao[i, j] - (docao[i - 1, j - 1])) /
Math.Sqrt(2));
if (max == (docao[i, j] - docao[i - 1, j - 1]) /
Math.Sqrt(2))
```

```
  direct = 32;
}
if ((i - 1 >= 0) && docao[i - 1, j] != -9999) //64
{
max = Math.Max(max, (docao[i, j] - docao[i - 1, j]));
if (max == (docao[i, j] - docao[i - 1, j]))
direct = 64;
.....................................................

huong[i, j] = direct;//gán kết quả vào huong[,]
}
});
//kiem lai sau khi doc xong:
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
{
Console.Write(huong[i, j] + "  "); //In kết quả tính được
ra màn hình
}
Console.WriteLine();
};
```

Ta cũng có thể nhận thấy điểm khác biệt trong phần này so với phương pháp tuần tự là cấu trúc của vòng lặp for đã có thay đổi, vòng lặp for đã được thêm phần parallel phía trước và các kiểu ký tự cũng đã thay đổi.

3.5.3. Tính toán song song tích lũy dòng chảy theo D8

Trong phần này, tôi sẽ sử dụng thuật toán Floyd để tính tích lũy song song đó tôi cũng sẽ viết code song song cho thuật toán này. Dựa vào bài toán tôi đã trình bày ở phần 2.4 thì tôi sẽ dùng thuật toán Floy để tìm đường đi ngắn nhất, một chiều có thể có của tất cả các điểm trên đồ thị. Sau đó tôi cộng dồn các điểm lại về cho ra kết quả tích lũy. Chi tiết bài làm của tôi như sau:

Theo như thuật toán Floyd tôi cần tạo ma trận D_0 để đưa vào tính toán. Trước hết tôi tạo ra một mảng có tên là matran1, có kích thước là [dòng nhân cột, dòng nhân cột], giá trị ban đầu của matran1 sẽ được gán như sau: các ô nằm trên đường chéo có giá trị bằng 0 và các ô còn lại sẽ có giá trị là $+\infty$ (nghĩa là các ô không đi trực tiếp qua

đây). điều kiện đường đi của tôi như sau: tôi tạo một biến k = dòng*số cột + cột khi đó các đường đi có thể có là các hướng kết quả của phần tính D8 bằng 1, 2, 4, 8, 16, 32, 64, 128 và ta lấy tọa độ cho các giá trị này

Với hướng [i, j] = 1 thì l = k + 1

hướng [i, j] = 2 thì l = k + số cột + 1

hướng [i, j] = 4 thì l = k + số cột

hướng [i, j] = 8 thì l = k + số cột – 1

hướng [i, j] = 16 thì l = k – 1

hướng [i, j] = 32 thì l = k – số cột – 1

hướng [i, j] = 64 thì l = k – số cột

hướng [i, j] = 128 thì l = k – số cột + 1

Lúc này ta gán nếu matran1là [k, l] thì matran1 = - 1 nghĩa là không chảy theo hướng này (một chiều). Ngược lại nếu matran1là [l, k] thì matran1[l, k] = 1 nghĩa là hướng chảy trực tiếp. Code của phần này như sau:

```
Parallel.For(0, size, i =>
{
for (int j = 0; j < size; j++)
{
  if (i == j)
  matran1[i, j] = 0;
else
matran1[i, j] = int.MaxValue;// ý nghĩa là không chảy trực
tiếp
}
});
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
{
k = i * col + j;
if (huong[i, j] == 1) l = k + 1;
```

```
        if (huong[i, j] == 2) l = k + col + 1;
        if (huong[i, j] == 4) l = k + col;
        if (huong[i, j] == 8) l = k + col - 1;
        if (huong[i, j] == 16) l = k - 1;
        if (huong[i, j] == 32) l = k - col - 1;
        if (huong[i, j] == 64) l = k - col;
        if (huong[i, j] == 128) l = k - col + 1;
        matran1[l, k] = 1; //1 mang ý nghĩa là chảy trực tiếp
        matran1[k, l] = -1; //-1 cũng mang ý nghĩa là không thể
        chảy (từ thấp lên cao)
        }
        });
        //gan them 1 matran để tính nhân ma trận và trừ 2 trường
        hợp độ cao từ thấp cũng ko thể chảy lên cao, độ cao bằng -
        9999 không cũng không chảy
        int p, q, r, s;
        Parallel.For(0, size, i =>
        {
        for (int j = 0; j < size; j++)
        {
        if (i != j)
        {
        p = i / col;
        q = i % col;
        r = j / col;
        s = j % col;
        if ((docao[p, q] > docao[r, s]) || (docao[p, q] == -9999))
        {
          matran1[i, j] = -1; // nếu matran1 cũng nằm trong 2
    trường hợp này thì matran1 cũng bằng - 1
        }
        // sau khi đã có D₀ là matran1 ta gán thêm 2 ma trận bằng
        với matran1
        }
          matran2[i, j] = matran1[i, j];
          matranmoi[i, j] = matran1[i, j];
        }
        });
```

Sau khi đã có D_0 ta sẽ thực hiện nhân ma trận theo phương pháp Floyd đến khi ma trận không đổi và lấy kết quả tích lũy:

```
int giatri = int.MaxValue; //tạo biến giatri với giá trị
ban đầu là cộng vô cùng
bool cothaydoi = false;
do
{
cothaydoi = false;
//nhan ma tran:
Parallel.For(0, size, i =>
{
for (int j = 0; j < size; j++)
{
if (matranmoi[i, j] > 0) //chỉ xét những trường hợp chảy
được
for (int k = 0; k < size; k++)
{
//còn lại trường hợp chảy được là trường hợp lớn hơn 0
if (matran1[i, k] >= 0 && matran2[k, j] >= 0)
if (matran1[i, k] != int.MaxValue && matran2[k, j] !=
int.MaxValue)
{
giatri = Math.Min(matranmoi[i, j], matran1[i, k] +
matran2[k, j]); //lấy min của phép toán nhân ma trận Floyd
if (matranmoi[i, j] > giatri) //có thay đổi
{
matranmoi[i, j] = giatri; //gán matran mới bằng giá trị vừa
tính được
cothaydoi = true;
}
}
}
}
});
//copy lai ma trận cũ bằng ma trận mới để tiếp tục tính
Parallel.For(0, size, i =>
{
for (int j = 0; j < size; j++)
{
//như vậy lần tính sau sẽ là A mũ 4, mũ 8,...
matran1[i, j] = matranmoi[i, j];
matran2[i, j] = matranmoi[i, j];
}
});
} while (cothaydoi == true);
```

```
//đếm số tích lũy
Parallel.For(0, size, i =>
{
int tl = 0;
for (int j = 0; j < size; j++)
{
if ((i != j) && (matran1[i, j] < int.MaxValue) &&
(matran1[i, j] >= 0))
  tl += 1; //cộng dồn để đếm số tích lũy
tichluy[i / col, i % col] = (tl).ToString();//chia để lấy
tọa độ trả về kết quả
}
});
Console.WriteLine();
Console.WriteLine("===========");
Console.WriteLine();
```

Sau khi đã có kết quả tích lũy, ta lấy ra ranh giới của để đưa vào mảng.

```
//tạo mảng ranh giới và gán cho mảng với giá trị là “0”
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
  ranhgioi[i, j] = "0";
});
//lấy ranh giới
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
{
if (huong[i, j] == -9999) ranhgioi[i, j] = "-9999";
}
});
//in ranh giới vào mảng kết quả
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
{
 ketqua[i, j] = ranhgioi[i, j];
}
});
//in tích lũy vào mảng kết quả ta được mảng kết quả cuối
cùng
```

```
Parallel.For(0, row, i =>
{
for (int j = 0; j < col; j++)
{
if (tichluy[i, j] != "0") ketqua[i, j] = tichluy[i, j];
}
});
//In kết quả hiển thị lên textbox
System.Text.StringBuilder s1 = new
System.Text.StringBuilder();
for (int i = 0; i < row; i++)
{
for (int j = 0; j < col; j++)
{
s1.Append(ketqua[i, j] + " ");
}
s1.AppendLine();
}
textBox1.Text = chuoi.ToString() + s1.ToString();
```

Đến đây thì việc cài đặt thuật toán song song coi như đã xong. Để hiển thị kết quả dưới dạng raster và hiển thị raster lên form tôi làm như sau:

Trước hết để xem kết quả dưới dạng raster tôi tạo form mới để chuyển dữ liệu file text (.txt) sang file raster (.img)

Tôi tạo form như hình sau:

Hình 3.5.1: Form chuyển dữ liệu sang dạng raster

- Nút ... trong khung bước 1 tôi dùng Opendialog để mở và chọn file text cần chuyển

```
openFileDialog1.Title = "Chọn tập tin: ";
openFileDialog1.Filter = "Dạng tập tin (*.txt)|*.txt";
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
label1.Text = openFileDialog1.FileName;
}
```

- Nút ... trong khung bước 2 tôi sử dụng Savedialog để chọn nơi lưu dữ liệu raster sau khi đã chuyển:

```
saveFileDialog1.Title = "Nơi lưu tập tin: ";
saveFileDialog1.Filter = "Dạng tập tin (*.img)|*.img";
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
{
label2.Text = saveFileDialog1.FileName;
}
```

- Nút convert tôi sử dụng gói chuyển dữ liệu của ArcEngine gói phát triển của ArcGis để chuyển. Tôi để làm như sau:

```
Geoprocessor tGp = new Geoprocessor();//sử dụng gói dữ liệu
ArcEngine
tGp.OverwriteOutput = true; //lấy dữ liệu ra
ESRI.ArcGIS.ConversionTools.ASCIIToRaster tASC = new
ESRI.ArcGIS.ConversionTools.ASCIIToRaster();// lấy công cụ
AsciiToRaster sử dụng
tASC.data_type = "FLOAT"; //định dạng dữ liệu dạng FLOAT
tASC.in_ascii_file = label1.Text; // lấy đường dẫn dữ liệu
vào
tASC.out_raster = label2.Text;// lấy dữ liệu ra
IGeoProcessorResult tGeoResult =
(IGeoProcessorResult)tGp.Execute(tASC, null); //thực hiện
chương trình

MessageBox.Show("ok"); //báo thực hiện xong
```

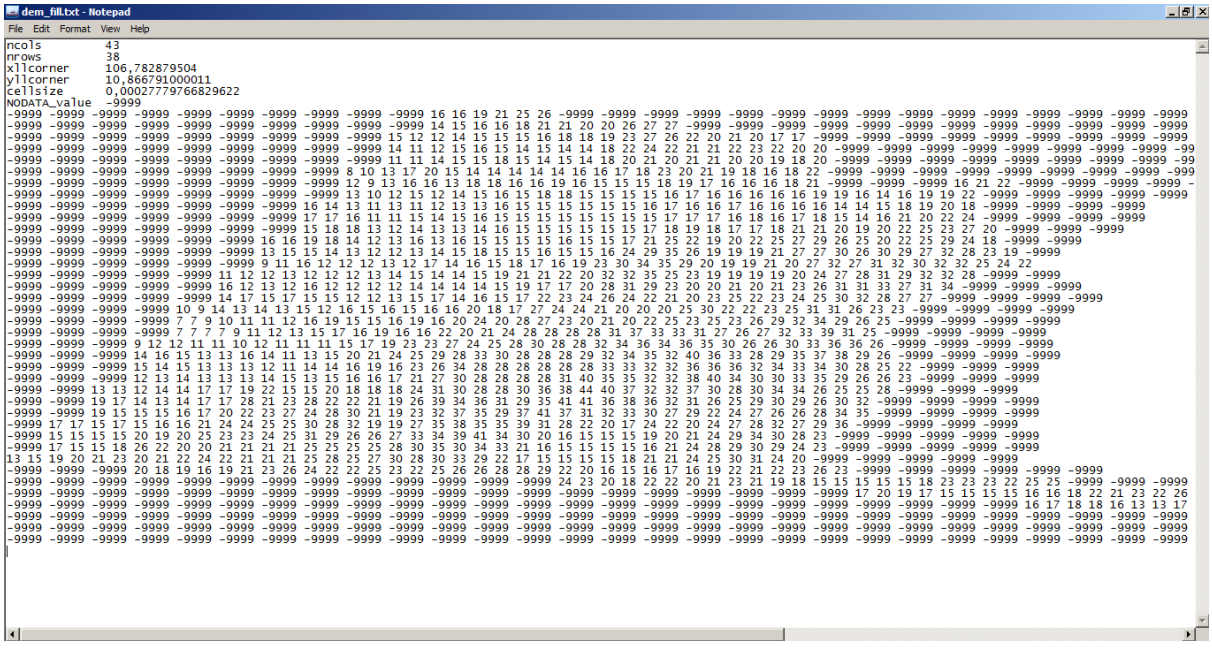
Để hiển thị raster lên form tôi tạo MapControl Application để hiển thị: Trên C# tôi vào File → New → Projet tôi chọn MapControl Application trong mục visual C#/ArcGis/Extending ArcObjects.

PHẦN 4. CÁC KẾT QUẢ NGHIÊN CỨU

4.1. Giới thiệu dữ liệu thử nghiệm

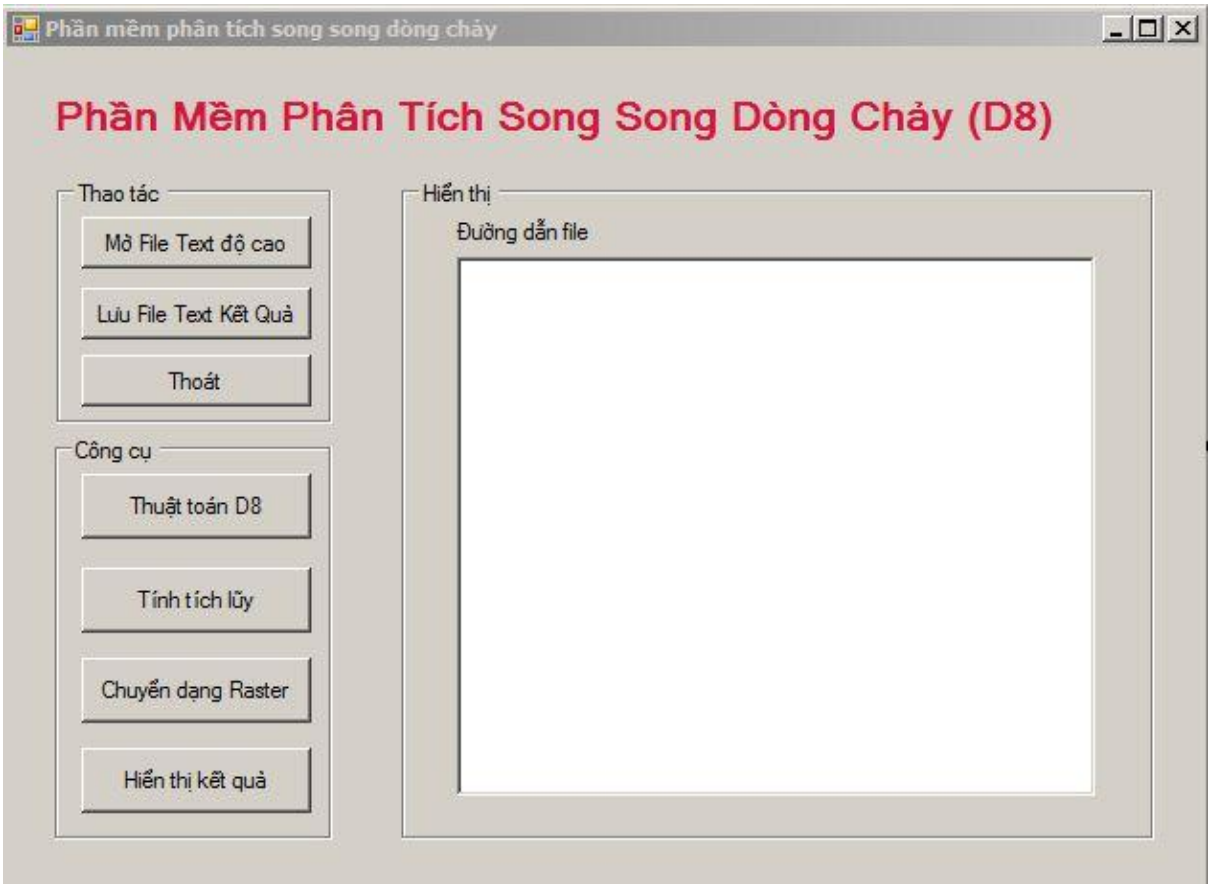
Phần mềm của tôi có thể chạy trên tất cả các dữ liệu DEM, đặc biệt vì phần mềm được thiết kế chạy đa CPU nên có thể chạy cả các DEM vừa và lớn. Trong bài của mình tôi xin chạy thử nghiệm với dữ liệu DEM của trường đại học Nông Lâm Tp.HCM.

Bộ dữ liệu độ cao thử nghiệm của tôi bao gồm toàn bộ 118 héc - ta diện tích trường đại học Nông Lâm Tp.HCM, với 1634 điểm, mỗi điểm cách nhau là 1 mét, điểm cực tây nam nằm ở tọa độ (lattitude: 106,782879504, longitude: 10,866791000011). Toàn bộ các điểm trên đã được sử dụng công cụ Fill trong Arctoolbox của ArcGis để xóa các điểm lồi và lõm không mong muốn trên bộ dữ liệu. Cụ thể file text đó có hình dạng như sau:



Hình 4.1: Bộ dữ liệu thử nghiệm trên phần mềm phân tích dòng chảy

4.2. Nhóm công cụ xây dựng trong chương trình



Hình 4.2. :Form chính của phần mềm phân tích song song dòng chảy theo D8

Phần mềm gồm có 3 nhóm mục, thứ nhất là nhóm thao tác, thứ 2 là nhóm công cụ trên phần mềm, thứ 3 là nhóm hiển thị kết quả thực hiện dạng text.

Nhóm mục thao tác:

- Mở file text độ cao: Đây là nút dùng để mở file text ArcGis mà ta muốn xử lý.
- Lưu file text kết quả: Đây là nút dùng để Lưu lại các kết quả mà ta tính được qua các thuật toán như Thuật toán D8 và tính tích lũy trở lại vào file text ArcGis.
- Thoát: Nút dừng phần mềm và thoát khi ta không muốn thực hiện nữa.

Nhóm mục công cụ:

- Thuật toán D8: Đây là nút dùng thuật toán tính hướng dòng chảy D8 để tính các hướng có thể có của một ô từ file text đã nhập vào.

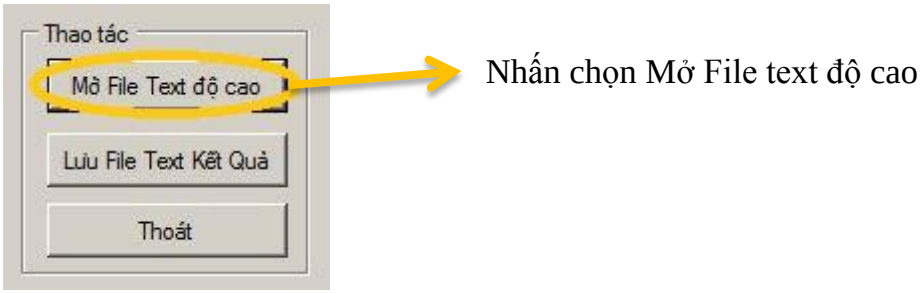
- Tính tích lũy: Từ hướng dòng chảy D8 đã tính được ta dùng nút này để tính tích lũy dòng chảy.
- Chuyển dạng raster: Sau khi đã tính được hướng D8 hay tích lũy dòng chảy, ta lưu kết quả lại thành file text và từ file text này ta sẽ dùng nút này để chuyển sang form khác bắt đầu chuyển dạng dữ liệu từ file text thành dạng raster, kết quả sẽ được hiển thị rõ hơn ở dạng này.
- Hiển thị kết quả: Nút này dùng để chuyển sang form mới. Trong form này sử dụng ArcEngine để hiển thị dữ liệu bản đồ mà ta muốn xem. Ta sẽ hiển thị file raster trên form này.

Nhóm mục hiển thị dạng tex: Là khung hiển thị phía bên tay phải, khung sẽ hiển thị kết quả mỗi khi ta load file text lên, kết quả của các phép tính hướng D8, tính tích lũy. Ta có thể dễ dàng theo dõi kết quả trong khung này.

4.3. Các kết quả thực hiện được trong 2 ứng dụng phân tích hướng dòng chảy

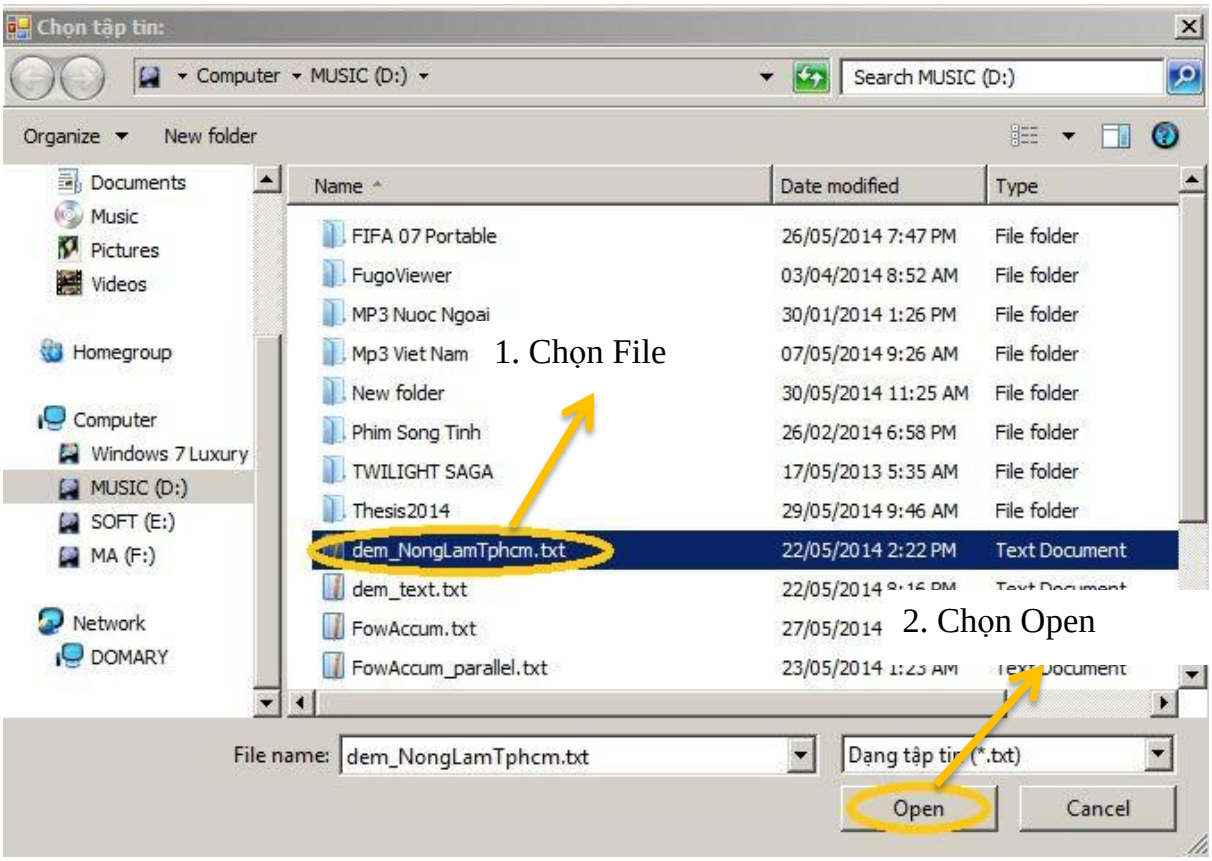
Qua các bước thực hiện với bộ dữ liệu DEM trường đại học Nông Lâm Tp.HCM phần mềm phân tích dòng chảy, cho ta những kết quả như sau:

Đầu tiên để mở file text ArcGis, trong khung thao tác ta nhấn nút mở file text độ cao như trong hình 4.3.1



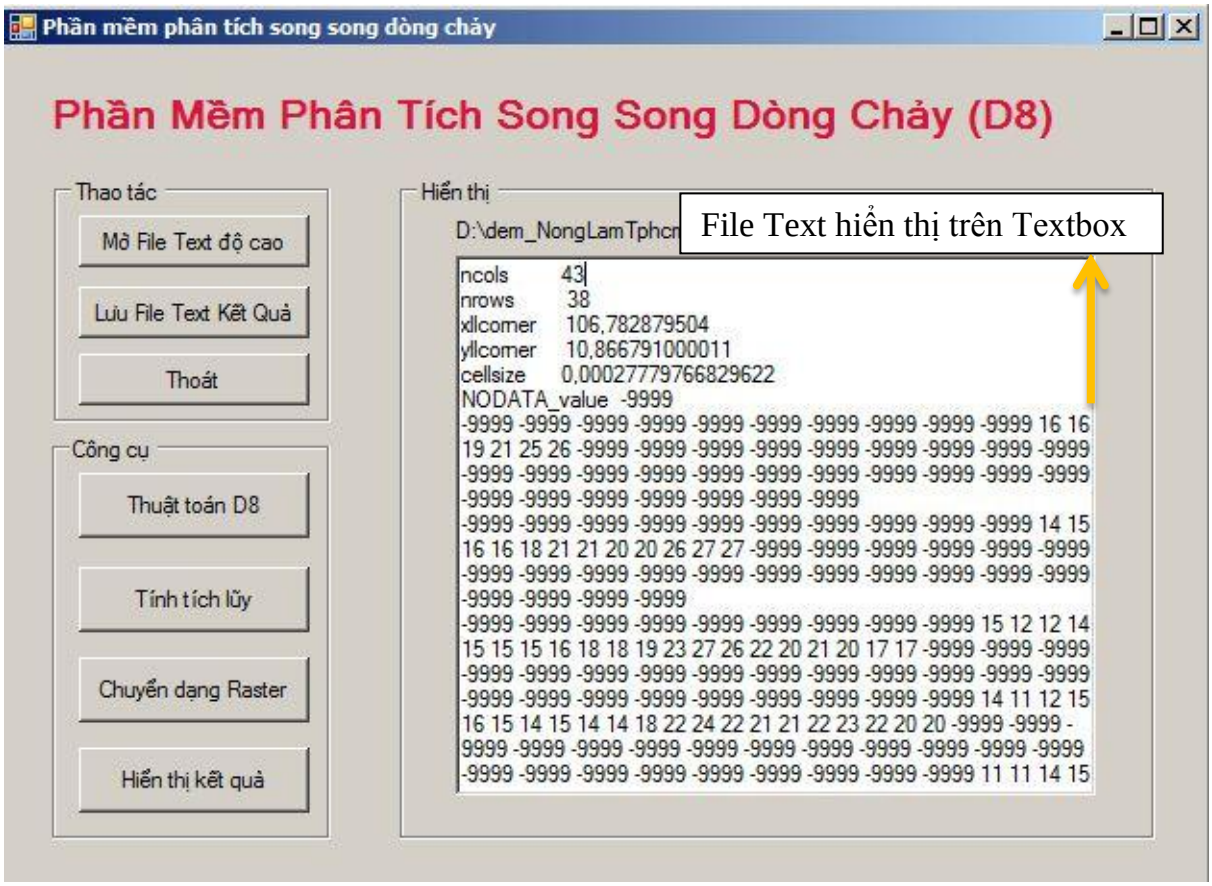
Hình 4.3.1 Mở file text độ cao

Sau đó sẽ xuất hiện bảng như trong hình 4.3.2 ta tìm đến đường dẫn nơi lưu file text ArcGis và chọn file text xong ta nhấn nút Open

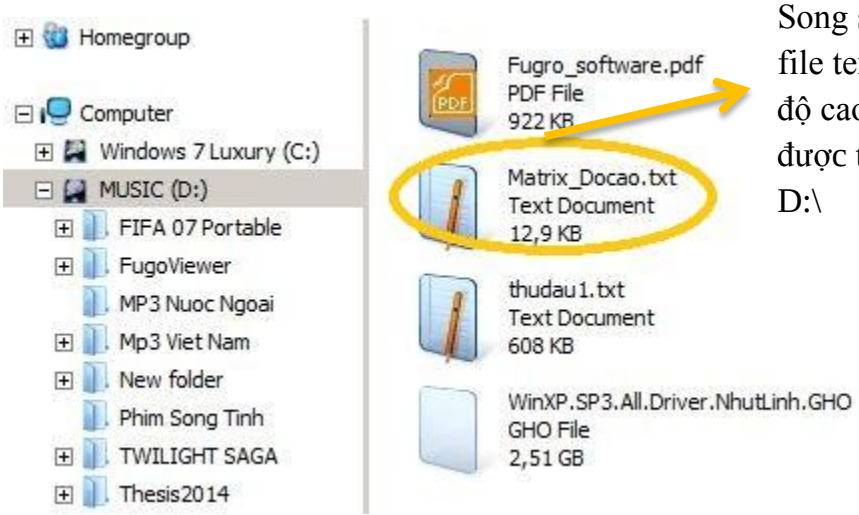


Hình 4.3.2: Chọn file text

Sau bước này, phần mềm sẽ hiển thị file text lên textbox và cắt riêng dữ liệu độ cao lưu vào một file text riêng có tên là Matrix_Docao.txt lưu trong đĩa D:\\.



Hình 4.3.3: File text hiển thị trên textbox



Song song đó thì file text lưu riêng độ cao cũng sẽ được tạo trong đĩa D:\

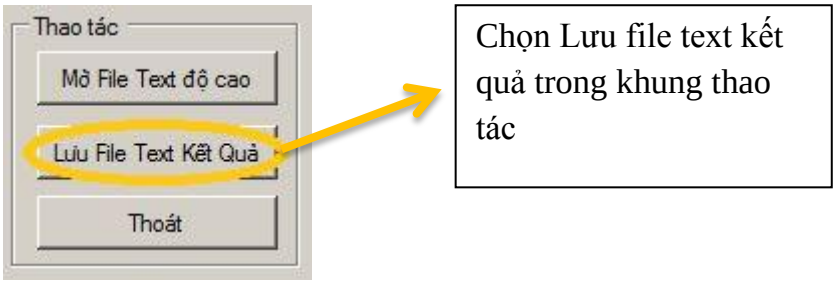
Hình 4.3.4: file text mới được tạo trong đĩa D:\

Sau khi đã nạp dữ liệu độ cao vào phần mềm, ta bắt đầu đi tìm hướng dòng chảy bằng cách nhấn nút thuật toán D8 trong khung công cụ.

Hình 4.3.5: Chọn nút bắt đầu tính thuật toán D8

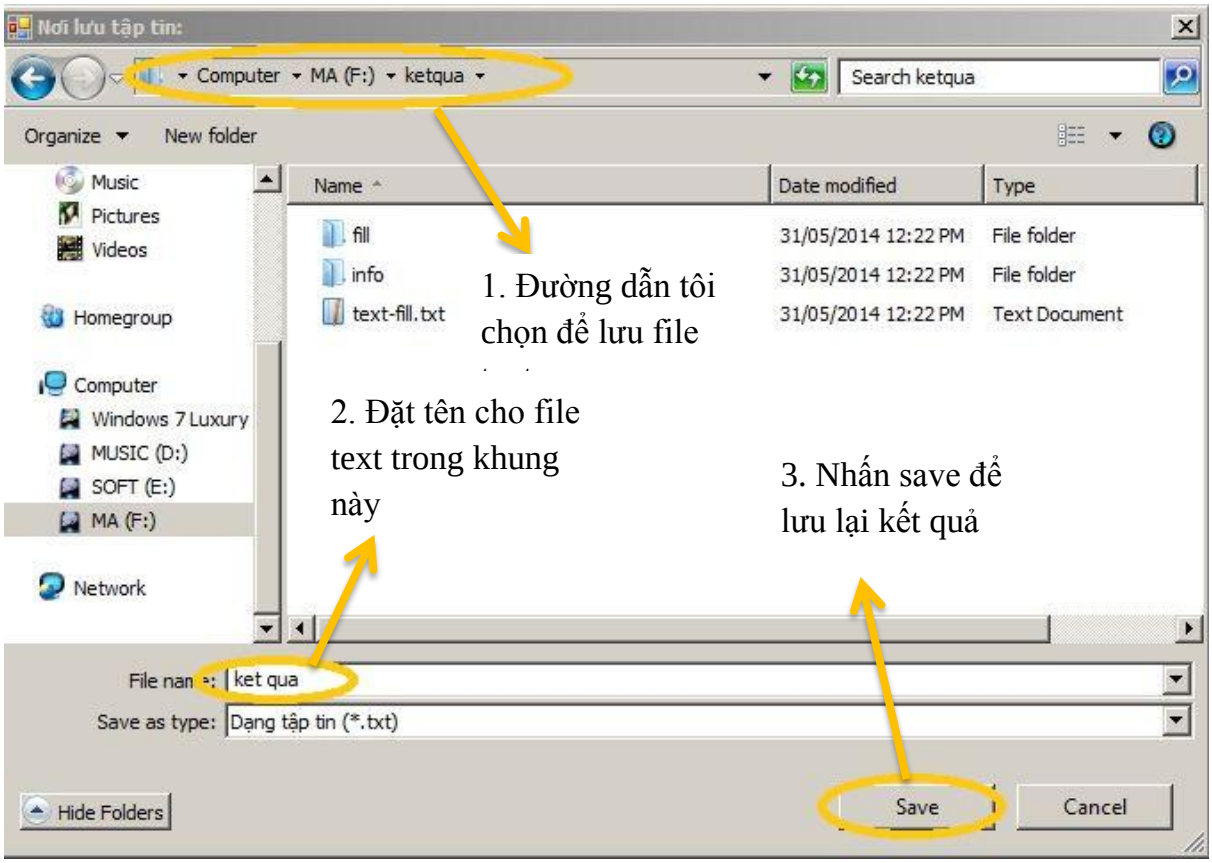
Sau quá trình tính toán, phần mềm in ra kết quả tính được lên khung hiển thị, ta có thể dễ dàng theo dõi kết quả ở đây. Để lưu kết quả tính được vào file text ta làm như sau:

Trong khung thao tác, ta chọn nút lưu file text kết quả



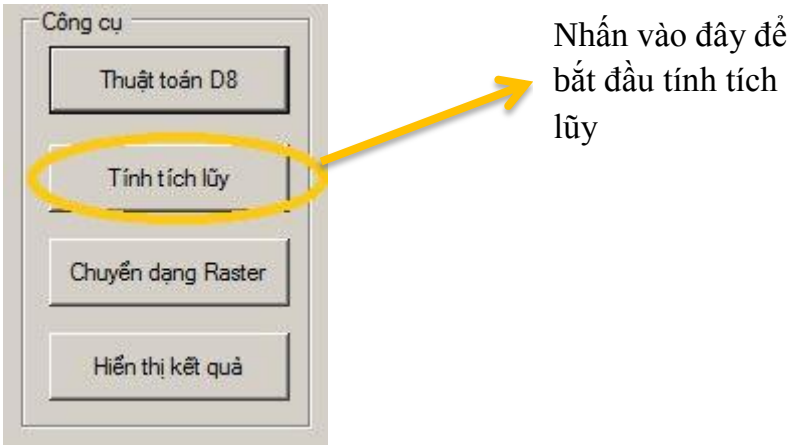
Hình 4.3.5: Chọn nút Lưu file text kết quả

Xuất hiện hộp thoại mới, trong hộp thoại này ta chọn nơi lưu và đặt tên cho file text vừa tính được, sau đó nhấn save để lưu lại.



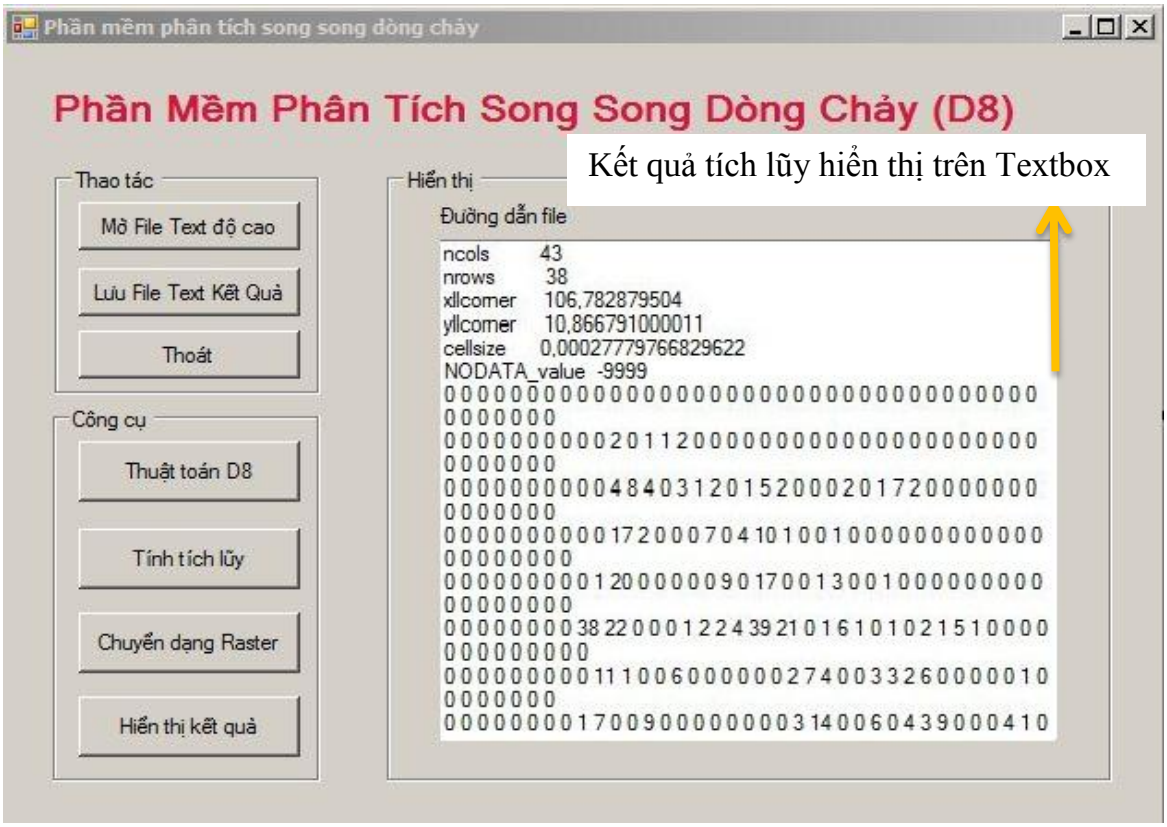
Hình 4.3.6: Chọn tên và đường dẫn lưu

Sau khi đã có kết quả của tính hướng D8 ta sẽ dùng kết quả này để tính tích lũy, muốn tính tích lũy ta nhấn vào chọn nút tính tích lũy trong khung thao tác:



Hình 4.3.7: Chọn nút tính tích lũy

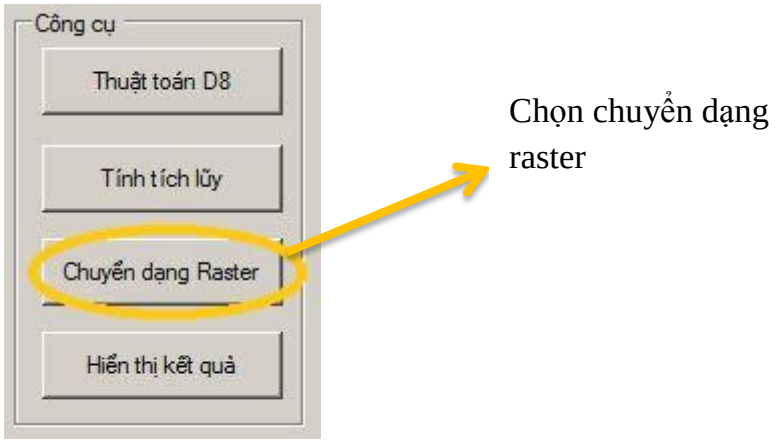
Lúc này chương trình sẽ tự động tính toán dòng tích lũy và khi có kết quả máy sẽ in kết quả ra khung hiển thị, kết quả như sau:



Hình 4.3.8: Kết quả tích lũy in trên Textbox

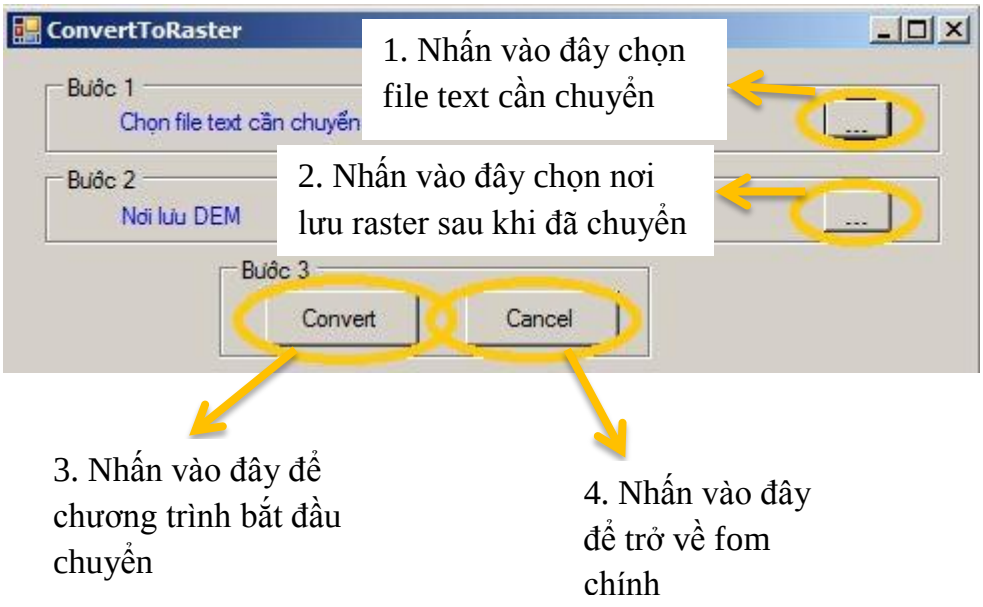
Lưu kết quả tích lũy, cũng như phần tính thuật toán D8 ta chọn nút lưu file text kết quả trong khung thao tác sẽ xuất hiện một hộp thoại cho ta chọn ổ đĩa lưu và lưu tên file tex kết quả. Xong các bước ta sẽ được file text kết quả với tên và đường dẫn như ta đã lưu.

Để chuyển file text kết quả sang dạng raster ta làm như sau. Trong khung thao tác ta chọn chuyển dạng raster để chuyển đổi định dạng file text sang dạng raster.



Hình 4.3.9: Chọn nút chuyển file text sang dạng raster

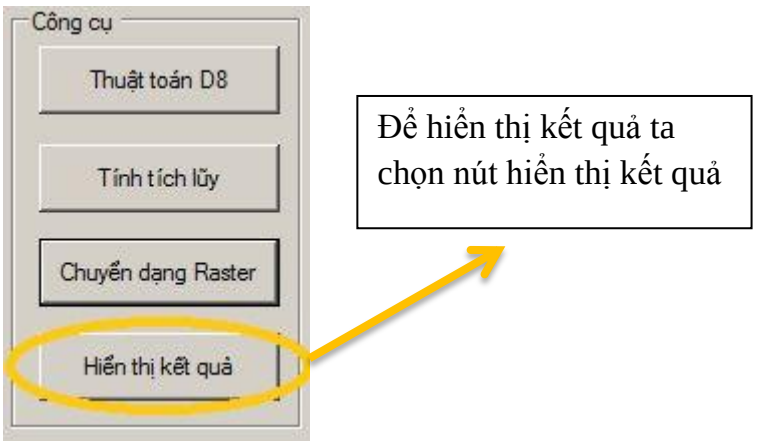
Lúc này phần mềm sẽ chuyển sang form ConvertToRaster để ta có thể dễ dàng thực hiện. Trong Form mới này ta làm tuần tự theo các bước đã được chỉ dẫn. Các bước như sau:



Hình 4.3.10: Sử dụng chức năng chuyển sang dạng raster

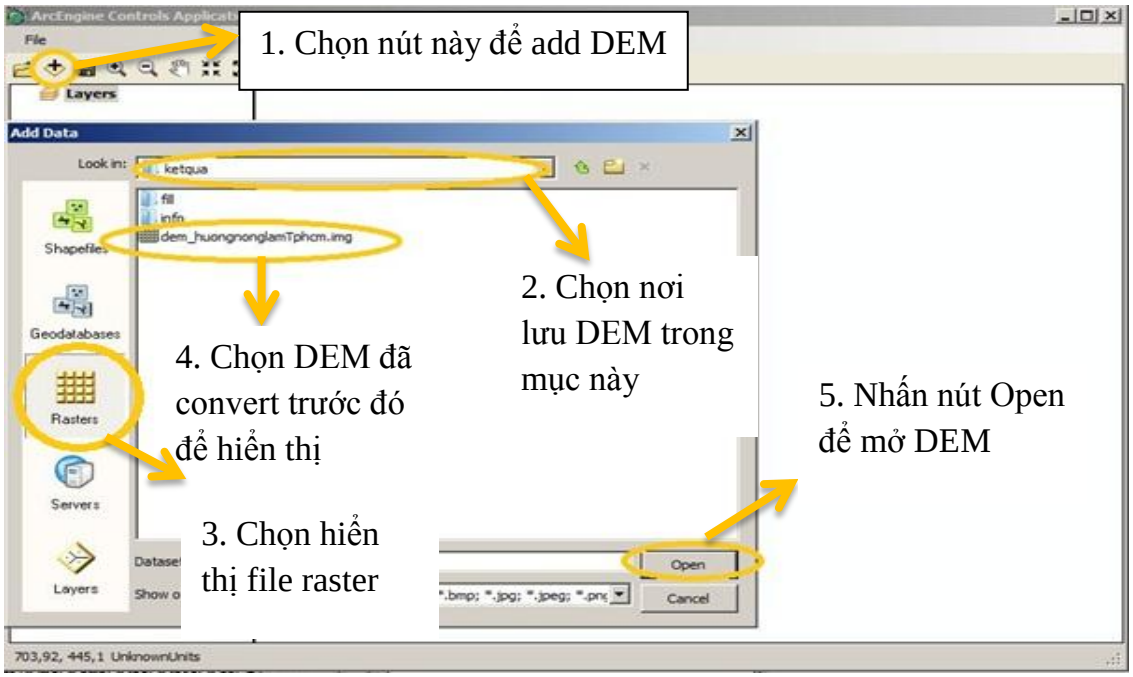
Thực hiện tuần tự theo các bước, chương trình sẽ tự chuyển dữ liệu sang dạng raster, chuyển xong phần mềm sẽ báo "OK" và ta chỉ cần vào đường dẫn mà ta chọn trước đó để xem kết quả.

Sau khi convert xong, để xem kết quả dạng raster trên form ta làm như sau. Trên form chính trong khung công cụ ta chọn nút hiển thị kết quả.



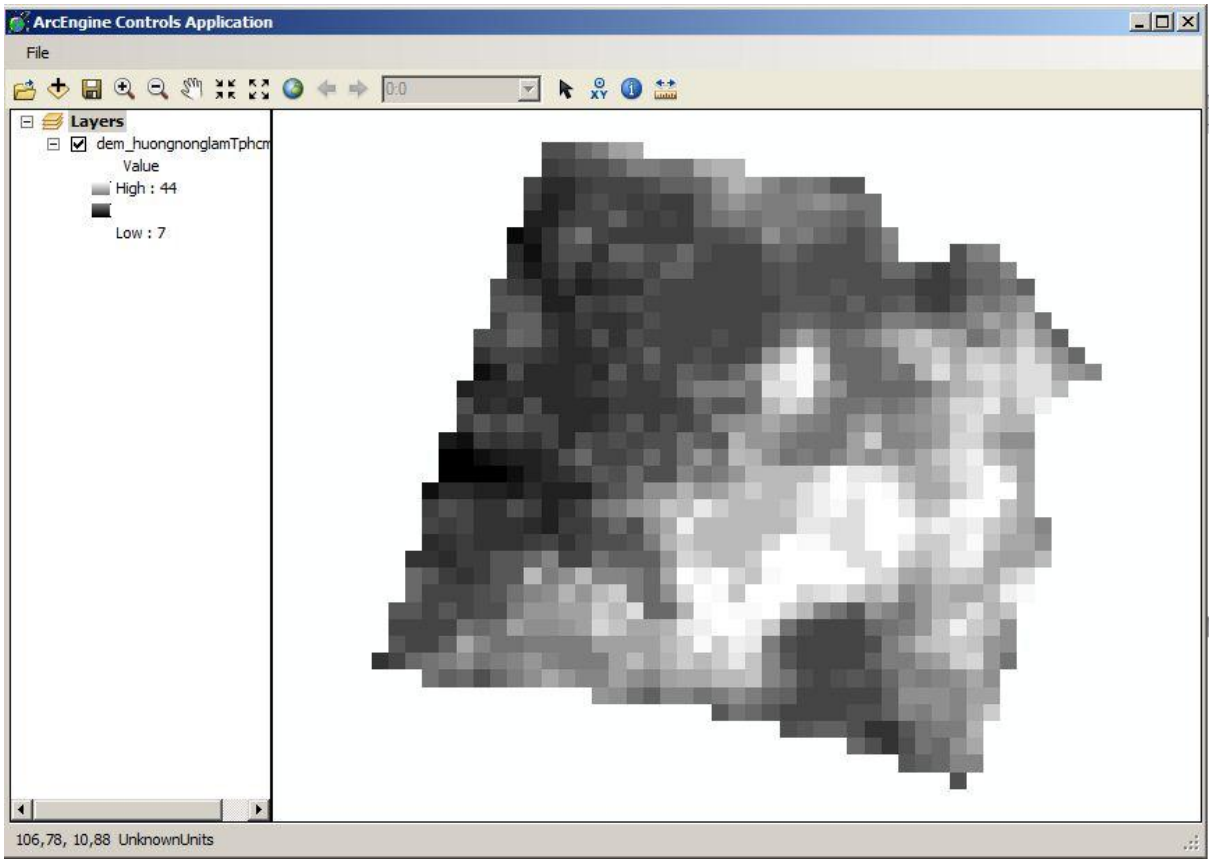
Hình 4.3.11: Chọn nút hiển thị raster lên Form

Sau khi ta chọn nút hiển thị kết quả, phần mềm sẽ tự động chuyển sang Form mới. Trong form này ta làm như sau. Trong form mới ta chọn nút  để add DEM vào form. Xuất hiện hộp thoại mới ta chọn DEM trong hộp thoại này.



Hình 4.3.12: Chọn file raster hiển thị lên form

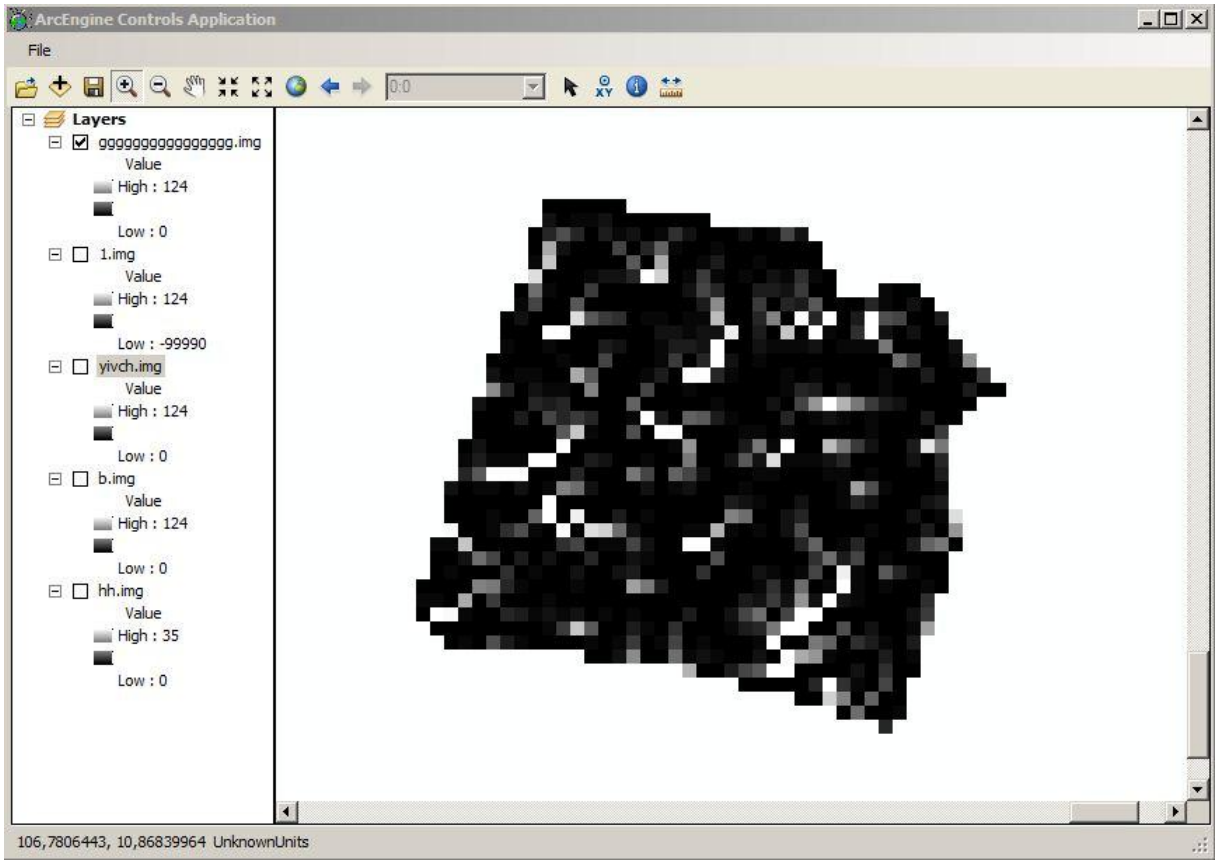
Nhấn nút Open, nếu phần mềm thông báo gì thì nhấn OK. Kết quả của thuật toán tính hướng D8 trên bộ dữ liệu độ cao trường đại học Nông Lâm Tp.HCM dưới dạng raster hiển thị trên Form như sau:



Hình 4.3.13: Kết quả thuật toán D8 dạng raster

Kết quả hiển thị 8 hướng theo D8, chúng ta có thể nhìn rõ trên form này, các ô có cùng hướng sẽ hiển thị cùng một cấp độ xám, theo đó các hướng sẽ được sắp xếp theo thứ tự lớn dần từ 1 đến 128, các hướng càng lớn sẽ có cấp độ xám càng nhỏ. Ngược lại các ô càng nhỏ sẽ có cấp độ xám càng lớn (màu càng đen).

Còn đây là kết quả của thuật toán tìm tích lũy dòng chảy của dữ liệu DEM trường đại học Nông Lâm Tp.HCM



Hình 4.3.14: Kết quả tích lũy dạng raster hiển thị trên form

Kết quả cho ta thấy rõ các dòng tích lũy. Màu tối thể hiện cho những nơi ít và không có ô nào chảy về. Màu càng sáng thể hiện tại điểm đó càng tập trung nhiều ô chảy về. ở góc dưới cùng bên trái có tọa độ thể hiện rõ vị trí của từng điểm ta muốn xét bằng cách rê chuột lên vị trí muốn xét, điều này rất đặc biệt. Dựa ứng dụng này ta có thể nhận thấy rõ dòng chảy của một lưu vực lớn mà ta muốn xét, từ đó ta có thể áp dụng cho các công trình như xây dựng đường xá, cống rãnh hay những vùng thường xuyên có lũ lụt xảy ra, từ đó ta có thể chú ý đặc biệt và cảnh báo cho các vùng ở các vị trí này.

Bảng so sánh kết quả thời gian thực hiện 2 thuật toán được thực hiện trên core i5 – 3230M (4 Cpu – 2.6 Ghz).

Thuật toán / Phép toán	Thuật toán tuần tự	Thuật toán song song
Tính hướng D8	2 giây 626	1 giây 260
Tính tích lũy	2 phút 53 giây 243	1 phút 20 giây 854
Tổng thời gian	2 phút 55 giây 869	1 phút 22 giây 124

Bảng 4.3: So sánh thời gian thực hiện 2 thuật toán tuần tự và song song

Bảng 4.3 so sánh thời gian thực hiện 2 thuật toán tuần tự và song song kết quả cho ta thấy là quá khác biệt, thời gian tiết kiệm được của thuật toán song song là rất lớn so với thuật toán tuần tự. Đặc biệt, nếu bộ dữ liệu càng lớn và những máy được trang bị những lõi cpu đời mới sẽ càng cho ta những kết quả tốt hơn, khả quan hơn, thời gian tiết kiệm được có thể lên đến hàng giờ đồng hồ, thậm chí với những dữ liệu lớn và siêu lớn thì kết quả sẽ tăng lên gấp bội, lên đến hàng ngày, hàng tuần...

PHẦN 5. KẾT LUẬN VÀ KIẾN NGHỊ

5.1. Kết luận

Kết quả, nghiên cứu đã xây dựng được ứng dụng phân tích song song thuật toán định hướng dòng chảy trên bề mặt với các tính năng và ưu điểm sau:

- Ứng dụng có khả năng đọc mở file text ArcGis, lưu dữ liệu kết quả tính được vào file text riêng để ta dễ dàng theo dõi và tiếp tục tính toán về sau.
- Ứng dụng có khả năng sử dụng file text ArcGis nhập vào để xác định hướng dòng chảy (theo D8).
- Ứng dụng có khả năng dùng kết quả của hướng dòng chảy đã xác định trước đó để tính toán sự tích lũy dòng chảy về một ô.
- Ứng dụng có khả năng chuyển dữ liệu dạng file text ArcGis sang dữ liệu dạng raster và hiển thị khác kiểu dữ liệu như dữ liệu bản đồ (shapefile), geodatabases, đặc biệt là raster lên form ứng dụng.
- Ứng dụng đã được sử dụng và điều chỉnh thuật toán song song sao cho ứng dụng có thể khai thác tối đa khả năng của máy tính, luôn luôn đạt tốc độ lớn nhất, tiết kiệm thời gian ngắn nhất cho người sử dụng.

5.2. Kiến nghị

Kết quả của tính tích lũy trong ứng dụng trên đã cho ta bức tranh tổng quát về sự tích lũy dòng chảy của một lưu vực lớn, kết quả hình thành các đường tích lũy mà dòng chảy đi qua. Dựa vào đó ta có thể áp dụng xây dựng cho các công trình giao thông, cầu cống sao cho hiệu quả cao nhất và cảnh báo những vùng có nguy cơ lũ lụt tránh rủi ro hết sức có thể về người và của.

Ứng dụng còn có thể được phát triển lên thành các phiên bản mới với các công cụ lấy kết quả từ tính hướng D8 và tích lũy để xây dựng thêm như công cụ tìm dòng (Stream line), công cụ tìm liên kết dòng (Stream link), tìm cửa xả (watershed) để ứng dụng thêm đa dạng và phục vụ được nhiều ngành, nhiều lĩnh vực từ đó đời sống con nguòi ngày càng được nâng cao hơn.

PHẦN 6. TÀI LIỆU THAM KHẢO

Tiếng việt:

- [1]. **Pgs. Ts. Nguyễn Kim Lợi, ThS. Lê Cảnh Định, Ths. Trần Thống Nhất, 2009,** Hệ thống thông tin địa lý nâng cao, NXB Nông Nhiệp.
- [2]. **Ks. Nguyễn Duy Liêm**, **2013,**Chuyên đề Swat, biên soạn.
- [3]. **Ks. Lê Minh Hoàng**, **2012**, Giải thuật và lập trình, bài giảng chuyên đề, Đại học Sư phạm Hà Nội.
- [4]. **Pgs. Ts. Nguyễn Đức Nghĩa, 2008,** Tính toán song song, bài giảng môn học, NXB Bác Khoa Hà Nội.

Tiếng Anh

- [5]. **Ole Mark,Terry van Kalken,K. Rabbi, Jesper Kjelds,** A mouse GIS study of the drainage in Dhake city.
- [6]. **Peter Van Capelleveen,** Urban drainage network modeling better analyzed using ArcView 3D analyst.
- [7] **Jurgen Garbrecht and Lawrence W Martz, June 1997** The assignment of drainage direction over flat surfaces in raster digital elevation models. Journal of Hydrology, 193:204–213.
- [8] **D.M. Mark, 1988,** Modelling in Geomorphological Systems, chapter Network models in geomorphology.
- [9] **John O'Callaghan and David Mark, December 1984,** The extraction of drainage networks from digital elevation data. Computer Vision, Graphics, and Image Processing.
- [10] **Chase Wallis, Dan Watson, David Tarboton, and Robert Wallace, 2009**, Parallel Flow-Direction and Contributing Area Calculation for Hydrology Analysis in Digital Elevation Models. In International Conference on Parallel and Distributed Processing Techniques and Applications.

Website

- [11]. www.esri.com

- [12]. www.msdn.microsoft.com
- [13]. www.hcmuaf.edu.vn
- [14]. www.hochiminhcity.gov.vn
- [15]. www.stackoverflow.com
- [16]. www.swat.tamu.edu
- [17]. www.proceedings.esri.com
- [18]. www.help.arcgis.com

PHỤ LỤC

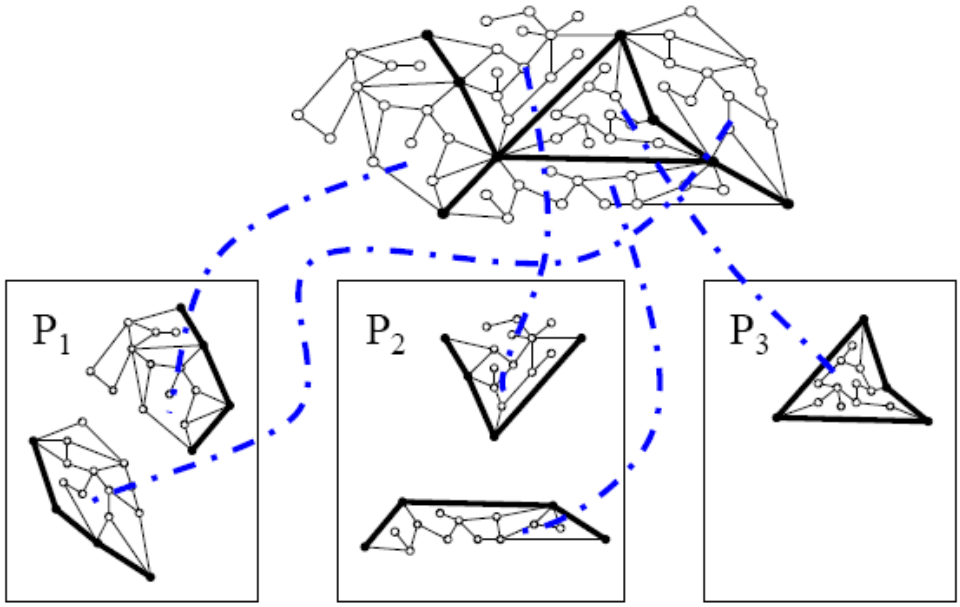
Phụ lục 1: Lý thuyết về tìm đường đi ngắn nhất

Hiện tại, nhiều thuật toán song song tìm đường đi ngắn nhất trên một đồ thị. Chúng ta sẽ nêu một trong những thuật toán, cụ thể là thuật toán do tiến sĩ Lyudmil Aleksandrov đề xuất.

Giả sử, chúng ta có một đồ thị G, n đỉnh. Khi đó, thuật toán sẽ là:

- Giai đoạn tiền xử lý:

- Bước 0: chia đồ thị n đỉnh thành r đồ thị con. Mỗi đồ thị có n/r đỉnh và mỗi đồ thị con có $\sqrt{\frac{n}{r}}$ đỉnh biên ngoài của đồ thị con.
- Bước 1: Tải các đồ thị vào p bộ xử lý.



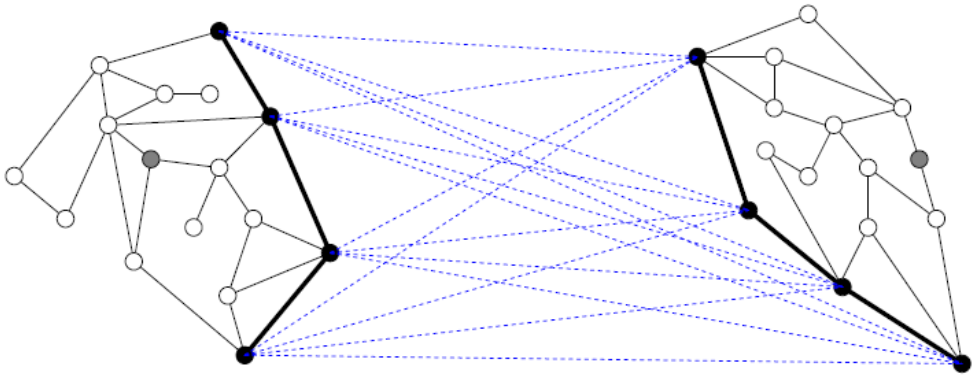
Hình PL1: Hình minh họa việc phân bố r=5 đồ thị con vào p=3 bộ xử lý

- Bước 1: (tiếp theo) Sau đó, mỗi bộ xử lý con sẽ chuyển toàn bộ các đỉnh biên ngoài đến tất cả các bộ xử lý con khác để hình thành một đồ thị G^* có cạnh như G (nhưng chỉ có các đỉnh biên ngoài).
- Bước 2: Tính toán trong mỗi bộ xử lý. Mỗi bộ xử lý sẽ tính toán đường đi ngắn nhất.

- Bước 3: Mỗi bộ xử lý sẽ trau đổi thông tin về biên ngoài. Khi đó giá trị nhỏ nhất sẽ được cập nhật trên đồ thị G^*.
- Bước 4: Mỗi bộ xử lý con sẽ tính toán giá trị ngắn nhất từ các đỉnh còn lại đến các đỉnh của đồ thị G^*.

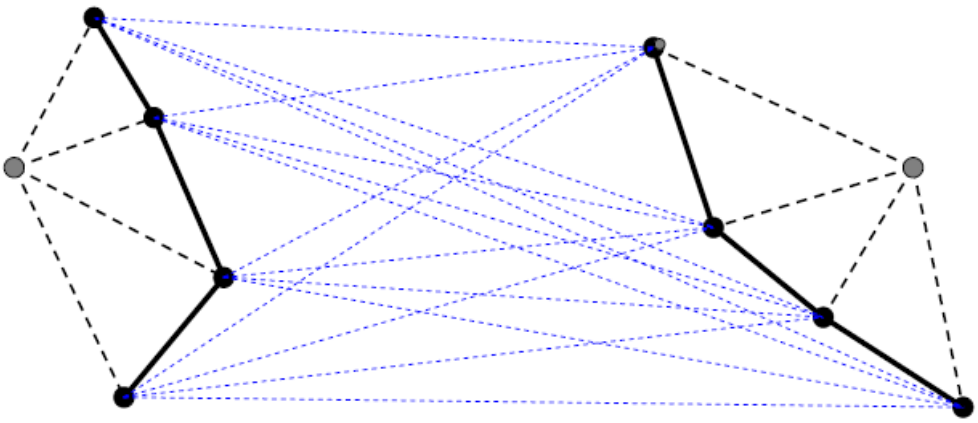
  - Giai đoạn xử lý: Khi có yêu cầu tìm đường đi ngắn nhất:

- Bước 1: Bộ xử lý trung tâm sẽ xác định chính xác vị trí các đỉnh và các bộ xử lý chứa các đỉnh nguồn và đích.



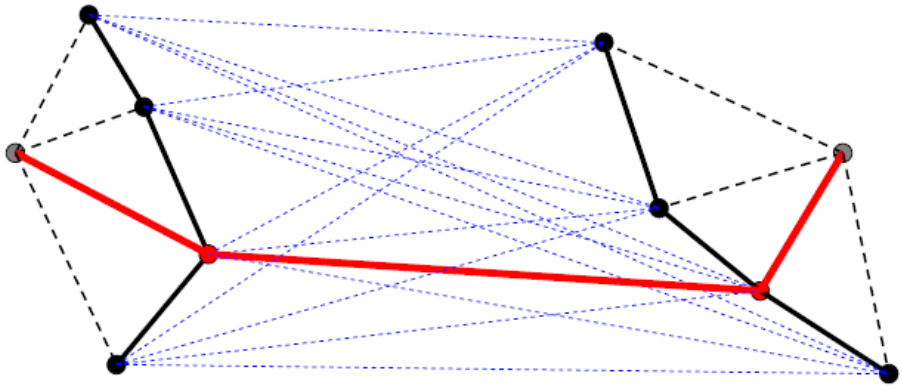
Hình PL2: Nhập điểm nguồn và điểm đích

- Bước 2: Tính giá trị trọng số ngắn nhất giữa mọi đỉnh ngoài của đồ thị chứa điểm nguồn đến đồ thị chứa điểm đích.



Hình PL3 : Tìm khoảng cách ngắn nhất giữa tập điểm biên của đồ thị con chứa điểm nguồn đến đồ thị con chứa điểm đích.

- Bước 3: Đường đi được hình thành bằng cách kết hợp đường đi ngắn nhất từ điểm nguồn đến các đỉnh ngoài của đồ thị chứa điểm nguồn với đường đi tìm được ở bước trên và đường đi ngắn nhất từ điểm đích đến các đỉnh ngoài của đồ thị chứa điểm đích. Kết thúc thuật toán.



Hình PL4 : Xác định được đường đi ngắn nhất và kết thúc thuật toán

Để cài đặt được thuật toán trên, chúng ta phải xây dựng những đối tượng và phương thức như sau:

- Đối tượng đồ thị G,
- Xác định đối tượng các đỉnh trong, đỉnh ngoài của các đồ thị con.
- Đồ thị G^* (bao gồm tập các đỉnh ngoài của các đồ thị con và cạnh của đồ thị G),
- Các node liên kết.

Thuật toán trên đã được chứng minh các vấn đề sau:

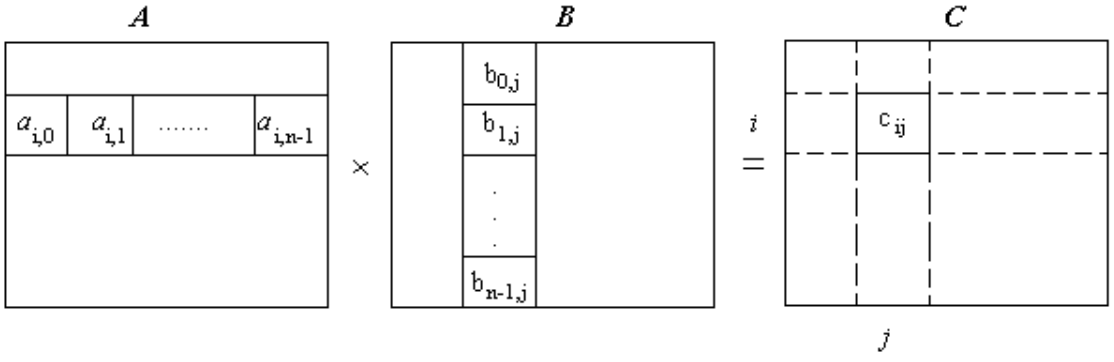
- Thuật toán giải quyết được vấn đề tìm đường đi ngắn nhất.
- Việc truyền thông giữa các node (bộ xử lý) là cực tiểu.
- ...

Từ đây, với bản đồ, chúng ta có thể xây dựng thành đồ thị với các đỉnh là giao giữa các đường và các cạnh là các đường. Đối với các dữ liệu raster, chúng ta có thể phân chia theo không gian đều.

Phụ lục 2: Thuật toán Fox nhân ma trận

Phép toán nhân ma trận là một trong những phép toán có số lượng tính toán lớn. Với hai ma trận A=(a_{ij}) và B = (b_{ij}) với kích thước giả định là nxn. Tích ma trận A.B = C = (c_{ij}) là một ma trận kích thước nxn với mỗi phần tử ma trận C được xác định như sau:

$$c_{ij} = \sum a_{ik} b_{kj}, \text{ với } k=0,n-1 \text{ và } i=0,1,\ldots,n-1 \text{ và } j = 0,1,\ldots,n-1.$$



$$c_{ij} = a_{i0}b_{0j} + a_{i1}b_{1j} + \ldots + a_{in-1}b_{n-1j}.$$

Với thuật toán nhân ma trận được cài đặt theo định nghĩa, ta có độ phức tap của thuật toán là **O**(n^3). Để thực hiện thuật toán Floyd, ta chỉ thực hiện các công việc sau:

- Ma trận a và b là ma trận kề của đồ thị.
- Thay Phép nhân bằng Phép cộng.
- Thay Phép cộng bằng phép lấy giá trị cực tiểu (min).

Vì phép nhân ma trận cần ít nhất tính toán cho n^2 phần tử, do đó, ta cố gắng thực hiện một thuật toán giảm mức độ phức tạp thành O(n^2). Thuật toán Fox về nhân ma trận với ý tưởng thực hiện n bước (từ bước 0 đến bước n-1). Trong mỗi bước, từng giá trị cij được cập nhật. Cụ thể như sau:

- Bước 0: $c_{ij} \leftarrow a_{ii} \times b_{ij}$,
- Bước k: ($1 \le k < n$): $c_{ij} \leftarrow c_{ij} + a_{i\underline{k}} \times b_{\underline{k}j}$, trong đó $\underline{k} = (i+k) \bmod n$.

Điều này nghĩa là, ở bước 0, c_{ij} được tính giá trị là $a_{ii} \times b_{ij}$. Trong các bước tiếp theo giá trị c_{ij} được cộng dồn giá trị hiện hữu với giá trị mới được xác định là: $a_{ik} \times b_{kj}$.

Tư tưởng của thuật toán nhân ma trận Fox là giá trị của c_{ij} được xác định bằng tổng các tích của các phần tử kế tiếp $a_{i,i+k}$ trong cùng hàng đối với ma trận A và phần tử kế tiếp $b_{i+k,j}$ trong cùng cột đối với ma trận B. Do đó, hai vấn đề đặt ra:

- (1) Chỉ số của các phần tử A và B trong các bước $1 \leq k < n$ khi $i+k >= n$
- (2) Xác định vị trí đối với trường hợp giá trị cij cần tổng các tích của các phần tử nằm bên trái cùng dòng đối với ma trận A và nằm phía trên cùng cột đối với ma trận B.

Tổng hợp hai vấn đề trên, ta chọn giá trị $\underline{k} = (i+k) \bmod n$ sẽ đáp ứng hai yêu cầu (1) và (2). Trong thực nghiệm, công thức ở các bước lặp $1 \leq k < n$ có ý nghĩa đảm bảo giá trị c_{ij} được cộng dồn một giá trị mà vẫn được định dạng công thức: $\underline{c}_{ij} = \underline{a}_{ii} \times \underline{b}_{ij}$. Để làm được việc này, trên thực tế, người ta sẽ thực hiện việc dịch chuyển xoay vòng các giá trị của ma trận A và B. Cụ thể là:

- Thực hiện xoay dòng đối với ma trận A, đưa dòng i+1 thành dòng i, các giá trị dòng 0 được đưa xuống dòng n-1.
- Tương tự, thực hiện việc xoay cột đối với ma trận B, đưa cột j+1 thành cột j, các giá trị cột 0 được chuyển sang cột n-1.

Ứng dụng trong tìm đường đi ngắn nhất

Phép nhân ma trận A và B trên có thể được song song hóa với số bộ xử lý p ($p < n$). Trước tiên, các phần tử của ma trận được phân phối phù hợp số lượng đến p bộ xử lý nhằm đảm bảo việc tính toán tương đối cân bằng trên các bộ xử lý. Nếu A, B, C đều là các ma trận vuông dạng qxq, ta nên phân chia thành các khối ma trận con có kích thước $\underline{n} \times \underline{n}$ với $q = \sqrt{p}$ và $\underline{n} = n/q$. Tại mỗi bước $k>0$, tiến trình pij ($0 \leq i,j \leq q$) tính giá trị của khối con c_{ij} là tích của $a_{i,\underline{k}}$ (được broadcast bởi tiến trình $p_{i,k}$ trên cùng dòng i) với $b_{k,j}$ từ tiến trình "láng giềng phía nam" của tiến trình $p_{\underline{k},j}$…

Tóm lại, khi cài đặt trong một lưới bộ xử lý kích thước pxp (với n=kp) thì ma trận A được tách thành các ma trận A_{ij}, tương tự ma trận B được tách thành các ma trận B_{ij}. Và mỗi ma trận A_{ij} và B_{ij} được giao cho mỗi bộ xử lý p_{ij}. Dưới đây là minh họa ma trận A(4x4) được tách thành 4 ma trận để 4 bộ xử lý tính toán:

$$A_{00} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \quad A_{01} = \begin{pmatrix} a_{02} & a_{03} \\ a_{12} & a_{13} \end{pmatrix}$$

$$A_{10} = \begin{pmatrix} a_{20} & a_{21} \\ a_{30} & a_{31} \end{pmatrix} \quad A_{11} = \begin{pmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{pmatrix}$$

Khi đó, ma trận C=A.B được xác định sau k(=n) giai đoạn tính toán như sau:

$\mathbf{C}_{ij} = \mathbf{A}_{ii}\mathbf{B}_{ij} + \mathbf{A}_{i,i+1}\mathbf{B}_{i+1,j} + ... + \mathbf{A}_{iq-1}\mathbf{B}_{q-1j} + \mathbf{A}_{i0}\mathbf{B}_{0j} + ... + \mathbf{A}_{i,i-1}\mathbf{B}_{i-1,j}$.

Từ đó, ta có mã giả cài đặt thuật toán như sau:

```
// tiến trình/bộ xử lý p theo dòng là i, theo cột là j
q = sqrt(p);
dest = ((i-1) mod q, j);
source = ((i+1) mod q, j);
for (stage = 0; stage < q; stage++) {
    k_bar = (i + stage) mod q;
(a) Broadcast A[i,k_bar] across process row i;
(b) C[i,i] = C[i,i] + A[i,k_bar]*B[k_bar,i];
(c) Send B[k_bar,j] to dest; Receive
    B[(k_bar+l) mod q,j] from source;
}
```