**Chương 1 : Giới thiệu đề tài - đặt vấn đề**

**1.1) Nhu cầu về High-performance computing**

- Thời đại Internet : xem video trực tuyến, ...: các máy tính, thiết bị di động, điện thoại ngày 1 mạnh hơn dẫn đến nhu cầu xem trực tuyến ngày 1 cao. Phía server phải có khả năng xử lý mạnh phục vụ cùng 1 lúc hàng nghìn người.

- Sự mở rộng của các lĩnh vực Khoa học-Công nghệ, Hàng không, Kinh tế/Tài chính/Thương mại, Y tế, Quân sự : các lĩnh vực ngày càng phổ cập và phát triển mọi lúc mọi nơi --> đòi hỏi hạ tầng tính toán cao

- Nhu cầu về xử lý thông tin nhanh, lớn, chính xác dẫn tới cần một hạ tầng tính toán high performance

**1.2) Môi trường tính toán (phần cứng ngày nay)**

- Phục vụ cho mục đích trên có những cách tiếp cận sau đây

**+ Hosts mạnh** : CPU mạnh, RAM tốt, Bus nhanh, GPU, card mạng

Mạnh : - Chất lượng phần cứng được cải thiện bao gồm kích thước giảm đi, giá thành rẻ nhưng tốc độ và tính chính xác của xử lý lại cao lên.

Yếu : - Có 1 số ứng dụng thì có thể xử lý trên 1 máy nhưng giá thành cao, nhưng chưa chắc đã tối ưu xử lý vấn đề cùa High performance computing, nhưng ngược lại vẫn có thể giải quyết trên nhiều máy nhỏ hơn và chi phí thấp hơn . 🡺 giải pháp cluster/grid ra đời.

**+ Cluster**

Mạnh : gồm nhiều máy tính trong đó 1 máy tính chịu trách nhiệm phân task

Yếu : 1 server quản lý thì nguy cơ xảy ra risk (server phân chia, điều phối task bị hỏng và không có thay thế) => grid ra đời

**+ Grid**

Mạnh : gồm nhiều máy tính, vừa client, vừa là server cho các máy tính khác trong việc cùng giải quyết 1 nhiệm vụ nào đó

Yếu : Khó quản trị hơn, đặc biệt trong môi trường biến động

**- Kết luận** : môi trường phần cứng có thể không hetegeneous

**1.3) Môi trường lập trình (phần mềm)**

- Phục vụ High-performance computing

+ Nhóm MPI (process) : thực hiện và phân chia task đồng đều trên host

+ Multithread programming : thực hiện task trên core + time sharing

+ Công nghệ GPU : so sánh với CPU, giới thiệu về thế mạnh khi lập trình với CUDA

- Các ứng dụng hiện nay mới khai thác chủ yếu trên đồng thời 2 môi trường, chưa khai thác trên đồng thời 3 môi trường

**1.4) Mục đích bài luận - kết luận**

- Chính vì chưa có kinh nghiệm khai thác trên 3 môi trường lập trình nên luận văn này tập trung thử nghiệm, giải quyết vấn đề này

- Xử lý video (xem trực tuyến) là 1 ứng dụng thích hợp để thử nghiệm vấn đề trên

- đề tài khảo cứu ".." được triển khai trong luận văn này sẽ thể hiện cả 2 vấn đề trên.

P/S : làm trên cluster (1 đại diện chủ đạo giao tiếp ra ngoài)

#########################

**Chương 2 : Các môi trường lập trình**

**2.1) Kiến trúc hỗ trợ lập trình song song trên nhiều Hosts**

**2.1.1) Môi trường lập trình (vừa trên cluster, vừa trên grid)**

* **MPI**

- MPI là một môi trường lập trình ứng dụng mạng chạy trên nhiều host đồng thời. Mỗi host có một hay nhiều process (multi-process) thực thi task của nó. Trong quá trình thực thi task của nó, process tương ứng có thể trao đổi dữ liệu với các process khác trên các host còn lại.

- Môi trường lập trình MPI được chuẩn hóa và cho phép dùng chung mã nguồn (hoặc thay đổi không đáng kể) trên nhiều platform khác nhau nhằm khai thác tối đa phần cứng mạng hiện có.

* **Gridware**

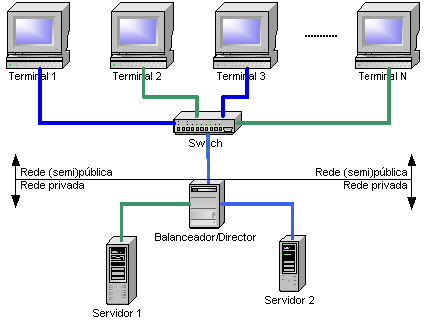
- Là môi trường lập trình song song trên nhiều host. Gridware ngoài việc bao gồm môi trường lập trình (edit, debug) như MPI thì nó còn cung cấp môi trường thực thi song song (deploy, quản trị, khai thác tài nguyên và ứng dụng), được ứng dụng trong tính toán hiệu năng cao.

**-** Một số Gridware điển hình **:** Alces Gridware, Oracle grid engine, Univa Grid engine .

**2.1.2) Mô hình phần cứng đi kèm : grid và cluster**

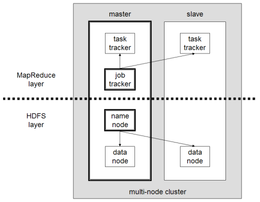
**Cluster** : hệ thống máy tính kết nối với nhau, thường thông qua mạng LAN, cùng xử lý các tác vụ liên quan đến ứng dụng.

* ***Kiến trúc***

 - Thường 1 máy tính đóng vai trò Server có nhiệm vụ phân phối các tasks cho các máy tính khác trong cluster và nhận kết quả từ chúng, rồi tổng hợp va xử lý cho ra kết quả cuối cùng của ứng dụng.

*(cluster: wikipedia)*

- Để có thể trao đổi thông tin giữa Master với các máy khác, hệ điều hành của chúng thường có middleware (thư viện) dùng để hỗ trợ chúng trao đổi thông tin.



*(cluster wikipedia : A multi-node Hadoop cluster)*

* ***Hoạt động***

Có 2 hoạt động cụ thể sau đây :

* Task scheduling
* Khi các máy tính của người dùng truy cập vào cluster để yêu cầu xử lý thông tin thì nhiệm vụ của Master là lập lịch, phân task cho các máy slave còn lại trong hệ thống. Đối với một cluster mà các slave của nó không đồng nhất về phần cứng (CPU-GPU), thì hiệu quả làm việc của mỗi slave sẽ rất khác nhau, nên việc phân chia công việc sẽ phức tạp hơn rất nhiều.
* Các công việc cần thực hiện được lưu trữ chung trong một kho (task/job pool). Trong quá trình thực thi một task ở một slave có thể phát sinh thêm nhiều task mới. Chúng cũng được đưa về kho công việc nói trên để chờ phân phối khi có slave trống.
* Node failure management

Các trường hợp rủi ro và phương án giải quyết :

* 1 slave bị sự cố (mất điện, hư hỏng, bị rút khỏi cluster…): phát hiện sự cố, công việc dở dang được giao lại cho slave khác, cô lập nó khỏi cluster để bảo vệ tài nguyên chung.
* Slave đã được phục hồi/thêm mới : Master phải phát hiện và tiếp tục phân phối việc mới cho nó.
* ***Công dụng***
* Phục vụ cho High Performance Computing : xử lý thông tin song song và hiệu quả.
* ***Khuyết điểm***
* Dễ bị rủi ro khi Master bị hỏng và không có phương án dự phòng.
* ***Một số hệ thống cluster***
* Hệ thống Beowulf (middleware): các máy tính khác tương tác đến Master (Server lập lịch, quản lý các máy tính còn lại). Master có 2 network interface, một để giao tiếp với các máy tính còn lại, một để giao tiếp với bên ngoài. Các server được quản lý thường có cùng hệ điều hành, nhưng có thể khác phiên bản; thường có cùng tài nguyên.



(wikipedia)

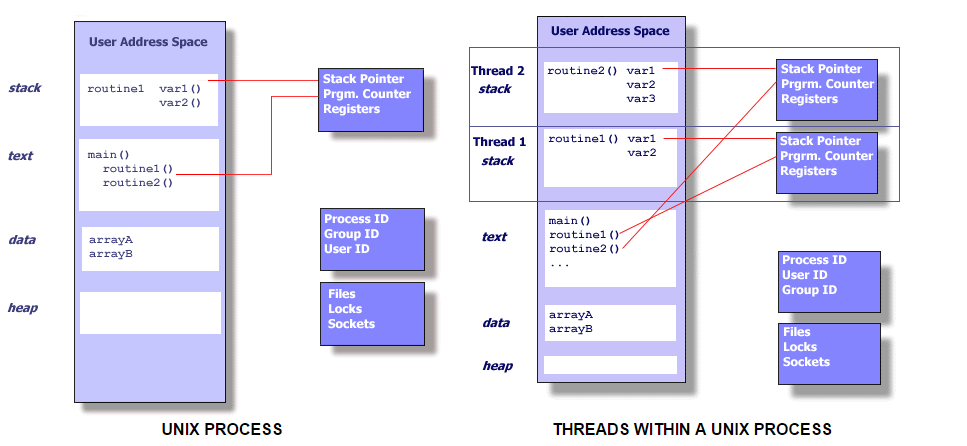
**Grid** : là hệ thống các máy tính kết nối với nhau để thực hiện các ứng dụng chung, nhưng khác với cluster mỗi máy tính trong hệ thống là 1 master (đồng thời nó cũng là slave cho các master khác). Grid chính là một hệ thống tính toán phân tán mà trong đó tài nguyên của một máy được chia sẻ với các máy khác.

* ***Công dụng:*** Tránh khuyết điểm của Cluster : khi Master node bị sự cố thì ảnh hưởng đến toàn cluster. Ngược lại grid quản lý phần cứng, phần mềm và dữ liệu một cách phân tán và động, đảm bảo toàn hệ thống hoạt động suôn sẻ dù có sự cố xảy ra đối với phần cứng, phần mềm hoặc dữ liệu.
* ***Khuyết điểm:*** Khó quản trị hơn, đặc biệt trong môi trường biến động.

**2.2) Kiến trúc hỗ trợ lập trình song song trên nhiều core**

**2.2.1) Multi-thread programming**

* *Mô hình : (VD trên hệ điều hành Unix)*



*(*[*https://techmaster.vn/posts/33604/su-khac-nhau-giua-process-va-thread*](https://techmaster.vn/posts/33604/su-khac-nhau-giua-process-va-thread)*)*

*// Giải thích sự khác biệt giữa tiểu trình và tiến trình : khởi động nhanh, tiến trình phải lưu lại trong đĩa cứng còn tiểu trình thì lưu các thanh ghi*

* *Cơ chế hoạt động:*

*+ 1 process có thể gọi nhiều tiểu trình (đa luồng)*

*+ Sử dụng chung vùng nhớ để xử lý các công việc quy mô nhỏ chỉ ở phạm vi trong 1 process.*

*+ Các tiểu trình có cấu trúc chung, và cần phải điều phối 1 cách hiệu quả để khỏi tranh chấp tài nguyên hay quyền truy cập vùng găng.*



*(*[*https://computing.llnl.gov/tutorials/openMP/#ProgrammingModel*](https://computing.llnl.gov/tutorials/openMP/#ProgrammingModel)*)*

*// giải thích hình sơ đồ.*

* *Công dụng*

*+ Tốn ít tài nguyên so với tiến trình. (chưa rõ)*

*+ Xử lý nhanh hơn. (chưa rõ)*

*+ Hạn chế được sự phức tạp, cồng kềnh của vấn đề “Message-passing”.*

**2.2.2) Time sharing (nằm trên)**

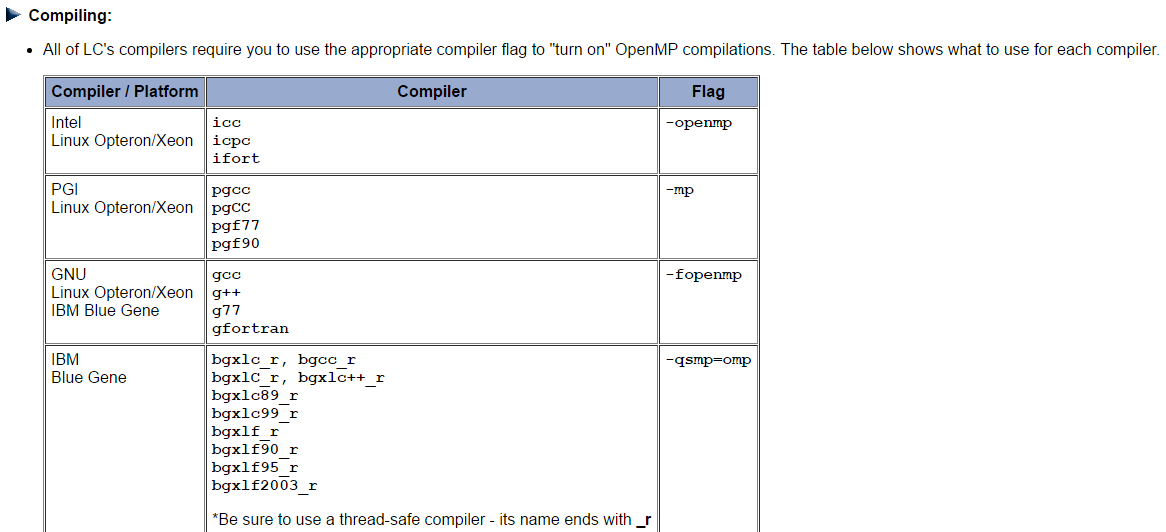
- Là việcchia sẻ thời gian sữ dụng của một hoặc nhiều core (trên cùng một host) giữa các tiến trình /tiểu trình. Việc chia sẻ này được thực hiện ở hai cấp độ: thực thi ở mức hệ điều hành (multitasking) và ở mức độ lập trình (multiprogramming).

**2.2.3) Môi trường lập trình điển hình**

Có các môi trường điển hình để lập trình song song như là :

**OpenMP: 🡪 sửa lại cấu trúc**

* Là API được cung cấp cho hoạt động đa tiểu trình, chia sẻ vùng nhớ chung. OpenMP gồm 3 thành phần:
* Compiler Directives
* Các comments trong mã nguồn, sẽ được trình biên dịch bỏ qua trừ khi ta định rõ chúng là các cờ hiệu *(Gốc: Compiler directives appear as comments in your source code and are ignored by compilers unless you tell them otherwise - usually by specifying the appropriate compiler flag)*.
* Các cờ hiệu biên dịch của từng môi trường:



* Mục đích :
  + - Khai báo vùng thực hiện song song
    - Chia mã nguồn ra các block để thực thi tiểu trình
    - Chú thích vòng lặp
    - Đánh dấu section của mã nguồn
    - Điều phối tiểu trình
* Runtime Library Routines
* Bao gồm các thư viện run-time để cho thực thi, sử dụng trong chương trình

*(Gốc: The OpenMP API includes an ever-growing number of run-time library routines)*

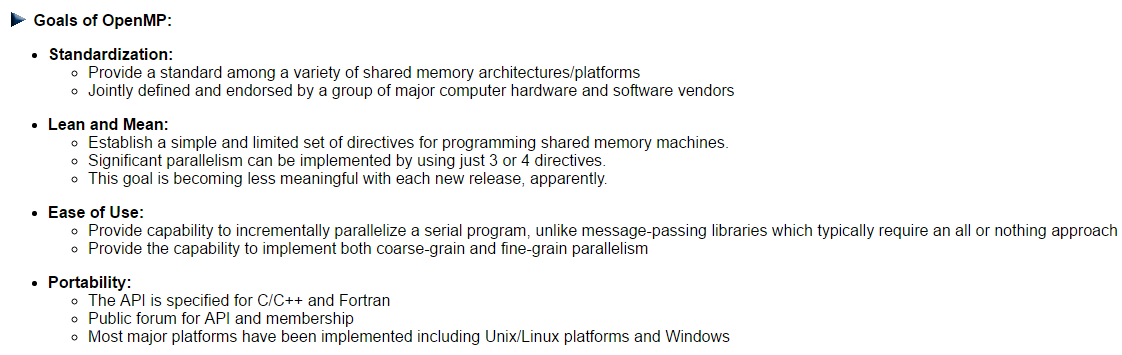
* Mục đích sử dụng :
* Cài đặt và truy vấn đến một lượng các tiểu trình : ID, ID cha, kích thước.
* Cài đặt và quản lý các khóa của tiểu trình
* Truy vấn thời gian và giải pháp
* Environment Variables
* Kiểm soát việc chạy các đoạn mã nguồn song song trong suốt quá trình chạy của chương trình

*(Gốc: OpenMP provides several environment variables for controlling the execution of parallel code at run-time)*

* Công dụng
* Kiểm soát tiểu trình (kích thước vùng nhớ stack, chính sách chờ)
* Binding thread vào vi xử lý (processors)
* Kiểm soát việc chạy song song nhiều công việc bao gồm giới hạn mức lồng vào bao nhiêu tiểu trình nhỏ cho 1 tiểu trình lớn
* Mục tiêu của OpenMP là: (đưa lên trên)
* Standardization : Tiêu chuẩn giữa các kiến trúc và môi trường mà cho phép chia sẻ vùng nhớ

+ Lean and Mean : Thiết lập các chỉ thị lập trình tạo nên các máy cho phép chia sẻ vùng nhớ

* Ease of use : cung cấp khả năng thiết lập sự đồng thời của chương trình, dù chương trình có các task cồng kềnh hay đơn giản.
* Portability : API được xây dựng trên C/C++, và Fortran và có thể được thực thi trên cả môi trường Windows lẫn Linux/Unix



**Socket programming :**

* Lập trình ứng dụng mạng trao đổi thông tin dựa vào socket, khác với trao đổi thông tin dựa vào message (RPC).

**2.3) Kiến trúc hỗ trợ lập trình trên GPU**

**2.3.1) GPU**

**Giới thiệu :**

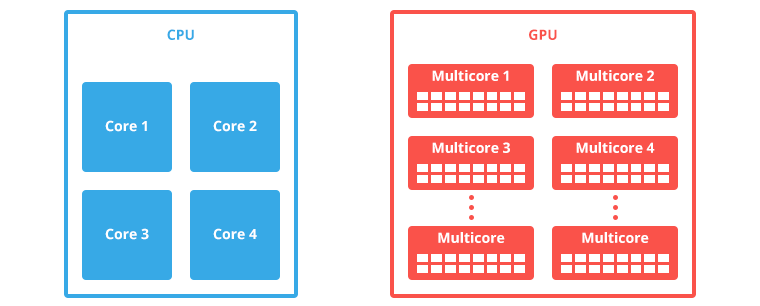
- GPU (Graphics Processing Unit) là phần cứng chuyên dụng được thiết kế để xử lý về hình ảnh, video trong máy tính.

- So với CPU, GPU sẽ xử lý khối lượng rất lớn các dữ liệu đồ họa một cách đồng thời tốt hơn vì nó được thiết kế theo cấu trúc song song cao cấp

- GPU có thể được tích hợp sử dụng trong hệ thống nhúng, điện thoại di động, máy tính cá nhân, máy trạm và máy chơi game. Trong các máy tính cá nhân, GPU có thể là card màn hình, hoặc được tích hợp luôn trên bo mạch chủ của máy. Tuy nhiên GPU được tích hợp sẵn trên máy sẽ không mạnh so với các GPU chuyên dụng được tích hợp trên card màn hình tháo rời.

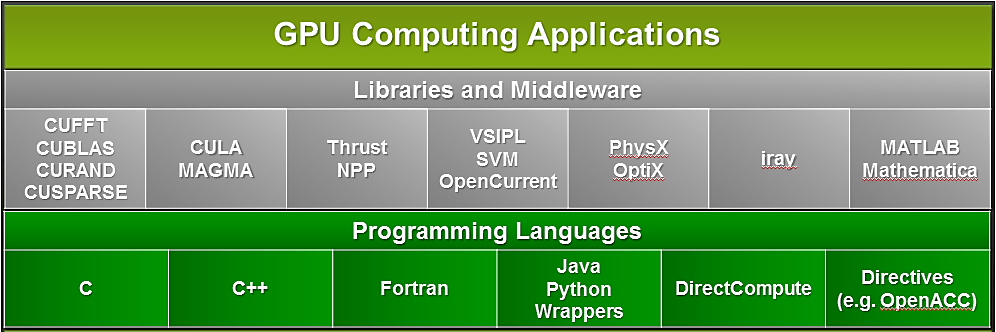
**Kiến trúc :**

* CPU được thiết kế gồm nhiều core (CPU của máy tính cá nhân hiện nay có khoảng 4 core). Khi được nhận một khối dữ liệu và thao tác, thì CPU sẽ chia ra cho nhiều Core xử lý. Mỗi Core có thể phải thực hiện khối thao tác rất lớn nếu máy phải xử lý dữ liệu liên quan đến đồ họa như hình ành hay video. Rõ ràng điều này la một bất lợi đối với CPU. Tuy nhiên, GPU được tích hợp bởi một lượng lớn chip xử lý nhỏ(multicore) cho phép xử lý đa luồng, song song với tốc độ tối ưu hóa hơn so với CPU.



[*https://huytd.github.io/posts/nhan-ma-tran-2.html*](https://huytd.github.io/posts/nhan-ma-tran-2.html)

**Mô hình ứng dụng tính toán dựa trên GPU:**

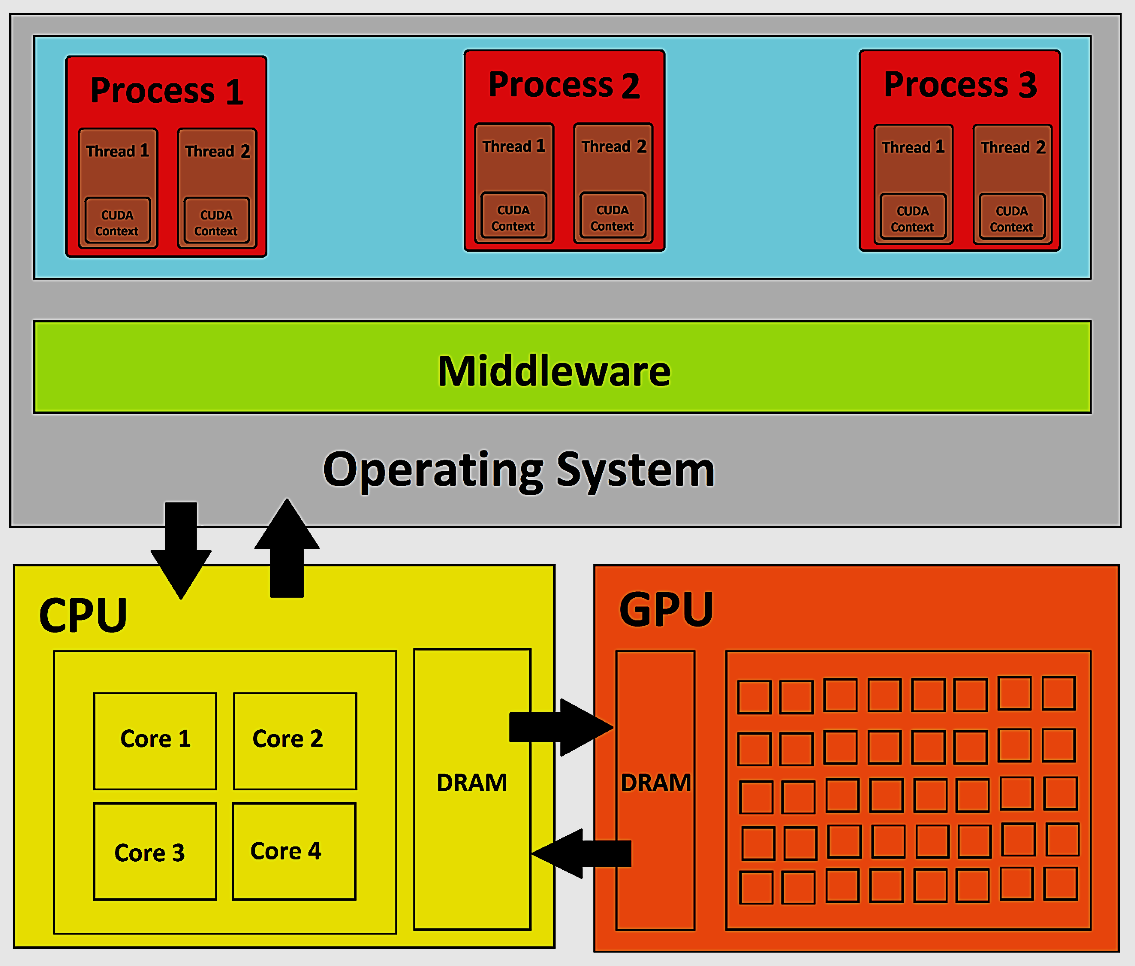


***http://strongartificialintelligence.com/projects***

* Gồm 2 lớp :
* Thư viện và middleware : CUDA (CUFFT, CUBLAS, CURAND…) và các nền tảng khác như MATLAB,..

- **Thư viện**: CUDA (CUFFT, CUBLAS, CURAND…) và các nền tảng khác như MATLAB…

- **Middleware** : lớp trung gian tiếp nhận, giao tiếp với các kernel của các tiến trình, tiểu trình (tạo ra bởi ứng dụng) sau đó sử dụng CUDA API để ghi lên và yêu cầu GPU xử lý. Middleware tiếp nhận tiến trình/tiểu trình theo cơ chế hàng đợi.



*(Luận văn : hình minh họa hoạt động của middleware)*

* Ngôn ngữ lập trình : C/C++, Fortran, Directives (OpenACC)…

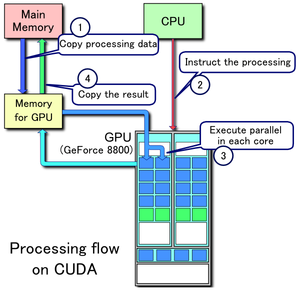
**2.3.2) CUDA**

**Giới thiệu :**

**-** CUDA (phát triển bởi NVIDIA) nằm ở lớp « libraries và middleware », là một trong những nền tảng cho phép giải quyết tính toán đồ họa trên GPU. CUDA hỗ trợ tính toán song song và cũng là một mô hình lập trình tận dụng khả năng xử lý của GPU. CUDA sử dụng ngôn ngữ C/C++ và Fortran.

**Hoạt động :**

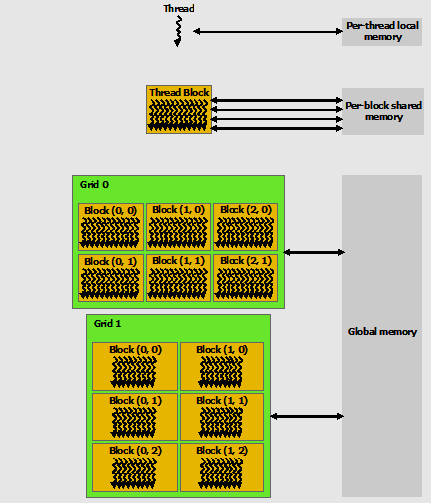
* **Mô hình xử lý tổng quát** :



*(CUDA: wikipedia)*

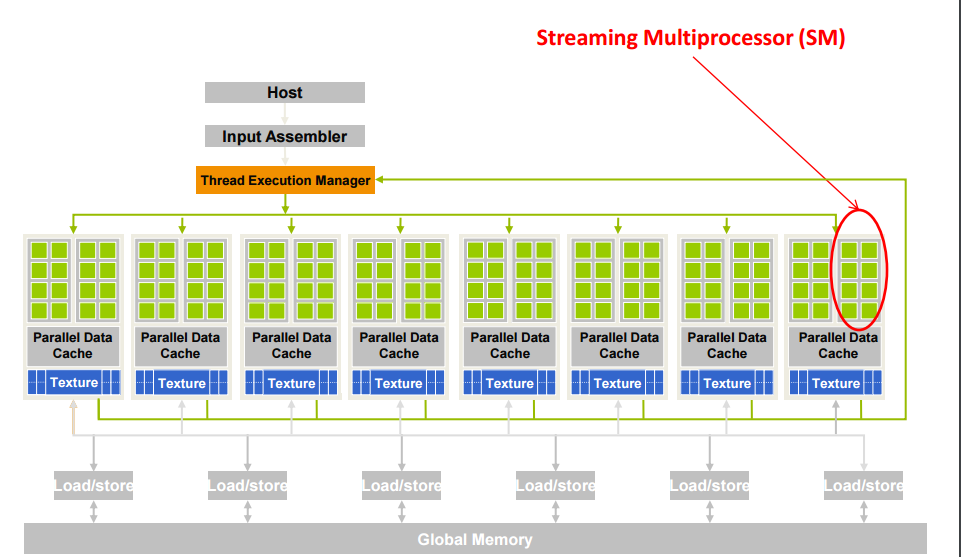
1. Đưa dữ liệu (kernel và data của nó) từ bộ nhớ chính (Host memory) vào vùng nhớ của GPU (device memory).
2. CPU ra lệnh GPU xử lý
3. Kernel được thực thi tại các core của GPU (kết quả trả về device memory).
4. Copy kết quả từ device memory về lại bộ nhớ chính (Host memory).

* **Quản lý tiểu trình** :



*(luận văn)*

Các tiểu trình đi vào GPU sẽ được phân phối ra ở nhiều Block. Các block này lại được phân phối vào trong các grid có kích thước n \* m (n là số Streaming Multiprocessor (SM) của GPU, m là số block của một SM).

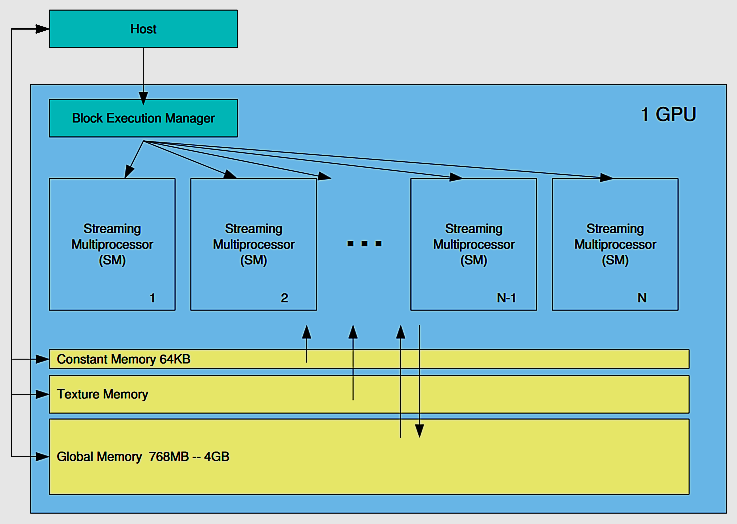
Mỗi block có một vùng nhớ chung (per-block shared memory) để các tiểu trình chia sẻ. Dĩ nhiên tiểu trình ở block này không thể sử dụng tài nguyên ở block khác. Tuy nhiên tất cả các block đều có thể sử dụng Global Memory – một vùng nhớ toàn cục của GPU.

*(http://cs.nyu.edu/courses/spring12/CSCI-GA.3033-012/lecture1.pdf)*

Mỗi SM của GPU chỉ có thể xử lý được tối đa 8 block tiểu trình. Rõ ràng khi nhìn trên hình rằng, ta thấy CPU càng có nhiều SM, thì sẽ xử lý được đồng thời nhiều task đi vào. Điều này đồng nghĩa với việc GPU càng có nhiều lõi thì càng được dữ liệu đồ họa nhanh và hiệu quả.

Giới hạn số tiểu trình mỗi block là 1024. Nếu block đó chứa quá số lượng tiểu trình và cần lượng tài nguyên quá so với SM, thì block đó sẽ không được thực thi.

**Giao tiếp giữa CPU và GPU:**



Trong CUDA, CPU đảm nhiệm phần thực thi toàn cục của chương trình còn GPU thì sẽ chờ lệnh giaao từ GPU. Quá trình giiao tiếp giữa CPU và GPU diễn ra theo trình tự như sau:

* Cấp phát bộ nhớ trên GPU
* Sao chép dữ liệu từ Host qua GPU
* Thực thi kernel trên GPU
* Sao chép dữ liệu từ GPU trở về Host
* Giải phóng bộ nhớ đã cấp trên GPU

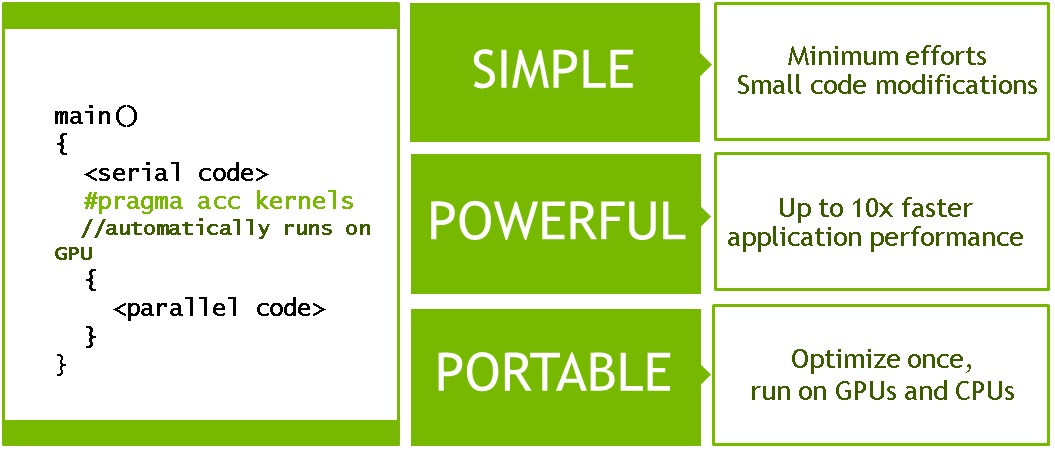
**2.3.3) OpenACC**

**Giới thiệu:**

OpenACC là một mô hình lập trình (cũng có thể gọi là tiêu chuẩn) được thiết kế để tăng tốc độ xử lý dữ liệu của GPU một cách đơn giản nhất mà không đòi hỏi quá nhiều nỗ lực gia công mã nguồn (More science, less programming). Môi trường OpenACC đơn giản hóa việc tính toán song song trong những hệ thống mà mỗi thiết bị không đồng nhất với nhau về năng lực phần cứng.

OpenACC được phát triển bằng ngôn ngữ C/C++, Fortran.

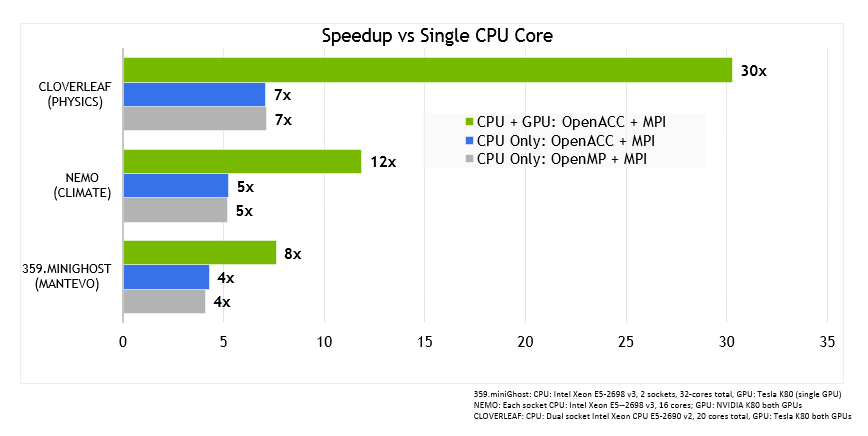
**Công dụng :**



*(https://developer.nvidia.com/openacc/overview)*

* Simple : Giảm thiểu công sức của người lập trình khi gia công mã nguồn
* Powerful : Cùng một mã nguồn nhưng hiệu năng tính toán và xử lý cao hơn 10 lần
* Portable : có thể chạy trên mọi môi trường GPUs/CPUs

**So sánh thực tiễn:**



*(https://developer.nvidia.com/openacc/overview)*

**Chương 3 : Ứng dụng minh họa**

**3.1) Phát biểu bài toán**

**3.1.1) Yêu cầu của khách hàng**

Ngày nay nhu cầu theo dõi bằng camera từ xa đang trở nên phổ biến hơn vì các lý do như an ninh, trình diễn, hội nghị, tư vấn/đào tạo từ xa. Đối với hệ thống đơn giản ngày nay, các luồng video camera được đưa vào hệ thống và được xử lý tập trung thành một luồng duy nhất vào một server. Tuy nhiên trên thực tế, ta có những yêu cầu quan sát rất phức tạp - những tòa nhà, chung cư, cơ quan hay siêu thị lớn. Trong hệ thống này, các camera được chia thành nhiều nhóm quan sát có nhiệm vụ theo dõi các khu vực khác nhau. Ví dụ như:

* Đối với chung cư/khu tập thể lớn hoặc cơ quan, sẽ có:
* Nhóm camera dành cho quan sát khuôn viên
* Nhóm camera theo dõi khu vực gửi xe
* Nhóm camera theo dõi hành lang, cầu thang đi bộ, thang máy…
* Đối với siêu thị: Ngoài các nhóm camera trên, siêu thị còn cần lắp đặt các nhóm camera đặc biệt như:
* Nhóm camera quan sát quầy thanh toán
* Nhóm camera theo dõi từng khu vực quầy hàng

Ngoài ra, thứ nhất là tuỳ vào quy mô theo dõi mà số lượng camera sẽ có thể thêm vào hoặc bớt đi thường xuyên tùy theo sự kiện nào đó như hội nghị, trình diễn, giới thiệu hàng hóa. Thứ hai là số lượng người dùng truy cập cũng sẽ thay đổi.

Vì vậy, xây dựng một hệ thống có khả năng tính toán hiệu suất cao nhưng có khả năng tiết giảm được kinh phí đầu tư thiết bị và băng thông là một yêu cầu cấp thiết. Điều này đòi hỏi rất nhiều vào công việc lập trình sao cho lợi thế phần cứng được khai thác một cách tối đa.

**3.1.2) Tổng quát về ứng dụng**

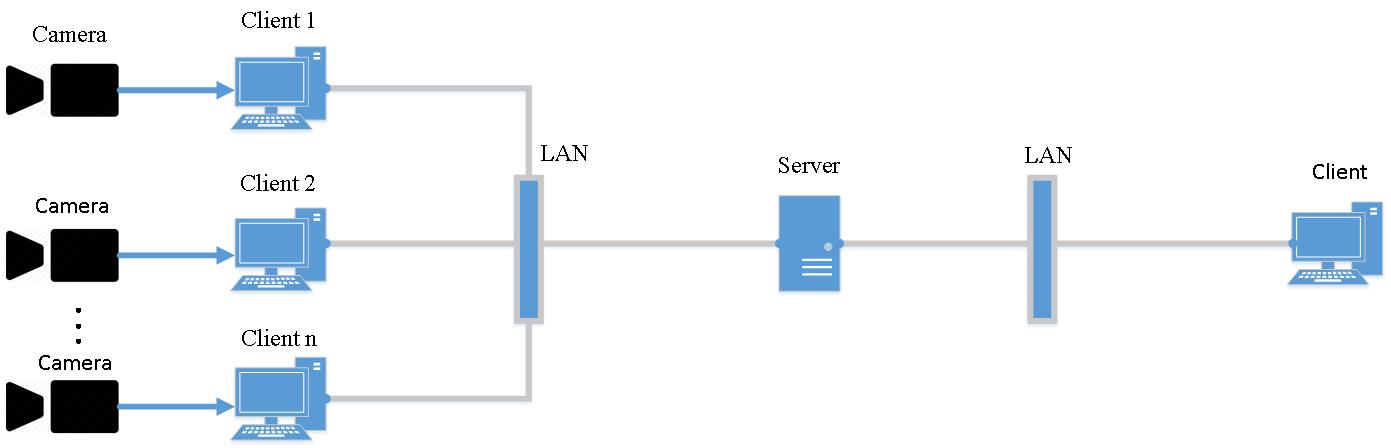
Các camera được chia thành nhiều nhóm khác nhau là , số lượng camera trong mỗi nhóm là và có thể thay đổi tùy theo sự kiện. Song song với đó, số lượng người có nhu cầu xem cũng có thể tăng lên hay giảm đi đột biến tùy theo quy mô theo dõi. Vấn đề được đặt ra có thể là băng thông bị hạn chế, hoặc lợi thế thiết bị phần cứng không đủ đáp ứng nhu cầu.

Ý tưởng của luận văn được đưa ra là xây dựng một hệ thống nhận nhiều luồng video từ rất nhiều camera ở các vị trí khác nhau mà ứng dụng của hệ thống cho phép nén các luồng này thành một luồng duy nhất sao cho phù hợp với băng thông mạng hiện có của mỗi người dùng. Bản chất của việc này là thực hiện giảm kích thước frame ảnh từ các luồng, sau đó đặt chúng vào vị trí tương ứng trên một frame duy nhất để thu được frame chứa tất cả các frame ảnh từ các luồng khác. Công việc này sẽ được áp dụng một nền tảng tính toán mới nhưng rất nhiệu quả là công nghệ GPU. Vì GPU xử lý đồ họa hiệu quả hơn so với CPU nhưng kinh phí lại ngày càng rẻ đi.

**3.1.3) Mô hình ban đầu**

Luận văn năm 2016 của Trần Văn Hoàng và Đào Tặng Thưởng – khoa Công nghệ Thông tin Đại học Khoa học Tự nhiên TPHCM đã đề cập tới vấn đề này và đưa ra phương án giải quyết : hệ thống một Host.

Hệ thống này nhận video từ nhiều camera. Các luồng video tập trung vào một server, sau đó ứng dụng sẽ nén lại thành một luồng duy nhất ngay tại server đó. Khi đó các máy client (người dùng) nếu có nhu cầu theo dõi thì phải kết nối đến server thông qua mạng LAN. Server có thể chỉ có CPU không có GPU, hoặc có một hay nhiều GPU.

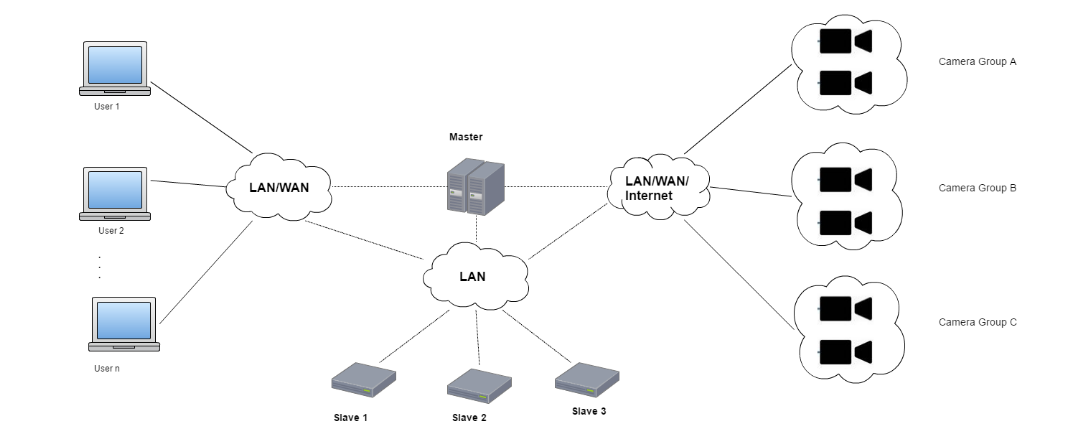


Tuy nhiên hệ thống của Hoàng và Thưởng có khác biệt là chỉ có thể phục vụ cho số lượng nhỏ người dùng, và số lượng camera cũng không thể quá lớn vì nó chỉ có một server. Do đó nhiệm vụ của chúng ta là phải mở rộng được ra nhiều host, như vậy khối lượng công việc tập trung cho một server sẽ giảm đi đáng kể. Điều này làm hệ thống xử lý nhanh hơn và phục vụ được quy mô rộng lớn hơn.

Để giải quyết vấn đề của chúng ta, giải pháp được kế thừa trong luận văn của Hoàng và Thưởng là:

* Máy server chỉ có một CPU: công đoạn nén video được thực hiện trên host memory.
* Máy server có một hoặc nhiều GPU: công đoạn nén video được thực hiện trên device của GPU.

**3.1.4) Mô hình mở rộng được đề nghị**



Trên cơ sở tiếp nối ứng dụng luận văn 29-06 của Hoàng và Thưởng : phát triển một mạng cluster có quy mô rộng hơn gồm nhiều host thay vì một host như trước kia.

Trong ứng dụng này, các camera sẽ được phân thành nhiều nhóm theo dõi các khu vực khác nhau. Ứng dụng phục vụ nhu cầu người dùng khi họ muốn theo dõi một khu vực nhất định nào đó.

Cluster gồm có hai thành phần và chức năng của chúng là :

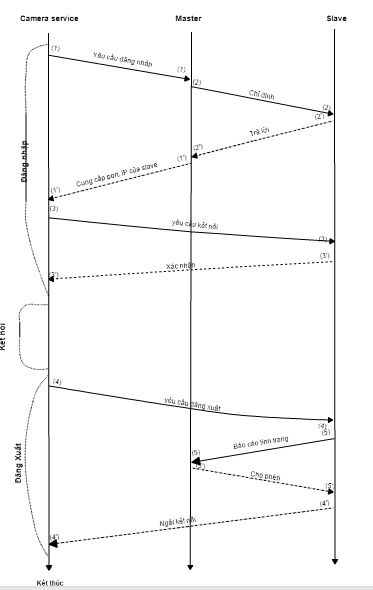
* Master: chịu trách nhiệm quản lý các slave (server) như giao tiếp cho phép người dùng và các camera truy cập ra vào hệ thống, phân phối công việc, lập lịch cho các slave và chịu trách nhiệm điều phối lại công việc khi một trong các slave bị xảy ra sự cố.
* Slave: chịu trách nhiệm capture lại hình ảnh các camera (trong cùng một nhóm) thu được và chuyển giao trực tiếp cho máy người dùng.

Sự tương tác giữa các camera, người dùng và cluster được chia thành 5 kịch bản tương ứng với các module chức năng sau đây:

* Chức năng 1 (*quản lý camera : đăng nhập, phân phối và đăng xuất*) Camera đăng nhập vào hệ thống thông qua Master (camera phải được gia nhập vào một nhóm). Master sẽ cung cấp cho camera port, địa chỉ IP của slave đảm nhận nhóm mà camera thuộc về. Kịch bản này chỉ xảy ra một lần duy nhất. Trước khi camera được tháo ra khỏi nhóm, thì trước tiên phải thực hiện thủ tục đăng xuất với Master. Nếu Camera đổi khu vực, thì phải đăng xuất kết nối qua Master và đăng nhập lại từ đầu thông qua master.
* Chức năng 2 (*quản lý user: đăng nhập, phân phối và đăng xuất*) - người dùng đăng nhập vào hệ thống để yêu cầu quan sát một khu vực nào đó: người dùng sẽ gửi yêu cầu đăng nhập tới Master; Master sẽ cung cấp port, địa chỉ IP của slave đảm nhận khu vực người dùng muốn theo dõi để người dùng kết nối tới. Kịch bản này chỉ xảy ra một lần duy nhất. Nếu người dùng đổi khu vực theo dõi thì phải đăng xuất kết nối với Master và đăng nhập lại qua Master. Nếu người dùng thôi theo dõi, thì chỉ cần phải thực hiện thủ tục đăng xuất với Master.
* Chức năng 3 (*Giao tiếp giữa Master và Slave*) - Master quản lý : Master có nhiệm vụ phân phối công việc capture hình ảnh của camera mới đăng nhập và lập lịch cho các slave. Mỗi slave sẽ được Master giao đảm nhận xử lý dữ liệu hình ảnh từ một hay nhiều khu vực có camera nào đó. Mạng giao tiếp giữa các slave và Master là cục bộ (LAN). Master còn có nhiệm vụ phân phối lại công việc trong trường hợp đặc biệt: slave bị hư, thêm/bớt slave… Hơn nữa mỗi slave phải có nghĩa vụ thông báo thường xuyên tình trạng của mình cho Master (có thể định kỳ hoặc khi có thay đổi).
* Chức năng 4 (*Giao tiếp giữa slave và camera*) : slave nhận trực tiếp các luồng video từ nhiều camera (trong cùng một nhóm/khu vực) và nén chúng lại ngay sau khi camera thực hiện đăng nhập với Master. Slave và camera giao tiếp thông qua mạng LAN, WAN hoặc Internet mà không còn phải qua trung gian Master. Luồng hình ảnh capture từ camera chính là tài nguyên của slave.
* Chức năng 5 *(Giao tiếp giữa slave và máy người dùng) :* Slave chịu trách nhiệm phân phối luồng video đã nén cho người dùng thông qua mạng LAN/WAN ngay sau khi user đăng nhập lần đầu tiên vào hệ thống. Lúc này máy tính user sẽ giao tiếp trực tiếp với slave để xem hình ảnh từ camera mà không phải qua trung gian Master nữa. Ngoài ra trong chức năng này, người dùng có thể phóng to hay thu nhỏ màn hình video tùy theo ý của mình.

**3.1.5) Kịch bản của mỗi chức năng**

* **Quản lý camera: đăng nhập, phân phối và đăng xuất**



1. Yêu cầu đăng nhập: camera khi được gắn vào hệ thống sẽ gửi yêu cầu đăng nhập đến Master. Trong yêu cầu này gồm có ID nhóm của camera.
2. Chỉ định: Master chỉ định việc thu hình của camera cho slave phụ trách nhóm đó

(2’) Trả lời : Slave sẽ trả lời cho Master để cho biết đã nhận chỉ định và cho Master biết nó có thể phụ trách camera này hay không. Nếu không, Master sẽ gửi chỉ định tới slave khác.

(1’) Master trả lời lại camera để xác nhận kết quả đăng nhập của camera. Nếu kết quả trả lời gồm có port, IP của slave tức là camera đăng nhập thành công, còn kết quả trả lời là ký tự rỗng tức là đăng nhập không thành công.

1. Nếu Master cho phép camera đăng nhập, thì camera sẽ gửi yêu cầu kết nối tới slave theo port và IP đã được Master cung cấp cho.

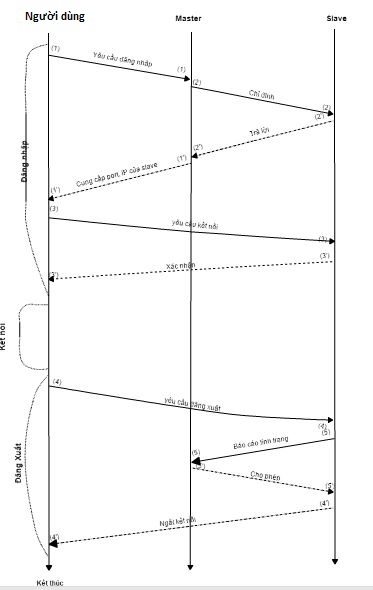
(3’) Slave gửi trả lại xác nhận kết nối : thành công hay không. Nếu thành công, cả hai bên bắt đầu thực hiện kết nối và slave sẽ capture hình trực tiếp từ camera.

1. Sau một thời gian kết nối, camera muốn rời khỏi hệ thống thì phải gửi yêu cầu đăng xuất tới slave phụ trách nó.
2. Báo cáo tình trạng: slave nhắn tới master rằng có camera sẽ rời hệ thống, yêu cầu sự cho phép của Master. Master sẽ cập nhật lại trạng thái của slave vào lịch quản lý của mình.

(5’) Master gửi xác nhận cho phép camera đăng xuất khỏi hệ thống đến lại cho slave.

(4’) Slave ngắt kết nối với camera, hủy tài nguyên cấp phát cho camera đó.

* **Quản lý user: đăng nhập, phân phối và đăng xuất**

****

1. Yêu cầu đăng nhập: Người dùng có nhu cầu theo dõi sẽ gửi yêu cầu đăng nhập đến Master. Trong yêu cầu này gồm có ID khu vực (nhóm mà camera quan sát).
2. Chỉ định: Master chỉ định việc đáp ứng nhu cầu xem video của người dùng cho slave phụ trách nhóm đó.

(2’) Slave sẽ trả lời cho Master để cho biết đã nhận chỉ định và cho Master biết nó có thể phụ trách camera này hay không. Nếu không, Master sẽ gửi chỉ định tới slave khác.

(1’) Master trả lời lại người dùng để xác nhận kết quả đăng nhập. Nếu kết quả trả lời gồm có port, IP của slave tức là người dùng đăng nhập thành công, còn kết quả trả lời khác tức là đăng nhập không thành công.

1. Nếu Master cho phép người dùng đăng nhập, thì người dùng sẽ gửi yêu cầu kết nối trực tiếp tới slave theo port và IP đã được Master cung cấp cho.
2. Sau một thời gian theo dõi, người dùng muốn hủy thì phải gửi yêu cầu đăng xuất tới slave.
3. Báo cáo tình trạng: slave nhắn tới master rằng có người dùng sẽ rời hệ thống, yêu cầu sự cho phép của Master. Master sẽ cập nhật lại trạng thái của slave vào lịch quản lý của mình.

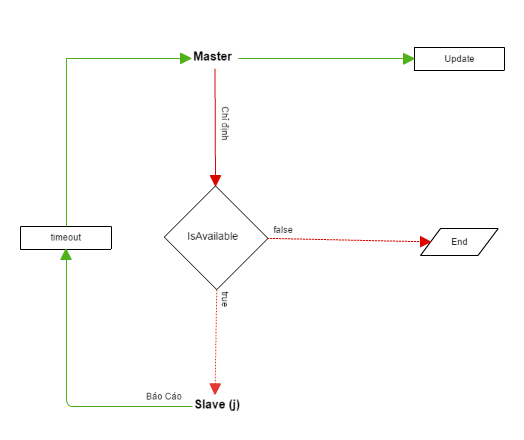
(5’) Master gửi xác nhận cho phép người dùng đăng xuất khỏi hệ thống đến lại cho slave.

(4’) Slave ngắt kết nối với người dùng, hủy tài nguyên cấp phát cho người dùng đó

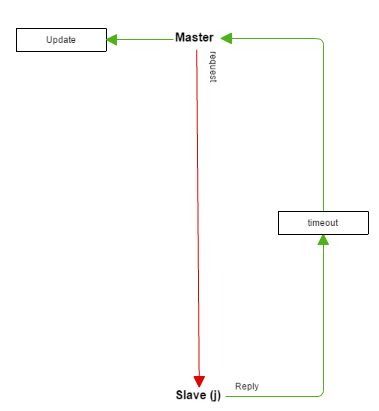
* **Phân phối camera cho slave**

**Quy trình hoạt động**

* Khi một camera đăng nhập vào hệ thống cùng với ID nhóm hay khu vực mà nó quan sát thì Master có nhiệm vụ chỉ định công việc cho slave phụ trách nhóm camera đó. Master sẽ kiểm tra trong “lịch phân công” (“bảng trạng thái”) của mình xem slave có thể phụ trách thu hình của camera này hay không theo các điều kiện sau: số lượng camera của nhóm đang kết nối, khả năng hoạt động của slave… Các thông tin này thường xuyên được cập nhật vào “lịch phân công” dựa theo các báo cáo định kỳ mà các slave gửi cho Master. Nếu các điều kiện này không được thỏa (isAvailable = false) thì việc chỉ định sẽ kết thúc. Lúc đó một bảng thông báo sẽ xuất hiện trên màn hình camera cho biết lý do của việc đăng nhập không thành công.
* Các camera gửi báo cáo tình trạng của mình cho Master theo định kỳ. Master sẽ cập nhật liên tục “lịch phân công” của mình. Để phát hiện sự cố, ta dùng cơ chế timeout: khi thời gian một slave không gửi báo cáo vượt quá t, tức là slave đã bị sự cố.



**Quy trình xử lý sự cố**

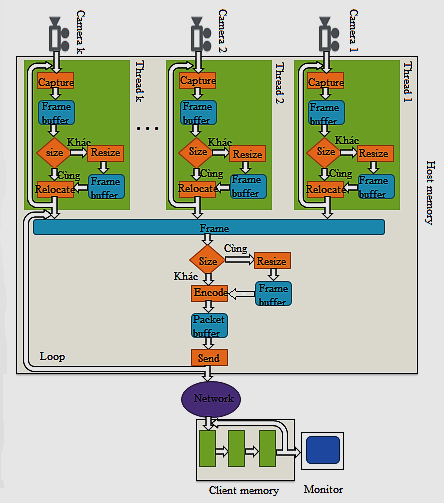
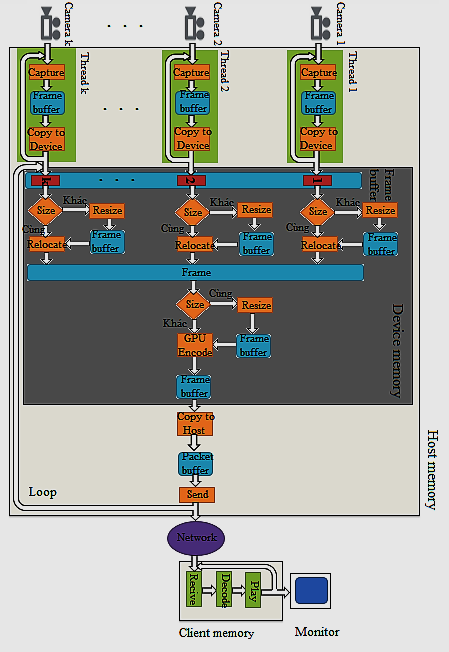
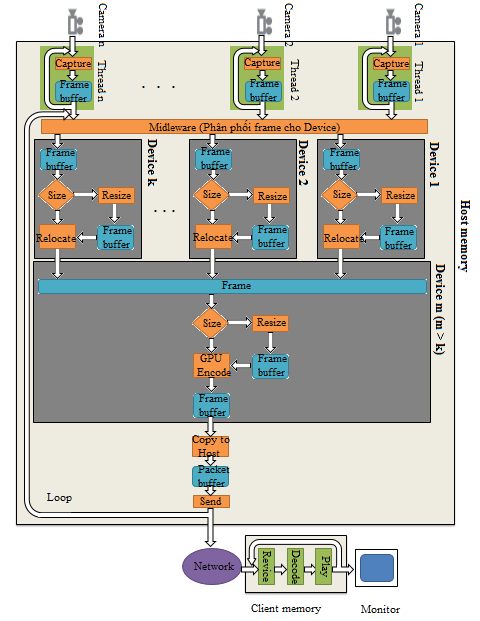


Master sẽ gửi cho Slave một yêu cầu request tới Slave nghi bị sự cố và đợi phản hồi từ slave đó trong khỏang thời gian timeout.

Nếu qua timeout mà Slave không phản hồi thì Master sẽ tự động cập nhật vào bảng trạng thái.

Ngược lại thì Master sẽ tiếp tục kiểm tra nếu Slave đủ điều kiện phụ trách việc thu hình từ camera.

Reply từ slave chính là các thông tin về tình trạng của nó.

* **Giao tiếp giữa slave và camera**
* Môi trường phần cứng chỉ có CPU
* Môi trường phần cứng có 1 GPU:
* Môi trường phần cứng có nhiều GPU
* **Giao tiếp giữa slave và máy người dùng**

**Nhận xét**

So với hệ thống một host, thì hệ thống cluster này có nhiều điểm ưu việt hơn vì nó có nhiều slave, cho phép điều tiết: tăng hoặc giảm số lượng host để đáp ứng khi số lượng camera cũng như người dùng tăng cao hoặc giảm đi.

**3.2) Phân tích thiết kế**

**3.2.1) Kịch bản xử lý trên các môi trường phần cứng**

**Môi trường phần cứng chí có CPU**

Một client yêu cầu tới Master

If (NoCoreRest)

//process

**Môi trường phần cứng chỉ có một GPU**

**….**

**Môi trường phần cứng có nhiều GPU**

**….**

**3.3) Môi trường cài đặt và thử nghiệm**

- Hệ điều hành

- IDE

- Chọn kiểu Cluster/Grid

- Chọn Multi/single GPU

**3.4) Framework và ngôn ngữ LTr sử dụng**

- Framework và công dụng

+ CUDA : ...

...

- Ngôn ngữ Ltr và công dụng

+ C++ : chạy ở đâu, công đoạn nào

P/S : xem xét ý 3 và 4

#########################

**Chương 4 : Đánh giá thử nghiệm và kết luận**

1) Đánh giá

- Ưu điểm của ứng dụng

- Khuyết điểm của ứng dụng

2) Tiến độ

- Mục tiêu đã hoàn thành

- Mục tiêu chưa hoàn thành

3) Chiến lược phát triển - mở rộng

- Phát triển trên grid

- Âm thanh + hình ảnh

4) Kết luận

#########################

THAM KHẢO