**Chương 1 : Đặt vấn đề và giới thiệu đề tài**

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

- Trong thời đại Internet, các máy tính, thiết bị di động, điện thoại ngày một mạnh hơn dẫn đến nhu cầu xem video trực tuyến, video call, hội nghị trực tuyến qua Internet… ngày một cao. Để đáp ứng được điều đó, phía server phải có khả năng xử lý mạnh, phục vụ cùng 1 lúc hàng nghìn người.

- Việc ứng dụng CNTT vào các lĩnh vực Khoa học-Công nghệ, Hàng không, Kinh tế, Tài chính, Thương mại, Y tế, Quân sự… ngày càng phổ cập, sâu rộng và mọi lúc mọi nơi dẫn đến cần một hạ tầng tính toán có hiệu năng cao.

**1.2) Môi trường tính toán – Phần cứng**

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

**+ Host mạnh**:

* Có nghĩa là CPU phải mạnh (đa nhân,tốc độ cao, tiêu thụ năng lượng thấp), RAM lớn với độ trễ thấp, Bus nhanh, có GPU mạnh (đa nhân, tốc độ khá cao, tiêu thụ năng lượng ít) và card mạng tốc độ cao.
* Ưu điểm của cách tiếp cận này là: Phù hợp với trào lưu chất lượng phần cứng ngày càng 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.
* Tuy nhiên có một yếu điểm là một số ứng dụng cần HPC có thể xử lý trên 1 máy nhưng giá thành cao, chưa chắc đã tối ưu, khi nó vẫn có thể giải quyết trên nhiều máy nhỏ hơn và chi phí thấp hơn. Đó là nguồn gốc cho sự ra đời của các mô hình HPC cluster/grid.

**+ Cluster**

* Gồm nhiều máy tính trong đó 1 máy tính (gọi là Master) chịu trách nhiệm phân công việc cho các máy tính còn lại (gọi là slaves).
* Điểm yếu của mô hình này là khi Master có sự cố thì nguyên cả cluster ngưng hoạt động nếu không có back-up host. Đây cũng là điểm đưa đến việc grid ra đời.

**+ Grid**

* Gồm nhiều máy tính, mỗi máy tính vừa client, vừa là server cho các máy tính khác trong việc cùng giải quyết 1 nhiệm vụ nào đó.
* Điểm yếu quan trọng nhất là khó quản trị hệ thống hơn, đặc biệt trong môi trường biến động.

**- Kết luận** : Cần tận dụng đồng thời thế mạnh của các host ngày nay và thế mạnh của một cluster/grid đã thành công trong quá khứ. Cụ thể hơn là tận dụng được các điểm mạnh của:

* Một mạng LAN tốc độ cao gồm nhiều host (multi-hosts)
* Trong đó có nhiều host mạnh (multi-CPU-cores) và
* Có thể có card GPU (multi-GPU-cores)

Đây được xem là môi trường tính toán không đồng nhất (heterogenous computing enviroment).

**1.3) Môi trường lập trình – Phần mềm**

- Để thực hiện High-performance computing trên môi trường tính toán không đồng nhất nói trên, dẫn đến việc lập trình một ứng dụng cho phép thực hiện trên 3 môi trường (riêng rẽ hay đồng thời) sau:

+ Multi-hosts: Ứng dụng được xây dựng gồm nhiều tiến trình. Một (hay một số) tiến trình chạy trên một host. Các tiến trình cùng được thực hiện để cùng hoàn thành nhiệm vụ chung được giao. Trong quá trình thực hiện chúng có thể trao đổi với nhau. Một trong những cơ chế trao đổi thông tin/dữ liệu giữa các tiến trình trên các hosts là Message Passing Interface. Cơ chế này cho phép trao đổi theo kiểu point-to-point hay point-to-multipoint (collective) giữa các tiến trình, về bản chất là dựa trên sockets và TCP programming.

+ Multi-CPU-cores: Trong môi trường này, một tiến trình chính có thể tạo ra nhiều tiểu trình (Multithread programming) cùng phối hợp để cùng thực hiện nhiệm vụ được giao. Các tiểu trình này sẽ được hệ điều hành luân phiên hay đồng thời cho thực hiện trên các CPU cores theo cơ chế multi-core time sharing.

+ Multi-GPU-cores: Trong môi trường này, một tiến trình hay tiểu trình đang được thực thi trên một CPU core có thể yêu cầu tất cả các GPU cores đồng thời thực hiện một tiểu trình đặc biệt (gọi là kernel). Thông thường kernel có cùng một code, nhưng khi thực thi (kernel instances) lại dùng các vùng dữ liệu khác nhau. Đây chính là cơ chế cho phép chúng cùng thực thi một công việc.

**1.4) Kết luận**

- Các ứng dụng HPC hiện nay mới khai thác chủ yếu trên 1 hoặc đồng thời 2 môi trường, chưa khai thác trên đồng thời 3 môi trường, do đó cần thử nghiệm lập trình một ứng dụng HPC chạy đồng thời trên 3 môi trường tính toán không đồng nhất nói trên.

- Việc theo dõi bằng camera đã trở nên phổ biến, tuy nhiên thường với số lượng camera hạn chế. Trong một số trường hợp (như đã nói ở mục ), số lượng camera có thể lên tới nhiều trăm cái. Lúc đó, một server có thể không đủ mạnh (hoặc sẽ là rất đắt tiền) để xử lý tất cả các luồng video từ các IP cameras đổ về. Việc dùng một cluster máy tính giá rẽ có trang bị GPU cards sẽ là một lựa chọn kinh tế cho dự án này.

- Từ 2 nhận xét trên, đề tài “ ” được chọn làm đề tài của luận văn này.

**Chương 2 : Các môi trường lập trình và tính toán song song**

**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**

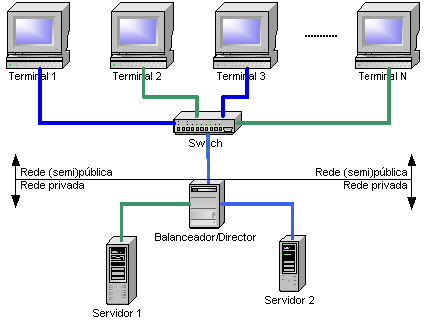
- Là môi trường lập trình song song trên nhiều host. Gridware ngoài việc bao gồm môi trường lập trình (edit, debug) như MPI thì nó còn cung cấp môi trường thực thi song song (deploy, quản trị, khai thác tài nguyên và ứng dụng), được ứng dụng trong tính toán hiệu năng cao.

**-** 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**

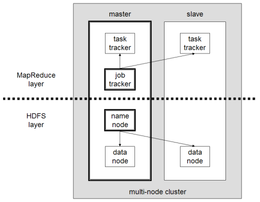
**Cluster** : hệ thống máy tính kết nối với nhau, thường thông qua mạng LAN, cùng xử lý các tác vụ liên quan đến ứng dụng.

* ***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)

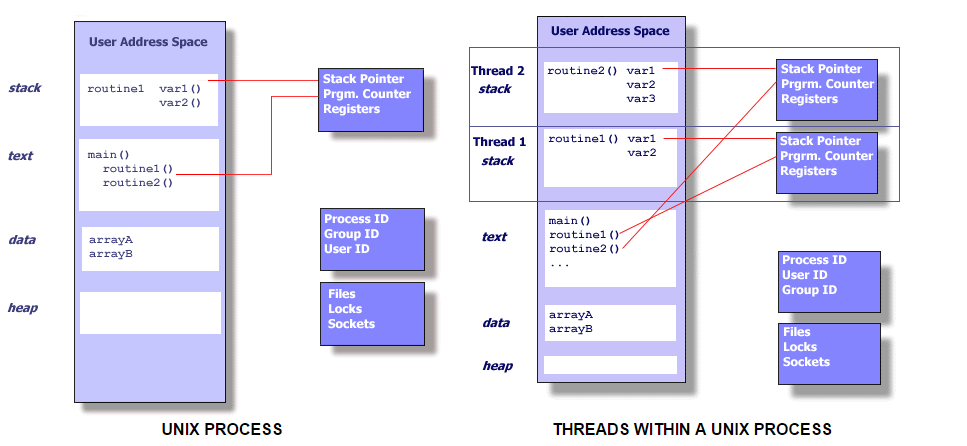
**Grid** : là hệ thống các máy tính kết nối với nhau để thực hiện các ứng dụng chung, nhưng khác với cluster mỗi máy tính trong hệ thống là 1 master (đồng thời nó cũng là slave cho các master khác). Grid chính là một hệ thống tính toán phân tán mà trong đó tài nguyên của một máy được chia sẻ với các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

**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)**

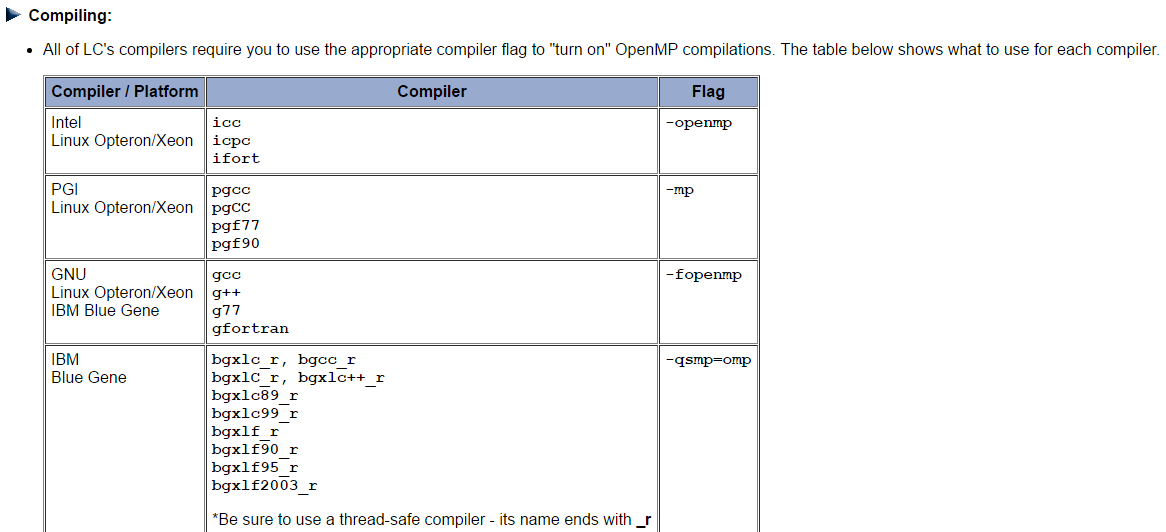
- Là việcchia sẻ thời gian sữ dụng của một hoặc nhiều core (trên cùng một host) giữa các tiến trình /tiểu trình. Việc chia sẻ này được thực hiện ở hai cấp độ: thực thi ở mức hệ điều hành (multitasking) và ở mức độ lập trình (multiprogramming).

**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)*

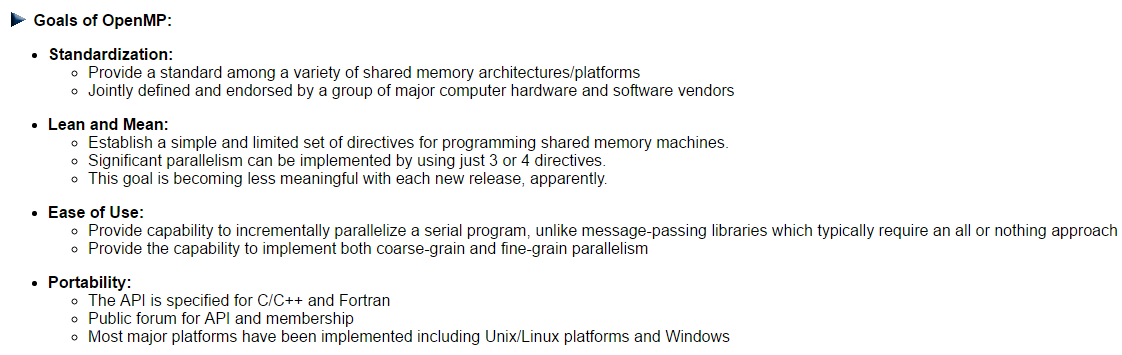
* Mục đích sử dụng :
* Cài đặt và truy vấn đến một lượng các tiểu trình : ID, ID cha, kích thước.
* Cài đặt và quản lý các khóa của tiểu trình
* Truy vấn thời gian và giải pháp
* Environment Variables
* Kiểm soát việc chạy các đoạn mã nguồn song song trong suốt quá trình chạy của chương trình

*(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 song song trên nhiều GPU cores**

**2.3.1) GPU**

**Giới thiệu :**

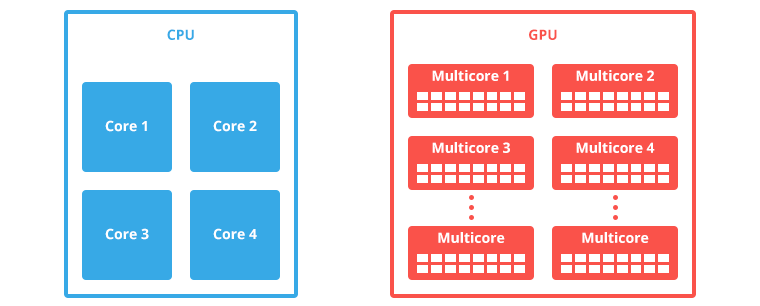
- GPU (Graphics Processing Unit) là phần cứng chuyên dụng được thiết kế để xử lý về hình ảnh, video trong máy tính.

- 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.

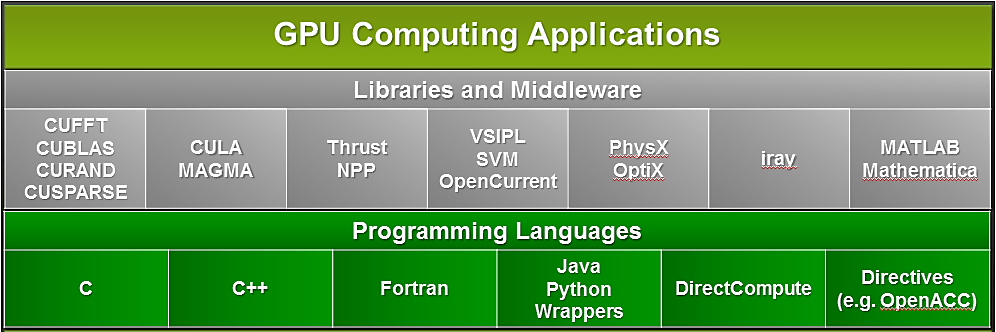
**Kiến trúc :**

* 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:**

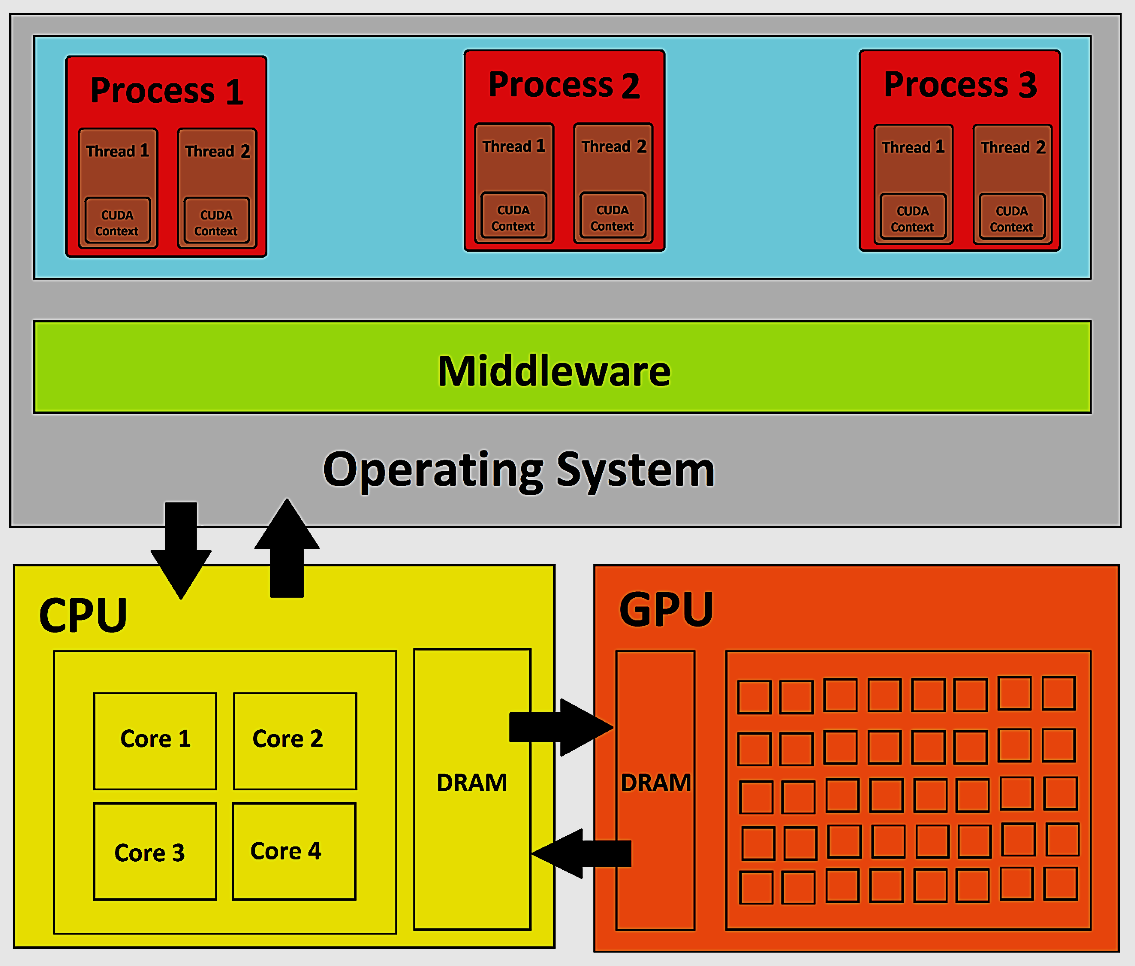


***http://strongartificialintelligence.com/projects***

* 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)…

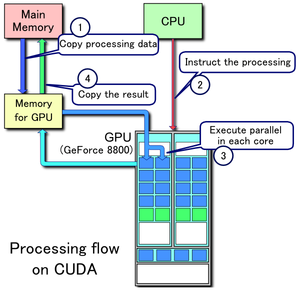
**2.3.2) CUDA**

**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 :**

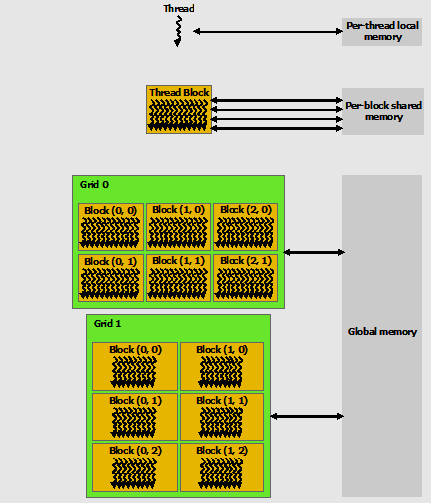
* **Mô hình xử lý tổng quát** :



*(CUDA: wikipedia)*

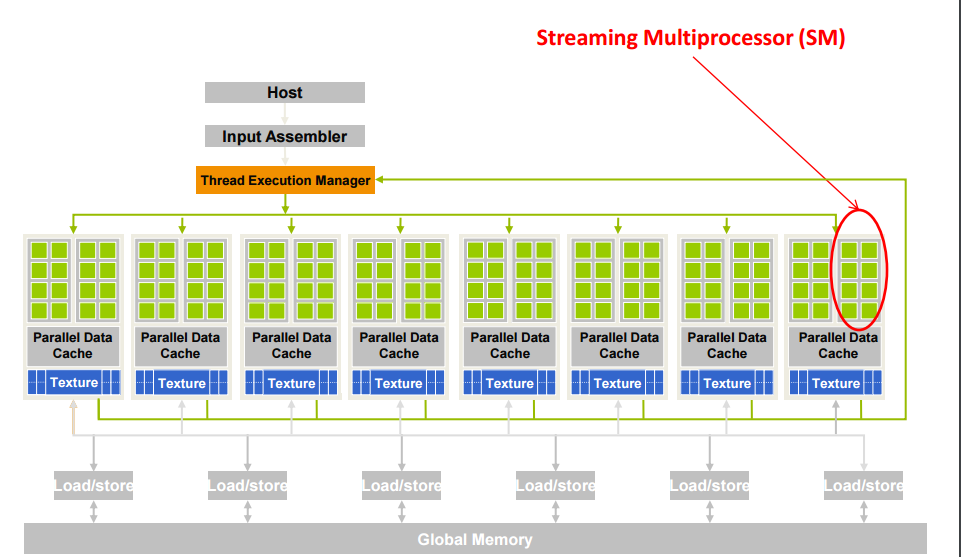
1. Đưa dữ liệu (kernel và data của nó) từ bộ nhớ chính (Host memory) vào vùng nhớ của GPU (device memory).
2. CPU ra lệnh GPU xử lý
3. Kernel được thực thi tại các core của GPU (kết quả trả về device memory).
4. Copy kết quả từ device memory về lại bộ nhớ chính (Host memory).

* **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).

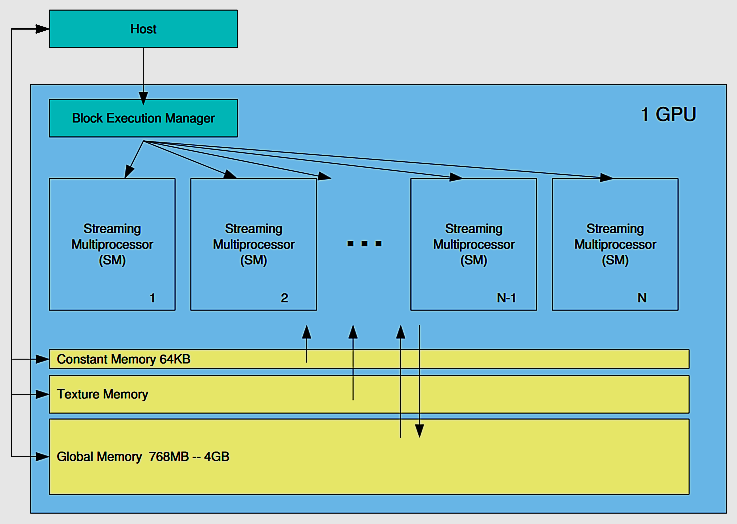
Mỗi block có một vùng nhớ chung (per-block shared memory) để các tiểu trình chia sẻ. Dĩ nhiên tiểu trình ở block này không thể sử dụng tài nguyên ở block khác. Tuy nhiên tất cả các block đều có thể sử dụng Global Memory – một vùng nhớ toàn cục của GPU.

*(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

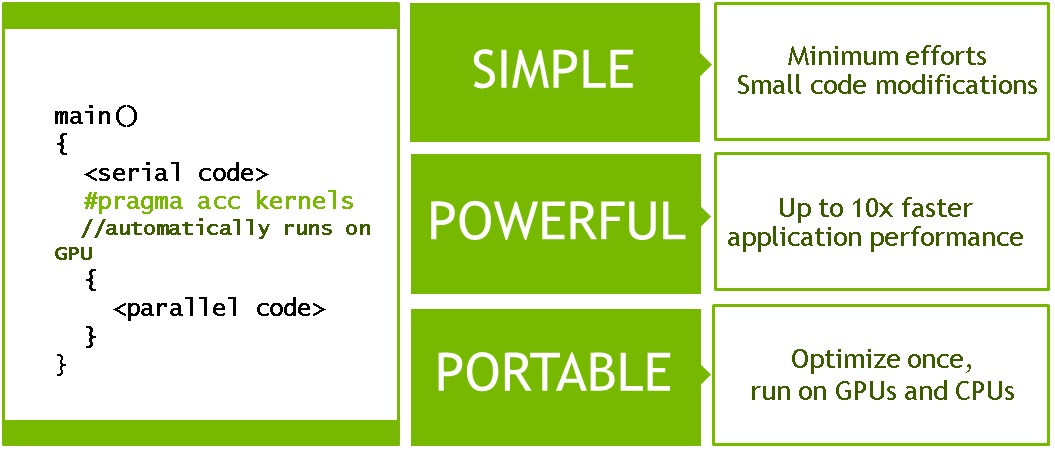
**2.3.3) OpenACC**

**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.

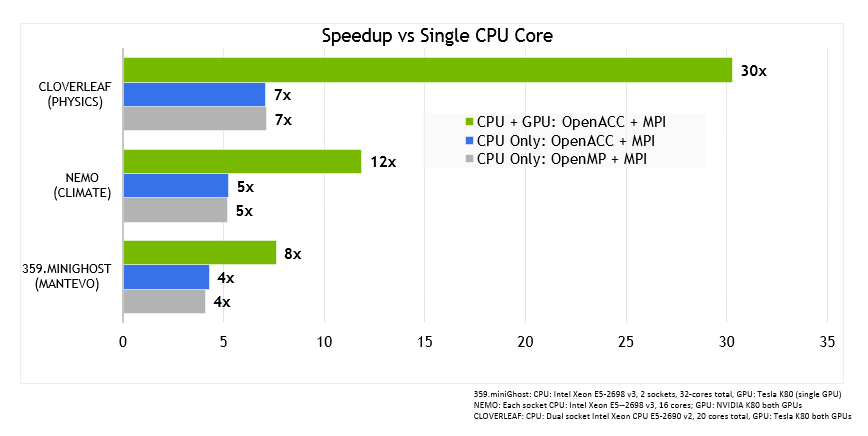
**Công dụng :**



*(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**

Ngày nay nhu cầu quan sát/theo dõi bằng camera từ xa đang trở nên phổ biến hơn đối với các dịch vụ an ninh, trình diễn sản phẩm, hội nghị, tư vấn, đào tạo từ xa… Trong các hệ thống đơn giản, các luồng video từ các camera được đưa vào hệ thống và được xử lý tập trung thành một luồng duy nhất (gồm nhiều khung hình) trên một server và người dùng được cung cấp luồng video này khi đăng nhập vào. Tuy nhiên trên thực tế, những đối tượng cần quan sát rất phức tạp, có thể gồm nhiều tòa nhà hay khu vực lớn như cơ quan, chung cư, hay bến xe, bến cảng, sân bay, siêu thị lớn… Trong các 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 bến cảng, sân bay, siêu thị: Ngoài các nhóm camera trên, 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/dịch vụ

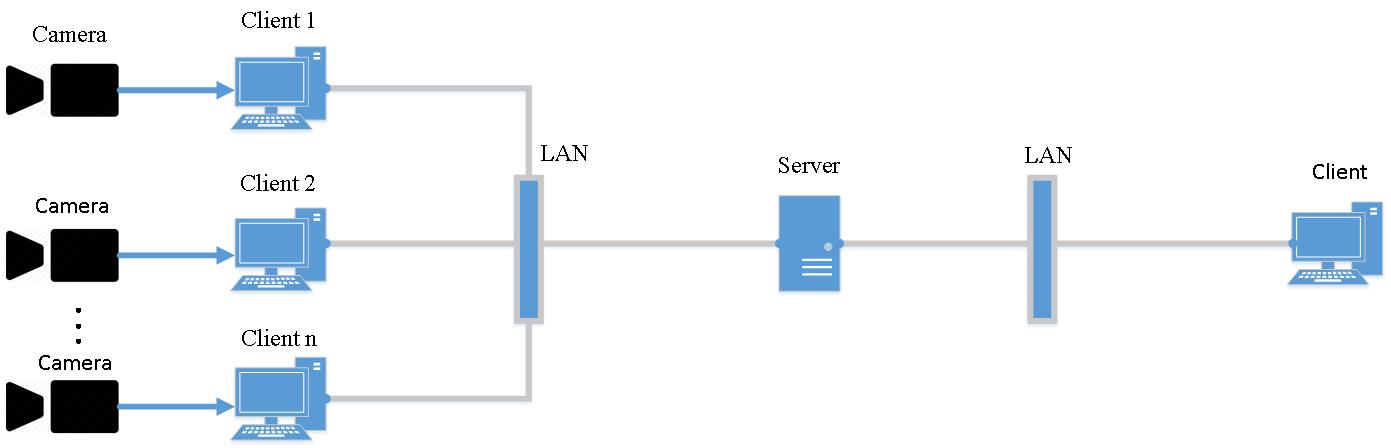
Ngoài ra, tùy theo sự kiện nào đó như hội nghị, trình diễn, giới thiệu hàng hóa/sản phẩm mới… mà một số lượng camera sẽ có thể thêm vào hoặc bớt đi. Hệ thống cũng cần tính tới khả năng số lượng người dùng truy cập cũng sẽ thay đổi theo thời gian và theo sự kiện.

Vì vậy, việc xây dựng một hệ thống có khả năng đáp ứng được yêu cầu phức tạp trên, đồng thời có khả năng tiết giảm được kinh phí đầu tư thiết bị và băng thông là một yêu cầu cấp thiết.

**3.2 Giải pháp 1 server**

Luận văn năm 2016 của Trần Văn Hoàng và Đào Tặng Thưởng [1] đã đề cập tới vấn đề ~~này~~ trên và đưa ra phương án giải quyết : hệ thống một Host (server).

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.

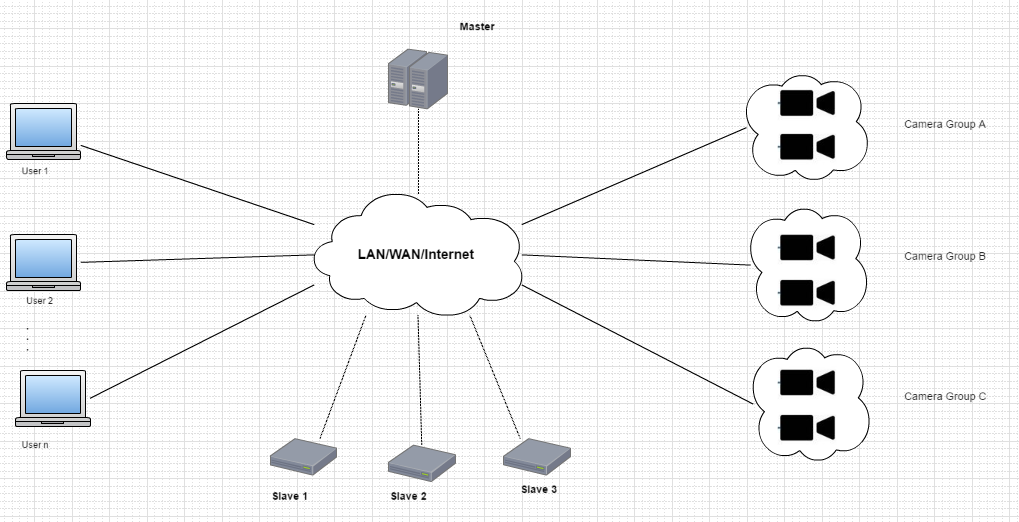


Tuy nhiên hệ thống [1] chỉ đáp ứng với số lượng camera không quá lớn vì nó chỉ có một server (chưa bàn đến số lượng người dùng khai thác hệ thống đồng thời). Do đó nhiệm vụ của luận văn này là phải mở rộng được ra nhiều host, nhằm tăng số lượng camera vói chi phí thấp nhất có thể.

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.

**3.3) Mô hình mở rộng được đề nghị**



Trên cơ sở tiếp nối ứng dụng luận văn [1] một cluster có quy mô lớn hơn, gồm nhiều host thay vì một host như trước kia được phát triễn.

Trong ứng dụng này, các camera sẽ được phân thành nhiều nhóm theo dõi các khu vực khác nhau, nhằm vừa đáp ứng yêu cầu của người dùng khi họ muốn theo dõi một khu vực nhất định nào đó, nhưng vẫn đảm bảo tính riêng tư hay tính giới hạn ở những phần còn lại.

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

* Master host: chịu trách nhiệm quản lý các slave hosts (servers), camera và nhóm camera, người dùng và nhóm người dùng. Cụ thể như:
  + Quản lý các slave host thông qua ip và port, liên tục cập nhật tình trạng của các nhóm camera/người dùng do slave phụ trách
  + Cho phép các camera đăng nhập/đăng xuất hệ thống và các nhóm camera;
  + Tương tự đối với người dùng;
  + Ánh xạ nhóm người dùng và nhóm camera, cho phép họ có quyền quan sát/theo dõi;
  + Phân phối nhóm camera cho slave xử lý;
  + Phân phối lại nhóm camera cho slave khác khi slave phụ trách gặp sự cố.
* Slave host:
  + Tiếp nhận nhóm camera, nhóm người dùng do Master giao slave phụ trách
  + Chịu trách nhiệm nhận các luồng video từ các camera (trong cùng một nhóm) gởi đến,
  + Giải mã chúng, xử lý tích hợp các khung hình tạo thành 1 luồng video mới
  + Mã hóa luồng video mới này trước khi chuyển giao cho nhóm người dùng đã đăng ký trước.

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

* Chức năng 0 (*quản lý slave hosts : đăng nhập, quản lý hiện trạng của slave và đăng xuất*): Chức năng này cho phép Admin đưa một slave host vào hệ thống, theo dõi nó suốt quá trình hoạt động và rút khỏi hệ thống khi cần. Trước mắt chức năng này chưa được cài đặt nên Admin phải tự cấu hình tay.
* Chức năng 1 (*quản lý camera : đăng nhập, phân phối đến slave và đăng xuất*): Admin hệ thống sẽ đăng nhập Camera vào hệ thống thông qua Master (camera phải được gia nhập vào một nhóm). Master sẽ cung cấp cho camera service: 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 rút ra khỏi nhóm, thì trước tiên phải thực hiện thủ tục đăng xuất với Master. Nếu Camera đổi khu vực, thì phải đăng xuất khỏi Master và đăng nhập lại từ đầu thông qua Master.
* Chức năng 2 (*quản lý user: đăng nhập, phân phối đến slave và đăng xuất*): Người dùng đăng nhập vào hệ thống để yêu cầu quan sát một khu vực nào đó (do 1 nhóm camera cung cấp hình ảnh) thì người dùng sẽ gửi yêu cầu đăng nhập tới Master; Master sẽ cung cấp port, địa chỉ IP của slave đảm nhận khu vực (nhóm camera) 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 ngưng theo dõi, thì chỉ cần phải thực hiện thủ tục đăng xuất với Master.
* Chức năng 3 (*Giao tiếp giữa Master và Slave*): Master có nhiệm vụ phân phối một nhóm camera cho một slave host còn năng lực xử lý. Mỗi slave host sẽ đảm nhận xử lý dữ liệu hình ảnh từ một hay nhiều camera thuộc một khu vực nào đó. Mạng kết nối giữa các slave host và Master thường là cục bộ (LAN), nhưng cũng có thể là WAN, thậm chí qua Internet (trong phiên bản này là qua LAN). Master còn có nhiệm vụ phân phối lại công việc trong trường hợp đặc biệt như: slave host bị hư, thêm/bớt slave host… Hơn nữa mỗi slave host 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 host và camera service*) : slave host nhận trực tiếp các luồng video từ nhiều camera service (trong cùng một nhóm/khu vực) và nén chúng lại ngay sau khi camera thực hiện đăng nhập với Master và được Master gán cho slave host đó. 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 host chịu trách nhiệm phân phối luồng video đã nén cho người dùng thông qua mạng LAN/WAN hay Internet, ngay sau khi user đăng nhập lần đầu tiên vào hệ thống và vào nhóm người dùng tương ứng. Lúc này máy tính user sẽ giao tiếp trực tiếp với slave host liên quan để xem hình ảnh tổng hợp từ các camera cùng nhóm mà không phải qua trung gian Master nữa. Ngoài ra trong chức năng này, người dùng có thể xem chi tiết từng khung hình của luồng video nào đó tùy theo ý thích của mình (có thể được cài đặt tùy theo thời gian cho phép hay không).

**3.4) Kịch bản cho mỗi chức năng**

* **Chức năng 1: Quản lý camera-đăng nhập, phân phối đến slave và đăng xuất**

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Camera service** | **Master** | **Slave** |
| 1 | Gửi yêu cầu đăng nhập vào hệ thống cho Master (\*1). Yêu cầu gồm có tên, IP và port của camera. | Nhận yêu cầu đăng nhập vào camera service |  |
| 2 |  | Tìm kiếm slave phụ trách nhóm mà camera thuộc về dựa trên thông tin cập nhật thường xuyên của “lịch phân công” hay còn gọi là “bảng tình trạng”. |  |
| 3 |  | Gửi mệnh lệnh yêu cầu slave đảm nhận camera theo IP, port của slave. Yêu cầu gồm có tên, IP và port của camera. | Nhận và phân tích mệnh lệnh từ Master |
| 4 |  | Nhận trả lời từ slave  Cập nhật thông tin vào bảng tình trạng. | Gửi trả lời “chấp nhận” |
| 5 | Nhận trả lời “đồng ý đăng nhập” từ Master | Gửi trả lời cho camera có nội dung cho phép Camera service đăng nhập vào hệ thống. Message gồm có IP, port, và tên của Slave sẽ phụ trách. |  |
| 6 | **Kịch bản 4: Giao tiếp giữa slave host và camera service** | | |
| 7 | Gửi yêu cầu đăng xuất khỏi hệ thống tới Master | Nhận yêu cầu đăng xuất từ Camera service |  |
| 8 |  | Gửi đến Slave yêu cầu “ngắt kết nối với camera service này” | Nhận yêu cầu ngắt kết nối của camera từ master |
| 9 |  |  | Ngưng đọc hình ảnh từ camera. |
| 10 |  |  | Giải phóng tài nguyên dành cho camera đó và đóng kết nối với port và IP của camera đó. |
| 11 |  | Nhận trả lời từ Slave, sau đó cập nhật bảng tình trạng. | Gửi trả lời “cho phép đăng xuất” đến Master. |
| 12 | Nhận trả lời từ Master. | Gửi trả lời “cho phép đăng xuất” tới camera |  |
| 13 | Đóng kết nối với Port và IP của slave. |  |  |

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.

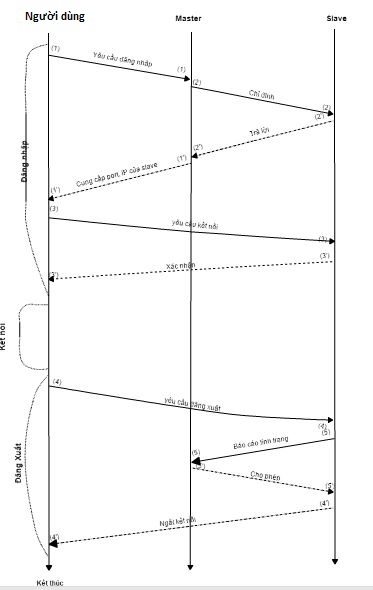
(3’) Slave gửi trả lại xác nhận kết nối : thành công hay không. Nếu thành công, cả hai bên bắt đầu thực hiện kết nối và slave sẽ capture hình trực tiếp từ camera.

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.

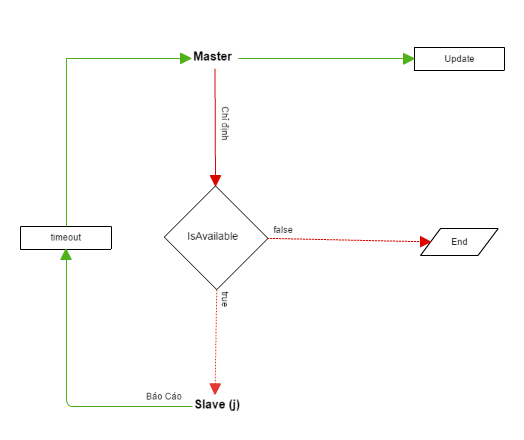
(5’) Master gửi xác nhận cho phép người dùng đăng xuất khỏi hệ thống đến lại cho slave.

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

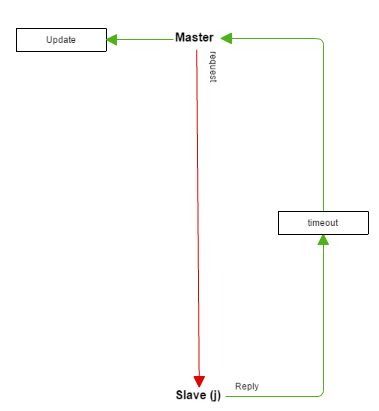
* **Giao tiếp giữa Master và slave**

**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**



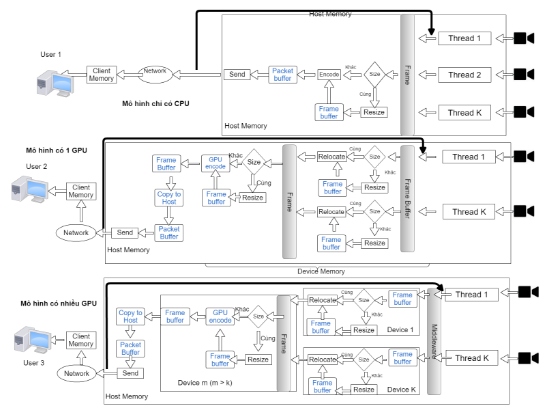
Master sẽ gửi cho Slave một yêu cầu request tới Slave nghi bị sự cố và đợi phản hồi từ slave đó trong khỏang thời gian timeout.

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.

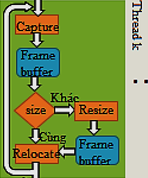
Reply từ slave chính là các thông tin về tình trạng của nó.

* **Giao tiếp giữa slave và camera**



**Cơ chế xử lý ảnh trong môi trường phần cứng chỉ có CPU**

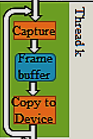
* Mô tả công việc của tiểu trình: Host tạo ra các tiểu trình tương ứng với mỗi camera để capture hình ảnh thu được vào Buffer. Nếu ảnh khác size với tất cả các ảnh khác ở các tiểu trình kia thì sẽ được resize lại cho cùng kích thước rồi đưa vào một frame buffer trước khi được relocate vào Main Frame chính.



* Kế tiếp, nếu điều kiện cùng kích thước được thỏa mãn thì frame\_main sẽ được Resize về kích thước chuẩn trước khi được Encode bằng thuật toán nén ảnh H.264 để giảm dung lượng ảnh. Nếu đều kiện cùng kích thước không thỏa mãn thì frame\_main sẽ thực hiện encode mà không cần thực hiện Resize ở bước này.
* Cuối cùng thì frame\_main sẽ được nhận tại client memory, được thực hiện decode sau đó sẽ được trình chiếu lên cho người dùng xem.

**Cơ chế xử lý ảnh trong môi trường phần cứng có 1 GPU**

* Mô tả công việc của tiểu trình: Host tao ra các tiểu trình capture ảnh từ camera tương ứng vào buffer, sau đó được copy vào Device memory.



* Sau đó Device nhận các frame ảnh từ tiểu trình rồi kiểm tra điều kiện về kích thước của frame so với các frame khác. Nếu chúng cùng kích thước thì sẽ được relocate trực tiếp vào frame\_main. Nếu không thì phải thực hiện resize về cùng một kích thước vào các frame buffer trước khi được relocate vào frame\_main.
* Kế tiếp, nếu điều kiện cùng kích thước của các frame trong frame\_main được thỏa mãn thì chúng sẽ được resize về kích thước chuẩn trước khi thực hiện thuật toán nén ảnh H.264 rồi encode trên GPU ngay sau đó. Nếu điều kiện cùng kích thước không thỏa mãn thì không cần phải qua bước resize trước khi được encode trên GPU. Ảnh sau khi encode được chép vào lại Host memory trước khi gửi đến client
* Cuối cùng, frame\_main được nhận tại client memory, được giải mã rồi trình chiếu lên cho người dùng xem

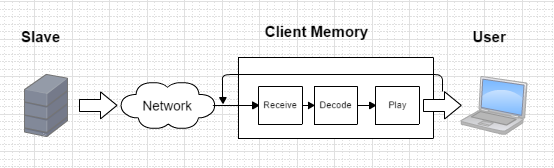
**Cơ chế xử lý ảnh trong môi trường phần cứng có nhiều GPU**

* Mô tả công việc của tiểu trình: Host tạo ra các tiểu trình tương ứng với mỗi camera để capture ảnh từ camera vào buffer. Ảnh từ buffer được chuyển đến các Middleware trước khi phân phối đến các Device trong GPU.



* Sau đó, mỗi Device sẽ lấy ảnh từ Middleware vào các Frame buffer. Device kiểm tra điều kiện so sánh kích thước frame ảnh này với kích thước chuẩn. Nếu khác thì sẽ phải thực hiện resize trước rồi sau đó mới relocate vào frame ảnh có kích thước chuẩn (frame\_main). Còn nếu cùng thì được relocate ngay, không cần phải resize.
* Kế tiếp, nếu điều kiện cùng kích thước của các frame được thỏa mãn thì chúng sẽ được Device resize về kích thước chuẩn trước khi thực hiện thuật toán nén ảnh H.264 rồi encode trên GPU ngay sau đó. Nếu điều kiện cùng kích thước không thỏa mãn thì không cần phải qua bước resize trước khi được encode trên GPU. Ảnh sau khi encode được chép vào lại Host memory trước khi gửi đến client
* Cuối cùng, frame\_main được nhận tại client memory, được giải mã rồi trình chiếu lên cho người dùng xem.
* **Giao tiếp giữa slave và máy người dùng**

**Client Memory:**



* Hình ảnh được gửi theo trình tự:

Receive : Nhận hình ảnh đã nén và gửi từ slave thông qua LAN/WAN (Network)

Decode: Giải mã

Play: trình chiếu lên máy tính của người dùng

* Quá trình xử lý trong Client Memory này được diễn ra theo vòng lặp.

**3.1.6) 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.

**3.5) Phân tích thiết kế**

Sau đây là mã giả xử lý trên các môi trường phần cứng cho các kịch bản trên.

**3.5.1) Môi trường phần cứng chỉ có CPU**

- Mã giả và CTDL cho Master:

- Mã giả và CTDL cho slave: Chủ yếu là mã giả từ [1]

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

- Mã giả và CTDL cho Master:

- Mã giả và CTDL cho slave: Chủ yếu là mã giả từ [1]

**3.5.3) Môi trường phần cứng có nhiều GPU**

**-** Do thiếu thiết bị nên mã giả cho môi trường này không được triễn khai

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

- Chọn hệ điều hành và midleware cho Slave và Master (clustware)

- Chọn môi trường lập trình (IDE)

- Chọn kiểu giữa Cluster/Grid và Multi/single GPU

- Mô hình thử nghiệm

- Framework và công dụng

+ CUDA : ...

...

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

**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

[1] Luận văn năm 2016 của Trần Văn Hoàng và Đào Tặng Thưởng – khoa Công nghệ Thông tin Đại học Khoa học Tự nhiên TPHCM