BỘ GIÁO DỤC VÀ ĐÀO TẠO

VIỆN HÀN LÂM KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

Nguyễn Thị Hoan

NGHIÊN CỨU GIẢI PHÁP QUẢN LÝ TÀI NGUYÊN HỆ THỐNG TÍNH TOÁN HIỆU NĂNG CAO DỰA TRÊN MÔI TRƯỜNG MÃ NGUỒN MỞ

LUẬN VĂN THẠC SĨ NGÀNH: MÁY TÍNH

Hà Nội - Năm 2023

BỘ GIÁO DỤC VÀ ĐÀO TẠO

VIỆN HÀN LÂM KHOA HỌC VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

Nguyễn Thị Hoan

NGHIÊN CỨU GIẢI PHÁP QUẢN LÝ TÀI NGUYÊN HỆ THỐNG TÍNH TOÁN HIỆU NĂNG CAO DỰA TRÊN MÔI TRƯỜNG MÃ NGUỒN MỞ

Chuyên ngành: Hệ thống thông tin
Mã số: 8480104

LUẬN VĂN THẠC SĨ NGÀNH: MÁY TÍNH

NGƯỜI HƯỚNG DẪN KHOA HỌC:

TS. Ngô Hải Anh

Hà Nội - Năm 2023

LỜI CAM ĐOAN

Tôi xin cam đoan đề tài nghiên cứu trong luận văn này là công trình nghiên cứu của tôi dựa trên những tài liệu, số liệu do chính tôi tự tìm hiểu và nghiên cứu. Chính vì vậy, các kết quả nghiên cứu đảm bảo trung thực và khách quan nhất. Đồng thời, kết quả này chưa từng xuất hiện trong bất cứ một nghiên cứu nào. Các số liệu, kết quả nêu trong luận văn là trung thực nếu sai tôi hoàn chịu trách nhiệm.

Tác giả luận văn

Nguyễn Thị Hoan

LỜI CẢM ƠN

Tôi xin chân thành cảm ơn Khoa Công nghệ thông tin và Viễn thông – Học Viện Khoa học và Công nghệ đã tạo điều kiện thuận lợi cho tôi thực hiện đề tài tốt nghiệp này.

Đặc biệt, xin bày tỏ lòng biết ơn sâu sắc đến thầy TS. Ngô Hải Anh đã trực tiếp hướng dẫn và giúp đỡ tôi trong suốt quá trình thực hiện luận văn.

Cảm ơn quý thầy cô Khoa trong Công nghệ thông tin và Viễn thông, cùng với các thầy cô, ban Lãnh đạo, phòng Đào tạo, các phòng chức năng của Học viện Khoa học và Công nghệ tận tình giảng dạy và truyền đạt kiến thức suốt khóa học.

Sau cùng, tôi nói lời cảm ơn đến các anh chị em, bạn bè đồng nghiệp trong cùng gia đình đã giúp đỡ, đóng góp ý kiến để tôi hoàn thành luận văn này.

Mặc dù đã có nhiều cố gắng song trong phạm vi và khả năng cho phép chắc hẳn sẽ khó tránh khỏi những thiếu sót. Mong tiếp tục nhận được sự cảm thông, góp ý để xây dựng đề tài hoàn thiện hơn.

Trân trọng cảm ơn!

Tác giả luận văn

Nguyễn Thị Hoan

MỤC LỤC

DANH MỤC CÁC KÝ HIỆU, CÁC CHỮ CÁI VIẾT TẮT

STT	Từ viết tắt	Từ tiếng Anh	Diễn giải/tạm dịch
1.	HPC	High-Performance Computing	Tính toán hiệu năng cao
2.	OS	Operating System	Hệ điều hành
3.	DC	Data Center	Trung tâm dữ liệu
4.	CPU	Central Processing Unit	Bộ điều khiển trung tâm
5.	HPLinpack	High Performance Linpack	Chương trình Linpack cho hệ thống hiệu năng cao
6.	CAD	Computer Aided Design	Nền tảng phần mềm hỗ trợ thiết kế
7.	CAM	Computer Aided Manufacturing	Nền tảng phần mềm hỗ trợ sản xuất
8.	MPI	Message Passing Interface	Giao diện chuyển tiếp thông tin
9.	Eth	Ethernet	Một tập hợp các giao thức để kết nối các thiết bị trong mạng có dây
10.	NFS	Network File System	Hệ thống tập tin mạng
11.	NIS	Network Information Service	Dịch vụ thông tin mạng
12.	NTP	Network Time Protocol	Giao thức thời gian mạng
13.	DHCP	Dynamic Host Configuration Protocol	Giao thức cấu hình địa chỉ động
14.	SSH	Secure Socket Shell	Giao thức kết nối điều khiển bảo mật
15.	HTTP	Hyper Text Transfer Protocol	Giao thức truyền siêu văn bản

16.	NAT	Network Address Translation	Dịch địa chỉ mạng
17.	BIOS	Basic Input/Output System	Hệ thống xuất nhập cơ bản
18.	Ib	InfiniBand	Chuẩn giao tiếp truyền dẫn dữ liệu hiệu suất cao
19.	IPMI	Intelligent Platform Management Interface	Giao diện quản lý nền tảng thông minh
20.	RAID	Redundant Array of Independent Disks	Cách thức gộp các ổ đĩa thành một khối có dự phòng

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

MỞ ĐẦU

1. Lý do chọn đề tài

1.1 Tính cấp thiết

Ngày nay, tính toán khoa học đang trở nên ngày càng quan trọng trong nhiều lĩnh vực nghiên cứu. Các vấn đề dần trở nên phức tạp hơn, đòi hỏi sự phối hợp của đội ngũ các nhà nghiên cứu thuộc nhiều lĩnh vực chuyên môn khác nhau. Trong ngành tính toán khoa học, tính toán hiệu năng cao (HPC - viết tắt của cụm từ High-Performance Computing) đóng vai trò cốt lõi, bổ sung các phương pháp nghiên cứu khoa học công nghệ truyền thống (tức là viết một lý thuyết hoặc thiết kế trên giấy, sau đó tiến hành thử nghiệm hoặc xây dựng hệ thống) bằng cách thực hiện các tính toán số học trên các hiện tượng và/hoặc các thử nghiệm mà có thể quá khó (ví dụ xây dựng một đường hầm thông gió lớn), quá đắt (ví dụ tạo một máy bay phản lực chở khách để thử nghiệm va chạm), quá chậm (ví dụ nghiên cứu sự biến đổi khí hậu hoặc sự tiến hóa của dải ngân hà), quá nguy hiểm (ví dụ thử nghiệm vũ khí hạt nhân) hoặc gây tranh cãi (ví dụ nghiên cứu tế bào gốc). Tính toán hiệu năng cao tạo điều kiện thuận lợi cho các nhà nghiên cứu, những người muốn thực hiện một lượng rất lớn các phép tính lặp đi lặp lại trên một lượng lớn dữ liệu để đạt được các kết quả hợp lệ trong một khoảng thời gian hợp lý.

Tính toán hiệu năng cao đang được sử dụng bởi các nhà nghiên cứu trên toàn thế giới, từ các nhóm lớn đến các cá nhân, để bổ sung cho phương pháp nghiên cứu khoa học truyền thống. Với các phần cứng tính toán hiệu năng cao có giá thành hợp lý cùng một hệ điều hành tự do, các phần mềm mã nguồn mở như Linux, tính toán hiệu năng cao đang nằm trong tầm với của các cơ quan tổ chức có ngân sách nhỏ hẹp. Tuy nhiên để có thể triển khai được hiệu quả các lợi ích mà tính toán hiệu năng cao cung cấp, người ta cần có hiểu biết chi tiết về kiến trúc của hệ thống tính toán hiệu năng cao, cũng như làm chủ các công cụ phát triển và các kỹ thuật lập trình nâng cao đặc trưng của tính toán hiệu năng cao. Việc phát triển ứng dụng khoa học dựa trên tính toán hiệu năng cao thường bị hạn chế bởi việc thiếu nhân lực được đào tạo trong lĩnh vực này. Do đó, việc có thêm hiểu biết về tính toán hiệu năng cao sẽ giúp các nhà nghiên cứu sử dụng hiệu quả hơn các tài nguyên tính toán hiệu năng cao và tương tác tốt hơn với các đồng nghiệp.

Trên cơ sở đó tôi đã lựa chọn đề tài: "***Nghiên cứu giải pháp quản lý tài nguyên hệ thống tính toán hiệu năng cao dựa trên môi trường mã nguồn mở***". Đây là một hướng nghiên cứu khá thiết thực, rất đáng quan tâm trong bối cảnh hiện nay, khi mà cuộc cách mạng công nghiệp 4.0 đã và đang ảnh hưởng sâu rộng tới tất cả các lĩnh vực trong cuộc sống.

1.2 Tình hình nghiên cứu và những vấn đề đặt ra trong đề tài

- Tình hình nghiên cứu trên thế giới và ở Việt Nam

Những hệ thống HPC trên thế giới được phát triển để đáp ứng các yêu cầu nghiên cứu khoa học, mô phỏng, xây dựng mô hình, tính toán giả lập… Hiện nay các hệ thống HPC có thể nói là đã phổ biến trên thế giới, tuy nhiên thường chỉ tập trung tại các Data Center lớn hoặc Dịch vụ trên Cloud. Ở đó người ta xây dựng các hệ thống máy chủ tính toán và lưu trữ lớn, kết nối mạng với nhau để phục vụ công việc tính toán hiệu năng cao. Người ta cũng phát triển ra các phần mềm quản lý, cài đặt sẵn các thư viện tính toán khoa học để phục vụ người dùng với các loại hình bài toán khác nhau. Tuy nhiên, giá thành để thuê các mô hình như vậy khá đắt cũng như không phải mô hình nào cũng có được thư viện tính toán chúng ta cần. Việc thuê hệ thống sẽ thường được tính theo số lượng CPU là thời gian sử dụng, trong khi chúng ta khó có thể ước tính được chúng ta cần bao nhiêu CPU và bao nhiêu giờ để có thể hoàn thành xong bài toán. Do đó chi phí này hoàn toàn rất khó ước tính. [1]

Trong khi đó, thực tế cho thấy, tại các trường đại học, họ đang có rất nhiều máy tính cũng như máy chủ, họ có nhu cầu tận dụng năng lực tính toán của chúng để tránh lãng phí. Vậy nhu cầu tự xây dựng và vận hành một hệ thống tính toán hiệu năng cao để giảm thiểu chi phí trong nghiên cứu là rất thực tế.

Danh sách các hệ thống siêu máy tính phục vụ Tính toán hiệu năng cao hàng đầu trên thế giới (Chúng ta có thể tham khảo danh sách các hệ thống siêu máy tính hàng đầu trên thế giới phục vụ mục đích tính toán hiệu năng cao tại: https://www.top500.org/lists/top500/). Dưới đây là 10 hệ thống có năng lực xử lý cao nhất tại thời điểm đầu năm 2023:

Bảng 1.2. Hệ thống siêu máy tính mạnh nhất thế giới đầu năm 2023

STT	Tên hệ thống	Cores	Năng lực cực đại (PFLOPS)
1	Frontier - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE DOE/SC/Oak Ridge National Laboratory United States	8.699.904	1.194,00
2	Supercomputer Fugaku - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, Fujitsu RIKEN Center for Computational Science Japan	7.630.848	442,01
3	LUMI - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, HPE EuroHPC/CSC Finland	2.220.288	309,10
4	Leonardo - BullSequana XH2000, Xeon Platinum 8358 32C 2.6GHz, NVIDIA A100 SXM4 64 GB, Quad-rail NVIDIA HDR100 Infiniband, Atos EuroHPC/CINECA Italy	1.824.768	238,70
5	Summit - IBM Power System AC922, IBM POWER9 22C 3.07GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM DOE/SC/Oak Ridge National Laboratory United States	2.414.592	148,60
6	Sierra - IBM Power System AC922, IBM POWER9 22C 3.1GHz, NVIDIA Volta GV100, Dual-rail Mellanox EDR Infiniband, IBM / NVIDIA / Mellanox DOE/NNSA/LLNL United States	1.572.480	94,64
7	Sunway TaihuLight - Sunway MPP, Sunway SW26010 260C 1.45GHz, Sunway, NRCPC National Supercomputing Center in Wuxi China	10.649.600	93,01
8	Perlmutter - HPE Cray EX235n, AMD EPYC 7763 64C 2.45GHz, NVIDIA A100 SXM4 40 GB, Slingshot-10, HPE DOE/SC/LBNL/NERSC United States	761.856	70,87
9	Selene - NVIDIA DGX A100, AMD EPYC 7742 64C 2.25GHz, NVIDIA A100, Mellanox HDR Infiniband, Nvidia NVIDIA Corporation United States	555.520	63,46
10	Tianhe-2A - TH-IVB-FEP Cluster, Intel Xeon E5-2692v2 12C 2.2GHz, TH Express-2, Matrix-2000, NUDT National Super Computer Center in Guangzhou China	4.981.760	61,44

- Sự cần thiết phải tiến hành nghiên cứu

Nhu cầu trong việc tính toán, mô phỏng các mô hình, sự vật, hiện tượng trong cuộc sống đang ngày một lớn, nhất là trong lĩnh vực nghiên cứu. Tuy nhiên, không phải ai cũng biết cách để xây dựng và sử dụng các hệ thống hỗ trợ việc giải các bài toán của mình.

Một trong những vấn đề của các bài toán khoa học là phải thực hiện rất nhiều lệnh tính toán giống nhau cho một tập hợp bộ dữ liệu đầu vào cực lớn và phải giải quyết được việc tính toán trong khoảng thời gian hợp lý. Nếu thực hiện việc tính toán trên các hệ thống máy tính cá nhân hay máy chủ đơn lẻ thông thường thì thời gian để giải một bài toán này có thể mất hàng tháng, hàng năm đến nhiều năm. Để rút ngắn quá trình tính toán, người ta tìm cách chia nhỏ bài toán ra làm các phần, mỗi phần chạy trên một bộ xử lý hoặc một máy tính khác nhau, sau đó gộp kết quả lại. Đó chính là tư duy trong việc tính toán hiệu năng cao hay còn gọi là HPC.

Nhiều người thường nhầm lẫn HPC là các siêu máy tính, tuy nhiên điều này chưa đúng lắm. Các máy tính cá nhân thông thường cũng có thể được ghép lại để tạo thành một hệ thống HPC, và bên cạnh đó các siêu máy tính cũng có thể ghép lại với nhau, hoặc ghép vào cùng với các máy tính cá nhân để tạo thành một hệ thống HPC. Do đó không thể gọi HPC là siêu máy tính vì HPC có thể có siêu máy tính hoặc không.

Nghiên cứu trong luận văn sẽ đề xuất một quy trình cho việc tự xây dựng một hệ thống tính toán hiệu năng cao đơn giản, có thể ghép nối từ một vài máy tính cá nhân bình thường hoặc cao cấp hơn là ghép nối các hệ thống máy chủ có bộ xử lý tốt hơn, nhằm xây dựng được hệ thống phục vụ việc tính toán song song. Nhờ đó, việc xử lý các bài toán khoa học được đẩy nhanh lên gấp nhiều lần so với việc chạy bài toán theo cách thông thường.

2. Mục đích nghiên cứu

- Tìm hiểu hệ thống tính toán hiệu năng cao: Quản lý tài nguyên, Quản trị người dùng, Cấp phát tài nguyên.

- Đề xuất mô hình hệ thống tính toán hiệu năng cao và xây dựng thử nghiệm hệ thống sử dụng nền tảng mã nguồn mở, bao gồm nhóm chức năng sau:

+ Quản lý tài nguyên.

+ Quản lý cấp phát tài nguyên tính toán.

+ Quản lý người dùng và phân quyền sử dụng.

- Đánh giá hệ thống tính toán hiệu năng cao đã được xây dựng.

3. Nội dung nghiên cứu

Luận văn tập trung nghiên cứu những vấn đề sau:

- Tổng quan về tính toán hiệu năng cao, trong đó có giới thiệu và nêu các khái niệm cơ bản về tính toán hiệu năng cao, các giải pháp tính toán hiệu năng cao, giới thiệu phần mềm Rocks Cluster, kiến trúc hệ thống tính toán hiệu năng cao sử dụng Rocks Cluster.

- Xây dựng hệ thống tính toán hiệu năng cao sử dụng Rocks Cluster: Mô hình triển khai; Các chức năng của hệ thống; Quản lý người dùng và phân quyền sử dụng; Quản lý lệnh tính toán;

- Thử nghiệm và đánh giá hệ thống tính toán hiệu năng cao vừa xây dựng: Phương pháp thử nghiệm và các chỉ tiêu đánh giá: Đánh giá hiệu năng hệ thống sử dụng HPLinpack; Đánh giá tốc độ tính toán dựa trên bài toán mẫu; Tổng kết kết quả đánh giá hệ thống.

4. Phương pháp nghiên cứu

Sử dụng phương pháp nghiên cứu lý thuyết kết hợp với thực nghiệm, đánh giá, mô phỏng.

5. Cơ sở khoa học và tính thực tiễn của đề tài

Tính toán Hiệu năng cao nói chung đề cập đến thực tiễn tổng hợp sức mạnh tính toán theo cách mang lại hiệu suất cao hơn nhiều so với máy tính thông thường hoặc máy trạm để giải quyết các vấn đề lớn trong khoa học, kỹ thuật hoặc kinh doanh.

Trong môi trường nghiên cứu, học tập, với kinh phí giới hạn, chúng ta luôn có thể tự xây dựng cho mình một hệ thống tính toán hiệu năng cao có đầy đủ thành phần, chức năng bằng cách này hay cách khác.

Thông qua luận văn của mình, tôi muốn trình bày một trong những giải pháp đơn giản, phổ biến để xây dựng một hệ thống tính toán hiệu năng cao, tận

dụng phần cứng là các máy tính, máy chủ đơn lẻ sẵn có kết hợp với bộ phần mềm mã nguồn mở Rocks Cluster.

6. Những đóng góp của luận văn

Các kết quả nghiên cứu của luận văn có thể giúp xây dựng hệ thống tính toán hiệu năng cao có khả năng bảo đảm các chức năng:

- Quản lý tài nguyên
- Quản lý cấp phát tài nguyên tính toán
- Quản lý người dùng và phân quyền sử dụng

Ngoài ra kết quả nghiên cứu cũng hướng dẫn bạn đọc:

- Có thể tự xây dựng cho mình công cụ để đánh giá được năng lực của hệ thống
- Hiểu biết cụ thể hơn về ứng dụng thực tế của hệ thống thông qua bài toán mẫu
- Đánh giá được khả năng cải thiện tốc độ xử lý các bài toán khoa học khi ứng dụng hệ thống vào thực tế.

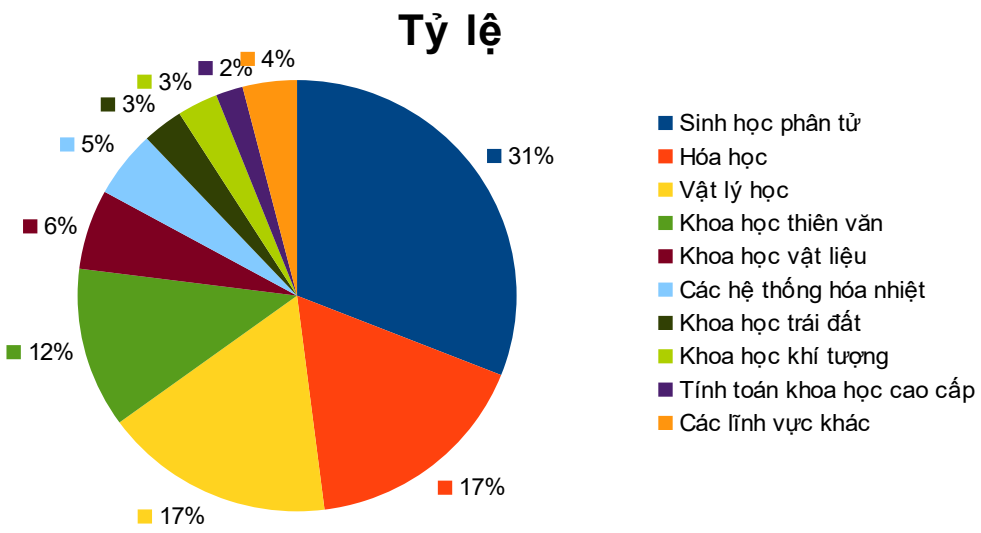
Chương 1. TỔNG QUAN VỀ TÍNH TOÁN HIỆU NĂNG CAO

1.1. Giới thiệu về tính toán hiệu năng cao

Ngày nay, tính toán khoa học đang trở nên ngày càng quan trọng trong nhiều lĩnh vực nghiên cứu. Các vấn đề dần trở nên phức tạp hơn, đòi hỏi sự phối hợp của đội ngũ các nhà nghiên cứu thuộc nhiều lĩnh vực chuyên môn khác nhau. Trong ngành tính toán khoa học, tính toán hiệu năng cao đóng vai trò cốt lõi, bổ sung các phương pháp nghiên cứu khoa học công nghệ truyền thống (tức là viết một lý thuyết hoặc thiết kế trên giấy, sau đó tiến hành thử nghiệm hoặc xây dựng hệ thống) bằng cách thực hiện các tính toán số học trên các hiện tượng và/hoặc các thử nghiệm mà có thể quá khó (ví dụ xây dựng một đường hầm thông gió lớn), quá đắt (ví dụ tạo một máy bay phản lực chở khách để thử nghiệm va chạm), quá chậm (ví dụ nghiên cứu sự biến đổi khí hậu hoặc sự tiến hóa của dải ngân hà), quá nguy hiểm (ví dụ thử nghiệm vũ khí hạt nhân) hoặc gây tranh cãi (ví dụ nghiên cứu tế bào gốc).

Tính toán hiệu năng cao tạo điều kiện thuận lợi cho các nhà nghiên cứu, những người muốn thực hiện một lượng rất lớn các phép tính lặp đi lặp lại trên một lượng lớn dữ liệu để đạt được các kết quả hợp lệ trong một khoảng thời gian hợp lý. Một số lĩnh vực khoa học đã và đang sử dụng tính toán hiệu năng cao:

- Sinh học, chuỗi di truyền, y học công nghệ gen
- Hóa học lượng tử và vật lý tương đối
- Vũ trụ và thiên văn học
- Vật liệu cao phân tử
- Động lực học chất lưu
- Dự báo thời tiết và mô hình hóa môi trường
- Giả lập thử nghiệm va chạm
- Kinh tế, tài chính



Hình 1.1 1. Số liệu theo Extreme Science & Engineering Discovery Environment

Tính toán hiệu năng cao mang lại nhiều hiệu quả kinh tế, có thể kể ra sau đây một số ví dụ:

- Trong lĩnh vực hàng không, các tối ưu hóa hoạt động hậu cần được đánh giá trên các hệ thống tính toán hiệu năng cao tiết kiệm xấp xỉ 100 triệu USD cho mỗi hãng hàng không mỗi năm.

- Trong lĩnh vực thiết kế xe hơi, các công ty lớn sử dụng trên 500 CPU trong CAD và CAM để thử nghiệm va chạm, tính toàn vẹn cấu trúc và khí động lực học, điều đó giúp tiết kiệm trên 1 tỷ USD cho mỗi công ty mỗi năm.

- Trong ngành công nghiệp bán dẫn, các công ty bán dẫn sử dụng các hệ thống lớn (trên 500 CPU) để giả lập các thiết bị điện tử và kiểm định logic, điều đó giúp tiết kiệm 1 tỷ USD cho mỗi công ty mỗi năm.

- Trong ngành công nghiệp chứng khoán, tính toán hiệu năng cao tiết kiệm 15 tỷ USD mỗi năm cho tiền thế chấp nhà cửa ở Mỹ.

Tóm lại, tính toán hiệu năng cao đang được sử dụng bởi các nhà nghiên cứu trên toàn thế giới, từ các nhóm lớn đến các cá nhân, để bổ sung cho phương pháp nghiên cứu khoa học truyền thống. Với các phần cứng tính toán hiệu năng cao có giá thành hợp lý cùng một với Hệ điều hành và các phần mềm mã nguồn mở, việc xây dựng các hệ thống tính toán hiệu năng cao đang nằm trong tầm

với của các cơ quan tổ chức có ngân sách nhỏ hẹp. Tuy nhiên để có thể triển khai được hiệu quả các lợi ích mà tính toán hiệu năng cao cung cấp, người ta cần có hiểu biết chi tiết về kiến trúc của hệ thống tính toán hiệu năng cao, cũng như làm chủ các công cụ phát triển và các kỹ thuật lập trình nâng cao đặc trưng của tính toán hiệu năng cao. Việc phát triển ứng dụng khoa học dựa trên tính toán hiệu năng cao thường bị hạn chế bởi việc thiếu nhân lực được đào tạo trong lĩnh vực này. Do đó, việc có thêm hiểu biết về tính toán hiệu năng cao sẽ giúp các nhà nghiên cứu sử dụng hiệu quả hơn các tài nguyên tính toán hiệu năng cao và tương tác tốt hơn với các đồng nghiệp.

Có nhiều định nghĩa khác nhau cho cụm từ "tính toán hiệu năng cao" (HPC, High Performance Computing), tùy vào việc quan niệm "hiệu năng cao" nghĩa là gì. Đó có thể là việc có thể giải quyết được những bài toán lớn, phức tạp hay việc có thể nhận được kết quả nhanh chóng, mà những việc này không thể thực hiện được trên các máy tính thông thường. Do đó khái niệm tính toán hiệu năng cao thường đồng nghĩa với khái niệm "siêu tính toán". Tuy nhiên, không có giới hạn định mức về một hệ thống máy tính cần phải mạnh như thế nào để được coi là "hiệu năng cao", vì khả năng tính toán của các bộ xử lý đã tăng nhanh trong những năm qua, một giới hạn vào thời điểm hiện tại có thể nhanh chóng trở nên lỗi thời trong tương lai gần.

Tính toán hiệu năng cao theo cách hiểu phổ biến nhất được định nghĩa là việc sử dụng một lượng lớn (ở cấp độ nhiều hơn hẳn) so với giới hạn tài nguyên của các máy trạm cá nhân hiện tại, và thường yêu cầu tính toán đồng thời, còn gọi là tính toán song song. Tính toán hiệu năng cao do đó bao gồm một tập hợp hệ thống phần cứng, các công cụ phần mềm, các ngôn ngữ lập trình, và các cơ chế lập trình song song đủ mạnh để thực hiện các phép tính mà trước đây không thể thực hiện được [13].

1.2. Các giải pháp tính toán hiệu năng cao

Có nhiều giải pháp cho tính toán hiệu năng cao. Mô hình thông dụng nhất là sử dụng các máy chủ chuyên dụng hoặc máy tính cá nhân bố trí tập trung lại và kết nối để tạo thành một cấu trúc được gọi là cụm máy tính. Cụm máy tính là một tập hợp các máy tính được kết nối với nhau, làm việc cùng nhau, để chúng có thể được xem như một hệ thống duy nhất. Cụm máy tính thường được triển khai để cải thiện hiệu năng và tính khả dụng so với một máy

tính đơn, và tiết kiệm chi phí hơn nhiều so với một máy tính đơn có cùng tốc độ. Mỗi nút trong cụm máy tính được thiết lập để thực hiện cùng một công việc được điều khiển và lên lịch bằng phần mềm. Các thành phần của một cụm thường được kết nối với nhau qua mạng cục bộ có tốc độ cao, với mỗi nút chạy thực thể hệ điều hành của riêng nó. Trong đa số các trường hợp, tất cả các nút sử dụng cùng phần cứng và hệ điều hành. Sau đây là bảng so sánh thông tin chung về một số phần mềm cụm máy tính đáng chú ý [14]:

Bảng 1.2 1. So sánh một số phần mềm cụm máy tính

Phần mềm	Đơn vị phát triển	Trạng thái phát triển	Giấy phép	Các nền tảng được hỗ trợ	Chi phí	Hỗ trợ trả phí
DIET	INRIA, SysFera, mã nguồn mở	Đang phát triển	CeCILL	Unix-like, Windows, Mac OS X, AIX	Miễn phí	Có
Enduro/X	Mavimax, Ltd.	Đang phát triển	GPLv2 or Commercia	Linux, FreeBSD, MacOS, Solaris, AIX	Miễn phí / Tính phí	Có
Apache Mesos	Apache	Đang phát triển	Apache license v2.0	Linux	Miễn phí	Có
OpenHPC	OpenHPC project	Đang phát triển	Free software	Linux (CentOS)	Miễn phí	Có
Rocks Cluster Distribution	Mã nguồn mở/NSF grant	Đang phát triển	Mã nguồn mở	CentOS Linux	Miễn phí	Không
Spectrum LSF	IBM	Đang phát triển	Proprietary	Unix, Linux, Windows	Miễn phí / Tính phí	Có
Platform Cluster Manager	Platform Computing	Đang phát triển	Mã nguồn mở	Linux	Miễn phí	Không
SynfiniWay	Fujitsu	Đang phát triển	Commercial	Unix, Linux, Windows	Tính phí	Không
ProActive	INRIA, ActiveEon, mã nguồn mở	Đang phát triển	GPL	Unix-like, Windows, Mac OS X	Miễn phí	Không
Techila Distributed Computing	Techila Technologies Ltd.	Đang phát triển	Proprietary	Linux, Windows	Tính phí	Có

Phần mềm	Đơn vị phát triển	Trạng thái phát triển	Giấy phép	Các nền tảng được hỗ trợ	Chi phí	Hỗ trợ trả phí
Engine						
Altair Grid Engine	Altair	Đang phát triển	Proprietary	*nix/Windows	Tính phí	Không
Oracle Grid Engine | Oracle Grid Engine (Sun Grid Engine, SGE)	Altair	Đang phát triển	Proprietary	*nix/Windows	Tính phí	Không
TORQUE Resource Manager	Adaptive Computing	Đang phát triển	Proprietary	Linux, *nix	Tính phí	Có
Moab Cluster Suite	Adaptive Computing	Đang phát triển	Proprietary	Linux, Mac OS X, Windows, AIX, OSF/Tru-64, Solaris, HP-UX, IRIX, FreeBSD & other UNIX platforms	Tính phí	Có
PBS Pro	Altair	Đang phát triển	AGPL or Proprietary	Linux, Windows	Miễn phí / Tính phí	Có
Proxmox Virtual Environment	Proxmox Server Solutions	Đang phát triển	Open-source AGPLv3	Linux, Windows, other OS are known to work and are community supported	Miễn phí	Có

Qua các nghiên cứu sơ bộ, nội dung luận văn sẽ tìm hiểu và quyết định xây dựng hệ thống tính toán hiệu năng cao dựa trên bản phân phối Rocks Clusters (http://www.rocksclusters.org/) vì nhiều ưu điểm và tính phù hợp với điều kiện thực tế:

- Mã nguồn mở giúp tiết kiệm chi phí giấy phép, đồng thời cho phép khả năng sửa đổi phần mềm để có được những tính năng mới hoặc tốt hơn.

- Phần mềm đang được phát triển, có thể nâng cấp hoặc nhận các bản cập nhật bảo mật.

- Phần mềm được sử dụng phổ biến trên các hệ thống tính toán hiệu năng cao, có lượng người dùng lớn, việc tìm giải pháp từ cộng đồng mã nguồn mở cho các vấn đề gặp phải cũng dễ dàng và nhanh chóng hơn.

- Dễ dàng cài đặt, có nhiều module bổ sung đã được tích hợp hoặc đóng gói sẵn, giúp đơn giản hóa việc thêm các tính năng mới vào hệ thống.

1.3. Giới thiệu phần mềm Rocks Cluster

Bản phân phối cụm Rocks (ban đầu được gọi là NPACI Rocks) là một bản phân phối Linux được xây dựng với mục đích phục vụ các cụm tính toán hiệu năng cao. Nó được xây dựng lần đầu bởi National Partnership for Advanced Computational Infrastructure và Trung tâm Siêu máy tính San Diego (SDSC) vào năm 2000. Rocks ban đầu dựa trên bản phân phối Red Hat Linux, tuy nhiên các phiên bản hiện đại của Rocks dựa trên CentOS, với một trình cài đặt Anaconda được sửa đổi để đơn giản hóa việc cài đặt hàng loạt lên nhiều máy tính. Rocks bao gồm nhiều công cụ (chẳng hạn MPI) không phải là một phần của CentOS nhưng là các thành phần tích hợp giúp chuyển một nhóm các máy tính thành một cụm tính toán hiệu năng cao.

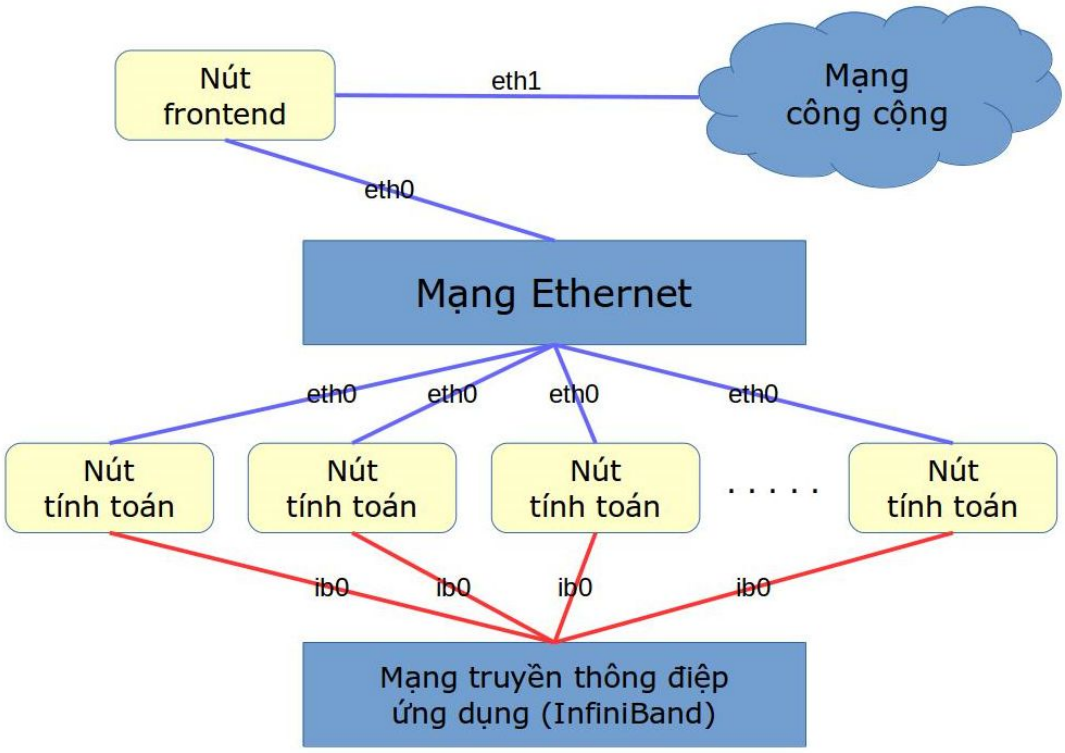
Việc cài đặt có thể được tùy chỉnh bằng các gói phần mềm bổ sung vào lúc cài đặt hoặc bằng cách sử dụng gói phần mềm bên ngoài do người dùng cung cấp (gọi là các roll). Các roll mở rộng hệ thống bằng cách tích hợp một cách trong suốt và tự động vào các cơ chế quản lý và đóng gói được sử dụng bởi phần mềm cơ bản, đơn giản hóa một cách đáng kể việc cài đặt và cấu hình một số lượng lớn máy tính. Trên 10 roll đã được tạo, bao gồm SGE, Condor, Lustre, Java, Ganglia...

Rocks là phần mềm được sử dụng cho các tổ chức học viện, chính phủ và thương mại, được sử dụng trong gần 1400 cụm tính toán, trên cả năm châu lục. Cụm tính toán mang tính hàn lâm lớn nhất được đăng ký có 8632 CPU là GridKa, được điều hành bởi Học viện Công nghệ Karlsruhe ở Karlsruhe, Đức. Cũng có một số cụm nhỏ hơn, đại diện cho các bước đầu để xây dựng các hệ thống lớn, cũng như trong các khóa học về thiết kế cụm tính toán.

Rocks là một phần mềm mã nguồn mở, phiên bản được triển khai sử dụng là 7.0, tên mã Manzanita, dựa trên nền CentOS 7.4, được phát hành vào ngày 01 tháng 12 năm 2017 [15].

1.4. Kiến trúc hệ thống tính toán hiệu năng cao sử dụng Rocks Cluster

Có 3 hệ thống tính toán hiệu năng cao đã được luận văn tiến hành triển khai và thử nghiệm và thử nghiệm: CPU, GPU và lai giữa CPU và GPU. Cả ba hệ thống tính toán hiệu năng cao đều sử dụng kiến trúc có lược đồ như hình bên dưới.



Hình 1.4 1. Lược đồ kiến trúc hệ thống tính toán hiệu năng cao

Thiết lập phần cứng cho một cụm tính toán về cơ bản bao gồm các thành phần sau [4]:

- **Nút frontend**: là một máy của cụm thực hiện giao tiếp với thế giới bên ngoài. Nhiều dịch vụ mạng (NFS, NIS, DHCP, NTP, MySQL, HTTP, …) chạy trên nút này. Hầu hết các thao tác quản trị hệ thống được thực hiện trên frontend. Frontend cũng là nơi người dùng đăng nhập, đệ trình các công việc, biên dịch mã nguồn, v.v... Nút này cũng có thể hoạt động như một bộ định

tuyến cho các nút khác trong cụm bằng cách sử dụng dịch địa chỉ mạng (NAT). Frontend sử dụng hai giao diện ethernet: giao diện ethernet đầu tiên (eth0) kết nối với các nút tính toán trong mạng riêng, giao diện ethernet thứ hai (eth1) kết nối với thế giới bên ngoài và sử dụng địa chỉ IP theo quy định.

- **Nút tính toán**: đây là các máy còn lại của cụm thực hiện nhiệm vụ tính toán. Các nút tính toán có thể tháo ra lắp vào hệ thống một cách linh động. Phần mềm trên frontend cho phép cài đặt lại toàn bộ hệ điều hành trên mọi nút tính toán trong một khoảng thời gian ngắn (cỡ 10 phút). Các nút tính toán không nhìn thấy được trên Internet công cộng.

- **Mạng Ethernet**: tất cả các nút của cụm tính toán được kết nối qua ethernet trên mạng riêng. Mạng này được sử dụng để quản trị, giám sát và chia sẻ tập tin cơ bản. Hệ thống sử dụng một chuyển mạch (switch) Gigabit Ethernet để kết nối giao diện ethernet đầu tiên (eth0) trên tất cả các nút của cụm tính toán. Sau khi kết nối phần cứng, các nút tính toán cần được thiết lập trong BIOS để khởi động không cần bàn phím.

- **Mạng truyền thông điệp ứng dụng**: tất cả các nút tính toán được kết nối với nhau qua giao diện mạng InfiniBand (ib0) với chuyển mạch InfiniBand. InfiniBand là một loại liên kết mạng máy tính có độ trễ thấp, băng thông cao, cho phép các chương trình tính toán song song truyền thông điệp hiệu năng cao. InfiniBand hiện là chuẩn liên kết nối được sử dụng phổ biến nhất trong các siêu máy tính.

Chương 2. XÂY DỰNG HỆ THỐNG TÍNH TOÁN HIỆU NĂNG CAO SỬ DỤNG ROCKS CLUSTER

2.1. Mô hình triển khai

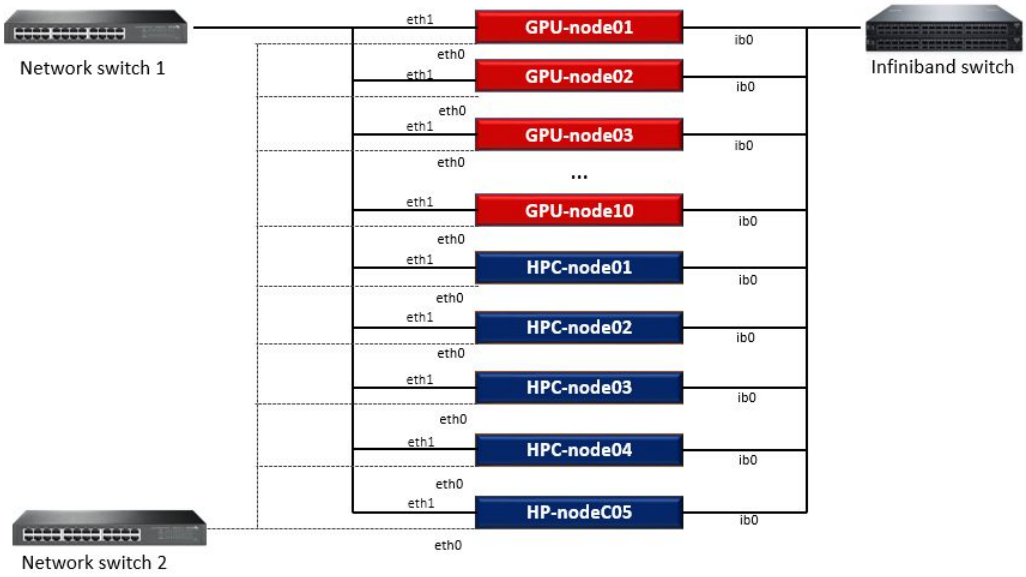
Để xây dựng mô hình như đã đề cập ở trên, về cơ sở hạ tầng phần cứng, chúng tôi có 1 máy Headnode làm máy điều khiển, 5 máy CPU và 10 máy GPU với cấu hình cơ bản như sau:

Máy Headnode và máy CPU:

- Bộ nhớ RAM DDR4 dung lượng 128GB
- 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz

Máy GPU:

- Bộ nhớ RAM DDR4 dung lượng 128GB
- 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
- Bộ xử lý đồ họa NVIDIA Tesla P100



Hình 2.1 1. Hệ thống máy tính toán hiệu năng cao

Mỗi máy (bao gồm máy không có bộ xử lý đồ họa lẫn máy có bộ xử lý đồ họa) có hai cổng giao diện mạng Gigabit Ethernet (eth0 và eth1 trong hình vẽ trên) và một cổng giao diện mạng tốc độ cao 40 Gb/s Infiniband (ib0 trong hình vẽ trên). 24 cổng giao diện mạng Ethernet đầu tiên và 24 cổng giao diện mạng Ethernet thứ hai được nối vào hai chuyển mạch (switch) Ethernet. 24 cổng giao

diện Infiniband được nối vào chuyển mạch Infiniband. Mỗi chuyển mạch có 24 cổng.

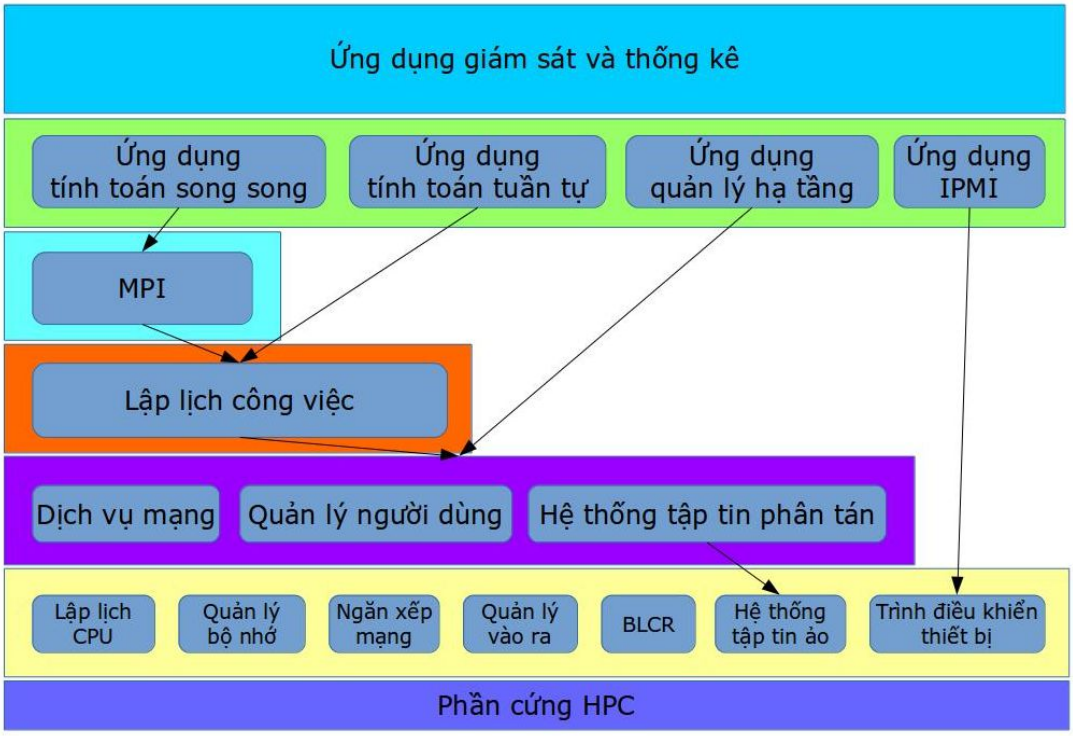
2.2. Các chức năng của hệ thống

Bảng sau đây liệt kê các tính năng đã xây dựng của hệ thống tính toán hiệu năng cao cùng các module phần mềm đảm bảo tính năng tương ứng. OS, Base và HPC là ba roll bắt buộc không thể thiếu của Rocks. SGE, Ganglia là các roll tùy chọn đã được tích hợp mặc định trong Rocks 6.1.1. Chúng tôi đã kết hợp SGE với một số kịch bản cho phép thiết lập một cơ chế đơn giản hóa việc sử dụng SGE của người dùng. GlusterFS là roll cài thêm không có trong Rocks. Các module tính năng khác như Intel MPI, HPLinpack, BLCR... không được cung cấp dưới dạng roll nhưng được cài thêm [3].

Bảng 2.2 1. Các tính năng của hệ thống tính toán hiệu năng cao đã xây dựng

Tính năng	Chi tiết tính năng	Module
Quản lý tài nguyên vật lý	Quản lý hiện trạng CPU	Roll OS
	Quản lý hiện trạng bộ nhớ RAM	
	Quản lý hiện trạng đĩa cứng	
	Quản lý hiện trạng NIC	
	Quản lý hiện trạng cổng điều khiển	
Quản lý dịch vụ mạng	Bật tắt dịch vụ mạng	Roll OS Roll Base
	Cài đặt dịch vụ mạng mới	
	Quản lý các cổng dịch vụ	
Quản lý cấp phát tài nguyên tính toán	Thiết lập giới hạn tài nguyên tính toán cho chương trình ứng dụng	
	Hỗ trợ các ứng dụng đồ họa tương tác	Roll SGE
	Hỗ trợ các ứng dụng xử lý dữ liệu tự động theo mảng công việc	
	Hỗ trợ các chương trình song song	
	Quản lý các công việc	
Quản lý người dùng và phân quyền sử dụng	Tạo người dùng và phân quyền sử dụng	Roll Base
	Thay đổi hoặc xóa thông tin về người dùng và quyền sử dụng	
Hỗ trợ tính toán hiệu năng cao trên môi trường MPI	Tính toán hiệu năng cao trên các nút tính toán không có GPU	Roll HPC OpenMPI MPICH
	Tính toán hiệu năng cao trên các nút tính	

Tính năng	Chi tiết tính năng	Module
	toán có GPU	Intel MPI HPLinpack
	Tính toán hiệu năng cao trên các nút tính toán không đồng nhất	
	Đo hiệu năng tính toán hiệu năng cao	
Quản lý phân chia tài nguyên lưu trữ	Hạn ngạch đĩa cứng	Roll GlusterFS
	Ghép nối ảo các ổ đĩa cứng	
Bảo đảm an toàn và an ninh dữ liệu lưu trữ	Cơ chế dự phòng phần cứng	RAID Roll GlusterFS
	Cơ chế dự phòng phần mềm	
Sao lưu và khôi phục dữ liệu chia sẻ	Sao lưu dữ liệu	
	Khôi phục dữ liệu	
Giám sát tài nguyên tính toán	Xem trạng thái của các nút tính toán	
	Xem chi tiết hiện trạng của các nút tính toán	Roll Ganglia
Giám sát hệ thống mạng hạ tầng tính toán	Xem địa chỉ và trạng thái giao diện mạng	Roll OS Roll Base
	Kiểm tra khả năng kết nối mạng	
Hỗ trợ quản lý sự kiện liên quan đến hạ tầng tính toán	Thiết lập trạng thái mặc định cho các nút tính toán khi khởi động	Roll OS
	Cài đặt tự động các nút tính toán	
	Thông báo sự cố phần cứng	
Hỗ trợ phân tích thống kê sử dụng hạ tầng tính toán	Thống kê về thời gian sử dụng của	Roll OS Roll Base
	Thống kê về thời gian chạy chương trình	
Quản lý nền tảng thông minh	Thiết lập các tùy chọn quản lý nền tảng	
	Đọc các giá trị cảm biến phần cứng	IPMI
	Bật tắt các nút tính toán từ xa	
Phòng chống rủi ro trong tính toán	Lưu trạng thái của chương trình đang chạy	Roll SGE BLCR
	Phục hồi trạng thái của chương trình đang chạy	



Hình 2.2 1. Các khối chức năng cơ bản của một hệ thống tính toán hiệu năng cao

Các tính năng phần mềm của hệ thống tính toán hiệu năng cao bao gồm:

- Quản lý tài nguyên vật lý: tính năng này được thực hiện bởi hạt nhân hệ điều hành Linux bao gồm: lập lịch CPU, quản lý bộ nhớ, quản lý ngăn xếp mạng, quản lý hệ thống vào ra thông qua các trình điều khiển thiết bị...

- Quản lý dịch vụ mạng: cung cấp khả năng bật tắt cũng như duy trì các dịch vụ mạng cần thiết trên hệ thống chẳng hạn NFS (Network File System), DHCP (Dynamic Host Configuration Protocol), NTP (Network Time Protocol), HTTP (Hyper Text Transfer Protocol), SSH (Secure Shell)...

- Quản lý người dùng: cung cấp khả năng bổ sung, sửa đổi, xóa người dùng, nhóm người dùng trên hệ thống, cũng như quản lý các quyền có liên quan trong môi trường đa người dùng, bao gồm cả danh sách điều khiển truy cập.

- Quản lý phân chia tài nguyên lưu trữ, bảo đảm an toàn và an ninh dữ liệu lưu trữ, sao lưu và khôi phục dữ liệu lưu trữ: thực hiện việc phân chia dung lượng đĩa cứng cho người dùng theo các hạn ngạch, đáp ứng nhu cầu lưu trữ vượt dung lượng của một ổ đĩa đơn với sự hỗ trợ của hệ thống tập tin phân tán, cũng như các cơ chế sao lưu dự phòng để đảm bảo an toàn dữ liệu lưu trữ, tránh

hoặc giảm nguy cơ mất dữ liệu khi có sự cố.

- Quản lý cấp phát tài nguyên tính toán: thực hiện việc cấp phát tự động các tài nguyên tính toán như số lượng CPU, lượng bộ nhớ, thời gian sử dụng hệ thống theo yêu cầu của từng ứng dụng tính toán, tự động chọn các nút tính toán thích hợp để chạy một phiên ứng dụng tính toán nào đó nhằm đảm bảo tính cân bằng và hiệu quả. Tính năng này được thực hiện bởi một phần mềm gọi là trình lập lịch công việc (job scheduler), trong đó các phiên ứng dụng tính toán cần chạy được gọi là các công việc (job).

- Hỗ trợ tính toán hiệu năng cao: các ứng dụng tính toán hiệu năng cao thường là các ứng dụng tính toán song song, liên kết nhiều CPU trên nhiều máy khác nhau để thực hiện đồng thời một công đoạn tính toán. Để thực hiện điều này, chúng cần một cơ chế trao đổi dữ liệu tốc độ cao giữa các máy. Giao diện truyền thông điệp (MPI tức Message Passing Interface) cung cấp một cơ chế như vậy. Hệ thống cung cấp nhiều môi trường MPI cho các ứng dụng tính toán song song qua các thư viện MPI khác nhau, cũng như cung cấp khả năng sử dụng đơn vị xử lý đồ họa (GPU tức Graphics Processing Unit) để tăng tốc độ tính toán.

- Hỗ trợ quản lý sự kiện liên quan đến hạ tầng tính toán: bao gồm việc thiết lập trạng thái của các nút tính toán khi có nguồn điện trở lại (sau sự cố mất điện), tự động cài đặt lại hệ điều hành trên các nút tính toán khi khởi động, cài đặt phần mềm tự động trên hàng loạt nút tính toán, thông báo khi có lỗi phần cứng hoặc khi đến giới hạn cảnh báo (chẳng hạn sắp hết dung lượng lưu trữ).

- Giám sát và thống kê: bao gồm theo dõi tình trạng của các nút tính toán, kiểm tra tính kết nối của hệ thống mạng, thống kê thời gian sử dụng của người dùng, thời gian chạy chương trình ứng dụng, lưu lượng và dung lượng sử dụng v.v...

- Quản lý nền tảng thông minh: bao gồm việc bật tắt từ xa các nút tính toán theo yêu cầu, đọc các giá trị cảm biến phần cứng cũng như thiết lập các tùy chọn điện năng với các mức hoạt động/tiết kiệm điện khác nhau. Tính năng này được hỗ trợ thông qua giao diện quản lý nền tảng thông minh (IPMI tức Intelligent Platform Management Interface).

- Phòng chống rủi ro trong tính toán: cho phép lưu lại trạng thái của

chương trình đang chạy và tiếp tục nó vào một thời điểm khác. Tính năng này giúp người dùng tiết kiệm thời gian, không phải chạy lại toàn bộ chương trình từ đầu khi xảy ra sự cố mất điện, được hỗ trợ bởi module BLCR trong hạt nhân Linux.

2.3. Quản lý người dùng và phân quyền sử dụng

2.3.1 Nguyên tắc và cách cấp phát

Mỗi người dùng đăng ký sử dụng hệ thống tính toán hiệu năng cao được cung cấp một tài khoản truy cập. Người dùng có thể cung cấp trước mật khẩu hoặc nhận mật khẩu ban đầu và sau đó tự do thay đổi. Trong trường hợp quên mật khẩu truy cập, người dùng có thể yêu cầu người quản trị đặt lại mật khẩu.

Người dùng truy cập hệ thống rocks qua giao thức ssh (secure shell). Giao thức này thiết lập một cơ chế xác thực dựa trên khóa. Vào lần đầu người dùng truy cập, khóa ssh sẽ được lưu trên máy của người dùng. Hệ thống tạo ra một thư mục riêng của người dùng (thư mục <username>) trong /state/partition1/home/. Rocks tự động đồng bộ thư mục /state/partition1/home/<username> với /home/<username>. Người dùng khi truy cập sẽ được tự do sử dụng thư mục cá nhân của mình /home/<username>.

Mỗi người dùng có thể được đặt vào các nhóm người dùng khác nhau. Mỗi nhóm người dùng có thể phục vụ cho một mục đích riêng, chẳng hạn phân theo các đơn vị trong tổ chức.

Người dùng có thể bị giới hạn dung lượng lưu trữ và tài nguyên tính toán (CPU, RAM) trong trường hợp ảnh hưởng đến những người dùng khác.

2.3.2 Thao tác quản lý và phân quyền sử dụng cho người dùng

Việc quản lý người dùng và phân quyền sử dụng trên cụm tính toán dựa trên Roll Base và nó tận dụng các tiện ích quản lý người dùng của nền tảng Linux.

Sử dụng tài khoản root, người quản trị hệ thống có thể tạo tài khoản mới, thiết lập mật khẩu cho người dùng. Người dùng có thể thay đổi mật khẩu sau khi đã truy cập hệ thống. Người quản trị cũng có thể thêm một nhóm người dùng mới.

Mỗi người dùng có thể được sửa đổi để thêm vào các nhóm khác nhau. Việc phân người sử dụng vào các nhóm khác nhau có thể giúp dễ dàng phân các mức quyền khác nhau có liên quan đến các dịch vụ hệ thống. Tuy nhiên một người dùng bất kỳ luôn có một nhóm chính, mặc định nhóm chính có tên trùng với tên người dùng. Mỗi người dùng được hệ thống xác định bằng một định danh gọi là UID (user id) và mỗi nhóm được hệ thống xác định bằng một định danh gọi là GID (group id). UID và GID là cơ sở để hệ thống thực hiện phân quyền trên hệ thống tập tin. Theo quy ước, các UID và GID dưới 500 dành cho các dịch vụ hệ thống (có thể hiểu là các người dùng ảo), các UID và GID từ 500 trở lên dành cho các tài khoản người dùng thực tế. Danh sách người dùng được lưu trong /etc/passwd và danh sách nhóm được lưu trong /etc/group. Người quản trị sử dụng các tiện ích quản lý của Linux để thao tác thay vì sửa đổi trực tiếp các tập tin này.

Với các tài khoản không còn hiệu lực, người quản trị có thể xóa một người dùng hoặc một nhóm người dùng.

2.4. Quản lý lệnh tính toán

2.4.1 Giới thiệu về module lập lịch SGE và phương pháp dựa trên SGE

Rocks Cluster được tích hợp một công cụ là Sun Grid Engine (SGE) [16] [17], cung cấp tính năng quản lý việc cấp phát tài nguyên tính toán với một số cải tiến. Phương pháp này cho phép đơn giản hóa việc sử dụng của người dùng. Người sử dụng có thể dễ dàng sử dụng một cách hiệu quả hệ thống tính toán hiệu năng cao mà không cần phải mất thời gian để tìm hiểu về SGE như thực trạng của các hệ thống hiện có. Về cơ bản, hệ thống tính toán hiệu năng cao chỉ nên sử dụng một module lập lịch duy nhất. Trong số các giải pháp lập lịch đã nghiên cứu, chúng tôi lựa chọn SGE do nó được tích hợp sẵn trong Rocks 6.1.1.

Sun Grid Engine (SGE) còn được biết đến với tên CODINE (Computing in Distributed Networked Environments) - Tính toán trong Môi trường Mạng Phân tán - là một hệ thống phần mềm cụm máy tính theo mô hình điện toán lưới, thực hiện nhiệm vụ quản lý hàng đợi công việc và xử lý hàng loạt, do Sun Microsystems phát triển và hỗ trợ. Công nghệ Grid Engine được thương mại hóa và phát hành dưới tên Oracle Grid Engine sau khi Oracle mua lại Sun Microsystems. Tiếp sau đó là Univa Grid Engine khi tập đoàn Univa sở hữu

quyền sở hữu tài sản trí tuệ và thương hiệu đối với công nghệ này. Dự án mã nguồn mở gốc của Grid Engine đã đóng, nhưng các phiên bản của công nghệ này vẫn mở dưới giấy phép mã nguồn chuẩn công nghiệp của Sun (SISSL). Các dự án mới sử dụng mã nguồn gốc của dự án và được biết dưới tên Son of Grid Engine và Open Grid Scheduler, về cơ bản có chức năng tương tự. Phiên bản mã nguồn mở của công nghệ Grid Engine được triển khai trên hệ thống tính toán hiệu năng cao tại Trung tâm Tin học và Tính toán.

Grid Engine thường được sử dụng trên một cụm tính toán hiệu năng cao và chịu trách nhiệm tiếp nhận, lập lịch, điều khiển và quản lý việc thực thi phân tán một số lượng lớn các công việc tuần tự, song song hoặc có tương tác người dùng. Nó cũng quản lý và lập lịch việc cấp phát tài nguyên phân tán như bộ xử lý, bộ nhớ, không gian đĩa và giấy phép phần mềm.

Các tính năng trong phiên bản Grid Engine được sử dụng:

- Đặt chỗ trước
- Mảng công việc
- Điều khiển hạn ngạch tài nguyên dựa trên quy tắc
- Thi hành từ xa tăng cường (không sử dụng các tiến trình rshd/rlogind/sshd bên ngoài)
- Đa cụm
- Daemon được quản lý bởi Service Management Facility trên Solaris
- Hỗ trợ giả TTY cho các công việc tương tác
- Kiểm tra đệ trình công việc
- Trình cài đặt giao diện đồ họa
- Lập lịch và ràng buộc thread theo hình thái (topology)
- Tích hợp Hadoop, tích hợp Amazon EC2 cho điện toán đám mây

Các tính năng khác của SGE bao gồm:

- Nhiều giải thuật lập lịch nâng cao cho phép cấp phát tài nguyên dựa trên chính sách
- Hàng đợi cụm

- Tính năng chịu lỗi - Grid Engine tiếp tục hoạt động miễn là còn một hoặc nhiều máy
- Điểm kiểm tra (checkpoint) công việc
- Mảng và tác vụ công việc
- DRMAA (API công việc)
- Đặt trước tài nguyên
- Báo cáo trạng thái XML và giao diện web xml-qstat
- Thống kê tình trạng sử dụng
- Console thống kê và báo cáo (ARCO)
- make song song: distmake, dmake và qmake của riêng SGE
- Tích hợp Flexlm và quản lý giấy phép phần mềm đa cụm

Grid Engine là một phần mềm đa nền tảng, có khả năng chạy trên nhiều nền tảng bao gồm cả Linux. Một cụm Grid Engine điển hình chứa một máy chính (master) và một hoặc nhiều máy thi hành. Nhiều máy chính thứ cấp (shadow master) cũng có thể được cấu hình như biện pháp dự phòng, chúng sẽ thực hiện vai trò của máy chính khi máy chính gốc có sự cố. Các triển khai đáng chú ý của SGE bao gồm:

- Sun Grid
- Siêu máy tính TSUBAME ở học viện Công nghệ Tokyo, đứng thứ 7 trên danh sách TOP500 vào tháng 6/2006
- Siêu máy tính Ranger ở Trung tâm Tính toán Cấp cao Texas (TACC). Ranger có hiệu năng đỉnh 504 Tflops, là siêu máy tính mạnh thứ 4 trong danh sách TOP500 năm 2008
- Trung tâm Siêu máy tính San Diego (SDSC)
- Phòng thí nghiệm Động lực học Chất lưu Địa vật lý (NOAA GFDL)

Phần mềm Grid Engine đáp ứng các yêu cầu của tính toán lưới (grid computing). Một lưới (grid) là một tập hợp các tài nguyên tính toán nhằm thực hiện các công việc. Ở dạng đơn giản nhất, đối với người dùng, một lưới xuất hiện như một hệ thống lớn cung cấp một điểm truy cập đơn nhất vào các tài

nguyên phân tán. Ở dạng phức tạp hơn, một lưới có thể cung cấp nhiều điểm truy cập cho người dùng. Trong tất cả các trường hợp, người dùng đối xử với lưới như một tài nguyên tính toán đơn nhất. Phần mềm quản lý tài nguyên như Grid Engine chấp nhận các công việc do người dùng đệ trình, nó sử dụng các chính sách quản lý tài nguyên để lập lịch các công việc cần chạy trên các hệ thống thích hợp trong lưới. Người dùng có thể đệ trình hàng triệu công việc một lúc mà không cần phải bận tâm các công việc được chạy ở đâu.

Có ba lớp lưới chính, biến đổi theo quy mô từ các hệ thống đơn giản tới các khu phức hợp siêu máy tính sử dụng hàng ngàn bộ xử lý:

- Lưới cụm (cluster grid) là lớp đơn giản nhất. Lưới cụm được tạo nên từ một tập hợp các máy (host) làm việc cùng nhau. Một lưới cụm cung cấp một điểm truy cập duy nhất cho người dùng.

- Lưới campus (campus grid) cho phép nhiều bộ phận trong một tổ chức lớn chia sẻ các tài nguyên. Các tổ chức có thể sử dụng lưới campus để xử lý nhiều công việc khác nhau, từ các hoạt động kinh doanh định kỳ, tới việc khai thác dữ liệu, xử lý hình ảnh và nhiều hơn thế...

- Lưới toàn cầu (global grid) là một tập hợp các lưới campus vượt qua biên giới của các tổ chức để tạo nên các hệ thống ảo rất lớn. Người dùng có truy cập vào sức mạnh tính toán vượt xa các tài nguyên hiện có trong phạm vi tổ chức của họ.

Phần mềm Grid Engine cung cấp sức mạnh và tính linh động mà các lưới campus cần. Nó cũng có tác dụng cho các lưới cụm. Hệ thống Grid Engine điều phối việc cấp phát sức mạnh tính toán dựa trên các chính sách tài nguyên được thiết lập bởi đội ngũ kỹ thuật và quản trị của tổ chức. Phần mềm sử dụng các chính sách này để xem xét các tài nguyên tính toán khả dụng trong lưới campus, sau đó sẽ thu thập và cấp phát các tài nguyên một cách tự động, tối đa hóa việc sử dụng trên toàn lưới campus. Để cho phép hợp tác trong lưới campus, những người sở hữu dự án sử dụng lưới phải làm các việc sau:

- Thỏa thuận về các chính sách

- Phát triển các chính sách linh động để điều chỉnh thủ công cho các yêu cầu của một dự án cụ thể

- Quản lý và thi hành các chính sách một cách tự động

Phần mềm Grid Engine có thể dàn xếp quyền lợi của nhiều dự án đang cạnh tranh các tài nguyên tính toán.

Sau đây là các thành phần quan trọng nhất của một hệ thống Grid Engine:

*** Máy (host)**

Có bốn kiểu máy cơ bản đối với hệ thống grid engine:

- Máy chính: máy chính là trung tâm của toàn bộ các hoạt động trên cụm. Máy chính chạy daemon sge_qmaster và daemon lập lịch sge_schedd. Cả hai daemon điều khiển tất cả các thành phần của hệ thống grid engine, chẳng hạn như các hàng đợi và công việc. Các daemon bảo trì bảng trạng thái của các thành phần, quyền truy cập của người dùng. Theo mặc định, máy chính cũng là một máy quản trị và một máy đệ trình.

- Các máy thực thi: các máy thực thi là các hệ thống có quyền thực thi các công việc. Do đó các máy thực thi có các thực thể hàng đợi gắn kèm với chúng. Các máy thực thi thi hành daemon sge_execd.

- Các máy quản trị: các máy quản trị là các máy có quyền thực hiện bất kỳ hoạt động quản trị nào đối với hệ thống grid engine.

- Các máy đệ trình: các máy đệ trình cho phép người dùng đệ trình và điều khiển chỉ các công việc hàng loạt (batch job). Cụ thể, một người dùng đã đăng nhập vào một máy đệ trình có thể đệ trình các công việc với lệnh qsub, có thể quản lý các trạng thái công việc với lệnh qstat, và có thể sử dụng giao diện người dùng qmon của hệ thống grid engine.

*** Daemon**

Có ba daemon cung cấp tính năng của hệ thống grid engine:

- Daemon chính sge_qmaster: là trung tâm của các hoạt động quản lý và lập lịch trong cụm, sge_qmaster bảo trì bảng thông tin về các máy, hàng đợi, công việc, tải trọng hệ thống và quyền người dùng. sge_qmaster nhận quyết định lập lịch từ sge_schedd và yêu cầu các hành động từ sge_execd trên các máy thực thi thích hợp.

- Daemon lập lịch sge_schedd: daemon lập lịch bảo trì một view cập nhật về trạng thái của cụm với sự trợ giúp của sge_qmaster. Daemon lập lịch thực hiện các quyết định lập lịch sau:

- Các công việc nào cần gửi tới các hàng đợi nào
- Cách sắp xếp lại và xác định lại độ ưu tiên cho các công việc để bảo trì độ ưu tiên và thời hạn.

Sau đó dameon chuyển các quyết định này cho sge_qmaster, daemon sẽ khởi tạo các hành động cần thiết

- Daemon thực thi sge_execd: daemon thực thi chịu trách nhiệm cho các thực thể hàng đợi trên máy của nó và cho việc chạy các công việc trong các thực thể hàng đợi này. Theo định kỳ, daemon thực thi chuyển tiếp thông tin chẳng hạn như trạng thái hoặc tải trọng công việc trên máy của nó cho sge_qmaster.

*** Hàng đợi**

Hàng đợi là một nơi để chứa một lớp các công việc được phép chạy đồng thời trên một hoặc nhiều máy. Một hàng đợi xác định các tính chất nào đó của công việc. Trong suốt thời gian thực thi, một công việc đang chạy được gắn với hàng đợi của nó. Liên kết với một hàng đợi ảnh hưởng đến một số thứ có thể xảy ra với một công việc. Ví dụ, nếu một hàng đợi bị treo, tất cả các công việc gắn với hàng đợi đó cũng sẽ bị treo.

Các công việc không cần phải được đệ trình trực tiếp tới một hàng đợi. Chỉ cần xác định hồ sơ yêu cầu của công việc. Một hồ sơ có thể bao gồm các yêu cầu như bộ nhớ, hệ điều hành, phần mềm khả dụng, … Phần mềm grid engine tự động gửi công việc tới một hàng đợi thích hợp và một máy thích hợp có tải trọng thực thi nhẹ. Nếu một công việc được đệ trình tới một hàng đợi cụ thể, công việc được ràng buộc với hàng đợi này. Như thế, các daemon của hệ thống grid engine không thể chọn một thiết bị thích hợp hơn hoặc một thiết bị có tải trọng nhẹ hơn.

Một hàng đợi có thể nằm trên một máy đơn, hoặc có thể mở rộng trên nhiều máy. Vì lý do này, các hàng đợi của hệ thống grid engine còn được gọi là các hàng đợi cụm (cluster queue). Các hàng đợi cụm cho phép người dùng và người quản trị làm việc với một cụm các máy thực thi thông qua một cấu hình hàng đợi duy nhất. Mỗi máy được gắn với một hàng đợi cụm nhận được thực thể hàng đợi (queue instance) riêng từ hàng đợi cụm.

*** Lệnh cho người dùng**

Đây là một tập hợp các chương trình dòng lệnh cho phép thực hiện các công việc sau:

- Quản lý các hàng đợi
- Đệ trình và xóa các công việc
- Kiểm tra trạng thái của công việc
- Treo hoặc cho phép các hàng đợi và công việc

Hệ thống grid engine cung cấp nhiều chương trình dòng lệnh.

QMON là giao diện người dùng đồ họa của hệ thống grid engine.

2.4.2 Triển khai tính năng quản lý cấp phát tài nguyên tính toán

SGE đã được tích hợp mặc định trong bản phân phối Rocks dưới dạng một module (roll). Hệ thống được thiết lập để các chương trình tính toán mà người dùng đã đăng ký tự động chạy qua SGE, tăng hiệu quả cũng như đơn giản hóa các thao tác của người dùng. Người dùng không cần phải hiểu về cơ chế hoạt động của SGE, chỉ cần khai báo các nhu cầu cần thiết cho chương trình tính toán. Người quản trị có thể thiết lập các giới hạn tài nguyên dựa trên các khai báo này. Sau đây là một số tùy chọn khai báo liên quan đến các giới hạn tài nguyên tính toán:

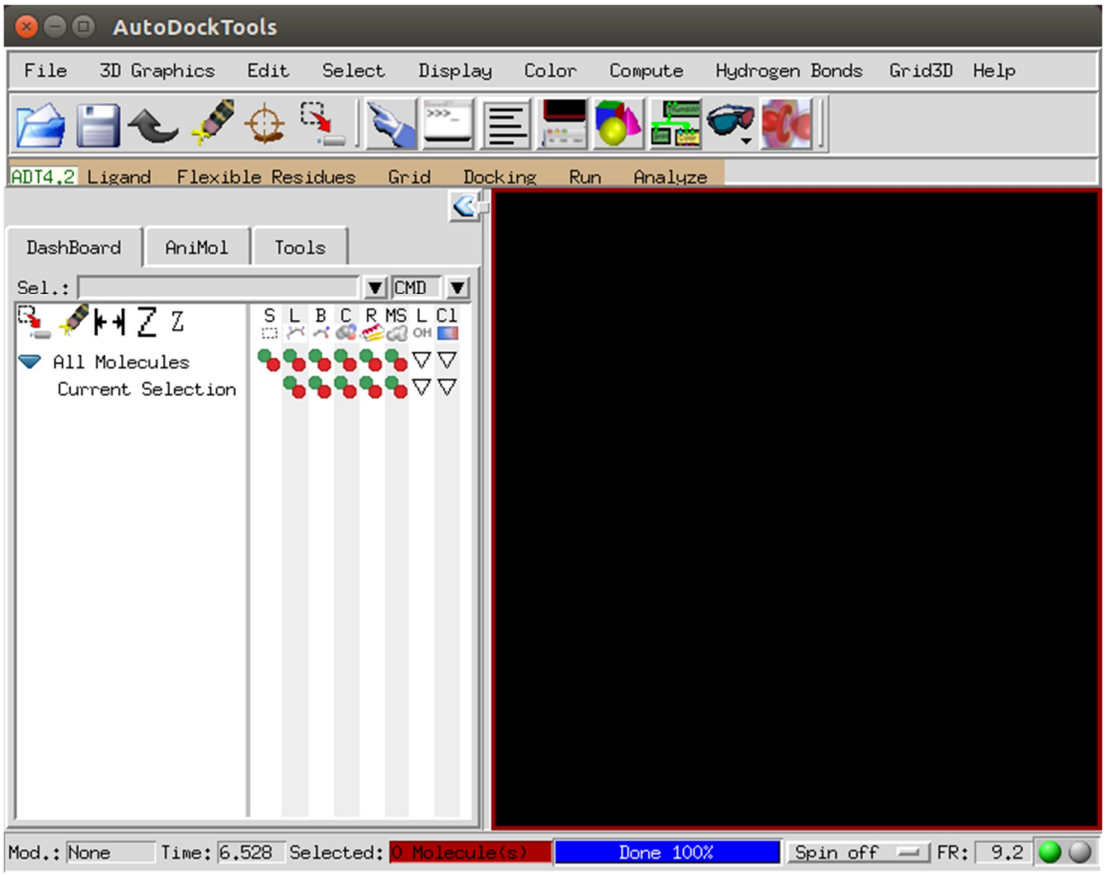
Bảng 2.4 1. Một số tùy chọn liên quan đến tài nguyên tính toán

Tùy chọn	**Ý nghĩa**
Kiến trúc CPU	- Chỉ chạy chương trình trên các nút tính toán có kiến trúc CPU được chỉ định (Intel hoặc AMD). - Tùy chọn này thường chỉ cần thiết nếu chương trình có yêu cầu bắt buộc về kiến trúc CPU.
Thời gian chạy	- Thời gian chạy tối đa của chương trình. - Giá trị mặc định của SGE là 8 giờ.
Bộ nhớ	- Lượng bộ nhớ ảo (bao gồm cả các phân vùng tráo đổi) tối đa tính theo GB. - Nếu chương trình là tuần tự, đây là giới hạn lượng bộ nhớ tổng cộng. - Nếu chương trình là song song, đây là giới hạn lượng bộ nhớ tối đa được cấp phát cho mỗi bộ xử lý (không phải lượng bộ nhớ tổng cộng).
Bộ nhớ vật lý	- Lượng bộ nhớ thực (RAM vật lý) tối đa tính theo GB.

Tùy chọn	Ý nghĩa
	- Nếu chương trình là tuần tự, đây là giới hạn lượng bộ nhớ thực tổng cộng. - Nếu chương trình là song song, đây là giới hạn lượng bộ nhớ thực tối đa được cấp phát cho mỗi bộ xử lý (không phải lượng bộ nhớ thực tổng cộng). - Giá trị của giới hạn bộ nhớ vật lý nên luôn nhỏ hơn hoặc bằng giá trị của bộ nhớ nói trên, và tất nhiên không vượt quá lượng RAM vật lý của một nút tính toán. Tỷ lệ bộ nhớ thực/bộ nhớ ảnh hưởng đến các thao tác "tráo đổi trang" (paging) trong quá trình thực thi chương trình, do đó có thể tác động đến hiệu quả của chương trình.
Số lượng CPU	- Chỉ định môi trường song song (ví dụ OpenMPI) và số lượng CPU cần sử dụng.

SGE chủ yếu được sử dụng cho các chương trình chạy ở chế độ tự động (không tương tác, hay chế độ batch), tuy nhiên nó cũng hỗ trợ các chương trình chạy ở chế độ tương tác (cần các thao tác của người dùng, chẳng hạn chương trình với giao diện đồ họa).

Để chạy các chương trình giao diện đồ họa trong SGE, hệ thống đã được thiết lập để cho phép tính năng X11 Forwarding. X11 Forwarding, là một cách thức sử dụng giao thức SSH để cho phép người dùng chạy các ứng dụng đồ họa ở trên server và tương tác với các ứng dụng đó thông qua màn hình và các thiết bị xuất nhập trên máy client của họ. Trên máy cá nhân, người dùng cần sử dụng một trình client X11, chẳng hạn tùy chọn -X trong ssh đã tích hợp trong các bản phân phối Linux, hoặc Xming kết hợp với Putty trên Windows, X11 trên Mac OS. Hình bên dưới là minh họa việc chạy ứng dụng AutoDockTools chạy trên hệ thống, với cửa sổ giao diện đồ họa được hiển thị trên máy người dùng.



Hình 2.4 1. Giao diện AutoDockTools

Một tính năng khác của SGE là hỗ trợ chạy mảng công việc (job array). Mảng công việc là việc thực hiện đồng thời nhiều tác vụ (task) của cùng một công việc không tương tác (batch job) nhưng với các dữ liệu đầu vào khác nhau. SGE hỗ trợ tính năng mảng công việc, tính năng này đặc biệt hữu ích khi cần chạy một chương trình với tập dữ liệu đầu vào có số lượng lớn. Tình huống này rất phổ biến trong lĩnh vực nghiên cứu, ví dụ các thử nghiệm tính toán để xác định lựa chọn tối ưu cho dữ liệu đầu vào.

Giả sử chúng ta cần chạy chương trình program với 1000 tập tin dữ liệu đầu vào, từ input.1 đến input.1000. Thay vì phải gọi chương trình program một cách thủ công 1000 lần hoặc đặt chúng trong một kịch bản để chạy các tác vụ lần lượt từ 1 đến 1000, mảng công việc SGE cho phép chỉ cần gọi chương trình program 1 lần để chạy tự động 1000 tác vụ theo cách song song, tức là nhiều tác vụ được thực hiện đồng thời. Các tác vụ sẽ chạy độc lập với nhau, mỗi tác vụ sử dụng tập tin đầu vào riêng và tạo ra tập tin kết quả tương ứng. SGE tự động quyết định bao nhiêu tác vụ hoạt động đồng thời, tùy thuộc vào tình trạng

của hệ thống tại thời điểm chạy. Theo cách này, một tác vụ khởi động sau có thể hoàn thành trước một tác vụ được khởi động trước nó.

SGE cũng được thiết lập để chạy các chương trình tính toán trong môi trường song song. Người dùng chỉ cần gõ tên chương trình MPI hay qua mpirun, chương trình luôn được đưa vào hàng đợi SGE để phân phối trên các nút tính toán.

*** Quản lý tác vụ, người dùng**

Người quản trị hệ thống có thể xem thông tin về công việc của người dùng, thông tin về các máy trong cụm, thông tin về hàng đợi trên các máy cũng như thông tin về công việc của người dùng trong giao diện đồ họa qmon của SGE.

*** Chính sách lập lịch**

Hệ thống lập lịch được cấu hình để sử dụng chính sách chia sẻ công bằng. Ban đầu, tất cả người dùng có quyền ưu tiên như nhau khi sử dụng các tài nguyên hệ thống. Nếu một người dùng sử dụng liên tục hệ thống tính toán trong thời gian gần đây, mức độ ưu tiên của người đó sẽ giảm đi so với những người dùng khác nếu tất cả người dùng cùng có nhu cầu tính toán vào thời điểm hiện tại. Tuy nhiên sau một thời gian gián đoạn không sử dụng hệ thống, mức độ ưu tiên của người dùng sẽ lại được phục hồi về giá trị tối đa như ban đầu.

2.4.3 Sử dụng hệ thống

Để có thể đưa lệnh lên chạy trên hệ thống, người dùng chuẩn bị file kịch bản (***script_file***) gồm các nội dung như sau:

Script_file	**Giải thích**
#!/bin/bash *#PBS -N* ***program_name*** *#PBS -j oe* *#PBS -m abe* *#PBS -l* ***select=1:ncpus=12:mem=8G*** *#PBS -q* ***para _cpu*** *cd $PBS_O_WORKDIR* ***./program.sh***	- ***program_name***: tên chương trình tính toán. - ***select=1:ncpus=12:mem=8G***: các tham số yêu cầu số lượng tài nguyên gồm số node, số CPU, dung lượng bộ nhớ RAM. - ***para_cpu***: phân loại chương trình tính toán đơn, tính toán song song… + long_cpu: sử dụng khi chạy chương trình đơn tính toán trên một CPU.

	+ para_cpu: sử dụng khi chạy chương trình song song, sử dụng thư viện MPI/OpenMP. - ***./program.sh***: đường dẫn và câu lệnh chạy chương trình tính toán.

Sau khi đã chuẩn bị file kịch bản ***script_file***, người dùng đưa lệnh tính toán lên hệ thống bằng lệnh:

 # qsub script_file

Xem danh sách các lệnh đang chạy trên hệ thống bằng câu lệnh:

 # qstat

qstat còn cho người dùng biết trạng thái câu lệnh của mình đưa lên hệ thống như thế nào. Các trạng thái điển hình bao gồm:

Queued: Lệnh đang nằm trong hàng chờ

Held: Lệnh bị dừng

Running: Lệnh đang chạy

Finished: Lệnh đã hoàn thành

```
[root@pbs01 bin]# qstat -u anna

pbs01:
                                                                  Req'd  Req'd   Elap
Job ID          Username Queue    Jobname    SessID NDS TSK Memory Time  S Time
--------------- -------- -------- ---------- ------ --- --- ------ ----- - -----
1155108.pbs01   anna     sleep    L19V1.o10L  40097   1  12   12gb 720:0 R 719:3
1180915.pbs01   anna     sleep    WU.o10Lng   26206   1  12   12gb 720:0 R 456:5
1180916.pbs01   anna     sleep    L165V2o10L    751   1  12   12gb 720:0 R 456:5
1180917.pbs01   anna     sleep    L165V15o10  35789   1  12   12gb 720:0 R 456:5
1254285.pbs01   anna     sleep    L19.15o10L  22294   1  12    4gb 720:0 R 124:4
1254286.pbs01   anna     sleep    L13V3LNG    19176   1  12   12gb 720:0 R 124:4
1265817.pbs01   anna     sleep    L13V4o10LN  10711   1  12    4gb 720:0 R 49:38
```

Hình 2.4 2. Kiểm tra trạng thái lệnh – Liệt kê theo người dùng

```
[root@login01 ~]# qstat -Qf p72
Queue: p72
    queue_type = Execution
    Priority = 70
    total_jobs = 28
    state_count = Transit:0 Queued:17 Held:0 Waiting:0 Running:11 Exiting:0 Beg
        un:0
    max_queued = [u:PBS_GENERIC=20]
    resources_max.walltime = 72:00:00
    resources_min.ncpus = 8
    resources_default.walltime = 72:00:00
    default_chunk.allowed_queues = p72
    resources_available.fairshare_enable = True
    resources_available.fairshare_set = public
    resources_available.ncpus = 412
    resources_assigned.mem = 1870gb
    resources_assigned.mpiprocs = 360
    resources_assigned.ncpus = 360
    resources_assigned.nodect = 17
    max_run = [u:PBS_GENERIC=7]
    max_run_res.ncpus = [u:PBS_GENERIC=96]
    enabled = True
    started = True
```

Hình 2.4 3. Kiểm tra trạng thái lệnh – Thông tin chi tiết của một lệnh tính toán

Xóa lệnh đang chạy bằng câu lệnh:

 # qdel job_id

2.4.4 Quản trị hệ thống

Bên cạnh các câu lệnh dành cho người dùng sử dụng hệ thống, còn có các câu lệnh quản trị hệ thống dành cho quản trị viên như sau:

pbsnodes: câu lệnh hiển thị các Node và tính trạng tài nguyen của các Node trong hệ thống

qhold: dừng (hold) một lệnh tính toán

qrls: tiếp tục xử lý (release) một lệnh tính toán

tracejob: lấy thông tin chi tiết về một lệnh tính toán

pbs-report: báo cáo chung về hệ thống

pbs_probe: báo cáo tình trạng hệ thống

```
[vinay@hpc01 1DEDIC]$ pbsnodes -a
hpc01
     Mom = hpc01.aero.iitb.ac.in
     Port = 15002
     pbs_version = 19.1.2
     ntype = PBS
     state = job-busy
     pcpus = 4
     jobs = 1.hpc01/0, 1.hpc01/1, 2.hpc01/2, 2.hpc01/3
     resources_available.arch = linux
     resources_available.host = hpc01
     resources_available.mem = 3029528kb
     resources_available.ncpus = 4
     resources_available.vnode = hpc01
     resources_assigned.accelerator_memory = 0kb
     resources_assigned.hbmem = 0kb
     resources_assigned.mem = 0kb
     resources_assigned.naccelerators = 0
     resources_assigned.ncpus = 4
     resources_assigned.vmem = 0kb
     resv_enable = True
     sharing = default_shared
     last_state_change_time = Wed Jun 29 10:35:14 2022
     last_used_time = Wed Jun 29 10:33:20 2022
```

Hình 2.4 4. Lệnh pbsnodes kiểm tra tài nguyên hệ thống

Chương 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ HỆ THỐNG TÍNH TOÁN HIỆU NĂNG CAO VỪA XÂY DỰNG

3.1. Phương pháp thử nghiệm và các chỉ tiêu đánh giá

Năng lực xử lý của siêu máy tính được đánh giá thông qua tiêu chí là: Số lượng phép tính số học dấu phẩy động mà hệ thống có thể thực hiện mỗi giây. Tốc độ thực hiện phép tính dấu chấm động trên giây, được viết tắt là FLOPS[1] (**FL**oating-point **O**perations **P**er **S**econd) [18].

Các nhà cung cấp máy tính và nhà cung cấp dịch vụ thường đề cập đến hiệu suất cao nhất theo lý thuyết (Rpeak) của hệ thống của họ, tính bằng FLOPS. Rpeak của hệ thống được tính bằng cách nhân số lượng bộ xử lý với tốc độ xung nhịp của bộ xử lý, sau đó nhân với số phép toán dấu phẩy động mà bộ xử lý có thể thực hiện trong một giây đo đạc bởi các chương trình tính điểm chuẩn tiêu chuẩn, chẳng hạn như LINPACK DP TPP, HPC Challenge (HPCC) hay SPEC [2][10][11].

FLOPS là đơn vị đo hiệu suất tính toán số của máy tính. Các phép tính dấu phẩy động thường được sử dụng trong các lĩnh vực như nghiên cứu tính toán khoa học. Frank H. McMahon, thuộc Phòng thí nghiệm quốc gia Lawrence Livermore, đã phát minh ra thuật ngữ FLOPS và MFLOPS (Mega-FLOPS) để ông có thể so sánh các siêu máy tính thời đó theo số phép tính dấu chấm động mà chúng thực hiện mỗi giây.

Trường hợp máy tính có một CPU, ta có thể áp dụng công thức sau:

$$\text{FLOPS} = \text{cores} \times \frac{\text{cycles}}{\text{second}} \times \frac{\text{FLOPs}}{\text{cycle}}.$$

FLOPS trên hệ thống HPC có thể được tính bằng phương trình sau:

$$\text{FLOPS} = \text{racks} \times \frac{\text{nodes}}{\text{rack}} \times \frac{\text{sockets}}{\text{node}} \times \frac{\text{cores}}{\text{socket}} \times \frac{\text{cycles}}{\text{second}} \times \frac{\text{FLOPs}}{\text{cycle}}.$$

[1] Chữ "S" trong từ viết tắt "FLOPS" là viết tắt của "giây" và được sử dụng kết hợp với "P" (cho "per") để biểu thị tốc độ. Tốc độ mỗi giây "FLOPS" thường bị hiểu sai thành dạng số nhiều của "FLOP" (viết tắt của "phép tính dấu phẩy động").

Công cụ thường được sử dụng để tính toán sức mạnh của hệ thống HPC là Linpack [19]. Linpack là một công cụ đánh giá tốc độ thực thi các phép toán dấu phẩy động của bộ xử lý. Kỹ thuật này áp dụng một thuật toán toán học tiêu chuẩn dựa trên việc giải đồng thời cùng một lúc các phương trình toán học tuyến tính. Jack Dongarra đã xây dựng Linpack để cung cấp cho người dùng một thước đo tiêu chuẩn về hiệu suất máy tính. Linpack rất hữu ích trong việc đánh giá tốc độ thực tế mà máy tính sẽ chạy một hoặc nhiều chương trình [7][8].

Có ba điểm chuẩn LINPACK khác nhau:

- Linpack Fortran n=100: được sử dụng để đo tốc độ của máy tính khi giải các phương trình tuyến tính bằng cách sử dụng ma trận hoặc mảng số chứa 100 số.

- Linpack n=1000: tương tự như điểm chuẩn n=100. Nó được sử dụng cho một ma trận gồm 1000 số.

- HPLinpack (High-Performance Linpack - HPL) [5]: Hai công cụ kể trên không phù hợp để thử nghiệm các máy tính song song. Để đánh giá điểm Linpack của hệ thống tính toán hiệu năng cao, chúng ta sử dụng công cụ HPL[6]. HPL có thể đánh giá hiệu năng của các cụm máy tính hiệu suất cao thông qua dựa trên một bài kiểm tra để giải các phương trình đơn tuyến tính bậc N thông qua phép thử Gauss. Điểm FLOPS cực đại là số phép tính dấu phẩy động mà máy tính có thể thực hiện mỗi giây. Điểm FLOPS cực đại này có thể được chia thành hai loại, điểm FLOPS lý thuyết và điểm FLOPS thực tế. Điểm FLOPS cực đại theo lý thuyết là số phép toán dấu phẩy động mà máy tính có thể thực hiện theo lý thuyết mỗi giây. Điểm FLOPS cực đại theo lý thuyết được xác định bởi tốc độ xung nhịp của CPU. Điểm FLOPS cực đại theo lý thuyết được tính công thức sau:

Công thức	**Giải thích**
$P_{Flops} = CPU_{Clock} * N_{CPU} * N_{Operation}$	P_{Flops} : Điểm FLOPS cực đại theo lý thuyết CPU_{Clock} : Tốc độ xung nhịp của CPU N_{CPU} : Số lõi CPU $N_{Operation}$: Số phép tính dấu phẩy động mà CPU thực hiện trên mỗi chu kỳ.

Trong khi đó điểm FLOPS thực tế sẽ được đánh giá bằng cách sử dụng HPL.

3.2. Đánh giá hiệu năng hệ thống sử dụng HPLinpack

Như đã đề cập ở Chương 2 – mục 2.1, hệ thống được xây dựng có cấu hình như sau :

Máy Headnode và máy CPU x5:

- Bộ nhớ RAM DDR4 dung lượng 128GB
- 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz

Máy GPU x10:

- Bộ nhớ RAM DDR4 dung lượng 128GB
- 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
- Bộ xử lý đồ họa NVIDIA Tesla P100

```
[root@cnode01 ~]#
[root@cnode01 ~]# free -h
              total        used        free      shared  buff/cache   available
Mem:           125G        4.2G         78G        184M         42G        120G
Swap:          8.0G        264M        7.7G
[root@cnode01 ~]#
[root@cnode01 ~]#
[root@cnode01 ~]#
```

Hình 3.2 1. Dung lượng RAM của các máy CPU / GPU

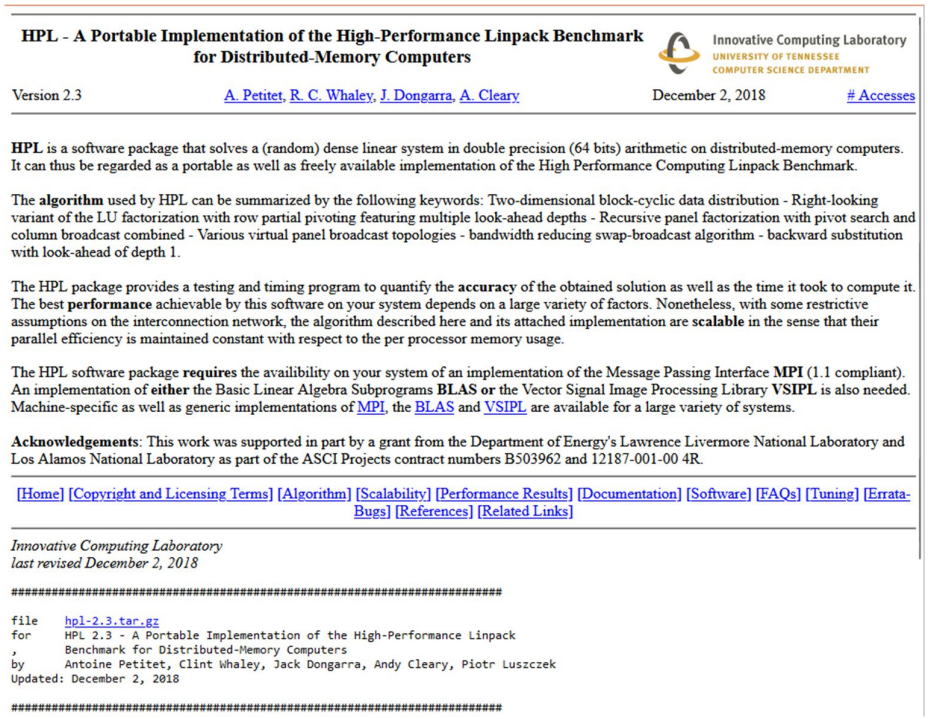
```
[root@cnode01 ~]#
[root@cnode01 ~]# lscpu
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                56
On-line CPU(s) list:   0-55
Thread(s) per core:    2
Core(s) per socket:    14
Socket(s):             2
NUMA node(s):          2
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 85
Model name:            Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
Stepping:              4
CPU MHz:               1000.073
CPU max MHz:           3700.0000
CPU min MHz:           1000.0000
BogoMIPS:              5200.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              1024K
L3 cache:              19712K
NUMA node0 CPU(s):     0-13,28-41
NUMA node1 CPU(s):     14-27,42-55
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx f
xsr sse sse2 ss ht tm pbe syscall nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology
 nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx smx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pc
id dca sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb c
at_l3 cdp_l3 invpcid_single intel_ppin intel_pt ssbd mba ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsba
se tsc_adjust bmi1 hle avx2 smep bmi2 erms invpcid rtm cqm mpx rdt_a avx512f avx512dq rdseed adx smap clflushopt clwb
 avx512cd avx512bw avx512vl xsaveopt xsavec xgetbv1 cqm_llc cqm_occup_llc cqm_mbm_total cqm_mbm_local dtherm ida arat
 pln pts hwp hwp_act_window hwp_pkg_req pku ospke md_clear spec_ctrl intel_stibp flush_l1d
[root@cnode01 ~]#
```

Hình 3.2 2. Thông số bộ xử lý CPU của các máy CPU / GPU

Với bộ xử lý CPU trên từng máy, mỗi máy có 2 bộ xử lý gắn trên 2 Socket, mỗi Socket có 14 lõi (Core), mỗi lõi hỗ trợ 2 luồng (Thread). Như vậy tổng cộng số luồng tính toán là 2x14x2 = 56 luồng (56 CPU)

Bản cài đặt HPL được phân phối tại đường dẫn

https://netlib.org/benchmark/hpl/



HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers

Innovative Computing Laboratory
UNIVERSITY OF TENNESSEE
COMPUTER SCIENCE DEPARTMENT

Version 2.3 | A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary | December 2, 2018 | # Accesses

HPL is a software package that solves a (random) dense linear system in double precision (64 bits) arithmetic on distributed-memory computers. It can thus be regarded as a portable as well as freely available implementation of the High Performance Computing Linpack Benchmark.

The **algorithm** used by HPL can be summarized by the following keywords: Two-dimensional block-cyclic data distribution - Right-looking variant of the LU factorization with row partial pivoting featuring multiple look-ahead depths - Recursive panel factorization with pivot search and column broadcast combined - Various virtual panel broadcast topologies - bandwidth reducing swap-broadcast algorithm - backward substitution with look-ahead of depth 1.

The HPL package provides a testing and timing program to quantify the **accuracy** of the obtained solution as well as the time it took to compute it. The best **performance** achievable by this software on your system depends on a large variety of factors. Nonetheless, with some restrictive assumptions on the interconnection network, the algorithm described here and its attached implementation are **scalable** in the sense that their parallel efficiency is maintained constant with respect to the per processor memory usage.

The HPL software package **requires** the availibility on your system of an implementation of the Message Passing Interface **MPI** (1.1 compliant). An implementation of **either** the Basic Linear Algebra Subprograms **BLAS or** the Vector Signal Image Processing Library **VSIPL** is also needed. Machine-specific as well as generic implementations of MPI, the BLAS and VSIPL are available for a large variety of systems.

Acknowledgements: This work was supported in part by a grant from the Department of Energy's Lawrence Livermore National Laboratory and Los Alamos National Laboratory as part of the ASCI Projects contract numbers B503962 and 12187-001-00 4R.

[Home] [Copyright and Licensing Terms] [Algorithm] [Scalability] [Performance Results] [Documentation] [Software] [FAQs] [Tuning] [Errata-Bugs] [References] [Related Links]

Innovative Computing Laboratory
last revised December 2, 2018

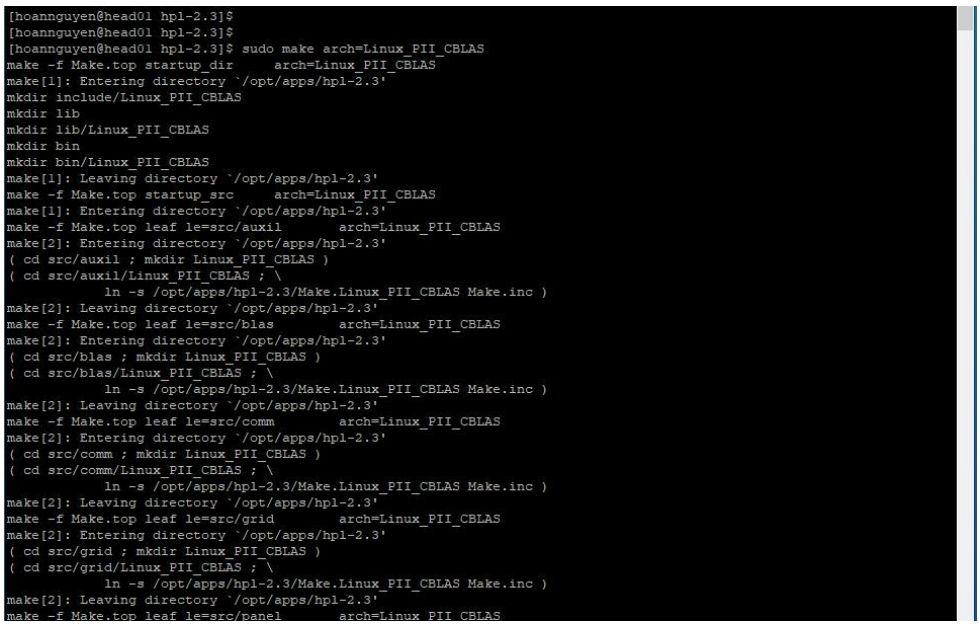
```
########################################################################

file    hpl-2.3.tar.gz
for     HPL 2.3 - A Portable Implementation of the High-Performance Linpack
,       Benchmark for Distributed-Memory Computers
by      Antoine Petitet, Clint Whaley, Jack Dongarra, Andy Cleary, Piotr Luszczek
Updated: December 2, 2018

########################################################################
```

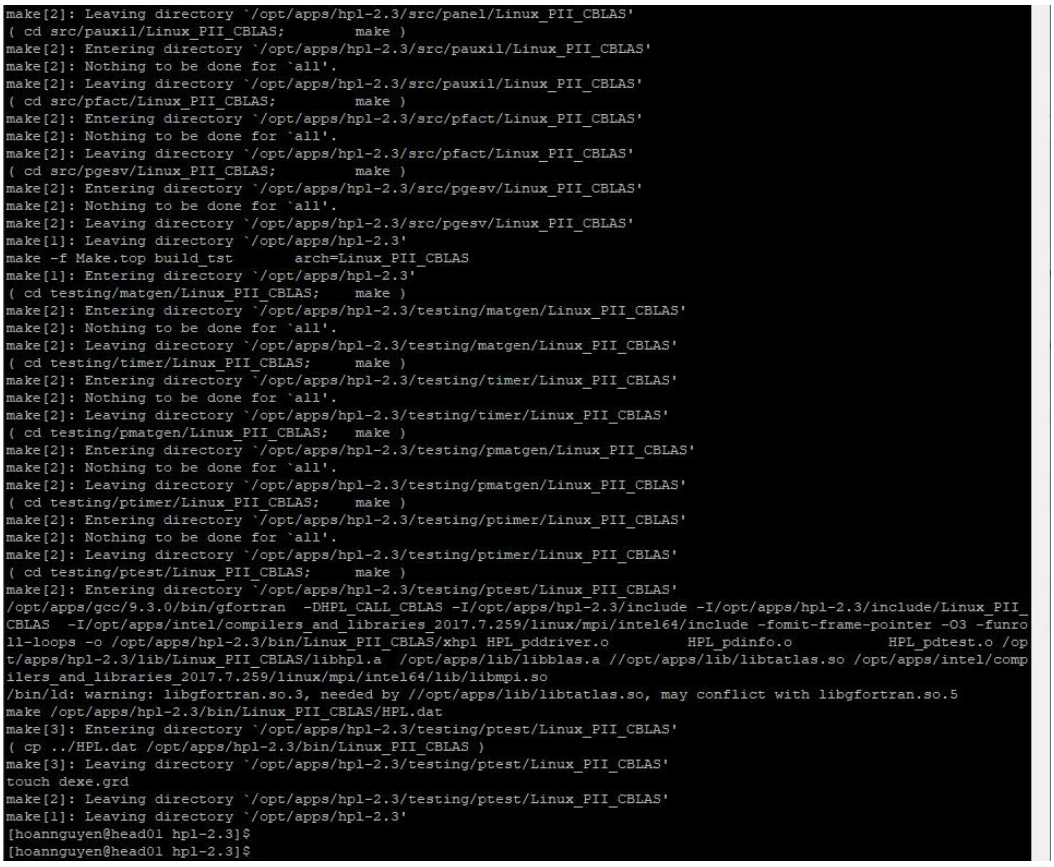
Hình 3.2 3. Giao diện chính trang HPL

Tiến hành biên dịch chương trình HPL trên hệ thống của chúng ta



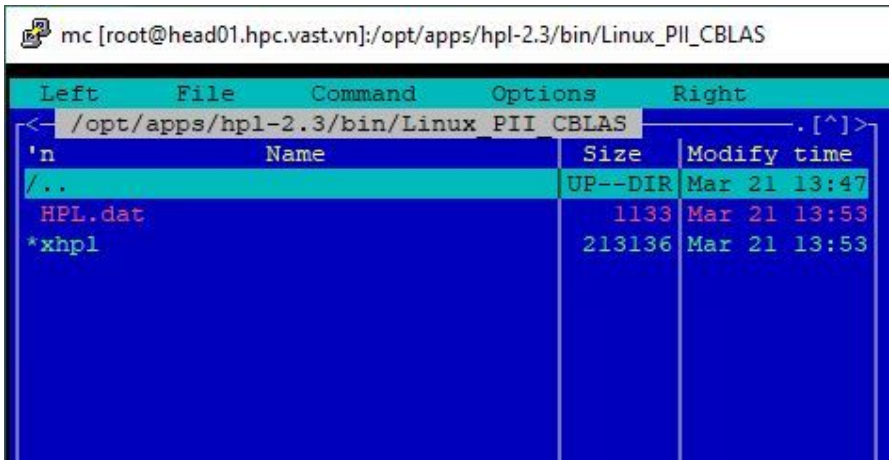
```
[hoannguyen@head01 hpl-2.3]$
[hoannguyen@head01 hpl-2.3]$
[hoannguyen@head01 hpl-2.3]$ sudo make arch=Linux_PII_CBLAS
make -f Make.top startup_dir     arch=Linux_PII_CBLAS
make[1]: Entering directory `/opt/apps/hpl-2.3'
mkdir include/Linux_PII_CBLAS
mkdir lib
mkdir lib/Linux_PII_CBLAS
mkdir bin
mkdir bin/Linux_PII_CBLAS
make[1]: Leaving directory `/opt/apps/hpl-2.3'
make -f Make.top startup_src     arch=Linux_PII_CBLAS
make[1]: Entering directory `/opt/apps/hpl-2.3'
make -f Make.top leaf le=src/auxil       arch=Linux_PII_CBLAS
make[2]: Entering directory `/opt/apps/hpl-2.3'
( cd src/auxil ; mkdir Linux_PII_CBLAS )
( cd src/auxil/Linux_PII_CBLAS ; \
            ln -s /opt/apps/hpl-2.3/Make.Linux_PII_CBLAS Make.inc )
make[2]: Leaving directory `/opt/apps/hpl-2.3'
make -f Make.top leaf le=src/blas        arch=Linux_PII_CBLAS
make[2]: Entering directory `/opt/apps/hpl-2.3'
( cd src/blas ; mkdir Linux_PII_CBLAS )
( cd src/blas/Linux_PII_CBLAS ; \
            ln -s /opt/apps/hpl-2.3/Make.Linux_PII_CBLAS Make.inc )
make[2]: Leaving directory `/opt/apps/hpl-2.3'
make -f Make.top leaf le=src/comm        arch=Linux_PII_CBLAS
make[2]: Entering directory `/opt/apps/hpl-2.3'
( cd src/comm ; mkdir Linux_PII_CBLAS )
( cd src/comm/Linux_PII_CBLAS ; \
            ln -s /opt/apps/hpl-2.3/Make.Linux_PII_CBLAS Make.inc )
make[2]: Leaving directory `/opt/apps/hpl-2.3'
make -f Make.top leaf le=src/grid        arch=Linux_PII_CBLAS
make[2]: Entering directory `/opt/apps/hpl-2.3'
( cd src/grid ; mkdir Linux_PII_CBLAS )
( cd src/grid/Linux_PII_CBLAS ; \
            ln -s /opt/apps/hpl-2.3/Make.Linux_PII_CBLAS Make.inc )
make[2]: Leaving directory `/opt/apps/hpl-2.3'
make -f Make.top leaf le=src/panel       arch=Linux_PII_CBLAS
```

.....



Hình 3.2 4. Quá trình biên dịch công cụ HPL

Sau khi biên dịch chương trình xong, ta có được file chạy của chương trình trong thư mục “bin”



Hình 3.2 5. Công cụ HPL đã được biên dịch thành công

Sử dụng lệnh để kiểm tra tình trạng Node tính toán của chúng ta, chọn ra một Node để chạy chương trình HPL

```
[hoannguyen@head01 ~]$ pbsnodes cnode04
cnode04
     Mom = cnode04.hpc.vast.vn
     ntype = PBS
     state = <various>
     pcpus = 56
     Priority = <various>
     resv_enable = True
     sharing = default_shared
     last_state_change_time = Thu Jan  1 08:00:00 1970
     last_used_time = Thu Jan  1 08:00:00 1970
     resources_available.arch = linux
     resources_available.host = cnode04
     resources_available.hpmem = 0b
     resources_available.mem = 131236864kb
     resources_available.ncpus = 56
     resources_available.ngpus = 0
     resources_available.qlist = para_cpu,openmp
     resources_available.vmem = 279117824kb
     resources_available.vnode = <various>
     resources_available.accelerator_memory = 0kb
     resources_available.hbmem = 0kb
     resources_available.naccelerators = 0
     resources_assigned.mem = 0kb
     resources_assigned.ncpus = 0
     resources_assigned.ngpus = 0
     resources_assigned.vmem = 0kb
     resources_assigned.accelerator_memory = 0kb
     resources_assigned.hbmem = 0kb
     resources_assigned.naccelerators = 0
```

Hình 3.2 6. Trạng thái của Node cnode04

Để chạy chương trình HPL và có được kết quả đo tốt (cực đại), ta cần phải đưa tham số phù hợp vào trong file cấu hình tham số đo đạc HPL.dat [20].

Các tham số quan trọng liên quan đến quá trình đo đạc là [21]:

N: Kích cỡ của bài toán.

Chương trình Linpack sẽ tạo ra bài toán với ma trận có kích thước

$(N^2.8)$ Bytes.

Trong thực tế, để có được kết quả tốt nhất, cần chạy chương trình với kích thước bài toán phù hợp với bộ nhớ của hệ thống. Dung lượng bộ nhớ mà HPL sử dụng chính là độ lớn của ma trận hệ số tương quan. Nếu sử dụng *N*

nhỏ, thì sẽ không đánh giá được hết năng lực của hệ thống. Nếu sử dụng *N* lớn quá, thì có thể gây ra việc hết dung lượng bộ nhớ RAM và phải sử dụng vùng nhớ Swap trên ổ cứng, cũng sẽ làm suy giảm hiệu suất. Ta nên sử dụng ma trận có kích thước khoảng 70% đến 80% dung lượng bộ nhớ, còn chừa một phần cho hệ điều hành cùng các tiến trình khác.

Ví dụ nếu kích thước bộ nhớ là 128Gb trong trường hợp của chúng ta thì ta có 70% bộ nhớ là 89.6GB. Để xác định N ta có biểu thức:

$$\boldsymbol{N = \sqrt{89.6GB\ /\ 8}}$$

Với các Node tính toán của chúng ta, bộ nhớ là

128GB = 137.438.953.472 Byte

Ta có bảng giá trị cho N như sau:

Bảng 3.2 1. Giá trị của N đối với trường hợp bộ nhớ 128GB

Percent	$N = \sqrt{137.438.953.472 * Percent / 8}$
70%	109.663
80%	117.234
90%	124.345

Trường hợp ta sử dụng 2 Node tính toán, tổng bộ nhớ là

128GB * 2 = 256GB = 274.877.906.944 Byte

Ta có bảng giá trị cho N như sau:

Bảng 3.2 2. Giá trị của N đối với trường hợp bộ nhớ 256GB

Percent	$N = \sqrt{274.877.906.944 * Percent / 8}$
70%	155.086
80%	165.794
90%	175.851

P và Q : Kích thước Ma trận xử lý

Giá trị của *P* và *Q* đại diện cho kích thước của ma trận xử lý trong bài toán của chúng ta (*P* và *Q* là số dòng và số cột của ma trận xử lý). Tích *P*×*Q* sẽ là tổng các tiến trình mà chúng ta muốn xử lý đồng thời (xử lý song song). Ví

dụ của một Node của chúng ta đang có 56 bộ xử lý và ta muốn đánh giá năng lực của toàn bộ Node thì ta cho lệnh chạy trên 56 bộ xử lý đồng thời cùng lúc.

Thông thường ta sẽ đặt *P* nhỏ hơn hoặc bằng *Q*, và giữ giá trị *P*, *Q* chênh lệch càng ít càng tốt (ma trận càng gần với hình vuông càng tốt), khi đó kết quả đo đạc càng gần với năng lực cực đại của hệ thống. Ta có thể thử nhiều kích thước ma trận khác nhau để so sánh.

Ta có bảng tham khảo giá trị của P và Q như sau:

Bảng 3.2 3. Giá trị của P/Q tính theo số bộ xử lý

Số bộ xử lý	**Giá trị của P**	**Giá trị của Q**
8	2	4
10	2	5
16	4	4
20	4	5
…	…	…
56	7	8
112	8	14
224	14	16

NB : Kích thước khối

Giá trị NB là kích thước của khối. HPL sử dụng NB để phân phối dữ liệu cũng như các tính toán chi tiết. Từ quan điểm trong việc phân phối dữ liệu, kích thước của khối càng nhỏ thì việc phân bổ càng đồng đều. Từ quan điểm tính toán, giá trị NB quá nhỏ sẽ hạn chế hiệu suất tính toán. Giá trị khối nhỏ cũng làm tăng khối lượng trao đổi thông tin.

Giá trị NB thường trong khoảng 100 - 256.

Giá trị NB được khuyến cáo là bội số của P * Q.

Ví dụ nếu P * Q = 8, sử dụng giá trị NB là bội số của 8 trong khoảng từ 100 và 256, ví dụ 8 * 20 = 160.

Chúng ta cần thử các giá trị để chọn ra được kết quả tốt nhất.

Sau khi đã xây dựng được tham số cho file HPL.dat, ta chạy lệnh đo đạc theo cấu trúc như sau:

Đo trên 1 Node:

 # mpirun -np 56 --host cnode01 ./xhpl

Đo trên 2 Node:

 # mpirun -np 112 --host cnode01,cnode02 ./xhpl

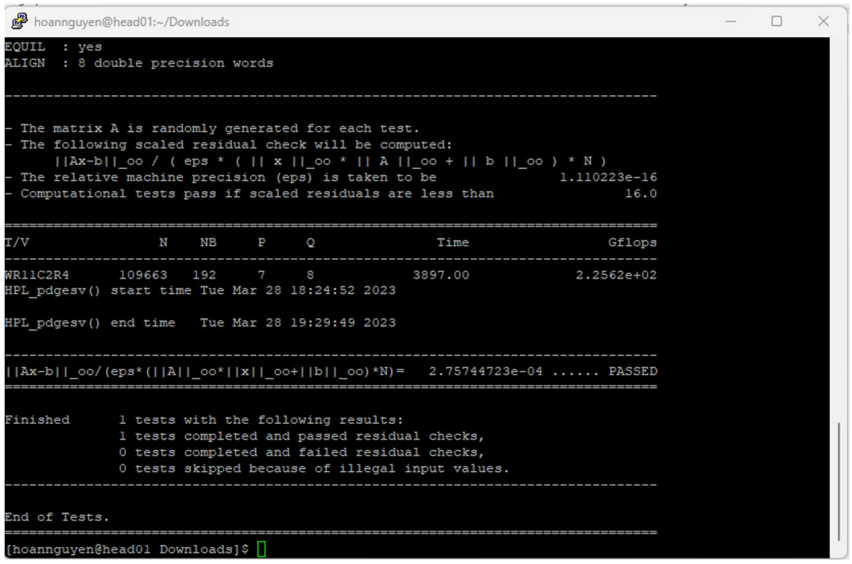
…

Trong đó:

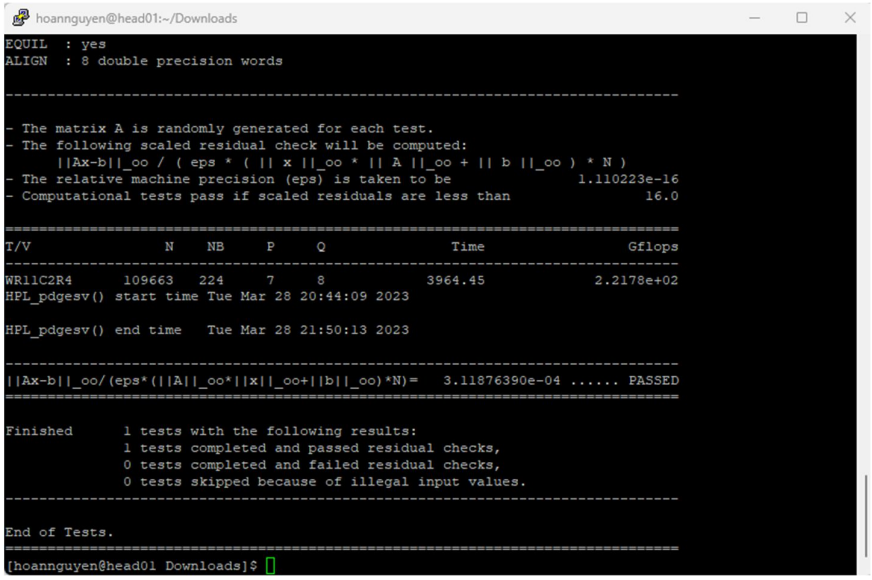
-np: là tổng số luồng xử lý song song mà ta muốn sử dụng

--host: là tên các Node mà chúng ta muốn đẩy lệnh đo đạc lên

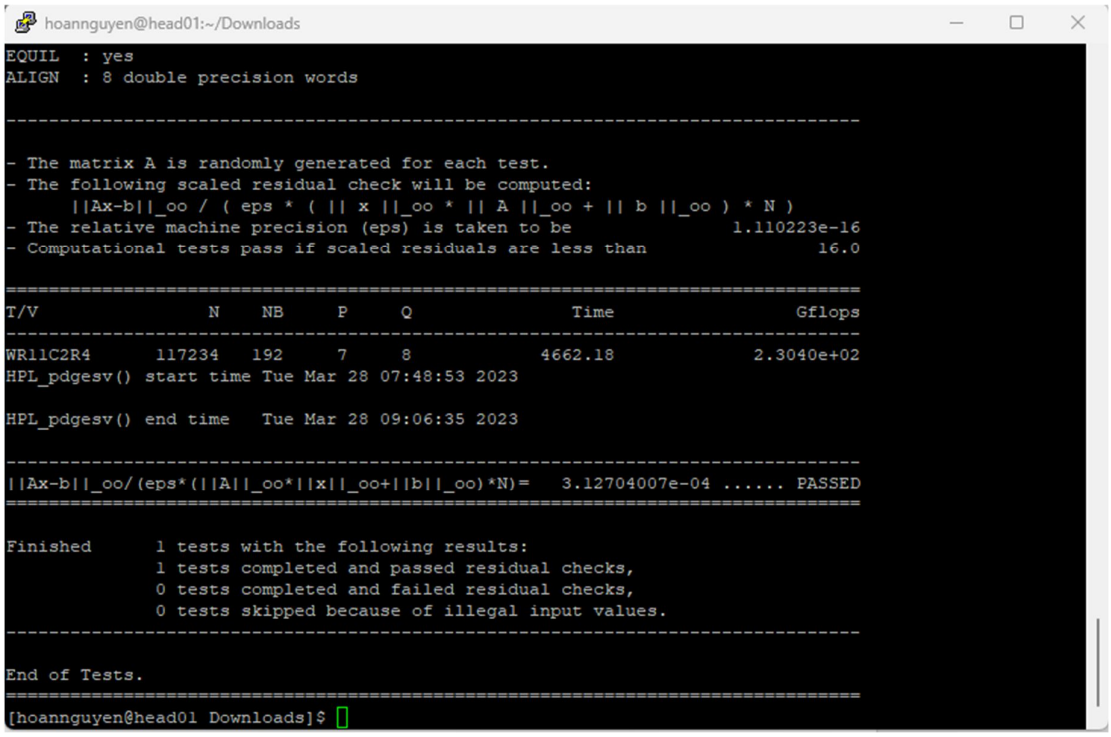
Một vài hình ảnh kết quả của việc đo đạc như sau:



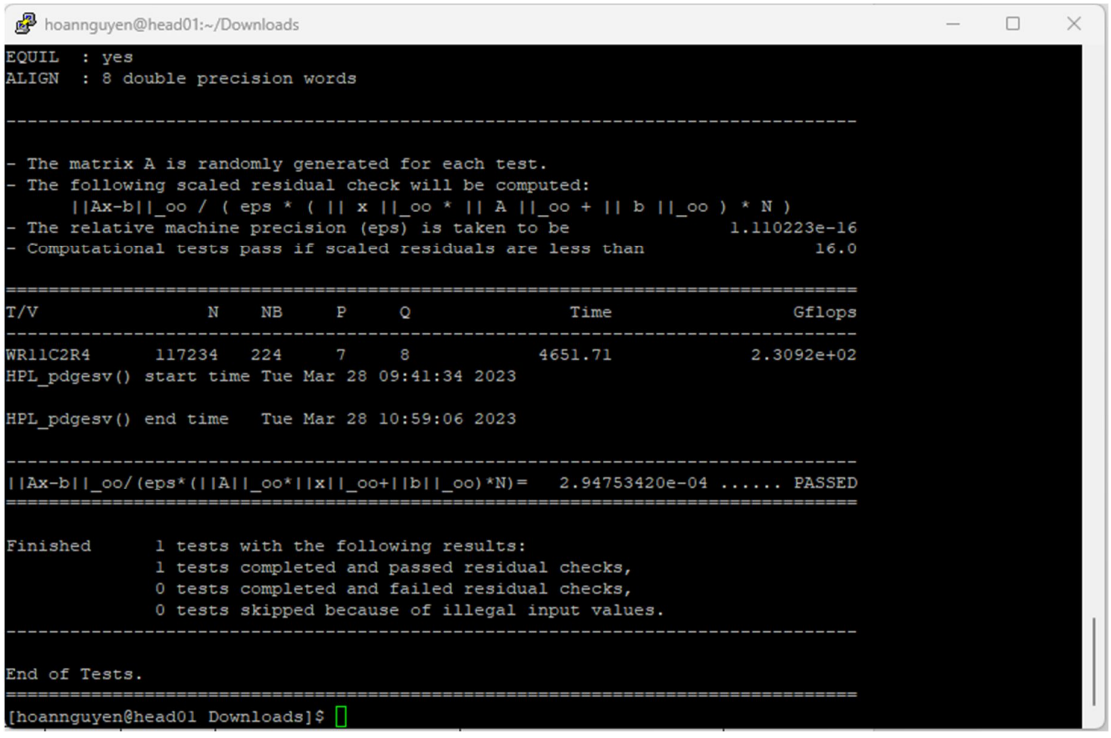
Hình 3.2 7. Đo trên 1 Node với N tính theo 70% bộ nhớ và NB =192



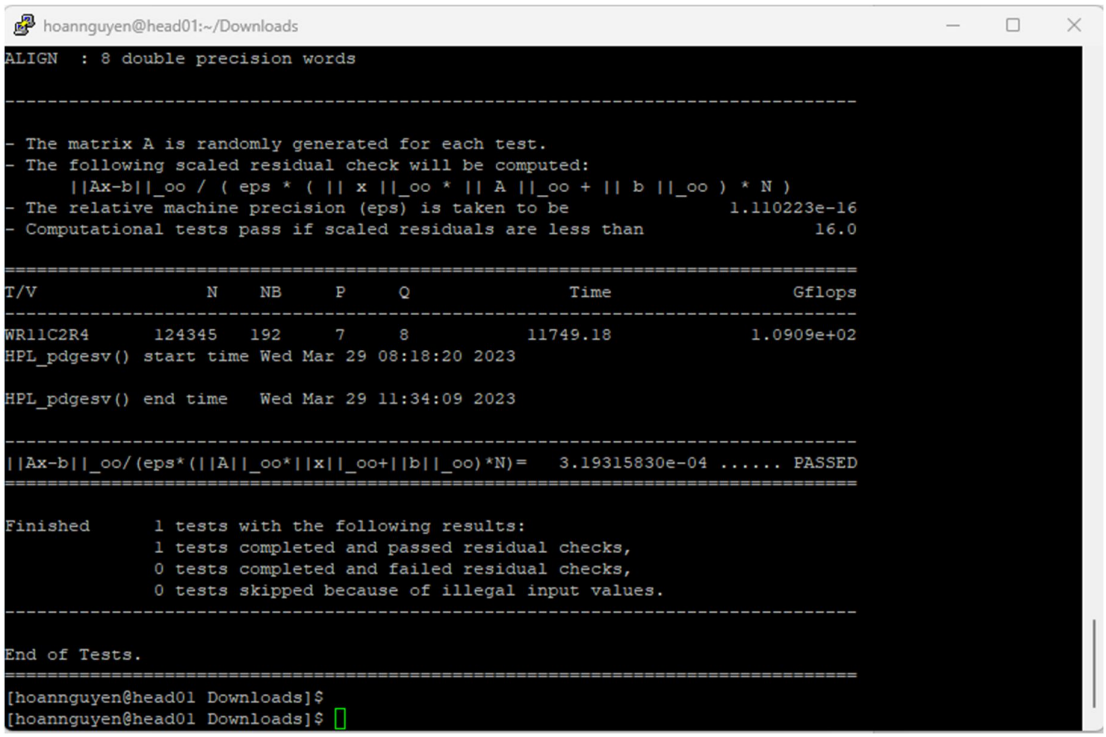
Hình 3.2 8. Đo trên 1 Node với N tính theo 70% bộ nhớ và NB =224



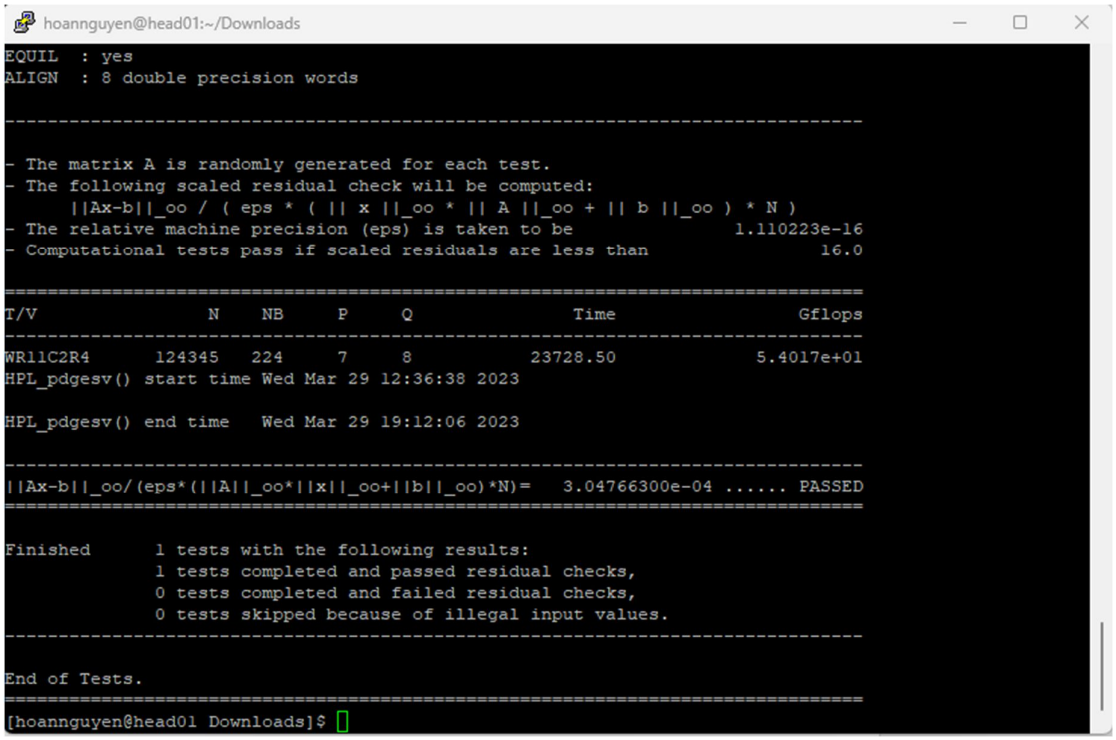
Hình 3.2 9. Đo trên 1 Node với N tính theo 80% bộ nhớ và NB =192



Hình 3.2 10. Đo trên 1 Node với N tính theo 80% bộ nhớ và NB =224



Hình 3.2 11. Đo trên 1 Node với N tính theo 90% bộ nhớ và NB =192



Hình 3.2 12. Đo trên 1 Node với N tính theo 90% bộ nhớ và NB =224

Bảng tổng hợp các kết quả đo được trên 1 Node CPU như sau:

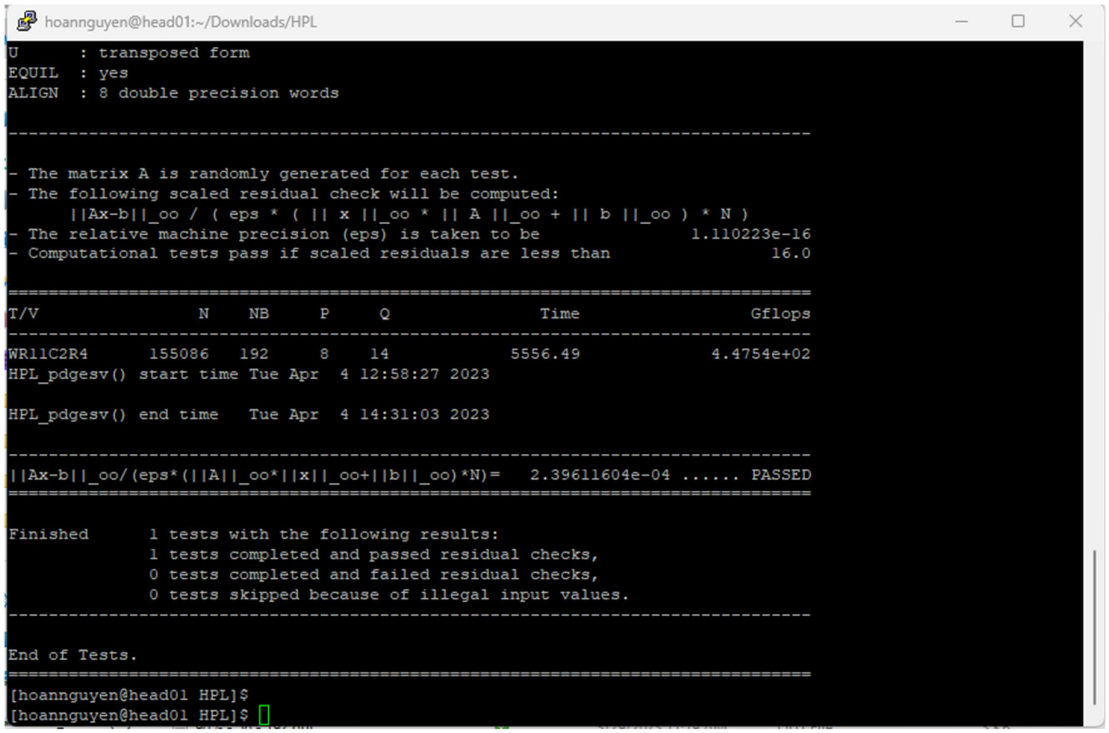
Bảng 3.2 4. Tổng hợp kết quả đo được trên 1 Node CPU

Số Node	Số Core (bộ xử lý)	N	P	Q	NB	Tổng thời gian đo đạc (hh:mm:ss)	Kết quả (Gflops)
1	56	109.663 (70% mem)	7	8	100	01:14:07	197,69
1	56	109.663 (70% mem)	7	8	112	01:09:46	210,00
1	56	109.663 (70% mem)	7	8	192	01:04:57	225,62
1	56	109.663 (70% mem)	7	8	224	01:06:04	221,78
1	56	109.663 (70% mem)	7	8	256	01:05:14	224,61
1	56	109.663 (70% mem)	7	8	300	01:06:23	220,71
1	56	117.234 (80% mem)	7	8	100	01:28:50	201,53
1	56	117.234 (80% mem)	7	8	112	01:24:18	212,37
1	56	117.234 (80% mem)	7	8	192	01:17:42	**230,40**
1	56	117.234 (80% mem)	7	8	224	01:17:31	**230,92**
1	56	117.234 (80% mem)	7	8	256	01:20:26	222,56
1	56	117.234 (80% mem)	7	8	300	01:20:11	223,24
1	56	124.345 (90% mem)	7	8	100	01:47:56	197,92
1	56	124.345 (90% mem)	7	8	112	01:42:27	208,51
1	56	124.345 (90% mem)	7	8	192	03:15:49	109,09
1	56	124.345 (90% mem)	7	8	224	06:35:28	54,01
1	56	124.345 (90% mem)	7	8	256	07:01:22	50,69
1	56	124.345 (90% mem)	7	8	300	08:56:56	39,78

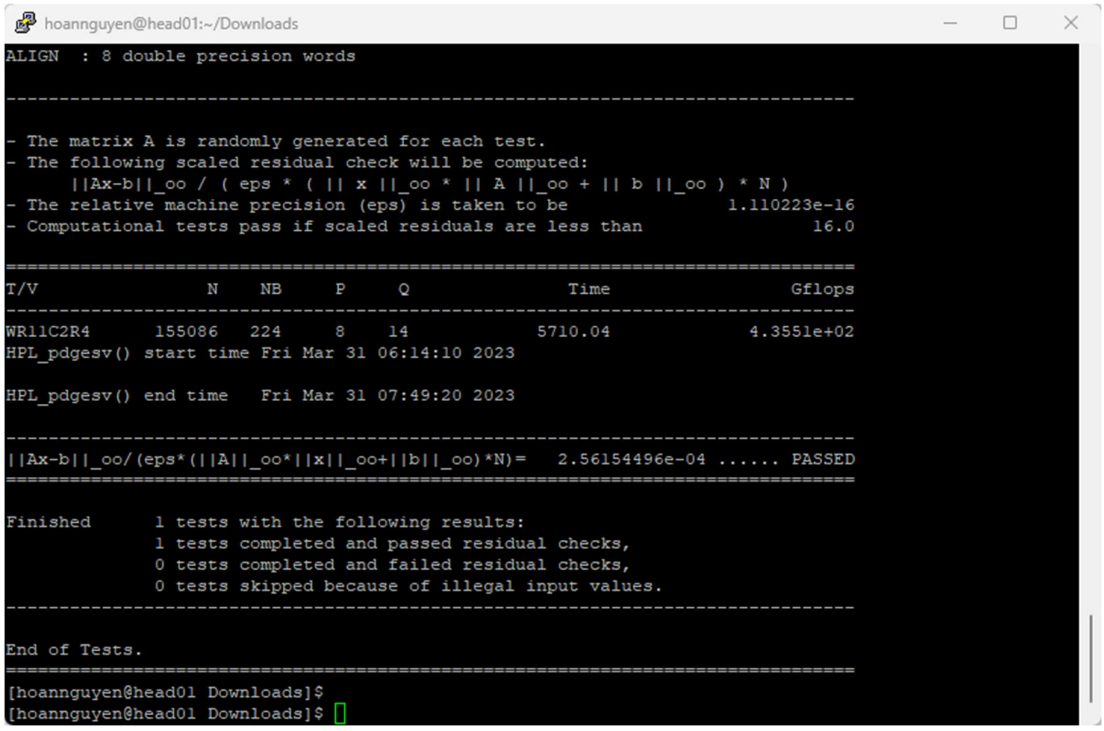
Từ kết quả trên ta thấy :

- Giá trị N tính theo 80% dung lượng bộ nhớ RAM cho kết quả đo tốt nhất

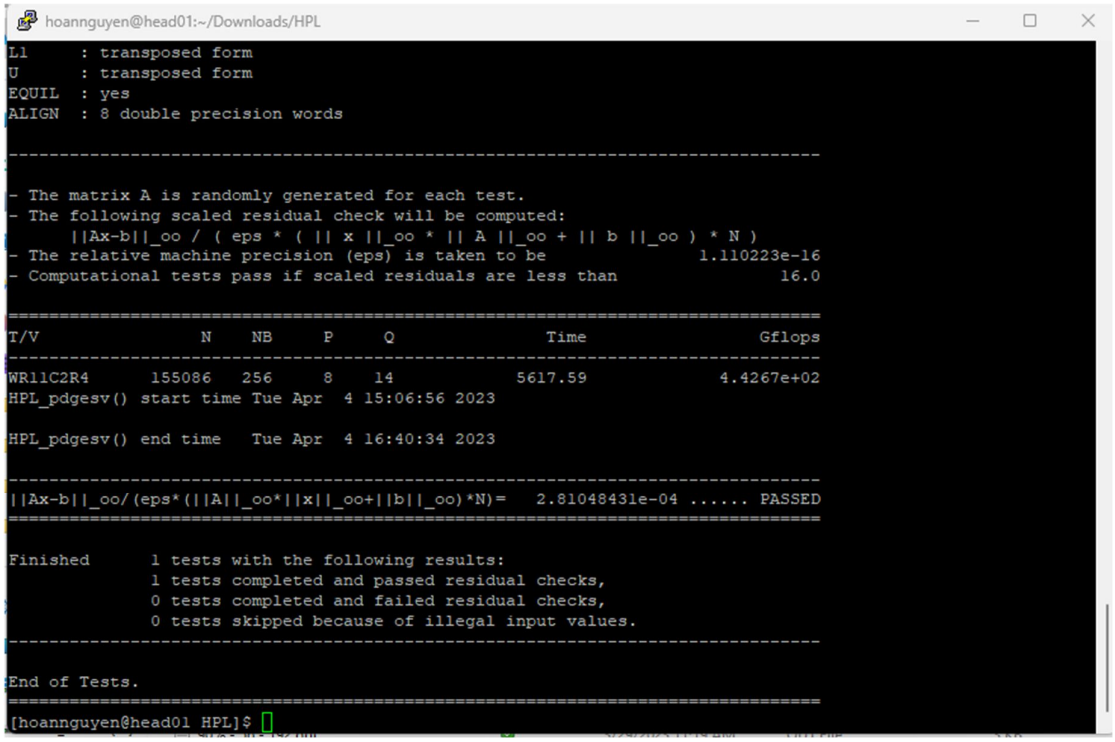
- Giá trị NB trong khoảng 192 đến 256 cho kết quả đo tốt nhất
- Năng lực xử lý đo trên 1 Node là khoảng **230 Gflops**



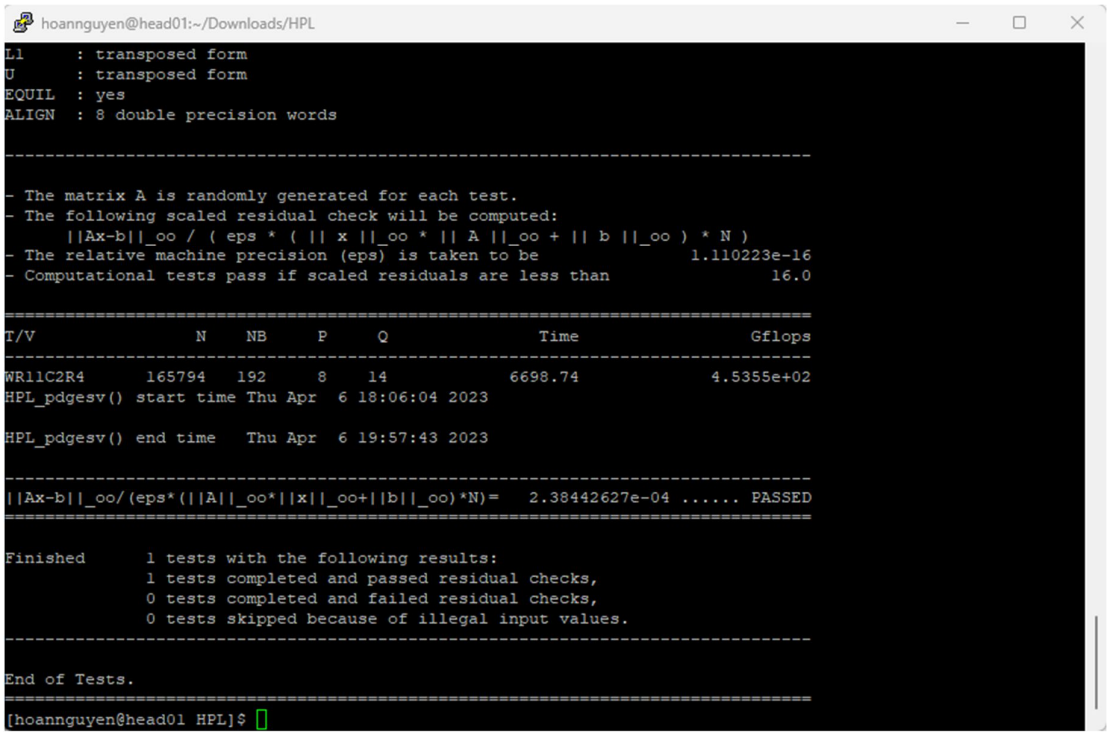
Hình 3.2 13. Đo trên 2 Node với N tính theo 70% bộ nhớ và NB =192



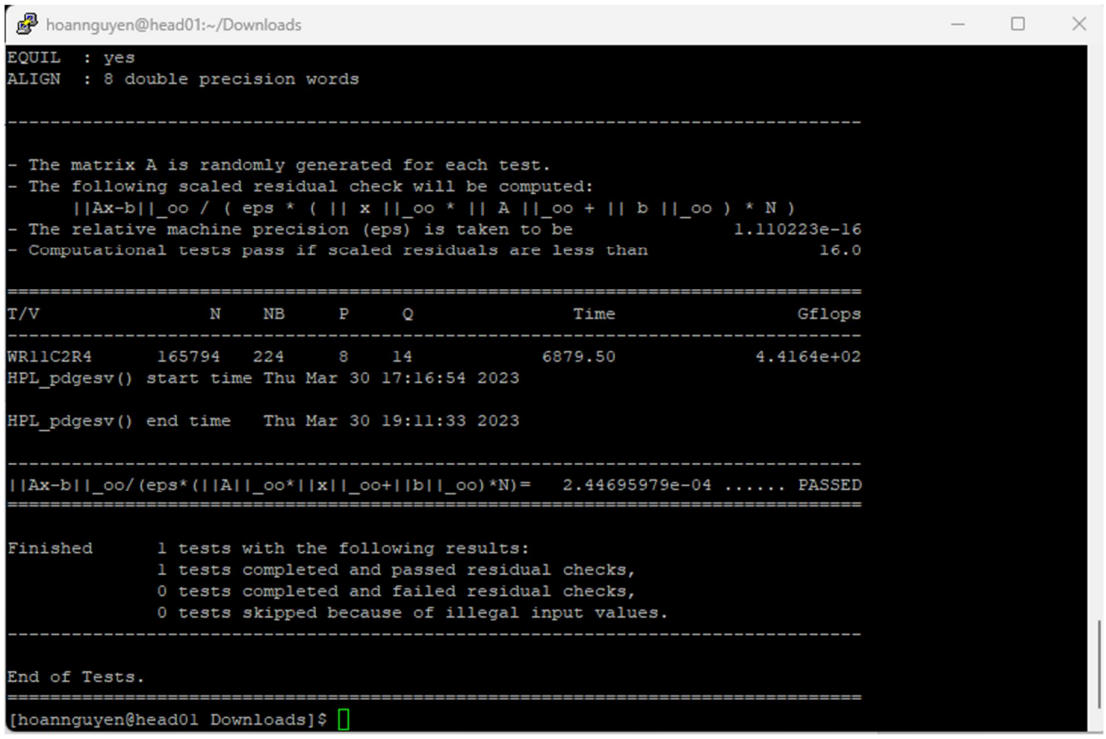
Hình 3.2 14. Đo trên 2 Node với N tính theo 70% bộ nhớ và NB =224



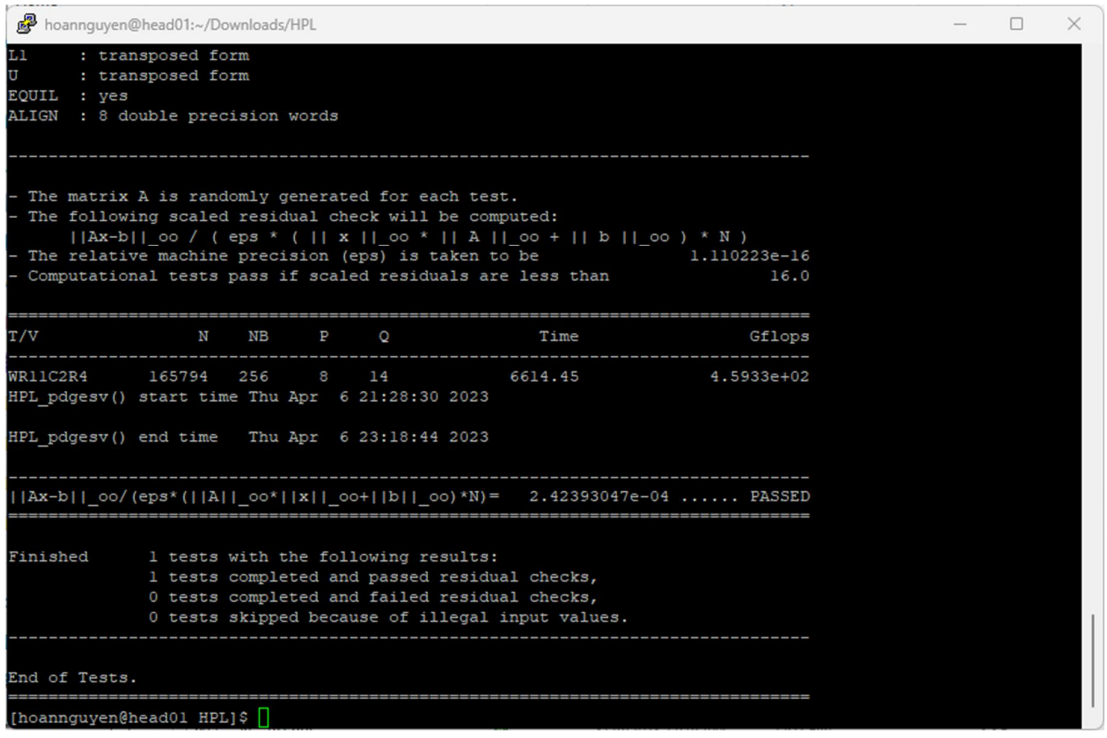
Hình 3.2.15. Đo trên 2 Node với N tính theo 70% bộ nhớ và NB =256



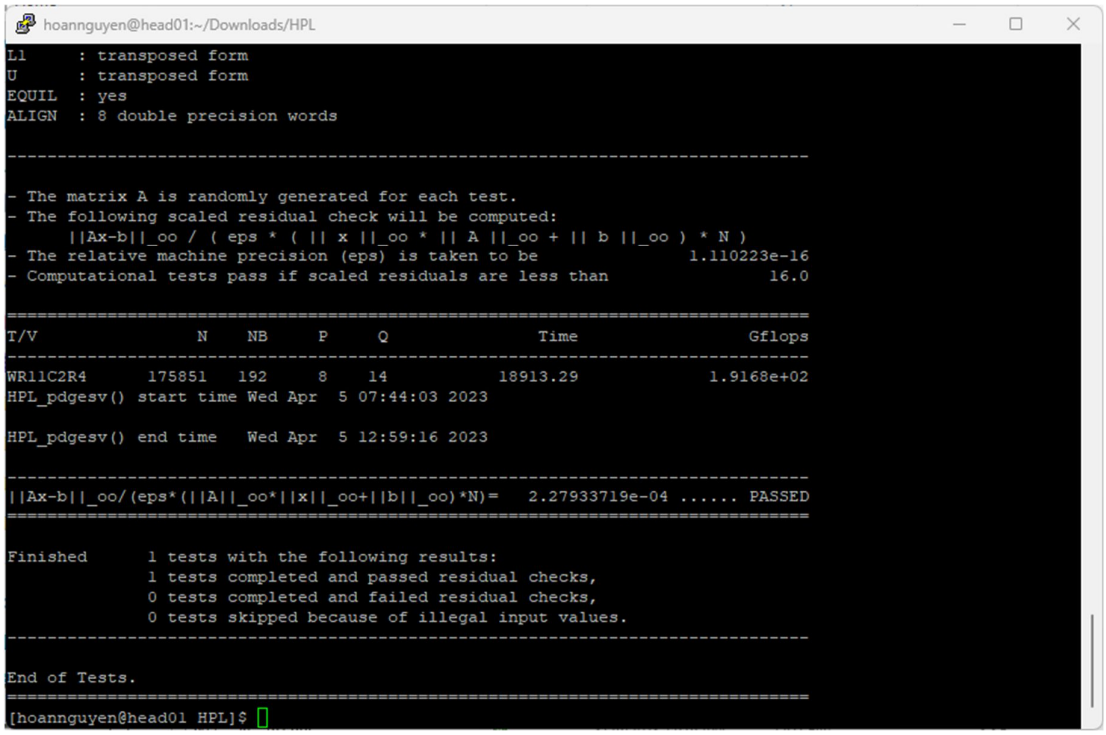
Hình 3.2 16. Đo trên 2 Node với N tính theo 80% bộ nhớ và NB =192



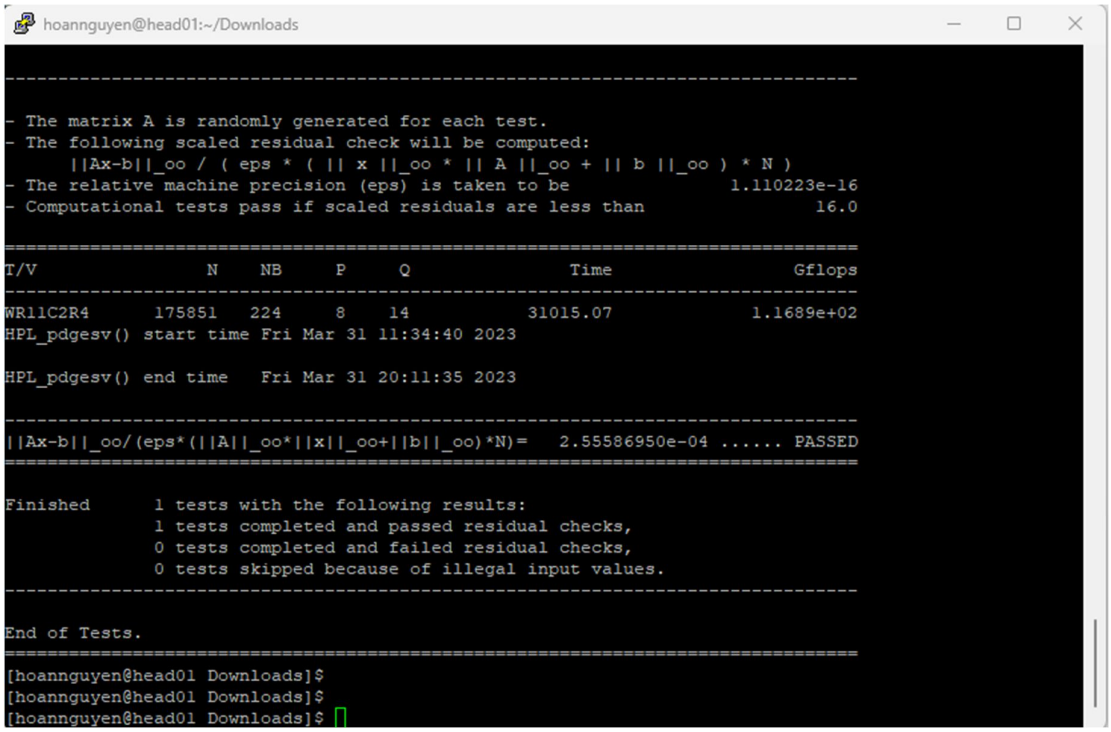
Hình 3.2 17. Đo trên 2 Node với N tính theo 80% bộ nhớ và NB =224



Hình 3.2 18. Đo trên 2 Node với N tính theo 80% bộ nhớ và NB =224



Hình 3.2 19. Đo trên 2 Node với N tính theo 90% bộ nhớ và NB =192



Hình 3.2 20. Đo trên 2 Node với N tính theo 90% bộ nhớ và NB =224

Bảng tổng hợp các kết quả đo được trên 2 Node CPU như sau:

Bảng 3.2 5. Tổng hợp kết quả đo được trên 2 Node CPU

Số Node	Số Core (bộ xử lý)	N	P	Q	NB	Tổng thời gian đo đạc (hh:mm:ss)	Kết quả (Gflops)
2	112	155.086 (70% mem)	8	14	100	01:48:54	380,54
2	112	155.086 (70% mem)	8	14	112	01:42:37	403,84
2	112	155.086 (70% mem)	8	14	192	01:32:36	447,54
2	112	155.086 (70% mem)	8	14	224	01:35:10	435,51
2	112	155.086 (70% mem)	8	14	256	01:33:37	442,67
2	112	155.086 (70% mem)	8	14	300	01:55:27	358,99
2	112	165.794 (80% mem)	8	14	100	02:06:45	399,50
2	112	165.794 (80% mem)	8	14	112	02:18:24	365,87
2	112	165.794 (80% mem)	8	14	192	01:51:29	**453,55**
2	112	165.794 (80% mem)	8	14	224	01:54:39	441,64
2	112	165.794 (80% mem)	8	14	256	01:50:14	**459,33**
2	112	165.794 (80% mem)	8	14	300	01:54 :26	442,50
2	112	175.851 (90% mem)	8	14	100	02:30:19	401,96
2	112	175.851 (90% mem)	8	14	112	03:14:57	309,92
2	112	175.851 (90% mem)	8	14	192	05:15:13	191,68
2	112	175.851 (90% mem)	8	14	224	08:36:55	116,89
2	112	175.851 (90% mem)	8	14	256	09:33:55	105,28

Từ kết quả trên ta thấy :

- Giá trị N tính theo 80% dung lượng bộ nhớ RAM cho kết quả đo tốt nhất

- Giá trị NB trong khoảng 192 đến 256 cho kết quả đo tốt nhất

- Năng lực của 2 Node đo được gấp đôi 1 Node, tiệm cận với giá trị **460 Gflops**.

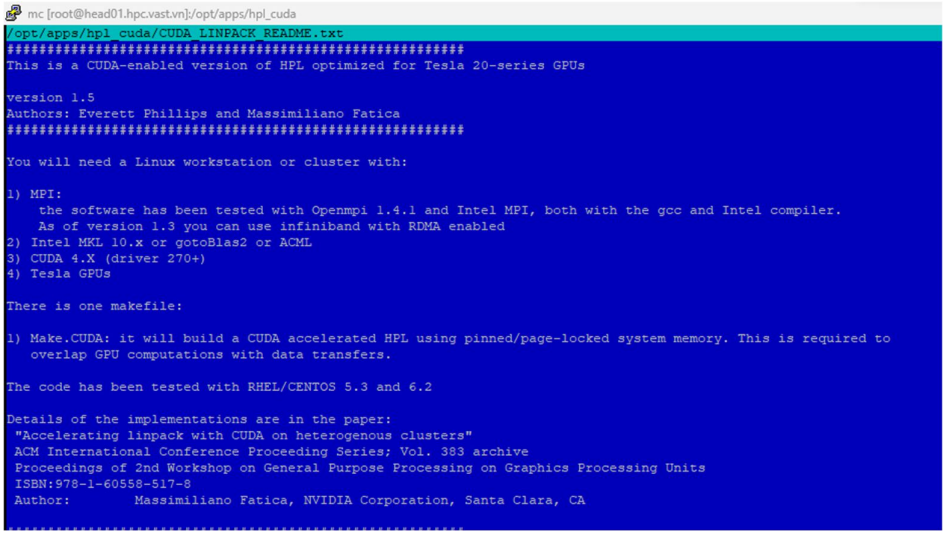
Tương tự, chúng ta thực hiện đo trên các Node có bộ xử lý GPU.

Hệ thống của chúng ta bao gồm 10 Node GPU có cấu hình như sau :

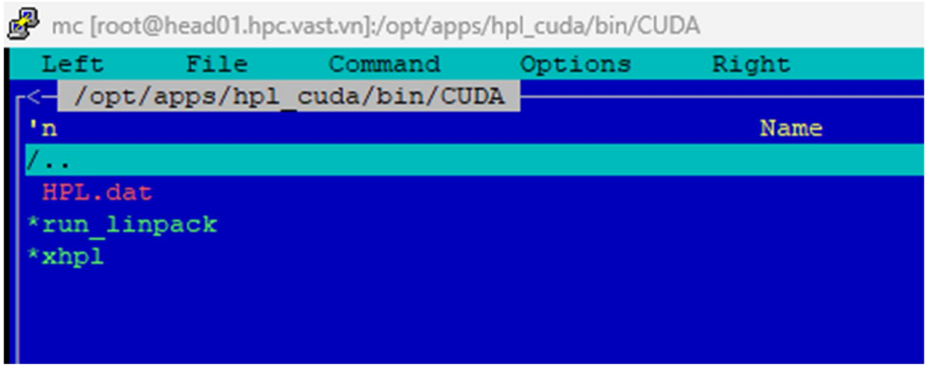
Máy GPU x10:

- Bộ nhớ RAM DDR4 dung lượng 128GB
- 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz
- Bộ xử lý đồ họa NVIDIA Tesla P100

Để thực hiện việc đo năng lực bộ xử lý GPU, ta sử dụng CUDA Linpack được xây dựng để đo năng lực xử lý cho card đồ họa NVDIA Tesla.



Hình 3.2 21. Thông tin về phần mềm CUDA Linpack



Hình 3.2 22. Chương trình CUDA Linpack đã biên dịch trên hệ thống

Để thực hiện đo đạc Node GPU trong hệ thống, ta cần đưa tham số vào 2 file bao gồm:

1. File "HPL.dat": ý nghĩa của các tham số như đã đề cập bên trên

2. File "run_linpack": có các tham số quan trọng như sau

CPU_CORE_PER_GPU: tỷ lệ CPU trên GPU của mỗi Node, trong trường hợp của chúng ta, mỗi Node GPU có 1 card GPU và 56 coré CPU nên con số này là 56

CUDA_DGEMM_SPLIT: tỷ lệ phân bố bài toán DGEMM (Double precision GEneral Matrix Multiplication) cho bộ xử lý GPU. Nó được tính bằng (GPU GFPLOPS)/ (GPU GFPLOPS + CPU GFPLOPS)

Trong trường hợp của chúng ta GPU GFLOPS có giá trị lý thuyết là 4.700 GFLOPS. CPU GFLOPS đo được ở trên là 230 GFLOPS. Do đó ta tính được:

CUDA_DGEMM_SPLIT = 4.700 / (4.700 + 230) = 0.95

Thực hiện phép đo trên hệ thống ta được kết quả đo năng lực của một Node GPU như sau:

- Giá trị N tính theo 80% dung lượng bộ nhớ RAM cho kết quả đo tốt nhất

- Giá trị NB trong khoảng 192 đến 256 cho kết quả đo tốt nhất

- Năng lực xử lý đo trên 1 Node GPU là khoảng **3.750 Gflops** (**3,75 TFLOPS**)

Cuối cùng chúng ta thực hiện phép đo tổng thể trên toàn bộ hệ thống.

Do hệ thống của chúng ta không đồng nhất, gồm cả các Node CPU và GPU nên ta thực hiện phép đo làm 2 phần:

- Đo đạc trên tất cả các Node CPU

- Đo đạc trên tất cả các Node GPU

Trong phần kiểm tra đánh giá hệ thống này, hệ thống sẽ được chạy bài toán Linpack với độ phức tạp khác nhau và với cấu hình khác nhau. Kết quả của các bài toán thử nghiệm được liệt kê trong bảng dưới đây:

TT	Số nút tính toán	Thông số	CPU/GPU	Hiệu suất
1	5	- Bộ nhớ RAM DDR4 dung lượng 128GB - 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz	CPU	- Giá trị N tính theo 80% dung lượng bộ nhớ RAM cho kết quả đo tốt nhất - Giá trị NB trong khoảng 192 đến 256 cho kết quả đo tốt nhất ⇨ Năng lực của 5 Node CPU là khoảng **1.145 Gflops (1,145 TFLOPS)**.
2	10	- Bộ nhớ RAM DDR4 dung lượng 128GB - 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz - Bộ xử lý đồ họa NVIDIA Tesla P100	GPU	- Giá trị N tính theo 80% dung lượng bộ nhớ RAM cho kết quả đo tốt nhất - Giá trị NB trong khoảng 192 đến 256 cho kết quả đo tốt nhất ⇨ Năng lực của 10 Node GPU là khoảng **37.480 Gflops (37,48 TFLOPS)**.
3	15	- 05 máy CPU, cấu hình: + Bộ nhớ RAM DDR4 dung lượng 128GB + 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz - 10 máy GPU, cấu hình: + Bộ nhớ RAM DDR4 dung lượng 128GB + 02 x Bộ xử lý Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz Bộ xử lý đồ họa NVIDIA Tesla P100	Sử dụng đồng thời cả CPU và GPU	⇨ Năng lực của 10 Node GPU + 05 Node CPU là khoảng **1,45 + 37,48 = 38.93 TFLOPS**

➢ Từ kết quả trên cho thấy : Tổng năng lực xử lý toàn hệ thống là:

1,45 + 37,48 = 38.93 TFLOPS

3.3. Đánh giá tốc độ tính toán dựa trên bài toán mẫu

Tính toán song song được thực hiện dựa trên cơ sở các bài toán có thể phân đoạn ra được hay các phép thử lặp đi lặp lại một công thức với các tham số khác nhau.

Nếu thực hiện các phép toán trên chỉ bằng một luồng xử lý, thời gian để nhận được kết quả có thể rất lâu, tính bằng hàng tháng, hàng năm. Khi đó, kỹ thuật tính toán song song giúp các nhà khoa học giảm bớt thời gian chờ đợi trong việc xử lý dữ liệu và sẽ nhận được kết quả nhanh hơn nhiều lần.

Ví dụ ta có phép tính tổng của các số trong một dãy với hàng tỷ tỷ phần tử. Nếu xử lý trên một luồng, ta sẽ mất thời gian để thực hiện phép cộng hàng tỷ tỷ lần. Nhưng nếu ta chia đôi dãy số ra và thực hiện phép tính trên hai nửa bằng hai tiến trình khác nhau, sau đó tổng hợp lại thì thời gian giảm xuống gần như một nửa. Cứ như thế, số luồng xử lý song song càng nhiều thì thời gian có được kết quả càng được rút ngắn.

Một ví dụ khác, ta xây dựng được một mô hình dự báo nhiệt độ trong ngày và muốn đánh giá độ chính xác của mô hình đó. Dữ liệu để đánh giá là dữ liệu khí tượng của 10 năm trước, bao gồm các thông số về khí tượng đã đo được trên các thiết bị quan trắc, theo chu kỳ 5 phút.

Khi đó ta sẽ phải làm rất nhiều phép thử lặp đi lặp lại với cùng một công thức, nhưng tham số đầu vào để tính toán khác nhau. Sau đó tổng hợp lại kết quả cuối cùng để kết luận.

Thay vì thực hiện hàng triệu, hàng tỷ phép tính trên một luồng xử lý, ta có thể chia bộ tham số ra làm nhiều phần và đưa lên các luồng xử lý khác nhau để tính toán. Mỗi luồng đảm nhận một phần của bài toán. Cuối cùng tổng hợp lại. Từ đó thời gian để xử lý toàn bộ bài toán giảm đi nhiều lần.

Trong phần tiếp theo, ta sẽ xây dựng một chương trình tính toán song song với các đặc điểm như sau[9][12]:

- Viết bằng ngôn ngữ C.
- Sử dụng thư viện MPI để phân bổ các phần của bài toán chạy trên các Node khác nhau [22].
- Đầu vào là một mảng A chứa các số thập phân từ 1 đến N.
- Quá trình tính toán giả lập việc xử lý từng phần tử trong mảng A (tham số đầu vào). Kết quả trả về sau khi tính xử lý sẽ được tính tổng lại.
- Tư duy của việc tính toán song song ở đây là các phép toán thực hiện trên từng phần tử của mảng A là độc lập. Do đó chúng ta có thể phân bố nó trên các luồng tính toán khác nhau rồi lấy tổng các kết quả cuối cùng lại.
- Chúng ta sẽ đo đạc thời gian chương trình cần để xử lý bài toán khi sử dụng số luồng tính toán khác nhau.

Chương trình của chúng ta như sau:

```
==========================================
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <unistd.h>
  // Kích thước của mảng A
#define n 500
  // Định nghĩa mảng A với N phần tử thập phân
float a[n];
  // Định nghĩa mảng A2 dùng tạm cho các tiến trình phụ
float a2[10000];
int main(int argc, char* argv[])
{
   // Khai báo biến để lưu thời gian bắt đầu và kết thúc
   double time_diff;
   time_t time_start, time_end;
   int pid, np,
      elements_per_process,
      n_elements_recieved;
   // np là biến lưu trữ số luồng xử lý
   // pid là biến lưu trữ ID của luồng xử lý. Luồng xử lý chính sẽ có ID là 0.
   MPI_Status status;
   // Tạo tiến trình song song
   MPI_Init(&argc, &argv);
   // Kiểm tra số lượng tiến trình và ID của các tiến trình
   MPI_Comm_rank(MPI_COMM_WORLD, &pid);
   MPI_Comm_size(MPI_COMM_WORLD, &np);
   // Nếu là tiến trình chính, xử lý các lệnh liên quan như khởi tạo tham số, xử
lý một phần công việc, phân bổ công việc, thu nhận kết quả, tổng hợp kết quả
   if (pid == 0) {
        // Khởi tạo mảng ngẫu nhiên A
        srand(time(NULL));
        for(long i=0; i<n; i++)
        {
```

```
            // Tạo mảng ngẫu nhiên A gồm các phần tử thập phân có giá trị từ
0 đến 100
            a[i] = (float)rand()/(float)(RAND_MAX/100);
        }
        printf("\n");
        printf("Number of array element = %d\n", n);
        // Bắt đầu tính thời gian xử lý
        time_start = time(NULL);
        // Phân chia công việc cho các tiến trình phụ
        long index, i;
        elements_per_process = n / np;
        // Kiểm tra nếu có nhiều hơn 1 tiến trình thì phân chia công việc
        if (np > 1) {
                    // Phân chia công việc
                    for (i = 1; i < np - 1; i++) {
                    index = i * elements_per_process;
                    MPI_Send(&elements_per_process,
                                1, MPI_DOUBLE, i, 0,
                                MPI_COMM_WORLD);
                    MPI_Send(&a[index],
                                elements_per_process,
                                MPI_DOUBLE, i, 0,
                                MPI_COMM_WORLD);
                     }
                    // với tiến trình cuối, nếu công việc còn dư (không chia đều)
thì phân bổ nốt
                    index = i * elements_per_process;
                    long elements_left = n - index;
                    MPI_Send(&elements_left,
                            1, MPI_DOUBLE,
                            i, 0,
                            MPI_COMM_WORLD);
                    MPI_Send(&a[index],
                            elements_left,
                            MPI_DOUBLE, i, 0,
                            MPI_COMM_WORLD);
        }
```

```
        // Công việc của tiến trình chính
        double sum = 0;
        for (i = 0; i < elements_per_process; i++)
        {
                // Hàm ScienceCaculation giả lập việc xử lý dữ liệu của các phần
tử trong mảng A
                double result = ScienceCaculation(a[i]);

                // Sau khi xử lý xong thì tính tổng
                        sum += result;
        }
        // Hiển thị tổng của tiến trình chính
        printf("Partial sum of process 00 is : %f\n", sum);
                // Thu nhận kết quả của các tiến trình phụ
        double tmp;
        for (i = 1; i < np; i++) {
                        MPI_Recv(&tmp, 1, MPI_DOUBLE,
                        MPI_ANY_SOURCE, 0,
                        MPI_COMM_WORLD,
                        &status);
                        int sender = status.MPI_SOURCE;
                // Hiển thị kết quả của các tiến trình phụ
                printf("Partial sum of process %02d is : %f\n", i, tmp);
                        sum += tmp;
        }
        // Kết thúc việc tính thời gian xử lý
        time_end = time(NULL);
        // Hiển thị kết quả cuối cùng sau khi tổng hợp
        printf("\n=> Sum of array is : %f\n\n", sum);
        // Hiển thị thời gian bắt đầu và kết thúc
        printf("=> Start time: %s", ctime(&time_start));
        printf("=> End time: %s", ctime(&time_end));
        // Tính toán và hiển thị tổng thời gian xử lý
        time_diff = difftime(time_end, time_start);
        int minutes = (time_diff)/60;
        int seconds = (time_diff - (minutes*60));
```

```
        printf("\n=> Execution time = %d minutes %d seconds\n", minutes, seconds);
    }
    // Các tiến trình phụ nhận nhiệm vụ từ tiến trình chính, xử lý là trả lại kết quả
    else {
        // Nhận nhiệm vụ từ tiến trình chính
        MPI_Recv(&n_elements_recieved,
                1, MPI_DOUBLE, 0, 0,
                MPI_COMM_WORLD,
                &status);
        // Lưu trữ dữ liệu được giao vào mảng phụ A2
        MPI_Recv(&a2, n_elements_recieved,
                MPI_DOUBLE, 0, 0,
                MPI_COMM_WORLD,
                &status);
        // Xử lý dữ liệu
        double partial_sum = 0;
        for (long i = 0; i < n_elements_recieved; i++)
        {
                    // Hàm ScienceCaculation giả lập việc xử lý dữ liệu của các phần tử trong mảng A
                double result = ScienceCaculation(a[i]);

                // Sau khi xử lý xong thì tính tổng
                    partial_sum += result;
        }
        // Trả lại kết quả cho tiến trình chính
        MPI_Send(&partial_sum, 1, MPI_DOUBLE,
                0, 0, MPI_COMM_WORLD);
    }
     // Kết thúc việc xử lý và kết thúc chương trình
    MPI_Finalize();
     return 0;
}
==========================================
```

Sau khi đã xây dựng được chương trình, ta chạy lệnh dịch chương trình bằng trình dịch C của MPI (mpicc)

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ mpicc mpi_sum_final.c -o mpi_sum_final
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$
```

Hình 3.3 1. Biên dịch chương trình

Tiến hành chạy chương trình với các trường hợp xử dụng số luống tính toán song song khác nhau:

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 25352.008523

=> Sum of array is : 25352.008523

=> Start time: Sun Apr  9 09:06:24 2023
=> End time: Sun Apr  9 09:08:54 2023

=> Execution time = 2 minutes 30 seconds
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$
```

Hình 3.3 2. Chạy chương trình chỉ với 1 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 2 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 11908.187214
Partial sum of process 01 is : 12856.158870

=> Sum of array is : 24764.346083

=> Start time: Sun Apr  9 09:09:21 2023
=> End time: Sun Apr  9 09:10:36 2023

=> Execution time = 1 minutes 15 seconds
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$
```

Hình 3.3 3. Chạy chương trình với 2 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 4 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 6223.712117
Partial sum of process 01 is : 6422.230754
Partial sum of process 02 is : 6088.303507
Partial sum of process 03 is : 6237.764449

=> Sum of array is : 24972.010827

=> Start time: Sun Apr  9 09:11:48 2023
=> End time: Sun Apr  9 09:12:26 2023

=> Execution time = 0 minutes 38 seconds
[hoannguyen@head01 TEST_C_SUM]$
```

Hình 3.3 4. Chạy chương trình với 4 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 6 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 4095.472495
Partial sum of process 01 is : 4341.234050
Partial sum of process 02 is : 4336.107818
Partial sum of process 03 is : 4080.419673
Partial sum of process 04 is : 4432.550572
Partial sum of process 05 is : 4182.342198

=> Sum of array is : 25468.126806

=> Start time: Sun Apr  9 09:13:30 2023
=> End time: Sun Apr  9 09:13:56 2023

=> Execution time = 0 minutes 26 seconds
[hoannguyen@head01 TEST_C_SUM]$
```

Hình 3.3 5. Chạy chương trình với 6 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 8 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 3002.128123
Partial sum of process 01 is : 3043.999907
Partial sum of process 02 is : 3182.069477
Partial sum of process 03 is : 2917.210823
Partial sum of process 04 is : 3104.581284
Partial sum of process 05 is : 3394.120894
Partial sum of process 06 is : 2923.491926
Partial sum of process 07 is : 3552.924749

=> Sum of array is : 25120.527184

=> Start time: Sun Apr  9 09:14:32 2023
=> End time: Sun Apr  9 09:14:52 2023
```

Hình 3.3 6. Chạy chương trình với 8 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 10 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 2471.673846
Partial sum of process 01 is : 2757.649288
Partial sum of process 02 is : 2428.303288
Partial sum of process 03 is : 2555.372956
Partial sum of process 04 is : 2595.618163
Partial sum of process 05 is : 2583.294558
Partial sum of process 06 is : 2088.656870
Partial sum of process 07 is : 2433.771359
Partial sum of process 08 is : 2275.873406
Partial sum of process 09 is : 2378.626528
```

Hình 3.3 7. Chạy chương trình với 10 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 12 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 2249.086500
Partial sum of process 01 is : 1974.103337
Partial sum of process 02 is : 1773.663071
Partial sum of process 03 is : 1987.283656
Partial sum of process 04 is : 2279.980721
Partial sum of process 05 is : 2191.658942
Partial sum of process 06 is : 1844.108001
Partial sum of process 07 is : 1671.171274
Partial sum of process 08 is : 1984.646743
Partial sum of process 09 is : 1916.458462
Partial sum of process 10 is : 1804.171233
Partial sum of process 11 is : 2503.719283
```

Hình 3.3 8. Chạy chương trình với 12 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 14 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 1472.344580
Partial sum of process 01 is : 1804.501503
Partial sum of process 02 is : 1751.447763
Partial sum of process 03 is : 1422.860222
Partial sum of process 04 is : 1788.462417
Partial sum of process 05 is : 1915.711220
Partial sum of process 06 is : 1725.861900
Partial sum of process 07 is : 1618.387060
Partial sum of process 08 is : 1869.445060
Partial sum of process 09 is : 1769.821010
Partial sum of process 10 is : 1785.204344
Partial sum of process 11 is : 1579.030295
Partial sum of process 12 is : 1777.034941
Partial sum of process 13 is : 2257.522207
```

Hình 3.3 9. Chạy chương trình với 14 luồng xử lý

```
[hoannguyen@head01 TEST_C_SUM]$
[hoannguyen@head01 TEST_C_SUM]$ mpirun -np 16 ./mpi_sum_final

Number of array element = 500
Partial sum of process 00 is : 1534.722153
Partial sum of process 01 is : 1613.123228
Partial sum of process 02 is : 1221.520806
Partial sum of process 03 is : 1803.647117
Partial sum of process 04 is : 1486.557848
Partial sum of process 05 is : 1759.964611
Partial sum of process 06 is : 1489.245840
Partial sum of process 07 is : 1433.579573
Partial sum of process 08 is : 1212.487826
Partial sum of process 09 is : 1422.308760
Partial sum of process 10 is : 1272.341550
Partial sum of process 11 is : 1453.795484
Partial sum of process 12 is : 1757.375852
Partial sum of process 13 is : 1487.646823
Partial sum of process 14 is : 1405.374985
Partial sum of process 15 is : 1721.049891
```

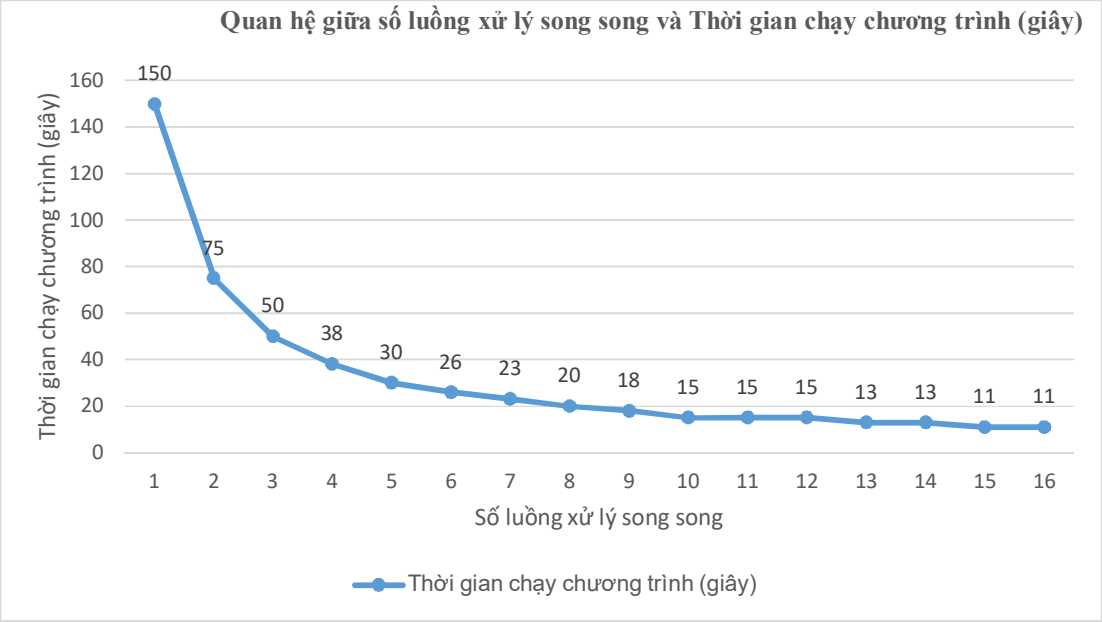
Hình 3.3 10. Chạy chương trình với 16 luồng xử lý

Ta lập được bảng tổng hợp kết quả như sau:

Bảng 3.1. Quan hệ giữa số luồng xử lý và thời gian chạy chương trình

Số luồng xử lý song song	Thời gian chạy chương trình (giây)
1	150
2	75
3	50
4	38
5	30
6	26
7	23
8	20
9	18
10	15
11	15
12	15
13	13
14	13
15	11
16	11

Ta có biểu đồ như sau:



Hình 3.3 11. Quan hệ giữa số luồng xử lý và thời gian chạy chương trình

Ta có thể thấy thời gian xử lý giảm đi gần N lần khi số luồng xử lý tăng lên gấp N lần.

Bảng 3.3 1. Danh sách một vài bài toán khoa học thực tế tính toán trên hệ thống HPC và thống kê hiệu suất sử dụng CPU-GPU

TT	User	Tên người dùng	Đơn vị	Vấn đề tính toán	Song song/Tuần tự	Tên chương trình (công cụ tính toán)	Tài nguyên sử dụng	Thời gian tính toán (giờ)	**Năm**	**Hiệu suất CPU**	**Hiệu suất GPU**
									2022	16.84	3.01
1	nddung	Nguyễn Đức Dũng	Viện CNTT	Training mô hình trí tuệ nhân tạo	Tuần tự	python	CPU: 1 GPU:1 MEM:16GB	100.67		12.39	0.00
										18.70	4.59
2	nvtrang	Nguyễn Văn Tráng	Viện Kỹ thuật Nhiệt đới	Bài toán sử dụng phương pháp phiếm hàm mật độ để nghiên cứu cấu trúc và tính chất các hợp chất bán dân hữu cơ dị vòng ngưng tụ chứa O, B, N, định hướng làm vật liệu phát quang trong OLED	Song song	Gausian	CPU:24 MEM:16GB	6.62		20.11	0.00
										26.81	26.52
										45.16	43.38
										39.45	34.86
3	nthong	Nguyễn Tuấn Hồng	Trung tâm Phát triển công nghệ cao	Tính toán mô phỏng Density Theory Functional (DFT) cho quá trình tách nước (H2O thành H2/O2) với các vật liệu nano nền Fe, Ni oxit trên graphene. Tính toán mô phỏng trên cấu trúc MoS2	Song song	Quantum Espresso	CPU:48 MEM:50GB	53.33		32.84	28.93
										29.60	2.24
										26.48	6.90
										27.21	32.14
4	pmquan	Phạm Minh Quân	Viện Hóa học các hợp chất TN	Mô phỏng lắp ghép phân tử	Tuần tự	Gromacs	CPU:1 MEM:1GB	6.47		35.20	23.02
										44.90	30.65
										53.70	24.21
5	nnquynh	Nguyễn Ngọc Quỳnh	Viện Khoa học và Kỹ Thuật Hạt nhân	Chạy chương trình mô phỏng MCNP, PHITS	Song song	python	CPU:20 MEM:90GB	1.96		84.58	44.48
										64.25	30.87
										45.66	26.44
										86.72	22.28
6	nttmai	Nguyễn Thị Thanh Mai	Viện CNTT	Training mô hình trí tuệ nhân tạo	Tuần tự	python	CPU:1 GPU:1 MEM:12GB	207.79		94.50	23.69
										85.02	22.66

3.4. Tổng kết kết quả đánh giá hệ thống

Như đã đo đạc ở trên, với một Node CPU có 2 CPU x Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz và 128GB RAM, ta đo được năng lực của trong khoảng 230 Gflops.

Với hệ thống của chúng ta có 5 Node CPU và 10 Node GPU cùng cấu hình 2 CPU x Intel(R) Xeon(R) Gold 6132 CPU @ 2.60GHz và 128GB RAM, ta có thể ước lượng năng lực CPU vào khoảng 230x15 = 3,45 Tflops; Năng lực của 5 Node CPU là khoảng 1.145 Gflops (1,145 TFLOPS); Năng lực của 10 Node GPU là khoảng 37.480 Gflops (37,48 TFLOPS); Tổng năng lực xử lý toàn hệ thống là: 1,45 + 37,48 = 38.93 TFLOPS.

Thông qua bài toán mẫu, ta hình dung ra được các ứng dụng, giải pháp và hiệu quả của hệ thống. Bài toán chưa tính đến các yếu tố khác như dung lượng bộ nhớ RAM yêu cầu khi chạy đa luồng, băng thông yêu cầu để trao đổi bản tin giữa các luồng. Tuy nhiên chúng ta có thể thấy được hiệu quả thực sự của hệ thống trong việc tăng tốc độ tính toán cho các bài toán khoa học.

Hiện nay trên thế giới đã có hàng nghìn hệ thống tính toán hiệu năng cao với năng lực tới hàng trăm, hàng nghìn Peta flops (1 Peta = 1.000 Tera = 1.000.000 Giga). Tuy nhiên, trong điều kiện học tập và nghiên cứu hạn chế và kinh phí, ta hoàn toàn có thể tự xây dựng cho mình một hệ thống tính toán song song bằng cách kết hợp năng lực xử lý của nhiều máy chủ cũ hoặc nhiều máy tính cá nhân bình thường. Thông qua phần mềm Rocks Cluster, các máy chủ và máy tính cá nhân riêng lẻ sẽ được phối hợp với nhau để tạo nên một hệ thống chung, chia sẻ năng lực tính toán, giúp đem lại kết quả nhanh hơn nhiều lần cho các bài toán khoa học.

KẾT LUẬN VÀ KIẾN NGHỊ

KẾT LUẬN

Ngày nay, việc sử dụng các hệ thống tính toán hiệu năng cao trong các ngành khoa học và kỹ thuật đã và đang thay đổi cơ bản tiến trình nghiên cứu khoa học. Các ngành khoa học cơ bản trước đây dần đã chuyển đổi với thành phần không thể thiếu là "tính toán" như "sinh học tính toán", "hoá học tính toán", "vật lý tính toán", "vật liệu tính toán", "cơ học tính toán", "địa vật lý tính toán"... Các ngành khoa học này có điểm chung là xử lý thông tin, phân tích và dự báo kết quả bằng tính toán, mô phỏng sử dụng nền tảng tính toán hiệu năng cao. Trong thời đại Cách mạng công nghiệp 4.0, các trung tâm tính toán hiệu năng cao cũng là hạ tầng số để triển khai các nền tảng xử lý dữ liệu lớn (Big Data), trí tuệ nhân tạo (AI) ứng dụng trong tất cả các lĩnh vực của đời sống.

Ta có thể thấy các hệ thống tính toán hiệu năng cao đóng vai trò rất lớn trong cách lĩnh vực nghiên cứu khoa học hiện nay.

Thông qua nội dung đã thực hiện trong luận văn này, chúng ta nắm được các nội dung sau:

- Hiểu khái quát về các hệ thống tính toán hiệu năng cao và tầm quan trọng của chúng trong các lĩnh vực nghiên cứu khoa học.

- Hiểu được mô hình chung của các hệ thống tính toán hiệu năng cao, các khối chức năng cũng như sự liên quan giữa chúng.

- Hiểu được vai trò của các khối chức năng.

- Có thêm kiến thức về phần mềm Rocks Cluster cùng các chức năng của nó. Đây là một phần mềm tính toán hiệu năng cao mã nguồn mở, dễ xây dựng và vận hành, thuận lợi cho việc triển khai và ứng dụng trong môi trường học tập, nghiên cứu với mức đầu tư thấp.

Bên cạnh đó, luận văn còn giới thiệu cụ thể việc triển khai các công cụ đánh giá năng lực hệ thống, từ đó có thể đem đến cho người đọc những hiểu biết sâu hơn nhằm:

- Tự xây dựng được công cụ để đánh giá năng lực hệ thống tính toán hiệu năng cao của mình.

- Hình dung ra được ứng dụng cụ thể của các hệ thống tính toán hiệu năng cao thông qua bài toán mẫu.

- Hình dung ra được hiệu quả của các hệ thống tính toán hiệu năng cao thông qua việc chạy bài toán mẫu và đo đạc thời gian xử lý bài toán trong các điều kiện tính toán khác nhau.

KIẾN NGHỊ

Do khả năng và thời gian hạn chế nên luận văn chưa đề cập được một số nội dung như:

- Chưa có điều kiện để so sánh Rock Cluster với các bộ phần mềm tính toán hiệu năng cao khác.

- Chưa đề cập đến việc đánh giá năng lực xử lý của các card đồ họa trên hệ thống GPU.

- Bài toán mẫu đã thể hiện được ứng dụng cũng như hiệu quả của hệ thống tính toán hiệu năng cao. Tuy nhiên sẽ thực tế hơn nếu nó gắn với một bài toán khoa học cụ thể, và có kết quả cụ thể gắn với thực tế đời sống.

Tôi mong sẽ nhận được nhiều ý kiến đóng góp để tiếp tục phát triển hướng nghiên cứu của mình nhằm đem lại cho người đọc một bộ tài liệu chi tiết và đầy đủ hơn. Xin chân thành cảm ơn!

DANH MỤC TÀI LIỆU THAM KHẢO

I. Tài liệu tiếng Việt

1. ThS. Phạm Văn Cường 2007, *BÀI GIẢNG TÍNH TOÁN SONG SONG*, HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG - Trang 5;

2. Thoại Nam 2017, *Nghiên cứu thiết kế hệ thống tính toán hiệu năng cao 50-100 Tflops*, Đề tài nghiên cứu khoa học cấp trường, Trường Đại học Bách khoa, Đại học Quốc gia TP. Hồ Chí Minh – Trang 12;

3. PGS. TSKH. Phạm Huy Điển, 2013-2015 "Phát triển tính toán khoa học chuyên ngành trên cơ sở máy tính hiệu năng cao chia sẻ tài nguyên tại Viện Hàn lâm Khoa học và Công nghệ Việt Nam", đề tài Độc lập cấp Viện Hàn lâm Khoa học và Công nghệ Việt Nam – Trang 32-36.

II. Tài liệu tiếng Anh

4. Robert McLay; Karl W. Schulz; William L. Barth; Tommy Minyard 2011, Best practices for the deployment and management of production HPC clusters, SC '11: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis – Page 934;

5. Draško Tomić; Dario Ogrizović 2012, Running High Performance Linpack on CPUGPU clusters, 2012 Proceedings of the 35th International Convention MIPRO – Page 415-416;

6. A.T. Wong et al., “Esp: A System Utilization Benchmark,” Proc. IEEE/ACM SC2000 Conf., IEEE CS Press, 2000; http://citeseer.ist.psu.edu/wong00esp.html – Page 60;

7. M. Berry et al., “The Perfect Club Benchmarks: Effective Performance Evaluation of Supercomputers,” Int’l J. Super-computer Applications, vol. 3, no. 3, 1989 – Page 5-40;

8. R.W. Hockney and M. Berry, “Parkbench Report: Public International Benchmarking for Parallel Computers,” Scientific Programming, vol. 3, no. 2, 1994 – Page 101-146;

9. R. Eigenmann et al., “Performance Evaluation and Bench-marking with Realistic Applications,” SPEC HPG Benchmarks: Performance Evaluation with Large-Scale Science and Engineering Applications, MIT

Press, 2001 – Page 40-48;

10. V. Aslot et al., “Specomp: A New Benchmark Suite for Measuring Parallel Computer Performance,” Proc. Workshop OpenMP Applications and Tools, LNCS 2104, Springer-Verlag, 2001 – Page 1-10;

11. D. Bailey et al., The NAS Parallel Benchmarks 2.0, tech. report NAS-95-020, NASA Ames Research Center, Dec. 1995 – Page 4-11;

12. B. Armstrong and R. Eigenmann, “Benchmarking and Performance Evaluation with Realistic Applications,” A Methodology for Scientific Benchmarking with Large-Scale Applications, MIT Press, 2001 – Page 109–127.

III. Trang web

13. https://www.ibm.com/topics/hpc;

14. https://en.wikipedia.org/wiki/Comparison_of_cluster_software;

15. http://www.rocksclusters.org/;

16. http://star.mit.edu/cluster/docs/latest/guides/sge.html;

17. http://talby.rcs.manchester.ac.uk/~ri/_linux_and_hpc_lib/sge_intro.html;

18. https://en.wikipedia.org/wiki/FLOPS;

19. https://encyclopedia.pub/entry/37858;

20. https://netlib.org/benchmark/hpl/tuning.html;

21. https://github.com/open-power/op-benchmark-recipes/blob/master/standard-benchmarks/HPL/Linpack_HPL.dat_tuning.md;

22. https://www.geeksforgeeks.org/sum-of-an-array-using-mpi/.