Tạp chí Khoa học Đại học Huế: Khoa học Tự nhiên; ISSN 1859–1388

Tập 127, Số 2A, 2018, Tr. 159–168; DOI: 10.26459/hueuni-jtt.v127i2A.4795

MỞ RỘNG MÔ HÌNH HOẠT ĐỘNG CỦA CAPE BẰNG CÁCH SỬ DỤNG MÁY ẢO

Đỗ Xuân Huyền[1], Hà Viết Hải[2]*

[1] Trường Đại học Khoa học, Đại học Huế, 77 Nguyễn Huệ, Huế, Việt Nam
[2] Trường Đại học Sư phạm, Đại học Huế, 32 Lê Lợi, Huế, Việt nam

Tóm tắt : Lập trình song song để đáp ứng yêu cầu về tốc độ xử lý cho các bài toán đòi hỏi tốc độ cao đã trở thành tất yếu trong những năm gần đây, khi tốc độ xung nhịp của mỗi lõi CPU đã gần như không tăng nữa. OpenMP, chuẩn lập trình song song cho các kiến trúc sử dụng bộ nhớ chia sẻ vì vậy càng được sử dụng rộng rãi cùng với sự phổ cập của các bộ vi xử lý đa lõi. CAPE (Checkpointing Aided Parallel Execution) là hướng tiếp cận để mở rộng OpenMP cho kiến trúc bộ nhớ phân tán. Các phân tích lý thuyết cũng như các kết quả thực nghiệm đã chứng minh được là CAPE có khả năng cung cấp hiệu năng hoạt động cao cũng như khả năng tương thích hoàn toàn với chuẩn OpenMP. Bài viết này đánh giá hiệu năng hoạt động của CAPE khi mở rộng mô hình hoạt động của CAPE trên mạng máy tính sử dụng bộ vi xử lý đa lõi theo hướng sử dụng mô hình đa tiến trình trên mỗi nút tính toán bằng cách sử dụng máy ảo.

Từ khóa: CAPE, Checkpointing Aided Parallel Execution, OpenMP, Tính toán song song, Tính toán phân tán, Tính toán hiệu năng cao, HPC

1 Mở đầu

1.1 OpenMP

OpenMP là một API cung cấp một mức trừu tượng hóa cao để viết các chương trình song song. Nó bao gồm một tập các biến môi trường, các chỉ thị và hàm, được xây dựng để hỗ trợ việc dễ dàng biến một chương trình tuần tự trên ngôn ngữ cơ sở là C/C++ hoặc Fortran thành một chương trình song song theo mô hình đa luồng.

Do sử dụng cấu trúc cơ sở là luồng (thread), mặc nhiên mô hình bộ nhớ của OpenMP là bộ nhớ chia sẻ, trong đó không gian nhớ được sử dụng chung giữa các luồng. Đã có nhiều nỗ lực để cài đặt OpenMP trên các kiến trúc sử dụng bộ nhớ phân tán nhưng chưa có phương án nào thành công trên cả hai mặt là tương thích hoàn toàn với chuẩn OpenMP và có hiệu năng cao. Những phương án nổi bật nhất có thể kể đến là sử dụng SSI 1; SCASH 2; sử dụng mô hình RC 3; biên dịch thành MPI 4,5; sử dụng Global Array 6; Cluster OpenMP 7. Cluster OpenMP là sản phẩm thương mãi của Intel cũng đòi hỏi phải sử dụng thêm các chỉ thị riêng của nó (không nằm trong

* *Liên hệ: haviethai@dhsphue.edu.vn*

Nhận bài: 6–5–2018; Hoàn thành phản biện: 31–5–2018; Ngày nhận đăng: 31–5–2018

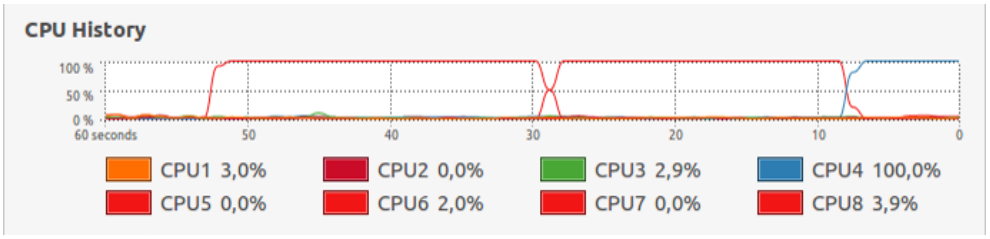
chuẩn OpenMP) trong một số trường hợp và vì vậy, nó cũng chưa cung cấp được một cài đặt tương thích hoàn toàn với OpenMP.

2 CAPE

CAPE 8 được phát triển với mục tiêu đưa OpenMP lên hệ thống sử dụng bộ nhớ phân tán, điển hình là cluster, lưới và đám mây. CAPE sử dụng đơn vị song song cơ sở là tiến trình (process), thay cho luồng như trong OpenMP nguyên bản để có thể vận hành trên các hệ thống sử dụng bộ nhớ phân tán. Mô hình bộ nhớ được sử dụng chính là mô hình bộ nhớ chia sẻ đồng bộ trễ, với khung nhìn riêng của các tiến trình chính là bộ nhớ cục bộ của tiến trình, và việc đồng bộ được thực hiện qua các cơ chế ngầm định cũng như cài đặt các chỉ thị đồng bộ. Đối với mô hình hoạt động, trong CAPE tất cả các nhiệm vụ quan trọng nhất của mô hình fork-join của OpenMP đều được cài đặt một cách tự động dựa trên kỹ thuật chụp ảnh tiến trình, bao gồm việc phân chia công việc cho các tiến trình, trích rút kết quả thực hiện trên các tiến trình phụ và cập nhật kết quả vào không gian nhớ của tiến trình chính.

CAPE được triển khai trên hệ thống mạng theo mô hình Chính-Phụ (Master-Slave). Trong đó nút chính (Master) đóng vai trò như luồng chính trong mô hình hoạt động OpenMP và các nút phụ (Slave/nút tính toán) đảm nhận vai trò của các luồng tính toán. Theo đó, nút chính thực hiện các phần mã của luồng chính, phân chia công việc cho các luồng phụ và chứa kết quả thực hiện được của tất cả các luồng sau khi thực hiện xong khối mã song song. Chi tiết về mô hình triển khai của CAPE xin tham khảo tại 9. CAPE đã được phát triển và cài đặt ban đầu để thực hiện được các chỉ thị song song OpenMP như parallel for, parallel section. Các phân tích lý thuyết cũng như số liệu thực nghiệm đã chứng minh được tính hiệu năng của CAPE, với hệ số tăng tốc đo được nằm trong khoảng 75% đến 90% so với hệ số gia tăng tối đa lý thuyết và xấp xỉ bằng 90% MPI 810.

Tuy nhiên, với mô hình hoạt động hiện tại của CAPE thì tại mỗi nút tính toán chỉ có một tiến trình thực hiện các câu lệnh của chương trình ứng dụng. Hơn nữa, tiến trình này cũng chỉ là một tiến trình tuần tự chứ không phải đa luồng. Do đó tại mỗi nút tính toán lúc đó, các câu lệnh của chương trình ứng dụng được thực hiện một cách tuần tự. Điều này có thể thấy rõ khi quan sát biểu đồ đo đạc các thông số vận hành của hệ thống khi đang chạy CAPE, như trong Hình 1. , biểu diễn tỷ lệ hoạt động của các lõi CPU tại một nút tính toán trang bị CPU 4 lõi. Qua biểu đồ có thể dễ dàng nhận thấy có sự luân phiên sử dụng giữa các lõi CPU nhưng tại mỗi thời điểm chỉ có một lõi CPU được khai thác chính, hết công suất, trong khi các lõi khác hầu như không hoạt động. Như vậy, tài nguyên tính toán của hệ thống còn đang bị lãng phí và khai thác chúng một cách hiệu quả có thể làm giảm thời gian thực hiện chương trình, tức là tăng được hiệu suất hoạt động của nó.



Hình 1. Tỷ lệ hoạt động của lõi tại các nút tính toán khi chạy một tiến trình

3 Giải pháp mở rộng mô hình hoạt động của CAPE trên mạng máy tính đa lõi

Việc khai thác khả năng của các bộ xử lý đa lõi trong CAPE có thể được tiến hành cả ở nút chính và các nút phụ. Trong phạm vi của bài báo này, chỉ có các nút phụ được xem xét.

Theo mô hình hoạt động hiện thời của CAPE, tại các vùng song song, mỗi nút phụ được giao một phần công việc của đoạn mã song song và thực hiện trên một tiến trình duy nhất. Chính điều này dẫn đến việc hầu như chỉ có một lõi CPU là được sử dụng với tỷ lệ cao, còn các lõi khác thì ở trạng thái nghỉ.

Để tăng được hiệu suất của chương trình, rõ ràng trong khoảng thời gian thực hiện tính toán ở các nút phụ, cần song song hóa các đoạn mã này, tức là phải tìm cách để có đa tiến trình hoặc đa luồng thực hiện. Chúng tôi đã sử dụng đa tiến trình trên mỗi nút phụ bằng cách cho thực hiện song song nhiều lần chương trình ứng dụng trên mỗi nút, mỗi lần thực hiện sẽ được chạy trong một tiến trình độc lập, có số hiệu khác nhau và được phân một phần công việc khác nhau của đoạn mã tính toán song song. Giải pháp này đã được chúng tôi cài đặt thử nghiệm 9, với cluster gồm các máy tính trang bị CPU Intel Core i3 (2 lõi x 2 siêu phân luồng), tốc độ 3.5GHz, RAM 4GB, chạy hệ điều hành Ubuntu 14.04, kết nối bằng mạng Ethernet tốc độ 100Mb/s (máy thực). Các thử nghiệm đã cho kết quả tốc tăng tốc được gần gấp đôi khi cho chạy 2 tiến trình ứng dụng đồng thời trên mỗi nút tính toán. Đây là một kết quả rất tốt và hợp lý về mặt hiệu năng. Tuy nhiên, giải pháp này gặp phải vấn đề về tính ổn định. Số liệu thực nghiệm cho thấy tình huống các tranh chấp về mặt tài nguyên và IP của hệ thống trên các nút tính toán dẫn đến dừng hệ thống lên đến đến gần 30% trường hợp thử nghiệm.

Để khắc phục tình huống các tranh chấp về mặt tài nguyên và IP của hệ thống, chúng tôi đã sử dụng nhiều máy ảo trên mỗi nút phụ, mỗi máy ảo chạy một tiến trình của đoạn mã tính toán song song. Đây cũng là nội dung chính của bài báo này.

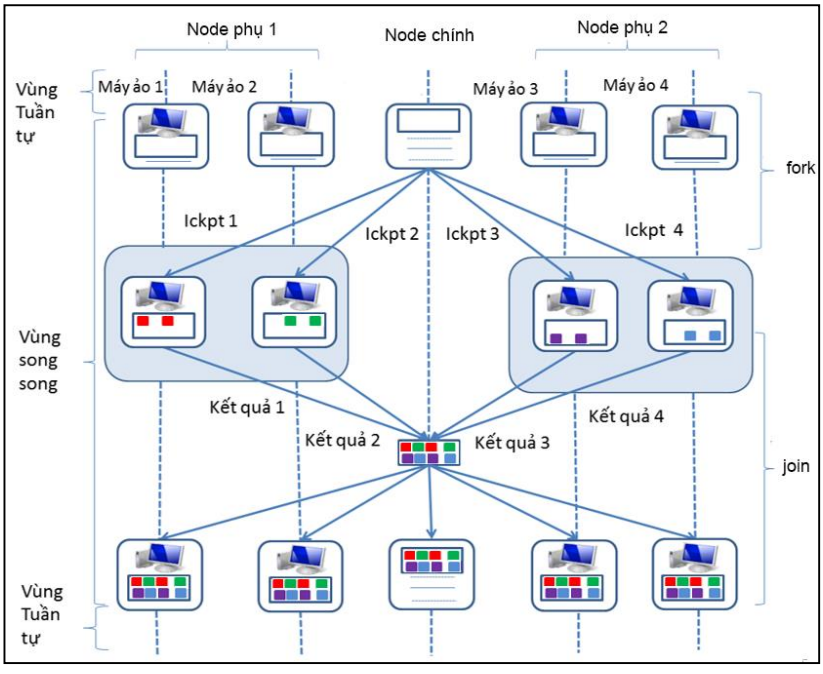
4 Sử dụng đa tiến trình trên các nút tính toán bằng cách sử dụng máy ảo

4.1 Nguyên tắc

Máy ảo (Virtual Machine) là một phần mềm mô phỏng một hệ thống máy tính. Máy ảo chạy trên hệ điều hành hiện tại - hệ điều hành chủ và cung cấp phần cứng ảo tới hệ điều hành khách. Đối với những hệ điều hành khách, máy ảo lại hiện diện như một cỗ máy vật lý thực sự. Ta có thể cài đặt nhiều máy ảo lên cùng một máy thực và cho chúng chạy đồng thời, với các chương trình khác nhau được chạy trên mỗi máy ảo.

Ý tưởng khởi đầu của giải pháp sử dụng đa tiến trình trên các nút tính toán bằng cách sử dụng máy ảo xuất phát từ vấn đề tương tranh giữa các tiến trình trong giải pháp sử dụng đa tiến trình trên mỗi nút phụ nói trên. Vấn đề này xảy ra do các tiến trình chạy trên cùng một máy (vật lý) nên có chung một địa chỉ IP. Khi chuyển qua sử dụng các máy ảo, trình ứng dụng của người dùng và trình theo dõi được cài đặt và thực hiện trên một máy ảo, và có nhiều máy ảo như vậy được cài đặt và vận hành song song trên mỗi máy vật lý của các nút phụ. Nhờ tính chất độc lập của các máy ảo, dù có cùng chạy trên một máy tính thì mỗi máy ảo vẫn có thể có địa chỉ IP riêng, do đó nguy cơ xung đột giữa các tiến trình do tranh chấp IP được giải quyết.

Hình 2. trình bày mô hình của giải pháp này, với 1 nút chính thực hiện tiến trình chính trên máy thực và 2 nút phụ, mỗi nút sử dụng 2 máy ảo, mỗi máy ảo thực hiện 1 tiến trình của chương trình ứng dụng.



Hình 2. Mô hình sử dụng nhiều máy ảo trên mỗi nút phụ

4.2 Cách thức cài đặt

Chúng tôi đã cài đặt giải pháp này với máy ảo KVM (Kernel-based Virtual Machine) là một máy ảo hóa đầy đủ cho Linux trên nền tảng phần cứng x86. KVM cung cấp một kiến trúc ảo hóa căn bản và một module xử lý chuyên biệt. Trình quản trị KVM cho phép thực hiện nhiều máy ảo trên cùng một máy tính vật lý, trong đó mỗi máy có một địa chỉ IP khác nhau và điều này là rất hữu hiệu cho việc cấu hình CAPE. Nhờ khả năng này mà ta có thể dễ dàng chạy nhiều tiến trình của người dùng trên mỗi nút phụ mà không bị các xung đột về IP. Thực tế việc cài đặt khá dễ dàng, chỉ cần cài đặt CAPE và trình ứng dụng trên một máy ảo, sau đó nhân bản nó thành nhiều bản trên cùng một máy. Công việc còn lại chỉ là viết lại Trình phân phối để cấu hình CAPE trên các máy ảo này.

Dưới đây là ví dụ về mã của Trình phân phối được viết lại để thay cho mã ban đầu ở 9. Hệ thống bao gồm một máy đóng vai trò của nút chính, hai máy khác đóng vai trò của các nút tính toán, mỗi máy chạy 2 máy ảo.

```
1.    #!bin/sh
2.    folder=/home/hahai/cape2/cdv9
3.    prog=mulmt
4.    num_nodes=4
5.    master=172.16.1.1
6.    node1a=172.16.1.20
6a.   node1b=172.16.1.24
7.    node2a=172.16.1.52
7a.   node2a=172.16.1.60
8.    ${folder}/dbpf -f ${folder}/${prog} -a ${master} –k
            ${num_nodes} -c ${master} -o 0 &
9.    ssh ${node1a} ${folder}/dbpf -f ${folder}/${prg}
     -a ${master} -k ${num_nodes} -o 1 &
9a.   ssh ${node1b} ${folder}/dbpf -f ${folder}/${prg}
     -a ${master} -k ${num_nodes} -o 2 &
10.   ssh ${node2a} ${folder}/dbpf -f ${folder}/${prg}
     -a ${master} -k ${num_nodes} -o 3 &
10a.  ssh ${node2b} ${folder}/dbpf -f ${folder}/${prg}
     -a ${master} -k ${num_nodes} -o 4 &
11.   exit 0
```

Hình. 3. Trình phân phối với các máy ảo

Các thay đổi trong mã trên bao gồm:

- Dòng 4: được chỉnh sửa để có 4 tiến trình phụ trên 4 máy ảo;
- Dòng 6: được chỉnh sửa tương ứng với IP của máy ảo đầu tiên trên node 1;
- Dòng 6a: một nhân bản của dòng 6 và địa chỉ IP tương ứng với máy ảo thứ 2 trên node 1;
- Dòng 7 và 7a: tương tự dòng 6 và 6a nhưng áp dụng cho node phụ thứ 2;
- Dòng 9: được sửa đổi để tham chiếu đến máy ảo 1a của node phụ thứ 1;
- Dòng 9a: một nhân bản của dòng 9 nhưng chỉ số tiến trình là 2 và tham chiếu tới node 1b, IP của máy ảo thứ 2 trên node 1. Như vậy, trên node vật lý thứ thứ 1, có 2 lần trình ứng dụng được thực hiện, tức là có 2 tiến trình với chỉ số khác nhau được tạo cho đoạn mã song song;
- Dòng 10 và 10b: được xử lý tương tự như với dòng 9 và 9a, có chỉ số tiến trình lần lượt là 3 và 4.

4.3 Đánh giá hiệu năng của giải pháp

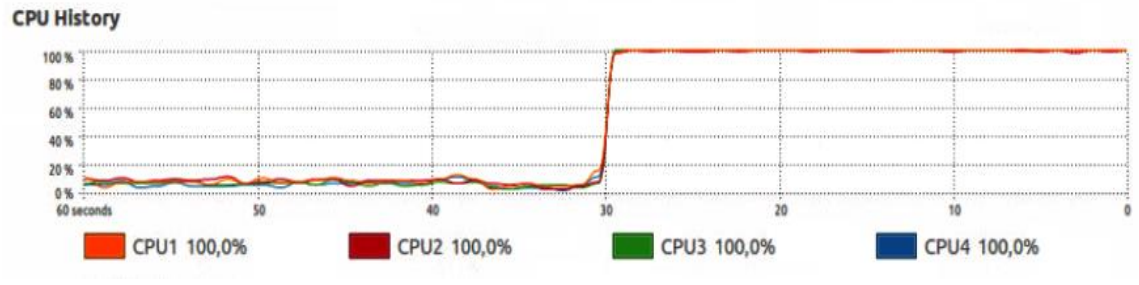
Chúng tôi thực hiện việc đánh giá giải pháp này trên một Cluster các máy tính để bàn có CPU Intel Core i3, 2 lõi x 2 hyperthread (được tính như 4 lõi), nối mạng Ethernet 100 Mb/s. Đây là một cấu hình khá phổ dụng hiện nay ở các trường đại học.

Đánh giá tỷ lệ sử dụng CPU

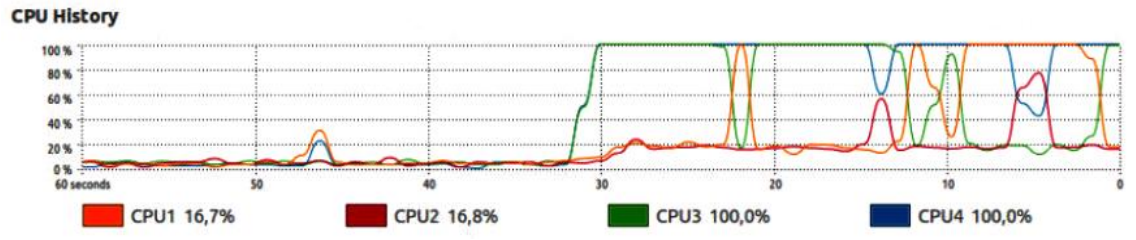
Về khả năng khai thác các lõi trên hệ thống máy tính đa lõi, chúng tôi thu được kết quả tương tự như với trường hợp sử dụng đa tiến trình trên máy thực đã giới thiệu trong mục 2. Trong đó, tỷ lệ khai thác 4 lõi của CPU là 100% khi sử dụng 4 máy ảo trên mỗi nút vật lý và 50% trong trường hợp sử dụng 2 máy ảo. Biểu đồ đo tỷ lệ sử dụng được chụp ảnh và trình bày lần lượt trong

Hình **4.** và

Hình **5.** . Lý do cho việc chọn thử nghiệm với 2 và 4 máy ảo trên mỗi máy tính là để tương ứng với cấu hình CPU 2 lõi x 2 siêu phân luồng, như vậy chỉ có thể chạy song song thật sự cho 2 x 2 tiến trình.



Hình 4. Tỷ lệ khai thác các lõi khi chạy 4 máy ảo

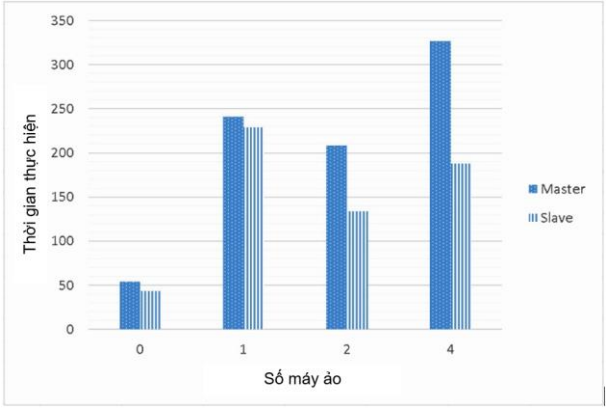


Hình 5. Tỷ lệ khai thác các lõi khi chạy 2 máy ảo

Đánh giá hiệu năng thực hiện chương trình

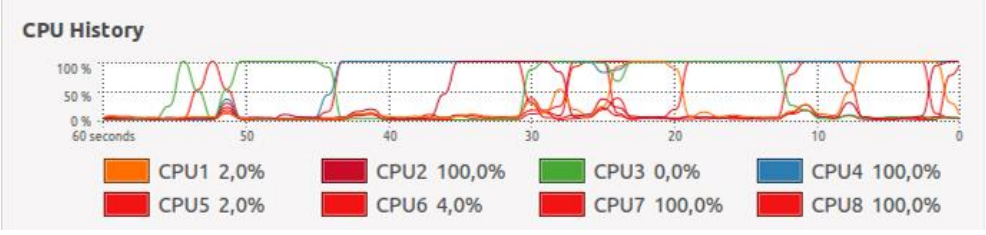
Để đánh giá giải pháp này về mặt hiệu năng, chúng tôi đã thử nghiệm trên bài toán nhân ma trận với kích thước ma trận là 6000x6000, sử dụng 21 nút vật lý (1 nút chính và 20 nút phụ), số lượng máy ảo trên mỗi nút phụ biến thiên với các giá trị 1, 2 và 4. Thời gian thực hiện được so sánh với giải pháp CAPE hiện tại, tức là chạy tiến trình trên máy thực. Biểu đồ so sánh các thời gian chạy được trình bày trên Hình 6. trong đó nhóm 0 tương ứng với trường hợp chạy 1 tiến trình trên máy thực. Theo nguyên lý hoạt động chung của CAPE, các số liệu tương tự cũng có thể thu được khi thử nghiệm cho các bài toán có chứa vòng lặp for lớn có thể song song hóa với cấu trúc parallel for của OpenMP.

Như biểu đồ thể hiện, mặc dù CPU làm việc nhiều hơn, kết quả về mặt hiệu năng là không như mong đợi, thời gian thực hiện trong trường hợp sử dụng máy ảo luôn lớn hơn thời gian khi sử dụng máy thực (vật lý). Ngay cả trong trường hợp tốt nhất là sử dụng 2 máy ảo trên mỗi nút phụ, thời gian trên nút phụ và nút chính tăng lên khoảng 3 đến 4 lần. Điều này là kết quả của chi phí của máy ảo lên thời gian thực hiện và trong trường hợp này, chi phí này là lớn hơn nhiều so với lợi ích thu lại khi chạy nhiều tiến trình của của trình ứng dụng trên các máy ảo này. Do đó, càng sử dụng nhiều máy ảo thì các chương trình càng chạy chậm lại, dẫn đến sự suy giảm hiệu năng.



Hình 6. Thời gian thực hiện chương trình với số máy ảo khác nhau

Để kiểm chứng lại việc giảm hiệu năng của hệ thống khi sử dụng máy ảo chúng tôi đã thử nghiệm chạy một chương trình tuần tự nhân hai ma trận kích thước 6000x6000 viết bằng ngôn ngữ C chạy trên một máy tính cài hệ điều hành Ubuntu 14.04, máy ảo KVM, có cấu hình CPU Core i7, 4 lõi x 2 hyperthread (được tính như 8 lõi), RAM 8 GB có cài các máy ảo với các kịch bản khác nhau để kiểm chứng: kịch bản chạy chương trình trên máy thực, trên máy 1 máy ảo sử dụng cả 8 lõi CPU, chạy trên 2 và 4 máy ảo mỗi máy ảo sử dụng 2 lõi CPU, chạy trên 6 vào 8 máy ảo sử dụng mỗi máy ảo 01 lõi CPU, dung lượng RAM được phân đều cho các máy ảo. Để đơn giản trong việc thử nghiệm ở kịch bản sử dụng nhiều máy ảo với chương trình được lập trình tuần tự, chúng tôi chạy đồng thời chương trình nhân ma trận kích thước 6000x6000 ở các máy ảo. Ví dụ: Trường hợp chạy 2 máy ảo nếu thời gian chạy trên mỗi máy ảo tương đương với máy thực thì có nghĩa đã tăng tốc được khoảng gấp đôi. Kết quả thực nghiệm với số liệu như ở Bảng 1. cho thấy việc sử dụng máy ảo luôn làm giảm tốc độ thực hiện chương trình và mức độ suy giảm trong trường hợp này khá lớn. Đối với trường hợp tốt nhất là sử dụng 4 máy ảo thì mức độ suy giảm cũng tới khoảng 1.8 lần, trong khi đó CPU vẫn thường xuyên sử dụng tối đa 100% công suất của 4 lõi và có sự luân phiên sử dụng các lõi CPU theo thời gian, như thể hiện ở Hình 7. .



Hình 7. Trạng thái sử dụng CPU khi chạy chương trình trên 4 máy ảo

Bảng 1. Kết quả thực nghiệm mức độ giảm hiệu năng khi sử dụng máy ảo

Số máy ảo	Kích thước ma trận	Tổng thời gian thực hiện chương trình	Tổng thời gian/số lượng máy	% suy giảm hiệu năng so với máy thực	Ghi chú
0	6000	954	954	0%	Máy thực
1	6000	4,542	4,542	476%	01 máy ảo
2	6000	5,172	2,586	271%	02 máy ảo
4	6000	6,829	1,707	179%	04 máy ảo
6	6000	10,715	1,786	187%	06 máy ảo
8	6000	21,755	2,719	285%	08 máy ảo

4.4 Đánh giá ưu điểm và nhược điểm của giải pháp

Tương tự như với giải pháp sử dụng đa tiến trình trên máy thực, ưu điểm nổi bật của giải pháp này là tính đơn giản, hầu như mã chương trình đều được giữ nguyên (trừ chỉnh sửa trong Trình phân phối). Tất nhiên là giải pháp này có phức tạp hơn do phải có thêm một bước là cài đặt và sử dụng các máy ảo, tuy nhiên việc này không có vấn đề gì về mặt kỹ thuật, chỉ cần sử dụng một kỹ thuật máy ảo sẵn có. Ưu điểm thứ 2 là ở mỗi tiến trình được cài đặt trên mỗi máy ảo có IP khác nhau nên tránh được xung đột và các lỗi gây ra do xung đột, một lỗi thường xảy ra trong giải pháp sử dụng nhiều tiến trình trên các máy thực.

Tuy nhiên, nhược điểm lớn của giải pháp này là nó làm tăng thời gian thực hiện của chương trình chứ không là làm giảm như kỳ vọng. Điều này làm cho giải pháp không có ý nghĩa gì khi xét về mặt hiệu năng, tức là không có ý nghĩa gì cho việc nâng cao hiệu suất hoạt động của hệ thống.

5 Kết luận và hướng phát triển

Bài báo này đề xuất ý tưởng và thực hiện cài đặt thử nghiệm đánh giá hiệu năng triển khai CAPE theo hướng sử dụng đa tiến trình/đa luồng trên các hệ thống máy tính đa lõi bằng cách sử dụng máy ảo. Kết quả thực nghiệm giải pháp được trình bày cụ thể trong mục 3, hoạt động không có lỗi, khai thác được tỷ lệ làm việc của CPU cao, nhưng lại giảm hiệu năng hoạt động khi so sánh với trường hợp triển khai trên máy thực, do chi phí của máy ảo lớn hơn phí phí khác thác tối đa tỷ lệ làm việc của CPU. Để khai thác tốt hơn các bộ vi xử lý đa lõi, thay vì mở rộng mô hình hoạt động theo hướng đa tiến trình bằng cách sử dụng các máy ảo như trên, chúng tôi định hướng sẽ sử dụng đa luồng trên mỗi nút tính toán. Theo nhận định ban đầu, sử dụng đa luồng sẽ giúp song song hóa các đoạn mã tính toán nhưng tránh được các lỗi đụng độ về IP của giải pháp sử dụng đa tiến trình trên máy thực cũng như không gặp phải vấn đề suy giảm hiệu suất do sử dụng các máy ảo. Tuy nhiên, để thực hiện được việc này thì phải nâng cấp trình checkpointer của CAPE để theo dõi được các tiến trình đa luồng. Đây là một việc có thể thực hiện được nhưng rất phức tạp, đòi hỏi nhiều thời gian và công sức. Trong thời gian tới, chúng tôi sẽ tập trung nghiên cứu phát triển CAPE theo hướng này.

Tài liệu tham khảo

1. D. Margery, G. Vallée, R. Lottiaux, C. Morin, J. Berthou. Kerrighed: A SSI Cluster OS Running OpenMP. *Proceeding of the Fifth European Workshop on OpenMP (EWOMP 2003)*. Aachen, Germany, September, 2003
2. M. Sato, H. Harada, A. Hasegawa and Y. Ishikaw, Cluster-enabled OpenMP: An OpenMP compiler for the SCASH software dis-tributed shared memory system. *Journal Scientific Programming*, Volume 9 Issue 2,3 (2001).
3. J. Tao, W. Karl, C. Trinitis. Implementing an OpenMP Execution Environment on InfiniBand Clusters. *Proceeding of the First International Workshop on OpenMP (IWOMP 2005)*. Eugene, Oregon, 2005.

4. A. Saa-Garriga, D. Castells-Rufas, J. Carrabina (2015). OMP2MPI: Automatic MPI code generation from OpenMP programs. *Proceedings of the Workshop on High Performance Energy Efficient Embedded Systems (HIP3ES 2015), Collocated with HIPEAC 2015 Conference*. Amsterdam, Holland, January 2015.
5. Jacob A.C. et al. Exploiting Fine - and Coarse - Grained Parallelism Using a Directive Based Approach. *Lecture Notes in Computer Science*, vol 9342. Springer, Cham, pp. 30-41, 2015.
6. L. Huang and B. Chapman and Z. Liu. Towards a more efficient implementation of OpenMP for clusters via translation to Global Arrays. *Journal of Parallel Computing 31*, pp 1114–1139, 2005.
7. J.P Hoeflinger (Intel). Cluster OpenMP* for Intel® Compilers, 2010. Online tại: https://software.intel.com/en-us/articles/cluster-openmp-for-intel-compilers. Truy cập 6/5/2018.
8. Viet Hai Ha and Éric Renault. Improving Performance of CAPE using Discontinuous Incremental Checkpointing. *Proceedings of the IEEE International Conference on High Performance and Communications 2011 (HPCC-2011)*. Banff, Canada, September 2011.
9. Hà Viết Hải, Nguyễn Cảnh Hoài Đức, Đỗ Xuân Huyền. Sử dụng nhiều tiến trình trên các nút tính toán để gia tăng hiệu năng hoạt động của CAPE trên mạng các máy tính đa lõi. *Tạp chí Khoa học, Đại học Huế*, Tập 121, số 7-A, 2016.
10. Van Long Tran, Eric Renault, Viet Hai. Analysis and evaluation of the performance of CAPE. *Proceeding of The 16th IEEE International Conference on Scalable Computing and Communications*. Toulouse, France, 8/2016.

EXTEND THE CAPE'S DEPLOYMENT MODEL BY VIRTUAL MACHINE

Do Xuan Huyen[1*], Ha Viet Hai[2*]

[1] College of Sciences, Hue University, 77 Nguyen Hue, Hue, Vietnam
[2] College of Education, Hue University, 32 Le Loi, Hue. Vietnam

Abstract. Parallel programming to meet the increasing demand for more computing power and higher processing speed, as the clock speed of each CPU core has not increased. OpenMP, the parallel programming standard for shared memory architecture, is widely used. CAPE (Checkpointing Aided Parallel Execution) is an approach to extend OpenMP for distributed memory architecture. Theoretical demonstrations and experiments have proved that CAPE has potential competence to become a fully compliance and high-performance implement of OpenMP for distributed-memory systems. This article evaluates the performance of CAPE when extending CAPE's execution model on a multi-core processor-based computer network using virtual machine.

Keywords: CAPE, Checkpointing Aided Parallel Execution, OpenMP, Parallel Computing, Distributed Computing, High Performance Computing, HPC