CAPE - CÀI ĐẶT OPENMP DỰA TRÊN KỸ THUẬT CHỤP ẢNH TIẾN TRÌNH - HIỆN TẠI VÀ HƯỚNG PHÁT TRIỂN

Hà Viết Hải

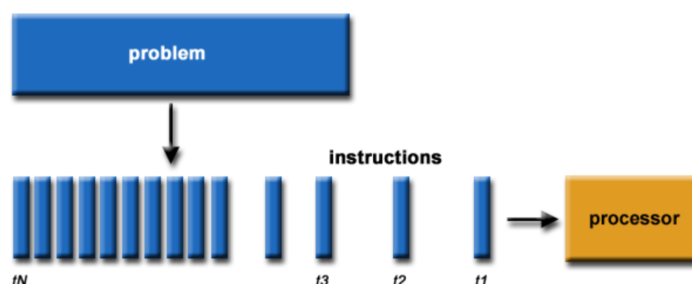
Đại học Sư phạm, Đại học Huế

Tóm tắt: *OpenMP, với tính đơn giản, dễ học, dễ lập trình và hiệu năng cao đã nhanh chóng trở thành chuẩn lập trình song song cho các kiến trúc sử dụng bộ nhớ chia sẻ. Cùng với sự phổ dụng của các máy tính đa lõi (multicore), OpenMP ngày càng được sử dụng rộng rãi. Tuy nhiên, chưa có cài đặt nào của OpenMP có thể chạy được trên các kiến trúc sử dụng bộ nhớ phân tán – điển hình là các cluster, grid – mà đảm bảo được đồng thời cả hai tính chất là tương thích hoàn toàn với chuẩn này và cung cấp được hiệu năng cao. CAPE (Checkpointing Aided Parallel Processing) – một kỹ thuật cài đặt OpenMP cho các kiến trúc phân tán – là một hướng triển vọng để làm được điều này. Bài viết này giới thiệu CAPE từ ý tưởng cơ bản, những kết quả đạt được cùng với những hướng nghiên cứu trong tương lai.*

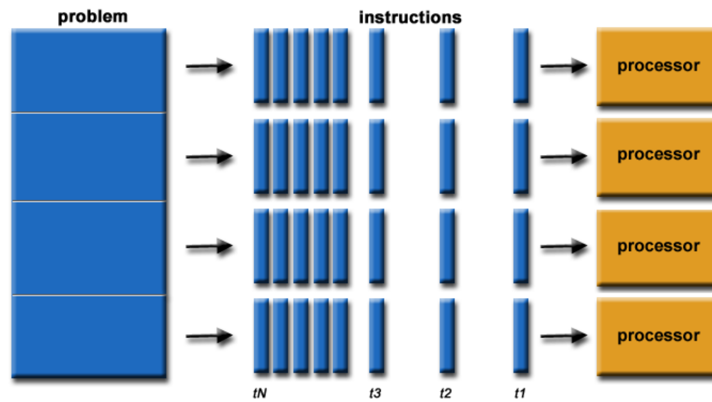
I. LẬP TRÌNH SONG SONG VÀ MỘT SỐ VẤN ĐỀ LIÊN QUAN

1. Khái niệm lập trình song song

Đồng hành với việc tăng tốc độ hoạt động của các bộ vi xử lý, yêu cầu về tốc độ tính toán của các bài toán lớn cũng không ngừng tăng theo. Do vậy, luôn có những bài toán cần tốc độ xử lý vượt quá tốc độ xử lý của một chương trình tuần tự, trên một kiến trúc xử lý đơn. Điều này càng đúng trong thời điểm hiện tại, khi các kiến trúc vi xử lý đã gần đạt đến giới hạn cuối cùng về kích thước của đơn vị cơ sở, cũng đồng nghĩa với việc tăng tốc độ của các một bộ xử lý đơn lõi lên gấp đôi – theo định luật Moore – sau 50 năm nghiệm đúng đã sắp đến thời điểm kết thúc. Để giải quyết vấn đề cốt lõi này, xử lý song song (parellel processing) là hướng giải quyết khả thi duy nhất. Nguyên lý chung của xử lý song song là chia bài toán lớn (theo số lần thực hiện các câu lệnh hoặc theo kích thước dữ liệu hoặc đồng thời theo cả hai) thành nhiều phần con và cho thực hiện một cách song song các phần này trên các đơn vị xử lý khác nhau. Nguyên lý này khác biệt căn bản với nguyên lý hoạt động của mô hình xử lý tuần tự (sequential processing) là thực hiện lần lượt các xử lý, xử lý sau chỉ tiến hành khi đã xong xử lý trước. Hình 1 và Hình 2 [1] lần lượt mô tả nguyên lý hoạt động của hai mô hình này.



Hình 1. Xử lý tuần tự



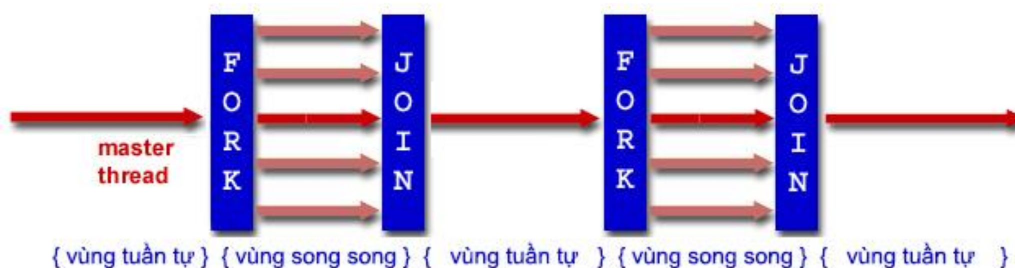
Hình 2. Xử lý song song

Tuy có ưu điểm về mặt tốc độ tính toán, mô hình xử lý song song cũng có những nhược điểm nhất định. Thứ nhất là nó đòi hỏi phải có những kiến trúc phần nền (cả phần cứng và phần mềm) hỗ trợ. Thứ hai là phải có công cụ lập trình khai thác khả năng của phần nền này để tạo giao diện cho người lập trình có thể tạo các chương trình xử lý song song. Yêu cầu thứ nhất được đáp ứng qua các mô hình đa bộ xử lý, đa lõi (trên CPU và mới đây là GPU) cùng các kiến trúc mạng. Yêu cầu thứ hai cũng đã liên tục phát triển tương ứng với các phát triển của phần nền mà điển hình là các mô hình đa tiến trình (multiprocess), đa tiểu tiến trình (multithread), truyền thông điệp (message passing) và mới đây là các công cụ ở bậc cao hơn như OpenMP, Cuda và OpenCL.

2. 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.

OpenMP sử dụng mô hình thực hiện *fork-join* với cấu trúc song song cơ sở là thread. Ban đầu, chương trình chỉ có một thread chính, thực hiện các đoạn mã tuần tự. Mỗi khi gặp một chỉ thị song song của OpenMP, thread chính sinh ra một đội (team) làm việc bao gồm chính nó (thread chính) và một tập các thread phụ (pha fork) và công việc được phân chia cho mỗi thread trong đội. Sau khi các thread phụ hoàn thành phần mã của nó, kết quả được cập nhật vào không gian nhớ của thread chính và chúng có thể kết thúc hoạt động (pha join). Như vậy, sau pha này thì chương trình chỉ còn lại một tiến trình như ban đầu. Các pha fork-join có thể được tiến hành nhiều lần trong một chương trình và có thể lồng nhau. Hình 3 mô tả mô hình hoạt động của một chương trình OpenMP có một số đoạn trình thực hiện tuần tự và song song nối tiếp nhau.



Hình 3. Mô hình hoạt động của OpenMP

Điểm thuận tiện đối với người lập trình là họ có thể bắt đầu viết chương trình theo kiểu tuần tự trên ngôn ngữ cơ sở, biên dịch và chạy thử nó. Sau đó, việc song song hóa có thể được

tiến hành từng bước bằng cách thêm các chỉ thị biên dịch OpenMP vào những nơi cần thiết. Các chỉ thị này sẽ được một trình biên dịch C/C++ hoặc Fortran có hỗ trợ OpenMP biên dịch thành dạng mã thực thi. OpenMP cung cấp các chỉ thị để song song hóa những đoạn mã tiềm năng nhất cho việc xử lý song song, bao gồm các vòng lặp for, các đoạn mã được thực hiện song song như nhau và không như nhau trên các đơn vị xử lý. Đây cũng chính là lý do chủ yếu để xét về góc độ người lập trình, OpenMP trở nên dễ học, dễ sử dụng nhưng về mặt hiệu quả thì lại có thể cung cấp được hiệu năng hoạt động cao. Ví dụ dưới đây minh họa việc biến một vòng lặp `for` của C thành vòng lặp `for` song song dạng OpenMP, trong đó chỉ cần một chỉ thị `#pragma` duy nhất được thêm vào mã nguồn ban đầu.

```
# pragma omp parallel for
    for (i = 0; i < N; i++)
        Mã của thân của vòng lặp for;
```

Do sử dụng cấu trúc cơ sở là 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 tiến trình. Tuy nhiên, để tăng tốc độ tính toán, OpenMP sử dụng mô hình bộ nhớ đồng bộ trễ (Relaxed Consistency). Các tiến trình có thể sử dụng bộ nhớ cục bộ riêng để tăng tốc độ truy cập. Việc đồng bộ hóa giữa không gian nhớ của các tiến trình được thực hiện ngầm ở đầu và cuối các cấu trúc song song, hoặc tường minh bằng cách sử dụng chỉ thị `flush`. Tuy nhiên, hiện tại thì OpenMP vẫn chỉ mới được cài đặt một cách hoàn chỉnh dưới dạng sử dụng bộ nhớ chia sẻ (multiCPU, multicore) do sự phức tạp của việc cài đặt tất cả các yêu cầu của OpenMP trên các kiến trúc sử dụng bộ nhớ khác. Đây chính là động lực để nhiều nghiên cứu với mục tiêu cài đặt OpenMP trên các kiến trúc sử dụng bộ nhớ phân tán.

3. Các cài đặt của OpenMP trên các kiến trúc sử dụng bộ nhớ phân tán

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 [3]; SCASH [4]; sử dụng mô hình RC [5]; biên dịch thành MPI [6][7]; sử dụng Global Array [8]; Cluster OpenMP [9].

Phương án sử dụng một ảnh chung hệ thống (Single System Image - SSI) như một bộ nhớ chung cho các tiến trình trong hệ thống là hướng trực quan nhất để cài đặt OpenMP trên các kiến trúc phân tán. Do có bộ nhớ chung, các chương trình OpenMP có thể dễ dàng được biên dịch để chạy trên các tiến trình trên các máy tính khác nhau của hệ thống. Tuy nhiên, do việc truy cập bộ nhớ chung trên mạng và việc đồng bộ hóa các truy cập này tiêu tốn nhiều thời gian, tiếp cận này không thể cung cấp hiệu năng cao. Một thử nghiệm thực tế [3] đã cho thấy tốc độ xử lý của chương trình lại tỷ lệ nghịch với số tiến trình, thay vì tỷ lệ luận như mong đợi. Để giảm sự trả giá về thời gian do đặt toàn bộ không gia nhớ của các tiến trình lên trên một SSI, các phương án như SCASH chỉ ánh xạ các biến chia sẻ giữa các tiến trình lên phần bộ nhớ chung, còn tiến trình vẫn sử dụng bộ nhớ cục bộ. Một phương án khác theo hướng này là sử dụng mô hình bộ nhớ đồng bộ hóa trễ (Relaxed Consistency Memory Model). Tuy nhiên, các phương án này đều gặp khó khăn trong việc tự động xác định các biến chia sẻ cũng như trong việc cài đặt một số cấu trúc bộ nhớ chia sẻ khác nên chúng đều không thể tương thích hoàn toàn với chuẩn OpenMP.

Các phương án biên dịch OpenMP thành dạng MPI có ưu điểm về hiệu năng nhưng vẫn không thể cài đặt được tất cả các cấu trúc và chỉ thị của OpenMP. Ngay cả với Cluster OpenMP, một sản phẩm thương mãi của Intel cũng đòi hỏi phải sử dụng thêm các chỉ thị

riêng của nó (không nằm trong 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.

4. Kỹ thuật chụp ảnh tiến trình (Checkpointing)

Chụp ảnh tiến trình (đánh dấu tiến trình) là kỹ thuật chụp và lưu trữ trạng thái của một chương trình đang vận hành sao cho nó có khả năng khôi phục lại trạng thái ở các thời điểm sau đó [10]. Kỹ thuật chụp ảnh tiến trình được sử theo nhiều hướng khác nhau như gia tăng khả năng chịu lỗi của chương trình theo mô hình `rollback`, di trú tiến trình, backup hệ thống…

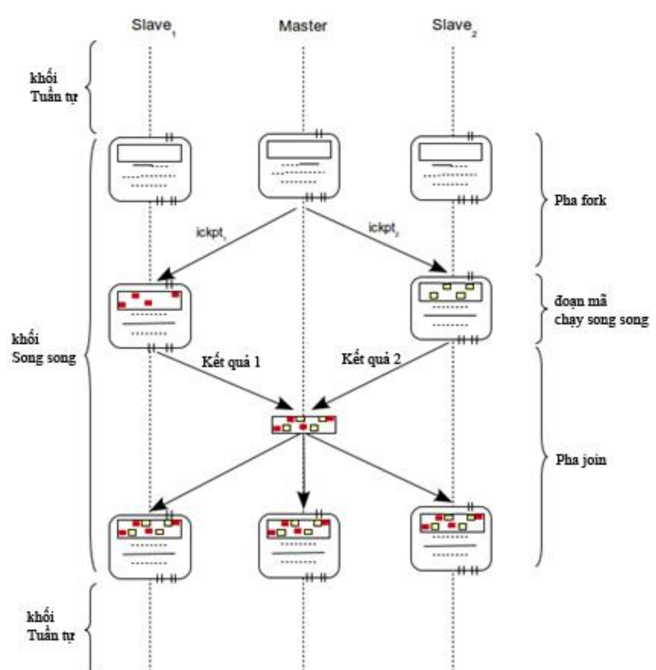
Kỹ thuật chụp ảnh tiến trình có thể được phân chia vào hai nhóm: chụp ảnh đầy đủ (Complete Checkpointing) và chụp ảnh gia tăng (Incremental Checkpointing). Ở nhóm thứ nhất, toàn bộ không gian nhớ của tiến trình và một số thông số hệ thống khác của nó được chụp và lưu trữ cho mỗi ảnh của tiến trình (checkpoint). Đây là kiểu đơn giản nhưng các ảnh có kích thước lớn và dễ có dư thừa dữ liệu cho những ảnh chụp liên tiếp nhau của tiến trình. Ở nhóm thứ hai, mỗi ảnh của tiến trình chỉ lưu lại những phần không gian nhớ và thông số hệ thống của tiến trình đã bị cập nhật kể từ khi khởi tạo chương trình hoặc kể từ lần chụp ảnh trước. Để biết được những phần bị cập nhật này, chương trình được chụp ảnh thường được giám sát và chụp ảnh bởi một tiến trình chụp ảnh (checkpointer) chạy song song với nó. Trong kỹ thuật này, các ảnh tiến trình phải được chụp một cách kế tiếp nhau.

Để phục vụ một cách tối ưu cho CAPE, trong đó có yêu cầu chụp ảnh từng phần không liên tục của tiến trình, chúng tôi đã phát triển kỹ thuật Chụp ảnh tiến trình Gia tăng Rời rạc (Discontinuos Incremental Checkpointing – DICKPT) [11]. Kỹ thuật này đặt cơ sở trên kỹ thuật chụp ảnh tiến trình gia tăng nói trên, với bổ sung khả năng chụp ảnh từng đoạn rời rạc của một tiến trình. Bên cạnh khả năng mới này, các phân tích và số liệu thực nghiệm đã chứng minh hiệu suất cao của nó trong cả hai mặt là giảm kích thước ảnh chụp và giảm thời gian thực hiện quá trình chụp ảnh.

II. CAPE DỰA TRÊN KỸ THUẬT ĐÁNH DẤU TIẾN TRÌNH GIA TĂNG RỜI RẠC

1. Mô hình vận hành

CAPE là một hướng mới để cài đặt OpenMP trên các hệ thống phân tán. CAPE sử dụng đơn vị song song cơ sở là tiến trình, thay cho thread như trong OpenMP nguyên bản để có thể vận hành trên các hệ thống sử dụng bộ nhớ phân tán. Với CAPE, tất cả các nhiệm vụ quan trọng nhất của quá trình mô hình fork-join đều được cài đặt một cách tự động dựa trên kỹ thuật đánh dấu 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… Phiên bản đầu tiên của CAPE sử dụng các ảnh chụp tiến trình đầy đủ đã chứng minh được khả năng thực hiện của CAPE [12]. Tuy nhiên, do các ảnh chụp tiến trình đầy đủ có kích



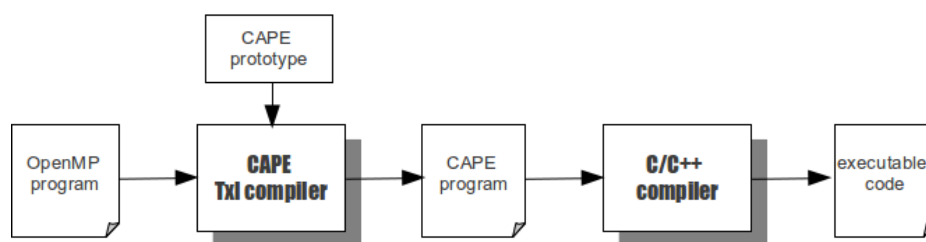
Hình 4. Mô hình hoạt động của CAPE dựa trên kỹ thuật chụp ảnh tiến trình gia tăng rời rạc

thước lớn dẫn đến lưu lượng lớn dữ liệu được chuyển giữa các tiến trình. Mặt khác, việc trích rút kết quả tính toán tại các tiến trình phụ dựa trên sự so sánh các ảnh chụp tiến trình đầy đủ có số lượng phép so sánh rất lớn. Cả hai điều này đã làm giảm hiệu suất vận hành cũng như khả năng mở rộng (scalability) của CAPE. Những yếu điểm này đã được khắc phục trong phiên bản thứ hai của CAPE [13], dựa trên kỹ thuật DICKPT. Hình 4 mô tả mô hình hoạt động của phiên bản này, trên đó hệ thống bao gồm 3 node, một node chạy tiến trình chính (master) và hai node phụ, mỗi node chạy một tiến trình phụ (slave) thực hiện công việc tính toán. Để đơn giản hóa cho việc trình bày, trong các khối mã song song, tiến trình chính chỉ làm nhiệm vụ phân chia công việc tới các tiến trình phụ và nhận kết quả từ chúng. Điều này là không bắt buộc và tiến trình chính có thể thực hiện một phần việc tính toán trong các khối mã song song.

Khởi đầu, chương trình được khởi tạo trên tất cả các node và các đoạn mã tuần tự được thực hiện như nhau trên tất cả các node đó. Khi gặp một cấu trúc song song OpenMP, tiến trình chính phân chia công việc tới các tiến trình phụ bằng cách gửi đến mỗi tiến trình phụ một ảnh chụp tiến trình gia tăng. Lưu ý là ảnh chụp này có kích thước rất bé, thường là chỉ vài byte, do nó được lấy chứa kết quả của việc thực hiện một số câu lệnh đơn giản và không làm thay đổi nhiều không gian nhớ. Ở mỗi node phân tán, tiến trình phụ nhận ảnh chụp tiến trình, tích hợp nó vào trong không gian nhớ của mình và khởi tạo quá trình chụp ảnh tiến trình DICKPT. Sau đó tiến trình phụ thực hiện việc tính toán được giao và trích rút kết quả thực hiện bằng một ảnh chụp tiến trình. Kết quả này được gửi về tiến trình chính, nơi nhận và tổ hợp chúng lại, sau đó tích hợp vào trong không gian nhớ của nó, cũng như gửi lại cho các tiến trình phụ để đồng bộ hóa không gian nhớ của tất cả các tiến trình, chuẩn bị cho việc thực hiện các đoạn mã tiếp theo của chương trình. Sau bước này, không gian nhớ của mỗi tiến trình đều đã có kết quả thực hiện của phần mã song song, vì thế, chúng xem như đều đã thực hiện xong phần việc của đoạn mã này.

2. Mô hình chuyển đổi cho các cấu trúc song song OpenMP

Các mô hình chuyển đổi được trình biên dịch của CAPE sử dụng để chuyển đổi các cấu trúc OpenMP sang dạng mã tương đương sử dụng các hàm CAPE, trong đó không còn các chỉ thị OpenMP nữa. Kết quả nhận được sau đó lại tiếp tục được biên dịch bởi một trình biên dịch C/C++ bình thường thành dạng mã thực thi và có thể chạy trên nền hỗ trợ CAPE. Hình 5 biểu diễn các bước trong quá trình biên dịch một chương trình OpenMP (với ngôn ngữ cơ sở là C/C++) thành mã thực thi trên nền CAPE. Các cấu trúc trong phiên bản CAPE hiện tại được xây dựng với giả thiết là các câu lệnh song song thỏa mãn các điều kiện Bernstein, tức là mỗi phần mã thành phần có thể được thực hiện một cách độc lập, đầu vào và đầu ra của chúng không liên quan nhau và trong quá trình thực hiện, không có sự trao đổi dữ liệu với nhau. Chi tiết của mô hình chuyển đổi có thể tìm thấy trong [13].



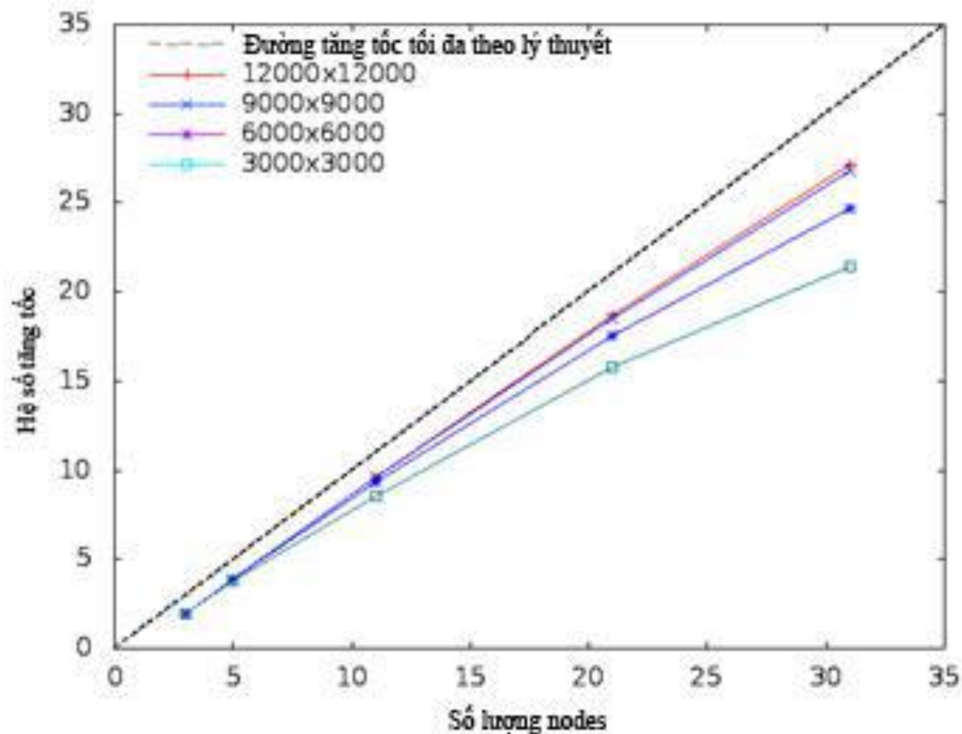
Hình 5. Quá trình biên dịch chương trình OpenMP thành mã thực thi trên nền CAPE

3. Kết quả đã đạt được

CAPE đã được phát triển và cài đặt bước đầu để thực hiện được các chỉ thị song song OpenMP như `parallel for`, `parallel section`. Chúng tôi đã tiến hành thực nghiệm

trên một bài toán nhân ma trận vuông, với kích thước thay đổi từ 3000x3000 đến 12000x12000. Các ma trận thuộc dạng bình thường với các phần tử mang giá trị khác 0. Hệ thống phần nền được sử dụng là một cluster các máy tính để bàn, mỗi máy trang bị CPU Intel®Core™2 Duo E8400 3GHz, RAM 2GB, chạy hệ điều hành Ubuntu 10.10, kết nối bằng mạng Ethernet tốc độ 100MB/s. Mỗi thử nghiệm được thực hiện ít nhất là 10 lần, với khoảng tin cậy tối thiểu là 90%.

Hình 6 biểu diễn hệ số tăng tốc (speedup) của CAPE trên số lượng các máy và các kích thước ma trận khác nhau. Đường đứt nét biểu diễn sự gia tăng tối đa lý thuyết. Biểu đồ này cho thấy một cách rõ ràng rằng giải pháp CAPE đã đạt được kết quả rất tốt, với hệ số tăng tốc đo được nằm trong khoảng 75% đến 90% so với hệ số gia tăng tối đa lý thuyết.



Hình 6. Hệ số tăng tốc của CAPE

III. CÁC HƯỚNG PHÁT TRIỂN SẮP TỚI

1. Tiếp tục xây dựng để CAPE tương thích hoàn toàn với OpenMP

Để CAPE trở thành một cài đặt hoàn chỉnh của OpenMP, các vấn đề sau còn phải giải quyết:

1. Cài đặt hoàn chỉnh các xử lý gộp kết quả thực hiện từ các node phụ và phân phối lại kết quả tổng hợp đến các node phụ này, nhằm làm cho CAPE có thể chạy được các chương trình OpenMP chứa nhiều đoạn mã song song.
2. Phát triển và cài đặt các mô hình xử lý các biến chia sẻ, nhằm vượt qua ràng buộc chỉ chạy được các bài toán thỏa mãn điều kiện Bernsteins.

2. Nâng cao hiệu suất hoạt động của CAPE

Để nâng cao được hệ số tăng tốc của CAPE, có một số hướng phát triển sau:

1. Trên các hệ thống mạng máy tính sử dụng các bộ vi xử lý đa lõi, tìm các phương pháp để thực hiện song song các đoạn mã song song OpenMP trên mỗi máy tính toán và song song hóa quá trình nhận kết quả xử lý cũng như phân phối kết quả tổng hợp tại máy chính.
2. Trên các hệ thống có sử dụng GPU (Graphic Processing Unit), tìm cách để khai thác khả năng của GPU trong việc thực hiện chương trình ứng dụng.

3. Thuận lợi

- Đã có được một số điều kiện vật chất căn bản để tiến hành thử nghiệm CAPE trong khi tiến hành nghiên cứu. Các phòng máy tính ở trường Sư phạm, ĐH Huế có cấu hình các máy tính thành phần tương đối mạnh, hoạt động ổn định và được nối mạng để thử nghiệm CAPE.
- Đã có được hợp tác nghiên cứu quốc tế bao gồm một nhóm của TS Eric Renault ở Pháp và nhóm của TS Hà Viết Hải ở Việt Nam.

4. Khó khăn

- Chưa có hệ thống mạng tốc độ cao để thử nghiệm CAPE trên môi trường này.
- Khó tìm được thêm người đồng nghiên cứu cũng như các học viên cao học, nghiên cứu sinh để tiếp tục phát triển CAPE.

IV. KẾT LUẬN

CAPE, với những nguyên lý cơ bản và các kết quả thực nghiệm ban đầu đã chứng tỏ được tiềm năng to lớn để trở thành một cài đặt tương thích hoàn toàn và có hiệu năng cao cho chuẩn OpenMP trên các kiến trúc sử dụng bộ nhớ phân tán. Các nghiên cứu vẫn còn đang được tiếp tục theo nhiều hướng để phát triển CAPE theo hướng cài đặt hoàn chỉnh chuẩn OpenMP cũng như theo hướng khai thác được các kiến trúc mới của các bộ xử lý để tăng thêm hiệu năng hoạt động.

TÀI LIỆU THAM KHẢO

[1] Blaise Barney, Lawrence Livermore, *Introduction to Parallel Computing*, https://computing.llnl.gov/tutorials/parallel_comp/#Whatis, (truy cập 29/11/2015).

[2] OpenMP Architecture Review Board, *OpenMP Application Programming Interface*, http://www.openmp.org/mp-documents/openmp-4.5.pdf, (truy cập 29/11/2015).

[3] Christine Morin, Renaud Lottiaux, Geoffroy Vallée, Pascal Gallard, Gaël Utard, R. Badrinath and Louis Rilling., Kerrighed: *A Single System Image Cluster Op-erating System for High Performance Computing*, Euro-Par 2003 Parallel Pro-cessing, Klagenfurt, Austria, LNCS 2790, pp. 1291–1294(2003).

[4] Mitsuhisa Sato, Hiroshi Harada, Atsushi Hasegawa and Yutaka 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).

[5] Sven Karlsson, Sung-Woo Lee, Mats Brorsson, Sahni Sartaj, Viktor K. Prasanna and Shukla Uday, *A fully compliant OpenMP implementation on software distributed shared memory*. Proceedings of the International Conference on High Performance Computing, Bangalore, India, LNCS 2552, pp. 195–206 (2002).

[6] Ayon Basumallik and Rudolf Eigenmann, *Towards automatic translation of OpenMP to MPI*, Proceedings of the 19th annual international conference on Supercomputing, Cambridge, MA, pp. 189–198 (2005).

[7] Beniamino Di Martino, Dieter Kranzlmüller and Jack Dongarra, *Implementing OpenMP for clusters on top of MPI*, Proceedings of 12th European PVM/MPI Users' Group Meeting Sorrento, LNCS, Volume 3666, pp 148–155 (2005).

[8] Lei Huang and Barbara Chapman and Zhenying Liu, *Towards a more efficient implementation of OpenMP for clusters via translation to Global Arrays*, Journal of Parallel Computing 31, pp 1114–1139(2005).

[9] Jay P. Hoeflinger, *Extending OpenMP to Clusters - White paper*, http://assets.devx.com/goparallel/19403.pdf (truy cập 29/11/2015).

[10] Jame S. Plank, *An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance*, Technical Report UTCS-97-372, Department of Computer Science, University of Tennessee, Juilly (1997).

[11] Viet Hai Ha and Éric Renault, *Discontinous Incremental: A New Approach Towards Extremely Lighweight Checkpoints*, Proceedings of the IEEE Incremental Symposium on Computer Networks and Distributed System 2011 (CNDS 2011), Tehran, Iran (2011).

[12] Eric Renault, *Distributed Implementation of OpenMP Based on Checkpointing Aided Parallel Execution*, International Workshop on OpenMP (IWOMP), Beijing, China, LNCS 4935, pp.183-193(2007).

[13] Viet Hai Ha and Éric Renault, *Improving Performance of CAPE using Discontinuous Incremental Checkpointing*, Proceedings of the International Conference on High Performance and Communications 2011 (HPCC-2011), Banff, Canada (2011).