ĐẠI HỌC HUẾ

TRƯỜNG ĐẠI HỌC KHOA HỌC

ĐỖ XUÂN HUYỀN

CẢI TIẾN MÔ HÌNH CAPE CHO HỆ THỐNG TÍNH TOÁN ĐA LÕI

NGÀNH: KHOA HỌC MÁY TÍNH

MÃ SỐ: 9480101

TÓM TẮT LUẬN ÁN TIẾN SĨ KHOA HỌC MÁY TÍNH

HUẾ - NĂM 2023

Công trình được hoàn thành tại:
Trường Đại học Khoa học, Đại học Huế

Người hướng dẫn khoa học:

1. TS. Hà Viết Hải, Đại học Sư phạm, Đại học Huế
2. GS. Éric Renault, LIGM, University Gustave Eiffel, CNRS, ESIEE Paris, Marne la Vallee, France.

Phản biện 1: .....

.....

Phản biện 2: .....

.....

Phản biện 3: .....

.....

Luận án sẽ được bảo vệ tại Hội đồng chấm luận án cấp Đại học Huế họp tại..

.....

.....

vào lúc.....giờ.....ngày.....tháng.....năm .....

Có thể tìm hiểu luận án tại thư viện .....

.....

.....

.....

MỞ ĐẦU

OpenMP là một API có mục tiêu là bổ sung khả năng lập trình song song cho các chương trình gốc được viết ở ngôn ngữ C, C++ và Fortran, chạy trên kiến trúc sử dụng bộ nhớ chia sẻ (máy tính có nhiều CPU và/hoặc CPU đa lõi). OpenMP đơn giản, dễ học, dễ dùng và cung cấp hiệu năng cao nên nhanh chóng trở thành chuẩn lập trình song song cho các kiến trúc này. Tuy nhiên, OpenMP không chạy được trên các hệ thống sử dụng bộ nhớ phân tán (như cluster, grid). Điều này dẫn đến một ý tưởng và động lực cho nhiều nghiên cứu để chuyển đổi OpenMP lên các kiến trúc sử dụng bộ nhớ phân tán. Thực hiện ý tưởng trên, trong nhiều năm qua, đã có nhiều nhóm nghiên cứu cố gắng thực hiện việc đưa OpenMP trên hệ thống máy tính sử dụng bộ nhớ phân tán bằng cách xây dựng các trình biên dịch để dịch tự động chương trình OpenMP thành chương trình có khả năng chạy trên các hệ thống này. Đồng thời với việc xây dựng các chương trình biên dịch, một số tiếp cận còn đòi hỏi phải xây dựng một nền tảng (platform) bổ sung cho hệ thống để có thể chạy được các chương trình đã được biên dịch. Tuy nhiên, ngoại trừ CAPE [12]-[19], chưa có công trình nào thành công trên cả hai mặt là tương thích hoàn toàn với OpenMP và có hiệu năng cao. Một số công trình nghiên cứu nổi bật có thể nhắc đến như SSI [6]; Cluster OpenMP [11]; SCASH [7]; sử dụng mô hình HLRC [24]; biên dịch thành MPI [8][9]; sử dụng Global Array [10]; libMPNode [30] cải tiến SSI; OMPC [34] sử dụng trình biên dịch riêng. Hiện tại, OpenMP nói chung và phát triển OpenMP trên các hệ thống sử dụng bộ nhớ phân tán nói riêng là chủ đề chính của chuỗi hội thảo quốc tế hàng năm IWOMP, với lần thứ 19 sẽ được tổ chức tại Đại học Bristol, Anh vào tháng 9 năm 2023 (https://www.iwomp.org/).

CAPE (Checkpointing Aided Parallel Execution) là một tiếp cận dựa trên kỹ thuật chụp ảnh tiến trình để cài đặt API OpenMP trên các hệ thống máy tính sử dụng bộ nhớ phân tán. CAPE do GS. Éric Renaut phát minh. Phiên bản CAPE thứ hai do TS. Hà Viết Hải, TS. Trần Văn Long phát triển đã cải tiến rõ rệt hiệu năng của CAPE, làm cho nó tiệm cận với hiệu năng của MPI là phương pháp có khả năng cung cấp hiệu năng cao nhất cho lập trình song song trên các thống phân tán. Tuy nhiên, ở cả hai phiên bản của CAPE, các máy tính tham gia trong hệ thống đều được khai thác theo quan điểm sử dụng các bộ xử lý đơn lõi, do đó chưa khai thác được hết các khả năng của các bộ vi xử lý đa

lõi là trường hợp phổ biến hiện nay. Điều này dẫn đến việc lãng phí tài nguyên khi sử dụng CAPE trên hệ thống máy tính có CPU đa lõi rất phổ biến hiện nay. Để khắc phục được hạn chế này, mô hình hoạt động của CAPE cần phải được tổ chức theo hướng cho phép chương trình chạy song song trên từng nút phụ (song song mức thứ 2) để khai thác tốt hơn tài nguyên hệ thống và từ đó tăng tốc độ tính toán. Đây chính là động lực để luận án đề xuất mô hình hoạt động CAPE mới nhằm khắc phục những hạn chế chưa khai thác tài nguyên hệ thống máy tính sử dụng CPU đa lõi. Để thực hiện mục tiêu này, những vấn đề sau cần được tiếp tục phát triển: (1) Phát triển một mô hình CAPE mới để song song hóa một cách hiệu quả các đoạn mã tính toán trên từng nút nút phụ (nút tính toán); (2) Phát triển kỹ thuật chụp ảnh tiến trình cũng như giải quyết vấn đề chia sẻ đồng bộ dữ liệu của CAPE cho phù hợp với các mô hình thực hiện được song song hóa 2 mức ở mục trên.

Từ những trình bày trên, đề tài **"Cải tiến mô hình CAPE cho hệ thống tính toán đa lõi"** trở nên có tính thời sự và cấp thiết để đáp ứng nhu cầu cung cấp một giải pháp cài đặt tương thích hoàn toàn OpenMP và có hiệu năng cao trên các kiến trúc sử dụng bộ nhớ phân tán.

Mục tiêu nghiên cứu của luận án là phát triển mô hình CAPE trên hệ thống tính toán đa lõi để nâng cao hiệu năng hoạt động của hệ thống.

Cụ thể:

- *Mục tiêu 1:* Nghiên cứu và đề xuất mô hình hoạt động của CAPE trên hệ thống tính toán đa lõi.
- *Mục tiêu 2:* Nghiên cứu và xây dựng phiên bản kỹ thuật chụp ảnh đa tiến trình phù hợp với mô hình CAPE trên hệ thống tính toán đa lõi.
- *Mục tiêu 3:* Nghiên cứu và đề xuất giải pháp chia sẻ dữ liệu của CAPE trên hệ thống tính toán đa lõi.
- *Mục tiêu 4:* Phát triển hệ thống phần mềm tương ứng với mô hình CAPE mới đề xuất và đánh giá hiệu năng của nó so với mô hình CAPE trước đó và với MPI (kỹ thuật cung cấp hiệu năng tốt nhất hiện nay trên các hệ thống phân tán).

CHƯƠNG I : TỔNG QUAN NGHIÊN CỨU

1.1. Tính toán hiệu năng cao

Tính toán Hiệu năng cao (High Performance Computing (HPC)) thường đề cập đến việc xử lý các phép tính phức tạp ở tốc độ cao trên nhiều máy chủ song song.

1.2. Tính toán song song

Tính toán song song (Parallel Computing) hay xử lý song song (Parallel Processing): là quá trình xử lý thông tin trong đó nhấn mạnh việc nhiều đơn vị dữ liệu được xử lý đồng thời bởi một hay nhiều bộ xử lý để giải quyết một bài toán.

Hệ số Tăng tốc (speedup) : hệ số tăng tốc của chương trình song song là tỉ số giữa thời gian thực hiện trong tình huống sử dụng chương trình tuần tự và thời gian thực hiện cũng công việc đó của chương trình song song.

Theo luật Amdahl [36], dự đoán về hệ số tăng tốc độ tối đa của chương trình xử lý song song được tính theo công thức sau:

$$S_{\text{latency}} = \frac{1}{(1-p) + \frac{p}{s}}$$

Trong đó

$S_{latency}$: hệ số tăng tốc lý thuyết toàn cục

p: tỉ lệ có thể song song hóa của thuật toán tuần tự

s: là số bộ xử lý song song.

1.3. Máy tính đa CPU, CPU đa lõi và đa luồng

Các hệ thống máy tính xử lý song song có thể được chia thành hai loại theo các cách tổ bộ nhớ khác nhau, một loại sử dụng hệ thống bộ nhớ chia sẻ (shared-memory) và loại kia sử dụng hệ thống bộ nhớ phân tán (distributed memory). Đối với hệ thống sử dụng bộ nhớ chia sẻ, có các kiến trúc thông dụng là máy tính đa CPU (muti-CPU), máy tính sử dụng CPU đa lõi (multi-core) và loại kết hợp cả 2 kiến trúc này.

Đa luồng (Multi-threading) là khả năng xử lý nhiều luồng cùng lúc của CPU. Khi có nhiều nhiệm vụ khác nhau, CPU có thể làm việc với chúng một cách song song

(parallel) đối với CPU chỉ có một lõi hoặc làm việc đồng thời (concurrent) đối với CPU đa lõi. Lõi là nơi thực hiện các nhiệm vụ và mỗi lõi chỉ chạy 1 nhiệm vụ tại một thời điểm. Chính vì lí do này mà có càng nhiều lõi thì CPU càng thực hiện được nhiều nhiệm vụ cùng lúc. Công nghệ siêu phân luồng của Intel sẽ cải thiện hiệu suất xử lý CPU lên tới 30% [75].

Hầu hết các phiên bản hệ điều hành phổ biến hiện nay như Windows, phiên bản phân phối của Linux đều hỗ trợ cho CPU đa lõi cũng như đa luồng.

Mục tiêu và phạm vi nghiên cứu cải tiến CAPE của luận án này hướng đến áp dụng cho hệ thống máy tính sử dụng CPU đa lõi. Cần lưu ý rằng CAPE, tương tự như MPI, là ngôn ngữ trừu tượng bậc cao có bao gồm thư viện runtime ở tầng ứng dụng, không can thiệp điều khiển trực tiếp phần cứng của CPU đa lõi mà sử dụng lại các dịch vụ của hệ điều hành cung cấp để chạy chương trình.

1.4. OpenMP

OpenMP là một giao diện lập trình (API) cung cấp một mức trừu tượng hóa cao để viết các chương trình song song cho các hệ thống tính toán hiệu năng cao với ưu điểm dễ học, dễ sử dụng. Để viết chương trình song song với OpenMP, lập trình viên có thể bắt đầu bằng cách viết một chương trình tuần tự trên ngôn ngữ gốc (C/C++ hoặc Fortran), sau đó thêm dần vào các chỉ thị của OpenMP để chỉ định những phần việc nào cần được thực hiện song song. Việc chia sẻ dữ liệu cũng như đồng bộ dữ liệu được tiến hành một cách ngầm định hoặc tường minh qua các chỉ thị đơn giản. Với cách tiếp cận này, OpenMP rất dễ học, dễ sử dụng, đòi hỏi ít công sức lập trình. Tuy nhiên, OpenMP vẫn chỉ mới được cài đặt một cách hoàn chỉnh cho các kiến trúc sử dụng bộ nhớ chia sẻ do sự phức tạp của việc cài đặt tất cả các yêu cầu của OpenMP trên các kiến trúc sử dụng bộ nhớ khác.

1.5. Các công trình nổi bật về chuyển đổi OpenMP lên hệ thống sử dụng bộ nhớ phân tán.

Các công trình nổi bật về chuyển đổi OpenMP lên hệ thống bộ nhớ phân tán có thể liệt kê là: Phương pháp sử dụng SSI làm bộ nhớ chung cho tất cả cácluồng; Phương pháp ánh xạ một phần không gian nhớ của luồng; Phương pháp sử dụng mô hình HLRC;

Phương pháp kết hợp với MPI; Phương pháp dựa trên Mảng toàn cục (Global Array - GA); Phương pháp CAPE đơn luồng.

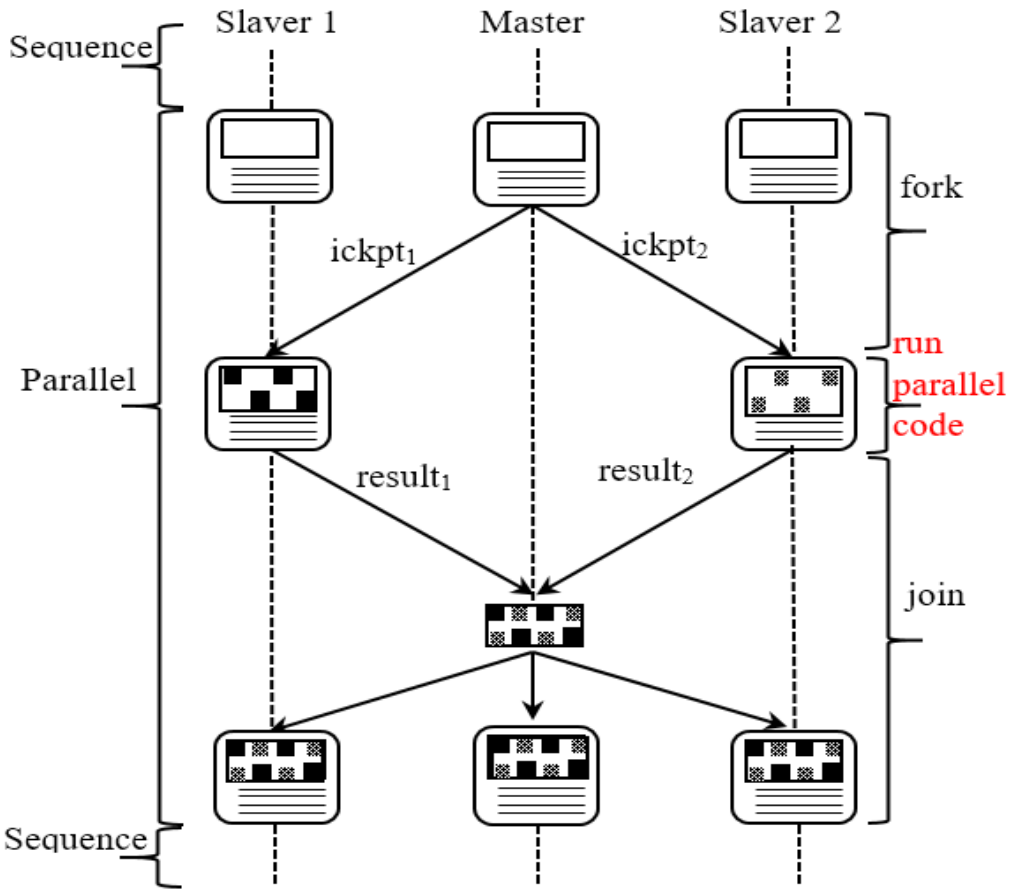
1.6. Tổng hợp đánh giá về các phương pháp chuyển đổi OpenMP trên kiến trúc bộ nhớ phân tán

Luận án đã tổng hợp, đánh giá ưu nhược điểm của các phương pháp và chọn hướng tiếp tục cải tiến CAPE theo hướng song song hóa phần việc tại các nút phụ của hệ thống.

1.7. Phương pháp CAPE đơn luồng

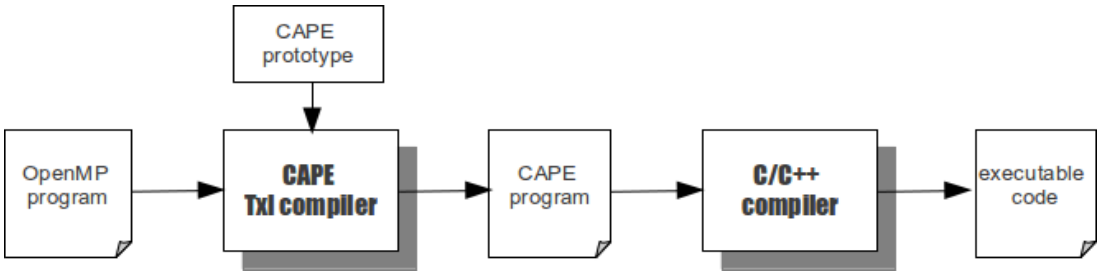
Trong những phiên bản CAPE được tiến hành ở các nghiên cứu trước luận án này, các phần việc thực hiện tại các nút phụ được đảm trách bởi một tiến trình đơn luồng. Vì vậy, trong luận án này chúng tôi gọi chung CAPE ở các phiên bản trước là CAPE đơn luồng.

Mô hình hoạt động của CAPE được minh hoạ trong Hình 1.6.



Hình 1.6. Mô hình hoạt động của CAPE đơn luồng

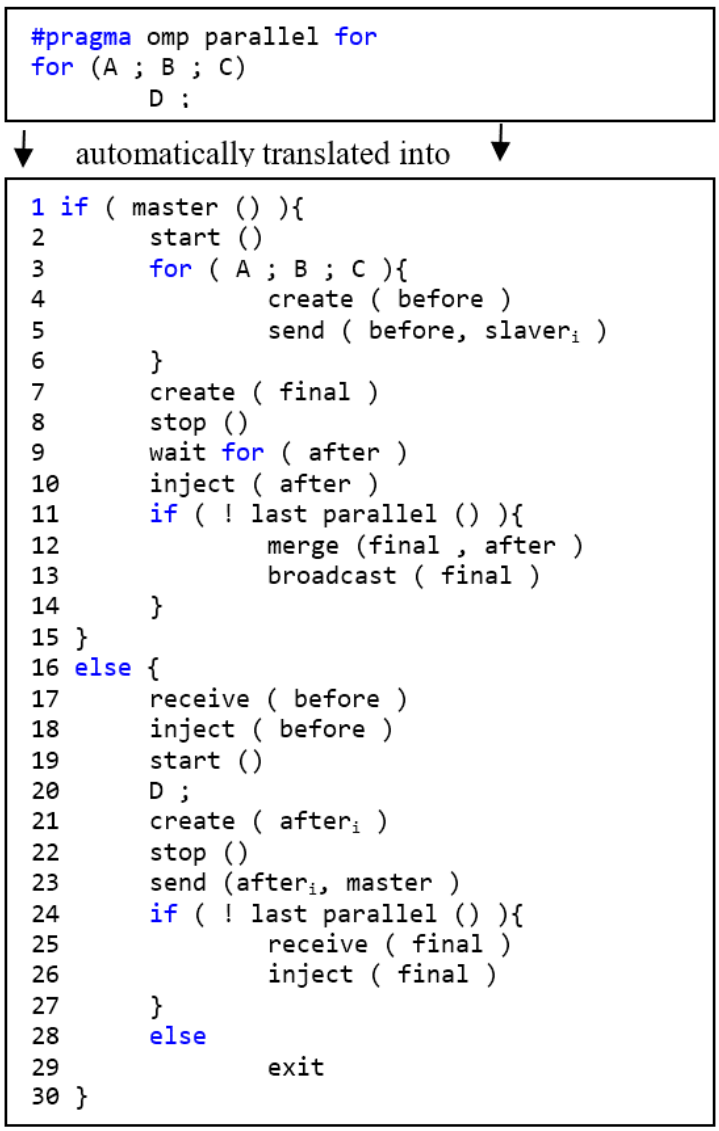
Quy trình biên dịch mã từ OpenMP sang CAPE được mô tả trong Hình 1.7.



Hình 1.7. Quy trình biên dịch chương trình OpenMP thành chương trình CAPE

Trong quy trình trên, chương trình OpenMP ban đầu được trình biên dịch của CAPE dịch thành chương trình nguồn dạng CAPE, trong đó các cấu trúc song song của OpenMP được thay thế bằng các hàm CAPE (ở dạng mã C) nhờ các khuôn dạng chuyển đổi của CAPE. Sau đó, chương trình ở dạng CAPE (không còn chứa các chỉ thị OpenMP) được tiếp tục biên dịch thành chương trình thực thi bằng một chương trình dịch C/C++ thông thường.

Hình 1.8 trình bày một khuôn dạng để chuyển đổi cấu trúc omp parrallel for của OpenMP thành chương trình nguồn dạng CAPE.



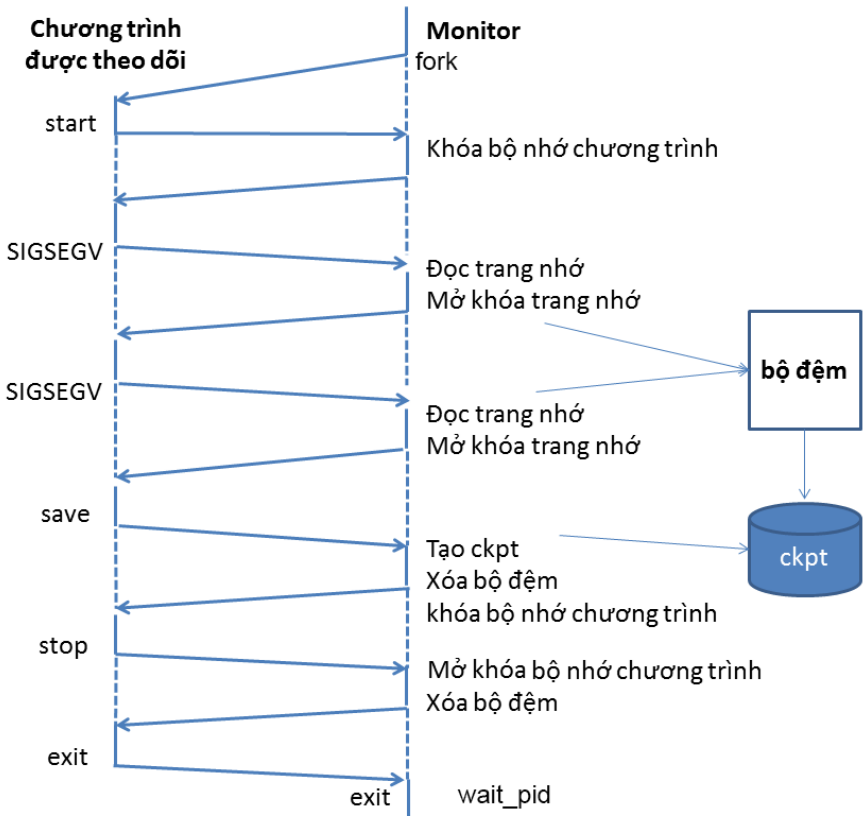
Hình 1.8. Khuôn dạng biên dịch cấu trúc `omp parallel for` trong CAPE đơn luồng

Kỹ thuật chụp ảnh tiến trình áp dụng cho CAPE đơn luồng

Chụp ảnh tiến trình [35] là kỹ thuật chụp và lưu trữ trạng thái của một tiến trình đang vận hành sao cho nó có khả năng khôi phục lại trạng thái ở các thời điểm sau đó. Khi một chương trình được chạy, trạng thái của nó được thể hiện qua các giá trị trong không gian nhớ (memory space) của tiến trình, giá trị trong các thanh ghi và trạng thái của hệ điều hành.

Kỹ thuật chụp ảnh tiến trình đầy đủ sẽ lưu toàn bộ thông tin của tiến trình vào trong mỗi ảnh chụp tiến trình. Kỹ thuật chụp ảnh tiến trình gia tăng chỉ lưu mỗi ảnh chụp những giá trị đã được thay đổi so với ảnh chụp ngay trước nó nhằm kích thước ảnh chụp.

Để phục vụ một cách tối ưu cho CAPE, Kỹ thuật chụp ảnh tiến trình gia tăng rời rạc DICKPT (Discontinuos Incremental Checkpointing) đã được phát triển vào những năm 2009-2011 [13]. Kỹ thuật này dựa trên cơ sở kỹ thuật chụp ảnh tiến trình gia tăng và cung cấp thêm khả năng chụp ảnh từng phần riêng biệt tương ứng với từng đoạn mã rời rạc của chương trình khi nó được thực hiện.



Hình 1.12. Nguyên tắc hoạt động của của Kỹ thuật chụp ảnh tiến trình gia tăng rời rạc

Cấu trúc dữ liệu cho ảnh chụp tiến trình

Ảnh chụp tiến trình do DICKPT tạo ra chứa hai bộ dữ liệu: a) giá trị thanh ghi - tức là tất cả các giá trị thanh ghi tại thời điểm chụp, và b) không gian địa chỉ của chương trình được giám sát - tức là tất cả dữ liệu của chương trình được giám sát có sửa đổi so với chụp ảnh trước đó.

Phần lớn dữ liệu của ảnh chụp tiến trình là tập trung ở bộ dữ liệu lưu trữ dữ liệu thay đổi của chương trình.

Phần tổ chức và xử lý dữ liệu của ảnh chụp tiến trình là word (từ) (4- byte kích thước bộ nhớ để lưu trữ 1 từ). Trong [12] đã đề xuất 4 cách tổ chức bộ nhớ tối ưu kích thước của ảnh chụp tiến trình: *Cấu trúc đơn từ - Single data (SD); Cấu trúc dữ liệu liên tiếp -*

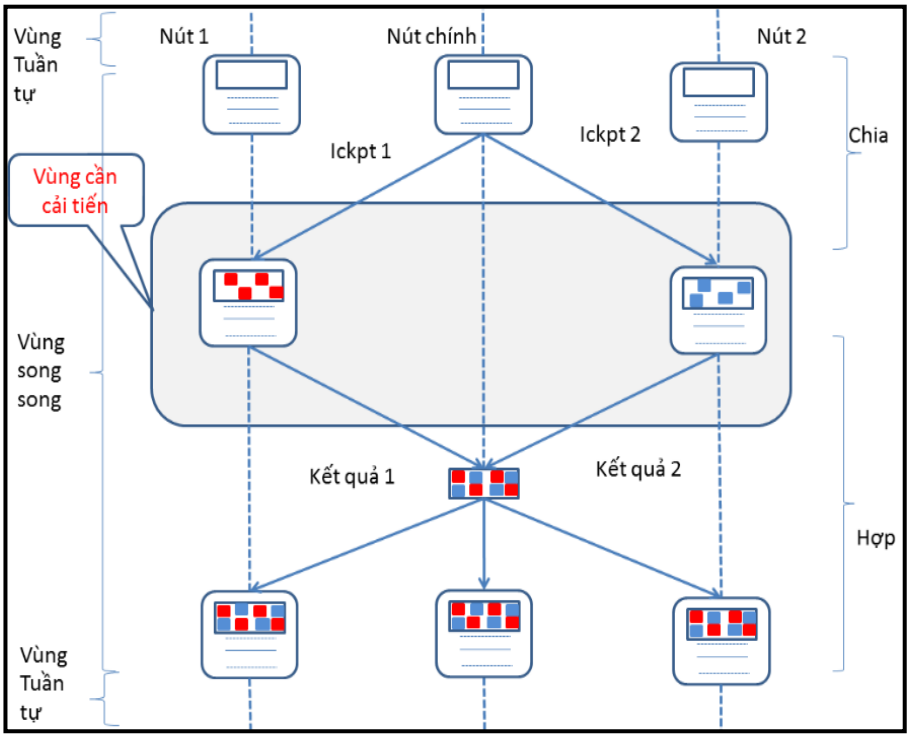
Several successive data (SSD); Cấu trúc sơ đồ ma trận - Many data (MD); Cấu trúc toàn trang nhớ- Entire page (EP).

Các kết quả của chương này được công bố ở công trình [CT1].

CHƯƠNG II: CAPE TRÊN HỆ THỐNG TÍNH TOÁN ĐA LÕI

2.1. Nguyên lý chung

Mô hình hoạt động hiện tại của CAPE đơn luồng được trình bày ở Hình 2.1.



Hình 2.1: Mô hình hoạt động của CAPE và vùng cần cải tiến để khai thác khả năng của các bộ xử lý đa lõi

Để tăng được hiệu năng của chương trình, rõ ràng trong khoảng thời gian thực hiện tính toán ở các nút phụ, cần song song hóa các đoạn mãtương ứng, tức là phải tìm cách để nó được chạy bởi đa tiến trình hoặc đa luồng (song song mức thứ 2). Trên Hình 2.1, vùng khoanh tô màu nền đậm hơn chính là vùng cần cải tiến. Dựa theo nguyên tắc hoạt động của CAPE hiện tại, có 3 phương pháp có thể thực hiện điều này: 1) 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 (Phương pháp 1); 2) Sử dụng song song trên mỗi nút phụ bằng

cách cho thực hiện song song chương trình ứng dụng trên các máy ảo, tạo nhiều máy ảo trên từng máy thật (Phương pháp 2); và 3) Song song hóa đoạn mã tính toán trên nút phụ theo mô hình đa luồng, nghĩa là song song 2 mức: mức 1 liên quan đến phân chia và chạy đồng thời các tác vụ trên nhiều máy, trong khi mức 2 liên quan đến việc sử dụng đa luồng trên mỗi nút phụ để chạy song song phần công việc được chia cho nó (Phương pháp 3).

Phương pháp 1) đã được thử nghiệm và công bố kết quả ở [64] với kết quả là đã tăng được hiệu năng hoạt động nhưng hệ thống hoạt động không ổn định. Luận án này đã thực hiện nghiên cứu và thực nghiệm cả phương pháp 2) và 3), với chi tiết được trình bày trong các phần tiếp theo.

2.2. Mô hình CAPE song song hai mức dựa trên phương pháp sử dụng nhiều máy ảo

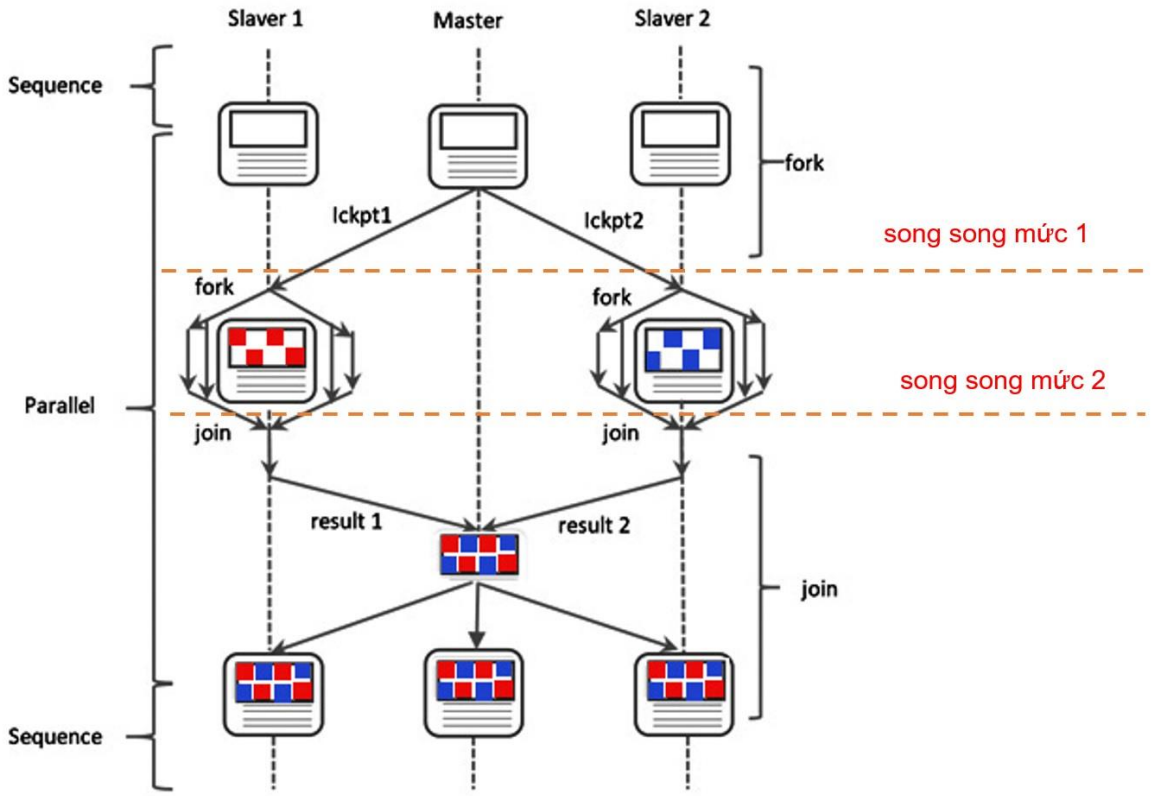
Luận án đã cài đặt và đáng giá thực nghiệm theo hướng này. Tuy nhiên kết quả về mặt hiệu năng là không như mong đợi, thời gian thực hiện trong trường hợp sử dụng nhiều máy ảo luôn lớn hơn thời gian sử dụng một máy vật lý đơn luồng (CAPE đơn luồng).

2.3. Mô hình CAPE song song hai mức đa luồng[1]

2.3.1. Ý tưởng thực hiện

Nguyên tắc căn bản của phương pháp này là cài đặt các cấu trúc song song của OpenMP qua 2 mức như minh họa ở Hình 2.7.

[1] Đa luồng (Multithreading) là khả năng xử lý nhiều luồng cùng lúc của CPU. Khi có nhiều nhiệm vụ khác nhau, CPU có thể làm việc với chúng một cách song song (parallel) đối với CPU chỉ có một lõi hoặc làm việc đồng thời (concurrent) đối với CPU đa lõi. Lõi là nơi thực hiện các nhiệm vụ và mỗi lõi chỉ chạy 1 nhiệm vụ tại một thời điểm.



Hình 2.7. Mô hình hoạt động song song 2 cấp của CAPE đa luồng

2.3.1. Khuôn mẫu dịch cấu trúc `omp parallel` trong CAPE đa luồng

Như trong Hình 2.8, đoạn mã song song của OpenMP, theo mô hình CAPE song song hai mức đa luồng (gọi tắt là CAPE đa luồng), được biên dịch thành một tập hợp các lệnh thuộc dạng ngôn ngữ C/C++ chứa cả chỉ thị OpenMP và CAPE. Điều này cung cấp khả năng chạy ở 2 mức song song. Mức 1 liên quan đến phân chia và chạy đồng thời các tác vụ trên nhiều máy, trong khi mức 2 liên quan đến việc sử dụng nhiều luồng trên mỗi nút phụ.

```
#pragma omp parallel for
for (A ; B ; C)
        D ;
```

automatically translated into

```
1 if ( master () ){
2       start ()
3       for ( A ; B ; C ){
4               create ( before )
5               send ( before, slaver_i )
6       }
7       create ( final )
8       stop ()
9       wait for ( after )
10      inject ( after )
11      if ( ! last parallel () ){
12              merge (final , after )
13              broadcast ( final )
14      }
15 }
16 else {
17      receive ( before )
18      inject ( before )
19      start ()
20      #pragma omp parallel for
21      for (A_i ; B_i ; C_i)
22              D ;
23      create ( after_i )
24      stop ()
25      send (after_i, master )
26      if ( ! last parallel () ){
27              receive ( final )
28              inject ( final )
29      }
30      else
31              exit
32 }
```

Hình 2.8. Khuôn mẫu dịch cấu trúc `parallel for` trong CAPE đa luồng

2.4. Phân tích hiệu năng hoạt động của CAPE đa luồng

2.4.1. Hệ số tăng tốc lý thuyết theo luật Amdahl

Đối với CAPE đa luồng thì đoạn mã song song này thuộc dạng một tiến trình đa luồng. Do đó, số mức song song tối ưu sẽ bằng với số lõi của CPU và khi đó khả năng tăng tốc tối đa của phần này sẽ xấp xỉ với số lõi theo luật Amdahl, nếu bỏ qua các chi phí liên quan đến việc đồng bộ dữ liệu, tổ chức và thực hiện tiến trình đa luồng.

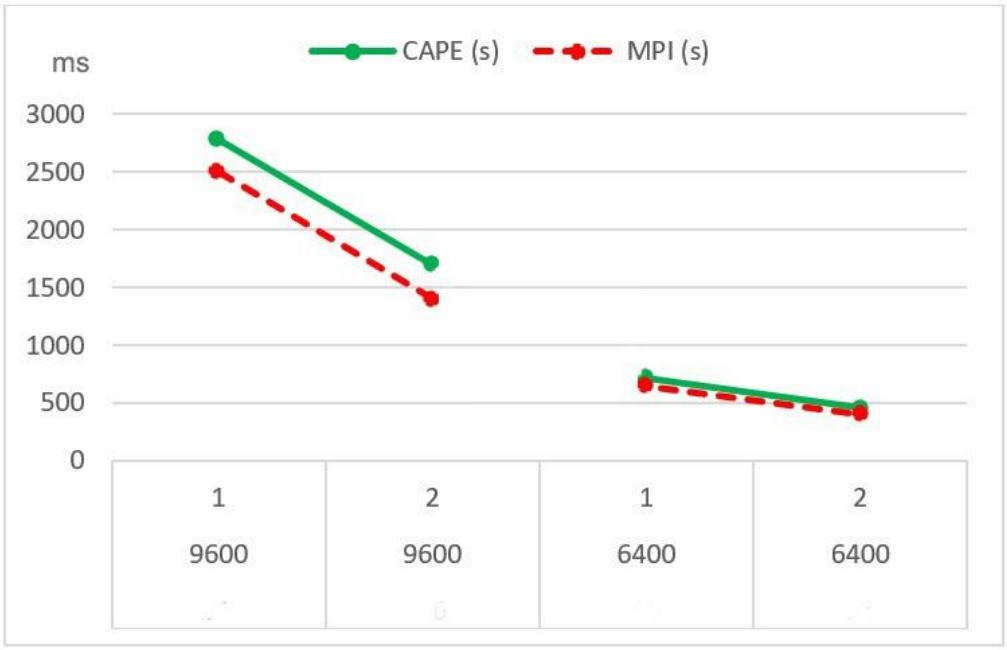
2.4.2. Kết quả thực nghiệm và đánh giá

2.4.2.1. Hệ thống thực nghiệm và bài toán thực nghiệm

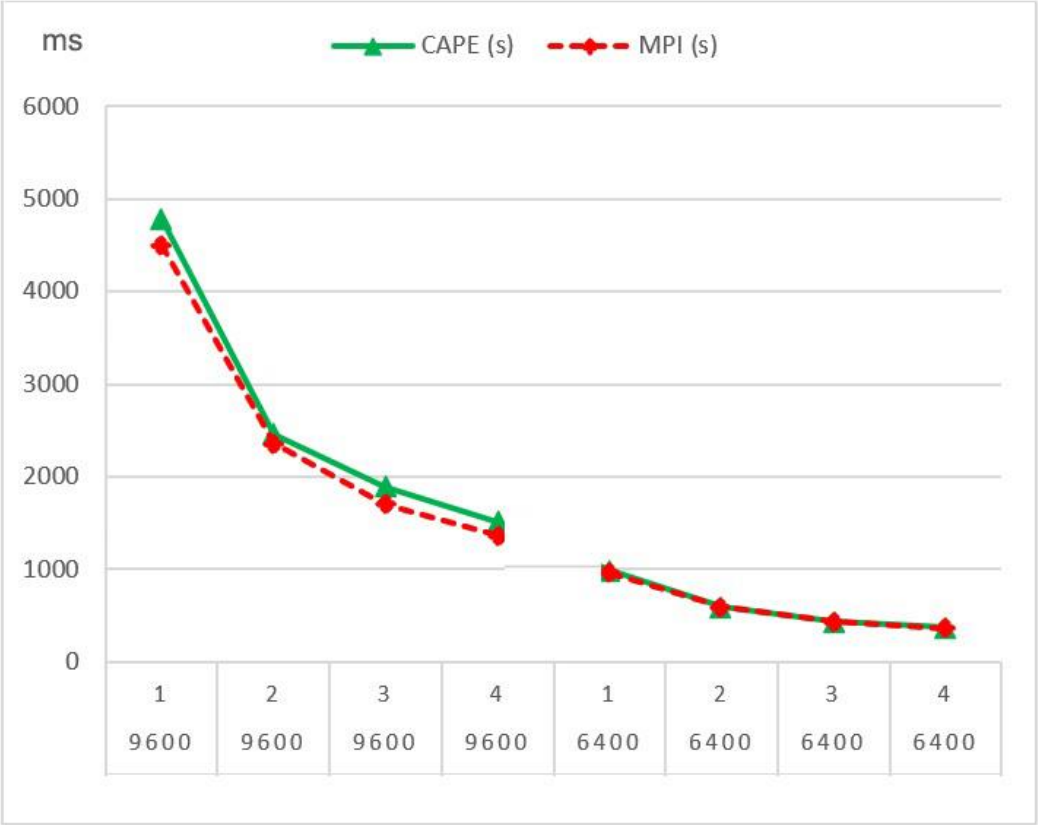
Các thực nghiệm đã tiến hành chạy chương trình nhân hai ma trận vuông có các kích thước lần lượt là 9600x9600 và 6400x6400, với số luồng tại các nút slave biến thiên từ 1 đến 2 trên hệ thống 1 (cluster gồm 16 nút slave); và 1 đến 4 luồng trên hệ thống 2 (cluster gồm 8 nút slave).

2.4.2.2. Kết quả thực nghiệm và phân tích, đánh giá

2.4.2.2.1. Kết quả và đánh giá theo thời gian thực hiện



Hình 2.9. Thời gian thực hiện trên cluster 16 máy biến thiên theo số lõi sử dụng



Hình 2.10. Thời gian thực hiện trên trên cluster 8 máy biến thiên theo số lõi sử dụng

So sánh giữa CAPE đa luồng và CAPE đơn luồng

Đối với cả 2 hệ thống và 2 kích thước bài toán trên đó (9600x9600 và 6400x6400), thời gian thực hiện chương trình của CAPE đa luồng luôn bé hơn của CAPE đơn luồng. Điều này chứng minh một cách rõ ràng mô hình hoạt động đa luồng đã mang lại hiệu năng thực hiện tốt hơn mô hình đơn luồng trước đây.

Với hệ thống cluster 8 máy, biểu đồ ở Hình 2.10 cho thấy độ dốc tăng tốc từ chuyển đổi 1 lõi thành 2 lõi luôn lớn hơn độ dốc khi tăng từ 2 lõi lên 3 lõi, cũng như từ 3 lõi lên 4 lõi. Điều này có thể là do cấu trúc CPU Intel(R) Core(TM) i3, tuy được xem là có 4 lõi nhưng thực chất chỉ có 2 lõi vật lý và mỗi lõi chứa 2 siêu phân luồng.

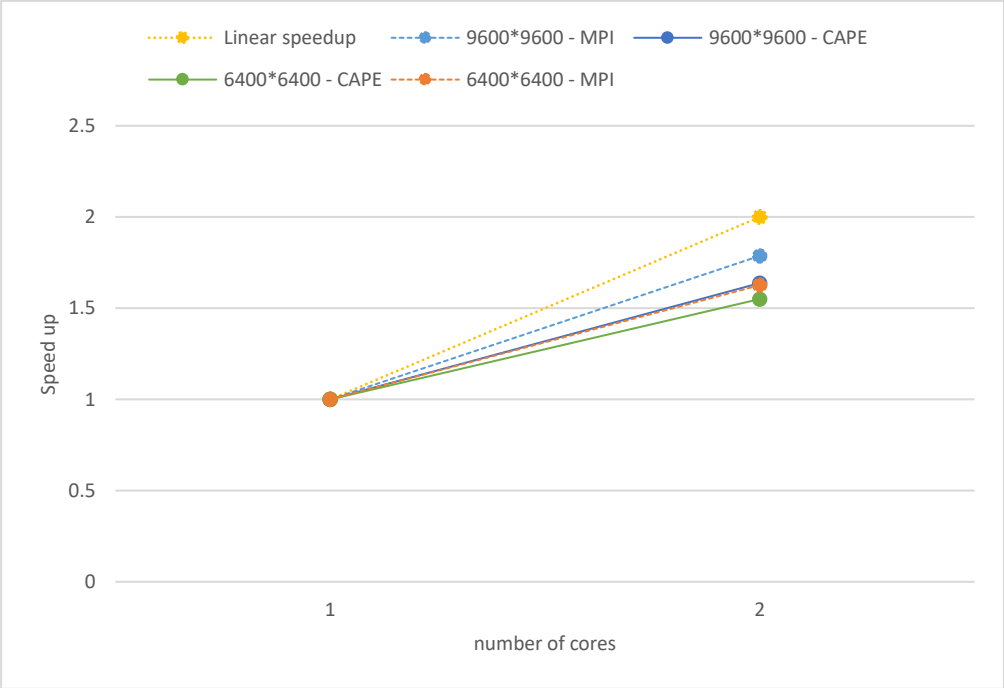
So sánh giữa CAPE và MPI

Trong tất cả các trường hợp, thời gian chạy chương trình theo MPI luôn nhanh hơn hơn CAPE, nhưng với một tỷ lệ tương đối ổn định, thể hiện qua đồ thị của chúng gần như song song với nhau. Điều này cũng chứng tỏ CAPE vận hành ổn định trong trường hợp tăng hoặc giảm số lõi CPU sử dụng. Hơn nữa, mức chênh lệch thời gian chạy chương trình của CAPE và MPI chỉ ở mức xấp xỉ 8%. Mức chênh lệch này là

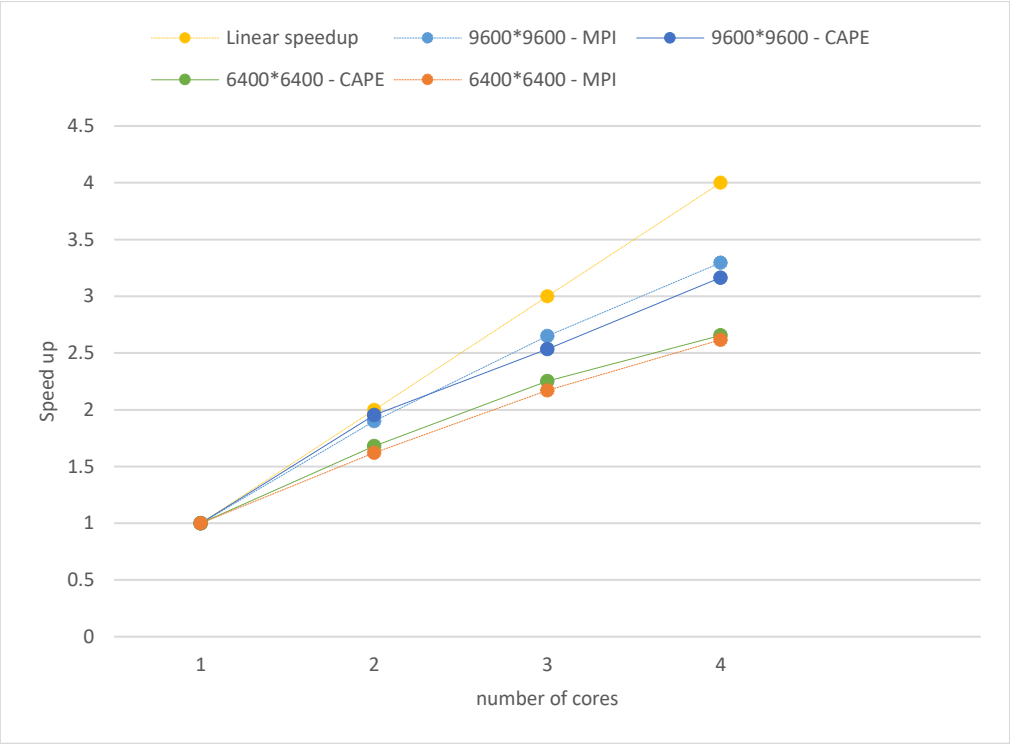
không cao và có nguyên nhân từ việc CAPE luôn tốn chi phí thời gian để thực hiện việc giám sát và chụp ảnh tiến trình. Với mức chênh lệch này, có thể nói CAPE đã tiệm cận được với phương pháp lập trình song song trên hệ thống sử dụng bộ nhớ phân tán có khả năng cung cấp hiệu năng cao nhất là MPI. Kết hợp với đặc tính nổi bật của OpenMP là tính dễ học, dễ sử dụng, tiết kiệm công sức và thời gian lập trình, có thể nói CAPE có khả năng cung cấp một cách thức lập trình song song ưu việt trên hệ thống sử dụng bộ nhớ phân tán.

2.4.2.2.2. Kết quả và đánh giá theo hệ số tăng tốc

Kết quả tính toán được biểu diễn trên các Hình 2.11 và Hình 2.12.



Hình 2.11. Hệ số tăng tốc theo số lõi CPU sử dụng trên hệ thống 16 nút phụ



Hình 2.12. Hệ số tăng tốc theo số lõi CPU sử dụng trên hệ thống 8 nút phụ

So sánh giữa CAPE đa luồng và CAPE đơn luồng

Trong cả hai hình trên, đồ thị của CAPE đơn luồng chỉ có 1 điểm duy nhất tương ứng với trường hợp sử dụng 1 lõi. Đối sánh với các trường hợp CAPE đa luồng sử dụng 2 đến 4 lõi, ta thấy chúng gần như tạo thành 1 đường tăng tốc tăng tuyến tính. Một cách chính xác, tốc độ tăng tốc của CAPE và MPI cao hơn trong trường hợp sử dụng 2 lõi là 1,90-1,95 lần khi so sánh với trường hợp sử dụng 1 lõi. Đồng thời khi sử dụng 4 lõi của CPU (thực chất là 2 lõi thực và 2 siêu phân luồng) thì hệ số tăng tốc vẫn cao, có thể lên tới 3.16 so với trường hợp sử dụng 1 lõi CPU. Các đường tăng tốc này gần như tiệm cận với đường tăng tốc lý thuyết lý tưởng ở trên cùng, chứng tỏ hệ số tăng tốc này là rất tốt.

So sánh giữa CAPE và MPI

Các đường gấp khúc của CAPE gần như gần sát với các đường của MPI thể hiện CAPE có hệ số tăng tốc ổn định và xấp xỉ với MPI. Khi kích thước ma trận nhỏ hơn hệ số tăng tốc thấp hơn do thời gian phân chia công việc và đồng bộ hóa dữ liệu chiếm một tỷ lệ cao hơn so với thời gian tính toán.

Lưu ý lại một lần nữa, MPI là phương pháp cung cấp hiệu năng cao nhất cho việc lập trình song song trên hệ thống sử dụng bộ nhớ phân tán. Do đó hệ số tăng tốc của CAPE như trên đã là rất tốt.

Các kết quả của chương này được công bố ở công trình [CT2] [CT5].

CHƯƠNG III: KỸ THUẬT CHỤP ẢNH TIẾN TRÌNH CHO CAPE ĐA LUỒNG

3.1. Khái niệm Kỹ thuật chụp ảnh tiến trình

Khái niệm và nguyên lý cơ bản của Kỹ thuật chụp ảnh tiến trình đã được trình bày trong phần giới thiệu về CAPE đơn luồng.

3.2. Kỹ thuật chụp ảnh tiến trình gia tăng

Kỹ thuật chụp ảnh tiến trình gia tăng sẽ không lưu toàn bộ dữ liệu trạng thái tiến trình vào mỗi ảnh chụp tiến. Thay vào đó, mỗi ảnh chụp chỉ lưu những giá trị đã được thay đổi so với ảnh chụp ngay trước nó.

3.3. Phát triển Kỹ thuật chụp ảnh tiến trình gia tăng rời rạc cho CAPE đa luồng

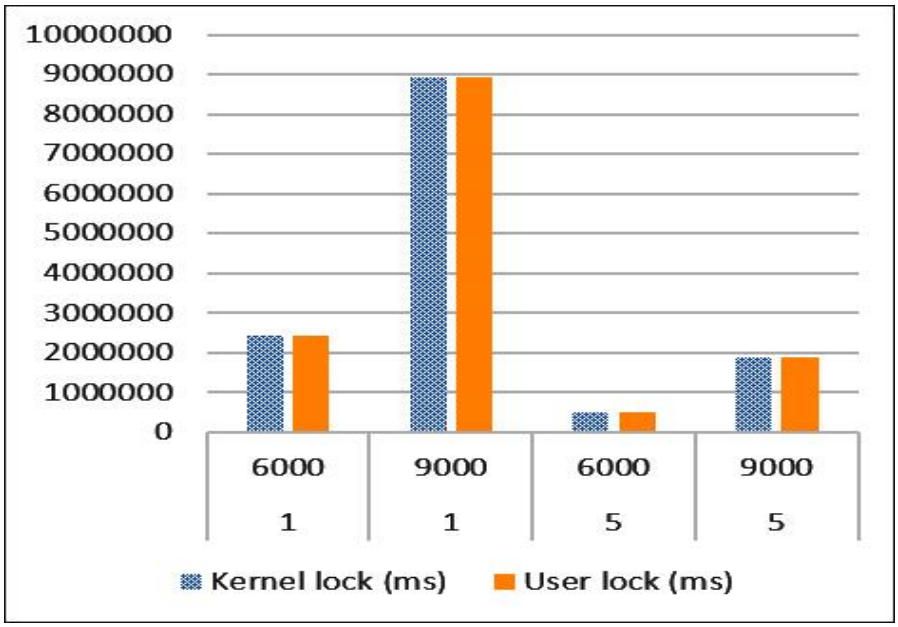
3.3.1. Kỹ thuật chụp ảnh tiến trình gia tăng rời rạc

Kỹ thuật chụp ảnh tiến trình gia tăng rời rạc dựa trên cơ sở kỹ thuật chụp ảnh tiến trình gia tăng và cung cấp thêm khả năng chụp ảnh từng phần riêng biệt tương ứng với từng đoạn mã rời rạc của chương trình khi nó được thực hiện nhằm tối ưu cho CAPE.

3.3.2. Chọn lựa không gian phát triển trình chụp ảnh tiến trình

Trình chụp ảnh tiến trình (Checkpointer) có thể được phát triển ở trong Không gian nhân (Kernel space) hoặc trong Không gian người sử dụng (User space), với những ưu và nhược điểm khác nhau.

Kết quả thực nghiệm so sánh hiệu năng của hai kỹ thuật khóa vùng nhớ ở không gian nhân và không gian người sử dụng được thể hiện ở Hình 3.10.



Hình 3.10. So sánh hiệu năng của hai kỹ thuật khóa vùng nhớ khi áp dụng cho CAPE

Cả hai phương pháp khóa/mở_khóa đều cho hiệu suất về thời gian tương đương nhau. Do đó, chúng tôi đã chọn phương án sử dụng không gian người sử dụng để phát triển trình chụp ảnh tiến trình cho CAPE đa luồng, nhằm đơn giản hóa một phần công việc.

3.6.2. Các thách thức khi chuyển từ trình chụp ảnh tiến trình đơn luồng sang chụp ảnh tiến trình đa luồng

Thách thức thứ nhất: Sự khác biệt địa chỉ giữa các luồng khi đồng bộ ảnh chụp tiến trình.

Thách thức thứ hai: giới hạn kỹ thuật của hệ điều hành trong việc một tiến trình thực hiện giám sát một tiến trình tiến trình khác.

Các thách thức này đã được chúng tôi giải quyết một cách trọn vẹn.

3.6.3. Kết quả phát triển Trình chụp ảnh tiến trình mới cho CAPE đa luồng

Với những nghiên cứu, phân tích, thử nghiệm được nêu ở mực trên, trình chụp ảnh tiến trình mới cho CAPE đa luồng được xây dựng dưới dạng một thư viện cung cấp các hàm có chức năng tương như các hàm của DICKPT monitor và DICKPT driver trong trình chụp ảnh tiến trình của CAPE đơn luồng trước đây.

Do đã gộp cả phần monitor và driver vào trong cùng 1 thư viện DICKPT library, nên hoạt động của các hàm cũng đã đơn giản đi khá nhiều. Mặt khác, vì được phát triển

ở dạng một thư viện liên kết tĩnh nên sau khi trình ứng dụng đã được biên dịch xong thì nó có thể được chạy trên nền tảng CAPE mà không cần có sẵn thư viện này nữa.

Với việc chuyển hướng xây dựng trình ứng dụng sang ở mức không gian người sử dụng, nguyên tắc hoạt động của trình chụp ảnh tiến trình trước đây cũng có sự thay đổi nhỏ, trong đó hoạt động của trình ứng dụng và trình chụp ảnh tiến trình được thực hiện trong một tiến trình duy nhất. Tuy nhiên, sự thay đổi này không làm thay đổi nguyên lý hoạt động của CAPE.

Trình chụp ảnh tiến trình mới đã được sử dụng để thử nghiệm CAPE với mô hình hoạt động mới – CAPE đa luồng – với các kết quả được trình bày ở Chương 2.

Các kết quả của chương này được công bố ở công trình [CT4], [CT5].

CHƯƠNG IV: XỬ LÝ CÁC VẤN ĐỀ CHIA SẺ DỮ LIỆU CỦA CAPE ĐA LUỒNG

4.1. Sự khác nhau của mô hình chia sẻ dữ liệu của OpenMP trên hệ thống bộ nhớ chia sẻ và CAPE trên hệ thống bộ nhớ phân tán

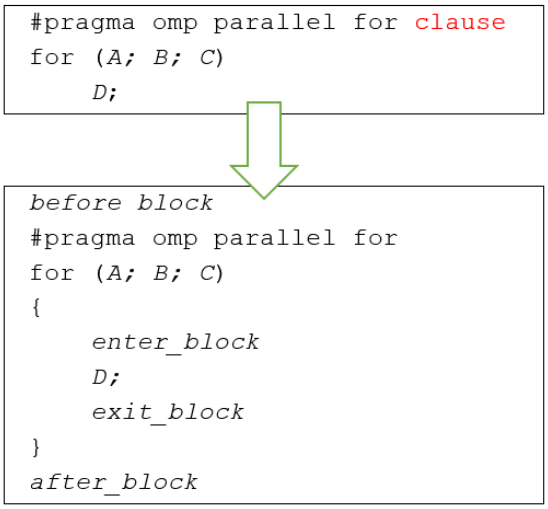
CAPE chuyển mô hình chia sẻ dữ liệu đồng bộ trễ bộ [2] của OpenMP lên lên hệ thống bộ nhớ phân tán.

4.2. Danh sách các chỉ thị và mệnh đề chia sẻ dữ liệu của OpenMP

Danh sách chỉ thị và mệnh đề chia sẻ dữ liệu của OpenMP gồm: `default (none|shared)`, `shared(list)`, `private(list)`, `firstprivate (list)`, `lastprivate(list)`, `copyin(list)`, `copyprivate(list)`, `reduction(list, ops)`.

4.3. Xử lý các mệnh đề dữ liệu chia sẻ của OpenMP trên CAPE

Nguyên tắc xử lý các mệnh đề chia sẻ của OpenMP trên CAPE đã được đề xuất trong [13], bằng cách xây dựng các khuôn dạng chuyển đổi mới hoặc bổ sung vào các khuôn dạng chuyển đổi đã có các câu lệnh để thực hiện các mệnh đề chia sẻ dữ liệu. Nguyên tác này được trình bày ở Hình 4.2 và và Hình 4.3, trong đó có bổ sung các câu lệnh để xử lý các mệnh đề dữ liệu chia sẻ của CAPE.



Hình 4.2. Khuôn dạng tổng quát việc chuyển đổi cấu trúc `parallel for` với các mệnh đề dữ liệu chia sẻ

```
0    init ( x ) ;
1    if ( master ( ) ) {
...
8          stop ( ) ;
8a         init ( update_list ) ;
9          wait_for ( after ) ;
9a         merge ( update_list, after ) ;
9b         receive_reduction ( x ) ;
9c         merge ( x, after ) ;
...
15   } else {
...
22         send ( after_i , master ) ;
22a        send ( x , master ) ;
```

Hình 4.3. Khuôn mẫu chuyển đổi của CAPE cho mệnh đề reduction

4.5. Luận giải tính đúng của cơ chế chuyển đổi các mệnh đề dữ liệu chia sẻ của OpenMP trên CAPE đa luồng

Đối với phiên bản CAPE đa luồng được đề cập trong luận án này có thêm điểm mới là các nút sẽ chạy đa luồng tuy nhiên như đã đề cập trong 2.3. Luận án đã luận giải được tính đúng của của cơ chê chuyển đổi các mệnh đề dữ liệu chia sẽ của OpenMP trên CAPE đa luồng.

Về cài đặt chương trình CAPE thực hiện các mệnh đề chia sẻ dữ liệu của OpenMP trên hệ thống bộ nhớ phân tán, nhóm nghiên cứu đã cài đặt thành công các mệnh đề `threadprivate(x)`, `private(x)`, `firstprivate(x)`, `lastprivate(x)`,

`copyin(x), copyprivate(x), reduction` với kết quả chạy chương trình giống với chương trình gốc OpenMP. Toàn bộ chương trình được công bố dưới dạng mã nguồn mở GitHub truy cập trực tuyến như ở PHỤ LỤC 1.

Các kết quả của chương này được công bố ở công trình [CT3]..

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN CỦA LUẬN ÁN

KẾT LUẬN

Luận án đã thành công trong việc mở rộng mô hình hoạt động của CAPE trước đây để khai thác tốt hơn khả năng của các bộ vi xử lý đa lõi trên các nút tính toán. Mô hình mới đã bổ sung thêm một mức song song khi thực hiện các đoạn mã tính toán trên các nút này, làm cho việc thực hiện một câu lệnh song song `S` dạng OpenMP được tiến hành song song theo 2 mức: 1) Các nút tính toán thực hiện song song từng phần S_i của `S` trên các nút tính toán (i đại diện cho số nút tính toán); và 2) Từng phần S_i tại mỗi nút được thực hiện một cách song song theo dạng 1 tiến trình đa luồng OpenMP. Các kết quả phân tích lý thuyết cũng như những kết quả thực nghiệm đã chứng minh mô hình mới đã giúp hiệu năng hệ thống khi áp dụng CAPE tăng tuyến tính với số lõi của các bộ xử lý trên các nút tính toán.

Việc cài đặt mô hình thực hiện mới đòi hỏi phải xây dựng lại công cụ căn bản và quan trọng nhất của CAPE là công cụ chụp ảnh tiến trình. Sự thay đổi so với công cụ trước đây là rất lớn, mang tính cốt lõi, để có thể xử lý được việc chụp được ảnh các tiến trình đa luồng tại các nút tính toán. Luận án đã phát triển thành công kỹ thuật chụp ảnh tiến trình phù hợp với CAPE đa luồng, chạy ở mức không gian người sử dụng, thay vì chạy ở không gian nhân của hệ điều hành như trước đây giúp có ưu điểm là tính ổn định cao với các phiên bản khác nhau của OS do chỉ gọi lại các hàm thư viện phổ thông của OS cung cấp.

Kết quả của luận án đã đạt được bao gồm:

1. Phân tích và thực nghiệm các hướng mở rộng mô hình CAPE trên hệ thống máy tính CPU đa lõi và chọn giải pháp khả thi để thực hiện [CT1] [CT2].

2. Tổng hợp và phân loại các kỹ thuật chụp ảnh tiến trình- kỹ thuật nền tảng lõi áp dụng cho CAPE- từ đó đề xuất áp dụng kỹ thuật chụp ảnh tiến trình phù hợp cho CAPE đa luồng [CT4]

3. Đề xuất mô hình hoạt động mới CAPE đa luồng, xây dựng và thực nghiệm thành công hệ thống phần mềm CAPE đa luồng áp dụng trên hệ thống tính toán đa lõi với hiệu năng có thể so sánh được với MPI - công cụ có khả năng cung cấp hiệu năng cao nhất trên các hệ thống này [CT5].

4. Xử lý các vấn đề chia sẻ dữ liệu của CAPE đa luồng [CT3].

HƯỚNG PHÁT TRIỂN LUẬN ÁN

Từ những kết quả đạt được trong luận án một số vấn đề cần được quan tâm nghiên cứu trong thời gian tới:

1. Mô hình hoạt động của CAPE có được phát triển tiếp tục theo hướng khai thác khả năng song song của các card đồ họa.

2. Nếu được một đơn vị/nhóm phát triển phần mềm hệ thống quan tâm đầu tư và tiếp tục phát triển một cách nghiêm túc, CAPE hoàn toàn có khả năng trở thành một công cụ tốt để đưa OpenMP lên các hệ thống bộ nhớ phân tán, tương đương như MPI, mang lại một giải pháp lập trình song song dễ học, dễ lập trình và có hiệu năng cao trên các hệ thống này. Chúng tôi cũng đã đưa mã nguồn của phần mềm CAPE lên Github – kho mã nguồn mở của cộng đồng – với định hướng cung cấp mã nguồn CAPE để cộng đồng cùng tiếp tục cùng nghiên cứu, phát triển.

DANH MỤC CÁC CÔNG TRÌNH LIÊN QUAN ĐẾN LUẬN ÁN

[CT1]. **Đỗ Xuân Huyền**, Hà Viết Hải (2018), "Các giải pháp mở rộng mô hình hoạt động của CAPE cho mạng máy tính sử dụng bộ vi xử lý đa lõi". *Tạp chí Khoa học và Công nghệ (Trường Đại học Khoa học - Đại học Huế)*, ISSN 2354-0842, Tập 12, Số 1, 2018, Tr. 51–61. http://joshusc.hueuni.edu.vn/jos_articles.php?article_id=385.

[CT2]. Hà Viết Hải, **Đỗ Xuân Huyền** (2018), "Mở rộng mô hình hoạt động của CAPE bằng sử dụng máy ảo". *Tạp chí Tạp chí Khoa học Đại học Huế: Kỹ thuật và Công nghệ*; ISSN 1859–1388, Tập 127, Số 2A, 2018, Tr. 159–168. http://jos.hueuni.edu.vn/index.php/hujos-tt/article/view/4795.

[CT3]. **Đỗ Xuân Huyền**, Hà Viết Hải, Trần Văn Long (2019), "Xử lý các mệnh đề về dữ liệu chia sẻ của OpenMP trên các hệ thống sử dụng bộ nhớ phân tán". *Kỷ yếu Hội nghị khoa học quốc gia lần thứ XII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR)*. Huế 2019, Tr. 577–583. https://doi.org/10.15625/vap.2019.00074.

[CT4]. **X. H. Do**, V. H. Ha, V. L. Tran and É. Renault, "The technique of locking memory on Linux operating system - Application in checkpointing," 2019 6th NAFOSTED Conference on Information and Computer Science (NICS), 2019, pp. 178-183. https://doi.org/10.1109/NICS48868.2019.9023816.

[CT5]. **Xuan Huyen Do**, Viet Hai Ha, Van Long Tran, Eric Renault (2021), "New execution model of CAPE using multiple threads on multi-core clusters", *ETRI Journal (SCIE—Q2)*, May, 2021. https://doi.org/10.4218/etrij.2020-0201.

HUE UNIVERSITY

UNIVERSITY OF SCIENCES

DO XUAN HUYEN

IMPROVED CAPE MODEL ON MULTI-CORE COMPUTING SYSTEMS

MAJOR: COMPUTER SCIENCE

CODE: 9480101

SUMMARY OF PHD. THESIS

HUE –2023

The thesis has been completed at:
University of Sciences, Hue University

Supervisor:

1. Dr. Ha Viet Hai, University of Education, Hue University
2. Prof. Eric Renault, University Gustave Eiffel, Paris, France.

Reviewer 1:

Reviewer 2:

Reviewer 3:

The thesis will be presented at the Committee of Hue University, to be held by Hue University at.

The thesis can be found at the following libraries:

- National Library of Vietnam
- Library Information Center University of Science, Hue University

PREFACE

OpenMP is an API whose goal is to add parallel programming capabilities to programs written in C, C++, and Fortran languages, running on shared memory architecture (computers with multiple CPUs and/or multi-core CPU). OpenMP is simple, easy to learn, easy to use, and provides high performance, so it is quickly becoming the parallel programming standard for shared memory architectures. However, OpenMP does not run on distributed memory systems (like clusters, grids). This led to an idea and impetus for many researchers to migrate OpenMP to distributed memory architectures. Implementing the above idea, over the years, many research groups have tried to implement OpenMP on distributed memory computer systems by building compilers to automatically translate OpenMP programs into programs that can execute on these systems. Simultaneously with building compiled programs, some approaches also require building an additional platform for the systems to be able to run compiled programs.

However, with the exception of CAPE [12]-[19] there is no successful work on both sides: fully compatible with OpenMP and have high performance. Some prominent research works that can be mentioned are SSI [6]; Cluster OpenMP [11]; SCASH [7]; using the HLRC model [24]; compiling to MPI [8][9]; using Global Array[10]; libMPNode [30] improved SSI; OMPC[34] uses its own compiler. Currently, OpenMP in general and OpenMP development on distributed memory systems in particular are the main topics of the IWOMP annual conference series, with the 19th conference will be held at the University of Bristol, UK in September 2023.

CAPE (Checkpointing Aided Parallel Execution) is a checkpointing-based approach to implementing OpenMP on distributed memory computing systems. Prof. Eric Renaut invented CAPE during the period he worked at the Institute of Telecommunication SudParis. The second version of CAPE was developed by Dr. Ha Viet Hai and Dr. Tran Van Long. The performance of CAPE in this version has been significantly improved, making it close to MPI's performance- the method can provide the highest performance for parallel programming on distributed systems. However, in both versions, the computers in the system are exploited from the point of view of using

single-core processors, thus not fully exploiting the capabilities of multi-core processors, which are commonplace nowadays. Therefore, there is a waste of resources when using CAPE on computer systems with multi-core CPUs. To overcome this limitation, the execution model of CAPE should be reorganized in the direction of allowing the program to run in parallel on each slave node (second-level parallelism) to better exploit system resources. This is the motivation for this thesis: propose a new execution CAPE model to overcome the limitations of not exploiting computer system resources using multi-core CPUs that have become popular today. To achieve this goal, the following issues need to be developed: (1) Develop a new CAPE execution model to efficiently parallelize the computational code on each slave node; (2) Develop a checkpointer as well as solve CAPE's shared-data problems to suitable for the two-level parallelization execution models mentioned above.

From the foregoing, the topic **"Improving CAPE model for multi-core computing systems"** becomes topical and urgent to meet the need to provide an implementation that is fully OpenMP compatible and has high performance on distributed memory architectures.

The research objective of the thesis is to develop a new execution model of CAPE on a multi-core computing system to improve the performance of the system.

Specifically:

- Objective 1: Propose a new execution model of CAPE on multi-core computing systems.
- Objective 2: Develop a new checkpointer that is suitable for the new execution model of CAPE on multi-core computer systems.
- Objective 3: Propose these solutions for shared data which is suitable for the new execution model of CAPE on multi-core computer systems;
- Objective 4: Develop a software system corresponding to the proposed new CAPE execution model and evaluate its performance while comparing with the previous CAPE execution model and with MPI (the technique which can provide the best current performance on distributed systems).

CHAPTER I: RESEARCH OVERVIEW

1.1. High performance computing

High performance computing (HPC) generally refers to the processing of complex calculations at high speed on many parallel servers.

1.2. Parallel computing

Parallel computing or parallel processing is a type of computation in which many calculations or processes are carried out simultaneously.

Speedup factor: The speedup factor of a parallel program is the ratio between the execution time in the case of using a sequential program and the time to execute the same job in the parallel program.

According to Amdahl's law [36], the prediction of the maximum speed increase factor of the parallel processing program is calculated by the following formula:

$$S_{\text{latency}} = \frac{1}{(1-p) + \frac{p}{s}}$$

with

$S_{latency}$: the maximum possible improvement of the overall system.

p: the ratio of the parallelizable code over the total execution time.

s: is the number of processors or the part of the task that speeds up due to the improved system resources.

1.3. Multi-CPU, Multi-core CPU, and Multithreading

Multi-CPU machine systems can be divided into two types according to different ways of organizing memory, one using a shared memory system and the other using a distributed memory system.

Multithreading is the CPU's ability to handle multiple threads at the same time. When there are many different tasks, the CPU can work on them in parallel for single-core CPUs or concurrently for multicore CPUs. Core is where tasks are done and each core only runs 1 task at a time. It is for this reason that the more cores there are, the more

tasks the CPU can perform at the same time. Intel's Hyper-Threading Technology improves CPU performance by up to 30% [75].

Most popular operating system versions such as Windows and Linux distributions support multi-core as well as multi-threaded CPUs.

The goal and scope of this thesis's CAPE improvement research are to apply to computer systems using multi-core CPUs. It should be noted that CAPE is similar to MPI which is a high-level abstraction language consisting of an application layer runtime library that does not interfere directly with the hardware of multi-core CPUs but calls operating system services to run programs.

1.4. OpenMP

OpenMP is a programming interface (API) that provides a high level of abstraction for writing parallel programs for HPC with the advantage of being easy to learn and use. To write parallel programs with OpenMP, a programmer can start by writing sequence programs in the original language (C/C++ or Fortran), then gradually add OpenMP directives to specify the parts of the code that needs to be parallelized. Data sharing as well as data synchronization is done implicitly or explicitly through simple directives. However, OpenMP is still only fully implemented for shared memory architectures due to the complexity of implementing all OpenMP requirements on top of other memory architectures.

1.5. Notable research works on implementing OpenMP on distributed memory systems

The notable research works on implementing OpenMP on distributed memory systems are the following: SSI method which builds a virtual common memory for all threads; Method mapping a portion of a stream's memory space; Method using HLRC delay synchronization model; Methods associated with MPI; Method based on Global Array (GA); CAPE - Method based on checkpointing technique.

1.6. Synthesis comparison and evaluation of methods of implementing OpenMP on distributed memory architectures

The thesis has synthesized, and evaluated the advantages and disadvantages of these methods and selected the direction of continuing to improve CAPE technique.

1.7. CAPE-single-threaded

In previous CAPE versions, portions of the tasks performed at the slave nodes were organized in the form of single-threaded execution. Therefore, in this thesis, we call this version of CAPE as CAPE-single-threaded.

The execution model of CAPE-single-threaded is illustrated in Figure 1.6.

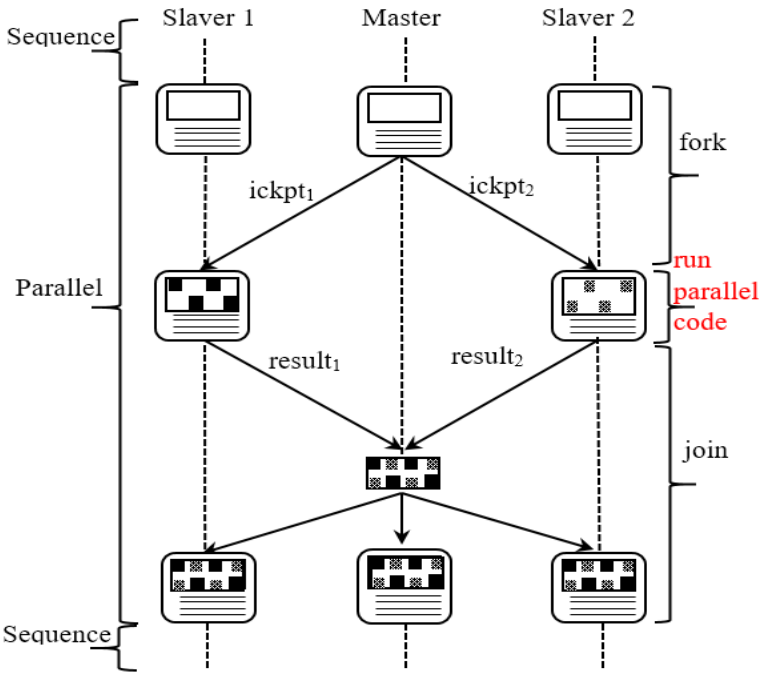


Figure 1.6. The execution model of CAPE-single-threaded.

The process of compiling an OpenMP program to a CAPE program is presented in Figure 1.7.

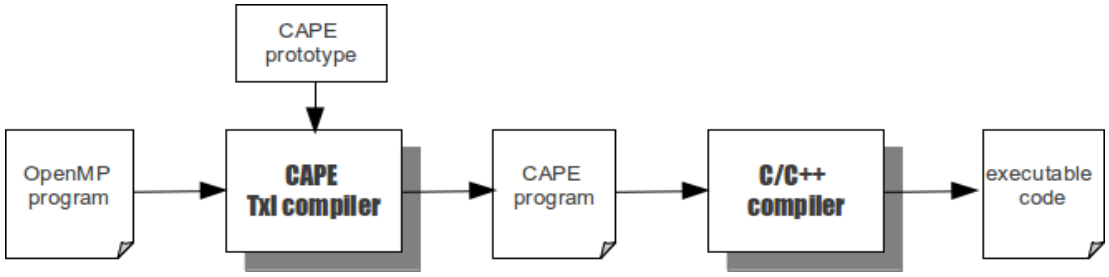


Figure 1.7. The process of compiling an OpenMP program into a CAPE program

In the above process, the CAPE library functions (C code) are automatically added to the CAPE program.

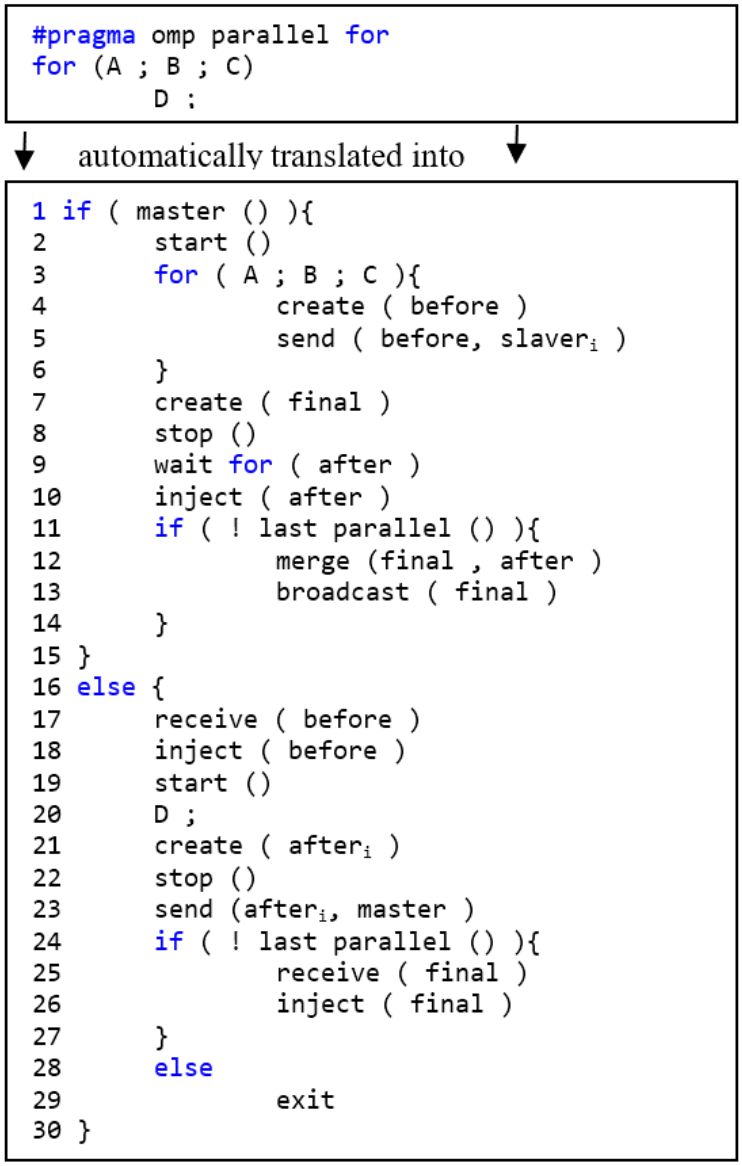


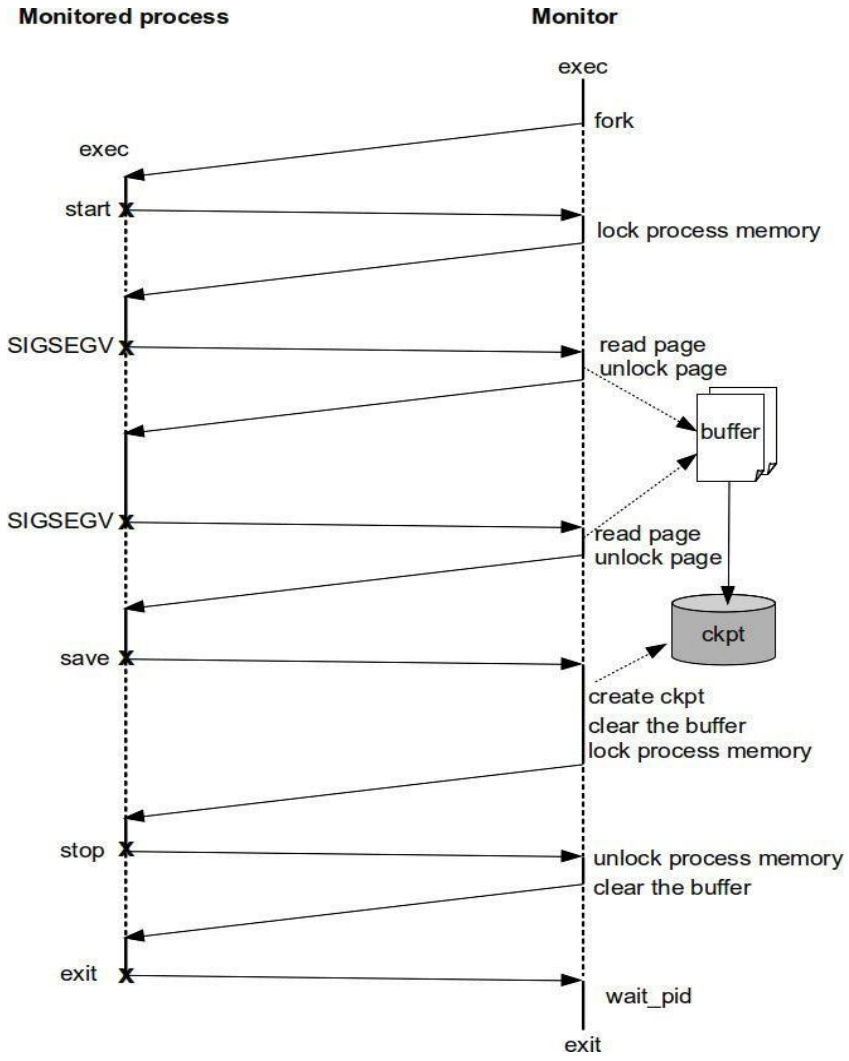
Figure 1.8. Prototype to translate `pragma omp parallel for` in CAPE's single-threaded.

Checkpointing technique applied to CAPE-single-threaded

Checkpointing is the technique that saves the image of a process at a point during its running time and allows it to be resumed from the saving's time if necessary [35]. Using checkpointing, processes can resume their execution from a checkpoint state when a failure occurs. So, there is no need to take time to initialize and execute it from the beginning. These techniques have been introduced for more than two decades. Nowadays, they are used widely for fault tolerance, application trace/debugging, roll-back/animated playback, and process migration.

The checkpointing technique can be divided into 2 types, based on the ways of saving checkpoints. While complete checkpointing saves all process memory space, incremental checkpointing only saves the modified data from the previous checkpoint. The second technique reduces the checkpoint's overhead and checkpoint's size.

For optimal use of CAPE, DICKPT (Discontinuos Incremental Checkpointing) was developed in the years 2009-2011 [13]. This technique is based on incremental checkpointing and provides the ability to capture discontinuous sections of the program's running time.



Hình 1.12. Principle of DICKPT

Data structure of checkpoints

The checkpoints generated by DICKPT contain two sets of data: a) register values – ie. all register values at the checkpoint's time, and b) process's modified memory

space – ie. all modified data in memory space of process as compared to the previous checkpoint.

For modified data, the memory granularity is world-level (4-byte word in this case), so in most cases, a lot of space is needed to store addresses and values of modified memory regions. In [12], authors have proposed four structures to optimize the checkpoint size: *Single data (SD)* with the structure of the data is {(addr, value)}; *Several successive data (SSD)* with the structure of type {(addr, size, values)}; *Many data (MD)* with the structure of type {(addr, map, values)}.; *Entire page (EP)* with the structure of the data is {(addr, values)} and the size is identified automatically by the page size.

The results of this chapter are published in [CT1].

CHAPTER II: CAPE EXECUTION MODEL ON MULTI-CORE COMPUTING SYSTEMS

2.1. General principles

The execution model of CAPE-single-threaded is illustrated in Figure 2.1.

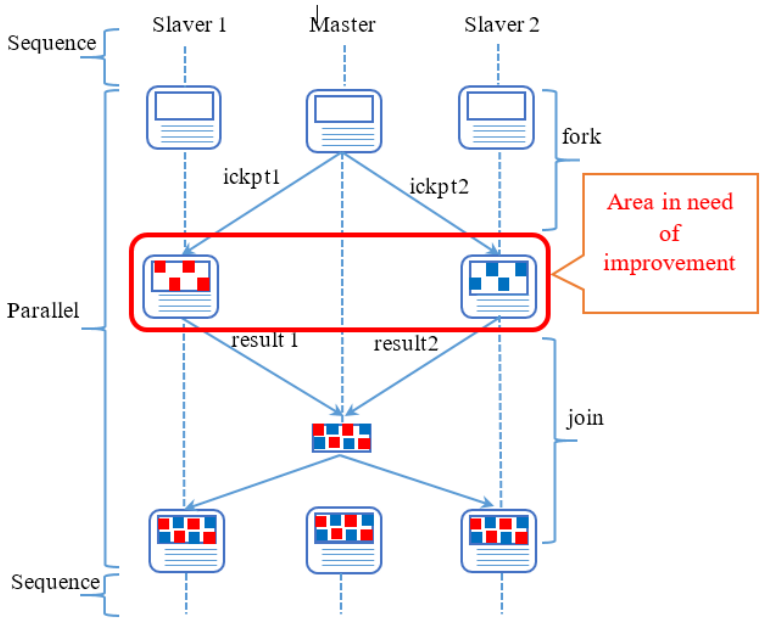


Figure 2.1: CAPE execution model and the regions needed to improve to better exploit the capabilities of multi-core processors

In order to increase the performance of the program, obviously during the computation time at the slave nodes, it is necessary to parallelize the parts of the program running on each slave node, by making them execute in form of multiprocessing or multithreading (2 parallel levels). In Figure 2.1, the area highlighted in red color is the area that needs improvements. According to the principle of CAPE, there are 3 methods that can do this: 1) Use multi-process on each slave node by executing multiple times of application programs on each node (Method 1); 2) Use parallelism on each slave node by parallelizing the execution of application programs on virtual machines, creating multiple virtual machines on each physical machine (Method 2); and 3) Parallelize the computation codes on the slave node in a multi-threaded model, i.e. 2 levels of parallelism: level 1 involves dividing and running tasks on multiple machines concurrently, while level 2 involves to using multithreading on each slave node (Method 3).

Method 1) has been tested and published in [64] with unfeasible results. This thesis has carried out detailed research and experiments on both methods 2) and 3), which are presented in the next sections.

2.2. CAPE execution model with two-level parallel based on the method of using multiple virtual machines

The thesis has experimented with this research direction, but the results in terms of performance are not as expected, the execution time in the case of using virtual machines is always longer than the time of CAPE-single-threaded.

2.3. CAPE execution model with two-level parallel based on the method of using multithreading on slave nodes[1]

2.3.1. Idea

The basic principle of this approach is to implement the parallel architectures of OpenMP through two levels as illustrated in Figure 2.7.

[1] Multithreading is the CPU's ability to handle multiple threads at the same time. When there are many different tasks, the CPU can work on them in parallel for single-core CPUs or concurrently for multicore CPUs. Core is where tasks are done and each core only runs 1 task at a time.

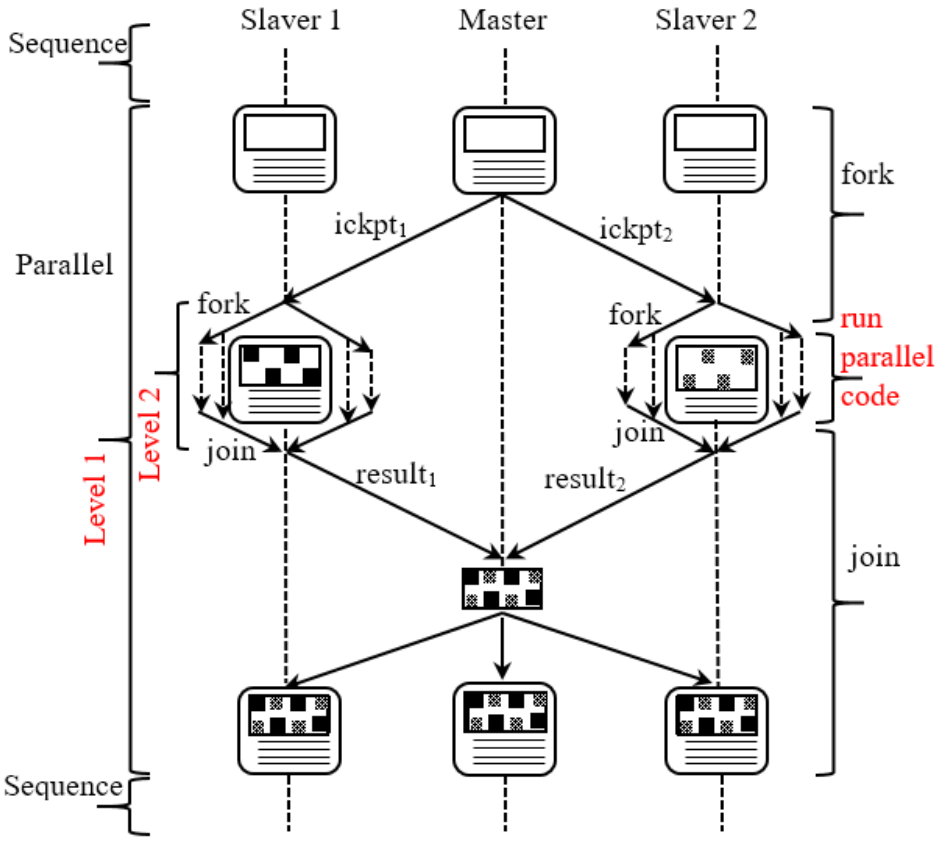


Figure 2.7. CAPE operation model with 2-level parallel based on the method of using multithreading on slave nodes

We call this exection model - “CAPE operation model with 2-level parallel based on the method of using multithreading on slave nodes” - as "**CAPE-multithreaded**".

2.3.1. Prototype to translate `pragma omp parallel for` in CAPE-multithreaded

As shown in Figure 2.8, the parallel OpenMP code, in CAPE-multithreaded execution model, is compiled into a set of C/C++ instructions that contain both OpenMP and CAPE directives. This provides the ability to run this code in form of 2 levels of parallelism. Level 1 involves dividing and running tasks on multiple computers concurrently, while level 2 involves using multithreading on each slave node.

```
#pragma omp parallel for
for (A ; B ; C)
        D ;
```

automatically translated into

```
1 if ( master () ){
2       start ()
3       for ( A ; B ; C ){
4               create ( before )
5               send ( before, slaver_i )
6       }
7       create ( final )
8       stop ()
9       wait for ( after )
10      inject ( after )
11      if ( ! last parallel () ){
12              merge (final , after )
13              broadcast ( final )
14      }
15 }
16 else {
17      receive ( before )
18      inject ( before )
19      start ()
20      #pragma omp parallel for
21      for (A_i ; B_i ; C_i)
22              D ;
23      create ( after_i )
24      stop ()
25      send (after_i, master )
26      if ( ! last parallel () ){
27              receive ( final )
28              inject ( final )
29      }
30      else
31              exit
32 }
```

Figure 2.8. Prototype to translate `pragma omp parallel for` in CAPE-multithreaded

2.4. CAPE-multithreaded performance analysis

2.4.1. Theoretical speedup factor according to Amdahl's law

For CAPE-multithreaded, this parallelization is in the form of a multithreaded process. Therefore, the optimal number of parallelisms will be equal to the number of cores of the CPU, and then the maximum acceleration of this section will approximate the number of cores according to Amdahl's law, ignoring the costs associated with data synchronization, organizing, and executing multi-threaded processes.

2.4.2. Experimental results and evaluation

2.4.2.1. Experimental system and experimental problem

Experiments were conducted by running a program to multiply two square matrices with sizes 9600x9600 and 6400x6400 respectively, with the number of threads at the slave nodes varying from 1 to 2 on system 1 (cluster of 16 slave nodes); and 1 to 4 threads on system 2 (cluster of 8 slave nodes).

2.4.2.2. Experimental results and evaluation analysis

2.4.2.2.1. Results and evaluations over time

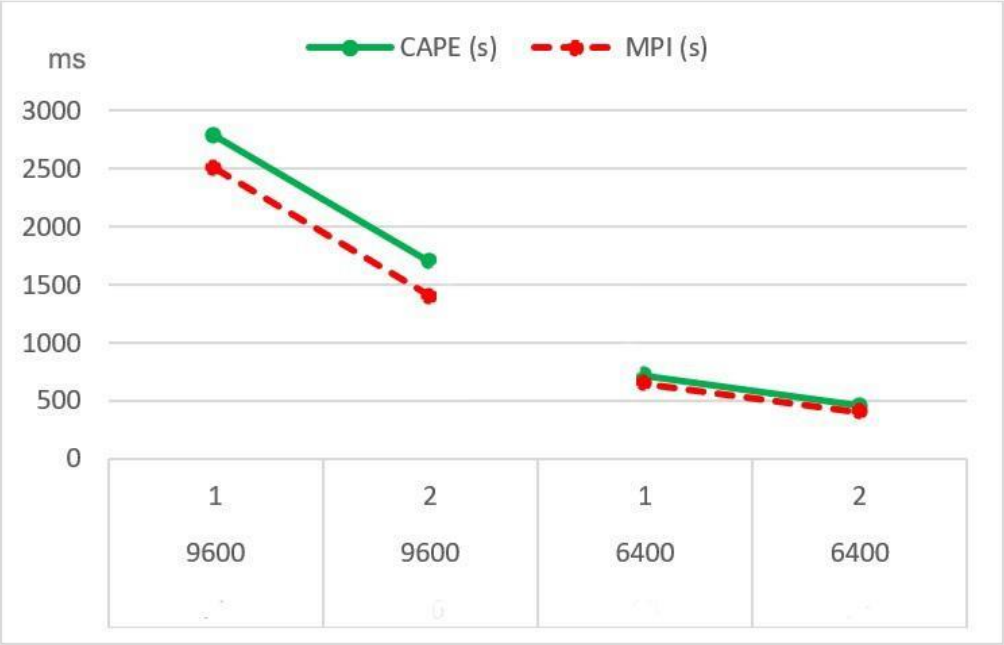


Figure 2.9. Execution time on a cluster of 16 slave nodes varies with the number of cores used

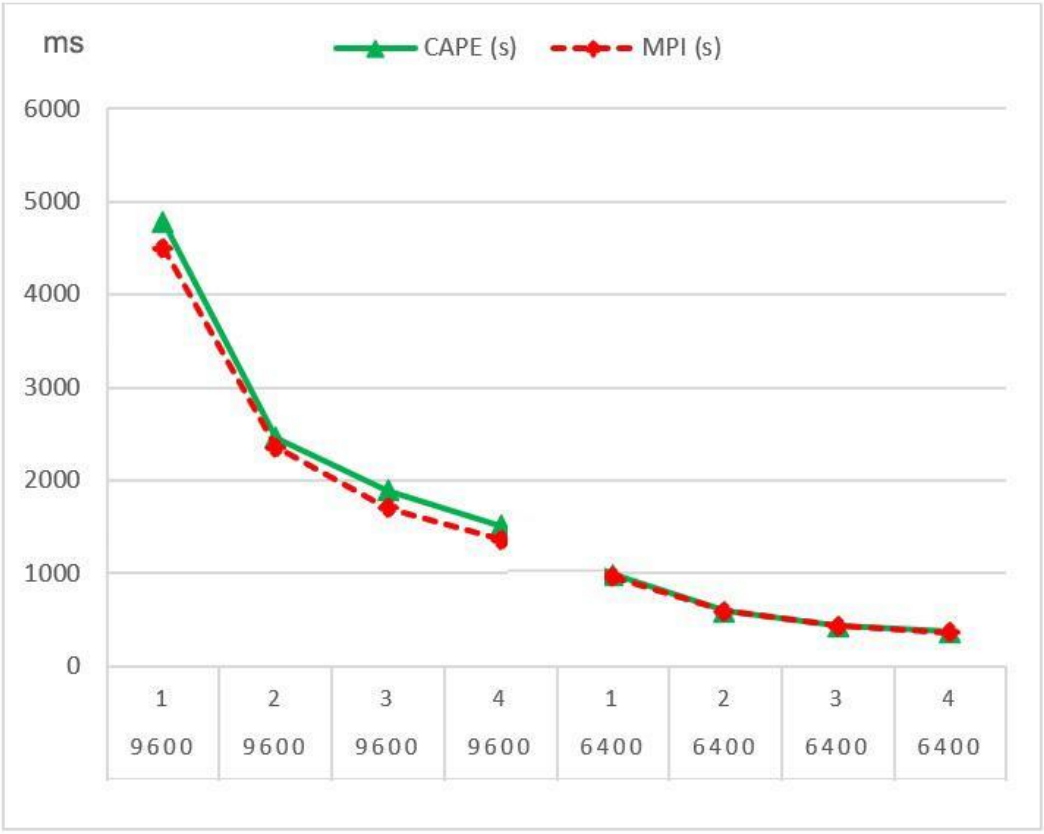


Figure 2.10. Execution time on a cluster of 8 slave nodes varies with the number of cores used

Comparison between CAPE-multithreaded and CAPE-single-threaded

For both experimental systems and the two problem sizes on them (9600x9600 and 6400x6400), the execution time program of CAPE-multithreaded is always less than the one of CAPE-single-threaded. This clearly proves that the multi-threading execution model has brought better performance than the previous single-threaded execution model.

With an 8-slave cluster system, the graph in Figure 2.10 shows that the acceleration gradient from 1-core to 2-core conversion is always greater than the slope when increasing from 2 cores to 3 cores, as well as from 3 cores to 4 cores. This is likely due to the Intel(R) Core(TM) i3 CPU architecture, which, although considered to have 4 cores, actually has only 2 physical cores and each core contains 2 logical cores with hyper-threading technology.

Comparison between CAPE and MPI

In all cases, the execution time of the program of MPI is always faster than CAPE-multithreaded, but at a relatively stable speed, as shown by their graphs being almost parallel to each other. This also proves that CAPE-multithreaded works stably in case of increasing or decreasing the number of cores used. Furthermore, the difference in execution time between CAPE and MPI is only approximately 8%. This difference is not high since CAPE always takes the time to perform monitoring and process checkpoints. With this difference, it can be said that CAPE-multithreaded has approached the best parallel programming method on a distributed memory system, which is MPI. Combined with OpenMP's outstanding features that are easy to learn, easy to use, and save effort and programming time, it can be said that CAPE is capable of providing a superior parallel programming method on distributed memory systems.

2.4.2.2.2. Results and evaluation over speedup

Calculation results are shown in Figures 2.11 and 2.12.

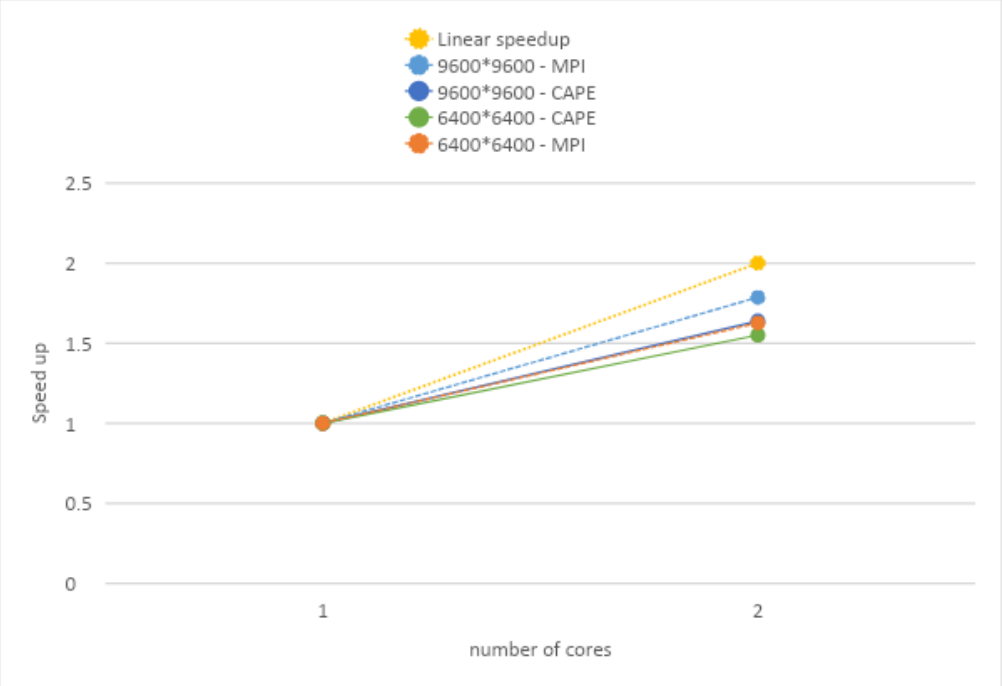


Figure 2.11. Speedup according to the number of CPU cores used on the cluster of 16 slave nodes

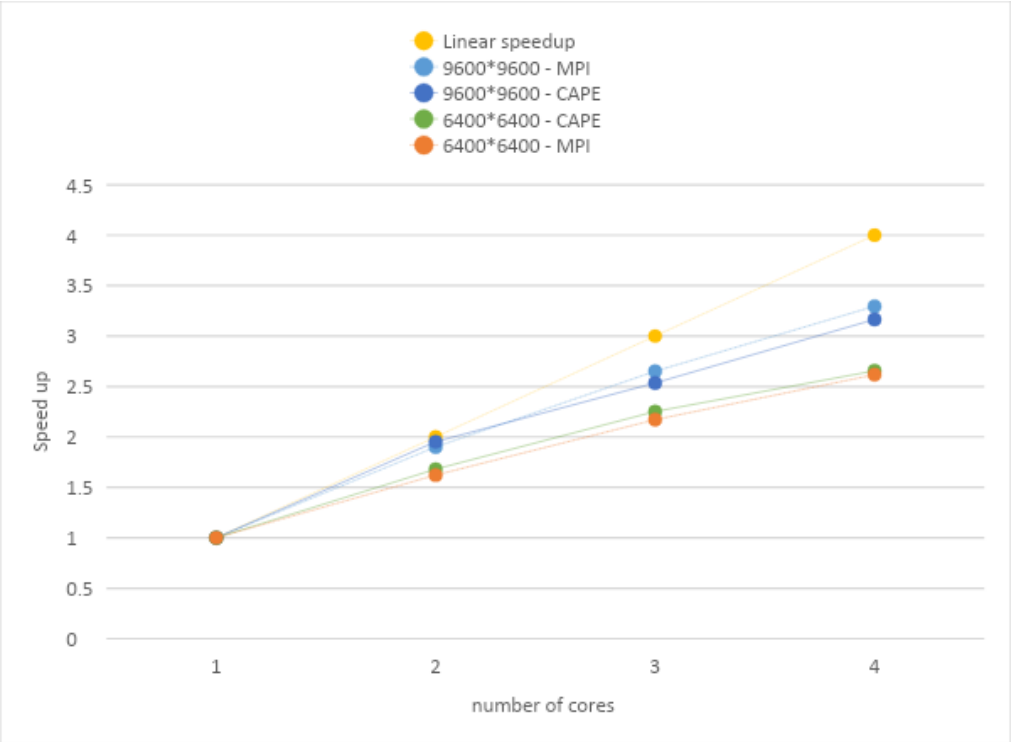


Figure 2.12. Speedup according to the number of cores used on the cluster of 8 slave-nodes

Comparison between CAPE-multithreaded and CAPE-single-threaded

In both figures above, the graphs of CAPE-single-threaded has only 1 single point corresponding to the case of 1-core used. Compared to the cases of CAPE-multithreaded using 2 to 4 cores, we see that they almost create a linear speedup curve. To be precise, CAPE and MPI speedup is 1.90-1.95 times higher in the case of using 2 cores when compared to the case of using 1 core. At the same time, when using 4 cores (actually 2 physical cores and 2 hyperthreading), the speedup is still high, up to 3.16 compared to the case of using 1 core (CAPE-single-threaded). Furthermore, this speedup line is almost asymptotic to the ideal theoretical acceleration line at the top, which proves that this speedup is very good.

Comparison between CAPE and MPI

The curves of CAPE is almost close to those of MPI, showing that CAPE has a stable acceleration coefficient and is close to MPI. When the matrix size is smaller, the speedup is lower because the time of division and data synchronization takes a higher proportion of the time of computation.

Note again, MPI is the method that provides the highest performance for parallel programming on distributed memory systems. Therefore, the speedup of CAPE as above is already very good.

The results of this chapter are published in [CT2] [CT5]

CHAPTER III: CHECKPOINTING TECHNIQUES FOR CAPE-MULTITHREADED

3.1. The concept of checkpointing

The concept and principles of checkpointing techniques were covered in the introduction to CAPE-single-threaded.

3.2. Incremental checkpointing techniques

It was covered in the introduction to CAPE-single-threaded.

3.3. Discontinuous Incremental Checkpointing Techniques for CAPE-multithreaded

3.3.1. Discontinuous Incremental Checkpointing Techniques

It was covered in the introduction to CAPE-single-threaded.

3.3.2. Selecting the space for the development of the checkpointing

Experimental results comparing the performance of two memory locking techniques in kernel space and user space are shown in Figure 3.10.

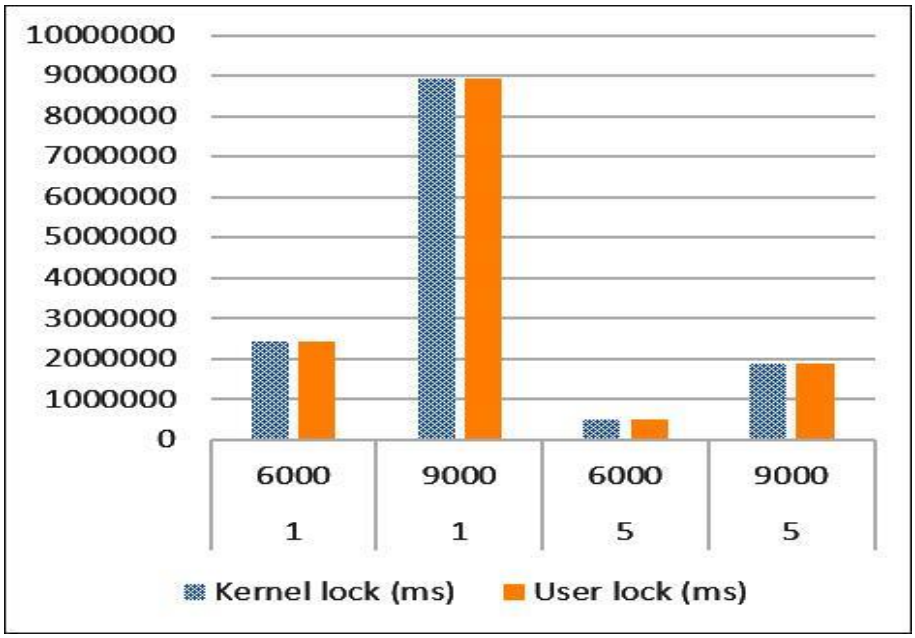


Figure 3.10. Performance comparison of the two memory locking techniques when applied to CAPE

Both locking/unlocking methods give similar time performance. Therefore, we chose to use user space to develop a checkpointing for CAPE-multithreading.

3.6.2. Challenges when changing from checkpointing applied for CAPE-single-threaded to CAPE-multi-threading

The most important challenges are

The first challenge: Address differences between address memory of threads when syncing checkpoints

The second challenge: the technical limitation of the operating system in which one process can monitor another multithreading process.

We have totally solved these challenges to develop the new chekpointer.

3.6.3. New checkpointing development results for CAPE-multithreading

The new checkpointer for CAPE-multithreaded is built in the form of a library that provides functions similar to the functions of the DICKPT monitor and DICKPT of the previous CAPE-single-threaded.

Due to the inclusion of both the monitor and the driver in the same DICKPT library, the operation of the checkpointing functions has also been simplified. Because it is developed as a statically linked library, after the application has been compiled, it

can be run under the execution model of CAPE without the CAPE library pre-installed on the slave machines.

With the change from building a checkpointing application to the user-space level, the working principle of the previous checkpointing has also changed slightly, in which the operation of the application and checkpointing application is performed in a single process. However, this change does not change the operating principle of CAPE.

The new checkpointing techniques were used to test CAPE-multithreading – with the results presented in Chapter 2.

The results of this chapter are published in [CT4], [CT5].

CHAPTER IV: HANDLING THE DATA-SHARING ISSUES OF MULTI-THREADED CAPE

4.1. The difference between OpenMP's data sharing model on shared memory systems and CAPE on distributed memory systems

CAPE changes from OpenMP's relaxed-consistency, shared memory model [2] to the Updated Home-based Lazy Release Consistency memory model. The principles of this model are already presented in the thesis of Dr. Ha Viet Hai. However, in this model, it is necessary to continue to process the data-sharing directives.

4.2. List of OpenMP's data sharing directives

OpenMP provides the following data sharing directives: default (none|shared), shared(list), private(list), firstprivate (list), lastprivate(list), copying (list), copyprivate(list), reduction(list, ops).

4.3. Handling OpenMP shared data clauses CAPE

The principle of handling OpenMP's shared clauses on CAPE has been proposed in [13], by building new prototypes of translation or adding to existing prototypes the instructions to perform the tasks of data sharing clauses. Figure 4.2 presents the new prototypes for the structure `omp parallel for` with the instructions to handle a data sharing clause in general case, and figure 4.3 presents the handling for the case of `reduction` clause.

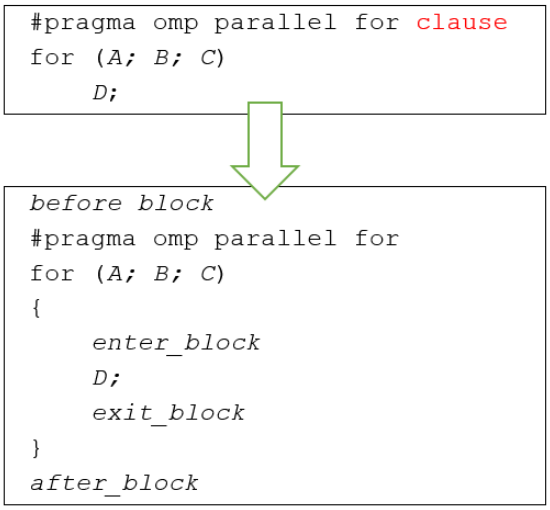


Figure 4.2. General prototype to retranslate the `parallel for` directive with shared data clause

```
0    init ( x ) ;
1    if ( master ( ) ) {
...
8          stop ( ) ;
8a         init ( update_list ) ;
9          wait_for ( after ) ;
9a         merge ( update_list, after ) ;
9b         receive_reduction ( x ) ;
9c         merge ( x, after ) ;
...
15   } else {
...
22         send ( after_i , master ) ;
22a        send ( x , master ) ;
```

Figure 4.3. Prototype to handle `reduction` clause

4.5. Explanation of OpenMP's shared data clause translate mechanism to CAPE-multithreaded

For the CAPE-multithreaded version mentioned in this thesis, there is a new feature that the slave nodes will run multithreaded process. However, as mentioned in 2.4, the multithreading parts on each node reuse OpenMP code and do not interfere with the memory area of this code, so according to the proof logic mentioned in this section, the cases that need to be proven above are enough.

Regarding the installation of the CAPE program that implements OpenMP's data-sharing clauses on the distributed memory system, the research team has successfully installed the following statements: `threadprivate(x)`, `private(x)`, `firstprivate(x)`, `lastprivate(x)`, `copyin(x)`, `copyprivate(x)`, `reduction` with the same results as the original OpenMP program. The entire program is published as open source on GitHub and available as presented in APPENDIX 1.

The results of this chapter are published in [CT3].

CONCLUSION

Conclusion

The thesis has succeeded in extending the previous CAPE execution model to better exploit the capabilities of multi-core computing systems. The new model has added a new level of parallelism when executing computational code on slave nodes, making the execution of an OpenMP parallel structure *S* in two levels of parallelism: 1) Many slave nodes perform different parts S_i of *S* in parallel (*i* represents the number of slave nodes), and 2) Each *Si* section at each slave node is executed in parallel in the form of an OpenMP multithreaded process. Theoretical analysis results as well as experimental results have proved that the new model has helped the system performance increase linearly with the number of cores of processors on the slave nodes.

Implementing the new CAPE execution model requires rebuilding the most important CAPE tool, the checkpointer. This requires many important changes in the checkpointer, to make it to be able to handle the tasks of checkpointing on multi-threaded processes at the slave nodes. The author has put a lot of time and effort into this work and has successfully developed a new checkpointer, which runs at user space level, instead of running in kernel space of the operating system as before.

Achievements of the thesis include:

1. Proposing the operation model of CAPE on multi-core computing system;

2. Building checkpointing technique suitable for CAPE model on multi-core computer system;

3. Propose CAPE's data-sharing solution on the multi-core computing system.

4. Build a software system corresponding to the proposed new CAPE model and evaluate the effectiveness of this model in comparison with the previous CAPE model and with MPI.

Future works

From the results obtained in the thesis, some future works are:

1. Continue to develop the execution model of CAPE in the direction of exploiting the capabilities of graphics cards.

2. If invested and seriously continued by a system software development organization, CAPE is fully capable of becoming a good tool to bring OpenMP on distributed systems, providing an easy-to-learn, easy-to-program, and high-performance parallel programming solution on these systems. We have published the source code of CAPE software on Github - the community's open source code repository - with the orientation of providing CAPE source code for the community to develop together.

LIST OF PUBLISHING BY AUTHOR

[CT1]. **Đỗ Xuân Huyền**, Hà Viết Hải (2018), "Các giải pháp mở rộng mô hình hoạt động của CAPE cho mạng máy tính sử dụng bộ vi xử lý đa lõi". *Tạp chí Khoa học và Công nghệ (Trường Đại học Khoa học - Đại học Huế)*, ISSN 2354-0842, Tập 12, Số 1, 2018, Tr. 51–61. http://joshusc.hueuni.edu.vn/jos_articles.php?article_id=385.

[CT2]. Hà Viết Hải, **Đỗ Xuân Huyền** (2018), "Mở rộng mô hình hoạt động của CAPE bằng sử dụng máy ảo". *Tạp chí Tạp chí Khoa học Đại học Huế: Kỹ thuật và Công nghệ*; ISSN 1859–1388, Tập 127, Số 2A, 2018, Tr. 159–168. http://jos.hueuni.edu.vn/index.php/hujos-tt/article/view/4795.

[CT3]. **Đỗ Xuân Huyền**, Hà Viết Hải, Trần Văn Long (2019), "Xử lý các mệnh đề về dữ liệu chia sẻ của OpenMP trên các hệ thống sử dụng bộ nhớ phân tán". *Kỷ yếu Hội nghị khoa học quốc gia lần thứ XII về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR)*. Huế 2019, Tr. 577–583. https://doi.org/10.15625/vap.2019.00074.

[CT4]. **X. H. Do**, V. H. Ha, V. L. Tran, and É. Renault, "The technique of locking memory on Linux operating system - Application in checkpointing," 2019 6th NAFOSTED Conference on Information and Computer Science (NICS), 2019, pp. 178-183. https://doi.org/10.1109/NICS48868.2019.9023816.

[CT5]. **Xuan Huyen Do**, Viet Hai Ha, Van Long Tran, Eric Renault (2021), "New execution model of CAPE using multiple threads on multi-core clusters", *ETRI Journal (SCIE-Q2)*, May, 2021. https://doi.org/10.4218/etrij.2020-0201.