ÁP DỤNG CHIẾN LƯỢC CHỌN VÙNG VÀ THUẬT TOÁN NSGA2 CHO ÁNH XẠ CÁC ỨNG DỤNG CÓ THỂ ĐIỀU CHỈNH CHẤT LƯỢNG LÊN NỀN TẢNG TÁI CẤU HÌNH NoC

NGUYỄN VĂN CƯỜNG[1], TRỊNH MINH THIÊN[2]

[1]Trường Đại học Công nghiệp Tp.HCM,
[2]Trường Đại học Phú Yên

nguyenvancuong@iuh.edu.vn,, thientrinh1976@gmail.com

Tóm tắt. Các hệ thống trên chip cấu hình lại được dựa trên FPGA và mạng trên chip (NoC: Network on Chip) là một xu hướng mới nhằm cung cấp hiệu năng cao, khả năng linh hoạt, cắt giảm chi phí và thời gian đưa sản phẩm ra thị trường cho các hệ thống nhúng. Bài toán ánh xạ các ứng dụng có thể điều chỉnh mức chất lượng lên nền tảng NoC cấu hình lại được không đồng nhất tại thời gian chạy với ràng buộc tài nguyên mà vẫn đảm bảo chất lượng dịch vụ tổng thể tối đa cho các ứng dụng là một thử thách lớn. Trong bài báo này, chúng tôi sử dụng một cách tiếp cận mới để giải quyết bài toán ánh xạ các ứng dụng có thể điều chỉnh mức chất lên nền tảng NoC có khả năng cấu hình lại được với ràng buộc tài nguyên mà vẫn đảm bảo chất lượng dịch vụ tổng thể tối đa cho các ứng dụng, đó là sự kết hợp giữa chiến lược chọn vùng gần lồi (near convex region) và thuật toán tiến hóa đa mục tiêu NSGA2. Kết quả mô phỏng chỉ ra rằng cách tiếp cận mới này cải thiện trung bình lên đến 36,11% chất lượng dịch vụ tổng thể so với một vài giải pháp đã tồn tại.
Từ khóa. Mạng trên chip, FPGA, ánh xạ, vùng cấu hình lại được, mức chất lượng, NSGA2, vùng gần lồi.

MAPPING OF QUALITY ADJUSTABLE APPLICATIONS ON NoC-BASED RECONFIGURABLE PLATFORMS USING REGION SELECTION STRATEGY AND NSGA2 ALGORITHM

Abstract. Network on Chip (NoC) and FPGA-based reconfigurable Systems on Chip are a new trend to provide high performance, flexibility, reducing cost and time to market for the embedded systems. The problem of mapping quality adjustable applications onto heterogeneous NoC-based reconfigurable platforms at run-time with resource constraints while ensuring the maximum overall quality of service of the applications is a big challenge. In this paper, we use a new approach to solve the problem of mapping quality adjustable applications onto heterogeneous NoC-based dynamically reconfigurable platforms under resource constraints while ensuring the maximum overall quality of service (QoS) of the applications. That is a combine between near convex region selection strategy and the NSGA2 algorithm. Simulation results show that this new approach improves on average up to 36,11% of overall QoS in comparison with some existing solutions.
Keywords. Network on Chip, FPGA, mapping, reconfigurable region, quality level, NSGA2, near covex region.

1. ĐẶT VẤN ĐỀ

Ngày nay, các thiết bị nhúng đang trở nên quan trọng trong cuộc sống của chúng ta. Xu hướng triển khai nhiều ứng dụng và tích hợp nhiều tính năng lên chúng ngày càng gia tăng trong khi tài nguyên của chúng luôn bị giới hạn. Điều này đòi hỏi người thiết kế phải có những giải pháp linh hoạt và hiệu quả. Các FPGA hiện đại không ngừng tăng số lượng tài nguyên trên nó, giá thành tiếp tục giảm, các tính năng mới được tích hợp đặc biệt là khả năng cấu hình lại từng phần động. Do vậy, FPGA là một lựa chọn thích hợp để phát triển nhanh một nền tảng nhúng đa lõi linh hoạt và có hiệu năng cao. Theo hướng này, một số hệ thống nhúng dựa trên FPGA đã được phát triển để hỗ trợ cho các ứng dụng đa phương tiện và các ứng

dụng xử lý tín hiệu [1-5]. Các ứng dụng này thường yêu cầu cơ sở hạ tầng truyền thông hiệu năng cao và có khả năng xử lý dữ liệu nhanh.

Nhằm cung cấp một cơ sở hạ tầng truyền thông hiệu năng cao để kết nối các phần tử xử lý (PE) khác nhau của một SoC phức tạp, mạng trên chip đã được đề xuất như là một giải pháp thay thế cho các kiến trúc truyền thống như Bus và kết nối điểm-điểm [6-7]. Ngoài ra, một nền tảng cấu hình lại không đồng nhất với sự linh hoạt của các bộ xử lý nhúng và hiệu quả tính toán của một số vùng cấu hình lại được dựa trên NoC đã chứng minh là có nhiều ưu điểm hơn so với các nền tảng đồng nhất [8-9]. Trong một nền tảng như vậy, các bộ xử lý nhúng thường được sử dụng để quản lý và thực hiện một số tác vụ có độ phức tạp thấp trong khi các vùng cấu hình lại trên FPGA được sử dụng để tăng tốc tính toán cho các tác vụ có độ phức tạp cao. Khả năng cấu hình động cho phép nền tảng FPGA thích nghi với các yêu cầu xử lý thay đổi của các ứng dụng. Ngoài ra, nhiều ứng dụng được thiết kế theo cách mà mức chất lượng của chúng có thể được điều chỉnh để phù hợp với khả năng xử lý của các nền tảng phần cứng. Do vậy, bài toán ánh xạ các ứng dụng có thể điều chỉnh mức chất lượng lên nền tảng tự cấu hình lại động dựa trên NoC không đồng nhất với ràng buộc tài nguyên trong khi vẫn đảm bảo QoS tổng thể tối đa cho các ứng dụng là bài toán cần được quan tâm nghiên cứu.

Trong nghiên cứu trước [10], chúng tôi đã xây dựng bài toán ánh xạ nói trên và đề xuất một thuật toán tìm kiếm đầy đủ để giải quyết bài toán này. Tuy nhiên, giải pháp này chỉ thực hiện hiệu quả trong trường hợp nền tảng phần cứng có kích thước nhỏ, số lượng ứng dụng triển khai lên nó không lớn và số mức chất lượng trong mỗi ứng dụng nhỏ. Khi kích thước mạng tăng, kích thước của các ứng dụng lớn thì giải pháp này không còn phù hợp vì thời gian thực hiện thuật toán ánh xạ lớn.

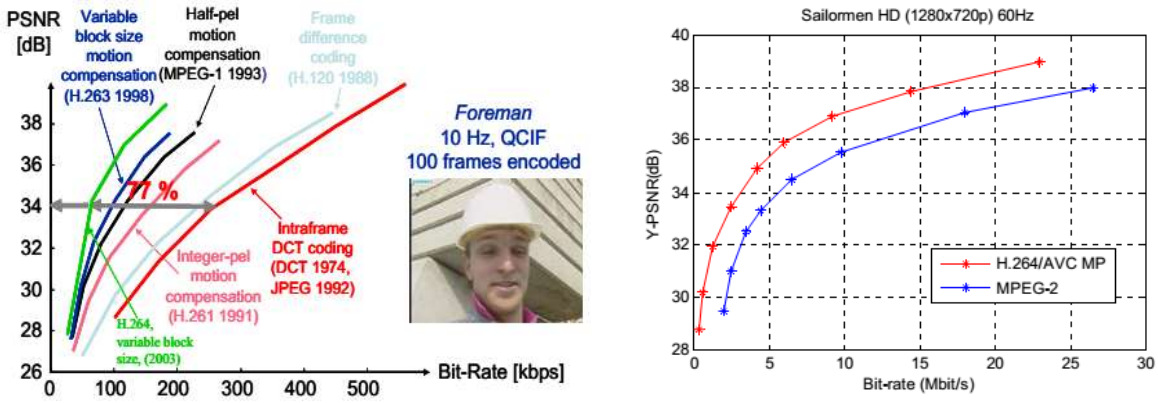
Để vượt qua giới hạn này, chúng tôi đề xuất một cách tiếp cận mới hiệu quả dựa trên chiến lược chọn vùng gần lồi kết hợp với thuật toán tiến hóa đa mục tiêu NSGA2 [11]. Đầu tiên, một vùng gần lồi sẽ được lựa chọn khi có ứng dụng đưa vào hệ thống, tiếp theo thuật toán NSGA2 sẽ thực hiện ánh xạ các tác vụ của ứng dụng này vào vùng gần lồi đã chọn. Giải pháp cho phép triển khai nhanh và linh hoạt các ứng dụng lên nền tảng. Ngoài ra, nó cũng có thể dễ dàng thêm vào hệ thống các ứng dụng mới trong tương lai.

Phần còn lại của bài báo được tổ chức như sau: Trong Mục 2, chúng tôi sẽ mô tả các định nghĩa và bài toán ánh xạ. Ứng dụng một vài chiến lược chọn vùng gần lồi và thuật toán tiến hóa đa mục tiêu NSGA2 vào giải quyết bài toán ánh xạ sẽ được giới thiệu trong Mục 3. Trong Mục 4, chúng tôi sẽ trình bày kết quả mô phỏng và thảo luận. Cuối cùng, kết luận sẽ được đưa ra trong Mục 5.

2. CÁC ĐỊNH NGHĨA VÀ BÀI TOÁN ÁNH XẠ

2.1. Mô hình ứng dụng

Mỗi ứng dụng có một đồ thị tác vụ cố định và chất lượng của mỗi ứng dụng có thể được điều chỉnh bằng cách thay đổi số lượng dữ liệu được xử lý bằng đồ thị tác vụ ngõ vào. Ví dụ, trong một ứng dụng đồ họa 3D các công cụ hiển thị hình ảnh 3D là cố định và phụ thuộc vào số lượng tam giác sử dụng để biểu diễn cho một đối tượng 3D, ta có thể có các chất lượng khác nhau cho các đối tượng đó [12]. Một trường hợp khác, đó là một ứng dụng truyền tải video qua giao thức http trong [13], trong đó các bộ giải mã video là cố định và số lượng các bit dữ liệu được sử dụng để biểu diễn nội dung video sẽ xác định chất lượng của video được giải mã. Hình 1 biểu diễn mối quan hệ giữa chất lượng video với số lượng bit dữ liệu thông qua thông số Bit-rate.



Hình 1: Ảnh hưởng của dữ liệu đến chất lượng video [14-15]

2.1.1. Đồ thị ứng dụng

Định nghĩa 1: Một đồ thị tác vụ ứng dụng được biểu diễn dưới dạng một đồ thị có hướng $ATG=(V,E)$, trong đó V là tập các tác vụ ứng dụng và E là tập tất cả các cạnh, mỗi cạnh trong đồ thị được kết nối giữa hai tác vụ như Hình 2.

Mỗi đỉnh $v_k \in V$ là một tác vụ với bộ thông số $(t_{\text{id}}, t_{\text{type}}, t_{\text{comp}}, t_{\text{reqcomp}})$. Trong đó, t_{id} là thông số nhận dạng; t_{type} là kiểu tác vụ (tác vụ phần cứng t_{HW} hoặc tác vụ phần mềm t_{SW}), các tác vụ phần cứng được tạo ra bởi ngôn ngữ phần cứng HDL và các tác vụ phần mềm được tạo ra bởi ngôn ngữ lập trình bậc cao như C/C++; t_{comp} là thời gian tính toán của tác vụ khi nó thực hiện trên tài nguyên phần cứng hoặc phần mềm. Một tác vụ được thực hiện trên tài nguyên phần cứng sẽ có tốc độ tính toán nhanh hơn khi nó thực hiện trên tài nguyên phần mềm [16-17]; t_{reqcomp} là thời gian yêu cầu tối thiểu để thực hiện hoàn thành một tác vụ trên tài nguyên phần cứng hoặc phần mềm.

Mỗi cạnh e_{rs} biểu diễn mối quan hệ giữa tác vụ v_r và v_s. Nó có các thông số như $(t_{\text{comm}}, t_{\text{reqcomm}})$. Trong đó, t_{comm} là trễ truyền thông khi truyền các gói tin từ tác vụ v_r đến v_s; t_{reqcomm} là yêu cầu trễ truyền thông tối thiểu khi truyền các gói tin từ tác vụ v_r đến v_s.

2.1.2. Mô hình chất lượng

Chúng ta xem xét trường hợp nhiều ứng dụng chạy đồng thời trên một thiết bị nhúng cầm tay, mỗi ứng dụng yêu cầu chạy tại các mức chất lượng khác nhau. Khi đó, thiết bị phải có khả năng thích nghi theo các yêu cầu của các ứng dụng bằng cách điều chỉnh cấp phát tài nguyên phần cứng cho từng ứng dụng tại thời gian chạy. Trong mục này, chúng tôi xem xét một ứng dụng được biểu diễn dưới nhiều mức chất lượng khác nhau.

Xem xét ứng dụng App_i có N_i mức chất lượng $Q_{i1}, Q_{i2}, Q_{i3}, ..., Q_{iN_i}$. Mỗi mức chất lượng yêu cầu một tập các tài nguyên bao gồm thời gian tính toán, thời gian truyền thông, diện tích phần cứng và năng lượng. Mức chất lượng cao hơn sẽ yêu cầu tài nguyên nhiều hơn [17-18]. Ví dụ, giả sử ứng dụng App_1 có các mức chất lượng $Q_{11} > Q_{12} > Q_{13} > \ldots > Q_{1N_1}$ thì thời gian cần thực hiện ứng dụng tại mỗi mức chất lượng tương ứng sẽ là $t_{11} > t_{12} > t_{13} > \ldots > t_{1N_1}$. Mỗi mức chất lượng Q_{ij} cung cấp cho người xem một cảm giác chất lượng, cảm giác chất lượng này có thể biểu diễn bởi một giá trị lợi ích B_{ij}. Chúng tôi sử dụng mô hình biểu diễn mối quan hệ giữa giá trị lợi ích và mức chất lượng của ứng dụng như đã trình bày trong [18]. Ứng dụng đạt mức chất lượng cao hơn thì sẽ có giá trị lợi ích lớn hơn. Ngược lại, mức

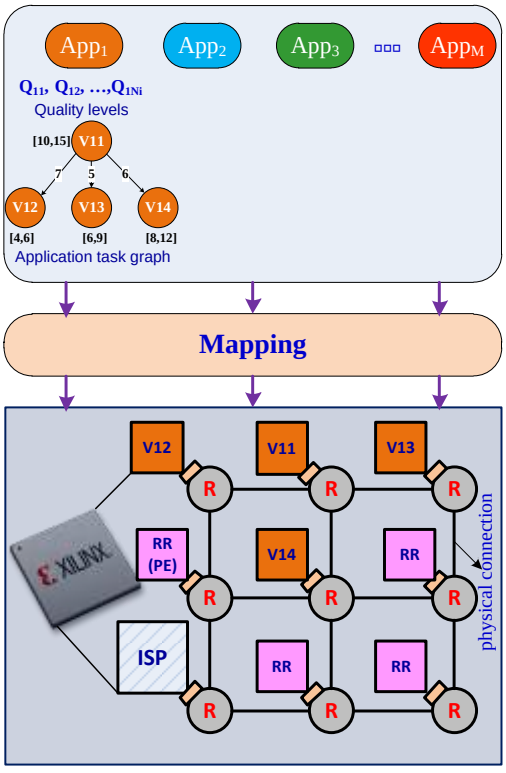
chất lượng thấp hơn thì giá trị lợi ích sẽ thấp hơn. Tổng quát, ứng dụng A_i có các mức chất lượng $Q_{i1} > Q_{i2} > Q_{i3} > \ldots > Q_{iN_i}$ thì giá trị lợi ích tương ứng sẽ là $B_{i1} > B_{i2} > B_{i3} > \ldots > B_{iN_i}$ $(0 < B \le 1)$.

2.2. Mô hình phần cứng

Nền tảng phần cứng được xem xét là một nền tảng cấu hình lại được được thực hiện trên FPGA dựa trên kiến trúc NoC như Hình 2. Nền tảng phần cứng bao gồm một vi xử lý nhúng ISP (Microblaze/ARM) và các PE phần cứng cấu hình lại được, chúng được kết nối với nhau qua mạng truyền thông NoC. NoC sử dụng cấu trúc lưới hai chiều, chuyển mạch gói, điều khiển luồng wormhole kết hợp với kênh ảo và thuật toán định tuyến XY. Tác vụ phần mềm được thực hiện trên ISP, tác vụ phần cứng được thực hiện trên các PE cấu hình lại được. Chúng tôi giả định rằng mỗi PE phần cứng có khả năng hỗ trợ chỉ một tác vụ. Trong khi đó, vi xử lý nhúng ISP có thể hỗ trợ một hoặc nhiều hơn một tác vụ và chịu trách nhiệm quản lý hệ thống bao gồm ánh xạ tác vụ, lập lịch tác vụ, kiểm soát tài nguyên và điều khiển cấu hình lại.

Định nghĩa 2: Một nền tảng phần cứng cấu hình lại được dựa trên NoC được biểu diễn bởi một đồ thị $NoC = N(P, L)$ trong đó, P là tập các phần tử xử lý PE, và L là tập các kết nối vật lý giữa hai bộ định tuyến.

Mỗi PE $p_{xy} \in P$ được biểu diễn bởi các thông số $\left(p_{\text{id}}, p_{\text{add}}, p_{\text{type}}, p_{\text{use}}\right)$, trong đó p_{id} là thông số nhận dạng PE, p_{add} là địa chỉ PE, p_{type} là kiểu PE gồm có PE cứng p_{HW} và PE mềm p_{SW}, p_{use} là thông số trạng thái của PE (rỗi và bận).



Hình 2: Mô hình hệ thống

2.3. Bài toán ánh xạ

Bài toán ánh xạ được xem xét như sau: Có *M* ứng dụng được triển khai đồng thời hoặc tuần tự lên nền tảng phần cứng. Các ứng dụng này có thể biết trước, tuy nhiên, thứ tự xuất hiện của chúng là không biết trước và không phụ thuộc nhau. Mục tiêu của bài toán ánh xạ là đạt được chất lượng dịch vụ tổng thể cực

đại khi triển khai *M* ứng dụng lên nền tảng phần cứng với các ràng buộc tài nguyên. Chúng tôi mở rộng bài toán ánh xạ này trong [10] theo cách tiếp cận do chúng tôi đề xuất như sau:

Cho một tập các đồ thị ứng dụng của M ứng dụng vào và trạng thái nền tảng phần cứng.

Tìm R vùng gần lồi trên nền tảng phần cứng và hàm ánh xạ *map(), **sao cho*** hàm *map()* thực hiện ánh xạ các tác vụ của ứng dụng thứ App_i vào vùng R_i, $\forall v_{ik} \in V$, $map(v_{ik}) \to p_{xy}$ trong R_i, $p_{xy} \in P$ với các mục tiêu sau:

$$B = \frac{1}{M}\left\{\sum_{i=1}^{M}\beta_i \sum_{j=1}^{N_i}\delta_{ij}B_{ij}\right\}_{\max} \quad (1)$$

$$T = \left\{\sum_{i=1}^{M}\sum_{j=1}^{N_i}\delta_{ij}t_{ij}\right\}_{\min} \quad (2)$$

Thỏa mãn các điều kiện: $\sum_{j=1}^{N_i}\delta_{ij} = 1$, $\delta_{ij} = 1$ nếu Q_{ij} được lựa chọn, 0 cho các trường hợp khác.

$$\begin{cases} \sum t_{\mathrm{HW}} \le \sum p_{\mathrm{HW}} \\ t_{\mathrm{comp}_{ij}} \le t_{\mathrm{reqcomp}_{ij}} \\ t_{\mathrm{comm}_{ij}} \le t_{\mathrm{reqcomm}_{ij}} \end{cases} \quad (3)$$

3. VÙNG GẦN LỒI VÀ THUẬT TOÁN NSGA2

3.1. Vùng gần lồi

Chọn các vùng gần lồi liền kề trên nền tảng để ánh xạ các tác vụ của các ứng dụng vào chúng là một giải pháp hiệu quả cho bài toán ánh xạ, nó đã được chứng minh trong các nghiên cứu [19-20]. Trong bài báo này, chúng tôi sử dụng hai chiến lược chọn vùng gần lồi để chọn một vùng gần lồi khi có một ứng dụng đưa vào hệ thống trước khi thuật toán ánh xạ thực hiện. Một là, chiến lược chọn vùng gần lồi NF trong [20], chiến lược này cố gắng tìm các PE trống trong nền tảng để tạo thành một vùng gần lồi cho ứng dụng vào với xem xét tổng khoảng cách Manhattan (MD: Manhattan Distance) tối thiểu. Hai là, chiến lược chọn vùng do chúng tôi đề xuất trong [21]. Chiến lược này thực hiện theo hai bước: Bước 1, tính toán cấp phát số lượng PE cứng/mềm tương ứng với mức chất lượng của ứng dụng yêu cầu. Bước 2, tìm một vùng gần lồi dựa trên phương pháp góc quét hình học và sử dụng thêm ràng buộc khoảng cách MD tối thiểu để chọn một vùng gần lồi tối ưu cho ứng dụng.

3.2. Thuật toán NSGA2

Sau khi một vùng gần lồi được chọn, thuật toán NSGA2 sẽ thực hiện việc ánh xạ các tác vụ của ứng dụng vào vùng lồi đã chọn. Mục đích của bước này là đặt các tác vụ của ứng dụng vào vùng gần lồi đã chọn sao cho mức chất lượng của ứng dụng có thể đạt được cao nhất, giảm thiểu trễ truyền thông cho ứng dụng.

Với thuật toán NSGA2, mỗi một cá thể chính là một cách ánh xạ các tác vụ của ứng dụng vào vị trí các PE được chọn vùng.

3.2.1. Khởi tạo quần thể cha

Để khởi tạo quần thể cha, chúng ta cần phải biết kích thước vùng của ứng dụng ánh xạ (size), kích thước quần thể và số nhiễm sắc thể của một cá thể (size_chrom) là bao nhiêu? Đối với ứng dụng được chọn vùng thì kích thước vùng của ứng dụng ánh xạ sẽ bằng số nhiễm sắc thể của một cá thể. Ngược lại, đối với ứng dụng không được chọn vùng thì kích thước vùng của ứng dụng ánh xạ sẽ lớn hơn hoặc bằng số nhiễm sắc thể của một cá thể. Ví dụ khởi tạo quần thể cho một ứng dụng có năm tác vụ ánh xạ trên vị trí chọn vùng PE = {0, 1, 2, 5, 6} thì size = size_chrom = 5 và PE_id = {0, 1, 2, 3, 4}.

3.2.2. Đánh giá quần thể

Đánh giá quần thể bản chất là đi đánh giá các mục tiêu và độ vi phạm ràng buộc của từng cá thể trong quần thể. Trong bài toán ánh xạ này, mỗi cá thể sẽ có hai mục tiêu và các ràng buộc được trình bày ở nhóm công thức (1), (2), và (3). Ngoài ra, có thêm một ràng buộc khác khá quan trọng đó là mỗi cá thể trong quần thể phải đạt được mức chất lượng tối thiểu. Nếu cá thể nào không thỏa mãn đạt được mức chất lượng thấp nhất sẽ được coi là vi phạm ràng buộc.

3.2.3. Sắp xếp thứ hạng và tính khoảng cách hội tụ cho quần thể cha

Toán tử so sánh trội ($\succ_n$) được xác định dựa trên hai tiêu chí:

- Độ vi phạm ràng buộc (constraint).
- Các mục tiêu (objectives).

Sau khi đánh giá quần thể, thuật toán tiếp tục sắp xếp thứ hạng và tính khoảng cách hội tụ cho từng cá thể trong quần thể. Hình 3 trình bày thuật toán tìm nhóm cá thể có cùng thứ hạng. Trong thuật toán này, tất cả các cá thể trong quần thể *P* sẽ kiểm tra với nhóm cá thể không trội *P'*. Đầu tiên, thuật toán đưa tạm thời một cá thể *p* trong quần thể *P* vào nhóm cá thể không trội *P'*. Sau đó cá thể *p* so sánh với tất cả các cá thể trong *P'*. Nếu cá thể *p* trội hơn bất kì cá thể *q* nào trong *P'* thì sẽ loại cá thể *q* đó khỏi *P'*. Trái lại, nếu bất kì cá thể *q* nào trong *P'* trội hơn cá thể *p* thì sẽ loại cá thể *p* ra khỏi *P'*. Theo cách này, nếu không tồn tại cá thể *q* nào trong *P'* trội hơn *p* thì *p* chính thức sẽ được đưa vào trong *P'*. Khi thuật toán kết thúc thì tất cả các cá thể trong *P'* sẽ không trội hơn nhau hay nói cách khác là các cá thể này có cùng thứ hạng với nhau.

```
P' = find-nondominated-front(P )
   P' = ∅
   for each p ∈ P ∧ p ∉ P'
      P' = P' ∪ {p}                          //thêm tạm thời p vào P'
      for each q ∈ P'∧ q ≠ p                  //so sánh p với tất cả thành viên của P'
         if p ≻n q then P' = P' \ {q}         //nếu p trội hơn q thì loại bỏ q khỏi P'
         else if q ≻n p then P' = P' \ {p} //nếu q trội hơn p thì loại bỏ p khỏi P'
       end for
   end for
```

Hình 3: Thuật toán tìm nhóm cá thể cùng thứ hạng

Sau khi tìm được nhóm cá thể có cùng thứ hạng (F_i), một thuật toán tính khoảng cách hội tụ cho từng cá thể trong nhóm cá thể đó sẽ được thực hiện. Hình 4 mô tả thuật toán tính khoảng cách hội tụ nhóm cá thể F_i. Đầu tiên, thuật toán sẽ xem kích thước của nhóm cá thể F_i. Sau đó sẽ khởi tạo khoảng cách hội tụ cho từng cá thể trong F_i bằng không và bắt đầu thực hiện sắp xếp nhóm cá thể F_i theo chiều tăng dần của mục tiêu thời gian. Sau khi sắp xếp, những cá thể đứng ở đầu và cuối sẽ có khoảng cách hội tụ là vô cùng. Còn lại, khoảng cách hội tụ của cá thể thứ *i* sẽ bằng hiệu của mục tiêu thời gian của cá thể *(i+1)* trừ đi mục tiêu thời gian của cá thể *(i - 1)*. Tương tự như vậy, đối với mục tiêu là giá trị lợi ích.

```
crowding-distance-assignment( F_i )
   L = | F_i |                          // số lượng cá thể trong nhóm cá thể F_i
   for each i, set F_i[i]_distance = 0    // khởi tạo khoảng cách hội tụ cho mỗi cá thể
   end for
   for each objective m                   // xét với từng mục tiêu m
      F_i = sort(F_i,m)                   // sắp xếp nhóm cá thể F_i theo mục tiêu m
      F_i[1]_distance = F_i[L]_distance = ∞ // khởi tạo khoảng cách hội tụ nghiệm đầu và cuối
      for i = 2 to (L-1)                  // xét từ nghiệm thứ 2 đến L-1 trong F_i
         F_i[i]_distance = F_i[i]_distance + (F_i[i+1].m - F_i[i-1].m) //tính độ hội tụ của nghiệm i
       end for
   end for
```

Hình 4: Thuật toán tính khoảng cách hội tụ

Hình 5 trình bày thuật toán sắp xếp thứ hạng và tính khoảng cách hội tụ cho quần thể cha ban đầu. Thuật toán này sẽ lần lượt tìm nhóm cá thể F_{rank} từ thứ hạng thứ nhất đến thứ hạng thứ n sau đó tính khoảng cách hội tụ cho nhóm cá thể đó. Sau mỗi lần tìm được F_{rank} sẽ loại bỏ khỏi P với mục đích giảm kích thước của P cũng như dễ dàng để tìm nhóm cá thể F_{rank} tiếp theo. Thuật toán sẽ kết thúc khi P trống hay nói cách khác thuật toán đã hoàn thành việc tìm ra các nhóm cá thể không trội cùng thứ hạng và tính khoảng cách hội tụ cho chúng.

```
F = fast-nondominated-sort(P )
    i = 1
    while P ≠ ∅
        F_i = find-nondominated-front(P )
        P = P \ F_i
        i = i +1
    end while
```

Hình 5: Thuật toán sắp xếp thứ hạng và tính khoảng cách hội tụ

3.2.4. Lựa chọn cá thể bố mẹ trong quần thể cha

Việc lựa chọn cá thể bố mẹ trong quần thể cha nhằm mục đích chọn các cá thể bố mẹ tốt để thực hiện phép lai ghép để tạo ra các cá thể con. Khi chọn ra hai cá thể trong quần thể, việc lựa chọn cá nào tốt hơn để thực hiện lai ghép dựa trên ba tiêu chí đó là:

- Độ vi phạm ràng buộc (constraint).
- Thứ hạng (rank).
- Khoảng cách hội tụ (crowd-dist).

Đối với tiêu chí độ vi phạm ràng buộc và thứ hạng thì giá trị nhỏ hơn sẽ tốt hơn. Ví dụ a.contrain = 0.0 < b.contrain = 0.5 có nghĩa là a tốt hơn b. Tương tự, a.rank = 1 < b.rank = 3 thì a sẽ tốt hơn b. Nếu dựa trên ba tiêu chí trên mà hai cá thể a và b vẫn bằng nhau thì sẽ chọn ngẫu nhiên a hoặc b. Hình 6 mô tả thuật toán lựa chọn một cá thể tốt trong hai cá thể a và b.

```
individual = tournament(a, b)
    if a.contrain < b.contrain then
        individual = a
    else if a.contrain > b.contrain then
        individual = b
    else
        if a.rank < b.rank then
            individual = a
        else if a.rank > b.rank then
            individual = b
        else
            if a.crowd-dist > b.crowd-dist then
                individual = a
            else if a.crowd-dist < b.crowd-dist then
                individual = b
            else
                if randomperc() ≤ 0.5 then
                    individual = a
                else
                    individual = b
                end if
            end if
        end if
    end if
```

Hình 6: Thuật toán lựa chọn

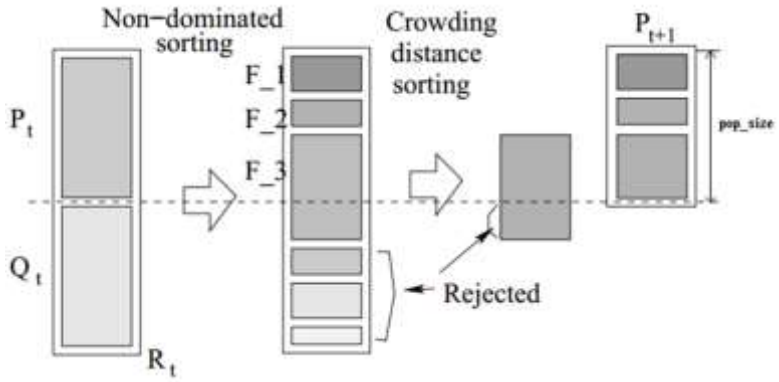
3.2.5. Chéo hóa và đột biến

Chéo hóa: Sau khi chọn ra cá thể bố và mẹ thì sẽ thực hiện chéo hóa để tạo ra hai cá thể con. Trong nghiên cứu này, chúng tôi thực hiện phương pháp chéo hóa nhị phân thứ tự hai điểm trên từng nhiễm sắc thể. Để áp dụng được phương pháp này, nhiễm sắc thể phải được mã hóa nhị phân. Việc chéo hóa nhị phân trên từng nhiễm sắc thể có thể tạo ra cá thể con tốt hơn và hạn chế việc tạo ra các cá thể con trùng nhau so với việc áp dụng phương pháp chéo hóa thứ tự hai điểm mà không mã hóa.

Đột biến: Quá trình đột biến cũng phụ thuộc vào xác suất đột biến. Nếu xác suất ngẫu nhiên mà nhỏ hơn hoặc bằng xác suất đột biến thì bit 0 sẽ chuyển sang bit 1 và ngược lại, bit 1 chuyển sang bit 0.

3.2.6. Sắp xếp thứ hạng, tính khoảng cách hội thụ cho quần thể cha, con và tạo quần thể cha mới

Sau khi hoàn thành việc đánh giá quần thể con, quần thể cha và quần thể con sẽ được hợp nhất để sắp xếp thứ hạng, tính khoảng cách hội tụ và chọn ra *pop_size* cá thể tốt nhất để tạo ra một quần thể cha mới. Như ví dụ ở Hình 7, quần thể cha *Pt* và quần thể con *Qt* được hợp thành một quần thể mới gọi là *Rt*. Tiếp đến quần thể *Rt* được sắp xếp thứ hạng và tính khoảng cách hội tụ. Như trong ví dụ, nhóm cá thể *F_1* có thứ hạng bằng một và có kích thước nhỏ hơn *pop_size* sẽ được đưa vào quần thể cha mới *Pt+1*. Tương tự như vậy, nhóm cá thể F_2 sẽ được đưa vào quần thể *Pt+1*. Tiếp tục tìm thấy nhóm cá thể F_3 nhưng lúc này khi đưa vào quần thể *Pt+1* thì đã vượt quá kích thước *pop_size*. Do vậy cần tính khoảng cách hội tụ của nhóm cá thể *F_3* và thực hiện sắp xếp để chọn ra những cá thể có độ quy tụ lớn nhất để đưa vào quần thể *Pt+1* để đạt đủ kích thước *pop_size* cá thể. Hình 8 mô tả thuật toán chọn lọc cá thể để tạo ra quần thể cha mới.



Hình 7: Quá trình chọn lọc để tạo ra quần thể cha mới

$R_t = P_t \cup Q_t$
$P_{t+1} = \emptyset$ //quần thể cha mới
i = 1
while $|P_{t+1}|$ < pop_size
  F_i = **find-nondominated-front**(R_t)// tìm nhóm cá thể F_i ở thứ hạng thứ i
  $R_t = R_t \setminus F_i$ // xóa F_i khỏi R_t
  if $|P_{t+1}| + |F_i| \leq$ pop_size **then**
    crowding-distance-assignment(F_i)
    $P_{t+1} = P_{t+1} \cup F_i$
    i = i +1
  else
    crowding-distance-assignment(F_i)
    quick_sort(F_i, compare) // sắp xếp theo chiều giảm dần crowding distance
    $P_{t+1} = P_{t+1} \cup F_i[1 : (\text{pop\_size} - |P_{t+1}|)]$ //hoàn thành tạo quần thể cha mới
  end if
end while

Hình 8: Thuật toán chọn lọc cá thể để tạo ra quần thể cha mới

4. KẾT QUẢ MÔ PHỎNG VÀ THẢO LUẬN

4.1. Thiết lập mô phỏng

Chương trình mô phỏng được viết bằng ngôn ngữ C++ chạy trên hệ điều hành Ubuntu 14.04 64bit, Intel Core i5-2520M 2.5 GHz, RAM 4GB. Các nền tảng phần cứng không đồng nhất có thể cấu hình lại được dựa trên NoC có kích thước khác nhau lần lượt là 6x6, 7x7, 8x8, và 9x9. Mỗi nền tảng gồm một ISP và một số vùng cấu hình lại được. ISP có thể thực hiện được tối đa đến 6 tác vụ, trong khi vùng cấu hình lại được chỉ thực hiện duy nhất 1 tác vụ. Bộ dữ liệu thực hiện mô phỏng gồm 20 ứng dụng có kích thước khác nhau thay đổi từ 7 đến 26 tác vụ được tạo ra từ công cụ TGFF trong [22]. Để đơn giản, chúng tôi xem xét mỗi ứng dụng với bốn mức chất lượng (ví dụ, Q_{i1}, Q_{i2}, Q_{i3} and Q_{i4}). Thời gian tính toán của các tác vụ, thời gian truyền thông giữa các cặp tác vụ được tạo ra một cách ngẫu nhiên cho mức chất lượng cao nhất. Đối với mức chất lượng thấp hơn, các thông số này được tạo ra từ các giá trị cao nhất theo mô hình chất lượng đã trình bày trong [17]. Giá trị lợi ích tại các mức chất lượng khác nhau của các ứng dụng cũng được tạo ta theo mô hình này.

Kịch bản mô phỏng trên 4 nền tảng với tổng số tác vụ của các ứng dụng được ánh xạ lên nền tảng thay đổi từ (x*y) đến (x*y+5). Trong đó, x, y lần lượt là số hàng và số cột của nền tảng phần cứng. Mỗi kịch bản sẽ thực hiện thay đổi thứ tự ánh xạ các ứng dụng đi vào theo hoán vị của tổng số ứng dụng. Ví dụ, ánh xạ bốn ứng dụng lên nền tảng có kích thước 8x8 thì sẽ có 24 trường hợp.

Các thông số được sử dụng trong thuật toán NSGA2 được mô tả trong Bảng 1 bao gồm kích thước quần thể, số thế hệ, xác suất chéo hóa, xác suất đột biến.

Bảng 1: Thông số mô phỏng thuật toán NSGA2

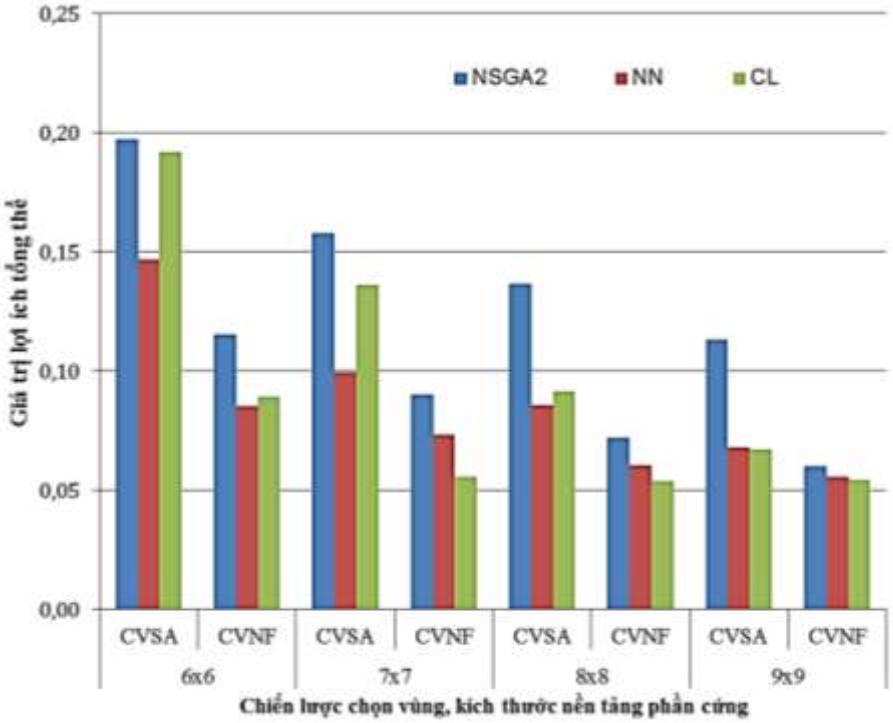
Thông số	**Giá trị**
Kích thước quần thể	600
Số thế hệ	1000
Xác suất chéo hóa	0,9
Xác suất đột biến	0,2

Để đánh giá tính hiệu quả của NSGA2, chúng tôi chọn hai thuật toán có liên quan đến ánh xạ đó là thuật toán Nearest Neighbor (NN) trong [23] và thuật toán của Chou Chen-Ling (CL) trong [19] để so sánh. Các thuật toán này cũng thực hiện ánh xạ các tác vụ của ứng dụng lên vùng gần lồi được tạo ra từ chiến lược chọn vùng NF (CVNF) trong [20] và chiến lược chọn vùng theo góc quét (SA) của chúng tôi (CVSA) trong [21]. Các thông số được thu thập và thống kê theo giá trị trung bình, bao gồm giá trị lợi ích tổng thể của các ứng dụng (OB: Overall Benefit), khoảng cách MD trung bình (AMD: Average Manhattan Distance), khoảng cách MD truyền thông trung bình (ACMD: Average Communication Manhattan Distance).

4.2. Kết quả và đánh giá

4.2.1. Đánh giá giá trị lợi ích tổng thể

Giá trị lợi ích đại diện cho mức độ hài lòng của người dùng khi nhận được một mức chất lượng nhất định. Khi người dùng nhận được chất lượng cao nhất của một ứng dụng, giá trị lợi ích bằng 1. Chất lượng thấp hơn sẽ có giá trị lợi ích nhỏ hơn [18]. Do vậy, chúng tôi sử dụng thông số này để đánh giá chất lượng đạt được cho các ứng dụng sau khi ánh xạ. Hình 9 chỉ ra giá trị OB của các ứng dụng đạt được khi triển khai các ứng dụng vào nền tảng phần cứng với các kích thước lần lượt là 6x6, 7x7, 8x8, và 9x9 theo các thuật toán khác nhau. Trong các trường hợp, thuật toán NSGA2 cho kết quả tốt hơn so với thuật toán NN và CL. Cụ thể, đối với chiến lược chọn vùng SA và nền tảng có kích thước 6x6, thuật toán NSGA2 cải thiện giá trị lợi ích tổng thể so với thuật toán NN và CL lần lượt 34,30%, và 2,57%. Đối với chiến lược chọn vùng NF, thuật toán NSGA2 cải thiện giá trị lợi ích tổng thể lần lượt là 35,78%, và 29,49%, so với thuật toán NN, CL.



Hình 9: Giá trị OB của các ứng dụng khi ánh xạ lên các nền tảng

Tương tự, đối với nền tảng phần cứng có kích thước 7x7, 8x8, và 9x9, thuật toán NSGA2 tăng giá trị lợi ích tổng thể so với các thuật toán NN và CL trong chiến lược chọn vùng SA lần lượt là [58,92%, 15,86%], [59,70%, 49,34%], và [66,20%, 69,25%]. Trong chiến lược chọn vùng NF, tăng lần lượt là [23,01%, 62,14%], [19,14%, 34,15%], và [7,60%, 10,30%].

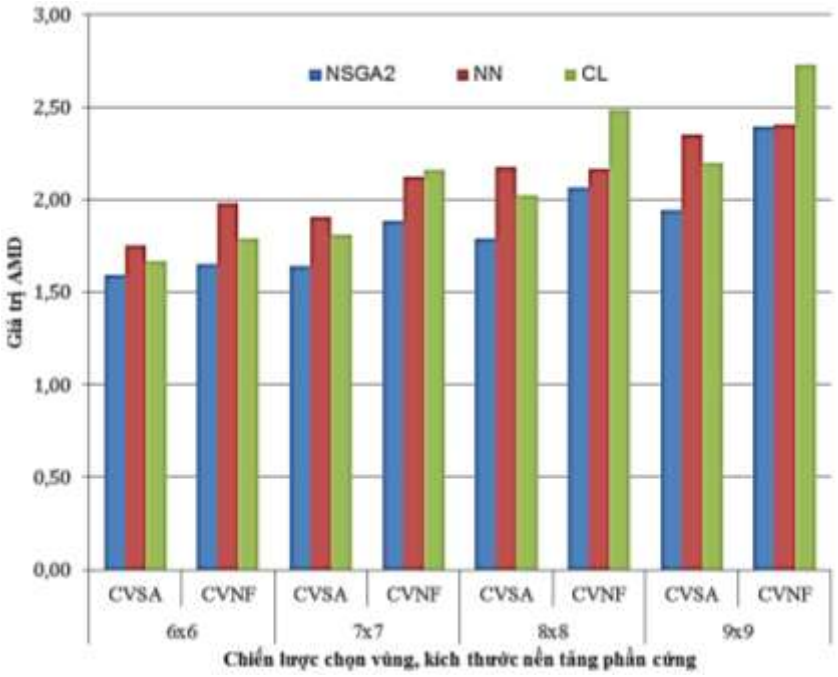
Từ kết quả này, chúng ta cũng dễ thấy rằng, tùy theo chiến lược chọn vùng mà tính hiệu quả của các thuật toán triển khai đạt được cũng khác nhau. Cụ thể, theo chiến lược chọn vùng SA thì tính hiệu quả của thuật toán NSGA2 đạt được tăng dần theo kích thước nền tảng phần cứng so với các thuật toán khác. Cụ thể, đối với nền tảng phần cứng có kích thước 9x9, giá trị lợi ích tổng thể thuật toán NSGA2 đạt được tăng lần lượt 66,20% và 69,25% so với thuật toán NN và CL trong khi kích thước nền tảng phần cứng 8x8 chỉ tăng lần lượt 59,70% và 49,34%. Ngược lại đối với chiến lược chọn vùng NF, tính hiệu quả của thuật toán NSGA2 giảm dần theo chiều tăng của kích thước nền tảng phần cứng. Cụ thể, đối với nền tảng phần cứng có kích thước 8x8, giá trị lợi ích tổng thể thuật toán NSGA2 đạt được lần lượt 19,14% và 34,15% so với thuật toán NN và CL nhưng khi kích thước nền tảng phần cứng tăng lên 9x9 thì kết quả giảm xuống còn chỉ 7,60% và 10,31%. Điều này cũng khẳng định được rằng, ngoài thuật toán ánh xạ thì chiến lược chọn vùng cũng ảnh hưởng đáng kể đến kết quả ánh xạ.

4.2.2. Đánh giá hiệu năng truyền thông

Tiếp theo, chúng tôi đánh giá hiệu năng truyền thông cho các thuật toán qua hai thông số AMD và ACMD. AMD là tỉ số giữa tổng số Hop của một ứng dụng đã được ánh xạ trên tổng các cạnh trong đồ thị tác vụ ứng dụng. Một thuật toán ánh xạ ứng dụng có AMD nhỏ hơn thì độ trễ và tiêu thụ năng lượng sẽ nhỏ hơn bởi vì các gói dữ liệu đi qua số lượng Hop nhỏ hơn. ACMD đại diện cho trễ truyền thông của các gói tin đi qua mỗi Hop. Tương tự, ACMD nhỏ thì trễ và tiêu thụ năng lượng của mạng cũng giảm.

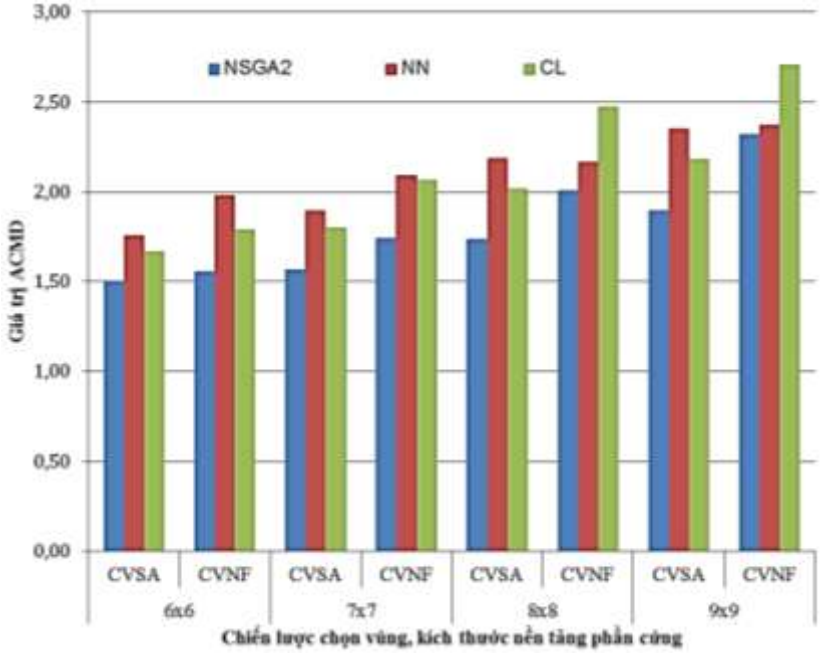
Giá trị AMD và ACMD của các ứng dụng khi triển khai lên các nền tảng với kích thước khác nhau (6x6, 7x7, 8x8 và 9x9) theo các thuật toán ánh xạ khác nhau được chỉ ra như Hình 10 và Hình 11. Các giá trị AMD và ACMD được cắt giảm lần lượt theo thuật toán NSGA2 và chiến lược chọn vùng SA so với thuật toán NN, CL cũng như chiến lược chọn vùng NF. Điều này có nghĩa rằng, thuật toán NSGA2 cho kết quả tốt hơn, tối ưu hơn về khoảng cách MD và giảm trễ truyền thông giữa các cặp tác vụ của ứng dụng sau khi ánh xạ so với các thuật toán khác. Chiến lược chọn vùng SA tạo ra các vùng có tổng khoảng cách MD

nhỏ hơn so với chiến lược chọn vùng NF. Kết quả này cho thấy, khi kết hợp giữa thuật toán NSGA2 với chọn vùng gần lồi sẽ giảm thiểu trễ truyền thông và tiết kiệm năng lượng cho hệ thống, cụ thể.



Hình 10: Giá trị AMD của các ứng dụng

Khi triển khai thuật toán NSGA2 với chiến lượt chọn vùng SA lên nền tảng có kích thước 6x6, thông số AMD được cắt giảm lần lượt 8,96% và 4,51% so với thuật toán NN và CL. Đối với các nền tảng 7x7, 8x8 và 9x9 thông số này được cải thiện [13,91%, 9,26%], [17,96%, 11,59%], và [17,42%, 11,51%]. Tương tự, đối với chiến lược chọn vùng NF, khi thực hiện ánh xạ thuật toán NSGA2 lên các nền tảng 6x6, 7x7, 8x8 và 9x9 thông số AMD được cắt giảm so với thuật toán NN và CL lần lượt là [16,51%, 7,56%], [11,24%, 12,82%], [4,71%, 16,75%], và [0,63%, 12,32%].



Hình 11: Giá trị ACMD khi triển khai các thuật toán

Thông số ACMD cũng được cắt giảm đáng kể khi thực hiện ánh xạ bởi thuật toán NSGA2 so với thuật toán NN và CL lên các chiến lược chọn vùng với các nền tảng 6x6, 7x7, 8x8 và 9x9. Cụ thể, đối với chọn vùng SA, thông số này lần lượt được cắt giảm [14,59%, 9,91%], [17,25%, 12,72%], [20,58%, 13,87%], và [19,44%, 13,30%]. Đối với chọn vùng NF, tỉ lệ phần trăm nhỏ hơn các thuật toán NN, CL là [21,47%, 13,10%], [16,56%, 15,49%], [7,44%, 18,91%], và [2,23%, 14,18%].

5. KẾT LUẬN

Trong bài báo này, chúng tôi đã đề xuất một cách tiếp cận mới để giải quyết bài toán ánh xạ các ứng dụng có thể điều chỉnh mức chất lên nền tảng NoC có khả năng cấu hình lại được với ràng buộc tài nguyên mà vẫn đảm bảo chất lượng dịch vụ tổng thể tối đa cho các ứng dụng, đó là kết hợp giữa chiến lược chọn vùng gần lồi và thuật toán tiến hóa đa mục tiêu NSGA2. Kết quả đạt được cho thấy giải pháp đã đề xuất hoàn toàn cho phép triển khai nhiều ứng dụng, các ứng dụng có kích thước lớn lên nền tảng phần cứng cấu hình lại được một cách linh hoạt, cải thiện chất lượng dịch vụ tổng thể của các ứng dụng sau khi triển khai chúng lên nền tảng phần cứng.

LỜI CẢM ƠN

Chúng tôi xin chân thành cảm ơn Trường Đại học Công nghiệp TP. Hồ Chí Minh vì đã hỗ trợ kinh phí để chúng tôi hoàn thành đề tài với mã số 182.QN01.

TÀI LIỆU THAM KHẢO

[1] Flasskamp Martin, Gregor Sievers, Johannes Ax, Christian Klarhorst, Thorsten Jungeblut, Wayne Kelly, Michael Thies, and Mario Porrmann, "Performance estimation of streaming applications for hierarchical MPSoCs", in Proceedings of the 2016 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools, p. 3,2016.

[2] Hsiao Pei-Yung, Shih-Yu Lin, and Shih-Shinh Huang, "An FPGA based human detection system with embedded platform". Microelectron. Eng., vol. 138, pp. 42–46, 2015.

[3] Kim Dong-Jin, Yeon-Jeong Ju, and Young-Seak Park, "An Implementation of SoC FPGA-based Real-time Object Recognition and Tracking System". IEMEK J. Embed. Syst. Appl., vol. 10, no. 6, pp. 363–372, 2015.

[4] Leibo L I U, WANG Dong, CHEN Yingjie, Z H U Min, Y I N Shouyi, and W E I Shaojun, "An Implementation of Multiple-Standard Video Decoder on a Mixed-Grained Reconfigurable Computing Platform". IEICE Trans. Inf. Syst., vol. 99, no. 5, pp. 1285–1295, 2016.

[5] Luo Junwen, Graeme Coapes, Terrence Mak, Tadashi Yamazaki, Chung Tin, and Patrick Degenaar, "Real-Time Simulation of Passage-of-Time Encoding in Cerebellum Using a Scalable FPGA-Based System". IEEE Trans. Biomed. Circuits Syst., vol. 10, no. 3, pp. 742–753, 2016.

[6] Benini Luca and Giovanni De Micheli, "Networks on chips: a new SoC paradigm". Computer (Long. Beach. Calif), vol. 35, no. 1, pp. 70–78, 2002.

[7] Pang Ke, Virginie Fresse, Suying Yao, and Otavio Alcantara De Lima, "Task mapping and mesh topology exploration for an FPGA-based network on chip". Microprocess. Microsyst., vol. 39, no. 3, pp. 189–199, 2015.

[8] Abdelfattah Mohamed S, Andrew Bitar, and Vaughn Betz, "Take the highway: Design for embedded NoCs on FPGAs", in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pp. 98–107, 2015.

[9] Kumar Rakesh, Dean M Tullsen, Parthasarathy Ranganathan, Norman P Jouppi, and Keith I Farkas, "Single-ISA heterogeneous multi-core architectures for multithreaded workload performance". ACM SIGARCH Comput. Archit. News, vol. 32, no. 2, p. 64, 2004.

[10] Nguyen Van Cuong, Nguyen Trong Bang, Le Dinh Tuyen, Pham Ngoc Nam, "Dynamic Mapping of Quality Adjustable Applications on NoC-based Reconfigurable Platforms", in The 2016 International Conference on Advanced Technologies for Communications (ATC), Hanoi, Vietnam, pp. 321–326, 2016.

[11] Vinícius, M., da Silva, C., Nedjah, N., & de Macedo Mourelle, L., "Optimal ip assignment for efficient noc-based system implementation using nsga-ii and microga". International Journal of Computational Intelligence Systems, 2(2), pp. 115-123, 2009.

[12] Ngoc N Pham, W van Raemdonck, Gauthier Lafruit, Geert Deconinck, and Rudy Lauwereins, "A qos framework for interactive 3d applications", in 10th Int. Conf. in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG-2002), pp. 317–325, 2002.

[13] Le Hung T, Hai N Nguyen, Nam Pham Ngoc, Anh T Pham, Hoa Le Minh, and Truong Cong Thang, "Quality-driven bitrate adaptation method for HTTP live-streaming", in 2015 IEEE International Conference on Communication Workshop (ICCW), pp. 1771–1776, 2015.

[14] Ghanbari M, D Crawford, M Fleury, E Khan, J Woods, H Lu, and R Razavi, "Future performance of video codecs". Video Netw. Lab. (November 2006), 2006.

[15] Wiegand Thomas, Heiko Schwarz, Anthony Joch, Faouzi Kossentini, and Gary J Sullivan, "Rate-constrained coder control and comparison of video coding standards". IEEE Trans. circuits Syst. video Technol., vol. 13, no. 7, pp. 688–703, 2003.

[16] Davis Don, Srinivas Beeravolu, and Ranjesh Jaganathan, "Hardware/Software Codesign for platforms FPGA". Xilinx Inc, 2005.

[17] Ngoc N Pham, G Lafruit, S Vernalde, and R Lauwereins, "Real-Time 3D Applications on Mobile Platforms With Run-Time Reconfigurable Hardware Accelerator", pp. 25–29, 2002.

[18] Ngoc Nam Pham, Gauthier Lafruit, Geert Deconinck, and Rudy Lauwereins, "A fast QoS adaptation algorithm for MPEG-4 multimedia applications", in International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services, pp. 92–105, 2002.

[19] Chou Chen-Ling and Radu Marculescu, "User-aware dynamic task allocation in networks-on-chip", in 2008 Design, Automation and Test in Europe, pp. 1232–1237, 2008.

[20] Chou Chen Ling, Umit Y. Ogras, and Radu Marculescu, "Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels". IEEE Trans. Comput. Des. Integr. Circuits Syst., vol. 27, no. 10, pp. 1866–1879, 2008.

[21] Nguyen Van Cuong, Le Dinh Tuyen, Dao Vu Tuan, Tran Thanh Hai, Pham Ngoc Nam, "Heuristics for Dynamic Mapping of Quality Adjustable Applications on NoC-based Reconfigurable Platforms", The Journal of Science & Technology of Technical Universities, 2017.

[22] Dick Robert P, David L Rhodes, and Wayne Wolf, "TGFF: task graphs for free", in Proceedings of the 6th international workshop on Hardware/software codesign, pp. 97–101, 1998.

[23] Carvalho Ewerson, Ney Calazans, and Fernando Moraes, "Heuristics for dynamic task mapping in NoC-based heterogeneous MPSoCs", in 18th IEEE/IFIP International Workshop on Rapid System Prototyping (RSP'07), pp. 34–40, 2007.

Ngày nhận bài:24/10/2018

Ngày chấp nhận đăng:10/02/2019