**ĐẠI HỌC QUỐC GIA HÀ NỘI**
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**Phạm Văn Hưởng**

# MỘT SỐ PHƯƠNG PHÁP TỐI ƯU TRONG CÁC GIAI ĐOẠN PHÁT TRIỂN PHẦN MỀM NHÚNG

Chuyên ngành: Kỹ thuật phần mềm

Mã số: 62 48 01 03

TÓM TẮT LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

**Hà Nội – 2015**

Công trình được hoàn thành tại:

**Trường Đại học Công nghệ - Đại học Quốc gia Hà Nội.**

Người hướng dẫn khoa học: PGS.TS. Nguyễn Ngọc Bình

Phản biện 1: TS. Phan Nguyên Hải

Phản biện 2: PGS.TS. Nguyễn Đình Hóa

Phản biện 3: PGS.TS. Trịnh Văn Loan

Luận án tiến sĩ được bảo vệ trước hội đồng cấp Đại học Quốc gia chấm luận án tiến sĩ họp tại Phòng 212, Nhà E3, Trường Đại học Công nghệ, 144 Xuân Thủy, Cầu Giấy, Hà Nội.

Vào hồi 9 giờ ngày 28 tháng 7 năm 2015.

Có thể tìm hiểu luận án tại:

- Thư viện Quốc gia Việt Nam
- Trung tâm Thông tin – Thư viện, Đại học Quốc gia Hà Nội

# MỞ ĐẦU

## 1. Tính cấp thiết và ý nghĩa khoa học

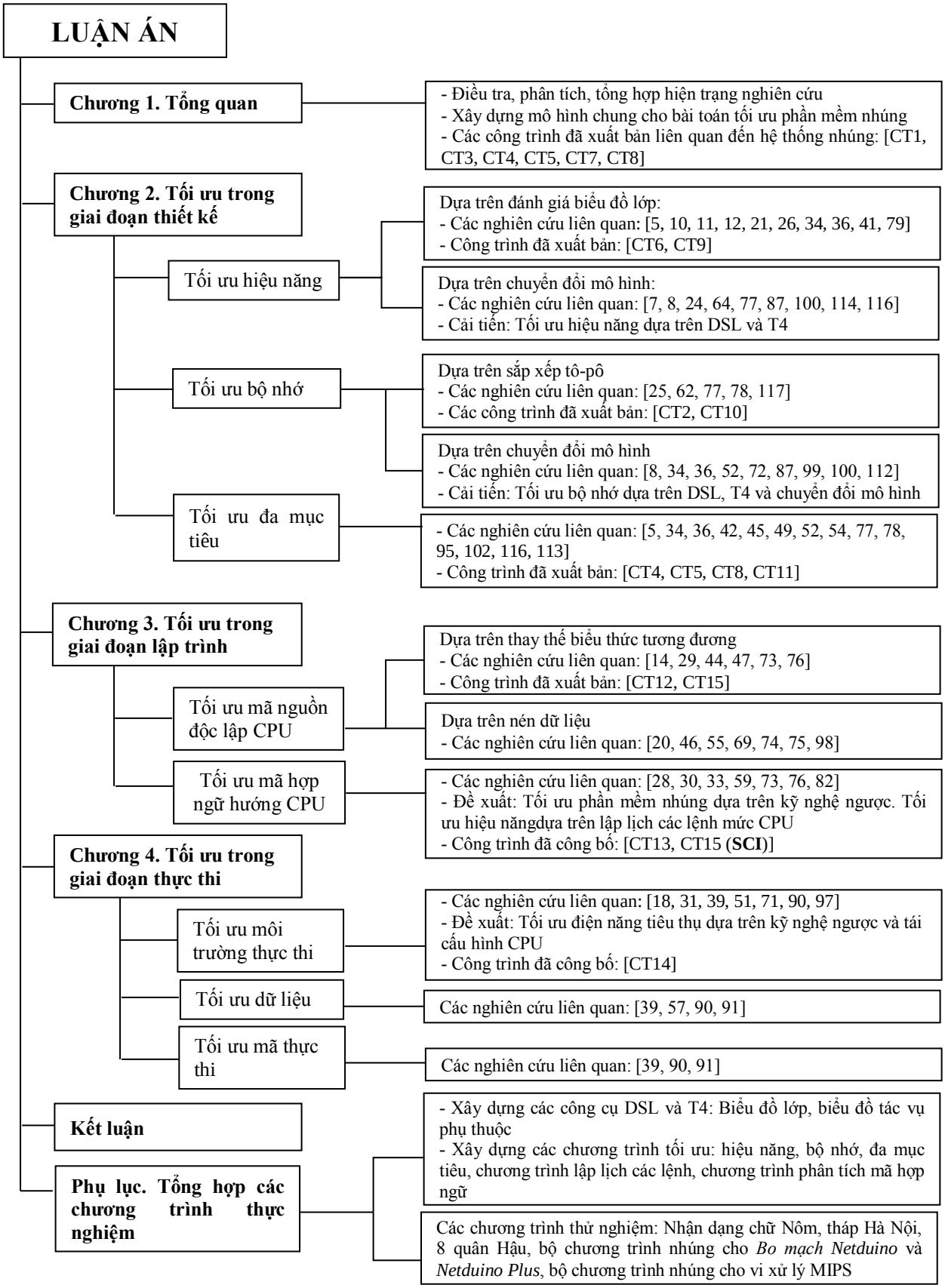
Hệ thống nhúng và phần mềm nhúng là một định hướng quan trọng, chiến lược trong xu thế phát triển mạnh mẽ của công nghệ thông tin. Các sản phẩm nhúng có mặt trong mọi lĩnh vực đời sống như ti-vi, tủ lạnh, máy giặt, ô-tô, v.v. Phần mềm nhúng thường thực thi trong môi trường giới hạn về tài nguyên phần cứng như tốc độ xử lý của CPU, dung lượng bộ nhớ, thời gian sống của pin, v.v. Do đó vấn đề tối ưu phần mềm nhúng có vai trò hết sức quan trọng và mang tính thời sự, cấp thiết. *Về mặt lý thuyết*, các phương pháp tối ưu được phát triển, thực nghiệm trong đề tài sẽ góp phần giải quyết những khó khăn như tối ưu trong giai đoạn thiết kế, tối ưu đa mục tiêu, tối ưu hướng đến các CPU chuyên dụng và tối ưu trong giai đoạn thực thi. *Về thực tiễn*, đề tài có khả năng ứng dụng thực tế, góp phần giải quyết một số vấn đề đang được quan tâm của các công ty, doanh nghiệp phát triển hệ thống nhúng, phần mềm nhúng như hiệu năng, năng lượng, v.v.

## 2. Các đóng góp của luận án

- Xây dựng mô hình tối ưu chung và đề xuất cách tiếp cận tối ưu theo kỹ nghệ ngược trong các giai đoạn phát triển phần mềm nhúng.
- Đề xuất, phát triển phương pháp lập lịch các lệnh hợp ngữ theo thuật toán di truyền để tối ưu hiệu năng và điện năng tiêu thụ cho các kiến trúc CPU khác nhau.
- Đề xuất, phát triển phương pháp mới tối ưu điện năng tiêu thụ kết hợp cả phần cứng và phần mềm hệ thống nhúng dựa trên kỹ nghệ ngược và tái cấu hình CPU.
- Xây dựng các độ đo, hàm đánh giá hiệu năng, bộ nhớ và đề xuất phương pháp tối ưu hiệu năng và phương pháp tối ưu đa mục tiêu. Đề xuất, phát triển phương pháp tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp Tô-pô.
- Cải tiến phương pháp tối ưu hiệu năng, bộ nhớ dựa trên chuyển đổi mô hình của Anne, K. với đề xuất dùng DSL, T4.
- Cải tiến phương pháp loại bỏ các biểu thức con chung để tối ưu hiệu năng trong GCC dựa trên thay thế các biểu thức tương đương.

Các kết quả của luận án và nội dung nghiên cứu liên quan đã được công bố trong 3 bài báo quốc tế trong đó có 1 bài ở tạp chí SCI, 2 bài báo ở tạp chí trong nước, và 10 bài báo trong các Kỷ yếu hội nghị quốc tế được xuất bản bởi IEEE, IEICE, SPIE.

# 3. Cấu trúc tổng thể của luận án

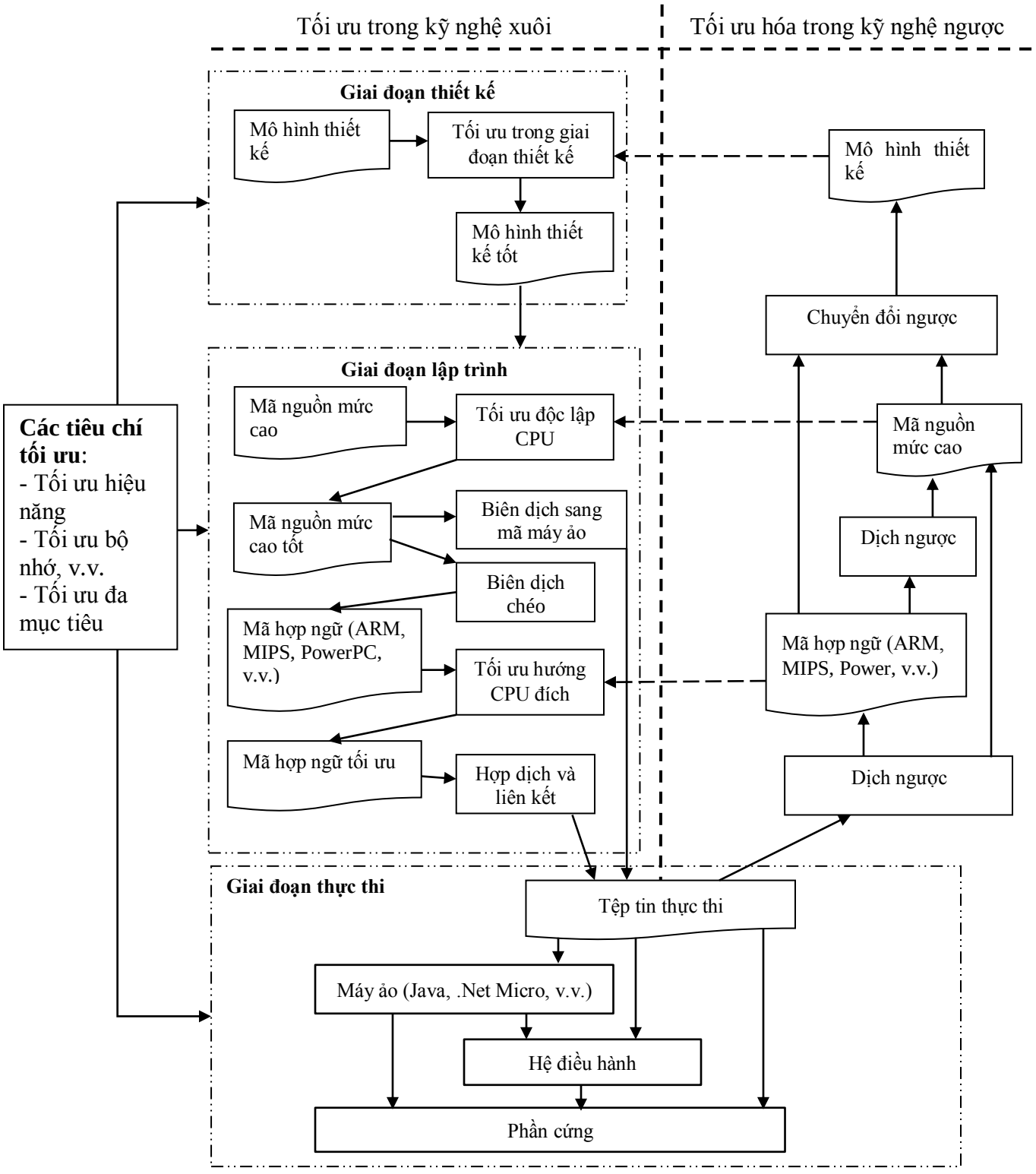


# Chương 1. TỔNG QUAN

## 1.1. Tổng quan về tối ưu phần mềm hệ thống nhúng

Trong luận án này, đầu tiên chúng tôi điều tra, phân tích các nghiên cứu liên quan để xây dựng mô hình chung cho bài toán tối ưu phần mềm nhúng như trong Hình 1.1. Bài toán tối ưu phần mềm nhúng được chia thành hai hướng tiếp cận chính là tối ưu trong kỹ nghệ xuôi và tối ưu hóa kết hợp với kỹ nghệ ngược. Hướng tiếp cận tối ưu trong kỹ nghệ xuôi, bắt đầu từ đặc tả yêu cầu, có thể thiết kế phần mềm nhúng theo các mô hình thiết kế khác nhau và dựa trên các phương pháp tối ưu trong giai đoạn thiết kế để lựa chọn các mô hình tốt. Trong giai đoạn lập trình, từ các mô hình thiết kế tốt, có thể lập trình phần mềm nhúng theo mã nguồn mức cao độc lập CPU và thực hiện các phương pháp tối ưu trên mã nguồn mức cao. Vấn đề tối ưu phần mềm nhúng trong giai đoạn thiết kế và tối ưu mã nguồn mức cao cũng tương tự như phần mềm thông thường. Mã nguồn mức cao được biên dịch chéo để tạo thành mã hợp ngữ gắn với một CPU nhúng cụ thể. Trong mức mã hợp ngữ, các phương pháp tối ưu mức thường mang tính đặc thù theo kiểu kiến trúc CPU và môi trường phần cứng cụ thể của mỗi hệ thống nhúng. Mã hợp ngữ có thể được biên dịch và liên kết để tạo ra các tệp tin thực thi. Trong giai đoạn thực thi, các phương pháp tối ưu phần mềm nhúng chủ yếu tập trung vào tối ưu môi trường thực thi, đặc tả dữ liệu và tái cấu hình CPU.

Căn cứ vào các nghiên cứu về phương pháp tối ưu trong kỹ nghệ xuôi, chúng tôi cũng đưa ra hướng tiếp cận tối ưu hóa dựa trên kỹ nghệ ngược. Kỹ nghệ ngược là một khía cạnh quan trọng trong tái kỹ nghệ phần mềm. Đây là một xu hướng nghiên cứu mới và triển vọng trong phát triển phần mềm nói chung. Kỹ nghệ ngược có thể được thực hiện theo các mức khác nhau như từ mã thực thi dịch ngược sang mã hợp ngữ, từ mã hợp ngữ có thể dịch ngược sang mã nguồn mức cao, từ mã nguồn mức cao được chuyển ngược thành các mô hình thiết kế. Mã hợp ngữ cũng có thể được chuyển ngược thành mô hình mà không cần thông qua mã nguồn mức cao. Đầu ra tại mỗi mức trong kỹ nghệ ngược có thể được tối ưu theo mức tương ứng trong kỹ nghệ xuôi. Như vậy tối ưu hóa trong kỹ nghệ ngược là sự kết hợp giữa kỹ nghệ ngược và mức tối ưu tương ứng trong kỹ nghệ xuôi.




**Hình 1.1:** Mô hình tối ưu chung trong phát triển phần mềm nhúng

## 1.2. Hiện trạng và thách thức

Tối ưu có thể được thực hiện trong các giai đoạn phát triển phần mềm nhúng như thiết kế, lập trình, thực thi. Trong mỗi giai đoạn, chúng tôi phân tích tình hình nghiên cứu hiện tại và các thách thức đặt ra để từ đó chỉ ra các khoảng trống khoa học sẽ được thực hiện trong các chương tiếp theo của luận án. Phần này gồm các nội dung chính sau:

- Hiện trạng và thách thức trong giai đoạn thiết kế
- Hiện trạng và thách thức trong giai đoạn lập trình
- Hiện trạng và thách thức trong giai đoạn thực thi.

## 1.3. Phương pháp và nội dung nghiên cứu

Theo mô hình chung đã đưa ra trong Hình 1.1, tối ưu phần mềm nhúng là bài toán phức tạp bao gồm nhiều tiêu chí tối ưu, và có thể tiến hành trong các giai đoạn khác nhau và có hai cách tiếp cận là dựa trên kỹ nghệ xuôi và kỹ nghệ ngược. Mục tiêu nghiên cứu trong luận án nhằm xây dựng một khung nhìn tổng thể về tối ưu phần mềm nhúng theo các giai đoạn trong vòng đời phần mềm và nghiên cứu các phương pháp tối ưu một cách hệ thống từ giai đoạn thiết kế đến triển khai. Trên cơ sở đó, các nghiên cứu trong luận án sẽ góp phần làm nền tảng ban đầu để giải quyết bài toán tối ưu tổng thể một cách hệ thống theo cả kỹ nghệ xuôi và kỹ nghệ ngược. Trong mỗi giai đoạn tối ưu, chúng tôi hệ thống, phân nhóm và đánh giá phương pháp tối ưu làm cơ sở lý thuyết để đưa ra một số cải tiến các phương pháp hiện tại cũng như đề xuất và phát triển một số phương pháp tối ưu mới nhằm góp phần giải quyết bài toán tối ưu tổng thể. Theo đó, trong phạm vi luận án, chúng tôi sẽ tập trung chủ yếu vào các phương pháp tối ưu trong kỹ nghệ xuôi với các nội dung nghiên cứu cụ thể sau:

- Tổng hợp, hệ thống hóa các nghiên cứu liên quan và xây dựng mô hình chung về vấn đề tối ưu phần mềm nhúng bao gồm cả kỹ nghệ xuôi và kỹ nghệ ngược.
- Nghiên cứu, đề xuất và phát triển một số phương pháp tối ưu phần mềm nhúng trong giai đoạn thiết kế như tối ưu hiệu năng, tối ưu bộ nhớ và tối ưu đa mục tiêu theo hướng tiếp cận dựa trên đánh giá trực tiếp các mô hình. Phát triển phần mềm nhận dạng chữ Nôm trên điện thoại di động và tổng hợp các chương trình nhúng khác để thử nghiệm và đánh giá các phương pháp tối ưu.
- Nghiên cứu, cải tiến và phát triển một số phương pháp tối ưu trong giai đoạn lập trình theo hai mức: mã nguồn mức cao độc lập CPU và mã hợp ngữ hướng đến các CPU nhúng. Thực nghiệm về các mức tối ưu trong bộ công cụ biên dịch nguồn mở GCC để đánh giá các phương pháp tối ưu và xây dựng các công cụ biên dịch chéo để thử nghiệm cho các loại CPU như MIPS, ARM, PowerPC.
- Nghiên cứu các phương pháp tối ưu trong giai đoạn thực thi. Đề xuất phương pháp tối ưu điện năng tiêu thụ dựa trên tái cấu hình CPU và kỹ nghệ ngược.

# Chương 2. TỐI ƯU PHẦN MỀM NHÚNG TRONG GIAI ĐOẠN THIẾT KẾ

Các nghiên cứu về tối ưu trong giai đoạn thiết kế được chia thành ba cách tiếp cận đó là tối ưu dựa trên mô phỏng, dựa trên SPE và dựa trên đánh giá trực tiếp từ các mô hình đặc tả phần mềm. Theo cách tiếp cận dựa trên mô phỏng, từ các đặc tả phần mềm sẽ sinh mã mô phỏng và thực thi mã mô phỏng trên môi trường thật hoặc môi trường giả lập để thống kê, đánh giá nhằm lựa chọn mô hình tốt. Theo cách tiếp cận SPE, từ đặc tả kiến trúc phần mềm sẽ chuyển sang các mô hình hiệu năng sau đó đánh giá mô hình hiệu năng để chọn thiết kế tốt. Tuy nhiên cách tiếp cận SPE chỉ dùng cho lớp bài toán tối ưu hiệu năng trong giai đoạn thiết kế. Cách tiếp cận dựa trên đánh giá trực tiếp các đặc tả phần mềm là một cách tiếp cận mới, hiện tại có rất ít nghiên cứu và chỉ tập trung vào đánh giá hiệu năng phần mềm. Kết quả tối ưu phần mềm nhúng trong giai đoạn thiết kế nhằm đạt được các mô hình phần mềm tốt và các lựa chọn như môi trường, công cụ, thư viện, nền tảng được đưa ra sớm. Ngoài ra đạt được sự phân chia phần cứng – phần mềm tốt cũng là một kết quả tối ưu mức hệ thống có ý nghĩa trong giai đoạn này.

Tối ưu phần mềm nhúng trong giai đoạn thiết kế, ngoài các mục tiêu về hiệu năng, điện năng tiêu thụ, bộ nhớ, chi phí, v.v. còn các mục tiêu tối ưu mang tính đặc thù trong giai đoạn thiết kế như tính tin cậy, tính an toàn, tính tái sử dụng, tính tái cấu trúc. Đây là các mục tiêu tối ưu cụ thể. Đồng thời, các phương pháp tối ưu được chia thành ba nhóm: tối ưu đơn mục tiêu, tối ưu đa mục tiêu và tối ưu chuyển từ đa mục tiêu sang đơn mục tiêu. Trong chương này, sau khi tổng hợp nghiên cứu các phương pháp tối ưu hiện tại, chúng tôi đề xuất và phát triển một số phương pháp tối ưu đó là phương pháp tối ưu hiệu năng phần mềm nhúng dựa trên đánh giá biểu đồ lớp và dựa trên chuyển đổi mô hình, tối ưu mức chiếm dụng bộ nhớ dựa trên sắp xếp Tô-pô, dựa trên chuyển đổi mô hình và tối ưu đa mục tiêu.

## 2.1. Tối ưu hiệu năng trong giai đoạn thiết kế

### 2.1.1. Tối ưu hiệu năng dựa trên biểu đồ lớp

#### i. Ý tưởng

Ý tưởng cơ bản của phương pháp tối ưu hiệu năng phần mềm nhúng dựa trên biểu đồ lớp là phân tích, đánh giá trực tiếp các thành phần trong biểu đồ lớp và xây dựng hàm đánh giá hiệu năng để lựa chọn biểu đồ lớp tốt.

### ii. Xây dựng các độ đo và hàm đánh giá hiệu năng

- **Các tham số từ biểu đồ lớp**

**Bảng 2.1.** Các tham số sử dụng để đánh giá hiệu năng

| Tham số | Ký hiệu | Mô tả |
|---|---|---|
| Biến tĩnh | $a_s^j$ | Là thuộc tính tĩnh thứ *j* của một lớp. Được cấp phát bộ nhớ ngay khi nạp chương trình. |
| Biến đối tượng | $a_o^j$ | Là thuộc tính thứ *j* của đối tượng. Được cấp phát bộ nhớ khi đối tượng được tạo. |
| Tham số phương thức | $p_k$ | Là tham số thứ *k* của một phương thức |
| Tập các lớp | $C$ | Là tập các lớp trong một biểu đồ |
| Tập các phương thức tĩnh | $M_s^i$ | Tập các phương thức tĩnh của lớp thứ *i* |
| Tập các biến tĩnh | $V_s^i$ | Tập các thuộc tính tĩnh của lớp thứ *i* |
| Tập các phương thức đối tượng | $M_o^i$ | Tập các phương thức đối tượng trong lớp thứ *i* |
| Tập các biến đối tượng | $V_o^i$ | Tập các thuộc tính đối tượng trong lớp thứ *i* |
| Tập các tham số | $P_m^j$ | Tập các tham số của phương thức thứ *j* |
| Biến tham chiếu | $v_r^j$ | Biến tham chiếu để gọi một phương thức thứ *j* |
| Kiểu trả về | $r_e^j$ | Kiểu dữ liệu trả về từ một phương thức thứ *j* |

- **Các độ đo ảnh hưởng đến hiệu năng**

**Bảng 2.2.** Các độ đo ảnh hưởng đến hiệu năng

| Độ đo | Ký hiệu | Công thức tính | Chỉ số |
|---|---|---|---|
| Kích thước các biến tĩnh | $S_1$ | $S_1 = \sum_{i=1}^{\lvert C\rvert} \sum_{j=1}^{\lvert V_s^i\rvert} \text{size}(a_s^j)$ | (2.1) |
| Kích thước các phương thức tĩnh | $S_2$ | $S_2 = \sum_{i=1}^{\lvert C\rvert} \sum_{j=1}^{\lvert M_s^i\rvert} \text{size}(v_r^j)$ | (2.2) |
| Kích thước thực thi các phương thức tĩnh | $S_3$ | $S_3 = \sum_{i=1}^{\lvert C\rvert} \sum_{j=1}^{\lvert M_s^i\rvert} (\text{size}(r_e^j) + \sum_{k=1}^{\lvert P_m^j\rvert} \text{size}(p_k))$ | (2.3) |
| Kích thước các biến đối tượng | $S_4$ | $S_4 = \sum_{i=1}^{\lvert C\rvert} \sum_{j=1}^{\lvert V_o^i\rvert} \text{size}(a_o^j)$ | (2.4) |

| | | | |
|---|---|---|---|
| Kích thước các phương thức đối tượng | $S_5$ | $S_5 = \sum_{i=1}^{\lvert C\rvert} \sum_{j=1}^{\lvert M_o^i\rvert} \text{size}(v_r^{\ j})$ | (2.5) |
| Kích thước thực thi các phương thức đối tượng | $S_6$ | $S_6 = \sum_{i=1}^{\lvert C\rvert} \sum_{j=1}^{\lvert M_o^i\rvert} (\text{size}(r_e^{\ j}) + \sum_{k=1}^{\lvert P_m^j\rvert} \text{size}(p_k))$ | (2.6) |

- **Hàm đánh giá hiệu năng**

Sau khi tham số hóa và xây dựng các độ đo ở trên, chúng tôi xây dựng hàm đánh giá hiệu năng $f_p$ theo các độ đo như trong công thức (2.7). Trong công thức (2.7), $\alpha$, $\beta$, $\gamma$ và $\varepsilon$ là các hệ số phụ thuộc của hiệu năng vào độ đo tương ứng và $S_1$ đến $S_6$ được tính theo các công thức từ (2.1) đến (2.6). Các hệ số phụ thuộc được xác định trên cơ sở phân tích quá trình thực thi chương trình hướng đối tượng đã trình bày chi tiết trong luận án.

$$f_p = \alpha \times (S_1 + S_2) + \beta \times S_3 + \gamma \times (S_4 + S_5) + \varepsilon \times S_6 \quad (2.7)$$

### iii. Thực nghiệm

**Bảng 2.3.** Tổng hợp tham số, độ đo và hàm đánh giá hiệu năng chương trình *Netduino_8digit*

| **Biểu đồ** | **Số lớp** | **Số phương thức tĩnh** | **Số thuộc tính tĩnh** | **Số phương thức động** | **Số thuộc tính động** | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $f_p$ (Số thao tác bộ nhớ) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 3 | 0 | 15 | 9 | 12 | 0 | 12 | 60 | 24 | 124 | 1192 |
| 2 | 4 | 3 | 0 | 17 | 9 | 12 | 0 | 12 | 68 | 24 | 137 | 1307 |
| *3* | *2* | *10* | *5* | *7* | *4* | *40* | *8* | *84* | *28* | *16* | *45* | *879* |
| 4 | 4 | 3 | 5 | 15 | 4 | 12 | 17 | 12 | 60 | 7 | 121 | 1154 |
| 5 | 5 | 9 | 0 | 11 | 9 | 36 | 0 | 69 | 44 | 24 | 80 | 1112 |

## 2.1.2. Tối ưu hiệu năng dựa trên chuyển đổi mô hình

### i. Ý tưởng

Ý tưởng của phương pháp tối ưu hiệu năng phần mềm nhúng trong giai đoạn thiết kế dựa trên biến đổi mô hình là dựa vào các phép chuyển đổi trên mô hình để đưa mô hình thiết kế ban đầu về mô hình tối ưu.

### ii. Các phép biến đổi trên mô hình

Để tối ưu hiệu năng chúng tôi đưa ra ba phép biến đổi là thu gọn kiểu dữ liệu, chuyển đổi tham số của các phương thức thành các thành viên dữ liệu của lớp và chuyển đổi từ tĩnh sang động và ngược lại.

### iii. Thực nghiệm

**Bảng 2.4.** Tổng hợp kết quả tối ưu và thử nghiệm thực tế

| Chương trình thử nghiệm | Mô hình ban đầu | | Mô hình tối ưu | |
|---|---|---|---|---|
| | $f_p$ (Số thao tác bộ nhớ) | Thời gian thực thi thực tế (μs) | $f_p$ (Số thao tác bộ nhớ) | Thời gian thực thi thực tế (μs) |
| Netduino_8digit | 959 | 4985508 | 730 | 4957343 |
| Netduino_LCD | 1113 | 8345647 | 985 | 8143654 |
| Netduino_SerialPort | 600 | 937369 | 485 | 837569 |

## 2.2. Tối ưu bộ nhớ trong giai đoạn thiết kế

### 2.2.1. Tối ưu bộ nhớ dựa trên sắp xếp Tô-pô

#### i. Ý tưởng

Ý tưởng cơ bản của phương pháp này như sau: phần mềm nhúng được thực thi theo một tập các tác vụ thỏa mãn một đồ thị phụ thuộc; các tác vụ này có thể thực hiện theo các thứ tự khác nhau mà không làm thay đổi kết quả; mỗi thứ tự thực thi chính là một sắp xếp Tô-pô trên đồ thị phụ thuộc; đánh giá và lựa chọn sắp xếp Tô-pô có dung lượng bộ nhớ chiếm dụng ít nhất.

#### ii. Đồ thị phụ thuộc và sắp xếp Tô-pô

Phần mềm nhúng có thể được đặc tả bằng một đồ thị tác vụ phụ thuộc. Mỗi tác vụ được định nghĩa như một hàm với tên, kiểu trả về và danh sách tham số. Các tác vụ có thể độc lập hoặc phụ thuộc lẫn nhau. Theo đó, chương trình có thể biểu diễn bằng đồ thị phụ thuộc $G$ và được định nghĩa theo công thức (2.9).

$$G = <U, V> \text{ với } U = \{u_i \mid i = 1..N\} \quad \text{và} \quad V \subseteq \{v_{ij} = (u_i, u_j);\ i, j = 1..N\} \qquad (2.9)$$

Trong đó:

- Mỗi đỉnh $u_i$ tương ứng với một tác vụ $i$
- Mỗi cạnh $v_{ij}$ cho biết tác vụ $j$ phụ thuộc vào tác vụ $i$ và chỉ được thực hiện khi tác vụ $i$ đã kết thúc.

Từ đồ thị phụ thuộc, có thể tìm được nhiều cách thực thi chương trình, mỗi chuỗi Tô-pô biểu diễn một cách thực thi. Phương pháp tối ưu này dựa vào giá trị của giá trị của hàm đánh giá để tìm chuỗi Tô-pô của chương trình có mức chiếm dụng bộ nhớ ít nhất.

### iii. Xây dựng hàm đánh giá bộ nhớ trên chuỗi Tô-pô

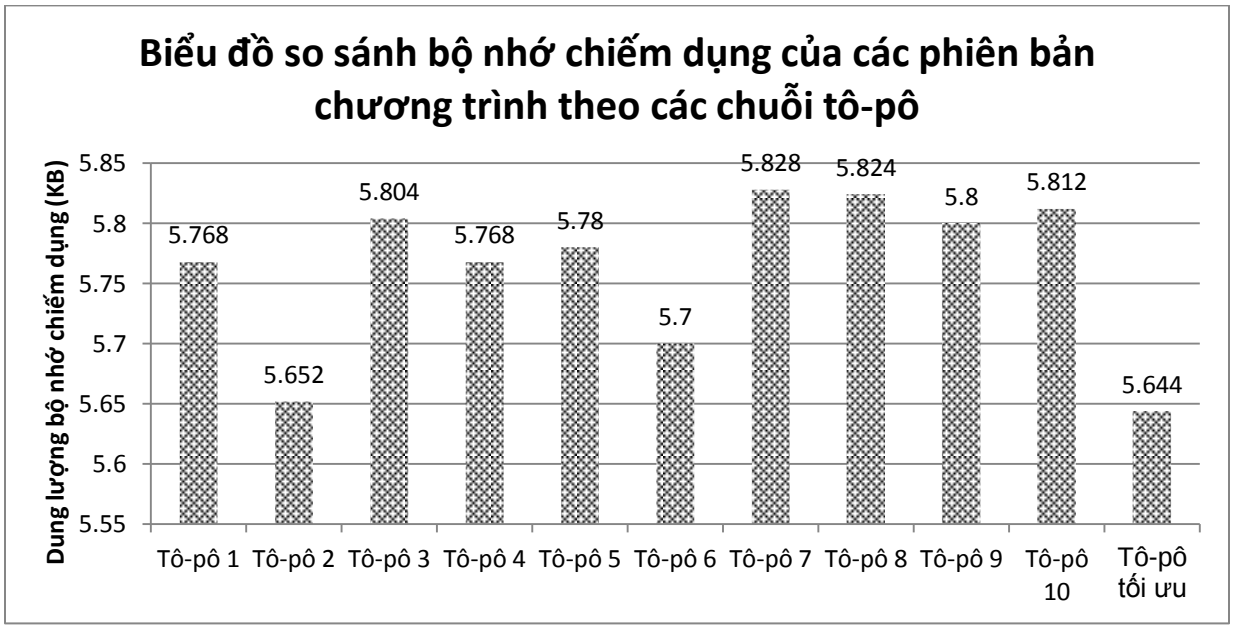
Để đánh giá chuỗi Tô-pô có sử dụng bộ nhớ hiệu quả nhất, trong phần này chúng tôi xây dựng hàm đánh giá dung lượng bộ nhớ chiếm dụng của chương trình trên mỗi chuỗi Tô-pô. Dựa trên việc phân tích quá trình cấp phát bộ nhớ khi thực thi chương trình và mức chiếm dụng bộ nhớ gây ra bởi mỗi tác vụ, chúng tôi xây dựng hàm đánh giá kích thước bộ nhớ chiếm dụng trên chuỗi Tô-pô như công thức (2.10):

$$f_m = \sum_{i=1}^{N} (N - i) \times \text{size}(r_i) \tag{2.10}$$

Trong đó:

- $f_m$ là hàm đánh giá mức độ chiếm dụng bộ nhớ của chương trình
- $N$ là số tác vụ
- $r_i$ là kiểu dữ liệu trả về của tác vụ thứ $i$.

### iv. Thực nghiệm




**Hình 2.1:** Mức chiếm dụng bộ nhớ thực tế

### 2.2.2. Tối ưu bộ nhớ dựa trên biến đổi mô hình

#### i. Ý tưởng

Ý tưởng chính của phương pháp này là dựa trên hàm đánh giá và các phép chuyển đổi mô hình để đưa mô hình thiết kế ban đầu về mô hình có dung lượng bộ nhớ chiếm dụng tối ưu.

#### ii. Xây dựng các phép chuyển đổi mô hình để tối ưu bộ nhớ

Trong phần này, chúng tôi chứng minh tính đúng đắn của hai phép biến đổi phân chia cấu trúc và gộp cấu trúc của Anne, K. để sử dụng trong chương trình tối ưu. Ngoài ra, chúng tôi cũng đưa ra phép biến đổi rút gọn kiểu dữ liệu như đã trình bày trong phần tối ưu hiệu năng và phép chuyển các thành phần tĩnh thành động để áp dụng cho tối ưu bộ nhớ.

#### iii. Thực nghiệm

**Bảng 2.5.** Tổng hợp kết quả tối ưu và thử nghiệm thực tế

| **Chương trình thử nghiệm** | **Mô hình ban đầu** | | **Mô hình tối ưu** | |
|---|---|---|---|---|
| | $f_m$ (byte) | Bộ nhớ chiếm dụng thực tế (KB) | $f_m$ (byte) | Bộ nhớ chiếm dụng thực tế (KB) |
| Mô-đun nhận dạng chữ Nôm | 783 | 65 | 645 | 64 |
| Tháp Hà Nội | 562 | 15 | 473 | 14 |
| 8 quân Hậu | 400 | 26 | 315 | 24 |

## 2.3. Tối ưu đa mục tiêu dựa trên biểu đồ lớp

#### i. Ý tưởng

Phương pháp tối ưu này nhằm tìm ra mô hình dữ liệu cân bằng nhất giữa hiệu năng và dung lượng bộ nhớ chiếm dụng của phần mềm nhúng. Ý tưởng cơ bản của phương pháp này là phân tích các tham số trực tiếp từ mô hình để xây dựng các hàm mục tiêu và dựa trên nguyên lý Pareto để tìm mô hình dữ liệu có phân phối cân bằng nhất giữa hiệu năng và bộ nhớ chiếm dụng. Mô hình dữ liệu áp dụng trong nghiên cứu này là biểu đồ lớp rút gọn.

### ii. Xây dựng các hàm mục tiêu

Kế thừa các tham số và độ đo trong phần tối ưu hiệu năng dựa trên biểu đồ lớp, chúng tôi thực hiện một số sửa đổi để xây dựng các hàm mục tiêu tối ưu. Toàn bộ các tham số trong Bảng 2.1 và các độ đo $S_1$, $S_2$, $S_4$ và $S_5$ trong Bảng 2.2 theo các công thức (2.1), (2.2), (2.4) và (2.5) được sử dụng lại. Các công thức tính $S_3$, $S_6$ được chúng tôi định nghĩa lại để phù hợp với bài toán tối ưu đa mục tiêu như sau:

$$S_3 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_S^i|} \sum_{k=1}^{|P_m^j|} \text{size}(p_k) \tag{2.24}$$

$$S_6 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_O^i|} \sum_{k=1}^{|P_m^j|} \text{size}(p_k) \tag{2.25}$$

- **Các hàm mục tiêu thành phần**

**Bảng 2.6.** Các hàm mục tiêu thành phần

| Hàm mục tiêu thành phần | Công thức tính | Chỉ số |
|---|---|---|
| Hàm mục tiêu hiệu năng | $f_1 = \frac{S_1 + S_2}{S_4 + S_5} + \frac{S_1}{S_3} + \frac{S_4}{S_6}$ | (2.26) |
| Hàm mục tiêu bộ nhớ | $f_2 = \frac{S_4 + S_5}{S_1 + S_2} + \frac{S_3}{S_1} + \frac{S_6}{S_4}$ | (2.27) |

- **Hàm mục tiêu toàn cục**

$$f = w_1 \times f_1 + w_2 \times f_2 \tag{2.28}$$

Trong đó: $w_1$, $w_2$ là trọng số của các hàm mục tiêu và $w_1 + w_2 = 1$.

### iii. Thực nghiệm

**Bảng 2.7.** Tổng hợp tham số, độ đo và các hàm mục tiêu của chương trình *Netduino_8digit*

| Biểu đồ | Số lớp | Số phương thức tĩnh | Số thuộc tính tĩnh | Số phương thức động | Số thuộc tính động | $S_1$ | $S_2$ | $S_3$ | $S_4$ | $S_5$ | $S_6$ | $f_1$ | $f_2$ | $f$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 3 | 3 | 0 | 15 | 9 | 12 | 0 | 12 | 60 | 24 | 124 | 0,34 | 30 | 9,238 |
| 2 | 4 | 3 | 0 | 17 | 9 | 12 | 0 | 12 | 68 | 24 | 137 | 0,31 | 30 | 9,217 |
| 3 | 2 | 10 | 5 | 7 | 4 | 40 | 8 | 84 | 28 | 16 | 45 | 1,54 | 14,23 | 5,347 |
| 4 | 4 | 3 | 5 | 15 | 4 | 12 | 17 | 12 | 60 | 7 | 121 | 1,91 | 20,3 | 7,426 |
| *5* | *5* | *9* | *0* | *11* | *9* | *36* | *0* | *69* | *44* | *24* | *80* | *0,83* | *30* | *9,581* |

# Chương 3. TỐI ƯU PHẦN MỀM NHÚNG TRONG GIAI ĐOẠN LẬP TRÌNH

Tối ưu mã nguồn phần mềm nhúng gồm hai giai đoạn là tối ưu mã nguồn độc lập kiến trúc CPU và tối ưu mã nguồn hướng đến một kiến trúc CPU cụ thể. Vấn đề tối ưu mã nguồn phần mềm đã được nghiên cứu từ lâu và hầu hết các trình biên dịch đều hỗ trợ các lựa chọn tối ưu. Tuy nhiên, các nghiên cứu về tối ưu mã nguồn hướng đến kiến trúc CPU đích trong các chương trình biên dịch chéo còn ít phổ biến hơn. Để tiếp cận có hệ thống, trong chương này, chúng tôi nghiên cứu theo hai mức tối ưu là tối ưu mã nguồn độc lập và phụ thuộc vào kiến trúc CPU đích. Trong mỗi mức tối ưu chúng tôi cũng tóm lược lại các phương pháp, kỹ thuật tối ưu hiện tại sau đó cải tiến hoặc đề xuất triển, khai phương pháp mới.

## 3.1. Tối ưu mã nguồn mức cao độc lập máy đích

### 3.1.1. Cải tiến tối ưu cục bộ dựa trên thay thế biểu thức tương đương

#### i. Ý tưởng

Ý tưởng chính của cải tiến này là phân tích mã nguồn, tìm và thay thế các biểu thức tương đương dựa trên các tính chất toán học. Theo đó mọi biểu thức tương đương trong chương trình được thay thế bằng một biểu thức và chỉ phải tính giá trị một lần nên rút ngắn được thời gian tính toán.

#### ii. Xác định và thay thế biểu thức tương đương

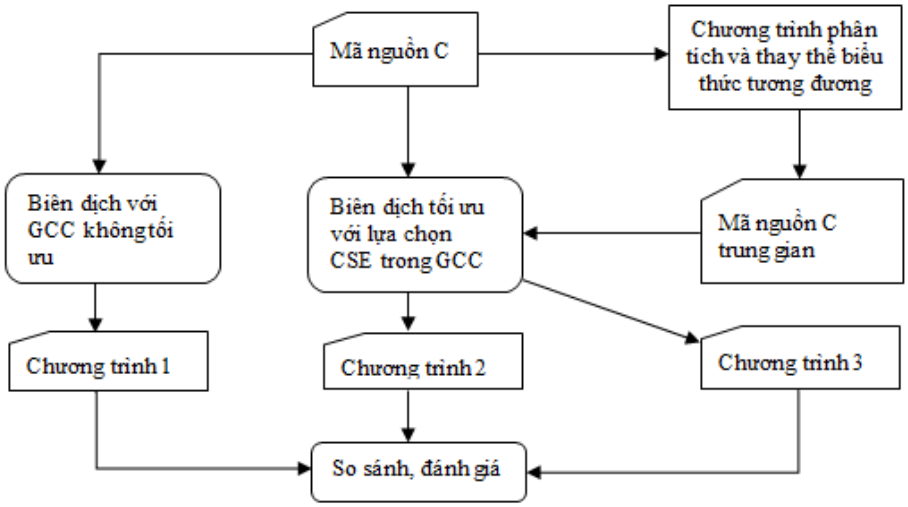
- **Xác định biểu thức tương đương:** Để chứng minh hai biểu thức tương đương một cách tự động, đầu tiên phân tích và xây dựng cây biểu thức để chứng minh hai biểu thức tương đương.
- **Thay thế biểu thức tương đương:** Các biểu thức tương đương nhau được thay bằng một biểu thức đại diện trong các khối cơ bản.

#### iii. Kết hợp thay thế biểu thức tương đương với tối ưu cục bộ trong GCC

Sau khi đã thay thế các biểu thức tương đương bằng một biểu thức đại diện, chương trình được biên dịch bằng GCC với lựa chọn loại bỏ các biểu thức con chung để tối ưu.

### iv. Thực nghiệm

Để đánh giá phương pháp này, chúng tôi tiến hành thực nghiệm theo mô hình trong Hình 3.1. Kết quả thực nghiệm được tổng hợp trong Bảng 3.1.




**Hình 3.1:** Mô hình thực nghiệm phương pháp thay thế biểu thức tương đương

**Bảng 3.1.** Tổng hợp thời gian thực thi

| Lần thực thi | sumNnumber | | | Fibonacci | | | sumPrimes | | |
|---|---|---|---|---|---|---|---|---|---|
| | P1 | P2 | P3 | P1 | P2 | P3 | P1 | P2 | P3 |
| 1 | 0,333 | 0,323 | 0,282 | 0,503 | 0,501 | 0,491 | 0,153 | 0,151 | 0,139 |
| 2 | 0,336 | 0,328 | 0,283 | 0,511 | 0,502 | 0,502 | 0,149 | 0,145 | 0,142 |
| 3 | 0,342 | 0,341 | 0,273 | 0,495 | 0,501 | 0,487 | 0,155 | 0,153 | 0,140 |
| 4 | 0,343 | 0,339 | 0,275 | 0,501 | 0,489 | 0,495 | 0,151 | 0,148 | 0,138 |
| 5 | 0,343 | 0,337 | 0,278 | 0,508 | 0,504 | 0,488 | 0,147 | 0,141 | 0,136 |
| 6 | 0,347 | 0,332 | 0,267 | 0,513 | 0,507 | 0,493 | 0,150 | 0,145 | 0,143 |
| 7 | 0,339 | 0,350 | 0,265 | 0,506 | 0,505 | 0,501 | 0,148 | 0,147 | 0,139 |
| 8 | 0,338 | 0,336 | 0,271 | 0,508 | 0,503 | 0,485 | 0,145 | 0,143 | 0,137 |
| 9 | 0,347 | 0,344 | 0,264 | 0,503 | 0,501 | 0,498 | 0,151 | 0,149 | 0,140 |
| 10 | 0,345 | 0,343 | 0,275 | 0,511 | 0,506 | 0,495 | 0,149 | 0,141 | 0,137 |
| **Thời gian trung bình (s)** | **0,3413** | **0,3373** | **0,2733** | **0,5059** | **0,5019** | **0,4935** | **0,1498** | **0,1463** | **0,1391** |

## 3.1.2. Cải tiến hiệu năng phần mềm nhúng dựa trên nén dữ liệu

Ngoài phương pháp cải tiến dựa trên thay thế biểu thức tương đương, trong phần này chúng tôi cũng phát triển phương pháp tối ưu hiệu năng dựa trên nén dữ liệu. Ý tưởng chính của phương pháp này là cải tiến thời gian truyền thông tin trên đường truyền dựa trên nén dữ liệu.

## 3.2. Tối ưu mã hợp ngữ hướng đến các CPU hệ thống nhúng

### 3.2.1. Tối ưu hiệu năng dựa trên lập lịch các lệnh

**i. Ý tưởng**

Ý tưởng chính của phương pháp này là tìm một thứ tự thực hiện lệnh sau cho tổng kích thước các đoạn trễ nhỏ nhất đối với kiến trúc đường ống lệnh và đạt mức song song hóa cao nhất đối với kiến trúc siêu vô hướng.

**ii. Xây dựng hàm đánh giá hiệu năng**

**Hàm đánh giá hiệu năng cho kiến trúc đường ống lệnh**

Giả sử CPU có kiến trúc $N_s$-đoạn, thời gian mỗi đoạn là $T_s$ và kích thước hàng đợi lệnh lấy về là $N_s$. Trong trường hợp kiến trúc CPU tuần tự, thời gian hoàn thành một câu lệnh là $N_s \times T_s$. Trong trường hợp CPU có kiến trúc đường ống lệnh và các lệnh độc lập dữ liệu, thời gian để thực hiện xong $N_I$ câu lệnh là $(N_s + N_I - 1) \times T_s$. Do đó, thời gian thực hiện một câu lệnh được tính theo công thức (3.5). Tuy nhiên, trong trường hợp một lệnh phụ thuộc dữ liệu vào câu lệnh đứng trước nó thì câu lệnh sẽ kết thúc sau lệnh đứng ngay trước một khoảng thời gian lớn hơn $T_s$. Thời gian trễ (*stall*) tùy thuộc vào kiểu phụ thuộc dữ liệu giữa các câu lệnh. Chúng tôi xây dựng hàm đánh giá hiệu năng chính là hàm tính tổng thời gian trễ như trong công thức (3.6).

$$t_a = \frac{(N_s + N_I - 1) \times T_s}{N_I} \tag{3.5}$$

$$f_p = \sum_{i=1}^{N_I} s_i \tag{3.6}$$

Trong đó:

- $f_p$: Hàm đánh giá hiệu năng được tính bằng tổng độ trễ
- $N_I$: Số câu lệnh
- $s_i$: Khoảng thời gian trễ của câu lệnh $i$

Trong kiến trúc đường ống $N_s$-đoạn, để thực hiện lệnh $i$ cần xét sự phụ thuộc dữ liệu của lệnh $i$ với $N_s - 1$ lệnh đứng trước. Độ trễ được tính bằng thời gian lớn nhất mà lệnh $i$ phải đợi một trong $N_s - 1$ lệnh đứng trước. Hơn nữa, theo kiến trúc đường ống lệnh, nếu hai lệnh độc lập dữ liệu thì thời gian trễ của lệnh sau là $T_s$. Do đó, chúng tôi xây dựng công thức (3.7) để tính độ trễ của lệnh $i$.

$$s_i = \max_{0 \le j < N_s - 1} (n_d - j) \times T_s \tag{3.7}$$

Trong đó: $n_d$ là số đoạn mà câu lệnh $i$ phải chờ câu lệnh thứ $i - 1 - j$.

**Hàm đánh giá hiệu năng cho kiến trúc siêu vô hướng**

Ý tưởng của kiến trúc siêu vô hướng là nhiều câu lệnh có thể được thực hiện song song trong cùng một giai đoạn. Điều này đòi hỏi có thêm các đơn vị chức năng trong CPU. Hàm đánh giá hiệu năng cho cả hai kiểu *in-order* và *out-of-order* được xây dựng như trong công thức (3.8).

$$\begin{aligned} f_p^k &= k - 1 + x_k \\ x_k &= \begin{cases} 1 \; if \; S_I^k \neq \emptyset \\ 0 \; if \; S_I^k = \emptyset \end{cases} \\ S_I^k &= S_I^{k-1} \setminus E_I^{k-1} \cup S_N^k \end{aligned} \tag{3.8}$$

Trong đó:

- $S_I^k$ là nhóm lệnh đã giải mã trong cửa sổ lệnh tại bước $k$; tổng số lệnh trong nhóm bằng độ rộng phát lệnh.
- $E_I^{k-1}$ là tập các lệnh đã được thực thi tại bước $k - 1$
- $S_N^k$ là các lệnh tiếp theo trong chuỗi lệnh ban đầu; các lệnh này sẽ được chuyển từ chuỗi ban đầu vào cửa sổ lệnh tại bước $k$.

### iii. Áp dụng thuật toán di truyền để lập lịch và xây dựng chương trình tối ưu

Sau khi xây dựng hàm đánh giá hiệu năng trên một chuỗi Tô-pô lệnh, chúng tôi áp dụng thuật toán di truyền (GA) để lựa tìm chuỗi Tô-pô có hiệu năng tốt nhất. Mỗi nhiễm sắc thể là một chuỗi Tô-pô với vị trị các gen chính là thứ tự thực hiện lệnh và mỗi gen có giá trị là thứ tự câu lệnh trong chương trình ban đầu. Hàm đánh giá hiệu năng được sử dụng là hàm thích nghi.

### iv. Đánh giá phương pháp

**Bảng 3.2.** Kết quả tối ưu hiệu năng dựa trên lập lịch cho kiến trúc đường ống lệnh

| STT | Chương trình | Thời gian thực thi (cycle) | | Thời gian tiết kiệm (%) |
|---|---|---|---|---|
| | | Không lập lịch | Lập lịch | |
| 1 | Fibonacci | 565343 | 565283 | 0,01 |
| 2 | Sum N numbers | 3763356 | 3763104 | 0,01 |
| 3 | ArraySum | 25611 | 25550 | 0,24 |
| 4 | Quick Sort | 89214 | 89109 | 0,12 |
| 5 | Bubble Sort | 380851 | 380734 | 0,03 |
| 6 | Binary Search | 17557 | 17546 | 0,06 |
| 7 | Hanoi | 263279 | 263162 | 0,04 |
| 8 | Permutation | 901932 | 901844 | 0,01 |
| 9 | Matrix Multiply | 21159 | 21047 | 0,53 |
| **Trung bình** | | | | **0,12** |

**Bảng 3.3.** Kết quả tối ưu hiệu năng dựa trên lập lịch cho kiến trúc siêu vô hướng *in-order*

| STT | Chương trình | Thời gian thực thi (cycle) | | Thời gian tiết kiệm (%) |
|---|---|---|---|---|
| | | Không lập lịch | Lập lịch | |
| 1 | Fibonacci | 476859 | 476799 | 0,01 |
| 2 | Sum N Numbers | 3657599 | 3654666 | 0,08 |
| 3 | ArraySum | 23336 | 23264 | 0,31 |
| 4 | Quick Sort | 84540 | 83804 | 0,87 |
| 5 | Bubble Sort | 369612 | 362247 | 1,99 |
| 6 | Binary Search | 16026 | 15967 | 0,37 |
| 7 | Hanoi | 247401 | 240475 | 2,80 |
| 8 | Permutation | 828657 | 820090 | 1,03 |
| 9 | Matrix Multiply | 19370 | 19229 | 0,73 |
| **Trung bình** | | | | **0,91** |

**Bảng 3.4.** Kết quả tối ưu hiệu năng dựa trên lập lịch cho kiến trúc siêu vô hướng *out-of-order*

| STT | Chương trình | Thời gian thực thi (cycle) | | Thời gian tiết kiệm (%) |
|---|---|---|---|---|
| | | Không lập lịch | Lập lịch | |
| 1 | Fibonacci | 341382 | 341239 | 0,04 |
| 2 | Sum N Number | 1842041 | 1499085 | 18,62 |
| 3 | ArraySum | 17936 | 17884 | 0,29 |
| 4 | Quick Sort | 49945 | 47868 | 4,16 |
| 5 | Bubble Sort | 199455 | 189497 | 4,99 |
| 6 | Binary search | 11386 | 11334 | 0,46 |
| 7 | Hanoi | 130955 | 125535 | 4,14 |
| 8 | Permutation | 456412 | 454883 | 0,34 |
| 9 | Matrix Multiply | 13210 | 13092 | 0,89 |
| **Trung bình** | | | | **2,50** |

### 3.2.3. Tối ưu điện năng tiêu thụ dựa trên lập lịch các lệnh

#### i. Ý tưởng

Ý tưởng chính của phương pháp này là dựa trên bảng tiêu thụ điện năng của các lệnh để xây dựng hàm đánh giá điện năng trên mỗi chuỗi Tô-pô và áp dụng thuật toán di truyền để tìm chuỗi Tô-pô tốt nhất.

#### ii. Xây dựng bảng tiêu thụ điện năng

Điện năng tiêu thụ của chương trình bao gồm điện năng tiêu thụ của các lệnh và điện năng hao phí khi chuyển một lệnh sang lệnh khác. Điện năng tiêu tốn khi thực hiện một lệnh không phụ thuộc và thứ tự thực hiện. Theo đó sự khác biệt về điện năng tiêu thụ của chương trình theo các thứ tự thực hiện lệnh khác nhau do điện năng hao phí khi chuyển lệnh gây ra. Chúng tôi xây dựng bảng tiêu thụ điện năng là ma trận vuông, mỗi phần tử biểu diễn năng lượng hao phí khi chuyển từ lệnh *i* sang lệnh *j*. Để đo điện năng hao phí khi chuyển lệnh, chúng tôi sử dụng công cụ *SimplePower* cho tập lệnh MIPS.

#### iii. Áp dụng thuật toán di truyền để lập lịch

Thuật toán di truyền được sử dụng để lập lịch, tìm chuỗi Tô-pô ít hao phí điện năng nhất. Hàm thích nghi được lập trình dựa vào bảng tiêu thụ điện năng. Căn cứ vào thứ tự thực hiện lệnh trong mỗi chuỗi Tô-pô và điện năng hao phí khi chuyển lệnh trong bảng tiêu thụ điện năng sẽ tính được tổng điện năng hao phí.

#### iv. Đánh giá phương pháp

Điện năng xấp xỉ bằng $V^2 C_c f_x \times t = V^2 C_c f_x \times \frac{1}{f_x} = V^2 C_c$ với $t$ là chu kỳ đồng hồ, $V$ là điện áp và $f_x$ là tần số đồng hồ. $V$ không thay đổi nên điện năng tiêu thụ tỉ lệ thuận với $C_c$.

**Bảng 3.5.** Đánh giá điện năng tiêu thụ thông qua $C_c$ khi lập lịch theo GA

| STT | Chương trình | Điện dung chuyển mạch $C_c$ (pF) | | Điện năng tiết kiệm (%) |
|---|---|---|---|---|
| | | Không lập lịch | Lập lịch | |
| 1 | Quick Sort | 2077771,2516 | 1903200,8108 | 8,40 |
| 2 | Bubble Sort | 12365628,8339 | 11235754,7681 | 9,13 |
| 3 | Binary Search | 11500,0162 | 11179,6200 | 2,78 |
| 4 | Hanoi | 6341659,6373 | 6063484,9539 | 4,39 |
| 5 | Heap Sort | 3598279,6097 | 3427785,7324 | 4,74 |
| 6 | Permutation | 24001185,1045 | 23654096,0146 | 1,44 |
| 7 | Matrix Multiply | 110309,0015 | 103655,2787 | 6,03 |
| **Trung bình** | | | | **9,89** |

# Chương 4. TỐI ƯU PHẦN MỀM NHÚNG TRONG GIAI ĐOẠN THỰC THI

Tối ưu trong giai đoạn thực thi là giai đoạn tối ưu cuối cùng, khi chương trình được thực thi trong các môi trường và thiết bị cụ thể. Cũng như tối ưu trong các giai đoạn khác, trong giai đoạn thực thi, có thể thực hiện tối ưu đơn mục tiêu theo các tiêu chí như: hiệu năng, bộ nhớ, năng lượng, v.v. hay tối ưu đa mục tiêu. Tối ưu trong giai đoạn này gặp thách thức khi can thiệp, sửa đổi mã thực thi của chương trình cũng như việc tái cấu trúc, dịch ngược và tái biên dịch rất phức tạp. Việc tối ưu trong giai đoạn này phụ thuộc chính vào môi trường thực thi và dữ liệu. Theo đó, các kỹ thuật tối ưu chương trình trong giai đoạn thực thi được phân thành ba nhóm chính đó là tối ưu môi trường thực thi, tối ưu dữ liệu và tối ưu chương trình thực thi. Trong chương này, trên cơ sở phân tích, tổng hợp các nghiên cứu liên quan, chúng tôi đề xuất và triển khai phương pháp tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU.

## 4.1. Tối ưu môi trường thực thi

### 4.1.1. Các kỹ thuật tối ưu môi trường thực thi tiêu biểu

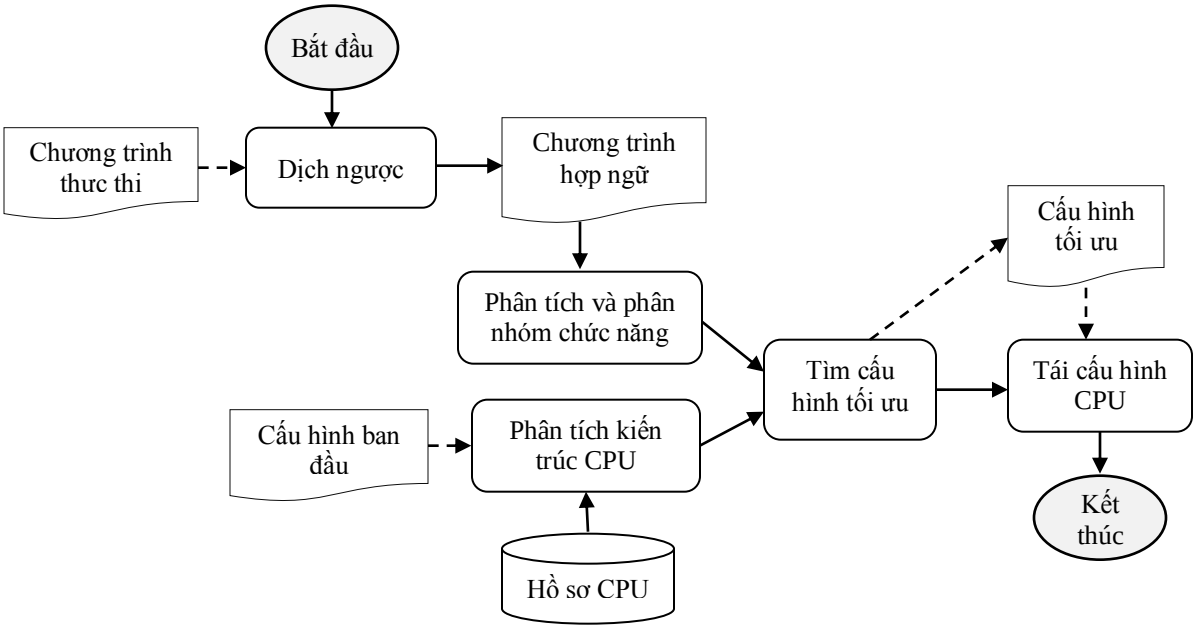
- Kỹ thuật biên dịch tạm
- Phương pháp tối ưu dựa trên lập lịch tiến trình
- Tối ưu dựa trong thời gian thực thi dựa trên chuyên biệt hóa

### 4.1.2. Tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU

**i. Ý tưởng**

Phương pháp tối ưu này dựa trên sự kết hợp phần cứng, phần mềm trong hệ thống nhúng. Ý tưởng của phương pháp như sau: Dịch ngược mã thực thi sang mã hợp ngữ; phân tích mã hợp ngữ để tìm các đơn vị chức năng được sử dụng trong chương trình; cấu hình CPU để tắt các đơn vị chức năng không được sử dụng.

### ii. Quy trình tối ưu




**Hình 4.1:** Quy trình nghiên cứu và thực nghiệm tối ưu năng lượng dựa trên cấu hình CPU

### iii. Thực nghiệm

**Bảng 4.1.** Tổng hợp kết quả tối ưu điện năng tiêu thụ dựa trên tái cấu hình CPU

| STT | Chương trình | Năng lượng tiêu thụ với cấu hình tối ưu (W) | Năng lượng tiết kiệm | |
|---|---|---|---|---|
| | | | (W) | (%) |
| 1 | Fibonacci | 91,234 | 33,265 | 26,72 |
| 2 | Sum N numbers | 102,222 | 22,277 | 17,89 |
| 3 | ArraySum | 96,387 | 28,112 | 22,58 |
| 4 | Quick Sort | 96,387 | 28,112 | 22,58 |
| 5 | Bubble Sort | 106,693 | 17,806 | 14,30 |
| 6 | Binary Search | 96,387 | 28,112 | 22,58 |
| 7 | Hanoi | 106,693 | 17,806 | 14,30 |
| 8 | Heap | 106,693 | 17,806 | 14,30 |
| 9 | Permutation | 106,693 | 17,806 | 14,30 |
| 10 | Matrix Multiply | 114,699 | 9,8 | 7,87 |
| **Điện năng tiết kiệm trung bình** | | | | **17,74** |

## 4.2. Các cách tiếp cận tối ưu khác trong giai đoạn thực thi

- Tối ưu dựa trên cải tiến môi trường truyền dữ liệu
- Tối ưu chương trình thực thi dựa trên mã tự sửa

# KẾT LUẬN

Trong phần này, chúng tôi tổng hợp các kết quả nghiên cứu, các đóng góp chính của luận án và các vấn đề chưa giải quyết cũng như định hướng các nghiên cứu tiếp theo. Đầu tiên, luận án cung cấp một mô hình chung về vấn đề tối ưu trong phát triển phần mềm nhúng. Chúng tôi đã tổng hợp, phân nhóm, phân tích và đánh giá các nghiên cứu liên quan để đưa ra mô hình chung cho bài toán tối ưu phần mềm nhúng. Dựa trên mô hình chung chúng ta có thể tiếp cận, nghiên cứu vấn đề tối ưu phần mềm nhúng một cách đầy đủ và hệ thống. Các kết quả và đóng góp chính của luận án được tóm lược dưới đây.

*Về vấn đề tối ưu phần mềm nhúng trong giai đoạn thiết kế*, chúng tôi đã tiếp cận nghiên cứu theo ba tiêu chí là tối ưu hiệu năng, tối ưu bộ nhớ và tối ưu đa mục tiêu. Trong tiêu chí tối ưu hiệu năng, chúng tôi đã đề xuất và phát triển hai phương pháp tối ưu là tối ưu hiệu năng dựa trên biểu đồ lớp và tối ưu hiệu năng dựa trên chuyển đổi mô hình. Các kết quả nghiên cứu trong tiêu chí tối ưu này đã được xuất bản trong [CT6, CT9]. Trong tiêu chí tối ưu bộ nhớ, chúng tôi đã đề xuất và phát triển hai phương pháp tối ưu mới đó là tối ưu dung lượng bộ nhớ chiếm dụng dựa trên sắp xếp Tô-pô và tối ưu bộ nhớ dựa trên chuyển đổi mô hình. Các kết quả nghiên cứu về tiêu chí tối ưu này được công bố trong [CT2, CT10]. Trong tiêu chí tối ưu đa mục tiêu, chúng tôi đã đề xuất và phát triển phương pháp tối ưu đa mục tiêu dựa trên biểu đồ lớp. Các kết quả nghiên cứu liên quan đến tiêu chí tối ưu này, đặc biệt là vấn đề tối ưu đa mục tiêu dựa trên nguyên lý Pareto đã được công bố trong [CT4, CT5, CT8, CT11]. Ngoài ra, trong quá trình nghiên cứu, thực nghiệm các phương pháp tối ưu phần mềm nhúng trong giai đoạn thiết kế, chúng tôi cũng nghiên cứu mở rộng cho hệ thống nhúng. Các kết quả nghiên cứu này cũng được xuất bản bao gồm [CT1, CT3, CT4, CT5, CT7, CT8].

*Về vấn đề tối ưu phần mềm nhúng trong giai đoạn lập trình*, chúng tôi tiếp cận nghiên cứu theo hai mức chính tối ưu mã nguồn mức cao độc lập kiến trúc đích và tối ưu mã nguồn hợp ngữ hướng đến các CPU cụ thể. Trong mức tối ưu mã nguồn mức cao độc lập kiến trúc đích, chúng tôi đã hệ thống lại các phương pháp tối ưu cơ bản và dựa trên đó đề xuất hai phương pháp tối ưu là cải tiến hiệu năng dựa trên thay thế các biểu thức tương đương và tối ưu hiệu năng phần mềm điện thoại di động trong môi trường phân tán. Các kết quả nghiên cứu trong phần này đã được xuất bản trong [CT12, CT15 (SCI)]. Trong mức tối ưu mã nguồn hợp ngữ hướng đến các kiến trúc CPU, chúng tôi tập trung nghiên cứu vào hai tiêu chí tối ưu chính là tối ưu hiệu năng và tối ưu năng lượng. Trong mức tối ưu này, chúng tôi đã đề xuất và triển khai phương pháp lập lịch để tối ưu hiệu năng và

điện năng tiêu thụ cho kiến trúc CPU đơn lệnh, đường ống lệnh và siêu vô hướng. Phương pháp tối ưu này đã được công bố trong công trình [CT13, CT15].

*Về vấn đề tối ưu trong giai đoạn thực thi*, đầu tiên chúng tôi đã tổng hợp các phương pháp, kỹ thuật tối ưu hiện có. Trên cơ sở đó, đề xuất và triển khai phương pháp tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU. Nghiên cứu này được công bố trong công trình [CT14].

Ngoài các công cụ DSL, T4 và các chương trình tối ưu, trong quá trình thực nghiệm, chúng tôi cũng xây dựng một số chương trình thử nghiệm như mô-đun nhận dạng chữ Nôm trên điện thoại di động, bộ chương trình thử nghiệm cho MIPS và tổng hợp các chương trình nhúng thử nghiệm cho *Bo mạch Netduino* và *Netduino Plus*. Chương trình thử nghiệm quan trọng nhất là chương trình nhận dạng chữ Nôm trên điện thoại di động trên điện thoại di động hoặc trong môi trường phân tán.

Bên cạnh những đóng góp trên, luận án vẫn còn một số vấn đề chưa giải quyết. Trong giai đoạn thiết kế, các nội dung nghiên cứu trong luận án mới chỉ tập trung vào một số mô hình hệ thống tĩnh như biểu đồ lớp và biểu đồ luồng tác vụ. Các mô hình động của phần mềm như biểu đồ hoạt động, biểu đồ tuần tự, v.v. vẫn chưa được nghiên cứu. Đồng thời các độ đo chất lượng khác cũng chưa được phân tích, kết hợp vào phương pháp tối ưu đa mục tiêu đã xây dựng. Trong giai đoạn lập trình vấn đề phân tích, đánh giá để tìm các đoạn mã tiêu tối nhiều năng lượng và thời gian thực thi lớn cũng chưa được nghiên cứu. Khi tối ưu mã nguồn hợp ngữ hướng đến các CPU đích mới xét các hệ thống đơn CPU mà chưa giải quyết bài toán tối ưu trong các hệ thống đa CPU.

Dựa trên nền tảng các nghiên cứu trong luận án và các vấn đề chưa giải quyết, chúng tôi sẽ tiếp tục tiến hành các nghiên cứu sâu hơn. Về tối ưu trong giai đoạn thiết kế, tiếp tục nghiên cứu các phương pháp tối ưu dựa trên các mô hình động và tích hợp các độ đo chất lượng phần mềm khác vào phương pháp tối ưu đa mục tiêu. Trong giai đoạn lập trình, tiến hành nghiên cứu về tối ưu cho các hệ thống đa CPU, tối ưu trong môi trường không đồng nhất, lập lịch cho các CPU đa nhân. Hơn nữa, chúng tôi cũng sẽ nghiên cứu các thuật toán tiến hóa để giảm độ phức tạp tính toán cho các thuật toán tối ưu. Ngoài ra, trong giai đoạn thực thi, chúng tôi sẽ nghiên cứu sâu hơn các phương pháp phân lớp cho các chương trình trong kỹ thuật JIT.

# DANH MỤC CÁC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN


[CT1]. P.V. Huong, N.N. Binh and B.N Hai (2011), "A Pareto Optimal Configuration at design Phase for SoC Platform Based on the Genetic Algorithm", *Proceedings of IEICE ICDV*, Hanoi, pp. 160-165. (ISBN: 978-4-88522-258-1 C3055).

[CT2]. P.V. Huong, N.N. Binh and P.N. Thanh (2012), "Optimizing occupied Memory of Embedded Software in the design phase", *Journal of Computer Science and Cybernetics*, V.28, N.3, pp. 234-244.

[CT3]. P.V.Huong, N.N. Binh (2012), "Design and Generating Code for Embedded Systems Based on DSL and T4", *Journal of Computer Science and Cybernetics*, V.28, N.4, pp. 323-332.

[CT4]. P.V. Huong, N.N. Binh (2012), "Embedded System Architecture Design and Optimization at the Model Level", *International Journal of Computer and Communication Engineering (IJCCE)*, Vol. 1, No. 4, pp. 345-349. (ISSN: 2010-3743).

[CT5]. P.V. Huong, N.N. Binh, B.N. Hai and V.V. Phuc (2012), "Hardware-Software Co-Design to Optimize Embedded System by Pareto Principle and DSL", *Proceedings of IEICE ICDV*, Hanoi, pp. 52-57. (ISBN: 978-4-88522-264-2 C3055).

[CT6]. P.V. Huong, N.N. Binh, N.T. Huyen, N.T. Duong and T.N. Phu (2012), "Embedded Software Performance Optimization Based on Generating the Simulation Code of Functions", *Proceedings of IEICE ICDV*, Hanoi, pp. 149-154. (ISBN: 978-4-88522-264-2 C3055).

[CT7]. P.V. Huong, N.N Binh (2012), "Embedded System Design and Code Generation by Using the DSL and T4", *Proceedings of International Conference on Electronics Engineering and Informatics (ICEEI)*, Phuket Thailan, pp. 155-160. (ISBN: 978-981-07-3331-5, ISSN: 2010-460X).

[CT8]. P.V. Huong, N.N. Binh (2012), "An Approach to Design Embedded Systems by Multi-objective Optimization", *Proceedings of the 2012 International Conference on Advanced Technologies for Communication (IEEE ATC 2012)*, Hanoi, pp. 165-169 (ISBN: 978-1-4673-4350-3, ISSN: 2162-1020, IEEE Catalog Number: CFB12ATC-PRT).

[CT9]. P.V. Huong, N.N Binh (2012), "Class Diagram Based Evaluation of Software Performance", *Proceedings of International Conference on Information and Digital Engineering (ICIDE)*, SPIE, Vol. 8768, Singapore, pp. 211-217. (DOI: 10.1117/12.2008322, http://spie.org/x648.html?product_id=2008322).

[CT10]. P.V. Huong, N.N. Binh, P.N. Thanh (2012), "Embedded Software Memory Optimization Based on the DSL and Topological Sort", *Proceedings of International Conference on Software and Intelligent Information*, Singapore, pp. 1-5. (DOI: 10.1117/12.2011266, http://spie.org/x648.html?product_id=2011266).

[CT11]. P.V. Huong, N.N. Binh and B.N. Hai (2013), "Multi-objective Optimization for Embedded Software at Model Level Based on DSL and T4", *International Journal of Engineering Research & Technology (IJERT)*, Vol. 2 Issue 9, India, pp. 1229-1236.

[CT12]. P.V. Huong, N.N. Binh and B.N. Hai (2013), "Optimizing Source Code of Embedded Software Based on Replacing Equivalent Expression", *Proceedings of IEICE ICDV*, TP Ho Chi Minh, pp. 193-198.

[CT13]. P.V. Huong, B.N. Hai and N.N. Binh (2014), "An Approach to Instruction Scheduling at the Processor Architecture Level for Optimizing Embedded Software", *Proceedings of the 2014 International Conference on Advanced Technologies for Communication* (IEEE ATC), Hanoi, pp. 226-231.

[CT14]. P.V. Huong, N.N. Binh and V.V. Phuc (2014), "A New Approach to Optimizing the Power Consumption of Existed Embedded Systems Based on the Combination of Hardware and Software", *Proceedings of IEICE ICDV*, Hanoi, pp. 36-40.

[CT15]. N.N. Binh, P.V. Huong and B.N. Hai (2015), "A new approach to embedded software optimization based on reverse engineering", *IEICE Trans. INF. & SYST*, Vol.E98-D, No.6, pp. 1166-1175 (**SCI indexed**).