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

LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

Hà Nội – 2015

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

LUẬN ÁN TIẾN SĨ NGÀNH CÔNG NGHỆ THÔNG TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:

PGS. TS. Nguyễn Ngọc Bình

Hà Nội – 2015

LỜI CAM ĐOAN

Tên tôi là Phạm Văn Hưởng, là nghiên cứu sinh ngành Công nghệ thông tin, chuyên ngành Kỹ thuật phần mềm, khóa K16, trường Đại học Công nghệ – Đại học Quốc gia Hà Nội. Luận án tiến sĩ "*Một số phương pháp tối ưu trong các giai đoạn phát triển phần mềm nhúng*" là công trình nghiên cứu của riêng tôi. Luận án là kết quả của quá trình làm việc nghiêm túc, tài liệu tham khảo có trích dẫn rõ ràng. Các công trình khoa học trình của tôi bày trong luận án đã được sự cho phép sử dụng của đồng tác giả. Tôi xin cam kết và chịu trách nhiệm nếu có sai sót.

Hà Nội, ngày 4 tháng 8 năm 2015

Nghiên cứu sinh

Phạm Văn Hưởng

LỜI CẢM ƠN

Trước hết em xin gửi lời cảm ơn sâu sắc tới PGS.TS. Nguyễn Ngọc Bình đã tận tình chỉ bảo và sửa chữa sai sót giúp em hoàn thành luận án này.

Xin trân trọng cảm ơn các Thầy, Cô trong Hội đồng đánh giá luận án và các Thầy, Cô phản biện đã nhận xét, chỉ ra những khiếm khuyết để luận án được hoàn thiện hơn.

Xin trân trọng cảm ơn các Thầy, Cô trường Đại học Công nghệ - Đại học Quốc gia Hà Nội. Phong cách giảng dạy và sự chỉ bảo nhiệt tình của các Thầy, Cô đã thực sự đem những kiến thức quý báu cũng như những định hướng công nghệ tốt.

Xin chân thành cảm ơn các anh, chị, em trong nhóm nghiên cứu về hệ thống nhúng và phần mềm nhúng tại trường Đại học Công nghệ - Đại học Quốc gia Hà Nội về các góp ý hữu ích và sự phối hợp nghiên cứu.

.

Hà Nội, ngày 4 tháng 8 năm 2015

Nghiên cứu sinh

Phạm Văn Hưởng

MỤC LỤC

BẢNG THUẬT NGỮ

STT	Từ viết tắt	Cụm từ tiếng Anh	Mô tả
1	ALU	Arithmetic and Logic Unit	Bộ lôgic và số học
2	ARM	Advanced RISC Machine	Máy RISC tiên tiến
3	AVR	Alf and Vegard's RISC	Tên một loại vi điều khiển của tập đoàn Atmel, dựa theo tên của hai nhà thiết kế chip đầu tiên
4	CFG	Control Flow Graph	Biểu đồ luồng điều khiển
5	CISC	Complex Instruction Set Computer	Máy tính với tập lệnh phức tạp
6	CPU	Central Processing Unit	Bộ xử lý trung tâm
7	CSE	Common Subexpression Elimination	Loại bỏ biểu thức con chung
8	DAG	Directed Acyclic Graph	Đồ thị có hướng không chu trình
9	DSL	Domain Specific Language	Ngôn ngữ chuyên biệt miền
10	DTG	Dependent Task Graph	Đồ thị tác vụ phụ thuộc
11	ES	Embedded Software	Phần mềm nhúng
12	ESD	Embedded Software Design	Thiết kế phần mềm nhúng
13	ESMO	Embedded Software Memory Optimization	Tối ưu bộ nhớ phần mềm nhúng
14	ESPO	Embedded Software Performance Optimization	Tối ưu hiệu năng phần mềm nhúng
15	FPU	Floating Point Unit	Đơn vị số dấu phẩy động
16	GCC	GNU Compiler Collection	Bộ các trình biên dịch mã nguồn mở GNU
17	GNU	GNU's Not Unix	Hệ điều hành nguồn mở hướng tương thích với Unix
18	GZIP	GNU Zip	Kỹ thuật nén GNU Zip
19	IEEE	Institute of Electrical and Electronics Engineers	Viện kỹ thuật điện và điện tử
20	J2ME	Java to Micro Edition	Nền tảng Java cho các thiết bị

			có tính chất nhỏ, gọn
21	JIT	Just In Time	Kỹ thuật biên dịch tạm
22	JVM	Java Virtual Machine	Máy ảo Java
23	LED	Light Emitting Diode	Điốt phát quang
24	LCD	Liquid Crystal Display	Màn hình tinh thể lỏng
25	MEM	Maximum Entropy Model	Mô hình Entropy cực đại
26	MIPS	Microprocessor without Interlocked Pipeline Stages	Bộ vi xử lý không cần các tầng ống lệnh đồng bộ
27	MOO	Multi-objective Optimization	Tối ưu đa mục tiêu
28	MSIL	Microsoft Intermediate Language	Ngôn ngữ trung gian Microsoft
29	MSQ	Measures of Software Quality	Độ đo chất lượng phần mềm
30	OMT	Optimizing Embedded Software Based on Model Transformation	Tối ưu phần mềm nhúng dựa trên chuyển đổi mô hình
31	OOP	Object-Oriented Programming	Lập trình hướng đối tượng
32	OS	Operating System	Hệ điều hành
33	OSC	Optimizing Embedded Software Based on Simulation Code	Tối ưu phần mềm nhúng dựa trên mã mô phỏng
34	PC	Personal Computer	Máy tính cá nhân
35	PHP	Hypertext Preprocessor	Ngôn ngữ lập trình web PHP
36	PMO	Pareto Multi-objective Optimization	Tối ưu đa mục tiêu Pareto
37	RAM	Random Access Memory	Bộ nhớ truy xuất ngẫu nhiên
38	RISC	Reduced Instruction Set Computer	Máy tính với tập lệnh rút gọn
39	ROM	Read Only Memory	Bộ nhớ chỉ đọc
40	SDK	Software Development Kit	Bộ công cụ phát triển phần mềm
41	SOAP	Simple Object Access Protocol	Giao thức truy xuất đối tượng đơn giản
42	SoC	System on Chip	Hệ thống tích hợp trên một chip
43	SOO	Single Objective Optimization	Tối ưu đơn mục tiêu
44	SPE	Software Performance Engineering	Kỹ nghệ hiệu năng phần mềm
45	SPO	Software Performance	Tối ưu hiệu năng phần mềm

		Optimization	
46	SQL	Structured Query Language	Ngôn ngữ truy vấn có cấu trúc
47	T4	Text Template Transformation Toolkit	Bộ công cụ chuyển đổi dựa trên mẫu văn bản
48	UML	Unified Modeling Language	Ngôn ngữ mô hình hóa thống nhất
49	XML	eXtensible Markup Language	Ngôn ngữ đánh dấu mở rộng

BẢNG CÁC KÝ HIỆU

Ký hiệu	Mô tả
a_s^j	Biến tĩnh thứ *j* trong một lớp
a_o^j	Biến đối tượng thứ *j* trong một lớp
p_k	Tham số thứ *k* trong một phương thức
C	Tập các lớp trong một biểu đồ
M_s^i	Tập các phương thức tĩnh trong lớp thứ *i*
V_s^i	Tập các biến tĩnh trong lớp thứ *i*
M_o^i	Tập các phương thức đối tượng trong lớp thứ *i*
V_o^i	Tập các biến đối tượng trong lớp thứ *i*
P_m^j	Tập các tham số trong phương thức thứ *j*
v_r^j	Biến tham chiếu để gọi phương thức thứ *j*
r_e^j	Kiểu trả về của phương thức thứ *j*
α	Hệ số phụ thuộc của hiệu năng vào các thành phần tĩnh
β	Hệ số phụ thuộc của hiệu năng vào kích thước thực thi phương thức tĩnh
γ	Hệ số phụ thuộc của hiệu năng vào các thành phần đối tượng
ε	Hệ số phụ thuộc của hiệu năng vào kích thước thực thi phương thức đối tượng
f_p	Hàm đánh giá hiệu năng
f_m	Hàm đánh giá bộ nhớ
f_e	Hàm đánh giá điện năng tiêu thụ
f_c	Hàm đánh giá chi phí
f	Hàm mục tiêu toàn cục
f_1	Hàm mục tiêu hiệu năng
f_2	Hàm mục tiêu bộ nhớ
S_1	Kích thước các biến tĩnh
S_2	Kích thước các phương thức tĩnh
S_3	Kích thức thực thi các phương thức tĩnh
S_4	Kích thước các biến đối tượng
S_5	Kích thước các phương thức đối tượng

S_6	Kích thước thực thi các phương thức đối tượng
G	Đồ thị tác vụ phụ thuộc
U	Tập đỉnh trong đồ thị phụ thuộc
V	Tập cạnh trong đồ thị phụ thuộc
N	Số tác vụ
M	Kích thước bộ nhớ chiếm dụng
S_c	Kích thước bộ nhớ chứa mã lệnh
S_e	Kích thước vùng nhớ heap được cấp phát động khi thực thi tác vụ
r_i	Kiểu dữ liệu trả về của tác vụ i
D_s	Kích thước bộ nhớ chứa dữ liệu tĩnh
x	Một cấu trúc dữ liệu
a_i	Thành phần dữ liệu thứ i trong cấu trúc
m_i	Hàm thứ i trong cấu trúc
n	Tần số sử dụng của một thành phần trong cấu trúc
A_o	Tổng số thành phần dữ liệu trong x
M_o	Tổng số hàm trong một cấu trúc
K	Số cấu trúc mới được tạo ra
t_0	Thời gian cư trú trong bộ nhớ của một byte dữ liệu
w_i	Trọng số của hàm mục tiêu i
t_u	Thời gian trung bình để nén một byte dữ liệu
t_d	Thời gian trung bình để giải nén một byte
t_c	Thời gian trung bình để truyền một byte dữ liệu
r	Tỉ lệ nén
T	Tổng thời gian thực thi
T_1	Thời gian xử lý trên điện thoại
T_2	Thời gian xử lý trên máy chủ
T_3	Thời gian dữ liệu đi trên đường truyền
T_d	Thời gian giảm
T_i	Thời gian tăng lên
T_t	Tổng thời gian tiết kiệm
N_I	Số câu lệnh trong chương trình
N_s	Số giai đoạn trong kiến trúc đường ống lệnh
T_s	Khoảng thời gian của một giai đoạn trong đường ống

t_a	Thời gian thực hiện một lệnh
s_i	Thời gian trễ của câu lệnh thứ i
n_d	Số đoạn trễ của một câu lệnh
$S_I^{\ k}$	Nhóm lệnh đã giải mã trong cửa sổ lệnh tại bước k
$E_I^{\ k-1}$	Tập các lệnh đã được thực thi tại bước $k-1$
$S_N^{\ k}$	Các lệnh tiếp theo trong chuỗi lệnh ban đầu
f_p^k	Hàm đánh giá hiệu năng, được định nghĩa đệ quy tại bước k
P_{CPU}	Điện năng tiêu thụ của CPU
P_d	Điện năng tiêu thụ động
P_s	Điện năng tiêu thụ do đoản mạch
P_l	Điện năng lãng phí
C_c	Điện dung
f_x	Tần số đồng hồ trong CPU
V_c	Điện áp
τ	Tỉ lệ số cổng lôgic thay đổi trạng thái
E_1	Năng lượng tiêu thụ của chương trình với cấu hình đầy đủ
E_2	Năng lượng tiêu thụ của chương trình với cấu hình tối ưu
ΔE	Năng lượng tiết kiệm
N_u	Tổng số đơn vị chức năng trong CPU
N_{u1}	Tổng số đơn vị chức năng được sử dụng trong CPU

DANH MỤC CÁC BẢNG

DANH MỤC CÁC HÌNH VẼ

MỞ ĐẦU

Ngày nay, 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. Theo thống kê trong [107] năm 1999, có đến 99% số vi xử lý thuộc về các hệ thống nhúng. Thống kê trong [94] năm 2011 cũng chỉ ra: có tới trên 95% số chip sản xuất được dùng cho các ứng dụng nhúng. Các sản phẩm nhúng có mặt trong mọi lĩnh vực đời sống như ti-vi, tủ lạnh, máy giặt, lò vi sóng, máy ảnh số, ô-tô, v.v. Hầu hết các máy công nghiệp hiện nay đều có chứa các hệ nhúng, nhất là các thiết bị y tế, giao thông, quân sự, thông tin liên lạc, vũ trụ, v.v.

Sự hội tụ của công nghệ truyền thông không dây với thiết bị tính toán cầm tay cùng với sự thúc đẩy của công nghệ vi điện tử và công nghệ nanô là nền tảng cho sự phát triển mạnh mẽ của phần cứng hệ thống nhúng. Gắn liền với sự phát triển không ngừng của công nghệ phần cứng, phần mềm hệ thống nhúng cũng được nghiên cứu và phát triển sâu, rộng. Do nhu cầu của thị trường đòi hỏi các thiết bị phải có nhiều chức năng thân thiện với người dùng, có mức độ thông minh ngày càng cải thiện dẫn đến vai trò và tầm quan trọng của các hệ thống nhúng ngày càng cao trong nền kinh tế quốc dân.

Từ tất yếu khách quan, công nghệ phần mềm nhúng đã trở thành lĩnh vực công nghệ then chốt cho sự phát triển kinh tế của nhiều quốc gia trên thế giới tiêu biểu như Mỹ, Nhật Bản, Hàn Quốc, Phần Lan và Trung quốc. Ở Việt Nam, chưa có sự liên kết chặt chẽ giữa trường đại học, viện nghiên cứu với các công ty, doanh nghiệp; chưa phối hợp rộng rãi giữa nghiên cứu hàn lâm và phát triển ứng dụng; phần lớn các công ty phần mềm đều tập trung xây dựng phần mềm ứng dụng hoặc gia công phần mềm [1]. Số lượng các công ty phát triển hệ thống nhúng và phần mềm nhúng còn ít và chủ yếu làm theo yêu cầu của các đối tác nước ngoài. Mặc dù vậy, hệ thống nhúng và phần mềm nhúng vẫn là một xu hướng phát triển của công nghệ thông tin và bước đầu được nghiên cứu, phát triển trong nhiều trường đại học, viện nghiên cứu và một số công ty, doanh nghiệp. Nghiên cứu và phát triển hệ thống nhúng và phần mềm nhúng góp phần quan trọng giúp chúng ta có thể theo kịp, rút ngắn khoảng cách tụt hậu về công nghệ thông tin và truyền thông so với các nước trong khu vực và trên thế giới.

Phát triển hệ thống nhúng là bài toán tổng thể gồm nhiều khía cạnh và được thực hiện trong các giai đoạn như: đặc tả yêu cầu, phân tích, thiết kế, phát triển, mô phỏng và sản xuất. Thiết kế hệ thống nhúng bao gồm bốn mức đó là mức hệ thống, mức vi xử lý (CPU), mức lôgic và mức mạch. Thiết kế mức hệ thống bao gồm thiết kế một hệ thống tính toán và phân chia phần cứng – phần mềm. Sau đó, giai đoạn phát triển hệ nhúng cũng được chia thành phát triển phần cứng và phát triển phần mềm. So với phần mềm thông

thường, phần mềm nhúng thường gắn liền và phụ thuộc vào môi trường phần cứng. 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.

Căn cứ vào xu hướng khách quan, hiện trạng nghiên cứu và nhu cầu thực tiễn, đề tài "*Một số phương pháp tối ưu trong các giai đoạn phát triển phần mềm nhúng*" là đề tài mang tính thời sự, cấp thiết, có ý nghĩa khoa học và có triển vọng trong ứng dụng thực tiễn. *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. Đồng thời các phương pháp này cũng cung cấp nền tảng và mở ra hướng mới cho các nghiên cứu tiếp theo. *Về thực tiễn*, đề tài có khả năng phát triển và ứng dụng thực tế, góp phần giải quyết một số vấn đề đang được quan tâm hàng đầu của các công ty, doanh nghiệp phát triển hệ thống nhúng, phần mềm nhúng như hiệu năng, năng lượng tiêu thụ, chi phí, v.v.

Dựa trên cơ sở khoa học và khả năng ứng dụng thực tiễn, luận án đã được triển khai với những đóng góp chính sau:

- Xây dựng mô hình tối ưu chung và đề xuất cách tiếp cận tối ưu theo cả kỹ nghệ xuôi và kỹ nghệ ngược.
- 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; được thực hiện tự động một phần dựa trên DSL và T4. 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. [7, 8] 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.

Ngoài ra, luận án cũng hướng đến xây dựng và thử nghiệm phần mềm nhận dạng chữ Nôm trên điện thoại di động để kiểm chứng một số phương pháp tối ưu. Đây là một phần mềm thực nghiệm có đóng góp quan trọng trong bảo tồn di sản văn hóa quốc gia. Luận án được thực hiện theo cấu trúc trong Hình 1 với các nội dung sau:

***Chương 1*: Tổng quan**. Trong chương này, chúng tôi hệ thống hóa, tổng hợp các nghiên cứu liên quan để xây dựng mô hình chung về tối ưu phần mềm nhúng. Theo mô

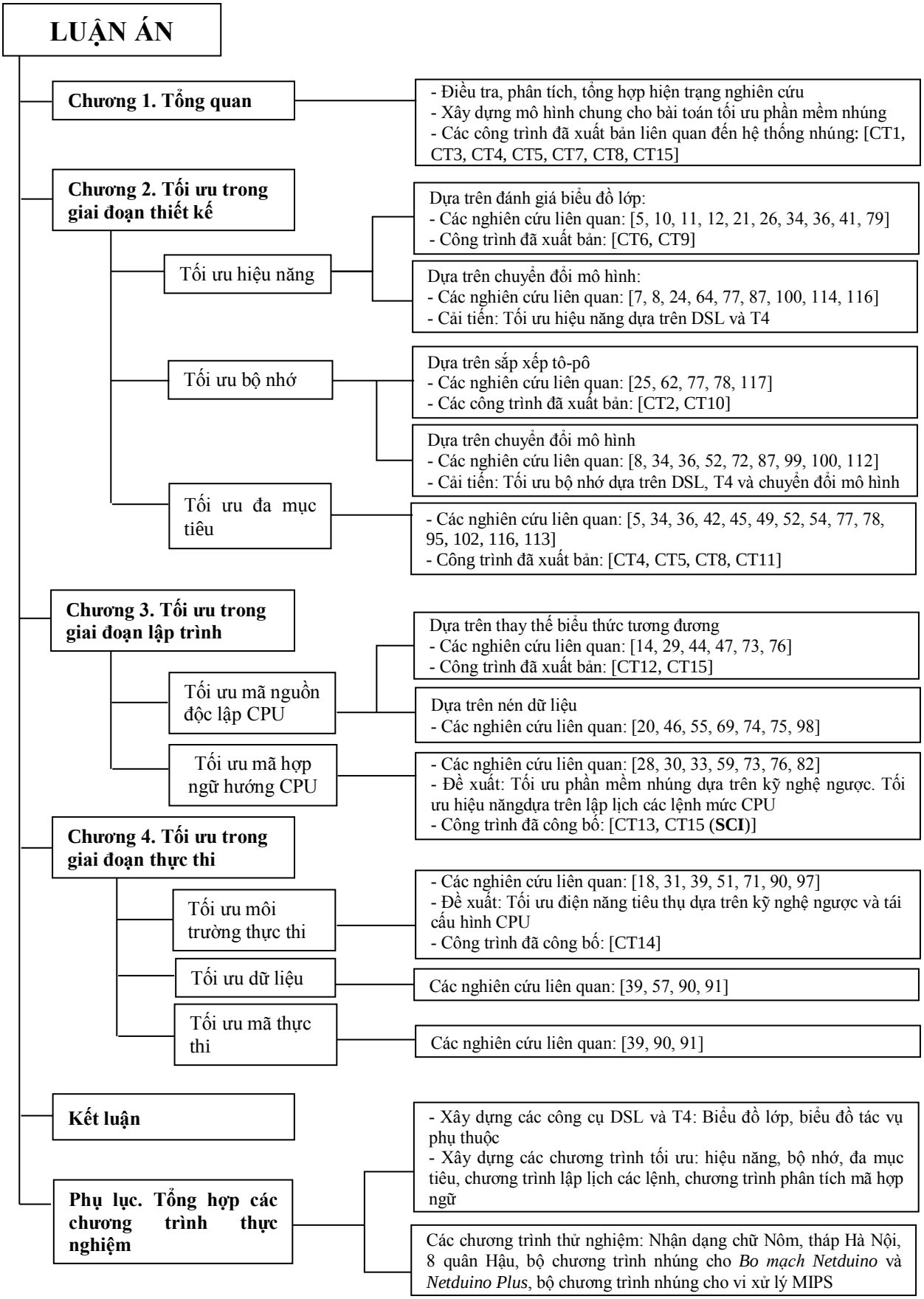
hình chung, tối ưu phần mềm nhúng có thể tiến hành theo các tiêu chí tối ưu khác nhau như hiệu năng, bộ nhớ, điện năng tiêu thụ v.v. và tối ưu đa mục tiêu. Tối ưu có thể được thực hiện trong các giai đoạn phát triển phần mềm nhúng như thiết kế, lập trình, thực thi. Trong mỗi giai đoạn, chúng tôi phân tích tình hình nghiên cứu hiện tại, xác định các vấn đề chưa giải quyết để từ đó chỉ ra các khoảng trống khoa học sẽ được thực hiện trong các chương tiếp theo của luận án.

***Chương 2*: Tối ưu phần mềm nhúng trong giai đoạn thiết kế.** Để tiến hành nghiên cứu và thực nghiệm theo mô hình tối ưu chung, đầu tiên chúng tôi nghiên cứu và triển khai một số phương pháp tối ưu trong giai đoạn thiết kế. Về tối ưu hiệu năng, chúng tôi đề xuất phương pháp tối ưu mới dựa trên đánh giá biểu đồ lớp, ngôn ngữ chuyên biệt miền (DSL – Domain Specific Language), công cụ sinh mã T4 và cải tiến phương pháp tối ưu hiệu năng dựa trên chuyển đổi mô hình. Về tối ưu bộ nhớ, chúng tôi đã đề xuất, phát triển phương pháp tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô và cải tiến phương pháp tối ưu bộ nhớ dựa trên chuyển đổi mô hình. Về tối ưu đa mục tiêu, chúng tôi đã đề xuất và phát triển phương pháp tối ưu dựa trên biểu đồ lớp và nguyên lý Pareto.

***Chương 3*: Tối ưu phần mềm nhúng trong giai đoạn lập trình.** Giai đoạn lập trình (implementation phase) theo nghĩa rộng bao gồm các công việc như viết mã, biên dịch, kiểm thử đơn vị, v.v. Đây là giai đoạn tối ưu được nghiên cứu phổ biến nhất. Phần đầu chương tổng hợp về quá trình biên dịch chéo và các mức tối ưu trong giai đoạn lập trình. Phần tiếp theo trình bày các phương pháp tối ưu mã nguồn mức cao độc lập CPU. Trong mức tối ưu này, chúng tôi tóm lược cơ sở lý thuyết, đề xuất một cải tiến cho phương pháp tối ưu cục bộ đó là loại bỏ biểu thức con chung dựa trên thay thế biểu thức tương đương và triển khai phương pháp tối ưu dựa trên nén dữ liệu. Trong mức tối ưu mã hợp ngữ, chúng tôi cũng tóm lược cơ sở lý thuyết sau đó đề xuất và phát triển phương pháp lập lịch các lệnh trong chương trình để tối ưu hiệu năng và điện năng tiêu thụ.

***Chương 4*: Tối ưu phần mềm nhúng trong giai đoạn thực thi**. Nội dung chương này sẽ tóm lược cơ sở lý thuyết cũng như trình bày các phương pháp tối ưu theo ba hướng tiếp cận chính đó là tối ưu môi trường thực thi, tối ưu hướng dữ liệu và tối ưu mã thực thi. Trên cơ sở lý thuyết đã tổng hợp, chúng tôi đã đề xuất và phát triển phương pháp mới để tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU.

Kết luận: Phần này sẽ tổng hợp, đánh giá các nội dung nghiên cứu và thực nghiệm về các phương pháp tối ưu đã trình bày trong các chương trước. Trong mỗi giai đoạn, chúng tôi tóm lược các đóng góp của luận án và trình bày các kết quả nghiên cứu đã công bố trong các kỷ yếu hội nghị, tạp chí về các phương pháp tối ưu. Sau đó, chúng tôi cũng chỉ ra hạn chế và phạm vi áp dụng của mỗi phương pháp, các vấn đề chưa giải quyết và đưa ra một số hướng nghiên cứu tiếp theo.



Hình 1: Cấu trúc tổng thể của luận án

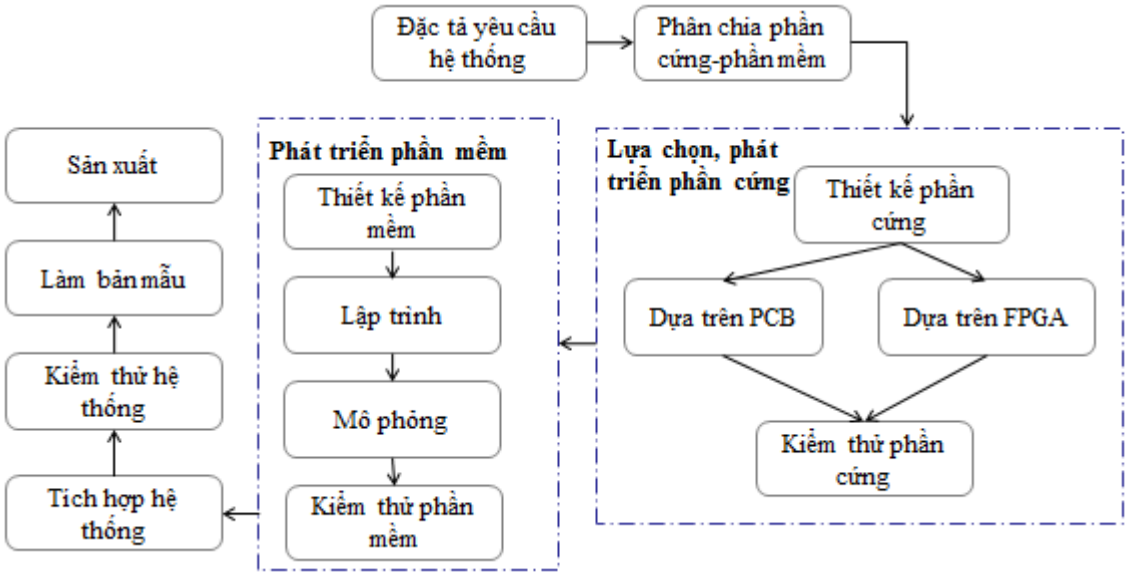
Chương 1. TỔNG QUAN

Trong phát triển phần mềm nhúng, tối ưu là bài toán có ý nghĩa khoa học và mang tính thực tiễn cao. Tối ưu phần mềm nhúng có phạm vi nghiên cứu rộng và có thể được triển khai trong các giai đoạn khác nhau của vòng đời phần mềm. Bài toán tối ưu phần mềm nhúng cũng có thể được tiếp cận theo các mục tiêu tối ưu và các mức tối ưu khác nhau. Do đó, để có thể triển khai nghiên cứu một cách hệ thống và đầy đủ, trong chương này, chúng tôi trình bày tổng quan về phát triển, tối ưu hệ thống nhúng, phần mềm nhúng và xây dựng mô hình tối ưu chung bao gồm các giai đoạn và tiêu chí tối ưu. Trong mỗi giai đoạn, chúng tôi tổng hợp, phân tích và đánh giá các nghiên cứu liên quan để phân nhóm các phương pháp tối ưu cũng như chỉ ra các khó khăn, thách thức và những vấn đề cần giải quyết. Trên cơ sở đó, chúng tôi đưa ra cách tiếp cận, phương pháp và nội dung nghiên cứu, triển khai trong luận án. Nội dung chương này được bố cục như sau: *Mục 1.1* trình bày tổng quan về tối ưu hệ thống nhúng, phần mềm nhúng và mô hình chung cho bài toán tối ưu phần mềm nhúng, *Mục 1.2* trình bày về hiện trạng nghiên cứu và các thách thức trong tối ưu phần mềm nhúng, *Mục 1.3* trình bày về phương pháp và nội dung nghiên cứu, *Mục 1.4* tổng kết chương.

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

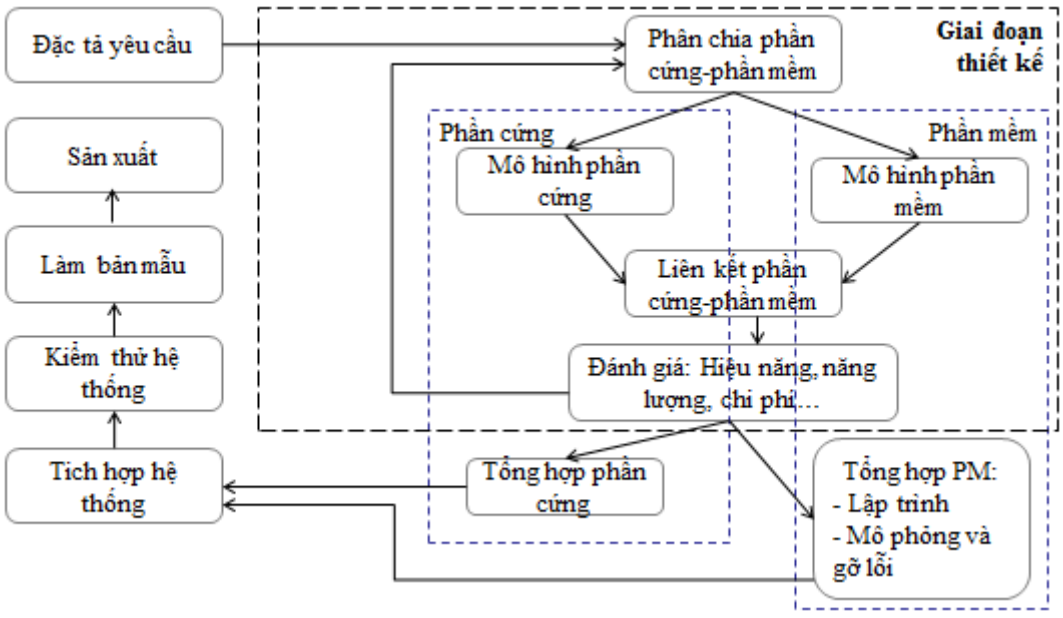
Hệ thống nhúng là hệ thống máy tính cùng phần mềm được gắn theo một hệ thống khác để điều khiển hoạt động và xử lý thông tin của hệ thống đó. Khái niệm "*phần mềm nhúng*" trong luận án được hiểu là phần mềm chạy trong các hệ thống nhúng. Các hệ thống nhúng có thể được phát triển theo hai phương pháp phổ biến là phương pháp truyền thống và phương pháp đồng thiết kế. Quy trình phát triển hệ thống nhúng theo phương pháp truyền thống được minh họa trong Hình 1.1 [47]. Căn cứ vào đặc tả yêu cầu, mỗi tác vụ hệ thống có thể được phân thành tác vụ phần cứng hoặc tác vụ phần mềm. Các tác vụ phần cứng được thực hiện theo các mạch tích hợp có sẵn hoặc xây dựng mới còn các tác vụ phần mềm được thực hiện bởi CPU. Phát triển phần cứng là giai đoạn tiếp theo sau khi phân chia phần cứng – phần mềm. Giai đoạn phát triển phần mềm được thực hiện sau khi đã phát triển phần cứng. Tiếp đến là giai đoạn tích hợp phần mềm lên hệ thống phần cứng, kiểm thử hệ thống, làm bản mẫu và sản xuất. Phương pháp truyền thống có hai hạn

chế chính là thời gian phát triển lâu và không linh động do phần mềm được phát triển cho phần cứng cố định và không thể thay đổi việc phân chia phần cứng – phần mềm.



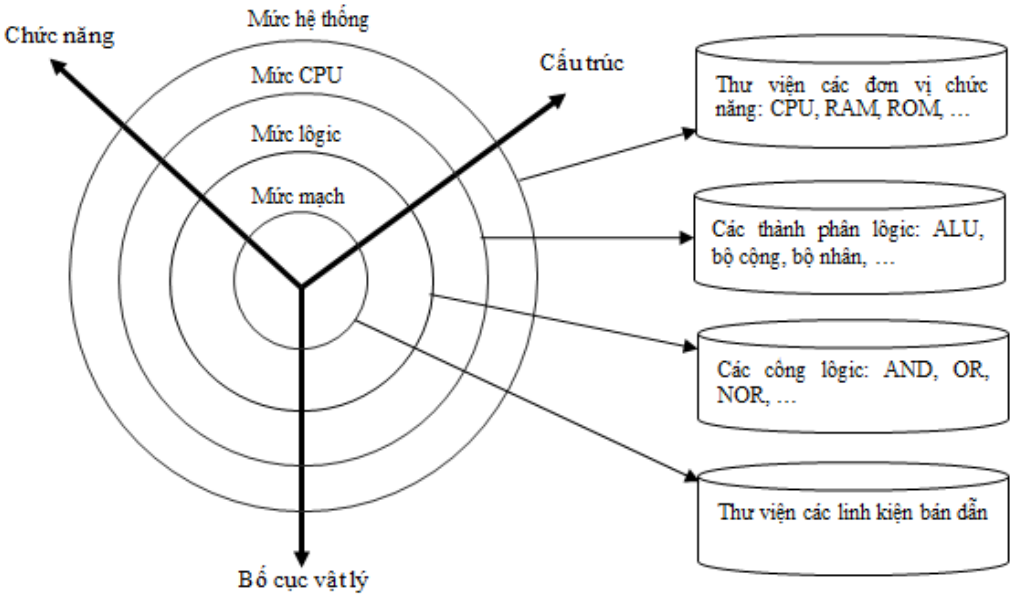
Hình 1.1: Quy trình phát triển hệ thống nhúng theo phương pháp truyền thống

Phương pháp đồng thiết kế phần cứng – phần mềm là cách tiếp cận mới, khắc phục các hạn chế của phương pháp truyền thống. Quy trình phát triển hệ thống nhúng theo phương pháp đồng thiết kế được chỉ ra trong Hình 1.2 [47]. Khác với phương pháp truyền thống, theo phương pháp này, phần cứng và phần mềm được phát triển đồng thời và việc phân chia phần cứng – phần mềm có thể được thay đổi dựa trên việc đánh giá hiệu năng, mức tiêu thụ năng lượng, chi phí, v.v. Do đó, đồng thiết kế cũng là cách tiếp cận để tối ưu mức hệ thống nhằm lựa chọn một phân chia phần cứng – phần mềm tốt nhất.



Hình 1.2: Quy trình phát triển hệ thống nhúng theo phương pháp đồng thiết kế

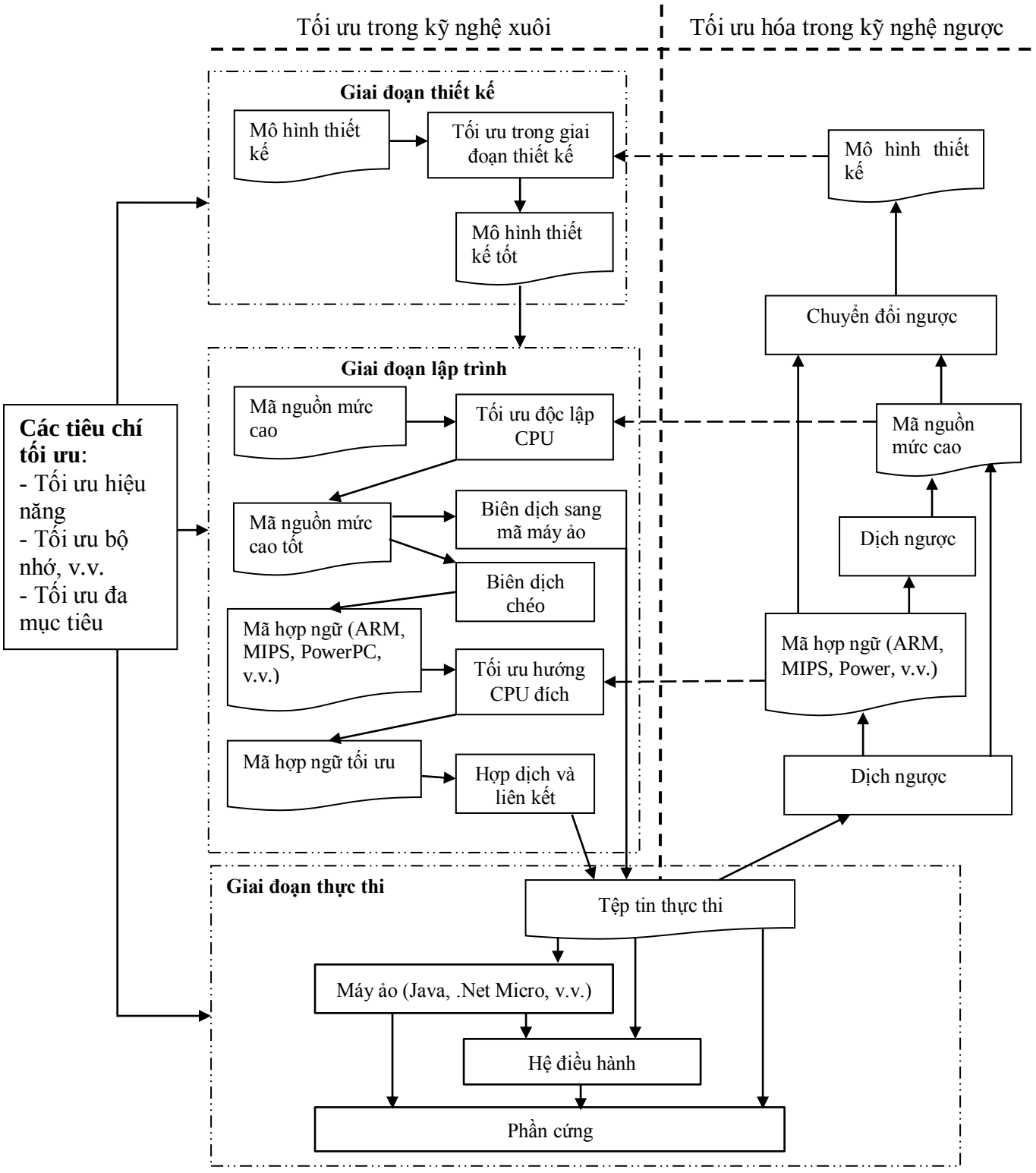
Hai phương pháp phát triển hệ thống nhúng trên đều gồm ba giai đoạn chính là phát triển phần cứng, phát triển phần mềm và tích hợp. Quy trình phát triển phần mềm nhúng cũng như phát triển phần mềm thông thường, gồm các bước chính theo mô hình thác nước đó là phân tích, thiết kế, lập trình, kiểm thử, triển khai và bảo trì. Điểm khác biệt chính là việc lập trình phần mềm nhúng thường gắn với một CPU và thường phải sử dụng môi trường phát triển chéo. Phát triển phần cứng hệ thống nhúng thường được thực hiện theo mô hình chữ Y như trong Hình 1.3. Phát triển phần cứng hệ thống nhúng được chia thành bốn mức từ mức mạch đến mức hệ thống và được thiết kế theo ba khía cạnh là chức năng, cấu trúc và bố cục vật lý. Trong mức hệ thống, dựa vào các thành phần cơ bản như CPU, RAM, ROM, cache, v.v. để thiết kế, xây dựng lên hệ thống nhúng. Trong mức CPU, các phần tử cơ bản là các đơn vị chức năng như ALU, CU, bộ cộng, bộ nhân, các thanh ghi, v.v. được kết hợp để thiết kế, xây dựng CPU và các thành phần mức trên khác. Các phần tử cơ bản trong mức CPU lại được thiết kế và xây dựng từ các cổng lôgic như AND, OR, NOT, XOR, v.v. trong mức lôgic. Các cổng lôgic lại được thiết kế và xây dựng từ các thiết bị bán dẫn trong mức mạch.



Hình 1.3: Quy trình thiết kế và xây dựng phần cứng

Căn cứ vào quy trình phát triển ở trên, bài toán tối ưu hệ thống nhúng bao gồm tối ưu cục bộ phần cứng, tối ưu cục bộ phần mềm và đồng thiết kế – *tối ưu dựa trên phân chia phần cứng – phần mềm*. Tương ứng với các mức phát triển, tối ưu cục bộ phần cứng có thể thực hiện ở bốn mức đó là tối ưu mức hệ thống, tối ưu mức CPU, tối ưu mức lôgic và tối ưu mức mạch. Tối ưu cục bộ phần mềm nhúng cũng phức tạp và đa dạng hơn nhiều so với tối ưu phần mềm thông thường do sự phụ thuộc vào phần cứng và môi trường phát triển. Tối ưu phần mềm nhúng thường gồm nhiều tiêu chí tối ưu như hiệu năng, mức tiêu thụ điện năng, bộ nhớ, v.v. và có thể thực hiện trong các giai đoạn khác nhau như giai

đoạn thiết kế, giai đoạn lập trình và giai đoạn thực thi [24, 29, 33, 38, 42]. Sau khi đã điều tra, phân tích và đánh giá các nghiên cứu về tối ưu phần mềm nhúng và dựa trên các phương pháp phát triển hệ thống nhúng ở trên, chúng tôi tổng hợp và xây dựng mô hình chung cho bài toán tối ưu trong phát triển phần mềm nhúng như trong Hình 1.4.



Hình 1.4: Mô hình tối ưu tổng thể trong phát triển phần mềm nhúng

Theo mô hình chung trong Hình 1.4, 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. Mô hình tối ưu chung này áp dụng cho cả phần mềm nhúng đã có và phần mềm nhúng xây dựng mới. Hướng tiếp cận tối ưu trong kỹ nghệ xuôi, bắt đầu từ đặc tả yêu cầu, chúng ta có thể thiết kế phần mềm nhúng theo các mô hình 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, phần mềm nhúng có thể được lập trình theo mã nguồn mức cao độc lập CPU và các phương pháp tối ưu có thể được thực hiện 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. Tuy nhiên, trong giai đoạn sau, mã nguồn mức cao được biên dịch chéo để tạo thành mã hợp ngữ gắn với một loại CPU cụ thể. Do đó, trong mức mã hợp ngữ, các phương pháp tối ưu thường mang tính đặc thù theo kiến trúc CPU và môi trường phần cứng cụ thể của mỗi hệ thống nhúng. Trong giai đoạn thực thi, phần mềm nhúng đã được tích hợp vào hệ thống nhúng nên 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. Trong giai đoạn này, các tệp tin thực thi có thể được thực hiện ngay trên phần cứng, có thể được thực hiện thông qua hệ điều hành nhúng hoặc được thực hiện trên máy ảo. Máy ảo lại có thể được thực hiện ngay trên phần cứng hoặc được thực hiện thông qua hệ điều hành [15, 21, 61]. Các kiểu môi trường thực thi khác nhau cũng có các phương pháp tối ưu khác nhau và mang tính đặc thù. Ngoài ra, mã nguồn mức cao cũng có thể được biên dịch trực tiếp sang mã máy ảo. Khi thực hiện, mã máy ảo lại được thông dịch chéo sang mã hợp ngữ. Do đó, trong giai đoạn thực thi, chúng ta có thể tiến hành các phương pháp tối ưu trên mã máy ảo.

Hướng tiếp cận tối ưu trong kỹ nghệ xuôi chỉ có thể áp dụng khi xây dựng mới một phần mềm nhúng. Để tối ưu hóa các phần mềm nhúng đã có chúng ta cần nghiên cứu, áp dụng các kỹ thuật trong kỹ nghệ ngược. Kỹ nghệ ngược là một khía cạnh quan trọng trong tái kỹ nghệ phần mềm. Đây là một xu hướng mới và có nhiều triển vọng trong phát triển phần mềm. 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. Theo đó, hướng tiếp cận tối ưu trong kỹ nghệ ngược bao gồm các phương pháp tối ưu trong kỹ nghệ xuôi kết hợp với các kỹ thuật chuyển đổi ngược. Đầu ra tại mỗi mức trong kỹ nghệ ngược có thể được tối ưu theo mức tương ứng trong kỹ nghệ xuôi. Do đó, để giải quyết bài toán tối ưu tổng thể, trước hết cần nghiên cứu các phương pháp, kỹ thuật tối ưu trong kỹ nghệ xuôi.

1.1.1. Tối ưu đơn mục tiêu

Mỗi phần mềm nhúng thường phát triển cho một loại CPU với tập thanh ghi và kiến trúc tập lệnh khác nhau như kiến trúc tập lệnh rút gọn (RISC), kiến trúc tập lệnh phức tạp (CISC), v.v. Hơn nữa, mỗi phần mềm nhúng cũng làm việc với một số thiết bị ngoại vi cụ thể như đèn giao thông, đầu đọc thẻ, màn hình cảm ứng, v.v. Bên cạnh đó, phần mềm nhúng thường được thực hiện trong các ràng buộc chặt chẽ do hạn chế của phần cứng cũng như yêu cầu của các ứng dụng nhúng thời gian thực. Đồng thời việc kiểm thử và dò lỗi của phần mềm nhúng cũng rất khó khăn vì phải thực hiện trên các *Bo mạch* khác nhau hoặc thực hiện mô phỏng trong các bộ công cụ khác nhau. Ngoài ra, môi trường và công cụ phát triển phần mềm nhúng cũng rất đa dạng và thường phải xây dựng các môi trường phát triển chéo, trình biên dịch chéo. Bộ công cụ này chạy trên PC nhưng biên dịch ra mã đích chạy trên các vi xử lý đích khác như ARM [93], MIPS [50], v.v. hay chạy trên các vi điều khiển như AVR [86], 8051 [9], v.v. Do đó, việc tối ưu phần mềm nhúng càng có ý nghĩa hơn so với tối ưu phần mềm thông thường. Tối ưu phần mềm nhúng tập trung vào các tiêu chí sau: tối ưu hiệu năng (f_p) thường dựa trên thời gian thực thi phần mềm nhúng; tối ưu điện năng tiêu thụ (f_e) nhằm giảm mức tiêu thụ điện năng và kéo dài thời gian sống của pin; tối ưu bộ nhớ (f_m) hướng đến tối ưu mức chiếm dụng bộ nhớ hoặc tối giản kích thước phần mềm nhúng và tối ưu chi phí (f_c) nhằm giảm chi phí mà vẫn thỏa mãn các yêu cầu của phần mềm nhúng [49, 92, 106, 113]. Trong tối ưu phần mềm nhúng, tối ưu đơn mục tiêu thường tập trung cải tiến, tối ưu một trong các tiêu chí trên. Tuy nhiên tối ưu mỗi mục tiêu cần đặt trong các ràng buộc trên các mục tiêu còn lại. Do đó bài toán tối ưu đơn mục tiêu trong phát triển phần mềm nhúng được chia thành các loại theo các đặc tả từ (1.1) đến (1.4).

- **Tối ưu hiệu năng**

$$\begin{cases} f_p \rightarrow \min \\ f_e \le f_e^0 \\ f_m \le f_m^0 \\ f_c \le f_c^0 \end{cases} \tag{1.1}$$

- **Tối ưu bộ nhớ**

$$\begin{cases} f_m \rightarrow \min \\ f_p \le f_p^0 \\ f_e \le f_e^0 \\ f_c \le f_c^0 \end{cases} \tag{1.2}$$

- **Tối ưu tiêu thụ điện năng**

$$\begin{cases} f_e \rightarrow \min \\ f_p \leq f_p^{\,0} \\ f_m \leq f_m^{\,0} \\ f_c \leq f_c^{\,0} \end{cases} \quad (1.3)$$

- **Tối ưu chi phí**

$$\begin{cases} f_c \rightarrow \min \\ f_p \leq f_p^{\,0} \\ f_e \leq f_e^{\,0} \\ f_m \leq f_m^{\,0} \end{cases} \quad (1.4)$$

1.1.2. Tối ưu đa mục tiêu

Như trình bày trong phần trước, mỗi hệ thống nhúng, phần mềm nhúng thường có nhiều mục tiêu tối ưu. Trong quá trình tối ưu, các mục tiêu tối ưu thường không được thỏa mãn đồng thời. Tối ưu tiêu chí này có thể làm tiêu chí khác xấu đi. Do đó, một vấn đề đặt ra trong tối ưu phần mềm nhúng là tối ưu đa mục tiêu. Tối ưu đa mục tiêu không hướng đến tối ưu một tiêu chí cụ thể mà luôn cải tiến mọi mục tiêu tối ưu để hướng đến tính cân bằng của hệ thống. Phương pháp điển hình để giải quyết bài toán tối ưu đa mục tiêu là dựa trên nguyên lý Pareto [42, 113]. Nguyên lý này hướng đến phân phối cân bằng giữa các mục tiêu trong miền tối ưu Pareto. Để triển khai phương pháp này, chúng ta cần xây dựng các hàm mục tiêu tối ưu thành phần và hàm mục tiêu toàn cục. Hàm mục tiêu toàn cục được xây dựng dựa trên các hàm mục tiêu thành phần và trọng số tương ứng; trọng số thường gắn với độ ưu tiên của mục tiêu tối ưu thành phần. Hệ thống được đưa về miền tối ưu Pareto khi giá trị hàm mục tiêu toàn cục cực đại hoặc cực tiểu tùy theo cách xây dựng các hàm tối ưu. Nguyên lý Pareto là điển hình để giải quyết vấn đề tối ưu trong thiết kế hệ thống nói chung nhưng còn ít được nghiên cứu trong tối ưu phần mềm nhúng. Vấn đề cốt lõi để áp dụng nguyên lý này là xây dựng các hàm mục tiêu cục bộ tương ứng với các tiêu chí tối ưu; sau đó xây dựng hàm mục tiêu toàn cục để đánh giá dựa trên các hàm cục bộ và mảng trọng số.

1.1.3. Các giai đoạn tối ưu

Trong mô hình chung, tối ưu phần mềm nhúng có thể được thực hiện trong các giai đoạn khác nhau như giai đoạn thiết kế, giai đoạn lập trình và giai đoạn thực thi. Trong mỗi giai đoạn, phần mềm nhúng có thể được tiến hành tối ưu theo đơn mục tiêu hoặc đa mục tiêu và theo các cách tiếp cận riêng.

- **Tối ưu trong giai đoạn thiết kế**: Đây là hướng nghiên cứu quan trọng và triển vọng vì tối ưu trong giai đoạn thiết kế mang lại nhiều lợi ích hơn trong các giai đoạn sau. Hơn nữa, không thể có một phần mềm tốt trên một thiết kế tồi. Mục tiêu của tối ưu trong giai đoạn thiết kế là lựa chọn các mô hình thiết kế tốt nhất tương ứng với mỗi tiêu chí tối ưu hay tối ưu đa mục tiêu. Ngoài ra, vấn đề tối ưu trong giai đoạn thiết kết còn nhằm sớm đưa ra được các lựa chọn tốt về công nghệ, cấu hình và môi trường phát triển.
- **Tối ưu trong giai đoạn lập trình**: Sản phẩm chính tạo ra trong giai đoạn lập trình là mã nguồn phần mềm và mã thực thi nhưng việc tối ưu trong giai đoạn lập trình thường tập trung vào tối ưu mã nguồn. Hơn nữa, trong quá trình lập trình phần mềm nhúng phải thực hiện việc biên dịch chéo để tạo ra mã thực thi trên CPU máy đích khác với CPU máy chủ. Do đó, tối ưu mã nguồn phần mềm nhúng cũng được chia thành hai mức là tối ưu mã nguồn mức cao độc lập CPU đích và tối ưu mã hợp ngữ hướng đến các CPU cụ thể. Đồng thời, trong giai đoạn lập trình cũng có thể thực hiện tối ưu đơn mục tiêu như hiệu năng, bộ nhớ, năng lượng và chi phí hay tối ưu đa mục tiêu.
- **Tối ưu trong giai đoạn thực thi**: Trong giai đoạn thực thi, các chương trình đã được dịch sang mã máy hoặc mã máy ảo và được tích hợp trong các hệ thống nhúng. Do đó, tối ưu trong giai đoạn này thường khó thực hiện và ít được nghiên cứu. Tuy nhiên, do sự đa dạng về môi trường phần cứng nên tối ưu trong giai đoạn này có vai trò quan trọng trong tối ưu phần mềm nhúng. Tối ưu trong giai đoạn thực thi gồm ba hướng tiếp cận chính đó là tối ưu môi trường thực thi, tối ưu hướng dữ liệu và tối ưu mã thực thi hoặc mã máy ảo. Tối ưu môi trường thực thi thường được thực hiện trong các trình thông dịch với kỹ thuật biên dịch tạm và các kỹ thuật chuyên biệt hóa cũng như vấn đề lập lịch tiến trình. Tối ưu hướng dữ liệu tập trung chủ yếu vào nén dữ liệu và loại bỏ dư thừa các thông điệp trong mỗi giao thức truyền thông. Tối ưu mã thực thi bao gồm hai hướng tiếp cận chính là cải tiến các đoạn mã tự sửa để chuyên biệt hóa và lựa chọn cấu hình CPU.

1.2. Hiện trạng và thách thức

1.2.1. Hiện trạng và thách thức trong giai đoạn thiết kế

Trong phần này, chúng tôi tổng hợp và phân tích các nghiên cứu liên quan trong mỗi cách tiếp cận tối ưu trong giai đoạn thiết kế để làm cơ sở đưa ra các cải tiến cũng như đề xuất, triển khai các phương pháp tối ưu mới. Các nghiên cứu về tối ưu trong thời gian thiết kế được chia thành ba cách tiếp cận đó là tối ưu dựa trên mô phỏng, dựa trên kỹ

nghệ hiệu năng phần mềm (SPE – Software Performance Engineering) và dựa trên đánh giá trực tiếp từ các mô hình đặc tả phần mềm. *Cách tiếp cận tối ưu dựa trên SPE* nhằm tối ưu hiệu năng phần mềm dựa trên việc chuyển mô hình thiết kế thành mô hình hiệu năng và đánh giá trên mô hình hiệu năng [6, 10, 12, 13, 37, 79, 96]. Các nghiên cứu theo hướng tiếp cận này tập trung chủ yếu vào việc chuyển mô hình thiết kế về mô hình hiệu năng. Phần mềm nhúng được đặc tả kiến trúc theo ngôn ngữ mô hình hóa thống nhất (UML – Unified Modeling Language) có bổ sung thêm thông tin hiệu năng [7, 12, 17, 115, 116]. Các đặc tả này được chuyển sang mô hình hiệu năng để đánh giá. Các biểu đồ UML thường được sử dụng theo cách tiếp cận này như biểu đồ ca sử dụng, biểu đồ lớp, biểu đồ hoạt động, biểu đồ tuần tự, biểu đồ thành phần và biểu đồ triển khai. Các biểu đồ này sau đó được trích xuất thông tin để chuyển sang các biểu đồ hiệu năng như mạng hàng đợi, mạng hàng đợi phân tầng, mạng Petri, v.v. Các biểu đồ hiệu năng được sử dụng để đánh giá hiệu năng hệ thống và tối ưu [52, 63, 79, 95].

Cách tiếp cận tối ưu dựa trên mô phỏng cũng được quan tâm nghiên cứu nhiều trong giai đoạn thiết kế [25, 64, 85, 87, 106]. Ý tưởng chung của cách tiếp cận này là từ các mô hình phần mềm, sinh mã mô phỏng chức năng và thực thi mã mô phỏng để có thể lựa chọn kiến trúc hoặc đưa ra những quyết định tốt ngay trong thời gian thiết kế. Tuy nhiên, cách tiếp cận sinh mã mô phỏng dựa trên DSL còn ít được nghiên cứu, phát triển. Điển hình cho cách tiếp cận này là nghiên cứu của Thompson, C., White, J., Dougherty, B. và Schmidt, D. [102]. Trong nghiên cứu này, các tác giả xây dựng ngôn ngữ mô hình hóa DSL dựa trên khung làm việc nguồn mở Eclipse để xây dựng kiến trúc phần mềm trên điện thoại di động, sinh mã mô phỏng và thực hiện trên mã mô phỏng để đánh giá hiệu năng và cân bằng với thời gian sống của pin. Cũng theo hướng tiếp cận này, trong [5], Anastasopoulos, M. và Muthig, D. đã nghiên cứu tối ưu ứng dụng dựa trên kỹ nghệ hướng mô hình và mẫu sinh mã cho các dòng sản phẩm. Trong [41, 77, 95, 108, 110, 114], các tác giả tiếp cận theo hướng DSL để đặc tả mô hình thành phần và đặc tả môi trường cho các nền tảng như Android, J2ME, .NET Compact, v.v. và các giao thức truyền thông (http, https, SOAP, v.v.) để sinh mã mô phỏng chức năng tương ứng với môi trường đã chọn. Sau đó chạy mã mô phỏng trên môi trường thực tế để đánh giá và lựa chọn môi trường phát triển tốt nhất cho phần mềm trên điện thoai di động. Cách tiếp cận này hạn chế do mô hình thành phần mô tả kiến trúc phần mềm mức cao và chỉ biểu diễn được các khía cạnh tĩnh của phần mềm. Hơn nữa việc lựa chọn môi trường phát triển tốt nhất trong thời gian thiết kế có phạm vi ứng dụng hạn chế.

Cách tiếp cận tối ưu dựa trên đánh giá trực tiếp các mô hình phần mềm là một hướng nghiên cứu mới và tập trung chủ yếu vào các mô hình phần mềm hướng đối tượng. Trong các biểu đồ UML đặc tả phần mềm, biểu đồ lớp đặc tả khung nhìn kiến trúc lôgic

của phần mềm và được sử dụng thường xuyên nhất khi thiết kế phần mềm hướng đối tượng. Theo cách tiếp cận này, đầu tiên chúng ta cần xây dựng được các độ đo, các hàm đánh giá hiệu năng, bộ nhớ và các mục tiêu tối ưu khác dựa trên phân tích và trích xuất các tham số từ mô hình. Các mô hình tối ưu được lựa chọn căn cứ vào giá trị của các hàm đánh giá. Các độ đo đánh giá chất lượng phần mềm trong giai đoạn thiết kế dựa vào biểu đồ lớp đưa ra trong [78, 79, 112, 116] tập trung vào độ phức tạp kiến trúc, khả năng bảo trì, khả năng tái sử dụng, khả năng đóng gói. Tuy nhiên, hiện tại vẫn chưa có các độ đo và các hàm đánh giá để thực hiện tối ưu các mục tiêu quan trọng đối với phần mềm nhúng như hiệu năng, bộ nhớ và điện năng tiêu thụ. Trong nghiên cứu [4], tác giả đưa ra một số tham số như số lượng lớp, số lượng phương thức, số lượng thuộc tính và kiểu dữ liệu, v.v. để phân tích sự phụ thuộc hiệu năng theo các tham số này. Trong nghiên cứu này, tác giả cũng chưa đặc tả định lượng và xây dựng công thức đánh giá hiệu năng từ mô hình.

Trên cơ sở tình hình nghiên cứu đã phân tích ở trên, vấn đề tối ưu phần mềm nhúng trong giai đoạn thiết kế còn có các thách thức như sau:

- Khó khăn khi đánh giá hiệu năng, bộ nhớ của phần mềm nhúng mức mô hình. Việc đánh giá các mô hình phần mềm nhúng chủ yếu tập trung vào các độ đo chất lượng phần mềm như tính linh động, tính bao gói, tính tái sử dụng và tính dễ bảo trì.
- Chưa có các độ đo đánh giá về hiệu năng và bộ nhớ từ các mô hình thiết kế. Hiện tại chỉ có một số nghiên cứu đưa ra ý tưởng và phân tích các tham số trong mô hình tác động đến hiệu năng mà chưa có độ đo và công thức cụ thể.
- Chưa có mô hình chung và nền tảng lý thuyết về tối ưu trong giai đoạn thiết kế và chỉ có một số nghiên cứu nhằm đưa ra các quyết định tối ưu trong thời gian thiết kế như lựa chọn môi trường, giao thức dựa trên mô phỏng và chưa có nghiên cứu về tối ưu theo hướng tiếp cận dựa trên đánh giá trực tiếp mô hình. Chỉ có một số ít nghiên cứu đánh giá hiệu năng mức mô hình bằng cách chuyển các mô hình đặc tả thành mô hình hiệu năng.
- Chưa có nghiên cứu về tối ưu đa mục tiêu dựa trên biểu đồ lớp. Có nhiều mục tiêu và các ràng buộc trong tối ưu phần mềm nhúng. Các tiêu chí tối ưu thường không được thỏa mãn đồng thời và có thể mâu thuẫn do đó cần phải thực hiện phương pháp tối ưu đa mục tiêu.

1.2.2. Hiện trạng và thách thức trong giai đoạn lập trình

Theo mô hình tối ưu chung đã xây dựng, có hai mức tối ưu phần mềm nhúng trong giai đoạn lập trình đó là tối ưu mã nguồn mức cao độc lập kiến trúc đích và tối ưu mã hợp ngữ cho các CPU nhúng. *Các phương pháp tối ưu mã nguồn mức cao, độc lập CPU* cho phần mềm nhúng cũng tương tự như tối ưu phần mềm thông thường. Các phương pháp này đã được nghiên cứu từ thập niên 80 và được thể hiện trong các trình biên dịch. Các tiêu chí tối ưu phổ biến trên mã nguồn mức cao như giảm mức tiêu thụ điện năng, tối ưu

hiệu năng và tối ưu kích thước mã nguồn. Tối ưu mã nguồn để giảm tiêu thụ điện năng trong các nghiên cứu [22, 69, 82, 111] bao gồm các kỹ thuật chính như biến đổi mã nguồn để giảm truy xuất bộ nhớ và các biến đổi trên vòng lặp. Các kỹ thuật tối ưu kích thước mã nguồn cũng được nghiên cứu rộng rãi và có ý nghĩa đặc biệt quan trọng đối với phần mềm nhúng. Trong các nghiên cứu [7, 26, 29, 44, 48, 75, 83], các tác giả đã trình bày một số kỹ thuật tối ưu kích thước mã nguồn điển hình như loại bỏ mã chết, thay thế mã và cấp phát bộ nhớ động. Trong tối ưu mã nguồn có nhiều mục tiêu tối ưu và các mục tiêu tối ưu thường mâu thuẫn nhau. Do đó, vấn đề tối ưu đa mục tiêu cũng được đề xuất và triển khai. Điển hình như trong nghiên cứu [27], tác giả đã đề xuất phương pháp tối ưu đa mục tiêu cho mã nguồn phần mềm dựa trên nguyên lý Pareto.

Các phương pháp tối ưu mã hợp ngữ hướng đến các CPU đích còn ít được nghiên cứu và áp dụng cho phần mềm thông thường nhưng có ý nghĩa đặc biệt quan trọng đối với phần mềm nhúng. Mỗi phần mềm nhúng thường được phát triển cho một loại CPU. Các loại CPU có kiến trúc, tập thanh ghi và tập lệnh hợp ngữ khác nhau. Hơn nữa, với mỗi loại CPU khác nhau có thể có các phương pháp tối ưu khác nhau. Tối ưu phần mềm nhúng mức mã hợp ngữ tập trung vào các tiêu chí tối ưu chính là hiệu năng, năng lượng, chi phí và kích thước chương trình [19, 28, 30, 59, 87, 88].

Mặc dù đã được nghiên cứu rộng rãi nhưng vấn đề tối ưu phần mềm nhúng trong giai đoạn lập trình vẫn còn các thách thức sau:

- Các phương pháp tối ưu mã nguồn độc lập kiến trúc CPU đích đã được nghiên cứu và triển khai trong các trình biên dịch tuy nhiên vẫn chưa giải quyết việc phân tích mã nguồn để tìm ra các đoạn mã ảnh hưởng đến hiệu năng hoặc tiêu tốn năng lượng nhất. Chưa áp dụng luật Pareto 80/20 để tăng hiệu quả và rút ngắn thời gian tối ưu trong các chương trình dịch.
- Phương pháp tối ưu mã nguồn mức cao dựa trên loại bỏ các biểu thức con chung tuy đã được tích hợp trong các trình biên dịch nhưng vẫn chưa xét đến các biểu thức tương đương.
- Phương pháp tối ưu đa mục tiêu chưa được nghiên cứu và áp dụng trong các chương trình dịch.
- Tối ưu mức mã hợp ngữ gặp nhiều khó khăn trong quá trình xây dựng phần mềm nhúng do phải biên dịch chéo hướng đến các CPU đích khác nhau. Tối ưu mã hợp ngữ cho các loại CPU khác nhau cũng có các đặc trưng riêng.
- Khó khăn khi đánh giá mức tiêu thụ điện năng, hiệu năng cho mỗi chương trình hợp ngữ đặc biệt là với kiến trúc đường ống lệnh và kiến trúc siêu vô hướng.
- Tối ưu mức mã hợp ngữ cũng gặp khó khăn và chưa được giải quyết trong các hệ thống đa CPU, đặc biệt là với các hệ thống không đồng nhất.

1.2.3. Hiện trạng và thách thức trong giai đoạn thực thi

Tối ưu trong giai đoạn thực thi được chia thành ba nhóm chính đó là tối ưu môi trường thực thi, tối ưu hướng dữ liệu và tối ưu mã thực thi. Phần lớn các nghiên cứu trong gian đoạn này tập trung vào tối ưu môi trường thực thi. Môi trường thực thi có thể chia thành các loại chính là hệ điều hành, môi trường máy ảo, môi trường thông dịch mã nguồn sang mã máy và môi trường phần cứng.

Trong mức hệ điều hành, hầu hết các nghiên cứu tập trung vào các phương pháp lập lịch tiến trình để tối ưu hiệu năng hay điện năng tiêu thụ của các tiến trình ưu tiên. Điển hình như trong nghiên cứu [18], Cheung, T. L. và cộng sự đã xây dựng nền tảng hỗ trợ mô hình quyết định Markov để tối ưu tài nguyên cho các ứng dụng đàm thoại. Trong môi trường máy ảo, mã máy ảo cần được thông dịch sang mã máy đích để thực thi và mã nguồn của chương trình đã được biên dịch sang mã máy ảo trước khi thực thi trên máy ảo. Do đó các phương pháp tối ưu chương trình trong môi trường máy ảo tập trung chủ yếu vào tối ưu mức trình thông dịch. Kỹ thuật tối ưu phổ biến là kỹ thuật biên dịch tạm (JIT – Just In Time) như trong nghiên cứu [104]. Trong kỹ thuật này, thay vì dịch và thực thi từng câu lệnh, trình thông dịch sẽ biên dịch các đoạn lệnh và lưu trữ mã máy đích trong bộ đệm để thực thi. Kỹ thuật này đã được triển khai trong hai máy ảo phổ biến nhất là máy ảo Java và máy ảo .NET. Ngoài ra các kỹ thuật cấp phát và quản lý bộ nhớ động nhằm cải tiến hiệu quả sử dụng bộ nhớ cũng được tích hợp trong các môi trường máy ảo [68, 90]. Trong môi trường thông dịch trực tiếp từ mã nguồn sang mã máy như máy chủ web PHP, ngoài các kỹ thuật biên dịch tạm, quản lý động bộ nhớ, các kỹ thuật tối ưu mã nguồn cũng có thể được áp dụng do môi trường này thông dịch từ mã nguồn để thực thi. Tuy nhiên tối ưu mã nguồn trong các trình thông dịch thường mang tính đặc thù so với trong môi trường biên dịch tĩnh [39, 57, 67, 90] như tối ưu động và có thể tái cấu trúc mã nguồn. Ngoài ra, trong giai đoạn thực thi cũng có thể thực hiện các kỹ thuật tối ưu dựa trên cải tiến môi trường phần cứng hoặc kết hợp phần cứng và phần mềm.

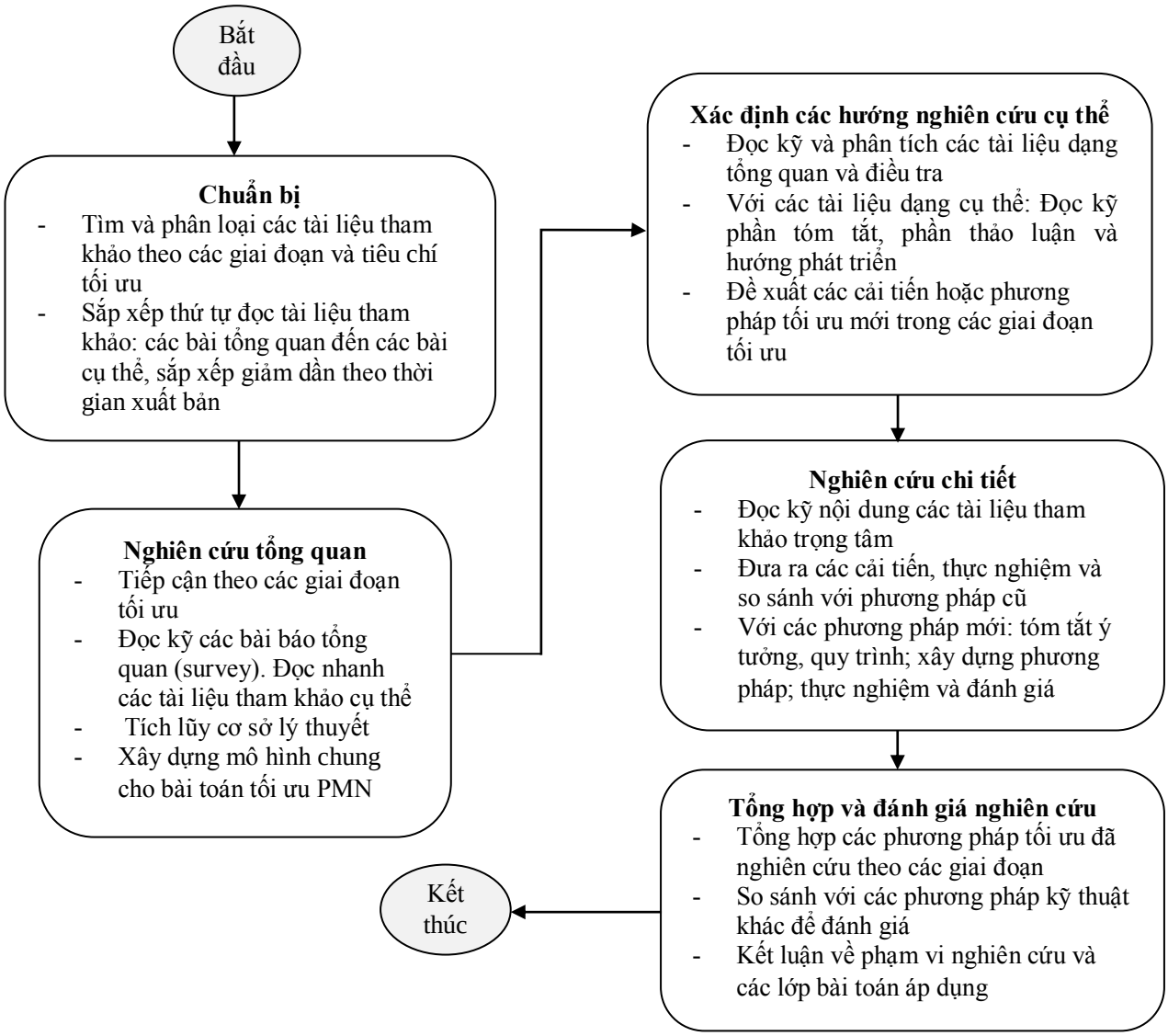
Bên cạnh tối ưu môi trường thực thi, các nghiên cứu tối ưu trong giai đoạn này còn tập trung theo hướng tối ưu mã thực thi. Phổ biến nhất là các nghiên cứu tối ưu dựa trên chuyên biệt hóa về mã nguồn và thuật toán. Ý tưởng chính của chuyên biệt hóa là tùy theo mỗi môi trường thực thi khác nhau mà thực thi các đoạn mã chuyên biệt nhằm cải tiến hiệu năng, năng lượng và bộ nhớ. Các nghiên cứu điển hình theo hướng tiếp cận này như [91, 113, 117]. Ngoài ra, một kỹ thuật tối ưu trong hướng tiếp cận này cũng được nghiên cứu phổ biến là kỹ thuật mã tự sửa như trong nghiên cứu trong [91].

Hướng tiếp cận tối ưu thứ ba trong giai đoạn thực thi nhằm tối ưu dữ liệu. Tối ưu dữ liệu bao gồm tối ưu trong xử lý dữ liệu và tối ưu trong truyền dữ liệu. Các phương pháp đặc tả dữ liệu để tối ưu hiệu năng tiêu biểu như [57]. Ý tưởng chính của phương

pháp này là phân nhóm, đặc tả dữ liệu theo các miền ứng dụng cụ thể, mỗi loại dữ liệu được thực thi bằng các đoạn mã tương ứng. Các phương pháp tối ưu môi trường truyền dữ liệu nhằm cải tiến thời gian truyền thông cho các ứng dụng phân tán. Hai phương pháp chính theo hướng tiếp cận này là nén dữ liệu và loại bỏ dư thừa trong các thông điệp [74].

Căn cứ vào tình hình nghiên cứu trên đây, các thách thức đặt ra khi tối ưu phần mềm nhúng trong giai đoạn thực thi có thể được tổng hợp như sau:

- Khó áp dụng các kỹ thuật tối ưu trên mã thực thi. Việc đánh giá các trạng thái động để chuyên biệt mã gặp khó khăn.
- Khi tối ưu môi trường truyền thông cần phải đánh giá thời gian truyền dữ liệu và thời gian thực hiện các xử lý tối ưu trên thiết bị.
- Chưa có nghiên cứu về tối ưu dựa trên kỹ nghệ ngược cũng như việc kết hợp cả phần cứng và phần mềm để cải tiến môi trường thực thi.



Hình 1.5: Quy trình nghiên cứu và triển khai trong luận án

1.3. Phương pháp và nội dung nghiên cứu

1.3.1. Phương pháp nghiên cứu

Để thực hiện các nội dung nghiên cứu trong luận án, chúng tôi tiến hành nghiên cứu theo các phương pháp chính đó là tiếp cận từ trên xuống, phân loại và hệ thống hóa lý thuyết, phân tích và tổng hợp lý thuyết. Theo phương pháp tiếp cận từ trên xuống, chúng tôi điều tra, tổng hợp và nghiên cứu tổng thể sau đó nghiên cứu chuyên sâu từng phần. Theo phương pháp phân loại và hệ thống hóa lý thuyết, chúng tôi thu thập các nghiên cứu về tối ưu phần mềm nhúng, phân loại và xây dựng mô hình chung cho bài toán tối ưu. Theo phương pháp phân tích và tổng hợp lý thuyết, với mỗi tiêu chí tối ưu, chúng tôi tổng hợp các nghiên cứu liên quan để đề xuất cải tiến hoặc phương pháp tối ưu mới.

Áp dụng các phương pháp trên, luận án được thực hiện theo quy trình trong Hình 1.5. Đầu tiên, chúng tôi thực hiện điều tra, tổng hợp và so sánh các nghiên cứu hiện có về tối ưu phần mềm nhúng. Trên cơ sở các nghiên cứu tiền đề này, chúng tôi hệ thống hóa và xây dựng bức tranh chung về tối ưu phần mềm nhúng. Dựa trên đó, chúng tôi giới hạn phạm vi nghiên cứu, đề xuất và phát triển một số phương pháp tối ưu trong các giai đoạn phát triển phần mềm nhúng như thiết kế, lập trình, thực thi theo các tiêu chí tối ưu như hiệu năng, mức tiêu thụ điện năng, chi phí, v.v. và tối ưu đa mục tiêu. Cuối cùng chúng tôi tiến hành thực nghiệm để đánh giá phương pháp theo các chương trình như nhận dạng chữ Nôm trên điện thoại di động, bộ chương trình nhúng cho *Bo mạch Netduino* và *Netduino Plus*, tháp Hà Nội, 8 quân Hậu và các chương trình nhúng cho vi xử lý MIPS chạy trên phần mềm mô phỏng *SimpleScalar*.

1.3.2. Nội dung nghiên cứu

Theo mô hình chung về tối ưu trong phát triển phần mềm nhúng đã đưa ra trong Hình 1.4 và căn cứ vào tình hình nghiên cứu hiện tại, chúng tôi tập trung nghiên cứu chủ yếu về các phương pháp tối ưu phần mềm nhúng theo quy trình phát triển trong kỹ nghệ xuôi. Mục tiêu của luận án nhằm nghiên cứu, cải tiến, đề xuất và triển khai một số phương pháp tối ưu trong các giai đoạn phát triển phần mềm nhúng để làm nền tảng giải quyết bài toán tối ưu tổng thể. Tối ưu và tối ưu hóa phần mềm nhúng là bài toán phức tạp, có phạm vi nghiên cứu rộng và liên quan nhiều đến phần cứng. Tối ưu hóa kết hợp với kỹ nghệ ngược có ý nghĩa thực tiễn và phạm vi ứng dụng rộng rãi đối với các hệ thống nhúng, phần mềm nhúng đã tồn tại. Tuy nhiên để có thể thực hiện tối ưu hóa, trước hết cần có các phương pháp tối ưu cụ thể trong kỹ nghệ xuôi.

Trong mỗi giai đoạn tối ưu, chúng tôi hệ thống hóa, phân nhóm và đánh giá các nghiên cứu liên quan làm cơ sở lý thuyết để đưa ra một số cải tiến cho các phương pháp đã có cũng như đề xuất và phát triển một số phương pháp tối ưu mới nhằm góp phần giải quyết các thách thức đặt ra như mô tả trong phần trước. Theo đó, phạm vi nghiên cứu của luận án 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, phát triển phương pháp tối ưu điện năng tiêu thụ dựa trên tái cấu hình CPU và kỹ nghệ ngược.

Trong các nội dung nghiên cứu trên, các phương pháp tối ưu trong giai đoạn thiết kế và tối ưu mã nguồn mức cao độc lập CPU được giới hạn nghiên cứu, thực nghiệm cho các hệ thống theo kiến trúc Von Neuman. Các phương pháp tối ưu dựa trên lập lịch các lệnh hợp ngữ được nghiên cứu đến mức kiến trúc CPU với cả ba kiểu kiến trúc là đơn lệnh, đường ống lệnh và siêu vô hướng trong phạm vi các lệnh đơn chu trình.

1.4. Tổng kết chương

Tối ưu phần mềm nhúng là bài toán phức tạp, có phạm vi nghiên cứu rộng, bao gồm nhiều tiêu chí tối ưu và có thể tiến hành trong các giai đoạn khác nhau. Các tiêu chí tối ưu bao gồm tối ưu hiệu năng, tối ưu bộ nhớ, tối ưu điện năng tiêu thụ, tối ưu chi phí, v.v. và tối ưu đa mục tiêu. Trong mỗi giai đoạn phát triển phần mềm nhúng, chúng ta có thể nghiên cứu, thực nghiệm chuyên sâu theo các tiêu chí tối ưu khác nhau.

Căn cứ vào mô hình chung cho bài toán tối ưu phần mềm nhúng, trong mỗi giai đoạn phát triển, chúng tôi phân tích tình hình nghiên cứu hiện tại, xác định các vấn đề chưa giải quyết để đưa ra nội dung nghiên cứu và thực nghiệm trong luận án. Theo đó, luận án sẽ tập trung nghiên cứu các phương pháp tối ưu trong kỹ nghệ xuôi. Phát triển phần mềm trong kỹ nghệ xuôi gồm ba giai đoạn chính là thiết kế, lập trình và thực thi. Trong giai đoạn thiết kế, mục tiêu của bài toán tối ưu nhằm tìm ra các biểu đồ thiết kế tốt, các quyết định tối ưu khi lựa chọn công nghệ, môi trường, v.v. Trong giai đoạn lập trình, bài toán tối ưu có thể tiến hành theo hai mức: tối ưu mã nguồn mức cao độc lập CPU – dùng cho cả phần mềm nhúng và phần mềm thông thường, tối ưu mã hợp ngữ hướng đến các CPU đích – mang đặc thù của phần mềm nhúng. Trong giai đoạn thực thi, bài toán tối ưu có thể được tiến hành theo ba khía cạnh là môi trường thực thi, dữ liệu và mã thực thi.

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

Để triển khai các nội dung nghiên cứu theo cách tiếp cận chi tiết hóa mô hình tối ưu chung, trong chương này, chúng tôi trình bày một số phương pháp tối ưu phần mềm nhúng trong giai đoạn thiết kế. 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 có các mục tiêu tối ưu mang tính đặc thù như tính tin cậy, tính linh động, 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. Kết quả tối ưu phần mềm nhúng trong giai đoạn thiết kế là đạt được các mô hình phần mềm tốt và sớm đưa ra được các lựa chọn như môi trường, công nghệ, thư viện và nền tảng phát triển. 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.

Căn cứ vào hiện trạng nghiên cứu và các thách thức đặt ra đã được trình bày trong *Mục 1.2.1 Chương 1*, trong giai đoạn thiết kế, chúng tôi đề xuất và triển khai ba phương pháp tối ưu mới đó là tối ưu hiệu năng phần mềm nhúng dựa trên đánh giá biểu đồ lớp, tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô, DSL và T4, tối ưu đa mục tiêu dựa trên biểu đồ lớp. Đồng thời, chúng tôi cũng cải tiến phương pháp tối ưu dựa trên chuyển đổi mô hình [7, 8] để tối ưu hiệu năng và bộ nhớ. Cấu trúc của chương như sau: *Mục 2.1* trình bày về các phương pháp tối ưu hiệu năng; *Mục 2.2* trình bày về các phương pháp tối ưu bộ nhớ; *Mục 2.3* trình bày về tối ưu đa mục tiêu; *Mục 2.4* tổng kết các nội dung đã trình bày.

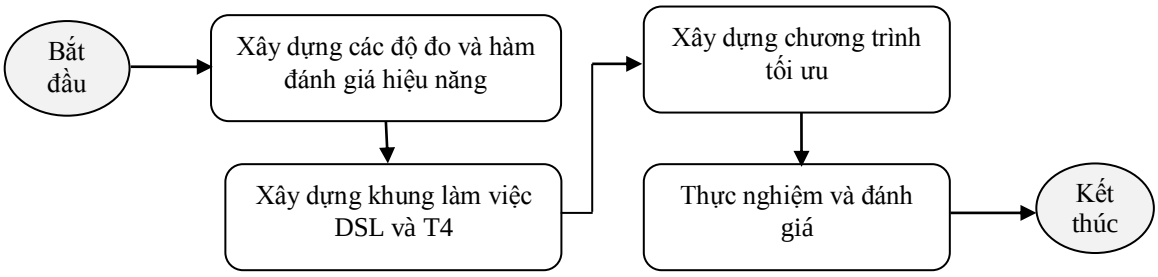
2.1. Tối ưu hiệu năng trong giai đoạn thiết kế

Thuật ngữ "hiệu năng" có hàm ý rộng; tùy theo các lĩnh vực mà có ý nghĩa khác nhau. Trong phát triển phần cứng hiệu năng có thể bao gồm tốc độ xử lý của CPU, tốc độ truy xuất bộ nhớ. Trong phát triển phần mềm, hiệu năng có thể bao gồm thời thực thi chương trình, thời gian phân tích lỗi, thời gian biên dịch, v.v. Khái niệm hiệu năng chúng tôi sử dụng trong các phương pháp này dựa trên số lần thao tác bộ nhớ. Theo đó, hàm đánh giá hiệu năng có giá trị nhỏ hơn tương ứng với hiệu năng tốt hơn.

2.1.1. Tối ưu hiệu năng dựa trên biểu đồ lớp

i. Ý tưởng và quy trình nghiên cứu

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. Phương pháp tối ưu này được nghiên cứu và thực nghiệm theo quy trình trong Hình 2.1. Theo quy trình này, đầu tiên chúng tôi xây dựng các độ đo ảnh hưởng đến hiệu năng và hàm đánh giá hiệu năng. Sau đó chúng tôi xây dựng khung làm việc DSL và T4 để thiết kế và sinh đặc tả dạng văn bản tự động từ mô hình. Trong bước tiếp theo chúng tôi xây dựng chương trình tối ưu dựa trên hàm đánh giá hiệu năng để chọn mô hình tốt. Cuối cùng chúng tôi tiến hành thực nghiệm và đánh giá.



Hình 2.1: Quy trình nghiên cứu và thực nghiệm tối ưu dựa trên 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 phương thức thứ *j*
Kiểu trả về	r_e^j	Kiểu dữ liệu trả về từ phương thức thứ *j*

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

- **Các tham số từ biểu đồ lớp**

Để xây dựng hàm đánh giá hiệu năng trên biểu đồ lớp, đầu tiên chúng tôi dẫn xuất các tham số ảnh hưởng đến hiệu năng trực tiếp từ biểu đồ lớp. Các tham số này được mô tả cụ thể trong Bảng 2.1.

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

Trước khi xây dựng hàm đánh giá hiệu năng từ biểu đồ lớp, chúng tôi phân tích các thành phần trong lớp, cấu trúc của lớp và biểu đồ lớp để xây dựng các công thức cho các độ đo ảnh hưởng đến hiệu năng trong Bảng 2.2. Trong các độ đo này, hàm size() là hàm lấy kích thước của kiểu dữ liệu tương ứng với mỗi thành phần.

Bảng 2.2. Các độ đo ảnh hưởng đến hiệu năng

Độ đo	Ký hiệu	Mô tả
Kích thước các biến tĩnh	S_1	Tổng kích thước các thuộc tính tĩnh trong một biểu đồ lớp
Kích thước các phương thức tĩnh	S_2	Tổng kích thước của các phương thức tĩnh trong một biểu đồ lớp
Kích thước thực thi các phương thức tĩnh	S_3	Tổng kích thước bộ nhớ sử dụng khi thực thi các phương thức tĩnh (đánh giá theo mô hình)
Kích thước các biến đối tượng	S_4	Tổng kích thước các thuộc tính đối tượng
Kích thước các phương thức đối tượng	S_5	Tổng kích thước các phương thức đối tượng
Kích thước thực thi các phương thức đối tượng	S_6	Tổng kích thước bộ nhớ sử dụng khi thực thi các phương thức đối tượng (đánh giá theo mô hình)

Kích thước các biến tĩnh

Độ đo này được tính bằng tổng số dung lượng bộ nhớ được cấp phát tĩnh cho mọi thuộc tính tĩnh của mọi lớp trong biểu đồ. Các thành phần tĩnh được cấp phát bộ nhớ ngay khi nạp chương trình vào bộ nhớ. Với các ký hiệu trong Bảng 2.1 và Bảng 2.2, kích thước các thuộc tính tĩnh trong một biểu đồ lớp được tính theo công thức (2.1).

$$S_1 = \sum_{i=1}^{|C|} \sum_{j=1}^{|V_s^i|} \text{size}(a_s^j) \tag{2.1}$$

Kích thước các phương thức tĩnh

Các phương thức tĩnh không thuộc đối tượng cụ thể nào, thường được gọi thông qua tên lớp và được cấp phát bộ nhớ ngay khi nạp chương trình. Theo các tham số trong Bảng 2.1 và Bảng 2.2, độ đo này được tính theo công thức (2.2).

$$S_2 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_S^i|} \text{size}(v_r^{\ j}) \tag{2.2}$$

Kích thước thực thi các phương thức tĩnh

Độ đo này là kích thước bộ nhớ sử dụng khi thực thi một phương thức tĩnh. Khi gọi thực thi một phương thức tĩnh, đầu tiên cần cấp phát bộ nhớ để lưu trữ các tham số và sau khi thực hiện xong cần lưu lại kết quả trả về trong bộ nhớ. Do đó, theo các tham số trong Bảng 2.1 và Bảng 2.2, độ đo này được định nghĩa như công thức (2.3).

$$S_3 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_S^i|} (\text{size}(r_e^{\ j}) + \sum_{k=1}^{|P_m^j|} \text{size}(p_k)) \tag{2.3}$$

Kích thước các biến đối tượng

Các biến đối tượng chỉ được cấp phát bộ nhớ khi đối tượng được tạo. Các biến đối tượng được truy xuất thông qua tham chiếu đến đối tượng. Độ đo này là tổng kích thước của mọi biến đối tượng trong các lớp. Từ các tham số trong Bảng 2.1 và Bảng 2.2, độ đo này được định nghĩa như công thức (2.4).

$$S_4 = \sum_{i=1}^{|C|} \sum_{j=1}^{|V_O^i|} \text{size}(a_o^{\ j}) \tag{2.4}$$

Kích thước các phương thức đối tượng

Các phương thức đối tượng được cấp phát bộ nhớ khi tạo đối tượng và chỉ được sử dụng khi đối tượng đã tồn tại. Do đó, theo các tham số trong Bảng 2.1 và Bảng 2.2, độ đo này được định nghĩa như công thức (2.5).

$$S_5 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_O^i|} \text{size}(v_r^{\ j}) \tag{2.5}$$

Kích thước thực thi các phương thức đối tượng

Kích thước thực thi các phương thức đối tượng là tổng kích thước bộ nhớ cần thiết cấp phát thêm để chứa các tham số và kết quả trả về của phương thức đối tượng. Theo các tham số trong Bảng 2.1 và Bảng 2.2, độ đo này được định nghĩa như công thức (2.6).

Trong công thức (2.6), tham số ngầm định (con trỏ *this*) cũng được bổ sung vào danh sách tham số $P^j{}_m$ của mỗi phương thức đối tượng.

$$S_6 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_O^i|} (\text{size}(r_e{}^j) + \sum_{k=1}^{|P_m^j|} \text{size}(p_k)) \quad (2.6)$$

- **Hàm đánh giá hiệu năng**

Hàm đánh giá hiệu năng được xây dựng để đánh giá hiệu năng của phần mềm từ biểu đồ lớp. Chúng tôi xây dựng hàm đánh giá hiệu năng f_p theo các độ đo tĩnh như trong công thức (2.7). Trong công thức (2.7), α, β, γ và ε là các hệ số phụ thuộc của hiệu năng vào các độ đo tương ứng và S_1 đến S_6.

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

Mục đích của hàm đánh giá hiệu năng không sử dụng để ước lượng hiệu năng mà nhằm đánh giá biểu đồ lớp nào có hiệu năng tốt hơn. Hơn nữa, trong giai đoạn thiết kế, chúng ta không thể đánh giá hiệu năng chính xác tuyệt đối vì chưa có mã nguồn và chưa thực thi chương trình. Do đó để xác định các hệ số phụ thuộc từ biểu đồ lớp, chúng tôi giả thiết mỗi thao tác cấp phát bộ nhớ tĩnh hoặc động và thao tác truy xuất bộ nhớ có thời gian như nhau và được tính bằng một thao tác bộ nhớ. Giá trị các hệ số phụ thuộc được xác định dựa trên phân tích sự ảnh hưởng của quá trình cấp phát và truy xuất bộ nhớ đến thời gian thực thi của một chương trình hướng đối tượng.

Hàm đánh giá hiệu năng trong nghiên cứu này được sử dụng để đánh giá số lần thao tác bộ nhớ. Do đó, hàm đánh giá hiệu năng có giá trị càng nhỏ càng tốt. Khi chương trình được yêu cầu thực thi, đầu tiên mã nguồn các lớp sẽ được nạp vào bộ nhớ, các biến tĩnh và các phương thức tĩnh cũng được cấp phát bộ nhớ trong thời điểm nạp. Do các thành phần tĩnh chỉ mất một lần cấp phát tĩnh và một lần truy xuất bộ nhớ để sử dụng sau khi nạp nên $\alpha = 2$.

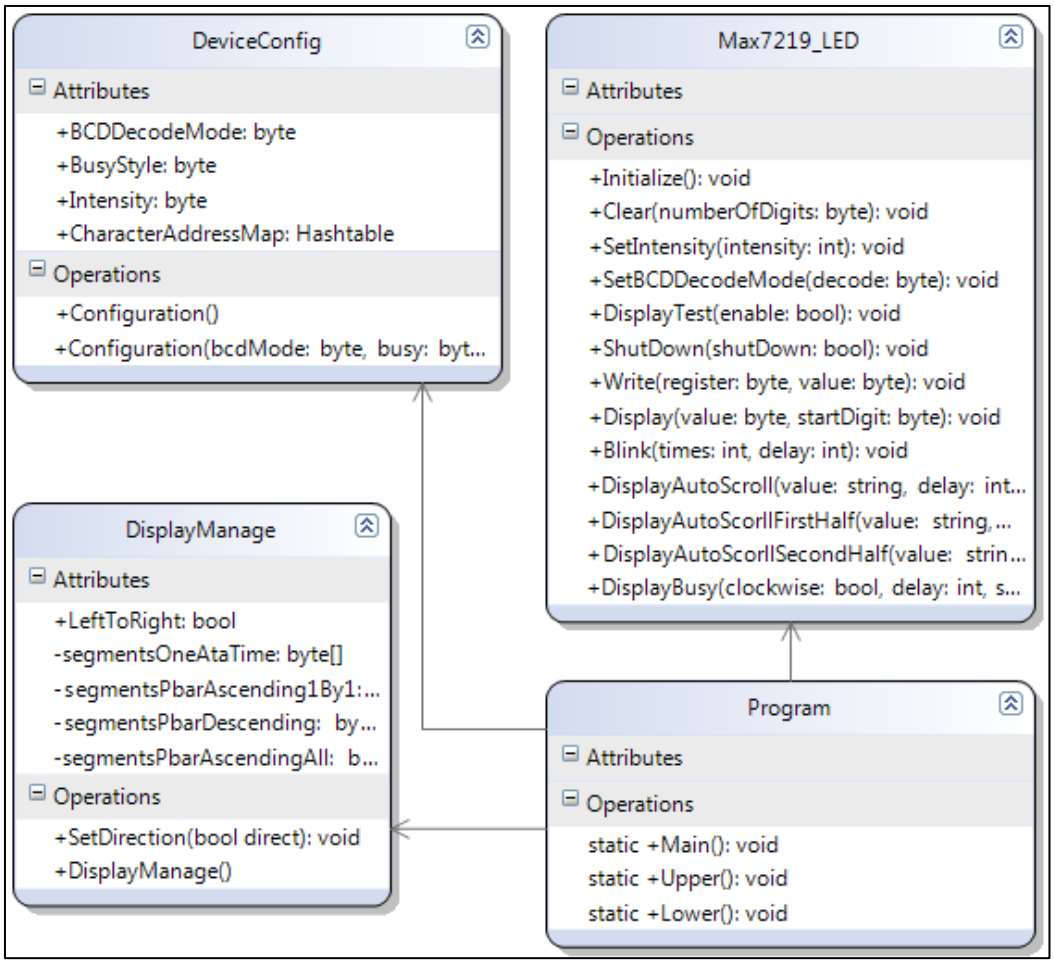
Khi sử dụng các biến đối tượng và các phương thức đối tượng cần tạo và cấp phát bộ nhớ cho đối tượng trước. Khi tạo đối tượng, cần thực hiện hai bước là truy xuất bộ nhớ để thực hiện câu lệnh tạo đối tượng và cấp phát động bộ nhớ cho các biến và phương thức đối tượng. Do đó cần một thao tác cấp phát động và hai thao tác truy xuất bộ nhớ để có thể sử dụng các thành phần thuộc đối tượng nên có thể thiết lập $\gamma = 3$.

Khi thực thi một phương thức tĩnh, cần một thao tác truy xuất bộ nhớ để trỏ đến tập lệnh của phương thức, một lần cấp phát bộ nhớ cho các tham số, tối thiểu một thao truy xuất bộ nhớ đã cấp phát cho tham số và sau khi thực hiện cần một lần truy xuất bộ nhớ để tham chiếu đến dữ liệu trả về. Theo đó, quá trình thực thi một phương thức tĩnh cần ít nhất một thao tác cấp phát và ba thao tác truy xuất bộ nhớ. Suy ra có thể gán $\beta = 4$.

Khi thực thi một phương thức đối tượng, cần phải tạo đối tượng trước rồi mới gọi thông qua biến tham chiếu đối tượng nên cần một thao tác cấp phát động và hai thao tác truy xuất bộ nhớ. Sau khi đã tạo đối tượng, quá trình thực thi phương thức đối tượng cũng giống phương thức tĩnh nên cũng cần thêm ít nhất một thao tác cấp phát và ba thao tác truy xuất bộ nhớ. Như vậy toàn bộ quá trình thực hiện của phương thức đối tượng cần ít nhất hai thao tác cấp phát và năm thao tác truy xuất bộ nhớ nên có thể thiết lập $\varepsilon = 7$.

iii. Ví dụ minh họa các độ đo và hàm đánh giá

Để minh họa rõ ràng hơn về các độ đo và hàm đánh giá đã trình bày trong các phần trước, chúng tôi phân tích một biểu đồ lớp của chương trình *Netduino_8digit* điều khiển điốt phát quang (LED – Light Emitting Diode) 8 số trên *Bo mạch Netduino Plus*. Đây là một trong các chương trình nhúng được sử dụng trong quá trình thực nghiệm; được tổng hợp và trình bày trong *Phụ lục P.2.5.1*. Biểu đồ lớp của chương trình được thể hiện như trong Hình 2.2. Từ đặc tả dạng văn bản của biểu đồ lớp, chương trình tối ưu sẽ tự động trích xuất các tham số, tính các độ đo và tính giá trị của hàm đánh giá hiệu năng. Kết quả được chỉ ra trong Hình 2.3.



Hình 2.2: Một biểu đồ lớp của chương trình *Netduino_8digit*

Hình 2.3: Kết quả tính các độ đo và hàm đánh giá hiệu năng

iv. Phát triển khung làm việc DSL và T4

Sau khi xây dựng hàm đánh giá hiệu năng, để triển khai thực nghiệm chúng tôi xây dựng khung làm việc DSL và T4. Mục đích xây dựng khung làm việc này để thiết kế các biểu đồ lớp trong giao diện đồ họa và sinh đặc tả dạng văn bản từ mô hình một cách tự động. Khung làm việc này đã được chúng tôi xây dựng như mô tả trong *Phụ lục P.1.1*. Một phần mềm nhúng có thể được thiết kế theo các biểu đồ lớp khác nhau trong khung làm việc này. Từ khung làm việc, dựa trên mẫu T4, mỗi biểu đồ sẽ tự động sinh ra đặc tả dạng văn bản để làm đầu vào cho chương trình tối ưu.

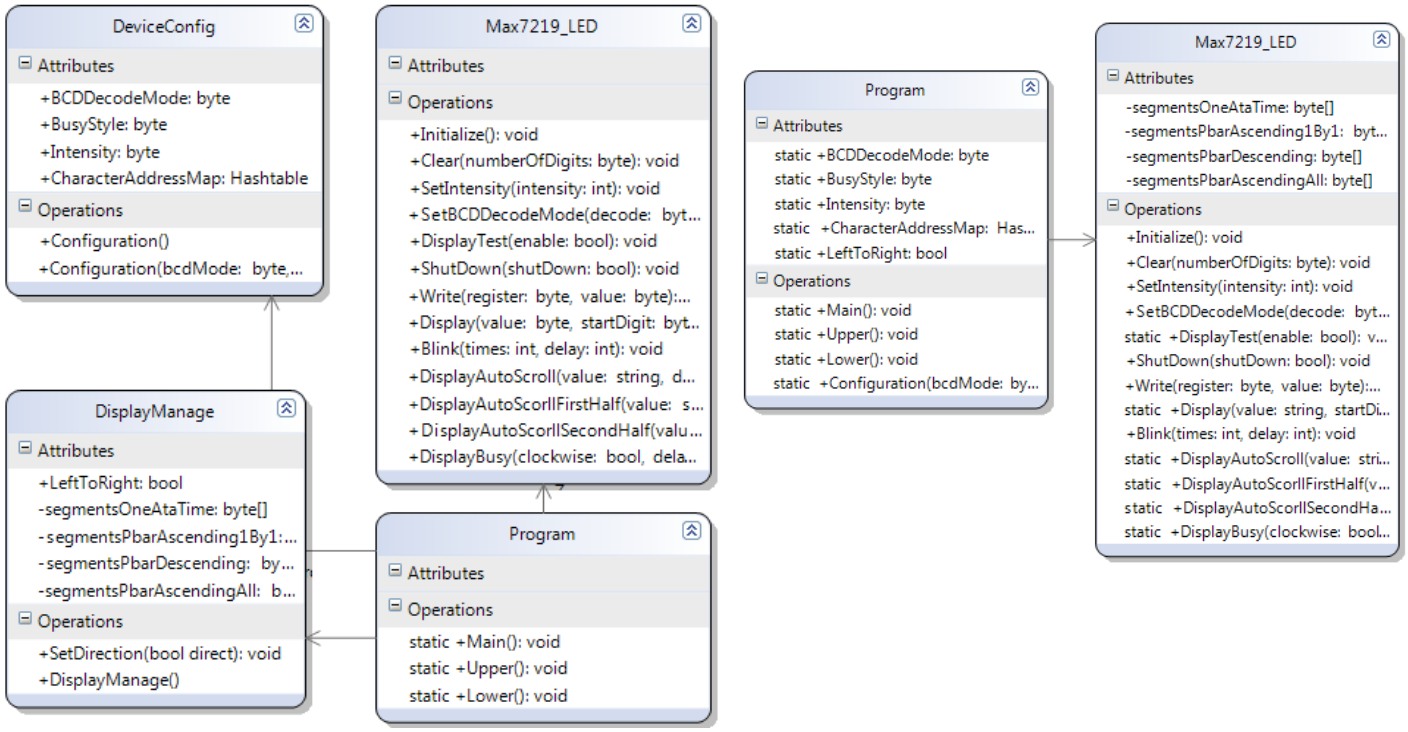
v. Xây dựng chương trình tối ưu

Để thực hiện phương pháp tối ưu hiệu năng phần mềm nhúng dựa trên đánh giá trực tiếp biểu đồ lớp, chúng tôi đã xây dựng chương trình tối ưu như mô tả chi tiết trong *Phụ lục P.1.2*. Chương trình nhận các đặc tả dạng văn bản của tập các mô hình đã được thiết kế trong khung làm việc DSL và T4 để phân tích các tham số. Sau khi phân tích tham số, chương trình sẽ tính các độ đo và hàm đánh giá hiệu năng trên mỗi đặc tả. Sau đó chương trình sẽ dựa trên giá trị của hàm đánh giá hiệu năng để lựa chọn biểu đồ tốt nhất và vẽ biểu đồ hiệu năng của các mô hình.

vi. Thực nghiệm

- **Thực nghiệm kiểm chứng hàm đánh giá hiệu năng**

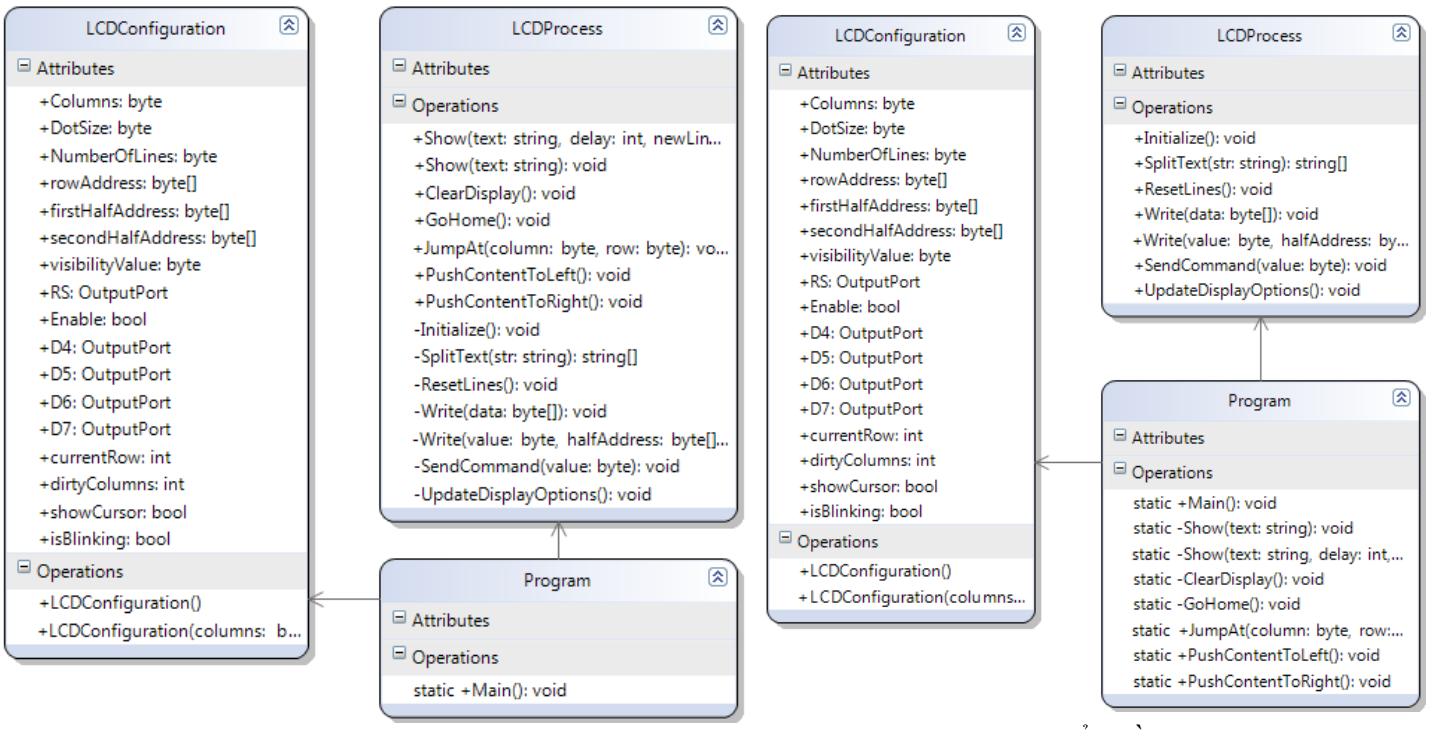
Trong thực nghiệm này, chúng tôi sử dụng ba chương trình nhúng cho *Bo mạch Netduino* và *Netduino Plus* là chương trình điều khiển đèn LED *Netduino_8digit*, chương trình hiển thị dòng chữ trên màn hình tinh thể lỏng (LCD - Liquid Crystal Display) *Netduino_LCD* và chương trình thao tác với cổng nối tiếp *Netduino_SerialPort*. Các chương trình này được trình bày cụ thể trong *Phụ lục P.2.5*. Trong giai đoạn thiết kế, mỗi chương trình được thiết kế theo hai biểu đồ lớp khác nhau biểu diễn hai kiến trúc lôgic của phần mềm. Sau đó tính toán các độ đo theo công thức (2.1) đến (2.6), tính giá trị hàm đánh giá hiệu năng của mỗi biểu đồ lớp và vẽ biểu đồ so sánh hiệu năng hai mô hình của mỗi chương trình. Hình 2.4 là các biểu đồ lớp của chương trình *Netduino_8digit*, Hình 2.5 là các biểu đồ lớp của chương trình *Netduino_LCD* và Hình 2.6 là các biểu đồ lớp của chương trình *Netduino_SerialPort*.



a. Biểu đồ 1

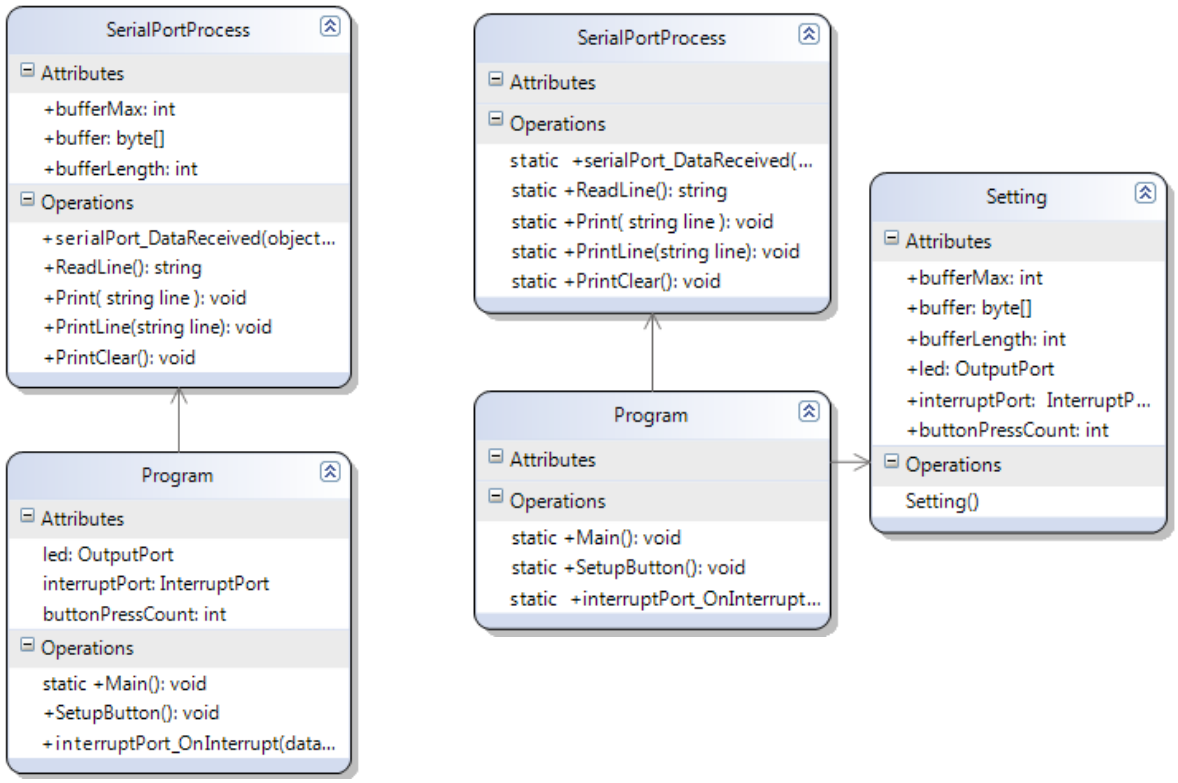
b. Biểu đồ 2

Hình 2.4: Các biểu đồ lớp của chương trình *Netduino_8digit*



a. Biểu đồ 1 b. Biểu đồ 2

Hình 2.5: Các biểu đồ lớp của chương trình *Netduino_LCD*



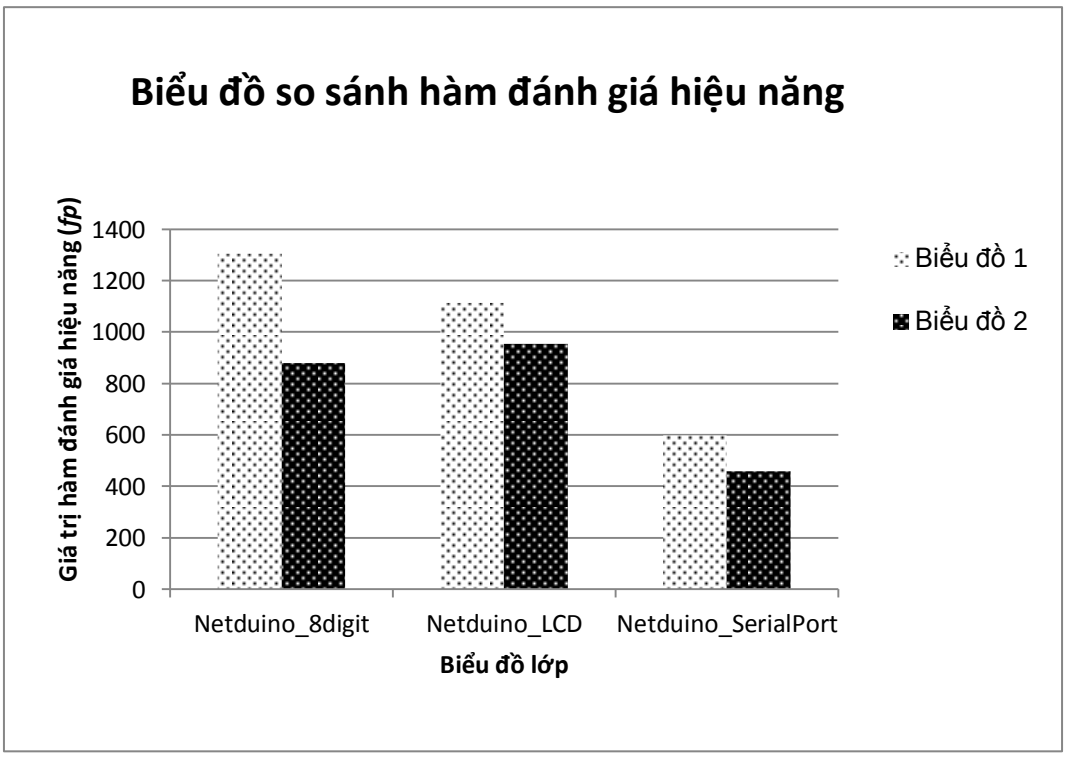
a. Biểu đồ 1 b. Biểu đồ 2

Hình 2.6: Các biểu đồ lớp của chương trình *Netduino_SerialPort*

Từ đặc tả dạng văn bản của các biểu đồ lớp đã thiết kế trong khung làm việc DSL và T4, chúng tôi xây dựng chương trình để phân tích các tham số, tính các độ đo và giá trị của hàm đánh giá tương ứng với mỗi biểu đồ. Kết quả tính toán được thống kê như trong Bảng 2.3. Theo kết quả này, biểu đồ so sánh hiệu năng của các biểu đồ lớp được trình bày như trong Hình 2.7.

Bảng 2.3. Thống kê các độ đo và giá trị hàm đánh giá hiệu năng

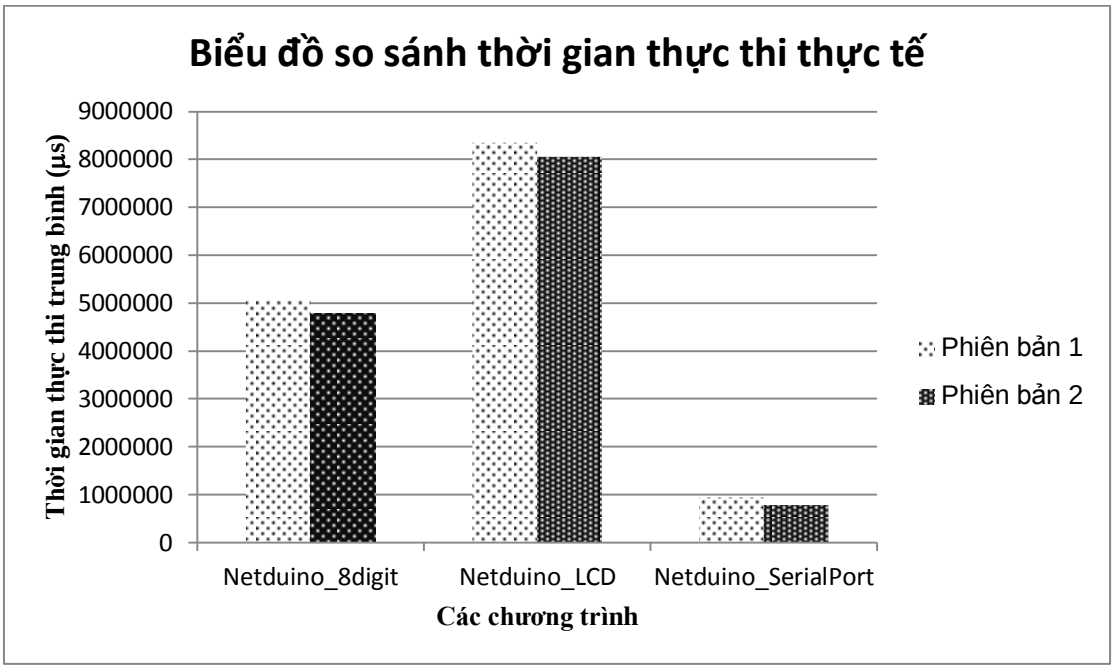
Chương trình	**Biểu đồ lớp**	**Số lớp**	**Số phương thức tĩnh**	**Số thuộc tính tĩnh**	**Số phương thức động**	**Số thuộc tính động**	S_1	S_2	S_3	S_4	S_5	S_6	f_p (Số thao tác bộ nhớ)
Netduino_ 8digit	1	4	3	0	17	9	12	0	12	68	24	137	1307
	2	2	10	5	7	4	40	8	84	28	16	45	879
Netduino_ LCD	1	3	1	0	16	17	4	0	4	64	47	108	1113
	2	3	8	0	9	17	32	0	47	36	47	65	956
Netduino_ SerialPort	1	2	1	0	7	6	4	0	4	28	24	60	600
	2	3	8	0	1	6	32	0	64	4	24	8	460



Hình 2.7: Biểu đồ so sánh hàm hiệu năng trên hai biểu đồ lớp

Trong bước thực nghiệm tiếp theo, mỗi biểu đồ lớp được lập trình thành một phiên bản của chương trình và được thực hiện trên cùng một môi trường để đánh giá hiệu năng. Để tránh ảnh hưởng của việc lập trình trong mỗi phương thức đến hàm đánh giá hiệu

năng, mã nguồn và thuật toán của một phương thức trong hai phiên bản là giống nhau; chỉ khác ở các câu lệnh liên quan đến việc cấp bộ nhớ, sử dụng thuộc tính và gọi phương thức. Sau khi xây dựng, các chương trình ví dụ được thực thi trong cùng một môi trường như mô tả trong Bảng 2.5 để tính thời gian thực thi thực tế. Mỗi chương trình được thực hiện 100 lần để tính thời gian thực thi trung bình. Kết quả thống kê thời gian thực thi được tổng hợp trong Bảng 2.4. Biểu đồ so sánh thời gian thực thi thực tế của hai phiên bản chương trình xây dựng từ hai biểu đổ lớp tương ứng được mô tả như trong Hình 2.8.



Hình 2.8: Biểu đồ so sánh thời gian thực thi thực tế

Bảng 2.4. Tổng hợp thời gian thực thi của các chương trình

	Thời gian thực thi trung bình (µs)		
Chương trình	**Netduino_8digit**	**Netduino_LCD**	**Netduino_SerialPort**
Phiên bản 1	5054135	8345647	937369
Phiên bản 2	4777056	8054425	781022

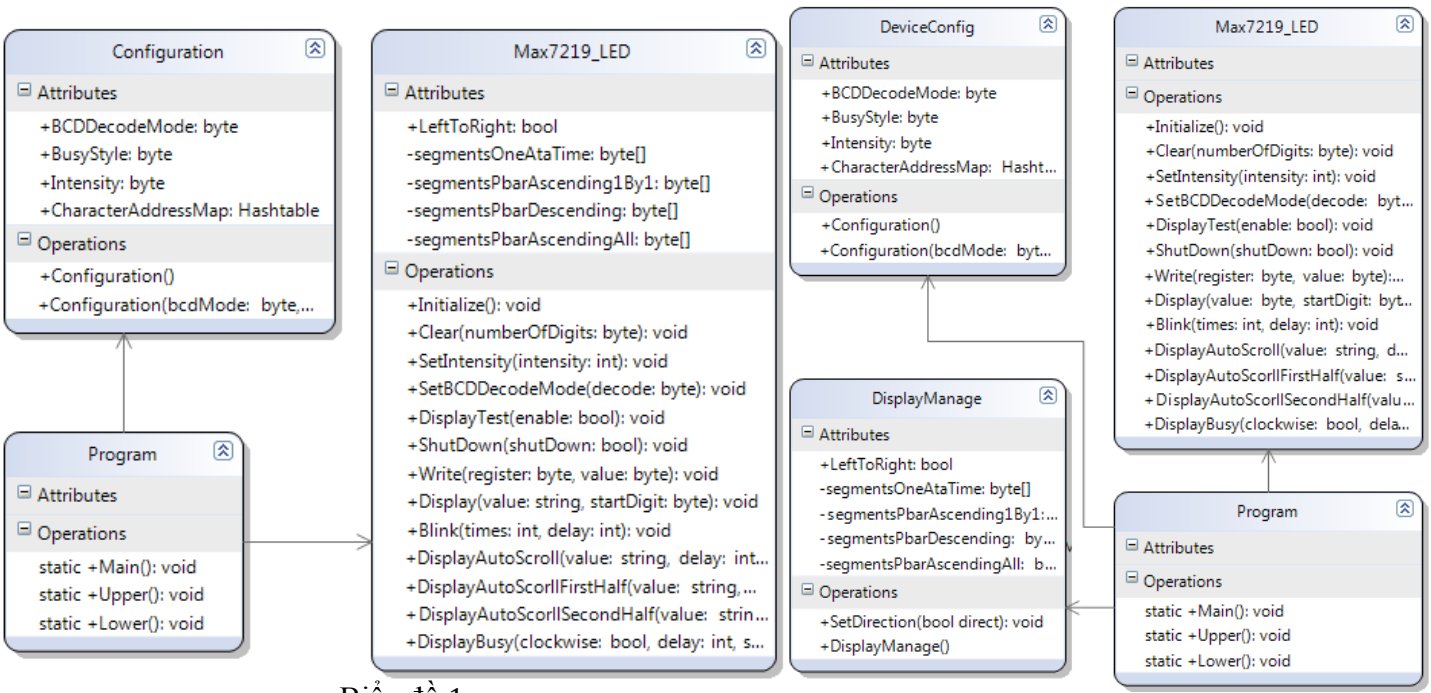
Bảng 2.5. Môi trường thực thi các chương trình ví dụ

Vi điều khiển	AT91SAM7X512
Vi xử lý	ARM7
Kiến trúc tập lệnh	RISC
Bo mạch	Netduino và Netduino Plus
Máy ảo	.NET Micro Framework 4.2
Phần mềm mô phỏng	Microsoft Visual 2010 Express

Từ kết quả thực nghiệm tổng hợp trong Bảng 2.3 và Bảng 2.4 cùng với các biểu đồ so sánh trong Hình 2.7 và Hình 2.8 cho thấy kết quả hàm đánh giá hiệu năng từ biểu đồ lớp phù hợp với thời gian thực thi thực tế. Các chương trình xây dựng theo các mô hình có giá trị hàm đánh giá hiệu năng nhỏ hơn đều có thời gian thực thi nhỏ hơn. Tuy nhiên, từ kết quả đánh giá hiệu năng mức mô hình và hiệu năng thực tế cũng chỉ ra hàm hiệu năng chúng tôi xây dựng chỉ để đánh giá và chọn mô hình có hiệu năng tốt mà không thể sử dụng để ước lượng hiệu năng.

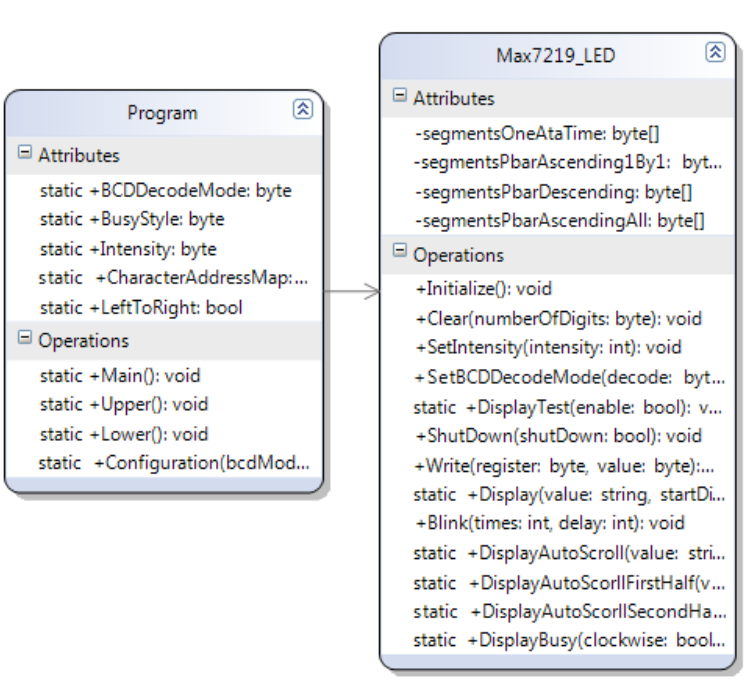
- **Thực nghiệm tối ưu hiệu năng dựa trên đánh giá biểu đồ lớp**

Sau khi kiểm chứng hàm đánh giá hiệu năng, chúng tôi tiến hành thử nghiệm phương pháp tối ưu dựa trên khung làm việc DSL và T4 cùng với chương trình tối ưu đã xây dựng. Trong thực nghiệm này, chúng tôi sử dụng chương trình *Netduino_8digit* để thử nghiệm. Chương trình này được mô tả chi tiết trong *Phụ lục P.2.5.1*. Đầu tiên, chúng tôi sử dụng khung làm việc DSL để thiết kế năm biểu đồ lớp khác nhau. Năm biểu đồ lớp này sẽ được tự động chuyển sang đặc tả dạng văn bản dựa trên các mẫu T4 đã tích hợp trong khung làm việc. Đặc tả dạng văn bản của các mô hình được đưa vào chương trình tối ưu để phân tích và tính giá trị hàm đánh giá hiệu năng. Sau đó chương trình sẽ dựa vào giá trị hàm đánh giá hiệu năng để chọn biểu đồ lớp tốt nhất. Hình 2.9 chỉ ra năm biểu đồ lớp của chương trình *Netduino_8digit* đã được thiết kế trong khung làm việc DSL và T4. Kết quả phân tích tham số, tính giá trị các độ đo và giá trị hàm đánh giá được tổng hợp trong Bảng 2.6. Biểu đồ phân tích hiệu năng và kết quả tối ưu được chỉ ra trong Hình 2.10.

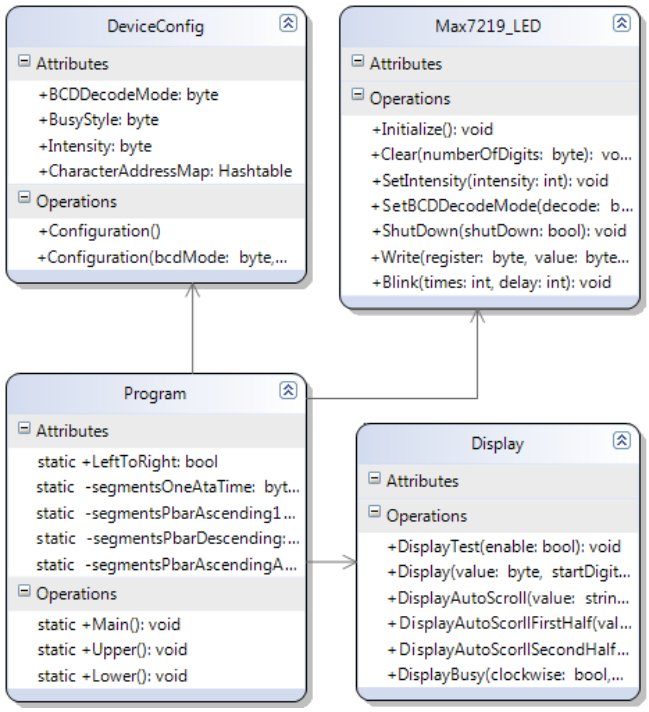


a. Biểu đồ 1

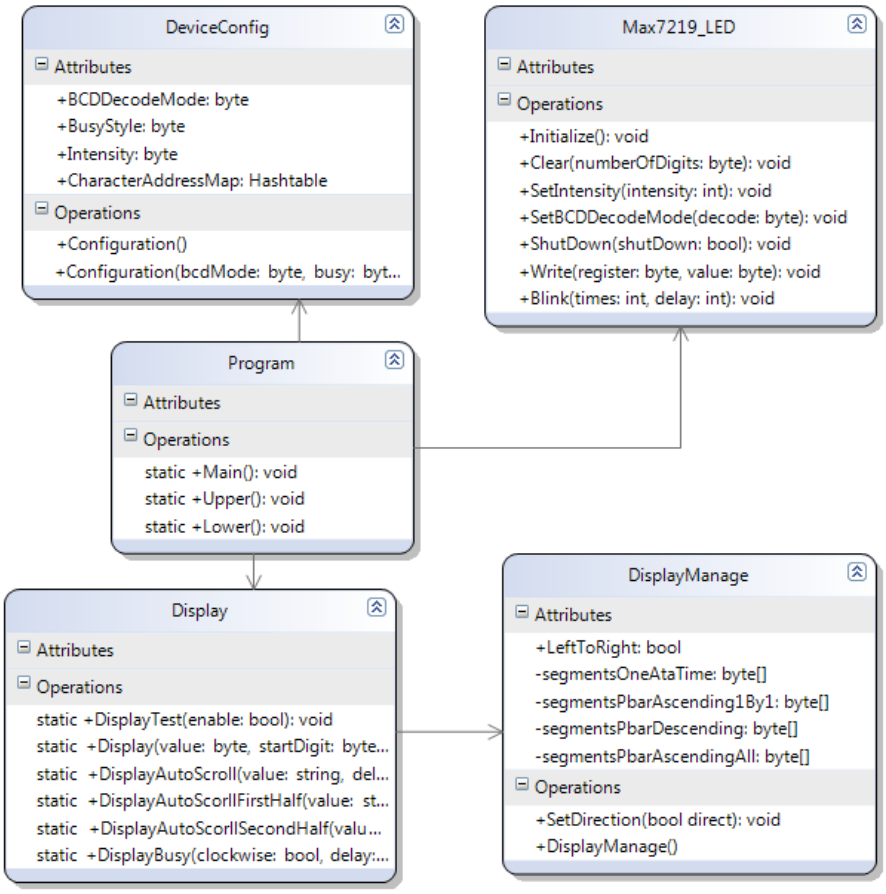
b. Biểu đồ 2



c. Biểu đồ 3

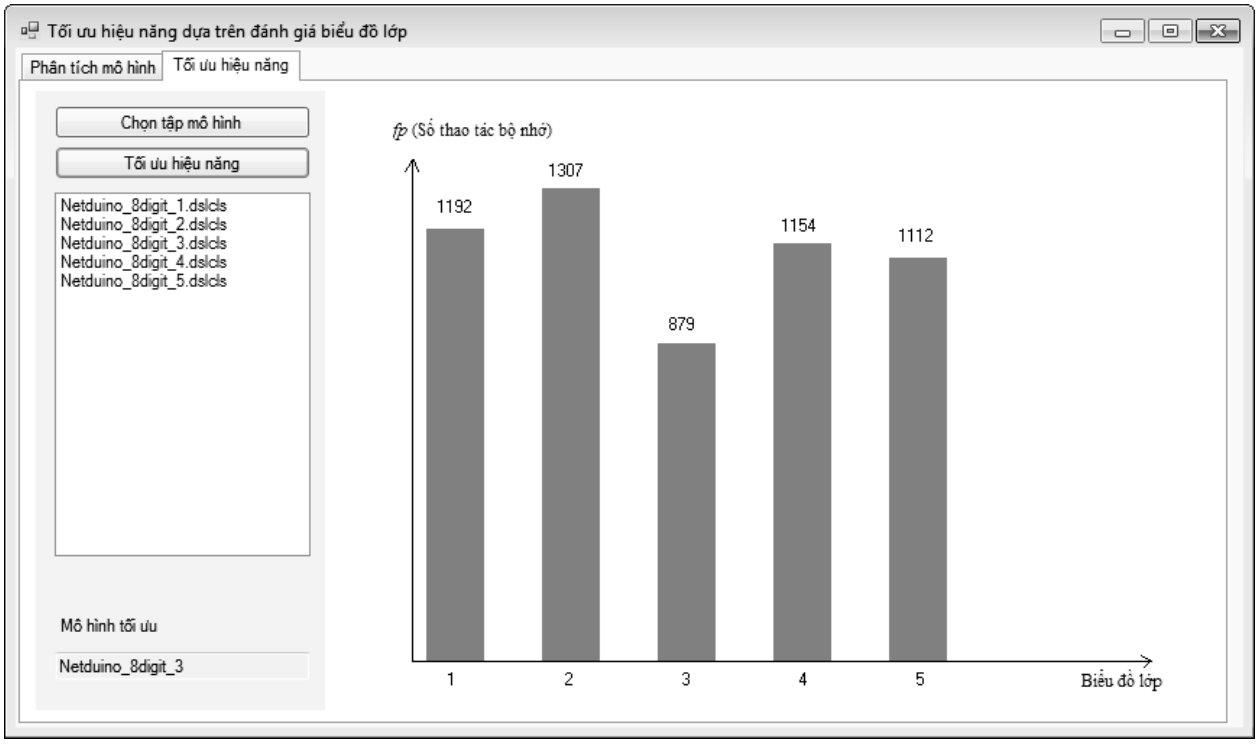


d. Biểu đồ 4



e. Biểu đồ 5

Hình 2.9: Tập các biểu đồ lớp của chương trình *Netduino_8digit*



Hình 2.10: Kết quả tối ưu hiệu năng dựa trên biểu đồ lớp

Bảng 2.6. 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

vii. Đánh giá phương pháp

Phương pháp này giúp lựa chọn một kiến trúc phần mềm có hiệu năng tốt áp dụng cho cả phần mềm nhúng và phần mềm thông thường. Việc lựa chọn biểu đồ có hiệu năng tốt có thể ảnh hưởng đến các độ đo khác như tính tái sử dụng, tính bao gói, độ phức tạp kiến trúc, v.v. Do đó, dựa trên hàm đánh giá hiệu năng để lựa chọn biểu đồ lớp tốt có thể kết hợp với các ràng buộc trên các độ đo khác như trong nghiên cứu [36]. Hơn nữa, biểu đồ lớp của phần mềm nhúng cũng tương tự như phần mềm thông thường nhưng có nhiều

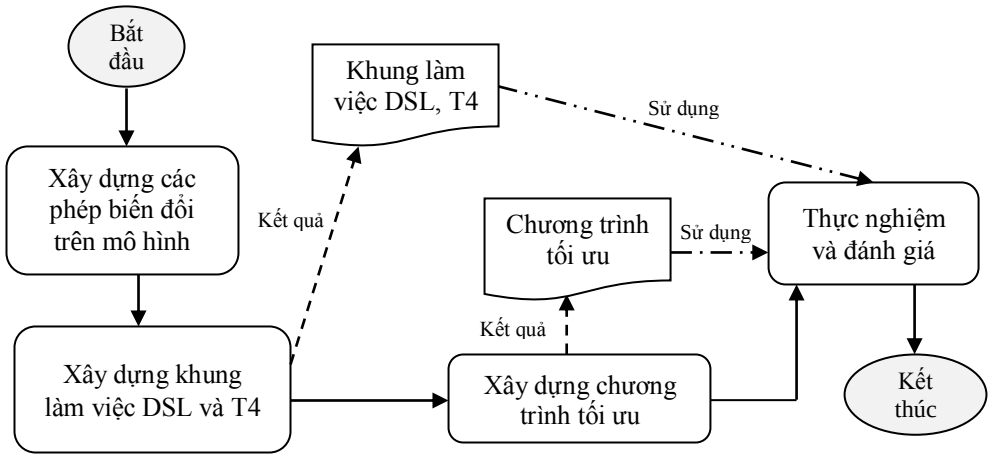
ràng buộc hơn như kiểu dữ liệu, miền giá trị, v.v. nên việc sử dụng DSL, T4 để xây dựng biểu đồ và sinh đặc tả tự động là có cơ sở và triển vọng.

Trong phương pháp này, đầu vào là một tập các biểu đồ lớp đã có, chương trình sẽ phân tích các thành phần để tính các độ đo và hàm đánh giá để lựa chọn biểu lớp tốt nhất mà chưa đánh giá sự nhất quán và tương đương ngữ nghĩa giữa các mô hình. Đồng thời, biểu đồ lớp chúng tôi sử dụng trong nghiên cứu này cũng không xét trường hợp các lớp lồng nhau, các phương thức ảo cũng như không hỗ trợ kế thừa và giao diện. Do đó, việc tối ưu trên các biểu đồ lớp phức tạp, đầy đủ và đảm bảo sự nhất quán ngữ nghĩa giữa các mô hình cũng là một hướng nghiên cứu mở trên cơ sở nghiên cứu này. Ngoài ra, nghiên cứu này còn được sử dụng làm cơ sở cho các nghiên cứu sâu hơn về tối ưu trong kỹ nghệ ngược như sau: từ mã nguồn có thể sinh ngược ra biểu đồ lớp, tái cấu trúc và dựa vào hàm đánh giá để lựa chọn biểu đồ lớp tốt.

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

i. Ý tưởng và quy trình nghiên cứu

Ý tưởng của phương pháp tối ưu hiệu năng phần mềm nhúng trong giai đoạn thiết kế dựa trên chuyển đổi mô hình là dựa vào các phép biến đổi trên mô hình để đưa mô hình thiết kế ban đầu về mô hình tối ưu. Quy trình nghiên cứu và thực nghiệm phương pháp tối ưu này được thể hiện như trong Hình 2.11.



Hình 2.11: Quy trình nghiên cứu và thực nghiệm tối ưu hiệu năng dựa trên chuyển đổi mô hình

ii. Các phép biến đổi trên mô hình

Để tối ưu hiệu năng chúng tôi đưa ra và chứng minh 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. Ba phép biến đổi này được chứng minh

dựa trên công thức (2.7) và được sử dụng để chuyển đổi mô hình ban đầu về mô hình tối ưu.

- **Thu gọn kiểu dữ liệu**

Đây là phép biến đổi đơn giản nhưng hiệu quả trong cải tiến hiệu năng và bộ nhớ. Tư tưởng của phép biến đổi này là dựa trên miền giá trị để chuyển kiểu của các thành viên dữ liệu và các tham số về kiểu dữ liệu nhỏ nhất bao phủ miền giá trị. Tính đúng đắn của phép biến đổi này được chứng minh dựa trên công thức (2.7). Dựa vào miền giá trị đặt ra khi thiết kế mà kiểu dữ liệu của các thuộc tính và các tham số được đưa về kiểu nhỏ nhất nên S_1, S_3, S_4 và S_6 giảm đi; S_2 và S_5 không thay đổi. Đồng thời các hệ số phụ thuộc α, β, γ và ε không thay đổi nên giá trị hàm đánh giá hiệu năng giảm đi. Do đó, sử dụng phép biến đổi thu gọn kiểu dữ liệu có thể cải tiến hiệu năng phần mềm.

Để thực hiện phép biến đổi này, trong quá trình thiết kế mô hình dữ liệu cần chỉ rõ miền giá trị của mỗi thuộc tính. Do đó, khi thiết kế siêu mô hình để xây dựng công cụ DSL trong phần sau, mỗi thành phần dữ liệu cũng cần định nghĩa thêm thuộc tính miền giá trị.

- **Chuyển tham số thành thuộc tính**

Dựa trên hàm đánh giá hiệu năng trong phần trước, chúng ta có thể chứng minh được thời gian thực thi một phương thức với tham số truyền vào sẽ lớn hơn thời gian thực thi phương thức đó khi sử dụng các thành phần dữ liệu thay cho các tham số. Để đảm bảo không làm thay đổi ngữ nghĩa khi thực hiện phép biến đổi này, các tham số của các phương thức tĩnh sẽ được chuyển thành các thành phần dữ liệu tĩnh và các tham số của các phương thức động sẽ được chuyển thành các thành phần dữ liệu động.

Giả sử số tham số của các phương thức đối tượng được chuyển thành thuộc tính đối tượng là Po và số tham số của các phương thức tĩnh chuyển thành thuộc tính tĩnh là Ps. Khi đó các độ đo S_2 và S_5 không thay đổi vì phép biến đổi này không thay đổi số lượng và kiểu các phương thức. Các độ đo khác và hàm hiệu năng có giá trị mới được tính như sau:

$$S_1' = S_1 + \sum_{i=1}^{Ps} \text{size}(p_i)$$

$$S_3' = S_3 - \sum_{i=1}^{Ps} \text{size}(p_i)$$

$$S_4' = S_4 + \sum_{i=1}^{Po} \text{size}(p_i)$$

$$S_6' = S_6 - \sum_{i=1}^{Po} \text{size}(p_i)$$

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

$$f_p - f_p' = \alpha \times (S_1 - S_1') + \beta \times (S_3 - S_3') + \gamma \times (S_4 - S_4') + \varepsilon \times (S_6 - S_6')$$

$$= (\beta - \alpha) \times \sum_{i=1}^{Ps} \text{size}(p_i) + (\varepsilon - \gamma) \times \sum_{i=1}^{Po} \text{size}(p_i) \quad (2.8)$$

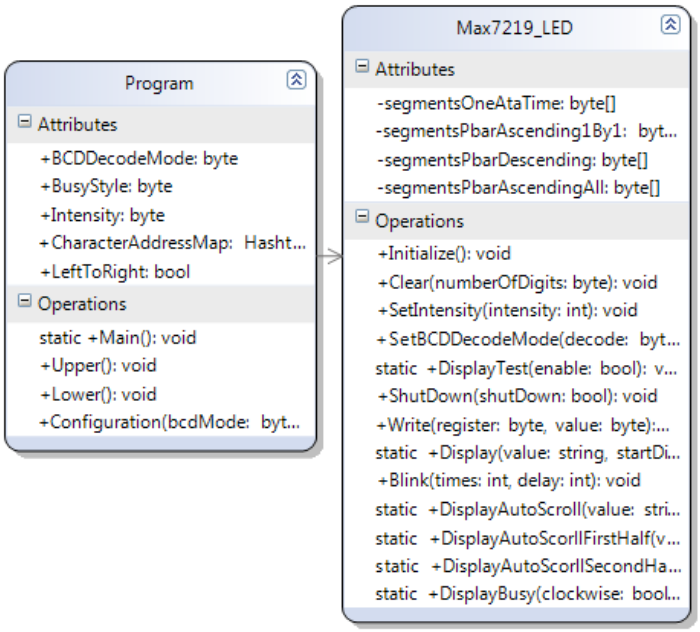
Theo công thức (2.8), do $\alpha < \beta$ và $\gamma < \varepsilon$ nên f_p lớn hơn f_p'. Điều này chứng minh có thể cải tiến hiệu năng dựa trên phép biến đổi này. Mặc dù phép biến đổi này cải tiến hiệu năng nhưng cũng làm tăng dung lượng bộ nhớ chiếm dụng. Do đó, khi xây dựng chương trình biến đổi tự động trên mô hình để tối ưu hiệu năng, cần xác định thêm các ràng buộc để không làm dung lượng bộ nhớ chiếm dụng tăng nhiều. Điều này rất quan trọng đối với phần mềm nhúng khi bộ nhớ trong các thiết bị nhúng thường có kích thước nhỏ.

- **Chuyển các thành phần động thành tĩnh**

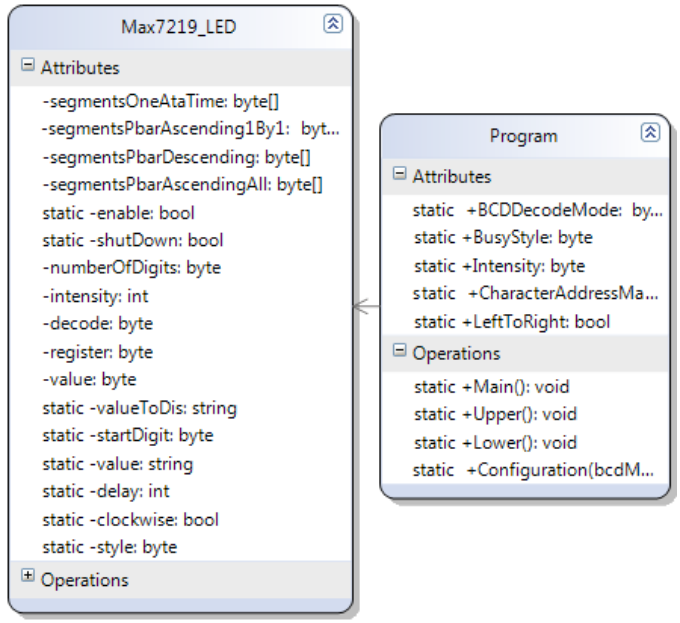
Như đã trình bày trong phần xây dựng hàm đánh giá, chương trình sẽ thực thi nhanh hơn khi sử dụng các phương thức và các thuộc tính tĩnh. Do đó, để cải tiến hiệu năng, chúng tôi cũng đưa ra phép biến đổi là chuyển các thuộc tính đối tượng thành các thuộc tính tĩnh và chuyển các phương thức đối tượng thành phương thức tĩnh. Tính đúng đắn của phép biến đổi này được chứng minh dựa trên hàm đánh giá hiệu năng trong công thức (2.7). Khi thực hiện phép biến đổi này, S_1, S_2, S_3 tăng lên và S_4, S_5, S_6 giảm đi. Kích thước S_1 tăng lên bằng kích thước S_4 giảm đi, kích thước S_2 tăng lên bằng kích thước S_5 giảm đi và kích thước S_3 tăng lên bằng kích thước S_6 giảm đi. Theo phân tích trong phần trước, $\alpha < \gamma$ và $\beta < \varepsilon$ nên sau khi thực hiện phép biến đổi giá trị hàm hiệu năng sẽ giảm đi. Nghĩa là hiệu năng đã được cải tiến.

iii. Ví dụ minh họa tối ưu hiệu năng dựa trên biến đổi mô hình

Để minh họa các bước tối ưu hiệu năng dựa trên biến đổi mô hình, chúng tôi sử dụng chương trình thử nghiệm *Netduino_8digit*. Đầu tiên, chúng tôi sử dụng khung làm việc DSL và T4 để thiết kế mô hình dữ liệu của chương trình như trong Hình 2.12. Từ mô hình dữ liệu này, sử dụng mẫu T4 để tự động sinh đặc tả dạng văn bản như trong Hình 2.14. Chương trình tối ưu sẽ phân tích đặc tả dạng văn bản của mô hình ban đầu, thực hiện các phép biến đổi để đưa về đặc tả dạng văn bản cho mô hình tối ưu như trong Hình 2.15. Chúng tôi đã xây dựng khung làm việc DSL, T4 như trình bày trong *Phụ lục P.1.1* và xây dựng chương trình tối ưu như trong *Phụ lục P.1.5*. Biểu đồ so sánh hiệu năng dựa trên giá trị hàm đánh giá hiệu năng được trình bày như trong Hình 2.16. Mô hình dữ liệu tối ưu được thể hiện như trong Hình 2.13.



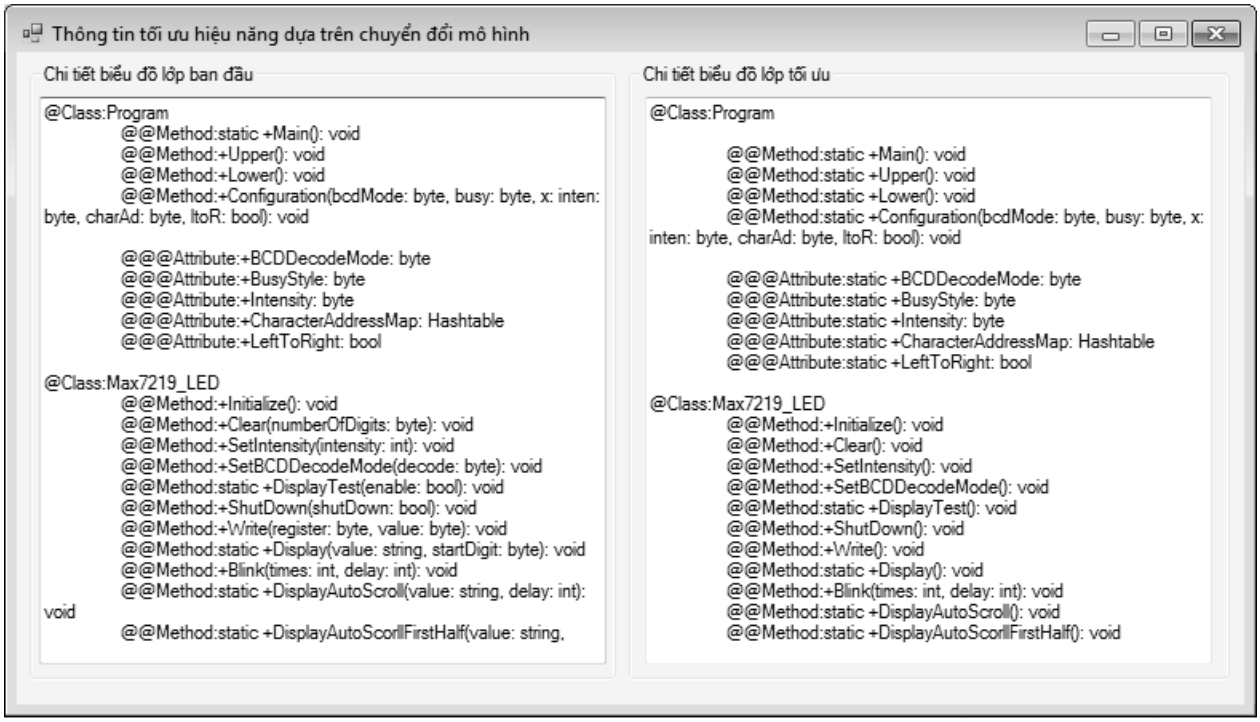
Hình 2.12: Mô hình ban đầu



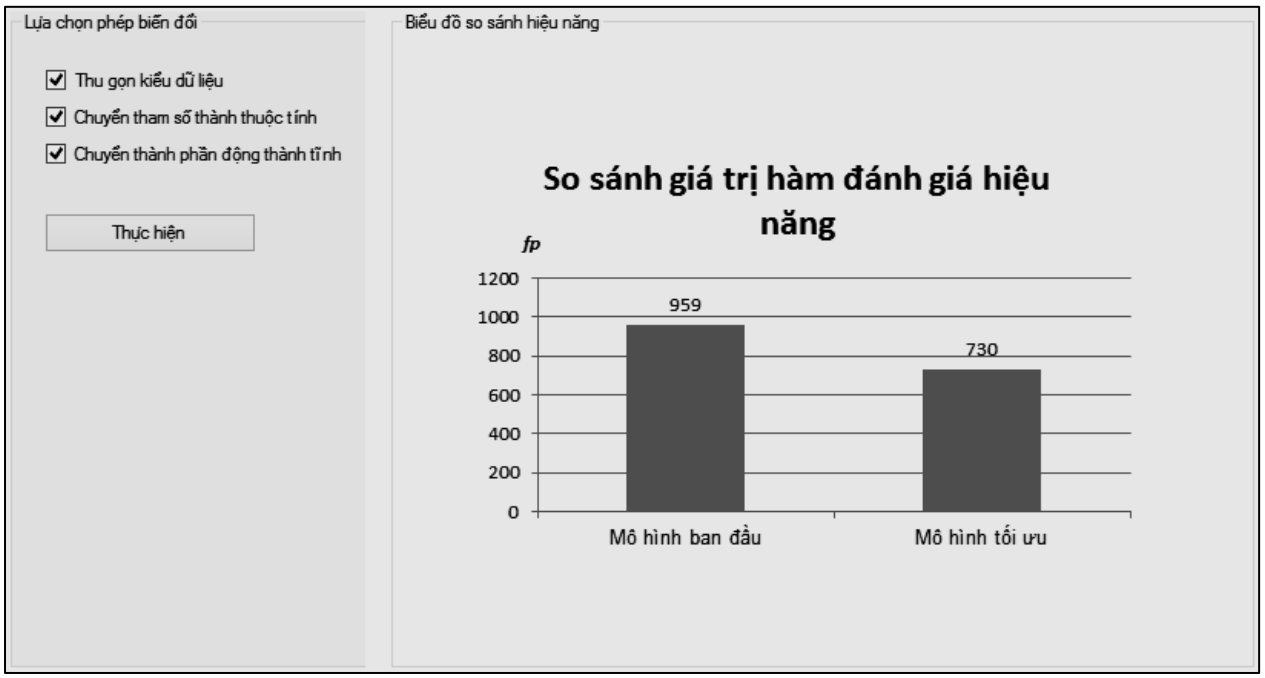
Hình 2.13: Mô hình tối ưu

```
@Class:Program
        @@Method:static +Main(): void
        @@Method:static +Upper(): void
        @@Method:static +Lower(): void
        @@Method:static +Configuration(bcdMode: byte, busy: b
        @@@Attribute:static +BCDDecodeMode: byte
        @@@Attribute:static +BusyStyle: byte
        @@@Attribute:static +Intensity: byte
        @@@Attribute:static +CharacterAddressMap: Hashtable
        @@@Attribute:static +LeftToRight: bool
@Class:Max7219_LED
        @@Method:+Initialize(): void
        @@Method:+Clear(): void
        @@Method:+SetIntensity(): void
        @@Method:+SetBCDDecodeMode(): void
        @@Method:static +DisplayTest(): void
        @@Method:+ShutDown(): void
        @@Method:+Write(): void
        @@Method:static +Display(): void
        @@Method:+Blink(times: int, delay: int): void
        @@Method:static +DisplayAutoScroll(): void
        @@Method:static +DisplayAutoScorllFirstHalf(): void
        @@Method:static +DisplayAutoScorllSecondHalf(): void
```

Hình 2.14: Đặc tả dạng văn bản được sinh tự động từ mô hình



Hình 2.15: Minh họa đặc tả ban đầu và đặc tả tối ưu



Hình 2.16: Lựa chọn phép chuyển đổi và biểu đồ so sánh hiệu năng

iv. Thực nghiệm

Mục đích của thực nghiệm này nhằm kiểm chứng các phép biến đổi và kết quả tối ưu. Các bước thực nghiệm tiến hành theo ví dụ minh họa trong phần trước. Ngoài ra, để kiểm chứng kết quả tối ưu theo hàm đánh giá hiệu năng và kết quả thực tế, chúng tôi xây dựng các chương trình thử nghiệm theo mô hình ban đầu và mô hình tối ưu. Các chương

trình thử nghiệm được thực thi trong cùng một môi trường; mỗi chương trình được thực hiện 100 lần; tính thời gian thực thi trung bình để đánh giá hiệu năng thực tế của chương trình theo mô hình ban đầu và mô hình tối ưu. Trong thực nghiệm này, chúng tôi thực thi ba chương trình nhúng cho *Bo mạch Netduino* và *Netduino Plus* đó là *Netduino_8digit*, *Netduino_LCD* và *Netduino_SerialPort* trong môi trường mô phỏng Microsoft Visual 2010 Express. Kết quả thực nghiệm được thống kê trong Bảng 2.7. Trong Bảng 2.7 giá trị hàm đánh giá hiệu năng và giá trị hiệu năng thực tế được tính theo micrô giây (μs).

Bảng 2.7. 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

v. Đánh giá phương pháp

Kết quả thực nghiệm trong Bảng 2.7 cho thấy kết quả tối ưu hiệu năng dựa trên biến đổi mô hình phù hợp với kết quả thực tế. Phương pháp tối ưu này giúp cải tiến hiệu năng đáng kể trong giai đoạn thiết kế phần mềm nhúng. So với nghiên cứu ban đầu của Anne, K. [7, 8], trong phương pháp cải tiến này, chúng tôi đã bổ sung ba phép biến đổi mô hình và triển khai tự động dựa trên DSL và T4.

Tuy đạt được cải tiến về hiệu năng nhưng phương pháp này cũng làm giảm khả năng bảo trì, khả năng tái sử dụng, tính cấu trúc của hệ thống và làm tăng kích thước bộ nhớ chiếm dụng. Cụ thể như so sánh giữa Hình 2.12 và Hình 2.13, mô hình tối ưu có nhiều thành phần tĩnh hơn nên chiếm dụng bộ nhớ nhiều hơn; các tham số trong phương thức được chuyển thành thuộc tính nên làm giảm tính bao gói nên khó bảo trì và tái sử dụng hơn; các phương thức tĩnh chỉ truy xuất được các thuộc tính tĩnh và chỉ gọi được các phương thức tĩnh khác nên cũng ảnh hưởng xấu đến ngữ nghĩa. Vì vậy chỉ nên thực hiện phép biến đổi này đối với các thuộc tính, phương thức có tần số sử dụng lớn. Phương pháp này có thể được áp dụng cho cả phần mềm nhúng và phần mềm thông thường. Khi thực hiện các phép biến đổi cần đảm bảo về ngữ nghĩa và các ràng buộc về các độ đo chất lượng khác như trong nghiên cứu [36]. Mặt khác, tuy còn hạn chế nhưng nghiên cứu này là cơ sở để kết hợp với kỹ nghệ ngược góp phần giải quyết bài toán tối ưu tổng thể đó là: từ mã nguồn sinh ngược biểu đồ, thực hiện các phép biến đổi trên mô hình để đạt được cấu trúc tốt.

2.2. Tối ưu bộ nhớ trong giai đoạn thiết kế

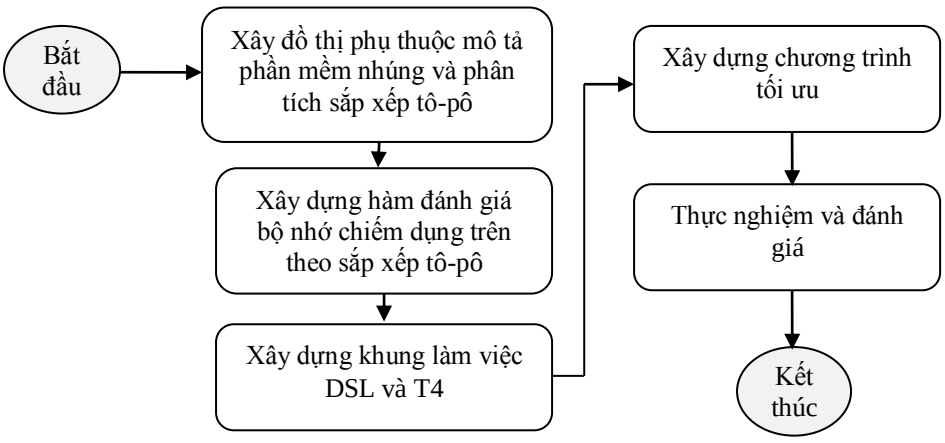
Theo cách tiếp cận tối ưu dựa trên đánh giá trực tiếp các mô hình thiết kế, trong phần này chúng tôi đề xuất và phát triển một phương pháp tối ưu mới đó là tối ưu bộ nhớ chiếm dụng của phần mềm nhúng dựa trên sắp xếp tô-pô, DSL và T4. Chúng tôi cũng cải tiến phương pháp tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình do Anne, K. đề xuất [7, 8]. Hai phương pháp tối ưu bộ nhớ mức mô hình trong phần này được xây dựng dựa trên kế thừa và cải tiến các phương pháp tối ưu hiệu năng trong phần trước. Để thiết kế và sinh đặc tả tự động từ mô hình, chúng tôi kế thừa khung làm việc DSL và T4 đã được xây dựng trong phần trước và sửa đổi cho phù hợp. Để tối ưu, chúng tôi cũng phân tích mô hình, xây dựng các hàm đánh giá bộ nhớ và viết chương trình tối ưu. Các nội dung tiếp theo trong phần này sẽ trình bày chi tiết về hai phương pháp tối ưu này.

2.2.1. Tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô

i. Ý tưởng và quy trình nghiên cứu

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.

Trong phương pháp này, mỗi chương trình được mô tả theo một tập các tác vụ và quan hệ giữa các tác vụ. Tập tác vụ được mô hình hóa bằng một đồ thị có hướng gọi là đồ thị tác vụ phụ thuộc. Mỗi đỉnh trong đồ thị biểu diễn một tác vụ. Mỗi cạnh biểu diễn sự phụ thuộc giữa các tác vụ. Thứ tự thực hiện các tác vụ theo các chuỗi tô-pô không làm thay đổi ngữ nghĩa của chương trình nhưng có hiệu năng và mức chiếm dụng bộ nhớ khác nhau. Từ tập các chuỗi tô-pô trên đồ thị phụ thuộc, dựa vào giá trị hàm đánh giá bộ nhớ để tìm chuỗi tô-pô có mức chiếm dung lượng bộ nhớ nhỏ nhất khi thực thi chương trình.



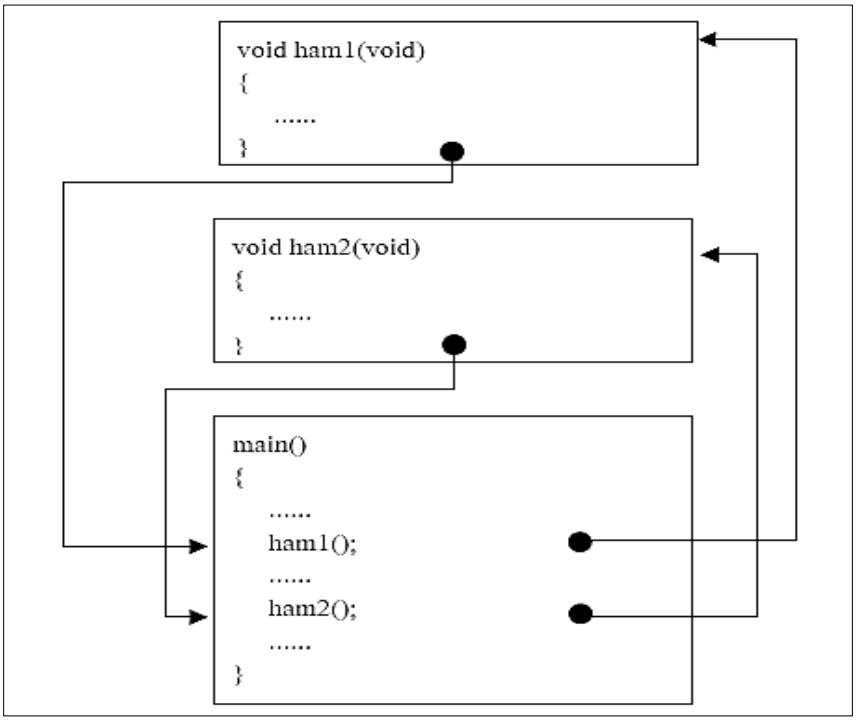
Hình 2.17: Quy trình nghiên cứu phương pháp tối ưu bộ nhớ dựa trên sắp xếp tô-pô

Để triển khai phương pháp này, chúng tôi tiến hành nghiên cứu và thực nghiệm theo quy trình trong Hình 2.17. Đầu tiên, chúng tôi định nghĩa đồ thị tác vụ phụ thuộc đặc tả phần mềm nhúng và phân tích sắp xếp tô-pô trên đồ thị phụ thuộc. Tiếp theo, chúng tôi xây dựng và chứng minh hàm đánh giá dung lượng bộ nhớ chiếm dụng khi thực thi phần mềm nhúng trên một chuỗi tô-pô. Để thiết kế đồ thị phụ thuộc trong giao diện đồ họa và tự động sinh đặc tả dạng văn bản và chuyển sang biểu diễn toán học của đồ thị chúng tôi cũng xây dựng khung làm việc DSL và T4. Sau đó, chúng tôi xây dựng chương trình tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô. Cuối cùng, chúng tôi tiến hành thực nghiệm để kiểm tra và đánh giá phương pháp tối ưu.

ii. Đồ thị phụ thuộc và sắp xếp tô-pô

- **Đồ thị phụ thuộc**

Các chương trình nhúng thường được viết bằng ngôn ngữ C/C++. Cấu trúc chương trình viết bằng C/C++ đều có cấu trúc chương trình như trong Hình 2.18. Theo cấu trúc này, chương trình có duy nhất một hàm chính đóng vai trò điểm vào để thực thi chương trình. Các hàm khác được gọi thực thi bên trong hàm chính. Các ngôn ngữ lập trình phát triển từ C/C++ cũng có cấu trúc chương trình tương tự.



Hình 2.18: Cấu trúc một chương trình C/C++

Trong các ngôn ngữ hướng đối tượng phổ biến như C#, Java, v.v. chương trình muốn thực thi trực tiếp cần phải có duy nhất một lớp chứa một hàm chính khai báo tĩnh và phạm vi truy xuất toàn cục. Theo đó, phần mềm nhúng theo cấu trúc này 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. Như vậy 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:

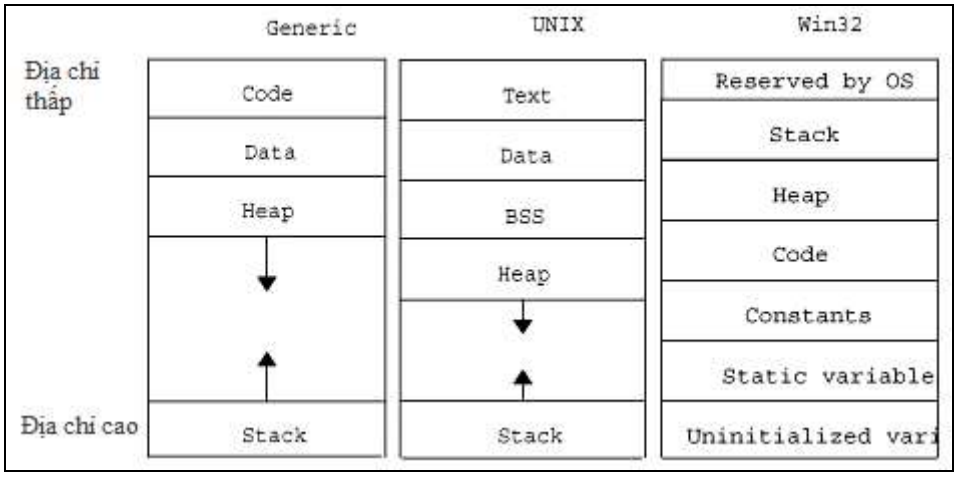
- Mỗi đỉnh u_i tương ứng với một tác vụ thứ i
- Mỗi cạnh v_{ij} cho biết tác vụ thứ j phụ thuộc vào tác vụ thứ i và chỉ được thực hiện khi tác vụ thứ i đã kết thúc.

- **Sắp xếp tô-pô trên đồ thị phụ thuộc**

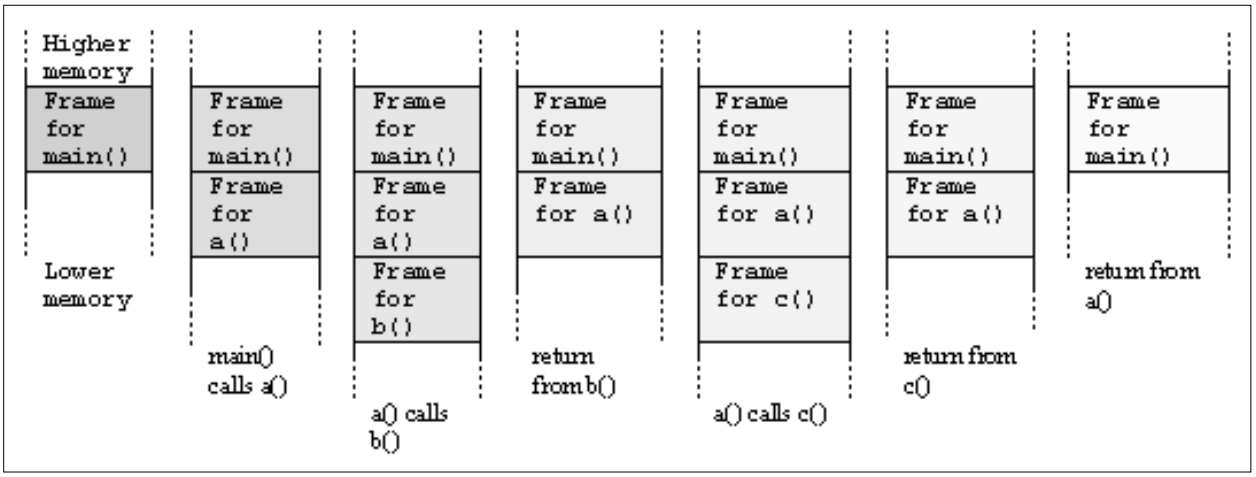
Như mô tả ở trên, chương trình bao gồm một tập các tác vụ và được biểu diễn bằng một đồ thị phụ thuộc. Với cùng một tập tác vụ nhưng thứ tự thực hiện khác nhau sẽ ảnh hưởng đến hiệu quả sử dụng bộ nhớ và hiệu năng của chương trình. Từ một tập tác vụ, có thể tồn tại nhiều thứ tự thực hiện khác nhau thỏa mãn đồ thị phụ thuộc mà không làm thay đổi kết quả chương trình. Đây chính là sắp xếp tô-pô trên đồ thị phụ thuộc. Theo đó, phương pháp tối ưu này nhằm tìm chuỗi tô-pô có mức chiếm dụng bộ nhớ ít nhất dựa trên giá trị của hàm đánh giá bộ nhớ.

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

Trong phần này, chúng tôi xây dựng hàm đánh giá dung lượng bộ nhớ chiếm dụng của chương trình trên mỗi chuỗi tô-pô. Đầu tiên, chúng tôi phân tích quá trình cấp phát bộ nhớ và bố cục bộ nhớ khi thực thi chương trình. Hình 2.19 minh họa bố cục bộ nhớ trong quá trình thực thi chương trình như sau: vùng nhớ tĩnh – được cấp phát ngay khi nạp chương trình, chứa các thành phần dữ liệu tĩnh và mã nguồn; vùng ngăn xếp – cung cấp các trang nhớ để lưu trữ các biến cục bộ, các tham số cho mỗi hàm. Sau khi hàm kết thúc, các trang nhớ đã cấp phát cho hàm đó bị thu hồi; vùng nhớ heap – chứa các thành phần bộ nhớ được cấp phát và thu hồi động (ví dụ `alloc`, `malloc` trong C). Quá trình cấp phát tự động và thu hồi bộ nhớ khi thực hiện chương trình bao gồm một hàm chính và các tác vụ (các hàm thành phần) được chỉ ra trong Hình 2.20.



Hình 2.19: Tổ chức bộ nhớ khi thực hiện chương trình



Hình 2.20: Cấp phát và giải phóng vùng nhớ ngăn xếp khi thực thi hàm

Giả sử chương trình có N tác vụ và u_i là tác vụ thứ i trong chuỗi tô-pô. u_i có kiểu dữ liệu trả về r_i. Khi thực hiện u_i, các biến cục bộ và tham số của u_i sẽ được cấp phát trong vùng ngăn xếp và nó sẽ được giải phóng khi u_i thực hiện xong. Do đó, tổng kích thước cấp bộ nhớ sử dụng để thực thi các tác vụ trong các chuỗi tô-pô như nhau. Tuy nhiên, sau khi các tác vụ thực hiện xong, cần lưu kết quả trả về trong các biến cục bộ của chương trình và các biến này được cấp phát trong ngăn xếp khi khai báo và gán bằng giá trị trả về của tác vụ. Đồng thời, trong thời gian thiết kế, các tác vụ chưa có mã nguồn thực thi nên không thể đánh giá được thời gian thực hiện. Mặt khác, thời gian thực hiện của một tác vụ không phụ thuộc vào vị trí trong chuỗi tô-pô nhưng thứ tự của tác vụ trong chuỗi sẽ ảnh hưởng đến dung lượng bộ nhớ chiếm dụng. Do đó, không mất tính tổng quát, chúng tôi giả thiết mỗi tác vụ thực hiện trong một đơn vị thời gian. Khi đó, để lưu trữ kết quả trả về từ tác vụ thứ i, cần khai báo một biến và cấp phát bộ nhớ cho biến tại bước i. Nếu khai báo biến này trước bước i thì sẽ làm tăng mức độ chiếm dụng bộ nhớ của chương trình. Sau khi khai báo và cấp phát bộ nhớ, biến này sẽ tồn tại đến khi chương

trình thực hiện xong. Nên mức độ chiếm dụng bộ nhớ của biến phụ thuộc vào kích thước kiểu dữ liệu trả về của tác vụ thứ *i* và được tính bằng tích của (*N* - *i*) và kích thước kiểu trả về của tác vụ thứ *i*. Từ việc phân tích mức chiếm dụng bộ nhớ gây ra bởi mỗi tác vụ, chúng tôi xây dựng hàm đánh kích thước bộ nhớ chiếm dụng của chương trình phụ thuộc vào thứ tự thực thi theo 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 i.

Giải thích công thức (2.10)

Với giả thiết trên, tổng kích thước bộ nhớ chiếm dụng của chương trình được tính gồm bộ nhớ chứa mã lệnh, bộ nhớ chứa dữ liệu tĩnh, ngăn xếp và bộ nhớ được cấp phát động, cụ thể như sau:

$$M \geq S_c \times N + D_s \times N + \max_{i \leq N}(S_e^i) + \sum_{i=1}^{N} (N - i) \times \text{size}(r_i) \tag{2.11}$$

Trong đó:

- S_c là kích thước bộ nhớ chứa mã lệnh
- D_s là kích thước bộ nhớ chứa dữ liệu tĩnh
- S_e^i là kích thước vùng nhớ heap được cấp phát động khi thực thi tác vụ i.

Vùng nhớ chứa đoạn mã lệnh và vùng nhớ chứa dữ liệu tĩnh có kích thước không thay đổi, được cấp phát ngay từ khi nạp chương trình và tồn tại đến khi chương trình kết thúc; chương trình có *N* tác vụ tuần tự và mỗi tác vụ thực hiện trong một đơn vị thời gian như giả thiết trên thì thời gian thực hiện của chương trình có thể đánh giá bằng *N*; kích thước bộ nhớ chiếm dụng tính bằng kích thước bộ nhớ nhân với thời gian tồn tại nên $S_c \times N$ là kích thước bộ nhớ chiếm dụng của đoạn mã lệnh và $D_s \times N$ là kích thước bộ nhớ chiếm dụng của đoạn dữ liệu tĩnh. $\max_{i \leq N}(S_e^i)$ bằng nhau với mọi chuỗi thực thi.

Suy ra $S_c \times N + D_s \times N + \max_{i \leq N}(S_e^i)$ là như nhau với mọi chuỗi tô-pô.

Do đó, hàm đánh giá kích thước bộ nhớ chiếm dụng của chương trình trên một chuỗi tô-pô có thể được định nghĩa như công thức (2.10).

iv. Xây dựng chương trình tối ưu bộ nhớ dựa trên sắp xếp tô-pô

Sau khi phát triển khung làm việc DSL và T4 để thiết kế và sinh đặc tả dạng văn bản, chúng tôi xây dựng chương trình thực hiện phương pháp tối ưu bộ nhớ dựa trên sắp xếp tô-pô. Chương trình nhận đầu vào là tệp tin mô tả dạng văn bản của đồ thị phụ thuộc và phân tích tệp tin để chuyển sang biểu diễn toán học của đồ thị phụ thuộc. Trong chương trình, chúng tôi đã lập trình hàm đánh giá mức chiếm dụng bộ nhớ theo công thức (2.10), xây dựng mô-đun tìm các chuỗi tô-pô thỏa mãn đồ thị phụ thuộc và lựa chọn chuỗi tô-pô tốt nhất.

Khi thực hiện chương trình theo các chuỗi tô-pô của các tác vụ, kích thước bộ nhớ sử dụng của chương trình theo mỗi chuỗi là như nhau nhưng thời gian chiếm dụng bộ nhớ khác nhau. Chuỗi tô-pô được chọn là chuỗi có f_m nhỏ nhất. Để tìm chuỗi tô-pô tốt nhất, chúng tôi xây dựng mô-đun lựa chọn chuỗi tô-pô thực hiện các bước sau: nhận đầu vào là tập các chuỗi tô-pô thỏa mãn đồ thị phụ thuộc; duyệt và tính f_m cho mỗi chuỗi; lựa chọn chuỗi có f_m nhỏ nhất.

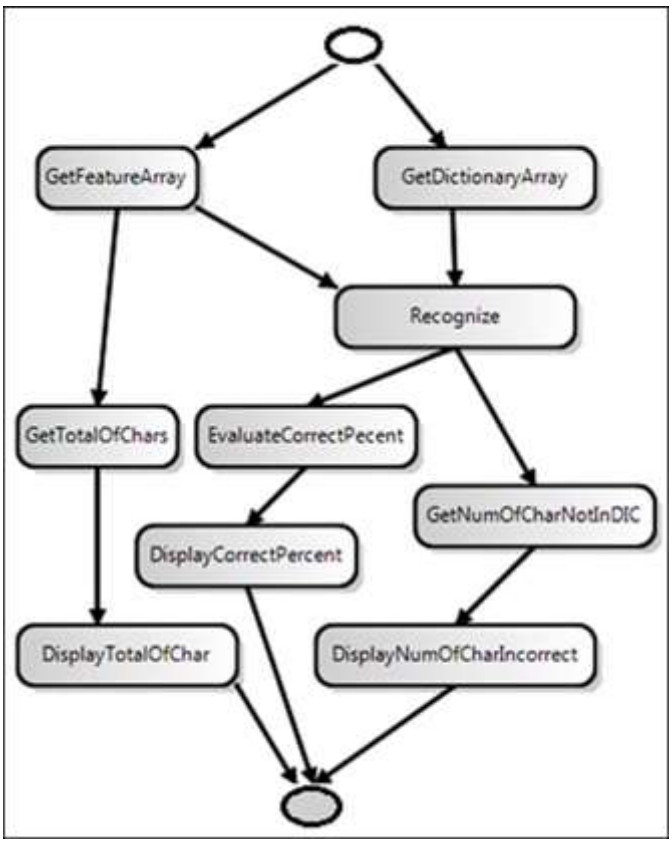
Thuật toán tìm một chuỗi tô-pô từ đồ thị có hướng [60] có độ phức tạp đa thức $O(|U| + |V|)$, với U là tập đỉnh, V là tập cạnh theo định nghĩa (2.9). Tuy nhiên để tìm mọi chuỗi tô-pô trong tập U thì độ phức tạp là $O(|U|! \times (|U| + |V|))$. Để giảm độ phức tạp, chúng tôi xây dựng thuật toán tìm mọi chuỗi tô-pô với độ phức tạp nhỏ hơn. Thuật toán đưa ra này dựa trên ý tưởng: một chuỗi có thứ tự bất kỳ chỉ cần thỏa mãn tập cạnh V là một chuỗi tô-pô. Thuật toán được mô tả như sau:

```
Lặp với mỗi chuỗi tp trong tập có |U|! chuỗi
{
     Gán bienNho bằng true
     Lặp với mỗi cạnh v_i,j trong tập cạnh V
     {
          Nếu u_i đứng sau u_j trong tp thì
          {
               Gán bienNho bằng false
               Kết thúc vòng lặp trong
          }
     }
     Nếu bienNho bằng true thì thêm tp vào tập các chuỗi tô-
pô
}
```

Thuật toán của chúng tôi có độ phức tạp là $O(|U|! \times |V|)$. Trong chương trình, chúng tôi đã lập trình theo thuật toán này để tìm mọi chuỗi tô-pô thỏa mãn đồ thị phụ thuộc. Chi tiết về chương trình tối ưu này được trình bày cụ thể trong *Phụ lục P.1.4*.

v. Ví dụ minh họa phương pháp tối ưu bộ nhớ dựa trên sắp xếp tô-pô

Để minh họa rõ hơn phương pháp tối ưu bộ nhớ trong giai đoạn thiết kế dựa trên sắp xếp tô-pô, chúng tôi trình bày một ví dụ cụ thể về các bước thực hiện trong quá trình tối ưu. Chương trình thử nghiệm là mô-đun nhận dạng chữ Nôm trên điện thoại di động như trình bày trong *Phụ lục P.2.1*. Đầu tiên chúng tôi sử dụng khung làm việc DSL và T4 để xây dựng đồ thị phụ thuộc đặc tả chương trình như trong Hình 2.21. Sử dụng mẫu T4 để tự động chuyển sang đặc tả dạng văn bản của đồ thị phụ thuộc như trong Hình 2.22. Sau đó sử dụng chương trình tối ưu để tìm chuỗi tô-pô có dung lượng bộ nhớ chiếm dụng ít nhất. Hình 2.23 minh họa các chuỗi tô-pô thỏa mãn đồ thị phụ thuộc và Hình 2.24 chỉ ra các chuỗi tô-pô có dung lượng bộ nhớ chiếm dụng nhỏ nhất.



Hình 2.21: Đồ thị phụ thuộc mô-đun nhận dạng chữ Nôm

```
1   [!TaskName]Recognize
2   [!ReturnType]void
3   [!Parameters]string[MAX]
4                   [!$edge]GetFeatureArray>Recognize
5                   [!$edge]GetDictionaryArray>Recognize
6   [!TaskName]GetFeatureArray
7   [!ReturnType]string[MAX]
8   [!Parameters]
9   [!TaskName]EvaluateCorrectPecent
10  [!ReturnType]double
11  [!Parameters]
12                  [!$edge]Recognize>EvaluateCorrectPecent
13  [!TaskName]GetDictionaryArray
14  [!ReturnType]bool
15  [!Parameters]
16  [!TaskName]GetTotalOfChars
17  [!ReturnType]int
18  [!Parameters]
19          [!$edge]GetFeatureArray>GetTotalOfChars
20          [!TaskName]DisplayNumOfCharIncorrect
21  [!ReturnType]bool
22  [!Parameters]TextBox txtIncorrect, int num
23          [!$edge]GetNumOfCharNotInDIC>DisplayNumOfCharIncorrect
24          [!TaskName]GetNumOfCharNotInDIC
```

Hình 2.22: Đặc tả dạng văn bản của đồ thị phụ thuộc

```
560   GetFeatureArray  GetDictionaryArray  Recognize  GetNumOfCharNotInDIC  EvaluateCorrectPecent
561   GetFeatureArray  GetDictionaryArray  Recognize  GetNumOfCharNotInDIC  EvaluateCorrectPecent
562   GetFeatureArray  GetDictionaryArray  Recognize  GetNumOfCharNotInDIC  EvaluateCorrectPecent
563   GetFeatureArray  GetDictionaryArray  Recognize  GetNumOfCharNotInDIC  EvaluateCorrectPecent
564   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
565   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
566   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
567   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
568   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
569   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
570   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
571   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
572   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
573   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetNumOfCharNotInDIC
574   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  GetN
575   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  GetN
576   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  GetN
577   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Disp
578   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Disp
579   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  GetN
580   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  GetN
581   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  GetN
582   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Disp
583   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Disp
584   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Disp
585   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Disp
586   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
587   GetFeatureArray  GetDictionaryArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
```

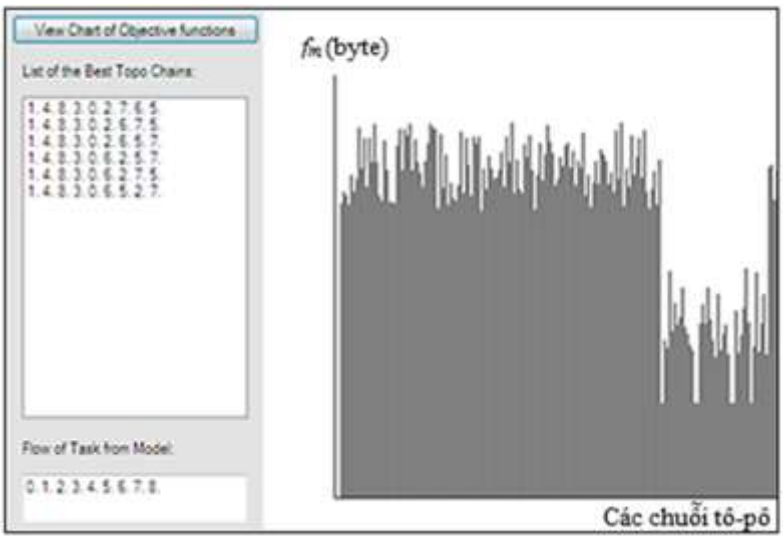
Hình 2.23: Minh họa một phần tập các chuỗi tô-pô

```
1   GetDictionaryArray  GetFeatureArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
2   GetDictionaryArray  GetFeatureArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
3   GetDictionaryArray  GetFeatureArray  Recognize  EvaluateCorrectPecent  DisplayCorrectPercent
4   GetDictionaryArray  GetFeatureArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Displ
5   GetDictionaryArray  GetFeatureArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Displ
6   GetDictionaryArray  GetFeatureArray  Recognize  EvaluateCorrectPecent  GetTotalOfChars  Displ
```

Hình 2.24: Các chuỗi tô-pô có dung lượng bộ nhỏ chiếm dụng nhỏ nhất

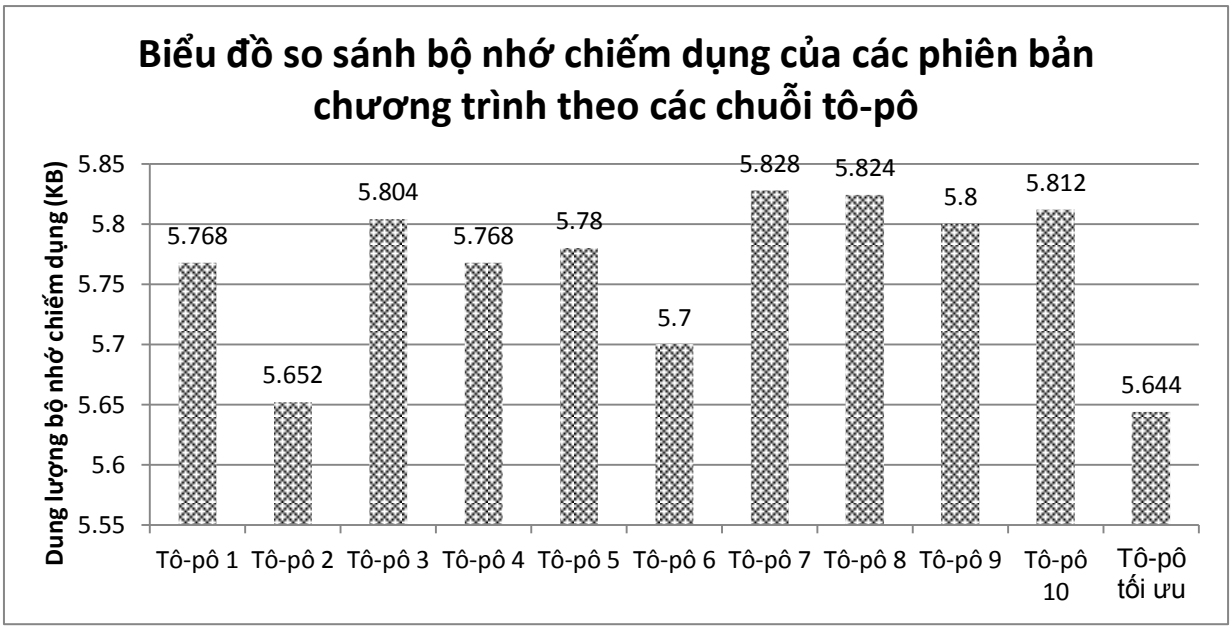
vi. Thực nghiệm

Thực nghiệm được tiến hành theo hai giai đoạn. Trong giai đoạn một, chúng tôi thiết kế đồ thị phụ thuộc của mô-đun nhận dạng chữ Nôm như đã minh họa trong Hình 2.21. Thực hiện chương trình tối ưu, chúng tôi thu được 294 chuỗi tô-pô từ hoán vị của tập tác vụ và tìm được sáu chuỗi tô-pô có mức độ chiếm dụng bộ nhớ nhỏ nhất như chỉ ra trong Hình 2.25. Phần bên phải trong Hình 2.25 minh họa đồ thị chiếm dụng bộ nhớ của chương trình theo các chuỗi tô-pô.



Hình 2.25: Biểu đồ chiếm dụng bộ nhớ theo các chuỗi tô-pô

Trong giai đoạn hai, chúng tôi tiến hành xây dựng mô-đun nhận dạng chữ Nôm trên điện thoại di động và hoán vị thứ tự thực hiện các hàm theo một chuỗi tô-pô tốt nhất và theo 10 chuỗi tô-pô ngẫu nhiên khác với chuỗi tô-pô tốt nhất thu được 11 phiên bản của chương trình. Thực thi 11 phiên bản này trên cùng một môi trường và thống kê dung lượng bộ nhớ chiếm dụng. Biểu đồ trong Hình 2.26 chỉ ra dung lượng bộ nhớ chiếm dụng của mỗi phiên bản theo các chuỗi tô-pô tương ứng.



Hình 2.26: Mức chiếm dụng bộ nhớ thực tế

Kết quả tổng hợp về dung lượng bộ nhớ chiếm dụng trình bày trong thực nghiệm này khẳng định ảnh hưởng của sắp xếp tô-pô đến mức chiếm dụng bộ nhớ của chương trình trong quá trình thực thi. Phân tích ảnh hưởng để tìm ra chuỗi tô-pô chiếm dụng bộ nhớ nhỏ nhất có nhiều ý nghĩa trong không gian bộ nhớ giới hạn của phần mềm nhúng. Đồng thời thực nghiệm cũng cho thấy sự phù hợp của hàm đánh giá dung lượng bộ nhớ chiếm dụng với kết quả thực tế.

vii. Đánh giá phương pháp

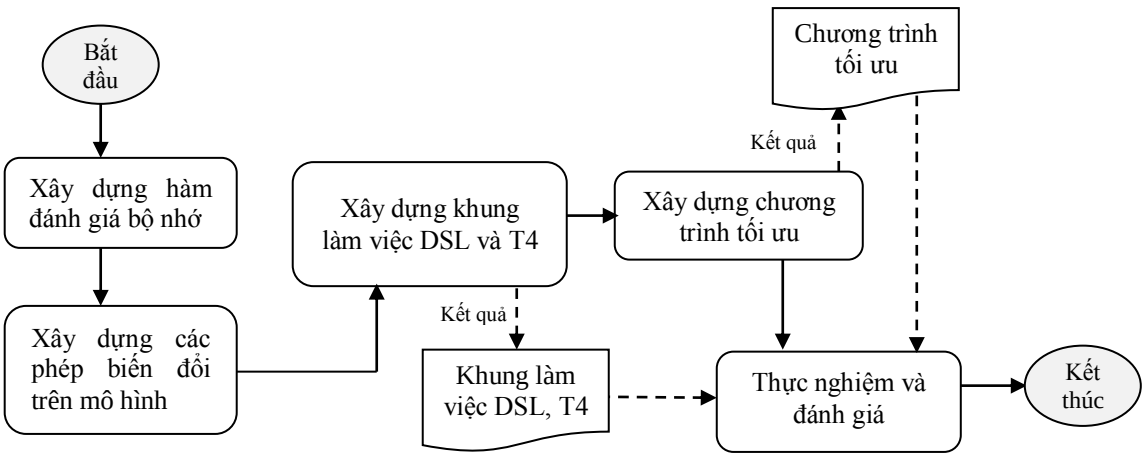
Phương pháp tối ưu này vẫn còn một số vấn đề cần giải quyết đó là không gian hoán vị của tập tác vụ lớn khi có nhiều tác vụ và sắp xếp tô-pô chỉ áp dụng với đồ thị không có chu trình. Để giải quyết vấn đề đồ thị có chu trình, chúng tôi đưa ra giải pháp là chuyển đổi đồ thị có chu trình về đồ thị không chu trình bằng cách gộp các tác vụ trong chu trình thành một tác vụ lớn hơn. Khi có nhiều tác vụ, không gian hoán vị lớn, chúng tôi cũng đề xuất giải pháp là không sử dụng thuật toán vét cạn để tìm mọi chuỗi tô-pô như đã trình bày mà sử dụng các thuật toán tìm kiếm kinh nghiệm, gần tối ưu như thuật toán di truyền.

Tuy trong nghiên cứu, đồ thị tác vụ phụ thuộc được tạo ra trong giai đoạn thiết kế và phạm vi ứng dụng không nhiều nhưng phương pháp tối ưu này cũng hết sức có ý nghĩa trong bài toán tối ưu tổng thể. Phương pháp này có thể sử dụng làm cơ sở, kết hợp với kỹ nghệ ngược để thực hiện tối ưu hóa cho phần mềm nhúng như sau: từ mã nguồn hoặc mã hợp ngữ sinh ngược ra đồ thị phụ thuộc và sắp xếp lại thứ tự thực hiện để tối ưu bộ nhớ chiếm dụng mà không làm thay đổi ngữ nghĩa.

2.2.2. Tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình

i. Ý tưởng và quy trình nghiên cứu

Kế thừa hướng tiếp cận tối ưu dựa trên biến đổi mô hình để tối ưu hiệu năng đã trình bày, trong phần này, chúng tôi phát triển phương pháp tối ưu bộ nhớ trong giai đoạn thiết kế dựa trên chuyển đổi mô hình với DSL và T4. 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. Để xây dựng phương pháp tối ưu này, chúng tôi tiến hành nghiên cứu và thực nghiệm theo quy trình trong Hình 2.27.



Hình 2.27: Quy trình nghiên cứu và thực nghiệm phương pháp tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình

ii. Xây dựng hàm đánh giá dung lượng bộ nhớ chiếm dụng

Mô hình dữ liệu trừu tượng chúng tôi xây dựng và sử dụng trong nghiên cứu này là biểu đồ lớp rút gọn, không xét tính kế thừa và phạm vi truy xuất cũng như các lớp trừu tượng và các giao diện. Để làm cơ sở cho các phép chuyển đổi và tối ưu trên mô hình, chúng tôi cải tiến các công thức từ (2.1) đến (2.6) để xây dựng hàm đánh giá bộ nhớ chiếm dụng. Trong phần này, các thành phần trong mô hình được bổ sung thuộc tính mới là tần số sử dụng n và các công thức tính các độ đo S_1 đến S_6 được thay đổi như trong các công thức từ (2.12) đến (2.17).

$$S_1 = \sum_{i=1}^{|C|} \sum_{j=1}^{|V_S^i|} n_j \times \text{size}(a_s^j) \quad (2.12)$$

$$S_2 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_S^i|} n_j \times \text{size}(v_r^j) \quad (2.13)$$

$$S_3 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_S^i|} n_j \times \sum_{k=1}^{|P_m^j|} \text{size}(p_k) \quad (2.14)$$

$$S_4 = \sum_{i=1}^{|C|} \sum_{j=1}^{|V_O^i|} n_j \times \text{size}(a_o^j) \quad (2.15)$$

$$S_5 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_O^i|} n_j \times \text{size}(v_r^j) \quad (2.16)$$

$$S_6 = \sum_{i=1}^{|C|} \sum_{j=1}^{|M_O^i|} n_j \times \sum_{k=1}^{|P_m^j|} \text{size}(p_k) \quad (2.17)$$

Dựa trên các độ đo từ S_1 đến S_6 đã được cải tiến theo công thức trên, chúng tôi phân tích sự ảnh hưởng của các độ đo này đến dung lượng bộ nhớ chiếm dụng để xây dựng hàm đánh giá bộ nhớ. Trong quá trình thực thi một chương trình hướng đối tượng, các thành phần tĩnh sẽ được cấp phát bộ nhớ và tồn tại đến khi kết thúc chương trình; các thành phần thuộc đối tượng bắt đầu được cấp phát bộ nhớ khi đối tượng được tạo và tồn tại đến khi hủy đối tượng; các tham số và các biến cục bộ bên trong phương thức được cấp phát bộ nhớ khi phương thức được gọi và bị thu hồi bộ nhớ khi phương thức kết thúc. Do đó, có thể thiết lập hệ số phụ thuộc của mức chiếm dụng bộ nhớ vào S_3 và S_6 là 1, hệ số phụ thuộc vào S_4 và S_5 là 2, hệ số phụ thuộc vào S_1 và S_2 là 3. Theo đó, hàm đánh giá mức chiếm dụng bộ nhớ f_m có thể tính theo công thức (2.7) với $\alpha = 3$, $\beta = 1$, $\gamma = 2$ và $\varepsilon = 1$. Đồng thời, chúng tôi cũng không sử dụng f_m để ước lượng dung lượng bộ nhớ chiếm dụng mà chỉ để đánh giá xem mô hình nào chiếm dụng bộ nhớ ít hơn. Hàm đánh giá bộ nhớ cũng được sử dụng để chứng minh các phép biến đổi như thu gọn kiểu dữ liệu, chuyển các thành phần tĩnh thành động và sử dụng để tối ưu trong kỹ nghệ ngược khi tần số n được thống kê dựa trên phân tích mã nguồn.

iii. Xây dựng các phép biến đổi mô hình để tối ưu bộ nhớ

Trong nghiên cứu [7, 8], Anne, K. đã trình bày các phép biến đổi trên mô hình như phân chia cấu trúc, gộp cấu trúc, loại bỏ thuộc tính dư thừa, chuyển đổi thuộc tính, tạo

thuộc tính tạm, v.v. Tuy nhiên tác giả chưa chứng minh tính đúng đắn và cũng không đưa ra giải pháp cũng như các tiêu chí để thực hiện tự động các biến đổi này. 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.

- **Phân chia cấu trúc**

Anne, K. đưa ra phép biến đổi này nhằm phân chia một cấu trúc dữ liệu trừu tượng thành nhiều cấu trúc con theo tần số truy xuất thành phần dữ liệu và tần số sử dụng các hàm. Các thành phần có tần số không lệch nhau nhiều sẽ được tách thành cấu trúc riêng. Tuy nhiên, vấn đề cốt yếu của phân chia là xác định tần số, phương pháp phân chia, sự tương đương của các mô hình và áp dụng phương pháp thì tác giả vẫn chưa đề cập. Do đó, để cải tiến và ứng dụng, chúng tôi giới hạn bài toán, chứng minh phương pháp, phát triển phương pháp phân chia dựa trên thuật toán phân cụm K-mean [43] và áp dụng cụ thể trong biểu đồ lớp.

Để chứng minh phương pháp này, chúng tôi mô tả một cấu trúc x trong mô hình dữ liệu như trong biểu thức (2.18) và giả thiết như sau: phạm vi truy xuất các thành phần là toàn cục, các cấu trúc được cấp phát động và thời gian cư trú trong bộ nhớ của một byte dữ liệu là t_0 trong một thao tác truy xuất thành phần dữ liệu hoặc kết quả trả về từ một hàm.

$$x = <\{a_i \mid i = 1..A_o\}, \{m_j \mid j = 1..M_o\}> \quad (2.18)$$

Trong đó:

- a_i là một thành phần dữ liệu của x
- A_o là tổng số thành phần dữ liệu trong x
- m_j là một hàm trong x
- M_o là tổng số hàm trong x.

Phép biến đổi này được mô tả như trong biểu thức (2.19). Trong đó K là số cấu trúc mới được tạo ra.

$$x \rightarrow \{x_i \mid i = 1..K\} \quad (2.19)$$

Giả sử size() là hàm lấy kích thước của một thành phần dữ liệu hoặc một hàm, Size(x) là kích thước bộ nhớ chiếm dụng theo thời gian của cấu trúc x. Vì thời gian chiếm dụng bộ nhớ của cấu trúc x phải bằng thời gian chiếm dụng của thành phần có tần số lớn nhất nên Size(x) có thể được tính như công thức (2.20).

$$\begin{aligned}\text{Size}(x) &= t_0 \times \max(\max_{i \le Ao}(n_i), \max_{j \le Mo}(n_j)) \times (\sum_{i=1}^{A_0} \text{size}(a_i) + \sum_{j=1}^{M_0} \text{size}(m_j)) \\ &= t_0 \times \max_{i \le (A_0 + M_0)}(n_i) \times (\sum_{i=1}^{A_0} \text{size}(a_i) + \sum_{j=1}^{M_0} \text{size}(m_j))\end{aligned} \quad (2.20)$$

Khi cấu trúc *x* được chia thành *K* cấu trúc, tổng bộ nhớ chiếm dụng gọi là Size*(*x*) và được tính như công thức (2.21).

$$\begin{aligned}\text{Size}^*(x) &= t_0 \times \sum_{i=1}^{K} \max(\max_{j \le A_0^i}(n_j), \max_{j \le M_0^i}(n_j)) \times (\sum_{j=1}^{A_0^i} \text{size}(a_j) + \sum_{j=1}^{M_0^i} \text{size}(m_j)) \\ &\le t_0 \times \max_{i \le K}(\max(\max_{j \le A_0^i}(n_j), \max_{j \le M_0^i}(n_j))) \times \sum_{i=1}^{K} (\sum_{j=1}^{A_0^i} \text{size}(a_j) + \sum_{j=1}^{M_0^i} \text{size}(m_j))\end{aligned} \quad (2.21)$$

Trong đó A_o^i là số thuộc tính của cấu trúc thứ *i* và M_o^i là số phương thức của cấu trúc thứ *i*.

Ta có:

$$A_0 = \sum_{i=1}^{K} A_0^i \text{ và } M_0 = \sum_{i=1}^{K} M_0^i$$

Suy ra: $$\max_{i \le (A_0 + M_0)}(n_i) = \max_{i \le K}(\max(\max_{j \le A_0^i}(n_j), \max_{j \le M_0^i}(n_j))) \quad (2.22)$$

và

$$\sum_{i=1}^{K} (\sum_{j=1}^{A_0^i} \text{size}(a_j) + \sum_{j=1}^{M_0^i} \text{size}(m_j)) = \sum_{i=1}^{A_0} \text{size}(a_i) + \sum_{i=1}^{M_0} \text{size}(m_i) \quad (2.23)$$

Do đó, từ các công thức (2.20), (2.21), (2.22) và (2.23) suy ra:

$$\text{Size}(x) \ge \text{Size}^*(x)$$

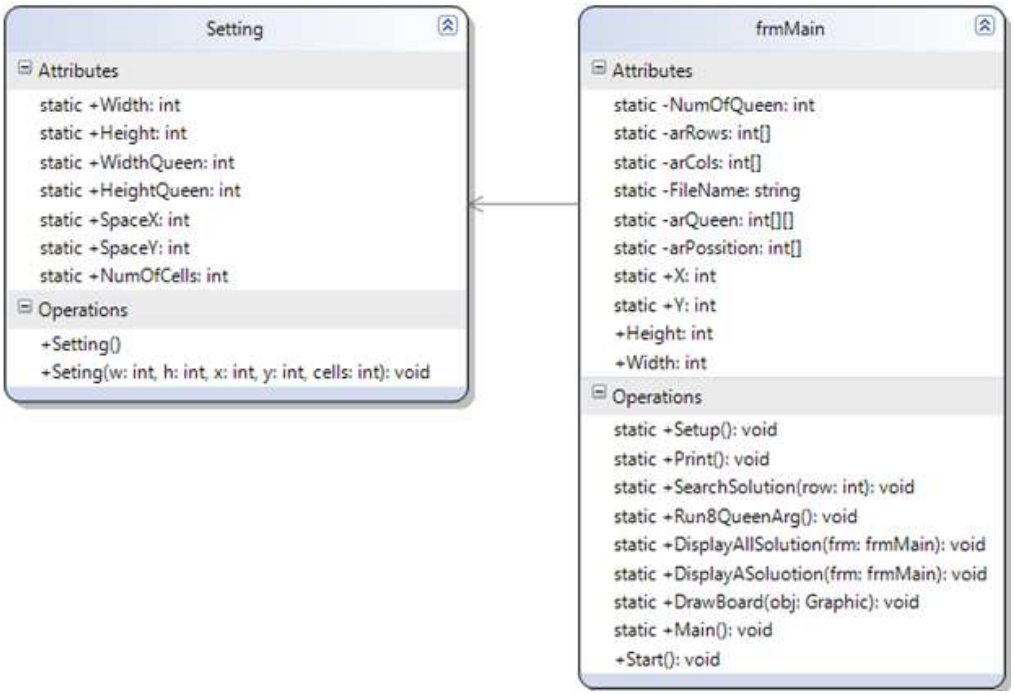
- **Phép biến đổi gộp cấu trúc**

Phép biến đổi này dựa trên đánh giá toàn bộ mô hình dữ liệu sau khi đã phân tích các cấu trúc. Phép biến đổi phân chia dựa trên phân tích từng cấu trúc để phân chia. Điều này dẫn đến có thể tồn tại nhiều cấu trúc khác nhau mà tần số truy xuất của các thành phần lại như nhau. Khi đó phép biến đổi gộp cấu trúc được sử dụng để hợp nhất các cấu

trúc có cùng tần số truy xuất của các thành phần thành một cấu trúc. Theo đó, phép biến đổi này làm cho cấu trúc của mô hình dữ liệu chặt chẽ hơn. Chúng ta có thể chứng minh phép biến đổi gộp cấu trúc tương tự như phép phân chia cấu trúc bằng cách hợp nhất mọi cấu trúc trong mô hình thành một cấu trúc sau đó mới phân chia. Khi đó bài toán được đưa về dạng phân chia cấu trúc đã chứng minh.

iv. Ví dụ minh họa phương pháp tối ưu bộ nhớ dựa trên chuyển đổi mô hình

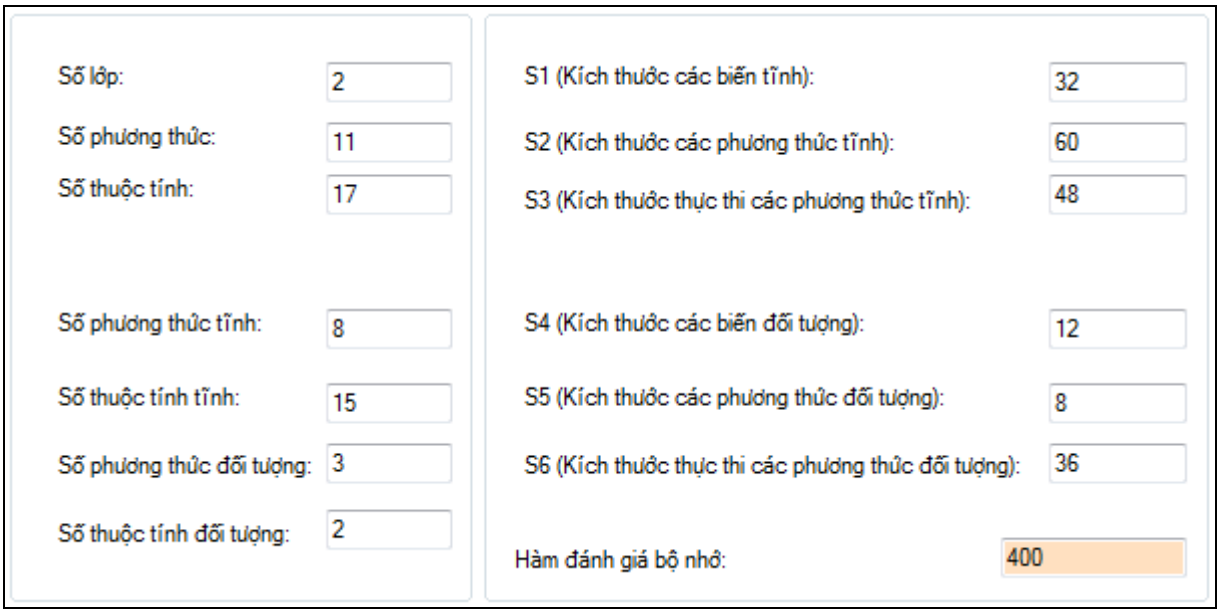
Nội dung phần này nhằm minh họa rõ hơn các bước chuyển đổi mô hình để tối ưu thông qua chương trình 8 quân Hậu. Chương trình này được trình bày cụ thể trong *Phụ lục P.2.4*. Mô hình dữ liệu ban đầu của chương trình được thiết kế dựa trên khung làm việc DSL và T4 như trong Hình 2.28. Sử dụng mẫu T4 để sinh đặc tả dạng văn bản như trong Hình 2.29. Kết quả phân tích mô hình và giá trị hàm đánh giá bộ nhớ được thể hiện trong Hình 2.30. Lựa chọn các phép chuyển đổi để đưa về mô hình để tối ưu như trong Hình 2.32 và hiển thị biểu đồ so sánh mức chiếm dụng như trong Hình 2.31.



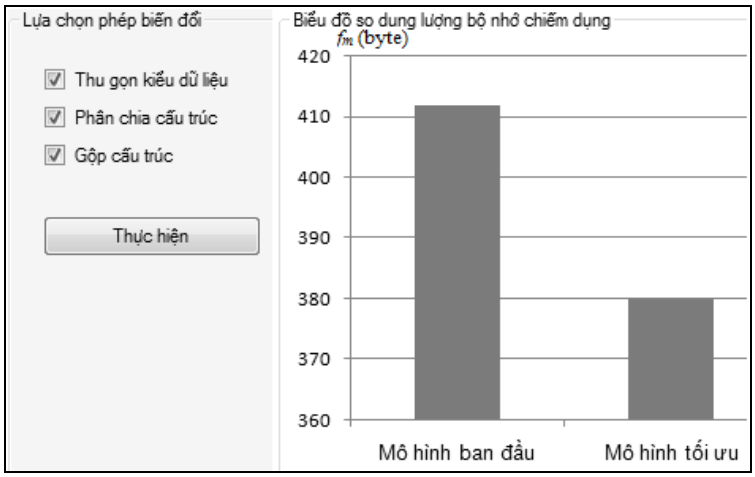
Hình 2.28: Mô hình dữ liệu ban đầu

```
1  @Class:Setting
2          @@Method:+Setting() _@@@@mFrequency: 1
3          @@Method:+Seting(w: int, h: int, x: int, y: int, cells: int):
4
5          @@@Attribute:static +Width: int     _@@@@aFrequency: 1
6          @@@Attribute:static +Height: int    _@@@@aFrequency: 1
7          @@@Attribute:static +WidthQueen: int    _@@@@aFrequency: 1
8          @@@Attribute:static +HeightQueen: int   _@@@@aFrequency: 1
9          @@@Attribute:static +SpaceX: int    _@@@@aFrequency: 1
10         @@@Attribute:static +SpaceY: int    _@@@@aFrequency: 1
11         @@@Attribute:static +NumOfCells: int    _@@@@aFrequency: 10
```

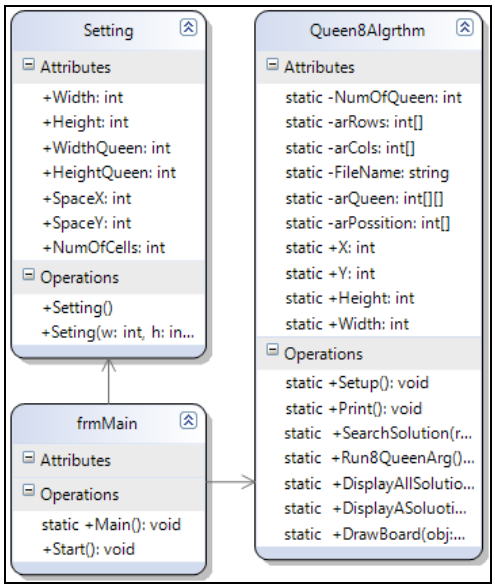
Hình 2.29: Minh họa đặc tả dạng văn bản của mô hình ban đầu



Hình 2.30: Phân tích mô hình và tính giá trị hàm đánh giá bộ nhớ



Hình 2.31: Lựa chọn phép biến đổi và so sánh mức chiếm dụng bộ nhớ



Hình 2.32: Mô hình tối ưu bộ nhớ chiếm dụng

v. Thực nghiệm

Để kiểm chứng các phép chuyển đổi tối ưu bộ nhớ chiếm dụng cũng như tính đúng đắn của hàm đánh giá bộ nhớ chúng tôi sẽ tiến hành thực nghiệm với ba chương trình là mô-đun nhận dạng chữ Nôm trên điện thoại di động, tháp Hà Nội và 8 quân Hậu. Các chương trình này được trình bày chi tiết trong *Phụ lục P.2.1, P.2.3* và *P.2.4*. Các bước thực nghiệm được tiến hành theo ví dụ minh họa trong phần trước. Ngoài ra, để kiểm chứng kết quả tối ưu theo hàm đánh giá bộ nhớ và kết quả thực tế, chúng tôi xây dựng các chương trình thử nghiệm theo mô hình ban đầu và mô hình tối ưu. Các chương trình thử

nghiệm thực tế được lập trình theo cùng thuật toán và chỉ khác nhau về cấu trúc. Bảng 2.8 tổng hợp các giá trị các hàm đánh giá bộ nhớ và dung lượng bộ nhớ chiếm dụng thực tế.

Bảng 2.8. 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

vi. Đánh giá phương pháp

Kết quả thực nghiệm trong Bảng 2.8 cho thấy kết quả tối ưu dựa bộ nhớ dựa trên biến đổi mô hình phù hợp với kết quả thực tế. Phương pháp tối ưu này giúp cải tiến dung lượng bộ nhớ đáng kể trong giai đoạn thiết kế phần mềm nhúng. Chất lượng tối ưu của phương pháp này phụ thuộc chủ yếu vào các phép chuyển đổi. So với phương pháp ban đầu của Anne, K. [7, 8], trong cải tiến này, chúng tôi đã bổ sung thêm phép biến đổi thu gọn kiểu dữ liệu, chứng minh tính đúng đắn của phép biến đổi phân chia cấu trúc và tiến hành tự động dựa trên DSL và T4.

Khi sử dụng phương pháp tối ưu này, tính đóng gói của hệ thống có thể bị phá vỡ và cũng chỉ có thể sử dụng trong giới hạn là các thành phần dữ liệu, các hàm bình đẳng và có thể truy xuất từ các cấu trúc khác nhau. Tuy nhiên phương pháp này cũng có ý nghĩa vì với phần mềm nhúng thì mục tiêu về bộ nhớ và hiệu năng là cần thiết hơn các độ đo chất lượng khác. Đồng thời, nghiên cứu này cũng là cơ sở để kết hợp với kỹ nghệ ngược nhằm tái cấu trúc để tối ưu mức chiếm dụng bộ nhớ của phần mềm nhúng.

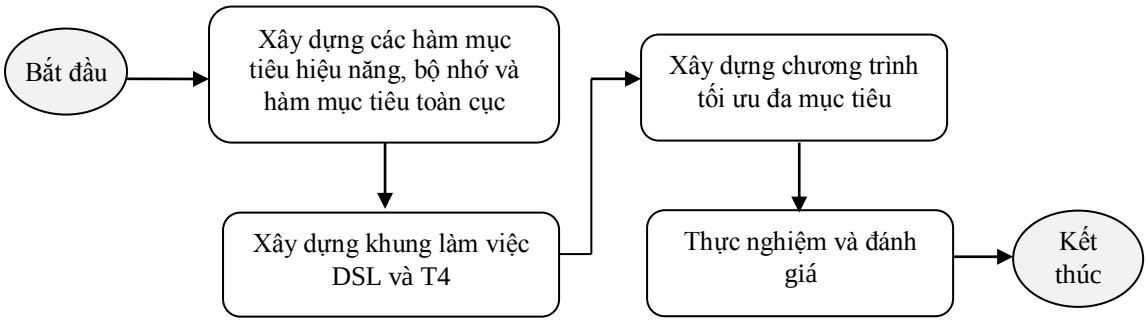
2.3. Tối ưu đa mục tiêu dựa trên biểu đồ lớp

Trong quá trình tối ưu phần mềm nhúng, không thể thỏa mãn đồng thời hai mục tiêu tối ưu hiệu năng và bộ nhớ. Cải tiến hiệu năng có thể làm tăng dung lượng bộ nhớ chiếm dụng và ngược lại. Vấn đề này có thể được giải quyết dựa trên tối ưu đa mục tiêu. Tối ưu đa mục tiêu trong thiết kế phần mềm nhúng là một hướng nghiên cứu mới và còn nhiều thách thức. Dựa trên cơ sở lý thuyết và thực nghiệm về tối ưu hiệu năng và tối ưu bộ nhớ, trong phần này, chúng tôi đề xuất và phát triển phương pháp tối ưu đa mục tiêu dựa trên biểu đồ lớp và nguyên lý Pareto.

i. Ý tưởng và quy trình nghiên cứu

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 có phân phối cân bằng nhất giữa hiệu năng và bộ nhớ chiếm dụng. Mô hình được áp dụng trong nghiên cứu này là biểu đồ lớp rút gọn. Chúng tôi không đặc tả giao diện và lớp trừu tượng cũng như không xét đến tính kế thừa và mức che dấu thông tin từ biểu đồ lớp. Để thiết kế và chuyển tự động sang đặc tả dạng văn bản của mô hình nhằm trích xuất các thông tin mong muốn, chúng tôi tiếp cận theo cách sử dụng DSL và T4.

Quy trình nghiên cứu và thực nghiệm phương pháp tối ưu này được chỉ ra trong Hình 2.33. Đầu tiên, chúng tôi phân tích biểu đồ lớp để xây dựng hàm mục tiêu hiệu năng, hàm mục tiêu bộ nhớ và hàm mục tiêu toàn cục. Để thiết kế mô hình dữ liệu và sinh đặc tả tự động, chúng tôi xây dựng khung làm việc DSL và T4. Sau đó, chúng tôi sử dụng khung làm việc DSL và T4 để thiết kế biểu đồ lớp như đã trình bày trong phần tối ưu hiệu năng và bộ nhớ. Khung làm việc này được trình bày chi tiết trong *Phụ lục P.1.1*. Tiếp theo, chúng tôi xây dựng chương trình tối ưu đa mục tiêu dựa trên nguyên lý Pareto như trong *Phụ lục P.1.3* để tìm biểu đồ lớp cân bằng nhất. Cuối cùng chúng tôi tiến hành thực nghiệm và đánh giá phương pháp.



Hình 2.33: Quy trình tối ưu đa mục tiêu dựa trên biểu đồ lớp

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 định nghĩa lại để phù hợp với bài toán tối ưu đa mục tiêu.

- **Kích thước tham số của các phương thức tĩnh**

Độ đo này là kích thước bộ nhớ sử dụng khi thực thi một phương thức tĩnh. Khi phương thức tĩnh được gọi thực thi, đầu tiên cần cấp phát bộ nhớ để lưu trữ các tham số

truyền vào. Do đó, dựa trên các tham số trong Bảng 2.1, độ đo này có thể được định nghĩa như công thức (2.24).

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

- **Kích thước tham số của các phương thức đối tượng**

Kích thước tham số của các phương thức đối tượng là tổng kích thước bộ nhớ cần thiết cấp phát để chứa các tham số truyền vào khi thực thi phương thức. Theo các tham số trong Bảng 2.1, độ đo này có thể được định nghĩa như công thức (2.25).

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

- **Hàm mục tiêu hiệu năng**

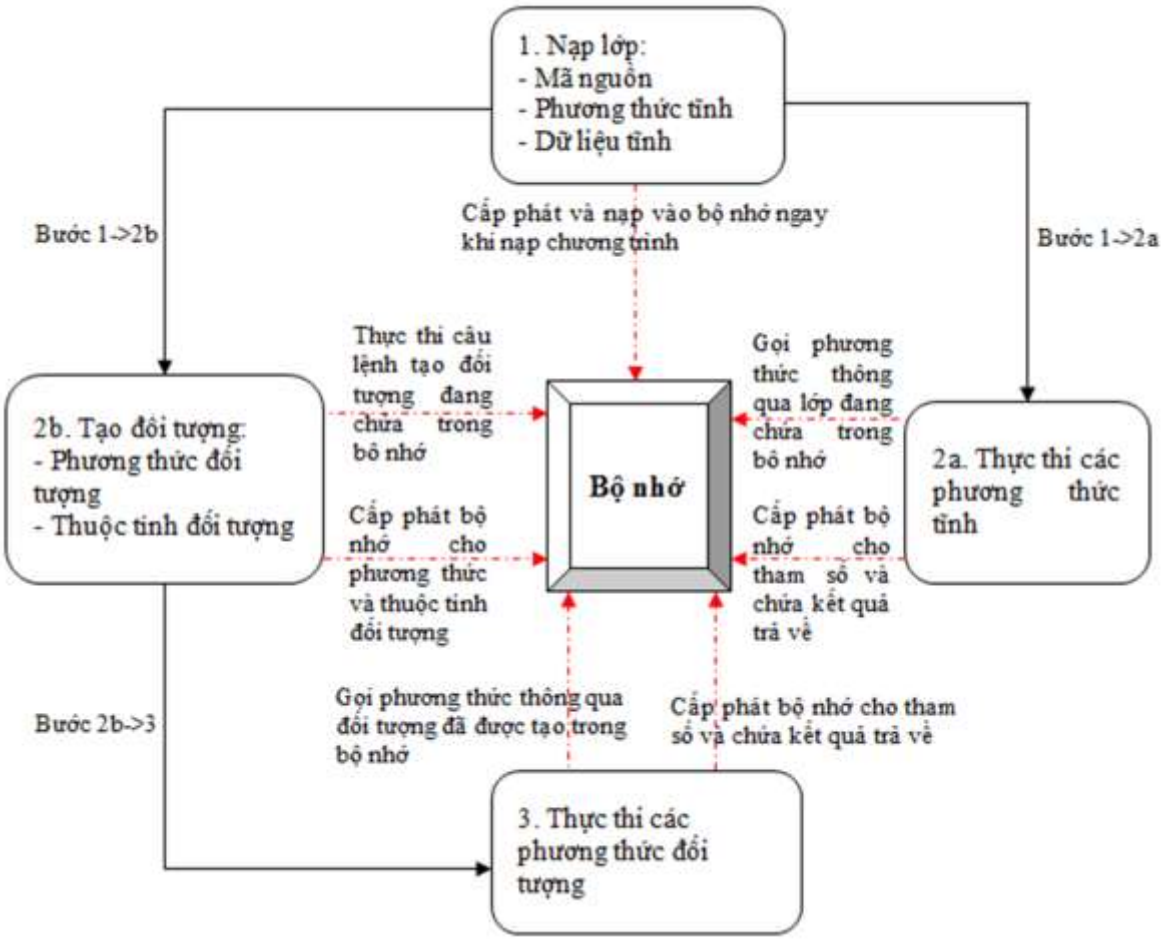
Dựa vào các độ đo đã định nghĩa, chúng tôi sẽ xây dựng hàm mục tiêu hiệu năng, mục tiêu bộ nhớ và hàm mục tiêu toàn cục để thực hiện nguyên lý Pareto nhằm lựa chọn mô hình cân bằng giữa hiệu năng và bộ nhớ. Theo nguyên lý Pareto, khi tăng giá trị của tiêu chí tối ưu này mà không làm giảm các tiêu chí tối ưu khác thì hệ thống thuộc miền tối ưu Pareto. Nghĩa là, trong quá trình tối ưu cần đưa các giá trị trong không gian tìm kiếm về miền tối ưu Pareto. Trong quá trình thực thi một chương trình hướng đối tượng, mỗi loại thuộc tính và phương thức sẽ được cấp phát bộ nhớ và được gọi theo cách khác nhau ảnh hưởng đến hiệu năng và mức chiếm dụng bộ nhớ của chương trình. Để xây dựng các hàm mục tiêu, chúng tôi phân tích hoạt động của một chương trình hướng đối tượng như trong Hình 2.34 và phân tích sự phụ thuộc của hiệu năng vào các thuộc tính, các phương thức và các tham số trong biểu đồ lớp.

Khi sử dụng các thành phần tĩnh, chương trình sẽ thực hiện nhanh hơn khi sử dụng các thành phần động do được cấp phát bộ nhớ tĩnh và được nạp vào bộ nhớ ngay khi nạp chương trình nên hiệu năng sẽ tỉ lệ thuận với $(S_1 + S_2)/(S_4 + S_5)$. Trong một lớp, khi truy xuất dữ liệu, nếu dùng thuộc tính tĩnh thì sẽ nhanh hơn dùng tham số truyền cho các phương thức tĩnh nên hiệu năng tỉ lệ với S_1/S_3. Với một đối tượng, khi truy xuất dữ liệu, nếu dùng các thuộc tính đối tượng thì sẽ nhanh hơn dùng tham số truyền cho các phương thức đối tượng nên hiệu năng tỉ lệ với S_4/S_6. Do đó, hàm mục tiêu hiệu năng có thể được tính theo công thức (2.26).

$$f_1 = \frac{S_1 + S_2}{S_4 + S_5} + \frac{S_1}{S_3} + \frac{S_4}{S_6} \tag{2.26}$$

- **Hàm mục tiêu bộ nhớ**

Sử dụng các thành phần tĩnh làm cho chương trình chiếm dụng bộ nhớ nhiều hơn so với sử dụng các thành phần động do các thành phần tĩnh được cấp phát bộ nhớ tĩnh và chỉ được thu hồi khi chương trình kết thúc nên dung lượng bộ nhớ sử dụng sẽ tỉ lệ thuận với $(S_4 + S_5)/(S_1 + S_2)$. Trong một lớp, với cùng một đối tượng dữ liệu, nếu dùng thành phần dữ liệu tĩnh thì sẽ chiếm dụng bộ nhớ nhiều hơn dùng tham số truyền cho các phương thức tĩnh vì bộ nhớ cấp phát cho thành phần dữ liệu tĩnh chỉ được giải phóng khi chương trình kết thúc. Do đó dung lượng bộ nhớ chiếm dụng tỉ lệ thuận với S_3/S_1. Với một đối tượng, với mỗi thành phần dữ liệu, nếu dùng biến đối tượng thì sẽ tốn nhiều bộ nhớ hơn dùng tham số truyền cho các phương thức động vì các biến đối tượng chỉ bị thu hồi khi hủy đối tượng còn các tham số bị giải phóng bộ nhớ khi phương thức kết thúc. Do đó dung lượng bộ nhớ chiếm dụng tỉ lệ thuận với S_6/S_4. Từ sự phụ thuộc của dung lượng bộ nhớ chiếm dụng vào các độ đo này, hàm mục tiêu bộ nhớ có thể được xây dựng theo công thức (2.27).



Hình 2.34: Mô hình cấp phát, truy xuất bộ nhớ trong quá trình thực thi chương trình hướng đối tượng

$$f_2 = \frac{S_4 + S_5}{S_1 + S_2} + \frac{S_3}{S_1} + \frac{S_6}{S_4} \quad (2.27)$$

- **Hàm mục tiêu toàn cục**

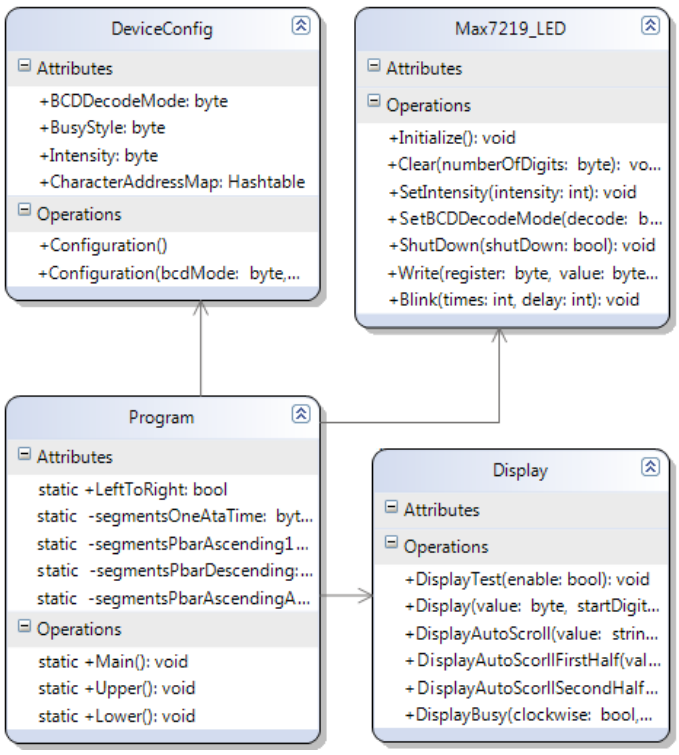
Dựa trên các hàm mục tiêu thành phần, để áp dụng nguyên lý Pareto, chúng tôi xây dựng hàm mục tiêu toàn cục như công thức (2.28).

$$f = w_1 \times f_1 + w_2 \times f_2 \quad (2.28)$$

Trong đó: w_1, w_2 là trọng số của các hàm mục tiêu và $w_1 + w_2 = 1$. Các trọng số này thể hiện độ quan trọng và mức ưu tiên của các mục tiêu tối ưu thành phần. Tùy theo mức ưu tiên khác nhau của các mục tiêu tối ưu thành phần trong mỗi hệ thống, có thể lựa chọn các trọng số khác nhau. Đồng thời khi w_1 hoặc w_2 bằng 0, bài toán tối ưu đa mục tiêu sẽ trở thành tối ưu đơn mục tiêu.

iii. Ví dụ minh họa tối ưu đa mục tiêu dựa trên biểu đồ lớp

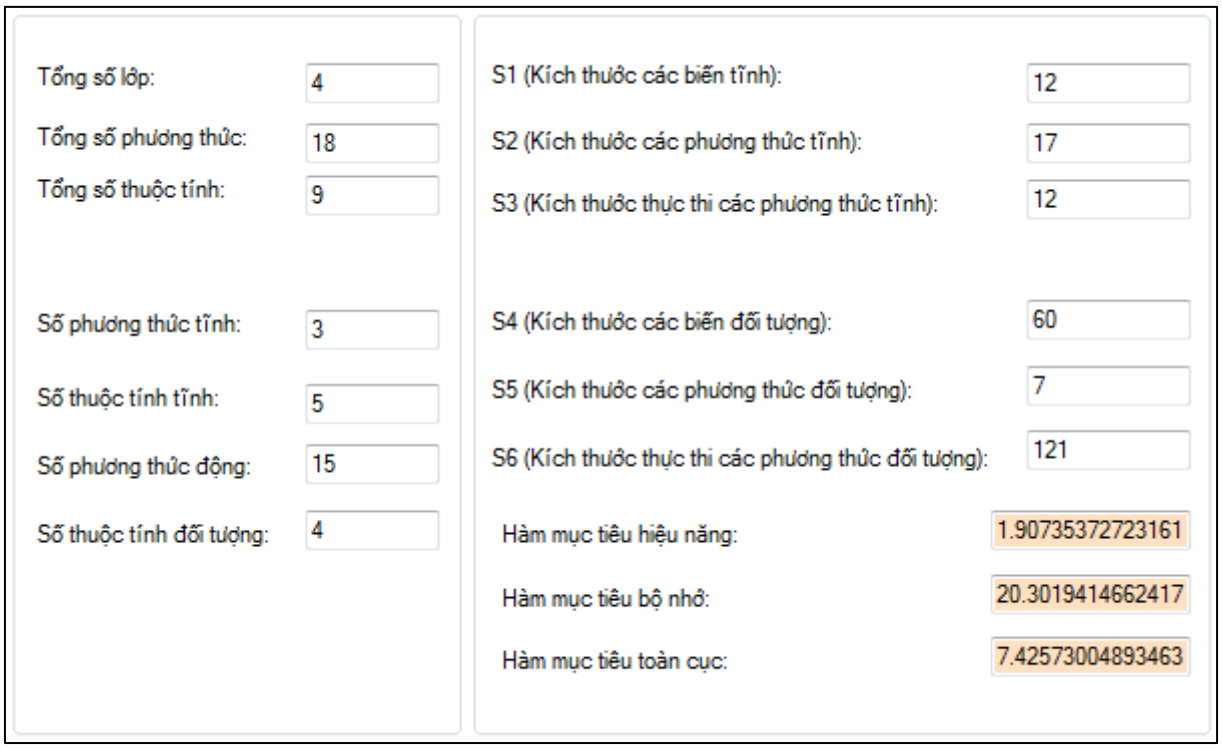
Trong phần này, chúng tôi trình bày một ví dụ cụ thể để minh họa phương pháp tối ưu đa mục tiêu dựa trên biểu đồ lớp đã phát triển. Đầu tiên các biểu đồ lớp được thiết kế dựa trên khung làm việc DSL và T4 như trong Hình 2.35. Các biểu đồ trực quan được chuyển tự động sang đặc tả dạng văn bản như trong Hình 2.36. Kết quả phân tích mô hình để tính toán các độ đo và các hàm mục tiêu như minh họa trong Hình 2.37.



Hình 2.35: Một biểu đồ lớp của chương trình *Netduino_8digit*

```
1  @Class:DeviceConfig
2          @@Method:+Configuration()
3          @@Method:+Configuration(bcdMode:
4          @@@Attribute:+BCDDecodeMode: byte
5          @@@Attribute:+BusyStyle: byte
6          @@@Attribute:+Intensity: byte
7          @@@Attribute:+CharacterAddressMap
8  @Class:Program
9          @@Method:static +Main(): void
10         @@Method:static +Upper(): void
11         @@Method:static +Lower(): void
12         @@@Attribute:static +LeftToRight:
13         @@@Attribute:static -segmentsOneA
14         @@@Attribute:static -segmentsPbar
15         @@@Attribute:static -segmentsPbar
16         @@@Attribute:static -segmentsPbar
17 @Class:Max7219_LED
18         @@Method:+Initialize(): void
19         @@Method:+Clear(numberOfDigits: b
```

Hình 2.36: Đặc tả dạng văn bản của biểu đồ lớp



Hình 2.37: Kết quả phân tích tham số và tính giá trị các hàm mục tiêu

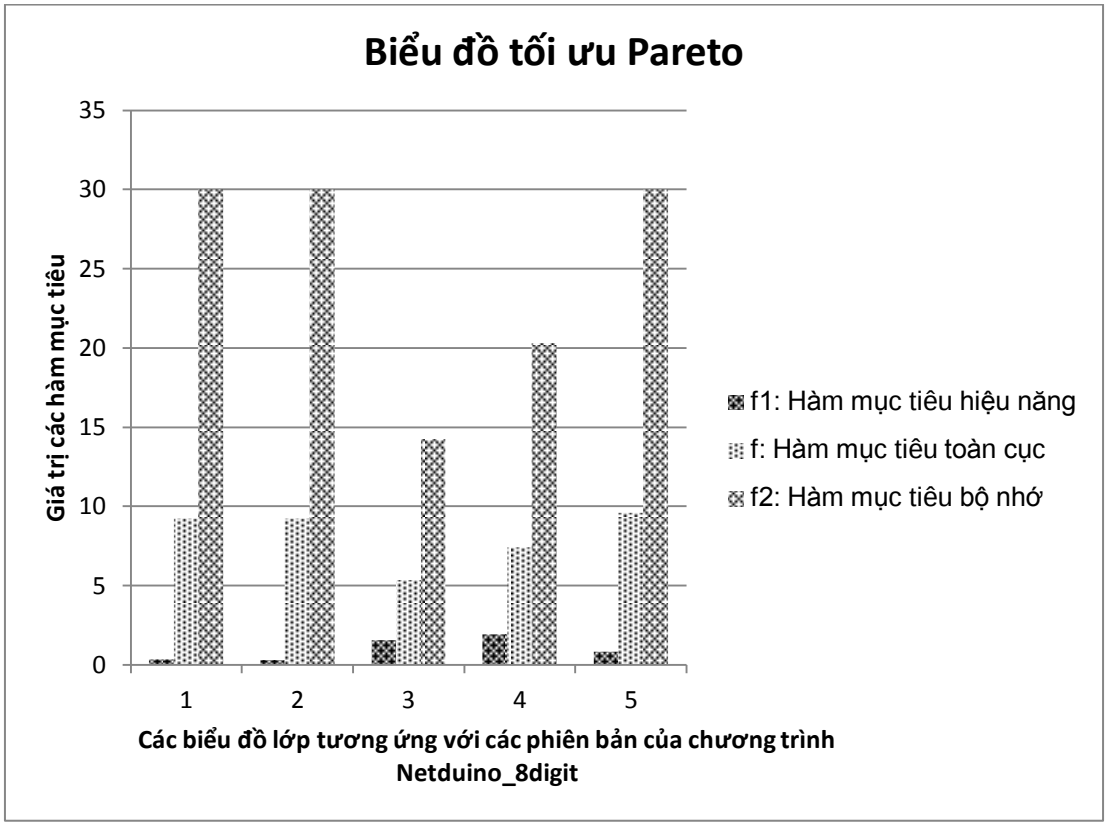
iv. Thực nghiệm

Để thử nghiệm và kiểm chứng phương pháp tối ưu đa mục tiêu dựa trên biểu đồ lớp, chúng tôi tiến hành thực nghiệm với chương trình nhúng điều khiển việc hiển thị đèn LED 8 số *Netduino_8digit*. Chương trình ví dụ này được trình bày chi tiết trong *Phụ lục P.2.5.1*. Trong thực nghiệm này, chúng tôi thiết kế năm biểu đồ lớp khác nhau theo khung làm việc DSL và T4. Năm biểu đồ lớp này sẽ được chuyển tự động sang năm tệp tin đặc tả dạng văn bản. Năm tệp tin này được đưa vào chương trình tối ưu để thực hiện tối ưu đa mục tiêu. Chương trình tối ưu đã được chúng tôi xây dựng như trong *Phụ lục P.1.3*. Trong thực nghiệm, chúng tôi sử dụng trọng số w_1 = 0,7 và trọng số w_2 = 0,3. Kết quả tổng hợp các hàm mục tiêu thành phần và hàm mục tiêu toàn cục được trình bày trong Bảng 2.9. Kết quả tối ưu được thể hiện như biểu đồ trong Hình 2.38. Trong Hình 2.38, biểu đồ lớp thứ 5 có giá trị hàm mục tiêu toàn cục lớn nhất. Do đó biểu đồ lớp này là biểu đồ tối ưu đa mục tiêu cân bằng giữa hiệu năng và dung lượng bộ nhớ chiếm dụng.

v. Đánh giá phương pháp

Phương pháp tối ưu này phù hợp với phát triển phần mềm nhúng hướng đối tượng. Tuy nhiên, trong phạm vi nghiên cứu này, chúng tôi chỉ xét đến hai mục tiêu tối ưu quan trọng của phần mềm nhúng là hiệu năng và bộ nhớ. Trong các nghiên cứu sâu hơn, chúng

tôi sẽ tiếp tục phát triển phương pháp này với các mục tiêu chất lượng phần mềm khác như tính tái sử dụng, tính khả chuyển và tính bao gói.



Hình 2.38: Biểu đồ thống kê kết quả tối ưu Pareto

Bảng 2.9. Tổng hợp tham số, độ đo và giá trị các hàm mục tiêu

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*

2.4. Tổng kết chương

Tối ưu trong giai đoạn thiết kế có vai trò quan trọng trong phát triển phần mềm nhúng. Tối ưu trong giai đoạn thiết kế thường đem lại nhiều lợi ích hơn tối ưu trong các giai đoạn sau. Tuy nhiên, vấn đề tối ưu phần mềm nhúng cũng như tối ưu phần mềm nói chung trong giai đoạn thiết còn nhiều thách thức. Nghiên cứu về tối ưu phần mềm nhúng

trong giai đoạn thiết kế là một hướng nghiên cứu triển vọng và còn nhiều vấn đề cần giải quyết. Trong chương này, chúng tôi đã đề xuất một số cải tiến cũng như phát triển một số phương pháp tối ưu mới theo tiêu chí tối ưu hiệu năng, bộ nhớ và tối ưu đa mục tiêu.

Về tiêu chí tối ưu hiệu năng, chúng tôi đã đề xuất và phát triển mới phương pháp tối ưu hiệu năng dựa trên biểu đồ lớp và cải tiến phương pháp tối ưu hiệu năng dựa trên chuyển đổi mô hình. Trong phương pháp tối ưu hiệu năng dựa trên biểu đồ lớp, chúng tôi đã phân tích biểu đồ lớp, xây dựng các độ đo và xây dựng hàm đánh giá hiệu năng làm tiêu chí để lựa chọn mô hình tối ưu. Để thiết kế mô hình và tự động sinh đặc tả dạng văn bản, chúng tôi đã xây dựng khung làm việc DSL và T4. Để thử nghiệm phương pháp tối ưu này, chúng tôi đã xây dựng chương trình tối ưu nhận đầu vào là tập các mô hình thiết kế được đặc tả theo dạng văn bản để phân tích tham số, tính các độ đo và hàm đánh giá hiệu năng nhằm lựa chọn mô hình tốt nhất. Phương pháp tối ưu mới này đã được công bố trong công trình [CT9]. Trong phương pháp tối ưu hiệu năng dựa trên chuyển đổi mô hình, chúng tôi cũng phân tích biểu đồ lớp, xây dựng và chứng minh các phép biến đổi trên mô hình để đưa mô hình dữ liệu ban đầu về mô hình dữ liệu tối ưu. Để có thể tiến hành một số bước tự động, chúng tôi cũng xây dựng khung làm việc DSL và T4 cho phép thiết kế mô hình và sinh đặc tả tự động. Ngoài ra, về tiêu chí tối ưu hiệu năng, theo cách tiếp cận dựa trên DSL và T4, chúng tôi cũng đã đề xuất và phát triển phương pháp tối ưu hiệu năng dựa trên sinh mã mô phỏng. Nghiên cứu này đã được công bố trong công trình [CT6].

Về tiêu chí tối ưu bộ nhớ, chúng tôi đã đề xuất và phát triển phương pháp mới là tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô và đưa ra cải tiến cho phương pháp tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình. Trong phương pháp tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô, chúng tôi phát triển khung làm việc DSL và T4 cho phép đặc tả phần mềm nhúng như một đồ thị tác vụ phụ thuộc và tự động sinh đặc tả dạng văn bản từ đồ thị phụ thuộc. Các tác vụ của phần mềm nhúng có thể thực hiện theo các chuỗi tô-pô khác nhau thỏa mãn đồ thị phụ thuộc mà không làm thay đổi kết quả của chương trình. Chương trình tối ưu dựa vào hàm đánh giá mức chiếm dụng bộ nhớ trên mỗi chuỗi tô-pô để tìm chuỗi tô-pô tốt nhất. Các nội dung liên quan đến phương pháp đề xuất này đã được xuất bản trong công trình [CT2] và [CT10]. Trong phương pháp tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình, chúng tôi đã chứng minh tính đúng đắn của phép biến đổi phân chia cấu trúc và cải tiến phép biến đổi này để có thể áp dụng cho biểu đồ lớp. Ngoài ra, chúng tôi cũng đưa ra phép biến đổi thu gọn kiểu dữ liệu để tối ưu. Đồng thời, chúng tôi cũng phát triển khung làm việc DSL, T4 và xây dựng chương trình để tiến hành tối ưu tự động.

Về vấn đề tối ưu đa mục tiêu, chúng tôi đã phát triển phương pháp tối ưu đa mục tiêu dựa trên biểu đồ lớp và nguyên lý Pareto. Dựa trên phân tích quá trình thực thi của một chương trình hướng đối tượng, chúng tôi xây dựng hàm mục tiêu hiệu năng, hàm mục tiêu bộ nhớ và hàm mục tiêu toàn cục. Chúng tôi cũng xây dựng chương trình tối ưu đa mục tiêu dựa trên biểu đồ lớp để thử nghiệm và đánh giá. Phương pháp đề xuất này đã được xuất bản trong công trình [CT11]. Ngoài ra, nghiên cứu về tối ưu đa mục tiêu, chúng tôi cũng đề xuất một số phương pháp tối ưu trong thiết kế hệ thống nhúng và đồng thiết kế phần cứng – phần mềm. Các nghiên cứu liên quan này đã được xuất bản trong các công trình [CT1, CT4, CT5, CT8].

Nhìn chung, các phương pháp tối ưu đã đề xuất và phát triển chủ yếu tập trung vào mô hình kiến trúc mức cao và thứ tự thực thi mà chưa nghiên cứu về các mô hình động như biểu đồ tuần tự, biểu đồ hoạt động, v.v. Mô hình kiến trúc chỉ bao gồm các thành phần tĩnh như các thuộc tính, các khai báo phương thức mà không đặc tả được các hành vi động cũng như các thao tác trong mỗi phương thức nên không thể phân tích và đánh giá được tính thời gian thực. Các hành vi động và tính thời gian thực lại là một khía cạnh quan trọng trong hầu hết các hệ thống nhúng. Do đó, vấn đề đặc tả hình thức, mô hình hóa các hành vi động, các đặc tả thời gian và các ràng buộc về phần cứng hệ thống nhúng để tối ưu phần mềm nhúng đảm bảo ràng buộc thời gian thực là một hướng nghiên cứu mở, quan trọng dựa trên cơ sở những nghiên cứu trong luận án.

Bên cạnh đó, các phương pháp tối ưu đề xuất trong chương này chỉ tập trung vào hai tiêu chí là hiệu năng và bộ nhớ. Do đó, các phương pháp này có thể được nghiên cứu sâu hơn dựa trên kết hợp với các độ đo chất lượng khác và mở rộng với các mô hình phức tạp. Đồng thời, các nghiên cứu trong chương này cũng được sử dụng làm nền tảng, có thể được cải tiến, mở rộng góp phần giải quyết bài toán tối ưu tổng thể bao gồm cả hướng tiếp cận tái kỹ nghệ. Hơn nữa, trong giai đoạn thiết kế chưa có mã nguồn nên các phương pháp tối ưu thường tập trung vào kiến trúc và luồng thực thi các tác vụ. Do đó, mã nguồn chương trình được cài đặt theo mô hình tối ưu cũng cần phải tiếp tục cải tiến trong các giai đoạn sau. Xuất phát từ yêu cầu này cũng như để đảm bảo tính hệ thống trong vòng đời phần mềm, chúng tôi tiếp tục nghiên cứu và thực nghiệm vấn đề tối ưu phần mềm nhúng trong giai đoạn lập trình. Các phương pháp tối ưu trong giai đoạn lập trình được trình bày chi tiết trong chương tiếp theo.

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

Các phương pháp tối ưu trong giai đoạn thiết kế tuy đem lại nhiều lợi ích nhưng chỉ thực hiện được ở mức kiến trúc vì chưa có mã nguồn. Kết quả tối ưu trong giai đoạn thiết kế là các mô hình, kiến trúc phần mềm tốt tùy theo các mục tiêu tối ưu. Kiến trúc tốt là tiêu chí quan trọng nhưng không đủ để quyết định chất lượng phần mềm. Do đó, trong giai đoạn lập trình, mã nguồn phần mềm được lập trình theo các mô hình tối ưu cũng cần được cải tiến để đáp ứng các mục tiêu tối ưu.

Tối ưu phần mềm trong giai đoạn lập trình được nghiên cứu và triển khai phổ biến nhất so với các giai đoạn khác trong vòng đời phần mềm. Các phương pháp tối ưu phần mềm trong giai đoạn lập trình tập trung chủ yếu vào tối ưu mã nguồn. Nhiều phương pháp, kỹ thuật tối ưu mã nguồn đã được tích hợp sẵn trong các chương trình dịch. Dựa trên các phương pháp tối ưu mã nguồn phần mềm thông thường, kết hợp với các ràng buộc về môi trường và thiết bị, các phương pháp tối ưu mã nguồn phần mềm nhúng cũng đã được nghiên cứu và triển khai. Tuy nhiên, khi phát triển phần mềm nhúng cần phải thực hiện quá trình biên dịch chéo. Theo đó, tối ưu mã nguồn phần mềm nhúng gồm hai mức tối ưu đó là tối ưu mã nguồn mức cao độc lập kiến trúc CPU và tối ưu mã hợp ngữ hướng đến một kiến trúc CPU cụ thể.

Vấn đề tối ưu mã nguồn mức cao đã được nghiên cứu từ lâu nhưng vấn đề tối ưu mã hợp ngữ hướng đến kiến trúc CPU đích còn ít phổ biến. Trong giai đoạn lập trình, trên cơ sở phân tích hiện trạng nghiên cứu và các thách thức đặt ra như trong *Mục 1.2.2 Chương 1*, chúng tôi đã đề xuất một số cải tiến và phương pháp tối ưu áp dụng cho phần mềm nhúng. Để đảm bảo tính hệ thống và đầy đủ hơn, chúng tôi nghiên cứu theo cả hai mức tối ưu là tối ưu mã nguồn mức cao và tối ưu mã hợp ngữ. Trong mỗi mức tối ưu, chúng tôi tóm lược cơ sở lý thuyết sau đó cải tiến hoặc đề xuất, triển khai phương pháp mới. Nội dung chương này được trình bày theo cấu trúc sau: *Mục 3.1* trình bày về quá trình biên dịch chéo và các mức tối ưu; *Mục 3.2* trình bày về tối ưu mã nguồn mức cao độc lập kiến trúc đích; *Mục 3.3* trình bày về vấn đề tối ưu chương trình hợp ngữ hướng đến các CPU đích; *Mục 3.4* tổng kết các nội dung đã trình bày.

3.1. Quá trình biên dịch chéo và các mức tối ưu

Khi phát triển phần mềm thông thường, các công cụ phát triển như trình soạn thảo, trình biên dịch, công cụ gỡ lỗi và mã đích được thực hiện trên cùng một kiến trúc CPU đích. Do đó việc phát triển này đơn giản hơn nhiều so với phát triển phần mềm nhúng. Trong phát triển phần mềm nhúng, do hạn chế về tốc độ CPU, bộ nhớ và các thiết bị ngoại vi nên các công cụ phát triển được thực hiện trên CPU mạnh của máy chủ và tạo ra mã đích chạy trên CPU khác. CPU máy chủ và CPU máy đích khác nhau nên cần tạo môi trường phát triển chéo và trình biên dịch chéo để phát triển phần mềm nhúng. Quá trình biên dịch chéo được tổng hợp trong Hình 3.1. Đầu tiên, mã nguồn viết bằng ngôn ngữ mức cao sẽ được đưa qua bộ tiền xử lý để loại bỏ chú thích, thay thế các macro cũng như thực hiện các tối ưu bước đầu để tạo ra mã nguồn trung gian mức cao. Mã nguồn trung gian mức cao sẽ được đưa qua trình biên dịch chéo để chuyển thành mã hợp ngữ. Trình biên dịch chéo cũng có thể thực hiện các kỹ thuật tối ưu trên mã nguồn trung gian mức cao trước khi dịch [48, 73, 76]. Các kỹ thuật tối ưu trong bước này tập trung vào biến đổi mã nguồn độc lập nền tảng, không hướng đến một CPU cụ thể nào. Mã hợp ngữ sẽ được đưa vào trình hợp dịch chéo để chuyển sang mã máy. Trình hợp dịch chéo có thể tiến hành các kỹ thuật tối ưu trên mã hợp ngữ của một CPU cụ thể trước khi chuyển sang mã máy tái định vị. Mã máy tái định vị có thể được trình tải/liên kết chuyển sang mã máy tuyệt đối và tiến hành nạp vào ROM. Các kỹ thuật tối ưu cũng có thể được thực hiện trên mã máy. Tối ưu mức mã máy có thể thực hiện trên chương trình thực thi hoặc tối ưu tập lệnh máy trong CPU.

3.2. Tối ưu mã nguồn mức cao độc lập máy đích

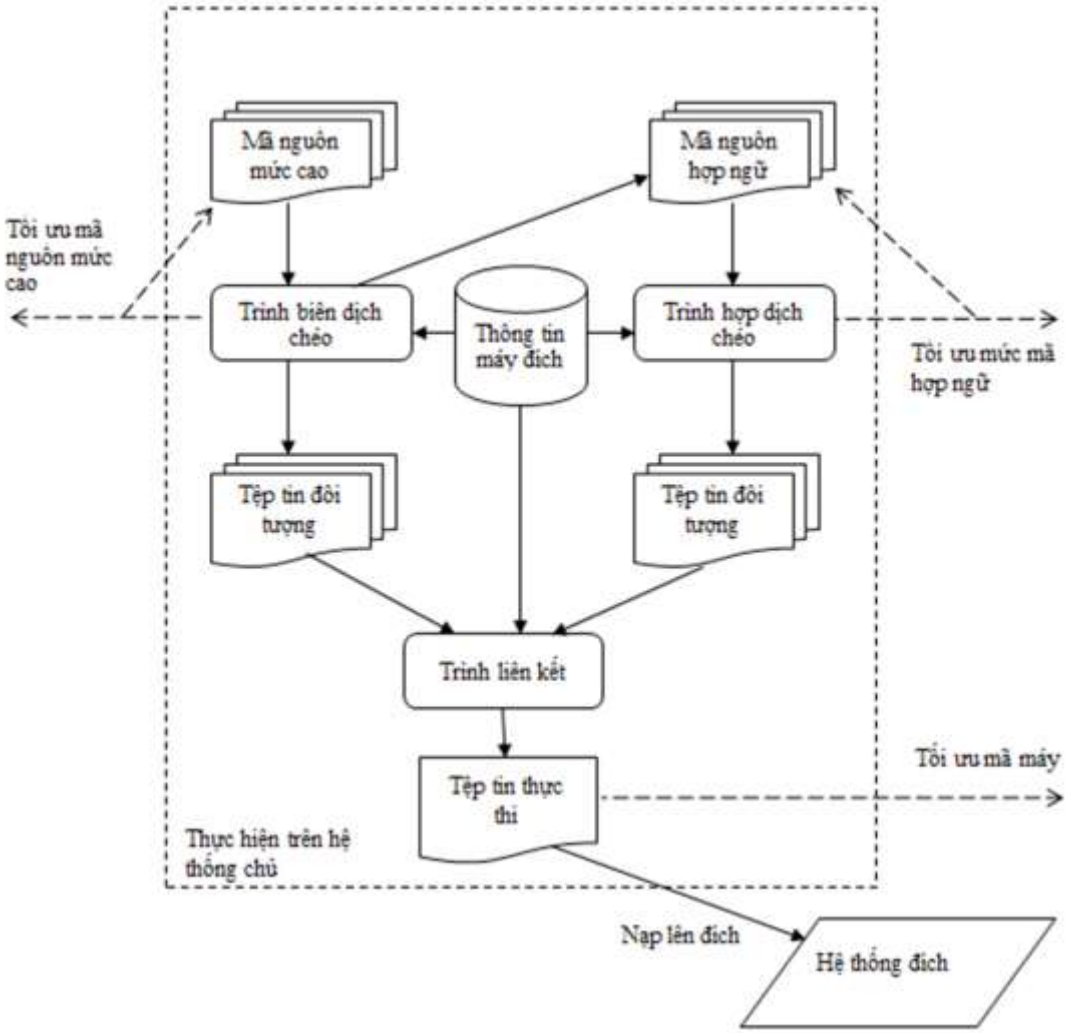
3.2.1. Cơ sở lý thuyết về tối ưu mã nguồn mức cao

i. Các biến đổi cơ bản

- **Loại bỏ dư thừa**

Phép biến đổi này nhằm xác định hai hay nhiều tính toán tương đương để loại bỏ các tính toán dư thừa và chỉ giữ lại một. Các kiểu loại bỏ dư thừa bao gồm:

- Các giá trị tính được: Kết hợp giá trị của các ký hiệu để tính và xác định xem hai hay nhiều biểu thức có cùng giá trị.
- Loại bỏ các biểu thức con chung: Xác định các biểu thức có các toán hạng giống nhau.
- Lan truyền hằng và bản sao giá trị: Xác định các biến có giá trị hằng hoặc bản sao để thay thế giá trị này cho biến.



Hình 3.1: Quá trình biên dịch chéo và các mức tối ưu

- **Thực hiện các tính toán trong thời gian biên dịch**

Biến đổi này nhằm kiểm tra các biểu thức tính toán được trong thời gian biên dịch để thay thế nó. Việc thay thế này sẽ cải tiến được hiệu năng và năng lượng tiêu thụ cũng như kích thước mã nguồn. Có hai cách biến đổi là gộp các hằng số và lan truyền các hằng số.

Gộp các hằng số: tính toán giá trị biểu thức với các toán hạng là hằng số và thay kết quả này cho các hằng số. Ví dụ xét biểu thức sau:

```
chuVi = (333.0/106.0) * duongKinh
```

Trong biểu thức này, `333,0/106,0` là một biểu thức hằng nên được tính toán và gộp thành biểu thức sau:

```
chuVi = 3.14159 * duongKinh
```

Lan truyền các giá trị hằng: trong biến đổi này, mỗi biến được thay thế bằng một giá trị hằng đã gán cho nó trước đó và trước khi có sự thay đổi về giá trị của biến. Ví dụ xét đoạn mã sau:

```
PI = 3.14159
chuVi = PI * duongKinh
```

Trong đoạn mã này PI là biến đã được gán giá trị hằng và chưa có sự thay đổi nào về giá trị nên PI được thay thế bằng giá trị đã gán trước đó. Kết quả như sau:

```
chuVi = 3.14159 * duongKinh
```

- **Thay thế các biểu thức con chung**

Khi các biểu thức con được chứa trong các biểu thức con khác cần tính toán giá trị biểu thức con một lần và thay thế kết quả này vào mọi biểu thức chứa biểu thức con. Biến đổi này cải tiến được cả về hiệu năng, kích thước và tiêu thụ điện năng. Xét đoạn mã sau:

```
X = (a + b * c)  * 10
Y = a + b * c – 100 + d
Z =  a + b * c + e*f
T = a + b * c – d*e/f
```

Trong đoạn mã này, `a + b * c` là biểu thức con chung do đó chúng ta có thể tính toán và thay thế như sau:

```
temp = a + b * c
X = temp * 10
Y = temp – 100 + d
Z = temp + e*f
T = temp – d*e/f
```

- **Giảm các lệnh phức tạp**

Phép biến đổi này nhằm thay thế các lệnh phức tạp, tiêu tốn năng lượng như lệnh nhân, chia, lũy thừa bằng các lệnh đơn giản và hiệu quả hơn như lệnh cộng, lệnh dịch, v.v. Bảng 3.1 minh họa lệnh nhân được thay bằng lệnh cộng.

Bảng 3.1. Thay thế lệnh phức tạp

Mã ban đầu	**Mã thay thế**

`for i = 1 to 10 do`
`  …`
`  X = i * 5`
`  …`
`end`
	`X = 0`
	`for i = 1 to 10 do`
	`  …`
	`  X += 5`
	`  …`
	`end`

- **Di chuyển mã**

Phép biến đổi này di chuyển một lệnh hoặc một đoạn mã đến các vị trí khác nhau nhằm đạt được các mục tiêu tối ưu. Bảng 3.2 minh họa kỹ thuật di chuyển mã để giảm kích thước, giảm tần số thực hiện và di chuyển mã ra ngoài vòng lặp.

Bảng 3.2. Di chuyển mã để tối ưu

Đoạn mã ban đầu	**Đoạn mã cải tiến**
Di chuyển mã để giảm kích thước	

`if (a < b) then`
`  Z = X * c * 2`
`else`
`  Y = X * c * 2 + 10`
	`temp = X * c * 2`
	`if (a < b) then`
	`  Z = temp`
	`else`
	`  Y = temp + 10`

Di chuyển mã để tăng hiệu năng	

`if (a < b) then`
`  Z = X * 2`
`else`
`  Y = 10`
`G = X * 2`
	`if (a < b) then{`
	`  temp = X * 2`
	`  Z = temp`
	`}`
	`else{`
	`  Y = 10`
	`  temp = X * 2`
	`}`
	`G = temp`

Di chuyển mã ra ngoài vòng lặp	
`while (i < (max - 2))`	`temp = max – 2` `while (i < temp)`

- **Loại bỏ mã chết**

Mã chết là những đoạn mã không được thực thi theo bất kỳ một đường nào trong chương trình. Loại bỏ mã chết sẽ cải tiến chương trình về mọi tiêu chí. Ví dụ một đoạn mã chết như minh họa sau.

```
kiemTra = 0
if (kiemTra) {…}
```

- **Lan truyền bản sao đơn giản hơn**

Là phép biến đổi thay thế đối tượng được gán bằng đối tượng gán. Điều này chỉ có ý nghĩa khi truy xuất đến đối tượng được gán phức tạp hơn đối tượng gán. Biến đổi này được minh họa trong hai đoạn mã dưới đây.

```
x[i] = a
sum = x[i] + a
```

Thành

```
x[i] = a
sum = a + a
```

ii. Phân tích mã nguồn

Trước khi cải tiến mã nguồn, cần thực hiện phân tích mã nguồn chương trình. Để hỗ trợ các kỹ thuật tối ưu, phân tích mã nguồn nhằm phân chia chương trình thành các khối nhỏ hơn và các luồng điều khiển. Trong hầu hết các trình biên dịch, chương trình được chia thành các khối cơ bản và xây dựng đồ thị luồng điều khiển giữa các khối cơ bản để tiến hành tối ưu theo các mức khác nhau. Tối ưu cục bộ được thực hiện trong mỗi khối cơ bản. Tối ưu vòng lặp và tối ưu toàn cục được tập trung vào quan hệ giữa các khối cơ bản.

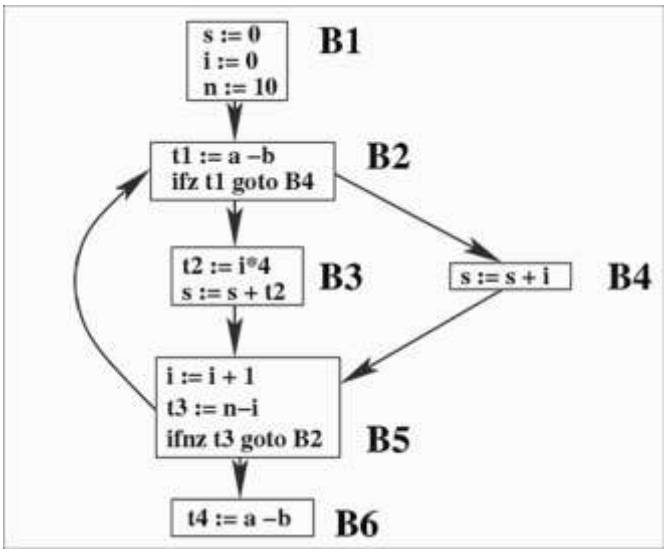
- **Khối cơ bản**

Khối cơ bản là một chuỗi tuần tự các câu lệnh với một điểm vào duy nhất là lệnh đầu tiên và một điểm ra duy nhất là lệnh cuối cùng của khối lệnh. Theo đó sẽ không có các cấu trúc rẽ nhánh và cấu trúc lặp trong khối cơ bản. Một khối cơ bản có thể minh họa như sau:

```
L3: t5 := a * b
    t6 := t5 + c
    d := t2 * t2
    if d = 0 goto L4
```

- **Biểu đồ luồng điều khiển trong chương trình**

Sau khi đã xác định được các khối cơ bản, cần xây dựng các biểu đồ luồng điều khiển (CFG) để phân tích quá trình thực thi trong chương trình nhằm hỗ trợ tối ưu toàn cục cũng như tối ưu các cấu trúc lặp. Biểu đồ luồng điều khiển là đồ thị có hướng biểu diễn cách tổ chức các khối cơ bản trong chương trình. Mỗi nút trong đồ thị là một khối cơ bản. Mỗi cạnh biểu diễn một luồng thực thi từ một khối cơ bản đến một khối khác. Hình 3.2 minh họa các khối cơ bản và đồ thị luồng điều khiển. Trong phân tích mã nguồn, biểu đồ luồng dữ liệu thường sử dụng để xác định vòng lặp và hỗ trợ tối ưu toàn cục như lan truyền bản sao, thay thế biểu thức chung dựa trên các biểu đồ con dẫn xuất từ biểu đồ luồng điều khiển.



Hình 3.2: Đồ thị luồng điều khiển

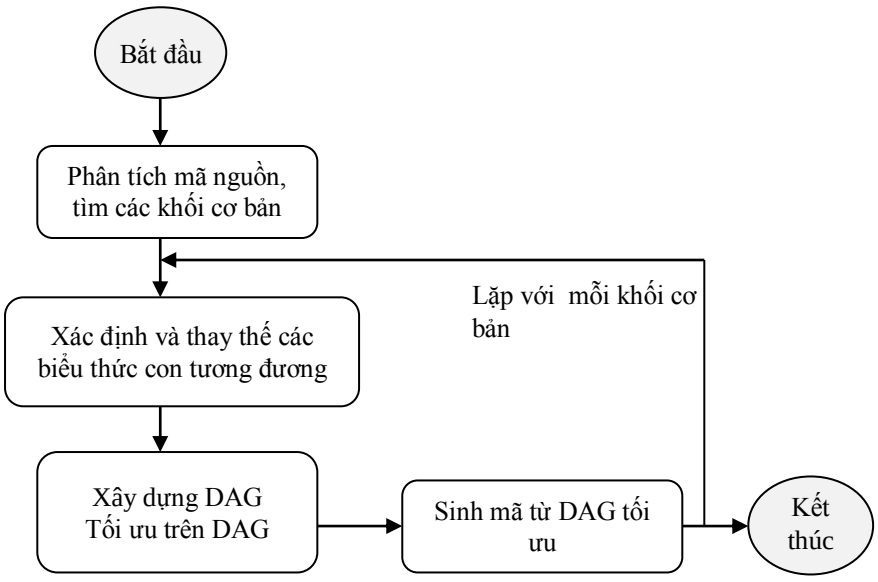
3.2.2. Cải tiến tối ưu cục bộ dựa trên thay thế biểu thức tương đương

i. Ý tưởng và quy trình nghiên cứu

Trong phần này, chúng tôi đề xuất một cải tiến cho các phương pháp tối ưu cục bộ dựa trên thay thế các biểu thức tương đương. 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.

Trong mỗi khối cơ bản, vấn đề loại bỏ các biểu thức con chung để tối ưu được dùng phổ biến nhất. Phương pháp loại bỏ biểu thức con chung phổ biến nhất là dựa trên đồ thị có hướng không chu trình (DAG). Theo phương pháp này, đầu tiên cần xây dựng DAG của mỗi khối cơ bản sau đó thực hiện các biến đổi tối ưu trên DAG và sinh mã từ DAG tối ưu. Tuy nhiên, phương pháp tối ưu truyền thống này chưa xét đến sự thay thế của các biểu thức tương đương [3]. Nghĩa là chỉ thay thế các biểu thức giống nhau mà không thay thế các biểu thức khác nhau nhưng tương đương nhau dựa trên các tính chất toán học.

Quy trình tối ưu theo phương pháp cải tiến này được chúng tôi thực hiện như trong Hình 3.3. Đầu tiên, chúng tôi phân tích mã nguồn để xác định các khối cơ bản. Sau đó, chúng tôi phân tích mỗi khối cơ bản để tìm các biểu thức tương đương, các biểu thức con tương đương và các biểu thức tính toán được. Các biểu thức con tương đương là một phần trong một biểu thức tương đương với một phần trong một biểu thức khác. Các biểu thức, biểu thức con tương đương sẽ được thay thế bằng biểu thức ban đầu. Các biểu thức có khả năng tính toán được cũng được tính và thay thế bằng giá trị của biểu thức. Tiếp đến, chúng tôi xây dựng DAG của mỗi khối cơ bản và thực hiện các biến đổi trên DAG để tối ưu. Từ DAG tối ưu sẽ sinh ngược lại mã nguồn của mỗi khối cơ bản.



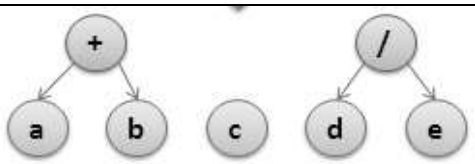
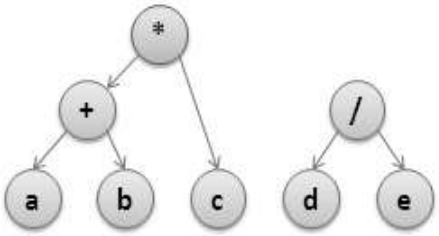
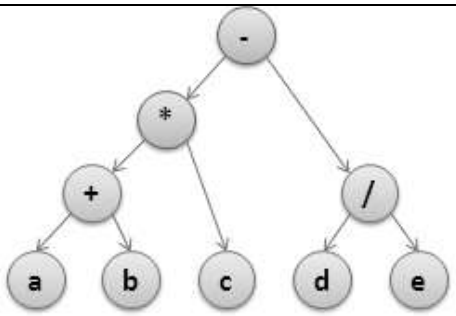
Hình 3.3: Quy trình tối ưu dựa trên biểu thức tương đương

ii. Xác định và thay thế biểu thức tương đương

- **Xác định biểu thức tương đương**

Điểm quan trọng trong cải tiến của chúng tôi là cần phải xác định các biểu thức tương đương dựa trên các tính chất toán học. Hai biểu thức có thể được chứng minh tương đương dựa trên cây biểu thức. Bảng 3.3 minh họa các bước phân tích, xây dựng cây biểu thức. Trong bước 1, biểu thức ban đầu được bổ sung thêm các dấu ngoặc đơn để phân biệt các phép toán theo mức ưu tiên. Dựa trên các toán hạng đơn là các hằng, biến đã phân tích trong bước 2, các cây con cho các phép toán ưu tiên cao nhất sẽ được xây dựng như trong bước 3. Quá trình sẽ tiếp tục cho các phép toán mức ưu tiên tiếp theo như trong bước 4 và bước 5. Trong mỗi bước, có thể chỉ ra hai biểu thức tương đương dựa trên tính chất giao hoán. Hai biểu thức tương đương trong mọi bước thì mới tương đương nhau.

Bảng 3.3. Minh họa các bước chứng minh hai biểu thức tương đương

Các bước	Biểu thức 1	Cây biểu thức	Biểu thức 2
	`(a+b)*c-d/e`		`c*(b+a)-d/e`
1	`((a+b)*c)-(d/e)`		`(c*(b+a))-(d/e)`
2			
3	`(a+b)` `(d/e)`		`(b+a)` `(d/e)`
4	`(a+b)*c`		`c*(a+b)`
5	`((a+b)*c)-(d/e)`		`(c*(b+a))-(d/e)`

- **Thay thế biểu thức tương đương**

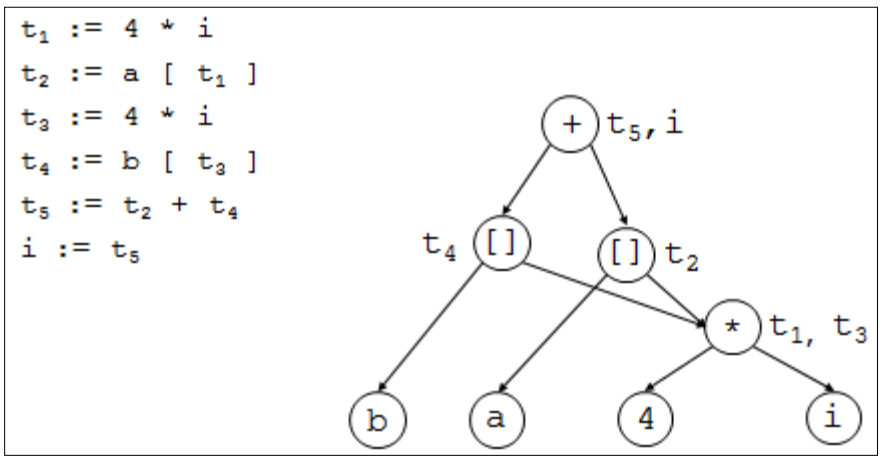
Dựa trên phương pháp xác định hai biểu thức tương đương như trình bày trong phần trước, trong phần này, chúng tôi sẽ thay thế các biểu thức tương đương nhau bằng một biểu thức đại diện trong các khối cơ bản. Việc thay thế này được thực hiện trước các kỹ thuật tối ưu nhằm cải tiến chất lượng tối ưu khi loại bỏ các biểu thức con chung. Đầu tiên, duyệt từng câu lệnh trong khối cơ bản để tìm các biểu thức và biểu thức con. Sau đó xác định các biểu thức tương đương trong tập biểu thức. Cuối cùng, thay thế các biểu thức tương đương bằng biểu thức đại diện sẽ được khối mã bao gồm nhiều biểu thức hoặc biểu thức con giống nhau. Các biểu thức giống nhau này sẽ được loại bỏ khi thực hiện kỹ thuật tối ưu sẽ trình bày trong phần tiếp theo.

iii. Xây dựng DAG của khối cơ bản

Các phương pháp tối ưu mã nguồn cục bộ trong các khối cơ bản thường dựa trên đồ thị có hướng không chu trình biểu diễn khối mã. Dựa trên DAG để thực hiện các kỹ thuật tối ưu như loại bỏ biểu thức con chung, loại bỏ mã chết, v.v. DAG của một khối cơ bản được mô tả như sau:

- Các nốt lá được gán nhãn với định danh duy nhất (tên biến hoặc hằng)
- Các nốt trong được gán nhãn bằng ký hiệu phép toán
- Mỗi nốt có thể có một danh sách các nhãn
- Các cạnh thể hiện quan hệ giữa phép toán và toán hạng
- Nốt trong biểu diễn giá trị tính được thì định danh trên nhãn lưu giữ giá trị

Để xây dựng DAG của khối cơ bản, có thể sử dụng thuật toán trong nghiên cứu [73]. Hình 3.4 minh họa một khối cơ bản và DAG tương ứng. Trong đó mỗi nốt lá biểu diễn một giá trị hoặc một định danh, mỗi nốt trong biểu diễn một lệnh và các nốt con của nốt trong biểu diễn định nghĩa mới nhất của các toán hạng.



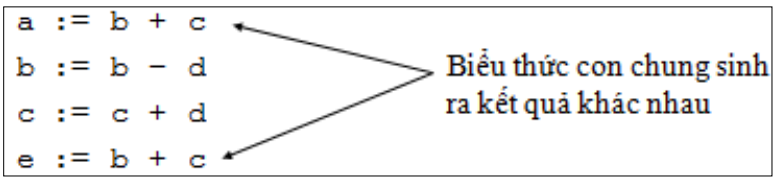
Hình 3.4: Khối cơ bản và DAG tương ứng

iv. Tối ưu cục bộ dựa trên DAG

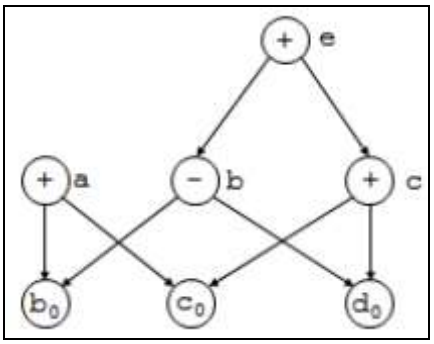
Sau khi đã thay thế các biểu thức tương đương bằng một biểu thức đại diện, chúng tôi sử dụng các phép biến đổi trên DAG như loại bỏ các biểu thức con chung và loại bỏ mã chết để tối ưu cục bộ. Ý tưởng của loại bỏ các biểu thức con chung là tính giá trị của biểu thức con một lần chứa vào biến và thay thế biến này cho mọi biểu thức con chung còn lại. Theo đó chương trình không phải tính toán lại biểu thức nên rút được thời gian thực thi. Loại bỏ mã chết là loại bỏ các đoạn mã không được thực thi bởi bất kỳ một đường nào trong chương trình. Loại bỏ các đoạn mã này giúp cải tiến về mọi tiêu chí tối ưu.

- **Loại bỏ các biểu thức con chung**

Các biểu thức con chung được loại bỏ bằng việc xây dựng DAG. Tuy nhiên, các biểu thức con chung trong các vị trí khác nhau phải có cùng kết quả tính toán mới được loại bỏ. Hình 3.5a sau minh họa biểu thức con chung có giá trị khác nhau nên không thể loại bỏ và phải được biểu diễn lại và xây dựng DAG như trong Hình 3.5b.



a. Biểu thức con chung không được loại bỏ

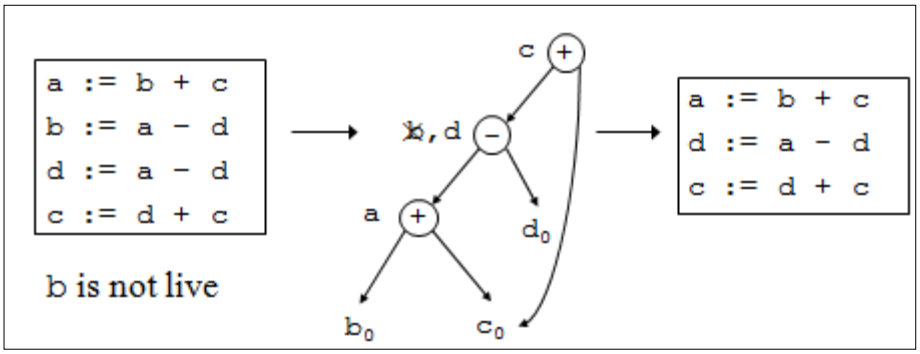


b. DAG được xây dựng lại

Hình 3.5: Minh họa xử lý loại bỏ biểu thức con chung

- **Loại bỏ đoạn mã chết**

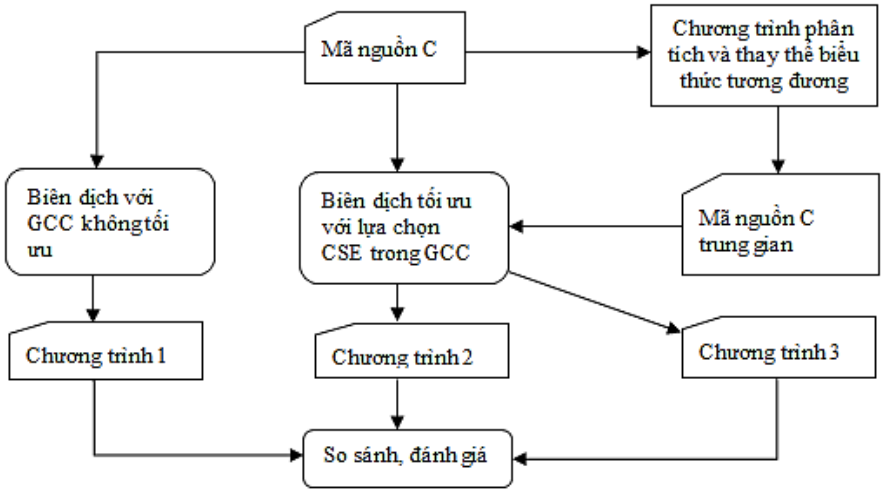
Mã sinh từ DAG sẽ không chứa các đoạn mã chết như minh họa trong Hình 3.6.



Hình 3.6: Mã sinh ra từ DAG không chứa mã chết

v. Thực nghiệm và đánh giá

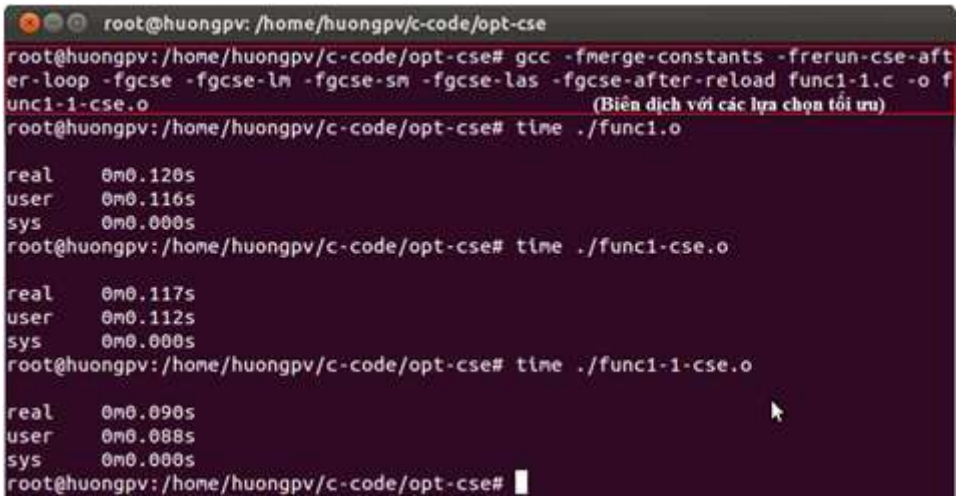
Trong phần này, để kiểm chứng phương pháp tối ưu, chúng tôi tiến hành thực nghiệm theo mô hình như trong Hình 3.7. Đầu vào của chương trình thay thế biểu thức tương đương là một chương trình nguồn C. Đầu ra là một chương trình nguồn C với các biểu thức tương đương nhau đã được thay thế. Từ chương trình C ban đầu được biên dịch với GCC không lựa chọn tối ưu được chương trình 1 và được biên dịch với các lựa chọn tối ưu nhằm loại bỏ biểu thức con chung (CSE) được chương trình 2. Chương trình C đã được thay thế với các biểu thức con tương đương cũng được biên dịch bằng GCC với các lựa chọn tối ưu như vậy được chương trình 3. Quá trình biên dịch với các lựa chọn tối ưu trong GCC được minh họa trong Hình 3.8. Sau đó, chúng tôi sử dụng lệnh `time` của hệ điều hành Ubuntu đo thời gian thực thi của mỗi chương trình 10 lần và lấy giá trị trung bình. Khi sử dụng lệnh `time` có ba tham số ra: *real* là tổng thời gian thực hiện tiến trình từ khi bắt đầu đến khi kết thúc, *user* là thời gian sử dụng CPU thực tế của tiến trình trong chế độ người dùng và *sys* là thời gian sử dụng CPU thực tế của tiến trình mức nhân hệ điều hành. Do đó, để tránh ảnh hưởng của các tiến trình khác, trong thực nghiệm chúng tôi lấy thời gian thực thi theo tham số *user*. Môi trường thực nghiệm cụ thể được chỉ ra trong Bảng 3.4. Kết quả thực nghiệm đánh giá hiệu năng của ba chương trình được tổng hợp trong Bảng 3.5 và Hình 3.9.



Hình 3.7: Mô hình thực nghiệm thay thế biểu thức tương đương

Bảng 3.4. Môi trường thực nghiệm

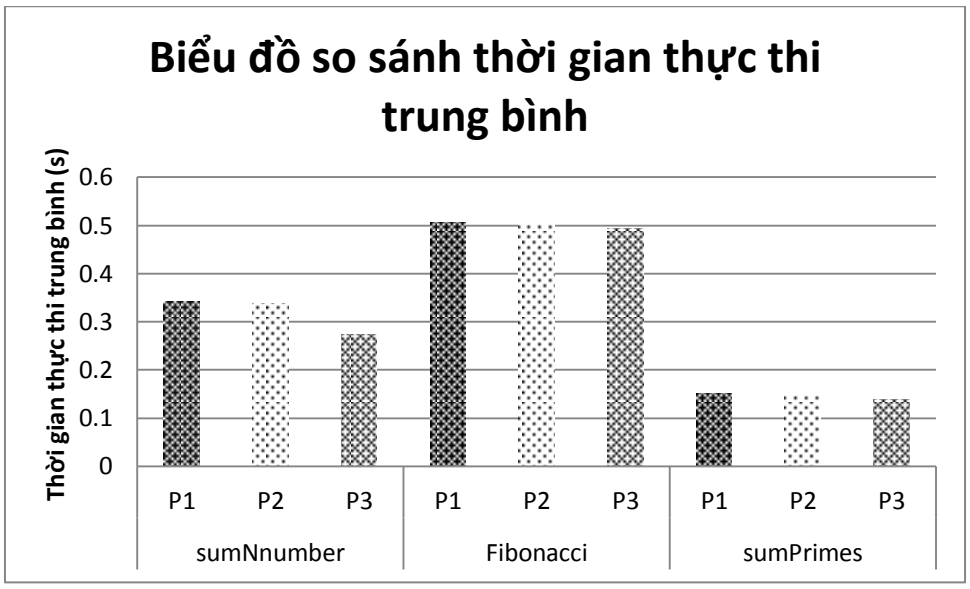
Vi xử lý	Intel Core i5 – 2.26
Bộ nhớ	2 GB
Hệ điều hành	Ubuntu 11.0.4
Chương trình dịch	GCC 4.6.4
Công cụ đo thời gian thực thi	Lệnh `time` của Ubuntu



Hình 3.8: Biên dịch và lựa chọn tối ưu với GCC

Bảng 3.5. Thời gian thực thi và kích thước chương trình

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	0,35	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**



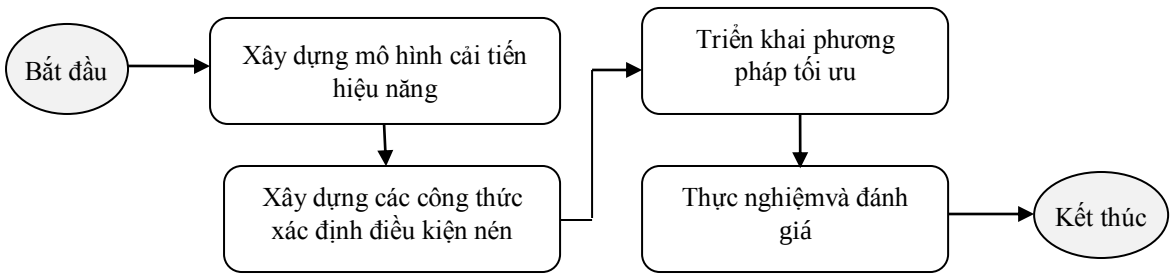
Hình 3.9: So sánh thời gian thực thi của các phiên bản chương trình

Trong thực nghiệm trên, chúng tôi đã sử dụng bộ công cụ biên dịch GCC phiên bản 4.6.4 [84] để biên dịch với các lựa chọn tối ưu và đánh giá thời gian thực thi. Bộ công cụ biên dịch GCC 4.6.4 hỗ trợ các lựa chọn tối ưu thay thế các biểu thức con giống nhau nhưng không hỗ trợ tính năng thay thế các biểu thức tương đương. Biểu đồ so sánh hiệu năng trong Hình 3.9 chỉ ra chương trình 1 được biên dịch không lựa chọn tối ưu có thời gian thực thi trung bình lớn nhất, chương trình 2 được biên dịch với các lựa chọn tối ưu loại bỏ biểu thức con chung có thời gian thực thi trung bình ngắn hơn và chương trình 3 được thay thế các biểu thức tương đương trước khi được biên dịch với các lựa chọn tối ưu loại bỏ các biểu thức chung có thời gian thực thi trung bình nhỏ nhất. Kết quả thực nghiệm cho thấy việc phân tích và thay thế các biểu thức tương đương trong phương pháp cải tiến của chúng tôi giúp rút ngắn thời gian thực thi. Cải tiến này có nhiều ý nghĩa hơn trong phát triển phần mềm nhúng với môi trường thực thi hạn chế.

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

i. Ý tưởng và quy trình nghiên cứu

Trong phần này, chúng tôi đề xuất và triển khai phương pháp cải tiến hiệu năng phần mềm nhúng trên điện thoại di động trong các môi trường phân tán 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. Quy trình nghiên cứu và thực nghiệm phương pháp tối ưu này được chỉ ra như trong Hình 3.10.



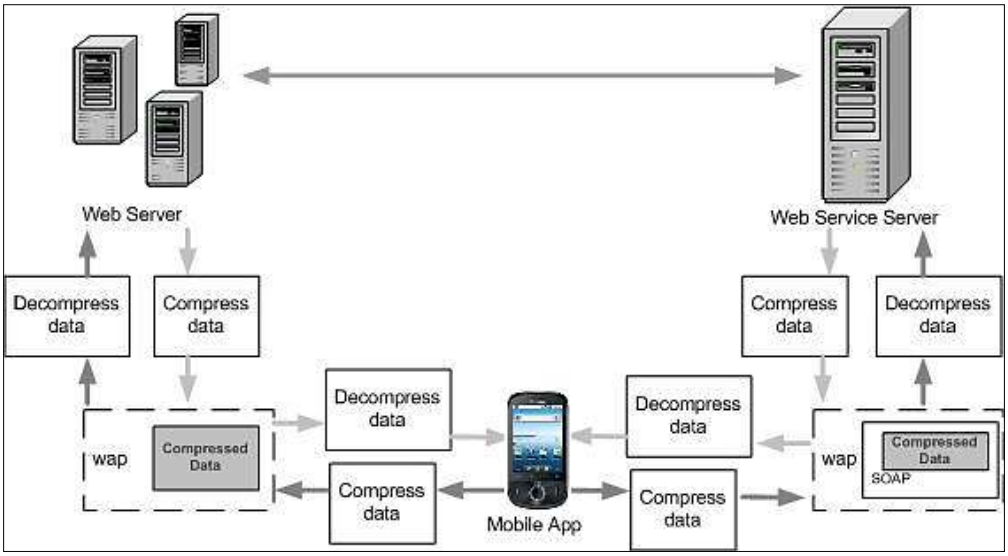
Hình 3.10: Quy trình nghiên cứu, triển khai tối ưu dựa trên nén dữ liệu

ii. Mô hình cải tiến hiệu năng

Phần mềm điện thoại di động trong môi trường phân tán thường thực hiện theo mô hình dịch vụ web và sử dụng giao thức truy xuất đối tượng đơn giản (SOAP). Theo mô hình dịch vụ web, phần mềm trên điện thoại di động đóng gói dữ liệu trong một gói SOAP, gửi lên máy chủ dịch vụ web, triệu gọi thực hiện một dịch vụ từ xa và hiển thị kết quả trả về từ dịch vụ web. Do đó thời gian thực thi ứng dụng điện thoại di động trong môi trường phân tán có thể được tính theo công thức (3.1):

$$T = T_1 + T_2 + T_3 \quad (3.1)$$

Trong đó T là tổng thời gian thực thi, T_1 là thời gian xử lý trên điện thoại di động, T_3 là thời gian dữ liệu đi trên đường truyền và T_2 là thời gian xử lý trên máy chủ. Do đó vấn đề cải tiến, tối ưu phần mềm phân tán trên điện thoại di động gồm ba phần chính: tối ưu cục bộ trên điện thoại di động, tối ưu cục bộ máy chủ dịch vụ web và tối ưu tiêu thụ băng thông để tăng tốc độ truyền dữ liệu. Mô hình cải tiến hiệu năng phần mềm nhúng phân tán dựa vào nén dữ liệu được mô tả như trong Hình 3.11.



Hình 3.11: Mô hình cải tiến hiệu năng dựa vào nén dữ liệu

iii. Xác định điều kiện nén

Khi dữ liệu được nén, chương trình cần thời gian để nén trước gửi đi và thời gian giải nén sau khi nhận được để có thể xử lý. Do đó T_1 và T_2 tăng lên còn T_3 giảm xuống. Gọi thời gian tăng lên là T_i và T_i được tính theo công thức (3.2). Thời gian giảm xuống T_d được tính theo công thức (3.3). Trong đó, S_F là kích thước tệp tin dữ liệu, thời gian trung bình để nén một byte dữ liệu là t_u, thời gian trung bình để giải nén một byte là t_d, thời gian trung bình để truyền một byte dữ liệu là t_c và r là tỉ lệ nén. Từ công thức (3.2) và (3.3), chúng tôi tính tổng thời gian thực thi giảm đi T_t theo công thức (3.4).

$$T_i = S_F \times t_u + S_F \times r \times t_d \tag{3.2}$$

$$T_d = S_F \times (1 - r) \times t_c \tag{3.3}$$

$$T_t = T_d - T_i \tag{3.4}$$

Theo các công thức trên, khi sử dụng kỹ thuật nén dữ liệu để giảm tổng thời gian thực thi, cần xác định được điều kiện nén. Chỉ khi thời gian tiết kiệm được trên đường truyền lớn hơn tổng thời gian nén và giải nén trên điện thoại di động và trên máy chủ thì

dữ liệu mới được nén và gửi đi. Do đó điều kiện nén được xác định theo công thức (3.2), (3.3) và (3.4) khi T_t bằng 0.

iv. Thực nghiệm và đánh giá

Để kiểm chứng mô hình cải tiến hiệu năng dựa trên nén dữ liệu, trong thực nghiệm này, chúng tôi thử nghiệm với chương trình nhận dạng chữ Nôm trên điện thoại di động theo trong môi trường phân tán. Chương trình này được mô tả chi tiết trong *Phụ lục P.2.2*. Chương trình thử nghiệm này được xây dựng theo hai phiên bản có nén và không nén dữ liệu. Sau đó chúng tôi biên dịch và thực thi hai phiên bản trên cùng một môi trường như trong Bảng 3.6. Mỗi phiên bản được thực thi 10 lần để tính thời gian thực thi trung bình. Bảng 3.7 tổng hợp thời gian thực thi phiên bản không nén dữ liệu của chương trình và Bảng 3.8 tổng hợp thời gian thực thi phiên bản nén dữ liệu. Biểu đồ so sánh hiệu năng được trình bày trong Hình 3.12.

Bảng 3.6. Môi trường phát triển và thực hiện ứng dụng

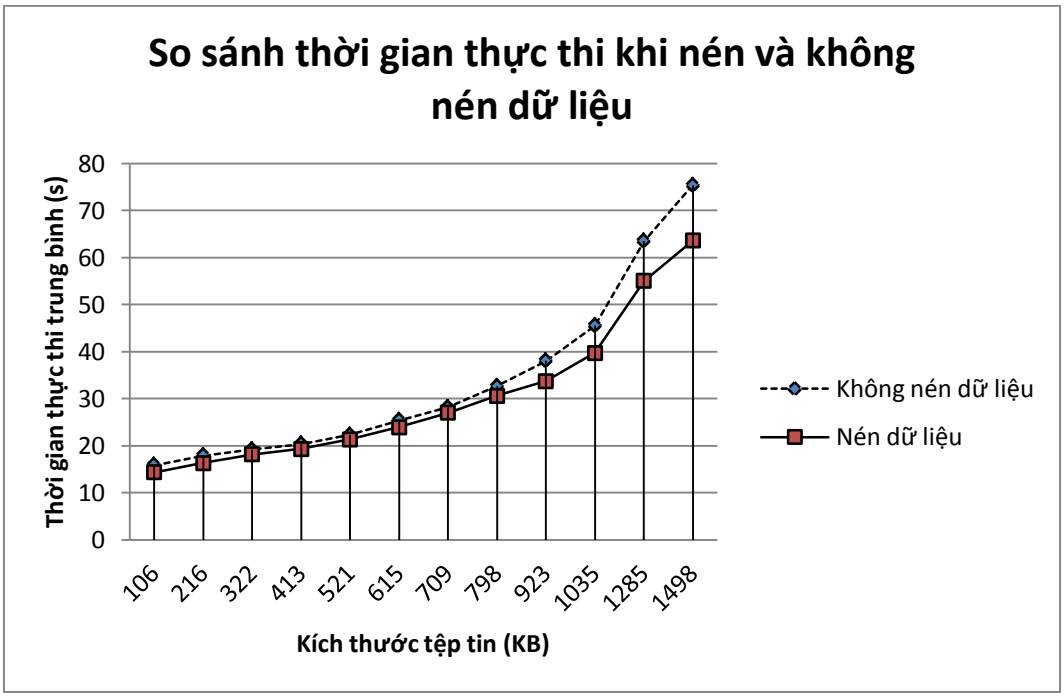
Công nghệ	**Mô-đun trên điện thoại**	**Dịch vụ web trên máy chủ**
Hệ điều hành	Windows Mobile	Windows
Thư viện	.NET Compact framework	.NET framework
Ngôn ngữ	C#	C#
Công cụ	Visual Studio.NET 2008	Visual Studio.NET 2008

Bảng 3.7. Tổng hợp thời gian thực thi phiên bản không nén

Lần thực hiện	**Kích thước tệp tin (KB)**											
	106	**216**	**322**	**413**	**521**	**615**	**709**	**798**	**923**	**1035**	**1285**	**1498**
1	16,3	18,2	19,5	20,7	22,1	25	27,5	32,7	38	46,2	65,0	76,2
2	15,0	17,9	19,7	20,4	22,7	24,5	27,8	32,2	38,2	45,5	63,5	75,7
3	14,7	16,5	18,6	19,8	21,5	24,8	29,1	33,1	37,5	45,1	63,7	76,3
4	15,1	17,5	19,3	20,5	20,8	26,3	28,0	34,0	39,0	46,1	64,2	76,1
5	16,4	18,5	20,1	20,1	21,4	25,1	27,5	33,5	38,5	46,4	62,8	75,3
6	15,7	18,1	19,4	21,2	22,3	25,7	28,4	32,3	37,6	47,0	63,1	74,8
7	16,3	16,9	19,5	19,6	22,7	26,5	27,0	31,7	36,8	45,2	63,3	75,0
8	16,5	18,3	18,6	20,3	23,5	26,0	28,2	31,5	39,2	43,9	62,9	74,6
9	16,2	18,8	18,3	20,4	22,6	24,7	29,0	33,4	37,7	44,5	63,5	75,2
10	16,2	18,35	19,18	20,34	23,5	25,3	29,3	32,5	38,1	45,2	63,0	75,0
Thời gian trung bình (s)	15,84	17,91	19,22	20,33	22,31	25,39	28,18	32,69	38,06	45,51	63,5	75,42

Bảng 3.8. Tổng hợp thời gian thực thi phiên bản nén dữ liệu

Lần thực hiện	Kích thước tệp tin (KB)											
	106	**216**	**322**	**413**	**521**	**615**	**709**	**798**	**923**	**1035**	**1285**	**1498**
1	14,0	17,5	18,0	20,0	22,5	23,5	26,5	30,7	33,5	39,2	55,1	64,1
2	13,0	16,0	18,7	20,7	22,3	24,2	26,8	30,2	35,2	39,5	55,5	65,7
3	12,5	15,5	19,0	19,2	21,1	23,8	27,1	31,0	34,5	40,1	53,7	62,1
4	15,0	17,0	19,3	18,5	20,8	24,3	27,3	30,4	34,1	39,1	54,8	63,1
5	13,4	16,5	16,9	20,1	20,4	24,1	27,5	31,2	33,4	39,4	53,8	62,3
6	14,7	15,0	17,0	19,0	21,3	23,7	26,4	30,5	34,6	41,0	55,1	64,7
7	16,0	14,9	18,5	19,6	20,7	24,5	27,0	30,7	32,8	41,2	57,3	65,0
8	15,0	17,3	17,6	18,3	22,5	24,0	27,2	30,5	33,2	39,3	55,9	64,6
9	14,5	16,8	18,3	18,9	20,6	23,7	26,9	30,4	32,7	39,5	54,2	62,2
10	15,5	17,0	18,5	19,1	21,42	23,7	27,2	30,6	33,4	39,3	55,2	62,5
Thời gian trung bình (s)	14,36	16,35	18,18	19,34	21,36	23,95	26,99	30,62	33,74	39,76	55,06	63,63



Hình 3.12: So sánh thời gian thực thi khi nén và không nén dữ liệu

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

Tối ưu mã hợp ngữ là mức tối ưu đặc thù của hệ thống nhúng và có hai nhân tố chính ảnh hưởng đến tối ưu là kiến trúc CPU đích và kiến trúc máy đích. Khác với các hệ thống máy tính thông thường, CPU trong hệ thống nhúng đơn giản nhưng đa dạng hơn rất

nhiều. Mỗi loại CPU có số lượng thanh ghi khác nhau, tập lệnh máy khác nhau và kiểu kiến trúc khác nhau. Có hai kiểu kiến trúc tập lệnh CPU phổ biến nhất là RISC và CISC. RISC là kiến trúc tập lệnh phổ biến nhất của các CPU hệ thống nhúng với tập lệnh rút gọn, các lệnh có cùng độ dài và không có các phép toán trực tiếp giữa thanh ghi và bộ nhớ. CISC là kiến trúc tập lệnh phức tạp với số lượng lệnh nhiều hơn, độ dài các lệnh không cố định và cho phép thực hiện các phép toán giữa thanh ghi và bộ nhớ. Trình hợp dịch phải biết kiến trúc CPU đích để thực hiện các kỹ thuật tối ưu hiệu quả hơn. Ngoài ra, với mỗi kiến trúc CPU khác nhau cũng cần có các phương pháp tối ưu phù hợp. Có ba kiểu kiến trúc bên trong CPU đó là kiến trúc đơn lệnh, kiến trúc đường ống lệnh (pipeline) và kiến trúc siêu vô hướng (superscalar). So với tối ưu cho kiến trúc đơn lệnh, tối ưu cho kiến trúc đường ống lệnh, siêu vô hướng phức tạp hơn và còn nhiều thách thức. Đồng thời với mỗi kiến trúc CPU đích cũng cần quan tâm đến số các đơn vị chức năng như đơn vị lôgic và số học (ALU) và đơn vị xử lý dấu phẩy động (FPU). Nếu CPU có nhiều đơn vị chức năng sẽ cho phép nhiều lệnh thực hiện đồng thời. Do đó trình hợp dịch có thể thực hiện tối ưu dựa trên song song hóa bên trong CPU.

Nhân tố chính thứ hai tác động đến các phương pháp tối ưu là kiến trúc máy đích. Có hai kiến trúc máy đích phổ biến là Von Neumann và Harvard. Kiến trúc Harvard cải tiến hiệu năng hơn so với kiến trúc Von Neumann do sự tách biệt của bộ nhớ chương trình và bộ nhớ dữ liệu cũng như có đường truyền riêng cho mỗi loại bộ nhớ này. Trong kiến trúc máy đích cũng cần quan tâm đến phân cấp bộ nhớ và sử dụng bộ nhớ ẩn (cache). Kích thước bộ nhớ ẩn và tỉ lệ truyền dữ liệu giữa bộ nhớ chính và bộ nhớ ẩn cũng ảnh hưởng đến hiệu năng và điện năng tiêu thụ. Do đó khi tối ưu mức hợp ngữ, kiến trúc máy đích là một nhân tố quan trọng mà trình hợp dịch chéo phải xem xét.

Ngoài hai nhân tố chính ảnh hưởng đến các phương pháp tối ưu, với mỗi mục tiêu tối ưu khác nhau cần có phương pháp phù hợp và hiệu quả. Cũng như trong các giai đoạn khác, trong mức tối ưu này, mục tiêu tối ưu có thể là hiệu năng, tiêu thụ điện năng, kích thước phần mềm nhúng, v.v. và tối ưu đa mục tiêu. Các phương pháp tối ưu mức mã hợp ngữ tập trung vào ba hướng chính đó là lập lịch các lệnh, phân bổ thanh ghi và các biến đổi mức lệnh hợp ngữ. Nội dung chi tiết trong phần này sẽ tóm lược cơ sở lý thuyết cũng như cải tiến và đề xuất một số phương pháp tối ưu theo các mục tiêu tối ưu trên.

3.3.1. Các phương pháp tối ưu cơ bản dựa trên mã hợp ngữ

i. Thay đổi thứ tự thực hiện lệnh

Thay đổi thứ tự thực hiện lệnh là kỹ thuật được sử dụng trong hầu hết các CPU hiệu năng cao trong đó thứ tự thực hiện lệnh ban đầu được phá vỡ để thay thế bằng thứ tự thực hiện tốt hơn dựa trên tính sẵn sàng của dữ liệu vào. Kỹ thuật này nhằm tăng khả

năng tận dụng CPU, tránh trường hợp CPU rỗi khi đợi nhận dữ liệu cho câu lệnh bằng cách có thể xử lý các câu lệnh tiếp sau có khả năng thực hiện ngay. Ví dụ trong khi đợi lấy dữ liệu để thực hiện phép cộng số thực, có thể thực hiện luôn phép cộng số nguyên vì bộ cộng số thực và bộ cộng số nguyên là hai mạch phần cứng khác nhau. Một số CPU hỗ trợ tính năng này như PowerPC 601, MIPS R10000, HP PA-8000, AMD K5, v.v.

Bảng 3.9. Kỹ thuật đổi tên thanh ghi

Đoạn mã ban đầu	Sau khi đổi tên thanh ghi

`mov eax, [mem1]`
`imul eax, 6`
`mov [mem2], eax`
`mov ebx, [mem3]`
`add ebx, 2`
`mov [mem4], ebx`
	`mov eax, [mem1]`
	`imul eax, 6`
	`mov [mem2], eax`
	`mov eax, [mem3]`
	`add eax, 2`
	`mov [mem4], eax`

ii. Đổi tên thanh ghi

Kỹ thuật này làm tăng tính độc lập để có thể thực hiện song song hóa trong CPU bằng cách đưa các thanh ghi lôgic về cùng tên trong một khối lệnh. Trong các thao tác ghi dữ liệu, thanh ghi lôgic chung sẽ được cấp phát các thanh ghi vật lý khác nhau. Do đó có thể tính toán độc lập và song song. Để thực hiện kỹ thuật này, CPU phải hỗ trợ tính năng đổi tên thanh ghi và phải có nhiều thanh ghi. Kỹ thuật này được minh họa trong Bảng 3.9.

iii. Tránh sử dụng các thanh ghi thành phần

Khi sử dụng các thanh ghi thành phần có thể làm tăng độ trễ của các lệnh phụ thuộc dữ liệu. Xét ví dụ sau:

```
mov eax, [mem1]        ; 32 bit memory operand
imul eax, 6
mov [mem2], eax
mov ax, [mem3]         ; 16 bit memory operand
add ax, 2
mov [mem4], ax
```

Trong đoạn mã trên, lệnh `mov ax, [mem3]` độc lập dữ liệu với lệnh `imul eax, 6` nhưng `ax` là 16 bit thấp của thanh ghi `eax` nên nó phải chờ lệnh `imul eax, 6` thực hiện xong. Có thể giải quyết vấn đề này bằng kỹ thuật đổi tên thanh ghi. Nhưng hầu hết các CPU không hỗ trợ đổi tên thanh ghi thành phần nên sử dụng thanh ghi thành phần trong trường hợp này sẽ làm tăng thời gian chờ.

iv. Thay thế lệnh phức tạp thành các lệnh tương đương

Kỹ thuật này cũng nhằm tăng tính độc lập dữ liệu giữa các lệnh trong chương trình. Ví dụ trong Bảng 3.10, lệnh `push eax` được thay bằng hai lệnh `sub esp, 4` và `mov [esp], eax`. Do đó lệnh `call SomeFunction` chỉ phải chờ lệnh `sub esp, 4` thay vì phải chờ lệnh `push eax`.

Bảng 3.10. Thay thế lệnh phức tạp bằng các lệnh đơn giản

Đoạn mã ban đầu	Đoạn mã thay thế

`push eax`
`call SomeFunction`
	`Sub esp, 4`
	`Mov [esp], eax`
	`call SomeFunction`

v. Xem xét các đơn vị thực thi

Trong mỗi CPU bao gồm nhiều đơn vị thực thi và các đơn vị thực thi được xây dựng bằng các mạch phần cứng nên có thể thực hiện nhiều công việc song song. Ví dụ CPU có hai bộ cộng số nguyên thì có thể thực hiện đồng thời hai phép cộng nguyên độc lập dữ liệu. Mỗi CPU cũng thường có một bộ cộng số thực dấu chấm động và một bộ nhân dấu chấm động nên có thể thực hiện một phép cộng dấu chấm động và một phép nhân dấu chấm động đồng thời. Nhiều lệnh có thể thực hiện trong nhiều chu kỳ đồng hồ. Điều này dẫn đến thời gian chờ thực hiện lệnh tiếp theo bị kéo dài. Để cải tiến vấn đề này, nhiều CPU hỗ trợ kỹ thuật đường ống lệnh. Để thực hiện đường ống lệnh, các lệnh cần độc lập dữ liệu.

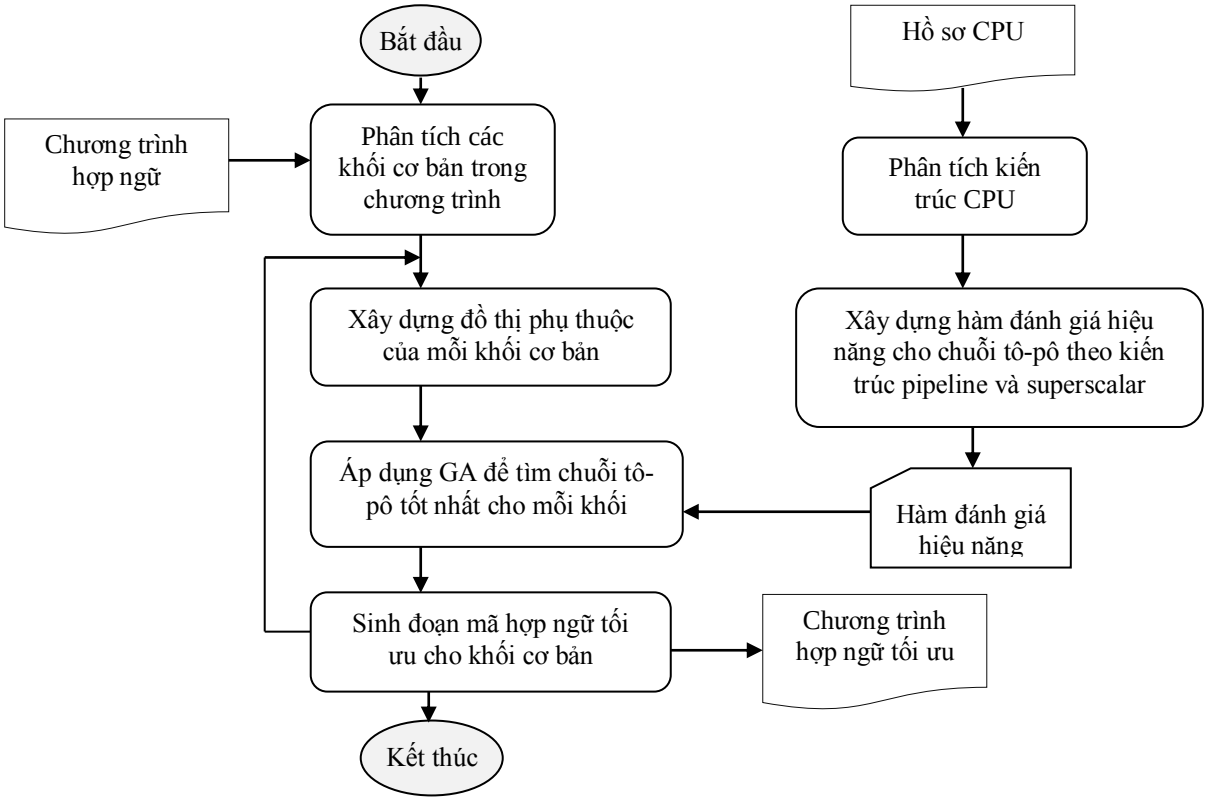
vi. Phá vỡ các chuỗi phụ thuộc

Để có thể thay đổi thứ tự thực hiện ban đầu nhằm cải tiến hiệu năng, việc phá vỡ các chuỗi phụ thuộc là cần thiết. Độ trễ của chuỗi phụ thuộc bằng tổng thời gian của các lệnh trong chuỗi và không thể rút gọn do đó chỉ có thể tháo gỡ chuỗi phụ thuộc mới có thể rút ngắn thời gian thực hiện của chuỗi. Việc tháo gỡ chuỗi phụ thuộc có thể thực hiện bằng các kỹ thuật làm tăng tính độc lập dữ liệu của các câu lệnh trong chuỗi.

vii. Hạn chế sử dụng lệnh nhảy và chương trình con

Các lệnh nhảy thường thay đổi luồng điều khiển của chương trình và các lệnh nhảy xa phải chuyển đến câu lệnh nằm trong đoạn nhớ khác nên việc dùng nhiều lệnh nhảy trong chương trình cũng làm tăng thời gian thực thi. Cần hạn chế hoặc loại bỏ các lệnh nhảy bằng các lệnh tương đương để đạt hiệu năng tốt hơn. Khi chuyển điều khiển cần thay đổi giá trị trong cặp thanh ghi đoạn/độ dời và phải lưu lại giá trị trong các thanh ghi

của CPU vào ngăn xếp cũng như khôi phục giá trị các thanh ghi khi trở lại thực hiện chương trình chính. Do đó hạn chế dùng chương trình con cũng giúp cải tiến hiệu năng.



Hình 3.13: Quy trình tối ưu hiệu năng dựa trên lập lịch các lệnh

3.3.2. Tối ưu hiệu năng dựa trên lập lịch các lệnh

i. Ý tưởng và quy trình nghiên cứu

Trên cơ sở các phương pháp tối ưu cơ bản mức mã hợp ngữ, trong phần này, chúng tôi đề xuất một phương pháp lập lịch các lệnh để tối ưu hiệu năng cho các vi xử lý có kiến trúc đường ống lệnh và kiến trúc siêu vô hướng dựa trên thuật toán di truyền (GA). Ý tưởng chính của phương pháp này là tìm một thứ tự thực hiện lệnh sao cho tổng kích thước các đoạn trễ (stall) nhỏ nhất đối với kiến trúc đường ống lệnh và đạt mức song song hóa cao nhất đối với kiến trúc siêu vô hướng. Quy trình nghiên cứu và triển khai phương pháp tối ưu này được chỉ ra trong Hình 3.13.

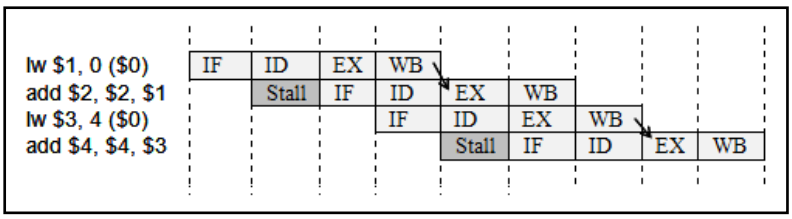
ii. Xây dựng hàm đánh giá hiệu năng

Để có thể áp dụng thuật toán lập lịch nhằm tối ưu hiệu năng, đầu tiên, chúng tôi xây dựng hàm đánh giá hiệu năng trên mỗi chuỗi tô-pô. Một chuỗi tô-pô chính là một thứ tự thực hiện lệnh thỏa mãn đồ thị phụ thuộc. Trong phần này, chúng tôi xây dựng hàm đánh giá cho cả hai loại kiến trúc vi xử lý là đường ống lệnh và siêu vô hướng.

Hàm đánh giá hiệu năng cho kiến trúc đường ống lệnh

Để không làm thay đổi ngữ nghĩa, tập lệnh hợp ngữ trong mỗi khối cơ bản chỉ có thể được thực hiện theo các chuỗi tô-pô trên đồ thị phụ thuộc. Trong kiến trúc đường ống, các lệnh được thực hiện gối chồng nhau nhưng vẫn phải thỏa mãn đồ thị phụ thuộc. Giả sử CPU có kiến trúc N_s-đoạn, mỗi đoạn được thực hiện trong một chu kỳ đồng hồ và thời gian mỗi đoạn là T_s. Trong trường hợp kiến trúc CPU là 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 hoàn thành trung bình của 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 và được minh họa trong Hình 3.14. Trong đó, *IF* là giai đoạn lấy lệnh, *ID* là giai đoạn giải mã lệnh, *EX* là giai đoạn thực thi và *WB* là giai đoạn ghi kết quả.

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



Hình 3.14: Minh họa thời gian trễ trong kiến trúc đường ống lệnh

Từ phân tích trên, có thể chỉ ra: tập lệnh thực hiện theo thứ tự khác nhau sẽ có thời gian thực hiện khác nhau do tổng thời gian trễ khác nhau. Theo đó, chúng tôi xây dựng hàm đánh giá hiệu năng chính là hàm tính tổng thời gian trễ như trong công thức (3.6).

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

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.

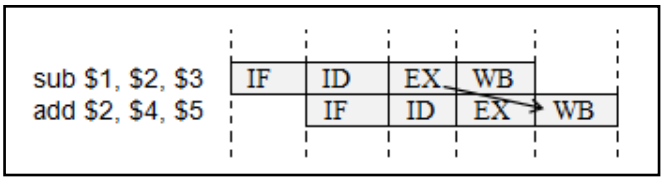
Trong kiến trúc đường ống N_s-đoạn, để thực hiện lệnh i cần xét sự phụ thuộc dữ liệu của lệnh i với $N_s - 1$ lệnh đứng trước. Độ trễ được tính bằng thời gian lớn nhất mà

lệnh i phải đợi một trong $N_s - 1$ lệnh đứng trước. Theo đó, chúng tôi xây dựng công thức (3.7) để tính độ trễ của lệnh i.

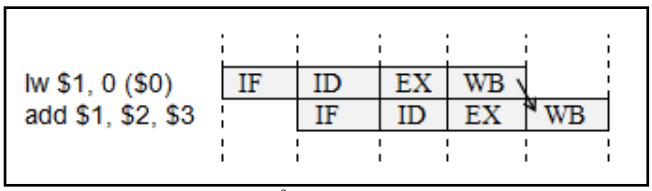
$$s_i = \max_{0 \le j < N_s - 1} (n_d - j) \times T_s \tag{3.7}$$

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

Giá trị của n_d phụ thuộc vào kiểu phụ thuộc dữ liệu giữa các lệnh và số đoạn của kiến trúc đường ống lệnh. Có ba kiểu phụ thuộc dữ liệu là *đọc sau ghi* như trong Hình 3.14, *ghi sau đọc* như trong Hình 3.15 và *ghi sau ghi* như trong Hình 3.16. Như minh họa trong Hình 3.15 và 3.16, mặc dù hai lệnh phụ thuộc dữ liệu nhưng độ trễ vẫn bằng 0. Theo đó, khi lập trình hàm đánh giá hiệu năng, chúng ta cần phải xét các kiểu phụ thuộc dữ liệu này.



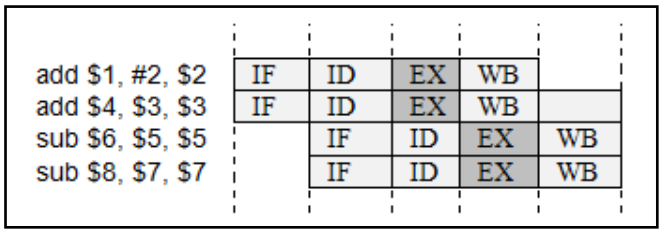
Hình 3.15: Kiểu phụ thuộc *ghi sau đọc*



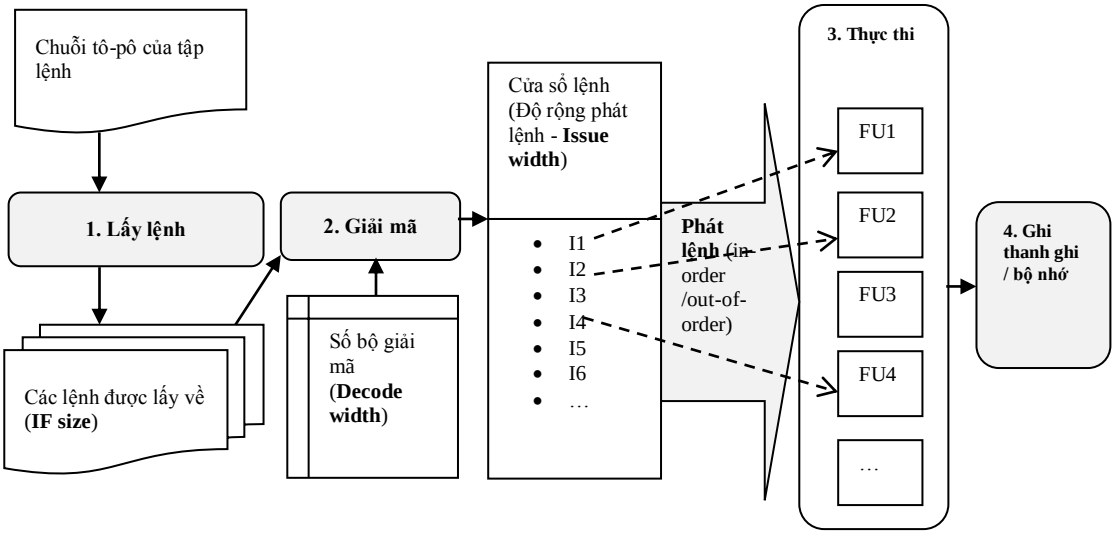
Hình 3.16: Kiểu phụ thuộc *ghi sau ghi*

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 phải có thêm các đơn vị chức năng trong CPU. Ví dụ, CPU có hai bộ cộng thì có thể thực hiện song song hai phép cộng tại một thời điểm. Hình 3.17 minh họa hoạt động bên trong CPU có kiến trúc siêu vô hướng và Hình 3.18 minh họa việc thực thi song song bên trong CPU theo kiến trúc siêu vô hướng.



Hình 3.18: Minh họa thực hiện lệnh trong kiến trúc siêu vô hướng



Hình 3.17: Hoạt động bên trong CPU có kiến trúc siêu vô hướng

Để xây dựng hàm đánh giá hiệu năng, chúng tôi phân tích quá trình thực hiện các câu lệnh trong một CPU siêu vô hướng. Như trong Hình 3.17, thứ tự tô-pô của chuỗi lệnh trong chương trình cũng chính là thứ tự nạp lệnh vào CPU và cũng là thứ tự giải mã lệnh. Cửa sổ lệnh chính là một vùng nhớ đệm trong CPU, chứa các lệnh đã giải mã nhưng chưa được gửi đến đơn vị chức năng tương ứng. Kích thước của cửa sổ lệnh còn gọi là độ rộng phát lệnh, nó xác định tối đa bao nhiêu lệnh có thể được phát đồng thời đến các đơn vị chức năng. Dựa trên kiểu phát lệnh, kiến trúc siêu vô hướng gồm hai kiểu thực thi đó là *in-order* và *out-of-order*. Trong kiểu *in-order*, các lệnh được thực hiện song song nhưng vẫn tuân theo thứ tự của các lệnh trong chương trình. Theo đó, phần cứng điều vận trong CPU luôn kiểm tra sự phụ thuộc dữ liệu trong thời gian chạy. Chỉ khi câu lệnh tiếp theo không phụ thuộc dữ liệu vào câu lệnh trước và có đơn vị chức năng rỗi thì nó mới được gửi đến đơn vị chức năng rỗi này để thực hiện. Trong kiểu *out-of-order*, nếu một câu lệnh bị kẹt do phụ thuộc dữ liệu hoặc thiếu tài nguyên, CPU sẽ tìm câu lệnh tiếp theo độc lập dữ liệu và tài nguyên sẵn sàng để thực hiện. Quá trình tìm kiếm dừng khi số lượng các lệnh được phát bằng kích thước của độ rộng phát lệnh. Như chỉ ra trong Hình 3.18, thời gian lấy lệnh và giải mã không phụ thuộc vào thứ tự thực thi. Đồng thời, thời gian kiểm tra sự phụ thuộc dữ liệu của các lệnh đã giải mã chỉ phụ thuộc vào độ rộng phát lệnh và không đáng kể nên có thể bỏ qua. Do đó thời gian thực thi của chương trình sẽ phụ thuộc vào độ rộng phát lệnh và thứ tự thực thi. Hàm đánh giá hiệu năng cho cả hai kiểu *in-order* và *out-of-order* có thể được xây dựng như trong công thức (3.8).

$$f_p^k = k - 1 + x_k$$

$$x_k = \begin{cases} 1 \; if \; S_I^k \neq \emptyset \\ 0 \; if \; S_I^k = \emptyset \end{cases} \tag{3.8}$$

$$S_I^k = S_I^{k-1} \setminus E_I^{k-1} \cup S_N^k$$

Trong đó:

- $S_I{}^k$ là tập các lệnh đã giải mã trong cửa sổ lệnh tại bước k; tổng số lệnh trong tập này 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 cũng áp dụng thuật toán di truyền để lựa tìm chuỗi tô-pô có hiệu năng tốt nhất. Mỗi nhiễm sắc thể là một chuỗi tô-pô với vị trí 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. Hình 3.19 minh họa cấu trúc dữ liệu như một nhiễm sắc thể được sử dụng trong chương trình.

Các đỉnh	v_1	v_2	v_3	v_4	v_5	v_6	v_7
Thứ tự	2	6	4	5	1	3	7

Hình 3.19: Biểu diễn một nhiễm sắc thể trong GA

Áp dụng thuật toán di truyền, chúng tôi xây dựng chương trình tối ưu thực hiện các công việc chính sau: phân tích chương trình để tìm các khối cơ bản, xây dựng đồ thị phụ thuộc cho mỗi khối cơ bản, lập trình các hàm đánh giá và thuật toán di truyền để tìm chuỗi tô-pô tốt nhất trên đồ thị phụ thuộc của một khối cơ bản. Chương trình được mô tả cụ thể trong *Phụ lục P.1.7*.

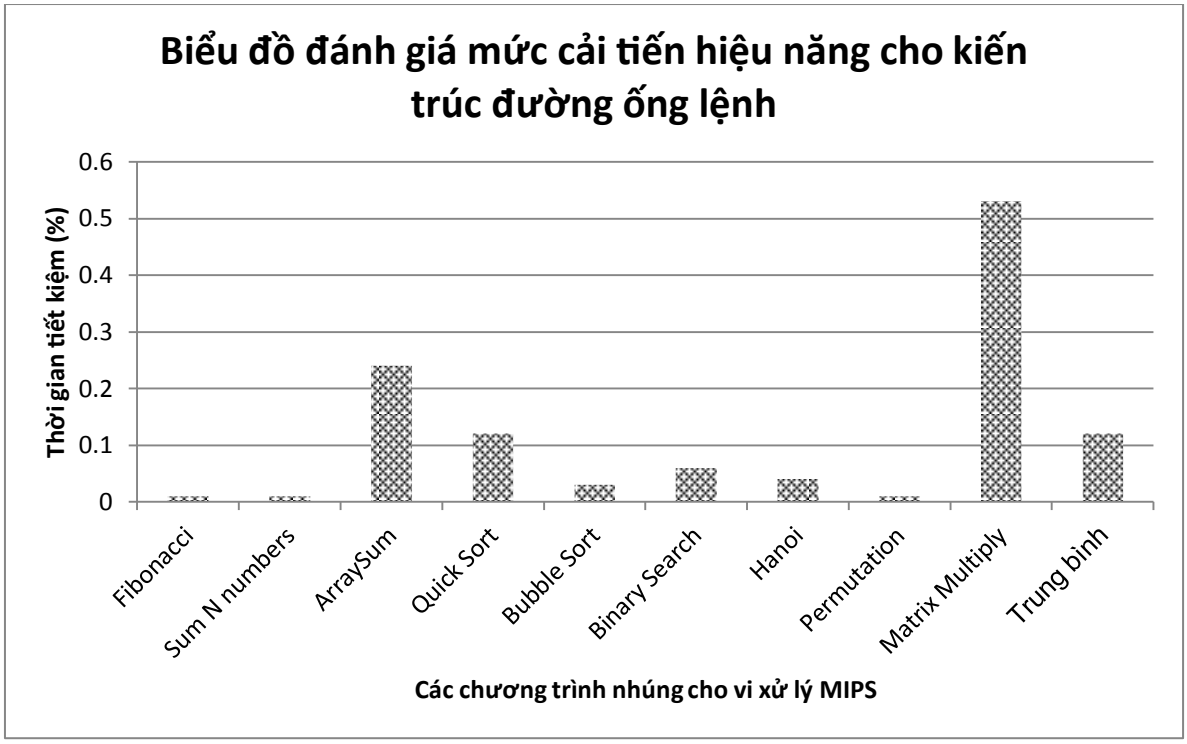
iv. Thực nghiệm và đánh giá

Trong thực nghiệm này, chúng tôi sử dụng *SimpleScalar* để mô phỏng vi xử lý MIPS. Đầu tiên, tạo một trình biên dịch chéo dựa trên GCC để biên dịch mã nguồn C sang mã hợp ngữ cho vi xử lý MIPS. Sử dụng chương trình tối ưu để lập lịch các lệnh hợp ngữ cho các kiểu kiến trúc CPU khác nhau nhằm tối ưu hiệu năng. Như vậy, một chương trình C ban đầu sẽ tương ứng với hai chương trình hợp ngữ đó là chương trình không

được lập lịch và chương trình được lập lịch (tối ưu). Hai chương trình hợp ngữ này được dịch chéo sang mã thực thi MIPS mà không sử dụng các lựa chọn tối ưu của GCC. Cấu hình *SimpleScalar* theo các kiến trúc CPU khác nhau. Chạy mô phỏng chương trình thực thi trên *SimpleScalar* và thống kê kết quả theo chu kỳ đồng hồ (cycle). Đầu tiên, chúng tôi cấu hình CPU mô phỏng theo kiến trúc đường ống lệnh 4 đoạn ($N_s = 4$) và thực hiện các chương trình nhúng cho MIPS được kết quả như trong Bảng 3.11. Mức độ cải tiến hiệu năng được chỉ ra như trong Hình 3.20. Với kiến trúc đường ống lệnh, thời gian tiết kiệm được là 0,12%. Tiếp theo, chúng tôi cấu hình CPU mô phỏng theo kiến trúc siêu vô hướng với cách thực thi là *in-order* và tiến hành thực nghiệm trên các chương trình nhúng cho MIPS đã có. Kết quả thực nghiệm được thống kê trong Bảng 3.12 và biểu đồ đánh giá mức độ cải tiến hiệu năng được chỉ ra trong Hình 3.21. Với kiến trúc siêu vô hướng *in-order*, thời gian tiết kiệm được là 0,91%. Cuối cùng, chúng tôi cấu hình CPU mô phỏng theo kiến trúc siêu vô hướng với cách thực thi *out-of-order* và thực nghiệm với cùng bộ chương trình nhúng trong hai cấu hình trước. Bảng 3.13 tổng hợp các kết quả thực nghiệm và Hình 3.22 trình bày biểu đồ đánh giá mức độ cải tiến hiệu năng trong trường hợp này. Với kiến trúc siêu vô hướng *out-of-order*, thời gian tiết kiệm được là 2,5%.

Bảng 3.11. Tổng hợp kết quả tối ưu hiệu năng dựa trên lập lịch cho kiến trúc đường ống lệnh

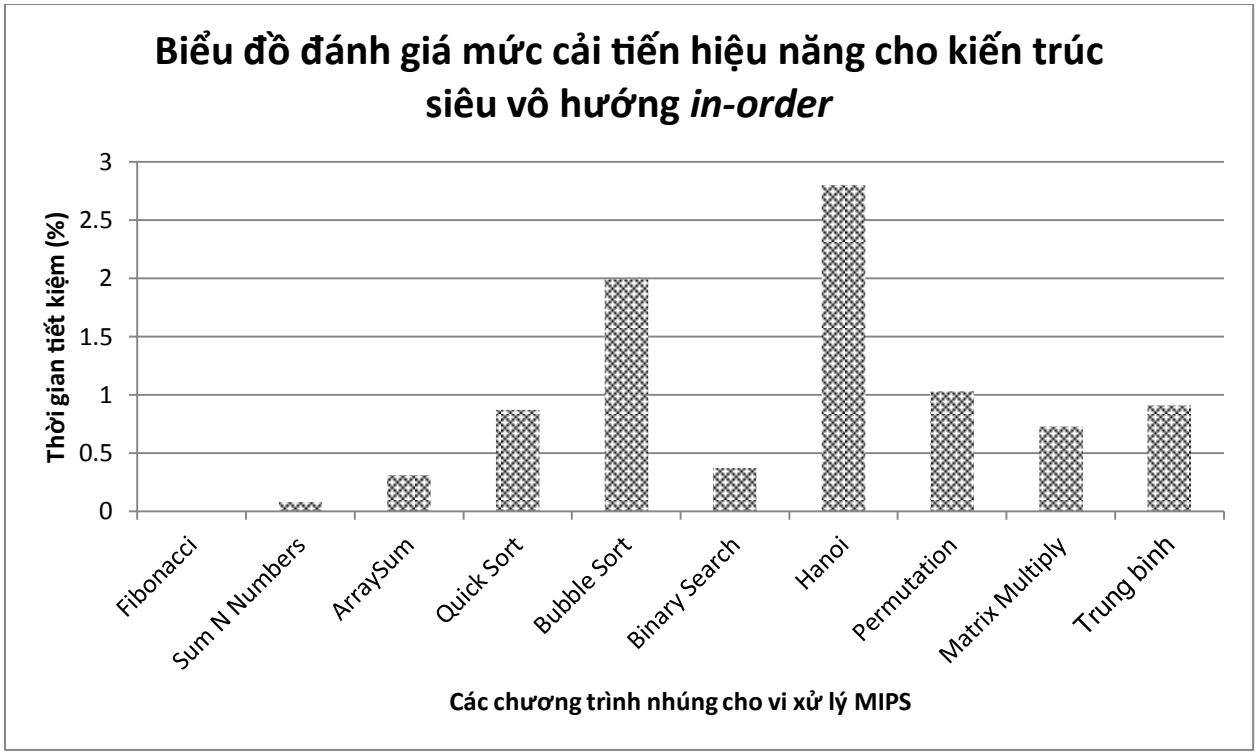
STT	Chương trình	Thời gian thực thi (cycle)		Thời gian tiết kiệm (%)
		Không lập lịch	Lập lịch	
1	Fibonacci	565343	565283	0,01
2	Sum N numbers	3763356	3763104	0,01
3	ArraySum	25611	25550	0,24
4	Quick Sort	89214	89109	0,12
5	Bubble Sort	380851	380734	0,03
6	Binary Search	17557	17546	0,06
7	Hanoi	263279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



Hình 3.20: Biểu đồ đánh giá mức cải tiến hiệu năng dựa trên lập lịch các lệnh cho kiến trúc đường ống lệnh

Bảng 3.12. Tổng hợp kết quả tối ưu hiệu năng dựa trên lập lịch cho kiến trúc siêu vô hướng *in-order*

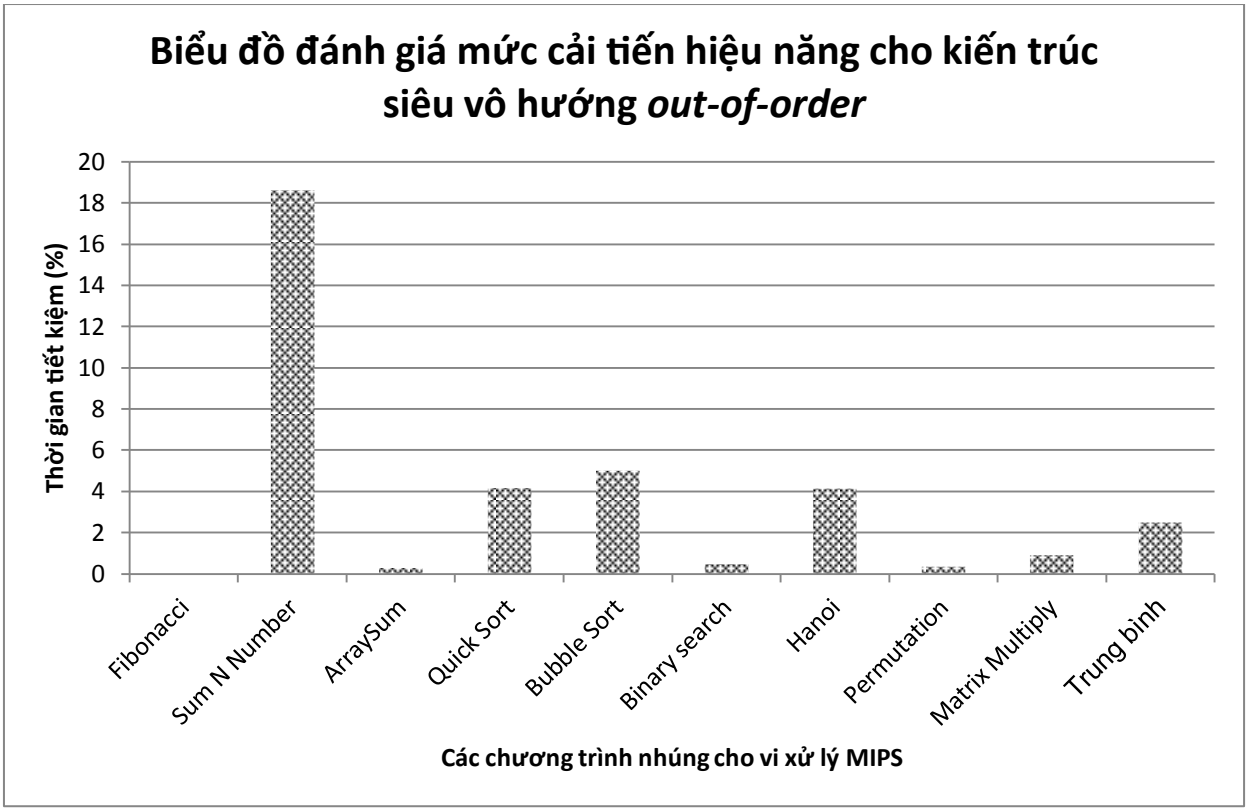
STT	Chương trình	Thời gian thực thi (cycle)		Thời gian tiết kiệm (%)
		Không lập lịch	Lập lịch	
1	Fibonacci	476859	476799	0,01
2	Sum N Numbers	3657599	3654666	0,08
3	ArraySum	23336	23264	0,31
4	Quick Sort	84540	83804	0,87
5	Bubble Sort	369612	362247	1,99
6	Binary Search	16026	15967	0,37
7	Hanoi	247401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



Hình 3.21: Biểu đồ đánh giá mức cải tiến hiệu năng dựa trên lập lịch các lệnh cho kiến trúc siêu vô hướng *in-order*

Bảng 3.13. Tổng hợp 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

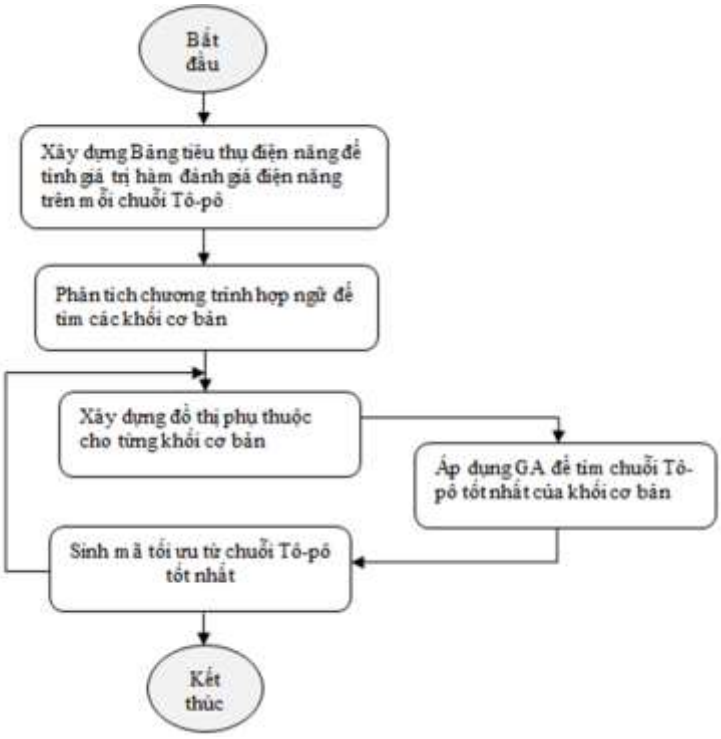


Hình 3.22: Biểu đồ đánh giá mức cải tiến hiệu năng dựa trên lập lịch các lệnh cho kiến trúc siêu vô hướng *out-of-order*

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

i. Ý tưởng và quy trình nghiên cứu

Tiếp tục nghiên cứu theo hướng tiếp cận tối ưu dựa trên lập lịch, trong phần này chúng tôi phát triển phương pháp tối ưu điện năng tiêu thụ dựa trên lập lịch các lệnh theo thuật toán di truyền. 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. Quy trình triển khai được chỉ ra trong Hình 3.23.



Hình 3.23: Quy trình tối ưu điện năng tiêu thụ dựa trên lập lịch các lệnh

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 vào 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 [58]. Vì vậy trong phương pháp này, 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. Kế thừa thuật toán và chương trình trong phần trước, chúng tôi xây dựng lại hàm thích nghi để lập lịch trong nghiên cứu này. Hàm thích nghi được xây dựng dựa vào bảng tiêu thụ điện năng. Dựa 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. Thực nghiệm và đánh giá

Với phương pháp và các chương trình thử nghiệm được tái sử dụng từ phần trước, trong thực nghiệm này, chúng tôi sử dụng công cụ *SimplePower* để đo điện năng tiêu thụ nhằm đánh giá phương pháp tối ưu. Mỗi chương trình với mã nguồn C được biên dịch chéo thành mã hợp ngữ cho MIPS; chương trình hợp ngữ được đưa qua chương trình lập lịch để thu được chương trình hợp ngữ tối ưu; biên dịch chéo sang mã thực thi mà không sử dụng lựa chọn tối ưu nào của GCC. Mô phỏng và đo năng lượng tiêu thụ cho hai phiên bản của cùng một chương trình để đánh giá. Trong Bảng 3.14 và 3.15, điện năng tiêu thụ được đánh giá theo điện dung vì công cụ *SimplePower* ước lượng công suất tiêu thụ tỉ lệ thuận với $V^2C_cf_x$ [111] trong đó V là điện áp nguồn nuôi, f_x là tần số đồng hồ và C_c là điện dung chuyển mạch của các thành phần trong CPU. Trong mỗi chu kỳ đồng hồ, điện năng xấp xỉ bằng $V^2C_cf_x \times t = V^2C_cf_x \times \frac{1}{f_x} = V^2C_c$ với t là chu kỳ đồng hồ. Do V trong CPU không thay đổi nên điện năng tiêu thụ tỉ lệ thuận với C_c. Theo đó, việc đánh giá điện năng tiêu thụ tương đương với đánh giá điện dung chuyển mạch.

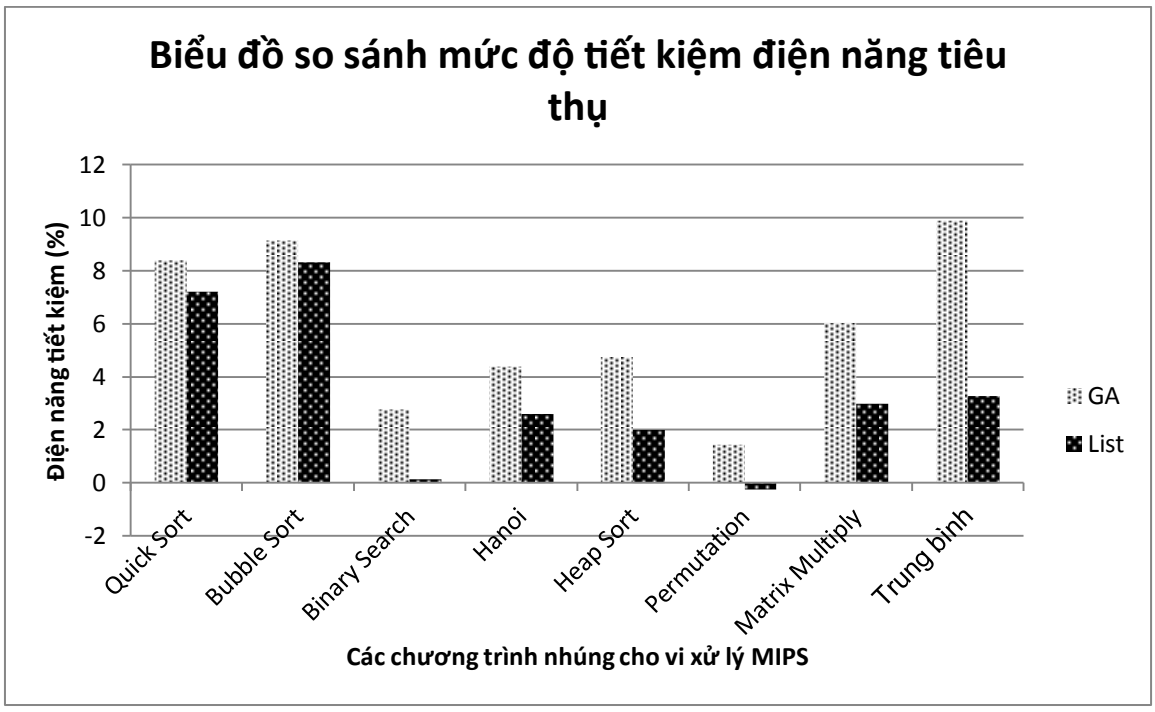
Trong phần này, chúng tôi cũng xây dựng chương trình lập lịch các lệnh theo thuật toán lập lịch *List* do Tiwari, V. và cộng sự sử dụng [58] và được kết quả như trong Bảng 3.15. Biểu đồ so sánh mức độ tiết kiệm điện năng khi lập lịch theo *List* và theo GA được chỉ ra trong Hình 3.24. Theo kết quả này, lập lịch với giải thuật GA theo đề xuất của chúng tôi đạt kết quả tốt hơn do *List* là thuật toán tham ăn nên dễ rơi vào tối ưu cục bộ.

Bảng 3.14. Đánh giá điện năng tiêu thụ thông qua C_c khi lập lịch theo GA

STT	Chương trình	Điện dung chuyển mạch (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

Bảng 3.15. Đánh giá điện năng tiêu thụ thông qua C_c khi lập lịch theo thuật toán List

STT	Chương trình	Điện dung chuyển mạch (pF)		Điện năng tiết kiệm (%)
		Không lập lịch	Không lập lịch	
1	Quick sort	2077771,2516	1928151,7644	7,20
2	Bubble sort	12365628,8339	11335245,3421	8,33
3	Binary search	11500,0162	11483,0485	0,14
4	Hanoi	6341659,6373	6176948,4327	2,60
5	Heap sort	3598279,6097	3526183,7633	2,00
6	Permutation	24001185,1045	24057651,7652	-0.24
7	Matrix Multiply	110309,0015	107020,6322	2,98
Trung bình				3,29



Hình 3.24: Biểu đồ so sánh mức độ tiết kiệm điện năng tiêu thụ dựa trên lập lịch các lệnh theo thuật toán GA và List

3.4. Tổng kết chương

Trong giai đoạn lập trình, tối ưu phần mềm nhúng có thể được thực hiện theo hai mức khác nhau là tối ưu mã nguồn mức cao độc lập kiến trúc đích, tối ưu mã hợp ngữ hướng đến các CPU đích. *Về tối ưu mã nguồn mức cao độc lập kiến trúc đích*, đầu tiên, chúng tôi tóm lược cơ sở lý thuyết về các phương pháp tối ưu đã nghiên cứu. Dựa trên đó chúng tôi đã đề xuất và phát triển hai phương pháp cải tiến hiệu năng đó là cải tiến hiệu năng dựa trên thay thế biểu thức tương đương và cải tiến hiệu năng phần mềm nhúng trong môi trường phân tán dựa trên nén dữ liệu. Trong phương pháp cải tiến hiệu năng dựa trên thay thế các biểu thức tương đương, chúng tôi đã xây dựng chương trình phân tích và thay thế các biểu thức tương đương, tiến hành thực nghiệm và đánh giá mức độ cải tiến. So các lựa chọn tối ưu trong bộ công cụ biên dịch GCC, cải tiến của chúng tôi đạt hiệu năng tốt hơn. Cải tiến này đã được công bố trong công trình [CT12]. Trong phương pháp dựa trên nén dữ liệu, chúng tôi đã đưa ra mô hình triển khai, xây dựng các công thức để đánh giá mức cải tiến thời gian thực thi và xác định điều kiện nén. Phương pháp này cũng được thử nghiệm với phần mềm nhận dạng chữ Nôm trên điện thoại di động theo mô hình dịch vụ web. Đây là thực nghiệm có ý nghĩa quan trọng trong bảo tồn di sản văn hóa Việt Nam.

Về tối ưu mức mã hợp ngữ hướng đến các CPU đích, chúng tôi đã tiếp cận theo các mục tiêu tối ưu như tối ưu hiệu năng và điện năng tiêu thụ. Các tiêu chí trong mức tối ưu này mang tính đặc thù của phần mềm nhúng vì nó hướng đến các CPU cụ thể. Trong tiêu chí tối ưu hiệu năng, sau khi tóm lược các phương pháp tối ưu cơ bản, chúng tôi đã đề xuất và phát triển phương pháp tối ưu mới đó là tối ưu hiệu năng dựa trên lập lịch các lệnh theo thuật toán di truyền cho kiến trúc đường ống lệnh và kiến trúc siêu vô hướng (cả *in-order* và *out-of-order*). Trong tiêu chí tối ưu điện năng tiêu thụ, chúng tôi đã đề xuất và phát triển phương pháp lập lịch để tối ưu điện năng tiêu thụ dựa trên thuật toán di truyền. Hai phương pháp tối ưu đề xuất này đã được công bố trong các công trình [CT13, CT15].

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

Trong giai đoạn thực thi, phần mềm nhúng đã được tích hợp trong các hệ thống nhúng cụ thể. Do sự đa dạng của phần cứng hệ thống nhúng cũng như môi trường thực thi nên tối ưu phần mềm nhúng trong giai đoạn này cũng có vai trò quan trọng. Cũng như tối ưu trong các giai đoạn khác, trong giai đoạn thực thi, chúng ta 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, mã 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 hướng dữ liệu và tối ưu hóa chương trình thực thi. Nội dung chương này sẽ tiếp cận và trình bày các kỹ thuật tối ưu theo ba nhóm này:

- Tối ưu môi trường thực thi: Bao gồm kỹ thuật biên dịch tạm, lập lịch tiến trình, tổ chức lưu trữ đệm dữ liệu và cải tiến môi trường phần cứng
- Tối ưu hướng dữ liệu: Bao gồm các kỹ thuật tối ưu dựa trên chuyên biệt hóa dữ liệu, nén dữ liệu và cải tiến môi trường truyền dữ liệu
- Tối ưu chính chương trình thực thi: Bao gồm tái cấu trúc mã thực thi, chuyên biệt hóa mã thực thi, v.v.

Trên cơ sở phân tích hiện trạng nghiên cứu và các thách thức như trong *Mục 1.2.3 Chương 1*, trong chương này, chúng tôi đề xuất và phát triển phương pháp tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU. Đây là phương pháp tối ưu mới dựa trên sự kết hợp phần cứng và phần mềm. Nội dung chương được cấu trúc như sau: *Mục 4.1* trình bày về các kỹ thuật tối ưu môi trường thực thi, *Mục 4.2* trình bày về các kỹ thuật cải tiến môi trường truyền dữ liệu, *Mục 4.3* trình bày các phương pháp tối ưu hóa mã thực thi và *Mục 4.4* tổng kết chương.

4.1. Tối ưu môi trường thực thi

4.1.1. Kỹ thuật biên dịch tạm

Trong các phần mềm nhúng mức cao như phần mềm trên điện thoại di động, chương trình được thực thi trong các môi trường phần cứng đa dạng. Do yêu cầu chạy phần mềm trên nhiều thiết bị phần cứng khác nhau dẫn đến sự ra đời của máy ảo. Chương trình nguồn không được dịch ra mã máy đích cụ thể mà được biên dịch ra dạng mã trung gian chạy trên máy ảo. Ví dụ mã bytecode chạy trên máy ảo Java hay mã MSIL chạy trên máy ảo .NET. Khi thực thi, mã máy ảo sẽ được dịch sang mã máy đích để thực hiện. Các thế hệ đầu của máy ảo thường sử dụng trình thông dịch – dịch và chạy lần lượt từng lệnh nên hiệu năng thấp. JIT là kỹ thuật biên dịch tạm trong các trình thông dịch nhằm hỗ trợ tối ưu hiệu năng. JIT dựa trên hai ý tưởng chính là biên dịch tại thời gian chạy và biên dịch động. Theo đó, JIT cho phép thực thi các kỹ thuật tối ưu trong thời gian chạy vượt qua khả năng của các trình biên dịch tĩnh bằng việc điều chỉnh các tham số và tối ưu theo các đầu vào thực tế cùng với một số nhân tố thời gian chạy khác.

Mục đích chung của các kỹ thuật JIT là đạt được tính ưu việt về hiệu năng của biên dịch tĩnh và đảm bảo tính khả chuyển và tùy biến của các kỹ thuật thông dịch mã máy ảo. Nói cách khác kỹ thuật JIT nhằm đạt ba mục tiêu chính là hiệu năng, tính khả chuyển và biên dịch động tại thời gian thực thi. Sử dụng JIT không chỉ nhanh hơn nhiều so với thông dịch truyền thống mà trong một số trường hợp nó có thể đạt hiệu năng tốt hơn biên dịch tĩnh. JIT có các ưu điểm chính sau:

- Tối ưu hướng đến CPU đích và hệ điều hành cụ thể. Khi tối ưu mã nguồn như đề cập trong chương trước, việc chỉ rõ biên dịch cho kiến trúc đích nào cũng cải tiến hiệu năng đáng kể. Tuy nhiên có nhiều trường hợp trong quá trình biên dịch tĩnh chỉ xác định họ kiến trúc mà chưa xác định CPU cụ thể nào. Do đó, việc biên dịch được trì hoãn đến giai đoạn thực thi nhằm xác định rõ CPU đích cũng góp phần cải tiến hiệu năng hơn.
- Hệ thống có thể thu thập các thông tin thống kê về các chương trình và môi trường thực thi thực tế để tổ chức lại và biên dịch nhằm cải tiến hiệu năng.
- Hệ thống có thể thực hiện các tối ưu toàn cục mà không mất các ưu điểm của liên kết động. Vì liên kết động chỉ được thực hiện trong giai đoạn thực thi.
- JIT cho phép tái tổ chức mã thực thi dễ dàng nên tận dụng bộ nhớ đệm tốt hơn.

4.1.2. Phương pháp tối ưu dựa trên lập lịch tiến trình

Vấn đề tối ưu trong giai đoạn thực thi thường không tập trung vào một phần mềm nhúng cụ thể mà tập trung vào môi trường thực thi để tối ưu toàn bộ hệ thống nhúng. Một số hướng tiếp cận mới đã ứng dụng trí tuệ nhân tạo, dựa trên các mô hình học máy để

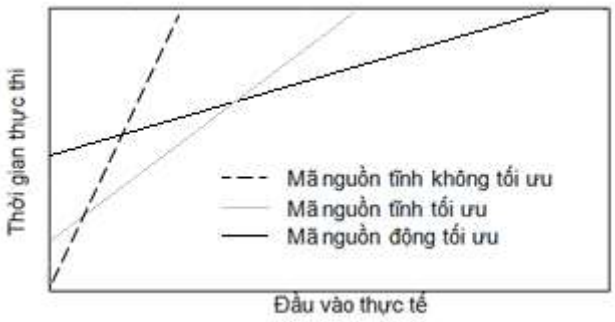
quyết định ngừng thực hiện một số ứng dụng hiện thời nhằm tiết kiệm thời gian CPU hoặc tiết kiệm năng lượng cho ứng dụng thời gian thực. Điển hình như trong nghiên cứu [14], các tác giả đã áp dụng việc xây dựng mô hình quyết định Markov để quản lý các tiến trình. Nghiên cứu này sử dụng mô hình Markov để tạo bộ tối ưu kiểm soát ứng dụng đang thực thi, đưa ra quyết định kích hoạt, tạm dừng tiến trình dựa trên các độ đo như thời gian cuộc gọi, thời gian gửi tin nhắn, đồng bộ dữ liệu, v.v. Từ đó có thể ước lượng mức năng lượng và tiết kiệm năng lượng cho các ứng dụng có độ ưu tiên cao kết thúc. Phương pháp này không tập trung vào tối ưu năng lượng tiêu thụ của phần mềm mà tập trung vào thời gian, tác vụ để đưa ra tham số và biểu thức hóa nhằm phân bổ tài nguyên tối ưu. Trong nghiên cứu này, tác giả đã trình bày một thư viện lập trình mô hình Markov để tối ưu phần mềm trên điện thoại di động và minh họa bằng thực nghiệm trên nền tảng Android của Google.

4.1.3. Tối ưu trong thời gian thực thi dựa trên chuyên biệt hóa

Cách tiếp cận chuyên biệt hóa trong thời gian thực thi để tối ưu hiệu năng đã được nghiên cứu rộng rãi và đã được áp dụng trong một số máy ảo như máy ảo Java, máy ảo .NET, v.v. Có ba cách tiếp cận chuyên biệt hóa để tối ưu trong thời gian thực thi đó là chuyên biệt mã nguồn, chuyên biệt hóa dựa trên dữ liệu và chuyên biệt thuật toán. Trong chuyên biệt hóa mã nguồn, tùy theo giá trị của các tham số đầu vào trong thời gian thực thi mà mỗi lệnh hoặc đoạn lệnh được phân lớp vào các mẫu chuyên dụng hoặc được thay thế bằng đoạn lệnh cụ thể để thực thi với hiệu năng tốt hơn. Trong chuyên biệt dữ liệu, tùy theo loại dữ liệu, kích thước, mục đích sử dụng mà dữ liệu được phân nhóm theo các mẫu chuyên dụng khác nhau về kích thước, loại bộ nhớ (bộ nhớ ẩn, bộ nhớ trong hoặc thanh ghi) và kiểu truy xuất. Trong chuyên biệt thuật toán, các thuật toán thực hiện cũng được phân lớp theo các mẫu chuyên dụng. Mỗi mẫu chuyên dụng đã được lập trình để tối ưu hiệu năng.

i. Chuyên biệt hóa mã nguồn

Chuyên biệt hóa mã nguồn có thể được thực hiện trong một máy đích cụ thể hoặc trong các máy ảo. Trong các máy đích cụ thể, mã nguồn có thể được phân lớp theo các lệnh chuyên dụng được thực hiện bằng phần cứng để cải tiến hiệu năng hoặc được thực hiện bởi CPU. Trong các máy ảo, các đoạn lệnh có thể được phân lớp theo các mẫu hoặc được cụ thể hóa theo các giá trị đầu vào thực tế trong thời gian chạy. Sau đó có thể thực hiện các kỹ thuật tối ưu mã nguồn trước khi chuyển cho các chương trình dịch để biên dịch và thực thi. So sánh giữa biên dịch tĩnh với biên dịch động mã nguồn được chuyên biệt hóa và tối ưu trong thời gian thực thi được minh họa như trong Hình 4.1.



Hình 4.1: So sánh giữa tối ưu tĩnh và tối ưu động mã nguồn đã chuyên biệt hóa

Bảng 4.1. Chuyên biệt hóa mã nguồn

(1) Mã nguồn ban đầu	(2) Mã nguồn được chuyên biệt hóa (size = 5, u = {14,0,38,12,1})

`int dot_product(int size, int u[], int v[]) {`
`int res = 0;`
`for (i = 0; i < size; i++) {`
`  res = res + u[i] * v[i];`
`}`
`return res;`
`}`
	`int dot_product_1(int v[]) {`
	`int res = 0;`
	`for (i = 0; i < 5 ; i++) {`
	`res = res + {14,0,38,12,1}[i] * v[i];`
	`}`
	`return res;`
	`}`

(3) Tối ưu mã nguồn chuyên biệt dựa trên gỡ vòng lặp	(4) Tối ưu dựa trên loại bỏ mã không cần thiết

`int dot_product_1(int v[]) {`
`  int res = 0;`
`  res = res + 14 * v[0];`
`  res = res + 0 * v[1];`
`  res = res + 38 * v[2];`
`  res = res + 12 * v[3];`
`  res = res + 1 * v[4];`
`  return res;`
`}`
	`int dot_product_1(int v[]) {`
	`  int res;`
	`  res = 14 * v[0];`
	`  res = res + 38 * v[2];`
	`  res = res + 12 * v[3];`
	`  res = res + v[4];`
	`  return res;`
	`}`

Như chỉ ra trong Hình 4.1, trong quá trình thực thi, với dữ liệu đầu vào và các điều kiện thực thi thực tế, mã nguồn động tối ưu có thời gian thực thi ban đầu cao nhưng thời gian thực thi tăng chậm do các kỹ thuật tối ưu được áp dụng với dữ liệu và các ràng buộc thực thi thực tế. Để sinh mã nguồn động, các kỹ thuật chuyên biệt hóa mã nguồn sẽ được áp dụng. Một kỹ thuật chuyên biệt hóa mã nguồn phổ biến nhất là thay thế các biến bằng

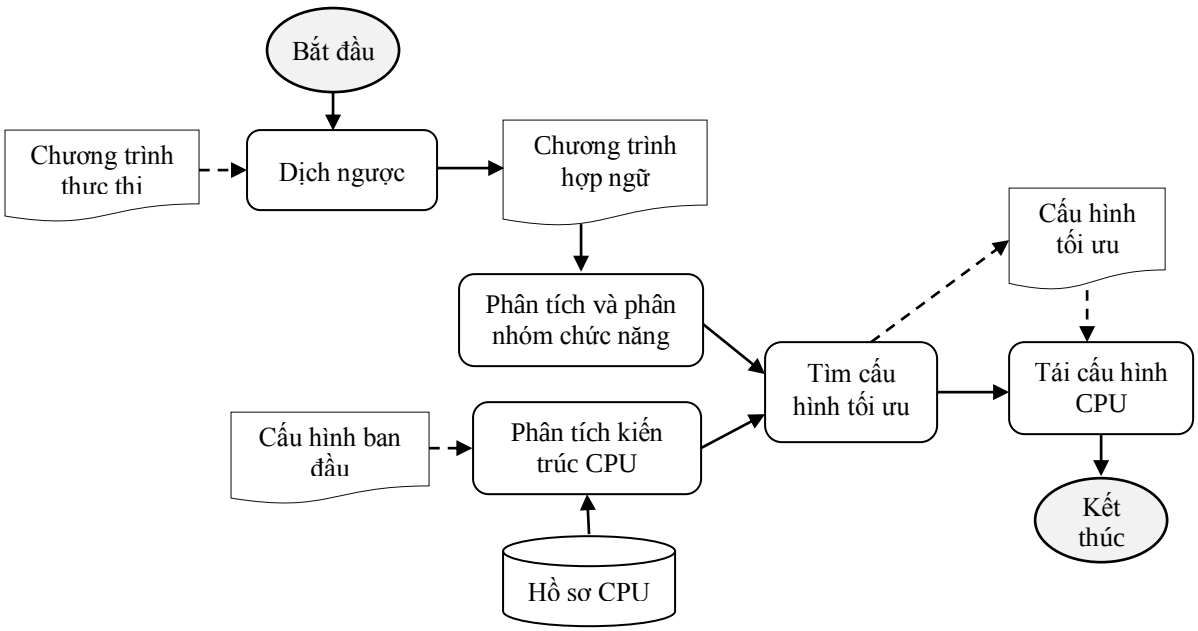
các giá trị thực tế trong mỗi quá trình thực thi cụ thể. Bảng 4.1 minh họa mã nguồn ban đầu, mã nguồn được chuyên biệt hóa và mã nguồn tối ưu.

ii. Chuyên biệt hóa thuật toán

Chuyên biệt hóa thuật toán trong thời gian thực thi là phương pháp tạo các thuật toán hiệu quả cho các tác vụ hướng tính toán theo các loại cụ thể. Nghĩa là mỗi thuật toán sẽ được lập trình theo một số phiên bản khác nhau, tùy theo dữ liệu và các ràng buộc thực tế để lựa chọn phiên bản hiệu quả nhất. Cũng như chuyên biệt hóa mã nguồn dựa trên mẫu, các mẫu chuyên biệt của thuật toán có thể được xây dựng theo phần mềm hoặc phần cứng.

iii. Chuyên biệt dựa trên hóa dữ liệu

Chuyên biệt dữ liệu là kỹ thuật phân đoạn thời gian trong chương trình. Cụ thể, kỹ thuật chuyên biệt dữ liệu chia chương trình thành hai giai đoạn: giai đoạn nạp và giai đoạn đọc. Giai đoạn nạp đánh giá các cấu trúc tĩnh và tạo vùng đệm chứa các giá trị của các cấu trúc tĩnh. Giá trị của các cấu trúc tĩnh có thể xuất hiện trong các ngữ cảnh động. Giai đoạn đọc đánh giá các thành phần xây dựng động, tham chiếu đến các vùng đệm của dữ liệu được chuyên biệt cho các giá trị của các cấu trúc tĩnh. Chuyên biệt dữ liệu còn thể hiện bằng việc phân loại và tổ chức lưu trữ dữ liệu phân cấp để tối ưu truy xuất.



Hình 4.2: Quy trình nghiên cứu và thực nghiệm tối ưu năng lượng dựa trên cấu hình CPU

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

i. Ý tưởng và quy trình nghiên cứu

Tối ưu trong giai đoạn thực thi dựa trên kết hợp phần cứng, phần mềm là hướng nghiên cứu mới và triển vọng. Theo hướng tiếp cận này, chúng tôi đề xuất phương pháp mới để tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU. Ý tưởng của phương pháp này 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. Quy trình tối ưu được chỉ ra trong Hình 4.2.

ii. Giải thích phương pháp tối ưu

Phân tích điện năng tiêu thụ của CPU

Để chứng minh tính đúng đắn, khả thi của phương pháp tối ưu, chúng tôi phân tích điện năng tiêu thụ của các thành phần bên trong CPU, tập trung vào các đơn vị chức năng trong CPU. Điện năng tiêu thụ bởi CPU bao gồm ba nhân tố chính đó là điện năng tiêu thụ động, điện năng đoản mạch và điện năng lãng phí khi cường độ dòng điện qua các bóng bán dẫn như trong trong công thức (4.1) [66].

$$P_{CPU} = P_d + P_s + P_l \quad (4.1)$$

Trong đó:

- P_{CPU} là điện năng tiêu thụ của CPU
- P_d là điện năng tiêu thụ động
- P_s là điện năng tiêu thụ do đoản mạch
- P_l là điện năng lãng phí.

Điện năng tiêu thụ động P_d tiêu tốn do các tụ phóng điện và nạp điện trong quá trình thay đổi trạng thái của các cổng lôgic trong CPU. Chỉ khi thực hiện lệnh, các cổng lôgic mới thay đổi trạng thái và gây ra P_d. Năng lượng tiêu thụ động tỉ lệ thuận với tần số đồng hồ, điện dung và bình phương điện áp. Theo đó, năng lượng tiêu thụ này có thể được tính gần đúng theo công thức (4.2) [66].

$$P_d \sim {}^1/_2 C_c V_c^2 \tau f_x \quad (4.2)$$

Trong đó:

- C_c là điện dung
- f_x là tần số đồng hồ
- V_c là điện áp
- τ là tỉ lệ số cổng lôgic thay đổi trạng thái; $0 \leq \tau \leq 1$.

Tiêu thụ điện năng do đoản mạch khi các bóng bán dẫn thay đổi trạng thái. Khi đó một mạch điện ngắn được nối đất trực tiếp gây mất điện năng. Điện năng tiêu thụ này phụ thuộc vào số cổng lôgic và độ phức tạp của cổng. Điện năng lãng phí được ước lượng ở mức bên trong bóng bán dẫn. Điện năng tiêu thụ này gây ra do dòng điện luôn chạy qua các mạch hở ngay cả khi bóng bán dẫn không được sử dụng. Điện năng tiêu thụ động và điện năng tiêu thụ do đoản mạch đều phụ thuộc vào tần số đồng hồ trong CPU còn điện năng lãng phí chỉ phụ thuộc vào điện áp trong CPU.

Chứng minh tính khả thi

Để chứng minh tính khả thi của phương pháp tối ưu này, chúng tôi giả thiết như sau: một CPU có N_u đơn vị chức năng; khi thực hiện một phần mềm nhúng chỉ có N_{u1} đơn vị chức năng được sử dụng; CPU có khả năng cấu hình để bật/tắt các đơn vị chức năng. Gọi năng lượng tiêu thụ của CPU khi thực hiện phần mềm nhúng với cấu hình đầy đủ (mọi đơn vị chức năng đều bật) là E_1 và E_2 là điện năng tiêu thụ của CPU khi thực hiện phần mềm nhúng với cấu hình chỉ bật những đơn vị chức năng được sử dụng. Trên cơ sở phân tích điện năng tiêu thụ ở trên, E_1 có thể được tính theo công thức (4.3) và E_2 có thể được tính theo công thức (4.4).

$$E_1 = \sum_{i=1}^{N_u} (P_d^i + P_s^i + P_l^i) \tag{4.3}$$

$$E_2 = \sum_{i=1}^{N_{u1}} (P_d^i + P_s^i + P_l^i) \tag{4.4}$$

Từ (4.3) ta có:

$$E_1 = \sum_{i=1}^{N_{u1}} (P_d^i + P_s^i + P_l^i) + \sum_{i=1}^{N_u - N_{u1}} (P_d^i + P_s^i + P_l^i) \tag{4.5}$$

Từ (4.4) và (4.5) suy ra:

$$E_1 - E_2 = \sum_{i=1}^{N_u - N_{u1}} (P_d^i + P_s^i + P_l^i) \tag{4.6}$$

Vì khi các đơn vị chức năng không được sử dụng, các cổng lôgic không thay đổi giá trị, các bóng bán dẫn không thay đổi trạng thái nên $P_d = 0$. Do đó, từ công thức (4.6) suy ra điện năng tiết kiệm được khi cấu hình vô hiệu các đơn vị chức năng không được sử dụng như sau:

$$\Delta E = E_1 - E_2 = \sum_{i=1}^{N_u - N_{u1}} P_l^i + P_s^i \tag{4.7}$$

iii. Thực nghiệm và đánh giá

Để triển khai thực nghiệm và đánh giá phương pháp, chúng tôi sử dụng các công cụ mô phỏng CPU có khả năng cấu hình và công cụ đánh giá điện năng tiêu thụ. Công cụ mô phỏng CPU, *SimpleScalar*, được sử dụng để mô phỏng hoạt động của CPU khi thực hiện các chương trình cụ thể theo các cấu hình khác nhau. Công cụ *Sim-Wattch* được sử dụng để ước lượng điện năng tiêu thụ khi chạy một chương trình trên một cấu hình cụ thể. Đồng thời, chúng tôi cũng xây dựng chương trình phân tích mã hợp ngữ và hồ sơ CPU để tìm cấu hình tối ưu như trình bày trong *Phụ lục P.1.6*.

Sau khi sử dụng *objdump* trong GCC để dịch ngược và sử dụng chương trình phân tích mã hợp ngữ để tìm cấu hình tối ưu, chúng tôi tiến hành chạy mô phỏng trên các cấu hình khác nhau để ước lượng điện năng tiêu thụ như trong Hình 4.3. Các kết quả tổng hợp điện năng tiêu thụ của CPU khi thực hiện một chương trình được chỉ ra trong Hình 4.4. Bảng 4.2 tổng hợp và so sánh các kết quả thực nghiệm với các chương trình khác nhau. Trong Bảng 4.2, cấu hình ban đầu được dùng chung cho mọi chương trình để đánh giá điện năng tiêu thụ khi chưa tối ưu. Sau khi thực hiện chương trình tối ưu trong phần trước, tương ứng với mỗi chương trình sẽ có một cấu hình tối ưu cụ thể. Chạy mô phỏng các chương trình này với cấu hình tối ưu để đánh giá điện năng tiêu thụ sau khi đã tối ưu. Kết quả thực nghiệm trong Bảng 4.2 cho thấy phương pháp tối ưu này có thể tiết kiệm được trung bình 17,74% điện năng tiêu thụ. Biểu đồ đánh giá mức điện năng tiết kiệm được mô tả như trong Hình 4.5.

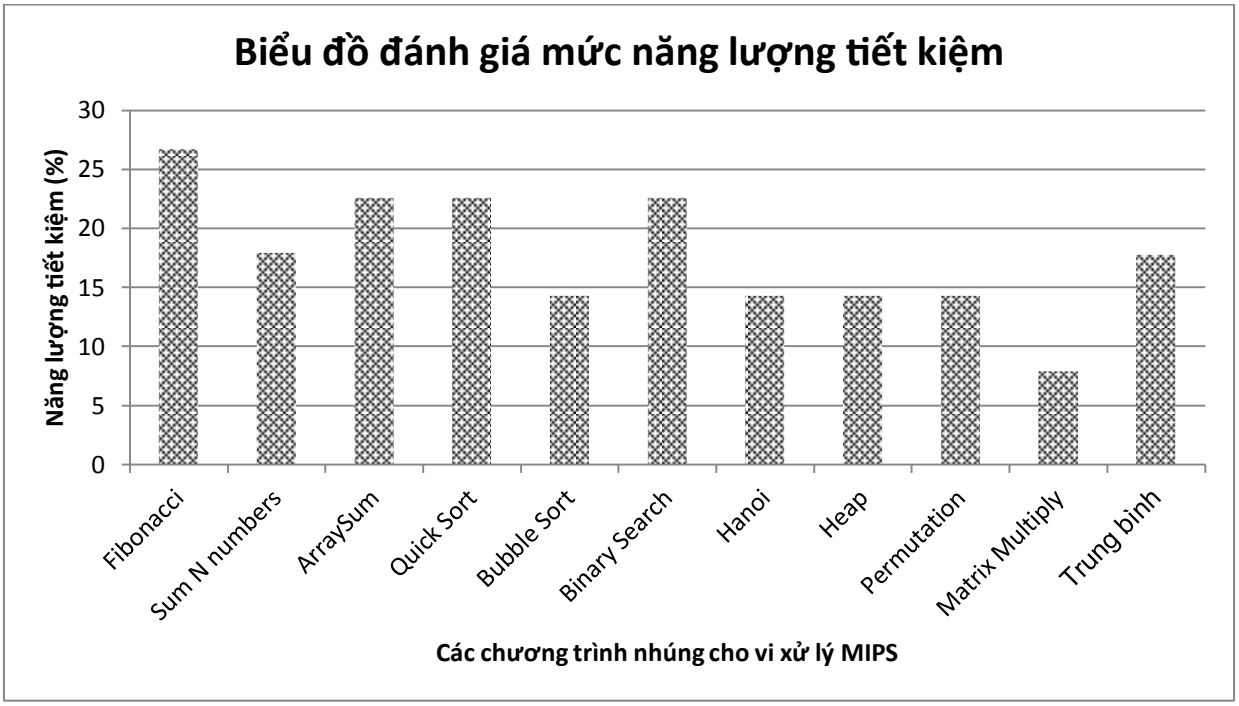
Hình 4.3: Thực hiện *Sim-Wattch* để mô phỏng tiêu thụ năng lượng

```
Total Power Consumption: 100.484
Total Clock Power: 36.3691  (36.9%)
Int ALU Power: 9.32026  (9.46%)
FP ALU Power: 28.5621  (29%)
Instruction Cache Power Consumption: 2.54423  (2.58%)
 decode_power (W): 0.724888
 wordline_power (W): 0.057312
 bitline_power (W): 1.14691
 senseamp_power (W): 0.192
 tagarray_power (W): 0.423116
Itlb_power (W): 0.269848 (0.274%)
Data Cache Power Consumption: 2.59947  (2.64%)
 decode_power (W): 0.325528
 wordline_power (W): 0.0983677
 bitline_power (W): 1.35618
```

Hình 4.4: Một phần kết quả tiêu thụ điện năng

Bảng 4.2. 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	9,800	7,87
Điện năng tiết kiệm trung bình				17,74



Hình 4.5: Biểu đồ đánh giá mức điện năng tiết kiệm từ cấu hình tối ưu

4.2. Tối ưu dựa trên cải tiến môi trường truyền dữ liệu

Nội dung phần này trình bày về cải tiến môi trường truyền dữ liệu trong và giữa các ứng dụng nhúng để tối ưu hiệu năng và hạn chế truy xuất để tiết kiệm năng lượng. Phương pháp tiếp cận phổ biến là cải tiến các gói thông điệp truyền thông, loại bỏ dữ liệu dư thừa trước khi trao đổi. Điển hình như trong nghiên cứu [6], tác giả tiến hành cải tiến

hiệu năng của truyền thông điệp bằng cách loại bỏ các phần thừa trong thông điệp SOAP. Nội dung chủ yếu tập trung vào tối ưu nội dung thông điệp SOAP. Phần dư thừa được loại bỏ và phần tĩnh (các thông tin cấu trúc không thay đổi) của thông điệp được xem là siêu dữ liệu được lưu trữ trong không gian siêu dữ liệu chia sẻ. Thực nghiệm trong [6] chỉ ra có thể tiết kiệm được 83% kích thước thông điệp và 41% kích thước bộ nhớ. Thông điệp SOAP có thể ứng dụng trong các môi trường khác nhau. SOAP dựa trên XML độc lập nền tảng, khi chuyển dữ liệu từ các dạng khác sang SOAP sẽ làm tăng thêm thời gian xử lý và kích thước.

4.3. Tối ưu hóa chương trình thực thi dựa trên mã tự sửa

Mã tự sửa (SMC – Self-modifying code) là đoạn mã có khả năng tự thay thế các chỉ thị của nó trong quá trình thực thi để cải tiến hiệu năng. Mã tự sửa được xem như một thay thế của phương pháp "thiết lập cờ" và xử lý các nhánh của chương trình điều kiện; thường được sử dụng để giảm số lần kiểm tra một điều kiện. Việc tự sửa đổi mã thực thi có thể được thực hiện tại các thời điểm sau:

- Chỉ thực hiện trong quá trình khởi tạo: Dựa trên các tham số vào, khi tiến trình được mô tả như một cấu hình phần mềm phù hợp với phần cứng. Sự thay thế của các điểm vào chương trình cũng là một phương pháp tự sửa mã gián tiếp. Tuy nhiên việc này cũng dẫn đến sự cùng tồn tại của một hoặc nhiều đường lệnh thay thế trong bộ nhớ. Điều này làm cho kích thước chương trình tăng lên.
- Thay thế trong quá trình thực thi: Dựa trên các trạng thái chương trình cụ thể, việc sửa đổi và thay thế có thể đạt được trong quá trình thực thi.

Các kỹ thuật tự thay đổi mã của chương trình trong thời gian thực thi có thể được thực hiện ở các mức ngôn ngữ khác khác nhau. Có bốn nhóm kỹ thuật chính đó là nạp chồng các câu lệnh đã tồn tại, trực tiếp tạo ra tập lệnh trong bộ nhớ, tạo hoặc sửa đổi mã nguồn theo các chương trình dịch nhỏ hoặc một trình thông dịch động, tạo động toàn bộ chương trình khi thực thi. Kỹ thuật mã tự sửa đổi được sử dụng cho các mục đích sau:

- Tối ưu bán tự động vòng lặp theo ngữ cảnh
- Sinh mã trong thời gian thực thi hoặc chuyên biệt hóa thuật toán trong thời gian thực thi hoặc thời gian nạp
- Thay đổi các trạng thái được nhúng trong một đối tượng khác của các đối tượng
- Áp dụng trong các hệ thống tính toán tiến hóa như lập trình di truyền
- Ẩn mã để tránh các kỹ thuật dịch ngược
- Giới hạn tập lệnh thực thi trong các điều kiện thực tế để cải tiến hiệu năng
- Sửa đổi chương trình để tránh lỗi.

4.4. Tổng kết chương

Tối ưu trong giai đoạn thực thi tuy còn ít được nghiên cứu và khó triển khai nhưng có vai trò quan trọng do sự đa dạng về môi trường phần cứng của các thiết bị nhúng. Phần mềm nhúng thường chạy trên các CPU chuyên dụng nên việc cấu hình và tối ưu cũng khác nhau trong các môi trường thực thi. Hơn nữa do yêu cầu độc lập nền tảng của nhiều phần mềm nhúng và yêu cầu về hiệu năng nên việc áp dụng các kỹ thuật tối ưu trong giai đoạn thực thi là cần thiết. Đồng thời các kỹ thuật tối ưu trong giai đoạn này phần lớn là các kỹ thuật tối ưu động. Nghĩa là mã đích tối ưu tùy thuộc vào môi trường thực thi và ngữ cảnh cụ thể của chương trình.

Về tối ưu môi trường thực thi, chúng tôi đã tổng hợp ba kỹ thuật chính đó là kỹ thuật biên dịch tạm, lập lịch tiến trình và kỹ thuật chuyên biệt hóa. Kỹ thuật biên dịch tạm được tích hợp trong các trình thông dịch để biên dịch các đoạn mã và lưu trữ trong bộ đệm để thực thi mà không dịch và chạy từng câu lệnh. Kỹ thuật lập lịch tiến trình dựa trên mô hình Markov để lựa chọn tiến trình có mức ưu tiên cao nhằm cải tiến hiệu năng khi thực thi tiến trình này. Kỹ thuật chuyên biệt hóa thực hiện việc chuyên biệt hóa mã nguồn, dữ liệu và thuật toán theo ngữ cảnh thực thi cụ thể để cải tiến hiệu năng. Điểm quan trọng nhất theo hướng tiếp cận này là chúng tôi đã đề xuất và triển khai phương pháp tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU. Phương pháp đề xuất này đã được công bố trong công trình [CT14].

Về tối ưu môi trường truyền dữ liệu, chúng tôi tóm lược và trình bày về kỹ thuật loại dư thừa dữ liệu để rút ngắn thời gian truyền thông trong các phần mềm nhúng phân tán. *Về tối ưu mã thực thi*, chúng tôi cũng trình bày kỹ thuật tối ưu dựa trên mã tự sửa. Đây là kỹ thuật tối ưu mã thực thi quan trọng đối với phần mềm nhúng đảm bảo hai yêu cầu là tính linh động và cải tiến hiệu năng.

KẾT LUẬN

Trong phần này, chúng tôi sẽ 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. Theo cách tổ chức từ trên xuống, đầu tiên luận án cung cấp một mô hình chung về vấn đề tối ưu trong phát triển phần mềm nhúng. Trong mô hình chung, chúng tôi đã trình bày vấn đề tối ưu phần mềm nhúng theo các giai đoạn thiết kế, lập trình và thực thi cũng như phân tích kết quả và các phương pháp tiếp cận nghiên cứu tối ưu theo từng tiêu chí tối ưu trong mỗi giai đoạn. Dựa trên mô hình tối ưu chung, chúng tôi đã tiến hành nghiên cứu chi tiết, chuyên sâu về các phương pháp tối ưu phần mềm nhúng trong ba giai đoạn này theo tiêu chí hiệu năng, bộ nhớ và tối ưu đa mục tiêu. Các nội dung nghiên cứu trong luận án và các nghiên cứu liên quan đã được chúng tôi công bố trong 2 bài báo đăng trên tạp chí quốc tế, 2 bài báo đăng trên tạp chí uy tín trong nước, 10 bài báo đăng trong các Kỷ yếu của hội nghị quốc tế có phản biện được xuất bản bởi IEEE, IEICE, SPIE và 1 bài báo đăng trong tạp chí SCI [CT15]. Các 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 tiêu chí tối ưu hiệu năng, tối ưu bộ nhớ và tối ưu đa mục tiêu. Trong tiêu chí tối ưu hiệu năng, về mặt lý thuyết, chúng tôi đã đề xuất, phát triển một phương pháp mới là tối ưu hiệu năng dựa trên biểu đồ lớp và cải tiến phương pháp tối ưu hiệu năng dựa trên chuyển đổi mô hình. Về thực nghiệm, chúng tôi đã xây dựng khung làm việc DSL, T4 cho phép thiết kế mô hình dữ liệu như biểu đồ lớp và xây dựng các chương trình tối ưu tương ứng. Các kết quả nghiên cứu trong tiêu chí tối ưu này đã được xuất bản trong hai Kỷ yếu hội nghị khoa học quốc tế sau:

- **P.V. Huong**, N.N. Binh, "*Class Diagram Based Evaluation of Software Performance*", Proceedings of The 2012 International Conference on Information and Digital Engineering, SPIE, Vol. 8768, pp. 211-217, Singapore, October 2012.
- **P.V. Huong**, N.N. Binh, N.T. Huyen, N.T. Duong and T.N. Phu, "*Embedded Software Performance Optimization Based on Generating the Simulation Code of Functions*", Proceedings of IEICE ICDV, pp. 149-154, Hanoi, August 2012.

Trong tiêu chí tối ưu bộ nhớ, về mặt lý thuyết, chúng tôi đã đề xuất, phát triển một phương pháp mới là tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô và cải tiến phương pháp tối ưu bộ nhớ dựa trên chuyển đổi mô hình. Về thực nghiệm, chúng tôi đã xây dựng

khung làm việc DSL và T4 để thiết kế, sinh mã và xây dựng hai chương trình tối ưu tương ứng. Các kết quả nghiên cứu về tiêu chí tối ưu này bao gồm:

- **P.V. Huong**, N.N. Binh and P.N. Thanh, "*Optimizing occupied Memory of Embedded Software in the design phase*", Journal of Computer Science and Cybernetics (JCC) Hanoi, Vietnam, 2012.
- **P.V. Huong**, N.N. Binh, P.N. Thanh, "*Embedded Software Memory Optimization Based on the DSL and Topological Sort*", Proceedings of The 2012 International Conference on Software and Intelligent Information (ICSII), Singapore, October 2012.

Trong tiêu chí tối ưu đa mục tiêu, về nghiên cứu lý thuyết, chúng tôi đã đề xuất và phát triển một phương pháp mới là tối ưu đa mục tiêu dựa trên biểu đồ lớp. Về thực nghiệm, chúng tôi đã xây dựng chương trình tối ưu đa mục tiêu dựa trên biểu đồ lớp và phát triển khung làm việc DSL và T4 cho phép thiết kế và sinh mã. Các kết quả nghiên cứu liên quan đến vấn đề tối ưu đa mục tiêu dựa trên nguyên lý Pareto đã được công bố bao gồm:

- **P.V. Huong**, N.N. Binh and B.N. Hai, "*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, September 2013, India, pp. 1229-1236.
- **P.V. Huong**, N.N. Binh, "*An Approach to Design Embedded Systems by Multi-objective Optimization*", Proceedings of The 2012 International Conference on Advanced Technologies for Communication (IEEE ATC 2012), pp. 165-169, Hanoi, October 2012.
- **P.V. Huong**, N.N. Binh and B.N. Hai, "*A Pareto Optimal Configuration at design Phase for SoC Platform Based on the Genetic Algorithm*", Proceedings of IEICE ICDV, pp. 160-165, Hanoi, August 2011. (ISBN: 978-4-88522-258-1 C3055).

Ngoài ra, khác với phần mềm thông thường, phát triển phần mềm nhúng thường gắn liền với các hệ thống nhúng và vấn đề đồng thiết kế phần cứng – phần mềm cũng là một phương pháp tối ưu quan trọng. Phương pháp tối ưu này nhằm tìm ra một phân chia tốt nhất cho các tác vụ thành tác vụ phần cứng hay tác vụ phần mềm. Nghiên cứu về đồng thiết kế dựa trên DSL, T4 và nguyên lý Pareto đã được công bố trong công trình sau:

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

Mặt khác, 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:

- **P.V. Huong**, N.N. Binh, "*Design and Generating Code for Embedded Systems Based on DSL and T4*", Journal of Computer Science and Cybernetics (JCC) Hanoi, Vietnam, 2012.
- **P.V. Huong**, N.N. Binh, "*Embedded System Design and Code Generation by Using the DSL and T4*", Proceedings of 2012 International Conference on Electronics Engineering and Informatics (ICEEI), pp. 155-160, Phuket Thailan, September 2012. (ISBN: 978-981-07-3331-5, ISSN: 2010-460X).
- **P.V. Huong**, N.N. Binh, "*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, November 2012. (ISSN: 2010-3743).

Về vấn đề tối ưu phần mềm nhúng trong giai đoạn lập trình, chúng tôi tiếp cận nghiên cứu theo hai mức tối ưu đó là tối ưu mã nguồn mức cao độc lập kiến trúc đích và tối ưu mã hợp ngữ hướng đến các CPU cụ thể. Trong mức tối ưu mã nguồn mức cao, về mặt lý thuyết, chúng tôi đã đề xuất một cải tiến cho phương pháp tối ưu hiệu năng đó là thay thế các biểu thức tương đương. Ngoài ra, chúng tôi cũng triển khai phương pháp tối ưu hiệu năng phần mềm trên điện thoại di động trong môi trường phân tán dựa trên nén dữ liệu với một thử nghiệm quan trọng là phần mềm nhận dạng chữ Nôm. Kết quả nghiên cứu trong phần này đã được xuất bản trong Kỷ yếu của Hội nghị khoa học quốc tế sau:

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

Trong mức tối ưu mã 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. Về nghiên cứu lý thuyết, chúng tôi đã đề xuất, triển khai một phương pháp mới là tối ưu hiệu năng và điện năng tiêu thụ dựa trên lập lịch các lệnh theo thuật toán di truyền. Phương pháp đề xuất này áp dụng cho cả kiến trúc đường ống lệnh và kiến trúc siêu vô hướng. Đồng thời, một phần nội dung nghiên cứu trong phần này đã được công bố trong công trình sau:

- **P.V. Huong**, B.N. Hai and N.N. Binh, "*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, 2014.

Các nghiên cứu liên quan về tối ưu trong giai đoạn này cũng được công bố trong tạp chí **SCI** sau:

- N.N. Binh, **P.V. Huong** and B.N. Hai, "A new approach to embedded software optimization based on reverse engineering", *IEICE Trans. INF. & SYST*, Vol.E98-D, No.6, 2015.

Về vấn đề tối ưu phần mềm nhúng trong giai đoạn thực thi, chúng tôi cũng đề xuất và triển khai phương pháp tối ưu điện năng tiêu thụ dựa trên kỹ nghệ ngược và tái cấu hình CPU. Phương pháp này kết hợp cả phần cứng và phần mềm, đã đạt được kết quả tốt, có triển vọng áp dụng cho các hệ thống nhúng và phần mềm nhúng đã có. Kết quả nghiên cứu đã được công bố trong công trình sau:

- **P.V. Huong**, N.N. Binh and V.V. Phuc, "*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, 2014.

Bên cạnh những đóng góp và kết quả nghiên cứu khả quan, luận án vẫn còn một số vấn đề chưa giải quyết trong các giai đoạn tối ưu. Trong giai đoạn thiết kế, các nội dung nghiên cứu trong luận án mới chỉ tập trung vào một số mô hình 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. Trong giai đoạn lập trình vấn đề phân tích, đánh giá để tìm các đoạn mã tiêu tốn 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 và kiến trúc RISC mà chưa giải quyết bài toán tối ưu trong các hệ thống đa CPU cũng như chưa xét kiến trúc CISC. Trong giai đoạn thực thi, vấn đề phân lớp các bài toán để áp dụng kỹ thuật biên dịch tạm chưa được giải quyết.

Trên cơ sở các vấn đề chưa giải quyết và các hướng nghiên cứu mở trong tối ưu phần mềm nhúng, chúng tôi sẽ tiếp tục tiến hành các nghiên cứu sâu hơn theo các định hướng sau. Trong giai đoạn thiết kế, chúng tôi sẽ tiếp cận theo SPE để nghiên cứu các phương pháp tối ưu dựa trên các mô hình động cũng như việc hình thức hóa, bổ sung các ràng buộc thời gian thực và các tính chất đặc thù của hệ thống nhúng vào mô hình. Mặt khác, chúng tôi cũng tổng hợp, nghiên cứu và tích hợp các độ đo chất lượng phần mềm khác vào phương pháp tối ưu đa mục tiêu. Trong giai đoạn lập trình, chúng tôi sẽ dựa trên cơ sở lập lịch đa CPU cho các tiến trình của hệ điều hành để giải quyết bài toán tối ưu trong môi trường đa CPU. 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. Trong giai đoạn thực thi, chúng tôi sẽ nghiên cứu phương pháp phân lớp cho các chương trình trong kỹ thuật JIT và vấn đề tối ưu mã máy ảo. Ngoài ra, chúng tôi cũng hướng đến áp dụng kỹ nghệ ngược cho phần cứng để kiểm chứng và tối ưu mức kiến trúc CPU hoặc lựa chọn cấu hình tối ưu.

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 (in IEEE Xplore, 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, in IEEE Xplore), 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**).

TÀI LIỆU THAM KHẢO

Tiếng Việt

[1] Nguyễn Ngọc Bình (2005), "*Một số ý kiến về phát triển, ứng dụng công nghệ thông tin của Việt Nam*", Hội thảo Quốc gia về "Hội nhập Quốc tế về Khoa học và Công nghệ", Bộ Khoa học và Công nghệ, tr. 32-40.

Tiếng Anh

[2] Abdelzaher, T. F. and Shin, K. G. (1995), "Optimal combined task and message scheduling in distributed real-time systems", *Proceedings of the 16th IEEE Real-Time Systems Symposium*, IEEE Computer Society, Washington, DC, USA, pp. 162-170.

[3] Aho, A. V., Lam, M. S., Sethi, R. and Ullman, J. D. (2007), "Compilers: Principles, Techniques, & Tools" , *Pearson*, ISBN 978-81-317-2101-8.

[4] Alsaadi, A. (2006), "Applying the UML class diagram in the performance analysis", *Proceedings of the Third European conference on Formal Methods and Stochastic Models for Performance Evaluation*, Springer-Verlag, Berlin, Heidelberg, pp. 148-165.

[5] Anastasopoulos, M. and Muthig, D. (2007), "Optimizing Model-driven Development by deriving Code Generation Patterns from Product line architectures", *D-67661 Kaiserslautern*, Germany, pp. 50-61.

[6] Andolfi, F., Aquilani, F., Balsamo, S. and Inverardi, P. (2000), "Deriving performance models of software architectures from message sequence charts", *Proceedings of the 2nd international workshop on Software and performance*, ACM, New York, NY, USA, pp. 47-57.

[7] Anne, K., C. B. and Temmerman, M. (2009), *Abstract and Concrete Data Type Optimizations at the UML and C/C++ Level for Dynamic Embedded Software*, IGI Global.

[8] Anne, K., P. G. and Janssens, D. (2008), "Transformation Language Integration Based on Profiles and Higher Order Transformations", *First International Conference on Software Language Engineering* (5), pp. 208-226.

[9] Ayala, K. (1996), "8051 Microcontroller: Architecture, Programming and Applications", *Delmar Cengage Learning*, 2nd Edition, p. 350.

[10] Balsamo, S., Inverardi, P. and Mangano, C. (1998), "An approach to performance evaluation of software architectures", *Proceedings of the 1st international workshop on Software and performance*, ACM, New York, NY, USA, pp. 178-190.

[11] Balsamo, S., Marco, A. D., Inverardi, P. and Simeoni, M. (2004), "Model-Based Performance Prediction in Software Development: A Survey," *IEEE Transactions on Software Engineering* (30:5), pp. 295-310.

[12] Balsamo, S. and Marzolla, M. (2005), "Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models", *Proceedings of the 5th International Workshop on Software and Performance*, ACM, New York, NY, USA, 2005, pp. 37-42.

[13] Balsamo, S. and Mirandola, R. (2006), "Efficient Performance Models in Component-Based Software Engineering", *EUROMICRO-SEAA, IEEE*, 2006, pp. 64-71.

[14] Barr, M. and Massa, A. (2006), *Programming Embedded Systems: With C and GNU Development Tools*, O'Reilly Media, Inc.

[15] Baskiyar, S. and Meghanathan, N. (2005), "A Survey of Contemporary Real-time Operating Systems", *Academic Journal, Vol. 29 Issue 2*, pp. 233-240.

[16] Bernardo, M., Ciancarini, P. and Donatiello, L. (2000), "A process algebraic description language for the performance analysis of software architectures", *Proceedings of the 2nd*

international workshop on Software and performance, ACM, New York, NY, USA, pp. 1-11.

[17] Binh, N. N., Imai, M., Shiomi, A. and Hikichi, N. (1995), "A hardware/software codesign method for pipelined instruction set processor using adaptive database.", *Shirakawa, I., ed., ASP-DAC, ACM*, pp. 1-10.

[18] Cheung, T. L., Okamoto, K., Maker, F., Liu, X. and Akella, V. (2009), "Markov decision process (MDP) framework for optimizing software on mobile phones", *Proceedings of the seventh ACM international conference on Embedded software*, ACM, New York, NY, USA, pp. 11-20.

[19] Cho, Y. Y., Moon, J. B. and Kim, Y. C. (2004), "A system for Performance Evaluation of Embedded Software", *Transactions on Engineering, Computing and Technology V1, ISSN*, pp. 1305-5313.

[20] Corporation, N. (2003), "Optimizing the Client/Server Communication for Mobile Applications, Part 1, 2 and 3", *Instituto Superior de Engenharia do Porto*.

[21] Crespo, A., Harbour, M. B. (2008), "Operating System Support for Embedded Real-Time Applications", *EURASIP J. Emb. Sys*, pp. 35-48.

[22] Creus, G. and Kuulusa, M. (2007), "Optimizing Mobile Software with Built-in Power Profiling", *Mobile Phone Programming*, Springer Netherlands, pp. 449-462.

[23] Dave, B. P., Lakshminarayana, G. and Jha, N. K. (1999), "Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Trans. VLSI Syst*, pp. 92-104.

[24] Dean, J., Grove, D. and Chambers, C. (1995), "Optimization of Object-Oriented Programs using Static Class Hierarchy Analysis", *Springer-Verlag*, pp. 77-101.

[25] De Oliveira Castro, P., Louise, S. and Barthou, D. (2010), "Reducing memory requirements of stream programs by graph transformations", *High Performance Computing and Simulation (HPCS), 2010 International Conference*, pp. 171-180.

[26] Diwan, A., McKinley, K. S. and Moss, J. E. B. (2001), "Using types to analyze and optimize object-oriented programs," *ACM Trans. Program. Lang. Syst.*, pp. 30-72.

[27] Erbas, C., Cerav-Erbas, S. and Pimentel, A. D. (2006), "Multiobjective optimization and evolutionary algorithms for the application mapping problem in multiprocessor system-on-chip design," *Trans. Evol. Comp* (10:3), pp. 358-374.

[28] Fog, A. (2012), *The microarchitecture of Intel, AMD and VIA CPUs: An optimization guide for assembly programmers and compiler makers*, Technical report, Technical University of Denmark.

[29] Fog, A. (2011), *Optimizing software in C++: An optimization guide for Windows, Linux and Mac platforms*, Technical report, Technical University of Denmark.

[30] Fog, A. (2009), *Optimizing subroutines in assembly language: An optimization guide for x86 platforms*, Technical report, Publishers of British Archaeological Reports, 2400 Copenhagen NV, Denmark.

[31] Frederic Ogel, B. F. and Piumarta, L. (2003), "On Reflexive and Dynamically Adaptable Environments for Distributed Computing", *ICDCSW '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, IEEE Computer Society, Providence, Rhode Island, pp. 112-120.

[32] Gal, A., Fröhlich, P. H. and Franz, M. (2002), *An Efficient Execution Model for Dynamically Reconfigurable Component Software*, Technical report.

[33] Galuzzi, C. and Bertels, K. (2011), "The Instruction-Set Extension Problem: A Survey," *ACM Trans. Reconfigurable Technol. Syst*, pp. 1-28.

[34] Genero, M., Olivas, J., Piattini, M. and Romero, F. (2001), "Using Metrics to Predict OO Information Systems Maintainability", *Proceedings of the 13th International Conference on Advanced Information Systems Engineering*, Springer-Verlag, London, UK, UK, pp. 388-401.

[35] Genero, M., Piattini, M. and Calero, C. (2002), "Empirical Validation of Class Diagram Metrics", *Proceedings of the 2002 International Symposium on Empirical Software Engineering*, IEEE Computer Society, Washington, DC, USA, pp. 195-215.

[36] Genero, M., Piattini, M. and Caleron, C. (2005), "A survey of metrics for UML class diagrams," *Journal of Object Technology*, pp. 59-92.

[37] Gomaa, H. and Menasce, D. A. (2000), "Design and performance modeling of component interconnection patterns for distributed software architectures", *Proceedings of the 2nd international workshop on Software and performance*, ACM, New York, NY, USA, pp. 117-126.

[38] Goss, C. F. (2013), "Machine Code Optimization - Improving Executable Object Code", *CoRR (abs/1308.4815)*, pp. 1-10.

[39] Greg, A., Drew, D., T. K. and Debray, S. (2008), *Software Optimization at Link-time And Run-time*, Technical report, Gregg Townsend, Rick Schlichting, Matti Hiltunen.

[40] Gregory, A., Saumya, D., B. S. and Legendre, M. (2001), "Using Link-Time Optimization to Improve the Performance of MPI Programs", *Preliminary results*, pp. 23-30.

[41] Hammond, K. and Michaelson, G. (2003), "A domain-specific language for real-time embedded systems", *Proceedings of the 2nd international conference on Generative programming and component engineering*, Springer-Verlag New York, Inc., New York, NY, USA, pp. 37-56.

[42] Harman, M. and Tratt, L. (2007), "Pareto optimal search based refactoring at the design level.", *GECCO*, ACM, pp. 1106-1113.

[43] Hartigan, J. A. and Wong, M. A. (1979), "Algorithm AS 136: A K-Means Clustering Algorithm", *Journal of the Royal Statistical Society*, pp. 100–108.

[44] Hsieh, P. (2007), *Programming Optimization*, Technical report, Redhat Corp.

[45] Islam, S. and Suri, N. (2007), "A multi variable optimization approach for the design of integrated dependable real-time embedded systems", *Proceedings of the 2007 international conference on Embedded and ubiquitous computing*, Springer-Verlag, Berlin, Heidelberg, pp. 517-530.

[46] Izosimov, V., Pop, P., Eles, P. and Peng, Z. (2005), "Design Optimization of Time and Cost-Constrained Fault-Tolerant Distributed Embedded Systems", *Design Automation and Test in Europe Conference*, pp. 864-869.

[47] Jeon, S. U. (2008), "Introduction to the Design of Embedded System", *SE Lab, Korea Advanced Institute of Science & Technology*, pp. 1-35.

[48] Jones, M. J. (2005), "Optimization in GCC", *Linux journal* (3), pp. 1-15.

[49] Jung, H. W. and Choi, B. (1999), "Optimization models for quality and cost of modular software systems", *European Journal of Operational Research*, pp. 613 - 619.

[50] Kane, G. and Heinrich, J. (1991), "MIPS RISC Architecture", *Prentice Hall*, 2nd Edition, p. 544.

[51] Kaya, K. and Uçar, B. (2009), "Exact Algorithms for a Task Assignment Problem", *Parallel Processing Letters*, pp. 451-465.

[52] Kim, S. and Barry, M. (2003), "Uml-based object-oriented metrics for architecture complexity analysis", *Future Gener. Comput. Sys.*, pp. 125-134.

[53] Koziolek, A. and Reussner, R. (2011), "Towards a generic quality optimisation framework for component-based system models", *Proceedings of the 14th international ACM Sigsoft symposium on Component based software engineering*, ACM, New York, NY, USA, pp. 103-108.

[54] Krasner, J. (2005), *Optimizing tecnology choises for device software*, Technical report, Nokia forum.

[55] Kubásek, M. (2006), *Caching Optimization in Service Oriented Architecture*, Technical report, Nokia forum.

[56] Léger, A., Folliot, B. and Cailliau, D. (2001), "Platform for Software Reconfiguration in Embedded Systems", *European Research Seminar on Advances in Distributed Systems, Bertinoro*, Italy, pp. 1-10.

[57] Lawall, J. and Muller, G. (1999), *Faster Run-time Specialized Code Using Data Specialization*, Technical report.

[58] Lee, M. T., Tiwari, V., Malik, S., Fujita, M. (1997), "Power Analysis and Minimization Techniques for Embedded DSP Software." *IEEE Trans. on VLSI Systems*, pp. 123-135.

[59] Lee, W., You, K. and Sung, W. (2002), "Software optimization of MPEG audio layer-III for a 32 bit RISC processor", *APCCAS (1)*, pp. 435-438.

[60] Leiserson, C. E., Rivest, R. L., S. (2001), "Section 22.4: Topological sort", Introduction to Algorithms (2nd ed.), *MIT Press and McGraw-Hill*, pp. 549-552.

[61] Luis, F. (2009), "A Survey of Operating Systems Infrastructure for Embedded Systems", *Department of Informatics, University of Federal de Santa Catarina, Brazil*, pp. 1-14.

[62] Manju, A., Sudhanshu, A., V. K. S. (2010), "Optimal redundancy allocation in complex systems", *Journal of Quality in Maintenance Engineering* (16), pp. 413-424.

[63] Manso, M. E., Genero, M. and Piattini, M. (2003), "No-redundant metrics for UML class diagram structural complexity", *Proceedings of the 15th international conference on Advanced information systems engineering*, Springer-Verlag, Berlin, Heidelberg, pp. 127-142.

[64] Marchesi, M. (1998), "OOA Metrics for the Unified Modeling Language", *Proceedings of the 2nd Euromicro Conference on Software Maintenance and Reengineering (CSMR'98)*, IEEE Computer Society, Washington, DC, USA, pp. 67-80.

[65] Martens, A., Koziolek, H., Becker, S. and Reussner, R. (2010), "Automatically Improve Software Architecture Models for Performance, Reliability, and Cost Using Evolutionary Algorithms", *Proceedings of the First Joint WOSP/SIPEW International Conference on Performance Engineering*, ACM, New York, NY, USA, pp. 105-116.

[66] Martonosi, M., Brooks D. and Bose P. (2001), "*Modeling and analyzing CPU power and performance: metrics, methods, and abstraction*", Tutorial at SIGMETRICS, pp. 1-27.

[67] Masuhara, H. and Yonezawa, A. (2001), "Run-Time Bytecode Specialization", *Proceedings of the Second Symposium on Programs as Data Objects*, Springer-Verlag, London, UK, UK, pp. 138-154.

[68] Masuhara, H. and Yonezawa, A. (1999), *Generating Optimized Residual Code in Run-Time Specialization*, Technical report.

[69] Matthew, R. A., Stephen, F., D. P. G. M. H. and Sweeney, P. F. (2005), "A Survey of Adaptive Optimization in Virtual Machines", *Proceedings of the IEEE, 92*, pp. 449-466.

[70] Menasce, D. A. and Gomaa, H. (1998), "On a language based method for software performance engineering of client/server systems", *Proceedings of the 1st international workshop on Software and performance*, ACM, New York, NY, USA, pp. 63-69.

[71] Mohan, R., Saumya, D., M. H. and Schlichting, R. (2002), *System Call Clustering: A Profile-Directed Optimization Technique*, Technical report, ACR-9720738 and CCR-0113633.

[72] Murphy, G. C. and Saenko, E. (2008), "Predicting memory use from a class diagram using dynamic information", *Proceedings of the 1st international workshop on Software and performance*, ACM, New York, NY, USA, pp. 145-151.

[73] Novillo, D. (2004), *From Source to Binary: The Inner Workings of GCC*, Technical report, Redhat Corp.

[74] Oh, S., Aktas, M. S., Pierce, M. and Fox, G. C. (2006), "Optimizing web service messaging performance using a context store for static data", *Proceedings of the 5th WSEAS international conference on Telecommunications and informatics*, Stevens Point, Wisconsin, USA, pp. 334-342.

[75] Oh, S. and Fox, G. C. (2007), "Optimizing Web Service messaging performance in

mobile computing," *Future Gener. Comput. Syst.* (23:4), pp. 623-632.
[76] Org, G. (2001), *Options That Control Optimization*, Technical report, Free Software Foundation.
[77] Ozgur, T. (2009), *Domain-Specific Modeling: A Practical Approach A comparison of Microsoft DSL Tools and Eclipse Modeling Frameworks in the context of Model-Driven Development*, LAP Lambert Academic Publishing, Germany, 2009.
[78] Panda, P. R., Catthoor, F., Dutt, N. D., Danckaert, K., Brockmeyer, E., Kulkarni, C., Vandercappelle, A. and Kjeldsberg, P. G. (2001), "Data and memory optimization techniques for embedded systems," *ACM Trans. Des. Autom. Electron. Syst.*, pp. 149-206.
[79] Petriu, D. and Woodside, M. (2004), "A Metamodel for Generating Performance Models from UML Designs", *The Unified Modeling Language. Modelling Languages and Applications*, Springer Berlin Heidelberg, pp. 41-53.
[80] Petriu, D. C. and Woodside, C. M. (2003), "Performance analysis with UML: layered queueing models from the performance profile", *Kluwer Academic Publishers, Norwell, MA*, USA, pp. 221-240.
[81] Peymandoust, A., Micheli, G. D. and Simunic, T. (2002), "Complex library mapping for embedded software using symbolic algebra", *DAC*, ACM, pp. 325-330.
[82] Peymandoust, A., Simunic, T. and Micheli, G. D. (2002), "Low Power Embedded Software Optimization Using Symbolic Algebra", *DATE, IEEE Computer Society*, pp. 1052-1058.
[83] Piumarta, I. (2004), *Dynamic code generation for C and C++*, Technical report, LIP6/CNRS and INRIA.
[84] Poole, J. D. (2001), "Model-Driven Architecture: Vision, Standards and Emerging Technologies", *In ECOOP 2001, Workshop on Metamodeling and Adaptive Object Models*, pp. 135-144.
[85] Pospiech, F. and Olsen, S. (2003), "Embedded software in the SoC world. How HdS helps to face the HW and SW design challenge", *Custom Integrated Circuits Conference, 2003 - Proceedings of the IEEE 2003* , pp. 653-658.
[86] Redkin, P.P. (2010), "Microcontrollers Atmel AVR32 architecture family AT32UC3", *Tekhnosfera*, p.784.
[87] Risco-Martin, J. L., Mittal, S., Atienza, D., Hidalgo, J. I. and Lanchares, J. (2008), "Optimization of dynamic data types in embedded systems using DEVS/SOA-based modeling and simulation", *Proceedings of the 3rd international conference on Scalable information systems*, ICST, Belgium, Belgium, pp. 1-11.
[88] Robert, M., S. W. and Debray, S. (2000), "Code Specialization Based on Value Profiles", *Proc. 7th. International Static Analysis Symposium (SAS 2000)*, Springer LNCS, pp. 340-359.
[89] Saumya, D., W. E. and Muth, R. (2000), "Compiler Techniques for Code Compression," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, pp. 378-415.
[90] Schwarz, B. (2002), *Post Link-Time Optimization on the Intel IA-32 Architecture*, Technical report.
[91] Shankar, A., Sastry, S. S., Bodk, R. and Smith, J. E. (2005), "Runtime Specialization with Optimistic Heap Analysis", *Proceedings of the 20th Annual ACM SIGPLAN Conference*, ACM, New York, NY, USA, pp. 327-343.
[92] Shin, Y., Myers, S. and Gupta, M. (2010), "Saving Energy on WiFi with Required IPsec", *in Jajodia, S. and Zhou, J., ed.,'Security and Privacy in Communication Networks'*, Springer Berlin Heidelberg, pp. 144-161.
[93] Seal, D. (2001), "ARM Architecture Reference Manual", *Addison Wesley*, 2nd Edition, ISBN-13: 078-5342737196.
[94] Sifakis, J. (2011), "A vision for computer science – the system perspective", *Central*

European Journal of Computer Science, Vol.1, No.1, pp.108-116.

[95] Sivonen, S. (2008), *Domain-specific modelling language and code generator for developing repository-based Eclipse plug-ins*, Technical report.

[96] Smith, C. U., Lladó, C. M., Cortellessa, V., Marco, A. D. and Williams, L. G. (2005), "From UML models to software performance results: an SPE process based on XML interchange formats", *Proceedings of the 5th international workshop on Software and performance*, ACM, New York, NY, USA, pp. 87-98.

[97] Somu, P., Haifeng, H. M. and Debray, S. (2006), "Profile-Guided Specialization of an Operating System Kernel", *Proceedings of Binary Instrumentation and Applications*, pp. 1-10.

[98] Srinivasan, S., Fang, Z., Iyer, R., Zhang, S., Espig, M., Newell, D., Cermak, D., Wu, Y., Kozintsev, I. and Haussecker, H. (2009), "Performance characterization and optimization of mobile augmented reality on handheld platforms", *Proceedings of the 2009 IEEE IISWC*, IEEE Computer Society, Washington, DC, USA, pp. 128-137.

[99] Suhendra, V. (2009), *Memory Optimizations for Time-predictable Embedded Software*, Technical report.

[100] Temmerman, M., Daylight, E. G., Catthoor, F., Demeyer, S. and Dhaene, T. (2007), "Optimizing Data Structures at the Modeling Level in Embedded Multimedia," *J. Syst. Archit.*, pp. 539-549.

[101] Thepayasuwan, N. and Doboli, A. (2004), "Hardware-Software Co-Design of Resource Constrained Systems on a Chip.", *ICDCS Workshops*, IEEE Computer Society, pp. 818-823.

[102] Thompson, C., White, J., Dougherty, B. and Schmidt, D. (2009), "Optimizing Mobile Application Performance with Model–Driven Engineering", *in Lee, S. and Narasimhan, P., ed.,'Software Technologies for Embedded and Ubiquitous Systems'*, Springer Berlin Heidelberg, pp. 36-46.

[103] Tong, Q., Zou, X., Tong, H., Gao, F. and Zhang, Q. (2008), "Hardware/Software Partitioning in Embedded System Based on Novel United Evolutionary Algorithm Scheme", *Proceedings of the 2008 International Conference on Computer and Electrical Engineering*, IEEE Computer Society, Washington, DC, USA, pp. 141-144.

[104] Toshio, S., Toshiaki, Y., M. K. H. K. and Nakatani, T. (2001), "A dynamic optimization framework for a Java just-in-time compiler", *Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA 01)*, pp. 180-195.

[105] Tourwé, T. and Meuter, W. (1999), "Optimizing Object-Oriented Languages through Architectural Transformations", *in Jähnichen, S., ed.,'Compiler Construction*', Springer Berlin Heidelberg, pp. 244-258.

[106] Tran, T. and Wietfeld, C. (2009), "Approaches for optimizing the performance of a mobile SAML-based emergency response system", *EDOCW 13th*, pp. 148-156.

[107] Turley, J (1999), "Embedded Processors by the Numbers", *Embedded Systems Programming*, Vol. 12, No. 5, pp. 13-14.

[108] Van, D., A., Klint, P. and Visser, J. (2000), "Domain-specific languages: an annotated bibliography," *SIGPLAN Not.* , pp. 26-36.

[109] Vasanth, B., E. D. and Banerjia, S. (2000), "Dynamo: A Transparent Dynamic Optimization System", *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*', pp. 1-12.

[110] Williams, L. G. and Smith, C. U. (1998), "Performance evaluation of software architectures", *Proceedings of the 1st international workshop on Software and performance*, ACM, New York, NY, USA, pp. 164-177.

[111] Wu, Y. (2000), "Architectural Level Power Estimation And Experimentation", *Doctoral thesis of Computer Science and Engineering*, The Pennsyvania State University, pp. 9-11.

[112] Xu, J., Woodside, M. and Petriu, D. (2003), "Performance Analysis of a Software Design Using the UML Profile for Schedulability, Performance and Time", *in Proc. 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, Springer-Verlag, pp. 291-310.

[113] Yang, P. (2004), *Pareto optimization based run-time task scheduling for embedded systems*, Technical report.

[114] Yang, Z. (2009), *A domain-specific modeling approach for component-based software development*, Ball State University, Muncie, IN, USA.

[115] Yi, T. and Wu, F. (2004), "Empirical analysis of entropy distance metric for UML class diagrams," *SIGSOFT Softw. Eng. Notes*, pp. 1-6.

[116] Yi, T., Wu, F. and Gan, C. (2004), "A comparison of metrics for UML class diagrams," *SIGSOFT Softw. Eng. Notes*, pp. 1-6.

[117] You, K., kyu Choi, Y., Choi, J. and Sung, W. (2011), "Memory Access Optimized VLSI for 5000-Word Continuous Speech Recognition," *Signal Processing Systems* (63:1), pp. 95-105.

Phụ lục. TỔNG HỢP CÁC CHƯƠNG TRÌNH THỰC NGHIỆM

Các chương trong luận án đã trình bày về các phương pháp tối ưu mới cũng như cải tiến một số phương pháp tối ưu đã có theo ba giai đoạn chính trong vòng đời phần mềm. Các phương pháp tối ưu đề xuất trong luận án đã được kiểm chứng bằng thực nghiệm. Trong mỗi phương pháp tối ưu, chúng tôi xây dựng chương trình tối ưu và đánh giá kết quả dựa trên việc thực thi một số ví dụ thực tế. Theo đó, phụ lục này nhằm hệ thống toàn bộ các chương trình thực nghiệm và các chương trình ví dụ được sử dụng để thực nghiệm trong các chương trước. Đầu tiên, chúng tôi tổng hợp về các khung làm việc DSL và T4. Đây là bộ công cụ cho phép thiết kế các mô hình phần mềm theo DSL đã định nghĩa và lấy các thông tin đặc tả tự động từ mô hình dựa trên T4. Khung làm việc này được sử dụng để thiết kế các mô hình phần mềm gốc. Thông tin đặc tả mô hình gốc được sử dụng làm đầu vào cho các chương trình tối ưu. Tiếp theo chúng tôi sẽ trình bày về các chương trình tối ưu đã được xây dựng và sử dụng trong các thực nghiệm. Để tiến hành thực nghiệm trong mỗi phương pháp tối ưu cần có các chương trình thử nghiệm. Thực thi các chương trình thử nghiệm này trong các môi trường cụ thể để đánh giá kết quả tối ưu. Phụ lục gồm các phần chính sau: *P.1* tổng hợp các công cụ DSL, T4 và các chương trình tối ưu; *P.2* trình bày về các chương trình thử nghiệm.

P.1. Các chương trình và công cụ tối ưu

Để triển khai thực nghiệm trong các phương pháp tối ưu chúng tôi đã phát triển các công cụ hỗ trợ thiết kế mô hình và sinh đặc tả tự động từ mô hình cũng như xây dựng các chương trình tối ưu. Các công cụ thiết kế cho phép xây dựng các mô hình phần mềm ban đầu và sinh thông tin đặc tả dạng văn bản từ mô hình. Mục đích của việc chuyển mô hình sang đặc tả dạng văn bản nhằm lấy các tham số cần thiết sử dụng trong quá trình tối ưu. Thông tin đặc tả dạng văn bản được đưa vào các chương trình tối ưu để tiến hành thử nghiệm các phương pháp tối ưu. Chúng tôi đã xây dựng các chương trình tối ưu cụ thể trong các phương pháp tối ưu. Nội dung phần này sẽ tổng hợp các công cụ và chương trình tối ưu đã được xây dựng.

P.1.1. Khung làm việc DSL và T4

Khung làm việc DSL và T4 là bộ công cụ cho phép thiết kế các mô hình phần mềm cũng như tự động chuyển mô hình sang đặc tả dạng văn bản. Trong luận án, chúng

tôi đã xây dựng một số khung làm việc DSL và T4 để thiết kế các mô hình phần mềm khác nhau như biểu đồ lớp đơn giản, đồ thị phụ thuộc tác vụ. Để xây dựng một khung làm việc DSL và T4, trước tiên chúng tôi định nghĩa một DSL cho một miền ứng dụng cụ thể. Một số ví dụ DSL cụ thể như đặc tả cấu trúc dữ liệu trừu tượng như biểu đồ lớp đơn giản, ngôn ngữ SQL hay DSL đặc tả kiến trúc hệ thống nhúng, v.v. Sau khi định nghĩa DSL, chúng tôi xây dựng siêu mô hình biểu diễn cú pháp và ngữ nghĩa của DSL. Sau đó, siêu mô hình được biên dịch và đóng gói thành một công cụ cho phép thiết kế phần mềm, hệ thống theo DSL đã định nghĩa. Để thực hiện việc chuyển đổi tự động từ mô hình sang đặc tả dạng văn bản cũng như lấy các tham số từ mô hình, chúng tôi xây dựng các mẫu T4 để tích hợp vào bộ công cụ này.

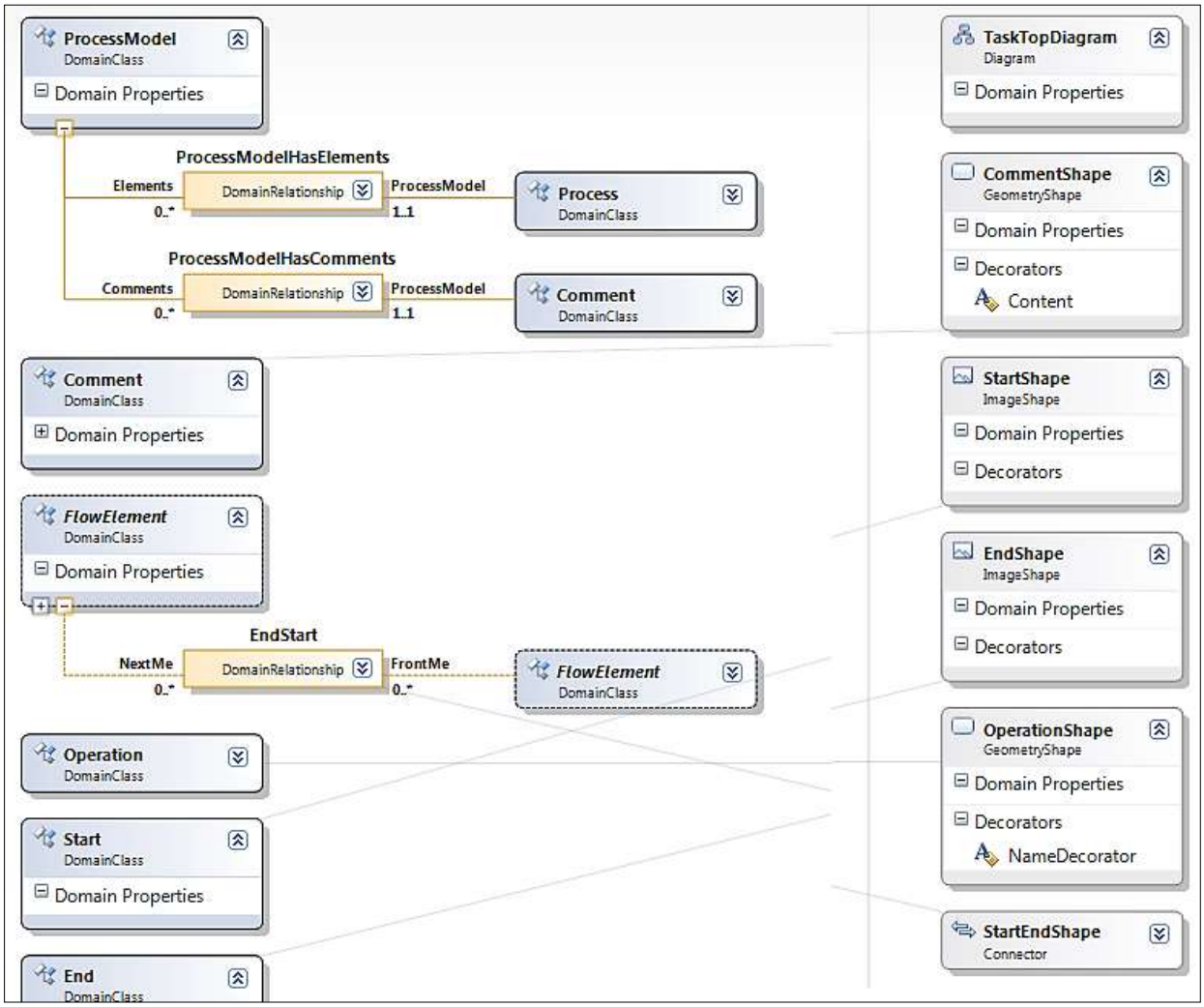
Để xây dựng khung làm việc DSL và T4, chúng tôi sử dụng phần mềm Visual Studio .NET 2010. Phần mềm này cho phép xây dựng các siêu mô hình, thiết kế giao diện công cụ và các ký hiệu trong giao diện thiết kế cũng như biên dịch và đóng gói công cụ. Trong quá trình nghiên cứu và thực nghiệm, dựa trên công cụ này chúng tôi đã xây dựng các khung làm việc DSL và T4 như khung làm việc để thiết kế mô hình dữ liệu trừu tượng, khung làm việc để xây dựng đồ thị tác vụ phụ thuộc và khung làm việc cho phép thiết kế hệ thống nhúng.

i. Khung làm việc DSL và T4 để xây dựng đồ thị tác vụ phụ thuộc

Khung làm việc này được phát triển để hỗ trợ triển khai phương pháp tối ưu bộ nhớ dựa trên sắp xếp tô-pô đã đề cập trong Chương 2. Để xây dựng khung làm việc này, đầu tiên chúng tôi định nghĩa một DSL cho phép xây dựng đồ thị phụ thuộc trong giao diện đồ họa. Đồ thị phụ thuộc gồm một tập các tác vụ được thực hiện trong hàm chính và quan hệ giữa các tác vụ. Mỗi tác vụ sẽ được lập trình thành một hàm khi xây dựng chương trình và được mô tả trong đồ thị như một nguyên mẫu hàm với tên, tham số và kiểu trả về. Sau đó, chúng tôi sử dụng Visual Studio .NET để xây dựng siêu mô hình đặc tả ngữ nghĩa cho DSL. Bản chất siêu mô hình là một tệp tin XML lưu trữ định nghĩa các lớp lôgic, định nghĩa các lớp ký hiệu đồ họa và các ánh xạ giữa các lớp lôgic và lớp ký hiệu như minh họa trong Hình P.2. Siêu mô hình và các thiết lập xây dựng công cụ được biên dịch và đóng gói thành khung làm việc cho DSL đã định nghĩa. Quá trình định nghĩa DSL và xây dựng siêu mô hình như các bước sau:

- Định nghĩa các thành phần lôgic: lớp Process, lớp Task, lớp Comment, lớp Start, lớp End, quan hệ EndStart và các ràng buộc
- Tạo các ký hiệu hình học tương ứng với mỗi thành phần lôgic ở trên. Các ký hiệu này sẽ sử dụng để thiết kế trong giao diện đồ họa khi DSL được triển khai

- Xây dựng siêu mô hình để lưu trữ các định nghĩa và ánh xạ giữa lớp lôgic và ký hiệu hình học như trong Hình P.1, trong đó phía bên trái biểu diễn các lớp lôgic mô tả ngữ nghĩa của siêu mô hình và phía bên phải là các lớp mô tả các ký hiệu đồ họa trong hộp công cụ sau khi khung làm việc DSL được xây dựng.



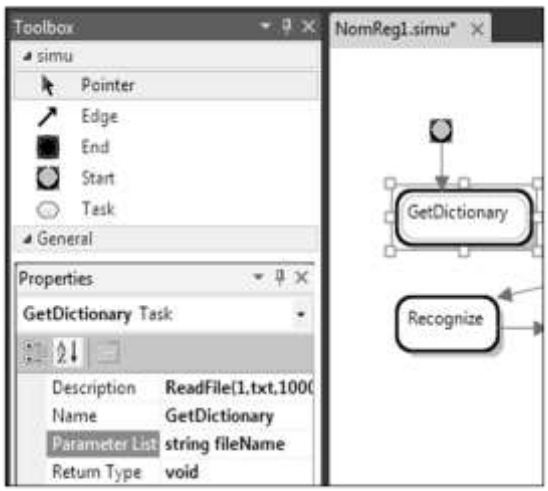
Hình P.1: Một phần siêu mô hình cho DSL thiết kế đồ thị tác vụ phụ thuộc

Trong Hình P.2, văn phạm mô tả mô hình thiết kế bằng DSL cũng là một tệp tin XML chứa thông tin mô tả các ký hiệu hình học trong mô hình thực tế. Quá trình sinh mã là quá trình phân tích và đối sánh văn phạm XML của mô hình DSL thực tế với văn phạm XML của siêu mô hình, gồm các bước sau: phân tích tệp tin XML của DSL để lấy các mô tả của các lớp ký hiệu thực tế; phân tích tệp tin XML của siêu mô hình để lấy mô tả về các lớp ký hiệu, các lớp ngữ nghĩa và ánh xạ giữa chúng; đối sánh thông tin để lấy thông tin ngữ nghĩa của mô hình thực tế.

```
DslDefinition.dsl*
  <Relationships>...</Relationships>
  <Types>...</Types>
  <Shapes>
    <GeometryShape Id="7076fb59-a90a-4daf-8a98-bad1e7a5e66d" Description="Description for dp.TaskTop.CommentShape"
      <ShapeHasDecorators Position="InnerTopLeft" HorizontalOffset="0" VerticalOffset="0">
        <TextDecorator Name="Content" DisplayName="Content" DefaultText="Content" />
      </ShapeHasDecorators>
    </GeometryShape>
    <ImageShape Id="ed91f70e-d407-4ecc-8948-01946526a110" Description="Description for drpham.TaskTop.StartShape"
    <ImageShape Id="1990526f-90bf-4ef6-813f-3cae3f97ec3d" Description="Description for drpham.TaskTop.EndShape" Nam
    <GeometryShape Id="119604d1-1ae8-43f9-b8e7-7138fc746371" Description="Description for drpham.TaskTop.OperationS
      <ShapeHasDecorators Position="Center" HorizontalOffset="0" VerticalOffset="0">
        <TextDecorator Name="NameDecorator" DisplayName="Name Decorator" DefaultText="NameDecorator" />
      </ShapeHasDecorators>
    </GeometryShape>
    <SwimLane Id="802430eb-b69b-4f17-a36f-91294acecc5d" Description="Description for drpham.TaskTop.ProcessSwimLane
      <Decorators>
        <SwimLaneHasDecorators Position="InnerTopLeft" HorizontalOffset="0" VerticalOffset="0">
          <TextDecorator Name="TextDecorator" DisplayName="Text Decorator" DefaultText="TextDecorator" />
        </SwimLaneHasDecorators>
      </Decorators>
    </SwimLane>
  </Shapes>
  <Connectors>...</Connectors>
  <XmlSerializationBehavior Name="TaskTopSerializ" Namespace="drpham.TaskTop">...</XmlSerializationBehavior>
  <ExplorerBehavior Name="TaskTopExplorer" />
  <ConnectionBuilders>...</ConnectionBuilders>
  <Diagram Id="a9cb8a18-ac0e-4" Description="Description for" Name="TaskTopDiagram" DisplayName="Minimal Languag"
  <Designer CopyPasteGeneration="CopyPasteOnly" FileExtension="taskop" EditorGuid="17070542-9dc7-4">...</Designer>
  <Explorer ExplorerGuid="41fb44bc-dbba-4" Title="TaskTop Explore">...</Explorer>
</Dsl>
```

Hình P.2: Văn phạm XML mô tả siêu mô hình

Sau khi đã xây dựng siêu mô hình và thiết lập giao diện thiết kế cho khung làm việc, chúng tôi tiến hành biên dịch và đóng gói khung làm việc. Hình P.3 minh họa khung làm việc DSL và T4 đã được xây dựng. Sử dụng khung làm việc này cho phép thiết kế và mô tả đồ thị phụ thuộc tác vụ trong giao diện đồ họa cũng như chuyển đổi tự động mô hình trực quan sang đặc tả dạng văn bản. Việc chuyển đổi tự động được thực hiện theo công nghệ sinh mã T4. Hình P.4 minh họa một mẫu T4 chúng tôi đã xây dựng. Đặc tả dạng văn bản của đồ thị tác vụ phụ thuộc được sử dụng làm đầu vào cho chương trình tối ưu bộ nhớ chiếm dụng dựa trên sắp xếp tô-pô.



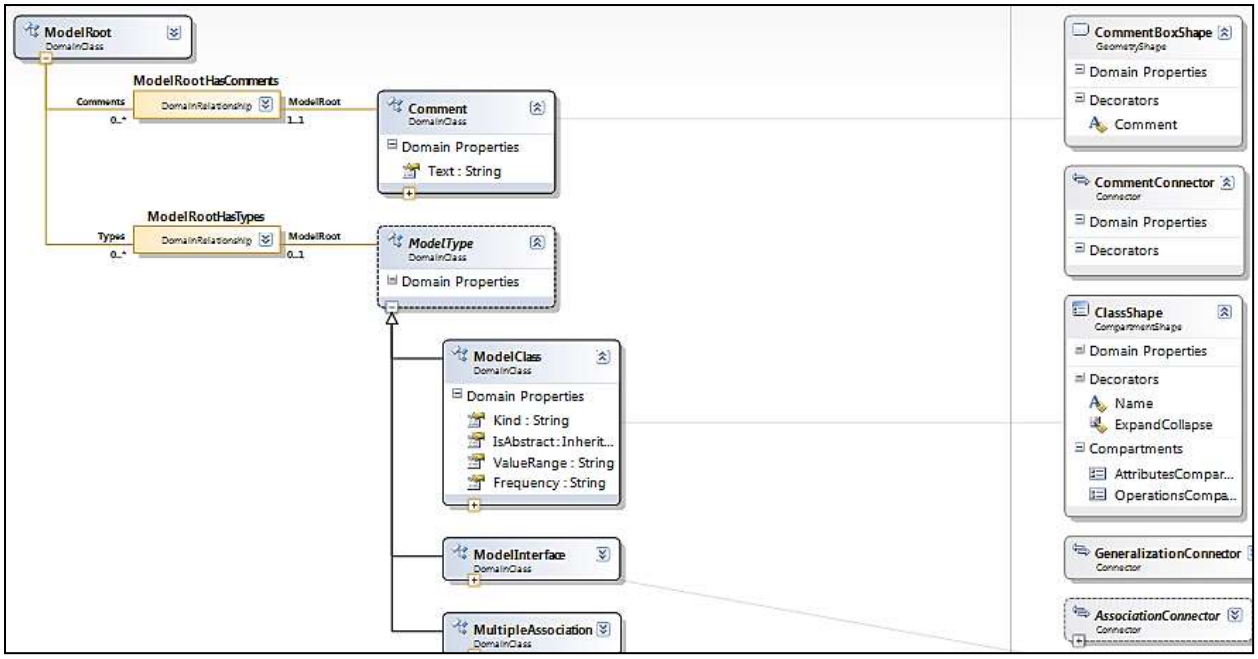
Hình P.3: Thiết kế đồ thị tác vụ phụ thuộc trong khung làm việc

```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.ModelingT
<#@ output extension=".txt" #>
<#@ TaskTop processor="TaskTopDirectiveProcessor" requires="fileName='Test.ta
<#
  // When you change the DSL Definition, some of the code below may not work.
  // Moi Process nhu 1 noi chua cho 1 do thi
  foreach (Process element in this.ProcessModel.Elements)
  {
    //voi moi dinh (FlowElement) trong Process hien tai
    foreach(FlowElement op in element.FlowElements)
    {
        String type = op.GetType().ToString();
        if(type == "drpham.TaskTop.Operation")
        {
            Operation opTemp = (Operation)op;
#>
            [!TaskName]<#=opTemp.Name#>
            [!ReturnType]<#=opTemp.ReturnType#>
            [!Parameters]<#=opTemp.ParameterList#>
<#
            //So canh di den dinh EndStart.GetLinksToFrontMe(opTemp).Count
            // Duyet tren moi canh den dinh
            if(EndStart.GetLinksToFrontMe(opTemp).Count > 0)
            {
                foreach (EndStart e in EndStart.GetLinksToFrontMe(opTemp))
                {
                    Operation opSource = e.SourceFlowElement as Operation;
                        if(opSource != null)
                        {
                            // write the current edge
        #>
                            [!$edge]<#= opSource.Name#>><#= opTemp.Name#>
        <#
                        }
```

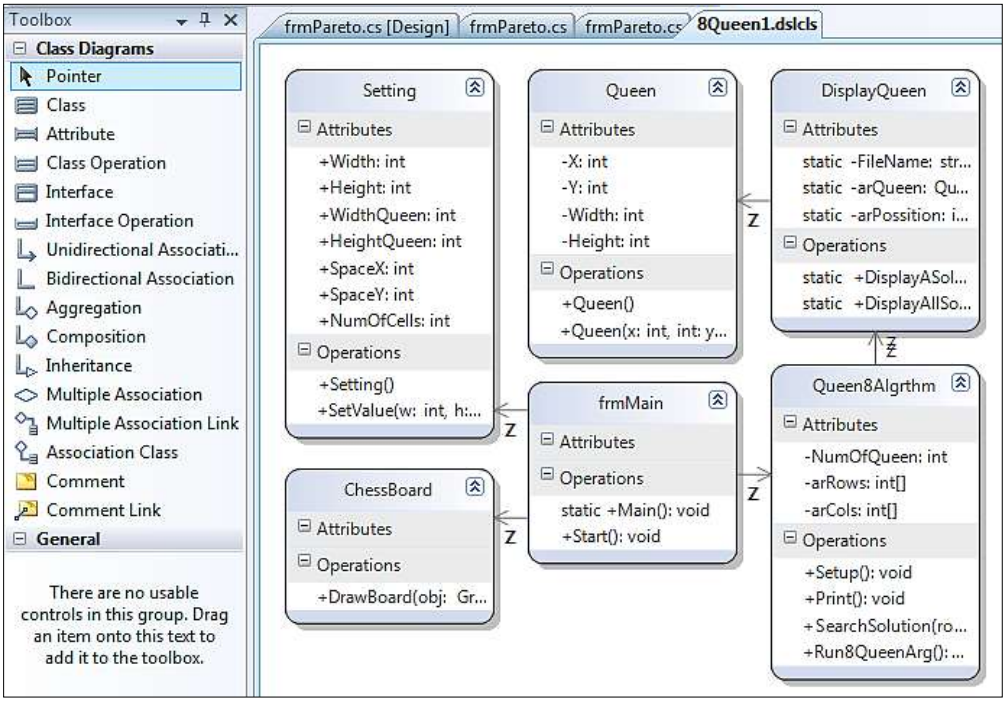
Hình P.4: Mẫu T4 sinh mã đặc tả đồ thị phụ thuộc

ii. Khung làm việc DSL và T4 để thiết kế mô hình dữ liệu trừu tượng

Để thực hiện tối ưu phần mềm nhúng trong giai đoạn thiết kế dựa trên chuyển đổi mô hình, chúng tôi xây dựng khung làm việc DSL và T4 cho phép thiết kế và sinh đặc tả dạng văn bản. Mô hình dữ liệu trừu tượng của hệ thống bao gồm một tập các cấu trúc và quan hệ giữa các cấu trúc. Mỗi cấu trúc bao gồm một tập các thành phần dữ liệu và tập các thao tác. Mỗi thành phần dữ liệu được đặc tả bao gồm tên, kiểu dữ liệu, miền giá trị và tần số truy xuất được ước lượng. Mỗi thao tác được đặc tả gồm tên, kiểu dữ liệu trả về và tần số thực hiện cũng được ước lượng. Khung làm việc này cũng được xây dựng theo các bước tương tự như xây dựng khung làm việc trước. Đầu tiên chúng tôi định nghĩa các lớp lôgic và các lớp ký hiệu như trong Bảng P.1. Sau đó xây dựng siêu mô hình như trong Hình P.5. Phía bên trái là các lớp lôgic biểu diễn các lớp, các thuộc tính, các phương thức và quan hệ giữa các thành phần trong biểu đồ lớp. Các lớp ký hiệu tương ứng với lớp ngữ nghĩa được xây dựng trong phần bên phải Hình P.5. Sau khi biên dịch và đóng gói, có thể sử dụng khung làm việc này để thiết kế mô hình dữ liệu và sinh đặc tả dạng văn bản như trong Hình P.6. Các lớp ký hiệu được xây dựng trong Hình P.5 sẽ được thể hiện trong hộp công cụ phía bên trái trong Hình P.6. Chúng tôi cũng xây dựng các mẫu T4 và tích hợp vào khung làm việc này để sinh mã tự động như trong Hình P.7.



Hình P.5: Thiết kế siêu mô hình cho DSL đặc tả mô hình dữ liệu trừu tượng



Hình P.6: Sử dụng khung làm việc để thiết kế mô hình dữ liệu trừu tượng

Hình P.7 mô tả mẫu T4 được sử dụng để sinh đặc tả dạng văn bản tự động từ mô hình. Trong mẫu T4, văn bản tĩnh được kết hợp với mã C# sử dụng để lấy đặc tả của các phần tử từ mô hình được chứa trong chỉ dẫn <# #>. Tệp tin văn bản đặc tả mô hình có phần mở rộng *.txt*.

Bảng P.1. Các lớp ngữ nghĩa và trực quan chính trong siêu mô hình

Lớp ngữ nghĩa	**Mô tả**	**Lớp trực quan tương ứng**
Comment	Lớp biểu diễn chú thích	CommentBoxShape
ModelClass	Lớp biểu diễn một cấu trúc	ClassShape
ModelAttribute	Lớp biểu diễn một thành phần dữ liệu	Không có
ModelOperation	Lớp biểu diễn một hàm	Không có
ModelMultiAssociation	Lớp biểu diễn liên kết giữa hai cấu trúc	AssociationConnector

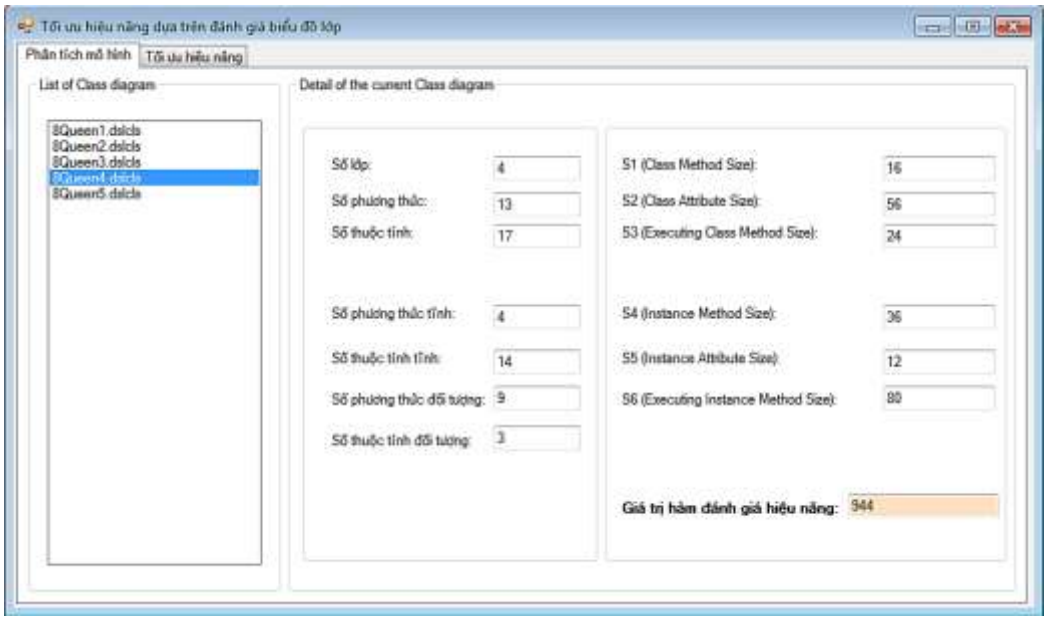
```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.VSHost.
<#@ output extension=".txt" #>
<#@ P03_Pareto_Class processor="P03_Pareto_ClassDirectiveProcessor"
<#
  // Cac bien luu tru, thong ke tham so tu bieu do
  string[] arClass;
  arClass = new string[this.ModelRoot.Types.Count];
  //duyet cac class trong model:
  foreach (ModelType type in this.ModelRoot.Types)
  { //Hien thi ten Class
#>
    <#= "@Class:" #><#= type.Name #>
<#
    //Lay thong tin lop hien tai
    ModelClass modelClass = type as ModelClass;
    if (modelClass != null)
    {
        //get methods:
        if(modelClass.Operations.Count >0)
        {
#>
<#
            foreach(ClassOperation op in modelClass.Operations)
            {
#>
            @@Method:<#=op.Name#>
<#
            }
        }
```

Hình P.7: Minh họa mẫu T4 lấy đặc tả từ biểu đồ lớp

P.1.2. Chương trình tối ưu hiệu năng dựa trên đánh giá biểu đồ lớp

Để triển khai thực nghiệm tối ưu hiệu năng dựa trên đánh giá mô hình trong Chương 2, chúng tôi xây dựng chương trình tối ưu như trong Hình P.8. Đầu vào của chương trình là một tập các biểu đồ lớp của phần mềm đặc tả theo theo dạng văn bản. Đặc tả dạng văn bản đã được chuyển tự động từ mô hình sau khi thiết kế mô hình dựa trên khung làm việc DSL và T4 đã được trình bày trong phần *P.1.1*. Đầu ra là biểu đồ lớp có hiệu năng tốt nhất được đặc tả theo dạng văn bản hoặc mô hình trực quan. Chương trình bao gồm các chức năng chính cho phép phân tích, đánh giá và lựa chọn mô hình tối ưu. Chức năng đầu tiên là phân tích mô hình và trích xuất các tham số ảnh hưởng đến hiệu

năng. Chức năng thứ hai là tính toán các độ đo ảnh hưởng đến hiệu năng từ S_1 đến S_6 dựa trên các tham số đã lấy được từ mô hình. Chức năng tiếp theo sẽ duyệt và tính giá trị hàm đánh giá hiệu năng dựa trên các độ đo S_1 đến S_6. Cuối cùng dựa trên giá trị của hàm đánh giá hiệu năng, chúng tôi sẽ thu được mô hình có hiệu năng tốt nhất trong tập mô hình ban đầu. Để viết chương trình này, chúng tôi sử dụng ngôn ngữ C#, nền tảng và môi trường phát triển tích hợp Visual Studio .NET 2008. Hình P.9 minh họa một phần mã nguồn chương trình tối ưu hiệu năng. Đoạn mã này phân tích từng dòng trong tệp tin đặc tả mô hình và trích xuất các lớp (bắt đầu băng @), các thuộc tính (bắt đầu bằng @@@), phương thức (bắt đầu bằng @@) và các đặc điểm của mỗi thành phần để tính các độ đo rồi tính giá trị hàm đánh giá hiệu năng theo công thức (2.7).



Hình P.8: Giao diện chương trình tối ưu dựa trên đánh giá mô hình

```
397         /// <summary>
398         /// Tinh gia tri ham danh gia hieu nang
399         /// de lua chon mo hinh toi uu
400         /// </summary>
401         private void lstListDiagram_SelectedIndexChanged(object sender, EventArgs e)
402         {
403             string fName = lstListDiagram.SelectedItem.ToString();
404             string file = fName.Substring(0, fName.IndexOf('.'));
405             file = folderPath +@"\"+ file + ".txt";
406             string[] arLine = GetAllLines(file);
407             Lay cac mang thanh phan trong class diagram
427             tinh cac do do:
440             Hien thi cac do do
449             //ham danh gia hieu nang:
450             //f_p=2*(S1+S2 )+4*S3+3*(S4+S5)+7*S6
451             double fPerform = 2 * ((double)sizeOfClassAtts + sizeOfClassMethod)
452                 + 4 * sizeOfExecutingClassMethod + 3*(sizeOfInstanceAtts + sizeOfIns
453                 + 7*sizeOfExecutingInstanceMethod;
454
455             if (Double.IsInfinity(fPerform))
456             {
457                 fPerform = MAX;
458             }
459             txtPerformanceFunction.Text = fPerform.ToString();
460         }
```

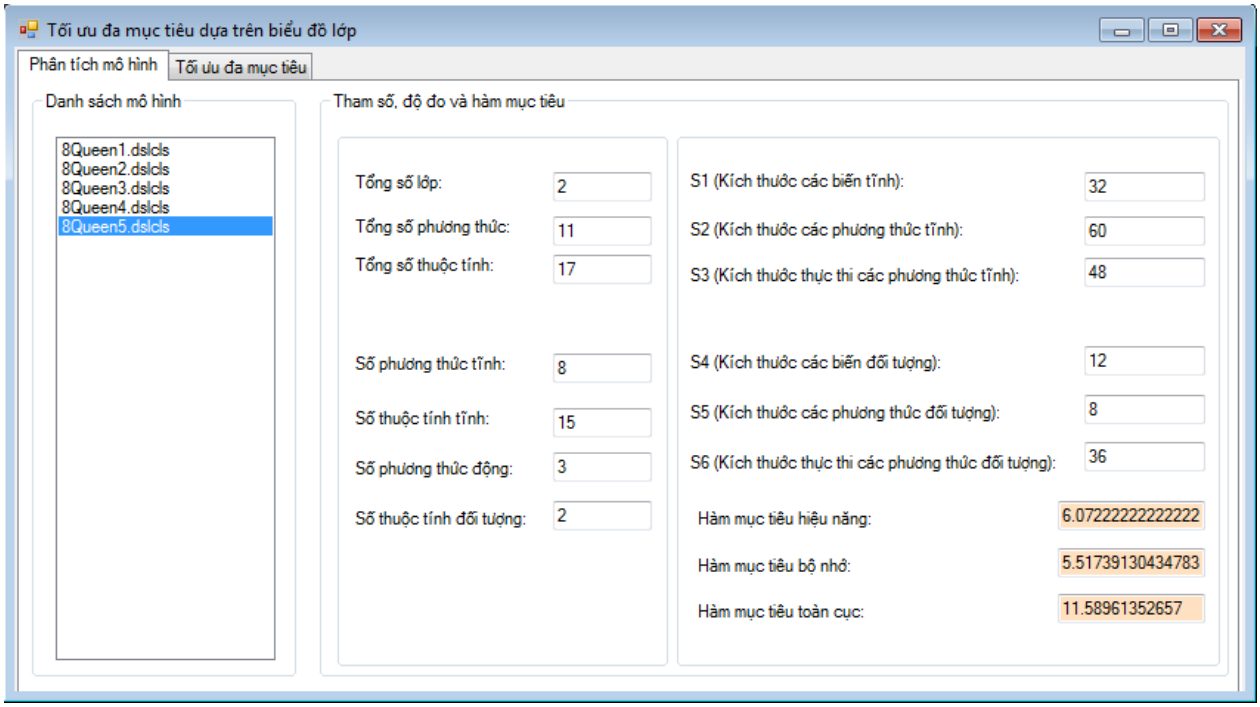
Hình P.9: Minh họa mã nguồn chương trình tối ưu hiệu năng

P.1.3. Chương trình tối ưu đa mục tiêu dựa trên biểu đồ lớp

Chương trình tối ưu đa mục tiêu được chúng tôi xây dựng để triển khai thử nghiệm phương pháp tối ưu đa mục tiêu Pareto và đánh giá mô hình đã trình bày trong chương 2. Chương trình nhận đầu vào là một tập các biểu đồ lớp khác nhau đã được thiết kế cho một ứng dụng và được theo định dạng văn bản. Đầu ra của chương trình là biểu đồ lớp cân bằng nhất giữa các mục tiêu tối ưu. Chương trình này gồm các chức năng chính sau:

- Phân tích, trích xuất các tham số cho mỗi biểu đồ lớp trong tập đầu vào
- Tính toán các độ đo S_1 đến S_6
- Tính giá trị các hàm mục tiêu hiệu năng, hàm mục tiêu bộ nhớ và hàm mục tiêu toàn cục
- Lựa chọn mô hình có hàm mục tiêu toàn cục lớn nhất. Đây là mô hình cân bằng nhất giữa mục tiêu hiệu năng và bộ nhớ.

Để xây dựng chương trình này, chúng tôi cũng sử dụng ngôn ngữ C# trên nền tảng .NET 3.5 và trong môi trường Visual Studio .Net 2008. Chương trình đã được xây dựng với giao diện như minh họa trong Hình P.10.



Hình P.10: Chương trình tối ưu đa mục tiêu Pareto dựa trên đánh giá mô hình

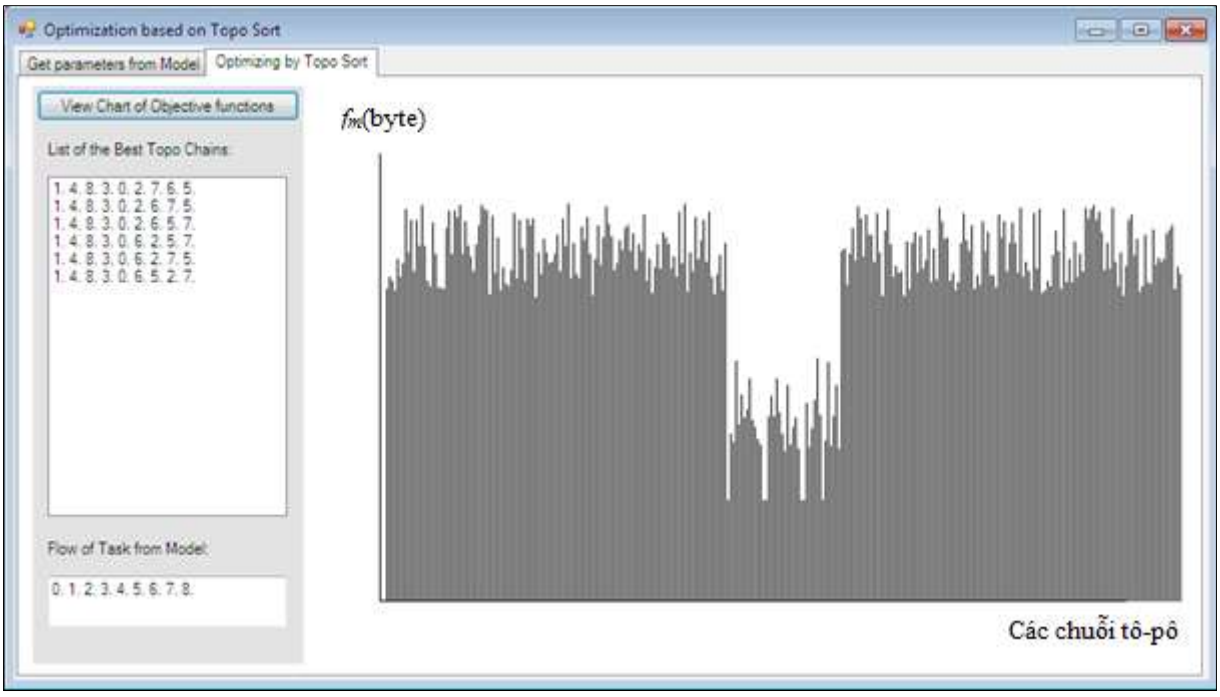
P.1.4. Chương trình tối ưu bộ nhớ dựa trên sắp xếp tô-pô

Như đã trình bày trong Chương 2, phương pháp tối ưu dựa trên sắp xếp tô-pô nhằm tối ưu dung lượng bộ nhớ chiếm dụng trong quá trình thực thi chương trình. Khi thực thi một chương trình trong hệ thống nhúng, một hàm chính được gọi thực thi đầu

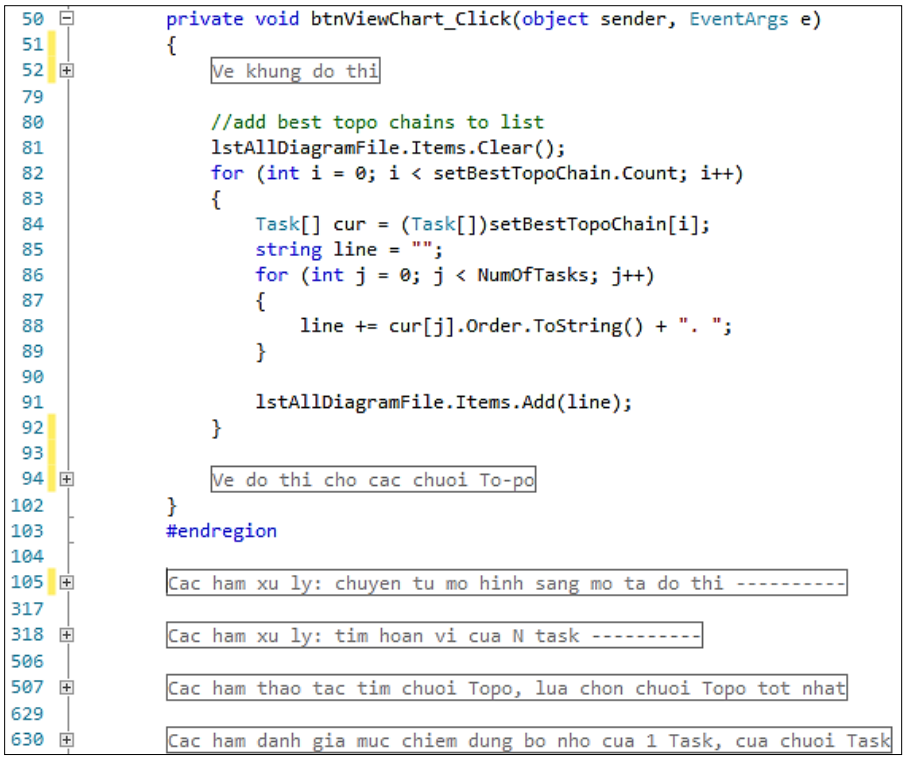
tiên và trên nền hàm chính sẽ thực hiện các tác vụ khác. Mỗi tác vụ được lập trình thành một hàm. Chuỗi thực thi các tác vụ có thể thay đổi mà không làm ảnh hưởng đến kết quả thực thi. Mỗi chuỗi thực thi các tác vụ phải thỏa mãn sắp xếp tô-pô của đồ thị phụ thuộc. Phương pháp tối ưu bộ nhớ dựa trên sắp xếp tô-pô nhằm tìm ra chuỗi thực thi các tác vụ có dung lượng bộ nhớ chiếm dụng nhỏ nhất. Để thử nghiệm phương pháp này, chúng tôi đã xây dựng chương trình tối ưu bộ nhớ dựa trên sắp xếp tô-pô với giao diện như trong Hình P.11. Đầu vào của chương trình là đồ thị phụ thuộc giữa các tác vụ đã được chuyển từ mô hình trực quan sang dạng văn bản. Đầu ra của chương trình là chuỗi tô-pô có dung lượng bộ nhớ chiếm dụng ít nhất khi thực thi chương trình theo chuỗi này. Chương trình gồm các chức năng chính sau:

- Lấy và phân tích các tham số từ đồ thị tác vụ phụ thuộc
- Tìm các chuỗi tô-pô trên đồ thị tác vụ phụ thuộc
- Đánh giá dung lượng bộ nhớ chiếm dụng trên mỗi chuỗi tô-pô
- Tìm chuỗi tô-pô có dung lượng bộ nhớ chiếm dụng nhỏ nhất.

Để đánh giá dung lượng bộ nhớ chiếm dụng trên mỗi chuỗi tô-pô, chúng tôi đã xây dựng công thức và lập trình hàm đánh giá theo công thức (2.14). Chương trình cũng được viết bằng ngôn ngữ C# trên nền tảng .NET 3.5 và trong môi trường Visual Studio .Net 2008. Hình P.12 minh họa một số hàm xử lý chính trong mã nguồn chương trình.



Hình P.11: Chương trình tối ưu mức chiếm dụng bộ nhớ dựa trên sắp xếp tô-pô



Hình P.12: Mã nguồn chính chương trình tối ưu bộ nhớ dựa trên sắp xếp tô-pô

P.1.5. Chương trình tối ưu dựa trên chuyển đổi mô hình

Sau khi xây dựng khung làm việc DSL và T4 để thiết kế và lấy thông tin đặc tả tự động từ mô hình chúng tôi xây dựng chương trình tối ưu dựa trên chuyển đổi mô hình. Đầu vào của chương trình là thông tin đặc tả mô hình và bộ tham số tương ứng với mô hình đã được trích xuất tự động từ khung làm việc DSL và T4. Đầu ra là mô hình tối ưu được biểu diễn dạng toán học trong phần trước hoặc dạng trực quan. Giá trị của hàm đánh giá trên mô hình ban đầu và mô hình tối ưu cũng được thể hiện trong chương trình để kiểm chứng. Trong nghiên cứu này, chúng tôi xây dựng chương trình với ba lựa chọn tối ưu là tối ưu hiệu năng, tối ưu dung lượng bộ nhớ chiếm dụng và tối ưu đa mục tiêu. Chương trình tối ưu đã được xây dựng như trong Hình P.13 và một phần mã nguồn chương trình được minh họa trong Hình P.14.

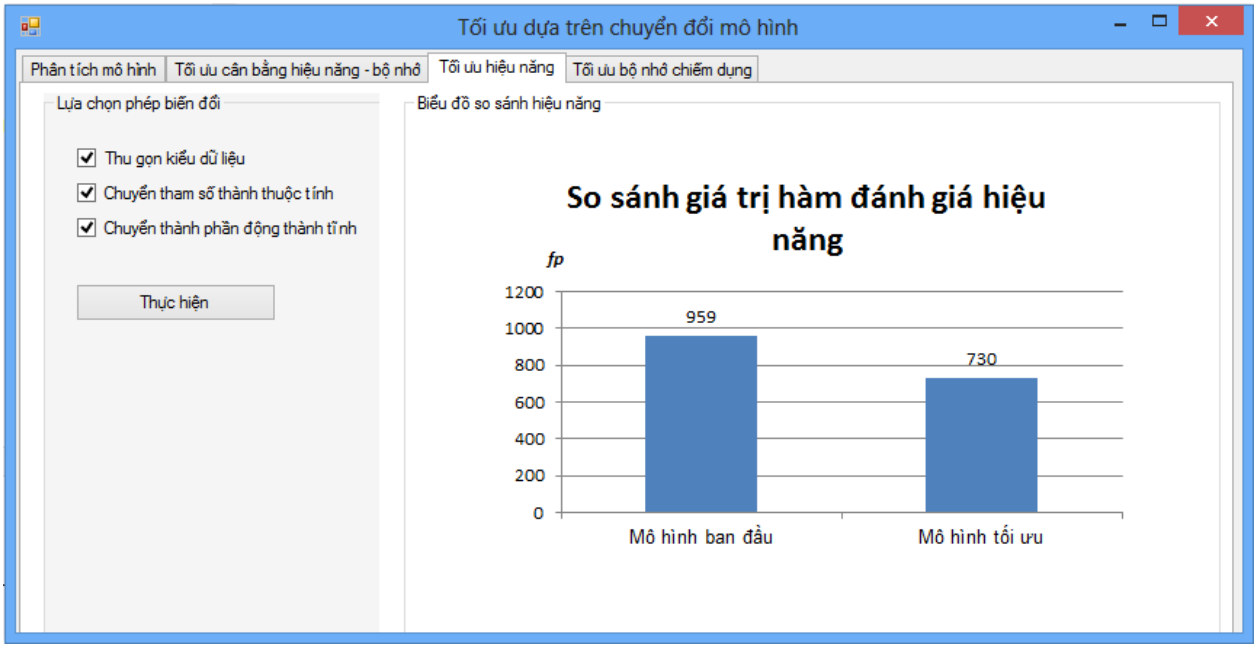
i. Chuyển đổi mô hình để tối ưu hiệu năng

Để thực hiện tối ưu hiệu năng dựa trên chuyển đổi mô hình, chúng tôi lập trình ba phép chuyển đổi đó là thu gọn kiểu dữ liệu, chuyển các tham số thành các thành phần dữ liệu và chuyển các thành phần dữ liệu cũng như các phương thức động thành tĩnh. Trong phép chuyển đổi thứ nhất, chương trình sẽ căn cứ vào miền giá trị của các thành viên dữ liệu để chuyển về kiểu nhỏ nhất có thể. Hai phép chuyển đổi sau được thực hiện theo ràng buộc về tần số sử dụng. Chỉ các phương thức nào có tần số sử dụng lớn hơn hoặc bằng hơn tần số ngưỡng mới được thực hiện phép chuyển đổi thứ hai. Chỉ các thành viên dữ

liệu và phương thức có tần số lớn hơn hoặc bằng tần số ngưỡng mới được thực hiện phép chuyển đổi thứ ba.

ii. Chuyển đổi mô hình để tối ưu bộ nhớ chiếm dụng

Để thực hiện tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình, chúng tôi lập trình ba phép chuyển đổi là thu gọn kiểu dữ liệu, phân chia và gộp cấu trúc. Phép chuyển đổi đầu tiên cũng được thực hiện như trong phần tối ưu hiệu năng. Hai phép biến đổi sau được thực hiện dựa trên tần số sử dụng. Trong mô hình dữ liệu, các thành phần dữ liệu và các hàm có thêm thuộc tính tần số truy xuất hoặc thực hiện. Tần số này sẽ được người thiết kế ước lượng. Mô hình dữ liệu được sử dụng làm đầu vào cho chương trình tối ưu để thực hiện các phép biến đổi. Trong chương trình tối ưu, chúng tôi lập trình phép biến đổi phân chia cấu trúc dựa trên thuật toán phân cụm K-mean với số phân cụm tùy chọn. Số phân cụm chính là số cấu trúc được phân chia từ cấu trúc ban đầu. Chương trình tối ưu phân tích và chuyển từ mô hình trực quan sang biểu diễn toán học của mô hình dựa trên T4. Sau đó thực hiện phép biến đổi để đạt được mô hình tối ưu dạng tập hợp. Từ mô hình tối ưu dạng tập hợp, có thể sinh ngược lại mô hình trực quan hoặc sinh mã.



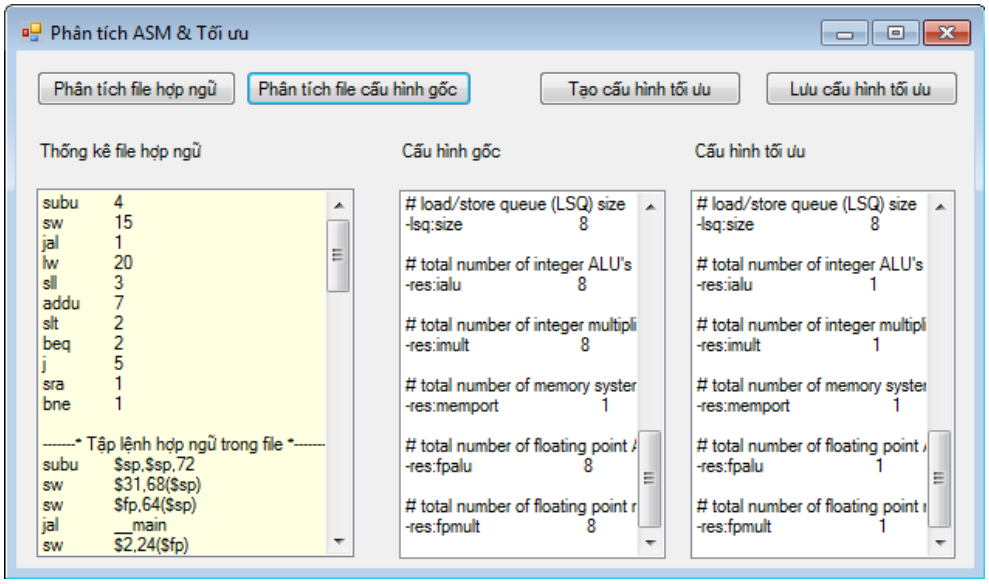
Hình P.13: Giao diện chương trình tối ưu dựa trên chuyển đổi mô hình

```
308         Cac ham ve do thi
396         /// <summary>
397         /// Phan tich bieu do hien tai trong class
398         /// </summary>
399         private void lstListDiagram_SelectedIndexChanged(object sender, EventArgs e)
400         {
401             string fName = lstListDiagram.SelectedItem.ToString();
402             string file = fName.Substring(0, fName.IndexOf('.'));
403             file = folderPath +@"\"+ file + ".txt";
404             string[] arLine = GetAllLines(file);
405             Lay cac mang thanh phan trong class diagram
425             //tinh cac do do:
426             //size method:
427             int sizeOfClassMethod = MeasuresFromClass.SizeOfMethods(NumOfClassMethod);
428             int sizeOfInstanceMethod = MeasuresFromClass.SizeOfMethods(NumOfInstanceMethod);
429
430             //size attributes:
431             int sizeOfClassAtts = MeasuresFromClass.SizeOfAtts(arClassAtt, NumOfClassAtt);
432             int sizeOfInstanceAtts = MeasuresFromClass.SizeOfAtts(arInstanceAtt, NumOfInstanceAtt);
433
434             //size executing method
435             int sizeOfExecutingClassMethod = MeasuresFromClass.SizeToExecuteMethod(arClassMethod, N
436             int sizeOfExecutingInstanceMethod = MeasuresFromClass.SizeToExecuteMethod(arInstanceMet
437
438             Tinh gia tri cac ham danh gia
463         }
```

Hình P.14: Mã nguồn phân tích mô hình

P.1.6. Chương trình phân tích mã hợp ngữ để tìm cấu hình tối ưu

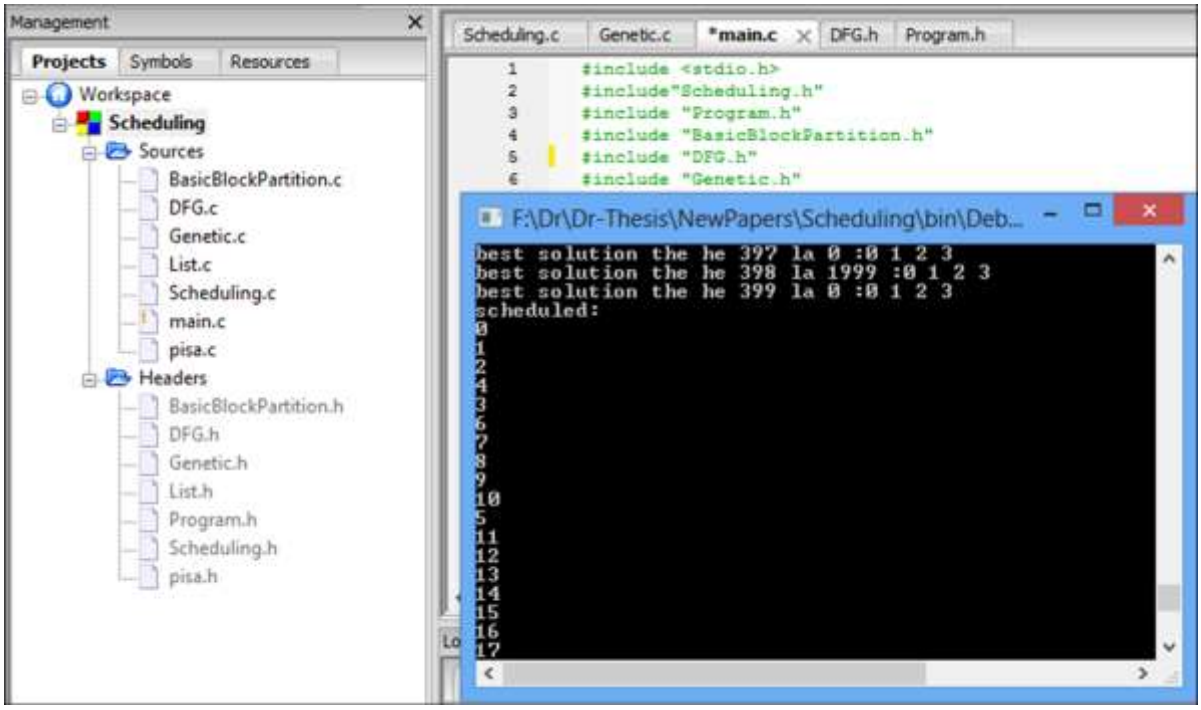
Để tìm được cấu hình tối ưu điện năng tiêu thụ cho một hệ thống nhúng đã có, chúng tôi xây dựng một chương trình phân tích mã hợp ngữ và hồ sơ CPU đơn giản như minh họa trong Hình P.15. Từ mã hợp ngữ được dịch ngược trong phần trước, chương trình duyệt và phân tích từng dòng lệnh để phân nhóm các đơn vị chức năng được sử dụng trong phần mềm nhúng. Chương trình cũng duyệt hồ sơ CPU để tìm tập các đơn vị chức năng của CPU. Dựa trên kết quả phân tích mã hợp ngữ và hồ sơ CPU, chương trình sẽ tìm các đơn vị chức năng được sử dụng và ghi ra tệp tin cấu hình mới chỉ bật các đơn vị chức năng này. Dựa trên tệp tin cấu hình được tạo ra từ chương trình phân tích, có thể tái cấu hình lại CPU để tối ưu điện năng tiêu thụ.



Hình P.15: Chương trình phân tích và tạo cấu hình tối ưu

P.1.7. Chương trình lập lịch các lệnh để tối ưu

Để tiến hành tối ưu mã hợp ngữ gắn với các CPU cụ thể, chúng tôi xây dựng chương trình lập lịch để tối ưu hiệu năng và điện năng tiêu thụ. Đầu vào của chương trình là một tệp tin hợp ngữ đã được biên dịch chéo từ một tệp tin nguồn C. Chương trình hợp ngữ được phân tích thành các khối cơ bản. Với mỗi khối cơ bản, chương trình xây dựng đồ thị phụ thuộc và áp dụng thuật toán di truyền để tìm chuỗi tô-pô tốt nhất. Tùy theo mục tiêu tối ưu và loại kiến trúc CPU, hàm thích nghi trong thuật toán di truyền sẽ được lập trình theo các phiên bản khác nhau. Mã hợp ngữ trong mỗi khối cơ bản được sắp xếp lại theo thứ tự tô-pô tốt nhất. Kết hợp các đoạn mã tối ưu của các khối cơ bản để tạo chương trình hợp ngữ tối ưu. Chương trình này được chúng tôi xây dựng như minh họa trong Hình P.16.



Hình P.16: Giao diện chương trình lập lịch

P.2. Các chương trình sử dụng trong kiểm chứng

Trong các phần đầu chương, chúng tôi đã trình bày một số chương trình tối ưu cho mỗi phương pháp tối ưu. Để đánh giá các phương pháp tối ưu cũng như các chương trình tối ưu đã xây dựng, chúng tôi sử dụng các chương trình ví dụ để thực nghiệm. Nội dung phần này sẽ trình bày về các chương trình ví dụ được sử dụng trong quá trình thực nghiệm. Các chương trình thử nghiệm được chúng tôi lập trình hoặc sử dụng mã nguồn mở. Mỗi chương trình thử nghiệm có thể được tổ chức theo các phiên bản mã nguồn khác

nhau tương ứng với các mô hình trong giai đoạn thiết kế. Tuy nhiên các thuật toán được giữ nguyên để không ảnh hưởng tới giá trị các hàm đánh giá. Một số chương trình thử nghiệm được chúng tôi lập trình như chương trình nhận dạng chữ Nôm trên điện thoại di động và chương trình nhận dạng chữ Nôm trong môi trường phân tán. Một số chương trình nguồn mở được chúng tôi biên soạn lại mã nguồn theo các mô hình phần mềm để tiến hành thực nghiệm như tháp Hà Nội, tám quân Hậu và nhân ma trận.

P.2.1. Chương trình nhận dạng chữ Nôm trên PocketPC

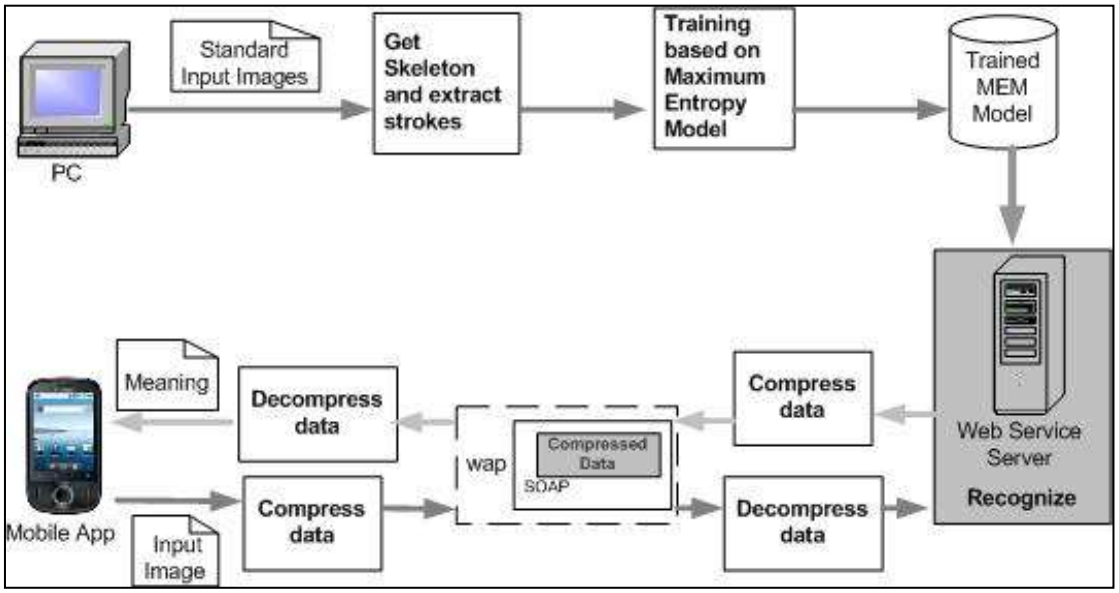
Hệ thống nhận dạng chữ Hán-Nôm gồm ba giai đoạn chính: tiền xử lý, huấn luyện và nhận dạng. Trong hệ thống này, chúng tôi phát triển phương pháp nhận dạng theo mô hình cực đại Entropy kết hợp với trích chọn đặc trưng theo tọa độ điểm đen của khung xương. Phương pháp nhận dạng này đạt được độ chính xác 76,63%. Dữ liệu sử dụng là tập ảnh tạo từ bộ mã Nôm unicode và truyện Kiều 1866. Giai đoạn huấn luyện, tập mẫu vào chuẩn gồm ba loại font: chuẩn, nghiêng, đậm 3×8488 ký tự được tạo ra từ bộ mã unicode, phân đoạn văn bản, tách ký tự, lấy khung xương, trích chọn đặc trưng, lấy mã unicode tương ứng của ký tự, tính phân phối Entropy. Kết quả huấn luyện là mô hình phân phối Entropy của các ký tự. Trong giai đoạn nhận dạng, hệ thống cần nạp mô hình phân phối Entropy, tách ký tự và trích chọn đặc trưng và đưa tập đặc trưng vào mô hình để tính toán và chọn nhãn có xác suất lớn nhất.

Hình P.17: Chương trình nhận dạng chữ Nôm

Để thử nghiệm các phương pháp tối ưu chúng tôi sử dụng mô-đun nhận dạng trong hệ thống nhận dạng chữ Nôm. Mô-đun nhận dạng này có các phiên bản được lập trình theo các kiến trúc khác nhau. Tuy nhiên, thuật toán và mã nguồn trong các phương thức là không thay đổi. Do đó có thể sử dụng mô-đun nhận dạng này để đánh giá hiệu năng, mức chiếm dụng bộ nhớ mà không bị ảnh hưởng bởi thuật toán. Giao diện mô-đun nhận dạng được minh họa trong Hình P.17.

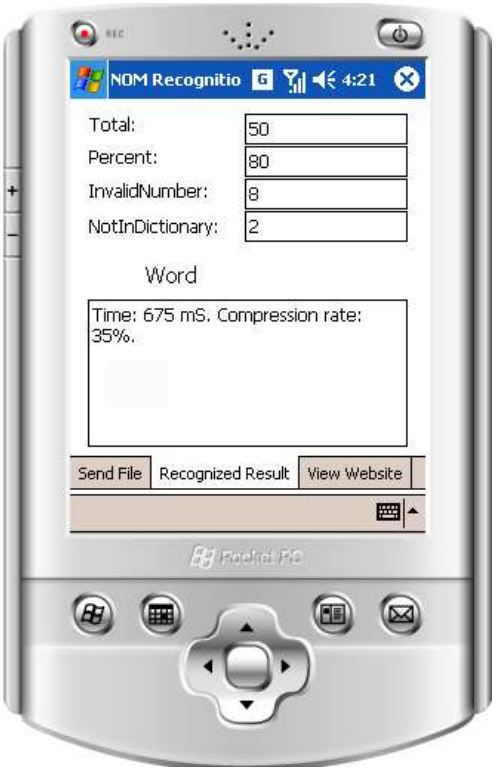
P.2.2. Chương trình nhận dạng chữ Nôm theo dịch vụ web

Để thử nghiệm phương pháp tối ưu hiệu năng phần mềm nhúng trong môi trường phân tán dựa trên nén dữ liệu, chúng tôi xây dựng và sử dụng chương trình nhận dạng chữ Nôm trong môi trường phân tán dựa trên dịch vụ web. Mô hình hoạt động của chương trình được chỉ ra trong Hình P.18. Chương trình có mô-đun nhận dạng đặt trên máy chủ và mô-đun I/O đặt trên điện thoại di động. Ảnh cần nhận dạng được chụp từ điện thoại di động, nén theo thuật toán GZIP sau đó được đặt trong gói SOAP và gửi lên dịch vụ nhận dạng trên máy chủ xử lý. Mô-đun nhận dạng trên máy chủ khi nhận được dữ liệu sẽ giải nén, tiến hành tiền xử lý, phân đoạn, tách ký tự, trích chọn đặc trưng và nhận dạng dựa vào mô hình Entropy cực đại (MEM) đã huấn luyện. Kết quả nhận dạng sẽ được nén nếu kích thước vượt quá giới hạn. Cuối cùng kết quả được giải nén và hiển thị trên điện thoại di động. Mô hình thuật toán được mô tả cụ thể như trong Hình P.19. Giao diện trên điện thoại di động của chương trình được minh họa trong Hình P.20. Hình P.21 minh họa mã nguồn xử lý trên máy phục vụ dịch vụ web và Hình P.22 minh họa mã nguồn xử lý trên điện thoại di động.



Hình P.18: Mô hình cải tiến hiệu năng Nhận dạng chữ Nôm phân tán

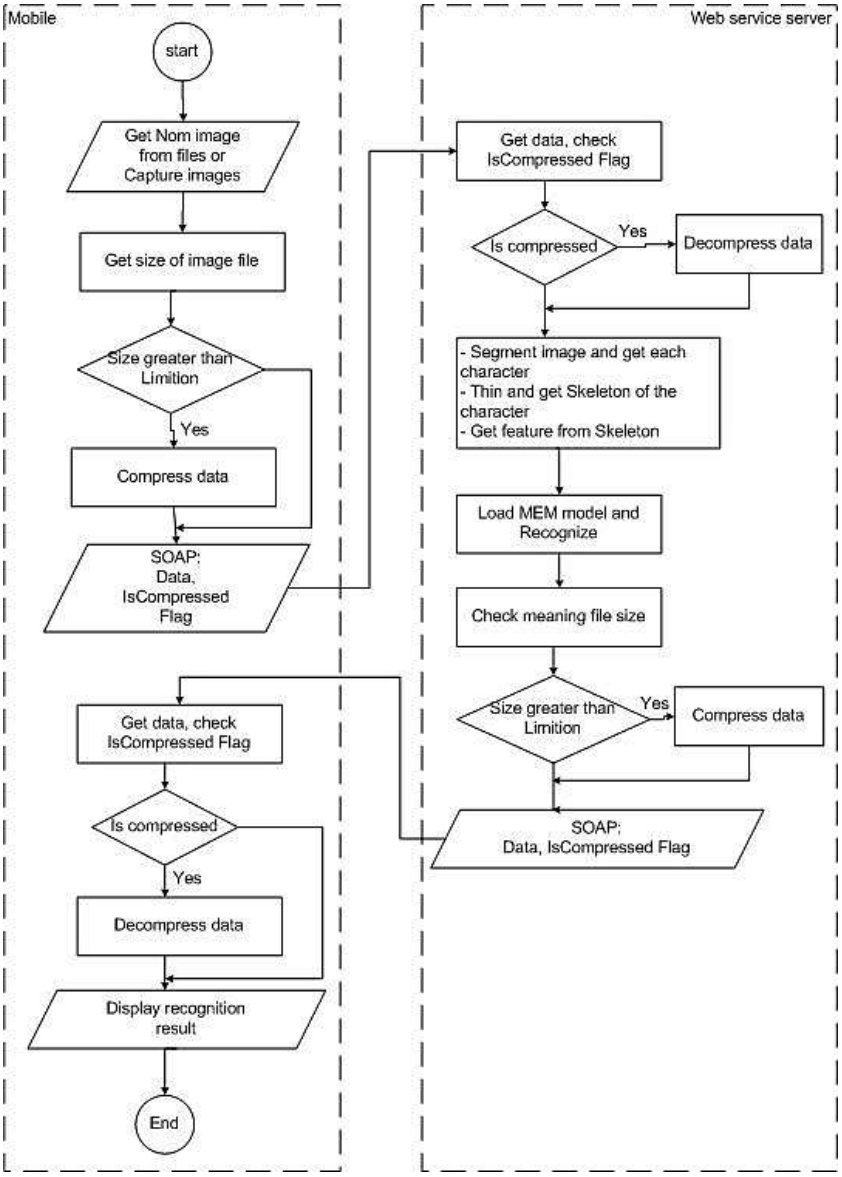
.



Hình P.20: Giao diện mô-đun nhận dạng chữ Nôm trên điện thoại di động

```
using System;
using System.Web;
using System.Collections;
using System.Web.Services;
using System.Web.Services.Protocols;


/// <summary>
/// Summary description for NomMEMReg
/// </summary>
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class NomMEMReg : System.Web.Services.WebService
{

    public NomMEMReg()...

    [WebMethod(Description="Method nhan dang. Tra ve mang ket qua")]
    public double[] Recognize_ResultArray(string fileName)...

    [WebMethod(Description = "Method nhan dang. Tra ve chuoi ket qua")]
    public string[] Recognize_ResultString(string fileName)...
}
```

Hình P.21: Mã nguồn dịch vụ web



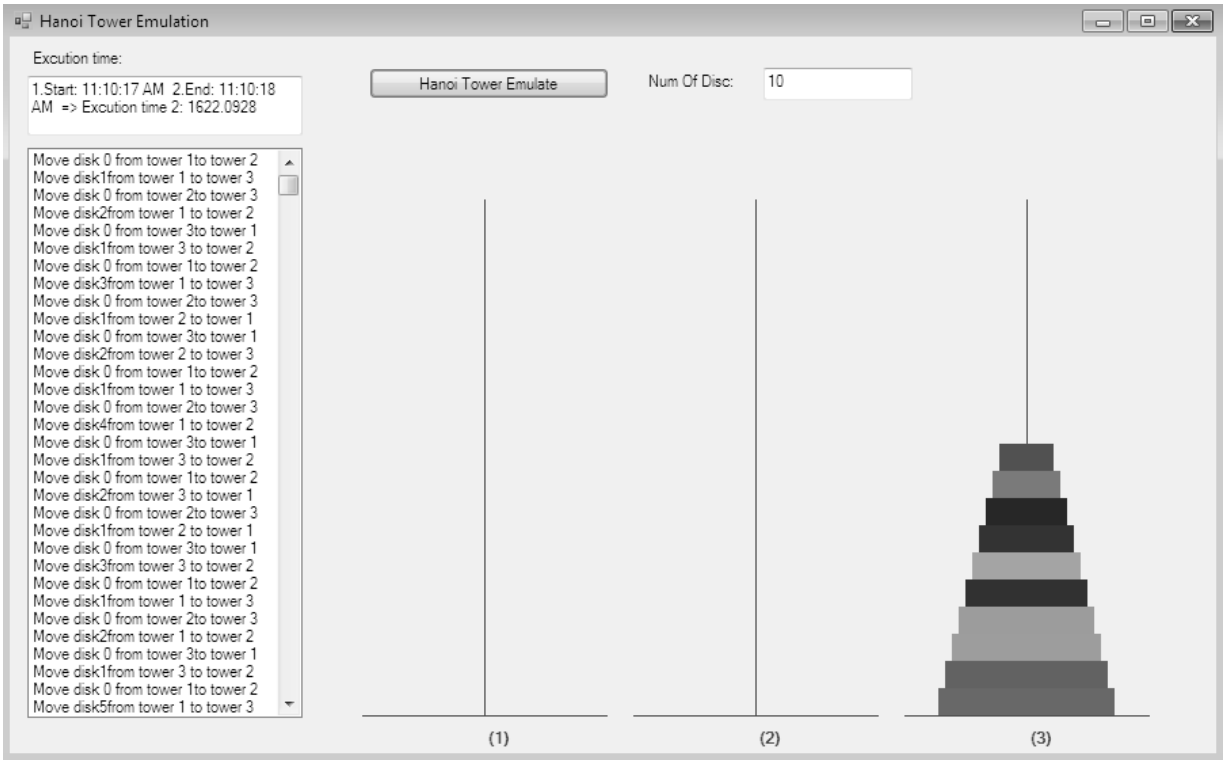
Hình P.19: Cải tiến hiệu năng nhận dạng chữ Nôm phân tán trên điện thoại di động

```
public partial class frmMainApp : Form
{
    /// <summary>
    /// Chua ten file anh
    /// </summary>
    public string strFileName = "";

    public frmMainApp()...

    /// <summary>
    /// Mo file de upload len server
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void btnOpen_Click(object sender, EventArgs e)...

    /// <summary>
    /// Upload file len server chua webservice
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void btnUpload_Click(object sender, EventArgs e)...

    private void btnReg_Click(object sender, EventArgs e)...
}
```

Hình P.22: Mã nguồn xử lý trên điện thoại di động

P.2.3. Tháp Hà Nội

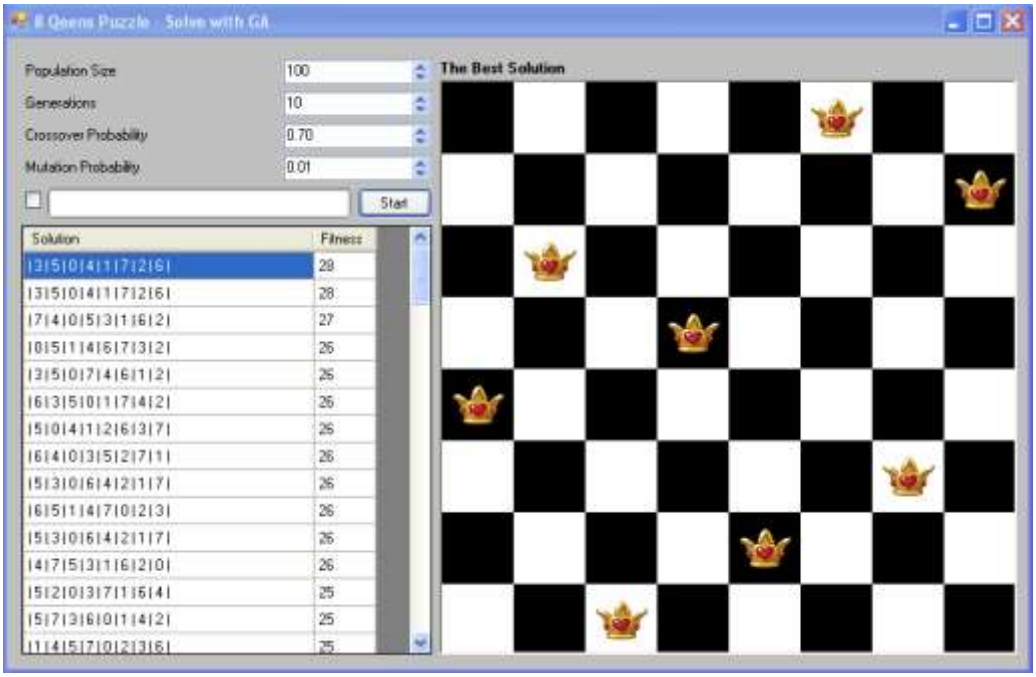
Để thử nghiệm phương pháp tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình, chúng tôi sử dụng chương trình ví dụ mô phỏng thuật toán đệ quy tháp Hà Nội. Tháp Hà Nội là bài toán nổi tiếng: sử dụng đệ quy để chuyển các đĩa từ tháp (1) sang tháp (3) theo đúng thứ tự, sử dụng tháp trung gian (2). Chương trình sẽ mô phỏng lại các bước chuyển đĩa bằng giao diện đồ họa và ghi lại mọi bước chuyển như trong Hình P.23. Chương trình này không được lập trình mà chúng tôi sử dụng mã nguồn mở sau đó biên soạn lại theo các mô hình thiết kế khác nhau để đánh giá.



Hình P.23: Giao diện chương trình tháp Hà Nội

P.2.4. Chương trình 8 quân Hậu

Chương trình 8 quân Hậu trong giao diện đồ họa được sử dụng để thử nghiệm phương pháp tối ưu bộ nhớ chiếm dụng dựa trên chuyển đổi mô hình. Chương trình này được chúng tôi sử dụng mã nguồn mở và biên soạn lại theo các phiên bản khác nhau. Mỗi phiên bản tương ứng với một mô hình phần mềm. Tuy nhiên, thuật toán và mã nguồn trong mỗi phương thức không được thay đổi để tránh ảnh hưởng của thuật toán đến việc đánh giá. Thực thi các phiên bản này trong cùng một môi trường để đánh giá kết quả tối ưu. Giao diện chương trình được chỉ ra trong Hình P.24.



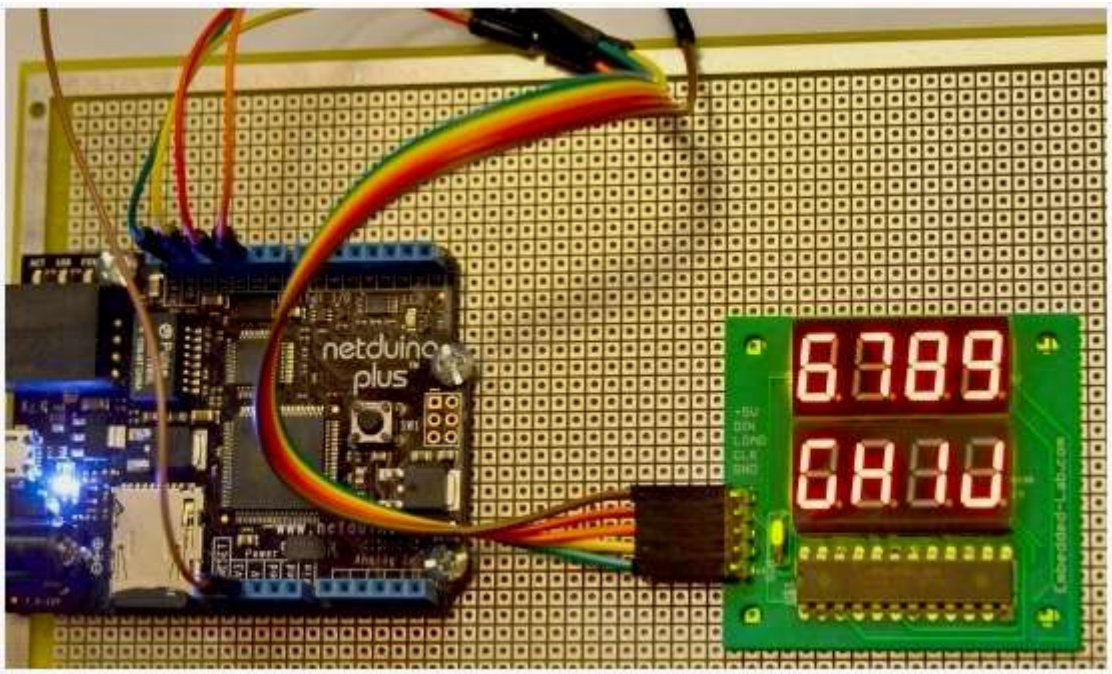
Hình P.24: Giao diện chương trình 8 quân Hậu

P.2.5. Các chương trình nhúng cho *Bo mạch Netduino* và *Netduino Plus*

Trong phần này, chúng tôi trình bày ba ví dụ về chương trình nhúng tích hợp trên *Bo mạch Netduino Plus* với vi xử lý ARM7 chứa trong vi điều khiển AT91SAM7X512 đó là *Netduino_8digit, Netduino_LCD* và *Netduino_SerialPort*. Đây là ba ví dụ nhúng điển hình được sử dụng để kiểm chứng phương pháp tối ưu hiệu năng dựa trên biểu đồ lớp, tối ưu hiệu năng dựa trên chuyển đổi mô hình và tối ưu đa mục tiêu. Các chương trình này và các thông tin về hệ thống nhúng tương ứng được tra cứu và tải về từ website http://embedded-lab.com/blog/?page_id=6225. Chúng tôi sử dụng mã nguồn của các chương trình này, sửa đổi và tái cấu trúc theo các biểu đồ lớp khác nhau, thực hiện mô phỏng để đánh giá kết quả tối ưu.

P.2.5.1. Chương trình điều khiển đèn LED 8 số *Netduino_8digit*

Netduino_8digit là chương trình hiển thị chuỗi 8 số trên đèn LED được tích hợp trong hệ thống nhúng như trong Hình P.25. *Bo mạch Netduino và Netduino Plus* là các nền tảng phát triển hệ thống nhúng nguồn mở dựa trên ngôn ngữ hướng đối tượng C# và .Net Micro Framework. Chương trình Netduino_8digit điều khiển việc hiển thị chuỗi số trên đèn LED được nạp vào vi điều khiển AT91SAM7X512 qua cổng USB. Chương trình được phát triển trong IDE *Microsoft Visual Express 2010* với *.NET Micro Framework SDK v4.2* và *Netduino SDK v4.2.2.0 (32-bit)*. Một phần mã nguồn chương trình được minh họa như trong Hình P.26.



Hình P.25: Hệ thống nhúng điều khiển đèn LED 8 số

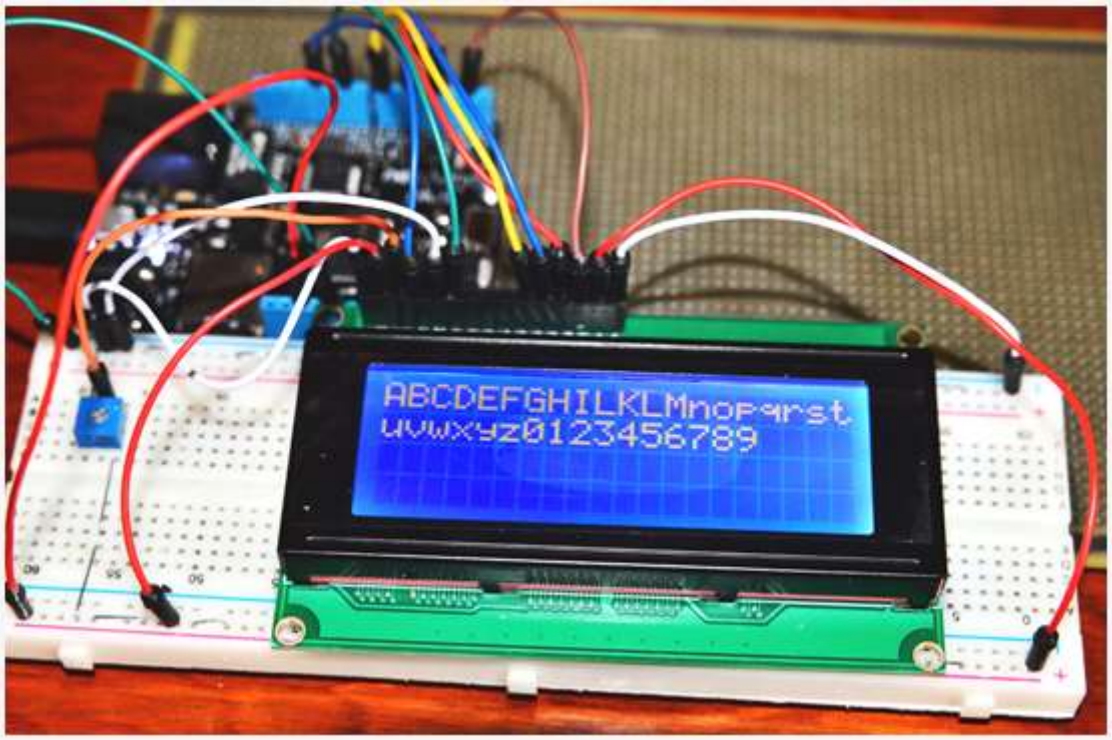
```
using System;
using System.Collections;
using System.Threading;
using Microsoft.SPOT;
using Microsoft.SPOT.Hardware;

namespace EmbeddedLab.NetduinoPlus.Day4.Display
{
    public class Max7219_LED
    {
        Constructor


        #region Public Methods
        public void Initialize()...
        public void Clear(byte numberOfDigits = 8)
        {
            for (byte i = 1; i <= numberOfDigits; i++)
                Write(i, 0x00);
        }
        public void SetIntensity(Intensity intensity)
        {
            Write((byte)RegisterAddressMap.Intensity, (byte)intensity);
        }
        public void SetBCDDecodeMode(BCDDecodeMode decodeMode)
        {
            Write((byte)RegisterAddressMap.DecodeMode, (byte)decodeMode);
        }
```

Hình P.26: Một phần mã nguồn chương trình *Netduino_8digit*

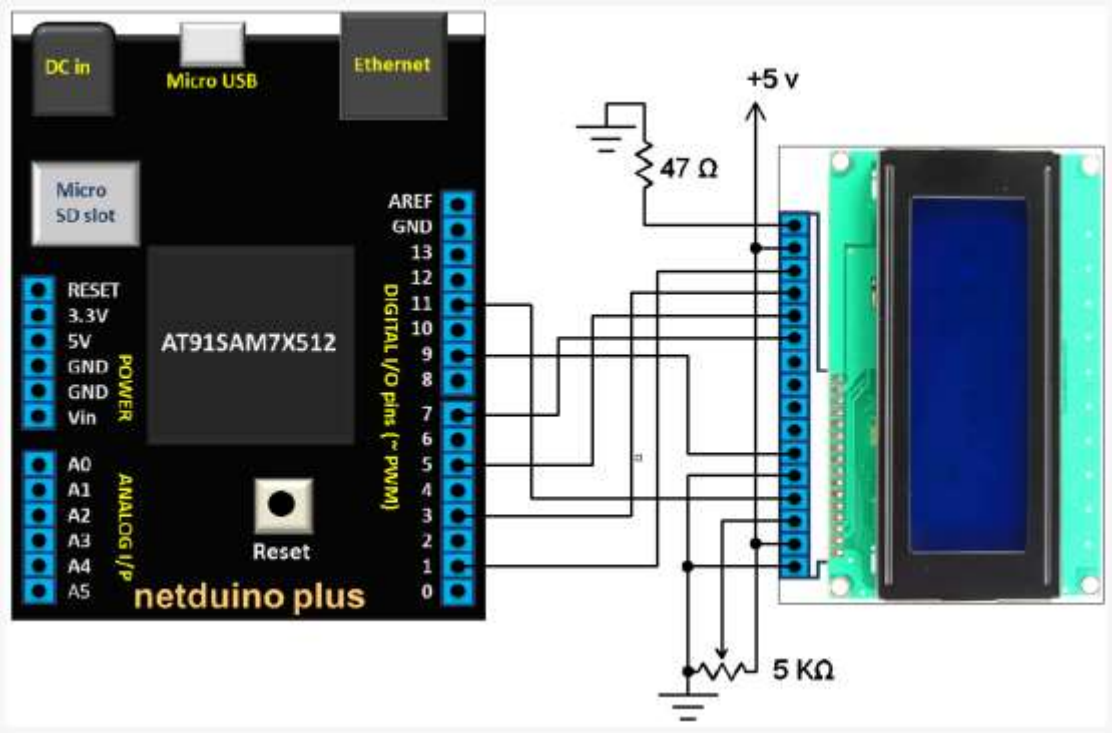
P.2.5.2. Chương trình hiển thị chuỗi ký tự trên màn hình *Netduino_LCD*

Màn hình LCD là thiết bị phổ biến để hiển thị chuỗi ký tự trong thời gian thực trên các hệ thống nhúng. Chương trình *Netduino_LCD* điều khiển việc hiển thị chuỗi ký tự

trên màn hình LCD với kết quả như trong Hình P.27. Chương trình được chạy trên *Bo mạch Netduino Plus* kết nối với màn hình LCD *HD44780U* (http://lcd-linux.sourceforge.net/pdfdocs/hd44780.pdf) như minh họa trong Hình P.28. Hình P.29 minh họa một phần mã nguồn của chương trình.



Hình P.27: Hệ nhúng hiển thị màn hình với *Bo mạch Netduino Plus* và màn hình *HD44780U*



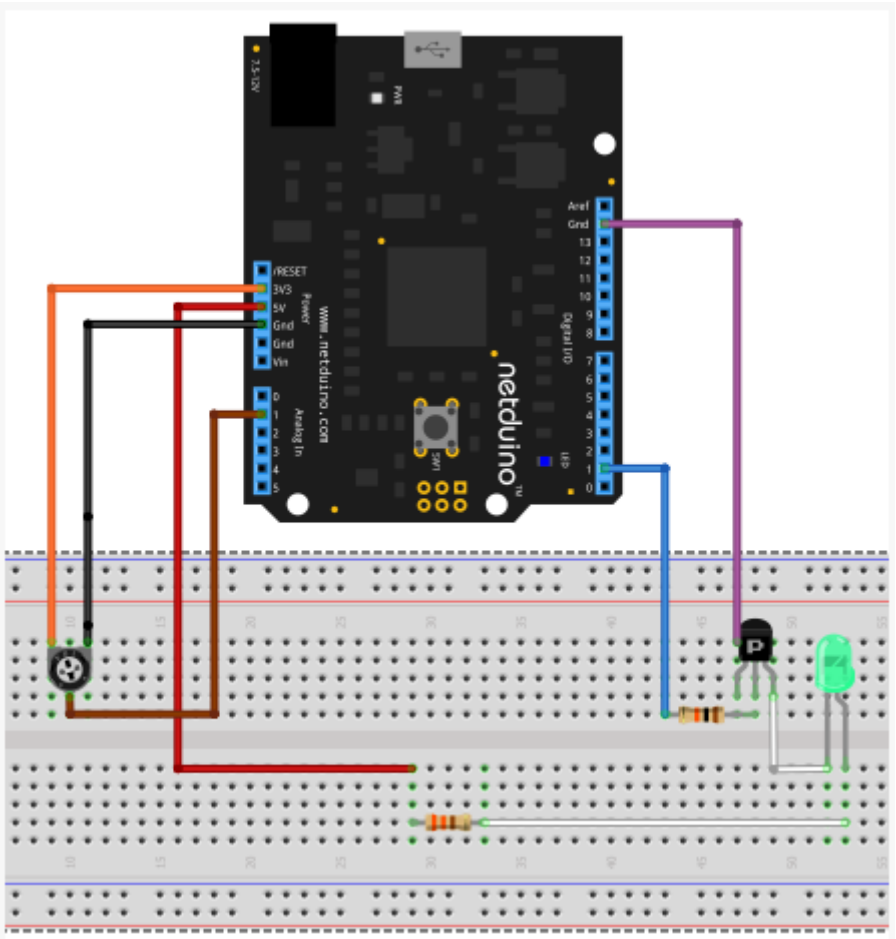
Hình P.28: Sơ đồ kết nối *Bo mạch Netduino Plus* với màn hình *HD44780U*

```
using System;
using System.Threading;
using Microsoft.SPOT.Hardware;
using System.Text;
namespace EmbeddedLab.NetduinoPlus.Day2.Display
{
    public class LCDProcess
    {
        Constructor


        #region Public Methods
        public void Show(string text, int delay, bool newLine)...

        public void Show(string text)...


        public void ClearDisplay()
        {
            SendCommand((byte)Command.Clear);
            currentRow = 0;
            dirtyColumns = 0;
        }
        public void GoHome()...
        public void JumpAt(byte column, byte row)
        {
            if (NumberOfLines == (byte)Operational.DoubleLIne) row = (byte)(row % 4);
            else row = (byte)(row % 2);

            SendCommand((byte)((byte)Command.SetDdRam | (byte)(column + rowAddress[row])));
        }
```

Hình P.29: Một phần mã nguồn chương trình *Netduino_LCD*

P.2.5.3. Chương trình thao tác với cổng nối tiếp *Netduino_SerialPort*

Chương trình *Netduino_SerialPort* cung cấp một lớp thư viện làm việc với các cổng nối tiếp của Bo mạch Netduino và các thiết bị vào ra. Chương trình cũng được viết bằng ngôn ngữ C# với .Net Micro Framework giống như chương trình *Netduino_8digit* và *Netduino_LCD*. Hình P.30 minh họa sử dụng lớp thư viện để điều khiển điốt nhấp nháy. Hình P.31 minh họa một phần mã nguồn của chương trình.



Hình P.30: Sơ đồ ghép nối *Bo mạch Netduino* điều khiển điốt

```
namespace SerialPortLearn
{
    public class SerialPortProcess
    {
        static SerialPort serialPort;

        const int bufferMax = 1024;
        static byte[] buffer = new Byte[bufferMax];
        static int bufferLength = 0;

        public SerialPortProcess(string portName = SerialPorts.COM2, int baudRate = 9600,
            Parity parity = Parity.None, int dataBits = 8, StopBits stopBits = StopBits.One)...

        private void serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            lock (buffer)
            {
                int bytesReceived = serialPort.Read(buffer, bufferLength, bufferMax - bufferLength);
                if (bytesReceived > 0)
                {
                    bufferLength += bytesReceived;
                    if (bufferLength >= bufferMax)
                        throw new ApplicationException("Buffer Overflow.  Send shorter lines, or incre
                }
            }
        }
```

Hình P.31: Một phần mã nguồn chương trình *Netduino_SerialPort*

P.2.6. Các chương trình nhúng cho vi xử lý MIPS

Ngoài các chương trình thử nghiệm đã trình bày trong các phần trước, chúng tôi cũng sử dụng các chương trình nhúng mức thấp được biên dịch chéo, nạp và chạy trực tiếp trên phần cứng hệ thống nhúng mà không thông qua máy ảo hoặc hệ điều hành. Các chương trình thử nghiệm trong phần này có mã nguồn C, được biên dịch chéo sang mã hợp ngữ MIPS, được thi trong công cụ mô phỏng *SimpleScalar* và được tổng hợp trong Bảng P.2.

Bảng P.2. Tổng hợp các chương trình thử nghiệm cho MIPS

STT	Chương trình	Mô tả
1	Fibonacci	Chương trình hiển thị chuỗi số Fibonacci trên màn hình
2	Sum N numbers	Chương trình tính tổng của N số nguyên
3	ArraySum	Chương trình tính tổng các số thực trong một mảng
4	Quick Sort	Chương trình sắp xếp nhanh
5	Bubble Sort	Chương trình sắp xếp chuỗi số theo phương pháp nổi bọt
6	Binary Search	Chương trình tìm kiếm nhị phân
7	Hanoi	Chương trình tháp Hà Nội với giao diện dòng lệnh
8	Permutation	Chương trình hoán vị chuỗi số
9	Matrix Multiply	Chương trình nhân hai ma trận

BẢNG CHỈ MỤC

J

K

L

M

N

O

P

R

S

T

U

X