Kỷ yếu Hội nghị Quốc gia lần thứ X về Nghiên cứu cơ bản và ứng dụng Công nghệ thông tin (FAIR), Đà Nẵng, ngày 17-18/08/2017
DOI: 10.15625/vap.2017.00065

MỘT PHƯƠNG PHÁP CẢI THIỆN HIỆU NĂNG CÁC ỨNG DỤNG TRONG ANDROID TRÊN CÁC CHIP ĐA NHÂN

Bùi Hữu Phúc[1, 2], Phạm Văn Hưởng[3], Nguyễn Ngọc Bình[1,*]

[1] Trường ĐH Công nghệ, ĐHQGHN, [2] Học viện An ninh Nhân dân, [3] Học viện Kỹ thuật Mật mã

phucbh.hvan@gmail.com, huongpv@gmail.com, nnbinh@vnu.edu.vn

TÓM TẮT: *Trong xu hướng phát triển mạnh mẽ của các thiết bị Android, đặc biệt là thiết bị Android có bộ xử lý đa nhân, vấn đề cải tiến mã nguồn để phát huy khả năng xử lý song song đang là một thách thức đặt ra. Bài báo này đề xuất phương pháp cải thiện hiệu năng cho các ứng dụng nhúng Android trên bộ xử lý đa nhân dựa trên kỹ nghệ ngược, biến đổi và thay thế mã nguồn. Từ các ứng dụng Android đã có, chúng tôi dịch ngược thành mã nguồn Java, sau đó thực hiện phân tích để tìm các hàm tác động chính đến hiệu năng theo luật Pareto 8/2 và thực hiện chuyển đổi mã nguồn Java từ tuần tự sang song song để cải thiện hiệu năng.*

Từ khóa: *Tối ưu phần mềm nhúng, kỹ nghệ ngược, hiệu năng, thiết bị Android, vi xử lý đa nhân.*

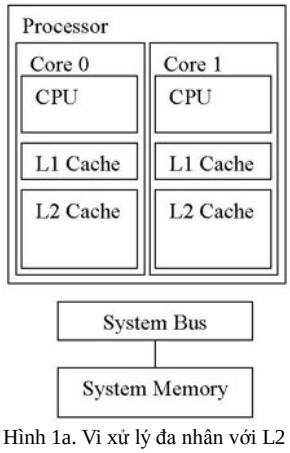
I. GIỚI THIỆU

Ngày nay, thiết bị thông minh đã trở lên quen thuộc và là một phần tất yếu của cuộc sống. Các thiết bị thông minh có mặt trong nhiều lĩnh vực kinh tế - xã hội như y tế, giáo dục, văn phòng, kinh doanh, giải trí, v.v. Với sự phát triển mạnh mẽ của công nghệ phần cứng, bộ vi xử lý đa nhân đã được dùng cho các thiết bị thông minh và đang trở lên phổ biến. Theo đó, việc tối ưu tốc độ, tiết kiệm điện năng cho phần mềm trên thiết bị thông minh cũng đang được quan tâm nghiên cứu, phát triển. Đây cũng là điểm nhấn để các công ty phần mềm tăng cường khả năng cạnh tranh. Hiện tại, Android là hệ điều hành được sử dụng phổ biến hơn trên các thiết bị thông minh so với iOS và Windows Phone. Việc phát triển phần mềm trên nền tảng Android đã phổ biến nhưng hầu hết chưa tận dụng được khả năng của bộ vi xử lý đa nhân. Bài toán tối ưu phần mềm Android trên bộ vi xử lý đa nhân còn ít được nghiên cứu. Mặt khác, hầu hết các ứng dụng Android đã đóng gói và lưu trữ trên trên các kho ứng dụng của Google đều được lập trình cho mô hình đơn CPU. Vấn đề tối ưu hóa các ứng dụng Android đóng gói cho các vi xử lý đa nhân vẫn đang là một thách thức đặt ra. Để giải quyết vấn đề này, bài báo đưa ra cách tiếp cận mới để cải thiện hiệu năng các ứng dụng đóng gói cho vi xử lý đa nhân dựa trên kỹ nghệ ngược, chuyển đổi và thay thế mã nguồn.

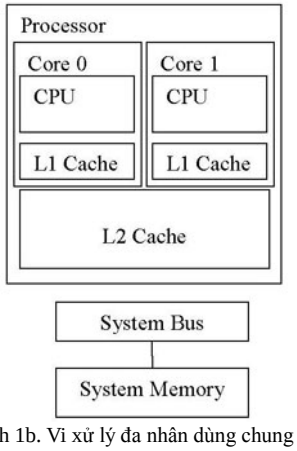
Trong bài báo, chúng tôi tập trung nghiên cứu các ứng dụng Android, vi xử lý đa nhân trong thiết bị chạy hệ điều hành Android, dịch ngược mã bytecode và tối ưu dựa trên song song hóa. Nội dung còn lại của bài báo được tổ chức như sau: *Mục 2* tổng hợp, phân tích các nghiên cứu liên quan; *Mục 3* trình bày về ý tưởng đề xuất và các bước phát triển phương pháp tối ưu; *Mục 4* trình bày về thực nghiệm và đánh giá; *Mục 5* kết luận và hướng phát triển.

II. CÁC NGHIÊN CỨU LIÊN QUAN

Vấn đề tối ưu trong phát triển phần mềm trên các hệ thống nhúng, các thiết bị thông minh với vi xử lý đa nhân ngày càng được nghiên cứu và phát triển rộng rãi. Trong các nghiên cứu [9, 15], tác giả đã tổng hợp về kiến trúc của vi xử lý đa nhân, hệ thống nhúng đa nhân và vấn đề tối ưu phần mềm trong hệ nhúng đa nhân. Nói chung, các bộ vi xử lý đa nhân có kiến trúc chia sẻ hoặc không chia sẻ bộ nhớ đệm L2 như trong Hình 1 và các hệ nhúng đa nhân cũng có hai dạng kiến trúc phổ biến là đồng nhất và không đồng nhất.



Hình 1a. Vi xử lý đa nhân với L2 riêng



Hình 1b. Vi xử lý đa nhân dùng chung L2

Hình 1. Các bộ vi xử lý đa nhân

[*] A Visiting Professor at Hosei University, Tokyo, Japan.

Trên cơ sở đó, bài toán tối ưu hệ thống nhúng đa nhân cũng chia thành ba loại: 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. Vấn đề tối ưu phần mềm trong các hệ thống nhúng đa nhân bao gồm các mục tiêu tối ưu chính như tối ưu hiệu năng [1, 3-7, 11], điện năng tiêu thụ [2, 10, 12, 13, 14], tối ưu đa mục tiêu [8], v.v. và có thể thực hiện ở các mức khác nhau như mức mô hình, mức mã nguồn, mức mã máy ảo, mức hợp ngữ và mã máy. Đồng thời, bài toán tối ưu phần mềm trong hệ nhúng, thiết bị thông minh đa nhân thường tập trung giải quyết các thách thức: phân chia chức năng tổng thể thành các tác vụ song song, phân phối tác vụ vào các nhân rỗi, lập lịch tác vụ, và quản lý liên kết nội [9].

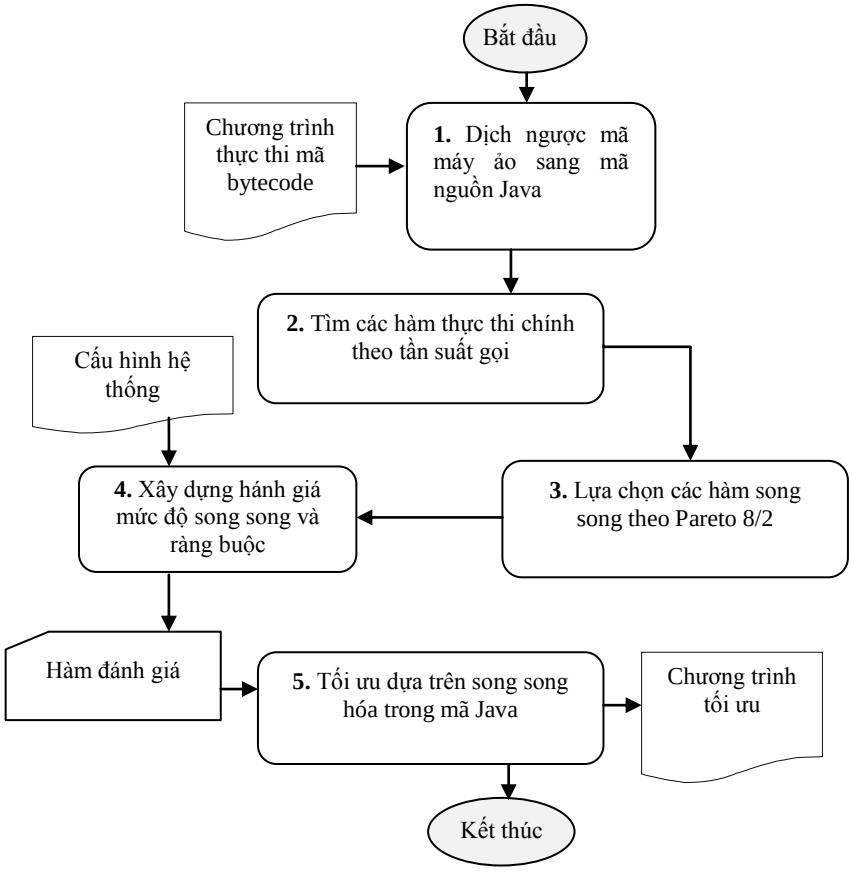
Kết quả phân tích và tổng hợp các nghiên cứu liên quan cho thấy bài toán tối ưu phần mềm trong các hệ thống nhúng, các thiết bị Android đa nhân chủ yếu tập trung vào việc song song hóa mã nguồn hoặc dữ liệu, phân bổ tác vụ, lựa chọn cấu hình, phân cấp bộ nhớ. Tuy nhiên, các nghiên cứu này, không thể áp dụng để tối ưu hóa các ứng dụng đã được biên dịch, đóng gói và cũng chưa nghiên cứu về sự tương quan giữa mã máy ảo và mã máy. Do đó, phương pháp đề xuất trong bài báo sẽ kết hợp kỹ nghệ ngược với các phương pháp tối ưu đã có để tối ưu hóa phần mềm đóng gói trong các thiết bị Android đa nhân. Nghiên cứu trong bài báo cũng tập trung vào sự tương quan giữa mã máy ảo, mã máy và lựa chọn các hàm để tối ưu theo luật Pareto 8/2.

III. PHÁT TRIỂN PHƯƠNG PHÁP

3.1. Ý tưởng và mô hình tổng thể

Chúng tôi đề xuất và phát triển phương pháp tối ưu này dựa trên ý tưởng kết hợp việc thực đồng thời cả mã bytecode trên máy ảo Java và mã máy được biên dịch từ mã nguồn Java trong một ứng dụng Android cùng với việc song song hóa mã Java cho các vi xử lý đa nhân. Mô hình tối ưu tổng thể được chỉ ra trong Hình 2. Đầu tiên, chúng tôi dịch ngược mã bytecode của các ứng dụng Android sang mã nguồn Java; phân tích mã nguồn Java để tìm các hàm xử lý chính dựa trên tần suất gọi hàm và số vòng lặp, số dòng lệnh trong hàm; xây dựng hàm đánh giá để thực hiện việc song song hóa mã nguồn Java; tối ưu dựa trên song song hóa mã nguồn JavaViệc đánh giá, lựa chọn các hàm xử lý chính dựa trên luật Pareto 8/2 [16]. Việc xây dựng hàm đánh giá dựa trên phân tích cấu hình hệ thống và mã nguồn Java.

.



Hình 2. Mô hình tối ưu tổng thể

3.2. Dịch ngược mã bytecode sang mã Java

Kỹ nghệ ngược là định hướng nghiên cứu có phạm vi ứng dụng rộng rãi trong nhiều lĩnh vực như tối ưu, tái kỹ nghệ phần mềm, an toàn thông tin, v.v. Kỹ nghệ ngược có thể thực hiện theo các mức khác nhau từ mã máy sang mã hợp ngữ, từ mã hợp ngữ sang mã nguồn mức cao, từ mã nguồn mức cao sang mô hình, từ mã máy sang mã máy ảo, v.v. Trong

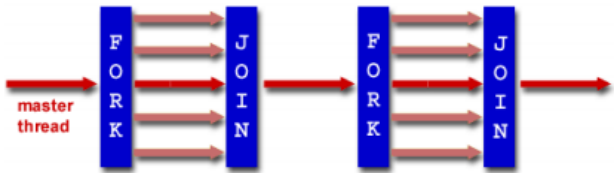
phạm vi nghiên cứu của bài báo, chúng tôi không nghiên cứu các kỹ thuật dịch ngược mà chỉ sử dụng công cụ dịch ngược mã byte sang mã nguồn Java. Mã nguồn Java này được sử dụng làm đầu vào cho bài toán tối ưu trong *Mục 3.4*. Trong bài báo, chúng tôi sử dụng công cụ *AndroChef Java Decompiler* để dịch ngược mã bytecode sang mã Java.

3.3. Lựa chọn các hàm để tối ưu theo luật Pareto 8/2

Sau khi dịch ngược mã bytecode sang mã nguồn Java, chúng tôi sử dụng công cụ *Code Analysis Tool (CAST)* để phân tích nhằm tìm ra các hàm tác động chính đến hiệu năng và thực hiện tối ưu trên các hàm này. Theo luật Pareto 8/2 [16], chỉ 20% các hàm thực thi chính sẽ ảnh hưởng đến 80% hiệu năng hệ thống. Luật này đã áp dụng thành công trong nhiều lĩnh vực khoa học, kinh tế, xã hội. Do đó, chúng tôi cũng dựa trên ý tưởng này để tìm 20% hàm có số lần được gọi nhân với kích thước hàm lớn nhất.

3.4. Mô hình bài toán và điều kiện song song

20% các hàm Java tác động chính đến hiệu năng sẽ được chuyển đổi mã nguồn theo mô hình lập trình song song để cải tiến hiệu năng theo mô hình Fork – Join như trong Hình 3 [15].



Hình 3. Mô hình Fork – Join

Như mô tả trong Hình 3, để thực hiện việc song song hóa cần thêm thời gian phân chia và đồng bộ các luồng. Do đó, trong một số trường hợp, thực thi song song còn làm hiệu năng kém đi. Vấn đề thực hiện song song hóa khi nào và sử dụng bao nhiêu luồng để cải tiến hiệu năng vẫn đang là thách thức đặt ra. Theo khảo sát trong bài báo, hiện vẫn chưa có nghiên cứu nào giải quyết vấn đề này. Hơn nữa, do việc đánh giá hiệu năng cũng như điện năng tiêu thụ dựa trên mã nguồn mức cao cũng không chính xác tuyệt đối. Để giải quyết vấn đề này, chúng tôi mô hình hóa hệ thống và xây dựng công thức để xác định điều kiện thực hiện việc song song hóa và số luồng thực thi song song với giả thiết là các luồng thực thi song song cân bằng nhau, được tải đều trên các nhân của vi xử lý. Đồng thời, phạm vi nghiên cứu trong bài báo cũng tập trung vào song song hóa khi thực thi các cấu trúc lặp.

Giả sử bộ vi xử lý hệ thống có N_c nhân, tần số xung nhịp của mỗi nhân là f_x, số xung nhịp trên 1 chu kỳ đồng hồ là f_s, mỗi lệnh được thực hiện trung bình trong C chu kỳ đồng hồ, thời gian truy xuất 1 byte nhớ là T_M. Với mỗi cấu trúc lặp trong hàm, giả sử độ phức tạp (số bước lặp) là M và số câu lệnh trong cấu trúc lặp là N_s. Theo giả thiết này, số lần thực hiện 1 câu lệnh sau khi hoàn thành vòng lặp được đánh giá như công thức (1).

$$S = M \times N_s \tag{1}$$

Thời gian thực thi 1 câu lệnh T_s có thể được đánh giá theo công thức (2). Do đó, tổng thời gian hoàn thành vòng lặp khi thực hiện trên 1 nhân T_1 được đánh giá theo công thức (3).

$$T_s = (1/f_x) \times f_s \times C = (f_s \times C)/f_x \tag{2}$$

$$T_1 = M \times N_s \times f_s \times C / f_x \tag{3}$$

Gọi N_t là số luồng thực thi song song. Khi thực hiện đa luồng, cần thêm thời gian phân luồng và tổng hợp kết quả trong bộ đệm, nên có thể đánh giá thời gian phân luồng và đồng bộ theo thời gian truy xuất bộ nhớ. Theo đó, tổng thời gian hoàn thành vòng lặp theo đa luồng T_2 có thể được đánh giá theo công thức (4).

$$T_2 = N_t \times T_M + T_1/N_t \tag{4}$$

Từ (3) và (4) suy ra điều kiện để thực hiện đa luồng và số luồng song song phải thỏa mãn hệ bất phương trình (5).

$$\begin{cases} T_2 < T_1 \\ N_t <= N_c \end{cases} \tag{5}$$

3.5. Chuyển đổi mã nguồn theo mô hình song song

Sau khi xác định được điều kiện song song hóa và số luồng, chúng tôi tiến hành chuyển đổi mã nguồn Java từ đơn luồng sang đa luồng để cải tiến hiệu năng cho hệ thống đa nhân. Điều kiện chuyển đổi mã nguồn được thực hiện theo công thức (5). Kết quả minh họa như trong Hình 4.

Mã nguồn tuần tự (D)

```
int var5 = MainActivity.this.rows;
int var6 = MainActivity.this.cols;
int var7 = MainActivity.this.rows;

for(int var2=0; var2<var5; ++var2){
   for(int var3=0; var3<var7; ++var3){
      for(int var4=0; var4<var6; ++var4){
         Double[] var8 = MainActivity.this.matrixC[var2];
         var8[var3] += MainActivity.this.matrixA[var2][var4]
         * MainActivity.this.matrixB[var4][var3];
      }
   }
}
```

Mã nguồn song song (D')

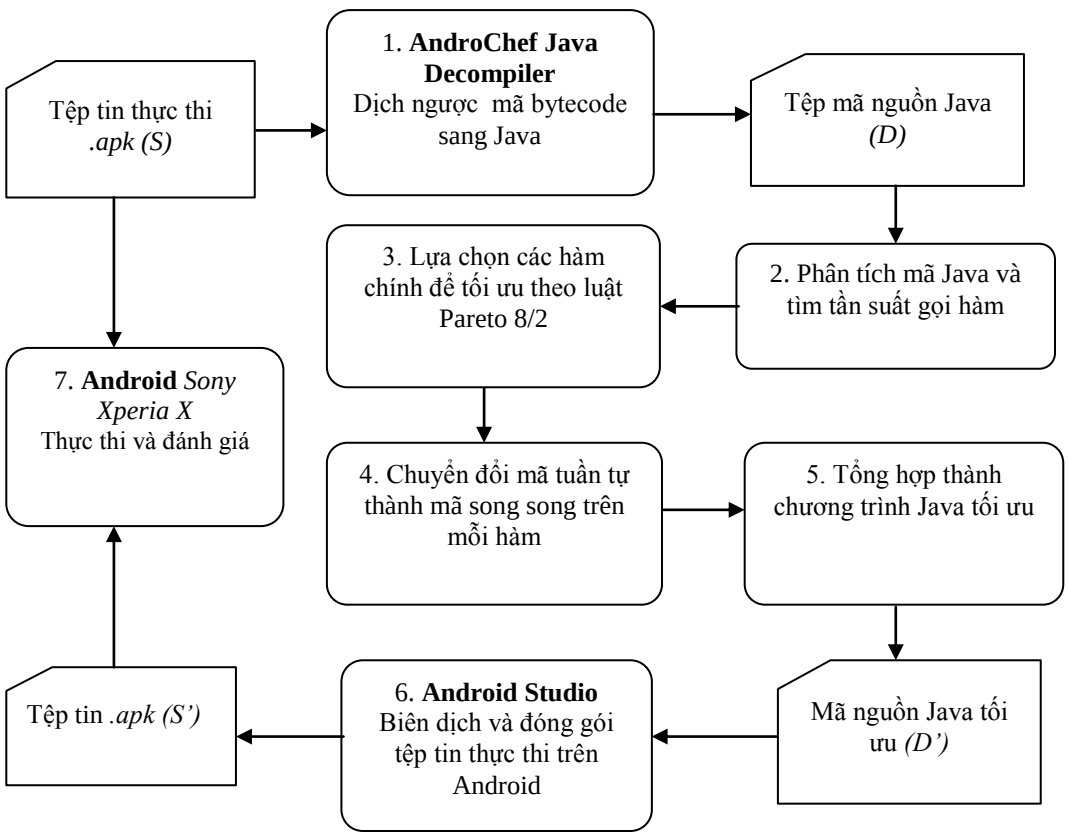
```
@Override
protected void doInBackground(void ... params){
   for(int i=0; i < B.length; ++i){
      C[row][col] += A[row][i] * B[i][col];
   }

   for(int i=0; i < rows; ++i){
      for(int j=0; j < cols; ++j){
         async[runTasks] = new MultThread(i, j, matrixA,
         matrixB, matrixC, getBaseContext());
         runTasks++;
      }
   }
}
```

Hình 4. Minh họa chuyển đổi mã nguồn tuần tự sang song song

IV. THỰC NGHIỆM

4.1. Mô hình, dữ liệu, môi trường và kết quả thực nghiệm

Để đánh giá hiệu quả của phương pháp tối ưu, chúng tôi triển khai mô hình thực nghiệm như trong Hình 5, với bộ chương trình thử nghiệm được mô tả trong Bảng 1 trên môi trường thực nghiệm như trong Bảng 2. Đầu tiên từ một tệp tin thực thi *S* ban đầu, chúng tôi sử dụng phần *mềm AndroChef Java Decompiler* để dịch ngược sang mã Java (1); thống kê lời gọi hàm và kích thước hàm (2); áp dụng luật Pareto 8/2 để chọn tập các hàm tác động chính đến hiệu năng và chọn lọc các hàm thỏa mãn ràng buộc để thực hiện đa luồng (3); chuyển đổi mã nguồn đơn luồng thành đa luồng và tổng hợp thành chương trình Java tối ưu (4, 5); thực hiện biên dịch lại được tệp tin *S'* trong *Android Studio* (6); thực thi chương trình *S* ban đầu và chương trình tối ưu *S'* cùng trên thiết bị *Android Sony Xperia X* để đánh giá (7); kết quả thực nghiệm được thống kê trong Bảng 3.



Hình 5. Mô hình thực nghiệm

Bảng 1. Bộ chương trình thử nghiệm

STT	Tên chương trình	Mô tả
01	Nhân ma trận 900x200	Nhân ma trận A [900x200] với ma trận B [200x900]
02	Nhân ma trận 800x300	Nhân ma trận A [800x300] với ma trận B [300x800]
03	Nhân ma trận 750x350	Nhân ma trận A [750x350] với ma trận B [350x750]
04	Nhân ma trận 700x400	Nhân ma trận A [700x400] với ma trận B [400x700]
05	Nhân ma trận 600x500	Nhân ma trận A [600x500] với ma trận B [500x600]
06	Nhân ma trận 350x850	Nhân ma trận A [350x850] với ma trận B [850x350]
07	Nhân ma trận 850x250	Nhân ma trận A [850x250] với ma trận B [250x850]
08	Nhân ma trận 550x600	Nhân ma trận A [550x600] với ma trận B [600x550]
09	Nhân ma trận 450x700	Nhân ma trận A [450x700] với ma trận B [700x450]
10	Nhân ma trận 650x450	Nhân ma trận A [650x450] với ma trận B [450x650]

Bảng 2. Môi trường thực nghiệm

Thiết bị	Sony Xperia X
CPU	Snapdragon 650 6 nhân 64-bit
Số nhân	6 nhân
Bộ nhớ trong	64GB
Bộ nhớ đệm	3GB
Hệ điều hành	Android 7.0

Bảng 3. Kết quả thực nghiệm

STT	Chương trình	Thời gian thực thi trung bình (mS)	
		Phiên bản gốc (S)	Phiên bản tối ưu (S')
01	Nhân ma trận 900x200	5060	255
02	Nhân ma trận 800x300	5407	303
03	Nhân ma trận 750x350	5385	352
04	Nhân ma trận 700x400	5686	354
05	Nhân ma trận 600x500	5427	512
06	Nhân ma trận 500x600	3423	499
07	Nhân ma trận 400x700	5955	407
08	Nhân ma trận 350x850	4987	355
09	Nhân ma trận 300x800	4532	362
10	Nhân ma trận 200x900	5013	513

4.2. Đánh giá kết quả thực nghiệm

Như kết quả thực nghiệm tổng hợp trong Bảng 3 Mục 4.2, phương pháp tối ưu đề xuất rút ngắn được khoảng 85% thời gian thực hiện. So với một số phương pháp tối ưu khác phương pháp này tốt hơn về mặt hiệu năng nhưng chưa có ràng buộc chặt chẽ về mặt điện năng tiêu thụ. Ưu điểm của phương pháp là xác định được điều kiện song song và áp dụng cho các phần mềm đã biên dịch và đóng gói.

V. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Phương pháp tối ưu đề xuất trong bài báo theo một cách tiếp cận mới, kết hợp kỹ nghệ ngược và chuyển đổi mã nguồn từ mô hình tuần tự sang song song để cải thiện hiệu năng phần mềm trên thiết bị, hệ nhúng đa nhân. Điểm mới của phương pháp là xác định rõ số luồng, điều kiện song song hóa và được áp dụng cho các phần mềm đã biên dịch, đóng gói. Một ưu điểm nữa của phương pháp này là có thể áp dụng cho các chương trình Android đã đóng gói với mã tuần tự biên dịch trên chip đơn nhân. Tuy nhiên, vấn đề dịch ngược không thể áp dụng cho các chương trình Android sử dụng kỹ thuật làm rối mã, mã hóa mã nguồn khi biên dịch và độ chính xác của mã nguồn phụ thuộc vào công cụ dịch ngược. Mặc dù đạt được kết quả khích lệ nhưng phương pháp này vẫn còn một số hạn chế như: chưa gắn với ràng buộc về điện năng tiêu thụ, chỉ song song hóa mã nguồn, tập trung vào các cấu trúc lặp, chưa thực hiện song song hóa dữ liệu và việc song song hóa mã nguồn chưa được thực hiện tự động. Ngoài ra, khi tái biên dịch lại mã bytecode, kích thước chương trình tối ưu có thể lớn hơn kích thước chương trình ban đầu.

Trên cơ sở kết quả đạt được và hạn chế của bài báo, trong các nghiên cứu tiếp theo, chúng tôi tiếp tục nghiên cứu, phát triển phương pháp tối ưu theo điện năng tiêu thụ, đa mục tiêu và tối ưu dựa trên song song hóa dữ liệu. Đồng thời, chúng tôi cũng nghiên cứu, khắc phục vấn đề gia tăng kích thước và phát triển phương pháp tối ưu đa mục tiêu - cân bằng giữa hiệu năng và kích thước chương trình.

TÀI LIỆU THAM KHẢO

[1] Astrụm Fransson, D., "Utilizing Multicore Processors with Streamed Data Parallel Applications for Mobile Platforms" (2012:100), Technical report, KTH, Electronic, Computer and Software Systems, ECS, 43, 2012.

[2] Couto, M.; Cunha, J. & Fernandes, J. P. (2015), GreenDroid: A Tool for Analysing Power Consumption in the Android Ecosystem, in 'Proceedings of the 13th International Conference Informatics 2015', pp. 73-78, 2015.

[3] Dhuha Basheer Abdullah, M. M. A. H. (2013), "Developing Parallel Application on Multi-core Mobile Phone", International Journal of Advanced Computer Science and Applications (IJACSA) **4** (11).

[4] Fauberteau, F.; Midonnet, S. & Qamhieh, M. (2011), "Partitioned Scheduling of Parallel Real-time Tasks on Multiprocessor Systems", SIGBED Rev. **8**(3), 28-31.

[5] Feljan, J. & Carlson, J. (2014), Task Allocation Optimization for Multicore Embedded Systems, in "The 40th Euromicro Conference on Software Engineering and Advanced Applications".

[6] Guo, Y.; Xu, Y. & Chen, X. (2017), Freeze It If You Can: Challenges and Future Directions in Benchmarking Smartphone Performance, in "Proceedings of the 18th International Workshop on Mobile Computing Systems and Applications", ACM, New York, NY, USA, pp. 25-30.

[7] Hu, R. X.; Hu, W.; Zuo, Z. Y.; Wang, M.; Xu, J. & Lv, Y. K. (2013), A Novel Optimization Approach Based on Scratchpad Memory for Mobile LBS, in "Manufacturing Engineering and Process II", Trans Tech Publications, pp. 935-938.

[8] Hussein, A.; Payer, M.; Hosking, A. & Vick, C. A. (2015), Impact of GC Design on Power and Performance for Android, in "Proceedings of the 8th ACM International Systems and Storage Conference", ACM, New York, NY, USA, pp. 13:1-13:12.

[9] Kornaros, G. (2010), Multi-Core Embedded Systems, CRC Press, Inc., Boca Raton, FL, USA.

[10] Naik, B. A. & Chavan, R. (2015), "Article: Optimization in Power Usage of Smartphones", International Journal of Computer Applications **119**(18), 7-13.

[11] Nogueira, L. & Pinho, L. M. (2012), Server-based Scheduling of Parallel Real-time Tasks, in "Proceedings of the Tenth ACM International Conference on Embedded Software", ACM, New York, NY, USA, pp. 73-82.

[12] Rao, K.; Wang, J.; Yalamanchili, S.; Wardi, Y. & Ye, H. (2017), "Application-Specifc Performance-Aware Energy Optimization on Android Mobile Devices", 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA).

[13] Spiliopoulos, V. (2016), "Improving Energy-Efficiency of Multicores using First-Order Modeling", PhD thesis, Uppsala UniversityUppsala University, Computer Architecture and Computer Communication, Division of Computer Systems.

[14] Tseng, P. H.; Hsiu, P. C.; Pan, C. C. & Kuo, T. W. (2014), User-Centric Energy-Efficient Scheduling on Multi-Core Mobile Devices, in "Proceedings of the 51st Annual Design Automation Conference", ACM, New York, NY, USA, pp. 85:1-85:6.

[15] Uchiyama, K.; Arakawa, F.; Kasahara, H.; Nojiri, T.; Noda, H.; Tawara, Y.; Idehara, A.; Iwata, K. & Shikano, H. (2012), Heterogeneous multicore processor technologies for embedded systems, Springer New York.

[16] Bunkley, N. (2008), "Joseph Juran, 103, Pioneer in Quality Control, Dies", New York Times.

A METHOD OF OPTIMIZING PERFORMANCE FOR ANDROID APPLICATIONS ON MULTICORE PROCESSORS

Bui Huu Phuc, Pham Van Huong, Nguyen Ngoc Binh

ABSTRACT: *In the growing trend of Android devices, especially those with multi-core processors, the problem of improving the source code in order to promote parallel processing becomes a real challenge. This paper proposes a method for optimizing the performance of Android applications on the multi-core processors based on reverse engineering, source code transformation and replacement techniques. From the existing Android applications, we decompile into Java source code before performing an analysis to identity the functions which have the main effect to the performance according to Pareto 8/2 rule, then convert Java source code from sequential to parallel for optimization.*

Keywords: *Embedded software optimization, reverse engineering, performance, Android equipment, multi-core processors.*