Tạp chí Phát triển Khoa học và Công nghệ – Kĩ thuật và Công nghệ, 2(4):277-287

Open Access Full Text Article

Bài nghiên cứu

Các tiếp cận song song của giải thuật di truyền trên kiến trúc MIC của bộ đồng xử lý Intel Xeon Phi

Nguyễn Quang Hùng*, Trần Ngọc Anh Tú, Thoại Nam

Use your smartphone to scan this QR code and download this article

TÓM TẮT

Ngày nay, giải thuật di truyền được sử dụng phổ biến trong nhiều ngành như tin sinh học, khoa học máy tính, trí tuệ nhân tạo, tài chính… Giải thuật di truyền được áp dụng nhằm tạo ra lời giải chất lượng cao cho các bài toán tối ưu phức tạp trong các ngành trên. Đã có nhiều nghiên cứu dựa trên kiến trúc phần cứng mới được đề nghị với mục đích tăng tốc độ thực thi giải thuật di truyền càng nhanh càng tốt. Một số nghiên cứu đề xuất các giải thuật di truyền song song trên các hệ thống có bộ xử lý đa nhân (multicore CPU) và/hoặc có các bộ xử lý đồ họa (Graphics Processing Unit - GPU). Tuy nhiên, rất ít giải pháp đề xuất giải thuật di truyền có thể được chạy trên các hệ thống có sử dụng các bộ đồng xử lý (co-processor) mới Intel Xeon Phi (Intel Xeon Phi có kiến trúc Many Intergrated Core (MIC)). Vì lý do đó, chúng tôi đề xuất và phát triển giải pháp hiện thực giải thuật di truyền trên kiến trúc MIC của Intel Xeon Phi. Nghiên cứu này sẽ trình bày các tiếp cận song song giải thuật di truyền trên một và nhiều bộ đồng xử lý Intel Xeon Phi theo các phương pháp gồm: (i) mô hình lập trình Intel Xeon Phi dạng Offload và Native; và (ii) mô hình kết hợp MPI và OpenMP. Giải thuật di truyền đề xuất để tìm lịch tối ưu cho bài toán lập lịch của các máy ảo lên các máy vật lý với mục tiêu tối ưu năng lượng tiêu thụ. Các kết quả đánh giá bằng mô phỏng cho thấy tính khả thi của việc hiện thực giải thuật di truyền trên một hoặc phân bố trên nhiều Intel Xeon Phi. Giải thuật di truyền trên một hay phân bố trên nhiều Intel Xeon Phi luôn cho kết quả về thời gian thực thi giải thuật nhanh hơn thực thi giải thuật tuần tự và khả năng tìm ra lời giải tốt hơn nếu sử dụng nhiều Intel Xeon Phi hơn. Kết quả nghiên cứu này có thể được áp dụng cho các meta-heuristic khác như tìm kiếm TABU, Ant Colony Optimization.

Từ khoá: Xử lý song song, Genetic algorithm, Xeon Phi, MIC, Meta-heuristic

Khoa Khoa học và Kỹ thuật Máy tính, Trường Đại học Bách khoa, ĐHQG-HCM, Việt Nam

Liên hệ

Nguyễn Quang Hùng, Khoa Khoa học và Kỹ thuật Máy tính, Trường Đại học Bách khoa, ĐHQG-HCM, Việt Nam

Email: nqhung@hcmut.edu.vn

Lịch sử

- Ngày nhận: 09-10-2019
- Ngày chấp nhận: 20-11-2019
- Ngày đăng: 31-12-2019

DOI : 10.32508/stdjet.v2i4.612

Check for updates

VNU-HCM Press

GIỚI THIỆU

Trong những năm gần đây, sự phát triển mạnh mẽ của lĩnh vực Tính Toán Hiệu Năng Cao (High-Performance Compting - HPC) với sự xuất hiện của các siêu máy tính cấu hình mạnh có tốc độ tính toán rất cao. Hầu hết các siêu máy tính đều tận dụng sức mạnh của các bộ xử lý trung tâm (CPU) đa nhân (multicore), thiết bị gia tốc (accelerator) chẳng hạn bộ xử lý Đồ họa (Graphics Processing Unit - GPU) hoặc bộ đồng xử lý (co-processor) Intel Xeon Phi[1]. Hầu hết các nhà nghiên cứu và các lập trình viên thực thi các ứng dụng của họ trên các hệ thống dùng CPU đa nhân và/hoặc trang bị thêm các GPU[2]. Hiện tại, có ít ứng dụng chạy trên các hệ thống được trang bị phần cứng mới là bộ đồng xử lý Intel Xeon Phi có kiến trúc Many Intergrated Core (MIC). Các hệ thống siêu máy tính hàng đầu được trang bị bộ đồng xử lý Intel Xeon Phi khá nhiều: Tiane-2 (MilkyWay-2), Thunder, cascade, SuperMUC…. Hệ quả là không có nhiều ứng dụng có thể chạy được trên các hệ thống siêu máy tính như Tiane-2 (MilkyWay-2), Thunder, cascade, SuperMUC này nói chung; và đặc biệt là SuperNode-XP - hệ thống tính toán hiệu năng cao gồm 24 nút tính toán mạnh trang bị các bộ đồng xử lý Intel Xeon Phi tại trường Đại học Bách khoa, ĐHQG-HCM nói riêng bởi vì hệ thống SuperNode-XP này sử dụng bộ đồng xử lý Intel Xeon Phi. Điều này dẫn đến việc rất khó để tận dụng hoàn toàn sức mạnh tính toán của các siêu máy tính. Do vậy, động cơ cho nghiên cứu này nhằm hiện thực và đánh giá hiệu năng của meta-heuristic (ví dụ: giải thuật di truyền, giải thuật tìm kiếm TABU, Ant Colony Optimization, …) trên các hệ thống tính toán hiệu năng cao có trang bị các bộ đồng xử lý Intel Xeon Phi là cần thiết và cấp bách. Giải thuật di truyền (Genetic Algorithm - GA) là một giải thuật có khối lượng tính toán lớn nhưng có thể song song hóa đạt hiệu suất cao. Với mục tiêu tìm thấy lời giải tốt nhất và giảm thời gian thực thi các giải thuật di truyền, bộ đồng xử lý Intel Xeon Phi hoặc GPU thường được sử dụng để đạt được mục đích đó.

Bài báo này sẽ trình bày tổng kết kết quả nghiên cứu song song hóa giải thuật di truyền trên một và nhiều bộ đồng xử lý Intel Xeon Phi. Phương pháp của chúng tôi sử dụng: (i) mô hình kết hợp MPI và OpenMP; và

Trích dẫn bài báo này: Quang Hùng N, Anh Tú T N, Nam T. **Các tiếp cận song song của giải thuật di truyền trên kiến trúc MIC của bộ đồng xử lý Intel Xeon Phi**. *Sci. Tech. Dev. J. - Eng. Tech.;* 2(4):277-287.

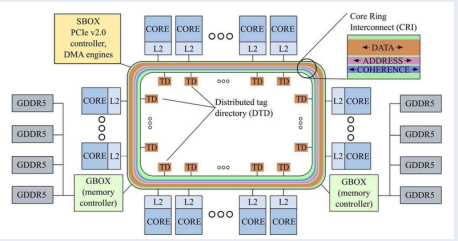
cả (ii) hai mô hình OpenMP dạng native và offload để song song hóa giải thuật di truyền (GA) lên các nút tính toán hiệu năng cao (HPC) có sử dụng một hay nhiều bộ đồng xử lý Intel Xeon Phi. Dựa vào công việc trước[3] chúng tôi giải quyết các vấn đề phân công nhiệm vụ trên các quá trình MPI và các thread của OpenMP trên Intel Xeon Phi. Thực hiện của chúng tôi đã thử nghiệm trên hệ thống 50-TFlops Supernode-XP, là cụm của các nút tính toán hiệu năng cao (mỗi nút gồm CPU đa nhân và hai (x2) bộ đồng xử lý Intel Xeon Phi Knights Corner (KNC) với 2x61 lõi = 122 lõi). Kết quả mô phỏng của các giải thuật cho thấy giải thuật di truyền được đề xuất thực thi trên 2 Intel Xeon Phi (122 lõi) giảm thời gian thực hiện đáng kể so với GA tuần tự và cho kết quả tìm ra lời giải nhanh hơn so với chỉ dùng 1 bộ đồng xử lý Intel Xeon Phi (61 lõi). Khác với nghiên cứu trước[4] đã công bố chỉ hiện thực giải thuật di truyền theo mô hình Offload của Intel Xeon Phi, bài báo này trình bày hiện thực giải thuật di truyền trên mô hình Native và mô hình phân bố trên nhiều KNC. Kết quả so sánh cả ba mô hình Offload, Native và phân bố được đánh giá trên cùng tập dữ liệu.

Phần còn lại của bài báo được tổ chức như sau. Trong phần **Kiến thức cơ sở và các nghiên cứu liên quan**, chúng tôi thảo luận về các công trình liên quan đến nghiên cứu này, và giới thiệu một số kiến thức cơ bản như kiến trúc MIC, bộ đồng xử lý Intel Xeon Phi, lập trình trên Intel Xeon Phi. Chi tiết về triển khai giải thuật di truyền trên Intel Xeon Phi theo các mô hình song song khác nhau được trình bày ở phần **Phương pháp đề xuất**. Trong phần **Kết quả nghiên cứu**, chúng tôi trình bày các kết quả đánh giá bằng mô phỏng cho các giải thuật di truyền trên KNC với các mô hình song song khác nhau. Tiếp theo là phần **Thảo luận** và phần **Kết luận** ở mục cuối.

KIẾN THỨC CƠ SỞ VÀ CÁC NGHIÊN CỨU LIÊN QUAN

Kiến trúc Many Integrated Core (MIC) và Intel Xeon Phi

Từ phương diện của một người lập trình, am hiểu về kiến trúc MIC[5] là một điều hết sức cần thiết để có thể khai thác tối đa hiệu suất của Intel Xeon Phi. Trong bài báo này, những kết quả thực nghiệm đo đạc đánh giá hiệu năng được thực hiện trên hệ thống có lắp đặt Intel Xeon Phi Knights Corner (KNC). KNC có chứa 61 lõi (CPU core) và các bộ điều khiển bộ nhớ (Memory Controllers), chúng được kết nối với nhau thông qua bus dạng ring được gọi là Core Ring Interconnect (CRI) (**Hình 1**). KNC có bộ nhớ cache hai mức (two-level cache). Cache level 1 có 32KB, level 2 có 512KB là vùng nhớ được dành riêng cho mỗi core. Ngoài ra, còn có vùng nhớ chung từ 6GB đến 16GB được kết nối thông qua bus dạng ring (**Hình 1**), băng thông bộ nhớ có thể đạt được khoảng 350 GB/s. Bên cạnh đó kiến trúc NUMA (non-uniform memory access) của KNC cũng là một điều đáng được quan tâm, do đó đảm bảo tính cục bộ về dữ liệu trong ứng dụng bất cứ khi nào có thể vì thời gian truy cập bộ nhớ giữa các cores phụ thuộc vào khoảng cách giữa chúng. Tốc độ của mỗi core dao động trong khoảng 1.0 đến 1.5 GHz (CPU có tần số rất cao từ 2 GHz đến 4 GHz) trong KNC. KNC có khoảng 61 cores, KNL (Knights Landing), Intel Xeon Phi thế hệ thứ hai có khoảng 72 cores. Trong mỗi core, người lập trình có thể kích hoạt lên tới 4 hardware threads, và khoảng 244 threads mỗi card Xeon Phi. Ngoài ra sự tồn tại của các vector units (512-bit vector registers) cũng giúp gia tăng đáng kể tốc độ thực thi chương trình. KNC cho phép ta thực hiện các phép tính SIMD trên 16 số độ chính xác đơn và 8 số độ chính xác kép. KNC được kết nối thông qua cổng PCIe tốc độ cao và được điều phối bởi CPU. Với tất cả những đặc điểm kỹ thuật được đề cập ở bên trên về bộ nhớ, core, vector units của KNC, hiệu suất cao nhất về mặt lý thuyết (theoretical peak performance) có thể đạt được là khoảng 1200 GFLOPS (giga floating point operation per second) độ chính xác kép và 2400 GFLOPS độ chính xác đơn. Một node tính toán có thể trang bị, lắp đặt nhiều card KNC tạo nên một nút tính toán có hiệu suất cao.

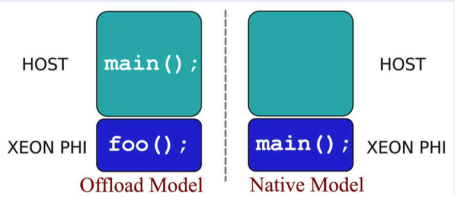


Hình 1: **Kiến trúc của KNC.**

Lập trình trên Intel Xeon Phi

Việc lập trình trên thiết bị đồng xử lý (coprocessor) Intel Xeon Phi được đánh giá là khá dễ so với lập trình GPU của NVIDIA thông qua CUDA (Compute Unified Device Architecture) bởi vì phần code được chạy trên CPU và Intel Xeon Phi là gần như giống nhau hoàn toàn. Trong một nút tính toán, có hai mô hình lập trình được sử dụng phổ biến đó là: native và offload (**Hình 2**). Các ứng dụng offload sẽ được khởi chạy trên host CPU, sau đó giao tiếp để thực hiện quá

trình tính toán trên coprocessor. Trong khi đó các ứng dụng native được chạy trên Xeon Phi một cách trực tiếp mà không cần phải thông qua CPU.



Hình 2: **Mô hình lập trình trên Intel Xeon Phi.**

OpenMP

OpenMP (Open Multi-Processing)[6] được chọn sử dụng để song song hoá chương trình, thay vì những framework hoặc là thư viện khác như Intel Cilk or TBB vì tính đơn giản và dễ sử dụng . OpenMP là một khung phần mềm hướng tính toán dành cho mô hình lập trình bộ nhớ chung (shared memory). OpenMP có rất nhiều chỉ thị (directives) có thể được sử dụng để song song hoá code theo nhiều hướng khác nhau. Tuy nhiên, chúng ta chỉ cần nắm rõ một vài chỉ thị là có thể hoàn toàn phát huy hết tiềm năng của các cores, và vector register trên Intel Xeon Phi như: **#pragma omp parallel , #pragma omp simd , #pragma omp for** .

Sự phân bố các threads vào các cores trên Intel Xeon Phi :

Để tận dụng khả năng tính toán của Intel Xeon Phi, tất cả các threads phải được sử dụng. Tuy nhiên, các threads có thể di chuyển (migrate) giữa các cores, điều đó phụ thuộc vào quyết định định thời của hệ điều hành. Điều này dẫn tới sự suy giảm đáng kể về mặt hiệu suất, nguyên nhân là vì các threads sau khi đã di chuyển (migration) phải lấy data từ trong cache của core cũ sang core mới. Trong các kỹ thuật tối ưu cho Intel Xeon Phi, có một kỹ thuật gọi là thread affinity. Kỹ thuật này có thể giúp ta hạn chế sự di chuyển của các thread bằng cách cấu hình biến môi trường **KMP_AFFINITY** vào các mode như: scatter, compact, và các mode khác. Trong đó scatter và compact là hai mode thường hay được sử dụng. Để hiểu rõ hơn về thread affinity ở 2 mode compact and scatter, chúng ta sẽ tham khảo ví dụ sau đây. Giả sử chúng ta có 60 threads cần gán vào hệ thống có 60 cores, mỗi core có thể chạy 4 threads cùng 1 lúc. Ở chế độ compact các threads sẽ được đặt gần nhau nhất có thể, do đó các threads được đánh số từ 0 đến 59 sẽ được gán vào 15 cores đầu tiên. Còn trong chế độ scatter, các threads sẽ được đặt xa nhau nhất có thể, do đó 60 threads sẽ được phân bố đều vào 60 cores.

Giải thuật di truyền

Giải thuật di truyền (GA) được sử dụng để tìm lời giải tối ưu toàn cục cho nhiều ứng dụng kỹ thuật và khoa học trước đây. Giải thuật di truyền cũng được sử dụng để giải quyết các vấn đề khác nhau trong khoa học và kỹ thuật[7] . Theo nghiên cứu của Arenas và cộng sự[8] , có nhiều nghiên cứu đã thử nghiệm giải thuật di truyền cho nhiều bài toán khác nhau có sử dụng một hay nhiều thiết bị GPU. Nghiên cứu của Einkemmer[2] cố gắng tìm một thiết kế tối ưu cho cánh máy bay. Tuy nhiên, đây là một vấn đề tối ưu hóa phi tuyến và các thuật toán tối ưu hóa truyền thống không hiệu quả. Các tác giả cho rằng giải thuật có thể được song song hóa tốt và nó có thể được hưởng lợi đáng kể từ các phần cứng GPU hoặc Intel Xeon Phi.

Một số nghiên cứu[9–11] gần đây sử dụng giải thuật di truyền cho bài toán lập lịch các công việc, bài toán lập lịch máy ảo[12] trên môi trường điện toán đám mây. Nghiên cứu của Pinel và cộng sự[10] đề xuất giải pháp giải thuật di truyền song song (Parallel Cellular Genetic Algorithm) trên GPU để giải quyết các bài toán lập lịch rất lớn các công việc độc lập. Nghiên cứu khác[3] cũng hiện thực giải thuật di truyền trên GPU cho bài toán phân bổ máy ảo lên các máy vật lý hướng mục tiêu tối thiểu năng lượng tiêu thụ của các máy vật lý, các tác giả đã đạt được chương trình thực thi nhanh và lời giải tốt hơn khi sử dụng GPU so với CPU. Bài nghiên cứu này sẽ trình bày kết quả tổng kết nghiên cứu và hiện thực giải thuật di truyền cho bộ đồng xử lý Intel Xeon Phi theo mô hình Offload và giải thuật di truyền phân bố trên nhiều bộ đồng xử lý Intel Xeon Phi sử dụng MPI + OpenMP.

Một nghiên cứu gần đây[13] phát triển thư viện pyMIC-DL tương tự NumPy[14], pyMIC-DL là một thư viện cho Học sâu (Deep learning), trên bộ đồng xử lý Intel Xeon Phi KNC. Đánh giá cho thấy khi thực thi chương trình Học Sâu trên pyMIC-DL có hiệu năng cao hơn (tính theo GFLOPS) so với dùng thư viện NumPy trên bộ xử lý đa lõi (Multicore CPU).

PHƯƠNG PHÁP ĐỀ XUẤT

Chúng tôi sẽ trình bày ba (03) giải thuật di truyền song song theo mô hình Intel Xeon Phi Native, Offload và Distributed trên các bộ đồng xử lý Intel Xeon Phi.

Giải thuật GAMIC (Offload): GAMIC (Offload) là giải thuật di truyền theo mô hình Intel Xeon Phi Offload. Giải thuật 1 trình bày mã giả cho giải thuật di truyền theo mô hình Intel Xeon Phi Offloading, ký

hiệu: **GAMIC (Offload)** . Giải thuật GAMIC (Offload) được đề xuất thực thi: khởi tạo nhiễm sắc thể, đánh giá thể lực, trao đổi chéo, đột biến và chọn lọc chạy trên một (1) bộ đồng xử lý Intel Xeon Phi (KNC). Bộ nhớ trên KNC chỉ được phân bổ dữ liệu một lần và tái sử dụng nhiều lần để giảm độ trễ, giảm tải (lưu dữ liệu). Hơn nữa, để giảm thời gian truy cập bộ nhớ chính kết quả chỉ được ghi trở lại bộ nhớ chính của host khi quá trình tính toán kết thúc (kết quả trung gian sẽ không được lưu trữ hoặc ghi lại vào bộ nhớ chính của host).

Giải thuật 1 : Giải thuật **GAMIC (Offload)**

Input: N – number of generations, D - dataset of virtual machines and physical machines, M – number of KNC threads

Output: None

1: Initialize some populations of chromosomes with random values in the search space

2: exe_time = omp_get_wtime()

3: #pragma offload target(mic) free_if(0) { 4: for some N generations do

// Call Crossover ▶Hàm 3

5: crossover(NSToffspring);

// Call Evaluation ▶Hàm 2

6: evaluation(NSToffspring, fitness_offspring, startList, endList);

// Call Selection ▶Hàm 5

7: selection(NSToffspring, fitness_offspring);

// Call Mutation ▶Hàm 4

8: mutation(NSToffspring);

// Call Evaluation ▶Hàm 2

9: evaluation(NSToffspring, fitness_offspring, startList, endList);

10: selection(NSToffspring, fitness_offspring);

11: end for } 12: Memory Deallocation on KNC

13: texe = omp_get_wtime() – exe_time

14: Write the best fitness value, texe to output file

Giải thuật 2 : Giải thuật **GAMIC (Native)**

Input: N – number of generations, D - dataset of virtual machines and physical machines, M – number of KNC threads

Output: None

1: Initialize some populations of chromosomes with random values in the search space

2: exe_time = omp_get_wtime()

3: #pragma omp parallel for simd 4: for some N generations do

// Call Crossover ▶Hàm 3

5: crossover(NSToffspring);

// Call Evaluation ▶Hàm 2

6: evaluation(NSToffspring, fitness_offspring, startList, endList);

// Call Selection ▶Hàm 5

7: selection(NSToffspring, fitness_offspring);

// Call Mutation ▶Hàm 4

8: mutation(NSToffspring);

// Call Evaluation ▶Hàm 2

9: evaluation(NSToffspring, fitness_offspring, startList, endList);

10: selection(NSToffspring, fitness_offspring);

11: end for 13: Memory Deallocation on KNC

14: texe = omp_get_wtime() – exe_time

15: Write the best fitness value, texe to output file

Giải thuật GAMIC (Native): Giải thuật GAMIC (Native) là giải thuật di truyền được thiết kế và hiện thực theo mô hình Intel Xeon Phi Native. Mã giả của giải thuật GAMIC (Native) được trình bày trong Giải thuật 2. GAMIC (Native) về cơ bản tương tự GAMIC (Offload) chỉ khác là trong tất cả hàm di truyền (Crossover, Evaluation, Selection, Mutation) việc song song hóa các vòng lặp được thực hiện bằng OpenMP # **pragma omp parallel for simd** và toàn bộ chương trình GAMIC (Native) (kể cả dữ liệu cho việc tính toán) đều chạy trên một (01) bộ đồng xử lý Intel Xeon Phi.

Giải thuật GAMIC (Distributed): Giải thuật GAMIC_distributed là giải thuật di truyền được thiết kế và hiện thực theo mô hình phân bố trên các bộ đồng xử lý Intel Xeon Phi (KNCs). Chúng tôi trình bày mã giả cho giải thuật di truyền phân tán (GAMIC_distributed) được đề xuất lên các bộ đồng xử lý Intel Xeon Phi (KNC), và các bộ đồng xử lý Intel Xeon Phi có thể giao tiếp với nhau qua mạng tốc độ cao InfiniBand tốc độ cao 48 Gb/s, trong Giải thuật 3. GAMIC_distributed gọi thủ tục GA_Evolution() và các hàm di truyền khác (Crossover, Evaluation, Selection, Mutation). Các hàm gồm khởi tạo nhiễm sắc thể, đánh giá về fitness, lai chéo, đột biến và lựa chọn sẽ được thực hiện trên KNC. Dữ liệu tải lên bộ nhớ của KNC chỉ được ghi một lần và được sử dụng lại nhiều lần để giảm độ trễ và giảm tải (còn giữ lại dữ liệu). Hơn nữa, để tránh mất thời gian chuyển dữ liệu ra bộ nhớ chính (ngoài host), kết quả chỉ được tải trở lại bộ nhớ chính khi thủ tục tính toán trên KNC kết thúc; kết quả trung gian sẽ không được lưu trữ hoặc ghi lên bộ nhớ chính của máy vật lý. Trên KNC, song song về dữ liệu (data parallelism) sẽ được thực hiện trên tất cả các tác vụ đánh giá (evaluation), lai chéo (crossover), đột biến (mutation), lựa chọn (selection) bằng cách sử dụng các chỉ thị của OpenMP. Tạo ra các số ngẫu nhiên là một chức năng chính trong GA, nhưng hàm rand() không thể thực thi song song, điều này khiến chương trình cực kỳ chậm, thậm chí chậm hơn giải thuật di truyền tuần tự (SGA)[3] . Giải thuật di truyền tuần tự (SGA) là giải thuật di truyền

tương tự GAMIC (Offload) chỉ được thực thi trên 1 lõi (1 CPU core hay 1 MIC core). Hàm rand() không được reentrant hoặc thread-safe , vì nó sử dụng trạng thái ẩn được sửa đổi trên mỗi lần gọi hàm rand() [15] . Chúng tôi cố gắng giải quyết vấn đề này bằng cách sử dụng lrand48() [16] , mặt khác, phiên bản GPU sử dụng CUDA thư viện curand.h để giải quyết nó [5].

Để tối ưu hóa mã, giải thuật GAMIC_distributed sử dụng các chỉ thị OpenMP pragma để thực hiện nhằm tận dụng lợi thế của kiến trúc MIC. Khi chúng tôi lập trình với Intel Xeon Phi, để tối đa hóa hiệu suất, đặc biệt là trong KNC, chúng tôi còn dùng các kỹ thuật lệnh song song SIMD chẳng hạn như các lệnh vector hóa (sử dụng # **pragma omp simd**), để tận dụng lợi thế của thanh ghi vector 512-bit, đa luồng (sử dụng # **pragma omp parallel** hoặc # **pragma omp for**), để sử dụng tối đa 61 lõi, 4 luồng trên mỗi lõi và cuối cùng là bộ nhớ, để tối đa hiệu suất chương trình trên KNC. Chúng tôi trình bày mã giả của hàm GA_Evolution và các hàm evaluation, crossover, mutation, selection bên dưới. Chúng tôi thêm một số chỉ thị được đề cập ở trên đã sử dụng trong các hàm trên.

Giải thuật 3 : Giải thuật **GAMIC (Distributed)**

Input: N – number of generations, D - dataset of virtual machines and physical machines, M – number of KNC threads

Output: None

1: Initialize some populations of chromosomes with random values in the search space

2: Call MPI Init()

3: Call MPI Comm rank(MPI COMM WORLD, &rank)

4: Call MPI Comm size(MPI COMM WORLD, &nproc)

5: exe_time = omp_get_wtime();
for (some test) do

6: Call GA_Evolution(generation)

7: Call MPI_Gather(&best_sol, 1, MPI FLOAT, solutions, 1, MPI FLOAT, MASTER, MPI COMM WORLD)

8: if (rank == MASTER) then

9: Choose the best solutions among nodes and write to file

10: end if

11: texe = omp_get_wtime() – exe_time;

12: end for

13: Call MPI_Finallize()

14: Write the best fitness, texe to output file

Hàm 1: void GA_Evolution(int N)

Input: N – number of generations, D - dataset of virtual machines and physical machines, M – number of KNC threads, P – number of MPI processes

Output: None

1: Initialize some clones using # **pragma omp parallel for simd**

2: for (int i =0 ; i¡sizePop*ntasks ; i++) do

3: NSToffspring[i] = mangNST[i];

4: end for

5: Call MPI_Init()

6: Using # **pragma omp parallel for simd** to create list of events (starting, ending) of tasks

7: Call evaluation for parents

8: Cloning the fitness array fitnesses of offspring using (# **pragma omp parallel for simd**)

9: for (g = 0; g < N; g++) {

10: crossover(NSToffspring);

11: evaluation(NSToffspring, fitness_offspring, startList, endList);

12: selection(NSToffspring, fitness_offspring);

13: mutation(NSToffspring);

14: evaluation(NSToffspring, fitness_offspring, startList, endList);

15: selection(NSToffspring, fitness_offspring);

16: }

17: Write the best fitness value to file

Hàm 2 : void evaluation (int *nst, float* fitness, int* eventStartList, int* eventEndList)

Input: nst – array of chromosomes, fitness – array of fitnesses, eventStartList – pointer for VM's start event list, eventEndList - pointer for VM's end event list.

Output: None

1: **#pragma omp parallel for**

2: for (i = 0; i <sizePop; i++) {

3: Calculate fitness of a chromosome. See the flowchart below (Hình 3).

4: }

Hàm 3 : void crossover(int* NSToffspring)

Input: NSToffspring – array of offsprings

Output: None

1: **#pragma omp parallel for**

2: for (i=0 ; i<sizePop ; i++){

3: int cross_point = my_parallel_rand(0, 0.5 * ntasks);

4: int neighbor = my_parallel_rand(0, sizePop - 1);

5: if (neighbor == i) { neighbor = (i+1)%sizePop; }

6: **#pragma omp simd**

7: for (p = 0 ; p < ntasks/2 - 1 ; p++) {

8: NSToffspring[i*ntasks+cross_point+p] = mang_NST[neighbor*ntasks + cross_point + p];

9: }

10: }

Hàm 4 : void mutation (int* NSToffspring)

Input : NSToffspring – array of offsprings

Output : None

```
1: #pragma omp parallel for simd
2: for (int i = 0; i < ntasks*sizePop; i++) {
3: if (((float)lrand48() / RAND_MAX) < rateMutan)
{
4: NSToffspring[i] = my_parallel_rand(0, numMa-
chine -1);
5: } }
```

Hàm 5 : void selection(int* NSToffspring, float* fitness_offspring)

Input: NSToffspring – array of offsprings, n – length of C, fitness_offspring - array of fitness of offsprings

Output: None

```
// reset rankings
1: #pragma omp parallel for simd
2: for(i= 0; i < 2*sizePop; i++) {
3: rankings[i] = i; }
// copy key sort
4: float keySort[2*sizePop];
5: #pragma omp parallel for simd
6: for(i= 0; i < sizePop; i++) { keySort[i] =
mang_fitness[i]; keySort[i+sizePop] = fit-
ness_offspring[i]; }
// sort rankings[]by key
7: sort_by_key_greater(keySort, 2*sizePop, rank-
ings);
// move top NST
8: int temNST[sizePop*ntasks];
9: #pragma omp parallel for
10: for (i =0 ; i< sizePop ; i++) {
11: // copy to temNST
12: if ( rankings[i] < sizePop) {
13: #pragma omp simd
14: for (p = 0; p<ntasks; p++) {
15: temNST[i * ntasks + p]
= mang_NST[rankings[i]*ntasks +p ];
16: }
17: } else {
18: #pragma omp simd
19: for (p = 0; p<ntasks; p++) {
20: temNST[i * ntasks + p] = NSToffspring [ (rank-
ings[i] - sizePop) *ntasks + p]; }
21: }
22: }
// copy to parentsNST
23: #pragma omp parallel for simd
24: for (i = 0 ; i < sizePop*ntasks; i++) {
25: mang_NST[i] = temNST[i];
26: NSToffspring[i] = temNST[i]; }
27: // copy to parents fitness
28: #pragma omp parallel for simd
29: for (i =0 ; i< sizePop ; i++) {
30: mang_fitness[i] = keySort[i];
31: fitness_offspring[i] = keySort[i];
32: }
```

Hàm evaluation() được gọi để đánh giá fitness của từng nhiễm sắc thể (NST) trong mảng NST, công thức tính fitness của mỗi NST phụ thuộc vào bài toán. Trong phần mô phỏng với bài toán lập lịch máy ảo lên các máy vật lý với mục tiêu năng lượng tiêu thụ các máy vật lý nhỏ nhất thì hàm tính fitness được trình bày như trong hàm 2. Chỉ dẫn OpenMP là **#pragma omp parallel for** được dùng để song song vòng lặp đánh giá toàn bộ mảng NST, mỗi thread của KNC sẽ đánh giá một NST (mỗi KNC có từ 61 thread đến 244 thread) , mỗi thread của mô hình OpenMP thực hiện tính toán cho mỗi NST như lưu đồ tại **Hình 3**, các thead thực thi song song trên KNC.

Để tạo ra quần thể con mới, hàm crossover() là bước tiếp theo của sự tiến hóa. Hàm crossover là quá trình chọn hai nhiễm sắc thể (NST) ngẫu nhiên để tạo thành hai cái NST mới. Chỉ thị OpenMP là **#pragma omp parallel for** được dùng để thực thi vòng lặp song song trên các thread của KNC (mỗi KNC có từ 61 thread đến 244 thread). Một NST sẽ giao với NST khác được chọn ngẫu nhiên cùng với điểm chéo bởi một vị trí được chỉ định. Sau đó hai đoạn của hai NST cha mẹ sẽ được trao đổi để tạo ra hai những cái mới bằng cách cắt phần từ vị trí để ghép chéo với nhau. Chỉ thị OpenMP là **#pragma omp simd** (lệnh vectorization của KNC) được sử dụng để đẩy nhanh quá trình trao đổi các đoạn này.

Hàm mutation() thì khi điều kiện đột biến thỏa mãn thì chọn 1 vị trí theo theo số ngẫu nhiên từ 0 đến (N - 1) để thực hiện đột biến. Ở bài toán lập lịch máy ảo thì N là tổng số số máy ảo cần gán lên các máy vật lý. Hàm mutation có thời gian thực thi ngắn và chỉ thị OpenMP là **#pragma omp parallel for** cũng được dùng.

Hàm selection() thì sắp xếp (sorting) mảng nhiễm sắc thể (NST) theo giá trị fitness giảm dần. Quá trình sắp xếp này dùng cách sắp xếp (sorting) song song[17] trên các thread của KNC.

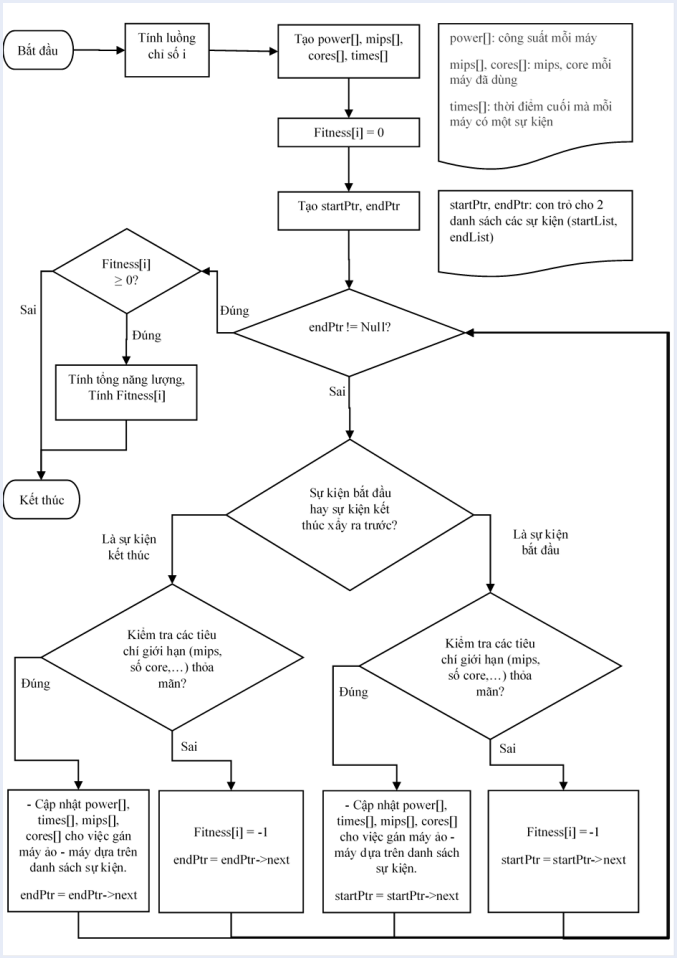
KẾT QUẢ NGHIÊN CỨU

Môi trường

Môi trường đánh giá các giải thuật di truyền gồm GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) như trong **Hình 4**.

Kết quả nghiên cứu

Kết quả nghiên cứu đánh giá ba (03) giải thuật di truyền gồm GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) được trình bày trong các Bảng 1 và 2 và Hình 6 . Với kích thước quần thể (population size) càng lớn thì thời gian thực thi của các



Hình 3: **Lưu đồ kiểm tra và tính toán lượng giá của các nhiễm sắc thể.**

Codename	CPU	Coprocessor
Model	Intel Xeon E5-2680V3	Intel Xeon Phi 7120P
Microarchitecture	Sandy Bridge EP	Intel Many Intergrated Core
Clock frequency	2.50/3.30 GHz	1.24/1.33 GHz
Memory Size	128 GB	16 GB
Cache	30.0 MB SmartCache	30.5 MB L2
Max Memory Bandwidth	68 GB/s	352 GB/s
Core/Threads	12/24	61/244

Hình 4: **Cấu hình phần cứng CPU và bộ đồng xử lý Intel Xeon Phi (KNC).**

giải thuật di truyền càng lớn. Bảng 1 là thời gian thực thi của các giải thuật di truyền gồm GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed), trong đó bài toán lập lịch với kích thước đầu vào là N = 500 máy ảo x M = 500 máy vật lý, số thế hệ là 500 (**Hình 5**). **Bảng 2** là thời gian thực thi của các giải thuật di truyền gồm GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed), trong đó bài toán lập lịch với kích thước đầu vào là N = 1000 máy ảo x M = 1000 máy vật lý, số thế hệ là 500 (**Hình 6**). Tập dữ liệu cho các máy ảo và các máy vật lý có thể download tại URL: https://cutt.ly/peLEoEQ.

Giải thuật di truyền GAMIC_Distributed có thời gian thực thi (execution time) phụ thuộc vào thời gian tính toán (compute time) và thời gian giao tiếp (communication time), thời gian giao tiếp phụ thuộc vào số lượng KNC tham gia vào quá trình tính toán. Khi cả hai KNC trên cùng một máy vật lý thì thời gian giao tiếp rất nhỏ. Thời gian thực thi gần tương đương của các giải thuật di truyền gồm GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) trong hai **Bảng 1** và **Bảng 2** .

THẢO LUẬN

Từ các kết quả thử nghiệm ở trên (Mục Kết quả nghiên cứu), nếu kích thước bài toán nhỏ và khối lượng tính toán ít thì giải thuật di truyền song song theo cơ chế Native (GAMIC_native) đề xuất thực thi hoàn toàn trên một (01) MIC có 61 lõi và bộ nhớ 16GB

Bảng 1: Thời gian thực thi của các giải thuật di truyền GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) cho bài toán lập lịch có kích thước 500 máy ảo và 500 máy vật lý

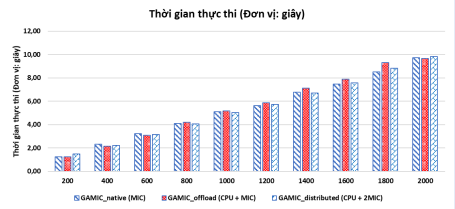
Kích thước quần thể	GA serial (Dùng 1 nhân trong MIC)	GAMIC_native (1 MIC)	GAMIC_offload (1 CPU + 1 MIC)	GAMIC_distributed (1 CPU + 2 MIC)		
				Compute time	Communication time	Execution time
200	31,68	1,23	1,23	1,47	0,00042	1,47
400	63,77	2,31	2,16	2,22	0,00012	2,22
600	95,89	3,21	3,08	3,16	0,00011	3,16
800	127,37	4,10	4,21	4,06	0,00008	4,06
1000	159,72	5,10	5,18	5,03	0,00021	5,03
1200	190,70	5,62	5,88	5,72	0,00015	5,72
1400	223,53	6,76	7,13	6,71	0,00031	6,71
1600	255,34	7,46	7,88	7,58	0,00018	7,58
1800	287,41	8,53	9,30	8,83	0,00009	8,83
2000	319,34	9,74	9,68	9,84	0,00026	9,84

Ghi chú: một (01) MIC là một (01) KNC.

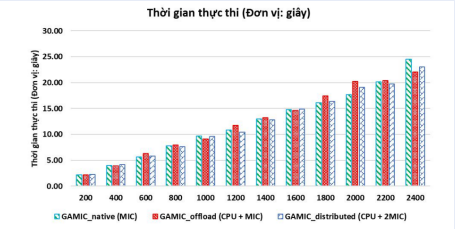
Bảng 2: Thời gian thực thi (đơn vị: giây) của các giải thuật di truyền GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) cho bài toán lập lịch có kích thước 1000 máy ảo và 1000 máy vật lý

Kích thước quần thể	GAMIC_native (1 MIC) (Đơn vị: giây)	GAMIC_offload (1 CPU + 1 MIC) (Đơn vị: giây)	GAMIC_distributed (1 CPU + 2 MIC)		
			Thời gian tính toán (Đơn vị: giây)	Thời gian giao tiếp (Đơn vị: giây)	Thời gian thực thi (Đơn vị: giây)
200	2,17	2,18	2,24	0,000291	2,24
400	3,96	3,89	4,17	0,000105	4,17
600	5,66	6,30	5,80	0,000328	5,80
800	7,78	7,99	7,63	0,000101	7,63
1000	9,66	9,07	9,58	0,000013	9,58
1200	10,81	11,77	10,46	0,000263	10,46
1400	13,01	13,21	12,85	0,000056	12,85
1600	14,78	14,60	14,89	0,000014	14,89
1800	16,08	17,42	16,33	0,000143	16,33
2000	17,70	20,20	19,09	0,000019	19,09
2200	20,12	20,42	19,70	0,000433	19,70
2400	24,53	22,02	23,05	0,000013	23,05

Ghi chú: một (01) MIC là một (01) KNC.



Hình 5: Thời gian thực thi của các giải thuật di truyền GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) cho tập dữ liệu 500 máy ảo và 500 máy vật lý.



Hình 6: Thời gian thực thi của các giải thuật di truyền GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed) cho tập dữ liệu 1000 máy ảo và 1000 máy vật lý.

sẽ nhanh hơn giải thuật di truyền song song theo các cơ chế khác là Offload (GAMIC_offload) đề xuất trên cùng 1 KNC, và mô hình kết hợp MPI và OpenMP (GAMIC _ distributed) trên hai (02) KNC với 122 lõi. Do giới hạn về bộ nhớ 16 GB của thiết bị MIC nên khi kích thước bài toán lớn hơn 1000 và kích thước quần thể lớn từ 2400 thì giải thuật GAMIC_native có thời gian thực thi lớn hơn so với hai giải thuật khác là GAMIC_offload và GAMIC_distributed sử dụng một (01) CPU và hai (02) MIC. Giải thuật di truyền GAMIC _ distributed này có thời gian thực thi (execution time) phụ thuộc vào thời gian tính toán (compute time) và thời gian giao tiếp (communication time) trình bày trong Bảng 1 và Bảng 2 . Thời gian giao tiếp phụ thuộc vào số lượng KNC tham gia vào quá trình tính toán. Khi cả hai KNC trên cùng một máy vật lý thì thời gian giao tiếp rất nhỏ. Thời gian thực thi gần tương đương của các giải thuật di truyền gồm GAMIC (Native), GAMIC (Offload) và GAMIC (Distributed). Giải thuật di truyền phân bố GAMIC_distributed có xác suất tìm thấy lời giải cao hơn các giải thuật khác. Khi kích thước quần thể nhỏ (từ 200 đến 800), không phải lúc nào các giải thuật di truyền theo mô hình Native và Offload cũng tìm thấy lời giải và không tìm thấy nếu kích thước quần thể nhỏ dưới 200. Kết quả tìm thấy giá trị tốt nhất ổn định ở giải thuật di truyền GAMIC_distributed.

KẾT LUẬN

Tổng kết, chúng tôi đã đề xuất các mô hình Native, Offload, và phân bố cho giải thuật di truyền trên bộ đồng xử lý Intel Xeon Phi (KNC). Mô hình Native là dễ lập trình. Mô hình Offload này nên dùng cho những hệ thống có bộ xử lý (CPU) yếu, kết hợp với bộ đồng xử lý Intel Xeon Phi (KNC) để tăng tốc tính toán của giải thuật. Mô hình phân bố (Distributed) thì có thời gian thực thi chương trình là tổng thời gian tính toán của CPU hay KNC chậm nhất cộng với thời gian giao tiếp, nhưng bù lại mô hình phân bố này cho xác suất tìm được nghiệm cao hơn 2 model kia (Native và Offload) với giải thuật di truyền cho bài toán lập lịch máy ảo. Việc sử dụng lập trình MPI lai đối xứng (symmetric) và OpenMP để giảm thời gian thực hiện đáng kể so với chương trình tuần tự. Các kết quả thú vị. Ngoài ra, MPI lai đối xứng và OpenMP được đề xuất có thể sử dụng cụm máy chủ HPC có nhiều máy chủ có các bộ đồng xử lý Intel Xeon Phi để trả về một giải pháp tốt hơn so với GAMIC (Native) và GAMIC (Offload). Với kích thước bài toán có dữ liệu đầu vào lớn chúng tôi đề xuất sử dụng mô hình phân bố dùng MPI lai đối xứng với OpenMP tương tự giải thuật GAMIC (Distributed) được đề xuất trong bài báo này. Mở rộng, kết quả nghiên cứu này có thể áp dụng cho các meta-heuristic khác như giải thuật tìm kiếm TABU, tối ưu đàn kiến (Ant Colony Optimization).

LỜI CẢM ƠN

Nghiên cứu được tài trợ bởi Đại học Quốc Gia Thành phố Hồ Chí Minh (ĐHQG-HCM) trong khuôn khổ đề tài mã số đề tài C2017-20-09.

DANH MỤC CÁC TỪ VIẾT TẮT

CPU: Central Processing Unit (Bộ xử lý)
GA: Genetic Algorithm (Giải thuật di truyền)
GPU: Graphic Processing Unit (Bộ xử lý đồ họa)
HPC: High Performance Computing (Tính toán hiệu năng cao)
KNC: Intel Xeon Phi Knights Corner © (Bộ đồng xử lý KNC)
KNL: Knights Landing (Bộ đồng xử lý KNL)
MIC: Many Integrated Core (Nhiều nhân tích hợp)
PATS: Power Aware Task Scheduling (Bài toán lập lịch công việc nhận biết năng lượng)

XUNG ĐỘT LỢI ÍCH

Nhóm tác giả xin cam đoan rằng không có bất kỳ xung đột lợi ích nào trong công bố bài báo.

ĐÓNG GÓP CỦA CÁC TÁC GIẢ

Nguyễn Quang Hùng và Thoại Nam tham gia vào việc đưa ra ý tưởng viết bài, thu thập dữ liệu, và viết bài báo.

Trần Ngọc Anh Tú tham gia viết chương trình, đánh giá kết quả, thu thập dữ liệu, và viết bài báo.

TÀI LIỆU THAM KHẢO

1. Jeffers J, Reinders J. Intel Xeon Phi Coprocessor High Performance Programming. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc; 2013.
2. Einkemmer L. Evaluation of the Intel Xeon Phi 7120 and NVIDIA K80 as accelerators for two-dimensional panel codes. PLOS ONE. 2017;12(6):e0178156–e0178156. Available from: 10.1371/journal.pone.0178156;https://dx.doi.org/10.1371/journal.pone.0178156.
3. Quang-Hung N, Tan LT, Phat CT, Thoai N. A GPU-Based Enhanced Genetic Algorithm for Power-Aware Task Scheduling Problem in HPC Cloud. In: Linawati, Mahendra MS, Neuhold EJ, Tjoa AM, , You I, editors. ICT-EurAsia. vol. 8407. Springer; 2014. p. 159–169.
4. Quang-Hung N, Tran ATN, Thoai N. Implementing Genetic Algorithm Accelerated By Intel Xeon Phi. Proceedings of the Eighth International Symposium on Information and Communication Technology - SoICT. 2017;p. 249–254.
5. Jeffers J, Group TC. Intel® Many Integrated Core Architecture: An Overview and Programming Models; 2012.
6. OpenMP. Available: https://www.openmp.org/. [Accessed: 02-Jan-2019].
7. Talbi EG. METAHEURISTICS FROM DESIGN TO IMPLEMENTATION. JohnWiley & Sons; 2009.
8. Arenas G, Mora AM, Romero G, Castillo PA. GPU computation in bioinspired algorithms: a review. In: Cabestany J, Rojas I, Joya G, editors. Advances in Computational Intelligence. vol. 6691. Springer; 2011. p. 433–440.
9. Quang-Hung N, Nien PDP, Nam NNH, Tuong NH, Thoai N. A Genetic Algorithm for Power-Aware Virtual Machine Allocation in Private Cloud. In: Mustofa K, Neuhold EJ, Tjoa AM, Weippl E, , You I, editors. ICT-EurAsia'13. vol. 7804. Springer-Verlag; 2013. p. 183–191.
10. Pinel F, Dorronsoro B, Bouvry P. Solving very large instances of the scheduling of independent tasks problem on the GPU. Journal of Parallel and Distributed Computing. 2013;73(1):101–110. Available from: 10.1016/j.jpdc.2012.02.018;https://dx.doi.org/10.1016/j.jpdc.2012.02.018.
11. Nguyen-Duc T, Quang-Hung N, Thoai N. BFD-NN: Best Fit Decreasing-Neural Network for Online Energy-Aware Virtual Machine Allocation Problems. Proceedings of the Seventh Symposium on Information and Communication Technology - SoICT '16. 2016;p. 243–250.
12. Quang-Hung N, Le DK, Thoai N, Son NT. Heuristics for Energy-Aware VM Allocation in HPC Clouds. In: Future Data and Security Engineering: First International Conference. vol. 8860; 2014. p. 248–261.
13. Tran ATN, Nguyen HP, Nguyen MT, Diep TD, Quang-Hung N, Thoai N. PyMIC-DL: A Library for Deep Learning Frameworks Run on the Intel® Xeon PhiTM Coprocessor. Proc - 20th Int Conf High Perform Comput Commun 16th Int Conf Smart City 4th Int Conf Data Sci Syst HPCC/SmartCity/DSS 2018. 2018;p. 226–234.
14. Walt S, Colbert S, Varoquaux G. The NumPy Array: A Structure for Efficient Numerical Computation. Computing in Science & Engineering. 2011;13(2):22–30. Available from: 10.1109/MCSE.2011.37.
15. rand() - pseudo-random number generator, Linux man page; 2017.
16. rand48(3) - Linux man pages; 2017.
17. Quinn J. Parallel Computing (2Nd Ed.): Theory and Practice. McGraw-Hill, Inc; 1997.

Science & Technology Development Journal – Engineering and Technology, 2(4):277-287

Open Access Full Text Article

Research Article

Parallel approaches of genetic algorithm in the MIC architecture of the Intel Xeon Phi

Nguyen Quang Hung, Anh-Tu Ngoc Tran, Nam Thoai

ABSTRACT

Today, genetic algorithms are widely used in many fields such as bioinformatics, computer science, artificial intelligence, finance ... Genetic algorithms are applied to create high quality solutions for complex optimization problems in the above industries. There have been many studies based on the proposed new hardware architecture that aims to speed up the execution of genetic algorithms as quickly as possible. Some studies suggest parallel genetic algorithms on systems with multicore CPUs and / or graphics processing units (GPUs). However, very few solutions propose a genetic algorithm that can be run on systems that use the new Intel Xeon Phi co-processor (Intel Many-Integrated Core (MIC) architecture). For that reason, we propose and develop the study of the genetic algorithm on high-performance computing systems with Intel Xeon Phi co-processors. This study will present the results of parallel approaches of genetic algorithm on one and more Intel Xeon Phi co-processors by the following methods: (i) Intel Xeon Phi programming model Offload and Native; and (ii) a combined model of MPI and OpenMP. The proposed genetic algorithm can find the optimal schedule for the energy-efficient scheduling problem of virtual machines on physical machines with the goal of minimization total energy consumption. The results of the simulations show the feasibility of implementing a genetic algorithm on one or many Intel Xeon Phi. Genetic algorithm on one or more distributed Intel Xeon Phi always results in faster algorithm execution time than sequential genetic algorithm and the ability to find better solutions using more Intel Xeon Phi. This research result can be applied to other meta-heuristic like TABU search, Ant Colony Optimization.

Key words: Parallel processing, Genetic algorithms, Xeon Phi, MIC, Meta-heuristic

Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology, VNU-HCM, Vietnam

Correspondence

Nguyen Quang Hung, Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology, VNU-HCM, Vietnam

History

- Received: 09-10-2019
- Accepted: 20-11-2019
- Published: 31-12-2019

DOI : 10.32508/stdjet.v2i4.612

Check for updates

VNU-HCM Press

Cite this article : Quang Hung N, Ngoc Tran A, Thoai N. **Parallel approaches of genetic algorithm in the MIC architecture of the Intel Xeon Phi**. *Sci. Tech. Dev. J. – Engineering and Technology;* 2(4):277-287.