**TRỪU TƯỢNG - ABSTRACT**

Mô phỏng chính xác là điều cần thiết cho việc thiết kế và đánh giá phù hợp bất kỳ nền tảng điện toán nào. Trước xu hướng hiện tại hướng tới kỷ nguyên điện toán không đồng nhất CPU-GPU, các nhà nghiên cứu cần một khung mô phỏng có thể mô hình hóa cả hai loại thiết bị điện toán và sự tương tác của chúng. Trong bài viết này, chúng tôi giới thiệu Multi2Sim, một bộ công cụ mã nguồn mở, mô-đun và có thể cấu hình đầy đủ, cho phép mô phỏng cấp độ ISA của CPU x86 và GPU AMD Evergreen. Tập trung vào mẫu GPU AMD Radeon 5870, chúng tôi giải quyết tính chính xác của việc mô phỏng chương trình cũng như độ chính xác của mô phỏng kiến trúc bằng cách sử dụng bộ điểm chuẩn OpenCL của AMD. Khả năng mô phỏng được thể hiện bằng nghiên cứu khám phá kiến trúc sơ bộ và các ví dụ về đặc tính khối lượng công việc. Mã nguồn dự án, các gói điểm chuẩn và hướng dẫn chi tiết cho người dùng được cung cấp công khai tại *www.multi2sim.org* .

**1. GIỚI THIỆU - INTRODUCTION**

GPU đã trở thành một thành phần quan trọng của nền tảng Điện toán hiệu năng cao (HPC) bằng cách tăng tốc các phần dữ liệu song song đòi hỏi khắt khe hơn bao giờ hết của một loạt ứng dụng. Sự thành công của điện toán GPU đã khiến các nhà nghiên cứu bộ vi xử lý ở cả giới học viện và ngành công nghiệp tin rằng điện toán không đồng nhất CPU-GPU không chỉ là một giải pháp thay thế mà còn là tương lai của HPC. Giờ đây, GPU đang hiển thị dưới dạng bộ tăng tốc tích hợp cho các nền tảng có mục đích chung [8, 5, 9]. Động thái này nhằm mục đích tận dụng khả năng kết hợp của kiến trúc CPU đa lõi và GPU nhiều lõi.

Khi nghiên cứu điện toán không đồng nhất CPU-GPU có đà phát triển, nhu cầu cung cấp một môi trường mô phỏng mạnh mẽ trở nên quan trọng hơn. Khung mô phỏng cung cấp một số lợi ích cho các nhà nghiên cứu. Chúng cho phép đánh giá các thiết kế tiền silicon và thu được kết quả hoạt động cho nhiều điểm thiết kế. Một số bộ mô phỏng CPU hỗ trợ mô phỏng ở cấp độ ISA đã được phát triển [11, 14] và được sử dụng thành công trong một loạt các nghiên cứu kiến trúc. Mặc dù có những công cụ hiện có sẵn để mô phỏng GPU ở cấp độ ngôn ngữ trung cấp (ví dụ: PTX) [12, 13], cộng đồng nghiên cứu vẫn thiếu một khung có sẵn công khai tích hợp cả mô phỏng chức năng nhanh và mô phỏng kiến trúc chi tiết chính xác theo chu kỳ ở cấp độ ngôn ngữ trung cấp. Cấp độ ISA xem xét mô hình CPU-GPU không đồng nhất thực sự.

Trong bài báo này, chúng tôi trình bày Multi2Sim, một khung mô phỏng cho tính toán CPU-GPU. Khung đề xuất tích hợp một mô hình có sẵn công khai của dòng GPU AMD Evergreen song song dữ liệu [3] 1 với mô phỏng bộ xử lý x86 siêu vô hướng, đa luồng và đa lõi. Công trình này cũng cung cấp cái nhìn sâu sắc quan trọng về kiến trúc của GPU AMD Evergreen, bằng cách mô tả các mô hình quy trình hướng dẫn và phân cấp bộ nhớ của chúng tôi, ở mức độ sâu hơn so với công trình công cộng trước đây, theo hiểu biết tốt nhất của chúng tôi, đã thực hiện trước đây.

Multi2Sim được cung cấp dưới dạng bộ công cụ dòng lệnh dựa trên Linux, được thiết kế tập trung vào việc trình bày giao diện thân thiện với người dùng. Nó chạy các ứng dụng OpenCL mà không cần sửa đổi mã nguồn và cung cấp một số khả năng thiết bị cho phép nghiên cứu đặc tính ứng dụng, tối ưu hóa mã, tối ưu hóa trình biên dịch và thiết kế kiến trúc phần cứng. Để minh họa tiện ích và sức mạnh của bộ công cụ của chúng tôi, chúng tôi báo cáo về nhiều kết quả thử nghiệm dựa trên điểm chuẩn lấy từ SDK xử lý song song tăng tốc (APP) SDK 2.5 [1] của AMD.

Phần còn lại của bài viết này được tổ chức như sau. Phần 2 giới thiệu mô hình mô phỏng chức năng trong Multi2Sim. Phần 3 trình bày kiến trúc GPU Evergreen và mô phỏng của nó. Phần 4 báo cáo đánh giá thử nghiệm của chúng tôi. Chúng tôi tóm tắt công việc liên quan ở Phần 5 và kết luận bài viết ở Phần 6.

**2. DỰ ÁN MULTI2SIM**

Dự án Multi2Sim bắt đầu dưới dạng khung mô phỏng chính xác theo chu kỳ , mã nguồn mở , miễn phí nhắm vào các CPU x86 siêu vô hướng, đa luồng và đa lõi. Khung mô phỏng CPU bao gồm hai thành phần phần mềm tương tác chính: *bộ mô phỏng chức năng* và *bộ mô phỏng kiến trúc* . Trình mô phỏng chức năng (tức là *trình mô phỏng* ) bắt chước việc thực thi chương trình khách trên bộ xử lý x86 gốc, bằng cách diễn giải chương trình nhị phân và tái tạo động hành vi của nó ở cấp độ ISA. Trình mô phỏng kiến trúc (tức là trình mô phỏng *chi tiết* hoặc *thời gian* ) thu được dấu vết của lệnh x86 từ trình mô phỏng chức năng và theo dõi việc thực thi các cấu trúc phần cứng bộ xử lý trên cơ sở từng chu kỳ .

Phiên bản hiện tại của trình mô phỏng chức năng CPU hỗ trợ thực thi một số bộ điểm chuẩn khác nhau mà không cần bất kỳ nỗ lực chuyển nào, bao gồm các bộ điểm chuẩn đơn luồng (ví dụ: SPEC2006 và Mediabench ), điểm chuẩn song song đa luồng (SPLASH-2 và PARSEC 2.1) , cũng như mã người dùng tự biên dịch tùy chỉnh. Trình mô phỏng kiến trúc mô hình hóa các đường ống siêu vô hướng nhiều lõi với khả năng thực thi không theo thứ tự, hệ thống phân cấp bộ nhớ hoàn chỉnh với sự kết hợp bộ đệm, mạng kết nối và các thành phần bổ sung.

Multi2Sim tích hợp một mô hình có thể cấu hình cho dòng GPU AMD Evergreen thương mại (ví dụ: Radeon 5870). Các bản phát hành mới nhất hỗ trợ đầy đủ cả mô phỏng chức năng và kiến trúc của GPU, tuân theo cùng một mô hình tương tác giữa chúng như đối với mô phỏng CPU. Trong khi trình mô phỏng GPU cung cấp dấu vết của các hướng dẫn Evergreen thì trình mô phỏng chi tiết theo dõi thời gian thực thi và trạng thái kiến trúc.

Tất cả các chương trình mô phỏng đều bắt đầu bằng việc thực thi mã CPU. Giao diện cho trình mô phỏng GPU là Ngôn ngữ tính toán mở (OpenCL). Khi các chương trình OpenCL được thực thi, các phần máy chủ (tức là CPU) của chương trình sẽ được chạy bằng cách sử dụng các mô-đun mô phỏng CPU. Khi gặp lệnh gọi API OpenCL, chúng sẽ bị chặn và được sử dụng để thiết lập hoặc bắt đầu mô phỏng GPU.

**2.1 Mô hình lập trình OpenCL**

OpenCL là một khung lập trình tiêu chuẩn công nghiệp được thiết kế đặc biệt để phát triển các chương trình nhắm vào các nền tảng điện toán không đồng nhất, bao gồm CPU, GPU và các loại thiết bị xử lý khác [7]. Mô hình lập trình của OpenCL nhấn mạnh việc xử lý song song bằng cách sử dụng mô hình *đa dữ liệu chương trình đơn* (SPMD), trong đó một đoạn mã duy nhất, được gọi là *kernel* , ánh xạ tới nhiều tập hợp con dữ liệu đầu vào, tạo ra một lượng lớn thực thi song song.

Hình 1 cung cấp cái nhìn về hệ thống phân cấp các phần tử thực thi cơ bản được xác định trong OpenCL. Một phiên bản của hạt nhân OpenCL được gọi là một *mục công việc* , có thể truy cập vào vùng *bộ nhớ riêng của chính nó* . Các hạng mục công việc được sắp xếp thành *các nhóm công việc* với hai thuộc tính cơ bản: i ) các mục công việc đó trong cùng một nhóm công việc có thể thực hiện các hoạt động đồng bộ hóa hiệu quả và ii) các mục công việc trong cùng một nhóm công việc có thể chia sẻ dữ liệu thông qua *bộ nhớ cục bộ* có độ trễ thấp . Tổng số các nhóm công việc tạo thành *ND-Range* (lưới các nhóm mục công việc) và chia sẻ một *bộ nhớ chung chung* .

**2.2 Mô phỏng OpenCL**

Ngăn xếp cuộc gọi của chương trình OpenCL chạy trên Multi2Sim khác với ngăn xếp cuộc gọi gốc bắt đầu từ thư viện OpenCL

cuộc gọi, như trong Hình 2. Khi lệnh gọi hàm API OpenCL được thực hiện, việc triển khai thời gian chạy OpenCL (libm2s-opencl.so) của chúng tôi sẽ xử lý cuộc gọi. Cuộc gọi này bị chặn bởi mô-đun mô phỏng CPU, mô-đun này chuyển quyền điều khiển sang mô-đun GPU ngay khi ứng dụng khách khởi chạy quá trình thực thi nhân thiết bị. Cơ sở hạ tầng này cho phép các tệp nhị phân x86 chưa sửa đổi (các chương trình máy chủ OpenCL được biên dịch sẵn) chạy trên Multi2Sim với khả năng tương thích hoàn toàn nhị phân với môi trường gốc.

**3. MÔ PHỎNG KIẾN TRÚC GPU AMD EVERGREEN**

Phần này trình bày kiến trúc của một thiết bị GPU AMD Evergreen chung, tập trung vào các thành phần phần cứng dành cho tính toán mục đích chung của hạt nhân OpenCL. Là một trong những điểm mới của bài viết này, các sơ đồ khối và mô tả sau đây cung cấp một số thông tin chi tiết về quy trình hướng dẫn, thành phần bộ nhớ và kết nối, có xu hướng được các nhà cung cấp GPU lớn giữ kín và vẫn không được ghi lại trong các công cụ hiện có. Tất cả các chi tiết kiến trúc được trình bày đều được mô hình hóa chính xác trên Multi2Sim, như được mô tả tiếp theo.

**3.1 Kiến trúc GPU Evergreen**

GPU bao gồm một *bộ điều phối siêu luồng* , một loạt *các đơn vị tính toán độc lập* và *hệ thống phân cấp bộ nhớ* . Bộ điều phối siêu phân luồng xử lý Phạm vi ND và ánh xạ các nhóm công việc đang chờ vào các đơn vị điện toán có sẵn. Khi một nhóm công việc được chỉ định cho một đơn vị tính toán, nhóm đó sẽ vẫn ở trong đơn vị tính toán đó cho đến khi quá trình thực thi hoàn tất. Khi một nhóm công việc thực thi, các mục công việc sẽ tìm nạp và lưu trữ dữ liệu thông qua hệ thống phân cấp bộ nhớ chung, được tạo thành từ hai cấp độ bộ đệm, kết nối và bộ điều khiển bộ nhớ. Hình 3a hiển thị sơ đồ khối của thiết bị điện toán gia đình Evergreen.

Một đơn vị tính toán bao gồm ba *công cụ thực thi* , một *bộ nhớ cục bộ* và một *tệp đăng ký* . Ba công cụ thực thi, được gọi là công cụ *điều khiển luồng* (CF), *logic số học* (ALU) và *kết cấu* (TEX), được dành để thực thi các phần khác nhau của nhị phân hạt nhân OpenCL, được gọi lần lượt là các mệnh đề CF, ALU và TEX (xem Phần 3.2). Sơ đồ khối của đơn vị tính toán được minh họa trong Hình 3b. Công cụ ALU chứa một tập hợp *các lõi luồng* , mỗi lõi dành cho việc thực hiện các phép toán số học của một mục công việc. Các lệnh ALU được tổ chức dưới dạng gói VLIW 5 chiều, được tạo tại thời điểm biên dịch. Mỗi lệnh trong gói VLIW được thực thi trên một trong 5 làn VLIW tạo thành lõi luồng.

GPU Evergreen định nghĩa khái niệm *mặt sóng* là một nhóm các mục công việc thực thi theo kiểu Đa dữ liệu một lệnh (SIMD). Mỗi lệnh được thực thi đồng thời bởi mọi mục công việc bao gồm mặt sóng, mặc dù mỗi mục công việc sử dụng dữ liệu riêng tư của nó để tính toán. Mô hình này đơn giản hóa phần cứng tìm nạp lệnh bằng cách triển khai một giao diện người dùng chung cho toàn bộ mặt sóng.

**3.2 Kiến trúc tập lệnh Evergreen (ISA)**

Khi trình mô phỏng chức năng GPU nhận hạt nhân OpenCL để thực thi, một vòng lặp mô phỏng sẽ bắt đầu bằng cách tìm nạp, giải mã và thực thi các lệnh Evergreen. Định dạng cơ bản của ISA AMD Evergreen có thể được quan sát trong mã mẫu từ Hình 4.

Hội Evergreen sử dụng định dạng dựa trên mệnh đề. Việc thực thi kernel bắt đầu bằng lệnh CF. Lệnh CF ảnh hưởng đến luồng điều khiển chương trình chính (như trường hợp của lệnh CF 03), ghi dữ liệu vào bộ nhớ chung (04) hoặc chuyển điều khiển sang mệnh đề phụ, chẳng hạn như mệnh đề ALU (00, 02) hoặc TEX khoản (01). Mệnh đề ALU chứa các hướng dẫn thực hiện các phép toán logic số học và truy cập bộ nhớ cục bộ, trong khi mệnh đề TEX được dành riêng cho các hoạt động đọc bộ nhớ chung.

Hướng dẫn ALU được đóng gói thành gói VLIW. Một gói VLIW được chạy lần lượt trên lõi luồng, trong đó mỗi nhãn lệnh ALU phản ánh làn VLIW được gán cho lệnh đó. Toán hạng lệnh ALU có thể là bất kỳ đầu ra nào từ gói VLIW được thực thi trước đó bằng cách sử dụng các thanh ghi đặc biệt *Véc-tơ trước* (PV) hoặc *Vô hướng trước* (PS). Cuối cùng, *bộ nhớ không đổi* là bộ lưu trữ bổ sung có thể truy cập toàn cầu được khởi tạo bởi CPU, bộ nhớ này cũng có thể được sử dụng làm toán hạng lệnh ALU (KC).

Từ cuộc thảo luận ở trên về các đặc điểm của Evergreen ISA, chúng ta có thể quan sát thấy một số điểm khác biệt quan trọng khi làm việc với các ngôn ngữ trung cấp cấp cao hơn, chẳng hạn như IL của AMD [4] hoặc PTX của NVIDIA [6]. Ví dụ: trong ISA Evergreen của AMD có một số lượng hạn chế các thanh ghi có mục đích chung , do đó, có những hạn chế về cách hình thành các gói VLIW và có các quy tắc cụ thể để nhóm các mệnh đề hình thành các lệnh máy. Nói chung, có nhiều thuộc tính của ISA được máy chạy trực tiếp mà không cần xem xét làm việc với ngôn ngữ trung gian. Do đó, độ chính xác hiệu suất đáng kể có thể đạt được bằng mô phỏng ISAlevel .

*3.2.1 Mô hình thực thi hạt nhân*

Khi hạt nhân OpenCL được khởi chạy bởi chương trình máy chủ, cấu hình NDRange sẽ được cung cấp cho GPU. Sau đó, các nhóm làm việc được tạo và gán lần lượt cho các đơn vị tính toán khi chúng có sẵn tài nguyên thực thi. Số lượng nhóm công việc có thể được chỉ định cho một đơn vị điện toán được xác định bởi bốn giới hạn phần cứng: *i* ) số lượng nhóm công việc tối đa được hỗ trợ trên mỗi đơn vị điện toán, *ii* ) số mặt sóng tối đa trên mỗi đơn vị điện toán, *iii* ) số lượng thanh ghi trên một đơn vị điện toán và *iv* ) dung lượng bộ nhớ cục bộ trên một đơn vị điện toán . Tối đa hóa số lượng nhóm làm việc được chỉ định trên mỗi đơn vị điện toán là một quyết định nhạy cảm về hiệu suất và có thể được đánh giá trên Multi2Sim.

Mỗi nhóm công việc được chỉ định cho một đơn vị điện toán được phân chia thành các mặt sóng, sau đó được đặt vào *nhóm mặt sóng sẵn sàng* . Công cụ CF chọn các mặt sóng từ nhóm mặt sóng để thực thi, dựa trên thuật toán *lập lịch mặt sóng* . Mặt sóng mới bắt đầu chạy mệnh đề CF chính của tệp nhị phân hạt nhân OpenCL và sau đó sinh ra các mệnh đề ALU và TEX thứ cấp. Thuật toán lập kế hoạch mặt sóng là một tham số nhạy cảm về hiệu suất khác, có thể được đánh giá bằng Multi2Sim.

Khi một mặt sóng được trích xuất khỏi nhóm, nó chỉ được chèn trở lại khi lệnh CF được thực thi hoàn thành. Điều này đảm bảo rằng chỉ có một lệnh CF duy nhất đang bay vào bất kỳ lúc nào đối với một mặt sóng nhất định, tránh nhu cầu dự đoán nhánh hoặc thực hiện suy đoán trong trường hợp điều khiển luồng bị ảnh hưởng. Hình phạt về hiệu suất đối với việc tuần tự hóa này được ẩn đi bằng cách chồng chéo việc thực hiện các mặt sóng khác nhau. Việc xác định mức độ xảy ra tình trạng thực thi chồng chéo và nguyên nhân gây tắc nghẽn là những lợi ích bổ sung của việc mô phỏng thực thi bằng Multi2Sim.

*3.2.2 Sự phân kỳ của hạng mục công việc*

Trong mô hình thực thi SIMD, *sự phân kỳ mục công việc* là hiệu ứng phụ được tạo ra khi lệnh nhánh có điều kiện được giải quyết khác nhau đối với bất kỳ mục công việc nào trong mặt sóng. Để giải quyết sự phân kỳ của mục công việc trong quá trình thực thi SIMD, ISA Evergreen cung cấp cho mỗi mặt sóng một *mặt nạ hoạt động* . Mặt nạ hoạt động là một bản đồ bit, trong đó mỗi bit thể hiện trạng thái hoạt động của một mục công việc riêng lẻ trong mặt sóng. Nếu một mục công việc được gắn nhãn là không hoạt động, kết quả của bất kỳ tính toán số học nào được thực hiện trong lõi luồng liên quan của nó sẽ bị bỏ qua, ngăn mục công việc thay đổi trạng thái kernel.

Chiến lược phân kỳ mục công việc này cố gắng hội tụ tất cả các mục công việc lại với nhau trên tất cả các đường dẫn thực thi có thể, chỉ cho phép những mục công việc đang hoạt động có điều kiện thực thi phù hợp với luồng lệnh hiện được tìm nạp để tiếp tục thực thi. Để hỗ trợ các lệnh gọi thủ tục và điều kiện lồng nhau, một *ngăn xếp mặt nạ hoạt động* được sử dụng để đẩy và bật các mặt nạ hoạt động, sao cho mặt nạ hoạt động ở trên cùng của ngăn xếp luôn đại diện cho mặt nạ hoạt động của các mục công việc hiện đang thực thi. Khi sử dụng Multi2Sim, các nhà nghiên cứu có sẵn số liệu thống kê liên quan đến sự phân kỳ của mục công việc (xem Phần 4.3).

**3.3 Quy trình hướng dẫn**

Trong một đơn vị tính toán, các công cụ CF, ALU và TEX được tổ chức dưới dạng các đường dẫn hướng dẫn. Hình 5 trình bày sơ đồ khối của đường dẫn lệnh của mỗi động cơ. Trong mỗi quy trình, các quyết định về chính sách lập lịch, độ trễ và kích thước bộ đệm phải được đưa ra. Những yếu tố tinh vi này có ý nghĩa về hiệu suất và mang lại cơ hội khác cho các nhà nghiên cứu được hưởng lợi từ việc thử nghiệm các quyết định thiết kế trong Multi2Sim.

Công cụ CF (Hình 5a) chạy mệnh đề CF của hạt nhân OpenCL. Giai đoạn *tìm nạp* chọn một mặt sóng mới từ nhóm mặt sóng trong mỗi chu kỳ, chuyển đổi giữa chúng theo mức độ chi tiết của một lệnh CF duy nhất. Các hướng dẫn từ các mặt sóng khác nhau được diễn giải ở giai đoạn *giải mã* theo kiểu vòng tròn. Khi lệnh CF kích hoạt mệnh đề phụ, công cụ thực thi tương ứng (công cụ ALU hoặc TEX) sẽ được phân bổ và lệnh CF vẫn ở giai đoạn *thực thi* cho đến khi mệnh đề phụ hoàn thành. Các lệnh CF khác từ các mặt sóng khác có thể được thực hiện tạm thời, miễn là chúng không yêu cầu một công cụ thực thi bận rộn. Việc thực thi lệnh CF (bao gồm tất cả các lệnh chạy trong mệnh đề phụ, nếu có) kết thúc theo thứ tự trong giai đoạn giai đoạn *hoàn chỉnh* . Mặt sóng được đưa trở lại nhóm mặt sóng, khiến nó lại trở thành ứng cử viên cho việc tìm nạp lệnh. Việc ghi bộ nhớ chung được chạy không đồng bộ trong chính công cụ CF mà không yêu cầu công cụ phụ.

Công cụ ALU được dành cho việc thực thi các mệnh đề ALU từ mặt sóng được phân bổ (Hình 5b). Sau giai đoạn *tìm nạp* và *giải mã* , các lệnh VLIW đã giải mã sẽ được đặt vào *bộ đệm gói VLIW* . Giai đoạn *đọc* sử dụng gói VLIW và đọc toán hạng nguồn từ tệp thanh ghi và/hoặc bộ nhớ cục bộ cho từng mục công việc trong mặt sóng. Giai đoạn *thực thi* sẽ phát hành một phiên bản của gói VLIW cho mỗi lõi luồng trong mỗi chu kỳ. Số lượng lõi luồng trong đơn vị điện toán có thể nhỏ hơn số lượng mục công việc trong mặt sóng. Do đó, mặt sóng được chia thành *các mặt sóng phụ* , trong đó mỗi mặt sóng phụ chứa nhiều mục công việc bằng số lõi luồng trong một đơn vị điện toán. Kết quả tính toán được ghi trở lại toán hạng đích (tệp đăng ký hoặc bộ nhớ cục bộ) ở giai đoạn *ghi* .

Công cụ TEX (Hình 5c) được dành cho việc thực thi các lệnh tìm nạp bộ nhớ chung trong mệnh đề TEX. Các byte lệnh TEX được lưu trữ vào bộ đệm lệnh TEX sau khi được tìm nạp và giải mã. Địa chỉ bộ nhớ cho từng mục công việc trong mặt sóng được đọc từ tệp đăng ký và yêu cầu đọc tới hệ thống phân cấp bộ nhớ chung được thực hiện ở giai đoạn *đọc* . Việc đọc bộ nhớ chung đã hoàn thành sẽ được xử lý theo thứ tự ở giai đoạn *ghi* . Dữ liệu được tìm nạp được lưu trữ vào các vị trí tương ứng của tệp đăng ký cho từng mục công việc. Thời gian tồn tại của một lần đọc bộ nhớ được mô hình hóa chi tiết trong toàn bộ hệ thống phân cấp bộ nhớ chung, như được chỉ định trong các phần sau.

**3.4 Hệ thống con bộ nhớ**

Hệ thống con bộ nhớ GPU chứa các thành phần khác nhau để lưu trữ và truyền dữ liệu. Với Multi2Sim, hệ thống con bộ nhớ có khả năng cấu hình cao, bao gồm các cài đặt có thể tùy chỉnh về số lượng cấp độ bộ đệm, dung lượng bộ nhớ, kích thước khối, số lượng ngân hàng và cổng. Mô tả các thành phần bộ nhớ cho mô hình Evergreen như sau:

**Tệp đăng ký (GPR)** . Multi2Sim cung cấp một mô hình không có tranh chấp về quyền truy cập tệp đăng ký. Trong một chu kỳ nhất định, thanh ghi có thể được truy cập đồng thời bởi các động cơ TEX và ALU bằng các mặt sóng khác nhau. Các hạng mục công việc bên trong và giữa các mặt sóng luôn truy cập vào các bộ thanh ghi khác nhau.

**Bộ nhớ cục bộ** . Một mô-đun bộ nhớ cục bộ riêng biệt có trong mỗi đơn vị điện toán và được mô hình hóa trong Multi2Sim với độ trễ, số lượng ngân hàng, cổng và kích thước khối phân bổ có thể định cấu hình. Trong nhân OpenCL, quyền truy cập vào bộ nhớ cục bộ được lập trình viên xác định bằng cách chỉ định phạm vi của biến, quyền truy cập của biến sau đó được biên dịch thành các hướng dẫn lắp ráp riêng biệt. Tranh giành bộ nhớ cục bộ được mô hình hóa bằng cách tuần tự hóa các truy cập vào cùng một ngân hàng bộ nhớ bất cứ khi nào không có cổng đọc hoặc ghi. Ngoài ra, việc kết hợp quyền truy cập bộ nhớ được xem xét bằng cách nhóm các quyền truy cập đó từ các mục công việc khác nhau vào cùng một khối bộ nhớ.

**Bộ nhớ toàn cầu** . Tất cả các đơn vị tính toán đều có thể truy cập bộ nhớ chung GPU. Nó được trình bày cho lập trình viên dưới dạng một phạm vi bộ nhớ riêng biệt và được triển khai dưới dạng phân cấp bộ nhớ được quản lý bởi phần cứng nhằm giảm độ trễ truy cập. Trong Multi2Sim, hệ thống phân cấp bộ nhớ chung có số lượng cấp độ bộ đệm và kết nối có thể định cấu hình. Một cấu hình khả thi được hiển thị trong Hình 6a, sử dụng bộ đệm L1 riêng cho mỗi đơn vị điện toán và nhiều bộ đệm L2 được chia sẻ giữa các tập hợp con của đơn vị điện toán. Bộ đệm L1 thường cung cấp thời gian truy cập tương tự như bộ nhớ cục bộ, nhưng chúng được quản lý một cách minh bạch bằng phần cứng, tương tự như cách quản lý phân cấp bộ nhớ trên CPU.

**Các mạng kết nối** . Mỗi bộ đệm trong hệ thống phân cấp bộ nhớ chung được kết nối với bộ đệm cấp thấp hơn (hoặc bộ nhớ chung) bằng mạng kết nối. Các kết nối được tổ chức dưới dạng kết nối điểm-điểm bằng cách sử dụng một bộ chuyển mạch, có sơ đồ khối kiến trúc được trình bày trong Hình 6b. Một switch chứa hai mạng con bên trong tách rời nhau, mỗi mạng dành cho việc chuyển gói theo các hướng ngược nhau.

**Hàng đợi truy cập bộ đệm** . Mỗi bộ nhớ đệm có một bộ đệm nơi các yêu cầu truy cập được xếp vào hàng đợi, như trong Hình 6c. Một mặt, bộ đệm truy cập cho phép ghi không đồng bộ nhằm ngăn chặn tình trạng ngừng hoạt động trong quy trình hướng dẫn. Mặt khác, việc kết hợp truy cập bộ nhớ được xử lý trong bộ đệm truy cập ở mọi cấp độ của hệ thống phân cấp bộ nhớ chung (cả bộ đệm và bộ nhớ chung). Mỗi chuỗi các mục tiếp theo trong hàng đợi truy cập đọc hoặc ghi vào cùng một khối bộ đệm được nhóm thành một truy cập bộ nhớ thực tế duy nhất. Mức độ kết hợp phụ thuộc vào kích thước khối bộ nhớ, kích thước hàng đợi truy cập và kiểu truy cập bộ nhớ, đồng thời là số liệu rất nhạy cảm về hiệu suất có thể đo lường được bằng Multi2Sim.

**4. ĐÁNH GIÁ THỰC NGHIỆM**

Phần này trình bày một tập hợp các thử nghiệm nhằm xác nhận và chứng minh một loạt các tính năng mô phỏng chức năng và kiến trúc có sẵn với Multi2Sim. Tất cả các mô phỏng đều dựa trên mô hình GPU cơ bản giống với GPU AMD Radeon 5870 thương mại, có thông số phần cứng được tóm tắt trong Bảng 1.

Đối với các nghiên cứu về hiệu suất của trình mô phỏng, các mô phỏng được chạy trên một máy có bốn bộ xử lý Intel Xeon lõi tứ (2,27GHz, bộ nhớ đệm 8MB, 24GB DDR3). Các đánh giá thử nghiệm được thực hiện bằng cách sử dụng một tập hợp con các ứng dụng AMD OpenCL SDK [1], thể hiện nhiều hành vi ứng dụng và kiểu truy cập bộ nhớ [16]. Các ứng dụng được thảo luận trong bài viết này được liệt kê trong Bảng 2, trong đó chúng tôi bao gồm mô tả ngắn gọn về các chương trình và các đặc điểm của tập dữ liệu đầu vào tương ứng.

**4.1 Xác nhận**

Phương pháp xác thực của chúng tôi để thiết lập độ trung thực của trình mô phỏng GPU đã xem xét tính chính xác của cả mô hình mô phỏng chức năng và kiến trúc, mặc dù chúng tôi tuân theo hai phương pháp xác thực khác nhau. Đối với trình mô phỏng chức năng, tính chính xác của bộ giải mã lệnh được xác thực bằng cách so sánh mã đã được phân tách với đầu ra Evergreen do trình biên dịch AMD tạo ra. Chúng tôi cũng xác thực tính chính xác của việc thực thi từng điểm chuẩn bằng cách so sánh đầu ra của ứng dụng mô phỏng với đầu ra của ứng dụng chạy trực tiếp trên CPU. Tất cả các mô phỏng đều tạo ra kết quả chính xác về mặt chức năng cho tất cả các chương trình được nghiên cứu và đưa vào các bộ vấn đề.

Về độ trung thực của mô hình kiến trúc, kết quả hiệu suất của Multi2Sim đã được so sánh với hiệu suất thực thi gốc (ở đây đề cập đến phần cứng Radeon 5870 thực tế), sử dụng mười kích cỡ đầu vào khác nhau trong phạm vi được hiển thị trong Bảng 2 (cột *Phạm vi đầu vào* ). Vì mô hình kiến trúc của chúng tôi dựa trên chu kỳ và quá trình thực thi gốc được đo bằng thời gian thực thi hạt nhân nên việc so sánh trực tiếp các số liệu của chúng tôi là một thách thức. Để chuyển đổi các chu kỳ mô phỏng thành thời gian, chúng tôi sử dụng tần số xung nhịp ALU được ghi lại là 850 MHz của phần cứng 5870. Thời gian thực thi gốc được tính là thời gian trung bình của 1000 lần thực thi hạt nhân cho mỗi điểm chuẩn. Thời gian thực thi hạt nhân gốc được đo bằng cách sử dụng trình lược tả AMD APP [2]. Thời gian thực thi do trình lược tả APP cung cấp không bao gồm các chi phí chung như thiết lập hạt nhân và I/O thiết bị chủ [2]. Hình 7a và Hình 7b lần lượt mô phỏng xu hướng hiệu suất thời gian thực thi và thời gian thực thi gốc (chỉ có bốn điểm chuẩn được hiển thị cho rõ ràng). Hình 7c cho thấy phần trăm chênh lệch về hiệu suất đối với nhiều lựa chọn điểm chuẩn hơn. Giá trị được hiển thị cho mỗi điểm chuẩn trong Hình 7c là giá trị trung bình của sai số phần trăm tuyệt đối cho mỗi đầu vào của điểm chuẩn. Đối với những trường hợp độ chính xác của mô phỏng giảm, Hình 8 hiển thị các xu hướng chi tiết, dẫn đến phân tích sau. Trong Hình 8a, chúng tôi hiển thị mối tương quan giữa thời gian thực thi gốc và thời gian thực hiện mô phỏng cho các điểm chuẩn được nghiên cứu. Đối với một số điểm chuẩn (ví dụ: *Biểu đồ* hoặc *Gauss đệ quy* ), thời gian thực hiện thay đổi đáng kể. Tuy nhiên, chúng tôi vẫn thấy mối tương quan chặt chẽ giữa từng điểm thực thi gốc và kết quả mô phỏng liên quan của chúng cho tất cả các điểm chuẩn. Nói cách khác, sự thay đổi về quy mô vấn đề đối với điểm chuẩn có tác động tương đối đến hiệu suất đối với cả quá trình thực thi gốc và mô phỏng. Đường xu hướng tuyến tính được biểu diễn bằng thuật toán khớp đường cong nhằm giảm thiểu khoảng cách bình phương giữa mỗi điểm dữ liệu và chính nó. Đối với các điểm chuẩn được mô hình hóa chính xác bằng trình mô phỏng, các điểm dữ liệu nằm trên đường 45 *◦* . Lý do xuất hiện các độ dốc khác nhau có thể là do thiếu sự thể hiện chính xác về hệ thống phân cấp bộ nhớ trong GPU 5870, bao gồm các yếu tố sau: **Thiết kế đường dẫn bộ nhớ chuyên dụng.** AMD Radeon 5870 bao gồm hai đường dẫn từ đơn vị tính toán đến bộ nhớ [2], mỗi đường dẫn có đặc tính hiệu suất khác nhau. Đường *dẫn nhanh* chỉ thực hiện các thao tác cơ bản, chẳng hạn như tải và lưu trữ các loại dữ liệu 32 bit. Đường *dẫn hoàn chỉnh* hỗ trợ các hoạt động nâng cao bổ sung, bao gồm nguyên tử và lưu trữ cho các loại dữ liệu phụ 32 bit. Thiết kế này không được dùng nữa trong các kiến trúc GPU sau này để có bố cục thông thường hơn [17], tương tự như bố cục hiện được triển khai trong Multi2Sim. **Kết nối bộ đệm.** Thông số kỹ thuật của mạng kết nối giữa bộ đệm L1 và L2 chưa được công bố. Chúng tôi sử dụng một phép tính gần đúng trong đó bốn bộ nhớ đệm L2 được chia sẻ giữa các đơn vị điện toán (Bảng 1). **Thông số bộ đệm.** Độ trễ và tính liên kết của các cấp độ khác nhau của hệ thống phân cấp bộ đệm không được biết đến. Một số nguyên nhân dẫn đến sự thiếu chính xác của mô phỏng có thể được quy cho các tham số bộ nhớ đệm, như trong Hình 8, trong đó sai số phần trăm là tối thiểu đối với các trường hợp trong đó tỷ lệ trúng bộ nhớ đệm gốc và tỷ lệ trúng bộ nhớ đệm mô phỏng ít khác nhau nhất.

**4.2 Tốc độ mô phỏng**

Đối với các điểm chuẩn được sử dụng trong bài viết này, chi phí mô phỏng của Multi2Sim được biểu thị trong Hình 9 dưới dạng hàm giảm tốc độ theo thời gian thực thi gốc. Tốc độ chậm mô phỏng chức năng trung bình là 8700 *×* (113 giây) và thời gian mô phỏng kiến trúc trung bình là 44000 *×* (595 giây). Cần lưu ý rằng thời gian mô phỏng không nhất thiết liên quan đến thời gian thực hiện tự nhiên (ví dụ: mô phỏng một lệnh có độ trễ 100 chu kỳ nhanh hơn mô phỏng mười lệnh 1 chu kỳ), do đó các kết quả này chỉ nhằm mục đích cung cấp một số mẫu đại diện cho chi phí mô phỏng.

Hiệu suất mô phỏng cũng đã được đánh giá cho mô phỏng kiến trúc trên GPGPUSim , trình mô phỏng GPU dựa trên NVIDIA [10]. Trình mô phỏng này đã được sử dụng làm hỗ trợ thử nghiệm cho các nghiên cứu gần đây về tính toán GPU, chẳng hạn như khám phá các triển khai bộ điều khiển bộ nhớ thay thế [18] và nhóm động các luồng (mục công việc) để giảm thiểu hình phạt phân kỳ luồng [15]. Để thực hiện so sánh này, điểm chuẩn SDK APP đã được điều chỉnh để chạy trên GPGPUSim . Hình 9c cho thấy sự chậm lại về hiệu suất so với thực thi gốc, trung bình khoảng 90000 *×* (1350 giây).

**4.3 Đặc tính điểm chuẩn**

Là một nghiên cứu điển hình về mô phỏng GPU, phần này trình bày đặc điểm ngắn gọn về các điểm chuẩn OpenCL được thực hiện trên Multi2Sim, dựa trên phân loại lệnh, chiếm dụng gói VLIW và phân kỳ luồng điều khiển. Các số liệu thống kê này có tính chất động và được Multi2Sim báo cáo như một phần của báo cáo mô phỏng.

Hình 10a cho thấy các hỗn hợp lệnh Evergreen được thực thi bởi mỗi hạt nhân OpenCL. Các loại lệnh là các lệnh luồng điều khiển (nhảy, thao tác ngăn xếp và đồng bộ hóa), đọc bộ nhớ chung, ghi bộ nhớ chung, truy cập bộ nhớ cục bộ và các phép toán logic số học. Các phép toán logic số học tạo thành phần lớn các lệnh được thực thi (đây là những khối lượng công việc thân thiện với GPU).

Hình 10b biểu thị mức chiếm dụng trung bình của các gói VLIW được thực thi trong lõi luồng của công cụ ALU của GPU. Nếu lệnh VLIW sử dụng ít hơn 5 vị trí thì sẽ có các làn VLIW nhàn rỗi trong lõi luồng, dẫn đến việc sử dụng không đúng mức các tài nguyên thực thi có sẵn. Trình biên dịch Evergreen cố gắng tối đa hóa việc chiếm chỗ khe VLIW, nhưng có một giới hạn trên được áp đặt bởi tính song song ở cấp độ lệnh có sẵn trong mã hạt nhân. Kết quả cho thấy chúng tôi hiếm khi sử dụng hết 5 slot (ngoại trừ *SobelFilter* nhờ tỷ lệ hướng dẫn ALU cao) và trường hợp xấu nhất là chỉ có một khe được lấp đầy thường xuyên xảy ra.

Cuối cùng, Hình 10c minh họa hiệu ứng phân kỳ luồng điều khiển giữa các hạng mục công việc. Khi các mục công việc trong mặt sóng thực thi theo kiểu SIMD khác nhau theo các điều kiện nhánh, toàn bộ mặt sóng phải đi qua tất cả các đường dẫn thực thi có thể có. Do đó, sự phân kỳ mục công việc thường xuyên có tác động tiêu cực đến hiệu suất. Đối với mỗi điểm chuẩn trong Hình 10c, mỗi bước màu trong một thanh biểu thị một đường dẫn luồng điều khiển khác nhau thông qua chương trình. Nếu một thanh có một bước duy nhất thì chỉ có một đường dẫn được thực hiện bởi tất cả các hạng mục công việc cho hạt nhân đó. Nếu có *n* bước thì *n* đường dẫn luồng điều khiển khác nhau sẽ được thực hiện bởi các hạng mục công việc khác nhau. Lưu ý rằng các màu khác nhau được sử dụng ở đây với mục đích duy nhất là phân định các bước của thanh, nhưng không có ý nghĩa cụ thể nào được gán cho mỗi màu. Kích thước của mỗi bước biểu thị tỷ lệ phần trăm của các mục công việc đã sử dụng đường dẫn luồng điều khiển đó cho hạt nhân. Kết quả hiển thị điểm chuẩn với các đặc điểm phân kỳ sau:

*•* Không có sự phân kỳ luồng điều khiển nào cả ( *URNG* , *DCT* ).

*•* Nhóm phân kỳ có kích thước logarit giảm dần do số lần lặp vòng lặp khác nhau ( *Giảm* ,

*DwtHaar1D* ).

*•* Nhiều nhóm phân kỳ tùy thuộc vào dữ liệu đầu vào ( *BinomialOption* 2 ).

**4.4 Thăm dò kiến trúc**

Mô hình GPU kiến trúc được cung cấp trong Multi2Sim cho phép các nhà nghiên cứu thực hiện đánh giá không gian thiết kế lớn. Là một mẫu về tính linh hoạt của mô phỏng, phần này trình bày ba nghiên cứu điển hình, trong đó hiệu suất thay đổi đáng kể đối với các giá trị tham số đầu vào khác nhau. Trong mỗi trường hợp, chúng tôi so sánh hai điểm chuẩn về độ nhạy kiến trúc của chúng. Hiệu suất được đo bằng cách sử dụng số lượng lệnh trên mỗi chu kỳ (IPC), trong đó số lượng lệnh được tăng thêm một cho toàn bộ mặt sóng, bất kể số lượng mục công việc bao gồm.

Hình 11a cho thấy tỷ lệ hiệu suất tương ứng với số lượng đơn vị tính toán. Tổng băng thông bộ nhớ do bộ nhớ chung cung cấp được chia sẻ bởi tất cả các đơn vị điện toán, do đó việc tăng số lượng đơn vị điện toán sẽ làm giảm băng thông khả dụng cho mỗi nhóm công việc được thực thi . Băng thông bộ nhớ khả dụng cho thiết bị trong thử nghiệm này chỉ tăng giữa số lượng đơn vị tính toán là bội số của 5 khi thêm L2 mới (Bảng 1). Khi tổng băng thông cạn kiệt, xu hướng (như được thấy trong khoảng từ 10-15 đến 15-20 đơn vị tính toán) sẽ không thay đổi. Điểm này được nhận thấy rõ ràng khi chúng tôi tăng số lượng đơn vị điện toán trong các hạt nhân chuyên sâu về điện toán với tỷ lệ lệnh ALU-to-Fetch cao (ví dụ: *URNG* ) và ít hơn trong các điểm chuẩn sử dụng nhiều bộ nhớ (ví dụ: *Histogram* ).

Hình 11b trình bày hiệu suất đạt được bằng cách thay đổi số lượng lõi luồng trên mỗi đơn vị điện toán. Trong *tùy chọn nhị thức* kernel, chúng tôi quan sát một hàm bước, trong đó mỗi bước tương ứng với bội số của kích thước mặt sóng (64). Hành vi này là do số lượng lõi luồng xác định số lượng mặt sóng phụ (hoặc khe ghép kênh thời gian) mà lõi luồng xử lý cho mỗi gói VLIW. Khi số lượng lõi luồng tăng làm giảm số lượng mặt sóng phụ (ví dụ: 15 đến 16, 21 đến 22 và 31 đến 32), hiệu suất sẽ được cải thiện. Khi số lượng lõi luồng khớp với số lượng mục công việc trên mỗi mặt sóng, tình trạng thắt cổ chai do việc sử dụng lõi luồng được tuần tự hóa sẽ biến mất. Hiệu ứng này không được quan sát thấy đối với *ScanLargeArrays* do công suất sử dụng mặt sóng thấp hơn.

Hình 11c thể hiện tác động của việc tăng kích thước bộ đệm L1. Đối với các điểm chuẩn thiếu vị trí tạm thời và thể hiện các bước truy cập lớn trong luồng dữ liệu, hiệu suất không nhạy cảm với việc tăng kích thước bộ đệm, như đã thấy trong *Giảm* . Ngược lại, điểm chuẩn có tính cục bộ nhạy cảm hơn với những thay đổi về kích thước bộ đệm L1, như đã quan sát đối với *FloydWarshall* .

**5. CÔNG VIỆC LIÊN QUAN**

Mặc dù có sẵn nhiều trình mô phỏng CPU hoàn thiện ở nhiều cấp độ khác nhau, nhưng các trình mô phỏng GPU vẫn còn ở giai đoạn sơ khai. Tiếp tục có nhu cầu ngày càng tăng về các trình mô phỏng GPU kiến trúc để mô hình hóa GPU ở cấp độ ISA. Và trong tương lai gần , chúng ta sẽ thấy nhu cầu cấp thiết hơn về một khung mô phỏng không đồng nhất CPU-GPU thực sự. Phần này tóm tắt ngắn gọn các trình mô phỏng nhắm mục tiêu GPU hiện có.

Barra [12] là một trình mô phỏng chức năng cấp ISA nhắm vào GPU NVIDIA G80. Nó chạy các tệp thực thi CUDA mà không có bất kỳ sửa đổi nào. Do đặc tả ISA G80 của NVIDIA không được công bố rộng rãi nên trình mô phỏng dựa vào ISA được thiết kế ngược do một dự án học thuật khác cung cấp. Tương tự như cách tiếp cận của chúng tôi, Barra chặn các lệnh gọi API đến thư viện CUDA và định tuyến lại chúng đến trình mô phỏng. Thật không may, nó bị giới hạn ở mô phỏng chức năng GPU, thiếu mô hình mô phỏng kiến trúc.

GPGPUSim [10] là một trình mô phỏng chi tiết mô hình kiến trúc GPU tương tự như kiến trúc của NVIDIA. Nó bao gồm lõi đổ bóng, các kết nối, lập lịch khối luồng ( nhóm làm việc ) và phân cấp bộ nhớ. Multi2Sim mô hình hóa ISA và kiến trúc GPU khác (Evergreen). GPGPUSim có thể cung cấp cho chúng tôi cái nhìn sâu sắc quan trọng về các vấn đề thiết kế dành cho GPU. Tuy nhiên, Multi2Sim cũng hỗ trợ mô phỏng CPU trong cùng một công cụ cho phép nghiên cứu kiến trúc bổ sung về các kiến trúc không đồng nhất.

Ocelot [13] là một khung biên dịch động và mô phỏng chức năng được sử dụng rộng rãi, hoạt động ở cấp độ ISA ảo. Lấy mã CUDA PTX của NVIDIA làm đầu vào, nó có thể mô phỏng hoặc dịch động mã đó sang nhiều nền tảng như CPU x86, GPU NVIDIA và GPU AMD. Ocelot có các mục tiêu khác với mô phỏng kiến trúc GPU, do đó, có một chức năng mở rộng không được Multi2Sim cung cấp hoặc nhắm mục tiêu, khiến chúng trở thành công cụ bổ sung.

Khi so sánh với tác phẩm trước, Multi2Sim độc đáo ở các khía cạnh sau. Đầu tiên, nó mô hình hóa ISA gốc của GPU có bán trên thị trường. Thứ hai, nó cung cấp mô phỏng kiến trúc của GPU thực với độ chính xác dễ điều khiển. Thứ ba, Multi2Sim là khung mô phỏng không đồng nhất CPU-GPU, có thể được sử dụng để đánh giá các kiến trúc sắp tới trong đó CPU và GPU được hợp nhất trên silicon và chia sẻ không gian địa chỉ bộ nhớ chung [8].

**6. KẾT LUẬN**

Trong bài viết này, chúng tôi đã trình bày Multi2Sim, một khung mô phỏng chính thức hỗ trợ cả mô phỏng kiến trúc chi tiết và chức năng nhanh cho CPU x86 và GPU Evergreen ở cấp độ ISA. Nó là mô-đun, có thể cấu hình đầy đủ và dễ sử dụng. Bộ công cụ này được duy trì tích cực và có sẵn dưới dạng dự án mã nguồn mở miễn phí tại *www.multi2sim.org* , cùng với các gói điểm chuẩn, hướng dẫn sử dụng hoàn chỉnh, danh sách gửi thư và diễn đàn đang hoạt động.

Công việc đang diễn ra đối với Multi2Sim bao gồm việc mở rộng hỗ trợ điểm chuẩn bằng cách tăng phạm vi phủ sóng ISA của Evergreen. Các bản phát hành trong tương lai sẽ bao gồm một mô hình cho kiến trúc AMD Fusion, trong đó CPU và GPU chia sẻ hệ thống phân cấp bộ nhớ và không gian địa chỉ chung trên toàn cầu. Việc hỗ trợ bộ nhớ dùng chung cho các kiến trúc không đồng nhất làm nổi bật tiềm năng của Multi2Sim, vì không có trình mô phỏng nào khác có thể cung cấp số liệu thống kê kiến trúc hữu ích trong loại môi trường này. Sự phát triển hiện tại cũng bao gồm hỗ trợ cho các ứng dụng OpenGL và khám phá các phần mở rộng ngôn ngữ OpenCL. Vì Multi2Sim hiện đang được sử dụng bởi một số nhóm nghiên cứu hàng đầu nên chúng tôi tin rằng đây là cơ hội tuyệt vời để đẩy nhanh nghiên cứu về các kiến trúc song song, không đồng nhất.

**Sự nhìn nhận**

Công việc này được hỗ trợ một phần bởi Giải thưởng NSF EEC-0946463 và thông qua sự hỗ trợ và quyên góp từ AMD và NVIDIA. Các tác giả cũng xin cảm ơn Norman Rubin (AMD) vì những lời khuyên và phản hồi của ông cho công việc này.