SUMMER CONFERENCE 2023

10주년기념

(사)ICT플랫폼학회 2023 하계학술대회 논문집

PTL Volume 10-1 : ISSN 2288 -8195

2023년 7월 27일 (목) - 28일 (금)

SETEC 컨벤션센터 3세미나실

◎ 주 최 ICTPS ICT 플랫폼학회

◎ 후 원 FUNCTION12 LG씨엔에스 SJ정보통신 SK broadband 국제 펫 대체의학협회 대보정보통신 대신정보통신 세림TSG 시야 INSIGHT 쌍용정보통신주식회사 SsangYong Information & Communications Corp. ICTWAY 아이씨티웨이(주) 올포랜드 인라스 ENLACE Genians KITC 한국IT컨설팅 KOREA IT CONSULTING 한국정보공학(주) Korea Information Engineering Services HUEVERTECH 국산ICT기업협의회

ICTPS

16:00~16:20	휴식	
16:20~17:00	**15. CUDA 프로그래밍의 코드 빌딩 블록에서 저전력 소비 코드 패턴 식별** 진예진, 김현태, 김영철(홍익대학교) **16. IoT 대상 봇넷 공격 탐지 프로세스 설계** 진호준, 전규현, 전승호, 서정택(가천대학교) **17. DNP 3.0 전력계통 대상 AI 기반 이상탐지 시스템 제안** 지일환, 김동현, 전승호, 서정택(가천대학교) **18. ICS환경에서 기계학습 기반 공격 패킷 생성 방법론 연구** 이주현, 전승호, 송인성, 서정택(가천대학교) **19. 다중 로봇 운영에 대한 연구** 김동원(인하공업전문대학)	**일반 부분3** 좌장 : 여상수(목원대)
17:00~17:40	**20. 데이터 스트림 분석을 이용한 PLC 원격 제어로직 삽입 공격 탐지 기법** 이민규, 전승호, 서정택(가천대학교) **21. 칼만 필터를 사용한 센서 데이터 시공간 예측 방법 제시** 윤경주, 조재혁(전북대학교) **22. 블록체인 기반 의료정보 공유 기법 비교분석** 정경재(고려대학교), 이민섭(고려대학교), 천지영(서울사이버대학교) **23. 딥러닝을 이용한 CCTV 영상의 품질개선과 무인감시시스템** 고대식((주)토탈시스), 송석일(한국교통대학교) **24. 사각지대 제로화를 위한 이기종 다채널 영상처리용 가상 CCTV 시스템** 고대식 ((주)토탈시스)	**일반 부분4** 좌장 : 배영철(전남대)
17:40~18:00	**우수논문상 시상 및 경품 추첨**	김현 (부천대 교수)
사전등록하고 7월 27일 현장 참석자에 한하여 추첨을 통하여 경품을 드립니다.		
18:20~	**저녁 만찬 및 정보교류 [홍영재 장수청국장-학여울 삼성역 2번출구 사이]**	

*** 사전등록사이트: https://ictps.or.kr/46/write_form**

(발표 논문 1논문 당 1인 이상 필수 등록을 제외한 세미나 참석은 무료 등록입니다)

CUDA 프로그래밍의 코드 빌딩 블록에서 저전력 소비 코드 패턴 식별

진예진*, 김현태**, 김영철***

Code Patterns Identification of Low-Power Consumption on the Code building blocks of CUDA Programming

Ye-Jin Jin, Hyun-Tae Kim** and R. YoungChul Kim****

요 약

현재 국산 AI 반도체와 탑재할 소프트웨어 개발에 초점을 두고 진행 중에 있다. 그러나 기존에는 CPU 기반 절차식 프로그래밍 언어인 C/C++에서의 핵심 코드에서 저전력 코드 정의로 에너지 소비 절약 연구를 진행되고 있었다[1]. 앞으로는 방대한 데이터 처리를 위해, CPU 기반 또는 GPU 기반 CUDA 프로그래밍 언어의 핵심 프로그래밍 구조 내의 저전력 코드 식별이 필요하다. 이러한 문제를 해결하기 위해, CUDA 코드 내의 저전력 코드 패턴 식별 및 정의하고 기존 절차식 프로그래밍 언어와의 차이점을 분석한다. 이를 통해 데이터 관점에서 AI SW의 저전력 코드 패턴을 식별하여 전력 소비를 줄이고자 한다. 또한, 전력 소비 측정을 통해 저전력 코드 패턴 분석으로 증명하고자 한다. 특히 방대한 데이터 처리하는 AI SW에 식별된 코드 사용으로 에너지 소비 절약을 기대한다.

Key words

CUDA programming, C programming, GPU, CPU, Low Power Consumption

I. 서 론

최근 병렬 컴퓨팅에 대한 관심이 높아지고 있다. 그래픽 픽셀 계산만 하던 GPU를 CPU의 응용프로그램의 계산에 적용한 GPGPU (General Pupose GPU)는 빠른 연산 능력을 가지고 있다. AI 시대에 빅데이터를 다루기 위해 GPGPU의 기술은 발전하고 이에 따라 사용량도 증가하고 있다.

그러나, 고성능 컴퓨팅을 통해 계산 복잡도가 증가하면서 전력 소모량 또한 높아지고 있다[2]. 따라서 전력 소모를 줄이기 위해 효율적

* 홍익대학교 소프트웨어융합학과 일반대학원 석사과정 (yejin_jin@g.hongik.ac.kr)

** 홍익대학교 소프트웨어융합학과 일반대학원 석사과정 (hyuntaekim@g.hongik.ac.kr)

*** 홍익대학교 소프트웨어융합학과 교수 (bob@hongik.ac.kr)

인 코드 패턴 및 알고리즘 사용이 필요한 시점이다.

본 논문에서는 GPGPU의 종류 중 하나인 CUDA와 CPU를 사용하는 C의 전력 소비량을 비교한다. Code Building Blocks 중 반복문에 대해 분석하고, 그 결과를 바탕으로 전력의 측면의 효율적인 패턴을 정의한다.

Ⅱ. 기존 연구

2.1 전력 측정 도구

연구에서 CUDA와 C 코드의 CPU와 GPU 사용 전력을 각각 측정하여 합산하였다. CPU의 전력은 Intel Power Gadget, GPU의 전력은 Nvidia-smi을 사용하였다.

Intel Power Gadget은 Intel Core 프로세서에서 사용할 수 있는 소프트웨어 기반 전력 사용량 모니터링 도구이다[2]. 프로세서의 에너지 카운터를 사용하여 실시간 프로세서 패키지 전력 정보(Watt)를 모니터링할 수 있다. 해당 도구 사용 시, csv 형식으로 전력 사용량(Watt), 온도(℃), 주파수(GHz) 등의 정보를 확인할 수 있다. 본 연구에서는 전력 사용량(Power Draw)을 추출하여 사용하였다.

Nvidia-smi(NVIDIA System Management Interface)는 NVIDIA GPU 장치의 관리 및 모니터링을 돕기 위한 NVML (NVIDIA Management Library) 기반의 명령줄 유틸리티이다[3]. 해당 유틸리티를 통해 관리자는 GPU 장치 상태를 일반 텍스트 출력 및 파일로 확인할 수 있다. 본 연구에서는 10ms 단위로 장치의 상태를 확인하고, 텍스트 파일로 저장하여 필요한 부분만을 추출한다.

2.2 C언어 기존 연구

기존 반복문의 전력 비교 연구[4]에 따르면, 이중 do-while문의 전력 소비가 가장 적고, 인수 증가 이중 for문의 전력 소비가 가장 높다. 해당 연구를 통해 개발자는 전력 측면의 효율적인 코드 작성이 가능하다.

하지만 기존 연구에서는 반복 크기가 100만인 이중 반복문의 소비 전력만을 측정하였다. 이는 AI SW를 다루기엔 매우 적은 크기이다. 따라서 기존 크기를 늘려 1억 번 반복하도록 하였고, 단일 반복문의 소비 전력을 중점적으로 측정하였다.

Ⅲ. 본 연구

3.1 전력 측정 환경 및 프로세스

전력 측정을 위해 구축한 환경은 표 1과 같다.

표 1. 연구 구축 환경

IDE (Integrated Development Environment)	Visual Studio 2022
GPU	Nvidia RTX 5000
CPU	Intel i9-10900X
GPGPU	CUDA v12.1

전력 측정 시, 외부 변수를 차단하기 위해 전력 측정 조건을 설정하였다.

1. Background 실행 App: 모두 종료
2. Intel Power Gadget 실행

2-1. CPU의 전력: 10-12W 10초 이상 유지

3. Nvidia-smi를 실행

3-1. CPU의 전력: 20-22W 10초 이상 유지

코드 블록의 실행 시간을 이용하여 CPU와 GPU에서 측정된 전력량을 그래프로 나타내고, 넓이를 계산하여 소비 전력량을 측정하였다.

3.2 전력 측정 코드

전력을 측정하는 반복문은 인수 증가 for문, 인수 감소 for문, while문이다. 다음 그림 1은 본 연구에서 사용한 테스트 코드이다. main 함수 내에서 반복문이 포함된 커널 및 함수를 100번 호출하여 1부터 1억까지 곱한다. 해당 코드 앞뒤에 시간 기록 함수를 삽입하여 코드 블록의 실행 시간을 기록하였다.

CUDA는 C와 다르게 메모리 할당이 필요하

여 불가피하게 메모리 할당 함수가 추가로 사용되었다. C언어에서는 CUDA의 threadID와 같은 기능을 기본적으로 수행할 수 없기 때문에 반복문에 따로 변수를 선언하여 비슷한 기능을 수행할 수 있게 하였다.

	C	Cuda

Main	#define N 1000000
	
	int main() {
	  int arr[N] = { 0 };
	  for (int i = 0; i < N; i++) {
	  fn1(arr);
	  }
	  return 0;
	}
		#define N 10000000
		
		int main() {
		  int arr[N] = { 0 };
		  int* dev_arr;
		  int size = N * sizeof(int);
		  cudaMalloc((void**)&dev_arr, size);
		  cudaMemcpy(dev_arr, arr, size, cudaMemcpyHostToDevice);
		  int threads = 1024;
		  int blocks = (N + threads - 1) / threads;
		  fn1 << <blocks, threads >> > (dev_arr);
		  cudaMemcpy(arr, dev_arr, size, cudaMemcpyDeviceToHost);
		  return 0;
		}

	For ++	For --	While

C	int tid = 0;
	void fn1(int* arr) {
	  int x = tid;
	  for (int i = 1; i <= 100; i++) {
	  x *= x;
	  }
	  arr[tid] += x;
	  tid++;
	}
		int tid = 0;
		void fn1(int* arr) {
		  int x = tid;
		  for (int i = 100; i < 0; i--) {
		  x *= x;
		  }
		  arr[tid] += x;
		  tid++;
		}
			int tid = 0;
			void fn1(int* arr) {
			  int x = tid;
			  int i = 0;
			  while (i < 100) {
			  x *= x;
			  i++;
			  }
			  arr[tid] += x;
			  tid++;
			}

Cuda	__global__ void fn1(int* arr) {
	  int tid = blockIdx.x *
	blockDim.x + threadIdx.x;
	  int x = tid;
	  for (int i = 1; i <= 100; i++) {
	  x *= x;
	  }
	  arr[tid] += x;
	}
		__global__ void fn1(int* arr) {
		  int tid = blockIdx.x *
		blockDim.x + threadIdx.x;
		  int x = tid;
		  for (int i = 100; i < 0; i--) {
		  x *= x;
		  }
		  arr[tid] += x;
		}
			__global__ void fn1(int* arr) {
			  int tid = blockIdx.x *
			blockDim.x + threadIdx.x;
			  int x = tid;
			  int i = 0;
			  while (i < 100) {
			  x *= x;
			  i++;
			  }
			  arr[tid] += x;
			}

그림 1. C 언어와 CUDA 언어 내의 반복문 코드 비교

3.3 전력 측정 결과

전체적으로 CUDA에서의 소비 전력량이 C에서의 소비 전력량에 비해 적게 측정되었다. 이는 GPGPU가 연산에 빠르고 구조적인 특징을 가지고 있기 때문이다. C에서의 CPU 전력 측정 결과는 기존의 반복문 전력 연구 결과와 동일하다. 그러나 GPU를 포함하는 경우, 인수 증가 for문의 전력이 가장 낮고, while문의 전력이 가장 높다. CUDA에서의 반복문 전력 측정 결과는 인수 감소 for문이 0.0038Wh로 가장 낮고, 인수 증가 for문이 0.00445Wh로 가장 높다. 따라서 저전력 측면에서 가장 효율적인 패턴은 for문의 인수 감소 패턴이라고 정의할 수 있다. 그러나 GPU의 소비 전력만을 확인하였을 때, 대체로 비슷하다. 데이터 처리량을 늘려 더욱 명확한 차이를 확인해보아야 한다. 표2는 전력 측정 결과를 나타낸 표이다.

표 2. 전력 측정 결과

[단위: Wh]		For++	For--	While
C	CPU	0.033513	0.029166	0.02936
	GPU	0.03938	0.04445	0.06504
	Total	0.072893	0.073616	0.0944
CUDA	CPU	0.00320	0.00261	0.00275
	GPU	0.00125	0.00118	0.00117
	Total	0.00445	0.00380	0.00391

IV. 결론 및 향후 연구

본 연구에서는 GPGPU의 전력 과소비에 대한 문제점을 극복하고자, Code Building Blocks 중 반복문에 대한 저전력 코드의 패턴을 분석하였다. 연구 결과, 인수 증가 for문의 소비 전력이 가장 크고 인수 감소 for문의 소비 전력이 가장 작은 것으로 나타났다. 따라서 인수 감소 for문을 저전력 코드 패턴으로 정의한다.

본 연구에서 측정한 GPU의 전력 차이가 크지 않다. 1억으로 정한 데이터 처리량을 추후 연구에서 더욱 늘린다면, 더욱 명확한 결과가 나타날 것으로 기대한다. 또한, 다양한 Code Building Block의 Bad Energy Pattern을 정의한다면 AI SW에서의 전력 문제를 해결할 수 있을 것이다.

ACKNOWLEDGMENT

본 연구는 2023년도 문화체육관광부의 재원으로 한국콘텐츠진흥원(과제명: 인공지능 기반 사용자 대화형 멀티모달 인터랙티브 스토리텔링 3D장면 저작 기술 개발, 과제번호: RS-2023-00227917, 기여율: 50%) 지원과 2023년도 정부(교육부)의 재원으로 한국연구재단 기초연구사업(과제명: NLP BERT Model 기반 자동 리팩토링을 통한 무결점 코드화 연

구, 과제번호: No.2021R1I1A3050407, 기여율: 50%)의 지원을 받아 수행된 연구임.

참 고 문 헌

[1] 박경문, 김병서, 김영철, (2018), 초소형 IoT 디바이스의 저전력 최적운용 기술 분석 및 알고리즘 요소기술 연구, 한국전자통신연구원 보고서

[2] Huang, Yanhui, Bing Guo, and Yan Shen. GPU energy consumption optimization with a global-based neural network method, IEEE Access 7, 2019.

[3] Intel, https://www.intel.com/content/www/us/en/developer/articles/tool/power-gadget.html

[4] Nvidia, https://developer.nvidia.com/nvidia-system-management-interface

[5] 안현식, 고급 프로그래밍 코드 내 전력 소비 측정 통한 저전력 코드 패턴 메카니즘 식별 가이드, ICT 플랫폼 학회, Vol. 7-1, 2019, pages 15-18.