(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2020년12월23일
(11) 등록번호 10-2194513
(24) 등록일자 2020년12월17일

(51) 국제특허분류(Int. Cl.)
G06F 9/48 (2018.01)
(52) CPC특허분류
G06F 9/4881 (2013.01)
G06F 9/3836 (2013.01)
(21) 출원번호 10-2019-0073406
(22) 출원일자 2019년06월20일
심사청구일자 2019년06월20일
(56) 선행기술조사문헌
KR101885211 B1
(뒷면에 계속)

(73) 특허권자
배재대학교 산학협력단
대전광역시 서구 배재로 155-40 (도마동)
(72) 발명자
정희경
대전광역시 서구 둔산로 155, 112동 1303호(둔산동, 크로바아파트)
김경환
대전광역시 유성구 엑스포로 448, 502동 701호 (전민동, 엑스포아파트)
(74) 대리인
유병욱, 한승범

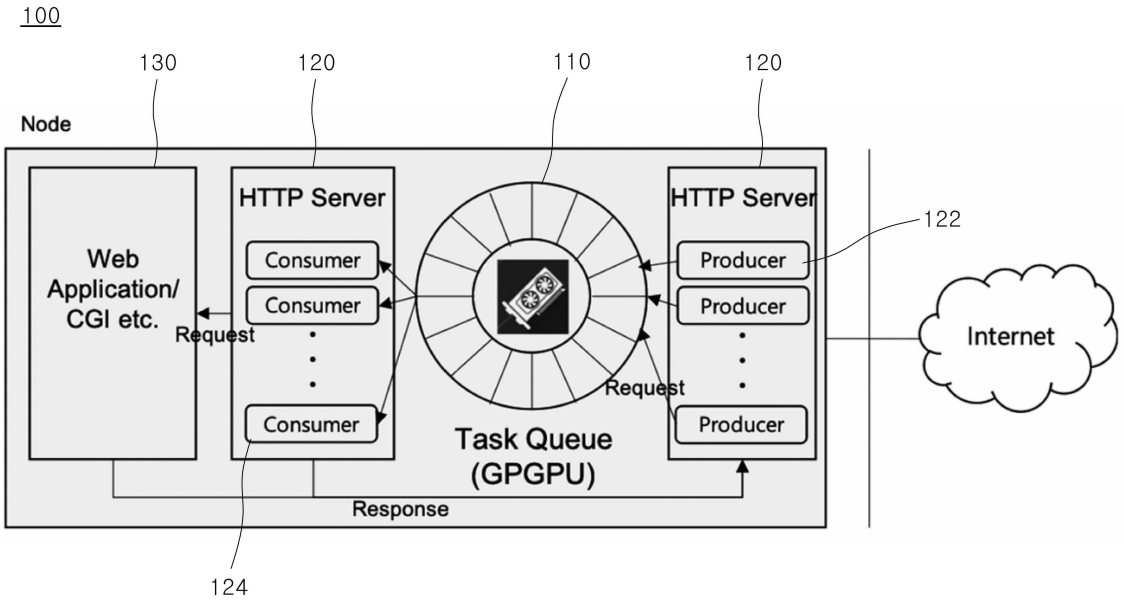
전체 청구항 수 : 총 11 항 심사관 : 유진태

(54) 발명의 명칭 **GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템 및 방법**

(57) 요 약

본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템은 GPU에 생성되고 GPGPU의 다중 스레드 처리 능력을 활용하여 웹 서비스의 사용자 요청을 처리하는 태스크 큐; 및 클라이언트로부터 상기 사용자 요청에 대응하는 HTTP 요청(Request)이 있으면 이에 응답하여 복수의 프로듀서 스레드(Producer Thread) 및 복수의 컨슈머 스레드(Consumer Thread)를 생성하고, 상기 복수의 프로듀서 스레드를 통해 상기 HTTP 요청에 관한 패킷을 받아 상기 태스크 큐에 전송하며, 상기 태스크 큐에 의해 전처리 된 데이터에 대해 상기 복수의 컨슈머 스레드를 통해 오류 발생 여부를 확인하고 확인 결과 오류가 없으면 상기 데이터를 웹 어플리케이션 서버에 전달하는 HTTP 서버를 포함한다.

대 표 도



(52) CPC특허분류
G06F 9/3855 (2013.01)

(56) 선행기술조사문헌
KR101832167 B1
KR1020150084098 A
KR1020170116439 A
KR101827540 B1
KR1020100057831 A

명 세 서

청구범위

청구항 1

GPU에 생성되고 GPGPU의 다중 스레드 처리 능력을 활용하여 웹 서비스의 사용자 요청을 처리하는 태스크 큐; 및

클라이언트로부터 상기 사용자 요청에 대응하는 HTTP 요청(Request)이 있으면 이에 응답하여 복수의 프로듀서 스레드(Producer Thread) 및 복수의 컨슈머 스레드(Consumer Thread)를 생성하고, 상기 복수의 프로듀서 스레드를 통해 상기 HTTP 요청에 관한 패킷을 받아 상기 태스크 큐에 전송하며, 상기 태스크 큐가 원형 큐 알고리즘을 이용하여 대량으로 발생하는 상기 HTTP 요청에 관한 패킷의 데이터가 상기 태스크 큐의 용량을 초과하지 않도록 전처리 하면, 상기 태스크 큐에 의해 전처리 된 데이터에 대해 상기 복수의 컨슈머 스레드를 통해 상기 전처리 된 데이터의 오류 발생 여부를 확인하고 확인 결과 오류가 없으면 상기 데이터를 웹 어플리케이션 서버에 전달하는 HTTP 서버

를 포함하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 2

삭제

청구항 3

제1항에 있어서,

상기 태스크 큐는

GPGPU 기술 기반의 CUDA(Compute Unified Device Architecture)를 사용하여 구현되는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 4

제3항에 있어서,

상기 CUDA는

커널 함수를 이용하여 HTTP 패킷 오류 처리 함수를 호출하여 상기 GPU에 복수의 스레드와 함께 생성하고, 상기 HTTP 패킷 오류 처리 함수는 상기 커널 함수에서 정의된 태스크 큐 데이터의 구조체와 그 구조체의 크기를 파라미터로 받아서 상기 HTTP 요청에 관한 패킷의 적합성을 확인하여 에러 플래그(Error Flag)에 값을 넣어 반환 처리하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 5

제1항에 있어서,

상기 컨슈머 스레드는

상기 태스크 큐에 의해 전처리 된 데이터를 비동기 방식으로 읽어 들여서 오류 발생 여부를 확인하고, 확인 결과 오류가 발생하면 상기 프로듀서 스레드에게 오류 코드에 대한 결과를 전송하여 상기 프로듀서 스레드를 통해 사용자에게 바로 그 결과를 전송하고, 확인 결과 오류가 없으면 상기 웹 어플리케이션 서버에 상기 데이터를 전달하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 6

제1항에 있어서,

상기 컨슈머 스레드는

HTTP 프로토콜을 처리할 수 있는 기능을 통해 상기 프로듀서 스레드에서 생성한 태스크 큐를 접근하여 비동기 방식으로 상기 전처리 된 데이터를 읽어 들이되, 상기 읽어 들인 데이터는 이미 상기 GPU에서 데이터의 정합성, HTTP의 오류, 사용자 입력 값의 오류 중 적어도 하나를 확인하여 구조체에 저장하고 있으므로 상기 구조체의 값을 참고하여 상기 데이터의 오류 발생 여부를 확인하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 7

제6항에 있어서,

상기 프로듀서 스레드에서 생성한 태스크 큐의 스레드는

사용자 요청의 데이터를 검사하여 오류가 발생하면 부울 형 구조체 변수에 오류 발생 여부와 오류 코드를 저장하고,

상기 컨슈머 스레드는

상기 부울 형 구조체 변수의 값을 참고하여 상기 전처리 된 데이터의 오류 발생 여부를 확인하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 8

제1항에 있어서,

상기 태스크 큐는

CPU의 부하로 인해 상기 사용자 요청을 처리하지 못할 경우 클라이언트에게 대기 시간 및 대기 순번을 포함하는 대기 정보를 전송하여 사용자에게 안내하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템.

청구항 9

클라이언트로부터 웹 서비스의 사용자 요청에 대응하는 HTTP 요청(Request)이 있으면 이에 응답하여 HTTP 서버가 복수의 프로듀서 스레드(Producer Thread) 및 복수의 컨슈머 스레드(Consumer Thread)를 생성하는 단계;

상기 HTTP 서버가 상기 복수의 프로듀서 스레드를 통해 상기 HTTP 요청에 관한 패킷을 받아 GPU에 생성된 태스크 큐에 전송하는 단계;

상기 태스크 큐가 GPGPU의 다중 스레드 처리 능력을 활용하여 상기 HTTP 요청을 처리하는 단계;

상기 태스크 큐가 원형 큐 알고리즘을 이용하여 대량으로 발생하는 상기 HTTP 요청에 관한 패킷의 데이터가 상기 태스크 큐의 용량을 초과하지 않도록 전처리 하면, 상기 HTTP 서버가 상기 태스크 큐에 의해 전처리 된 데이터에 대해 상기 복수의 컨슈머 스레드를 통해 상기 전처리 된 데이터의 오류 발생 여부를 확인하는 단계; 및

확인 결과 오류가 없으면, 상기 HTTP 서버가 상기 전처리 된 데이터를 웹 어플리케이션 서버에 전달하는 단계

를 포함하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법.

청구항 10

삭제

청구항 11

제9항에 있어서,

상기 태스크 큐는

GPGPU 기술 기반의 CUDA를 사용하여 구현되는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법.

청구항 12

제11항에 있어서,

상기 CUDA는

커널 함수를 이용하여 HTTP 패킷 오류 처리 함수를 호출하여 상기 GPU에 복수의 스레드와 함께 생성하고, 상기 HTTP 패킷 오류 처리 함수는 상기 커널 함수에서 정의된 태스크 큐 데이터의 구조체와 그 구조체의 크기를 파라미터로 받아서 상기 HTTP 요청에 관한 패킷의 적합성을 확인하여 에러 플래그(Error Flag)에 값을 넣어 반환 처리하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법.

청구항 13

제9항에 있어서,

상기 컨슈머 스레드는

HTTP 프로토콜을 처리할 수 있는 기능을 통해 상기 프로듀서 스레드에서 생성한 태스크 큐를 접근하여 비동기 방식으로 상기 전처리 된 데이터를 읽어 들이되, 상기 읽어 들인 데이터는 이미 상기 GPU에서 데이터의 정합성, HTTP의 오류, 사용자 입력 값의 오류 중 적어도 하나를 확인하여 구조체에 저장하고 있으므로 상기 구조체의 값을 참고하여 상기 전처리 된 데이터의 오류 발생 여부를 확인하는 것을 특징으로 하는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법.

발명의 설명

기 술 분 야

[0001] 본 발명의 실시예들은 GPGPU(General-Purpose Computing on Graphics Processing) 기반 태스크 큐를 활용한 웹 서비스 제공 시스템 및 방법에 관한 것이다.

배 경 기 술

[0003] 인터넷에서의 웹 서버는 불특정 다수의 사용자 요청을 처리하기 위하여 항상 준비되어 있고 매우 개방적이다. 하지만 대량의 사용자 요청이 발생할 경우 장애가 발생하거나 원활한 웹 서비스가 불가능한 경우가 발생한다.

[0004] 평상시에는 사용자 요청량을 처리하는데 문제가 없던 웹 서버도 갑자기 발생하는 대량의 사용자 요청을 처리하지 못하고 웹 서버의 다운이 발생하기도 한다. 웹 서버의 다운이 발생하면 시스템을 다시 부팅하거나 또는 웹 서버의 재가동에 상당한 시간을 투자하여 정상화 시킬 수 있다.

[0005] 이러한 문제를 대비하기 위하여 웹 서버의 자원을 늘리고 동적으로 자원을 할당할 수 있는 시스템인 클라우드 서비스 등을 활용하고 문제 해결을 위해 노력하고 있다. 이런 노력들은 대부분 많은 비용을 발생시키고, 어떤 경우에는 완전한 해결 방법을 제시하지 못하고 있다.

[0006] 한편, GPGPU는 HPC 분야, 게임, 시뮬레이션, 인공지능 분야 등에 많이 이용되고 그 수가 점점 증가하고 있다. GPU를 사용하여 많은 작업들을 수행하고 GPU의 이용 분야도 다양하게 발전하고 있다.

[0007] 최신의 GPU는 그래픽 처리뿐 아니라 GPGPU 프로그램을 지원하는 고성능의 부동 소수점 연산 기능을 제공하고 있

다. 또한 대량의 스레드를 생성하여 처리할 수 있도록 그 성능도 계속 발전하고 있다. 이런 GPU의 성능을 일반 어플리케이션에서 사용할 수 있도록 GPU가 계속 발전하고, GPU가 CPU를 보조하여 발전하면 범용의 어플리케이션의 실행 성능이 향상되고, 고성능의 범용 어플리케이션을 수행하기 위한 하드웨어도 더 비용을 절감할 수 있다.

[0008] 관련 선행기술로는 대한민국 공개특허공보 제10-2017-0116439호(발명의 명칭: 태스크 스케줄링 방법 및 장치, 공개일자: 2017.10.19)가 있다.

발명의 내용

해결하려는 과제

[0010] 본 발명의 일 실시예는 GPGPU를 활용하여 대량의 요구사항을 처리할 수 있는 태스크 큐를 설계하여 HTTP의 일부를 GPU에서 처리하여 CPU의 작업을 줄여 성능을 향상시킬 수 있는 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템 및 방법을 제공한다.

[0012] 본 발명이 해결하고자 하는 과제는 이상에서 언급한 과제(들)로 제한되지 않으며, 언급되지 않은 또 다른 과제(들)은 아래의 기재로부터 당업자에게 명확하게 이해될 수 있을 것이다.

과제의 해결 수단

[0014] 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템은 GPU에 생성되고 GPGPU의 다중 스레드 처리 능력을 활용하여 웹 서비스의 사용자 요청을 처리하는 태스크 큐; 및 클라이언트로부터 상기 사용자 요청에 대응하는 HTTP 요청(Request)이 있으면 이에 응답하여 복수의 프로듀서 스레드(Producer Thread) 및 복수의 컨슈머 스레드(Consumer Thread)를 생성하고, 상기 복수의 프로듀서 스레드를 통해 상기 HTTP 요청에 관한 패킷을 받아 상기 태스크 큐에 전송하며, 상기 태스크 큐에 의해 전처리 된 데이터에 대해 상기 복수의 컨슈머 스레드를 통해 오류 발생 여부를 확인하고 확인 결과 오류가 없으면 상기 데이터를 웹 어플리케이션 서버에 전달하는 HTTP 서버를 포함한다.

[0015] 상기 태스크 큐는 원형 큐 알고리즘을 이용하여 대량으로 발생하는 상기 사용자 요청이 상기 태스크 큐의 용량을 초과하지 않도록 처리할 수 있다.

[0016] 상기 태스크 큐는 GPGPU 기술 기반의 CUDA(Compute Unified Device Architecture)를 사용하여 구현되되, 상기 CUDA의 스레드가 원형 큐와 같이 작동하여 어플리케이션 처리가 가능하도록 구현될 수 있다.

[0017] 상기 CUDA는 커널 함수를 이용하여 HTTP 패킷 오류 처리 함수를 호출하여 상기 GPU에 복수의 스레드와 함께 생성하고, 상기 HTTP 패킷 오류 처리 함수는 상기 커널 함수에서 정의된 태스크 큐 데이터의 구조체와 그 구조체의 크기를 파라미터로 받아서 상기 HTTP 요청에 관한 패킷의 적합성을 확인하여 에러 플래그(Error Flag)에 값을 넣어 반환 처리할 수 있다.

[0018] 상기 컨슈머 스레드는 상기 태스크 큐의 데이터를 비동기 방식으로 읽어 들여서 오류 발생 여부를 확인하고, 확인 결과 오류가 발생하면 상기 프로듀서 스레드에게 오류 코드에 대한 결과를 전송하여 상기 프로듀서 스레드를 통해 사용자에게 바로 그 결과를 전송하고, 확인 결과 오류가 없으면 상기 웹 어플리케이션 서버에 상기 데이터를 전달할 수 있다.

[0019] 상기 컨슈머 스레드는 HTTP 프로토콜을 처리할 수 있는 기능을 통해 상기 프로듀서 스레드에서 생성한 태스크 큐를 접근하여 비동기 방식으로 상기 데이터를 읽어 들이되, 상기 읽어 들인 데이터는 이미 상기 GPU에서 데이터의 정합성, HTTP의 오류, 사용자 입력 값의 오류 중 적어도 하나를 확인하여 구조체에 저장하고 있으므로 상기 구조체의 값을 참고하여 상기 데이터의 오류 발생 여부를 확인할 수 있다.

[0020] 상기 프로듀서 스레드에서 생성한 태스크 큐의 스레드는 사용자 요청의 데이터를 검사하여 오류가 발생하면 부울 형 구조체 변수에 오류 발생 여부와 오류 코드를 저장하고, 상기 컨슈머 스레드는 상기 부울 형 구조체 변수의 값을 참고하여 상기 데이터의 오류 발생 여부를 확인할 수 있다.

[0021] 상기 태스크 큐는 CPU의 부하로 인해 상기 사용자 요청을 처리하지 못할 경우 클라이언트에게 대기 시간 및 대기 순번을 포함하는 대기 정보를 전송하여 사용자에게 안내할 수 있다.

[0022] GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법은 클라이언트로부터 웹 서비스의 사용자 요청에 대응하는

HTTP 요청(Request)이 있으면 이에 응답하여 HTTP 서버가 복수의 프로듀서 스레드(Producer Thread) 및 복수의 컨슈머 스레드(Consumer Thread)를 생성하는 단계; 상기 HTTP 서버가 상기 복수의 프로듀서 스레드를 통해 상기 HTTP 요청에 관한 패킷을 받아 GPU에 생성된 태스크 큐에 전송하는 단계; 상기 태스크 큐가 GPGPU의 다중 스레드 처리 능력을 활용하여 상기 HTTP 요청을 처리하는 단계; 상기 HTTP 서버가 상기 태스크 큐에 의해 전처리 된 데이터에 대해 상기 복수의 컨슈머 스레드를 통해 오류 발생 여부를 확인하는 단계; 및 확인 결과 오류가 없으면, 상기 HTTP 서버가 상기 데이터를 웹 어플리케이션 서버에 전달하는 단계를 포함한다.

[0023] 상기 태스크 큐는 원형 큐 알고리즘을 이용하여 대량으로 발생하는 상기 사용자 요청이 상기 태스크 큐의 용량을 초과하지 않도록 처리할 수 있다.

[0024] 상기 태스크 큐는 GPGPU 기술 기반의 CUDA를 사용하여 구현되되, 상기 CUDA의 스레드가 원형 큐와 같이 작동하여 어플리케이션 처리가 가능하도록 구현될 수 있다.

[0025] 상기 CUDA는 커널 함수를 이용하여 HTTP 패킷 오류 처리 함수를 호출하여 상기 GPU에 복수의 스레드와 함께 생성하고, 상기 HTTP 패킷 오류 처리 함수는 상기 커널 함수에서 정의된 태스크 큐 데이터의 구조체와 그 구조체의 크기를 파라미터로 받아서 상기 HTTP 요청에 관한 패킷의 적합성을 확인하여 에러 플래그(Error Flag)에 값을 넣어 반환 처리할 수 있다.

[0026] 상기 컨슈머 스레드는 HTTP 프로토콜을 처리할 수 있는 기능을 통해 상기 프로듀서 스레드에서 생성한 태스크 큐를 접근하여 비동기 방식으로 상기 데이터를 읽어 들이되, 상기 읽어 들인 데이터는 이미 상기 GPU에서 데이터의 정합성, HTTP의 오류, 사용자 입력 값의 오류 중 적어도 하나를 확인하여 구조체에 저장하고 있으므로 상기 구조체의 값을 참고하여 상기 데이터의 오류 발생 여부를 확인할 수 있다.

[0028] 기타 실시예들의 구체적인 사항들은 상세한 설명 및 첨부 도면들에 포함되어 있다.

발명의 효과

[0030] 본 발명의 일 실시예에 따르면, GPGPU를 활용하여 대량의 요구사항을 처리할 수 있는 태스크 큐를 설계하여 HTTP의 일부를 GPU에서 처리하여 CPU의 작업을 줄여 성능을 향상시킬 수 있다.

[0031] 본 발명의 일 실시예에 따르면, CPU의 부하가 많이 걸려 사용자 요구를 처리하지 못할 경우 태스크 큐에서 사용자에게 대기 시간 및 대기 순번을 안내할 수 있어서 사용자가 무한정 기다리는 것을 방지할 수 있다.

[0032] 본 발명의 일 실시예에 따르면, 웹 서비스의 사용자 요청을 처리하기 위해 GPGPU의 다중 스레드 처리 능력을 활용할 수 있는 태스크 큐를 개발하여 HTTP 요청을 태스크 큐에 저장하므로 GPU에서 HTTP를 전처리 하여 웹 서버의 부담을 경감할 수 있다.

[0033] 본 발명의 일 실시예에 따르면, 태스크 큐를 원형 큐 알고리즘을 적용하여 대량으로 발생하는 사용자의 요청이 태스크 큐 용량을 초과할 수 없도록 함으로써 웹 사용자의 요청을 웹 서버에서 처리 가능한 요청으로 받아들여서 웹 서버가 보다 안정적으로 웹 서비스를 처리하도록 할 수 있다.

도면의 간단한 설명

[0035] 도 1은 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템을 설명하기 위해 도시한 블록도이다.

도 2는 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법을 설명하기 위해 도시한 흐름도이다.

도 3은 GPGPU 기반 태스크 큐의 CUDA 커널 함수에서 호출하여 사용되는 HTTP 패킷 오류처리 함수의 일례를 나타낸 도면이다.

도 4는 HTTP 서버의 Producer를 구현한 일례를 도시한 도면이다.

도 5는 HTTP 서버의 Consumer를 구현한 일례를 도시한 도면이다.

도 6은 각각의 실험 시나리오별 성능을 나타낸 도면이다.

도 7은 두 서버(GPGPU 기반 태스크 큐를 사용하지 않은 서버와 GPGPU 기반 태스크 큐 서버)의 오류율을 나타낸 도면이다.

발명을 실시하기 위한 구체적인 내용

[0036] 본 발명의 이점 및/또는 특징, 그리고 그것들을 달성하는 방법은 첨부되는 도면과 함께 상세하게 후술되어 있는 실시예들을 참조하면 명확해질 것이다. 그러나, 본 발명은 이하에서 개시되는 실시예들에 한정되는 것이 아니라 서로 다른 다양한 형태로 구현될 것이며, 단지 본 실시예들은 본 발명의 개시가 완전하도록 하며, 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자에게 발명의 범주를 완전하게 알려주기 위해 제공되는 것이며, 본 발명은 청구항의 범주에 의해 정의될 뿐이다. 명세서 전체에 걸쳐 동일 참조 부호는 동일 구성요소를 지칭한다.

[0037] 또한, 이하 실시되는 본 발명의 바람직한 실시예는 본 발명을 이루는 기술적 구성요소를 효율적으로 설명하기 위해 각각의 시스템 기능구성에 기 구비되어 있거나, 또는 본 발명이 속하는 기술분야에서 통상적으로 구비되는 시스템 기능 구성은 가능한 생략하고, 본 발명을 위해 추가적으로 구비되어야 하는 기능 구성을 위주로 설명한다. 만약 본 발명이 속하는 기술분야에서 통상의 지식을 가진 자라면, 하기에 도시하지 않고 생략된 기능 구성 중에서 종래에 기 사용되고 있는 구성요소의 기능을 용이하게 이해할 수 있을 것이며, 또한 상기와 같이 생략된 구성 요소와 본 발명을 위해 추가된 구성 요소 사이의 관계도 명백하게 이해할 수 있을 것이다.

[0038] 또한, 이하의 설명에 있어서, 신호 또는 정보의 "전송", "통신", "송신", "수신" 기타 이와 유사한 의미의 용어는 일 구성요소에서 다른 구성요소로 신호 또는 정보가 직접 전달되는 것뿐만이 아니라 다른 구성요소를 거쳐 전달되는 것도 포함한다. 특히 신호 또는 정보를 일 구성요소로 "전송" 또는 "송신"한다는 것은 그 신호 또는 정보의 최종 목적지를 지시하는 것이고 직접적인 목적지를 의미하는 것이 아니다. 이는 신호 또는 정보의 "수신"에 있어서도 동일하다.

[0040] 이하에서는 첨부된 도면을 참조하여 본 발명의 실시예들을 상세히 설명하기로 한다.

[0041] 도 1은 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템을 설명하기 위해 도시한 블록도이다.

[0042] 도 1을 참조하면, 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템(100)은 태스크 큐(110), HTTP 서버(120) 및 웹 어플리케이션 서버(130)를 포함하여 구성될 수 있다.

[0043] 상기 HTTP 서버(120)의 컨슈머(Consumer) 스레드(124)에서 GPGPU 프로그래밍 방법으로 GPU에 태스크 큐(110)를 생성하고, 상기 GPU에서 실행할 C 코드도 함께 GPU에 전송하여 대량의 스레드를 C 코드 실행으로 생성할 수 있다. 상기 HTTP 서버(120)의 컨슈머 스레드(124)에서 스레드 풀을 생성하고 관리하며, 상기 HTTP 서버(120)의 프로듀서(Producer) 스레드(122)에서도 태스크 큐(110)에 접근이 가능하도록 GPGPU 프로그래밍 방법으로 상기 GPU를 사용한다. 상기 프로듀서 스레드(122)에서 사용자 요청을 받으면 바로 태스크 큐(110)에 요청을 전송할 수 있다.

[0044] 이하에서는 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 시스템(100)의 구성요소들(110, 120, 122, 124, 130) 각각에 대해서 구체적으로 설명한다.

[0045] 상기 태스크 큐(110)는 GPU에 생성되고, GPGPU의 다중 스레드 처리 능력을 활용하여 웹 서비스의 사용자 요청을 처리할 수 있다. 이때, 상기 태스크 큐(110)는 원형 큐 알고리즘을 이용하여 대량으로 발생하는 상기 사용자 요청이 상기 태스크 큐의 용량을 초과하지 않도록 처리할 수 있다.

[0046] 상기 태스크 큐(110)는 GPGPU 기술 기반의 CUDA(Compute Unified Device Architecture)를 사용하여 구현될 수 있다. 여기서, 상기 CUDA의 스레드(Thread)는 원형 큐와 같이 작동하여 어플리케이션 처리가 가능하도록 구현될 수 있다.

[0047] 또한, 상기 CUDA는 커널 함수를 이용하여 HTTP 패킷 오류 처리 함수를 호출하여 생성할 수 있는데, 이때 상기 HTTP 패킷 오류 처리 함수는 복수의 스레드와 함께 상기 GPU에 생성될 수 있다.

[0048] 상기 HTTP 패킷 오류 처리 함수는 상기 커널 함수에서 정의된 태스크 큐 데이터의 구조체와 그 구조체의 크기를 파라미터로 받아서 상기 웹 서비스의 사용자 요청, 즉 HTTP 요청에 관한 패킷의 적합성을 확인하고, 그 확인 결과에 따라 에러 플래그(Error Flag)에 값을 넣어 반환 처리할 수 있다.

[0049] 한편, 상기 태스크 큐(110)는 CPU의 부하로 인해 상기 사용자 요청(HTTP 요청)을 처리하지 못할 경우, 클라이언트에게 대기 시간 및 대기 순번을 포함하는 대기 정보를 전송할 수 있다. 이에 따라 사용자는 상기 사용자 요청에 대한 대기 정보를 상기 클라이언트(예: PC, 스마트폰 등)의 화면을 통하여 안내를 받을 수 있다.

[0050] 상기 HTTP 서버(120)는 클라이언트로부터 상기 사용자 요청에 대응하는 HTTP 요청(Request)이 있으면 이에 응답하여 복수의 프로듀서 스레드(Producer Thread)(122) 및 복수의 컨슈머 스레드(Consumer Thread)(124)를 생성할 수 있다.

[0051] 상기 HTTP 서버(120)는 상기 복수의 프로듀서 스레드(122)를 통해 상기 HTTP 요청에 관한 패킷을 받아 상기 태스크 큐(110)에 전송할 수 있다. 그리고, 상기 HTTP 서버(120)는 상기 태스크 큐(110)에 의해 전처리 된 데이터에 대해 상기 복수의 컨슈머 스레드(124)를 통해 오류 발생 여부를 확인할 수 있다.

[0052] 여기서, 상기 컨슈머 스레드(124)는 상기 태스크 큐(110)의 데이터를 비동기 방식으로 읽어 들여서 오류 발생 여부를 확인할 수 있다.

[0053] 구체적으로, 상기 컨슈머 스레드(124)는 HTTP 프로토콜을 처리할 수 있는 기능을 통해 상기 프로듀서 스레드(122)에서 생성한 태스크 큐(110)를 접근하여 비동기 방식으로 상기 데이터를 읽어 들일 수 있다. 그런데, 상기 읽어 들인 데이터는 이미 상기 GPU에서 데이터의 정합성, HTTP의 오류, 사용자 입력 값의 오류 등을 확인하여 구조체에 저장하고 있다. 그러므로, 상기 컨슈머 스레드(124)는 상기 구조체의 값을 참고하여 상기 데이터의 오류 발생 여부를 확인할 수 있다.

[0054] 다시 말해, 상기 프로듀서 스레드(122)에서 생성한 태스크 큐(110)의 스레드는 상기 사용자 요청의 데이터를 검사하여 오류가 발생하면 부울 형 구조체 변수에 오류 발생 여부와 오류 코드를 저장할 수 있다. 따라서, 상기 컨슈머 스레드(124)는 상기 부울 형 구조체 변수의 값을 참고하여 상기 데이터의 오류 발생 여부를 확인할 수 있다.

[0055] 상기 컨슈머 스레드(124)는 오류 발생 여부의 확인 결과, 오류가 발생하면 상기 프로듀서 스레드(122)에게 오류 코드에 대한 결과를 전송하여 상기 프로듀서 스레드(122)를 통해 사용자에게 바로 그 결과를 전송할 수 있다.

[0056] 반면에, 오류 발생 여부의 확인 결과 오류가 없으면, 상기 컨슈머 스레드(124)는 상기 웹 어플리케이션 서버(130)에 상기 데이터를 전달할 수 있다.

[0058] 이상에서 설명된 장치는 하드웨어 구성 요소, 소프트웨어 구성 요소, 및/또는 하드웨어 구성 요소 및 소프트웨어 구성 요소의 조합으로 구현될 수 있다. 예를 들어, 실시예들에서 설명된 장치 및 구성 요소는, 예를 들어, 프로세서, 컨트롤러, ALU(arithmetic logic unit), 디지털 신호 프로세서(digital signal processor), 마이크로컴퓨터, FPA(field programmable array), PLU(programmable logic unit), 마이크로프로세서, 또는 명령(instruction)을 실행하고 응답할 수 있는 다른 어떠한 장치와 같이, 하나 이상의 범용 컴퓨터 또는 특수 목적 컴퓨터를 이용하여 구현될 수 있다. 처리 장치는 운영 체제(OS) 및 상기 운영 체제 상에서 수행되는 하나 이상의 소프트웨어 애플리케이션을 수행할 수 있다. 또한, 처리 장치는 소프트웨어의 실행에 응답하여, 데이터를 접근, 저장, 조작, 처리 및 생성할 수도 있다. 이해의 편의를 위하여, 처리 장치는 하나가 사용되는 것으로 설명된 경우도 있지만, 해당 기술분야에서 통상의 지식을 가진 자는, 처리 장치가 복수 개의 처리 요소(processing element) 및/또는 복수 유형의 처리 요소를 포함할 수 있음을 알 수 있다. 예를 들어, 처리 장치는 복수 개의 프로세서 또는 하나의 프로세서 및 하나의 컨트롤러를 포함할 수 있다. 또한, 병렬 프로세서(parallel processor)와 같은, 다른 처리 구성(processing configuration)도 가능하다.

[0059] 소프트웨어는 컴퓨터 프로그램(computer program), 코드(code), 명령(instruction), 또는 이들 중 하나 이상의 조합을 포함할 수 있으며, 원하는 대로 동작하도록 처리 장치를 구성하거나 독립적으로 또는 결합적으로(collectively) 처리 장치를 명령할 수 있다. 소프트웨어 및/또는 데이터는, 처리 장치에 의하여 해석되거나 처리 장치에 명령 또는 데이터를 제공하기 위하여, 어떤 유형의 기계, 구성요소(component), 물리적 장치, 가상 장치(virtual equipment), 컴퓨터 저장 매체 또는 장치, 또는 전송되는 신호 파(signal wave)에 영구적으로, 또는 일시적으로 구체화(embody)될 수 있다. 소프트웨어는 네트워크로 연결된 컴퓨터 시스템 상에 분산되어서, 분산된 방법으로 저장되거나 실행될 수도 있다. 소프트웨어 및 데이터는 하나 이상의 컴퓨터 판독 가능 기록 매체에 저장될 수 있다.

[0061] 도 2는 본 발명의 일 실시예에 따른 GPGPU 기반 태스크 큐를 활용한 웹 서비스 제공 방법을 설명하기 위해 도시한 흐름도이다.

[0062] 여기서 설명하는 웹 서비스 제공 방법은 본 발명의 하나의 실시예에 불과하며, 그 이외에 필요에 따라 다양한 단계들이 부가될 수 있고, 하기의 단계들도 순서를 변경하여 실시될 수 있으므로, 본 발명이 하기에 설명하는 각 단계 및 그 순서에 한정되는 것은 아니다.

[0063] 도 2를 참조하면, 먼저 클라이언트가 웹 서버에 사용자의 요청을 보내면(1. Http Request), HTTP 서버(120)의 프로듀서 스레드(122)는 사용자의 HTTP Request를 받아 바로 GPU의 태스크 큐(110)에 패킷을 전송한다(2. Http Packet sending Task Queue).

[0064] 이에 따라, GPU의 스레드는 패킷 데이터를 저장하고 HTTP의 패킷 오류, 사용자 데이터 오류 등을 확인하여 오류의 내용을 변수에 저장하고 대기한다.

[0065] 다음으로, 상기 HTTP 서버(120)의 컨슈머 스레드(124)는 태스크 큐(110)의 데이터를 비동기 방식으로 읽어서 GPU 스레드에서 처리한 오류 발생 여부를 확인한다(3. Http Packet Reading Task Queue).

[0066] 이때, 확인 결과 상기 데이터에 오류가 발생하면, 상기 컨슈머 스레드(124)는 상기 프로듀서 스레드(122)에게 오류 코드에 대한 결과를 전송하고, 상기 프로듀서 스레드(122)는 사용자에게 바로 결과를 전송한다(4. Error Packet Response).

[0067] 반면, 확인 결과 상기 데이터에 오류 발생이 없으면, 상기 컨슈머 스레드(124)는 웹 어플리케이션 서버(130)에 상기 데이터를 전달한다(5. Http Packet Forwarding).

[0068] 여기서, 상기 프로듀서 스레드(122)는 TCP/IP 소켓 서버 프로그래밍 방법으로 작성한다. 즉, 클라이언트에서 HTTP 요청이 있으면 서버 모듈에서 요청을 받아 바로 스레드를 생성한다. 스레드에서는 태스크 큐(110)에 데이터를 푸시(Push) 하여 저장하고 상기 컨슈머 스레드(124) 및 웹 어플리케이션 서버(130)의 응답을 기다린다.

[0069] 상기 웹 어플리케이션 서버(130)로부터 HTTP 응답을 수신하면(6. Http Response), 상기 HTTP 서버(120)는 클라이언트에 응답을 전송하고(7. Http Response) 스레드를 종료한다.

[0070] 이와 같은 본 발명의 일 실시예에 따르면, 상기 컨슈머 스레드(124)는 HTTP 프로토콜(Protocol)을 처리할 수 있는 기능을 통해 상기 프로듀서 스레드(122)에서 생성한 태스크 큐(110)를 접근하여 비동기 방식으로 데이터를 읽어 들인다.

[0071] 읽어 들인 데이터는 이미 GPU에서 데이터의 정합성, HTTP의 오류, 사용자 입력 값의 오류 등을 확인한 데이터가 구조체에 저장되어 있으므로, 상기 컨슈머 스레드(124)는 상기 구조체의 값을 참고하여 데이터의 오류가 발생하면 바로 프로듀서 스레드(122)에게 오류의 응답을 보낸다. 상기 프로듀서 스레드(122)는 사용자의 응답을 처리한다.

[0072] 상기 프로듀서 스레드(122)에서 생성된 태스크 큐(110)의 스레드는 사용자 요청의 데이터를 검사하여 오류가 발생하면 부울 형의 구조체 변수에 오류 발생 여부와 오류 코드를 저장한다. 상기 컨슈머 스레드(124)에서는 부울 형의 변수 값을 확인하여 오류가 발생하면 오류 코드와 함께 상기 프로듀서 스레드(122)에 바로 응답을 보낸다. 상기 프로듀서 스레드(122)는 오류가 없으면 HTTP 패킷을 상기 웹 어플리케이션 서버(130)에 전송한다.

[0073] 한편, 본 발명의 일 실시예에 따르면 상기 컨슈머 스레드(124)는 GPU에 태스크 큐(110)를 생성한다. GPU에 대량의 스레드를 생성하기 위하여, 상기 컨슈머 스레드(124)는 CUDA(Compute Unified Device Architecture) 어플리케이션을 작성하여 생성하고, GPU에서 비동기 방식으로 사용자 요청의 데이터를 수신하기 위하여 스레드를 생성하고 기다린다.

[0074] 상기 컨슈머 스레드(124)는 사용자 요청을 태스크 큐(110)에서 수신하면 오류 코드를 확인하여 오류 발생이 있으면 바로 상기 프로듀서 스레드(122)에 오류를 통보한다. 반면, 오류 발생이 없으면 상기 컨슈머 스레드(124)는 상기 웹 어플리케이션 서버(130)에 데이터를 전송하여 사용자 요청을 처리한다. 상기 웹 어플리케이션 서버(130)에서 처리한 사용자 요청에 대한 응답은 상기 프로듀서 스레드(122)에 의해서 사용자에게 전송된다.

[0076] 시스템 구현

[0078] GPGPU 기반의 태스크 큐를 CUDA를 사용하여 구현한다. 또한 태스크 큐의 알고리즘은 원형 큐 알고리즘을 사용하고, CUDA 스레드가 원형 큐와 같이 작동하여 간단한 어플리케이션 처리가 가능하도록 구현하였다.

[0079] 도 3은 GPGPU 기반 태스크 큐의 CUDA 커널 함수에서 호출하여 사용되는 HTTP 패킷 오류처리 함수의 일례를 나타낸 도면이다.

[0080] 도 3을 참조하면, HTTP 패킷 오류처리 함수는 CUDA의 커널 함수에서 호출되어 GPU에 대량의 스레드와 함께 생성된다. CUDA 커널 함수에서 정의된 task_queue_data의 구조체와 그 구조체의 크기를 파라미터로 받아서 HTTP 패킷의 적합성을 확인하여 error_flag에 값을 넣어 반환한다.

[0082] HTTP 서버 Producer와 Consumer 서버를 구현하였다. Producer 서버는 사용자의 요구를 받아 데이터를 태스크 큐에 전송하는 역할을 수행하고, Consumer 서버는 태스크 큐에서 사용자의 데이터를 읽어 오류 처리와 웹 어플리케이션에 사용자의 요구사항을 전송하는 역할을 수행한다. 도 3과 같이 Producer 서버는 사용자 클라이언트로부터 HTTP Request 받아서 task_queue_data 구조체에 처리할 데이터를 분리하여 저장하여 태스크 큐에 전송하는 멀티 스레드 서버이다.

[0083] Producer 서버는 클라이언트로부터 HTTP Request 패킷을 수신 받는다. HTTP Request 원본 데이터로부터 Http Request 데이터를 구조체에 분리하여 저장하며, 사용자의 http_user_data 구조체에 사용자가 입력한 데이터를 분리하여 저장한다. 이렇게 각각의 데이터를 구조체에 모두 저장하면 task_queue의 구조체가 완성되고, task_queue의 구조체를 GPGPU 태스크 큐에 push한다.

[0084] Producer가 태스크 큐의 Rear값을 Consumer와 공유하기 위하여 공유 메모리에 저장된 shared_mem 구조체를 공유한다. 도 4와 같이 HTTP 서버 Producer를 구현하였다. Consumer 서버의 구조는 먼저 태스크 큐의 Front 정보를 공유 메모리로부터 읽기 위하여 공유 메모리를 초기화 한다. 다음에 태스크 큐로부터 데이터를 읽어 error_flag를 확인하여 데이터에 오류가 없으면 웹 어플리케이션 서버(WAS)에 HTTP Request를 Forwarding 한다.

[0086] Consumer 서버의 스레드 구조가 간단한 이유는 GPU의 태스크 큐에서 데이터 처리에 대한 프로그램을 수행하여 Consumer 서버에서는 오류 확인 후 바로 WAS 서버에 데이터를 전송할 수 있기 때문이다. 따라서 스레드 구조가 간단한 Consumer 서버를 구현하였다.

[0087] Consumer 서버는 태스크 큐로부터 데이터를 읽어 데이터의 오류를 확인하고 사용자의 요구를 처리하기 위하여 웹 어플리케이션 서버에 전송하는 기능을 구현한 멀티 스레드 서버 프로그램이다. 도 5와 같이 HTTP 서버의 Consumer를 구현하였다.

[0089] 실험 시나리오 및 실험 결과

[0091] 실험은 3가지의 시나리오를 선정하여 GPGPU 기반 태스크 큐를 사용하지 않은 서버와 GPGPU 기반의 태스크 큐 서버에 동일한 방법으로 실험을 진행하였다.

[0092] 시나리오 1은 안정적인 서비스에서 GPGPU 기반 태스크 큐를 사용하지 않은 서버와 GPGPU 기반 태스크 큐의 처리 속도를 테스트하기 위해 실험을 진행하였다.

[0093] 시나리오 2는 짧은 시간에 1,000개의 스레드를 발생시키고 3,000개의 HTTP Request를 요청하는 실험을 진행하였다. 이 실험은 안정성과 성능을 모두 비교할 수 있는 실험이다.

[0094] 시나리오 3은 대량의 스레드를 발생시켜 두 서버의 안정성과 속도를 테스트 하였다. 총 9,000개의 HTTP Request를 발생시켜 서버가 다운되는 현상을 확인하기 위해서 실험을 진행하였다.

[0096] 도 6은 각각의 실험 시나리오별 성능을 나타낸 도면이다.

[0097] 도 6을 참조하면, 두 서버의 성능을 비교하면 약 2배에서 1.3배의 성능 차이(향상)를 볼 수 있다.

[0098] 스레드의 수가 적은 경우 HTTP Request에서는 성능의 차이가 많이 나고 스레드의 수가 증가하고 HTTP Request의 수가 증가하면서 두 서버의 속도 차가 줄어들고 있는 현상을 보인다. GPGPU 기반 태스크 큐를 사용하지 않은 서버와 GPGPU 기반 태스크 큐 서버 간 많은 스레드를 발생시키면 두 서버 간 성능 점점 비슷해지는 현상이 발생한다. 이것은 GPGPU 기반 태스크 큐 서버가 CPU에서 GPU에 많은 양의 데이터가 복사가 발생하면 발생하는 속도 저하 현상이다.

[0099] GPGPU 기반 태스크 큐 서버는 CPU에서 GPU에 데이터를 복사할 때 PCI(Peripheral Component Interconnect) Express 인터페이스를 사용하여 복사가 진행된다. PCI Express 인터페이스의 속도가 느려서 발생하는 현상으로 CPU와 GPU의 데이터 복사 속도를 향상시키면 HTTP Request 증가로 발생하는 속도 저하 현상을 줄일 수 있다.

[0101] 도 7은 두 서버(GPGPU 기반 태스크 큐를 사용하지 않은 서버와 GPGPU 기반 태스크 큐 서버)의 오류율을 나타낸 도면이다.

[0102] 도 7을 참조하면, 시나리오 1번에서는 두 서버 모두 안정적으로 HTTP Request를 처리하고 있다. 시나리오 1번은 성능을 확인하기 위한 실험으로 일반적인 HTTP Request의 처리는 오류가 발생하지 않고 있다.

[0103] 시나리오 2번은 GPGPU 기반 태스크 큐를 사용하지 않은 서버에서 2.43%의 오류가 발생하였다. 짧은 시간에 대

량의 스레드를 발생하여 HTTP Request를 요청하자 요청이 거부되는 현상이 발생하였다. 반면, GPGPU 기반 태스크 큐 서버의 경우 HTTP Request가 모두 안정적으로 처리가 되었다.

[0104] 시나리오 3번은 시나리오 2번보다 더 많은 스레드와 HTTP Request를 요청하였다. GPGPU 기반 태스크 큐를 사용하지 않은 서버의 경우 오류율이 15.4%가 발생하였다. 하지만 서버의 다운되는 현상은 없었으며 나머지 HTTP Request의 처리는 정상적으로 수행되었다. 반면, GPGPU 기반 태스크 큐 서버는 모두 HTTP Request의 처리가 가능하였다.

[0106] 도 6 및 도 7에서 보는 바와 같이, GPGPU 기반 태스크 큐를 사용하지 않은 서버보다 GPGPU 기반 태스크 큐 서버가 성능과 안정성 면에서 모두 우수하다는 것을 알 수 있었다.

[0108] 이와 같이, 본 발명의 일 실시예에서는 웹 서비스의 대량 요청을 처리하고 웹 서버의 안전성을 위하여 GPGPU 기반 태스크 큐를 NVIDIA사의 CUDA를 이용하여 구현하였다.

[0109] 본 발명의 실험 결과는 GPGPU 기반 태스크 큐를 사용한 서버의 성능이 GPGPU 기반 태스크 큐를 사용하지 않은 서버의 성능보다 136%에서 233%까지 향상된 결과를 볼 수 있다. 인공지능, 게임 및 고성능 과학기술 계산 등에 많이 사용되는 GPU를 일반 어플리케이션의 성능 향상에 이용할 수 있는 결과를 보였다.

[0110] 기존 웹 서비스의 구성으로 많은 양의 사용자 요청을 처리하기 위해서 웹 서버의 성능을 향상시키거나 다수 의 웹 서버를 설치하여 운영할 경우 많은 비용이 발생한다. 하지만, 본 발명에서 제시한 GPGPU 기반의 태스크 큐를 사용할 경우 하나의 서버에서 대량의 사용자 요청을 처리할 때 서버에 GPU를 설치하여 활용할 경우, 보다 안정적이고 고성능의 웹 서비스를 저렴한 비용에 제공할 수 있다.

[0111] 또한, GPGPU 기반 태스크 큐 서버에서 보다 많은 HTTP Request를 처리할 수 있는 다중 GPU를 제공할 경우, 웹 서버의 성능을 더 향상시킬 수 있을 것으로 기대된다. 다시 말해, 기존 웹 서비스에 대량의 사용자 요청이 있을 경우, 웹 서비스 중단에 대한 문제를 GPU를 활용하여 저렴한 비용으로 해결할 수 있을 것으로 기대된다.

[0112] GPGPU는 HPC 분야, 게임, 시뮬레이션, 인공지능 분야 등에 많이 이용되고 그 수가 점점 증가하고 있다. GPU를 사용하여 많은 작업들을 수행하고 GPU의 이용 분야도 다양하게 발전하고 있다.

[0113] 최신의 GPU는 그래픽 처리뿐 아니고 GPGPU 프로그램을 지원하는 고성능의 부동 소수점 연산 기능을 제공하고 있다. 또한 대량의 스레드를 생성하여 처리할 수 있도록 그 성능도 계속 발전하고 있다. 이런 GPU의 성능을 일반 어플리케이션에서 사용할 수 있도록 GPU가 계속 발전하고, GPU가 CPU를 보조하여 발전하면 범용의 어플리케이션의 실행 성능이 향상되고 고성능의 범용 어플리케이션을 수행하기 위한 하드웨어도 더 비용을 절감할 수 있다.

[0115] 실시예에 따른 방법은 다양한 컴퓨터 수단을 통하여 수행될 수 있는 프로그램 명령 형태로 구현되어 컴퓨터 판독 가능 매체에 기록될 수 있다. 상기 컴퓨터 판독 가능 매체는 프로그램 명령, 데이터 파일, 데이터 구조 등을 단독으로 또는 조합하여 포함할 수 있다. 상기 매체에 기록되는 프로그램 명령은 실시예를 위하여 특별히 설계되고 구성된 것들이거나 컴퓨터 소프트웨어 당업자에게 공지되어 사용 가능한 것일 수도 있다. 컴퓨터 판독 가능 기록 매체의 예에는 하드 디스크, 플로피 디스크 및 자기 테이프와 같은 자기 매체(magnetic media), CDROM, DVD와 같은 광기록 매체(optical media), 플롭티컬 디스크(floptical disk)와 같은 자기-광 매체(magneto-optical media), 및 롬(ROM), 램(RAM), 플래시 메모리 등과 같은 프로그램 명령을 저장하고 수행하도록 특별히 구성된 하드웨어 장치가 포함된다. 프로그램 명령의 예에는 컴파일러에 의해 만들어지는 것과 같은 기계어 코드뿐만 아니라 인터프리터 등을 사용해서 컴퓨터에 의해서 실행될 수 있는 고급 언어 코드를 포함한다. 상기된 하드웨어 장치는 실시예의 동작을 수행하기 위해 하나 이상의 소프트웨어 모듈로서 작동하도록 구성될 수 있으며, 그 역도 마찬가지이다.

[0116] 이상과 같이 실시예들이 비록 한정된 실시예와 도면에 의해 설명되었으나, 해당 기술분야에서 통상의 지식을 가진 자라면 상기의 기재로부터 다양한 수정 및 변형이 가능하다. 예를 들어, 설명된 기술들이 설명된 방법과 다른 순서로 수행되거나, 및/또는 설명된 시스템, 구조, 장치, 회로 등의 구성요소들이 설명된 방법과 다른 형태로 결합 또는 조합되거나, 다른 구성요소 또는 균등물에 의하여 대치되거나 치환되더라도 적절한 결과가 달성될 수 있다.

[0117] 그러므로, 다른 구현들, 다른 실시예들 및 청구범위와 균등한 것들도 후술하는 청구범위의 범위에 속한다.

부호의 설명

[0119] 110: 태스크 큐

120: HTTP 서버

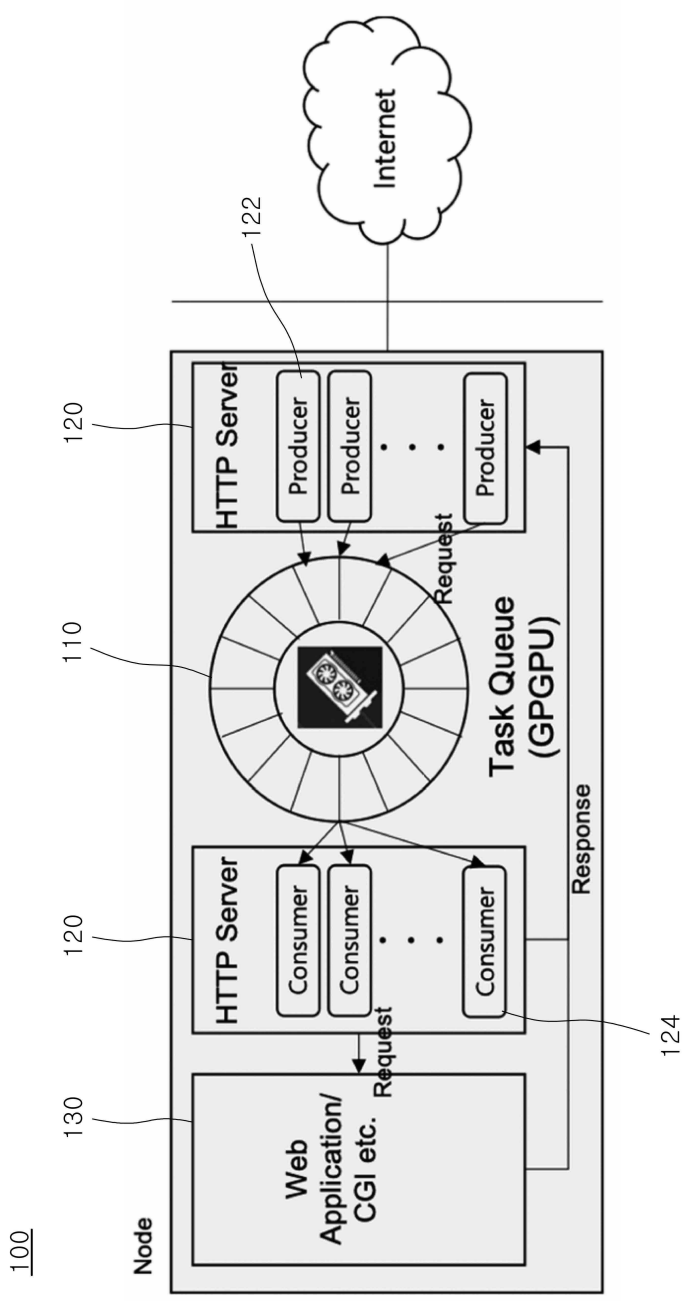
122: Producer 스레드

124: Consumer 스레드

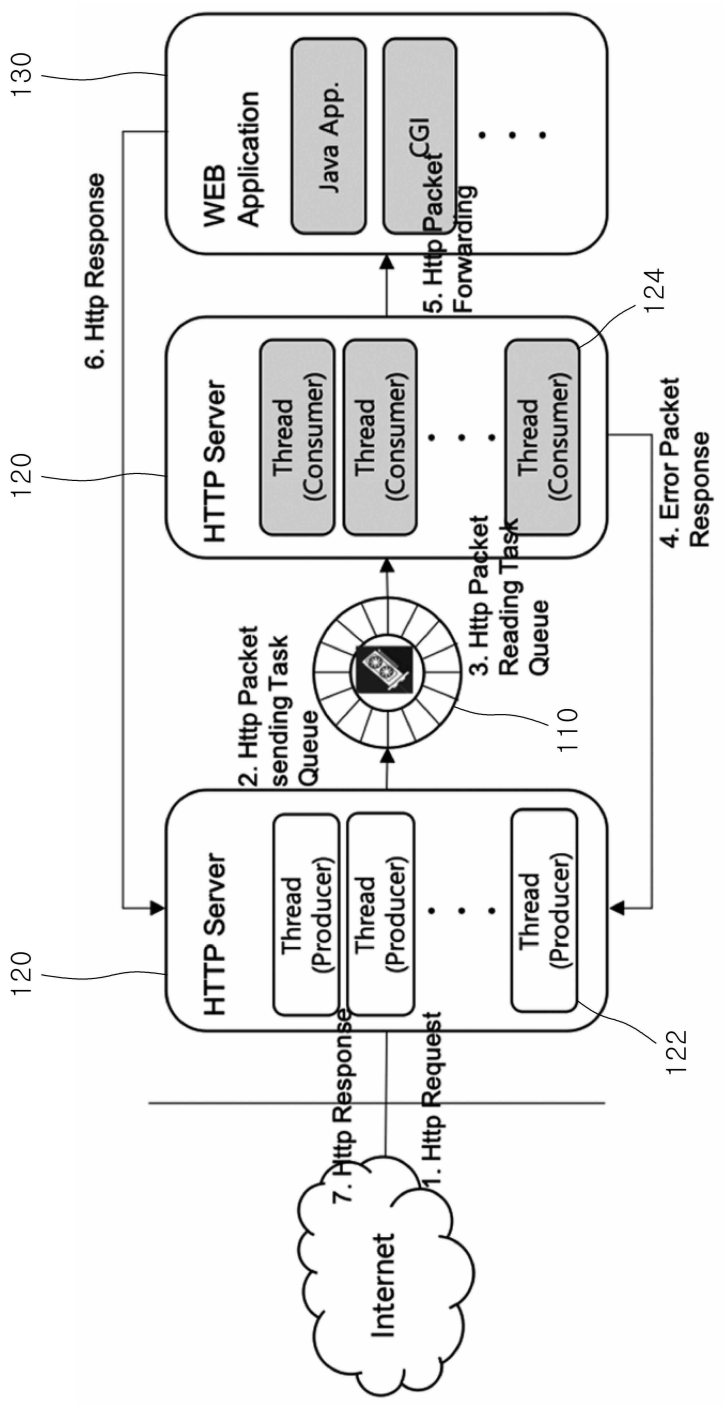
130: 웹 어플리케이션 서버

도면

도면1



도면2


130
WEB Application
Java App.
CGI
. . .
6. Http Response
5. Http Packet Forwarding
120
HTTP Server
Thread (Consumer)
Thread (Consumer)
Thread (Consumer)
124
4. Error Packet Response
3. Http Packet Reading Task Queue
2. Http Packet sending Task Queue
110
120
HTTP Server
Thread (Producer)
Thread (Producer)
Thread (Producer)
122
7. Http Response
1. Http Request
Internet

도면3

```
Var: TaskQueueData(0),
     Size(0),
     JSONStruct(0),
     HttpUserData(0),
    ErrorFlag(0);

parserJSON(TaskQueueData, JSONStruct);
parserHTTPData(TaskQueueData, HttpUserData);

ErrorFlag = validationUserData(JSONStruct,
HttpUserData, Size);

return ErrorFlag;
```

도면4

```
Var: Error(0), SemId(0), HttpPacketLen(0),
     HttpUserDataLen(0), HttpRequestLen(0),
     HttpPacket(0), JSONData(0), JSONLen(0),
     HttpRequest(0), HttpUserData(0), TaskQueue(0),
     SharedMem(0);

SemId = InitSharedMemory(SharedMem);
if SemId < 0 then
begin
   return Error;
end

Error = HttpRequestRecv(HttpPacket, HttpPacketLen);
if error < 0 then
begin
  return Error;
end

Error = HttpRequest(HttpPacket, HttpPacketLen,
                HttpRequest, HttpRequestLen);
if error < 0 then
begin
  return Error;
end
Error = HttpUserdata(HttpPacket, http_packet_len,
          HttpUserData, HttpUserDataLen);
```

도면5

```
Var: Error(0), SemId(0),
    TaskQueue(0), SharedMem(0);

SemId = InitSharedMemory(SharedMem);
if SemId < 0 then
begin
  return SemId;
end

Error = TaskQueuePop(TaskQueue, SharedMem.Front);

if Error < 0 then
begin
  return Error;
end

if TaskQueue.ErrorFlag.HttpProtocolError ||
    TaskQueue.ErrorFlag.UserDataError then
begin
   return -1;
end

Error = WASForwarding(TaskQueue.HttpRequest,
                TaskQueue.HttpReqeustLen);
if Error < 0 then
begin
  return Error;
end

return 0;
```

도면6

시나리오	thread 수	HTTP Request	실행시간		성능비
			태스크 큐 미사용	태스크 큐 사용	
1	100	300	2,566ms	1,097ms	233%
2	1,000	3,000	5,440ms	3,775ms	144%
3	3,000	9,000	8,937ms	6,524ms	136%

도면7

시나리오	thread 수	HTTP Request	Error %	
			태스크 큐 미사용	태스크 큐 사용
1	100	300	0.0%	0.0%
2	1,000	3,000	2.43%	0.0%
3	3,000	9,000	15.4%	0.0%