(19) 대한민국특허청(KR)
(12) 등록특허공보(B1)

(45) 공고일자 2018년08월02일
(11) 등록번호 10-1871961
(24) 등록일자 2018년06월21일

(51) 국제특허분류(Int. Cl.)
G06F 9/52 (2018.01)
(21) 출원번호 10-2014-7012038
(22) 출원일자(국제) 2012년10월31일
심사청구일자 2017년10월25일
(85) 번역문제출일자 2014년05월02일
(65) 공개번호 10-2014-0088550
(43) 공개일자 2014년07월10일
(86) 국제출원번호 PCT/US2012/062768
(87) 국제공개번호 WO 2013/066988
국제공개일자 2013년05월10일
(30) 우선권주장
13/288,833 2011년11월03일 미국(US)
(56) 선행기술조사문헌
Shivali 외 2명. 'Distributed Generalized Dynamic Barrier Synchronization', International Conference on Distributed Computing and Networking, 2011, pp.143-154.
Xiao 외 1명. 'Inter-block GPU communication via fast barrier synchronization', 2010 IEEE International Symposium on Parallel & Distributed Processing, 2010, pp.1-12.
US20060212868 A1
US20090037707 A1

(73) 특허권자
어드밴스드 마이크로 디바이시즈, 인코포레이티드
미국 캘리포니아 95054 산타 클라라 어거스틴 드라이브 2485
(72) 발명자
호웨스 리 더블유.
미국 캘리포니아 95054 산타 클라라 유니트 401 리버 사이드 코트 406
가스터 베네딕트 알.
미국 캘리포니아 95061 산타 크루즈 섬머 스트리트 53
(뒷면에 계속)
(74) 대리인
박장원

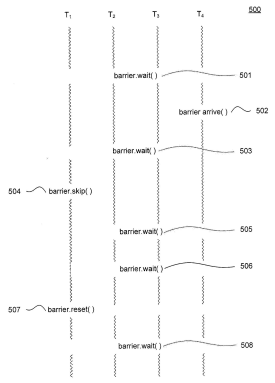
전체 청구항 수 : 총 20 항 심사관 : 유진태

(54) 발명의 명칭 **작업항목 동기화를 위한 방법 및 시스템**

(57) 요 약

하나 이상의 프로세서 상에서 작업항목을 동기화하기 위한 방법, 시스템 및 컴퓨터 프로그램 제품 실시예가 개시된다. 그 실시예는 그룹으로부터의 제1 작업항목에 의해 배리어 스킵 명령어를 실행하는 단계, 및 실행된 배리어 스킵 명령어에 응답하여, 복수의 포인트 중 어느 하나에서 제1 작업항목이 배리어에 도달하도록 요구함이 없이 시퀀스 내 복수의 포인트에서 그룹으로부터의 다른 작업항목을 동기화하도록 배리어를 재구성하는 단계를 포함한다.

대 표 도 - 도5



(72) 발명자

호우스톤 마이클 씨.

미국 캘리포니아 95014 쿠퍼티노 콜룸버스 애비뉴 21330

맨티 마이클

미국 플로리다 32825 올란도 피나르 드라이브 1620

리어서 마크

미국 캘리포니아 95032 로스 가토스 몬트클레어 로드 265

루빈 노먼

미국 메사추세츠 02139 캠브리지 에쎅스 스트리트 22

엠버링 브라이언 디.

미국 캘리포니아 94401 산 마테오 체스터톤 플레이스 214

명 세 서

청구범위

청구항 1

하나 이상의 프로세서 상에서 작업항목의 그룹을 동기화하는 방법으로서,

상기 그룹으로부터의 제1 작업항목에 의해 배리어 스킵 명령어를 실행하는 단계; 및

상기 실행된 배리어 스킵 명령어에 응답하여, 복수의 포인트 중 어느 하나에서 상기 제1 작업항목이 배리어에 도달하도록 요구함이 없이 그리고 상기 복수의 포인트 중 어느 하나에서 상기 제1 작업항목이 상기 배리어에 동기화해야함이 없이 진행하는 것을 허용하면서, 시퀀스 내 상기 복수의 포인트에서 상기 그룹으로부터의 다른 작업항목을 동기화하도록 상기 배리어를 재구성하는 단계를 포함하는, 동기화 방법.

청구항 2

제1항에 있어서,

동기화 포인트의 시퀀스에서 상기 그룹을 동기화하도록 상기 배리어를 구성하는 단계를 더 포함하되, 상기 시퀀스는 상기 복수의 포인트를 포함하는 것인 동기화 방법.

청구항 3

제1항에 있어서, 다른 작업항목을 동기화하도록 상기 배리어를 재구성하는 단계는,

상기 배리어와 연관된 스킵 카운트를 증분하는 단계를 포함하되, 상기 스킵 카운트는 상기 그룹으로부터의 모든 작업항목이 상기 배리어에 도달하였는지 결정하는데 사용되는 것인 동기화 방법.

청구항 4

제1항에 있어서,

상기 복수의 포인트 중 제1 및 제2 포인트에서 상기 다른 작업항목을 동기화하는 단계를 더 포함하되, 상기 제1 작업항목은 상기 제1 및 제2 포인트에서 상기 배리어에 도달하지 않은 것인 동기화 방법.

청구항 5

제4항에 있어서, 상기 다른 작업항목을 동기화하는 단계는,

상기 배리어에 도달하는 상기 다른 작업항목의 각각에 대하여, 그것이 상기 다른 작업항목 중 마지막 것인지 결정하는 단계; 및

상기 다른 작업항목 중 상기 마지막 것이 상기 배리어에 도달할 때, 프로세싱을 재개하도록 모든 상기 다른 작업항목을 블로킹 해제하는 단계를 포함하는 것인 동기화 방법.

청구항 6

제5항에 있어서, 그것이 상기 다른 작업항목 중 상기 마지막 것인지 결정하는 단계는,

스킵 카운트와 방문 카운트의 합을 릴리스 임계값과 비교하는 단계를 포함하되, 상기 스킵 카운트는 상기 스킵 명령어가 실행될 때 증분되고, 상기 방문 카운트는 작업항목이 상기 배리어에 도달할 때 증분되고, 상기 릴리스 임계값은 상기 그룹 내 작업항목의 수에 대응하는 것인 동기화 방법.

청구항 7

제4항에 있어서, 상기 다른 작업항목을 동기화하는 단계는,

상기 제1 포인트에서 상기 다른 작업항목을 동기화하는 단계;

상기 다른 작업항목만을 동기화하도록 상기 배리어를 일부-재설정하는 단계; 및

상기 다른 작업항목만을 동기화하도록 상기 배리어의 상기 일부-재설정에 뒤따라 상기 제2 포인트에서 상기 다른 작업항목을 동기화하는 단계를 포함하는 것인 동기화 방법.

청구항 8

제7항에 있어서, 상기 다른 작업항목만을 동기화하도록 상기 배리어를 일부-재설정하는 단계는,

어느 작업항목도 상기 배리어에 도달하지 않았음을 나타내는 초기값으로 상기 배리어와 연관된 방문 카운트를 설정하는 단계; 및

스킵 카운트의 값을 지속시키는 단계를 포함하되,

상기 방문 카운트는 상기 배리어에 도달하는 작업항목의 수를 표현하고, 상기 스킵 카운트는 작업항목이 배리어 스킵 명령어를 발행할 때 증분되는 것인 동기화 방법.

청구항 9

제1항에 있어서,

상기 그룹으로부터의 어느 작업항목에 의해 배리어 재설정 명령어를 실행하는 단계; 및

상기 배리어 재설정 명령어에 응답하여, 상기 그룹을 동기화하기 위해 상기 그룹으로부터의 모든 작업항목이 상기 배리어에 도달하는 것을 요구하도록 상기 배리어를 더 재구성하는 단계를 더 포함하는 동기화 방법.

청구항 10

제9항에 있어서, 모든 작업항목을 요구하도록 상기 배리어를 더 재구성하는 단계는,

어느 작업항목도 상기 배리어에 도달하지 않았음을 나타내는 초기값으로 상기 배리어와 연관된 방문 카운트를 설정하는 단계; 및

어느 작업항목도 상기 배리어와 연관된 배리어 스킵 명령어를 발행하지 않았음을 나타내는 초기값으로 상기 배리어와 연관된 스킵 카운트를 설정하는 단계를 포함하는 것인 동기화 방법.

청구항 11

제1항에 있어서, 상기 작업항목의 그룹은 작업그룹이고, 상기 작업그룹은 그래픽 프로세싱 유닛의 프로세싱 엘리먼트에서 실행 중인 것인 동기화 방법.

청구항 12

제1항에 있어서, 상기 그룹은 그래픽 프로세싱 유닛에서 실행하는 웨이브프론트의 2개 이상의 작업그룹으로부터의 작업항목을 포함하는 것인 동기화 방법.

청구항 13

제1항에 있어서, 상기 그룹은 CPU 상에서 실행하는 작업항목을 포함하는 것인 동기화 방법.

청구항 14

제1항에 있어서, 동기화 포인트의 시퀀스에서 상기 그룹을 동기화하도록 상기 배리어를 구성하는 단계는 라이브러리 함수에 포함된 명령어에 응답하여 수행되는 것인 동기화 방법.

청구항 15

시스템으로서,

하나 이상의 프로세서;

상기 하나 이상의 프로세서 상에서 실행하는 작업항목의 그룹; 및

상기 하나 이상의 프로세서에 의해 실행될 때, 상기 하나 이상의 프로세서로 하여금,

상기 그룹으로부터의 제1 작업항목에 의해 배리어 스킵 명령어를 실행하고; 그리고

상기 실행된 배리어 스킵 명령어에 응답하여, 복수의 포인트 중 어느 하나에서 상기 제1 작업항목이 배리어에 도달하도록 요구함이 없이 그리고 상기 복수의 포인트 중 어느 하나에서 상기 제1 작업항목이 상기 배리어에 동기화해야함이 없이 진행하는 것을 허용하면서, 시퀀스 내 상기 복수의 포인트에서 상기 그룹으로부터의 다른 작업항목을 동기화하도록 상기 배리어를 재구성하게 야기하도록 구성된 배리어 동기화 모듈을 포함하는 시스템.

청구항 16

제15항에 있어서, 상기 배리어 동기화 모듈은, 상기 하나 이상의 프로세서에 의해 실행될 때, 상기 하나 이상의 프로세서로 하여금,

동기화 포인트의 시퀀스에서 상기 그룹을 동기화하도록 배리어를 구성하게 야기하도록 더 구성되되, 상기 시퀀스는 상기 복수의 포인트를 포함하는 것인 시스템.

청구항 17

제15항에 있어서, 상기 하나 이상의 프로세서는 이종 컴퓨팅 시스템의 일부분인 것인 시스템.

청구항 18

제15항에 있어서, 상기 배리어 동기화 모듈은, 상기 하나 이상의 프로세서에 의해 실행될 때, 상기 하나 이상의 프로세서로 하여금,

상기 그룹으로부터의 어느 작업항목에 의해 배리어 재설정 명령어를 실행하고; 그리고

상기 배리어 재설정 명령어에 응답하여, 상기 그룹을 동기화하기 위해 상기 그룹으로부터의 모든 작업항목이 상기 배리어에 도달하는 것을 요구하도록 상기 배리어를 더 재구성하게 야기하도록 더 구성되는 것인 시스템.

청구항 19

커맨드를 저장하는 비일시적인 컴퓨터 가독 매체(computer readable medium)로서, 상기 커맨드는, 실행되면,

그룹으로부터의 제1 작업항목에 의해 배리어 스킵 명령어를 실행하는 단계; 및

상기 실행된 배리어 스킵 명령어에 응답하여, 복수의 포인트 중 어느 하나에서 상기 제1 작업항목이 배리어에 도달하도록 요구함이 없이 그리고 상기 복수의 포인트 중 어느 하나에서 상기 제1 작업항목이 상기 배리어에 동기화해야함이 없이 진행하는 것을 허용하면서, 시퀀스 내 상기 복수의 포인트에서 그룹으로부터의 다른 작업항목을 동기화하도록 상기 배리어를 재구성하는 단계를 포함하는 방법을 야기하는 것인 비일시적인 컴퓨터 가독 매체.

청구항 20

제19항에 있어서, 상기 커맨드는, 실행되면,

동기화 포인트의 시퀀스에서 그룹을 동기화하도록 배리어를 구성하는 단계를 더 포함하는 방법을 야기하는 것인 비일시적인 컴퓨터 가독 매체.

발명의 설명

기 술 분 야

[0001] 본 발명은 일반적으로는 작업항목 동기화에 관한 것이다.

배 경 기 술

[0002] 그래픽 프로세싱 유닛(GPU)는 일반적으로는, 단일 명령어 다중 데이터(SIMD) 디바이스의 경우 또는 데이터-병렬 프로세싱에서와 같이, 병렬 데이터 스트림 상에 동일 명령어를 실행하기에 이상적으로 적합한 다중 프로세싱 엘리먼트를 포함한다. 많은 컴퓨팅 모델에 있어서, 중앙 프로세싱 유닛(CPU)은 GPU와 같은 다른 프로세서에 대해 그래픽 프로세싱과 같은 핸드-오프 전문 기능 및 제어 프로세서 또는 호스트로서 기능한다.

[0003] 각각의 CPU가 다중 프로세싱 코어를 갖는 멀티-코어 CPU는 GPU 상에서 이용가능한 것들과 유사한 전문 기능(그래픽 프로세싱)을 위한 프로세싱 능력을 제공한다. 멀티-코어 CPU 또는 GPU의 계산 코어 중 하나 이상은 동일

다이(예컨대, AMD Fusion(상표명)) 또는 대안으로 다른 다이(예컨대, NVIDIA GPU를 갖는 Intel Xeon(상표명))의 일부분일 수 있다. 최근에는, CPU 및 GPU 둘 다의 특성을 갖는 하이브리드 코어(예컨대, CellSPE(상표명), Intel Larrabee(상표명))가 범용 GPU(GPGPU) 스타일 컴퓨팅을 위해 제안되어 왔다. GPGPU 스타일의 컴퓨팅은 제어 코드를 주로 실행하고 GPU에 퍼포먼스 크리티컬 데이터-병렬 코드를 분담하도록 CPU를 사용하는 것을 지지한다. GPU는 주로 가속기로 사용된다. 멀티-코어 CPU 및 GPGPU 컴퓨팅 모델의 조합은 가속기 표적으로서 CPU 코어 및 GPU 코어 둘 다를 아우른다. 멀티-코어 CPU 코어 중 다수는 많은 영역에 있어서 GPU에 필적할만한 성능을 갖는다.

[0004] CPU 및 GPU를 갖는 이종 컴퓨팅 플랫폼을 위해 수개의 프레임워크가 개발되어 왔다. 이들 프레임워크는 Stanford University에 의한 BrookGPU, NVIDIA에 의한 쿠다(CUDA: compute unified device architecture), 및 Khronos Group으로 명명된 산업 컨소시엄에 의한 OpenCL을 포함한다. OpenCL 프레임워크는 사용자가 GPU용 애플리케이션을 생성할 수 있다 C-같은 개발 환경을 제안한다. OpenCL은 사용자가, 예컨대, 데이터-병렬 계산과 같은 일부 계산을 GPU에 분담하기 위한 명령어를 특정할 수 있게 한다. OpenCL은 또한 이종 또는 다른 컴퓨팅 시스템 내에서 코드가 컴파일 및 실행될 수 있는 런타임 환경 및 컴파일러를 제공한다.

[0005] OpenCL, CUDA 및 많은 저레벨 GPU 중간 언어에 의해 구체화된 컴퓨팅 모델은 때로는 단일 명령어 다중 스레드(single instruction multiple thread: "SIMT")라고 알려져 있다. SIMT 모델의 빈번하게 사용되는 구현에 있어서, 벡터 상에 하드웨어 마스크 세트를 사용하는 SIMD 실행은 하드웨어에서 이용가능한 것보다는 더 미세한 그레인으로 스레딩을 시뮬레이팅하도록 사용된다.

[0006] CPU 및 GPU가 둘 다 많은 유형의 코드의 실행에 사용될 수 있는 컴퓨팅 모델을 효율적으로 이용하기 위해서는, 더 융통성 있는 스레드 동기화 모델이 필요로 된다. OpenCL에서 이용가능한 관용적 작업항목 동기화는, 예컨대, 전통적 그래픽 프로세싱 태스크가 아닌 작업항목과 사용될 때 덜 효율적일 수 있다. 예컨대, 그래픽 프로세싱 태스크에서의 각자의 동시다발적 작업항목이 종종 발산하지 않을 수 있는 반면, CPU 연관된 애플리케이션에서는 실행의 발산이 비교적 더 높을 수 있다. OpenCL에서 제공된 동기화 모델은, 스스로, 작업항목의 그러한 동적 거동을 처리하기에 불충분하다.

[0007] OpenCL에 있어서, 작업그룹 내 작업항목은 동기화될 수 있다. 작업그룹 내에서, 배리어 명령어는 어느 것이 배리어를 지나 진행할 수 있기 전에 작업그룹에서의 모든 작업항목이 배리어에 도달해야 한다는 시맨틱스(semantics)로 발행될 수 있다. "배리어"는 위에서 언급된 시맨틱스가 만족될 때까지 그에 도달하는 모든 프로세스를 블로킹한다. 그 후 그것은 그 블로킹된 프로세스를 릴리스하여 그 각자의 프로세싱을 재개한다. 그렇지만, OpenCL 및 다른 유사한 관용적 프레임워크에 있어서, 제어 흐름 내 배리어 사용은 심하게 제한된다.

[0008] 도 1a는 다른 작업항목이 로딩된 값을 획득하여 올 수 있는 그룹-공유된 메모리 공간에 값을 로딩하기 위해 (도 1a에서는 커널이라고 지칭되는) 단일 작업항목의 사용을 예시하고 있다. 값을 로딩하는 작업항목과 더불어, 작업그룹 내 다른 작업항목은 그룹 내 모든 작업항목이 배리어에 도달할 때까지 배리어 너머로 진행하지 못하게 블로킹된다.

[0009] 그룹-공유된 메모리에 값을 로딩하는 위 동작은 또한 커널 코드로부터 호출되는 라이브러리 함수 내로부터 행해질 수 있다. 도 1b는 배리어 명령어를 포함하는 라이브러리 함수를 예시하고 있다. 도 1c는 라이브러리 함수를 호출하는 커널이다. 도 1c에서의 코드는 지정된 작업항목이 공유된 영역에 데이터를 로딩 완료할 때까지 대응하는 라이브러리를 호출하는 모든 작업항목을 블로킹하는 동작을 예시하고 있다.

[0010] 도 1d는 라이브러리 내부 배리어에 호출을 놓는 것이 잘못된 동작에 이를 수 있는 일례를 예시하고 있다. 예컨대, 조건문 중 하나가 함수로의 호출을 갖지 않는, 조건문을 갖는 커널로부터의 배리어 명령어를 포함하는 라이브러리 함수를 호출하는 것은 데드로크에 이를 수 있다. 이것은 그룹의 모든 작업항목이 그에 도달하였을 때에만 배리어가 릴리스할 것인 한편 조건이 이행되지 않는 하나 이상의 작업항목은 배리어에 전혀 도달하지 않을 것이기 때문이다.

[0011] 더욱, OpenCL이 작업항목 동기화 프레임워크에 있어서, 커널을 실행하는 작업그룹 내 작업항목은 어느 것이라도 실행을 계속하도록 허용되기 전에 배리어 명령어를 실행해야 한다. 배리어 명령어는 커널을 실행하는 작업그룹 내 모든 작업항목에 의해 마주쳐져야 한다(즉, 명령어 스트림에서 도달되어야 한다). 배리어 명령어가 조건문 내부에 있으면, 그때 모든 작업항목은 어느 작업항목이 조건문에 들어가고 배리어를 실행하면 조건문에 들어가야 한다. 배리어 명령어가 루프 내부에 있으면, 모든 작업항목은 어느 것이라도 배리어 너머로 실행을 계속하도록 허용되기 전에 루프의 각각의 반복을 위해 배리어 명령어를 실행해야 한다. 이들 제약은 프로세싱 자원을 최

적으로 이용하기 위한 시스템의 그리고 프로그래머의 능력을 제한할 수 있다.

[0012] 그래서, 필요한 것은 작업항목 동기화의 더 융통성 있고 효율적인 사용을 가능하게 하는 방법 및 시스템이다.

발명의 내용

[0013] 작업항목의 더 효율적이고 융통성 있는 스케줄링을 위한 방법 및 시스템이 개시된다. 동기화 그룹 내 작업항목의 실행에서 일어나는 후속 배리어가 그룹으로부터의 그 이탈을 공시한 작업항목을 대기하지 않게 되도록 작업항목이 그것이 그 동기화 그룹을 영구적으로 떠남을 나타내는 기술이 개시된다. 작업항목이 동기화 그룹의 다른 작업항목과 동기화를 계속하기 위해 그 그룹에 재합류할 수 있는 추가적 기술이 개시된다. 개시된 기술은 여러 다양한 상황에서 프로그래밍하는데 있어서 개선된 프로세싱 효율성 및 융통성에서의 상당한 이점을 내놓을 수 있다.

[0014] 개시된 방법, 시스템 및 컴퓨터 프로그램 제품 실시예는 그룹으로부터의 제1 작업항목에 의해 배리어 스킵 명령어를 실행하는 단계, 및, 실행된 배리어 스킵 명령어에 응답하여, 제1 작업항목이 복수의 포인트 중 어느 하나에서의 배리어에 도달할 것을 요구함이 없이 시퀀스 내 복수의 포인트에서의 그룹으로부터의 다른 작업항목을 동기화하도록 배리어를 재구성하는 단계를 포함한다.

[0015] 본 발명의 추가적 실시예, 특징 및 이점과 더불어, 본 발명의 다양한 실시예의 구조 및 동작이 수반 도면을 참조하여 아래에서 상세하게 설명된다.

도면의 간단한 설명

[0016] 명세서의 일부를 구성하고 그에 편입되어 있는 수반 도면은 본 발명의 실시예를 예시하고, 위에서 주어진 일반적 설명 및 아래에서 주어지는 실시예의 상세한 설명과 함께, 본 발명의 원리를 설명하도록 역할한다. 도면에 있어서:

도 1a 내지 도 1d는 의사코드로 관용적 배리어 동기화 예를 나타내는 예시도;

도 2a는 본 발명의 일 실시예에 따른 배리어 스킵 명령어를 (의사코드로) 나타내는 예시도;

도 2b는 실시예에 따라 라이브러리 호출을 사용하는 배리어 스킵 명령어를 갖는 커널을 (의사코드로) 나타내는 예시도;

도 3은 실시예에 따라 배리어 재설정 명령어를 (의사코드로) 나타내는 예시도;

도 4a 내지 도 4c는 실시예에 따라 배리어 동기화를 위한 예시적 유스 케이스의 의사코드 샘플을 나타내는 예시도;

도 5는 실시예에 따라 수개의 작업항목의 시간에 따른 예시적 흐름의 예시도;

도 6은 실시예에 따라 작업항목 동기화를 위한 방법의 예시도;

도 7은 일 실시예에 따라 작업항목 동기화를 위한 시스템의 블록 선도;

도 8은 일 실시예에 따라 작업항목 동기화 모듈의 블록 선도.

발명을 실시하기 위한 구체적인 내용

[0017] 본 발명이 특정 애플리케이션에 대한 예시적 실시예로 본 명세서에서 설명되고 있지만, 본 발명은 그에 국한되는 것은 아님을 이해해야 한다. 본 명세서에서 제공된 교시에 접근하는 당업자는 본 발명의 범위 및 본 발명이 중요하게 이용될 부가적 분야 내 부가적 수정, 애플리케이션 및 실시예를 인식할 것이다.

[0018] 본 발명의 실시예는, 어떠한 컴퓨터 시스템, 컴퓨팅 디바이스, 엔터테인먼트 시스템, 미디어 시스템, 게임 시스템, 통신 디바이스, PDA, 또는 하나 이상의 프로세서를 사용하는 어떠한 시스템에서라도 사용될 수 있다. 본 발명은 시스템이 이종 컴퓨팅 시스템을 포함하는 경우에 특히 유용할 수 있다. "이종 컴퓨팅 시스템"은, 그 용어가 본 명세서에서 사용되는 바와 같이, 다수 종류의 프로세서가 이용가능한 컴퓨팅 시스템이다.

[0019] GPU에 있어서, 프로세싱 엘리먼트에 할당된 작업항목은 "작업그룹"이라고 지칭된다. 병렬로 실행을 위해 발행되는 2개 이상의 작업항목은 "웨이브프론트"이다. 작업그룹은 하나 이상의 웨이브프론트를 포함할 수 있다. 작업그룹의 작업항목을 동기화하는 것과 관련하여 실시예가 주로 설명되고 있지만, 본 발명의 교시는 공유된 메모리

로의 액세스를 갖는 어느 하나 이상의 프로세서 및/또는 프로세서 그룹을 가로지르는 작업항목을 동기화하도록 적용될 수 있다. 본 명세서에서 사용되는 바와 같이 용어 "커널"은 동일 코드 베이스를 갖는 병렬의 하나 이상의 작업항목으로서 실행되는 프로그램 및/또는 프로세싱 로직을 지칭한다. 일부 실시예에 있어서 용어 "작업항목" 및 "스레드"는 상호교환가능함을 주목해야 한다. 본 개시에 있어서 "작업항목"과 "스레드"의 상호교환가능성은, 예컨대, 실시예에서의 모델에서 구체화되는 작업항목의 실행의 융통성 있는 시뮬레이팅된 또는 진정한 독립성의 예시이다.

[0020] 본 발명의 실시예는 동시다발적 작업항목 간 더 효율적 및 더 융통성 있는 동기화를 가능하게 함으로써 시스템의 성능을 상당히 개선할 수 있다. 예컨대 SIMD 또는 SIMT 프레임워크를 사용하여 아주 다수의 동시다발적 작업항목을 실행하는 GPU, 멀티-코어 CPU 또는 다른 프로세서에 있어서, 실시예는 명령어 흐름에서의 수개의 포인트에서 그 실행을 동기화하는 작업항목 그룹을 일부 작업항목이 떠나고 그리고/또는 재합류 가능하게 함으로써 효율성을 개선한다. 예컨대, 특정 작업항목이 동기화 그룹의 나머지와 더 이상의 동기화를 요구하지 않을 때, 그것은 그 자신을 동기화 그룹으로부터 영구적으로 제거하도록 배리어 스킵 명령어를 발행할 수 있다.

[0021] 특정 작업항목이 동기화 그룹에 다시 포함되어야 하면 배리어 재설정 명령어가 후속하여 발행될 수 있다. 실제로는, 스킵 명령어는 대응하는 작업항목이 배리어가 재설정되기 전에는 어떠한 포인트에서도 배리어에 도달하지 않을 것임을 선언한다. 재설정 명령어는 동일 배리어가 재사용되도록 허용한다.

[0022] 작업항목이 배리어를 영구적으로 스킵 가능하게 함으로써, 다양한 성능 개선이 달성될 것이다. 예컨대, (도 2a 내지 도 2b에 예시된 바와 같이) 프로그래밍 융통성이 상당히 개선되고, 그리하여 더 효율적 코드의 생성을 허용한다. (특히 다수의 동시다발적 작업항목을 갖는 시스템에 있어서) 루프 거동이 또한 개선된다.

[0023] 예컨대, (배리어 상에서 도착 및 대기 동작을 포함하는 시스템을 포함하는) 관용적 시스템에 있어서, 모든 작업항목은 가장 긴 반복 작업항목이 그 반복을 완료할 때까지 배리어를 포함하는 루프에 의해 저지된다. 대조적으로, 본 발명의 실시예에 있어서, 그룹과의 추가적 동기화를 요구하지 않는 작업항목은 시스템에서 데드로크를 야기함이 없이 루프를 빠져나갈 수 있다.

[0024] 프로세싱 속도에서의 개선이 전력 효율성에서의 개선과 더불어 실현된다.

[0025] 도 2는 본 발명의 일 실시예에 따라 배리어 스킵 명령어(예컨대, b.skip())가 사용되는 함수를 (의사코드로) 예시하고 있다. 명명된 배리어 b가 선언되고, 조건 루프를 빠져나간 후 스킵 명령어가 발행된다. 루프 내에서, 작업항목은 특정 조건이 만족될 때까지 b 상에서 대기한다.

[0026] (도 2a에 예시된) "theKernel" 함수에 대응하는 모든 작업항목이 루프 동안 동일 또는 유사한 수의 반복을 반복하지는 않을 수 있다. 각각의 작업항목이 루프를 빠져나감에 따라, 스킵 명령어가 발행된다. 빠져나가는 작업항목에 의한 스킵 명령어의 발행은 빠져나가는 작업항목이 배리어에 또다시 도달하지는 않을 것임을 루프를 아직 떠나지 않은 다른 작업항목에 나타낸다. 그리하여, 배리어는, 적어도, "배리어 재설정"이라고 지칭되는 또 다른 명령어가 발행될 때까지 현재 및 후속 인스턴스화에 있어서 그 빠져나가는 작업항목을 대기하는 것을 회피하도록 재구성될 수 있다. 예컨대, 배리어에 도달하도록 요구되는 작업항목의 수가 그 빠져나가는 작업항목을 설명하도록 감축될 수 있다.

[0027] 관용적 시스템에 있어서, 루프 내 배리어 함수(예컨대, barrier() 또는 barrier().wait)를 포함한 작업항목은 효율적으로 처리될 수 없다. 예컨대, 데드로크를 방지하기 위해, 각각의 반복에 있어서 모든 작업항목에 의해 배리어가 도달(방문)되도록 그룹 내 모든 작업항목이 동일 횟수 반복하게 강제할 필요가 있었다. 그러한 접근법은 작업항목이 서로 다른 실행 경로를 가질 수 있는 환경에서는 명확히 낭비적일 것이다.

[0028] 또 다른 관용적 동기화 명령어는 배리어 도착이다. 그렇지만, 도착 명령어는 현재 배리어로부터의 호출자를 릴리스할 뿐이다. 예컨대, 도 2a에 있어서 스킵 명령어가 도착 명령어로 교체되면, 배리어는 여전히 그 빠져나가는 작업항목이 후속 배리어 인스턴스에서 배리어에 도달하도록 요구할 것이다. 그리하여, 도착 명령어는 작업항목이 그 자신을 동기화 그룹으로부터 영구적으로 제거할 수 있는 메커니즘을 제공하지 않는다.

[0029] 도 2b는 본 발명에 따라, 선택된 작업항목이 다른 작업항목과 공유된 공간에 데이터를 복사하도록 허용하는 라이브러리 함수 "loadFunction"를 의사코드로 예시하고 있다. 라이브러리 함수는 배리어 대기 명령어(예컨대, b.wait())로의 2개의 호출을 포함하고 있다. 2개의 배리어 대기 명령어는 라이브러리 함수를 호출하는 어떠한 작업항목에 의해서라도 발행될 것이다.

[0030] 도 2b에 있어서, 커널 함수 "theKernel"은 특정 조건이 만족될 때 loadFunction을 호출한다.

[0031] (도 2b에 있어서) 조건문의 else 부분에서의 스킵 명령어는 조건을 만족하지 않는 모든 작업항목이 스킵 명령어를 호출함을 보장한다. 그리하여, 조건을 만족하지 못하는 모든 작업항목은 배리어의 후속 인스턴스 둘 다로부터 면제될 것이고, 배리어 인스턴스는 이들 작업항목 상에서 대기하지 않을 것이다. 명령어 흐름에 있어서 배리어의 인스턴스는 동기화 포인트라고 지칭될 수 있다. 2개의 동기화 포인트는 라이브러리 함수에서 배리어 대기(b.wait())로의 2개의 호출에 대응한다. 각각의 작업항목은 loadFunction을 호출하여 그로써 2개의 동기화 포인트에서 배리어에 도달하든지 또는 배리어의 어느 후속 인스턴스에 도달해야 하는 것으로부터 그 자신을 면제하는 스킵 명령어를 호출하든지 하기 때문에 데드로크는 일어나지 않을 것이다.

[0032] 대조적으로, 스킵 명령어 대신에 관용적 도착 명령어가 도 2b에서 사용되었다면, 데드로크가 일어날 것이다. 예컨대, 조건을 만족하지 않는 각각의 작업항목이 도착 명령어를 실행하더라도, 도착 명령어는 발행하는 작업항목을 바로 다음의 배리어로부터 면제할 뿐이다. 발행하는 작업항목은 제2 동기화 포인트에서 배리어에 도달할 것으로 예상될 것이고, 그래서, 데드로크가 일어날 것이다.

[0033] 도 3은 일 실시예에 따라 배리어 재설정 명령어(예컨대, b.reset)의 사용을 의사코드로 예시하고 있다. 커널 함수 "theKernel"는 도 2b에 예시된 바와 같이 특정 조건이 만족될 때 loadFunction을 호출한다. 그렇지 않으면, 그것은 스킵 명령어를 발행하고 재설정 명령어가 뒤따른다.

[0034] 도 2b와 관련하여 위에서 설명된 바와 같이, 스킵 명령어의 발행은 조건을 만족하지 못하는 모든 작업항목이 후속 동기화 포인트 둘 다로부터 면제될 것임을 보장한다. 동기화 포인트에서의 배리어는 이들 작업항목 상에서 대기하지 않을 것이다. 배리어 재설정 명령어는 배리어를 그 원래 구성으로 재설정한다. 예컨대, 실제로는 스킵 명령어가 배리어에 대한 동기화 그룹의 사이즈를 감축한 반면, 재설정 명령어는 그룹 내 작업항목에 의해 발행된 어느 선행 스킵 명령어의 효과를 역전시킨다. 재설정 명령어의 구현은 재설정이 완료되기 전에 어느 보류 배리어가 동기화되는 것을 요구한다. 일부 실시예에 따라, 배리어 재설정은 재설정이 적용되는 배리어 인스턴스에 관련되는 모든 작업항목을 가로질러 동기화 포인트를 야기하는 자체-동기화 명령어로서 구현된다. 다른 실시예에 있어서, 사용자는 재설정 명령어에서의 동기화를 보장하도록 하나 이상의 동기화 포인트를 포함할 수 있다. 도 4a는 일 실시예에 따라 그 그룹 사이즈에 독립적으로 정의된 사이즈를 갖는 배리어의 사용을 의사코드로 예시하고 있다. 예컨대, 배리어 b는, 배리어가 호출되어 나오는 작업그룹에 얼마나 많은 작업항목이 있는지에 무관하게, 원래 동기화 그룹 사이즈 16으로 생성될 수 있다. 16의 정의된 사이즈로 배리어 b를 생성하는 것은 배리어 b 상에서 대기 명령어를 호출하는 어느 프로세스가 배리어 b 상에서 동기화하는 그룹 내 작업항목 중 어느 15개와 동기화 가능하게 한다.

[0035] 정의된 사이즈를 갖는 배리어는 특정 수의 작업항목만이 소정 조건을 만족할 것임이 알려져 있는 애플리케이션에 대해 생성될 수 있다. 이러한 방식으로, 도 4a에서 도시된 바와 같이, 조건을 만족하는 작업항목은 배리어 b 상에서 동기화할 수 있는 한편, 다른 것들은 "else" 경로를 취한다. 배리어가 조건을 충족하는 정의된 수의 작업항목을 대기할 뿐이기 때문에, 데드로크 조건은 일어날지 않을 것이고, 조건을 만족하지 않는 작업항목은 스킵 명령어를 발행하도록 요구되지 않는다.

[0036] 도 4b의 예시적 예에 있어서, 배리어는 2개의 작업항목 그룹을 동기화하도록 사용된다. 하나의 그룹은 특정 조건을 만족한다. 그 조건을 만족하지 않는 다른 그룹은 배리어 b1 및 b2를 사용하여 별개로 동기화될 수 있다. b1 동기화 그룹 또는 b2 동기화 그룹에 속하는 작업항목은 그것이 속하지 않는 그룹에 스킵 명령어를 발행할 수 있다.

[0037] 도 4c의 예시적 예에 있어서, 의사코드로 계층적 배리어가 사용된다. 제1 특정 조건을 만족하는 작업항목은 배리어 b1 상에서 대기 명령어 상에서 동기화하고, 다른 작업항목은 배리어 b1를 스킵한다. 제1 특정 조건을 만족하는 그들 작업항목은 또 다른 배리어 b2와 마주친다. 제2 배리어 b2와 마주치는 작업항목 중 제2 조건을 만족하는 것들은 b2 상에서 대기한다 - 다른 것들은 b2를 스킵한다. 의사코드로 나타낸 바와 같이 배리어 b2는 배리어 b1의 현재 상태에 기반하여 생성된다.

[0038] 더 구체적으로, 배리어 b2의 사이즈(즉, b2가 동기화를 시행하는 작업항목의 수)는 b1 상에서 스킵 명령어를 발행하지 않은 작업항목의 수로 표현된다. 도 4c에 있어서, 이러한 수는 제1 조건을 만족한 작업항목의 수이다. 다른 실시예는 b1이 스킵될 때 배리어 업데이트 b2의 기저 구현을 요구할 수 있다. 더욱, b2의 업데이트에서 일부 상황에서 일어날 수 있는 레이스 조건을 회피하기 위해, b1 스킵 동작은 b1 스킵 동작 전에 동기화 포인트를 가짐으로써 보호될 수 있다.

[0039] 도 5는, 본 발명의 일 실시예에 따라, 수개의 작업항목(T1, T2, T3, T4)의 시간에 걸친 예시적 흐름(500)을 예

시하고 있다. 도시된 바와 같이, T1, T2, T3 및 T4는 동시다발적으로 또는 실질적으로 동시다발적으로 시작한다. 각각의 작업항목에서의 제1 배리어 대기 명령어는 그것이 동기화 포인트(501)에서 동기화하도록 야기한다. 동기화 포인트(501)에서의 동기화는 작업항목(T1, T2, T3, T4)이 (501)에서 도착할 것들 중 마지막 작업항목을 대기하고 그 후 동시다발적으로 또는 실질적으로 동시다발적으로 실행을 재개하는 것과 관련된다.

[0040] 포인트(502)에서, 작업항목(T4)은 배리어 도착 명령어를 발행한다. 배리어 도착 명령어는 T4 상에서 대기하지 않도록 배리어의 다음 인스턴스에 통지하고, T4는 배리어의 제2 인스턴스에서 동기화해야함이 없이 진행한다.

[0041] 동기화 포인트(503)에서, 작업항목(T1, T2, T3)은 명령어 스트림 내 제2 배리어 대기 명령어에 기반하여, 배리어의 제2 인스턴스 상에서 동기화한다. 모든 작업항목(T1, T2, T3)이 동기화 포인트(503)에서 도착할 때, 그것들은 동시다발적으로 또는 실질적으로 동시다발적으로 실행을 재개한다.

[0042] 포인트(504)에서, T1은 배리어 스킵 명령어를 발행한다. 배리어 스킵 명령어는 T1 상에서 대기하지 않도록 배리어의 어느 후속 인스턴스에라도 통지하고, T1은 배리어의 후속 인스턴스에서 동기화해야함이 없이 진행한다.

[0043] 동기화 포인트(505)에서는, 작업항목(T2, T3, T4)이 동기화한다. T4가 ((502)에서) 이전에 배리어 도착 명령어를 발행하였지만, 이러한 이전의 발행은 다음의 일어나는 배리어로부터(즉, 동기화 포인트(503)에서) T4를 면제하였을 뿐임을 주목하라. 동기화 포인트(505)에서의 동기화 후에, T2, T3, 및 T4는 진행하여 동기화 포인트(506)에서 또다시 동기화한다. 동기화 포인트(506)에서 동기화될 때, T2 및 T3는 각각 그 각자의 명령어 스트림에서 배리어 대기 명령어의 4개의 인스턴스와 마주쳤을 것이다. 이때, T4는 3개의 인스턴스와 마주칠 뿐이었을 것이다. 동기화 포인트(505, 506)를 선행하는 (504)에서 T1이 배리어 스킵 명령어를 발행했기 때문에 동기화 포인트(505, 506)에서의 배리어는 T1에 대해 블로킹하지 않는다.

[0044] (506)의 일어남 후에, 적시에 포인트(507)에서, T1은 배리어 재설정 명령어를 발행한다. 이것은 배리어를 그 원래 구성으로 재설정한다. 원래 배리어는 모든 4개의 작업항목(T1, T2, T3, T4) 상에서 동기화하도록 구성되었다. 재설정은 동기화 포인트(507)에서 T1이 T2, T3, 및 T4와 동기화하게 야기한다. 포인트(507)에서의 동기화는 재설정과 연관된 사용자-특정 동기화 명령어에 의해 또는 자체-동기화 명령어로서 재설정의 구현에 의해 달성될 수 있다.

[0045] 포인트(507)에서의 동기화에 뒤따라, T1-T4는 각각의 작업항목에서의 대기 명령어에 기반하여 동기화 포인트(508)에서 동기화하도록 진행한다.

[0046] 도 6은 일 실시예에 따른 작업항목 동기화를 위한 예시적 방법(600)을 예시하고 있다.

[0047] 동작(602)에서, 작업항목 그룹이 시작된다. 작업항목은 똑같은 코드의 복수의 작업항목일 수 있다. 똑같은 코드의 각자의 작업항목에서 시행되는 실제의 명령어 시퀀스는, 조건 평가 등에 의존하여, 동일할 수도 다를 수도 있다. 또 다른 실시예에 따라, 작업항목은 모두 똑같은 코드는 아니고, 서로 하나 이상의 동기화 포인트를 공유하는 어느 작업항목으로 이루어질 수 있다.

[0048] 복수의 작업항목은 동시다발적으로 또는 비-동시다발적으로 시작될 수 있다. 작업항목은 CPU에서, GPU에서, 2개 이상의 GPU에서, 2개 이상의 CPU 코어에서, 또는 하나 이상의 GPU와 하나 이상의 CPU 코어의 어떠한 조합에서라도 실행될 수 있다. 일 실시예에 따라, 복수의 작업항목은 GPU의 하나의 프로세싱 엘리먼트 상에서 실행하는 작업그룹이다.

[0049] 동작(604)에서, 배리어 b가 생성된다. 배리어 b는 배리어 b를 선언하는 작업항목 상에서 명령어를 실행함으로써 생성 또는 인스턴스화될 수 있다. 일 실시예에 의하면, 시스템이 배리어 b의 선언의 제1 인스턴스와 마주칠 때, 배리어의 인스턴스가 메모리에 생성된다. 후속하여 배리어 b를 선언하는 작업항목이 이미 생성된 배리어 b로의 레퍼런스를 수신한다. 배리어의 기저 구현은 각자의 프레임워크 및/또는 시스템에서 다를 수 있다. 카운팅 세마포어(counting semaphore)는 상기 시맨틱스를 갖는 배리어가 구현될 수 있는 예시적 메커니즘이다.

[0050] 메모리 내 배리어 b 오브젝트의 생성은 동적 메모리 및/또는 하드웨어 내 하나 이상의 메모리 로케이션 및/또는 레지스터를 초기화하는 것을 포함한다. 예컨대, 배리어 b와 관련하여, 수개의 카운트는 아래에 설명되는 바와 같이 유지되도록 요구된다. 배리어 오브젝트와 더불어 모든 카운트는 그들 메모리 로케이션으로의 쓰기 및 읽기에 있어서 적절한 동시다발성 제어 메커니즘으로 동적 메모리에 유지될 수 있다. 또 다른 실시예에 있어서, 배리어 b에 대응하는 오브젝트가 동적 메모리에서 인스턴스화되는 반면, 대응하는 카운트는 특정 하드웨어 레지스터에 유지된다.

[0051] 동작(606)에서는, 배리어 b를 초기화하는데 요구되는 다양한 카운트가 결정되고 그들 초기값이 설정된다. 방문

하는 작업항목의 수("방문 카운트")는 배리어 b에 도달한 작업항목의 수를 정의한다. 작업항목은 그것이 배리어 대기 명령어 또는 등가물을 발행할 때 배리어에 "도달"하였다. 방문 카운트는 0으로 초기화될 수 있다. 배리어 b 상에서 배리어 스킵 명령어를 실행한 작업항목의 수는 스킵된 카운트("스킵 카운트")로 추적될 수 있다. 배리어 릴리스 임계값("릴리스 임계값")은 배리어가 계속 대기하고 있는 작업항목의 수이다. 일 실시예에 따라, 배리어 b는 동작(602)에서 시작된 그룹 내 작업항목의 수와 동일한 릴리스 임계값으로 초기화된다. 또 다른 실시예에 의하면, 배리어 b는 동작(602)에서 시작된 그룹의 사이즈와 무관하게 정의된 사이즈로 (동작(604)에서) 생성된다.

[0052] 동작(608)에서, 명령어 흐름으로부터 어느 수의 다른 명령어를 실행한 후, 작업항목 x는 동기화 명령어에 도착한다. 동기화 명령어는, 국한되는 것은 아니지만, 배리어 대기, 배리어 도착, 배리어 스킵, 및 배리어 재설정 중 하나일 수 있다.

[0053] 동작(610)에 있어서, 동기화 명령어가 배리어 대기 명령어인지 결정이 이루어지고, 그러하다면, 방법(600)은 동작(612)으로 진행한다.

[0054] 동작(612)에서, 방문 카운트가 업데이트된다. 일 실시예에 따라, 방문 카운트는 작업항목 x이 배리어에 도달하였음을 나타내도록 1만큼씩 증분된다.

[0055] 동작(614)에서, 업데이트된 방문 카운트 및 스킵 카운트의 합이 릴리스 임계값과 비교된다. 합이 릴리스 임계값보다 크거나 같으면, 그때 작업항목 x은 배리어에 도착할 마지막 작업항목이고, 배리어는 동작(618)에서 릴리스된다. 배리어를 릴리스하는 것은, 일 실시예에 따라, 하나 이상의 카운트 값이 재설정되고 그리고 블로킹된 작업항목이 실행을 재개하도록 야기한다.

[0056] 배리어의 릴리스가 동작(620)에서 방문 카운트의 재설정 및 동작(622)에서 배리어 상에서 블로킹된 모든 작업항목의 재개를 야기한다. 동작(620)에서, 일 실시예에 따라, 방문 카운트가 0으로 설정된다. 방문 카운트를 재설정하는 것은 앞서 일어난 배리어 도착 명령어의 어떠한 효과도 지우는 것임을 주목하라. 그렇지만, 방문 카운트를 재설정하는 것은 앞서 일어난 어떠한 배리어 스킵 명령어의 효과도 지우지 않는다. 블록으로부터 배리어 릴리스의 이벤트에서 행해지는 바와 같이, 방문 카운트만의 재설정은 배리어 상에서 "일부-재설정" 동작으로 생각될 수 있다. 본 명세서에서 사용되는 바와 같이 용어 "일부-재설정"은 스킵 명령어를 발행하지 않은 작업항목에 적용하는 배리어의 일부가 재설정됨을 전한다.

[0057] 동작(622)에서, 모든 블로킹된 작업항목이 실행을 재개한다. 일 실시예에 따라, 블로킹된 작업항목은 세마포어 상에서 대기하고 있을 수 있고, 세마포어는 그 블로킹된 작업항목이 실행을 재개할 수 있도록 재설정된다. 예컨대, 하드웨어 또는 소프트웨어로 구현된 카운팅 세마포어는 작업항목을 블로킹하도록 사용될 수 있다. 동작(622)을 완료한 후에, 방법(600)은 동작(608)으로 진행한다.

[0058] 동작(614)에서, 방문 카운트와 스킵 카운트의 합이 릴리스 임계값과 같지 않다고 결정되면, 그때 동작(616)에서 작업항목 x이 블로킹된다. 작업항목 x의 블로킹은, 일 실시예에 따라, 작업항목이 세마포어 상에서 대기하도록 야기함으로써 수행될 수 있다. 후속하여 동작(618-622)이 배리어에 대해 일어날 때 작업항목 x이 릴리스될 것이다. 동작(619)은, 예컨대 동작(618-622)에 의해, 배리어의 릴리스시 작업항목 x의 실행의 계속을 표현한다.

[0059] 동작(610)에서 명령어가 배리어 대기 명령어가 아니라고 결정되면, 그때 동작(624)에서는, 명령어가 배리어 도착인지 결정된다. 예이면, 그때 동작(626)에서 방문 카운트가 업데이트된다. 일 실시예에 따라, 방문 카운트는 작업항목 x이 배리어에 도달하였음을 나타내도록 1만큼씩 증분된다. 방문 카운트를 업데이트한 후에, 동작(628)에서, 작업항목 x는 명령어 스트림의 그 실행을 계속한다. 후속하여, 다음 동기화 명령어가 마주쳐질 때 프로세싱은 동작(608)으로 진행한다.

[0060] 동작(624)에서 명령어가 배리어 도착 명령어가 아니라고 결정되면, 그때 동작(630)에서, 명령어가 배리어 스킵인지 결정된다. 예이면, 그때 동작(632)에서, 스킵 카운트가 업데이트된다. 일 실시예에 따라, 스킵 카운트는 1만큼씩 증분된다. 그때 동작(636)에서, 작업항목 x는 실행을 계속하고 다음 동기화 명령어가 마주쳐질 때 동작(608)으로 진행한다.

[0061] 동작(630)에서 명령어가 배리어 스킵 명령어가 아니라고 결정되면, 그때 동작(638)에서 명령어가 배리어 재설정인지 결정된다. 예이면, 그때 동작(640)에서 배리어 b가 재설정된다. 일 실시예에 따라, 배리어 재설정 명령어에 응답하여 배리어를 재설정하는 것은 방문 카운트 및 스킵 카운트를 0으로 설정하는 것을 포함한다. 그리하여, 배리어 재설정이 배리어 b 상에서 실행된 후에, 배리어 b는 그 원래 상태로 재설정된다. 일부 상황에 있어서, 그룹 내 모든 작업항목이 배리어에서 동기화되는 것은 아니면 레이스 조건이 일어날 수 있다. 그래서,

일부 실시예에 있어서, 배리어 재설정 명령어는 자체-동기화 명령어로서 구현될 수 있고 모든 작업항목을 가로지르는 동기화 포인트를 야기할 수 있다. 다른 실시예에 있어서, 사용자는 재설정 명령어에서의 동기화를 보장하기 위한 하나 이상의 동기화 포인트를 포함할 수 있다. 동작(640)에 뒤따라, 프로세싱은 다음의 동기화 명령어가 명령어 스트림에서 마주쳐질 때 동작(608)으로 계속할 수 있다.

[0062] 단계(638)에서 명령어가 배리어 재설정 명령어가 아니라고 결정되면, 그때 동작(639)에서 작업항목 x가 실행을 계속할 수 있다. 동작(639)을 뒤따라, 프로세싱은 다음 동기화 명령어가 명령어 스트림에서 마주쳐질 때 동작(608)으로 계속할 수 있다.

[0063] 또 다른 실시예에 따라, 배리어에 대해 별개의 스킵 카운트 및 릴리스 임계값을 유지하는 대신에, 릴리스 임계값만이 유지될 수 있다. 예컨대, 릴리스 임계값은 작업항목이 배리어 스킵 명령어를 실행하였음을 반영하도록 감분될 수 있다. 그때, 배리어 재설정 명령어가 발행될 때, 카운트의 재설정은 릴리스 임계값을 그 원래 치수로 재설정하는 것을 포함할 것이다. 실제로는, 이러한 접근법에 따라, 배리어 스킵을 실행하는 작업항목의 수가 동기화 그룹을 떠나는 것으로 생각될 수 있다.

[0064] 도 7은 일 실시예에 따른 작업항목 동기화를 위한 시스템을 예시하는 블록 선도이다. 도 7에 있어서, 일례의 이종 컴퓨팅 시스템(700)은 CPU(701)와 같은 하나 이상의 CPU, 및 GPU(702)와 같은 하나 이상의 GPU를 포함할 수 있다. 이종 컴퓨팅 시스템(700)은 또한 시스템 메모리(703), 영속 저장 디바이스(704), 시스템 버스(705), 입/출력 디바이스(706) 및 배리어 동기화기(709)를 포함할 수 있다.

[0065] CPU(701)는 상업적으로 이용가능한 제어 프로세서 또는 커스텀 제어 프로세서를 포함할 수 있다. CPU(701)는, 예컨대, 이종 컴퓨팅 시스템(700)의 동작을 제어하는 제어 로직을 실행한다. CPU(701)는 2개의 CPU 코어(741, 742)를 갖는 멀티-코어 CPU와 같은 멀티-코어 CPU일 수 있다. CPU(701)는, 어떠한 제어 회로에라도 부가하여, 각자 CPU 코어(741, 742)의 CPU 캐시 메모리(743, 744)를 포함한다. CPU 캐시 메모리(743, 744)는, 각자, CPU 코어(741, 742) 상에서 애플리케이션의 실행 동안 명령어 및/또는 파라미터 값을 임시로 저장하도록 사용될 수 있다.

[0066] 예컨대, CPU 캐시 메모리(743)는 CPU 코어(741) 상에서 제어 로직 명령어의 실행 동안 시스템 메모리(703)로부터 하나 이상의 제어 로직 명령어, 변수 값, 또는 상수 파라미터 값을 임시로 저장하도록 사용될 수 있다. CPU(701)는 또한 전문 벡터 명령어 프로세싱 유닛을 포함할 수 있다. 예컨대, CPU 코어(742)는 벡터화된 명령어를 효율적으로 프로세싱할 수 있는 스트리밍 SIMD 익스텐션(SSE) 유닛을 포함할 수 있다. 당업자는 CPU(701)가 선택된 예에서보다 더 많거나 더 적은 CPU 코어를 포함할 수 있고 또한 캐시 메모리를 갖지 않든지 또는 더 복잡한 캐시 메모리 계층을 갖든지 할 수 있음을 이해할 것이다.

[0067] GPU(702)는 상업적으로 이용가능한 그래픽 프로세서 또는 커스텀 설계된 그래픽 프로세서를 포함할 수 있다. 예컨대, GPU(702)는 선택된 기능을 위한 전문 코드를 실행할 수 있다. 일반적으로, GPU(702)는 디스플레이 상에서 이미지의 그래픽 파이프라인 계산 및 렌더링과 같은 그래픽 기능을 실행하도록 사용될 수 있다.

[0068] GPU(702)는 GPU 글로벌 캐시 메모리(710) 및 하나 이상의 컴퓨팅 유닛(712, 713)을 포함한다. 그래픽 메모리(707)는 GPU(702)에 포함되거나 그에 결합될 수 있다. 각각의 컴퓨팅 유닛(712, 713)은 각자 GPU 로컬 메모리(714, 715)와 연관된다. 각각의 컴퓨팅 유닛은 하나 이상의 GPU 프로세싱 엘리먼트(PE)를 포함한다. 예컨대, 컴퓨팅 유닛(712)은 GPU 프로세싱 엘리먼트(721, 722)를 포함하고, 컴퓨팅 유닛(713)은 GPU PE(723, 724)를 포함한다.

[0069] 각각의 GPU 프로세싱 엘리먼트(721, 722, 723, 724)는 적어도 하나의 전용 메모리(PM)(731, 732, 733, 734)와 각자 연관되어 있다. 각각의 GPU PE는 스칼라 및 벡터 부동-소수점 유닛 중 하나 이상을 포함할 수 있다. GPU PE는 또한 역-제곱근 유닛 및 사인/코사인 유닛과 같은 특수 목적 유닛을 포함할 수 있다. GPU 글로벌 캐시 메모리(710)는 시스템 메모리(703)와 같은 시스템 메모리, 및/또는 그래픽 메모리(707)와 같은 그래픽 메모리에 결합될 수 있다.

[0070] 시스템 메모리(703)는 동적 램(DRAM)과 같은 적어도 하나의 비-영속 메모리를 포함할 수 있다. 시스템 메모리(703)는 애플리케이션 또는 다른 프로세싱 로직의 일부의 실행 동안 프로세싱 로직 명령어, 상수 값 및 변수 값을 저장할 수 있다. 예컨대, 배리어 동기화기(709)의 제어 로직 및/또는 다른 프로세싱 로직은 CPU(701)에 의한 배리어 동기화기(709)의 실행 동안 시스템 메모리(703) 내에 거주할 수 있다. 본 명세서에서 사용되는 바와 같이, 용어 "프로세싱 로직"은 제어 흐름 명령어, 계산을 수행하기 위한 명령어, 및 자원으로의 연관 액세스를 위한 명령어를 지칭한다.

[0071] 영속 메모리(704)는 자기 디스크, 광학 디스크 또는 플래시 메모리와 같이 디지털 데이터를 저장할 수 있는 하나 이상의 저장 디바이스를 포함한다. 영속 메모리(704)는 예컨대 배리어 동기화기(709)의 명령어 로직의 적어도 일부를 저장할 수 있다. 이종 컴퓨팅 시스템(700)의 시동시에, 운영체제 및 다른 애플리케이션 소프트웨어는 영속 저장소(704)로부터 시스템 메모리(703) 내로 로딩될 수 있다.

[0072] 시스템 버스(705)는 주변 컴포넌트 인터커넥트(PCI) 버스, 산업 표준 아키텍처(ISA) 버스, 또는 그러한 디바이스를 포함할 수 있다. 시스템 버스(705)는 또한 이종 컴퓨팅 시스템(700)의 컴포넌트를 포함하는 컴포넌트를 결합하는 기능성과 함께 랜(LAN)과 같은 네트워크를 포함할 수 있다.

[0073] 입/출력 인터페이스(706)는 키보드, 마우스, 디스플레이 및/또는 터치 스크린과 같은 사용자 입/출력 디바이스를 접속시키는 하나 이상의 인터페이스를 포함한다. 예컨대, 사용자 입력은 키보드 및 마우스 접속 사용자 인터페이스(706)를 통해 이종 컴퓨팅 시스템(700)에 제공될 수 있다. 이종 컴퓨팅 시스템(700)의 출력은 사용자 인터페이스(706)를 통해 디스플레이하도록 출력될 수 있다.

[0074] 그래픽 메모리(707)는 시스템 버스(705)에 그리고 GPU(702)에 결합되어 있다. 그래픽 메모리(707)는, 일반적으로는, GPU에 의한 고속 액세스를 위해 시스템 메모리(703)로부터 전송된 데이터를 저장하도록 사용된다. 예컨대, GPU(702)와 그래픽 메모리(707) 사이의 인터페이스는 시스템 버스 인터페이스(705)보다 수배 더 빠를 수 있다.

[0075] 배리어 동기화기(709)는 GPU(702) 및 CPU(701) 중 어느 하나 또는 둘 다에서 함수 및 프로세싱 로직을 동기화하는 로직을 포함한다. 배리어 동기화기(709)는 컴퓨터 내 프로세서 그룹을 가로질러 전반적으로, 각각의 개개 프로세서에서, 그리고/또는 프로세서의 각각의 프로세싱 엘리먼트 내에서 작업항목을 동기화하도록 구성될 수 있다. 배리어 동기화기(709)는 도 8과 관련하여 아래에 더 설명된다. 당업자는 배리어 동기화기가 소프트웨어, 펌웨어, 하드웨어 또는 그 조합을 사용하여 구현될 수 있음을 이해할 것이다. 소프트웨어로 구현될 때, 예컨대, 배리어 동기화기(709)는 컴파일되고 실행할 때 시스템 메모리(703)에 거주하는 C 또는 OpenCL로 쓰인 컴퓨터 프로그램일 수 있다. 소스 코드 형태 및/또는 컴파일된 실행가능한 형태로, 배리어 동기화기(709)는 영속 메모리(704)에 저장될 수 있다. 일 실시예에 있어서, 배리어 동기화기(709)의 기능성의 일부 또는 전부는 마스크워크/포토마스크의 발생을 통해 궁극적으로 제조 프로세스를 구성하는 것을 가능하게 하여 본 명세서에서 설명된 본 발명의 태양을 구체화하는 하드웨어 디바이스를 발생시키도록 베릴로그(Verilog), RTL, 넷리스트와 같은 하드웨어 기술 언어로 특정된다.

[0076] 당업자는 이종 컴퓨팅 시스템(700)이 도 7에 도시된 컴포넌트를 더 많이 또는 더 적게 포함할 수 있음을 이해할 것이다. 예컨대, 이종 컴퓨팅 시스템(700)은 하나 이상의 네트워크 인터페이스, 및 또는 OpenCL 프레임워크와 같은 소프트웨어 애플리케이션을 포함할 수 있다.

[0077] 도 8은 일 실시예에 따라 배리어 동기화기(800)의 예시이다. 배리어 동기화기(800)는 작업항목 블로킹 모듈(802), 배리어 릴리스 모듈(804), 배리어 작업항목 그룹 모듈(806), 배리어 스킵 모듈(808), 및 배리어 재설정 모듈(810)을 포함한다. 더욱, 배리어 동기화기(800)는 배리어 레지스터(812)를 포함할 수 있다. 일 실시예에 의하면, 배리어 동기화기(800)는 배리어 동기화기(709)에 포함된다.

[0078] 작업항목 블로킹 모듈(802)은 배리어 상에서 하나 이상의 작업항목을 블로킹하도록 동작한다. 배리어는 세마포어(예컨대, 카운팅 세마포어) 및 레지스터를 사용하여 구현될 수 있다. 작업항목 블로킹은 블로킹된 작업항목이 세마포어 상에서 대기하게 야기함으로써 구현될 수 있다. 세마포어는 하드웨어 또는 소프트웨어로 구현될 수 있다. 작업항목은 배리어 대기 명령어가 마주쳐질 때 블로킹될 수 있다. 작업항목 블로킹 모듈은, 예컨대, 방법(600)의 동작(616)을 포함하는 동작을 구현하기 위한 프로세싱 로직을 포함할 수 있다.

[0079] 배리어 릴리스 모듈(804)은 충분한 수의 작업항목이 그에 도달하였을 때 배리어를 릴리스하도록 동작한다. 위에서 설명된 바와 같이, 배리어는 세마포어를 사용하여 구현될 수 있고, 배리어를 릴리스하는 것은 세마포어를 릴리스하는 것을 포함할 수 있다. 작업항목은 배리어 대기 명령어가 마주쳐지고 그것이 배리어에 도달할 작업항목의 수에 대한 요건을 완성할 마지막 작업항목이라고 판명될 때 릴리스될 수 있다. 배리어 릴리스 모듈은, 예컨대, 방법(600)의 동작(618-622)을 포함하는 동작을 구현하기 위한 프로세싱 로직을 포함할 수 있다.

[0080] 배리어 작업항목 그룹 모듈(806)은 다양한 실행 작업항목 중 동기화 그룹을 추적하도록 동작한다. 배리어 작업항목 그룹 모듈(806)은 또한 그룹 메이크업에 따라 배리어를 생성 및 초기화하도록 동작할 수 있다. 일 실시예에 의하면, 배리어 릴리스 모듈(806)은 또한, 예컨대, 방법(600)의 동작(602-606)을 포함하는 동작을 구현하기 위한 프로세싱 로직을 포함할 수 있다.

[0081] 배리어 스킵 모듈(808)은 배리어 스킵 명령어를 구현하도록 동작한다. 예컨대, 배리어 스킵 명령어는 대응하는 배리어에 대한 스킵 카운트가 업데이트되도록 야기할 것이다. 일 실시예에 의하면, 배리어 스킵 모듈(808)은 방법(600)의 동작(630-636)을 포함하는 동작을 구현하기 위한 프로세싱 로직을 포함할 수 있다.

[0082] 배리어 재설정 모듈(810)은 배리어 재설정 명령어를 구현하도록 동작한다. 예컨대, 배리어 재설정 명령어는 배리어와 관련된 방문 카운트 및 스킵 카운트가 재설정되고, 그리고/또는 릴리스 임계값이 조절되도록 야기할 것이다. 배리어 재설정 모듈(810)은, 예컨대, 방법(600)의 동작(638-640)을 포함하는 동작을 구현하기 위한 프로세싱 로직을 포함할 수 있다.

[0083] 배리어 레지스터(812)는 배리어와 관련된 하드웨어 및/또는 소프트웨어 레지스터를 포함한다. 배리어 레지스터(812)는 복수의 레지스터 및/또는 메모리 로케이션을 포함하는 배리어 레코드(814)를 포함할 수 있다. 예시의 배리어 레코드(814)는 배리어 식별자(822), 로크(824), 블로킹된 작업항목 카운트(826), 도착한 작업항목 카운트(828), 스킵된 작업항목 카운트(830) 및 임계값(832)을 포함한다. 배리어 식별자(822)는 배리어의 메모리 로케이션 또는 레지스터를 고유하게 식별하기 위한 포인터, 인덱스 또는 다른 식별자일 수 있다.

[0084] 로크(824)는 프로세스가 블로킹되는 세마포어 또는 다른 엔티티로의 포인터 또는 레퍼런스일 수 있다. 블로킹된 작업항목 카운트(826)는 배리어 상에서 블로킹되는 작업항목의 수이다. 도착한 작업항목 카운트(828)는 배리어 도착 명령어를 발행한 작업항목의 수이다. 스킵된 작업항목 카운트(830)는 배리어 스킵 명령어를 발행한 작업항목의 수이다. 임계값(832)은 릴리스 임계값, 또는 동기화 그룹의 그룹 사이즈이다.

[0085] 결론

[0086] 개요 및 요약 절은 발명자(들)에 의해 고려되는 바와 같은 본 발명의 하나 이상의 그렇지만 모두는 아닌 예시적 실시예를 제시할 수 있고, 그리하여, 본 발명 및 첨부 청구범위를 어떤 식으로도 한정하려는 의도가 아니다.

[0087] 본 발명은 특정 기능 및 그 관계의 구현을 예시하는 기능적 빌딩 블록의 도움으로 위에서 설명되었다. 이들 기능적 빌딩 블록의 경계는 설명의 편의를 위해 본 명세서에서는 임의로 정의되었다. 특정 기능 및 그 관계가 적절하게 수행되는 한 대체 경계가 정의될 수 있다.

[0088] 특정 실시예의 상기 설명은 본 발명의 일반적 본질을 아주 충분히 드러낼 것이어서 타인은, 당업자의 지식을 적용함으로써, 본 발명의 일반적 개념으로부터 벗어남이 없이 과도한 실험 없이 그러한 특정 실시예를 다양한 애플리케이션을 위해 쉽게 수정 및/또는 적응시킬 수 있다. 그래서, 그러한 적응 및 수정은, 본 명세서에서 제시된 교시 및 가이드에 기초하여, 개시된 실시예의 균등물의 의미 및 범위 내에 있는 것으로 의도된다. 본 명세서에서의 어법 또는 용어는 한정이 아닌 설명의 목적을 위한 것으로 이해되는 것이어서, 본 명세서의 용어 또는 어법은 교시 및 가이드에 비추어 당업자에 의해 해석되는 것이다.

[0089] 본 발명의 폭 및 범위는 위 설명의 예시적 실시예의 어느 것에 의해서도 한정되어서는 안 되며, 오직 이하의 청구범위 및 그 균등물에 의해서만 정의되어야 한다.

도면

도면1a

```
Kernel loadKernel(global float *data, local float *sharedData)
{
  // get workitem 0 in the group load the group's constant from data
  // and place it in the shared array
  if( get_locaLid(0) == 0) {
    sharedData[0] = data[get_group_id(0)];
  }
  // Ensure all workitems wait for the operation to complete
  barrier(CLK_GLOBAL_MEM_FENCE);
  ... // do something using the shared value
}
```

도면1b

```
float loadFunction (global float *data, local float *sharedData)
{
  // get workitem 0 in the group load the group's constant from data
  // and place it in the shared array
  if( get_locaLid(0) == 0) {
    sharedData[0] = data[get_group_id(0)];
  }

// Ensure all workitems wait for the operation to complete
  barrier(CLK_GLOBAL_MEM_FENCE);
  return sharedData[0];
}
```

도면1c

```
kernel theKernel(global  float *data, local float *sharedData)
{
  float val= loadFunction(data, sharedData);
  // do something using val
}
```

도면1d

```
kernel theKernel(global float *data, local float *sharedData)
{
  If ( someCondition ) {
    float val= loadFunction(data, sharedData);
    // do something using val
  } else {
    // we know we don't need to do that load here
  }
}
```

도면2a

```
kernel theKernel(global float *data, local float *sharedData)
{
  barrier b;
  while( someCondition ) {
    b. wait();
    // do something
  }
  b.skip();
}
```

도면2b

```
kernel theKernel(global float *data, local float *sharedData)
{
  barrier b;
  if( someCondition ) {
    // pass the barrier object down to the called function
    float val= loadFunction(data, sharedData, b);
    // do something using val
  } else {
    // Instead of arrive, skip here. That way loadFunction can use the barrier
    //  more than once while maintaining correct behavior. If arrive were used
    // waiting a second time in loadFunction would
    // cause a wait on this workitem too, which is undesired behavior.
    b.skip();
    // we know we don't need to do that load here
  }
}

float loadFunction (global float *data, local float *sharedData, barrier b)
{
  // get workitem 0 in the group load the group's constant from data
  // and place it in the shared array
  if( get_localId(0) == 0) {
    sharedData[0] = data[get_group_id(0)];
  }
  // Wait specifically on the barrier we passed:
  b.wait();
  // Communicate in some way
  // Wait a second time
  b.wait();
  return sharedData[0];
}
```

도면3

```
kernel theKernel(global float *data, local float *sharedData)
{
   barrier b;
   if( someCondition ) {
      // pass the barrier object down to the called function
      float val= loadFunction(data, sharedData, b);
      // do something using val
   } else {
      // Instead of arrive, skip here. That way loadFunction can use the barrier more than once while
      // maintaining correct behavior. If arrive were used waiting a second time in load Function would
      // cause a wait on this workitem too, which is undesired behavior.
      b.skip();
      // we know we don't need to do that load here
    }
    b.reset();
    //Do more
    b.wait();
    // Do more post synchronization of *all* workitems
}
```

도면4a

```
kernel theKernel(global float *data, local float *sharedData)
{
   // A barrier for 16 workitems
   barrier b(16);
   If ( someCondition that applies only to 16 workitems in the set) {
      // pass the barrier object down to the called function
      float val= loadFunction(data, sharedData, b);
      // do something using val
   } else {
      // We know that the first condition dealt with only 16 workitems. The rest will
follow this path
      // and hence do not need to notify the barrier of their wish to skip.
   }
}
```

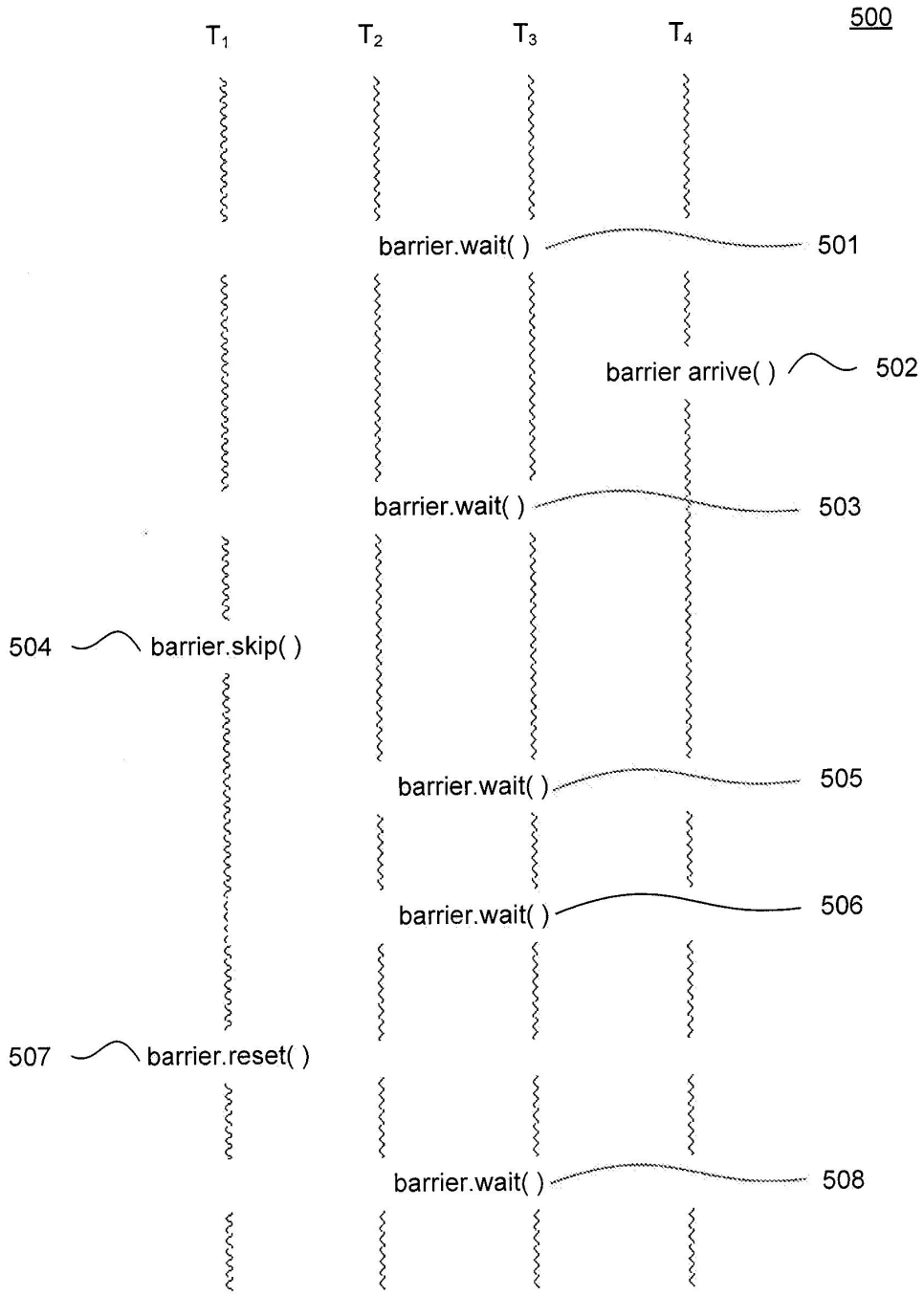
도면4b

```
kernel theKernel(global float *data, local float *sharedData)
{
  // create two barriers to seperately synchronize two groups of workitems
  barrier b1, b2;
  If ( someCondition that applies only to b1 workitems) {
        // do something
        b2.skip();
        b1.wait();
        // do something
  } else { // else applies only to b2 workitems
        // do something
        b1.skip();
        b2.wait();
        // do something
  }
}
```

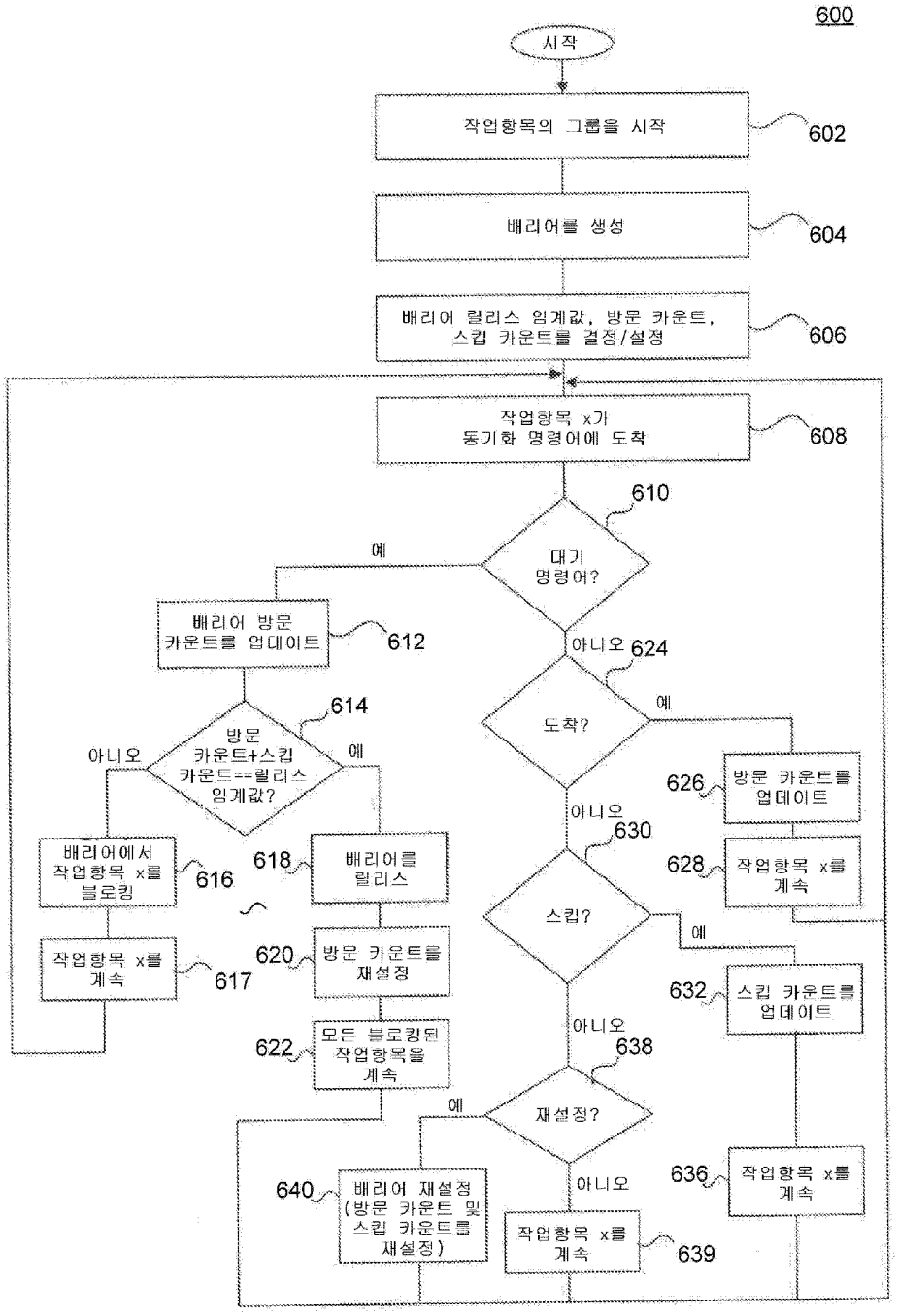
도면4c

```
kernel theKernel(global float *data, local float *sharedData)
{
  // create outer barrier b1
  barrier b1;
  If ( someCondition ) {
        // do something
        b1.wait();
        // do something
        // create inner barrier b2
        barrier b2(b1);
        if ( someCondition2 ) {
              b2.wait();
        }
        b2.skip();
        // do something
  }
  // do something
  b1.skip();
  // do something
}
```

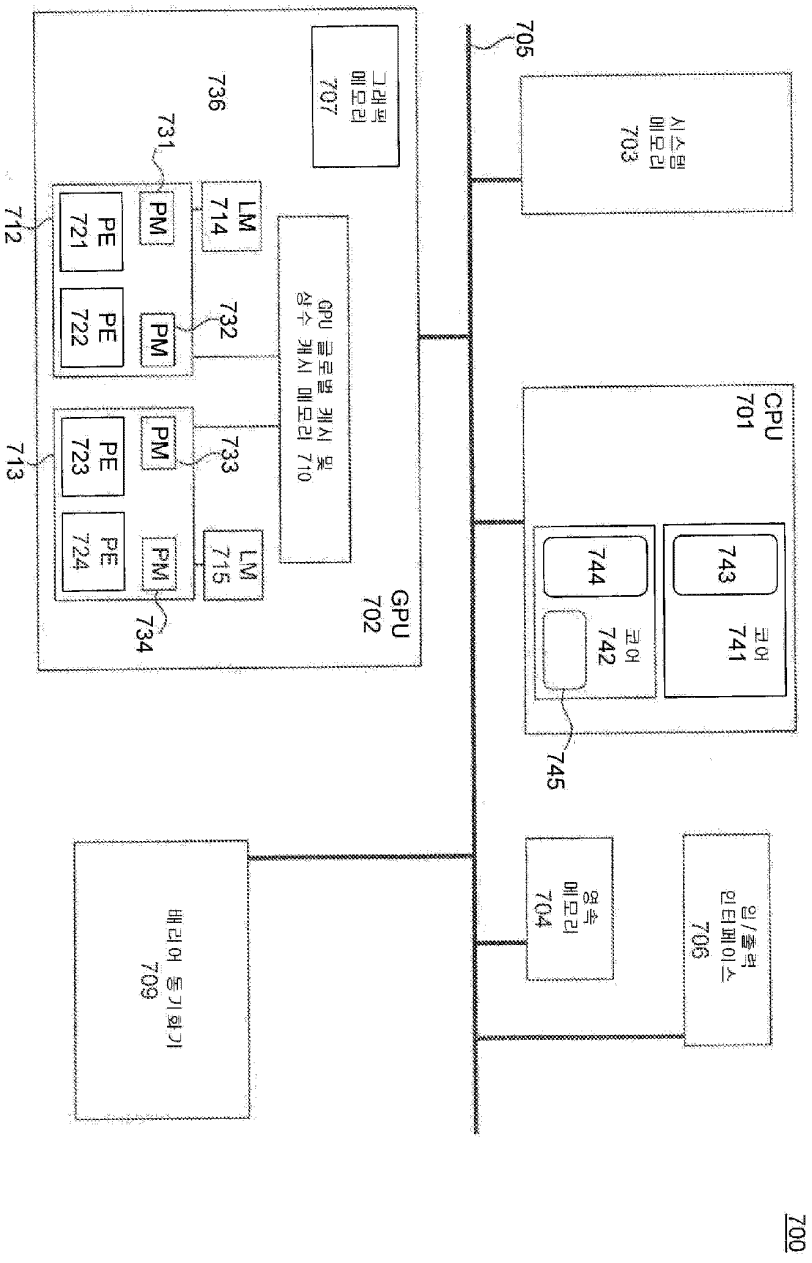
도면5


500
T1
T2
T3
T4
barrier.wait( )
501
barrier arrive( )
502
barrier.wait( )
503
504
barrier.skip( )
barrier.wait( )
505
barrier.wait( )
506
507
barrier.reset( )
barrier.wait( )
508

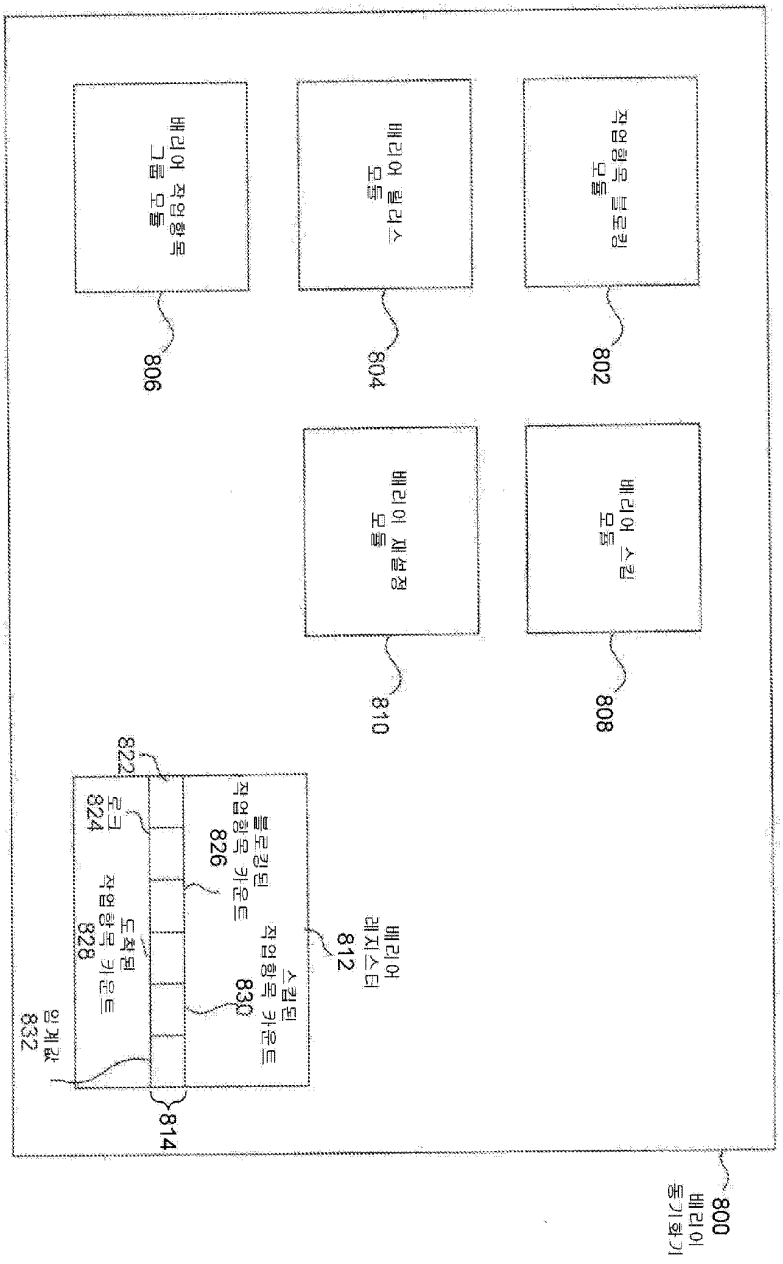
도면6


600
시작
작업항목의 그룹을 시작
602
배리어를 생성
604
배리어 릴리스 임계값, 방문 카운트, 스킵 카운트를 결정/설정
606
작업항목 x가 동기화 명령어에 도착
608
610
대기 명령어?
예
배리어 방문 카운트를 업데이트
612
아니오
624
도착?
예
614
방문 카운트+스킵 카운트==릴리스 임계값?
아니오
예
626
방문 카운트를 업데이트
아니오
630
628
작업항목 x를 계속
배리어에서 작업항목 x를 블로킹
616
618
배리어를 릴리스
스킵?
예
작업항목 x를 계속
617
620
방문 카운트를 재설정
632
스킵 카운트를 업데이트
아니오
622
모든 블로킹된 작업항목을 계속
638
재설정?
예
아니오
640
배리어 재설정 (방문 카운트 및 스킵 카운트를 재설정)
636
작업항목 x를 계속
작업항목 x를 계속
639

도면7


700
705
시스템 메모리 703
CPU 701
코어 741
743
코어 742
744
745
GPU 702
그래픽 메모리 707
736
731
712
713
PE 721
PE 722
PE 723
PE 724
PM
LM 714
LM 715
732
733
734
GPU 글로벌 캐시 및 상수 캐시 메모리 710
영속 메모리 704
입/출력 인터페이스 706
배리어 동기화기 709

도면8



【심사관 직권보정사항】

【직권보정 1】

【보정항목】 청구범위

【보정세부항목】 청구항 9, 18

【변경전】

도달하도록 요구하도록

【변경후】

도달하는 것을 요구하도록

【직권보정 2】

【보정항목】 청구범위

【보정세부항목】 청구항 19-20

【변경전】

상기 그룹

【변경후】

그룹