GPU presentation

If you are confused about simt you can find a tutorial explaining it in the extra learning document.

Review

AUDI

A cache is a method of memory storage that gets the value from on disk and brings it closer. In cpus there is usually an L1, L2, and L3 cache.

A process is an instance of a program (a bunch of code in storage) being run by the OS.

The address space of an OS is a range of memory locations that are assigned for a process. Address spaces are split into sections such as the code and data section. The code section stores all the information on how a process is supposed to run while the data section stores memory elements for a process.

NIKESH

Review

Processes have a lifecycle which they go through. A simple example is a ready, running, blocked/waiting, terminated lifecycle. In this example a process would be loaded in and become ready. When it is chosen to run it would run until it stops and becomes terminated or more likely when it needs i/o or information that it doesn’t have it goes into the blocked/waiting cycle. When the information is ready the process becomes ready again and the cycle continues.

A cpu instruction cycle works by first fetching the instruction that needs to be run before decoding the instruction to send it to the correct part of the cpu. When it is sent it starts to execute and the cycle continues.

NIKESH

Review

Parallelism is the ability to perform multiple operations simultaneously.

One from of parallelism happens because all of the steps in the instruction cycle only require the input of the previous step but doesn’t require it to run.

For example the decode step needs the fetched instruction but after the instruction is given the fetch step doesn’t need to run. For this reason the steps are run altogether with the fetch step giving the information to the decode step and then running again so that when the decode step gives its information off to the execute step it can be run again with the information that the fetch step has gotten in the meantime.

Review

AUDI

A compiler translates a program into machine code or instructions that the cpu can work with. Most compilers nowadays optimize code to be faster by making them contain less actual instructions and optimizing how they access memory.

The most simple definition of a thread is multiple processes being running simultaneously in the same address space. They share their data sections with each other and this causes problems that need to be handled with certain techniques that will be talked about later.

NIKESH

Integrated vs stand-alone gpus

Gpus are sometimes integrated into a computer so that they stand with the cpu and share memory. One example would be the Intel I9, with integrated a GPU in the CPU. Stand-alone gpus (sometimes called discrete gpus) have their own memory resources and are connected to the cpu & motherboard externally (most times through PCIe such as 2080 TI that you can plug into your motherboard).

Gpus usually assume there is a cpu because gpus are hyper specialized to work in parallel as it is more efficient to focus on assigning the instructions given than building a different module that can handle running a computer on its own.

NIKESH

Gpu/cpu interaction

The cpu sends an instruction to the gpu and allocates memory for use by the gpu using the gpus drivers. Then the cpu uses the kernel to tell the gpu what to run.

Gpus are optimized to be very specialized for very repetitive tasks because of they were originally intended for graphics processing.

The way gpus do this is by utilizing threading with vector registers. A regular register from what you might be familiar with are also called a scalar register, meaning it only holds 1 value. A vector register holds multiple values. Then a select number of threads are “combined” into warps(NVIDIA) or wavefronts(AMD) and are run together.

Programs are written as if there is only 1 thread and the hardware later runs them with multiple threads.

NIKESH

Gpu manufacturers

When using a gpu you are very much at the mercy of your gpu manufacturers. AMD and Nvidia are the main ones you will run into and you can code for you gpu with Cuda and OpenCL and are translated to languages such as PTX assembly(assembly for threads).

We can assume general implementations (i.e. the general purpose on gpu or gpgpu) such as having to work with memory being accessed by multiple threads. An example of how they mitigated this problem is by using a cache that is built to miss a lot is an effective way of getting relevant information close to the execution units is used in gpus in order to bridge these extremes (pg 41).

Gpu instruction cycle

NIKESH

The gpu has 3 stage parallel design with the instruction fetch and a decode which are both used in the simt portion of the design and an execute which uses simd.

An SM would fetch the instruction from a program counter to be run by the gpu.

SIMT means Single input multiple threads and will involve a single instruction that is given to multiple threads.

SIMD means Single input multiple data and will be discussed later.

Gpus schedule by warps/wavefronts instead of processes like cpus.

NIKESH

GPU Components

The physical components of a GPU are the streaming multiprocessor(SM) or the compute unit(CU). SMs can be thought of in a similar way to a cpu core. They store multiple alus, fpus, and other necessary things for threading on a gpu. Modern gpus have 100-144 SMs. These SMs are grouped into a Graphics processing Cluster (GPC).

The L2 cache is shared across SMs however the L1 cache is not.

NIKESH

SIMT intro

An instruction is run on every single thread in a warp with each not necessarily having the same data fed into them, but they operate the same instruction.

For example one thread may execute [add register1,2] another may execute [add register1, 3] and another may execute [add register1, 2].

The main purpose of SIMT is to be a software abstraction.

What would mess up the SIMT flow?

Specifically what would mess up all these threads running the same instructions at the same time?

The answer

Conditionals (if-statements). If-statements would mess the execution because one thread may recognize the condition as true while another may recognize it is as false due to all threads being fed different data. In these case two different paths would be run.

SIMT during execution

AUDI

When the threads come across a conditional then the thread execution branches into two groups, the ones that do execute the conditional and the ones that don’t.

The branching is called a thread divergence and it involves the executing group of the not executing group (whichever the designers feel like) running all threads in that group.

Let’s assume that we want to run the execute group first. If we have a branch if equal instruction and threads 1 and 2 are equal and threads 3 and 4 are not then threads 1 and 2 will execute until they reach another branch. Then branches 3 and 4 would run.

Basic CPU Threading

AUDI

Older versions of gpus used a stack to determine which location to go to next. However there was a problem of threading of “simt deadlock”.

To understand simt deadlock we must understand locking of threads. In cpus when threads are about to access something that has a risk of being accessed by multiple threads and therefore modified in a way that we make it so only one thread runs.

Normally this wouldn’t be a problem because the lock is eventually removed but if the thread stops running before it unlocks it then nothing will ever happen. Modern gpus use a different system.

SIMT

AUDI

When the branches come back together they perform what is called a thread convergence. When this happens all threads run together again.

The mechanism that reconverges threads is called a barrier protection mask in conjunction with a barrier state field to “map out” the points where branches happen ahead of time.

A thread state is used to determine whether threads in a warp are yielded, at a convergence barrier or ready. A thread rPC state, which is on every thread in the warp, tells the next location to branch to when the next branch divergence or convergence happens. The thread active field is specifically for specifying if a thread is active or not.

SIMD

NIKESH

SIMD is also prevalent in cpus it can guessed to be similar enough.

The SIMD/vector registers work by having “packing” registers or taking multiples of the actual number of bits in the register.

For example a 128 bit SIMD register may take 4 32 bit inputs and stuff them together in one register.

NIKESH

SIMD

These registers then apply the operation that was done during the SIMT portion.

For example if the SIMT portion added numbers then the SIMD will also add.

These additions would be done over input given by the individual threads, so all threads might not get the exact same numbers to add.

Simd is also very commonly present in cpus with intel cpus using avx (advanced vector extension), sse and mmx. The simt will forward the input with the control flow handled while the simd is able to do all of the traditional maths.

AUDI

Banking

Gpus use banks to group register files together to access them in parallel.

A gpu uses a system where an input is given to an arbiter which selects a register to be read/written from multiple connected register files called banks and sends them to the execution units via a crossbar.

This does not allow for the reading of multiple registers from one bank. In the worst cases this can cause extreme inefficiency. For that reason an operand collector is used to schedule the reads so that the maximum number of registers can be read per cycle. (pg 58)

Implementing this can cause significant hazards however there are certain things that can be done to fix it that we won’t cover (pg 60).

Swizzled Banks

audi

Banks are swizzled meaning that register files will not be stacked on top of each other in the same way. The register files will be stacked in a “mod 1 fashion”.

What that means is that a theoretical 4x4 register file will start will go from register1 to register 16 with the bottom leftmost part of the 4x4 grid being filled up by register1 and the registers continuing till the grid is full. The register file on top of that one will however start from register 16 at its bottom leftmost grid space and the right grid space will house register 1.

This is done so that threads will not all access the same register and have a bank conflict for every single thread.

Operand Collector

Audi

The operand collector runs after the simt has taken its course. The use of the operand collector is to allow banking to happen more effectively. This is done by getting the values that instructions need to run strategically into the bank so that multiple warps can read their specific instructions simultaneously.

For example if warp1 wants to access [register1] and [register5] and warp2 wants to access [register1] and [register2] then the first cycle of the instruction would get the instructions for warp2 and register1 from warp1. Assuming we have a 4 port bank this is done because of the swizzled bank and because register1 and register5 can both be read simultaneously.

Memory

AUDI

Scratchpad memory which is managed by the programmer is also present.

Scratchpad memory is direct mapped which means that the memory address is given and the physical location that memory address is tied it.

When reading/writing with scratchpad memory the arbiter determines whether there is a bank conflict. If there is a bank conflict among some of the threads the threads split into the groups that are and aren’t in conflict. The in-conflict “replays” or sent to the instruction pipeline to be executed again.

Memory

audi

When reading from a cache the memory locations that need to be accessed are first “coalesced”. What that means is that locations that are close to each other and will be read in together so as to not have to have multiple reads. The arbiter determines whether the cache can handle a cache miss.

If it can then the cache is checked. If there is a cache hit then the value is written back to the registers. If there is a cache miss then the registers are given a replay order and the request info is sent to a pending request table (PRT). The PRT saves vital information and when the cache miss is resolved the information from the PRT is sent to the registers so that they can call the cache again.

L1 Texture Cache

audi

Gpus were originally designed for graphics processing so a texture cache is implemented to give memory for textures as fast as possible. It first checks if there is a cache hit or cache miss through a tag array and sends it to one of two FIFO (First in first out) buffers. If there is a cache miss then the FIFO buffer is called the miss request FIFO and it handles requesting information from memory. It then sends it to a reorder buffer which makes it acceptable to a controller.

When the data is ready, either after a cache hit or miss then the controller gets it from the cache and sends it to the texture filter which sends it to the registers.

Modern gpus combine the L1 cache and texture cache.

NIKESH

Parallelism types

There are three types of parallelism: instruction level, thread level and data level parallelism.

ILP refers to executing multiple instruction from one thread at once such as pipelining, breaking down the execution steps to speed up processing, and out of order execution, finding instructions that are not very intensive to complete and reordering the line of execution around them to keep threads busy.

DLP commonly refers performing the same operation on different data points such as SIMD technology

TLP refers to parallelism across multiple threads or cores.

Can y’all point out parallelism in the lesson?

Parallelism

NIKESH

One major problem with parallelism is that occasionally a computer will want to do something with an instruction that has not been resolved. This means that the value is a variable and the value pointed to by a variable is not yet known to the computer because it is in memory or the cache. When this happens the computer needs to wait for the variable to be resolved.

Being able to predict which values need to be resolved and which aren’t is a necessity for making parallelism fast because the instruction cycle needs to wait for circumvent itself in order to resolve the variable. An instruction cache is used to have instructions that can be done quickly on hand.

NIKESH

Hazards

Hazards are primarily a problem with parallelism that involve a variable being read or written to when the value is not as it is intended. An example is a write after read (WAR hazard) which occurs when a read that is supposed to read a value that will be written is read before and gives a value that is not correct.

A way to handle hazards is using a scoreboard or a reservation station. A reservation station is not good for gpus however a scoreboard can be made to (pg 54). A scoreboard in a cpu would assign single bit for each register and set the bit when a value is being written to it. The register would only allow the register to be written/read from when the bit is not set. Modern scoreboards are 3 or 4 bits and block instructions from being put in the instruction cache to be scheduled at all.

I-cache vs d-cache

NIKESH

In the L1 cache for cpus there is an i cache for instructions and a d-cache for data. The d-cache is a cache that yall are familiar with. The i-cache is responsible for instructions that can be run immediately by a cpu.

Instructions that require i/o or are reliant on variables that haven’t been modified yet need to wait but instructions that are just adding registers or numbers can be run immediately. These are but in the i-cache so that the cpu can pipeline itself effectively.

Gpus run a similar system with their instruction caches.

Additional research

Should you want to look into gpus we first recommend looking at the textbook in certain sections. We also recommend that you first get a decent understanding of cpus, parallelism and threading before you start with looking at gpus.

The gpu textbook has a lot of research portions.