Cache Aware Programming on Multicores

Contents of Lecture 10

- Cache Misses
- Reduce Communication
- Impove Locality
- Data Prefetching

Cache Misses

- Cold miss: first time a variable is accessed.
- Capacity miss: miss due to cache size regardless of associativity.
- Conflict miss: miss due to limited associativity.
- True sharing miss: essential miss since it communicates data
- False sharing miss: non-essential miss. We could have ignored the invalidation

Reduce Communication

- Ideally, each processor should work on its own data and no other should be involved. No communication.
- With communication, try to balance amount of tasks and communication.
- Small tasks gives better load balancing but usually more communication.

Exploiting a single consumer

- Suppose only one thread should be informed about an event. Instead of having one spin lock that all threads wait on, let each waiting thread have a unique spin lock so only one needs to know about the event.
- The "producer" needs to know which "consumer" to wake up of course.
- This technique can be used e.g. with spin locks in large machines.
- Instead of one spin lock we use an array of spin-locks and at an unlock the next element in the array is unlocked.
- Through this only the next thread in line will know about it.

False Sharing Miss

- Assume a cache block size of two words.

Access	*Processor 1*	*Processor 2*	*Comment*
1	`Load 0`		Cold miss
2		`Load 1`	Cold miss
3		`Store 1`	Invalidation
4	`Load 0`		False sharing miss

Effects of larger cache block size:

- Increased benefit from spatial locality (prefetching within block)
- The larger risk of suffering from false sharing.

True Sharing Miss

Access	*Processor 1*	*Processor 2*	*Comment*
1	`Load 0`		Cold miss
2		`Load 1`	Cold miss
3		`Store 1`	Invalidation
4	`Load 0`		True sharing miss
5	`Load 1`		Reads a new value

- While we *cannot* know it at the time of Access 4, that miss is a true sharing miss (which we realize at Access 5).

Reducing False Sharing

- Make sure each thread uses its own cache block.
- Bad idea to put multiple spin locks in an array.
- Put pointers to spin locks in the array instead.
- Cache block size usually not fixed for an architecture.
- Some Power processors use 32 bytes while e.g. 970MP uses 128 bytes.

Improve Locality

- Loop reordering and blocking as in the uniprocessor case
- Small data structures.
- Be aware of how they are accessed.
- With large structs, put the fields used at the same time near each other to make them be fetched in the same cache miss.

Data Prefetching

- The purpose is to fetch data so that it is available in the cache when it's needed.
- Compilers and hardware can do this for matrix codes.
- This is very difficult on recursive data structures such as lists or trees.
- Suppose we have a loop which traverses a list or tree.
- To prefetch a node needed e.g. three iterations ahead, we need to dereference multiple pointers where each dereference can result in a cache miss.
- In a superscalar processor with out-of-order execution of load instructions (i.e. a relaxed memory consistency model), this can possibly be useful.
- In a processor with a blocking cache, the pipeline will halt at the first cache miss and make the prefetching almost useless.

An Approach to Prefetching Nodes

- The problem with lists and trees is that we usually do not know the address of a node needed in the future.
- This is true if we allocate memory with standard methods such as `malloc` while an arena is more predictable.
- However, assume the size of a data structure is fixed for some time.
- Then we can put pointers to the nodes in an array in the expected order of traversal, and then we may be able to prefetch nodes sufficiently in advance.
- This can be useful if we will traverse a data structure multiple times.
- An example: the control flow graph in the course project, or the dominator tree of a control flow graph in an optimizing compiler.

More difficulties

- For shared data we intend to modify, it can be useful to prefetch it in exclusive mode, meaning that we request ownership of the cache block.
- The effect of this is:
  - Reduced write penalty in a sequentially consistent machine.
  - Reduced write traffic in all machines.
- However, with the ownership requests, there is a risk that we introduce additional cache misses!
- Measurements are needed, but note they are dependent both on the
  - Input data
  - Machine parameters such as number of processors, cache sizes, and latency.

Hardware-based Data Prefetching

- Several processors, including 970MP, do prefetching of array references.
- They work by discovering a constant stride and then predict which blocks will be required.
- Power also supports software programmable prefetch engines.

Cache-Miss Initiated Software Controlled Prefetch Engines

- A paper by Michel Dubois and myself evaluated the following:
  - Treat L2 cache misses as leight weight exceptions — there soon will not be much to do for the processor to do anyway.
  - Such exceptions do not involve the OS kernel but simply jump to a special place in the program.
  - For certain references in certain loops, the compiler has created an exception handler which will program a prefetch engine.
- The exception handler is a part of the function's control flow graph so it has access to all local variables which are register allocated both for the function and the exception handler.
- Therefore the exception handler can compute what to prefetch while the L2 cache miss is being serviced.
- The instruction overhead of always prefetching is removed.
- Knowing whether to insert prefetch instructions or not can be impossible, e.g. for `memcpy`.

Software Controlled Stream Prefetch on Power

- Four data streams can be prefetched concurrently
- The basic instruction is `dst` — data stream touch
- One of the instruction fields is a two bit stream selector
- Other parameters:
  - Prefetch unit size S in 16-byte blocks: value is 0 .. 31 where 0 means 32.
  - N Number of units to prefetch
  - Distance D in bytes between two units (i.e. stride)
- Use as follows:
  - `#include <altivec.h> vec_dst(addr`