Summary HPSC 1.4-1.8

Erik Amézquita

Multicore architectures

Reasons to have multicore architectures

- Clock frequency can't go up due to heating, energy, etc.
- ILP hits its physical and theoretical limits

This architecture provides

- Energy efficiency
- Explicit task paralelism

Pyramid of shared memory

- Core: Registers and cache L1
  - Socket: caches L2 and L3
    - Node: motherboard and RAM
      - Network

Cache coherence

If one copy is altered the rest of copies must be updated or invalidated accordingly. The cache lines can be described as in the Modified Shared Invalid (MSI) coherence protocol as:

- Modified: must be written on RAM
- Shared: present in at least one cache unmodified
- Invalid: not in cache or modified and not updated

Snooping: Finding whether a given cache line is M or S

Possible solutions to the cache coherence problem include

- Snooping: requests for data are sent to all caches. Cores can "listen" to all bus traffic. Invalidating is much cheaper than updating.
  - This doesn't scale up
- Tag directory: keep record of what is where. May work when dealing with large number of cores.

A related problem is *false sharing*, when a cache line is constantly moved between cores, thus being constantly modified and invalidated.

Node architecture and sockets

When working with several sockets, these may have a Non-Uniform Memory Access (NUMA) design. The memory attached to its socket is faster to access than the memory attached to another socket. This can lead to the *First-touch phenomenom* which reduces gains of parallelization. That is, when dealing with arrays, these might end completely allocated in the memory of a particular socket. Thus this particular socket would be the only one to deal efficiently with such array. A solution is to parallelize the initiation loop of the array.

Locality and data reuse

Keep data as close as possible to the processor. Minimize data transfer.

Data use and arithmetic intensity

Arithmetic intensity: operation per byte ratio (# operations)/(# data items) It tells us how possible is to reuse data and hide latency. Strive for high arithmetic intensity.

Locality

- Temporal: use of data within a short time of its last use
- Spatial: Maximize the number of used data items per cache line
- Core: Write accesses that are spatially or temporally close near and perform them on the same core.

Programming strategies for high performance

Peak performance

Computed theoritically based solely on CPU specifications and clock cycle. Does not take into account bandwidth and latency issues. Never achievable in practice.

Pipelining

When traditional pipelining is not possible, unrolling the cycles might alleviate. Thus the processor is always performing some operation. Too much unrolling however might lead to register spill and hamper the performance. The number of unrolling usually won't divide the number of total iterations, thus it will also require additional cleanup code to deal with the last iterations.

Cache size

As observed in ICA2 assignment, the performance drops when cache levels are filled up.

Cache lines

Important to consider strides and reading of cache lines when dealing with nested loops. Keep in mind to maximize spatial locality.

Loop tiling

Break a loop int two loops. Thus an array can be split in blocks (tiles) such that each block fits into the cache. Hence the operations dealing with the block are highly efficient. This requires more operations but less bandwidth.

Optimization strategies

Most top-notch optimization strategies are highly dependent on the machine's architecture. Thus the high-efficiency is usually non-portable.

Further topics

Power consumption

The faster the clock cycle of a processor, the more energy a processor consumes. This also implies that the cores will radiate more heat. Physically we've reached the clock cycle limits for a single processor. Thus a multicore architecture and parallel computing are the only way to leap forward.

Operating system effects

An OS might interrupt a computation to run other non vital daemons in the background. This affects the processing times and contaminates the cache. Thus HPSC is usually ran with a minimal OS.