Algorithm Engineering
Exam Assignments

Sheela Orgler

Friedrich Schiller Universität, Jena

1 Assignment

1.1 Describe how parallelism differs from concurrency

Concurrency refers to two or more tasks being executed, but not simultaenously. Parallelism is about the exection of multiple tasks at the same time. Parallelism is a subset of concurrency. If a system is parallel it is also concurrent, but not the other way around.

1.2 What is fork-join parallelism?

Fork-join parallelism is a concept where we have a master thread that divides into more threads. This opens a parallel regions allowing simultaneous excecution. In between these parallel regions the execution is sequential following the master thread.
We can imagine execution as a line that is followed from starting to end point. With fork-join parallelism this line is our master thread. Following our master thread at some point the thread divides into two or more threads. These threads are joined after some time to our master thread. In these sections where we have two or more threads, execution will be parallel and after these sections sequential.

1.3 Chapter 1 – Computer Systems: A Programmer's Perspective

Discuss one thing - Caches As explained in Chapter 1 of the book "Computer Systems: A Programmer's Perspective" caches are helpful to deal with the processor-memory gap. The processor-memory gap refers to the difference in performance between processor and memory. The processor can read data from the register file, which is within the CPU almost 100 times faster than from memory. The gap between processor and memory is continuously growing.
Cache memory is used to bridge the gap. It is used to temporarily store information that will likely be used in the future. There are several levels of cache - L1, L2, L3 - which go from smallest to largest. A cache reduces the acess time to data in memory. Frequently used data and instructions are kept in the cache. Accessing data from the cache is faster.
Modern desktops, servers and industrial CPUs have at least three independent

caches: instruction cache, data cache and the TLB. The instruction cache is used to speed up executable instruction fetches. The data cache speeds up data fetch and store. It is organized into more cache levels:

- L1 cache or primary cache is fast but small and usually embedded in the processor chip (CPU).
- L2 cache or secondary cache is larger and can be embedded in the CPU or on a seperate chip or coprocessor containing a high-speed alternative system bus connecting cache and CPU.
- L3 cache is used to improve L1 and L2. L3 is usually double the speed of DRAM.

The small caches L1 are backed up by larger and slower caches L2, L3 to address the tradeoff between cache latency and hit rate. The faster cache is generally checked first. If that cache misses, the next chache is checked and so on before accessing main memory.
For example the ARM-base Apple M1 CPU has 8 cores: 4 high-performance and four high-efficiency cores. The four high-performance cores have a 192 KiB L1 cache for each of the cores and the four high-efficiency cores only have 128 KiB.
The Translation lookaside buffer is used to speed up the translation from virtual to physical addresses. It is part of the memory management unit and not directly related to CPU caches.

1.4 Paper – There's plenty of room at the Top: What will drive computer performance after Moore's law?

Explain figure "Performance gains after Moore's law ends" The figure illustrates the "Top" and "Bottom" referring to computer performance gains. The "Bottom" is about the miniaturization of computer components seen in the last decades.
Due to physical limits the opportunities for gains at the bottom will slowly come to an end. Nevertheless, there are still opportunities for growth at the "Top". The "Top" shows the three aspects where growth can be expected: software, algorithms, hardware architecture. These aspects are divided into technology, opportunity and examples all looking into the growth opportunities in the specific area.
Growth opportunities at the "Top":

- making software more efficient by performance engineering
- minimizing the time it takes to run and not the development time
- an increasing number of processor cores running parallel
- reengineer modularity to obtain performance gains

2 Assignment

2.1 What causes false sharing?

False sharing occurs, when threads on different processors modify variables on the same cache line. A cache line is the smallest unit of memory. Its length

depends on the underlying architecture and is typically 64 and in more recent architectures 128 bytes long.
False sharing happens when for example two threads on different processors modify variables on the same cache line. The first thread modifies one variable at the beginning of the cache line. The second thread tries to modify a variable at the end of the cache line. Due to the alteration of the first thread, the cache line is invalidated for the second thread and has to be reloaded.
False sharing only happens when variables are changed and not when they are only read. It may lead to a significant performance decrease.

2.2 How do mutual exclusion constructs prevent race conditions?

Race conditions occur when two threads modify the same data. Mutual exclusions give one thread exclusive access to the data. After execution exclusive access is returned and the next thread can execute.

2.3 Explain the differences betweent static and dynamic schedules in OpenMP.

Static and dynamic schedules differ in how the work (iterations of the for loop) is spread between threads.
Static schedules decide at the beginning and dynamic means that it is decided at runtime. Each thread will work on a chunk of values and then take the next chunk that hasn't been worked on by any thread. Static schedules perform better for balanced workloads and dynamic schedules for unbalanced workloads, in case the workload varies between different iterations of the for loop. The chunk size can be specified; for static schedules it is one per default.

2.4 What can we do if we've found a solution while running a parallel for loop in OpenMP, but still have many iterations left?

If we've found a solution within a parallel for loop, we can use continue. We don't continue calculating, but quickly iterate through the loop to increase performance.

2.5 Explain in your own words how std::atomic::compare_exchange_weak work

strd::atomic::compare_exchange_weak succeeds only if the value of the variable to be updated equals the first provided argument. For example if for final_solution.compare_exchange_weak(previous, i) final_solution is equal to previous. Previous is updated with i, if it fails.

3 Assignment

3.1 How does ordered clause in OpenMP work in conjunction with a parallel loop?

An ordered clause is used within a parallel region. It opens a region where the execution is sequential. It is used to prevent race conditions.

3.2 What is the collapse clause in OpenMP good for?

The collapse clause is used to parallelize for loops. It allows to specify how many loops are collapsed into one. The collapse clause is useful for balancing work and parallelize multiple loops to boost performance. When using the collapse clause a single loop is formed. The length of the new loop is equal to the multiplication of the length of each loop that has to be collapsed into one.

3.3 Explain how reductions work internally in OpenMP

A reduction is an operation from OpenMP that is used in parallel computing. It reduces the values into a single result. The syntax of the operation is: reduction(op: list). op refers to the operation (addition, subtraction, and so on). list refers to a list of variables that is reduced. For example: A reduction can be used within a parallel for loop: #**pragma** omp parallel **for** reduction(+ : sum) to compute the sum of the values 1, 2, 3. Every thread within the parallel region gets a local sum variable, where the sum is computed. The result will be written to the global sum variable declared outside of the parallel region.

3.4 What is the purpose of a barrier in parallel computing?

A barrier can be set within a parallel region. Setting a barrier means that the threads will execute the code up to that specific barrier, which is a specific point in execution. At this point the threads wait until all have reached the barrier and only then continue with execution. Barriers are used for synchronization in parallel regions, where they enforce a specific execution order.
Within OpenMP there are also implicit barriers. The most general barrier is the omp parallel region itself. The region makes sure that execution is parallel within that region. After the code from that region is executed in parallel execution will continue sequentially.

3.5 Explain the difference between the library routines: omp_get_num_threads(), omp_get_num_procs() and omp_get_max_threads().

omp_get_num_threads(): The routine is called inside a parallel region. It is used to get the number of threads used within a parallel region.
omp_get_num_procs(): The routine is used to get the number of logical cores.

It can be used to define the number of threads reasonable to use in a parallel region. Per default the number of threads is equal to the number of logical cores. omp_get_max_threads(): The routine is called outside a parallel region and returns the number of threads supported in a parallel region. When the value is changed with omp_set_num_threads() within a parallel region, the method returns the set value.

3.6 Clarify how the storage attributes private and firstprivate differ from each other.

Storage attributes are used to specify the usage of variables within a parallel region. It is used for variables declared outside a parallel region. There are three different storage attributes: shared, firstprivate and private.
Private attributes make sure that each thread gets an uninitialized copy of the variable. Firstprivate attributes make sure that each thread gets an identical initialized copy of the variable. The variable is a local variable for each thread. When using private or firstprivate storage attributes the value of the global variable remains unchanged.

3.7 Write in pseudo code how the computation of pi can be parallelized with simple threads.

```
initialize num_steps to 100000000
initialize width to 1.0
initialize sum to 0.0

do in parallel:
  initialize sum_local to 0.0
  for i in {0, num_steps} do
    set x to ((i + 0.5) * width)
        add sum_local to (1.0 / (1.0 + x * x))

        do sequentially:
          add sum sum_local

initialize pi to sum * 4 * width
return pi
```

4 Assignment

4.1 Explain how divide and conquer algorithms can be parallelized with tasks in OpenMP.

Divide and conquer algorithms recursively split down a problem until the pieces can be directly solved. Such algorithms can be paralleized in OpenMP using the constructs taks and taskwait for the recursive call. OpenMP tasks are independent work blocks. A queueing system dynamically handles the assignment of threads to the tasks. Each thread picks up a task from the queue and executes it. This goes on until the queue is empty. Tasks are defined within a parallel region and within a single construct, to make sure that one thread creates the tasks. Taskwait is a construct that waits until all tasks are executed.

4.2 Describe some ways to speed up merge sort.

- Tasks - with task and taskwait
- Avoid copying - to prevent that additional memory is created recursively and make sure that initially memory is instantiated and reused
- if Clause - task will only be created, if a certain condition is true
- final Clause - tasks won't be created anymore, if a condition is true

4.3 What is the idea behind multithreaded merging?

The goal is to sort and merge two arrays using a divide and conquer algorithm. To do that we look at the median of the array with more elements. Then we look at the other array and compare the elements smaller than the median and merge these with the respective elements from the other array. That way merge is executed in parallel.

4.4 Read What every systems programmer should know about concurrency. https://assets.bitbashing.io/papers/concurrency-primer.pdf Discuss two things you find particularly interesting.

The order of written code can be changed as compiler try to optimize it and rewrite code to run faster on the targeted hardware. RAM hasn't speeded up the way CPU processors have. This creates a widening gap between the instruction fetch and the time needed to retrieve data from memory. To prevent reordering we can use atomic types in C or C++.
Atomicity means that something can't be divided into smaller pieces. Threads need to use atomic reads and writes to share data. If they are not atomic they are so-called torn reads and writes. To ensure atomicity one needs to make sure that variables used for thread synchronization aren't larger than the CPU word size.

5 Assignment

5.1 What is CMake?

CMake is a script language used to create build files that can be executed across different platforms. The build files are executed using a compiler. CMake provides tools to build, test and package software.

5.2 What role do targets play in CMake?

Targets define what is built. They are executables and libraries. Targets have constructors to build executables, libraries or tests. Targets can be thought of as objects with different properties. To change these properties we can use member functions. Within member functions the properties can be specified. Properties can be flags, directories or linked libraries.

5.3 How would you proceed to optimize code?

The most important thing is to try solving the problem. Once that is done and the code works, I would look at the performance. If performance can be improved, I would look at ways to do so. I would try parallelizing sections of code, try finding ways to improve the algorithm. I would continue doing so until the performance is good.

6 Assignment

6.1 Name some characteristics of the instruction sets: SSE, AVX(2), AVX-512

SSE, AVX2, AVX-512 are instruction sets on Intel CPUs. They differ in the supported vector length and the number of registers.
SSE: 128-bit vector length, 1999 - 2009 years of launch, 16 registers
AVX2: 256-bit vector length, 2011 / 2013 years of launch, 16 registers
AVX-512: 512-bit vector length, 2017 year of launch, 32 registers
CPUs for high-performance computing support the AVX-512 instruction set, wehereas modern mainstream CPUs support AVX2 at most.

6.2 How can memory aliasing affect performance?

Memory aliasing occurs when two pointers point to the same memory location. The compiler doesn't know if two pointers might point to the same address. To ensure safe optimizations, the compiler always assumes pointer aliasing. Therefore, we can give the compiler hints, to ensure that no memory aliasing was made. This can be done using the restrict keyword (__restrict__ for gcc). Memory aliasing is important for pointers as well as references.

6.3 What are the advantages of unit stride (stride-1) memory access compared to accessing memory with larger strides (for example stride-8)?

stride-1 - access sequential elements in memory, where all of the fields are equally distant to one another. The distance in this case is called stride. The difference between stride-1 and larger strides stride-n is that every nth element is accessed. Stride-1 is faster than larger strides. There is a maximum bandwith that can be

loaded from memory. Using larger strides leads to loading a higher bandwidth without fully using it.
We can check this by looking at the summation of two arrays. Summing up every 8th element is more or less as performant as summing up every element of the array.

6.4 When would you prefer arranging records in memory as a Structure of Arrays?

There are two ways to arrange records in memory: as Array of Structures or Structure of Arrays. Structure of Arrays: one struct with multiple arrays. Array of Structures: one array with the elements for each entry.
For example when we want to save the x, y and z coordinates. When using a Structure of Arrays we have one struct and three arrays for each coordinate. If we use an Array of Structures we have a struct array with three entries. Each entry storing the values of the x, y and z coordinates.
A Structure of Arrays is good for vectorization. In our example with the coordinates, we can access all the entries of a specific coordinate through the respective array. With a SoA the arrays are kept seperate for each structure field. Memory access is contiguous if we perform vectorization of the structure instances. When performing operations on one array of the structure a SoA leads to better bandwidth usage.

7 Assignment

7.1 Explain three vectorization clauses of your choice that can be used with #pragma omp simd.

#pragma omp simd transforms a loop into a SIMD loop.
Some vectorization clauses with #pragma omp simd are:

- safelen - The safelen clause allows to define a boundary for the length of vectors used within the loop. It is used to ensure the correctness of the program.
- aligned - The aligned clause is used to specify the alignment (for example 64 bytes) for the pointers used within the loop.
- collapse - The collapse clause is used to combine multiple for loops into one.

7.2 Give reasons that speak for and against vectorization with intrinsics compared to guided vectorization with OpenMP.

Vectorization with intrinsics is used to manage vectorization manually. It is basically wrappers around the corresponding assembly instructions.
Guided vectorization with OpenMP is easier to use but there is no guarantee that the compiler can vectorize the code.

7.3 What are the advantages of vector intrinsics over assembly code?

Vector intrinsics are easier to use than writing assembly code. One vector intrinsic often consists of more than one assembly instruction.

7.4 What are the corresponding vectors of the three intrinsic data types: __m256, __m256d and __m256i.

The intrinsic data types can hold floats, doubles or integers. Vectors can be interpreted as bytes (8 bits), shorts (16 bits), integers (32 bits), longs (64 bits) or quads (128 bits).

- __m256 - a vector of eight 32-bit floating point values
- __m256d - a vector of four 64-bit double values
- __m256i - a vector of signed or unsigned integer values

8 Assignment

8.1 Explain the naming conventions for intrinsic functions.

Naming conventions: _<vector_size>_<operations>_<suffix>

- _<vector_size>: specifies the size of the vector returned by the intrinsic function
- _<operations>: specifies the operation, like add, sub, mult
- _<suffix>: specifies the data type of the input arguments

Example: __m256d _mm256_add_pd (__m256d a, __m256d b)
Vector size: m256d - 64-bit floating point values
Operation: add - add the corresponding elements of two vectors
Suffix: pd - double precision (epi, epu - signed or unsigned)

8.2 What do the metrics latency and throughput tell you about the performance of an intrinsic function?

Latency is the number of cycles an intrinsic takes until its result is available. Throughput specifies how many cylces it takes to start the next intrinsic function of the same kind. Latency and throughput can vary depending on the underlying CPU. For the add intrinsic latency varies from two to four and the throughput is 0.5. A latency of four means that it takes four cycles until the result is available. A throughput of 0.5 means that within one cycle two instructions from the intrinsic can be started.
In general there are two units on the CPU to compute such functions. Because there are two units the throughput is 0.5. Within one cycle two add intrinsic functions can be started. It takes up to four cycles until a result is available.

8.3 How do modern processors realize instruction-level parallelism?

Instruction-level parallelism (superscalar) refers to several instructions which are being evaluated simultaneously on different functional units. Multiple instructions can be executed at the same time within a cycle using these different functional units but one CPU core.

8.4 How may loop unrolling affect the execution time of compiled code?

Loop unrolling is a technique to make use of instruction-level parallelism. It can be used for scalar or vector operations. The body of the loop is replicated and the logic is adjusted accordingly. The loop is unrolled by a factor k, which determines the number of times the body is replicated.
Loop unrolling may lead to an incresed demand for registers depending on the factor used. It can increase compile time and the program size because the body is replicated and additional space in the instruction cache is needed. Loop unrolling is more effective when using vector instructions to exploit instruction-level parallelism.

8.5 What does a high IPC value (instructions per cycle) mean in terms of the performance of an algorithm?

IPC (instructions per cycle) is an aspect of the processors's performance referring to the average number of instructions executed in one clock cylce. It an indication for the systems performance. A high IPC indicates that the CPU is efficiently used. To calculate the instructions per cycle the number of instructions is divided by the cycles.

9 Assignment

9.1 How do bandwidth-bound computations differ from compute-bound computations?

Bandwith-bound computations have memory access as bottleneck. When loading data main memory is accessed. For compute-bound computations the bottleneck is the CPU and main memory or RAM are not accessed. When optimizing the cache usage, bandwith-bound computations show more effective results.

9.2 Explain why temporal locality and spatial locality can improve program performance.

In modern computers caches are used to reduce the processor-memory gap. For this to work programs need to establish good temporal and spatial locality. The cache holds elements for the CPU that are often accessed to avoid access to main memory. The CPU can access the data directly through the cache.

Spatial locality means that elements should be stored efficiently. If one element of a cache line is accessed the whole cache line is loaded. Therefore, programs should be written in such a way that the other elements are used aswell. Everything that is loaded should be used as efficiently as possible.
Temporal locality means that programs should be written in such a way that the data is used as often as possible. Blocking is a mechanism which makes use of temporal locality. The computation is divided into blocks that are united afterwards.

9.3 What are the differences between data-oriented design and object-oriented design?

Data-oriented design (DOD) is a way to design programs that make efficient use of the CPU cache. DOD modifies the input to obtain the desired output. In object-oriented design everything revolves around defining, producing and operating on objects. Objects interact with each other through functions. In data-oriented design functionality and data is seperated. Functions are general purpose. It shifts the perspective onto the data and how it is laid out in memory and read from there. DOD uses Structures of Arrays (SoA) instead of Array of Structures (AoS) for data that is often used.

9.4 What are streaming stores?

Streaming stores are a way to write data from the CPU directly to random access memory (RAM) and not through cache. It can lead to better performance, if the data is not used. Another way to use streams is streaming loads to load data directly from RAM to the CPU.

9.5 Describe a typical cache hierarchy used in Intel CPUs.

Modern Intel CPUs typically have an L1 and L2 cache within each core. The L1 cache is divided into an L1 data cache and an L1 instruction cache. L2 is the unified cache for data and instructions.
Outside of the CPU cores there is an L3 cache, which is shared by all cores and represents the interface to main memory. The caches size goes from the smallest L1 to the largest L3 cache.
Each cache can be thought of as a hash table with for example 64 rows. Each row has a number of buckets, where each bucket is a cache line. The size for a cache line for Intel CPUs is 64 bytes.

9.6 What are cache conflicts?

Caches are basically hardware hash tables and as such they suffer from conflicts. Each hash bucket can contain N cache lines. Cache conflicts occur when more cache lines are put into a bucket than it can hold.

10 Assignment

10.1 Name and explain some useful compiler flags during development.

- Wall - enables compiler's warning messages
- g - generates debug information
- fsanitize=address - detects out-of-bounds-access, use-after-free and memory leaks
- fsanitize=undefined - detects undefined behavior at runtime

10.2 How could Intel oneAPI help you write better programs?

Intel oneAPI is a toolkit containing Intel compilers supporting CPUs, GPUs, FPGAs and other accelerators. It provides a cross-architecture programming model including performance libraries, analyzer and debugger tools as well as domain-specific toolkits.

10.3 What can we learn from the following quote? Premature optimization is the root of all evil (Donald Knuth).

Premature optimization refers to blindly optimizing everything without thinking about it. It is better to first profile and find bottlenecks that can be optimized. The approach shouldn't be too invasive especially in a program written by others. It is better to keep things simple.

11 Assignment

11.1 What is Cython?

Cython is a programming language. It is a superset of Python used to write C / C++ extensions for Python. Cython helps improve performance of Python programs.

11.2 Describe an approach how Python programs can be accelerated with the help of Cython.

Cython can be used in sections of Python programs that are performance critical. Python is weak in performance and Cython can be used to improve speed. This can be done by identifying sections in Python code that slow down performance. These so called critical sections are outsourced to Cython. In Cython the code sections are written in C / C++ or Cython. As Cython is a superset of Python valid syntax in Python is valid in Cython aswell. The Cython modules can be imported in a Python program and therefore improve performance.

11.3 Describe two ways for compiling a .pyx Cython module.

To compile a .pyx Cython module the command cythonize can be used. It compiles the file into a C/C++ file. After that the file is compiled into an extension module that is importable from Python.
Another way to compile a .pyx file is to use pyximport. This is used to automatically recompile and reload the module. The line "import pyximport" and "pyximport.install()" is used at the start of the Python script.

11.4 Name and describe two compiler directives in Cython.

Compiler directives are set through a special header comment. This comment is written before any code. Additional flags set in the command line overwrite the ones set as header comments.
boundscheck: This directive can be set to true or false and is set to true by default. If set to false, Cython assumes that indexing operations won't cause any index errors. Conditions that would cause index errors may instead cause segfaults or data corruption.
wraparound: This directive is set to true by default. In Python arrays and sequences can be indexed relative to the end. Negative indexing is allowed in Python, but not supported in C. If the directive wraparound is set to false, Cython can't check or correctly handle negative indices. If set to true, negative indexing usually raises an index error. For safe usage it is recommended to only use this for code that doesn't process negative indices.

11.5 What is the difference between, def, cdef, and cpdef when declaring a Cython function?

- def - is a normal python function
- cdef - can be called only in cython
- cpdef - can be called in cython and python

11.6 What are typed memoryviews especially useful for in Cython?

Typed memoryviews are used to pass Python or Numpy arrays to Cython. It provides a view on the underlying memory. Typed memoryviews provide access to memory buffers. They are especially useful for those underlying Numpy arrays. Memoryviews provide inside on the underlying memory layout, which can be used for code optimization.

12 Assignment

12.1 What are extension types in the context of Python?

Extension types can be used to wrap C / C++ code looking like Python objects. They can be used to boost performance. Cython makes it easy to create extension types.

12.2 How do extension types data fields in Cython differ from data fields in Python classes?

Extension type data fileds in Cython can be accessed only within Cython code. To access fields from Python an access level - readonly of public - for that field has to be provided.

12.3 Give a simple description of how to wrap C / C++ code in cython?

We have a C / C++ library we want to use in Python. In our Python code we include the library using #include "pi.h". In Cython we describe all we need with compiler directives.

13 Assignment

13.1 Delimit from each other the following SSD parts: Cells, Pages and Blocks.

In SSDs cells are used to store digital bits. Cells are NAND-based and can store from one to four bits depending on the type of NAND. A group of cells forms a page. The smallest unit to read or write with SSDs is a page. It can only be written once. This means that there are no operations to overwrite a page. When data is updated it is written to another page. The old page is marked as invalid until it is erased. Blocks are usually 512KB, 1MB (about 128 or 256 pages). The smallest unit to erase is a block.

13.2 What is the purpose of garbage collection in SSDs?

Garbage collection is used to periodically optimize an SSD and maintain performance. Garbage collection claims blocks that have more invalid pages than a given threshold. The blocks can later be used for write operations. It also moves realted data segments next to each other.

13.3 What is the purpose of wear leveling in SSDs?

War leveling is used to prolong the life of an SSD. It should prevent an unbalanced usage of pages, meaning that certain pages are overused and others not at all.

13.4 Tell some interesting things about SSDs with an M.2 form factor.

The Host Interface provides an interface between host and SSD, such as PCI, SATA or SAS. To efficiently communicate with the interface the protocol AHCI or NVMe is used. NVMe protocols are more efficiently with SATA interface. The M.2 form factor exists for both, PCIe and SATA interfaces. It has multiple key slots, which have to be checked for compatibility. Performance is higher for write operations than read operations.

13.5 What influence do garbage collection and wear leveling have on write amplification of an SSD?

Write amplification happens when writing data and the data to write is multiplied into a disproportional amount of data on the SSD. Garbage collection and wear levelling can increase write amplification.

13.6 Discuss three different recommendations for writing code for SSDs.

Compact data structures should be used when writing code for SSDs. The smallest unit to update in SSDs is a page. Even if only a single bit is updated, it will always result in an SSD write of at least 4 KB. The same goes for reading. Therefore, small updates should be avoided and compact data structures should be used.
Code for SSDs should **avoid full SSD usage**. This influences the writing performance and write amplification factor. A full SSD causes more blocks to be moved around to free a block, which affects performance.
An SSD has multiple levels of internal parallelism. For /**small IO multiple threads** should be used. When reading small pages, multiple threads make efficient use of the internal SSD parallelism. For big IO only a few threads suffice, as we can make use of an SSD's internal parallelism.

13.7 How could the CPU load for IO be reduced?

To reduce CPU load one can use asynchronous IO system calls instead of blocking ones. Furthermore, the OS buffering has to be diabled.

13.8 How could you solve problems that do not fit in DRAM without major code adjustments?

To solve problems that do not fit in RAM one can use memory mapped files on flash. This can also be done using Numpy for computations that don't fit into memory.