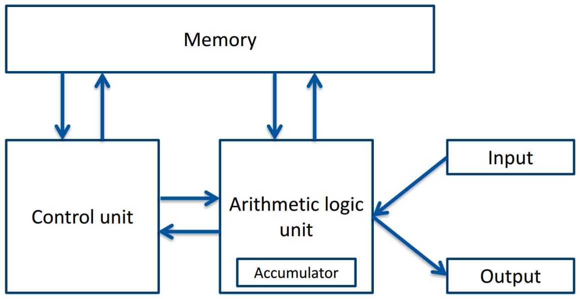
Contents

Chapter 1

Modern Architectures

1.1 Von Neuman architecture

1. instruction decode: determine operation and operands
2. fetch operands from memory
3. perform some operations with them
4. write the result back into memory
5. continue with next instruction



Chapter 2

Serial Optimization

There are two basic rules in optimization:

1. In a world of highly parallel computer architectures only highly scalable codes will survive
2. Single core performance no longer matters since we have many of them and uses scalable codes

But anyway, optimizing the serial code runtime also optimizes the parallel code runtime, since the code reduces the overall amount of needed computer power. So **the first goal when optimizing an existing code must be optimizing the serial code**. Figure 2.1 shows the steps how to optimize serial code.

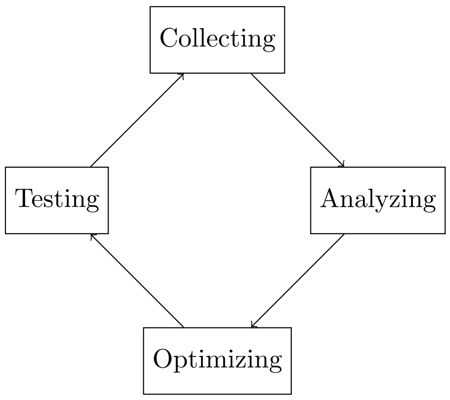


Figure 2.1: Steps to improve serial code.

Lets go a bit more into detail of the algorithem summed up in figure 2.1. The first step must be collecting informations. There are serveral ways to get these informations,

1. statically analysis of the source code
2. retrieving information about a programs runtime behavior (e.g. by **??**) → dynamic approach

3. Usually one determines how much time is spent in a certain function possibly identify Hot Spots

If it is known where the most runtime is spent in, it is necessary to find out why so much time is spent in this part of the code. Therefore the collected data must because analyzed. The first step to the answer is to determine which factors stall the performance (e.g. by hardware counters). The next steps are obviously optimizing the code and testing. The size of the test is important. If the test size is to small the performance behavior may not change signicantly. It if is to large the test may need to much time to be done. If testing is finished the algorithm starts at data collection again.
Definition 2.0.1 (Hot Spot)**.** Once there exists some kind of hotspot. In these hotspots 90% of the runtime is spent, but the size of code, and so the size of the hotspot is only about 10% of the overall code. Nowadays this approximation does not hold in generals but hotspot analysis is still important.

2.1 Monitoring

A common method for optimiziation a system's activities is *monitoring*. In the following there will be serveral definitions that are needed during the monitoring process.
Definition 2.1.1 (Event)**.** An event is an predefined change in the systems's state. The definition depends on the measured metric, e.g. memory reference, processor interrupts, appilaction processing, disk access, network activity.
Definition 2.1.2 (Profile)**.** The profile is an aggregated picture of an application program. For exampleaccumulated runtime spent in each function.
Definition 2.1.3 (Trace)**.** A trace is a log/sequenc of individual Events. This includes event type and important system parameter.
Definition 2.1.4 (Overhead)**.** The overhead is the Pertrubation introduced by the monitoring technique.

The level of the monitoring implementation can be devided up into

1. Hardware monitoring
2. Software monitoring
3. Hybrid monitoring

2.1.1 Event- and sample driven triggers

For the monitoring there is some kind of trigger needed. One way is the **event-driven** triggers. With these triggers the performance measurement is done if and only if the pre-selected Event occurs. So the Overhead depend on the number of the pre-selected Events. So the calculation of the Overhead is not easy. Furthermore this kind of trigger can significantly alter the programs behavior. This trigger is good for tool with low-frequency events. The other kind of trigger is called **sample-driven** trigger. Here the performance is measured over snapshots at fixe time intervals. So the Overhead of this technique is independent of the number of specific events. It takes snapshots on a specific frequency which the Overhead depends on. But this trigger obviously does not have to measure every occurence of a specific Event, infrequent Events may not be covered at all. It just produces a statical view on the overall behavior of the system. So only very long runs are likely to produce a comparable result. view on the overall behavior of a system.

	Event Trigger	Sample Trigger
Precision	Exact	Probabilistic
Pertrubation	$O(f(N_{events}))$	fixed
overhead	$O(f(N_{events}))$ Depends on: · event types instrumented · program behavior · overhead per event	constant Depends on: · Sampling rate · Overhead per sample

Table 2.1: Comparison between even- and sample-driven triggers.

Basic Block Counting

One way to implement monitoring is to use **Basic Block Counting**. A **basic block** is a sequence of instructions that has no branches in or out of the sequence. This method just adds an instruction to the block to count the number of times the block is executed. After the termination the measurements from an execution can be represented as an histogram. In contrary to sample-driven trigger, block counting gives the exact number of times a block was executed. But this method also includes a siginficant runtime overhead. This is the reason why block counting can have a severe impact on program's behavior/performance.