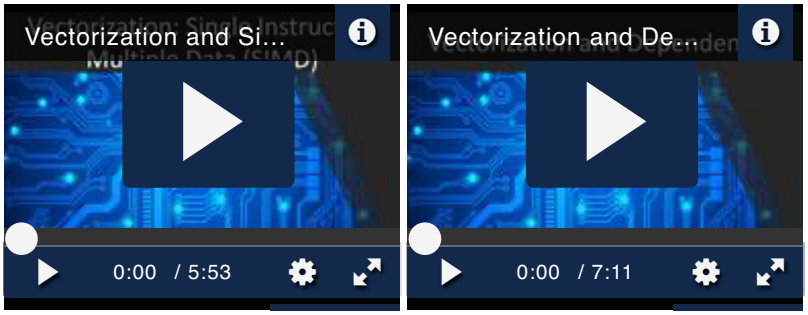
PRE23.1. 23 Text and Videos

Videos



Video credits: Geoffrey Herman, Text credits: Geoffrey Herman

The Big Picture

A statement T is said to be data dependent on statement S if

- S executes before T in the original sequential/scalar program
- S and T access the same data item
- At least one of the accesses is a write.

By this definition, we have three types of data dependencies

- True Dependence: A Read After Write (RAW) dependency
- Anti Dependence: A Write After Read (WAR) dependency
- Output Dependence: A Write After Write (WAW) dependency

When optimizing the performance of our code using various parallelism techniques, we can potentially change the order of execution of various statements in our programs. Dependencies indicate an execution order in our program that must be honored (i.e., if T is dependent on S, we cannot execute T before S). A true dependency is an integral part of the code and can not be removed. Anti-dependence and output dependencies are dependencies are called name dependencies because they can potentially be removed from

PRE 23

Assessment overview

Total points: 75/75

Score: 100%

Question PRE23.1

Value: 1

Total points: — /1

Auto-graded questior

Previous question

Next question

Personal Notes

No attached notes

Attach a file

Add text note

the code by changing the name of a variable.

Single-Instruction Multiple Data (SIMD) instructions take advantage of potential parallelism in our code by performing multiple loads or multiple arithmetic/logical operations with a single instruction. SIMD instructions use a separate datapath from the one we have discussed so far that has its own vector register file and its own vector ALU. In 32-bit Intel SSE processors, the vector registers store 128 bits and the vector ALU performs operations on 128 bits of data at a time. These vector registers/ALU can be parsed differently based on what data type we are working with. For example, if we were adding 1 to every element of an `int` (32 bit data type) array, we could load four elements (128/32 = 4) of the array at the same time, perform four integer additions at the same time, and store back to four elements of the array at the same time. In contrast, if we were adding 1 to every element of a `char` (8 bit data type) array, we could load 16 (128/8 = 16) elements of the array at the same time, perform 16 char additions at the same time, and store back to 16 elements of the array at the same time. We call this process, vectorizing our code.

By performing multiple loads, adds, or stores at the same time, we are changing the execution order of a program. For example, if our original program had a load, add, store, load, add, store, the vectorized code would effectively perform load, load, add, add, store, store. Consequently, when vectorizing our code, we need to make sure that this reordering does not change the order of any dependencies.

Mark as read

- [ ] (a) I've read this!

Select all possible options that apply.

Save & Grade *Unlimited attempts* Save only