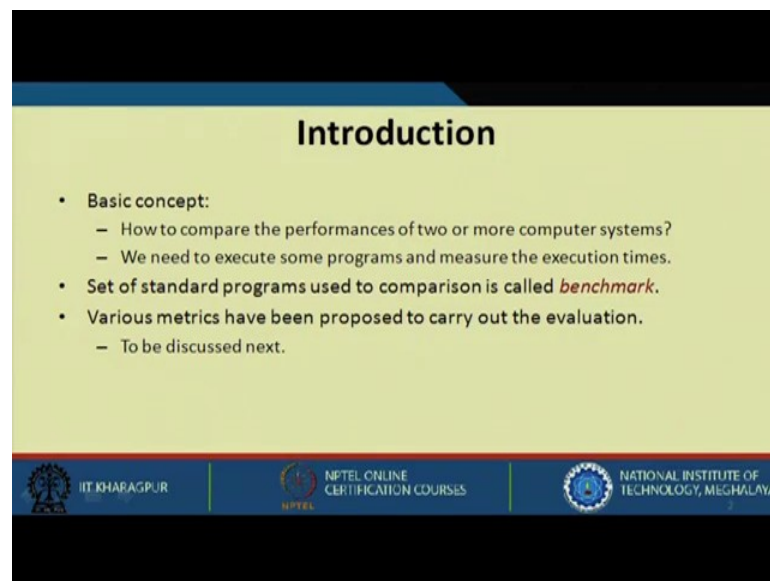


**Computer Architecture and Organization**  
**Prof. Kamalika Datta**  
**Department of Computer Science and Engineering**  
**National Institute of Technology, Meghalaya**

**Lecture – 13**  
**Choice of Benchmarks**


Welcome to the next lecture. So, in this lecture we will be seeing the choice of benchmark. So, in the previous lecture what we have said is that how do we compute the execution time of a program. And then we know that these are the ways through which you execute you can compute the execution time of the program. Now still how you can say that this computer is better than this? Now comes choice of benchmark; that means, what you will be using which benchmarks to use such that by executing this set of instruction we can compare performance.

(Refer Slide Time: 01:10)



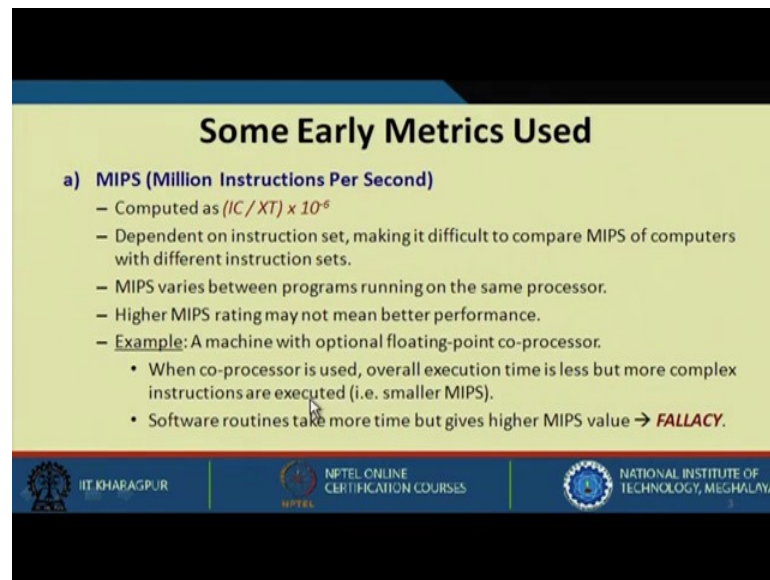
**Introduction**

- **Basic concept:**
  - How to compare the performances of two or more computer systems?
  - We need to execute some programs and measure the execution times.
- **Set of standard programs used to comparison is called *benchmark*.**
- **Various metrics have been proposed to carry out the evaluation.**
  - To be discussed next.



So, basically here the basic concept is how to compare the performances of two or more computers. This is what we are discussing from the last lecture as well. So, we need to execute some programs and measure the execution times. Set of standard programs are used for this for this comparison and we call this as benchmarks. So, benchmarks are nothing but some set of programs, which are set as benchmarks basically for this comparison. And various metrics have been proposed to carry out the evaluation we will be discussing that next.

(Refer Slide Time: 01:54)



**Some Early Metrics Used**

a) **MIPS (Million Instructions Per Second)**

- Computed as  $(IC / XT) \times 10^6$
- Dependent on instruction set, making it difficult to compare MIPS of computers with different instruction sets.
- MIPS varies between programs running on the same processor.
- Higher MIPS rating may not mean better performance.
- **Example:** A machine with optional floating-point co-processor.
  - When co-processor is used, overall execution time is less but more complex instructions are executed (i.e. smaller MIPS).
  - Software routines take more time but gives higher MIPS value → **FALLACY.**

Logos: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

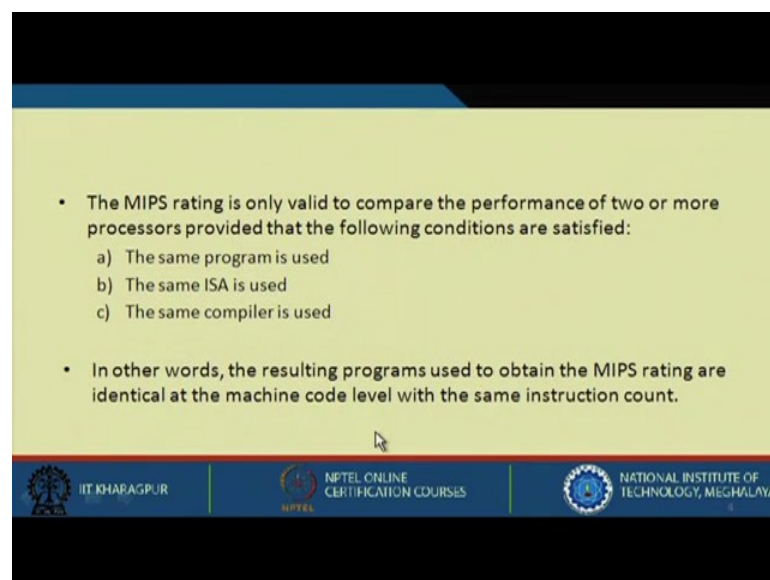
So, let us see some early metrics that are used. One is millions instructions per second (MIPS); this is computed as instruction count divided by execution time into  $10^6$ . So, this is dependent on what. So, basically we are trying to get how many millions of instructions that are executed per second. This is dependent on instruction set, which makes it difficult to compare MIPS of various computers with different instruction set, because see different computers will have different instruction set different kind of architectures, but then how we can say that this can be a metric that can be used to evaluate for all. So, this becomes difficult.

So, this is dependent on the instruction set. MIPS also varies between programs running on the same processor; why does this varies? Because different compilers will generate different codes; suppose say you have run a particular program and it has generated some set of a codes; the same program can be run on another compiler and may generate little different code. So, at that point of the time also this MIPS will be different and also it has been observed that higher MIPS rating may not mean better performance. So, we cannot say that if the MIPS rating is high, that means it performs much better. Let us take an example a machine with optional floating-point coprocessor.

So, when aco processor, meaning is we have a machine with an optional floating-point coprocessor. So, when coprocessor is used overall execution time will be less because you are using a coprocessor in which the task will can be performed in a much faster

fashion. So, in turn your execution time will become less, but for doing you may use some complex instructions. So, if you use complex instructions then it will give you a smaller MIPS value. So, when you use a coprocessor your overall execution time becomes less, but you are using more complex instruction for execution. That is the MIPS will be much less. Same way for a software routine it takes more time, but it is giving higher MIPS value why because they will be more number of instruction that are getting executed, but in turn the time using a software routine will be much more. So, this is a fallacy this is a problem here. So, we are using a coprocessor which is making the entire process faster, but still we are getting smaller MIPS, but another which is using a software routine which is getting higher MIPS, but at the same time it takes more time as well.

(Refer Slide Time: 05:38)



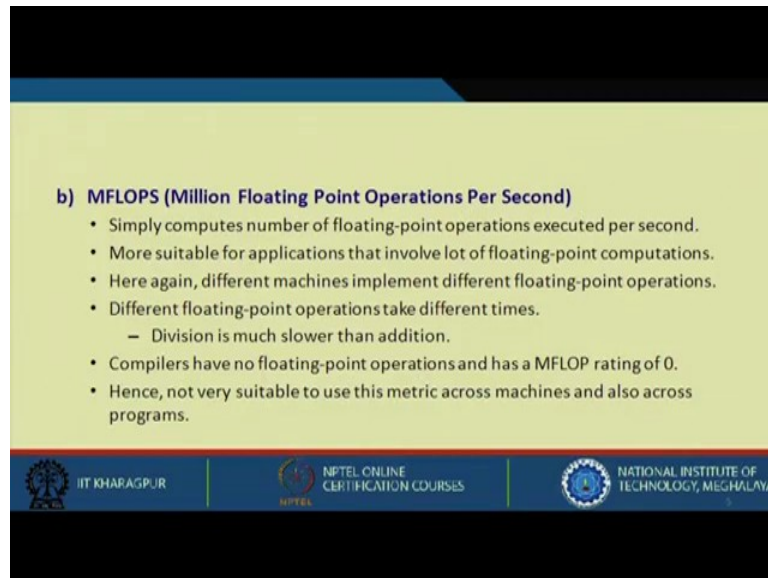
- The MIPS rating is only valid to compare the performance of two or more processors provided that the following conditions are satisfied:
  - a) The same program is used
  - b) The same ISA is used
  - c) The same compiler is used
- In other words, the resulting programs used to obtain the MIPS rating are identical at the machine code level with the same instruction count.

So, MIPS rating is only valid to compare the performance of 2 or more processors provided that the following conditions are satisfied. What are the factors? First one is the same program is used. The same instruction set architectures, used set of instruction should be same and the same compiler is used. If you have all these things together, then only we can say that MIPS rating can be taken for performance comparison.

So, in other words we can say that the resulting programs used to obtain the MIPS rating are identical at the machine code level with the same instruction count. You must have

same instruction count you must have same those machine code level instructions, then only you can say that you can use MIPS as a metric to evaluate the performance.

(Refer Slide Time: 06:53)



The slide features a light green background with a dark blue header and footer. The main content is a bulleted list under the heading 'b) MFLOPS (Million Floating Point Operations Per Second)'. The footer contains three logos: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA.

**b) MFLOPS (Million Floating Point Operations Per Second)**

- Simply computes number of floating-point operations executed per second.
- More suitable for applications that involve lot of floating-point computations.
- Here again, different machines implement different floating-point operations.
- Different floating-point operations take different times.
  - Division is much slower than addition.
- Compilers have no floating-point operations and has a MFLOP rating of 0.
- Hence, not very suitable to use this metric across machines and also across programs.

This is million floating point operations per second (MFLOPS). So, it simply computes the number of floating point operations executed per second. Now this obviously, will be more suitable for certain applications where we will be using floating point computation. Let us say for certain application there are not so much floating point instructions.

So, in that MFLOPS will be much less, but that does not mean that the performance of that is poor. So, here again different machines implement different floating point operations. Different floating point operation takes different times. Addition of a floating point will take less time may be compared to the division of a floating point.

So, we cannot really say that how well MFLOPS as a metric will give you a correct performance evaluation. Compilers have no floating point operation and has MFLOPS rating as 0. Hence this is not very suitable metric across machine and across programs. MFLOPS cannot be used as a metric across any machines or across any programs because different machines have different features different characteristics. So, it might not be a good idea to rely upon a metric like MFLOP.




(Refer Slide Time: 08:53)

**Example 1**

- Consider a processor with three instruction classes A, B and C, with the corresponding CPI values being 1, 2 and 3 respectively. The processor runs at a clock rate of 1 GHz. For a given program written in C, two compilers produce the following executed instruction counts.

	Instruction Count (in millions)		
	For IC <sub>A</sub>	For IC <sub>B</sub>	For IC <sub>C</sub>
Compiler 1	7	2	1
Compiler 2	12	1	1

Compute the MIPS rating and the CPU time for the two program versions.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES |  NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us take an example. Consider a processor with three instruction classes A, B and C with the corresponding CPI values being 1, 2 and 3 respectively. The processor runs at a clock rate of 1 GHz. So, for a given program written in C, two compilers produce the following executed instruction counts.

So, instruction count for A type instruction is 7 for compiler 1, 2 for type B, and 1 for C type C classes. Similarly, for compiler 2 the number of instruction count for A type is 12, B type is 1 and C type is 1. Let us see how do we compute the MIPS rating and the CPU time for the two program versions. So, we have been given with the CPI values for the various types of instruction that is A, B and C as 1, 2 and 3 respectively and the processor runs at a clock rate of 1 GHz. So, these are the parameters that are already given.

(Refer Slide Time: 10:21)

MIPS = Clock Rate (MHz) / CPI      CPI = CPU Execution Cycles / Instruction Count

CPU Time = Instruction Count x CPI / Clock Rate

• Solution:

– **For compiler 1:**

$$CPI_1 = (7 \times 1 + 2 \times 2 + 1 \times 3) / (7 + 2 + 1) = 14 / 10 = 1.40$$
$$MIPS\ Rating_1 = 1000\ MHz / 1.40 = 714.3\ MIPS$$
$$CPU\ Time_1 = ((7 + 2 + 1) \times 10^6 \times 1.40) / (1 \times 10^9) = 0.014\ sec$$

– **For compiler 2:**

$$CPI_2 = (12 \times 1 + 1 \times 2 + 1 \times 3) / (12 + 1 + 1) = 17 / 14 = 1.21$$
$$MIPS\ Rating_2 = 1000\ MHz / 1.21 = 826.4\ MIPS$$
$$CPU\ Time_2 = ((12 + 1 + 1) \times 10^6 \times 1.21) / (1 \times 10^9) = 0.017\ sec$$

*MIPS rating indicates that compiler 2 is faster, while in reality the reverse is true*

IIT KHARAGPUR      NPTEL ONLINE CERTIFICATION COURSES

Let us see how we will calculate the MIPS rating and the CPU time. So, MIPS is clock rate in MHz divided by CPI. And CPI is CPU execution cycle divided by instruction count. And CPU time will be instruction count multiplied by CPI divided by clock rate or multiplied by clock period. Let us say for compiler 1, 7 is the total number of instruction type A which is multiplied with 1 that is the CPI for that particular type A instruction. Similarly, 2 multiplied by 2. So, basically we are doing 7 multiplied by 1, 2 multiplied by 2, and 3 multiplied by 1 one for compiler 1. So, 7 multiplied by 1, 2 multiplied by 2, and 1 multiplied by 3 divided by total number of instruction. Total number of instruction was  $7 + 2 + 1 = 10$ . So,  $14 / 10$  which is coming to 1.40.

Similarly, MIPS rating will be 1000 MHz divided by 1.40. So, we converted it to MHz because we have to find out in terms of MIPS. So, that is 1000 divided by 1.40 that is coming to 714.3. Now what is the CPU time? The CPU time can be calculated by  $7 + 2 + 1$ . So, we get the time as 0.014 seconds. So, this much second it is taking to execute for compiler 1 and MIPS rating for this is 714.3.

Let us take for the next compiler in the similar fashion we compute the CPI  $12 + 1 + 1$  divided by total number of instruction. And we get 1.21 as the CPI similarly MIPS rating can be find out by 1000 MHz divided by 1.21, that comes to 826.4 MIPS. And similarly for CPU time we will use the same instruction count multiplied by CPI divided by clock rate, which is coming down to 0.017.



So, now you see that the MIPS of this is higher. So, it has got higher millions instruction per second, but the execution time of compiler 1 is less. So, here you can clearly see that the execution time of compiler 1 is less, but the MIPS of compiler 2 is more. So, MIPS cannot be the right choice for in such cases. So, MIPS rating indicates that compile it 2 is faster while in reality the reverse is true.

(Refer Slide Time: 13:57)

### Example 2

A loop in C

```
for (k=0; k<1000; k++)
{
  A[k] = A[k] + s;
}
```


```


Loop: LW   $t3, 0($t1)
      ADDI $t6, $t2, 4000
      LW   $t4, 0($t3)
      ADD  $t5, $t4, $t3
      SW   $t5, 0($t2)
      ADDI $t2, $t2, 4
      BNE $t6, $t2, Loop
```


MIPS32 Code

- The code is executed on a processor that runs at 1 GHz (C = 1 nsec).
- There are four instruction types with CPI values as shown in the table.
- We show some calculations next.

Instruction Type	CPI
ALU	2
LOAD	5
STORE	6
BRANCH	3

 IIT KHARAGPUR

 NPTEL ONLINE CERTIFICATION COURSES

 NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, let us take an example. So, this is a C loop. What we are doing inside the C loop, we are simply adding a constant value stored in variable s to A[k], and we are storing back in A[k].

Similarly, for the next one and this is going on in a loop let us write the assembly language code for this particular C code segment. So, these are the few things you have to consider. \$t1 stores the address of s, s is a variable which is a constant some value is stored here, and \$t3 stores the value of s and \$t2 points to the first location of this array of this particular array. So, here initially what we are doing we are loading the word from this location 0 of \$t1, \$t1 is having the address of s. So, value of s is stored in \$t3. We are adding an immediate value to \$t2, \$t2 points to the first location of the array. And we have to compute something for this thousand times. So, we are multiplying 4000; we are adding 4000 to \$t2 because each are 4, 4, 4 by, so, 4 multiplied by 1000. So, which is coming down to we are adding it to \$t2 and we are storing it in \$t6. So, \$t6 stores the final value. So, we have to go till that value to execute it.




Next word inside the loop these are the following statement, that are getting executed first what we are doing we are loading the word from the first location of the array that is A[0]. We are storing it in \$t4. Then what we are doing the value of s is stored in \$t3 and the array value is stored in \$t4. So, \$t3 and \$t4 we have to add and we are storing it in \$t5. So, finally, we are adding \$t4 and \$t3 and we are storing it in \$t5 and finally, we are storing back this \$t5 the added value in 0 of \$t2. So, in that location we are again storing it back. And finally, what we have to do we need to increment to the next location. So, the first part is done now we are moving to the next location. So, for the next location it is added with 4 again and then it is transferred there.

Now, branch if not equal we are doing such that whether we have reached to that point or not, \$t6 is equal to \$t2 because at every point we are adding 4 to it. So, when it reaches the last element it will come out of the loop, when till it is not equal \$t6 is not equal to \$t2 is not equal to \$t6 we will loop, when it is equal it will come out of the loop. So, these are the following assembly language code that we are executing for this set of codes the code is executed on a processor that runs at 1 GHz that is the clock period is one nanosecond, there are 4 instruction types with CPI values are shown in this table.

Now, see ALU operation which are ADDI; these are ALU operation. And the CPI of those operations is 2. Similarly, you have load. Load is LW the CPI is 5, you have SW the CPI is 6, and you have a BNE type of instruction where the CPI is 3.

(Refer Slide Time: 18:49)

- The code has 2 instructions before the loop and 5 instructions in the body of the loop that executes 1000 times.
  - Total instruction count  $IC = 5 \times 1000 + 2 = 5002$ .
- Number of instructions executed and fraction  $F_i$  for each instruction type:
  - $IC_{ALU} = 1 + 2 \times 1000 = 2001$ ,  $F_{ALU} = 2001 / 5002 = 0.4 = 40\%$
  - $IC_{LOAD} = 1 + 1 \times 1000 = 1001$ ,  $F_{LOAD} = 1001 / 5002 = 0.2 = 20\%$
  - $IC_{STORE} = 1000$ ,  $F_{STORE} = 1000 / 5002 = 0.2 = 20\%$
  - $IC_{BRANCH} = 1000$ ,  $F_{BRANCH} = 1000 / 5002 = 0.0 = 20\%$
- Total CPU clock cycles =  $2001 \times 2 + 1001 \times 5 + 1000 \times 6 + 1000 \times 3 = 18,007$  cycles
- Average CPI = CPU clock cycles / IC =  $18007 / 5002 = 3.6$
- Execution time = IC x CPI x C =  $5002 \times 3.6 \times 1 \times 10^{-9} = 18.0 \mu\text{sec}$

 IIT KHARAGPUR
  NPTEL ONLINE CERTIFICATION COURSES
  NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA



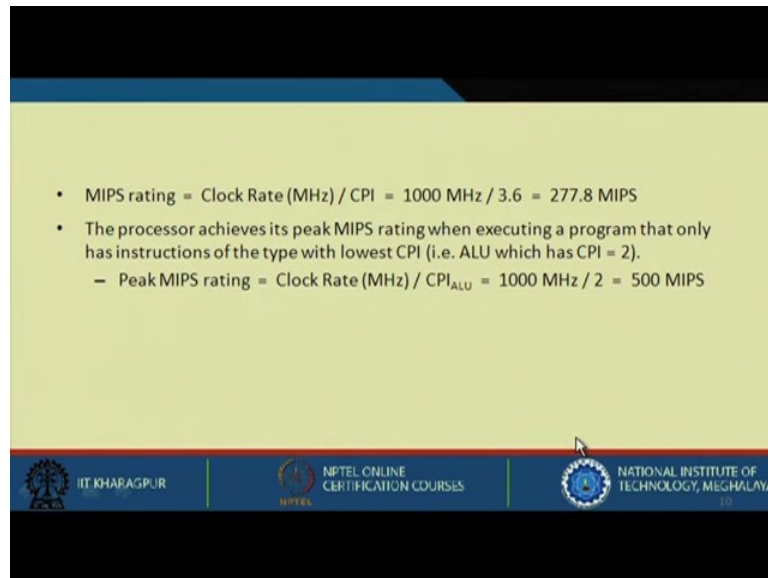
Now, let us see the code has two instructions before the loop and 5 instructions in the body of the loop that executes 1000 times. So, outside the body of the loop you see you have two instructions. And inside you have 5 instructions 1 2 3 4 5 and each of these instructions is executed 1000 times because this loop is executing 1000 times. So, what will be the total instruction count? There are 5 instructions and each instruction executes 1000 times. So, 5000 and 2 instructions outside the loop plus 2 it will become 5002. 2 number of instructions executed and fraction  $F_i$  for each instruction type. So, let us calculate this number of instruction executed and fraction of instruction  $F_i$  for each instruction type.

Let us first calculate the total number of instruction. So, outside the loop there is one ALU instruction and inside the loop there are 2 ALU instructions. So, these 2 ALU instructions each will be executed 1000 times. And this instruction will get executed one time. So, inside the loop there are 2 ALU instructions that are executed 1000 times, and this is executed one more time. So, 2001, similarly you can calculate for all load store and branch store and branch are only 1, 1 instructions are there this is store and this is from this is executed thousand time this is executed 1000 times.

So, what is the frequency --- total number of instruction of such kind divided by the total number of instructions, which is coming to 0.4, that is 40%. This is coming 20%, this is coming 20%, and this is coming 20%. Now how do we calculate this is the frequency of a loop operation, this is the frequency of load type and so on. So, total CPU clock cycles are  $2001 \times 2$ ,  $1001 \times 5$ ,  $1000 \times 6$ ,  $1000 \times 3$ . So, we are taking all this from this CPI we are multiplying the CPI with the total number of instruction that we have found out previously and we are getting the total cycles as 18007. So, this is the total CPU clock cycle divided by instruction count you get the CPU as 3.6.

Now, you can calculate the total execution time which is IC which is total instruction which is 5002 CPI that we have calculated, and this is the clock period which is coming to 80 microsecond this is how we calculate it.

(Refer Slide Time: 22:03)



• MIPS rating = Clock Rate (MHz) / CPI = 1000 MHz / 3.6 = 277.8 MIPS

• The processor achieves its peak MIPS rating when executing a program that only has instructions of the type with lowest CPI (i.e. ALU which has CPI = 2).

- Peak MIPS rating = Clock Rate (MHz) / CPI<sub>ALU</sub> = 1000 MHz / 2 = 500 MIPS

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, how do you see the clock rating? Clock rating will be clock rate divided by CPU that is coming to 277.8 MIPS. So, the processor achieved its peak MIPS rating when executing a program that only has instructions of type with lower CPU CPI that is ALU type instruction. So, if you only execute such kind of instruction where the CPI is less that is you see the CPI of ALU type is only 2, but for store load branch is moved.

Now, if you only execute ALU, which only those type of instruction where CPI is less then you can get the peak MIPS rating that is coming to 500 MIPS, but if you use with this mixture where the CPI is found out by calculating taking into consideration all the types of instruction and the frequency at which all these instructions are occurring then it will be coming to something lesser MIPS.

(Refer Slide Time: 23:16)

**Choosing Programs for Benchmarking**

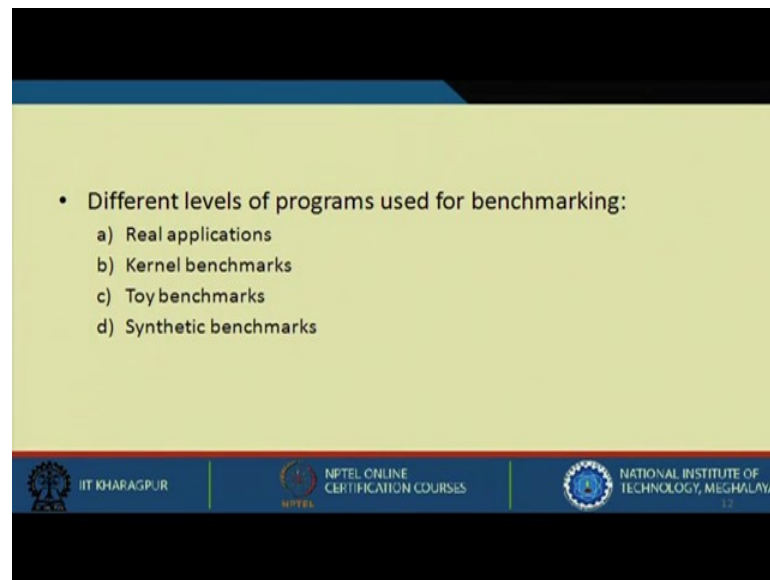
- Suppose you are trying to buy a new computer and you have several alternatives.
  - How to decide which one will be best for you?
- The best way to evaluate is to run the actual applications that you are expected to run (*actual target workload*), and find out which computer runs them the fastest.
  - Not possible for everyone to do this.
  - We often rely on other methods that are *standardized* to give us a good measure of performance.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, next let us see choosing programs for benchmarking. Now how do we choose programs for this benchmarking? Suppose we are trying to buy a new computer and there are several alternatives possible. So, how to decide upon which one is the best. The best way that is that can be used is to run the actual application that you are expected to run that is the actual target workload; that means, that particular computer you will be using more floating point operation. So, you should have such kind of program in place that you will run on that machine and you will see that what is the performance coming.

So, actually you are running certain kind applications that will give you the best result. So, choosing the programs for benchmarking is really very important, but not possible for everyone to do this while purchasing. So, what we do we often rely on the methods that are standardized to give us a good measure of performance. So, there are some standardized methods that are used which can be considered as a good measure for this performance.

(Refer Slide Time: 24:30)



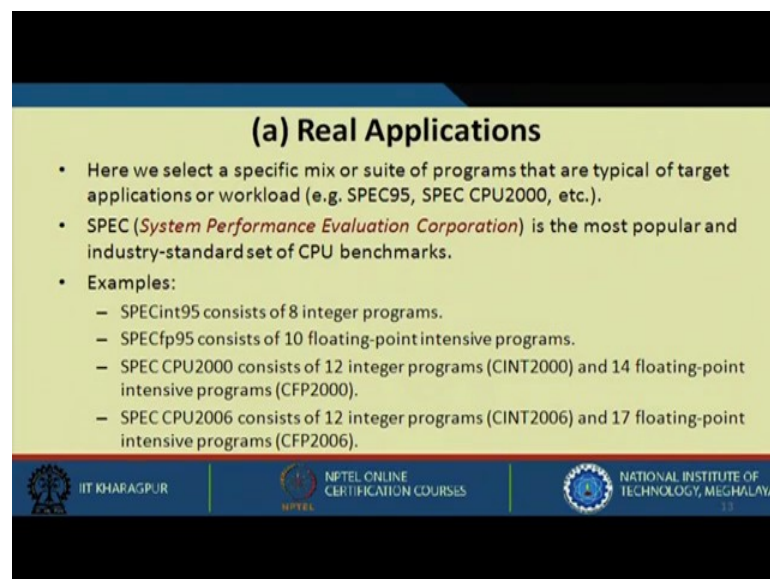
• Different levels of programs used for benchmarking:

- a) Real applications
- b) Kernel benchmarks
- c) Toy benchmarks
- d) Synthetic benchmarks

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

So, different levels of programs are used for benchmarking --- one is real application, can be kernel benchmarks, some toy benchmarks, and some synthetic benchmarks. Let us see an overview of all these things.

(Refer Slide Time: 24:45)



**(a) Real Applications**

- Here we select a specific mix or suite of programs that are typical of target applications or workload (e.g. SPEC95, SPEC CPU2000, etc.).
- SPEC (*System Performance Evaluation Corporation*) is the most popular and industry-standard set of CPU benchmarks.
- Examples:
  - SPECint95 consists of 8 integer programs.
  - SPECfp95 consists of 10 floating-point intensive programs.
  - SPEC CPU2000 consists of 12 integer programs (CINT2000) and 14 floating-point intensive programs (CFP2000).
  - SPEC CPU2006 consists of 12 integer programs (CINT2006) and 17 floating-point intensive programs (CFP2006).

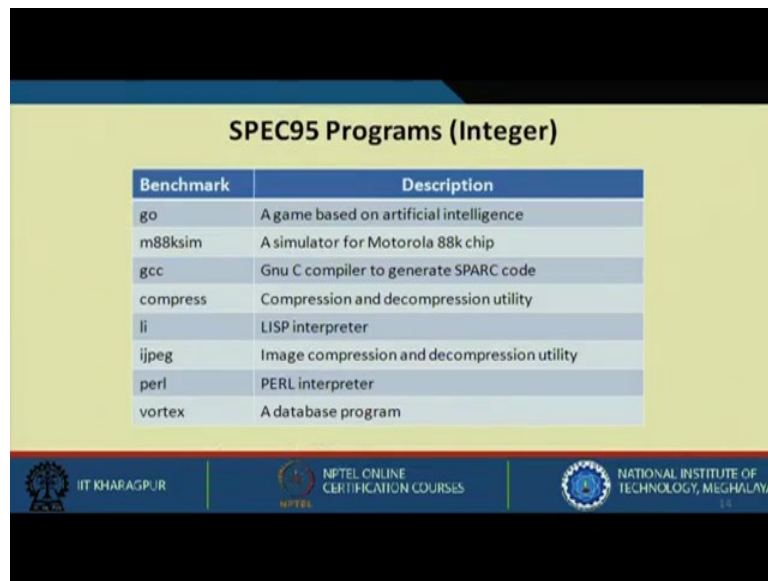
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

What are real applications? We select a specific mix of suit of programs that are typical of large application or workload. Some of the examples are SPEC95, CPU2000, etc. SPEC stands for System Performance Evaluation Corporation and this is the most popular and industry standard set of CPU benchmarks. So, SPECint95 consists of 8

integer programs. SPECfp95 consists of 10 floating-point intensive programs. SPEC CPU2000 consists of 12 integer programs and 14 floating-point intensive programs, and SPEC CPU2006 consists of 12 integer programs.

So, as we are moving from 95 to 2000 to 2006 the numbers are increasing. So, we are putting more workload because their advancement in these clock speed is increasing. So, we can actually perform more operations. So, that is how it is moving.

(Refer Slide Time: 26:12)



Benchmark	Description
go	A game based on artificial intelligence
m88ksim	A simulator for Motorola 88k chip
gcc	Gnu C compiler to generate SPARC code
compress	Compression and decompression utility
li	LISP interpreter
jpeg	Image compression and decompression utility
perl	PERL interpreter
vortex	A database program

The slide also features logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA at the bottom.

So, these are SPEC95 programs (integer) --- what all kind of programs are present; a game based on artificial intelligence, a simulator for motorola 88k chip, a gnu compiler, compression and decompression utility, lisp interpreter, image compression and decompression utility, perl interpreter, a database program. So, the SPEC95 program consists of these following benchmarks.

(Refer Slide Time: 26:47)

SPEC95 Programs (Floating-Point)	Benchmark	Description
	tomcatv	A mesh generation program
swim	Shallow water modeling	
su2cor	Quantum physics Monte Carlo simulation	
hydro2d	Solving hydrodynamic Navier Stokes equations	
mgrid	Multigrid solver on 3D potential field	
applu	Solving parabolic/elliptical differential equations	
trub3d	Simulates turbulence in a cube	
apsi	Solver for distribution of pollutant	
fpppp	Quantum chemistry simulation	
wave5	Simulation of plasma physics	

Similarly, SPEC95 programs consist of these programs. And these are all these SPEC95 programs of floating point programs. So, they have a mesh generation program, shallow water modeling, quantum physics, Monte Carlo simulation, solving hydrodynamic Navier Stokes equation, multi grid solver on 3D potential field, quantum chemistry simulation and so on. So, these SPEC95 programs were having these programs.

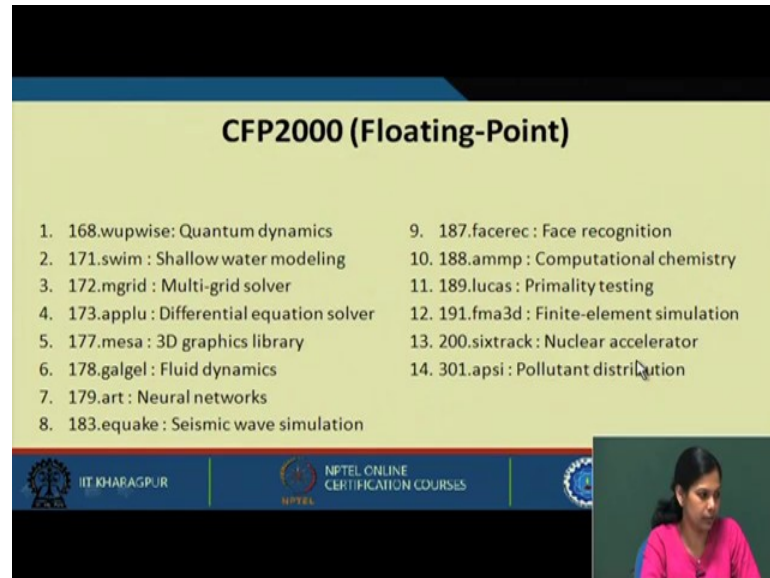
(Refer Slide Time: 27:32)

CINT2000 (Integer)	
1. 164.gzip : compression	9. 254.gap : Group theory interpreter
2. 175.vpr : FPGA placement / routing	10. 255.vortex : Object-oriented database
3. 176.gcc : C compiler	11. 256.bzip2 : Compression
4. 181.mcf : Combinatorial optimization	12. 300.twolf : VLSI Place / Route
5. 186.crafty : Chess playing	
6. 197.parser : Word processing	
7. 252.con : Computer visualization	
8. 253.perlbnk : PERL interpreter	



Similarly, CINT2000 (integer) consists of these following programs. So, these are the 12 programs. They added something new which is VLSI place and route. This group theory interpreter was also added which was not there previously in SPEC95.

(Refer Slide Time: 27:55)



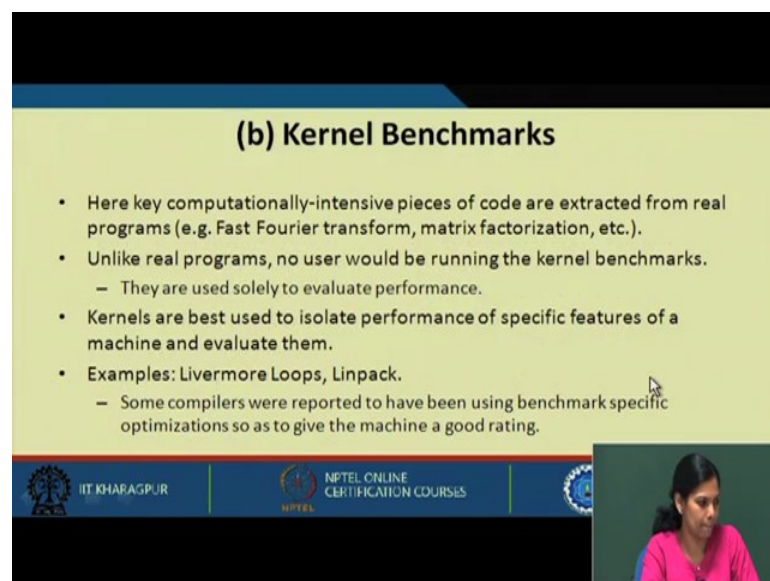
**CFP2000 (Floating-Point)**

1. 168.wupwise: Quantum dynamics	9. 187.facerec : Face recognition
2. 171.swim : Shallow water modeling	10. 188.amp : Computational chemistry
3. 172.mgrid : Multi-grid solver	11. 189.lucas : Primality testing
4. 173.applu : Differential equation solver	12. 191.fma3d : Finite-element simulation
5. 177.mesa : 3D graphics library	13. 200.sixtrack : Nuclear accelerator
6. 178.galgel : Fluid dynamics	14. 301.apsi : Pollutant distribution
7. 179.art : Neural networks	
8. 183.quake : Seismic wave simulation	

The slide includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL. A small video inset shows a woman in a pink shirt.

So, these are some of the programs of CFP2000. So, it has got quantum dynamics these are already there neural networks were added pollutant distribution is added nuclear accelerator is added and many more.

(Refer Slide Time: 28:16)



**(b) Kernel Benchmarks**

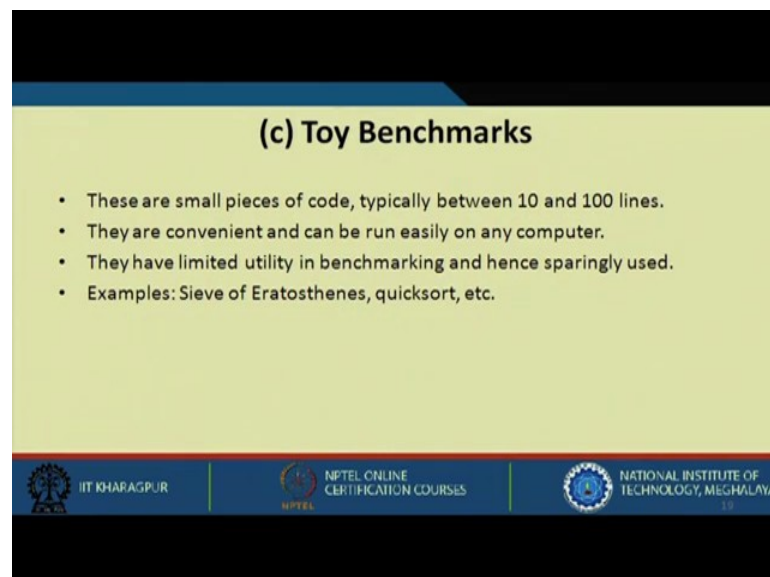
- Here key computationally-intensive pieces of code are extracted from real programs (e.g. Fast Fourier transform, matrix factorization, etc.).
- Unlike real programs, no user would be running the kernel benchmarks.
  - They are used solely to evaluate performance.
- Kernels are best used to isolate performance of specific features of a machine and evaluate them.
- Examples: Livermore Loops, Linpack.
  - Some compilers were reported to have been using benchmark specific optimizations so as to give the machine a good rating.

The slide includes logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NPTEL. A small video inset shows a woman in a pink shirt.

Now, let us see what kernel benchmark is. Here what happens basically is that key computationally intensive pieces of code are extracted from real programs. So, let us say there is part of the program, where the computation requirement is moved. So, they take out those part of the program from there and what they do unlike real programs no user would be running the kernel benchmarks they are solely used to evaluate performance. This is just used to evaluate the performance and as we know that kernels are also best to isolate performance of specific features of a machine and evaluate them some of the examples are Livermore loops, LINPAC, etc. And some compilers were reported to have been using benchmark specific optimizations. So, as to give the machine a good rating; that means, let us say we have so many benchmarks now.

So, now these are already available and you can use this to evaluate your performance. So, some compilers typically use some of those features to accelerate the speed of those programs only, but it may not work for any application it may work for specifically for those applications, but if you are not using such kind of constructs that are used in those programs then you will not be getting better result.

(Refer Slide Time: 30:10)



**(c) Toy Benchmarks**

- These are small pieces of code, typically between 10 and 100 lines.
- They are convenient and can be run easily on any computer.
- They have limited utility in benchmarking and hence sparingly used.
- Examples: Sieve of Eratosthenes, quicksort, etc.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

These are some toy benchmarks are also use the code typically between 10 to 100 lines, and they are convenient and can be run easily on any computer. They have limited utility in benchmarking and hence sparingly used.

(Refer Slide Time: 30:33)

**(d) Synthetic Benchmarks**

- Somewhat similar in principle to kernel benchmarks.
  - They try to match the average frequency of operations and operands of a large set of programs.
- Synthetic benchmarks are further removed from reality than kernels, as kernel code is extracted from real programs, while synthetic code is created artificially to match an average execution profile.
- Examples: Whetstone, Dhrystone, etc.
  - These are not real programs.
- Some drawbacks with synthetic benchmarks are discussed next.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, coming to synthetic benchmarks, what do you mean by synthetic? We know that this particular machine has this much type of ALU operation, this much type of store and load operation, etc. So, some synthetic benchmarks are generated, which will actually resemble to that particular frequency of operation which is performed for certain programs, but they are synthetic as they are not real benchmarks basically.

So, somewhat similar to the principles to kernel benchmarking, they try to match the average frequency of operations and operands of a large program. Just now what I have said let us say we have a program and we know that for this program 80% will be such kind of instruction ALU operation and 20% will be store-load operation. So, we also generate a particular program such that it uses same kind of features 80% will be ALU and 20% will be other. Synthetic benchmarks are further removed from reality than kernels, as kernel code is extracted from real programs while synthetic code is created artificially to match an average execution profile. So, these are made artificially to match an average execution profile. Some of the examples are Whetstone and Dhrystone; these are not real programs.

So, we came to end of lecture 13. So, where we have seen that what the choice of the benchmark, how do you choose a particular benchmark. So, that entirely depends on the application for which you are designing or you require the CPU for what kind of application.

Thank you.