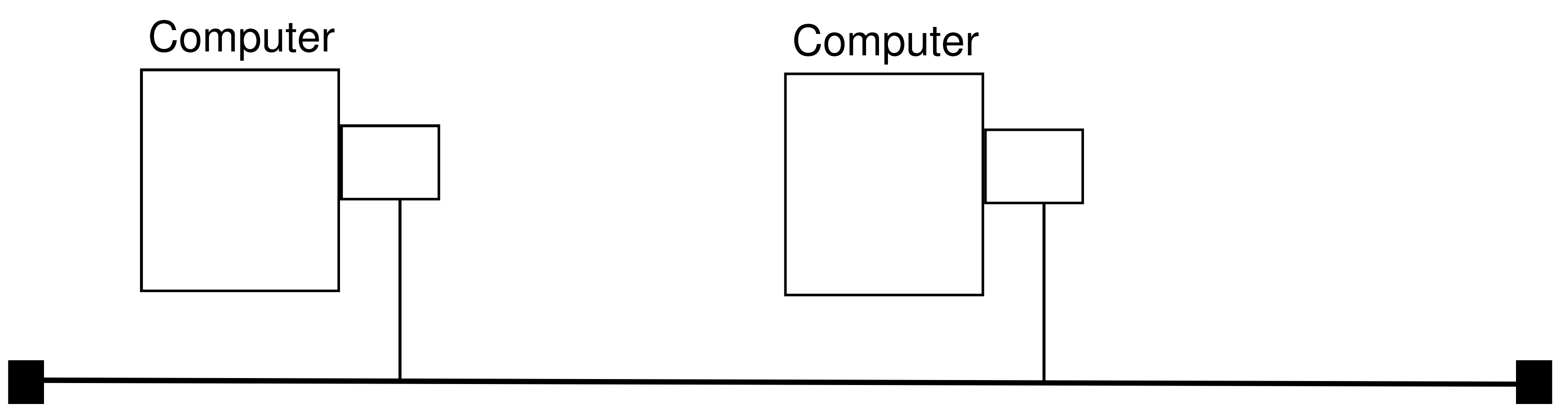
Announcements

- Project #6 is due Thursday at 5:00 PM
- Course Evaluations
  - Please fill them out!
- Final is Sat 8:00 – 10:00 am
  - This room
- Extra Office hours
  - W 1:30-2:30

Ethernet

- 10 Mbps (to 100 Gbps)
- mili-second latency
- limited to several kilometers in distance
- variable sized units of transmission
- Conceptually a bus based protocol
  - requests to use the network can collide
- addresses are 48 bits
  - unique to each interface



Switched Ethernet

- Logically it is still a bus
- Physically, it is a star configuration
  – the hub is at the center of the network
- Switches provide:
  – better control of hosts
    - possible to restrict traffic to only the desired target
    - can shutdown a host's connection at the hub if its Ethernet device is misbehaving
  – easier wiring
    - can use twisted pair wiring
- 100 Mbps/1Gbps Ethernet
  – is only available with switches
- 10Gbps Ethernet
  – Requires cat-6 (to 100 feet) or cat-7 wiring (to 100 meters)

Ethernet Collisions

- If one host is sending, other hosts must wait
  - called Carrier Sense with Multiple Access (CSMA)
- Possible for two hosts to try to send at once
  - each host can detect this event (cd- Collision Detection)
  - both hosts must re-send information
    - if they both try immediately, will collide again
    - instead each waits a random interval then tries again
- Only provides statistical guarantee of transmission
  - however, the probability of success if higher than the probability of hardware failures and other events

My Research Interests

- Parallel Computing
  - There are limits to how fast one processor can run
  - solution: use more than one processor
- Issues in parallel computing design
  - do the processors share memory?
    - is the memory "uniform"?
    - how do processors cache memory?
  - if not how do they communicate?
    - message passing
    - what is the latency of message passing

Parallel Processing

- What happens in parallel?
- Several different processing steps
  - pipeline
  - simple example: grep foo | sort > out
  - called: *multiple instruction multiple data* (MIMD)
- The same operation
  - every processor runs the same instruction (or no-instruction)
  - called: *single instruction multiple data* (SIMD)
  - good for image processing
- The same program
  - every processor runs the same program, but not "lock step"
  - called: *single program multiple data* (SPMD)
  - most common model

Issues in effective Parallel Computation

- Getting enough parallelism
  - Limited by what is left serial
  - Even 10% serial limited to a speedup of 10x even with infinite numbers of processors
- Load balancing
  - every processor should to have some work to do.
- Latency hiding/avoidance
  - getting data from other processors (or other disks) is slow
  - need to either:
    - hide the latency
      - processes can "pre-fetch" data before they need it
      - block and do something else while waiting
    - avoid the latency
      - use local memory (or cache)
      - use local disk (of file buffer cache)
- Limit communication bandwidth
  - use local data
  - use "near" data (i.e. neighbors)

My Research:

- Given a parallel program and a machine
- Try to answer performance related questions
  - Why is the programming running so slowly?
  - How do I fix it?
- Issues:
  - how to measure a program without changing it?
  - how do you find (and then present) the performance problem, not tons of statistics?
- Techniques:
  - dynamic data collection
  - automated search
  - analysis of process interactions

Large Scale Computing

- Today (5/2017)
  - 44 systems with more than 128k processors
  - More than 429 systems >= 16k processors
  - World's fastest computer (Sunway TaihuLight in China)
    - 10,649,600 cores
    - Uses 15.37 MW of electricity
    - Smallest core count of top500 – 5,904

Improving Code by Running It

- Auto-tuning: key Idea:
  - Automated cycle of: measure and actuate change
  - As program runs, it (hopefully) gets faster
- Why:
  - Many parameters impact performance
  - Optimal performance for a given system depends on:
    - Details of the processor
    - Details of the inputs (workload)
    - Which nodes are assigned to the program
    - Other things running on the system
  - Tuning these parameters by hand is tedious and slow

Auto-tuning Motivation

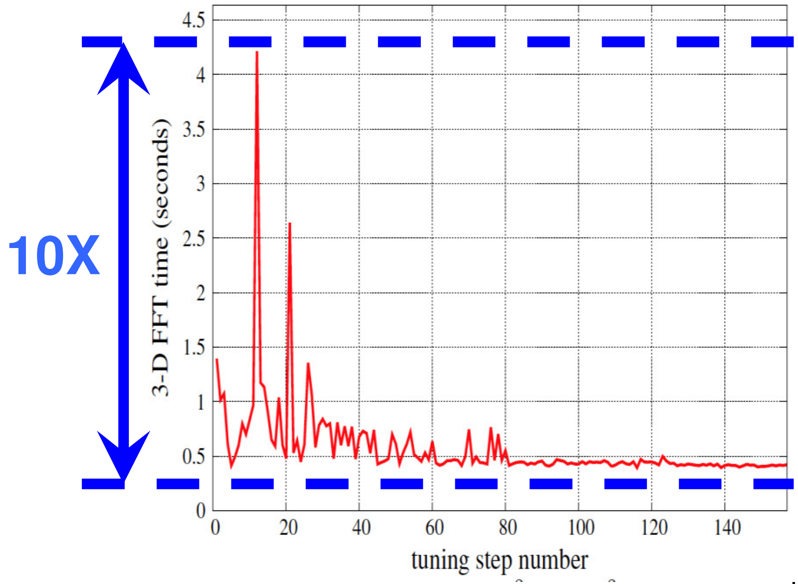
- Example, a dense matrix multiple kernel
- Various Options:
  - Original program: 30.1 sec
  - Hand Tuned (by developer): 11.4 sec
  - Auto-tuned of hand-tuned: 15.9 sec
  - Auto-tuned original program: 8.5 sec
- What Happened?
  - Hand tuning prevented analysis
    - Auto-tuned transformations were then not possible

What is online auto-tuning?

- Extreme late binding of decisions about:
  - Compiler optimizations
  - Algorithms
  - Library parameters
  - Applications parameters
  - Hardware?
- Reacting to a changing world
  - Hardware problems
  - Properties of data sets
- Changing anything at runtime that
  - Changes performance
  - Doesn't not change answer (output)

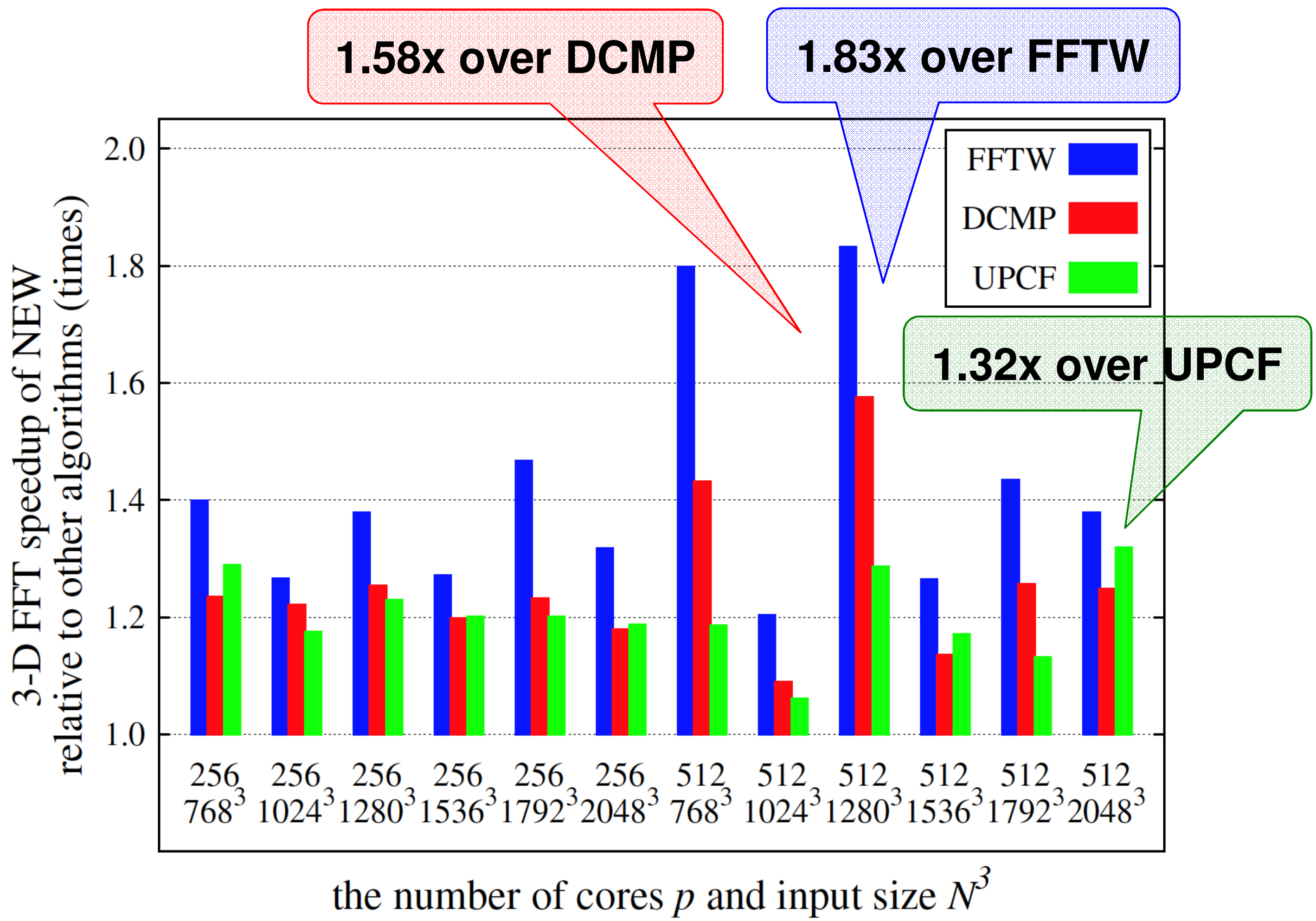
Example: Auto-tunable FFT Libraries

- Works on a 3-D Array of Complex Numbers
  - Parallelization via a 2-D decomposition to increase scaling.
- 24 Parameters
  - Two communication tile sizes
  - Two communication window sizes
  - Eight `MPI_Test()` frequencies
  - Eight sub-tile sizes
  - …
- Why Auto-Tuning?
  - 10X performance variance
  - A huge # possible configurations
  - Various system environments



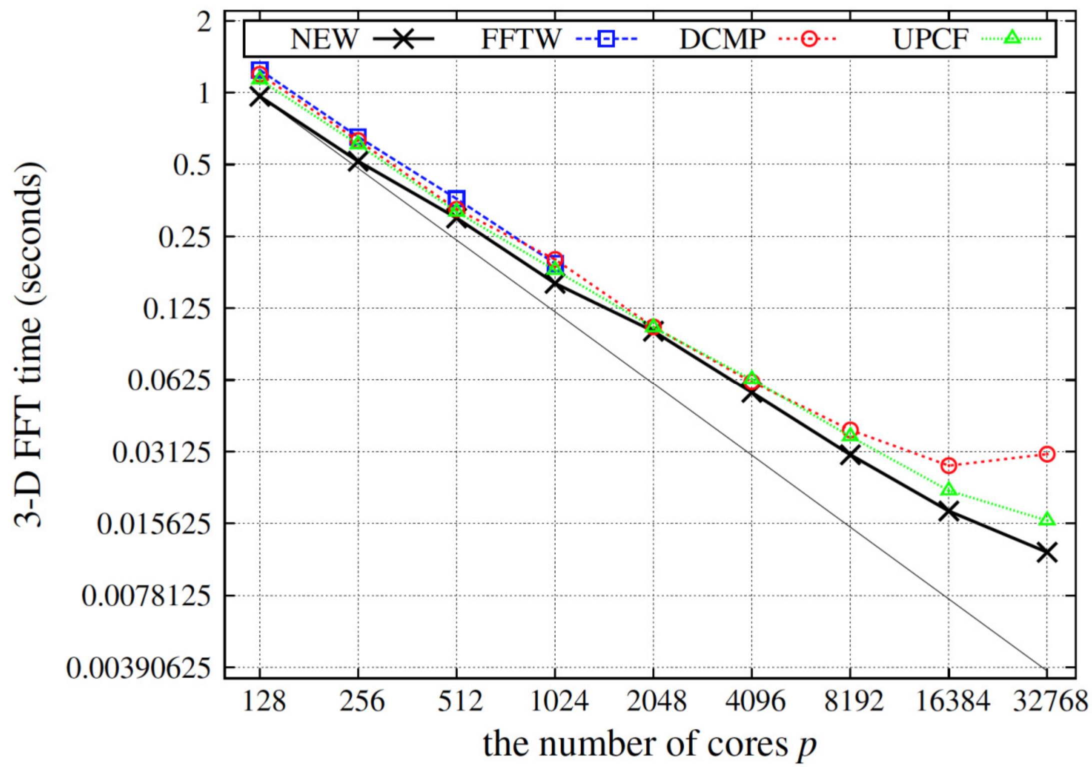
13

Speedup of NEW over Other Methods



Strong Scaling

- $N^3 = 1024^3$, $p = 128 - 32768$



15