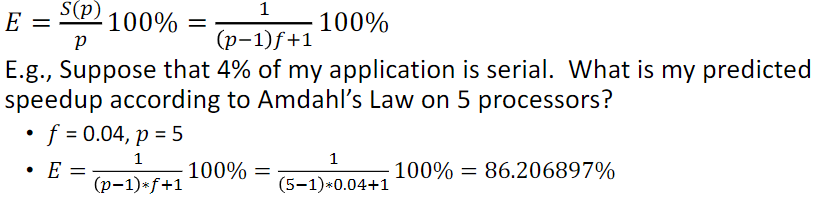
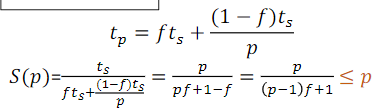
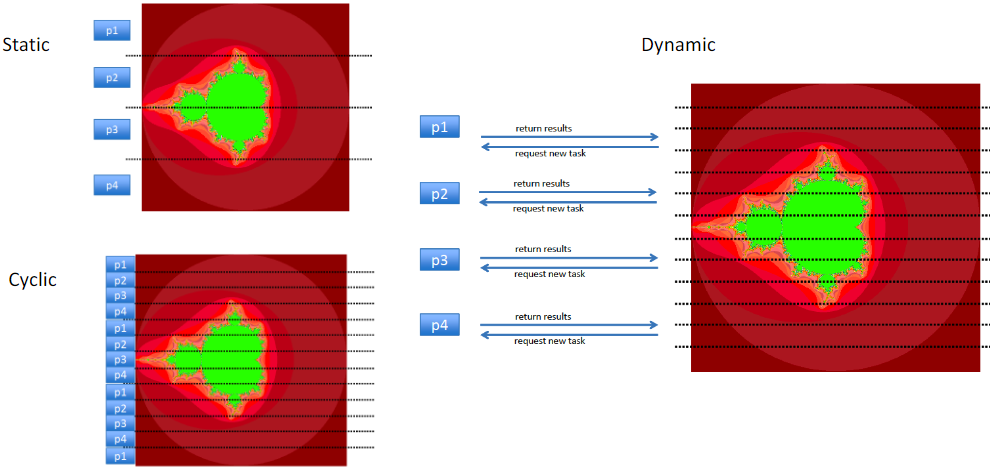
1. Give an example of demand for computation speed
   1. **Use of a supercomputer to increase the precipitation forecasts from 4000 locations to 2.7 million locations**
2. In a sentence, define speedup (slide 1.12)
   1. **“How much faster the program becomes once some computing resources are added”**
   2. 
3. In a sentence, define efficiency (1.13)
   1. **“Ratio of performance improvement per individual unit of computing resource”**
   2. 
4. Describe two limiting factors in achieving maximum parallelism (1.13)
   1. **Non-parallelization of code (can’t parallelize it), and communication overhead**
5. Describe linear speedup? What types of applications will exhibit linear speedup?
   1. **Linear speedup is achieved when S(p) is approximately equal to p (speedup is close to being equal to # of processors)**
   2. **Not sure about types of applications…ones with little-to-no communication overhead?**
6. What is superlinear speedup? What can cause superlinear speedup? (1.12)
   1. **Superlinear speedup is when S(p) > p (speedup is greater than the # of processors)**
   2. **This can be caused by “poor sequential reference implementation, memory caching, I/O Blocking”**
7. In a sentence, define Amdahl’s Law. Give the equation for Amdahl’s Law. (1.12)
   1. **formula which gives the theoretical speedup in latency of the execution of a task at fixed workload that can be expected of a system whose resources are improved.**
   2. 
   3. ^^f is the non-parallelizable portion of the program, ts is sequential runtime, tp is parallelized runtime, p is number of processors
8. Suppose that I get a speedup of 8 when I run my application on 10 processors.
   1. What is the efficiency?
      1. **E = S(p)/p = 8/10 = 0.8**
   2. According to Amdahl's Law, what portion is serial?
      1. 0.8 = 1 / (p-1)f + 1
      2. 0.8 = 1 / (10-1)f +1
      3. 0.8 = 1 / 9f + 1
      4. **f = 0.027777778 (so 2.7% of program is serial/sequential)**
   3. What is the speedup on 20 processors?
      1. S(20) = p / (p-1)f + 1
      2. S(20) = 20 / 19f + 1
      3. S(20) = 20 / 19(0.0277778) + 1
      4. **S(20) = 13.0909 speedup**
   4. What is the best speedup that I could hope for?
      1. **Around 30? But this supposed program has sublinear speedup (highest efficiency is with less processors, like 8)**
9. For each of the following type of parallel computers, briefly describe its architecture, provide its schematic, and name a programming model (1.15-18)
   1. SIMD
      1. **Single instruction stream, multiple data stream**
      2. **Basic parallel computer, partitions data up and performs the same instructions on each portion**
      3. **Programming model?**
   2. Shared memory
      1. **One processor, multiple threads**
      2. **All threads have read/write access to the same memory**
      3. **Programming models:**
         1. **Threads (pthread) –programmer manages all parallelism**
         2. **OpenMP: Compiler extensions handle parallelization through in-code markers**
         3. **Vendor libraries (e.g. Intel math kernel libraries)**
   3. Distributed memory – message passing
      1. **Multiple processors, local memory**
      2. **Data passed via messages through communication network**
      3. **Programming models:**
         1. **MPI: standardized message passing library**
         2. **MPI + OpenMP (hybrid model)**
         3. **MapReduce programming model**
   4. Distributed shared memory
10. Briefly describe the three benchmarking procedures (1.20)
    1. LINPACK
       1. **(Linear Algebra Package): Dense Matrix Solver**
       2. **Measure of the system’s floating point computing power (FLOPS)**
    2. HPCC - High-Performance Computing Cluster
       1. **Group of programs to benchmark how fast your computer can do them**
       2. **HPL (LINPACK to solve linear system of equation)**
       3. **DGEMM (Double Precision General Matric Multiply)**
       4. **STREAM (Memory bandwidth)**
       5. **PTRANS (Parallel Matrix Transpose to measure processors communication)**
       6. **RandomAccess(Random memory updates)**
       7. **FFT (double precision complex discrete fouriertransform)**
       8. **Communication bandwidth and latency**
    3. SHOC
       1. **Used for Scalable Heterogeneous Computing…Non-traditional systems (GPU)**
11. Briefly describe the three supercomputing rankings: (1.21)
    1. TOP500
       1. **Rank the supercomputers based on their LINPACK score**
    2. GREEN500
       1. **Rank the supercomputers with emphasis on energy usage (LINPACK / power consumption)**
    3. GRAPH500
       1. **Rank the supercomputers with emphasis on energy usage (LINPACK / power consumption)**
12. Give the command to load the gcc compiler version 5.3.0 and the openmpi library version 10.3.1 into your environment
    1. **module load openmpi/1.10.3-gcc/5.4.0-cuda9\_2**
13. What is a message tag? (4.12)
    1. **an integer to identify the message. The programmer is responsible for managing tags**
14. What is an MPI communicator? What is the purpose of the communicator? (4.7)
    1. **MPI defines communicator groups for point-to-point and collective communications**
       1. **Unique IDs (rank) are defined for individual processes within a communicator group**
15. In MPI, what is a process rank? What routine do you use to determine a process rank? (4.7)
    1. **Each process has a rank which is a unique ID for that process. To determine a process’s rank, use the MPI\_Comm\_rank() routine.**
16. True or false: In MPI you set the number of processes when you write the source code.
    1. **False**
17. Explain the basic functionality of the collective calls and how they are the same or different from each other. (4.24-31 + 7.5)
    1. **MPI\_Bcast: Broadcasts a message from the process with rank "root" to all other processes of the group**
    2. **MPI\_Scatter: Distributes distinct messages from a single source task to each task in the group**
    3. **MPI\_Gather: Gathers distinct messages from each task in the group to a single destination task (the reverse operation of MPI\_Scatter)**
    4. **MPI\_Reduce: Distributes distinct messages from a single source task to each task in the group**
    5. **MPI\_Barrier: A basic mechanism for synchronism for synchronizing processes –inserted at the point in each process where it must wait. All processes can continue from this point when all the processes have reached it.**
18. With a short sentence, describe the role of MPI.COMM\_WORLD
    1. **MPI\_Comm\_world is the default communicator of MPI. It groups all the processes that are running at the time that it’s called together and makes sure that they can all communicate with one another.**
19. What is “Pleasantly Parallel?” List two characteristics of pleasantly parallel (5.2)
    1. **A computation that can obviously be divided into a number of completely different parts, each of which can be executed by a separate process.”**
    2. **2 characteristics:**
       1. **No communication or very little communication among the processes**
       2. **Each process can do its tasks without any interaction with the other processes**
20. Describe three workload assignment approaches for pleasantly parallel using a few sentences for each approach (5.4-13)
    1. **Static: divide the data/work up evenly amongst the processes**
    2. **Cyclic: divide the data/work up evenly into # of chunks > # of processes, and each process will cover multiple smaller sections in a given order (p1, p2, p3, p4)**
    3. **Dynamic: no set order of processes to complete work. Data is divided up into certain number of chunks (greater than # of processes), then each upcoming chunk is assigned to the next process that is finished with its last chunk (dynamic)**
    4. 
21. How do the worker processes know when to terminate in the dynamic workload assignment model? (5.12)
    1. **Use MPI\_Status, which sends the rank of the sender, the tag of the message, and the length of the message. The programmer can then use this information to find out when the worker processes should terminate.**
22. With a short sentence each, describe the two basic partitioning approaches in parallel programming.
    1. **Data partitioning: dividing the data up between the # of processes running, each process does the same work on the data…this is what we did in Assignment 2**
    2. **Task partitioning: divide the work itself (tasks) up amongst the # of processes, each process performs its task(s) on the entire data…this requires all the tasks to be able to be performed independently of each other**
23. A scientist wants to calculate the sum of 8,192 integers. The scientist has a 16-cores CPU that takes 0.1ns to transfer one integer from one core to another. Construct a tree that describes the divide-and-conquer process to calculate the sum. Calculate the communication time for the “divide” period and the communication time for the “conquer” period.



* 1. **Make sure your tree has 16 elements at the bottom level (# of CPUs).**
  2. **Divide time is (4096+2048+1024+512)(0.1) = 768ns**
     1. **These numbers are the number in the leaves at each level**
  3. **Conquer time is (1+1+1+1)(0.1) = 0.4ns**
     1. **Only send back one number for each level, one number is the sum**
  4. **Total time is 768.4ns**

1. Understand the message passing mechanism of parallel sorting algorithms for odd-even transposition sorting, merge sorting, quick sorting, and bucket sorting. For example, given 8 processes ranked 0 through 7, sort the following array of integers in ascending order using the odd-even transposition sorting process. You should clearly indicate which integers belong to which process at every single stage of the sorting process. [13, 7, 14, 11, 10, 4, 15, 12, 3, 6, 16, 1, 2, 8, 9, 5] (9.3-17?)
   1. **See notepad**
2. Be prepared to answer programming questions of the following type: (4.14 send/recv, 4.26 scatter, 4.28 gather, 4.30 reduce, 6.22 alltoall)
   1. Using only MPI\_Send and MPI\_Recv, implement MPI\_Reduce.
      1. <https://gist.github.com/rmcgibbo/7178576>
   2. Using only MPI\_Send and MPI\_Recv, implement MPI\_Scatter.
      1. <https://stackoverflow.com/questions/52797233/problems-implementing-mpi-scatter-with-mpi-send-and-mpi-recv>
   3. Using only MPI\_Send and MPI\_Recv, implement MPI\_Gather.
      1. Reverse of above link^^^^
   4. Using only MPI\_Send and MPI\_Recv, implement MPI\_Alltoall.
3. Explain if the following MPI code segment is correct or not, and why:

Process 0 executes:

comm.Recv(yourdata, source = 1, tag = 0, status = status)

comm.Send(mydata, des = 1, tag = 0)

Process 1 executes:

comm.Recv(yourdata, source = 0, tag = 0, status = status)

comm.Send(mydata, des = 0, tag = 0)

* 1. **This is a bad implementation…will result in a deadlock!**

1. Understand and be able to read/trace through all codes discussed in class lectures.
2. MPI basic datatypes vs. MPI derived datatypes (8.2-11)
   1. **Basic: MPI\_INT, MPI\_CHAR, MPI\_LONG, etc.**
   2. **Derived: MPI\_Type\_contiguous, MPI\_Type\_vector, MPI\_Type\_indexed, MPI\_Type\_struct**
   3. **Contiguous: Produces a new data type by making count copies of an existing data type**
   4. **Vector: Creates a vector (strided) datatype**
      1. **count: number of blocks**
      2. **blocklen: number of elements in each block**
      3. **stride: number of elements between start of each block**
      4. **It is similar to contiguous, but allows for regular gaps (stride) in the displacements**
   5. **Indexed: Creates an indexed datatype**
      1. **count: number of blocks**
      2. **blocklens: number of elements in each block**
      3. **indices: displacement of each block in multiples of old\_type**
   6. **Struct: Creates a struct datatype**
      1. **count: number of blocks**
      2. **blocklens: number of elements in each block**
      3. **indices: byte displacement of each block**
      4. **old\_types: type of elements in each block**
3. How does MPI parallel I/O work? (10.8-11)
   1. **Multiple processes reading/writing to a file simultaneously**
   2. **Four stages (10.8-11)**
      1. **Open File**
      2. **Set File View (optional)**
      3. **Read or Write Data**
      4. **Close File**