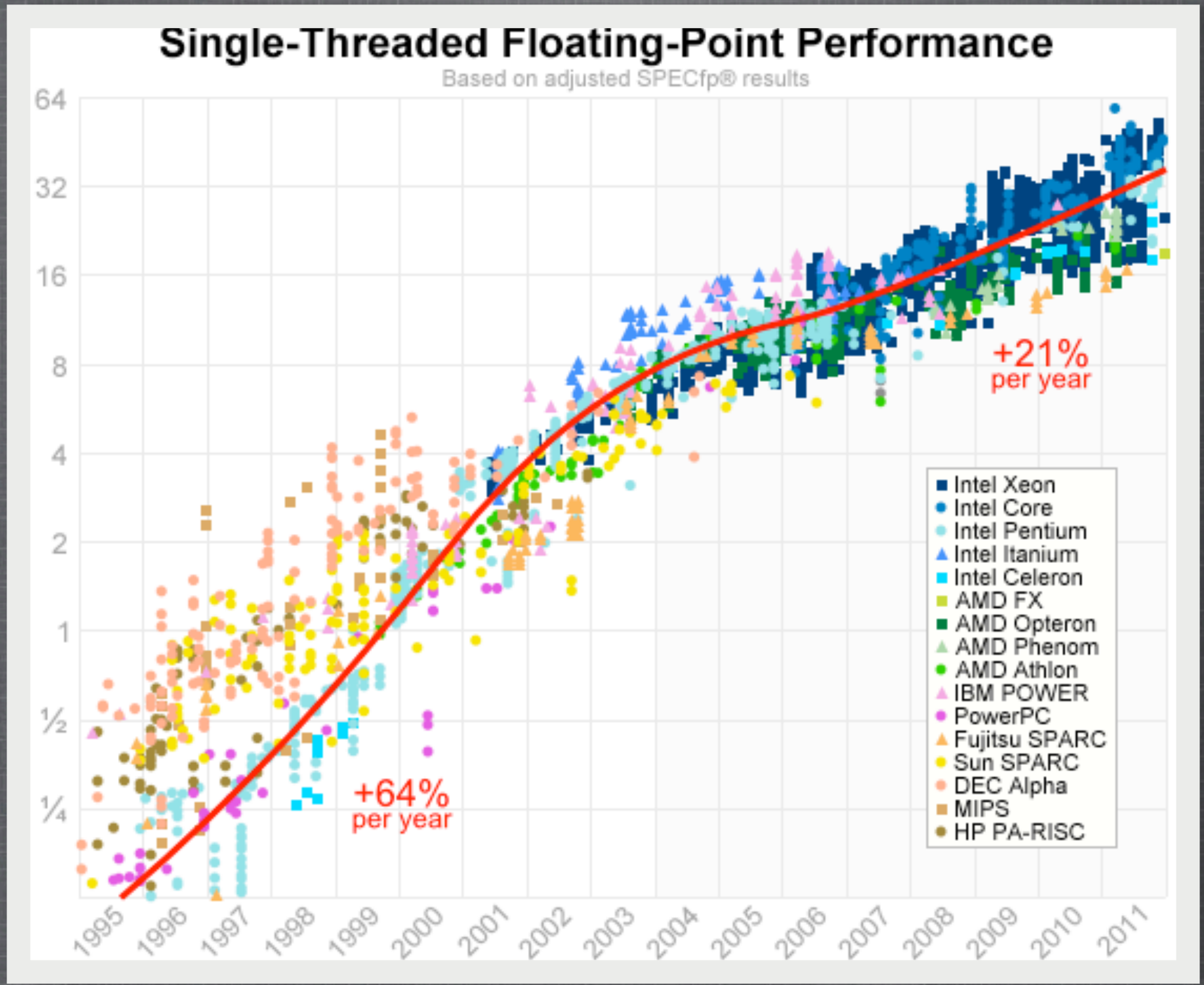
PARALLEL PROCESSING



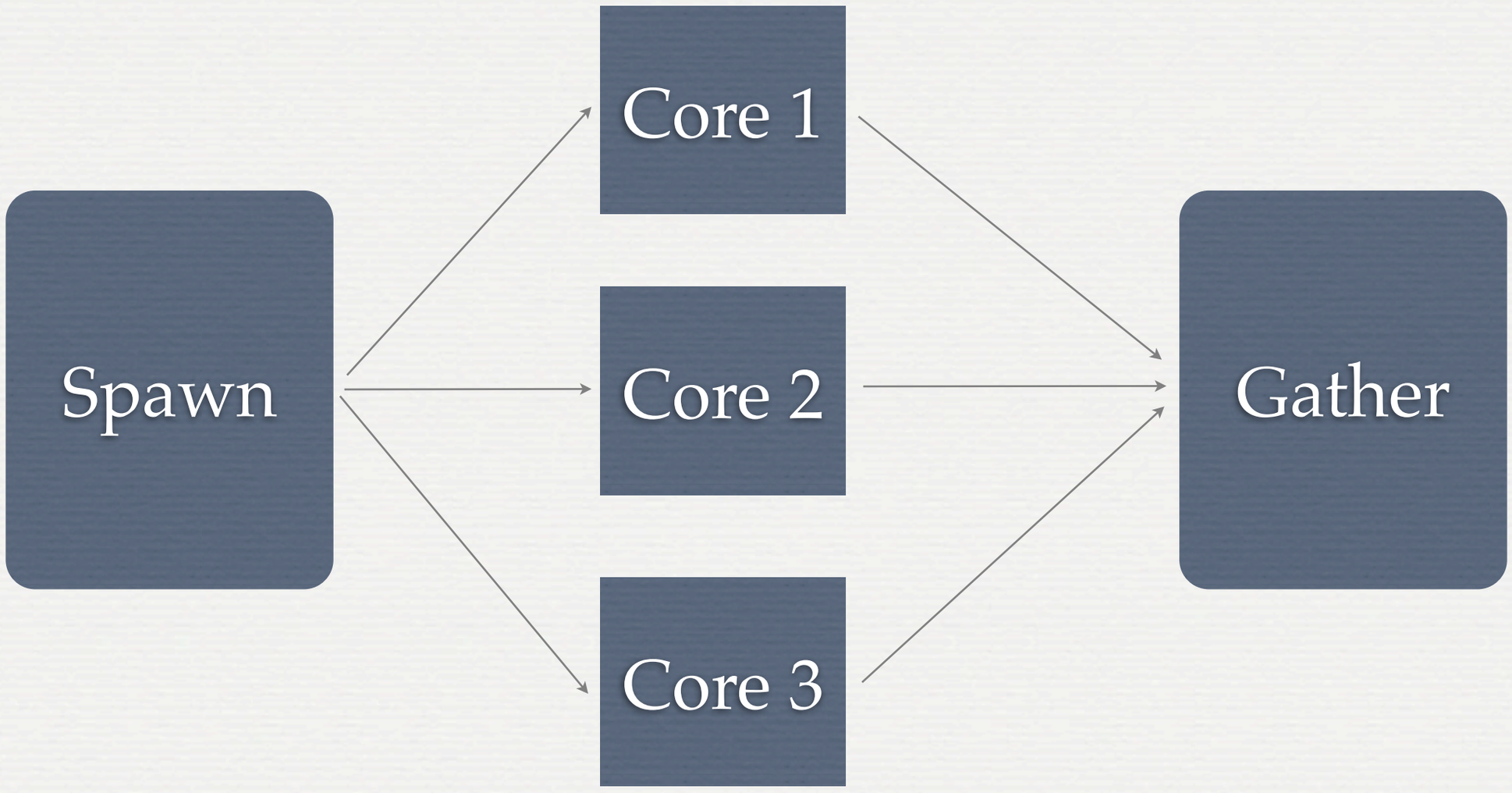
WHY PARALLEL PROCESSING?

Single core performance growth has slowed. More cost-effective to add multiple cores.

THREADS VS PROCESSES

- Both can be used to do parallel processing
  - A process can launch multiple threads (threads run only on the same node)
    - A thread runs within a process and has access to the memory and resources available to that process
- A user or program can launch multiple processes (they can run on the same node or across nodes) and communicate via IPC
- Different tools for working with threads and processes, although the goal is the same

PARALLEL EXECUTION



Spawn -> Compute -> Gather

BENEFITS VS COSTS

- Using more cores can potentially do more work
- But!
  - Each core needs access to the data (spawn)
  - Each core needs to output results (gather)
  - Additional programming and processing time required to implement a solution in parallel

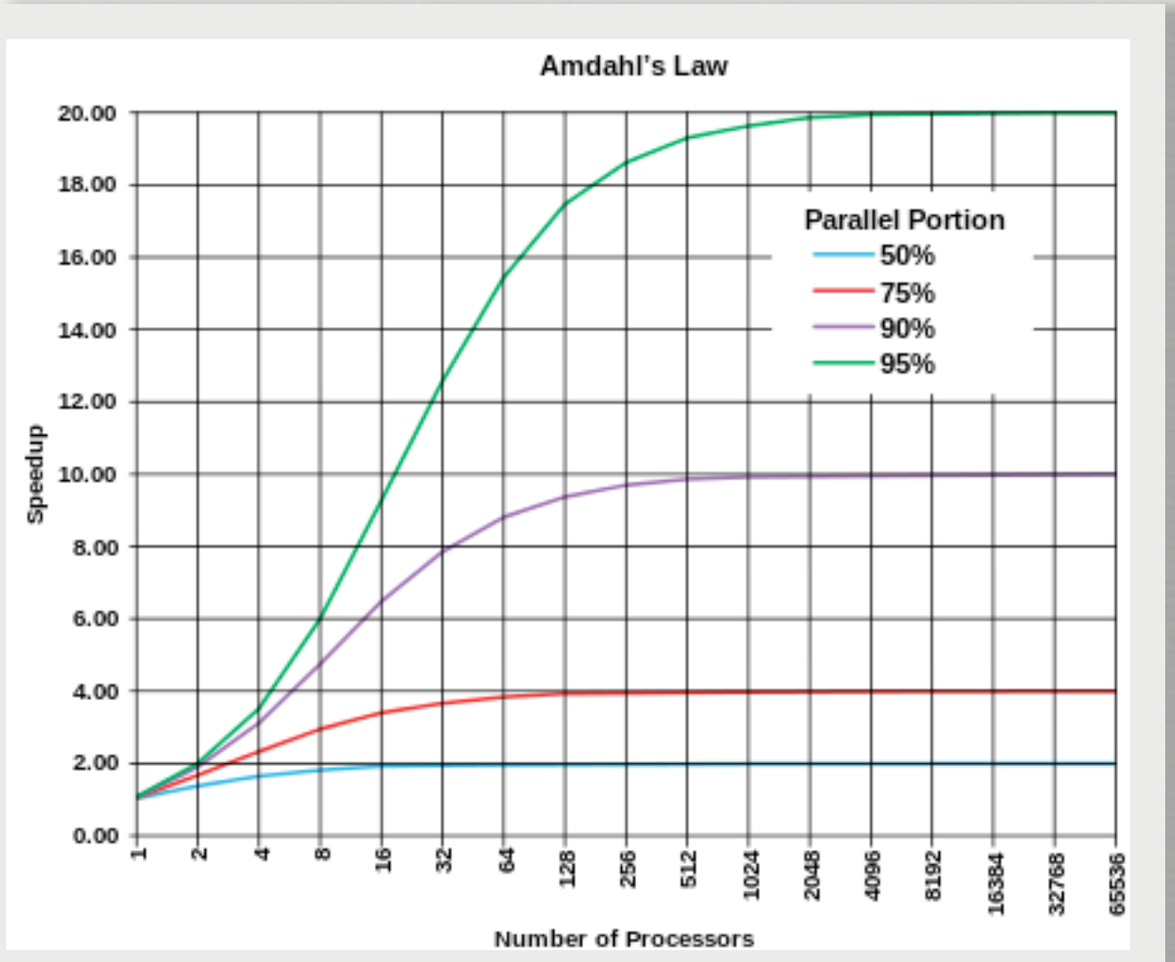
DEPENDENCIES

- Can you break the entire problem (except for the spawn and gather) into independent chunks?
  - Yes - potential candidate for parallel execution
  - No - the benefits of parallelization depend directly on the time spent executing parallel code
    - Identify the portions of your code that do not have dependencies
  - We will not discuss solutions with dependencies

AMDAHL'S LAW

$$\lim_{P \to \infty} \frac{1}{\frac{1-\alpha}{P} + \alpha} = \frac{1}{\alpha}$$

- P=Number of Cores
- alpha=proportion of time NOT spent in parallel
- Diminishing Returns to increasing P for a problem of fixed size



EXAMPLE

- Calculate the column means of a data matrix
  - Divide data into equal groups (or blocks) of rows for parallel processing (spawn)
  - Is the array in memory or does it need to be read from disk?
  - Memory access is at least an order of magnitude faster than disk access (depends on whether the access is sequential or random)

EXAMPLE (CONTINUED)

- Within each group, calculate the sum for each column
  - Store result (what if one core wants to write the result at the same time as another core?)
- Sum the results for each group and divide by the sample size

EXAMPLE (CONTINUED)

- If the data is already in memory then the spawn and gather will be very efficient. If the data must be read from disk, the time spent performing the sum will be small relative to the read time
- If the groups are large, numerous, and the amount of processing time for each group is high then it may be worthwhile to execute in parallel
- Both programming and processing time should be considered when determining whether to implement a solution in parallel

SHOULD I GO PARALLEL?

- Careful tuning of a single threaded application may be a better investment
- Does your program take days or weeks to find a solution?
- How often will you run the code?
- Before implementing a parallel solution spend time assessing the expected benefit

IMPLICIT VS EXPLICIT

- Not necessary to develop a custom solution for each problem
- SAS, Stata, and Matlab all have internal parallel algorithms that will do much if not all of the hard work for you.
    - They will spawn the threads and gather the results using well tested routines
    - Your job is to manage the number of threads
        - Amdahl's law shows that there are diminishing returns to additional cores
- Programs assume all cores on a node are available for parallel processing
    - On a large node this may not be desirable (return for using each additional core may not be worth the opportunity cost)

TAKE CONTROL OF IMPLICIT THREADS

- SAS
  - THREADS and CPUCOUNT=X
  - NOTHREADS
- Stata
  - set processors X
- Matlab
  - -singleCompThread to restrict jobs to the approximate usage of a single core
  - Launch multiple Matlab processes on a single node, each with -singleCompThread enabled

PMON

- Assumes no dependencies across parallel execution code blocks
- Can be used with any program that can be launched with a shell script
- Currently runs on a single node
  - A monitoring script runs in the background and keeps a set amount of processes running
- Runs pre, main, post bash scripts for each group
  - Three job failures at each stage are allowed
- Example: processing a separate job for each NAICS sector

PBS PRO

- Job scheduler on the SSG that can be used to replicate PMON functionality
- qsub -J 1-100 script.sh
- ${PBS_ARRAY_INDEX} is passed to the shell running each job
- Conditional execution
  - JOB_ID_1=$(qsub pre_job.sh)
  - JOB_ID_2=$(qsub -W depend=afterok:$JOB_ID_1 main_job.sh)
  - JOB_ID_3=$(qsub -W depend=afterok:$JOB_ID_2 post_job.sh)
- Spawn and gather depends on a shared filesystem if run in a cluster
- Does not handle failed jobs
- For info see: http://www.osc.ox.ac.uk/content/pbs#PBScondExecution

CONCLUSION

- Use implicit parallel algorithms where possible
  - Manage threads
- Tune single threaded applications
- Implement an explicit parallel solution only after careful consideration of the costs and benefits