COMP 322: Fundamentals of Parallel Programming

Lecture 4: Abstract Performance Metrics, Parallel Array Sum, Amdahl's Law

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

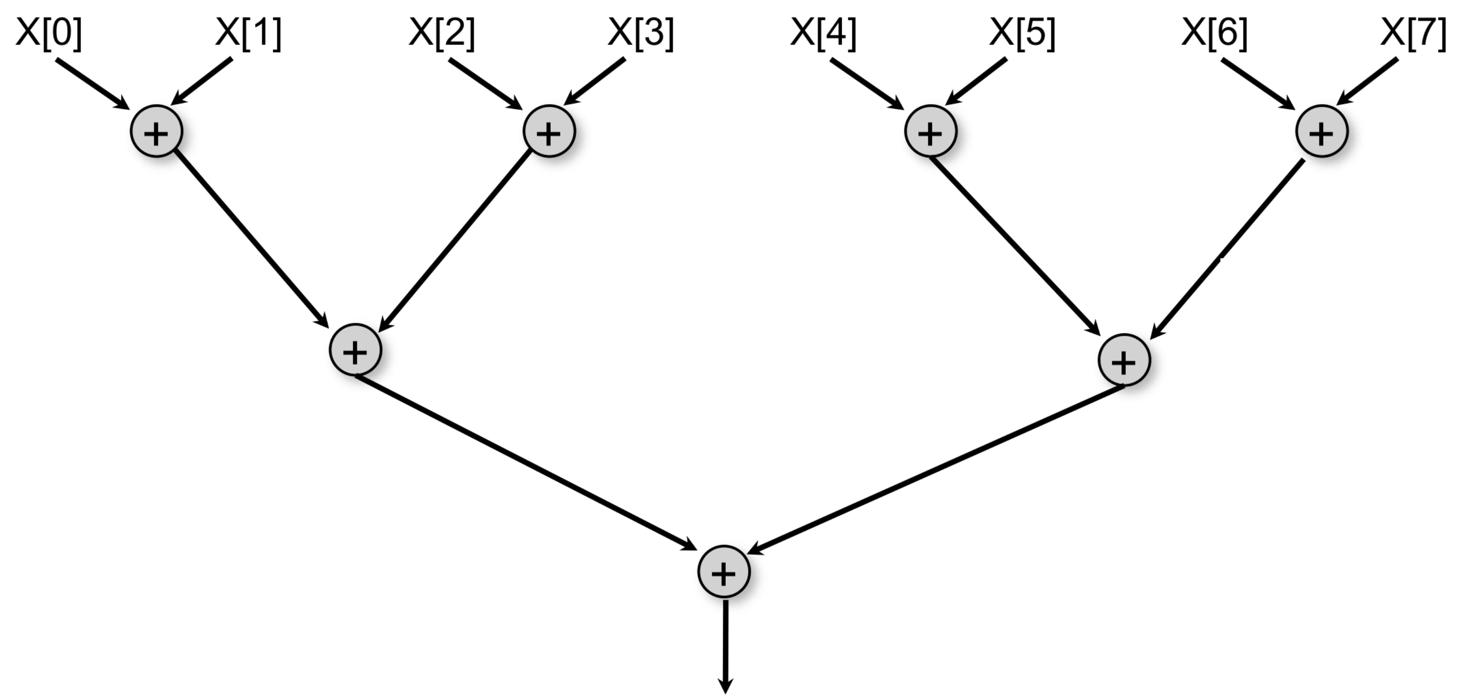
Abstract Performance Metrics

- **Basic Idea**
  - Count operations of interest, as in big-O analysis
  - Abstraction ignores overheads that occur on real systems
- **Calls to doWork()**
  - Programmer inserts calls of the form, doWork(N), within a step to indicate abstraction execution of N application-specific abstract operations
    - e.g., adds, compares, stencil ops, data structure ops
  - Multiple calls dynamically add to the execution time of current step in computation graph
- **Abstract metrics are enabled by calling**
  - **System.setProperty(HjSystemProperty.abstractMetrics.propertyKey(), "true");**
- **If an HJlib program is executed with this option, abstract metrics are printed at end of program execution with WORK(G), CPL(G), Ideal Parallelism = WORK(G)/ CPL(G)**

Parallel Speedup

- Define Speedup(P) = T_1 / T_P
  - Factor by which the use of P processors speeds up execution time relative to 1 processor, for a fixed input size
  - For ideal executions without overhead, 1 <= Speedup(P) <= P
  - Linear speedup
    - When Speedup(P) = k*P, for some constant k, 0 < k < 1
- Ideal Parallelism = Parallel Speedup on an unbounded number of processors

Reduction Tree Schema for computing Array Sum in parallel



Assume input array size = S, and each add takes 1 unit of time:

- WORK(G) = S-1
- CPL(G) = log2(S)
- Estimate T_P = WORK(G)/P + CPL(G) = (S-1)/P + log2(S)
  - Within a factor of 2 of any schedule's execution time

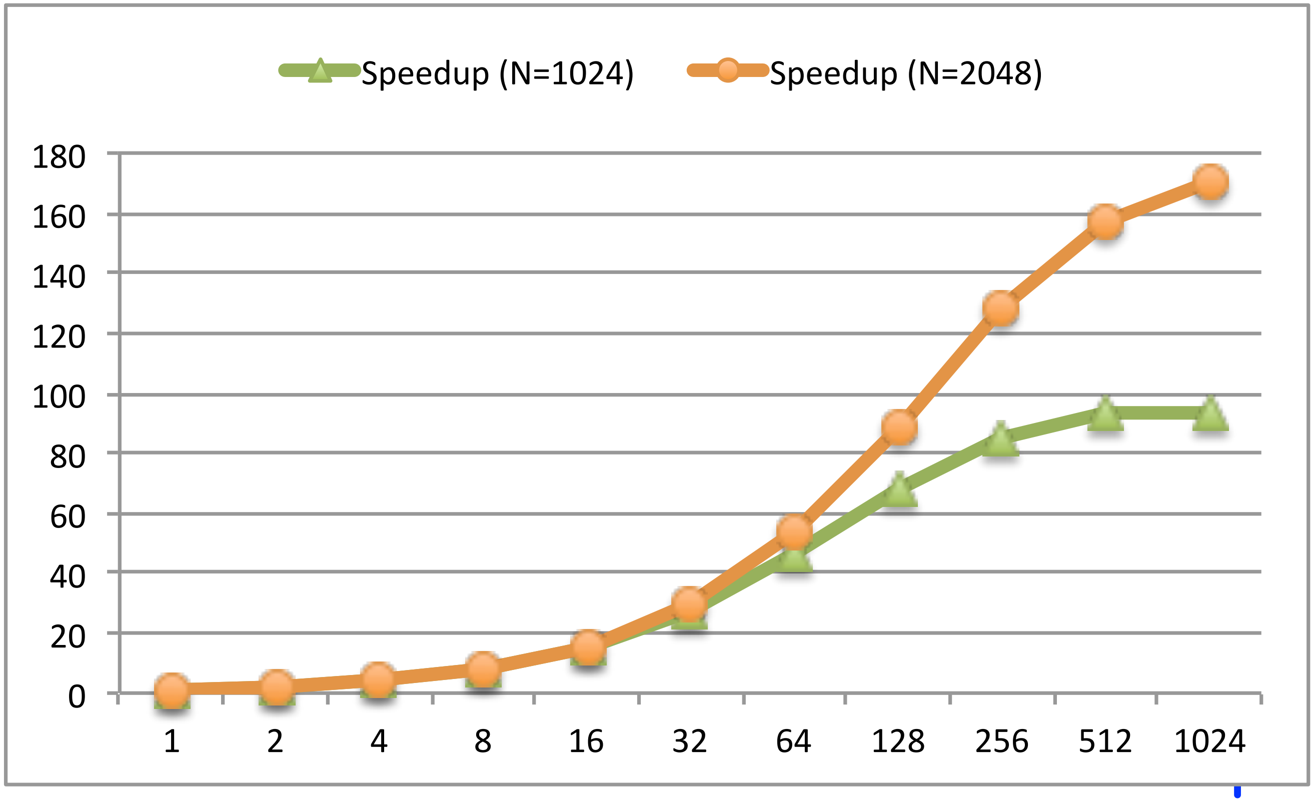
How many processors should we use?

- Define Efficiency(P) = Speedup(P)/ P = $T_1/(P * T_P)$
  - Processor efficiency --- figure of merit that indicates how well a parallel program uses available processors
  - For ideal executions without overhead, 1/P <= Efficiency(P) <= 1
- Half-performance metric
  - $S_{1/2}$ = input size that achieves Efficiency(P) = 0.5 for a given P
  - Figure of merit that indicates how large an input size is needed to obtain efficient parallelism
  - A larger value of $S_{1/2}$ indicates that the problem is harder to parallelize efficiently
- How many processors to use?
  - Common goal: choose number of processors, P for a given input size, S, so that efficiency is at least 0.5

ArraySum: Speedup as function of array size, S, and number of processors, P

- Speedup(S,P) = T(S,1)/T(S,P) = S/(S/P + $\log_2$(S))
- Asymptotically, Speedup(S,P) --> S/$\log_2$S, as P --> infinity

Speedup(S,P)



Amdahl's Law [1967]

- If $q \leq 1$ is the fraction of WORK in a parallel program that must be executed sequentially for a given input size S, then the best speedup that can be obtained for that program is $Speedup(S,P) \leq 1/q$.
- Observation follows directly from critical path length lower bound on parallel execution time
  - CPL >= q * T(S,1)
  - T(S,P) >= q * T(S,1)
  - Speedup(S,P) = T(S,1)/T(S,P) <= 1/q
- This upper bound on speedup simplistically assumes that work in program can be divided into sequential and parallel portions
  - Sequential portion of WORK = q
    - also denoted as f_S (fraction of sequential work)
  - Parallel portion of WORK = 1-q
    - also denoted as f_p (fraction of parallel work)
- Computation graph is more general and takes dependences into account

Illustration of Amdahl's Law: Best Case Speedup as function of Parallel Portion

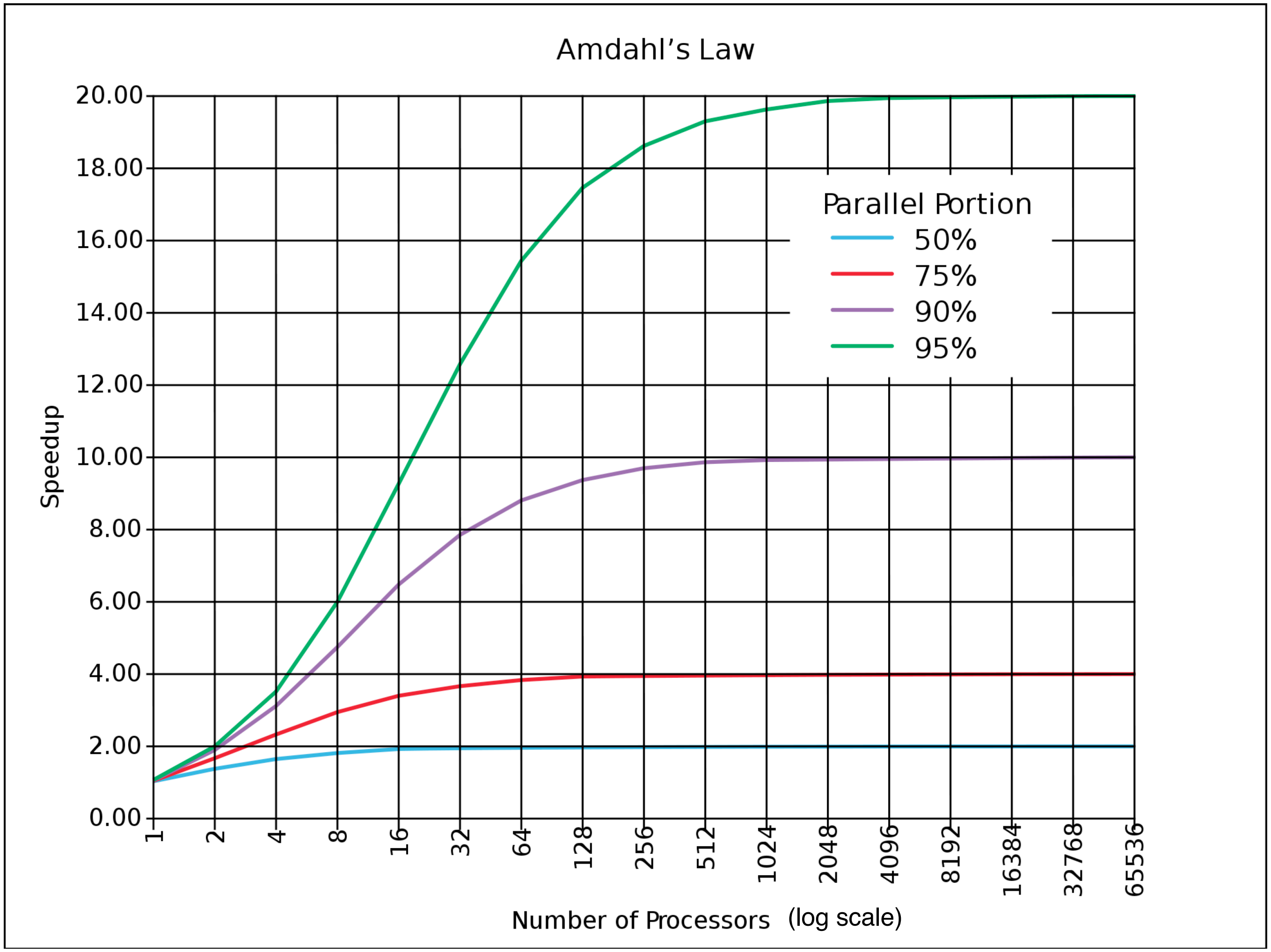


Figure source: http://en.wikipedia.org/wiki/Amdahl's law