Scalability Metrics

Parallel Processing

- In parallel processing, rather than having a **single program** execute tasks in a sequence, parts of the program are instead **split** such that the program is executed **concurrently** (i.e. at the same time), by multiple entities.
- This implies that the execution time of a program can be made **arbitrarily small** by making the tasks **decomposition finer** in granularity.
- However, **concurrent** tasks might need to **communicate** with other tasks to exchange data.
- Results in **communication overhead**.
- The **tradeoff** between the granularity of task **decomposition** and its corresponding **overheads** often determines the parallel **performance** of a program.

Parallelization Benefits

- aids in achieving **speedup** (aids in solving problem in less time).
- facilitates solving **bigger problems**.
- allows a problem that is **too big to fit in the memory** of one processor to be broken up such that it is able to fit in the memories of multiple processors.

Parallelization Overheads

- **Interprocess interactions**
  - due to communications between processors while sharing tasks data.
- **Idling of Processing Elements**
  - caused due to load imbalance, synchronization, or unparallelizable serial parts of program.
- **Excess Computation**
  - Best serial algorithm is difficult to parallelize; parallelizing sequential algorithm involve excess computation overhead.

Performance Metrics

- It is imperative to **study** the performance of **parallel programs**
  - to **determine** the best algorithm.
  - to **evaluate** hardware platforms.
  - to **examine** the benefits from parallelism.

- a **number of metrics** are used to analyze the **performance** of parallel algorithm's

Parallel Overhead

- **P**= Total number of processing elements.

- $\mathbf{T_{total}}$= total time collectively spent by all the processing elements.
- $\mathbf{T_{total}}$ = **PTp** (where is Tp is time spent by a processing element).
- **Ts**= serial time.

- **Total parallel overhead**, **To**- time spent by all processing elements in non-useful work

- $\mathbf{To = T_{total} - Ts = PTp - Ts}$

Speedup

- The **speedup** of a **parallel code** is how much **faster** it runs in parallel.
- If the time it takes to run a code on one processors is **Ts** and the time it takes to run the same code on P processors is **Tp**, then the **speedup** is given by **ratio of a serial runtime to the parallel runtime**

- **Speedup, S = Ts / Tp**

- **Ts= Serial runtime of best sequential algorithm**

Efficiency

- **Speedup = P (can be delivered only in ideal parallel system with P processing elements)**
- **Processing** elements cannot **devote 100% time to computations.**
- **Efficiency**
  - a **measure** of how much of your **available processing** power is being **used** The simplest way to think of it is as the speedup per processor. This is equivalent to defining efficiency as the time to run P models on P processors to the time to run one model on one processor.
  - defined as **fraction** of **time** for which a **processing element** is usefully **employed**,
  - **ratio of speedup to the number of processing elements**

- **Efficiency, E = S/P**

- This gives a more **accurate measure** of the true efficiency of a **parallel program** than **CPU usage**, as it considers **redundant calculations** as well as **idle time**.

Scaling

- **Now that we have developed a parallel algorithm, a natural next question is, "does the algorithm scale?"**
  - **Efficiency, E can be written as = S/P = Ts / (PTp)**
    - **Or**
      - **$E = 1/(1+T_o/Ts)$**

Note: The total overhead function **To** is an increasing function of **P** .

- For a **given problem size**
  - T_s remains constant
  - T_o **increases** with **increase** in the number of processing elements, **P**.
  - Efficiency, **E** of the parallel program **decreases.**

Amdahl's Law

- **Amdahl's Law** shows us that a program will have **diminishing returns** in terms of speedup as the **number of processors** is **increased**.
- However, it does not place a limit on the weak scaling that can be achieved by the program, as the program may allow for bigger classes of problems to be solved as more processors become available.
- The **advantages of parallelism for weak scaling** are summarized by John Gustafson in Gustafson's Law.

Isoeffeciency Metric of Scalability

- An **isoefficiency function** can be obtained in terms of the problem size as a function of **P (no. of processing elements)** to keep **efficiency**, **E**, constant.

- **This function determines:** the ease with which a parallel system can maintain a constant efficiency.

- Aids in **achieving speedups** in increasing **proportion** to the number of processing elements **P.**

Speedup Factors

- **The primary issue with speedup is the communication to computation ratio. To get a higher speedup, you can:**

- Maximize data locality.
- Minimize volume of data exchange.
- Minimize frequency of interactions.
- Minimize contention and hot-spots.

Speedup Factors

- Overlap computations with interactions
- Replicate data or computations.
- Use group communications instead of point-to-point primitives.
- Overlap interactions with other interactions.

Acknowledgements

Adapted from:

- Chapter 3 slides, by A. Grama, of the text: Principles of Parallel Algorithm Design by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar

Thank You!