Page No.
Date

U-4 (HPC)

Analytical Modeling of Parallel Programs.

Q- Sources of Overhead in Parallel programs.

- sequential algorithm → its runtime (evaluated by)
- parallel runtime → depends on input size
  → parameters of machine (communication)
  → ~~depends on input~~ no. of processors.

- execution time of parallel algo → Input size + no. of processing elements.
- combination of parallel algo & parallel architecture.

Diff overhead -

1) Interprocessor communication -

① Processor working on non-trivial parallel problem will need to talk.

② source of parallel processing overhead → time spent to interact & commⁿ data b/w processing elements.

③ data dependancy exists in problem & no. of processing element working on same data.

④ Two processoring element are independent they can't work in parallel.

2) Idling (Load Balance (im)) -

- process may become idle because of many reason.
- Diff. factors that contribute to idling of processing elements like load imbalance, synchronization, preserve of serial comopnent in dynamic substack generated.
- Dynamic task generation application, it's very difficult to predict size of subtasks assigned to processor in advance.
- To maintain uniform workload among processors, dynamic subtasks generated
- dynamic task generation appln → subtasks assigned to processor in advance.
- maintain uniform workload among processors. Can't dvide problem statistically b/w processing elements.

3) Excess Computation -

- computation → not by serial version.
- serial algo → difficult to parallelize or are repeated across process to minimize communication
- excess = parallel - Best serial
- Best serial program → diff factors parallel code → excess comp

* Performance measure & Analysis -

Metrics → 1) Efficiency 2) SpeedCost 3) cost 4) Parallel Overhead 5) Execution time

- speed up - captures relative benefit of solving a problem in parallel.

$S = \text{speedup} = \frac{T_D}{T_P}$ (time taken by best sequential algorm / by single process) [time taken by p processors].

1) * Execution Time → Time to complete a task & denoted by Texe.

i) Sequential or serial runtime (T_S) - time elapse → b/w start of execution. end of execution → sequential computer serial runtime.

ii) Parallel Runtime (T_P) - time that elapse from moment first processor starts to moment last processor finishes execution.

2) * Total Overhead -

- denoted by T_o.
- time spent by parallel program - time spent by serial program.
- ~~$S = T_S/T_P$~~ • $T_o = P T_P - T_S$ (P is processing element)

~~T_S → execution time on a single processor~~

~~T_P → execution time using a parallel processor.~~

- ~~Speedup S is given by $S = \Theta(n/\log n)$~~

3) Speed Up -

- faster the code runs → parallel over serial execution.
- evaluates → benefit of solving problem in parallel.
- $s = \frac{T_s}{T_P}$ (exection time on single processor) (~~speedup~~ execution time using a parallel processor).

speedup $S = \Theta(n/\log n)$.

4) Efficiency -

- measure of fraction → time → processing element (processor) is engaged.
- Ratio of speedup to no. of processing elements

$$E = \frac{\text{speedup}}{\text{Processing elements}} = \frac{S}{P} \qquad E = \Theta(1/\log n)$$

5) Cost -

- cost of parallel algo → or program cost = parallel running time / no. of processors.
- cost reflects sum of time → spends solving problem.
- cost is referred as work or processor time product.
- cost of system $pT_p = n \log n$.

* Amdahl's Law -

- consider a fixed computational workload W & a number of processors p.
- let execution rate of i processor be R_i so for sequential processors ($i=1$) $R_i = 1$ & $R_n = n$.
- Assume sequential workload is αW ($0 \le \alpha \le 1$) & parallel workload is $(1-\alpha)W$
- Used for fixed workload parallel system.
- no. of processing elements or processors increase, time required for execution must reduce.
- speedup for n processors according to execution must reduce. amdahl's law can given as-

$$S(n) = \frac{t_s}{f t_s + (1-f) t_s / n}$$

t_s = total time required on sequential system.

f = fraction of code.

$1-f$ = parallelizable part.

2) Gustafson's Law -

- overcome the drawbacks of Amdahl's law.
- relaxed the problem size from being fixed to be of any size.
- instead of having fixed problem size or fixed workload we must assume → fixed execution time.
- case of huge problem size → increase our system size.

$$s'(n) = \frac{n + s(1-n)}{1} \qquad s'(n) = n + s(1-n).$$

eqn → scaled speedup factor of Gustafson's law.

✱ Effect of Granuality on Performance -

- No. of task divisions $\propto$ Granuality.
- Fine Granuality → Decomposition into large no. of tasks.
- Coarse Granuality → Decomposition into small no. of tasks.
- Degree of concurrency $\propto$ Granuality.
- practice we do fine granuality.
- less processing elements than available is called scaling down.
- n inputs & p processing elements & $p < n$ we can design as parallel algo works for n processing element by using n/p virtual.
- processing elements by n/p workload at each processing unit increases by a factor of n/p.

Granuality impacts -

- affects performance of parallel computers.
- fine granualarity increases speedup.
- However synchronization overload ~~navigation~~ negatively affects ~~pipeline~~ performance.

* Scalability of parallel systems –

- capacity of increase speedup in proportion to no. of processing elements.
- reflects parallel system's ability to utilize increasing processing resources effectively.
- efficiency is given as – $E = \frac{S}{P} = \frac{T_s}{pT_p} = \frac{1}{1 + \frac{T_2}{T_s}}$
- Total overheard to increasing f^n of p.
- every parallel program always has a sequential that runs for time ts.
- overload increases superlinearly due to communication overhead, idealing, excess computation
- As $p \uparrow$ $E \downarrow$ goes down $E \propto 1/p$
- scale more than efficiency shall decrease.
- efficiency may be more than efficiency shall decrease.
- efficiency may increase if problem size is increased keeping no. of processors constant.

* Minimum execution time & min cost –

- no. of processors increase parallel runtime continues to descrease & asymptoically approaches a minimum values, goes on inceasing after a min val.
- give min execution time as –

$$\frac{d}{dp} T_p^{min} = 0.$$

gives no. of processing elements for which Tp is min.

- max. no. of processing elements is bounded by degree of concurrency.

$$T_p^{min} = \frac{w + T_0(w, c[w])}{c[w]}$$

* Asymptotic Analysis of parallel ~~alg~~ programs -

Example of adding n. nos. with n processing elements -

$T_p = \Theta(\log n)$. $S = \Theta(n/\log n)$.

* ~~Matrix~~ ~~Matrix Vector Multiplication~~

- evaluates behaviour of parallel algo → input size approaches infinity
- characterizing algo complexity & performance scaling.
- examines how algorithmic efficiency changes with increasing problem
- common asymptotic notation → Big O, Theta & Big Omega.
- provides insight into scalability & efficiency of parallel algo.
- identify dominant computational & costs.
- guides algo selection & optimization.
- with lower asymptotic complexities are preffered for large scale computation
- facilitates comparision b/w diff parallel algo & system config.

* Matrix Vector Multiplication -

- computes product of a matrix & vector, resulting in new vector
- fundamental operation in linear algebra & computational science.
- can be parallelized to exploit concurrency & improve performance.
- algo for include row-wise, columwise & block wise approaches.
- parallelized aim to distribute computational workload across multiple processing units efficiently.
- optimize data locality & minimize commun^n overhead.
- key building block in scientific & engineering applications & img processing.
- essential for accelerating computations in large scale numerical simulations.
- using specialized computing devices like GPUs enhaces performance of matrix - vector multiplication
- optimization techniques like loop unrolling, data blocking & vectorization further improve efficiency of parallel matrix vector multiplication algo.

Page No.
Date

* Matrix Matrix Multiplication Algorithm –

- computes product of two matrices using parallel processing.
- divides matrices using parallel processing into smaller blocks & distributes among multiple processing.
- each process computes a portion of resulting matrix independently.
- reduces overall time & leverage concurrent processing.
- utilizes techniques like data processing partioning & commⁿ among processes
- scales well with no. of available processing units, improving performance.
- implemented using parallel algo like Cannon's algo.
- optimizes resource utilization by distributing workload across multiple processors.
- overheads is minimized by efficiently exchanging data b/w processors.

* Execution Rate & Redundancy –

Redundancy → measures increase in required computation when using more processing units

Execution Rate ↓ to defined on per component basis. & assigned a sample time based on sample times of components it is conected to.