Parallel Computations for Large-Scale Problems

Lecture 1. Introduction

Def: T_s^* → execution time of the fastest serial algorithm.

T_p → total execution time of P processors

Parallel speedup $S_p = T_s^* / T_p$.

Efficiency in distributed memory machines

* Data Parallelism : Similar processing
* Data Partitioning : partition into adjacent regions
* Relaxed Algorithm : Embarrasingly parallel : No sync
* Synchronous Iterations : Do synchronous.
* Replicated Workers : Central pool.
* Pipelined Computation : arranged in a structure

Lecture 2. Performance Evaluation

Amdahl's Law : Assume serial part $t_s = f T_1$. $\Rightarrow$

$$T_p = f T_1 + \frac{(1-f)T_1}{P} \Rightarrow S_p = \frac{T_1}{T_P} = \frac{P}{1+(P-1)f}.$$

$$\Rightarrow \lim_{P\to\infty} S_p = \frac{1}{f}$$

Gustafson's Law : Assume T_p constant. $\Rightarrow$

$$T_1 = f' T_p + (1-f') P T_p \Rightarrow S_p' = f' + (1-f')P$$

Def: Parallel efficiency $\eta_p = S_p / P$.

Cost $C = T_p P$.

Parallel Execution Time:

$T_p = t_{comp} + t_{comm}$; $t_{comp,i} = f_i(n, P)$;

$t_{comp} = t_{comp,1} + t_{comp,2} + \cdots$; Similarly

$t_{comm} = t_{comm,1} + t_{comm,2} + \cdots$; $t_{comm,i} = t_{startup} + w t_{data}$.

Lecture C : C-programming

* GDB
  - gcc -g -Wall -o prog prog.c -lm
  - gdb prog
    - In gdb : type run to start your program
    - bt : prints current call stack (list of nested functions)
    - p x : prints value of variable x
    - break file.c:123 : set break point
    - continue : continues execution
    - clear 1 : remove break point 1.
    - l : list program code
* Valgrind
* Gprof.
* IDE

Lecture 3: Introduction to MPI.

Blocking send: int MPI_send

Blocking Receive: int MPI_Recv

Non-blocking: MPI_Isend ; MPI_Wait.

Data Movement Routines

* Broadcast * Scattering $\Leftrightarrow$ Gathering

Global Computation Routines

* Reduce

MPI Calls:

* MPI_INIT: establishes an environment.
* MPI_COMM_SIZE: return # of processes
* MPI_COMM_RANK: return rank of the process.
* MPI_SEND * MPI_RECV
* MPI_FINALIZE: exit MPI cleanly.

Lecture 4. Image Reconstruction and Poisson's equation

The common denominator: use a uniform mesh.

* Load-balanced linear data distribution
* Load-balanced scatter distribution

Fill the Ghost Cells: Communication

$\Rightarrow$ Safe solution: red-black coloring.

Red-Black Communication.

* Associate each process with a color (red or black) in the p and q directions such that no neighbor has the same color.

Optimal process Topology: Minimize $\Phi(P) = \left(\frac{P}{M} + \frac{Q}{N}\right)$.

$\Rightarrow P = \sqrt{\frac{M}{N} R}$, $R = P \times Q$

Surface to Volume Ratio:

* T_{comp} is proportional to $I_p * J_q$.
* T_{comm} is proportional to $2(I_p + J_q)$

Gauss-Seidal Iteration: Change value immediately

Lexicographic Order: for $n = 1..N$; for $m = 1..M$: $u(m,n)$

Pipelined Computations: Gauss-Seidal. Red-black ordering.

Lecture 5. Matrix Multiplication and Collective Communication

Addition of Matrices:

Embarrasingly parallel, all data access purely local.

Recursive Doubling for taking sums.

Matrix-Vector Multiplication

* Use a load-balanced data distribution on $R = P \times Q$ processors in array topology to store A.
* Once data distributed, all individual components can be computed in parallel using the Recursive Doubling Algorithm

Matrix-Matrix Multiplication.

Assume $M=N=K$ and $R=P\times P$.

Cannon's Algorithm:

* In a first step, assume $P=M$.
* Data distribution: processor $(m-1, n-1)$ holds $a_{m,n}$ $b_{m,n}$, $c_{m,n}$
* On the "diagonal" processors $(m-1, n-1)$, parts of the sum $c_{m,m} = a_{m,1}\cdot b_{1m} + \ldots + a_{m,m}\cdot b_{m,m} + \ldots + a_{m,M}\cdot b_{M,m}$ are available.
* Shifting the rows of A cyclic to the right and those of B cyclic downwards provides the next term.
* Repeat this cyclic exchange ones again completes the calculation of the diagonal elements.

Algorithm:

① Initially, processor (p,q) has elements $a_{p+1,q+1}$, $b_{p+1,q+1}$

② Shift the rows of A and the columns of B into the structure described above.

③ For $k=1 \ldots N$

1. Multiply the local values on each processor
2. Shift the rows of A and the columns of B cyclically by one processor

④ If nessassary: undo step 1.

Definition of PageRank

* Let m be a webpage
* Let F_m be the set of pages m points to
* Let B_m be the set of pages that point to m.
* Let $N_m = |F_m|$ be the number of forward links, and c be a normalization factor
* Then we define PageRank $R_m = c\sum_{n\in B_m} \frac{R_n}{N_n}$
* Define a matrix A by $a_{mn} = \begin{cases} 1/N_n, & \text{if there is link from } n \text{ to } m \\ 0, & \text{otherwise.} \end{cases}$
* Let $R=(R_1, \ldots, R_M)^T$ be the pagerank vector. Then it holds $R = cAR$.

The Power Method for Eigenvalue Problems.

* Given a square matrix A and an initial guess $x^{[0]} \neq 0$
* Form the sequence $x^{[k+1]} = Ax^{[k]}$, $k=0, 1, \ldots$
* For many A, $\{x^{[k]} / \|x^{[k]}\|\}$ converges towards an eigenvector to the largest eigenvalue c of A.
* An estimation c_k of c is given by $c_k = \frac{\langle x^{[k+1]}, x^{[k]} \rangle}{\langle x^{[k]}, x^{[k]} \rangle}$

Vector Transposition

* If $P=Q$, MPI_Alltoall can be used to transpose y.
* The general case is equivalent to one recursive doubling step.

Lecture 6. Linear Systems of Equations by Gaussian Elimination

Gaussian Elimination

Idea: For $n = 1, \dots, N-1$: Use equation n for the elimination of x_n from the equations $n+1, \dots, N$

For $n = N, \dots, 1$: Compute x_n from the n-th equation

Forward elimination + Backsubstitution

Pivoting + LU decomposition

Use data partition to distribute

Forward substitution : Sequential algorithm

Use pipelined computation : Lots of communication

Use Data Distribution μ :

* Assume a row data distribution μ
* The last process $(P-1)$ must wait for x_1 to arrive.
* x_1 is ready to send if the first processor (0) has done its work.
* In an ideal world, further communication is completely overlapped by computation.
* $P \times Q$ processor mesh gives rather poor performance.

Lecture 7. Parallel Sorting

Symmetric Distributed Compare And Exchange

Linear data distribution place N/P consecutive elements on each processor. First step is local sorting.

Then do the merging. Complexity $(2N/P - 1)t_a$.

Parallel Bubble Sort : Odd-Even Sort

2 phases, all processes are comparing. Guaranteed to terminate after $N/2$ odd-even and even-odd steps.

Bitonic Mergesort : Bitonic Sort implemented on $P = N = 2^D$ processor has a time complexity of $O(\log^2 N)$.

Bucket Sort : Uniformly distributed keys.

Lecture 8. Algorithms on Graph.

Assume (connected) weighted undirected graph.

Algorithms For Dense Graphs

Prim's algorithm : Inner loop can be parallelized.

Distribute V load balanced over P processes.

Distribute the weighted adjacency matrix A correspondingly over a $1 \times P$ process mesh.

Single Source Shortest Paths - Dijkstra's Algorithm

Similar to Prim's algorithm

Floyd's Algorithm : Parallel innermost loop

Algorithms For Spares Graphs