

Tutorial 2

Performance of Parallel Systems

CS3210 – 2020/21 Semester 1

Learning Outcomes

1. Analyze parallelism using task dependence graphs
2. Apply parallel programming patterns
3. Understand scalability of parallel programs by analyzing speedup
4. Understand & compute the various metrics that can be used to measure the performance of parallel systems

1. **[Task Parallelism]** Study the following parbegin – parend code fragments:

Fragment 1

```
parbegin
do
  parbegin
    do
      parbegin
        do
          A
          C
        end
        parallel
          B
        parend
      F
    end
  parallel
    D
  parend
  G
end
parallel
  E
parend
H
```

Fragment 2

```
A
parbegin
do
  parbegin
    B
    parallel
      C
    parend
  F
end
parallel
do
  parbegin
    D
    parallel
      E
    parend
  G
end
parend
H
```

Parbegin-parend is a parallel programming pattern that allows us to explicitly specify parallel regions in code. parbegin allows us to define the start of a parallel/concurrent region. This will spawn a set of threads that will execute subsequent statements until it reaches parend. Parend denotes the end of a parallel region and all threads spawned by the matching parbegin are joined/halted.

In our example, statements in a parbegin-parend region are only executed in parallel with another statement, if they both statements are explicitly defined to run in parallel, through the use of the parallel keyword. That is to say, A; B; in a parallel region will run sequentially, unless it is defined like so: A parallel B.

The time taken to execute the various tasks that are run in both fragments varies as follows:

Task	A	B	C	D	E	F	G	H
Time units	5	5	5	14	14	2	10	4

- (a) Sketch out the task dependence graphs for both Fragments 1 & 2.
- (b) Plot the degree of task parallelism over time for both Fragments 1 & 2. Derive the average and maximum parallelism.
- (c) What is the maximum achievable speedup for both Fragments 1 & 2, given infinite resources?
- (d) If we have only 2 processing elements, what is the maximum achievable speedup for both Fragments 1 & 2?
2. **[Parallel Programming Models]** Consider the problem of matrix-vector multiplication: $A \cdot b = c$, where A is a 2D array of size $N \times N$, and b and c are 1D arrays of size N . Each element of the result is computed as follows: $c_i = \sum_{j=0}^{N-1} a_{ij} \cdot b_j$
- (a) What type of parallelism does the matrix-vector multiplication problem entail – data or task? Which type of architecture from Flynn's taxonomy would you choose to solve it? Briefly explain your choice.
- (b) Which parallel programming pattern (e.g. pipeline, producer-consumer, etc.) would you use to solve the matrix-vector multiplication problem?
3. **[Performance of Parallel Systems]** Consider two processors P_1 and P_2 which have the instruction set. P_1 has a clock rate of 6 GHz, P_2 has a clock rate of 2 GHz. The instructions of the processors can be partitioned into three classes. The following table specifies the CPI values for each instruction class + processor combination. We assume that there are three compilers C_1 , C_2 , and C_3 available for both processors.

Consider a specific program X . All three compilers generate machine programs which lead to the execution of the same number of instructions. But the instruction classes are represented with different proportions according to the following table:

Class	CPI for P1	CPI for P2	% of Instructions		
			C1	C2	C3
FP Arithmetic	4	2	30	30	50
Integer ADD/SUB	6	4	50	20	20
Integer MUL	8	3	20	50	30

Table 1: CPI & proportion for each instruction class

- (a) If C1 is used for both processors, how much faster is P1 than P2?
- (b) If C2 is used for both processors, how much faster is P1 than P2?
- (c) Which of the three compilers is best for P1?
- (d) Which of the three compilers is best for P2?
4. **[Speedup & Theoretical Maximum Speedup]** Consider the following program with a sequential portion and a parallel portion for problem size N :

Sequential: N instructions	Parallel: N^2 instructions
-------------------------------------	-------------------------------------

For simplicity, we assume each instruction can be executed in 2 cycles on a 1 GHz processor.

- (a) (Amdahl's Law) Suppose N is 100, calculate the speedup achieved with 10 and 100 processors respectively. What is the upper limit of speedup?
- (b) (Gustafson's Law) Calculate the time needed to run $N = 100$ on a single core. Suppose we are given the same amount of time, what is the problem size we can solve with 10 and 100 processors? What is the speedup achieved for those problem size and processor count?