앞에서 decompose 통해서 Task 만들어졌는데... Task의 특성 / Interaction 특성

① Task type

- static / dynamic : task decomposition을 미리 예측할 수 있는지?
  - static : data, recursive decomposition
  - dynamic : speculative, exploratory decomposition

- uniform / non-uniform : task size가 일정한지!? (size → computation volume)

- Data size 와 computation size 관계
  - Input = output < computation : sorting
  - Input = computation > output : MIN
  - computation > Input : exploratory

② Interaction type

- static / dynamic

- regular / irregular

- Read only / Read-write
  - ↳ shared variable에 대한 write는 항상 synchronization 생각하기
    - lock / semaphore / conditional variable

- two-side / one-side : one task가 another task에게 operation 할 때, another task와 반응 해야 하는지 아닌지!?
  - [send / receive] [read / write]

Characteristics of Tasks

computation

key characteristics

Task가 발생하는 Type (static / dynamic)

- task generation strategies
- associated data size
- associated work

Task generation

- Static task generation (task가 어떤 애들인지 미리 알고있음)
  - identify concurrent tasks a priori
  - typically for data/recursive decompositions
  - example
    - → matrix operations, graph algo., static graph, finding minimum in list
- Dynamic task generation (we don't know how much computation to do)
  - identify concurrent tasks as computational unfolds
  - typically for exploratory / speculative decompositions
  - example
    - → puzzle solving, game playing, quick sort pivot에 따라 size of subarray variant!

Array로 치면, 각 task에게 할당받은 Array element 개수 같냐!?!?

Task size

- uniform : all the same size
- non-uniform : known / unknown .. case by case
  - ex) quicksort ... size of partition(task) depends on pivot

Data size — Computation

서로 linear 관계 $f(\text{input}) = ax + b$

- size(input) < size(computation) → 15 puzzle (exploratory decomposition)
- size(input) = size(computation) > size(output) → find min.
- size(input) = size(output) < size(computation) → sort

data ≒ context

- data size implication
  - small data : task context can easily migrate to another thread
  - large data : tie task to thread
    - → can avoid communicating task context (PC, ...), moving data
    - → large data 경우에는 interact / store 보다 reconstruct / recompute 훨씬 나을지도!?

large, Temporary data는 store 하는 것보다 reconstruct(recompute)가 낫다.

Characteristics of Task Interactions

Communication

- To share data and work for synchronization

classification criteria	• static / dynamic • regular / irregular • read-only / read-write • one-sided / two-sided
Static / Dynamic	
Static interaction	: task dependency / interaction timings are known priori ex) Matrix Multiplication ··· 언제 각 output element가 계산되는지 알 수 있음!
Dynamic Interaction	: timing / interacting tasks cannot be determined priori ex) 15 puzzle, quick-sort - message-passing 관계에서는 harder to code ··· sender/receiver 동시 ready X
Regular / Irregular	
Regular Interaction	: interactions have a pattern that has some structure ex) mesh (모든 node들이 interact 가능 ··· bandwidth saturation↑) ring (one node는 two neighbor만 interact 가능) - schedule communication to avoid conflicts on network link 쉽게 가능
Irregular Interaction	: interactions with no pattern - no well-defined topology, harder to handle

- ex) Sparse matrix-vector multiplication
  - → static task generation (task가 무엇인지 미리 알고있음)
  - → access pattern for b depends on structure of Sparse Matrix A
  - (즉, A의 sparse 형태에 따라서 b에 접근 필요/불필요 정해짐)

각 node 마다 communication도 Input data에 의해 정해지니 패턴이 Irregular 해짐

Characteristics of Task Interactions (cont.)

Read-only / Read-write

Read-only Interaction : tasks only read data associated with other tasks

- ex) parallel matrix multiplication

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \begin{pmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{pmatrix}$$

shared input

(a)

Task 1: $C_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1}$
Task 2: $C_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$
Task 3: $C_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1}$
Task 4: $C_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$

Read-write Interaction : read and modify data associated with other tasks

- ex) priority queue-based heuristic search for 15 puzzle
  - → 기존의 exploratory 15 puzzle = exhaustive search (each state is equally valued)
  - → priority queue based heuristic search에서는 current state에서 moved tiles를 기준으로 priority를 설정해서 queue에 pop/push. 각 task의 priority R/W 하는 것 예시!
- requires synchronization (avoid r/w and w/w ordering races)
  - lock, semaphore
    - → task가 작거나 many threads의 경우 task의 priority queue에 two frequent access 가능

write 할 때마다 same lock 사용할 때 task 크기가 fine 해서 request frequency가 높아지면 최악임!

이런 경우에는
- priority queue를 여러개로 만들거나
- task를 키우거나
- task의 process로 migrate 하는 방법도 있음!

Two-sided / one-sided

Two-sided Interaction : task가 끝나려면, 다른 task와의 interaction이 필수

- ex) send, receive
- Producer/Consumer problem … message passing/signal 필수!

One-sided Interaction : initiated/completed independently

- ex) Read, write … 다른 task가 one task의 R/W 해도 그 one task는 상관 x
- key-value store (message-passing 없이는 쓰일 수 없음)