Understanding parallelism

SpeedUp vs Efficiency

SpeedUp (S_p): relative reduction of execution time whn using P processors with respect sequential.

Efficiency (Eff_p): it is a measure of the fraction of time for which processing element is usefull.

Escalability

> - Strong: resources x2 -> scalability x2
> - Weaak: resources x2 w. proportional work

Amdahl's law

Par_Fraction

> $\varphi = T_{seq\_time\_of\_par\_part} / T_{seq\_exec}$

$$S_p = \frac{T_1}{T_p} = \frac{T_1}{(1-\varphi) \times T_1 + (\varphi \times T_1/P)}$$

$$S_p = \frac{1}{((1-\varphi) + \varphi/P)}$$

Note: If P approach to infinit, φ/P approach to 0, then S_p = 1/(1-φ).

Ex:

```
seq - 25s
par - 50s
seq - 25s
      100s
```

> φ = 100/50 = 0,5 SpeedUp $_{par}$ = 50/10 = 5
>
> SpeedUp = 100/60 = 1.67

Sources of overhead

> - task creation
> - barrier sync
> - task sync
> - exclusive access to data
> - data sharing
> - Idleness
> - Computation (extra work to obtain a palalel algorithm)
> - Memory (extra memory to obtain a palalel algorithm)
> - Contention (competition for the access to shared resources)

$$T_p = (1-\varphi) \times T_1 + \varphi \times T_1/p + overhead$$

How to model data sharing overload?

Example:

Jacobi solver

> $T_{calc} = (N^2/P)*t_{boddy}$
>
> $T_p = T_{calc} + T_{comm}$
>
> Tcomm = $2(t_s+t_w*N)$

Data sharing modeling

Example 4

```
void compute(int n, double *u, double *utmp) {
    int i, j;
    double tmp;
    for (i = 1; i < n-1; i++) {
        for (j = 1; j < n-1; j++) {
            tmp = u[n*(i+1) + j] + u[n*(i-1) + j] + // elements u[i+1][j]
and u[i-1][j]
            u[n*i + (j+1)] + u[n*i + (j-1)] - // elements u[i][j+1] and
u[i][j-1]
            4 * u[n*i + j]; // element u[i][j]
            u[n*i + j] = tmp/4; // element u[i][j]
        }
    }
}
```

Each cpu with n^2/P elements. Tasks compute segments of n/c rows by c columns.

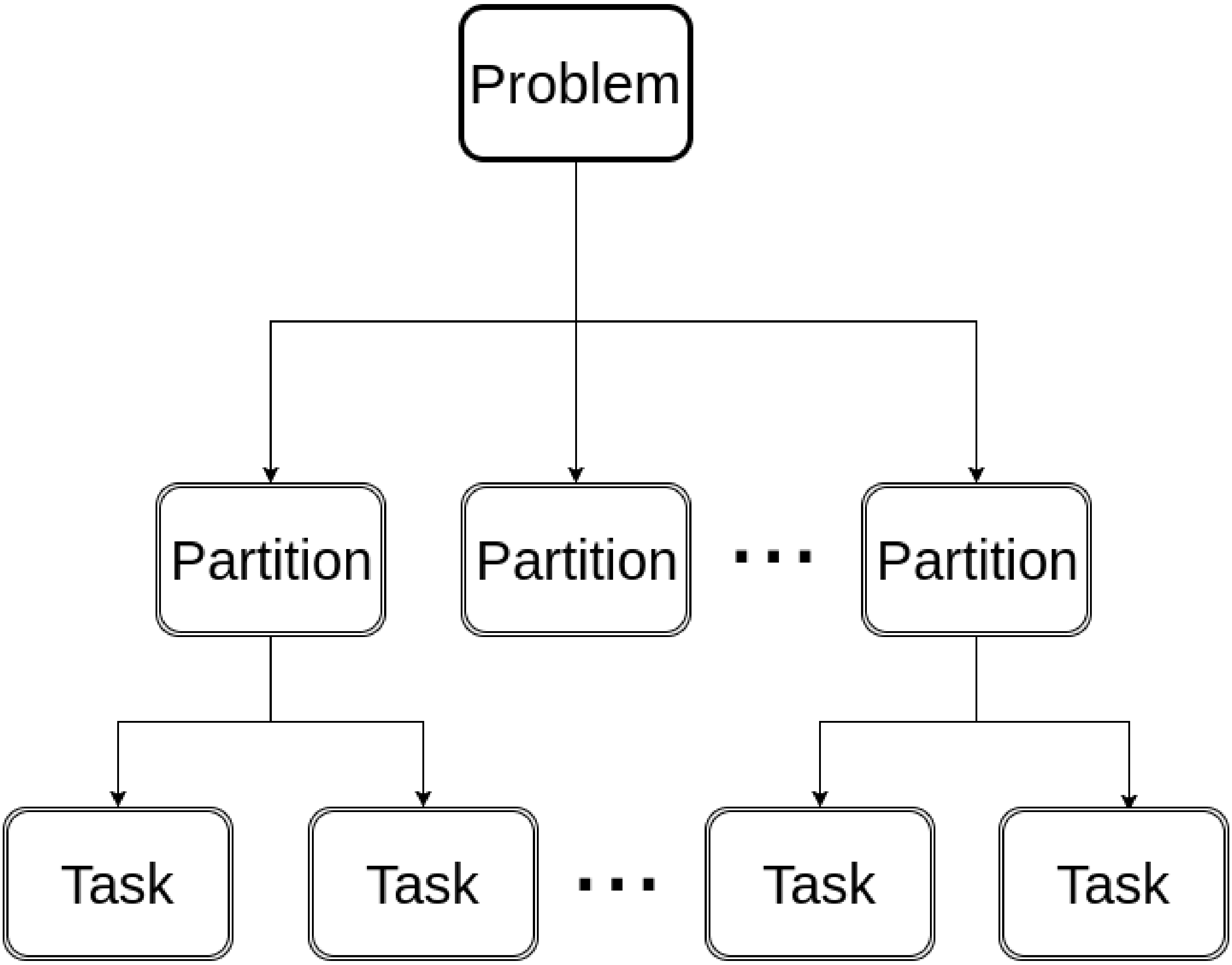
Then, time acquire the following form:

$$T_P = (\frac{n}{c} + P - 1) \times (\frac{n}{P} \times c) \times t_{body} \quad +$$

$$(t_s + n \times t_w) + ((\frac{n}{c} + P - 2) \times (t_s + c \times t_w))$$

$$S_P = \frac{T_1}{T_P} = \frac{n^2 \times t_{body}}{T_P}$$

Task decomposition



Types:

- Lineal Task decomposition

> Code block or procedure

- Iterative task decomposition

> Iterative constructs

- Recursive task decomposition

> Recursive procedures

Decomposition Strategies

Leaf strategy

Create one task sequentially for each leaf of task tree.

> Less tasks
>
> Less overhead

Tree strategy

Creator one task for each invocation.

> More Tasks
>
> More Overhead

Cut-off control

If tree strategy is in use, when a certain number of task are created or the granulation si too small or in a certain number of recursive calls; you can change the strategy to Leaf in order to reduce overhead.

Task ordering constraints

Dependences

Constrains in the paralel execution of tasks

- Task ordering costraints

> They force the execution of tasks in a requiered order

- Data sharing constraints

> They force the access to data to fulfil certain properties.

Task ordering constraints

- Control flow constraints

> The creation of a task depends on the outcome (decision) of one or more previous tasks.

- Data flow constraints

> The execution of a task can not startuntil one or more previous tasks have computed some data.

Task synchronization in Open MP

- Thread barriers

> Wait for all threads to finish previous work.

- Task barriers
  - taskwait

>```
>        Suspends the current task waiting on the completion of child
>tasks of the current task. (stand-alone directive).
>```

```
*  taskgorup
```

> ```
>          Suspends the current task at the end of structured block
> waiting on completion of child tasks of the current task and their
> descendent tasks
> ```

- Task dependences

Task dependences

- IN
- Out
- InOut

You can creat a dependence fro a part of a task. For example a coss-iteration dependence.

Task ordering constrains

- Task ordering constraints
- Data sharing constraints

Problem 1

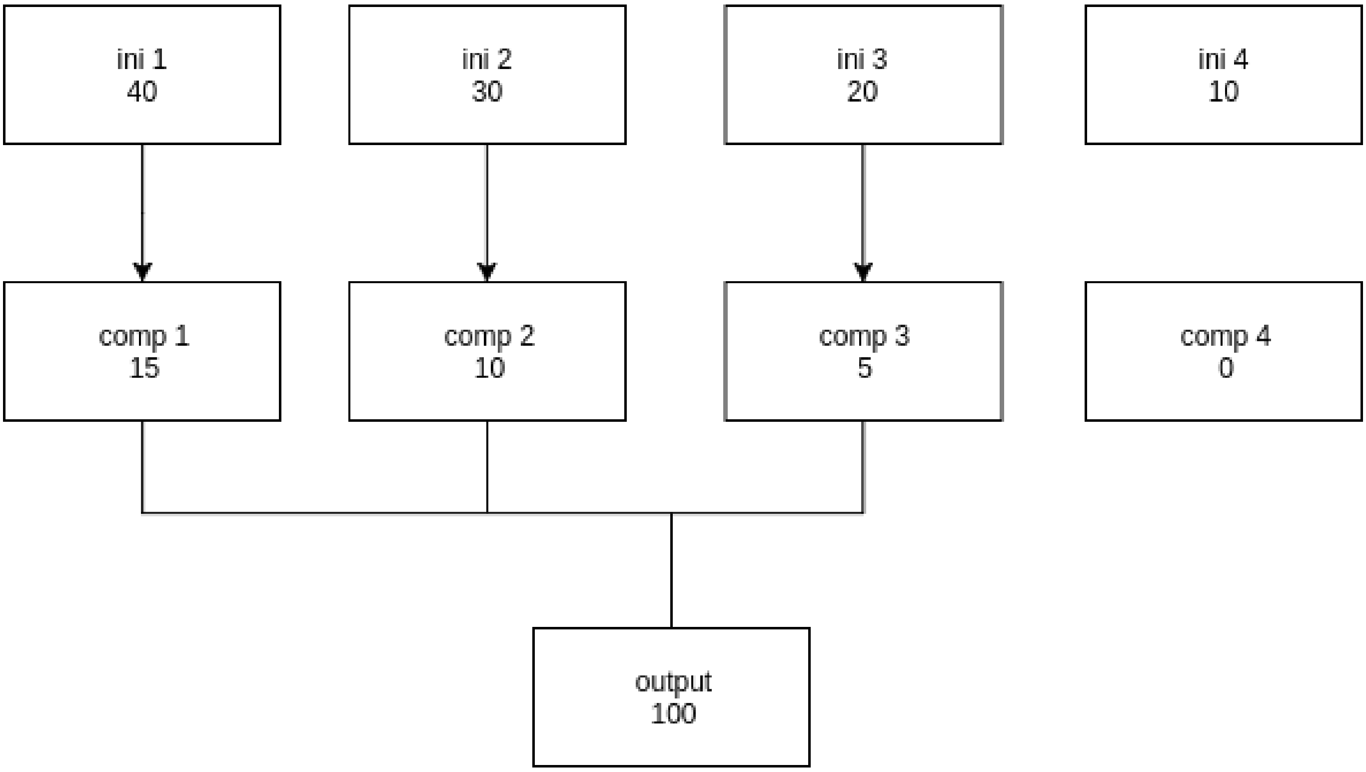
Given the following C code with tasks identified using the T areador API:

```
#define N 4
int m[N][N];

// initialization
for (int i=0; i<N; i++) {
    tareador_start_task ("for_initialize");
    for (int k=i; k<N; k++) {
        if (k == i) modify_d(&m[i][i], i, i);
        else {
            modify_nd (&m[i][k], i, k);
            modify_nd (&m[k][i], k, i);
        }
    }
    tareador_end_task ("for-initialize");
}

// computation
for (int i=0; i<N; i++) {
    tareador_start_task ("for_compute");
    for (int k=i+1; k<N; k++) {
        int tmp = m[i][k];
```

```
            m[i][k] = m[k][i];
            m[k][i] = tmp;
        }
        tareador_end_task ("for-compute");
    }

    // print results
    tareador_star_task ("output");
    print_results(m);
    tareador_end_task ("output");
```

Assuming that: 1) the execution of the modify_d routine takes 10 time units and the execution of the modify_nd routines takes 5 time units; 2) each internal iteration of the computation loop (i.e. each internal iteration of the for_compute task) takes 5 time units; and 3) the execution of the output task takes 100 time units.

1. Draw the task dependence graph (TDG), indicating for each node its cost in terms of execution time (in time units).



2. Compute the values for T1, T∞, the parallel fraction (phi) as well as the potential parallelism

$T_1 = 230$

$T_{inf} = 155$

Phy = 130/230

Par = T1/Tinf = 230/155

3. Indicate which would be the most appropriate task assignment on two processors in order to obtain the best possible "speed up". Calculate T2 and S2.

cpu 1	ini1	ini4	comp1		Output
cpu 2	ini2	ini3	comp2	comp3	