Table 1: Global Parameters and Notations

Notation	Description
Φ	Set of DAGs (Directed Acyclic Graphs) with tasks and dependencies
D	Set of deadlines $\{D_1, D_2, \ldots, D_n\}$ for each DAG
C	Set of heterogeneous cores with voltage-frequency levels
P	Power consumption profiles for tasks on different cores
TDP_{chip}	Thermal Design Power limit for the entire chip
TDP_{core}	Thermal Design Power limit per core
W	Task workload matrix containing execution times for each task
$T_{i,k}$	Task instance to be scheduled
$arrival\_time_{i,k}$	Arrival time of task instance (i, k)
$deadline_{i,k}$	Deadline of task instance (i, k)
S	Current schedule state
$partitions$	Available time slots per core
PPL	Power profile list tracking chip-level power usage

Algorithm 1 Schedule multiple DAGs on multicore environment

Inputs: $\Phi, D = \{D_1, D_2, \ldots, D_n\}, C, P, TDP_{\text{chip}}, TDP_{\text{core}}, W$
Outputs: S or **FAILURE**
function SCHEDULEDAGS
    $(schedule, free\_slots, power\_usage, all\_tasks, hyperperiod) \leftarrow$ SetupDAGEnvironment()
    while $all\_tasks \neq \emptyset$ **do**
        $current\_task \leftarrow$ select_highest_priority(all_tasks)
        $allocated \leftarrow$ ScheduleTask($current\_task$)
        if $allocated = 0$ **then**
            return FAILURE # Task could not be scheduled
        end if
        all_tasks.remove($current\_task$)
    end while
    return *schedule* # Successfully scheduled all tasks
end function

Algorithm 2 Setup DAG Environment

Inputs: $\Phi, D, C, P, TDP_{\text{chip}}, TDP_{\text{core}}, W$
Outputs: S
function SETUPDAGENVIRONMENT
    $hyperperiod \leftarrow LCM(D_1, D_2, ..., D_n)$ # Calculate hyperperiod
    $all\_tasks \leftarrow \emptyset$ # Store all task instances
    for each DAG $\phi \in \Phi$ **do**
        $repeat\_count \leftarrow hyperperiod/D_\phi$ # Number of repetitions
        for all tasks i in ϕ **do**
            for $k \leftarrow 1$ to $repeat\_count$ **do**
                $task\_start \leftarrow (k-1) \times D_\phi$
                $task\_deadline \leftarrow k \times D_\phi$
                all_tasks.add($(i, k, task\_start, task\_deadline)$)
            end for
        end for
    end for
    $schedule \leftarrow [\emptyset] \times |C|$ # Initialize empty schedule
    $free\_slots \leftarrow \{c : [(0, hyperperiod)] \text{ for } c \in C\}$ # Track free slots
    $power\_usage \leftarrow [0] \times hyperperiod$ # Track power consumption
    return $(schedule, free\_slots, power\_usage, all\_tasks, hyperperiod)$
end function

Algorithm 3 Allocate Core for Task with Power-Aware Scheduling

Inputs: $\Phi, D = \{D_1, D_2, \ldots, D_n\}, C, P, TDP_{\text{chip}}, TDP_{\text{core}}, W$
Outputs: S
function SCHEDULETASK($T_{i,k}$)
    $start\_time \leftarrow \max(arrival\_time_{i,k}, \text{predecessor\_finish}(T_{i,k}))$
    $cores\_sorted \leftarrow \text{sort\_cores\_by\_execution\_time}(C, T_{i,k})$
    $scheduled \leftarrow false$
    while not scheduled **do**
        $found\_valid\_slot \leftarrow false$
        $selected\_core \leftarrow null$
        $selected\_partition \leftarrow null$
        for all core c in $cores\_sorted$ **do**
            if $found\_valid\_slot$ **then**
                break
            end if
            for all partition p in $partitions[c]$ **do**
                if $found\_valid\_slot$ **then**
                    break
                end if
                if $start\_time \geq p.start$ **and** $start\_time + W_i \leq p.end$ **then**
                    if $start\_time + W_i \leq deadline_{i,k}$ **then**
                        $can\_schedule \leftarrow true$
                        for $t \leftarrow start\_time$ **to** $start\_time + W_i - 1$ **do**
                            if $PPL[t] + P_{i,c} > TDP_{chip}$ **or** $P_{i,c} > TDP_{core}$ **then**
                                $can\_schedule \leftarrow false$
                                break
                            end if
                        end for
                        if $can\_schedule$ **then**
                            $selected\_core \leftarrow c$
                            $selected\_partition \leftarrow p$
                            $found\_valid\_slot \leftarrow true$
                            break
                        end if
                    end if
                end if
            end for
        end for
        if $found\_valid\_slot$ **then**
            $finish \leftarrow start\_time + W_i$
            Add $(start\_time, finish, T_{i,k})$ to $S[selected\_core]$
            for $t \leftarrow start\_time$ **to** $finish - 1$ **do**
                $PPL[t] \leftarrow PPL[t] + P_{i,c}$
            end for
            $p \leftarrow selected\_partition$
            if $start\_time > p.start$ **and** $finish < p.end$ **then**
                Remove p from $partitions[selected\_core]$
                Add $(p.start, start\_time)$ to $partitions[selected\_core]$
                Add $(finish, p.end)$ to $partitions[selected\_core]$
            else if $start\_time = p.start$ **and** $finish < p.end$ **then**
                Set $p.start$ to $finish$
            else if $start\_time > p.start$ **and** $finish = p.end$ **then**
                Set $p.end$ to $start\_time$
            else
                Remove p from $partitions[selected\_core]$
            end if
            return 1 # Successfully allocated
        else
            if $start\_time + W_i \leq deadline_{i,k}$ **then**
                return 0 # Failed to allocate within deadline
            end if
            $start\_time \leftarrow start\_time + 1$
        end if
    end while
end function