PROCESSOR VOLTAGE SCHEDULING FOR REAL-TIME TASKS WITH NON-PREEMPTIBLE SECTIONS

FAN ZHANG & SAMUEL T. CHANSON
DEPARTMENT OF COMPUTER SCIENCE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY CLEAR WATER BAY, KOWLOON, HONG KONG{ZHANGFAN,CHANSON} @CS.UST.HK

SPEAKER：吳俊逸 20210526

OUTLINE

1. Introduction
2. Static Blocking-aware Voltage Scheduling
3. Dynamic Blocking-aware Voltage Scheduling
4. Performance Evaluation
5. Conclusion

3 INTRODUCTION

- More and more personal computing and communication devices are becoming portable and mobile.Most of them are powered by batteries with limited power capacity.
- $P = C \times f \times (Vs)^2$
- The performance boost comes at the cost of higher energy consumption. The limited battery power has become a major concern.
- As the processor may not be fully utilized all the time, the variation in system load can be exploited to reduce power dissipation.
- In this paper, we shall refer to job scheduling with voltage scaling simply as "voltage scheduling" for short.

4

INTRODUCTION

- As observed in [2], the voltage transition delay is very short. We therefore assume the voltage transition cost is negligible and the voltage can be adjusted at any time (whether inside or outside a blocking section). We also assume the processor power follows formula (l), which in our case can be simplified to $P = K \times (Vs)^3$ where K is a constant.
- formula (l)：$P = C \times f \times (Vs)^2$

- [2] P. Pillai and K. G. Shin. "Real-time dynamic voltage scaling for low-power embedded operating systems". In Proceedings of the 18th ACM Symposium on Operating Systems Principles, pages 89-102, 2001.

5 STATIC BLOCKING-AWARE VOLTAGE SCHEDULING

- In the static scheme, the processor voltage is changed only when a new task arrives or when an existing task terminates.
- The objective is to find a minimum H(task set T at processor speed H) such that the inequalities are satisfied.
- If the re-calculated H exceeds 1, the newly arrived task is not admitted to the system and the original H value is restored.
- Otherwise the system will run at speed H until the value is changed again.

$$\forall k, 1 \leq k \leq n, \sum_{i=1}^{k} \frac{E_i \cdot \frac{1}{H}}{P_i} + \frac{B_k \cdot \frac{1}{H}}{P_k} \leq 1.$$

MultiMedia Lab NTUST EE

6 A DUAL-SPEED SWITCHING ALGORITHM

- The basic dual speed algorithm can be extended to further reduce energy consumption. This is achieved by shortening the lengths of high speed intervals. In the extension, a high speed interval can be terminated at once if one of the following conditions occurs:

  i) A job whose deadline is later than or equal to End_H starts execution,

  ii) The processor becomes idle.

A DUAL-SPEED SWITCHING ALGORITHM


```
/* H and L are recomputed by the static speed algorithm as a
task joins or leaves the system. Initially the processor speed is
L. End_H indicates the end of the high-speed interval. If the
system is not in a high speed interval, End_H = -1. Initially
End_H = -1. */
When job J_{i,j} arrives:
    if Priority(J_{i,j}) > Priority(current job)
        if Preempt_Current_Job() is successful
            Execute J_{i,j};
        else /* J_{i,j} is blocked*/
            Set_Speed(H); /*Set the processor speed at H*/
            End_H = max(End_H, d_{current_job});
            /* d is the deadline of the job */
        end if
    end if

When the end of high speed interval is reached:
    End_H = -1;
    Set_Speed(L); /*Set the processor speed at L*/
```

Figure 1. The dual speed (DS) algorithm.

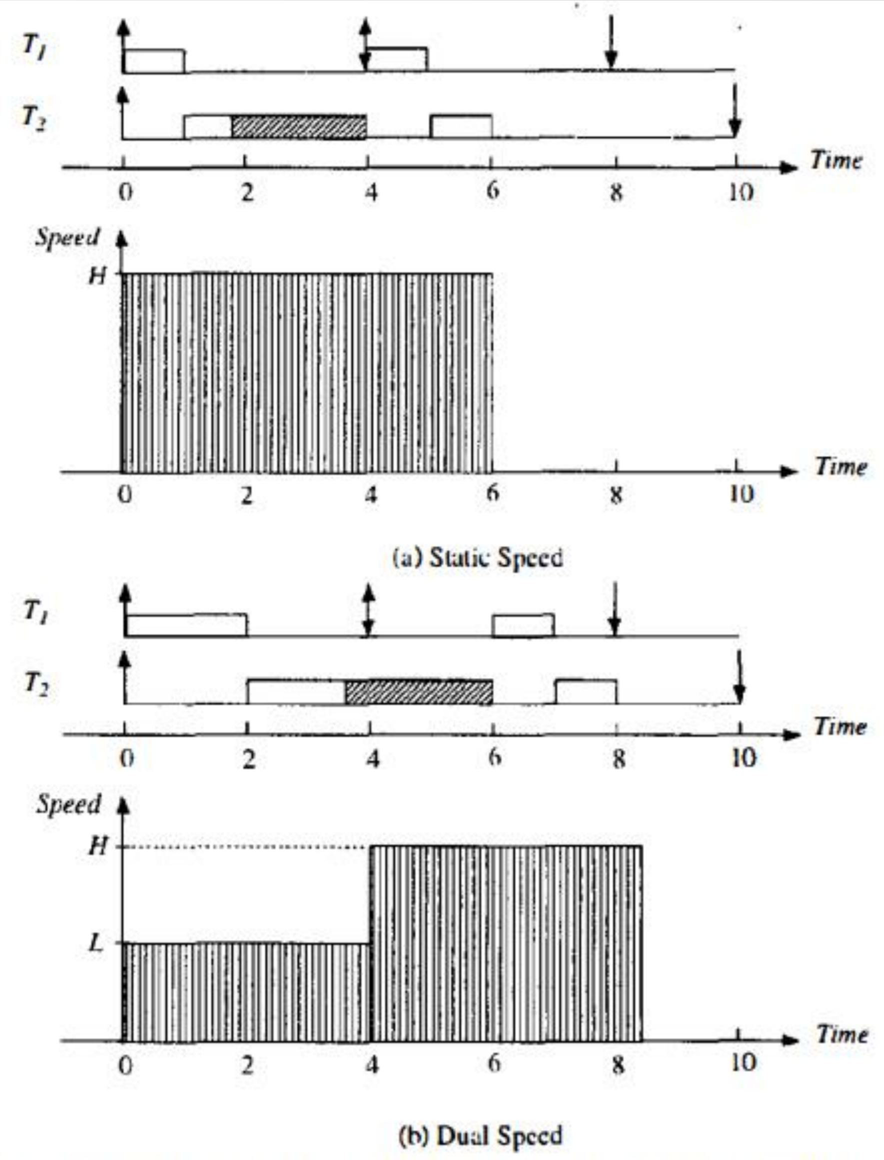

Figure 2. Comparison of static speed and dual speed algorithms.

A DYNAMIC RECLAIMING ALGORITHM

- When a job completes early, the processor would have some idle time. If this portion of time can be redistributed to the other pending jobs, the processor speed can be further reduced.
- The speed at which the processor should operate is $S = E \times Smax / R$

(E = execution time ； Smax = the maximum processor speed ； R = run time)

- The system maintains a Free Run Time list called the FRT-list to collect the run time not consumed. Similar to the CASH queue in [IS]
- In this way, DSDR effectively reclaims unused run time for redistribution, which in turn reduces the processor idle time and leads to decreased processor speed.

9 A DYNAMIC RECLAIMING ALGORITHM

Ji

- J_i: the current job of task T_i. (Since at any time each task can only have one job present, no ambiguity is introduced. Correspondingly, d_i indicates the deadline of job J_i.)

Rr

- $R_i^r(t)$: the available run time of job J_i at time t.

Rf

- $R_i^F(t)$: the run time in the FRT-list that can be used by job J_i at time t.

Et

- $E_i^r(t)$: the worst-case residue execution time of job J_i under the maximum speed $S_{\max}$ at time t.

When a new job (J_i) arrives:
$E_i^r(t) = E_i$;
$R_i^r(t) = E_i/L$;
if Priority(J_i) > Priority(current job)
if Preempt_Current_Job() is successful
Select J_i to run;
else /* the job is blocked */
$base\_speed = H$;
$End\_H = \max(End\_H, d_{current\_job})$;
Set_Speed(H);
end if
end if

When job J_i is selected to run:
if J_i is executed for the first time && $base\_speed == H$
/* first run reclamation */
Insert_To_FRT($R_i^r(t) - E_i/H$, End_H);
$R_i^r(t) = E_i/H$;
end if
Set_Speed($\frac{E_i^r(t)}{R_i^F(t)+R_i^r(t)}$);
Execute J_i;

When job J_i completes:
if $R_i^r(t) > 0$
Insert_To_FRT($R_i^r(t)$, d_i);
/*early completion reclamation*/
end if

When the end of high speed interval is reached:
$End\_H = -1$;
$base\_speed = L$;

Figure 3. The core DSDR algorithm.

PERFORMANCE EVALUATION

- Simulation Setup：
- We assume a maximum processor speed (=1) and a minimum processor speed (=0.1). Speed levels between the two bounds are discrete and spaced by 0.1.
- Generated tasks whose periods belonged to one of three ranges: long period (1000~5000ms), middle period (100~ 1000ms) and short period (20~100ms). The WCETs of the tasks in the three categories were (1~1000ms), (1~100ms) and (1~20ms), respectively.

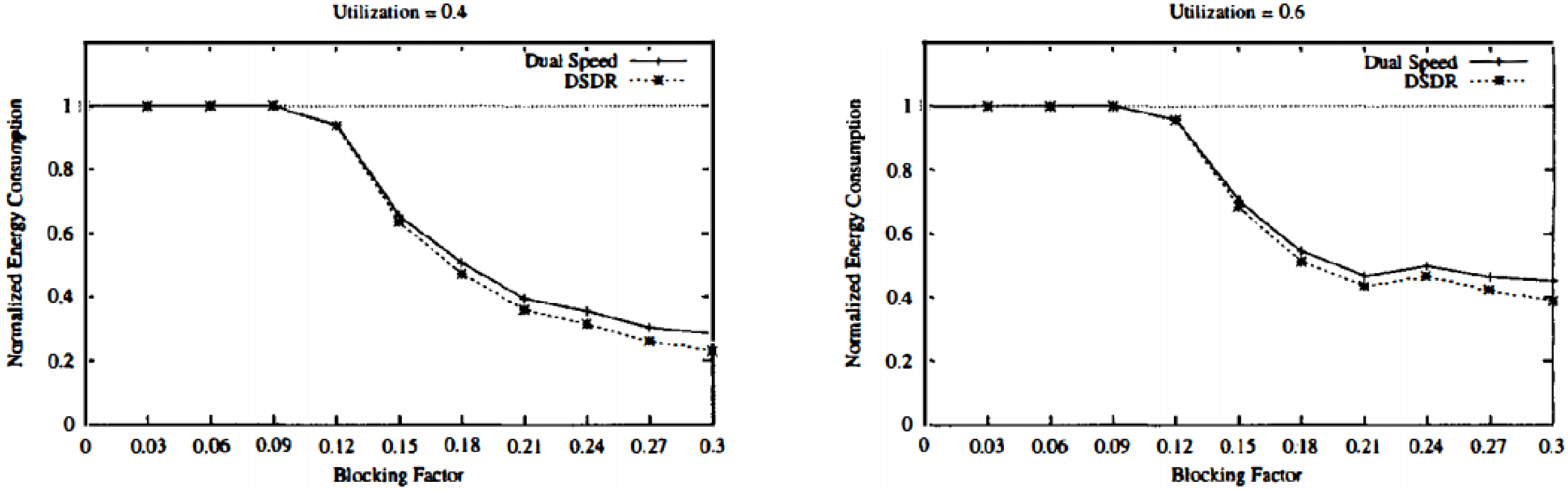

Figure 5. Normalized energy consumption with varying blocking factor.

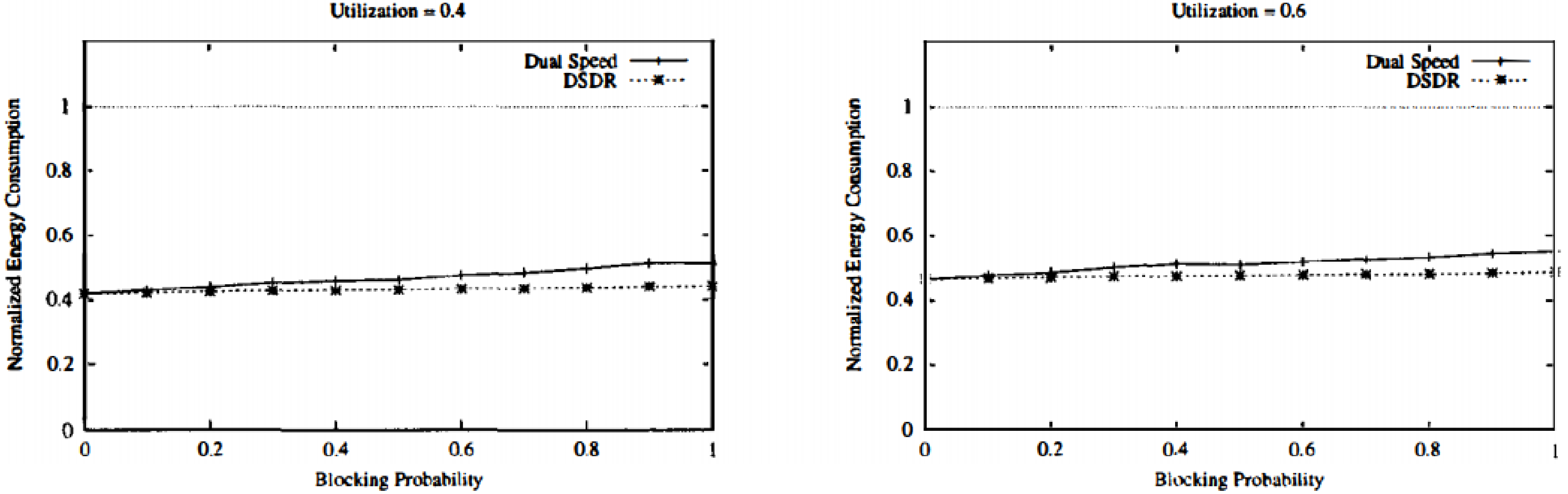

Figure 6. Normalized energy consumption with varying blocking probability.

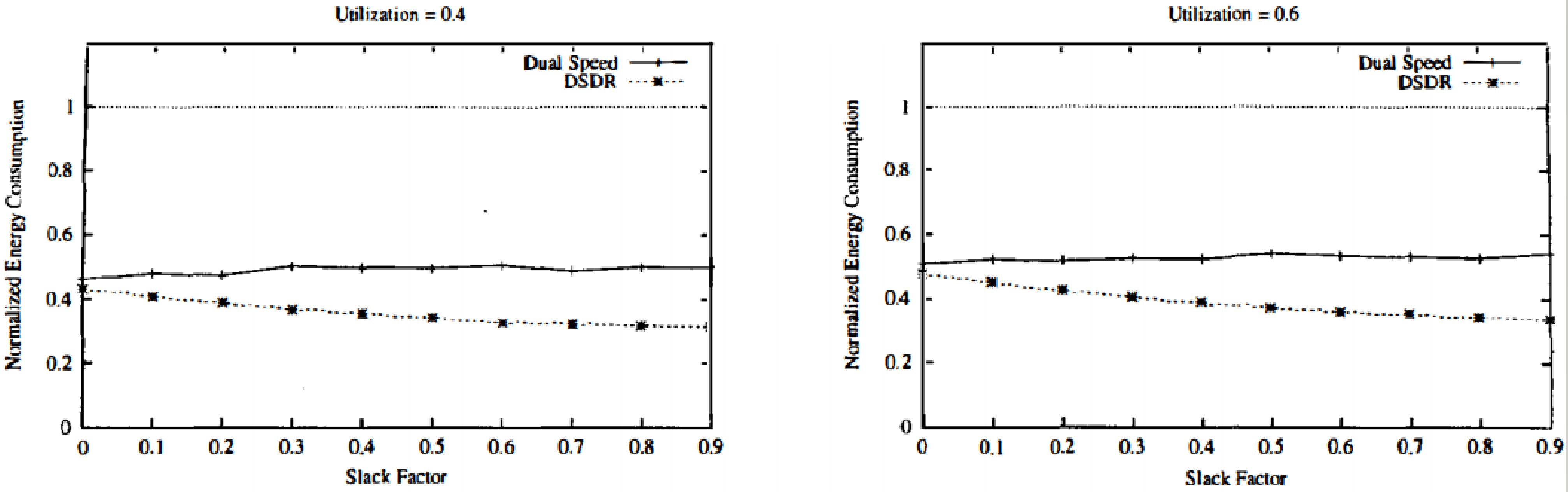

Figure 7. Normalized energy consumption with varying slack factor.

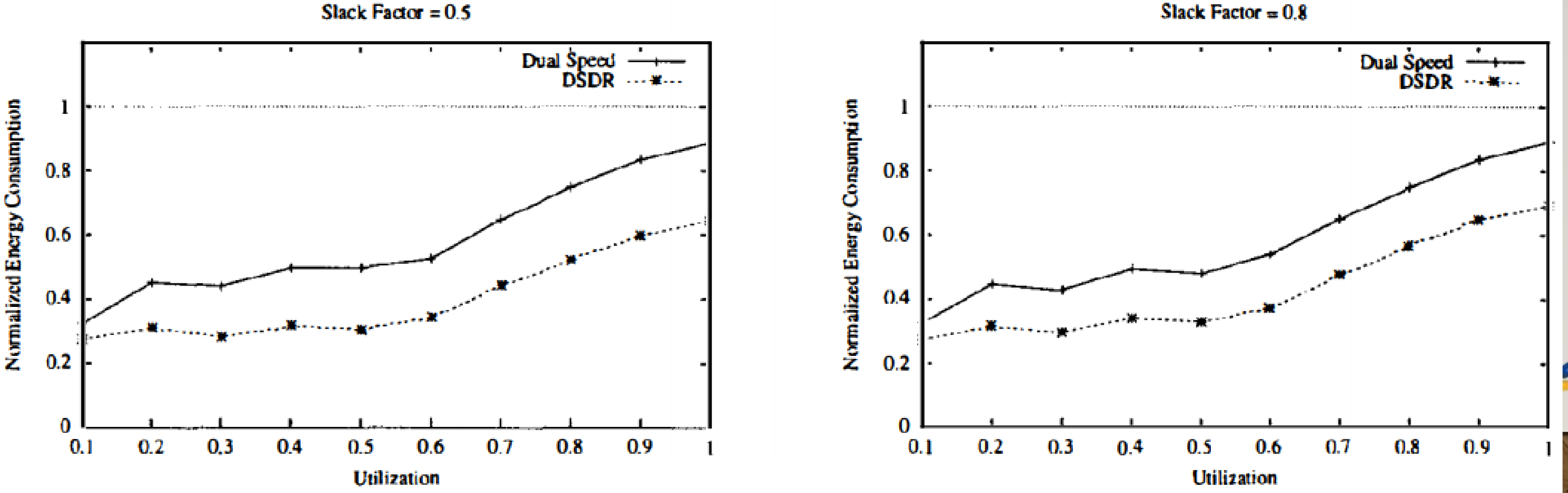

Figure 8. Normalized energy consumption with varying processor utilization.

13 CONCLUSION

1. The static speed scheme is based on the stack resource policy (SRP) [13] and calculates a minimal feasibly static speed.
2. Instead of always operating at one static speed, the dual speed algorithm lowers the processor speed to the utilization speed in some intervals.
3. A reclaiming mechanism is used to collect the unused run time and redistribute it to jobs that are able to make use of it.

- The results show that both dynamic algorithms can significantly reduce energy consumption compared with the static speed algorithm in all scenarios.

END

Thank you for listening