Politecnico di Torino

Energy Management for IoT

Dynamic Power Management

LAB 1 report

Master degree in Embedded Systems

Referents: Prof. Daniele Jahier Pagliari, Massimo Poncino

Authors: Group 2

Alessio Salta, Diamante Simone Crescenzo

November 1, 2022

Contents

1 Introduction

The main purpose of this lab is to understand how *dynamic power management* (DPM) policies work by using workloads related to different applications. In order to analyze energy savings a *DPM simulator* is provided. Such simulator implements a *power state machine* (PSM) with three states: **run**, **idle** and **sleep**. Each state has a certain power consumption, as well as transitions to another state. Transitions from **idle** to **sleep** state and vice-versa are not considered in the original implementation of the simulator. It is also worth mentioning that transitions have a cost. In particular, they need a certain time and some energy to be performed and this, together with the workload and the adopted DPM policy, can have a severe impact on power savings.

2 DPM simulator and testing

2.1 Workload analysis

Workloads are necessary to properly run the DPM simulator. In this case, they come as textual files in which data about *arrival time* and *duration* (in ms) of different *work items* are specified in each row. The two provided workloads include items of similar duration but with different arrival times ($4ms$ intervals for workload 1 and $100ms$ intervals for workload 2), but both of them are organized in a similar manner: the sequence of work items is repeated with a period of $120s$. Within these iterations the arrival times of the different work items are set as described above. In this way the timeout policy is able to save a significant amount of energy by driving the system to the **idle** state, unless the timeout value is set to really high values (i.e. very close, equal, or greater than the workload period).

2.2 DPM testing

2.2.1 Part1 (*run-idle* states)

The *part 1* of the analysis uses the *timeout policy* of the DPM simulator, which in this case cycles between **run** and **idle** states. When in the **idle** state, the machines consumes a power of $0.57mW$. In this configuration the system can only perform **run-idle** transitions (and vice-versa) and each of them requires $400\mu s$ and $10\mu J$ of energy.

2.2.2 Part2 (*run-sleep* states)

The simulator can be modified by forcing transitions to the **sleep** state. In this condition the system consumes $90\mu W$ of power only. By doing so, the system can exclusively perform **run-sleep** transitions (and vice-versa). However, transitioning among **run** and **sleep** states requires more time and energy. In particular, going from **sleep** to **run** requires $4ms$ and $2mJ$ of energy which means 10 times more time and 200 times the energy with respect to a **idle-run** transition, meanwhile going from **run** to **sleep** requires $1ms$ and $20\mu J$ of energy which means 2.5 times more time and twice the energy with respect to a **idle-run** transition. The slight code change can be observed in the code section below.

2.2.3 Automatizing the Analysis Process

In order to automatize the simulation process, we have implemented a small script (available in this section) which performs the following operations:

- Creates (or clears) a textual file, with a proper name to distinguish which states of the PSM are being used, in which the results of each simulation run will be stored. A pretty simple data file format has been selected based on two columns, the first one containing the actual value of T_{TO} which is being applied to the policy and the second storing the corresponding energy consumption.
- Executes a *fine-grained* simulation by using timeout values in the range $[0 - 999]$, with intervals of $1ms$. This choice comes from the need to observe the non-linear behavior of the power consumption when the chosen T_{TO} is lower than the interval between two work items.

- Executes a *coarse-grained* simulation by using timeout values from $1000ms$ to $130000ms$, with intervals of $1000ms$ between two consecutive simulations. This is needed to capture a wider picture of the overall power consumption of the system.
- Extracts the timeout and the energy consumption values obtained with the DPM timeout policy and saves them to dedicated files, in order to simplify further data elaborations. The script takes in input a string of user's choice, which will be added to the standard filename, plus the mode in which the timeout policy is running: *ri* (for **run** to **idle** transitions), and *rs* (for **run** to **sleep** transitions).

Saved data are then used to implement time-energy graphs by means of Matlab scripts to allow their analysis. These scripts are very similar one another, in fact only labels, input file names, workload reference, and other minor parameters change, so in the script section only the one used in the first analysis is reported, but the remaining ones can be easily derived.

2.2.4 Part1 and Part2 Discussion

Results coming from the simulations show that the *timeout policy* is able to save a considerable amount of energy even for high timeout values: for example, by choosing a $T_{TO} = 60s$ the energy consumption is still half of the one obtained without a DPM policy. As expected, the consumed energy increases linearly with the timeout value. The timeout-energy curve shows different behaviours depending on the timeout value T_{TO}: for values smaller than the arrival times of two consecutive work items (called T_{cons} for simplicity)[1] the slope of the curve is steeper making the timeout policy more effective.
Furthermore, if T_{TO} is set equal to (or greater than) the idle period between two consecutive iterations[2] the energy saved with the policy is zero.
Figure 3 shows the trend of energy savings for **run-idle** transitions (for workloads 1 and 2). The behaviour reflects the one observed while analyzing data related to the energy consumption.

Focusing on the **sleep** state, as expected the energy consumption increases almost linearly with the timeout value, while the normalized saved energy decreases following a similar trend. Results are visible in figures 4 and 5. One interesting aspect is the following: when T_{TO} is slightly smaller than the interval between two consecutive work items there is a local minima in the saved energy curve, after which the function rises again. By analyzing experimental results a local maxima can be found when T_{TO} is slightly above T_{cons}. Two quantities mainly concur to this behaviour:

- *Number of transitions*: if we increase T_{TO} there will be less transitions between the two states, and this reduces the energy for transitions.
- *Total time in state **run***: the higher T_{TO} the more energy the system will consume, due to a bigger period in which it will be in the **run** state.

For certain T_{TO} the energy for transitions gives a contribution that makes the saved energy equal to cases in which we have less transitions but the total time in state **run** is higher (higher T_{TO}). This is more evident for workload 2.

2.2.5 Extra: *run-idle-sleep* states

The timeout policy used for Part1 and Part2 could be further improved by allowing both **run-idle** and **run-sleep** transitions in order to obtain even better energy savings. The main focus here has been how to switch from a low power state (i.e. **idle**) to an even lower one (i.e. **sleep**), since the PSM under analysis does not allow a direct transition among them. The solution to this issue is to switch back to the *run* state before going to *sleep* when starting from *idle*. Here some key aspects are worth to be mentioned:

- Two timeout values are used in this modified policy, namely the *timeout* T_{TO} and the *timeout-sleep* T_{TOS}. Given the system entering an inactivity period, T_{TO} is the time after which the DPM will force it to **idle** (in the same way the original timeout policy does), meanwhile after T_{TOS} expires the system will be forced to **sleep**.

[1] $T_{cons} = 4ms$ for workload 1 and to $100ms$ for workload 2
[2] refers to an entire workload cycle of $120s$

- This policy can be set through the *-tis* custom argument when running the DPM simulator followed by both T_{TO} and T_{TOS} values.
- Whenever T_{TOS} expires, the system is temporarily brought back to the **run** state while setting the *idle_sleep_flag*. The latter is crucial since the system needs to be aware that the **run** state is not linked to an activity period but it is just part of a transition towards the **sleep** state. The **run-idle-sleep** policy and the *idle_sleep_flag* handler function are highlighted in the code section below.
- The double transition described in the previous point is very expensive in terms of time and energy with respect to a simple one, therefore some accurate analysis must be carried out to decide a proper value for both timeouts.

The power analysis of this policy has been again carried out by means of scripts. The easiest (brute force) approach to gather data would have been running a nested loop which would simulate the system with all T_{TO} and T_{TOS} values in the range of interest (0-130000ms). Besides feasible, this approach is not smart nor useful to our purposes. By exploiting the results obtained in the previous phases of the lab, we selected the T_{TO} which yielded the best power saving, namely 0ms. At this point, only an iteration over T_{TOS} values has been necessary and the fine/coarse grained approach of the previous sections has been reused.

Results for *workload 1* reveal that the best sleep timeout is $T_{TOS} = 4ms$ for which the energy consumption is about 24.2% better with respect using **run-sleep** transitions only, and about 28.3% better with respect using **run-idle** transitions only. For T_{TOS} lower than 4ms, the transition to **sleep** is not worth due to the waste of energy caused by frequent transitions in the sampling phase (i.e. T_{BE} is not respected). On the other hand, for T_{TOS} larger than 4ms and up to 116s power consumption increases linearly with respect to the timeout. When T_{TOS} is between 116s and 119s, the energy cost is actually greater than the **run-idle** policy because the transition to **sleep** still occurs but not for long enough to actually reabsorb it. Finally, when T_{TOS} is greater than or equal to 120s, the policy degenerates simply into the run-idle one.
For what concerns *workload 2* the same discussion still holds except for the specific timeout values. It can be observed that a more spread distribution is obtained because this workload contains a bit more heterogeneous work items.
A graphical plot of this analysis can be appreciated in figures 6 and 7.

2.2.6 Part3 (Predictive Policy)

The *part 3* of the experimental measures with the DPM simulator is related to the set of *predictive policies*; in particular, the analysis focuses on the *threshold policy* (code can be found below), implemented by using two adaptive thresholds. The approach followed in this part is similar to the one adopted in *part 1* and *part 2*: a small script, which is very similar to the one used for the two previous parts, is used to automatize simulations. In details, it performs the following activities:

- Executes a simulation in which the only variable parameter is the **idle** threshold, selected in a range from $1ms$ up to the chosen value for the **sleep** threshold, with intervals of $1ms$. The **sleep** threshold value is fixed and it is determined by analyzing the provided workloads. In particular, the workload 1 threshold is $50ms$ and the workload 2 one is of $30ms$. The choice of this values is described in the discussion section below.
- Executes a simulation with a fixed **idle** threshold of $1ms$ and a variable **sleep** threshold. In this case, the range for the **sleep** threshold is user-defined, with steps of $1ms$ each, starting from a value for which $T_{threshold,sleep} > T_{threshold,idle}$.
- Extracts the timeout and the energy consumption values obtained with the DPM history policy and saves them to dedicated files, in order to simplify further data elaborations. The script takes in input a value which can be either 1 or 2 used to tell the simulator which *workload* to use (and which will be added to the standard filename), plus a set of values for regression-based[3] and threshold policies.

[3]The regression-based policy was not exploited in this session.

Saved data are then used to elaborate time-energy graphs through the usage of Matlab scripts in order to visualize the behavior corespondent to different approaches.

2.2.7 Part3 Discussion

Data extracted from the simulation demonstrate that by keeping the **sleep** threshold fixed and progressively increasing the **idle** one, the energy consumption related to workloads 1 and 2 becomes constant after a certain value. In particular, the **sleep** threshold has been fixed to $50ms$ for workload 2, and so the **idle** threshold is in the range [1-50] *ms*. As for workload 1 the above mentioned threshold has been set to *30*ms. These values are based on the analysis of the two workloads: short work items are usually followed by elements of the same type after a very small interval; long work items, instead, are followed by considerably long idle periods. For both workloads, work items whose duration is in the range [1-5]ms are followed by another work item in a relatively short time; if the duration is outside this interval it is very likely that a long idle period will follow. Results of the above mentioned analyses are visible in figures 8 and 9. We can notice that in workload 1, for very low sleep thresholds, the energy consumption is pretty high if compared with bigger values. This is due to the time needed for the **sleep-idle** transition ($4ms$ for the adopted PSM), which in some cases is more than the idle time between two consecutive work items. For what concerns workload 2, instead, the idle time between two short work items is around $100ms$, so it is possible to save a significant amount of energy this time by fixing a really low sleep threshold instead. The approach followed in this situation was to set the **idle** threshold to $1ms$ when testing a variable **sleep** threshold and the **sleep** to $30ms$ when testing the **idle** one; in this way it is possible to exploit idleness between consecutive work items and to go in *sleep* state when a longer idle period arrives.

2.3 Summary

The *threshold* policy is able to save a considerable quantity of energy but if compared to other policies it can show, of course, advantages and disadvantages. With respect to *timeout* policies we can observe that, for both **run-idle** and **run-sleep** transitions, the *threshold* policy is more stable for both workloads, in the sense that the energy consumption follows a **step** function which means that it remains constant within certain intervals of timeout. On the other hand, the *timeout* policy linearly increases the energy consumption together with the timeout figure. In fact, the saved energy curve for timeout policies experiences a linear decrease for most of the tested $T_{TO}s$, while for *threshold* the curve is again a step function. The *timeout* policy including all the transitions (**run-idle-sleep**) experiences the latter behavior as well.

2.3.1 Workload 1 Summary

The obtained results of minimum *consumed energy* figures for the first workload are the following:

- $698\mu J$ for the **run-idle** *timeout* policy with T_{TO} of $0ms$
- $202\mu J$ for the **run-sleep** *timeout* policy with T_{TO} of $4ms$
- $197\mu J$ for the **run-idle-sleep** *timeout* policy with T_{TO} of $0ms$ and T_{TOS} of $4ms$
- $197\mu J$ for the **run-idle-sleep** *threshold* policy with $T_{threshold,idle}$ of $1ms$ and fixed $T_{threshold,sleep}$ of $50ms$
- $197\mu J$ for the **run-idle-sleep** *threshold* policy with fixed $T_{threshold,idle}$ of $1ms$ and $T_{threshold,sleep}$ from $32ms$ up to $180ms$

The *threshold* policy is able to achieve the minimum energy consumption. For this workload, in particular, the *timeout* policy including all possible states produces the same result obtained with the above mentioned policy, this because the best result is obtained with a T_{TOS} of $4ms$ which prevents the PSM to go to the **sleep** state for a too short inactive time.

2.3.2 Workload 2 Summary

The obtained results of minimum *consumed energy* figures for the second workload are the following:

- $693\mu J$ for the **run-idle** *timeout* policy with T_{TO} of $0ms$
- $386\mu J$ for the **run-sleep** *timeout* policy with T_{TO} of $0ms$
- $195\mu J$ for the **run-idle-sleep** *timeout* policy with T_{TO} of $0ms$ and T_{TOS} of $118ms$
- $193\mu J$ for the **run-idle-sleep** *threshold* policy with $T_{threshold,idle}$ of $1ms$ and fixed $T_{threshold,sleep}$ of $30ms$
- $193\mu J$ for the **run-idle-sleep** *threshold* policy with fixed $T_{threshold,idle}$ of $1ms$ and $T_{threshold,sleep}$ from $5ms$ up to $229ms$

The *threshold* policy is able to achieve the minimum energy consumption once again. For workload 2, the last implemented policy is even slightly better than the *timeout* policy including all possible states.

2.3.3 Conclusions

As highlighted and analyzed in the data plots and in the summaries, the best energy saving is achieved by the *threshold based history policy*. The reasons of this result achieved by our analysis resides in the high degree of determinism related to the two analyzed workloads, being related to a very cyclic embedded-system-like scenario. By studying the evolution of arrival and duration times of all the work items it has been possible to properly tune the thresholds in order to converge towards the most convenient power state at each given inactivity period. It is worth mentioning that our history-based policy is not a dynamic one, it is highly correlated to the previously mentioned analysis, so it cannot be generally extended to any system but only to the subset of those which have a well defined and cyclic sequence of tasks. This choice comes from the availability of very specific workload descriptions, which could even verify in a real application if we only think to a custom sensing system, and we focused our attention in finding the most effective energy saving solution more than just a generic one.

2.4 Bash Scripts

2.4.1 DPM run-idle-sleep script

```
#!/bin/bash
#simple script for automatizing simulation of the timeout DPM policy

# simple progress bar function
prog() {
    local w=45 p=$1 tot=$2;  shift
    # create a string of spaces, then change them to dots
    printf -v dots "%*s" "$(( $p*$w/$tot ))" ""; dots=${dots// /.};
    # print those dots on a fixed-width space plus the percentage etc.
    printf "\r\e[K|%-*s| %3d %% %s" "$w" "$dots" "$p" "$*";
}

# prints usage info on terminal
usage_info() {
    echo " Usage: ./dmp_script_wl1.sh [destination file tag] [setting]"
    echo "      -destination file tag"
    echo "          string to be attached to default filename"
    echo "      -setting"
    echo "          ri  : run-idle setting"
    echo "          rs  : run-sleep setting"
    echo "          ris : run-idle-sleep setting (extra)"
}

if [[ $# -lt 2 ]]
then
    echo "[ERROR]: missing argument, 2 needed."
    usage_info
    exit 1
fi

# reads input argument to set correct destination filename
if [[ $2 = "ri" ]]
then
    # RUN-IDLE POLICY setup
    dest_filename="dpmres_wl${1}_run_idle.txt";
elif [[ $2 = "rs" ]]
then
    # RUN-SLEEP POLICY setup
    dest_filename="dpmres_wl${1}_run_sleep.txt";
elif [[ $2 = "ris" ]]
then
    # RUN-IDLE-SLEEP POLICY setup (extra)
    dest_filename="dpmres_wl${1}_run_idle_sleep.txt";
else
    echo "[ERROR]: missing or wrong argument."
    usage_info
    exit 1
fi

#declare the number of iterations
fine_iter=1000
coarse_iter=130
```

```
# builds simulation files
make

# starts fine-grained simulation cycles
printf "\n"
echo -n "" > $dest_filename
for ((i=0; i < $fine_iter; i++));
do
    prog $((i+1)) $fine_iter Fine-grained phase
        echo -n "$((i)) " >> $dest_filename
    if [[ $2 = "ris" ]]
    then
        ./dpm_simulator -tis 0 $((i)) \
          -psm example/psm.txt -wl ../workloads/workload_${1}.txt | \
          grep -o "Energy w DPM = "[0-9]*.[0-9]* |\
          grep -o "[0-9]*.[0-9]*$" >> $dest_filename
    else
        ./dpm_simulator -t $((i)) \
          -psm example/psm.txt -wl ../workloads/workload_${1}.txt |\
          grep -o "Energy w DPM = "[0-9]*.[0-9]* | \
          grep -o "[0-9]*.[0-9]*$" >> $dest_filename
    fi
done

printf "\nDone.\n\n"

# starts coarse-grained simulation cycles
for ((i=1; i <= $coarse_iter; i++));
do
    prog "$i" $coarse_iter Coarse-grained phase
    echo -n "$((i*1000)) " >> $dest_filename
    if [[ $2 = "ris" ]]
    then
        ./dpm_simulator -tis 0 $((i*1000)) \
          -psm example/psm.txt -wl ../workloads/workload_${1}.txt | \
          grep -o "Energy w DPM = "[0-9]*.[0-9]* | \
          grep -o "[0-9]*.[0-9]*$" >> $dest_filename
    else
        ./dpm_simulator -t $((i*1000)) \
          -psm example/psm.txt -wl ../workloads/workload_${1}.txt | \
          grep -o "Energy w DPM = "[0-9]*.[0-9]* | \
          grep -o "[0-9]*.[0-9]*$" >> $dest_filename
    fi
done

printf "\nDone.\n\n"
```

2.4.2 DPM history-based script

```
#!/bin/bash
# simple script for automatizing simulation of the
#  history-based DPM policy

# simple progress bar function
prog() {
    local w=45 p=$1 tot=$2;  shift
    # create a string of spaces, then change them to dots
    printf -v dots "%*s" "$(( $p*$w/$tot ))" ""; dots=${dots// /.};
    # print those dots on a fixed-width space plus the percentage etc.
    printf "\r\e[K|%-*s| %3d %% %s" "$w" "$dots" "$p" "$*";
}

# prints usage info on terminal
usage_info() {
    echo "  WARNING: currently the only supported history policy is \
    the threshold one"
    echo "  Usage: ./dpm_hystory_script.sh [destination file tag] \
    [regression coefficients] [thresholds]"
    echo "      -wokload and destination tag"
    echo "          string to be attached to default filename"
    echo "          and denoting the workload under analysis"
    echo "      -regression coefficients"
    echo "          numeric values for history policy using \
    non-linear regression"
    echo "          (currently supporting only 1 value)"
    echo "      -thresholds"
    echo "          numeric values for threshold-based \
    predictive policy"
    echo "          (currently 2 values are required)"
}

#check number of arguments provided
if [[ $# -lt 4 ]]
then
    echo "[ERROR]:  missing arguments, 4 required."
    usage_info
    exit 1
fi

#clear destination file if already existent
> dpmres_history_idle_threshold_wl_$1.txt
> dpmres_history_sleep_threshold_wl_$1.txt

#declare number of iterations
sleep_threshold=300
idle_threshold=50

#build simulation files
make

#call the dpm_simulator with a variable idle threshold
#the idle threshold is incremented by one until
#it reaches the value of the sleep threshold
```

```
for (( i = 1; i <= $4; i++))
do
    prog "$i" $4 Idle threshold phase
        echo -n "$((i)) " >> dpmres_history_idle_threshold_wl_$1.txt
        ./dpm_simulator -h $2 $((i)) $4 \
        -psm example/psm.txt -wl ../workloads/workload_$1.txt | \
        grep -o "Energy w DPM = "[0-9]*.[0-9]* |\
         grep -o "[0-9]*.[0-9]*$" >> \
         dpmres_history_idle_threshold_wl_$1.txt
done

#call the dpm_simulator with
#variable sleep_thresholds
for i in {1..300..1}
do
    if [ $i -gt $3 ]
    then
        prog "$i" $sleep_threshold Sleep threshold phase
        echo -n "$((i)) " >> dpmres_history_sleep_threshold_wl_$1.txt
        ./dpm_simulator -h $2 $3 $((i)) \
         -psm example/psm.txt -wl ../workloads/workload_$1.txt | \
        grep -o "Energy w DPM = "[0-9]*.[0-9]* | \
        grep -o "[0-9]*.[0-9]*$" >> \
        dpmres_history_sleep_threshold_wl_$1.txt
    fi
done

printf "\nDone.\n\n"
```

2.5 Customized C code

This section highlights the portions of code which have been modified for the purpose of our analysis.

2.5.1 DPM Timeout RUN-IDLE or RUN-SLEEP

```
case DPM_TIMEOUT:
        /* Day 2: EDIT */
        if (t_curr >= t_inactive_start + tparams.timeout)
        {
            //*next_state = PSM_STATE_IDLE;
            *next_state = PSM_STATE_SLEEP;
        }
        else
        {
            *next_state = PSM_STATE_RUN;
        }
        break;
```

2.5.2 DPM Timeout RUN-IDLE-SLEEP

```
// DPM Timeout Policy which uses both IDLE and SLEEP states
    case DPM_TIMEOUT_IDLE_SLEEP:
        if (t_curr >= t_inactive_start + tparams.timeout_sleep &&
             prev_state == PSM_STATE_IDLE)
        {
            // when SLEEP timeout expires and system is in IDLE state
                // goes back to RUN state before SLEEP
            *next_state = PSM_STATE_RUN;
            // sets IDLE-RUN-SLEEP transition flag
            idle_sleep_flag = 1;
        }
        else if (t_curr >= t_inactive_start + tparams.timeout_sleep &&
                  prev_state != PSM_STATE_IDLE)
        {
            // when SLEEP timeout expires and system is NOT
            //  in IDLE state anymore
            *next_state = PSM_STATE_SLEEP; // goes to SLEEP
                    // resets IDLE-RUN-SLEEP transition flag
            idle_sleep_flag = 0;
        }
        else if (t_curr >= t_inactive_start + tparams.timeout)
        {
            // when IDLE timeout expires goes to IDLE
            *next_state = PSM_STATE_IDLE;
        }
        else
        {
            // if no timeout is expired keeps system in RUN state
            *next_state = PSM_STATE_RUN;
        }
        break;
```

2.5.3 IDLE-RUN-SLEEP Transition Handler

```
// when IDLE-RUN-SLEEP transition flag is set, adds an additional call
//  to dpm_decide_state() method
    // to force the transition back to RUN state before SLEEP state
    if (idle_sleep_flag)
    {
        if (!dpm_decide_state(&curr_state, prev_state, t_curr,
            t_inactive_start, history, sel_policy, tparams,
            hparams))
        {
            printf("[error] cannot decide next state!\n");
            return 0;
        }
        if (curr_state != prev_state)
        {
            if (!psm_tran_allowed(psm, prev_state, curr_state))
            {
                printf("[error] prohibited transition!\n");
                return 0;
            }
            // takes into account transition overheads
            //  in term of energy
            psm_energy_t e_tran = psm_tran_energy(psm,
                                  prev_state, curr_state);
            psm_time_t t_tran = psm_tran_time(psm,
                                  prev_state, curr_state);
            n_tran_total++;
            e_tran_total += e_tran;
            e_total += e_tran;
            t_tran_total += t_tran;
            t_curr += t_tran;
        }
        else
        {
            // spend one simulation time-step in the current
            //  state, then re-evaluate
            psm_time_t time_unit =
                        SIMULATION_TIME_STEP / PSM_TIME_UNIT;
            e_total +=
                psm_state_energy(psm, curr_state, time_unit);
            t_curr += time_unit;
            t_state[curr_state] += time_unit;
            // time spent in RUN while there was no work
            //  to be done
            if (curr_state == PSM_STATE_RUN)
                t_waiting += time_unit;
        }
    }
```

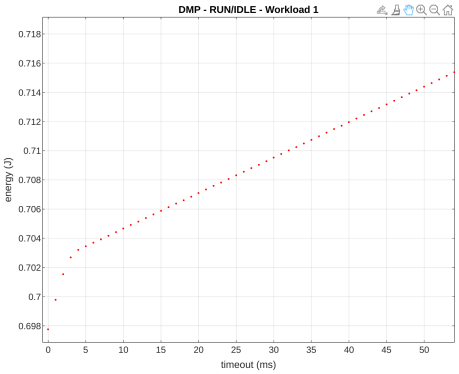
2.5.4 DPM History Threshold

```
case DPM_HISTORY:
        if (history[0] > hparams.threshold[1])
        {
            // when the last active time is greater than the highest
            //  threshold ("sleep" threshold) go to sleep (a long idle
            //  interval has came up)
            *next_state = PSM_STATE_SLEEP;
        }
        else if (history[0] >= hparams.threshold[0] &&
                  history[0] <= hparams.threshold[1])
        {
            // go to idle if the last active time is between the
            //  highest threshold and the "idle" threshold
            *next_state = PSM_STATE_IDLE;
        }
        else
        {
            // remain in run state
            *next_state = PSM_STATE_RUN;
        }
        break;
```

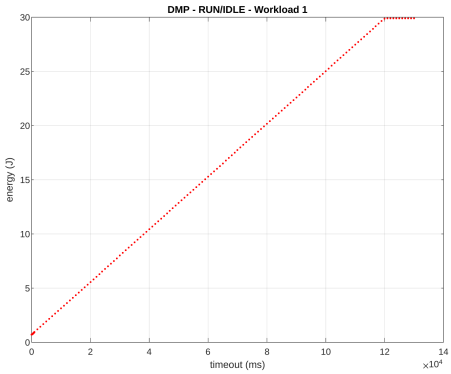
2.6 Matlab Data Plots

```
%% DPM - WORKLOAD 1 - RUN/IDLE
% data extraction from file
DPM_data = dlmread('dpmres_wl1_run_idle.txt');
timeout = DPM_data(:,1);
energy = DPM_data(:,2);

% timeout vs consumed energy plot
figure(1)
plot(timeout, energy, 'r.', 'LineWidth', 4)
xlabel("timeout (ms)");
ylabel("energy (J)");
grid on;
title("DMP - RUN/IDLE - Workload 1")

% timeout vs normalized saved energy plot
TOTAL_ENERGY = 29.8790976000;   % energy for WL1 w/o DPM
saved_energy_normalized = zeros(size(energy));

for i = 1:size(energy)
    saved_energy_normalized(i) =
        ((TOTAL_ENERGY-energy(i))/TOTAL_ENERGY)*100;
end

figure(2)
plot(timeout, saved_energy_normalized, 'g.', 'LineWidth', 4)
xlabel("timeout (ms)");
ylabel("normalized saved energy (%)");
grid on;
title("DMP - RUN/IDLE - Workload 1")
```

2.7 Data Plots

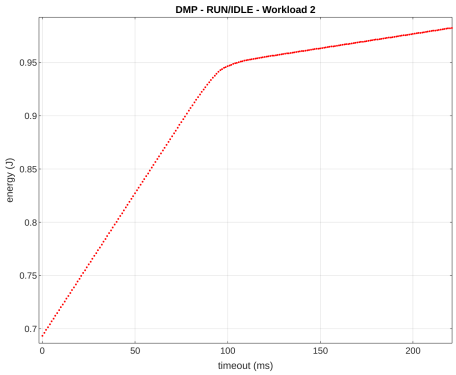


(a) *Initial section of the timeout-energy curve for run-idle transitions (workload 1)*

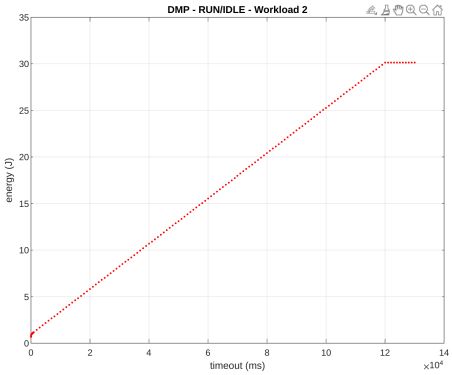


(b) *Global view (workload 1)*

Figure 1: Timeout-energy curve for workload 1 (run-idle transitions)

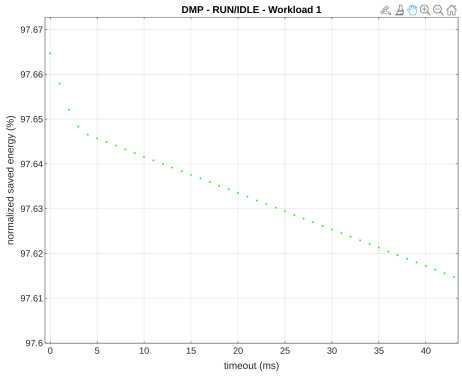


(a) *Initial section of the timeout-energy curve for run-idle transitions (workload 2)*

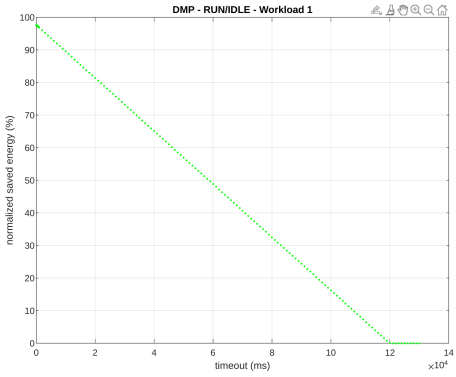


(b) *Global view (workload 2)*

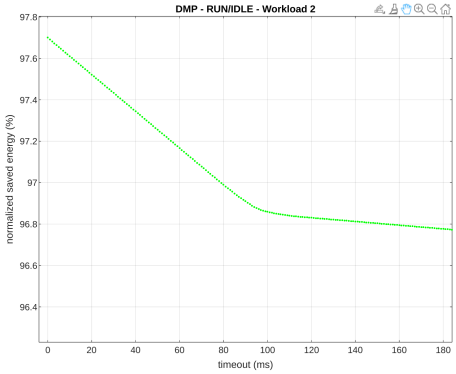
Figure 2: Timeout-energy curve for workload 2 (run-idle transitions)



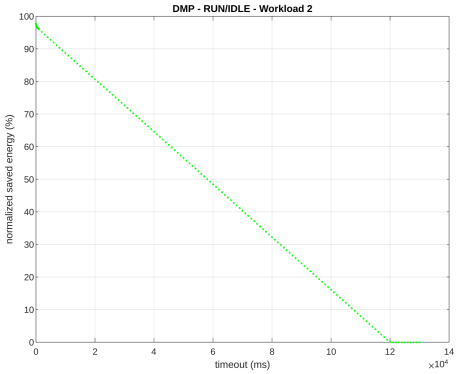
(a) *Initial section of the timeout-saved energy curve for run-idle transitions (workload 1)*



(b) *Global view (workload 1)*

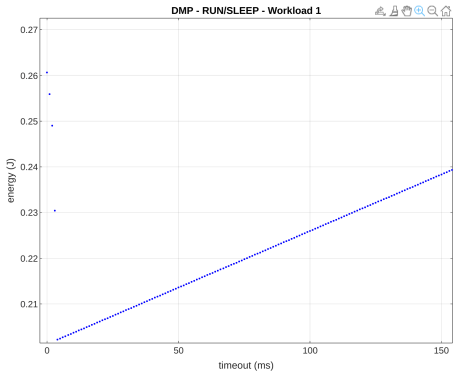


(c) *Initial section of the timeout-saved energy curve for run-idle transitions (workload 2)*

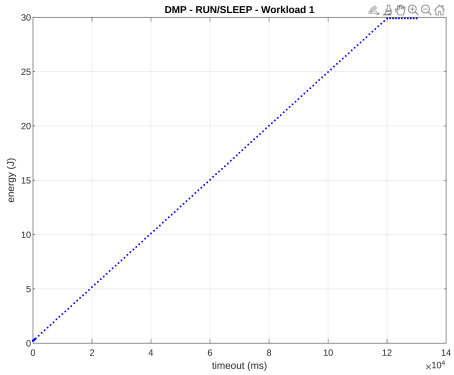


(d) *Global view (workload 2)*

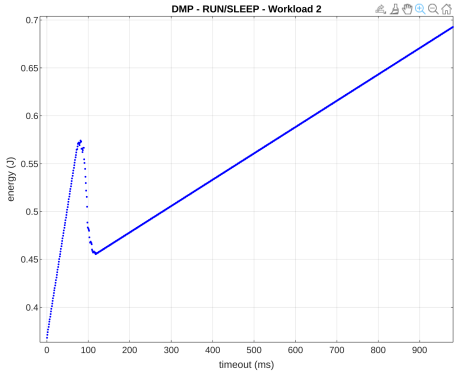
Figure 3: Timeout-saved energy curve for workload 1 and 2 (run-idle transitions)



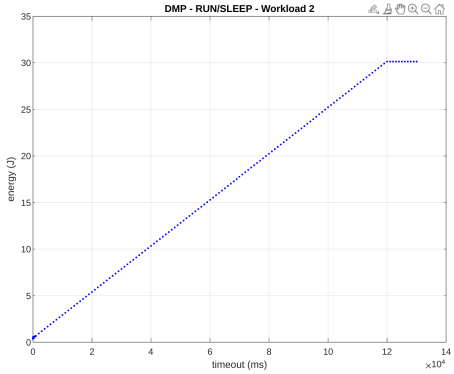
(a) *Initial section of the timeout-energy curve for run-sleep transitions (workload 1)*



(b) *Global view (workload 1)*

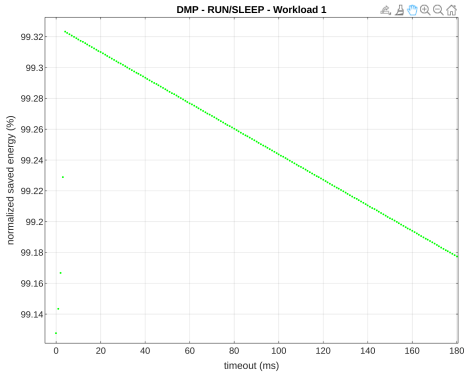


(c) *Initial section of the timeout-saved energy curve for run-idle transitions (workload 2)*

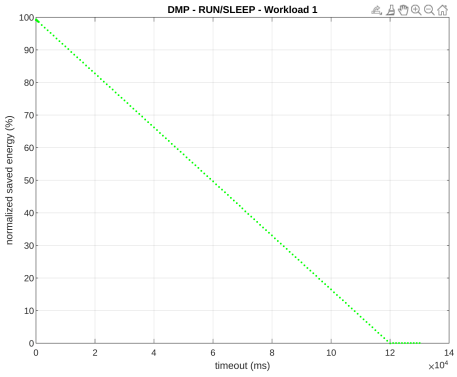


(d) *Global view (workload 2)*

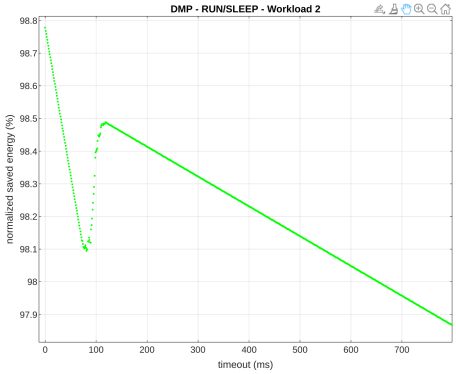
Figure 4: Timeout-energy curve for workload 1 and 2 (run-idle transitions)



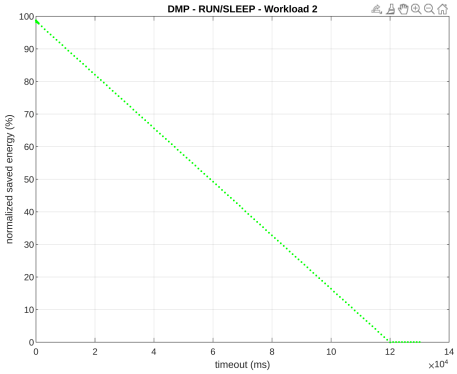
(a) *Initial section of the timeout-saved energy curve for run-sleep transitions (workload 1)*



(b) *Global view (workload 1)*

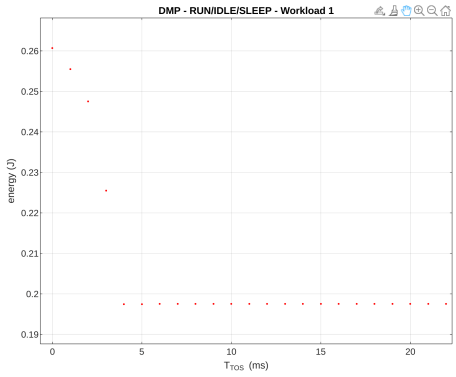


(c) *Initial section of the timeout-saved energy curve for run-sleep transitions (workload 2)*

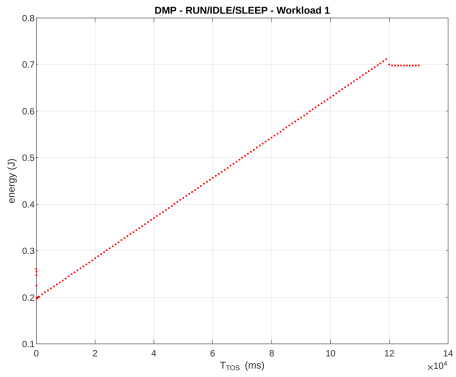


(d) *Global view (workload 2)*

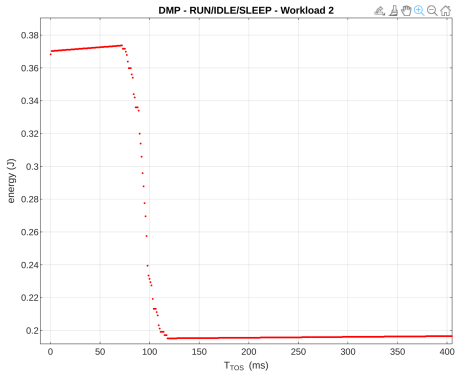
Figure 5: Timeout- normalized saved energy curve for workload 1 and 2 (run-sleep transitions)



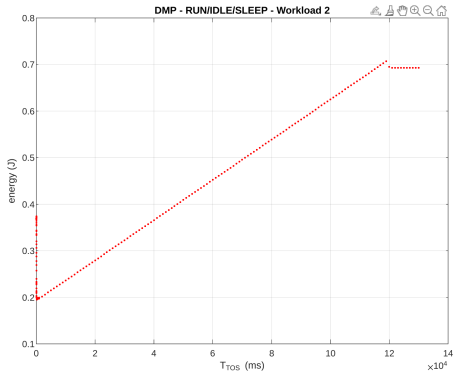
(a) *Initial section of the T_{TOS}-energy curve for run-idle-sleep transitions (workload 1)*



(b) *Global view (workload 1)*

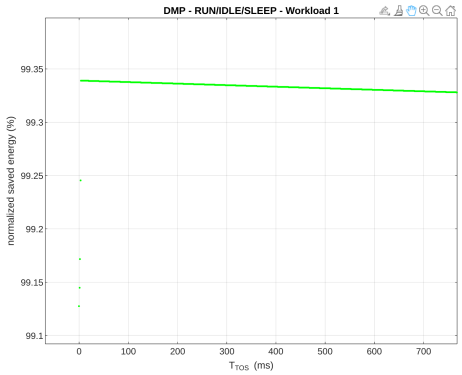


(c) *Initial section of the T_{TOS}-saved energy curve for run-idle-sleep transitions (workload 2)*

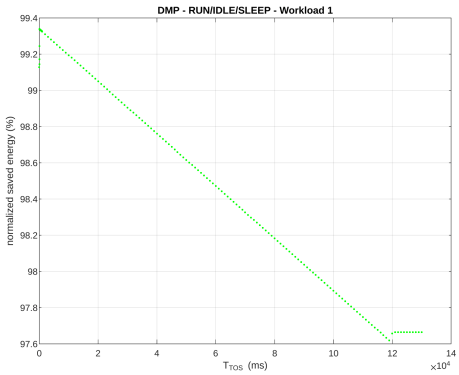


(d) *Global view (workload 2)*

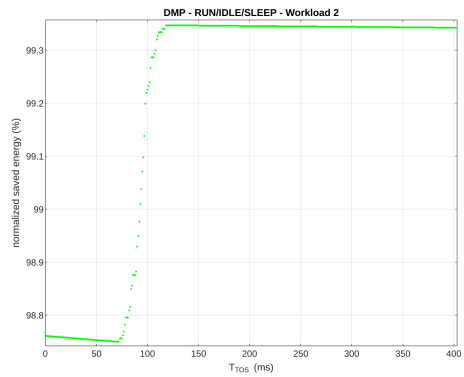
Figure 6: T_{TOS}-energy curve for workload 1 and 2 (run-idle-sleep transitions)



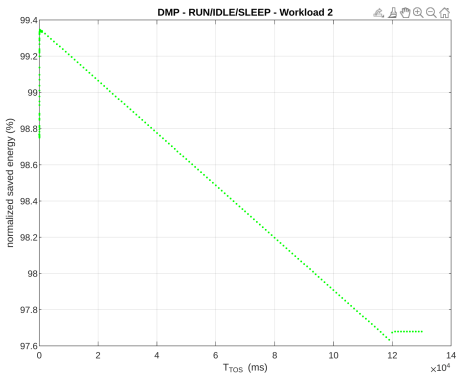
(a) *Initial section of the T_{TOS}-saved energy curve for run-idle-sleep transitions (workload 1)*



(b) *Global view (workload 1)*

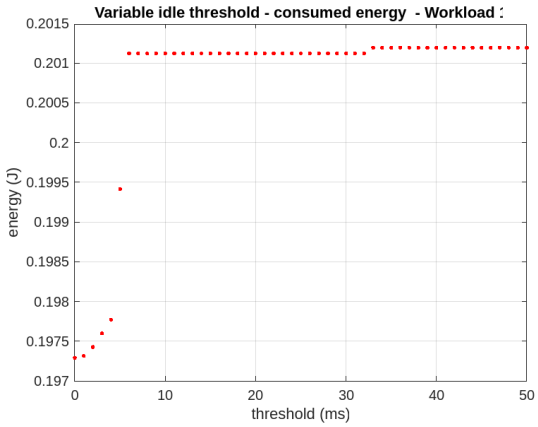


(c) *Initial section of the T_{TOS}-saved energy curve for run-idle-sleep transitions (workload 2)*

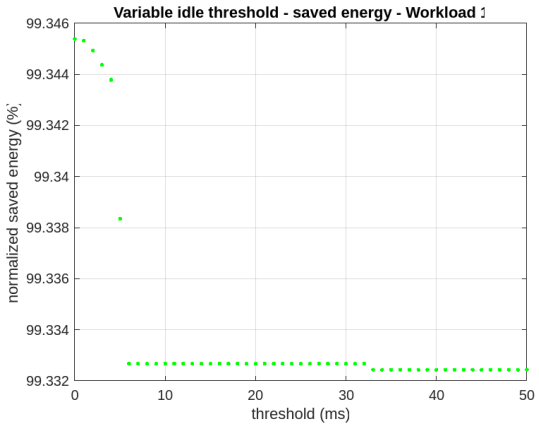


(d) *Global view (workload 2)*

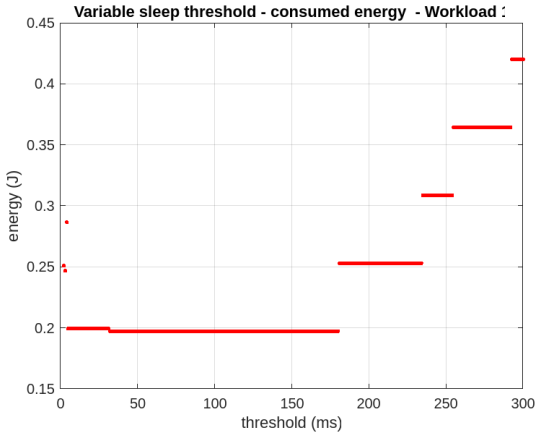
Figure 7: T_{TOS}-normalized saved energy curve for workload 1 and 2 (run-idle-sleep transitions)



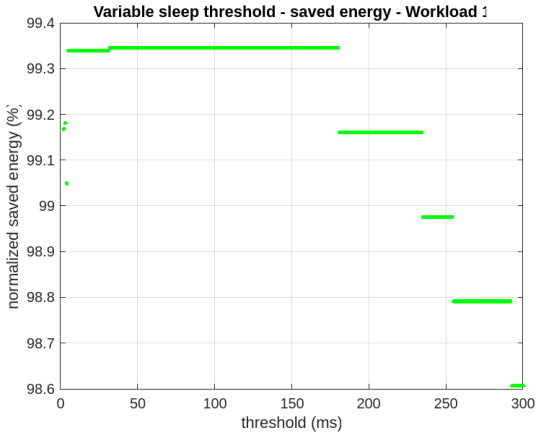
(a) *threshold-energy curve for a variable idle threshold (workload 1)*



(b) *Normalized saved energy curve for a variable idle threshold (workload 1)*

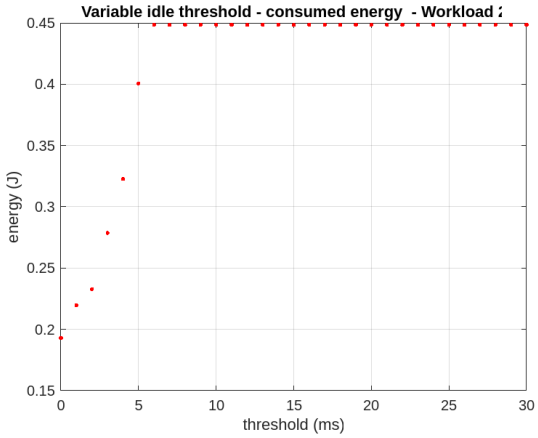


(c) *threshold-energy curve for a variable sleep threshold (workload 1)*

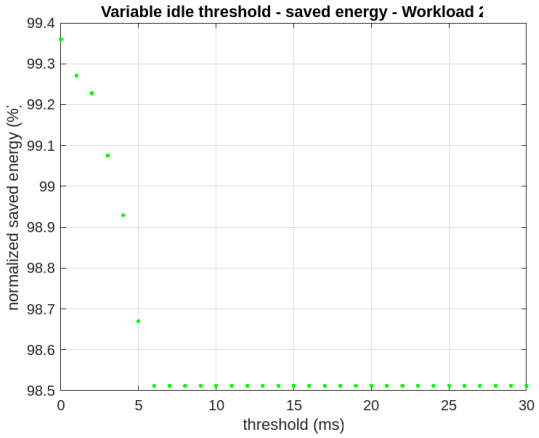


(d) *Normalized saved energy curve for a variable idle threshold (workload 1)*

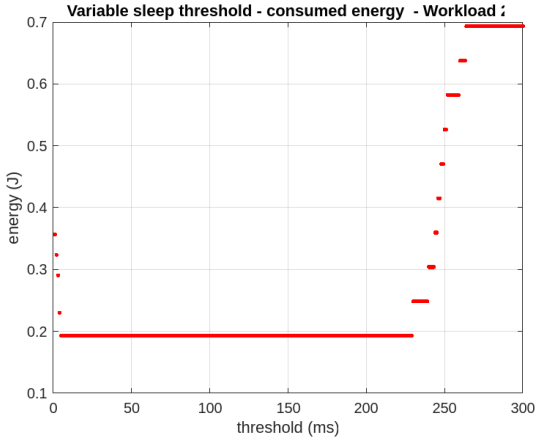
Figure 8: threshold-energy and threshold-saved energy curves for workload 1



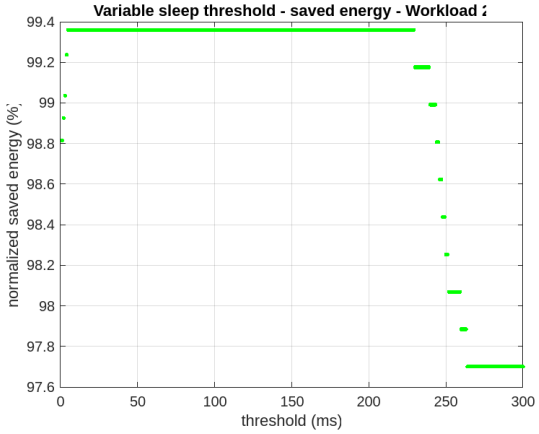
(a) *threshold-energy curve for a variable idle threshold (workload 2)*



(b) *Normalized saved energy curve for a variable idle threshold (workload 2)*



(c) *threshold-energy curve for a variable sleep threshold (workload 2)*



(d) *Normalized saved energy curve for a variable idle threshold (workload 2)*

Figure 9: threshold-energy and threshold-saved energy curves for workload 2