Finite state machine-based DRAM power management with early resynchronisation

J.H. Park and Y. Chu

Abstract: An efficient operating system-based power management scheme for DRAM in the multi-programming/time-sharing environment is presented. Formulas for evaluating the condition of positive energy gain are developed and a finite state machine (FSM) for selecting the best power mode for a given idle time is designed. In the proposed scheme, the scheduler selectively assigns the most efficient power mode to each idle memory bank at context switching time based on the FSM. For computing the idle time, two efficient and practical prediction methods are developed and tested for performance. The proposed scheme achieves further energy saving by starting the resynchronisation of idle memory banks as early as possible. In the aspects of the prediction method and the early resynchronisation method, multiple versions of the proposed scheme are developed and tested for performance. The proposed scheme utilises events occurring at context switching time in the multiprogramming/time-sharing environment and the scheduling ensures the maximum energy gain without significantly degrading the performance. The proposed scheme is tested with a simulated system, and the experimental results demonstrate the efficiency of the scheme. In the experiment, the energy gain by using the proposed scheme ranges from 17.84% to 52.21% depending on the time quantum sizes tested.

1 Introduction

In the modern computing era, mobile, hand-held and embedded systems that depend on the battery power are widely used in many application areas, and the demand for the reduced power consumption has been increased. Low-power design techniques are also applicable to the systems that do not depend on the battery power because the heat reduction also is an important issue. Among several system components that are able to employ power reduction techniques, we focus on the main memory system (DRAM) since it is a major power consuming component of a computing system. Recent studies show that memory could consume 50% more power than processor in real server systems [1, 2] and the power consumption of embedded systems heavily depends on the DRAM architecture and management scheme [3–6]. In fact, DRAM can occupy more than 50% of the total power dissipation in some embedded systems [4, 6].

Together with low-power cache design techniques [7–14] and power-efficient on-chip memory system design techniques that replace the cache with a small on-chip SRAM module [15–17], methods for reducing the power consumption of DRAM modules have been proposed in the literature [2, 10, 18–25]. The efforts are mainly based on the recent technology that makes a DRAM module operate in multiple power modes. For instance, Direct Rambus DRAM (RDRAM) technology provides four different power modes: active, standby, napping and power-down [26]. Each power mode consumes different amount of energy, and lower power modes require resynchronisation costs in energy and time to be activated for read and write operations. As the basic rationale, turning down the power of unused hardware components can obviously save the system's power dissipation, but the components should be turned down only when the idle time is long enough for positive energy gain. The goal of the research is to utilise the manufacturer provided multiple power mode technology for achieving maximum energy gain.

In DRAM power mode management research [2, 19–25], memory bank partitioning method has been used to utilise the technological advantages of manipulating multiple power modes for turning down unused banks' power. Approaches for selecting efficient power modes for partitioned memory banks are grouped into compiler-based, hardware-assisted and operating system-based methods with their own advantages and disadvantages. Among them, the operating system-based method is known as the most flexible and does not require high costs [2, 20–24]. Since operating system-based power management schemes are flexible and do not require high costs, they have also been used for reducing power consumption of I/O devices and CPU. Some of the approaches can be found in [27–36].

In this paper, we propose an efficient operating system-based DRAM power management scheme that selectively transits power modes of memory banks based on a FSM during run time. In addition to the power mode change strategy (selecting the best mode from multiple power modes), the proposed scheme achieves further energy gain by using the methods of starting the resynchronisation as early as possible for reducing the resynchronisation cost. Since the scheme is generic which does not aim a specific commercial DRAM model or a specific platform, it can be applied to various systems including general purpose

© The Institution of Engineering and Technology 2007
doi:10.1049/iet-dtt:20060113
Paper first received 28th July 2006 and in revised form 28th February 2007
J.H. Park is with the Department of Computer Science, State University of New York, New Paltz, NY 12561, USA
Y. Chu is with the Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS 39762, USA
E-mail: park@cs.newpaltz.edu

systems and embedded systems. The proposed scheme reflects the multiprogramming/time-sharing environment in which context switching occurs among multiple processes by assuming the system in which all pages of a process are allocated into one memory bank; that is sequential first-touch page allocation [21]. In this paper, we use the term DRAM interchangeably with RDRAM. The contributions of our work are as follows: (i) utilisation of events occurring at context switching time; (ii) development of the best power mode selection scheme based on a FSM; (iii) development of efficient and practical prediction methods for computing idle time; and (iv) development of early-resynchronisation methods that reduce the resynchronisation costs.

The remainder of this paper is organised as follows. Background and related works are described briefly in Section 2. Conditions for achieving the positive energy gain and a FSM for DRAM power mode transition are provided in Section 3. The proposed power-aware scheduling algorithm and early resynchronisation methods are described in Sections 4 and 5, respectively. Experimental results from simulations are presented in Section 6, and Section 7 concludes the paper.

2 Background and related work

2.1 RDRAM power modes

Recent RDRAM technology makes each memory chip to be activated independently [26]. This implies that researchers can develop efficient power-saving methods for the memory system without sacrificing the bandwidth [19]. In general, memory bank partitioning method has been used to utilise the technological advantages of manipulating multiple power modes for reducing the energy consumption of RDRAM modules [2, 19–25]. Since there appeared variations of unit energy consumption values of RDRAM power modes and resynchronisation costs in the literature [2, 19, 23, 26], we use an abstract model derived from the commercial RDRAM model used in [19] for our research. Fig. 1 shows this abstract RDRAM power model with energy and time values. In the figure, arrows represent the abstract energy transition directions, and ° energy values are negligible costs. Values shown in Fig. 1 are normalised to constants x and c. In fact, 8 MB RDRAM module consumes 3.57 nJ per cycle in the active mode in 3.3 V, 2.5 ns clock cycle time technology [19]. According to [19], we set constants, x and c, in Fig. 1 as 3.57 and 2 clock cycles, respectively. RDRAM is known for delivering fast bandwidth performance (1.6 GB/s) per watt – three times higher than PC100 SDRAM and even more (about 10 times) than conventional DRAM [37–39]. In the Mobile PC market, RDRAM has been used as main memory since it can be optimised for low cost, low power, higher performance and smaller space/volume [38].

In this paper, we use two assumptions as follows: (i) the memory system consists of smaller modules in rows (banks) and columns; and (ii) each bank consists of one RDRAM module that is operated in four different operation modes as shown in Fig. 1.

2.2 Related work

Efforts for reducing the power consumption of DRAM by selectively turning down memory bank’s power are found in the literature in the aspects of compiler-based, hardware-assisted and operating system-based.

Delaluz et al. [19] proposed a compiler-based method for reducing power on DRAM modules (banks) by determining idle periods of each memory module at compile time. Although this method does not have resynchronisation cost overhead since it completely predicts the memory bank usage at compile time for the target data, it has several drawbacks. The scheme is conservative since all information are not available/analysable at compile time; only arrays and loop constructs are involved in the practice, and only single programming case is considered. Thus, the approach handles very limited situations and it assumes the direct mapping of address to the physical memory (i.e. without virtual memory space). Udayakumaran et al. [25] proposed a compiler-based scheme in which data allocation and memory partitioning were performed jointly. It uses application-specific partitioning of the scratchpad memory based on the temporal locality.

Self-monitored run-time approaches that need hardware support were proposed in [19]. The idea is that the memory system automatically transits to lower power modes based on the information captured by the supporting hardware. Three schemes are proposed based on adaptive threshold value (ATP), constant threshold value (CTP) and history of inter-access time (HBP). Among them, the history-based approach showed the best performance. The HBP method can transit the power mode of a bank directly from active to a lower power mode, by simply assuming that the next inter-access time is the same as the previous inter-access time.

Since the compiler-based approaches have certain limitations and the hardware-oriented approaches need additional energy consumptions on those supporting hardware for checking bank usage, alternative methods based on the operating system were discussed in the literature [2, 20–24]. We briefly review some representative approaches in this section. Lebeck et al. [23] proposed a scheme for reducing DRAM power consumption by a power-aware page allocation strategy. Underlying idea is the clustered page allocation in which an application’s pages are allocated into a minimum number of chips (banks) and unused banks are in low-power modes during the execution of the application. The approach is able to cooperate with hardware support for determining each bank’s power state either statically or dynamically. Similar to [23], Li et al. [2] proposed an improved approach in the literature. Two major contributions of the approach are eliminating the need for the application-specific parameter tuning and providing a tool for avoiding an excessive amount of performance degradation by stopping the power saving. Approaches proposed in [2, 23] consider only single programming case and there was no consideration of multiprogramming situations; that is aspects related to the occurrences of context switching. Delaluz et al. [20] proposed a scheduler-based approach that uses a bank usage table to predict and reset all banks’ power modes for each process at context switching time.
The mechanism is based on the idea of temporal locality; that is the used banks during the previous time quantum might be used again during the next time quantum. When a process is rescheduled, the scheduler activates all banks that were used in the previous time quantum to avoid resynchronisation costs. This approach considers only two power modes (active and low) and might have a considerable amount of mispredictions since the prediction is based only on the prior usage of banks. In general, operating system-based DRAM power management schemes can be used with hardware-oriented methods [20, 23] or other operating system-based power management schemes such as dynamic voltage scaling for CPU. Fan et al. [40] showed an improved performance from using a combined power management scheme for both DRAM and CPU, comparing to the individual usage.

3 Positive energy gain for DRAM module

In this section, we develop formulas for computing power consumption of DRAM modules and describe the process of creating a FSM for selecting the best power mode for a given idle time.

3.1 Power consumption ratio

The simplest power-saving scheme for DRAM banks is turning down unused bank’s power to the lower mode if there exist only two power modes, namely active and low. But the consideration is whether the turning down to the low mode actually yields energy gain or not. In the case of reusing the idle bank in a very short period of time, turning down the bank’s power mode brings about more power consumption because of the resynchronisation cost that is the cost of reactivating the bank for read and write operations.

In the ideal case in which the resynchronisation starts in advance with no latency overhead (refer to Fig. 6a in Section 5), the total power consumption for one idle period (in a lower power mode) is

$$T_iE_i + T_i(E_a - E_i)$$

which in turn is

$$(T_i - T_r)E_i + T_iE_a$$

(1a)

where $T_i$ is the bank’s idle time, $T_r$ is the resynchronisation cost time, $E_i$ is the unit energy consumed in the lower mode and $E_a$ is the unit energy consumed in the active mode. During the resynchronisation, the unit energy cost is $E_a$. Thus, when $(T_i > T_r)$ energy gain by using a lower power mode during the idle time in each memory bank is

$$T_iE_a - ((T_i - T_r)E_i + T_iE_a)$$

which in turn is

$$(T_i - T_r)(E_a - E_i)$$

(1b)

In the case of considering the resynchronisation latency (refer to Fig. 6b in Section 5), the total power consumption for one idle period (in a lower power mode) is

$$T_iE_i + T_rE_a$$

(2a)

and the energy gain by using a lower power mode during the bank’s idle time is

$$T_iE_a - (T_iE_i + T_rE_a)$$

which in turn is

$$T_i(E_a - E_i) - T_rE_a$$

(2b)

Thus, when $T_i > T_r (E_a/(E_a - E_i))$ a positive energy gain is expected.

In the ideal case in which the resynchronisation starts early enough for having no latency, the ratio of power consumption by using a lower power mode over using single power mode (always active) is

$$\frac{(T_i - T_r)E_i + T_iE_a}{T_iE_a} = (E_i/E_a) + (1 - E_i/E_a)(T_i/T_r)$$

(3)

It is obvious that the ratio is less than 1 when $T_i > T_r$. In the case of considering the latency, the ratio is

$$\frac{T_iE_i + T_rE_a}{T_iE_a} = (E_i/E_a) + (T_r/T_i)$$

(4)

As described earlier, the boundary idle time is $T_i(E_a/(E_a - E_i))$ and when $T_i$ is greater than the boundary idle time the power consumption ratio is less than 1 and the gain is $T_i(E_a - E_i) - T_rE_a$, as shown in formula (2b).

Fig. 2 illustrates the relationship between the power consumption ratio and $T_i$ values for both ideal and latency cases, based on the RDRAM power model shown in Fig. 1 and formulas (3) and (4). In the figure, only three RDRAM power modes (active, standby, napping) are used, and the power-down mode is excluded in the illustration since the values of the power-down mode are far beyond the values of other three modes. Fig. 2 demonstrates how mistakenly selected power modes affect the energy gain negatively. Fig. 2 also shows the power consumption ratio gap between the ideal case and the latency case.

As illustrated in Fig. 2, with all $T_i$ values we can save maximally $(T_iE_i)/(T_iE_a)$ power consumption ratio, which is obtained by subtracting (3) from (4), by starting the resynchronisation as early as possible. For example, by using active and napping modes, when the idle time is 20c the gap is 6.67% of power consumption ratio; that is 6.67% better power consumption ratio without the resynchronisation latency. This is the motivation of our efforts for reducing the gap by using the proposed early resynchronisation scheme that is described in Section 5.

3.2 FSM for power mode selection

By considering only two power modes such as active and low modes, the boundary idle time for the positive energy gain for the ideal case can be easily found from Fig. 1 since it is the resynchronisation cost from the low mode
to the active mode. For example, by using only active and napping modes, we have positive energy gain when $T_i > 15c$. However, in the case of using more than two power modes, we need to compute boundary idle times for all pairs of the power modes. In this paper, we use term break-even idle time ($T_{be}$) for the boundary idle time from which positive energy gain is obtained. Computation of $T_{be}$ between two power modes $L_1$ and $L_2$ is based on solving the following equation for $T_i$

$$
(E_{L_1}/E_4) + (1 - E_{L_1}/E_4)(T_{ei}/T_i)
$$

$$=(E_{L_2}/E_4) + (1 - E_{L_2}/E_4)(T_{ei}/T_i)
$$

where $E_{L_1}$ and $E_{L_2}$ represent the unit energy consumed in mode $L_1$ and $L_2$, respectively, and $T_{ei}$ and $T_{ei}$ represent the resynchronisation cost (time) of mode $L_1$ and $L_2$, respectively. Values of $E_{L_1}$, $E_{L_2}$, $T_{ei}$ and $T_{ei}$ can be found from the RDRAM power model shown in Fig. 1. Analogously, in the case of considering the resynchronisation latency the computation is based on solving the following equation for $T_i$

$$
(E_{L_1}/E_4) + (T_{ei}/T_i) = (E_{L_2}/E_4) + (T_{ei}/T_i)
$$

The resulting $T_{be}$ graph, which is used for finding the best power mode for a given $T_i$ value, is shown in Fig. 3. In the figure, values shown on edges are $T_{be}$ values between two connected nodes (power modes) in the ideal case in which the resynchronisation does not cause extra time overhead. $T_{be}$ values for the latency case in which the resynchronisation starts after $T_i$ expires are also shown in the figure.

Using the break-even graph, we can extract the information of selecting the best power mode, which yields the biggest energy gain, for a given idle time. Since there are six $T_{be}$ values in Fig. 3, there exist seven regions each of which has the different order of finding the best and the worst power modes. In the ideal case, for instance with $(1c < T_i \leq 15c)$, standby is the best power mode and active, napping and power-down follow; with $(15c < T_i \leq 91.453c)$, standby is the best power mode and napping, active and power-down follow; with $(91.453c < T_i \leq 4500c)$, napping is the best power mode and standby, active and power-down follow, and so on. However, only the best power mode for each region is needed in our research, and Fig. 4 illustrates the seven regions and the best power mode for each region. In the figure, $t_1, \ldots, t_6$ represent $T_{be}$ values, and actual values shown are for the ideal case. Actual $T_{be}$ values for the latency case can be found in Fig. 3.

Based on $T_{be}$ values shown in Fig. 3, the best power mode in each region shown in Fig. 4 and the power mode transition directions shown in Fig. 1, we can build a FSM for DRAM power mode transition as depicted in Fig. 5. In the figure, unused edges are not shown and the edges with dotted lines (e and f) are possible transition directions for the further enhancement of the scheme, while it is not used in practice. In the FSM, state transition from active to one of four modes occurs when a bank becomes idle (e.g. a process releases CPU) and state transition from one of four modes to active occurs when a bank becomes active (e.g. a process is dispatched). Detailed power mode selection process using the FSM is presented in Section 4.

**Fig. 4** Best power modes for regions divided by $T_{be}$ values

**Fig. 5** FSM for power mode transition

---

**Fig. 3** Break-even idle times

**Fig. 5** FSM for power mode transition

---

4 Context switching time power management

In this section, we propose an efficient context switching time DRAM power management scheme by using the FSM that we developed in the previous section. Context switching is a key feature in the multiprogramming/time-sharing system where many decisions are made and system states are changed at that time. We utilise the events occurring at context switching time to alter power states of memory banks to reduce the power consumption. We assume that the memory array is divided into smaller banks and unused banks are power-downed during the entire period of the current time quantum (or, until the processor is pre-empted based on the event). Thus the scheme is coarse-grained; that is during the current time quantum (or, until CPU pre-emption) no power transitions occur. Since we can implement the scheme by only operating system’s modules, this approach does not require special hardware support. In our system model, processes are loaded into contiguous memory spaces. As mentioned in Section 1, the underlying assumption is the allocation of
compute the current waiting time ($T_w$):

\[ T_w = T_{\text{disp}} - T_{\text{pre-emption}} \]

update accumulated avg. waiting time ($T_{\text{avg}}$):

\[ T_{\text{avg}} = \frac{T_{\text{avg}} \times (n-1) + T_w}{n}; \quad n = \text{number of times this process has gained CPU so far} \]

set future idle time ($T_i$) of the process:

\[ T_i = T_{\text{avg}} \]

$Bi$’s power mode is Active;

at context switching-out time,

select the best power mode:

on FSM, new-state = $\delta$(Active, $T_i$);

power mode of switching-out process = new-state;

from all processes ($P_1 \ldots P_m$) in $B_i$, $/m$ is the number of processes:

find $P_i$ which has the highest priority, based on the ready-queue information;

$Bi$’s power mode = $P_i$’s power mode;

} //for each process

Algorithm-2

for each process,

let $B_i$ be the bank in which the process resides;

at dispatching time,

$Bi$’s power mode = Active;

compute $T_i$ of $B_i$;

from all processes ($P_1 \ldots P_m$) in $B_i$, $/m$ is the number of processes:

find $P_i$ which has the highest priority, based on the ready-queue information;

let $p$ = position (order) of $P_i$ in the ready-queue;

$T_i = p \times Q$; $/Q$ is time quantum

select the best power mode:

on FSM, new-state = $\delta$(Active, $T_i$);

$Bi$’s power mode = new-state;

} //for each process

Although the proposed scheme is designed under the assumption of allocating a process into one bank, it can be modified and applied to the case of allocating a processor to multiple banks, for example random page allocation or contiguous allocation of relatively big processes. The basic difference is considering multiple banks, where a process is distributed, at dispatching time and pre-emption time instead of considering single bank. This will require slightly more complex control operations and storage in PCB for keeping the bank usage. With our proposed scheme, demand paging or bigger size bank can be used to adapt to the system in which relatively big programs are used.

5 Enhancement with early resynchronisation

In this section, we propose an enhancement scheme that reduces the power consumption gap between the ideal case and the latency case. The resulting DRAM power management scheme yields the better energy gain than that from the scheme described in Algorithm-1 and Algorithm-2. Fig. 6c illustrates the idea of the early resynchronisation for reducing the resynchronisation cost. As mentioned earlier, the resynchronisation cost is significant in both time ($T_i$) and energy ($T_i E_i$). It becomes bigger as we select the lower mode. The proposed scheme described in Section 4 avoids heavy resynchronisation cost by selecting the best power mode based on the FSM. However, there still...
remains resynchronisation overheads caused by the gap between the ideal case and the latency case. Thus the rationale of the enhancement is to make the scheme as close as reflecting the ideal case with acceptable scheduling time overhead. With the enhancement, we can achieve the additional energy gain up to $T_i E_1$ that can be derived by subtracting (1a) from (2a), or by subtracting (2b) from (1b) in Section 3.1.

As shown in Fig. 6c, the early resynchronisation yields the reduced overheads in both time and energy, compared to the latency case. In the figure, tail represents the remaining overhead portion whose length is $T_i - (D - R)$, where $D$ and $R$ represent dispatch time and early resynchronisation start time, respectively. In the early resynchronisation case, the total power consumption for one idle period (in a lower power mode) is

$$T_i E_1 + T_i E_a - (D - R)E_1$$

In fact, this is the generic formula for the power consumption, and formula (1a) for the ideal case and formula (2a) for the latency case in Section 3.1 are extreme cases of this in which $(D - R) = T_i$ and $(D - R) = 0$, respectively. The scheduler keeps a data structure named bank usage table for keeping the power modes of banks in the system. The table size is ignorable since each cell keeps only one power mode.

The proposed early resynchronisation scheme is implemented in two ways, namely coarse-grained and fine-grained. In the coarse-grained method, operations for the early resynchronisation are activated only at context switching time. Thus, the enhanced DRAM power management scheme with the coarse-grained early resynchronisation method is still a context switching time scheme whose scheduling time overhead is not significant. In the fine-grained method, operations for the early resynchronisation are activated at each clock cycle. This is an extreme case with the closest implementation towards the ideal case. Our experimental results show that the performance of the fine-grained method is very close to that of the ideal case. The performance difference comes from the occurrence of unexpected events. After an idle bank is activated early, there is a possibility of $T_i$ becoming longer, for instance an I/O done job is placed in the ready-queue based on its priority. Consequently, the resynchronisation is done too early and that brings about the performance difference between the ideal case and the fine-grained early resynchronisation case. However, the fine-grained method requires an excessive amount of scheduling time overhead since it activates the operation at each clock cycle. Based on the affordable scheduling time overhead, we can activate the early resynchronisation operation at every certain amount of cycles that lies between the fine-grained case and the coarse-grained case.

Algorithm 3 describes the early resynchronisation scheme for both coarse-grained and fine-grained methods. As described in the algorithm, all banks except currently active ones are checked for starting the resynchronisation as early as possible, either at context switching (dispatching) time (coarse-grained) or at every certain number of clock cycles (fine-grained). The performance of the early resynchronisation methods is described in Section 6.

Algorithm 3

At each dispatching time (coarse-grained) or each cycle (fine-grained)

\[\text{for } i = 1 \text{ to } n / n \text{ is number of banks in the system }\]

\[\text{if (mode-of-$B_i$} \neq \text{ active), }\]

\[\text{find } P_i \text{ which has the highest priority, based on the ready-queue information; }\]

\[\text{let } p = \text{ position of } P_i \text{ in the ready-queue; }\]

\[\text{compute remaining idle time } (T_i_{\text{remain}}):\]

\[T_i_{\text{remain}} = p \times Q; / Q \text{ is time quantum}\]

\[\text{if } (T_i_{\text{remain}} \leq T_i \text{ of mode-of-$B_i$})\]

\[\text{mode-of-$B_i$ = active; / starts resynchronisation}\]

\]

6 Experimental results

We developed a system simulator to test the proposed scheme. The simulator consists of three major components: a simulated CPU based on MIPS R2000 model, a simulated memory and a conventional multiprogramming/time-sharing operating system. Fig. 7 shows a conceptual view of the system simulator. The loader selects processes from the job pool (we assume that all jobs arrived at time 0 and are available); the low-power scheduler includes the power saving techniques described in Algorithms 1–3. The simulated CPU is pre-empted when the current time quantum expires or an I/O event occurs. In the simulated memory there are eight equal-sized banks, and we used small bank size, such as 1 KB or 2 KB, with simple test programs, for example arithmetic computation programs including loops and I/O. The relative sizes of the test programs to the bank size are 15.6–93.7%. Some assumptions used in our experiment are: (i) contiguous allocation of a process into one bank, for example sequential first-touch page allocation; (ii) I/O device is the hard disk; and (iii) extra scheduling cost is ignored when measuring the efficiency of the proposed scheme. For each process, the first available contiguous space in a bank is allocated for both code and data. In the case of manipulating relatively big programs, our scheme can be used with reasonably big

Fig. 6 Early resynchronisation

Fig. 7 Conceptual view of the system simulator
In the case of using bigger bank size, performance (in power saving) degradation is expected since a bigger portion of memory is active compared to the case of using smaller bank. Delaluz et al. [20] reported the effect of bank size to the power saving performance in which the total memory size is fixed. It was observed that decreasing the number of banks degraded the power performance in a proportional manner.

For reflecting the behaviour of the multiprogramming/time-sharing and memory allocation/deallocation, we used a number of test programs that are 128 MIPS-based machine codes. The average multiprogramming degree (checked at each context switching time) was 19–20 in our experiment. Since the time quantum size heavily affects the performance of the time-sharing system, we also tested with different quantum sizes that are 16–80% of the average job execution time. In the simulated I/O, only one I/O queue is used for holding I/O interrupted jobs for a constant amount of time. As used in many scheduling algorithms, a priority strategy (shortest remaining time first) is used for selecting the position in the ready-queue for the I/O done job.

For performance measurement, we tested the following cases to demonstrate the efficiency of the proposed scheme: (i) do-nothing: all eight banks are in active modes all the time; (ii) ideal case: no resynchronisation time overhead; (iii) latency case: maximum resynchronisation time overhead; (iv) early resynchronisation with coarse-grained method; and (v) early resynchronisation with fine-grained method. The base case for the comparison is 'do-nothing' in which all banks are always active, and performance is shown as normalised values to the base case.

In the simulated system, energy measurement is done at each clock; that is at each cycle, power modes of banks are checked and the consumed energy is computed. For measuring the resynchronisation cost with respect to both time and energy, we used formulas developed in Sections 3.1 and 5. Energy measurement for the ideal case is done as follows. In the ideal case, there exists no timing overhead (latency), but we should consider the resynchronisation energy cost. As shown in formula (1a), power consumption during the idle time in the ideal case is 
\[ (T_1 - T_2)E_1 + T_2E_2. \]

With the energy measurement done at each cycle, we only counted \( T_2E_2 \). Thus the amount \( T_1(E_1 - E_2) \) should be added when the current idle time is terminated that is at dispatching time. \( E_1 \) (lower mode unit energy) varies depending on the mode as shown in Fig. 1. The computations of the time and energy overheads in other cases are analogous, and we skip the description.

The following figures show the performance of the proposed scheme with varying time quantum sizes. The time quantum values are normalised to the average job length (execution time). Fig. 8 shows the performance of two prediction methods, that are average-waiting-time method ('average' in Fig. 8) and ready-queue-position method ('position' in Fig. 8). The values shown in the figure are energy gain percentages of using the versions of the proposed scheme over do-nothing (always active) case. In both ideal and latency cases, using the ready-queue-position method shows better performance on the energy gain with all the time quantum sizes tested.

Fig. 9 shows the efficiency of the proposed scheme. In the figure, power consumption values (%) are normalised to the values of do-nothing (always active) case (100%). Prediction method used is the ready-queue-position method since it showed better performance than the average-waiting-time method as illustrated in Fig. 8. As shown in Fig. 9, both of the early resynchronisation methods show better performances than the latency case with all the time quantum sizes tested. One exception is that the early-coarse method does not show extra energy gain over the latency case with relatively big quantum sizes (over 50%) since the context switching rarely occurs in such condition and hence, very rare chances of early resynchronisations. The early-fine method yields approximately the same (slightly worse) performance as the ideal case since it manages the reactivation of idle banks at each clock. As we anticipated, time quantum size also affects the performance. In general, with all versions of the proposed scheme, the shorter time quantum yields the better energy gain since there occurs more frequent context switching and thus more power mode changes.

Fig. 10 shows the time overhead caused by the resynchronisation. Overhead values are normalised to the latency case in which time overhead is always 100%. In Fig. 10, the ideal case and the early-fine case are not recognisable since there exist no resynchronisation latency (0%) in the ideal case and a negligible amount of latency in the early-fine case. In fact, the values in the early-fine case range from 0.0126 to 0.304% depending on the time quantum sizes tested. With relatively big quantum sizes (over 50%), the early-coarse case does not yield any gain over the latency case and its performance is shown in Fig. 9.

Fig. 11 shows the energy overhead caused by the resynchronisation. Overhead values shown are normalised to

---

**Fig. 8** Comparison of prediction methods

**Fig. 9** Efficiency of power-saving schemes: values are normalised to the values of do-nothing (always active, 100%)
between the energy gain of the ideal case and the latency case, two methods of the early resynchronisation scheme were developed and tested. With all the time quantum sizes tested, the fine-grained method showed much better performance than the coarse-grained method, but it might require a significant amount of scheduling time overhead since the method activates the operation at each clock. Of course, the coarse-grained method also showed the satisfying performance, as expected, with much less scheduling time overhead than the fine-grained case since it activates the operation at only context switching time.

The experimental results from the simulation demonstrated the efficiency of the proposed scheme in the multi-programming/time-sharing environment. The energy gain by using the proposed scheme ranges from 17.84 to 52.21% depending on the time quantum sizes tested in our experiment. Like other operating system-based DRAM power management schemes, the proposed scheme can be used with other approaches shown in the literature including hardware-supported approaches or dynamic voltage scaling for CPU for achieving further energy saving.

8 References