Low-energy Scheduling Algorithms for Wearable Fall Pre-impact Detection System

MN Nyan[a]; Francis EH Tay[a,b]; D Guo[a] ; L Xu[a]; KL Yap[a]
[a]Department of Mechanical Engineering, National University of Singapore, Singapore
[b]Medical Devices Group, Institute of Bioengineering and Nanotechnology, Singapore
LK Goh[c]; B Veeravalli[d];
[c]Communication system department, Institute for Infocomm Research, Singapore
[d]Department of Electrical Engineering, National University of Singapore, Singapore

ABSTRACT

In this paper, novel low-energy static and dynamic scheduling algorithms with low computational complexities for heterogeneous multiprocessor systems are proposed. Since battery life of the system plays a critical role in wearable embedded systems, the algorithms are useful for energy consumption reduction in Body Area Network (BAN)-based wearable multiprocessor systems in healthcare applications. Our developed BAN-based fall pre-impact detection system is used in this investigation. Based on simulation results using the algorithms, it is found that the battery life can be extended up to 41.6 percent more of its normal life without the algorithms.

Keywords: Energy-aware scheduling, embedded systems, heterogeneous multiprocessor system

1. INTRODUCTION

Falls are a major care and cost burden to health and social services world-wide. Falls have traditionally been recognized as one of the "giants" for geriatric medicine [1]. In this scenario, one of the key concerns in preventing or reducing the severity of injury in the elderly is to detect fall in its descending phase before the impact (pre-impact fall detection). A few groups have attempted to detect falls prior to impact [2-4]. Efficient feedback approaches such as inflatable hip protector are also investigated for fall injury minimization [5-6]. Our aim is to develop a wearable faint fall pre-impact detection system that can detect fall in its inception [4]. For the comfort of the user, the whole system is based on the Body Area Network (BAN).

The BAN comprises a central processing unit (CPU) and two wireless sensor sets (thigh sensor set (TS), and waist sensor set (WS)) located on thigh and waist (Fig.1) [4]. Each sensor set has its own processor for front-end data processing such as data sampling, filtration and wireless data transmission and the CPU has a more powerful processor to process the data received from sensor sets. The BAN is a heterogeneous multiprocessor system as processors have different characteristics such as processing capability and power consumption. Every unit is battery-operated with its own battery. Since the battery life of the system plays a critical role in battery-operated embedded systems, energy consumption minimization has become a major concern. Nowadays, modern embedded processors enable the dynamic voltage/frequency scaling (DVS) technique, which allows slowing down the processor speed to lower down the energy consumption. Therefore, specially designed energy-aware multiprocessor scheduling algorithms select different running speeds for different tasks (different parts of the algorithm for different purposes) of the pre-impact detection algorithm such that the total energy consumption on processors is minimized. Along this line of work, there has been no work specifically addressing the task scheduling problem for the heterogeneous BAN systems. Only few algorithms that address a model of application with multiple deadlines for heterogeneous systems [7], and slack reclamation algorithm for homogeneous systems [8] were developed. In [9,10], efficient static scheduling algorithms are proposed for reducing energy consumption in heterogeneous multiprocessor systems during the design-time phase. However, they are not suitable for dynamic slack reclamation during runtime. In this paper, novel critical-path based low-energy and low-complexity scheduling algorithms are presented for use in heterogeneous multiprocessor system during both design-time phase and runtime to reduce energy consumption.

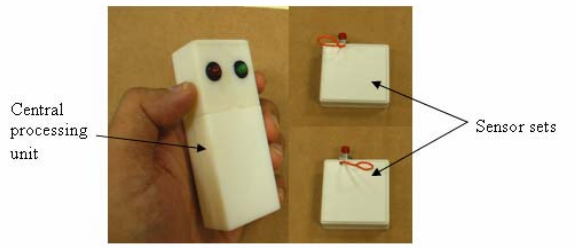

Fig.1. BAN-based wearable faint fall pre-impact detection system

2. METHODS

2.1 BAN-based fall pre-impact detection algorithm

In BAN-based wearable faint fall pre-impact detection system, each sensor set (TS and WS) contains one TI MSP430 processor, one MMA7260Q (±4g, 300mV/g) tri-axial micro-machined accelerometer and two ADXRS150 (±150°/sec) yaw rate gyroscopes measuring in lateral and sagittal directions. Data are sampled at 47samples/sec sampling rate (sampling interval: 21276.6us). Intel® PXA255 Processor (400MHz) is used in the CPU. Chipcon CC2420 Zigbee transceivers are applied for data communication between sensor sets and the CPU. The process flow of pre-impact detection algorithm is shown in Fig. 2 [4].

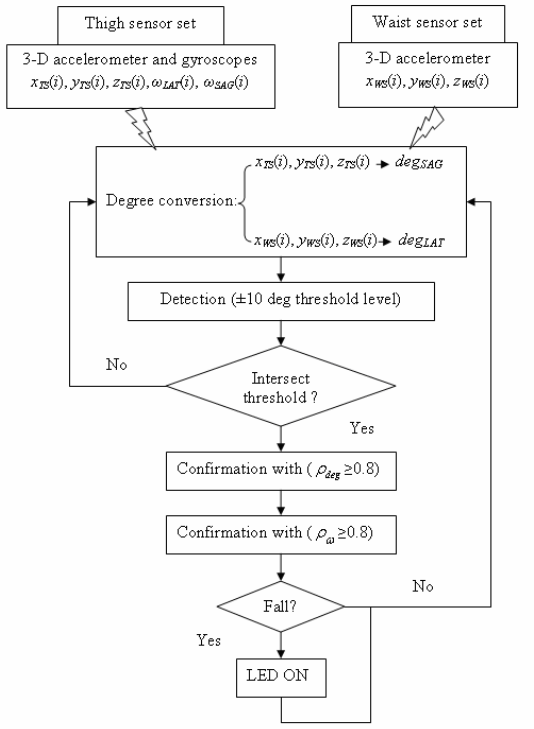

Fig. 2. Fall pre-impact detection algorithm

2.2 Critical-path based low-energy scheduling algorithms

Critical-path based low-energy scheduling algorithms are developed assuming that the system may have embedded processors with dynamic voltage/frequency scaling (DVS) features [11].

Scheduling algorithms are written in C++ programming. Steps implemented in scheduling algorithms are listed as follows.

(*i) Directed acyclic graph (DAG)*

The target algorithm (Fig. 2) run on the multiprocessor system can be considered as a set of tasks. Table 1 lists the abstracted tasks and their functionalities. Certain tasks also have precedence relations. For example, the filtering can be done only after sampling of the signals. There are also deadline constraints imposed on the tasks. As the target algorithm runs periodically, it needs to be completed within the sampling period. Including precedence relations and deadline constraints, the algorithm is specified as a directed acyclic graph (DAG) G_s = (T,E), called task graph, which consists of a set of dependent tasks ($\tau \in$ T) connected by edges ($\varepsilon \in$ E) (Fig.3).

Table 1. Tasks and their functionalities

Tasks	Functionality	Explanation
$\tau_1^{Ta}, \tau_1^{Wa}, \tau_1^{Tg}, \tau_1^{Wg}$	Analog to digital conversion	T: thigh, W: waist, a:accelerometer, g: gyroscope
$\tau_2^{Ta}, \tau_2^{Wa}, \tau_2^{Tg}, \tau_2^{Wg}$	Filter	
τ_3^{Ta}, τ_3^{Wa}	Accelerometer calibration	
τ_4^{Ta}, τ_4^{Wa}	Degree conversion	
τ_5	Correlate acceleration data	
τ_6	Correlate gyroscope data	
τ_7	Fall pre-impact detection	

A node $\tau \in$ T in the task graph denotes a task. The precedence constraints between tasks are represented by the edges $\varepsilon \in$ E. If there exists an edge $\varepsilon :: \tau_i \rightarrow \tau_j$, it means that τ_j can only be executed by after τ_i completes its execution. Each node τ is associated with a tuple (w, a) for each processor, where *w* and *a* denote the worst-case execution time (WCET) and average-case execution time (ACET) for the corresponding

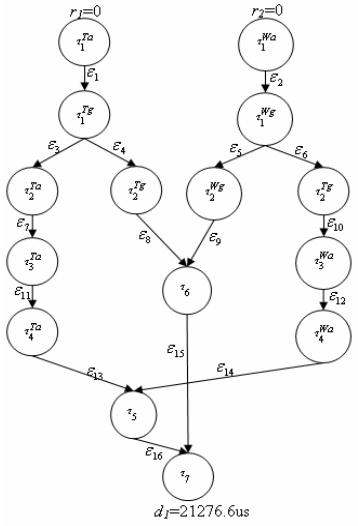

Fig. 3. Task graph G_s for fall pre-impact detection

processor respectively. WCET is the maximum possible execution time of a task while ACET is the average execution time of a task over a period of time. We can obtain the WCET by determining the longest execution time in a certain number of runs, e.g., 100 runs. Similarly, the ACET can be obtained by averaging the execution times of these runs. Table 2 shows the WCET and ACET of all tasks in G_s. Both the WCET and ACET for a task are measured at the maximum speed of the processor. Each edge ε is associated with a weight l (worst-case communication time, WCCT) representing a communication link between two dependent tasks when they are scheduled on two different processors.

Table 2. Tasks with their WCET and ACET

Tasks	WCET (us)	ACET (us)
$\tau_1^{Ta}, \tau_1^{Wa}, \tau_1^{Tg}, \tau_1^{Wg}$	7.88	7.88
$\tau_2^{Ta}, \tau_2^{Wa}, \tau_2^{Tg}, \tau_2^{Wg}$	123	111
τ_3^{Ta}, τ_3^{Wa}	141	141
τ_4^{Ta}, τ_4^{Wa}	1493	1285
τ_5	3839	1417
τ_6	3667	1509
τ_7	1861	748

There is no communication cost if tasks are executed on the same processor. Each root node is associated with an arrival time r (for which it can begin its execution) and each sink node is associated with a deadline d (the time by which it must complete its execution). But in this application, it is considered that all tasks share a common deadline (d_1). In our application, d_1 is the period of the task graph which is 21276.6 us.

(ii) Initial tightest schedule and development of a new DAG graph

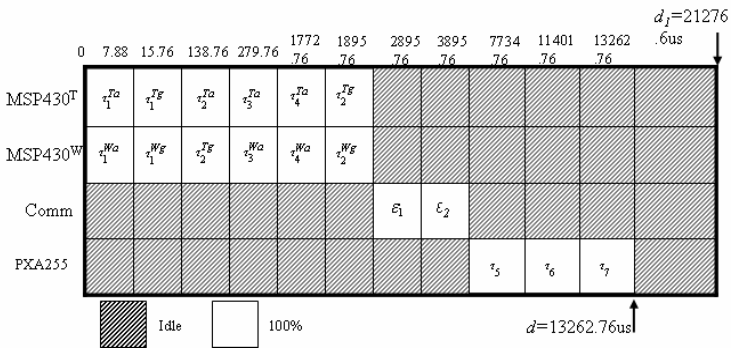

Fig. 4. Initial tightest schedule for the task graph in Fig. 3

In the original DAG graph G_s, tasks are not allocated on the processors (MSP430^{T,W} and PXA255). After allocating the tasks to the processors, an initial tightest schedule with minimum schedule length is obtained by readjusting and allocating the execution order of tasks on each processor and communication events for wireless channels. In this initial tightest schedule, tasks are scheduled as tight as possible so that d, which is the end time of the last task τ_7, is minimized. In our application, the data from τ_2^{Ta} to τ_5 and from τ_2^{Tg} to τ_6 are sent together. Similarly, the data from τ_2^{Wa} to τ_5 and from τ_2^{Wg} to τ_6 are sent together. The resulting initial tightest schedule is shown in Fig. 4. In this paper, the initial tightest schedule is generated using a list scheduling method. Based on the originally specified task graph G_s and the initial tightest schedule, a new DAG graph G = (V,E) is constructed. The set of vertices V contains all the tasks as in the original task graph G_s and the communication events allocated to the communication links in the tightest schedule. If an edge $\varepsilon ::v_i \rightarrow v_j \in E$ exists between v_i and v_j, it means that v_i is a direct predecessor of v_j in the original task graph, or v_i is scheduled just ahead of v_j on the same processor. Arrival time associated with root nodes and deadlines associated with sink nodes in G_s are still kept with the corresponding nodes in the newly constructed DAG (Fig. 5).

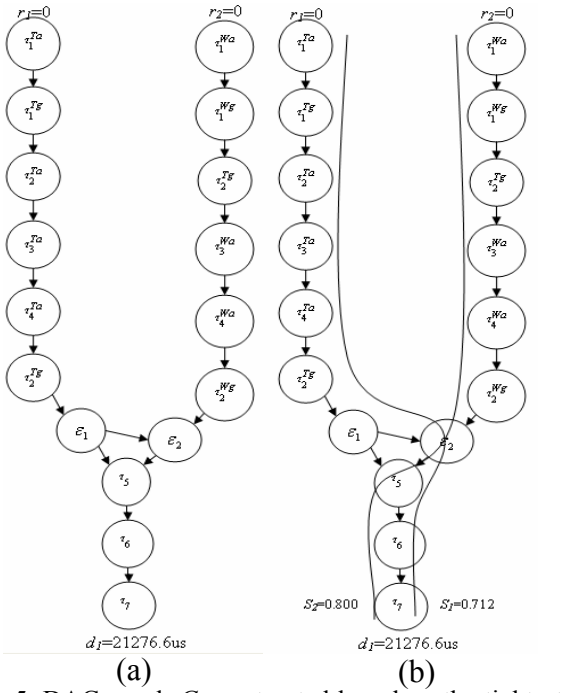

Fig. 5. DAG graph G constructed based on the tightest schedule (a) and identified critical paths (b)

(iii) Critical-path identification

In DAG graph, a path always starts from a root node with arrival time r_{root} ($r_{1,2}$=0) and ends at a sink node with deadline d_{sink}.(d_1=21276.6us) Scaling factor for a path k is defined as

$$S_k = \frac{(d_{\sin k} - r_{root}) - (\sum_{i \in path\ \ k} w_i + \sum_{i \in path\ \ k} l_i)}{\sum_{i \in path\ \ k} w_i} \quad (4)$$

where w, l represent WCET, WCCT and total available slack (the amount of unused or available time in a path) on the path k is given by $(d_{\sin k} - r_{root}) - (\sum_{i \in path\ \ k} w_i + \sum_{i \in path\ \ k} l_i)$.

Therefore, the scaling factor S_k then represents the ratio of allocated slack for a task over its WCET, i.e., distributing the total slack on path k evenly among all tasks along the path. Scaling factors, using Eqn. 4, i.e.,

$$S_1 = \frac{(d_{\sin k} - r_{root}) - (\Sigma_{i \in path\ \ 1} w_i + \Sigma_{i \in path\ \ 1} l_i)}{\Sigma_{i \in path\ \ 1} w_i}$$

$$= \frac{21276.6 - 13262.76}{11262.76} = 0.712,$$

$$S_2 = \frac{(d_{\sin k} - r_{root}) - (\Sigma_{i \in path\ \ 2} w_i + \Sigma_{i \in path\ \ 2} l_i)}{\Sigma_{i \in path\ \ 2} w_i}$$

$$= \frac{21276.6 - 12262.76}{11262.76} = 0.800,$$

(based on the WCETs shown in Table 2) on each path are shown in Fig. 5b. A critical path for a pair of nodes (v_i,v_l) is defined as the one that has minimum scaling factor among all paths that pass through v_i and ends at v_l. For example, if two paths ($\tau_1 \rightarrow \varepsilon_1 \rightarrow \varepsilon_2 \rightarrow \tau_5$ with S_0=0.4 and $\tau_1 \rightarrow \tau_5$ with S_1=0.8) are available between two tasks τ_1 and τ_5, the path with minimum scaling factor is taken (Fig. 6). In our case, the set of critical paths P is shown in Fig. 5b.

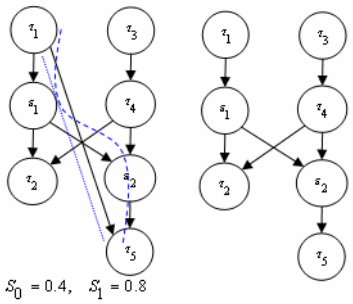

Fig. 6. Another task graph example is used to clarify the critical path

(iv) Static scheduling algorithm

The critical-path based static scheduling (CPSS) algorithm (Algorithm 1) uses the path information track-and-update scheme to distribute slack over tasks, based on the identified set of critical paths. By evenly distributing slack over tasks, the slack time that can be allocated to a task τ_i along the path k is then computed by $S_k \cdot w_i$ and the speed reduction ratio (compared to the maximum speed of the processor) for this task can be set to S_{τ_i}=1.0+ S_k. In each iteration, the most critical path m with minimum scaling factor is identified. The speed reduction ratios for all unallocated tasks along the path m are computed and allocated to every task on the path. If a task τ_k on path m is located on multiple paths, it does not use up the slack available from the rest of unallocated paths. The remaining slack time not used by task τ_k for such a path j is $(S_j - S_m) \cdot w_k$. The scaling factor for each such path j should be updated according to equation

Algorithm 1. Critical-path based static scheduling algorithm

Compute the scaling factor S_k for each path $k \in P$ using Eqn. 4.
while P is not empty **do**
Identify the path m with minimum scaling factor S_m;
for each unallocated task τ_k on path m **do**
Task τ_k is allocated slack time $S_m.w_k$ and
The speed reduction ratio S_{τ_k} is set to be 1.0+S_m;
for each path $j \in P$ that τ_k is also located on **do**
Update scaling factor of path j using Eqn. 5
Remove path j from P if S_j is equal to zero;
end for
end for
Remove path m from P_i
end while

$$S'_j = S_j + \frac{(S_j - S_m) \cdot w_k}{\Sigma_{i \in path\ \ j\ \ \&\ \ i\ \ not\ \ allocated} w_i} \quad (5)$$

such that this remaining slack can be reclaimed by other unallocated tasks along path j. The most critical path m will be exempted from the set P afterwards. The same process of identifying most critical path and updating scaling factors of associated paths continues until all the tasks are allocated. In this way, speed reduction ratio is figured out for every task on each processor and the least amount of energy is consumed. In CPSS scheduling, it is assumed that all tasks consume their WCETs.

(v) Dynamic scheduling algorithm

During runtime, the tasks do not always consume their WCET. Unused time can be used to reduce the energy consumption further. The critical-path based dynamic scheduling algorithm (CPDS) is therefore used to reclaim the unused time during runtime. To reduce the runtime complexity, we use a two-phase framework. During the design-time phase, i.e., CPSS, the initial tightest schedule is obtained and then the set

of critical paths is identified. During the runtime phase, we apply dynamic CPDS to determine the running clock speed for each scheduled task using the critical-path information track-and-update scheme. The CPDS algorithm uses the static schedule results obtained from the CPSS algorithm assuming every task takes its ACET. When a task is due to be scheduled during runtime, it

Algorithm 2. Critical-path based dynamic scheduling algorithm

function before schedule τ_k;

Identify the path m with minimum total slack among associated paths with τ_k, i.e., $L_m = \min_{\tau_k \in path\ \ i} L_i$;

if $(S_{avg-\tau_k} - 1.0) * w_k \le L_m$ **then**

$S_{\tau_k} = S_{avg-\tau_k}$;

else

$S_{\tau_k} = 1.0 + L_m / w_k$;

end if

for each path j that τ_k is located on **do**

Update the total slack L_j using Eqn. 6

end for

end function

function after schedule τ_k

for each path j that τ_k is located on **do**

Update the total slack of path j using Eqn. 7;

end for

end function

aggressively runs at the statically computed average-case speed, under the condition that the deadlines are met even when all the subsequent tasks consume their WCETs. The path information track-and-update scheme keeps track of the minimum total available slack from the task's associated critical paths and guarantees that the task does not use more than that, which therefore satisfies the various constraints at all times. The pseudo code for the CPDS algorithm is given in Algorithm 2. Initially each path j is assigned with the total available slack $L_j = (d_{sink} - r_{root}) - (\sum_{i \in path\ \ j} w_i + \sum_{i \in path\ \ j} l_i)$. Before and after a task completes its execution, the total slack of its associated paths are updated and tracked. When task τ_k is due to be scheduled, the path m with minimum total slack L_m is identified from all paths associated with τ_k. Task τ_k is constrained to utilize a maximum slack of L_m. From this, the maximum allowable speed for the task τ_k is computed, in order to guarantee that the execution for the subsequent tasks can be completed before their deadlines even if they take their WCETs. Suppose that the statically computed average-case speed reduction ratio for task τ_k is denoted as $S_{avg-\tau_k}$. If the statically allocated average-case slack time $(S_{avg-\tau_k} - 1.0) \cdot w_k$ is less than or equal to L_m, τ_k can take the speed reduction ratio of $S_{avg-\tau_k}$ and is allocated with slack time $L_{\tau_k} = (S_{avg-\tau_k} - 1.0) \cdot w_k$, or else it can only aggressively use up the slack time $L_{\tau_k} = L_m$ and its speed reduction ratio is set to $S_{\tau_k} = 1.0 + L_m / w_k$. After the execution speed for τ_k is allocated, the total available slack on each path j that τ_k is located on will be updated by

$$L'_j = L_j - L_{\tau_k}. \quad (6)$$

After τ_k completes its execution, a slack of $S_{\tau_k} \cdot (w_k - a_k)$ is generated and it shall be distributed over its associated paths. The total available slack on these paths are updated by

$$L'_j = L_j + S_{\tau_k} \cdot (w_k - a_k). \quad (7)$$

The information on the total available slack on paths is carried from one node to another with the communication events between these two nodes. It can be guaranteed that each task to be scheduled always gets the latest path information, due to the precedence relationship between tasks.

3. EXPERIMENTAL RESULTS

We conducted a simulation study to evaluate the performance of applying DVS to the fall pre-impact detection system using the CPSS and CPDS algorithms. We measure the worst-case (WCET) and average case execution times (ACET) of all the tasks as shown in Table 2. We use the values that are reported in [12] for the power consumption of the MSP430 microcontroller at different voltages and frequencies. For the PXA255 processor, we obtained the values for the combined power consumption of the processor and the gumstix motherboard from [13]. First we present the CPSS algorithm. As shown in Fig. 5b, the scaling factors for all critical paths are

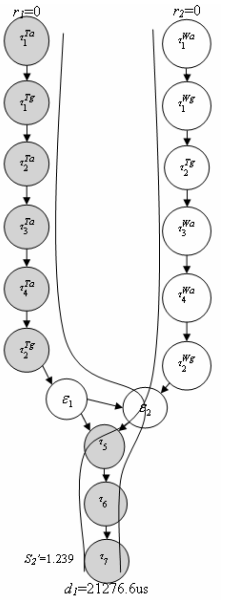

Fig. 7. Update of paths' scaling factors during the execution of the CPSS algorithm

computed and we identify that path 1 has the minimum scaling factor. In Fig. 5b**,** each task τ_k located on path 1 is allocated a slack of $0.712 \cdot w_k$. The scaling factor of path 2 is then updated to 1.239 using Eqn. 5 since the tasks τ_5, τ_6 and τ_7 also lie on path 2 (Fig. 7). Each task τ_k on path 2 that is yet to be allocated is allocated a slack of each $1.239 \cdot w_k$. (Fig. 7). Based on the slack that are allocated to the tasks, the computational frequencies and voltage levels at which to execute the tasks are known. However, since the voltage levels and frequencies are continuous and real processors support only discrete voltages and frequencies; we apply the method in [14] to map the continuous voltage levels and frequencies to the two neighboring discrete levels. Assuming that all tasks always execute at their WCET, the total energy consumption of the system (three processors) in one iteration of the pre-impact detection algorithm is 12770 uJ if no DVS is applied and 10843 uJ if DVS is applied using CPSS.

For the dynamic case, we generate the execution times of each task in the DAG for 500 cycles assuming that the execution times are normally distributed around the ACET. In CPDS, we first obtain a static schedule using CPSS assuming that the tasks execute at their ACET. We then apply the CPDS algorithm as described in Section 2.2(v) to the system. The results are as follows. For the case when no DVS is used, the total energy consumption for three processors ($MSP430^T$, $MSP430^W$, and PXA255) is 4.35 J (0.37+0.37+4.28 J). On the other hand, when CPDS is used, the total energy consumption is 3.08 J (0.032+0.032+3.02 J). Table 3 shows the battery life extension of 1000mAh battery using CPDS algorithm for gumstix motherboard with the PXA255 processor. From the results we see that CPDS is able to reduce the energy consumption of the system significantly by using DVS.

Table 3. Battery life extension of 1000mAh battery using CPDS algorithm on the gumstix motherboard with the PXA255 processor

	Voltage (V)	Average Current (mA)	Battery life (hours)
Gumstix motherboard with PXA255	4.5	$\frac{4.28J}{4.5V \times 500 \times 0.0212766s} = 89.4mA$	11.19
DVS with CPDS	4.5	$\frac{3.02J}{4.5V \times 500 \times 0.0212766s} = 63.1mA$	15.85
Note: 0.0212766s is one iteration execution time of the detection algorithm using DVS processor with the CPDS algorithm			

4. CONCLUSIONS

In this paper, we propose low-energy static and dynamic scheduling algorithms. Our algorithms are based on the analysis of critical paths that represent all the timing and precedence constraints imposed. We designed a track-and-update scheme to keep track of and to update the scaling factor (or available slack) for each critical path. By using this new scheme, our static (CPSS) and dynamic (CPDS) algorithms exhibited a low computational complexity compared to other scheduling algorithms [8]. The timing and precedence constraints are also guaranteed to be satisfied. The simulation results conclusively demonstrated that our proposed scheduling algorithms save considerably energy compared to the real execution on the heterogeneous multiprocessor systems.

Acknowledgement: The authors would like to acknowledge the support for the projects "MEMSWear II: Mission critical wearable embedded systems for elderly care" (grant-R265000229305) and "Design of Adaptive and Hybrid Energy-&-QoS Aware Heterogeneous Multiprocessor Scheduling Strategies for Embedded Systems" (grant-R263000375305) by A*STAR (Agency for Science, Technology and Research) SERC, Singapore, under EHS-II Programme.

REFERENCES

[1]Marks, R., Allegrante, J.P., Ronald MacKenzie, C., Lane, J.M., Hip fractures among the elderly: causes, consequences and control. Ageing Research Reviews 2(1), 2003, 57-93.

[2]Bourke, A.K., Lyons, G.M., A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. Medical Engineering & Physics, In Press, Corrected Proof, Available online 11 January 2007.

[3]Wu, G., Distinguishing fall activities from normal activities by velocity characteristics. Journal of Biomechanics 33(11), 2000, 1497-1500.
[4]Nyan, M.N., Tay, E.H.F., Murugasu, E.. Faint Fall Onset Detection System. United State Patent Office Document (Provisional Patent), No. 60/885,956, 2007.
[5]Davidson, M.E., System for protection from falls. United States Patent Office Document, US2004/0003455, 2004.
[6]Ulert, I.A., Hip Protector. United States Patent Office Document, US2002/0078484, 2002.
[7]Luo, J and Jha, N. K., Static and dynamic variable voltage scheduling algorithms for real-time heterogeneous distributed embedded systems. Proceedings of the 2002 conference on Asia South Pacific design automation/VLSI Design, 2002, 719.
[8]Hua, S., Qu, Q., Power minimization techniques on distributed real-time systems by global and local slack management. In IEEE/ACM Asia South Pacific Design Automation Conference (ASP-DAC), 2005.
[9]Lee Kee Goh, Bharadwaj Veeravalli, and Sivakumar Viswanathan, An Energy-aware Gradient-based Scheduling Heuristic for Heterogeneous Multiprocessor Embedded Systems," Proceedings of International Conference on High Performance Computing (HiPC), Lecture Notes in Computer Science 4873, 2007, pp. 331-341.
[10]Lee Kee Goh, Bharadwaj Veeravalli, and Sivakumar Viswanathan, Design of Fast and Efficient Energy-aware Gradient-based Scheduling Algorithms for Heterogeneous Embedded Multiprocessor Systems," To appear in IEEE Transactions on Parallel and Distributed Systems, 2008.
[11]Yanhong Liu, Bharadwaj Veeravalli, and Sivakumar Viswanathan, "Novel Critical-Path based Low-Energy Scheduling Algorithms for Heterogeneous Multiprocessor Real-Time Embedded Systems," Proceedings of International Conference on Parallel and Distributed Systems (ICPADS), December 2007.
[12]Cho, Y., Kim, Y., Chang, N., PVS: Passive Voltage Scaling for Wireless Sensor Networks. Proceedings of 2007 International Symposium on Low Power Electronics and Design, 2007, 135-140.
[13]http://docwiki.gumstix.org/Power_spec/.
[14]Ishihara, T., Yasuura, H., Voltage Scheduling Problem for Dynamically Variable Voltage Processors. Proceedings of International Symposium on Low Power Electrionics and Design. 1998.