2013 International Conference on Parallel and Distributed Systems

Cost Minimization for Scheduling Parallel, Single-threaded, Heterogeneous, Speed-scalable Processors

Rashid Khogali & Olivia Das
Department of Electrical & Computer Engineering
Ryerson University
Toronto, Canada
{rkhogali, odas }@ee.ryerson.ca

Abstract— **We introduce an online scheduling algorithm to optimally assign a set of arriving heterogeneous tasks to heterogeneous speed-scalable processors. The goal of our algorithm is to minimize the total cost of response time and energy consumption (TCRTEC) of the tasks. We have three contributions that constitute the algorithm. First, we propose a novel task dispatching strategy for assigning the tasks to the processors. Second, we propose a novel preemptive service discipline called Smallest remaining Computation Volume Per unit Price of response Time (SCVPPT) to schedule the tasks on the assigned processor. Third, we propose a dynamic speed-scaling function that explicitly determines the optimum processing rate of each task. In our work, the processors are heterogeneous in that they may differ in their hardware specifications with respect to maximum processing rate and power functions. Tasks are heterogeneous in terms of computation volume and processing requirements. We also consider that the unit price of response time for each task is heterogeneous. Each task's unit price of response time is allowed to differ because the user may be willing to pay higher/lower unit prices for certain tasks; thereby increasing/decreasing their optimum processing rates. In our SCVPPT discipline, a task's scheduling priority is influenced by its remaining computation volume as well as its unit price of response time. Our simulation results show that SCVPPT outperforms the two known service disciplines, Shortest Remaining Processing Time (SRPT) and the First Come First Serve (FCFS), in terms of minimizing the TCRTEC performance metric. The results also show that the algorithm's dispatcher outperforms the well known Round Robin dispatcher when the processors are heterogeneous. We focus on multi-buffer, single-threading where a set of tasks is allocated to a given processor, but only one task is processed at a time until completion unless preemption is dictated by the service discipline.**

Keywords- speed scaling, parallel computing, single-threaded, heterogeneous processors, multi-processor scheduling, energy and response time cost, mobile.

I. INTRODUCTION

Energy consumption is a major constraint in today's computing devices. This is especially true for portable devices, for example laptops and mobile phones which rely on batteries for energy. It is well known that the energy consumption grows proportionally to s^{α} where s is the processor speed and α is a constant > 1 (e.g. [4, 5, 11, 15, 26]). This implies that higher speed leads to faster execution but incurs high energy consumption. One way to reduce energy consumption is to employ dynamic speed-scaling (see, e.g. [8, 27]) where the speed of the processors can be changed dynamically depending on the workload. The aim is to reduce processor speed at times of low workload.

Contemporary portable computing devices such as the recent versions of mobile phones, Tablets, iPads and gaming consoles (for example, the PSPVita [25]) utilize multiple processors. Multiple parallel processors are mostly used to improve overall processing performance needed for multi-media applications. In the domain of scheduling, considerable attention has been given to single processor architecture [3, 7, 8, 9, 22, 26]. Fewer have considered multiprocessors [4, 5, 12, 13, 19].

Although current architectures mostly consist of homogenous collection of processors, several works suggest that future chip architectures would consist of heterogeneous processors [10, 24]. Gupta et al. [16] further suggest that scheduling heterogeneous processors is a more challenging task compared to homogeneous processors.

This paper investigates the problem of online scheduling of the arriving heterogeneous tasks to multiple heterogeneous speed-scalable processors. The tasks are heterogeneous in terms of computation volume and processing requirements. The processors are heterogeneous in terms of their hardware specifications with respect to maximum processing rate and power functions.

The goal of any speed-scaled multiprocessor scheduling algorithm is: (i) to minimize the response time given energy as a budget, (e.g. [23]) or (ii) to minimize the energy consumption as long as the task deadlines are not violated, (e.g. [22, 26]) or (iii) to optimize a tradeoff between energy consumption and response time (e.g. [3, 9, 18]). The objective of our work is to minimize the total cost (in terms of dollars) of energy and response time (**TCRTEC**). In our work, the user determines unit cost of response time of a task's execution. Unlike any speed scaling algorithm, we explicitly factor the input of a user with respect to determining the unit cost of response time for executing each task. This allows the user to influence the degree of a task's execution in the economy-performance continuum. The user or OS can set the unit price of energy for all tasks depending on the actual unit price of energy in a given geographical region and time of day.

This paper introduces a new multiprocessor speed-scaled scheduling algorithm named "*Single-threading Multi-Buffer Scheduling & Parallel Processing Algorithm* (**SMBSPP**)." The goal of this algorithm is to minimize the performance metric, TCRTEC. It makes three key contributions:

- A novel task dispatcher which assigns a task to a given processor based on the *Minimum among Minimized Costs of Virtually Introducing the Task to each Processing Stream* (**MMCVITPS).** It dictates which of the processors should process each arriving task/s based on classifying a set of minimized potential aggregate cost functions each associated with a processing stream.
- A dynamic speed-scaling function, which we name, "*Optimum Single-threading Speed Scaling Function*" (**OSTSCF**) that determines the optimum processing rate of a given processor as a function of the following:
    - The parameters of the processor's power function.
    - The unit price of energy.
    - The sum of the unit prices of response time of all tasks residing in the processor's buffer.
- A novel preemptive service discipline called Smallest remaining Computation Volume Per unit Price of response Time (**SCVPPT**) to schedule the tasks on the assigned processor. This discipline minimizes the **TCRTEC** performance metric and also conveniently allows the user to dynamically upgrade or degrade the priority of tasks.

The first two contributions are achieved through solving a set of multidimensional convex optimization problems.

In this work, we focus on multi-buffer, single-threading where a set of tasks is allocated to a given processor, but only one task is processed at a time until completion unless preemption is dictated by the service discipline. In order to find the optimal speed of a processor, the maximum allowable rate of the processor and the minimum recommended rate of execution for a task are considered as constraints.

Our simulation results show that our MMCVITPS dispatcher works well with heterogeneous processors and outperforms the classic Round Robin dispatcher. The results also show that our SCVPPT scheduling discipline matches or outperforms the two known service disciplines, Shortest Remaining Processing Time (SRPT) and the First Come First Serve (FCFS), in terms of minimizing the TCRTEC performance metric. SRPT policy always selects for service the task that has the least remaining service time and it is a preemptive policy. FCFS, on the other hand, is a non-preemptive policy that selects the tasks for service in order of their arrivals.

This paper is organized as follows. Section II presents a brief description of prior related work. Section III builds background theory and assumptions. Section IV utilizes Section III to formally state the problem and synthesize the algorithm. Section V describes the SMBSPP algorithm. Section VI provides simulation results that evaluate the overall performance of the algorithm using a variety of performance metrics. Also in this section, we demonstrate the performance of the algorithm's dispatcher in comparison to the Round Robin dispatcher under various traffic conditions and service disciplines.

II. RELATED WORK

In this section, we provide a concise summary of prior related work that is most relevant to this paper.

In the past, when energy was not a major concern, the objective of scheduling algorithms was to minimize the total response time (also called flow time) of all tasks where processors were running at fixed speeds (e.g [20, 1]). The response time is the time elapsed since a task arrives until it is completed.

The study of energy-efficient speed-scaled scheduling was initiated by Yao et al. in [26]. They considered deadline-based scheduling for a single processor where the jobs need to complete by their given deadlines. The goal was to minimize energy consumption. Assuming the processor's power consumption ($P(s)$) is a convex function of processor speed (s), where $P(s) = s^{\alpha}$ for $\alpha > 1$, they considered scheduling a sequence of tasks on a single variable speed processor. Each task has a required deadline, release time and processing volume (analogous to the number of CPU cycles required to execute a task). They allow pre-emption, where a task is allowed to resume on the same processor after being interrupted. They proposed an optimal offline algorithm (YDS) to solve the task scheduling problem in polynomial time. In the same work, they further introduced two online algorithms, namely, Optimum Available (OA) and Average Rate (AR). They proved that AR has an energy competitive ratio of $(2\alpha)^{\alpha} / 2$. Bansal, Kimbrel and Pruhs [7] worked on OA and proved it to have an energy competitive ratio of exactly α^{α} . To solve for multiprocessor case, Angel et al. [5] considered the problem of scheduling a set of tasks with deadlines, release dates and processing requirements, on parallel (speed scalable) processors so as to minimize the total energy consumption. They considered migration where a task is allowed to resume its execution on a different processor. They also allowed pre-emption. They name their optimal scheduling algorithm BAL which has a time complexity of $O(nf(n)\log U)$ where, n is the number of jobs, $f(|V|)$ is the computational complexity of solving a maximum flow in a layered graph with $O|V|$ vertices and U is the range of all processor speed values divided by the targeted accuracy. Independently, Albers et al. [2] considered the same multiprocessor speed scaling problem with migration, and obtained an optimal scheduling algorithm that is fully combinatorial and has a time complexity of $O(n^2 f(n))$. Angel et al. [5] compared their

BAL algorithm to the one of Albers et al. [2] and stated that when the target precision is sufficiently high, the algorithm of Albers et al. [2] is superior to BAL, otherwise if the target accuracy is relaxed, BAL's algorithm is indeed superior.

Among energy efficient scheduling algorithms, several studies have considered minimizing the total response time, given a set energy budget (e.g. [23]). In particular, Pruhs et al. [23] considered offline scheduling to minimize the total response time on a single processor, for a given amount of energy. They gave a polynomial time optimal algorithm for the special case when jobs are of unit size.

To better understand the tradeoff between response time and energy, Albers et al. [3] proposed minimizing the sum of total response time and energy for a single processor. They presented an online algorithm that is $8.3e\left(\frac{3+\sqrt{5}}{2}\right)^{\alpha}$ competitive for jobs of unit size. This result was improved by Bansal et al. [9] who showed that this algorithm is 4-competitive. Bansal et al. [9] also gave the first constant competitive algorithm for arbitrary size jobs. The multiprocessor case was first discussed by Bunde [12] that presented an offline approximation algorithm for unit size jobs. However, Lam et al. [19] presented the first constant competitive online algorithm for arbitrary job sizes. In [19], jobs are clustered and then round robin dispatched to the processors independently for each cluster. Then they apply the BPS online algorithm given by Bansal et al. [9] to each processor.

In this paper, we present an online scheduling algorithm that minimizes the cost of response time plus energy for the heterogeneous multiprocessor case.

III. PRELIMINARY DEFINITIONS & ASSUMPTIONS

A. A Task

A task comprises of a set of base instructions, usually with a minimum processing requirement that is enforced in advance by the programmer during software architectural planning. Mathematically, we model a task, $T_k \in T$ as a vector with the following two parameters.

$$T_k = (B_k, p_{\min,k})$$

- B_k is the task's remaining computational volume in base instructions (*n*).
- $p_{\min,k}$ is the task's minimum recommended processing rate in base instructions per second (*n.Hz.*) .

B_k is measured in *base* instructions so as to consistently measure a task's instructions or computation volume. For example, *multiplication* and *addition* operations are not treated as commensurable instructions, but is each translated to some number of base operations or floating point operations. For this example, the number of base instructions for a *multiplication* operation generally exceeds that of an *addition* operation. The unit of a base instruction is *n*. Note that a base instruction can take any arbitrary number of fixed clock cycle/s. We assume a base instruction requires 10 Kilo clock cycles in our experiments.

$p_{\min,k}$ is a software constraint imposed by the software designer. It is fixed and optional, but crucial in identifying the minimum processing rate of executing the task by a given processor. An example is when a game is made up of a task/s, the game refresh rate is heavily influenced by $p_{\min,k}$ and if it is not satisfied, the game may be unplayable. We also enforce $p_{\min,k} > 0$ because we want to eliminate the trivial zero-processing rate condition.

B. A User Profile

A User Profile comprises of a set of unit cost sensitivity factors or unit prices that are specified by the user through a profile setting integrated in the operating software of the computing device. This profile setting could be an energy saving profile, a performance intensive profile or any other custom profile that is specified by the user. If the user chooses not to specify a custom profile setting, a default setting can be implemented by the programmer that is a balanced tradeoff between an energy saving profile and a performance intensive profile.

Mathematically, we model a user profile vector $U_k \in U$ associated with a task $T_k \in T$ as $U_k = (u_\varepsilon, u_{t,k})$, where:

u_ε - Unit price of energy measured in *\$/Joule*, where $0 < u_\varepsilon < \infty$.

$u_{t,k}$ - Unit price of response time measured in *\$/Second*, where $0 < u_{t,k} < \infty$.

One practical way to calibrate these cost sensitivity factors is to use the actual unit prices of energy and time in a given geographical region and time of day. For instance, in Ontario, Canada the regulated price of energy during peak hours is 12.4 *¢ / kWh* [17] and the minimum wage of employment as of May 2013 in Ontario Canada is CD\$10.25/hour [21]. This translates to $u_\varepsilon = 3.\overline{4}x10^{-8}$ *\$/Joule* and $u_{t,k} = 2.847\overline{2}x10^{-3}$ *\$/Second.* This is merely a suggestion as we are not enforcing the notion that the unit price of time for a specific individual should always be dictated by his/her hourly pay. Ideally a given user should set $u_{t,k}$ to any price he/she can afford or believes is the price of a second of his/her life.

Note that the unit price of energy for all tasks need not be different (this explains the missing *k* subscript in comparison to $u_{t,k}$) and can be set by the OS, but the unit price of response time for each task may be different because we allow the user to influence the priority of a task's execution through the following ways:

- If a user is willing to pay more for a task's response time, the algorithm's speed scaling function (**OSTSCF**) will increase hence executing the task at a faster rate at the expense of energy and vice versa.
- Under our proposed service discipline, **SCVPPT** (which is a generalized version of SRPT) will prioritize the task accordingly to Smallest remaining computation volume per unit price of response time $\left(B_k / u_{t,k}\right)$. Therefore a user can maintain or even improve the priority of a large task by accepting higher unit price of response time or even degrade the priority of a small non-urgent task by setting a sufficiently small unit price of response time.

C. *A Processing Stream*

A processing stream consists of a (core) processor ($\vec{P}_{s,j}$) and a corresponding memory Queue ($\vec{Q}_{s,j}$). A processing stream is distinguished among other parallel processing streams by the j^{th} index, where $1 \leq j \leq m$. The vector notation in $\vec{P}_{s,j}$ and $\vec{Q}_{s,j}$ is purely symbolic to denote hardware. Likewise, the 's' subscript denotes 'stream' and is not an index.

1) *Stream Processor:* Each processing stream's processor ($\vec{P}_{s,j}$) executes a task (stored in the first index of its multi-buffer) at a processing rate of $P_{s,j}$ base instructions per second (*n.Hz*). We assume each and every stream processor can be dynamically speed-scaled.

We have $p_{\min,k} \leq P_{s,j} \leq P_{Max,j}$ where $P_{Max,j}$ is the maximum operating frequency in base instructions per second of the j^{th} processing stream's processor; it is a constraint imposed by the hardware specification of the computing device (processor). For a given task $T_k \in T$, its minimum processing rate, $p_{\min,k}$, is a software constraint imposed by the software designer and is generally lower than $P_{Max,j}$ for analytical and practical purposes.

2) *Memory Queue (Multi-Buffer):* A memory queue $\vec{Q}_{s,j}$ of the j^{th} processing stream stores N_j tasks at some instance in time. Therefore $0 \leq N_j < \infty$. in other words, N_j is the occupancy of the j^{th} processing stream's memory queue.

- $N_j = 0$: denotes that the memory queue of the j^{th} processing stream is empty.
- At any given time, the j^{th} stream processor processes a task stored in the first index $(1, j)$ of the memory queue.

D. *A Task's Processing Rate and, Execution Time*

P_k is a task's ($T_k \in T$) processing rate/speed in base instructions per second (n.Hz). t_k is the task's expected execution time in seconds. We relate P_k to t_k in the next section. Overhead switching times prior to processing are assumed to be negligible in comparison to execution times.

E. *A Task's Energy & Power Consumption*

For a task: $T_k \in T$, let Pow_k be the task's expected power function in Watts and let ε_k be the task's expected energy function in Joules when processed by the j^{th} processor. Let us initially assume the task's processing rate (P_k) is time invariant or constant over its execution time (t_k).

$$Pow_k = \lambda_j (P_k)^{\alpha_j} \quad \textit{(Watts)} \tag{1}$$

λ_j, measured in $(J.S^{\alpha_j-1}.n^{-\alpha_j})$, is the energy inefficiency factor or the scaling factor of the j^{th} processor's power function and we assume $\lambda_j > 0$.

α_j is the exponent of the j^{th} processor's power function and it is assumed to be a constant. [18] suggests that $\alpha_j = 1.8$ is a good approximation for CMOS based processors and that $\alpha_j \in (1,3]$ holds for most computer systems comprising of disks, processing chips and servers. We exclude the overhead energy consumed when processors switch speed and also assume the processors consume zero power when idle.

We know that power consumption is the rate of energy consumption; this implies the following.

$$\varepsilon_k = \int_0^{t_k} Pow_k dt = \int_0^{t_k} \lambda_j (P_k)^{\alpha_j} dt = \lambda_j (P_k)^{\alpha_j} t_k \tag{2}$$

B_k relates t_k to P_k, and happens to be the task's remaining computation volume in base instructions (*n*).

$$t_k = \frac{B_k}{P_k} \quad \textit{(Seconds)} \tag{3}$$

Using (2) and (3), we deduce:

$$\varepsilon_k = \lambda_j B_k (P_k)^{\alpha_j - 1} \quad \textit{(Joules)} \tag{4}$$

F. *Decision Algorithm*

The decision algorithm performs three main functions as follows:

1) *Dispatcher:* Addresses which processing stream among the *m* processing streams should process a given task.

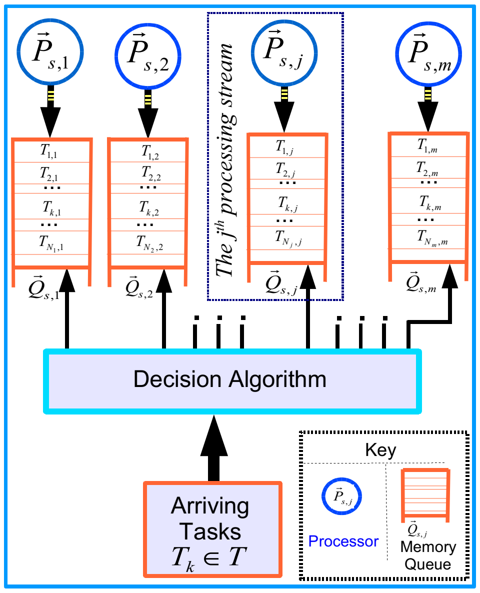
2) *Speed-scaling function:* Determines the explicit optimum processing rate of executing a task/s .

3) *Service discipline / policy:* Specifies the order or discipline in which tasks should be serviced.

IV. PROBLEM FORMULATION

A. *Processing Streams with Multiple Buffers*

Fig. 1 illustrates the parallel multi-buffer scenario where each processing stream has a memory queue that has a capacity to store an arbitrary finite number of tasks. For a set of arriving tasks, we are essentially trying to find the optimum dispatcher, speed scaling function and service discipline that minimizes the total cost of response time and energy consumption (**TCRTEC**) of executing these tasks where the unit price of response time is heterogeneous.



$\vec{P}_{s,1}$
$\vec{P}_{s,2}$
The j^{th} processing stream
$\vec{P}_{s,j}$
$\vec{P}_{s,m}$
$T_{1,1}$ $T_{2,1}$ $T_{k,1}$ $T_{N_1,1}$
$T_{1,2}$ $T_{2,2}$ $T_{k,2}$ $T_{N_2,2}$
$T_{1,j}$ $T_{2,j}$ $T_{k,j}$ $T_{N_j,j}$
$T_{1,m}$ $T_{2,m}$ $T_{k,m}$ $T_{N_m,m}$
$\vec{Q}_{s,1}$ $\vec{Q}_{s,2}$ $\vec{Q}_{s,j}$ $\vec{Q}_{s,m}$
Decision Algorithm
Arriving Tasks $T_k \in T$
Key
$\vec{P}_{s,j}$ Processor
$\vec{Q}_{s,j}$ Memory Queue

Fig. 1: The parallel multi-buffer scenario

B. *The Cost Function of the j^{th} Processing Stream*

Let us assume that the j^{th} processing stream has N_j tasks already queued up in its corresponding memory queue $(\vec{Q}_{s,j})$. Let the aggregate cost function of the j^{th} processing stream be C_j . This cost function aggregates the total cost of expected response time and energy consumption of these N_j tasks. Also let $C_{k,j}$ be the cost of response time and energy consumption of the task stored at the k^{th} index of the $\vec{Q}_{s,j}$ memory queue/multi-buffer.

Using vector notation and dot product operations, we have:

$$C_j = \sum_{k=1}^{N_j} C_{k,j} = \sum_{k=1}^{N_j} \left\{ U_k \bullet \left[\varepsilon_k , \sum_{r=1}^{k} t_r \right] \right\}$$

More explicitly, using (3) and (4) we have:

$$C_j = \sum_{k=1}^{N_j} \left\{ u_\varepsilon \lambda_j B_k (P_k)^{\alpha_j - 1} + \left(\frac{B_k}{P_k} \sum_{r=k}^{N_j} u_{t,r} \right) \right\} \qquad (5)$$

C. *The Minimized Cost Function of the j^{th} processing stream*

For each j^{th} stream, we have an " N_j " dimensional optimization problem. The adjustable parameters are the processing rates (P_k) of the tasks: $T_k \in T \mid k \in \{1,2...N_j\}$ as well as their service sequence in the j^{th} processing stream. We optimize C_j as defined by (5) as follows.

$$\frac{\partial C_j}{\partial P_k} = (\alpha_j - 1) u_\varepsilon \lambda_j B_k (P_k)^{\alpha_j - 2} - \frac{B_k}{P_k^{\,2}} \sum_{r=k}^{N_j} u_{t,r} = 0$$

for $k \in \{1,2...N_j\}$

.

The solution of our optimization problem is as follows:

$$P_k = \left(\frac{1}{(\alpha_j - 1) u_\varepsilon \lambda_j} \sum_{r=k}^{N_j} u_{t,r} \right)^{\frac{1}{\alpha_j}} \quad \text{for } k \in \{1,2..N_j\}$$

and $\alpha_j \in (1,3]$

Using a Hessian matrix [14], it was confirmed that this set of critical points (processing rates) globally minimizes C_j .

D. *The Minimized Constrained Cost Function of the j^{th} Processing Stream*

$\forall T_k \in T \mid T_k \in \vec{Q}_{s,j}$, let us factor in the task and processor stream processing constraints mentioned earlier (III.C.1). We enforce $P_{Max,j} \geq P_k \geq p_{\min,k}$ where, $p_{\min,k}$ is the task's minimum recommended execution rate in base instructions per second (*n.Hz.*). The minimum constrained cost function that factors the processing constraints is:

$$C_{j_{\min}}(N_j) = \sum_{k=1}^{N_j} \left(u_\varepsilon \lambda_j B_k (P^*_k)^{\alpha_j - 1} + \frac{B_k}{P^*_k} \sum_{r=k}^{N_j} u_{t,r} \right) \qquad (6)$$

and (7)

$$P^*_k = \begin{cases} \left(\frac{1}{(\alpha_j - 1)u_\varepsilon \lambda_j}\sum_{r=k}^{N_j} u_{t,r}\right)^{\frac{1}{\alpha_j}}, \text{ if } P_{Max,j} \geq \left(\frac{1}{(\alpha_j - 1)u_\varepsilon \lambda_j}\sum_{r=k}^{N_j} u_{t,r}\right)^{\frac{1}{\alpha_j}} \geq p_{\min k} \\ p_{\min k}, \text{ if } \left(\frac{1}{(\alpha_j - 1)u_\varepsilon \lambda_j}\sum_{r=k}^{N_j} u_{t,r}\right)^{\frac{1}{\alpha_j}} < p_{\min k} \\ P_{Max,j}, \text{ if } \left(\frac{1}{(\alpha_j - 1)u_\varepsilon \lambda_j}\sum_{r=k}^{N_j} u_{t,r}\right)^{\frac{1}{\alpha_j}} > P_{Max,j} \end{cases}$$

for $k \in \{1,2...N_j\}$ and $\alpha_j \in (1,3]$

P^*_k is the optimum constrained processing rate of potentially executing the task stored in the k^{th} index of the $\vec{Q}_{s,j}$ memory queue.

V. ALGORITHMS DESCRIPTION

This section describes the SMBSPP algorithm. First we describe our MMCVITPS dispatcher and our SCVPPT scheduling policy. Then we present our algorithm.

*A. SMBSPP Algorithm's Dispatcher (**MMCVITPS**)*

Before presenting the complete algorithm description (V.C), let us describe in words how its dispatcher (MMCVITPS) works. For an arriving task, MMCVITPS hypothetically or virtually assumes the potential aggregate cost of virtually introducing the task (according to a service discipline) to each of the processing streams. It then virtually minimizes the aggregate cost function of each processing stream by again virtually re-adjusting the processing rates of all tasks in the queues (of each processing stream) including the task in question. It then finally decides on the processing stream with the lowest potential (minimized) aggregate cost. This decision will dynamically affect the speed function of the chosen processing stream's processor. We mathematically describe the dynamic speed scaling function in section V.C.

*B. SMBSPP Algorithm's Service Discipline (**SCVPPT**)*

In this service discipline, arriving tasks are sorted in each processing stream's memory queue or multi-buffer from the lowest index (highest priority) according to their Smallest remaining computation volume per unit price of response time $(B_k / u_{t,k})$.

*C. Single-threading Multi-buffer Scheduling & Processing Algorithm (**SMBSPP**)*

1. User or OS specifies u_ε for all tasks and may specify different $u_{t,k}$ for each $T_k \in T$.
2. For an arriving task, $T_k \in T$, we evaluate and compare the minimum potential processing cost, $C_{j\min}(N_j + 1)$ of virtually introducing and processing the arriving task in each of the available processing streams $(1 \leq j \leq m)$. The task virtually acquires a position index according to $B_k / u_{t,k}$ (**SCVPPT**) in each of the processing streams.
3. Using (6) and (7), the task should follow a stream j* such that : $C_{j^*\min}(N_{j^*} + 1) = \min_{1\leq j \leq m}\{C_{j\min}(N_j + 1)\}$ thereby it acquires the position index according to $(B_k / u_{t,k})$ **(SCVPPT)** and will be processed by the $\vec{P}_{s,j^*}$ processor at some adjusted optimum processing rate.
4. Update N_{j^*} .
5. The task stored at system index $(1, j^*)$ i.e., the task T_{1,j^*} , is executed by the $\vec{P}_{s,j^*}$ processor at the optimum adjusted processing rate defined below[6]:

$$P_{s,j^*} = \begin{cases} \left(\frac{1}{(\alpha_{j^*} - 1)u_\varepsilon \lambda_{j^*}}\sum_{r=1}^{N_{j^*}} u_{t,r}\right)^{\frac{1}{\alpha_{j^*}}}, \text{ if } P_{Maxj^*} \geq \left(\frac{1}{(\alpha_{j^*} - 1)u_\varepsilon \lambda_{j^*}}\sum_{r=1}^{N_{j^*}} u_{t,r}\right)^{\frac{1}{\alpha_{j^*}}} \geq p_{\min 1} \\ p_{\min 1}, \text{ if } \left(\frac{1}{(\alpha_{j^*} - 1)u_\varepsilon \lambda_{j^*}}\sum_{r=1}^{N_{j^*}} u_{t,r}\right)^{\frac{1}{\alpha_{j^*}}} < p_{\min 1} \\ P_{Maxj^*}, \text{ if } \left(\frac{1}{(\alpha_{j^*} - 1)u_\varepsilon \lambda_{j^*}}\sum_{r=1}^{N_{j^*}} u_{t,r}\right)^{\frac{1}{\alpha_{j^*}}} > P_{Maxj^*} \end{cases}$$

6. Repeat steps 4 & 5 whenever a task/s is either dynamically introduced or deleted in $\vec{Q}_{s,j^*}$ [3].
7. Once the execution of the task T_{1,j^*} is complete or terminated, the indices of all tasks in memory queue $\vec{Q}_{s,j^*}$ are shifted down by one creating room for another task.[2, 3]
8. If any task or tasks in $\vec{Q}_{s,j^*}$ are deleted/cancelled, each alive task in $\vec{Q}_{s,j^*}$ is shifted to the minimum available slot starting from the first index to preserve task priority before steps 4 & 5.[2, 3]
9. If we are to enforce FCFS queuing service policy or we are not allowed to exercise preemption, whenever a task enters the queue of a processing stream it acquires the Smallest Empty Index (SEI), also in step 2, while calculating the virtual cost of introducing the task to each processing stream, the arriving task virtually acquires the SEI[4].
10. Ignore steps 2 & 3 when processors are homogeneous and instead utilize Round Robin dispatching[5].

Steps 2 & 3 summarize the SMBSPP algorithm's default dispatcher (**MMCVITPS**) under the **SCVPPT** service discipline. Step 5 describes the speed scaling function (**OSTSCF**).

Notes pertaining to algorithm's description:

[2] Steps 7 and 8 are maintenance operations that facilitate the long-run functionality of the algorithm.
[3] Steps 3, 4, 7 and 8 can be implemented by *adjuster* modules that dynamically make changes and keep track of the memory queue environment of each processing stream.
[4] Step 9 may degrade the performance of the algorithm.
[5] Step 10 improves the algorithm's computational complexity when processors are homogeneous, but should not be conducted when processors differ in terms of their power degree constants or maximum processing rates.

VI. SIMULATIONS

A. Performance Metrics

Table I provides a list (with abbreviations and standard units) of some performance metrics. All experiments are run using the OSTCF speed scaling function.

TABLE I
PERFORMANCE METRICS

METRIC	DEFINITION	UNITS
TET	Total execution time of executing N tasks	ms
TET/N	Average execution time of executing N tasks	ms/task
TCRTEC	Total cost of response time and energy consumption for executing N tasks	CDN$
TCRTEC/N	Average cost of response time and energy consumption for executing N tasks	CDN$/task
ST	System time of executing N tasks: amount of time that at least one processor is active	ms
TSSC	Total cost of system time and energy consumption for executing N tasks	CDN$
TSSC/N	Average cost of system time and energy consumption for executing N tasks	CDN$/task

In Table I, the metrics in bold are used to evaluate the algorithm.

B. Simulation I: Sensitivity of SMBSPP Algorithm To inter-arrival periods

The preliminary simulation assumptions are as follows:

- We have an N number of homogenous tasks each with a computation volume of 100 base instructions.
- We have three processors. Their power functions have the following energy inefficiency factors: $\lambda_1 = 1.08$, $\lambda_2 = 1.0$ and $\lambda_3 = 0.92$ $(J.S^{\alpha_1 - 1}.n^{-\alpha_1})$ with the following corresponding exponents: $\alpha_1 = \alpha_2 = \alpha_3 = 1.8$
- In this simulation, the computation volumes and unit price of response times for all tasks are homogenous so as to eliminate the effect of service disciplines, i.e. FCFS, SRPT and SCVPPT all behave in the same way.
- The unit price of energy is $u_\varepsilon = 3.\overline{4}x10^{-8}$ *$/Joule* and the unit price of response time is $u_t = 2.8472\overline{2}x10^{-3}$ *$/Second* (see section III.B for details).
- For each simulation iteration, we utilize the TET/N, TCRTEC/N and TSSC/N performance metrics to evaluate the effect of deterministic and stochastic arrival periods.
- All this was repeated for growing values of N (simulation iterations).
- Results were confirmed using discrete-time based simulations written in Java.

Following these assumptions, the figures below summarize the simulation results.

TABLE II
INTERPRETATION OF INTER-ARRIVAL PERIODS

INTER-ARIVAL PERIOD	INTERPRETATION
μ = 0ms	Extreme (batch arrivals)
μ = 26.1ms	Heavy
μ = 50ms	Almost ideal
μ ≥ 156.4ms	Minimal (no traffic)

Fig. 2(**a**) exhibits how the SMBSPP algorithm utilizes dynamic speed-scaling to adapt to various traffic conditions (speed increases with aggregate occupancy pricing).

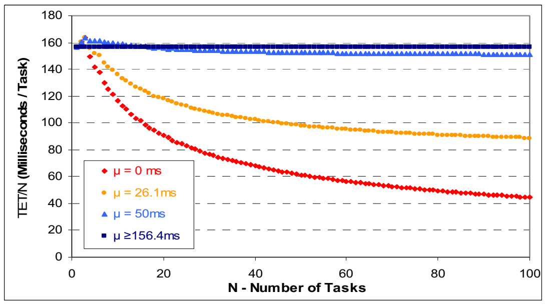


Fig. 2 (**a**)

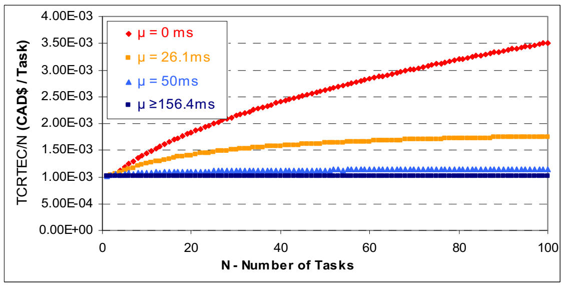


Fig.2 (**b**)

Fig. 2: Showing Effect of Deterministic Arrival Periods (μ) by: Average Execution Time for N Homogeneous Tasks (**a**), and Average Cost of Response Time & Energy Consumption for N Homogeneous Tasks (**b**)

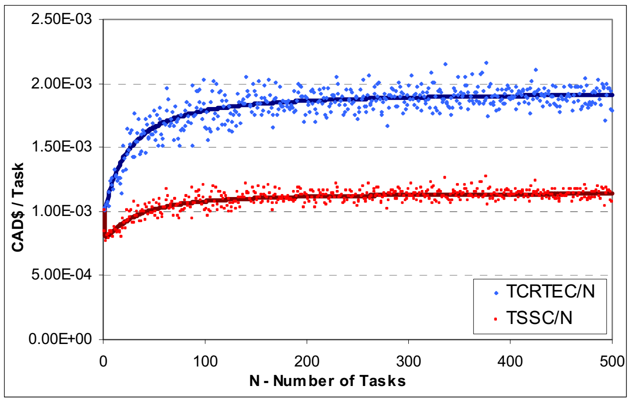


Fig. 3: Average Cost of Response Time & Energy Consumption Versus Average Cost of System Time & Energy Consumption for N Homogeneous Tasks under Exponentially Distributed Arrival Periods with a Mean of 1/μ (μ =26ms: heavy traffic) (The results for deterministic arrival periods is interpolated by continuous curves).

In Fig. 3, the TSSC/N performance metric is a convenient metric in the sense that it is actually the amount in dollars per task that it costs to lease out computation services. The reason why the TSSC/N curve falls way below the TCRTEC/N metric is due to multiple processors working in parallel; where the TSSC/N metric charges the global timescale as can be experienced by a user while TCRTEC/N factors response times of each task leading to multiple aggregation of delays. The fact that the algorithm has a fairly constant TSSC/N curve under heavy stochastic traffic conditions reveals its robustness.

C. *Simulation II: Comparing SMBSPP Algorithm's Dispatcher (MMCVITPS) Versus Round Robin Dispatcher under FCFS, SRPT and SCVPPT service disciplines.*

The preliminary simulation assumptions are as follows:

- We have an N number of heterogeneous tasks whose computation volumes is Gaussian distributed with a mean of 100 base instructions and a standard deviation of 20% mean.
- We have three processors with the following power function parameters (respectively):

  $\lambda_1 = 1.08$, $\lambda_2 = 1.0$ and $\lambda_3 = 0.92$ $(J.S^{\alpha_1 - 1}.n^{-\alpha_1})$

  and $\alpha_1 = 1.944$, $\alpha_2 = 1.8$ and $\alpha_3 = 1.656$.
- The unit price of energy is $u_\varepsilon = 3.\overline{4}x10^{-8}$ *$/Joule* and the unit price of response time is Gaussian distributed with a mean of $u_t = 2.84\overline{72}x10^{-3}$ *$/Second* and a standard deviation of 25 % of the mean.
- For each simulation iteration, the SMBSPP Algorithm runs using its default Dispatcher (MMCVITPS) and independently runs using the Round Robin Dispatcher using the **same input data** for various service disciplines.
- All this is repeated for growing values of N (simulation iterations).
- We assume very heavy traffic conditions with exponentially distributed arrival periods .

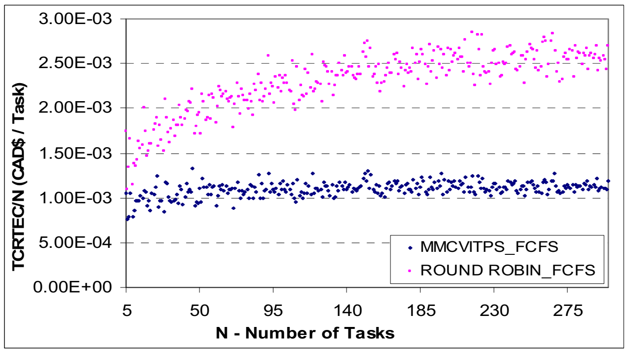


Fig. 4 (**a**)

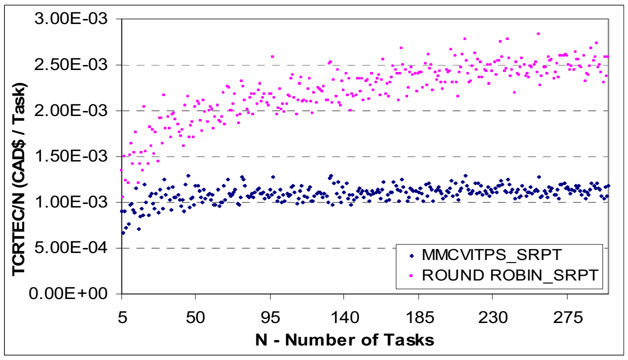


Fig. 4(**b**)

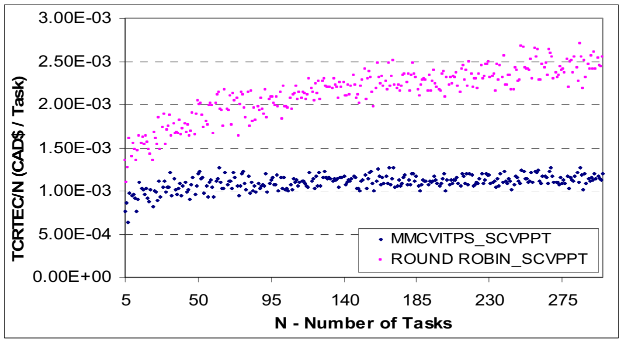


Fig. 4 (**c**)

Fig. 4: MMCVITPS Versus Round Robin for N Heterogeneous Tasks under Exponentially Distributed Arrival Periods (heavy traffic) with Heterogeneous Unit Prices of Response Time under FCFS (**a**), SRPT (**b**) and SCVPPT (**c**) Service Disciplines.

In this simulation, the power function parameters were conservatively chosen to differ from the mean by at most 8%. Presumably, let this 8% deviation be attributed to: the manufacturing error of fabricating homogeneous processors, failing to achieve equal temperature environments for all processors or even the **intentional** fabrication of heterogeneous processors due to design budget constraints.

In Fig. 4(a-c) we show that the algorithms dispatcher (MMCVITPS) out performs the Round Robin dispatcher under the FCFS, SRPT and SCVPPT service disciplines under heavy stochastic traffic conditions. The tasks have heterogeneous computation volumes and heterogeneous unit prices of response time.

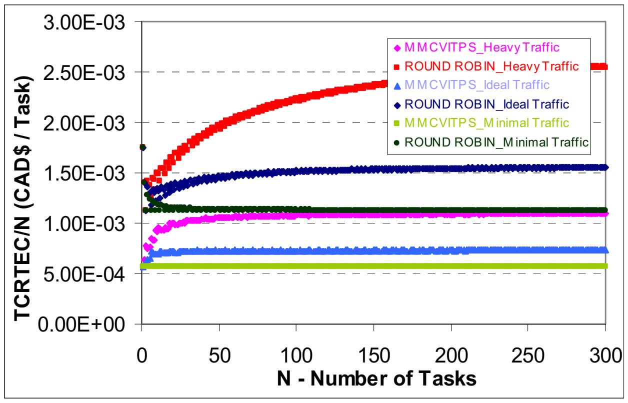


Fig. 5: MMCVITPS Versus Round Robin for N Homogeneous Tasks under Three Main Deterministic Arrival Periods with Homogeneous Unit Prices of Response Time. (The three service disciplines are equivalent and have no effect in this scenario).

Fig. 5 shows that the MMCVITPS dispatcher outperforms the Round Robin dispatcher under three main deterministic arrival periods that correspond to very heavy, ideal and minimal traffic conditions. In Figs. 4(a-c) and 5 the processors' power function parameters were conservatively chosen to differ from the mean by at most 8% yet the MMCVITPS dispatcher drastically outperformed the Round Robin dispatcher with cost savings exceeding 100% on average.

D. *Simulation III: Evaluating SMBSPP Algorithm's Dispatcher (MMCVITPS) under FCFS, SRPT and SCVPPT service disciplines.*

Using the assumptions of Simulation II, we compare the MMCVITPS dispatcher under the three service disciplines. In Figure 6, we show that the SCVPPT service discipline always minimizes TCRTEC making it the most ideal for the SMBSPP algorithm with its default dispatcher. We recommend that the SCVPPT service discipline be implemented in any online speed-scaling algorithm that aims to minimize TCRTEC and considers tasks with heterogeneous unit prices of response time.

All of the simulation results are consistently scalable in terms of considering tasks with substantially larger computation volumes, but the simulation run times will take longer and will require a calibration of the inter-arrival periods (and their categorizations). In practice, we lightly suggest implementing the MMCVITPS dispatcher in ad-hoc hardware to guarantee performance, but the actual cost of doing so warrants further investigation. However, we are currently working on enhancing its computational complexity.

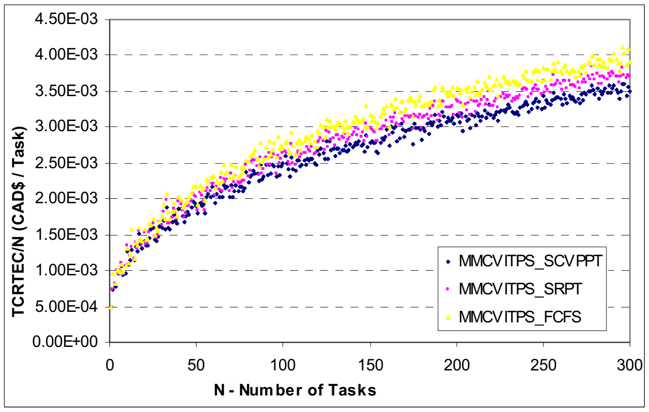


Fig. 6: MMCVITPS Dispatcher Performance under SCVPPT, SRPT and FCFS Service Disciplines for N Heterogeneous Tasks that have Exponentially Distributed Arrival Periods with a Mean of 1/μ (almost extreme traffic of μ = 2ms) with Heterogeneous Unit Prices of Response Time (Gaussian distributed).

VIII. CONCLUSION

We have synthesized and simulated an online multiprocessor scheduling algorithm (SMBSPP) for optimum parallel computing of portable devices or energy-aware workstations. We focused on single threading where no processor executes more than a single task at any given time until completion unless preemption is dictated by the service discipline e.g. SCVPPT. The SMBSPP algorithm provides some insights. It tells us that the optimum processing rate of a task is not a function of its computation volume (B_k). It also tells us once a task is dynamically included into a given memory queue of a processing stream, the optimum processing rate of the currently processed task (stored at the first index of the queue) is likely to change. The processing rate changes because the aggregate cost function of all tasks in the queue has changed and there exists a time dependency among tasks in the processing stream's memory queue due to single-threading. The algorithm explicitly finds a globally optimum solution for each aggregate cost function associated with each processing stream. This globally optimum solution minimizes the total cost of both energy consumption and response time of tasks in each processing stream. The solution explicitly obtains the optimum processing rates of each task in all memory queues. We believe this robustness of the algorithm being able to handle dynamic inclusion of heterogeneous tasks at run-time makes it appealing among hardware architectural planers and software programmers of portable computing devices.

Assuming each processing stream has roughly n tasks queued up, the algorithm's default dispatcher (MMCVITPS) has a worse case computational complexity of $O(n^2)$ with heterogeneous pricing and $O(n)$ with homogenous pricing, and when it uses the Round Robin dispatcher, it has a worse case computational complexity of $O(1)$. In terms of the TCRTEC/N metric, we demonstrated that the algorithms default dispatcher (MMCVITPS) significantly out performs the Round Robin dispatcher under the FCFS, SRPT and SCVPPT service disciplines for various stochastic and deterministic traffic conditions where the degree of processor heterogeneity was mild (power function parameters were conservatively chosen to differ from the mean by at most 8%), yet the MMCVITPS dispatcher drastically outperformed the Round Robin dispatcher with cost savings exceeding 100% on average. In fact, we do not recommend the use of the Round Robin dispatcher in systems that utilize heterogeneous processors. If the SMBSPP algorithm is to be implemented in devices with homogeneous processors, the Round Robin dispatcher would be more ideal to use because it would produce results equal to MMCVITPS, but with a lower worse case computational complexity as mentioned previously.

Through simulation, we demonstrated that the SMBSPP algorithm with its default dispatcher (MMCVITPS), service discipline (SCVPPT) and speed-scaling function (OSTSCF) has a fairly constant TSSC/N curve under heavy stochastic traffic conditions; this reveals the algorithm's robustness. It makes it suitable to be implemented in energy aware work stations or "green" computational devices that utilize parallel processors and want to maintain a fairly stable (constant) operation cost under unpredictable heavy traffic conditions.

The proposed SCVPPT service discipline always matches or outperforms the FCFS and SRPT service disciplines as evaluated by the TCRTEC performance metric. When implemented in the algorithm, the SCVPPT and SRPT service disciplines each have computational complexities of $O(\log n)$. where n is the occupancy of a given processor's memory queue. SCVPPT behaves exactly like SRPT when the unit price of response time is fixed and equivalent for all tasks; thereby it minimizes total response time. SCVPPT is sort of a generalized version of SRPT but is flexible. It allows a user to maintain or even improve the priority of a large task by accepting to set/pay a higher unit price of response time or even degrade the priority of a small non-urgent task by setting a sufficiently small unit price of response time. This is a dynamic feature that is absent in both FCFS and SRPT service disciplines. We recommend that the SCVPPT service discipline be implemented in any online speed-scaling algorithm that aims to minimize TCRTEC and considers tasks with heterogeneous unit prices of response time.

ACKNOWLEDGMENT

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] N. Avrahami, and Y. Azar. "*Minimizing total flow time and total completion time with immediate dispatching*". SPAA, pp. 11–18, 2003.

[2] Albers, S., Antoniadis, A. and Greiner, G. "*On Multi-Processor Speed Scaling with Migration*". SPAA, pp. 279–288, 2011.

[3] Albers, S. and Fujiwara, H. "*Energy-efficient algorithms for flow time minimization*". Proc. 23rd Annual Symposium on Theoretical Aspects of Computer Science (STACS), Springer LNCS 3884, pp. 622–633, 2006.

[4] Albers, S., Muller, F. and Schmelzer, S. "*Speed Scaling on Parallel Processors*". SPAA, pp. 289-298, 2007.

[5] Angel, E., Bampis, E., Kacem, F. and Letsios, D. "*Speed Scaling on Parallel Processors with Migration**". Euro-Par, pp.128-140, 2012.

[6] Asanović, K., et al., "*The Landscape of Parallel Computing Research: A View from Berkeley*" EECS Department, University of California, Berkeley, pp.22, Tech. Rep. UCB/EECS-2006-183, December 2006.

[7] Bansal, N., Kimbrel, T. and Pruhs, K. "*Dynamic speed scaling to manage energy and temperature*" Proc. 45th Annual IEEE Symposium on Foundations of Computer Science, pp. 520–529, 2004.

[8] Bansal, N., Kimbrel T. and Pruhs, K. "*Speed scaling to manage energy and temperature*", J. ACM 54 (1) , pp. 1–39, 2007.

[9] Bansal, N., Pruhs, K., Stein, C., "*Speed scaling for weighted flow time*". In: Proc. of 18th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA'07), pp. 805–813, 2007.

[10] Bower, F.A., Sorin, D.J. and L.P. Cox. "*The impact of dynamically heterogeneous multicore processors on thread scheduling*". Micro, IEEE, 28(3), pp. 17 –25, 2008.

[11] Brooks, D.M., Bose, P., Schuster, S.E., Jacobson, H., Kudva, P.N., Buyuktosunoglu, A., Wellman, J.-D., Zyuban, V., Gupta, M., Cook, P.W., "*Power-aware microarchitecture: design and modeling challenges for next-generation microprocessors*". IEEE MICRO 20(6), pp. 26–44, 2000.

[12] Bunde, D.P., "*Power-aware scheduling for makespan and flow*", SPAA, pp. 190–196, 2006.

[13]Koufaty et al., "*Bias scheduling in heterogeneous multi-core architectures,*" EuroSys 2010

[14] Gradshteyn, I. S. and Ryzhik, I. M. "*Hessian Determinants*" §14.314 in Tables of Integrals, Series, and Products, 6th ed. San Diego, CA: Academic Press, pp. 1069, 2000.

[15] Greiner, G. , Nonner, T. and Souza, A. "*The bell is ringing in speed-scaled multiprocessor scheduling*", SPAA, pp. 11-18, 2009.

[16] Gupta, A., Im, S., Krishnaswamy, R., Moseley, B. and Pruhs, K., "*Scheduling heterogeneous processors isn't as easy as you think*", Proc. of the Twenty-Third Annual ACM-SIAM Symp. on Discrete Algorithms pp. 1242-1253.

[17] Hydro One. (2013, May). "BUILDING YOUR BILL: prices & rates" [Online].Available:http://www.hydroone.com/RegulatoryAffairs/RatesPrices/Pages/Default.aspx Access on 2013, June 19.

[18] LLH Andrew, M Lin, A Wierman "*Optimality, fairness, and robustness in speed scaling designs*", SIGMETRICS '10 Proceedings of the ACM SIGMETRICS international conference on Measurement and modeling of computer systems, Pages 37-48, 2010.

[19] Lam, T.W., Lee, L.-K., To, I.K.-K., Wong, P.W.H. "*Competitive non-migratory scheduling for flow time and energy*". In: Proc. of the 20th Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA'08), pp. 256–264, 2008.

[20] Pruhs, K., Sgall, J. and Torng, E. "*Online scheduling*". In J. Leung, editor, Handbook of Scheduling: Algorithms, Models and Performance Analysis, pp. 15-1–15-41. CRC Press, 2004.

[21] Ontario Ministry of Labour. (2013, May). "Minimum Wage" [Online].Available:http://www.labour.gov.on.ca/english/es/pubs/guide/minwage.php. Access on 2013, June 19.

[22] Pruhs, K., Uthaisombut, P. and Woeginger, G. "*Getting the best response for your erg*" Proc. 9th Scandinavian Workshop on Algorithm Theory (SWAT), Springer LNCS 3111, pp.15–25, 2004.

[23] Pruhs, K., van Stee, R. and Uthaisombut, P., "*Speed scaling of tasks with precedence constraints*", Theory Comput. Syst. 43 (1), pp. 67–80, 2008.

[24] Tomer, Y., Morad, Uri C.Weiser, Avinoam Kolodny, Mateo Valero, and Eduard Ayguade. "*Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors*". IEEE Comput. Archit. Lett., 5:4–, January 2006.

[25] Wikipidea. (2013, Feb 1). *"PlayStation Vita"* Wikipidea [Online]. Available: http://en.wikipedia.org/wiki/PlayStation_Vita. Access on 2013, Mar 10.

[26] Yao, F., Demers, A. and Shenker, S. "*A scheduling model for reduced CPU energy*" Proc. 36th Annual Symposium on Foundations of Computer Science, pp.374–382, 1995.

[27] Yuan, L., and Qu, G. "*Analysis of energy reduction on dynamic voltage scaling-enabled systems*", IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 24 (12), pp. 1827–1837, 2005.