Time of Use Tariff parameter Estimation: A Data Analysis Approach on Multicore Systems

Amit Kalele
Center of Excellence for High Performance Computing
Tata Consultancy Services, Pune
kalele.amit@tcs.com

Kiran Narkhede
Center of Excellence for High Performance Computing
Tata Consultancy Services, Pune
narkhede.kiran@tcs.com

Mayank Bakshi
Center of Excellence for High Performance Computing
Tata Consultancy Services, Pune
mayank.bakshi@tcs.com

ABSTRACT

Increased use of solar energy is forcing energy companies to devise new time of use (ToU) tariff scheme to counter revenue losses. Designing ToU tariff scheme is a complex multistage problem. The adoption of smart meters and availability of high performance multicore systems has opened up newer and better ways of tariff design. The design of ToU tariff schemes typically involves identifying various demand periods which is accomplished by analyzing the intraday consumption patterns across various geographies. The optimal tariff parameters for all the demand periods is then computed by solving a constrained optimization problem. In this paper, we present a present multi dimensional grid search approach to compute the optimal tariff parameters for a ToU scheme. The grid search method was then efficiently implemented on Nvidia GPUs with dynamic parallelism. The annual energy consumption data processing for nearly 0.3 million consumers and computation of consumption pattern and demand periods was carried out using MPI based parallel processing on an Intel Haswell system.

Keywords

Time of use tariff Modeling; HPC; GPU; Quadratic Optimization; Parallel Computing; Big Data Analytics

1. INTRODUCTION

With the advent of efficient and cost effective conversion of solar energy, its domestic usage is rapidly increasing. The traditional energy companies are facing erosion in revenues due to increasing usage of solar energy. Even though the demand for conventional energy is reduced, the companies have to maintain full capacity distribution network and equipment. Simply increasing the tariff is not an option for the energy companies as it is regulated by authorities and it also has negative impact with more consumers opting for solar options. This takes its toll on revenues. To counter these financial losses, energy companies are devising new tariff schemes, which has bearing in variable charges based on energy demand.

Energy companies can apply new tariff rates for different time of the usage by analyzing consumption patterns [7, 11], [5] and [3], while consumers can benefit by actively changing their energy usage time. For example, users can maximize consumption during off peak times and can reduce their energy bills. In the ToU tariff, the usage of energy or electricity for different periods of a day is charged at different rates. Typically, periods which observe higher demand are termed as peak periods and are charged at higher rates. Similarly, periods with low demand are termed as off peak periods and are charged at lower rate. Depending on the tariff scheme, the intermediate hours can further have multiple periods and their respective changes.

Computing optimal tariff parameters for peak, off-peak or intermediate hours such that revenue deficit can be minimized while adhering to the regulatory guidelines is a complex computational problem. This problem involves optimization and computing consumption pattern of the entire consumer base [10, 9, 2].

Computing consumption pattern requires processing and analyzing years of consumption data (metered readings) of entire consumer base. Several consumer categories, different geographical locations and respective regulatory constraints further add to the computational complexity. These problems naturally fall in the realms of high performance computing and data analytics and can be effectively solved using available technologies, like Hadoop map-reduce or spark or MPI based cluster processing etc.

In this paper, we consider nearly 0.3 million consumers, each with one year of consumption data with half hourly readings. And a ToU tariff scheme which has four different periods of energy demand. We also present a two phased approach to compute optimal tariff parameter for the each demand period. The first phase involves filtering and analysis of the entire data for computing the over all consumption pattern and the demand periods. The second phase computes the optimal tariff parameters for each energy demand period.

We now enumerate key features of the work and provide a brief outline of the paper.

1 A quadratic optimization problem for computing optimal parameters for the considered ToU tariff scheme is formulated. This optimization problem is further translated into a 5-dimensional grid search problem.

2 A randomized adaptive n-dimensional grid search al-

ICPE '17 Companion, April 22 - 26, 2017, L'Aquila, Italy

ISBN 978-1-4503-4899-7/17/04...$15.00

DOI: http://dx.doi.org/10.1145/3053600.3053623

gorithm is presented and efficiently implemented on Nvidia GPUs exploiting dynamic parallelism technology. This algorithm starts with a coarse search grid and iteratively performs finer searches only in specific regions of the search space. The regions are formed dynamically.

3 One year of consumption data for 0.3 million consumers with daily readings with an interval of 30 minutes was provided in CSV files. We present a MPI based approach to filter the data and calculate annual consumption of each consumer and total annual consumption of all consumers. This data was further processed to calculate four distinct demand periods.

The following table presents the glossary of terms and variables used in the paper.

Terms	Definition
ToU	Time of Use tariff
$\alpha_{fix}, \alpha_o, \alpha_s, \alpha_p, \alpha_c$	Tariff rate for fixed, off peak, shoulder, peak and critical peak demands
E_o, E_s, E_p, E_c	Energy consumption for fixed, off peak, shoulder, peak and critical peak periods
$\hat{R}$, R and Δ_t	Revenue under ToU, Reference revenue and time period in days respectively
$*$	General notation to represent the {fix, o, s, p, c}
l_*, u_* and τ_*	Lower and upper bound on values of α_* and discretization step size

Table 1: Glossary of Terms

2. TARIFF SCHEME

Traditionally, domestic consumer tariffs have only two components namely daily fixed charge ($/day$) and a flat tariff rate ($/kWh$). The revenue computation based on this scheme can be described as follows. Let Δ_t denote the time period under consideration expressed in days, α_{fix} denote the fixed per day charge, α_{flat} denote the flat tariff rate and $E(\Delta_t)$ denote the energy consumption for the time period Δ_t. The revenue R for the time period Δ_t can given as:

$$R(\Delta_t) = \{\Delta_t \cdot n_{user} \cdot \alpha_{fix}\} + \{\alpha_{flat} \times E(\Delta_t)\} \quad (1)$$

But with the cheap solar energy, this model is not effective and to mitigate the financial losses, new tariff schemes are being devised and experimented. These schemes introduce variable tariffs ($/KWh$) based on the intra-day demand for the energy along with the fixed tariff ($/day$) [16]. A typical energy demand in a 24 hour cycle can be described by the figure (1) below:

Based on the above representative energy demand pattern, a ToU model or scheme can be defined. Instead of a flat tariff rate for the entire day, we consider four demand periods namely off-peak, shoulder, peak and critical peak.

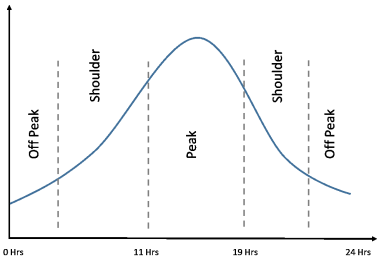

Figure 1: Energy Demand in a day

Under this model, each demand period is charged at different rate and a fixed per day charge is also levied. The time of use tariff pricing is well research area and more details can be found in [1, 6].

The off-peak, shoulder and peak demand periods are computed from the consumption pattern, but the critical peak period is calculated as follows. From the previous yearâĂŹs consumption data, twelve maximum consumption days are selected and aggregate consumption in these twelve days is considered and charged with critical peak rate.

Let α_* and E_* denote the tariff rate or parameter and the energy consumption for the $(*)$ demand period respectively. Then the revenue $\hat{R}$ under this model for a time period Δ_t can be given as follows:

$$\hat{R}(\Delta_t) = \{\Delta_t \cdot n_{user} \cdot \alpha_{fix}\} + C_{var}(\Delta_t) \quad (2)$$

Where α_{fix} is the fixed per day charge and C_{var} is defined as:

$$\begin{aligned} C_{var} = \alpha_o \cdot \sum_{nuser} E_o(\Delta_t) + \alpha_s \cdot \sum_{nuser} E_s(\Delta_t) \\ + \alpha_p \cdot \sum_{nuser} E_p(\Delta_t) + \alpha_c \cdot \{\sum_{j=1}^{12} E_c^j\} \end{aligned} \quad (3)$$

The above equations defines the revenue calculation under the ToU scheme. To calculate the revenue $\hat{R}$ it is required to know the α_* and E_*. We consider the problem of computing α_* and E_* in the following sections.

3. TARIFF RATE OPTIMIZATION

The ToU model discussed in the previous section levy charges as per the severity of the demand. It can be seen from the equation 3 that the revenue $\hat{R}$ directly depends on the the tariff rate parameters α_*. The energy companies cannot choose these parameters arbitrarily. The choice of these parameters is governed by the following constraints.

- The resulting revenue under ToU model $\hat{R}$ and the reference revenue R should be approximately equal $\hat{R} \approx R$. If the choice of α_* results in very high revenue i.e. $\hat{R} \gg R$, then it will be a violation of regulatory guidelines. On the other hand if rate parameters are such that the resulting revenue is too low i.e. $\hat{R} \ll R$, then it is a financial loss to the company which of course is not acceptable.

- The value of each parameter α_* must satisfy their respective range.

- The critical demand days should be charged at higher rate than peak demand.

The desired tariff rate parameters satisfying above constraints can be computed by solving the following standard optimization.

PROBLEM 3.1. *Let $\hat{R}$ be the revenue under considered ToU model and is defined by equations 2 and 3 then*

$$\begin{aligned} \underset{\alpha_*}{\textit{minimize}} \quad & \text{abs}\{\hat{\text{R}}(\alpha_*) - \text{R}\} \\ \textit{subject to} \quad & 0.9 \leq \alpha_{fix} \leq 1.0 \\ & 0.166 \leq \alpha_o \leq 0.199 \\ & 0.2 \leq \alpha_s \leq 0.3 \\ & 0.3 \leq \alpha_p \leq 1.0 \\ & 0.5 \leq \alpha_c \leq 5.0 \\ & \alpha_c \geq \alpha_p \end{aligned}$$

Since the absolute difference is of interest for minimization rather than the simple difference, the problem can be formulated in multiple ways. One such way would be to consider square of the difference $\{\hat{R}(\alpha_*) - R\}^2$ which results in a modified cost function:

$$\underset{\alpha_*}{\text{minimize}} \quad \{\hat{\text{R}}^2 + \text{R}^2 - 2\hat{\text{R}}\text{R}\}$$

The gradient based methods like GRG algorithm are well suited for solving such problems [4]. But the parameter computed from GRG were suboptimal and resulted in loss of $\approx \$19K$.

3.1 Grid search Approach

The cost function $abs(\hat{R}(\alpha_*) - R)$ minimization would result in best approximation of original revenue R. But in practice, especially when revenue is in order of hundreds of million dollars, it would be sufficient to have absolute revenue difference below some threshold δ, where δ could be of order of few tens of dollars rather than the global minimum.

Under this practical scenario, the objective of finding the global minimum can be relaxed and the optimization problem can be formulated as a search problem with a threshold. In what follows, we present the grid search problem and an approach to on GPUs.

The traditional way of performing parameter optimization has been grid search, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the parameter space bounded by the range of parameters.

Since the parameters are real-valued, it would be necessary to discretize the parameter space (with some finite step size) before applying grid search. We present the mathematical formulation of the search problem below.

As defined earlier, let $(\alpha_{fix}, \alpha_o, \alpha_s, \alpha_p, \alpha_c)$ be the five tariff parameters. Let $(l_*, \cdots, u_*)$ denote the lower and upper bound on the each parameter and τ_* be the respective discretization step size. The space of search points $\mathcal{S}$ can be defined as the Cartesian product of the $\{l_{fix}, l_{fix} + \tau_{fix} \cdots, u_{fix}\} \times \{l_o, l_o + \tau_o, \cdots, u_o\} \times \{l_s, l_s + \tau_s, \cdots, u_s\} \times \{l_p, l_p + \tau_p, \cdots, u_p\} \times \{l_c, l_c + \tau_c, \cdots, u_c\}$. The cardinality of the search space $\mathcal{S}$ depends on each of the τ_* and is given as:

$$|\mathcal{S}| = \prod_* \left(\frac{u_* - l_*}{\tau_*}\right) \tag{4}$$

An arbitrary point s in the search space $\mathcal{S}$ is given as a five tuple representing each of the five search parameter α_* respectively. The optimization problem defined in the section 3 can be re formulated as a 5-dimensional grid search problem as follows.

PROBLEM 3.2. *Find one or more tuple $\{s_i, \delta\}$ where $s_i \in \mathcal{S}$ and $\delta \in \mathbb{R}$ such that the $abs(\hat{R}(\Delta_t) - R) \leq \delta$ for some sufficiently small δ and given time period Δ_t.*

The success of the search depends mainly on the two parameters (τ_*, δ). The discretization step τ_* defines the size of the overall search space. Larger values of τ_* would result in very few search points and may not yield a solution for the chosen δ. On the other hand, very small values would result in huge number of search points and the searching would become time consuming process.

Similarly a large value of δ would result in too many solutions which may not be of much interest as it is desired that $\hat{R} \approx R$. On the other side, very small value of δ would become too stringent and may not result in any solution.

Typically, randomized and adaptive search approaches works best in such situations, which refines these parameters in recursive fashion. These methods are well known in computer science literature [8] and further details can be found in [18].

In the following sections, we present a multi stage adaptive search approach which based on the outcome, refines these parameters in successive stages for better results.

3.2 Recursive grid search

The recursive approach initiates search with a moderate size grid and threshold on the whole space. Algorithm then recursively defines newer search regions around the grid points which satisfy the initial bounds. The newer regions are discretized with smaller steps and search is initiated with lower threshold in these new regions.

We now present the mathematical algorithm for the recursive grid search. For the sake of clarity, we describe the different function of the algorithm and then detail out the final algorithm using these functions.

Algorithm 1 `computeS()`, Computes search points

```
Read values of p, τ*, l*, u*, threshold δ
Initialize S = ∅
if p = NULL then
    Set upper and lower range for α* to default
else
    Set l* = (l* + r*) and u* = (u* − r*) for some r*
end if
for α_fix = l_fix to u_fix − 1: τ_fix do
    for α_o = l_o to u_o − 1: τ_o do
        for α_s = l_s to u_s − 1: τ_s do
            for α_p = l_p to u_p − 1: τ_p do
                for α_c = α_p to u_c − 1: τ_fix do
                    s = {α_fix, α_o, α_s, α_p, α_c}
                    S ← S ∪ s
                end for
            end for
        end for
    end for
end for
return S
```

The algorithm `computeS()` generates all the grid points to be searched either on entire space or around a point p. The values of r_* could be different for each parameter α_*.

Algorithm 2 `computeP()`, Computes solution points

```
Load consumption data E_o, E_s, E_p, E_c
Load the values of time period Δ_t, threshold δ
Load the set of search points S
while S ≠ ∅ do
    Choose a point s ∈ S
    Compute R̂(Δ_t) at s
    if |R̂(Δ_t) − R| ≤ δ then
        P ← P ∪ {s, δ}
    end if
    S ← S \ {s}
end while
return P
```

The algorithm 2 computes the solution to the search problem within the set of search points $\mathcal{S}$. The revenue $\hat{R}$ is computed using equations 2 and 3. The reference revenue R is an input parameter to the algorithm.

Algorithm 3 Recursive Grid Search

```
Read parameter δ, Set number of stages N_s
Initialize set of points P = {{p, δ} | p = NULL}
while N_s ≠ 0 do
    for Each p ∈ P do
        S_p=computeS(p, τ_*, l_*, u_*, r_*)
        P_p = computeP(S_p, δ, Δ_t)
    end for
    P ← ∪P_p
    update(δ, τ_*, r_*)
    N_s ← (N_s − 1)
end while
Sort P w.r.t δ
return {s, δ} ∈ P for smallest δ
```

The algorithm 3 computes the desired solution. The algorithm terminates after executing N_s stages. At each stage, the threshold δ is reduced to a lower value. New search grids are formed around all the points obtained in the previous stage ($\cup\mathcal{P}_p$) with smaller discretization steps τ_*. The function `update()`, simply scales down the values of δ, τ_*, r_* such that the range constraints on α_* are not violated.

Instead of having fixed number of stages N_s it is possible to describe the algorithm with dynamic number of stages. This can be achieved by simply replacing the loop **while** $N_s \neq 0$ **do** with **while** $\delta > c$ **do**. If a solution is found then the algorithm at least achieves $\delta = c$.

There is no way one can guarantee a solution, however with a suitable choice of δ and τ_* an acceptable solution can be obtained. In the next section, we present our GPU implementation of the above algorithm. The implementation is specific to Nvidia GPUs and carried out in `CUDA C` [12].

4. IMPLEMENTATION DETAILS

In principle, the recursive grid search algorithm described in section 3 is embarrassingly parallel. Each point in the search space can be evaluated independently. The modern GPUs, with large number of cores, are best suited for such problems. The Kepler series GPUs from Nvidia offers several highly desired features such as HyperQ and Dynamic parallelism [13].

The Dynamic parallelism enables launching kernels from a already executing kernel based on some conditions. This feature is best suited for the above presented grid search algorithm. To exploit the dynamic parallelism feature, the above described algorithm was implemented in slightly different manner.

The computations begins with a kernel for some moderate size δ and τ_* and instead of the search points, the search ranges were distributed equally among all the threads. Each thread computes and evaluates the points in its range and if the point is found to satisfy the condition, the thread creates a new search grid around that point with finer τ_* and launches next stage search kernel with smaller δ.

In this arrangement, at any instance, different threads may execute different stages of the computation. If no solution is found for the current stage, the thread returns to the next search point from the previous stage. This can be best seen as depth first search, where the algorithm gives preference to the next stage before finishing the existing stage. The following steps broadly describe the GPU implementation.

1. Initialize the threshold δ and step size τ_* for each parameter. Also initialize number of stages N_s.
2. Let N_{t_0} be the number of initial threads. Divide each range $(l_*, \ldots, u_*)$ into N_{t_0} equal parts.
3. Initiate a CUDA kernel with N_{t_0} threads. Each thread computes $|\hat{R} - R|$ for all the points in its range.
4. If $|\hat{R} - R| \leq \delta$ for some point p point then the corresponding thread does following:
   (a) Update N_s, δ, τ_* and r_* and compute new l_* and u_*. If $N_s \neq 0$ then
   (b) Let N_{t_1} be the number of threads. Divide each range $(l_*, \ldots, u_*)$ into N_{t_1} equal parts.
   (c) Initiate a CUDA kernel with N_{t_1} threads. Each thread computes $|\hat{R} - R|$ for all the points in its range.
   (d) If $|\hat{R} - R| \leq \delta$ for some point p point then goto step 4a and launch next stage kernel. Else goto step 3 and resume computation with next point.

The performance of the above implementation in terms of speed and the accuracy is presented in later section.

To compute the optimal tariff parameters α_* either by solving the optimization problem or by grid search, it is required to compute the energy consumptions E_* for the respective periods. In the following section, we present our approach for processing the annual consumption data and computation of consumption pattern.

5. DATA PROCESSING

The figure 1 shows the representative consumption pattern. To identify various demand periods and the respective consumption, it is required to know the actual consumption pattern of the entire consumer base on the 24 hour cycle from the available data. In this section we discuss

simple algorithms and their implementation for computing consumptions for the different demand periods.

The metered readings were available for 30 minutes interval, which translates to 48 readings per day per consumer. The annual data for nearly 0.3 million consumers was provided in 12 CSV files (one for each month) and each file contained data for consumers in unsorted fashion.

The task of computing annual consumption for the entire consumer base could be achieved by simply adding all the rows. However it was also required to calculate per consumer revenue under new and old tariff. This task requires sorting data with respect to consumers IDs, which was carried out in parallel due to large volume of data.

Following steps were carried out to sort and validate the required data. All the steps were implemented using `MPI` [15] and `C++` with Standard Template Library (`STL`) [19].

1. A typical single day record contains one row with consumer ID, date, category of consumer, region and 48 consumption readings for that particular date. Input data (.CSV files) is equally partitioned among n ranks. Each rank then sorts the input data using consumer number as key and all records for that customer is accumulated into a unique file `consumerID.csv`.

2. The part files for each consumer were merged into single file. Since there were large number of files, this task was also equally divided among all ranks. This merging leads to new refined data base with one file per consumer containing yearly consumption record.

3. The per consumer data was again sorted as per dates. Each processed consumer file (`consumerID.csv`) contains 365 rows and each row containing 48 consumption readings.

Several data queries were implemented on the processed data. For the sake of brevity, we present one such algorithm in the following. To compute the critical peak consumption E_c, it is required to compute the 12 maximum consumptions days from the entire data. It can be seen that the last column in the data stores the sum of the 48 readings for the respective rows. Following algorithm was implemented to compute the 12 maximum consumption days.

The function `sumRow` simply sums all rows from consumer(i).csv file into a single row of 48 reading. The function `mergeSort12` merges data from two files and sorts it based on the entries in the last column, which represents the total consumption for that day or date. Depending on the available memory, the practical implementation of the above algorithm loads multiple files, rather than processing them one by one. The function `sort12` sorts list12 data from all the ranks in descending order.

To compute the consumption E_* for different demand periods, following steps were executed:

- All the outputs of `sumRow` (which are vectors of length 48) were added to get a single vector of 48 readings denoted as $e[i]$ where, $i = 0$ to 47. The vector $e[i]$ represents the annual consumption of the entire consumer base. The vector $e[i]$ is plotted in figure 2 on 24 hour scale. The graph depicts the actual consumption pattern.

- The consumption for each demand period E_* is computed by adding the indices of consumption vector e corresponding to each demand period.

Algorithm 4 Compute critical & per user consumption

Let there are r ranks and N_T total files. Each rank processes $n = \frac{N_T}{n}$ files
Each rank does the following
Initialize a file `list12.csv` with 12 rows with all the readings set to 0
for $i = 0, n-1$ **do**
  Load the file consumer(i).csv
  E_i = `sumRows`(consumer(i).csv)
  `mergerSort12`(list12.csv, consumer(i).csv)
  `update`(list12.csv)
  // The update function simply picks up the 12 maximum entries from the merged data and over writes into list12.csv
end for
if $rank = 0$ **then**
  Gather all list12.csv from other ranks into data.csv
  `sort12`(data.csv)
end if
return E_i, E_c as 12 maximum readings with dates

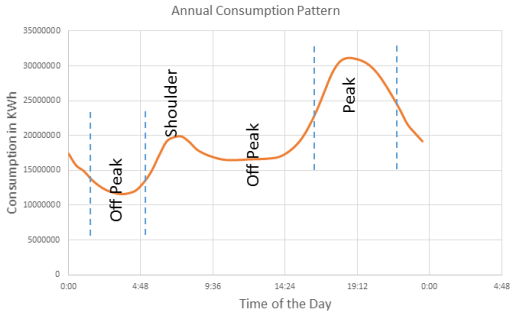

Figure 2: Computed Annual Consumption Pattern

The computation of E_c was described in algorithm 4. With the knowledge of all the E_* it is possible to execute the grid search and compute the optimal tariff parameters α_*.

6. EXPERIMENTAL SETUP AND RESULTS

We now describe our setup in terms of the type of hardware and software used and present results obtained for the tariff parameters computation. We begin with the summary on the type of hardware used.

6.1 Hardware and Software used

- Hardware
  - **Host** Intel Ivy Bridge 2.1 GHz, dual socket, 6 cores/socket, 16GB RAM.
  - **Device** Nvidia's Kepler $K20X$ GPU with 796 MHz 2496 cores, 5GB RAM.
  - Intel Xeon E5 2697 v3, 2.6 GHz, 14 cores/socket, Dual socket, 64GB RAM, 2 threads per core with HT.
- Software: Intel and GNU C / C++ Compilers, CUDA for GPU Programming, STL for generic algorithms

and classes, OpenMPI and OpenMP for parallel computations

6.2 Results

We begin by noting that all the computations were carried out with double precision and the numerical accuracy of the computations were not compromised in any hardware setup.

It is desired that the data processing and the grid search both should be completed in reasonable time if not in the real time. The most compute intensive step 1 and 2 in section 5 is to sort and create per user data. These steps are carried out in parallel. Apart from the compute time it is required that the absolute difference $|\hat{R} - R|$ i.e. δ should be very small. For the reference, the original revenue R is of order of \$400 million. In this case any $\delta \leq$ \$100 would suffice for the purpose. Following are the various performance results obtained.

- Data processing: The sorting of the data and creating the per user data took nearly 1 hour on Haswell system. To sort and create per user data (the step 1 in section 5) from the original data took 45 minutes. The step 2 took another 11 minutes. The original CSV files were partitioned among all the cores.
- Grid search: The grid search was executed on GPU system. The number of stages were set to 4. It took 382 milliseconds to complete the search with all 4 stages. The threshold δ was initially set to \$10 and in the final stage it was reduced to \$0.001. The total 4 solutions were obtained in the final stage. Nearly 2.5 billion grid points were searched on the GPU. The grid search was also implemented on Intel Haswell system on the similar lines of the GPU implementation but the details were omitted for the sake of brevity. The parallelization was carried out with `OpenMP Parallel For` pragma [14]. The number of stages were set to 4. The result obtained on GPU were reproduced on this system in 3.08 seconds.

7. CONCLUSIONS AND FUTURE SCOPE

The objective of the work was to effectively solve problem of computing desired tariff parameter rather than any qualitative analysis of the considered ToU model or to define a new ToU model or scheme.

The recursive grid search was effective and fast because lot of unproductive computations were cut down by the approach. Instead of searching with very fine grid resulting in too many points, it searches adaptively. Only potential regions were searched with details.

In this paper we presented how GPUs can be leveraged to address the tariff parameter optimization problem. A revenue loss of approximately \$19K was reduced to only $\leq$ \$0.001 with the results obtained by the grid search method. Also the algorithms work fast enough to take several runs with different settings.

We conclude this paper on the note that smart brute force methods and multicore systems can be used to effectively solve complex computational and data analysis problems.

8. REFERENCES

[1] H. S. A. Hatami and M. K. Sheikh-El-Eslami. A stochastic-based decision-making framework for an electricity retailer: Time-of-use pricing and electricityportfolio optimization. *IEEE Transactions On Power System*, 26 No 4:1808–1816, 2011.

[2] J. Andruszkiewicz. Time of use tariff design for domestic customers integrating the management goals of efficient energy purchase and delivery. In *In Proceedings of the 12-th International Conference on European Energy Market, EEM*.

[3] D. S. K. B. D. Pitt. Application of data mining techniques to load profiling. In *In Proceedings of the 21-st International Conference on Power Industry Computer Applications, PICA*, pages 131–136, 1999.

[4] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2009.

[5] D. W. Caves and L. Christensen. Econometric Analysis of Residential Time-Of-Use Electricity Pricing Experiments. *The Journal of Econometrics*, 14:287–307, 1980.

[6] E. Celebi and J. D. Fuller. Time-of-use pricing in electricitymarkets under different market structures. *IEEE Transactions On Power System*, 27 No 3:1170–1181, 2012.

[7] S. I. G. F. Gerbec D., Gasperic S. An approach to customers daily load profile determination. In *In Proceedings of the IEEE Power Engineering Society Summer Meeting*, pages 587–591, 2002.

[8] J. Hart and A. Shogan. Semi-greedy heuristics: An empirical study. *Operations Research Letters*, 6:107–114, 1987.

[9] S. Kauser and M. Shaikh. Time of use pricing - india, a case study. In *In Proceedings of the International Conference on Power Systems, ICPS '09*.

[10] R. Li and et al. The effect of a mandatory time-of-use pricing reform on residential electricity use. *Applied Energy, Elsevier*, 162(15):1530–1536, January 2016.

[11] R. S. J. P. Mutanen A, Ruska M. Customer Classification and Load Profiling Method for Distribution Systems. *IEEE Transactions On Power Delivery*, 26 No 3:1755–1763, 2011.

[12] NVIDIA. Cuda. https://developer.nvidia.com/about-cuda/.

[13] NVIDIA. Dynamic parallelism. http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-dynamic-parallelism.

[14] OpenMP. Openmp library. http://openmp.org/.

[15] OpenMPI. Open message passing interface. http://www.openmpi.org/.

[16] G. T. P. Yang and M. Nehorai. A Game-Theoretic Approach for Optimal Time-of-Use Electricity Pricing. *IEEE Transactions On Power Systems*, 28 No 2, 2013.

[17] P. G. Panapakidis IP. Alexiadis M.C. An approach to customers daily load profile determination. In *In Proceedings of the IEEE Power Engineering Society Summer Meeting*, pages 587–591, 2002.

[18] L. Pitsoulis and M. Resende. *Greedy randomized adaptive search procedures*. Oxford University Press, 2002.

[19] STL. Standard template library. https://isocpp.org/.