ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Τεχνικές μηχανικής μάθησης για διαχείριση ισχύος και πόρων σε πολυπύρηνα συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ του

Αναστασάκη Μιχαήλ-Ανδρέα

Επιβλέπων

Δημήτριος Σούντρης, Αναπληρωτής Καθηγητής

Athens, March 2015

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Τεχνικές μηχανικής μάθησης για διαχείριση ισχύος και πόρων σε πολυπύρηνα συστήματα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ του

Αναστασάκη Μιχαήλ-Ανδρέα

Επιβλέπων

Δημήτριος Σούντρης, Αναπληρωτής Καθηγητής

Εγκρίθηκε από την τριμελή επιτροπή την ημερομηνία εξέτασης

....	....	....
Δημήτριος Ι. Σούντρης	Νεκτάριος Κοζύρης	Κιαμάλ Πεκμεστζή
Αναπληρωτής Καθηγητής	Καθηγητής	Καθηγητής

Athens, March 2015

Michail - Andreas Anastasakis

Electrical and Computer Engineering Graduate of NTUA

Περίληψη

Η διαχείριση ενέργειας και η βελτιστοποίηση της απόδοσης ενός πολυ πύρηνου συστήματος αποτελεί μια πρόκληση για τα μοντέρνα υπολογιστικά συστήματα. Καθώς η τεχνολογία εξελίσσεται , ο αριθμός των πυρήνων πάνω σε ενα chip αποτελεί πολύ σημαντικό παράγοντα για την διαχείριση της ενέργειάς του . Η συνολική διαθέσιμη ισχύς είναι ο βασικός περιοριστικός παράγοντας σε ότι αφορά την βελτιστοποίηση της απόδοσης. Συνεπώς ένας διαχειριστής που λαμβάνει υπόψιν την συνολική δομή του ςηιπ απαιτείται ώστε το σύστημα να λειτουργεί στο μέγιστο επίπεδο. Μερικές μέθοδοι έχουν δοκιμάσει να προσεγγίσουν το θέμα αυτό χωρίς όμως να εμβαθύνουν σε συγκεκριμένα σημεία κλειδιά.

Ένα σημαντικό σημείο κλειδί είναι η επεκτασιμότητα αφού ο αριθμός των πυρήνων στο chip έχει μεγάλη επίδραση στην συνολική κατανάλωση ισχύος και την εφαρμοζόμενη πολιτική διαχείρισης. Σε αυτή την διπλωματική πολλές και διαφορετικές δομές ενός πολυ πύρηνου συστήματος έχουν αναλυθεί έτσι ώστε να ληφθεί υπόψιν η σημασία της επεκτασιμότητας.

Ένα επιπλέον σημαντικό σημείο κλειδί είναι η αξιοποίηση της γνώσης για την μελοντική κατάσταση του συστήματος έτσι ώστε να προβούμε σε αλλαγές που θα βελτίωσουν την κατάσταση αυτή. Έχοντας την δυνατότητα να προβλέψουμε την μελλοντική κατάσταση του πυρήνα διευρύνονται οι διαθέσιμες DVFS επιλογές μας έτσι ώστε να μεγιστοποιήσουμε την απόδοση του συστήματος πάντα κάτω απο συγκεκριμένους περιορισμούς σε ότι αφορά την ενέργεια. Η διπλωματική αυτή προσεγγίζει το πρόβλημα μέσω απο μεθόδους μηχανικής μάθησης βασιζόμενη σε δεδομένα που έχουν παρθεί από τα PARSEC-Suite benchmarks καθώς και την ανάλυση τους απο το MATLAB

Λέξεις κλειδιά Μηχανικές μέθοδοι μάθησης , προσομοιωτής Sniper , Δυναμική διαχείριση ισχύος , Δυναμική αλλαγή συχνότητας και τάσης , Πολυπύρηνο σύστημα

Abstract

The tasks of power management and performance optimization on a multi-core system has been a challenge for modern computer systems. As technology scales , the number of cores on a chip has become an important factor for the chip power management. A core's power budget is the main limitation for it's performance optimization so a manager that takes into consideration the chip's overall configuration is required in order to perform at a top level. The combination of chip's core configuration and therefore an adjusting power budget is the key for the optimal manager. A few methods have tried to approach this matter but not in a deep level on certain key points.

One important key point is scalability , since the number of the chip's cores has a big impact on the total power consumption and the applied managing policies. In this diploma thesis many different configurations of a multi-core system have been analyzed so as to take into consideration the important matter of scalability and the impact of the core's number on a chip.

Another important key is taking into consideration the future state of the chip in order to be able to perfom in advance changes that will improve it's state. Being able to predict the future power constraints open different DVFS options in order to maximize the performance under that constraint.This diploma approaches this matter by defining and implementing machine learning functions based on data of the PARSEC-Suite benchmarks and the analysis through MATLAB.

The purpose of this diploma thesis is to analyze the behavior of a multi-core system under different power constraints and with the acquired knowledge to create a DVFS manager that with different applied policies can maximize the power savings of the chip and maximize it's throughput. Different tools have been used that provide this required knowledge and will be thoroughly explained in next section...

Keywords Dynamic Voltage Frequency Setting , Machine Learning , Sniper Simulator , Dynamical Power Management , Multi Core System

Contents

List of Figures

Dedicated to my friends and family.

Introduction

The main problem this diploma thesis tries to solve is the effective resource management on a multi core system. This means that while having constraints such as a specific power budget the resources can be allocated properly. The way this problem is approached is by creating a manager system that will monitor the behavior of the cores and will change dynamically the V-F settings of the cores. An important aspect of this approach is that the decision made by the manager will be the result of the predicted behavior of the cores.
To further explain a predictive system is created that after a certain analysis can provide the expected behavior of the chip's cores. The manager will communicate with this system and based on the received values , will perform certain actions in order to maximize the performance and have an efficient resource allocation. Two types of manager have been developed and will be explained in the next section , a *Individual Manager* and a *Global Manager* . The policies that these two managers follow and the advantages and misadvantages that they present will be fully explained.
Furthermore another important aspect of this approach is that the available resources , in our case the power budget , depends and accordingly adjust to the number of the active cores on the chip. This makes our approach more realistic and exhibits interesting results about the importance and the effect of the active cores on the power budget.

The diploma thesis is devided into 3 Chapters.

- Chapter 1 Framework. In this chapter the structure of the framework that was setup up is explained. The main components and the communication between them are presented. We explain the implementation of the Global Manager / Individual Manager , the predictive model and provide as well a theoretic background of the machine learning methods that were analysed.

- Chapter 2 DVFS Strategy. In this chapter we analyze the strategy and the policies that were followed by the managers.First we present the data that were used for this diploma thesis , which were used for the machine learning training as well.Also the implementation and the details around the Individual and Global Managers are presented.

- Chapter 3 Experimental Results. In this chapter we explain and characterize the results of our implementations.First an analysis of the used benchmark is made and is followed by the results of the manager implementation on these benchmarks.Finally there is a cross validation sector where the implemented system is used on unknown and new benchmarks followed by relative to them figures.

Related Work

The problem has been approached with other methods in the past which have been studied and are explained below. [5] has presented a power control system that controls the power of the chip by partinioning the power budget to cores per application and then further allocate it based on the core criticality.The power control system that can manipulate the power budget by adjusting the frequencies of the cores on the chip is a PI controller.F [2] has approached the problem based on the thermal constrains on a chip. It proposes a model that takes into consideration each core budget and temperature in order to produce the optimal setting of active corenumber and V/F level. [4] has proposed a global manager that controls the chip in order to increase Power Savings in the cost of performance. This paper proposes many different policies but lacks to analyze the behavior of the manager on a high number of cores.

Chapter 1

Framework

Framework Setup

In this chapter the overall framework that was implemented for this diploma thesis will be explained. The basic components are described individually and the details of the strategy that was followed are explained in the next chapter.

1.1 Multi-Core System

A 32-core system has been simulated for the purpose of this diploma . The simulation was performed using the Sniper Simulator. The system architecture that has been used is the *Intel Processor Nehalem* architecture and specifically the *Xeon X5550 Gainestown* processor. The details of this processor are the following :

- 32 KB L1 Instruction Cache per core.
- 32 KB L1 Data Cache per core.
- 256 KB L2 Cache per core.
- 8192 KB L3 Cache Shared per 4 cores.
- DVFS Transition latency of 2000 ns

A overall structure of the chip is displayed below.

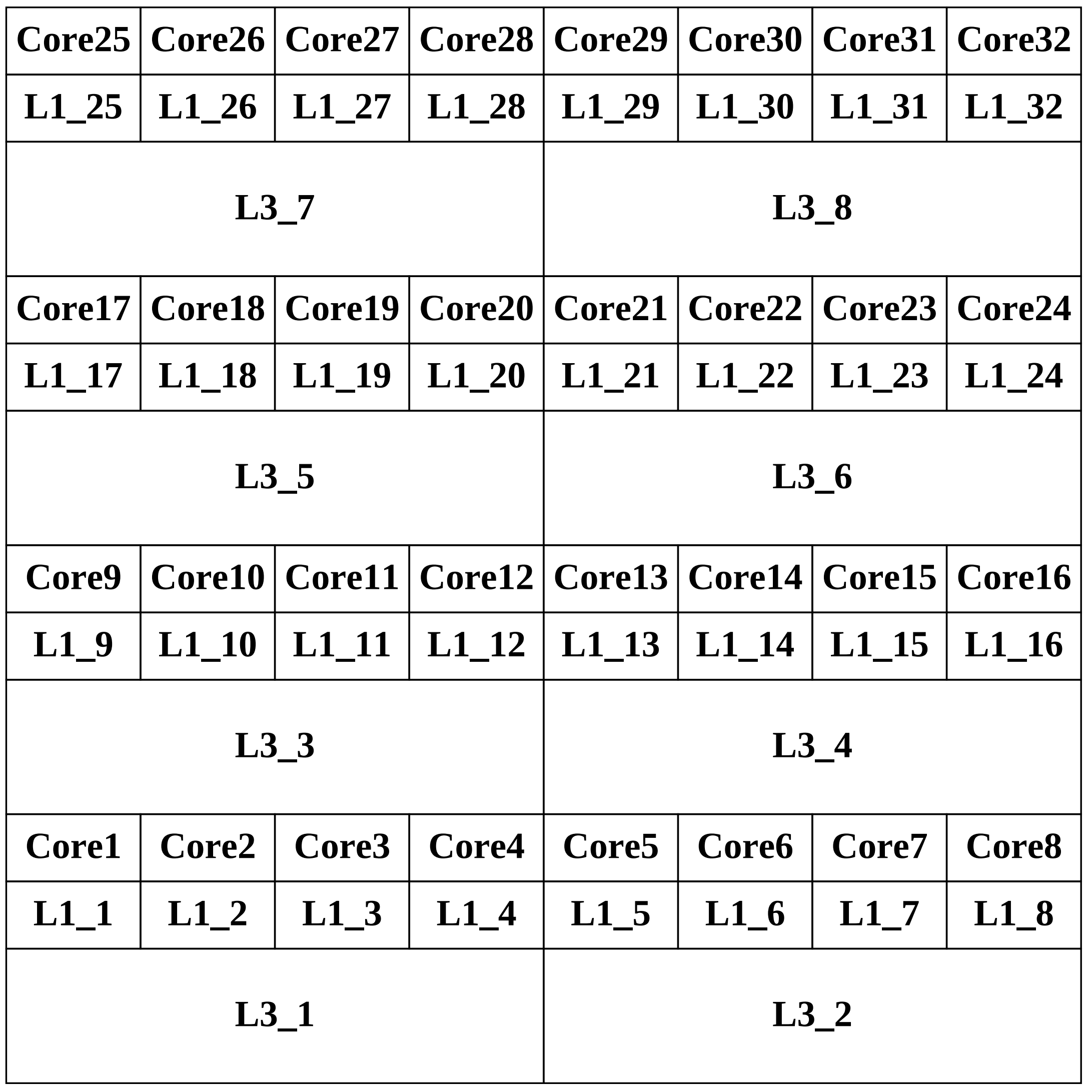

Core25 Core26 Core27 Core28 Core29 Core30 Core31 Core32
L1_25 L1_26 L1_27 L1_28 L1_29 L1_30 L1_31 L1_32
L3_7 L3_8
Core17 Core18 Core19 Core20 Core21 Core22 Core23 Core24
L1_17 L1_18 L1_19 L1_20 L1_21 L1_22 L1_23 L1_24
L3_5 L3_6
Core9 Core10 Core11 Core12 Core13 Core14 Core15 Core16
L1_9 L1_10 L1_11 L1_12 L1_13 L1_14 L1_15 L1_16
L3_3 L3_4
Core1 Core2 Core3 Core4 Core5 Core6 Core7 Core8
L1_1 L1_2 L1_3 L1_4 L1_5 L1_6 L1_7 L1_8
L3_1 L3_2

Figure 1.1: Nehalem Floorplan.

1.2 Global Manager / Individual Manager

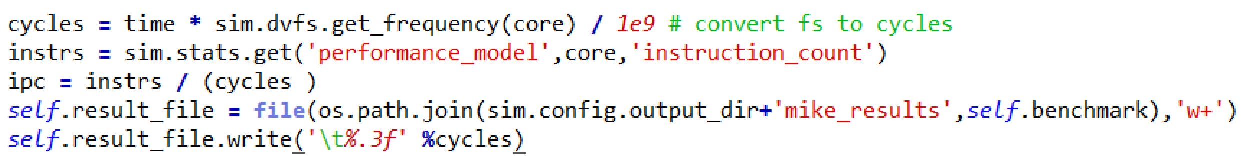
A system that will be called every interval and will be provided with the all the core states by the Sniper Simulator.
Core State - as core state I define the following key metrics of every core on the chip:

- Core IPC.
- Core Accesses (L1 Data Cache Accesses + L1 Instruction Cache Accesses + L2 Cache Accesses)
- Core Misses (L1 Data Cache Misses + L1 Instruction Cache Misses + L2 Cache Misses)
- Core Power (In that specific time interval and measured in Watt)

The Manager is implemented in Python and integrated with Sniper's Python API. The manager gathers the required information from Sniper and performs the DVFS actions every interval that he is triggered during the simulation. In order to get the data from the Sniper Simulator , the Python API is used and the data are stored in a file.

- *sim.stats.get(general_ category , core number ,specific_ metric)* - This calls receives as arguments the general category of the requested value metric , the number of the core and the name of the specific metric.
- *sim.dvfs.get_frequency(core number)* - This call receives as argment the number of the core on the chip and utilizes the DVFS implementation of Sniper and returns the value of the applied frequency on that core.

Below we can see how the IPC of a core is computed and stored in the result file.



```
cycles = time * sim.dvfs.get_frequency(core) / 1e9 # convert fs to cycles
instrs = sim.stats.get('performance_model',core,'instruction_count')
ipc = instrs / (cycles )
self.result_file = file(os.path.join(sim.config.output_dir+'mike_results',self.benchmark),'w+')
self.result_file.write('\t%.3f' %cycles)
```

FIGURE 1.2: Compute and Save Results

1.3 Machine Learning Methods / Predictive Model

1.3.1 Machine Learning Methods

In this sector the concept of the predictive model is described and the results of the 3 different predictive methods that were evaluated are presented. All methods have been evaluated and analyzed using MATLAB.
In order to maximize the efficiency of the system I decided to create an individual predictive model for each one of the frequencies that are available through DVFS. This will make the modeling more precise and accurate on the predicted values.A predictor will receive the four metrics of each core state and predict these four metric in the next interval.
The evaluation of the predictive methods will be made based on the `Root mean square Error ( RMSE)` whose definition is provided below .

$$RMSD(\hat{y}) = \sqrt{\hat{y}} = \sqrt{E(\hat{y} - y)^2}] \tag{1.1}$$

, where y is the original value and $\hat{y}$ the predicted value.
The 3 machine learning methods that were analyzed are :

- Linear Regression
  In linear regression, data are modeled using linear predictor functions .A linear predictor function is a linear function (linear combination) of a set of coefficients and explanatory variables whose value is used to predict the outcome of a dependent variable .

  $$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n,$$

  FIGURE 1.3: Linear Regression.

  In this case the variable y is the variable whose value we want to predict and everytime is one of the four core state metrics in the next interval.
  The variables x are the four core state metrics in the current interval.
  The variables b are the coefficients that are created and adjusted during training , using the gathered data from the simulations.

- Regression Decision Tree
  A Decision Tree is a set of leafs (nodes) that given certain input values will predict the value of a target value. Each internal node (branch) corresponds to one of the input variables and a comparison that is applied.

Pending on the values of the input variables a certain path is followed on the tree up until a final node (leaf) is reached. In Figure 2.3 we can see a simplified vesion of a Regression Tree that predicts the power of a core. In the branches x1 is the current IPC of the core , x2 is the current core cache accesses , x3 is the current core cache misses and x4 is the current core power. The values on the leafs are the core power that we predict the core will have in the next interval. As we can see the branches perfom comparisons of the core metrics to certain values in order to conclude to a predict value.

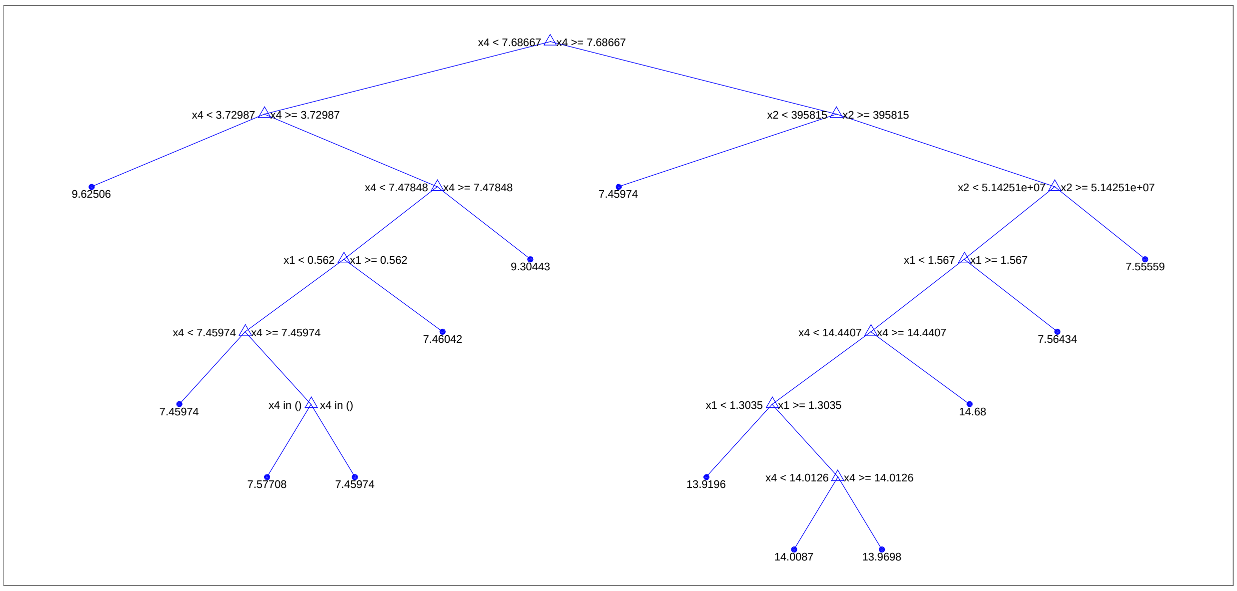


FIGURE 1.4: Regression Tree.

- Tree Bagger

  Tree Bagger is an ensemble(forest) of a number of decision trees and in our case Regression Decision Trees.

  Provided with a training set X = x1, . . . , xn with responses Y = y1 the tree bagger repeatedly selects a random sample with replacement of the training set and fits trees to these samples. So we choose a subset of n samples from the set and train a decision tree using these X_n samples with their Y_n responses. After that we follow the same procedure with more samples in order to train all our trees.

  Sampling with replacement means that after we have chosen a part of the training set and train a tree with it, we return this part back to the total of the training set. In our case the number of Regression Decision Trees in the forest is 50 in order to maximize the performance.

  When the Tree Bagger is provided with a row of input data , these data are provided to all it's trees. Each regression tree responds with their predicted value and the Tree Bagger takes a democratic (non weighted) average vote from all the trees.

The result of that vote is provided by the following formula :

$$\hat{f} = \frac{1}{N}\sum_{1}^{N} \hat{f}_b(x) \tag{1.2}$$

, where $\hat{f}_b(x)$ is the function that provided the result of a single tree , N is the total number of trees and $\hat{f}$ is the result of all the votes provided by the trees.

- Neural Network
  A neural network is a multi layer system with weighted nodes. In our case we had a feedforward neural network with 1 hidden layers with 10 neurons . At each layer the sum of the products of the weights and the inputs is calculated in order to predict the target value and the weights are adjusted so as to meanimize the mean square error between the target and the output of the neural network.

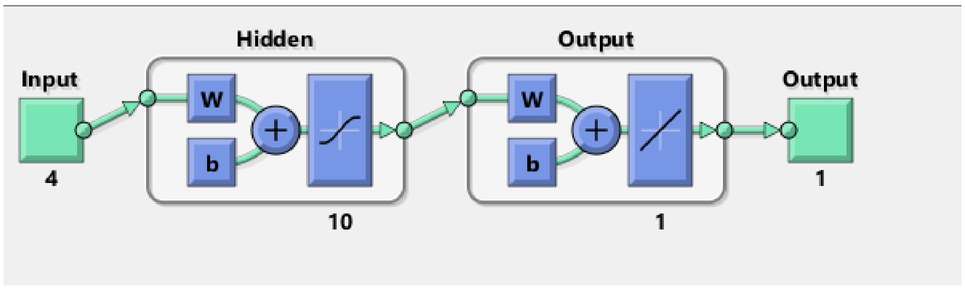


FIGURE 1.5: Neural network.

The data that were gather from the analysis were grouped in one matrix per frequency in the following structure. Each row represents the core state at a certain interval. For all the machine learning methods we used the next core state at the target values in order to train the system on the core behavior. In that way the model at each frequency will be able to predict the expected behavior (four core metric values) in the next interval.

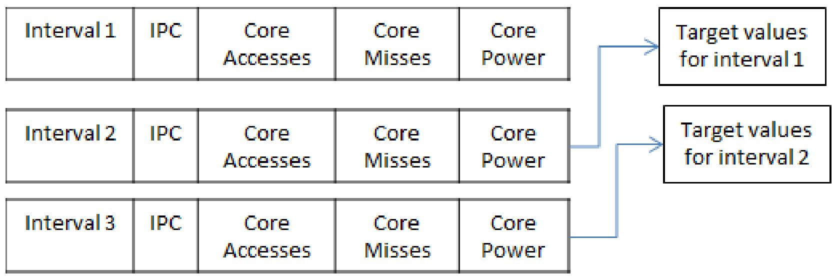


FIGURE 1.6: Data Structure Start.

Below we can see the results of the 3 predictive methods on the four core state metrics. As I mentioned before each frequency will have it's own predictor in order to maximize the performance so there is one table corresponding to every DVFS frequency level.

1 GHz

Method / Metric RMSE	IPC	Core Accesses	Core Misses	Core Energy (Watt)
Linear Prediction	0.0610	215025	6638	0.454
Regression Decision Tree	0.0499	300149	6304	0.284
Tree Bagger	0.0506	163033	4243	0.291
Neural Network	0.0540	312569	37148	0.393

1.4 GHz

Method / Metric RMSE	IPC	Core Accesses	Core Misses	Core Energy (Watt)
Linear Prediction	0.073	297890	9008	0.95938
Regression Decision Tree	0.064	314077	7188	0.579
Tree Bagger	0.059	218945	5499	0.593
Neural Network	0.064	30232432	1337908	0.870

1.8 GHz

Method / Metric RMSE	IPC	Core Accesses	Core Misses	Core Energy (Watt)
Linear Prediction	0.083	404095	11551	3.031
Regression Decision Tree	0.055	318115	7583	1.479
Tree Bagger	0.056	254763	5562	1.534
Neural Network	0.069	29051678	833066	3.42

2.1 GHz

Method / Metric RMSE	IPC	Core Accesses	Core Misses	Core Energy (Watt)
Linear Prediction	0.087	473089	13309	4.580
Regression Decision Tree	0.057	328961	8018	1.854
Tree Bagger	0.059	281094	6051	1.973
Neural Network	0.060	29732137	8625543	3.52

2.7 GHz

Method / Metric RMSE	IPC	Core Accesses	Core Misses	Core Energy (Watt)
Linear Prediction	0.096	608153	16344	9.026
Regression Decision Tree	0.065	368014	10129	3.703
Tree Bagger	0.067	285246	7183	3.885
Neural Network	0.095	33911649	5186520	10.77

3.3 GHz

Method / Metric RMSE	IPC	Core Accesses	Core Misses	Core Energy (Watt)
Linear Prediction	0.103	738344	18981	15.763
Regression Decision Tree	0.067	411699	10298	5.858
Tree Bagger	0.070	366206	9501	6.297
Neural Network	0.097	36742346	6572670	8.400

From these tables we can see that the method of *Regression Decision Tree* is better when we want to predict the **IPC** and **Energy** of a core. On the other hand the method of *Tree Bagger* is better when we want to predict the **Accesses** and the **Misses** of a core. The numbers of the root mean square error of the acceses and misses are relatively small compared to the mean value of Accesses and Misses which is in the scale of 2 to 20 millions pending on the frequency level.The methods of Linear Prediction and Neural Network present an rmse higher than the other 2 methods for all metrics and different frequencies.

Below the figures of the 4 metrics that characterise each core are presented. In each figure the predicted value of a metric is plotted as a dashed red line and the actual value of the matrix in the next interval as a blue line.
In the first sub-plot of each group we can see the predicted **IPC** and the actual value of the **IPC** in the next interval.
In the second sub-plot we can see the metrics of **Core Acceses**.
In the third the metrics of **Core Misses**.
In the forth the metrics of **Core Power**.
The core configuration in these plots is **1.8 GHz and 4 active cores**.

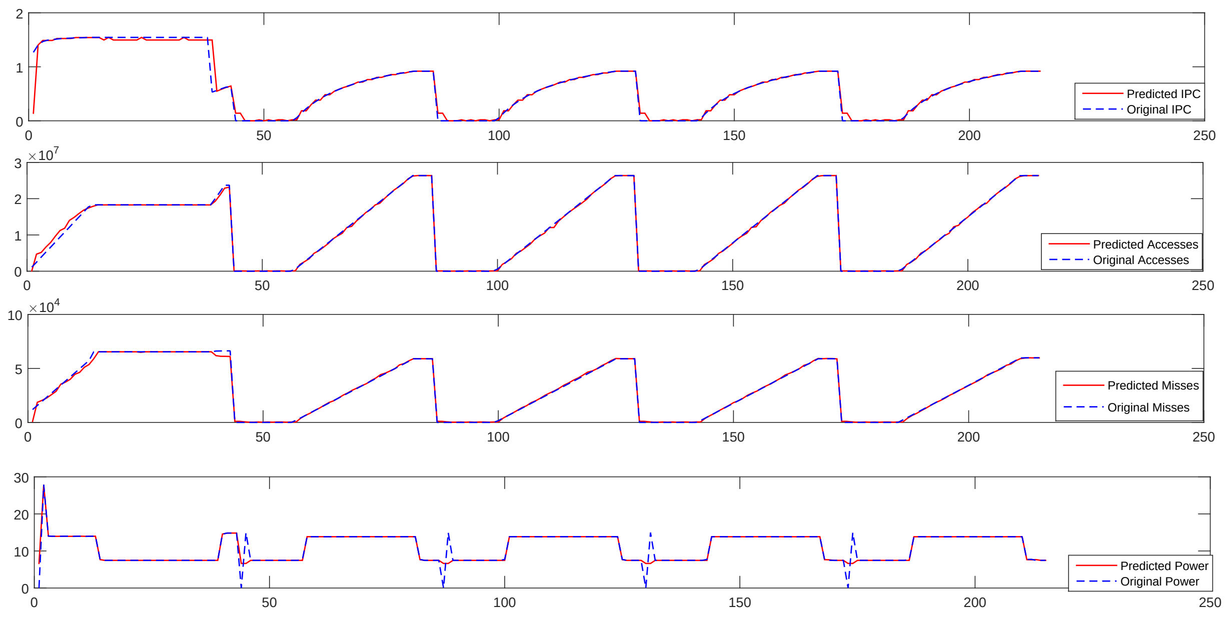


FIGURE 1.7: Blackscholes 1.8GHz 4 Cores

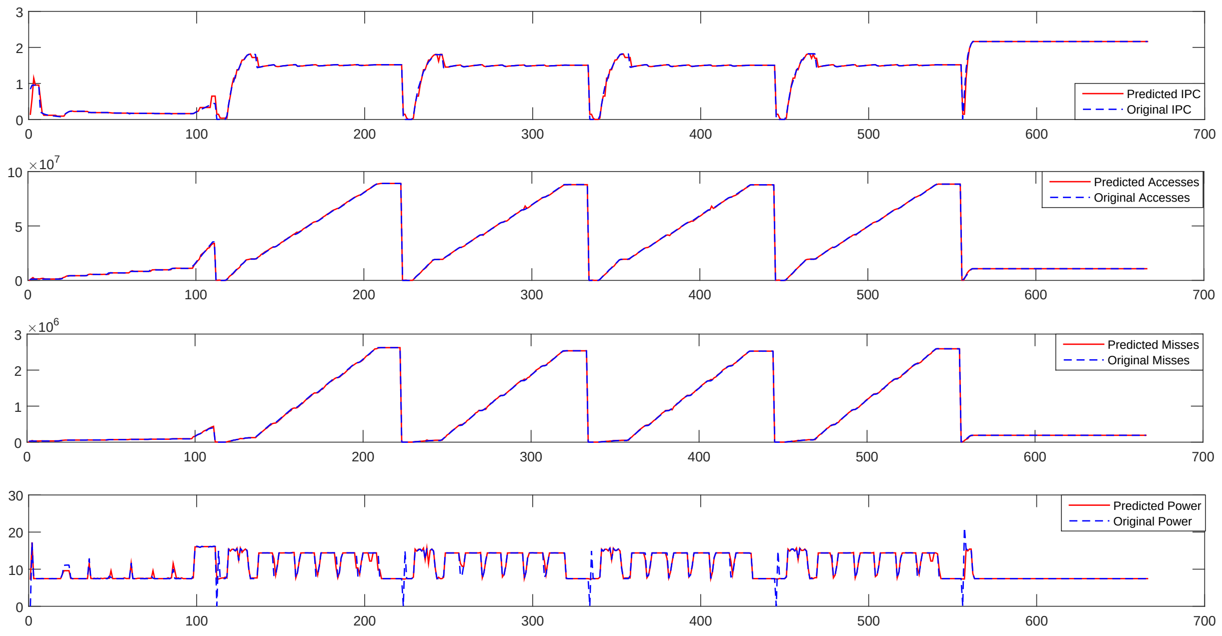


FIGURE 1.8: Bodytrack 1.8GHz 4 Cores.

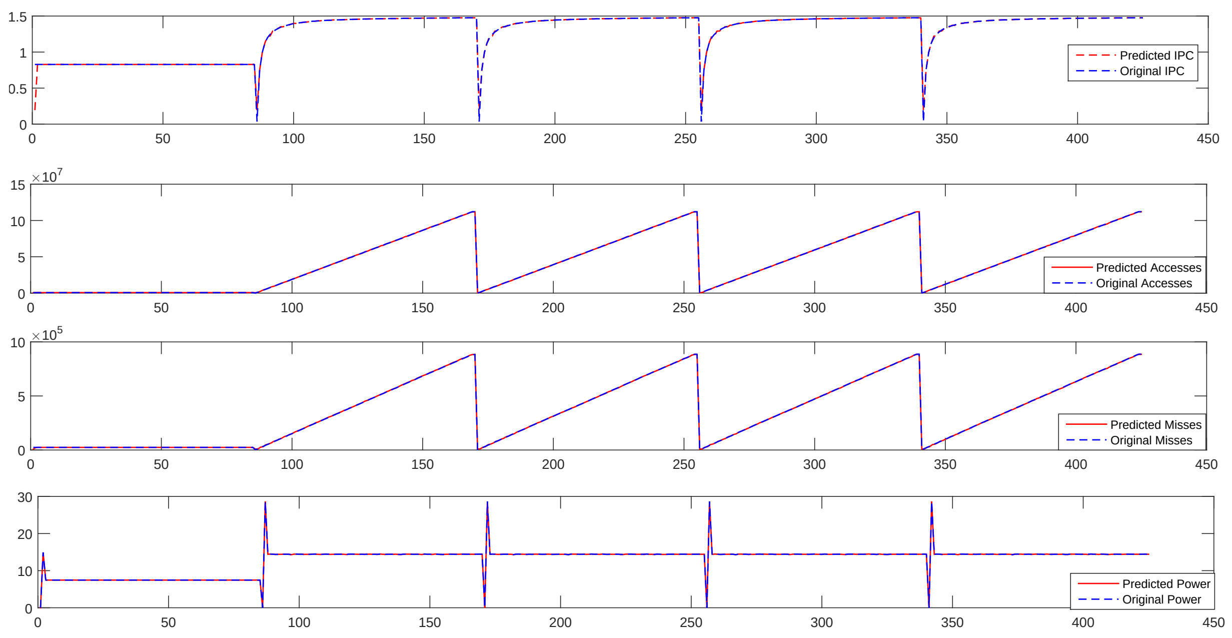


FIGURE 1.9: Swaptions 1.8GHz 4 Cores.

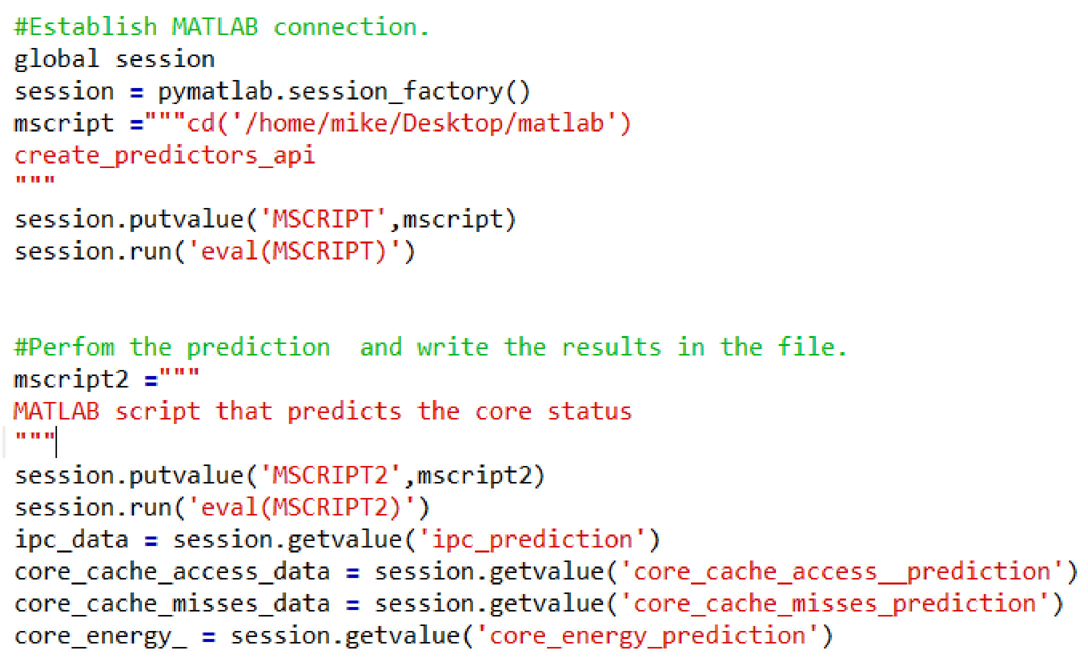
As we can see the predicted values of the four metrics are close to the original values on all 3 benchmarks. As a result when the predictor is called to give his estimation of the core state in the next interval , that estimation will be accurate.

1.3.2 Predictive Model

Predictive System - a system that will be called every interval during the execution of a benchmark by the **Global Manager** / **Individual Manager** and provided the state of each core on the chip at that interval the system can predict the state of those cores on the next one.
Since the Sniper's Python API was used and studied , the *Predictive system* was also created in Python. The connection between the predictive system and MATLAB was established using the PyMATLAB package. In the PyMATLAB implementation we have the following api calls and variables:

- *pymatlab.session_factory()* - sets up the MATLAB enviromenmt and returns a session that can receives scripts and parse them into MATLAB commands.
- *mmscript* - A block of text that includes the MATLAB commands we want to execute.

- *session.putvalue(MATLAB_ name , PYTHON_ name)* - transfers the variable from the Python interface into the MATLAB enviroment that has been created in this session.
- *session.run('command_ text')* - executes the command into the MATLAB enviroment. If there is no variable found with the specific name an error is returned.
- *session.getvalue(MATLAB_ name)* - returns the value of the variable from the MATLAB enviroment. This function gives the ability to retrieve new or modified data from the MATLAB environment.



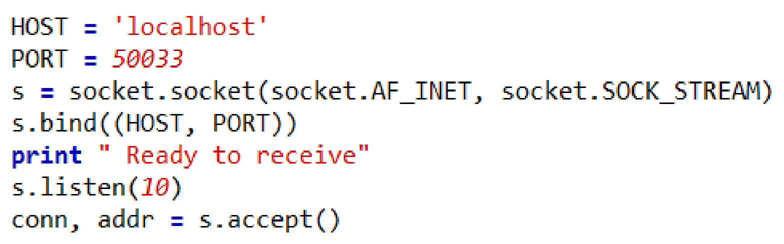
```
#Establish MATLAB connection.
global session
session = pymatlab.session_factory()
mscript ="""cd('/home/mike/Desktop/matlab')
create_predictors_api
"""
session.putvalue('MSCRIPT',mscript)
session.run('eval(MSCRIPT)')


#Perfom the prediction  and write the results in the file.
mscript2 ="""
MATLAB script that predicts the core status
"""
session.putvalue('MSCRIPT2',mscript2)
session.run('eval(MSCRIPT2)')
ipc_data = session.getvalue('ipc_prediction')
core_cache_access_data = session.getvalue('core_cache_access__prediction')
core_cache_misses_data = session.getvalue('core_cache_misses_prediction')
core_energy_ = session.getvalue('core_energy_prediction')
```

FIGURE 1.10: Connection with PyMATLAB

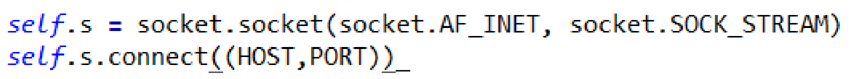
1.4 Manager Predictor Communication

At the start of the simulation the manager and the predictive system establish a socket connection in order to be able to communicate and secure that valid data are being used at every interval. The socket system that has been used ensures the fact that the Predictive System is blocked until the new data has are available.



```
HOST = 'localhost'
PORT = 50033
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((HOST, PORT))
print " Ready to receive"
s.listen(10)
conn, addr = s.accept()
```

FIGURE 1.11: Connection Predictor



```
self.s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
self.s.connect((HOST,PORT))
```

FIGURE 1.12: Connection Manager

The manager collects the status data of all the cores and stores them in the file *current_core_status.txt*. After that is complete , the manager informs the Predictive System that the data are ready by sending a message through the socket connection.

The whole structure of the framework and the mentioned components can be shown in the picture below:

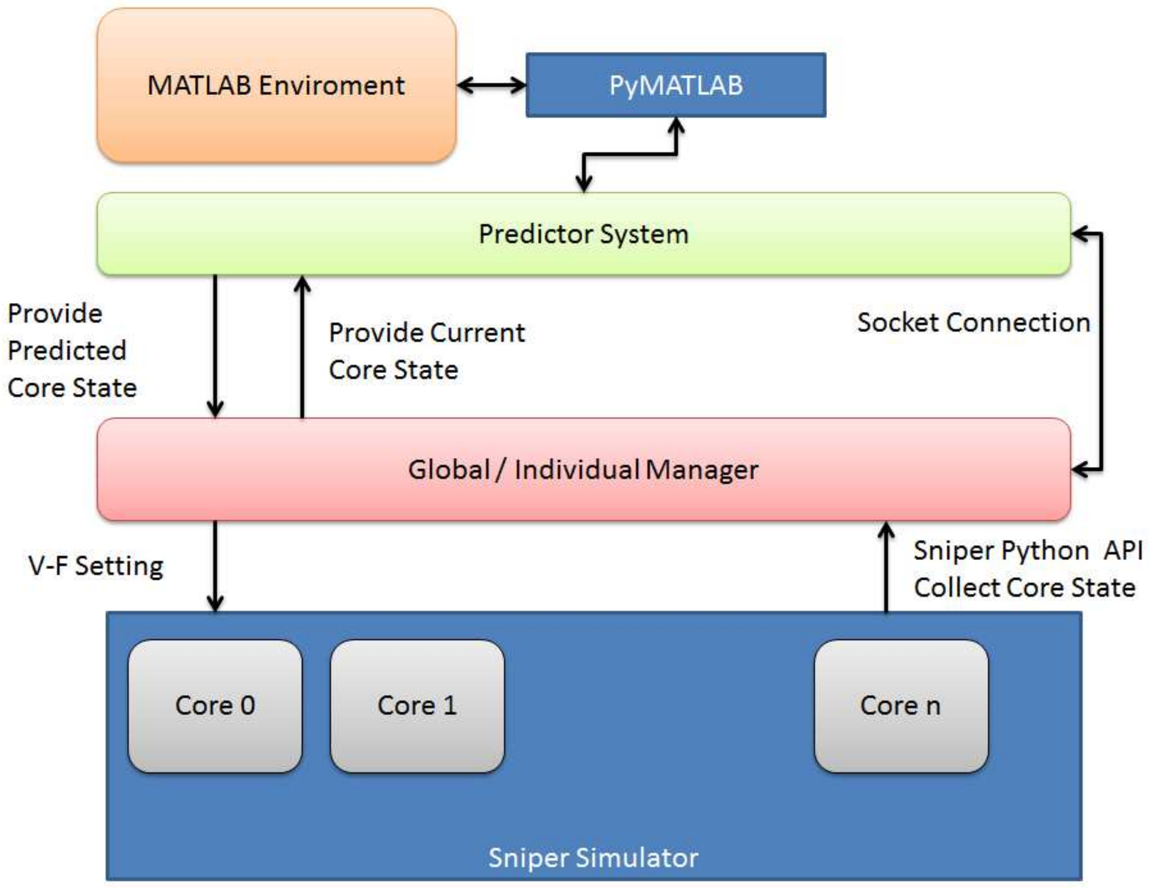

MATLAB Enviroment
PyMATLAB
Predictor System
Provide Predicted Core State
Provide Current Core State
Socket Connection
Global / Individual Manager
V-F Setting
Sniper Python API Collect Core State
Core 0
Core 1
Core n
Sniper Simulator

FIGURE 1.13: Framework.

1.5 Sniper Simulator

In order to simulate the multi-core system the Sniper Multi-Core Simulator has been used , which offers a plethora of benchmarks and allowed chip configurations. Sniper provides all the required information of the status of a chip during run time using precise tools that offer accurate and valid results.

Furthermore Sniper provides a DVFS implementation , with a sufficient scale of different Voltage-Frequency levels , which is essential to the functionality of the manager system that I have implemented in this diploma thesis.

In order to fully understand the functionalities and the abilities of Sniper the provided Sniper Python API was thoroughly studied. This Python API was further extended at certain points so as to include new possibities to the simulator that are essential to the DVFS Manager and the connection with the Predictor System that was implemented in this diploma thesis.

1.6 McPat

Intel's McPAT , an integrated power, area, and timing modeling framework for multicore and manycore architectures , has been used to obtain a precise power state of the chip. McPAT takes into consideration the configuration of the chip and provides the power state of the cores , L1 Data Caches , L1 Instruction Caches , L2 Caches McPAT has been integrated with Sniper during the Python API in order to be used in certain intervals.

1.7 ArchFP and HotSpot

ArchFP , a floorplaing tool which generates floorplan files, and *HotSpot* , an accurate and fast thermal model , have been studied for the Nehalem's system architecture but have not been fully implemented in the model.The floorplan that was produced to represent the chip architecture is shown below.

1.8 MATLAB

The *MATLAB* enviroment provides numerous functionalities and was used in order to implement the machine learning methods of the manager and the data analyses as well. MATLAB offers many different machine learning approaches which have been tested and evaluated in order to choose the best methon on a performance level. Furthermore MATLAB was used in order to analyze the benchmark data provided by SNIPER and the decision of key values for the different policies that have been applied. The provided illustrations in the next sectors of this diploma are generated by MATLAB.

1.9 Python

1.9.1 PyMATLAB

The *PyMATLAB* package lets Python users interface and communicate with MATLAB from Python in order to be able to send data from Python to MATLAB's workspace to be able to run Matlab function on the data . This package was essential to the implementation of the Predictor System since it gave the ability to fully exploit the MATLAB functionalities.

1.10 Framework Links

- **Sniper Simulator** - http://snipersim.org/w/The_Sniper_Multi-Core_Simulator
- **ArchFP** - http://lava.cs.virginia.edu/archfp/
- **HotSpot** - http://lava.cs.virginia.edu/HotSpot/index.htm
- **PyMATLAB packet** - https://pypi.python.org/pypi/pymatlab

Chapter 2

DVFS Strategy

2.1 Data Analysis

For the purpose of this diploma thesis the PARSEC-Suite benchmarks were used as data for the machine learning analysis and the evaluation. Each benchmark that was chosen represents a different application behavior in order to cover a bigger spectrum of applications and each benchmark is characterized by a multi-threading behavior.For each benchmark , the simulation was executed with a different combination of active cores (running threads) and DVFS level which are provided below. In the table below we can see the benchmarks :

Benchmark Name	**Benchmark Behavior**
Blackscholes	low-barrier
Bodytrack	high-barrier
Swaptions	no-barrier / high-lock workload

The different DVFS levels provided by Sniper are the following :

Frequency (GHz)	1	1.4	1.8	2.1	2.2	3.3
Voltage	0.8	0.9	1.1	1.2	1.4	1.6

The number of active cores for the execution of the benchmarks are :

Active Cores	2	4	8	16	32

From the plethora of combinations we can see that the gathered data characterize many different core behaviours and the created predictor will not be biased towards a specific pattern. In order to gather the required data , I created a Python data-gatherer program that was integrated with Sniper and called repeatedly at a certain interval time by Sniper.

2.2 Stategy Analysis

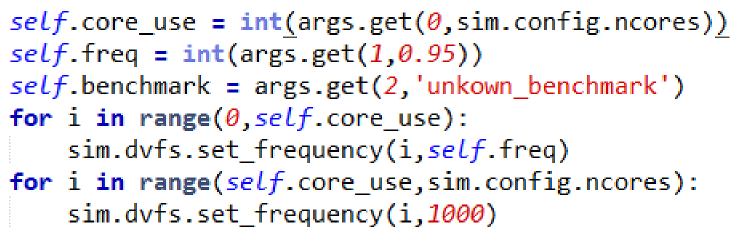
2.2.1 Normal Execution.

The normal execution is the base of all the implementations . Every benchmark simulation is executed on a 32 core chip as I previously mentioned. On the different configurations the number of cores varies between 2 , 4 , 8 , 16 , 32 cores and the rest of the 32 overall cores that are not used in each case are considered idle.A configuration that implies that the total number of cores and the number of active cores are equal , would not take into consideratin the power of idle cores and would not provide realistic results. During the normal execution a frequency is selected from the available DVFS levels and applied to all the running cores. The rest of the idle cores are set to 1GHz which is chosen as the idle core frequency since it's the lowest that can be applied because of the McPAT constrain.

This implementation provided no DVFS management during run time and is expected to overcome the power threshold of the chip. For the comparison with the other 2 strategies , this implementation will be used as the base for comparsion.
Simulations of a chip with a high number of active cores and a high level of frequency may not seem useful because they will not provide real comparable data with a power saving techniques but these simulations are are vital to the training of the predictive model since they will insert the information of over-budgeting and the model will be able to predict those core states and prevent them.
The implementation of the normal execution on the Sniper Simulator is based on Python API calls triggered by a custom script that is called at the start of the simulation.
In this implementation we use two provided variables by sniper

- *sim.dvfs.set_frequency(core number,applied frequency)* - This call receives as argument the number of the core on the chip and the frequency we want to apply
- *sim.config.ncores* - This variable provides the number of the total cores that we have simulated on the chip.

The created script makes the following calls at the start of the simulation.



```
self.core_use = int(args.get(0,sim.config.ncores))
self.freq = int(args.get(1,0.95))
self.benchmark = args.get(2,'unkown_benchmark')
for i in range(0,self.core_use):
    sim.dvfs.set_frequency(i,self.freq)
for i in range(self.core_use,sim.config.ncores):
    sim.dvfs.set_frequency(i,1000)
```

FIGURE 2.1: Normal Execution

Python Implementation of the managers

2.2.2 Individual Manager.

In this stategy an *Individual Manager* is implemented that will monitor the behavior of every core and decide whether the core should increase it's frequency level pending on the budget it has. In order to choose the individual power budget that each core will have to respect, the number of active cores has been taken into consideration. This means that as the number of active cores increases the individual power budget of each core will decrease correspondingly, this is a more realistic approach than setting a fixed constant as core power budget.This means that each core has the knowledge of how many cores are active but doesn't take into consideration the over-all chip budget. The correspondence between **Individual Power Budget** and **Core Number** was decided based on the TSocket paper [2] . As we can see when there number of active cores is low , 2 or 4 , the budget per core is high but as the number of active cores increases the power budget drops by a big margin.

An **Individual Manager** is monitoring every core during the simulation and records the cores state (the 4 metrics of the core) at every interval. The manager communicates with the predictive system as described above.

First Individual Manager checks that the core will not overcome it's individual budget in the next interval. If that comparison is satisfied that means that the core has still power that can be spent. If the comparison is not satisfied that means that the core is about to over come it's individual budget. In that case Individual Manager takes the decision to lower the frequency to the next frequency level on the DVFS table.

$$Power_{predicted} < Power_{budget} \tag{2.1}$$

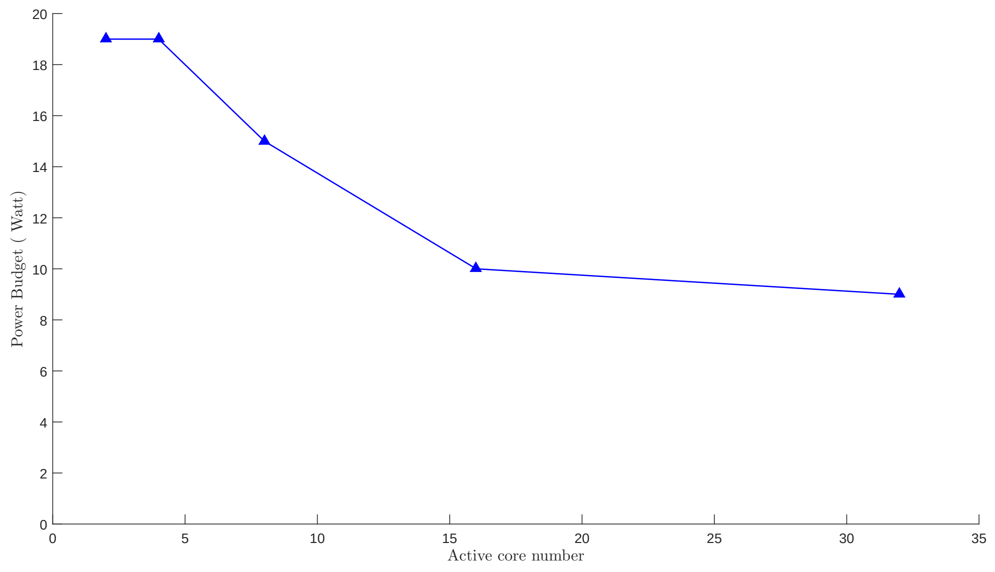


FIGURE 2.2: Individual Power Budget.

Second the Individual Manager further investigates the core state since it has room for improvement. This comparison provides the information about the value of a possible frequency increase and power consumption. If the *Predicted IPC* is greater than the current one that means that the core is going to increase it's throughput so an increase in the frequency is going to enhance that and as a result improve it's performance.

$$IPC_{current} < IPC_{predicted} \tag{2.2}$$

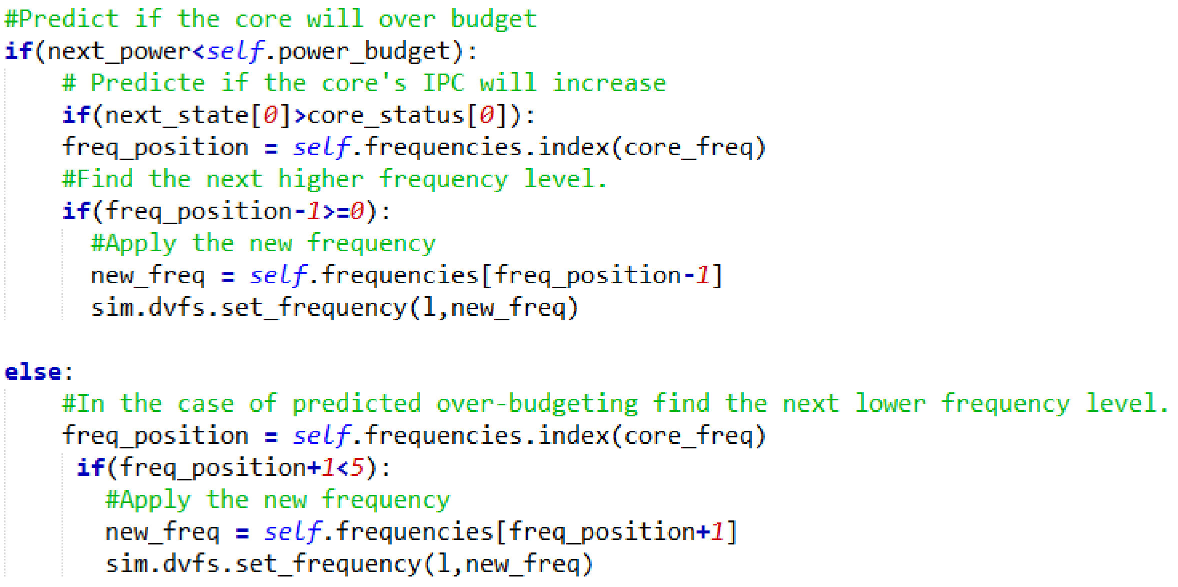
In addition the Individual Manager performs this check at the start of every interval :

$$Power_{current} < Power_{budget} \tag{2.3}$$

In that way we take into consideration the possibility of an error on the prediction which will result in over-budgeting

If both policies are satisfied then the Invidual manager increases the frequency level of the core to the next frequency level on the DVFS table. By following a policy that contains these steps the **Individual Manager** is able to control the power of a core under the budget and wisely increase it's frequency.
This model is expected to work well on an individual level but without taking into consideration the overall chip power budget , as the number of the active cores increase

then the phenomenon of over-budgeting on a chip level will appear. The results will be displayed in the Experimental section.



```
#Predict if the core will over budget
if(next_power<self.power_budget):
    # Predicte if the core's IPC will increase
    if(next_state[0]>core_status[0]):
    freq_position = self.frequencies.index(core_freq)
    #Find the next higher frequency level.
    if(freq_position-1>=0):
      #Apply the new frequency
      new_freq = self.frequencies[freq_position-1]
      sim.dvfs.set_frequency(l,new_freq)

else:
    #In the case of predicted over-budgeting find the next lower frequency level.
    freq_position = self.frequencies.index(core_freq)
     if(freq_position+1<5):
       #Apply the new frequency
       new_freq = self.frequencies[freq_position+1]
       sim.dvfs.set_frequency(l,new_freq)
```

Figure 2.3: Individual Manager Code.

2.2.3 Global Manager

In this strategy a *Global Manager* is implemented that will monitor the behavior of all the cores . This means that this manager will have information about the overall chip state and pending on the policy that he follows , can conclude to actions that will benefit the overall performance of the chip.

Global Manager Policy

The goal of the Global Manager will be to maximize the throughput of the chip. As a result the Manager will monitor all the active cores and evaluate which combination of frequency levels on the active cores will result in the maximum throughput while keeping the chip power consumption below the desired budget.
At every interval the Global Manager sets the idle cores to to 1GHz and perfoms power allocation to the active cores based on their criticality. Both the power allocation method and the core criticality will be defined below:

Core Criticality

A core is considered critical when an increase of it's throughput will result into an increase in the overall performance , compared to a similar increase on the other cores. It is essential to choose a criticality approach that performs it's task efficiently because

if chosen poorly this will result in a non optimal power allocation. With that in mind I define core criticaly as explained below :
4 approaches about core criticality have been followed in order to understand which one gives the best results and promotes a proper power allocation. It is worth mentioning all three approaches which are :

- *Core's Cache Acceses* in the next interval
- *IPC* of the core in the next interval
- the increase of *Cache Accesses* which is a result of $Accesses_{predicted} - Acceses_{current}$
- $\frac{IPC}{L2Misses}$

After various measurements on benchmarks I have come to the conclusion that a core's criticality is best characterized by the number of it's **Cache Accesses** on the next interval. In order to decide which approach is the most efficient , two main points were considered:

1. The chip total power at every interval remains under the power budget.
2. The throughput of the benchmark simulation , which is the total time of execution of the benchmark.

From a total of 15 simulations , which is the results of the 3 benchmarks over the 5 different active core number configurations , Cache Accesses gave the best result on a 13 of them. On the rest 2 configurations the best result occured when the core criticality was measured by the $\frac{IPC}{L2Misses}$ but not with a great difference from the Cache Accesses approach. The advantage of the Cache Accesses approach over the IPC one is that it can perform better on benchmarks with a barrier lock.

To further explain when a core of a multi -threaded application is running a thread that blocks at a certain point on the interval , then it will have a slighty high IPC but lack in Cache Accesses since it is not contributing in the benchmark execution. In that way the Core's Cache Acceses approach can identify this situation and re allocate a portion of such a core's budget to the rest of the active cores more efficiently than the IPC approach.
On applications that have no barrier and high workload both approaches provide an efficient power allocation with the Cache Acceses approach performing better on the throughput level.
In the following figures of the benchmark bodytrack we can see the that when the metric

of Acceses is used we have the best performance which is the lowest simulation execution time. Moreover in the cases where the $\frac{IPC}{L2Misses}$ performs faster , we can see that the chip power consumption exceeds the provided budget. As will be displayed in Chapter 3 with the experimental results the policy of keeping the chip power consumption below the budget is also respected.

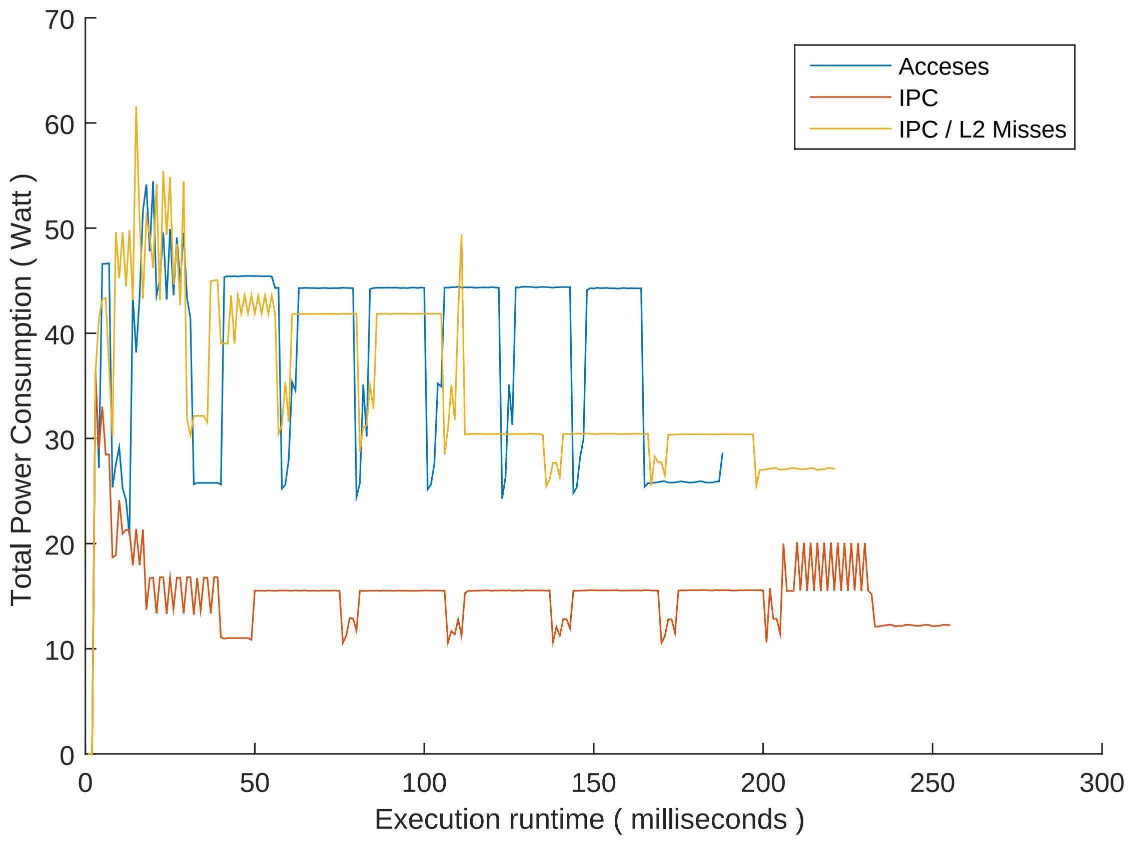


Figure 2.4: 2 Core criticality Factors.

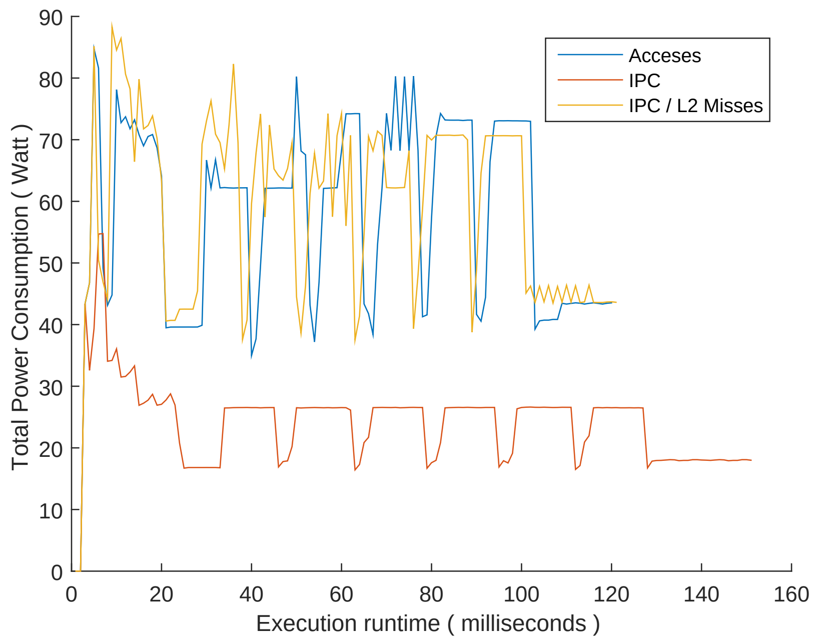


FIGURE 2.5: 4 Core criticality Factors.

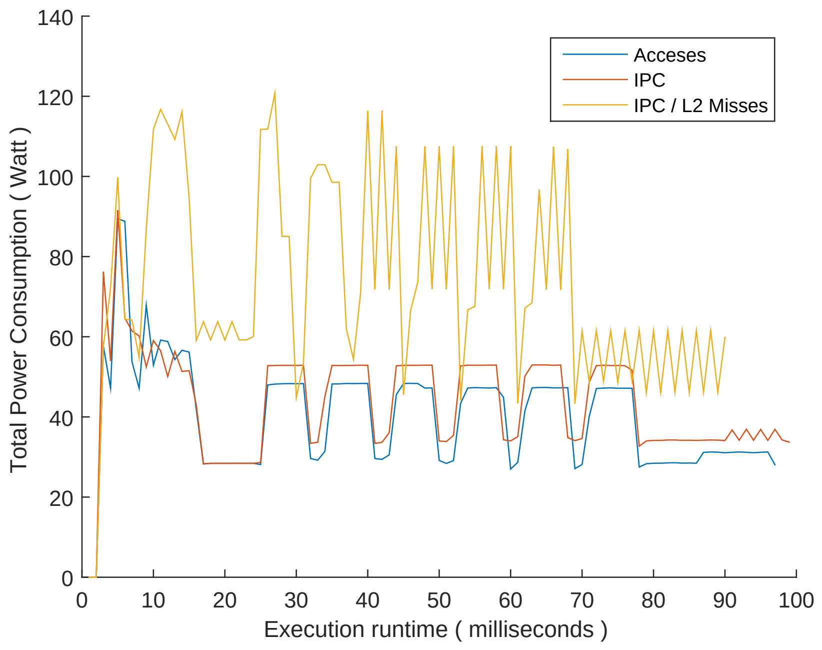


FIGURE 2.6: 8 Core criticality Factors.

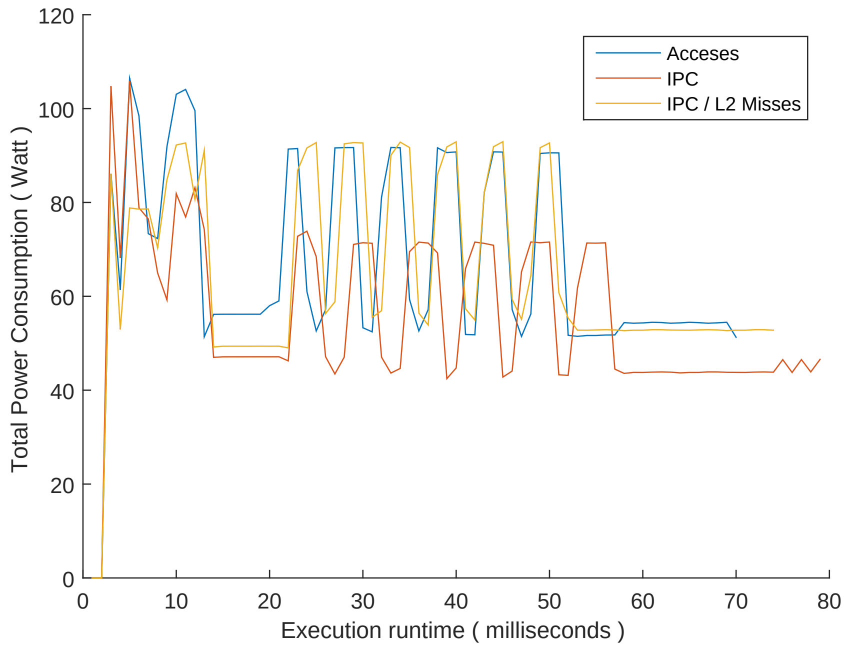


FIGURE 2.7: 16 Core criticality Factors.

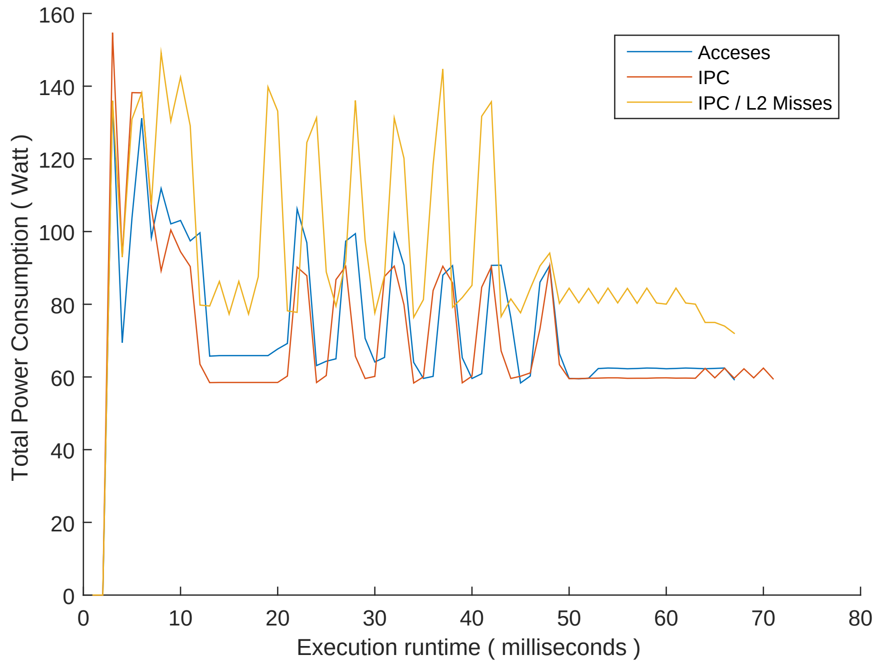


FIGURE 2.8: 32 Core criticality Factors.

Power Allocation

One important task of the Global Manager is deciding how to allocate the chip power budget to the active cores. This needs to be done in an efficient way by taking into consideration both the current performance and the potential increase in performance and power consumption of each core. For this reason the Global Manager needs a metric to measure the criticality of each core and then allocate the power budget properly.

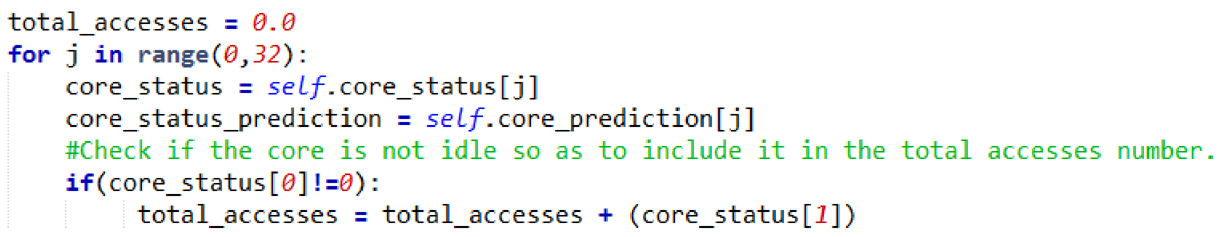
Implementation

After defining the *Core Criticality* a method to allocate the chip power budget is required.Since the core criticality is defined by the Core Cache Accesses then the partition of core X of the total power budget of N active cores will be calculated from this formula:

$$Power_{budget} \cdot \frac{PredictedCoreAccesses_x}{\sum_0^N PredictedCoreAccesses} \tag{2.4}$$

During every interval the Global Manager will sum the cache accesses of the active cores. A core is considered active if it's IPC is greater than zero.
We define the following variables:

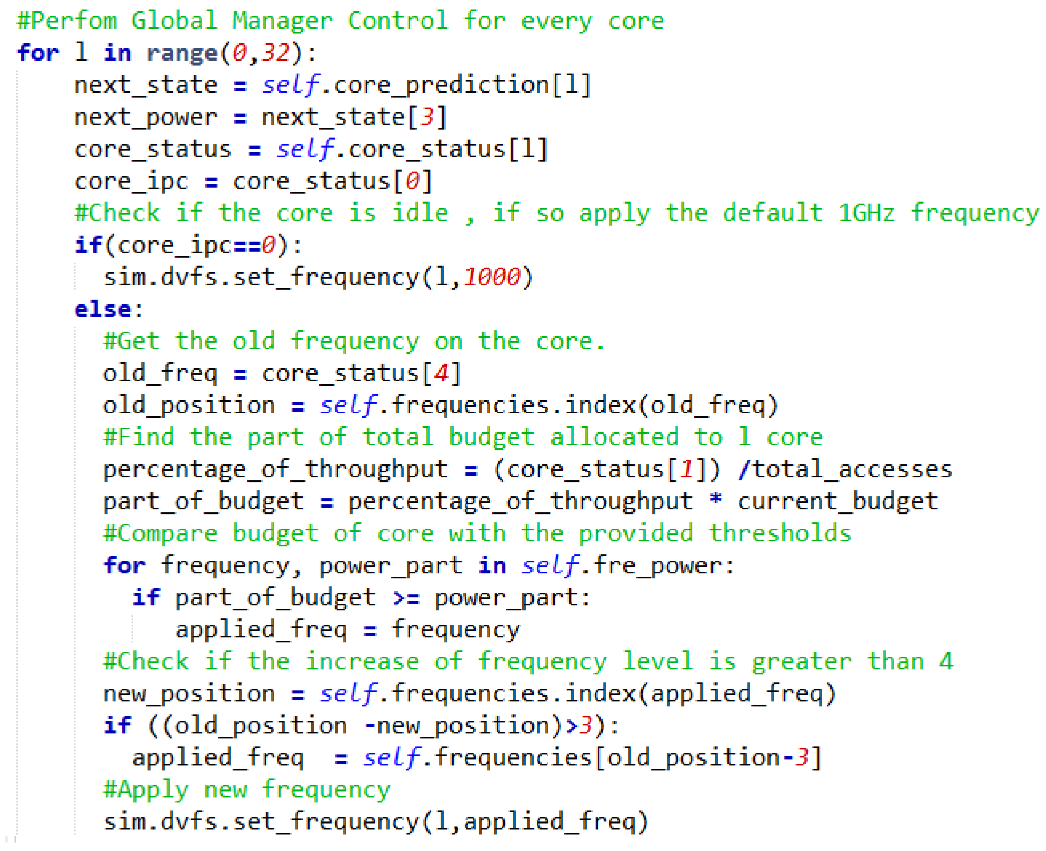
- *core_ status* - an array 1x4 including the core status as we defined in the start. IPC , Core Accesses , Core Misses , Core Energy
- *core_ status_ prediction* - an array 1x4 including the predicted core status in the next interval.



```
total_accesses = 0.0
for j in range(0,32):
    core_status = self.core_status[j]
    core_status_prediction = self.core_prediction[j]
    #Check if the core is not idle so as to include it in the total accesses number.
    if(core_status[0]!=0):
        total_accesses = total_accesses + (core_status[1])
```

Figure 2.9: Global Manager Code.

Furthermore the Global Manager will apply the frequency 1.0 GHz on the idle cores and evaluate the part of the budget for the rest of the cores based on provided formula.
In addition the Global Manager limits the increase of the frequency level by 3 steps in order to to avoid core power spikes.The implementation can be found in the source code below.



```
#Perfom Global Manager Control for every core
for l in range(0,32):
    next_state = self.core_prediction[l]
    next_power = next_state[3]
    core_status = self.core_status[l]
    core_ipc = core_status[0]
    #Check if the core is idle , if so apply the default 1GHz frequency
    if(core_ipc==0):
      sim.dvfs.set_frequency(l,1000)
    else:
      #Get the old frequency on the core.
      old_freq = core_status[4]
      old_position = self.frequencies.index(old_freq)
      #Find the part of total budget allocated to l core
      percentage_of_throughput = (core_status[1]) /total_accesses
      part_of_budget = percentage_of_throughput * current_budget
      #Compare budget of core with the provided thresholds
      for frequency, power_part in self.fre_power:
        if part_of_budget >= power_part:
           applied_freq = frequency
      #Check if the increase of frequency level is greater than 4
      new_position = self.frequencies.index(applied_freq)
      if ((old_position -new_position)>3):
        applied_freq  = self.frequencies[old_position-3]
      #Apply new frequency
      sim.dvfs.set_frequency(l,applied_freq)
```

FIGURE 2.10: Global Manager Policy Code.

Frequency - Power Budget

When the power budget has been allocated as partitions properly to all the active cores the Global Manager needs to decide what is the corresponding frequency that needs to be applied on each core in order to meet the power requirements.
For this decision to be made a threshold is set for each one of the 6 frequency levels. This threshold is a requirement for a core's power budget to meet if the specific frequency is to be applied on that core. This method provides a safety against over-budgeting since the Global Manager will not apply a frequency on a core if that core has the potential to reach a power consumption out of it's limits.

In order to calculate the thresholds for the frequency levels , I used the analyzed data that were gathered. By using the boxplots of the power consumptions on every level a value between the **third quartile** and the **median** was used as a threshold. Below we can see the boxplots of all the frequencies :

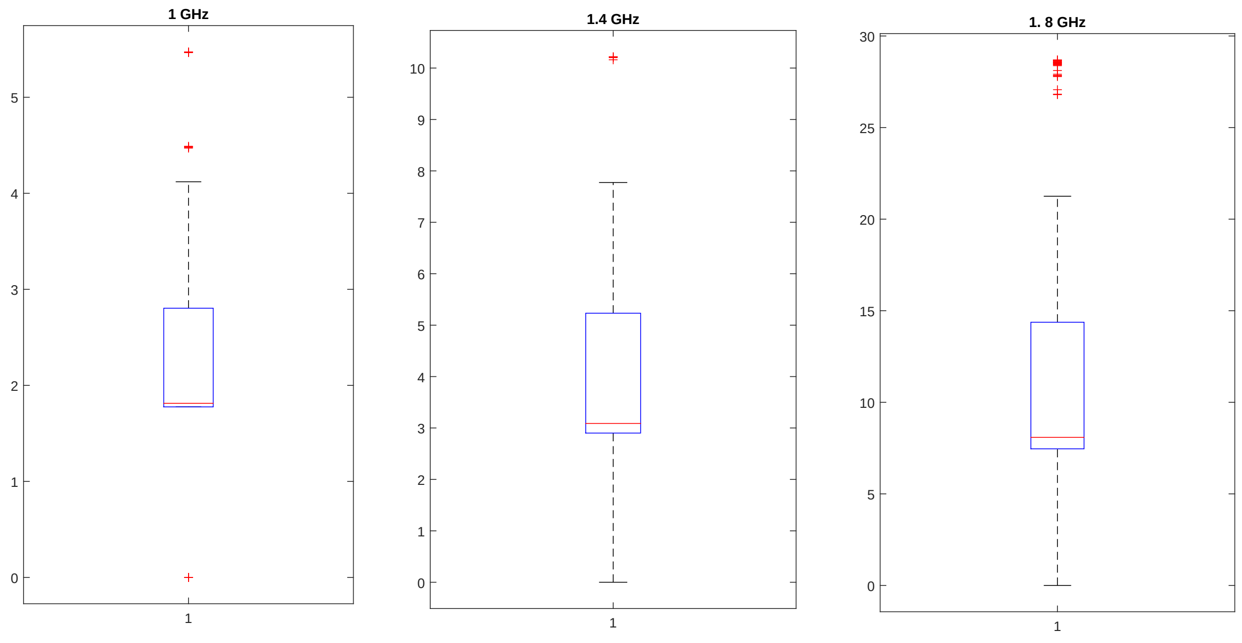


FIGURE 2.11: 1.0GHz , 1.4GHz , 1.8 GHZ Boxplots

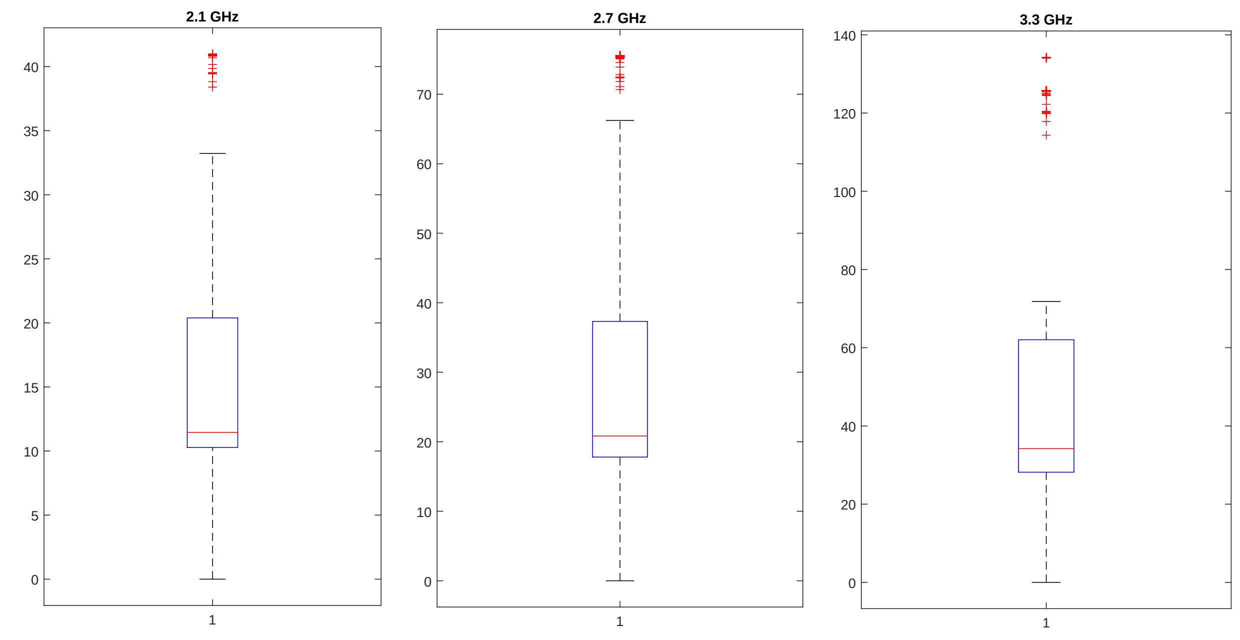


FIGURE 2.12: 2.1GHz , 2.7GHz , 3.3 GHZ Boxplots

Here is the table of the power threshold and the frequencies:

Power Threshold. (Watt)	**Frequency Level (GHz)**
2	1.0
4.0	1.4
11.0	1.8
18.40	2.1
30.0	2.7
55.0	3.3

Chip Power Budget

The chip power budget has been decided by taking into consideration the number of active cores. In the same philosophy as in the Individual Manager , the chip power budget needs to scale with the number of active cores because that provides a more realistic behavior. For the specific benchmarks that have been used , the bodytrack benchmarks use 2 extra cores (threads) so that is why the chip power budget for bodytrack has been presented seperately from the other two.

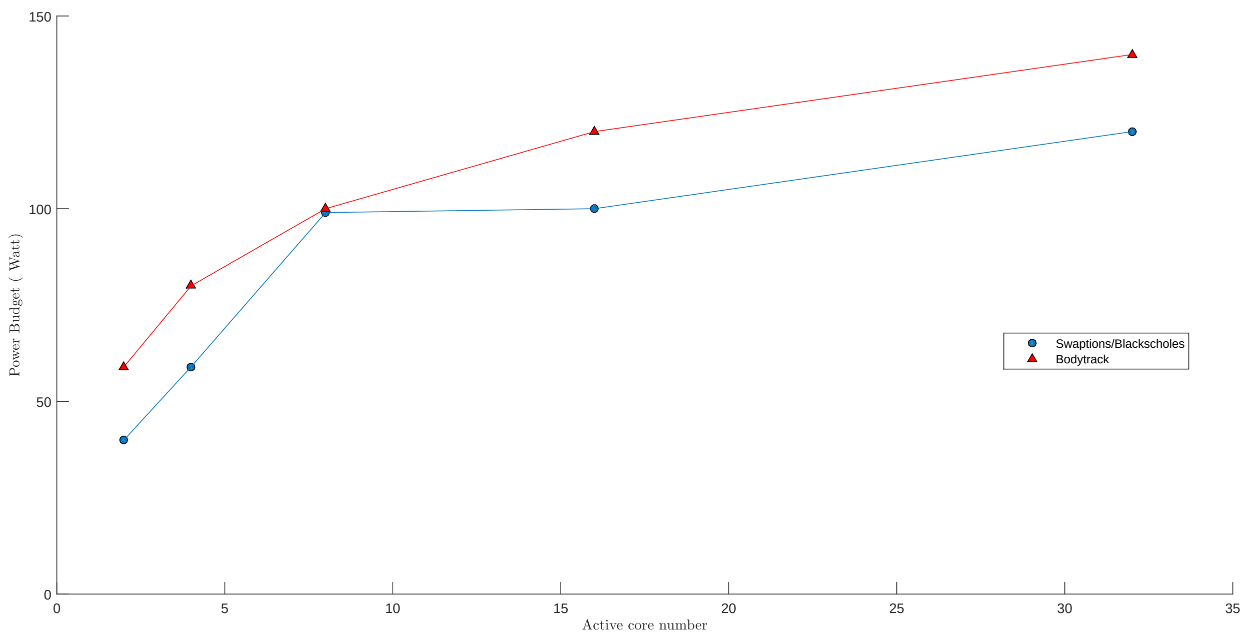


FIGURE 2.13: Global Chip Power Budget

Chapter 3

Experimental Results

In this section we will analyse the behavior of the three benchmarks that were used and the impact of the manager implementations on these benchmarks.

3.1 Benchmark Analysis

3.1.1 Blackscholes

The first benchmark that was used is *blackscholes* which is characterized by a small working set and low sharing data behavior. This means that blackscholes presents low barrier behavior. In the following figures we can see the effect of the number of active cores on the **energy consumption** which is diplayed in KJoule and the **total execution time** of the benchmark.The different values for the number of the active cores are 2 , 4 , 8 , 16 , 32 . Blakscholes spawns an extra thread from the number of active cores which will be the master thread with lower throughput than the rest slave threads.
As the number of active cores increases from 2 up to 8 the total energy consumption decreases, this means that the workload is distributed evenly to the active cores.However as the number of active cores further increases we can see that the total energy consumption increases. This is a result of the lock behavior that characterizes Blackscholes as the further increase of active cores does not translate into better work load allocation.
Concerning the total execution time we can see that when the active core number changes from 2 to 4 the execution time decreases by a margin of 30 %. However the decrease of the benchmark execution time does not follow the corresponding increase of the active core. This occurs since the greater number of active cores does not offer significant increase to the throughput because of the lock behavior.

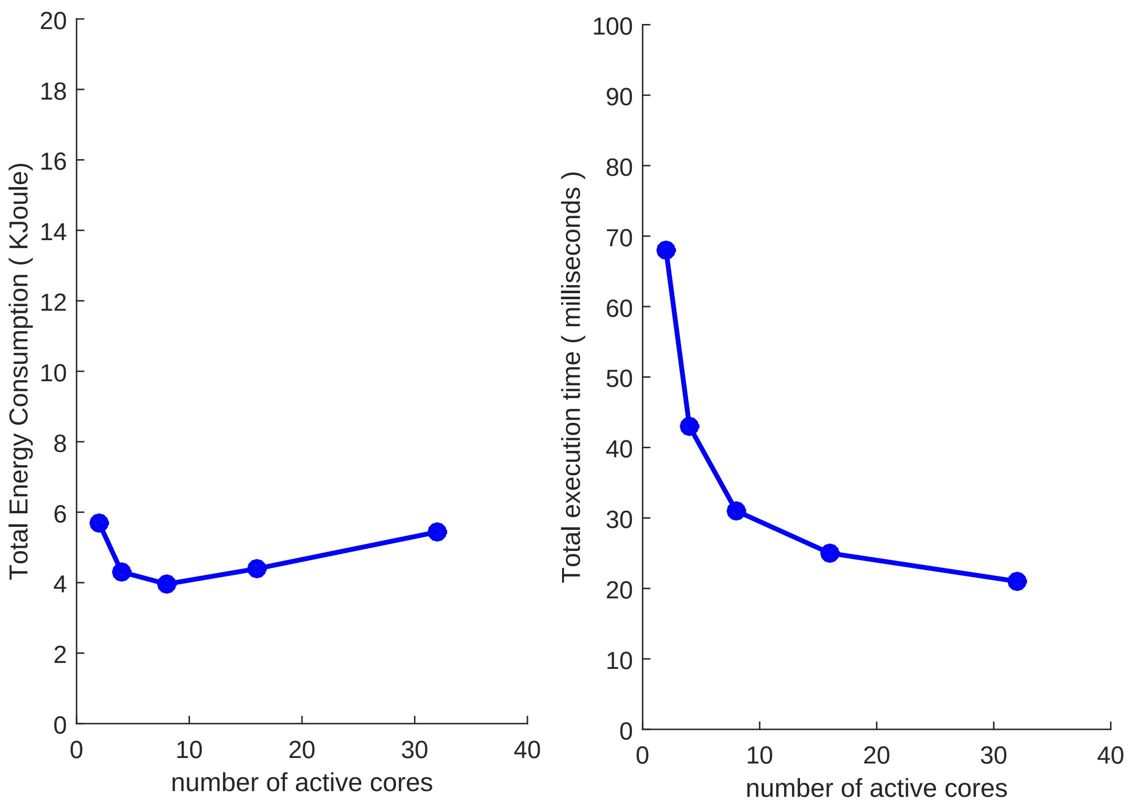


FIGURE 3.1: Blackscholes Analysis.

3.1.2 Bodytrack

The second benchmark that was used is *bodytrack* which is characterized by a medium working set and a high sharing data behavior. Bodytrack is expected to present similar behavior with blackscholes.Bodytrack similarily spawn extra threads but in this case the extra threads are two.As we can see in the first figure the execution time of bodytrack drops significantly as the number of cores increases from 2 to 4 which is a result of the high working set of bodytrack. However as the number of active cores increases the lock behavior of bodytrack limits the decrease of the runtime in the same way as was observed for blackscholes.Furthermore we observe a small decrease in the energy consumption when the number of the active cores is low which is a result of the proper workload allocation.However the energy consumption follows the increase of the active cores when the number of active cores is higher than 8. This is the same phenomenon that was observed for blackscholes as well.

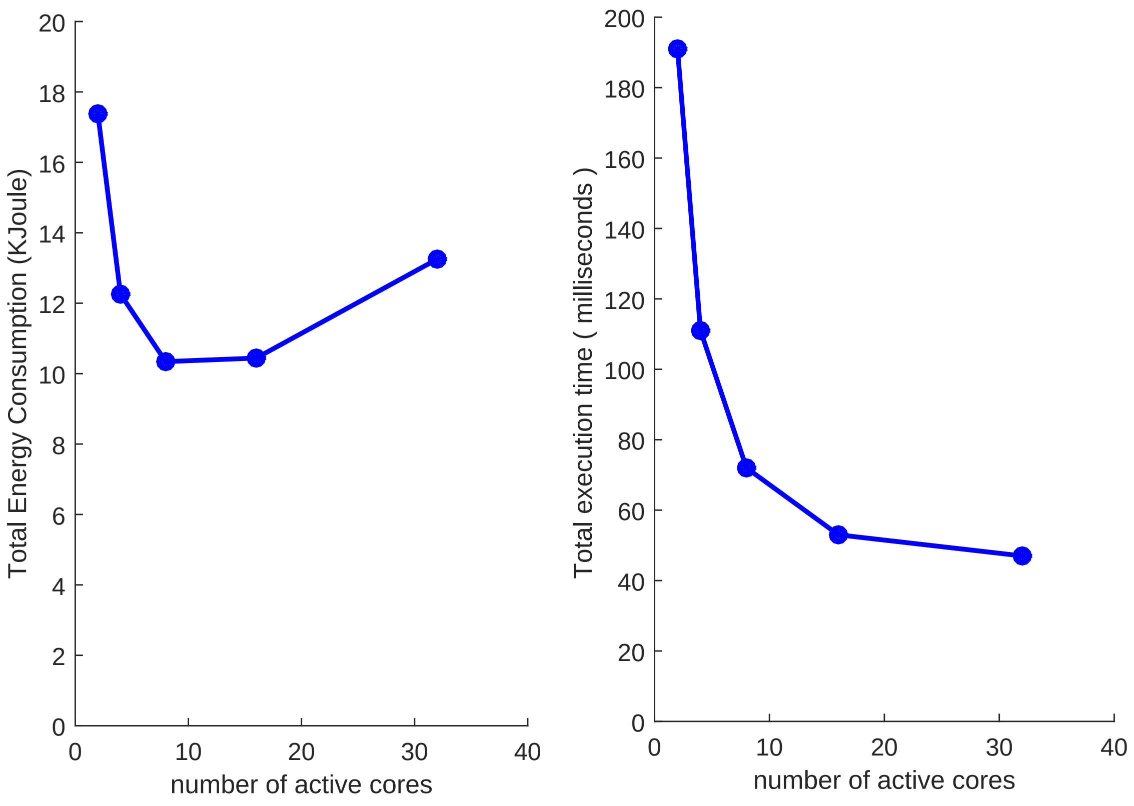


FIGURE 3.2: Blackscholes Analysis.

3.1.3 Swaptions

The third benchmark that was used is *swaptions* which is characterized by a high workload with very low lock behavior. Swaptions is different from the previous benchmarkss concerning it's behavior and is expected to contribute to the prediction of non-lock but high workload application behaviors.Below we can see that since there is no bottleneck for the swaptions benchmark the total energy consumptions decreases as the number of active cores increases. This means that the workload is properly allocated to the newly used cores and that is described in the Total execution time figure. We can see that the total execution time decreases accordingly with the increase of the number of active cores.

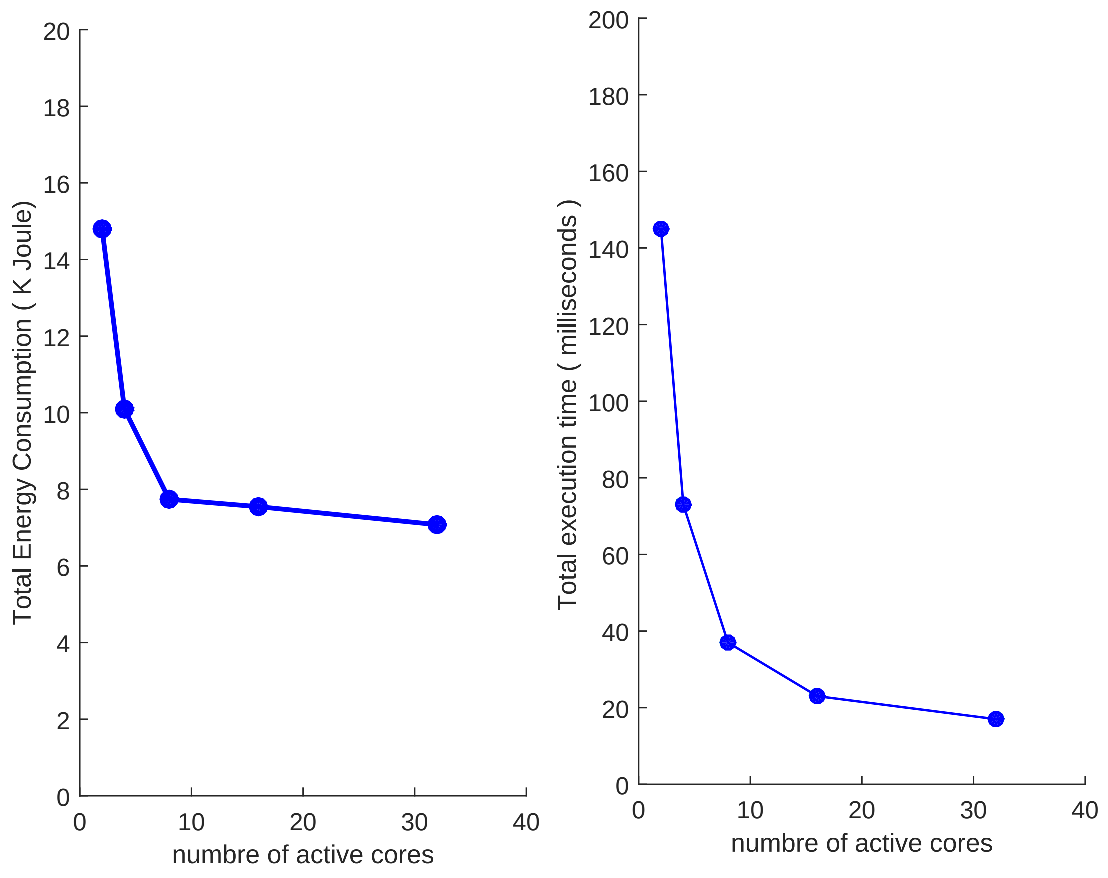


FIGURE 3.3: Swaptions Analysis.

3.2 Individual Manager Analysis

The individual manager as described before monitors each core and ensures that the cores power consumption during every interval will not exceed the core power budget. As displayed in Figure 2.2 the power budget allocated to each core depends on the active core number in order to implement a more realistic simulation. Below we display the power consumption of all the active cores during the simulation and the overall chip power consumption for the three benchmarks and all the different active core configurations :

3.2.1 Blackscholes

For the configurations of 2 and 4 active cores , the chip power consumption is below the required threshold of 120 Watt and the individual power consumption of the active cores exceeds the budget 2 times in each case. As we can see the performance of the individual manager is satisfying for these configurations as expected since when the number of

active cores is rather small , the possibility of chip overbudgeting is low.
However for the configuration of 8 active cores , the chip power consumption exceed several times the threshold of 120 Watt. For the last 2 configurations of 16 and 32 cores , we can see the individual power consumption is constrained below the individual core budget , but as expected the overall power chip exceeds the threshold. As it was expected the individual manager for all the configuration keeps the each core power consumption below the provided budget but as the number of active cores scales , the overall chip power consumption reaches non realistic power consumption values.

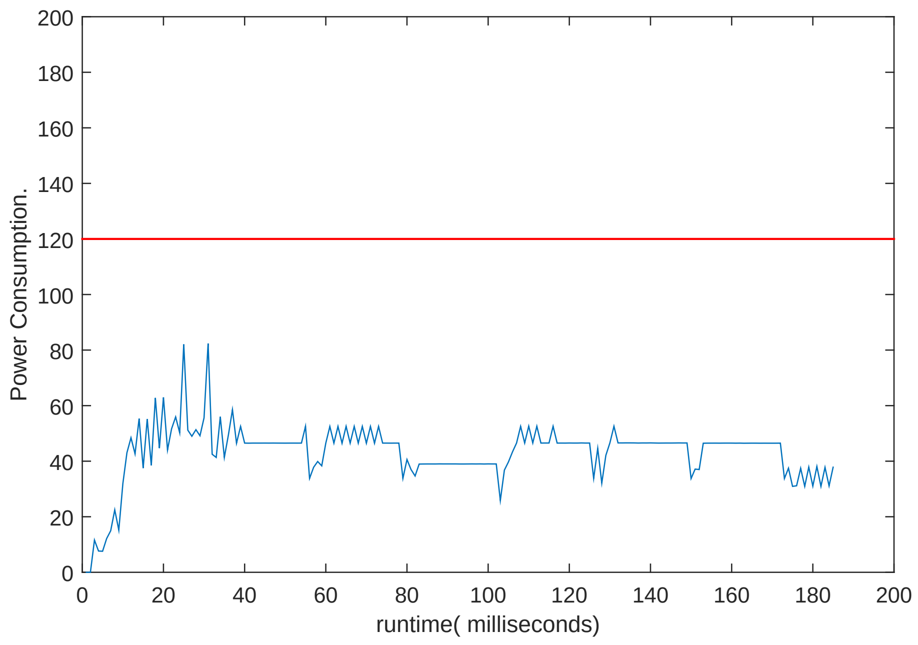


Figure 3.4: Blackscholes 2 active cores Power Consumption.

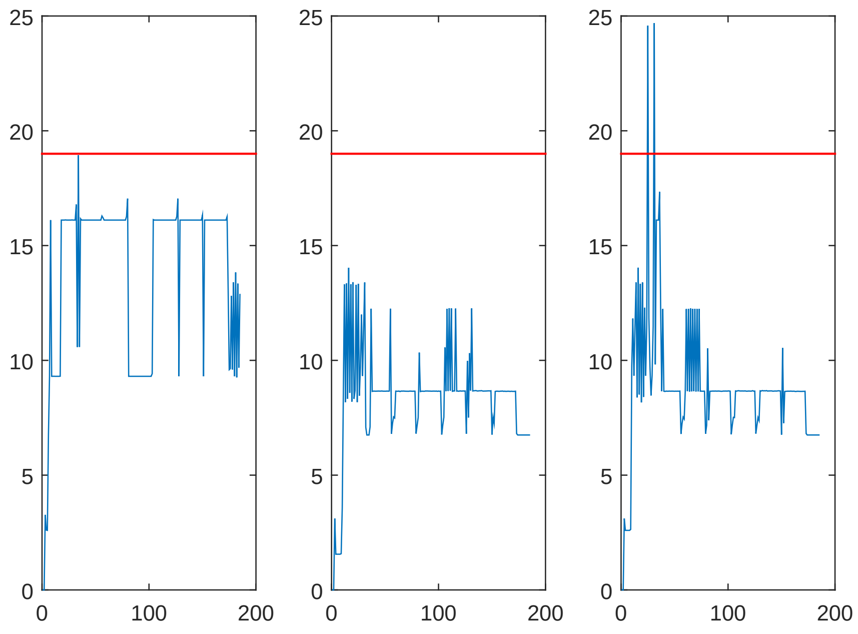


FIGURE 3.5: Blackscholes 2 active cores + master core.

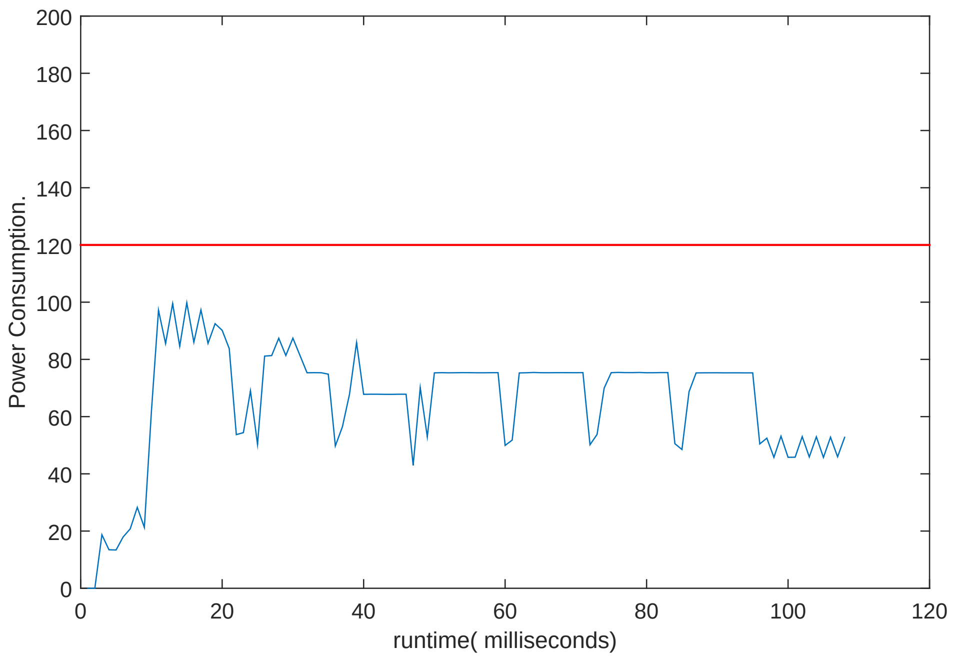


FIGURE 3.6: Blackscholes 4 active cores Power Consumption.

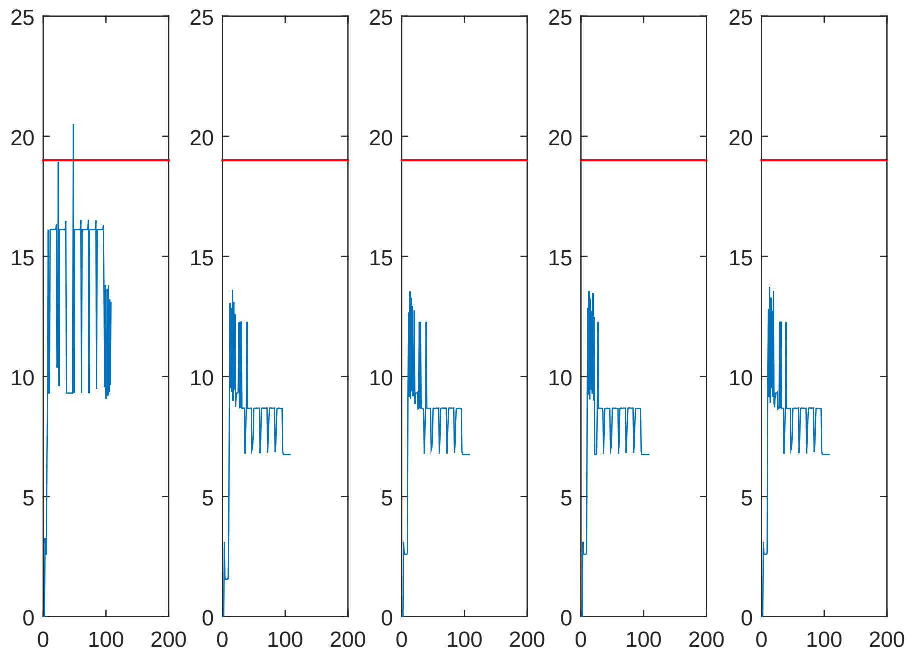


FIGURE 3.7: Blackscholes 4 active cores + master core.

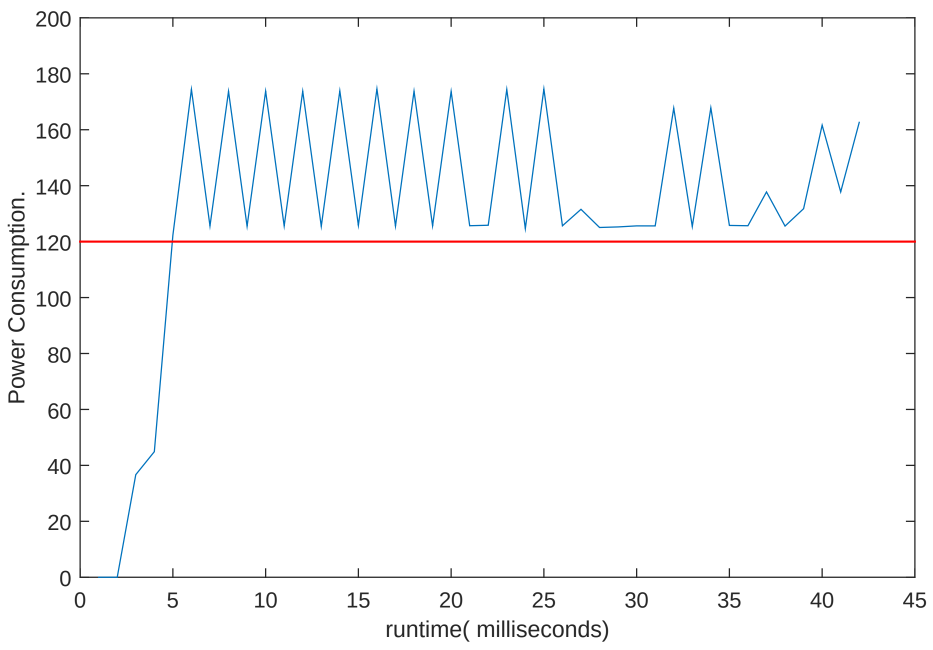


FIGURE 3.8: Blackscholes 8 active cores Power Consumption.

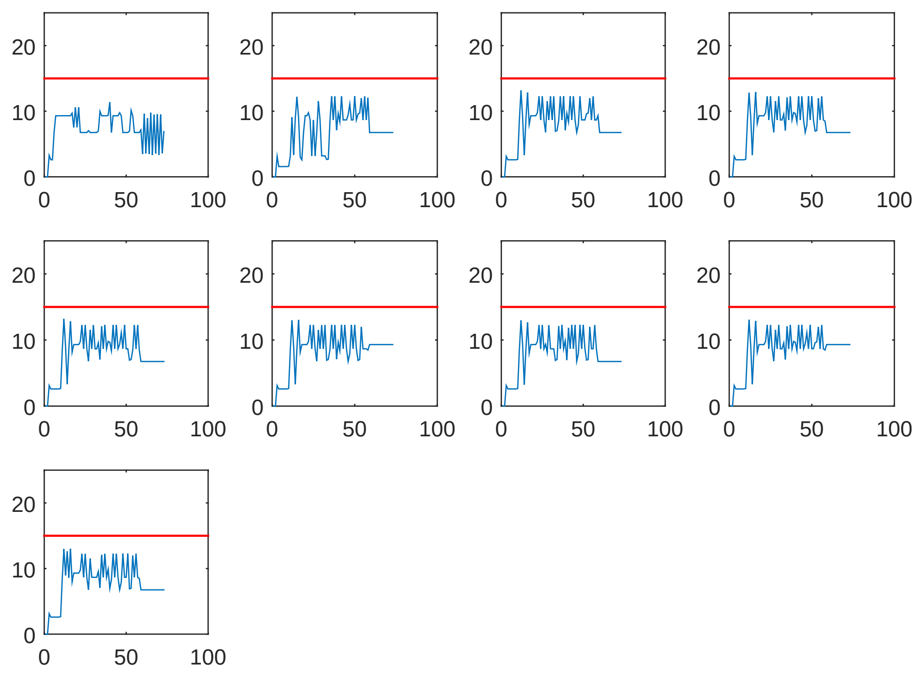


Figure 3.9: Blackscholes 8 active cores + master core.

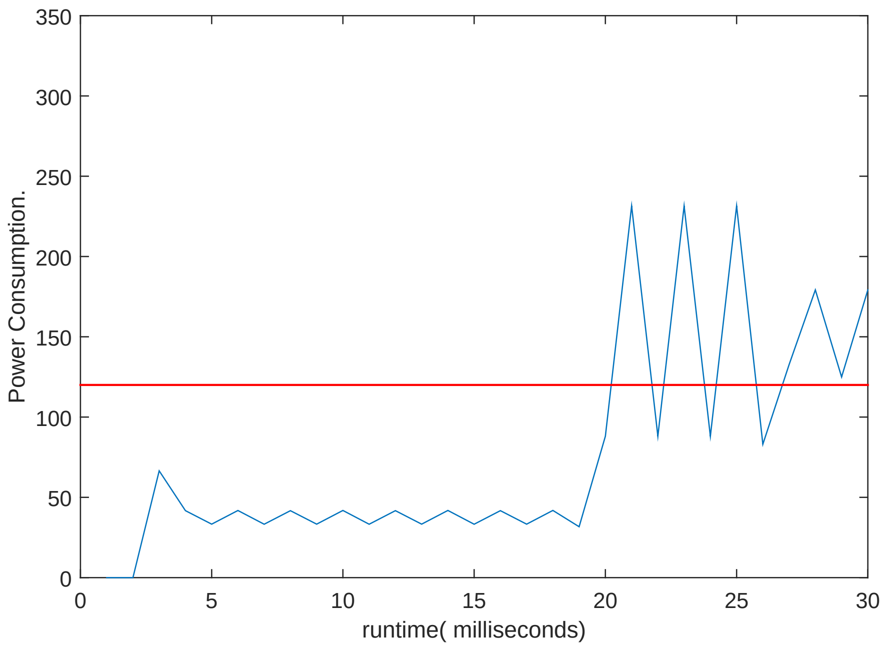


Figure 3.10: Blackscholes 16 active cores Power Consumption.

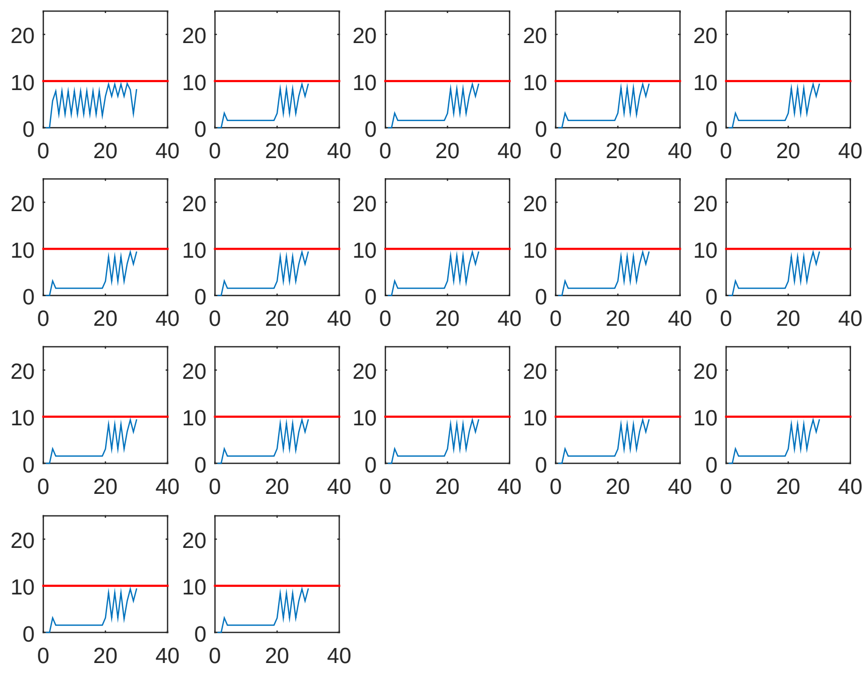


FIGURE 3.11: Blackscholes 16 active cores + master core.

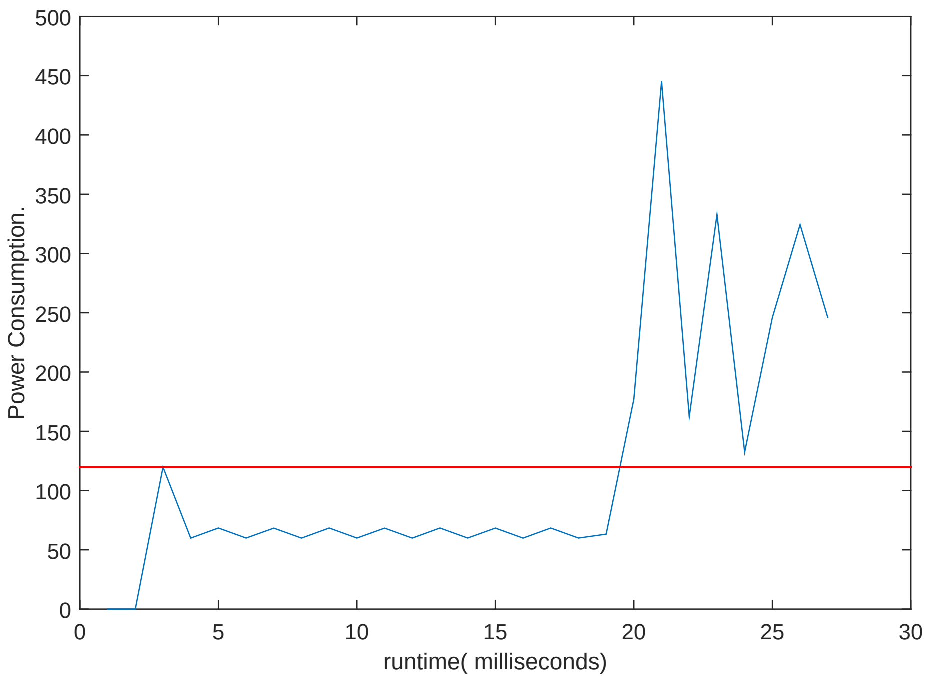


FIGURE 3.12: Blackscholes 32 active cores Power Consumption.

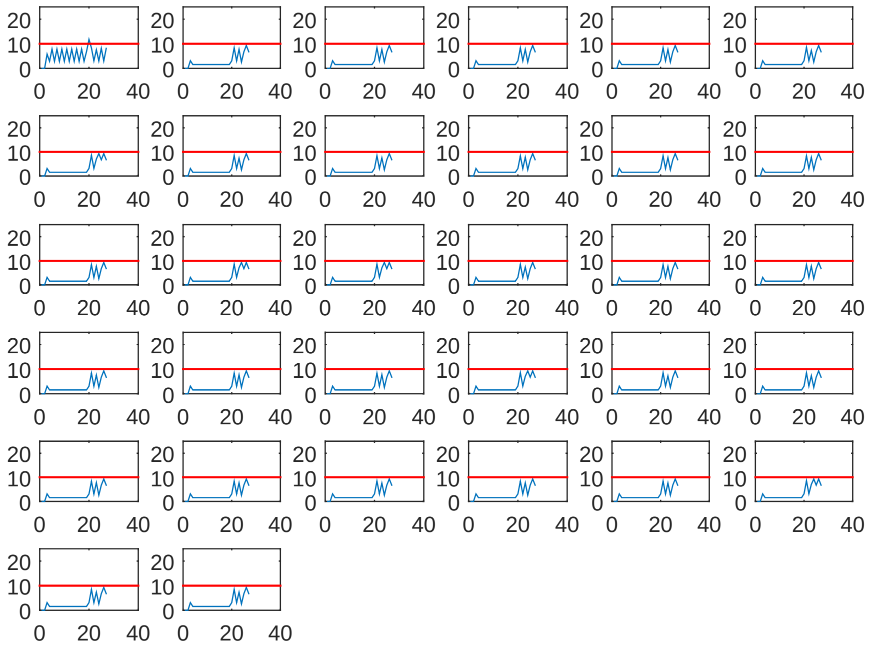


Figure 3.13: Blackscholes 32 active cores + master core.

3.2.2 Swaptions

As we can see the individual power consumptions exceeds the according power budget only on the configuration of 16. On the rest of the configuration the individual manager has kept the core power consumption below the threshold. For the overall chip power as expected only for the configurations with a small amount of cores the chip power level is kept below the required threshold.
As the number of the cores increases the overall power budget follows accordingly which as a results leads to overbudgeting for all the configurations expect for the configuration of 2 and 4 cores.
Furthermore we can see that since swaptions is a benchmark with no barries , the individual manager has an easier task to control the power consumption and predict each core's behavior. This can be also determined from the figures of the individual core consumption.

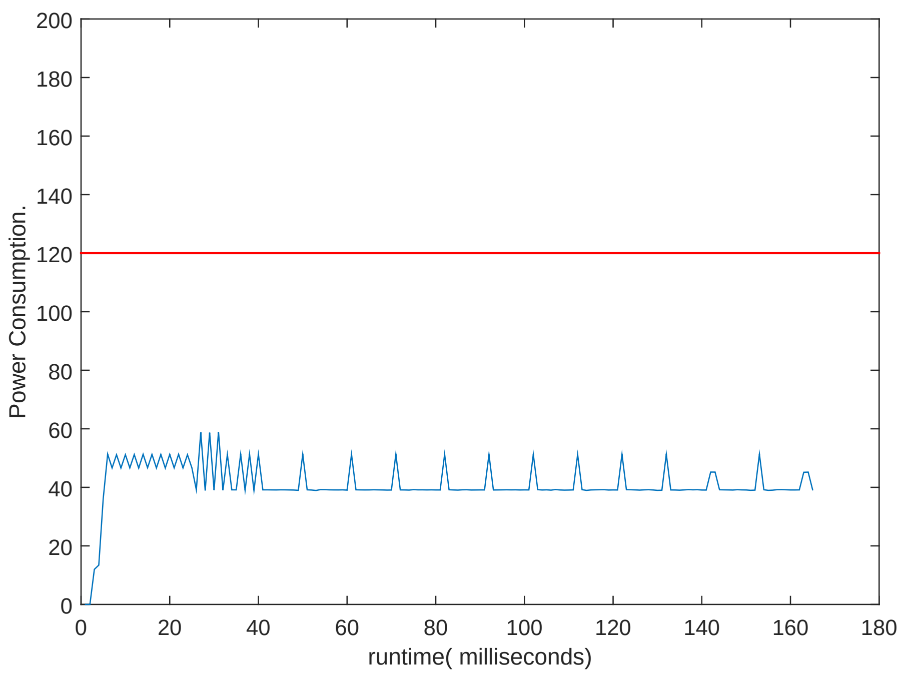


Figure 3.14: Swaptions 2 active cores Power Consumption.

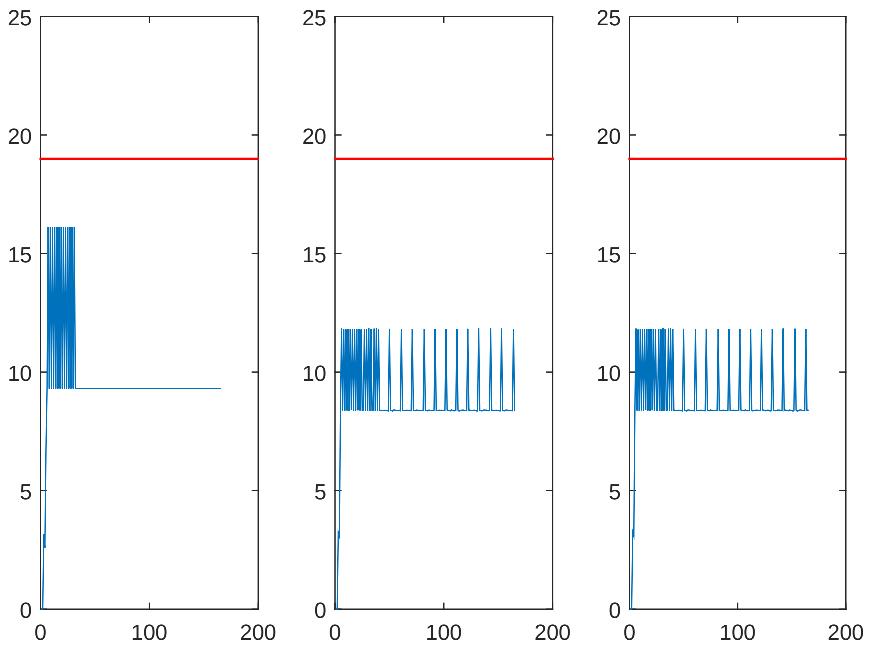


Figure 3.15: Swaptions 2 active cores + master core.

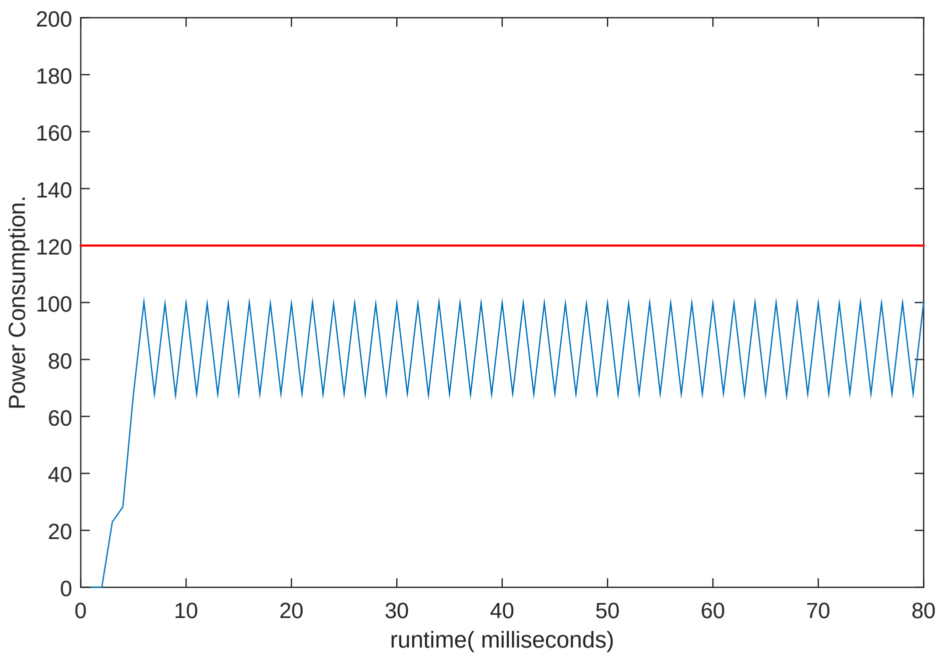


FIGURE 3.16: Swaptions 4 active cores Power Consumption.

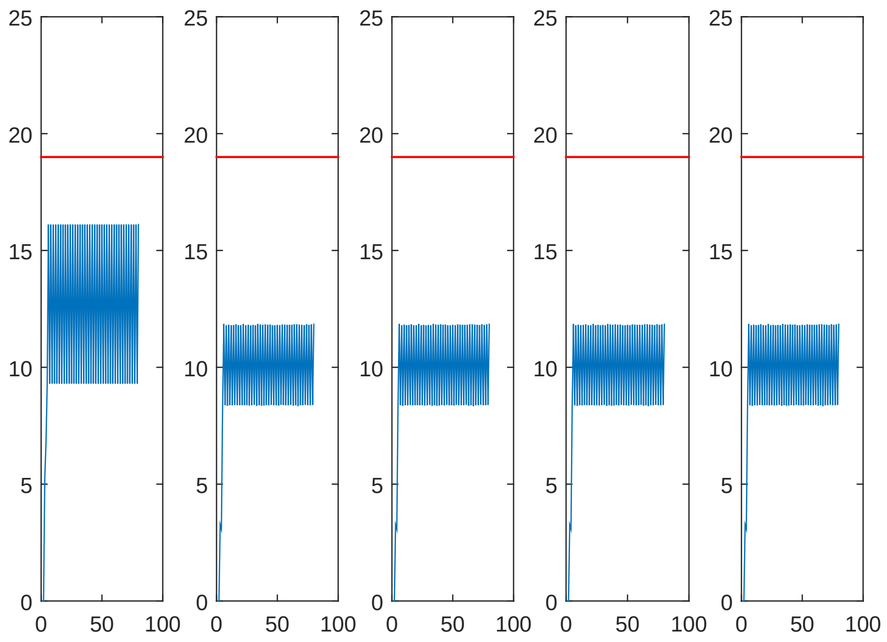


FIGURE 3.17: Swaptions 4 active cores + master core.

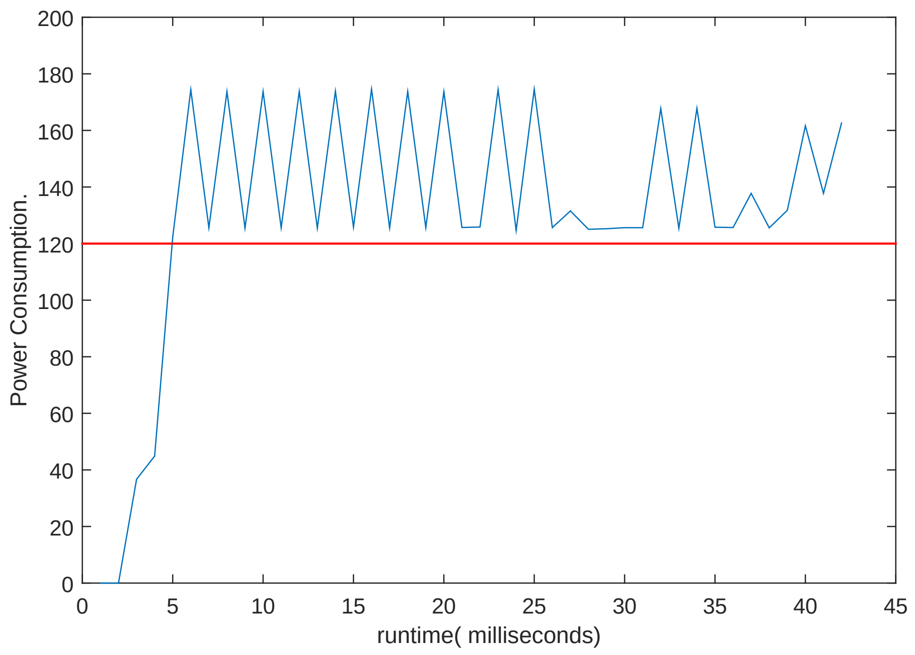


FIGURE 3.18: Swaptions 8 active cores Power Consumption.

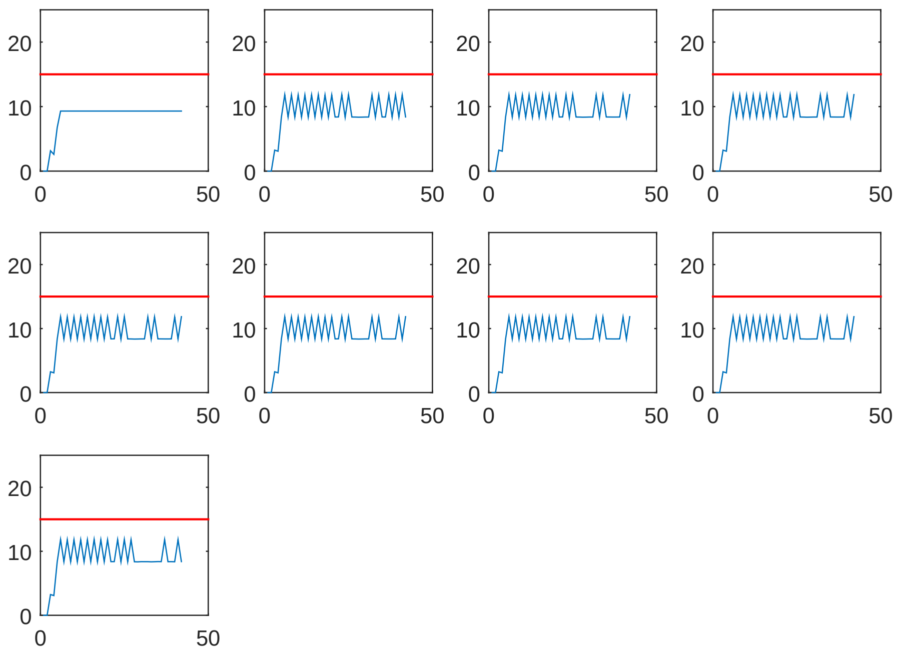


FIGURE 3.19: Swaptions 8 active cores + master core.

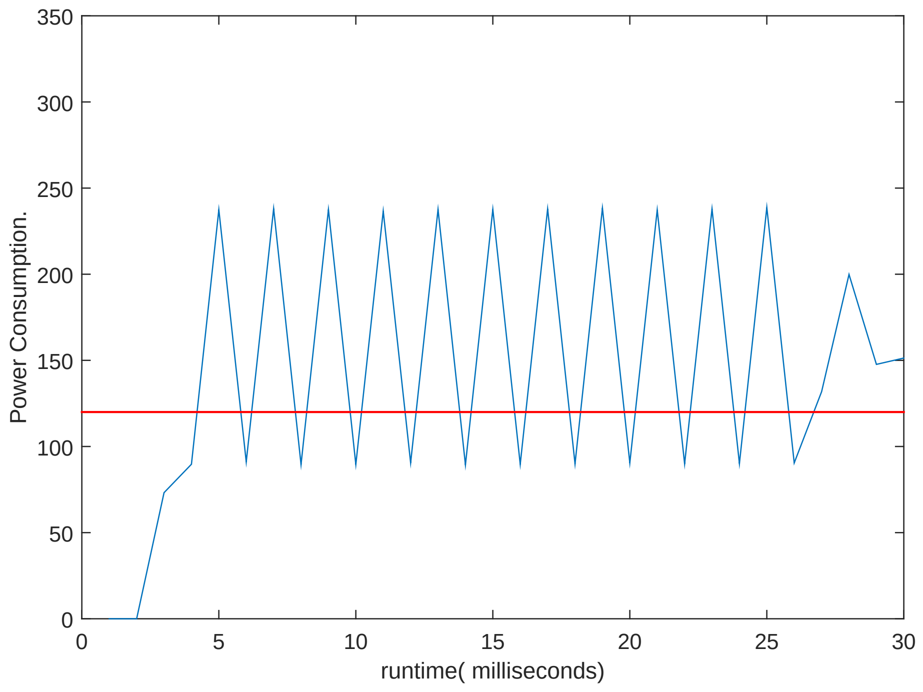


FIGURE 3.20: Swaptions 16 active cores Power Consumption.

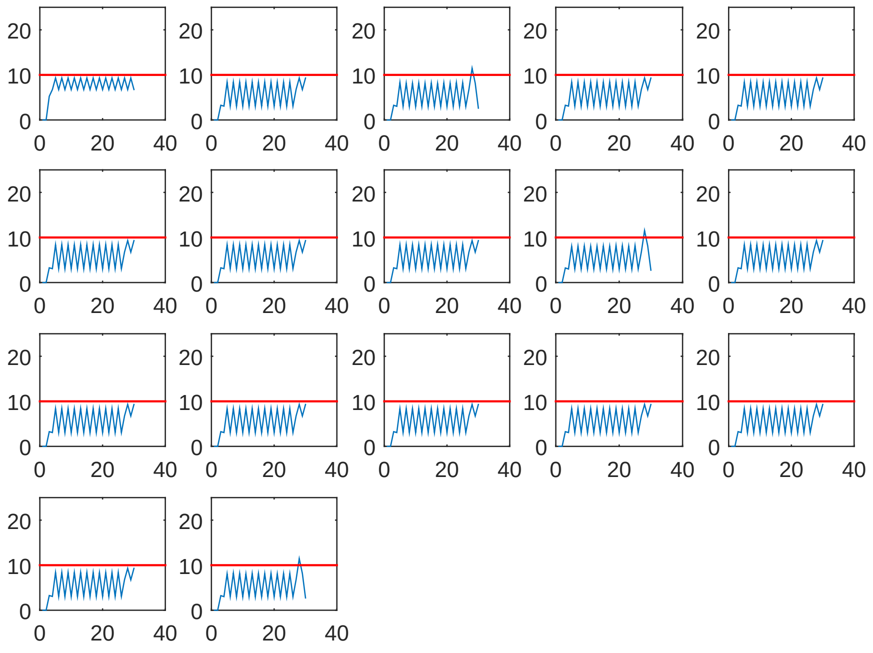


FIGURE 3.21: Swaptions 16 active cores + master core.

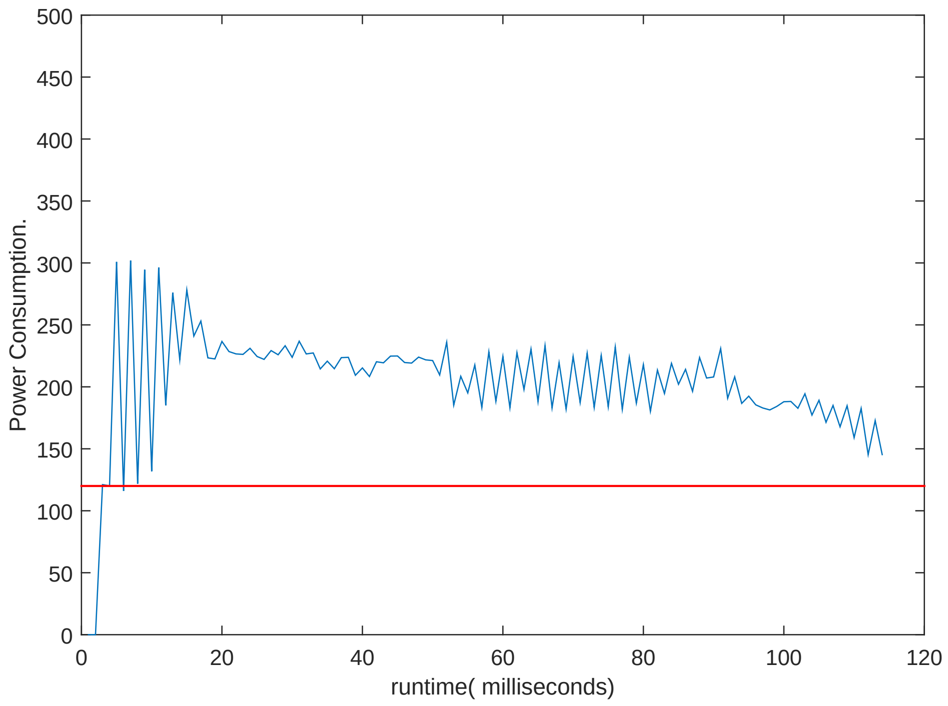


FIGURE 3.22: Swaptions 32 active cores Power Consumption.

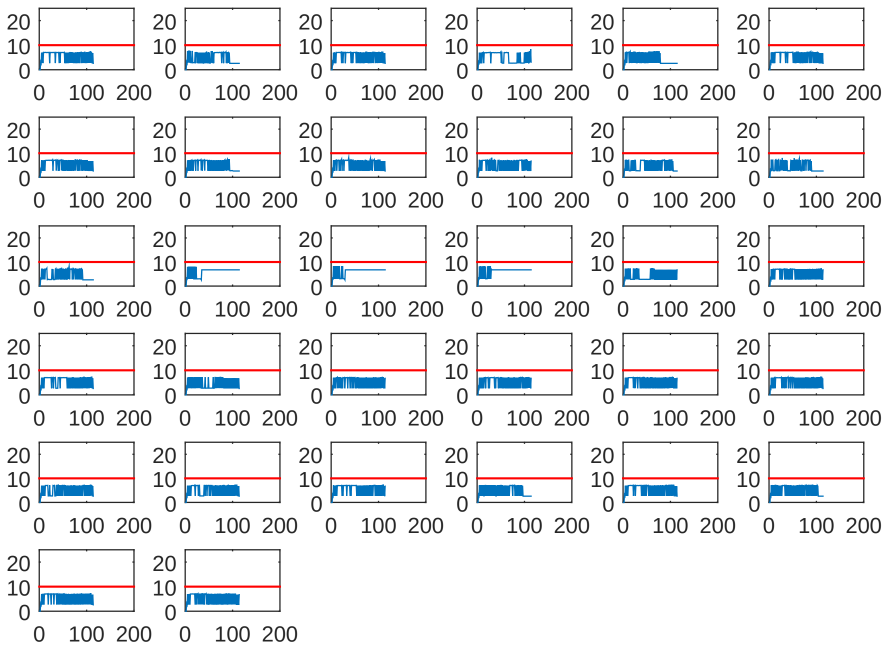


FIGURE 3.23: Swaptions 32 active cores + master core.

3.2.3 Bodytrack

In the configuration of 2 active cores the total chip budget is below the threshold of 120 Watt and the power consumption of each core exceeds the core budget only 5 times. The rest of the core power levels are below the budget.
In the configuration of 4 active cores the total chip budget is still below the threshold of 120 Watt and as we can see the power consumption of each core exceeds the core budget only 2 times.
In the next three configurations the total chip budget is above the threshold for most of the runtime of the simulation. This is an expected phenomenon since the individual manager doesn't take into consideration the over-all chip budget but instead monitors every core individually. Furthermore we can see that the individual core budget exceeds the specifed budget in more instances for the configuration of 16 and 32 active cores. This phenomenon occurs because of the burst of throughput that characterizes the slave threads when they spawn.

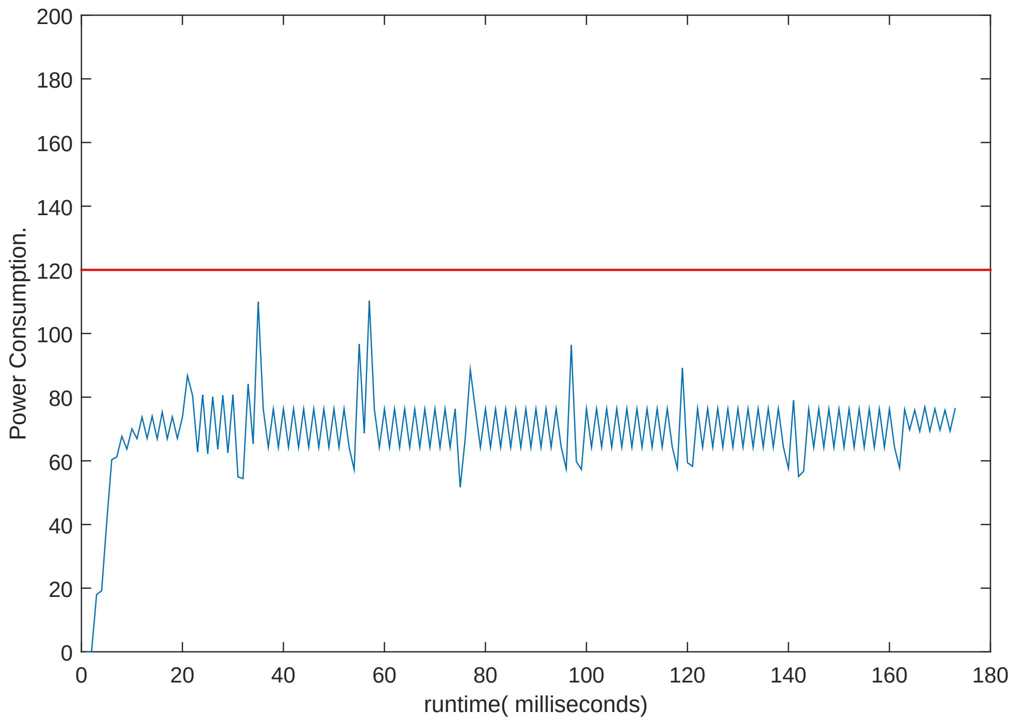


Figure 3.24: Bodytrack 2 active cores Power Consumption.

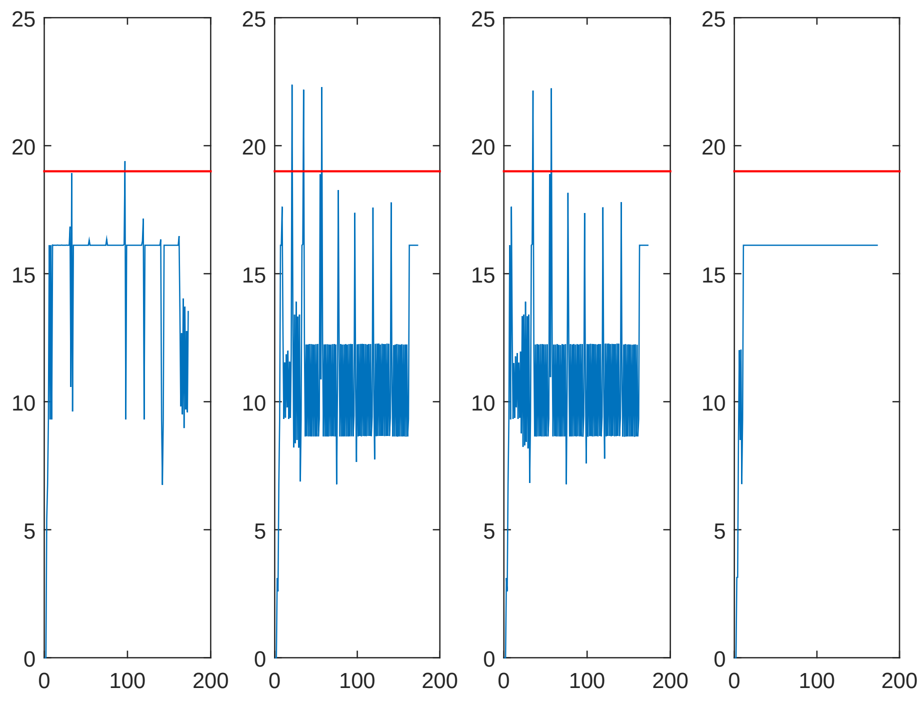


FIGURE 3.25: Bodytrack 2+1 active cores + master core.

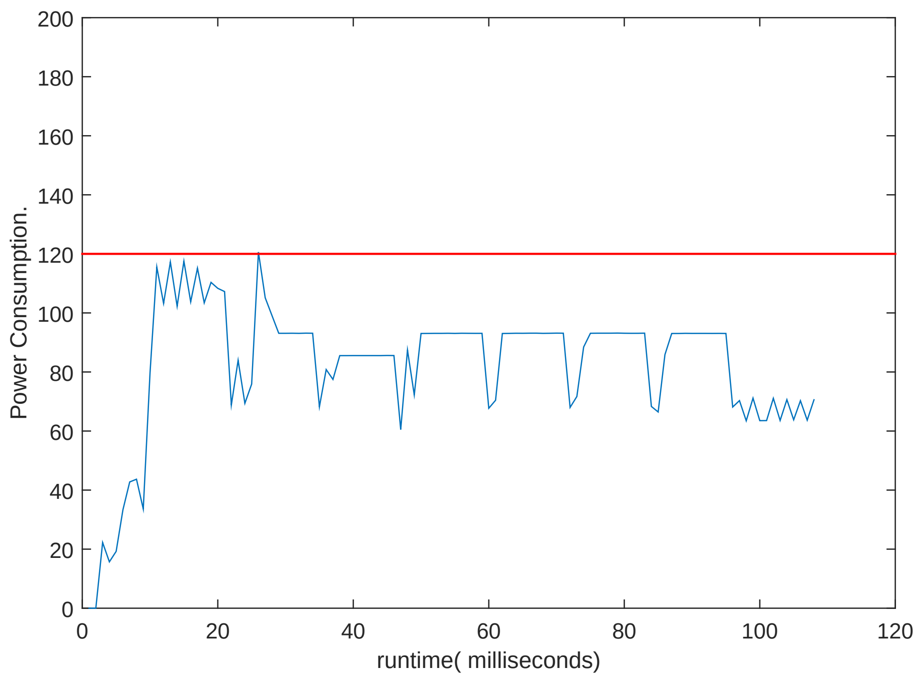


FIGURE 3.26: Bodytrack 4 active cores Power Consumption.

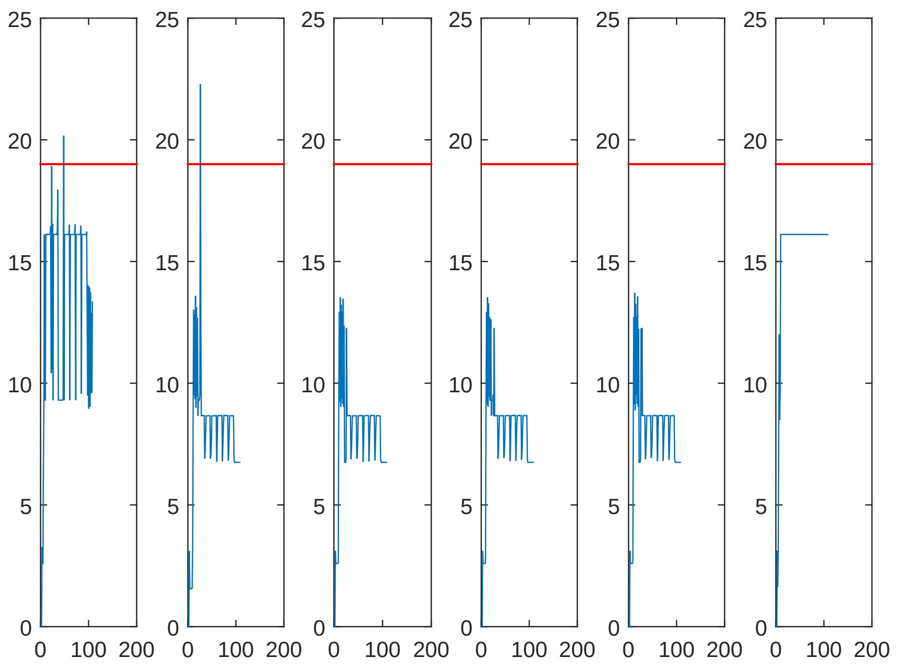


FIGURE 3.27: Bodytrack 4+1 active cores + master core.

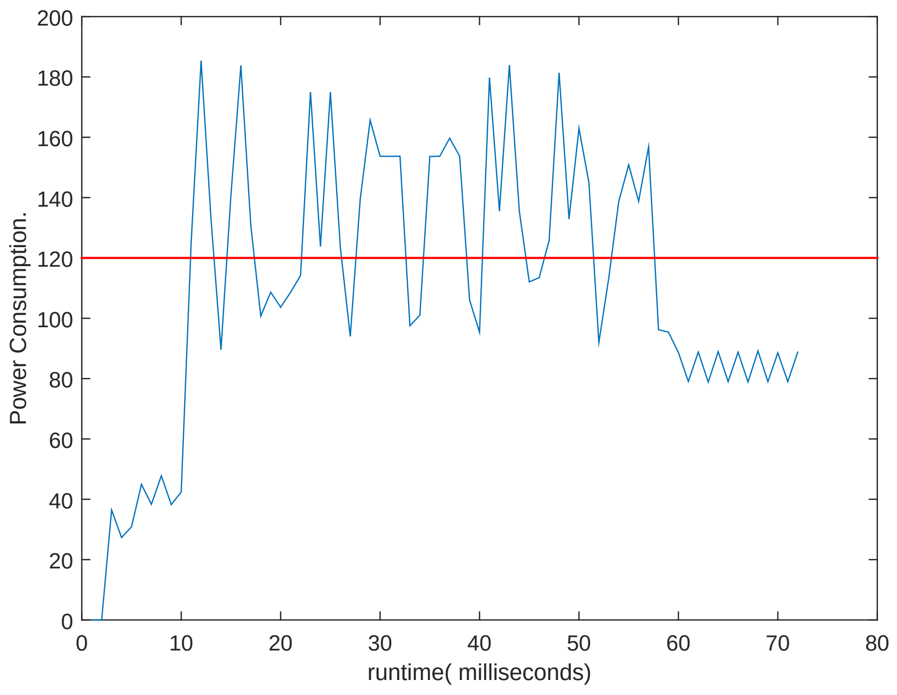


FIGURE 3.28: Bodytrack 8 active cores Power Consumption.

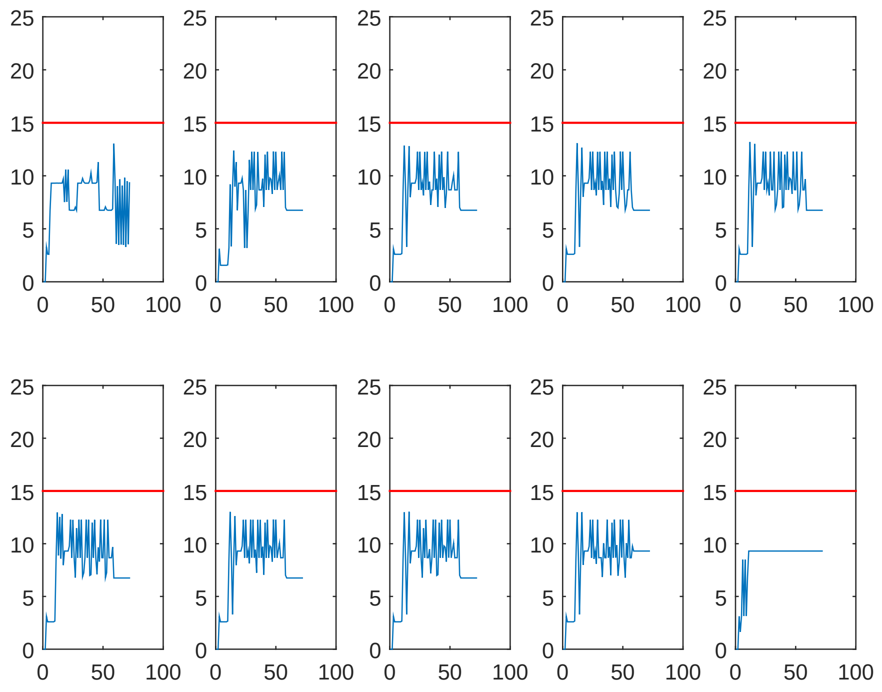


FIGURE 3.29: Bodytrack 8+1 active cores + master core.

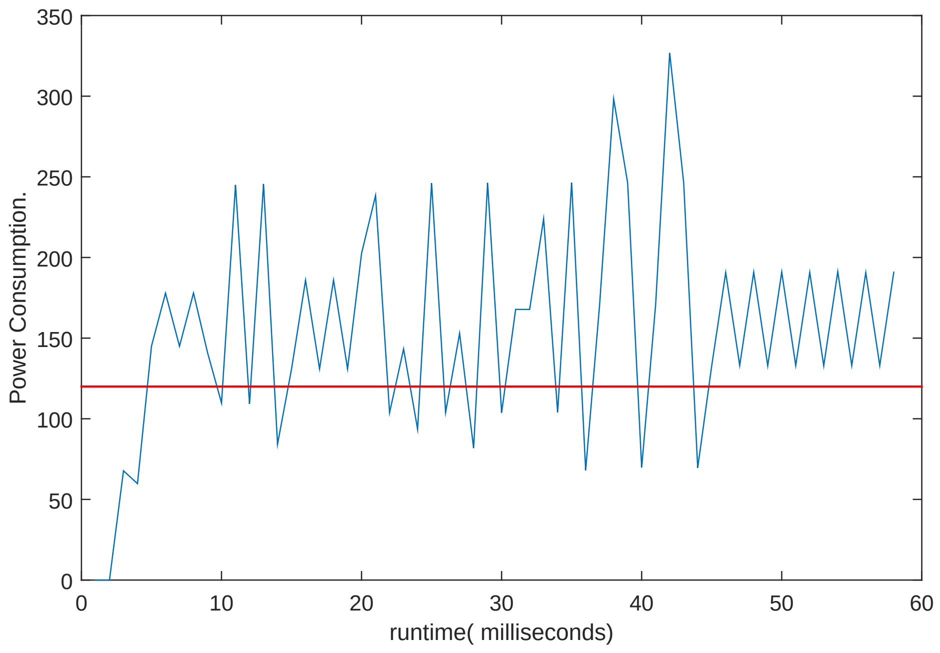


FIGURE 3.30: Bodytrack 16 active cores Power Consumption.

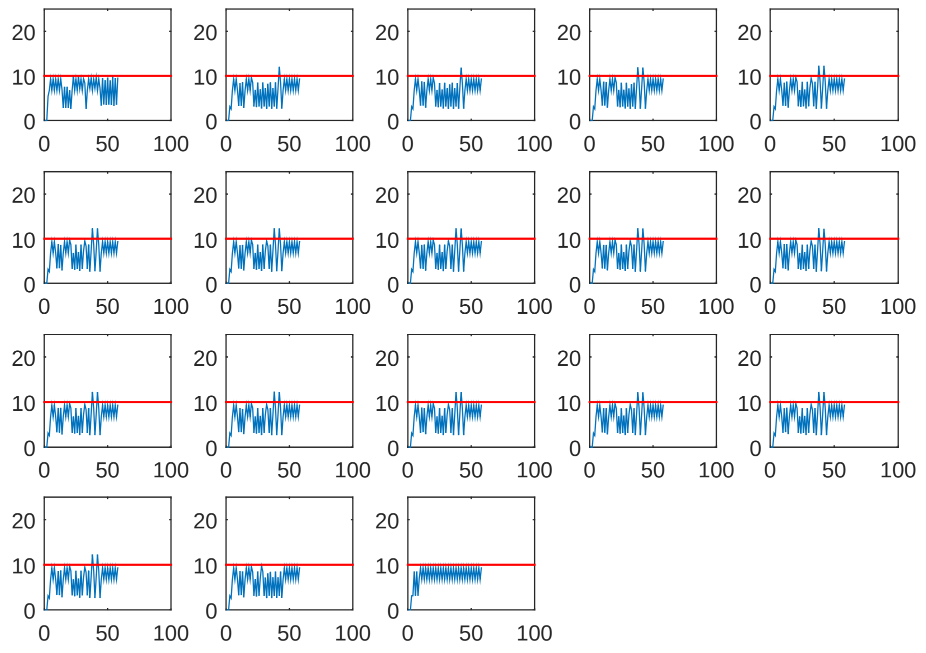


Figure 3.31: Bodytrack 16+1 active cores + master core.

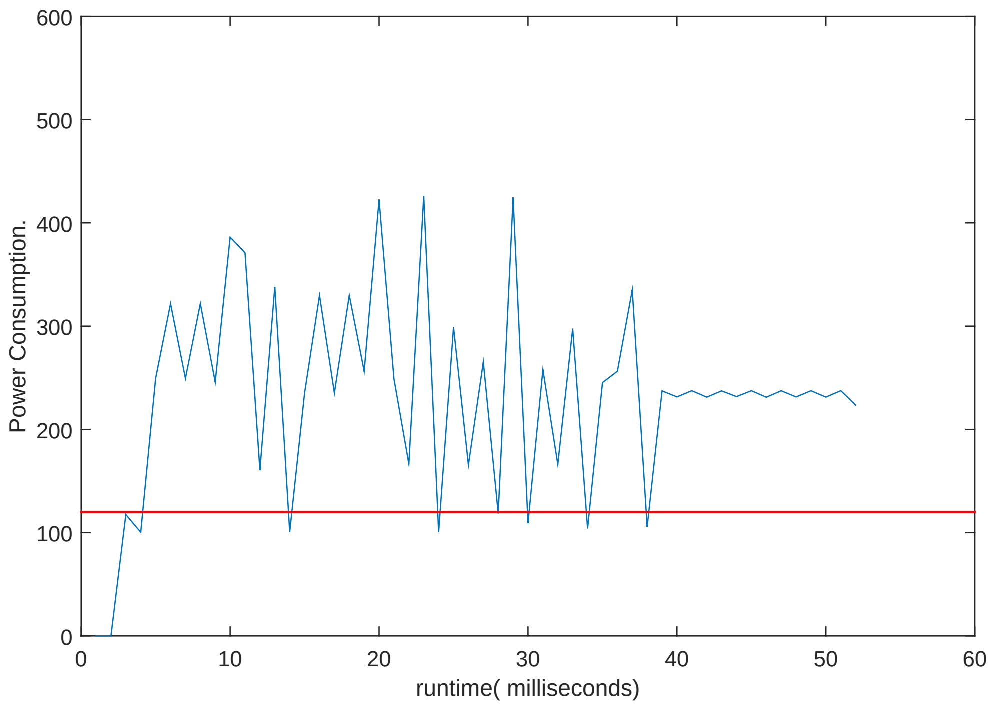


Figure 3.32: Bodytrack 32 active cores Power Consumption.

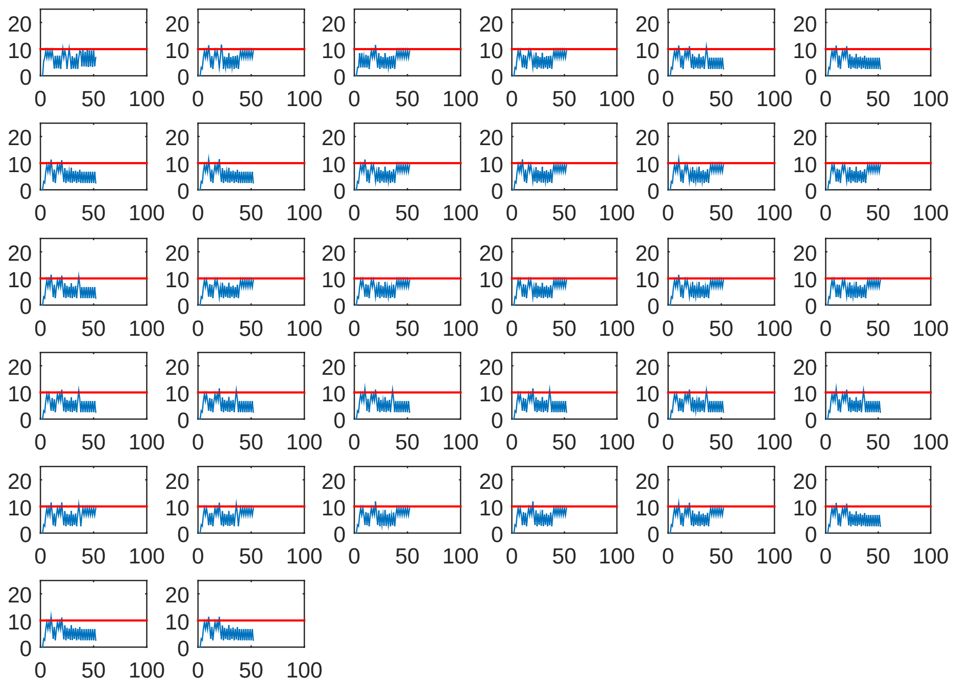


Figure 3.33: Bodytrack 32+1 active cores + master core.

3.3 Global Manager Analysis

The Global Manager as described before follows the steps below:

- Define the chip power budget based on the active cores number.
- Perform power allocation to the active cores based on their criticality
- V-F setting based on the correlation between the frequency levels and the power budget that has been assigned to each core.

Furthermore we analyze the behavior of the Global Manager for the 3 benchmarks compared with
a) The basic execution of the benchmark when all the cores run at 1.8 GHz.
b)The Individual Manager.
As shown in the figures below the overall chip power consumption is well below the threshold and rarely commits overbudgeting. The reason behind the gap between the chip power consumption and the chip power budget is the specified DVFS levels. To further explain , since the manager has a distinct number of different frequencies that

can apply on each core , there is also a certain spectrum of power consumption values that one core can satisfy. As a result when the total chip power budget is allocated to all the active cores , each core individually can reach up to a certain point of that budget. In the case that this individual budget can be reached on a satisfying level , then we have the phenomenon of the decreasing gap between the chip power budget and the total chip power consumption.

However when that is not the case then the gap between those values translantes into the non existance of an optimal frequency level to satisfy the budget limitations.

Furthermore we can see the **Power Savings** and the **Performance Degration** of the 3 benchmarks. for the different configurations in the 3 following tables.

The relationship used to compute them is the following , where X is either of the 2 metrics. :

$$\frac{mean(X_{normal}) - mean(X_{globalmanager})}{mean(X_{normal})} \tag{3.1}$$

Below we describe the effect of the Global Manager on all 5 different configurations of active cores , and the impact the manager has on the performance and the power consumption of the chip.

3.3.1 Blackscholes

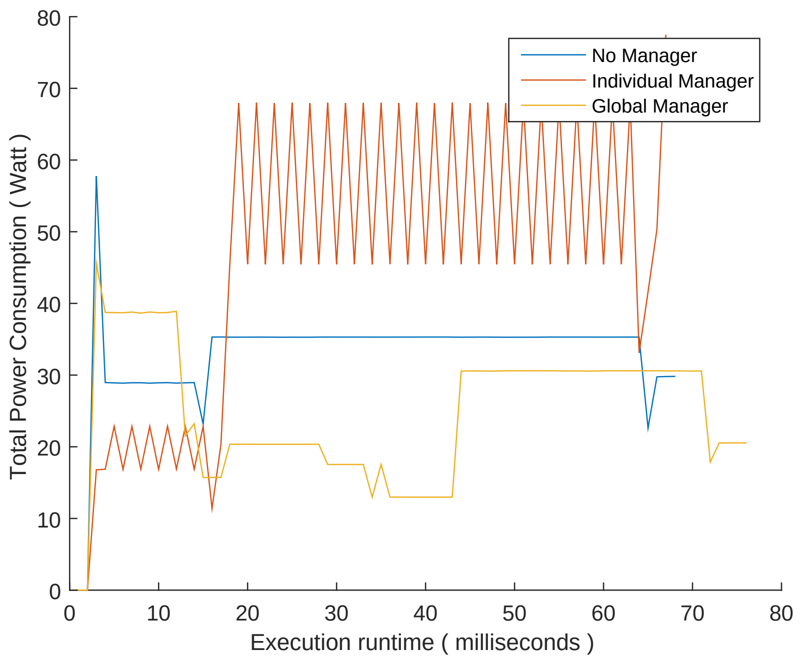


FIGURE 3.34: Blackscholes 2 active cores . Technique Comparison.

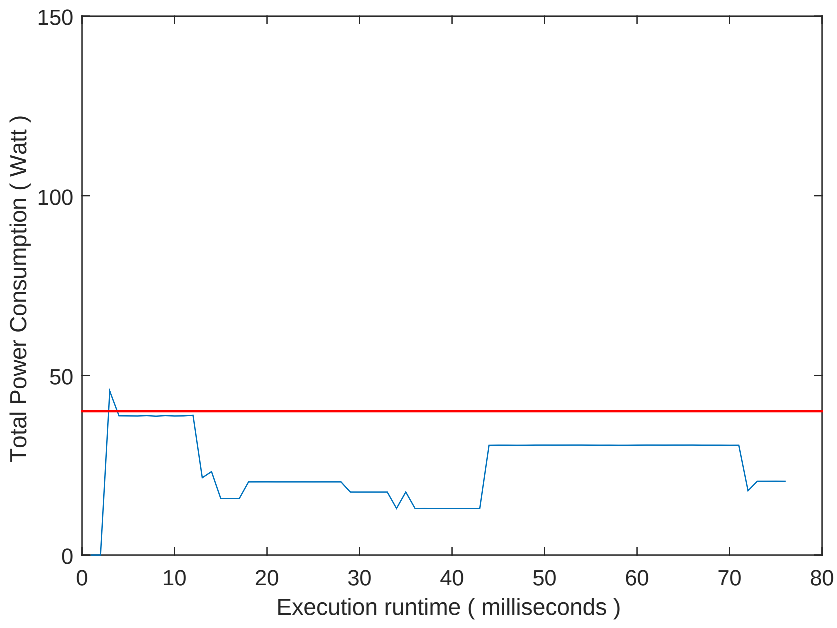


FIGURE 3.35: Blackscholes 2 active cores + master core.

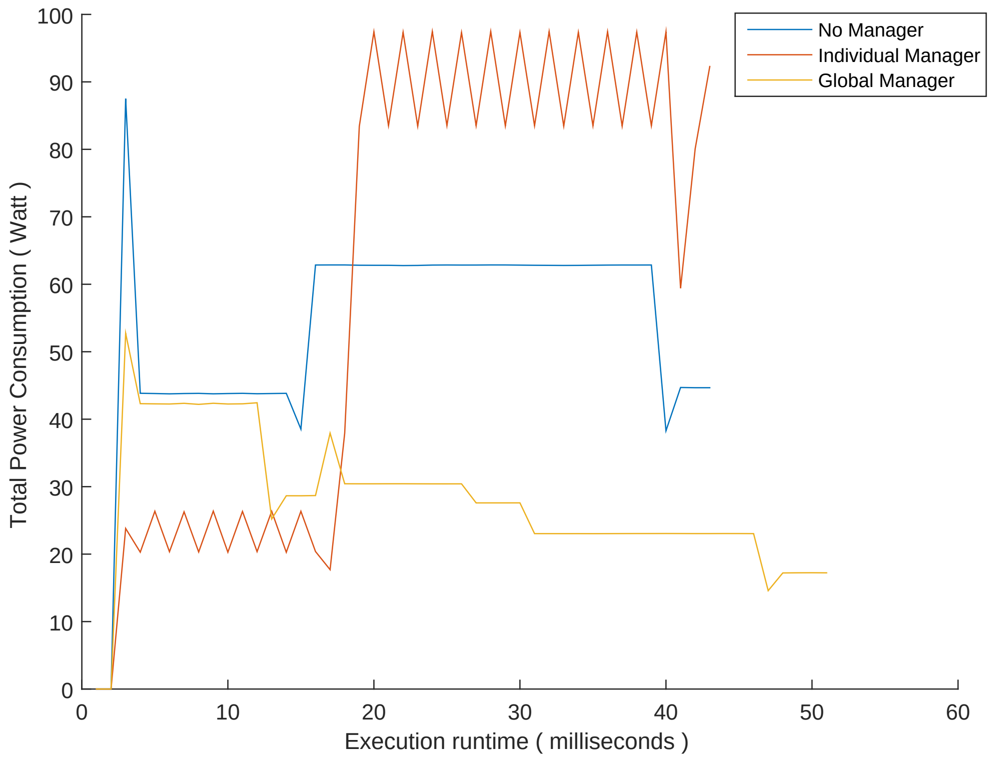


FIGURE 3.36: Blackscholes 4 active cores . Technique Comparison.

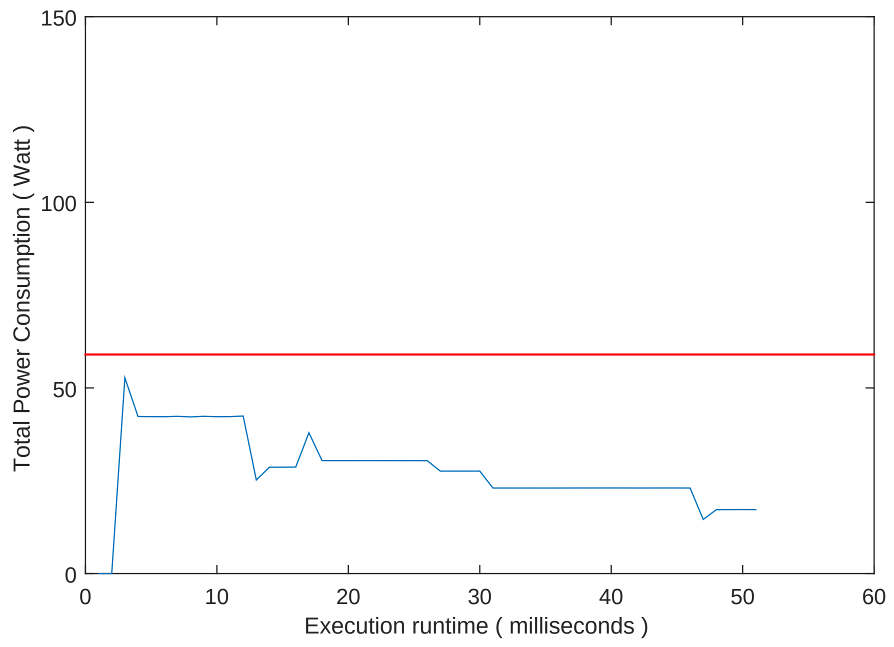


FIGURE 3.37: Blackscholes 4 active cores + master core.

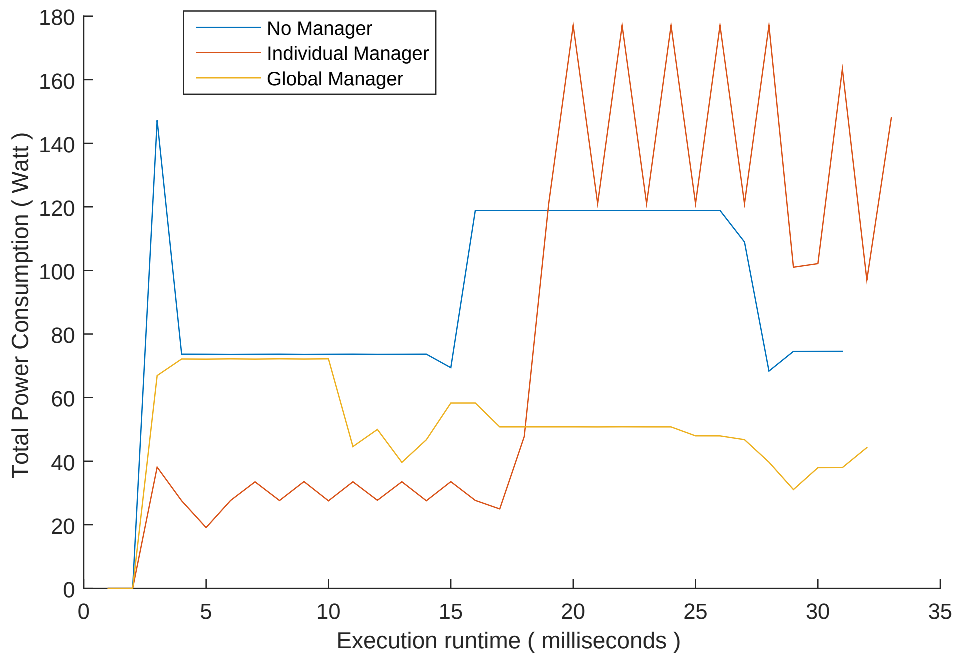


FIGURE 3.38: Blackscholes 8 active cores . Technique Comparison.

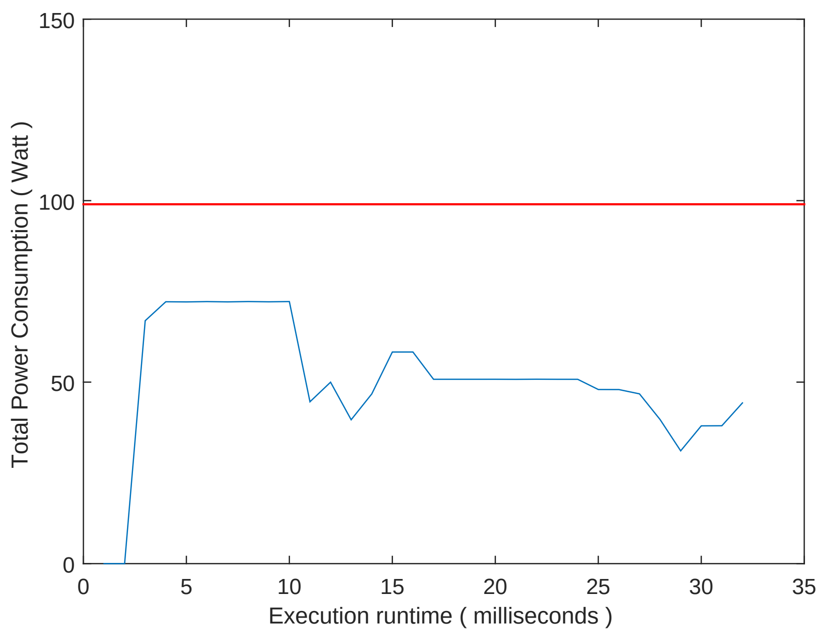


FIGURE 3.39: Blackscholes 8 active cores + master core.

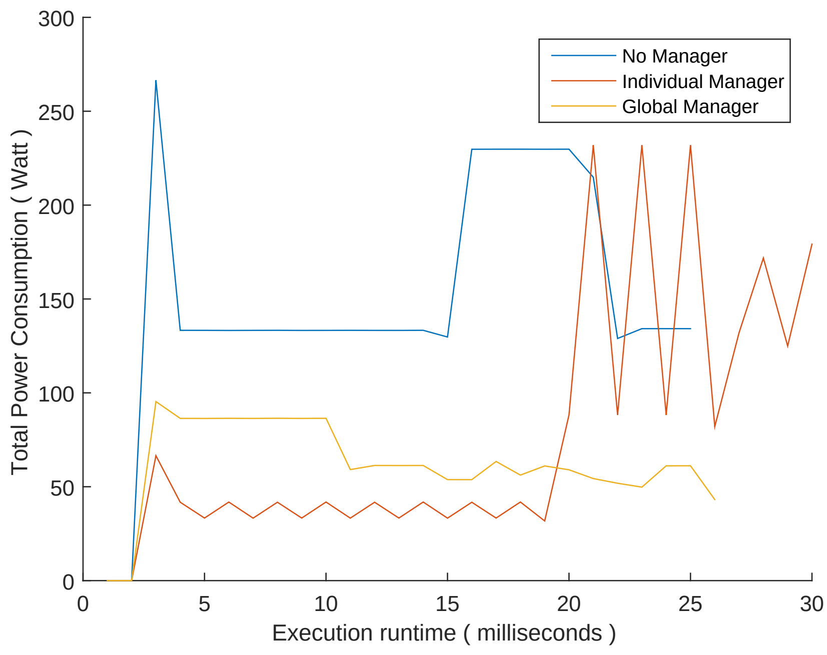


FIGURE 3.40: Blackscholes 16 active cores . Technique Comparison.

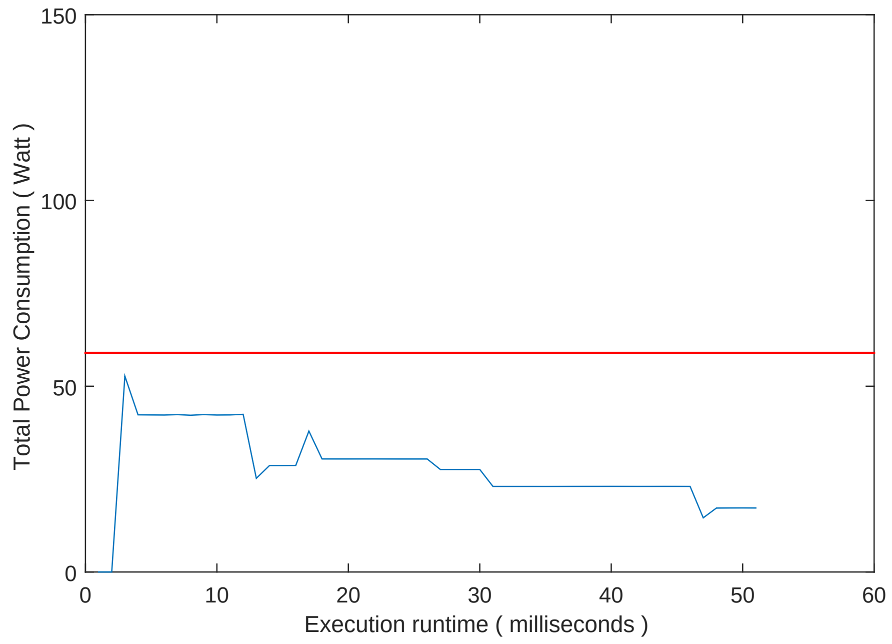


Figure 3.41: Blackscholes 16 active cores + master core.

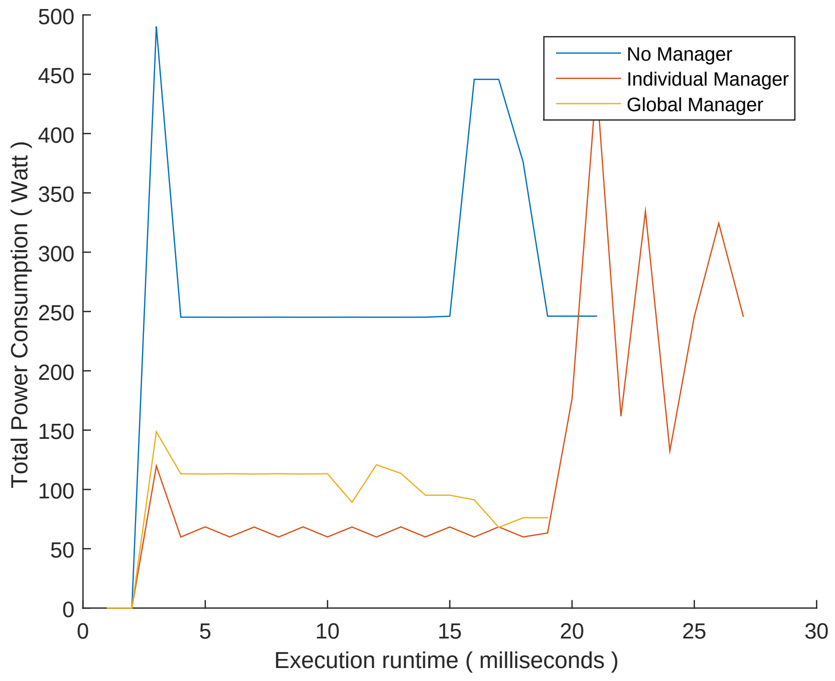


Figure 3.42: Blackscholes 32 active cores . Technique Comparison.

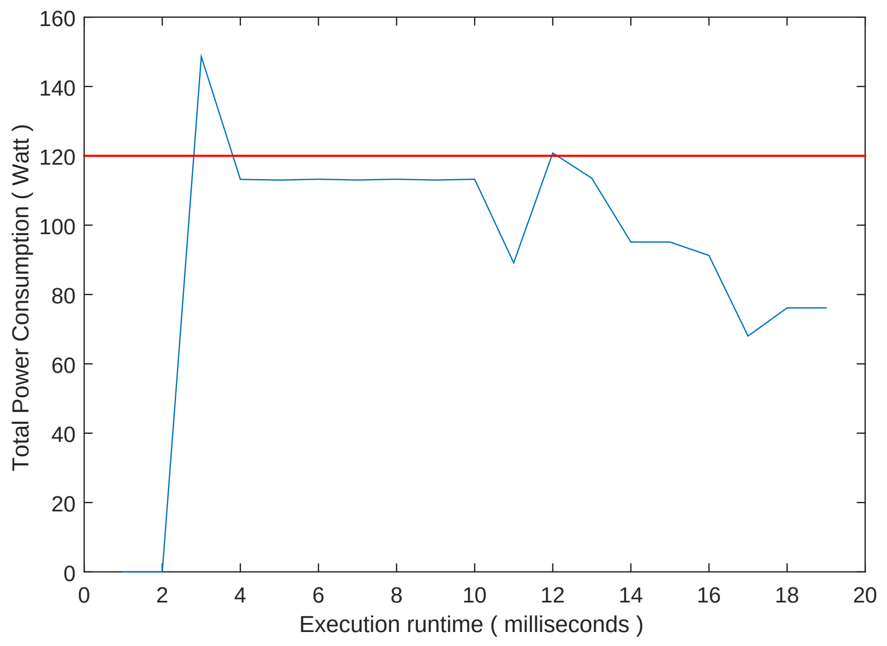


FIGURE 3.43: Blackscholes 32 active cores + master core.

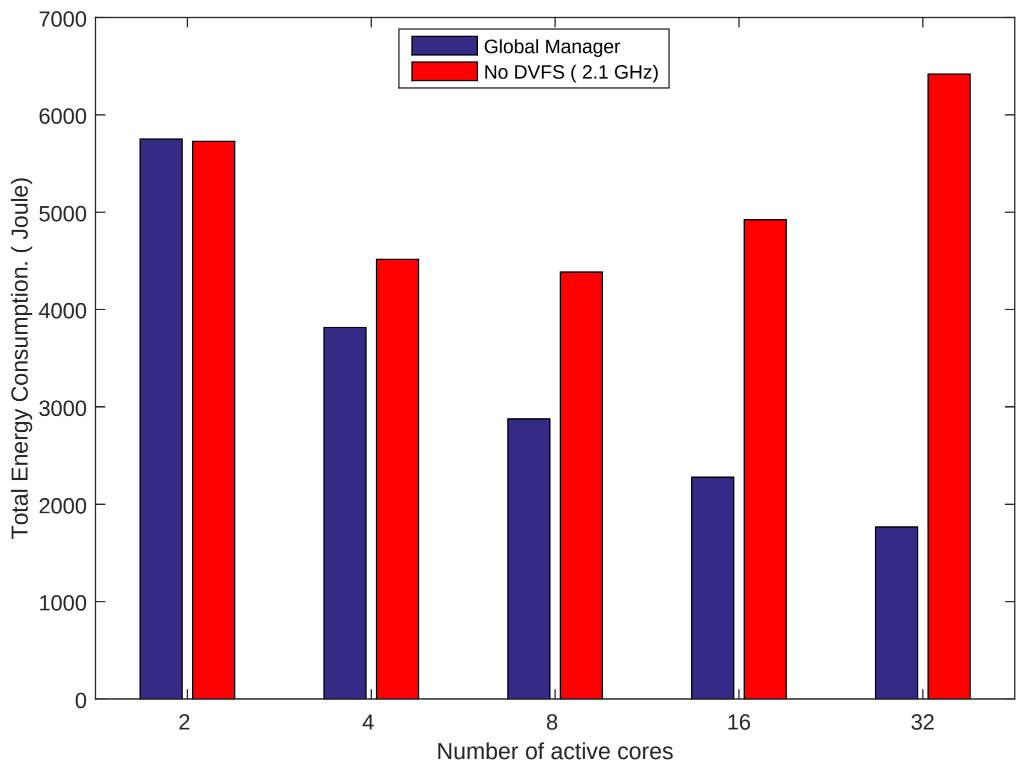


FIGURE 3.44: Total Energy Consumption.

Number of active cores	**Power Saving**	**Performance Degration**
2	24.5 %	10.5 %
4	47.7 %	15.6 %
8	42.9 %	3.12 %
16	58.7 %	3.84 %
32	64.1 %	0 %

As we can see in the figures above , overbudgeting occurs only in the case of the 32 active cores at the start of the simulation. For the rest of the configurations the global manager keeps the total chip power consumption below the threshold on satisfying levels.
As we can see below the global manager maintains the chip power consumption below the power budget limit. Over budgeting occurs only a few times which appear to happen at the beggining of the simulation and are a result of the high burst of throughput by spawning threads.
From the provided table the power savings increase as the number of the active cores increases however the performance degration does not increase accordingly. This can be explained by the fact that our implementation manages to perform a power allocation that maximizes the performance and that the blackscholes benchmark is a combination of low lock-behavior and low workload. As a result the increasing number of active cores without the proper allocation of the power budget to certain cores does not translate into higher throughput. The global manager is capable of this proper allocation and that is why the performance degration reaches these minimum values.
Finally the Figure 3.44 shows that the total energy consumption of the chip using the Global Manager decreases greatly as the number of cores increases.

3.3.2 Bodytrack

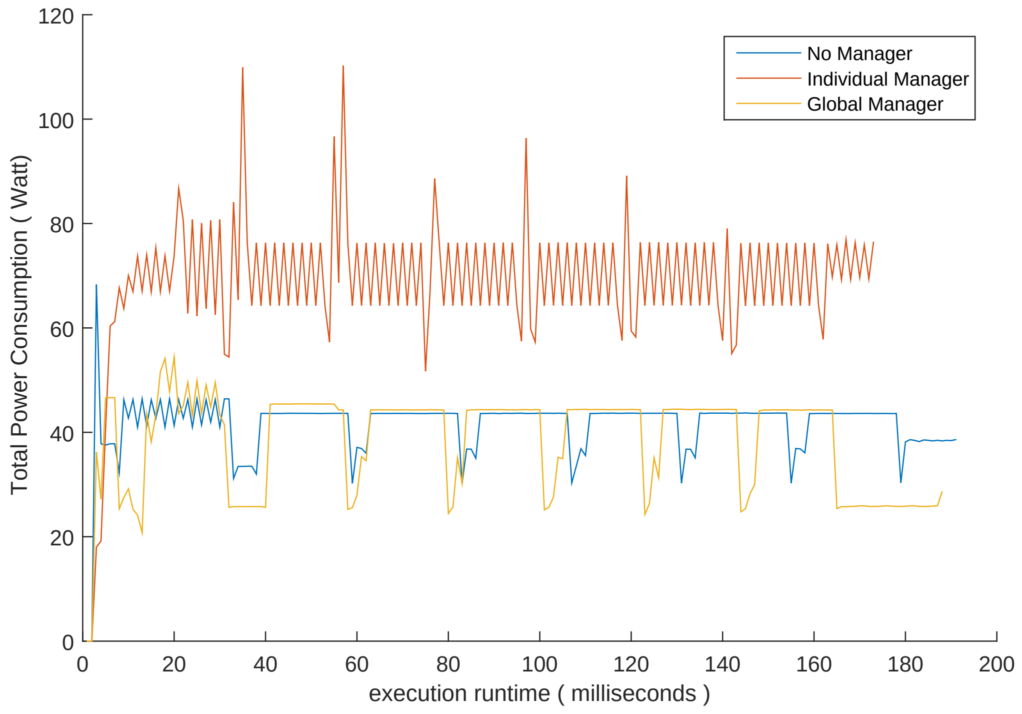


FIGURE 3.45: Bodytrack 2 active cores . Technique Comparison.

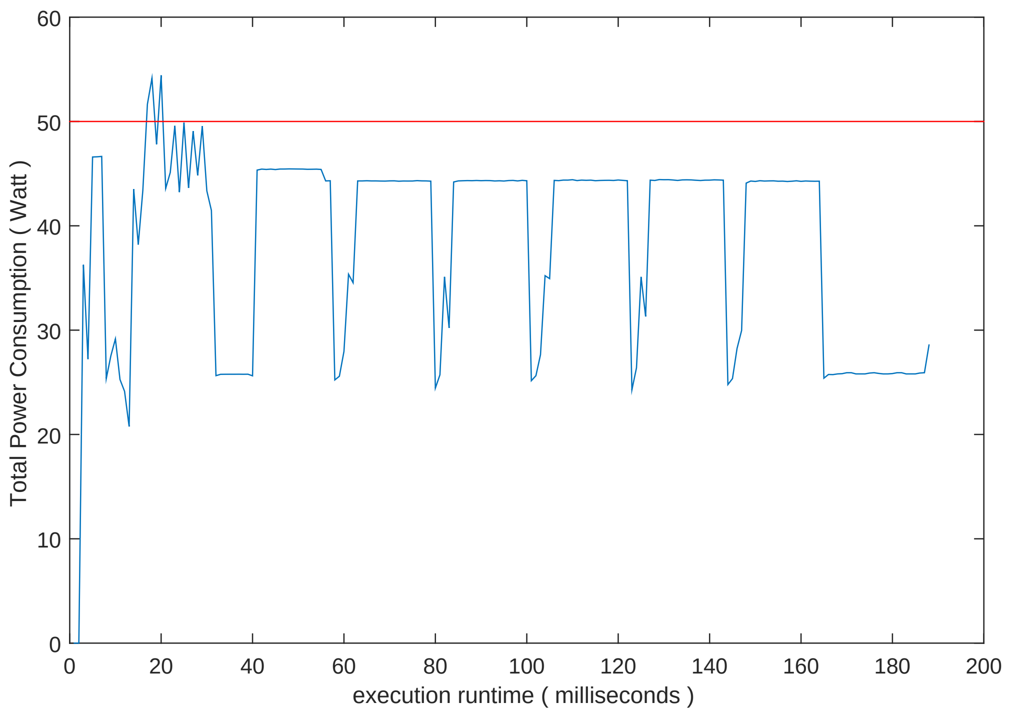


FIGURE 3.46: Bodytrack 2 active cores + master core.

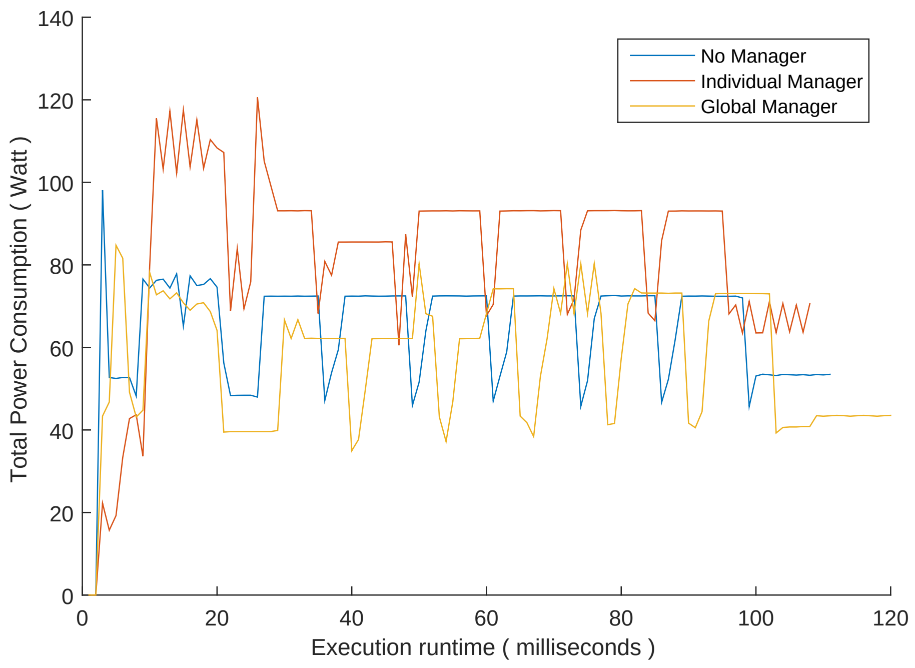


FIGURE 3.47: Bodytrack 4 active cores . Technique Comparison.

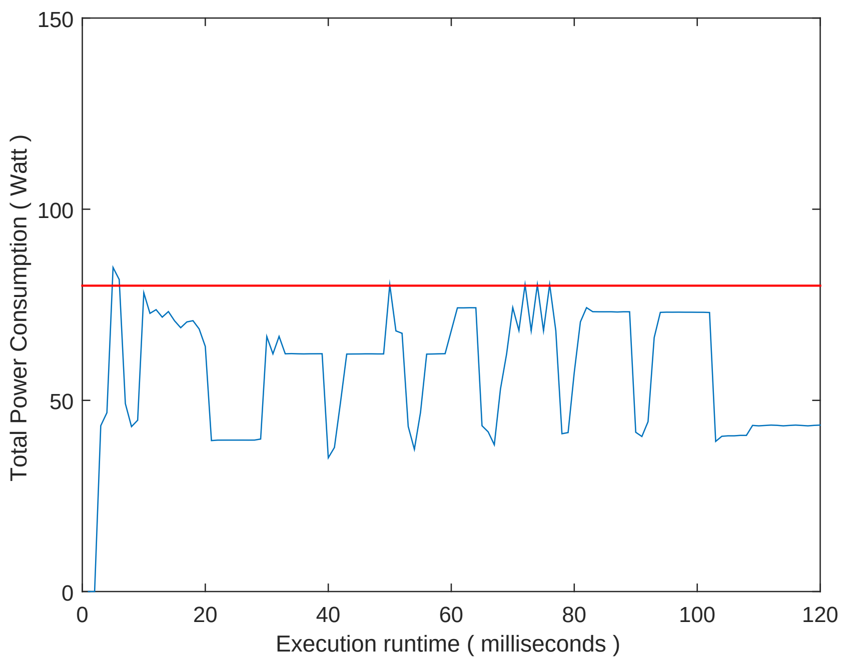


FIGURE 3.48: Bodytrack 4 active cores + master core.

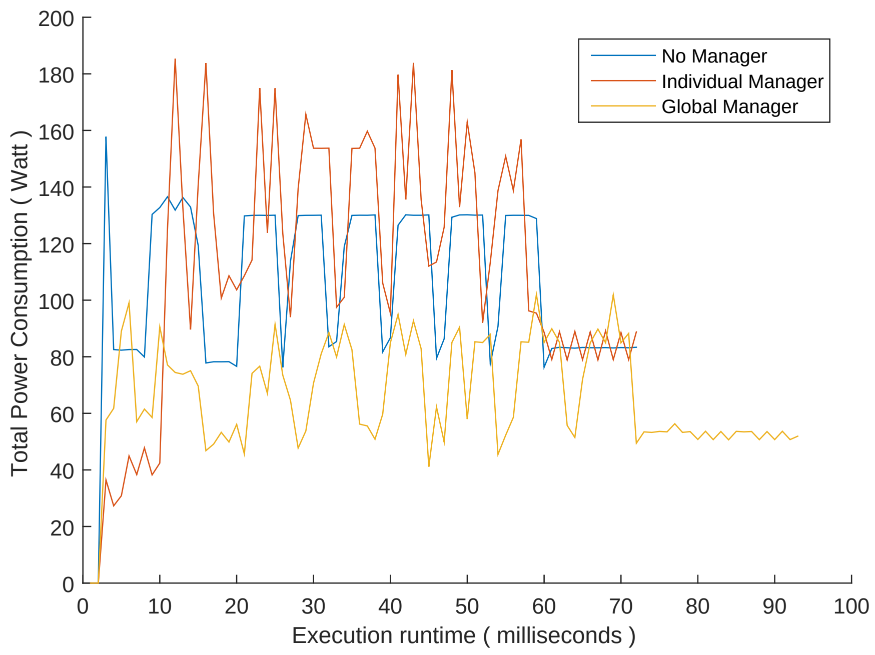


FIGURE 3.49: Bodytrack 8 active cores . Technique Comparison.

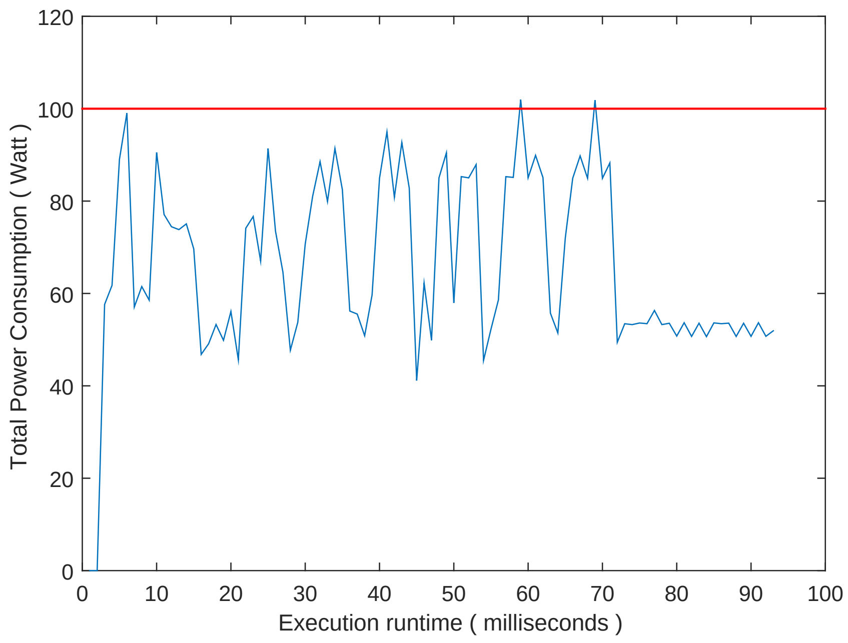


FIGURE 3.50: Bodytrack 8 active cores + master core.

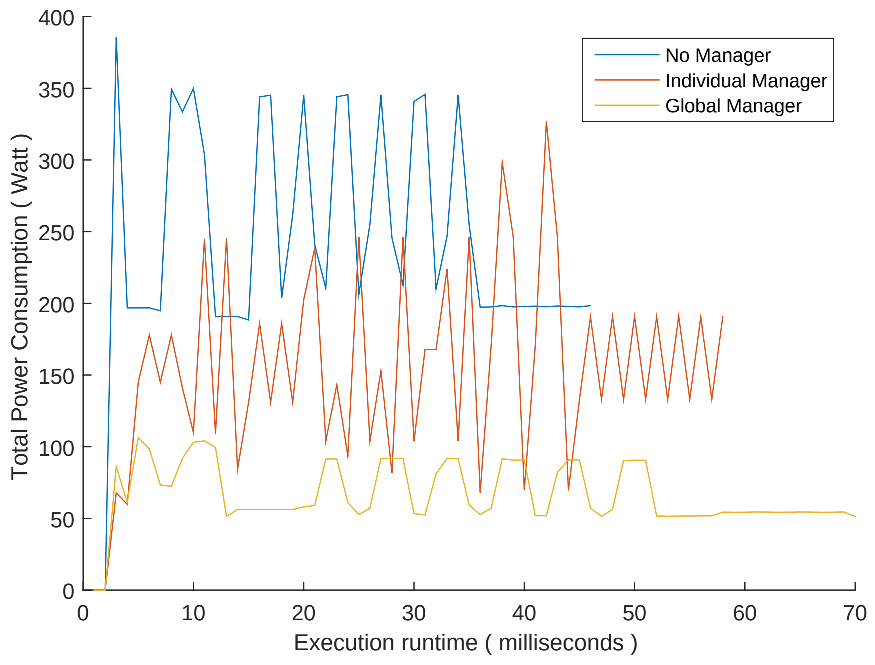


Figure 3.51: Bodytrack 16 active cores . Technique Comparison.

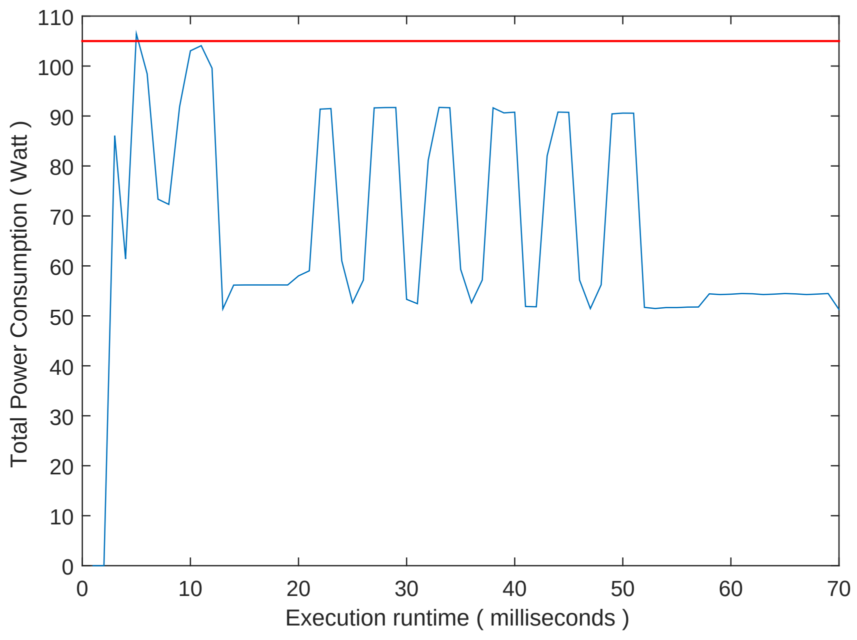


Figure 3.52: Bodytrack 16 active cores + master core.

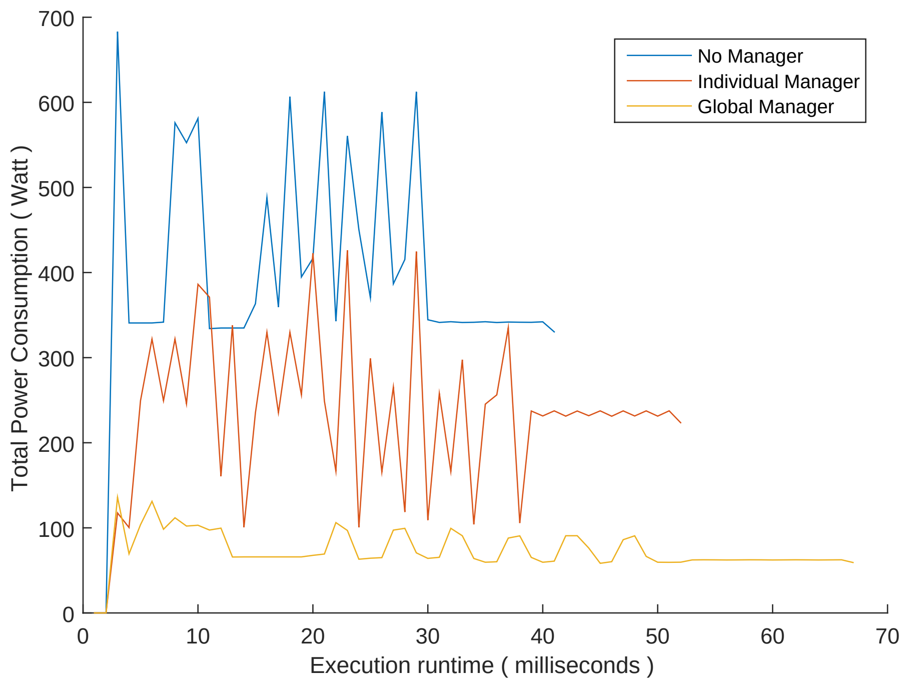


FIGURE 3.53: Bodytrack 32 active cores . Technique Comparison.

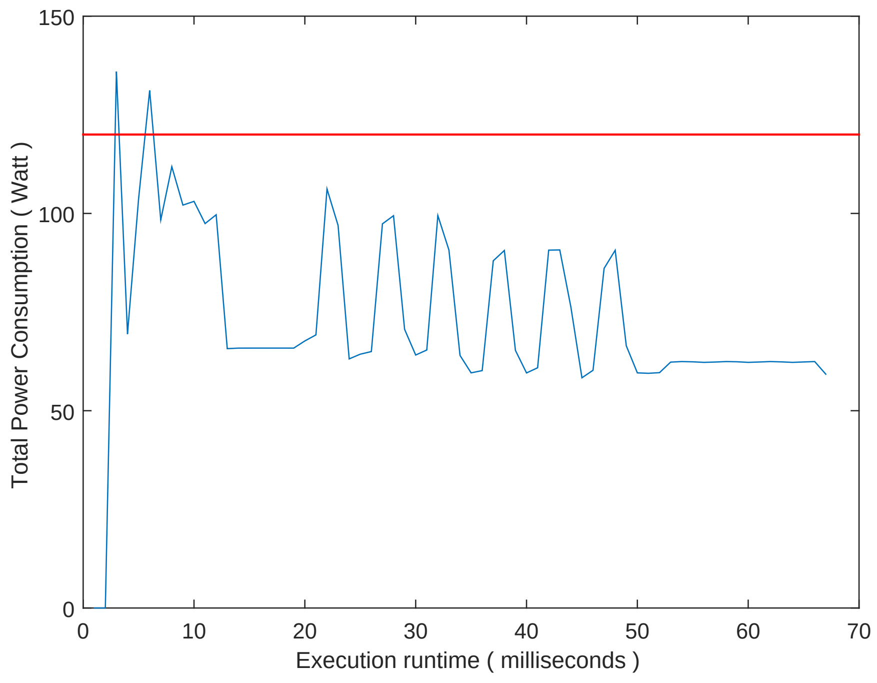


FIGURE 3.54: Bodytrack 32 active cores + master core.

Number of active cores	**Power Saving**	**Performance Degration**
2	33.9 %	11.7 %
4	36.7 %	20.0 %
8	61.7 %	25.7 %
16	61.4 %	24.2 %
32	73.9 %	29.8 %

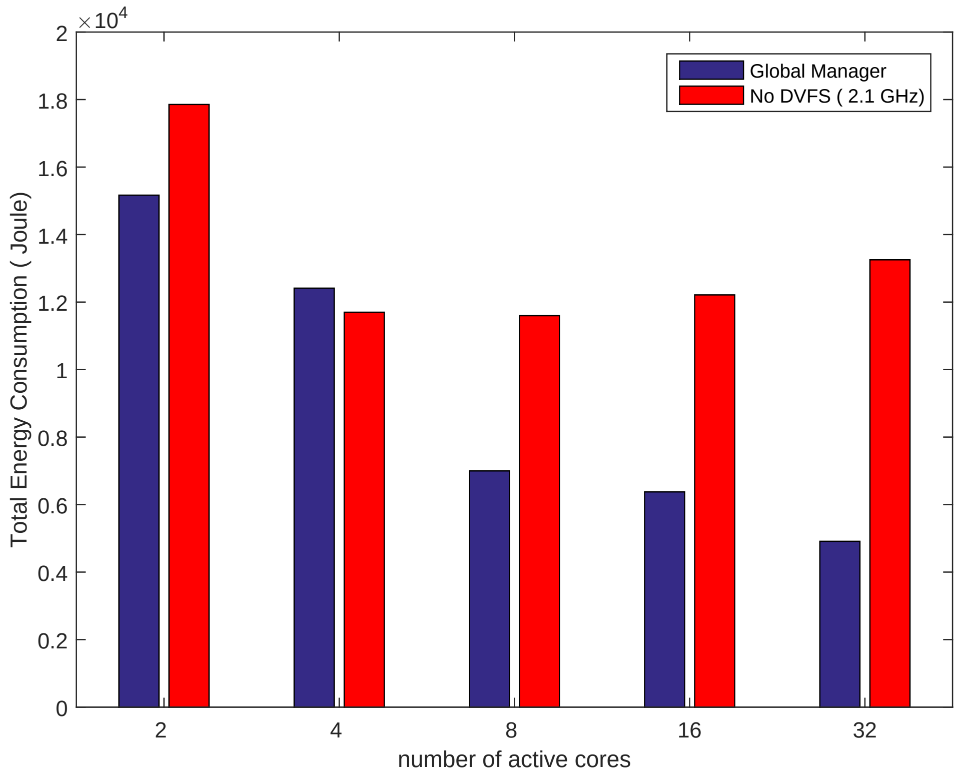


FIGURE 3.55: Total Energy Consumption.

The overall performance degration does not exceed 30% since the high frequency of the base case , that is compared with the global manager , does not translate into higher throughput.However the high-lock behavior limits the global manager from reaching the low performance degration values that were reaches for blackscholes. On the power consumption level as we can see again the global manager satisfies all the power budget limitations with a few expections and the power savings increase with the increase of the active cores as expected.

Finally from the figure 3.55 we can see that there is an increasing difference between the total energy consumption of a system without the global manager and of a system monitored by him.

3.3.3 Swaptions

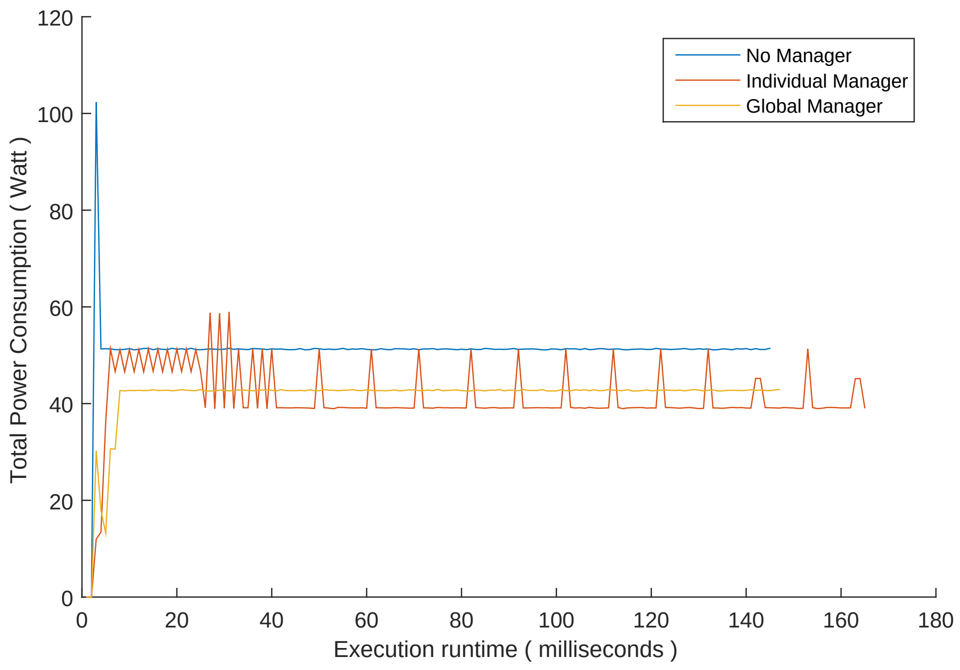


FIGURE 3.56: Swaptions 2 active cores . Technique Comparison.

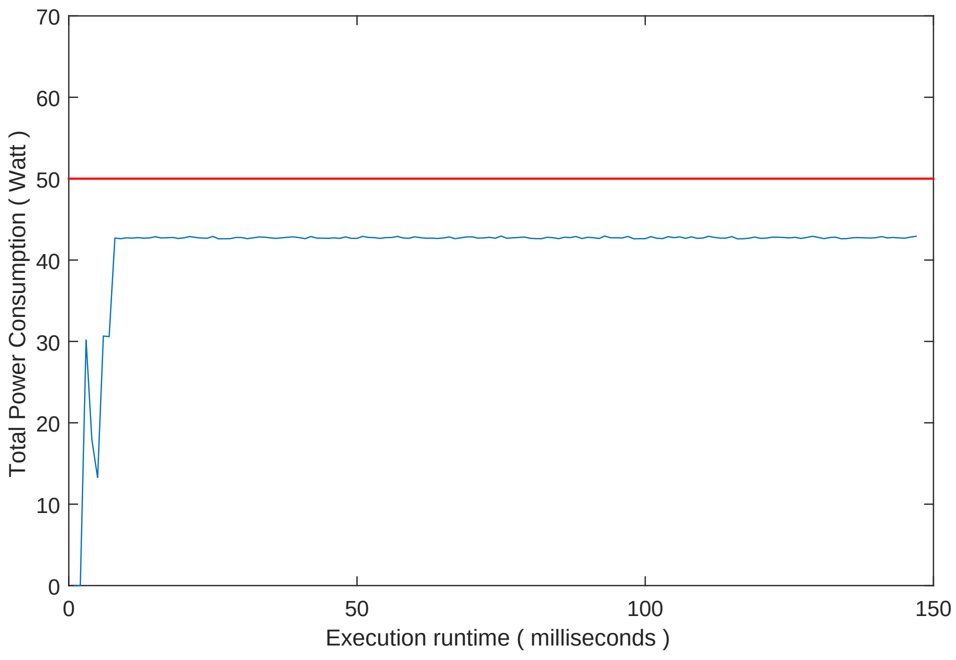


FIGURE 3.57: Swaptions 2 active cores + master core.

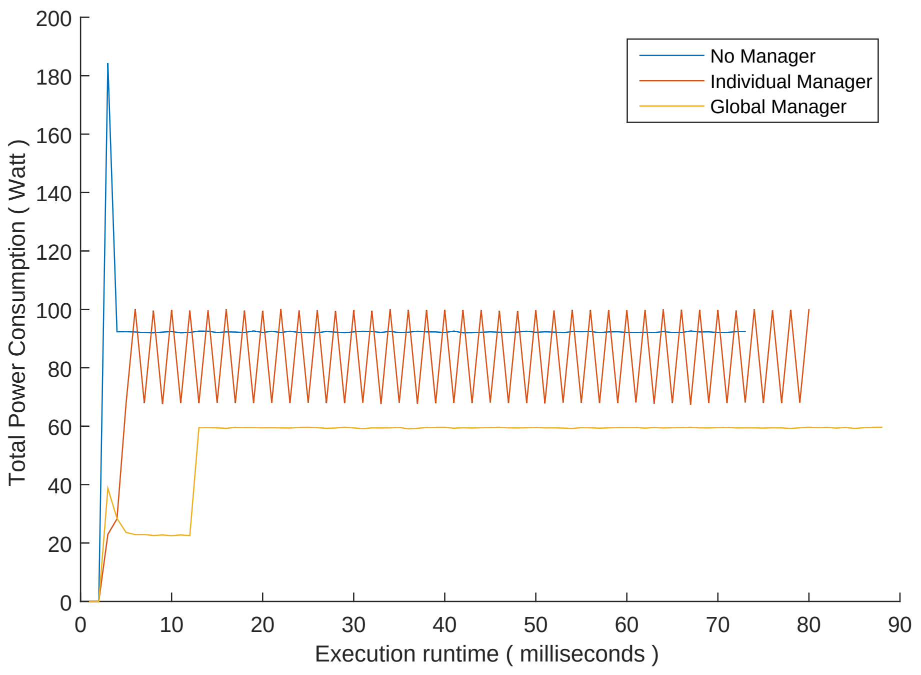


FIGURE 3.58: Swaptions 4 active cores . Technique Comparison.

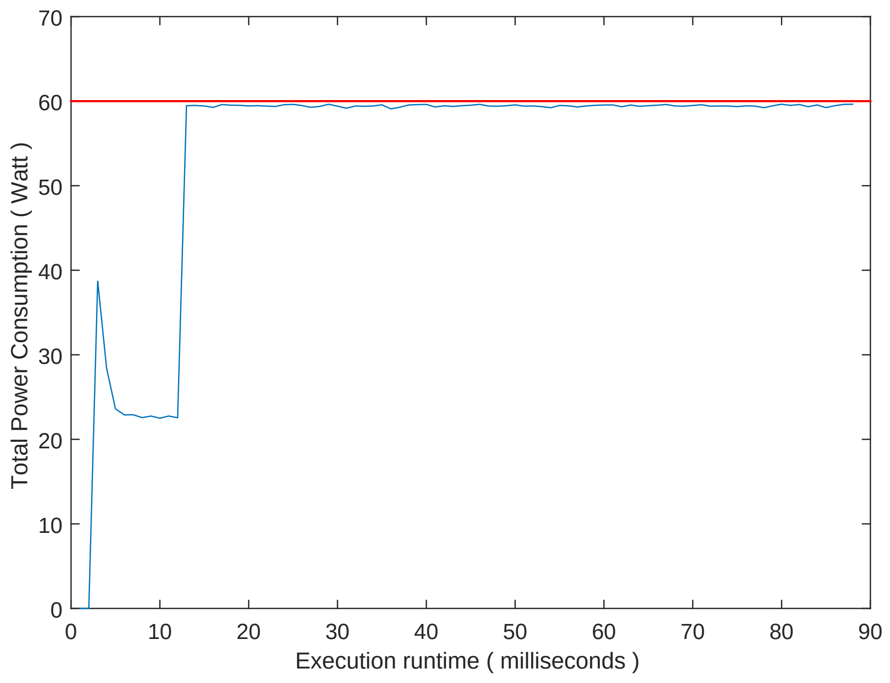


FIGURE 3.59: Swaptions 4 active cores + master core.

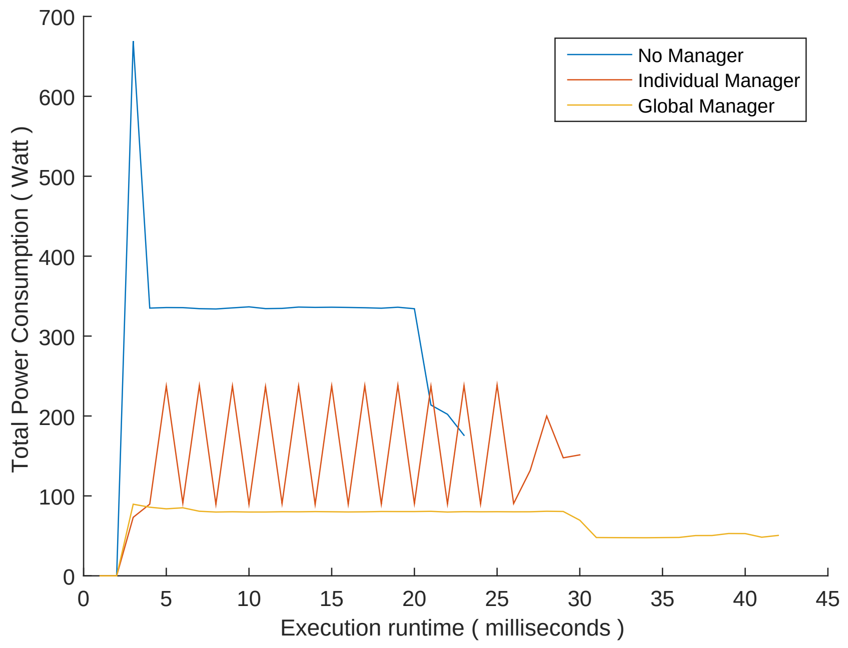


FIGURE 3.60: Swaptions 16 active cores . Technique Comparison.

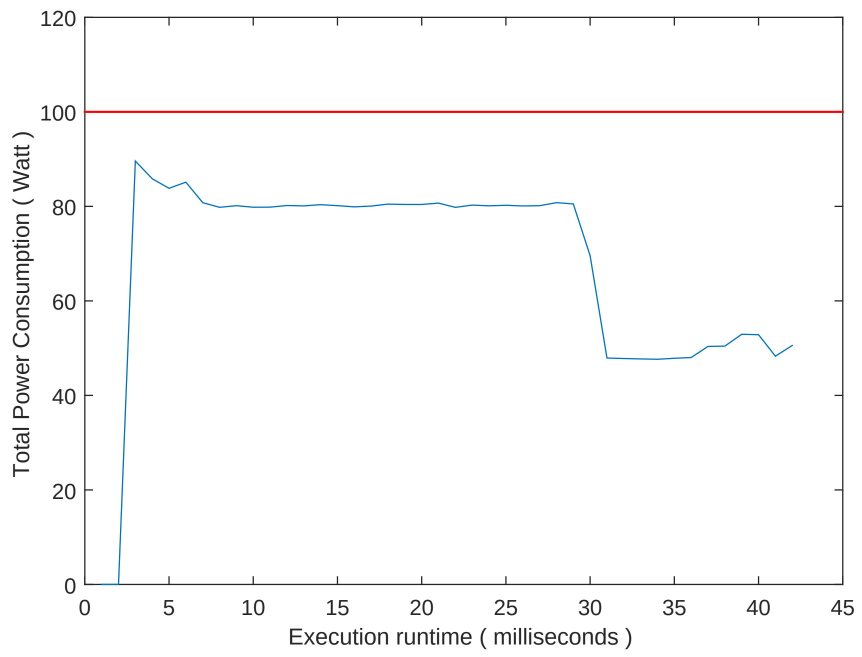


FIGURE 3.61: Swaptions 16 active cores + master core.

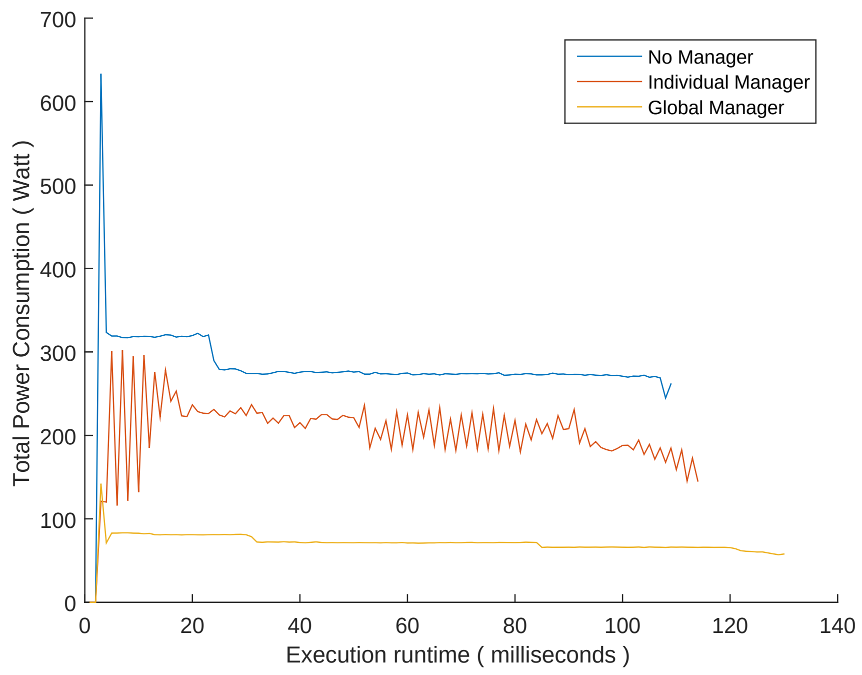


FIGURE 3.62: Swaptions 32 active cores . Technique Comparison.

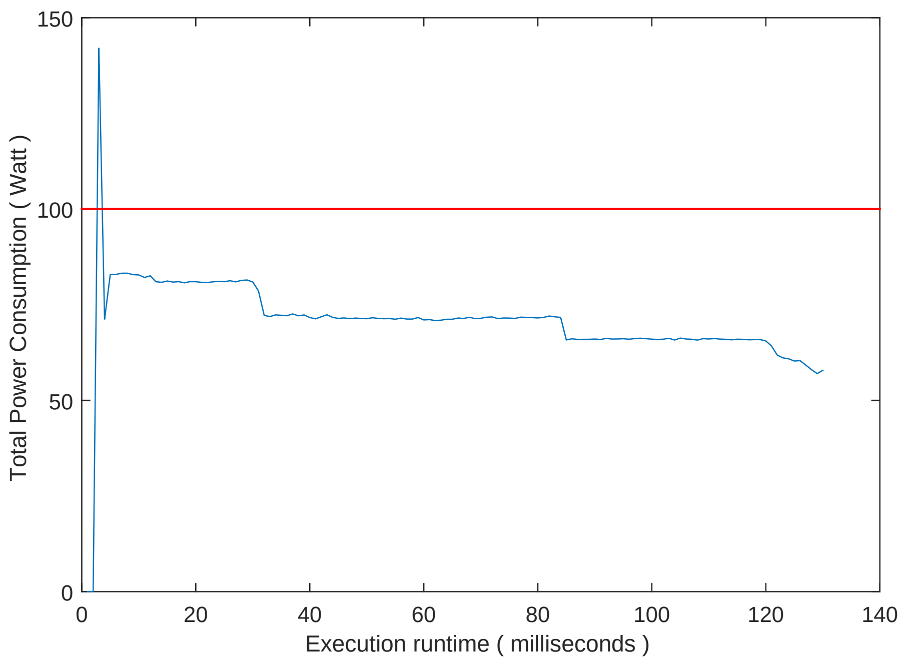


FIGURE 3.63: Swaptions 32 active cores + master core.

Number of active cores	Power Saving	Performance Degration
2	18.3 %	1.36 %
4	40.4 %	17.0 %
8	75.0%	33.9 %
16	77.5 %	38.09 %
32	74.7 %	16.1 %

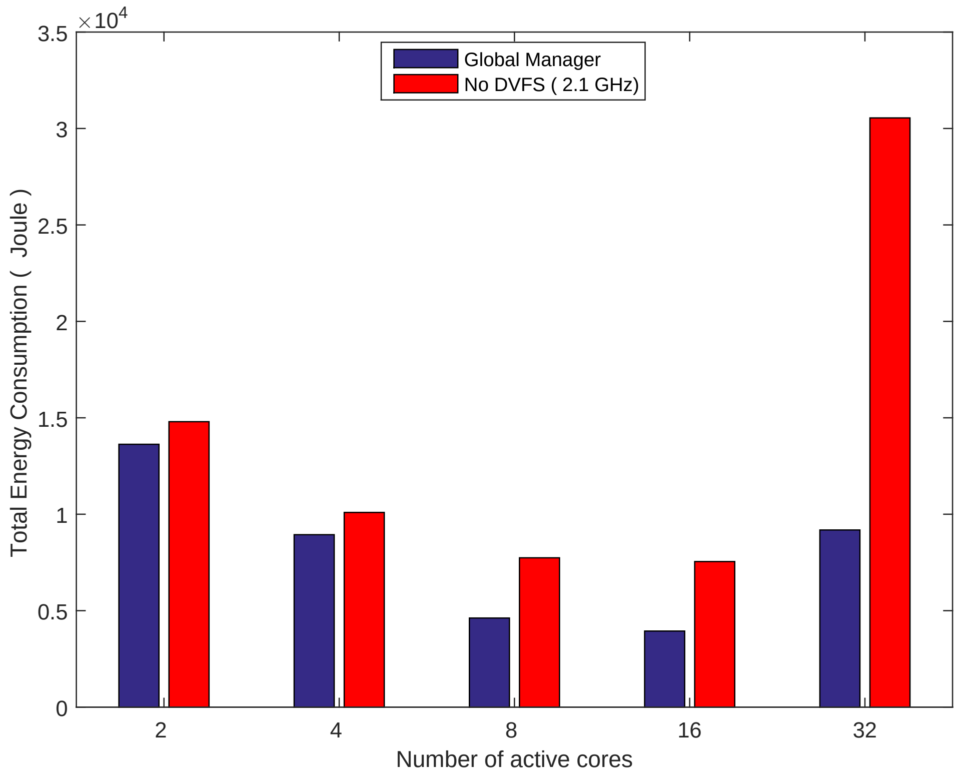


FIGURE 3.64: Total Energy Consumption.

As displayed in the previous figures the global manager keeps the total chip power consumption below the chip power budget for all the configurations. Since swaptions is a non blocking benchmark and has straigthforward implementation we can see that the total chip power consumption is more stable and the distance from the total chip budget is shorther than the previous benchmarks.For swaptions , the performance degration is greater and reaches a maximum of 39% with the advantage of high power savings that reach a maximum of 77%. This is also a result of the non-blocking behavior of this benchmark since the increase of the active cores has a direct impact on the performance. For the swaption benchmarks figure 3.64 shows us that the use of the global manager results in a great decrease in the total energy consumption of the chip especially as the number of active cores increases.

3.3.4 Cross Validation

Our predictor system has been trained using the simulation results and core values of the 3 benchmarks :

- Blackscholes
- Bodytrack
- Swaptions.

However it is valuable to see the behavior of the global manager against benchmarks that present unknown behavior and evaluate the manager's performance.The information about the PARSEC-Suite benchmarks was provided by [1]

x264

x264 is media processing application that is characterized by a medium working set and a high sharing and exchange data behavior. This is an application similar to bodytrack but with a more heavy lock-behavior. We simulated the x264 benchmark using the *development* input size , on a configuration of 32 cores.
From the figure belove we can see the behavior of the system is satisfying. The manager succeed in controlling the chip power consumption with a few expections of overbudgeting. We can also observe the consumption reaches values close to the budget which translates into a good power alocation.

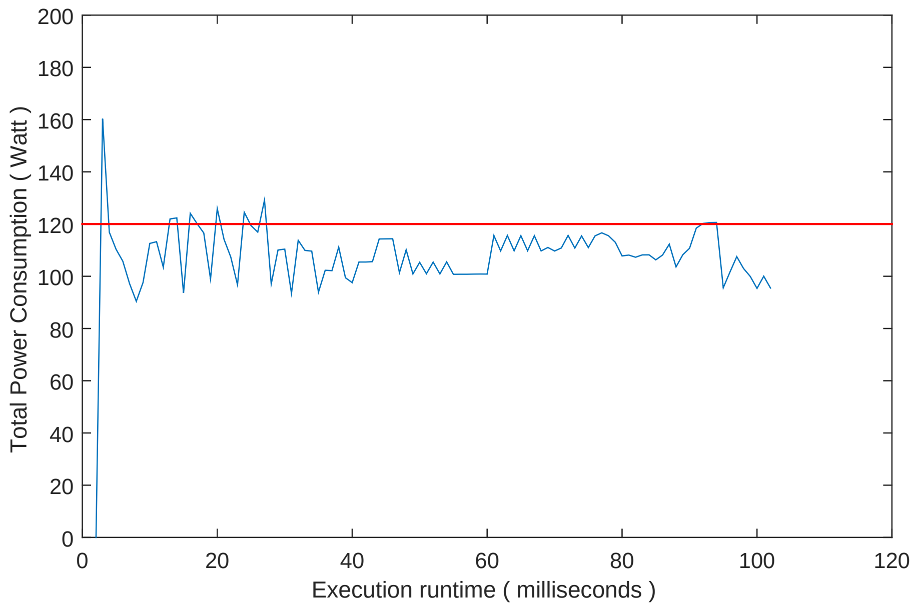


FIGURE 3.65: x264 32 active cores + master core.

Below we can also see the performance degration and power saving compared to the execution of benchmark x264 on a configuration of 32 cores at 2.1GHz

Power Saving	**Performance Degration**
67.3 %	10.5 %

Fluidanimate

Fluidanimate is an animation application with a large working set and a low sharing behavior. We simulated the fluidanimate benchmark using the *development* input size , on a configuration of 32 cores

As we can see the chip power consumption exceeds the provided budget only at the start of the simulation and then stays well below the threshold. As a result in a case of a benchmark the combination of high workload with a lock-behavior the global manager's performance is satisfying.

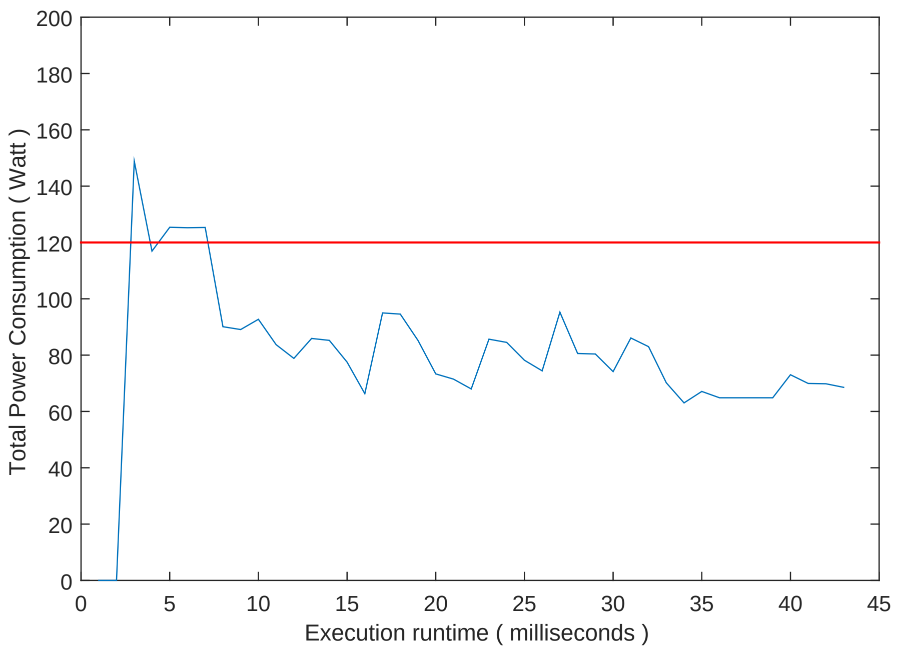


FIGURE 3.66: Fluidanimate 32 active cores + master core.

Below we can also see the performance degration and power saving compared to the execution of benchmark fluidanimate on a configuration of 32 cores at 2.1GHz

Power Saving	**Performance Degration**
78.9 %	36.2 %

Facesim

Facesim is also an animation application with the same characteristics as fluidanimate. That is , a large working set and a low sharing behavior.We simulated the fluidanimate benchmark using the *development* input size , on a configuration of 32 cores
As we can see the chip power consumption exceeds again the budget only at the start of the simulation and then stayed on a satisfying level below the chip power budget.

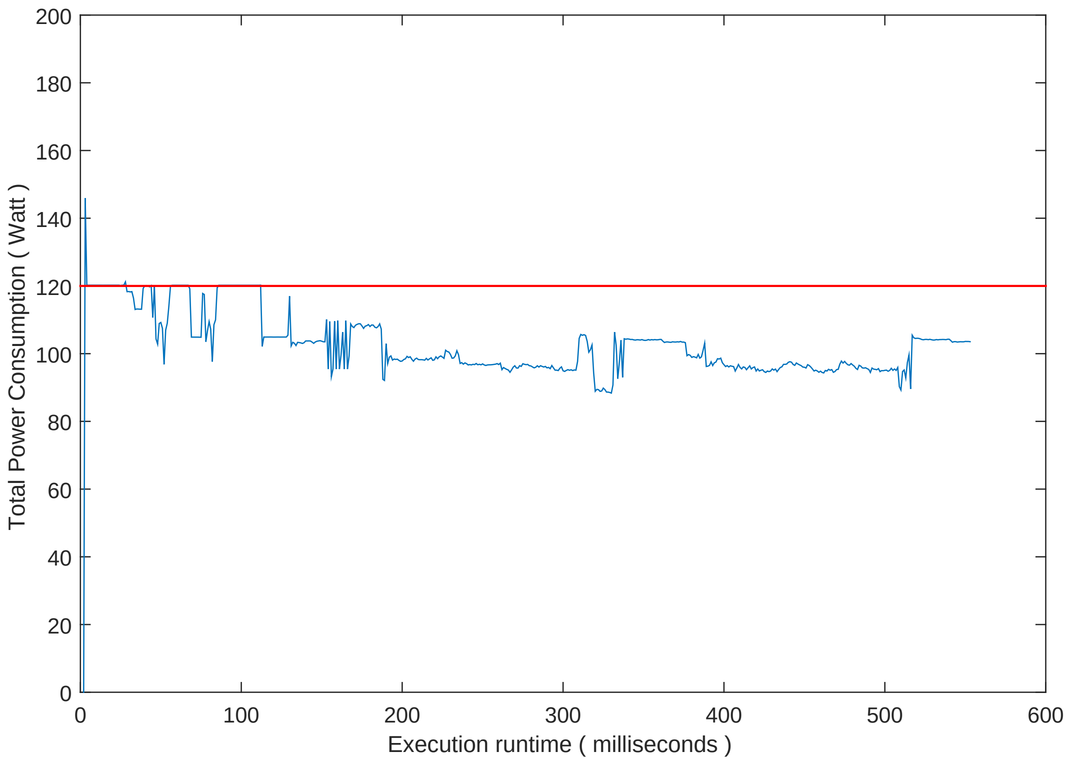


FIGURE 3.67: Facesim 32 active cores + master core.

Power Saving	Performance Degration
79.12 %	0 %

Vips

Vips is a media processing application that is characterized by a medium working set low sharing behavior. Vips is similar to fluidanimate but with a smaller working set we expect the chip power consumption to be closer to the overall budget.Indeed the global manager succeed in controlling the chip power consumption and fully exploit the provided budget.

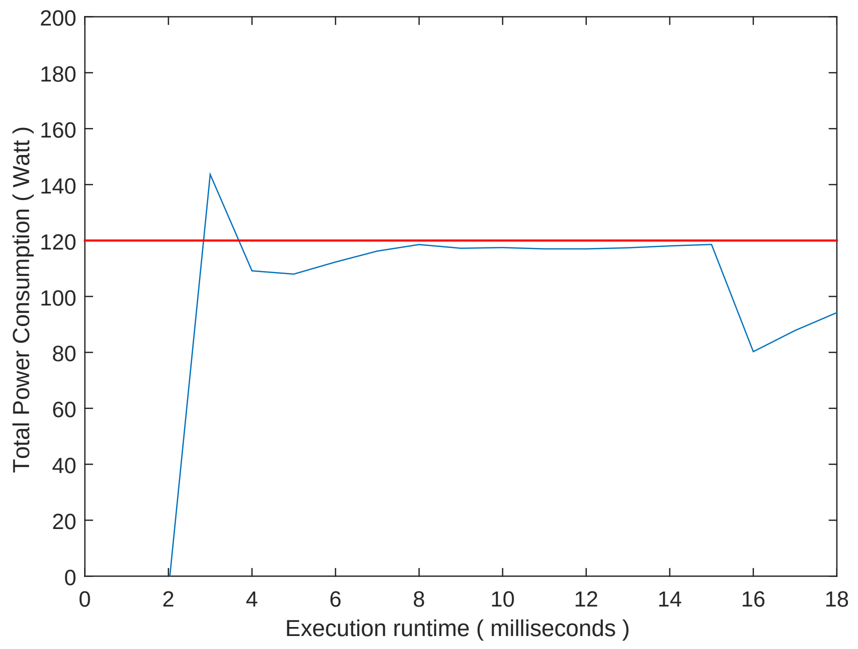


FIGURE 3.68: Vips 32 active cores + master core.

Power Saving	Performance Degration
75.2 %	5.5 %

Chapter 4

Conclusion.

In this diploma thesis the structure and stategy of a power allocation managment system was described. We described the components that were implemented and are essential to the various actions the manager performs. Furthermore we analyzed the performance of the different manager systems on various configurations and new benchmarks, specifically:

The **Individual Manager** that was created performed well for configurations with a low number of cores, for these cases the chip power consumption remained below a normal power level . However as the number of cores increased the Individual Manager did not take into consideration the chip power consumption but only monitored the per core power consumption failed to maintain below a normal power level. For all the configurations the per core power consumption stayed below the power threshold that was applied by the manager and there was rarely a phenomenon of core over budgeting.

The **Global Manager** showed promising results with a high power saving and a relatively small performance degration. The Global Manager maintained the chip power consumption below the provided chip power budget with rare cases of overbudgeting which is a very important factor. Pending on the behavior of the application the global manager was able to exploit the power budget at it's capacity . As displayed the Global Manager can adapt very well to different number of cores with a correspoding different power budget. Furthermore through the Cross validation it was proved that the Global Manager can perform very well controlling application that are unknown to him , since no data from such applications where used for his training. For these cases the Global Manager is able to maintain the chip power consumption below the set budget quite successfully.

Future Work This diploma has provided a structure that can be easily modified in order to apply new policies or different type of managers. One possible expansion for the manager that was studied but not fully implemented is the dynamic increase and decrease of the active cores that are used by the running application. This new tool with the combination of DVFS may provide more promising results. Moreover the integration of many different benchmarks on a chip is a interesting topic for future work. Although the current Manager may be able to control the cores even if they belong to different applications, adding this knowledge to the system and possibly using a Cluster Manager per application's cores may provide promising results.

Bibliography

[1] *The PARSEC Benchmark Suite: Characterization and Architectural Implications.* Christian Bienia , Sanjeev Kumar , Jaswinder Pal Singh , Kai Li , Princeston University ,2008

[2] *TSocket : Thermanl Sustainable Power Budgeting.* Xing Hu , Yi Xu , Jun Ma , Guoqing Chen , Yu Hu , Yuan Xie , State Key Laboratory of Computer Architecture , Institute of Computing Technology.

[3] *Pack and Cap: Adaptive DVFS and Thread Packing Under Power Caps.* Ryan Cochran , Can Hankendi , Ayse K. Coskun , Sherief Reda , School of Engineering Brown University , ECE DEpartment Boston University.

[4] *An Analysis of Efficient Multi-Core Global Power Management Policies: Maximizing Performance for a Given Power Budget.* Canturk Isci , lper Buyuktosunoglu , Chen-Yong Cher , Pradip Bose , Margaret Martonosi , Princeston University

[5] *Scalable Power Control for Many-Core Architectures Running Multi-threaded Applications.* Kai Ma , Xue Li , Ming Chen , Xiaorui Wang , Department of Electrical and Computer Engineering The Ohio State University Columbus