Performance per watt analysis of Intel i7 processor

Aniket Kumar
cs20m002@iitp.ac.in
Ankit Kumar
cs20m004@iitp.ac.in
Jayanarayan Thakurdas Tudu
jtt@iittp.ac.in

ABSTRACT
In this paper, we describe our experiences in using a framework for power/ performance run-time management for the Intel core family. Here we are interested in finding performance per watt on intel icy-lake professor. We setup are simulation tools and done some experiment to get the insight visualisation, what we taught in classrooms. We visualise performance and efficiency trade-off, we work with cache and changes some hyper parameter to observe performance per watt. We are working with already existing tools and observe for improvement for subtle details in power and performance.

INTRODUCTION

We are trying to measure performance per watt in terms of power usage and elapsed time. We are mainly interested in finding dynamic power of the intel i7. For this we used Sniper simulator and Mcpat framework.

Sniper

Sniper is a next generation parallel, high-speed and accurate x86 simulator. This multi-core simulator is based on the interval core model and the graphite simulation infrastructure, allowing for fast and accurate simulation and for trading off simulation speed for accuracy to allow a range of flexible simulation options when exploring different homogeneous and heterogeneous multi-core architectures.

McPAT

McPAT, an integrated power, area, and timing modeling framework that supports comprehensive design space exploration for multicore and manycore processor configurations ranging from 90nm to 22nm. McPAT allows the user to specify low-level configuration details. It also provides default values when the user chooses to only specify high-level architectural parameters. The key components of McPAT are the hierarchical power, area, and timing models . The optimizer for determining circuit-level implementations, and the internal chip

representation that drives the analysis of power, area, and timing. Most of the parameters in the internal chip representation, such as cache capacity and core issue width, are directly set by the input parameters. McPAT's hierarchical structure allows it to model structures at a very low level, and yet still allows an architect to focus on the high-level configuration. The optimizer determines unspecified parameters in the internal chip representation, focusing on major regular structures.

HOW AND WHAT TO MEASURE IN INTEL I7

- We first measure power vs #cores in intel i7.
- We will then measure performance by varying associativity of the cache in all level in intel i7
- We will then measure performance based on cache replacement policy in all three level in intel i7

EXPERIMENTAL SETUP

- We are using Sniper for simulation (native installation)
- gainestown.cfg configuration file for as intel i7 (Nehalem Microarchitecture)
- McPAT.py for power visulation
- fft.c (fast furior transform) as testing application

DEMONSTRATION OF THE EXPERIMENT

Let first discuss the file structure of the Sniper Simulator

```
/sniper
  |     /config
  |       |     /gainestrom.cfg
  |     /tools
  |       |     /mcpat.py
  |     /test
  |       |     /fft
  |       |       |     /Makefile
  |       |       |     /fft.c
  |       |     /spintool
  |       |     /fft_dvfs
```

So, above is the hierarchical file structure of sniper simulator.

So, to run an application and analyse the power consumption, we have to follow these steps

1. We have to go inside the application folder. For, demonstration purpose we are using “fft”
2. Inside an application folder we will find a “application.c” and a Makefile. Note, if we want to run own application we can do so but program written by an user is not upto benchmark performance.
3. MakeFile
   - **TARGET= fft**
   - **include ../shared/Makefile.shared**
   - **$(TARGET): $(TARGET).o**
   - **$(CC) $(TARGET).o -lm $(SNIPER_LDFLAGS) -o $(TARGET)**
   - **run_$(TARGET):**
   - **../../run-sniper -v -n 1 -c “gainestown.cfg” -s “script.py” --roi -- ./fft -p 1**
   - **CLEAN_EXTRA=viz**

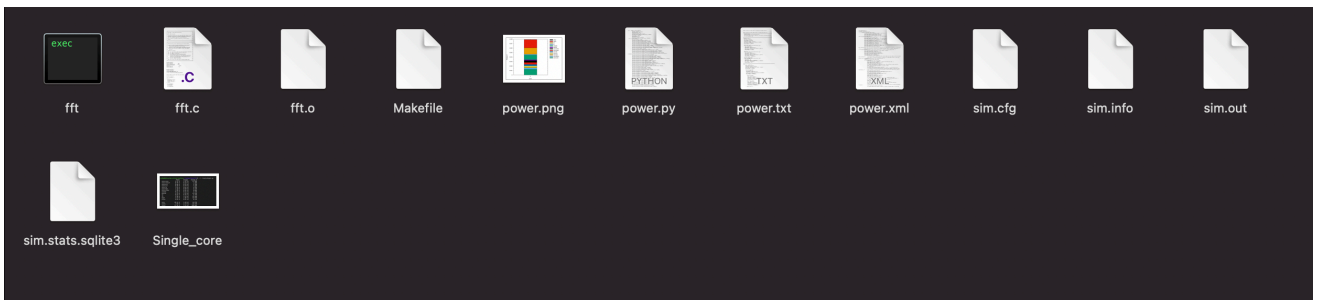
   Here,

   -c configuration file flag

   -s script file flag

   -n number of cpu cores

4. $ make run
5. After the sniper simulation complete the simulation, it will generates some output files, like power.xml, power.png, power.py etc



6. We can use this power.xml file in McPAT.py and sim.cfg to generate the power statistics. Ex:

```
[cs20m002@csalab-virtual-machine:~/sniper/test/fft$ ../../tools/mcpat.py
                     Power      Energy    Energy %
  core-core         3.19 W     1.35 mJ     14.28%
  core-ifetch       0.95 W     0.40 mJ      4.25%
  core-alu          0.60 W     0.25 mJ      2.70%
  core-int          0.95 W     0.40 mJ      4.26%
  core-fp           1.55 W     0.66 mJ      6.95%
  core-mem          1.18 W     0.50 mJ      5.27%
  core-other        1.98 W     0.84 mJ      8.85%
  icache            0.61 W     0.26 mJ      2.71%
  dcache            2.73 W     1.15 mJ     12.21%
  l2                0.84 W     0.35 mJ      3.74%
  l3                3.38 W     1.43 mJ     15.10%
  dram              4.37 W     1.85 mJ     19.54%
  other             0.03 W     0.01 mJ      0.12%

  core             10.41 W     4.40 mJ     46.57%
  cache             7.55 W     3.19 mJ     33.77%
  total            22.36 W     9.45 mJ    100.00%
```

7. By these data, we are going to give our analysis and learning.

EXPERIMENT AND DATA COLLECTION

- **We first measure power vs #cores in intel i7**

For this question, we have collected 12 samples for 3 application program. For each application we have collected four sample. For each application we are varying number of cores from 1,2,4 8 .

- **We measure performance by varying associativity of the cache in all level in intel i7**

For this question, we have collected 18 samples for 3 application program. For each application we have collected 6 samples. For each application we are varying associativity from 1,2,4,8,32,64.

- **We measure performance based on cache replacement policy in all three level in intel i7**

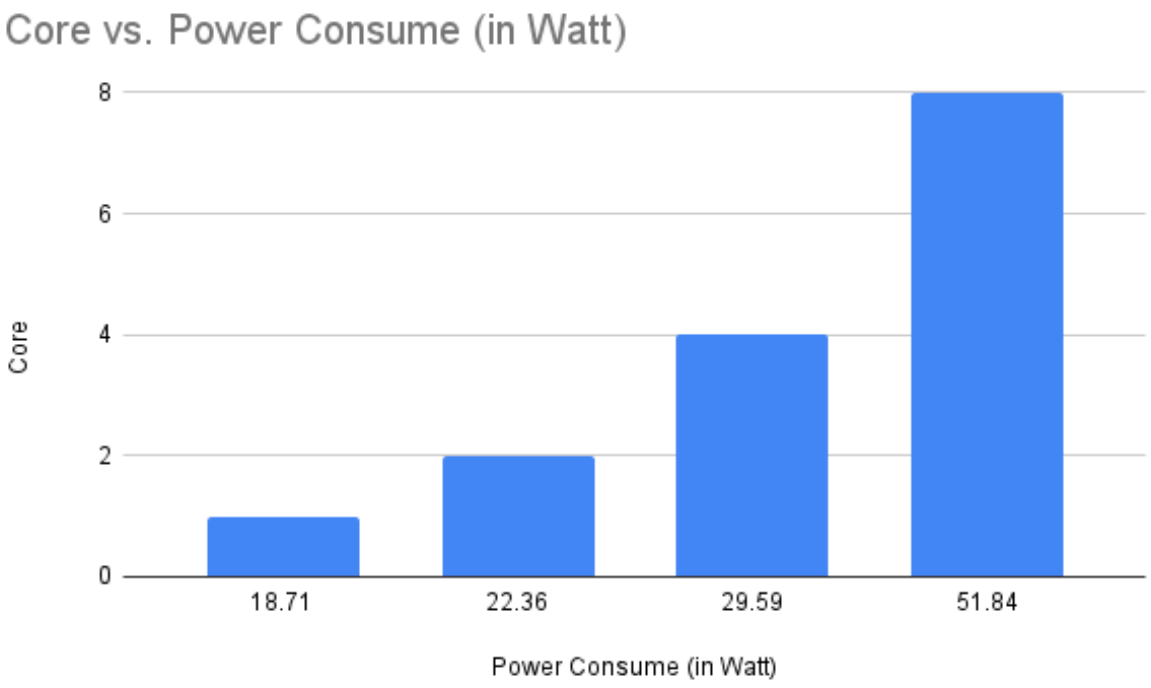
For this question, we have collected 9 samples for 3 application program. For each application we have collected 3 samples. For each application we are varying replacement policy as “lru”, “mru” and “random”.

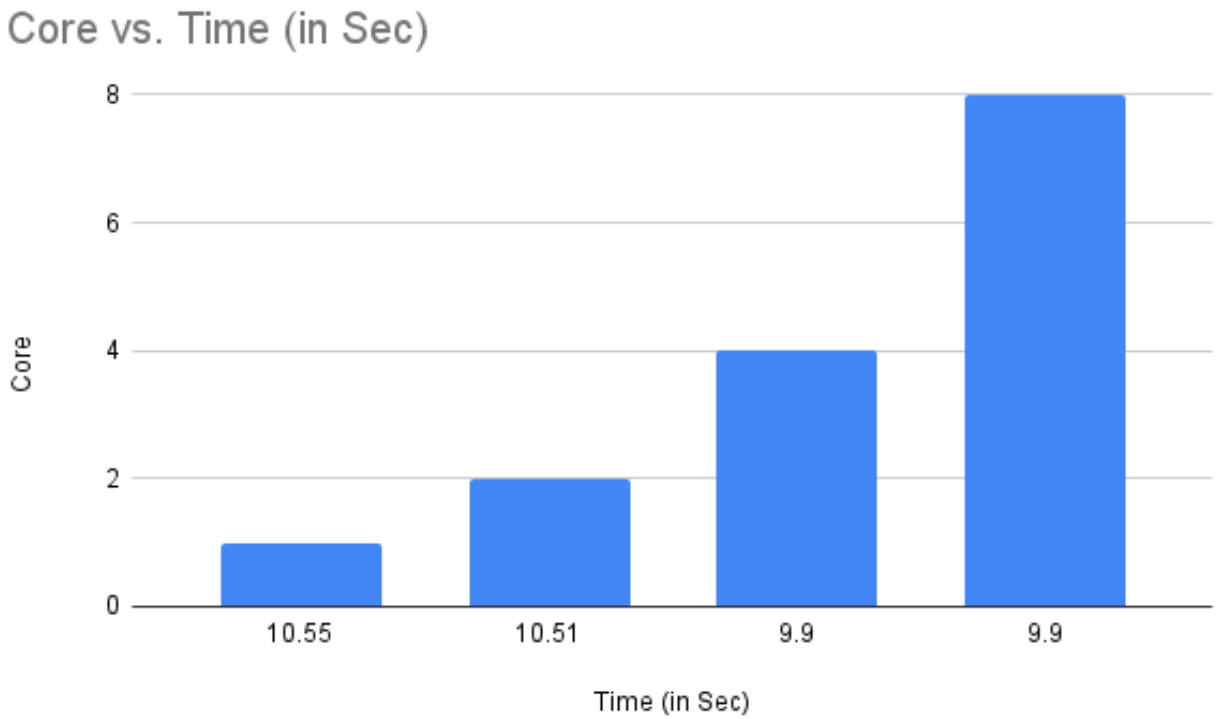
Note : Data and all codes and commands related to this experiment can be found on GitHub : “ https://github.com/cs20m002/CSA_Term_Project2020 ” in file intel_i7.xls**x**

RESULT AND OBSERVATION

- **We first measure power vs #cores in intel i7**

We have found that will increasing the number of cores we are getting better efficiency but at the same time power consumption of the cpu increased. For gaining only 1.1x speed in elapsed time, power consumption increased by 2x. So, performance and efficiency trade-off we have to think before design consideration





- **We measure performance by varying associativity of the cache in all level in intel i7**

While varying the associativity from 1,2,4,8,16,32 and 64, what we observe is we are getting best result at when associativity is in mid i.e. neither direct mapped nor fully associative. There is some catch here that we need to consider, as we are running program that might not meant for showing good result in direct and fully.

- **We measure performance based on cache replacement policy in all three level in intel i7**

We find out that for two of the applications “fft” and “fft_dvfs” Random replacement policy works better than MRU and LRU. And for spin loop application LRU gives better result.

FUTURE WORK

As we have seen this mcpat power visualisation tool is not giving us per core power consumption, so one can work on that. This is because today we have high frequency core for intensive work that take more power and low frequency core for regular work that take low power. So exact power per core will help in understanding of this core specific performance.

REFERENCES

1. https://github.com/duttresearchgroup/sniper-mem/blob/master/config/
2. https://ark.intel.com/content/www/us/en/ark/products/199314/intel-core-i7-10700t-processor-16m-cache-up-to-4-50-ghz.html
3. https://github.com/HewlettPackard/mcpat/blob/master/Documents/micro09.pdf
4. https://heirman.net/papers/carlson2014nodeperformance.pdf
5. https://www.it.uu.se/katalog/vassp447/caos_final.pdf
6. https://www.agner.org/optimize/microarchitecture.pdf